Cairo Pimenta Cheble Caplan

Desenvolvimento da unidade lógica programável para o upgrade da eletrônica de controle dos experimentos do LHC

Brasil

10 de Janeiro de 2017

Cairo Pimenta Cheble Caplan

Desenvolvimento da unidade lógica programável para o upgrade da eletrônica de controle dos experimentos do LHC

Dissertação de Pós-Graduação Stricto Sensu apresentada à Coordenação de Formação Científica do Centro Brasileiro de Pesquisas Físicas - CBPF do Ministério da Ciência, Tecnologia e Inovação para obtenção do título de Mestre no programa Mestrado Profissional em Física com ênfase em Instrumentação Científica.

Ministério da Ciência, Tecnologia e Inovação

Centro Brasileiro de Pesquisas Físicas – CBPF

Programa de Pós-Graduação

Orientador: André Massafferri Rodrigues

Coorientador: Federico Alessio

Brasil

10 de Janeiro de 2017

Cairo Pimenta Cheble Caplan

Desenvolvimento da unidade lógica programável para o upgrade da eletrônica de controle dos experimentos do LHC/ Cairo Pimenta Cheble Caplan. – Brasil, 10 de Janeiro de 2017-

349 p. : il. (algumas color.) ; 30 cm.

Orientador: André Massafferri Rodrigues

Tese (Mestrado) – Ministério da Ciência, Tecnologia e Inovação
Centro Brasileiro de Pesquisas Físicas – CBPF
Programa de Pós-Graduação, 10 de Janeiro de 2017.

1. LHCb. 2. Sistemas de controle de detectores. 3. Processamento de sinais digitais (DSP). 4. Controle e monitoramento de sistemas online. I. André Massafferri Rodrigues. II. Centro Brasileiro de Pesquisas Físicas. III. Faculdade de xxx. IV. Título

Cairo Pimenta Cheble Caplan

Desenvolvimento da unidade lógica programável para o upgrade da eletrônica de controle dos experimentos do LHC

Dissertação de Pós-Graduação Stricto Sensu apresentada à Coordenação de Formação Científica do Centro Brasileiro de Pesquisas Físicas - CBPF do Ministério da Ciência, Tecnologia e Inovação para obtenção do título de Mestre no programa Mestrado Profissional em Física com ênfase em Instrumentação Científica.

Brasil
10 de Janeiro de 2017

Aos meus pais Elizabeth e Pedro
e à minha irmã Shalimar.

Agradecimentos

Ao meu orientador, André Massafferri, pelas oportunidades oferecidas, desafios propostos, suporte, conselhos e pela paciência como orientador e amigo.

Ao Federico Alessio, meu supervisor de projeto no CERN, pela amizade, dicas, críticas e integração ao grupo do LHCb e do CERN.

Ao Maurício Féo, pela amizade desde o período da faculdade de engenharia no CEFET/RJ, por me apresentar o CBPF e pela parceria no grupo de pesquisa desde então.

Agradecimentos especiais são direcionados ao Leonardo Guedes, Leonardo Lessa, Laís Lavra, Ulisses Carneiro, pela amizade e diversas conversas desde o início do meu período no CBPF.

Ao Antonio Pellegrino, de Nikhef; João Barbosa e Ken Wyllie, do CERN, não apenas pela amizade como pela contribuição e suporte, sem as quais esse trabalho não seria possível.

Aos vários amigos e colegas com quem tive a oportunidade de conhecer pela colaboração CBPF e LHCb.

Ao meu amigo Gustavo Nascimento pelo apoio final na elaboração do trabalho.

Ao CNPQ, CBPF, LHCb e CERN que, como instituições, ofereceram suporte na formação acadêmica e profissional, financeiro e infraestrutura para a realização deste trabalho.

Resumo

O acelerador LHC no CERN definiu um novo programa de física de partículas que propõe medidas de alta precisão do Modelo Padrão e busca por Nova Física. Para realizar esse objetivo o LHC e seus experimentos necessitam de atualizações e melhorias em seus instrumentos, contemplando atividades de pesquisa e desenvolvimento em novos sistemas de transmissão de sinais de dados, temporização e controle.

Este trabalho consiste no desenvolvimento módulo eletrônico digital reprogramável para o controle e monitoramento da eletrônica de Front-End, onde são aproveitadas as conexões ópticas bidirecionais e os recursos dos chips GBT e GBT-SCA. O projeto está integrado à infraestrutura eletrônica do upgrade do LHCb e foi proposto como um pacote padrão para uso em outros experimentos do LHC.

Palavras-chave: LHCb. Sistemas de controle de detectores. Processamento de sinais digitais (DSP). Controle e monitoramento de sistemas online.

Abstract

The LHC accelerator at CERN has proposed a new particle physics program towards high-precision standard model measurements and search for new physics. To accomplish this goal the LHC and its experiments require upgrades and improvements on their instruments, covering research and development activities for data transport, timing and control signals between the experiment itself.

The work consists on the design and building of a reprogrammable digital module for control and monitoring of Front-End electronics, profiting from bidirectional optical connections and resources of the GBT and GBT-SCA chipsets. The project was born and integrated into the electronic infrastructure of the LHCb upgrade and was proposed as a standard package for use in other LHC experiments.

Keywords: LHCb. Detector control systems. Digital signal processing (DSP). Control and monitoring of online systems.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

ALICE	A Large Ion Collider Experiment
ATLAS	A Toroidal LHC ApparatuS
BE	Back-End
CBPF	Centro Brasileiro de Pesquisas Físicas
CCPC	Credit-card sized PC
CDC	Clock Domain Crossing
CERN	Centre Européen pour la Recherche Nucléaire
DAQ	Data Acquisition System
DDR	Double Data Rate
DSP	Digital Signal Processing
EC	Experiment Control
ECS	Experiment Control System
FE	Front-End
FCS	Frame Check Sequence
FEC	Forward Error Correction
FIFO	First-In First-Out
FPGA	Field-programmable gate array
GBT	Gigabit Bidirectional Trigger and Data link
GBTx	GigaBit Transceiver
GBT-SCA	The Slow Control Adapter ASIC for the GBT System
GPIO	General Purpose Input/Output
HDL	Hardware Description Language
HL-LHC	High Luminosity LHC

HLT	High Level Trigger
I2C	Inter-Integrated Circuit
IP	Intellectual Property ou Internet Protocol
IT	Inner Tracker
JTAG	Joint Test Action Group
LAFEX	Laboratório de Física Experimental de Altas Energias
LEP	Grande Colisor de Elétrons e Pósitrons
LHC	Large Hadron Collider
LHCb	Large Hadron Collider beauty
LHCf	Large Hadron Collider foward
Linac4	Linear accelerator 4
LLT	Low Level Trigger
LS1	Long Shutdown I
LS2	Long Shutdown II
LS3	Long Shutdown III
MP	Modelo Padrão
MEP	Multiple Event Packets
MGT	Multi-Gigabit Transceiver
MoEDAL	The Monopole & Exotics Detector at the LHC
OT	Outer Tracker
PCB	Printed Circuit Board
PCIe	Peripheral Component Interconnect Express
RICH	Ring Imaging Cherenkov Detector
SAT	Stand Alone Test Board
SC	Slow Control
SCADA	Supervisory Control And Data Acquisition

SciFI	Scintillating Fibre Tracker
SEU	Single Event Upset
SFP	Small form-factor pluggable transceiver
SM	Standard Model
SPI	Serial Peripheral Interface
TAP	Test Access Point
TFC	Trigger and Fast Control
TMR	Triple Modular Redundancy
TOTEM	TOTal Elastic and diffractive cross section Measurement
TTC	Trigger, Timing and Control
uTCA	MicroTCA
VELO	VErtex LOcator
VLDB	Versatile-Link Demo Board

Sumário

Introdução

Muitas das questões atuais da física, como a origem e a composição do universo, a assimetria entre matéria e antimatéria e a busca de partículas fora do Modelo Padrão da Física de Partículas, são pesquisadas em experimentos ligados ao LHC, o maior acelerador de partículas do mundo, do CERN, a Organização Europeia para a Pesquisa Nuclear, na fronteira entre a Suíça e a França. Dentre os grupos internacionais de pesquisadores, engenheiros, técnicos e associados, o Centro Brasileiro de Pesquisas Físicas (CBPF) tem participação ativa, através do Laboratório de Física Experimental de Altas Energias (LAFEX), nos experimentos LHCb e CMS.

Este texto descreve um trabalho multidisciplinar, na área de instrumentação científica aplicada a física de altas energias. Tal área envolve conhecimentos de diversos campos do conhecimento com destaque para engenharia e computação. Sendo por isso desafiadora sintetizá-los de forma fluida num texto.

A parte I do trabalho, chamada *Contexto da Pesquisa*, trata do contexto do projeto, introduzindo o acelerador de partículas LHC e seus experimentos dando ênfase ao upgrade do experimento LHCb, sua eletrônica, sistema de controle e ao subdetector de fibras cintilantes, objetivo inicial deste trabalho.

A parte II, chamada *Desenvolvimento*, trata do projeto em si. Nela são apresentadas as diferentes etapas do mesmo, escolhas, utilização de conceitos teóricos e seu estado atual.

A parte III, chamada *Resultados*, é intrinsecamente ligada a parte do Desenvolvimento. São apresentadas nesta seção validações utilizando metodologias de verificação de sistemas digitais divididas entre simulação e testes em hardware.

Por último a *Conclusão*, parte IV, discute de forma sintética o desenvolvimento e os resultados do trabalho, são também apresentadas duas seções dedicadas às publicações e procedimentos futuros do mesmo.

Este trabalho também apresenta duas apêndices, a primeira, chamada *Referencial Teórico* A, apresenta conceitos teóricos na área de eletrônica, instrumentação e computação necessários ao desenvolvimento do projeto e compreensão deste trabalho. A segunda, chamada *Código do projeto SOL40-SCA* B, representa de forma completa a última versão estável do código de desenvolvimento do projeto bem como arquivos auxiliares.

Parte I

Contexto da Pesquisa

1 O LHC e seu upgrade

A teoria do Modelo Padrão (MP) de física de partículas, também chamado de *Standard Model* (SM), foi comprovada com sucesso num nível de precisão excepcional nas últimas décadas por diversos experimentos. No entanto o MP apresenta algumas deficiências críticas como a unificação de todas as forças.

Os experimentos que ocorrem no CERN foram concebidos para investigar questões não respondidas no campo da física de altas energias. Um novo programa de física de partículas se iniciou com o LHC propiciando, como principal resultado, a descoberta do bóson de Higgs. Esta conquista é apenas o início de um intenso programa de atividades do LHC para melhor entendimento do MP e da Nova Física.

1.1 O colisor LHC

O Grande Colisor de Hádrons, denominado *Large Hadron Collider* (LHC) e representados nas figuras 1, 2 e 3 , da Organização Europeia para a Pesquisa Nuclear (CERN) é atualmente o maior e acelerador de partículas do mundo. Consistindo num anel de 27 km de comprimento construído nos antigos túneis do antigo Grande Colisor de Elétrons e Pósitrons, denominado *The Large Electron-Positron Collider* (LEP). O LHC é composto de magnetos supercondutores e estruturas de aceleração para aumentar a energia das partículas ao longo do caminho, sua operação teve início em 10 de setembro de 2008 e permanece como a última adição ao complexo de aceleradores do CERN.

Dentro do acelerador, dois feixes de partículas de altas energias são acelerados a velocidade próxima a da luz antes de colidirem. Os feixes trafegam em direções opostas em *beam pipes*, como dois tubos mantidos num vácuo super intenso, com pressões de vácuo na ordem de 10^{-10} a 10^{-11} mbar (1). O LHC foi projetado para colidir prótons, com energia de centro de massa de 14 TeV com uma luminosidade instantânea de $10^{34} cm^{-2} s^{-1}$, e íons pesados a uma energia de 2,9 TeV por núcleons e uma luminosidade de $10^{27} cm^{-2} s^{-1}$. Cada feixe de partícula é guiado por um forte campo magnético mantido por bobinas supercondutoras a temperatura de -271,3 °C. Isto requer que muito do acelerador esteja conectado a um sistema de distribuição de hélio líquido, resfriando os magnetos bem como outros recursos.

Há um total de sete experimentos que utilizam o LHC. Cada experimento tem propósitos diferentes, e é caracterizado pelos seus detectores.

Os maiores experimentos, em ordem do grandeza do maior para o menor, são o ATLAS e o CMS, que usam detectores de propósito geral para investigar a maior gamas

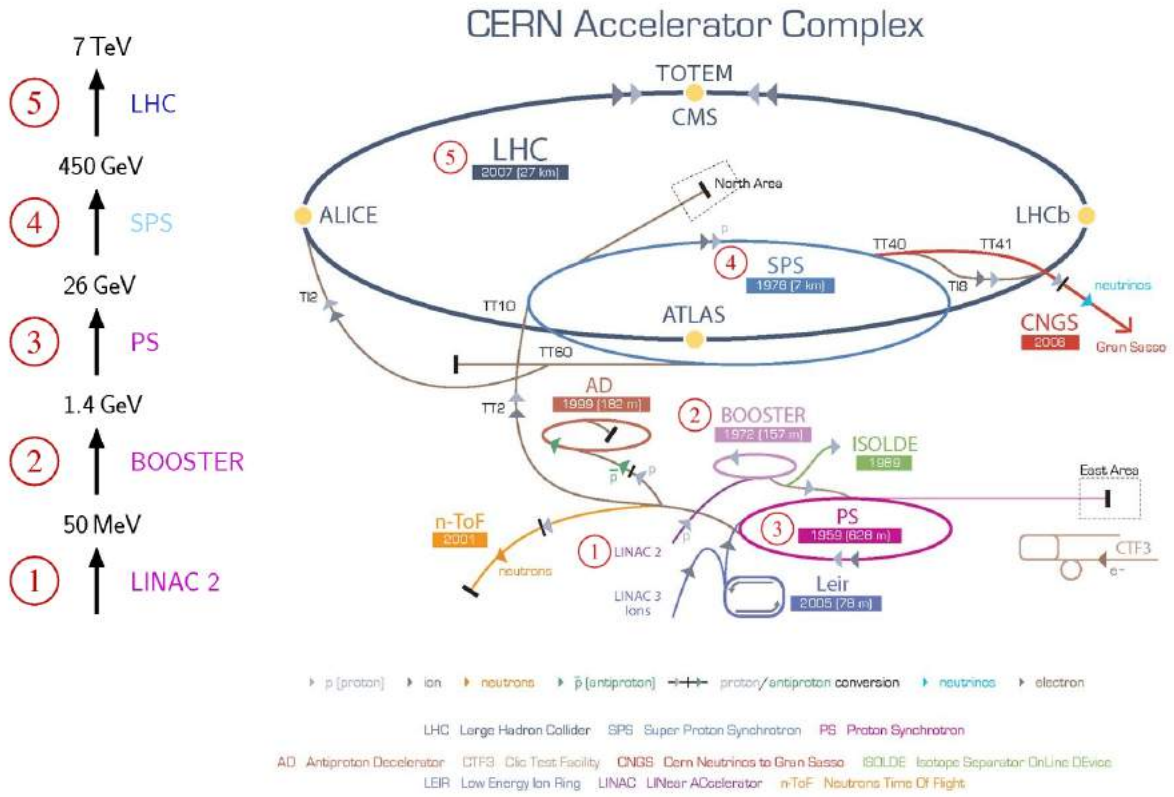


Figura 1 – Complexo de aceleradores e detectores do CERN.

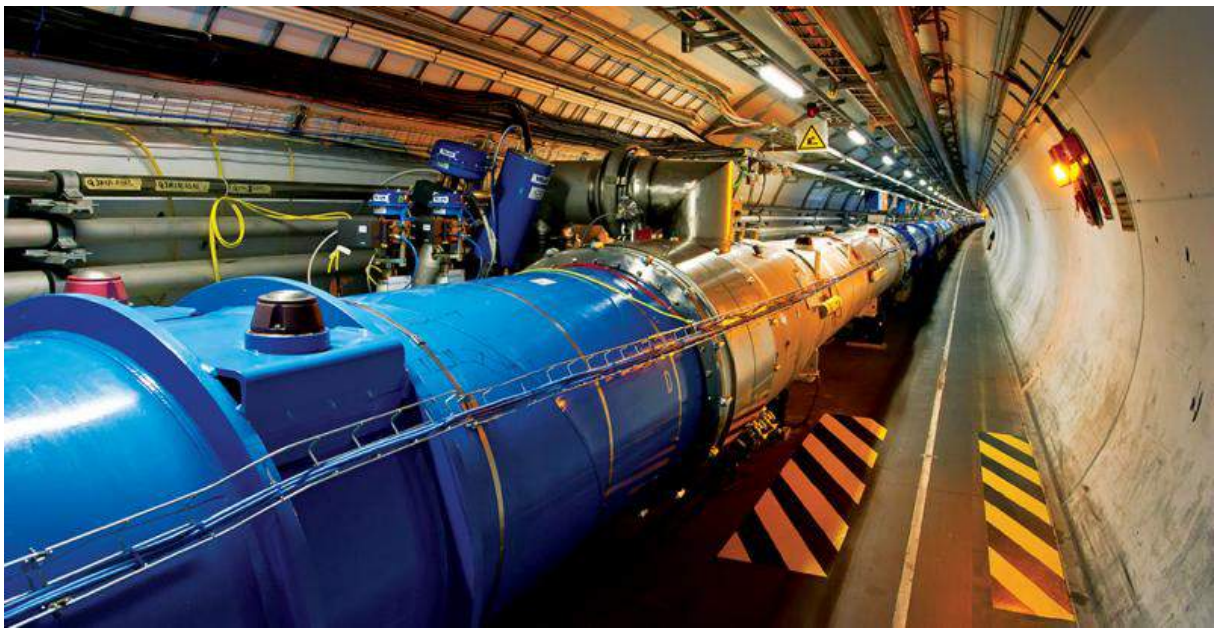


Figura 2 – Visão do túnel do LHC.

de processos físicos possível. O fato do CERN possuir dois detectores construídos de forma independente que podem ser usados com os mesmos objetivos é uma garantia para confirmação de novas descobertas físicas. Mantendo a ordem os seguintes são o ALICE, especializado na colisão de íons pesados, e o LHCb, dedicado ao estudo da violação da

Figura 3 – Visão aérea do LHC.

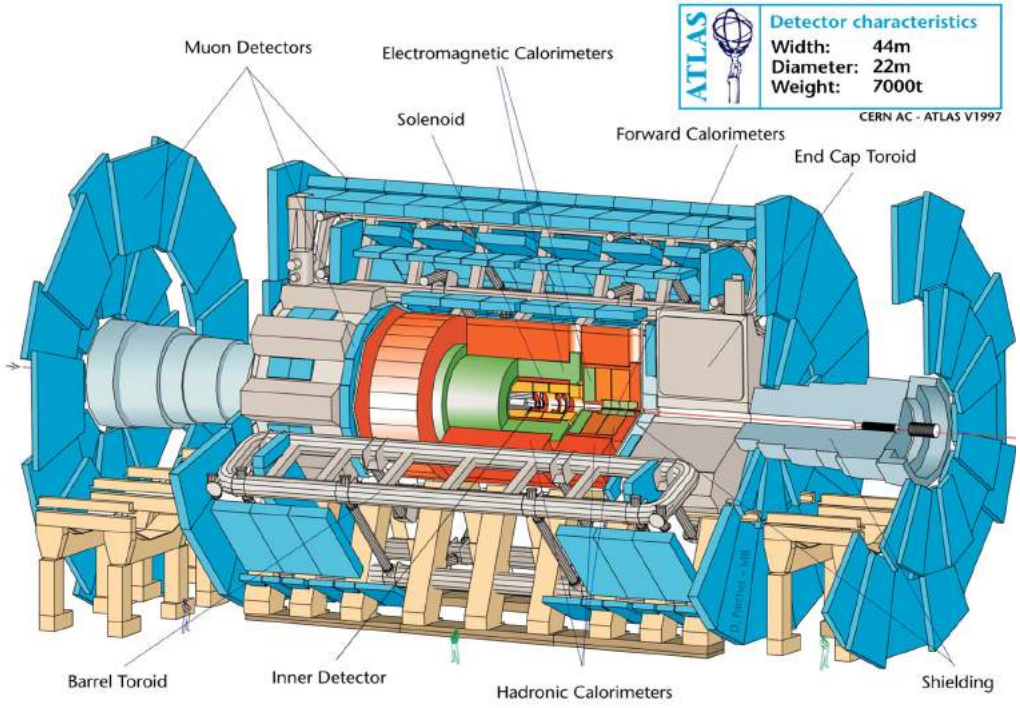


Figura 4 – Modelo tridimensional do ATLAS.

assimetria Carga-Paridade (CP), sendo experimentos especializados na busca ou exploração de fenômenos específicos, compondo a lista dos quatro detectores que residem em grandes cavernas subterrâneas ao longo do anel do LHC.

Os menores experimentos associados ao LHC são o MoEDAL, o TOTEM e o LHCf,

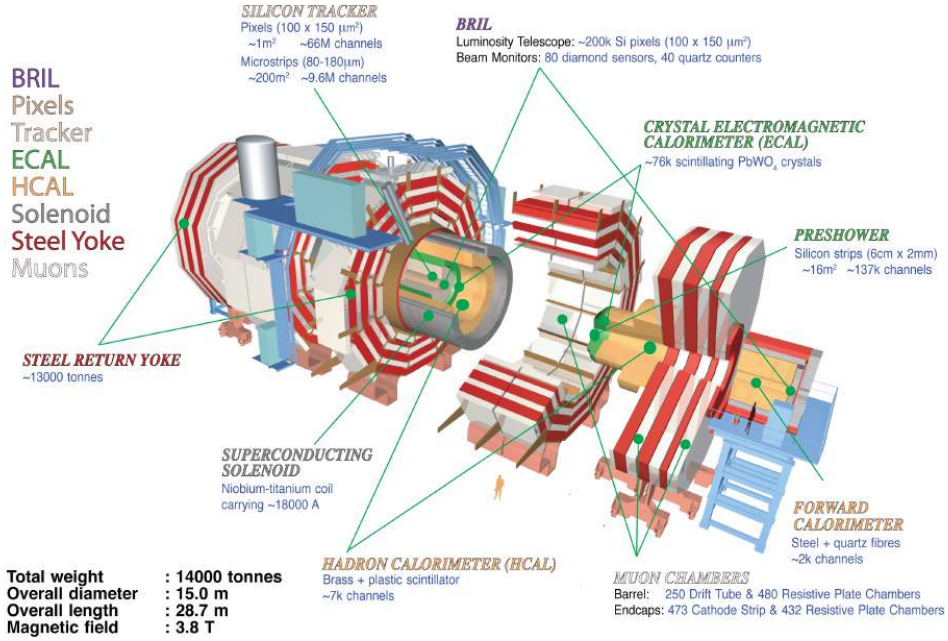


Figura 5 – Esquema em três dimensionais detalhado do CMS.

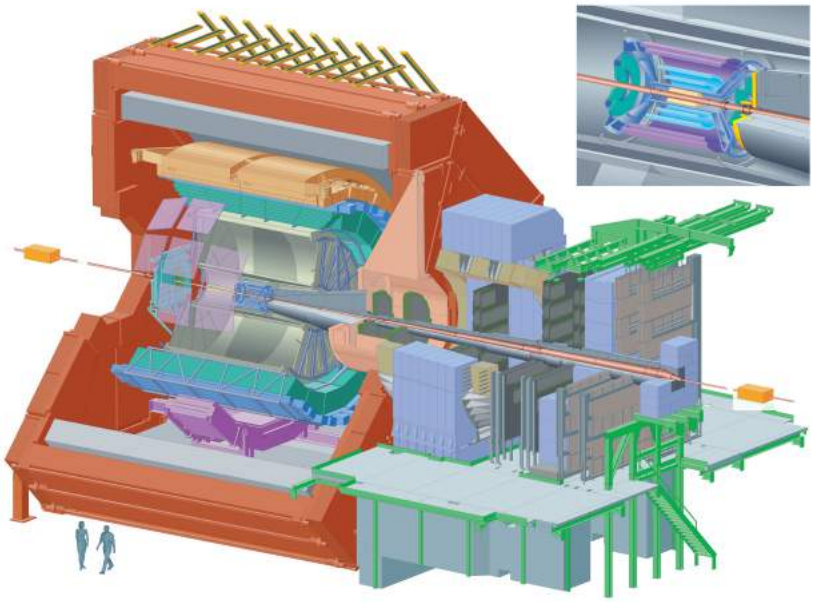


Figura 6 – Representação do experimento ALICE.

os dois últimos se focam em partículas resultadas de colisões que se mantém a um ângulo muito pequeno em relação ao feixe original. O TOTEM tem seus detectores posicionados em ambos os lados do ponto de interação do CMS, enquanto o LHCf é feito por dois detectores que está disposto a longo do caminho do feixe de partículas do LHC em dois locais a 140 metros de distância do ponto de colisão do ATLAS. O MoEDAL tem seus detectores localizados perto do LHCb para buscar partículas hipotéticas, previstas por alguns modelos teóricos, chamadas de monopolo magnético.

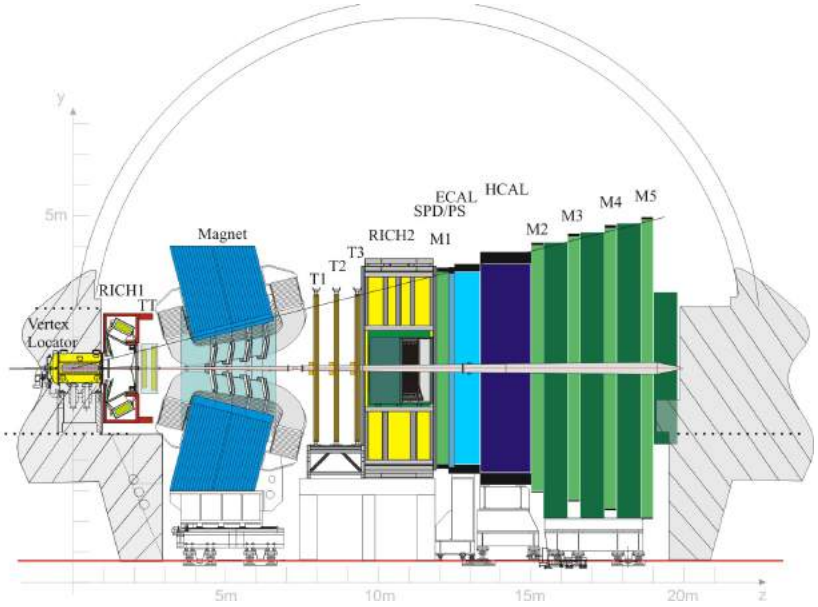


Figura 7 – Seção transversal do LHCb, corte no plano y-z.

1.2 LHCb

O experimento LHCb (2) é um experimento de alta precisão ligado ao acelerador LHC dedicado a busca por Nova Física ao medir precisamente seus efeitos em violações CP e decaimentos raros. Utilizando um método indireto, o LHCb é capaz de explorar efeitos que são fortemente suprimidos no Modelo Padrão (MP) de partículas elementares.

Ao operar no modo de colisões de próton-próton, o LHC é em grade parte uma fábrica de partículas de sabores pesados produzindo mais de 100 mil pares bb a cada segundo na luminosidade nominal do LHCb de $2.10^{33}\text{cm}^{-2}\text{s}^{-1}$. Considerando que pares bb são predominantemente produzidos nas direções frontais ou traseiras, o LHCb foi projetado como um espectrômetro e braço único, com os elementos do detector instalados ao longo da linha do feixe principal do LHC, cobrindo uma faixa de pseudorrapidez de $2 < \eta < 5$ o que complementa bem as faixas dos outros detectores do LHC.

O layout do detector mostrado na 7. Os principais subdetectores são respectivamente o detector de vértice (VELO), dois detectores Cherenkov para identificação de partículas (RICH1 e RICH2), um magneto, detectores de trajetórias (TT, T1- T3) e um sistema de calorímetros (SDP, PS, ECAL e HCAL), seguido de um sistema de múons (M1-M5).

1.2.1 Magneto

O magneto do LHCb tem por objetivo principal a criação de um forte campo magnético, suficientemente intenso para curvar as partículas criadas na colisão. Com esta informação e aliado ao sistema de rastreamento de *tracking* de partículas o experimento é capaz de reconstruir o momento relacionado às partículas além de sua carga. Por sua vez

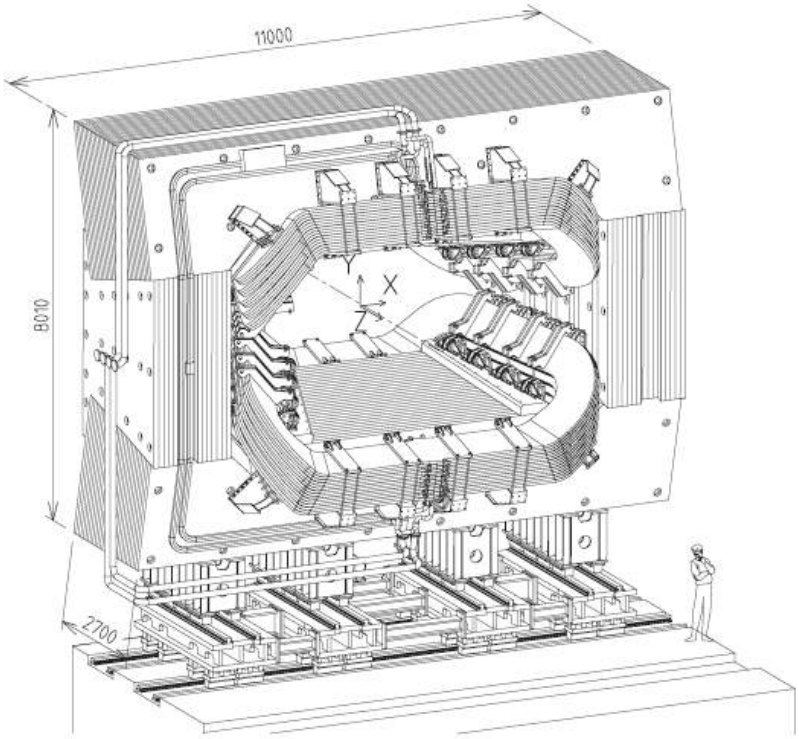


Figura 8 – Vista em perspectiva do magneto do LHCb.

o momento das partículas é também fundamental na identificação delas ao passarem pelos subdetectores RICH. Outra função importante do magneto é manter o campo magnético uniforme, garantindo uma boa medida do momento e favorecendo a identificação de partículas.

Estruturalmente o magneto é um dipolo magnético capaz de produzir um campo magnético integrado de 4 T m, cujo sentido pode ser invertido. Esta inversão é usada para o estudo e a redução de erros sistemáticos em medidas de assimetria. O magneto está representado na figura

1.2.2 VELO

O primeiro subdetector do LHCb é denominado *Vertex Locator* (VELO) ou subdetector de vértices, o subdetector mais próximo da região de colisão próton-próton e sujeito a maiores intensidades de radiação ionizante. Como parte do sistema de traços, o objetivo do VELO é medir as coordenadas de cada traço produzidos perto da região de colisão para posterior reconstrução, via software, dos vértices de produção (primário) e decaimento (secundário) dos mésons B. Além disso, medidas do tempo de vida dos mésons B e do parâmetro de impacto das partículas utilizadas no processo de rotulamento do seu sabor também são feitas a partir das posições dos traços na região de colisão. O VELO também fornece importantes informações para o processo de filtragem de eventos, enriquecendo as amostras B selecionadas.

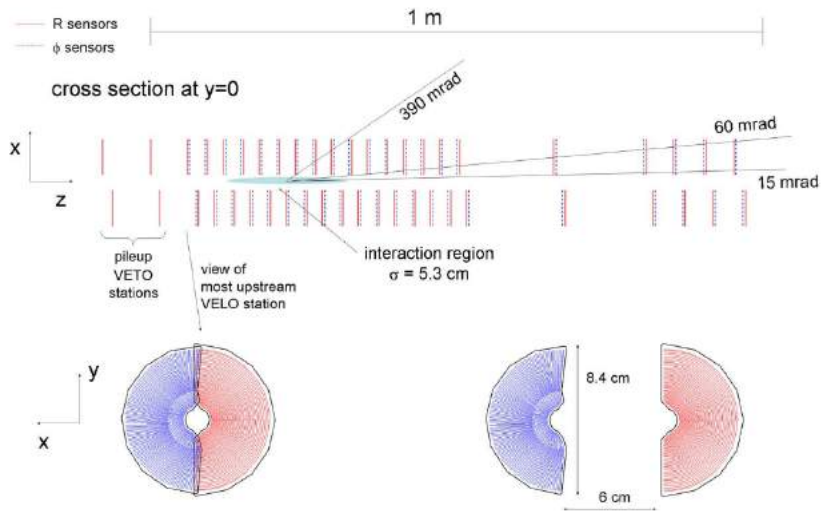


Figura 9 – Disposição do subdetector VErtex LOcator.

A estrutura do VELO é composta de 34 semidiscos instalados em volta da região de colisão, de forma a garantir que as partículas atravessem no mínimo 3 deles, mostrada na figura 9.

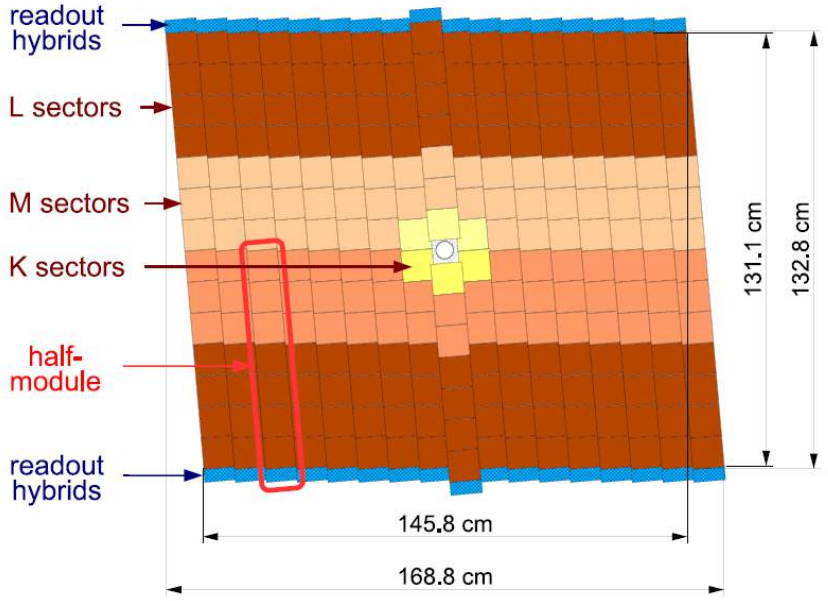
Os semidiscos de silício, perpendiculares a direção do feixe, fornecem uma cobertura radial (r,ϕ) de 1 cm a 6 cm, com um espaçamento de 4 cm entre eles para a região inferior do cano do feixe de partículas e 2 cm para a região superior. Como resultado desta geometria, pelo menos 3 semidiscos serão atravessados por partículas com ângulo polar superior a 15 mrad. Para partículas que percorrem trajetórias com ângulo inferior a 15 mrad são utilizados outros 5 semidiscos, posicionados a distâncias de 8 cm, 14 cm e 10 cm, de forma a garantir que elas atravessem 3 semidiscos no mínimo. Há um total de 48 semidiscos instalados ao redor do cano do feixe de prótons.

O VELO permite a reconstrução de vértices primários com resolução da ordem de 40 µm na direção z e 10 µm na direção ϕ. Para vértices secundários a resolução é menor devido a quantidade reduzida de semidiscos atravessados, variando entre 150 µm a 300 µm a depender do número de traços.

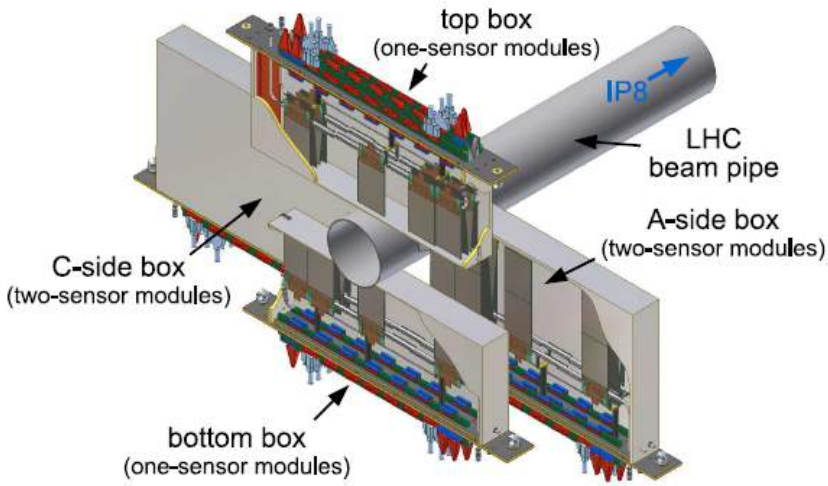
1.2.3 Silicon Tracker

O Silicon Tracker é na verdade composto de dois subdetectores, o Tracker Turicensis (TT) e o Inner Tracker (IT), ambos baseados em sensores de microtiras (*microstrips*) com aproximadamente 200 µm.

O primeiro subdetector, o TT, representado na figura 10a, se localiza entre o VELO e o magneto e é composto por suas estações com largura de 150 cm e altura de 130 cm. Cada estação é composta por quatro planos de sensores de silício com espessura de 500 µm. Um sensor é formado por 512 microtiras de silício espeçadas de 183 µm. Isto possibilita



(a) Representação do terceiro plano do Tracker Turicensis.



(b) Representaão do Inner Tracker.

uma resolução espacial de aproximadamente 50 µm.

As microtiras seguem orientações diferentes em cada plano. No primeiro plano a orientação é vertical; no segundo plano as microtiras são rotacionadas de -5° na direção do feixe; terceiro plano de 5° e no último as microtiras são dispostas verticalmente como no primeiro, devido a esta geometria esses planos são chamados de $x - u - v - x$.

O subdetector IT, representado na figura 10b, é localizado após o magneto em relação ao ponto de interação mas na região próxima ao feixe, onde há o maior fluxo de partículas. O sensores de silício seguem o mesmo layout de planos do TT, possuindo porém 384 tiras espaçadas de 198 µm.

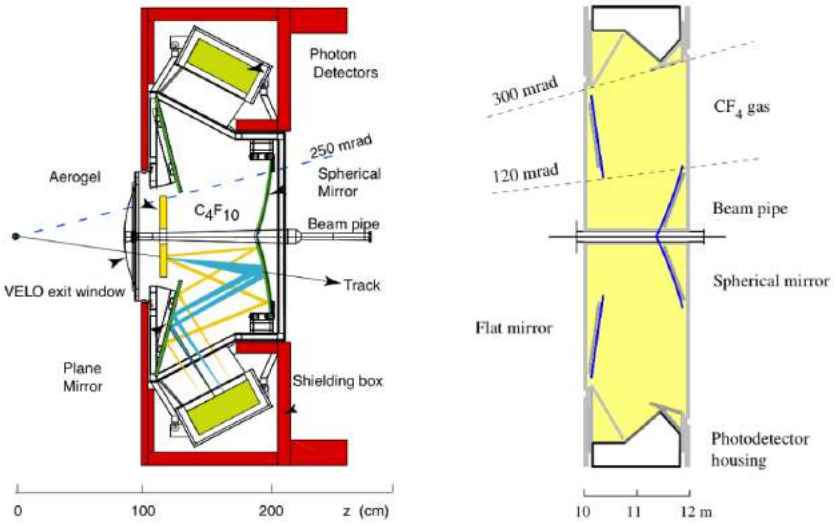


Figura 11 – O RICH1 é representado na esquerda e o RICH2 na direita.

1.2.4 Outer Tracker

O Outer Tracker (OT) é um detector de *straw tube* envolvendo o IT e nas estações de *tracking* T1-T3. Este detector determina a posição de partículas carregadas ao medir o tempo de *drift* nas *straw tubes*. O tempo máximo de *drift* é mantido abaixo de 50 ns e a resolução das coordenadas do *drift* é de cerca de 200 µm.

O módulos do OT são dispostos em três estações. Cada estação consiste de quatro camadas na mesma orientação de IT e TT.

1.2.5 RICH

Dois detectores *Ring Imaging Cherenkov* (RICH) são usados no LHCb. Os sistema RICH tem a tarefa de identificar partículas carregadas sobre a faixa de momento de 1 a 150 GeV/c. Isto é obtido ao utilizar três radiadores diferentes, o aerogel e C_4F_{10} no RICH1 (1-60 GeV/c) e CF_4 no RICH2 (15 a 100 GeV/c). O RICH1 é colocado entre o TT e o VELO, enquanto o RICH2 é situado entre as estações de tracking T3 e a primeira estação de múons.

Os dois subdetectores RICH usam detectores híbridos de fótons (HPDs) para medir a posição de fótons emitidos Cherenkov emitidos. A figura 11 mostra o design dos dois subdetectores.

1.2.6 Calorímetros

Os calorímetros são usados para identificação de partículas, medição de energia e posicionamento para elétrons, fótons e hádrons. Eles realizam um papel fundamental no primeiro nível do *trigger*, onde eles selecionam elétrons, fótons e hádrons candidatos com

uma energia transversal acima de um certo nível. Eles são posicionados após o RICH2 e da primeira câmara de múons, mas antes da parte principal do sistema de múons. O sistema do calorímetro é localizado entre as estações de múons M1 e M2.

O sistema consiste de algumas camadas, o *Scintillating Pad Detector* (SPD), o *Pre-Shower* (PS), o calorímetro eletromagnético (ECAL) e calorímetro de hádrons (HCAL). O SPD determina se as partículas que atingem o sistema de calorímetros são carregadas ou neutras, enquanto o PS indica caráter eletromagnético da partícula. Eles são usados no nível de *trigger* em associação com o ECAL para indicar a presenção de elétrons, fótons e píons neutros. O ECAL foi projetado para contem o chuveiramento de elétrons e fótons, enquanto o HCAL absorve completamente a energia dos hádrons.

Os módulos dos calorímetros consistem de *scintillating pads* alternados por absorvedores de chumbo ou ferro. Partículas incidentes interagem fortemente com os absorvedores criando *showers* de partículas secundárias que irão produzir luz enquanto passam através das *scintillating pads*. A luz é absorvida por fibras *wavelength-shift* e ambas as extremidades das fibras WLS são usadas para transmitir luz para fotomultiplicadoras multianodo (MAPMTs), localizadas na periferia do detector. A quantidade de luz coletada pode ser relacionada a energia da partícula incidente . A densidade de detecções dependem da distância do eixo do feixe e varia pela aceptância do calorímetro em duas ordems de magnitude.

1.2.7 Sistema de Múons

O sistema e múons consiste de cinco estações, de M1 a M5, localizadas em volta do cano do feixe de partículas e após os calorímetros. Cada estação contém câmaras cobertas com uma combinação de três gases, $CO_2, Argônio e CF_4$, que são ionizados pelos múons. A tecnologia de detecção resistente a radiação usada no sistema de múons é a *Multi-Wire Proportional Chamber* (MWPC) com 2 mm de espaçamento de fios e um pequeno espaçamento de gas de 5 mm.

Na região interna do M1 estão os detectores instalados são os *Triple-GEM*, pois a ocupância é alta para as MWPCs. Absorvedores de ferro com 80 cm de espessura são intercalados entre as estações M2 a M5 e atrás de M5. Juntos do sistema de calorimetria, eles removem o *background* hadrônico e protegem o detector de múons das partículas que saíram do feixe do LHC. O momento mínimo que um múon precisa para alcançar o M5 é de aproximadamente 6 GeV/c. O sistema de múons é usado no sistema de *trigger* para selecionar eventos com múons de alto momento transverso (pT), no *High Level Trigger* (HLT) e na análise offline para a identificação de múons.

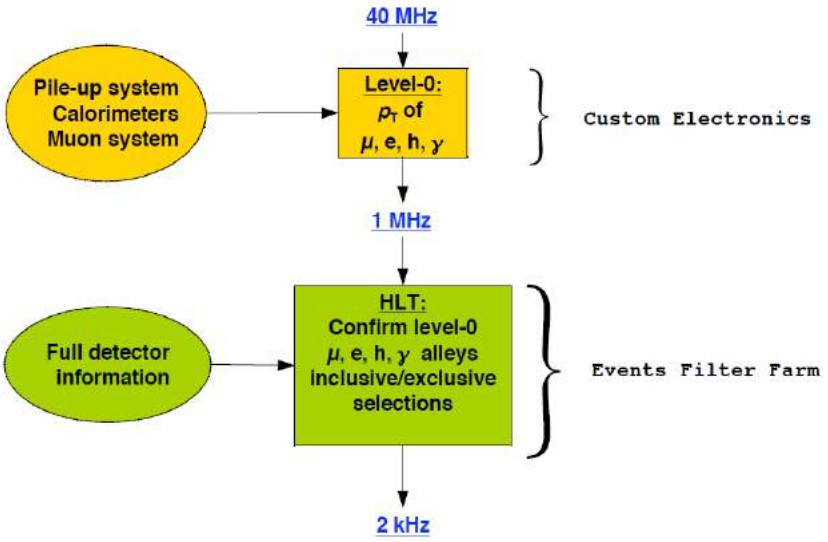


Figura 12 – Esquema do *trigger* do LHCb.

1.2.8 O sistema de *trigger*

O propósito do sistema de *trigger* no LHCb é detectar decaimentos raros e reduzir a taxa de 40 MHz de *bunch crossing* para um valor perto de 2 kHz, valor da taxa que eventos podem ser armazenados para análise offline. Logo, é de suma importância para o desempenho do experimento LHCb que o sistema de *trigger* seja eficiente e confiável.

O sistema de trigger do LHCb é implementado em dois níveis: O *Level Zero* (L0) e o *High Level Trigger* (HLT). O esquema do trigger é representado na figura 12.

1.2.8.1 O *trigger* Level 0

O *trigger* L0 opera a 5 mm em sincronia ao clock LHC. Ele utiliza eletrônica customizada para realizar uma decisão de *trigger* utilizando informação dos calorímetros, do sistema de múons e dos sensores de veto de *pile-up* do VELO. A decisão do L0 é baseada nos valores observados dos múons com maior p_T e os *clusters* de hádrons, elétrons e fótons com maior E_T. Em adição o detector de *pile-up* é usado para rejeitar múltiplos interações de cruzamentos.

A latência do L0, o tempo total entre *buch crossings* e o tempo de chegada do sinal de trigger L0 para todos a eletrônica de Front-End dos subdetectores é de 4 µs. A taxa máxima permitida de saída do *trigger* L0 é de 1,1 MHz.

1.2.8.2 O High Level Trigger

O High Level Trigger (HLT) é o segundo, e último, nível de trigger do LHCb, operando sobre os eventos passados pelo *trigger* L0. Ele é um grupo de algoritmos escritos em C++ que operam numa *farm* da ordem de 1000 nós de computadores de vários

núcleos de processamento. Ele tem acesso a informação do evento completa de todos os subdetectores. Seu objetivo é reduzir a taxa de entrada de 1,1 MHz, vinda de L0, para uma taxa de saída 5 kHz, onde os eventos são armazenados em disco para análise seguintes de forma online.

O HLT possui dois estágios para poder atender as limitações de tempo impostas pelos requisitos anteriores. São eles:

HLT1 *Trigger* responsável por reconstruir partículas no VELO e determinar a posição dos vértices primários (PV) no evento. Para limitar o consumo de CPU, a seleção de traços do VELO é feita baseada nos seus menores parâmetros de impacto para qualquer PV e suas qualidades. Para estes traços selecionados pelo VELO seus segmentos de traços nas estações T são procurados para determinar eu momento (p), chamado de *forward tracking*. A maioria dos eventendos não interessantes são rejeitados nesse estágio, reduzindo a taxa a aproximadamente 30 kHz.

HLT2 O HLT2 reduz a taxa de eventos para 5 kHz pela reconstrução de um evento completo com informação de todos os subdetectores e realiza a última seleção por *trigger*. Ele realiza um reconhecimento completo de padrões para encontrar todas as partículas de um evento, utilizando os traços do VELO como sementes. Então um grupo de seleções diferentes é aplicado atendendo aos requisitos físicos, procurando cobrir todos os decaimentos do méson B com um vértice deslocado e com pelo menos duas partículas carregadas no estado final.

1.2.9 Sistema Online

O sistema online, ilustrado na figura 13, tem a função de transferir dados da eletrônica de Front-End para armazenamento permanente sob condições controladas, incluindo dados de aquisição, controle do experimento e sincronismo com o clock do LHC. Por essa razão o sistema online pode dividido em três componentes chamados DAQ, o sistema de aquisição de dados; TFC, o sistema de temporização ou sincronismo e controle rápido e o ECS, o sistema de controle do experimento.

1.2.9.1 O sistema DAQ

O propósito do sistema de DAQ é transportar dados pertencentes a um dado *bunch crossing* e identificados pelo *trigger*, da eletrônica de FE para armazenamento permanente. Os princípio que regem o design de sua arquitetura são a simplicidade quanto ao número de protocolos e de componentes com funcionalidades simples; escalabilidade para reagir a parâmetros variáveis de operação, como tamanho de eventos, taxas do *trigger* ou as necessidade do CPU para os algoritmos de *trigger*; uso se links ponto a ponto

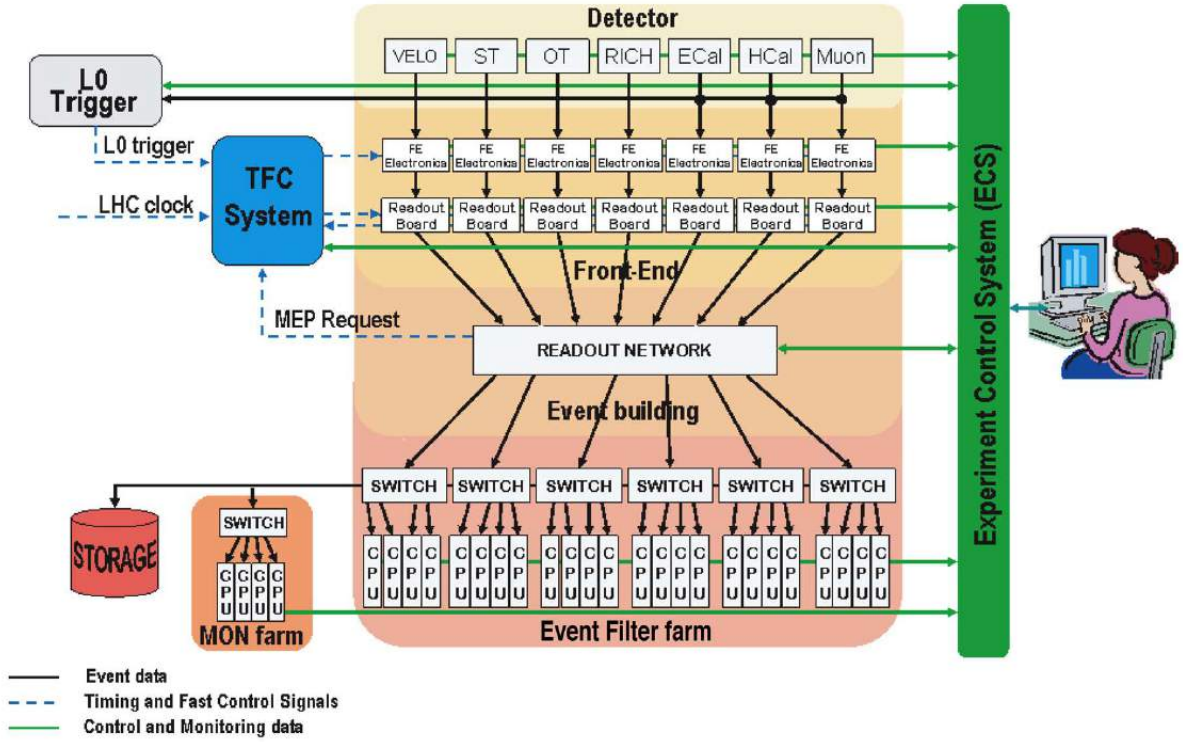


Figura 13 – Arquitetura do sistema online do LHCb

apenas e o emprego de componentes comerciais, chamando-os de produtos *Commercial off-the-shelf* (COTS), tanto quanto for possível.

Todos os dados recebidos da eletrônica de FE do experimento são coletados numa placa de leitura padronizada, denominada TELL1 e representada por um diagrama simplificado em 14. Estes dados são recebidos por interfaces ópticas ou elétricas analógicas e processadas em quatro FPGAs para pré-processamento, onde processamento em modo comum, supressão de zeros ou compressão de dados são realizadas dependendo das necessidades individuais de cada detector. Os fragmentos de dados resultantes são coletados por um quinto FPGA, chamado de SyncLink, e formatados em pacotes IP que são enviados em seguida ao sistema de DAQ por meio de placas Gigabit Ethernet. As interfaces da placa com o Experiment Control System (ECS) ocorrem por meio de um Credit-Card sized PC (CCPC) montados na placa. Sinais de clock e sincronismo, como triggers, são transmitidos através da interface do Trigger, Timing and Control (TTC) embarcada. O controle de fluxo ao sistema TFC é realizada por meio de um sinal de *throttle* criado pelo FPGA de SyncLink.

Na *farm* de CPUs, o algoritmo do HLT seleciona interações que, caso sejam consideradas relevantes, são selecionadas para serem enviadas ao armazenamento permanente. A redução esperada da taxa de *triggers* após o HLT é de 1 MHz para ∼2 kHz, resultando num fator de 500. É esperada do sistema de armazenamento ∼40 TB, que deve fornecer um espaço em *buffer* suficiente para lidar com possíveis interrupções na transferência para

o a área de armazenamento permanente no CERN.

Links do tipo Gigabit-Ethernet foram escolhidos para a tecnologia de transmissão, em maior parte por ser um padrão largamente adotado comercialmente, tendo como consequência o baixo custo de dispositivos. Isto garante velocidades de transmissão da ordem de 10 Mb/s a 10 Gb/s aliada a possibilidade de emprego de *switches* largos, com mais de 1200 portas por chassi.

A TELL1 oferece portas de 4 Gb Ethernet como estágios de saída dos dados de DAQ. Alguns deles são alimentados em grandes redes chaveadas fornecendo conectividade entre elas e os nós da *farm* de CPUs. Para superar o desperdício de banda significativo no quadro do padrão Ethernet, o conceito de pacotes multieventos (Multi Event Packets-MEP) foi idealizado, onde os dados de cerca de 10 *triggers* são coletados em um pacote IP e transferido em seguida pela rede.

O tamanho da *farm* de CPU rodando o *trigger* HLT é determinado pelo tempo médio de execução do algoritmo do HLT por evento, mas também possivelmente pela banda máxima num nó individual de processamento. Se o tempo de execução for pequeno, a largura de banda de entrada pode representar o fator limitante e o número de *boxes* pode precisar ser aumentado. Os algoritmos do HLT são executados numa *farm* de tamanho considerável.

A qualidade dos dados obtidos é checada numa *farm* de monitoramento que recebe eventos aceitos pela HTL e hospeda algoritmos criados por usuários, que determinam resultados como as eficiências dos canais ou a resolução de massa do detector.

1.2.9.2 Timing and Fast Control

O sistema de *Timing and Fast Control* (TFC) alimenta todos os estágios da leitura de dados do detector LHCb entre a FE e a *farm* de processamento online ao distribuir um sinal de clock síncrono a taxa de colisões do LHC, o *trigger* L0, sinal de reset síncrono e comandos de controle rápidos. O sistema é uma combinação de componentes eletrônicos comum a todos os experimentos do LHC e eletrônica customizada do LHCb. A arquitetura do TFC, mostrada na figura 15 pode ser descrita em tempos de três componentes principais, a rede de distribuição do TFC, a rede de regulagem (*throttle*) do *trigger* e o TFC mestre ou *Readout Supervisor*.

A rede de distribuição óptica do TFC com transmissores e receptores é baseada no sistema TTC, desenvolvido no CERN para uso pelos experimentos do LHC. Em adição a transmissão do clock síncrono ao feixe do LHC, o protocolo apresenta um canal de *trigger* de baixa latência e um segundo canal com os dados do usuários na forma de quadros usados para codificar os comandos de controle. O *switch* foi desenvolvido e introduzido na rede de distribuição para permtir um particionamento dinâmico do detector LHCb para

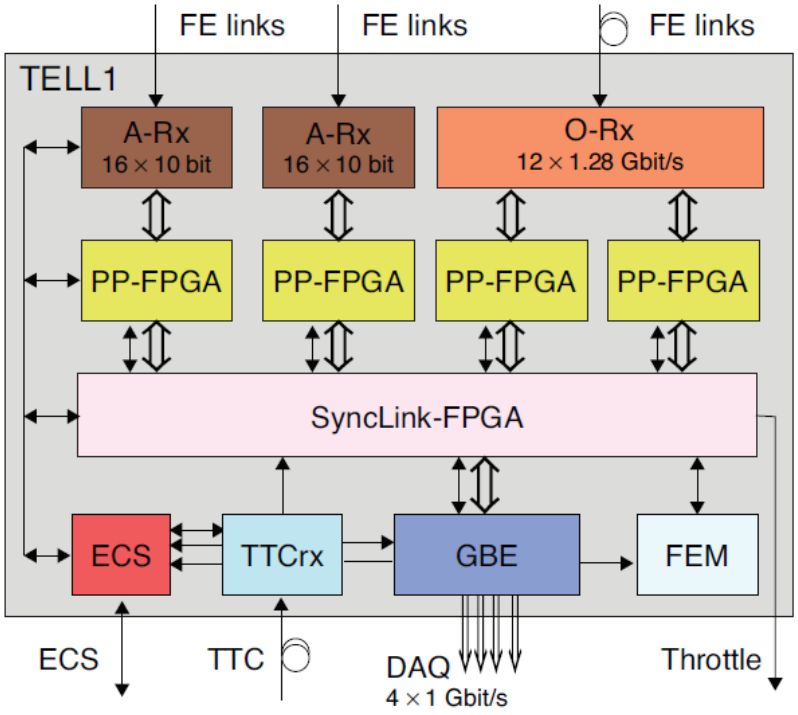


Figura 14 – Diagrama em blocos simplificado da TELL1.

suportar atividades independentes e concorrentes dos subdetectores como comissionamento, calibração e testes.

A rede óptica de regulagem é usada para transmitir *back-pressure* de partes assíncronas do sistema de leitura para o *Readout Supervisor* no caso de congestionamento do caminho de dados. A rede incorpora um *Throttle Switch* para suportar alguns requisitos do sistema de leitura, como o de ser particionável e de permitir que módulos realizem um OU lógico dos sinais de regulagem para cada subsistema localmente.

O coração do sistema, o *Readout Supervisor*, implementa a interface entre o sistema de *trigger* do LHCB e a cadeia de leitura. Ele sincroniza as decisões de *trigger* e comandos síncronos ao feixe com o clock do LHC e e com o sinal de órbits provido pelo LHC. Ele também é capaz de produzir uma variedade de auto-*triggers* para o testes e calibração dos subdetectores, realizar controle de *trigger* como uma função da carga no sistema de leitura. De forma a realizar balanceamento dinâmico de cargas entre os nós na farm de processamento *online*, o *Readout Supervisor* também seleciona e divulga o destino do próximo grupo de eventos às placas de leitura (*Readout Boards*) baseadas num esquema de crédito onde os nós das *farms* enviam requisições de dados diretamente ao *Readout Supervisor*.

O *Readout Supervisor* transmite para um banco de dados sobre a rede de leitura que é anexada aos dados eventos, e que contém o identificador do evento, a hora e a fonte do *trigger*.

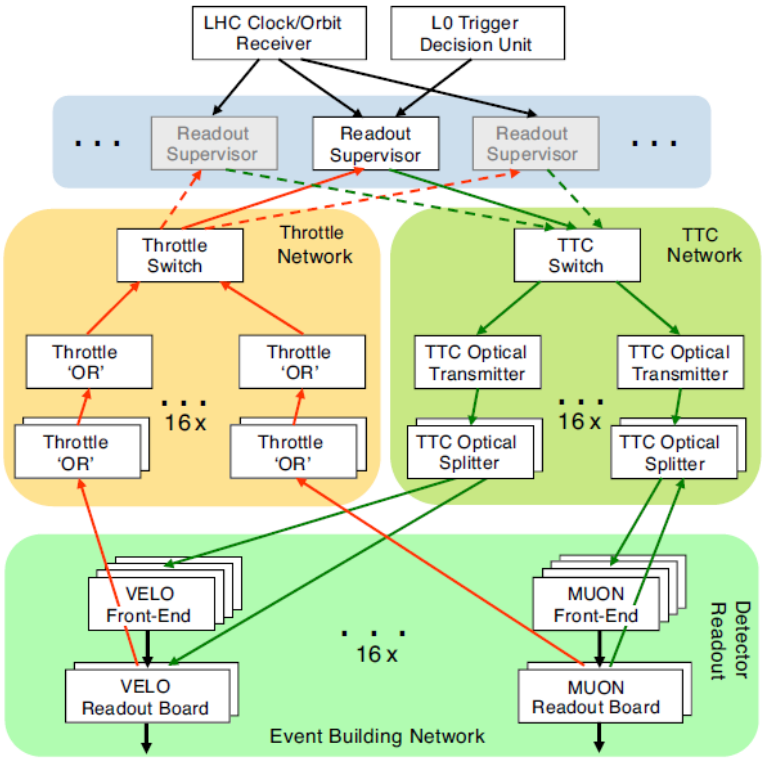


Figura 15 – Diagrama esquemático da arquitetura do TFC.

1.2.9.3 Experiment Control System

O *Experiment Control System* (ECS), sistema de controle do experimento em tradução livre, assegura o controle e monitoramento do estado operacional de todo o detector LHCb. Ele engloba não apenas os domínios de controle de um detector tradicional, tais como altas e baixas tensões, temperaturas, fluxos de gás, ou pressões mas também o controle e monitoramento dos sistemas de *Trigger*, TFC e DAQ . Os componentes de hardware do ECS são um pouco diversificados, principalmente como uma consequência da variedade de equipamentos a serem controlados, variando de *crates* padrão e fontes de alimentação para placas eletrônicas individuais. No LHCb, um grande esforço foi feito para minimizar o número de diferentes tipos de interfaces e barramentos eletrônicos.

Três barramentos eletrônicos de campo foram utilizados no experimento, o SPECS, Sigla para "The Serial Protocol for ECS", consiste num barramento serial de 10 Mb/s para controle de acesso à eletrônica de FE, *Controlled Area Network* (CAN) e Fast Ethernet. Os dois primeiros, SPECS e CAN, são utilizados para equipamentos localizados em áreas sujeitas a altas doses de radiação; enquanto o Ethernet é usado para controlar PCs e placas eletrônicas, através de CCPCs montados diretamentes sobre elas. Esta escolha permite o uso de PCs normais sobre interfaces padrões Ethernet para controle da eletrônica de leitura.

O software do ECS é baseado no PVSS II, um sistema SCADA (*Supervisory Control*

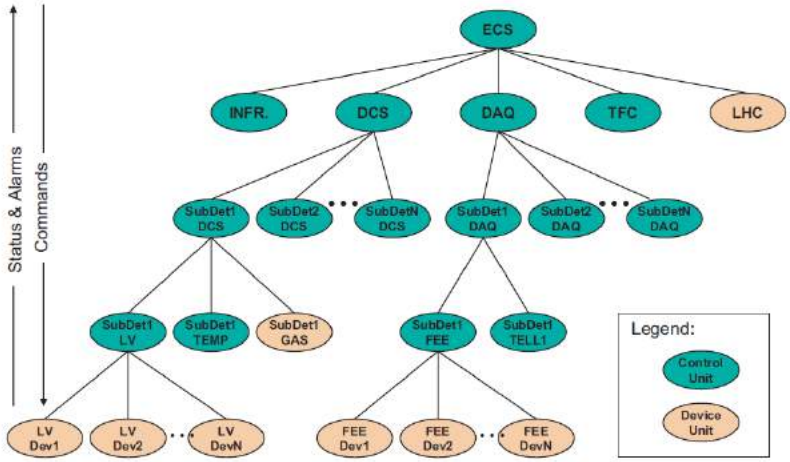


Figura 16 – Arquitetura do sistema ECS.

And Data Acquisition) comercial hoje chamado WinCC(3). Este kit de ferramentas provê a infraestrutura necessária para a construção do sistema ECS, como uma configuração de banco de daados e comunicação entre componentes distribuídos, bibliotecas gráficas para construir painéis de operação, e um sistema de alarme e componentes, como clientes OPC. A figura 16 mostra a arquitetura do ECS, projetado como um sistema hierárquico e distribuído baseado no PVSS.

As unidades de dispositivos, na figura 16, designam componentes de acesso de baixo nível que modelam o dispositivo físico e tipicamente se comunicam diretamente com o hardware. Em geral, eles implementam apenas uma máquina de estados bastante simples que é alimentada exclusivamente pela *Control Unit* associada. Exemplos de *Devices Units* são fontes de alimentação e processos de software, tais como os processos do HLT. As *Control Units* são subsistemas de alta tensão (HV), componentes que controlam o conjunto de *crates* de um subdetector de uma *subfarm* inteira da *Event Filter Farm*. As *Control Units* podem ser controladas por outras *Control Units*, permitindo a construção de uma hierarquia com profundidade arbitrária. O sequenciamento no sistema ECS é obtido utilizando o pacote de máquinas de estados finitas (*Finite State Machines*), baseada no SMI++, que permite a criação lógica complexa necessária para, por exemplo, implementar sequenciamentos elaborados ou recuperação automática de erros.

Os componentes distribuídos do sistema ECS são ligados a uma grande rede Ethernet que consiste de várias centenas de ligações Gigabit e Fast Ethernet.

Unidades de controle pode ser controlado por outras unidades de controle, para permitir a construção de uma hierarquia de profundidade arbitrária. sequenciamento do Estado no sistema ECS é conseguido usando um pacote de máquina de estados finitos, baseado em SMI ++ que permite a criação de lógica complexa necessária, por exemplo, para a implementação de sequenciamento elaborado ou recuperação de erro automática.

Os componentes distribuídos do sistema ECS são ligados com uma grande rede Ethernet que consiste de várias centenas de ligações Gigabit e Fast Ethernet.

1.3 Upgrade do LHC

O Conselho do CERN produziu em 2006 a Estratégia Europeia para a Física de Partículas (4), sucedido por uma atualização em maio de 2013 (5). O programa define as prioridades para a Física de Partículas no CERN levando em consideração a descoberta do bóson de Higgs em 2012. Ele contém uma mensagem chave para a realização do programa do LHC de Alta Luminosidade, chamado de High Luminosity LHC (HC-LHC):

> *Europe's top priority should be the exploitation of the full potential of the LHC, including the high-luminosity upgrade of the machine and detectors with a view to collecting ten times more data than in the initial design, by around 2030. This upgrade program will also provide further exciting opportunities for the study of flavour physics and the quark-gluon plasma.*

Um intenso programa de upgrade já começou no CERN tanto para a cadeia de aceleradores do LHC quanto para seus experimentos. O desempenho que o LHC previu para as próximas décadas necessita de uma intensa consolidação e upgrade dos detectores assim como estudos de simulação para entender suas otimizações com uma função do desempenho da física e de seus objetivos.

1.3.1 As fases do upgrade do LHC

Ainda em 2012, com as condições finais de operação do LHC, uma luminosidade integrada de cerca de $30 fb^{-1}$ foi acumulada com um luminosidade pico de $7,7 \times 10^{33} cm^{-2} s^{-1}$. Os dados das colisões foram armazenados utilizando energias de centro de massa de 7 e 8 TeV levando a grande descoberta de um boson de Higgs em 126 GeV (6, 7).

O LHC sofreu um upgrade para energias de 13 a 14 TeV com um espaçamento entre grupos de colisões de 25 ns e uma luminosidade instantânea de $10^{34} cm^{-2} s^{-1}$ durante o primeiro *Long Shutdown* (LS1, de 2013 a 2015). Os novos parâmetros permitem uma luminosidade integrada superior a 50 fb^{-1} antes do próximo *shutdown*. O planejamento atual de upgrades prevê dois *shutdowns* futuros (8), o LS2 e o LS3, mostrados na figura 17, que mostra uma linha do tempo das fases de upgrade do LHC.

Durante o LS2 a cadeia do injetor e o LHC serão melhorados com a integração do Linac4 no complexo de injeção(9). Isto torna possível um aumento da luminosidade de cerca de $2x10^{34} cm^{-2} s{-}1$ entregue durante a retomada da aquisição de dados em 2020, correspondendo a 55 a 80 interações por cruzamento (*pile-up*) com espaçamentos de colisões de 25 ns. Uma luminosidade integrada total de 300 fb^{-1} é esperada antes do LS3, estendendo o alcance para descoberta de nova física.

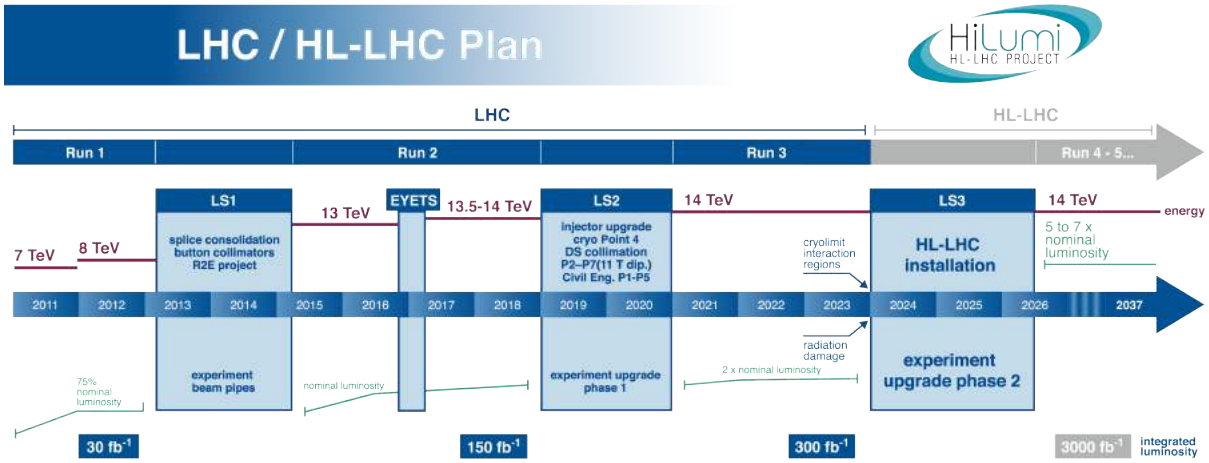


Figura 17 – Roteiro do upgrade do LHC para os próximos anos

Para o LS3 o LHC será atualizado para o High Luminosity-LHC (HL-LHC) para fornecer pacotes de colisões mais populados e densos nas regiões de colisão do ATLAS e CMS. Serão alcançadas luminosidades de pico virtuais de 5 a $7 \times 10^{34} cm^{-2} s{-}1$ no começo das injeções no LHC, correspondendo a um número de até 200 interações por cruzamento e uma luminosidade integrada total de 3000 fb^{-1}, um fator 10 vezes maior que o Run3.

A consolidação e upgrade do LHC são necessárias para alcançar o desempenho previsto para as próximas décadas. Estes objetivos apenas serão possíveis com a integração de novos recursos e substituição de algums componentes da maquinaria do LHC. As operações em se tornarão cada vez mais desafiantes para os experimenos com o crescimento das taxas de colisões e *pile-up*. Os principais upgrades do LHCb, retratado na seção 3, e do ALICE ocorrerão durante o LS2 com a reestruturação da eletrônica de aquisição de dados e a substituição de alguns de seus subsistemas com o intuito de melhorar a precisão das medidas e ultrapassar os limites impostos pelas restrições do detector, relacionados ao aumento na luminosidade integrada (10), (11).

As deterioração do desempenho do detector causada pelas doses de radiação integrada também deverão de ser tratadas na era do HL-LHC. O ATLAS e o CMS exigirão atualizações substanciais para acompanhar as melhorias do LHC. Um programa de upgrade sincronizadas com os *long shutdowns* foi proposto em (12), (13), (14). As principais atualizações ocorrerão no LS3 para substituir seus sistemas devido a danos causados pela radiação, obsolescência ou incapacidade de adquirir dados na mesma taxa que o HL-LHC, bem como para manter o desempenho adequado para a física num ambiente com alta taxa de *pile-up*.

2 O programa GBT

Devido à alta luminosidade do feixe prevista para o upgrade do LHC, o HL-LHC (8), os experimentos precisarão de links de dados de alta velocidade e componentes eletrônicos capazes de suportar altas doses de radiação. O programa GBT é parte do projeto Radiation Hard Optical Link (15), uma composição de projetos dedicados à transmissão de dados por links ópticos em experimentos físicos futuros do acelerador LHC, consistindo, basicamente, em um *chipset* e seu protocolo associado (16) (17).

O *chipset* do GBT aborda esta necessidade implementando um link de fibra óptica bidirecional e resistente à radiação de 4,8 Gb/s entre a sala de contagem e os detectores de cada experimento.

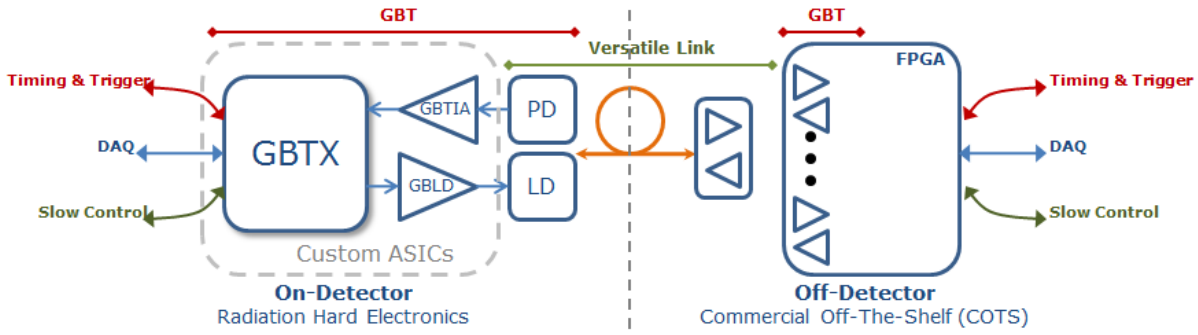
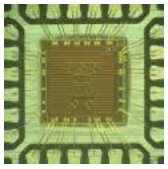


Figura 18 – Arquitetura do sistema GBT.

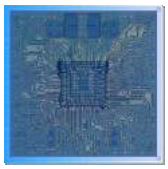
A arquitetura incorpora sinais de temporização e *trigger*, dados do detector e dados de controle, todos em um link físico, fornecendo, desta maneira, uma solução econômica para a transmissão de dados num experimento de física de partículas. O *chipset* é composto pelos seguintes elementos: GBTX (18), chip tolerante à radiação usado para transmissão de dados taxa de 4,8 Gb/s; GBTIA (19), um amplificador de transimpendância para recepção de dados via fibra óptica; GBLD, um alimentador de laser transmissão; GBT-SCA (20), um chip adaptador para geração de sinais de controle genéricos e o GBT-FPGA (21), implementação em FPGA do chip de transmissão de dados do sistema GBT. A disposição de todos estes componentes formam um sistema próprio, que é mostrado na figura 18. Nela é possível dividir o sistema em duas partes: À esquerda, os componentes localizados dentro dos detectores, em área submetida à radiação; à direita, os componentes fora dos detectores, localizados nas salas de contagem utilizando FPGA e transdutores ópticos comerciais e livres da influência da radiação.

Todo o *chipset* foi implementado na tecnologia microeletrônica comercial CMOS de 130nm, se beneficiando de sua resistência intrínseca a radiação ionizante.

(a) Vista interna do ASIC GBTIA.



(b) Vista interna do GBLD.



(c) Imagem interna do ASIC GBTx.

Figura 19 – Componentes físicos básicos do sistema GBT.

2.1 GBTIA

O GBTIA (19), representado na figura 19a, é um amplificador de transimpendândia que recebe dados a 4,8 Gb/s à partir de um fotodiodo. Este dispositivo foi projetado especialmente para lidar com a degradação do desempenho dos diodos PIN. Em particular o GBTIA pode lidar com altas correntes de fuga do fotodiodo com apenas uma degradação moderada de sua sensibilidade.

2.2 GBLD

O GBLD (22), representado na figura 19b, é um ASIC com função de alimentar o laser com dados modulados a 4,8 Gb/s. Ele implementa, adicionalmente, a equalização pré- e de-ênfase de forma programável, um recurso que permite sua otimização para diferentes respostas do laser.

2.3 GBTx

O GBTx é um chip tolerante à radiação usado para implementar links ópticos bidirecionais de alta velocidade para experimentos de física de altas energias, onde, da largura de banda da transmissão de dados total de 4,8 Gb/s, entre 3,2 e 4,48 Gb/s está disponível para o usuário.

O link provê ao usuário três caminhos de dados diferentes. São eles: Temporização e Controle de *Trigger* (*Timing and Trigger Control* - TTC), Aquisição de Dados (*Data Acquisition - DAQ*) e Controle Geral ou Lento (*Slow Control - SC*).

Na prática, os três caminhos lógicos não precisam necessariamente ser separados fisicamente para serem agrupados no mesmo link óptico, como na figura 20. O objetivo de tal arquitetura é permitir que um único link bidirecional seja usado para dados de aquisição, *trigger*, distribuição de controle de temporização, também chamado de controle rápido ou *fast control*, e controle geral e monitoramento de experimento, também chamado de controle lento ou *slow control*. Este link estabelece uma conexão de ponto-a-ponto, bidirecional (duas fibras), de latência constante que pode operar com alta confiabilidade num ambiente bastante severo, com elevado índice de radiação, típico de experimentos de física de altas energias do LHC.

O desenvolvimento do link proposto é conceitualmente dividido em duas partes distintas mas complementares: O chip do link GBT e os componentes ópticos do Versatile Link. O Versatile Link seleciona e qualifica fibras e componentes optoeletrônicos para uso sob radiação. O GBT desenvolve e qualifica ASICs resistentes à radiação necessários.

O link é implementado por uma combinação de componentes customizados e módulos comerciais prontos, classificados como *Commercial Off-The-Shelf* (COTS). Na sala de contagem do experimento os receptores e transmissores são implementados utilizando componentes COTS e FPGAs. Embarcado nos experimentos, os receptores e transmissores são implementados pelo *chipset* GBT e os componentes optoeletrônicos Versatile Link. Esta arquitetura distingue claramente a eletrônica das salas de contagem, denominada eletrônica de Back-End (BE), da eletrônica acoplada ao detector, denominada Front-End (FE), por causa da grande diferença dos ambientes quanto à radiação. A eletrônica empregada na FE trabalha num ambiente de radiação hostil, que exige componentes projetados de forma customizada. Os componentes da sala de contagem operam num ambiente livre de radiação e podem ser implementados empregando componentes COTS. O uso de componentes COTS na sala de contagem permite que esta parte da ligação obtenha a máxima vantagem das mais recentes tecnologias e componentes comerciais, como por exemplo FPGAs com muitas links de interface ((23),(24),(25),(26)), permitindo concentração e processamento de dados de forma eficiente vindos de muitas placas de FE para serem implementados de maneira bastante compacta e eficiente em sistemas de interfaceamento para *trigger* e DAQ.

2.4 GBT-FPGA

O GBT-FPGA (21) consiste na parte do sistema GBT destinada a ser usada fora do detector e livre de radiação, implementado num dispositivo FPGA comercial programado para ser compatível com o protocolo GBT e prover a interface para sistemas fora do detector.

Iniciado em 2009 com o objetivo de emular o GBTx e testar seus primeiros protótipos,

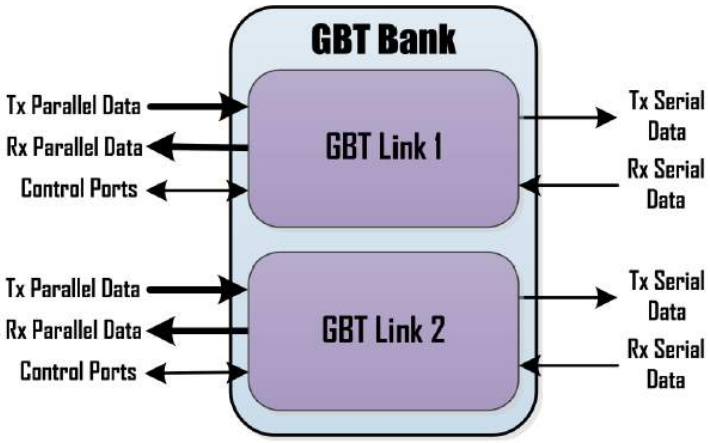


Figura 20 – Instância básica do bloco GBT-FPGA.

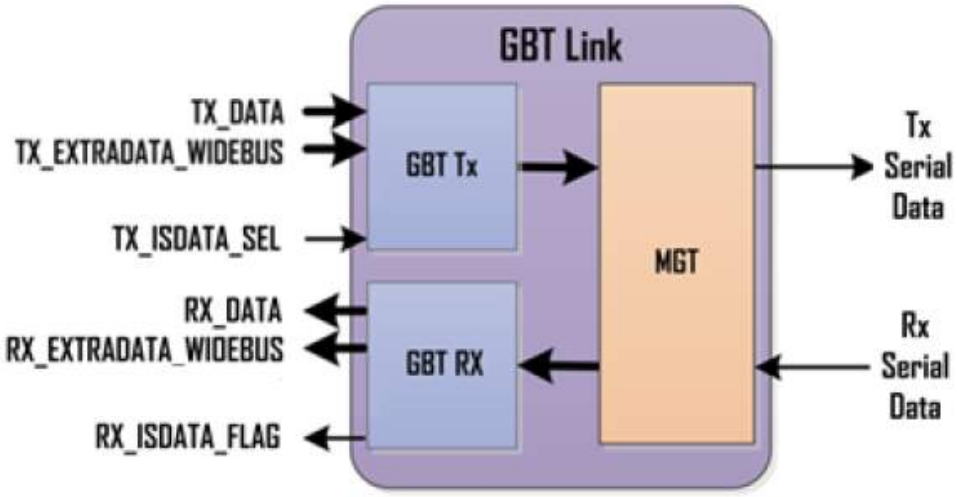


Figura 21 – Diagrama simplificado do GBT Link

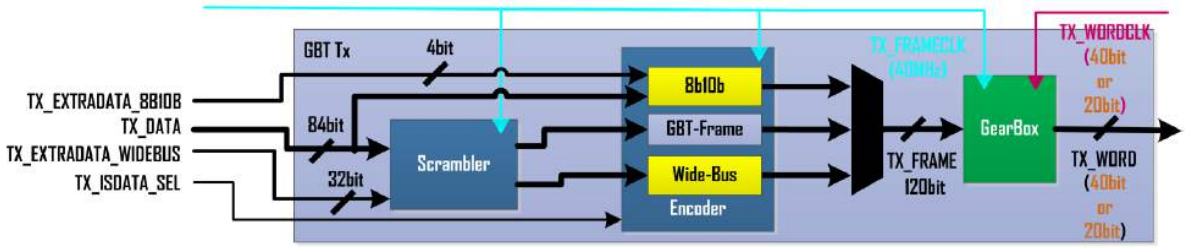


Figura 22 – Arquitetura interna do caminho de dados de transmissão no GBT-FPGA

o GBT-FPGA, hoje, é uma biblioteca completa, destinada a ser usada em FPGAs Altera e Xilinx, permitindo o uso de um ou mais links GBT. A forma como o GBT-FPGA é utilizado é definida partindo de um único módulo, mostrado na figura 20, chamado GBT Bank. O GBT-Bank pode incluir vários GBT Links. Cada GBT Link é composto por um GBT Tx 22, um GBT Rx 23 (juntos chamados de GBT Logic) e um Multi-Gigabit Transceiver (MGT). Os recursos de clock são externos ao GBT-Bank, de modo que o usuário pode utilizar dois tipos diferentes chamados *Standard* (Padrão) ou *Latency-Optimized* (Latência

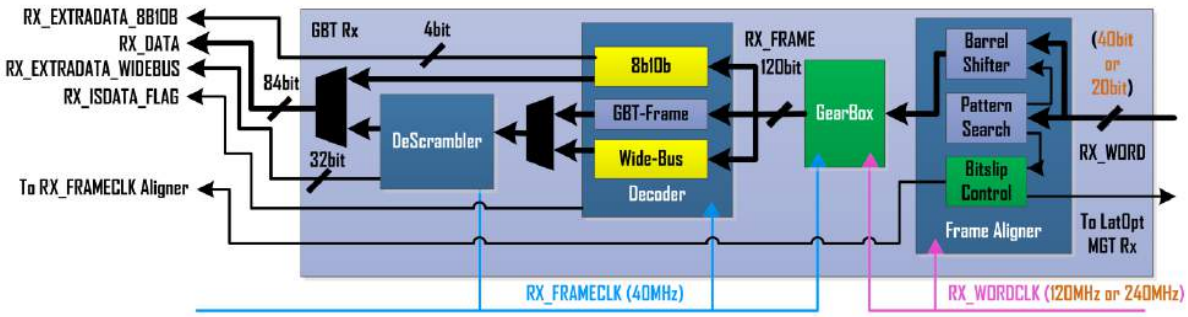


Figura 23 – Arquitetura interna do caminho de dados de recepção no GBT-FPGA

otimizada).

2.5 O Projeto E-Port

O projeto E-Port (27, 28), mostrado nas figuras 24 e 25, é uma das peças fundamentais para a implementação do fluxo de dados de *slow control* do sistema GBT. Compreende tanto uma interface de rede lógica para conexões de ponto a ponto, baseada no protocolo de comunicação *High-Level Data Link Control* (HDLC) (29), e também de um padrão elétrico próprio de interconexão chamado E-Link, compatível com padrão de sinalização diferencial *Scalable Low-Voltage Signaling* (SLVS) (30).

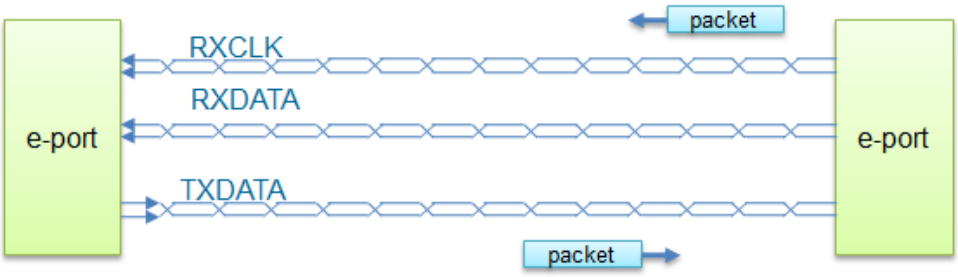


Figura 24 – Diagrama exemplo da comunicação de dois blocos E-Ports

Projetado pelo CERN, o bloco E-Port se trata de um bloco de Propriedade Intelectual (IP Core) que tem por objetivo estabelecer a comunicação do GBTx com outros circuitos integrados numa FE. O bloco foi construído utilizando tecnologia CMOS de 130 nm resistente a radiação, oferecendo a seguinte lista de recursos:

- Interface de comunicação do tipo E-Link.
- Interface lógica de dados Altera Atlantic (31) com barramento de 16 bits.
- Links seriais bidirecionais, cada link transmitido em um par de sinais diferenciais.
- Modo de operação Master, capaz de iniciar comandos de controle HDLC, ou Slave.

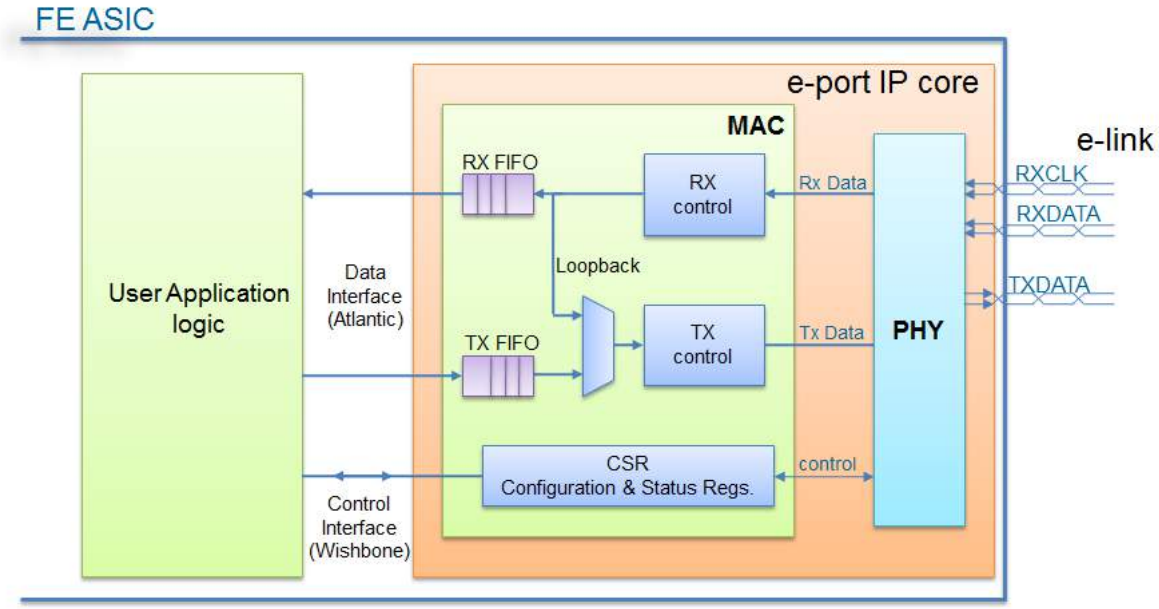


Figura 25 – Diagrama de blocos interno do bloco E-Port

- Operação em *Double Data Rate* (DDR), oferecendo velocidade de operação de 80 Mb/s quando utilizado um clock de 40 MHz.
- Protocolo de transmissão orientado à conexão HDLC.
- Tolerantes a efeitos de radiação do tipo *Single Event Upset* (SEU) (32), utilizando técnicas como Redundância Modular Tripla (Triple Modular Redundancy - TMR) (33) e Código de Hamming (7,4) (34).
- Bloco de detecção e correção de erros dos dados transmiticos CRC-16 CCITT (35), como parte do protocolo HDLC.
- Velocidades de 80, 120, 160 ou 200 Mb/s.

Complementando a parte lógica, a E-Link, por sua vez, é um padrão de interface elétrico para transmissão de dados sobre PCB ou cabos elétricos, cujo objetivo é transferir dados a uma taxa de até 320 Mb/s por alguns metros de distância. Baseada no SLVS, representado na figura 26. A E-Link se torna distinta por suportar níveis de radiação da ordem de MGy sob um campo magnético de 4T.

2.6 GBT-SCA

O chip GBT-SCA, representado na figura 27, de *GBT Slow Control Adapter*, é o chip especializado em prover interfaces eletrônicas de *slow control* (SC), isto é, ele provê

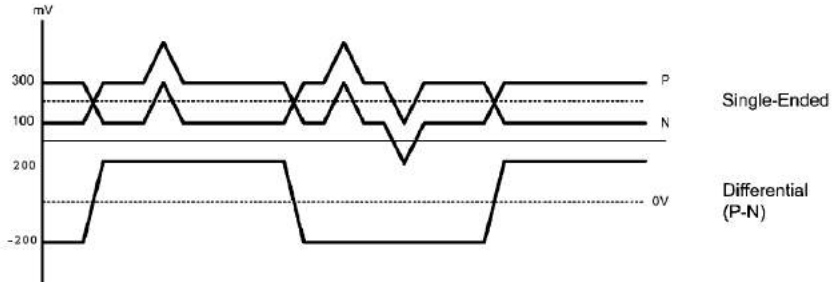


Figura 26 – Esquema do padrão de sinalização SLVS

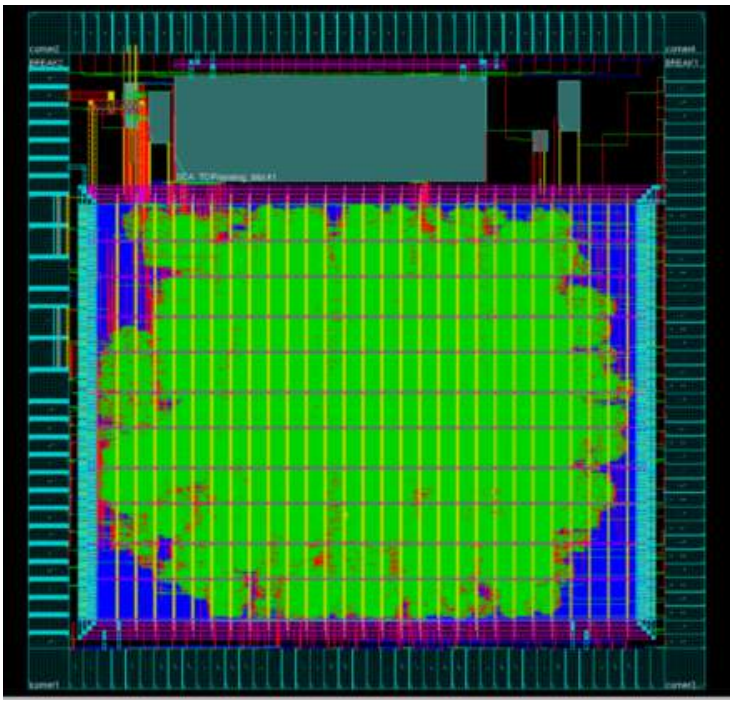


Figura 27 – Vista interna do ASIC GBT-SCA

funcionalidades de controle e monitoração para os mais diversos sensores e atuadores auxiliares presentes nas próximas atualizações de detectores do LHC e *High Luminosity LHC* (HL-LHC). Como características principais oferece suporte para seis tipos de barramentos eletrônicos ou funcionalidades que cobrem as necessidades de controle e monitoramento de dispositivos localizados nas FE de cada experimento, operando sob um clock de 40 MHz e links bidirecionais na velocidade de 80 Mb/s. O chip foi distribuído em duas versões, a primeira disponível em 2015 e a segunda a partir de 2016, onde as principais diferenças entre elas foram as melhorias no bloco para conversão de sinais analógicos em digitais.

Do ponto de vista do SCA e das informações de *slow control*, o protocolo GBT é totalmente transparente. Isto ocorre de forma que o GBT-FPGA encapsula as informações de controle vindas dos computadores ou sistemas eletrônicos na sala de contagem, a informação encapsulada trafega pelo links ópticos até chegar a eletrônica embarcada no detector, é então desencapsulada no GBTx que, finalmente, às entrega de forma integral ao GBT-SCA através de uma e-link. A figura 28 deixa claro o caminho de dados das

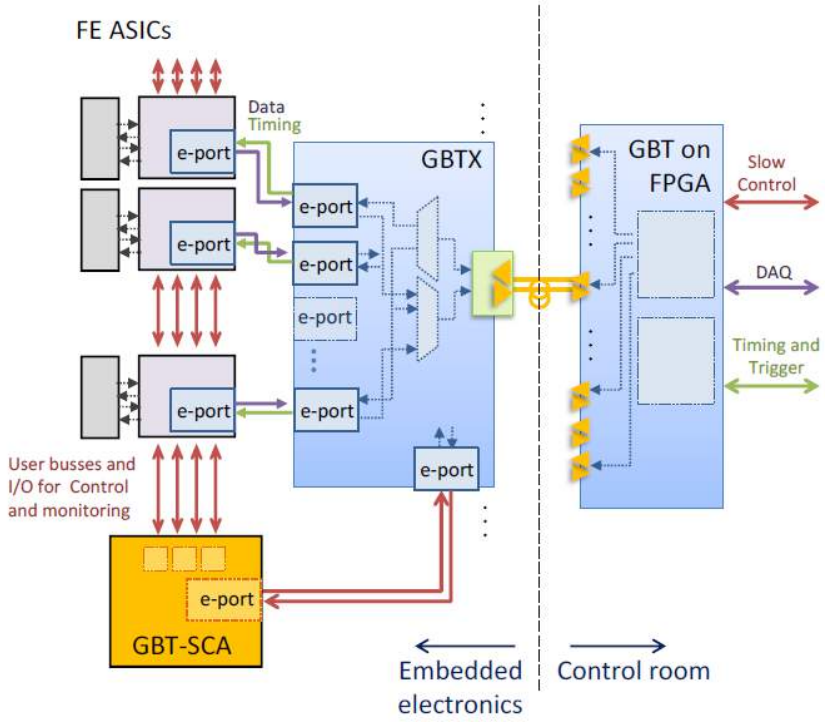


Figura 28 – Sistema GBT do ponto de vista do SCA.

informações de *slow control* ao SCA. A figura 29 mostra a arquitetura lógica do chip, onde são destacadas as unidades lógicas do chip, também listados abaixo:

2 E-Ports Duas interfaces E-Ports independentes usadas para comunicação com um chip GBT. Umas dessas interfaces é chamada de primária e precisa estar conectada em todo o caso. A segunda é utilizada se, por exemplo, dois GBTs desejarem compartilhar o mesmo SCA.

Network Controller (NC) Um bloco dedicado a supervisão do próprio SCA e ativação de seus canais.

Arbiter Árbitro que define qual canal deve ter controle sobre o link de comunicação com a interface de supervisão do chip, indiretamente enviando pacotes de dados ao usuário.

16 I2C Channels Dezesseis controladores de barramentos eletrônicos do tipo I²C independentes, capazes de realizar transações I²C com clocks nas frequências de 100 KHz, 200 KHz, 400 KHz ou 1 MHz nos modos de endereçamento de 7 ou 10 bits.

JTAG Channel Implementação de um controlador JTAG mestre programável pelo usuário, permitindo operações de dispositivos JTAG de 305 Hz a 20 MHz em pacotes de até 128 bits.

GPIO Channel Canal utilizado para o uso de 32 sinais como portas de entrada ou saída genéricas, oferecendo ainda suporte a interrupções.

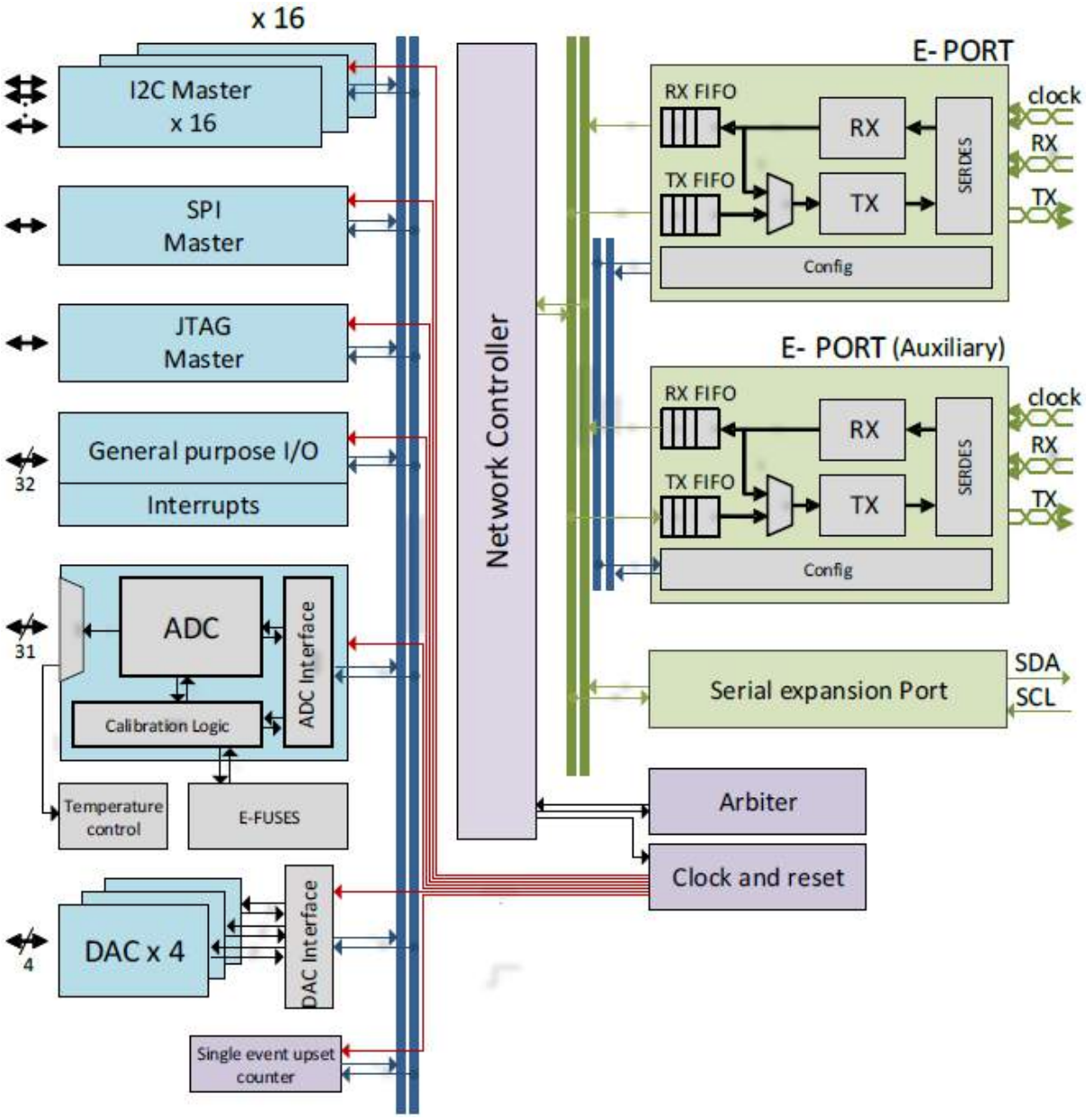


Figura 29 – Arquitetura interna do GBT-SCA.

SPI Channel Implementação de um controlador SPI mestre, permitindo operações de dispositivos SPI na taxa de 305 Hz a 20 MHz em pacotes de até 128 bits com suporte aos 4 modos padrões de operação. Este canal suporta 8 dispositivos diferentes acoplados a ele, através dos seus sinais de SS, mas ainda expansível caso faça uso de sinais do canal de GPIO.

DAC Channel Canal que implementa 4 conversores digitais-analógicos (DAC) independentes, cada um com 8 bits de resolução, para níveis de 0 a 1 V.

ADC Channel Canal que implementa um conversor analógico-digital (ADC) de 32 canais, onde um deles é reservado para o medidor de temperatura interno. O conversor ADC tem por características suportar sinais de entrada de 0,0 a 1,0 V, resolução de 12 bits

Figura 30 – Formato do quadro de dados GBT Frame.

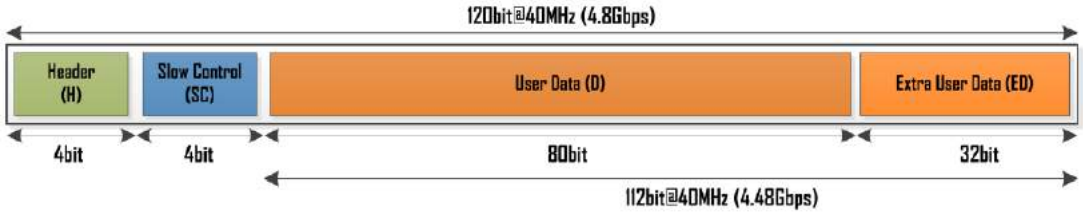


Figura 31 – Formato do quadro de dados Wide-Bus.

e tempos de conversão de 670 µs na primeira versão do chip e de 150 µs us na atual.

2.7 O protocolo GBT

O sistema GBT pode ser tratado como um bloco em caixa preta para usuários que trabalham nos experimentos do CERN. Tanto projetistas de BE como FE, no entanto, precisam saber como usar a interface de comunicação do GBT, como o formato do quadro de dados do mesmo. O ASIC GBTx e o firmware GBT-FPGA suportam dois formatos de quadros, chamados GBT Frame e Wide-Bus Frame e descritos respectivamente nas figuras 30 e 31, ambos consistem em uma palavra de dados de 120 bits.

O GBT Frame emprega o algoritmo de Reed-Solomon (36) para corrigir séries de bits com erros, causados por *Single Event Upsets* (SEU), comuns em ambientes sujeitos a radiação ionizante. Para isso um campo chamado *Forward Error Correction* (FEC) é usado para transmitir um código de verificação de 32 bits. Ao final sobram 80 bits de dados, permitindo uma taxa de transferência de $40MHz \times 80bits = 3,2Gb/s$. O modo de transmissão GBT Frame pode ser usado para transferir informações de Aquisição de Dados (DAQ); Temporização, Trigger e Controle Rápido (TTC) e Controle de Experimento (EC).

O modo Wide-Bus utiliza a mesma estrutura de campos do GBT Frame com exceção da ausência do campo de FEC. Não há verificação e correção de erros neste modo, no entanto o campo de dados é maior, possuindo 112 bits no total e permitindo uma taxa de transferência de $40MHz \times 112bits = 4,48Gb/s$.

Os dois tipos de quadros contêm outros dois campos em comum: o cabeçalho (*Header* - H) e o campo de *Slow Control* (SC), obrigatórios para trafegar informações de controle para a FE.

O campo de *Header*, de 4 bits, que é transmitido no início de cada quadro é

necessário para sincronizar o fluxo de dados a nível de quadro; isto é, a repetição de valores válidos neste campo indica que a transmissão está funcional, enquanto que repetições de valores inválidos indica que a conexão foi perdida. Para evitar que erros de transmissão ocorram devido a falsa identificação do *header*, este também é coberto pelo processo de correção de erros do campo FEC.

O último campo corresponde aos 4 bits do *slow control*, e é dedicado à execução de rotinas e operações de controle que não precisam necessariamente ter temporização precisa. Este campo pode ser subdividido em outros dois campos, o campo de controle interno (*Internal Control* - IC), responsável pelo próprio controle do GBTx na FE, e o campo de controle externo (*External Control* - EC), utilizado para controle do GBT-SCA principal na FE.

Os dois campos internos do SC utilizam o protocolo de comunicação E-Link/E-Port (27) para serialização e encapsulamento dos pacotes de controle para o GBTx e GBT-SCA, permitindo uma taxa de dados bidirecional de $40MHz \times 2bits = 80Mb/s$ cada; porém, os comandos e os formatos dos pacotes de controle dos GBTx e GBT-SCA são diferentes.

Mais informações do protocolo de transmissão GBT podem ser encontradas tanto no manual do chip GBTx (37) ou no manual do firmware do projeto GBT-FPGA (38).

3 O upgrade do experimento LHCb

Durante o período de 2010 a 2012 o LHCb apresentou uma boa performance (39) acumulando ∼3 fb^{-1} dados, havendo previsões de acumular outros ∼5 fb^{-1} entre 2015-2018. No entanto, o LHCb é limitado em termos de capacidade de transferência de dados a uma frequência de 1 MHz, sendo especialmente ineficiente para os canais hadrônicos no hardware do L0 *Trigger*. Assim, os documentos de *Letter Of Intent* (10), *Framework TDR* (40) e de *Trigger and Online TDR* (41) descrevem os planos para um detector atualizado que permitirão ao LHCb aumentar sua eficiência em decaimentos com múons em canais hadrônicos por um fator de 20 e coleta para 50 fb^{-1}.

O experimento LHCb está se preparando para atingir os regimes de maior luminosidade que serão oferecidos em 2018 pelo acelerador LHC, no CERN. Com a luminosidade de 2.10^{33}cm^{-2}s^{-1} a taxa de interações visíveis cresce de 10 MHz para 26 MHz. Além disso são previstos altos níveis de radiação e de ocupação de traços de partículas nos detectores, o que inviabiliza o reconhecimento de padrão implementados por algoritmos de determinação de trajetórias. Para se adequar a tal cenário serão realizadas modificações

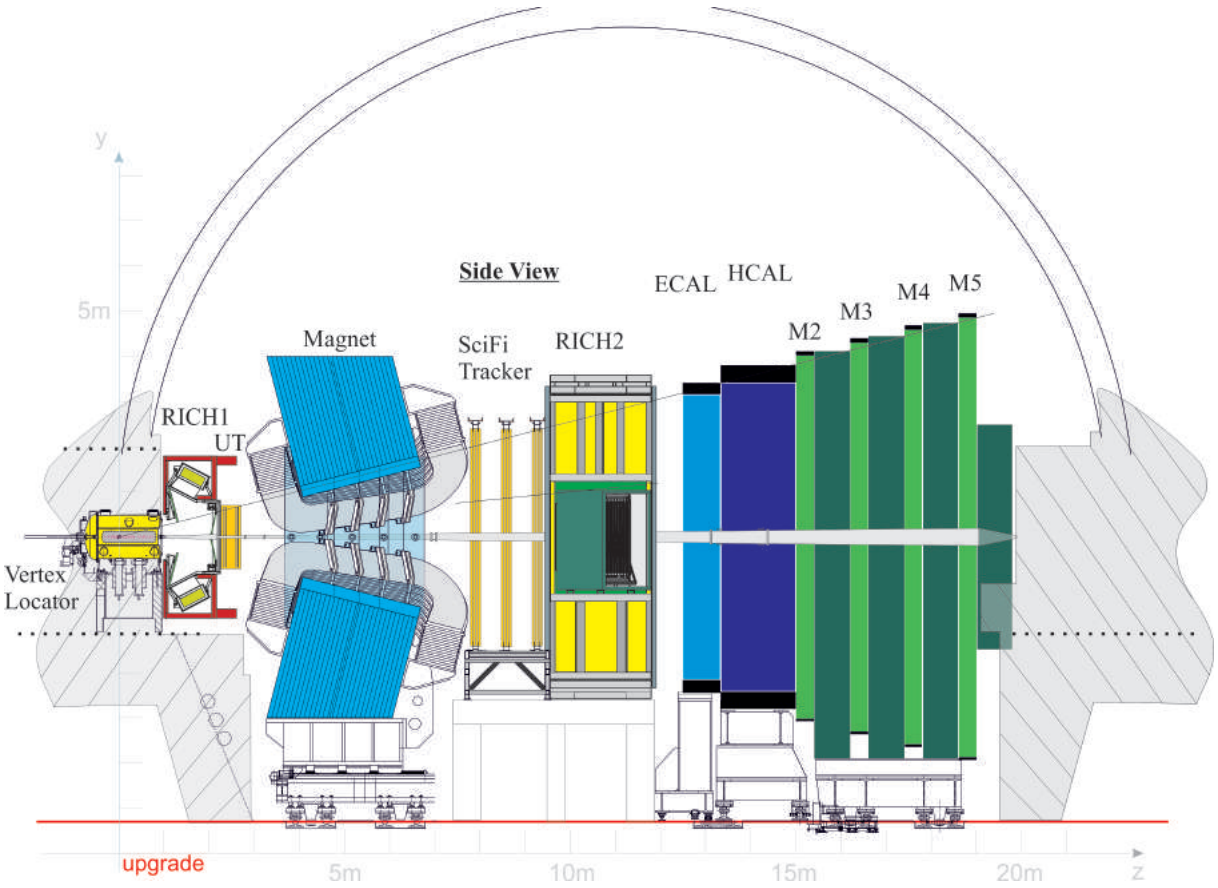


Figura 32 – Vista do detector LHCb atualizado.

importantes, com a substituição de vários detectores, da eletrônica de leitura (Front-End - FE) e de aquisição de dados. Uma das principais modificações se dará no sistema de Tracking. O sistema atual é dividido em três partes: *Tracking-UpStream*, composto pelos detectores Vertex Locator e *Trigger-Tracker*, seguido de dipolo magnético, e do sistema de *Tracking-Downstream*. Este último contém um setor interno, denominado *Inner Tracker* (IT), utilizando tecnologia de Silício, e um setor externo, denominado *Outer Tracker* (OT), que utiliza detectores a gás, denominados *straw-tubes*. De forma a reduzir os níveis de ocupação bem como de espalhamento múltiplo causado pela grande densidade de material presente na eletrônica de Front-End de detectores de Silício foi decidido que o sistema de *Tracking-Downstream* será unificado em uma única tecnologia, mais leve, e que preserve a mesma resolução espacial.

O novo sistema de Tracking que está sendo desenvolvido para o upgrade do LHCb, denominado SciFi, utilizará a tecnologia de fibras finas de 250 µm da Kuraray e leitura de SiPM da Hamamatsu, tanto para o *Inner Tracker* como para o *Outer Tracker*. São previstos aproximadamente 200 módulos de 500x52 cm^2 de área, contendo 16 SiPMs de 128 canais cada, totalizando 2048 canais para cada detector. Para tal está sendo desenvolvida eletrônica de Front-End dedicada, contendo um chip denominado PACIFIC, em desenvolvimento na França, o qual vai integrar tanto a parte analógica como a digital num único ASICS, uma FPGA resistente a radiação, e unidades de transmissão utilzando os chips GBT (GigaBit Transceiver), dedicado a envio de dados a 2,56 Gb/s, e GBT-SCA , dedicado a linha de controle, ambos em desenvolvimento no CERN para utilização em regimes de altas doses de radiação. O novo sistema de aquisição do LHCb fará uso de um módulo único de formatação dos dados denominado TELL40, controlado por um computador onboard CCPC (*Credit-Card PC*) integrado ao sistema ECS (*Experiment Control System*) do experimento. Este módulo pode ser configurado de diferentes maneiras, já que usa uma FPGA, de alta densidade e velocidade de processamento Stratix V, da Altera. Uma das configurações importantes no contexto deste projeto consiste no SOL40, a qual se insere no sistema de *Trigger e Fast Control* (TFC), como será visto na próxima seção.

3.1 O upgrade da arquitetura eletrônica de *readout* do LHCb

Com o objetivo de remover as principais limitações do detector LHCb atual, a estratégia para o upgrade do experimento (42) consiste basicamente em remover o

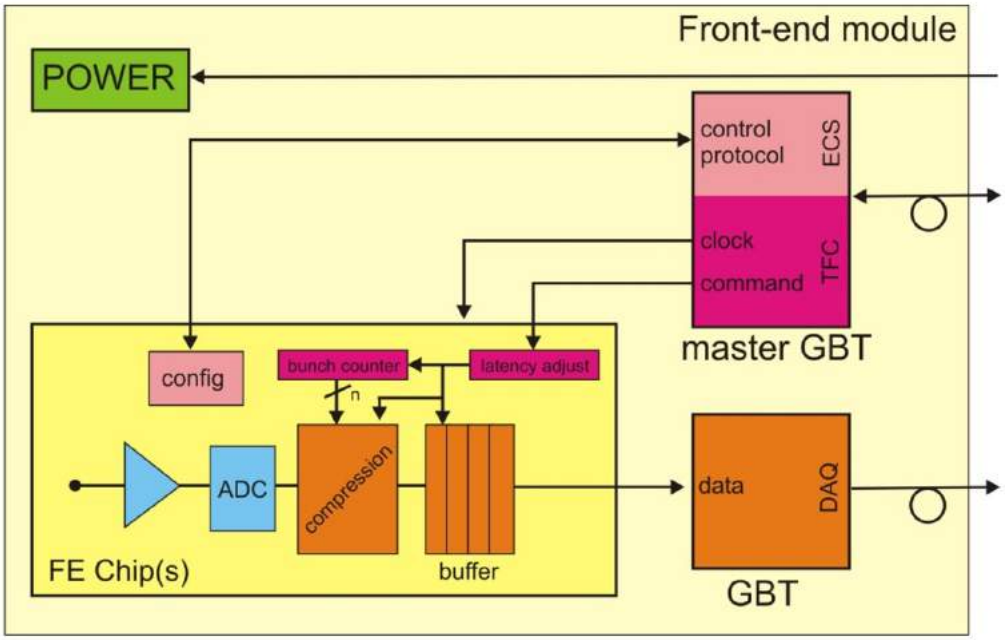


Figura 33 – Diagrama de um modelo de unidade de eletrônica de FE genérica no *upgrade*.

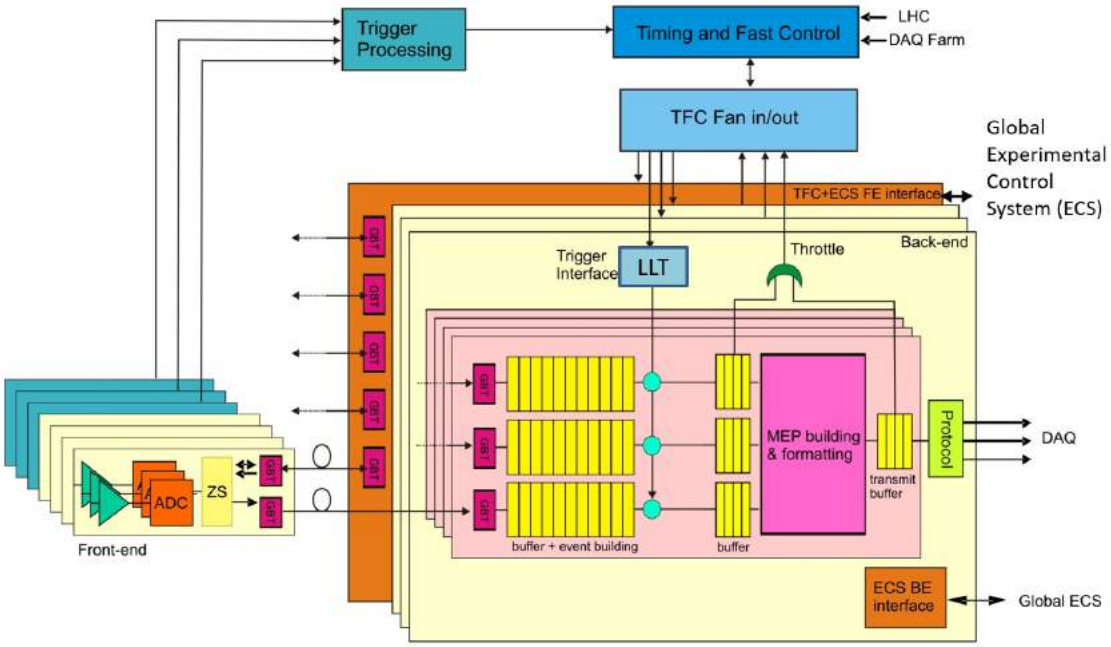


Figura 34 – Esquema simplificado da arquitetura eletrônica no *upgrade* do LHCb.

trigger em *hardware* de primeiro nível (*L0 trigger*), os eventos do LHC são registrados e transmitidos a uma velocidade de 40 MHz, resultando numa rede de sistemas de acquisição operando a ∼40 Tb/s. Todos os eventos estarão então disponíveis na *farm* de processamento onde um trigger implementado em software completamente flexível fará a seleção de eventos, onde sua saída de dados será a uma taxa aproximada de 20 KHz de eventos em disco. Isto irá maximizar a eficiência dos sinais a uma alta taxa de eventos.

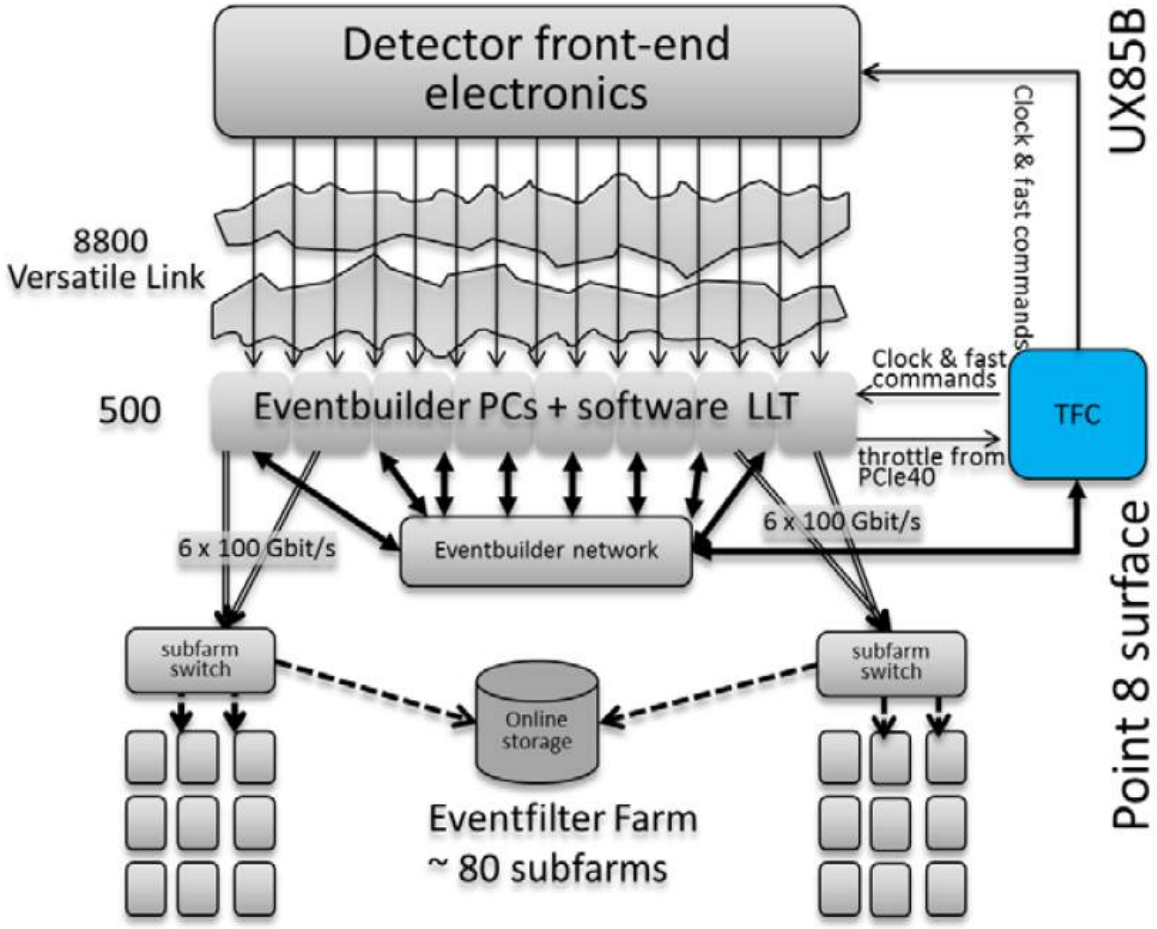


Figura 35 – Arquitetura do sistema de aquisição de dados do upgrade do LHCb

As consequências diretas deste método são que alguns dos subdetectores do LHCb terão que ser completamente reprojetados para suportar a luminosidade média de $2.10^{33}\text{cm}^{-2}\text{s}^{-1}$ e todo o detector será equipado com uma eletrônica de FE completamente nova. Somando a isso, a arquitetura de *readout*, termo usado para aquisição de dados, terá de ser reconstruída para suportar a nova largura de banda, da ordem de Tb/s, e o fluxo de dados a uma frequência de 40 MHz. A figura 35 mostra a nova arquitetura de *readout* (43) do programa de atualização do LHCb. Deve ser levado em conta, no entanto, que apesar do sistema final não possuir um trigger, o então trigger L0 feito em hardware será feito em software. Isto é normalmente referido como um trigger de baixo nível em software (*Software LLT*) e seu propósito principal é o de permitir uma instalação em estágios da rede de DAQ, gradualmente aumentando a taxa de leitura de dados dos atuais 1 MHz aos finais 40 MHz. Isto no entanto não irá alterar a taxa de eventos gravados na FE, que irá operar completamente sem um trigger, independente da taxa de saída de dados do DAQ. A figura 36 mostra um modelo abstrato da eletrônica de *readout* do LHCb atualmente, enquanto que a figura 37 mostra o modelo previsto para o upgrade do LHCb.

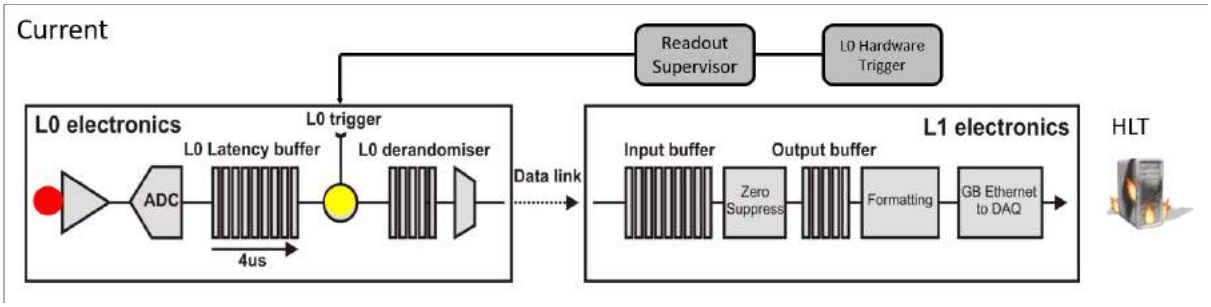


Figura 36 – Diagrama simplificado da arquitetura atual, tal como em 2016, da eletrônica do LHCb, com o L0 Trigger em hadware e L1 Trigger e HLT em software.

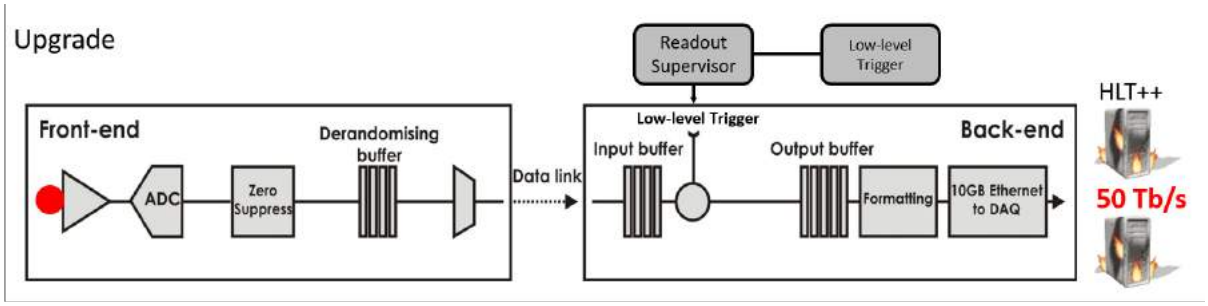


Figura 37 – Diagrama simplificado da arquitetura após o *upgrade* da eletrônica do LHCb, com o LLT e HLT.

Para manter o sincronismo do sistema de leitura, controlar a eletrônica de FE e distribuir o clock e as informações de sincronismo por todo o sistema de leitura, um upgrade do sistema centralizado de Controle Rápido e de Temporização (TFC, apresentado na figura 35) foi previsto, como um upgrade do sistema de TFC atual (44). O *upgrade* do sistema de TFC será então interfaceado com todos os elementos na arquitetura de *readout* ao se aproveitar das capacidades de comunicação bidirecionais dos links ópticos e transceivers do FPGA e de um alto nível de interconectividade. Em particular, o sistema TFC irá lucrar bastante dos recursos dos circuitos integrados do projeto *GigaBit Trasceiver* (GBT) (17), atualmente desenvolvidos no CERN para a comunicação da eletrônica de suas FEs. O sistema TFC também será responsáveis por transmitir informações de controle de baixa velocidade (ECS), onde suas placas eletrônicas baseadas em FPGA será ligada ao ECS global do LHCb.

3.2 O controle de alta e baixa velocidade para a FE por intermédio do sistema de TFC

A figura 38 ilustra em detalhes a arquitetura lógica do sistema TFC atualizado. Um grupo de supervisores do sistema de leitura (normalmente chamados de S-ODIN) gerenciam centralizadamente a leitura síncrona e assíncrona de comandos, distribuindo o clock do LHC e gerenciando a expedição de eventos. Cada S-ODIN é associada com uma

partição de subdetectores que na realidade são *clusters* de placas de *readout* (TELL40) e placas de interfaceamento (SOL40). Enquanto as TELL40s são dedicadas a leitura de fragmentos de eventos das FE e enviá-los para o DAQ para processamento em software, as SOL40 são dedicadas à distribuir dados de controle de alta e baixa velocidade para as FE, o que é feito ao retransmitir informações de temporização e clock no link óptico para a FE, e por anexar informações do ECS no mesmo datagrama de dados. Por aproveitar das características do chipset GBT (17), comandos de alta velocidade, clock e controle de baixa velocidade são então transmitidos no mesmo link óptico bidirecional. Isto é uma novidade importante em relação ao experimento LHCb atual onde os controles de alta e baixa velocidade são transmitidos por redes de dados diferentes.

Na FE, as informações síncronas do controle de alta velocidade são decodificadas e distribuídas por um chip GBT mestre em cada placa de FE, também responsáveis por recuperar e distribuir o clock de maneira determinística. As informações de controle de baixa velocidade são transmitidas aos circuitos integrados GBT-SCA por meio do GBT mestre. O GBT-SCA é capaz de eficientemente distribuir dados de configuração do ECS para os chips da FE através de uma gama completa de barramentos e interfaces eletrônicas, de maneira genérica (20).Os dados de monitoramento são enviados de volta no *uplink* do mesmo link óptico seguindo o mesmo caminho de dados em sentido contrário, do GBT-SCA para o GBT Mestre e para a SOL40 correspondente, em seguida.

A espinha dorsal em *hardware* da arquitetura de aquisição é uma placa eletrônica do tipo PCIe Gen3 hospedada num PC comum. O mesmo hardware é usado para as placas TELL40, SOL40 e S-ODIN, o que as diferencia é o firmware usado em cada uma. A placa será equipada com até 48 links ópticos bidirecionais, um FPGA Arria 10 da Altera e uma conexão do tipo PCIe Gen3 de 16x com um PC com vários núcleos.

A Figure 3 mostra o esquemático da implementação da fusão das fluxos de informações de controle em alta e baixa velocidade no mesmo link óptico para a eletrônica de FE (45) no firmware da placa SOL40.

Um bloco de retransmissão e alinhamento de TFC extrai um máximo de 24 bits da palavra de dados completa do TFC que foi transmitido pelo S-ODIN codificando comandos de alta velocidade, informações de temporização e vários sinais de reset. Estes 24 bits são então retransmitidos para o link GBT para ser enviado a FE. A palavra de dados é gerada a 40 MHz e transmitida com uma latência constante. A palavra de dados de TFC originada do S-ODIN é usada para reconstruir o clock localmente no FPGA para então ser utilizada para alimentar a lógica dentro do firmware.

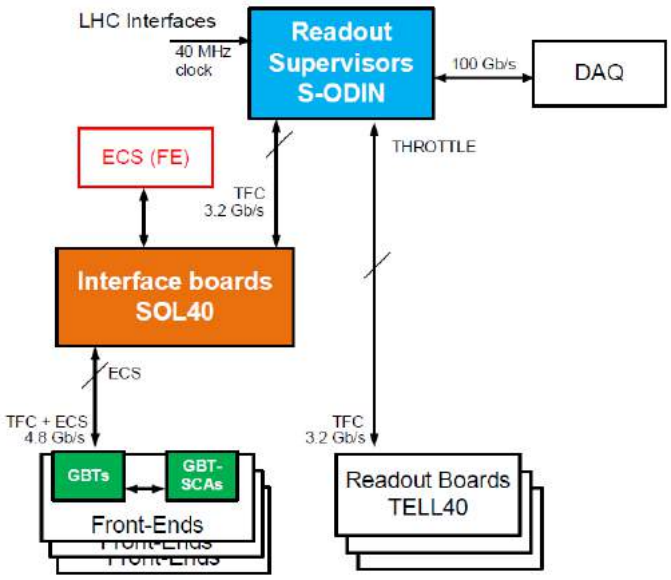


Figura 38 – Arquitetura lógica do sistema de TFC atualizado

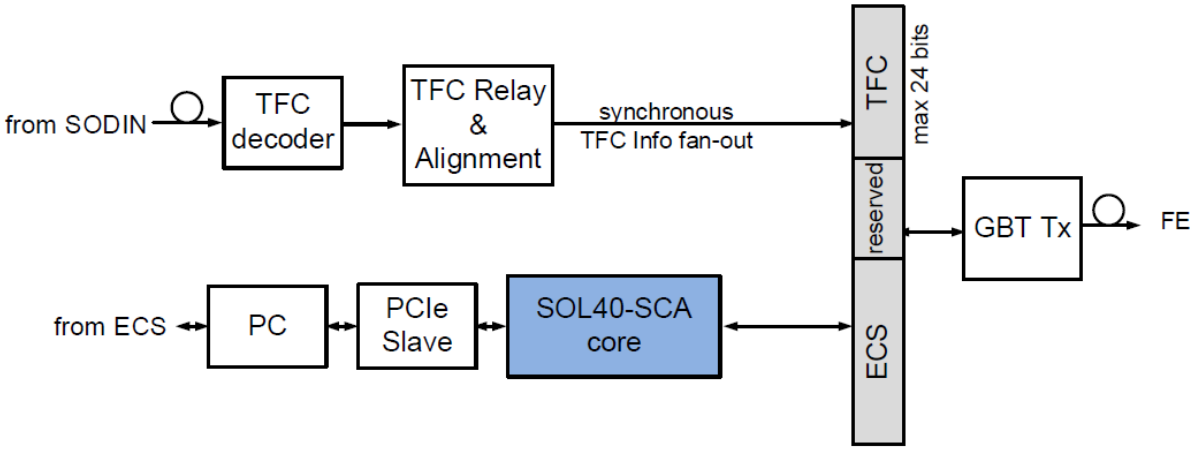


Figura 39 – Visão esquemática do algoritmo de união de informações de TFC e ECS no link GBT em direção a eletrônica de FE no firmware da placa SOL40

A respeito da parte de controle de baixa velocidade, o LHCb desenvolveu um núcleo de firmware, normalmente referido como SOL40-SCA, para controlar cada chip GBT-SCA localizado nas FEs, suportando todas as suas funcionalidades e protocolos. Sua localização dentro do firmware da SOL40 é destacado na figura 39. Isto é obtido porque o projeto foi desenvolvido para ser totalmente reconfigurável, ou seja, o protocolo do SCA escolhido pode ser selecionado em tempo real via comandos executados pelo sistema de ECS do LHCb (46) junto com dados de configuração. O destino de tais dados podem ser selecionados por uma máscara de configuração. O bloco foi desenvolvido para suportar um link GBT com até 16 GBT-SCAs conectados a ele. Este pode ser replicado tantas vezes quanto forem necessárias para atender a todos os links GBT conectados às placas SOL40. Ao final, o mesmo firmware viabilizará controlar toda eletrônica de FE do experimento LHCb, sendo um total de aproximadamente 2500 links ópticos bidirecionais e 90 placas SOL40.

Este bloco lógico é independente da tecnologia, desenvolvido em linguagem HDL, além de não fazer uso de qualquer elemento específico de alguma tecnologia e ser completamente agnóstico ao conteúdo do campo de dados. De forma simplificada ele é um gerenciador de SCAs genérico via links ópticos para a eletrônica de BE para os experimentos do LHC.

Parte II

Desenvolvimento

Esta parte do trabalho se dedica a descrever e explicar o desenvolvimento do projeto SOL40-SCA, dividido em dois capítulos.

O capítulo 4 descreve detalhes do desenvolvimento e implementação do projeto, suas motivações e contém duas seções para tratar do firmware do bloco FPGA E-Port e do bloco SOL40-SCA, onde o segundo utiliza e motivou a criação do primeiro.

O capítulo 5 apresenta as etapas de desenvolvimento do projeto ao longo dos seus dois anos e meio de existência, apresentando detalhes, hipóteses e escolhas usadas nas suas fases anteriores e planejamento para a futura versão v2.

4 O Bloco SOL40-SCA

Este capítulo descreve o objetivo deste trabalho, - O desenvolvimento de um firmware para interfaceamento da eletrônica de Back-End (BE) com os então futuros chips de controle GBT-SCA (20). Este último presente na eletrônica de Front-End (FE), para o upgrade do LHCb (44).

A necessidade do firmware foi enfatizada pela demanda do desenvolvimento do sistema de testes da eletrônica do subdetector SciFi (40), projeto de responsabilidade do grupo do LHCb no CBPF, onde ficou acordado, ao final de outubro de 2013, para André Massafferri, orientador deste trabalho e pesquisador no CBPF.

4.1 Descrição do bloco

O bloco se situa como parte fundamental do caminho de dados das informações de *Slow Control* (SC) do LHCb. Seu objetivo é fornecer uma maneira de controlar, com alto paralelismo e flexibilidade, vários dispositivos na eletrônica de FE interfaceados aos GBT-SCAs, através de links GBT. Suas principais características são listadas a seguir:

- Prover uma interface em hardware genérica (FPGA) entre o sistema ECS e a eletrônica de FE.
- Construir, codificar e decodificar pacotes compatíveis com o GBT-SCA.
- Serializar e desserializar pacotes de comandos enviados à eletrônica de FE de acordo as especificações do GBT e GBT-SCA.
- Suporte a todos os protocolos do GBT-SCA (SPI, I^2C, JTAG, GPIO e ADC+DAC).
- Suporte para todos os comandos e canais do GBT-SCA.
- Suporte para vários GBT-SCAs por link GBT e vários links GBT por FPGA.
- Possibilidade de retransmissão de pacotes e monitoramento da transmissão.
- Modularidade, i.e., componentes podem ser removidos se não forem necessários.
- Robustez, confiabilidade, programabilidade e flexibilidade.

Para entender o contexto do projeto, é fundamental explicar o programa GBT e a eletrônica de upgrade do LHCb. O GBT é o programa do CERN criado com o objetivo de prover um comunicação confiável entre a eletrônica de BE e FE nos próximos upgrades do

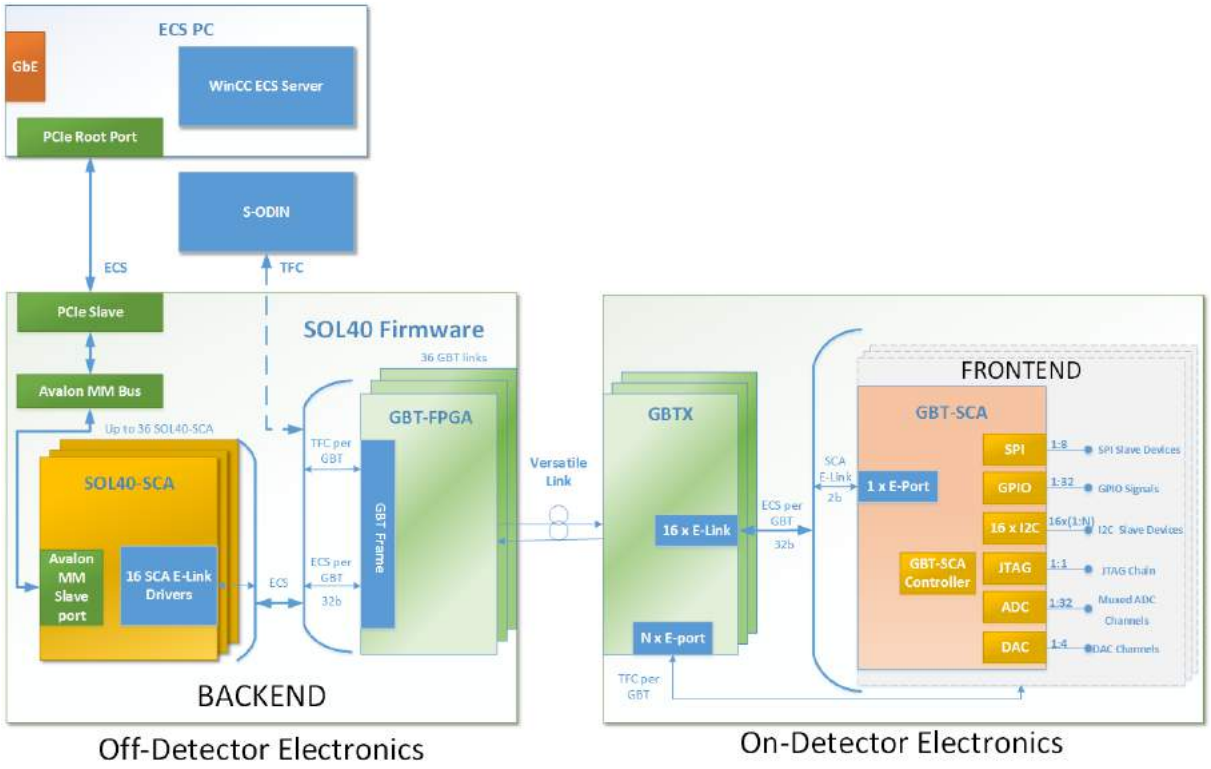


Figura 40 – Representação simplificada de todos os blocos envolvidos no caminho de dados do tipo ECS

experimentos do LHC, dentre seus elementos deve-se destacar o chip de interface de rede para fibras ópticas com tolerância a radiação a ser utilizado nas FEs, chamado GBTx, sua emulação lógica em FPGA flexível e implementado em alguns dispositivos comerciais, chamado GBT-FPGA, e seu chip adaptador para criação de canais ou barramentos comuns a funcionalidade de controle e monitoração de experimentos, chamado GBT-SCA. Detalhes do programa GBT podem ser encontrados na seção 2.

Como parte do upgrade da eletrônica do LHCb, o projeto está contido no firmware de Timing and Fast Control (TFC) e Experiment Control System (ECS). O bloco SOL40-SCA, nome técnico dado ao projeto, é implementado como parte do firmware da placa SOL40, utilizando o MiniDAQ (51) como seu protótipo em hardware, baseado num FPGA Stratix V 5SGXEA7N2F45C3 (52) da fabricante Altera com tecnologia de 28 nm.

4.2 O firmware FPGA E-Port

O núcleo da MAC Layer, parte fundamental do firmware SOL40-SCA explicado na seção 4.3.5, é um bloco chamado FPGA E-Port. Este bloco é baseado no original E-Port IP Core (20, 27), resumida em 2.5, mas com algumas diferenças fundamentais. O FPGA E-Port foi feito de maneira independente de dispositivos, em linguagem VHDL agnóstica quanto a fabricantes. Foi projetado sem as técnicas de tolerância a radiação,

como Redundância Modular Tripla (53) (chamada TMR ou *Triple Modular Redundancy*) e o código de Hamming (34) para integridade de dados das FIFOs, e sem uma porta auxiliar de comunicação, pois o bloco será usado na eletrônica de BE e a necessidade de redundância no caso de perda de comunicação ocorre na eletrônica de FE.

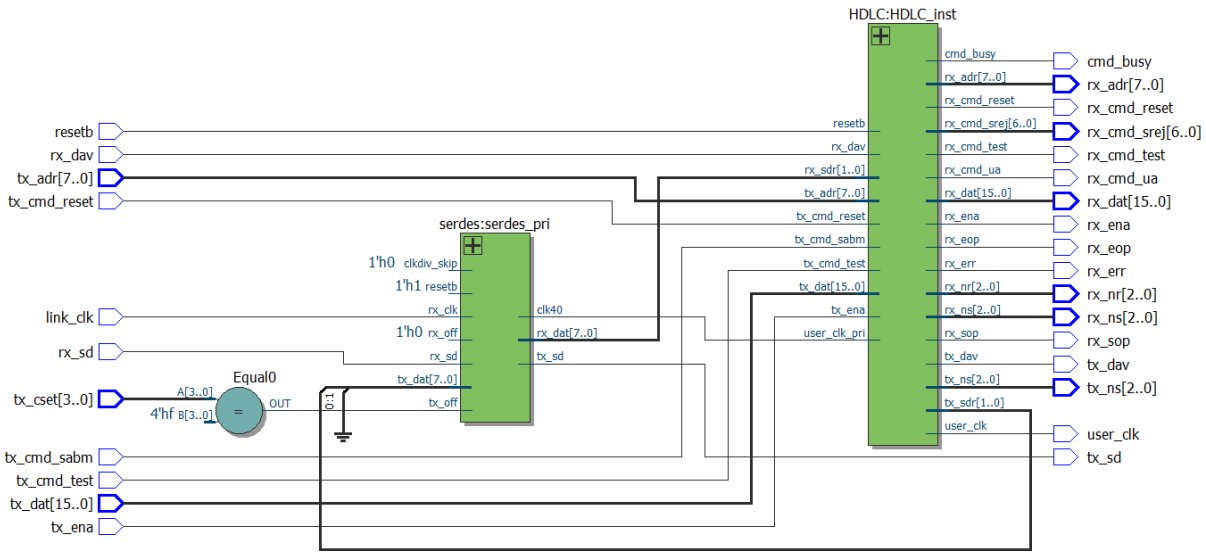


Figura 41 – Arquitetura do sistema de aquisição de dados do upgrade do LHCb

As funcionalidades fundamentais do bloco FPGA E-Port são as seguintes:

- Implementação em linguagem VHDL genérica do IP core E-Port, suportando todas as suas funcionalidades com exceção dos recursos de tolerância a radiação.
- Velocidade do Link de comunicação bidirecional de 80 Mb/s em DDR.
- Interface de comunicação compatível com o padrão E-link.
- Detecção de erros pelo utilizando o algoritmo de erros CRC-16 CCITT (35).
- Interface lógica de dados com padrão Altera Atlantic (31).
- Pacotes compatível com o padrão HDLC (29).
- Modos de configuração mestre (*Master*) ou escravo (*Slave*).

Os pacotes criados no FPGA E-Port, tal como a E-Port original, utiliza um subset de recursos do padrão HDLC. O formato do pacote gerado pelo bloco é mostrado na figura 42. Onde seus campos são os seguintes:

SOF *Start of Frame*, usados para delimitar o início ou o fim de um pacote. Como o HDLC utiliza a técnica de *bit-stuffing*, para manter o balanceamento DC dos sinais a serem

SOF	Address	Control	Payload	FCS	EOF
01111110	8 bits	8 bits	16*n bits	16 bits	01111110

Figura 42 – Composição de um quadro HDLC do bloco FPGA E-Port

transmitidos, a longa sequência de bits '1' quebra o balanceamento, permitindo sua distinção de outros tipos de dados.

Address Campo de endereço de dispositivos do padrão HDLC,já que a aplicação nos dados de SC só pode haver um GBT-SCA para cada conexão E-Port, ponto-a-ponto, este campo é ignorado no SOL40-SCA.

Control Informações de controle do protocolo HDLC. Os comandos que podem ser enviados pela E-Port mestre, localizado na BE no bloco SOL40-SCA pode enviar os seguintes códigos de comando em ordem de prioridade do maior para o menor:

RSET (*reset*) Para reiniciar o estado da E-Port escrava no GBT-SCA e torná-la ativa.

SABM (*connect*) *Set Asynchronous Balanced Mode*, solicita o estabelecimento de conexão com a E-Port escrava, útil para habilitar qual das duas E-ports do GBT-SCA estará ativa pra controle do mesmo.

TEST Utilizado para habilitar o modo de teste na E-port escrava, onde esta trabalha em *loopback* apenas retornando os pacotes enviados.

A E-Port escrava, como as localizadas nos GBT-SCA, podem enviar dois tipos de comandos:

SREJ *Selective Rejection* Enviado pela E-port escrava para a mestre como pedido para reenviar os dados de pacotes.

UA (*Command Acknowledge*) Enviado como resposta aos comandos RSET e SABM.

TEST Enviado como resposta ao TEST enviado pela E-Port mestre.

Payload Campo que contém os dados encapsulados pelo protocolo HDLC, ainda antes da etapa de *bit-stuffing*.

FCS *Frame Check Sequence*, campo que contém os dados usados para verificação de erros do pacote, gerados a partir do algoritmo CRC-CCITT de 16 bits.

EOF *End of Frame*, idêntico ao SOF, porém usado para delimitar o fim de um pacote.

ALMs used in final placement	ALMs used for memory	ALMs needed	DSP Blocks	M20Ks	Block Memory bits
295.2 (0.0)	331.5 (0.0)	20.0 (0.0)	0	0	0

Tabela 1 – Utilização de recursos de uma instância do FPGA E-Port em um FPGA Altera Stratix V.

A codificação do bloco em VHDL tem como arquivo principal o arquivo fpga_elink.vhd, mostrado na seção B.1 nos apêndices e representado em diagrama RTL na figura 41. Este faz parte da MAC Layer do bloco SOL40-SCA, apresentada na seção 4.3.5.

A implementação no modo E-Port mestre (Master), sem o uso dos sinais de endereço, deste bloco utiliza os recursos mostrados na tabela 1 quando instanciada no FPGA utilizado no código da BE da SOL40 no LHCb. Estes dados foram obtidos no relatório de construção de projetos de firmware no Altera Quartus (54).

4.3 O firmware SOL40-SCA

Esta seção se dedica a apresentar e explicar o funcionamento do firmware do projeto, até certo ponto semelhante a um *datasheet* ou folha de dados. Para codificação do projeto foram utilizadas os softwares Sigasi (55), um plugin proprietário especializado em desenvolvimento de sistemas eletrônicos na IDE do Eclipse (56), e o software Quartus II (54) da Altera; simulações foram feitas utilizando o Modelsim (57) da Mentor Graphics e o Doxygen (58) para geração de documentação e alguns dos diagramas do código. O desenvolvimento do código ocorreu em quatro fases ou versões, sendo retratada a versão estável e funcional atual, a v1.1, nas subseções seguintes.

O firmware do SOL40-SCA é formado por vários componentes codificados em linguagem VHDL 93 genérica (59). A figura 44 mostra um grafo de dependências de cada entidade do bloco SOL40-SCA. Como o seu propósito é ser um componente intermediário entre dois tipos de elementos em uma rede ponto a ponto, onde, de um lado está o sistema supervisório em software na sala de contagem, de outro os chips GBT-SCA nas FE.

A figura 40 mostra como o bloco se situa no contexto do fluxo de dados das informações de SC no upgrade do LHCb, o caso de uso principal deste projeto. Os sinais de controle tem início na geração de comandos nos software supervisório, construído como módulos SCADA para WinCC (3) no LHCb (60). Na parte superior da figura no bloco chamado ECS PC (Computador do ECS). O comando ECS gerado é transferido para a eletrônica de *Back-End* do experimento através de um barramento eletrônico *PCIe*. O pacote do comando ECS é endereçado ao bloco SOL40-SCA através de uma ponte de barramentos PCIe ao Avalon MM (61) dentro do firmware da SOL40.

Quando o comando chega ao SOL40-SCA este é armazenado numa fila de comandos,

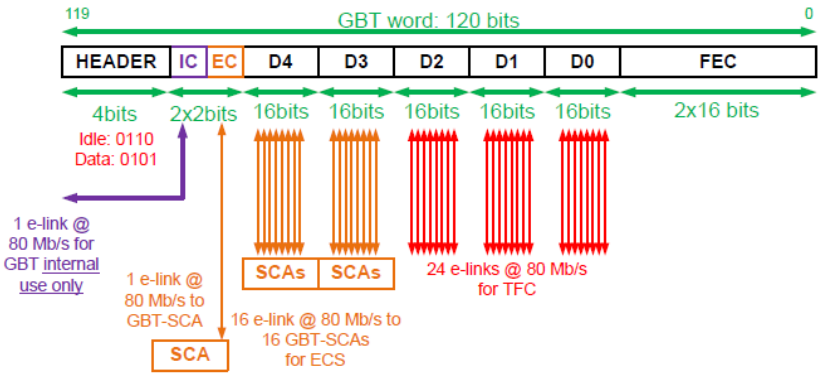


Figura 43 – Estrutura do quadro GBT utilizado pelos as informações de TFC e ECS no LHCb.

interpretado e traduzido numa série de comandos em pacotes de dados compatíveis com o GBT-SCA (20). Os pacotes SCA gerados são transmitidos indiretamente ao GBT-SCA associado ao elemento na eletrônica de FE em questão. Antes de serem transmitidos estes são encapsulados utilizando o protocolo HDLC e serializados, logo em seguida, para preencher parte do quadro de dados do sistema GBT (17, 21). Este dados serializados ocupam um barramento de 2 bits de largura que operam a 40 MHz definidos na parte lógica do padrão E-Link (28), se unindo, no LHCb, no quadro GBT dedicado a transmissão de controle e monitoração genéricos (ECS) e controle rápido e sincronia (TFC) (44).

O quadro GBT utilizado no LHCb(42), mostrado na figura 43, é o formato usado pelo bloco GBT-FPGA, implementação do GBT no FPGA da BE, e do GBTx, ASIC do sistema GBT na FE. Dentro dos 120 bits do quadro, se destacam os campos de IC (*Internal Control*), para o controle do próprio GBTx; o campo EC (*External Control*), usado para o controle do GBT-SCA principal na FE em questão, controlado pelo firmware do trabalho, além dos grupos, de 16 bits cada, D0, D1, D2, D3 e D4 de uso opcional compartilhado entre informações de ECS e TFC. Neste caso o ECS pode ocupar D3 e D4 como 8 interfaces E-Links cada, ou 16 SCAs associados. No total cada link GBT numa instância da SOL40-SCA é capaz de controlar 16+1 GBT-SCAs na FE do LHCb.

A transmissão de dados entre BE e FE ocorre através de links ópticos, definidos no projeto Versatile Link (62), onde, assim que um comando ECS chega no GBTx na eletrônica de FE, este é desencapsulado do protocolo GBT e transformado novamente num par de bits do padrão E-Link, ilustrado na direita da figura 40 dentro do bloco "On-Detector Electronics". O GBT-SCA associado ao E-Link do comando captura esse pacote serializado, interpreta e executa o comando, que pode ser uma leitura do canal ADC por exemplo.

Ao final da execução do comando o GBT-SCA obrigatoriamente gera um pacote de resposta ou *reply*, Este pacote de resposta atravessa o caminho contrário, se aproveitando da comunicação do link GBT para o caminho de dados controle ser bidirecional. Todas as

respostas do SCA correspondentes a um comando ECS são acumuladas no SOL40-SCA na forma de um pacote de resposta do tipo ECS, lido sob demanda, a partir do sistema supervisório.

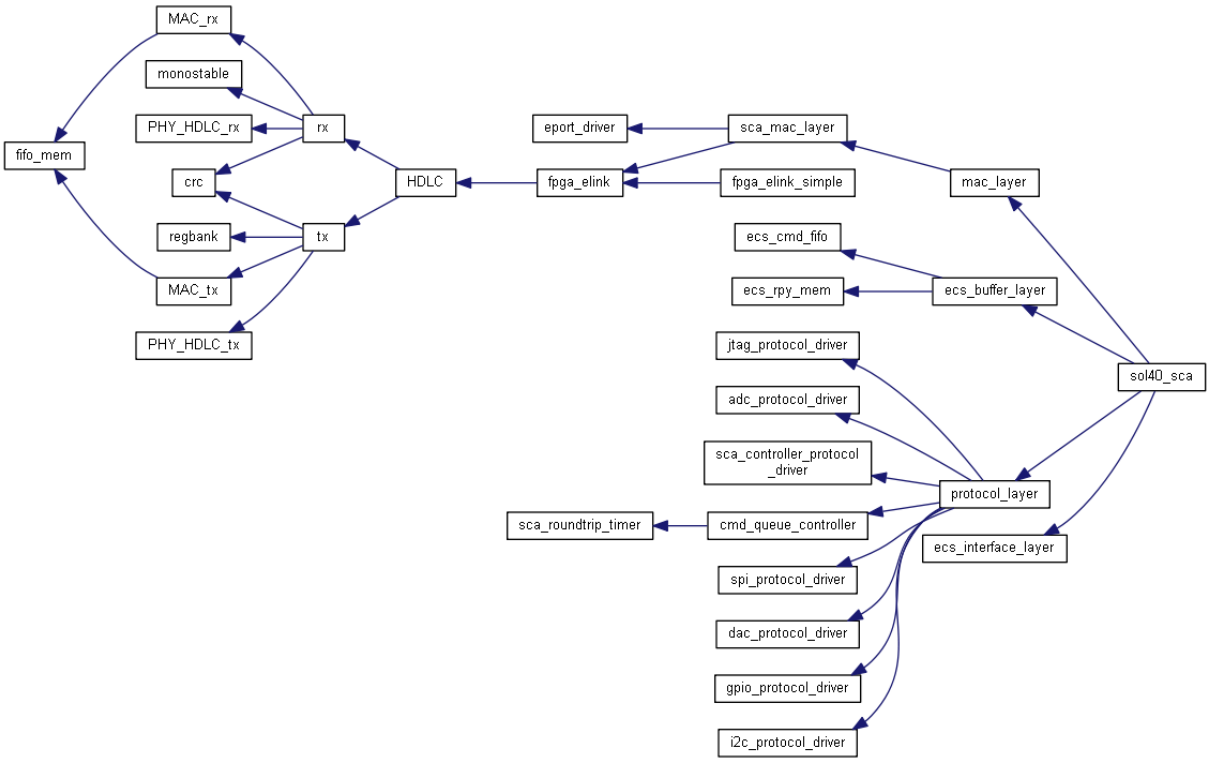


Figura 44 – Diagrama de dependências do projeto SOL40-SCA

O código do projeto é dividido em camadas de componentes, inspirado no modelo OSI (63) de comunicação em redes de computadores. A decisão de dividir o projeto em camadas foi dada ainda no início de seu design, e foi motivada por vários fatores. O primeiro fator foi o de prover um nível de abstração maior para o funcionamento do projeto, controlando seu nível de complexidade para que novos usuários ou desenvolvedores possam se familiarizar facilmente ao projeto; o segundo fator, não menos importante, foi o de limitar o fluxo de informações ou sinais que passam por cada bloco interno, isto é, modularizando e simplificando o escopo de cada unidade. Isto resulta numa grande vantagem nas fases de análise e testes do código, pois torna possível localizar com maior velocidade e precisão possíveis bugs e propor soluções (64). A figura 45 mostra as camadas da arquitetura do projeto, descritas nas seções 4.3.2, 4.3.3, 4.3.4, 4.3.5 separadamente.

4.3.1 Instruções de uso

O modelo de funcionamento do SOL40-SCA é construído em função do chip GBT-SCA, e é definido em três etapas: Configuração do chip, envio de pacotes de comandos e recepção de pacotes. A etapa de configuração do chip consiste no estabelecimento da conexão entre SOL40-SCA e GBT-SCA, através das regras de funcionamento do protocolo

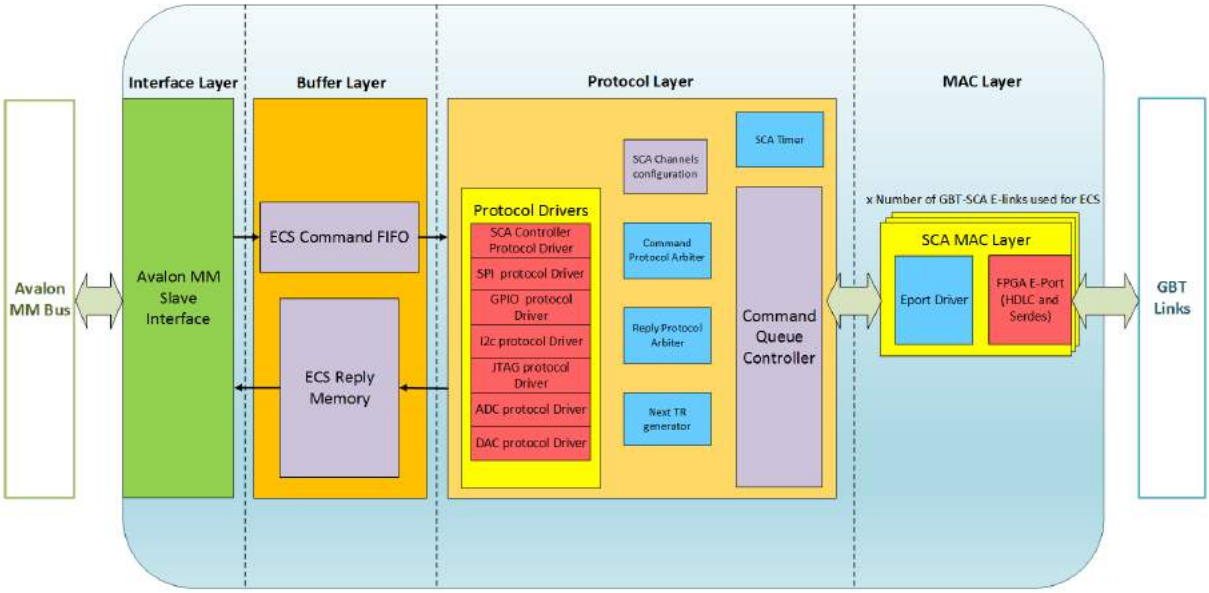


Figura 45 – Arquitetura atual do SOL40-SCA, na sua versão v1.1

HDLC (29) no bloco E-Port; ativação dos canais ou barramentos eletrônicos do SCA desejados, tratado na subseção 4.3.4.1, para finalmente enviar comandos a um canal específico.

O primeiro passo para uso do SOL40-SCA é sua instanciação num projeto de circuito digital aplicado a FPGAs. A tabela 2 lista os sinais externos ao bloco divididos em três grupos:

Control Interface A interface de controle, implementação de um barramento de dados Altera Avalon MM de 32 bits utilizada como interface de dados para o usuário controlar o bloco e GBT-SCAs por conseguinte (explicada em maiores detalhes na subseção 4.3.2).

Clocks Grupo que reúne os três sinais de clock que o bloco pode usar. O sinal ECS_CLK é usado na Control Interface e os sinais txClock40 e rxClock40 na comunicação com uma implementação do GBT, por padrão o bloco GBT-FPGA. Os sinais de clock GBT necessariamente operam a 40 MHz, no entanto, dependendo do tipo de implementação (38) estes podem apresentar defasagem entre si. Na versão atual do SOL40-SCA o sinal ECS_CLK deverá ser o mesmo sinal de clock usado em txClock40, ou problemas de *Clock Domain Crossing* (CDC) (65) poderão inviabilizar o uso do projeto, no firmware da SOL40, da eletrônica de upgrade do LHCb, os três sinais são iguais, evitando quaisquer tipo de problemas.

GBT E-Links Cada E-Link consiste num par de sinais que podem ser associados a um GBT-SCA na FE, este grupo contém um vetor de pares de sinais/E-Links que podem

Signal name		Direction	Meaning
Control Interface	ECS_ADDRESS_i[ECS_address_size-1:0]	In	Address signals to access the Register Map, equivalent to the address signal of an Avalon MM interface.
	READ_ECS_RESPONSE_i	In	Asserted to indicate a read transfer, equivalent to the read signal of an Avalon MM interface.
	ECS_RESPONSE_o[31:0]	Out	Data read signal, equivalent to the readdata signal of an Avalon MM interface.
	ECS_RESPONSE_VALID_o	Out	Signal to indicate the data contained in the ECS_RESPONSE_o signal is valid, equivalent to the readdatavalid signal of an Avalon MM interface.
	ECS_COMMAND_i[31:0]	In	Data write signal, equivalent to the writedata signal of an Avalon MM interface.
	WRITE_ECS_COMMAND_i	In	Asserted to indicate a write transfer, equivalent to the write signal of an Avalon MM interface.
Clocks	ECS_CLK	In	Clock used on the user control interface.
	txClock40	In	Clock used on the transmitter of the GBT.
	rxClock40	In	Clock used to on the receiver of the GBT.
	SRES_i	In	Reset signal, synchronous to the txClock40 clock.
GBT E-Links	tx_sd[0:SCA_Count-1][1:0]	Out	Array of Elink Tx data signals, where each Elink Tx may be associated to a pair of bits on a GBT transmitter frame.
	rx_sd[0:SCA_Count-1][1:0]	In	Array of Elink Rx data signals, where each Elink Rx may be associated to a pair of bits on a GBT receiver frame.

Tabela 2 – Sinais externos do SOL40-SCA

ser associadas a pares de bits em posições do quadro do GBT-FPGA pelo usuário, tanto para transmissão de dados nos sinais tx_sd quanto recepção nos sinais rx_sd.

O SOL40-SCA é controlado por meios de registradores endereçáveis através da interface de controle. Cada registrador possui 32 bits com o bit de maior índice mais à esquerda. Na tabela 3 estão listados seus endereços, suas permissões ou direções, como registradores de escrita (W) ou leitura (R), e suas funções.

O primeiro registrador, chamado ECS Control Register, é responsável pelo controle do próprio bloco e também único que não tem uma direção bem definida. Isto se deve ao seus campos internos terem suas próprias direções, mostrados na figura 46. As funções de cada campo são descritas a seguir:

Status of the link Par de bits que define o estado de uma conexão GBT-SCA. Pode assumir os valores 0x0 quando sob estado de RESET (nunca observado em condições normais do design); 0x1 em estado CONNECTING, isto é, quando o bloco tenta continuamente estabelecer conexão a um GBT-SCA; 0x2 quando em estado ACTIVE, quando o link ao GBT-SCA em questão está ativo e operacional; por último, 0x3 para DISABLED, quando a conexão foi desativada intencionalmente pelo usuário.

Reset Core Bit de escrita apenas, usado para reiniciar ou ressetar todo o firmware.

New ECS Reply Bit de leitura que indica que há um novo pacote de resposta disponível para o usuário ou sistema SCADA.

Send ECS Command Bit de escrita responsável pela ação de executar um novo comando ECS no firmware, destinado a uma ação em um GBT-SCA na FE.

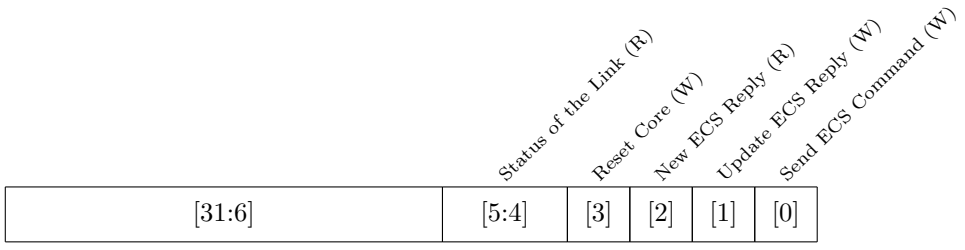


Figura 46 – Composição do registrador ECS Control Register

Os comandos ECS são armazenados antes de serem executados pelo bloco nos registradores ECS Command Header 0, ECS Command Header 1 e nos ECS Command

End.	Nome	Dir.	Significado
0x00	ECS Control Register	-	Registrador de Controle e Status.
0x04	ECS Reply Address	R/W	Endereço de checagem dos pacotes de ECS Replies.
0x08	ECS Command Header 0	R/W	Endereço e código de comando do ECS Command
0x0C	ECS Command Header 1	R/W	Comprimento de pacote e informações específicas de protocolo.
0x10	ECS Command Data 0	R/W	Primeiros quatros bytes de dados do ECS Command
0x14	ECS Command Data 1	R/W	Segundos quatros bytes de dados do ECS Command
0x18	ECS Command Data 2	R/W	Terceiros quatros bytes de dados do ECS Command
0x1C	ECS Command Data 3	R/W	Quartos quatros bytes de dados do ECS Command
0x20	ECS Reply Header 0	R	Endereço e código de comando da ECS Reply.
0x24	ECS Reply Header 1	R	Informaçoes de Erro, Comprimento de pacote e específicas de protocolo da ECS Reply
0x28	ECS Reply Data 0	R	Primeiros quatros bytes de dados da ECS Reply
0x2C	ECS Reply Data 1	R	Segundos quatros bytes de dados da ECS Reply
0x30	ECS Reply Data 2	R	Terceiros quatros bytes de dados da ECS Reply
0x34	ECS Reply Data 3	R	Quartos quatros bytes de dados da ECS Reply
0x40	ECS_RESET_SCA_0	W	Bits de Reset individual por GBT-SCA de números 31 ao 0.
0x44	ECS_RESET_SCA_1	W	Bits de Reset individual por GBT-SCA de números 63 ao 32.
0x50	ECS_DISABLE_SCA_0	R/W	Bits de desabilitação individual por GBT-SCA de números 31 ao 0.
0x54	ECS_DISABLE_SCA_1	R/W	Bits de desabilitação individual por GBT-SCA de números 31 ao 0.
0x60	ECS_TIMEOUT	R/W	Registrador com valor do tempo de expiração de comandos, em unidades múltiplas de 25,6 µs.

Tabela 3 – Mapa de registradores do SOL40-SCA.

Data 0, 1, 2 e 3; enquanto que o conteúdo dos pacotes de resposta ECS são armazenados nos registradores ECS Reply Header 0, ECS Reply Header 1 e nos ECS Reply Data 0, 1, 2 e 3. O conteúdo dos pacotes ECS são quase idênticos e mostrados na figura 47. A primeira linha, Header 0, carrega a informação do número do GBT-SCA onde o pacote é executado, o número do canal do GBT-SCA em SCA CH, além do código de operação do comando ECS. A segunda linha mostra o campo de erros de pacote chamado ECS ERR, exclusivo de pacotes de resposta, a quantidade de bytes utilizados na transmissão no campo ECS LEN e um terceiro campo, de 16 bits de largura, chamado Protocol Specific. A função do Protocol Specific é dar suporte aos chamados comandos ECS compostos, comandos que podem gerar vários comandos compatíveis com um canal do SCA em série. Atualmente só o protocolo I²C tem suporte a comandos compostos.

Os registradores ECS_RESET_SCA e ECS_DISABLE_SCA foram incluídos sob demanda de usuários, primeiro usado para reiniciar a conexão com um GBT-SCA em particular e o segundo para desativar o suporte a um dado GBT-SCA ou, sendo mais específico, a uma dada E-Link. O registrador ECS_TIMEOUT, também implementado sob demanda, provê a funcionalidade de contagem de tempo para cada pacote em execução no GBT-SCA, isto é, ele cronometra o tempo entre o envio de um comando SCA e a recepção da resposta de tal comando. Caso tal comando não responda em período de

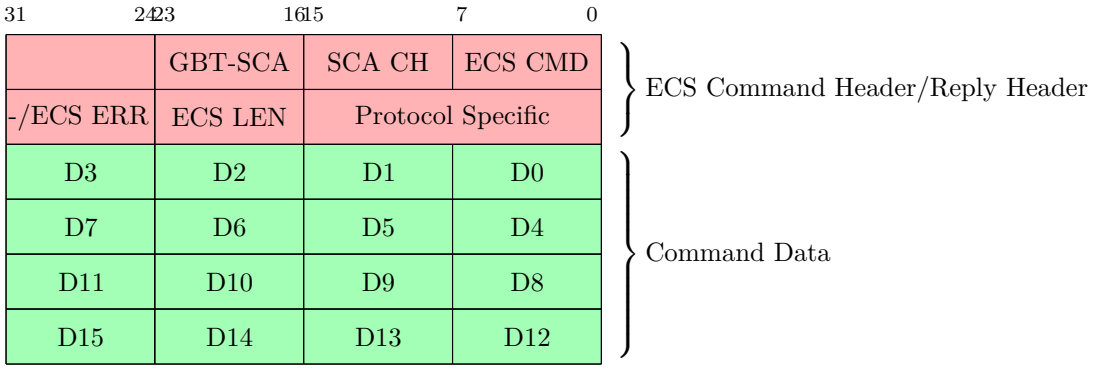


Figura 47 – Composição de um pacote ECS, tanto ECS Command ou ECS Reply.

tempo a espera pelo pacote no SOL40-SCA é interrompida, o link é dado como defeituoso e uma resposta ECS é criada imediatamente informando que o comando ECS associado não teve sucesso em sua execução.

O registrador de *Timeout* é configurado em unidades de 25,6 µs. Este valor foi escolhido devido a alguns parâmetros da arquitetura do bloco e do contexto de aplicação do SOL40-SCA. O primeiro parâmetro é a largura de dados do registrador, limitada em 16 bits, o segundo é o valor máximo que este registrador pode assumir.

É necessário que o usuário espere tempo suficiente para que uma operação de controle e monitoração seja executada com sucesso, antes que o comando seja dado como mal-sucedido. O GBT-SCA tem suporte a vários barramentos eletrônicos e a operação que apresenta o pior caso em tempo de execução são operações de transferência JTAG ou SPI com 128 bits de dados na frequência de 305 Hz de transferência. O tempo de execução estimado para uma operação deste tipo, aqui chamado de $\mathrm{RTT}_{\mathrm{max}}$ ou *Round Trip Time* máximo, é calculado na equação 4.4.

$$RTT_{max} = t_{ProcSPI128b} + t_{LocalDelay} + t_{FibersDelay} \tag{4.1}$$

$$= \frac{1}{305\,\mathrm{Hz} \times 128\,\mathrm{bit}} + 170\,\mathrm{ciclos} \times 25\,\mathrm{ns} + \frac{2 \times 100\,\mathrm{m}}{\frac{2}{3} \times 3 \times 10^{8}\,\mathrm{m\,s^{-1}}} \tag{4.2}$$

$$\approxeq 0{,}420\,\mathrm{s} + \cancel{4{,}250\,\mu\mathrm{s}} + \cancel{1\,\mu\mathrm{s}} \tag{4.3}$$

$$RTT_{max} \approxeq 0{,}420\,\mathrm{s} \tag{4.4}$$

Considerando que ECS_TIMEOUT deve compreender um tempo de espera máximo maior do que 0,420 s, com uma margem de segurança, um fator de escalonamento ou multiplicativo chamado SF deve ser calculado para realizar a transformação de cada unidade em ECS_TIMEOUT para unidades de tempo, que também pode ser entendido como o passo de tempo que cada unidade de ECS_TIMEOUT equivale. SF é expresso na

equação 4.6.

$$Timeout_{max} = (FFFF)_{16} = 2^{16} - 1 = 65535\, contagens \tag{4.5}$$

$$Timeout_{max} \times SF > RTT_{max} \tag{4.6}$$

$$65\,535 \times SF > 0{,}420\,\text{s} \tag{4.7}$$

$$SF > \frac{0{,}420\,\text{s}}{65\,535} \tag{4.8}$$

$$SF > 6{,}4\,\mu\text{s} \tag{4.9}$$

SF precisa ter um valor superior a 6,4 µs, considerando que o contador de *timeout* usará um clock fixo em 40 MHz e que operações de multiplicação ou divisão por 2 são de fácil manipulação via hardware por meio de deslocamento de bits, não necessitando de recursos de DSP num FPGA por exemplo. A expressão 4.9 pode então ser reescrita utilizando SF_WIDTH como a quantidade de bits do deslocamento em 4.10.

$$SF = 2^{\text{SF\_WIDTH}} \times \frac{1}{40\,\text{MHz}} > 6{,}4\,\mu\text{s} \tag{4.10}$$

$$\text{Considerando, por segurança, SF\_WIDTH} = 10 \tag{4.11}$$

$$SF = 2^{10} \times 25\,\text{ns} > 6{,}4\,\mu\text{s} \tag{4.12}$$

$$SF = 25{,}6\,\mu\text{s} > 6{,}4\,\mu\text{s} \tag{4.13}$$

$$\therefore \tag{4.14}$$

$$Timeout_{max} = 65\,535 \times SF = 65\,535 \times 25{,}6\,\mu\text{s} \tag{4.15}$$

$$Timeout_{max} \simeq 1{,}6\,\text{s} \tag{4.16}$$

Para fins práticos, os pacotes de dados de controle gerados ou retornados ao software SCADA serão chamados de pacotes ECS ou ECS Packets, o mesmo nome do sistema supervisório do LHCb, enquanto que os pacotes destinados ou vindos de um GBT-SCA, ainda que serializados, serão chamados de SCA, SCA Packet ou SCA Packets. Pacotes na forma de comandos, vindos do supervisório destinados ao SCA serão chamados de Command ou Commands, enquanto que pacotes gerados do SCA são chamadas de Reply ou Replies. Assim, caso um usuário deseje realizar uma operação de leitura de um sensor ligado ao SCA, por exemplo, serão gerados nesta sequência ECS Commands, SCA Commands, SCA Replies e ECS Replies.

A figura 48 mostra a sequência de operações que usuário do SOL40-SCA deve realizar quando opera em um dos barramentos eletrônicos do GBT-SCA, utilizando um diagrama de sequências UML (66). São elas:

1. Escrever o conteúdo dos dados do comando em ECS Command Header 0, ECS Command Header 1 e ECS Command DATA 0, 1, 2 e 3, conforme necessário, mostrados na seta WRITE ECS Command Registers. O usuário deve passar a

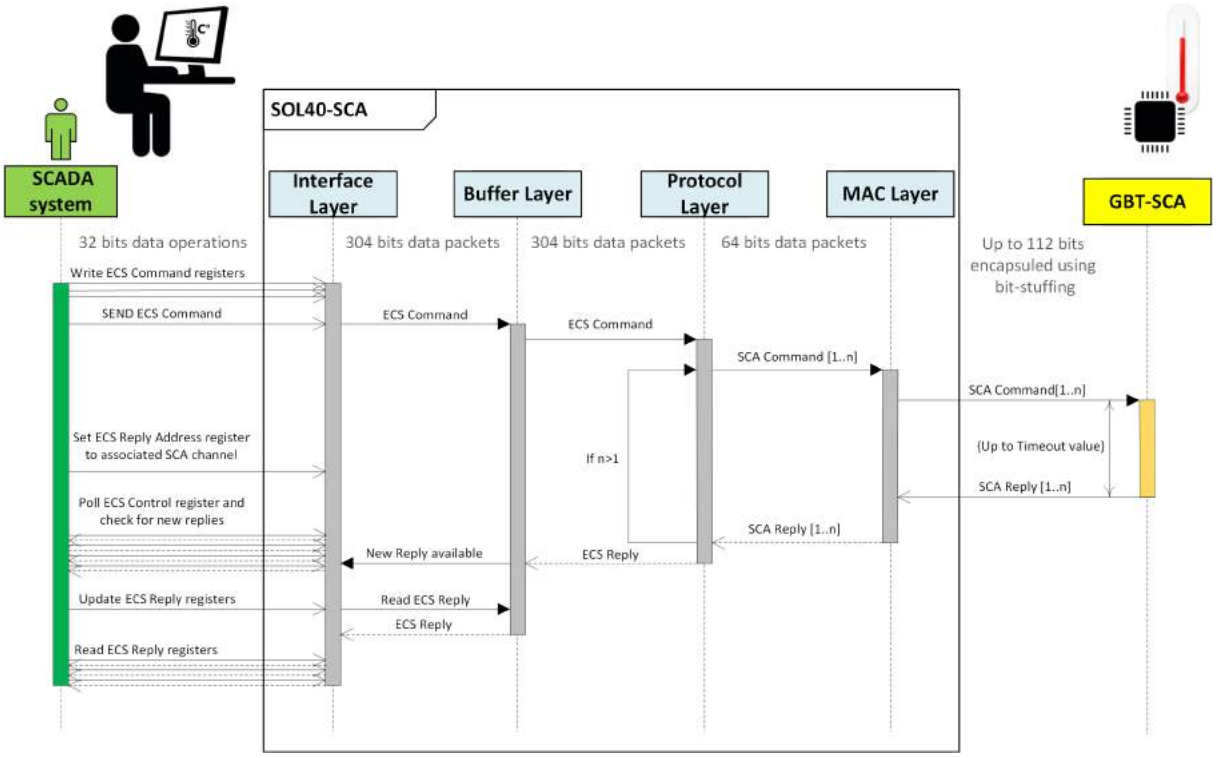


Figura 48 – Diagrama sequencial de operações do projeto.

informação de qual SCA, Canal, Código da operação e tamanho dos dados da operação além dos dados intrínsecos a mesma.

2. Comandar ao bloco do envio do comando ao chip, na seta SEND ECS Command. Etapa que corresponde ativar o bit de mesmo nome no registrador ECS Control Register.

3. Uma vez enviado o comando o usuário pode optar saber se o comando foi realizado corretamente lendo a ECS Reply associada. Esta etapa começa ajustando o registrador de endereços de Replies, chamado ECS Reply Address. Para informar sobre o estado de um dado Canal de um GBT-SCA, que pode informar posteriormente se há novos pacotes de resposta a serem tratados. Esta é associada a seta "Set ECS Reply Register".

4. O software do usuário deve realizar operações de leituras seguidas, ação chamada de *polling*, do registrador ECS Control Register a respeito do bit New ECS Reply. A mudança do valor deste bit dos valores lógicos 0 pra 1 indica que há um novo pacote de resposta, uma ECS Reply, não tratada.

5. Quando finalmente há uma nova Reply o usuário deve comandar o bloco a atualizar o conteúdo dos registradores de Reply, escrevendo um no bit Update ECS Reply no registrador de controle. Esta etapa é necessária para não interromper a leitura

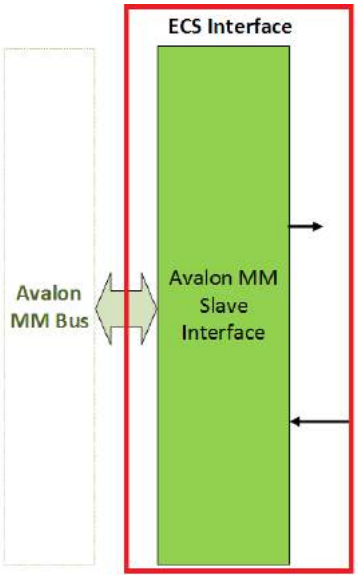


Figura 49 – Diagrama de blocos da Interface Layer

de uma Reply anterior, caso múltiplos processos de software ou usuários estejam compartilhando acesso ao bloco e facilitar a implementação de exclusão mútuas.

6. Finalmente, o usuário pode ler os registradores do pacote de resposta em ECS Reply Header 0 e 1, e ECS Reply Data 0, 1, 2, 3. Pode-se confirmar se o pacote realizado é realmente associado ao comando enviado comparando os dados de ECS Command Header e ECS Reply Header, em caso de operações de leituras, como, por exemplo, leituras de sensores I²C de até 128 bits as quais o usuário deve ler todos os registradores ECS Reply Data.

4.3.2 Interface Layer

A Interface layer é a porta de entrada que o usuário tem do SOL40-SCA. Sua principal função é prover o barramento eletrônico de dados para envio de comandos e recepção de respostas do bloco, além de controlar parâmetros do próprio bloco durante o tempo de execução.

A interface escolhida é o barramento de dados do tipo mapeável em memória chamado Avalon MM (61), criado pela Altera (23). A escolha se deve ao fato de que o projeto nasceu no firmware de TFC e SC do LHCb, que utiliza esse barramento para o envio e recepção de seus respectivos pacotes de dados. Ainda, de modo a prover uma maneira consistente de acesso ao fluxo das informações de pacotes via SCA e parâmetros internos do SOL40-SCA, uma série de registradores foi criada e são acessíveis por endereços específicos, mostrados na tabela 3.

O barramento Avalon MM, por sua vez, é derivado de um dos Registradores Bases de Endereços (*Base Address Register* - BARs), no caso BAR 0, do barramento PCIe (67)

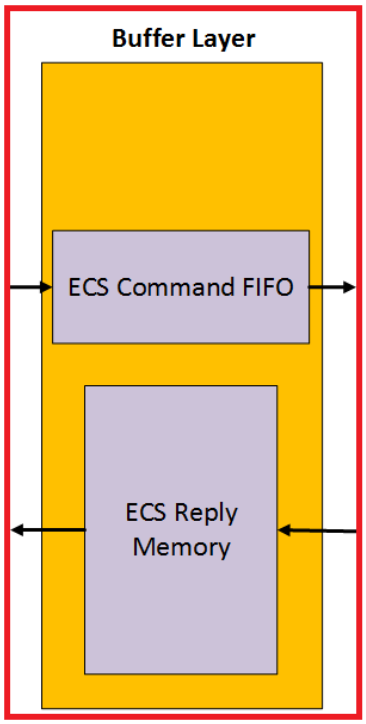


Figura 50 – Diagrama de blocos da Buffer Layer

da AMC40. Logo, uma interface *Avalon MM Slave* é usada na Interface Layer para realizar operações de leitura e escrita originadas e destinadas ao PC de controle, este na forma de um *Credit-Card PC* (CCPC) quando usado na AMC40 do MiniDAQ ou na forma de um PC comum quando usado junto da PCIe40.

4.3.3 Buffer Layer

Os comandos ECS são armazenados numa FIFO. Isto é necessário porque a frequência de clock usadas pela interface Avalon MM Slave é de 40 MHz e o tamanho da palavra de dados é fixa em 32 bits. Considerando que um comando ECS pode se estender por várias palavras de 32 bits, o comando ECS precisa ser armazenado para permitir a construção dos pacotes do tipo SCA correspondentes e transmiti-los pelos par de bits associado ao link GBT.

Uma FIFO, nestes contexto, chamada de ECS Commands FIFO, é dedicada a armazenar pacotes de comando ECS. Foi escolhido que cada link GBT terá uma ECS Command FIFO associada porque cada fluxo de dados do ECS corresponde a uma única sequência de vários comandos. No entando, eles são enviados de modo assíncrono para os GBT-SCAs e seus canais associados. Esta se torna uma maneira simples de criar um mecanismo de *back-pressure* e evitar congestionamento durante a construção e transmissão de pacotes. Isto também significa que o ECS pode enviar uma tabela de comandos de operações de escrita contínuas enquanto que o firmware lidará com a leitura e interpretação de comandos para cada canal.

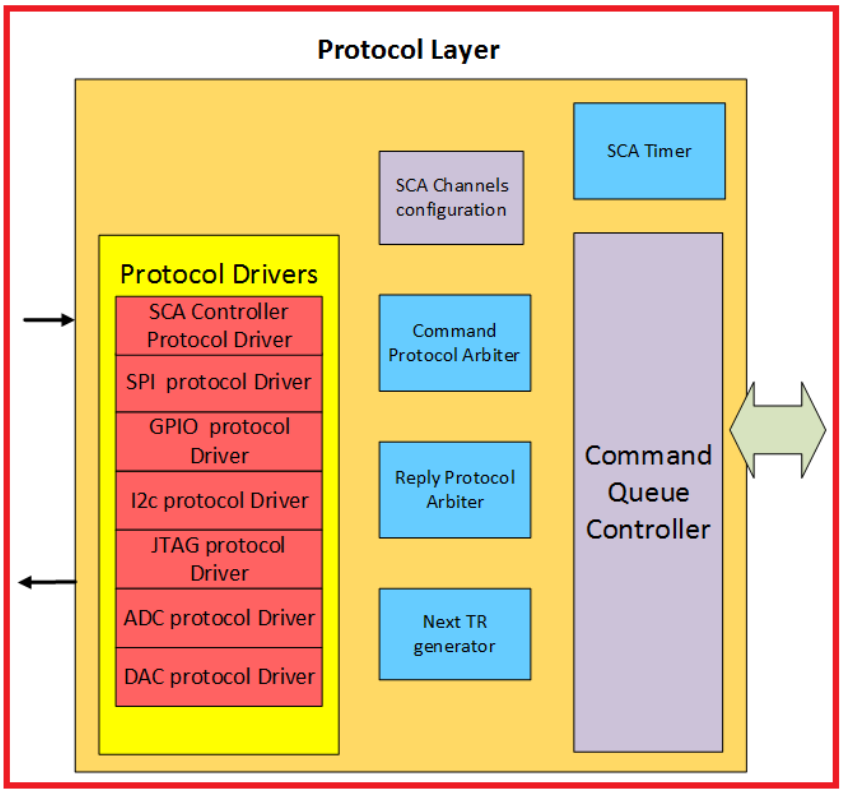


Figura 51 – Diagrama em blocos da Protocol Layer

Uma memória, chamada ECS Reply Memory, é dedicada a armazenar R, chamadas de *replies*, a um comando ECS específico. Esta foi concebida como uma estrutura RAM ao contrário de uma FIFO, para permitir que o software possa acessar a memória seguindo um mapeamento a partir do esquema de endereçamento estendido. O ECS pode então realizar *polling* esperando por uma reply específica a partir de um comando gerado previamente.

4.3.4 Protocol Layer

O chipset GBT-SCA suporta uma grande variedade de barramentos eletrônicos que podem ser interfaceados com chips da FE. Na chamada Protocol Layer, cada comando ECS é transformado ou traduzido em um ou mais comandos SCA, onde o protocolo específico para um dado canal do SCA é construído. Isto permite ao usuário a escolher de forma flexível quaisquer barramentos que quiser acessar simplesmente formatando-o no esquema mostrado na figura 47. Desta maneira o mesmo firmware pode ser utilizado para todas as combinações possíveis de barramentos na FE sem ser dependente das escolhas dos subdetectores nas FE.

Além disto, uma característica importante é que a Protocol Layer mantém informações a respeito dos comandos SCA gerados para fins de retransmissão de pacotes, gerenciamento da leitura de comandos ECS vindos da ECS Commands FIFO baseados num estado de ocupado ou não funcional (quando um SCA incorreto ou não disponível

TR	CH	LEN	CMD	DATA
8 bits	8 bits	8 bits	8 bits	0, 16 ou 32 bits

Figura 52 – Composição de um quadro HDLC do bloco FPGA E-Port

foi selecionado). Quando um pacote está pronto, este é transmitido para a MAC Layer. Isto é gerenciado por dois módulos, um dedicado a arbitragem de leitura ou escritas de comandos ECS, e outro à transmissão e recepção de comandos SCA.

Ao total, a Protocol Layer possui vários módulos assíncronos chamados de *protocol drivers*, responsáveis pela tradução dos comandos ECS em SCA correspondentes e vice-versa para replies, com detalhes nas subseções a seguir, são eles, um temporizador reconfigurável para fins de quantificação de tempo de execução de um comando no SCA, ou mesmo saber quando este não está mais disponível; duas unidades de arbitragem para multiplexação de pacotes SCA ou ECS para o *protocol driver* correspondente; além de uma unidade de buffer de pacotes chamado Command Queue Controller, responsável pela gerência de transmissão de pacotes para vários SCAs em paralelo. Todos esses módulos estão presentes na figura 51. A forma com que o Command Queue Controller gerencia cada comando ECS em execução reflete bastante as instruções de uso do projeto pelo usuário, isto pode ser na figura 53. Cada fila de comandos dentro deste bloco está em um dos estados listados. Caso a pilha esteja vazia ou já tenha sido finalizada ela estará em IDLE, para logo depois se candidatar a receber um novo grupo de comandos SCA do mesmo bloco no estado GET_SCA_CMD, vindos da Buffer Layer por um dos protocol drivers. Ao receber os novos comandos o estado da pilha ira para SEND_SCA_CMD quando a pilha disputará um SCA link da MAC Layer para transmitir um comando SCA.

Quando o comando SCA é transmitido para a MAC Layer, a pilha de comandos deve esperar o seu retorno no estado WAIT_SCA_RPY. Supondo que o comando tenha sido bem sucedido a pilha poderá retornar ao estado SEND_ECS_CMD, caso ela contenha mais comandos SCA a serem transmitidos, ou terminar por transmitir a ECS Reply, dependente de todas as SCA Replies dos comandos da pilha, para a memória de Replies na Buffer Layer.

4.3.4.1 SCA Controller protocol driver

Este módulo é responsável pela tradução dos comandos ECS para comunicação com o módulo de controle do chip GBT-SCA, chamado de SCA Controller ou Network Controller.

O SCA Controller possui internamente quatro registradores de 8 bits cada chamados CRA, CRB, CRC e CRD. A função de cada bit desses registradores é a de ativar um

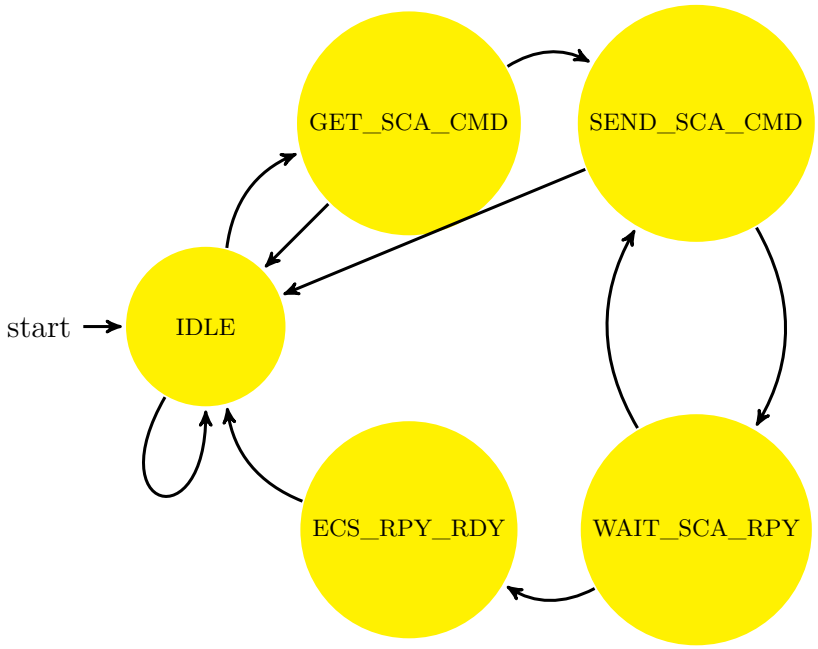


Figura 53 – Máquina de estados da cada slot no bloco Command Queue Controller

dos canais, nome dado também aos barramentos eletrônicos, que o SCA fornece. Estes bits seguem uma ordem lógica correspondente aos canais onde o mais à esquerda é o corresponde que ao canal de índice mais significativo, conforme descrito na tabela 4. A única exceção é o bit menos significativo do CRA.

Channel Name	Channel Description	Channel Number	Enable Register	Enable Channel Index
NC	SCA controller	0x00	-	-
SPI	Serial Peripheral master Interface	0x01	CRB	1
GPIO	Parallel I/O interface	0x02	CRB	2
I2C0	I2C Serial interface – master 0	0x03	CRB	3
I2C1	I2C Serial interface – master 1	0x04	CRB	4
I2C2	I2C Serial interface – master 2	0x05	CRB	5
I2C3	I2C Serial interface – master 3	0x06	CRB	6
I2C4	I2C Serial interface – master 4	0x07	CRB	7
I2C5	I2C Serial interface – master 5	0x08	CRC	0
I2C6	I2C Serial interface – master 6	0x09	CRC	1
I2C7	I2C Serial interface – master 7	0x0A	CRC	2
I2C8	I2C Serial interface – master 8	0x0B	CRC	3
I2C9	I2C Serial interface – master 9	0x0C	CRC	4
I2CA	I2C Serial interface – master 10	0x0D	CRC	5
I2CB	I2C Serial interface – master 11	0x0E	CRC	6
I2CC	I2C Serial interface – master 12	0x0F	CRC	7
I2CD	I2C Serial interface – master 13	0x10	CRD	0
I2CE	I2C Serial interface – master 14	0x11	CRD	1
I2CF	I2C Serial interface – master 15	0x12	CRD	2
JTAG	JTAG serial master interface	0x13	CRD	3
ADC	Analog to digital converter	0x14	CRD	4
DAC	Digital to analog converter	0x15	CRD	5

Tabela 4 – Tabela de canais do GBT-SCA com seus respectivos registradores de ativação, acessíveis por meio do canal SCA Controller.

Os registradores deste canal são acessíveis através dos comandos da tabela 5.

Command Name		CMD/ERR	D0	D1	D2	D3
NC_W_CRA	Command Reply	0x00 ERR	VALUE -	- -	- -	- -
NC_R_CRA	Command Reply	0x01 ERR	- VALUE	- -	- -	- -
NC_W_CRB	Command Reply	0x02 ERR	VALUE -	- -	- -	- -
NC_R_CRB	Command Reply	0x03 ERR	- VALUE	- -	- -	- -
NC_W_CRC	Command Reply	0x04 ERR	VALUE -	- -	- -	- -
NC_R_CRC	Command Reply	0x05 VALUE	- -	- -	- -	- -
NC_W_CRD	Command Reply	0x06 ERR	VALUE -	- -	- -	- -
NC_R_CRD	Command Reply	0x07 ERR	- VALUE	- -	- -	- -

Tabela 5 – Tabela de comandos do SCA Controller.

4.3.4.2 SPI protocol driver

O SCA fornece um canal para implementação do protocolo SPI (49) para comunicação com dispositivos compatíveis como sensores ou módulos de memória. As principais características do barramento SPI do SCA, totalmente suportadas pelo projeto, são:

- Velocidades de clock definidas de 305 Hz a 20 MHz.
- Suporte a até 8 dispositivos diferentes através dos pinos de SS, extensível caso os pinos do barramento GPIO sejam também utilizados.
- Buffer interno de 128 bits para transferências de dados.
- Operação nos modos de transferência de dados (0,0), (0,1), (1,0) e (1,1), permitindo sua compatibilidade com dispositivos que respeitam o padrão Motorola ou Texas Instruments.

Para poder realizar transferências SPI de um dado SCA deve-se, primeiro, assegurar que o SCA desejado esteja ativado e operante e, em seguida, ativar o canal SPI pelo NC do chip, ativando o bit 1 do registrador CRB. A tabela 6 mostra os comandos disponíveis do protocolo SPI, que são exatamente os mesmos do chip GBT-SCA com filtragem para campos inválidos pelo firmware. Uma operação SPI deve ocorrer da seguinte forma:

1. Preencher os registradores do buffer de dados através do comandos SPI_W_MOSI0, SPI_W_MOSI1, SPI_W_MOSI2 e SPI_W_MOSI3 conforme necessário. Utilizado apenas caso seja preciso a trasnferência de dados ao dispositivo ou sensor em questão.

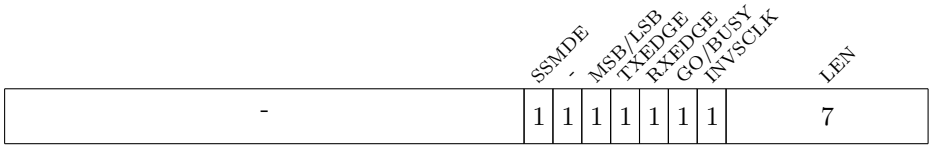


Figura 54 – Composição do registrador CONTROL do canal SPI.

2. Selecionar o dispositivo escravo no barramento SPI através do comandoSPI_W_SS.
3. Ajustar a frequência do clock através do comando SPI_W_FREQ.
4. Ajustar o tamanho da transferência em números de bits, o modo de transferência de dados, a ordem de transferência dos bits e a polaridade do sinal de clock através do comando SPI_W_CTRL.
5. Ajustar a frequência do clock através do comando SPI_W_CTRL.
6. Iniciar a transferência SPI com o comando SPI_GO.
7. Uma vez completada a transferência com sucesso, verificar o pacote de reply do SPI_GO por possíveis erros na operação.
8. Uma vez completada a transferência com sucesso, verificar o pacote de reply do SPI_GO por possíveis erros na operação.
9. Opcionalmente ler os dados de enviados pelo dispositivo SPI através dos comandos SPI_R_MISO0, SPI_R_MISO1, SPI_R_MISO2 e SPI_R_MISO3.

A forma como o registrador de ajuste de frequência, acessível pelos comandos SPI_W_FREQ e SPI_R_FREQ, é preenchendo o campo DIV. A frequência resulta da expressão matemática 4.17. Nela a frequência desejada pode assumir apenas valores discretos utilizando o clock de 40 MHz como base, onde a frequência mínima é de aproximadamente 305,2 Hz e a máxima de 20 MHz.

$$f_{TCK} = \frac{2 \times 10^7}{DIV + 1} \tag{4.17}$$

O registrador mais complexo do protocolo SPI é o de controle, acessível pelos comandos SPI_W_CTRL e SPI_R_CTRL. Seu conteúdo é definido na figura 54.

4.3.4.3 GPIO protocol driver

O canal GPIO do SCA fornece pinos digitais que podem ser utilizados para uso genérico, e são bastante flexíveis para seguir esse objetivo. Este canal possui 32 pinos independentes. As principais características do canal são:

Command Name		CMD/ERR	D3	D2	D1	D0
SPI_W_CTRL	Command	0x40	CTRL[7:0]	CTRL[15:8]		
	Reply	ERR				
SPI_R_CTRL	Command	0x41				
	Reply	ERR	CTRL[7:0]	CTRL[15:8]		
SPI_W_FREQ	Command	0x50	FREQ[7:0]	FREQ[15:8]		
	Reply	ERR				
SPI_R_FREQ	Command	0x51				
	Reply	ERR				
SPI_W_SS	Command	0x60	SS[7:0]			
	Reply	ERR				
SPI_R_SS	Command	0x61				
	Reply	ERR	SS[7:0]			
SPI_W_MOSI0	Command	0x00	D[7:0]	D[15:8]	D[23:16]	D[31:24]
	Reply	ERR				
SPI_R_MISO0	Command	0x01				
	Reply	ERR	D[7:0]	D[15:8]	D[23:16]	D[31:24]
SPI_W_MOSI1	Command	0x10	D[7:0]	D[15:8]	D[23:16]	D[31:24]
	Reply	ERR				
SPI_R_MISO1	Command	0x11				
	Reply	ERR	D[7:0]	D[15:8]	D[23:16]	D[31:24]
SPI_W_MOSI2	Command	0x20	D[7:0]	D[15:8]	D[23:16]	D[31:24]
	Reply	ERR				
SPI_R_MISO2	Command	0x21				
	Reply	ERR	D[7:0]	D[15:8]	D[23:16]	D[31:24]
SPI_W_MOSI3	Command	0x30	D[7:0]	D[15:8]	D[23:16]	D[31:24]
	Reply	ERR				
SPI_R_MISO3	Command	0x31				
	Reply	ERR	D[7:0]	D[15:8]	D[23:16]	D[31:24]
SPI_GO	Command	0x72				
	Reply	ERR	D[7:0]	D[15:8]	D[23:16]	D[31:24]

Tabela 6 – Tabela de comandos SPI protocol driver

- Cada pino pode ser configurado independentemente como sinal de entrada ou de saída (*tristates* em cada saída).
- O sinais podem ser amostrados na subida ou na descida do clock de referência.
- O clock usado para amostragem pode ser o clock do chip ou ainda um sinal externo conectado como sinal de entrada de uma das portas.
- Qualquer um dos sinais pode ser usado como gerador de interrupções do chip.

Como é possível notar em suas características, o GPIO é o único canal ou bloco capaz de gerar interrupções para seu sistema usuário, isto é, a eletrônica de Back-End, na forma de pacotes gerados de forma espontânea pelo SCA. No entanto, infelizmente, **a versão atual do projeto NÃO é preparada para tratar interrupções vindas do SCA**, logo ativá-las não faria efeito algum.

A tabela de comandos do GPIO protocol driver está disponível na tabela 7.

Command	Name	CMD/ERR	D3	D2	D1	D0
GPIO_W_DATAOUT	Command	0x10	D[7:0]	D[15:8]	D[23:16]	D[31:24]
	Reply	ERR				
GPIO_R_DATAOUT	Command	0x11				
	Reply	ERR	D[7:0]	D[15:8]	D[23:16]	D[31:24]
GPIO_R_DATAIN	Command	0x01				
	Reply	ERR	D[7:0]	D[15:8]	D[23:16]	D[31:24]
GPIO_W_DIRECTION	Command	0x20	D[7:0]	D[15:8]	D[23:16]	D[31:24]
	Reply	ERR				
GPIO_R_DIRECTION	Command	0x21				
	Reply	ERR	D[7:0]	D[15:8]	D[23:16]	D[31:24]
GPIO_W_INTENABLE	Command	0x60	D[7:0]	-	-	-
	Reply	ERR				
GPIO_R_INTENABLE	Command	0x61				
	Reply	ERR	D[7:0]	-	-	-
GPIO_W_INTSEL	Command	0x30	D[7:0]	D[15:8]	D[23:16]	D[31:24]
	Reply	ERR	-	-	-	-
GPIO_R_INTSEL	Command	0x31				
	Reply	ERR	D[7:0]	D[15:8]	D[23:16]	D[31:24]
GPIO_W_INTTRIG	Command	0x40	D[7:0]	D[15:8]	D[23:16]	D[31:24]
	Reply	ERR	-	-	-	-
GPIO_R_INTTRIG	Command	0x41	D[7:0]	D[15:8]	D[23:16]	D[31:24]
	Reply	ERR	-	-	-	-
GPIO_W_INTS	Command	0x70	D[7:0]	D[15:8]	D[23:16]	D[31:24]
	Reply	ERR	-	-	-	-
GPIO_R_INTS	Command	0x71	D[7:0]	D[15:8]	D[23:16]	D[31:24]
	Reply	ERR	-	-	-	-
GPIO_W_CLKSEL	Command	0x80	D[7:0]	D[15:8]	D[23:16]	D[31:24]
	Reply	ERR	-	-	-	-
GPIO_R_CLKSEL	Command	0x81	-	-	-	-
	Reply	ERR	D[7:0]	D[15:8]	D[23:16]	D[31:24]
GPIO_W_EDGESEL	Command	0x90	D[7:0]	D[15:8]	D[23:16]	D[31:24]
	Reply	ERR				
GPIO_R_EDGESEL	Command	0x91				
	Reply	ERR	D[7:0]	D[15:8]	D[23:16]	D[31:24]

Tabela 7 – Tabela de comandos do GPIO.

O barramento GPIO compartilha do mesmo domínio de alimentação para canais de caráter digital do chip, com tensão de alimentação fornecida pelo pino DVDD que, por sua vez, pode suportar a faixa de 1,2V a 3,3V. Dispositivos digitais que trabalham com níveis de tensão fora desta faixa podem ser usados empregando circuitos *level-shifters* apropriados.

4.3.4.4 I2C protocol driver

P barramento eletrônico mais utilizado do projeto é o I²C (49), este pode ser utilizado por projetistas de FEs utilizando quaisquer um dos 16 canais ou controladores I²C disponíveis no SCA. Assim como o canal SPI os canais I2C contém um *buffer* para dados de 128 bits. A tabela 8 mostra todos os comandos do I2C protocol driver, diretamente compatíveis com os comandos do canais I2C do SCA.

O I2C protocol driver é o único *protocol driver* que, durante o tempo da redação deste

Command	Name	CMD/ERR	D3	D2	D1	D0
I2C_W_CTRL	Command	0x30				VALUE
	Reply	ERR				
I2C _R_CTRL	Command	0x31				
	Reply	ERR				VALUE
I2C _R_STR	Command	0x11				
	Reply	ERR				VALUE
I2C_W_MSK	Command	0x21				VALUE
	Reply	ERR				
I2C_R_MSK	Command	0x21				
	Reply	ERR				VALUE
I2C_W_DATA0	Command	0x40	BYTE3	BYTE2	BYTE1	BYTE0
	Reply	ERR				
I2C_R_DATA0	Command	0X41				
	Reply	ERR	BYTE3	BYTE2	BYTE1	BYTE0
I2C_W_DATA1	Command	0x50	BYTE7	BYTE6	BYTE5	BYTE4
	Reply	ERR	-	-	-	-
I2C _R_DATA1	Command	0x51				
	Reply	ERR	BYTE7	BYTE6	BYTE5	BYTE4
I2C_W_DATA2	Command	0x60	BYTE11	BYTE10	BYTE9	BYTE8
	Reply	ERR	-	-	-	-
I2C _R_DATA2	Command	0x61				
	Reply	ERR	BYTE11	BYTE10	BYTE9	BYTE8
I2C_W_DATA3	Command	0x70	BYTE15	BYTE14	BYTE13	BYTE12
	Reply	ERR	-	-	-	-
I2C _R_DATA3	Command	0x71				
	Reply	ERR	BYTE15	BYTE14	BYTE13	BYTE12
I2C_S_7B_W	Command	0x82	-	-	DATA	ADR[6:0]
	Reply	ERR	-	-	-	STATUS
I2C _S_7B_R	Command	0x86	-	-	-	ADR[6:0]
	Reply	ERR	-	-	DATA	STATUS
I2C _S_10B_W	Command	0x8A	-	DATA	ADR[9:8]	ADR[7:0]
	Reply	ERR	-	-	-	STATUS
I2C _S_10B_R	Command	0x8E	-	-	ADR[9:8]	ADR[7:0]
	Reply	ERR	-	-	DATA	STATUS
I2C _M_7B_W	Command	0xDA	-	-	DATA	ADR[6:0]
	Reply	ERR	-	-	-	STATUS
I2C _M_7B_R	Command	0xDE	-	-	-	ADR[6:0]
	Reply	ERR	-	-	DATA	STATUS
I2C_RMW_AND	Command	0xC2	-	-	DATA	ADR[6:0]
	Reply	ERR	-	-	-	STATUS
I2C _RMW_OR	Command	0xC6	-	-	DATA	ADR[6:0]
	Reply	ERR	-	-	-	STATUS
I2C_RMW_XOR	Command	0xCA	-	-	DATA	ADR[6:0]
	Reply	ERR				STATUS

Tabela 8 – Tabela de comandos básicos do I2C.

texto, possui um grupo de comandos não diretamente associados ao GBT-SCA chamados comandos compostos ou *composite commands*. O objetivo dos comandos compostos é o de permitir ao sistema SCADA, ou qualuqer outro software de controle que usa o projeto, que este possa realizar transferências de pacotes com quantidade maior de dados que um simples comando SCA, aumentando a densidade de dados por pacote transferido entre o firmware e supervisório. No caso dos canais I2C, este permite que um comando enviado pelo usuário contenha informações de modo de endereçamento, endereço de dispositivo,

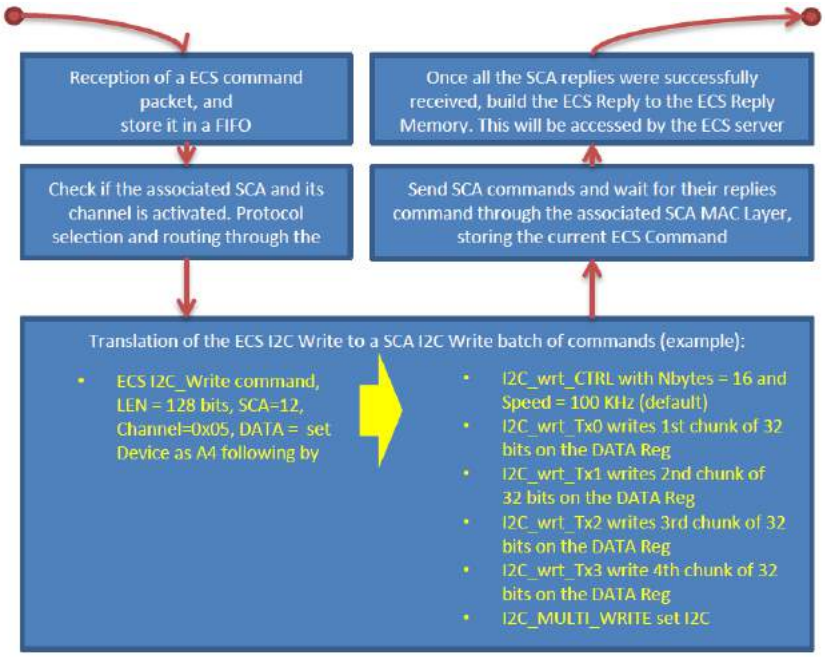


Figura 55 – Exemplo de operação de escrita I²C utilizando um comando composto

velocidade até 128 bits de dados de leitura ou escrita num único pacote ECS.

A figura 55 representa uma operação de escrita I²C de 128 bits de dados, um ECS composite command que cria 6 SCA commands, o máximo que pode ser criado atualmente.

4.3.4.5 JTAG protocol driver

O SCA apresenta um canal dedicado a funcionar como um dispositivo JTAG mestre (50) totalmente controlado via software, isto é, toda a lógica da implementação do *Test-Acess Point* e a máquina de estado do protocolo JTAG deve ser operada em software pelo usuário. O barramento JTAG, de forma geral, pode ser utilizado como canal de comunicação, depuração e programação de circuitos integrados. No contexto dos experimentos do CERN seu uso mais comum é o de programação e reprogramação de FPGAs localizados nas FEs, como seus firmwares podem ser danificados devido a efeitos de radiação na sua estrutura (68, 69).

As principais características do canal JTAG são listadas a seguir:

- Velocidades de clock definidas de 305 Hz a 20 MHz.
- Dois *buffers* independentes de 128 bits para os dados de TDI e TMS
- O *buffer* do sinal TDI é compartilhado com o sinal TDO, de forma que este sinais formam um registrador de deslocamento quando ligados a uma cadeia ou dispositivo JTAG.
- Suporte a um sinal de Reset assíncrono.

Command Name		CMD/ERR	D3	D2	D1	D0
JTAG_W_CTRL	Command	0x80	CTRL[7:0]	CTRL[15:8]	CTRL[23:16]	
	Reply	ERR				
JTAG_R_CTRL	Command	0x81	-	-	-	-
	Reply	ERR	CTRL[7:0]	CTRL[15:8]		
JTAG_W_FREQ	Command	0x90	FREQ[7:0]	FREQ[15:8]	-	-
	Reply	ERR	-	-	-	-
JTAG_R_FREQ	Command	0x91	-	-	-	-
	Reply	ERR	FREQ[7:0]	FREQ[15:8]	-	-
JTAG_W_TDO0	Command	0x00	D[7:0]	D[15:8]	D[23:16]	D[31:24]
	Reply	ERR	-	-	-	-
JTAG_R_TDI0	Command	0x01	-	-	-	-
	Reply	ERR	D[7:0]	D[15:8]	D[23:16]	D[31:24]
JTAG_W_TDO1	Command	0x10	D[7:0]	D[15:8]	D[23:16]	D[31:24]
	Reply	ERR	-	-	-	-
JTAG_R_TDI1	Command	0x11	-	-	-	-
	Reply	ERR	D[7:0]	D[15:8]	D[23:16]	D[31:24]
JTAG_W_TDO2	Command	0x20	D[7:0]	D[15:8]	D[23:16]	D[31:24]
	Reply	ERR	-	-	-	-
JTAG_R_TDI2	Command	0x21	-	-	-	-
	Reply	ERR	D[7:0]	D[15:8]	D[23:16]	D[31:24]
JTAG_W_TDO3	Command	0x30	D[7:0]	D[15:8]	D[23:16]	D[31:24]
	Reply	ERR	-	-	-	-
JTAG_R_TDI3	Command	0x31	-	-	-	-
	Reply	ERR	D[7:0]	D[15:8]	D[23:16]	D[31:24]
JTAG_W_TMS0	Command	0x40	D[7:0]	D[15:8]	D[23:16]	D[31:24]
	Reply	ERR	-	-	-	-
JTAG_R_TMS0	Command	0x41	-	-	-	-
	Reply	ERR	D[7:0]	D[15:8]	D[23:16]	D[31:24]
JTAG_W_TMS1	Command	0x50	D[7:0]	D[15:8]	D[23:16]	D[31:24]
	Reply	ERR	-	-	-	-
JTAG_R_TMS1	Command	0x51	-	-	-	-
	Reply	ERR	D[7:0]	D[15:8]	D[23:16]	D[31:24]
JTAG_W_TMS2	Command	0x60	D[7:0]	D[15:8]	D[23:16]	D[31:24]
	Reply	ERR	-	-	-	-
JTAG_R_TMS2	Command	0x61	-	-	-	-
	Reply	ERR	D[7:0]	D[15:8]	D[23:16]	D[31:24]
JTAG_W_TMS3	Command	0x70	D[7:0]	D[15:8]	D[23:16]	D[31:24]
	Reply	ERR	-	-	-	-
JTAG_R_TMS3	Command	0x71	-	-	-	-
	Reply	ERR	D[7:0]	D[15:8]	D[23:16]	D[31:24]
JTAG_GO	Command	0xA2	-	-	-	-
	Reply	ERR	-	-	-	-
JTAG_ARESET	Command	0xC0	-	-	-	-
	Reply	ERR	-	-	-	-
JTAG_GO_M	Command	0x80	-	-	-	-
	Reply	ERR	-	-	-	-

Tabela 9 – Tabela de comandos do JTAG protocol driver.

- Possibilidade de mudar as bordas de amostragens dos sinais bem como o sentido da transmissão dos bits dos *buffers* de dados internos.

O canal do JTAG protocol driver apresenta várias semelhanças no seu funcionamento com o canal SPI, porém possui o maior *buffer* de dados de todos os canais, somando 256 bits ao contrário dos 128 bits dos controladores SPI e I2C. O tamanho duplicado do *buffer* faz jus ao modo como uma operação JTAG é realizada por um controlador mestre, onde duas linhas seriais de dados são conduzidas em sincronia a um sinal de clock. Os sinais seriais em questão são os pinos TMS e TDI enquanto o clock é o sinal TCK. A tabela de comandos do canal JTAG é definida em 9.

A configuração da velocidade de operação do controlador JTAG segue o mesmo princípio do controlador SPI, é preciso usar escrever no registrador de controle de frequência, de 16 bits de largura, através do comando JTAG_W_FREQ. A fórmula 4.18 expressa como o valor da frequência é escolhido, onde a variável DIV corresponde ao valor escrito no registrador, com isso, f_{TCK} tem seus valores variando de aproximadamente 305 Hz até 20 MHz.

$$f_{TCK} = \frac{2 \times 10^7}{DIV + 1} \tag{4.18}$$

O canal JTAG é o único que necessita do dobro de escrita de operações de escrita em relação aos outros barramentos de transferências de dados, como I²C e SPI, relação intrínseca da operação do padrão JTAG por se utilizar um sinal para ajuste da máquina de estados dos seus dispositivos. Operações de 128 bits de transferências de dados para um dispositivo JTAG equivalem a 8 comandos de escrita de *buffer* do canal JTAG, 32 bits cada, além de até 3 operações extras, uma para ajuste de frequência da transmissão com o comando JTAG_W_FREQ; uma para ajuste de outros parâmetros da transmissão, como número de bits a serem amostrados, direção da transmissão serial em relação ao *buffer* interno e bordas de amostragem dos sinais através do comando JTAG_W_CTRL. O último restante, obrigatório no modo de transmissão padrão do canal, é chamado JTAG_GO, utilizado para dar início a transmissão no barramento JTAG.

4.3.4.6 ADC protocol driver

O SCA possui um canal conversor analógico-digital (ADC), o que é esperado num chip dedicado a supervisão e monitoramento de experimentos. Como o SCA reside em um ambiente submetido a consideráveis doses de radiação, um bloco IP para o ADC precisou ser especialmente desenvolvido (70, 71). O ADC do GBT-SCA foi um dos blocos que sofreram modificações no seu modo de operação durante o desenvolvimento deste projeto, as principais características do ADC das duas versão do SCA, distribuídos em protótipos de 2015 e 2016:

- Resolução espacial de 12 bits.
- 31 canais de entrada de sinais analógicos multiplexados.
- Um canal é ligado internamente no chip a um sensor de temperatura embarcado.
- Todas as entradas possuem um gerador de correntes de 100 µA chaveável, facilitando o uso de sensores de temperaturas resistivos (*Resistance Temperature Detectors* - RTD).
- Tensão para sinais analógicos de 0,0 V a 1,0 V.
- Tempo de conversão padrão de cerca de 150 µs na segunda versão do chip e 670 µs na primeira versão.
- Ganho e compensação de *offset* ajustáveis.

A lista de comandos do ADC protocol driver é mostrada na tabela 10.

Command	Name	CMD/ERR	D3	D2	D1	D0
ADC_GO	Command	0xB2				
	Reply	ERR	D[7:0]	D[12:8]		
ADC_W_INSEL	Command	0x30	D[5:0]			
	Reply	ERR				
ADC_R_INSEL	Command	0x31	D[5:0]			
	Reply	ERR				
ADC_W_CUREN	Command	0x40	D[7:0]	D[15:8]	D[23:16]	D[31:24]
	Reply	ERR				
ADC_R_CUREN	Command	0x41				
	Reply	ERR	D[7:0]	D[15:8]	D[23:16]	D[31:24]

Tabela 10 – Tabela de comandos básicos do ADC.

O conversor ADC do SCA possui uma arquitetura do tipo Wilkinson(72) de rampa simples, o que dita seu modo de operação ao usuário. De forma sintética, a medição é realizada comparando a tensão do sinal de entrada ao se converter num aumento de tensão linear gerado num capacitor de corrente contínua. Como a corrente que carrega o capacitor é fixa e bem conhecida e o valor da capacitância é estável, a tensão medida é simplesmente proporcional ao tempo necessário para carregar este capacitor. A conversão é feita ao parar um contador digital de 12 bits alimentado por um sinal de clock de 20 MHz quando o resultado das comparações entre a rampa e o sinal de entrada estão em seu máximo.

O ADC do GBT-SCA possui uma lista de comandos secundários chamados de *Advanced ADC commands* ou Comandos Avançados, mostrados na tabela 11 , eles permitem realizar ajustes finos nos parâmetros do conversor, como ajuste de *offset*, ganho, fornecer um número de identificação ao chip e adquirir o sinal de leitura entre as etapas de correções de *offset* e ganho.

As etapas de calibração inicial dos parâmetros do ADC são realizadas automaticamente durante a realização so *reset* do chip nas segunda versão do SCA, no entanto estes valores são zerados na primeira versão. A consequência principal dos valores de calibração no início da operação do chip é a de que o usuário deve configurar o ganho do ADC para o valor equivalente a 1,0 para poder ler o sinal analógico de entrada. Isto é feito escrevendo o valor de $2^{12\,bits} = (4096)_{16}$ no registrador de ganho.

Command	Name	CMD/ERR	D3	D2	D1	D0
ADC_R_DATA_OF_GA	Command	0x61				
	Reply	ERR	D[7:0]	D[15:8]	D[23:16]	D[31:24]
ADC_R_DATA_OF	Command	0xA1				
	Reply	ERR	D[7:0]	D[15:8]	D[23:16]	D[31:24]
ADC_R_DATA	Command	0x51				
	Reply	ERR	D[7:0]	D[15:8]	D[23:16]	D[31:24]
ADC_R_OFFSET	Command	0x61				
	Reply	ERR	D[7:0]	D[15:8]	D[23:16]	D[31:24]
ADC_W_CTRL	Command	0x10	D[4:0]			
	Reply	ERR				
ADC_R_CTRL	Command	0x11				
	Reply	ERR	D[4:0]			
ADC_GO_SingleSlope	Command	0x02				
	Reply	ERR				
ADC_W_GA_CAL_Reg	Command	0x70	D[7:0]	D[12:8]		
	Reply	ERR				
ADC_R_GA_CAL_Reg	Command	0x71	D[7:0]	D[12:8]		
	Reply	ERR				
ADC_W_ID	Command	0x90	D[7:0]	D[15:8]	D[23:16]	
	Reply	ERR				
ADC_R_ID	Command	0x91				
	Reply	D[7:0]	D[15:8]	D[23:16]		

Tabela 11 – Tabela de comandos avançados do ADC.

4.3.4.7 DAC protocol driver

O SCA também conta com um canal para criação de sinais analógicos, um conversor analógico digital (DAC) com 4 portas de saída. Suas principais características são listadas a seguir:

- 4 canais sinais de saída.
- Resolução espacial de 8 bits.
- Tensões de saída com valores de 0,0 V a 1,0 V.

As tabela de comandos deste protocol driver é idêntica ao do manual do SCA(73) e é mostra na tabela 12.

A operação do bloco é bastante simples, sabendo que a resolução espacial de cada conversor é de 8 bits. O valor aplicado nos registradores de dados através dos comandos

Command	Name	CMD/ERR	D3	D2	D1	D0
DAC_W_A	Command	0x10	D[7:0]	-	-	-
	Reply	ERR	-	-	-	-
DAC_R_A	Command	0x11	-	-	-	-
	Reply	ERR	D[7:0]	-	-	-
DAC_W_B	Command	0x20	D[7:0]	-	-	-
	Reply	ERR	-	-	-	-
DAC_R_B	Command	0x21	-	-	-	-
	Reply	ERR	D[7:0]	-	-	-
DAC_W_C	Command	0x30	D[7:0]	-	-	-
	Reply	ERR	-	-	-	-
DAC_R_C	Command	0x31	-	-	-	-
	Reply	ERR	D[7:0]	-	-	-
DAC_W_D	Command	0x40	D[7:0]	-	-	-
	Reply	ERR	-	-	-	-
DAC_R_D	Command	0x41	-	-	-	-
	Reply	ERR	D[7:0]	-	-	-

Tabela 12 – Tabela de comandos do DAC protocol driver.

DAC_W_A, DAC_W_B, DAC_W_C, DAC_W_D, respectivos para os canais A, B, C e D do DAC, seguem a fórmula 4.19.

$$V_{out} = \frac{(1,0) \times D[7:0]}{2^8 - 1} \tag{4.19}$$

4.3.5 MAC Layer

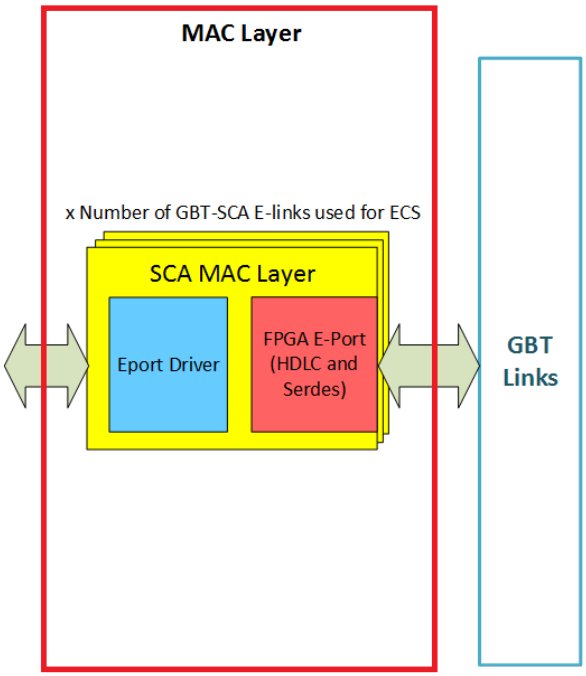


Figura 56 – Diagrama em blocos da MAC Layer.

A camada MAC Layer, derivada do conceito usado no modelo OSI (63) para protocolos de comunicação de dados chamado "Controle de Acesso ao Meio", é principalmente

responsável por encapsular o pacote de *payload* do SCA, por uma implementação da interface de comunicação Altera Atlantic (31), dentro do protocolo HDLC (29) e por serializá-lo em pares de bits, para então serem alocados em cada palavra GBT do link óptico através de interfaces do tipo e-link (27), padrão de interface elétrica criado no CERN baseado na sinalização SLVS. Além disso, a MAC Layer desserializa o fluxo de dados e extrai o *payload* quando uma réplica é recebida. Ela também suporta os comandos básicos do protocolo HDLC de reinicializar, estabelecer e testar conexão, além de detecção de erros de transmissão (35).

A MAC Layer pode ser definida como o bloco que comporta uma série de instâncias do bloco SCA MAC Layer. Este, por sua vez, é responsável pelo controle e transmissão de cada GBT-SCA associado ao projeto na forma de interfaces de comunicação E-Links. Por tratar de um E-Link cada, o bloco SCA MAC Layer é composto de uma instância do bloco FPGA E-Port pra implementar o modo de transmissão E-Port e um bloco chamado E-Port driver, que alimenta os pacotes e controla como usuário o FPGA E-Port, além de ter sua própria máquina de estados de operação, mostrada na figura 57. A máquina de estados 57 representa o estado E-Link da conexão do bloco a um GBT-SCA. O modo padrão de operação começa com o envio do comando HDLC de Reset que dá nome ao estado, seguido do teste de comunicação no estado TESTING, ativação da E-Link como porta de controle ativa do SCA em CONNECTING para então estar disponível para o usuário no estado ACTIVE, estas etapas são feitas de forma automátiva pelo firmware.

O usuário pode optar a qualquer momento mudar o estado da conexão. Isto compreende reiniciar a conexão do link, voltando para o estado de RESET de qualquer outro onde ela esteja; desativar um link ativado, onde o estado passa de ACTIVE, TESTING OU CONNECTING para DISABLED, ou, por fim, reativar uma dada conexão, indo de DISABLED para TESTING.

Um recurso adicional é a possibilidade de retransmitir um pacote se a transmissão de um comando anterior tiver sido falho. Isto é feito na MAC Layer, pois o protocolo completo, incluindo o de comunicação, já está prontos neste estágio. Um tempo de expiração programável é utilizado para aguardar a resposta dos pacotes dos SCA correspondentes e um bit programável transmitido dentro do comando ECS é usado para informar sobre a retransmissão de um pacote ou simplesmente enviar um aviso ao ECS sem retransmiti-lo. Outro recurso adicional é a capacidade de esperar pela resposta do GBT-SCA correspondente.

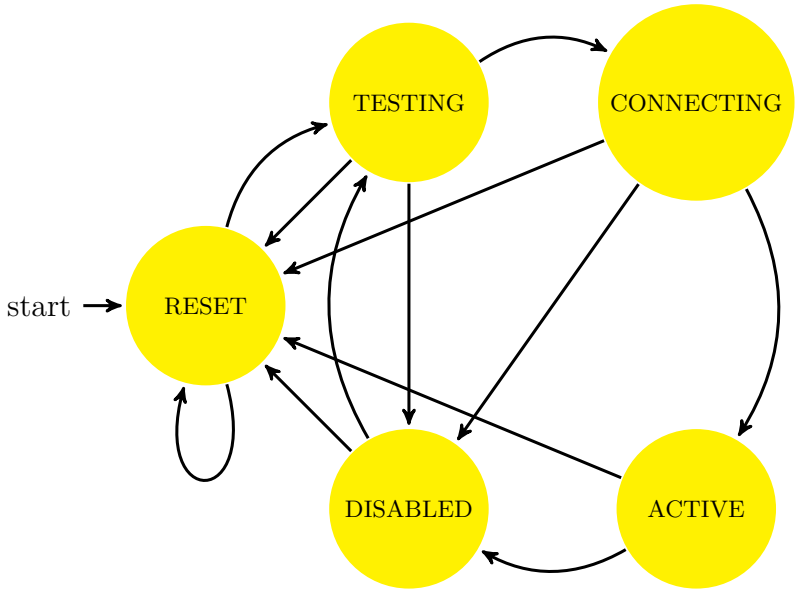


Figura 57 – Máquina de estados da MAC Layer para os estados de conexão de cada GBT-SCA.

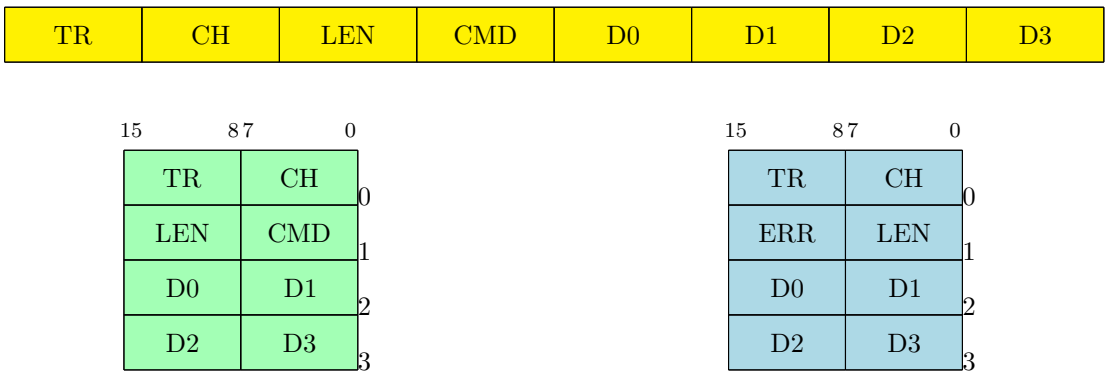


Figura 58 – Formato do quadro de dados dos E-Ports da SCA MAC Layer.

5 Histórico do projeto

Desde seu início, esse projeto pode ser definido em quatros fases ou versões de desenvolvimento onde são mantidas hoje duas versões em separado, a 1.1 funcional e estável e a 2.0, em desenvolvimento. Todo o código foi feito pelo aluno. Vale ressaltar, no entanto, que algumas etapas de desenvolvimento do trabalho foram feitas em parceria com diferentes grupos, a começar como parte dos grupos de TFC e ECS da eletrônica de upgrade do LHCb no CERN e de testes do subdetector SciFi Tracker no CBPF e em Nikhef. O projeto contou com fundamental colaboração dos grupo ESE do CERN quanto aos dados do GBT-SCA e de outros subdetectores do LHCb para realização de testes em aplicações reais. O cronograma deste trabalho é mostrado na figura 59.

5.1 Versão Preliminar 2014 - STFC-SCA

O primeiro conceito do bloco SOL40-SCA foi formulado durante o primeiro ano do projeto, em 2014, no CERN em cooperação com o supervisor do projeto Federico Alessio. Houve um processo de estudo e familiarização do aluno quanto ao funcionamento de detectores de física de altas energias, um dos motivos de sua participação na escola de

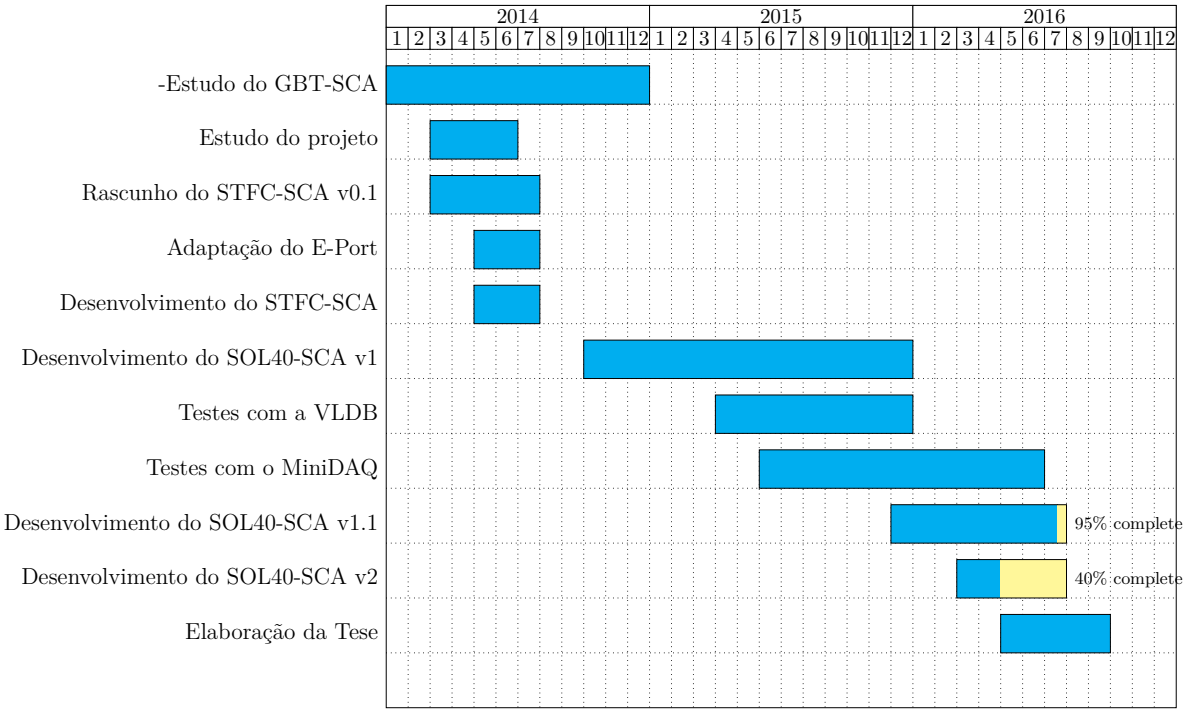


Figura 59 – Cronograma do projeto.

instrumentação do CERN em fevereiro de 2014 [1], e do detector LHCb em respeito a sua eletrônica. Havia também o interesse do grupo local de física experimental altas energias no CBPF da realização do projeto como uma das peças bases para o projeto do sistemas de testes do detector de fibras cintilantes (SciFi Tracker) (74). O foco do estudo no upgrade da eletrônica LHCb foi nas subáreas de eletrônica de controle (SC) e de temporização e sincronismo (TFC) e dos sistema de Back-End (BE), tratada de forma resumida na seção 3.

O bloco reside na eletrônica de BE como um intermediário entre os sistema supervisório do LHCb, o ECS, e a eletrônica de FE, em particular o chip GBT-SCA, tal como representado na figura 40. O ideal seria que todos esses componentes estivessem prontos, emulados ou pelo menos que sua operação externa fosse descrita precisamente em manuais ou documentos. Pensava-se que, até então, o desenvolvimento do chip GBT-SCA pelo grupo de sistemas eletrônicos para experimentos (ESE [2]) estava congelado, pois não haviam novidades ou publicações deste time desde cerca de 2012. A primeira parte sugerida seria criar uma emulação lógica do chip GBT-SCA à partir de seu manual publicado em 2012. O primeiro passo para o design da arquitetura do projeto foi definir a forma de comunicação que o bloco se comunicaria com o resto do firmware da placa SOL40 e seriam formatados para comunicação para com o bloco GBT-FPGA. Seguindo o conceito do upgrade do acelerador LHC para o LS2 se chamado de Super-LHC ou SLHC, outros sistemas eletrônicos do LHCb seguiram o mesmo conceito, como o gerador de informações de TFC ser no upgrade mudar seu nome de ODIN para S-ODIN, o novo bloco para gerenciamento de informações de SC relativas ao SCA, no caminho de dados do TFC, foi chamado de STFC-SCA.

Esta arquitetura se baseava no conceito do GBT-SCA possuir uma forma bem definida de comunicação com o usuário, onde a comunicação ocorre de forma bidirecional na forma de pacotes onde os pacotes enviados ao SCA são chamados de *SCA commands* ou commands e os pacotes de retorno são chamados de *SCA replies* ou replies. Cada comando (command) enviado ao SCA, por padrão, gera um pacote de retorno (reply).

Não havia, no início, a obrigatoriedade de a interface com a SOL40 do bloco ser do tipo Avalon MM, apenas um série de registradores ou portas customizadas. O STFC-SCA implementou um padrão de interface burst Avalon MM com suporte a Burst, pensando em facilitar a transferência em séries de dados de 32 bits de largura. Um dos requisitos desde o início foi que o bloco não poderia contar com suporte a interrupções ao ECS, esperando que este buscasse informações de pacotes de retorno por conta própria.

A figura 60 representa bem um dos primeiros designs propostos mostrando o bloco

[1] 5ª Escola Internacional de Trigger e Aquisição de Dados (ISOTDAQ 2014) - http://isotdaq2014.wigner.mta.hu/

[2] Electronics Systems for Experiments (ESE) - https://ph-dep-ese.web.cern.ch/ph-dep-ese/structure/ME.html

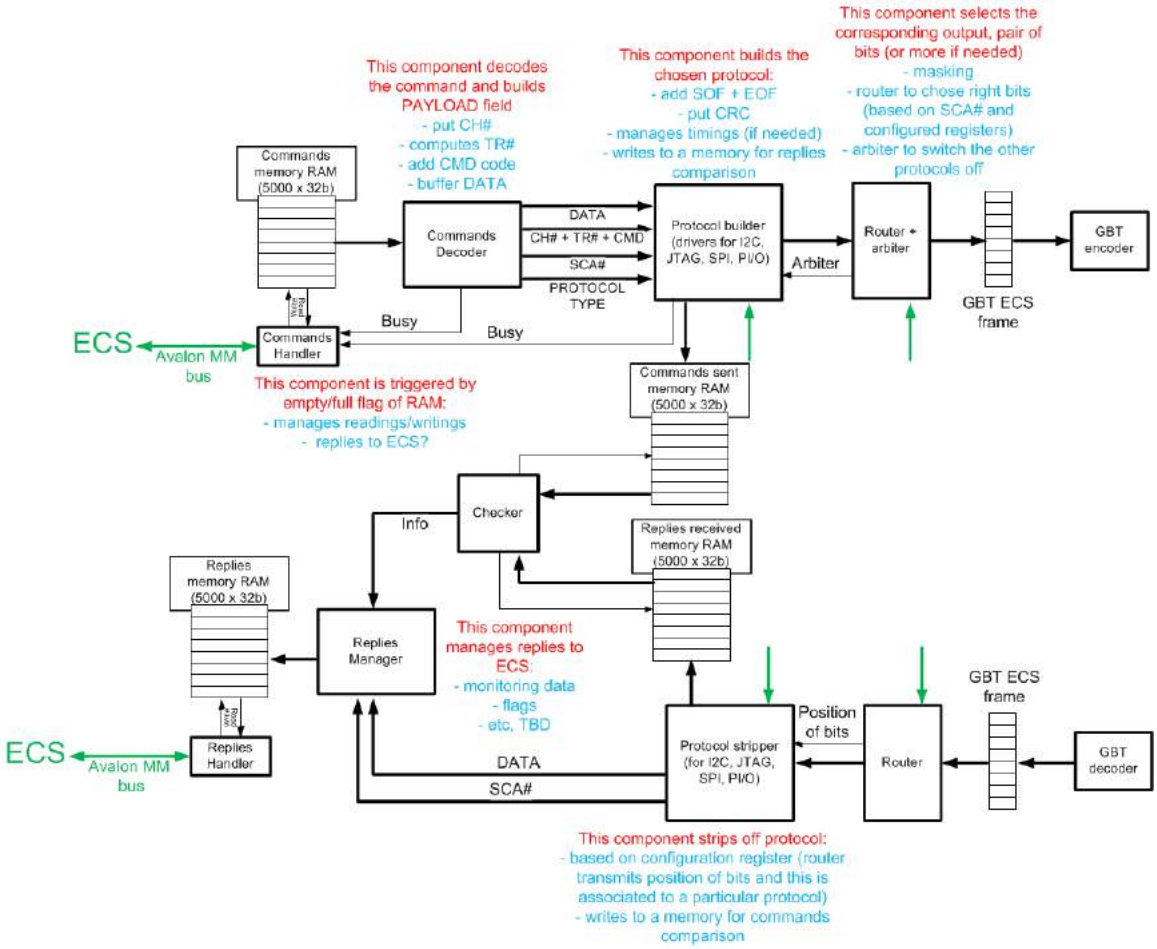


Figura 60 – Arquitetura proposta para o STFC-SCA.

como um todo, havia o conceito de paralelismo no envio e recepção de pacotes, suporte a múltiplos SCAs em posições diferentes do quadro do protocolo GBT no upgrade do LHCb (45). O desenvolvimento nessa etapa se baseava puramente em planejamento do sistema como um todo, codificação e simulação de blocos em separado, tais como testes unitários.

Até então se pensava que, antes da serialização dos pacotes SCA para serem transmitidos em pares de bits das E-links no quadro GBT, estes precisariam ser encapsulados utilizando um protocolo de comunicação tradicional, chamado HDLC (29). Isto só foi realmente notado depois da reunião do grupo de desenvolvimento do GBT-SCA, coordenado pelo engenheiro Kostas Kloukinas [3] e pelo então estudante de mestrado e novo responsável pelo desenvolvimento da parte lógica do GBT-SCA Alessandro Caratelli [4]. Nesta reunião também foi informado que o desenvolvimento de um emulador da parte lógica do GBT-SCA para FPGAs, utilizado para testar o chip antes de sua fabricação, estaria em estágio avançado, porém seu código ou uso, nesta forma, não seriam disponibilizados, somente o chip final.

[3] Kostas.Kloukinas@cern.ch
[4] Alessandro.Caratelli@cern.ch

Posteriormente, após reunião interna do aluno com o coordenador da eletrônica Ken Wyllie, o segundo entrou em contato com o desenvolvedor do bloco de implementação do HDLC no SCA, Sandro Bonacini [5] para que este pudesse disponibilizar seu código de implementação deste bloco, de forma confidencial, ao aluno, acelerando o desenvolvimento e garantindo a compatibilidade com o STFC-SCA. O bloco original se chamava E-Port e foi implementado em Verilog (75) com recursos específicos para tolerância a radiação e células instanciáveis para ASICs com tecnologia CMOS da IBM de 130 nm (76).

Para se diferenciar da implementação original para ASIC, o E-Port para FPGA seguiu o nome de FPGA E-Port, como um processo completo e pronto para ser distribuído para outros times de desenvolvimento de BE de detectores. Seu desenvolvimento durou um período de 3 meses, sendo mantido até hoje para correções de defeitos.

As mudanças fundamentais necessárias, neste período, no STFC-SCA motivaram a mudança de nome do projeto para SOL40-SCA, para reforçar a hierarquia do mesmo como interno ao firmware da SOL40.

5.2 Versão 2015 - v1.0

A escolha do nome SOL40-SCA se deu perto de agosto de 2014, para os preparativos da submissão de um artigo (77) do projeto na Conferência de Tópicos de Eletrônica Aplicada a Física de Partículas em sua edição 2014 (TWEPP 2014 [6]).

Além de ser a primeira versão do projeto que incorporou uma versão funcional do projeto FPGA E-Port, descrito em 4.2, foi também a partir desta que foi aplicado o conceito de camadas, inspirado no modelo OSI (63) da Organização Internacional para Padronização para protocolos de comunicação entre sistemas eletrônicos como redes de computadores. Elas então foram chamadas Interface Layer, Buffer Layer, Protocol Layer e MAC Layer, descritas a seguir:

A Interface Layer implementa um decodificador para a interface de comunicação do sistema ECS, a Avalon Memory-Mapped Interface (61) da Altera, foi introduzido diretamente no bloco para facilitar sua instanciação no SOL40 bem como modularizar seu uso em diferentes sistemas de BE de outros detectores.

A Buffer Layer serve como principal camada de persistência de dados de pacotes do bloco, provendo mecanismos de *Back-Pressure* conforme necessário.

A Protocol Layer basicamente traduz comandos gerados no sistema supervisório, chamados comandos ECS, em um ou mais pacotes compatíveis com o chip GBT-SCA, mantendo uma lista de comandos ECS em estado de execução. Oferece comandos de

[5] Sandro.Bonacini@cern.ch

[6] Topical Workshop on Electronics for Particle Physics - TWEPP 2014 - http://indico.cern.ch/event/299180/

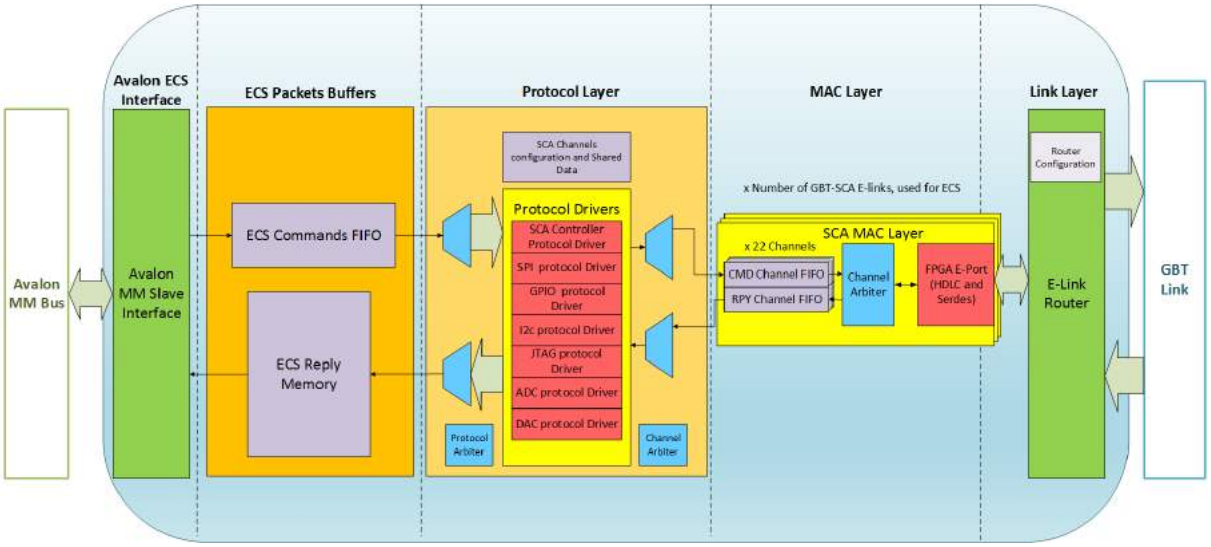


Figura 61 – Primeira arquitetura do projeto SOL40-SCA, formalizada no final de 2014. Destacando a presença da camada de roteamento de links Link Layer, FIFOs individuais de pacotes para cada SCA e suporte a vários links GBT.

transferência de dados em blocos de até 128 bits para operações I²C. Cada comando ECS, neste caso, pode ser decompor em até 6 comandos SCA.

A MAC Layer, derivada da sigla em inglês para "Controle de Acesso ao Meio", contém várias subunidades chamadas SCA MAC Layer, cada uma delas é responsável por uma E-Link e, por conseguinte, gerenciamento de conexões a cada SCA individualmente, permitindo operar vários SCAs em paralelo.

Nesta época havia uma última camada extra chamada "Link Layer", que seria responsável pelo roteamento de conexões de diferentes E-Links dos quadros GBT aos diferentes FPGA E-Ports da MAC Layer, em tempo de execução, como uma matrix de associação de 1:1. Útil caso um usuário decidisse que o par de bits da E-Link, num quadro GBT, correspondentes a certo GBT-SCA tivesse mudado de posição. Este conceito foi abandonado por ser possível que os usuários realizassem essa reassociação a nível de software e não de hardware. A arquitetura deste versão é mostrada na figura 61.

Esta foi a primeira versão que pode ser testada não apenas em simulações mas também em testes reais em FPGA, a partir desta versão o código passou a seguir mais de perto as regras de desenvolvimento do modelo de VHDL estruturado (78). Este modelo preza fortemente pelo uso de sinais estruturados na forma de pacotes ou transações ao contrário de vários sinais em separado com caminhos de dados semelhantes, o que já era prática de código desde o início do projeto. Além disso havia a restrição no número de processos em cada arquivo, um síncrono ao clock do bloco e outro para todos os eventos assíncronos.

Múltiplos *latches* indesejados foram corrigidos na implementação da Protocol Layer, o uso de *Look-Up Tables* (LUTs), ou *Adaptive Logic Modules* (ALMs) em dispositivos

Altera, foi economizado a partir da codificação correta de módulos de persistência de dados que foram mudados de registradores para memórias em blocos de memória distribuída (*Block Memory*).

Durante esta versão foram testados os protocolos I2C, SPI, GPIO, ADC e DAC. De início utilizando somente a placa GBTx SAT Board (79), placa de testes do chip GBTx, como Back-End fora da infraestrutura eletrônica do LHCb. Para facilitar o uso do bloco nesta placa, e também em outras configurações independentes onde o bloco pode ser aplicado foi desenvolvido um software chamado SOL40-SCA Evaluation Tool, explicado na parte III de Resultados deste trabalho, como uma ferramenta em software Tcl/Tk que opera dentro do ambiente da ferramenta de desenvolvimento de projetos FPGAs Quartus (54), da Altera.

5.3 Versão 2015 - v1.1

Versão atual do bloco. A versão v1.1, representada na figura 45, foi construída sobre o código da anterior porém com todos os protocolos do SCA corrigidos para operar corretamente. Excluiu o suporte a múltiplos links GBT da versão 1.0 por não ser tão intuitivo ao usuário, dada a forma que os GBT-SCA podem estar livremente distribuídos nos links GBT, e porque para a SOL40, no LHCb, cada bloco SOL40-SCA era instanciado uma única vez para cada link GBT.

Outros recursos interessantes, completando as funcionalidades do bloco, foram os registradores de Timeout, um registrador que define o máximo de tempo que um pacote deve ser enviado e retornar de um dado SCA - conceito conhecido como *Round-Trip Time* (RTT), e os registradores Disable SCA e Reset SCA, onde cada bit destes registradores é mapeado um dos dois E-Port do SCA, para desativação do link ou reset do SCA.

A versão 1.1 do projeto também foi a primeira a considerar com maior seriedade a questão dos múltiplos domínios de clocks, chamado de *Clock Domain Crossing - CDC* que podem existir derivadas dos blocos vizinhos. Isto é, a interface de dados para usuários do bloco, utilizando o barramento Avalon MM, possui um clock próprio; a interface GBT, a princípio, possui dois clocks diferentes, uma para transmissão de dados e outro para recepção de dados. A figura 62 mostra os domínios de clock das diferentes portas do SOL40-SCA, onde ECS Clock Domain é a região síncrona ao sinal ECS_CLK associado à interface ECS do tipo Avalon MM, enquanto os domínios GBT Rx Clock Domain e GBT Tx Clock Domain são associados, respectivamente, aos sinais rxClock40 e txClock40, derivados das conexões ao GBT.

Apesar dos três domínios de clocks permitidos, não houve problemas de transmissão nos dois tipos de configurações de testes onde o projeto foi empregado. A SOL40 do MiniDAQ utiliza o mesmo sinal de clock, o *bunch clock* do LHC ou BCLK, de 40 MHz,

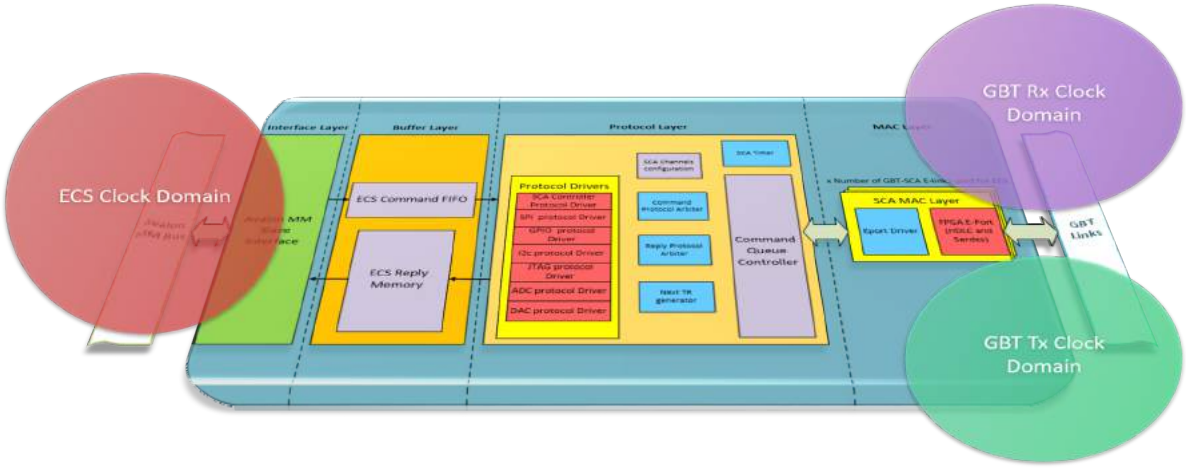


Figura 62 – Domínios de clocks do bloco SOL40-SCA.

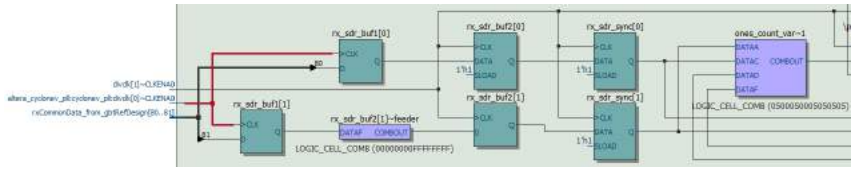


Figura 63 – 2 Flip-Flop synchronizer aplicado a recepção dos dados seriais na FPGA E-Port

no uso na interface ECS e nos caminhos de dados de transmissão e recepção do GBT-FPGA empregado na placa, nos sinais TX_FRAME_CLK e RX_FRAME_CLK, (21). A implementação do SOL40-SCA na GBTx SAT Board utiliza um bloco JTAG-to-Avalon MM Master (80) aliado a um script em TCl para enviar comandos ECS, utilizando o mesmo clock de transmissão do GBT, onde também não foram notados problemas que podem ser rastreáveis a causas originadas CDC.

Por precaução e também para facilitar o uso do bloco em configurações onde o *jitter* não é baixo ou determinístico como quando não há linhas ópticas ligando o FPGA com o SOL40-SCA ao GBT-SCA através de E-Links fisicamente, foi criada a opção de instanciar uma estrutura de sincronismo simples para transferir dados do domínio de clock de transmissão do GBT com o de recepção.

A estrutura chamada 2FF ou *2 Flip-Flop Synchronizer* (Sincronizador de dois flip-flops) (81, 65, 82), é uma forma simples usada para aumentar o tempo em que o sinal é livre de problemas de metaestabilidade, chamado tecnicamente de *Mean Time Before Failure* (MTBF), quando é transferido para outro domínio de clock. A figura 63 mostra a instância do sincronizador utilizado em cada uma das portas de recepção de dados E-Link nas FPGA-Eports do projeto, mostrada utilizando o visualizador RTL do Quartus. No entanto, o CDC entre o ECS_CLK, que pode não apenas ter fase mas também frequências diferentes em reação ao clock GBT, não pode ser tratado por uma solução como o 2FF.

5.4 Versão 2016 - v2.0

Ainda que a versão v1.1 seja funcional e estável, esta ainda não possui uma série de recursos previamente planejados, além de outros pedidos durante testes por usuários na versão atual. Alguns destes recursos são listados a seguir:

- Transferências de pacotes ECS com grande quantidade de dados para os protocolos SPI e JTAG, atualmente apenas o I²C, com seus comandos compostos, permite essa funcionalidade mas limitada a 128 bits de dados.
- Tratamento do cruzamento de domínios de clock dos dados que trafegam entre a interface ECS e a GBT.
- Suporte a interrupções do GBT-SCA, armazenando-as e tornando disponíveis ao ECS através de *polling* de seus registradores.

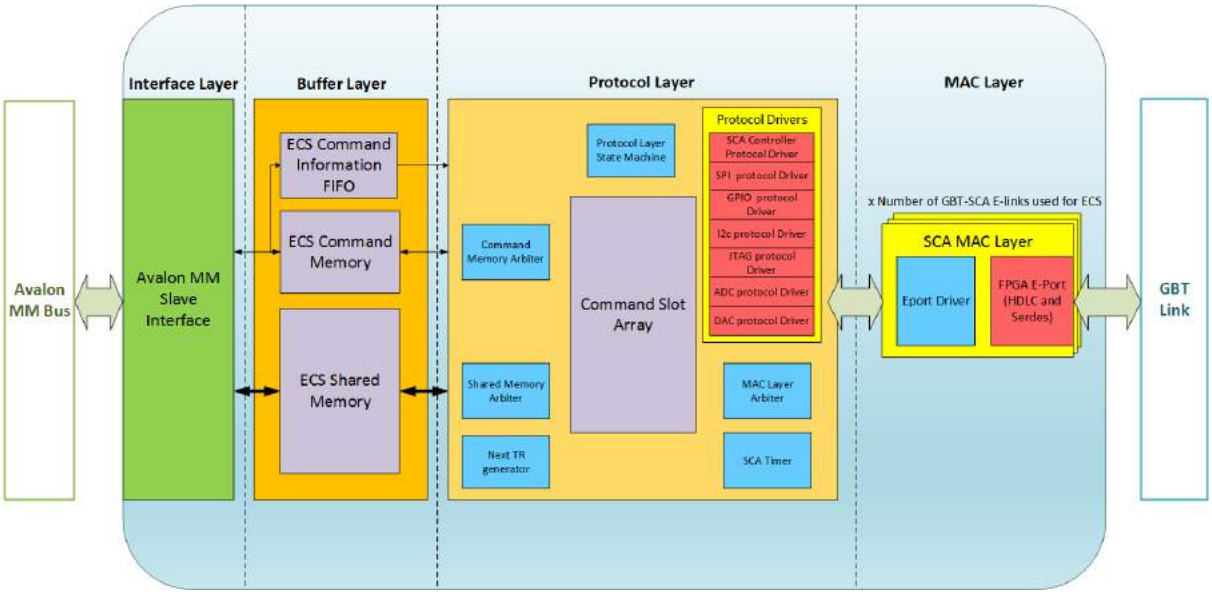
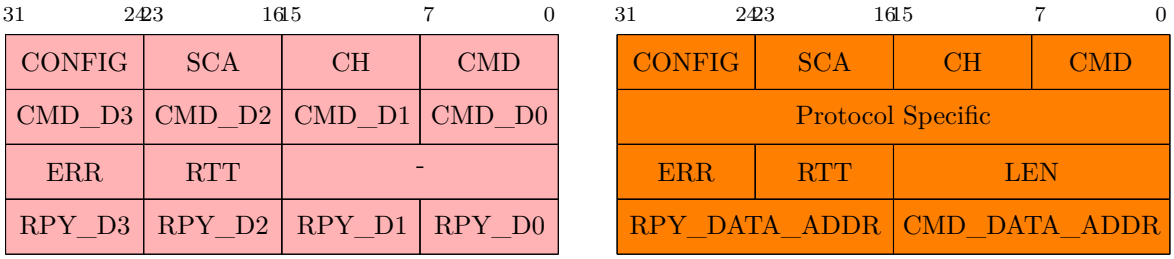


Figura 64 – Arquitetura proposta em Junho de 2016

Não obstante, requerimentos que antes não tinham urgência nas versões anteriores, que possuíam por objetivo maior servirem de protótipos, passaram a se tornar metas para esta versão. Entre eles:

- Disponibilização de uma área de armazenamento de blocos de dados ou mecanismo para uso de múltiplos barramentos do SCA, fundamental em casos de uso como a programação em paralelo de uma séries de FPGAs ou memórias na FE com o mesmo conteúdo, através dos barramentos JTAG, SPI ou I²C. *bitstream*.
- Múltiplos tipos de implementação para a interface ECS. Ainda que o barramento Avalon MM, da Altera, tenha suas especificações distribuídas livremente outros



(a) Mapa de um slot de memória para comandos simples.

(b) Mapa de um slot de memória para comandos compostos.

experimentos utilizam interconexões mapeáveis em memória do tipo AXI4 (83), da Xilinx, ou Wishbone (84), criado pela OpenCores.

A atual proposta para arquitetura da versão v2 do SOL40-SCA é mostrada na figura 64. As mudanças na arquitetura tem por objetivo permitir a transferência de grandes massas de dados para um dispositivo ligado ao GBT-SCA, diferentemente dos comandos compostos até então disponíveis no I2C protocol driver, descritos na seção 4.3.4.4, estes novos tipos de comandos compostos não podem ser limitados em 128 bits. O caso de uso visado para estes novos comandos ECS compostos é a programação de FPGAs via JTAG ou memórias via SPI localizados na eletrônica de FE de um subdetector com o mesmo conteúdo, isto é, o mesmo *bitstream*, em paralelo dado que cada FPGA da FE estaria associado a um SCA diferente. Para que tal funcionalidade seja implementada mudanças significativas na Buffer Layer e Protocol Layer foram propostas.

A Buffer Layer contém então dois blocos de memória, a primeira chamada ECS Command Memory e a segunda ECS Shared Memory, ambas de 32 bits de largura e diretamente endereçáveis através da Interface Layer. A forma como o usuário lida com comandos foi modificada em relação a versão v1.1. Todos os comandos e replies são armazenados na memória ECS Command Memory e são totalmente acessíveis ao usuário, não necessitando, por exemplo, de comandar a atualização dos registradores das Replies por exemplo. Isto reflete diretamente na região de memória utilizada pelo bloco, muito maior nessa arquitetura.

Para permitir flexibilidade e simplicidade ao lidar com os tipos de comandos simples e compostos, a estrutura da memória de comandos foi dividida em grupos chamados slots, mostrados nas figuras 65a e 65b, tal que cada slot possui 4 linhas de 32 bits da memória e pode armazenar dados de um comando. A forma como o usuário e o bloco reconhecem o tipo de comando em questão vem de um bit no campo CONFIG, comum as duas estruturas.

A diferença entre os comandos simples e os compostos está na forma com que eles armazenam os pacotes de dados, os primeiros no próprio slot de memória ECS Command Memory enquanto que os segundos terceirizam para a memória ECS Shared Memory, criada especialmente para armazenar grandes quantidades de dados e permitir compartilhamento

entre diferentes comandos ECS e, por sua vez, diferentes dispositivos ligados ao SCA. O tamanho das duas memórias é configurável pelo usuário. A FIFO chamada ECS Command FIFO tem o propósito de avisar a camada Protocol Layer da ordem dos novos comandos ECS enviados pelo usuário. Cada posição preenchida nesta FIFO aponta para um slot da ECS Command Memory com informações do novo comando.

Todas as três estruturas desta nova Buffer Layer são compostas de blocos que lidam com a questão do cruzamento de domínios de clock (CDC) dos dados síncronos ao ECS_CLK, do barramento Avalon MM na Interface Layer, e txClock40, o sinal de 40 MHz para transmissão de dados via GBT-FPGA. Isto é possível através do uso de blocos True Dual Port RAM (85) para implementação das duas memórias e de FIFOs Assíncronas (86) na FIFO de novos comandos, permitindo que um usuário, por exemplo, utilize um processador *softcore* a 100 MHz num FPGA para comandar o SOL40-SCA e um GBT-SCAs.

Parte III

Resultados

6 Metodologia

O bloco SOL40-SCA é composto de vários blocos ou componentes internos codificados em linguagem VHDL. Por se tratar de um projeto complexo, codificado em sua maior parte de forma descritiva ou comportamental, em linguagem de alto nível e de maneira genérica, metodologias mais comuns ao testes de software foram aplicadas ao contrário das tradicionais de testes de circuitos digitais.

A verificação do projeto pode ser dividida em dois tipos diferentes consistindo de três etapas. As análises foram realizadas através de simulações, descritas na seção 7, e testes em hardware, descritos na seção 8.

A validação do projeto se deu em três fases diferentes. A primeira etapa foi a simulação do firmware para FPGA do projeto, através do software ModelSim 10.3 (57) da empresa Mentor Graphics, um ambiente de simulação para código HDL. A segunda fase, de testes em hardware, foi realizada pela comunicação do bloco inserido no firmware usado no MiniDAQ e na placa GBTx SAT Board com o GBT-SCA contido no protótipo da placa VLDB. A terceira e última fase de testes foi constituída num arranjo de dispositivos usados no fluxo de dados do ECS, partindo de um computador e terminando num dispositivo ou sensor acoplado a um dos barramentos do GBT-SCA, neste último caso destacaram-se a leitura e gravação de registradores de um chip GBTx pela sua interface de programação I^2C (49) e a interface SPI (49) do ASIC Claro, que será usado na Front-End (FE) dos subdetectores RICH do LHCb. Todas etapas de validação realizadas até a data da realização deste texto foram de caráter estritamente funcional.

7 Simulações

O método de verificação mais comum é a simulação, isto é, o processo de criação do modelo de um sistema, executá-lo com padrões ou vetores testes de entrada num computador, examinando e analisando suas respostas nas saídas (87, 82), ilustrado na figura 66. O modelo pode ser o próprio circuito do sistema ou um circuito hipotético que incorpora informações de funcionalidade e temporização. Utilizar simulações permite que a operação de um sistema seja examinada em um computador para detectar erros sem de fato construir um sistema, no entanto não há garantia que os estímulos selecionados podem testar cada parte do sistema e verificar pela exatidão de todo o projeto. Enquanto simulações podem realizar checagens pontuais e detectar grandes erros num design, ela não pode garantir a ausência de erros;

Outra limitação de simulações vem da sua complexidade computacional. Operações em hardware são concorrentes e paralelas por natureza, de forma que é demorado modelar essas operações em um computador, que executa as etapas computacionais sequencialmente. Além da simulação, vários outros métodos são usados para verificação, incluindo a análise de *timing*, a verificação formal e emulação de hardware. Foram feitos testes unitários além

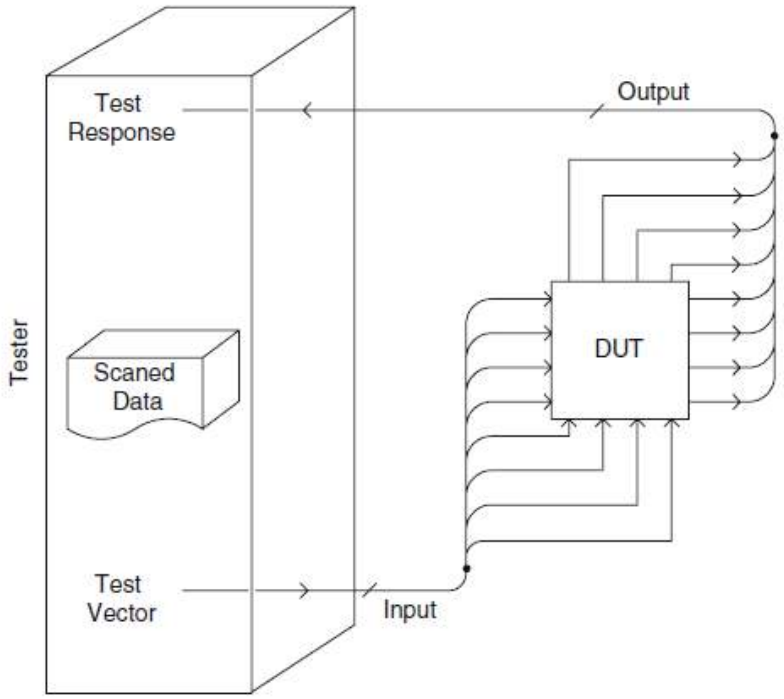


Figura 66 – Testador (*tester*) alimentando um Projeto sob Testes (*Design Under Test*) - DUT, elementos básicos de uma simulação.

de testes funcionais para os principais componentes do bloco FPGA E-Port e do próprio SOL40-SCA.

7.1 Simulação do bloco CRC

O primeiro bloco a ser simulado foi a adaptação do bloco E-Port original (**??**27), parte fundamental para a implementação do protocolo de comunicação com o GBT-SCA, onde este foi traduzido da linguagem de HDL Verilog para VHDL como exigência do LHCb, e eliminadas alguns recursos desnecessários para a implementação em FPGA na BE como iterados na seção sobre a MAC Layer 4.3.5.

A adaptação do bloco E-Port para o FPGA E-Port removeu as necessidades de implementação da redundância modular tripla (TMR) (53) pelo bloco não ser sujeito a radiação ionizante. Isto removeu, por conseguinte, o bloco do algoritmo de Hamming (34), responsável por verificar a integridade dos dados nas saídas das fifos do FPGA E-Port. A única unidade com maior nível de complexidade foi o bloco de CRC (35), parte do próprio padrão HDLC. Logo um dos blocos a serem simulados foi o CRC.

O CRC implementado pelo bloco E-Port segue o padrão do HDLC de CRC de 16 bits chamado CRC-CCITT, este realiza operações cíclicas em dados de 1 byte cada utilizando o polinômio $x^{16} + x^{12} + x^5 + 1$, representado pelo valor $(1021)_{16}$ em base hexadecimal, e valor inicial $(ffff)_{16}$. Para garantir a exatidão do bloco, foi utilizado o bloco CRC original do bloco E-Port, em Verilog, em paralelo com a nova implementação em VHDL e um exemplo oficial do CRC-CCITT, retirado de um código em C de uma apêndice do protocolo PPP (88).

O arquivo de simulação do CRC, chamado "crc-tb.vhd", está disponível nas apêndices em B.2, é responsável por instanciar o CRC do bloco FPGA E-Port, em B.1, e o CRC do E-Port, não disponível para distribuição. O arquivo cria os vetores de dados teste para alimentar os blocos e os grava num arquivo chamado "hw_csv_output.csv", onde cada linha contém os dados de entrada dos blocos, o resultado do CRC em VHDL seguido da versão em Verilog. A partir desse arquivo os arquivos são passados a um programa em C chamado "crc-hdlc.c", disponível em B.2, para comparação dos resultados com a implementação em C do protocolo PPP.

Os testes utilizam um vetor de dados aleatório, gerado a cada execução da simulação, com várias larguras de dados diferentes na de 1 a 10 bytes de dimensão, tamanho máximo utilizado nos pacotes encapsulados no GBT-SCA (73). O formato das ondas utilizadas na simulação é representado na figura 67, onde foi utilizado um sinal de clock de 40 MHz, o sinal d_valid como indicador de dados válidos na entrada d para cálculo do CRC. Os sinais crc_o e crc_o_vlog indicam respectivamente os valores do CRCs calculados pelo bloco CRC em VHDL e o bloco CRC em Verilog do E-Port original. A saída é atualizada

DATA_SIZE	DATA_COUNT	VLOG_MISMATCH	SW_MISMATCH
1	1000	0	64
2	1000	0	188
3	1000	0	147
4	1000	0	137
5	1000	0	151
6	1000	0	179
7	1000	0	128
8	1000	0	163
9	1000	0	138
10	1000	0	158

Tabela 13 – Sumário dos testes do bloco CRC

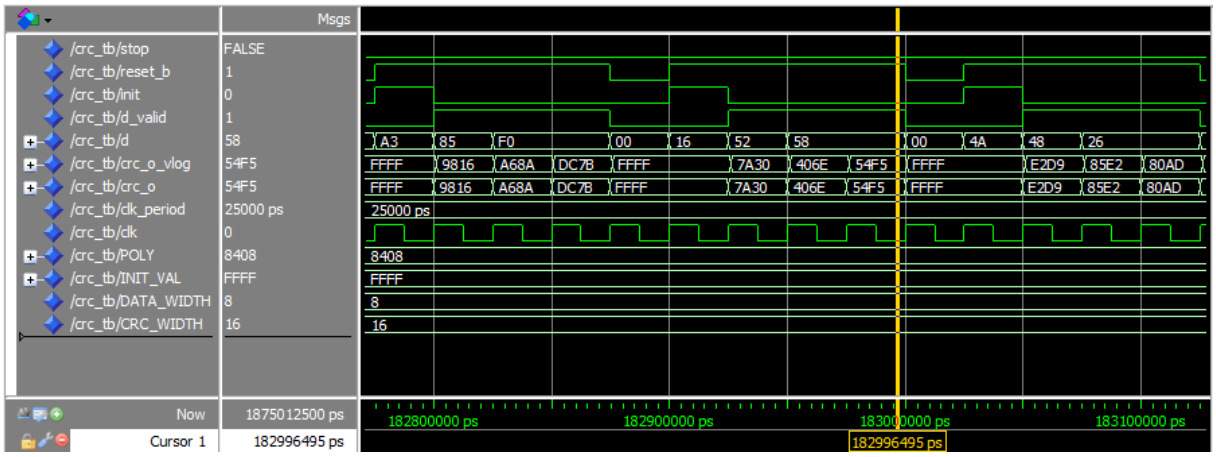


Figura 67 – Simulação do circuito de CRC.

de forma assíncrona, se atualizando diretamente com o valor dos dados de entrada.

O resumo dos resultados é representado na tabela 13, onde a coluna DATA_SIZE representa o tamanho dos dados testados em bytes, a coluna DATA_COUNT a quantidade de dados criados de forma aleatória, VLOG_MISMATCH representa o número de inconsistências entre o CRC em VHDL e o CRC em Verilog, e SW_MISMATCH o número entre o CRC em VHDL e o CRC em C do protocolo PPP utilizado como referência.

A análise dos resultados mostra que não houve inconsistências entre a implementação do CRC entre os projetos E-Port e FPGA E-Port original, mas se foi constatada uma média de 14,53% de inconsistências com o CRC do HDLC implementado no padrão PPP, o que gera preocupação quanto a exatidão deste algoritmo no projeto. No entanto, esta diferença não afeta o funcionamento do sistema, pois apenas as versões em VHDL e Verilog trabalham em conjunto na transmissão de pacotes SCA.

7.2 Simulação do bloco FPGA E-Port

A parte mais crítica do projeto SOL40-SCA para seu funcionamento e comunicação com o GBT-SCA é o bloco FPGA E-Port. Por se tratar de um projeto completo por

si próprio o bloco FPGA E-Port apresenta um nível de complexidade equivalente a todo o restante SOL40-SCA junto. Um novo bloco precisou ser projetado, baseado na implementação do E-Port original em Verilog para ASIC com lógica extra para tolerância a radiação e TMR. Este precisou ser escrito com VHDL, requisito do LHCb, de forma genérica para poder ser utilizado em qualquer FPGA.

Todas as operações do FPGA E-Port para comunicação com o GBT-SCA foram simuladas, o que inclui todo o procedimento de conexão, mostrado na figura 68, e transmissão de dados entre dois blocos FPGA E-Port assim como entre FPGA E-Port e o IP core do E-Port original.

A figura 68 ilustra a etapa de conexão HDLC, mostrando sinais de transmissão de uma E-Port mestre e recebidos num E-Port escravo, demarcados pela linha elink_s dando início às instâncias de sinais da porta escrava. O sinal de entrada tx_cmd_sabm, de SABM (*Set Asynchronous Balanced Mode*), indica uma demanda para que o E-Port mestre inicie a etapa de conexão com a porta escrava. Este comando é transformado num pacote em phy_tx_data, serializado e em seguida desserializado em phy_rx_data , já no E-Port escravo. Ao interpretar corretamente o pacote SABM, o E-Port escravo transmite um pulso no sinal de saída tx_cmd_ua, indicando que esta enviará um pacote de resposta do tipo UA (*Unnumbered Acknowledgment*), definido o sucesso da conexão entre as duas interfaces.

As Figuras 69a e 69b mostram a transmissão e recepção de uma cadeia de 64 bits de dados entre dois blocos FPGA E-Port, um com o papel de mestre e o outro como escravo. Esta simulação representa o tamanho máximo do *payload* do pacote de transmissão de dados usados em comunicações com o GBT-SCA, onde este *payload* é encapsulado no protocolo HDLC e serializado em 2 bits com *bit-stuffing* antes de serem transmitidos de fato. Foram feitas simulações funcionais de integridade dos pacotes de todos os outros blocos do firmware SOL40-SCA, porém não foi possível realizar uma simulação dos pacotes com o firmware do GBT-SCA, pois este não foi disponibilizado, sendo possível seu uso apenas com o hardware real na etapa de testes.

7.3 Simulação do SOL40-SCA

Durante o primeiro ano e meio do projeto, a única forma de verificá-lo depurá-lo foi por meio de simulações lógicas. Quase todos os blocos do SOL40-SCA, em certo ponto, tiveram arquivos de simulação associados para a realização de testes unitários. Esta etapa foi bastante útil, para não só garantir o funcionamento de cada bloco individualmente, como também permitiu a simulação de testes de integridade do projeto como um todo.

Durante o segundo ano do projeto, quando os equipamentos de hardware e os GBT-SCAs já estavam disponíveis, as simulações de testes de integridade foram aos poucos

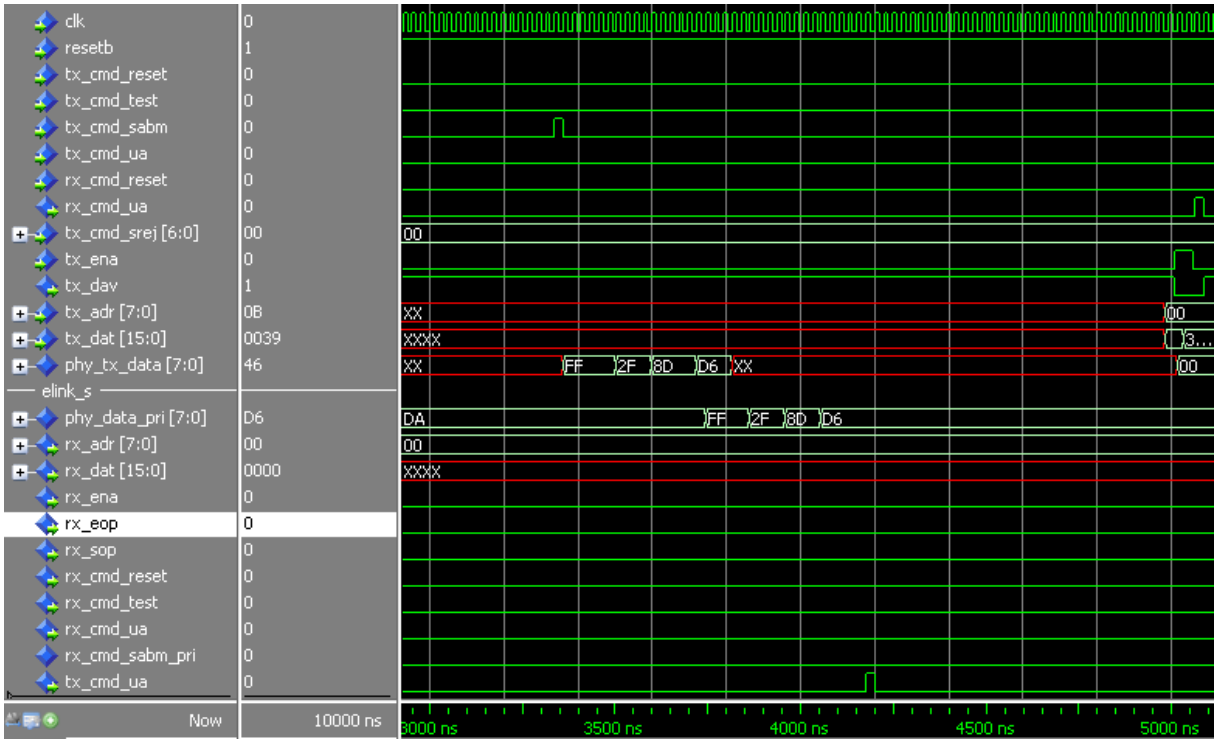
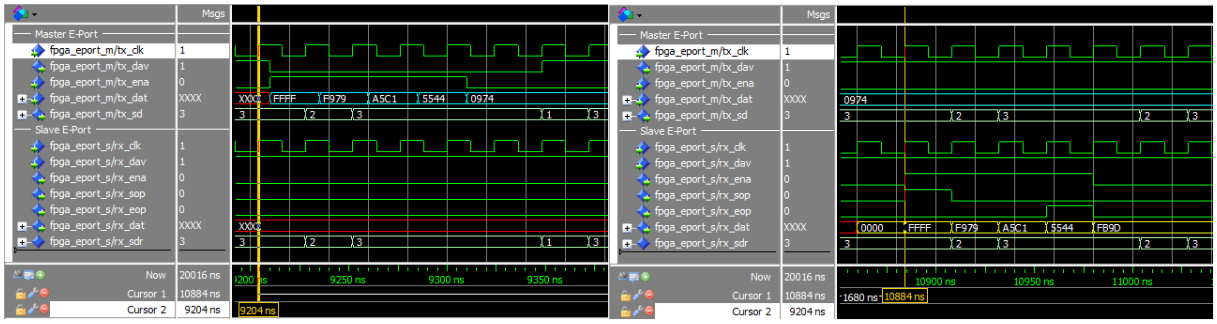


Figura 68 – Simulação do comando de conexão (SABM) do FPGA Eport



(a) (b)

Figura 69 – Simulação lógica da transmissão de dados entre dois FPGA E-Ports.

abandonadas, devido ao elevado grau de complexidade na preparação de estímulos e verificadores para os sinais de entrada e saída do projeto. Esta questão foi considerada na arquitetura da versão v2 do SOL40-SCA, onde os Protocol Drivers passaram a ser implementados como funções em pacotes ou bibliotecas, ao contrário de processos assíncronos em entidades VHDL.

Estes métodos permitem que o desenvolvedor possa criar ambientes de simulação de forma mais eficaz, além do emprego de um ambiente para verificação em código aberto em VHDL chamado OSVVM (89) já empregado como gerador de dados aleatórios na simulação do CRC.

8 Testes em Hardware

Os Testes em Hardware são a última etapa necessária para a entrega de um projeto de sistema digital. De forma mais concisa, a etapa também chamada de *In-System Verification*, garante que o projeto quando implementado no chip se comporta de maneira compatível com seus modelos simulados anteriormente. Diferentes tipos de verificações podem ser feitas, tais quais na etapa de simulação. Neste trabalho os métodos de verificação utilizados em hardware para o projeto foram prioritariamente de testes funcionais e de integridade.

É preciso deixar claro que o objetivo deste trabalho não foi de caracterizar o chip GBT-SCA, sua operação ou características elétricas de suas interfaces, mas de realizar testes de unidade, funcionais, de integridade e tolerância a erros do firmware do projeto SOL40-SCA. No entanto, como o SOL40-SCA está intrinsecamente ligado a operação do GBT-SCA e este trabalho serve como precursor de testes das interfaces de controle em vários *setups* de eletrônica de FE, foram tomados gráficos e testes de funcionamento destes três elementos da cadeia.

8.1 Equipamentos utilizados

Para a realização dos testes, foram preparados dois arranjos de equipamentos diferentes, cada um deles têm em comum versões dos primeiros protótipos da placa VLDB (Versatile Link Demonstrator Board) (90), parte do programa Versatile Link (91), que contém basicamente um chip GBTx, a primeira versão do GBT-SCA, módulos de comunicação óptica bidirecional VTRx (62, 92) e portas E-Link na forma de conectores HDMI. Todos estes elementos desenvolvidos de forma customizada pelo projeto Radiation Hard Optical Link (15) pelo CERN, que também inclui o sistema GBT, para a emulação de elementos comuns em FE dos futuros upgrades dos experimentos do LHC.

A figura 70 mostra todos os elementos utilizados nos dois diferentes arranjos do *setup* de testes do projeto, montado em laboratório do CERN. Os equipamentos estão numerados de 1 a 4 onde 1 mostra o MiniDAQ, 2 a GBTx SAT Board, 3 - A VLDB principal e 4 - a VLDB secundária.

Foram utilizadas duas placas VLDB para os testes, uma principal e outra secundária. Na VLDB principal, foi utilizado o GBTx como interface de rede para os dados de *Slow Control* e um GBT-SCA para a criação de barramentos eletrônicos genéricos de controle e monitoração, a partir dos sinais da interface E-Link gerados no GBTx. A VLDB secundária teve o propósito de servir como dispositivo I^2C escravo para testes do GBT-SCA na VLDB

principal, no caso o próprio GBTx desta foi utilizado como chip I²C. As subseções a seguir apresentam alguns detalhes dos principais hardwares utilizados nos testes.

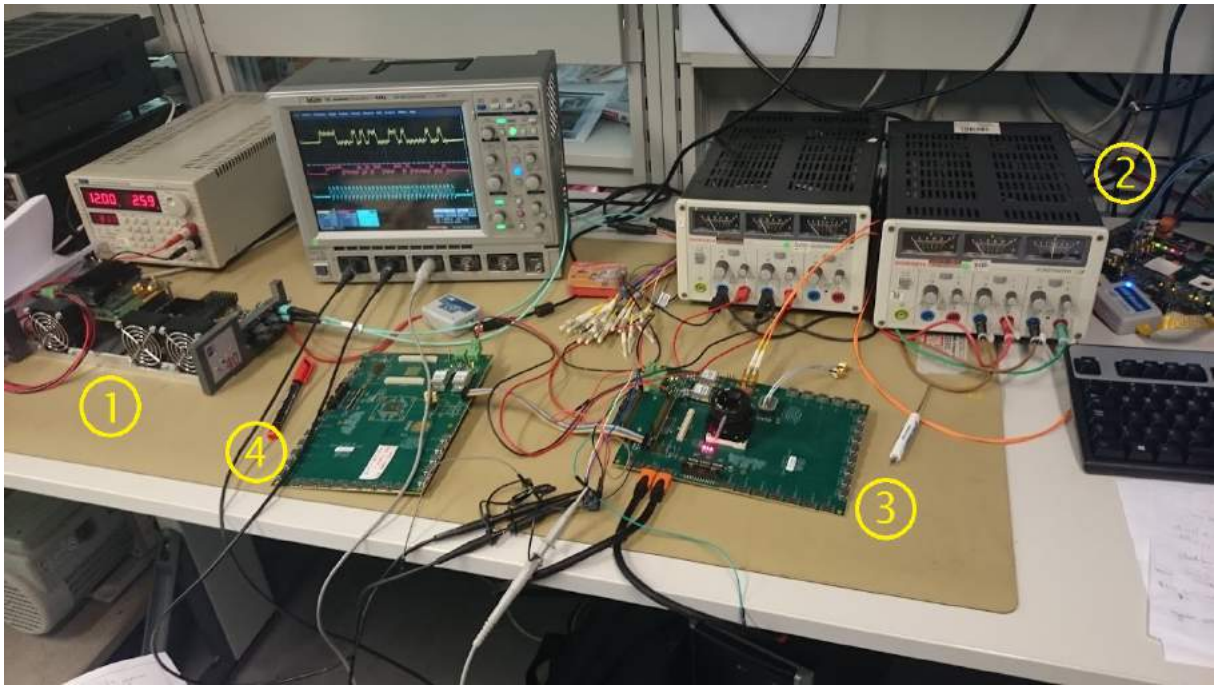


Figura 70 – Configuração dos equipamentos usada para os testes do bloco SOL40-SCA

8.1.1 VLDB

A placa de demonstração do projeto Versatile Link (The Versatile Link Demo Board - VLDB)(90), representada na figura 71, foi desenvolvida pelo grupo do CERN/PH/ESE como forma de um kit eletrônico de avaliação do ecossistema do projeto Versatile Link. Seu objetivo é o de prover, numa mesma placa, os quatro tipos de componentes eletrônicos resistentes à radiação desenvolvidos no CERN e comuns a todos os experimentos do LHC, sendo eles o GBTx(37), SCA(20), VTRx(62) e o FEASTMP(93).

Com a VLDB, projetistas de componentes eletrônicos dos experimentos do LHC podem, por exemplo realizar testes de tolerância de radiação num sistema completo de FE, independente quanto ao tipo de detectores ou sensores usados; familiarizar com os componentes do projeto Versatile Link e testar configurações diferentes quando a conectividade de sistemas de FE ao GBTx utilizando E-Links através de conectores HDMI.

As VLDBs podem ser disponibilizadas em kits contendo placas E-Links FMCs, adaptadores USB-I2C e módulos I/O para conectidade com o SCA. O primeiro lote de placas VLDB foi disponibilizado na forma de protótipos durante 2015, utilizando a primeira versão do GBT-SCA. Este primeiro protótipo apresentou um erro de design onde as conexões do GBT-SCA estavam invertidas, como mostrado na figura 72; ainda deste modo foi possível ligar os pinos de canais I²C, SPI e GPIO.

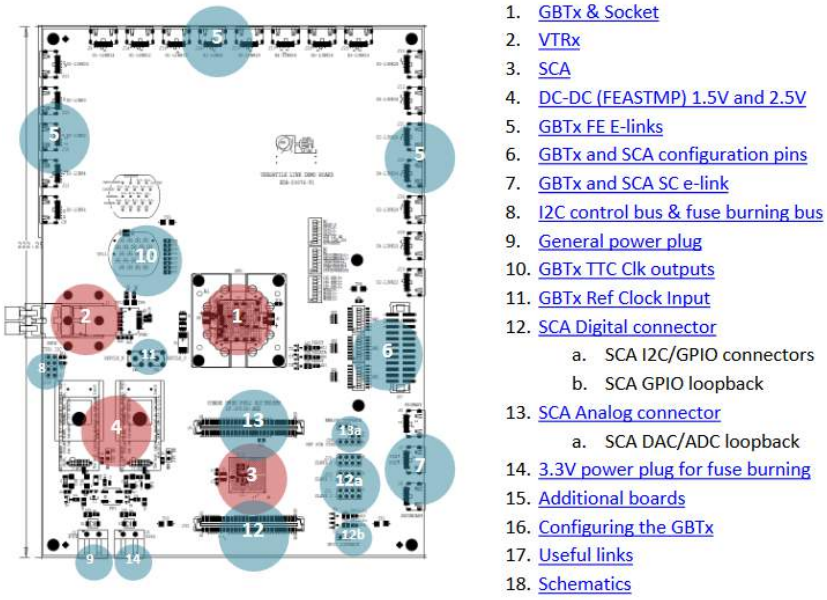


Figura 71 – Diagrama de componentes da VLDB.

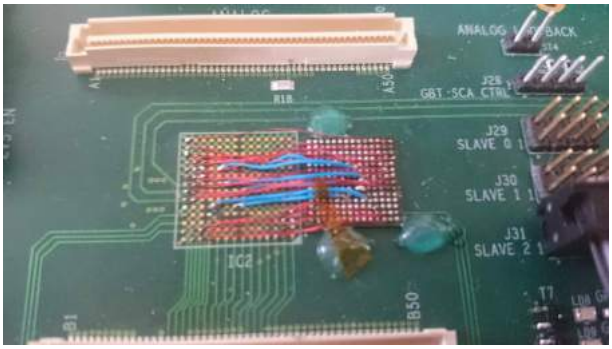


Figura 72 – Conexão do GBT-SCA ao protótipo da VLDB nos testes de 2015.

Os testes de 2016 utilizaram a primeira versão pronta da VLDB, ainda com o primeiro chip GBT-SCA. Nela foi possível utilizar todos os canais disponíveis do chip.

8.1.2 GBTx SAT

A placa GBTx SAT Board (79), sigla para *Stand-Alone Test Board* e mostrada na figura 73, foi desenvolvida no CERN para hospedar um chip GBTx, dois soquetes SFP+ para conexões de fibras ópticas compatíveis com o VTRx, uma conexão Ethernet e um FPGA Cyclone V da Altera (23). Seu propósito foi de servir de placa de avaliação e testes do GBTx quanto ao desenvolvimento, caracterização de transmissão de dados e testes de resistência à radiação.

O GBTx da placa tem seus parâmetros e configurações ajustados por uma ferramenta em interface gráfica desenvolvida em Java que se comunica com a placa pelo PC via Ethernet. O FPGA contém um firmware que serve de contrapartida para uso do GBTx, contendo uma instância do projeto GBT-FPGA (21), além de permitir o uso de E-Links

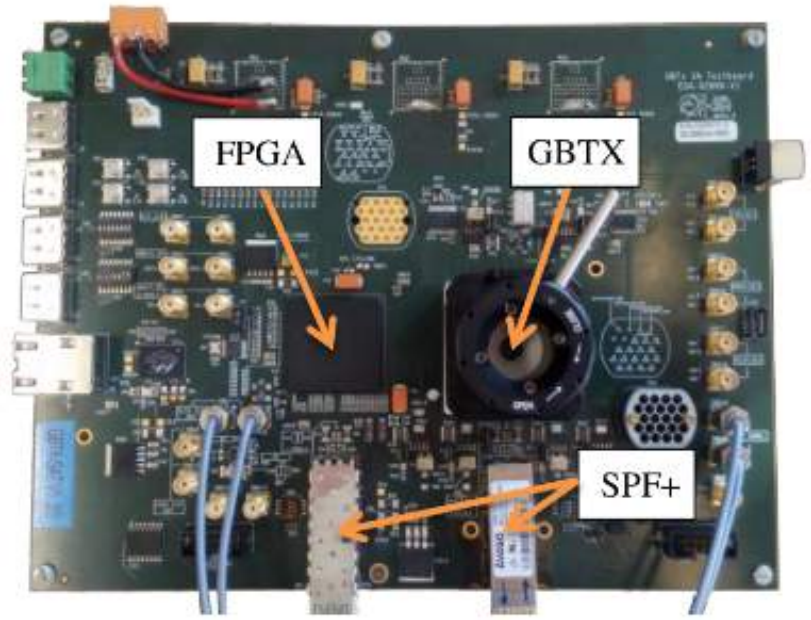


Figura 73 – Placa de testes do GBTx SAT Board.

externas e código de terceiros no FPGA.

A GBT-SAT Board foi utilizada na primeira configuração de testes do GBT-SCa, onde teve o papel de placa de BE, enquanto um dos protótipos da VLDB, contendo um GBTx e o GBT-SCA da figura 72, fez o papel de eletrônica de FE.

8.1.3 MiniDAQ

O MiniDAQ (51) é o primeiro kit de protótipo do LHCb para a eletrônica de BE no próximo upgrade do experimento (42), que ocorrerá durante o LS2. Este kit, mostrado na figura 74, compreende os seguintes componentes:

- 1 Placa AMC40
- 1 Placa AMC_TP(Test Pad), equipada com *Credit Card PC*(CCPC).
- 3 kits com 12 Fibras ópticas.
- 12 Adaptadores LC/LC.
- Cabo de programação USB Blaster (94).
- 1 Placa de rede 10 GbE com interface óptica.
- 1 Placa de rede GbE padrão.

Neste caso, o kit compreende tanto a eletrônica de BE, na placa AMC40 num FPGA Stratix V da Altera, quanto o software supervisório, que opera em módulos no

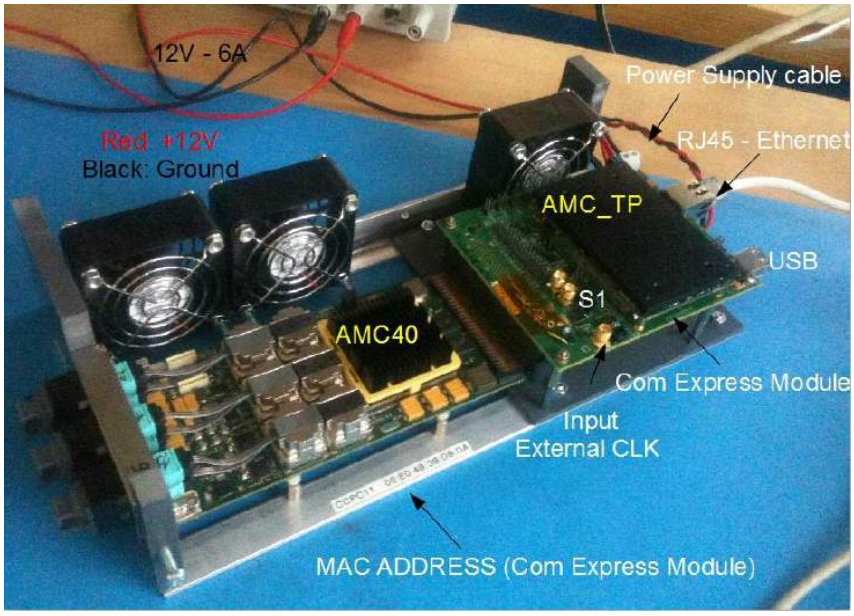


Figura 74 – Kit do MiniDAQ

sistema SCADA WinCC (3) no CCPC. Os testes específicos do LHCb foram realizados utilizando este kit como Back-End.

8.2 Testes

O primeiro *setup* de testes é baseado na placa eletrônica desenvolvida para avaliação e testes do chip GBTx, chamada GBTx SAT Board (79). Os principais elementos desta placa são o chip GBTx, um FPGA Cyclone V da fabricante Altera e conectores ópticos do tipo SFP. Utilizando esta placa como uma emulação da eletrônica Backend de um experimento, foi possível modificar o firmware do FPGA de modo a incluir o bloco SOL40-SCA, ligando-o à lógica existente de alimentação de dados do chip GBTx. Deste modo, o SOL40-SCA se comunicava com o GBT-SCA da VLDB principal através de uma ponte de dados óptica, transparente do ponto de vista operacional, através dos chips GBTx contidos nas duas diferentes placas.

O segundo setup dos testes em hardware utiliza o MiniDAQ (51) como portadora do bloco SOL40-SCA. Esta placa, específica do upgrade do experimento LHCb, serve como protótipo das placas de interface de Backend do LHCb (42, 44). Ela inclui um FPGA Stratix V com o firmware da eletrônica de aquisição (Readout), controle rápido (TFC) e controle lento (ECS), onde o firmware deste último é chamado de SOL40. o MiniDAQ também possui um módulo de *Credit-Card PC* que funciona como um computador linux operando o software do supervisório do sistema de controle e depuração da eletrônica, seja através de scripts ou através de módulos no sistema SCADA WinCC, da Siemens. Neste último foram realizados testes utilizando scripts e módulos WinCC, estes últimos de autoria de Maurício Féo do grupo de pesquisa no CBPF.

A figura 75 mostra a quantidade de recursos lógicos que a versão v1.1 do firmware do SOL40-SCA utilza no projeto no FPGA do MiniDAQ. Apesar do número de unidades de memória da Buffer e Protocol Layer serem gerenciáveis, o maior peso fica por conta das 16 instâncias do bloco SCA MAC Layer, uma por link GBT-SCA, que juntas correspondem a mais de 50% do uso de ALMs do bloco SOL40-SCA e que, por sua vez, corresponde a aproximadamente 25% da quantidade usada por todo o firmware do projeto.

Fitter Resource Utilization by Entity

	Compilation Hierarchy Node	[A] ALMs used in final placement	DSP Blocks	M20Ks
1	|Q_Top_Level_comp|	59219.2 (196.5)	2	1174
1	|Clocks_Generator:Clocks|	25.5 (9.5)	0	0
2	|Q_SODIN_Top_Level:SODIN|	4863.6 (0.0)	0	13
3	|Q_SOL40_Top_Level:SOL40|	15765.3 (0.0)	2	549
1	|Q_SOL40:SOL40_link1|	15765.3 (400.7)	2	549
1	|SC_IC:SC_IC_SOL40|	1608.5 (0.0)	0	512
2	|STFC_decoder_SOL40:TFC_decoder|	57.2 (16.2)	0	5
3	|fifo_pipeline_2bits:OUT_FE_VALID_PIPE_FIFO|	34.5 (0.0)	0	1
4	|fifo_pipeline_84bits:OUT_FE_PIPE_FIFO|	41.0 (0.0)	0	17
5	|fifo_pipeline_84bits:OUT_SODIN_PIPE_FIFO|	38.0 (0.0)	0	1
6	|fifo_pipeline_84bits:OUT_TELL40_PIPE_FIFO|	39.0 (0.0)	0	3
7	|sol40_sca:SOL40_SCA_CORE|	13546.5 (0.0)	2	10
1	|ecs_buffer_layer:ecs_buffer_layer_inst|	982.3 (0.0)	2	10
1	|ecs_cmd_fifo:ecs_cmd_fifo_inst|	17.8 (17.8)	0	5
1	|altsyncram:cmd_fifo[0].gbt_nr[7]__1|	0.0 (0.0)	0	5
1	|altsyncram_4kr1:auto_generated|	0.0 (0.0)	0	5
2	|ecs_rpy_mem:ecs_rpy_mem_inst|	964.5 (964.5)	2	5
1	|altsyncram:mem_rtl_0|	0.0 (0.0)	0	5
1	|altsyncram_4or1:auto_generated|	0.0 (0.0)	0	5
2	|ecs_interface_layer:ecs_interface_layer_inst|	280.8 (280.8)	0	0
3	|mac_layer:mac_layer_inst|	7586.1 (0.0)	0	0
1	|sca_mac_layer:\gen_by_gb...sca:0:sca_mac_layer_inst|	447.7 (22.5)	0	0
1	|eport_driver:eport_driver_inst|	89.7 (89.7)	0	0
2	|fpga_elink:fpga_elink_inst|	335.4 (0.0)	0	0
1	|HDLC:HDLC_inst|	335.4 (0.0)	0	0
1	|rx:rx_inst|	166.7 (2.6)	0	0
1	|MAC_rx:MAC_rx_pri|	121.3 (89.2)	0	0
1	|fifo_mem:fifo_mem_inst|	32.2 (19.7)	0	0
1	|altsyncram:mem_rtl_0|	12.5 (0.0)	0	0
1	|altsyncram_41n1:auto_generated|	12.5 (12.5)	0	0
2	|PHY_HDLC_rx:PHY_HDLC_rx_pri|	22.8 (22.8)	0	0
3	|crc:crc_pri_inst|	18.6 (18.6)	0	0
4	|monostable:monostable_rx_cmd_reset_pri|	1.3 (1.3)	0	0
2	|tx:tx_inst|	168.8 (0.7)	0	0
1	|MAC_tx:MAC_tx_inst|	96.0 (66.6)	0	0
1	|fifo_mem:fifo_mem_inst|	29.5 (18.1)	0	0
1	|altsyncram:mem_rtl_0|	11.3 (0.0)	0	0
1	|altsyncram_41n1:auto_generated|	11.3 (11.3)	0	0
2	|PHY_HDLC_tx:PHY_HDLC_tx_inst|	58.7 (58.7)	0	0
3	|crc:crc_inst|	10.8 (10.8)	0	0
4	|regbank:regbank_inst|	2.6 (2.6)	0	0

Figura 75 – Utilização de recursos do firmware do SOL40-SCA

8.3 Signal TAP

Uma parte fundamental para a depuração dos testes realizados em hardware foi a ferramenta Signal TAP (95) da Altera, um analisador lógico programável criado dentro da lógica do próprio FPGA. Quando ainda não havia uma ferramenta de software para o controle do firmware via JTAG, e mesmo a interface de dados para o usuário do bloco não era do tipo Avalon MM, o método usado era o de ativar sequências de operações de estímulo ao bloco em firmware que, por sua vez, eram ativados pela ferramenta In-System Source and Probes (96).

As figuras 76 e 77 ilustram como ocorriam os passos que o sistema em execução realizava, quando sob o comando de operação de leitura I²C. A primeira coluna da figura 76, na parte esquerda superior, mostra a interface ECS sendo ativada através do pulso do sinal de escrita de novos comandos WRITE_ECS_COMMAND_i, preenchendo, então, o sequência de dados do comando ECS no sinal ECS_COMMAND_i.

As etapas seguintes equivalem ao envio do comando ao SCA associado, através da janela de dados definida pelo sinal tx_ena no bloco Eport, e na recepção associado ao sinal rx_ena. A cada janela em que o sinal tx_ena está em nível alto, o E-Port recebe um novo pacote SCA correspondente a um comando em tx_dat . Neste caso um comando de escrita do registrador de endereço da memória do GBT-SCA, via I²C, e na coluna do meio o comando de leitura do dispositivo I²C.

A figura 77 mostra o resultado do comando de leitura da resposta ECS, resultante da recepção de um pacote novo vindo do SCA, representado na coluna da esquerda no sinal rx_dat, e do fim da operação de *polling* feita pelo usuário.

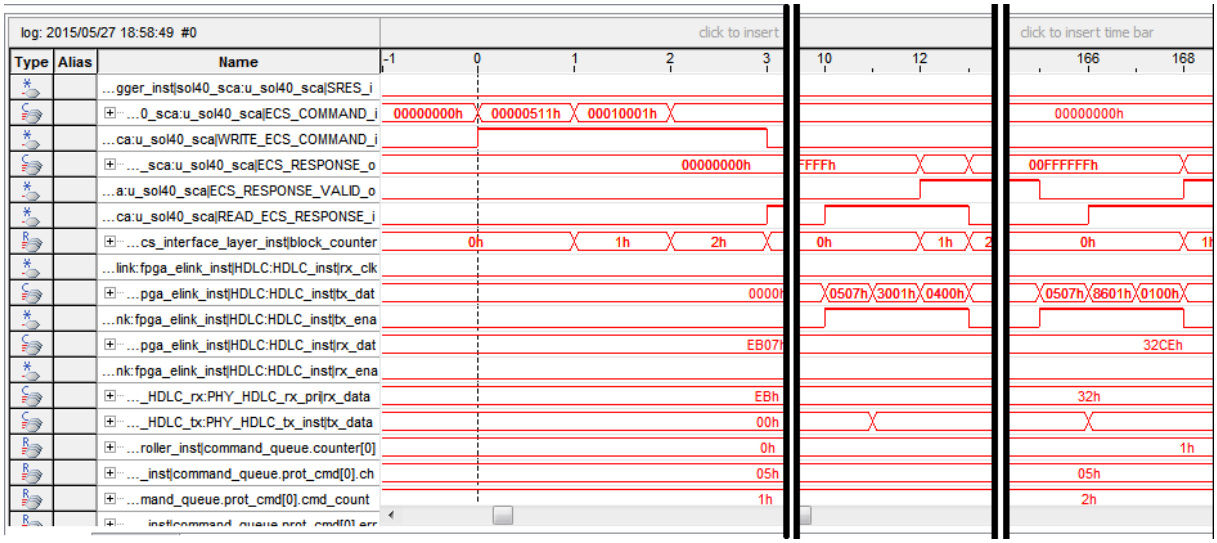


Figura 76 – Exemplo de um ciclo completo de uma operação de leitura utilizando o barramento I²C, primeira parte.

8.4 Software SOL40-SCA Evaluation Tool

Para a realização rápida e robusta dos testes funcionais deste primeiro setup foi criado, como parte deste trabalho, um software em linguagem Tcl/Tk (97, 98) para programação e comando do GBT-SCA, através da SOL40-SCA, a partir de um computador operando o software Quartus da Altera. A ferramenta, mostrada na figura 79 se conecta com o FPGA através do cabo de conexão JTAG, na USB Blaster (94), e então com o SOL40-SCA, por meio de um bloco chamado JTAG-to-Avalon MM(50, 99), ilustrado na

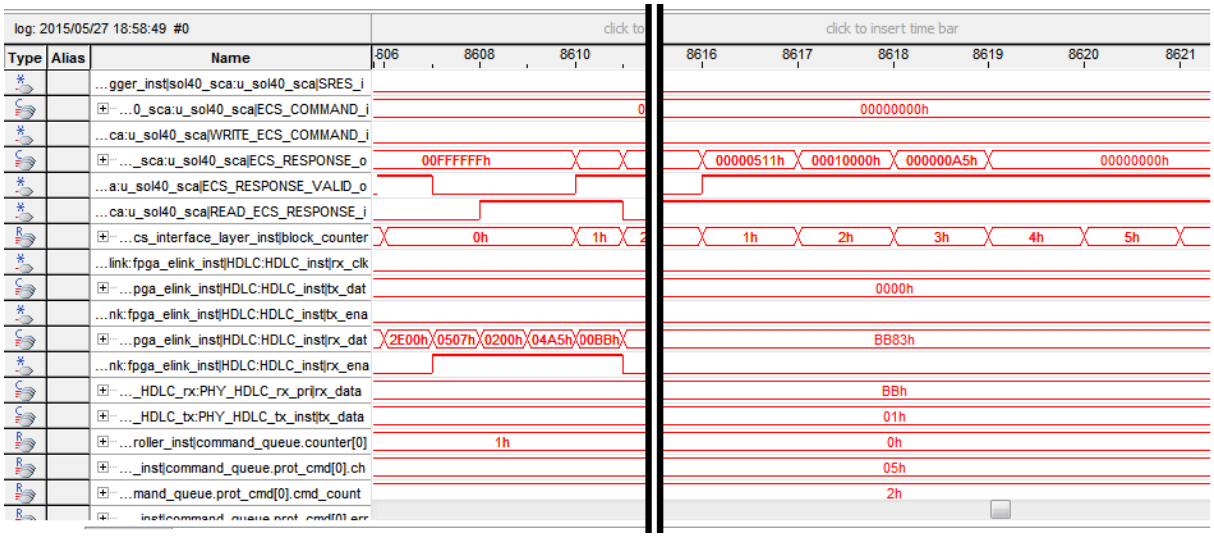


Figura 77 – Continuação da figura 76, segunda e última parte.

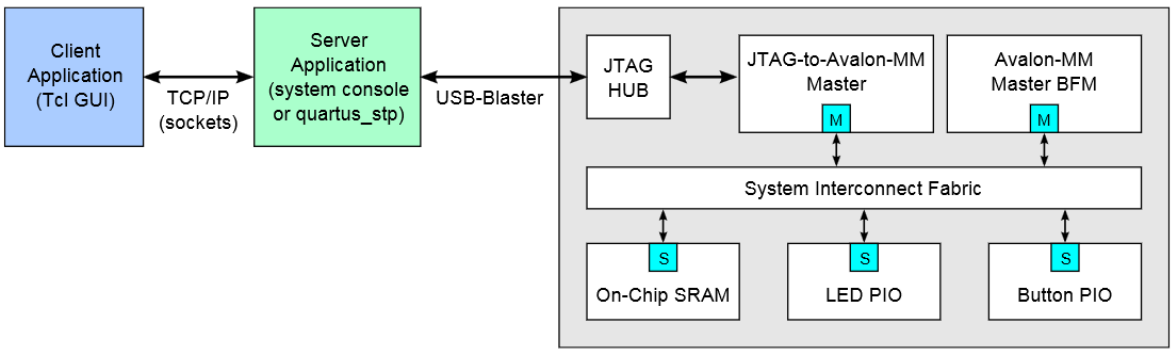


Figura 78 – Diagrama de blocos do método de comunicação do PC com o firmware através do bloco JTAG to Avalon MM.

figura 78. Este software foi baseado no trabalho Altera JTAG-to-Avalon-MM Tutorial de D. W. Hawkins (80).

O software foi usado como prova de conceito como uma maneira fácil de acessar circuitos lógicos internos ao firmware de um FPGA Altera durante o tempo de execução, sendo incorporado como opção de reserva de acesso aos blocos de controle do GBTx e outras funcionalidades no ambiente de desenvolvimento do firmware do MiniDAQ e inspirando uma ferramenta de softwares suporte para a incorporação do bloco ao projeto GBT-FPGA (100), isto é, não dependendo do funcionamento de interfaces Ethernet ou PCIe nesses sistemas.

A ferramenta permitia a opção de escolha do link GBT, recurso depreciado atualmente, escolha do link SCA, canal do SCA em CH e comando ECS em questão em CMD. O preenchimento dos outros campos do grupo ECS Command é opcional. Uma vez ajustando os parâmetros o usuário comandava o software a enviar o comando ao bloco. A escolha do SCA e canal para realização de *polling* era escolhida no grupo ECS Reply Address

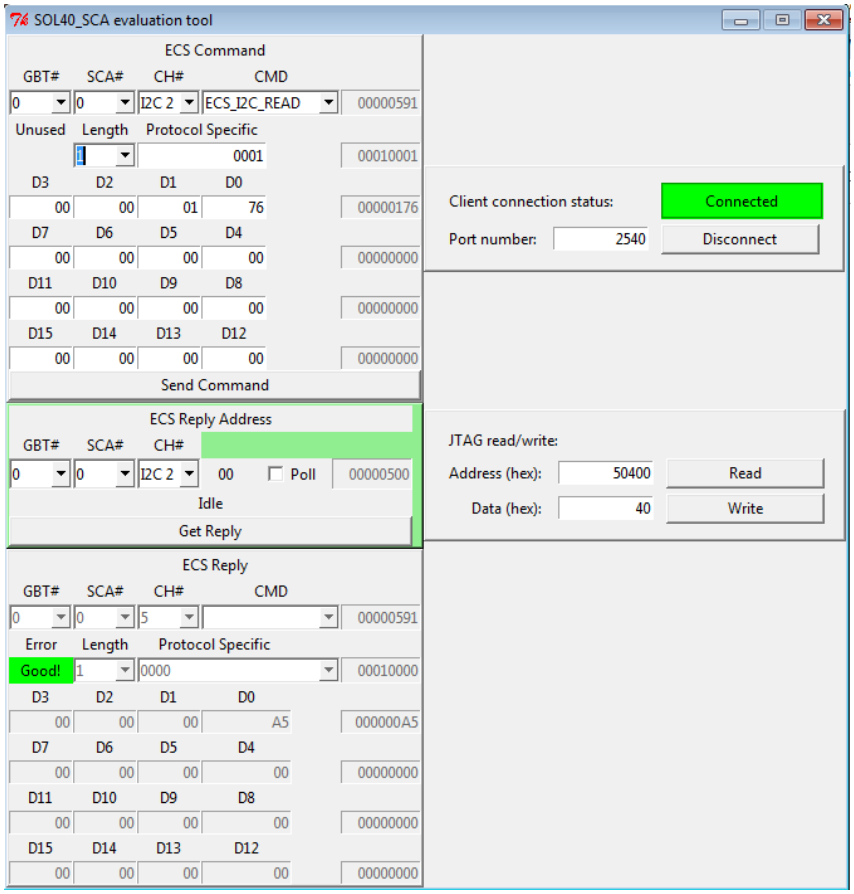


Figura 79 – Interface gráfica em Tcl/Tk para o software para testes do SOL40-SCA.

enquanto que a leitura era realizada pressionando o botão Get Reply, que preenchia os campos do grupo ECS Reply com os dados do pacote de respota ao comando original.

Para compreensão da cadeia de elementos envolvidos no primeiro *setup* o diagrama em blocos da figura 80 mostra , de forma abstrata, o papel de cada um deles. Começando na ferramenta SOL40-SCA evaluation tool

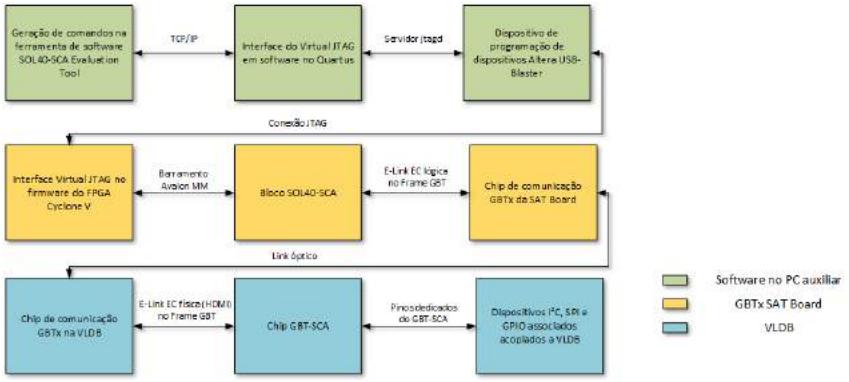
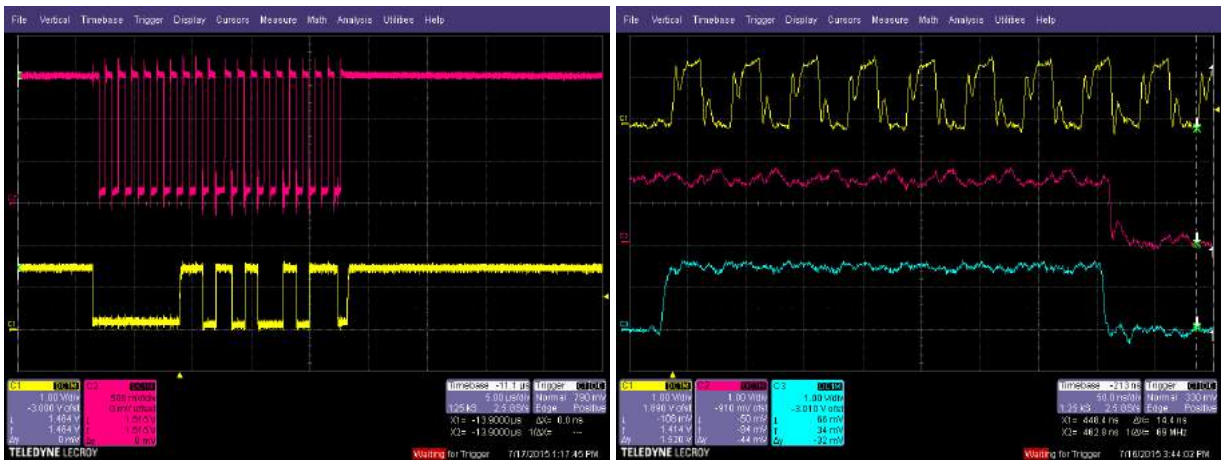


Figura 80 – Diagrama da cadeia de elementos do primeiro *setup* com a ferramenta em software.

8.5 Testes com dispositivos

As interfaces testadas e verificadas do GBT-SCA, quando utilizadas nos diferentes *setups* ,incluem os barramentos ADC, DAC, GPIO , JTAG, I²C e SPI. Para os três primeiros foram realizados testes simples, dado o estado frágil do protótipo da VLDB, onde o GBT-SCA disponível precisou ser ressoldado por *wire-bonding*, pois estava com um *footprint* invertido.

Como dispositivos I²C foram usados os chips GBTx, onde alguns dos resultados são representados através da ferramenta SignalTap da Altera na figura **??** e de medição por osciloscópio na figura 81a operando em até 1MHz, e uma placa de microcontrolador Arduino (101) por meio de um I²C *level shifter* baseado no chip PCA9306 da Texas Instruments (102). A interface SPI foi verificada em conjunto com o chip CLARO8v2 (103), que será usado no upgrade do subdetector RICH do LHCb (104) com diferentes velocidades, assim como em modo loopback. A figura 81b mostra o gráfico de medição de uma operação SPI com o chip CLARO operando na velocidade máxima de 20 MHz.



(a) Operação de leitura I²C de 1 byte a 1 MHz no GBTx.

(b) Operação SPI de 128b a 20 MHz no ASIC CLARO.

Figura 81 – Exemplos de operações I²C e SPI vistas num osciloscópio.

Ao final deste trabalho, todos os barramentos fornecidos pelo SCA foram testados. A lista de componentes usados para os teses em hardware é mostrada na tabela 14.

Back-End	Placa com o SCA	Dispositivo/Front-End	Software	Barramento do SCA	Observações
GBTx SAT	VLDB (90)	GBTx ASIC, DC/DC (93)	SOL40_SCA Evaluation tool	I²C, GPIO	Em parceria com Ken Wyllie e Federico Alessio.
GBTx SAT	VLDB	SALT8	SOL40_SCA Evaluation tool	I²C	Em parceria com Jianchun Wang, Ken Wyllie e Federico Alessio.
GBTx SAT	VLDB	CLARO ASIC	SOL40_SCA Evaluation tool	SPI	Em parceria com Roberto Malaguti e Angello Cotta Ramusino.
MiniDAQ	VLDB, SciFi Master Board	GBTx ASIC	SOL40_SCA Evaluation tool, Mauricio's WinCC panel	I²C, SPI	Em parceria com Mauricio Féo.
MiniDAQ	VLDB	GBTx ASIC, DC/DC, SPI memory, Kintex7 FPGA (105)	LHCb Hw Framework	I²C, SPI, GPIO, ADC, DAC	Em parceria com João Barbosa.
MiniDAQ	SciFi Master Board	Pacific ASIC	LHCb Hw Framework	I²C	Testado por Wilco Vink usando o modo de endereçamento I²C de 10 bit.
Xilinx KC705 + E-Link FMC	VLDB	I2C test board, Kintex 7 FPGA	Vivado	I²C, SPI, GPIO, JTAG	Testado por Raul Lesma através da ferramenta Vivado(106) em comunicação com o firmware através de um programa operando numa instância de microcontrolador Microblaze (107).

Tabela 14 – Tabela de arranjos dos equipamentos utilizados nos testes em hardware

Parte IV

Conclusão

Dentro do seu programa de atualização, o experimento LHCb, em parceria com o CBPF, desenvolveu um bloco de firmware para FPGA genérico, visando gerenciar os chips GBT-SCA das eletrônicas de Front-End. Isto foi obtido pela implementação de um código do tipo HDL, capaz de gerenciar todos os protocolos suportados pelo GBT-SCA sobre qualquer link GBT, de forma programável ou configurável em tempo de execução.

Este bloco é tão genérico que pode ser utilizado dentro de qualquer ambiente de BE, que tenha por característica a utilização do chipset GBT. Ultrapassando o uso dentro do upgrade do LHCb, o projeto passou a ser um item incorporado ao projeto GBT-FPGA (100) como firmware responsável pelo controle e monitoração das FE, utilizando o GBT-SCA como sua interface.

O bloco de firmware apresenta a versão v1.1 como funcional e estável, sendo usado nos sistemas de testes nos subdetectores do LHCb e pelo grupo de desenvolvimento de Sistemas Eletrônicas para Experimentos (PH-ESE) do CERN. Uma campanha de testes em conjunto com primeiros chips GBT-SCA foi realizada para fins de testar sua robustez, confiabilidade e compatibilidade, onde está sendo avaliado, sob algumas modificações, o uso deste bloco nos outros experimentos do LHC.

Em conclusão, as partes entregáveis deste trabalho são o firmware do projeto SOL40-SCA, o firmware do projeto FPGA E-Port, parte indispensável ao SOL40-SCA, e a ferramenta de software para comunicação com o SOL40-SCA, para projetos baseados em FPGAs da Altera, chamada SOL40-SCA Evalation Tool.

9 Trabalhos Futuros

O projeto do bloco SOL40-SCA não está finalizado e ainda precisa ser testado pelos grupos de outros experimentos do LHC. Recursos importantes, como facilitar a realização de grandes operações ou transferências de dados de forma paralela e suporte a interrupções do protocolo GPIO, ainda não foram implementadas.

O foco do projeto agora é investigar e adicionar essas duas funcionalidades, além de implementar outros recursos para tornar o projeto mais flexível e universal, como implementar funcionalidades de CDC, adicionar suporte para outras interfaces lógicas de comunicação do usuário com o bloco, via barramento AXI4 (83) ou Wishbone (84). Melhorar as análises de simulação ou testes em hardware, cujo objetivo seria realizar testes funcionais de longa duração ou cobrir todas os possíveis estados lógicos do sistema via vetores aleatórios de dados controlados nos sinais de entrada, como testes não-funcionais, também são possibilidades para garantir a total confiabilidade do projeto.

A partir de outubro de 2016 o desenvolvimento principal do projeto passará para o controle do CERN, e, em certo ponto, o código do projeto passará a integrar oficialmente o projeto GBT-FPGA.

10 Publicações Associadas

- CAPLAN, CAIRO; RODRIGUES, ANDRÉ ; GASPAR, CLARA; ALESSIO, FEDERICO ; WYLLIE, KEN; JACOBSSON, RICHARD . Development and validation of the ... LHCb upgrade. Notas Técnicas do CBPF, v. 6, p. 8-11, 2016.
  Disponível em <http://dx.doi.org/10.7437/NT2236-7640/2016.01.002>

- ALESSIO, F. ; CAPLAN, C. ; GASPAR, C. ; JACOBSSON, R. ; WYLLIE, K. . A generic firmware core to drive the Front-End GBT-SCAs for the LHCb upgrade. Journal of Instrumentation , v. 10, p. C02013-C02013, 2015.
  Disponível em <http://dx.doi.org/10.1088/1748-0221/10/02/C02013>

- ALESSIO, F.; BARBOSA J.; BARON, S.; CACHEMICHE, J.; CAPLAN, C.; GASPAR, C.; HACHON, F.; JACOBSSON, R.; WYLLIE, K. . Timing and Readout Control in the LHCb Upgraded Readout System. 20th Real Time Conference (RT2016) . IEEE Nuclear & Plasma Science Society.
  Disponível em <https://indico.cern.ch/event/390748/contributions/1825250/>

Fevereiro 2017 LESMA, R.; ALESSIO, F.; BARBOSA, J.; BARON, S.; CAPLAN, C.; LEITAO, P.; PECORARO, C.; PORRET, D.; WYLLIE, K. . The Versatile Link Demonstrator Board (VLDB). TWEPP 2016 - Topical Workshop on Electronics for Particle Physics.
Disponível em <http://stacks.iop.org/1748-0221/12/i=02/a=C02020>

Referências

1 EVANS, L.; BRYANT, P. Lhc machine. *Journal of Instrumentation*, v. 3, n. 08, p. S08001, 2008. Disponível em: <http://stacks.iop.org/1748-0221/3/i=08/a=S08001>. Citado na página 20.

2 THE LHCb Detector at the LHC. *JINST*, v. 3, n. 08, p. S08005, 2008. Disponível em: <http://stacks.iop.org/1748-0221/3/i=08/a=S08005>. Citado na página 24.

3 WCMS3Portfolio, SCADA System SIMATIC WinCC - HMI Software - Siemens. Disponível em: <http://w3.siemens.com/mcms/human-machine-interface/en/visualization-software/scada/pages/default.aspx>. Citado 3 vezes nas páginas 36, 64 e 113.

4 COUNCIL, T. C. *The European strategy for Particle Physics*. 2006. Disponível em: <http://council.web.cern.ch/council/en/EuropeanStrategy/ESStatement.pdf>. Citado na página 37.

5 KRAMMER, M. The update of the european strategy for particle physics. *Physica Scripta*, v. 2013, n. T158, p. 014019, 2013. Disponível em: <http://stacks.iop.org/1402-4896/2013/i=T158/a=014019>. Citado na página 37.

6 AAD, G. et al. Observation of a new particle in the search for the standard model higgs boson with the {ATLAS} detector at the {LHC}. *Physics Letters B*, v. 716, n. 1, p. 1 – 29, 2012. ISSN 0370-2693. Disponível em: <http://www.sciencedirect.com/science/article/pii/S037026931200857X>. Citado na página 37.

7 CHATRCHYAN, S. et al. Observation of a new boson at a mass of 125 gev with the {CMS} experiment at the {LHC}. *Physics Letters B*, v. 716, n. 1, p. 30 – 61, 2012. ISSN 0370-2693. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0370269312008581>. Citado na página 37.

8 SITE do projeto HL-LHC. Disponível em: <http://hilumilhc.web.cern.ch/about/hl-lhc-project>. Citado 2 vezes nas páginas 37 e 39.

9 ARNAUDON, L. et al. *Linac4 Technical Design Report*. Geneva, 2006. Revised version submitted on 2006-12-14 09:00:40. Disponível em: <https://cds.cern.ch/record/1004186>. Citado na página 37.

10 LETTER of Intent for the LHCb Upgrade. Geneva, 2011. Disponível em: <https://cds.cern.ch/record/1333091>. Citado 2 vezes nas páginas 38 e 50.

11 ABELEV, B. et al. *Upgrade of the ALICE Experiment: Letter of Intent*. Geneva, 2012. Disponível em: <https://cds.cern.ch/record/1475243>. Citado na página 38.

12 LETTER of Intent for the Phase-I Upgrade of the ATLAS Experiment. Geneva, 2011. Disponível em: <https://cds.cern.ch/record/1402470>. Citado na página 38.

13 ATLAS, C. *Letter of Intent for the Phase-II Upgrade of the ATLAS Experiment*. Geneva, 2012. Draft version for comments. Disponível em: <https://cds.cern.ch/record/1502664>. Citado na página 38.

14 COLLABORATION, C. *Technical proposal for the upgrade of the CMS detector through 2020*. [S.l.], 2011. Disponível em: <https://cds.cern.ch/record/1355706>. Citado na página 38.

15 RADIATION hard optical link. Disponível em: <http://ph-dep-ese.web.cern.ch/ph-dep-ese/optical_link/optical_link.html>. Citado 2 vezes nas páginas 39 e 109.

16 MOREIRA, P.; MARCHIORO, A.; KLOUKINAS. The GBT: A proposed architecure for multi-Gb/s data transmission in high energy physics. 2007. Disponível em: <https://cds.cern.ch/record/1091474>. Citado na página 39.

17 THE GBT Ser-Des ASIC prototype. *JINST*, v. 5, n. 11, p. C11022, 2010. Disponível em: <http://stacks.iop.org/1748-0221/5/i=11/a=C11022>. Citado 4 vezes nas páginas 39, 54, 55 e 65.

18 PAPOTTI, G. *Architectural studies of a radiation-hard transceiver ASIC in 0.13 um CMOS for digital optical links in High Energy Physics applications*. Tese (Doutorado) — Parma U., 2007. Presented Jan 2007. Disponível em: <http://cds.cern.ch/record/1407182>. Citado na página 39.

19 MENOUNI, M.; GUI, P.; MOREIRA, P. The GBTIA, a 5 Gbit/s Radiation-Hard Optical Receiver for the SLHC Upgrades. 2009. Disponível em: <http://cds.cern.ch/record/1235833>. Citado 2 vezes nas páginas 39 e 40.

20 CARATELLI, A. et al. The gbt-sca, a radiation tolerant asic for detector control and monitoring applications in hep experiments. *Journal of Instrumentation*, v. 10, n. 03, p. C03034, 2015. Disponível em: <http://stacks.iop.org/1748-0221/10/i=03/a=C03034>. Citado 6 vezes nas páginas 39, 55, 60, 61, 65 e 110.

21 MARIN, M. B. et al. The GBT-FPGA core: features and challenges. *J. Instrum.*, v. 10, n. 03, p. C03021, 2015. Disponível em: <http://cds.cern.ch/record/2158963>. Citado 5 vezes nas páginas 39, 41, 65, 98 e 111.

22 MAZZA, G. et al. The gbld: a radiation tolerant laser driver for high energy physics applications. *Journal of Instrumentation*, v. 8, n. 01, p. C01033, 2013. Disponível em: <http://stacks.iop.org/1748-0221/8/i=01/a=C01033>. Citado na página 40.

23 ALTERA Corporation. Disponível em: <http://www.altera.com/>. Citado 3 vezes nas páginas 41, 74 e 111.

24 XILINX, Inc. Disponível em: <http://www.xilinx.com/>. Citado na página 41.

25 MICROSEMI Corporation. Disponível em: <http://www.microsemi.com/>. Citado na página 41.

26 LATTICE Semiconductor Corporation. Disponível em: <http://www.latticesemi.com/>. Citado na página 41.

27 BONACINI, S.; KLOUKINAS, K.; MOREIRA, P. E-link: A Radiation-Hard Low-Power Electrical Link for Chip-to-Chip Communication. 2009. Disponível em: <https://cds.cern.ch/record/1235849>. Citado 5 vezes nas páginas 43, 49, 61, 90 e 105.

28 E-PORT IP Core specifications document v0.3. Citado 2 vezes nas páginas 43 e 65.

29 INFORMATION technology – Telecommunications and information exchange between systems – High-level data link control (HDLC) procedures. Geneva, CH, 2002. v. 2002. Disponível em: <http://www.iso.org/iso/catalogue_detail.htm?csnumber=37010>. Citado 5 vezes nas páginas 43, 62, 67, 90 e 94.

30 STANDARD, J. Scalable low-voltage signaling for 400mv (slvs-400). *JESD8-13, O ctober*, 2001. Citado na página 43.

31 CORPORATION, A. Atlantic interface. 2002. Disponível em: <http://www.altera.com/literature/fs/fs_atlantic.pdf>. Citado 3 vezes nas páginas 43, 62 e 90.

32 GAILLARD, R. Single event effects: Mechanisms and classification. In: *Soft Errors in Modern Electronic Systems*. [S.l.]: Springer, 2011. p. 27–54. Citado na página 44.

33 LYONS, R. E.; VANDERKULK, W. The use of triple-modular redundancy to improve computer reliability. *IBM Journal of Research and Development*, v. 6, n. 2, p. 200–209, April 1962. ISSN 0018-8646. Citado na página 44.

34 HAMMING, R. W. Error detecting and error correcting codes. *Bell System technical journal*, Wiley Online Library, v. 29, n. 2, p. 147–160, 1950. Citado 3 vezes nas páginas 44, 62 e 105.

35 PETERSON, W. W.; BROWN, D. T. Cyclic codes for error detection. *Proceedings of the IRE*, v. 49, n. 1, p. 228–235, Jan 1961. ISSN 0096-8390. Citado 4 vezes nas páginas 44, 62, 90 e 105.

36 REED, I. S.; SOLOMON, G. Polynomial codes over certain finite fields. *Journal of the Society for Industrial and Applied Mathematics*, v. 8, n. 2, p. 300–304, 1960. Disponível em: <http://dx.doi.org/10.1137/0108018>. Citado na página 48.

37 GBTX Manual. Disponível em: <https://espace.cern.ch/GBT-Project/GBTX/Manuals/gbtxManual.pdf>. Citado 2 vezes nas páginas 49 e 110.

38 GBT-FPGA User Guide - version 1.4. Disponível em: <https://espace.cern.ch/GBT-Project/GBT-FPGA/default.aspx>. Citado 2 vezes nas páginas 49 e 67.

39 PERFORMANCE of the LHCb Detector during the LHCb Proton Runs 2010-2012. *IEEE Nucl. Sci. Symp. Med. Imag. Conf.*, v. 2012, p. 1479, 2012. Disponível em: <http://dx.doi.org/10.1109/NSSMIC.2012.6551357>. Citado na página 50.

40 *Framework TDR for the LHCb Upgrade*, 2012. Citado 2 vezes nas páginas 50 e 60.

41 *LHCb Trigger and Online Upgrade Technical Design Report*, 2014. Citado na página 50.

42 WYLLIE, K. et al. *Electronics Architecture of the LHCb Upgrade*. Geneva, 2013. Previous version: 01/03/2013 Current version: 18/06/2013 (version 2.6) based on LHCb-INT-2011-006 (removed from CDS). Disponível em: <http://cds.cern.ch/record/1340939>. Citado 4 vezes nas páginas 51, 65, 112 e 113.

43 TRIGGER-LESS readout architecture for the upgrade of the LHCb experiment at CERN. *JINST*, v. 8, n. 12, p. C12019, 2013. Disponível em: <http://stacks.iop.org/1748-0221/8/i=12/a=C12019>. Citado na página 53.

44 TIMING and Fast Control for the Upgraded Readout Architecture of the LHCb experiment at CERN. *IEEE Trans. Nucl. Sci.*, v. 60, p. 3438, 2013. Disponível em: <http://dx.doi.org/10.1109/TNS.2013.2281210>. Citado 4 vezes nas páginas 54, 60, 65 e 113.

45 A New Readout Control system for the LHCb upgrade at CERN. *JINST*, v. 7, n. 11, p. C11010, 2012. Disponível em: <http://stacks.iop.org/1748-0221/7/i=11/a=C11010>. Citado 2 vezes nas páginas 55 e 94.

46 THE LHCb Experiment Control System: on the path to full automation, at 13th International Conference on Accelerator and Large Experimental Physics Control Systems. p. , pg. 20, 2011. Citado na página 56.

47 KAESLIN, H. *Top-Down Digital VLSI Design: From Architectures to Gate-Level Circuits and FPGAs*. Elsevier Science, 2014. ISBN 9780128007723. Disponível em: <https://books.google.com.br/books?id=PiWOAwAAQBAJ>. Citado na página 131.

48 LEBLEBICI, Y.; KIM, C.; KANG, S. *CMOS Digital Integrated Circuits Analysis & Design*. McGraw-Hill Education, 2014. ISBN 9780073380629. Disponível em: <https://books.google.com.br/books?id=8wSTygAACAAJ>. Citado na página 131.

49 HOROWITZ, P.; HILL, W. *The Art of Electronics*. 3. ed. Cambridge: Cambridge University Press, 2015. 1224 p. Citado 5 vezes nas páginas 79, 82, 103, 133 e 135.

50 IEEE Standard for Test Access Port and Boundary-Scan Architecture. *IEEE Std 1149.1-2013 (Revision of IEEE Std 1149.1-2001)*, p. 1–444, May 2013. Citado 3 vezes nas páginas 84, 115 e 137.

51 LHCB upgrade MiniDAQ Handbook. Citado 3 vezes nas páginas 61, 112 e 113.

52 ALTERA Stratix V Device Overview. Disponível em: <https://www.altera.com/content/dam/altera-www/global/en_US/pdfs/literature/hb/stratix-v/stx5_51001.pdf>. Citado na página 61.

53 JOHNSON, J. M.; WIRTHLIN, M. J. Voter insertion algorithms for fpga designs using triple modular redundancy. In: *Proceedings of the 18th Annual ACM/SIGDA International Symposium on Field Programmable Gate Arrays*. New York, NY, USA: ACM, 2010. (FPGA '10), p. 249–258. ISBN 978-1-60558-911-4. Disponível em: <http://doi.acm.org/10.1145/1723112.1723154>. Citado 2 vezes nas páginas 62 e 105.

54 QUARTUS Prime. Disponível em: <https://www.altera.com/products/design-software/fpga-design/quartus-prime/overview.html>. Citado 2 vezes nas páginas 64 e 97.

55 SIGASI Studio Creator. Disponível em: <http://www.sigasi.com/>. Citado na página 64.

56 ECLIPSE IDE. Disponível em: <https://eclipse.org/>. Citado na página 64.

57 MENTOR Graphics Modelsim. Disponível em: <https://www.mentor.com/products/fv/modelsim/>. Citado 2 vezes nas páginas 64 e 103.

58 DOXYGEN. Disponível em: <http://www.stack.nl/~dimitri/doxygen/>. Citado na página 64.

59 IEEE Standard VHDL Language Reference Manual. *ANSI/IEEE Std 1076-1993*, p. i–, 1994. Citado na página 64.

60 CARDOSO, L. G. et al. Controlling daq electronics using a scada framework. In: *2016 IEEE-NPSS Real Time Conference (RT)*. [S.l.: s.n.], 2016. p. 1–4. Citado na página 64.

61 AVALON Interface Specifications. 2015. Citado 3 vezes nas páginas 64, 74 e 95.

62 TROSKA, J. et al. Versatile transceiver developments. *Journal of Instrumentation*, v. 6, n. 01, p. C01089, 2011. Disponível em: <http://stacks.iop.org/1748-0221/6/i=01/a=C01089>. Citado 3 vezes nas páginas 65, 109 e 110.

63 ZIMMERMANN, H. Osi reference model - the iso model of architecture for open systems interconnection. *IEEE Transactions on Communications*, v. 28, n. 4, p. 425–432, April 1980. ISSN 0090-6778. Citado 3 vezes nas páginas 66, 89 e 95.

64 VLIET, H. van. *Software Engineering: Principles and Practice*. Wiley, 2000. ISBN 9780471975083. Disponível em: <https://books.google.com.br/books?id=0HEhAQAAIAAJ>. Citado na página 66.

65 CUMMINGS, C. E. Clock domain crossing (cdc) design & verification techniques using systemverilog. *SNUG-2008, Boston*, 2008. Citado 2 vezes nas páginas 67 e 98.

66 UML Sequence Diagrams. Disponível em: <http://www.uml-diagrams.org/sequence-diagrams.html>. Citado na página 72.

67 BUDRUK, R.; ANDERSON, D.; SHANLEY, T. *PCI Express System Architecture*. [S.l.: s.n.]. Citado na página 74.

68 HABINC, S. Suitability of reprogrammable fpgas in space applications. *Gaisler Research," Feasibility Report*, 2002. Citado na página 84.

69 SOOS, C. *SEUs Effects in FPGAs - How to deal with them?* 2009. Disponível em: <http://indico.cern.ch/event/56796/contributions/2054341/attachments/986860/1403271/SEUs_in_FPGAs_-_final.pdf>. Citado na página 84.

70 VERGINE, T. Mixed-signals integrated circuits for physical experiments. 2015. Citado na página 86.

71 VERGINE, T. et al. A 32-channel 12-bits 65nm wilkinson adc for cms central tracker. In: *Ph.D. Research in Microelectronics and Electronics (PRIME), 2014 10th Conference on*. [S.l.: s.n.], 2014. p. 1–4. Citado na página 86.

72 RIVETTI, A. *CMOS: Front-end Electronics for Radiation Sensors*. [S.l.]: CRC Press, 2015. v. 42. Citado na página 87.

73 GBT-SCA Manual V8.0. Disponível em: <https://espace.cern.ch/GBT-Project/GBT-SCA/Manuals/GBT-SCA_Manual_V8.0.pdf>. Citado 2 vezes nas páginas 88 e 105.

74 COLLABORATION, L. *LHCb Tracker Upgrade Technical Design Report*. [S.l.], 2014. Disponível em: <https://cds.cern.ch/record/1647400>. Citado na página 93.

75 IEEE Standard Hardware Description Language Based on the Verilog(R) Hardware Description Language. *IEEE Std 1364-1995*, p. 1–688, Oct 1996. Citado na página 95.

76 CERN IC Technologies and MPW support. Disponível em: <http://support-ictech-mpws.web.cern.ch/support-ICtech-MPWs/mpw_runs.htm>. Citado na página 95.

77 ALESSIO, F. et al. A generic firmware core to drive the front-end gbt-scas for the lhcb upgrade. *Journal of Instrumentation*, v. 10, n. 02, p. C02013, 2015. Disponível em: <http://stacks.iop.org/1748-0221/10/i=02/a=C02013>. Citado na página 95.

78 GAISLER, J. A structured vhdl design method. *Fault-tolerant microprocessors for space applications*, p. 41–50, 2011. Citado na página 96.

79 LEITAO, P. et al. Test bench development for the radiation hard gbtx asic. *Journal of Instrumentation*, v. 10, n. 01, p. C01038, 2015. Disponível em: <http://stacks.iop.org/1748-0221/10/i=01/a=C01038>. Citado 3 vezes nas páginas 97, 111 e 113.

80 HAWKINS, D. *Altera JTAG-to-Avalon MM Tutorial*. March, 2012. Disponível em: <http://www.ovro.caltech.edu/%7Edwh/correlator/pdf/altera_jtag_to_avalon_mm_tutorial.pdf>. Citado 2 vezes nas páginas 98 e 116.

81 CORPORATION, A. *White Paper - Understanding Metastability in FPGAs*. 2009. Disponível em: <https://www.altera.com/content/dam/altera-www/global/en_US/pdfs/literature/wp/wp-01082-quartus-ii-metastability.pdf>. Citado na página 98.

82 CHU, P. P. *RTL Hardware Design Using VHDL: Coding for Efficiency, Portability, and Scalability*. [S.l.]: Wiley-IEEE Press, 2006. ISBN 0471720925. Citado 2 vezes nas páginas 98 e 104.

83 XILINX, I. *AXI4-Lite IPIF v3.0 - LogiCORE IP Product Guide*. 2016. Disponível em: <http://www.xilinx.com/support/documentation/ip_documentation/axi_lite_ipif/v3_0/pg155-axi-lite-ipif.pdf>. Citado 2 vezes nas páginas 100 e 122.

84 OPENCORES. *Wishbone B4 - WISHBONE System-on-Chip (SoC)Interconnection Architecturefor Portable IP Cores*. 2010. Disponível em: <http://cdn.opencores.org/downloads/wbspec_b4.pdf>. Citado 2 vezes nas páginas 100 e 122.

85 EMBEDDED Memory (RAM: 1-PORT, RAM: 2-PORT, ROM: 1-PORT, and ROM: 2-PORT) User Guide. Citado na página 101.

86 CUMMINGS, C. E. Simulation and synthesis techniques for asynchronous fifo design. In: *SNUG 2002 (Synopsys Users Group Conference, San Jose, CA, 2002) User Papers*. [S.l.: s.n.], 2002. Citado na página 101.

87 NAVABI, Z. *Digital system test and testable design*. [S.l.]: Springer, 2011. Citado na página 104.

88 PERKINS, N. W. G. D. *The Point-to-Point Protocol for the Transmission of Multi-Protocol Datagrams Over Point-to-Point Links*. 1990. Citado na página 105.

89 OPEN Source VHDL Verification Methodology. Disponível em: <http://osvvm.org/>. Citado na página 108.

90 VLDB. Citado 3 vezes nas páginas 109, 110 e 119.

91 AMARAL, L. et al. The versatile link, a common project for super-lhc. *Journal of Instrumentation*, v. 4, n. 12, p. P12003, 2009. Disponível em: <http://stacks.iop.org/1748-0221/4/i=12/a=P12003>. Citado na página 109.

92 SOóS, C. et al. The versatile transceiver: towards production readiness. *Journal of Instrumentation*, v. 8, n. 03, p. C03004, 2013. Disponível em: <http://stacks.iop.org/1748-0221/8/i=03/a=C03004>. Citado na página 109.

93 FACCIO, F. et al. Feast2: A radiation and magnetic field tolerant point-of-load buck dc/dc converter. In: *2014 IEEE Radiation Effects Data Workshop (REDW)*. [S.l.: s.n.], 2014. p. 1–7. ISSN 2154-0519. Citado 2 vezes nas páginas 110 e 119.

94 USB-BLASTER Download Cable User Guide. Disponível em: <https://www.altera.com/content/dam/altera-www/global/en_US/pdfs/literature/ug/ug_usb_blstr.pdf>. Citado 2 vezes nas páginas 112 e 115.

95 DESIGN Debugging Using the SignalTap II Logic Analyzer. Citado na página 114.

96 DESIGN Debugging Using In-System Sources and Probes. Citado na página 114.

97 WELCH, B. B.; JONES, K.; HOBBS, J. *Practical programming in Tcl and Tk*. [S.l.]: Prentice Hall Professional, 2003. v. 1. Citado na página 115.

98 QUARTUS II Scripting Reference Manual. Disponível em: <https://www.altera.com/content/dam/altera-www/global/en_US/pdfs/literature/manual/tclscriptrefmnl.pdf>. Citado na página 115.

99 DEBUGGING with System Console Over TCP/IP. Disponível em: <https://www.altera.com/en_US/pdfs/literature/an/an624.pdf>. Citado na página 115.

100 STATUS update and future plans for the GBT-FPGA project at CERN. Citado 2 vezes nas páginas 116 e 121.

101 Citado na página 118.

102 SPARKFUN Level Translator Breakout - PCA9306. Disponível em: <https://www.sparkfun.com/products/11955>. Citado na página 118.

103 CARNITI, P. et al. Claro-cmos, a very low power asic for fast photon counting with pixellated photodetectors. *Journal of Instrumentation*, v. 7, n. 11, p. P11026, 2012. Disponível em: <http://stacks.iop.org/1748-0221/7/i=11/a=P11026>. Citado na página 118.

104 COLLABORATION, L. *LHCb PID Upgrade Technical Design Report*. Geneva, 2013. Disponível em: <http://cds.cern.ch/record/1624074>. Citado na página 118.

105 XILINX Kintex-7 FPGA KC705 Evaluation Kit. Disponível em: <https://www.xilinx.com/products/boards-and-kits/ek-k7-kc705-g.html>. Citado na página 119.

106 XILINX Vivado Design Suite. Disponível em: <http://www.xilinx.com/products/design-tools/vivado.html>. Citado na página 119.

107 XILINX MicroBlaze Soft Processor Core. Disponível em: <https://www.xilinx.com/products/design-tools/microblaze.html>. Citado na página 119.

APÊNDICE A – Referenciais Teóricos

Esta capítulo de apêndice apresenta teoria de ferramentas comuns ao entendimento deste projeto. São elas a seção de estrutura de um dispositivo FPGA, bem como as seções de barramentos eletrônicos comuns em sistemas de controle e monitoração, sendo elas I^2C, SPI e JTAG.

A.1 Estrutura de um FPGA

O FPGA, do inglês *Field-Programmable Gate* e esquematizado na figura 82, é um tipo de dispositivo lógico programável apresentado como um circuito integrado; isto é, sua estrutura funcional pode ser modificada após a fabricação, diferentemente de circuitos integrados tradicionais tais como microprocessadores ou chips de memória(47, 48).

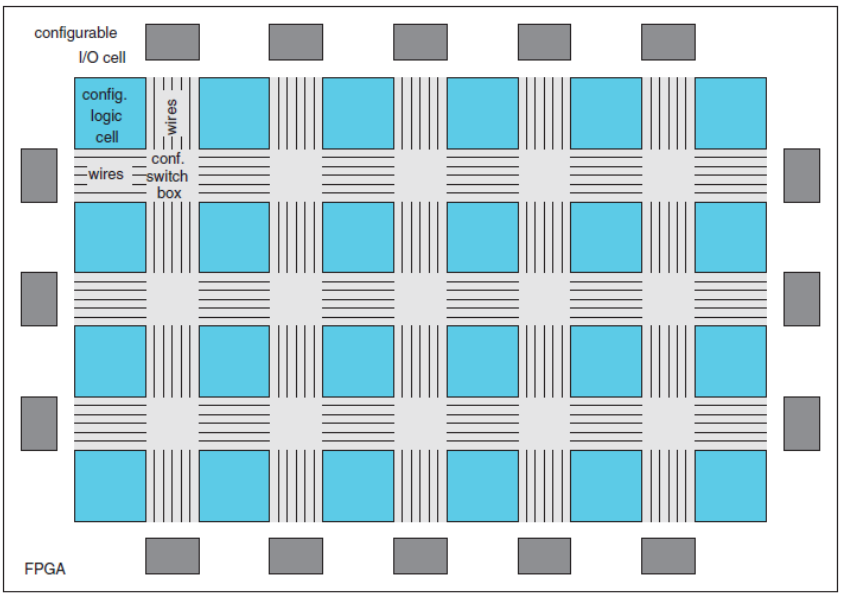


Figura 82 – Arquietura geral de um FPGA.

Um dispositivo FPGA contém milhares de células lógicas com interconexões programáveis, que são disponibilizadas ao usuário pelas sua capacidade de programação de hardware reprogramáveis. Isto torna esse tipo de dispositivo atrativo em várias aplicações, devido ao seu baixos custos inicial e rápido retorno, de forma que é possível transformar um pedaço de silício num circuito especializado para uma aplicação por meios puramente elétricos.

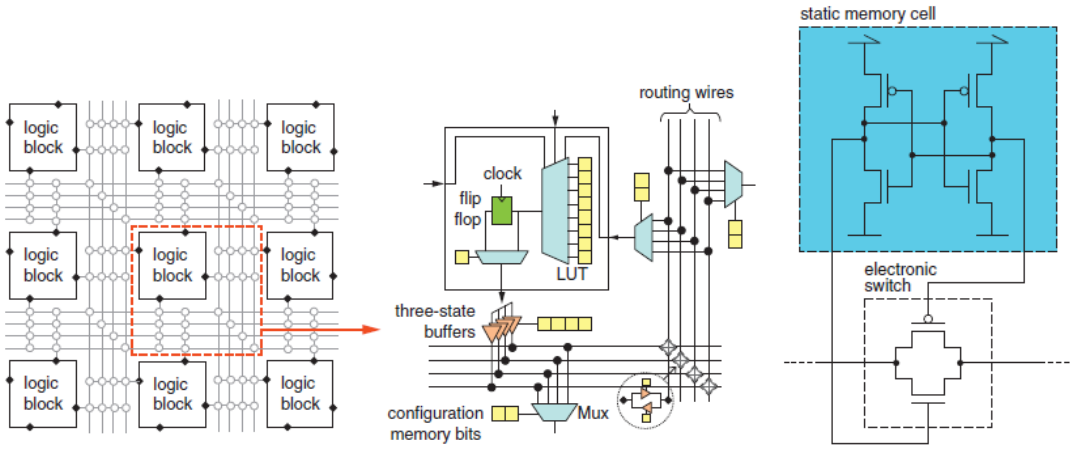
Um chip FPGA consiste de três elementos básicos: Blocos de entrada e saída (I/O *buffers*), uma matriz de células lógicas programáveis e estruturas de interconexões

programáveis. O que define a funcionalidade do FPGA é a forma como as interconexões são feitas, acopladas a uma estrutura de memória.

Atualmente, dois tipos de tecnologia de memória para configuração de FPGA se destacam. A primeira, chamada célula de memória RAM estática (Static RAM ou SRAM), é a tecnologia mais comum e que apresenta como vantagem o número ilimitado de reconfigurações e a possibilidade de reconfigurar parcialmente o chip durante sua operação. A figura 83b mostra o circuito esquemático deste tipo de célula.

A segunda tecnologia, baseada em memórias Flash, apresenta como vantagens reter o seu conteúdo e a configuração do FPGA, mesmo após um ciclo de energia, além de possuir maior resistência a radiação ionizante.

A figura 83a mostra a estrutura de uma célula lógica em FPGA, na figura da esquerda inserida como uma unidade na matriz de células lógicas em meio a interconexões elétricas e na direita os seus elementos internos na forma de um diagrama de blocos, onde se pode destacar as células de memória de configuração representadas em amarelo. Acerca dos elementos que compõem a célula lógica são listados:



(a) Exemplo da estrutura de uma célula lógica programável.

(b) Método de programação por célula de memória estática.

LUT Sigla para *Look-Up Table* e representada pelo trapézio grande em azul na figura, é o elemento básico de operações lógicas, de maneira que pode sintetizar qualquer função booleana dos seus sinais de entrada externos

Flip-Flop Representado em verde na figura, é o elemento básico de armazenamento de dados genéricos, formam registradores quando associados para armazenar um dado específico.

Mux Sigla para *Multiplizer* ou multiplexador e representado pelos trapézios em azul menores. São elementos que reproduzem na sua saída um de seus sinais pinos de

entrada a partir de valores lógicos dos seus pinos de seleção. Usado para selecioanr e rotear sinais através do chip.

As fabricantes de FPGAs podem apresentar customizações quanto a estrutura das suas células ou blocos lógicas, hoje é comum que esses dispositivos possuam unidades dedicadas para DSP, blocos de memória RAM embarcados e até mesmo estruturas mais complexas como conversores ADC ou microprocessadores na mesma pastilha de silício.

A.2 O barramento I²C

O *Inter-Integrated-Circuit* (I²C, I2C ou IIC) é um protocolo de comunicação serial e barramento eletrônico de dois fios utilizado para a comunicação de dispositivos de baixa velocidade, como microcontroladores, EEPROMs, conversores ADC e DAC, interfaces I/O e outros periféricos semelhantes em sistemas embarcados (49). O I²C foi inventado pela empresa Philips Semiconductors, atualmente NXP, e é usado por quase todos os principais fabricantes de circuitos integrados. Cada dispositivo I²C pode ser definido como mestre (Master) ou escravo (Slave), neste caso precisando de um endereço associado no barramento.

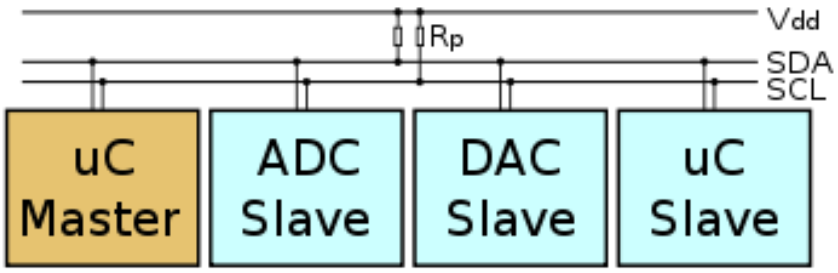


Figura 84 – Exemplo de esquema do barramento I²C com seus dispositivos.

O barramento I²C permite que haja vários dispositivos mestres e escravos, a comunicação é feita de forma serial exigindo que haja apenas dois sinais para a comunicação, características que o ajudaram a se tornar popular. As duas linhas de comunicação do I²C são do tipo coletor ou dreno aberto, sendo chamadas de *Serial Data Line* (SDA) para dados, e *Serial Clock Line* (SCL) para o sinal de clock tal que as duas devem ter no mínimo um resistor de *pull-up* cada.

O barramento I²C atualmente compreende três modos de velocidades: O Standard Mode, velocidade original da especificação de 100 Kb/s; O Fast Mode em 400 Kb/s e o Fast Mode Plus de 1 MHz. O I²C permite velocidades intermediárias pois o sinal de clock, que dita a velocidade da transferência de dados, é gerado pelo mestre de forma que o circuito do escravo seja síncrono a ele.

Figura 85 – Formato do quadro I²C com endereçamento de 7 bits.

Cada dispositivo escravo tem um endereço exclusivo. A especificação do I²C permite modos de endereçamento de 7 bits ou de 10 bits, o segundo retrocompatível com o primeiro.

O formato do quadro do barramento I²C é definido em campos de 8 bits intercalados por bits de sinalização. A figura 85 mostra o quadro I²C para transferências com modo de endereçamento de 7 bits, os campos da figura são definidos a seguir:

Start of Frame (S) Início do quadro, é definido como o evento em que a linha de dados SDA vai para nível baixo enquanto que a de clock SCL permanece em nível alto, controlado pelo dispositivo mestre.

Address Endereço do dispositivo escravo, valor de 7 bits que ocupa o primeiro os bits de 7 a 1 do primeiro byte transmitido.

$R/\overline{W}$ Bit de ativação do modo de leitura, quando em 1, ou escrita, quando 0.

Acknowledge (A) Bit de confirmação de transmissão. Neste caso o controle do clock é passado ao escravo.

Data Bytes de dados carregados pela transmissão I²C.

Stop of Frame (P) Condição de fim de transmissão, caracterizada pela mudança de nível da linha SDA de 0 para 1 quando a linha SCL já está em nível alto, controlado pelo dispositivo mestre.

O modo de endereçamento de 10 bits segue um modelo de quadro parecido com o de 7 bits mas com algumas particularidades. A primeira deles é que 2 bytes são usados para transmissão completa do campo de endereço, mas os primeiros bits do endereço possuem um padrão fixo, correspondente a uma faixa de endereço especial da especificação I²C. Em outras palavras, quando o primeiro byte de uma operação I²C tem o campo de endereço no padrão 11110XX, onde X são valores não fixados.

A segunda particularidade do modo de transmissão de 10b está no conteúdo diferente dos quadros para operação de leitura e escrita. Para o modo de escrita, o primeiro byte transmitido contém a sequência de bits 11110 e os 2 bits mais significativos do endereço de 10 bits, para indicar ao escravo que o próximo bytes serão escritos pelo mestre o campo de leitura ou escrita tem valor 0. O segundo byte de transmissão é todo preenchido pelos 8 bits restantes do endereço e os bytes seguintes são preenchidos com os dados a serem escritos. Estrutura representada na figura 86.

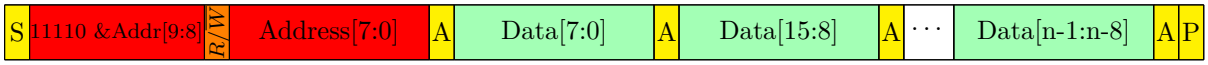


Figura 86 – Formato do quadro I²C em modo escrita usando endereçamento de 10 bits.

Figura 87 – Formato do quadro I²C em modo leitura usando endereçamento de 10 bits.

O modo de leitura em 10b também usa o primeiro byte para repetir a sequência 11110 de bits constantes, seguidos dos 2 bits mais significativos assim como é escolhido o valor 0 no campo de leitura ou escrita. O segundo byte também é preenchido com o bits restantes do endereço, mas dessa vez seguidos de uma nova condição de início do quadro chamada Início Repetido (S_r), nela o mestre se permite mudar de operação de escrita para leitura, mas precisa repetir o primeiro byte de endereços, um mecanismo para manter compatibilidade com as especificações do modo I²C de 7 bits. A estrutura deste quadro é mostrada na figura 87.

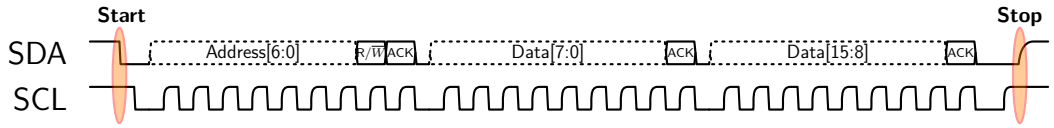


Figura 88 – Exemplo de diagrama de tempo de uma operação I²C.

A.3 O barramento SPI

O barramento para transmissão de dados SPI, do inglês *Serial Peripheral Interface*, é um protocolo de comunicação serial criado pela Motorola mais utilizados para transferência de dados de sensores eletrônicos e módulos de memórias (49). Com configurações bastante flexíveis ele é composto por quatro tipos de sinais:

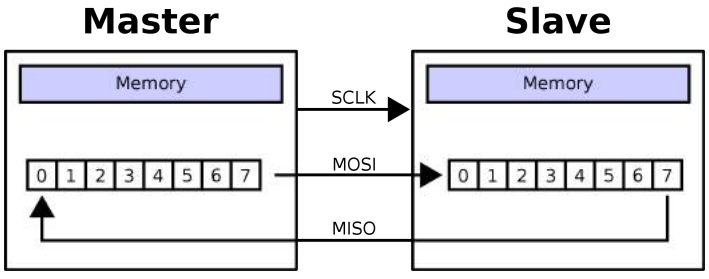


Figura 89 – Transferência de dados de dispositivos SPI.

SCLK Linha de clock do barramento eletrônico

MOSI Sigla de *Master Output - Slave Input*, linha de dados de saída do dispositivo SPI mestre e entrada de dados dos dispositivos SPI escravos.

MISO Sigla de *Master Input - Slave Output*, linha de dados de daída dos dispositivos SPI escravos e entrada do SPI mestre. É preciso considerar que apenas um dispositivo pode alimentar esta linha em dado período de tempo.

SS Sigla de *Slave Select*, grupo de sinais elétricos em que cada um é conectado a um dispositivo escravo diferente, usados para ativação individual do dispositivo no barramento.

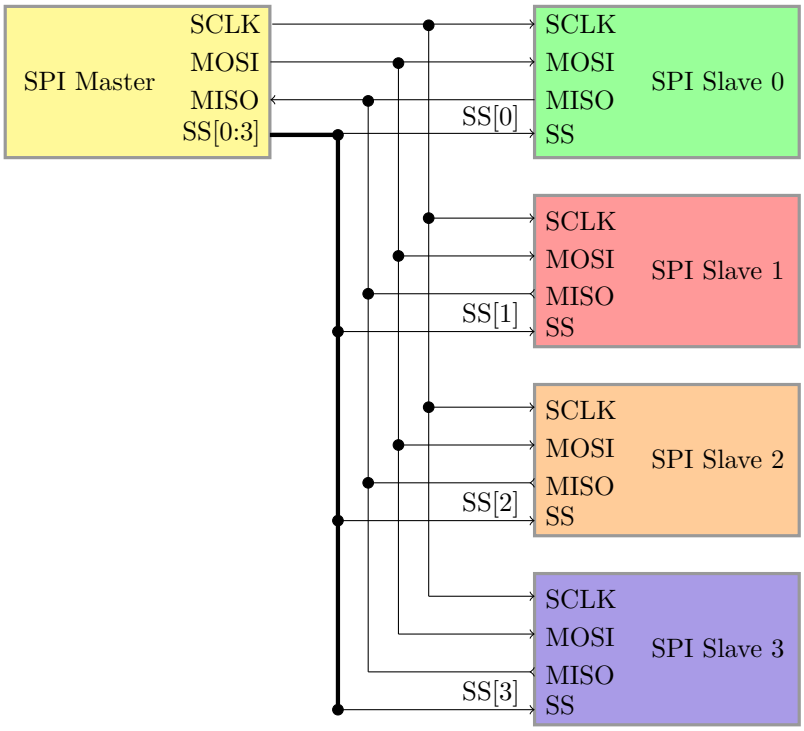


Figura 90 – Topologia de um barramento SPI.

A topologia típica de um barramento SPI é mostrada na figura 91, onde cada dispositivo escravo tem acesso direto às linhas dos sinais do dispositivo mestre, mas com uma linha do barramento SS individual cada. Esta configuração pode ser perigosa caso dois dispositivos, por descuido do dispositivo mestre, sejam ativados e alimentem o sinal MISO, o que poderia danificar tais dispositivos.

Uma segunda opção é o uso da topologia *Daisy-chain* ou encadeada, onde as linhas de saída de dados MOSI dos dispositivos escravos são ligadas às linhas MISO do dispositivo seguinte, e o último da cadeia se ligar a linha MISO do disposivo mestre. Nesta configuração nenhum sinal elétrico pode ser alimentado por mais de um dispositivo.

As transmissões de dados entre dispositivos SPI se comporta internamente como uma operação de rotação de bits dos seus *buffers* internos, de forma mais clara é como se

o último bit de um dos *buffers* envolvidos fosse ligado ao primeiro bit do seguinte e vice-versa, conforme ilustrado na figura 89. Desta forma é natural que o mesmo registrador de comunicação SPI de um dispositivo seja usado tanto para o envio quanto para recebimento de dados.

O barramento SPI não especifica valores de frequências para os sinais de clock ou do nível de ativação do sinal de seleção de dispositivos escravos, ficando a cargo de cada dispositivo escolher esses parâmetros. Por último, a transmissão possui quatro combinações de dois parâmetros de configuração diferentes relativos ao sinal de clock, são eles:

CPOL Define a polaridade do sinal de clock, ou seja, quando CPOL é igual a zero a linha de clock permanece me nível baixo quando ociosa, quando 1 ela permanece em nível alto.

CPHA Define a fase do sinal de clock em que as linhas de dados são atualizadas. Quando CPHA é igual a 0 os sinais MOSI e MISO são atualizados na borda de subida do clock, quando 1 na borda de descida.

A figura 91 mostra como agem as diferentes combinações dos parâmetros CPOL e CPHA.

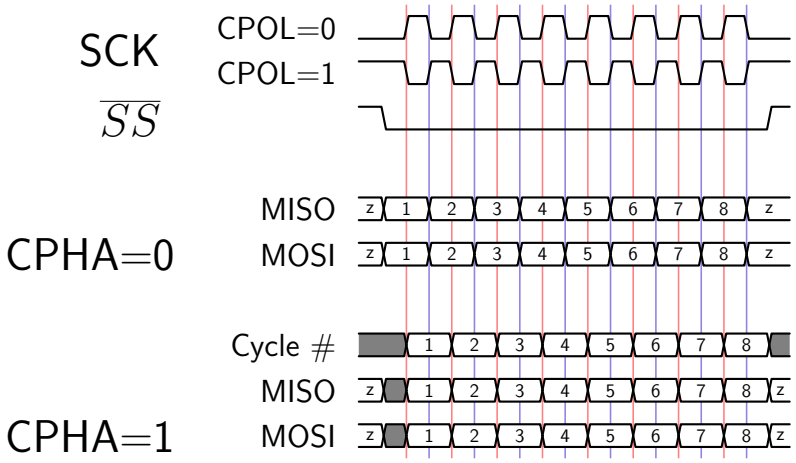


Figura 91 – Diagrama de tempo das coombinações de CPHA e CPOL no barramento SPI.

A.4 O protocolo JTAG

O JTAG nasceu da iniciativa da indústria eletrônica ao formar uma associação, em 1985, para desenvolver um método de verificar projetos e testar placas de circuito impresso após sua manufatura, a associação denominada *Joint Test Action Group* (JTAG) deu nome ao protocolo. Posteriormente em 1990 o Instituto de Engenheiros Eletricistas e Eletrônicos (IEEE) reuniu o resultado desses esforços na norma 1149.1-1990 (50), intitulando como *Standard Test Access Port* e *Boundary-Scan Architecture*.

Hoje o protocolo JTAG não é apenas utilizado para verificação de componentes eletrônicos, mas também se tornou um padrão conhecido para programação, reconfiguração e depuração de circuitos lógicas, FPGAs e softwares em sistemas embarcados.

O protocolo JTAG como barramento eletrônico, mostrado na figura 92 é composto dos seguintes sinais:

TDI Sigla para *Test Data In*, é o sinal com entrada de dados serial usado para instruções, dados de teste ou dados de programação. Os bits deste sinal são amostrados na borda de subida do sinal de clock TCK.

TDO Sigla para *Test Data Output*, sinal de saída dos dados de instrução, testes e programação. Os dados são atualizados na borda de subida do sinal de clock TCK.

TMS Sigla para *Test Mode Select*, sinal de entrada usado exclusivamente para atualizar o estado atual do dispositivo na máquina de estados do controlador TAP. Transições neste sinal também ocorrem na borda de subida de TCK.

TRST Sigla para *Test Reset input*: Sinal opcional no barramento, utilizado para reiniciar o circuito de *Boundary-Scan* de forma assíncrona. Ativo por nível baixo.s

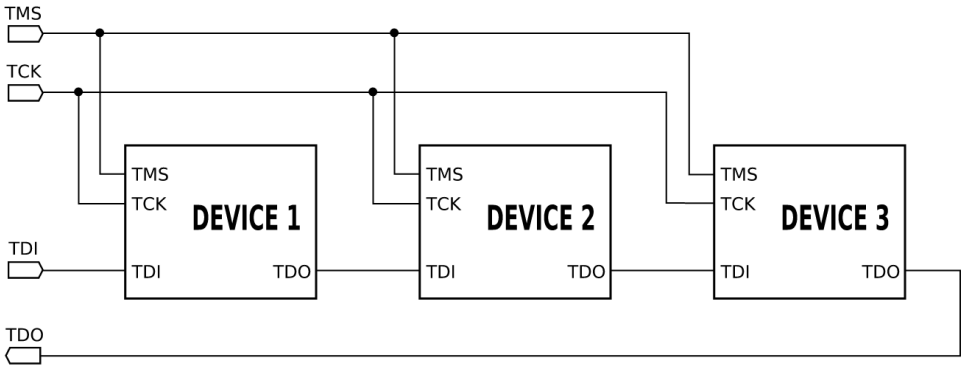


Figura 92 – Topologia de uma *chain* JTAG

O controlador do Test Access Port (TAP) é uma máquina de estado de 16 posições num dispositivo JTAG escravo cujas transições são controladas pelo sinal TMS, mostrada na figura 94. Ele controla o comportamento do sistema JTAG.

O dispositivo JTAG mestre se comunica com os TAPs ao manipular os sinais de TMS e TDI em sincronia com TCK. Desta maneira as transições dos sinais TMS, TDI e TCK criam as primitivas básicas da comunicação JTAG onde os protocolos de software de alto nível são construídos. A maior parte da máquina de estado suporta dois estados estáveis para transferência de dados. Cada controlador TAP possui um registrador de instruções (IR) e um de dados (DR), estes registradores são combinados através do pinos de TDI e TDO para formar um grande registrador de deslocamento, cuja estrutura é mostrada na figura 93, .

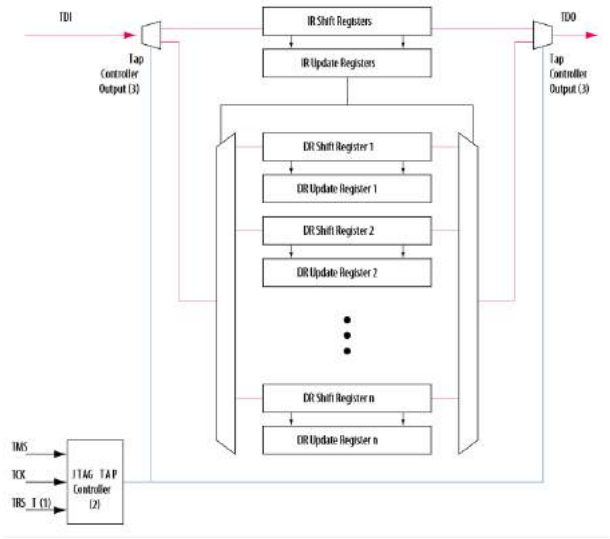


Figura 93 – Arquitetura interna de um circuito com suporte ao protocolo JTAG.

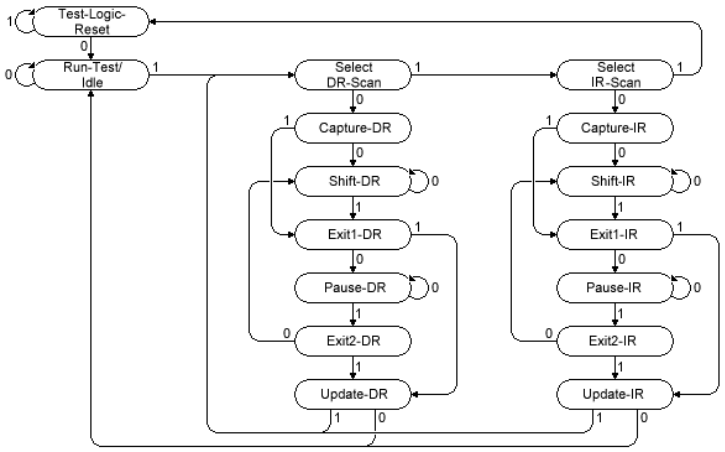


Figura 94 – Máquina de estados do controlador TAP.

APÊNDICE B – Código do projeto SOL40-SCA

B.1 Arquivos de Implementação do firmware

Código B.1 – Código do arquivo principal do firmmware sol40_sca.vhd

```
--------------------------------------------------------------------------------
--! @file sol40_sca.vhd
--! @author Cairo Caplan <cairo.caplan@cern.ch>
--! @date April, 2016
--------------------------------------------------------------------------------

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

use work.Constant_Declaration.all;
use work.SCA_Package.all;

entity sol40_sca is
   generic(
      BASE_ADDRESS         : std_logic_vector(ECS_address_size-1 downto 0)
   );
   port(
      ECS_CLK              : in std_logic;
      ECS_ADDRESS_i        : in std_logic_vector(ECS_address_size-1 downto 0);
       READ_ECS_RESPONSE_i    : in std_logic;
       ECS_RESPONSE_o         : out std_logic_vector(31 downto 0);
       ECS_RESPONSE_VALID_o   : out std_logic;
       ECS_COMMAND_i       : in std_logic_vector (31 downto 0);
       WRITE_ECS_COMMAND_i    : in std_logic;
       --
       txClock40           : in std_logic;
       rxClock40           : in std_logic;
       SRES_i              : in std_logic;
      --
       tx_sd               : out elink_ch_array_t(0 to SCA_COUNT - 1);
        rx_sd              : in elink_ch_array_t(0 to SCA_COUNT - 1));
end sol40_sca;

architecture rtl of sol40_sca is

   component ecs_interface_layer
      generic(BASE_ADDRESS : std_logic_vector(ECS_address_size - 1 downto 0));
      port(
         ECS_ADDRESS_i        : in  std_logic_vector(ECS_address_size - 1 downto 0);
         READ_ECS_RESPONSE_i  : in  std_logic;
         ECS_RESPONSE_o       : out std_logic_vector(31 downto 0);
         ECS_RESPONSE_VALID_o : out std_logic;
         ECS_COMMAND_i        : in  std_logic_vector(31 downto 0);
         WRITE_ECS_COMMAND_i  : in  std_logic;
```

```
      ecs_clk              : in  std_logic;
      SRES_i               : in  std_logic;
      ecs_cmd_int_ena_o    : out std_logic;
      ecs_cmd_int_av_i     : in  std_logic;
      ecs_cmd_int_data_o   : out ecs_packet_t;
      ecs_rpy_int_isRead_o : out std_logic;
      ecs_rpy_int_addr_o   : out ecs_addr_packet_t;
      ecs_rpy_int_data_i   : in  ecs_packet_t;
      new_rpy_i            : in  std_logic;
      sol40_sca_rst_o      : out std_logic;
      reset_sca_o          : out std_logic_vector(63 downto 0);
      disable_sca_o        : out std_logic_vector(63 downto 0);
      timeout_value_o       : out std_logic_vector(25 downto 0);
      sca_link_state_array : in  SCA_link_state_array_t
   );
end component ecs_interface_layer;


component ecs_buffer_layer
   port(clk                  : in  std_logic;
       rst                  : in  std_logic;
       ecs_cmd_int_av_o    : out std_logic;
       ecs_cmd_int_ena_i     : in  std_logic;
       ecs_cmd_int_data_i   : in  ecs_packet_t;
       ecs_rpy_int_isRead_i : in  std_logic;
       ecs_rpy_int_addr_i   : in  ecs_addr_packet_t;
       ecs_rpy_int_av_o     : out std_logic;
       ecs_rpy_int_data_o   : out ecs_packet_t;
       new_rpy_o            : out std_logic;
       ecs_rpy_prot_av_o    : out std_logic;
       ecs_rpy_prot_ena_i   : in  std_logic;
       ecs_rpy_prot_data_i  : in  ecs_packet_t;
       ecs_cmd_prot_ena_i   : in  std_logic;
       ecs_cmd_prot_av_o    : out std_logic;
       ecs_cmd_prot_data_o  : out ecs_packet_t);
end component ecs_buffer_layer;

component protocol_layer
   port(
      clk                 : in  std_logic;
      rst                 : in  std_logic;
      ecs_cmd_prot_ena_o  : out std_logic;
      ecs_cmd_prot_av_i   : in  std_logic;
      ecs_cmd_prot_data_i : in  ecs_packet_t;
      ecs_rpy_prot_av_i   : in  std_logic;
      ecs_rpy_prot_ena_o  : out std_logic;
      ecs_rpy_prot_data_o : out ecs_packet_t;
      timeout_value_i     : in  std_logic_vector(25 downto 0);
      sca_cmd_data_o      : out payload_vector_t(0 to SCA_COUNT - 1);
      sca_cmd_ena_o       : out std_logic_vector(0 to SCA_COUNT - 1);
      sca_cmd_rdy_i       : in  std_logic_vector(0 to SCA_COUNT - 1);
      sca_rpy_data_i      : in  payload_vector_t(0 to SCA_COUNT - 1);
      sca_rpy_prot_rdy_o  : out std_logic_vector(0 to SCA_COUNT - 1);
      send_sca_rpy_i      : in  std_logic_vector(0 to SCA_COUNT - 1);
      reconnnect_o        : out std_logic_vector(0 to SCA_COUNT - 1)
   );
end component protocol_layer;

component mac_layer
   port(
```

```
        tx_clk                : in  std_logic;
        rst                   : in  std_logic;
        sca_cmd_data_i        : in  payload_vector_t(0 to SCA_COUNT - 1);
        sca_cmd_ena_i         : in  std_logic_vector(0 to SCA_COUNT - 1);
        sca_cmd_rdy_o         : out std_logic_vector(0 to SCA_COUNT - 1);
        sca_rpy_data_o        : out payload_vector_t(0 to SCA_COUNT - 1);
        sca_rpy_prot_rdy_i    : in  std_logic_vector(0 to SCA_COUNT - 1);
        sca_rpy_en_o          : out std_logic_vector(0 to SCA_COUNT - 1);
        reconnnect_i          : in  std_logic_vector(0 to SCA_COUNT - 1);
        reset_sca_i           : in  std_logic_vector(63 downto 0);
        disable_sca_i         : in  std_logic_vector(63 downto 0);
        sca_ch_state_o        : out sca_ch_state_vector_t(0 to SCA_COUNT - 1);
        sca_link_state_array_o : out SCA_link_state_array_t;
        tx_sd                 : out elink_ch_array_t(0 to SCA_COUNT - 1);
        rx_clk                : in  std_logic;
        rx_sd                 : in  elink_ch_array_t(0 to SCA_COUNT - 1)
    );
  end component mac_layer;



  signal ecs_cmd_prot_ena : std_logic;
  signal ecs_cmd_prot_av : std_logic;
  signal ecs_rpy_prot_ena : std_logic;
  signal ecs_cmd_prot_data : ecs_packet_t;
  signal ecs_rpy_prot_av : std_logic;
  signal ecs_rpy_prot_data : ecs_packet_t;
  signal sca_cmd_data : payload_vector_t(0 to SCA_COUNT - 1);
  signal sca_cmd_ena : std_logic_vector(0 to SCA_COUNT - 1);
  signal sca_cmd_rdy : std_logic_vector(0 to SCA_COUNT - 1);
  signal sca_rpy_data : payload_vector_t(0 to SCA_COUNT - 1);
  signal send_sca_rpy : std_logic_vector(0 to SCA_COUNT - 1);
  signal sca_rpy_prot_rdy : std_logic_vector(0 to SCA_COUNT - 1);
  signal reconnnect       :   std_logic_vector(0 to SCA_COUNT - 1);
  signal sca_link_state_array : SCA_link_state_array_t;

  signal clk,rst :  std_logic;
  signal sca_ch_state : sca_ch_state_vector_t(0 to SCA_COUNT-1);
  signal ecs_cmd_int_ena : std_logic;
  signal ecs_cmd_int_av : std_logic;
  signal ecs_cmd_int_data : ecs_packet_t;
  signal ecs_rpy_int_ena : std_logic;
  signal ecs_rpy_int_addr : ecs_addr_packet_t := ((others=>'0'),(others=>'0'));
  signal ecs_rpy_int_av : std_logic;
  signal ecs_rpy_int_data : ecs_packet_t;
  signal ecs_rpy_int_isRead : std_logic;
  signal new_rpy : std_logic;
  signal sol40_sca_rst : std_logic;
  signal reset_sca : std_logic_vector(63 downto 0);
  signal disable_sca : std_logic_vector(63 downto 0);
  signal timeout_value : std_logic_vector(25 downto 0);


--
begin

  ecs_interface_layer_inst : component ecs_interface_layer
    generic map(
      BASE_ADDRESS => BASE_ADDRESS
    )
```

```
   port map(
      ECS_ADDRESS_i        => ECS_ADDRESS_i,
      READ_ECS_RESPONSE_i  => READ_ECS_RESPONSE_i,
      ECS_RESPONSE_o       => ECS_RESPONSE_o,
      ECS_RESPONSE_VALID_o => ECS_RESPONSE_VALID_o,
      ECS_COMMAND_i        => ECS_COMMAND_i,
      WRITE_ECS_COMMAND_i  => WRITE_ECS_COMMAND_i,
      ecs_clk              => ECS_CLK,
      SRES_i               => SRES_i,
      ecs_cmd_int_ena_o    => ecs_cmd_int_ena,
      ecs_cmd_int_av_i     => ecs_cmd_int_av,
      ecs_cmd_int_data_o   => ecs_cmd_int_data,
      ecs_rpy_int_isRead_o => ecs_rpy_int_isRead,
      ecs_rpy_int_addr_o   => ecs_rpy_int_addr,
      ecs_rpy_int_data_i   => ecs_rpy_int_data,
      new_rpy_i            => new_rpy,
      sol40_sca_rst_o      => sol40_sca_rst,
      reset_sca_o          => reset_sca,
      disable_sca_o        => disable_sca,
      timeout_value_o          => timeout_value,
      sca_link_state_array => sca_link_state_array
   );


ecs_buffer_layer_inst : component ecs_buffer_layer
   port map(
      clk                  => ECS_CLK,
      rst                  => rst,
      ecs_cmd_int_av_o    => ecs_cmd_int_av,
      ecs_cmd_int_ena_i     => ecs_cmd_int_ena,
      ecs_cmd_int_data_i   => ecs_cmd_int_data,
      ecs_rpy_int_isRead_i => ecs_rpy_int_isRead,
      ecs_rpy_int_addr_i   => ecs_rpy_int_addr,
      ecs_rpy_int_av_o     => ecs_rpy_int_av,
      ecs_rpy_int_data_o   => ecs_rpy_int_data,
      new_rpy_o            => new_rpy,
      ecs_rpy_prot_av_o    => ecs_rpy_prot_av,
      ecs_rpy_prot_ena_i   => ecs_rpy_prot_ena,
      ecs_rpy_prot_data_i  => ecs_rpy_prot_data,
      ecs_cmd_prot_ena_i   => ecs_cmd_prot_ena,
      ecs_cmd_prot_av_o    => ecs_cmd_prot_av,
      ecs_cmd_prot_data_o  => ecs_cmd_prot_data
   );

--clk <= txClock40;
rst <= SRES_i or sol40_sca_rst;

protocol_layer_inst : component protocol_layer
   port map(
          clk                 => txClock40,
           rst                 => rst,
           ecs_cmd_prot_ena_o  => ecs_cmd_prot_ena,
           ecs_cmd_prot_av_i   => ecs_cmd_prot_av,
           ecs_cmd_prot_data_i => ecs_cmd_prot_data,
           ecs_rpy_prot_av_i   => ecs_rpy_prot_av,
           ecs_rpy_prot_ena_o  => ecs_rpy_prot_ena,
           ecs_rpy_prot_data_o => ecs_rpy_prot_data,
           timeout_value_i  => timeout_value,
           sca_cmd_data_o      => sca_cmd_data,
           sca_cmd_ena_o       => sca_cmd_ena,
```

```
            sca_cmd_rdy_i   => sca_cmd_rdy,
            sca_rpy_data_i      => sca_rpy_data,
            sca_rpy_prot_rdy_o  => sca_rpy_prot_rdy,
            send_sca_rpy_i      => send_sca_rpy,
            reconnnect_o        => reconnnect);

  mac_layer_inst : component mac_layer
    port map(
      tx_clk                => txClock40,
      rst                   => rst,
      sca_cmd_data_i        => sca_cmd_data,
      sca_cmd_ena_i         => sca_cmd_ena,
      sca_cmd_rdy_o         => sca_cmd_rdy,
      sca_rpy_data_o        => sca_rpy_data,
      sca_rpy_prot_rdy_i    => sca_rpy_prot_rdy,
      sca_rpy_en_o          => send_sca_rpy,
      reconnnect_i          => reconnnect,
      reset_sca_i           => reset_sca,
      disable_sca_i         => disable_sca,
      sca_ch_state_o        => sca_ch_state,
      sca_link_state_array_o => sca_link_state_array,
      tx_sd                 => tx_sd,
      rx_clk                => rxClock40,
      rx_sd                 => rx_sd
    );


end architecture rtl;
```

```
--------------------------------------------------------------------------------
--! @file sca_package.vhd
--! @author Cairo Caplan <cairo.caplan@cern.ch>
--! @date April, 2016
--------------------------------------------------------------------------------

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
use ieee.math_real.all;

use work.Constant_Declaration.all;


package SCA_Package is

   ------------------------------ ECS
     ------------------------------------------------------------------------------------------
   -- From Constant_Declaration at amc40firmware\mini_daq\mini_daq\packages\
   -- BAR0 addresses width
   --constant ECS_address_size       : integer := 21;--19

   -- BAR0 data width
   --constant ECS_data_size          : integer := 32;




     ------------------------------------------------------------------------------------------------------------------------

   --! \name SOL40-SCA configurable Constants
   --! \{
   constant CMD_FIFO_ADDR_WIDTH  :  integer := 0;--1;
   constant SCA_COUNT            :  integer := 17;
   constant Q_COUNT           :  natural := 4;--:= 16;
   constant timeout_packet_value :  natural := (1_000 ms/25 ns );

   shared variable loopback_through_sca_en   :  std_logic := '0';
   --! \}



   --! \name Common used general types
   --! \{
   subtype byte_t is  std_logic_vector(7 downto 0);
   type byte_vector_t is array(integer range <>) of byte_t;
   type std_logic_2d_vector is array(integer range <>, integer range <>) of std_logic;
   type natural_vector_t is array(integer range <>) of natural;
   type natural_2d_array_t is array(integer range <>, integer range <>) of natural;
   type eport_data_array_t is array(natural range<>) of std_logic_vector(15 downto 0);
   --! \}

   --! \name SOL40-SCA Constants
   --! \{
   constant ECS_CONTROL_REGISTER :  byte_t:=  x"00";
   constant ECS_REPLY_ADDRESS    :  byte_t:=  x"04";
   constant ECS_COMMAND_HEADER_0 :  byte_t:=  x"08";
   constant ECS_COMMAND_HEADER_1 :  byte_t:=  x"0C";
```

```
constant ECS_COMMAND_DATA_0      :  byte_t:=  x"10";
constant ECS_COMMAND_DATA_1      :  byte_t:=  x"14";
constant ECS_COMMAND_DATA_2      :  byte_t:=  x"18";
constant ECS_COMMAND_DATA_3      :  byte_t:=  x"1C";
constant ECS_REPLY_HEADER_0      :  byte_t:=  x"20";
constant ECS_REPLY_HEADER_1      :  byte_t:=  x"24";
constant ECS_REPLY_DATA_0     :  byte_t:=  x"28";
constant ECS_REPLY_DATA_1     :  byte_t:=  x"2C";
constant ECS_REPLY_DATA_2     :  byte_t:=  x"30";
constant ECS_REPLY_DATA_3     :  byte_t:=  x"34";
constant ECS_RESET_SCA_0      :  byte_t:=  x"40";
constant ECS_RESET_SCA_1      :  byte_t:=  x"44";
constant ECS_DISABLE_SCA_0    :  byte_t:=  x"50";
constant ECS_DISABLE_SCA_1    :  byte_t:=  x"54";
constant ECS_TIMEOUT       :  byte_t:=  x"60";

--! DON'T CHANGE GBT-SCA SPECIFIC (CONSTANT)
constant CH_BY_SCA_COUNT         :  natural := 22;


--! Number of replies that could be contained on the reply buffer = rpy_list
constant RPY_PACK_COUNT       :  integer := 5;--50;
constant RPY_BUF_ADDR_WIDTH   :  integer := integer(ceil(log2(real(RPY_PACK_COUNT))));

constant PAYLOAD_DATA_MAX_LEN :  natural  := 256/8; -- 256 bits in bytes

--! TR numbers associated to interrupts, can not to be used by a command
constant interrupts_tr : byte_vector_t(0 to 1):= (x"00",x"ff");
--------------

--! This is the acknowledge field in the reply packet. In case of an error
--! has encountered it contains the correspondent error code otherwise
--! returns 0x00. In particular:
type ERR_mask_enum is(
   nothing,                   --bit 0
   invalid_channel_request,         --bit 1
   invalid_command_request,         --bit 2
   invalid_transaction_number_request, --bit 3
   invalid_request_length,          --bit 4
   channel_disabled,             --bit 5
   channel_busy,                 --bit 6
   command_en_treatment);           --bit 7

--! \}



--! GBT-SCA Payload Type
type payload_t is
record
   tr             :  byte_t;
   ch             :  byte_t;
   cmd_or_err        :  byte_t;
   len               :  byte_t;
   data           :  std_logic_vector(31 downto 0);
end record;

type payload_vector_t is array(integer range <>) of payload_t;
--type payload_2d_vector_t is array(integer range <>, integer range <>) of payload_t;
```

```
type payload_2d_vector_t is array(integer range <>, integer range <>) of payload_t;

type payload_ch_array_t    is array(natural range<>) of payload_vector_t(0 to CH_BY_SCA_COUNT - 1);
type std_logic_ch_array_t  is array(natural range<>) of std_logic_vector(0 to CH_BY_SCA_COUNT - 1);
type elink_ch_array_t   is array(natural range<>) of std_logic_vector(1 downto 0);

type ch_state_t is record
      busy        :  std_logic;
      last_tr_sent   :  byte_t;
end record ch_state_t;

type ch_state_vector_t  is array(0 to CH_BY_SCA_COUNT-1) of ch_state_t;
type sca_ch_state_vector_t is array(natural range <>) of ch_state_vector_t;


type payload_by_sca_t is array(0 to SCA_COUNT-1) of payload_t;
type ch_state_vector_by_sca_t is array(0 to SCA_COUNT-1) of ch_state_vector_t;
type std_logic_by_sca is array(0 to SCA_COUNT-1) of std_logic;



type payload_prot_t is
record
   --cmd_or_err         :  byte_t;
   len               :  byte_t;
   data           :  std_logic_vector(31 downto 0);
end record;


type payload_prot_vector_t is array(integer range <>) of payload_prot_t;

type prot_state_t is (IDLE, GET_SCA_CMD, SEND_SCA_CMD, WAIT_SCA_RPY, ECS_RPY_RDY);

type prot_state_vector_t is array(natural range<>) of prot_state_t;



--subtype sca_time_t is natural range 0 to (1_000_000 ns / 25 ns)-1;
-- 2*26 clock cycles = 2*26*25 ns = 1_677_721_600 ns  ~=1.6 s
subtype sca_time_t is integer range 0 to (2**26)-1;
type sca_time_array_t is array (natural range<>) of sca_time_t;


-- Types used by the protocol drivers on the protocol layer


type prot_cmd_t is record
   sca            :  natural range 0 to SCA_COUNT-1;
   tr          :  byte_t;
   ch          :  byte_t;
   ecs_cmd        :  byte_t;
   ecs_len        :  byte_t;
   sca_cmds    :  byte_vector_t(0 to 7);
   sca_cmds_data  :  payload_prot_vector_t(0 to 7);
   sca_rpys_data  :  payload_prot_vector_t(0 to 7);
   cmd_count      :  natural range 0 to 7;
   err            :  byte_t;
end record prot_cmd_t;
type prot_cmd_vector_t is array(natural range<>) of prot_cmd_t;
```

```
   type cmd_count_vector_t is array(natural range<>) of natural range 0 to 7;

   constant MAX_RETRANSMISSIONS  : natural := 3;
   type command_queue_t is record
      prot_cmd    :  prot_cmd_vector_t(0 to Q_COUNT-1);
      prot_state     :  prot_state_vector_t(0 to Q_COUNT-1);
      counter        :  cmd_count_vector_t(0 to  Q_COUNT-1);
      expiration_time   :  sca_time_array_t(0 to  Q_COUNT-1);
      retransm_nr    :  natural range 0 to MAX_RETRANSMISSIONS- 1;
   end record command_queue_t;

-- type payload_len_t is
-- record
--    cmd   :  std_logic_vector(4 downto 0);
--    rpy   :  std_logic_vector(4 downto 0);
--    tr :  byte_t;
-- end record;

   --! SCA Channel table
   type sca_ch_enum is(
      sca_controller,
      master_spi,
      master_gpio,
      master_i2c_0,
      master_i2c_1,
      master_i2c_2,
      master_i2c_3,
      master_i2c_4,
      master_i2c_5,
      master_i2c_6,
      master_i2c_7,
      master_i2c_8,
      master_i2c_9,
      master_i2c_10,
      master_i2c_11,
      master_i2c_12,
      master_i2c_13,
      master_i2c_14,
      master_i2c_15,
      master_jtag,
      adc,
      dac);

   type sca_ch_table_t is array(sca_ch_enum'left to sca_ch_enum'right) of byte_t;
   constant sca_ch_table : sca_ch_table_t := (
      sca_controller => std_logic_vector( to_unsigned(0, 8) ),
      master_spi     => std_logic_vector( to_unsigned(1, 8) ),
      master_gpio    => std_logic_vector( to_unsigned(2, 8) ),
      master_i2c_0   => std_logic_vector( to_unsigned(3, 8) ),
      master_i2c_1   => std_logic_vector( to_unsigned(4, 8) ),
      master_i2c_2   => std_logic_vector( to_unsigned(5, 8) ),
      master_i2c_3   => std_logic_vector( to_unsigned(6, 8) ),
      master_i2c_4   => std_logic_vector( to_unsigned(7, 8) ),
      master_i2c_5   => std_logic_vector( to_unsigned(8, 8) ),
      master_i2c_6   => std_logic_vector( to_unsigned(9, 8) ),
      master_i2c_7   => std_logic_vector( to_unsigned(10,8) ),
      master_i2c_8   => std_logic_vector( to_unsigned(11,8) ),
      master_i2c_9   => std_logic_vector( to_unsigned(12,8) ),
```

```
   master_i2c_10  => std_logic_vector( to_unsigned(13,8) ),
   master_i2c_11  => std_logic_vector( to_unsigned(14,8) ),
   master_i2c_12  => std_logic_vector( to_unsigned(15,8) ),
   master_i2c_13  => std_logic_vector( to_unsigned(16,8) ),
   master_i2c_14  => std_logic_vector( to_unsigned(17,8) ),
   master_i2c_15  => std_logic_vector( to_unsigned(18,8) ),
   master_jtag    => std_logic_vector( to_unsigned(19,8) ),
   adc         =>    std_logic_vector( to_unsigned(20,8) ),
   dac         =>       std_logic_vector( to_unsigned(21,8) )
);



type activated_channels_t is array(0 to SCA_COUNT-1) of std_logic_vector(CH_BY_SCA_COUNT-1 downto 0);


type packet_word_t is
record
   sca      :  std_logic_vector(4 downto 0);
   payload  :  payload_t;
end record;


--! \}

------------------------------------------------------------------------------------------
type ecs_packet_t is
record
   --gbt_nr          :  byte_t;
   sca_nr            :  byte_t;
   sca_ch            :  byte_t;-- range channels_enum'left to channels_enum'right;
   ecs_cmd           :  byte_t;
   err               :  byte_t;
   len               :  byte_t;
   protocol_specific :  std_logic_vector(15 downto 0);
   data           :  std_logic_vector(127 downto 0);
end record;


type ecs_addr_packet_t is
record
   sca_nr            :  byte_t;
   sca_ch            :  byte_t;
end record;

type ecs_packet_array_t is array(natural range<>) of ecs_packet_t;

type payload_stream_t is
record
   --! CH + TR + CMD + DATA (Max 256 bits)
   bit_stream     :  std_logic_vector(6*8 + 256 -1  downto 0);
   -- mininum of 3 byte and a maximum of 32 + 3
   effective_len  :  natural range 3 to 35;

end record;

type payload_stream_array_t is array(natural range <>) of payload_stream_t;

------------------------------------------------------------------
```

```
--! SCA controller commands dictionary

type sca_cmd_enum is (
   CRA_wrt, --! Write the SLVS transmit current level setting
   CRA_rea, --! Read the SLVS transmit current level setting
   CRB_wrt, --! Write bits 0 to 7 of the CH_EN register
   CRB_rea, --! Read bits 0 to 7 of the CH_EN register
   CRC_wrt, --! Write bits 8 to 15 of the CH_EN register
   CRC_rea, --! Read bits 8 to 15 of the CH_EN register
   CRD_wrt, --! Write bits 16 to 21 of the CH_EN register
   CRD_rea   --! Read bits 16 to 21 of the CH_EN register
);




type sca_cmd_table_t is array(sca_cmd_enum'left to sca_cmd_enum'right) of byte_t;

constant sca_cmd_table : sca_cmd_table_t :=
   (
   CRA_wrt => x"00",
   CRA_rea => x"01",
   CRB_wrt => x"02",
   CRB_rea => x"03",
   CRC_wrt => x"04",
   CRC_rea => x"05",
   CRD_wrt => x"06",
   CRD_rea => x"07"
);

type ecs_sca_cmd_enum is(
   ECS_CRA_wrt, --! Write the SLVS transmit current level setting
   ECS_CRA_rea, --! Read the SLVS transmit current level setting
   ECS_CRB_wrt, --! Write bits 0 to 7 of the CH_EN register
   ECS_CRB_rea, --! Read bits 0 to 7 of the CH_EN register
   ECS_CRC_wrt, --! Write bits 8 to 15 of the CH_EN register
   ECS_CRC_rea, --! Read bits 8 to 15 of the CH_EN register
   ECS_CRD_wrt, --! Write bits 16 to 21 of the CH_EN register
   ECS_CRD_rea, --! Read bits 16 to 21 of the CH_EN register
   ECS_CRA_wrt_rea, --! Write AND READ the SLVS transmit current level setting
   ECS_CRB_wrt_rea, --! Write AND READ bits 0 to 7 of the CH_EN register
   ECS_CRC_wrt_rea, --! Write AND READ bits 8 to 15 of the CH_EN register
   ECS_CRD_wrt_rea, --! Write AND READ bits 16 to 21 of the CH_EN register
   INVALID
);

type ecs_sca_cmd_table_t is array(ecs_sca_cmd_enum'left to ecs_sca_cmd_enum'right) of byte_t;

constant ecs_sca_cmd_table : ecs_sca_cmd_table_t :=
   (
   ECS_CRA_wrt => x"00",
   ECS_CRA_rea => x"01",
   ECS_CRB_wrt => x"02",
   ECS_CRB_rea => x"03",
   ECS_CRC_wrt => x"04",
   ECS_CRC_rea => x"05",
   ECS_CRD_wrt => x"06",
   ECS_CRD_rea => x"07",
   ECS_CRA_wrt_rea => x"08",
   ECS_CRB_wrt_rea => x"09",
```

```
      ECS_CRC_wrt_rea => x"0a",
      ECS_CRD_wrt_rea => x"0b",
      INVALID => x"FF"
   );




------------------------------------------------------------------
--! I2C commands dictionary

   type i2c_cmd_enum is (
      --! Start single byte write transmission 7 bit addr.
      I2C_S_7B_W,--0x82
      --! Start single byte read transmission 7-bit addr.
      I2C_S_7B_R,--0x86
      --! Start single byte write transmission 10-bit addr.
      I2C_S_10B_W,--0x8A
      --! Start single byte read transmission 10-bit addr.
      I2C_S_10B_R,--0x8E
      --! Start multi byte write transmission 7-bit addr.
      I2C_M_7B_W,--0xDA
      --! Start multi byte read transmission 7-bit addr.
      I2C_M_7B_R,--0xDE
      --! Start multi byte write transmission 10-bit addr.
      I2C_M_10B_W,--0x82
      --! Start multi byte read transmission 10-bit addr.
      I2C_M_10B_R,--0xE6
      --! Read ï¿½ modify ï¿½ write (AND with MASK reg.)
      I2C_RMW_AND,--0xC2
      --! Read ï¿½ modify ï¿½ write (OR with MASK reg.)
      I2C_RMW_OR,--0xC6
      --! Read ï¿½ modify ï¿½ write (XOR with MASK reg.)
      I2C_RMW_XOR,--0xCA

      ----Data register access
      --!Write the DATA TRANSMIT register bits 0 to 31
      I2C_W_DATA0,
      --!Read the DATA TRANSMIT register bits 0 to 31
      I2C_R_DATA0,
      --!Write the DATA TRANSMIT register bits 32 to 63
      I2C_W_DATA1,
      --!Read the DATA TRANSMIT register bits 32 to 63
      I2C_R_DATA1,
      --!Write the DATA TRANSMIT register bits 64 to 95
      I2C_W_DATA2,
      --!Read the DATA TRANSMIT register bits 64 to 95
      I2C_R_DATA2,
      --!Write the DATA TRANSMIT register bits 96 to 127
      I2C_W_DATA3,
      --!Read the DATA TRANSMIT register bits 96 to 127
      I2C_R_DATA3,

      --Control registers:
      --!Write the CONTROL register
      I2C_W_CTRL,
      --!Read the CONTROL register
      I2C_R_CTRL,
      --!Write the MASK register
```

```
    I2C_W_MSK,
    --!Read the MASK register
    I2C_R_MSK,
    --!Read the STATUS register
    I2C_R_STR


);

type i2c_cmd_table_t is array(i2c_cmd_enum'left to i2c_cmd_enum'right) of byte_t;

constant i2c_cmd_table : i2c_cmd_table_t :=(
    --! Start single byte write transmission 7 bit addr.
    I2C_S_7B_W=>x"82",
    --! Start single byte read transmission 7-bit addr.
    I2C_S_7B_R=>x"86",
    --! Start single byte write transmission 10-bit addr.
    I2C_S_10B_W=>x"8A",
    --! Start single byte read transmission 10-bit addr.
    I2C_S_10B_R=>x"8E",
    --! Start multi byte write transmission 7-bit addr.
    I2C_M_7B_W=>x"DA",
    --! Start multi byte read transmission 7-bit addr.
    I2C_M_7B_R=>x"DE",
    --! Start multi byte write transmission 10-bit addr.
    I2C_M_10B_W=>x"E2",
    --! Start multi byte read transmission 10-bit addr.
    I2C_M_10B_R=>x"E6",
    --! Read ï¿½ modify ï¿½ write (AND with MASK reg.)
    I2C_RMW_AND=>x"C2",
    --! Read ï¿½ modify ï¿½ write (OR with MASK reg.)
    I2C_RMW_OR=>x"C6",
    --! Read ï¿½ modify ï¿½ write (XOR with MASK reg.)
    I2C_RMW_XOR=>x"CA",

    ----Data register access
    --!Write the DATA TRANSMIT register bits 0 to 31
    I2C_W_DATA0 => x"40",
    --!Read the DATA TRANSMIT register bits 0 to 31
    I2C_R_DATA0 => x"71",
    --!Write the DATA TRANSMIT register bits 32 to 63
    I2C_W_DATA1 => x"50",
    --!Read the DATA TRANSMIT register bits 32 to 63
    I2C_R_DATA1 => x"61",
    --!Write the DATA TRANSMIT register bits 64 to 95
    I2C_W_DATA2 => x"60",
    --!Read the DATA TRANSMIT register bits 64 to 95
    I2C_R_DATA2 => x"51",
    --!Write the DATA TRANSMIT register bits 96 to 127
    I2C_W_DATA3 => x"70",
    --!Read the DATA TRANSMIT register bits 96 to 127
    I2C_R_DATA3 => x"41",

    --Control registers:
    --!Write the CONTROL register
    I2C_W_CTRL => x"30",
    --!Read the CONTROL register
    I2C_R_CTRL => x"31",
    --!Write the MASK register
    I2C_W_MSK => x"20",
```

```
    --!Read the MASK register
    I2C_R_MSK => x"21",
    --!Read the STATUS register
    I2C_R_STR => x"11"

);

type ecs_i2c_cmd_enum is (
    --! Start single byte write transmission 7 bit addr.
    ECS_I2C_S_7B_W,--0x82
    --! Start single byte read transmission 7-bit addr.
    ECS_I2C_S_7B_R,--0x86
    --! Start single byte write transmission 10-bit addr.
    ECS_I2C_S_10B_W,--0x8A
    --! Start single byte read transmission 10-bit addr.
    ECS_I2C_S_10B_R,--0x8E
    --! Start multi byte write transmission 7-bit addr.
    ECS_I2C_M_7B_W,--0xDA
    --! Start multi byte read transmission 7-bit addr.
    ECS_I2C_M_7B_R,--0xDE
    --! Start multi byte write transmission 10-bit addr.
    ECS_I2C_M_10B_W,--0x82
    --! Start multi byte read transmission 10-bit addr.
    ECS_I2C_M_10B_R,--0xE6

    --! Read and modify and write (AND with MASK reg.)
    ECS_I2C_RMW_AND,
    --! Read and modify and write (OR with MASK reg.)
    ECS_I2C_RMW_OR,
    --! Read and modify and write (XOR with MASK reg.)
    ECS_I2C_RMW_XOR,

    --Data register access
    --!Write the DATA TRANSMIT register bits 0 to 31
    ECS_I2C_W_DATA0,
    --!Read the DATA TRANSMIT register bits 0 to 31
    ECS_I2C_R_DATA0,
    --!Write the DATA TRANSMIT register bits 32 to 63
    ECS_I2C_W_DATA1,
    --!Read the DATA TRANSMIT register bits 32 to 63
    ECS_I2C_R_DATA1,
    --!Write the DATA TRANSMIT register bits 64 to 95
    ECS_I2C_W_DATA2,
    --!Read the DATA TRANSMIT register bits 64 to 95
    ECS_I2C_R_DATA2,
    --!Write the DATA TRANSMIT register bits 96 to 127
    ECS_I2C_W_DATA3,
    --!Read the DATA TRANSMIT register bits 96 to 127
    ECS_I2C_R_DATA3,


    --Control registers:
    --!Write the CONTROL register
    ECS_I2C_W_CTRL,
    --!Read the CONTROL register
    ECS_I2C_R_CTRL,
    --!Write the MASK register
    ECS_I2C_W_MSK,
    --!Read the MASK register
    ECS_I2C_R_MSK,
```

```
    --!Read the STATUS register
    ECS_I2C_R_STR,

    --! Start write transmission 7 bit addr.
    ECS_I2C_WRITE,
    --! Start read transmission 7-bit addr.
    ECS_I2C_READ,
    --! Start write transmission 10-bit addr.
    ECS_I2C_WRITE_EXT,
    --! Start read transmission 10-bit addr.
    ECS_I2C_READ_EXT,

    --Data register _write_and_read_ commands
    --!Write the DATA TRANSMIT register bits 0 to 31
    ECS_I2C_W_R_DATA0,
    --!Write the DATA TRANSMIT register bits 32 to 63
    ECS_I2C_W_R_DATA1,
    --!Write the DATA TRANSMIT register bits 64 to 95
    ECS_I2C_W_R_DATA2,
    --!Write the DATA TRANSMIT register bits 96 to 127
    ECS_I2C_W_R_DATA3,

    --!Write AND READ the CONTROL register
    ECS_I2C_W_R_CTRL,
    --!Write AND READ the MASK register
    ECS_I2C_W_R_MSK,
    INVALID
);

type ecs_i2c_table_t is array(ecs_i2c_cmd_enum'left to ecs_i2c_cmd_enum'right) of byte_t;

constant ecs_i2c_table : ecs_i2c_table_t :=(

    --! Start single byte write transmission 7 bit addr.
    ECS_I2C_S_7B_W=>x"82",
    --! Start single byte read transmission 7-bit addr.
    ECS_I2C_S_7B_R=>x"86",
    --! Start single byte write transmission 10-bit addr.
    ECS_I2C_S_10B_W=>x"8A",
    --! Start single byte read transmission 10-bit addr.
    ECS_I2C_S_10B_R=>x"8E",
    --! Start multi byte write transmission 7-bit addr.
    ECS_I2C_M_7B_W=>x"DA",
    --! Start multi byte read transmission 7-bit addr.
    ECS_I2C_M_7B_R=>x"DE",
    --! Start multi byte write transmission 10-bit addr.
    ECS_I2C_M_10B_W=>x"E2",
    --! Start multi byte read transmission 10-bit addr.
    ECS_I2C_M_10B_R=>x"E6",
    --! Read ï¿½ modify ï¿½ write (AND with MASK reg.)
    ECS_I2C_RMW_AND=>x"C2",
    --! Read ï¿½ modify ï¿½ write (OR with MASK reg.)
    ECS_I2C_RMW_OR=>x"C6",
    --! Read ï¿½ modify ï¿½ write (XOR with MASK reg.)
    ECS_I2C_RMW_XOR=>x"CA",

    ----Data register access
    --!Write the DATA TRANSMIT register bits 0 to 31
    ECS_I2C_W_DATA0 => x"40",
    --!Read the DATA TRANSMIT register bits 0 to 31
```

```
    ECS_I2C_R_DATA0 => x"41",
    --!Write the DATA TRANSMIT register bits 32 to 63
    ECS_I2C_W_DATA1 => x"50",
    --!Read the DATA TRANSMIT register bits 32 to 63
    ECS_I2C_R_DATA1 => x"51",
    --!Write the DATA TRANSMIT register bits 64 to 95
    ECS_I2C_W_DATA2 => x"60",
    --!Read the DATA TRANSMIT register bits 64 to 95
    ECS_I2C_R_DATA2 => x"61",
    --!Write the DATA TRANSMIT register bits 96 to 127
    ECS_I2C_W_DATA3 => x"70",
    --!Read the DATA TRANSMIT register bits 96 to 127
    ECS_I2C_R_DATA3 => x"71",

    --Control registers:
    --!Write the CONTROL register
    ECS_I2C_W_CTRL => x"30",
    --!Read the CONTROL register
    ECS_I2C_R_CTRL => x"31",
    --!Write the MASK register
    ECS_I2C_W_MSK => x"20",
    --!Read the MASK register
    ECS_I2C_R_MSK => x"21",
    --!Read the STATUS register
    ECS_I2C_R_STR => x"11",


    --! Start write transmission 7 bit addr.
    ECS_I2C_WRITE => x"90",
    --! Start read transmission 7-bit addr.
    ECS_I2C_READ=>x"91",
    --! Start write transmission 10-bit addr.
    ECS_I2C_WRITE_EXT=>x"92",
    --! Start read transmission 10-bit addr.
    ECS_I2C_READ_EXT=>x"93",

    --Data register _write_and_read_ commands
    --!Write the DATA TRANSMIT register bits 0 to 31
    ECS_I2C_W_R_DATA0=>x"94",
    --!Write the DATA TRANSMIT register bits 32 to 63
    ECS_I2C_W_R_DATA1=>x"95",
    --!Write the DATA TRANSMIT register bits 64 to 95
    ECS_I2C_W_R_DATA2=>x"96",
    --!Write the DATA TRANSMIT register bits 96 to 127
    ECS_I2C_W_R_DATA3=>x"97",

    --!Write AND READ the CONTROL register
    ECS_I2C_W_R_CTRL=>x"98",
    --!Write AND READ the MASK register
    ECS_I2C_W_R_MSK=>x"99",

    INVALID => x"FF"

);
-----------------------------------------------------------------
--! ADC commands dictionary

type adc_cmd_enum is (
    ADC_GO,
    ADC_W_INSEL,
```

```
    ADC_R_INSEL,
    ADC_W_CUREN,
    ADC_R_CUREN,
    --
    ADC_rea_DATA_Ofs_Ga,
    ADC_rea_DATA_Ofs,
    ADC_rea_DATA,
    ADC_rea_OFFSET,
    ADC_wrt_CTRL,
    ADC_rea_CTRL,
    ADC_GO_SingleSlope,
    ADC_wrt_GainCalibrReg,
    ADC_rea_GainCalibrReg,
    ADC_wrt_ID,
    ADC_rea_ID
);

type adc_cmd_table_t is array(adc_cmd_enum'left to adc_cmd_enum'right) of byte_t;

constant adc_cmd_table : adc_cmd_table_t :=
                  (
    ADC_GO=>x"B2",
    ADC_W_INSEL=>x"30",
    ADC_R_INSEL=>x"31",
    ADC_W_CUREN=>x"40",
    ADC_R_CUREN=>x"41",
    --
    ADC_rea_DATA_Ofs_Ga=>x"21",
    ADC_rea_DATA_Ofs=>x"A1",
    ADC_rea_DATA=>x"51",
    ADC_rea_OFFSET=>x"61",
    ADC_wrt_CTRL=>x"10",
    ADC_rea_CTRL=>x"11",
    ADC_GO_SingleSlope=>x"02",
    ADC_wrt_GainCalibrReg=>x"70",
    ADC_rea_GainCalibrReg=>x"71",
    ADC_wrt_ID=>x"90",
    ADC_rea_ID=>x"91"
);


type ecs_adc_cmd_enum   is (
    ECS_ADC_GO,
    ECS_ADC_W_INSEL,
    ECS_ADC_R_INSEL,
    ECS_ADC_W_CUREN,
    ECS_ADC_R_CUREN,
    --
    ECS_ADC_rea_DATA_Ofs_Ga,
    ECS_ADC_rea_DATA_Ofs,
    ECS_ADC_rea_DATA,
    ECS_ADC_rea_OFFSET,
    ECS_ADC_wrt_CTRL,
    ECS_ADC_rea_CTRL,
    ECS_ADC_GO_SingleSlope,
    ECS_ADC_wrt_GainCalibrReg,
    ECS_ADC_rea_GainCalibrReg,
    ECS_ADC_wrt_ID,
    ECS_ADC_rea_ID,
    INVALID
```

```
);

type ecs_adc_cmd_table_t is array(ecs_adc_cmd_enum'left to ecs_adc_cmd_enum'right) of byte_t;


constant ecs_adc_cmd_table : ecs_adc_cmd_table_t :=
                  (
   ECS_ADC_GO=>x"B2",
   ECS_ADC_W_INSEL=>x"30",
   ECS_ADC_R_INSEL=>x"31",
   ECS_ADC_W_CUREN=>x"40",
   ECS_ADC_R_CUREN=>x"41",
   --
   ECS_ADC_rea_DATA_Ofs_Ga=>x"21",
   ECS_ADC_rea_DATA_Ofs=>x"A1",
   ECS_ADC_rea_DATA=>x"51",
   ECS_ADC_rea_OFFSET=>x"61",
   ECS_ADC_wrt_CTRL=>x"10",
   ECS_ADC_rea_CTRL=>x"11",
   ECS_ADC_GO_SingleSlope=>x"02",
   ECS_ADC_wrt_GainCalibrReg=>x"70",
   ECS_ADC_rea_GainCalibrReg=>x"71",
   ECS_ADC_wrt_ID=>x"90",
   ECS_ADC_rea_ID=>x"91",
   INVALID=>x"FF"
);

--! DAC commands dictionary

type dac_cmd_enum is (
   DAC_A_wrt,
   DAC_A_read,
   DAC_B_wrt,
   DAC_B_read,
   DAC_C_wrt,
   DAC_C_read,
   DAC_D_wrt,
   DAC_D_read
);


type dac_cmd_table_t is array(dac_cmd_enum'left to dac_cmd_enum'right) of byte_t;

constant dac_cmd_table : dac_cmd_table_t :=
   (
   DAC_A_wrt=>x"10",
   DAC_A_read=>x"11",
   DAC_B_wrt=>x"20",
   DAC_B_read=>x"21",
   DAC_C_wrt=>x"30",
   DAC_C_read=>x"31",
   DAC_D_wrt=>x"40",
   DAC_D_read=>x"41"
);


type ecs_dac_cmd_enum is(
   ECS_DAC_A_wrt,
   ECS_DAC_A_read,
   ECS_DAC_B_wrt,
```

```
    ECS_DAC_B_read,
    ECS_DAC_C_wrt,
    ECS_DAC_C_read,
    ECS_DAC_D_wrt,
    ECS_DAC_D_read,
    INVALID
);

type ecs_dac_cmd_table_t is array(ecs_dac_cmd_enum'left to ecs_dac_cmd_enum'right) of byte_t;

constant ecs_dac_cmd_table : ecs_dac_cmd_table_t :=
    (
    ECS_DAC_A_wrt=>x"10",
    ECS_DAC_A_read=>x"11",
    ECS_DAC_B_wrt=>x"20",
    ECS_DAC_B_read=>x"21",
    ECS_DAC_C_wrt=>x"30",
    ECS_DAC_C_read=>x"31",
    ECS_DAC_D_wrt=>x"40",
    ECS_DAC_D_read=>x"41",
    INVALID => x"FF"
);


--! SPI commands dictionary-------------------------------------------------
type spi_cmd_enum is (
    SPI_GO,
    SPI_wrt_CTRL,
    SPI_rea_CTRL,
    SPI_wrt_SS,
    SPI_rea_SS,
    SPI_wrt_DIV,SPI_rea_DIV,
    SPI_MOSI_wrt_Tx0,
    SPI_MOSI_rea_Tx0,
    SPI_MOSI_wrt_Tx1,
    SPI_MOSI_rea_Tx1,
    SPI_MOSI_wrt_Tx2,
    SPI_MOSI_rea_Tx2,
    SPI_MOSI_wrt_Tx3,
    SPI_MOSI_rea_Tx3,
    SPI_MISO_rea_Rx0,
    SPI_MISO_rea_Rx1,
    SPI_MISO_rea_Rx2,
    SPI_MISO_rea_Rx3
);


type spi_cmd_table_t is array(spi_cmd_enum'left to spi_cmd_enum'right) of byte_t;

constant spi_cmd_table  :  spi_cmd_table_t:=(
    SPI_GO=>x"72",
    SPI_wrt_CTRL=>x"40",
    SPI_rea_CTRL=>x"41",
    SPI_wrt_SS=>x"60",
    SPI_rea_SS=>x"61",
    SPI_wrt_DIV=>x"50",
    SPI_rea_DIV=>x"51",
    --
    SPI_MOSI_wrt_Tx0=>x"00",
    SPI_MOSI_rea_Tx0=>x"01",
```

```
    SPI_MOSI_wrt_Tx1=>x"10",
    SPI_MOSI_rea_Tx1=>x"11",
    SPI_MOSI_wrt_Tx2=>x"20",
    SPI_MOSI_rea_Tx2=>x"21",
    SPI_MOSI_wrt_Tx3=>x"30",
    SPI_MOSI_rea_Tx3=>x"31",
    --
    SPI_MISO_rea_Rx0=>x"01",
    SPI_MISO_rea_Rx1=>x"11",
    SPI_MISO_rea_Rx2=>x"21",
    SPI_MISO_rea_Rx3=>x"31"
);


type ecs_spi_cmd_enum is(
    ECS_SPI_GO,
    ECS_SPI_W_CTRL,
    ECS_SPI_R_CTRL,
    ECS_SPI_W_SS,
    ECS_SPI_R_SS,
    ECS_SPI_W_FREQ,
    ECS_SPI_R_FREQ,
    ECS_SPI_MOSI_wrt_Tx0,
    ECS_SPI_MOSI_rea_Tx0,
    ECS_SPI_MOSI_wrt_Tx1,
    ECS_SPI_MOSI_rea_Tx1,
    ECS_SPI_MOSI_wrt_Tx2,
    ECS_SPI_MOSI_rea_Tx2,
    ECS_SPI_MOSI_wrt_Tx3,
    ECS_SPI_MOSI_rea_Tx3,
    ECS_SPI_MISO_rea_Rx0,
    ECS_SPI_MISO_rea_Rx1,
    ECS_SPI_MISO_rea_Rx2,
    ECS_SPI_MISO_rea_Rx3,
    --
    ECS_SPI_WRITE_TX_REG,
    ECS_SPI_READ_RX_REG,
    INVALID
);

type ecs_spi_cmd_table_t is array(ecs_spi_cmd_enum'left to ecs_spi_cmd_enum'right) of byte_t;
constant ecs_spi_cmd_table : ecs_spi_cmd_table_t :=(
    --ECS_SPI_GO => x"40",
    ECS_SPI_GO=>x"72",
    ECS_SPI_W_CTRL=>x"40",
    ECS_SPI_R_CTRL=>x"41",
    ECS_SPI_W_SS=>x"60",
    ECS_SPI_R_SS=>x"61",
    ECS_SPI_W_FREQ=>x"50",
    ECS_SPI_R_FREQ=>x"51",
    --
    ECS_SPI_MOSI_wrt_Tx0=>x"00",
    ECS_SPI_MOSI_rea_Tx0=>x"01",
    ECS_SPI_MOSI_wrt_Tx1=>x"10",
    ECS_SPI_MOSI_rea_Tx1=>x"11",
    ECS_SPI_MOSI_wrt_Tx2=>x"20",
    ECS_SPI_MOSI_rea_Tx2=>x"21",
    ECS_SPI_MOSI_wrt_Tx3=>x"30",
    ECS_SPI_MOSI_rea_Tx3=>x"31",
    --
```

```
   ECS_SPI_MISO_rea_Rx0=>x"01",
   ECS_SPI_MISO_rea_Rx1=>x"11",
   ECS_SPI_MISO_rea_Rx2=>x"21",
   ECS_SPI_MISO_rea_Rx3=>x"31",
   --
   ECS_SPI_WRITE_TX_REG => x"48",
   ECS_SPI_READ_RX_REG => x"49",
   INVALID  => x"FF"
);




--! JTAG commands dictionary-----------------------------------------------
type jtag_cmd_enum is (
--Configuration registers:
   JTAG_GO,JTAG_wrt_CTRL,JTAG_rea_CTRL,JTAG_wrt_DIV,JTAG_rea_DIV,
   --TMS buffer registers:
   JTAG_TMS_wrt_Tx0,JTAG_TMS_rea_Tx0,JTAG_TMS_wrt_Tx1,JTAG_TMS_rea_Tx1,JTAG_TMS_wrt_Tx2,
   JTAG_TMS_rea_Tx2,JTAG_TMS_wrt_Tx3,JTAG_TMS_rea_Tx3,
   --TDO buffer registers:
   JTAG_TDO_wrt_Tx0,JTAG_TDO_rea_Tx0,JTAG_TDO_wrt_Tx1,JTAG_TDO_rea_Tx1,
   JTAG_TDO_wrt_Tx2,JTAG_TDO_rea_Tx2,JTAG_TDO_wrt_Tx3,JTAG_TDO_rea_Tx3,
   --TDI buffer registers:
   JTAG_TDI_rea_Rx0, JTAG_TDI_rea_Rx1, JTAG_TDI_rea_Rx2, JTAG_TDI_rea_Rx3
);

type jtag_cmd_table_t is array(jtag_cmd_enum'left to jtag_cmd_enum'right) of byte_t;

constant jtag_cmd_table : jtag_cmd_table_t := (
   JTAG_GO=>x"A2",
   JTAG_wrt_CTRL=>x"80",
   JTAG_rea_CTRL=>x"81",
   JTAG_wrt_DIV=>x"90",
   JTAG_rea_DIV=>x"91",
   --
   JTAG_TMS_wrt_Tx0=>x"40",
   JTAG_TMS_rea_Tx0=>x"41",
   JTAG_TMS_wrt_Tx1=>x"50",
   JTAG_TMS_rea_Tx1=>x"51",
   JTAG_TMS_wrt_Tx2=>x"60",
   JTAG_TMS_rea_Tx2=>x"61",
   JTAG_TMS_wrt_Tx3=>x"70",
   JTAG_TMS_rea_Tx3=>x"71",
   --
   JTAG_TDO_wrt_Tx0=>x"00",
   JTAG_TDO_rea_Tx0=>x"01",
   JTAG_TDO_wrt_Tx1=>x"10",
   JTAG_TDO_rea_Tx1=>x"11",
   JTAG_TDO_wrt_Tx2=>x"20",
   JTAG_TDO_rea_Tx2=>x"21",
   JTAG_TDO_wrt_Tx3=>x"30",
   JTAG_TDO_rea_Tx3=>x"31",
   --
   JTAG_TDI_rea_Rx0=>x"01",
   JTAG_TDI_rea_Rx1=>x"11",
   JTAG_TDI_rea_Rx2=>x"21",
   JTAG_TDI_rea_Rx3=>x"31"

);
```

```
type ecs_jtag_cmd_enum is(
   ECS_JTAG_GO,ECS_JTAG_wrt_CTRL,ECS_JTAG_rea_CTRL,ECS_JTAG_wrt_DIV,ECS_JTAG_rea_DIV,
   --TMS buffer registers:

  ECS_JTAG_TMS_wrt_Tx0,ECS_JTAG_TMS_rea_Tx0,ECS_JTAG_TMS_wrt_Tx1,ECS_JTAG_TMS_rea_Tx1,ECS_JTAG_TMS_wrt_Tx2,
   ECS_JTAG_TMS_rea_Tx2,ECS_JTAG_TMS_wrt_Tx3,ECS_JTAG_TMS_rea_Tx3,
   --TDO buffer registers:
   ECS_JTAG_TDO_wrt_Tx0,ECS_JTAG_TDO_rea_Tx0,ECS_JTAG_TDO_wrt_Tx1,ECS_JTAG_TDO_rea_Tx1,
   ECS_JTAG_TDO_wrt_Tx2,ECS_JTAG_TDO_rea_Tx2,ECS_JTAG_TDO_wrt_Tx3,ECS_JTAG_TDO_rea_Tx3,
   --TDI buffer registers:
   ECS_JTAG_TDI_rea_Rx0, ECS_JTAG_TDI_rea_Rx1, ECS_JTAG_TDI_rea_Rx2, ECS_JTAG_TDI_rea_Rx3,

   --
   ECS_JTAG_WRITE_TMS,
   ECS_JTAG_READ_TMS,
   ECS_JTAG_WRITE_TDO,
   ECS_JTAG_READ_TDO,
   INVALID
);


type ecs_jtag_cmd_table_t is array(ecs_jtag_cmd_enum'left to ecs_jtag_cmd_enum'right) of byte_t;
constant ecs_jtag_cmd_table : ecs_jtag_cmd_table_t :=(
   ECS_JTAG_GO=>x"A2",
   ECS_JTAG_wrt_CTRL=>x"80",
   ECS_JTAG_rea_CTRL=>x"81",
   ECS_JTAG_wrt_DIV=>x"90",
   ECS_JTAG_rea_DIV=>x"91",
   --
   ECS_JTAG_TMS_wrt_Tx0=>x"40",
   ECS_JTAG_TMS_rea_Tx0=>x"41",
   ECS_JTAG_TMS_wrt_Tx1=>x"50",
   ECS_JTAG_TMS_rea_Tx1=>x"51",
   ECS_JTAG_TMS_wrt_Tx2=>x"60",
   ECS_JTAG_TMS_rea_Tx2=>x"61",
   ECS_JTAG_TMS_wrt_Tx3=>x"70",
   ECS_JTAG_TMS_rea_Tx3=>x"71",
   --
   ECS_JTAG_TDO_wrt_Tx0=>x"00",
   ECS_JTAG_TDO_rea_Tx0=>x"01",
   ECS_JTAG_TDO_wrt_Tx1=>x"10",
   ECS_JTAG_TDO_rea_Tx1=>x"11",
   ECS_JTAG_TDO_wrt_Tx2=>x"20",
   ECS_JTAG_TDO_rea_Tx2=>x"21",
   ECS_JTAG_TDO_wrt_Tx3=>x"30",
   ECS_JTAG_TDO_rea_Tx3=>x"31",
   --
   ECS_JTAG_TDI_rea_Rx0=>x"01",
   ECS_JTAG_TDI_rea_Rx1=>x"11",
   ECS_JTAG_TDI_rea_Rx2=>x"21",
   ECS_JTAG_TDI_rea_Rx3=>x"31",
   --
   ECS_JTAG_WRITE_TMS => x"52",
   ECS_JTAG_READ_TMS => x"53",
   ECS_JTAG_WRITE_TDO => x"54",
   ECS_JTAG_READ_TDO => x"55",
   INVALID => x"FF"
```

```
);


--! GPIO commands dictionary ------------------------------------------------

type gpio_cmd_enum is(
   --Read the DATA_IN register
   GPIO_REA_IN_DATA,
   --Read/Write DATA_OUT register
   GPIO_SET_OUT_DATA, GPIO_REA_OUT_DATA,
   --Read/Write IO_DIRECTION register
   GPIO_SET_IO_DIR, GPIO_REA_IO_DIR,
   --Read/Write INTERRUPT_ENABLE register
   GPIO_SET_INTEN, GPIO_REA_INTEN,
   --Read/Write INTERRUPT_ENABLE register
   GPIO_SET_TRIG, GPIO_REA_TRIG,
   --Read/Write GLOBAL INTERRUPT ENABLE register
   GPIO_SET_GIE, GPIO_REA_GIE,
   --Read/Write INTERRUPT register
   GPIO_SET_INT, GPIO_REA_INT,
   --Read/Write CLK_SEL register
   GPIO_SET_CLK_SEL, GPIO_REA_CLK_SEL,
   --Read/Write CLK_EDGE register
   GPIO_SET_EDG, GPIO_REA_EDG
);

type gpio_cmd_table_t is array(gpio_cmd_enum'left to gpio_cmd_enum'right) of byte_t;
constant gpio_cmd_table : gpio_cmd_table_t :=(
   GPIO_REA_IN_DATA=>x"01",
   GPIO_SET_OUT_DATA=>x"10",
   GPIO_REA_OUT_DATA=>x"11",
   GPIO_SET_IO_DIR=>x"20",
   GPIO_REA_IO_DIR=>x"21",
   GPIO_SET_INTEN=>x"20",
   GPIO_REA_INTEN=>x"21",
   GPIO_SET_TRIG=>x"30",
   GPIO_REA_TRIG=>x"31",
   GPIO_SET_GIE=>x"60",
   GPIO_REA_GIE=>x"61",
   GPIO_SET_INT=>x"70",
   GPIO_REA_INT=>x"71",
   GPIO_SET_CLK_SEL=>x"80",
   GPIO_REA_CLK_SEL=>x"81",
   GPIO_SET_EDG=>x"90",
   GPIO_REA_EDG=>x"91"
);


type ecs_gpio_cmd_enum is(
   --Read the DATA_IN register
   ECS_GPIO_REA_IN_DATA,
   --Read/Write DATA_OUT register
   ECS_GPIO_SET_OUT_DATA, ECS_GPIO_REA_OUT_DATA,
   --Read/Write IO_DIRECTION register
   ECS_GPIO_SET_IO_DIR, ECS_GPIO_REA_IO_DIR,
   --Read/Write INTERRUPT_ENABLE register
   ECS_GPIO_SET_INTEN, ECS_GPIO_REA_INTEN,
   --Read/Write INTERRUPT_ENABLE register
   ECS_GPIO_SET_TRIG, ECS_GPIO_REA_TRIG,
   --Read/Write GLOBAL INTERRUPT ENABLE register
```

```
   ECS_GPIO_SET_GIE, ECS_GPIO_REA_GIE,
   --Read/Write INTERRUPT register
   ECS_GPIO_SET_INT, ECS_GPIO_REA_INT,
   --Read/Write CLK_SEL register
   ECS_GPIO_SET_CLK_SEL, ECS_GPIO_REA_CLK_SEL,
   --Read/Write CLK_EDGE register
   ECS_GPIO_SET_EDG, ECS_GPIO_REA_EDG,
   INVALID
);

type ecs_gpio_cmd_table_t is array(ecs_gpio_cmd_enum'left to ecs_gpio_cmd_enum'right) of byte_t;
constant ecs_gpio_cmd_table : ecs_gpio_cmd_table_t :=(
   ECS_GPIO_REA_IN_DATA=>x"01",
   ECS_GPIO_SET_OUT_DATA=>x"10",
   ECS_GPIO_REA_OUT_DATA=>x"11",
   ECS_GPIO_SET_IO_DIR=>x"20",
   ECS_GPIO_REA_IO_DIR=>x"21",
   ECS_GPIO_SET_INTEN=>x"20",
   ECS_GPIO_REA_INTEN=>x"21",
   ECS_GPIO_SET_TRIG=>x"30",
   ECS_GPIO_REA_TRIG=>x"31",
   ECS_GPIO_SET_GIE=>x"60",
   ECS_GPIO_REA_GIE=>x"61",
   ECS_GPIO_SET_INT=>x"70",
   ECS_GPIO_REA_INT=>x"71",
   ECS_GPIO_SET_CLK_SEL=>x"80",
   ECS_GPIO_REA_CLK_SEL=>x"81",
   ECS_GPIO_SET_EDG=>x"90",
   ECS_GPIO_REA_EDG=>x"91",
   INVALID  =>x"FF"
);




--------------------------------------------------------------------------------
--type decoding_state_t is
  (decoding_sof,decoding_ch,decoding_tr,decoding_ack,decoding_protocol,decoding_eof);



constant CMD_MEM_ADDR_WIDTH      :  integer := 12;
constant RPY_MEM_ADDR_WIDTH      :  integer := 12;
constant MEM_DATA_WIDTH       :  integer := 32;

procedure i2c_rpy_builder(
   cmd            :  byte_t;
   signal rpy_p_len  :  out   natural range 0 to 31;
   signal is_var_len :  out   std_logic
);









type PROT_DRIVER_ENUM_t is (SCA_CTL, GPIO, SPI, I2C, JTAG, ADC, DAC);
```

```
   type sca_num_prot_t  is array (PROT_DRIVER_ENUM_t'left to PROT_DRIVER_ENUM_t'right) of natural range 0
    to SCA_COUNT -1;


   type prot_cmd_vector_by_prot_t is array(PROT_DRIVER_ENUM_t'left to PROT_DRIVER_ENUM_t'right) of
    prot_cmd_t;

   type elink_by_sca_t  is array(0 to SCA_COUNT-1) of std_logic_vector(1 downto 0);

   constant ECS_DATA_WIDTH :  natural  := 8+5*32;

   constant ECS_MEM_COUNT  :  natural := SCA_COUNT*CH_BY_SCA_COUNT;


-- constant ECS_DATA_WIDTH :  natural  := ecs_packet_t.ecs_cmd'length
--                            + ecs_packet_t.protocol_specific'length
--                            + ecs_packet_t.len'length
--                            + ecs_packet_t.data'length;

   --constant ECS_ADDR_WIDTH  :  natural  :=  byte_t'length
   --                         +  byte_t'length;

-- function ecs_packet_to_ecs_data_bus(ecs_packet  :  ecs_packet_t)
--    return std_logic_vector;
-- function ecs_packet_to_ecs_address_bus(ecs_packet  :  ecs_packet_t)
--    return std_logic_vector;

-- function sol40_data_to_ecs_packet (
-- ecs_addr : std_logic_vector(ECS_ADDR_WIDTH-1 downto 0);
-- ecs_data : std_logic_vector(ECS_DATA_WIDTH-1 downto 0)
-- )
-- return ecs_packet_t;


   --type SCA_link_state_t is (RESET, IDLE, TESTING, CONNECTING, READY, DISABLED);
   type SCA_link_state_t is (RESET, TESTING, CONNECTING, ACTIVE, DISABLED);
   --type SCA_link_state_array_t is array(0 to GBT_COUNT-1,0 to SCA_BY_GBT_COUNT-1) of SCA_link_state_t;
   type SCA_link_state_array_t is array(0 to SCA_COUNT-1) of SCA_link_state_t;

    function std_logic_to_integer( s : std_logic ) return integer;

end SCA_Package;




package body SCA_Package is




procedure i2c_cmd_builder(
   cmd            :  byte_t;
   signal cmd_p_len  :  out   natural range 0 to 31;
   signal has_len    :  out   std_logic;
   signal cmd_var_len   :  out   std_logic;
   signal rpy_var_len   :  out   std_logic
) is
```

```
begin

   case cmd is
      when x"00"|x"03"|x"83"|x"84"|x"85"|x"89" =>
         --C: CH# + TR# + CMD + A[7:0] + DW
         --or C: CH# + TR# + CMD + A[9:8] + A[7:0]
         cmd_p_len <= 5;
         cmd_var_len <= '0';
         rpy_var_len <= '0';
         if cmd = x"89" then
            has_len <= '0';
         else
            has_len <= '1';
         end if;
      when x"02"|x"87"|x"88" =>
         -- C: CH# + TR# + CMD + A[9:8] + A[7:0] + DW
         cmd_p_len <= 6;
         cmd_var_len <= '0';
         rpy_var_len <= '0';
         if cmd = x"87" then
            has_len <= '1';
         end if;
      when x"01"|x"80"|x"81"|x"82"|x"f0" =>
         -- C: CH# + TR# + CMD + A[7:0]
         -- R: CH# + TR# + ACK + DR
         cmd_p_len <= 4;
         cmd_var_len <= '0';
      when x"86" =>
         -- C: CH# + TR# + CMD + LEN + A[9:8] + A[7:0] + DW
         cmd_p_len <= 7;
         cmd_var_len <= '0';
         -- C: CH# + TR# + CMD + LEN + A[7:0] + DW
         -- C: CH# + TR# + CMD + LEN + A[7:0]
         -- R: CH# + TR# + ACK + DR (LEN -1 bytes)


      when x"f1"|x"f2"|x"f3"|x"f7"|x"ff" =>
         -- C: CH# + TR# + CMD
         cmd_p_len <= 3;
         cmd_var_len <= '0';
      when others =>
         cmd_p_len <= 0;
         cmd_var_len <= '0';
   end case;
end procedure i2c_cmd_builder;


procedure i2c_rpy_builder(
   cmd           :  byte_t;
   signal rpy_p_len  :  out   natural range 0 to 31;
   signal is_var_len :  out   std_logic
) is
   --variable rpy_p_len :  integer;

begin

   case cmd is
      when x"00"|x"01"|x"02"|x"03"|x"81"|x"82"|x"83"|x"84"|x"85"|x"86"|x"88"|x"f0"|x"f1" =>
         rpy_p_len <= 4;
```

```
         is_var_len <= '0';
      when x"f2" =>
         rpy_p_len <= 3;
         is_var_len <= '0';
      when x"87"|x"89" =>
         rpy_p_len <= 3;
         is_var_len <= '1';
      when others =>
         rpy_p_len <= 0;
         is_var_len <= '0';
   end case;
end procedure i2c_rpy_builder;



--function ecs_packet_to_ecs_data_bus(ecs_packet   :  ecs_packet_t)
--    return std_logic_vector is
--    variable ret : std_logic_vector(ECS_DATA_WIDTH-1 downto 0);
-- begin
--    ret := ecs_packet.ecs_cmd
--        & ecs_packet.protocol_specific & ecs_packet.len & ecs_packet.data;
--
--    return ret;
-- end function ecs_packet_to_ecs_data_bus;
--

--function ecs_packet_to_ecs_address_bus(ecs_packet   :  ecs_packet_t)
--    return std_logic_vector is
--    variable ret : std_logic_vector(ECS_ADDR_WIDTH-1 downto 0);
--    variable sca_nr_in_bits :  byte_t;
-- begin
--    if SCA_COUNT > 2**(sca_nr_in_bits'length) then
--       report "SCA Number field of the ECS Interface is not big enough" severity ERROR;
--    end if;
--    sca_nr_in_bits := std_logic_vector(to_unsigned( ecs_packet.sca_nr , sca_nr_in_bits'length));
--
--    ret := sca_nr_in_bits & ecs_packet.sca_ch;
--
--    return ret;
-- end function ecs_packet_to_ecs_address_bus;

--function sol40_data_to_ecs_packet (
-- ecs_addr : std_logic_vector(ECS_ADDR_WIDTH-1 downto 0);
-- ecs_data : std_logic_vector(ECS_DATA_WIDTH-1 downto 0)
--)
-- return ecs_packet_t is
-- variable ret : ecs_packet_t;
--begin
--
-- ret.sca_nr := ecs_addr (15 downto 8);
-- ret.sca_ch := ecs_addr (7 downto 0);
--
--
-- ret.ecs_cmd := ecs_data( ecs_data'high
--       downto ecs_data'high - ret.ecs_cmd'length + 1);
-- ret.protocol_specific := ecs_data( ecs_data'high - ret.ecs_cmd'length
--       downto ecs_data'high - ret.ecs_cmd'length - ret.protocol_specific'length + 1);
-- ret.len :=  ecs_data(ecs_data'high - ret.ecs_cmd'length - ret.protocol_specific'length
--       downto ecs_data'high - ret.ecs_cmd'length - ret.protocol_specific'length - ret.len'length + 1);
```

```
-- ret.data :=  ecs_data(ecs_data'high - ret.ecs_cmd'length - ret.protocol_specific'length - ret.len'length
--       downto ecs_data'high - ret.ecs_cmd'length - ret.protocol_specific'length - ret.len'length -
     ret.data'length + 1);
-- return ret;
--end function sol40_data_to_ecs_packet;


   function std_logic_to_integer( s : std_logic ) return integer is
      begin
        if s = '1' then
          return 1;
        else
          return 0;
        end if;
   end function;


end package body;
```

```
--------------------------------------------------------------------------------
--! @file ecs_buffer_layer.vhd
--! @author Cairo Caplan <cairo.caplan@cern.ch>
--! @date April, 2016
--------------------------------------------------------------------------------

library ieee;
use ieee.std_logic_1164.all;

use work.SCA_Package.all;

entity ecs_buffer_layer is
   port (
      clk            : in std_logic;
      rst            : in std_logic;
      --
      ecs_cmd_int_av_o     : out std_logic;
      ecs_cmd_int_ena_i    : in  std_logic;
       ecs_cmd_int_data_i  :  in ecs_packet_t;
      --
       ecs_rpy_int_isRead_i : in std_logic;
       ecs_rpy_int_addr_i : in  ecs_addr_packet_t;
       ecs_rpy_int_av_o   : out std_logic;
       ecs_rpy_int_data_o : out ecs_packet_t;
       new_rpy_o        : out std_logic;
       --

      ecs_rpy_prot_av_o   : out std_logic;
      ecs_rpy_prot_ena_i  : in  std_logic;
      ecs_rpy_prot_data_i : in  ecs_packet_t;
      --
      ecs_cmd_prot_ena_i  : in  std_logic;
      ecs_cmd_prot_av_o   : out std_logic;
      ecs_cmd_prot_data_o : out ecs_packet_t
      --



   );
end entity ecs_buffer_layer;

architecture RTL of ecs_buffer_layer is

   component ecs_cmd_fifo
      generic(FIFO_LENGTH : natural := 2);
      port(
          clk           :  in std_logic;
          rst                : in  std_logic;
          ecs_cmd_int_av_o   : out std_logic;
          ecs_cmd_int_data_i  : in  ecs_packet_t;
          ecs_cmd_int_ena_i    : in  std_logic;
          ecs_cmd_prot_ena_i  : in  std_logic;
          ecs_cmd_prot_av_o   : out std_logic;
          ecs_cmd_prot_data_o : out ecs_packet_t);
   end component ecs_cmd_fifo;

   component ecs_rpy_mem
      port(
         clk                 : in  std_logic;
```

```
         rst                  : in  std_logic;
         ecs_rpy_int_isRead_i : in  std_logic;
         ecs_rpy_int_addr_i   : in  ecs_addr_packet_t;
         ecs_rpy_int_av_o     : out std_logic;
         ecs_rpy_int_data_o   : out ecs_packet_t;
         new_rpy_o            : out std_logic;
         ecs_rpy_prot_av_o    : out std_logic;
         ecs_rpy_prot_ena_i   : in  std_logic;
         ecs_rpy_prot_data_i  : in  ecs_packet_t);
   end component ecs_rpy_mem;


begin

   ecs_cmd_fifo_inst : component ecs_cmd_fifo
      generic map(FIFO_LENGTH => 32)
      port map(
            clk                 => clk,
              rst                 => rst,
              ecs_cmd_int_av_o   => ecs_cmd_int_av_o,
              ecs_cmd_int_data_i    => ecs_cmd_int_data_i,
              ecs_cmd_int_ena_i  => ecs_cmd_int_ena_i,
              ecs_cmd_prot_ena_i  => ecs_cmd_prot_ena_i,
              ecs_cmd_prot_av_o   => ecs_cmd_prot_av_o,
              ecs_cmd_prot_data_o => ecs_cmd_prot_data_o);

   ecs_rpy_mem_inst : component ecs_rpy_mem
      port map(
         clk                 => clk,
         rst                 => rst,
         ecs_rpy_int_isRead_i => ecs_rpy_int_isRead_i,
         ecs_rpy_int_addr_i   => ecs_rpy_int_addr_i,
         ecs_rpy_int_av_o     => ecs_rpy_int_av_o,
         ecs_rpy_int_data_o   => ecs_rpy_int_data_o,
         new_rpy_o            => new_rpy_o,
         ecs_rpy_prot_av_o    => ecs_rpy_prot_av_o,
         ecs_rpy_prot_ena_i   => ecs_rpy_prot_ena_i,
         ecs_rpy_prot_data_i  => ecs_rpy_prot_data_i
      );
end architecture RTL;
```

```
--------------------------------------------------------------------------------
--! @file ecs_cmd_fifo.vhd
--! @author Cairo Caplan <cairo.caplan@cern.ch>
--! @date April, 2016
--------------------------------------------------------------------------------

library ieee;
use ieee.std_logic_1164.all;

use work.SCA_Package.all;


entity ecs_cmd_fifo is
   generic(
      FIFO_LENGTH :  natural:=2
   );
   port (
      clk                  : in std_logic;
      rst                     : in std_logic;

      --ECS Interface Layer
      ecs_cmd_int_av_o        :  out   std_logic;
      ecs_cmd_int_data_i         :  in ecs_packet_t;
      ecs_cmd_int_ena_i       :  in std_logic;

      --ECS Protocol Layer
      ecs_cmd_prot_ena_i         :  in std_logic;
      ecs_cmd_prot_av_o       :  out   std_logic;
      ecs_cmd_prot_data_o        :  out   ecs_packet_t

   );
end entity ecs_cmd_fifo;

architecture RTL of ecs_cmd_fifo is

   constant Length      :  natural:=FIFO_LENGTH;
   signal w_ptr,r_ptr   :  natural range 0 to Length - 1;
   signal cmd_fifo   :  ecs_packet_array_t(0 to Length - 1);
   signal full,empty :  std_logic;

begin

   empty <= '1' when (w_ptr=r_ptr) else '0';
   full  <= '1' when ((w_ptr=r_ptr-1 and r_ptr/=0) or (w_ptr=Length-1 and r_ptr=0)) else '0';

   fifo_control : process (clk) is
   begin
      if rising_edge(clk) then
         if rst = '1' then
            w_ptr <= 0;
            r_ptr <= 0;
         else
            if ecs_cmd_int_ena_i='1' and full='0' then
               --cmd_fifo(w_ptr) <= ecs_cmd_int_data_i;

               if w_ptr = Length-1 then
                  w_ptr <= 0;
               else
                  w_ptr <= w_ptr + 1;
```

```
            end if;
         end if;
         --ecs_cmd_prot_data_o <= cmd_fifo(r_ptr);
         if ecs_cmd_prot_ena_i='1' and empty='0' then
            --ecs_cmd_prot_data_o <= cmd_fifo(r_ptr);
            if r_ptr = Length-1 then
               r_ptr <= 0;
            else
               r_ptr <= r_ptr + 1;
            end if;
         end if;
      end if;
   end if;
end process fifo_control;

ecs_cmd_int_av_o <= not full;

ecs_cmd_prot_av_o <= not empty and not ecs_cmd_prot_ena_i;



fifo_data : process (clk) is
begin
   if rising_edge(clk) then
      if ecs_cmd_int_ena_i='1' and full='0' then
         cmd_fifo(w_ptr) <= ecs_cmd_int_data_i;
      end if;
      ecs_cmd_prot_data_o <= cmd_fifo(r_ptr);
   end if;
end process fifo_data;




end architecture RTL;
```

```
--------------------------------------------------------------------------------
--! @file ecs_rpy_mem.vhd
--! @author Cairo Caplan <cairo.caplan@cern.ch>
--! @date April, 2016
--------------------------------------------------------------------------------

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use IEEE.math_real.all;

use work.SCA_Package.all;

--! This block is coded to use Block RAM units, so no Reset should be used
--! and there's no need for an enable for Read operations
entity ecs_rpy_mem is
   port (
      clk : in std_logic;
      rst : in std_logic;

      --ECS Interface LAyer
      ecs_rpy_int_isRead_i    :  in std_logic;
      ecs_rpy_int_addr_i         :  in ecs_addr_packet_t;
      ecs_rpy_int_av_o        :  out   std_logic;
      ecs_rpy_int_data_o         :  out   ecs_packet_t;
      new_rpy_o               :  out   std_logic;

      --ECS Protocol Layer
      ecs_rpy_prot_av_o       :  out   std_logic;
      ecs_rpy_prot_ena_i         :  in std_logic;
      ecs_rpy_prot_data_i        :  in ecs_packet_t
   );
end entity ecs_rpy_mem;

architecture RTL of ecs_rpy_mem is
   --! Number of positions in the ECS Reply memory, at principle the
   --! number of number of GBT-SCAs * Channels per SCA
   constant ECS_MEM_COUNT  :  natural := SCA_COUNT*CH_BY_SCA_COUNT;

   type mem_t is array(natural range <>) of std_logic_vector(ECS_DATA_WIDTH-1 downto 0);
   signal mem :   mem_t(0 to ECS_MEM_COUNT-1 );



   signal new_rpy_by_ch    :  std_logic_vector(0 to ECS_MEM_COUNT-1):=(others=>'0');

begin

   name : process (clk) is
      variable addr_w,addr_r  :  natural range 0 to ECS_MEM_COUNT-1;
      variable  data_r  :  std_logic_vector(ECS_DATA_WIDTH - 1 downto 0);
   begin
      if rising_edge(clk) then
         addr_r :=  to_integer(unsigned(ecs_rpy_int_addr_i.sca_nr))*(CH_BY_SCA_COUNT)
               + to_integer(unsigned(ecs_rpy_int_addr_i.sca_ch));
         data_r := mem(addr_r);
         ecs_rpy_int_data_o.sca_nr <= ecs_rpy_int_addr_i.sca_nr;
         ecs_rpy_int_data_o.sca_ch <= ecs_rpy_int_addr_i.sca_ch;
```

```
        ecs_rpy_int_data_o.ecs_cmd <= data_r(data_r'high downto data_r'high -7);
        ecs_rpy_int_data_o.err <= data_r(data_r'high - 8 downto data_r'high - 15);
        ecs_rpy_int_data_o.len <= data_r(data_r'high - 16 downto data_r'high - 23);
        ecs_rpy_int_data_o.protocol_specific <= data_r(data_r'high - 24 downto data_r'high - 39);
        ecs_rpy_int_data_o.data <= data_r(data_r'high - 40 downto 0);

        --ecs_rpy_prot_av_o <= '1';
        ecs_rpy_int_av_o <= '1';

        addr_w := to_integer(unsigned(ecs_rpy_prot_data_i.sca_nr))*(CH_BY_SCA_COUNT)
              + to_integer(unsigned(ecs_rpy_prot_data_i.sca_ch));

        if ecs_rpy_prot_ena_i = '1' then
           mem(addr_w) <= ecs_rpy_prot_data_i.ecs_cmd
                       & ecs_rpy_prot_data_i.err
                       & ecs_rpy_prot_data_i.len
                       & ecs_rpy_prot_data_i.protocol_specific
                       & ecs_rpy_prot_data_i.data;
        end if;
     end if;
  end process name;

  new_rpy_manager : process(clk, new_rpy_by_ch) is
     variable addr_w,addr_r  :  natural range 0 to ECS_MEM_COUNT-1;
  begin
     if rising_edge(clk) then
        addr_w := to_integer(unsigned(ecs_rpy_prot_data_i.sca_nr))*(CH_BY_SCA_COUNT)
                + to_integer(unsigned(ecs_rpy_prot_data_i.sca_ch));
        addr_r := to_integer(unsigned(ecs_rpy_int_addr_i.sca_nr))*(CH_BY_SCA_COUNT)
                + to_integer(unsigned(ecs_rpy_int_addr_i.sca_ch));
        if rst='1' then
           new_rpy_by_ch <= (others=>'0');
        else
           if ecs_rpy_int_isRead_i='0' and ecs_rpy_prot_ena_i='1' then
              new_rpy_by_ch(addr_w) <= '1';
           elsif  ecs_rpy_int_isRead_i='1' and ecs_rpy_prot_ena_i='0'  then
              new_rpy_by_ch(addr_r) <= '0';
           elsif ecs_rpy_int_isRead_i='1' and ecs_rpy_prot_ena_i='1' then
              if addr_r /= addr_w then
                 new_rpy_by_ch(addr_r) <= '0';
                 new_rpy_by_ch(addr_w) <= '1';
              else
                 new_rpy_by_ch(addr_w) <= '1';
              end if;
           end if;
        end if;
     end if;
     new_rpy_o <= new_rpy_by_ch(addr_r);
  end process new_rpy_manager;

  ecs_rpy_prot_av_o <= not ecs_rpy_int_isRead_i and not ecs_rpy_prot_ena_i;



end architecture RTL;
```

```
--------------------------------------------------------------------------------
--! @file ecs_interface_layer.vhd
--! @author Cairo Caplan <cairo.caplan@cern.ch>
--! @date April, 2016
--------------------------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

use work.Constant_Declaration.all;
use work.SCA_Package.all;

entity ecs_interface_layer is
   generic(
      BASE_ADDRESS         : in std_logic_vector(ECS_address_size-1 downto 0)
   );
   port (
      ECS_ADDRESS_i        : in  std_logic_vector(ECS_address_size-1 downto 0);
       READ_ECS_RESPONSE_i    : in  std_logic;
       ECS_RESPONSE_o         : out std_logic_vector (31 downto 0);
       ECS_RESPONSE_VALID_o   : out std_logic;
       ECS_COMMAND_i       : in  std_logic_vector (31 downto 0);
       WRITE_ECS_COMMAND_i    : in  std_logic;
       --
       ecs_clk             : in  std_logic;
       SRES_i              : in  std_logic;
       --
      ecs_cmd_int_ena_o    : out std_logic;
      ecs_cmd_int_av_i     : in  std_logic;
      ecs_cmd_int_data_o      : out    ecs_packet_t;
      --
      ecs_rpy_int_isRead_o    : out std_logic;
      ecs_rpy_int_addr_o      : out    ecs_addr_packet_t;
      ecs_rpy_int_data_i      : in  ecs_packet_t;
      new_rpy_i            : in  std_logic;

      sol40_sca_rst_o         :  out std_logic;
      reset_sca_o          :  out   std_logic_vector(63 downto 0);
      disable_sca_o        :  out   std_logic_vector(63 downto 0);
      timeout_value_o         :  out std_logic_vector(25 downto 0);
      sca_link_state_array :  in SCA_link_state_array_t
    );
end entity ecs_interface_layer;

architecture RTL of ecs_interface_layer is

   signal address_reg         : ecs_addr_packet_t :=((others=>'0'),(others=>'0'));
   signal ecs_cmd_int_data    : ecs_packet_t;

   signal ecs_rpy_int_data_buf   : ecs_packet_t;
   signal sol40_sca_rst    : std_logic := '0';

   signal curr_SCA_link_state : SCA_link_state_t;
   signal disable_sca         : std_logic_vector(63 downto 0) := (others=>'0');
   signal timeout_effective   : natural range 0 to (2**16)*(2**10);
   -- Starts with a value of ~ 1,6 s
   signal timeout_25_6_us     : natural range 0 to (2**16)-1 := (2**16)-1;

   signal reset_sca        :  std_logic_vector(63 downto 0);
```

```
begin

   --assert ECS_data_size=32 report "Avalon/ECS_data_size is not 32, I'm not prepared for this" severity
     failure;
   --assert false report "ECS_address_size or address_MM_slave_ECS are not used anyway" severity NOTE;


   interface : process(ecs_clk, SRES_i) is
      variable ecs_cmd_int_ena_var  :  std_logic;
      variable ecs_rpy_int_isRead_var  :  std_logic;
      variable sol40_sca_rst_var    :  std_logic;
      variable sca_link_state       :  std_logic_vector(1 downto 0);
      variable reset_sca_var        :  std_logic_vector(63 downto 0);
      --variable sca_ch_state_
   begin
      ecs_cmd_int_ena_var := '0';
      --reset_sca_var:= (others=>'0');

      if SRES_i='1' then

         ecs_rpy_int_isRead_o <= '0';
         address_reg <= ((others=>'0'),(others=>'0'));
         ECS_RESPONSE_o <= (others=>'1');
         ECS_RESPONSE_VALID_o <= '0';
         ecs_rpy_int_isRead_o <= '0';
         ecs_cmd_int_ena_o <= '0';
         sol40_sca_rst_var := '1';
         reset_sca <= (others=>'1');
         disable_sca <= (others=>'0');
         timeout_25_6_us <= (2**16)-1;
         ecs_rpy_int_addr_o <= ((others=>'0'),(others=>'0'));
      elsif rising_edge(ecs_clk) then
         ecs_rpy_int_isRead_var  := '0';
         ecs_cmd_int_ena_var := '0';
         sol40_sca_rst_var := '0';
         reset_sca_var := (others => '0');
         if BASE_ADDRESS(ECS_address_size-1 downto 8) = ECS_ADDRESS_i(ECS_address_size-1 downto 8)
            and BASE_ADDRESS(7 downto 0)=x"00"
            and  ECS_ADDRESS_i(7 downto 0) >= x"00"
            and  ECS_ADDRESS_i(7 downto 0) <= x"ff"
            then

            if WRITE_ECS_COMMAND_i='1' then
               case ECS_ADDRESS_i(7 downto 0) is
                  --! Bit 0 - Go Write, Bit 1 - Go Read, Bit 2 - Reply Available, Bit 3 - Reset, Bit 4--
                  when ECS_CONTROL_REGISTER=>
                     if (ECS_COMMAND_i(3)='1') then
                        sol40_sca_rst_var:='1';
                     else
                        if (ECS_COMMAND_i(0)='1') then -- GO WRITE
                           ecs_cmd_int_ena_var := '1';
                        end if;

                        if (ECS_COMMAND_i(1)='1') then -- GO READ
                           ecs_rpy_int_isRead_var := '1';
                           ecs_rpy_int_data_buf <= ecs_rpy_int_data_i;
                        end if;
```

```
            end if;

         when ECS_REPLY_ADDRESS=>
            address_reg.sca_nr <= ECS_COMMAND_i(31-8 downto 31-15);
            address_reg.sca_ch <= ECS_COMMAND_i(31-16 downto 31-23);
         when ECS_COMMAND_HEADER_0=>
            --ecs_cmd_int_data.gbt_nr <= ECS_COMMAND_i(31 downto 31-7);
            ecs_cmd_int_data.sca_nr <= ECS_COMMAND_i(31-8 downto 31-15);
            ecs_cmd_int_data.sca_ch <= ECS_COMMAND_i(31-16 downto 31-23);
            ecs_cmd_int_data.ecs_cmd <= ECS_COMMAND_i(31-24 downto 31-31);
         when ECS_COMMAND_HEADER_1=>
            ecs_cmd_int_data.len <= ECS_COMMAND_i(23 downto 16);
            ecs_cmd_int_data.protocol_specific <= ECS_COMMAND_i(15 downto 0);
         when ECS_COMMAND_DATA_0=>
            ecs_cmd_int_data.data(31 downto 0) <= ECS_COMMAND_i;
         when ECS_COMMAND_DATA_1=>
            ecs_cmd_int_data.data(63 downto 32) <= ECS_COMMAND_i;
         when ECS_COMMAND_DATA_2=>
            ecs_cmd_int_data.data(95 downto 64) <= ECS_COMMAND_i;
         when ECS_COMMAND_DATA_3=>
            ecs_cmd_int_data.data(127 downto 96) <= ECS_COMMAND_i;
         when ECS_RESET_SCA_0=>
            --Least significant 32 bits of the RESET_SCA register
            reset_sca_var(31 downto 0) := ECS_COMMAND_i;
         when ECS_RESET_SCA_1=>
            --Most significant 32 bits of the RESET_SCA register
            reset_sca_var(63 downto 32) := ECS_COMMAND_i;
         when ECS_DISABLE_SCA_0=>
            --Least significant 32 bits of the DISABLE_SCA register
            disable_sca(31 downto 0) <= ECS_COMMAND_i;
         when ECS_DISABLE_SCA_1=>
            --Most significant 32 bits of the DISABLE_SCA register
            disable_sca(63 downto 32) <= ECS_COMMAND_i;
         when ECS_TIMEOUT=>
            timeout_25_6_us <=
               to_integer(unsigned(ECS_COMMAND_i(15 downto 0)));
         when others=>
            null;
      end case;
      ECS_RESPONSE_o <= (others=>'1');
      ECS_RESPONSE_VALID_o <= '0';
   elsif  READ_ECS_RESPONSE_i='1' then
      ECS_RESPONSE_VALID_o <= '1';
      case ECS_ADDRESS_i(7 downto 0) is
         when ECS_CONTROL_REGISTER=>
            case curr_SCA_link_state is
               when RESET | TESTING =>
                  sca_link_state := "00";
               when CONNECTING =>
                  sca_link_state := "01";
               when ACTIVE =>
                  sca_link_state := "10";
               when DISABLED =>
                  sca_link_state := "11";
            end case;

            ECS_RESPONSE_o(2) <= new_rpy_i; -- Reply available at the req address
            ECS_RESPONSE_o(5 downto 4) <= sca_link_state;
            ECS_RESPONSE_o(1 downto 0) <= (others=>'0');
            ECS_RESPONSE_o(3) <= '0';
```

```
                ECS_RESPONSE_o(31 downto 6) <= (others=>'0');

            when ECS_REPLY_ADDRESS=>
                ECS_RESPONSE_o(31-8 downto 31-15) <= address_reg.sca_nr;
                ECS_RESPONSE_o(31-16 downto 31-23) <= address_reg.sca_ch;
            when ECS_COMMAND_HEADER_0=>
                ECS_RESPONSE_o(31 downto 31-7) <= x"00";
                ECS_RESPONSE_o(31-8 downto 31-15) <= ecs_cmd_int_data.sca_nr;
                ECS_RESPONSE_o(31-16 downto 31-23) <= ecs_cmd_int_data.sca_ch;
                ECS_RESPONSE_o(31-24 downto 31-31) <= ecs_cmd_int_data.ecs_cmd;
            when ECS_COMMAND_HEADER_1=>
                ECS_RESPONSE_o(31 downto 24) <= (others=>'0');
                ECS_RESPONSE_o(23 downto 16) <= ecs_cmd_int_data.len;
                ECS_RESPONSE_o(15 downto 0) <= ecs_cmd_int_data.protocol_specific;
            when ECS_COMMAND_DATA_0=>
                ECS_RESPONSE_o <= ecs_cmd_int_data.data(31 downto 0);
            when ECS_COMMAND_DATA_1=>
                ECS_RESPONSE_o <= ecs_cmd_int_data.data(63 downto 32);
            when ECS_COMMAND_DATA_2=>
                ECS_RESPONSE_o <= ecs_cmd_int_data.data(95 downto 64);
            when ECS_COMMAND_DATA_3=>
                ECS_RESPONSE_o <= ecs_cmd_int_data.data(127 downto 96);
            when ECS_REPLY_HEADER_0=>
                ECS_RESPONSE_o(31 downto 31-7) <= x"00";
                ECS_RESPONSE_o(31-8 downto 31-15) <= ecs_rpy_int_data_buf.sca_nr;
                ECS_RESPONSE_o(31-16 downto 31-23) <= ecs_rpy_int_data_buf.sca_ch;
                ECS_RESPONSE_o(31-24 downto 31-31) <= ecs_rpy_int_data_buf.ecs_cmd;
            when ECS_REPLY_HEADER_1=>
                ECS_RESPONSE_o(31 downto 24) <=  ecs_rpy_int_data_buf.err;
                ECS_RESPONSE_o(23 downto 16) <= ecs_rpy_int_data_buf.len(7 downto 0);
                ECS_RESPONSE_o(15 downto 0) <= ecs_rpy_int_data_buf.protocol_specific;
            when ECS_REPLY_DATA_0=>
                ECS_RESPONSE_o <= ecs_rpy_int_data_buf.data(31 downto 0);
            when ECS_REPLY_DATA_1=>
                ECS_RESPONSE_o <= ecs_rpy_int_data_buf.data(63 downto 32);
            when ECS_REPLY_DATA_2=>
                ECS_RESPONSE_o <= ecs_rpy_int_data_buf.data(95 downto 64);
            when ECS_REPLY_DATA_3=>
                ECS_RESPONSE_o <= ecs_rpy_int_data_buf.data(127 downto 96);
            when ECS_RESET_SCA_0=>
                ECS_RESPONSE_o <= (others=>'0');
            when ECS_RESET_SCA_1=>
                ECS_RESPONSE_o <= (others=>'0');
            when ECS_DISABLE_SCA_0=>
                ECS_RESPONSE_o <= disable_sca(31 downto 0);
            when ECS_DISABLE_SCA_1=>
                ECS_RESPONSE_o <= disable_sca(63 downto 32);
            when ECS_TIMEOUT=>
                ECS_RESPONSE_o(15 downto 0) <=
                    std_logic_vector(to_unsigned(timeout_25_6_us,16));
            when others=>
                ECS_RESPONSE_o <= (others=>'1');
        end case;
    else
        ECS_RESPONSE_VALID_o <= '0';
        ECS_RESPONSE_o <= (others=>'0');

    end if;
    reset_sca <= reset_sca_var;
else
```

```
            ECS_RESPONSE_o <= (others=>'1');
            reset_sca <= (others=>'0');
            -------------------------------------------
            --INVALID ADDRESS AREA
            ECS_RESPONSE_VALID_o <= '0';
            -------------------------------------------
         end if;
         ecs_rpy_int_addr_o <= address_reg;
         ecs_rpy_int_isRead_o <= ecs_rpy_int_isRead_var;
         ecs_cmd_int_ena_o <= ecs_cmd_int_ena_var;
         --1 timeout_25_6_us count = 2**10 clock cycles
         -- = (2**10)*25ns = 25600ns = 25.6 us
         --Maximum timeout_25_6_us time = ((2**16)-1 counts)*(25.6 us)
         -- = 1677696 us = 1,6 s
         --Hope it doesn't synthesize a multiplier
         timeout_effective <= timeout_25_6_us*(2**10);



      end if;

      sol40_sca_rst <= sol40_sca_rst_var;
      reset_sca_o <= reset_sca;

   end process interface;

   disable_sca_o <= disable_sca;
   sol40_sca_rst_o <= sol40_sca_rst;
   reset_sca_o <= reset_sca;

   link_state_process : process (ecs_clk) is
      variable j     :  natural range 0 to SCA_COUNT-1:=0;
   begin
      if rising_edge(ecs_clk) then
         if to_integer(unsigned(address_reg.sca_nr)) < SCA_COUNT then
            j:= to_integer(unsigned(address_reg.sca_nr));
         end if;
         curr_SCA_link_state <= sca_link_state_array(j);
      end if;
   end process link_state_process;


   ecs_cmd_int_data_o <= ecs_cmd_int_data;
   timeout_value_o <= std_logic_vector(to_unsigned(timeout_effective,16+10));


end architecture RTL;
```

```
--------------------------------------------------------------------------------
--! @file crc.vhd
--! @author Cairo Caplan <cairo.caplan@cern.ch>
--! @date April, 2016
--------------------------------------------------------------------------------
-- Based on the work by Sandro Bonacini

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;



--  Default parameters produce a CRC16-CCITT output.
--  When checking, correct reception is signaled by a zero in the crc
--  output.
--
--  Parameters:
--
--  CRC_WIDTH      bit width of the crc output;
--
--  DATA_WIDTH     bit width of the data input (must be at least 2);
--
--  INIT_VAL       initialization value of the crc register,
--              suggested values are all-zeros and all-ones;
--
--  POLY           Polynomial (remeber to reverse it)
--              i.e. CCITT 1021h has POLY = 'h8408.
--
----------------------------------------------------------------------

entity crc is
   generic(
      CRC_WIDTH   :  natural := 16;
      DATA_WIDTH  :  natural  := 80;--16
      INIT_VAL :  std_logic_vector((2*8)-1 downto 0) := x"ffff";
      POLY     :  std_logic_vector((2*8)-1 downto 0) := x"8408"
   );
   port (
      d        :  in std_logic_vector(DATA_WIDTH-1 downto 0);
      init     :  in std_logic;
      d_valid     :  in std_logic;
      clk         :  in std_logic;
      reset_b     :  in std_logic;
      crc_o    :  out   std_logic_vector(CRC_WIDTH-1 downto 0)
   );
end entity crc;


architecture RTL of crc is

   --! Internal Wires and Regs
   signal next_crc      :  std_logic_vector(CRC_WIDTH-1 downto 0);
   signal crc_int    :  std_logic_vector(CRC_WIDTH-1 downto 0);

   type crc_array_t is array (integer range<>) of std_logic_vector((CRC_WIDTH-1) downto 0);

   --
```

```
   function crc_atom (
      crc_in   :  std_logic_vector(CRC_WIDTH - 1 downto 0);
      d     :  std_logic
   )
      return  std_logic_vector is
      variable value :  std_logic_vector(crc_in'high downto 0);
   begin
      if ((crc_in(0) xor d)='0') then
         value := std_logic_vector(unsigned(crc_in) srl 1);

       else
         value := (std_logic_vector(unsigned(crc_in) srl 1)) xor POLY((CRC_WIDTH-1) downto 0);
      end if;

      return value;
   end function crc_atom;



   function crc_calc (

      crc_i :  std_logic_vector(CRC_WIDTH-1 downto 0);
      d     :  std_logic_vector(DATA_WIDTH-1 downto 0)
      )
      return std_logic_vector is

      variable p_crc :  crc_array_t(0 to DATA_WIDTH-2);
      variable value    :  std_logic_vector(CRC_WIDTH-1 downto 0);
   begin

      p_crc(0) := crc_atom(crc_i, d(0));
      for i in 1 to DATA_WIDTH-2 loop
         p_crc(i) := crc_atom(p_crc(i-1), d(i));
      end loop;


      value := crc_atom(p_crc(DATA_WIDTH-2), d(DATA_WIDTH-1));

      return value;
   end function crc_calc;



begin



   d_or_crc_i_init : process(d,crc_int,init) is
   begin
      if (init = '1') then
         next_crc <= crc_calc(INIT_VAL, d);
      else
         next_crc <= crc_calc(crc_int,d);
      end if;
      --next_crc <= crc_calc(crc_i,d);

   end process d_or_crc_i_init;

   --! synopsys async_set_reset "reset_b"
   synopsys_async_set_reset : process (clk, reset_b) is
```

```
   begin
      if reset_b = '0' then
         crc_int <=  INIT_VAL;
      elsif rising_edge(clk) then

         if (init and (not d_valid))='1' then
            crc_int <=INIT_VAL;
         elsif (d_valid='1') then
            crc_int <= next_crc;
         else
            crc_int <= crc_int;
         end if;

      end if;
   end process synopsys_async_set_reset;

   crc_o <= crc_int;



end architecture RTL;
```

```
--!-------------------------------------------
--!   Based on the SRAM verilog file by Sandro Bonacini, CERN PH/ESE
--!
--!      @author Cairo Caplan, CBPF
--!-
--!-------------------------------------------------------------------------------

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity fifo_mem is
   generic(
      DATA_WIDTH  :  integer := 4;
      ADDR_WIDTH  :  integer := 3

   );
   port (
      addr_w, addr_r :  in std_logic_vector(ADDR_WIDTH-1 downto 0);
      data_r         :  out   std_logic_vector(DATA_WIDTH-1 downto 0);
      data_w         :  in std_logic_vector(DATA_WIDTH-1 downto 0);
      w_enable, clk_w   :  in std_logic
   );
end entity fifo_mem;

architecture RTL of fifo_mem is
   constant SIZE     :  integer  := 2**ADDR_WIDTH;

   --signal  wl_w_latched  :   std_logic_vector(SIZE-1 downto 0);
   type mem_t is array (natural range <>) of std_logic_vector(DATA_WIDTH-1 downto 0);
   signal mem  :   mem_t(SIZE-1 downto 0);


begin



   memory_write : process(clk_w, addr_r, mem) is
   begin
      if rising_edge(clk_w) then
         if w_enable='1' then
            mem(to_integer(unsigned(addr_w))) <= data_w ;-- after 1 ns;
         end if;
      end if;
      --wl_w(j) <= wl_w_latched(j) and clk_w_gated;
      data_r <= mem(to_integer(unsigned( addr_r )));
   end process memory_write;




end architecture RTL;
```

```
----------------------------------------------------------------------------
--! @file fpga_elink.vhd
--! @author Cairo Caplan <cairo.caplan@cern.ch>
--! @date April, 2016
----------------------------------------------------------------------------

library ieee;
use ieee.std_logic_1164.all;

entity fpga_elink is
   generic(
      --! If 1, port behaves as master, 0 as slave.
      MASTER            :  integer := 1;
      --! If 1, header field is present in the packet structure. The header field carries address and
     command information.
      HEADER_FIELD      :  integer := 1;
      --! Transmit and receive FIFO addressing width.
      ADDR_WIDTH        :  integer := 5;
      --! Maximum expected packet length
      MAX_PACKET_LENGTH :  integer := 16

      );
   port (
      --! \name Backend signals
      --! Backend signals
      --! \{
      ------------------------------------------------------------
      --! Reset signal , active on 1
      reset       :  in  std_logic;
      --! User clock
      user_clk    :  out   std_logic;
      --!
      cmd_busy    :  out   std_logic;

      --! Transfer Enable (Data Valid)
      rx_ena         :  out std_logic;
      --! Ready to receive data
      rx_dav         :  in  std_logic;
      --! Start of Packet
      rx_sop         :  out std_logic;
      --! End of Packet
      rx_eop         :  out std_logic;
      --! Test command receive flag
      rx_cmd_test    :  out   std_logic;
      --! Reset command receive flag
      rx_cmd_reset   :  out   std_logic;
      --! Unnumbered acknowledge receive flag
      rx_cmd_ua      :  out   std_logic;
      --! Received packet number
      rx_ns       :  out std_logic_vector(2 downto 0);
      --! Address Bus
      rx_adr         :  out   std_logic_vector(7 downto 0);
      --! Data Bus
      rx_dat         :  out std_logic_vector(15 downto 0);
      --! Last correctly received packet number
      rx_nr       :  out std_logic_vector(2 downto 0);
      --! Received SREJ command word
      rx_cmd_srej    :  out std_logic_vector(6 downto 0);
      --! Rx error, currently always 0 . . .
```

```
      rx_err         :  out std_logic;
      -------------------------------------------------------
      --! Transfer Enable (Data Valid)
      tx_ena         :  in std_logic;
      --! Ready to receive data
      tx_dav         :  out std_logic;
      --! Send test command flag
      tx_cmd_test    :  in std_logic;
      --! Send reset command flag
      tx_cmd_reset   :  in  std_logic;
      --! Send connect command flag
      tx_cmd_sabm    :  in std_logic;
      --! Transmitted packet number
      tx_ns          :  out std_logic_vector(2 downto 0);
      --! Send Address command flag
      tx_adr         :  in std_logic_vector(7 downto 0);
      --! Data bus
      tx_dat         :  in  std_logic_vector(15 downto 0);
      --! Transmitter current level setting (relevant on backend?)
      tx_cset        :  in std_logic_vector(3 downto 0);
      --! \}

      --! \name Elink signals
      --! Elink signals, to be connected to the GBT chip
      --! \{
      --!
      tx_clk         :  in std_logic;
      tx_sd          :  out std_logic_vector(1 downto 0);
      rx_clk         :  in std_logic;
      rx_sd       :  in std_logic_vector(1 downto 0)
      --! \}
   );



end entity fpga_elink;

architecture RTL of fpga_elink is

   --constant THRESHOLD :  integer:= 2**ADDR_WIDTH -MAX_PACKET_LENGTH-1;

   signal rx_sdr, tx_sdr      :  std_logic_vector(1 downto 0);

   component HDLC
      generic(MASTER            : integer := 0;
              HEADER_FIELD      : integer := 1;
              ADDR_WIDTH        : integer := 5;
              MAX_PACKET_LENGTH : integer := 16);
      port(tx_dat       : in  std_logic_vector(15 downto 0);
          rx_dat       : out std_logic_vector(15 downto 0);
          rx_dav       : in  std_logic;
          tx_clk       : in  std_logic;
          rx_clk       : in  std_logic;
          rx_ena       : out std_logic;
          rx_eop       : out std_logic;
          tx_dav       : out std_logic;
          rx_sop       : out std_logic;
          rx_err       : out std_logic;
          resetb       : in  std_logic;
          tx_ena       : in  std_logic;
```

```
        user_clk     : out std_logic;
        tx_adr       : in  std_logic_vector(7 downto 0);
        rx_adr       : out std_logic_vector(7 downto 0);
        tx_cmd_reset : in  std_logic;
        tx_cmd_test  : in  std_logic;
        tx_cmd_sabm  : in  std_logic;
        rx_cmd_reset : out std_logic;
        rx_cmd_test  : out std_logic;
        rx_cmd_ua    : out std_logic;
        tx_ns        : out std_logic_vector(2 downto 0);
        rx_nr        : out std_logic_vector(2 downto 0);
        rx_ns        : out std_logic_vector(2 downto 0);
        rx_cmd_srej  : out std_logic_vector(6 downto 0);
        cmd_busy     : out std_logic;
        tx_sdr       : out std_logic_vector(1 downto 0);
        rx_sdr       : in  std_logic_vector(1 downto 0));
    end component HDLC;

    signal resetb : std_logic;


begin
    resetb <= not reset;


    HDLC_inst : component HDLC
       generic map(MASTER            => MASTER,
                   HEADER_FIELD      => HEADER_FIELD,
                   ADDR_WIDTH        => ADDR_WIDTH,
                   MAX_PACKET_LENGTH => MAX_PACKET_LENGTH)
       port map(tx_dat       => tx_dat,
                rx_dat       => rx_dat,
                rx_dav       => rx_dav,
                tx_clk       => tx_clk,
                rx_clk       => rx_clk,
                rx_ena       => rx_ena,
                rx_eop       => rx_eop,
                tx_dav       => tx_dav,
                rx_sop       => rx_sop,
                rx_err       => rx_err,
                resetb       => resetb,
                tx_ena       => tx_ena,
                user_clk     => user_clk,
                tx_adr       => tx_adr,
                rx_adr       => rx_adr,
                tx_cmd_reset => tx_cmd_reset,
                tx_cmd_test  => tx_cmd_test,
                tx_cmd_sabm  => tx_cmd_sabm,
                rx_cmd_reset => rx_cmd_reset,
                rx_cmd_test  => rx_cmd_test,
                rx_cmd_ua    => rx_cmd_ua,
                tx_ns        => tx_ns,
                rx_nr        => rx_nr,
                rx_ns        => rx_ns,
                rx_cmd_srej  => rx_cmd_srej,
                cmd_busy     => cmd_busy,
                tx_sdr       => tx_sdr,
                rx_sdr       => rx_sdr
                );
```

```
    rx_sdr <= rx_sd;
    tx_sd <= tx_sdr;


end architecture RTL;
```

```
--------------------------------------------------------------------------------
--! @file HDLC.vhd
--! @author Cairo Caplan <cairo.caplan@cern.ch>
--! @date April, 2016
--------------------------------------------------------------------------------

library ieee;
use ieee.std_logic_1164.all;

entity HDLC is
   generic(
      MASTER            :  integer := 0;
      HEADER_FIELD      :  integer := 1;
      ADDR_WIDTH        :  integer := 5;
      MAX_PACKET_LENGTH :  integer := 16

   );
   port (
      tx_dat         :  in  std_logic_vector(15 downto 0);
      rx_dat         :  out std_logic_vector(15 downto 0);
      rx_dav         :  in  std_logic;
      tx_clk         :  in  std_logic;
      rx_clk         :  in std_logic;
      rx_ena         :  out std_logic;
      rx_eop         :  out std_logic;
      tx_dav         :  out std_logic;
      rx_sop         :  out std_logic;
      rx_err         :  out std_logic;
      resetb         :  in  std_logic;
      tx_ena         :  in std_logic;
      user_clk    :  out   std_logic;
      tx_adr         :  in std_logic_vector(7 downto 0);
      rx_adr         :  out   std_logic_vector(7 downto 0);
      tx_cmd_reset   :  in  std_logic;
      tx_cmd_test    :  in std_logic;
      tx_cmd_sabm    :  in std_logic;
      rx_cmd_reset   :  out   std_logic;
      rx_cmd_test    :  out   std_logic;
      rx_cmd_ua      :  out   std_logic;
      tx_ns          :  out std_logic_vector(2 downto 0);
      rx_nr       :  out std_logic_vector(2 downto 0);
      rx_ns       :  out std_logic_vector(2 downto 0);
      rx_cmd_srej    :  out std_logic_vector(6 downto 0);
      cmd_busy    :  out   std_logic;
      tx_sdr         :  out std_logic_vector(1 downto 0);
      rx_sdr         :  in std_logic_vector(1 downto 0)
   );
end entity HDLC;

architecture RTL of HDLC is

   constant THRESHOLD         :  integer := 2**ADDR_WIDTH -MAX_PACKET_LENGTH-1;
   --wire  rx_cmd_reset__1 => signal rx_cmd_reset_int and so on ...
   signal rx_cmd_reset_int_int   :  std_logic;
   signal tx_cmd_reset_int :  std_logic;

   signal rx_cmd_test_int  :  std_logic;
```

```
signal rx_vr         :  std_logic_vector(2 downto 0);

--wire [6:0]  tx_cmd_srej_1, tx_cmd_srej__1;  => tx_cmd_srej, tx_cmd_srej_int
signal   tx_cmd_srej    :  std_logic_vector(6 downto 0);
signal   tx_cmd_srej_int   :  std_logic_vector(6 downto 0);

component tx
   generic(HEADER_FIELD : integer := 1;
           ADDR_WIDTH   : integer := 12;
           THRESHOLD    : integer := 3072);
   port(tx_dat       : in  std_logic_vector(15 downto 0);
        clk          : in  std_logic;
        tx_ena       : in  std_logic;
        tx_adr       : in  std_logic_vector(7 downto 0);
        resetb       : in  std_logic;
        tx_cmd_reset : in  std_logic;
        tx_cmd_test  : in  std_logic;
        rx_cmd_reset : in  std_logic;
        tx_dav       : out std_logic;
        tx_sdr       : out std_logic_vector(1 downto 0);
        tx_ns        : out std_logic_vector(2 downto 0);
        tx_nr        : in  std_logic_vector(2 downto 0);
        tx_cmd_srej  : in  std_logic_vector(6 downto 0);
        tx_cmd_sabm  : in  std_logic;
        tx_cmd_ua    : in  std_logic;
        cmd_busy     : out std_logic);
end component tx;

component rx
   generic(HEADER_FIELD : integer := 1;
           ADDR_WIDTH   : integer := 8);
   port(rx_dat       : out std_logic_vector(15 downto 0);
        rx_ena       : out std_logic;
        rx_eop       : out std_logic;
        rx_sop       : out std_logic;
        rx_err       : out std_logic;
        rx_adr       : out std_logic_vector(7 downto 0);
        rx_cmd_reset : out std_logic;
        rx_cmd_test  : out std_logic;
        rx_cmd_ua    : out std_logic;
        tx_cmd_ua    : out std_logic;
        rx_cmd_srej  : out std_logic_vector(6 downto 0);
        tx_cmd_srej  : out std_logic_vector(6 downto 0);
        rx_vr        : out std_logic_vector(2 downto 0);
        rx_nr        : out std_logic_vector(2 downto 0);
        rx_ns        : out std_logic_vector(2 downto 0);
        clk          : in  std_logic;
        rx_clk       : in  std_logic;
        rx_dav       : in  std_logic;
        rx_sdr_pri   : in  std_logic_vector(1 downto 0);
        resetb       : in  std_logic;
        tx_cmd_reset : in  std_logic);
end component rx;

signal tx_cmd_sabm_int  : std_logic;
signal tx_cmd_ua_int    : std_logic;
signal tx_cmd_ua     : std_logic;
signal tx_cmd_test_int : std_logic;
signal user_clk_int  : std_logic;
```

```
   --signal active_aux_int : std_logic;
   signal rx_cmd_reset_int : std_logic;
begin


   rx_inst : component rx
      generic map(HEADER_FIELD => HEADER_FIELD,
                  ADDR_WIDTH   => ADDR_WIDTH)
      port map(rx_dat       => rx_dat,
               rx_ena       => rx_ena,
               rx_eop       => rx_eop,
               rx_sop       => rx_sop,
               rx_err       => rx_err,
               rx_adr       => rx_adr,
               rx_cmd_reset => rx_cmd_reset_int_int,
               rx_cmd_test  => rx_cmd_test_int,
               rx_cmd_ua    => rx_cmd_ua,
               tx_cmd_ua    => tx_cmd_ua,
               rx_cmd_srej  => rx_cmd_srej,
               tx_cmd_srej  => tx_cmd_srej,
               rx_vr        => rx_vr,
               rx_nr        => rx_nr,
               rx_ns        => rx_ns,
               clk      => tx_clk,
               rx_clk      => rx_clk,
               rx_dav       => rx_dav,
               rx_sdr_pri   => rx_sdr,
               resetb       => resetb,
               tx_cmd_reset => tx_cmd_reset_int);

   tx_inst : component tx
      generic map(HEADER_FIELD => HEADER_FIELD,
                  ADDR_WIDTH   => ADDR_WIDTH,
                  THRESHOLD    => THRESHOLD)
      port map(tx_dat       => tx_dat,
               clk          => user_clk_int,
               tx_ena       => tx_ena,
               tx_adr       => tx_adr,
               resetb       => resetb,
               tx_cmd_reset => tx_cmd_reset,
               tx_cmd_test  => tx_cmd_test_int,
               rx_cmd_reset => rx_cmd_reset_int_int,
               tx_dav       => tx_dav,
               tx_sdr       => tx_sdr,
               tx_ns        => tx_ns,
               tx_nr        => rx_vr,
               tx_cmd_srej  => tx_cmd_srej_int,
               tx_cmd_sabm  => tx_cmd_sabm_int,
               tx_cmd_ua    => tx_cmd_ua_int,
               cmd_busy     => cmd_busy);

   rx_cmd_reset_int <= '0' when MASTER /= 0 else rx_cmd_reset_int_int;
   rx_cmd_reset <= rx_cmd_reset_int;

   rx_cmd_test <= rx_cmd_test_int when MASTER /= 0 else '0';

   tx_cmd_reset_int <= tx_cmd_reset when MASTER /= 0 else '0';

   tx_cmd_test_int <= tx_cmd_test when MASTER /= 0 else rx_cmd_test_int;
```

```
   tx_cmd_sabm_int <= tx_cmd_sabm when MASTER /= 0 else '0';

   tx_cmd_ua_int <= '0' when MASTER /= 0 else tx_cmd_ua;

   tx_cmd_srej_int <= (others=>'0') when MASTER /= 0 else tx_cmd_srej;

   user_clk_int <= tx_clk;
   --active_aux <= active_aux_int;
   user_clk <= user_clk_int;

end architecture RTL;
```

```
----------------------------------------------------------------------------
--! @file MAC_rx.vhd
--! @author Cairo Caplan <cairo.caplan@cern.ch>
--! @date April, 2016
----------------------------------------------------------------------------

library ieee;
use ieee.std_logic_1164.all;

use ieee.numeric_std.all;

--Sandro's comments:
--// missing: rx_ns_o <=#1 rx_ns_i;
--// missing: rx_dat_o <=#1 rx_dat_i;
--// missing: rx_eop_o <=#1 rx_eop_i;
--// missing: rx_ena_o <=#1 rx_ena_i;
--// missing: rx_sop_o <=#1 rx_sop_i;
--// missing: rx_nr_o <=#1 rx_nr_i;
--// missing: header <=#1 header;
--// missing: rx_cmd_ua_o <=#1 rx_cmd_ua_i;
--// missing: rx_cmd_srej_o <=#1 rx_cmd_srej_i;
--// missing: tx_cmd_ua_o <=#1 tx_cmd_ua_i;
--// missing: tx_cmd_srej_o <=#1 tx_cmd_srej_i;
--// missing: rx_cmd_sabm_o <=#1 rx_cmd_sabm_i;
--// missing: rx_cmd_test_o <=#1 rx_cmd_test_i;
--// missing: rx_cmd_reset_o <=#1 rx_cmd_reset_i;
--// missing: phy_dvalid_old_o <=#1 phy_dvalid_old_i;


-- Signals not used:
--rx_dat_i, rx_sop_i, rx_cmd_test_i, cmd_busy, active, rx_cmd_ua_i, rx_cmd_srej_i
--tx_cmd_srej_i, rx_ns_i, tx_cmd_ua_i
--
entity MAC_rx is

   generic(
      ADDR_WIDTH  :  integer := 5;
      HEADER_FIELD:  integer  := 1
   );
   port (
      rx_dat            :  out   std_logic_vector(15 downto 0);
      rx_ena            :  out   std_logic;
      rx_ena_pre        :  out   std_logic;
      rx_eop            :  out   std_logic;
      rx_err            :  out   std_logic;
      rx_vr_o           :  out   std_logic_vector(2 downto 0);
      rx_nr_o           :  out   std_logic_vector(2 downto 0);
      rx_sop            :  out   std_logic;
      rx_adr            :  out   std_logic_vector(7 downto 0);
      clk               :  in std_logic;
      rx_dav            :  in std_logic;
      resetb            :  in std_logic;
      rx_cmd_reset      :  out   std_logic;
      rx_cmd_test       :  out   std_logic;
      phy_data       :  in std_logic_vector(7 downto 0);
      phy_dvalid        :  in std_logic;
      phy_dstrobe       :  in std_logic;
      crc_zero       :  in std_logic;
      rx_cmd_ua         :  out   std_logic;
```

```
      rx_cmd_srej       :  out   std_logic_vector(6 downto 0);
      tx_cmd_srej       :  out   std_logic_vector(6 downto 0);
      rx_ns          :  out   std_logic_vector(2 downto 0);
      rx_cmd_sabm       :  out   std_logic;

      tx_cmd_ua         :  out   std_logic;
      tx_cmd_reset      :  in std_logic
   );
end entity MAC_rx;

architecture RTL of MAC_rx is

   constant MAX_LENGTH  :  integer := 2**ADDR_WIDTH;

   signal fifo_nItems            :  std_logic_vector(ADDR_WIDTH-1 downto 0);
   signal fifo_nItems_complete      :  std_logic_vector(ADDR_WIDTH-1 downto 0);
   signal fifo_full           :  std_logic;
   signal rx_ns_minus_one        :  std_logic_vector(2 downto 0);


   signal disconnectb_resetb     :  std_logic;


   ----
   --signal fifo_addr_w_int         :  std_logic_vector(ADDR_WIDTH-1 downto 0);
   signal fifo_addr_w_int     :  std_logic_vector(ADDR_WIDTH-1 downto 0);

   signal fifo_addr_r_int        :  std_logic_vector(ADDR_WIDTH-1 downto 0);
   signal fifo_addr_w_last       :  std_logic_vector(ADDR_WIDTH-1 downto 0);
   signal header              :  std_logic_vector(7 downto 0);
   signal rx_cmd_reset_int       :  std_logic;
   signal rx_cmd_sabm_int        :  std_logic;
   signal phy_data_old           :  std_logic_vector(15 downto 0);
   signal phy_bytesel            :  std_logic;
   signal phy_data_lowbyte       :  std_logic_vector(7 downto 0);
   signal phy_data_old2          :  std_logic_vector(15 downto 0);

   signal phy_dvalid_old         :  std_logic;

   signal packet_length       :  std_logic_vector(15 downto 0);

   signal overflow               :  std_logic;

   signal rx_vr_int           :  std_logic_vector(2 downto 0);

   signal rx_adr_int          :  std_logic_vector(7 downto 0);

   signal rx_ena_pre_int         :  std_logic;

   signal rx_eop_int          :  std_logic;
   signal rx_ena_int          :  std_logic;




   signal fifo_data_w         :  std_logic_vector(16 downto 0);
   signal fifo_data_r         :  std_logic_vector(16 downto 0);
   signal fifo_addr_w_o    :  std_logic_vector(ADDR_WIDTH-1 downto 0);
   signal fifo_addr_r      :  std_logic_vector(ADDR_WIDTH-1 downto 0);
   signal fifo_w           :  std_logic;
```

```
   component fifo_mem
      generic(DATA_WIDTH : integer := 4;
              ADDR_WIDTH : integer := 3);
      port(addr_w, addr_r  : in  std_logic_vector(ADDR_WIDTH - 1 downto 0);
           data_r          : out std_logic_vector(DATA_WIDTH - 1 downto 0);
           data_w          : in  std_logic_vector(DATA_WIDTH - 1 downto 0);
           w_enable, clk_w : in  std_logic);
   end component fifo_mem;




begin

   rx_err <= '0';

   fifo_nItems <= std_logic_vector(unsigned(fifo_addr_w_int) - unsigned(fifo_addr_r_int));
   fifo_nItems_complete <= std_logic_vector(unsigned(fifo_addr_w_last) - unsigned(fifo_addr_r_int));
   fifo_full <= '1' when (to_integer(unsigned(fifo_nItems))=MAX_LENGTH-1) else '0';

   rx_ns_minus_one <= std_logic_vector(unsigned(header(3 downto 1)) -1 );


   disconnectb_resetb <= resetb;-- and (not disconnect_i);

   sync_process : process (clk, disconnectb_resetb) is

      variable rx_cmd_reset_var     :  std_logic;
      variable rx_cmd_sabm_var      :  std_logic;
      variable rx_cmd_ua_var     :  std_logic;
      variable rx_cmd_test_var      :  std_logic;
      variable fifo_addr_w_last_var :  std_logic_vector(ADDR_WIDTH-1 downto 0);
      variable fifo_addr_w_var   :  std_logic_vector(ADDR_WIDTH-1 downto 0);
      variable rx_vr_var         :  std_logic_vector(2 downto 0);
      variable tx_cmd_srej_var      :  std_logic_vector(6 downto 0);
      variable rx_cmd_srej_var      :  std_logic_vector(6 downto 0);
      variable packet_length_var :  std_logic_vector(15 downto 0);
      variable overflow_var         :  std_logic;
      variable phy_bytesel_var         :  std_logic;
   begin
      if disconnectb_resetb = '0' then
         fifo_addr_w_var := (others=>'0') ;
         phy_dvalid_old <= '0';
         fifo_addr_w_last_var := (others=>'0');
         rx_cmd_reset_var := '0';
         rx_cmd_test_var := '0';
         rx_vr_var := (others=>'0') ;
         rx_cmd_sabm_var := '0';
         tx_cmd_srej_var := (others=>'0');
         tx_cmd_ua <= '0' ;
      elsif rising_edge(clk) then
         if rx_cmd_reset_int = '1' then
            fifo_addr_w_var := (others=>'0') ;
            phy_bytesel_var := '0';
            phy_dvalid_old <= '0' ;
            fifo_addr_w_last_var := (others=>'0');
```

```
      phy_data_old <= (others=>'0') ;
      phy_data_lowbyte <= (others=>'0') ;
      phy_data_old2 <= (others=>'0') ;
      overflow_var := '0' ;
      rx_cmd_reset_var := '0';
      rx_cmd_test_var := '0';
      rx_vr_var := (others=>'0');
      rx_cmd_sabm_var := '0';
      tx_cmd_ua <='1' ;
   else
      rx_cmd_reset_var := '0';
      rx_cmd_test_var := '0';
      tx_cmd_srej_var := (others => '0');
      rx_cmd_srej_var :=  (others => '0');
      rx_cmd_sabm_var := '0';
      rx_cmd_ua_var := '0';
      tx_cmd_ua <= rx_cmd_sabm_int ;
      phy_data_old <= phy_data_old  ;
      fifo_addr_w_last_var := fifo_addr_w_last;
      phy_bytesel_var := phy_bytesel;
      fifo_addr_w_var := fifo_addr_w_int ;
      phy_data_lowbyte <= phy_data_lowbyte  ;
      phy_data_old2 <= phy_data_old2  ;
      packet_length_var := packet_length;
      phy_dvalid_old <= phy_dvalid  ;
      overflow_var := overflow;
      rx_vr_var := rx_vr_int;

      if (tx_cmd_reset = '1') or (rx_cmd_sabm_int ='1') then
         rx_vr_var := (others => '0');
      end if;

      if (phy_dvalid = '1' ) then
         if (phy_dstrobe = '1') then
            if (phy_bytesel = '1') then
               if (fifo_full = '1' or  overflow = '1') then
                  overflow_var := '1';
               else
                  fifo_addr_w_var := std_logic_vector(unsigned(fifo_addr_w_int) + 1);

                  -- fifo_addr_w_int <= fifo_addr_w_int + '1' ;
               end if;
               phy_data_old <= phy_data & phy_data_lowbyte ;
               phy_data_old2 <= phy_data_old;
               phy_bytesel_var := '0';
               packet_length_var :=  std_logic_vector(unsigned(packet_length) + 1);
               if (packet_length= (packet_length'range=>'0')) and (HEADER_FIELD /= 0) then -- HEADER
                  header <= phy_data ;
               end if;
            else
               phy_data_lowbyte <= phy_data ;
               phy_bytesel_var :='1';
            end if;
         end if;
      else
      --
         packet_length_var :=(others => '0');
         if (fifo_full='0') then
          overflow_var := '0';
         end if;
```

```
            phy_bytesel_var := '0';
         end if;


         if ((phy_dvalid='0') and phy_dvalid_old ='1') then
            if (crc_zero='1') then
               if ((to_integer(unsigned(packet_length)))=2) and (HEADER_FIELD /= 0 ) then
                  -- COMMAND FRAME
                  fifo_addr_w_var :=fifo_addr_w_last;


                  case phy_data_old2(15 downto 8) is
                  when x"8f" =>
                     rx_cmd_reset_var :='1';
                  when x"2f" =>
                     rx_cmd_sabm_var :='1';
                  when x"63" =>
                     rx_cmd_ua_var :='1';
                  when x"e3" =>
                     rx_cmd_test_var :='1';
                  when others =>
                     null;
                  end case;
                  --if (phy_data_old2_i[15:11] == 5'h0d) rx_cmd_srej_o[2:0] <=#1 phy_data_old2_i[10:8];
               else
                  if ((HEADER_FIELD=0) or ((HEADER_FIELD/=0) and (header(0)='0'))) then
                     -- INFORMATION FRAME END
                     fifo_addr_w_last_var :=  std_logic_vector(unsigned(fifo_addr_w_int)-1);
                     fifo_addr_w_var := std_logic_vector(unsigned(fifo_addr_w_int)-1) ;
                     -- Receive state variable:
                     rx_vr_var := std_logic_vector(unsigned(header(3 downto 1))+1);
                     if ( header(3 downto 1) = rx_vr_int) then
                        tx_cmd_srej_var :='1' & rx_ns_minus_one & rx_vr_int;
                     end if;

                     rx_nr_o <= header(7 downto 5);
                  else -- SREJ FRAME END
                     fifo_addr_w_var := fifo_addr_w_last ;
                     rx_cmd_srej_var :=  '1' & phy_data_old2(7 downto 5) & phy_data_old2(3 downto 1);
                  end if;
               end if;
            else -- CRC FAILED
               fifo_addr_w_var := fifo_addr_w_last ;
            end if;
         end if;
      end if;
   end if;



   rx_cmd_reset <= rx_cmd_reset_var;
   rx_cmd_reset_int <= rx_cmd_reset_var;
   --
   rx_cmd_sabm <= rx_cmd_sabm_var;
   rx_cmd_sabm_int <= rx_cmd_sabm_var;

   --
   rx_cmd_ua <= rx_cmd_ua_var;
   rx_cmd_test <= rx_cmd_test_var;
   fifo_addr_w_last <= fifo_addr_w_last_var;
```

```
      fifo_addr_w_int <= fifo_addr_w_var;

      rx_vr_int <= rx_vr_var;
      rx_vr_o <= rx_vr_var;

      tx_cmd_srej <= tx_cmd_srej_var;
      rx_cmd_srej <= rx_cmd_srej_var;

      packet_length <= packet_length_var;
      overflow <= overflow_var;
      phy_bytesel <= phy_bytesel_var;
   end process sync_process;


   combinatorial_logic : process ( fifo_addr_w_last, fifo_full, phy_bytesel,
      phy_data, phy_data_lowbyte, phy_data_old2, phy_dstrobe, phy_dvalid,
      phy_dvalid_old, fifo_addr_w_int) is
      variable fifo_w_int        :  std_logic;
      variable fifo_addr_w_var   :  std_logic_vector(ADDR_WIDTH-1 downto 0);
      variable fifo_data_w_var   :  std_logic_vector(16 downto 0);
   begin
      fifo_w_int := '0';
      fifo_addr_w_var := fifo_addr_w_int;
      fifo_data_w_var := '0' & phy_data & phy_data_lowbyte;

      if ((phy_dstrobe = '1') and (phy_dvalid = '1') and (fifo_full='0')) then
         if (phy_bytesel = '1') then
            fifo_w_int :='1';
         end if;
      end if;

      if (phy_dvalid='0') and (phy_dvalid_old='1') then
         if not ( (to_integer( unsigned(fifo_addr_w_int)-unsigned(fifo_addr_w_last)) =2) and
     (HEADER_FIELD/=0)) then
            fifo_addr_w_var := std_logic_vector(unsigned(fifo_addr_w_int)-2);
            fifo_data_w_var := '1' & phy_data_old2;
            fifo_w_int := '1';
         end if;
      end if;
      fifo_w <= fifo_w_int;
      --
      fifo_addr_w_o <= fifo_addr_w_var;
      --
      fifo_data_w <= fifo_data_w_var;
   end process combinatorial_logic;

-- FIFO READ INTERFACE
fifo_read_interface : process (clk, disconnectb_resetb) is
   variable fifo_addr_r_var   :  std_logic_vector(ADDR_WIDTH-1 downto 0);
   variable rx_ena_pre_var :  std_logic;
   variable rx_sop_var     :  std_logic;
   variable rx_ena_var     :  std_logic;
   variable rx_adr_var     :  std_logic_vector(7 downto 0);
   variable rx_eop_var     :  std_logic;
   variable rx_dat_var     :  std_logic_vector(15 downto 0);
   variable rx_ns_var      :  std_logic_vector(2 downto 0);
begin
   if disconnectb_resetb = '0' then
      fifo_addr_r_var := (others =>'0') ;
      --rx_sop_o <=#1 0;
```

```
   --rx_ena_o <=#1 0;
   --rx_adr_o <=#1 0;
   rx_ena_pre_var :='0';
elsif rising_edge(clk) then
   if rx_cmd_reset_int = '1' then
      fifo_addr_r_var := (others =>'0');
      rx_sop_var := '0';
      rx_ena_var := '0';
      rx_adr_var := (others => '0');
      rx_ena_pre_var := '0';
   else
      rx_adr_var := rx_adr_int;
      rx_ena_var := rx_ena_pre_int;
      fifo_addr_r_var := fifo_addr_r_int;
      rx_ena_pre_var := rx_ena_pre_int;

      rx_sop_var := '0';
      rx_eop_var := fifo_data_r(16) and rx_ena_pre_int;

      if (( rx_ena_pre_int='0') and (fifo_nItems_complete /= (fifo_nItems_complete'range => '0')) and
  (rx_dav = '1')) then
         rx_ena_pre_var := '1';
         rx_dat_var := fifo_data_r(15 downto 0);
         fifo_addr_r_var := std_logic_vector(unsigned(fifo_addr_r_int) + 1);
         if (HEADER_FIELD /=0) then
            rx_adr_var := fifo_data_r(7 downto 0);
            rx_ns_var := fifo_data_r(11 downto 9);
         else
            rx_sop_var :='1';
            rx_ena_var := '1';
            rx_eop_var := fifo_data_r(16);
         end if;
      end if;

      if (rx_eop_int='1' and rx_ena_int='1') then
         rx_ena_var  := '0';
         rx_ena_pre_var :='0';
      end if;

      if (rx_ena_pre_int ='1') then
         if (rx_ena_int='0') then
            rx_sop_var := '1';
         end if;
         rx_dat_var := fifo_data_r(15 downto 0);
         if (rx_eop_int='0') then
            fifo_addr_r_var := std_logic_vector(unsigned(fifo_addr_r_int) + 1);
         end if;
      end if;

      if (fifo_nItems_complete = (fifo_nItems_complete'range=>'0') ) then
         fifo_addr_r_var :=  fifo_addr_r_int;
         rx_ena_pre_var := '0';
         rx_ena_var := '0';
      end if;
   end if;
end if;

fifo_addr_r <= fifo_addr_r_var;
fifo_addr_r_int <=  fifo_addr_r_var;
--
```

```
   rx_ena_pre <= rx_ena_pre_var;
   rx_ena_pre_int <= rx_ena_pre_var;
   ---
   rx_sop <= rx_sop_var;
   rx_ena <= rx_ena_var;
   --
   rx_adr <= rx_adr_var;
   rx_adr_int <= rx_adr_var;
   --
   rx_eop <= rx_eop_var;
   rx_eop_int <= rx_eop_var;
   --
   rx_dat <= rx_dat_var;
   rx_ns <= rx_ns_var;
   --
   rx_ena_int <= rx_ena_var;


end process fifo_read_interface;

   fifo_mem_inst : component fifo_mem
      generic map(DATA_WIDTH => 17,
                  ADDR_WIDTH => ADDR_WIDTH)
      port map(addr_w   => fifo_addr_w_o,
               addr_r   => fifo_addr_r,
               data_r   => fifo_data_r,
               data_w   => fifo_data_w,
               w_enable => fifo_w,
               clk_w    => clk);


end architecture RTL;
```

```
--------------------------------------------------------------------------------
--! @file MAC_tx.vhd
--! @author Cairo Caplan <cairo.caplan@cern.ch>
--! @date April, 2016
--------------------------------------------------------------------------------

-- Original Comments:
-- // missing: tx_ena_old_o <=#1 tx_ena_old_i;
--
--Problems:
--
--
--

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity MAC_tx is
   generic(
      ADDR_WIDTH        :  integer := 5;
      THRESHOLD         :  integer := 6;
      HEADER_FIELD      :  integer  := 1
   );


   port (
      clk               :  in std_logic;
      resetb            :  in std_logic;
      ---backend side:
      tx_dat            :  in std_logic_vector(15 downto 0);
      tx_ena            :  in std_logic;
      tx_dav            :  out   std_logic;
      tx_adr            :  in std_logic_vector(7 downto 0);
      tx_nr          :  in std_logic_vector(2 downto 0);
      tx_ns          :  out   std_logic_vector(2 downto 0);
      tx_cmd_srej       :  in std_logic_vector(6 downto 0);
      tx_cmd_reset      :  in std_logic;
      tx_cmd_test       :  in std_logic;
      --phy side:
      --phy_bytesel        :  out   std_logic;
      phy_dvalid        :  out   std_logic;
      phy_tx_data       :  out   std_logic_vector(7 downto 0);
      phy_dstrobe       :  in std_logic;
      ------------------------------------
      crc               :  in std_logic_vector(15 downto 0);
      crc_strobe        :  out   std_logic;
      tx_cmd_sabm       :  in std_logic;
      tx_cmd_ua         :  in std_logic;
      cmd_busy       :  out   std_logic;
      rx_cmd_reset      :  in std_logic
   );
end entity MAC_tx;



architecture RTL of MAC_tx is
```

```
   constant IDLE     :  integer  := 0;
   constant START    :  integer  := 1;
   constant DATA     :  integer  := 2;
   constant END_state   :  integer  := 3;

   signal resetb_rx_cmd_resetb   :std_logic;

   signal fifo_nItems         :  std_logic_vector(ADDR_WIDTH-1 downto 0);

   signal tx_ena_old    :  std_logic_vector(1 downto 0);
   signal tx_dat_old    :  std_logic_vector(15 downto 0);
   signal overflow         :  std_logic;
   signal fifo_addr_w_last_int   :  std_logic_vector(ADDR_WIDTH-1 downto 0);
   signal state_int        :  std_logic_vector(1 downto 0);
   signal issue_cmd_int    :  std_logic_vector(11 downto 0);
   signal cmd_in_progress_int :  std_logic;

   signal fifo_addr_w_int     :  std_logic_vector(ADDR_WIDTH-1 downto 0);
   signal fifo_addr_r_int     :  std_logic_vector(ADDR_WIDTH-1 downto 0);

   signal tx_ns_int        :  std_logic_vector(2 downto 0);
   signal phy_dvalid_int      :  std_logic;
   signal phy_bytesel_int     :  std_logic;



   signal fifo_data_w         :  std_logic_vector(16 downto 0);
   signal fifo_data_r         :  std_logic_vector(16 downto 0);
   signal fifo_addr_w_o    :  std_logic_vector(ADDR_WIDTH-1 downto 0);
   signal fifo_addr_r_o    :  std_logic_vector(ADDR_WIDTH-1 downto 0);
   signal fifo_w           :  std_logic;

   component fifo_mem
      generic(DATA_WIDTH : integer := 4;
              ADDR_WIDTH : integer := 3);
      port(addr_w, addr_r  : in  std_logic_vector(ADDR_WIDTH - 1 downto 0);
           data_r          : out std_logic_vector(DATA_WIDTH - 1 downto 0);
           data_w          : in  std_logic_vector(DATA_WIDTH - 1 downto 0);
           w_enable, clk_w : in  std_logic);
   end component fifo_mem;

begin

   cmd_busy <=  issue_cmd_int(0);
   resetb_rx_cmd_resetb <= resetb and ( not rx_cmd_reset);
   --Get the number of free positions available in the fifo
   fifo_nItems <= std_logic_vector(unsigned(fifo_addr_w_int) - unsigned(fifo_addr_r_int));



   synchronous_process : process (clk, resetb_rx_cmd_resetb) is

      variable fifo_addr_w_var         :  std_logic_vector(ADDR_WIDTH-1 downto 0);
      variable tx_ena_old_var       :  std_logic_vector(1 downto 0);
      variable tx_dat_old_var       :  std_logic_vector(15 downto 0);
      variable tx_ns_var            :  std_logic_vector(2 downto 0);
      variable overflow_var            :  std_logic;
      --variable fifo_addr_w_last_o_int      :  std_logic_vector(ADDR_WIDTH-1 downto 0);
```

```
begin

   --If resetting
   if resetb_rx_cmd_resetb = '0' then
      fifo_addr_w_var := (others=>'0');
      tx_ena_old_var := (others=>'0');
      tx_dat_old_var := (others=>'0');
      tx_ns_var := (others=>'0');
      overflow_var := '0';
   elsif rising_edge(clk) then
      tx_ns_var := tx_ns_int;
      tx_dat_old_var := tx_dat_old;
      fifo_addr_w_var := fifo_addr_w_int;
      overflow_var := overflow;
      tx_ena_old_var := tx_ena_old(0) & tx_ena;

      --If tx_ena stands high and
      if ( ((tx_ena='1') or (tx_ena_old(0)='1' )) and (HEADER_FIELD/=0)) or ((tx_ena='1') and
  (HEADER_FIELD=0)) then
         tx_dat_old_var := tx_dat;
         if ((fifo_nItems /= (fifo_nItems'range=>'1')) and (overflow='0')) then
            fifo_addr_w_var := std_logic_vector(unsigned(fifo_addr_w_int) + 1);
         else
            overflow_var := '1';
            fifo_addr_w_var := fifo_addr_w_last_int;
         end if;
      end if;

      -- if tx_ena is being activated
      if ((tx_ena='0') and (tx_ena_old(0)='1')) then
         tx_ns_var := std_logic_vector(unsigned(tx_ns_int) + 1);
      end if;
      if (tx_cmd_reset = '1' or tx_cmd_sabm = '1') then
         tx_ns_var := (others=>'0');
      end if;

      if (tx_ena_old(0)='0') then
         fifo_addr_w_last_int <= fifo_addr_w_int;
      end if;


      if ((tx_ena='0') and (fifo_nItems /= (fifo_nItems'range=>'1'))) then
         overflow_var := '0';
      end if;

   end if;

   fifo_addr_w_o <= fifo_addr_w_var;
   fifo_addr_w_int <= fifo_addr_w_var;
   --
   tx_ena_old <= tx_ena_old_var;
   tx_dat_old <= tx_dat_old_var;
   --
   tx_ns <= tx_ns_var;
   tx_ns_int <= tx_ns_var;
   --
   overflow <= overflow_var;
   --fifo_addr_w_last_int <= fifo_addr_w_last_o_int;

end process synchronous_process;
```

```
tx_dav <= '0' when ((to_integer(unsigned(fifo_nItems)) >= THRESHOLD) or (tx_ena='1') or
  (tx_ena_old(0)='1')
      or (tx_ena_old(1)='1' and (HEADER_FIELD/=0)))
      else '1';



state_process : process (clk, resetb_rx_cmd_resetb) is
   variable fifo_addr_r_var      :  std_logic_vector(ADDR_WIDTH-1 downto 0);
   variable phy_bytesel_var      :  std_logic;
   variable phy_dvalid_var       :  std_logic;
   variable state_var            :  std_logic_vector(1 downto 0);
   variable issue_cmd_var        :  std_logic_vector(11 downto 0);
   variable cmd_in_progress_var  :  std_logic;
begin
   if resetb_rx_cmd_resetb = '0' then
      fifo_addr_r_var := (others=>'0');
      phy_bytesel_var := '0';
      phy_dvalid_var := '0';
      state_var := (others=>'0');
      issue_cmd_var  := (others=>'0');
   elsif rising_edge(clk) then
      state_var := state_int;
      phy_dvalid_var := phy_dvalid_int;
      phy_bytesel_var := phy_bytesel_int;
      fifo_addr_r_var := fifo_addr_r_int;
      issue_cmd_var := issue_cmd_int;
      cmd_in_progress_var := cmd_in_progress_int;

      if (issue_cmd_int = (issue_cmd_int'range=>'0') ) then
         if (tx_cmd_ua = '1' and HEADER_FIELD /= 0) then
            issue_cmd_var := x"063"; end if;
         if ((tx_cmd_srej /= (tx_cmd_srej'range=>'0')) and HEADER_FIELD /= 0) then
            issue_cmd_var := tx_cmd_srej & '0' & x"d"; end if;
         if (tx_cmd_sabm = '1' and HEADER_FIELD /= 0) then
            issue_cmd_var := x"02f"; end if;
         if (tx_cmd_test = '1' and HEADER_FIELD /= 0) then
            issue_cmd_var :=  x"0e3"; end if;
         if (tx_cmd_reset = '1' and HEADER_FIELD /= 0) then
            issue_cmd_var := x"08f"; end if;
      end if;


      if ((phy_dstrobe ='1') and (phy_dvalid_int='1')) then
         if (phy_bytesel_int = '1') then
            phy_bytesel_var := '0';
         else
            phy_bytesel_var := '1';
         end if;
      end if;
      --if on IDLE state
      case to_integer(unsigned(state_int)) is
         when IDLE =>
            phy_dvalid_var := '0';
            if (phy_dvalid_int='0') and
            ((fifo_nItems /= (fifo_nItems'range => '0')) or (issue_cmd_int /= (issue_cmd_int'range
```

```
=>'0'))) then
              cmd_in_progress_var := issue_cmd_int(0);
              phy_dvalid_var := '1';
              state_var := std_logic_vector( to_unsigned( START, state_var'length ));
           end if;
        when START=>
           phy_dvalid_var := '1';
           if (phy_dstrobe='1') then
              if (phy_bytesel_int='1') then
                 state_var :=  std_logic_vector( to_unsigned( DATA, state_var'length ));
                 if (cmd_in_progress_int='0') then
                    fifo_addr_r_var := std_logic_vector(unsigned(fifo_addr_r_int) + 1);
                 else
                    if (issue_cmd_int(11 downto 8)="0000") then
                       state_var := std_logic_vector( to_unsigned( END_state, state_var'length ));
                    end if;
                 end   if;
              end if;
           end if;
        when DATA=>
           phy_dvalid_var := '1';
           if (phy_dstrobe='1') then
              if (phy_bytesel_int='1') then
                 if (cmd_in_progress_int='0') then
                    fifo_addr_r_var := std_logic_vector(unsigned(fifo_addr_r_int) + 1);
                 else
                    state_var :=  std_logic_vector( to_unsigned(END_state, state_var'length ));
                 end if;
              end if;
           end if;

           if ((cmd_in_progress_int='0') and ((fifo_data_r(16)='1') or
(fifo_nItems=(fifo_nItems'range=>'0')))) then
              state_var := std_logic_vector( to_unsigned(END_state, state_var'length ));
           end   if;
        when END_state =>
           if (phy_dstrobe='1') then
              if (phy_bytesel_int='1') then
                 state_var := std_logic_vector( to_unsigned(IDLE, state_var'length ));
                 if (cmd_in_progress_int='1') then
                    cmd_in_progress_var := '0';
                    issue_cmd_var := (others=>'0');
                 end if;
              end if;
           end if;
        when others=>
           null;
     end case;
  end if;

  fifo_addr_r_o <= fifo_addr_r_var;
  fifo_addr_r_int <= fifo_addr_r_var;
  --
  --phy_bytesel <= phy_bytesel_var;
  phy_bytesel_int <= phy_bytesel_var;
  --
  phy_dvalid <= phy_dvalid_var;
  phy_dvalid_int <= phy_dvalid_var;
  --
  state_int   <= state_var;
```

```
   --
   issue_cmd_int <= issue_cmd_var;

   --
   cmd_in_progress_int <= cmd_in_progress_var;

end process state_process;



combinational_logic : process (cmd_in_progress_int, crc,
   fifo_data_r,
   issue_cmd_int,
    phy_bytesel_int, phy_dstrobe, phy_dvalid_int, state_int, tx_adr,
     tx_dat, tx_dat_old, tx_ena, tx_ena_old(0), tx_nr, tx_ns_int
) is
   variable phy_tx_data16  :  std_logic_vector(15 downto 0);
   variable fifo_w_int     :  std_logic;
   variable fifo_data_w_int   :  std_logic_vector(16 downto 0);
begin
   case to_integer(unsigned(state_int)) is
   when START =>
      if HEADER_FIELD/=0 then
         phy_tx_data16 := tx_nr & '0' & fifo_data_r(11 downto 9) & '0' & fifo_data_r(7 downto 0);
         if cmd_in_progress_int='1' then
            phy_tx_data16 :=  issue_cmd_int(7 downto 0) & x"ff";
         end if;
      else
         phy_tx_data16 := fifo_data_r(15 downto 0);
      end if;
   when DATA =>
      phy_tx_data16 :=  fifo_data_r(15 downto 0);
      if (cmd_in_progress_int='1') then
         phy_tx_data16(15 downto 8) := (others=>'0');
         phy_tx_data16(7 downto 0) := issue_cmd_int(10 downto 8) & '1' & issue_cmd_int(7 downto 5) &
 '1';
      end if;
   when END_state =>
      phy_tx_data16 := crc;
   when others =>
      phy_tx_data16 := fifo_data_r(15 downto 0);
   end case;

   if phy_bytesel_int= '1' then
      phy_tx_data <= phy_tx_data16(15 downto 8);
   else
      phy_tx_data <= phy_tx_data16(7 downto 0);
   end if;

   if to_integer(unsigned(state_int)) /=END_state then
      crc_strobe <=  phy_dstrobe and phy_dvalid_int and '1';
   else
      crc_strobe <=  phy_dstrobe and phy_dvalid_int and '0';
   end if;

   fifo_w_int :='0';
   fifo_data_w_int := '0' & tx_dat_old; -- fifo_data_w = tx_dat_old_i;
   if (HEADER_FIELD/=0) then
      if tx_ena='1' and ( tx_ena_old(0)='0') then
```

```
          fifo_data_w_int(7 downto 0) := tx_adr;
          fifo_data_w_int(8) := '0';
          fifo_data_w_int(11 downto 9) := tx_ns_int;
          fifo_data_w_int(16 downto 12) := "10000";
           --fifo_data_w_int := '1' & "0000" & tx_ns_int & '0' & tx_adr;
           fifo_w_int := '1';
       end if;
       if tx_ena_old(0)='1' then
          fifo_w_int :='1';
       end if;
    else
       if tx_ena='1' then
          fifo_data_w_int := (not tx_ena_old(0)) & tx_dat;
          fifo_w_int := '1';
       end if;
    end if;

     fifo_w <= fifo_w_int;
     --
     fifo_data_w <= fifo_data_w_int;


  end process combinational_logic;



  fifo_mem_inst : component fifo_mem
     generic map(DATA_WIDTH => 17,
                 ADDR_WIDTH => ADDR_WIDTH)
     port map(addr_w   => fifo_addr_w_o,
              addr_r   => fifo_addr_r_o,
              data_r   => fifo_data_r,
              data_w   => fifo_data_w,
              w_enable => fifo_w,
              clk_w    => clk);


end architecture RTL;
```

```
--------------------------------------------------------------------------------
--! @file monostable.vhd
--! @author Cairo Caplan <cairo.caplan@cern.ch>
--! @date April, 2016
--------------------------------------------------------------------------------

library ieee;
use ieee.std_logic_1164.all;

entity monostable is
   port (
      clk   :  in std_logic;
      A  :  in std_logic;
      Z  :  out std_logic
   );
end entity monostable;


--! Pulse generator with one clock period width, extracted
--! from "Use Synchronous Pulse Generators" in:
--! http://www.altera.com/literature/hb/qts/qts_qii51006.pdf
architecture RTL of monostable is


   component regbank
      generic(WIDTH : integer := 8);
      port(d   : in  std_logic_vector(WIDTH - 1 downto 0);
          rn  : in  std_logic;
          q   : out std_logic_vector(WIDTH - 1 downto 0);
          clk : in  std_logic);
   end component regbank;

   signal t1, t2 : std_logic;
begin

   sync_delay : process (clk) is
   begin
      if rising_edge(clk) then
         t1<= A;
         t2 <= t1;
      end if;
   end process sync_delay;

   Z <= t1 and (not t2);



end architecture RTL;
```

```
--------------------------------------------------------------------------------
--! @file PHY_HDLC_rx.vhd
--! @author Cairo Caplan <cairo.caplan@cern.ch>
--! @date April, 2016
--------------------------------------------------------------------------------

--Original comments:
--
--// missing: rx_dstrobe_o <=#1 rx_dstrobe_i;
--
--Problems:
-- signals not used: rx_dstrobe_i
--



library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity PHY_HDLC_rx is
   port (
         resetb      :  in std_logic;
         --optical link side:
         rx_clk      :  in std_logic;
         rx_sdr      :  in std_logic_vector(1 downto 0);
         --backend side:
         tx_clk      :  in std_logic;
         rx_data     :  out   std_logic_vector(7 downto 0);
         rx_dvalid   :  out std_logic;
         rx_dstrobe  :  out   std_logic
   );
end entity PHY_HDLC_rx;

architecture RTL of PHY_HDLC_rx is

   signal   rx_dvalid_int  :  std_logic;
   signal   rx_data_int    :  std_logic_vector(7 downto 0);
   signal   rx_reg_int     :  std_logic_vector(8 downto 0);
   signal   ones_count_int :  std_logic_vector(2 downto 0) := (others => '0');
   signal   bit_count_int  :  std_logic_vector(3 downto 0) := (others => '0');
   signal   start_int      :  std_logic;

   signal rx_sdr_buf1,rx_sdr_buf2,rx_sdr_sync   :std_logic_vector(1 downto 0);
begin

   phy_hdlc_rx : process (tx_clk, resetb) is
      variable rx_reg_var     :  std_logic_vector(8 downto 0);
      variable ones_count_var :  std_logic_vector(2 downto 0)  := (others => '0');
      variable rx_data_var :  std_logic_vector(7 downto 0);
      variable bit_count_var  :  std_logic_vector(3 downto 0) := (others => '0');
      variable rx_dvalid_var  :  std_logic;
      variable rx_dstrobe_var :  std_logic;
      variable start_var      :  std_logic;
   begin
      if resetb = '0' then
--       //rx_reg_o_int := 0;
--       //ones_count_o_int := 0;
--       //bit_count_o_int := 0;
```

```
--       //rx_dvalid_o_int := 0;
--       //rx_dstrobe_o_int := 0;
--       //start_o_int := 0;
--       //rx_dvalid_o_int := 0;
--       //rx_data_o_int := 0;
      elsif rising_edge(tx_clk) then
         rx_data_var := rx_data_int;
         start_var := start_int;
         rx_dvalid_var := rx_dvalid_int;
         bit_count_var := bit_count_int;
         ones_count_var := ones_count_int;
         rx_reg_var := rx_reg_int;
         rx_dstrobe_var := '0';

         if (rx_dvalid_int='1') then
            start_var := '0';
         end if;
         rx_reg_var := rx_sdr_sync(0) & rx_sdr_sync(1) & rx_reg_int(8 downto 2);
         bit_count_var := std_logic_vector(unsigned(bit_count_int) + 2);
         if ((to_integer(unsigned(bit_count_int))=8) and rx_dvalid_int='1') then
            rx_data_var := rx_reg_int(8 downto 1);
            bit_count_var :=  std_logic_vector(to_unsigned( 2 ,bit_count_var'length));
            rx_dstrobe_var := '1';
         end if;
         if ((to_integer(unsigned(bit_count_int))=9) and (rx_dvalid_int='1')) then
            rx_data_var := rx_reg_int(7 downto 0);
            bit_count_var :=  std_logic_vector(to_unsigned( 3 ,bit_count_var'length));
            rx_dstrobe_var := '1';
         end if;


         if ((to_integer(unsigned(ones_count_int))=4) and (rx_sdr_sync(0)='0') and (rx_sdr_sync(1)='1'))
      then
            rx_reg_var := rx_sdr_sync(1) & rx_reg_int(8 downto 1);
            bit_count_var := std_logic_vector(unsigned(bit_count_int) + 1);
            if (to_integer(unsigned(bit_count_int))=8) then
               bit_count_var := std_logic_vector(to_unsigned( 1 ,bit_count_var'length));
               rx_data_var := rx_reg_int(8 downto 1);
               rx_dstrobe_var := '1';
            end if;
            if (to_integer(unsigned(bit_count_int))=9) then
               bit_count_var := std_logic_vector(to_unsigned( 2 ,bit_count_var'length));
               rx_data_var := rx_reg_int(7 downto 0);
               rx_dstrobe_var := '1';
            end if;
         end if;

         if ((to_integer(unsigned(ones_count_int))=5) and (rx_sdr_sync(1)='0')) then
            rx_reg_var := rx_sdr_sync(0) & rx_reg_int(8 downto 1);
            bit_count_var :=  std_logic_vector(unsigned(bit_count_int) + 1);
            if (to_integer(unsigned(bit_count_int))=8) then
               bit_count_var := std_logic_vector(to_unsigned( 1 ,bit_count_var'length));
               rx_data_var := rx_reg_int(8 downto 1);
               rx_dstrobe_var := '1';
            end if;
            if (to_integer(unsigned(bit_count_int))=9) then
               bit_count_var := std_logic_vector(to_unsigned( 2 ,bit_count_var'length));
               rx_data_var := rx_reg_int(7 downto 0);
               rx_dstrobe_var := '1';
            end if;
```

```
      end if;

      if (rx_sdr_sync(0)='1' and rx_sdr_sync(1)='1') then
         if (to_integer(unsigned(ones_count_int))<6) then
            ones_count_var := std_logic_vector(unsigned(ones_count_int) + 2);
         end if;
         if (to_integer(unsigned(ones_count_int))=6) then
            ones_count_var := std_logic_vector(to_unsigned( 7 , ones_count_var'length));
         end if;
      else
         if (start_int='1')then
            rx_dvalid_var := '1';
         end if;
         if (rx_sdr_sync(0)='0') then
            ones_count_var := std_logic_vector(to_unsigned( 0 , ones_count_var'length));
         else
            ones_count_var := std_logic_vector(to_unsigned( 1 , ones_count_var'length));
         end if;
      end if;



      if (to_integer(unsigned(ones_count_int))=5) and (rx_sdr_sync(0)='0') and (rx_sdr_sync(1)='1') then
         rx_dvalid_var := '0';
         bit_count_var:= std_logic_vector(to_unsigned( 0 ,bit_count_var'length));
         start_var := '1';
      end if;

      if (to_integer(unsigned(ones_count_int))=6) then
         rx_dvalid_var := '0';
         bit_count_var:= std_logic_vector(to_unsigned( 1 ,bit_count_var'length));
         if (rx_sdr_sync(1)='0') then
            start_var := '1';
         else
            start_var := '0';
         end if;
      end if;

      if (to_integer(unsigned(ones_count_int))=7) then
         rx_dvalid_var := '0';
         bit_count_var:= std_logic_vector(to_unsigned( 0 ,bit_count_var'length));
         start_var:= '0';
      end if;

   end if;

   rx_reg_int <= rx_reg_var ;-- after 1 ns;
   --
   ones_count_int <= ones_count_var ;-- after 1 ns;
   --
   rx_data <= rx_data_var ;-- after 1 ns;
   rx_data_int <=  rx_data_var ;-- after 1 ns;
   --
   bit_count_int  <= bit_count_var ;-- after 1 ns;
   --
   rx_dvalid <= rx_dvalid_var ;-- after 1 ns;
   rx_dvalid_int <= rx_dvalid_var ;-- after 1 ns;
   --
   rx_dstrobe <= rx_dstrobe_var ;-- after 1 ns;
```

```
      --
      start_int <= start_var ;-- after 1 ns;

   end process phy_hdlc_rx;

-- synchronize_sdr_clock_domain : process (rx_clk, tx_clk) is
-- begin
--    if rising_edge(rx_clk) then
--       rx_sdr_buf1 <= rx_sdr;
--    end if;
--    if rising_edge(tx_clk) then
--       rx_sdr_buf2 <= rx_sdr_buf1;
--       rx_sdr_sync <= rx_sdr_buf2;
--    end if;
-- end process synchronize_sdr_clock_domain;
   rx_sdr_sync <= rx_sdr;



end architecture RTL;
```

```
--------------------------------------------------------------------------------
--! @file PHY_HDLC_tx.vhd
--! @author Cairo Caplan <cairo.caplan@cern.ch>
--! @date April, 2016
--------------------------------------------------------------------------------

--!-----------------------------------------
--!
--!   PHY_HDLC_tx in Vhdl
--!   Based on the work by Sandro Bonacini
--!
--!      Author Cairo Caplan, CBPF
--!
--!--------------------------------------------------------------------------------
--
--
--Original comments:
--// missing: tx_sdr_o <=#1 tx_sdr_i;
--// missing: tx_dstrobe_o <=#1 tx_dstrobe_i;
--
-- signals not used: tx_dstrobe_i

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity PHY_HDLC_tx is
   port (
      resetb            :  in std_logic;
      --link side:
      tx_clk            :  in std_logic;
      tx_sdr            :  out   std_logic_vector(1 downto 0);
      --mac side:
      tx_data           :  in std_logic_vector(7 downto 0);
      tx_dvalid         :  in    std_logic;
      tx_dstrobe        :  out   std_logic

   );
end entity PHY_HDLC_tx;


architecture RTL of PHY_HDLC_tx is
   constant IDLE     :  std_logic_vector(1 downto 0):= "00";
   constant START    :  std_logic_vector(1 downto 0):= "01";
   constant TX       :  std_logic_vector(1 downto 0):= "10";
   constant END_state   :  std_logic_vector(1 downto 0):= "11";

   signal state               :  std_logic_vector(1 downto 0);
   signal ones_count_int         :  std_logic_vector(2 downto 0);
   signal bit_count_int       :  std_logic_vector(3 downto 0);
   signal tx_sdr_int          :  std_logic_vector(1 downto 0);
   signal tx_reg_int          :  std_logic_vector(8 downto 0);
   signal tx_dvalid_internal_int :  std_logic;

begin


   phy_hdlc_tx : process (tx_clk, resetb) is
      variable tx_reg_var           :     std_logic_vector(8 downto 0);
```

```
    variable tx_sdr_var           :     std_logic_vector(1 downto 0);
    variable ones_count_var       :     std_logic_vector(2 downto 0);
    variable bit_count_var        :     std_logic_vector(3 downto 0);
    variable state_var            :     std_logic_vector(1 downto 0);
    variable tx_dstrobe_var       :     std_logic;
    variable tx_dvalid_internal_var  :     std_logic;
begin
    --if reset
    if resetb = '0' then
        tx_reg_var :=(others=>'1');
        tx_sdr_var(0) := '1';
        tx_sdr_var(1) := '1';
        --one_count=0 , bit_count=0, state=0, tx_dstrobe_o_int=0
        ones_count_var :=  std_logic_vector(to_unsigned( 0 , ones_count_var'length));
        bit_count_var := std_logic_vector(to_unsigned( 0 , bit_count_var'length));
        state_var :=  std_logic_vector(to_unsigned( 0  , state_var'length));
        tx_dstrobe_var := '0';
    --on rising edge of rx clk
    elsif rising_edge(tx_clk) then
        state_var := state;
        bit_count_var := bit_count_int;
        ones_count_var := ones_count_int;
        tx_sdr_var(1) := tx_sdr_int(1);
        tx_sdr_var(0) := tx_sdr_int(0);
        tx_reg_var := tx_reg_int;
        tx_dvalid_internal_var := tx_dvalid_internal_int;

        tx_dstrobe_var := '0';

        if (tx_reg_int(0)='1' and tx_reg_int(1)='1') then
            if (to_integer(unsigned(ones_count_int))<6) then
                ones_count_var := std_logic_vector(unsigned(ones_count_int) + 2);
            elsif (to_integer(unsigned(ones_count_int))=6) then
                ones_count_var := std_logic_vector(to_unsigned( 7 , ones_count_var'length));
            end if;
        else
            if (tx_reg_int(1)='0') then
                ones_count_var := std_logic_vector(to_unsigned( 0 , ones_count_var'length));
            else
                ones_count_var := std_logic_vector(to_unsigned( 1 , ones_count_var'length));
            end if;
        end if;

        if (tx_dvalid='0') then
            tx_dvalid_internal_var:= '0';
        end if;

        case (state) is
            --IDLE state is default
            when START =>
                bit_count_var := std_logic_vector(unsigned(bit_count_int) - 2);
                tx_reg_var := std_logic_vector(unsigned(tx_reg_int) srl 2);
                tx_sdr_var(0) := tx_reg_int(1);
                tx_sdr_var(1) := tx_reg_int(0);
                if ((to_integer(unsigned(bit_count_int))=2) or (to_integer(unsigned(bit_count_int))=1)) then
                    state_var := TX;
                    tx_reg_var(7 downto 0) := tx_data;
                    tx_dstrobe_var := '1';
                    bit_count_var:= std_logic_vector(to_unsigned( 8 , bit_count_var'length));
                end if;
```

```
      if (tx_dvalid='0') then
         state_var := IDLE;
      end if;

   when TX =>
      tx_reg_var := std_logic_vector(unsigned(tx_reg_int) srl 2);
      tx_sdr_var(0) := tx_reg_int(1);
      tx_sdr_var(1) := tx_reg_int(0);
      bit_count_var := std_logic_vector(unsigned(bit_count_int) - 2);

      if (to_integer(unsigned(ones_count_int))/=5)
         and ((to_integer(unsigned(ones_count_int))/=4)
            or (tx_reg_int(0)='0')) then
         if (to_integer(unsigned(bit_count_int))=3) then
            if (tx_dvalid_internal_int = '1' and tx_dvalid='1') then
               tx_dstrobe_var := '1';
               bit_count_var := std_logic_vector(to_unsigned( 9 , bit_count_var'length));
               tx_reg_var(8 downto 1) := tx_data;
            else
               bit_count_var := std_logic_vector(to_unsigned( 1 , bit_count_var'length));
               tx_reg_var(8 downto 1) := x"7e";
            end if;
         end if;
         if (to_integer(unsigned(bit_count_int))=2) then
            bit_count_var := std_logic_vector(to_unsigned( 8 , bit_count_var'length));
            if (tx_dvalid_internal_int ='1'  and tx_dvalid ='1') then
               tx_dstrobe_var := '1';
               tx_reg_var(7 downto 0) := tx_data;
            else
               tx_reg_var(8 downto 0) := "101111110";
               state_var := END_state;
            end if;
         end if;
      end if;


      if ((to_integer(unsigned(ones_count_int))=4) and tx_reg_int(0)='1') then
         tx_sdr_var(0) := '0';
         tx_sdr_var(1) := '1';
         tx_reg_var := std_logic_vector(unsigned(tx_reg_int) srl 1);
         ones_count_var := std_logic_vector(to_unsigned( 0 , ones_count_var'length));
         bit_count_var :=  std_logic_vector(unsigned(bit_count_int) - 1);
         if (to_integer(unsigned(bit_count_int))=2) then
            bit_count_var := std_logic_vector(to_unsigned( 9 , bit_count_var'length));
            if (tx_dvalid_internal_int = '1'  and tx_dvalid='1') then
               tx_reg_var(8 downto 1) := tx_data;
               tx_dstrobe_var := '1';
            else
               tx_reg_var(8 downto 1) := x"7e";
               state_var := END_state;
            end if;
         end if;
      end if;

      if (to_integer(unsigned(ones_count_int))=5) then
         tx_sdr_var(0) := tx_reg_int(0);
         tx_sdr_var(1) := '0';
         tx_reg_var := std_logic_vector(unsigned(tx_reg_int) srl 1);
```

```
            ones_count_var := (ones_count_var'range=>'0');
            ones_count_var(0) :=  tx_reg_int(0);

            bit_count_var := std_logic_vector(unsigned(bit_count_int) - 1);
            if (to_integer(unsigned(bit_count_int))=2) then
               bit_count_var := std_logic_vector(to_unsigned( 9 , bit_count_var'length));
               if (tx_dvalid_internal_int = '1' and tx_dvalid ='1') then
                  tx_dstrobe_var := '1';
                  tx_reg_var(8 downto 1) := tx_data;
               else
                  tx_reg_var(8 downto 1) := x"7e";
                  state_var := END_state;
               end if;
            end if;
         end if;

         if (to_integer(unsigned(bit_count_int))=1) then
            state_var := END_state;
            bit_count_var := std_logic_vector(to_unsigned( 7 , bit_count_var'length));
         end if;
      when END_state =>
         tx_reg_var := std_logic_vector(unsigned(tx_reg_int) srl 2);
         tx_sdr_var(0) := tx_reg_int(1);
         tx_sdr_var(1) := tx_reg_int(0);
         bit_count_var := std_logic_vector(unsigned(bit_count_int) - 2);
         if ((to_integer(unsigned(bit_count_int))=0) or (to_integer(unsigned(bit_count_int))=1)) then
            state_var := IDLE;
         end if;

         if ((to_integer(unsigned(ones_count_int))=5) and (tx_reg_int(0)='1')) then
            tx_sdr_var(0):= '0';
            if (tx_dvalid='1') then
               tx_dvalid_internal_var:= '1';
               state_var := TX;
               tx_reg_var(7 downto 0) := tx_data;
               tx_dstrobe_var := '1';
               bit_count_var:=  std_logic_vector(to_unsigned( 8 , bit_count_var'length));
            else
               tx_reg_var:= '0' & x"7f";
               state_var := IDLE;
            end if;
         end if;

         if (to_integer(unsigned(ones_count_int))=6) then
            tx_sdr_var(1):= '0';
            if (tx_dvalid='1') then
               tx_dvalid_internal_var:= '1';
               state_var := TX;
               tx_reg_var(6 downto 0) := tx_data(7 downto 1);
               tx_sdr_var(0) := tx_data(0);
               tx_dstrobe_var := '1';
               bit_count_var:=  std_logic_vector(to_unsigned( 7 , bit_count_var'length));

               ones_count_var := (others=>'0');--
               ones_count_var(0) := tx_data(0); --ones_count_o <=#1 tx_data[0];
            else
               tx_reg_var := '0' & x"7f";
               tx_sdr_var(0):= '1';
               state_var := IDLE;
            end if;
```

```
            end if;
         when others =>
            tx_reg_var := std_logic_vector(unsigned(tx_reg_int) srl 2);
            tx_sdr_var(0) := tx_reg_int(1);
            tx_sdr_var(1) := tx_reg_int(0);
            bit_count_var := std_logic_vector(to_unsigned( 8 , bit_count_var'length));
            tx_dvalid_internal_var := '0';
            if ((tx_reg_int(1)='0') or (tx_reg_int(0)='0'))then
               if (tx_dvalid='1') then
                  tx_sdr_var := "11";
                  tx_reg_var(7 downto 0) := x"7e";
                  bit_count_var :=  std_logic_vector(to_unsigned( 8 , bit_count_var'length));
                  state_var := START;
                  tx_dvalid_internal_var:= '1';
               else
                  tx_reg_var:= '0' & x"7f";
               end if;
            end if;
         end case;
      end if;


      tx_reg_int <= tx_reg_var;
      --
      tx_sdr <= tx_sdr_var;
      tx_sdr_int <= tx_sdr_var;
      --
      ones_count_int <= ones_count_var;
      bit_count_int <= bit_count_var;
      state <= state_var;
      --
      tx_dstrobe <= tx_dstrobe_var;
      --tx_dstrobe_int <= tx_dstrobe_var;
      --
      tx_dvalid_internal_int <= tx_dvalid_internal_var;

   end process phy_hdlc_tx;



end architecture RTL;
```

```
--------------------------------------------------------------------------------
--! @file regbank.vhd
--! @author Cairo Caplan <cairo.caplan@cern.ch>
--! @date April, 2016
--------------------------------------------------------------------------------
library ieee;
   use ieee.std_logic_1164.all;
   use ieee.numeric_std.all;


entity regbank is
   generic (
      WIDTH  : integer := 8
   );
   port (
      d      : in std_logic_vector(WIDTH - 1 downto 0);
      rn     : in std_logic;
      q      : out   std_logic_vector(WIDTH - 1 downto 0);
      clk    : in std_logic
   );
end regbank;

architecture arc of regbank is
begin

   process (clk, rn)
   begin
      if rn = '0' then
         q <= (others => '0');
      elsif rising_edge(clk) then
         q <= d;
      end if;
   end process;


end arc;
```

```
--------------------------------------------------------------------------------
--! @file rx.vhd
--! @author Cairo Caplan <cairo.caplan@cern.ch>
--! @date April, 2016
--------------------------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;

entity rx is
   generic(
      HEADER_FIELD   :  integer := 1;
      ADDR_WIDTH     :  integer  := 8
   );
   port (
      rx_dat      : out std_logic_vector(15 downto 0);
      rx_ena      : out std_logic;
      rx_eop      : out std_logic;
      rx_sop      : out std_logic;
      rx_err      : out std_logic;
      rx_adr      : out std_logic_vector(7 downto 0);
      rx_cmd_reset: out std_logic;
      rx_cmd_test : out std_logic;
      rx_cmd_ua   : out std_logic;
      tx_cmd_ua   : out std_logic;
      rx_cmd_srej : out std_logic_vector(6 downto 0);
      tx_cmd_srej : out std_logic_vector(6 downto 0);
      rx_vr    : out std_logic_vector(2 downto 0);
      rx_nr    : out std_logic_vector(2 downto 0);
      rx_ns    : out std_logic_vector(2 downto 0);
      clk         : in  std_logic;
      rx_clk      : in  std_logic;
      rx_dav      : in  std_logic;
      rx_sdr_pri  : in  std_logic_vector(1 downto 0);
      resetb      : in std_logic;
      tx_cmd_reset: in std_logic
   );
end entity rx;

architecture RTL of rx is
   --signal active_aux_int :  std_logic;

   component MAC_rx
      generic(ADDR_WIDTH   : integer := 5;
              HEADER_FIELD : integer := 1);
      port(rx_dat       : out std_logic_vector(15 downto 0);
          rx_ena       : out std_logic;
          rx_ena_pre   : out std_logic;
          rx_eop       : out std_logic;
          rx_err       : out std_logic;
          rx_vr_o      : out std_logic_vector(2 downto 0);
          rx_nr_o      : out std_logic_vector(2 downto 0);
          rx_sop       : out std_logic;
          rx_adr       : out std_logic_vector(7 downto 0);
          clk          : in  std_logic;
          rx_dav       : in  std_logic;
          resetb       : in  std_logic;
          rx_cmd_reset : out std_logic;
          rx_cmd_test  : out std_logic;
          phy_data     : in  std_logic_vector(7 downto 0);
```

```
        phy_dvalid   : in  std_logic;
        phy_dstrobe  : in  std_logic;
        crc_zero     : in  std_logic;
        rx_cmd_ua    : out std_logic;
        rx_cmd_srej  : out std_logic_vector(6 downto 0);
        tx_cmd_srej  : out std_logic_vector(6 downto 0);
        rx_ns        : out std_logic_vector(2 downto 0);
        rx_cmd_sabm  : out std_logic;
        tx_cmd_ua    : out std_logic;
        tx_cmd_reset : in  std_logic);
end component MAC_rx;

component crc
   generic(CRC_WIDTH  : natural                              := 16;
           DATA_WIDTH : natural                             := 16;
           INIT_VAL   : std_logic_vector(2 * 8 - 1 downto 0) := x"ffff";
           POLY       : std_logic_vector(2 * 8 - 1 downto 0) := x"8408");
   port(d       : in  std_logic_vector(DATA_WIDTH - 1 downto 0);
        init    : in  std_logic;
        d_valid : in  std_logic;
        clk     : in  std_logic;
        reset_b : in  std_logic;
        crc_o   : out std_logic_vector(CRC_WIDTH - 1 downto 0));
end component crc;

component PHY_HDLC_rx
   port(resetb     : in  std_logic;
        rx_clk     : in  std_logic;
        rx_sdr     : in  std_logic_vector(1 downto 0);
        tx_clk     : in  std_logic;
        rx_data    : out std_logic_vector(7 downto 0);
        rx_dvalid  : out std_logic;
        rx_dstrobe : out std_logic);
end component PHY_HDLC_rx;


component monostable
   port(clk : in  std_logic;
        A   : in  std_logic;
        Z   : out std_logic);
end component monostable;


signal phy_data_pri     : std_logic_vector(7 downto 0);
signal rx_cmd_srej_pri  : std_logic_vector(6 downto 0);
signal tx_cmd_srej_pri  : std_logic_vector(6 downto 0);
signal rx_vr_pri     : std_logic_vector(2 downto 0);
signal rx_nr_pri     : std_logic_vector(2 downto 0);
signal rx_ns_pri     : std_logic_vector(2 downto 0);
signal rx_dat_pri    : std_logic_vector(15 downto 0);
signal rx_adr_pri    : std_logic_vector(7 downto 0);
signal crc_pri       : std_logic_vector(15 downto 0);


--signal clk_activeb    : std_logic;
signal phy_dvalid_pri   : std_logic;
signal phy_dvalid_prib  : std_logic;

signal rx_cmd_sabm_pri  : std_logic;
```

```
   signal rx_cmd_reset_pri : std_logic;

   signal rx_ena_pri       : std_logic;

   signal rx_eop_pri       : std_logic;

   signal rx_sop_pri       : std_logic;

   signal rx_err_pri       : std_logic;
   signal rx_cmd_test_pri     : std_logic;
   signal rx_cmd_ua_pri    : std_logic;
   signal tx_cmd_ua_pri    : std_logic;
   signal crc_zero_pri     : std_logic;

   signal phy_dstrobe_pri     : std_logic;
   signal rx_ena_pre_pri      : std_logic;
   signal rx_cmd_reset_pri_monos : std_logic;

begin


   --clk_activeb <= not clk_active;

   phy_dvalid_prib <= not phy_dvalid_pri;


   rx_dat <=  rx_dat_pri;

   rx_ena <= rx_ena_pri;

   rx_eop <= rx_eop_pri;

   rx_sop <=   rx_sop_pri;

   rx_err <=  rx_err_pri;

   rx_adr <=  rx_adr_pri;

   rx_cmd_reset <=  rx_cmd_reset_pri_monos;

   rx_cmd_test <=  rx_cmd_test_pri;

   rx_cmd_ua <=   rx_cmd_ua_pri;

   tx_cmd_ua <=   tx_cmd_ua_pri;

   rx_cmd_srej <=  rx_cmd_srej_pri;

   tx_cmd_srej <=  tx_cmd_srej_pri;

   rx_vr <=   rx_vr_pri;

   rx_nr <=  rx_nr_pri;

   rx_ns <=   rx_ns_pri;

   crc_zero_pri <= '1' when crc_pri= (crc_pri'range=>'0') else '0';
```

```
   MAC_rx_pri : component MAC_rx
      generic map(ADDR_WIDTH   => ADDR_WIDTH,
                  HEADER_FIELD => HEADER_FIELD)
      port map(rx_dat       => rx_dat_pri,
               rx_ena       => rx_ena_pri,
               rx_ena_pre   => rx_ena_pre_pri,
               rx_eop       => rx_eop_pri,
               rx_err         => rx_err_pri,
               rx_vr_o        => rx_vr_pri,
               rx_nr_o        => rx_nr_pri,
               rx_sop       => rx_sop_pri,
               rx_adr       => rx_adr_pri,
               clk            => clk,
               rx_dav         => rx_dav,
               resetb         => resetb,
               rx_cmd_reset => rx_cmd_reset_pri,
               rx_cmd_test  => rx_cmd_test_pri,
               phy_data       => phy_data_pri,
               phy_dvalid     => phy_dvalid_pri,
               phy_dstrobe    => phy_dstrobe_pri,
               crc_zero       => crc_zero_pri,
               rx_cmd_ua    => rx_cmd_ua_pri,
               rx_cmd_srej  => rx_cmd_srej_pri,
               tx_cmd_srej  => tx_cmd_srej_pri,
               rx_ns        => rx_ns_pri,
               rx_cmd_sabm  => rx_cmd_sabm_pri,
               tx_cmd_ua    => tx_cmd_ua_pri,
               tx_cmd_reset   => tx_cmd_reset);



   crc_pri_inst : component crc
      generic map(
                  DATA_WIDTH => 8)
      port map(d       => phy_data_pri,
               init    => phy_dvalid_prib,
               d_valid => phy_dstrobe_pri,
               clk     => clk,
               reset_b => resetb,
               crc_o   => crc_pri);

   PHY_HDLC_rx_pri : component PHY_HDLC_rx
      port map(rx_data    => phy_data_pri,
              rx_clk     => rx_clk,
               rx_sdr       => rx_sdr_pri,
               tx_clk       => clk,
               rx_dvalid  => phy_dvalid_pri,
               rx_dstrobe => phy_dstrobe_pri,
               resetb       => resetb);


   monostable_rx_cmd_reset_pri : component monostable
      port map(
         clk => clk,
         A => rx_cmd_reset_pri,
         Z => rx_cmd_reset_pri_monos);
end architecture RTL;
```

```
--------------------------------------------------------------------------------
--! @file tx.vhd
--! @author Cairo Caplan <cairo.caplan@cern.ch>
--! @date April, 2016
--------------------------------------------------------------------------------

library ieee;
use ieee.std_logic_1164.all;

entity tx is
   generic(
      HEADER_FIELD:  integer := 1;
      ADDR_WIDTH  :  integer := 5;
      THRESHOLD   :  integer := 3072
   );
   port (
      tx_dat            :  in std_logic_vector(15 downto 0);
      clk               :  in std_logic;
      tx_ena            :  in std_logic;
      tx_adr            :  in std_logic_vector(7 downto 0);
      resetb            :  in std_logic;
      tx_cmd_reset      :  in std_logic;
      tx_cmd_test       :  in std_logic;
      rx_cmd_reset      :  in std_logic;
      tx_dav            :  out   std_logic;
      tx_sdr            :  out   std_logic_vector(1 downto 0);
      tx_ns          :  out   std_logic_vector(2 downto 0);
      tx_nr          :  in std_logic_vector(2 downto 0);
      tx_cmd_srej       :  in std_logic_vector(6 downto 0);
      tx_cmd_sabm       :  in std_logic;
      tx_cmd_ua         :  in std_logic;
      cmd_busy       :  out std_logic
   );
end entity tx;

architecture RTL of tx is



   component MAC_tx
      generic(ADDR_WIDTH   : integer := 12;
              THRESHOLD    : integer := 3072;
              HEADER_FIELD : integer := 1);
      port(tx_dat        : in  std_logic_vector(15 downto 0);
          tx_ena        : in  std_logic;
          tx_dav        : out std_logic;
          tx_adr        : in  std_logic_vector(7 downto 0);
          clk           : in  std_logic;
          tx_nr         : in  std_logic_vector(2 downto 0);
          tx_ns       : out std_logic_vector(2 downto 0);
          tx_cmd_srej   : in  std_logic_vector(6 downto 0);
          resetb        : in  std_logic;
          tx_cmd_reset  : in  std_logic;
          tx_cmd_test   : in  std_logic;
          --phy_bytesel : out std_logic;
          phy_dvalid  : out std_logic;
          phy_tx_data   : out std_logic_vector(7 downto 0);
          phy_dstrobe   : in  std_logic;
          crc           : in  std_logic_vector(15 downto 0);
```

```
        crc_strobe    : out std_logic;
        tx_cmd_sabm   : in  std_logic;
        tx_cmd_ua     : in  std_logic;
        cmd_busy      : out std_logic;
        rx_cmd_reset  : in  std_logic);
   end component MAC_tx;

   component crc
      generic(CRC_WIDTH  : natural                                := 16;
              DATA_WIDTH : natural                               := 16;
              INIT_VAL   : std_logic_vector(2 * 8 - 1 downto 0) := x"ffff";
              POLY       : std_logic_vector(2 * 8 - 1 downto 0) := x"8408");
      port(d       : in  std_logic_vector(DATA_WIDTH - 1 downto 0);
           init    : in  std_logic;
           d_valid : in  std_logic;
           clk     : in  std_logic;
           reset_b : in  std_logic;
           crc_o   : out std_logic_vector(CRC_WIDTH - 1 downto 0));
   end component crc;


   component PHY_HDLC_tx
      port(resetb     : in  std_logic;
           tx_clk     : in  std_logic;
           tx_sdr     : out std_logic_vector(1 downto 0);
           tx_data    : in  std_logic_vector(7 downto 0);
           tx_dvalid  : in  std_logic;
           tx_dstrobe : out std_logic);
   end component PHY_HDLC_tx;

   component regbank
      generic(WIDTH : integer := 8);
      port(d   : in  std_logic_vector(WIDTH - 1 downto 0);
           rn  : in  std_logic;
           q   : out std_logic_vector(WIDTH - 1 downto 0);
           clk : in  std_logic);
   end component regbank;

   --signal phy_bytesel       :  std_logic;
   signal  phy_tx_data     : std_logic_vector(7 downto 0);
   signal  phy_dvalid        : std_logic;
   signal  phy_dstrobe     : std_logic;
   signal   phy_resetb       : std_logic;

   signal crc_strobe         : std_logic;

   --signal clkb           : std_logic;
   signal crc_o            : std_logic_vector(15 downto 0);

   signal phy_tx_data_delayed : std_logic_vector(7 downto 0);


   signal phy_dvalidb : std_logic;

begin


   MAC_tx_inst : component MAC_tx
```

```
      generic map(ADDR_WIDTH   => ADDR_WIDTH,
                  THRESHOLD    => THRESHOLD,
                  HEADER_FIELD => HEADER_FIELD)
      port map(tx_dat        => tx_dat,
               tx_ena        => tx_ena,
               tx_dav        => tx_dav,
               tx_adr        => tx_adr,
               clk           => clk,
               tx_nr         => tx_nr,
               tx_ns       => tx_ns,
               tx_cmd_srej   => tx_cmd_srej,
               resetb        => resetb,
               tx_cmd_reset  => tx_cmd_reset,
               tx_cmd_test   => tx_cmd_test,
--             phy_bytesel => phy_bytesel,
               phy_dvalid  => phy_dvalid,
               phy_tx_data   => phy_tx_data,
               phy_dstrobe   => phy_dstrobe,
               crc           => crc_o,
               crc_strobe    => crc_strobe,
               tx_cmd_sabm   => tx_cmd_sabm,
               tx_cmd_ua     => tx_cmd_ua,
               cmd_busy      => cmd_busy,
               rx_cmd_reset  => rx_cmd_reset);



   regbank_inst : component regbank
      generic map(WIDTH => phy_tx_data'length)
      port map(d   => phy_tx_data,
               rn  => '1',
               q   => phy_tx_data_delayed,
               clk => clk);

   crc_inst : component crc
      generic map(
                  DATA_WIDTH => 8)
      port map(d       => phy_tx_data_delayed,
               init    => phy_dvalidb,
               d_valid => crc_strobe,
               clk     => clk,
               reset_b => resetb,
               crc_o   => crc_o);

   PHY_HDLC_tx_inst : component PHY_HDLC_tx
      port map(tx_data      => phy_tx_data,
               tx_dvalid    => phy_dvalid,
               tx_dstrobe => phy_dstrobe,
               tx_sdr     => tx_sdr,
               tx_clk       => clk,
               resetb       => phy_resetb);

   --
   --clkb <= not clk;

   phy_dvalidb <= not phy_dvalid;

   phy_resetb <= resetb and (not rx_cmd_reset);
```

```
end architecture RTL;
```

```
--------------------------------------------------------------------------------
--! @file channel_fifo.vhd
--! @author Cairo Caplan <cairo.caplan@cern.ch>
--! @date April, 2016
--------------------------------------------------------------------------------

library ieee;
use ieee.std_logic_1164.all;
use work.SCA_Package.all;

entity channel_fifo is
   port (
      --clocks
      rx_clk            :  in std_logic;
      tx_clk            :  in std_logic;
      --
      rst            : in std_logic;
      -- Interfaces with the Protocol Layer ------------------
      cmd_data_from_protocol_layer: in payload_t;
      rpy_data_to_protocol_layer :  out   payload_t;

      --! Ready to receive a new command from protocol layer
      cmd_fifo_ready_to_rcv_from_prot     :  out   std_logic;
      --! Receive a new command from protocol layer
      rcv_cmd_from_prot    :  in std_logic;

      --! Ready to send a new reply to protocol layer
      rdy_to_send_rpy_to_prot    :  out   std_logic;
      --! Request to send a new reply to protocol layer
      req_to_send_rpy_to_prot    :  in std_logic;

      --Interfaces with the Arbiter, then the SCA ---------------------
      cmd_data_to_arb      :  out   payload_t;
      rpy_data_from_arb :  in payload_t;

      --! Ready/Request to send a new command to the arbiter
      req_to_send_cmd_to_arb        :  out   std_logic;
      --! Command ack by the arbiter
      cmd_ack_by_arb       :  in std_logic;

      --! Ready to receive a new reply from the arbiter
      rdy_to_rcv_rpy_from_arb    :  out   std_logic;
      --! New Reply from Arbiter
      rpy_from_arb      :  in std_logic



      --

   );
end entity channel_fifo;

architecture RTL of channel_fifo is
   constant FIFO_SIZE         :  natural := (2**CMD_FIFO_ADDR_WIDTH);
   --constant CH_FIFO_COUNT      :  integer := (2**CMD_FIFO_ADDR_WIDTH);
   signal cmd_fifo            :  payload_vector_t(0 to FIFO_SIZE -1);
   signal rpy_fifo            :  payload_vector_t(0 to FIFO_SIZE -1);

   signal cmd_fifo_full    :  std_logic;
```

```
   signal cmd_fifo_empty      :  std_logic;
   --signal cmd_free_count    :  natural range 0 to CH_FIFO_COUNT-1;

   signal rpy_fifo_full    :  std_logic;
   signal rpy_fifo_empty      :  std_logic;
   --signal rpy_free_count    :  natural range 0 to CH_FIFO_COUNT-1;


   signal cmd_fifo_read_ptr   :  natural range 0 to FIFO_SIZE -1 := 0;
   signal cmd_fifo_write_ptr  :  natural range 0 to FIFO_SIZE -1 := 0;

   signal rpy_fifo_read_ptr   :  natural range 0 to FIFO_SIZE -1;
   signal rpy_fifo_write_ptr  :  natural range 0 to FIFO_SIZE -1;




   --! --! Ready to receive a new command from protocol layer
   signal cmd_fifo_rdy_prot_int  :  std_logic;

   signal rpy_data_from_arb_buff :  payload_t;
   signal rpy_from_arb_buff      :  std_logic;
   signal rpy_fifo_rdy_arb_int      :  std_logic;
   signal rpy_fifo_rdy_prot_int     :  std_logic;

   signal cmd_fifo_filling       :  natural range 0 to  FIFO_SIZE -1;
   signal rpy_fifo_filling       :  natural range 0 to  FIFO_SIZE -1;
   -- type fifo_state_t is (resetState, idleState, tryToWrite, writeToSCA,
--    waitForReturnTransaction, endOfTransaction
-- );
--
-- signal fifo_state, next_state    :   fifo_state_t;

begin

   fifo_management : process(tx_clk, rpy_from_arb) is
      variable cmd_fifo_filling_var :  natural range 0 to  FIFO_SIZE -1 :=0;
      variable rpy_fifo_filling_var :  natural range 0 to  FIFO_SIZE -1 :=0;
   begin
      if rising_edge(tx_clk) then
         if rst='1' then

            cmd_fifo_read_ptr <= 0;
            cmd_fifo_write_ptr<= 0;

            rpy_fifo_read_ptr <= 0;
            rpy_fifo_write_ptr<= 0;

            cmd_fifo_filling_var:= 0;
            rpy_fifo_filling_var := 0;
            req_to_send_cmd_to_arb <= '0';
         else

            ------COMMANDS DATA PATH:
            --"if there's a request to write a new command to the fifo
            if rcv_cmd_from_prot='1' and cmd_fifo_rdy_prot_int = '1' then
               cmd_fifo(cmd_fifo_write_ptr) <= cmd_data_from_protocol_layer;
               --move the write pointer to the next position
               if cmd_fifo_write_ptr = FIFO_SIZE - 1 then
```

```
            cmd_fifo_write_ptr <= 0;
         else
            cmd_fifo_write_ptr <= cmd_fifo_write_ptr + 1;
         end if;
         cmd_fifo_filling_var := cmd_fifo_filling_var + 1;
      end if;


      --if the arbiter has read the last command available
      if cmd_ack_by_arb = '1' then
         --cmd_fifo(cmd_fifo_write_ptr) <= cmd_data_from_protocol_layer;
         --move the read pointer to the next position
         if cmd_fifo_empty = '1' then
            cmd_fifo_read_ptr <= cmd_fifo_read_ptr;
         else
            if cmd_fifo_read_ptr = FIFO_SIZE - 1 then
               cmd_fifo_read_ptr <= 0;
            else
               cmd_fifo_read_ptr <= cmd_fifo_read_ptr + 1;
            end if;
            cmd_fifo_filling_var := cmd_fifo_filling_var - 1;
         end if;
      end if;

      cmd_data_to_arb <= cmd_fifo(cmd_fifo_read_ptr);


      -------REPLIES DATA PATH:
      --! If there is a new reply from the arbiter:
      if rpy_from_arb_buff='1' and rpy_fifo_rdy_arb_int = '1' then
         rpy_fifo(rpy_fifo_write_ptr) <= rpy_data_from_arb;
         --move the write pointer to the next position
         if rpy_fifo_write_ptr = FIFO_SIZE - 1 then
            rpy_fifo_write_ptr <= 0;
         else
            rpy_fifo_write_ptr <= rpy_fifo_write_ptr + 1;
         end if;
         rpy_fifo_filling_var := rpy_fifo_filling_var + 1;
      end if;

      --! If the Protocol Layer is ready to receive a reply and fifo is not empty
      if rpy_fifo_rdy_prot_int='1' and req_to_send_rpy_to_prot='1' then
         --rpy_fifo(rpy_fifo_write_ptr) <= rpy_data_from_arb;
         --move the read pointer to the next position, as the Protocol Layer
         --took the Reply
         if rpy_fifo_read_ptr = FIFO_SIZE - 1 then
            rpy_fifo_read_ptr <= 0;
         else
            rpy_fifo_read_ptr <= rpy_fifo_read_ptr + 1;
         end if;
         rpy_fifo_filling_var:= rpy_fifo_filling_var - 1;
      end if;
      rpy_data_to_protocol_layer <= rpy_fifo(rpy_fifo_read_ptr);

      req_to_send_cmd_to_arb <= not cmd_fifo_empty;
   end if;
end if;

cmd_fifo_filling <= cmd_fifo_filling_var;
rpy_fifo_filling <= rpy_fifo_filling_var;
```

```
      rpy_from_arb_buff <= rpy_from_arb;

   end process fifo_management;



   fifo_current_state : process (cmd_fifo_read_ptr, cmd_fifo_write_ptr, rpy_fifo_read_ptr,
    rpy_fifo_write_ptr) is
   begin
      if cmd_fifo_read_ptr = cmd_fifo_write_ptr then
         cmd_fifo_empty <= '1';
      else
         cmd_fifo_empty <= '0';
      end if;

      if (cmd_fifo_write_ptr = cmd_fifo_read_ptr + 1)
      or (cmd_fifo_read_ptr=0 and cmd_fifo_write_ptr = FIFO_SIZE -1 ) then
         cmd_fifo_full <= '1';
      else
         cmd_fifo_full <= '0';
      end if;

      if rpy_fifo_read_ptr = rpy_fifo_write_ptr then
         rpy_fifo_empty <= '1';
      else
         rpy_fifo_empty <= '0';
      end if;

      if (rpy_fifo_write_ptr = rpy_fifo_read_ptr + 1)
      or (rpy_fifo_read_ptr=0 and rpy_fifo_write_ptr = FIFO_SIZE -1 ) then
         rpy_fifo_full <= '1';
      else
         rpy_fifo_full <= '0';
      end if;

   end process fifo_current_state;


   cmd_fifo_rdy_prot_int <= '1' when cmd_fifo_filling /= FIFO_SIZE -1 else '0';
   cmd_fifo_ready_to_rcv_from_prot <= '1' when cmd_fifo_filling /= FIFO_SIZE -1 else '0';

   rpy_fifo_rdy_arb_int <= '1' when rpy_fifo_filling /= FIFO_SIZE -1 else '0';
   rdy_to_rcv_rpy_from_arb <= '1' when rpy_fifo_filling /= FIFO_SIZE -1 else '0';

   rpy_fifo_rdy_prot_int <= not rpy_fifo_empty;
   rdy_to_send_rpy_to_prot <= not rpy_fifo_empty;



   recv_reply_sync : process (rx_clk) is
   begin
      if rising_edge(rx_clk) then
            rpy_data_from_arb_buff <= rpy_data_from_arb;
      end if;
   end process recv_reply_sync;



end architecture RTL;
```

```
--------------------------------------------------------------------------------
--! @file eport_driver.vhd
--! @author Cairo Caplan <cairo.caplan@cern.ch>
--! @date April, 2016
--------------------------------------------------------------------------------

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

use work.SCA_Package.all;

entity eport_driver is
   port (
      --!
      tx_clk            :  in std_logic;
      rst               :  in std_logic;
      eport_en       :  in std_logic;

      --! \name Signals to the Elink
      --! \{

      rx_ena            : in std_logic;
      rx_dav            : out  std_logic;
      rx_sop            : in std_logic;
      rx_eop            : in std_logic;
      rx_ns             : in std_logic_vector(2 downto 0);
      rx_adr            : in std_logic_vector(7 downto 0);
      rx_dat            : in std_logic_vector(15 downto 0);
      rx_nr             : in std_logic_vector(2 downto 0);
      rx_cmd_srej       : in std_logic_vector(6 downto 0);
      rx_err            : in std_logic;
      tx_ena            : out  std_logic;
      tx_dav            : in std_logic;
      tx_ns             : in std_logic_vector(2 downto 0);
      tx_adr            : out  std_logic_vector(7 downto 0);
      tx_dat            : out  std_logic_vector(15 downto 0);
      tx_cset           : out  std_logic_vector(3 downto 0);
      --! \}

      --! \name Interface with Command Queue or Protocol Layer
      --! \{
       sca_cmd_data_i      : in payload_t;
       sca_rpy_data_o     : out  payload_t;
       sca_cmd_ena_i      : in std_logic;
       sca_cmd_av_o       : out  std_logic;
       sca_rpy_ena_i       : in std_logic;
       sca_rpy_av_o     : out  std_logic

      --! \}

       );

end entity eport_driver;

architecture RTL of eport_driver is


   --! Counter that decides which of thew Command FIFOs will be used to send to the Eport
```

```
   --signal cmd_arbiter    :  natural range 0 to SCA_CH_COUNT -1;

   --! Last TRansaction Number sent by each channel
   --signal TR_sent  :  byte_t;


   --signal cmd_rdy_to_tx_driver    :  std_logic := '0';
   signal cmd_data_to_tx_driver  :  payload_t;
   signal tx_driver_counter      :  natural range 0 to 3;

   --signal rpy_rdy_from_rx_receiver   :  std_logic;
   --signal rpy_data_from_rx_receiver: payload_t;
   signal rx_receiver_counter    :  natural range 0 to 3;



   type state_t is (IDLE, BUSY, BLOCKED);
   signal tx_state,rx_state   :  state_t;



-- type state_t is (resetState, scanState, startState, stopState,
--    resetSCAState, connectSCAState, waitresetORconnectState, waitForControllerState
-- );
----
-- signal state   :  state_t;
--
   signal tx_driver_running   :  std_logic := '0';



begin

   -- At least for now:
   tx_cset <= (others=>'0');
   tx_adr <= (others=>'1');



   sca_cmd_av_o <= '1' when (tx_state=IDLE and sca_cmd_ena_i='0') else '0';

   --tx_ena <= '1' when tx_driver_running='1' else '0';-- Test SOL40_SCA 06/05/2015

   --! Process that manages the sending of commands to the elink_tx_driver,
   --! then the SCA(e-Port)
   elink_tx_driver : process (tx_clk) is
      -- constant data_high   :  natural := cmd_data_to_tx_driver.data'high; -- Test SOL40_SCA 06/05/2015
      variable cmd_len  :  natural range 0 to 255;
   begin
      if rising_edge(tx_clk) then
         tx_ena <= tx_driver_running;-- Test SOL40_SCA 06/05/2015
         if rst = '1' then
            --cmd_arbiter <= sca_ch_enum'pos(sca_controller);
            tx_state <= BLOCKED;
            tx_dat <= (others=>'0');
         else
            case tx_state is
            when IDLE =>
               if eport_en='0' then
                  tx_state <= BLOCKED;
```

```
            elsif sca_cmd_ena_i='1' then
                  tx_state <= BUSY;
                  cmd_data_to_tx_driver <= sca_cmd_data_i;
                  tx_driver_counter <= 0;
                  tx_driver_running <= '1';
            end if;
            tx_dat <= (others=>'0');
         when BUSY =>
            if eport_en='0' then
               tx_state <= BLOCKED;
               tx_dat <= (others=>'0');
            elsif tx_dav='1' or tx_driver_running='1' then
               case tx_driver_counter is
               -- ( CH , TR )
               when 0=> tx_dat <= cmd_data_to_tx_driver.ch & cmd_data_to_tx_driver.tr;
               -- ( CMD , LEN )
               when 1=> tx_dat <= cmd_data_to_tx_driver.cmd_or_err & cmd_data_to_tx_driver.len;
               -- ( D0 , D1 )
               when 2=> tx_dat <= cmd_data_to_tx_driver.data(7 downto 0) &
   cmd_data_to_tx_driver.data(15 downto 8);
               -- ( D2 , D3 )
               when 3=> tx_dat <= cmd_data_to_tx_driver.data(23 downto 16) &
   cmd_data_to_tx_driver.data(31 downto 24);
               when others=> tx_dat <= (others=>'0');

               end case;

               cmd_len := to_integer(unsigned(cmd_data_to_tx_driver.len));
               if ((tx_driver_counter=1 and cmd_len=0) or (tx_driver_counter =2 and  cmd_len<=2) or
   (tx_driver_counter =3)) then --  and  cmd_len<32
                  --tx_driver_running <= '0';
                  tx_driver_counter <= 0;
                  tx_state <= IDLE;
                  tx_driver_running <= '0';
               else
                  tx_driver_running <= '1';
                  tx_driver_counter <= tx_driver_counter + 1;
               end if;
            end if;
         when BLOCKED =>
            if eport_en='1' then
               tx_state <= IDLE;
            end if;
            tx_dat <= (others=>'0');
         end case;
      end if;
    end if;
  end process elink_tx_driver;


-- rx_state <=  BLOCKED when enable='0' else
--           BUSY when (rx_ena='1') else
--           IDLE;


  --! Process that receives the sca replies through the atlantic bus from the Eport
  elink_rx_receiver : process(tx_clk) is
    --constant data_high :  natural := rpy_data_from_rx_receiver.data'high;
```

```
    --variable rpy_from_arb_var      :  std_logic;
begin
    if rising_edge(tx_clk) then
        if rst='1' then
            rx_state <= BLOCKED;
        else
            case rx_state is
            when BLOCKED =>
                sca_rpy_av_o <= '0';
                if eport_en='1' then
                    rx_state <= IDLE;
                    rx_dav <= '1';
                else
                    rx_dav <= '0';
                end if;
            when BUSY =>
                if sca_rpy_ena_i = '1' then
                    sca_rpy_av_o <= '0';
                    if eport_en='1' then
                        rx_state <= IDLE;
                        rx_dav <= '1';
                    else
                        rx_state <= BLOCKED;
                        rx_dav <= '0';
                    end if;
                else
                    rx_dav <= '0';
                end if;
            when IDLE =>
                if rx_ena='1' then
                    case rx_receiver_counter is
                    when 0=>
                        -- ( CH , TR )
                        sca_rpy_data_o.tr <= rx_dat(7 downto 0);
                        sca_rpy_data_o.ch <= rx_dat(15 downto 8);
                    when 1=>
                        -- ( LEN , ERR )
                        sca_rpy_data_o.cmd_or_err <= rx_dat(7 downto 0);
                        sca_rpy_data_o.len <= rx_dat(15 downto 8);
                        --31 downto 16 = data0
                    when 2=>
                        -- ( D0 , D1 )
                        sca_rpy_data_o.data(7 downto 0) <= rx_dat(15 downto 8);
                        sca_rpy_data_o.data(15 downto 8) <= rx_dat(7 downto 0);
                    when 3 =>
                        -- ( D2 , D3 )
                        sca_rpy_data_o.data(23 downto 16) <= rx_dat(15 downto 8);
                        sca_rpy_data_o.data(31 downto 24) <= rx_dat(7 downto 0);

                    end case;

                    if rx_receiver_counter/=3 and rx_eop='0' then
                        rx_receiver_counter <= rx_receiver_counter + 1;
                    elsif rx_eop = '1' then
                        rx_dav <= '0';
                        rx_receiver_counter <= 0;
                        rx_state <= BUSY;
                        sca_rpy_av_o <= '1';
                        --sca_rpy_data_o <= rpy_data_from_rx_receiver;
                    else -- rx_eop/='1' then
```

```
                    report "SCA MAC Layer - eport_driver : Receiving Packet is too long";
                end if;

            end if;
        end case;
      end if;
    end if;

  end process elink_rx_receiver;


end architecture RTL;
```

```
--------------------------------------------------------------------------------
--! @file mac_layer.vhd
--! @author Cairo Caplan <cairo.caplan@cern.ch>
--! @date April, 2016
--------------------------------------------------------------------------------

library ieee;
use ieee.std_logic_1164.all;

use work.SCA_Package.all;

--! @brief This is Media access control layer for the GBT-SCA links.
--! It can control manage data and encapsulate data to many GBT-SCA's E-links.
entity mac_layer is
   port (
      tx_clk            :  in  std_logic;
      rst               :  in  std_logic;
      --! Array of SCA command packets for each SCA
      sca_cmd_data_i    :  in  payload_vector_t(0 to SCA_COUNT - 1);
      sca_cmd_ena_i     :  in  std_logic_vector(0 to SCA_COUNT - 1);
      sca_cmd_rdy_o     :  out std_logic_vector(0 to SCA_COUNT - 1);
      --! Array of SCA reply packets for each SCA
      sca_rpy_data_o    :  out payload_vector_t(0 to SCA_COUNT - 1);
      sca_rpy_prot_rdy_i :  in  std_logic_vector(0 to SCA_COUNT - 1);
      sca_rpy_en_o      :  out std_logic_vector(0 to SCA_COUNT - 1);
      --!
      reconnnect_i      :  in std_logic_vector(0 to SCA_COUNT - 1);
      reset_sca_i       :  in std_logic_vector(63 downto 0);
      disable_sca_i     :  in std_logic_vector(63 downto 0);
      sca_ch_state_o    :  out sca_ch_state_vector_t(0 to SCA_COUNT-1);
      sca_link_state_array_o  : out SCA_link_state_array_t;
      --! Array of Tx signals for each E-Link
      tx_sd             :  out elink_ch_array_t(0 to SCA_COUNT-1);
      rx_clk            :  in std_logic;
      --! Array of Rx signals for each E-Link
      rx_sd             :  in elink_ch_array_t(0 to SCA_COUNT-1)
   );
end entity mac_layer;


--! @details The mac_layer is the last layer of the SOL40-SCA before the
--! logical E-links that goes to the GBT frame. It interfaces with the
--! protocol_layer by exchanging SCA compliant packets and implements data
--! transmission to the many GBT-SCAs, which can be attached to many GBT-Links
--! and be controlled by one instance of the SOL40-SCA. The management
--! of a single GBT-SCA is made by sca_mac_layer instances.
architecture RTL of mac_layer is

   component sca_mac_layer
      port(
         tx_clk            : in  std_logic;
         rst               : in  std_logic;
         sca_cmd_data_i    : in  payload_t;
         sca_cmd_ena_i     : in  std_logic;
         sca_cmd_rdy_o     : out std_logic;
         sca_rpy_data_o    : out payload_t;
         sca_rpy_prot_rdy_i : in  std_logic;
         send_sca_rpy_o    : out std_logic;
         reconnnect_i      : in  std_logic;
```

```
        disable_sca_i      : in  std_logic;
        reset_sca_i        : in  std_logic;
        sca_ch_state_o     : out ch_state_vector_t;
        sca_link_state_o   : out SCA_link_state_t;
        tx_sd              : out std_logic_vector(1 downto 0);
        rx_clk             : in  std_logic;
        rx_sd              : in  std_logic_vector(1 downto 0)
     );
  end component sca_mac_layer;


begin
  --! Generate for each GBT-SCA on the GBT link a sca_mac_layer instance to
  --! deal with the their data transmission and current state


     gen_by_sca : for j in 0 to SCA_COUNT-1 generate

        sca_mac_layer_inst : component sca_mac_layer
           port map(
              tx_clk             => tx_clk,
              rst                => rst,
              sca_cmd_data_i     => sca_cmd_data_i(j),
              sca_cmd_ena_i      => sca_cmd_ena_i(j),
              sca_cmd_rdy_o      => sca_cmd_rdy_o(j),
              sca_rpy_data_o     => sca_rpy_data_o(j),
              sca_rpy_prot_rdy_i => sca_rpy_prot_rdy_i(j),
              send_sca_rpy_o     => sca_rpy_en_o(j),
              reconnnect_i       => reconnnect_i(j),
              disable_sca_i      => disable_sca_i(j),
              reset_sca_i        => reset_sca_i(j),
              sca_ch_state_o     => sca_ch_state_o(j),
              sca_link_state_o   => sca_link_state_array_o(j),
              tx_sd              => tx_sd(j),
              rx_clk             => rx_clk,
              rx_sd              => rx_sd(j)
           );

     end generate gen_by_sca;

end architecture RTL;
```

```
--------------------------------------------------------------------------------
--! @file sca_mac_layer.vhd
--! @author Cairo Caplan <cairo.caplan@cern.ch>
--! @date April, 2016
--------------------------------------------------------------------------------

library ieee;
use ieee.std_logic_1164.all;
use work.SCA_Package.all;

entity sca_mac_layer is
   port (
      tx_clk : in std_logic;
      rst : in std_logic;
      -- Interface with the Protocol Layer
      sca_cmd_data_i          :  in  payload_t;
      sca_cmd_ena_i           :  in  std_logic;
      sca_cmd_rdy_o           :  out std_logic;
      --
      sca_rpy_data_o          :  out payload_t;
      sca_rpy_prot_rdy_i      :  in  std_logic;
      send_sca_rpy_o          :  out std_logic;
      reconnnect_i         :  in std_logic;
      -------
      disable_sca_i        :  in std_logic;
      reset_sca_i          :  in std_logic;
      -------
      sca_ch_state_o       :  out   ch_state_vector_t;
      sca_link_state_o     :  out SCA_link_state_t;
      ---------------------------------------
      tx_sd :  out std_logic_vector(1 downto 0);
      rx_clk   :  in std_logic;
      rx_sd    :  in std_logic_vector(1 downto 0)
   );
end entity sca_mac_layer;

architecture RTL of sca_mac_layer is



   signal sca_ch_state  :  ch_state_vector_t;

   component eport_driver
      port(tx_clk             : in  std_logic;
          rst                : in  std_logic;
          eport_en           : in  std_logic;
          rx_ena             : in  std_logic;
          rx_dav             : out std_logic;
          rx_sop             : in  std_logic;
          rx_eop             : in  std_logic;
          rx_ns              : in  std_logic_vector(2 downto 0);
          rx_adr             : in  std_logic_vector(7 downto 0);
          rx_dat             : in  std_logic_vector(15 downto 0);
          rx_nr              : in  std_logic_vector(2 downto 0);
          rx_cmd_srej        : in  std_logic_vector(6 downto 0);
          rx_err             : in  std_logic;
          tx_ena             : out std_logic;
          tx_dav             : in  std_logic;
          tx_ns              : in  std_logic_vector(2 downto 0);
```

```
        tx_adr             : out std_logic_vector(7 downto 0);
        tx_dat             : out std_logic_vector(15 downto 0);
        tx_cset            : out std_logic_vector(3 downto 0);
        sca_cmd_data_i     : in  payload_t;
        sca_rpy_data_o     : out payload_t;
        sca_cmd_ena_i      : in  std_logic;
        sca_cmd_av_o      : out std_logic;
        sca_rpy_ena_i      : in  std_logic;
        sca_rpy_av_o     : out std_logic);
end component eport_driver;

component fpga_elink
   generic(MASTER            : integer := 1;
           HEADER_FIELD      : integer := 1;
           ADDR_WIDTH        : integer := 5;
           MAX_PACKET_LENGTH : integer := 16);
   port(reset        : in  std_logic;
        user_clk     : out std_logic;
        cmd_busy     : out std_logic;
        rx_ena       : out std_logic;
        rx_dav       : in  std_logic;
        rx_sop       : out std_logic;
        rx_eop       : out std_logic;
        rx_cmd_test  : out std_logic;
        rx_cmd_reset : out std_logic;
        rx_cmd_ua    : out std_logic;
        rx_ns        : out std_logic_vector(2 downto 0);
        rx_adr       : out std_logic_vector(7 downto 0);
        rx_dat       : out std_logic_vector(15 downto 0);
        rx_nr        : out std_logic_vector(2 downto 0);
        rx_cmd_srej  : out std_logic_vector(6 downto 0);
        rx_err       : out std_logic;
        tx_ena       : in  std_logic;
        tx_dav       : out std_logic;
        tx_cmd_test  : in  std_logic;
        tx_cmd_reset : in  std_logic;
        tx_cmd_sabm  : in  std_logic;
        tx_ns        : out std_logic_vector(2 downto 0);
        tx_adr       : in  std_logic_vector(7 downto 0);
        tx_dat       : in  std_logic_vector(15 downto 0);
        tx_cset      : in  std_logic_vector(3 downto 0);
        tx_clk       : in  std_logic;
        tx_sd        : out std_logic_vector(1 downto 0);
        rx_clk       : in  std_logic;
        rx_sd        : in  std_logic_vector(1 downto 0));
end component fpga_elink;



signal cmd_busy : std_logic;
signal rx_ena : std_logic;
signal rx_dav : std_logic;
signal rx_sop : std_logic;
signal rx_eop : std_logic;
signal rx_cmd_test : std_logic;
signal rx_cmd_ua : std_logic;
signal rx_ns : std_logic_vector(2 downto 0);
signal rx_adr : std_logic_vector(7 downto 0);
signal rx_dat : std_logic_vector(15 downto 0);
signal rx_nr : std_logic_vector(2 downto 0);
```

```
   signal rx_cmd_srej : std_logic_vector(6 downto 0);
   signal rx_err : std_logic;
   signal tx_ena : std_logic;
   signal tx_dav : std_logic;
   signal tx_cmd_test : std_logic;
   signal tx_cmd_reset : std_logic;
   signal tx_cmd_sabm : std_logic;
   signal tx_ns : std_logic_vector(2 downto 0);
   signal tx_adr : std_logic_vector(7 downto 0);
   signal tx_dat : std_logic_vector(15 downto 0);
   signal tx_cset : std_logic_vector(3 downto 0);

   signal SCA_link_state   :  SCA_link_state_t;

   constant time_out    :  natural := (4_000 ns/25 ns );
   signal timer         :  natural range 0 to time_out;

   signal sca_cmd_av :  std_logic;
   signal enable : std_logic;


begin


   fpga_elink_inst : fpga_elink
      generic map(
         MASTER            => 1,
         HEADER_FIELD      => 1,
         ADDR_WIDTH        => 5,
         MAX_PACKET_LENGTH => 16
      )
      port map(
         reset        => rst,
         user_clk     => open,
         cmd_busy     => cmd_busy,
         rx_ena       => rx_ena,
         rx_dav       => rx_dav,
         rx_sop       => rx_sop,
         rx_eop       => rx_eop,
         rx_cmd_test  => rx_cmd_test,
         rx_cmd_reset => open,
         rx_cmd_ua    => rx_cmd_ua,
         rx_ns        => rx_ns,
         rx_adr       => rx_adr,
         rx_dat       => rx_dat,
         rx_nr        => rx_nr,
         rx_cmd_srej  => rx_cmd_srej,
         rx_err       => rx_err,
         tx_ena       => tx_ena,
         tx_dav       => tx_dav,
         tx_cmd_test  => tx_cmd_test,
         tx_cmd_reset => tx_cmd_reset,
         tx_cmd_sabm  => tx_cmd_sabm,
         tx_ns        => tx_ns,
         tx_adr       => tx_adr,
         tx_dat       => tx_dat,
         tx_cset      => tx_cset,
         tx_clk       => tx_clk,
         tx_sd        => tx_sd,
         rx_clk       => rx_clk,
```

```
      rx_sd        => rx_sd
   );



eport_driver_inst : eport_driver
   port map(tx_clk             => tx_clk,
            rst                => rst,
            eport_en           => enable,
            rx_ena             => rx_ena,
            rx_dav             => rx_dav,
            rx_sop             => rx_sop,
            rx_eop             => rx_eop,
            rx_ns              => rx_ns,
            rx_adr             => rx_adr,
            rx_dat             => rx_dat,
            rx_nr              => rx_nr,
            rx_cmd_srej        => rx_cmd_srej,
            rx_err             => rx_err,
            tx_ena             => tx_ena,
            tx_dav             => tx_dav,
            tx_ns              => tx_ns,
            tx_adr             => tx_adr,
            tx_dat             => tx_dat,
            tx_cset            => tx_cset,
            sca_cmd_data_i     => sca_cmd_data_i,
            sca_rpy_data_o     => sca_rpy_data_o,
            sca_cmd_ena_i      => sca_cmd_ena_i,
            sca_cmd_av_o      => sca_cmd_av,
            sca_rpy_ena_i      => sca_rpy_prot_rdy_i,
            sca_rpy_av_o     => send_sca_rpy_o);

fsm_sca_link_control : process (tx_clk) is
begin
   if rising_edge(tx_clk) then
      case SCA_link_state is
         when RESET =>
            --if rst = '1' or reset_sca_i='1' then
            if reset_sca_i='1' then
               SCA_link_state <= RESET;
            elsif disable_sca_i = '1' then
               SCA_link_state <= DISABLED;
            else
               SCA_link_state <= TESTING;
            end if;
            --
            for i in 0 to CH_BY_SCA_COUNT-1 loop
               sca_ch_state(i).busy <= '0';
               sca_ch_state(i).last_tr_sent <= (others=>'0');
            end loop;
            tx_cmd_reset <= '1';
            tx_cmd_sabm <= '0';
            tx_cmd_test <= '0';
            timer <= 0;
         when TESTING =>
            --if rst = '1' or reset_sca_i='1' then
            if reset_sca_i='1' then
               SCA_link_state <= RESET;
            elsif disable_sca_i = '1' then
               SCA_link_state <= DISABLED;
```

```
      elsif rx_cmd_test = '1' then
         SCA_link_state <= CONNECTING;
      else
         SCA_link_state <= TESTING;
         -------------------------
         if timer=0 and cmd_busy = '0' then
            tx_cmd_test <= '1';
            timer <= timer +1;
         elsif timer = time_out then
            timer <= 0;
            tx_cmd_test <= '0';
         elsif timer < time_out then
            timer <= timer +1;
            tx_cmd_test <= '0';
         end if;
      end if;
      tx_cmd_reset <= '0';
      tx_cmd_sabm <= '0';
   when CONNECTING =>
      --if rst = '1' or reset_sca_i='1' then
      if reset_sca_i='1' then
         SCA_link_state <= RESET;
      elsif disable_sca_i = '1' then
         SCA_link_state <= DISABLED;
      elsif rx_cmd_ua = '1' then
         SCA_link_state <= ACTIVE;
      else
         SCA_link_state <= CONNECTING;
         ----------------------------
         if timer=0 and cmd_busy = '0' then
            tx_cmd_sabm <= '1';
            timer <= timer +1;
         elsif timer = time_out then
            timer <= 0;
            tx_cmd_sabm <= '0';
         elsif timer < time_out then
            timer <= timer +1;
            tx_cmd_sabm <= '0';
         end if;
      end if;
      tx_cmd_reset <= '0';
      tx_cmd_test <= '0';
   when ACTIVE =>
      --if rst = '1' or reset_sca_i='1' then
      if reset_sca_i='1' then
         SCA_link_state <= RESET;
      elsif disable_sca_i = '1' then
         SCA_link_state <= DISABLED;
      elsif reconnnect_i='1' then
         SCA_link_state <= TESTING;
      end if;
      tx_cmd_reset <= '0';
      tx_cmd_test <= '0';
      tx_cmd_sabm <= '0';
      timer <= 0;
   when DISABLED =>
      --if rst = '1' or reset_sca_i='1' then
      if reset_sca_i='1' then
         SCA_link_state <= RESET;
      elsif disable_sca_i = '1' then
```

```
                SCA_link_state <= DISABLED;
            else
                SCA_link_state <= TESTING;
            end if;
            for i in 0 to CH_BY_SCA_COUNT-1 loop
                sca_ch_state(i).busy <= '1';
                sca_ch_state(i).last_tr_sent <= (others=>'0');
            end loop;
            tx_cmd_reset <= '0';
            tx_cmd_sabm <= '0';
            tx_cmd_test <= '0';
            timer <= 0;
        end case;
    end if;
end process fsm_sca_link_control;


enable <= '1' when (SCA_link_state=ACTIVE) else '0';

sca_cmd_rdy_o <= sca_cmd_av when (SCA_link_state=ACTIVE) else '0';

sca_ch_state_o <= sca_ch_state;

sca_link_state_o <= SCA_link_state;

end architecture RTL;
```

```
--------------------------------------------------------------------------------
--! @file adc_protocol_driver.vhd
--! @author Cairo Caplan <cairo.caplan@cern.ch>;
--! @date April, 2016
--------------------------------------------------------------------------------


library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

use work.SCA_Package.all;

entity adc_protocol_driver is
   port (
      -- Interface with the ECS CMD Protocol Arbiter
      prot_cmd_en_i     :  in std_logic;
      ecs_cmd_packet    :  in ecs_packet_t;
      -- Interface with the ECS RPY Protocol Arbiter
      prot_rpy_en_i     :  in std_logic;
      ecs_rpy_packet    :  out   ecs_packet_t;
      -- Interface with the Command Queue manager
      sca_cmd_batch_o      :  out prot_cmd_t;
      sca_rpy_batch_i      :  in prot_cmd_t;
      --Others
      tr_cmd_i       :  in byte_t
   );
end entity adc_protocol_driver;

architecture RTL of adc_protocol_driver is

begin


   adc_protocol_builder : process (
       prot_cmd_en_i, tr_cmd_i, ecs_cmd_packet
   ) is
      variable ecs_adc_cmd       :  ecs_adc_cmd_enum;
      variable new_cmd_batch     :  prot_cmd_t;
   begin


      new_cmd_batch.tr := x"00";
      new_cmd_batch.ch := x"FF";
      new_cmd_batch.cmd_count := 0;
      new_cmd_batch.sca := 0;

      new_cmd_batch.ecs_cmd := (others=>'0');
      new_cmd_batch.ecs_len := (others=>'0');
      new_cmd_batch.err := (others=>'0');

      new_cmd_batch.sca_cmds := (others=>(others=>'0'));
      new_cmd_batch.sca_cmds_data :=  (others=>(others=>(others=>'0')));
      new_cmd_batch.sca_rpys_data :=  (others=>(others=>(others=>'0')));

      ecs_adc_cmd := INVALID;


      if prot_cmd_en_i='1' then
```

```
      for i in ecs_adc_cmd_table_t'left to ecs_adc_cmd_table_t'right loop
         if ecs_cmd_packet.ecs_cmd = ecs_adc_cmd_table(i) then
            ecs_adc_cmd := i;
            exit;
         end if;
      end loop;

      new_cmd_batch.tr := tr_cmd_i;
      new_cmd_batch.ch := ecs_cmd_packet.sca_ch;
      new_cmd_batch.cmd_count := 0;
      --new_cmd_batch.gbt := to_integer(unsigned(ecs_cmd_packet.gbt_nr));
      new_cmd_batch.sca := to_integer(unsigned(ecs_cmd_packet.sca_nr));
      new_cmd_batch.ecs_cmd := ecs_cmd_packet.ecs_cmd;
      new_cmd_batch.ecs_len := ecs_cmd_packet.len;

      case ecs_adc_cmd is
         when ECS_ADC_GO =>
            new_cmd_batch.sca_cmds(0) := adc_cmd_table(ADC_GO);
            new_cmd_batch.sca_cmds_data(0).data := (others=>'0');
            new_cmd_batch.sca_cmds_data(0).len := (others=>'0');
            new_cmd_batch.cmd_count := 1;
         when ECS_ADC_W_INSEL =>
            new_cmd_batch.sca_cmds(0) := adc_cmd_table(ADC_W_INSEL);
            new_cmd_batch.sca_cmds_data(0).data(31 downto 0) := ecs_cmd_packet.data(31 downto 0);-- 12
downto 8
            new_cmd_batch.sca_cmds_data(0).len := std_logic_vector(to_unsigned( 4 ,8));
            new_cmd_batch.cmd_count := 1;
         when ECS_ADC_R_INSEL =>
            new_cmd_batch.sca_cmds(0) := adc_cmd_table(ADC_R_INSEL);
            new_cmd_batch.sca_cmds_data(0).data := (others=>'0');
            new_cmd_batch.sca_cmds_data(0).len := (others=>'0');
            new_cmd_batch.cmd_count := 1;
         when ECS_ADC_W_CUREN =>
            new_cmd_batch.sca_cmds(0) := adc_cmd_table(ADC_W_CUREN);
            new_cmd_batch.sca_cmds_data(0).data(31 downto 0) :=  ecs_cmd_packet.data(31 downto 0);
            new_cmd_batch.sca_cmds_data(0).len := std_logic_vector(to_unsigned( 4 ,8));
            new_cmd_batch.cmd_count := 1;
         when ECS_ADC_R_CUREN =>
            new_cmd_batch.sca_cmds(0) := adc_cmd_table(ADC_R_CUREN);
            new_cmd_batch.sca_cmds_data(0).data := (others=>'0');
            new_cmd_batch.sca_cmds_data(0).len := (others=>'0');
            new_cmd_batch.cmd_count := 1;
         when ECS_ADC_rea_DATA_Ofs_Ga =>
            new_cmd_batch.sca_cmds(0) := adc_cmd_table(ADC_rea_DATA_Ofs_Ga);
            new_cmd_batch.sca_cmds_data(0).data := (others=>'0');
            new_cmd_batch.sca_cmds_data(0).len := (others=>'0');
            new_cmd_batch.cmd_count := 1;
         when ECS_ADC_rea_DATA_Ofs =>
            new_cmd_batch.sca_cmds(0) := adc_cmd_table(ADC_rea_DATA_Ofs);
            new_cmd_batch.sca_cmds_data(0).data := (others=>'0');
            new_cmd_batch.sca_cmds_data(0).len := (others=>'0');
            new_cmd_batch.cmd_count := 1;
         when ECS_ADC_rea_DATA =>
            new_cmd_batch.sca_cmds(0) := adc_cmd_table(ADC_rea_DATA);
            new_cmd_batch.sca_cmds_data(0).data := (others=>'0');
            new_cmd_batch.sca_cmds_data(0).len := (others=>'0');
            new_cmd_batch.cmd_count := 1;
         when ECS_ADC_rea_OFFSET =>
            new_cmd_batch.sca_cmds(0) := adc_cmd_table(ADC_rea_OFFSET);
```

```
            new_cmd_batch.sca_cmds_data(0).data := (others=>'0');
            new_cmd_batch.sca_cmds_data(0).len := (others=>'0');
            new_cmd_batch.cmd_count := 1;
         when ECS_ADC_wrt_CTRL =>
            new_cmd_batch.sca_cmds(0) := adc_cmd_table(ADC_wrt_CTRL);
            new_cmd_batch.sca_cmds_data(0).data(4 downto 0) := ecs_cmd_packet.data(4 downto 0);
            new_cmd_batch.sca_cmds_data(0).len := std_logic_vector(to_unsigned( 1 ,8));
            new_cmd_batch.cmd_count := 1;
         when ECS_ADC_rea_CTRL =>
            new_cmd_batch.sca_cmds(0) := adc_cmd_table(ADC_rea_CTRL);
            new_cmd_batch.sca_cmds_data(0).data := (others=>'0');
            new_cmd_batch.sca_cmds_data(0).len := (others=>'0');
            new_cmd_batch.cmd_count := 1;
         when ECS_ADC_GO_SingleSlope =>
            new_cmd_batch.sca_cmds(0) := adc_cmd_table(ADC_GO_SingleSlope);
            new_cmd_batch.sca_cmds_data(0).data := (others=>'0');
            new_cmd_batch.sca_cmds_data(0).len := (others=>'0');
            new_cmd_batch.cmd_count := 1;
         when ECS_ADC_wrt_GainCalibrReg =>
            new_cmd_batch.sca_cmds(0) := adc_cmd_table(ADC_wrt_GainCalibrReg);
            new_cmd_batch.sca_cmds_data(0).data(12 downto 0) := ecs_cmd_packet.data(12 downto 0);
            new_cmd_batch.sca_cmds_data(0).len := std_logic_vector(to_unsigned( 2 ,8));
            new_cmd_batch.cmd_count := 1;
         when ECS_ADC_rea_GainCalibrReg =>
            new_cmd_batch.sca_cmds(0) := adc_cmd_table(ADC_rea_GainCalibrReg);
            new_cmd_batch.sca_cmds_data(0).data := (others=>'0');
            new_cmd_batch.sca_cmds_data(0).len := (others=>'0');
            new_cmd_batch.cmd_count := 1;
         when ECS_ADC_wrt_ID =>
            new_cmd_batch.sca_cmds(0) := adc_cmd_table(ADC_wrt_ID);
            new_cmd_batch.sca_cmds_data(0).data(31 downto 0) := ecs_cmd_packet.data(31 downto 0);
            new_cmd_batch.sca_cmds_data(0).len := std_logic_vector(to_unsigned( 4 ,8));
            new_cmd_batch.cmd_count := 1;
         when ECS_ADC_rea_ID =>
            new_cmd_batch.sca_cmds(0) := adc_cmd_table(ADC_rea_ID);
            new_cmd_batch.sca_cmds_data(0).data := (others=>'0');
            new_cmd_batch.sca_cmds_data(0).len := (others=>'0');
            new_cmd_batch.cmd_count := 1;
         when INVALID =>
            new_cmd_batch.sca_cmds(0) := x"FF";
            new_cmd_batch.sca_cmds_data(0).data := (others=>'0');
            new_cmd_batch.sca_cmds_data(0).len := (others=>'0');
            new_cmd_batch.cmd_count := 0;
      end case;
   end if;

   sca_cmd_batch_o <= new_cmd_batch;

end process adc_protocol_builder;

adc_protocol_decoder : process ( sca_rpy_batch_i, prot_rpy_en_i) is
   variable new_ecs_rpy_packet      :  ecs_packet_t;
   variable ecs_adc_cmd       :  ecs_adc_cmd_enum;
begin
   --new_ecs_rpy_packet.gbt_nr := (others=>'0');
   new_ecs_rpy_packet.sca_nr  := (others=>'0');
   new_ecs_rpy_packet.sca_ch  := (others=>'0');
   new_ecs_rpy_packet.ecs_cmd := (others=>'0');
   new_ecs_rpy_packet.data := (others=>'0');
   new_ecs_rpy_packet.len := (others=>'0');
```

```
 new_ecs_rpy_packet.protocol_specific  := (others=>'0');
 new_ecs_rpy_packet.err := (others=>'0');

 ecs_adc_cmd := INVALID;

 if prot_rpy_en_i='1' then

    for i in ecs_adc_cmd_table_t'left to ecs_adc_cmd_table_t'right loop
       if sca_rpy_batch_i.ecs_cmd = ecs_adc_cmd_table(i) then
          ecs_adc_cmd := i;
          exit;
       end if;
    end loop;
    new_ecs_rpy_packet.ecs_cmd := ecs_adc_cmd_table(ecs_adc_cmd);

    new_ecs_rpy_packet.sca_ch := sca_rpy_batch_i.ch;
    --new_ecs_rpy_packet.gbt_nr := std_logic_vector( to_unsigned( sca_rpy_batch_i.gbt, 8));
    new_ecs_rpy_packet.sca_nr := std_logic_vector( to_unsigned(sca_rpy_batch_i.sca,8));


    if sca_rpy_batch_i.err /= (sca_rpy_batch_i.err'range =>'0') then
       for j in 0 to sca_rpy_batch_i.err'high -1 loop
       if sca_rpy_batch_i.err(j)/='0' then
          -- \TODO What to do?
          report "There was an error on the GBT-SCA channel operation" & integer'image(j) severity
Note;

          exit;
          end if;
       end loop;
       new_ecs_rpy_packet.err := sca_rpy_batch_i.err;
    end if;


    case ecs_adc_cmd is
    when ECS_ADC_GO =>
       new_ecs_rpy_packet.data(31 downto 0) := sca_rpy_batch_i.sca_rpys_data(0).data(31 downto 0);
       new_ecs_rpy_packet.len := std_logic_vector( to_unsigned(4,8));
    when ECS_ADC_W_INSEL =>
       null;
    when ECS_ADC_R_INSEL =>
       new_ecs_rpy_packet.data(31 downto 0) := sca_rpy_batch_i.sca_rpys_data(0).data(31 downto 0);--4
downto 0
       new_ecs_rpy_packet.len := std_logic_vector( to_unsigned(4,8));
    when ECS_ADC_W_CUREN =>
       null;
    when ECS_ADC_R_CUREN =>
       new_ecs_rpy_packet.data(31 downto 0) := sca_rpy_batch_i.sca_rpys_data(0).data(31 downto 0);
       new_ecs_rpy_packet.len := std_logic_vector( to_unsigned(4,8));
    when ECS_ADC_rea_DATA_Ofs_Ga =>
       new_ecs_rpy_packet.data(31 downto 0) := sca_rpy_batch_i.sca_rpys_data(0).data(31 downto 0);
       new_ecs_rpy_packet.len := std_logic_vector( to_unsigned(4,8));
    when ECS_ADC_rea_DATA_Ofs =>
       new_ecs_rpy_packet.data(31 downto 0) := sca_rpy_batch_i.sca_rpys_data(0).data(31 downto 0);
       new_ecs_rpy_packet.len := std_logic_vector( to_unsigned(4,8));
    when ECS_ADC_rea_DATA =>
       new_ecs_rpy_packet.data(31 downto 0) := sca_rpy_batch_i.sca_rpys_data(0).data(31 downto 0);
       new_ecs_rpy_packet.len := std_logic_vector( to_unsigned(4,8));
    when ECS_ADC_rea_OFFSET =>
       new_ecs_rpy_packet.data(31 downto 0) := sca_rpy_batch_i.sca_rpys_data(0).data(31 downto 0);
```

```
            new_ecs_rpy_packet.len := std_logic_vector( to_unsigned(4,8));
         when ECS_ADC_wrt_CTRL =>
            null;
         when ECS_ADC_rea_CTRL =>
            new_ecs_rpy_packet.data(31 downto 0) := sca_rpy_batch_i.sca_rpys_data(0).data(31 downto 0);
            new_ecs_rpy_packet.len := std_logic_vector( to_unsigned(4,8));
         when ECS_ADC_GO_SingleSlope =>
            null;
         when ECS_ADC_wrt_GainCalibrReg =>
            null;
         when ECS_ADC_rea_GainCalibrReg =>
            new_ecs_rpy_packet.data(31 downto 0) := sca_rpy_batch_i.sca_rpys_data(0).data(31 downto 0);
            new_ecs_rpy_packet.len := std_logic_vector( to_unsigned(4,8));
         when ECS_ADC_wrt_ID =>
            null;
         when ECS_ADC_rea_ID =>
            new_ecs_rpy_packet.data(31 downto 0) := sca_rpy_batch_i.sca_rpys_data(0).data(31 downto 0);
            new_ecs_rpy_packet.len := std_logic_vector( to_unsigned(4,8));
         when INVALID =>
            null;
         end case;


      end if;

      ecs_rpy_packet <= new_ecs_rpy_packet;
   end process adc_protocol_decoder;

end architecture RTL;
```

```
--------------------------------------------------------------------------------
--! @file dac_protocol_driver.vhd
--! @author Cairo Caplan <cairo.caplan@cern.ch>
--! @date April, 2016
--------------------------------------------------------------------------------

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

use work.SCA_Package.all;

entity dac_protocol_driver is
   port (
      -- Interface with the ECS CMD Protocol Arbiter
      prot_cmd_en_i     :  in std_logic;
      ecs_cmd_packet    :  in ecs_packet_t;
      -- Interface with the ECS RPY Protocol Arbiter
      prot_rpy_en_i     :  in std_logic;
      ecs_rpy_packet    :  out   ecs_packet_t;
      -- Interface with the Command Queue manager
      sca_cmd_batch_o      :  out prot_cmd_t;
      sca_rpy_batch_i      :  in prot_cmd_t;
      --Others
      tr_cmd_i       :  in byte_t
   );
end entity dac_protocol_driver;

architecture RTL of dac_protocol_driver is

begin

   dac_protocol_builder : process (
       prot_cmd_en_i, tr_cmd_i, ecs_cmd_packet
   ) is
      variable ecs_dac_cmd       :  ecs_dac_cmd_enum;
      variable new_cmd_batch     :  prot_cmd_t;
   begin


      new_cmd_batch.tr := x"00";
      new_cmd_batch.ch := x"FF";
      new_cmd_batch.cmd_count := 0;
      new_cmd_batch.sca := 0;

      new_cmd_batch.ecs_cmd := (others=>'0');
      new_cmd_batch.ecs_len := (others=>'0');
      new_cmd_batch.err := (others=>'0');

      new_cmd_batch.sca_cmds := (others=>(others=>'0'));
      new_cmd_batch.sca_cmds_data :=  (others=>(others=>(others=>'0')));
      new_cmd_batch.sca_rpys_data :=  (others=>(others=>(others=>'0')));

      ecs_dac_cmd := INVALID;


      if prot_cmd_en_i='1' then

         for i in ecs_dac_cmd_table_t'left to ecs_dac_cmd_table_t'right loop
```

```
            if ecs_cmd_packet.ecs_cmd = ecs_dac_cmd_table(i) then
               ecs_dac_cmd := i;
               exit;
            end if;
         end loop;

         new_cmd_batch.tr := tr_cmd_i;
         new_cmd_batch.ch := ecs_cmd_packet.sca_ch;
         new_cmd_batch.cmd_count := 0;
         --new_cmd_batch.gbt := to_integer(unsigned(ecs_cmd_packet.gbt_nr));
         new_cmd_batch.sca := to_integer(unsigned(ecs_cmd_packet.sca_nr));
         new_cmd_batch.ecs_cmd := ecs_cmd_packet.ecs_cmd;
         new_cmd_batch.ecs_len := ecs_cmd_packet.len;

         case ecs_dac_cmd is
            when ECS_DAC_A_wrt =>
               new_cmd_batch.sca_cmds(0) := dac_cmd_table(DAC_A_wrt);
               new_cmd_batch.sca_cmds_data(0).data(7 downto 0) := ecs_cmd_packet.data(7 downto 0);
               new_cmd_batch.sca_cmds_data(0).len := std_logic_vector(to_unsigned( 1 ,8));
               new_cmd_batch.cmd_count := 1;
            when ECS_DAC_A_read =>
               new_cmd_batch.sca_cmds(0) := dac_cmd_table(DAC_A_read);
               new_cmd_batch.sca_cmds_data(0).data := (others=>'0');
               new_cmd_batch.sca_cmds_data(0).len := (others=>'0');
               new_cmd_batch.cmd_count := 1;
            when ECS_DAC_B_wrt =>
               new_cmd_batch.sca_cmds(0) := dac_cmd_table(DAC_B_wrt);
               new_cmd_batch.sca_cmds_data(0).data(7 downto 0) := ecs_cmd_packet.data(7 downto 0);
               new_cmd_batch.sca_cmds_data(0).len := std_logic_vector(to_unsigned( 1 ,8));
               new_cmd_batch.cmd_count := 1;
            when ECS_DAC_B_read =>
               new_cmd_batch.sca_cmds(0) := dac_cmd_table(DAC_B_read);
               new_cmd_batch.sca_cmds_data(0).data := (others=>'0');
               new_cmd_batch.sca_cmds_data(0).len := (others=>'0');
               new_cmd_batch.cmd_count := 1;
            when ECS_DAC_C_wrt =>
               new_cmd_batch.sca_cmds(0) := dac_cmd_table(DAC_C_wrt);
               new_cmd_batch.sca_cmds_data(0).data(7 downto 0) := ecs_cmd_packet.data(7 downto 0);
               new_cmd_batch.sca_cmds_data(0).len := std_logic_vector(to_unsigned( 1 ,8));
               new_cmd_batch.cmd_count := 1;
            when ECS_DAC_C_read =>
               new_cmd_batch.sca_cmds(0) := dac_cmd_table(DAC_C_read);
               new_cmd_batch.sca_cmds_data(0).data := (others=>'0');
               new_cmd_batch.sca_cmds_data(0).len := (others=>'0');
               new_cmd_batch.cmd_count := 1;
            when ECS_DAC_D_wrt =>
               new_cmd_batch.sca_cmds(0) := dac_cmd_table(DAC_D_wrt);
               new_cmd_batch.sca_cmds_data(0).data(7 downto 0) := ecs_cmd_packet.data(7 downto 0);
               new_cmd_batch.sca_cmds_data(0).len := std_logic_vector(to_unsigned( 1 ,8));
               new_cmd_batch.cmd_count := 1;
            when ECS_DAC_D_read =>
               new_cmd_batch.sca_cmds(0) := dac_cmd_table(DAC_D_read);
               new_cmd_batch.sca_cmds_data(0).data := (others=>'0');
               new_cmd_batch.sca_cmds_data(0).len := (others=>'0');
               new_cmd_batch.cmd_count := 1;
            when INVALID =>
               null;
         end case;
      end if;
```

```
   sca_cmd_batch_o <= new_cmd_batch;
end process dac_protocol_builder;

dac_protocol_decoder : process(sca_rpy_batch_i, prot_rpy_en_i) is
   variable new_ecs_rpy_packet      :  ecs_packet_t;
   variable ecs_dac_cmd       :  ecs_dac_cmd_enum;
begin
   --new_ecs_rpy_packet.gbt_nr := (others=>'0');
   new_ecs_rpy_packet.sca_nr  := (others=>'0');
   new_ecs_rpy_packet.sca_ch  := (others=>'0');
   new_ecs_rpy_packet.ecs_cmd := (others=>'0');
   new_ecs_rpy_packet.data := (others=>'0');
   new_ecs_rpy_packet.len := (others=>'0');
   new_ecs_rpy_packet.protocol_specific  := (others=>'0');
   new_ecs_rpy_packet.err := (others=>'0');

   ecs_dac_cmd := INVALID;

   if prot_rpy_en_i='1' then

      for i in ecs_dac_cmd_table_t'left to ecs_dac_cmd_table_t'right loop
         if sca_rpy_batch_i.ecs_cmd = ecs_dac_cmd_table(i) then
            ecs_dac_cmd := i;
            exit;
         end if;
      end loop;
      new_ecs_rpy_packet.ecs_cmd := ecs_dac_cmd_table(ecs_dac_cmd);

      new_ecs_rpy_packet.sca_ch := sca_rpy_batch_i.ch;
      --new_ecs_rpy_packet.gbt_nr := std_logic_vector( to_unsigned( sca_rpy_batch_i.gbt, 8));
      new_ecs_rpy_packet.sca_nr := std_logic_vector( to_unsigned(sca_rpy_batch_i.sca,8));
      if sca_rpy_batch_i.err /= (sca_rpy_batch_i.err'range =>'0') then
         for j in 0 to sca_rpy_batch_i.err'high -1 loop
         if sca_rpy_batch_i.err(j)/='0' then
            -- \TODO What to do?
            report "There was an error on the GBT-SCA channel operation" & integer'image(j) severity
 Note;

            exit;
            end if;
         end loop;
         new_ecs_rpy_packet.err := sca_rpy_batch_i.err;
      end if;

      new_ecs_rpy_packet.protocol_specific(15 downto 0) := (others=>'0');

      case ecs_dac_cmd is
         when ECS_DAC_A_wrt =>
            null;
         when ECS_DAC_A_read =>
            new_ecs_rpy_packet.data(7 downto 0) := sca_rpy_batch_i.sca_rpys_data(0).data(7 downto 0);
            new_ecs_rpy_packet.len := std_logic_vector( to_unsigned(1,8));
         when ECS_DAC_B_wrt =>
            null;
         when ECS_DAC_B_read =>
            new_ecs_rpy_packet.data(7 downto 0) := sca_rpy_batch_i.sca_rpys_data(0).data(7 downto 0);
            new_ecs_rpy_packet.len := std_logic_vector( to_unsigned(1,8));
         when ECS_DAC_C_wrt =>
            null;
         when ECS_DAC_C_read =>
```

```
            new_ecs_rpy_packet.data(7 downto 0) := sca_rpy_batch_i.sca_rpys_data(0).data(7 downto 0);
            new_ecs_rpy_packet.len := std_logic_vector( to_unsigned(1,8));
         when ECS_DAC_D_wrt =>
            null;
         when ECS_DAC_D_read =>
            new_ecs_rpy_packet.data(7 downto 0) := sca_rpy_batch_i.sca_rpys_data(0).data(7 downto 0);
            new_ecs_rpy_packet.len := std_logic_vector( to_unsigned(1,8));
         when INVALID =>
            null;
      end case;
   end if;

   ecs_rpy_packet <= new_ecs_rpy_packet;
end process dac_protocol_decoder;


end architecture RTL;
```

```
--------------------------------------------------------------------------------
--! @file gpio_protocol_driver.vhd
--! @author Cairo Caplan <cairo.caplan@cern.ch>;
--! modified by Joao Barbosa <joao.vitor.viana.barbosa@cern.ch>
--! @date April, 2016
--------------------------------------------------------------------------------

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

use work.SCA_Package.all;

--! @brief This is the protocol driver for the GPIO bus of the GBT-SCA.
--! This block currently does not support GPIO interrupts.
entity gpio_protocol_driver is
   port (
      --! @name Interface with the ECS CMD Protocol Arbiter
      --! @{
      --! Enable new
      prot_cmd_en_i     :  in std_logic;
      ecs_cmd_packet    :  in ecs_packet_t;
      --! @}
      --! @name Interface with the ECS RPY Protocol Arbiter
      --! @{
      prot_rpy_en_i     :  in std_logic;
      ecs_rpy_packet    :  out   ecs_packet_t;
      --! @}
      --! @name  Interface with the Command Queue manager
      --! @{
      sca_cmd_batch_o      :  out prot_cmd_t;
      sca_rpy_batch_i      :  in prot_cmd_t;
      --! @}
      --! Transaction Number of the Command
      tr_cmd_i       :  in byte_t
   );
end entity gpio_protocol_driver;



architecture RTL of gpio_protocol_driver is

begin

   --! @name GPIO Protocol Builder
   --! @brief This process builds a group of SCA payload command packets, based on the
   --! ECS CMD packet given CMD Protocol Arbiter on the Protocol Layer
   gpio_protocol_builder : process (
       prot_cmd_en_i, tr_cmd_i, ecs_cmd_packet
   ) is
      variable ecs_gpio_cmd   :  ecs_gpio_cmd_enum;
      variable new_cmd_batch  :  prot_cmd_t;
   begin


      new_cmd_batch.tr := x"00";
      new_cmd_batch.ch := x"FF";
      new_cmd_batch.cmd_count := 0;
      new_cmd_batch.sca := 0;
```

```
new_cmd_batch.ecs_cmd := (others=>'0');
new_cmd_batch.ecs_len := (others=>'0');
new_cmd_batch.err := (others=>'0');

new_cmd_batch.sca_cmds := (others=>(others=>'0'));
new_cmd_batch.sca_cmds_data :=  (others=>(others=>(others=>'0')));
new_cmd_batch.sca_rpys_data :=  (others=>(others=>(others=>'0')));

ecs_gpio_cmd := INVALID;


if prot_cmd_en_i='1' then

   for i in ecs_gpio_cmd_table_t'left to ecs_gpio_cmd_table_t'right loop
      if ecs_cmd_packet.ecs_cmd = ecs_gpio_cmd_table(i) then
         ecs_gpio_cmd := i;
         exit;
      end if;
   end loop;

   new_cmd_batch.tr := tr_cmd_i;
   new_cmd_batch.ch := ecs_cmd_packet.sca_ch;
   new_cmd_batch.cmd_count := 0;
   --new_cmd_batch.gbt := to_integer(unsigned(ecs_cmd_packet.gbt_nr));
   new_cmd_batch.sca := to_integer(unsigned(ecs_cmd_packet.sca_nr));
   new_cmd_batch.ecs_cmd := ecs_cmd_packet.ecs_cmd;
   new_cmd_batch.ecs_len := ecs_cmd_packet.len;

   case ecs_gpio_cmd is
      when ECS_GPIO_REA_IN_DATA =>
         new_cmd_batch.sca_cmds(0) := gpio_cmd_table(GPIO_REA_IN_DATA);
         new_cmd_batch.sca_cmds_data(0).data := (others=>'0');
         new_cmd_batch.sca_cmds_data(0).len := (others=>'0');
         new_cmd_batch.cmd_count := 1;
      when ECS_GPIO_SET_OUT_DATA =>
         new_cmd_batch.sca_cmds(0) := gpio_cmd_table(GPIO_SET_OUT_DATA);
         new_cmd_batch.sca_cmds_data(0).data := ecs_cmd_packet.data(31 downto 0);
         new_cmd_batch.sca_cmds_data(0).len := std_logic_vector(to_unsigned( 4 ,8));
         new_cmd_batch.cmd_count := 1;
      when ECS_GPIO_REA_OUT_DATA =>
         new_cmd_batch.sca_cmds(0) := gpio_cmd_table(GPIO_REA_OUT_DATA);
         new_cmd_batch.sca_cmds_data(0).data := (others=>'0');
         new_cmd_batch.sca_cmds_data(0).len := (others=>'0');
         new_cmd_batch.cmd_count := 1;
      when ECS_GPIO_SET_IO_DIR =>
         new_cmd_batch.sca_cmds(0) := gpio_cmd_table(GPIO_SET_IO_DIR);
         new_cmd_batch.sca_cmds_data(0).data := ecs_cmd_packet.data(31 downto 0);
         new_cmd_batch.sca_cmds_data(0).len := std_logic_vector(to_unsigned( 4 ,8));
         new_cmd_batch.cmd_count := 1;
      when ECS_GPIO_REA_IO_DIR =>
         new_cmd_batch.sca_cmds(0) := gpio_cmd_table(GPIO_REA_IO_DIR);
         new_cmd_batch.sca_cmds_data(0).data := (others=>'0');
         new_cmd_batch.sca_cmds_data(0).len := (others=>'0');
         new_cmd_batch.cmd_count := 1;
      when ECS_GPIO_SET_INTEN =>
         new_cmd_batch.sca_cmds(0) := gpio_cmd_table(GPIO_SET_INTEN);
         new_cmd_batch.sca_cmds_data(0).data := ecs_cmd_packet.data(31 downto 0);
         new_cmd_batch.sca_cmds_data(0).len := std_logic_vector(to_unsigned( 4 ,8));
         new_cmd_batch.cmd_count := 1;
```

```
        when ECS_GPIO_REA_INTEN =>
           new_cmd_batch.sca_cmds(0) := gpio_cmd_table(GPIO_REA_INTEN);
           new_cmd_batch.sca_cmds_data(0).data := (others=>'0');
           new_cmd_batch.sca_cmds_data(0).len := (others=>'0');
           new_cmd_batch.cmd_count := 1;
        when ECS_GPIO_SET_TRIG =>
           new_cmd_batch.sca_cmds(0) := gpio_cmd_table(GPIO_SET_TRIG);
           new_cmd_batch.sca_cmds_data(0).data := ecs_cmd_packet.data(31 downto 0);
           new_cmd_batch.sca_cmds_data(0).len := std_logic_vector(to_unsigned( 4 ,8));
           new_cmd_batch.cmd_count := 1;
        when ECS_GPIO_REA_TRIG =>
           new_cmd_batch.sca_cmds(0) := gpio_cmd_table(GPIO_REA_TRIG);
           new_cmd_batch.sca_cmds_data(0).data := (others=>'0');
           new_cmd_batch.sca_cmds_data(0).len := (others=>'0');
           new_cmd_batch.cmd_count := 1;
        when ECS_GPIO_SET_GIE =>
           new_cmd_batch.sca_cmds(0) := gpio_cmd_table(GPIO_SET_GIE);
           new_cmd_batch.sca_cmds_data(0).data(0) := ecs_cmd_packet.data(0);
           new_cmd_batch.sca_cmds_data(0).len := std_logic_vector(to_unsigned( 1 ,8));
           new_cmd_batch.cmd_count := 1;
        when ECS_GPIO_REA_GIE =>
           new_cmd_batch.sca_cmds(0) := gpio_cmd_table(GPIO_REA_GIE);
           new_cmd_batch.sca_cmds_data(0).data := (others=>'0');
           new_cmd_batch.sca_cmds_data(0).len := (others=>'0');
           new_cmd_batch.cmd_count := 1;
        when ECS_GPIO_SET_INT =>
           new_cmd_batch.sca_cmds(0) := gpio_cmd_table(GPIO_SET_INT);
           new_cmd_batch.sca_cmds_data(0).data := ecs_cmd_packet.data(31 downto 0);
           new_cmd_batch.sca_cmds_data(0).len := std_logic_vector(to_unsigned( 4 ,8));
           new_cmd_batch.cmd_count := 1;
        when ECS_GPIO_REA_INT =>
           new_cmd_batch.sca_cmds(0) := gpio_cmd_table(GPIO_REA_INT);
           new_cmd_batch.sca_cmds_data(0).data := (others=>'0');
           new_cmd_batch.sca_cmds_data(0).len := (others=>'0');
           new_cmd_batch.cmd_count := 1;
        when ECS_GPIO_SET_CLK_SEL =>
           new_cmd_batch.sca_cmds(0) := gpio_cmd_table(GPIO_SET_CLK_SEL);
           new_cmd_batch.sca_cmds_data(0).data := ecs_cmd_packet.data(31 downto 0);
           new_cmd_batch.sca_cmds_data(0).len := std_logic_vector(to_unsigned( 4 ,8));
        when ECS_GPIO_REA_CLK_SEL =>
           new_cmd_batch.sca_cmds(0) := gpio_cmd_table(GPIO_REA_CLK_SEL);
           new_cmd_batch.sca_cmds_data(0).data := (others=>'0');
           new_cmd_batch.sca_cmds_data(0).len := (others=>'0');
           new_cmd_batch.cmd_count := 1;
        when ECS_GPIO_SET_EDG =>
           new_cmd_batch.sca_cmds(0) := gpio_cmd_table(GPIO_SET_EDG);
           new_cmd_batch.sca_cmds_data(0).data := ecs_cmd_packet.data(31 downto 0);
           new_cmd_batch.sca_cmds_data(0).len := std_logic_vector(to_unsigned( 4 ,8));
        when ECS_GPIO_REA_EDG =>
           new_cmd_batch.sca_cmds(0) := gpio_cmd_table(GPIO_REA_EDG);
           new_cmd_batch.sca_cmds_data(0).data := (others=>'0');
           new_cmd_batch.sca_cmds_data(0).len := (others=>'0');
           new_cmd_batch.cmd_count := 1;
        when INVALID =>
           null;
     end case;
  end if;

  sca_cmd_batch_o <= new_cmd_batch;
```

```
end process gpio_protocol_builder;

--! @name GPIO Protocol Decoder
--! @brief This process decodes a group of SCA payload reply packets, given
--! by the MAC Layer, into a new ECS RPY packet to be passed to the RPY
--! Protocol Arbiter on the Protocol Layer
gpio_protocol_decoder : process(sca_rpy_batch_i, prot_rpy_en_i) is
   variable new_ecs_rpy_packet      :  ecs_packet_t;
   variable ecs_gpio_cmd         :  ecs_gpio_cmd_enum;
begin
   --new_ecs_rpy_packet.gbt_nr := (others=>'0');
   new_ecs_rpy_packet.sca_nr  := (others=>'0');
   new_ecs_rpy_packet.sca_ch  := (others=>'0');
   new_ecs_rpy_packet.ecs_cmd := (others=>'0');
   new_ecs_rpy_packet.data := (others=>'0');
   new_ecs_rpy_packet.len := (others=>'0');
   new_ecs_rpy_packet.protocol_specific  := (others=>'0');
   new_ecs_rpy_packet.err := (others=>'0');

   ecs_gpio_cmd := INVALID;

   if prot_rpy_en_i='1' then
      for i in ecs_gpio_cmd_table_t'left to ecs_gpio_cmd_table_t'right loop
         if sca_rpy_batch_i.ecs_cmd = ecs_gpio_cmd_table(i) then
            ecs_gpio_cmd := i;
            exit;
         end if;
      end loop;
      new_ecs_rpy_packet.ecs_cmd := ecs_gpio_cmd_table(ecs_gpio_cmd);

      new_ecs_rpy_packet.sca_ch := sca_rpy_batch_i.ch;
      --new_ecs_rpy_packet.gbt_nr := std_logic_vector( to_unsigned( sca_rpy_batch_i.gbt, 8));
      new_ecs_rpy_packet.sca_nr := std_logic_vector( to_unsigned(sca_rpy_batch_i.sca,8));

      if sca_rpy_batch_i.err /= (sca_rpy_batch_i.err'range =>'0') then
         for j in 0 to sca_rpy_batch_i.err'high -1 loop
         if sca_rpy_batch_i.err(j)/='0' then
            -- \TODO What to do?
            report "There was an error on the GBT-SCA channel operation" & integer'image(j) severity
 Note;

            exit;
            end if;
         end loop;
         new_ecs_rpy_packet.err := sca_rpy_batch_i.err;
      end if;

      new_ecs_rpy_packet.protocol_specific(15 downto 0) := (others=>'0');

      case ecs_gpio_cmd is
         when ECS_GPIO_REA_IN_DATA =>
            new_ecs_rpy_packet.data(31 downto 0) := sca_rpy_batch_i.sca_rpys_data(0).data;
            new_ecs_rpy_packet.len := std_logic_vector( to_unsigned(4,8));
         when ECS_GPIO_SET_OUT_DATA =>
            null;
         when ECS_GPIO_REA_OUT_DATA =>
            new_ecs_rpy_packet.data(31 downto 0) := sca_rpy_batch_i.sca_rpys_data(0).data;
            new_ecs_rpy_packet.len := std_logic_vector( to_unsigned(4,8));
         when ECS_GPIO_SET_IO_DIR =>
            null;
```

```
            when ECS_GPIO_REA_IO_DIR =>
               new_ecs_rpy_packet.data(31 downto 0) := sca_rpy_batch_i.sca_rpys_data(0).data;
               new_ecs_rpy_packet.len := std_logic_vector( to_unsigned(4,8));
            when ECS_GPIO_SET_INTEN =>
               null;
            when ECS_GPIO_REA_INTEN =>
               new_ecs_rpy_packet.data(31 downto 0) := sca_rpy_batch_i.sca_rpys_data(0).data;
               new_ecs_rpy_packet.len := std_logic_vector( to_unsigned(4,8));
            when ECS_GPIO_SET_TRIG =>
               null;
            when ECS_GPIO_REA_TRIG =>
               new_ecs_rpy_packet.data(31 downto 0) := sca_rpy_batch_i.sca_rpys_data(0).data;
               new_ecs_rpy_packet.len := std_logic_vector( to_unsigned(4,8));
            when ECS_GPIO_SET_GIE =>
               null;
            when ECS_GPIO_REA_GIE =>
               new_ecs_rpy_packet.data(31 downto 0) := sca_rpy_batch_i.sca_rpys_data(0).data;--only for
    testing TODO
               new_ecs_rpy_packet.len := std_logic_vector( to_unsigned(1,8));
            when ECS_GPIO_SET_INT =>
               null;
            when ECS_GPIO_REA_INT =>
               new_ecs_rpy_packet.data(31 downto 0) := sca_rpy_batch_i.sca_rpys_data(0).data;
               new_ecs_rpy_packet.len := std_logic_vector( to_unsigned(4,8));
            when ECS_GPIO_SET_CLK_SEL =>
               null;
            when ECS_GPIO_REA_CLK_SEL =>
               new_ecs_rpy_packet.data(31 downto 0) := sca_rpy_batch_i.sca_rpys_data(0).data;
               new_ecs_rpy_packet.len := std_logic_vector( to_unsigned(4,8));
            when ECS_GPIO_SET_EDG =>
               null;
            when ECS_GPIO_REA_EDG =>
               new_ecs_rpy_packet.data(31 downto 0) := sca_rpy_batch_i.sca_rpys_data(0).data;
               new_ecs_rpy_packet.len := std_logic_vector( to_unsigned(4,8));
            when INVALID =>
               null;
         end case;
      end if;

      ecs_rpy_packet <= new_ecs_rpy_packet;
   end process gpio_protocol_decoder;

end architecture RTL;
```

```
--------------------------------------------------------------------------------
--! @file i2c_protocol_driver.vhd
--! @author Cairo Caplan <cairo.caplan@cern.ch>
--! @date April, 2016
--------------------------------------------------------------------------------


library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

use work.SCA_Package.all;


entity i2c_protocol_driver is
   port (
      -- Interface with the ECS CMD Protocol Arbiter
      prot_cmd_en_i  :  in std_logic;
      ecs_cmd_packet :  in ecs_packet_t;
      -- Interface with the ECS RPY Protocol Arbiter
      prot_rpy_en_i  :  in std_logic;
      ecs_rpy_packet :  out   ecs_packet_t;
      -- Interface with the Command Queue manager
      sca_cmd_batch_o   :  out prot_cmd_t;
      sca_rpy_batch_i   :  in prot_cmd_t;
      --Others
      tr_cmd_i    :  in byte_t
   );
end entity i2c_protocol_driver;

architecture RTL of i2c_protocol_driver is

begin

   --For WRITE, READ commands:
   -- Command: ( STATUS bits(Reply only)=4b,
   --       NBytes = 4b,
   --       2b = Speed,3b not used, ADDR_6_0 = 7b)

   --For WRITE_EXT, READ_EXT
   -- Command: (STATUS bits(Reply only)=4b,
   --       NBytes = 4b,
   --       2b = Speed,  ADDR_9_0 = 10b)


   i2c_protocol_builder : process (
       prot_cmd_en_i, tr_cmd_i, ecs_cmd_packet
   ) is

      variable ecs_i2c_cmd       :  ecs_i2c_cmd_enum;
      variable protocol_specific :  std_logic_vector(15 downto 0);

      variable new_cmd_batch     :  prot_cmd_t;

      variable NByte          :  natural range 0 to 16;
      variable j              :  natural range 0 to 6;
      variable k              :  natural range 1 to 16;
   begin
```

```
new_cmd_batch.tr := x"00";
new_cmd_batch.ch := x"FF";
new_cmd_batch.cmd_count := 0;
new_cmd_batch.sca := 0;

new_cmd_batch.ecs_cmd := (others=>'0');
new_cmd_batch.ecs_len := (others=>'0');
new_cmd_batch.err := (others=>'0');

new_cmd_batch.sca_cmds := (others=>(others=>'0'));
new_cmd_batch.sca_cmds_data :=  (others=>(others=>(others=>'0')));
new_cmd_batch.sca_rpys_data :=  (others=>(others=>(others=>'0')));

ecs_i2c_cmd := INVALID;


if prot_cmd_en_i='1' then
   for i in ecs_i2c_table_t'left to ecs_i2c_table_t'right loop
      if ecs_cmd_packet.ecs_cmd = ecs_i2c_table(i) then
         ecs_i2c_cmd := i;
         exit;
      end if;
   end loop;
   protocol_specific := ecs_cmd_packet.protocol_specific;

   new_cmd_batch.tr := tr_cmd_i;
   new_cmd_batch.ch := ecs_cmd_packet.sca_ch;
   new_cmd_batch.cmd_count := 0;
   --new_cmd_batch.gbt := to_integer(unsigned(ecs_cmd_packet.gbt_nr));
   new_cmd_batch.sca := to_integer(unsigned(ecs_cmd_packet.sca_nr));

   new_cmd_batch.ecs_cmd := ecs_cmd_packet.ecs_cmd;

   for i in 0 to new_cmd_batch.sca_cmds'high loop
      new_cmd_batch.sca_cmds(i) := (others=>'0');
      new_cmd_batch.sca_cmds_data(i).data := (others=>'0');
      new_cmd_batch.sca_cmds_data(i).len := (others=>'0');
      new_cmd_batch.sca_rpys_data(i).data := (others=>'0');
      new_cmd_batch.sca_rpys_data(i).len := (others=>'0');
   end loop;
   j:=0;

   if to_integer(unsigned(ecs_cmd_packet.len)) >16 then
      NByte := 0;
      new_cmd_batch.err :=x"01";
   else
      NByte := to_integer(unsigned(ecs_cmd_packet.len));
   end if;
   new_cmd_batch.ecs_len := ecs_cmd_packet.len;
   report "NByte = " & natural'image(NByte);

   -- Protocol Specific field: {"XXXX",  Freq = 2b, I2C Address = 10b}


   case ecs_i2c_cmd is
      when ECS_I2C_S_7B_W =>
         new_cmd_batch.cmd_count := 1;
         new_cmd_batch.sca_cmds(0) := i2c_cmd_table(I2C_S_7B_W);
         new_cmd_batch.sca_cmds_data(0).data(15 downto 0) := ecs_cmd_packet.data(15 downto 0);
         new_cmd_batch.sca_cmds_data(0).len := std_logic_vector(to_unsigned(2, 8));
```

```
when ECS_I2C_S_7B_R =>
   new_cmd_batch.cmd_count := 1;
   new_cmd_batch.sca_cmds(0) := i2c_cmd_table(I2C_S_7B_R);
   new_cmd_batch.sca_cmds_data(0).data(7 downto 0) := ecs_cmd_packet.data(7 downto 0);
   new_cmd_batch.sca_cmds_data(0).len := std_logic_vector(to_unsigned(1, 8));
when ECS_I2C_S_10B_W =>
   new_cmd_batch.cmd_count := 1;
   new_cmd_batch.sca_cmds(0) := i2c_cmd_table(I2C_S_10B_W);
   new_cmd_batch.sca_cmds_data(0).data(23 downto 0) := ecs_cmd_packet.data(23 downto 0);
   new_cmd_batch.sca_cmds_data(0).len := std_logic_vector(to_unsigned(3, 8));
when ECS_I2C_S_10B_R =>
   new_cmd_batch.cmd_count := 1;
   new_cmd_batch.sca_cmds(0) := i2c_cmd_table(I2C_S_10B_R);
   new_cmd_batch.sca_cmds_data(0).data(15 downto 0) := ecs_cmd_packet.data(15 downto 0);
   new_cmd_batch.sca_cmds_data(0).len := std_logic_vector(to_unsigned(2, 8));
when ECS_I2C_M_7B_W =>
   new_cmd_batch.cmd_count := 1;
   new_cmd_batch.sca_cmds(0) := i2c_cmd_table(I2C_M_7B_R);
   new_cmd_batch.sca_cmds_data(0).data(7 downto 0) := ecs_cmd_packet.data(7 downto 0);
   new_cmd_batch.sca_cmds_data(0).len := std_logic_vector(to_unsigned(1, 8));
when ECS_I2C_M_7B_R =>
   new_cmd_batch.cmd_count := 1;
   new_cmd_batch.sca_cmds(0) := i2c_cmd_table(I2C_M_7B_R);
   new_cmd_batch.sca_cmds_data(0).data(7 downto 0) := ecs_cmd_packet.data(7 downto 0);
   new_cmd_batch.sca_cmds_data(0).len := std_logic_vector(to_unsigned(1, 8));
when ECS_I2C_M_10B_W =>
   new_cmd_batch.cmd_count := 1;
   new_cmd_batch.sca_cmds(0) := i2c_cmd_table(I2C_M_10B_W );
   new_cmd_batch.sca_cmds_data(0).data(15 downto 0) := ecs_cmd_packet.data(15 downto 0);
   new_cmd_batch.sca_cmds_data(0).len := std_logic_vector(to_unsigned(2, 8));
when ECS_I2C_M_10B_R =>
   new_cmd_batch.cmd_count := 1;
   new_cmd_batch.sca_cmds(0) := i2c_cmd_table(I2C_M_10B_R);
   new_cmd_batch.sca_cmds_data(0).data(15 downto 0) := ecs_cmd_packet.data(15 downto 0);
   new_cmd_batch.sca_cmds_data(0).len := std_logic_vector(to_unsigned(2, 8));
when ECS_I2C_RMW_AND =>
   new_cmd_batch.cmd_count := 1;
   new_cmd_batch.sca_cmds(0) := i2c_cmd_table(I2C_RMW_AND);
   new_cmd_batch.sca_cmds_data(0).data(15 downto 0) := ecs_cmd_packet.data(15 downto 0);
   new_cmd_batch.sca_cmds_data(0).len := std_logic_vector(to_unsigned(2, 8));
when ECS_I2C_RMW_OR =>
   new_cmd_batch.cmd_count := 1;
   new_cmd_batch.sca_cmds(0) := i2c_cmd_table(I2C_RMW_OR);
   new_cmd_batch.sca_cmds_data(0).data(15 downto 0) := ecs_cmd_packet.data(15 downto 0);
   new_cmd_batch.sca_cmds_data(0).len := std_logic_vector(to_unsigned(2, 8));
when ECS_I2C_RMW_XOR =>
   new_cmd_batch.cmd_count := 1;
   new_cmd_batch.sca_cmds(0) := i2c_cmd_table(I2C_RMW_XOR);
   new_cmd_batch.sca_cmds_data(0).data(15 downto 0) := ecs_cmd_packet.data(15 downto 0);
   new_cmd_batch.sca_cmds_data(0).len := std_logic_vector(to_unsigned(2, 8));
when ECS_I2C_W_DATA0 =>
   new_cmd_batch.cmd_count := 1;
   new_cmd_batch.sca_cmds(0) := i2c_cmd_table(I2C_W_DATA0);
   new_cmd_batch.sca_cmds_data(0).data(31 downto 0) := ecs_cmd_packet.data(31 downto 0);
   new_cmd_batch.sca_cmds_data(0).len := std_logic_vector(to_unsigned(4, 8));
when ECS_I2C_R_DATA0 =>
   new_cmd_batch.cmd_count := 1;
   new_cmd_batch.sca_cmds(0) := i2c_cmd_table(I2C_R_DATA0);
   new_cmd_batch.sca_cmds_data(0).data := (others=>'0');
   new_cmd_batch.sca_cmds_data(0).len := (others=>'0');
```

```
when ECS_I2C_W_DATA1 =>
   new_cmd_batch.cmd_count := 1;
   new_cmd_batch.sca_cmds(0) := i2c_cmd_table(I2C_W_DATA1);
   new_cmd_batch.sca_cmds_data(0).data(31 downto 0) := ecs_cmd_packet.data(31 downto 0);
   new_cmd_batch.sca_cmds_data(0).len := std_logic_vector(to_unsigned(4, 8));
when ECS_I2C_R_DATA1 =>
   new_cmd_batch.cmd_count := 1;
   new_cmd_batch.sca_cmds(0) := i2c_cmd_table(I2C_R_DATA1);
   new_cmd_batch.sca_cmds_data(0).data := (others=>'0');
   new_cmd_batch.sca_cmds_data(0).len := (others=>'0');
when ECS_I2C_W_DATA2 =>
   new_cmd_batch.cmd_count := 1;
   new_cmd_batch.sca_cmds(0) := i2c_cmd_table(I2C_W_DATA2);
   new_cmd_batch.sca_cmds_data(0).data(31 downto 0) := ecs_cmd_packet.data(31 downto 0);
   new_cmd_batch.sca_cmds_data(0).len := std_logic_vector(to_unsigned(4, 8));
when ECS_I2C_R_DATA2 =>
   new_cmd_batch.cmd_count := 1;
   new_cmd_batch.sca_cmds(0) := i2c_cmd_table(I2C_R_DATA2);
   new_cmd_batch.sca_cmds_data(0).data := (others=>'0');
   new_cmd_batch.sca_cmds_data(0).len := (others=>'0');
when ECS_I2C_W_DATA3 =>
   new_cmd_batch.cmd_count := 1;
   new_cmd_batch.sca_cmds(0) := i2c_cmd_table(I2C_W_DATA3);
   new_cmd_batch.sca_cmds_data(0).data(31 downto 0) := ecs_cmd_packet.data(31 downto 0);
   new_cmd_batch.sca_cmds_data(0).len := std_logic_vector(to_unsigned(4, 8));
when ECS_I2C_R_DATA3 =>
   new_cmd_batch.cmd_count := 1;
   new_cmd_batch.sca_cmds(0) := i2c_cmd_table(I2C_R_DATA2);
   new_cmd_batch.sca_cmds_data(0).data := (others=>'0');
   new_cmd_batch.sca_cmds_data(0).len := (others=>'0');
when ECS_I2C_W_CTRL =>
   new_cmd_batch.cmd_count := 1;
   new_cmd_batch.sca_cmds(0) := i2c_cmd_table(I2C_W_CTRL);
   new_cmd_batch.sca_cmds_data(0).data(7 downto 0) :=
      ecs_cmd_packet.data(7 downto 0) ;
   new_cmd_batch.sca_cmds_data(0).data(31 downto 8) := (others=>'0');
   new_cmd_batch.sca_cmds_data(0).len := std_logic_vector(to_unsigned(2, 8));
when ECS_I2C_R_CTRL =>
   new_cmd_batch.cmd_count := 1;
   new_cmd_batch.sca_cmds(0) := i2c_cmd_table(I2C_R_CTRL);
   new_cmd_batch.sca_cmds_data(0).data := (others=>'0');
   new_cmd_batch.sca_cmds_data(0).len := (others=>'0');
when ECS_I2C_W_MSK =>
   new_cmd_batch.cmd_count := 1;
   new_cmd_batch.sca_cmds(0) := i2c_cmd_table(I2C_W_MSK);
   new_cmd_batch.sca_cmds_data(0).data(7 downto 0) :=
      ecs_cmd_packet.data(7 downto 0) ;
   new_cmd_batch.sca_cmds_data(0).data(31 downto 8) := (others=>'0');
   new_cmd_batch.sca_cmds_data(0).len := std_logic_vector(to_unsigned(2, 8));
when ECS_I2C_R_MSK =>
   new_cmd_batch.cmd_count := 1;
   new_cmd_batch.sca_cmds(0) := i2c_cmd_table(I2C_R_MSK);
   new_cmd_batch.sca_cmds_data(0).data := (others=>'0');
   new_cmd_batch.sca_cmds_data(0).len := (others=>'0');
when ECS_I2C_R_STR =>
   new_cmd_batch.cmd_count := 1;
   new_cmd_batch.sca_cmds(0) := i2c_cmd_table(I2C_R_STR);
   new_cmd_batch.sca_cmds_data(0).data := (others=>'0');
   new_cmd_batch.sca_cmds_data(0).len := (others=>'0');
when ECS_I2C_WRITE | ECS_I2C_WRITE_EXT | ECS_I2C_READ | ECS_I2C_READ_EXT =>
```

```
            -- First write the I2C CTRL register
            --{'0',I2CNBYTE,I2CFREQ}
            new_cmd_batch.sca_cmds(0) := i2c_cmd_table(I2C_W_CTRL);
            new_cmd_batch.sca_cmds_data(0).data(7 downto 0) :=
               '0' & ecs_cmd_packet.len(4 downto 0) & protocol_specific(11 downto 10);
            new_cmd_batch.sca_cmds_data(0).data(31 downto 8) := (others=>'0');
            new_cmd_batch.sca_cmds_data(0).len := std_logic_vector(to_unsigned(1, 8));

            j := 1;

            if ecs_i2c_cmd = ECS_I2C_WRITE or ecs_i2c_cmd = ECS_I2C_WRITE_EXT then


               if NByte>1 then --MULTI BYTE WRITE

                  new_cmd_batch.cmd_count := 1 + (((NByte-1)/4) +1 ) + 1;
                  --On multi byte writes first you have to fill the data registers
                  -- then issue the I2C Write commmand

                  --k:=1;
                  --For each byte on the ecs_cmd_packet
                  for i in 1 to 4  loop
                     if i > ((NByte-1)/4)+1 then
                        exit;
                     end if;
                     new_cmd_batch.sca_cmds_data(i).data := ecs_cmd_packet.data(32*i -1 downto
32*(i-1));
                     new_cmd_batch.sca_cmds_data(i).len := std_logic_vector(to_unsigned(4, 8));
                     case (i) is
                     when 1 =>
                        new_cmd_batch.sca_cmds(1) := i2c_cmd_table(I2C_W_DATA0);
                     when 2=>
                        new_cmd_batch.sca_cmds(2) := i2c_cmd_table(I2C_W_DATA1);
                     when 3=>
                        new_cmd_batch.sca_cmds(3) := i2c_cmd_table(I2C_W_DATA2);
                     when others=> --4
                        new_cmd_batch.sca_cmds(4) := i2c_cmd_table(I2C_W_DATA3);
                     end case;
                     j := j + 1;
                  end loop;
                  if ecs_i2c_cmd = ECS_I2C_WRITE then
                     new_cmd_batch.sca_cmds(j) := i2c_cmd_table(I2C_M_7B_W);
                     new_cmd_batch.sca_cmds_data(j).data(31 downto 8) := (others=>'0');
                     new_cmd_batch.sca_cmds_data(j).data(7 downto 0) :=  "0" & protocol_specific(6
downto 0);
                     new_cmd_batch.sca_cmds_data(j).len := std_logic_vector(to_unsigned(1, 8));
                  else
                     new_cmd_batch.sca_cmds(j) := i2c_cmd_table(I2C_M_10B_W);
                     new_cmd_batch.sca_cmds_data(j).data(31 downto 16) := (others=>'0');
                     new_cmd_batch.sca_cmds_data(j).data(7 downto 0) := "011110" & protocol_specific(9
downto 8);
                     new_cmd_batch.sca_cmds_data(j).data(15 downto 8) := protocol_specific(7 downto 0);
                     new_cmd_batch.sca_cmds_data(j).len := std_logic_vector(to_unsigned(2, 8));
                  end if;
               else

                  new_cmd_batch.cmd_count := 2;
                  --Single Byte Operation
                  if ecs_i2c_cmd = ECS_I2C_WRITE then
```

```
                    new_cmd_batch.sca_cmds(j) := i2c_cmd_table(I2C_S_7B_W);
                    --7bit addr
                    new_cmd_batch.sca_cmds_data(j).data(7 downto 0) := '0' &
ecs_cmd_packet.protocol_specific(6 downto 0);
                    --1 byte data
                    new_cmd_batch.sca_cmds_data(j).data(15 downto 8) := ecs_cmd_packet.data(7 downto
0);
                    new_cmd_batch.sca_cmds_data(j).data(31 downto 16) := (others=>'0');
                    new_cmd_batch.sca_cmds_data(j).len := std_logic_vector(to_unsigned( 2 ,8));

                 else
                    new_cmd_batch.sca_cmds(j) := i2c_cmd_table(I2C_S_10B_W);

                    --10 bit addr
                    new_cmd_batch.sca_cmds_data(j).data(7 downto 0) := "011110" &
ecs_cmd_packet.protocol_specific(9 downto 8);
                    new_cmd_batch.sca_cmds_data(j).data(15 downto 8) :=
ecs_cmd_packet.protocol_specific(7 downto 0);
                    -- 1 byte data
                    new_cmd_batch.sca_cmds_data(j).data(23 downto 16) := ecs_cmd_packet.data(7 downto
0);
                    new_cmd_batch.sca_cmds_data(1).data(31 downto 24) := (others=>'0');
                    new_cmd_batch.sca_cmds_data(j).len := std_logic_vector(to_unsigned( 3 ,8));

                 end if;

              end if;
           elsif ecs_i2c_cmd = ECS_I2C_READ or ecs_i2c_cmd = ECS_I2C_READ_EXT then

              if NByte>1 then --MULTI BYTE READ

                 --new_cmd_batch.cmd_count := ((NByte+1)/4) + 2;
                 -- 1 I2c ctrl write + 1 I2c multibyte read + 1 up to 4 i2c data buffers read
                 new_cmd_batch.cmd_count := 1 + 1  + (((NByte-1)/4) +1 );

                 --On multi byte reads first you have to issue the I2C Read Command
                 -- then Read each of the data registers associated
                 if ecs_i2c_cmd = ECS_I2C_READ then
                    new_cmd_batch.sca_cmds(1) := i2c_cmd_table(I2C_M_7B_R);
                    new_cmd_batch.sca_cmds_data(1).data(7 downto 0) := "0" &
ecs_cmd_packet.protocol_specific(6 downto 0);
                    new_cmd_batch.sca_cmds_data(1).data(31 downto 8) := (others=>'0');
                    new_cmd_batch.sca_cmds_data(1).len := std_logic_vector(to_unsigned( 1 ,8));
                 else
                    new_cmd_batch.sca_cmds(1) := i2c_cmd_table(I2C_M_10B_R);
                    --10 bit addr
                    new_cmd_batch.sca_cmds_data(j).data(7 downto 0) := "011110" &
ecs_cmd_packet.protocol_specific(9 downto 8);
                    new_cmd_batch.sca_cmds_data(j).data(15 downto 8) :=
ecs_cmd_packet.protocol_specific(7 downto 0);

                    new_cmd_batch.sca_cmds_data(1).data(31 downto 16) := (others=>'0');
                    new_cmd_batch.sca_cmds_data(1).len := std_logic_vector(to_unsigned( 2 ,8));
                 end if;

                 --Read each of the required data registers needed
                 --i = total byte counter, j = 32 bit data buffer counter, k = in word byte counter
                 j:= 1;
                 k:=1;
                 for i in 1 to ecs_cmd_packet.data'length/8 loop
```

```
                    if i > NByte  then
                       exit;
                    end if;
                    if i > j*4 then
                       j:=j+1;
                       k:=1;
                    end if;
                    new_cmd_batch.sca_cmds_data(j+1).data := (others=>'0');
                    new_cmd_batch.sca_cmds_data(j+1).len := std_logic_vector(to_unsigned(0, 8));
                    k:=k+1;
                    case (j-1) is
                    when 0 =>
                       new_cmd_batch.sca_cmds(j+1) := i2c_cmd_table(I2C_R_DATA0);
                    when 1=>
                       new_cmd_batch.sca_cmds(j+1) := i2c_cmd_table(I2C_R_DATA1);
                    when 2=>
                       new_cmd_batch.sca_cmds(j+1) := i2c_cmd_table(I2C_R_DATA2);
                    when others=> --3
                       new_cmd_batch.sca_cmds(j+1) := i2c_cmd_table(I2C_R_DATA3);
                    end case;
                 end loop;
                 --
              else
                 new_cmd_batch.cmd_count := 2;

                 j:= 1;
                 if ecs_i2c_cmd = ECS_I2C_READ then
                    new_cmd_batch.sca_cmds(j) := i2c_cmd_table(I2C_S_7B_R);
                    new_cmd_batch.sca_cmds_data(j).data(31 downto 8) := (others=>'0');
                    --protocol_specific(6 downto 0) = 7 bits I2c address
                    new_cmd_batch.sca_cmds_data(j).data(7 downto 0) :=  "0" & protocol_specific(6
downto 0);
                    new_cmd_batch.sca_cmds_data(j).len := std_logic_vector(to_unsigned(1, 8));
                 else
                    new_cmd_batch.sca_cmds(j) := i2c_cmd_table(I2C_S_10B_R);
                    new_cmd_batch.sca_cmds_data(j).data(31 downto 16) := (others=>'0');
                    --protocol_specific(9 downto 0) = 10 bits I2c address
                    new_cmd_batch.sca_cmds_data(j).data(7 downto 0) := "011110" &
ecs_cmd_packet.protocol_specific(9 downto 8);
                    new_cmd_batch.sca_cmds_data(j).data(15 downto 8) :=
ecs_cmd_packet.protocol_specific(7 downto 0);
                    new_cmd_batch.sca_cmds_data(j).len := std_logic_vector(to_unsigned(2, 8));
                 end if;
              end if;
           end if;

        -- Finish by writing the Proper I2C command

        when ECS_I2C_W_R_DATA0 =>
           new_cmd_batch.cmd_count := 2;
           new_cmd_batch.sca_cmds(0) := i2c_cmd_table(I2C_W_DATA0);
           new_cmd_batch.sca_cmds_data(0).data := ecs_cmd_packet.data(31 downto 0);
           new_cmd_batch.sca_cmds_data(0).len := std_logic_vector(to_unsigned(4, 8));
           new_cmd_batch.sca_cmds(1) := i2c_cmd_table(I2C_R_DATA0);
           new_cmd_batch.sca_cmds_data(1).data := (others=>'0');
           new_cmd_batch.sca_cmds_data(1).len := (others=>'0');
        when ECS_I2C_W_R_DATA1 =>
           new_cmd_batch.cmd_count := 2;
           new_cmd_batch.sca_cmds(0) := i2c_cmd_table(I2C_W_DATA1);
           new_cmd_batch.sca_cmds_data(0).data := ecs_cmd_packet.data(31 downto 0);
```

```
            new_cmd_batch.sca_cmds_data(0).len := std_logic_vector(to_unsigned(4, 8));
            new_cmd_batch.sca_cmds(1) := i2c_cmd_table(I2C_R_DATA1);
            new_cmd_batch.sca_cmds_data(1).data := (others=>'0');
            new_cmd_batch.sca_cmds_data(1).len := (others=>'0');
         when ECS_I2C_W_R_DATA2 =>
            new_cmd_batch.cmd_count := 2;
            new_cmd_batch.sca_cmds(0) := i2c_cmd_table(I2C_W_DATA2);
            new_cmd_batch.sca_cmds_data(0).data := ecs_cmd_packet.data(31 downto 0);
            new_cmd_batch.sca_cmds_data(0).len := std_logic_vector(to_unsigned(4, 8));
            new_cmd_batch.sca_cmds(1) := i2c_cmd_table(I2C_R_DATA2);
            new_cmd_batch.sca_cmds_data(1).data := (others=>'0');
            new_cmd_batch.sca_cmds_data(1).len := (others=>'0');
         when ECS_I2C_W_R_DATA3 =>
            new_cmd_batch.cmd_count := 2;
            new_cmd_batch.sca_cmds(0) := i2c_cmd_table(I2C_W_DATA3);
            new_cmd_batch.sca_cmds_data(0).data := ecs_cmd_packet.data(31 downto 0);
            new_cmd_batch.sca_cmds_data(0).len := std_logic_vector(to_unsigned(4, 8));
            new_cmd_batch.sca_cmds(1) := i2c_cmd_table(I2C_R_DATA3);
            new_cmd_batch.sca_cmds_data(1).data := (others=>'0');
            new_cmd_batch.sca_cmds_data(1).len := (others=>'0');
         when ECS_I2C_W_R_CTRL =>
            new_cmd_batch.cmd_count := 2;
            new_cmd_batch.sca_cmds(0) := i2c_cmd_table(I2C_W_CTRL);
            new_cmd_batch.sca_cmds_data(0).data(7 downto 0) :=
               ecs_cmd_packet.data(7 downto 0) ;
            new_cmd_batch.sca_cmds_data(0).data(31 downto 8) := (others=>'0');
            new_cmd_batch.sca_cmds_data(0).len := std_logic_vector(to_unsigned(2, 8));
            new_cmd_batch.sca_cmds(1) := i2c_cmd_table(I2C_R_CTRL);
            new_cmd_batch.sca_cmds_data(1).data := (others=>'0');
            new_cmd_batch.sca_cmds_data(1).len := std_logic_vector(to_unsigned(2, 8));
         when ECS_I2C_W_R_MSK =>
            new_cmd_batch.cmd_count := 2;
            new_cmd_batch.sca_cmds(0) := i2c_cmd_table(I2C_W_MSK);
            new_cmd_batch.sca_cmds_data(0).data(7 downto 0) :=
               ecs_cmd_packet.data(7 downto 0) ;
            new_cmd_batch.sca_cmds_data(0).data(31 downto 8) := (others=>'0');
            new_cmd_batch.sca_cmds_data(0).len := std_logic_vector(to_unsigned(2, 8));
            new_cmd_batch.sca_cmds(1) := i2c_cmd_table(I2C_R_MSK);
            new_cmd_batch.sca_cmds_data(1).data := (others=>'0');
            new_cmd_batch.sca_cmds_data(1).len := std_logic_vector(to_unsigned(2, 8));
         when INVALID =>
            new_cmd_batch.err :=x"01";
         new_cmd_batch.ecs_cmd := x"FF";
         new_cmd_batch.cmd_count := 0;
      end case;
   end if;
   sca_cmd_batch_o <= new_cmd_batch;
end process i2c_protocol_builder;

i2c_protocol_decoder : process ( sca_rpy_batch_i, prot_rpy_en_i) is
   variable new_ecs_rpy_packet      :  ecs_packet_t;
   variable ecs_i2c_cmd       :  ecs_i2c_cmd_enum;

   variable j                 :  natural range 0 to 6;
   variable k                 :  natural range 1 to 16;
   variable NByte          :  natural range 1 to 16;
begin

   new_ecs_rpy_packet.err := (others=>'0');
   --new_ecs_rpy_packet.gbt_nr := (others=>'0');
```

```
    new_ecs_rpy_packet.sca_nr  := (others=>'0');
    new_ecs_rpy_packet.sca_ch  := (others=>'0');
    new_ecs_rpy_packet.ecs_cmd := (others=>'0');
    new_ecs_rpy_packet.data := (others=>'0');
    new_ecs_rpy_packet.len := (others=>'0');
    new_ecs_rpy_packet.protocol_specific  := (others=>'0');


    ecs_i2c_cmd := INVALID;

    if prot_rpy_en_i ='1' then
       for i in ecs_i2c_table_t'left to ecs_i2c_table_t'right loop
          if sca_rpy_batch_i.ecs_cmd = ecs_i2c_table(i) then
             ecs_i2c_cmd := i;
             exit;
          end if;
       end loop;

       new_ecs_rpy_packet.sca_ch := sca_rpy_batch_i.ch;
       --new_ecs_rpy_packet.gbt_nr := std_logic_vector( to_unsigned( sca_rpy_batch_i.gbt, 8));
       new_ecs_rpy_packet.sca_nr := std_logic_vector( to_unsigned(sca_rpy_batch_i.sca,8));

       new_ecs_rpy_packet.err := (others=>'0');
       if sca_rpy_batch_i.err /= (sca_rpy_batch_i.err'range =>'0') then
          for j in 0 to sca_rpy_batch_i.err'high -1 loop
          if sca_rpy_batch_i.err(j)/='0' then
                -- \TODO What to do?
                report "There was an error on the GBT-SCA channel operation" & integer'image(j) severity
Note;

                exit;
             end if;
          end loop;
          new_ecs_rpy_packet.err := sca_rpy_batch_i.err;
       end if;

       new_ecs_rpy_packet.ecs_cmd := ecs_i2c_table(ecs_i2c_cmd);
       if to_integer(unsigned(sca_rpy_batch_i.ecs_len)) >=16 then
          NByte := 16;
          new_ecs_rpy_packet.len :=  std_logic_vector(to_unsigned(16,8));
       else
          NByte := to_integer(unsigned(sca_rpy_batch_i.ecs_len));
          new_ecs_rpy_packet.len :=  sca_rpy_batch_i.ecs_len;
       end if;
       new_ecs_rpy_packet.protocol_specific(15 downto 0) := (others=>'0');


       case ecs_i2c_cmd is
       when ECS_I2C_WRITE | ECS_I2C_WRITE_EXT | ECS_I2C_READ | ECS_I2C_READ_EXT =>
          if ecs_i2c_cmd = ECS_I2C_WRITE or ecs_i2c_cmd = ECS_I2C_WRITE_EXT then
             new_ecs_rpy_packet.data:= (others=>'0');
             if NByte>1  then
                j := 1 + (((NByte-1)/4) +1 );
                new_ecs_rpy_packet.protocol_specific(15 downto 12) :=
                   sca_rpy_batch_i.sca_rpys_data(j).data(6) & --NOACK
                   sca_rpy_batch_i.sca_rpys_data(j).data(5) & --INVCOM
                   sca_rpy_batch_i.sca_rpys_data(j).data(3) & --LEVERR
                   sca_rpy_batch_i.sca_rpys_data(j).data(2);  --SUCC

             else
```

```
            --PROTOCOL SPECIFIC{15=NOACK,14=INVCOM,13=LEVERR,12=SUCC}
            new_ecs_rpy_packet.protocol_specific(15 downto 12) :=
               sca_rpy_batch_i.sca_rpys_data(1).data(6) & --NOACK
               sca_rpy_batch_i.sca_rpys_data(1).data(5) & --INVCOM
               sca_rpy_batch_i.sca_rpys_data(1).data(3) & --LEVERR
               sca_rpy_batch_i.sca_rpys_data(1).data(2);  --SUCC
               --new_ecs_rpy_packet.protocol_specific(11 downto 0):=
         end if;
      elsif ecs_i2c_cmd = ECS_I2C_READ_EXT or ecs_i2c_cmd = ECS_I2C_READ then

         --PROTOCOL SPECIFIC{15=NOACK,14=INVCOM,13=LEVERR,12=SUCC}
            new_ecs_rpy_packet.protocol_specific(15 downto 12) :=
               sca_rpy_batch_i.sca_rpys_data(1).data(6) & --NOACK
               sca_rpy_batch_i.sca_rpys_data(1).data(5) & --INVCOM
               sca_rpy_batch_i.sca_rpys_data(1).data(3) & --LEVERR
               sca_rpy_batch_i.sca_rpys_data(1).data(2);  --SUCC
         -- i = total byte counter, j = 32 bit data buffer counter, k = in data buffer byte counter
         if NByte>1 then --MULTI BYTE Read
            j := 1;
            k := 1;

            for i in 1 to ecs_cmd_packet.data'length/8 loop
               if i > NByte or i > 16 then
                  exit;
               end if;
               if i > j*4 then
                  j:=j+1;
                  k:=1;
               end if;
               --! j = {1,2,3,4} , data buffers starts at reply packet index 2, so we use j+1 =
{2,3,4,5}
               new_ecs_rpy_packet.data(i*8-1 downto (i-1)*8)  :=
sca_rpy_batch_i.sca_rpys_data(j+1).data(k*8-1 downto (k-1)*8);
               k:=k+1;
            end loop;
            j := j + 1;
         else
            new_ecs_rpy_packet.data(7 downto 0) :=  sca_rpy_batch_i.sca_rpys_data(1).data(15 downto
8);
         end if;
      end if;
   when ECS_I2C_RMW_AND | ECS_I2C_RMW_OR | ECS_I2C_RMW_XOR =>
      new_ecs_rpy_packet.data := (others=>'0');
      new_ecs_rpy_packet.len := std_logic_vector(to_unsigned(2, 8));
   when ECS_I2C_W_CTRL | ECS_I2C_W_MSK =>
      new_ecs_rpy_packet.data := (others=>'0');
      new_ecs_rpy_packet.len := std_logic_vector(to_unsigned(2, 8));
   when ECS_I2C_R_CTRL =>
      new_ecs_rpy_packet.data(7 downto 0) := sca_rpy_batch_i.sca_rpys_data(0).data(7 downto 0);
      new_ecs_rpy_packet.len := std_logic_vector(to_unsigned(1, 8));
   when ECS_I2C_R_MSK =>
      new_ecs_rpy_packet.data(7 downto 0) := sca_rpy_batch_i.sca_rpys_data(0).data(7 downto 0);
      new_ecs_rpy_packet.len := std_logic_vector(to_unsigned(1, 8));
   when ECS_I2C_R_STR  =>
      new_ecs_rpy_packet.data(7 downto 0) := sca_rpy_batch_i.sca_rpys_data(0).data(7 downto 0);
      new_ecs_rpy_packet.len := std_logic_vector(to_unsigned(1, 8));
   when ECS_I2C_S_7B_W | ECS_I2C_S_10B_W | ECS_I2C_M_7B_W | ECS_I2C_M_7B_R
      | ECS_I2C_M_10B_W | ECS_I2C_M_10B_R =>
         new_ecs_rpy_packet.data(7 downto 0) := sca_rpy_batch_i.sca_rpys_data(0).data(7 downto 0);
         new_ecs_rpy_packet.len := std_logic_vector(to_unsigned(2, 8));
```

```
        when ECS_I2C_S_7B_R | ECS_I2C_S_10B_R   =>
           new_ecs_rpy_packet.data(15 downto 0) := sca_rpy_batch_i.sca_rpys_data(0).data(15 downto 0);
           new_ecs_rpy_packet.len := std_logic_vector(to_unsigned(2, 8));
        when ECS_I2C_W_DATA0 | ECS_I2C_W_DATA1 | ECS_I2C_W_DATA2 | ECS_I2C_W_DATA3 =>
           new_ecs_rpy_packet.data := (others=>'0');
           new_ecs_rpy_packet.len := std_logic_vector(to_unsigned(2, 8));
        when ECS_I2C_R_DATA0 | ECS_I2C_R_DATA1 | ECS_I2C_R_DATA2 | ECS_I2C_R_DATA3 =>
           new_ecs_rpy_packet.data(31 downto 0) := sca_rpy_batch_i.sca_rpys_data(0).data;
           new_ecs_rpy_packet.len := std_logic_vector(to_unsigned(4, 8));
        when ECS_I2C_W_R_DATA0 | ECS_I2C_W_R_DATA1 | ECS_I2C_W_R_DATA2 | ECS_I2C_W_R_DATA3  =>
           new_ecs_rpy_packet.data(31 downto 0) := sca_rpy_batch_i.sca_rpys_data(1).data;
           new_ecs_rpy_packet.len := std_logic_vector(to_unsigned(4, 8));
        when ECS_I2C_W_R_CTRL | ECS_I2C_W_R_MSK =>
           new_ecs_rpy_packet.data(7 downto 0) := sca_rpy_batch_i.sca_rpys_data(1).data(7 downto 0);
           new_ecs_rpy_packet.len := std_logic_vector(to_unsigned(2, 8));
        when others=>
           new_ecs_rpy_packet.ecs_cmd := x"FF";
           new_ecs_rpy_packet.len := std_logic_vector(to_unsigned(0, 8));
        end case;
     end if;
     ecs_rpy_packet <= new_ecs_rpy_packet;
  end process i2c_protocol_decoder;

end architecture RTL;
```

```
--------------------------------------------------------------------------------
--! @file sca_controller_protocol_driver.vhd
--! @author Cairo Caplan <cairo.caplan@cern.ch>
--! @date April, 2016
--------------------------------------------------------------------------------

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

use work.SCA_Package.all;

entity sca_controller_protocol_driver is
   port (
      clk : in std_logic;
      rst : in std_logic;
      -- Interface with the ECS CMD Protocol Arbiter
      prot_cmd_en_i     :  in std_logic;
      ecs_cmd_packet    :  in ecs_packet_t;

      -- Interface with the ECS RPY Protocol Arbiter
      prot_rpy_en_i     :  in std_logic;
      ecs_rpy_packet    :  out   ecs_packet_t;

      -- Interface with the Command Queue manager
      sca_cmd_batch_o      :  out prot_cmd_t;
      sca_rpy_batch_i      :  in prot_cmd_t;
      --Others
      tr_cmd_i       :  in byte_t;
      --Internal Protocol Layer register regarding which SCA and which of its channels are enabled
      activated_channels_o:   out activated_channels_t
   );
end entity sca_controller_protocol_driver;

architecture RTL of sca_controller_protocol_driver is

   --! Register with the list of the activated channels, including the SCA itself. It updates with the
   --! CH_EN register on the SCA, plus the SCA enable as bit 0.
   signal activated_channels  :  activated_channels_t := (others=>(others=>'1'));

   signal err_rply            :  std_logic;
begin
   -- \TODO activated_channels register is written by two different process, this generates a race
     condition
   sca_controller_protocol_builder : process(ecs_cmd_packet,  prot_cmd_en_i, tr_cmd_i) is

      variable ecs_sca_cmd :  ecs_sca_cmd_enum;
      variable new_cmd_batch  :  prot_cmd_t;


   begin

      new_cmd_batch.tr := x"00";
      new_cmd_batch.ch := x"FF";
      new_cmd_batch.cmd_count := 0;
      new_cmd_batch.sca := 0;

      new_cmd_batch.ecs_cmd := (others=>'0');
      new_cmd_batch.ecs_len := (others=>'0');
```

```
new_cmd_batch.err := (others=>'0');

new_cmd_batch.sca_cmds := (others=>(others=>'0'));
new_cmd_batch.sca_cmds_data :=  (others=>(others=>(others=>'0')));
new_cmd_batch.sca_rpys_data :=  (others=>(others=>(others=>'0')));

ecs_sca_cmd := INVALID;

if prot_cmd_en_i = '1' then
   --\TODO Report if command is not valid
   for i in ecs_sca_cmd_table_t'left to ecs_sca_cmd_table_t'right loop
      if ecs_cmd_packet.ecs_cmd = ecs_sca_cmd_table(i) then
         ecs_sca_cmd := i;
         exit;
      end if;
   end loop;


   --protocol_data := ecs_cmd_packet_int.data(ecs_cmd_packet_int.data'high - 16 downto 0);
   new_cmd_batch.tr := tr_cmd_i;
   new_cmd_batch.ch := ecs_cmd_packet.sca_ch;

   --new_cmd_batch.gbt := to_integer(unsigned(ecs_cmd_packet.gbt_nr));
   new_cmd_batch.sca := to_integer(unsigned(ecs_cmd_packet.sca_nr));

   new_cmd_batch.ecs_cmd := ecs_cmd_packet.ecs_cmd;

   new_cmd_batch.cmd_count := 1;
   new_cmd_batch.ecs_len := x"01";

   case ecs_sca_cmd is
      when ECS_CRA_wrt | ECS_CRB_wrt | ECS_CRC_wrt | ECS_CRD_wrt  =>
         -- new_cmd_batch.sca_cmds_data(0).data (31 downto 31 - 7) :=
         -- new_cmd_batch.sca_cmds_data(0).data (31 - 8  downto 0) := (others=>'0');
         -- 06/05/2015:
         if ecs_sca_cmd = ECS_CRB_wrt then
            new_cmd_batch.sca_cmds_data(0).data(7 downto 0) :=
               ecs_cmd_packet.data(7 downto 1)& "0";
         elsif ecs_sca_cmd = ECS_CRD_wrt then
            new_cmd_batch.sca_cmds_data(0).data(7 downto 0) :=
               "00" & ecs_cmd_packet.data(5 downto 0);
         else
            new_cmd_batch.sca_cmds_data(0).data(7 downto 0) :=
               ecs_cmd_packet.data(7 downto 0);
         end if;
         new_cmd_batch.sca_cmds_data(0).data(31 downto 8) := (others=>'0');
         new_cmd_batch.sca_cmds_data(0).len := std_logic_vector(to_unsigned(1,8));
         if ecs_sca_cmd = ECS_CRA_wrt then
            --activated_channels(sca_num)(7 downto 1) <= ecs_cmd_packet.data(7 downto 1);
            new_cmd_batch.sca_cmds(0) := sca_cmd_table(CRA_wrt);
         elsif ecs_sca_cmd = ECS_CRB_wrt then
            --activated_channels(sca_num)(7 downto 1) <= ecs_cmd_packet.data(7 downto 1);
            new_cmd_batch.sca_cmds(0) := sca_cmd_table(CRB_wrt);
         elsif ecs_sca_cmd = ECS_CRC_wrt then
            --activated_channels(sca_num)(15 downto 8) <= ecs_cmd_packet.data(7 downto 0);
            new_cmd_batch.sca_cmds(0) := sca_cmd_table(CRC_wrt);
         elsif ecs_sca_cmd = ECS_CRD_wrt then
            new_cmd_batch.sca_cmds(0) := sca_cmd_table(CRD_wrt);
            --activated_channels(sca_num)(21 downto 16) <= ecs_cmd_packet.data(5 downto 0);
         end if;
```

```
         when ECS_CRA_rea | ECS_CRB_rea | ECS_CRC_rea | ECS_CRD_rea =>
            new_cmd_batch.sca_cmds_data(0).data (31 downto 0) := (others=>'0');
            new_cmd_batch.sca_cmds_data(0).len := std_logic_vector(to_unsigned(0,8));
            if ecs_sca_cmd = ECS_CRA_rea then
               --activated_channels(sca_num)(7 downto 1) <= ecs_cmd_packet.data(7 downto 1);
               new_cmd_batch.sca_cmds(0) := sca_cmd_table(CRA_rea);
            elsif ecs_sca_cmd = ECS_CRB_rea then
               --activated_channels(sca_num)(15 downto 8) <= ecs_cmd_packet.data(7 downto 0);
               --new_cmd_batch.sca_cmds_data(0).len := std_logic_vector(to_unsigned(1,8));
               new_cmd_batch.sca_cmds(0) := sca_cmd_table(CRB_rea);
            elsif ecs_sca_cmd = ECS_CRC_rea then
               new_cmd_batch.sca_cmds(0) := sca_cmd_table(CRC_rea);
               --activated_channels(sca_num)(21 downto 16) <= ecs_cmd_packet.data(5 downto 0);
            elsif ecs_sca_cmd = ECS_CRD_rea then
               new_cmd_batch.sca_cmds(0) := sca_cmd_table(CRD_rea);
            end if;
         when others=>
            new_cmd_batch.err :=x"01";
            new_cmd_batch.sca_cmds(0) := x"FF";
            new_cmd_batch.cmd_count := 0;
      end case;
   else

      new_cmd_batch.sca :=0;
      new_cmd_batch.ch := (others=>'X');
      new_cmd_batch.cmd_count := 0;
      new_cmd_batch.err := (others=>'X');
      new_cmd_batch.tr := (others=>'X');
      for i in 0 to new_cmd_batch.sca_cmds'high loop
         new_cmd_batch.sca_cmds(i) := (others=>'X');
         new_cmd_batch.sca_cmds_data(i).data := (others=>'X');
         new_cmd_batch.sca_cmds_data(i).len := (others=>'X');
         new_cmd_batch.sca_rpys_data(i).data := (others=>'X');
         new_cmd_batch.sca_rpys_data(i).len := (others=>'X');
      end loop;
   end if;



   sca_cmd_batch_o <=   new_cmd_batch;
end process sca_controller_protocol_builder;



-------------------------------------------------

sca_controller_protocol_decoder : process ( sca_rpy_batch_i, prot_rpy_en_i) is
   variable ecs_sca_cmd       :  ecs_sca_cmd_enum;
   variable new_ecs_rpy_packet      :  ecs_packet_t;

begin
   --new_ecs_rpy_packet.gbt_nr := (others=>'0');
   new_ecs_rpy_packet.sca_nr  := (others=>'0');
   new_ecs_rpy_packet.sca_ch  := (others=>'0');
   new_ecs_rpy_packet.ecs_cmd := (others=>'0');
   new_ecs_rpy_packet.data := (others=>'0');
   new_ecs_rpy_packet.len := (others=>'0');
   new_ecs_rpy_packet.protocol_specific  := (others=>'0');
```

```
--    if prot_rpy_en_i='1' then
         ecs_sca_cmd := INVALID;
         for i in ecs_sca_cmd_table_t'left to ecs_sca_cmd_table_t'right loop
            if sca_rpy_batch_i.ecs_cmd = ecs_sca_cmd_table(i) then
               ecs_sca_cmd := i;
               exit;
            end if;
         end loop;


         new_ecs_rpy_packet.sca_ch := sca_rpy_batch_i.ch;

         --new_ecs_rpy_packet.gbt_nr := std_logic_vector( to_unsigned(sca_rpy_batch_i.gbt, 8));
         new_ecs_rpy_packet.sca_nr := std_logic_vector( to_unsigned(sca_rpy_batch_i.sca,8));
         new_ecs_rpy_packet.err := (others=>'0');

         -- If one of the commands of the batched returned (on their replies) an error, put this first
     error code as the
         -- ECS Reply's command field
         if sca_rpy_batch_i.err /= (sca_rpy_batch_i.err'range =>'0') and prot_rpy_en_i='1' then
            for j in 0 to sca_rpy_batch_i.err'high -1 loop
            if sca_rpy_batch_i.err(j)/='0' then
                  -- \TODO What to do?
                  report "There was an error on the GBT-SCA channel operation" & integer'image(j) severity
     Note;

                  exit;
               end if;
            end loop;
            new_ecs_rpy_packet.err := sca_rpy_batch_i.err;
         end if;
            new_ecs_rpy_packet.ecs_cmd := ecs_sca_cmd_table(ecs_sca_cmd);
         case ecs_sca_cmd is
            when ECS_CRA_wrt | ECS_CRB_wrt | ECS_CRC_wrt | ECS_CRD_wrt  =>
               new_ecs_rpy_packet.len :=  (others=>'0');

            when ECS_CRA_rea | ECS_CRB_rea | ECS_CRC_rea | ECS_CRD_rea =>
               new_ecs_rpy_packet.data(31 downto 8) := (others=>'0');
               new_ecs_rpy_packet.data(7 downto 0) := sca_rpy_batch_i.sca_rpys_data(0).data(7 downto 0);

               new_ecs_rpy_packet.protocol_specific(15 downto 0) := (others=>'0');
               --new_ecs_rpy_packet.len := std_logic_vector(to_unsigned(1,16));
               new_ecs_rpy_packet.len := std_logic_vector(to_unsigned(1,8));
            when others =>
               --new_ecs_rpy_packet.ecs_cmd := x"FF";
               null;
         end case;
--    else
--       new_ecs_rpy_packet.sca_nr  := 0;
--       new_ecs_rpy_packet.sca_ch  := (others=>'X');
--       new_ecs_rpy_packet.ecs_cmd := (others=>'X');
--       new_ecs_rpy_packet.data := (others=>'X');
--       new_ecs_rpy_packet.len := (others=>'X');
--       new_ecs_rpy_packet.protocol_specific  := (others=>'X');
--
--    end if;
--


      if new_ecs_rpy_packet.ecs_cmd /= (new_ecs_rpy_packet.ecs_cmd'range => '0') then
         err_rply <= '1';
```

```
      else
         err_rply <= '0';
      end if;
      ecs_rpy_packet <= new_ecs_rpy_packet;
   end process sca_controller_protocol_decoder;


   activated_channels_reg_manager : process (clk, rst) is
      variable ecs_sca_cmd       :  ecs_sca_cmd_enum;
   begin
      if rst = '1' then
         for i in 0 to SCA_COUNT -1 loop
            activated_channels(i) <= (others=>'1');
         end loop;
      elsif rising_edge(clk) then
         if prot_rpy_en_i='1' and err_rply = '0' then
            ecs_sca_cmd := INVALID;
            for i in ecs_sca_cmd_table_t'left to ecs_sca_cmd_table_t'right loop
               if sca_rpy_batch_i.ecs_cmd = ecs_sca_cmd_table(i) then
                  ecs_sca_cmd := i;
                  exit;
               end if;
            end loop;
            -- \TODO see if this line is not required anymore without inferring a Latch
            activated_channels(sca_rpy_batch_i.sca)(0) <= '1';
            if ecs_sca_cmd = ECS_CRB_wrt then
               activated_channels(sca_rpy_batch_i.sca)(7 downto 1) <=
     sca_rpy_batch_i.sca_cmds_data(0).data(7 downto 1);
            elsif ecs_sca_cmd = ECS_CRC_wrt then
               activated_channels(sca_rpy_batch_i.sca)(15 downto 8) <=
     sca_rpy_batch_i.sca_cmds_data(0).data(7 downto 0);
            elsif ecs_sca_cmd = ECS_CRD_wrt then
               activated_channels(sca_rpy_batch_i.sca)(21 downto 16)
     <=sca_rpy_batch_i.sca_cmds_data(0).data(5 downto 0);
            elsif ecs_sca_cmd = ECS_CRB_rea then
               activated_channels(sca_rpy_batch_i.sca)(7 downto 1) <=
     sca_rpy_batch_i.sca_rpys_data(0).data(7 downto 1);
            elsif ecs_sca_cmd = ECS_CRC_rea then
               activated_channels(sca_rpy_batch_i.sca)(15 downto 8) <=
     sca_rpy_batch_i.sca_rpys_data(0).data(7 downto 0);
            elsif ecs_sca_cmd = ECS_CRD_rea then
               activated_channels(sca_rpy_batch_i.sca)(21 downto 16)
     <=sca_rpy_batch_i.sca_rpys_data(0).data(5 downto 0);
            else
               null;
            end if;
         else
            activated_channels <= activated_channels;
         end if;
      end if;
   end process activated_channels_reg_manager;


   activated_channels_o <= activated_channels;



end architecture RTL;
```

```
--------------------------------------------------------------------------------
--! @file spii_protocol_driver.vhd
--! @author Cairo Caplan <cairo.caplan@cern.ch>
--! @date April, 2016
--------------------------------------------------------------------------------

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

use work.SCA_Package.all;


entity spi_protocol_driver is
   port (
      -- Interface with the ECS CMD Protocol Arbiter
      prot_cmd_en_i     :  in std_logic;
      ecs_cmd_packet    :  in ecs_packet_t;
      -- Interface with the ECS RPY Protocol Arbiter
      prot_rpy_en_i     :  in std_logic;
      ecs_rpy_packet    :  out   ecs_packet_t;
      -- Interface with the Command Queue manager
      sca_cmd_batch_o      :  out prot_cmd_t;
      sca_rpy_batch_i      :  in prot_cmd_t;
      --Others
      tr_cmd_i       :  in byte_t
   );
end entity spi_protocol_driver;

architecture RTL of spi_protocol_driver is

--Protocol Specific SPI - {'0',INVSCLK,'0',IE = 1b, LSB=1b,TX_NEG= 1b,RX_NEG=1b,GO = 1b,   '0',Nbits = 7b}
--Control Register of the SPI --{'0',INVSCLK,'0',IE,LSB,TXNEG,RXNEG,GO/BUSY,"0",CHARLEN = 6b}

begin

--CHARLEN The value contained in the CharLen field represents how many bits will be transmitted during
-- the following transmission, from 1 to 128 (value 0 -> 128bit transmission)

   spi_protocol_builder : process (
      prot_cmd_en_i, tr_cmd_i, ecs_cmd_packet
   ) is
      variable ecs_spi_cmd       :  ecs_spi_cmd_enum;
      variable new_cmd_batch     :  prot_cmd_t;
      variable Nbits          :  natural range 1 to 128;
      variable CHARLEN        :  natural range 0 to 127;
      variable k              :  natural range 0 to 4;
   begin

      new_cmd_batch.tr := x"00";
      new_cmd_batch.ch := x"FF";
      new_cmd_batch.cmd_count := 0;
      new_cmd_batch.sca := 0;

      new_cmd_batch.ecs_cmd := (others=>'0');
      new_cmd_batch.ecs_len := (others=>'0');
      new_cmd_batch.err := (others=>'0');

      new_cmd_batch.sca_cmds := (others=>(others=>'0'));
```

```
new_cmd_batch.sca_cmds_data :=  (others=>(others=>(others=>'0')));
new_cmd_batch.sca_rpys_data :=  (others=>(others=>(others=>'0')));

ecs_spi_cmd := INVALID;


if prot_cmd_en_i='1' then
   for i in ecs_spi_cmd_table_t'left to ecs_spi_cmd_table_t'right loop
      if ecs_cmd_packet.ecs_cmd = ecs_spi_cmd_table(i) then
         ecs_spi_cmd := i;
         exit;
      end if;
   end loop;
   --protocol_data := ecs_cmd_packet_int.data(ecs_cmd_packet_int.data'high - 16 downto 0);

   new_cmd_batch.tr := tr_cmd_i;
   new_cmd_batch.ch := ecs_cmd_packet.sca_ch;
   new_cmd_batch.cmd_count := 0;
   --new_cmd_batch.gbt := to_integer(unsigned(ecs_cmd_packet.gbt_nr));
   new_cmd_batch.sca := to_integer(unsigned(ecs_cmd_packet.sca_nr));

   new_cmd_batch.ecs_cmd := ecs_cmd_packet.ecs_cmd;

   for i in 0 to new_cmd_batch.sca_cmds'high loop
      new_cmd_batch.sca_cmds(i) := (others=>'0');
      new_cmd_batch.sca_cmds_data(i).data := (others=>'0');
      new_cmd_batch.sca_cmds_data(i).len := (others=>'0');
      new_cmd_batch.sca_rpys_data(i).data := (others=>'0');
      new_cmd_batch.sca_rpys_data(i).len := (others=>'0');
   end loop;


   new_cmd_batch.ecs_len := ecs_cmd_packet.len;

   case ecs_spi_cmd is
      when ECS_SPI_GO =>
         --
         new_cmd_batch.sca_cmds(0) := spi_cmd_table(SPI_GO);
         new_cmd_batch.sca_cmds_data(0).data := (others=>'0');
         new_cmd_batch.sca_cmds_data(0).len := (others=>'0');
         new_cmd_batch.cmd_count := 1;
      when ECS_SPI_W_CTRL =>
         --
         new_cmd_batch.sca_cmds(0) := spi_cmd_table(SPI_wrt_CTRL);
         --
         new_cmd_batch.sca_cmds_data(0).data(31 downto 21) :=
            ecs_cmd_packet.data(31 downto 21) ;
         new_cmd_batch.sca_cmds_data(0).data(19 downto 16) :=
            ecs_cmd_packet.data(19 downto 16) ;

         --RESERVED UNMASKED BIT
         new_cmd_batch.sca_cmds_data(0).data(20) := '1';
         new_cmd_batch.sca_cmds_data(0).data(15 downto 0) :=
            (others=>'0');
         new_cmd_batch.sca_cmds_data(0).len :=
            std_logic_vector(to_unsigned(4, 8));
      when ECS_SPI_R_CTRL =>
         new_cmd_batch.sca_cmds(0) := spi_cmd_table(SPI_rea_CTRL);
         new_cmd_batch.sca_cmds_data(0).data := (others=>'0');
         new_cmd_batch.sca_cmds_data(0).len := (others=>'0');
```

```
      new_cmd_batch.cmd_count := 1;
   when ECS_SPI_W_SS =>
      new_cmd_batch.sca_cmds(0) := spi_cmd_table(SPI_wrt_SS);
      --new_cmd_batch.sca_cmds_data(0).data(23 downto 16) :=  ecs_cmd_packet.data(23 downto 16);
      new_cmd_batch.sca_cmds_data(0).data(31 downto 24) :=  ecs_cmd_packet.data(31 downto 24);
      new_cmd_batch.sca_cmds_data(0).len := std_logic_vector(to_unsigned( 4 ,8));
      new_cmd_batch.cmd_count := 1;
   when ECS_SPI_R_SS =>
      new_cmd_batch.sca_cmds(0) := spi_cmd_table(SPI_rea_SS);
      new_cmd_batch.sca_cmds_data(0).data := (others=>'0');
      new_cmd_batch.sca_cmds_data(0).len := (others=>'0');
      new_cmd_batch.cmd_count := 1;
   when ECS_SPI_W_FREQ =>
      new_cmd_batch.sca_cmds(0) := spi_cmd_table(SPI_wrt_DIV);
      new_cmd_batch.sca_cmds_data(0).data(31 downto 16) :=  ecs_cmd_packet.data(31 downto 16);
      new_cmd_batch.sca_cmds_data(0).len := std_logic_vector(to_unsigned( 4 ,8));
      new_cmd_batch.cmd_count := 1;
   when ECS_SPI_R_FREQ =>
      new_cmd_batch.sca_cmds(0) := spi_cmd_table(SPI_rea_DIV);
      new_cmd_batch.sca_cmds_data(0).data := (others=>'0');
      new_cmd_batch.sca_cmds_data(0).len := (others=>'0');
      new_cmd_batch.cmd_count := 1;
   when ECS_SPI_MOSI_wrt_Tx0 =>
      new_cmd_batch.sca_cmds(0) := spi_cmd_table(SPI_MOSI_wrt_Tx0);
      new_cmd_batch.sca_cmds_data(0).data :=  ecs_cmd_packet.data(31 downto 0);
      new_cmd_batch.sca_cmds_data(0).len := ecs_cmd_packet.len;
      new_cmd_batch.cmd_count := 1;
   when ECS_SPI_MOSI_rea_Tx0 =>
      new_cmd_batch.sca_cmds(0) := spi_cmd_table(SPI_MOSI_rea_Tx0);
      new_cmd_batch.sca_cmds_data(0).data := (others=>'0');
      new_cmd_batch.sca_cmds_data(0).len := (others=>'0');
      new_cmd_batch.cmd_count := 1;
   when ECS_SPI_MOSI_wrt_Tx1 =>
      new_cmd_batch.sca_cmds(0) := spi_cmd_table(SPI_MOSI_wrt_Tx1);
      new_cmd_batch.sca_cmds_data(0).data :=  ecs_cmd_packet.data(31 downto 0);
      new_cmd_batch.sca_cmds_data(0).len := ecs_cmd_packet.len;
      new_cmd_batch.cmd_count := 1;
   when ECS_SPI_MOSI_rea_Tx1 =>
      new_cmd_batch.sca_cmds(0) := spi_cmd_table(SPI_MOSI_rea_Tx1);
      new_cmd_batch.sca_cmds_data(0).data := (others=>'0');
      new_cmd_batch.sca_cmds_data(0).len := (others=>'0');
      new_cmd_batch.cmd_count := 1;
   when ECS_SPI_MOSI_wrt_Tx2 =>
      new_cmd_batch.sca_cmds(0) := spi_cmd_table(SPI_MOSI_wrt_Tx2);
      new_cmd_batch.sca_cmds_data(0).data :=  ecs_cmd_packet.data(31 downto 0);
      new_cmd_batch.sca_cmds_data(0).len := ecs_cmd_packet.len;
      new_cmd_batch.cmd_count := 1;
   when ECS_SPI_MOSI_rea_Tx2 =>
      new_cmd_batch.sca_cmds(0) := spi_cmd_table(SPI_MOSI_rea_Tx2);
      new_cmd_batch.sca_cmds_data(0).data := (others=>'0');
      new_cmd_batch.sca_cmds_data(0).len := (others=>'0');
      new_cmd_batch.cmd_count := 1;
   when ECS_SPI_MOSI_wrt_Tx3 =>
      new_cmd_batch.sca_cmds(0) := spi_cmd_table(SPI_MOSI_wrt_Tx3);
      new_cmd_batch.sca_cmds_data(0).data :=  ecs_cmd_packet.data(31 downto 0);
      new_cmd_batch.sca_cmds_data(0).len := ecs_cmd_packet.len;
      new_cmd_batch.cmd_count := 1;
   when ECS_SPI_MOSI_rea_Tx3 =>
      new_cmd_batch.sca_cmds(0) := spi_cmd_table(SPI_MOSI_rea_Tx3);
      new_cmd_batch.sca_cmds_data(0).data := (others=>'0');
```

```
      new_cmd_batch.sca_cmds_data(0).len := (others=>'0');
      new_cmd_batch.cmd_count := 1;
   when ECS_SPI_MISO_rea_Rx0 =>
      new_cmd_batch.sca_cmds(0) := spi_cmd_table(SPI_MISO_rea_Rx0);
      new_cmd_batch.sca_cmds_data(0).data := (others=>'0');
      new_cmd_batch.sca_cmds_data(0).len := (others=>'0');
      new_cmd_batch.cmd_count := 1;
   when ECS_SPI_MISO_rea_Rx1 =>
      new_cmd_batch.sca_cmds(0) := spi_cmd_table(SPI_MISO_rea_Rx1);
      new_cmd_batch.sca_cmds_data(0).data := (others=>'0');
      new_cmd_batch.sca_cmds_data(0).len := (others=>'0');
      new_cmd_batch.cmd_count := 1;
   when ECS_SPI_MISO_rea_Rx2 =>
      new_cmd_batch.sca_cmds(0) := spi_cmd_table(SPI_MISO_rea_Rx2);
      new_cmd_batch.sca_cmds_data(0).data := (others=>'0');
      new_cmd_batch.sca_cmds_data(0).len := (others=>'0');
      new_cmd_batch.cmd_count := 1;
   when ECS_SPI_MISO_rea_Rx3 =>
      new_cmd_batch.sca_cmds(0) := spi_cmd_table(SPI_MISO_rea_Rx3);
      new_cmd_batch.sca_cmds_data(0).data := (others=>'0');
      new_cmd_batch.sca_cmds_data(0).len := (others=>'0');
      new_cmd_batch.cmd_count := 1;
   when ECS_SPI_WRITE_TX_REG =>
      k:=0;
      while (4*k < to_integer(unsigned(ecs_cmd_packet.len)) and k<4)  loop
         case k is
            when 0 =>
               new_cmd_batch.sca_cmds(k) := spi_cmd_table(SPI_MOSI_wrt_Tx0);
            when 1 =>
               new_cmd_batch.sca_cmds(k) := spi_cmd_table(SPI_MOSI_wrt_Tx1);
            when 2 =>
               new_cmd_batch.sca_cmds(k) := spi_cmd_table(SPI_MOSI_wrt_Tx2);
            when 3 =>
               new_cmd_batch.sca_cmds(k) := spi_cmd_table(SPI_MOSI_wrt_Tx3);
            when others => null;
         end case;

         new_cmd_batch.sca_cmds_data(k).data :=ecs_cmd_packet.data(32*(k+1)-1 downto 32*k);
         new_cmd_batch.sca_cmds_data(k).len := x"04"; --lazy
         k:=k+1;
      end loop;
      new_cmd_batch.cmd_count := k;
   when ECS_SPI_READ_RX_REG =>
      k:=0;
      while (4*k < to_integer(unsigned(ecs_cmd_packet.len)) and k<4) loop
         case k is
            when 0 =>
               new_cmd_batch.sca_cmds(k) := spi_cmd_table(SPI_MISO_rea_Rx0);
            when 1 =>
               new_cmd_batch.sca_cmds(k) := spi_cmd_table(SPI_MISO_rea_Rx1);
            when 2 =>
               new_cmd_batch.sca_cmds(k) := spi_cmd_table(SPI_MISO_rea_Rx2);
            when 3 =>
               new_cmd_batch.sca_cmds(k) := spi_cmd_table(SPI_MISO_rea_Rx3);
            when others => null;
         end case;

         new_cmd_batch.sca_cmds_data(k).data :=(others=>'0');
         new_cmd_batch.sca_cmds_data(k).len := x"00";
         k:=k+1;
```

```
            end loop;
            new_cmd_batch.cmd_count := k;
         when INVALID =>
            null;
      end case;

   end if;

   sca_cmd_batch_o <= new_cmd_batch;
end process spi_protocol_builder;


spi_protocol_decoder : process ( sca_rpy_batch_i, ecs_cmd_packet.len, prot_rpy_en_i) is
   variable new_ecs_rpy_packet      :  ecs_packet_t;
   variable ecs_spi_cmd       :  ecs_spi_cmd_enum;
   variable j                 :  natural range 0 to 128;
   variable k                 :  natural;


begin

   --new_ecs_rpy_packet.gbt_nr := (others=>'0');
   new_ecs_rpy_packet.sca_nr  := (others=>'0');
   new_ecs_rpy_packet.sca_ch  := (others=>'0');
   new_ecs_rpy_packet.ecs_cmd := (others=>'0');
   new_ecs_rpy_packet.data := (others=>'0');
   new_ecs_rpy_packet.len := (others=>'0');
   new_ecs_rpy_packet.protocol_specific  := (others=>'0');
   new_ecs_rpy_packet.err := (others=>'0');

   ecs_spi_cmd := INVALID;

   if prot_rpy_en_i='1' then

      for i in ecs_spi_cmd_table_t'left to ecs_spi_cmd_table_t'right loop
         if sca_rpy_batch_i.ecs_cmd = ecs_spi_cmd_table(i) then
            ecs_spi_cmd := i;
            exit;
         end if;
      end loop;
      new_ecs_rpy_packet.ecs_cmd := ecs_spi_cmd_table(ecs_spi_cmd);

      new_ecs_rpy_packet.sca_ch := sca_rpy_batch_i.ch;
      --new_ecs_rpy_packet.gbt_nr := std_logic_vector( to_unsigned( sca_rpy_batch_i.gbt, 8));
      new_ecs_rpy_packet.sca_nr := std_logic_vector( to_unsigned(sca_rpy_batch_i.sca,8));


      if sca_rpy_batch_i.err /= (sca_rpy_batch_i.err'range =>'0') then
         for j in 0 to sca_rpy_batch_i.err'high -1 loop
         if sca_rpy_batch_i.err(j)/='0' then
            -- \TODO What to do?
            report "There was an error on the GBT-SCA channel operation" & integer'image(j) severity
  Note;

            exit;
            end if;
         end loop;
         new_ecs_rpy_packet.err := sca_rpy_batch_i.err;
      end if;
```

```
    new_ecs_rpy_packet.protocol_specific(15 downto 0) := (others=>'0');

    case ecs_spi_cmd is
       when ECS_SPI_GO =>
          new_ecs_rpy_packet.data(31 downto 0) := sca_rpy_batch_i.sca_rpys_data(0).data;
          new_ecs_rpy_packet.len := std_logic_vector( to_unsigned(4,8));
       when ECS_SPI_W_CTRL =>
          null;
       when ECS_SPI_R_CTRL =>
          new_ecs_rpy_packet.data(31 downto 16) := sca_rpy_batch_i.sca_rpys_data(0).data(31 downto
16);
          new_ecs_rpy_packet.len :=  std_logic_vector( to_unsigned(4,8));
       when ECS_SPI_W_SS =>
          null;
       when ECS_SPI_R_SS =>
          new_ecs_rpy_packet.data(31 downto 24) := sca_rpy_batch_i.sca_rpys_data(0).data(31 downto
24);
          new_ecs_rpy_packet.len := std_logic_vector( to_unsigned(4,8));
       when ECS_SPI_W_FREQ =>
          null;
       when ECS_SPI_R_FREQ =>
          new_ecs_rpy_packet.data(31 downto 16) := sca_rpy_batch_i.sca_rpys_data(0).data(31 downto
16);
          new_ecs_rpy_packet.len := std_logic_vector( to_unsigned(4,8));
       when ECS_SPI_MOSI_wrt_Tx0 =>
          null;
       when ECS_SPI_MOSI_rea_Tx0 =>
          new_ecs_rpy_packet.data(31 downto 0) := sca_rpy_batch_i.sca_rpys_data(0).data;
          new_ecs_rpy_packet.len := std_logic_vector( to_unsigned(4,8));
       when ECS_SPI_MOSI_wrt_Tx1 =>
          null;
       when ECS_SPI_MOSI_rea_Tx1 =>
          new_ecs_rpy_packet.data(31 downto 0) := sca_rpy_batch_i.sca_rpys_data(0).data;
          new_ecs_rpy_packet.len := std_logic_vector( to_unsigned(4,8));
       when ECS_SPI_MOSI_wrt_Tx2 =>
          null;
       when ECS_SPI_MOSI_rea_Tx2 =>
          new_ecs_rpy_packet.data(31 downto 0) := sca_rpy_batch_i.sca_rpys_data(0).data;
          new_ecs_rpy_packet.len := std_logic_vector( to_unsigned(4,8));
       when ECS_SPI_MOSI_wrt_Tx3 =>
          null;
       when ECS_SPI_MOSI_rea_Tx3 =>
          new_ecs_rpy_packet.data(31 downto 0) := sca_rpy_batch_i.sca_rpys_data(0).data;
          new_ecs_rpy_packet.len := std_logic_vector( to_unsigned(4,8));
       when ECS_SPI_MISO_rea_Rx0 =>
          new_ecs_rpy_packet.data(31 downto 0) := sca_rpy_batch_i.sca_rpys_data(0).data;
          new_ecs_rpy_packet.len := std_logic_vector( to_unsigned(4,8));
       when ECS_SPI_MISO_rea_Rx1 =>
          new_ecs_rpy_packet.data(31 downto 0) := sca_rpy_batch_i.sca_rpys_data(0).data;
          new_ecs_rpy_packet.len := std_logic_vector( to_unsigned(4,8));
       when ECS_SPI_MISO_rea_Rx2 =>
          new_ecs_rpy_packet.data(31 downto 0) := sca_rpy_batch_i.sca_rpys_data(0).data;
          new_ecs_rpy_packet.len := std_logic_vector( to_unsigned(4,8));
       when ECS_SPI_MISO_rea_Rx3 =>
          new_ecs_rpy_packet.data(31 downto 0) := sca_rpy_batch_i.sca_rpys_data(0).data;
          new_ecs_rpy_packet.len := std_logic_vector( to_unsigned(4,8));
       when ECS_SPI_WRITE_TX_REG =>
          null;
       when ECS_SPI_READ_RX_REG =>
          new_ecs_rpy_packet.data := sca_rpy_batch_i.sca_rpys_data(3).data &
```

```
                    sca_rpy_batch_i.sca_rpys_data(2).data &
                    sca_rpy_batch_i.sca_rpys_data(1).data &
                    sca_rpy_batch_i.sca_rpys_data(0).data;
               if to_integer(unsigned(ecs_cmd_packet.len))>=16 then
                  k:=16;
               else
                  k := to_integer(unsigned(ecs_cmd_packet.len));
               end if;
               j:= new_ecs_rpy_packet.data'length - k*8;
               report "j";
               for k in 0 to  new_ecs_rpy_packet.data'high loop
                  if k >=  new_ecs_rpy_packet.data'high-j then
                  --new_ecs_rpy_packet.data(new_ecs_rpy_packet.data'high downto
                  -- new_ecs_rpy_packet.data'high-j) :=(others=>'0');
                  new_ecs_rpy_packet.data(k) := '0';
                  end if;
               end loop;
            when INVALID =>
               null;
         end case;

      end if;

      ecs_rpy_packet <= new_ecs_rpy_packet;


   end process spi_protocol_decoder;

end architecture RTL;
```

```
--------------------------------------------------------------------------------
--! @file timeout_timer.vhd
--! @author Cairo Caplan <cairo.caplan@cern.ch>
--! @date April, 2016
--------------------------------------------------------------------------------


library ieee;
use ieee.std_logic_1164.all;

use work.SCA_Package.all;
use ieee.numeric_std.all;

entity sca_roundtrip_timer is
   port (
      clk            :  in std_logic;
      en          :  in std_logic;
      rst            :  in std_logic;
      timeout_value_i   :  in std_logic_vector(25 downto 0);
      time_o         :  out   sca_time_t

   );
end entity sca_roundtrip_timer;

architecture RTL of sca_roundtrip_timer is
   -- Current time of the Timer, counted in clock cycles
   signal time :  sca_time_t;
   -- Preset value of the Internal Timeout, set when the user changes the
   -- global value of timeout_value_i
   signal pre_internal_timeout_value : std_logic_vector(25 downto 0);
   -- Value of the Internal Timeout, the Timer counts until this value
   -- then goes back to 0
   signal internal_timeout_value : std_logic_vector(25 downto 0);
   -- Proper Time, using the current internal_timeout_value as base,
   -- of when the Timer can update its own internal_timeout_value.
   --  This is used to avoid Packets using a former timeout value to
   -- wait forever/stuck at the cmd_queue_controller block
   signal scheduled_change_in_internal_timeout : sca_time_t;
   -- Flag that indicates the user has changed the timeout value.
   -- So the internal_timeout needs to be updated
   signal timeout_has_changed :  std_logic := '0';
begin

   timeout_process : process (clk, rst) is
      variable timeout_has_changed_var :  std_logic;
   begin
      if rst = '1' then
         time <= 0;
         timeout_has_changed_var := '0';
         internal_timeout_value <= std_logic_vector(to_unsigned(10*(2**10),26));
         pre_internal_timeout_value <= std_logic_vector(to_unsigned(10*(2**10),26));
      elsif rising_edge(clk) then
         if en ='1' then
            --if time = sca_time_t'high then
            if time = to_integer(unsigned(timeout_value_i)) then
               time <= 0;
            elsif timeout_has_changed = '1'
               and scheduled_change_in_internal_timeout = time then
                  internal_timeout_value <= pre_internal_timeout_value;
```

```
                timeout_has_changed_var := '0';
            else
               time <= time + 1;
            end if;

            if (internal_timeout_value /= timeout_value_i and timeout_has_changed='0')
            or (pre_internal_timeout_value /= timeout_value_i and timeout_has_changed='1') then
               timeout_has_changed_var := '1';
               if time=0 then
                  scheduled_change_in_internal_timeout <= to_integer(unsigned(timeout_value_i));
               else
                  scheduled_change_in_internal_timeout <= time - 1;
               end if;
               pre_internal_timeout_value <= timeout_value_i;
            end if;


         end if;
      end if;
      timeout_has_changed <= timeout_has_changed_var;
   end process timeout_process;

   time_o <= time;

end architecture RTL;
```

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

use work.SCA_Package.all;


--! This block contain all the command queues to be sent to their respectives gbt-sca channels
entity cmd_queue_controller is
   port (
      clk               :  in std_logic;
      rst               :  in std_logic;
      --Interface with the ECS Packets Buffer Layer
      ecs_cmd_prot_ena_o      :  out   std_logic;
      ecs_cmd_prot_av_i    :  in std_logic;

      ecs_rpy_int_av_i     :  in std_logic;
      ecs_rpy_int_ena_o    :  out   std_logic;
      --Interface with the Protocol Drivers
      sca_cmd_batch_i         :  in prot_cmd_t;
      sca_rpy_batch_o         :  out  prot_cmd_t;
      --
      timeout_value_i         :  in std_logic_vector(25 downto 0);
      --Interface with the MAC Layer

      sca_cmd_data_o          :  out payload_vector_t(0 to SCA_COUNT - 1);
      sca_cmd_ena_o           :  out std_logic_vector(0 to SCA_COUNT -1);
      sca_cmd_rdy_i           :  in    std_logic_vector(0 to SCA_COUNT -1);
      --
      sca_rpy_data_i          :  in payload_vector_t(0 to SCA_COUNT - 1);
      sca_rpy_prot_rdy_o      :  out std_logic_vector(0 to SCA_COUNT -1);
      send_sca_rpy_i          :  in    std_logic_vector(0 to SCA_COUNT - 1);

      reconnnect_o         :  out   std_logic_vector(0 to SCA_COUNT -1)


   );
end entity cmd_queue_controller;

architecture RTL of cmd_queue_controller is


   signal command_queue :  command_queue_t;

   component sca_roundtrip_timer
      port(
         clk             : in  std_logic;
         en              : in  std_logic;
         rst             : in  std_logic;
         timeout_value_i : in  std_logic_vector(25 downto 0);
         time_o          : out sca_time_t
      );
   end component sca_roundtrip_timer;

   signal sca_time : sca_time_t;


begin
```

```
   sca_roundtrip_timer_inst : sca_roundtrip_timer
      port map(
         clk    => clk,
         en     => '1',
         rst    => rst,
         timeout_value_i => timeout_value_i,
         time_o => sca_time
      );


controller_state_machine : process (clk) is
   variable curr_state  :  prot_state_t;

   variable ecs_cmd_taken  :  std_logic;
   variable ecs_rpy_sent   :  std_logic;
   variable links_taken    :  std_logic_vector(0 to SCA_COUNT-1) := (others=>'0');
   variable sca_cmd_ena_var   :  std_logic_vector(0 to SCA_COUNT -1);

   variable curr_sca_index    :  natural range 0 to SCA_COUNT-1;

   variable reconnnect_var :  std_logic_vector(0 to SCA_COUNT -1);
begin

   reconnnect_var := (others=>('0'));

   if rising_edge(clk) then
         ecs_cmd_taken :='0';
         ecs_rpy_sent := '0';
         links_taken := (others=>'0');
      for i in 0 to command_queue.prot_cmd'high loop
            curr_state := command_queue.prot_state(i);
            curr_sca_index := command_queue.prot_cmd(i).sca;
         if rst = '1' then
            for i in 0 to command_queue.prot_cmd'high loop
                  command_queue.prot_cmd(i).sca <= 0;
                  command_queue.prot_cmd(i).ch <= (others=>'0');
                  command_queue.prot_cmd(i).cmd_count <= 0;
                  command_queue.prot_cmd(i).ecs_cmd <= (others=>'0');
                  for j in 0 to command_queue.prot_cmd(i).sca_cmds'high loop
                     command_queue.prot_cmd(i).sca_cmds(j) <= (others=>'0');
                     command_queue.prot_cmd(i).sca_cmds_data(j).data <= (others=>'0');
                     command_queue.prot_cmd(i).sca_cmds_data(j).len <= (others=>'0');
                  end loop;
                  command_queue.counter(i) <= 0;
                  command_queue.prot_cmd(i).err <= (others=>'0');
                  command_queue.prot_state(i) <= IDLE;
                  command_queue.prot_cmd(i).tr <= x"01";
            end loop;
            sca_cmd_ena_var := (others=>'0');
            sca_rpy_prot_rdy_o <= (others=>'0');
         else

               case curr_state is
                  when IDLE =>

                     if ecs_cmd_prot_av_i = '1' and ecs_cmd_taken ='0' then
                        ecs_cmd_taken := '1';
                        curr_state := GET_SCA_CMD;
                     end if;
                  when GET_SCA_CMD =>
```

```
                curr_state := SEND_SCA_CMD;
                command_queue.prot_cmd(i) <= sca_cmd_batch_i;

            when SEND_SCA_CMD =>

                if  sca_cmd_rdy_i(curr_sca_index)='1' and links_taken(curr_sca_index)='0' then
                    links_taken(curr_sca_index):='1';

                    sca_cmd_ena_var(curr_sca_index) := '1';
                    sca_cmd_data_o(curr_sca_index).ch <= command_queue.prot_cmd(i).ch;
                    sca_cmd_data_o(curr_sca_index).cmd_or_err <=
command_queue.prot_cmd(i).sca_cmds(command_queue.counter(i));
                    sca_cmd_data_o(curr_sca_index).data <=
command_queue.prot_cmd(i).sca_cmds_data(command_queue.counter(i)).data;
                    sca_cmd_data_o(curr_sca_index).len <=
command_queue.prot_cmd(i).sca_cmds_data(command_queue.counter(i)).len;
                    sca_cmd_data_o(curr_sca_index).tr <= command_queue.prot_cmd(i).tr;


                    curr_state := WAIT_SCA_RPY;
                    -- Set Expiration Time Value
                    if sca_time=0 then
                        command_queue.expiration_time(i) <=
                            to_integer(unsigned(timeout_value_i)-1);
                        --command_queue.expiration_time(i) <= sca_time_t'high;
                    else
                        command_queue.expiration_time(i) <= sca_time - 1;
                    end if;


                end if;
            when WAIT_SCA_RPY =>
                sca_cmd_ena_var(curr_sca_index) := '0';
                if send_sca_rpy_i(curr_sca_index)='1'
                and (sca_rpy_data_i(curr_sca_index).ch = command_queue.prot_cmd(i).ch ) then

                    command_queue.prot_cmd(i).sca_rpys_data(command_queue.counter(i)).data <=
sca_rpy_data_i(curr_sca_index).data;
                    command_queue.prot_cmd(i).sca_rpys_data(command_queue.counter(i)).len <=
sca_rpy_data_i(curr_sca_index).len;
                    if sca_rpy_data_i(curr_sca_index).cmd_or_err /=
(sca_rpy_data_i(curr_sca_index).cmd_or_err'range=>'0') then
                        command_queue.prot_cmd(i).err <= sca_rpy_data_i(curr_sca_index).cmd_or_err;
                        curr_state := ECS_RPY_RDY;
                    elsif (sca_rpy_data_i(curr_sca_index).tr /= command_queue.prot_cmd(i).tr) or
(sca_rpy_data_i(curr_sca_index).ch /=  command_queue.prot_cmd(i).ch) then
                        command_queue.prot_cmd(i).err <= x"01";
                        curr_state := ECS_RPY_RDY;
                    elsif (command_queue.counter(i))= command_queue.prot_cmd(i).cmd_count - 1 then
                        curr_state := ECS_RPY_RDY;
                    else
                        if (command_queue.counter(i) < command_queue.prot_cmd(i).cmd_count) then
                            command_queue.counter(i) <= command_queue.counter(i) + 1;
                            curr_state := SEND_SCA_CMD;
                        else
                            curr_state := ECS_RPY_RDY;
                            sca_rpy_prot_rdy_o(curr_sca_index) <= '1';
                        end if;
                    end if;
                    sca_rpy_prot_rdy_o(curr_sca_index) <= '1';
```

```
                    else
                       --If SCA didn‘t respond, end the packet queue transmission and assert Error 01h on
    the ECS Reply
                       if sca_time = command_queue.expiration_time(i) then
                          command_queue.prot_cmd(i).err <= x"01";
                          curr_state := ECS_RPY_RDY;
                          sca_rpy_prot_rdy_o(curr_sca_index) <= '1';
                          reconnnect_var(curr_sca_index) := '1';
                       end if;
                    end if;
                 when ECS_RPY_RDY =>
                    if ecs_rpy_sent='0' and ecs_rpy_int_av_i='1' then
                       curr_state := IDLE;
                       ecs_rpy_sent := '1';

                       sca_rpy_batch_o <= command_queue.prot_cmd(i);
                       command_queue.prot_cmd(i).ecs_cmd <= x"FF";
                    else
                       curr_state := ECS_RPY_RDY;
                    end if;
                    command_queue.counter(i) <= 0;
              end case;
              command_queue.prot_state(i) <= curr_state;

         end if;


      end loop;

      sca_cmd_ena_o <= sca_cmd_ena_var;

      ecs_cmd_prot_ena_o <= ecs_cmd_taken;
      ecs_rpy_int_ena_o <= ecs_rpy_sent;

      reconnnect_o <= reconnnect_var;
   end if;

end process controller_state_machine;


end architecture RTL;
```

```
--------------------------------------------------------------------------------
--! @file protocol_layer.vhd
--! @author Cairo Caplan <cairo.caplan@cern.ch>
--! @date April, 2016
--------------------------------------------------------------------------------

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

use work.SCA_Package.all;

entity protocol_layer is
   port (
      clk            :  in std_logic;
      rst            :  in std_logic;
      --ECS Protocol Layer
      ecs_cmd_prot_ena_o   :  out   std_logic;
      ecs_cmd_prot_av_i :  in std_logic;
      ecs_cmd_prot_data_i  :  in ecs_packet_t;

      ecs_rpy_prot_av_i :  in std_logic;
      ecs_rpy_prot_ena_o   :  out   std_logic;
      ecs_rpy_prot_data_o  :  out   ecs_packet_t;
      --
      timeout_value_i      :  in std_logic_vector(25 downto 0);
      --Interface with the MAC Layer

      sca_cmd_data_o       :  out payload_vector_t(0 to SCA_COUNT - 1);
      sca_cmd_ena_o        :  out std_logic_vector(0 to SCA_COUNT -1);
      sca_cmd_rdy_i        :  in    std_logic_vector(0 to SCA_COUNT -1);
      --
      sca_rpy_data_i       :  in payload_vector_t(0 to SCA_COUNT - 1);
      sca_rpy_prot_rdy_o   :  out std_logic_vector(0 to SCA_COUNT -1);
      send_sca_rpy_i       :  in    std_logic_vector(0 to SCA_COUNT -1);
      reconnnect_o      :  out   std_logic_vector(0 to SCA_COUNT -1)
      --
   );
end entity protocol_layer;

architecture RTL of protocol_layer is

   component sca_controller_protocol_driver
      port(clk             : in  std_logic;
          rst              : in  std_logic;
          prot_cmd_en_i       : in  std_logic;
          ecs_cmd_packet         : in  ecs_packet_t;
          prot_rpy_en_i       : in  std_logic;
          ecs_rpy_packet         : out ecs_packet_t;
          sca_cmd_batch_o     : out prot_cmd_t;
          sca_rpy_batch_i     : in  prot_cmd_t;
          tr_cmd_i            : in  byte_t;
          activated_channels_o   : out activated_channels_t);
   end component sca_controller_protocol_driver;

   component spi_protocol_driver
      port(prot_cmd_en_i   : in  std_logic;
          ecs_cmd_packet  : in  ecs_packet_t;
          prot_rpy_en_i   : in  std_logic;
```

```
        ecs_rpy_packet  : out ecs_packet_t;
        sca_cmd_batch_o : out prot_cmd_t;
        sca_rpy_batch_i : in  prot_cmd_t;
        tr_cmd_i        : in  byte_t);
end component spi_protocol_driver;

component gpio_protocol_driver
   port(prot_cmd_en_i   : in  std_logic;
        ecs_cmd_packet  : in  ecs_packet_t;
        prot_rpy_en_i   : in  std_logic;
        ecs_rpy_packet  : out ecs_packet_t;
        sca_cmd_batch_o : out prot_cmd_t;
        sca_rpy_batch_i : in  prot_cmd_t;
        tr_cmd_i        : in  byte_t);
end component gpio_protocol_driver;

component i2c_protocol_driver
   port(prot_cmd_en_i         : in  std_logic;
        ecs_cmd_packet        : in  ecs_packet_t;
        prot_rpy_en_i       : in  std_logic;
        ecs_rpy_packet        : out ecs_packet_t;
       sca_cmd_batch_o        : out prot_cmd_t;
        sca_rpy_batch_i     : in  prot_cmd_t;
        tr_cmd_i            : in  byte_t   );
end component i2c_protocol_driver;

component jtag_protocol_driver
   port(prot_cmd_en_i   : in  std_logic;
        ecs_cmd_packet  : in  ecs_packet_t;
        prot_rpy_en_i   : in  std_logic;
        ecs_rpy_packet  : out ecs_packet_t;
        sca_cmd_batch_o : out prot_cmd_t;
        sca_rpy_batch_i : in  prot_cmd_t;
        tr_cmd_i        : in  byte_t);
end component jtag_protocol_driver;

component adc_protocol_driver
   port(prot_cmd_en_i   : in  std_logic;
        ecs_cmd_packet  : in  ecs_packet_t;
        prot_rpy_en_i   : in  std_logic;
        ecs_rpy_packet  : out ecs_packet_t;
        sca_cmd_batch_o : out prot_cmd_t;
        sca_rpy_batch_i : in  prot_cmd_t;
        tr_cmd_i        : in  byte_t);
end component adc_protocol_driver;

component dac_protocol_driver
   port(prot_cmd_en_i   : in  std_logic;
        ecs_cmd_packet  : in  ecs_packet_t;
        prot_rpy_en_i   : in  std_logic;
        ecs_rpy_packet  : out ecs_packet_t;
        sca_cmd_batch_o : out prot_cmd_t;
        sca_rpy_batch_i : in  prot_cmd_t;
        tr_cmd_i        : in  byte_t);
end component dac_protocol_driver;

component cmd_queue_controller
   port(
      clk               : in  std_logic;
      rst               : in  std_logic;
```

```
        ecs_cmd_prot_ena_o : out std_logic;
        ecs_cmd_prot_av_i  : in  std_logic;
        ecs_rpy_int_av_i   : in  std_logic;
        ecs_rpy_int_ena_o  : out std_logic;
        sca_cmd_batch_i    : in  prot_cmd_t;
        sca_rpy_batch_o    : out prot_cmd_t;
        timeout_value_i    : in  std_logic_vector(25 downto 0);
        sca_cmd_data_o     : out payload_vector_t(0 to SCA_COUNT - 1);
        sca_cmd_ena_o      : out std_logic_vector(0 to SCA_COUNT - 1);
        sca_cmd_rdy_i      : in  std_logic_vector(0 to SCA_COUNT - 1);
        sca_rpy_data_i     : in  payload_vector_t(0 to SCA_COUNT - 1);
        sca_rpy_prot_rdy_o : out std_logic_vector(0 to SCA_COUNT - 1);
        send_sca_rpy_i     : in  std_logic_vector(0 to SCA_COUNT - 1);
        reconnnect_o       : out std_logic_vector(0 to SCA_COUNT - 1)
    );
  end component cmd_queue_controller;

  type activate_protocol_t is array(PROT_DRIVER_ENUM_t'left to PROT_DRIVER_ENUM_t'right) of std_logic;

  signal prot_cmd_en_by_prot : activate_protocol_t;


  type ecs_packet_by_prot_t is array(PROT_DRIVER_ENUM_t'left to PROT_DRIVER_ENUM_t'right) of ecs_packet_t;

  signal tr : byte_t;


  signal ecs_rpy_packet_by_prot : ecs_packet_by_prot_t;

  signal activated_channels : activated_channels_t := (others=>(others=>'1'));
--
  signal tr_sca_array  :  byte_vector_t(0 to SCA_COUNT -1) := (others=>(x"01"));
-- --



  signal sca_cmd_batch : prot_cmd_t;
  signal sca_rpy_batch : prot_cmd_t;



  signal ecs_cmd_prot_ena    :  std_logic;

  signal prot_rpy_en_by_prot    :  activate_protocol_t;
  signal ecs_rpy_int_ena : std_logic;
  signal sca_cmd_batch_by_prot : prot_cmd_vector_by_prot_t;



begin

  cmd_queue_controller_inst : component cmd_queue_controller
    port map(clk                => clk,
             rst                => rst,
             ecs_cmd_prot_ena_o => ecs_cmd_prot_ena,
             ecs_cmd_prot_av_i  => ecs_cmd_prot_av_i,
             ecs_rpy_int_av_i   => ecs_rpy_prot_av_i,
             ecs_rpy_int_ena_o  => ecs_rpy_int_ena,
             sca_cmd_batch_i    => sca_cmd_batch,
             sca_rpy_batch_o    => sca_rpy_batch,
```

```
          timeout_value_i    => timeout_value_i,
           sca_cmd_data_o     => sca_cmd_data_o,
           sca_cmd_ena_o      => sca_cmd_ena_o,
           sca_cmd_rdy_i  => sca_cmd_rdy_i,
           sca_rpy_data_i     => sca_rpy_data_i,
           sca_rpy_prot_rdy_o => sca_rpy_prot_rdy_o,
           send_sca_rpy_i     => send_sca_rpy_i,
           reconnnect_o    => reconnnect_o);

ecs_rpy_prot_ena_o <= ecs_rpy_int_ena;
ecs_cmd_prot_ena_o <= ecs_cmd_prot_ena;

sca_controller_protocol_driver_inst : component sca_controller_protocol_driver
   port map(clk                  => clk,
            rst                  => rst,
            prot_cmd_en_i        => prot_cmd_en_by_prot(SCA_CTL),
            ecs_cmd_packet       => ecs_cmd_prot_data_i,
            prot_rpy_en_i        => prot_rpy_en_by_prot(SCA_CTL),
            ecs_rpy_packet       => ecs_rpy_packet_by_prot(SCA_CTL),
            sca_cmd_batch_o      => sca_cmd_batch_by_prot(SCA_CTL),
            sca_rpy_batch_i      => sca_rpy_batch,
            tr_cmd_i          => tr,
            activated_channels_o => activated_channels);


spi_protocol_driver_inst : component spi_protocol_driver
   port map(prot_cmd_en_i   => prot_cmd_en_by_prot(SPI),
            ecs_cmd_packet  => ecs_cmd_prot_data_i,
            prot_rpy_en_i   => prot_rpy_en_by_prot(SPI),
            ecs_rpy_packet  => ecs_rpy_packet_by_prot(SPI),
            sca_cmd_batch_o => sca_cmd_batch_by_prot(SPI),
            sca_rpy_batch_i => sca_rpy_batch,
            tr_cmd_i        => tr);

gpio_protocol_driver_inst : component gpio_protocol_driver
port map(prot_cmd_en_i  => prot_cmd_en_by_prot(GPIO),
        ecs_cmd_packet  => ecs_cmd_prot_data_i,
        prot_rpy_en_i   => prot_rpy_en_by_prot(GPIO),
        ecs_rpy_packet  => ecs_rpy_packet_by_prot(GPIO),
        sca_cmd_batch_o => sca_cmd_batch_by_prot(GPIO),
        sca_rpy_batch_i => sca_rpy_batch,
        tr_cmd_i        => tr);


i2c_protocol_driver_inst : component i2c_protocol_driver
   port map(prot_cmd_en_i   => prot_cmd_en_by_prot(I2C),
            ecs_cmd_packet  => ecs_cmd_prot_data_i,
            prot_rpy_en_i   => prot_rpy_en_by_prot(I2C),
            ecs_rpy_packet  => ecs_rpy_packet_by_prot(I2C),
            sca_cmd_batch_o => sca_cmd_batch_by_prot(I2C),
            sca_rpy_batch_i => sca_rpy_batch,
            tr_cmd_i        => tr);

jtag_protocol_driver_inst : component jtag_protocol_driver
   port map(prot_cmd_en_i   => prot_cmd_en_by_prot(JTAG),
            ecs_cmd_packet  => ecs_cmd_prot_data_i,
            prot_rpy_en_i   => prot_rpy_en_by_prot(JTAG),
            ecs_rpy_packet  => ecs_rpy_packet_by_prot(JTAG),
            sca_cmd_batch_o => sca_cmd_batch_by_prot(JTAG),
            sca_rpy_batch_i => sca_rpy_batch,
```

```
            tr_cmd_i        => tr);

adc_protocol_driver_inst : component adc_protocol_driver
   port map(prot_cmd_en_i   => prot_cmd_en_by_prot(ADC),
            ecs_cmd_packet  => ecs_cmd_prot_data_i,
            prot_rpy_en_i   => prot_rpy_en_by_prot(ADC),
            ecs_rpy_packet  => ecs_rpy_packet_by_prot(ADC),
            sca_cmd_batch_o => sca_cmd_batch_by_prot(ADC),
            sca_rpy_batch_i => sca_rpy_batch,
            tr_cmd_i        => tr);

dac_protocol_driver_inst : component dac_protocol_driver
   port map(prot_cmd_en_i   => prot_cmd_en_by_prot(DAC),
            ecs_cmd_packet  => ecs_cmd_prot_data_i,
            prot_rpy_en_i   => prot_rpy_en_by_prot(DAC),
            ecs_rpy_packet  => ecs_rpy_packet_by_prot(DAC),
            sca_cmd_batch_o => sca_cmd_batch_by_prot(DAC),
            sca_rpy_batch_i => sca_rpy_batch,
            tr_cmd_i        => tr);



--! This process generates the TR field of the Payload packet, it is agnostic to the type of the
  command.
--! This value is retrieved on replies from the associated SCA.
next_tr_generator :  process(rst, clk)
begin
   if rst = '1' then
      for i in 0 to SCA_COUNT - 1 loop
         tr_sca_array(i) <= x"01";
      end loop;
   elsif rising_edge(clk) then
      if ecs_cmd_prot_av_i = '1' then
         --TR values of 255 are used for interruptions only, which could happen only on replies
         if tr_sca_array(to_integer(unsigned(ecs_cmd_prot_data_i.sca_nr))) >=
  std_logic_vector(to_unsigned(254,8)) then
            tr_sca_array(to_integer(unsigned(ecs_cmd_prot_data_i.sca_nr))) <=
  std_logic_vector(to_unsigned(1,8));
         else
            tr_sca_array(to_integer(unsigned(ecs_cmd_prot_data_i.sca_nr))) <=

  std_logic_vector(unsigned(tr_sca_array(to_integer(unsigned(ecs_cmd_prot_data_i.sca_nr)))) + 1);
         end if;
      end if;
   end if;
end process next_tr_generator;

tr <= tr_sca_array(to_integer(unsigned(ecs_cmd_prot_data_i.sca_nr)));

rpy_protocol_arbiter : process (ecs_rpy_int_ena, sca_rpy_batch.ch, ecs_rpy_packet_by_prot)
   variable prot_rpy_en_by_prot_var    :  activate_protocol_t;
   variable ecs_rpy_int_data_var       :  ecs_packet_t;
begin
   prot_rpy_en_by_prot_var:=(others=>'0');
   ecs_rpy_int_data_var := ecs_rpy_packet_by_prot(SCA_CTL);
   if ecs_rpy_int_ena='1' then
      case sca_rpy_batch.ch is
         when x"00" =>
            prot_rpy_en_by_prot_var(SCA_CTL) := '1';
```

```
            ecs_rpy_int_data_var := ecs_rpy_packet_by_prot(SCA_CTL);
         when x"01" =>
            prot_rpy_en_by_prot_var(SPI) := '1';
            ecs_rpy_int_data_var := ecs_rpy_packet_by_prot(SPI);
         when x"02" =>
            prot_rpy_en_by_prot_var(GPIO) := '1';
            ecs_rpy_int_data_var := ecs_rpy_packet_by_prot(GPIO);
         when x"03" | x"04" | x"05" | x"06" | x"07" | x"08" | x"09" | x"0A"
         | x"0B" | x"0C" | x"0D" | x"0E" | x"0F" | x"10" | x"11" | x"12" =>
            prot_rpy_en_by_prot_var(I2C) := '1';
            ecs_rpy_int_data_var := ecs_rpy_packet_by_prot(I2C);
         when x"13" =>
            prot_rpy_en_by_prot_var(JTAG) := '1';
            ecs_rpy_int_data_var := ecs_rpy_packet_by_prot(JTAG);
         when x"14" =>
            prot_rpy_en_by_prot_var(ADC) := '1';
            ecs_rpy_int_data_var := ecs_rpy_packet_by_prot(ADC);
         when x"15" =>
            prot_rpy_en_by_prot_var(DAC) := '1';
            ecs_rpy_int_data_var := ecs_rpy_packet_by_prot(DAC);
         when others =>
            report "rpy_protocol_arbiter: Invalid Channel, what to do?" severity warning;
            prot_rpy_en_by_prot_var := (others=>'0');
            ecs_rpy_int_data_var := ecs_rpy_packet_by_prot(SCA_CTL);
      end case;
   end if;

   ecs_rpy_prot_data_o <= ecs_rpy_int_data_var;
   prot_rpy_en_by_prot  <= prot_rpy_en_by_prot_var;
end process rpy_protocol_arbiter;




cmd_protocol_arbiter : process (ecs_cmd_prot_data_i.sca_ch, activated_channels,
        ecs_cmd_prot_data_i.sca_nr, sca_cmd_batch_by_prot, ecs_cmd_prot_ena
) is
   variable prot_cmd_en_by_prot_var :  activate_protocol_t;
   variable sca_cmd_batch_var    :  prot_cmd_t;
begin
   prot_cmd_en_by_prot_var := (others=>'0');
   sca_cmd_batch_var := sca_cmd_batch_by_prot(SCA_CTL);
   if ecs_cmd_prot_ena = '1' then

      if activated_channels(to_integer(unsigned(ecs_cmd_prot_data_i.sca_nr)))(0) = '1' and

  activated_channels(to_integer(unsigned(ecs_cmd_prot_data_i.sca_nr)))(to_integer(unsigned(ecs_cmd_prot_data_i.sca_ch)
  then
         case ecs_cmd_prot_data_i.sca_ch is
         when x"00" =>
            prot_cmd_en_by_prot_var(SCA_CTL) := '1';
            sca_cmd_batch_var := sca_cmd_batch_by_prot(SCA_CTL);

         when x"01" =>
            prot_cmd_en_by_prot_var(SPI) := '1';
            sca_cmd_batch_var := sca_cmd_batch_by_prot(SPI);
         when x"02" =>
            prot_cmd_en_by_prot_var(GPIO) := '1';
            sca_cmd_batch_var := sca_cmd_batch_by_prot(GPIO);
         when x"03" | x"04" | x"05" | x"06" | x"07" | x"08" | x"09" | x"0A"
```

```
            | x"0B" | x"0C" | x"0D" | x"0E" | x"0F" | x"10" | x"11" | x"12" =>
               prot_cmd_en_by_prot_var(I2C) := '1';
               sca_cmd_batch_var := sca_cmd_batch_by_prot(I2C);
            when x"13" =>
               prot_cmd_en_by_prot_var(JTAG) := '1';
               sca_cmd_batch_var := sca_cmd_batch_by_prot(JTAG);
            when x"14" =>
               prot_cmd_en_by_prot_var(ADC) := '1';
               sca_cmd_batch_var := sca_cmd_batch_by_prot(ADC);
            when x"15" =>
               prot_cmd_en_by_prot_var(DAC) := '1';
               sca_cmd_batch_var := sca_cmd_batch_by_prot(DAC);
            when others =>
               report "cmd_protocol_arbiter : Invalid Channel, what to do?" severity warning;
               --ecs_cmd_prot_arbiter <= SCA_CTL;
               prot_cmd_en_by_prot_var := (others=>'0');
               sca_cmd_batch_var := sca_cmd_batch_by_prot(SCA_CTL);
            end case;
         else
            -- \TODO Report/return the ECS when the channel is not activated or invalid
            report "cmd_protocol_arbiter : Channel not activated, what to do?" severity warning;
            prot_cmd_en_by_prot_var := (others=>'0');
            sca_cmd_batch_var := sca_cmd_batch_by_prot(SCA_CTL);
         end if;
      else
         prot_cmd_en_by_prot_var := (others=>'0');
         sca_cmd_batch_var := sca_cmd_batch_by_prot(SCA_CTL);
      end if;
      sca_cmd_batch <= sca_cmd_batch_var;
      prot_cmd_en_by_prot <= prot_cmd_en_by_prot_var;
   end process cmd_protocol_arbiter;


end architecture RTL;
```

B.2 Arquivos de Simulação

```
--------------------------------------------------------------------------------
--! @file crc_tb.vhd
--! @author Cairo Caplan <cairo.caplan@cern.ch>
--! @date April, 2016
--------------------------------------------------------------------------------

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use ieee.std_logic_textio.all;

use std.textio.all;                     -- basic I/O

library OSVVM;
use OSVVM.RandomPkg.all;
use OSVVM.CoveragePkg.all;

entity crc_tb is
```

```
end entity crc_tb;

architecture RTL of crc_tb is

   subtype byte_t is  std_logic_vector(7 downto 0);
   type byte_vector_t is array(integer range <>) of byte_t;

   component crc
      generic(CRC_WIDTH  : natural                                := 16;
              DATA_WIDTH : natural                               := 8;
              INIT_VAL   : std_logic_vector(2 * 8 - 1 downto 0) := x"ffff";
              POLY       : std_logic_vector(2 * 8 - 1 downto 0) := x"8408");
      port(d       : in  std_logic_vector(DATA_WIDTH - 1 downto 0);
           init    : in  std_logic;
           d_valid : in  std_logic;
           clk     : in  std_logic;
           reset_b : in  std_logic;
           crc_o   : out std_logic_vector(CRC_WIDTH - 1 downto 0));
   end component crc;


   constant DATA_WIDTH  :  integer := 8;
   constant CRC_WIDTH   :  integer  := 16;
   signal d : std_logic_vector(DATA_WIDTH - 1 downto 0);
   signal init : std_logic:='1';
   signal d_valid : std_logic;
   signal clk : std_logic := '0';
   signal reset_b : std_logic := '0';
   signal crc_o : std_logic_vector(CRC_WIDTH - 1 downto 0);

   constant clk_period : time := 25 ns;

   constant POLY  : std_logic_vector(2 * 8 - 1 downto 0) := x"8408";
   constant INIT_VAL : std_logic_vector(2 * 8 - 1 downto 0) := x"ffff";

     -- coverage object of (protected) type CovPType
   shared variable sv_coverage : CovPType;

   signal crc_o_vlog : std_logic_vector(CRC_WIDTH - 1 downto 0);

   signal stop :  boolean := false;

begin


   crc_inst : component crc
      generic map(CRC_WIDTH  => CRC_WIDTH,
                  DATA_WIDTH => DATA_WIDTH,
                  INIT_VAL   => INIT_VAL,
                  POLY       => POLY )
      port map(d       => d,
               init    => init,
               d_valid => d_valid,
               clk     => clk,
               reset_b => reset_b,
               crc_o   => crc_o);

   crc_iostate_inst : entity work.crc_iostate
      generic map(
         CRC_WIDTH  => CRC_WIDTH,
```

```
      DATA_WIDTH => DATA_WIDTH,
      INIT_VAL   => INIT_VAL,
      POLY       => POLY
   )
   port map(
      d       => d,
      init    => init,
      d_valid => d_valid,
      clk     => clk,
      reset_b => reset_b,
      crc_o   => crc_o_vlog,
      crc_i   => crc_o_vlog
   );



clk_stimulus : process is
begin
   if clk='0' then clk<='1'; else clk<='0'; end if;
   wait for clk_period/2;
   if stop then
      wait;
   end if;
end process clk_stimulus;




stimulus_process : process is
   constant max_bytes_nr   :  integer := 10;
   variable test_data      :  byte_vector_t(0 to max_bytes_nr-1);
   variable Rxy : RandomPType;             -- object for generating sensor indices
   variable buf_line :  line;
   file hw_crc_output_file : TEXT open WRITE_MODE is "hw_crc_output.csv";

begin


   write(buf_line, string'("DATA;HW_CRC;VLOG_CRC"));
   --Write results to file hw_crc_output.csv
   writeline(hw_crc_output_file, buf_line);


   for byte_count in 1 to max_bytes_nr loop -- number of sets of bytes tested
      for k in 1 to 10000 loop -- number of data provided
         for j in 0 to byte_count-1 loop
            test_data(j) := Rxy.RandSlv(8);
            hwrite(buf_line,test_data(j));
         end loop;
         write(buf_line, string'(";"));

         ---- Hardware CRC ----
         --Reset state
         reset_b <= '0';
         init <='0';
         d <= x"00";
         wait for clk_period;
         d_valid <= '0';
         --
```

```
            for i in 0 to byte_count-1 loop
               if i=0 then
                  init <='1';
               else
                  init <='0';
               end if;
               reset_b <= '1';
               d <= test_data(i);
               wait for clk_period;
               d_valid <='1';
            end loop;
            -- Finish
            wait for clk_period;
            d_valid <= '0';
            reset_b <='0';
            ---------------------
            hwrite(buf_line,crc_o);
            write(buf_line, string'(";"));
            hwrite(buf_line,crc_o_vlog);
            if (crc_o /= crc_o_vlog) then
               report "CRC mismatch! crc_o = " & to_hstring(crc_o)
               & ", crc_o_vlog = "  & to_hstring(crc_o_vlog)
                  severity note;

            end if;
            --Write results to screen
            writeline(hw_crc_output_file, buf_line);
         end loop;
      end loop;
      stop <= true;

      wait;



   end process stimulus_process;


end architecture RTL;
```

```
////////////////////////////////////////////////////////////////////////////
/// @file crc_hdlc.c
/// @author Cairo Caplan <cairo.caplan@cern.ch>
/// @date April, 2016
///
/// Based on the CRC16-CCITT implementation for the PPP/HDLC protocol(RFC 1171)
/// https://tools.ietf.org/html/rfc1171#page-44
////////////////////////////////////////////////////////////////////////////



#include <assert.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
   /*
    * u16 represents an unsigned 16-bit number.  Adjust the typedef for
    * your hardware.
    */
   typedef unsigned short u16;


   /*
    * FCS lookup table as calculated by the table generator in section 2.
    */
   static u16 fcstab[256] = {
      0x0000, 0x1189, 0x2312, 0x329b, 0x4624, 0x57ad, 0x6536, 0x74bf,
      0x8c48, 0x9dc1, 0xaf5a, 0xbed3, 0xca6c, 0xdbe5, 0xe97e, 0xf8f7,
      0x1081, 0x0108, 0x3393, 0x221a, 0x56a5, 0x472c, 0x75b7, 0x643e,
      0x9cc9, 0x8d40, 0xbfdb, 0xae52, 0xdaed, 0xcb64, 0xf9ff, 0xe876,
      0x2102, 0x308b, 0x0210, 0x1399, 0x6726, 0x76af, 0x4434, 0x55bd,
      0xad4a, 0xbcc3, 0x8e58, 0x9fd1, 0xeb6e, 0xfae7, 0xc87c, 0xd9f5,
      0x3183, 0x200a, 0x1291, 0x0318, 0x77a7, 0x662e, 0x54b5, 0x453c,
      0xbdcb, 0xac42, 0x9ed9, 0x8f50, 0xfbef, 0xea66, 0xd8fd, 0xc974,
      0x4204, 0x538d, 0x6116, 0x709f, 0x0420, 0x15a9, 0x2732, 0x36bb,
      0xce4c, 0xdfc5, 0xed5e, 0xfcd7, 0x8868, 0x99e1, 0xab7a, 0xbaf3,
      0x5285, 0x430c, 0x7197, 0x601e, 0x14a1, 0x0528, 0x37b3, 0x263a,
      0xdecd, 0xcf44, 0xfddf, 0xec56, 0x98e9, 0x8960, 0xbbfb, 0xaa72,
      0x6306, 0x728f, 0x4014, 0x519d, 0x2522, 0x34ab, 0x0630, 0x17b9,
      0xef4e, 0xfec7, 0xcc5c, 0xddd5, 0xa96a, 0xb8e3, 0x8a78, 0x9bf1,
      0x7387, 0x620e, 0x5095, 0x411c, 0x35a3, 0x242a, 0x16b1, 0x0738,
      0xffcf, 0xee46, 0xdcdd, 0xcd54, 0xb9eb, 0xa862, 0x9af9, 0x8b70,
      0x8408, 0x9581, 0xa71a, 0xb693, 0xc22c, 0xd3a5, 0xe13e, 0xf0b7,
      0x0840, 0x19c9, 0x2b52, 0x3adb, 0x4e64, 0x5fed, 0x6d76, 0x7cff,
      0x9489, 0x8500, 0xb79b, 0xa612, 0xd2ad, 0xc324, 0xf1bf, 0xe036,
      0x18c1, 0x0948, 0x3bd3, 0x2a5a, 0x5ee5, 0x4f6c, 0x7df7, 0x6c7e,
      0xa50a, 0xb483, 0x8618, 0x9791, 0xe32e, 0xf2a7, 0xc03c, 0xd1b5,
      0x2942, 0x38cb, 0x0a50, 0x1bd9, 0x6f66, 0x7eef, 0x4c74, 0x5dfd,
      0xb58b, 0xa402, 0x9699, 0x8710, 0xf3af, 0xe226, 0xd0bd, 0xc134,
      0x39c3, 0x284a, 0x1ad1, 0x0b58, 0x7fe7, 0x6e6e, 0x5cf5, 0x4d7c,
      0xc60c, 0xd785, 0xe51e, 0xf497, 0x8028, 0x91a1, 0xa33a, 0xb2b3,
      0x4a44, 0x5bcd, 0x6956, 0x78df, 0x0c60, 0x1de9, 0x2f72, 0x3efb,
      0xd68d, 0xc704, 0xf59f, 0xe416, 0x90a9, 0x8120, 0xb3bb, 0xa232,
      0x5ac5, 0x4b4c, 0x79d7, 0x685e, 0x1ce1, 0x0d68, 0x3ff3, 0x2e7a,
      0xe70e, 0xf687, 0xc41c, 0xd595, 0xa12a, 0xb0a3, 0x8238, 0x93b1,
      0x6b46, 0x7acf, 0x4854, 0x59dd, 0x2d62, 0x3ceb, 0x0e70, 0x1ff9,
      0xf78f, 0xe606, 0xd49d, 0xc514, 0xb1ab, 0xa022, 0x92b9, 0x8330,
      0x7bc7, 0x6a4e, 0x58d5, 0x495c, 0x3de3, 0x2c6a, 0x1ef1, 0x0f78
   };
```

```
#define PPPINITFCS      0xffff  /* Initial FCS value */
#define PPPGOODFCS      0xf0b8  /* Good final FCS value */

/*
 * Calculate a new fcs given the current fcs and the new data.
 */
u16 pppfcs(fcs, cp, len)
    register u16 fcs;
    register unsigned char *cp;
    register int len;
{
    assert(sizeof (u16) == 2);
    assert(((u16) -1) > 0);
    while (len--)
        fcs = (fcs >> 8) ^ fcstab[(fcs ^ *cp++) & 0xff];

    return (fcs);
}

const char* getfield(char* line, int num)
{
   const char* tok;
   for (tok = strtok(line, ";");
         tok && *tok;
         tok = strtok(NULL, ";\n"))
   {
      if (!--num)
         return tok;
   }
   return NULL;
}



int main(int argc, char *argv[]){

   if (argc>2){
      puts(argv[1]);

      FILE *hw_crc_output_file = fopen( argv[1], "r" );
      if ( hw_crc_output_file == 0 )
      {
         printf( "Could not open hardware simulation file\n" );
         exit(1);
      }
      unsigned int hw_crc;
      char data_str[20],hw_crc_str[8],vlog_crc_str[8];

      int DATA_SIZE = 0;
      unsigned char data[10];

      char results_str[1000];
      ///Check counters, up to 10 bytes of data = 20 hex characters
      int hw_crc_vlog_crc_mismatch[10];
      int hw_crc_sw_crc_mismatch[10];
      int total_crc_lines_by_data_size[10];

      for (int i=0;i<10;i++){
         hw_crc_vlog_crc_mismatch[i]=0;
```

```
   hw_crc_sw_crc_mismatch[i]=0;
   total_crc_lines_by_data_size[i]=0;
}

//Ignore first line of input file
char c;
do{
  c = fgetc(hw_crc_output_file);
}
while (c != '\n');

char line[100];
while(fgets(line,100,hw_crc_output_file)){
   sscanf(line,"%[^;];%[^;];%[^;]",
      data_str,hw_crc_str,vlog_crc_str);

   ///---Parce Hardware and Verilog CRCs simulated----------------
   char hex_chars[2];
   DATA_SIZE = 0;
   for (int i=0;i<(strlen(data_str)/2);i++){
      hex_chars[0]=data_str[2*i];
      hex_chars[1]=data_str[2*i+1];
      unsigned int byte_buf =
         (unsigned int)strtol(hex_chars, NULL, 16);
      //printf("0x%x ",byte_buf);
      data[i]=byte_buf;
      DATA_SIZE+=1;
   }
   int hw_crc = (int)strtol(hw_crc_str,NULL,16);
   int vlog_crc = (int)strtol(vlog_crc_str,NULL,16);

   ///---Software CRC calculation---------------------------------

   //CRC Initial Value
   u16 fcs=0xffff;
   fcs=pppfcs(fcs,data,DATA_SIZE);

   //---Printing-------------------------------------------------


   //Fill with with spaces before writing data
   for (int i=0;i<20-2*DATA_SIZE;i++){
      printf(" ");
   }
   printf("0x%s",data_str);
   printf("; ");
   printf("HW_CRC: 0x%04x",hw_crc);
   printf("; ");
   printf("VLOG_CRC: 0x%04x",vlog_crc);
   printf("; ");
   printf("SW_CRC: 0x%04x",fcs);
   printf("; ");
   if (hw_crc==fcs && hw_crc==vlog_crc){
      printf("\x1B[32m CRC OK \x1B[0m \n");
   }
   else{
      printf("\x1B[31m !!!CRC FAIL!!! ");
      if (hw_crc!=fcs){
         printf("HW_CRC != SW_CRC ");
         hw_crc_sw_crc_mismatch[DATA_SIZE-1]+=1;
```

```
            }
            if (hw_crc!=vlog_crc){
               printf("HW_CRC != VLOG_CRC ");
               hw_crc_vlog_crc_mismatch[DATA_SIZE-1]+=1;
            }
            printf("\x1B[0m \n");
         }
         total_crc_lines_by_data_size[DATA_SIZE-1]+=1;
      }


      fclose(hw_crc_output_file);

      FILE *results_file = fopen( argv[2], "w" );
      if ( results_file == 0 )
      {
         printf( "Could not open results summary file\n" );
         exit(1);
      }
      fprintf(results_file,"%DATA_SIZE;DATA_COUNT;VLOG_MISMATCH;SW_MISMATCH\n");
      for (int i=0;i<10;i++){
         fprintf(results_file,"%2d;%5d;%5d;%5d\n",
            i+1,
            total_crc_lines_by_data_size[i],
            hw_crc_vlog_crc_mismatch[i],
            hw_crc_sw_crc_mismatch[i]
            );
      }
      fclose(results_file);

   }
   return 0;

}
```

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use ieee.math_real.all;

entity elink_tb is
end entity elink_tb;

library ieee;
use ieee.numeric_std.all;
architecture RTL of elink_tb is

   constant HEADER_FIELD   :  integer :=1;

   signal clk : std_logic;
   signal reset : std_logic;

   component fpga_elink
      generic(MASTER            : integer := 1;
              HEADER_FIELD      : integer := 1;
              ADDR_WIDTH        : integer := 5;
              MAX_PACKET_LENGTH : integer := 16);
      port(reset        : in  std_logic;
          user_clk     : out std_logic;
          cmd_busy     : out std_logic;
          rx_ena       : out std_logic;
          rx_dav       : in  std_logic;
          rx_sop       : out std_logic;
          rx_eop       : out std_logic;
          rx_cmd_test  : out std_logic;
          rx_cmd_reset : out std_logic;
          rx_cmd_ua    : out std_logic;
          rx_ns        : out std_logic_vector(2 downto 0);
          rx_adr       : out std_logic_vector(7 downto 0);
          rx_dat       : out std_logic_vector(15 downto 0);
          rx_nr        : out std_logic_vector(2 downto 0);
          rx_cmd_srej  : out std_logic_vector(6 downto 0);
          rx_err       : out std_logic;
          tx_ena       : in  std_logic;
          tx_dav       : out std_logic;
          tx_cmd_test  : in  std_logic;
          tx_cmd_reset : in  std_logic;
          tx_cmd_sabm  : in  std_logic;
          tx_ns        : out std_logic_vector(2 downto 0);
          tx_adr       : in  std_logic_vector(7 downto 0);
          tx_dat       : in  std_logic_vector(15 downto 0);
          tx_cset      : in  std_logic_vector(3 downto 0);
          tx_clk       : in  std_logic;
          tx_sd        : out std_logic_vector(1 downto 0);
          rx_clk       : in  std_logic;
          rx_sd        : in  std_logic_vector(1 downto 0));
   end component fpga_elink;
```

```
signal tx_dat_m : std_logic_vector(15 downto 0);
signal rx_dat_m : std_logic_vector(15 downto 0);
signal rx_sd_m : std_logic_vector(1 downto 0);
signal link_clk_m_to_s : std_logic;
signal link_clk_s_to_m : std_logic;
signal rx_dav_m : std_logic := '1';
signal tx_sd_m : std_logic_vector(1 downto 0);
signal rx_ena_m : std_logic;
signal rx_eop_m : std_logic;
signal tx_dav_m : std_logic;
signal rx_sop_m : std_logic;
signal rx_err_m : std_logic;
signal tx_ena_m : std_logic;
signal user_clk_m : std_logic;
signal tx_adr_m : std_logic_vector(7 downto 0);
signal rx_adr_m : std_logic_vector(7 downto 0);
signal tx_cmd_test_m : std_logic;
signal tx_cmd_reset_m : std_logic;
signal tx_cmd_sabm_m : std_logic;
signal rx_cmd_test_m : std_logic;
signal rx_cmd_reset_m : std_logic;
signal rx_cmd_ua_m : std_logic;
signal tx_ns_m : std_logic_vector(2 downto 0);
signal tx_cset_m : std_logic_vector(3 downto 0);
signal rx_nr_m : std_logic_vector(2 downto 0);
signal rx_ns_m : std_logic_vector(2 downto 0);
signal rx_cmd_srej_m : std_logic_vector(6 downto 0);
signal cmd_busy_m : std_logic;


signal tx_dat_s : std_logic_vector(15 downto 0);
signal rx_dat_s : std_logic_vector(15 downto 0);
signal rx_sd_s : std_logic_vector(1 downto 0);
signal link_clk_s : std_logic;
signal rx_dav_s : std_logic;
signal tx_sd_s : std_logic_vector(1 downto 0);
signal rx_ena_s : std_logic;
signal rx_eop_s : std_logic;
signal tx_dav_s : std_logic;
signal rx_sop_s : std_logic;
signal rx_err_s : std_logic;
signal tx_ena_s : std_logic;
signal user_clk_s : std_logic;
signal tx_adr_s : std_logic_vector(7 downto 0);
signal rx_adr_s : std_logic_vector(7 downto 0);
signal tx_cmd_test_s : std_logic;
signal tx_cmd_reset_s : std_logic;
signal tx_cmd_sabm_s : std_logic;
signal rx_cmd_test_s : std_logic;
signal rx_cmd_reset_s : std_logic;
signal rx_cmd_ua_s : std_logic;
signal tx_ns_s : std_logic_vector(2 downto 0);
signal tx_cset_s : std_logic_vector(3 downto 0);
signal rx_nr_s : std_logic_vector(2 downto 0);
signal rx_ns_s : std_logic_vector(2 downto 0);
signal rx_cmd_srej_s : std_logic_vector(6 downto 0);
signal cmd_busy_s : std_logic;
```

```
   constant clk_period : time := 25_000 ps;

   ---
   signal tx_length_s   : integer;

   signal stop : boolean:=false;
   signal reset_m : std_logic;
   signal reset_s : std_logic;


begin

   fpga_eport_m : component fpga_elink
      generic map(
         MASTER            => 0,
                HEADER_FIELD      => HEADER_FIELD,
                ADDR_WIDTH        => 5,
                MAX_PACKET_LENGTH => 16
      )
      port map(
         reset        => reset_m,
         user_clk     => user_clk_m,
         cmd_busy     => cmd_busy_m,
         rx_ena       => rx_ena_m,
         rx_dav       => rx_dav_m,
         rx_sop       => rx_sop_m,
         rx_eop       => rx_eop_m,
         rx_cmd_test  => rx_cmd_test_m,
         rx_cmd_reset => rx_cmd_reset_m,
         rx_cmd_ua    => rx_cmd_ua_m,
         rx_ns        => rx_ns_m,
         rx_adr       => rx_adr_m,
         rx_dat       => rx_dat_m,
         rx_nr        => rx_nr_m,
         rx_cmd_srej  => rx_cmd_srej_m,
         rx_err       => rx_err_m,
         tx_ena       => tx_ena_m,
         tx_dav       => tx_dav_m,
         tx_cmd_test  => tx_cmd_test_m,
         tx_cmd_reset => tx_cmd_reset_m,
         tx_cmd_sabm  => tx_cmd_sabm_m,
         tx_ns        => tx_ns_m,
         tx_adr       => tx_adr_m,
         tx_dat       => tx_dat_m,
         tx_cset      => tx_cset_m,
         tx_clk       => link_clk_m_to_s,
         tx_sd        => tx_sd_m,
         rx_clk       => link_clk_s_to_m,
         rx_sd        => rx_sd_m
      );


   fpga_eport_s : component fpga_elink
      generic map(
         MASTER            => 0,
                HEADER_FIELD      => HEADER_FIELD,
                ADDR_WIDTH        => 5,
                MAX_PACKET_LENGTH => 16)
      port map(
         reset        => reset_s,
```

```
      user_clk     => user_clk_s,
      cmd_busy     => cmd_busy_s,
      rx_ena       => rx_ena_s,
      rx_dav       => rx_dav_s,
      rx_sop       => rx_sop_s,
      rx_eop       => rx_eop_s,
      rx_cmd_test  => rx_cmd_test_s,
      rx_cmd_reset => rx_cmd_reset_s,
      rx_cmd_ua    => rx_cmd_ua_s,
      rx_ns        => rx_ns_s,
      rx_adr       => rx_adr_s,
      rx_dat       => rx_dat_s,
      rx_nr        => rx_nr_s,
      rx_cmd_srej  => rx_cmd_srej_s,
      rx_err       => rx_err_s,
      tx_ena       => rx_ena_s,--tx_ena_s,
      tx_dav       => tx_dav_s,
      tx_cmd_test  => tx_cmd_test_s,
      tx_cmd_reset => tx_cmd_reset_s,
      tx_cmd_sabm  => tx_cmd_sabm_s,
      tx_ns        => tx_ns_s,
      tx_adr       => tx_adr_s,
      tx_dat       => rx_dat_s,--tx_dat_s,
      tx_cset      => tx_cset_s,
      tx_clk       => link_clk_s_to_m,
      tx_sd        => tx_sd_s,
      rx_clk       => link_clk_m_to_s,
      rx_sd        => rx_sd_s
   );



tx_cset_m <= (others=>'0');
tx_cset_s <= (others=>'0');



rx_sd_m <= tx_sd_s;
link_clk_m_to_s <= clk;

-- Delayed link from master to slave
rx_sd_s <= tx_sd_m after 2 ns;
link_clk_s_to_m <= clk after 2 ns;

reset_m <= reset;
reset_s <= reset;



clk_stimulus : process is
begin
   while stop=false loop
      if clk='0' then clk<='1'; else clk<='0'; end if;
      wait for clk_period/2;
   end loop;
   wait;
end process clk_stimulus;
```

```
slave_time_stim : process is
begin
   tx_dat_s <= (others=>'0');
   tx_adr_s <= (others=>'0');
   tx_ena_s <= '0';
   tx_cmd_reset_s <= '0' after 5 ns;
   tx_cmd_test_s <= '0' after 5 ns;

   rx_dav_s <= '1';

   --file write  . . .
   tx_dat_s <= (others=>'1');
   tx_length_s <= 0;

   reset <= '0';
   wait for clk_period;
   reset <= '1';
   wait for 10 ns;
   wait for clk_period;
   reset <= '0';

   wait;
end process slave_time_stim;


master_time_stim : process is
   variable seed1, seed2: positive;               -- seed values for random generator
       variable rand: real;   -- random real-number value in range 0 to 1.0
       constant range_of_rand : real := 2.0**16;    -- the range of random values created will be 0 to
  +1000.
begin

   tx_ena_m <= '0';
   tx_cmd_reset_m <= '0';
   tx_cmd_test_m <= '0';
   tx_cmd_sabm_m <= '0';

   tx_cmd_sabm_s <= '0';
   tx_cmd_reset_s <= '0';
   tx_cmd_test_s <= '0';


   wait for clk_period;
   wait for 10 ns;
   wait for clk_period;

   tx_cmd_reset_m <= '1';
   wait for clk_period;
   tx_cmd_reset_m <= '0';
   wait for 32*clk_period;

   --tx_cmd_test_m <= '1';
   wait for clk_period;
   --tx_cmd_test_m <= '0';
   wait for 32*clk_period;
   tx_cmd_test_m <= '1';
   wait for clk_period;
   tx_cmd_test_m <= '0';
   wait for 32*clk_period;
```

```
      wait for 32*clk_period;
      tx_cmd_sabm_m <= '1';
      wait for clk_period;
      tx_cmd_sabm_m <= '0';
      wait for 32*clk_period;

      --tx_cmd_sabm_m <= '1';
      wait for clk_period;
      --tx_cmd_sabm_m <= '0';
      for j in 0 to 10 loop
         wait for 200*clk_period;

         tx_adr_m <= x"00";

         for i in 0 to 4 loop
            wait for clk_period;
            tx_ena_m <= '1';
            uniform(seed1, seed2, rand);   -- generate random number
            tx_dat_m <= std_logic_vector( to_unsigned(integer(rand*range_of_rand-1.0),16));  -- rescale to
    0..1000, convert integer part
         end loop;
         tx_ena_m <= '0';
         wait for clk_period;
      end loop;

      wait for 20000 ns;
      stop<=true;
      wait;

   end process master_time_stim;

time_stim : process is
   variable curr_time   :  time;
begin
   wait for clk_period/2;
   --curr_time := now;
   --if curr_time >= 20000 ns then
   -- stop<=true;
   if stop=true then
      wait;
   end if;
end process time_stim;


end architecture RTL;
```

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use ieee.math_real.all;

use work.Constant_Declaration.all;

use work.SCA_Package.all;

entity sol40_sca_tb is
end entity sol40_sca_tb;


--! This testbench is composed by a sol40_sca block as the GBT-SCA manager on the back-end
--! and some fpga_elinks as the E-Ports of the GBT-SCAs on the Front-End
architecture RTL of sol40_sca_tb is
   constant BASE_ADDRESS : std_logic_vector(ECS_address_size - 1 downto 0) := "100" & x"0000";
   constant period_40M     :  time := 25 ns;

   component sol40_sca
      generic(BASE_ADDRESS : std_logic_vector(ECS_address_size - 1 downto 0));
      port(
         ECS_CLK              : in  std_logic;
         ECS_ADDRESS_i        : in  std_logic_vector(ECS_address_size - 1 downto 0);
         READ_ECS_RESPONSE_i  : in  std_logic;
         ECS_RESPONSE_o       : out std_logic_vector(31 downto 0);
         ECS_RESPONSE_VALID_o : out std_logic;
         ECS_COMMAND_i        : in  std_logic_vector(31 downto 0);
         WRITE_ECS_COMMAND_i  : in  std_logic;
         txClock40            : in  std_logic;
         rxClock40            : in  std_logic;
         SRES_i               : in  std_logic;
         tx_sd                : out elink_ch_array_t(0 to SCA_COUNT - 1);
         rx_sd                : in  elink_ch_array_t(0 to SCA_COUNT - 1)
      );
   end component sol40_sca;


   component fpga_elink
      generic(MASTER            : integer := 1;
              HEADER_FIELD      : integer := 1;
              ADDR_WIDTH        : integer := 5;
              MAX_PACKET_LENGTH : integer := 16);
      port(reset        : in  std_logic;
          user_clk     : out std_logic;
          cmd_busy     : out std_logic;
          rx_ena       : out std_logic;
          rx_dav       : in  std_logic;
          rx_sop       : out std_logic;
          rx_eop       : out std_logic;
          rx_cmd_test  : out std_logic;
          rx_cmd_reset : out std_logic;
          rx_cmd_ua    : out std_logic;
          rx_ns        : out std_logic_vector(2 downto 0);
          rx_adr       : out std_logic_vector(7 downto 0);
          rx_dat       : out std_logic_vector(15 downto 0);
          rx_nr        : out std_logic_vector(2 downto 0);
          rx_cmd_srej  : out std_logic_vector(6 downto 0);
          rx_err       : out std_logic;
```

```
        tx_ena       : in  std_logic;
        tx_dav       : out std_logic;
        tx_cmd_test  : in  std_logic;
        tx_cmd_reset : in  std_logic;
        tx_cmd_sabm  : in  std_logic;
        tx_ns        : out std_logic_vector(2 downto 0);
        tx_adr       : in  std_logic_vector(7 downto 0);
        tx_dat       : in  std_logic_vector(15 downto 0);
        tx_cset      : in  std_logic_vector(3 downto 0);
        tx_clk       : in  std_logic;
        tx_sd        : out std_logic_vector(1 downto 0);
        rx_clk       : in  std_logic;
        rx_sd        : in  std_logic_vector(1 downto 0));
end component fpga_elink;



signal address_MM_slave_ECS      :
   std_logic_vector(ECS_address_size - 1 downto 0) := BASE_ADDRESS;
signal read_MM_slave_ECS         :  std_logic;
signal readdata_MM_slave_ECS     :
   std_logic_vector(31 downto 0);
signal readdatavalid_MM_slave_ECS   :  std_logic;
signal writedata_MM_slave_ECS       :
   std_logic_vector(31 downto 0);
signal write_MM_slave_ECS        :  std_logic;
signal clock_MM_slave_ECS        :  std_logic;
signal reset_MM_slave_ECS        :  std_logic;
signal tx_sd                  :  elink_ch_array_t(0 to SCA_COUNT - 1);
signal rx_sd                  :  elink_ch_array_t(0 to SCA_COUNT - 1);

signal clk,rst                :  std_logic := '0';
signal cmd_busy_slave            :  std_logic_vector(0 to SCA_COUNT-1);
signal rx_ena_slave           :  std_logic_vector(0 to SCA_COUNT-1);



signal rx_dav_slave :   std_logic_vector(0 to SCA_COUNT-1) := (others=>'1');
signal rx_sop_slave :   std_logic_vector(0 to SCA_COUNT-1);
signal rx_cmd_test_slave : std_logic_vector(0 to SCA_COUNT-1);
signal rx_eop_slave : std_logic_vector(0 to SCA_COUNT-1);
signal rx_cmd_reset_slave : std_logic_vector(0 to SCA_COUNT-1);
signal rx_cmd_ua_slave : std_logic_vector(0 to SCA_COUNT-1);
signal rx_dat_slave : eport_data_array_t(0 to SCA_COUNT-1);
signal rx_cmd_srej_slave : std_logic_vector(0 to SCA_COUNT-1);
signal rx_err_slave : std_logic_vector(0 to SCA_COUNT-1);
signal tx_ena_slave : std_logic_vector(0 to SCA_COUNT-1) := (others=>'0');
signal tx_dav_slave : std_logic_vector(0 to SCA_COUNT-1);
--signal tx_cmd_test_slave : std_logic_vector(0 to SCA_COUNT-1);
--signal tx_cmd_reset_slave : std_logic_vector(0 to SCA_COUNT-1);
--signal tx_cmd_sabm_slave : std_logic_vector(0 to SCA_COUNT-1);
signal tx_dat_slave : eport_data_array_t(0 to SCA_COUNT-1) := (others=>(others=>'0'));


type slave_state_t is (IDLE, RCV, SND);
signal slave_state : slave_state_t := IDLE;

signal counter_slave, tx_counter_slave :  natural range 0 to 3;
```

```
   procedure test(   val    :  in std_logic;
               signal out_val :  out   std_logic
   ) is
begin
   out_val <= val;
   wait for period_40M;
   out_val <= not val;


end procedure test;


   --Format of a ECS  Packet:

   --|          | GBT-SCA# | SCA-CH#   | ECS CMD |
   --|        ECS-LEN      |  Protocol Specific |
   --|        DATA[0]      |       DATA[1]      |
   --+---------------------+--------------------+
   --|------------------- 32b-------------------|

   procedure write_ecs_i2c(
      sca_nr      :  in natural range 0 to SCA_COUNT -1;
      sca_ch_nr   :  in sca_ch_enum;
      i2c_cmd     :  in ecs_i2c_cmd_enum;
      data_len :  in natural range 0   to (128/8) - 1;
      prot_spec   :  in std_logic_vector(15 downto 0);
      data     :  in std_logic_vector(127 downto 0);
      signal we         :  out   std_logic;
      signal wdata      :  out   std_logic_vector(31 downto 0);
      signal addr :  out std_logic_vector(ECS_address_size-1 downto 0)
   ) is
      variable i  : natural range 0 to  natural(ceil(real(data_len)/real(32)))+2 ;
      variable addr_var : std_logic_vector(ECS_address_size-1 downto 0);
   begin
      addr_var := BASE_ADDRESS;
      i :=0;
      we <= '1';
      while i <  (natural(ceil(real(data_len)/real(32)))+2) loop

         case i is
         when 0=>
            addr_var(7 downto 0) := ECS_COMMAND_HEADER_0;
            addr <= addr_var;
            wdata <= x"00"
                  & std_logic_vector(to_unsigned(sca_nr,8))
                  & sca_ch_table(sca_ch_nr)
                  & ecs_i2c_table(i2c_cmd);
         when 1=>
            addr_var(7 downto 0) := ECS_COMMAND_HEADER_1;
            addr <= addr_var;
            wdata <= std_logic_vector(to_unsigned(data_len,16))
                  &  prot_spec;
         when others=>
            if i=2 then
               addr_var(7 downto 0) := ECS_COMMAND_DATA_0;
            elsif i=3 then
               addr_var(7 downto 0) := ECS_COMMAND_DATA_1;
            elsif i=4 then
               addr_var(7 downto 0) := ECS_COMMAND_DATA_2;
            else
```

```
            addr_var(7 downto 0) := ECS_COMMAND_DATA_3;
         end if;
         addr <= addr_var;
         wdata <= data(32*(i-1)-1 downto 32*(i-2));
      end case;

      wait for period_40M;

      i := i+1;
   end loop;
   we <= '0';
   wait for period_40M;
   we <= '1';
   addr_var(7 downto 0) := ECS_CONTROL_REGISTER;
   addr <= addr_var;
   wait for period_40M;
   we <= '0';
   report "function write_ecs_i2c was called with the following parameters:\n"
         & " sca_nr = " & natural'image(sca_nr) & "\n"
         & "   sca_ch_nr = " & sca_ch_enum'image(sca_ch_nr) & " i2c_cmd = " &
  ecs_i2c_cmd_enum'image(i2c_cmd) & "\n"
         & "   data_len = " & integer'image(data_len);
end procedure write_ecs_i2c;

procedure write_sca_ctrl_reg(
   sca_nr      :  in natural range 0 to SCA_COUNT -1;
   sca_cmd     :  in ecs_sca_cmd_enum;
   val         :  in byte_t;
   signal we   :  out   std_logic;
   signal wdata:  out   std_logic_vector(31 downto 0);
   signal addr :  out std_logic_vector(ECS_address_size-1 downto 0)
) is
   variable addr_var : std_logic_vector(ECS_address_size-1 downto 0);
begin
   addr_var := BASE_ADDRESS;
   we <= '1';
   addr_var(7 downto 0) := ECS_COMMAND_HEADER_0;
         addr <= addr_var;
   wdata <= x"00"
         & std_logic_vector(to_unsigned(sca_nr,8))
         & sca_ch_table(sca_controller)
         & ecs_sca_cmd_table(sca_cmd);
   wait for period_40M;
   addr_var(7 downto 0) := ECS_COMMAND_HEADER_1;
         addr <= addr_var;
   wdata <= x"00"
         & std_logic_vector(to_unsigned(1,8))
         & x"00"
         & x"00";
   wait for period_40M;
   addr_var(7 downto 0) := ECS_COMMAND_DATA_0;
         addr <= addr_var;
   wdata <= x"0000" & x"00" & val;
   wait for period_40M;
   we <= '0';
   wait for period_40M;
   we <= '1';
   addr_var(7 downto 0) := ECS_CONTROL_REGISTER;
   addr <= addr_var;
   wait for period_40M;
```

```
      we <= '0';
      report "function write_sca_ctrl_reg was called with the following parameters:\n"
            & " sca_nr = " & natural'image(sca_nr) & "\n"
            & "   sca_ch_nr = " & sca_ch_enum'image(sca_controller) & "\n";
            --& " sca_cmd = " & ecs_sca_cmd_table'image(sca_cmd);

   end procedure write_sca_ctrl_reg;

   procedure write_ecs_spi is

   begin

   end procedure write_ecs_spi;

   procedure write_ecs_adc is

   begin

   end procedure write_ecs_adc;

   procedure read_ecs_adc is

   begin

   end procedure read_ecs_adc;


   signal user_clk_slave : std_logic_vector(0 to SCA_COUNT - 1);

   signal user_clk_slave_ant : std_logic_vector(0 to SCA_COUNT - 1);

begin
   clock_MM_slave_ECS <= clk;
   reset_MM_slave_ECS <= rst;


   sol40_sca_inst : component sol40_sca
      generic map(
         BASE_ADDRESS => BASE_ADDRESS
      )
      port map(
         ECS_CLK              => clock_MM_slave_ECS,
         ECS_ADDRESS_i        => address_MM_slave_ECS,
         READ_ECS_RESPONSE_i  => read_MM_slave_ECS,
         ECS_RESPONSE_o       => readdata_MM_slave_ECS,
         ECS_RESPONSE_VALID_o => readdatavalid_MM_slave_ECS,
         ECS_COMMAND_i        => writedata_MM_slave_ECS,
         WRITE_ECS_COMMAND_i  => write_MM_slave_ECS,
         txClock40            => clock_MM_slave_ECS,
         rxClock40            => clock_MM_slave_ECS,
         SRES_i               => reset_MM_slave_ECS,
         tx_sd                => tx_sd,
         rx_sd                => rx_sd
      );

   gen_sca_eports : for i in 0 to SCA_COUNT - 1  generate
      fpga_elink_inst : fpga_elink
         generic map(MASTER            => 0,
                     HEADER_FIELD      => 1,
                     ADDR_WIDTH        => 5,
```

```
                    MAX_PACKET_LENGTH => 16)
         port map(
               reset        => rst,
                 user_clk     => user_clk_slave(i),
                 cmd_busy     => cmd_busy_slave(i),
                 rx_ena       => rx_ena_slave(i),
                 rx_dav       => rx_dav_slave(i),
                 rx_sop       => rx_sop_slave(i),
                 rx_eop       => rx_eop_slave(i),
                 rx_cmd_test  => rx_cmd_test_slave(i),
                 rx_cmd_reset => rx_cmd_reset_slave(i),
                 rx_cmd_ua    => rx_cmd_ua_slave(i),
                 rx_ns        => open,
                 rx_adr       => open,
                 rx_dat       => rx_dat_slave(i),
                 rx_nr        => open,
                 rx_cmd_srej  => open,
                 rx_err       => open,
                tx_ena       => tx_ena_slave(i),
                 tx_dav       => tx_dav_slave(i),
                 tx_cmd_test  => '0',
                 tx_cmd_reset => '0',
                 tx_cmd_sabm  => '0',
                 tx_ns        => open,
                 tx_adr       => x"88",
                 tx_dat       => tx_dat_slave(i),
                 tx_cset      => (others=>'0'),
                 tx_clk     => clk,
                 tx_sd        => rx_sd(i),
                 rx_clk => clk,
                 rx_sd        => tx_sd(i));


   end generate gen_sca_eports;



   clk_stimulus : process is
   begin
      if clk='0' then clk<='1'; else clk<='0'; end if;
      wait for period_40M/2;
   end process clk_stimulus;



   stimulus : process is
   begin

      rst <= '1';
      read_MM_slave_ECS <= '0';
      write_MM_slave_ECS <= '0';

      --setup of the slave eport (GBT-SCA side)
      for i in 0 to SCA_COUNT -1  loop
--       tx_ena_slave(i) <= '0';
         rx_dav_slave(i) <= '1';
      end loop;
```

```
      wait for period_40M;
      rst <= '0';

      wait for period_40M;

      write_sca_ctrl_reg(0, ECS_CRC_wrt, x"45",
         write_MM_slave_ECS,
         writedata_MM_slave_ECS,
         address_MM_slave_ECS);

--    write_ecs_i2c(
--       0,
--       master_i2c_1,
--       ECS_I2C_WRITE,
--       1,
--        x"8656",
--       (others=>'0'),
--       write_MM_slave_ECS,
--       writedata_MM_slave_ECS,
--       address_MM_slave_ECS
--    );
--
--
--
--    write_sca_ctrl_reg(3, ECS_CRA_wrt, x"89",
--       write_MM_slave_ECS,
--       writedata_MM_slave_ECS,
--       address_MM_slave_ECS);
--
--    write_sca_ctrl_reg(5, ECS_CRA_wrt, x"23",
--       write_MM_slave_ECS,
--       writedata_MM_slave_ECS,
--       address_MM_slave_ECS);
--
--    write_sca_ctrl_reg(15, ECS_CRA_wrt, x"AB",
--       write_MM_slave_ECS,
--       writedata_MM_slave_ECS,
--       address_MM_slave_ECS);
--
--    write_ecs_i2c(
--       5,
--       master_i2c_5,
--       ECS_I2C_WRITE,
--       1,
--        x"2323",
--       (others=>'0'),
--       write_MM_slave_ECS,
--       writedata_MM_slave_ECS,
--       address_MM_slave_ECS
--    );
--
--    write_ecs_i2c(
--       15,
--       master_i2c_7,
--       ECS_I2C_WRITE,
--       5,
--        x"34CD",
```

```
--       (others=>'1'),
--       write_MM_slave_ECS,
--       writedata_MM_slave_ECS,
--       address_MM_slave_ECS
--    );
--
--    write_ecs_i2c(
--       0,
--       master_i2c_5,
--       ECS_I2C_WRITE,
--       1,
--        x"7878",
--       (others=>'0'),
--       write_MM_slave_ECS,
--       writedata_MM_slave_ECS,
--       address_MM_slave_ECS
--    );
--
--
--    write_ecs_i2c(
--       0,
--       master_i2c_8,
--       ECS_I2C_WRITE,
--       1,
--        x"9797",
--       (others=>'0'),
--       write_MM_slave_ECS,
--       writedata_MM_slave_ECS,
--       address_MM_slave_ECS
--    );


      wait;


   end process stimulus;



   -- \TODO THIS PROCESS/PROCESS IS HORRIBLE
   slave_process : process(user_clk_slave) is
      variable rpy_data_from_rx_receiver: payload_t;
   begin

      if user_clk_slave'event then
       for i in 0 to user_clk_slave'high loop
         if user_clk_slave_ant(i)='0' and user_clk_slave(i) = '1' then
            if rx_ena_slave(i)='1' then
               case counter_slave is
               when 0=>
                  rpy_data_from_rx_receiver.tr := rx_dat_slave(i)(15 downto 8);
                  rpy_data_from_rx_receiver.ch := rx_dat_slave(i)(7 downto 0);
                  tx_ena_slave(i) <= '1';
                  tx_dat_slave(i) <=  rx_dat_slave(i);--rpy_data_from_rx_receiver.tr &
    rpy_data_from_rx_receiver.ch;
               when 1=>
                  rpy_data_from_rx_receiver.cmd_or_err := rx_dat_slave(i)(15 downto 8);
                  rpy_data_from_rx_receiver.len := rx_dat_slave(i)(7 downto 0);
                  tx_ena_slave(i) <= '1';
                  tx_dat_slave(i) <= (others=>'0');
```

```
                --31 downto 16 = data0
            when 2=>
                rpy_data_from_rx_receiver.data(31 downto 16) := rx_dat_slave(0);
                tx_ena_slave(i) <= '0';
                tx_dat_slave(i) <= (others=>'0');
            when 3 =>
                --15 downto 0 = data1
                rpy_data_from_rx_receiver.data(15 downto 0) := rx_dat_slave(0);
                tx_ena_slave(i) <= '0';
                tx_dat_slave(i) <= (others=>'0');
            when others=>
                tx_ena_slave(i) <= '0';
                tx_dat_slave(i) <= (others=>'0');
            end case;
            counter_slave <= counter_slave + 1;
         else
            tx_ena_slave(i) <= '0';
            tx_dat_slave(i) <= (others=>'0');
            counter_slave <= 0;
         end if;

--          if counter_slave/=3 and rx_eop_slave(0)='0' then
--             counter_slave <= counter_slave + 1;
--          elsif rx_eop_slave(0) = '1' then
--             --rx_dav_slave(0) <= '0';
--             counter_slave <= 0;
--          else -- rx_eop/='1' then
--             report "SCA MAC Layer - eport_driver : Receiving Packet is too long";
--          end if;
--          slave_state <= RCV;
--       elsif rx_ena_slave_ant(0) = '1' and rx_ena_slave(0) = '0' then
--          tx_counter_slave <= 0;
--          slave_state <= SND;
--       elsif slave_state = SND then
--          if tx_dav_slave(0)='1' or tx_ena_slave(0) <= '1'  then
--             case counter_slave is
--             when 0=>
--                tx_dat_slave(0) <= rpy_data_from_rx_receiver.tr & rpy_data_from_rx_receiver.ch;
--                tx_ena_slave(0) <= '1';
--                counter_slave <= counter_slave +1;
--             when 1 =>
--                --SCA Error field + SCA Len
--                tx_dat_slave(0) <= x"00" & x"00";
--                counter_slave <= counter_slave +1;
--             when others =>
--                tx_ena_slave(0) <= '0';
--                counter_slave <= 0;
--                slave_state <= IDLE;
--             end case;
--          end if;
--       end if;
      end if;
   end loop;
   user_clk_slave_ant <= user_clk_slave;
   end if;


end process slave_process;
```

```
end architecture RTL;
```

B.3 Arquivos de Teste

```
-- sol40_sca_avalon.vhd

-- Generated using ACDS version 13.1 162 at 2015.06.12.15:30:33

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity sol40_sca_avalon is
   port (
      clk_clk                       : in  std_logic                     := '0';             --
          clk.clk
      master_0_master_reset_reset   : out std_logic;                                        --
     master_0_master_reset.reset
      master_0_master_address       : out std_logic_vector(31 downto 0);                    --
     master_0_master.address
      master_0_master_readdata      : in  std_logic_vector(31 downto 0) := (others => '0'); --
              .readdata
      master_0_master_read          : out std_logic;                                        --
              .read
      master_0_master_write         : out std_logic;                                        --
              .write
      master_0_master_writedata     : out std_logic_vector(31 downto 0);                    --
              .writedata
      master_0_master_waitrequest   : in  std_logic                     := '0';             --
              .waitrequest
      master_0_master_readdatavalid : in  std_logic                     := '0';             --
              .readdatavalid
      master_0_master_byteenable    : out std_logic_vector(3 downto 0);                     --
              .byteenable
      reset_reset_n                 : in  std_logic                     := '0'              --
         reset.reset_n
   );
end entity sol40_sca_avalon;

architecture rtl of sol40_sca_avalon is
   component sol40_sca_avalon_master_0 is
      generic (
         USE_PLI     : integer := 0;
         PLI_PORT    : integer := 50000;
         FIFO_DEPTHS : integer := 2
      );
      port (
         clk_clk              : in  std_logic                     := 'X';             -- clk
         clk_reset_reset      : in  std_logic                     := 'X';             -- reset
         master_address       : out std_logic_vector(31 downto 0);                    -- address
         master_readdata      : in  std_logic_vector(31 downto 0) := (others => 'X'); -- readdata
         master_read          : out std_logic;                                        -- read
         master_write         : out std_logic;                                        -- write
         master_writedata     : out std_logic_vector(31 downto 0);                    -- writedata
         master_waitrequest   : in  std_logic                     := 'X';             -- waitrequest
         master_readdatavalid : in  std_logic                     := 'X';             -- readdatavalid
```

```
         master_byteenable    : out std_logic_vector(3 downto 0);                    -- byteenable
         master_reset_reset   : out std_logic                                        -- reset
      );
   end component sol40_sca_avalon_master_0;

   signal reset_reset_n_ports_inv : std_logic; -- reset_reset_n:inv -> master_0:clk_reset_reset

begin

   master_0 : component sol40_sca_avalon_master_0
      generic map (
         USE_PLI     => 0,
         PLI_PORT    => 50000,
         FIFO_DEPTHS => 2
      )
      port map (
         clk_clk              => clk_clk,                       --          clk.clk
         clk_reset_reset      => reset_reset_n_ports_inv,       --    clk_reset.reset
         master_address       => master_0_master_address,       --       master.address
         master_readdata      => master_0_master_readdata,      --             .readdata
         master_read          => master_0_master_read,          --             .read
         master_write         => master_0_master_write,         --             .write
         master_writedata     => master_0_master_writedata,     --             .writedata
         master_waitrequest   => master_0_master_waitrequest,   --             .waitrequest
         master_readdatavalid => master_0_master_readdatavalid, --             .readdatavalid
         master_byteenable    => master_0_master_byteenable,    --             .byteenable
         master_reset_reset   => master_0_master_reset_reset    -- master_reset.reset
      );

   reset_reset_n_ports_inv <= not reset_reset_n;

end architecture rtl; -- of sol40_sca_avalon
```

```
--------------------------------------------------------------------------------
--! @file sol40_sca_debugger.vhd
--! @author Cairo Caplan <cairo.caplan@cern.ch>
--! @date April, 2016
--------------------------------------------------------------------------------

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

use work.SCA_Package.all;
use work.Constant_Declaration.all;

entity sol40_sca_debugger is
   port (
   txClock40                  :  in std_logic;
   rxClock40                  :  in std_logic;
   rst                        :  in std_logic;
   --
   SC_EC_SC_EC_transmittoSCA       :   out   std_logic_vector(1 downto 0);
   SC_EC_SC_EC_receivedfromSCA     :   in std_logic_vector(1 downto 0)

   );
end entity sol40_sca_debugger;

architecture RTL of sol40_sca_debugger is

   component sol40_sca
      generic(BASE_ADDRESS : std_logic_vector(ECS_address_size - 1 downto 0));
      port(
         ECS_CLK              : in  std_logic;
         ECS_ADDRESS_i        : in  std_logic_vector(ECS_address_size - 1 downto 0);
         READ_ECS_RESPONSE_i  : in  std_logic;
         ECS_RESPONSE_o       : out std_logic_vector(31 downto 0);
         ECS_RESPONSE_VALID_o : out std_logic;
         ECS_COMMAND_i        : in  std_logic_vector(31 downto 0);
         WRITE_ECS_COMMAND_i  : in  std_logic;
         txClock40            : in  std_logic;
         rxClock40            : in  std_logic;
         SRES_i               : in  std_logic;
         tx_sd                : out elink_ch_array_t(0 to SCA_COUNT - 1);
         rx_sd                : in  elink_ch_array_t(0 to SCA_COUNT - 1)
      );
   end component sol40_sca;

   component sol40_sca_avalon
      port(clk_clk                       : in  std_logic                     := '0';
          master_0_master_reset_reset   : out std_logic;
          master_0_master_address       : out std_logic_vector(31 downto 0);
          master_0_master_readdata      : in  std_logic_vector(31 downto 0) := (others => '0');
          master_0_master_read          : out std_logic;
          master_0_master_write         : out std_logic;
          master_0_master_writedata     : out std_logic_vector(31 downto 0);
          master_0_master_waitrequest   : in  std_logic                     := '0';
          master_0_master_readdatavalid : in  std_logic                     := '0';
          master_0_master_byteenable    : out std_logic_vector(3 downto 0);
          reset_reset_n                 : in  std_logic                     := '0');
   end component sol40_sca_avalon;
```

```
   signal SC_EC_reset         : std_logic := '0';
   signal SC_EC_reset_signal  : std_logic;

   signal reset_reset_n_int   :  std_logic := '1';


   signal tx_sd : elink_ch_array_t(0 to SCA_COUNT - 1);
   signal rx_sd : elink_ch_array_t(0 to SCA_COUNT - 1);

   constant BASE_ADDRESS   :  std_logic_vector(ECS_address_size-1 downto 0):="10"& '1'&X"0" &
     X"400";--0x50400



   signal master_0_master_reset_reset : std_logic;
   signal master_0_master_address : std_logic_vector(31 downto 0);
   signal master_0_master_readdata : std_logic_vector(31 downto 0);
   signal master_0_master_read : std_logic;
   signal master_0_master_writedata : std_logic_vector(31 downto 0);
   signal master_0_master_write : std_logic;
   signal master_0_master_waitrequest : std_logic;
   signal master_0_master_readdatavalid : std_logic;
   signal master_0_master_byteenable : std_logic_vector(3 downto 0);
   signal reset_reset_n : std_logic;
   signal ECS_CLK : std_logic;
begin

   ecs_clk_gen : process (txClock40) is
   begin
      if rising_edge(txClock40) then
         if ECS_CLK='1' then
            ECS_CLK <= '0';
         else
            ECS_CLK <= '1';
         end if;
      end if;
   end process ecs_clk_gen;


   reset_reset_n_int <= SC_EC_reset_signal;
   sol40_sca_avalon_inst : component sol40_sca_avalon
      port map(
         clk_clk                      => txClock40,
         master_0_master_reset_reset  => master_0_master_reset_reset,
         master_0_master_address      => master_0_master_address,
         master_0_master_readdata     => master_0_master_readdata,
         master_0_master_read         => master_0_master_read,
         master_0_master_write        => master_0_master_write,
         master_0_master_writedata    => master_0_master_writedata,
         master_0_master_waitrequest  => '0',
         master_0_master_readdatavalid => master_0_master_readdatavalid,
         master_0_master_byteenable   => open,
         reset_reset_n                => reset_reset_n_int
      );

   sol40_sca_inst : component sol40_sca
      generic map(
         BASE_ADDRESS => BASE_ADDRESS
      )
      port map(
```

```
        ECS_CLK  => txClock40,
        ECS_ADDRESS_i        => master_0_master_address(ECS_address_size-1 downto 0),
        READ_ECS_RESPONSE_i  => master_0_master_read,
        ECS_RESPONSE_o       => master_0_master_readdata,
        ECS_RESPONSE_VALID_o => master_0_master_readdatavalid,
        ECS_COMMAND_i        => master_0_master_writedata,
        WRITE_ECS_COMMAND_i  => master_0_master_write,
        txClock40            => txClock40,
        rxClock40            => rxClock40,
        SRES_i               => master_0_master_reset_reset,
        tx_sd                => tx_sd,
        rx_sd                => rx_sd
    );


  SC_EC_SC_EC_transmittoSCA <= tx_sd(0);
  rx_sd(0) <= SC_EC_SC_EC_receivedfromSCA;

  SC_EC_reset_signal          <= rst or SC_EC_reset;


end architecture RTL;
```

```
# ------------------------------------------------------------------
# sol40_sca_eval_tool.tcl
#
# 17/07/2015 Cairo Caplan <cairo.caplan@cern.ch>
#
# Based on jtag_client.tcl tool by :
# 14/09/2011 D. W. Hawkins (dwh@ovro.caltech.edu)
#
# Altera JTAG socket client Tcl/Tk GUI to evaluate a stand-alone SOL40_SCA
# instance to control many GBT-SCA's and their devices.
#
# The client application can run from any Tcl/Tk shell, eg.,
# the following were tested under Windows
#
#  * wish (ActiveState ActiveTcl 8.4.16.0)
#  * quartus_stp (Quartus II 10.1)
#  * Modelsim-SE 6.5b
#  * The Quartus II Tcl console (Quartus II 10.1)
#
# The client does not work for;
#
#  * SystemConsole
#    Altera has somehow managed to break Tk support there
#
#  * Modelsim Altera-Edition (Modelsim-AE)
#    The 'free' version does not support Tk.
#
# ------------------------------------------------------------------
# Notes:
# ------
#
# 1. Command line operation
#
#    quartus_stp -t sol40_sca_eval_tool.tcl
#
# 2. Console operation
#
#    quartus_stp -s
#    tcl> source sol40_sca_eval_tool.tcl
#
# ------------------------------------------------------------------
# References
# ----------
# 1. Wheeler, Bert. Tcl/Tk 8.5 Programming Cookbook. Packt Publishing Ltd, 2011.
#
# 2. Brent Welch, "Practical Programming in Tcl and Tk",
#    3rd Ed, 2000.
#
# ------------------------------------------------------------------
package require BWidget

source ./jtag_client_cmds.tcl

proc detect_tool {} {
   global jtag argv0

   # Get the tool name
   set toolname [info nameofexecutable]
   if {[string length $toolname] == 0} {
```

```
      if {[info exists argv0]} {
         set toolname $argv0
      }
   }

   # Strip the name to just that of the application
   set toolname [file rootname [file tail $toolname]]

   # Example toolname strings;
   #
   #  ------------------------------------------------
   # | Application                  | String         |
   # |------------------------------|----------------|
   # |                              |                |
   # | Quartus II                   | quartus        |
   # | SystemConsole                | system-console |
   # | quartus_stp                  | quartus_stp    |
   # | Modelsim                     | vish           |
   # | ActiveState ActiveTcl Wish84 | wish84         |
   # |                              |                |
   #  ------------------------------------------------
   #
   set jtag(tool_ok) 0
   if {[string first wish $toolname] == 0} {
      # Toolname starts with 'wish'
      set jtag(tool) wish
      set jtag(tool_ok) 1
   } elseif {[string first vish $toolname] == 0} {
      # Modelsim starts vish.exe
      # (alternatively argv0 ends in vsim)
      #
      # Note:
      # Modelsim-Altera does not have Tk support
      # (package require Tk fails)
      #
      set jtag(tool) modelsim
      set jtag(tool_ok) 1
   } elseif {[string first quartus $toolname] == 0} {
      # Quartus also has a global called quartus
      # that can be detected using [info exists quartus].
      set jtag(tool) $toolname
      set jtag(tool_ok) 1
   }
   return
}

proc is_tool_ok {} {
   global jtag
   if {![info exists jtag(tool_ok)]} {
      detect_tool
   }
   return $jtag(tool_ok)
}

# Console message
detect_tool
if {![is_tool_ok]} {
   puts "Sorry, this script can only run using a tool with Tk support"
   return
}
```

```
puts [format " \nJTAG client running under %s\n " $jtag(tool)]

# ------------------------------------------------------------------
# Initialize Tk
# ------------------------------------------------------------------
#
# Check for the Tk package (fails for Modelsim-AE)
if {[catch {package require Tk}]} {

   # If Tk is not available, generate an error to the console
   error [concat \
      "Error: the package 'Tk' was not found. "\
      "Please use an application that supports Tcl/Tk." ]

}

# Quartus shells need to initialize Tk
if {[string compare $jtag(tool) quartus] == 0} {
   init_tk
}

# ------------------------------------------------------------------
# Command line argument?
# ------------------------------------------------------------------
#
# The Modelsim-SE GUI starts with an argc of 1 with an argv
# of '-gui'. So check the command line arguments are numbers
# before using them.
#
# Server port number?
if {$argc > 0} {
   set port [lindex $argv 0]
   if {![string is digit $port]} {
      unset port
   }
}
if {![info exists port]} {
   set port 2540
}
set jtag(port) $port
unset port

# ==================================================================
# Tk GUI
# ==================================================================
#
#------------------------------------------------------------------
# GUI helper procedures
#------------------------------------------------------------------
#
# Wish console show/hide
proc console_update {} {
   global jtag
   if {![info exists jtag(console)]} {
      return
   }
   if {$jtag(console) == 1} {
      puts "<console_update> show"
      console show
   } else {
```

```
      puts "<console_update> hide"
      console hide
   }
}

# Tool-specific exit sequence
# * 'quit' is a command in Modelsim, so use a different name
proc gui_quit {win} {
   global jtag
   if {[info exists jtag(socket)]} {
      client_close
   }
   puts "Quit"

    if {(([string compare $jtag(tool) wish] == 0) && ($jtag(console) == 1)) ||
         ([string compare $jtag(tool) modelsim] == 0) ||
         ([string first quartus $jtag(tool)] == 0)} {

         # File->Exit has a $win value corresponding to
         # the path to the menu element, eg. .topX.mbar.file.menu,
         # whereas exit using the window manager 'x' button
         # generates the $win value .topX.
         #
         # Only destroy the main window
         set win ".[lindex [split $win .] 1]"
         destroy $win
         return
   }
   # Otherwise exit (eg. wish, with console not visible)
   exit
}




# if [expr {[llength [info commands init_tk]] > 0}] {
   # init_tk
   # puts "lol"
# }


# . configure  -takefocus 0

#namespace eval sol40_sca_eval_tool {

proc socket_trace {name1 name2 op} {
   global jtag

#   name1 = jtag, name2 = socket, op = w or u
#  puts "socket trace: $name1 $name2 $op"

   switch $op {
      w {
         set jtag(port_status) "Connected"
         $jtag(port_status_label)  configure -bg green
         $jtag(port_status_button) configure -text "Disconnect"
      }
      u {
         set jtag(port_status) "Disconnected"
```

```
            $jtag(port_status_label)  configure -bg yellow
            $jtag(port_status_button) configure -text "Connect"

            # Since jtag(socket) was unset, setup a new
            # trace for the new variable with that name
            trace variable jtag(socket) wu socket_trace
         }
      }
}

# Check that an entry box widget only contains decimal digits
proc validate_isdigit {action new} {
   if {$action == 1} {
      # Insert
      return [string is digit $new]
   } else {
      # Delete
      return 1
   }
}

# Check that an entry box widget contains no more than 8
# hexdecimal digits
proc validate_isxdigit {action new} {
   set status 0
   if {$action == 1} {
      # Insert
      if {[string is xdigit $new] && ([string length $new] <= 8)} {
         set status 1
      }
   } else {
      # Delete
      set status 1
   }
   return $status
}

# Client connect/disconnect button callback
proc connect_button {} {
   global jtag
   if {[info exists jtag(socket)]} {
      client_close
   } else {
      client_open $jtag(port)
   }
}

# Pressing the read/write buttons will automatically connect
# to the client if needed. The trace on jtag(socket) will
# cause the connection status GUI elements to update
#
proc read_button {} {
   global jtag

   # The address and data entry boxes contain only digits,
   # so add the leading 0x to the address
   set data [jtag_read 0x$jtag(address)]

   # and then strip the response 0x
   set jtag(data) [format %X $data]
```

```
}

proc write_button {} {
   global jtag

   # The address and data entry boxes contain only digits,
   # so add the leading 0x to the command
   jtag_write 0x$jtag(address) 0x$jtag(data)
}


# Check that an entry box widget contains no more than 8
# hexdecimal digits
proc validate_cmd_D_byte {action new} {
   global ecs_command.data

   set status 0
   if {$action == 1} {
      # Insert
      if {[string is xdigit $new] && ([string length $new] <= 2)} {
         set status 1
         if {([string length $new] == 1)} {
            set new "0$new"
         }
         if {([string length $new] == 0)} {
            set new "00"
         }
      }
   } else {
      # Delete
      set status 1
   }
   set ecs_command.data(Data0)
     "${ecs_command.data(D3)}${ecs_command.data(D2)}${ecs_command.data(D1)}${ecs_command.data(D0)}"
   set ecs_command.data(Data1)
     "${ecs_command.data(D7)}${ecs_command.data(D6)}${ecs_command.data(D5)}${ecs_command.data(D4)}"
   set ecs_command.data(Data2)
     "${ecs_command.data(D11)}${ecs_command.data(D10)}${ecs_command.data(D9)}${ecs_command.data(D8)}"
   set ecs_command.data(Data3)
     "${ecs_command.data(D15)}${ecs_command.data(D14)}${ecs_command.data(D13)}${ecs_command.data(D12)}"


   return $status
}

proc update_ecs_command_data { } {
   global ecs_command.data
   set ecs_command.data(Data0)
     "${ecs_command.data(D3)}${ecs_command.data(D2)}${ecs_command.data(D1)}${ecs_command.data(D0)}"
   set ecs_command.data(Data1)
     "${ecs_command.data(D7)}${ecs_command.data(D6)}${ecs_command.data(D5)}${ecs_command.data(D4)}"
   set ecs_command.data(Data2)
     "${ecs_command.data(D11)}${ecs_command.data(D10)}${ecs_command.data(D9)}${ecs_command.data(D8)}"
   set ecs_command.data(Data3)
     "${ecs_command.data(D15)}${ecs_command.data(D14)}${ecs_command.data(D13)}${ecs_command.data(D12)}"
}

set num 6
set ScaleVal 5000
set hueval 0
```

```
set gbt_list {0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 \
   12 13 14 15 16 17 18 19 20 \
   21 22 23 24 25 26 27 28 29 \
   30 31 32 33 34 35}


set gbt_number_list {0 1 2 3 4\
   5 6 7 8 9 10 11 12 13 14 15\
   16 17 18 19 20 \
   21 22 23 24 25 26 27 28 29 \
   30 31 32 33 34 35}
set sca_number_list {0 1 2 3 4\
   5 6 7 8 9 10 11 12 13 14 15\
   16 17}
set channel_list {"SCA"
                  "SPI"
                  "GPIO"
                  "I2C 0"
                  "I2C 1"
                  "I2C 2"
                  "I2C 3"
                  "I2C 4"
                  "I2C 5"
                  "I2C 6"
                  "I2C 7"
                  "I2C 8"
                  "I2C 9"
                  "I2C 10"
                  "I2C 11"
                  "I2C 12"
                  "I2C 13"
                  "I2C 14"
                  "I2C 15"
                  "JTAG"
                  "ADC"
                  "DAC"
}


#
#

#"NC_W_CRA"  Write the SLVS transmit current level setting]
#"NC_R_CRA" Read the SLVS transmit current level setting
#"NC_W_CRB" Write bits 0 to 7 of the CH_EN register
#"NC_R_CRB" Read bits 0 to 7 of the CH_EN register
#"NC_W_CRC" Write bits 8 to 15 of the CH_EN register
#"NC_R_CRC"Read bits 8 to 15 of the CH_EN register
#"NC_W_CRD" Write bits 16 to 21 of the CH_EN register
#"NC_R_CRD" Read bits 16 to 21 of the CH_EN register

set ecs_sca_cmd_enum {
   "ECS_NC_W_CRA"
   "ECS_NC_R_CRA"
   "ECS_NC_W_CRB"
   "ECS_NC_R_CRB"
   "ECS_NC_W_CRC"
   "ECS_NC_R_CRC"
   "ECS_NC_W_CRD"
   "ECS_NC_R_CRD"
```

```
   "ECS_CRA_wrt_rea"
   "ECS_CRB_wrt_rea"
   "ECS_CRC_wrt_rea"
   "ECS_CRD_wrt_rea"
}

set ecs_spi_cmd_enum {
   "ECS_SPI_W_CTRL"
   "ECS_SPI_R_CTRL"
   "ECS_SPI_W_FREQ"
   "ECS_SPI_R_FREQ"
   "ECS_SPI_W_SS"
   "ECS_SPI_R_SS"
   "ECS_SPI_W_MOSI0"
   "ECS_SPI_R_MOSI0"
   "ECS_SPI_W_MOSI1"
   "ECS_SPI_R_MOSI1"
   "ECS_SPI_W_MOSI2"
   "ECS_SPI_R_MOSI2"
   "ECS_SPI_W_MOSI3"
   "ECS_SPI_R_MOSI3"
   "ECS_SPI_GO"
}

set ecs_gpio_cmd_enum {
      "ECS_GPIO_REA_IN_DATA"
      "ECS_GPIO_SET_OUT_DATA"
      "ECS_GPIO_REA_OUT_DATA"
      "ECS_GPIO_SET_IO_DIR"
      "ECS_GPIO_REA_IO_DIR"
      "ECS_GPIO_SET_INTEN"
      "ECS_GPIO_REA_INTEN"
      "ECS_GPIO_SET_TRIG"
      "ECS_GPIO_REA_TRIG"
      "ECS_GPIO_SET_GIE"
      "ECS_GPIO_REA_GIE"
      "ECS_GPIO_SET_INT"
      "ECS_GPIO_REA_INT"
      "ECS_GPIO_SET_CLK_SEL"
      "ECS_GPIO_REA_CLK_SEL"
      "ECS_GPIO_SET_EDG"
      "ECS_GPIO_REA_EDG"

}

set ecs_i2c_cmd_enum {
   "ECS_I2C_W_CTRL"
   "ECS_I2C_R_CTRL"
   "ECS_I2C_R_STR"
   "ECS_I2C_W_MSK"
   "ECS_I2C_R_MSK"
   "ECS_I2C_W_DATA0"
   "ECS_I2C_R_DATA0"
   "ECS_I2C_W_DATA1"
   "ECS_I2C_R_DATA1"
   "ECS_I2C_W_DATA2"
   "ECS_I2C_R_DATA2"
   "ECS_I2C_W_DATA3"
   "ECS_I2C_R_DATA3"
   "ECS_I2C_S_7B_W"
```

```
   "ECS_I2C_S_7B_R"
   "ECS_I2C_S_10B_W"
   "ECS_I2C_S_10B_R"
   "ECS_I2C_M_7B_W"
   "ECS_I2C_M_7B_R"
   "ECS_I2C_M_10B_W"
   "ECS_I2C_M_10B_R"
   "ECS_I2C_RMW_AND"
   "ECS_I2C_RMW_OR"
   "ECS_I2C_RMW_XOR"
   "ECS_I2C_WRITE"
   "ECS_I2C_READ"
   "ECS_I2C_WRITE_EXT"
   "ECS_I2C_READ_EXT"
   "ECS_I2C_W_R_DATA0"
   "ECS_I2C_W_R_DATA1"
   "ECS_I2C_W_R_DATA2"
   "ECS_I2C_W_R_DATA3"
   "ECS_I2C_W_R_CTRL"
   "ECS_I2C_W_R_MSK"
}


proc init_cmd_table { } {
   global ecs_cmd_table;
   global channel_code_table;

   set "channel_code_table(SCA)" "00"
   set "channel_code_table(SPI)"             "01"
   set "channel_code_table(GPIO)"            "02"
   set "channel_code_table(I2C 0)"           "03"
   set "channel_code_table(I2C 1)"      "04"
   set "channel_code_table(I2C 2)"     "05"
   set "channel_code_table(I2C 3)"     "06"
   set "channel_code_table(I2C 4)"     "07"
   set "channel_code_table(I2C 5)"     "08"
   set "channel_code_table(I2C 6)"     "09"
   set "channel_code_table(I2C 7)"     "0a"
   set "channel_code_table(I2C 8)"     "0b"
   set "channel_code_table(I2C 9)"     "0c"
   set "channel_code_table(I2C 10)" "0d"
   set "channel_code_table(I2C 11)" "0e"
   set "channel_code_table(I2C 12)" "0f"
   set "channel_code_table(I2C 13)" "10"
   set "channel_code_table(I2C 14)" "11"
   set "channel_code_table(I2C 15)" "12"
   set "channel_code_table(JTAG)"   "13"
   set "channel_code_table(ADC)"    "14"
   set "channel_code_table(DAC)"    "15"

   set ecs_cmd_table(ECS_NC_W_CRA) "00"
   set ecs_cmd_table(ECS_NC_R_CRA) "01"
   set ecs_cmd_table(ECS_NC_W_CRB) "02"
   set ecs_cmd_table(ECS_NC_R_CRB) "03"
   set ecs_cmd_table(ECS_NC_W_CRC) "04"
   set ecs_cmd_table(ECS_NC_R_CRC) "05"
   set ecs_cmd_table(ECS_NC_W_CRD) "06"
   set ecs_cmd_table(ECS_NC_R_CRD) "07"
   set ecs_cmd_table(ECS_CRA_wrt_rea) "08"
   set ecs_cmd_table(ECS_CRB_wrt_rea) "09"
```

```
set ecs_cmd_table(ECS_CRC_wrt_rea) "0a"
set ecs_cmd_table(ECS_CRD_wrt_rea) "0b"

set ecs_cmd_table(ECS_ADC_GO) "B2"
set ecs_cmd_table(ECS_ADC_set_InputLine) "30"
set ecs_cmd_table(ECS_ADC_rea_InputLine) "31"
set ecs_cmd_table(ECS_ADC_set_CurrentSource) "40"
set ecs_cmd_table(ECS_ADC_rea_CurrentSource) "41"

set ecs_cmd_table(ECS_ADC_rea_DATA_Ofs_Ga) "21"
set ecs_cmd_table(ECS_ADC_rea_DATA_Ofs) "A1"
set ecs_cmd_table(ECS_ADC_rea_DATA) "51"
set ecs_cmd_table(ECS_ADC_rea_OFFSET) "61"
set ecs_cmd_table(ECS_ADC_wrt_CTRL) "10"
set ecs_cmd_table(ECS_ADC_rea_CTRL) "11"
set ecs_cmd_table(ECS_ADC_GO_SingleSlope) "02"
set ecs_cmd_table(ECS_ADC_wrt_GainCalibrReg) "70"
set ecs_cmd_table(ECS_ADC_rea_GainCalibrReg) "71"
set ecs_cmd_table(ECS_ADC_wrt_ID) "90"
set ecs_cmd_table(ECS_ADC_rea_ID) "91"

set ecs_cmd_table(ECS_DAC_A_wrt) "10"
set ecs_cmd_table(ECS_DAC_A_read) "11"
set ecs_cmd_table(ECS_DAC_B_wrt) "20"
set ecs_cmd_table(ECS_DAC_B_read) "21"
set ecs_cmd_table(ECS_DAC_C_wrt) "30"
set ecs_cmd_table(ECS_DAC_C_read) "31"
set ecs_cmd_table(ECS_DAC_D_wrt) "40"
set ecs_cmd_table(ECS_DAC_D_read) "41"

set ecs_cmd_table(ECS_SPI_GO) "72"
set ecs_cmd_table(ECS_SPI_W_CTRL) "40"
set ecs_cmd_table(ECS_SPI_R_CTRL) "41"
set ecs_cmd_table(ECS_SPI_W_SS) "60"
set ecs_cmd_table(ECS_SPI_R_SS) "61"
set ecs_cmd_table(ECS_SPI_W_FREQ) "50"
set ecs_cmd_table(ECS_SPI_R_FREQ) "51"

set ecs_cmd_table(ECS_SPI_W_MOSI0) "00"
set ecs_cmd_table(ECS_SPI_R_MOSI0) "01"
set ecs_cmd_table(ECS_SPI_W_MOSI1) "10"
set ecs_cmd_table(ECS_SPI_R_MOSI1) "11"
set ecs_cmd_table(ECS_SPI_W_MOSI2) "20"
set ecs_cmd_table(ECS_SPI_R_MOSI2) "21"
set ecs_cmd_table(ECS_SPI_W_MOSI3) "30"
set ecs_cmd_table(ECS_SPI_R_MOSI3) "31"

set ecs_cmd_table(ECS_SPI_MISO_rea_Rx0) "01"
set ecs_cmd_table(ECS_SPI_MISO_rea_Rx1) "11"
set ecs_cmd_table(ECS_SPI_MISO_rea_Rx2) "21"
set ecs_cmd_table(ECS_SPI_MISO_rea_Rx3) "31"

set ecs_cmd_table(ECS_SPI_WRITE_TX_REG) "48"
set ecs_cmd_table(ECS_SPI_READ_RX_REG) "49"

set ecs_cmd_table(ECS_JTAG_GO) "A2"
set ecs_cmd_table(ECS_JTAG_wrt_CTRL) "80"
set ecs_cmd_table(ECS_JTAG_rea_CTRL) "81"
set ecs_cmd_table(ECS_JTAG_wrt_DIV) "90"
set ecs_cmd_table(ECS_JTAG_rea_DIV) "91"
```

```
set ecs_cmd_table(ECS_JTAG_TMS_wrt_Tx0) "40"
set ecs_cmd_table(ECS_JTAG_TMS_rea_Tx0) "41"
set ecs_cmd_table(ECS_JTAG_TMS_wrt_Tx1) "50"
set ecs_cmd_table(ECS_JTAG_TMS_rea_Tx1) "51"
set ecs_cmd_table(ECS_JTAG_TMS_wrt_Tx2) "60"
set ecs_cmd_table(ECS_JTAG_TMS_rea_Tx2) "61"
set ecs_cmd_table(ECS_JTAG_TMS_wrt_Tx3) "70"
set ecs_cmd_table(ECS_JTAG_TMS_rea_Tx3) "71"

set ecs_cmd_table(ECS_JTAG_TDO_wrt_Tx0) "00"
set ecs_cmd_table(ECS_JTAG_TDO_rea_Tx0) "01"
set ecs_cmd_table(ECS_JTAG_TDO_wrt_Tx1) "10"
set ecs_cmd_table(ECS_JTAG_TDO_rea_Tx1) "11"
set ecs_cmd_table(ECS_JTAG_TDO_wrt_Tx2) "20"
set ecs_cmd_table(ECS_JTAG_TDO_rea_Tx2) "21"
set ecs_cmd_table(ECS_JTAG_TDO_wrt_Tx3) "30"
set ecs_cmd_table(ECS_JTAG_TDO_rea_Tx3) "31"

set ecs_cmd_table(ECS_JTAG_TDI_rea_Rx0) "01"
set ecs_cmd_table(ECS_JTAG_TDI_rea_Rx1) "11"
set ecs_cmd_table(ECS_JTAG_TDI_rea_Rx2) "21"
set ecs_cmd_table(ECS_JTAG_TDI_rea_Rx3) "31"

set ecs_cmd_table(ECS_JTAG_WRITE_TMS) "52"
set ecs_cmd_table(ECS_JTAG_READ_TMS) "53"
set ecs_cmd_table(ECS_JTAG_WRITE_TDO) "54"
set ecs_cmd_table(ECS_JTAG_READ_TDO) "55"

set ecs_cmd_table(ECS_GPIO_REA_IN_DATA) "01"
set ecs_cmd_table(ECS_GPIO_SET_OUT_DATA) "10"
set ecs_cmd_table(ECS_GPIO_REA_OUT_DATA) "11"
set ecs_cmd_table(ECS_GPIO_SET_IO_DIR) "20"
set ecs_cmd_table(ECS_GPIO_REA_IO_DIR) "21"
set ecs_cmd_table(ECS_GPIO_SET_INTEN) "20"
set ecs_cmd_table(ECS_GPIO_REA_INTEN) "21"
set ecs_cmd_table(ECS_GPIO_SET_TRIG) "30"
set ecs_cmd_table(ECS_GPIO_REA_TRIG) "31"
set ecs_cmd_table(ECS_GPIO_SET_GIE) "60"
set ecs_cmd_table(ECS_GPIO_REA_GIE) "61"
set ecs_cmd_table(ECS_GPIO_SET_INT) "70"
set ecs_cmd_table(ECS_GPIO_REA_INT) "71"
set ecs_cmd_table(ECS_GPIO_SET_CLK_SEL) "80"
set ecs_cmd_table(ECS_GPIO_REA_CLK_SEL) "81"
set ecs_cmd_table(ECS_GPIO_SET_EDG) "90"
set ecs_cmd_table(ECS_GPIO_REA_EDG) "91"


#! Start single byte write transmission 7 bit addr.
set ecs_cmd_table(ECS_I2C_S_7B_W)    "82"
#! Start single byte read transmission 7-bit addr.
set ecs_cmd_table(ECS_I2C_S_7B_R)    "86"
#! Start single byte write transmission 10-bit addr.
set ecs_cmd_table(ECS_I2C_S_10B_W)    "8A"
#! Start single byte read transmission 10-bit addr.
set ecs_cmd_table(ECS_I2C_S_10B_R)    "8E"
#! Start multi byte write transmission 7-bit addr.
set ecs_cmd_table(ECS_I2C_M_7B_W)    "DA"
#! Start multi byte read transmission 7-bit addr.
set ecs_cmd_table(ECS_I2C_M_7B_R)    "DE"
```

```
#! Start multi byte write transmission 10-bit addr.
set ecs_cmd_table(ECS_I2C_M_10B_W)   "E2"
#! Start multi byte read transmission 10-bit addr.
set ecs_cmd_table(ECS_I2C_M_10B_R)    "E6"
#! Read Ã¯Â¿Â½ modify Ã¯Â¿Â½ write (AND with MASK reg.)
set ecs_cmd_table(ECS_I2C_RMW_AND)    "C2"
#! Read Ã¯Â¿Â½ modify Ã¯Â¿Â½ write (OR with MASK reg.)
set ecs_cmd_table(ECS_I2C_RMW_OR)    "C6"
#! Read Ã¯Â¿Â½ modify Ã¯Â¿Â½ write (XOR with MASK reg.)
set ecs_cmd_table(ECS_I2C_RMW_XOR)    "CA"

##Data register access
#!Write the DATA TRANSMIT register bits 0 to 31
set ecs_cmd_table(ECS_I2C_W_DATA0)      "40"
#!Read the DATA TRANSMIT register bits 0 to 31
set ecs_cmd_table(ECS_I2C_R_DATA0)      "41"
#!Write the DATA TRANSMIT register bits 32 to 63
set ecs_cmd_table(ECS_I2C_W_DATA1)      "50"
#!Read the DATA TRANSMIT register bits 32 to 63
set ecs_cmd_table(ECS_I2C_R_DATA1)      "51"
#!Write the DATA TRANSMIT register bits 64 to 95
set ecs_cmd_table(ECS_I2C_W_DATA2)      "60"
#!Read the DATA TRANSMIT register bits 64 to 95
set ecs_cmd_table(ECS_I2C_R_DATA2)      "61"
#!Write the DATA TRANSMIT register bits 96 to 127
set ecs_cmd_table(ECS_I2C_W_DATA3)      "70"
#!Read the DATA TRANSMIT register bits 96 to 127
set ecs_cmd_table(ECS_I2C_R_DATA3)      "71"

#Control registers:
#!Write the CONTROL register
set ecs_cmd_table(ECS_I2C_W_CTRL)      "30"
#!Read the CONTROL register
set ecs_cmd_table(ECS_I2C_R_CTRL)      "31"
#!Write the MASK register
set ecs_cmd_table(ECS_I2C_W_MSK)      "20"
#!Read the MASK register
set ecs_cmd_table(ECS_I2C_R_MSK)      "21"
#!Read the STATUS register
set ecs_cmd_table(ECS_I2C_R_STR)      "11"

#! Start write transmission 7 bit addr.
set ecs_cmd_table(ECS_I2C_WRITE)     "90"
#! Start read transmission 7-bit addr.
set ecs_cmd_table(ECS_I2C_READ)    "91"
#! Start write transmission 10-bit addr.
set ecs_cmd_table(ECS_I2C_WRITE_EXT)    "92"
#! Start read transmission 10-bit addr.
set ecs_cmd_table(ECS_I2C_READ_EXT)    "93"

#Data register _write_and_read_ commands
#!Write the DATA TRANSMIT register bits 0 to 31
set ecs_cmd_table(ECS_I2C_W_R_DATA0)    "94"
#!Write the DATA TRANSMIT register bits 32 to 63
set ecs_cmd_table(ECS_I2C_W_R_DATA1)    "95"
#!Write the DATA TRANSMIT register bits 64 to 95
set ecs_cmd_table(ECS_I2C_W_R_DATA2)    "96"
#!Write the DATA TRANSMIT register bits 96 to 127
set ecs_cmd_table(ECS_I2C_W_R_DATA3)    "97"
```

```
   #!Write AND READ the CONTROL register
   set ecs_cmd_table(ECS_I2C_W_R_CTRL)    "98"
   #!Write AND READ the MASK register
   set ecs_cmd_table(ECS_I2C_W_R_MSK)    "99"


}


   init_cmd_table

   set ecs_command "0"
   for {set i 0} {$i < 16} {incr i} {
      #puts "I inside first loop: $i"
      set ecs_command.data(D$i) "00"
      #puts "ecs_command.data(D$i) = $ecs_command.data(D$i)"
   }

   set ecs_command.data(CommandHeader0) "00000000"
   set ecs_command.data(CommandHeader1) "00040000"
   set ecs_command.data(Data0) "00000000"
   set ecs_command.data(Data1) "00000000"
   set ecs_command.data(Data2) "00000000"
   set ecs_command.data(Data3) "00000000"

   set  ecs_reply.data(CommandHeader0) "00000000"
   set  ecs_reply.data(CommandHeader1) "00000000"
   set  ecs_reply.data(Data0) "00000000"
   set  ecs_reply.data(Data1) "00000000"
   set  ecs_reply.data(Data2) "00000000"
   set  ecs_reply.data(Data3) "00000000"


   set ecs_command.data(ProtSpecific) "0000"

   set ecs_reply.Address      "00000000"

   # set hue(2).lol(5) 15
   # set hue(1).lol(56) "jhjsdhfikjsdbnkijbdsnkf"
   # puts "hue(2).lol(5) = ${hue(2).lol(5)}"
   # puts "hue(2).lol(5) = ${hue(1).lol(56)}"


set ecs_adc_cmd_enum {
      "ECS_ADC_GO"
      "ECS_ADC_set_InputLine"
      "ECS_ADC_rea_InputLine"
      "ECS_ADC_set_CurrentSource"
      "ECS_ADC_rea_CurrentSource"
      "ECS_ADC_rea_DATA_Ofs_Ga"
      "ECS_ADC_rea_DATA_Ofs"
      "ECS_ADC_rea_DATA"
      "ECS_ADC_rea_OFFSET"
      "ECS_ADC_wrt_CTRL"
      "ECS_ADC_rea_CTRL"
      "ECS_ADC_GO_SingleSlope"
      "ECS_ADC_wrt_GainCalibrReg"
      "ECS_ADC_rea_GainCalibrReg"
      "ECS_ADC_wrt_ID"
      "ECS_ADC_rea_ID"
```

```
}

set ecs_dac_cmd_enum {
      "ECS_DAC_A_wrt"
      "ECS_DAC_A_read"
      "ECS_DAC_B_wrt"
      "ECS_DAC_B_read"
      "ECS_DAC_C_wrt"
      "ECS_DAC_C_read"
      "ECS_DAC_D_wrt"
      "ECS_DAC_D_read"
}

set ecs_jtag_cmd_enum {
      "ECS_JTAG_GO"
      "ECS_JTAG_wrt_CTRL"
      "ECS_JTAG_rea_CTRL"
      "ECS_JTAG_wrt_DIV"
      "ECS_JTAG_rea_DIV"
      "ECS_JTAG_TMS_wrt_Tx0"
      "ECS_JTAG_TMS_rea_Tx0"
      "ECS_JTAG_TMS_wrt_Tx1"
      "ECS_JTAG_TMS_rea_Tx1"
      "ECS_JTAG_TMS_wrt_Tx2"
      "ECS_JTAG_TMS_rea_Tx2"
      "ECS_JTAG_TMS_wrt_Tx3"
      "ECS_JTAG_TMS_rea_Tx3"
      "ECS_JTAG_TDO_wrt_Tx0"
      "ECS_JTAG_TDO_rea_Tx0"
      "ECS_JTAG_TDO_wrt_Tx1"
      "ECS_JTAG_TDO_rea_Tx1"
      "ECS_JTAG_TDO_wrt_Tx2"
      "ECS_JTAG_TDO_rea_Tx2"
      "ECS_JTAG_TDO_wrt_Tx3"
      "ECS_JTAG_TDO_rea_Tx3"
      "ECS_JTAG_TDI_rea_Rx0"
      "ECS_JTAG_TDI_rea_Rx1"
      "ECS_JTAG_TDI_rea_Rx2"
      "ECS_JTAG_TDI_rea_Rx3"
      "ECS_JTAG_WRITE_TMS"
      "ECS_JTAG_READ_TMS"
      "ECS_JTAG_WRITE_TDO"
      "ECS_JTAG_READ_TDO"
      }

set cmd_list {}

#toplevel .top -takefocus 1
# button .top.j -anchor w -command hue -text HUE -textvariable hueval
# grid .top.j
# spinbox .top.sbSpinBox -state normal -values {5 7 8}
# grid .top.sbSpinBox -column 1 -row 1


proc UpdateCommandHeader0 { }  {
   global win_top;
   global cbCh;
   global cbCmd;
   global gbt_number_list;
   global sca_number_list;
```

```
global channel_list;
global channel_code_table;
global cmd_list;
global ecs_sca_cmd_enum;
global ecs_spi_cmd_enum;
global ecs_gpio_cmd_enum;
global ecs_i2c_cmd_enum;
global ecs_adc_cmd_enum;
global ecs_dac_cmd_enum;
global ecs_jtag_cmd_enum;

global ecs_cmd_table;
global ecs_command.data;

puts "[lindex $channel_list [$cbCh getvalue]]";

switch -regexp [lindex $channel_list [$cbCh getvalue]] {
 SCA* {
     puts "SCA Channel"
   set cmd_list $ecs_sca_cmd_enum;
 }
SPI* {
     puts "SPI Channel"
   set cmd_list $ecs_spi_cmd_enum;
 }
GPIO* {
     puts "GPIO Channel"
   set cmd_list $ecs_gpio_cmd_enum;
 }
 I2C* {
     puts "I2C Channel"
   set cmd_list $ecs_i2c_cmd_enum;
 }
JTAG* {
     puts "DAC Channel"
   set cmd_list $ecs_jtag_cmd_enum;
 }
ADC* {
     puts "ADC Channel"
   set cmd_list $ecs_adc_cmd_enum;
 }
DAC* {
     puts "DAC Channel"
   set cmd_list $ecs_dac_cmd_enum;
 }
default {
   puts "null"
   }
}
#destroy $cbCmd;
#grid [ComboBox $win_top.frEcsCommand.cbCmd -values $cmd_list -modifycmd {puts "[lindex $cmd_list [
  $win_top.frEcsCommand.cbCmd getvalue]]"} -width 6] -column 4 -row 3
$win_top.frEcsCommand.cbCmd configure -values $cmd_list

set GbtNr [lindex $gbt_number_list [$win_top.frEcsCommand.cbGbtNr getvalue]]
set ScaNr [lindex $sca_number_list [$win_top.frEcsCommand.cbScaNr getvalue]]
set Ch [lindex $channel_list [$win_top.frEcsCommand.cbCh getvalue]]
set Cmd [lindex $cmd_list [$win_top.frEcsCommand.cbCmd getvalue]]

if { [string length $Ch] == 0 || [string length $ScaNr] == 0  || [string length $GbtNr] == 0 || [string
```

```
     length $Cmd] == 0  }  {
       puts "Nope"
   } else {

      set GbtNrHex [format %02X $GbtNr ]
      set ScaNrHex [format %02X $ScaNr ]
      set ChHex   "$channel_code_table($Ch)"

      set CmdHex "$ecs_cmd_table($Cmd)"
      #set ecs_reply.Address "${GbtNrHex}${ScaNrHex}$channel_code_table($Ch)00"
      puts "${GbtNrHex}${ScaNrHex}${ChHex}$CmdHex"
      set ecs_command.data(CommandHeader0) "${GbtNrHex}${ScaNrHex}${ChHex}$CmdHex"
   }

}

proc UpdateCommandHeader1 { }  {
   global sca_number_list
   global ecs_command.data
   global win_top

   if {[string length ${ecs_command.data(ProtSpecific)} ] <= 4} {
      if {([string length ${ecs_command.data(ProtSpecific)}] == 3)} {
         set ecs_command.data(ProtSpecific) "0${ecs_command.data(ProtSpecific)}"
      }
      if {([string length ${ecs_command.data(ProtSpecific)}] == 2)} {
         set ecs_command.data(ProtSpecific) "00${ecs_command.data(ProtSpecific)}"
      }
      if {([string length ${ecs_command.data(ProtSpecific)}] == 1)} {
         set ecs_command.data(ProtSpecific) "000${ecs_command.data(ProtSpecific)}"
      }
      if {([string length ${ecs_command.data(ProtSpecific)} ] == 0)} {
         set ecs_command.data(ProtSpecific) "0000"
      }
   }

   set Len [lindex $sca_number_list [$win_top.frEcsCommand.cbLen getvalue]]

   if { [string length $Len] == 0 }  {
      set ecs_command.data(CommandHeader1) "00000000"
   } else {

      set LenHex [format %02X $Len ]
      set ecs_command.data(CommandHeader1) "00$LenHex${ecs_command.data(ProtSpecific)}"
   }

}


proc validate_Prot_Specific {action new} {
   global ecs_command.data

   set status 0
   if {$action == 1} {
      # Insert
      if {[string is xdigit $new] && ([string length $new] <= 4)} {
         set status 1
         puts "Before = $new"
         # if len = 3 , Have to fill with (8-3) zeroes
         for { set i 0} { $i < 4 - [string length $new] } {incr i} {
```

```
            set new "0$new"
         }
         puts "After = $new"
      }
   } else {
      # Delete
      set status 1
   }
   set ecs_command.data(ProtSpecific) $new
   return $status
}




set Poll false

proc SendCommand { } {
   global win_top

   global gbt_list
   global gbt_number_list
   global sca_number_list
   global channel_list
   global cmd_list
   global ecs_cmd_table
   global channel_code_table
   global ecs_command.data

   puts "\nECS Command String:"

   puts ${ecs_command.data(CommandHeader0)}
   puts ${ecs_command.data(CommandHeader1)}
   puts ${ecs_command.data(Data0)}
   puts ${ecs_command.data(Data1)}
   puts ${ecs_command.data(Data2)}
   puts ${ecs_command.data(Data3)}
   puts ""

   jtag_write 0x50408 0x${ecs_command.data(CommandHeader0)}
   jtag_write 0x5040c 0x${ecs_command.data(CommandHeader1)}
   jtag_write 0x50410 0x${ecs_command.data(Data0)}
   jtag_write 0x50414 0x${ecs_command.data(Data1)}
   jtag_write 0x50418 0x${ecs_command.data(Data2)}
   jtag_write 0x5041c 0x${ecs_command.data(Data3)}

   #Write GO
   jtag_write 0x50400 0x1

}

proc SetReplyAddress { } {
   global win_top
   global gbt_number_list
   global sca_number_list
   global ecs_reply.Address
   global channel_code_table
   global channel_list

   set GbtNr [lindex $gbt_number_list [$win_top.frPollReply.cbGbtNr getvalue]]
```

```
   set ScaNr [lindex $sca_number_list [$win_top.frPollReply.cbScaNr getvalue]]
   set Ch [lindex $channel_list [$win_top.frPollReply.cbCh getvalue]]

   if { [string length $Ch] == 0 || [string length $ScaNr] == 0  || [string length $GbtNr] == 0  }  {
      if { [string length [$win_top.frPollReply.txtReplyAddr get ] ] == 0  } {
         $win_top.frPollReply.txtReplyAddr configure -textvariable "00000000"
      }
   } else {
      set GbtNrHex [format %02X $GbtNr ]
      set ScaNrHex [format %02X $ScaNr ]
      set ecs_reply.Address "${GbtNrHex}${ScaNrHex}$channel_code_table($Ch)00"
      puts ${ecs_reply.Address}
   }

   jtag_write 0x50404 0x${ecs_reply.Address}

}

proc GetReply { } {

   global jtag
   global  ecs_reply.data

   jtag_write 0x50400 0x02

   set  ecs_reply.data(CommandHeader0) [string replace  [jtag_read "0x50420"] 0 1 ""]

   set  ecs_reply.data(CommandHeader1) [string replace  [jtag_read "0x50424"] 0 1 ""]


   set  ecs_reply.data(Data0)  [string replace  [jtag_read "0x50428"] 0 1 ""]



   set  ecs_reply.data(Data1) [string replace  [jtag_read "0x5042c"] 0 1 ""]

   set  ecs_reply.data(Data2) [string replace  [jtag_read "0x50430"] 0 1 ""]

   set  ecs_reply.data(Data3) [string replace  [jtag_read "0x50434"] 0 1 ""]


   ValidateReply

}

proc ValidateReply { } {
   global win_top
   global  ecs_reply.data

   scan [string range ${ecs_reply.data(CommandHeader0)} 0 1] %x GbtNr
   $win_top.frEcsReply.cbGbtNr configure -text $GbtNr

   scan [string range ${ecs_reply.data(CommandHeader0)} 2 3] %x ScaNr
   $win_top.frEcsReply.cbScaNr configure -text $ScaNr

   scan [string range ${ecs_reply.data(CommandHeader0)} 4 5] %x Ch
   $win_top.frEcsReply.cbCh configure -text $Ch
```

```
   scan [string range ${ecs_reply.data(CommandHeader1)} 2 3] %x Len
   $win_top.frEcsReply.cbLen configure -text $Len


   $win_top.frEcsReply.cbProtSpecific configure -text [string range ${ecs_reply.data(CommandHeader1)} 4 7]

   scan [string range ${ecs_reply.data(CommandHeader1)} 0 1] %x Error

   puts "ecs_reply.data(CommandHeader1) = ${ecs_reply.data(CommandHeader1)}"
   set lol [string range ${ecs_reply.data(CommandHeader1)} 0 1]
   puts "Lol = $lol"
   puts "Error  = ${Error}"

   switch $lol {
    01 {
        $win_top.frEcsReply.lbErrorVal configure -text "Timeout" -bg red
    }
   02 {
         $win_top.frEcsReply.lbErrorVal configure -text "Invalid ch req" -bg red
    }
   04 {
         $win_top.frEcsReply.lbErrorVal configure -text "Invalid cmd req" -bg red
    }
   08 {
      $win_top.frEcsReply.lbErrorVal configure -text "Invalid TR" -bg red
    }
   10 {
        $win_top.frEcsReply.lbErrorVal configure -text "Invalid Len" -bg red
    }
   20 {
        $win_top.frEcsReply.lbErrorVal configure -text "Ch disabled" -bg red
    }
   40 {
       $win_top.frEcsReply.lbErrorVal configure -text "Ch busy" -bg red
    }
   80 {
       $win_top.frEcsReply.lbErrorVal configure -text "cmd in treatment" -bg red
    }
    default {
       $win_top.frEcsReply.lbErrorVal configure -text "Good!" -bg green
    }
}

   set ecs_reply.data(D3) [string range ${ecs_reply.data(Data0)} 0 1]


   set ecs_reply.data(D3) [string range ${ecs_reply.data(Data0)} 0 1]
   set ecs_reply.data(D2) [string range ${ecs_reply.data(Data0)} 2 3]
   set ecs_reply.data(D1) [string range ${ecs_reply.data(Data0)} 4 5]
   set ecs_reply.data(D0) [string range ${ecs_reply.data(Data0)} 6 7]

   set ecs_reply.data(D7) [string range ${ecs_reply.data(Data1)} 0 1]
   set ecs_reply.data(D6) [string range ${ecs_reply.data(Data1)} 2 3]
   set ecs_reply.data(D5) [string range ${ecs_reply.data(Data1)} 4 5]
   set ecs_reply.data(D4) [string range ${ecs_reply.data(Data1)} 6 7]

   set ecs_reply.data(D11) [string range ${ecs_reply.data(Data2)} 0 1]
   set ecs_reply.data(D10) [string range ${ecs_reply.data(Data2)} 2 3]
   set ecs_reply.data(D9) [string range ${ecs_reply.data(Data2)} 4 5]
   set ecs_reply.data(D8) [string range ${ecs_reply.data(Data2)} 6 7]
```

```
   set ecs_reply.data(D15) [string range ${ecs_reply.data(Data3)} 0 1]
   set ecs_reply.data(D14) [string range ${ecs_reply.data(Data3)} 2 3]
   set ecs_reply.data(D13) [string range ${ecs_reply.data(Data3)} 4 5]
   set ecs_reply.data(D12) [string range ${ecs_reply.data(Data3)} 6 7]

   #$win_top.frEcsReply.txtData0 configure -text ${ ecs_reply.data(Data0)}
   # puts " ecs_reply.data(Data0) = ${ ecs_reply.data(Data0)}"
   # $win_top.frEcsReply.txtData1 configure -textvariable ${ ecs_reply.data(Data1)}
   # $win_top.frEcsReply.txtData2 configure -textvariable ${ ecs_reply.data(Data2)}
   # $win_top.frEcsReply.txtData3 configure -textvariable ${ ecs_reply.data(Data3)}


   # $win_top.frEcsReply.txtData0 configure -textvariable $huheue



}

#proc



proc create_gui_ecs  { }  {
   global gbt_list
   global gbt_number_list
   global sca_number_list
   global channel_list
   global cmd_list
   global cbCh
   global cbCmd
   global win_top
   global Poll

   global ecs_reply.Address
   global  ecs_reply.data



   frame $win_top.frEcsCommand -borderwidth 2 -relief raised
   grid $win_top.frEcsCommand  -column 1 -row 1
   grid [Label $win_top.frEcsCommand.lbEcsCommmandFrame -text "ECS Command" ] -column 1 -row 1 -columnspan
     6


   grid [Label $win_top.frEcsCommand.lbGbtNr -text "GBT#" ] -column 1 -row 2 -sticky nswe
   grid [Label $win_top.frEcsCommand.lbScaNr -text "SCA#" ] -column 2 -row 2 -sticky nswe
   grid [Label $win_top.frEcsCommand.lbCh -text "CH#" ] -column 3 -row 2 -sticky nswe
   grid [Label $win_top.frEcsCommand.lbCmd -text "CMD" ] -column 4 -row 2 -columnspan 2 -sticky nswe


   grid [ComboBox $win_top.frEcsCommand.cbGbtNr -values $gbt_number_list -modifycmd {UpdateCommandHeader0
     } -editable false -width 5 ] -column 1 -row 3 -sticky nswe
   grid [ComboBox $win_top.frEcsCommand.cbScaNr -values $sca_number_list -modifycmd {UpdateCommandHeader0
     } -editable false  -width 5 ] -column 2 -row 3 -sticky nswe
   set cbCh [ComboBox $win_top.frEcsCommand.cbCh -values $channel_list -modifycmd {UpdateCommandHeader0 }
     -editable false  -width 5 ]
   grid $cbCh -column 3 -row 3 -sticky nswe
```

```
set cbCmd [ComboBox $win_top.frEcsCommand.cbCmd -textvariable $cmd_list -modifycmd
  {UpdateCommandHeader0 } -width 15 ]
grid $cbCmd -column 4 -row 3 -columnspan 2 -sticky nswe


grid [entry $win_top.frEcsCommand.txtCommandHeader0 -width 10  -justify right -relief sunken
  -textvariable ecs_command.data(CommandHeader0) -validatecommand {validate_isxdigit %d %P} -validate
  all -state disabled ] -column 6 -row 3 -sticky nswe


#

grid [Label $win_top.frEcsCommand.lbNull -text "Unused" ] -column 1 -row 4  -sticky nswe
grid [Label $win_top.frEcsCommand.lbLen -text "Length" ] -column 2 -row 4 -sticky nswe
grid [Label $win_top.frEcsCommand.lbProtSpecific -text "Protocol Specific" ] -column 3 -row 4
  -columnspan 2 -sticky nswe
#-modifycmd {UpdateCommandHeader1 }
grid [ComboBox $win_top.frEcsCommand.cbLen -values $sca_number_list  -editable false  -width 5
  -modifycmd {UpdateCommandHeader1 } ] -column 2 -row 5 -sticky nswe

# -validatecommand {UpdateCommandHeader1 } -validate all
grid [entry $win_top.frEcsCommand.txtProtSpecific -width 10  -justify right -relief sunken
  -textvariable ecs_command.data(ProtSpecific)  -validatecommand {validate_Prot_Specific %d %P}
  -validate all ] -column 3 -row 5 -columnspan 2 -sticky nswe

grid [Label $win_top.frEcsCommand.lbSpace -text "" -width 5 ] -column 5 -row 5  -sticky nswe

grid [entry $win_top.frEcsCommand.txtCommandHeader1 -width 10  -justify right -relief sunken
  -textvariable ecs_command.data(CommandHeader1) -validatecommand {validate_isxdigit %d %P} -validate
  all -state disabled ] -column 6 -row 5 -sticky nswe
#

#

grid [Label $win_top.frEcsCommand.lbD3 -text "D3" ] -column 1 -row 7 -sticky nswe
grid [Label $win_top.frEcsCommand.lbD2 -text "D2" ] -column 2 -row 7 -sticky nswe
grid [Label $win_top.frEcsCommand.lbD1 -text "D1" ] -column 3 -row 7 -sticky nswe
grid [Label $win_top.frEcsCommand.lbD0 -text "D0" ] -column 4 -row 7 -sticky nswe


grid [entry $win_top.frEcsCommand.txtD3 -width 5  -justify right -relief sunken -textvariable
  ecs_command.data(D3) -validatecommand {validate_cmd_D_byte %d %P} -validate all ] -column 1 -row 8
  -sticky nswe
grid [entry $win_top.frEcsCommand.txtD2 -width 5  -justify right -relief sunken -textvariable
  ecs_command.data(D2) -validatecommand {validate_cmd_D_byte %d %P} -validate all ] -column 2 -row 8
  -sticky nswe
grid [entry $win_top.frEcsCommand.txtD1 -width 5  -justify right -relief sunken -textvariable
  ecs_command.data(D1) -validatecommand {validate_cmd_D_byte %d %P} -validate all ] -column 3 -row 8
  -sticky nswe
grid [entry $win_top.frEcsCommand.txtD0 -width 5  -justify right -relief sunken -textvariable
  ecs_command.data(D0) -validatecommand {validate_cmd_D_byte %d %P} -validate all ] -column 4 -row 8
  -sticky nswe

grid [entry $win_top.frEcsCommand.txtData0 -width 10  -justify right -relief sunken -textvariable
  ecs_command.data(Data0) -validatecommand {validate_isxdigit %d %P} -validate all -state disabled ]
  -column 6 -row 8  -columnspan 2 -sticky nswe
#

#
```

```
grid [Label $win_top.frEcsCommand.lbD7 -text "D7" ] -column 1 -row 9
grid [Label $win_top.frEcsCommand.lbD6 -text "D6" ] -column 2 -row 9
grid [Label $win_top.frEcsCommand.lbD5 -text "D5" ] -column 3 -row 9
grid [Label $win_top.frEcsCommand.lbD4 -text "D4" ] -column 4 -row 9


grid [entry $win_top.frEcsCommand.txtD7 -width 5  -justify right -relief sunken -textvariable
  ecs_command.data(D7) -validatecommand {validate_cmd_D_byte %d %P} -validate all ] -column 1 -row 10
  -sticky nswe
grid [entry $win_top.frEcsCommand.txtD6 -width 5  -justify right -relief sunken -textvariable
  ecs_command.data(D6) -validatecommand {validate_cmd_D_byte %d %P} -validate all ] -column 2 -row 10
  -sticky nswe
grid [entry $win_top.frEcsCommand.txtD5 -width 5  -justify right -relief sunken -textvariable
  ecs_command.data(D5) -validatecommand {validate_cmd_D_byte %d %P} -validate all ] -column 3 -row 10
  -sticky nswe
grid [entry $win_top.frEcsCommand.txtD4 -width 5  -justify right -relief sunken -textvariable
  ecs_command.data(D4) -validatecommand {validate_cmd_D_byte %d %P} -validate all ] -column 4 -row 10
  -sticky nswe

grid [entry $win_top.frEcsCommand.txtData1 -width 10  -justify right -relief sunken -textvariable
  ecs_command.data(Data1) -validatecommand {validate_isxdigit %d %P} -validate all -state disabled ]
  -column 6 -row 10  -columnspan 2 -sticky nswe
#
grid [Label $win_top.frEcsCommand.lbD11 -text "D11" ] -column 1 -row 11
grid [Label $win_top.frEcsCommand.lbD10 -text "D10" ] -column 2 -row 11
grid [Label $win_top.frEcsCommand.lbD9 -text "D9" ] -column 3 -row 11
grid [Label $win_top.frEcsCommand.lbD8 -text "D8" ] -column 4 -row 11


grid [entry $win_top.frEcsCommand.txtD11 -width 5  -justify right -relief sunken -textvariable
  ecs_command.data(D11) -validatecommand {validate_cmd_D_byte %d %P} -validate all ] -column 1 -row 12
  -sticky nswe
grid [entry $win_top.frEcsCommand.txtD10 -width 5  -justify right -relief sunken -textvariable
  ecs_command.data(D10) -validatecommand {validate_cmd_D_byte %d %P} -validate all ] -column 2 -row 12
  -sticky nswe
grid [entry $win_top.frEcsCommand.txtD9 -width 5  -justify right -relief sunken -textvariable
  ecs_command.data(D9) -validatecommand {validate_cmd_D_byte %d %P} -validate all ] -column 3 -row 12
  -sticky nswe
grid [entry $win_top.frEcsCommand.txtD8 -width 5  -justify right -relief sunken -textvariable
  ecs_command.data(D8) -validatecommand {validate_cmd_D_byte %d %P} -validate all ] -column 4 -row 12
  -sticky nswe

grid [entry $win_top.frEcsCommand.txtData2 -width 10  -justify right -relief sunken -textvariable
  ecs_command.data(Data2) -validatecommand {validate_isxdigit %d %P} -validate all -state disabled ]
  -column 6 -row 12  -columnspan 2 -sticky nswe
#

grid [Label $win_top.frEcsCommand.lbD15 -text "D15" ] -column 1 -row 13
grid [Label $win_top.frEcsCommand.lbD14 -text "D14" ] -column 2 -row 13
grid [Label $win_top.frEcsCommand.lbD13 -text "D13" ] -column 3 -row 13
grid [Label $win_top.frEcsCommand.lbD12 -text "D12" ] -column 4 -row 13


grid [entry $win_top.frEcsCommand.txtD15 -width 5  -justify right -relief sunken -textvariable
  ecs_command.data(D15) -validatecommand {validate_cmd_D_byte %d %P} -validate all ] -column 1 -row 14
  -sticky nswe
grid [entry $win_top.frEcsCommand.txtD14 -width 5  -justify right -relief sunken -textvariable
  ecs_command.data(D14) -validatecommand {validate_cmd_D_byte %d %P} -validate all ] -column 2 -row 14
  -sticky nswe
grid [entry $win_top.frEcsCommand.txtD13 -width 5  -justify right -relief sunken -textvariable
```

```
  ecs_command.data(D13) -validatecommand {validate_cmd_D_byte %d %P} -validate all ] -column 3 -row 14
  -sticky nswe
grid [entry $win_top.frEcsCommand.txtD12 -width 5  -justify right -relief sunken -textvariable
  ecs_command.data(D12) -validatecommand {validate_cmd_D_byte %d %P} -validate all ] -column 4 -row 14
  -sticky nswe

grid [entry $win_top.frEcsCommand.txtData3 -width 10  -justify right -relief sunken -textvariable
  ecs_command.data(Data3) -validatecommand {validate_isxdigit %d %P} -validate all -state disabled ]
  -column 6 -row 14  -columnspan 2 -sticky nswe
#

grid [Button $win_top.frEcsCommand.btnSend -text "Send Command"  -command { SendCommand } ] -column 1
  -row 16 -columnspan 6  -sticky nswe

# ------------------------------------------------------------------------------------
# ------------------------------------------------------------------------------------

frame $win_top.frPollReply -borderwidth 2 -relief raised -bg lightgreen
grid $win_top.frPollReply  -column 1 -row 2 -sticky nswe
grid [Label $win_top.frPollReply.frPollReplyFrame -text "ECS Reply Address" ] -column 1 -row 1
  -columnspan 6 -sticky nswe

grid [Label $win_top.frPollReply.lbGbtNr -text "GBT#" ] -column 1 -row 2 -sticky nswe
grid [Label $win_top.frPollReply.lbScaNr -text "SCA#" ] -column 2 -row 2 -sticky nswe
grid [Label $win_top.frPollReply.lbCh -text "CH#"  ] -column 3 -row 2 -sticky nswe

grid [ComboBox $win_top.frPollReply.cbGbtNr -values $gbt_number_list -modifycmd SetReplyAddress  -width
  5 ] -column 1 -row 3 -sticky nswe
grid [ComboBox $win_top.frPollReply.cbScaNr -values $sca_number_list -modifycmd SetReplyAddress -width
  5 ] -column 2 -row 3 -sticky nswe
grid [ComboBox $win_top.frPollReply.cbCh -values $channel_list -modifycmd SetReplyAddress -width 5 ]
  -column 3 -row 3 -sticky nswe


grid [Label $win_top.frPollReply.lbNotUsed -text "00" -width 5 ] -column 4 -row 3 -sticky nswe

grid [checkbutton $win_top.frPollReply.chkPoll  -text "Poll"  -variable Poll -justify left -width 5]
  -column 5 -row 3  -sticky nswe


grid [entry $win_top.frPollReply.txtReplyAddr -width 10  -justify right -relief sunken -textvariable
  ecs_reply.Address -state disabled ] -column 6 -row 3 -sticky nswe

grid [Label $win_top.frPollReply.lbStatusReply -text "Idle"] -column 1 -row 4  -columnspan 6 -sticky
  nswe

grid [Button $win_top.frPollReply.btnGetReply -text "Get Reply" -command { GetReply }] -column 1 -row 6
   -columnspan 8 -sticky nswe


# ------------------------------------------------------------------------------
# ------------------------------------------------------------------------------

frame $win_top.frEcsReply -borderwidth 2 -relief raised
grid $win_top.frEcsReply  -column 1 -row 3


grid [Label $win_top.frEcsReply.lbEcsReplyFrame -text "ECS Reply" ] -column 1 -row 1 -columnspan 6
  -sticky nswe
```

```
grid [Label $win_top.frEcsReply.lbGbtNr -text "GBT#" ] -column 1 -row 2 -sticky nswe
grid [Label $win_top.frEcsReply.lbScaNr -text "SCA#" ] -column 2 -row 2 -sticky nswe
grid [Label $win_top.frEcsReply.lbCh -text "CH#"  ] -column 3 -row 2 -sticky nswe
grid [Label $win_top.frEcsReply.lbCmd -text "CMD" ] -column 4 -row 2 -columnspan 2 -sticky nswe



grid [ComboBox $win_top.frEcsReply.cbGbtNr -values $gbt_number_list  -state disabled -width 5 ] -column
  1 -row 3 -sticky nswe
grid [ComboBox $win_top.frEcsReply.cbScaNr -values $sca_number_list  -state disabled -width 5 ] -column
  2 -row 3 -sticky nswe
grid [ComboBox $win_top.frEcsReply.cbCh -values $channel_list -state disabled -width 5 ] -column 3 -row
  3 -sticky nswe
grid [ComboBox $win_top.frEcsReply.cbCmd -values $cmd_list -state disabled -width 15] -column 4 -row 3
  -columnspan 2 -sticky nswe

# set cbCmd [ComboBox $win_top.frEcsCommand.cbCmd -textvariable $cmd_list -modifycmd
  {UpdateCommandHeader0 } -width 15 ]
# grid $cbCmd -column 4 -row 3 -columnspan 2 -sticky nswe

grid [entry $win_top.frEcsReply.txtCommandHeader0 -width 10  -justify right -relief sunken
  -textvariable ecs_reply.data(CommandHeader0) -validatecommand {validate_isxdigit %d %P} -validate all
  -state disabled ] -column 6 -row 3  -columnspan 2 -sticky nswe
#

grid [Label $win_top.frEcsReply.lbError -text "Error" ] -column 1 -row 4
grid [Label $win_top.frEcsReply.lbLen -text "Length" ] -column 2 -row 4
grid [Label $win_top.frEcsReply.lbProtSpecific -text "Protocol Specific" ] -column 3 -row 4 -columnspan
  2 -sticky nswe

grid [Label $win_top.frEcsReply.lbErrorVal -text "No Error" -bg "green" ] -column 1 -row 5 -sticky nswe
grid [ComboBox $win_top.frEcsReply.cbLen -values $sca_number_list -modifycmd {puts "[lindex
  $sca_number_list [$win.frEcsReply.cbScaNr getvalue]]"}  -state disabled -width 5 ] -column 2 -row 5
  -sticky nswe
grid [ComboBox $win_top.frEcsReply.cbProtSpecific -values $cmd_list -modifycmd {puts "[lindex $cmd_list
  [$win.frEcsReply.cbCmd getvalue]]"} -width 5 -state disabled ] -column 3 -row 5 -columnspan 3 -sticky
  nswe


grid [entry $win_top.frEcsReply.txtCommandHeader1 -width 10  -justify right -relief sunken
  -textvariable ecs_reply.data(CommandHeader1) -validatecommand {validate_isxdigit %d %P} -validate all
  -state disabled ] -column 6 -row 5 -columnspan 2 -sticky nswe
#

#

grid [Label $win_top.frEcsReply.lbD3 -text "D3" ] -column 1 -row 6
grid [Label $win_top.frEcsReply.lbD2 -text "D2" ] -column 2 -row 6
grid [Label $win_top.frEcsReply.lbD1 -text "D1" ] -column 3 -row 6
grid [Label $win_top.frEcsReply.lbD0 -text "D0" ] -column 4 -row 6

grid [entry $win_top.frEcsReply.txtD3 -state disabled -width 5 -justify right -relief sunken
  -textvariable  ecs_reply.data(D3) ] -column 1 -row 7 -sticky nswe
grid [entry $win_top.frEcsReply.txtD2 -state disabled -width 5 -justify right -relief sunken
  -textvariable  ecs_reply.data(D2) ] -column 2 -row 7 -sticky nswe
grid [entry $win_top.frEcsReply.txtD1 -state disabled -width 5 -justify right -relief sunken
  -textvariable  ecs_reply.data(D1) ] -column 3 -row 7 -sticky nswe
grid [entry $win_top.frEcsReply.txtD0 -state disabled -width 5 -justify right -relief sunken
```

```
  -textvariable  ecs_reply.data(D0) ] -column 4 -row 7 -sticky nswe

grid [entry $win_top.frEcsReply.txtData0 -width 10  -justify right -relief sunken -textvariable
  ecs_reply.data(Data0) -state disabled ] -column 6 -row 7 -sticky nswe

#

#

grid [Label $win_top.frEcsReply.lbD7 -text "D7" ] -column 1 -row 8
grid [Label $win_top.frEcsReply.lbD6 -text "D6" ] -column 2 -row 8
grid [Label $win_top.frEcsReply.lbD5 -text "D5" ] -column 3 -row 8
grid [Label $win_top.frEcsReply.lbD4 -text "D4" ] -column 4 -row 8


grid [entry $win_top.frEcsReply.txtD7 -state disabled -width 5 -justify right -relief sunken
  -textvariable  ecs_reply.data(D7) ] -column 1 -row 9 -sticky nswe
grid [entry $win_top.frEcsReply.txtD6 -state disabled -width 5 -justify right -relief sunken
  -textvariable  ecs_reply.data(D8) ] -column 2 -row 9 -sticky nswe
grid [entry $win_top.frEcsReply.txtD5 -state disabled -width 5 -justify right -relief sunken
  -textvariable  ecs_reply.data(D9) ] -column 3 -row 9 -sticky nswe
grid [entry $win_top.frEcsReply.txtD4 -state disabled -width 5 -justify right -relief sunken
  -textvariable  ecs_reply.data(D10) ] -column 4 -row 9 -sticky nswe

grid [entry $win_top.frEcsReply.txtData1 -width 10  -justify right -relief sunken -textvariable
  ecs_reply.data(Data1) -state disabled ] -column 6 -row 9 -sticky nswe

#
grid [Label $win_top.frEcsReply.lbD11 -text "D11" ] -column 1 -row 10
grid [Label $win_top.frEcsReply.lbD10 -text "D10" ] -column 2 -row 10
grid [Label $win_top.frEcsReply.lbD9 -text "D9" ] -column 3 -row 10
grid [Label $win_top.frEcsReply.lbD8 -text "D8" ] -column 4 -row 10


grid [entry $win_top.frEcsReply.txtD11 -state disabled -width 5 -justify right -relief sunken
  -textvariable  ecs_reply.data(D11) ] -column 1 -row 11 -sticky nswe
grid [entry $win_top.frEcsReply.txtD10 -state disabled -width 5 -justify right -relief sunken
  -textvariable  ecs_reply.data(D12) ] -column 2 -row 11 -sticky nswe
grid [entry $win_top.frEcsReply.txtD9 -state disabled -width 5 -justify right -relief sunken
  -textvariable  ecs_reply.data(D13) ] -column 3 -row 11 -sticky nswe
grid [entry $win_top.frEcsReply.txtD8 -state disabled -width 5 -justify right -relief sunken
  -textvariable  ecs_reply.data(D14) ] -column 4 -row 11 -sticky nswe

grid [entry $win_top.frEcsReply.txtData2 -width 10  -justify right -relief sunken -textvariable
  ecs_reply.data(Data2) -state disabled ] -column 6 -row 11 -sticky nswe
#

grid [Label $win_top.frEcsReply.lbD15 -text "D15" ] -column 1 -row 12
grid [Label $win_top.frEcsReply.lbD14 -text "D14" ] -column 2 -row 12
grid [Label $win_top.frEcsReply.lbD13 -text "D13" ] -column 3 -row 12
grid [Label $win_top.frEcsReply.lbD12 -text "D12" ] -column 4 -row 12


grid [entry $win_top.frEcsReply.txtD15 -state disabled -width 5 -justify right -relief sunken
  -textvariable  ecs_reply.data(D15) ] -column 1 -row 13 -sticky nswe
grid [entry $win_top.frEcsReply.txtD14 -state disabled -width 5 -justify right -relief sunken
  -textvariable  ecs_reply.data(D14) ] -column 2 -row 13 -sticky nswe
grid [entry $win_top.frEcsReply.txtD13 -state disabled -width 5 -justify right -relief sunken
  -textvariable  ecs_reply.data(D13) ] -column 3 -row 13 -sticky nswe
grid [entry $win_top.frEcsReply.txtD12 -state disabled -width 5 -justify right -relief sunken
```

```
     -textvariable  ecs_reply.data(D12) ] -column 4 -row 13 -sticky nswe

   grid [entry $win_top.frEcsReply.txtData3 -width 10  -justify right -relief sunken -textvariable
     ecs_reply.data(Data3) -state disabled ] -column 6 -row 13 -sticky nswe
   #

   #frame $win_top.frEcsCommmandActions
   #grid $win_top.frEcsCommmandActions  -column 2 -row 2



}


proc draw_client_status {win} {
   global jtag

   # Create a frame so that elements can be packed
   # and some padding added
   frame $win.f1

   # Title
   label $win.f1.l1 -text "Client connection status:"

   # Connection status text
   set jtag(port_status) "Disconnected"
   label $win.f1.l2 -textvariable jtag(port_status) \
      -width 15 -relief groove \
      -padx 10 -pady 5 -background yellow

   # Port number text
   label $win.f1.l3 -text "Port number:"

   # An numeric entry box with the default port number
   entry $win.f1.e1 -textvariable jtag(port) \
      -width 12 -justify right \
      -relief sunken -validate key \
      -validatecommand {validate_isdigit %d %P}

   # Copy the label path so the color can be updated
   set jtag(port_status_label) $win.f1.l2

   # A connect/disconnect button
   button $win.f1.b1 -text "Connect" \
      -command connect_button -padx 10 -width 14

   # Copy the button path so the text can be updated
   set jtag(port_status_button) $win.f1.b1

   # Arrange the elements into their frame
   #
   # The title
   grid $win.f1.l1 -column 0 -columnspan 2 -row 0 -sticky w -padx 5 -pady 2
   #
   # Connection status
   grid $win.f1.l2 -column 2 -row 0 -sticky w -padx 5 -pady 2
   #
   # The port connect elements
   grid $win.f1.l3 -column 0 -row 1 -sticky w -padx 5 -pady 2
   grid $win.f1.e1 -column 1 -row 1 -sticky w -padx 5 -pady 2
```

```
   grid $win.f1.b1 -column 2 -row 1 -sticky w -padx 5 -pady 2
   pack $win.f1    -padx 10 -pady 10
}

proc draw_read_write {win} {
   global jtag

   # Create a frame so that elements can be packed
   # and some padding added
   frame $win.f1

   # Title
   label $win.f1.l1 -text "JTAG read/write:"

   # Address text
   label $win.f1.l2 -text "Address (hex):"

   # A hex entry box with the default address
   set jtag(address) 0
   entry $win.f1.e1 -textvariable jtag(address) \
      -width 12 -justify right \
      -relief sunken -validate key \
      -validatecommand {validate_isxdigit %d %P}

   # Data text
   label $win.f1.l3 -text "Data (hex):"

   # A hex entry box with the default data
   set jtag(data) 0
   entry $win.f1.e2 -textvariable jtag(data) \
      -width 12 -justify right \
      -relief sunken -validate key \
      -validatecommand {validate_isxdigit %d %P}

   # Read/write buttons
   button $win.f1.b1 -text "Read" \
      -command read_button -padx 10 -width 14
   button $win.f1.b2 -text "Write" \
      -command write_button -padx 10 -width 14

   # Arrange the elements into their frame
   #
   # The title
   grid $win.f1.l1 -column 0 -columnspan 3 -row 0 -sticky w -padx 5 -pady 2
   #
   # Text + entry + button
   grid $win.f1.l2 -column 0 -row 1 -sticky e -padx 5 -pady 2
   grid $win.f1.e1 -column 1 -row 1 -sticky w -padx 5 -pady 2
   grid $win.f1.b1 -column 2 -row 1 -sticky w -padx 5 -pady 2
   grid $win.f1.l3 -column 0 -row 2 -sticky e -padx 5 -pady 2
   grid $win.f1.e2 -column 1 -row 2 -sticky w -padx 5 -pady 2
   grid $win.f1.b2 -column 2 -row 2 -sticky w -padx 5 -pady 2
   pack $win.f1    -padx 10 -pady 10
}



proc create_gui {win} {
    global jtag
```

```
    # Top-of-window menu bar
    # ----------------------
    #frame $win.mbar -borderwidth 1 -relief raised
   #draw_menu $win.mbar
   # The menu bar fills the window width
    #pack $win.mbar -fill x

    # Client status
    # ---------------
    frame $win.port -borderwidth 2 -relief raised
   draw_client_status $win.port
    grid $win.port -column 2 -row 1

    # JTAG read/write
    # ---------------
    frame $win.rw -borderwidth 2 -relief raised
   draw_read_write $win.rw
    grid $win.rw -column 2 -row 2

   # Change the title of the main frame

   wm title $win "SOL40_SCA evaluation tool"


   # # Turn off window resizing
   # # * first need to give it a reasonable size
   # wm minsize $win 400 250

   create_gui_ecs

   wm resizable $win 0 0

   # Connect the close (X) button to the gui_quit handler
   wm protocol $win WM_DELETE_WINDOW [list gui_quit $win]

   # Execute socket_trace when jtag(socket) changes
   trace variable jtag(socket) wu socket_trace

}


# ------------------------------------------------------------------
# GUI construction
# ------------------------------------------------------------------
#
# Create a top-level window in which to pack widgets
#  * Standard Tcl/Tk has the window . that can be used
#  * Quartus hides the top-level . window, so create a new one
#  * Modelsim uses the top-level . window, so create a new one
#
if {([string compare $jtag(tool) wish] == 0) ||
    ([string compare $jtag(tool) quartus_stp] == 0)} {
   # Hide the top-level window
   wm state . withdraw
}

# Create a new top-level window called .top
if {[winfo exists .top]} {
   # Quartus and wish can be used to start multiple GUIs
```

```
   # but that is not allowed as the socket trace gets messed up
   # (there is only one jtag(socket) variable)
   #
   # Note:
   #   puts " ", with a space between the quotes is needed
   #   under Quartus Tcl, otherwise the blank line is
   #   suppressed.
   #
   puts " ==> Error: the client GUI is already running!"
   puts " "
   puts "     If you want multiple clients, they must be started"
   puts "     using multiple processes."
   puts " "
   return
}


# Note: the variable 'top' is used below for the tkwait
set top [toplevel .top]

set win_top $top

create_gui $top



if {[string first "quartus_" $jtag(tool)] == 0} {
   # Quartus command-line tool-specific wait
   tkwait window $top
} elseif {[string compare $jtag(tool) wish] == 0} {
   # Display the Tcl console
   if {$jtag(console) == 1} {
      console show
   } else {
      console hide
   }
}




# if [expr {[llength [info commands wm]] > 0}] {
   # wm title .top "SOL40_SCA evaluation tool"
# }

# . configure  -padx 0 -pady 0 -takefocus 0

# tkwait window .top




# scale .top.scScale -from 10000 -orient vertical -to 0 -variable ScaleVal
# grid .top.scScale -column 2 -row 1
# listbox .top.lbGBT -exportselection true -selectmode single -setgrid true -listvariable gbt_list
# grid .top.lbGBT
# scrollbar .top.scrlBar
# grid .top.scrlBar
# scale .top.scScale1 -from 10000 -orient vertical -to 0 -variable ScaleVal
# grid .top.scScale1 -column 3 -row 1
```

```
# scale .top.scScale2 -from 10000 -orient vertical -to 0 -variable ScaleVal
# grid .top.scScale2 -column 4 -row 1
# set mb [menubutton .top.mbtnMenuButton -borderwidth 1 -justify left -state active -text ertertfdgfdsg
    -textvariable lol]
# grid .top.mbtnMenuButton
# set m [menu $mb.menu -tearoff 0]
# $mb configure -menu $m
# $m add radiobutton -label "Cut"  -variable lol
# $m add radiobutton -label "Open"  -variable lol
# $m add radiobutton -label "hue"  -variable lol
# $m add radiobutton -label "lol"  -variable lol
# $m add radiobutton -label "hua"  -variable lol
# set lol  Cut

#message .top.msgMEssage -text fdgfdgfdgdfg
# grid .top.msgMEssage

#wm resizable .top 800 600



#}




# Decimal to hex using format
#set inhex [format %x $initial]

# Hex to decimal using expr
#set back_in_decimal [expr 0x$inhex]
```