CENTRO BRASILEIRO DE PESQUISAS FÍSICAS

LAFEX - LABORATÓRIO DE FÍSICA EXPERIMENTAL

Desenvolvimento de Sistema de Controle para Eletrônica de Front-End do Detector de Fibras Cintilantes do LHCb

Maurício Féo Pereira Rivello de Carvalho

Dissertação de mestrado apresentada ao Centro Brasileiro de Pesquisas Físicas como parte dos requisitos necessários à obtenção do título de Mestre em Física.

Orientador: Prof. Dr. André Massafferri Rodrigues

Rio de Janeiro - RJ

Maio, 2016

Agradecimentos

À Raquel, meu incentivo, minha alegria, minha vida.

Aos meus pais, Osvaldo e Laís e meus sogros, Delio e Ermitas, que foram o meu suporte durante este período.

Ao meu orientador, André Massafferri, por todas as oportunidades e a confiança depositada em mim.

Ao Antonio Pellegrino, por proporcionar a continuação deste caminho.

Ao Federico Alessio, pela supervisão e orientação.

Aos meus colegas Leonardo Guedes e Cairo Caplan, por ter tornado esta jornada divertida.

A todos os que colaboraram direta ou indiretamente para esta realização.

Resumo

O experimento LHCb está se preparando para seu upgrade, que envolverá a troca de seu sistema de aquisição de dados e de grande parte de seus detectores e eletrônica adjacente.

O CBPF se insere neste contexto como o responsável pelo desenvolvimento de um sistema de teste para validação dos módulos eletrônicos e módulos de fibras cintilantes antes da fase de transporte e montagem do detector na área experimental subterrânea.

O presente projeto se caracteriza pela validação do módulo eletrônico MiniDAQ para ser utilizado como plataforma de aquisição de dados no sistema de teste, e o desenvolvimento de um sistema supervisório de controle e monitoramento para o sistema citado, cujo objetivo é configurar e monitorar todos os dispositivos da eletrônica de *front-end* a ser testada assim como os demais componentes do sistema, fornecendo painéis virtuais criados no software SCADA WinCC para possibilitar a criação de rotinas de teste para o futuro detector.

Abstract

The LHCb experiment is preparing for it's upgrade, which will involve the replacement of it's data acquisition system and most of its adjacent detectors and electronics.

The CBPF is inserted in this context as the responsible for the development of a test system for validation of electronic modules and scintillating fiber modules before the phase of shipping and assembly of the detector in the underground experimental area.

This project is characterized by the validation of the electronic module MiniDAQ to be used as a data acquisition platform in the test system, and the development of a supervisory monitoring and control system for the said platform, whose aim is to set up and monitor all devices electronic front-end to be tested as well as other system components, providing virtual panels created in the WinCC SCADA software to allow the creation of test routines for the future detector.

Sumário

Capítulo 1

Introdução

O "*Large Hadron Collider*" (LHC) é um acelerador de partículas, de 27km de perímetro, projetado para colidir prótons a uma energia de 14TeV e luminosidade de $10^{34} cm^{-2} s^{-1}$. Tendo iniciado sua operação no ano de 2009 até sua primeira interrupção programada de 2013 a 2015, os experimentos do LHC acumularam dados suficientes para, em análise posterior, possibilitar importantes descobertas como a do Bóson de Higgs (2012), do decaimento $B_s^0 \rightarrow \mu^+\mu^-$ (2012) e do pentaquark (2015). Operado pela Organização Europeia para Pesquisa Nuclear (CERN), o LHC já está novamente em operação. No entanto, o CERN já trabalha no próximo upgrade do acelerador, a ser instalado na próxima interrupção técnica de 2019 a 2020, e que proporcionará um aumento na sua luminosidade de 5 a 10 vezes a original.

O aumento da luminosidade provocada pelo upgrade do LHC trará consequências aos experimentos. O nível de radiação incidente se tornará substancialmente maior, exigindo a revalidação de todos os componentes localizados nas áreas expostas à radiação das colisões. O volume de dados produzidos também será significantemente maior. O projeto GBT visa oferecer uma solução comum aos problemas citados para todos os experimento. Este fornece um conjunto de componentes para estabelecer links ópticos que transmitam dados a uma taxa de 4.7GBps e que sejam resistentes a radiação.

O experimento "*Large Hadron Collider beauty*" (LHCb) também sofrerá *upgrades* para se adequar ao novo regime de luminosidade do LHC. O nível de ocupação no detector *Outer Tracker*, do sistema de trajetografia, se elevará além do máximo que o algoritmo de reconstrução de trajetórias suporta, exigindo a substituição completa deste detector por um de melhor resolução. Decidiu-se pela substituição completa das estações de trajetória

T1, T2 e T3 por um novo detector a base de fibras cintilantes (SciFi). Com o novo detector, também foi necessário desenvolver nova eletrônica de *front-end*.

Para lidar com o grande volume de dados, foi decidido que será substituída toda a eletrônica de aquisição do experimento, que registra eventos a uma taxa de no máximo 1.1MHz, por uma que fosse capaz de registrar todos os eventos gerados, à taxa de 40MHz que ocorrem os eventos. O sistema de trigger, que era composto por um trigger a nível de hardware (trigger de nível 0) para filtrar os eventos lidos pela eletrônica de aquisição, e um trigger a nível de software (trigger de alto nível) também foi reformulado. O trigger de nível 0 foi completamente extinto, uma vez que todo evento seria enviado para aquisição.

Inicialmente se escolheu utilizar uma arquitetura semelhante ao módulo atual do experimento, chamado de TELL1, em que haveria um módulo mãe com 4 conectores AMC onde seriam inseridos mezaninos. Estes mezaninos, chamados de TELL40, poderiam ser utilizados para aquisição de dados ou para controle, dependendo do firmware utilizado no FPGA. Eles possuem links GBTs e são capazes de fazer a aquisição a 40MHz. Os módulos TELL40 foram desenvolvidos e algumas unidades foram produzidas e distribuídas em um kit para desenvolvimento chamado MiniDAQ.

Posteriormente foi decidido se utilizar uma arquitetura diferente no experimento, e a idéia da TELL40 foi substituída pela PCIe40. Estas placas consistem em módulos semelhantes à TELL40 mas que se conectam diretamente aos computadores de aquisição de dados, através de uma interface PCIexpress. No entanto, para algumas aplicações que não tem os mesmos requisitos do experimento, continuou-se a utilizar o MiniDAQ. Dentre elas está o sistema de teste do detector SciFi.

O sistema de teste da eletrônica de *front-end* do SciFi ficou sob responsabilidade do Centro Brasileiro de Pesquisas Físicas (CBPF) quando, em outubro de 2013, o grupo de sabores pesados do Laboratório de Física Experimental (LAFEX) apresentou uma "Expressão de Interesse" em desenvolver e construir um sistema de testes automatizado, para validar o correto funcionamento do detector e seus componentes antes que estes fossem transportados para instalação no LHCb.

O sistema de teste compreende um módulo injetor de sinais, para se ligar à *front-end* a fim de emular os sinais do detector, e um módulo de aquisição de dados com a finalidade de validar se a *front-end* efetuou corretamente as etapas de leitura e processamento dos dados. O módulo injetor será produzido pelo CBPF, enquanto o módulo de aquisição será o MiniDAQ.

O objetivo do projeto detalhado nesta dissertação é a validação e preparo do módulo de aquisição (MiniDAQ) para uso no sistema de teste, e a criação de um sistema de controle capaz de configurar individualmente os componentes da *front-end*, utilizando o software de criação de sistemas supervisórios WinCC. Espera-se com os resultados obtidos, que se tenha todas as ferramentas para desenvolver as rotinas automatizadas de teste quando os componentes da *front-end* e o módulo injetor estiverem prontos para produção.

O MiniDAQ, no momento de início do presente trabalho, havia sido produzido recentemente e ainda se encontrava muito instável, com algumas funções em seu software e firmware não funcionando corretamente em ambientes diferentes, e o grupo do CBPF foi o primeiro usuário do módulo. O objetivo de validar o MiniDAQ incluía cooperar, no que fosse possível, para o lançamento de uma versão estável o suficiente para uso no sistema de teste. Isto demandou um extenso trabalho de depuração, relatando os erros e eventualmente possíveis soluções para os desenvolvedores. As correções que este trabalho proporcionou ao MiniDAQ também se estenderiam à PCIe40, por isto o trabalho de depuração também ajudou no desenvolvimento da mesma. O trabalho com o MiniDAQ é detalhado na seção 8.

De posse de um hardware estável e funcional, iniciou-se o desenvolvimento do sistema de controle para o sistema de teste. Enquanto a eletrônica de *front-end* do detector ainda estava em desenvolvimento, foram utilizadas placas alternativas que continham os mesmos componentes a fim de otimizar o tempo de trabalho. Quando o primeiro protótipo da *front-end* foi produzido, em uma semana era possível configurar todos os chips contidos na mesma através do sistema de controle desenvolvido. O desenvolvimento do sistema de controle para o sistema de teste é detalhado na seção 9.

Os capítulos 2 ao 7 tem por finalidade contextualizar o leitor sobre o detector e seus módulos para os quais o sistema de teste está sendo desenvolvido. Os resultados obtidos com a utilização dos elementos descritos neste trabalho são encontrados no capítulo 10 e a conclusão e perspectivas futuras no capítulo 11.

Capítulo 2

O Grande Colisor de Hádrons e seus Experimentos

O Grande Colisor de Hadrons (LHC) [1] é o principal acelerador e colisor de partículas do CERN. Localizado na fronteira entre a Suíça e a França, o LHC possui 27 Km de extensão e se encontra em um túnel que varia de 50 a 175 metros de profundidade. O LHC foi projetado para realizar colisões próton-próton e colisões entre íons pesados, a uma energia de até 14 TeV e luminosidade de até $10^{34} cm^{-2} s^{-1}$. As colisões ocorrem em 4 pontos específicos e ao redor destes pontos se encontram aparatos de medição que constituem os experimentos do LHC. O LHC possui 7 experimentos: ATLAS [2], CMS [3], LHCb [4], ALICE [5], TOTEM [6], LHCf [7] e MoEDAL.

Objetivo O objetivo do LHC é proporcionar colisões de partículas a energia e luminosidade suficientes para viabilizar a seus experimentos a execução de uma vasta gama de medidas relacionadas a física de partículas elementares a fim de acumular estatística suficiente para responder a questões fundamentais ainda em aberto ou não contempladas pelo modelos atuais que descrevem as interações físicas conhecidas. Entre os tópicos que o LHC visa estudar podemos citar a existência do Bóson de Higgs e suas propriedades, a teoria de supersimetria, a natureza da matéria escura, a violação da simetria entre matéria e anti-matéria e a existência de dimensões extras.

Os Experimentos Os experimentos ATLAS (*A Toroidal LHC Apparatus*) e CMS (*Compact Muon Solenoid*) são experimentos de propósito geral, que possibilitaram a descoberta

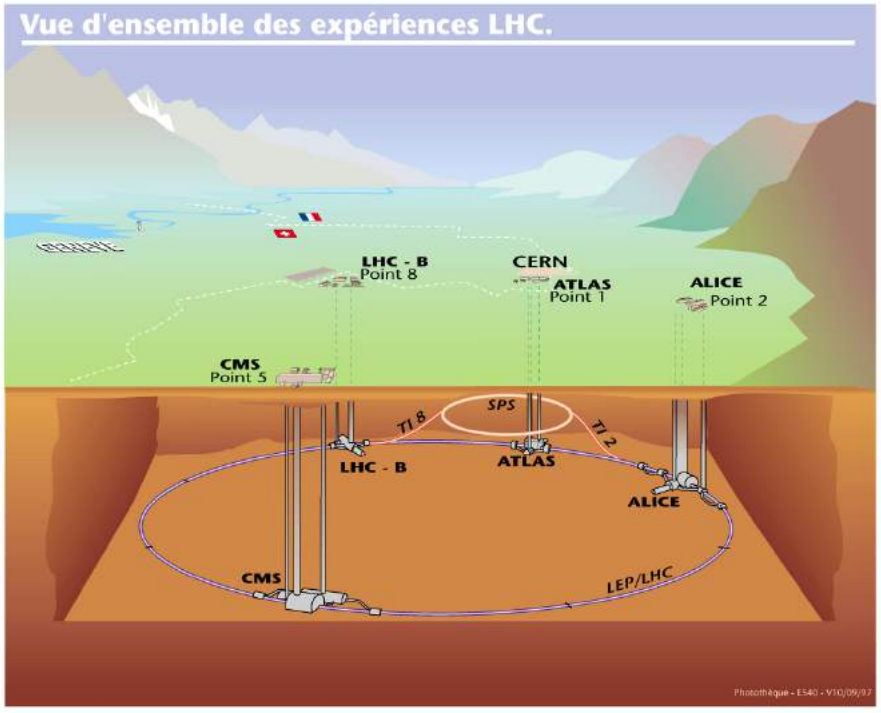

Figura 2.1: Representação do túnel do LHC e seus 4 principais experimentos.

do Bóson de Higgs no ano de 2012 a partir dos dados acumulados no primeiro *run* do LHC, e que visam também a busca de fenômenos inconsistentes com o Modelo Padrão.

O experimento ALICE (*A Large Ion Collider Experiment*) tem por principal objetivo estudar a física de um estado da matéria chamado de plasma de quark-glúons, em que se acredita que o universo se encontrava logo após o *Bigbang*, e que o LHC tenta reproduzir através da colisão de íons de chumbo. O ALICE também estuda colisões próton-próton para efeitos de comparação com os resultados de íons pesados e outras áreas em que seus resultados são competitivos com os demais experimentos.

O experimento LHCb (*Large Hádron Collider Beauty*) tem por objetivo principal o estudo da violação de simetria Carga-Paridade (CP) nas interações de hádrons B. Tal estudo poderia ajudar a explicar a assimetria de matéria e anti-matéria encontrada no Universo. O LHCb foi também o experimento responsável pela descoberta do Pentaquark em 2015. O experimento LHCb é discutido com mais detalhes na seção 3.

O experimento TOTEM (*TOTal Elastic and diffractive cross section Measurement*) visa o estudo da estrutura do próton através de medições da seção de choque das interações

proton-proton, sua dispersão elástica e processos difrativos.

O experimento LHCf (*LHC forward*) reliza a medição precisa de partículas neutras originadas das colisões numa direção bem próxima ao feixe, cujo ângulo é próximo de 0°. O estudo visa o melhor entendimento da interação de raios cósmicos com a atmosfera terrestre.

O experimento MoEDAL (*the Monopole and Exotics Detector At the LHC*) é dedicado à procura de partículas bem específicas, estáveis e exóticas, como monopolos magnéticos, provenientes das colisões próton-próton.

As colisões O LHC possui dois tubos paralelos onde dois feixes de prótons são conduzidos em direções opostas, próximos à velocidade da luz, com energia de até 7TeV cada. Em 4 pontos específicos, a uma frequência de 40MHz (a cada 25ns), os 2 feixes se cruzam ocasionando as colisões com energia total de 14TeV. Cada feixe é composto por 2808 *bunches*, que trafegam no vácuo existente nos tubos, da ordem de 10^{-10} a 10^{-11} mbar. Na Figura 2.2 é possível ver os dois tubos por onde trafegam os feixes.

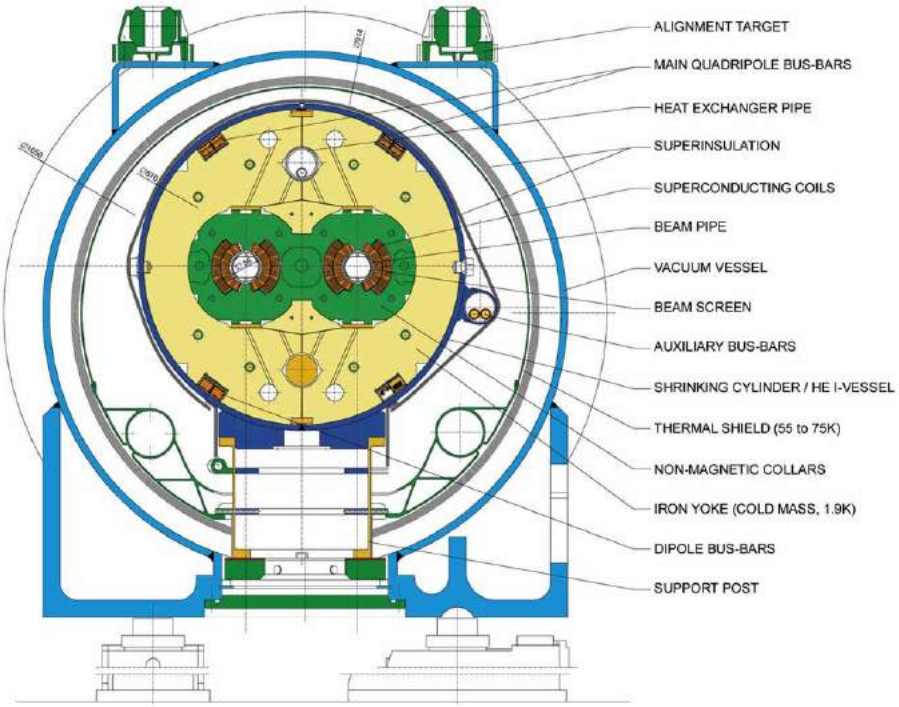

Figura 2.2: Corte transversal da vista frontal de um módulo dipolo do LHC.

Ao longo do tubo circular do LHC, os *bunches* são conduzidos, colimados e ace-

lerados por módulos magnéticos diferentes. O módulo de dipolo magnético é responsável por manter a curvatura do feixe. Este possui um eletro-ímã supercondutor de 15 metros, localizado ao redor de cada tubo, que provê um campo magnético de 8,33T. Para fornecer este campo, o ímã consome uma corrente de 13.000A e é mantido a uma temperatura de 1.9K pelo sistema de criogenia. Mais de 60% da extensão do LHC é composto por estes módulos, totalizando 1232 módulos. A Figura 2.2 mostra o corte transversal de um módulo dipolo. O módulo responsável pelo foco e colimação do feixe é o módulo quadripolo. Já a manutenção e aceleração dos *bunches* é executada pelas cavidades de radiofrequência, cuja frequência de alternação é sincronizada com os *bunches* de forma a sempre propiciar aceleração na região onde os bunches devem estar e uma força contrária nas regiões intermediárias, tendendo a conduzir desta forma os prótons para a região central dos *bunches*.

Cadeia de aceleração Os feixes de prótons que circulam no LHC tem sua origem em um cilindro de gás hidrogênio. Os prótons são separados no gás utilizando um equipamento chamado Duoplasmatron, que ioniza o gás através da aplicação de um forte campo elétrico. Os prótons deixam o Duoplasmatron a uma velocidade de 1,4% da velocidade da luz e são enviados a um quadripolo de rádio frequência, que foca o feixe e o acelera até 750keV. Este feixe é inserido em um acelerador linear denominado LINAC2, onde são acelerados até uma energia de 50MeV e então inseridos no "*Proton Synchrotron Booster*" (PSB). O PSB é um acelerador circular de 4 anéis, o qual acelera o feixe até 1,4GeV. Do PSB, o feixe é inserido no "*Proton Synchrotron*" (PS) o qual divide o feixe em 81 *bunches* espaçados e os acelera até 26GeV. Os *bunches* são então injetados no "*Super Proton Synchrotron*" (SPS), que os acelera até 450GeV. Os prótons levam, da fonte até a saída do SPS, um tempo inferior a 18s e então são injetados no LHC. Esta injeção ocorre por dois pontos diferentes, cada um destes levando a um dos dois tubos do LHC, inserindo os *bunches* em sentidos opostos em cada tudo. No LHC, os feixes podem ser acelerados até 7TeV. Conforme veremos no cronograma da Figura 4.1, o LHC até o ano de 2015 operou com uma energia máxima de 6,5TeV por feixe. Planeja-se alcançar colisões de 14TeV até o final do *run* de 2018.

Após acelerados até a energia desejada, os feixes de sentidos opostos começam então a serem colididos nos quatro pontos de cruzamento já citados. A Figura 2.3 ilustra as etapas de aceleração descritas, mostrando à esquerda a energia alcançada pelo feixe ao fim de cada etapa.

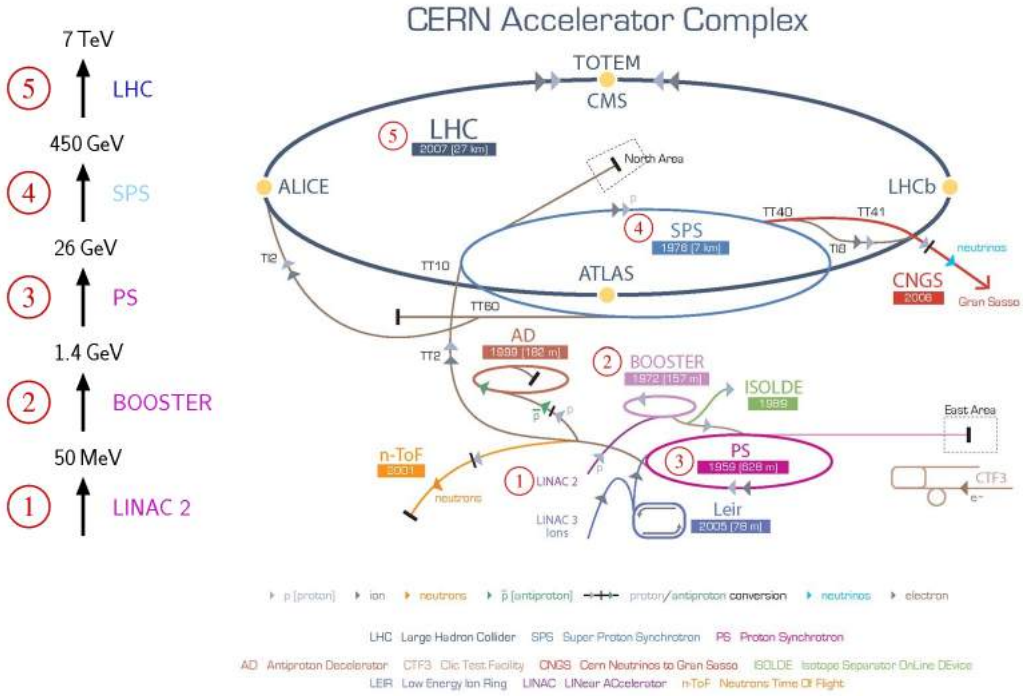

Figura 2.3: Cadeia de aceleração do LHC.

Capítulo 3

O Experimento LHCb

O experimento *Large Hadron Collider beauty* (LHCb) é um espectrômetro de braço único que cobre a região limitada pelos ângulos de 10mrad até 300mrad no plano horizontal e 10mrad até 250mrad no plano vertical a partir do ponto de colisão. O LHCb se encontra no local denominado como Ponto 8 do LHC, em uma região experimental escavada em torno de um dos 4 pontos de colisão. Ele é composto por um conjunto de detectores e elementos que permitem a identificação de partículas e reconstrução de sua trajetória. Dentre estes, os principais são: o Localizador de Vértice (VeLo), o Sistema de Trajetografia, o Sistema de Múons, o Magneto, os Calorímetros Hadrônicos e Eletromagnéticos, e os detectores *Ring-Imaging Cherenkov* (RICH).

O objetivo do LHCb é estudar a violação de simetria carga-paridade e processos raros no decaimento de hádrons B produzidos a partir das colisões do LHC.

O LHCb mede 21m de comprimento, por 10m de altura e 13m de largura. Este possui uma geometria frontal, se extendendo na direção do feixe conforme angulação já mencionada, devido à natureza dos hádrons B, que são gerados com momento em direção próxima à da colisão que o gerou. A fim de construir o detector mais eficiente possível com os recursos disponíveis, o experimento foi montado em apenas um dos sentidos da colisão. Desta forma, se perde a metade dos processos, mas sobra mais recursos para criação de um detector que gere resultados mais precisos. Na Figura 3.1 se pode ver a geometria do detector LHCb.

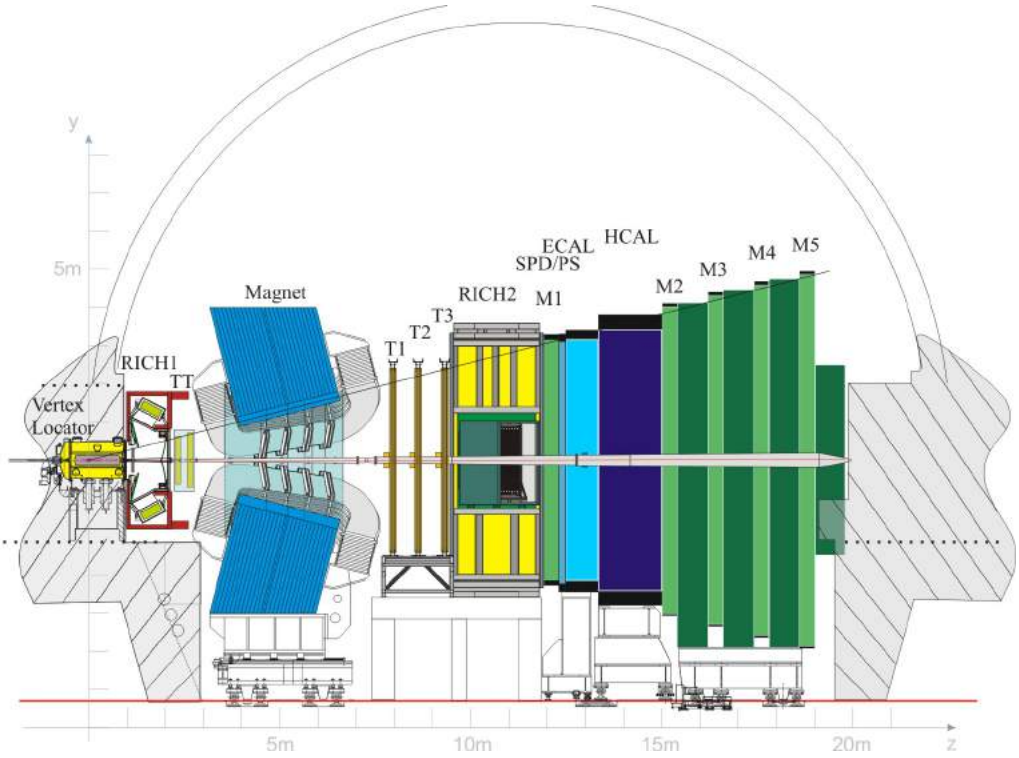

Figura 3.1: Vista lateral do detector do experimento LHCb.

3.1 VeLo

O Localizador de Vértice (VeLo) é o primeiro detector a interagir com o produto das colisões. Este é montado ao redor do ponto de colisão e tem por objetivo fazer a medição das coordenadas de trajetórias na região próxima ao ponto de colisão. As medidas efetuadas pelo VeLo são utilizadas na reconstrução das trajetórias e dos vértices de produção (primários) e de decaimento (secundários) dos mésons B. Os dados do VeLo também são utilizados na filtragem dos eventos, enriquecendo as amostras de mésons B selecionadas.

O VeLo é composto por semidiscos instalados ao redor da região de colisão, conforme mostrado na Figura 3.2. Os semidiscos são instalados em um recipiente de vácuo para separar o tubo do LHC da atmosfera da região do experimento. Eles são compostos por detectores de silício que proporcionam uma reconstrução de vértices primários com resolução da ordem de 40 μm na direção z e 10 μm na direção radial dos semidiscos ϕ. Para vértices secundários a resolução é um pouco pior e pode variar entre 150 μm e 300 μm dependendo do número de traços utilizados.

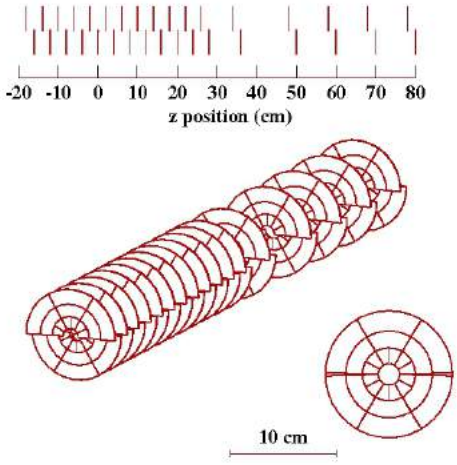

Figura 3.2: Geometria do VeLo. Ao todo são 34 semidiscos instalados ao redor da região de colisão.

3.2 Magneto

O LHCb possui um magneto cujo principal objetivo é gerar um campo magnético forte o suficiente para curvar as partículas carregadas criadas na colisão de forma que seja significativamente perceptível ao sistema de trajetografia.

Sabendo-se a intensidade do campo magnético e unindo às informações providas pelo sistema de trajetografia (descrito na seção 3.3) é possível medir o momento e a carga relacionada às partículas. Esta medida é crucial para identificação das partículas que atravessam o detector RICH.

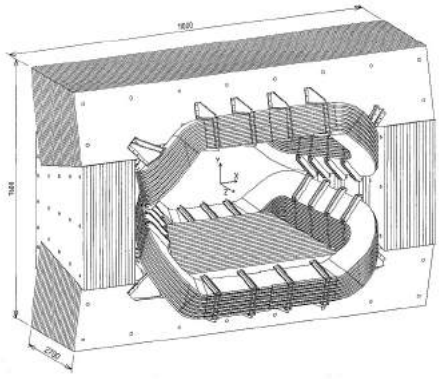

Figura 3.3: Geometria do magneto do LHCb.

O campo magnético é gerado no LHCb por um dipolo localizado próximo ao VeLo e, portanto, próximo à região de colisão. O poder de curvatura do magneto é da ordem de 4 Tm. Isto proporciona uma medição do momento das partículas de até $\delta p/p = 0.4\%$, com capacidade de manter o campo magnético uniforme dentro de uma margem de erro de aproximadamente 5% para partículas com ângulo polar entre 250 $mrad$ no plano vertical e 300 $mrad$ no plano horizontal.

3.3 Sistema de Trajetografia

O Sistema de Trajetografia é o sistema responsável por fornecer os dados necessários para reconstrução da trajetória de partículas carregadas que atravessam o experimento. Ele fornece uma estimativa de alta precisão do momento das partículas e também uma medida de massa de partículas instáveis. As informações de trajetória do sistema de trajetografia também auxiliam o detector RICH na identificação de partículas.

O Sistema de Trajetografia compreende o detector VeLo, as Estações de Trajetória e também utiliza o Magneto para obtenção da informação de momento e massa. Há 4 estações de trajetória no sistema: TT (*Trigger Tracker*), e as estações de trajetória T (T1, T2 e T3) que estão subdividas em *Inner Tracker* (IT) e *Outer Tracker* (OT). Na Figura 3.1 podemos ver discriminado a posição das estações TT, T1, T2 e T3 no LHCb.

Veremos a seguir que as estações de trajetória seguem um layout que se estende no plano vertical ou desviado de $\pm 5^\circ$ no plano vertical. Isto é feito para se otimizar a resolução espacial no eixo horizontal, em detrimento da resolução no eixo vertical. O eixo horizontal é o eixo de ação do Magneto sob as partículas, e esta estratégia permite medir com mais precisão o desvio causado nas partículas pelo campo magnético e, por consequência, melhorar indiretamente as medidas de momento e massa.

Trigger Tracker A estação TT está localizada antes do magneto e tem como principal objetivo reconstruir partículas de baixo momento que, quando entram na região do magneto, são desviadas para fora da aceptância do LHCb e, portanto, não atingem as estações de trajetória T. Além disso, devido à proximidade com o magneto, a existência de um campo magnético na região da TT permite uma medida de momento menos precisa porém útil para a seleção dessas partículas.

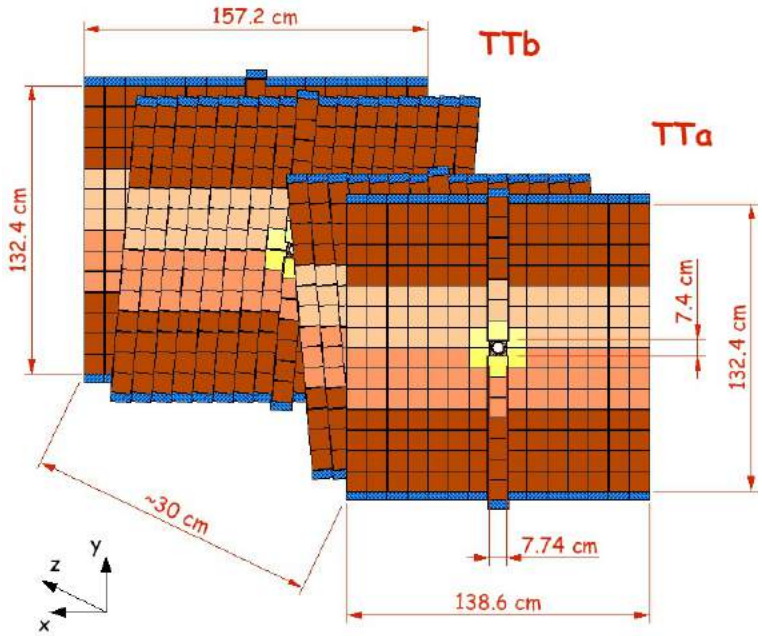

Figura 3.4: Layout dos 4 detectores da estação de trajetória TT.

O TT possui resolução espacial de aproximadamente $50\mu m$ e é composto por tiras de silício com 11 cm de comprimento, 7.8 cm de largura e 198 μm de espessura para fazer a leitura do sinal eletrônico quando uma partícula atravessa o detector. Ao todo são 4 planos, sendo que os dois mais externos têm tiras posicionadas verticalmente e os dois internos possuem tiras posicionadas em ângulos de $\pm 5°$, dependendo da posição do plano com relação ao tubo por onde passa o feixe. A estrutura desses planos está ilustrada na figura 3.4.

Inner e Outer Tracker O IT compreende a parte interna das estações T1, T2 e T3, que estão localizadas entre o magneto e os calorímetros. O OT compreende a parte externa das estações T1, T2 e T3. Os principais objetivos dos IT e OT são reconstruir a trajetória das partículas entre o VeLo e os calorímetros e medir o momento com o auxílio do magneto, medir com precisão o local onde as partículas atingem o RICH e conectar as medidas feitas pelo VeLo com as obtidas nos calorímetros. O IT e OT estão representados na figura 3.5.

O IT têm o mesmo formato , resolução e composição das TT e juntas são chamadas de *Silicon Tracker* no LHCb, devido à utilização de silício em seus detectores. O OT possuem resolução espacial de aproximadamente $200\mu m$ no eixo horizontal e é composto

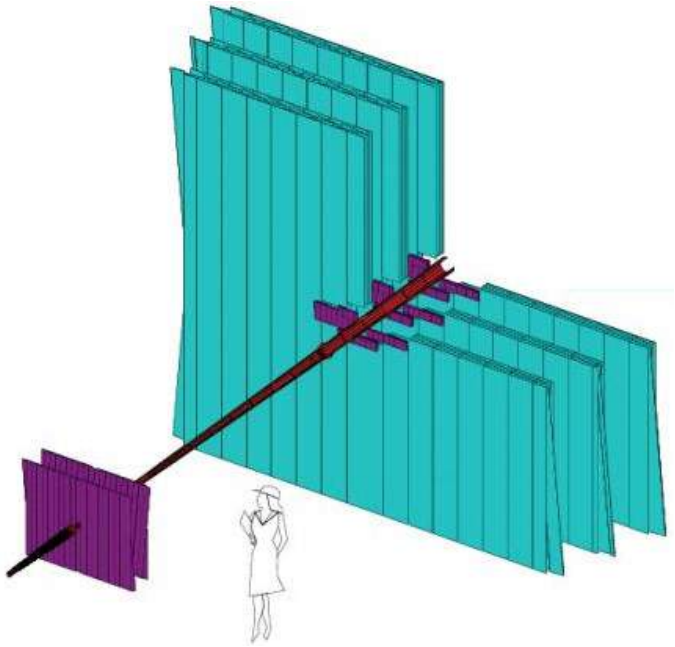

Figura 3.5: Posicionamento das estações de trajetória. Em roxo na parte inferior esquerda se encontra o TT. Na parte à direita, se encontra as 3 estações T1, T2 e T3. Em roxo, ao centro das estações T, se encontra o Inner Tracker. O restante das estações T, em azul claro, são as camadas do Outer Tracker.

por câmaras de arrasto em formato cilíndrico com 5 mm de diâmetro e 0.075 mm de espessura. A cobertura total do OT chega a uma região de 600 $cm \times 490$ cm enquanto que as IT cobrem a parte interna (região ao redor do feixe), uma região de aproximadamente 120 $cm \times 40$ cm.

3.4 RICH

O *Ring Imaging Cherenkov* (RICH) [8] é o detector responsável pela identificação de partículas carregadas com momento na faixa de 1GeV até 150GeV, tendo o auxílio do sistema de trajetografia. O seu funcionamento está fundamentado na radiação *Cherenkov*, que é a radiação emitida por partículas que possuem uma velocidade maior que a velocidade da luz no meio em que se propagam. Essa radiação emitida é então capturada após ser seguidamente refletida por espelhos dentro do RICH e tem seus ângulos de emissão e reflexão medidos. Com essas informações é possível medir a velocidade com que a partícula entrou no detector. Sabendo-se o momento então é possível determinar a massa e, a partir

desta, a identidade da partícula.

O LHCb possui 2 detectores do tipo RICH: o RICH1 está localizado logo após o VeLo e opera na faixa de 1GeV até 65GeV enquanto o RICH2 fica logo após as estações de trajetória e opera na faixa de 15GeV até 150GeV.

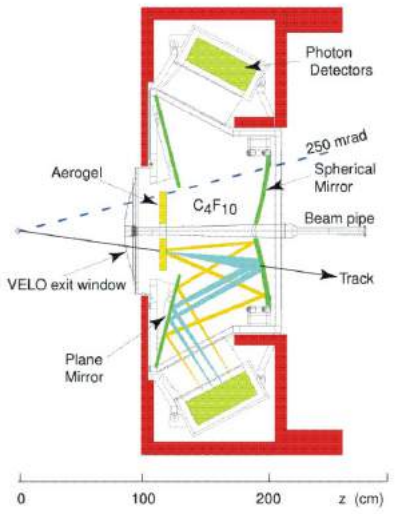

Figura 3.6: Vista lateral do detector RICH1.

3.5 Calorímetro

Os calorímetros são detectores que tem por objetivo medir a energia depositada pelas partículas que são desaceleradas ao atingirem o calorímetro. O LHCb possui dois tipos de calorímetros [9]: o Hadrônico (HCAL) e o Eletromagnético (ECAL). O HCAL tem por objetivo a identificação de partículas cuja interação principal é dada pela força forta, enquanto o ECAL visa a identificação de partículas que interagem por força eletromagnética.

O sistema de calorímetros ainda compreende o *Scintillating Pad Detector* (SPD) e *Pre-Shower Detector* (PS). O SPD é responsável pela identificação de partículas neutras ou carregadas enquanto que o PS difere entre estas partículas elétrons (carregada) e fótons (neutra). Na Figura 3.7 podemos ver a distribuição das células nos calorímetros do LHCb.

A estrutura do calorímetro varia de acordo com seu componente. O SPD, PS e ECAL são compostos por células organizadas em uma mesma estrutura, ilustrada na

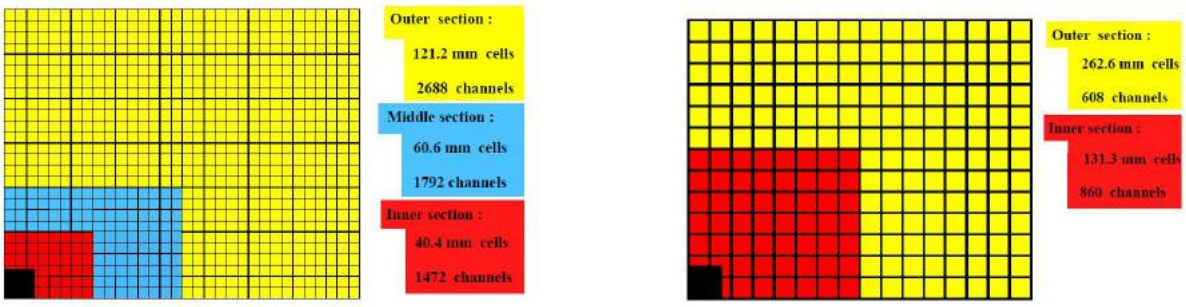

Figura 3.7: Ilustração da granularidade nos SPD, PS e ECAL (esquerda) e no HCAL (direita).

figura 3.7. O tamanho de cada célula é de 4×4 cm na parte interna do detector, 6×6 cm na parte intermediária e 12×12 cm na parte externa. As células do SPD e do PS são placas cintiladoras de 15 mm de espessura conectadas à fotomultiplicadora por fibra ótica. Já o ECAL alterna placas cintiladoras de 4 mm de espessura e pratos de chumbo de 2 mm de espessura. O HCAL alterna placas cintiladoras de 4 mm de espessura com pratos de ferro de 16 mm de espessura.

3.6 Sistema de Múons

O sistema de múons [10] é um conjunto de detectores formando 5 estações de múons (M1 a M5), que tem por objetivo fazer a medição da trajetória dos múons que atravessam o experimento. Este fica localizado ao final do experimento devido ao fato dos múons serem as únicas partículas que não são contidas pelo calorímetro. O sistema de múons cobre toda a aceptância do experimento e uma área de aproximadamente 435 m^2. A estação M1 está localizada à frente do PS enquanto que as outras 4 estações estão localizadas logo após o HCAL, separadas por filtros de ferro com as mesmas dimensões. Ao todo existem 1380 câmaras, de 20 tamanhos diferentes, instaladas de acordo com a ocupação de partículas nas diferentes regiões da estação.

Cada câmara de múons é composta por células preenchidas com uma combinação de 3 gases: dióxido de carbono, argônio e tetrafluorometano. Ao atravessar uma célula, o múon ioniza essa combinação de gases e um eletrodo localizado no centro da célula captura os elétrons resultantes da ionização, gerando uma corrente que é medida pela eletrônica de *front-end* do detector.

A Figura 3.8 mostra a posição das 5 estações de múons no experimento.

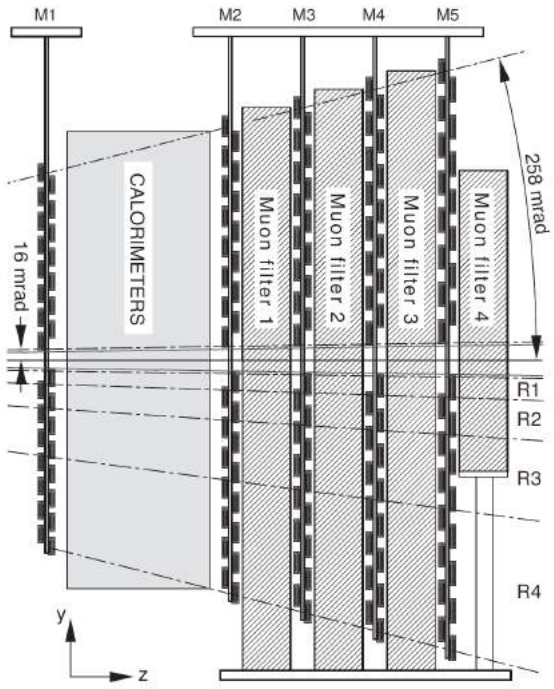

Figura 3.8: Vista lateral do sistema de múons, indicando a posição das estações M1 a M5.

3.7 Trigger

O sistema de *Trigger* do LHCb é o sistema responsavel por filtrar quais dentre os eventos terão ou não seus dados armazenados no *cluster* de armazenamento do LHCb.

Os detectores do LHCb capturam os eventos na mesma taxa em que ocorrem as colisões: 40MHz. O volume de dados gerados a cada evento é muito grande e a eletrônica utilizada para envio e aquisição de dados no experimento não possuem largura de banda suficiente para recuperação dos dados de todas as colisões. Por esta razão, os eventos são filtrados a fim de selecionar apenas os mais relevantes para o propósito do experimento. O objetivo final do Sistema de Trigger é reduzir a taxa de armazenamento de eventos de 40MHz para a ordem de kHz. O sistema é dividido em duas partes: o *Level-0 Trigger* (L0) e o *High-Level Trigger* (HLT).

L0 O Trigger de nível 0 é um trigger implementado a nível de hardware. Este recebe dados diretamente dos detectores para efetuar a decisão de manter ou não o evento. O objetivo do L0 é reduzir a taxa de eventos a 1MHz. O L0 é implementado em módulos eletrônicos na região do experimento.

O Trigger L0 recebe informações do VeLo, dos calorímetros, do sistema de múons e do sistema de *Pile-up*, que é um sistema responsável por identificar empilhamento de sinais nos mesmos canais dos detectores. Os critérios para decisão do sistema de trigger são baseados nos seguintes parâmetros:

- Energia total depositada nos calorímetros hadrônicos e eletromagnéticos.
- Momento transverso das trajetórias reconstruídas nas estações de múons.
- Identificação da partícula pela leitura dos calorímetros.
- Multiplicidade do número de trajetórias que atingem os calorímetros.
- Número de vértices primários e de trajetórias provenientes de um segundo vértice.

O valores de corte para cada um destes parâmetros são ajustados de forma a garantir a redução de eventos para o valor máximo desejado de 1MHz.

HLT O Trigger de Nível Superior é um trigger implementado a nível de software. O mesmo se trata de algoritmos que são executados nos computadores da sala de contagem, que recebem os dados do experimento e efetuam a reconstrução a fim de identificar candidatos a méson B de interesse. O filtra aplicado pelo HLT reduz o número de eventos enviados para armazenagem a uma taxa de poucos kHz.

3.8 Sistema Online

O Sistema Online do LHCb compreende todos os aspectos de computação *online* e os sistemas de controle do experimento LHCb. Ele provê a infraestrutura para a tomada de dados para os Triggers de alto nivel (HLT), assim como para controle, configuração e monitoramento de todo o experimento.

O Sistema Online compreende três componentes: DAQ (*Data Acquisition System*), o TFC (*Trigger and Fast Control*) e ECS (*Experiment Control System*).

DAQ O Sistema de Aquisição de Dados tem por objetivo transportar os dados proveniente dos detectores até o local de armazenagem permanente. Como visto na seção 3.7, apenas uma pequena parte dos dados é armazenada. A aquisição é engatilhada pelo Trigger de Nível 0 (L0) e o DAQ é responsável por garantir o envio de todos os dados referentes ao evento indicado pelo L0.

A leitura dos detectores é efetuada pela eletrônica de *front-end* e os dados são enviados a módulos eletrônicos de leitura chamados TELL1. Os dados recebidos pelo TELL1 são processados em quatro FPGAs. Nos FPGAs os dados são processados de acordo com a necessidade do detector de origem, podendo ser aplicada compressão de dados, supressão de zeros entre outros algoritmos. Após o processamento, os dados são coletados por um quinto FPGA (SyncLink) e formatados em um pacote raw IP, que é enviado ao sistema de aquisição de dados através de um módulo Gigabit-Ethernet de quatro canais. A interface com o ECS é feita através de um CCPC (*Credit Card PC*) montado como mezzanino na TELL1. Os sinais de sincronismo e clock são transmitidos através da interface de Trigger, Timing and Control (TTC). O fluxo de dados para o TFC é realizado através do sinal de trottle e é conduzido pelo FPGA SyncLink. O diagrama de blocos simplificado é apresentado na Figura 3.9.

TFC O Sistema de *Trigger and Fast Control* (TFC) é o sistema responsável por distribuir o clock de sincronismo do feixe do LHC, as decisões do *trigger* L0, sinais síncronos de reset e comandos de *fast control*. O TFC trata todos os estágios da leitura dos dados entre a eletrônica de *front-end* e os computadores de aquisição na sala de contagem. Ele não apenas distribui os sinais mas também garante que sinais como de trigger, de reset e de clock cheguem aos módulos de destino de forma síncrona.

ECS O Sistema de Controle de Experimento (ECS) é o sistema responsável pela configuração, controle e monitoramento operacional de todos os componentes do sistema online e dos detectores. Ele provê formas de enviar comandos aos diferentes equipamentos do sistema e de efetuar operações de escrita e leitura em registradores.

O ECS abrange alguns protocolos, dentre eles o *Serial Protocol for ECS* (SPECS), que é um protocolo próprio desenvolvido no CERN, *Controller Area Network* (CAN) e Ethernet, mas não se limita a estes padrões e para alguns dispositivos foi criado uma solução customizada para interface com o sistema ECS. O ECS é um sistema complexo,

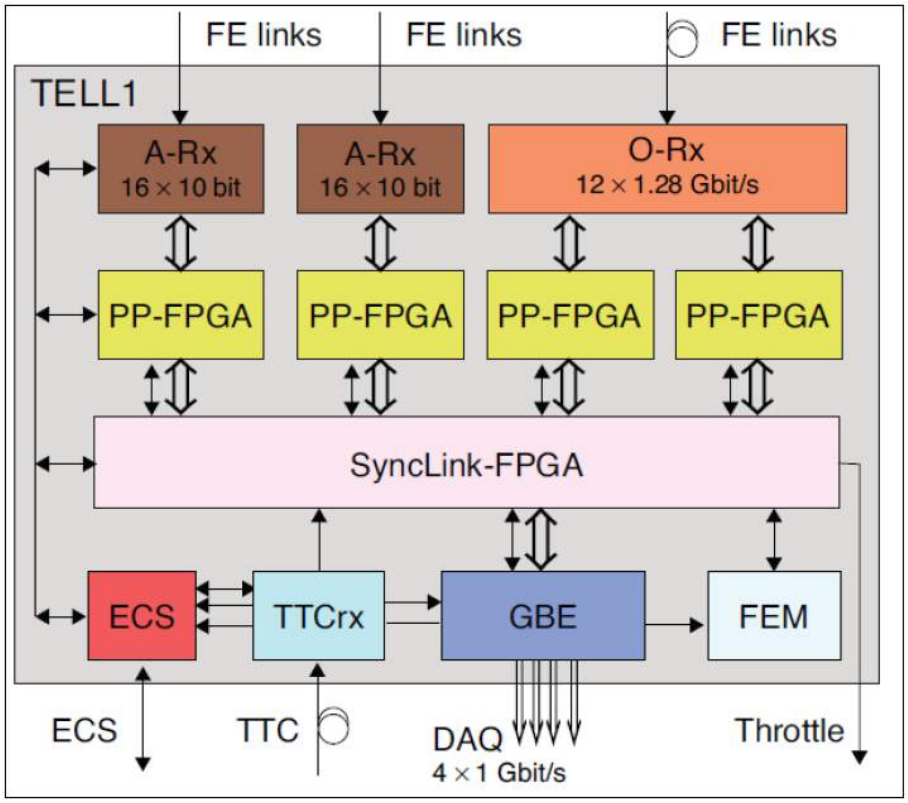

Figura 3.9: Diagrama de blocos simplificado do módulo de aquisição TELL1.

que compreende os módulos que fazem a comunicação com os equipamentos finais, os drivers para interface com estes módulos e um sistema de controle baseado no software WinCC da Siemens, que provê meios de gerenciar todos os módulos do sistema e fornece a interface com o usuário. Mais sobre o funcionamento de um sistema de controle WinCC é explicado na seção 9.

Dentre os itens controlados pelo ECS, podemos citar:

- Operações de detector
  - Sistema de distribuição de gás
  - Módulos de alta tensão
  - Módulos de alimentação
  - Controle de temperatura
- Sistemas de *Trigger* e DAQ

  - Eletrônicas de *front-end*
  - Reconstrução de eventos
  - Parâmetros de filtragem do HLT
- Infraestrutura do experimento
  - Sistema de ventilação e refrigeração
  - Distribuição elétrica
- Controle do Magneto, Sistemas do Acelerador, Sistemas de Segurança e etc.

Na Figura 3.10 temos uma ilustração simplificada do sistema de controle e sua cadeia de contole.

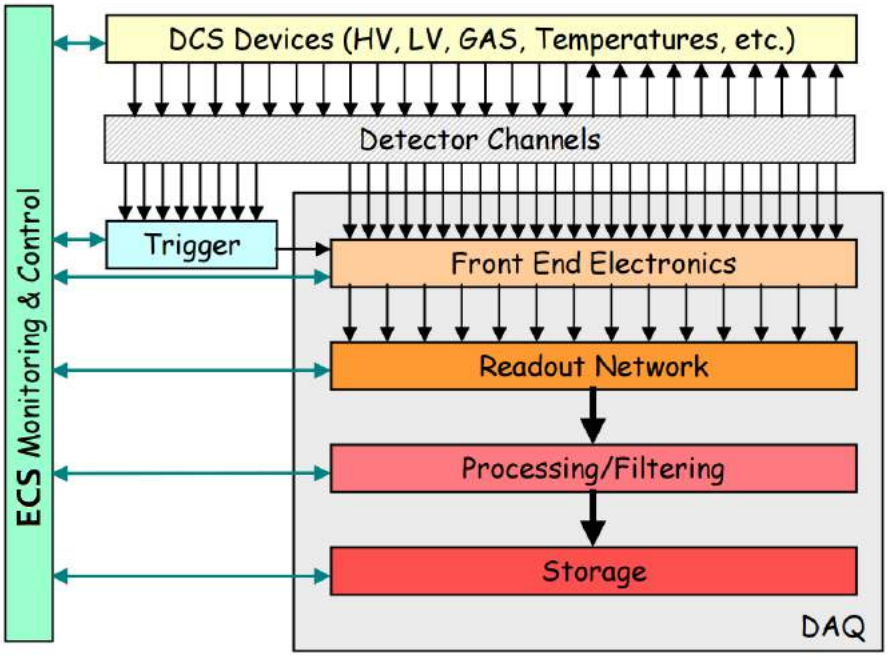

Figura 3.10: Escopo de controle do sistema ECS, mostrando as diferentes operações controladas em paralelo pelo ECS.

Capítulo 4

Upgrade do LHCb

Definição O LHC passará por uma atualização durante o período LS2 e LS3 que visam o aumento de sua luminosidade de 5 a 7 vezes a luminosidade nominal para a qual foi projetado. O projeto do novo LHC foi chamado de "High Luminosity LHC" (HL-LHC) [11]. Na Figura 4.1 se pode ver o cronograma de operação do LHC / HL-LHC para os próximos anos. De forma complementar, o experimento LHCb também sofrerá atualizações a fim de se adequar aos novos regimes de maior luminosidade que serão oferecidos pelo acelerador LHC. O LHCb terá seu upgrade realizado durante a próxima interrupção longa (LS2) do LHC, que está previsto para os anos de 2019 e 2020, tendo seu retorno em 2021.

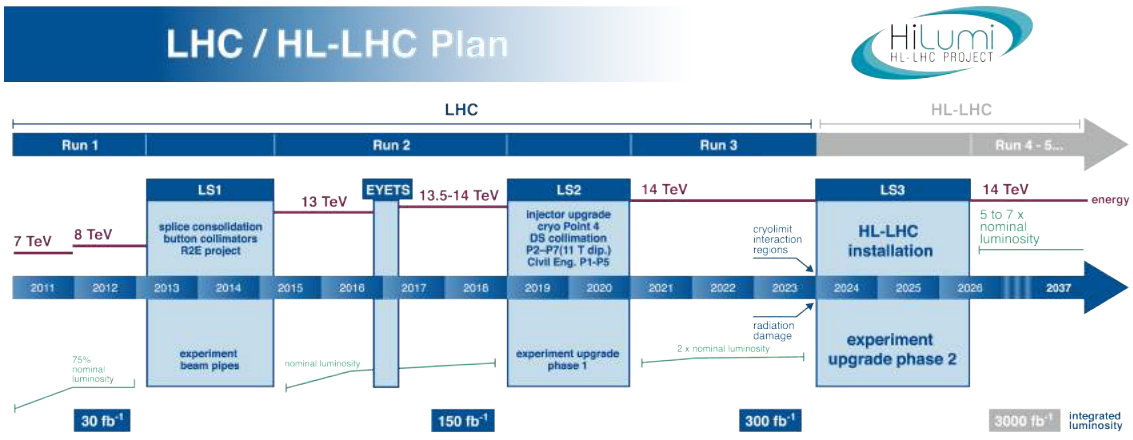

Figura 4.1: Cronograma de operação do LHC para os anos seguintes.

Resultado esperado. Com o upgrade do LHCb, espera-se elevar o seu rendimento físico em decaimentos com muon por um fator de 10, o rendimento dos canais hadrônicos por um fator de 20 e coletar dados a uma luminosidade aproximada de $1 - 2 \times 10^{33} cm^{-2} s^{-1}$. Isto corresponde a dez vezes a luminosidade do projeto atual e permitirá ao LHCb elevar o seu volume de dados coletados anualmente do atual $1 fb^{-1}$ de luminosidade integrada para 5 a $10 fb^{-1}$. Uma Carta de Interesse [12], um TDR de Framework [13] e um TDR de Trigger e Online [14] documentam os planos para uma atualização dos componentes do experimento.

Atualizações O upgrade do LHCb será baseado na extinção do trigger de nível 0 e aquisição de dados a uma taxa de 40MHz. O aumento na luminosidade do LHCb trará como consequência a elevação significativa do nível de radiação incidente, do volume de dados gerados e da ocupação nos canais dos detectores, entre outras. Os novos níveis de radiação exigem a validação de todos os componentes do sistema e o uso de eletrônica apropriada. O volume de dados exige substituição de toda eletrônica de aquisição do experimento e também da eletrônica de *front-end* (FEE) dos detectores. Por ter a FEE embutida no módulo de detecção, os detectores VELO, IT, TT e RICH também terão de ser substituídos. O novo nível de ocupação no detector *Outer Tracker* exige uma reformulação completa de sua arquitetura.

4.1 GBT

O GBT (*GigaBit Transceiver*) se refere a um projeto com o objetivo de desenvolver um conjunto de circuitos integrados (*chipset*) resistentes a radiação para permitir a implementação de um link óptico bidirecional de 4.8Gbps a ser utilizado no upgrade do LHC. O link implementado pelo GBT visa a transmissão de dados entre a eletrônica de *front-end* e *back-end*, atendendo simultaneamente aplicações como aquisição de dados, *trigger*, temporização e controle. O GBT possui um protocolo próprio de comunicação. No projeto GBT há um ramo chamado projeto GBT-FPGA que visa o desenvolvimento de um código HDL que permita a interface direta entre FPGAs e links baseados no GBT.

O *chipset* GBT compreende os seguintes circuitos integrados:

- GBTx é o chip que serializa e desserializa os dados do link, e implementa a camada

de protocolo do link, sendo responsável pela codificação e decodificação dos dados.

- GBT-SCA (*GBT-Slow Control Adapter*) é um chip responsável por fornecer interfaces de controle ao sistema. O GBT-SCA traduz comandos recebidos pelo link GBT em comandos em protocolos comuns utilizados para controle como I2C, SPI e JTAG. Adicionalmente o mesmo possui conversores analógico-digitais e digital-analógicos que podem ser utilizados para diferentes propósitos.

- GBTIA é um amplificador de transimpedância cujo objetivo é amplificar a corrente proveniente do diodo PIN. Ele opera a uma taxa de até 5Gbps e foi desenvolvido para lidar com a degradação de sensibilidade do diodo PIN devido a incidência de radiação.

- GBLD é o circuito integrado responsável por alimentar o transmissor com os dados, modulando a corrente fornecida ao laser para alcançar a melhor conversão eletro-óptica.

No upgrade do experimento LHCb, os chips GBT são utilizados na eletrônica de *front-end*, a fim de suprir a necessidade de um link óptico de alta velocidade e resistente a radiação. Já na eletrônica de aquisição, que fica em área sem incidência da radiação do LHC, não é necessário a utilização dos chips. Neste caso utiliza-se um módulo óptico comercial e a camada de protocolo do GBT é implementada em FPGA. O objetivo disto é reduzir o custo com a produção de circuitos integrados de aplicação específica. Na Figura 4.2 podemos ver o diagrama da implementação de um link GBT.

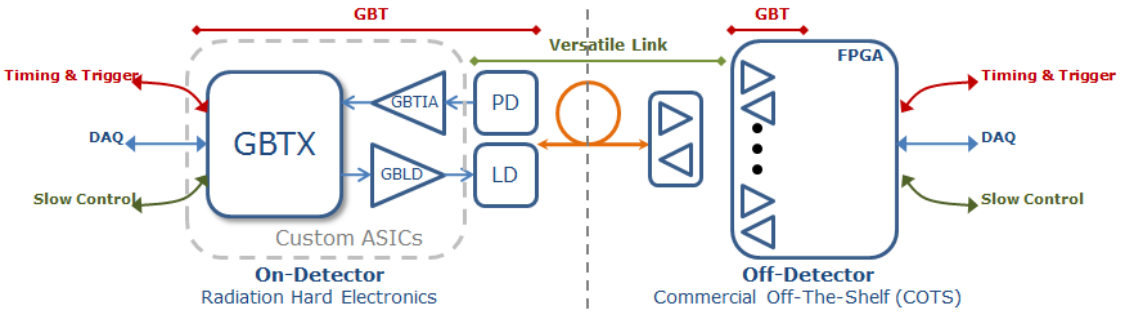

Figura 4.2: Diagrama apresentando aplicação dos chips do projeto GBT.

O GBT foi desenvolvido de forma a minimizar a perda de dados por "*Single Event Upsets*"(SEU), que é alteração do estado lógico de uma célula de memória (um bit de um

registrador, por exemplo) pela passagem de uma partícula ionizante. O GBT é produzido utilizando tecnologia CMOS de 130nm, que é menos susceptível a SEUs, e implementa uma arquitetura de tripla redundância. Esta consiste em criar 2 registradores adicionais para cada registrador original, que recebem sempre o mesmo valor, mas sempre que a alteração de algum deles, estes serão reescritos com o valor predominante. Desta forma para haver perda de dado por SEU, seria necessário que ocorresse um SEU no mesmo bit de 2 registradores diferentes.

O chip GBTx recupera sinal de clock do próprio link GBT. Isto significa que não é necessário um sinal de clock compartilhado. Os mesmos precisam apenas estar operando na mesma frequência. Uma PLL interna do GBTx receptor se calibra em função das bordas do sinal de dados recebido e a partir disto gera um clock para o desserializador.

O protocolo GBT é composto por palavras de 120 bits e fornece 3 configurações diferentes: o modo GBT, o modo *WideBus*, e o modo 8B/10B. No modo 8B/10B, os 120 bits da palavra GBT são simplesmente divididos em 12 palavras na decodificação 8B/10B. No modo GBT, 4 bits são dedicados a um cabeçalho que indica se a palavra é referente a dados válidos ou não, 4 bits são dedicados a controle, 80 bits são dedicados a dados e 32 bits são reservados para um algoritmo de correção de erro. A Figura 4.3 mostra o formato da palavra no modo GBT. O Modo *WideBus* tem os 4 bits de cabeçalho e controle como o modo GBT, no entanto, ele utiliza todos os 112 bits restantes para dados, abrindo mão do sistema de correção de erro.

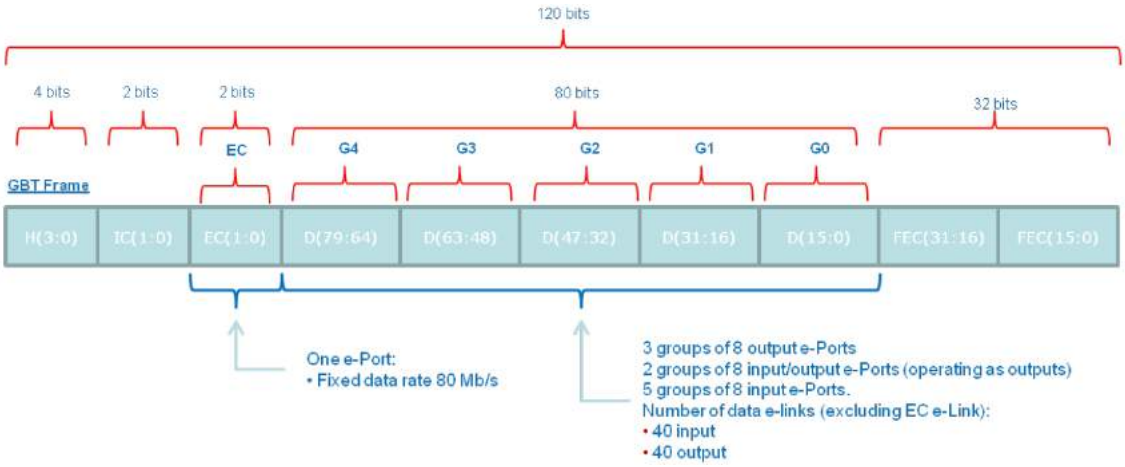

Figura 4.3: Estrutura da palavra de 120 bits do protocolo GBT.

O codificador do GBT é composto por 3 etapas: *scrambler*, *RS Encoder* e *Interleaver*. Podemos ver o diagrama de blocos do codificador e decodificador na Figura 4.4. O

RS Encoder (Codificador Reed-Solomon) só é utilizado no modo GBT e ele é o responsável pelo sistema de correção de erro. O *scrambler* é um algoritmo que troca os valores dos bits baseado nas palavras anteriores e numa palavra semente, cujo objetivo é impedir que, numa sequência de dados composta predominantemente por '0' ou '1', o valor DC médio do meio de transmissão não fique próximo a Gnd ou Vcc. Isto é um procedimento usado em padrões de comunicação para impedir a carga ou descarga completa de um capacitor de acoplamento que esteja na linha de dados, podendo causar um sinal alto ou baixo contínuo no receptor. O *Interleaver* é um algoritmo que troca a posição de bits específicos, com o objetivo de facilitar a reconstrução da palavra quando ocorre a perda de bits em sequência.

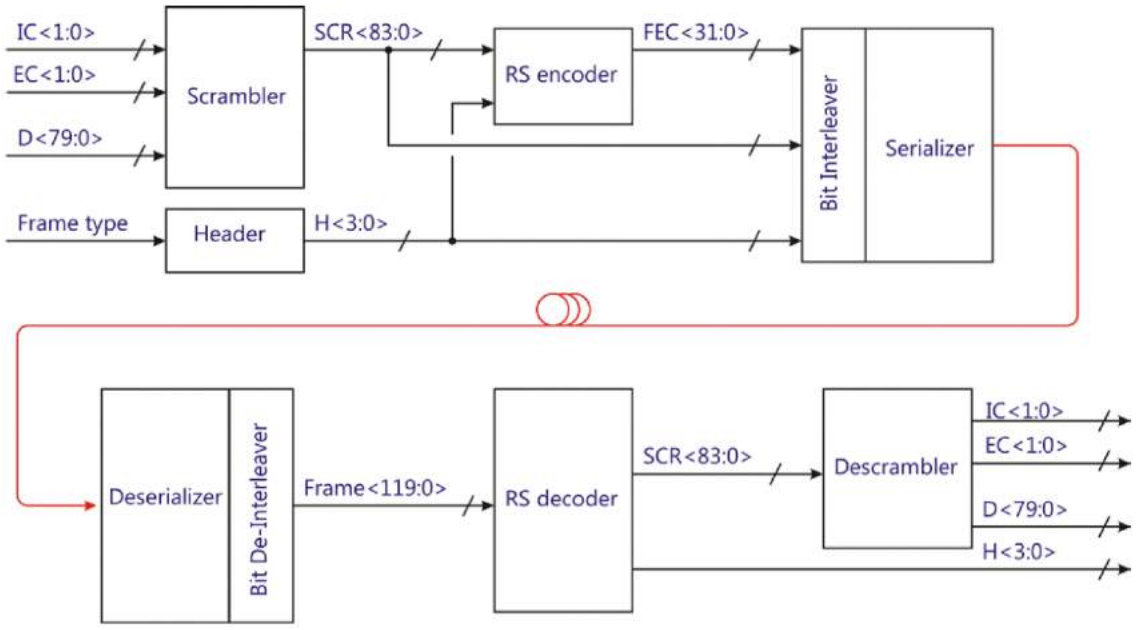

Figura 4.4: Diagrama de blocos do codificador e decodificar do GBT-FPGA.

4.2 Trigger e DAQ

Uma das maiores limitações do experimento atual é a necessidade de reduzir o número de eventos a serem lidos para uma taxa de 1.1MHz. Esta redução é alcançada através dos parâmetros de filtragem do *trigger* L0. Este é atualmente o maior gargalo na geração de resultados do experimento, e o principal fator que impede que o LHCb faça proveito do regime de maior luminosidade que o upgrade do LHC irá oferecer. Por esta razão, foi decidido que o *trigger* L0 seria eliminado completamente, e a leitura do

experimento seria realizada a uma taxa de 40MHz, a mesma das colisões, sem nenhuma filtragem a nível de hardware. A leitura dos eventos será transmitida diretamente para a eletrônica de *back-end* e a filtragem dos eventos será dada inteiramente pelo HTL, a nível de software. Um trigger implementado completamente por software adiciona um enorme grau de flexibilidade na escolha de eventos, permitindo que o experimento altere completamente os parâmetros base para a filtragem de eventos, antes limitado pelas conexões físicas do *trigger* L0.

A extinção do *trigger* L0 com leitura a 40MHz acarreta a substituição de toda eletrônica de aquisição de dados do experimento. Inicialmente, o formato pensado para o sistema de aquisição foi similar ao utilizado atualmente pelo experimento, mas tendo por principal diferença a largura de banda que suportaria um volume de dados bem maior. Este seria composto por um módulo mãe do padrão ATCA (*Advanced Telecommunications Computing Architecture*) que seria equipada com quatro módulos mezzaninos do padrão AMC (*Advanced Mezzanine Card*) e que recebeu o nome de AMC40. A AMC40 seria a interface entre a eletrônica de *front-end* e a rede de computação. Esta ficaria na sala de contagem e transmitiria os dados para a rede através de links ethernet. Cada AMC40 é capaz de processar dados a uma taxa de 108Gbits/s. Ela poderia ser utilizada tanto como módulo de aquisição de dados como também módulo de controle (TFC e ECS), dependendo da forma como é configurado seu FPGA.

A AMC40 teve algumas unidades fabricadas que foram entregues montadas em uma plataforma de teste chamada MiniDAQ, que era composta por um módulo contendo um CCPC (*Credit Card PC*) e um conector AMC que permitia a utilização da AMC40. O MiniDAQ e a AMC40 foi o sistema escolhido para ser utilizado no Sistema de Teste da Eletrônica do SciFi, sendo a ferramenta principal na realização do trabalho desta dissertação. O MiniDAQ e a AMC40 são explicados com mais detalhes no Capítulo 8

Em um momento posterior, um novo formato para o sistema de aquisição foi pensado. Um módulo de aquisição chamado PCIe40. Esta será a solução utilizada no experimento após o upgrade. Ela é semelhante à AMC40, mas possui um conector PCI express ao invés do conector AMC e um FPGA de maior capacidade. Esta foi desenvolvida com o intuito de embarcar a interface com a eletrônica de *front-end* nos próprios servidores que recebem os dados, por esta razão o conector PCI express.

A PCIe40 é composta por um FPGA Arria 10, 24 links ópticos de entrada para aquisição de dados e 12 links bidirecionais para controle e a interface PCIe a qual permite

conectá-la ao servidor para que seja feito a coleta dos dados. Há também uma versão de 48 links bidirecionais, usado somente para controle. Cada PCIe40 pode processar uma taxa máxima de dados de 100Gbits/s. Devido ao fato de que a PCIe40 será instalada em região sem incidência de radiação, não há necessidade de se utilizar os chips e componentes ópticos do projeto GBT. A PCIe40 utiliza módulos transceptores ópticos comerciais chamados MiniPODs e para decodificação, utiliza instâncias do firmware GBT-FPGA implementadas no FPGA Arria 10. A imagem de um protótipo da PCIe40 pode ser visto na Figura 4.5.

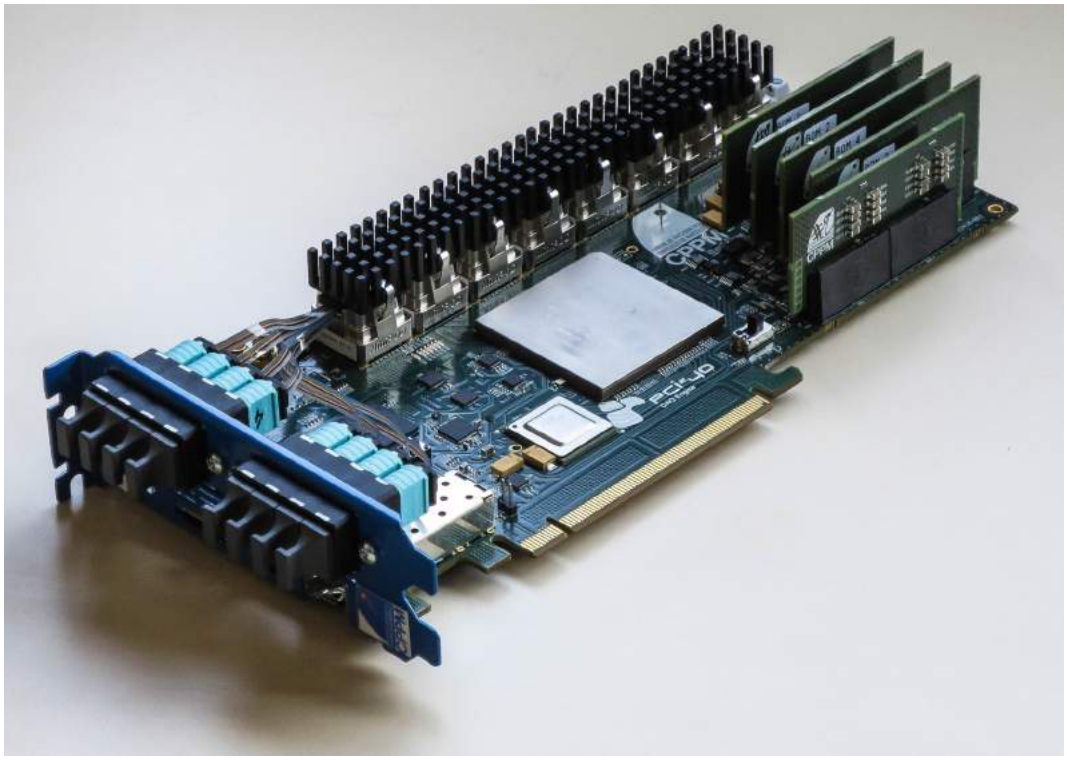

Figura 4.5: Protótipo da placa de aquisição PCIe40 a ser utilizada no experimento LHCb.

4.3 Upgrade dos Detectores Inner e Outer Tracker

As estações de trajetória T1, T2 e T3, formadas pelos detectores *Inner* e *Outer Tracker*, foram projetadas para prover sua melhor performance em colisões próton-próton com energia de centro de massa de 14TeV ocorrendo a cada 25ns (40MHz), e com luminosidade instantânea de no intervalo de aproximadamente $2 - 5 \times 10^{32} cm^{-2} s^{-1}$. O LHCb após o upgrade irá operar com luminosidade superior, na faixa de $1 - 2 \times 10^{33} cm^{-2} s^{-1}$. A

geometria do sistema de trajetografia atual foi escolhida de forma que a máxima ocupação nas regiões mais quentes do *Outer Tracker* fosse limitada a 10% quando sob luminosidade de $2 \times 10^{32} cm^{-2} s^{-1}$. Esta limitação é estabelecidade pela eficiência do algoritmo de reconstrução de trajetórias mediante elevada luminosidade. Melhorias foram feitas neste algoritmo que permitiram o *Outer Tracker* a operar com até 25% de ocupação sem perdas na eficiência da reconstrução de trajetórias, mas ainda assim, esta ocupação equivale a uma luminosidade de $5 \times 10^{32} cm^{-2} s^{-1}$, que ainda é de 2 a 4 vezes menor que a luminosidade esperada no upgrade. Por esta razão surgiu a necessidade de troca dos módulos do *Outer Tracker* nas regiões de maior luminosidade. Toda a eletrônica de *front-end* também precisaria ser trocada para se adequar aos novos requisitos de leitura a 40MHz.

Discutiu-se diversas possibilidades de acordo com os requerimentos do detector para o upgrade até que foi decidido a troca completa dos *Inner e Outer Tracker* por um detector formado por fibras cintilantes e com leitura realizada a partir de fotomultiplicadoras de silício (SiPM). Os principais requisitos deste novo detector estão listados a seguir:

- A eficiência de detecção deve ser superior a 99%.
- A resolução espacial no plano de atuação do magneto deve ser menor ou igual a 100μm.
- A eletrônica de leitura do detector deve operar a 40MHz e fornecer uma taxa de amostragem equivalente.
- O detector deve ser capaz de operar com a performance requisitada por uma luminosidade integrada de ao menos 50fb^{-1}.

O detector também deve ser limitado em dimensões a fim de caber no espaço ocupado pelo detector atual. O detector deve tolerar os níveis de radiação durante o tempo previsto de operação. O nível de radiação esperado no detector após luminosidade integrada de 50fb^{-1} na região de pico, conforme estudo e realizado em [15], é de aproximadamente 35kGy e 25kGy nas estações T1 e T3 respectivamente. Na Figura 4.6 podemos ver o gráfico desta radiação esperada no plano x-y, distando 783cm do ponto de colisão. Esta posição é atualmente a face do Outer Tracker da estação de trajetória T1.

O detector de fibras cintilantes já se encontra em fase de desenvolvimento e será abordado com detalhes no Capítulo 5.

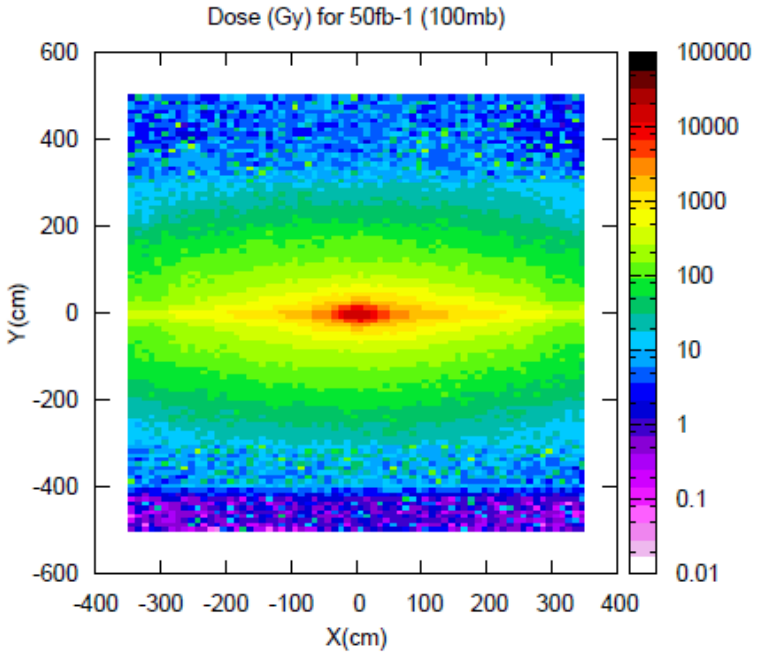

Figura 4.6: Dose esperada no plano x-y a uma distância z=783cm após uma luminosidade integrada de 50fb^{-1}.

Capítulo 5

O Detector de Fibras Cintilantes

O Detector de Fibras Cintilantes (SciFi) é um detector de trajetória de partículas carregadas, com eficiência superior a 99% e resolução espacial ao longo do eixo de influência do magneto inferior a $60\mu m$. O SciFi é o detector responsável por determinar as coordenadas em que as partículas carregadas cruzaram as estações de trajetória T1, T2 e T3. O objetivo do SciFi é fornecer as informações necessárias para reconstrução das trajetórias, determinação da massa e momento de partículas carregadas que atravessam o detector, e servir informações de posição das partículas necessárias ao sistema de identificação de partículas. Os parâmetros de massa e momento são indiretamente medidos graças à força aplicada pelo campo magnético do magneto, que modifica a trajetória das partículas.

O detector SciFi consistirá em três estações de trajetória, entre o magneto e o detector RICH2. Cada estação do SciFi será composta por 4 camadas de detecção. As 3 estações são centradas no mesmo ponto no eixo z, que é o centro do tubo do LHC. As camadas de detecção de cada estação se encontram, em sequência, nos ângulos 0°, +5°, -5° e 0° em relação ao eixo vertical, correspondendo aos planos X, U, V e X respectivamente. Há um espaço de 20cm entre cada camada de cada estação, e há um orifício circular no centro do detector pelo qual passa o tubo do feixe do LHC.

A pequena diferença de ângulo entre os planos de detecção é feita para maximizar a resolução na direção em que o magneto desvia as partículas, uma vez que a medida de massa e momento depende apenas do valor nesta direção. Por consequência, há um detrimento da resolução espacial no eixo y.

A Figura 5.1 apresenta um modelo 3D do detector, entre o magneto e o RICH2.

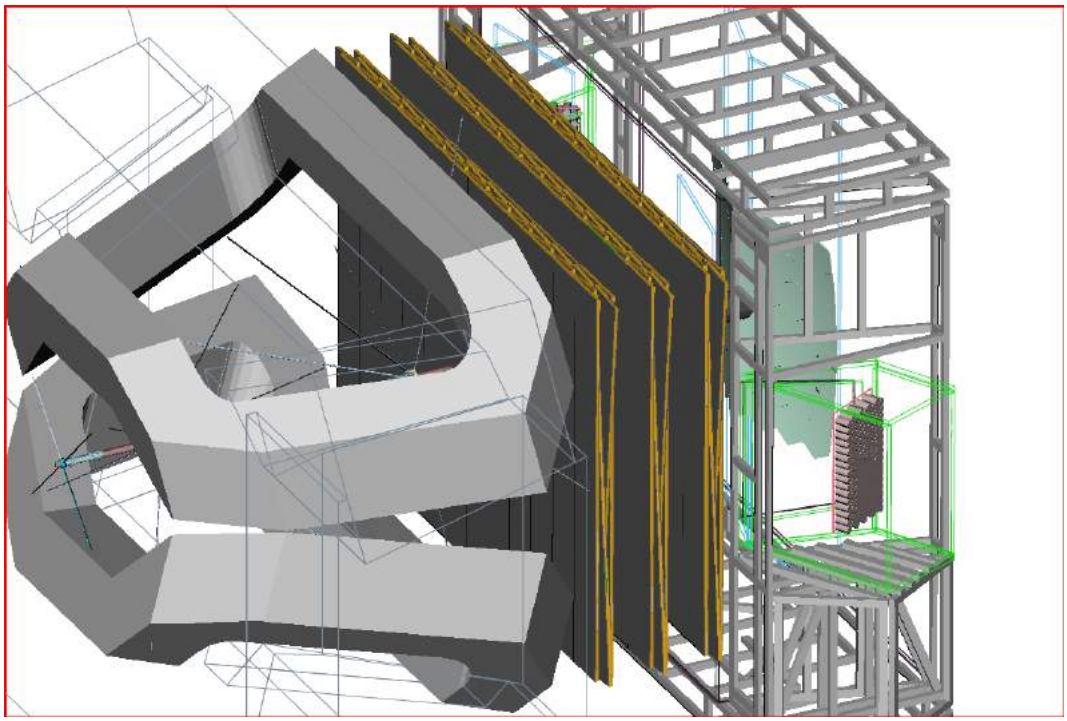

Figura 5.1: Modelo 3D das 3 estações que compõem o detector de fibras cintilantes, mostrado entre o magneto (à esquerda) e o detector RICH2 (estrutura à direita).

Cada camada de detecção é subdividida em 12 módulos de fibras cintilantes. Cada módulo possui 5m de altura por 0,52m de largura. O detector completo cobre a área de 5m de altura por 6m de largura, no plano X-Y. Ao total, o detector SciFi compreende 144 módulos.

A Figura 5.2 mostra como é o arranjo dos módulos e um comparativo de tamanho com uma pessoa de estatura média.

5.1 Composição de um Módulo do Detector

Cada módulo do detector é subdividido em oito *mats* de fibras cintilantes com 2,5m de comprimento. Quatro *mats* são postos lado a lado na parte superior e quatro na parte inferior. Os módulos que ficam no centro serão ligeiramente modificados para acomodar o tubo do feixe do LHC que cruza o centro do detector. Os *mats* são estruturas de 13cm de largura onde são coladas as fibras. As fibras são coladas de forma empilhada, encaixadas entre si, formando 6 camadas de fibras conforme visto na Figura 5.3.

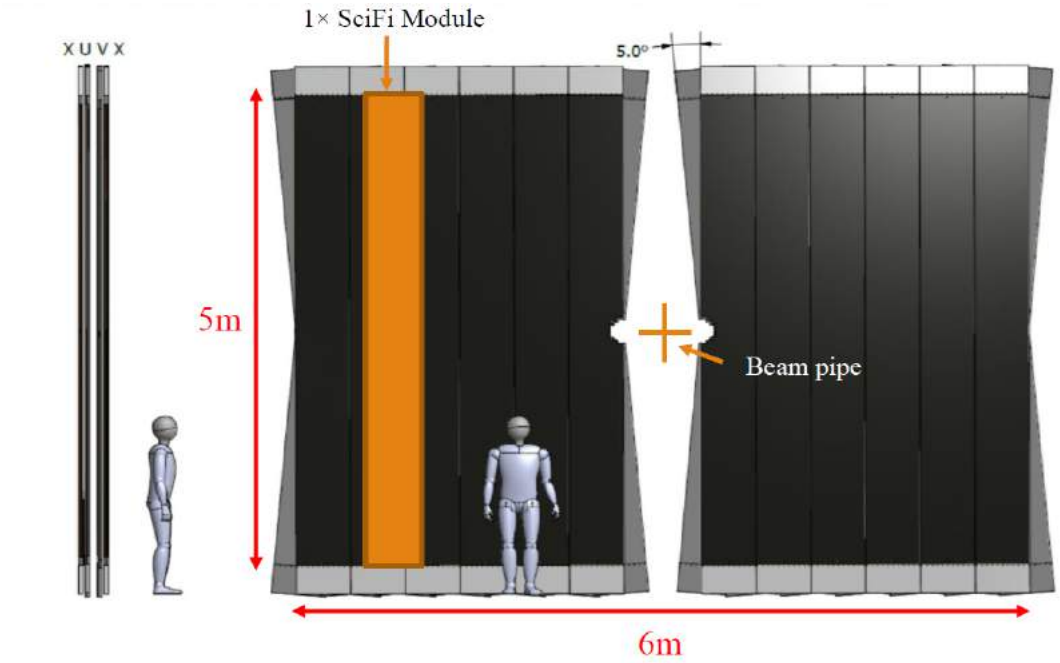

Figura 5.2: Modelo 3D do arranjo dos módulos do detector SciFi.

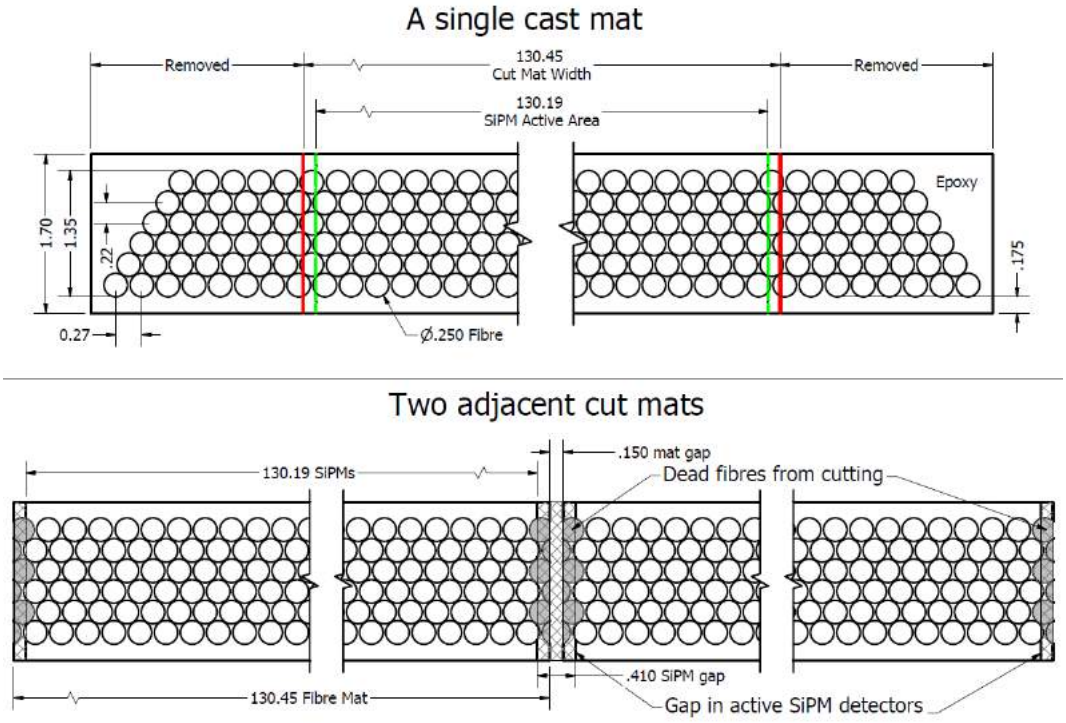

Figura 5.3: Ilustração do agrupamento da fibras em um *mat*.

Nas extremidades externas de cada módulo de fibra há um módulo de leitura chamado *Read-Out Box* (ROB), que abriga toda a eletrônica que efetua a leitura do detector. Nas extremidades internas, que ficam voltadas para o centro do detector, há espelhos para que a luz conduzida nesta direção seja refletida para a extremidade oposto onde há o módulo de leitura.

5.2 As Fibras Cintilantes

As fibras cintilantes são o elemento ativo de detecção do projeto SciFi. Estas são fibras de 250μm de diâmetro com um núcleo de poliestireno revestido com duas camadas cujos índices de refração são decrescentes de dentro para fora. A camada de revestimento interior é formada por polimetil-metacrilato (PMMA) e a camada exterior é feita de um polímero fluoretado. O projeto das camadas de revestimento das fibras são feitos de modo a otimizar a reflexão interna de luz, fazendo com que esta seja guiada pela fibra para as extremidades, onde são lidas por um sensor óptico. O núcleo das fibras é feito de material cintilante. Algumas partículas, ao cruzarem este material, depositam energia no mesmo, excitando os átomos ali contidos e causando uma reação que emite luz, a qual é chamada de cintilação.

A Figura 5.4 mostra uma ilustraçâo da fibra com suas camadas e o índice de refração de cada uma destas.

5.3 Módulos de Leitura

No final de cada módulo de fibra como os vistos na Figura 5.2, é anexado um módulo de leitura que recebe o nome de "*Read-out Box*" (ROB). Os ROBs são estruturas onde estão contidos os elementos de leitura do detector.

Um ROB é composto por uma *Cold Box*, que abriga 16 Fotomultiplicadoras de Silício (SiPM), e dois módulos da eletrônica de *front-end* (detalhada no Capítulo 6). A Figura 5.5 apresenta a visão explodida de um ROB, indicando seus diferentes componentes.

Uma das principais funções do ROB, que é de suma importância para o funcionamento do detector, é fornecer o acoplamento óptico entre as fibras e as SiPM. A Figura 5.6 exibe um corte da vista lateral da junção do ROB com o módulo de fibras. O ROB também

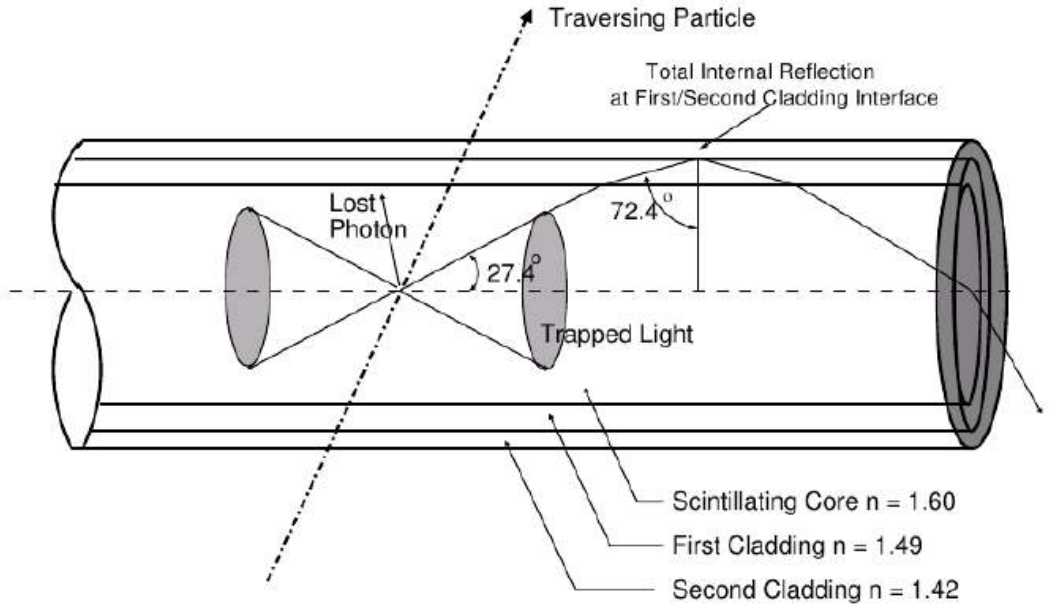

Figura 5.4: Luz é produzida no núcleo da fibra por uma partícula passante, e então é aprisionada até que se atinja a extremidade.

é responsável por fornecer resfriamento e estrutura mecânica aos componentes de leitura.

A placa de alumínio que compõe a parede do ROB, indicada na Figura 5.5 por *Cooling Plate*, atua como dissipadores para os chips de transmissão de dados utilizados na *Master Board*. Na parte inferior da Figura 5.5 podemos ver a *Cold Box*, cuja função já não é mais a dissipação de calor mas a conservação da temperatura interna. Esta é ligada ao sistema de resfriamento a fim de manter o conjunto de SiPMs em sua temperatura ótima de operação, que é aproximadamente -40°C. Na Figura 5.6 podemos ver o contato do cano do sistema de resfriamento *Cold pipe* à parte traseira do módulo do sensor SiPM.

5.4 Fotomultiplicadoras de Silício

Fotomultiplicadoras de silício (SiPM) são dispositivos de estado sólido utilizados para detecção de fótons. SiPMs são usadas para detecção da luz proveniente das fibras no detector do SciFi.

SiPMs são formadas por uma matriz de fotodiodos de avalanche em um substrato de silício. Tecnologias atuais permitem a criação de fotodiodos de avalanche com tamanhos

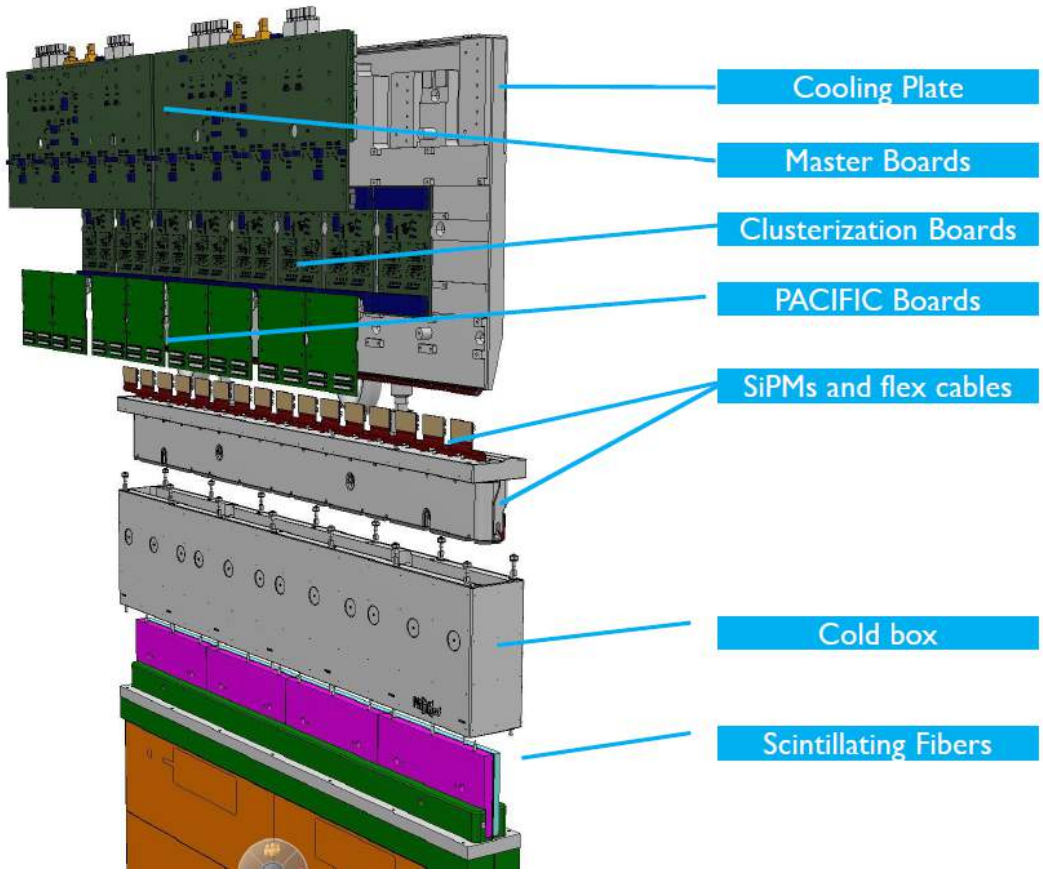

Figura 5.5: Visão explodida de uma *Read-out Box*.

de até 20μm, permitindo que uma matriz tenha até 1000 fotodiodos em um milímeto quadrado. Os diodos operam em modo geiger e são ligados em paralelo a saída da SiPM. Cada diodo corresponde a um pixel e o efeito avalanche pode ser disparado por um único fóton. Quando disparado, o diodo satura e gera um nível de corrente fixo e mensurável. Este nível independe do número de fótons incidente no mesmo píxel, pois um único fóton é o suficiente para saturar o diodo. A corrente medida na saída do SiPM será proporcional ao número de diodos saturados, ou seja, ao número de pixels iluminados.

Os modelos de SiPM cotados para uso no SciFi apresentam eficiência de fotodetecção de aproximadamente 40% e ganho de aproximadamente 2×10^6. Um dos modelos possui 128 canais, composto por 2 pastilhas de silício lado a lado, com 64 canais cada. Cada um dos 128 canais são formados por uma matriz de 4 por 24 pixels, totalizando 96 pixels por canal. Os píxels tem dimensões de 57,5$\mu m \times 62,5\mu m$.

A Figura 5.7 mostra a imagem de uma SiPM, com uma ampliação de seus canais e

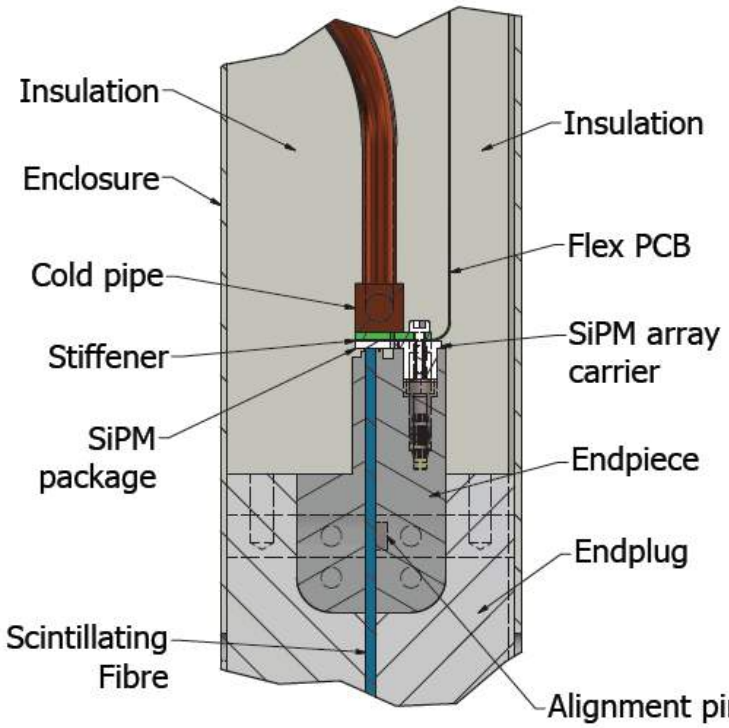

Figura 5.6: Corte da vista lateral da junção do módulo de fibras com o módulo ROB.

uma de um unico pixel.

Como podemos ver na Figura 5.8, quando uma partícula atravessa as fibras, é esperado que vários pixels na região das fibras atravessadas sejam iluminados. Embora cada canal tenha $250\mu m$ de espessura (Figura 5.7), o número de pixels iluminados nos canais adjacentes é uma indicação de quão próxima a partícula estava de qual canal. Comparando a quantidade de carga medida nos canais adjacentes e tirando uma média permite obtermos uma resolução bem inferior aos $250\mu m$ de largura do canal, obtendo os $60\mu m$ mencionados anteriormente.

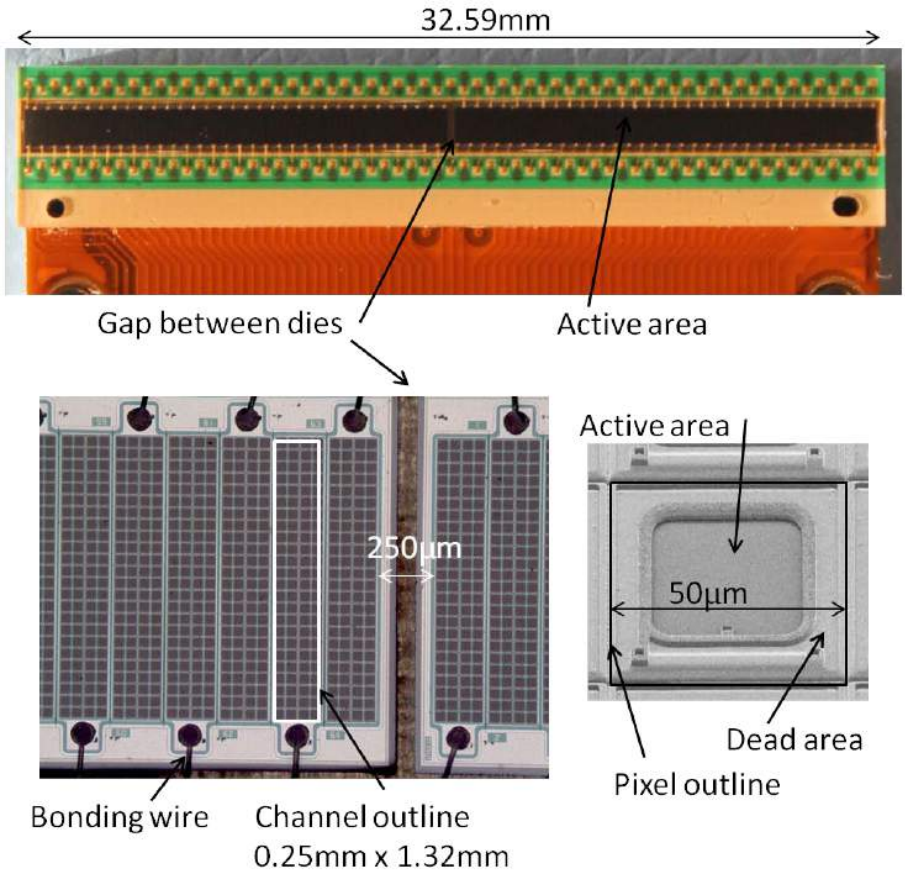

Figura 5.7: Topo: encapsulamento do sensor com 2 pastilhas de silício. Inferior esquerda: ampliação da parte central do sensor, onde se pode ver os canais individuais e o intervalo espacial entre eles. Inferior direita: ampliação de um único pixel.

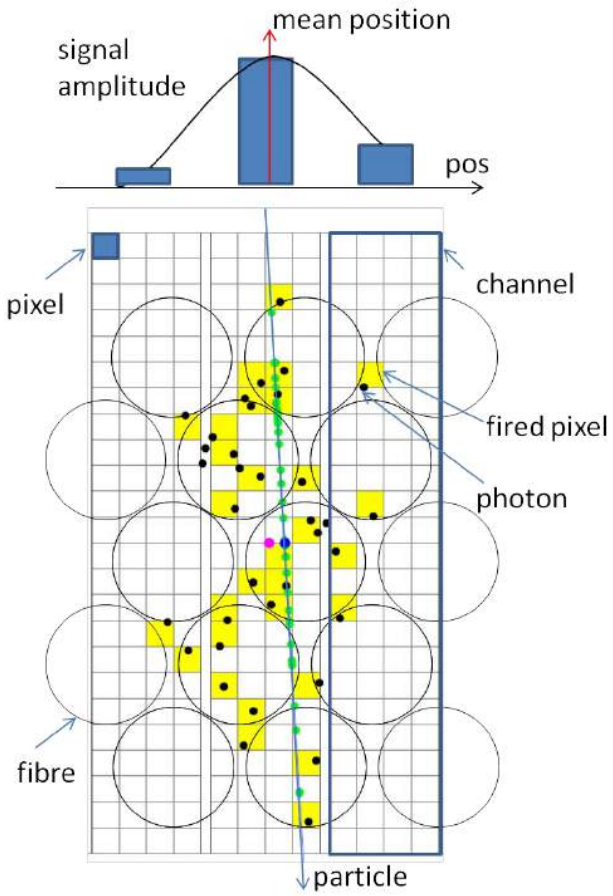

Figura 5.8: Fótons são gerados por cintilação no interior das fibras, e atingem os pixels marcados em amarelo.

Capítulo 6

Eletrônica de Front-End do SciFi

A eletrônica de *front-end* (FEE) é toda a eletrônica responsável pela interface entre o sinal da SiPM e o módulo de aquisição (contido na *back-end*). Ela é composta pelos módulos que ficam no local do experimento, juntos ao detector, e é responsável pela leitura do detector e envio dos dados à *back-end*. Adicionalmente a *FEE* também é responsável pelo controle e monitoramento do sistema do detector, recebendo comandos de controle e enviando relatórios solicitados sobre os status dos componentes. Como visto anteriormenete, a *front-end* ficará contida nos ROBs, que se encontrarão anexados na extremidade de cada módulo de fibras.

A eletrônica do SciFi é composta por 3 tipos de módulos que exercem funções de leitura, processamento e envio:

- **Pacific Board:** é o módulo responsável pela leitura, amplificação e digitalização dos dados provenientes das SiPM.
- **Clusterization Board:** é o módulo responsável pelo agrupamento dos dados em *clusters* para envio à *back-end*.
- **Master Board:** é o módulo que controla os demais, provê alimentação, clock, e é responsável pelo envio dos dados.

A *front-end* do SciFi utiliza os componentes do Projeto "*Giga-Bit Transceiver*" (GBT) [16]. Na *front-end* se encontram os chips GBTx e GBT SCA, ambos explicados na Seção 4.1. No SciFi, o GBT SCA é utilizado para comunicação com todos os dispositivos I2C da *front-end*.

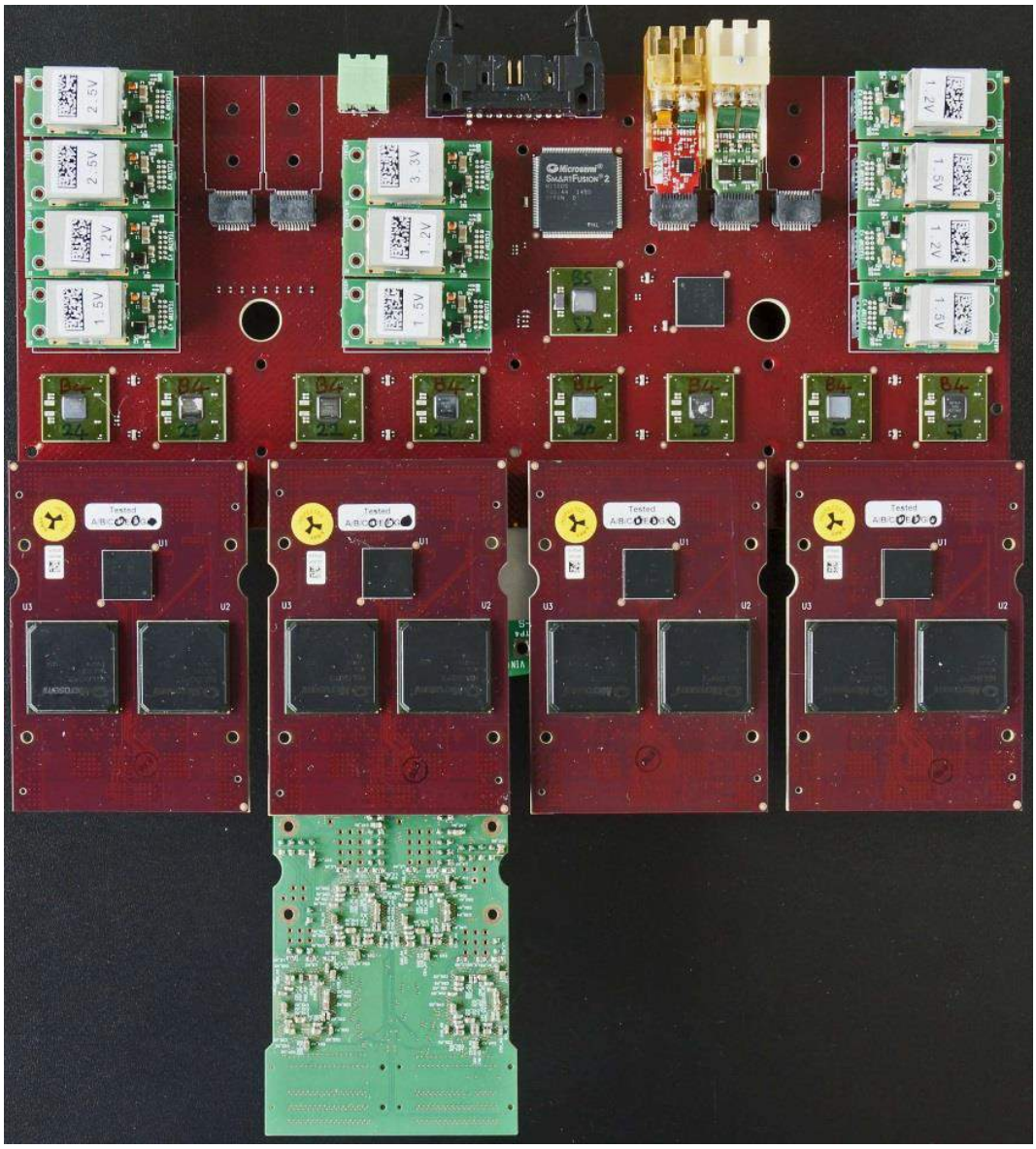

Figura 6.1: Imagem com protótipos das placas da *front-end*. Na parte superior se vê o protótipo da Master Board, com 11 conversores DC-DC plugados nela, e 2 módulos de transmissão ópticos. Na parte central, conectadas abaixo da Master Board se encontram 4 protótipos da Clusterization Board. Na parte inferior da imagem se vê um protótipo da Pacific Board conectado à segunda Clusterization Board.

6.1 Pacific Board e o Chip PACIFIC

A Pacific Board é o módulo que faz a leitura das SiPM. Ela contém 4 chips "*low-**P**ower **A**SIC for the s**CI**ntillating **FI**bres tra**C**ker*" (PACIFIC) [17].

PACIFIC é um circuito integrado de aplicação específica (ASIC) de sinal misto, desenvolvido com tecnologia 130nm CMOS, com o objetivo de ser o chip de leitura do SciFi. O PACIFIC possui 64 canais de entrada, consumindo menos de 8mW por canal e com taxa de amostragem de 40MHz. O PACIFIC é também controlado via I2C. Ele é responsável por amplificar e digitalizar o sinal das SiPM, convertendo estes para um valor digital de 2 bits.

Na Figura 6.2 podemos ver as etapas de processamento do sinal no chip PACIFIC.

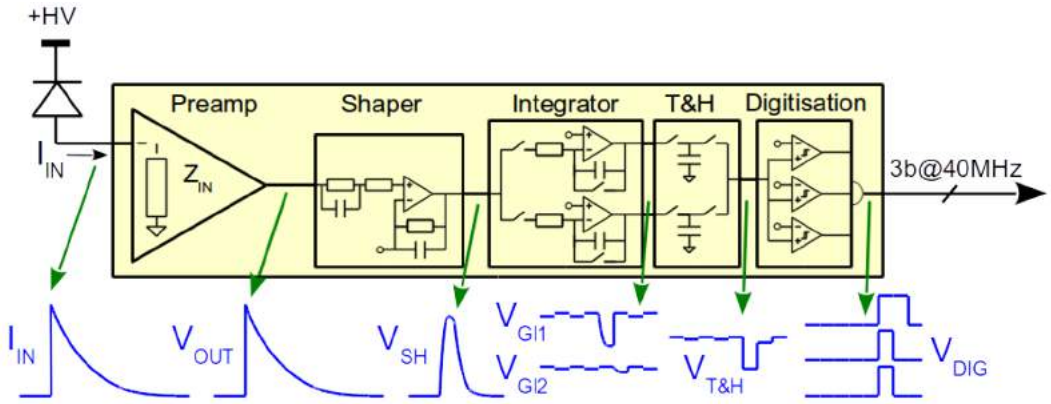

Figura 6.2: Cadeia de processamento de sinal por canal no chip PACIFIC.

Os itens da Figura 6.2 são explicados a seguir:

- O componente Preamp é um amplificador de transimpedância que converte o sinal de corrente proveniente da SiPM em um sinal idêntico com um ganho configurável aplicado. O ganho pode ser escolhido entre 4 opções (0,5x; 1x; 1,5x e 2x) através da interface I2C.

- O *Shaper* é um bloco de processamento de sinal que efetua cancelamento de polos e zeros. A finalidade é remover a cauda do sinal do SiPM e cancelar "*overshoots*"e "*undershoots*". Os valores para os polos e zeros, assim como no Preamp, também são configuráveis através de I2C.

- O *Integrator* é um bloco integrador, que utiliza a arquitetura clássica de integrador compreendendo um amplificador operacional com um resistor na entrada do sinal para a entrada negativa e um capacitor para a saída. Este bloco, no entanto, utiliza dois integradores que se alternam em função de clock, de modo a evitar tempo morto. Enquanto um está ativo, o outro está descarregando e vice-versa.

- O bloco T&H é um "*Track and Hold*", um circuito que amostra o valor do sinal e o mantém por um tempo, para que o sinal fique estável durante o tempo de conversão do conversor analógico-digital.
- o bloco "*Digitisation*"é o responsável pela digitalização do sinal. A saída do digitalizador é um sinal binário de 2 bits formado a partir de 3 comparadores cujos sinais de *thresholds* podem ser configurados pela interface I2C.

A Pacific Board contém 4 chips PACIFIC, permitindo portanto a leitura de 2 SiPM de 128 canais por placa PACIFIC. Cada Master Board suporta até 4 Pacific Boards, e por isso uma Master Board recebe o sinal de 8 SiPMs e consequentemente cada ROB lê 16 SiPMs.

Após digitalizado o sinal, este é enviado para a Clusterization Board.

6.2 Clusterization Board

A Clusterization Board é o módulo responsável pelo preparo dos dados vindos do PACIFIC para envio à *back-end*. Ela exerce duas funções principais: a formação de *clusters* e a supressão de zeros. A Clusterization Board possui um GBT SCA e dois FPGAs Igloo2 da Microsemi que são chamados *"Clusterization FPGAs"*. O papel do GBT SCA é a configuração dos FPGAs e dos chips da Pacific Board. Já os FPGAs são os responsáveis pelo tratamento dos dados e pelos algoritmos de supressão de zeros e clusterização. A cada Clusterization Board será conectada uma Pacific Board, e cada Master Board recebe 4 Clusterization Boards.

Os dados lidos do detector não são enviados de forma crua para a *back-end*, mesmo porque o volume de dados é grande demais para que fossem enviada a leitura de todos os canais de todas as SiPM. A maioria dos canais lê zero em grande parte dos eventos, uma vez que o número de partículas produzidas nas colisões é relativamente pequeno comparado ao número de canais que o detector possui. Este fato permite a utilização de uma técnica para lidar com o excesso de dados, que é a supressão dos zeros. Esta suprime os canais que lêem zero antes de enviar os dados. Assim salva-se banda que do contrário estaria sendo desperdiçada enviando zeros atrás de zeros. A supressão de zeros, por outro lado, pode aumentar o volume de dados enviados uma vez que quando se suprime alguns canais, é necessário enviar também a informação de quais canais foram ou não suprimidos. Se a

informação de quais canais são referentes aos dados enviados ocupar mais espaço do que o que se economiza com a supressão dos canais lidos zero, deixa de valer a pena a supressão de zeros. Os FPGAs de clusterização também são configuráveis via I2C e esta função pode ser habilitada ou desabilitada durante tempo de execução.

Outra importante função do FPGA de Clusterização é o algoritmo de clusterização. Este processa os dados dos canais do detector a fim de julgar o que será considerado uma partícula ou o que será considerado ruído. Na Figura 5.8 pudemos ver um exemplo de leitura nos canais da SiPM. Observe que uma partícula pode não atingir um único canal, mas ao invés, tende a iluminar mais pixels dos canais por onde sua trajetória mais se estendeu.

O FPGA de clusterização recebe os dados no formato de 2 bits por canal. Estes 2 bits significam qual o valor lido pelo PACIFIC em função dos thresholds. São 3 thresholds e por isto 4 possibilidades (2 bits). O algoritmo analisa se a carga total depositada em dado canal e nos adjacentes são o suficiente para considerar que a leitura se trata de uma partícula. Ou ainda, se o valor corresponde a duas ou mais partículas que acertaram canais adjacentes.

Na Figura 6.3 podemos ver exemplos de clusters.

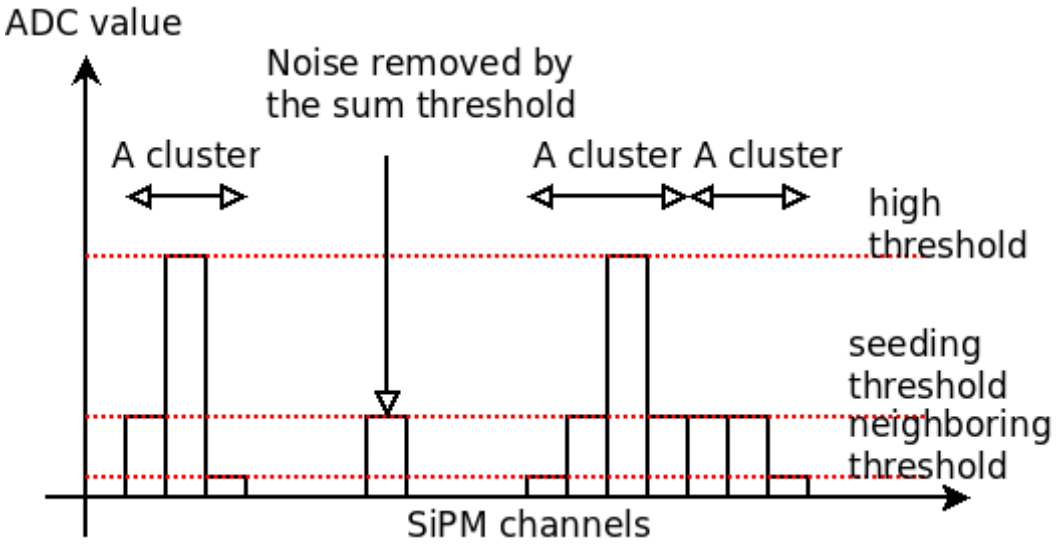

Figura 6.3: Leitura dos canais da SiPM com indicação do que seriam considerados clusters pelo algoritmo de clusterização.

À esquerda da Figura 6.3 houve uma leitura que foi considerado um *cluster*. Ao centro há uma leitura que o algoritmo não considerou como *cluster* pois o somatório do

valor do canal e adjacentes não somou o suficiente para representar uma partícula. Já a leitura da direita, o algoritmo interpretou como dois clusters. Primeiro porque o número de canais atingidos foi mais do que 4, e também porque a energia total seria o suficiente para considerar como dois.

6.3 Master Board

A Master Board é o módulo principal que compõe a eletrônica de *front-end*. Ela é responsável por gerenciar os demais componentes da *front-end*, fornecer alimentação, gerar e distribuir sinais de controle, clock, e enviar os dados provenientes da clusterização.

Os principais componentes da Master Board são listados a seguir, e podem ser vistos na Figura 6.4:

- **GBTx Master**: é o chip responsável por toda a comunicação com a *front-end*. É um GBTx configurado como mestre. Ele é a porta de entrada, que distribui todos os comandos enviados à *front-end* para os GBT SCAs através dos e-links. O GBTx Master está marcado como "MSTR GBT" na Figura 6.4.

- **SCA Master**: é o GBT SCA responsável pelo controle dos demais componentes da Master Board. Note que ele é chamado Master apenas por estar na Master Board, e não por ter algum tipo de configuração mestre-escravo. Este GBT SCA recebe comandos do GBTx Master através de um e-link e através de suas portas I2C é feito a configuração dos 8 GBTx de dados e do HouseKeeping FPGA.

- **GBTx de dados**: são os GBTx responsáveis pelo envio dos dados provenientes das Clusterization Boards. A saída de cada GBTx de dados é ligada a um canal dos transmissores ópticos VTTx.

- **HouseKeeping FPGA**: abreviado para HK FPGA, é o dispositivo responsável pelo gerenciamento local dos componentes da Master Board. Ele administra os sinais de reset dos demais componentes, status da alimentação e dos conversores DC, e controla a sequência de inicialização dos componentes. Trata-se de um FPGA Microsemi M2GL005-vf256, com configuração baseada em memória FLASH, o que contribui para ser resistente a radiação.

- **Módulos de transmissão óptica**: a Master Board utiliza módulos Versatile Link VTTx e VTRx. Versatile Link é um projeto de transmissores ópticos resistentes a radiação, para serem utilizados no upgrade do LHC. O projeto compreende alguns tipos de transmissores. Os utilizados na Master Board são 4 "*Versatile Transmitter-Transmitter*" (VTTx) e 1 "*Versatile Transmitter-Receiver*" (VTRx). O VTTx dispõe de 2 transmissores ópticos, cada um transmite os dados de 2 GBTx. O VTRx possui um transmissor e um receptor, sendo este o utilizado pelo GBTx Master, o único componente do módulo que possui um link bidirecional.

- **Fontes de alimentação**: a Master Board utiliza módulos conversores DC-DC para alimentar seus componentes e as placas anexadas. Ela utiliza módulos de 1,2V, 1,5V, 2,5V e 3,3V. Os módulos são projeto do CERN para um conversor DC-DC tolerante a radiação e pequeno em massa. Mais sobre os conversores pode ser encontrado em [18].

6.4 Caminho de Dados Alternativo

Pelas informações expostas nas seções anteriores, o caminho de dados do detector à eletrônica de *back-end* pode ser sintetizado da seguinte forma: ao incidir luz sobre os pixels da SiPM, serão gerados pulsos elétricos em sua saída. O chip PACIFIC recebe estes pulsos, amplifica, trata o sinal, digitaliza e o transforma em um sinal digital de 2 bits que correspondem a 4 faixas diferentes estebelecidas por 3 thresholds programados no PACIFIC. Os FPGAs de clusterização efetuam a leitura dos canais, analisam e formam os *clusters*, os agrupam em palavras do protocolo GBT e enviam aos chips GBTx na Master Board. Os GBTx serializam os dados e enviam por um módulo VTTx para a eletrônica de *back-end* que faz a aquisição.

Este método será utilizado para a região central do detector, no entando, para as extremidades onde a luminosidade é significantemente mais baixa, o volume de dados gerado também é menor. Não há necessidade de se utilizar 8 GBTx de dados e 8 canais ópticos de transmissão. Por esta razão, o caminho de dados da *front-end* como um todo foi projetada de forma a ser possível retirar metade dos GBTx e metade dos módulos VTTx sem haver prejuízo na transmissão dos dados além da redução da banda máxima.

Na Figura 6.4 podemos ver o diagrama ilustrando o camindo de dados.

Note que os dados saindo dos FPGAs da direita nas Clusterization Boards seguem sempre para o GBTx à direita no par imediatamente acima. Os GBTx da direita de cada par, segue para os VTTx também a direita dos pares. Se numerarmos os componentes da esquerda para a direita, teremos o seguinte:

Os dados dos FPGAs de clusterizaçâo 2, 4, 6 e 8 seguem para os GBTx 2, 4, 6 e 8, que então seguem para os VTTx 2 e 4. De forma análoga os FPGA ímpares seguem os GBTx e VTTx ímpares.

Existe uma seta pontilhada em cada Clusterization Board, partindo dos FPGAs de clusterização pares em sentido aos ímpares. Isto representa um barramento de dados implementado entre os FPGAs que torna possível o desvio dos dados dos FPGA pares aos ímpares, desta forma, possibilitando que todos os dados sejam transmitidos apenas pelos componentes ímpares.

Os pares de GBTx seguem para os VTTx de forma alternada justamente para que seja possível, nas placas que ficarão nas áreas de menor ocupação, não utilizar os GBTs e VTTx pares. Economizando assim 4 GBTx e 2 VTTx por Master Board.

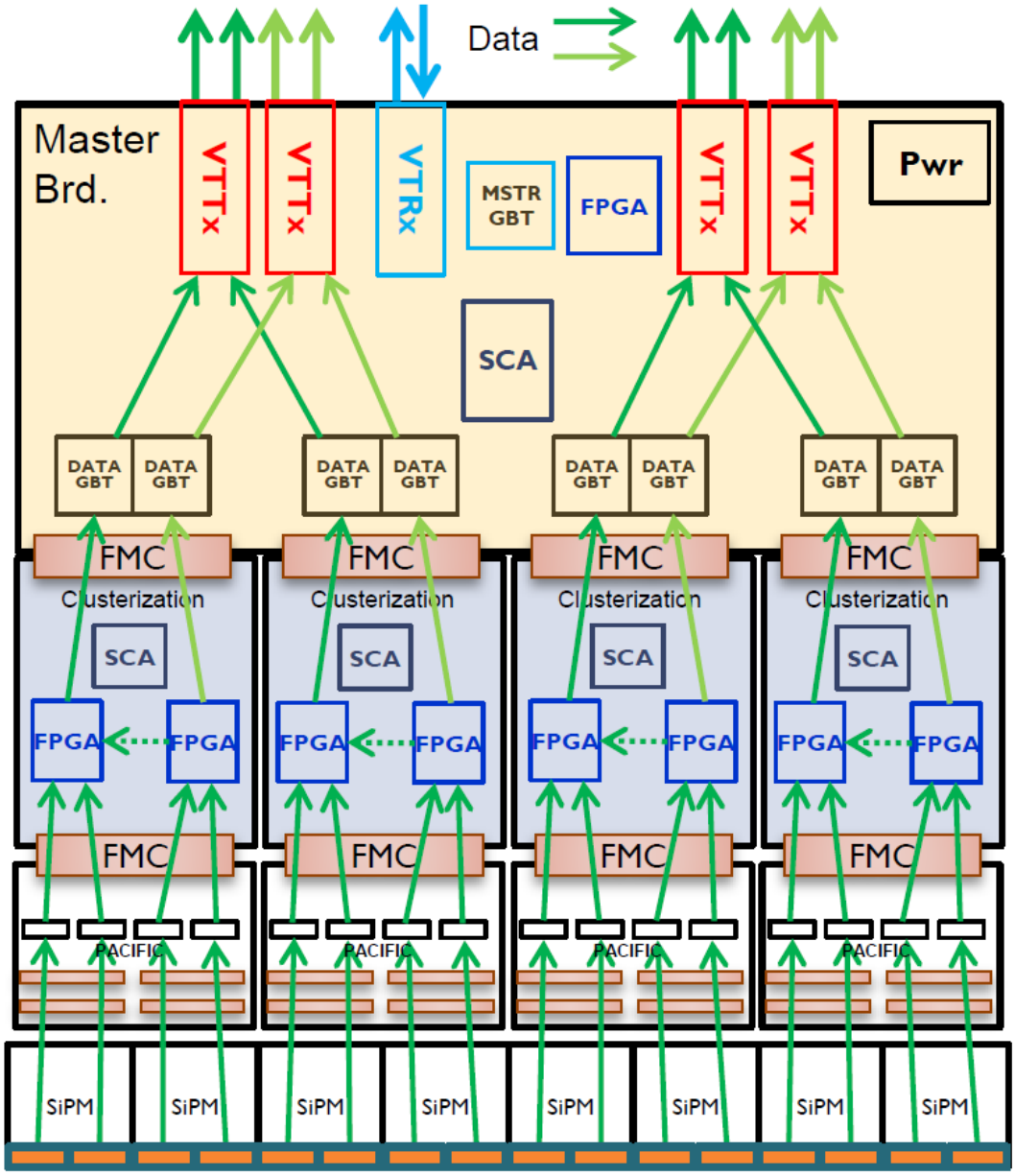

Figura 6.4: Diagrama de blocos do caminho de dados da eletrônica de *front-end*.

Capítulo 7

Sistema de Teste da Eletrônica do SciFi

Após a etapa de produção em massa a eletrônica de leitura precisará ser submetida a um processo de controle de qualidade antes da etapa de instalação. Um total de aproximadamente 320 módulos de *front-end* (com 10% sobressalente) deverão ser produzidos. O objetivo de um sistema de teste automático é garantir a qualidade individual de cada *Read-Out Box* em todos os postos de produção.

O sistema será capaz de executar as seguintes tarefas:

- detectar canais não funcionais;
- detectar canais ruidosos;
- verificar as funcionalidades do chip PACIFIC como uniformidade de ganho, características de *threshold*, DAC da tensão da SiPM, cancelamento de zeros e polos e a linha de base do *Shaper*;
- determinar *crosstalk* entre canais;
- detectar diferenças de fase intrínsecas de canais;
- verificar a integridade de fragmento de evento

Um diagrama funcional do sistema de teste é mostrado na Figura 7.1. Este consiste de três componentes principais em adição ao ROB sendo testado: um sistema de injeção,

uma unidade de controle e aquisição, e um computador. O sistema de injeção consistirá de um gerador de sinais, capaz de emitir pulsos em todos os canais, variando o tempo de fase do sinal de entrada e a quantidade de carga injetada. O MiniDAQ será utilizado como unidade de controle e aquisição. Um firmware dedicado compreenderá ambos os blocos de TFC e ECS, e um software de controle WinCC proverá a interface para execução dos testes de parâmetros na *front-end* (*threshold*, temporização, etc.) e armazenará os dados no PC. Um pacote ROOT automático de análise proverá um diagnóstico completo de caracterização do ROB.

Com o objetivo de executar o teste do módulo de leitura um sistema de injeção será produzido, em que será capaz de emular sinais de saída com as mesmas características de carga dos sensores SiPM. Ao total 2048 canais serão gerados independentemente com amplitudes variando de 25mV a 61mV, largura variando de 5ns em passos de 390ps e a defasagem variando também em passos de 390ps. Um grande número de passos é possível. Para lidar com o grande número de canais e consequentemente a alta densidade de componentes, será adotado um desenho fatorizável que compreende um módulo de controle, conectado ao MiniDAQ através de um link óptico, que sincroniza e controla 8 placas injetoras, conectadas à placa de controle através de 8 cabos HDMI padrão. Cada módulo injetor é composto por um FPGA e 16 grupos de 16 canais analógicos.

Com o intuito de garantir a sincronia de todos os 2048 canais, um cabeamento complexo, baseado em 16 cabos *ribbon* customizados com conectores ZIF no lado do injetor, foi desenvolvido conforme mostrado na Figura 7.2.

7.1 Placa de Controle do Sistema de Injeção

A placa de controle é uma verão reduzida da *Master Board* da eletrônica de *front-end*. Esta utiliza um link óptico VTRx assim como um chip GBT e GBT-SCA Para TFC a placa provê uma referência de tempo através do clock mestre (*deskewable*) e um pulso de calibração para selecionar que saída será disparada. Neste caso os caminhos físicos do GBT são ClkDes e linhas de comando. O *slow control* faz uso de um barramento I2C e uma porta paralela GPIO para configurar cada canal como ligado/desligado, amplitude, fase e largura e o reset da placa de injeção. A Figura 7.3 representa o diagrama de blocos da placa de controle do sistema de injeção.

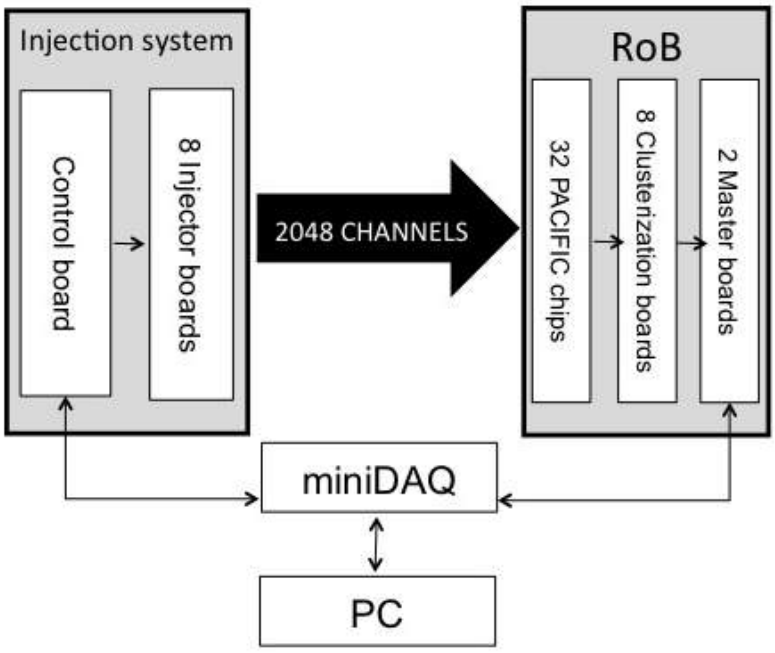

Figura 7.1: Diagrama de blocos do sistema de teste de ROB.

7.2 Placa Injetora do Sistema de Injeção

A placa injetora é dividida por um estágio digital, compreendendo um FPGA Cyclone IV e um DAC, um estágio analógico, organizados em grupos de 16 canais, compreendidos por deslocadores de nível, *buffer* e amplificadores.

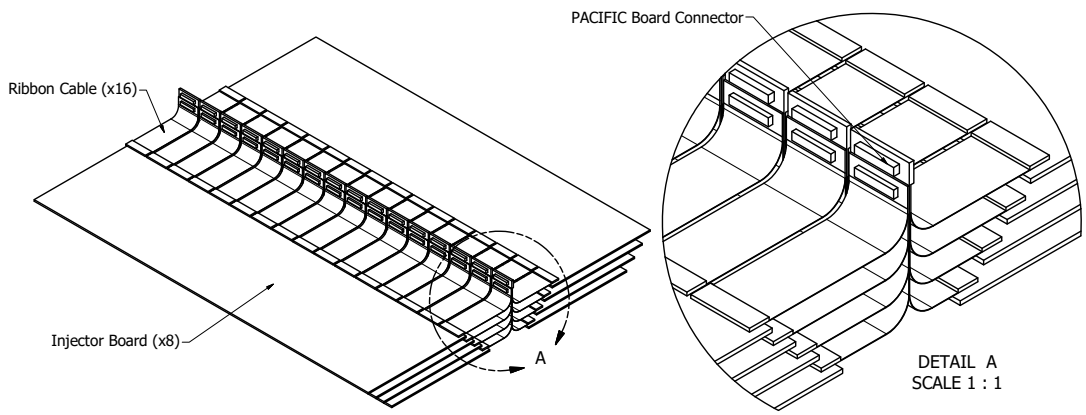

Figura 7.2: Geometria do sistema de injeção, destacando a conexão entre o chip PACIFIC e a placa injetora.

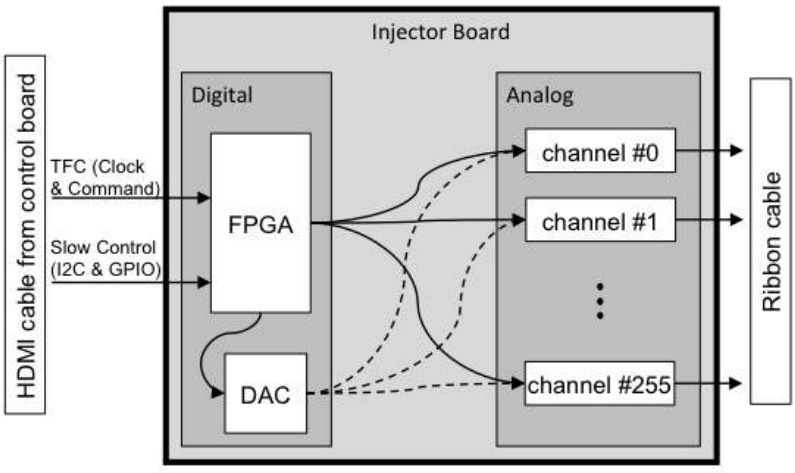

Figura 7.4: Diagrama de blocos da placa injetora.

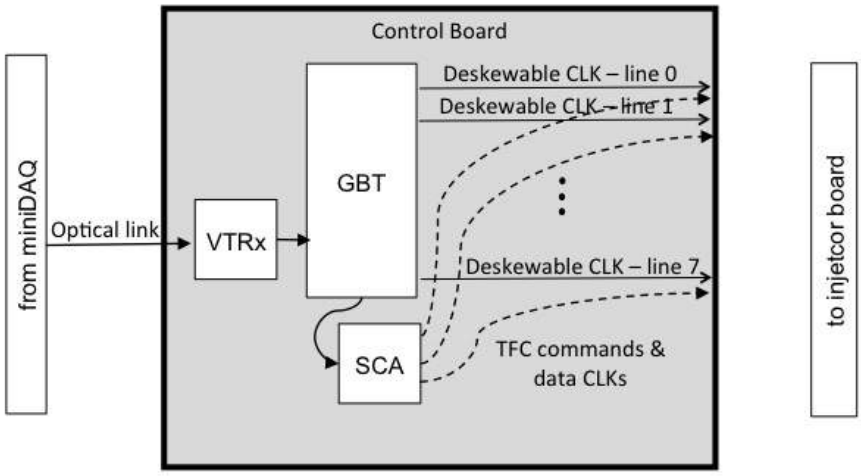

Figura 7.3: Diagrama de blocos da placa de controle do sistema de injeção.

O *trigger* de cada canal é implementado por uma máquina de estado no FPGA onde duas PLLs são usadas para controlar a defasagem e largura dos pulsos com resolução de 10^{-1}ns. O FPGA controla um DAC de 8 bits responsável por gerar um nível DC. O nível DC alimentará um circuito composto de deslocadores de nível de alta velocidade e amplificadores com a finalidade de controlar a amplitude de canais individuais. Como um estágio de saída, um divisor resistivo e um *buffer* foram utilizados para reescalar a combinação de amplitude e impedância.

7.3 Protótipo da Placa Injetora

Este conceito da placa injetora foi validado por um prtótipo de 16 canais, mostrado na Figura 7.5. Neste teste os pulsos de saída foram sincronizados por um gerador de clock externo e os parâmetros foram alterados diretamente do firmware do FPGA. A performance foi estudada cuidadosamente em todos os canais utilizando diferentes configurações e o total de carga depositada foi calculado. Foram escolhidos 4 configurações mostradas na Tabela 7.1 e na Figura 7.6 para qualificar quantitativamente o projeto. Além das funcionalidades dos sinais de saída, a viablidade do projeto da PCB, contendo a alta densidade de componentes, foi também demonstrado pelo protótipo.

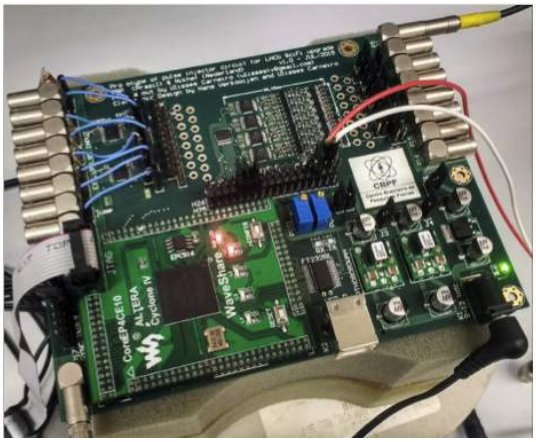

Figura 7.5: Fotografia do protótipo da placa injetora de 16 canais.

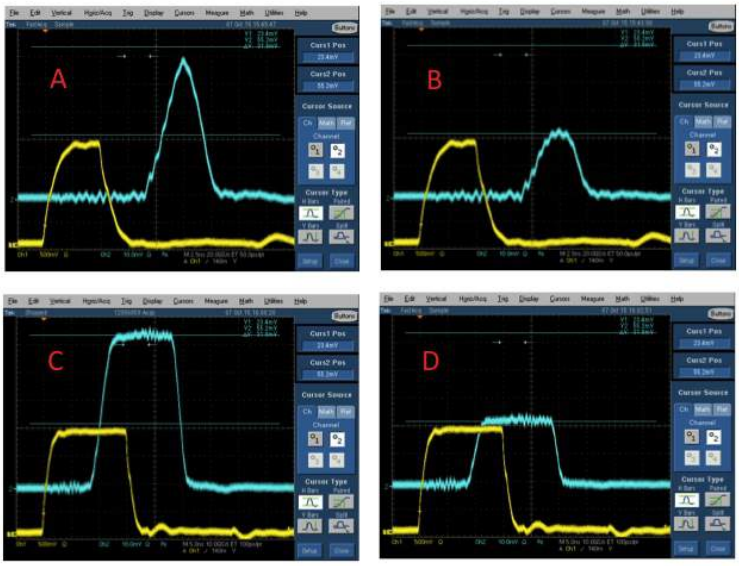

Figura 7.6: Resultados obtidos para o conjunto de configurações descrito na Tabela 7.1 utilizando um osciloscópio Tektronix TDS7254 de 10 GS/s. O sinal amarelo é a referência de tempo externa.

Tabela 7.1: Quatro configurações utilizadas para validação da placa injetora.

label	**amplitude**	**width**	**deposited charge**
A	50 mV	5.2 ns	3.1 pC
B	23 mV	5.2 ns	1.6 pC
C	55 mV	16 ns	14.6 pC
D	23 mV	16 ns	6.5 pC

Capítulo 8

MiniDAQ

O MiniDAQ (Figura 8.1) é uma plataforma autônoma de aquisição de dados, desenvolvida pelo Centro de Física de Partículas de Marselha (CPPM), inicialmente para ser usada como eletrônica de *back-end* do experimento LHCb. Ele é composto por um módulo no padrão AMC (*Advanced Mezzanine Card*) contendo um FPGA. Este módulo de um lado possui interface para 36 links ópticos bidirecionais, e do outro um conector do padrão AMC que por sua vez é conectado a um módulo que contém um pequeno computador que é chamado de CCPC (*Credit Card PC*) por ter dimensões semelhantes às de um cartão de crédito.

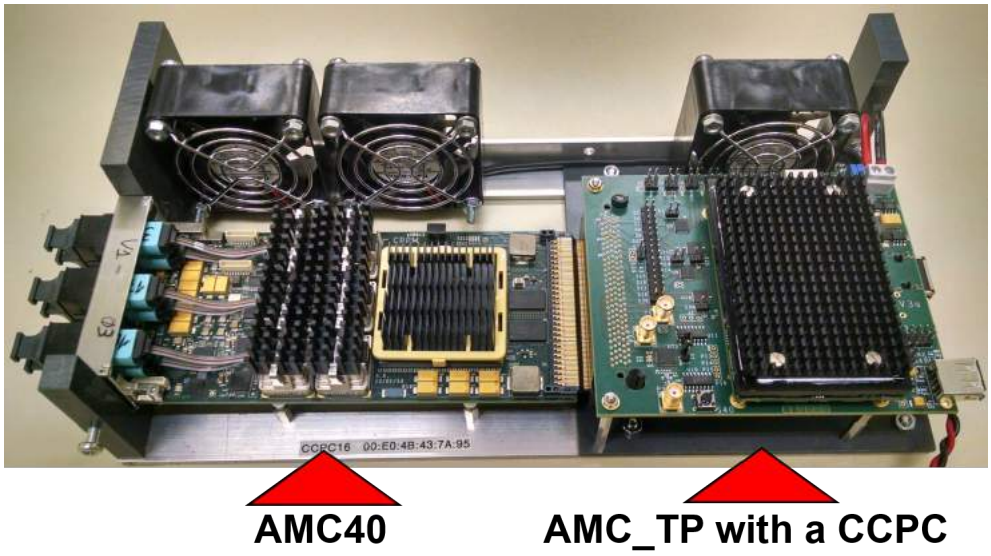

Figura 8.1: MiniDAQ

A primeira placa, que contém o FPGA e é utilizada para aquisição, envio e processamento dos dados, é chamada AMC40. A segunda, que contém o CCPC, é utilizada para controle e configuração e é chamada AMC_TP.

O MiniDAQ não será utilizado como *back-end* do experimento LHCb por não atender à alta demanda de transmissão de dados que o experimento exige. Esta limitação não é imposta pela velocidade dos links ópticos, mas sim devido ao número de células lógicas disponíveis no FPGA, que não é suficiente para sintetizar o número necessário de blocos ethernet e alcançar a performance necessária utilizando todos os canais ópticos. Os MiniDAQs já produzidos, no entanto, serão utilizados para outras aplicações, entre elas o sistema de teste do detector de fibras cintilantes.

8.1 AMC40

A AMC40 é responsável pela aquisição, processamento e envio de dados. Ela contém 36 links ópticos bidirecionais, dentre os quais 24 são utilizados para comunicação com um chip GBTx a 4.7Gbps e 12 para Ethernet a 10Gbps. Os principais componentes encontrados nela são:

- Um FPGA da família Stratix V GX (5SGXEA7N2F45C3N)
- Dois bancos de memória DDR3 de 16 bits.
- Três PLLs CDC62005 onde duas são utilizadas para prover um sinal de clock aos serializadores e desserializadores do FPGA e uma terceira é utilizada para limpeza do jitter do sinal de clock do TFC.
- Seis componentes ópticos de 12 canais cada.

O transmissor e receptor óptico escolhidos para ser utilizado na AMC40 foi o MiniPOD. O MiniPOD é um módulo que contém 12 transmissores ou receptores ópticos, alojados dentro de um encapsulamento onde se encaixa um cabo óptico flat de 12 vias. A AMC40 possui 3 MiniPODs receptores e 3 MiniPODs transmissores, formando o total de 36 links ópticos bidirecionais. Cada link pode alcançar uma velocidade de transmissão de 10 Gbps, suficiente para o protocolo GBT que utiliza 4.7 Gbps. Os links dedicados à Ethernet operam a 10 Gbps. A distância máxima alcançada é de 150m e o consumo

máximo de potência por par de módulos é de 3W. A radiação de laser proveniente do emissor é de classificação 3R.

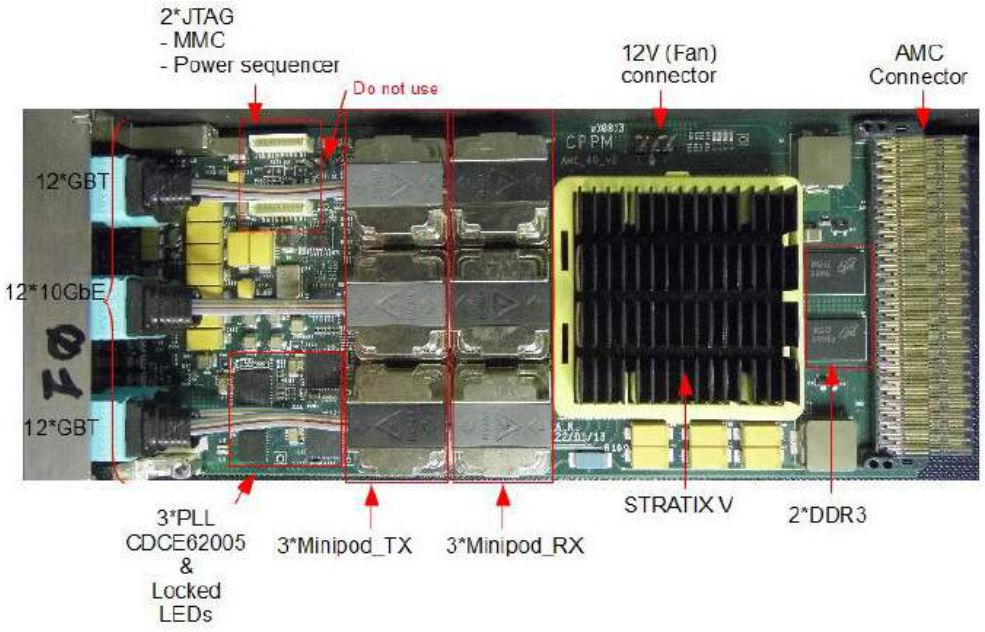

Figura 8.2: Vista superior da AMC40

Os cabos ópticos provenientes dos 6 MiniPODs terminam em 6 conectores MPO machos, que são fixados a uma estrutura metálica na extremidade da AMC40, que pode ser vista à esquerda da Figura 8.2. Nesta estrutura metálica também se encontram os LEDs indicativos, um interruptor de energização da placa e um conector JTAG para configuração do FPGA. Estes conectores ópticos são toda a interface da parte de aquisição e envio de dados. A parte frontal com a estrutura metálica da AMC40 pode ser vista na Figura 8.3.

Na extremidade oposta de onde se encontram os conectores ópticos, temos o conector AMC. Através deste conector é passada a alimentação de 3.3V e 12V para a AMC40 assim como sinais de trigger, clock e controle para o FPGA, e uma conexão PCIexpress geração 1 entre o CCPC da AMC_TP e o FPGA da AMC40.

A AMC40 pode ser usada para uma variedade de funções, tais como controle e aquisição de dados. Todos os caminhos de dados da placa (tanto os links ópticos quanto a interface PCIexpress do conector AMC) são ligados ao FPGA. Por consequência, o que define qual será a função da AMC40 é o firmware contido no FPGA.

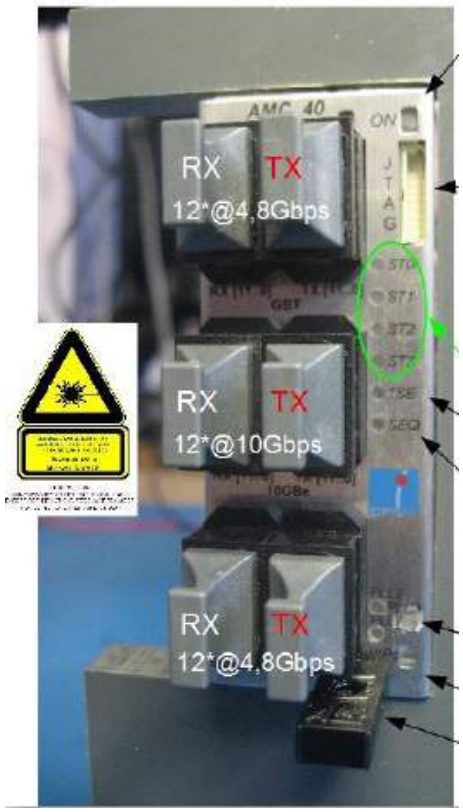

Figura 8.3: Parte frontal do MiniDAQ

8.1.1 Firmware do MiniDAQ

O firmware desenvolvido para o Strativ V da AMC40 é resultado de uma cooperação de diversos institutos ligados ao LHCb e utiliza o GBT-FPGA para implementação do protocolo GBT em FPGA.

O firmware compreende 2 blocos principais: A LLI (Low Level Interface) e o SOL40, que são descritos a seguir:

LLI: A LLI é a parte responsável pela aquisição de dados. Integrado nela há o codificador e decodificador de protocolo GBT do projeto GBT-FPGA. Na figura 8.4 temos uma sequência de blocos:

- Error Recovering: Este bloco é parte do protocolo GBT e implementa o método de correção de erro Reed-Solomon, dedicando 32 dos 112 bits de dados para esta tarefa. O usuário do MiniDAQ tem a opção de compilar um firmware que desvia os dados

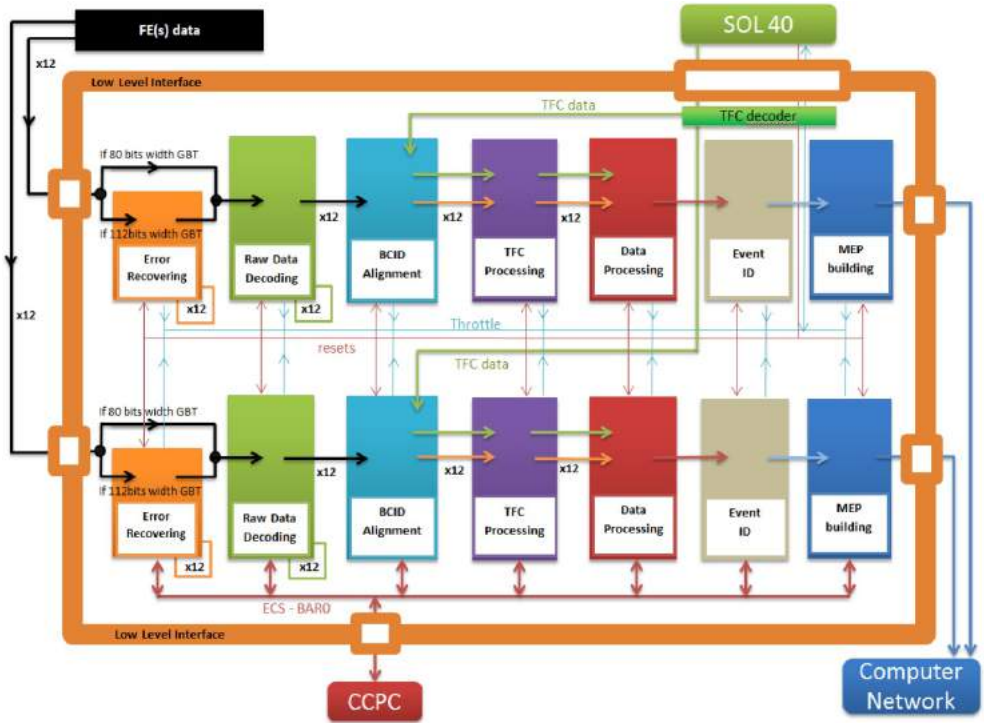

Figura 8.4: Diagrama de blocos simplificado do firmware do MiniDAQ.

deste bloco, de modo a aproveitar todos os 112 bits de dados da palavra GBT, mas sem a aplicação do algoritmo de correção de erro.

- Raw Data Decoding: Este bloco é onde ocorre a decodificação (no caso do receptor) ou a codificação (no caso do transmissor) da palavra de dados do GBT. É nele onde ocorrem as etapas de embaralhamento (*scrambling*) e entrelaçamento (*interleaving*) dos dados.

- BCID Alignment: Este bloco é responsável por alinhar os pacotes de dados provenientes de diferentes eventos. Não necessariamente os pacotes provenientes das diferentes *front-end*s chegarão em ordem de evento.

- TFC Processing: Este é o bloco responsável por processar os bits dedicados a TFC (Trigger and Fast Control) da palavra GBT.

- Data Processing: Este é o bloco onde o usuário final do MiniDAQ poderá inserir um pedaço customizado de firmware. É aqui onde ocorrerá o processamento de dados que é específico para cada grupo ou detector. Os demais blocos são comuns aos diferentes detectores e o usuário final não precisa alterá-los.

- Event ID: Neste bloco é onde será atribuído um número identificador ao evento que gerou os dados sendo processados antes que estes sejam enviados pela conexão ethernet.

- MEP building: Este bloco é responsável por montar pacotes ethernet com os dados de múltiplos eventos, de forma organizada, e enviar através da interface ethernet de 10 Gbps. Estes pacotes são chamados de MEP, que significa Pacote de Multiplos Eventos (Multi Event Packet). Neste bloco é onde está implementada a interface ethernet e seus protocolos. Ele utiliza serializadores e desserializadores do FPGA Stratix V para enviar estes dados a 10 Gbps para o computador de aquisição.

SOL40: O firmware SOL40 é o responsável pela parte de controle do firmware do Mini-DAQ. Ele recebe comandos de controle provenientes do CCPC. Os comandos de controle são divididos em 2 tipos:

- TFC (Trigger and Fast Control) é a parte de controle onde os comandos são responsáveis pela função de gatilho, sincronização, distribuição de clock e todas as operações que exijam uma resposta rápida, síncrona e precisa dos caminhos de dados. Na palavra GBT, 2 bits são dedicados à transmissão de comandos TFC. Estes são separados no chip GBT da *front-end* e disponibilizados em um dos e-ports do GBT, que é ligada ao GBT SCA Master. O GBT SCA Master interpreta estes comandos e reage de acordo, atuando nos demais componentes da placa através de seus sinais de controle e comunicação I2C.

- ECS (Experiment Control System) é a parte de controle onde os comandos são de configuração e monitoramento do sistema. Onde não há necessidade de uma resposta síncrona ou com tempos de resposta restritos. Os comandos ECS, assim como o TFC, também são transmitidos através de bits dedicados a eles na palavra GBT.

A função do firmware SOL40 é decodificar os comandos provenientes do CCPC e transformá-los nos comandos correspondentes para os respectivos GBT SCAs da *front-end*. Os comandos recebidos no firmware são escritos em registradores sintetizados pelo bloco do SOL40, e que estão também ligados ao barramento que faz a interface com o CCPC. Desta forma, para enviar comandos ao bloco SOL40, o software sendo executado no CCPC escreve os comandos nestes registradores e logo em seguida escreve em um registrador especial que engatilha a execução dos comandos.

8.2 CCPC

CCPC (Credit Card PC) é um tipo de computador de placa única, cujo padrão foi adotado por vários grupos da colaboração do LHCb.

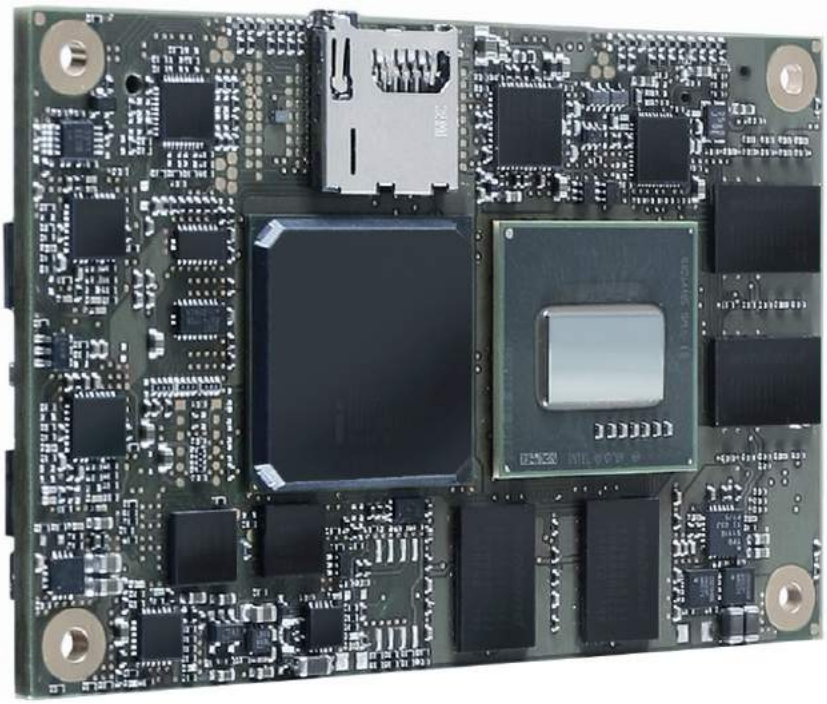

Figura 8.5: CCPC utilizado no MiniDAQ

O CCPC utilizado no MiniDAQ é fabricado pela KONTRON, da familia nanoETXexpress-TT, caracterizada por suas dimensões reduzidas de placa de circuito impresso, que medem 55mm x 84mm. Um modelo desta família pode ser visto na Figura 8.5. O modelo do CCPC contido na placa AMC_TP é o nETXe-TTc E680 1GB/mSD.

Os componentes do CCPC são:

- Processador Intem Atom de 1.6GHz, 1 núcleo com 2 threads e 512KB de L2 cache.
- 1GB de Memória RAM DDR2-800 de canal único (32bit).
- Processador gráfico Intel GMA 600 de 400MHz.
- Controlador Ethernet Broadcom BCM54610 com suporte até 1 Gbps.
- Interface PCIe, USB e suporte a cartão de memória micro SD.

Alguns modelos desta família de CCPCs podem utilizar um drive SSD para armazenamento, no entanto o CCPC utilizado no MiniDAQ não tem armazenamento local mas, ao invés, monta seu disco em um servidor remoto a partir da rede como veremos na seção 8.2.1. Esta decisão se dá devido à estrutura de rede do experimento LHCb, onde os computadores recebem a imagem do sistema a partir da rede, devido a diversas razões como manutenção escalável de todas as máquinas e segurança.

8.2.1 Boot do CCPC

O CCPC contido no MiniDAQ vem por padrão configurado para fazer boot pela rede, ou seja, ele utiliza uma imagem de sistema operacional hospedada em um disco na rede e acessível através de um servidor de boot utilizando o método chamado PXE (Pre eXecution Environment). Para realizar o boot, o CCPC precisa de um servidor DHCP para fornecimento de IP ao CCPC, um servidor TFTP para transferências dos arquivos do sistema do CCPC e um servidor NFS que administra o acesso a arquivos em disco rígido remoto através da rede.

O procedimento de boot ocorre nas seguintes etapas, correspondentes à numeração presente na Figura 8.6:

1. Ao inicializar, o CCPC procura um servidor DHCP na rede através de uma mensagem DHCPDISCOVER.

2. O servidor DHCP responde com um IP para o CCPC e mais três parâmetros: o endereço de um servidor TFTP, um arquivo para solicitar a este TFTP que é responsável por carregar o Kernel do Linux, e um caminho para ser montado o disco virtual do CCPC.

3. O cliente envia uma requisição TFTP ao servidor informado pelo DHCP, solicitando o arquivo também informado pelo DHCP.

4. O servidor TFTP responde a solicitação.

5. O cliente executa o arquivo recebido, que então carrega o kernel Linux. Quando este entra em execução, o sistema de arquivos raiz é montado no servidor NFS, no caminho informado no passo 1 pelo servidor DHCP.

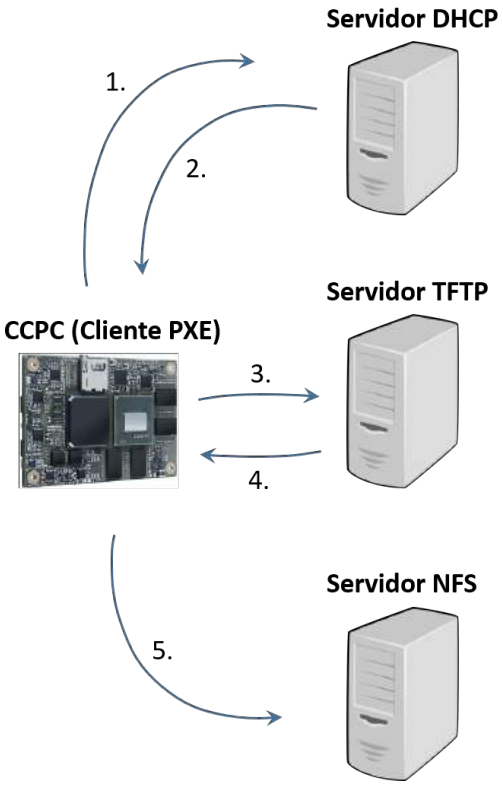

Figura 8.6: Procedimento de boot PXE.

Embora na Figura 8.6 os servidores DHCP, TFTP e NFS estejam ilustrados como hóspedes em máquinas diferentes, nada impede que os mesmos estejam na mesma máquina.

Todo o procedimento de configuração do servidor de boot utilizado durante este trabalho foi registrado para utilização em uma máquina virtual e pode ser visto no Apêndice C.

8.2.2 Máquina Virtual para Boot do CCPC

Configurar estes servidores para fazer o boot do CCPC corretamente não é uma tarefa trivial e exige um certo nível de conhecimento e tempo de trabalho. A forma como foi implementada também exigia uma máquina com sistema operacional Linux, o que nem sempre é o mais conveniente. Isto requeria a preparação de uma máquina com sistema Linux em todos os locais em que viesse a ter um setup que utilizasse o MiniDAQ onde o laboratório ou instituição em questão não já tivesse suporte a boot pela rede, com a imagem do CCPC.

Visando a simplificação deste processo, criamos um método alternativo de boot para o MiniDAQ que visa a possibilidade de tornar qualquer máquina um servidor de boot, independente do sistema operacional utilizado, e com pouco esforço e necessidade de configuração. Foi criada uma máquina virtual Linux com todas as bibliotecas e o software necessário, com o sistema já configurado para que o CCPC possa fazer o boot através dela.

Durante o processo, da primeira vez em que foi obtido sucesso na configuração de uma máquina com Scientific Linux 6.2, para realizar o boot do CCPC, foram registradas todas as etapas e comandos realizados desde a instalação do sistema até o momento em que o CCPC realizou o boot com sucesso. Após isto, foi instalado o software VirtualBox da Oracle em uma máquina hospedeira Windows, e foi criada uma máquina virtual de 32 bits onde foi instalado o sistema operacional Scientific Linux 6.2. Todos os passos realizados para configuração da máquina física foram reproduzidos nesta nova máquina virtual, havendo necessidade apenas de configurar o gerenciador de máquinas virtuais VirtualBox para utilizar diretamente a placa de rede utilizada para comunicação com o CCPC. Este método é comumente chamado de conexão de ponte entre o hardware virtual e o físico. Uma vez testado e obtido sucesso em realizar o boot do CCPC através da máquina virtual, a mesma foi desligada e uma imagem desta máquina virtual foi criada e colocada à disposição de toda colaboração, sendo divulgada a todos os usuários cadastrados na página do projeto do MiniDAQ. Como resultado, para qualquer usuário que queira utilizar o MiniDAQ, basta fazer o download do software VirtualBox e importar a imagem criada da máquina virtual de boot. Independente do sistema utilizado (dentre os mais populares Windows, Mac e Linux), este agora pode conectar o MiniDAQ à interface de rede escolhida para o VirtualBox e realizar o boot do CCPC ao ligar o MiniDAQ, uma vez que toda a configuração necessária dos servidores de boot já foi definida antes da exportação da imagem da máquina virtual.

A máquina virtual de boot se encontra atualmente disponível no site do projeto do MiniDAQ e é utilizada por vários grupos da colaboração do LHCb. A lista completa de comandos e procedimentos realizados para configuração da máquina virtual podem ser encontrados no Apêndice C.

8.3 Validação e Depuração do MiniDAQ

O grupo do CBPF foi o primeiro grupo da colaboração do LHCb a utilizar o MiniDAQ implementando todo o seu caminho de dados. Para utilizá-lo em nosso sistema, era nescessário validá-lo a fim de demonstrar que este era adequado para o sistema de teste do detector de fibras cintilantes e que todas as etapas do caminho de dados proposto funcionavam como esperado. Por não haver outros grupos já com experiência na utilização do MiniDAQ, a configuração deste foi um aprendizado acompanhado de vários desafios e identificação de falhas na programação do CCPC, FPGA ou da máquina responsável por fazer o boot do CCPC, assim como a busca por suas respectivas soluções.

O primeiro teste, naturalmente, foi apenas alimentar o MiniDAQ com a tensão especificada e observar se este respondia. Juntamente também foi verificado se os LEDs do painel frontal e os coolers funcionavam, se havia luz nos transmissores ópticos e se a cadeia JTAG da AMC40, na qual faz parte o FPGA Stratix V, era identificável a partir de um USB Blaster. Foi verificado que o MiniDAQ não apresentava nenhum sinal de mau funcionamento e então se iniciou a realização de testes mais específicos.

8.3.1 Teste da Camada Física

O propósito deste teste é a verificação do FPGA e a interface de configuração deste, e a validação da camada física por onde passam os dados. Esta compreende os transceptores ópticos (MiniPods), as conexões e cabeamento, e os transceptores integrados do FPGA.

Através de um computador executando o software Altera Quartus II 13.0 e um USB Blaster conectado à USB, podemos verificar e fazer o upload de novos firmwares através do conector JTAG provido no painel frontal. Com isto foi possível verificar que o FPGA estava devidamente acessível.

Foi utilizado um firmware desenvolvido pelo grupo de Marselha onde havia uma instância do bloco de depuração do Altera Transceiver Toolkit, que é uma ferramenta provida no software Quartus II 13.0, para tornar acessível a configuração dos transceptores do Stratix V durante tempo de execução, com propósito de depuração. O Transceiver Toolkit é uma ferramenta da Altera com funções integradas que agem diretamente nos transceptores do FPGA, podendo configurá-los para enviar diversos tipos de sequências binárias, algumas pseudo-aleatória, e verificar se estas estão sendo recebidas corretamente.

Além disto, este fornece dados estatísticos e gráficos como o diagrama de olho do sinal, conforme visto na Figura 8.7.

Os transmissores então foram ligados aos receptores numa configuração de loopback e os transmissores foram configurados para transmitirem uma certa sequencia aleatória, a partir da qual foi possível validar toda a cadeia de transmissão observando se os dados que chegavam nos receptores correspondentes estavam de acordo com os dados enviados. Isto foi realizado com sucesso.

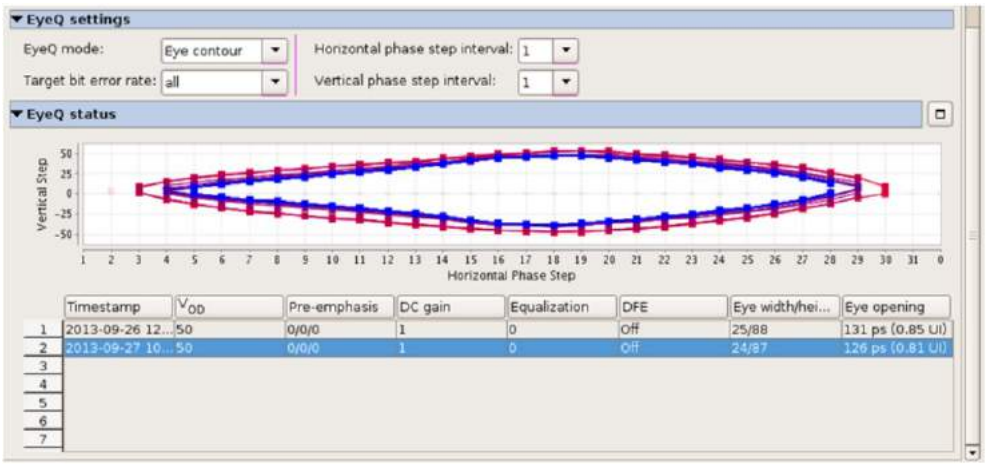

Figura 8.7: Interface do Transceiver Toolkit com o gráfico de olho sendo formado.

Este teste tinha como objetivo apenas atestar o bom funcionamento do equipamento utilizado a fim de excluir possíveis causas de erros que viessem a ocorrer nas próximas etapas. Após verificar que a taxa de erro e o gráfico de olho não apontavam nenhum indício de interrupção na cadeia de transmissão dos dados, foi possível concluir que a camada física estava funcionando corretamente e seguimos para a próxima etapa.

8.3.2 Configuração de Software do CCPC

Uma vez tendo o CCPC realizado o boot corretamente e estando disponível através de acesso SSH, a próxima etapa foi testar os softwares fornecidos pelos desenvolvedores, responsáveis pela interface entre o CCPC e o FPGA Stratix V encontrado na AMC40. Através desta interface seriam transferidos todos os comandos de Slow Control para o FPGA que, por sua vez, seriam encaminhados para os devidos dispositivos na eletrônica de *front-end*.

Este teste consistia em seguir as instruções disponibilizadas no MiniDAQ Handbook [19] para instalar os softwares desenvolvidos para o MiniDAQ.

O pacote de softwares é disponibilizado em um repositório GIT online o qual se tem acesso com as credenciais de uma conta online do CERN. A menos que o CCPC esteja instalado em uma rede pré-configurada para suporte a boot PXE, o que não foi o caso durante a configuração do MiniDAQ do sistema de testes, o usuário não terá acesso à internet a partir do CCPC. Por esta razão se faz nescessário realizar a clonagem do repositório GIT a partir do computador de boot do CCPC, seja este físico ou uma máquina virtual, para então transferir os softwares necessários para serem instalados no CCPC. Como alternativa também é possível adicionar os pacotes diretamente na imagem do sistema de arquivos que o CCPC utiliza, a partir da máquina onde o servidor de boot é executado. Após copiá-los, as instruções de instalação se encontram na seção 5 do MiniDAQ Handbook [19].

Ao clonar o repositório de software, o usuário deve se atentar à diferença de terminações de linha dos sistemas operacionais Windows, DOS e Linux. Enquanto Windows e DOS utilizam os caracteres "\r\n" (carriage return e line feed) como terminação de linha, sistemas baseados em Unix utilizam apenas "\n" (line feed). Caso o repositório seja clonado em Windows ou DOS, este deve ter seus arquivos fonte convertidos para o sistema de terminação do Unix, caso contrário haverá erro na compilação e o mesmo não é mostrado claramente, uma vez que é apenas mostrado a linha do código onde ocorreu o erro mas o caractere "\r" que está causando o erro é invisível. Na primeira tentativa de instação do CCPC, como nenhum outro grupo havia feito desta forma anteriormente e os desenvolvedores sempre utilizaram Linux, perdeu-se algumas horas de depuração até se entender que o problema era este. O problema foi reportado e uma nota foi adicionada às instruções contidas no MiniDAQ Handbook [19], recomendando o usuário a clonar o repositório a partir de uma máquina Linux.

Para se clonar o repositório e copiar o mesmo para o CCPC a partir da máquina virtual de boot, devem-se utilizar os seguintes comandos:

```
$ export GIT_SSL_NO_VERIFY=true
$ git clone https://usuario_cern@git.cern.ch/reps/amc40lli
$ scp -r amc40lli upgrade@192.168.1.51
```

O segundo erro encontrado foi após ter sido realizada a operação descrita no parágrafo

anterior durante o setup do MiniDAQ. Ao seguir as instruções do MiniDAQ Handbook [19], durante a instalação dos softwares, várias funções não foram encontradas devido a ausência de pacotes. Os desenvolvedores do MiniDAQ, ao criarem o pacote de software, dispunham de um sistema já com diversos pacotes instalados. Por esta razão não notaram que possivelmente alguns dos pacotes de seu ambiente de desenvolvimento não estariam disponíveis na imagem de sistema fornecida para o usuário final. Por esta razão, foram listadas as dependências encontradas para a instalação dos softwares do CCPC e foi comunicado aos desenvolvedores.

Os usuários que ainda encontrarem dependências ausentes para a instalação dos softwares do CCPC devem utilizar os seguintes comandos para download dos pacotes e envio ao CCPC:

```
$ yum install yum-utils
$ yumdownloader <pacotes>
$ scp <pacotes> upgrade@192.168.1.51
```

E após o envio, executa-se o seguinte no CCPC para instalação dos pacotes:

```
$ rpm -i <pacote>
```

Com todos os pacotes necessários instalados no CCPC, deve-se seguir as instruções contidas na seção 5.1.5.1 do MiniDAQ Handbook [19], "Drivers and libraries", para correta configuração de software do CCPC.

8.3.3 Interface entre o CCPC e o FPGA da AMC40

Após realizar corretamente a instalação do pacote de softwares do CCPC, a próxima etapa foi o teste da interface entre o CCPC e o FPGA contido na AMC40.

O teste consiste em tentar ler e escrever em registradores que são sintetizados a partir do firmware do FPGA na AMC40. Todo o controle do firmware da AMC40 é realizado por acesso a registradores, através de um barramento de dados implementado entre o CCPC e a AMC40. Por esta razão, se for possível fazer o upload de um firmware com

alguns registradores instanciados e, através dos softwares do CCPC, ler e escrever neste registrador, significa que a interface necessária para controle está funcionando corretamente no que diz respeito à camada de hardware, independente do firmware a ser utilizado posteriormente. Tendo sucesso neste teste, limitaríamos as futuras possíveis causas de erro no sistema de controle a software e firmware.

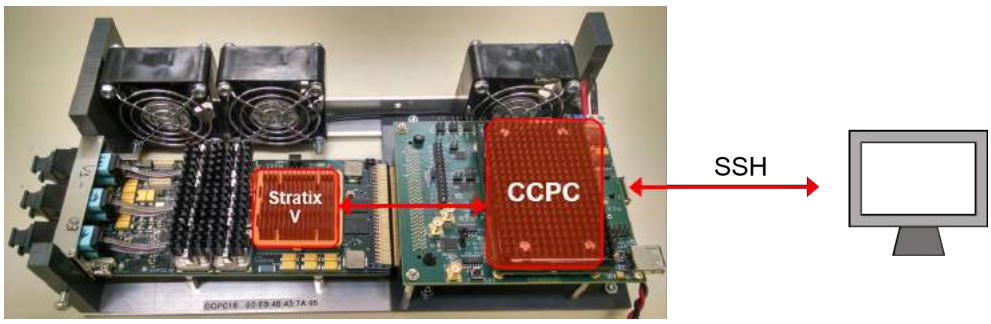

Figura 8.8: Teste de comunicação entre o CCPC e o FPGA da AMC40.

Antes de executar qualquer software que se comunique com a AMC40 é nescessário reescanear o barramento PCIe e carregar o driver. Existe um script para isto já preparado, no pacote de software do CCPC.

Dentro do pacote de software do CCPC, no diretório driver/, executa-se o comando:

```
$ sudo ./start.sh
```

O script também retorna os bancos de registradores detectados e informações sobre eles, funcionando assim como uma maneira de diagnóstico. Após este passo, os softwares de controle podem ser executados. A cada nova configuração do FPGA é necessário executar este script novamente.

Para acessar um registrador arbitrário foi utilizado o software test-bar0. Este se encontra no diretório "user/user_test/bin/" do pacote de software do CCPC. A utilização deste se dá através de comando de linha, com um comando para escrita onde se especifica o endereço do registrador e o valor a escrever, e um comando de leitura.

Os comando de escrita e leitura em registradores instanciados no firmware do FPGA são, respectivamente:

```
$ ./test-bar0 --val <valor> --write-at <endereco>
$ ./test-bar0 --read-at <endereco>
```

Para verificar se o acesso aos registradores estava se dando de maneira correta, foi efetuada a leitura do valor de um registrador e comparado ao seu valor padrão declarado no código fonte do firmware e, após um ciclo de escrita de um valor arbitrário, foi comparado novamente para verificar se o valor do registrador havia sido atualizado.

8.3.4 Loopback

Uma vez tendo acesso de leitura e escrita aos registradores do firmware no FPGA da AMC40, podemos testar toda a cadeia de dados que será utilizada na aplicação final do sistema de testes do SciFi. Neste a AMC40 será controlada pelo CCPC. Os comandos para a AMC40 chegam ao CCPC através de sua interface Ethernet, a mesma utilizada para a conexão SSH do teste. Os comandos de controle destinados à *front-end* são enviados pela AMC40 através do link óptico GBT de saída da AMC40 e os dados provenientes da *front-end* chegarão à AMC40 através dos links ópticos GBT de entrada. Após processados, estes serão enviados através do link óptico Ethernet de 10 Gbps.

O sistema original usará os links ópticos de Ethernet 10 Gbps e GBT bidirecionais, assim como a interface Ethernet 1 Gbps do CCPC. O objetivo do teste de Loopback é justamente testar todas as interfaces e todo o caminho de dados que será utilizado na aplicação do MiniDAQ no sistema de testes do SciFi. Este teste pode ser dividido em 4 etapas, que podem ser visualizadas na Figura 8.9. Estas etapas são descritas a seguir:

1. Esta etapa se caracteriza pela injeção de dados numa memória na AMC40, cujo firmware utilizará para enviá-los pela interface óptica. O primeiro passo é o carregamento do firmware padrão fornecido pelos desenvolvedores do MiniDAQ. Após isto, se conecta ao CCPC via SSH e carrega-se os drivers da mesma forma realizada na seção 8.3.3. Deve-se gerar um arquivo de dados, que podem ser genéricos e arbitrários, para ser injetado na memória do FPGA.

   Tendo pronto o arquivo de dados, que chamaremos aqui de “dados.txt”, utiliza-se o programa “injection” do pacote de software do CCPC, localizado no diretório

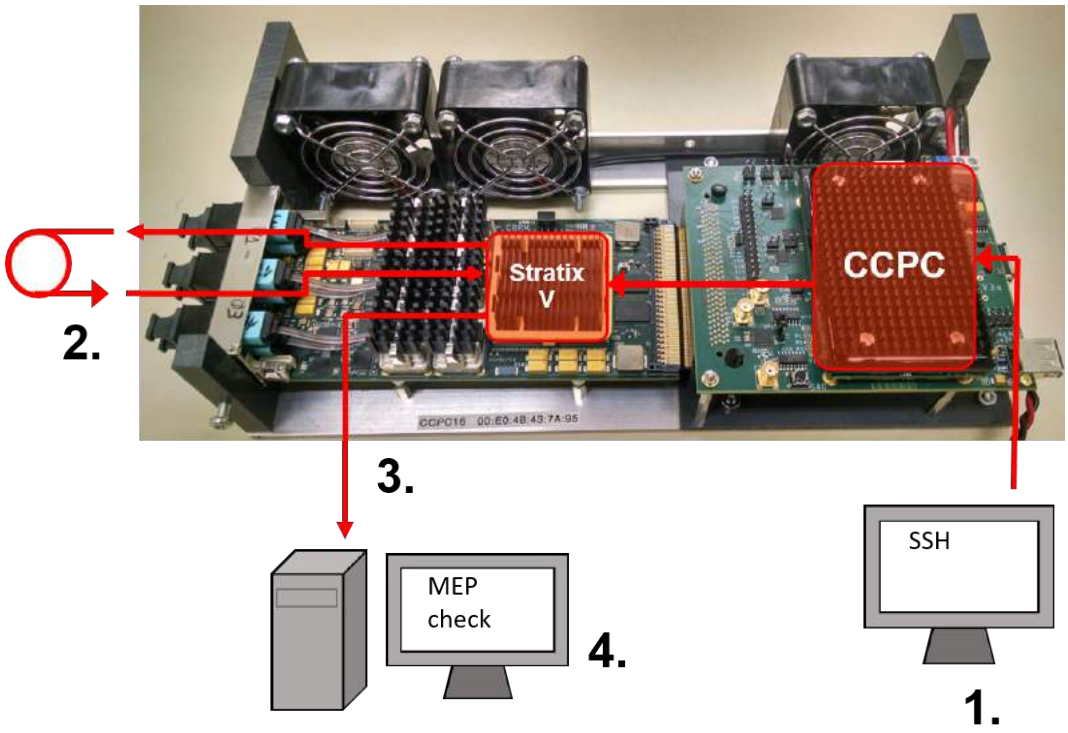

Figura 8.9: Teste de envio e aquisição de dados em loopback.

"lli/exec/". Este procedimento é descrito na seção 5.1.5.9 do MiniDAQ Handbook [19] e os comandos necessários para a injeção dos dados são:

```
$ ./injection --load-file dados.txt
$ ./injection --ram-size 16777215
$ ./injection --test-size 1
$ ./injection --start-memory
$ ./injection --stop-memory
```

Onde o parâmetro "–ram-size" recebe o tamanho da memória de onde serão enviados os dados. Neste exemplo foi utilizado o valor total da memória, de 16777215 bytes. O parâmetro "–test-size" se refere ao número de iterações do teste de envio. O valor 1 significa que só será enviado uma única vez o conteúdo da memória. O comando "–start-memory" diz ao firmware do FPGA para iniciar o envio dos dados da memória através do link óptico GBT. O comando "–stop-memory" encerra o envio. Deve-se

sempre encerrar um envio antes de iniciar novamente o procedimento, mesmo que este já tenha de fato terminado as iterações de envio de dados, o comando deve ser executado de qualquer forma.

2. Após os dados serem enviados pelos links ópticos GBT de saída da AMC40, estes retornarão aos links ópticos GBT de entrada, devido à conexão em configuração de loopback dos cabos de fibra óptica ligados aos conectores ópticos. No sistema real, os dados que chegarão na AMC40 serão os dados enviados da eletrônica de *front-end*. E não haverá envio de dados de experimento nos links de saída da AMC40, mas ao invés estes serão comandos de controle para os diversos componentes da frond-end.

3. Uma vez que os dados cheguem novamente no firmware, eles serão processados conforme explicado na seção 8.1.1 e enviados através da interface óptica de 10 Gbps Ethernet para o computador de aquisição. Para este teste, no computador de aquisição estava sendo utilizado o software Wireshark, de monitoramento de pacotes de rede, para registrar todos os pacotes ethernet que chegassem à interface de rede de 10 Gbps.

   Através do Wireshark é possível exportar um arquivo de texto contendo a informação já processada que chegou na interface de rede.

4. A quarta e última etapa do teste do caminho de dados é a de comparar os dados enviados pelo firmware com os dados que chegaram no computador de aquisição. Há um software desenvolvido por Paolo Durante, chamado de amc40_capchecker, e que faz parte do pacote de softwares do MiniDAQ. Este software compara automaticamente e gera um relatório de erros de quais as seções veio o erro.

   Nos primeiros testes realizados, foi obtida uma sequência de erros no software que, após investigação, pudemos concluir que era ocasionada pelo próprio software. Uma versão temporária foi desenvolvida para uso pessoal e singular, somente com o objetivo de não se perder mais tempo e permitir dar continuidade aos testes.

   Os erros encontrados foram também reportados ao responsável pelo desenvolvimento do software e na presente data este já se encontrava atualizado no repositório do MiniDAQ com as falhas corrigidas.

Tendo completado estes testes, foi possível verificar que todo o caminho de dados necessário para o sistema de testes estava funcional, querendo dizer que qualquer futuro

problema que viesse a ocorrer provavelmente seria por questão de implementação, e não causado por falha de hardware.

8.3.5 Firmware de Raw Data

O Firmware de Raw Data é uma versão modificada a partir do firmware original do MiniDAQ com o objetivo de propiciar ao usuário acesso e controle dos dados que chegam e são enviados pelos transceptores do FPGA da AMC40, antes destes serem codificados ou decodificados pelos blocos que implementam o protocolo GBT no firmware do MiniDAQ.

O Firmware foi desenvolvido devido à necessidade de depuração dos elementos do caminho de dados utilizado nos testes mencionados na seção 8.3.4. Em algumas situações, os dados não chegavam ao computador de aquisição, no fim do caminho de dados, em conformidade com os dados gerados e enviados. Pela análise dos dados que chegavam, não era possível identificar em qual etapa ocorreu o erro e também não era possível identificar um padrão no sinal enviado (para utilizar como trigger em um analizador de sinal), uma vez que o codificador do protocolo GBT embaralha os dados. Tendo uma forma de injetar os dados diretamente no transmissor, sem codificar os dados no protocolo GBT, e receber estes também diretamente do receptor, antes de passar pela decodificação, torna possível inferir se o erro ocorre na camada física do caminho de dados, nas etapas de codificação e decodificação, ou na etapa de processamento dos mesmos, que ocorre após a decodificação.

A modificação realizada no firmware consiste em um bloco, chamado de Diagnosis, que foi adicionado em paralelo aos demais ligados no barramento ECS vindo do CCPC. Foi dedicado um intervalo de endereços não utilizado anteriormente para os registradores implementados no bloco Diagnosis, e dele saem sinais de controle para o restante do firmware.

Imediatamente antes do transmissor do FPGA foi adicionado um multiplexador, controlado por um sinal proveniente do bloco Diagnosis. Com este sinal, é possível escolher a fonte de dados do transmissor entre os dados codificados do GBT do firmware padrão ou um barramento direto do bloco Diagnosis. Desta forma é possível injetar dados “crus” no transmissor.

Também imediatamente após a chegada dos dados no receptor, foi feita uma extensão do barramento até o bloco Diagnosis, de forma que este tenha acesso aos dados “crus” que estão chegando no transmissor.

Além do desvio dos dados, outros sinais de controle também foram extendidos ao bloco Diagnosis. O sinal "header_locked", responsável por dizer se o decodificador do GBT foi capaz de travar em um cabeçalho da palavra GBT, e o sinal "Rx_Data_Valid", responsável por informar quando as palavras que chegam no receptor são dados válidos. Antes da adição do bloco Diagnosis, era necessário utilizar um firmware com uma ferramenta do tipo SignalTap adicionada, a fim de verificar o valor dos sinais pela interface JTAG, exigindo hardware adicional e uma camada a mais de complexidade.

Os sinais de saída do bloco Diagnosis, que são sinais de monitoramento, são conectados a uma memória FIFO, cuja leitura é acionada por um acesso via ECS a um endereço específico, de modo que os dados podem ser lidos através do barramento ECS, por software executado no CCPC.

Os sinais de entrada, por sua vez, estão conectados a registradores que também possuem endereço reservado e podem ser escritos através do barramento ECS. Desta forma tudo era configurado por software no CCPC.

O firmware RawData trouxe ao usuário a flexibilidade de escrever dados diretamente no transmissor, e isto permitiu realizar vários testes, como por exemplo escrever um cabeçalho arbitrário, ou verificar a ordem de chegada dos bits antes destes serem embaralhados no decodificador GBT, e ainda permitiu a verificação de sinais internos do decodificador GBT que antes não estavam disponíveis ao usuário. Isto tornou este firmware peça fundamental durante a depuração do MiniDAQ.

8.3.6 Firmware GBT simplificado em Stratix IV

Além da versão RawData do firmware do MiniDAQ, uma outra ferramenta foi criada para depuração durante a validação do MiniDAQ. É uma versão simplificada do firmware do GBT, na qual foi implementada em um módulo de desenvolvimento de FPGA Stratix IV, a fim de criar uma versão de rápida compilação do GBT para ser utilizada durante os testes com o GBT real.

O tempo médio de compilação do firmware padrão do MiniDAQ, em um computador de escritório padrão do CERN, era de aproximadamente 1 hora e 30 minutos. Esta demora tornava inviável pequenos testes durante a validação do MiniDAQ como por exemplo trocar a ordem dos bits do cabeçalho. Embora seja apenas uma linha no código VHDL, todo o projeto era compilado novamente e perdia-se horas no total para fazer poucas alterações

em sequência.

Fez-se então uma versão simplificada de um firmware do GBT adaptado de uma das primeiras versões oficiais do GBT-FPGA, a ser sintetizado em um módulo de desenvolvimento de FPGA Stratix IV GX. Por ser formado apenas pelo GBT e uma pequena lógica de controle, o tempo de compilação deste firmware para o Stratix IV era de aproximadamente 4 minutos.

8.3.7 Testando o MiniDAQ com um chip GBT Real

Em um certo ponto durante o trabalho de teste e depuração do MiniDAQ, este já não apresentava nenhum problema na configuração de loopback. No entanto, o fato de um sistema de aquisição de dados funcionar ao adquirir dados enviados pelo próprio sistema é o mínimo que se espera de um sistema de aquisição. Isto não é indicativo de que o sistema funcionará na aquisição de dados reais provenientes de outros sistemas. Por isto foram realizados testes com um chip GBT real, o mesmo que será utilizado na eletrônica de *front-end* que enviará dados ao MiniDAQ no sistema de testes do SciFi.

O departamento de eletrônica do LHCb cedeu um módulo de teste do GBT chamado "GBTx SAT V1" (Placa superior da Figura 8.10) contendo um chip GBT, que poderia ser utilizado para verificar se os dados que chegavam eram decodificados ou se o GBT real e o GBT emulado no MiniDAQ ao menos faziam um "aperto de mãos", ou sejam, se travavam o decodificador no cabeçalho um do outro.

A placa utilizada no teste continha LEDs indicativos dos sinais de controle do GBT, como por exemplo sinal de travado no cabeçalho, sinal de dados válidos, etc. A placa também continha um FPGA emulando um chip GBT, com o qual através de um bloco de Signal Tap se podia monitorar os sinais de controle. Este FPGA se comunicava corretamente com o GBT real e por isso era uma ferramenta confiável de depuração.

O primeiro teste foi realizado conforme indicado na Figura 8.10. A entrada e saída óptica do GBT emulado no MiniDAQ foi ligada à interface óptica conectada ao GBT real. Os LEDs da placa não indicaram trava no cabeçalho. Esta era a primeira etapa necessária a ser realizada para uma comunicação bem sucedida. Sem travar ao cabeçalho as etapas restantes não funcionam.

Os sinais foram analizados também utilizando o SignalTAP e o FPGA da placa de teste do GBT, mas também não apresentavam pistas do motivo pelo qual ambos os GBT

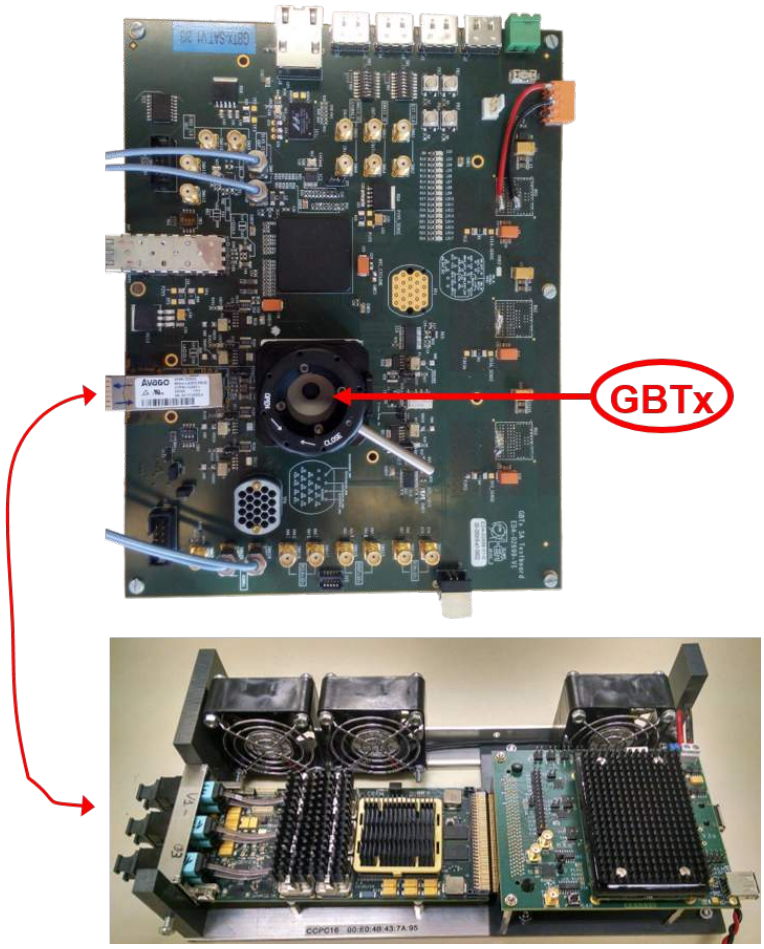

Figura 8.10: Teste de comunicação entre o MiniDAQ e o chip GBT.

não se comunicavam. Como alternativa, utilizamos um analizador de sinais digitais, porém devido ao sinal embaralhado pelo codificador GBT, se tornou impossível travar o analizador de sinais a algum padrão periódico. Neste ponto ainda não existia a versão RawData do firmware e por isto não foi possível emitir um sinal periódico através da interface óptica do MiniDAQ a fim de se analizar no instrumento de medição. A conclusão preliminar foi que simplesmente não funcionava, por razões ainda a investigar.

O segundo teste não foi realizado com um GBT real, mas ao invés um MiniDAQ com um firmware que já havia se comunicado com sucesso com um GBT real. Foram dois MiniDAQs com firmwares diferentes, conforme mostrado na Figura 8.11, sendo um deles funcional com o GBT real. Os GBTs emulados no firmware padrão do MiniDAQ eram baseados em versões antigas do projeto GBT-FPGA, que visa criar a versão oficial da emulação do GBT para as colaborações do CERN. A última versão do GBT-FPGA já havia

sido testada no MiniDAQ com a placa do teste anterior e havia funcionado corretamente. A diferença era que o firmware utilizado no MiniDAQ era um firmware que continha apenas a nova versão do GBT-FPGA. Não havia em paralelo todos os blocos da LLI que há no firmware padrão do MiniDAQ.

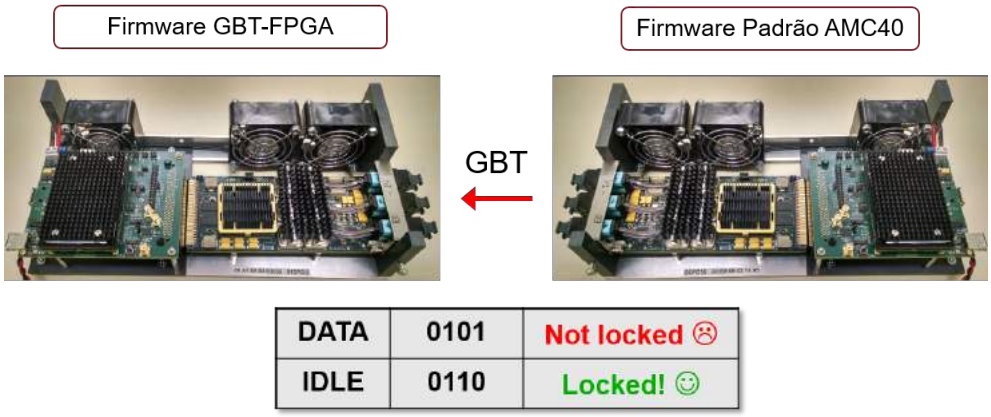

Figura 8.11: Teste de comunicação entre os GBTs emulados do firmware padrão do MiniDAQ e do firmware de teste do projeto GBT-FPGA.

Os dois MiniDAQs foram postos sob o mesmo sinal de clock e suas conexões ópticas foram interligadas. Foi observado que o MiniDAQ com o firmware original do projeto GBT-FPGA travava no cabeçalho sempre que o MiniDAQ com o firmware padrão enviava o cabeçalho de ocioso. Quando o firmware era comandado a enviar dados válidos, o MiniDAQ com o firmware GBT-FPGA perdia a trava no cabeçalho.

Estes resultados levaram à hipótese de que os bits enviados pelo MiniDAQ com o firmware padrão poderiam estar sendo enviados de forma invertida. Isto porque o cabeçalho referente ao estado ocioso ("0110") é simétrico enquanto o cabeçalho referende a dados válidos ("1010") não é, ou seja, mesmo com a ordem dos bits invertida, o cabeçalho de ocioso continuaria funcionando.

O próximo passo foi então retornar aos testes com o GBT real, mas desta vez com novas ferramentas que eram o firmware de Raw Data para o MiniDAQ e o kit de desenvolvimento Stratix IV com um GBT simplificado.

Para testar a hipótese da ordem de bits estar invertida, utilizou-se o firmware de Raw Data, injetando a cada palavra GBT enviada, uma sequência de bytes 0xF0, sendo os ultimos 4 bits o número 5 (cabeçalho de dados) ou número 6 (cabeçalho de ocioso). A princípio não houve trava em cabeçalho de nenhuma das parte. Tanto o GBT real quanto

o MiniDAQ não travava no cabeçalho da outra parte.

Ao realizar o mesmo teste, mas utilizando o kit do Stratix IV com o firmware do GBT simplificado ao invés do MiniDAQ, tivemos um resultado positivo. O GBT real travava com o cabeçalho enviado pelo Stratix IV sempre que invertíamos a ordem de bits do cabeçalho. Neste momento houve a confirmação de que os bits estavam invertidos, mas ainda não era claro o motivo pelo qual o MiniDAQ não funcionava com o GBT.

O GBT implementado no firmware padrão do MiniDAQ tinha como base o mesmo da versão simplificada do GBT no Stratix IV. Não se viam motivos para um firmware conseguir se comunicar com o GBT real e o outro não. Para investigar o porquê do ocorrido, o próximo passo foi realizar um teste que consistia em enviar a partir do Stratix IV a mesma palavra que ocasionava a trava de cabeçalho no GBT real, mas ao MiniDAQ. E com o firmware de Raw Data, monitorar os sinais de controle e a palavra que chegava no receptor. O resultado foi que o MiniDAQ travava no cabeçalho enviado pelo Stratix IV.

Se ambos os firmwares eram compatíveis e o hardware de ambas as placas já haviam sido testados e estavam funcionais, a conclusão foi que a causa mais provável era se tratar de um erro de configuração do setup.

Em teoria, dois GBTs não precisam de alinhamento dos sinais para se falar entre si. Pode haver uma defasagem entre os sinais de dados e os clocks do sistema. O receptor identifica a fase do sinal de entrada e alinha o clock com o sinal recebido. No entanto, se dois GBTs estiverem funcionando em clocks de frequências diferentes, estes não serão sincronizados um com o outro. O que ocorreu nesta sequência de teste é que, a placa GBTx SAT V1 utilizava como fonte de clock um sinal proveniente de oscilador integrado de 40,000 MHz enquanto o MiniDAQ utiliza uma fonte interna de clock de 40,079 MHz. A placa do kit de FPGA Stratix IV possui um oscilador configurável que no momento provia 120,000 MHz, e que era divisível internamente por PLLs instanciadas no firmware. Por isso os transmissores do GBT real e do kit do Stratix IV estavam sob a mesma frequência, e por isso travavam um com o outro. Ao testar a comunicação entre o MiniDAQ e o kit Stratix IV, utilizamos como fonte de clock para o MiniDAQ o clock proveniente do kit Stratix IV. Por estarem sob a mesma frequência, os dois GBTs emulados travaram no cabeçalho um do outro.

Apesar do receptor GBT emulado no MiniDAQ suceder em travar no cabeçalho enviado pelo GBT real quando sob o mesmo clock e considerando o cabeçalho invertido, ainda assim, o restante da palavra do GBT parecia aleatório e sem significado. Poderia

ser apenas devido ao fato da ordem dos bits estar invertida, mas também poderia ser por outros problemas associados à etapa de decodificação da palavra GBT. Visando previnir problemas futuros que possam vir a requerer mais tempo de depuração, foi determinado que o firmware padrão do MiniDAQ deveria passar a utilizar a versão oficial do GBT ao invés de apenas corrigir a versão utilizada.

Conclusões dos testes com o chip GBT real:

1. A ordem dos bits do cabeçalho na versão do GBT utilizada no MiniDAQ estava invertida, e quiçá o restante da palavra estaria também.
2. O MiniDAQ e o GBT precisam utilizar a mesma frequência de clock para se comunicarem.
3. Foi estabelecido que as implementações do GBT em FPGA deveriam seguir o projeto GBT-FPGA oficial pra evitar mais erros na decodificação de dados.
4. As ferramentas criadas para auxiliar na depuração do MiniDAQ, o firmware simplificado do GBT na Stratix IV e a versão Raw Data do firmware do MiniDAQ, se mostraram de suma importância na etapa de depuração e foram vitais na determinação da causa dos erros que ocorriam e a proposta das soluções acima.

Capítulo 9

Desenvolvimento do Sistema de Controle da FEE do SciFi

Esta seção aborda o desenvolvimento de um sistema básico de controle para a eletrônica de *front-end* do SciFi. O objetivo primário do trabalho descrito nesta seção era a preparação de um sistema de controle para ser utilizado no sistema de teste do detector SciFi. No entanto, até o momento não havia ainda um sistema de controle para a eletrônica de *front-end* do SciFi. Por esta razão este trabalho se tornou também o início do sistema de controle a ser utilizado no próprio experimento.

Assim como ocorreu na configuração do MiniDAQ, o grupo do CBPF também foi o primeiro usuário a escrever funções de controle para o MiniDAQ e, por esta razão, não foi um trabalho apenas de desenvolvimento do sistema de controle, mas também de teste e depuração das ferramentas necessárias para implementação do sistema de controle. A exemplo destas ferramentas podemos citar a camada de software do MiniDAQ responsável pela interface entre o projeto do WinCC e a AMC40, e o bloco de firmware responsável pelo controle, o SOL40, que interpretava e traduzia comandos recebidos pelo barramento ECS, vindos do CCPC, e enviava à *front-end* através do link GBT.

O software escolhido para desenvolvimento do sistema de controle do sistema de teste do SciFi foi o WinCC, conhecido anteriormente como PVSS. Este se trata de um sistema SCADA (Supervisory Control and Data Acquisition) fornecido pela Siemens e que já é um sistema amplamente utilizado no CERN, sendo inclusive o responsável pelo controle do LHC e seus quatro experimentos.

O WinCC por si só não é um sistema de controle, mas sim um software para criação de sistemas de controle. O WinCC é usado para se conectar a dispositivos de hardware do nossos sistema, adquirir dados e utilizá-los para supervisão, ou seja, monitorar o comportamento dos dispositivos do sistema e inicializá-los, configurá-los e operá-los. O principal motivo da escolha deste software foi por já existirem drivers adequados para o CCPC, facilidade de obter suporte no CERN, uma vez que é o sistema utilizado no LHC, e já haver licença disponível para uso no CERN.

9.1 Sistema de Controle

Um sistema de controle é um sistema que permite ao operador monitorar e atuar sobre parâmetros de diversos dispositivos e subsistemas. Estes parâmetros podem significar a leitura de sensores, entrada de atuadores ou variáveis do sistema.

No contexto da eletrônica do sistema de testes do SciFi, quase a totalidade dos parâmetros de controle do sistema são armazenados em diferentes registradores espalhados pelos dispositivos que compõem o sistema. Os valores que não se encontram armazenados em registradores, ainda assim são escritos ou lidos pela mesma interface de acesso aos registradores. Por esta razão, controlando o acesso aos registradores, pode-se atuar e monitorar todos os parâmetros do sistema de controle.

O sistema de controle do SciFi deve fornecer ao usuário meios de ler e escrever nos diversos registradores que compõem o sistema e pelos quais se pode controlar e monitorar o sistema. O sistema também deve fornecer uma estrutura para tratar e armazenas os dados que são lidos ou escritos nos registradores do sistema, de modo a se poder acessar estes dados de forma simplificada e distribuída, e também manter um histórico dos parâmetros para referência futura e demais funções de controle.

O sistema também deve fornecer meios de configurar alertas e ações a serem engatilhadas por certos eventos. Em outras palavras, deve fornecer uma forma de se criar uma lógica que, baseada nos valores dos parâmetros do sistema, possa tomar atitudes tanto autônomas de controle como de alerta ao operador.

No topo de todos estes itens, o sistema também deve prover uma interface homem máquina para que o operador possa, com facilidade, monitorar visualmente o que se passa no sistema. Esta interface é normalmente telas em monitores que exibem in-

formações importantes do sistema e que possuem controles com os quais se pode interagir a fim de modificar parâmetros do sistema. Usualmente os alertas gerados pela lógica de monitoramente são imagens com cores chamativas que aparecem na tela e sons emitidos por auto-falantes. É também através da interface de usuário que o operador pode acessar o histórico de dados e configurar o que for necessário no sistema.

9.2 Estrutura básica de um sistema WinCC

A arquitetura de um sistema WinCC é altamente distribuída e modularizada. Um sistema WinCC é composto por diversos processos, chamados de "Managers" na nomenclatura WinCC, e de estruturas de dados chamadas de datapoints.

De forma simplificada, os datapoints descrevem cada parâmetro do sistema: os parâmetos de hardware, os registradores, parâmetros de sensores e atuadores, etc. E os managers são os processos que lidam com estes datapoints, atualizando-os para condizer com os valores reais, imprimindo-os na tela, criando históricos e tomando atitudes de acordo com algoritmos baseados nos datapoints.

Datapoint é um tipo de estrutura de dado customizável e hierárquica que representa um parâmetro ou um conjunto de parâmetros do sistema de controle. Estes datapoints podem representar valores diretamente em registradores do seu sistema ou ser uma variação disso. Eles podem também agir como simples variáveis do seu sistema.

Cada datapoint tem um valor e um tipo atribuído a ele. Existem tipos predefinidos, como valores numéricos (inteiro, real, etc.), strings alfanuméricas, vetores e datapoints que podem aninhar outros datapoints hierarquicamente, permitindo assim criar estruturas complexas.

Os datapoints serão a descrição de todos os parâmetros do seu sistema de controle. Para cada hardware utilizado, por exemplo, deve-se ter um datapoint associado contendo outros datapoints representando os dispositivos deste hardware e seus parâmetros (que poderiam ser valores de registradores, por exemplo).

Um sistema típico WinCC é composto pelos seguintes managers, que podem ser vistos na Figura 9.1:

- **Event Manager (EVM):** É o coração do sistema e um Manager essencial, ou seja, é

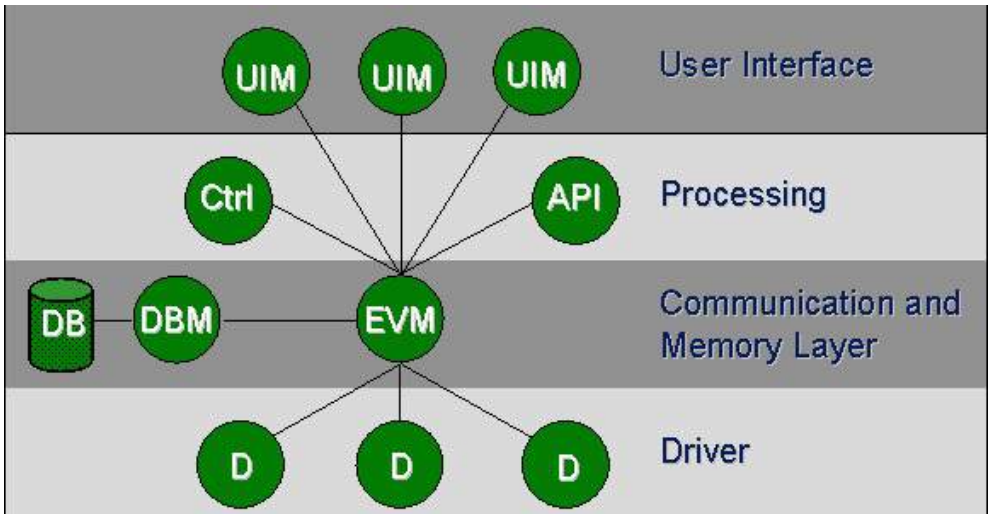

Figura 9.1: Estrutura de managers de um projeto em WinCC.

necessário que esteja em execução para funcionamento do sistema. Este é responsável por toda a comunicação entre os diferentes managers e atua como um gerente dos demais. Ele recebe dados dos drivers (D) e armazena no banco de dados (DB). Outros managers podem solicitar dados ao Event Manager, que busca no banco de dados ou solicita novos aos Drivers. Outros managers não se comunicam entre si, ou seja, toda comunicação entre managers é feita através do Event Manager.

- **Database Manager (DBM):** É o processo responsável por gerenciar o banco de dados onde fica armazenado todos os datapoints, e que fornece a interface para que outros managers possam acessá-lo.

- **User Interface Manager (UIM):** São processos criados que fornecem uma interface gráfica para o usuário, no formato de painéis gráficos, que podem ser criados utilizando o próprio WinCC. Através destes painéis o usuário pode monitorar visualmente e atuar nos diferentes parâmetros do sistema. O WinCC permite criar controles gráficos que são associados a datapoints, a fim de ter uma representação gráfica de um parâmetro. O nível de um tanque de combustível por exemplo pode ser representado por uma figura de um tanque cujo nível impresso varia de acordo com o valor do datapoint. E uma válvula por exemplo pode ser controlada a partir de um click na interface gráfica, esta por sua vez pode também estar associada à um datapoint que representa o sinal de controle de atuação da válvula real no sistema.

- **Drivers (D):** Os drivers são processos que fornecem a interface entre o sistema e

os dispositivos a serem controlados. Estes são específicos para cada dispositivo e precisam ser fornecidos pelo fabricante do aparelho ou criado de forma personalizada para cada hardware do sistema. São os drivers que de fato lêem e escrevem nos registradores do sistema, e atualizam os datapoints do sistema para condizerem com a realidade.

O Database Manager (DBM) junto com o Event Manager (EVM) são os requisitos mínimos para um sistema WinCC poder operar. Sem estes dois processos sendo executados, nenhum outro processo funcionará.

Os managers são todos processos independentes que se comunicam por protocolo TCP/IP. Mesmo quando sendo executados na mesma máquina a comunicação entre quaisquer dois managers é realizada via protocolo TCP/IP. Isto permite a distribuição do sistema por diferentes computadores. Desta forma podemos usar os painés de controle (Manager UIM) de qualquer computador com acesso à rede onde se encontra o Event Manager. E ainda os Drivers que fazem a interface com os dispositivos a serem controlados não precisam estar no mesmo computador onde está o Event Manager do sistema, permitindo o uso de computadores dedicados ao controle e monitoramento, e outros dispositivos a parte dedicados à interface com os sensores e atuadores do sistema. Isto também permite ter um banco de dados (DBM) distribuído, podendo ter dois ou mais bancos de dados em redundância em máquinas diferentes.

No sistema de controle do SciFi utilizaremos dois managers customizados: O Driver, pois por utilizarmos um módulo de fabricação própria da colaboração, o MiniDAQ, foi necessário criação também do componente driver, e os painéis com a interface gráfica de usuário.

Para o sistema de controle foi criado um novo projeto WinCC. Os Event Manager e Database Manager usados são os gerados por padrão na criação do projeto. Estes Managers, ou seja, o núcleo do projeto, ficará em execução em um computador que está ligado diretamente ao CCPC do MiniDAQ por cabo Ethernet. Este computador será usado para controle e aquisição de dados. O CCPC é ligado à AMC40 por interface PCIexpress. Para fazer a interface entre a AMC40 e os demais componentes do sistema de controle no WinCC, é utilizado um componente (Manager) Driver chamado CCSERV.

O Driver CCSERV, é o componente de software executado no CCPC que é responsável por monitorar e acessar os registradores sintetizados no FPGA da AMC40. Este

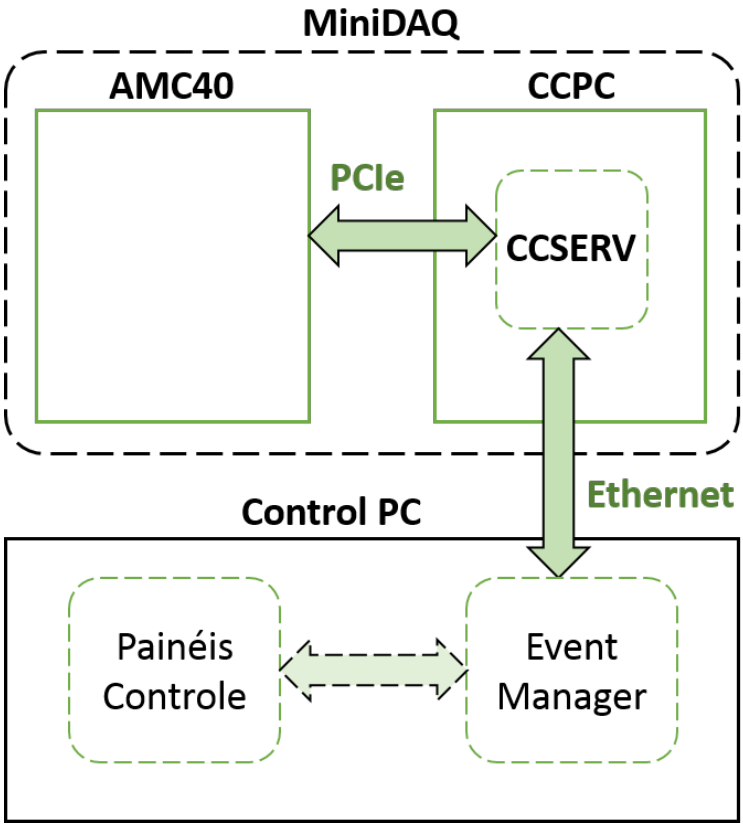

Figura 9.2: Diagrama de controle.

faz a interface entre o hardware (AMC40) e o Event Manager. A partir do Event Manager e através do CCSERV, outros Managers do sistema de controle tem acesso aos registradores da AMC40.

Note que o acesso aos registradores do firmware no FPGA da AMC40 permite acessar todos os demais registradores da eletrônica de *front-end*, uma vez que a interface para enviar comandos de controle para a *front-end* é toda implementada baseada nestes registradores da AMC40.

9.3 Cadeia de dados do sistema de controle

Conforme visto na seção 8.1.1, os comandos provenientes do sistema de controle são processados pelo bloco SOL40 no firmware do FPGA da AMC40 e enviados para a eletrônica de *front-end* através do link óptico GBT. Estes comandos chegam no GBT

Master. Dependendo do tipo de comando, pode ser endereçado a registradores no próprio GBT Master, ou a registradores em outros componentes da *front-end*, onde neste caso o mesmo é repassado aos GBT SCAs através dos E-links. Estes comandos repassados, por sua vez, podem ser endereçados ao próprio GBT SCA ou ainda serem comandos a ser repassados através da interface I2C, SPI ou JTAG para uma terceira camada de dispositivos.

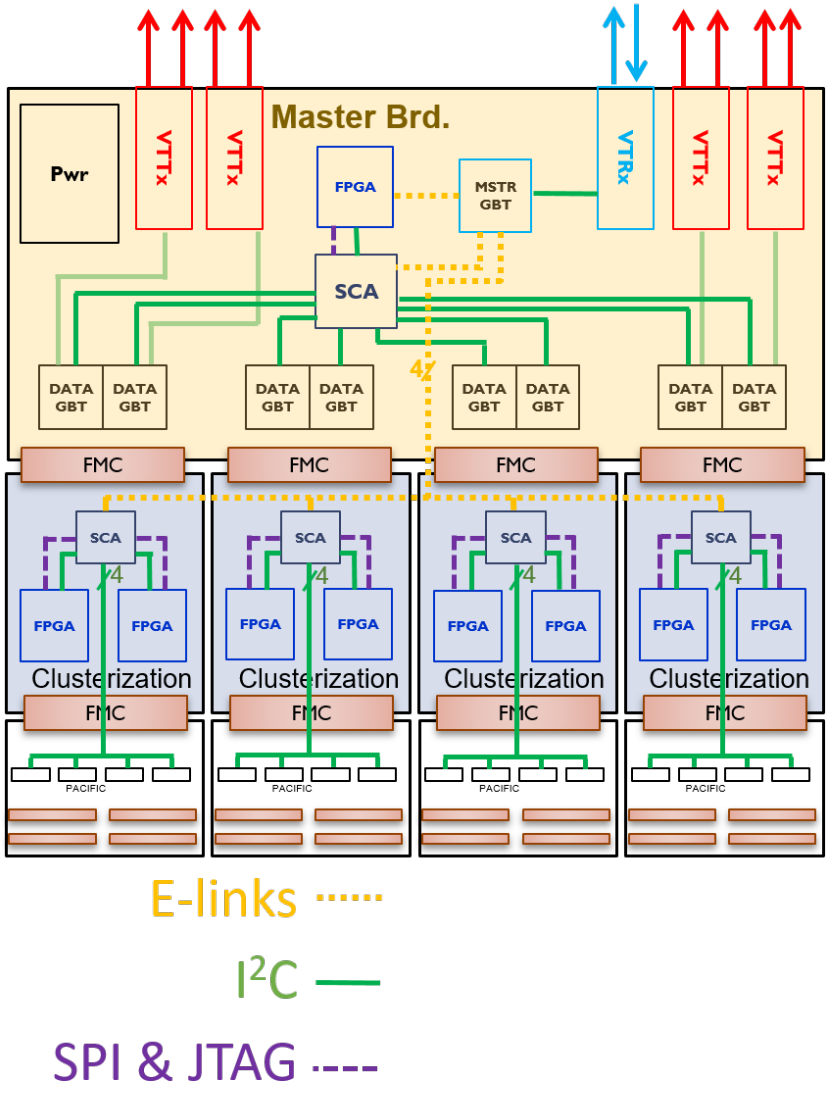

Figura 9.3: Diagrama dos caminhos de controle encontrados na eletrônica de *front-end*.

O caminho de controle da eletrônica de *front-end* pode ser visto na Figura 9.3. Ele é composto por:

- Um GBTx Master no topo, marcado na Figura 9.3 como “MSTR GBT”, que envia e recebe informações de controle através do par de links ópticos do VTRx.
- GBTs SCA, conectados ao GBT Master através de E-links, e que estão marcados na Figura 9.3 como “SCA”.
- Componentes I2C, SPI ou JTAG, que são acessados através de portas de comunicação de diferentes protocolos contidas nos GBTs SCA. Os GBTs de dados, os chips PACIFICs e os FPGAs de clusterização entre outros componentes da *front-end* se encaixam neste grupo.

9.4 Utilização da placa VLDB para desenvolvimento

O objetivo final do sistema de controle era acessar registradores contidos nos componentes da eletrônica de *front-end*, no entanto, no momento em que foi iniciado o desenvolvimento do sistema de controle ainda não havia protótipo da eletrônica de *front-end* produzido. No lugar da *front-end*, foi utilizado um módulo de demonstração chamada VLDB (Versatile Link Demo Board) onde havia um chip GBTx e um chip GBT SCA.

A placa VLDB (Figura 9.4) é um módulo de demonstração desenvolvida pelo CERN para prover um kit de avaliação do sistema Versatile Link, cujo objetivo é prover um design de referência com interfaces para os principais componentes comuns a todos os experimentos do LHC, compondo os principais elementos do Versatile Link.

Utilizando duas placas VLDB, é possível reproduzir até certo nível a cadeia de controle que será utilizada no experimento. Em uma das placas VLDB, o GBTx pode ser tratado como o GBTx Master, e seu GBT SCA como um dos 5 GBT SCAs ligados ao GBTx Master através de E-links. Como componente I2C, podemos utilizar o GBTx de uma segunda placa VLDB. Neste caso este segundo GBTx estaria representando um dos 8 GBTx de dados da *front-end*.

Desta forma toda a cadeia de controle pode ser representada por um setup com duas placas VLDBs, e desta forma foi possível iniciar o desenvolvimento do sistema de controle sem de fato ter um protótipo da eletrônica de *front-end* em mãos. Uma imagem do setup utilizado pode ser vista na Figura 9.5.

Figura 9.4: Vista superior da placa VLDB.

9.5 Camada de Software

Como já mencionado anteriormente, um dos principais papéis que o sistema de controle exerce é fazer a ponte entre valores de registradores no hardware do experimento e componentes visuais na tela do usuário operador. Para realizar esta tarefa, o sistema de controle conta com 4 importantes componentes de software, que podem ser vistos na lista abaixo.

Os dois primeiros foram desenvolvidos pelo CERN e já existiam previamente ao desenvolvimento do trabalho exposto neste documento. Os dois ultimos foram desenvolvidos pelo grupo do sistema de teste do SciFi e integram o objeto desta dissertação.

- Um componente driver, chamado CCSERV, que fica em execução no CCPC e é responsável pela ponte entre os registradores da AMC40 e o WinCC.
- Um componente do WinCC, chamado PVSS00dim, ao qual se conecta o driver CC-

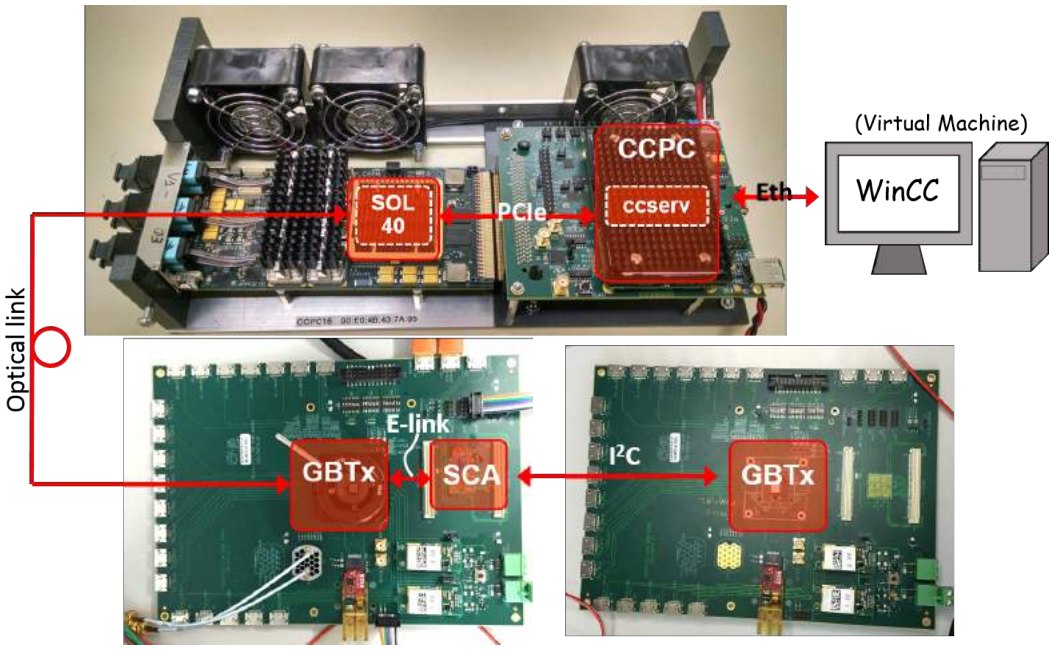

Figura 9.5: Setup com o MiniDAQ e placas VLDB para reproduzir a cadeia de controle encontrada na eletrônica de *front-end*.

SERV e é responsável pela interface entre o driver e os demais managers do projeto WinCC.

- Uma biblioteca específica para aplicação do SciFi, que abstraísse a escrita dos comandos nos registradores do CCPC a fim de criar funções compreensíveis para o usuário.

- Painéis com interface gráfica que utilizassem as funções da biblioteca anterior para exibir o status e controlar registradores do hardware no experimento.

CCSERV O componente driver CCSERV é um software feito para ficar em execução no CCPC. Ele é capaz de ler e escrever em registradores no FPGA da AMC40 através de uma conexão PCIexpress e também se comunicar com o sistema WinCC através do protocolo DIM. O CCSERV é também um servidor DIM.

DIM é um protocolo de gerenciamente de informação distribuída (Distributed Information Management) e que é utilizado para comunicação entre um ou mais CCPCs com o sistema WinCC. O DIM é um protocolo criado no CERN e amplamente utilizado no experimento LHCb.

O CCSERV, além de servir como uma ponte entre os registradores da AMC40 e o sistema WinCC, também oferece serviços com os quais se pode monitorar valores nestes registradores. É possível enviar um comando ao CCSERV para subscrever a determinados registradores de modo que este passa a realizar uma leitura dos tais registradores periodicamente. Sempre que uma mudança no valor de um dos registradores for registrada, o CCSERV atualizará o valor do datapoint correspondente.

PVSS00dim Como mencionado no parágrafo anterior, o CCSERV se comunica com o sistema WinCC através do protocolo DIM. O WinCC não possui suporte nativo a este protocolo, por isto foi necessário desenvolver um componente próprio para enviar e receber informações no protocolo DIM. Este componente é o chamado PVSS00dim e este fica em execução no sistema WinCC como um manager. Ele é tanto um servidor DIM como um cliente DIM.

Para enviar e receber dados e comandos através do PVSS00dim, os desenvolvedores fornecem uma API contendo diversas funções de utilização, incluindo funções de leitura e escrita de registradores e comandos tal qual o de subscrição de registradores. A API pode ser vista no diagrama da Figura 9.6.

Devido à esta API, do ponto de vista do desenvolvedor do sistema de controle, este não precisa ter conhecimentos acerca do protocolo DIM. As funções da API permitem enviar e receber estes dados abstraindo as camadas mais baixas do protocolo DIM.

Para escrever em um registrador na AMC40, por exemplo, basta passar à função de escrita o datapoint associado ao referido registrador e o valor a ser escrito.

Framework WinCC para software de controle do LHCb. A colaboração do LHCb criou um framework para WinCC chamado "lbfwhw", que compreende diversos componentes e bibliotecas comumente utilizadas no experimento. O componente PVSS00dim, por exemplo, faz parte deste framework.

O Framework LHCb é baseado no Framework JCOP, que é um framework criado para prover aos experimentos do LHC as principais funcionalidades de um sistema de controle e que são comuns a um grande número de usuários. O Framework LHCb nada mais é do que todo o Framework JCOP adicionado de funções de interesse dos grupos da colaboração LHCb.

O sistema de controle da *front-end* do SciFi utiliza o Framework LHCb para algumas funções. O Framework LHCb não só implementa um servidor e cliente DIM para estabelecer a comunicação entre o CCPC e o Event Manager, mas também provê diversas funções de leitura e escrita em registradores instanciados em FPGAs conectados a CCPCs via PCIexpress.

É através destas funções do Framework LHCb que o sistema de controle atua e monitora os registradores no firmware da AMC40. Mas note que o Framework LHCb não abrange a interface com os registradores da *front-end*. A mesma deve ser feita a partir da AMC40 através do link GBT. Uma solução foi implementada pelo grupo do sistema de teste do SciFi e se encontra descrito na seção 9.8.

A Figura 9.6 exibe o caminho que liga o painel no computador do operador ao registrador contido na AMC40. Quando o operador do sistema, através de um painel, atua sobre um controle gráfico representando um registrador, o Manager de interface de usuário (UIM) chama através do EVM uma função da API do JCOP, que por sua vez envia o comando ao driver (D) utilizando o protocolo DIM.

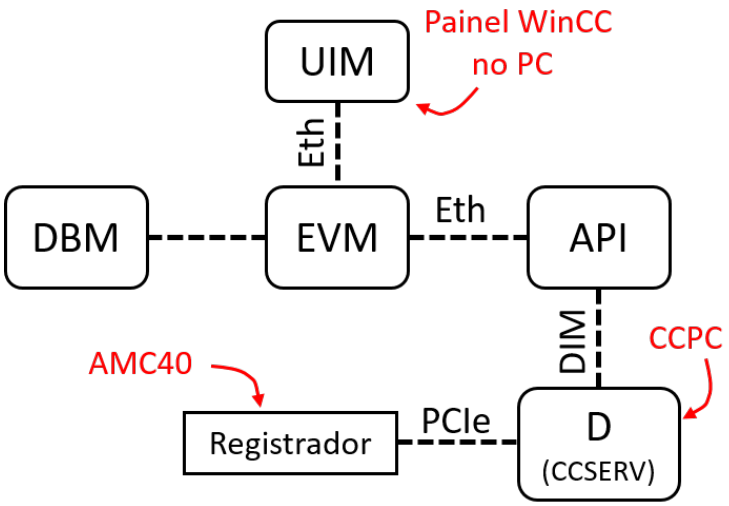

Figura 9.6: Caminho de dados do painel gráfico WinCC até um registrador no firmware da AMC40.

9.6 Bloco SOL40_SCA do firmware do MiniDAQ

O bloco SOL40_SCA é a parte do firmware do MiniDAQ que recebe os comandos do CCPC referente ao Slow Control e os envia para a *front-end* através do link GBT.

Ele interpreta comandos vindos do CCPC, gera e insere na palavra GBT a ser enviada pelo SOL40 os bits correspondentes a estes comandos que serão executados nos GBT SCAs contidos na *front-end*. A palavra GBT enviada pelo SOL40 para o GBT Master contém vários conjuntos de 2 bits chamados de e-links. Cada GBT SCA da *front-end* é ligado a um destes e-links e através destes recebe comandos.

Os comandos podem significar escrita ou leitura de registradores de configuração dos GBT SCAs ou execução de comandos referentes aos periféricos contidos no GBT SCA, como leitura do ADC, envio de dados via I2C e etc. Mesmo os comandos aos periféricos são na verdade comando de escrita em registradores que por sua vez são interpretados pelo GBT SCA. Isto significa que todos os comandos passados ao GBT SCA são essencialmente comandos de escrita em registradores.

Na *front-end* existem também componentes com os quais nos comunicamos através das portas I2C do GBT SCA, ou seja, registradores dos quais só podemos escrever através da porta I2C do GBT SCA. Para estes casos nós enviamos um comando de escrita I2C determinando em qual porta o componente alvo está ligado, e os dados a serem enviados pela I2C variam de acordo com a configuração do protocolo do dispositivo. Um protocolo I2C típico, por exemplo, recebe 7 bits de endereço, seguido de 1 bit que define se o comando é de leitura ou de escrita, o número de bytes a serem lidos ou escritos e então os dados a serem enviados.

Para enviar um comando ao bloco SOL40_SCA se deve fazê-lo através de seus registradores. Primeiro se escreve em um conjunto de 6 registradores o comando que se deseja enviar, e após isto se escreve em um bit específico de um registrador de controle que engatilha a leitura do comando contido nos 6 registradores mencionados. Após a leitura, o bloco SOL40_SCA processa o comando e inicia o processo de formação das palavras GBT para envio do comando aos dipositivos da *front-end*.

Os registradores onde se escreve o comando são 6 registradores de 32 bits, dos quais os dois primeiros definem o comando e os demais são reservados para dados. Os registradores para dados são utilizados em casos onde o comando exige uma carga de dados, como por exemplo escrita de dados através da porta I2C, JTAG e etc.

Uma ilustração destes registradores pode ser vista na Figura 9.7, obtida da nota técnica [20].

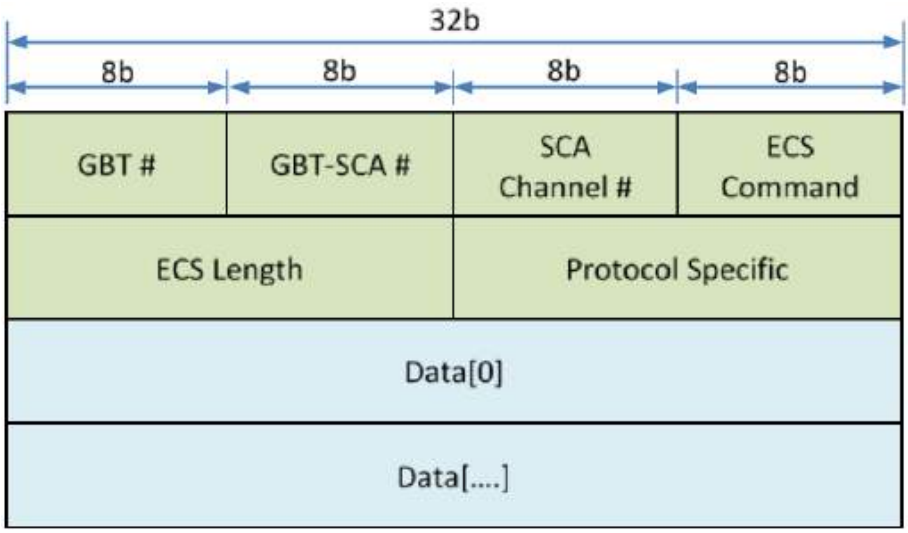

Figura 9.7: Ilustração dos campos contido nos registradores de comando do bloco SOL40_SCA. Ilustração retirada de [20].

Cada linha da Figura 9.7 corresponde a um registrador. No momento da elaboração deste texto eram utilizados 6 registradores, portanto a linha Data[....] corresponderia a 3 linhas adicionais, formando 4 registradores de dados (Data[0] ao Data[3]), além dos 2 registradores contendo o comando.

Uma breve explicação dos campos podem ser encontradas a seguir:

- GBT# é um número de 8 bits que corresponde ao número do link GBT através do qual se deseja enviar o comando. Como cada link é ligado a um único GBT Master, que por sua vez é único por *front-end*, na prática este campo define para qual *front-end* o comando está sendo enviado. No caso do sistema de teste do SciFi será utilizado sempre um único link ligado a uma única *front-end*, portanto este campo é sempre 0. Mas o bloco SOL40_SCA foi desenvolvido considerando a possibilidade de gerenciar diversas *front-end*s através de diversos links GBT.

- GBT-SCA# é um número de 8 bits que se refere a para qual GBT SCA se deseja enviar o comando. Como no sistema de chips GBT cada GBT SCA é ligado a um único e-link, na prática este número seleciona para qual e-link do GBT Master o comando será enviado.

- SCA Channel# é um número de 8 bits que se refere a qual canal do GBT SCA será enviado o comando. Note que internamente ao GBT SCA existem vários blocos periféricos e um bloco de configuração, e são todos referidos como canais. Este número define, por exemplo, se o comando é para escrita em um registrador interno de configuração, ou para registradores relacionados aos periféricos como ADC, transceptor I2C e etc.

- ECS Command é um número de 8 bits que se refere ao tipo de comando a ser enviado. Os comandos disponíveis estão listados na documentação do GBT SCA [21]. É neste campo que se define, por exemplo, se um comando de acesso à porta I2C é de escrita ou leitura.

- ECS Length é um número de 16 bits que corresponde ao volume de dados (em bytes) contidos nos registradores de dados. Com 4 registradores de dados, o volume total é de 16 bytes. Quando este campo especifica um volume de dados inferior ao máximo, o conteúdo adicional é ignorado.

- Protocol Specific é um campo reservado que depende do protocolo utilizado no GBT SCA.

- Data[0..3] corresponde aos registradores que contém os dados a serem enviados junto com o comando, quando o comando é um do tipo que requer dados. Caso contrário estes registradores são ignorados.

Com os registradores cujo o conteúdo foi explicado acima é possível escrever o comando em que se deseja enviar para o bloco SOL40_SCA.

Após escrito o comando, para acionar o envio deve-se escrever '1' no primeiro bit do registrador de controle.

Vemos uma ilustração do registrador de controle na Figura 9.8.

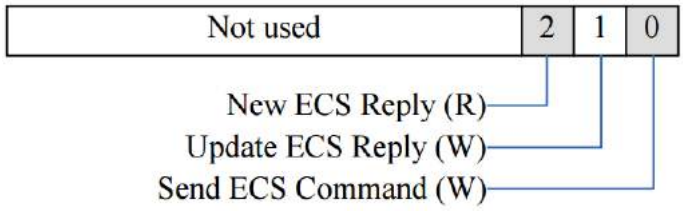

Figura 9.8: Ilustração do registrador de controle utilizado pelo bloco SOL40_SCA.

Adicionalmente, através do registrador de controle também se pode ver o status de resposta para os comandos de leitura, cujos dados de resposta são guardados também em um conjunto de 6 registradores. O bit 2 indica quando há uma nova resposta a ser lida, e quando já foi efetuada a leitura, pode-se limpar o registrador para a próxima resposta escrevendo '1' no bit 1 do registrador de controle.

9.7 Datapoints para descrição da Front-End do SciFi

Os principais parâmetros a que se deseja monitorar e atuar no sistema de controle do SciFi são os valores dos registradores. Conforme visto nas seções anteriores, um sistema WinCC é construído sobre uma estrutura de datapoints, e por esta razão é desejável criar uma estrutura de datapoints que descreva todo o seu sistema. No caso do sistema de teste do SciFi, foi criado um tipo de datapoint para cada dispositivo que se desejava controlar, contendo todos os registradores de interesse até o momento.

Os datapoints que representam os dispositivos (os chips) foram inseridos de forma hierárquica em datapoints de nível superior que representam as placas, a fim de tê-los organizados na mesma forma da estrutura real que se encontra na *front-end*. Veja na Figura 9.9 a representação hierárquica dos chips.

Cada bloco da Figura 9.9 representa um chip. Os quadrados pontilhados representam as placas em que estes estão contidos. Foi criado um datapoint correspondente para cada chip. Para as placas, foi criado um datapoint representando a Master Board, mas para a Clusterization Board e a PACIFIC Board, por fins de simplificação, foi criado um único datapoint chamado Cluster_Board que representa as duas placas.

Na Figura 9.12, retirada do editor de datapoints do WinCC, é possível ver que os chips PACIFIC e *cluster* FPGA se encontram em paralelo sob o mesmo datapoint (Cluster_Board).

Todos os datapoints representando os chips da *front-end* possuem uma estrutura semelhante em comum. São compostos por dois conjuntos de subdatapoints, sendo um chamado "config" e outro "registers". Dentro do grupo "config" há datapoints que guardam as configurações daquele hardware. Não representam valores físicos em um registrador ou algo do tipo, mas sim variáveis que configuram aquele hardware. Comum a todos os dispositivos no projeto há o datapoint "address", que guarda o endereço pelo qual o

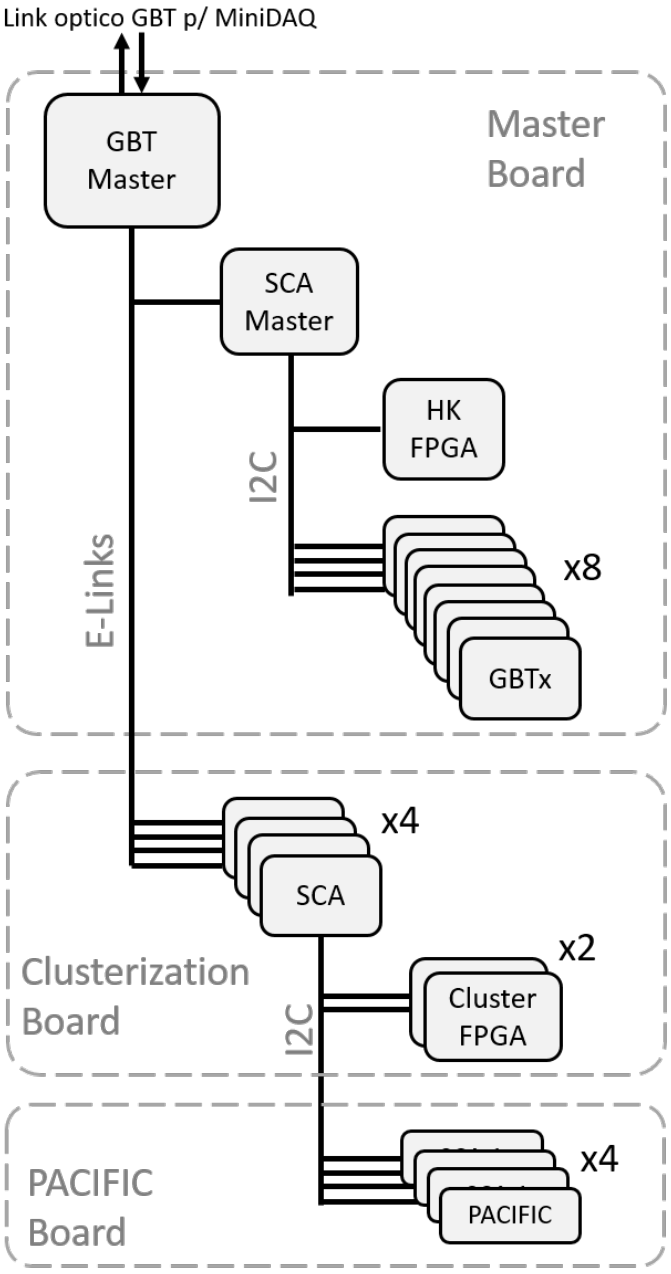

Figura 9.9: Diagrama dos dispositivos representados por datapoints no sistema de controle.

dispositivo em questão responde, e também o tamanho em bytes da palavra de endereço.

Interno ao grupo “registers” deverá existir um datapoint para cada registrador. Este conterá 3 valores: “address” é o endereço do registrador, “length” é o comprimento em bytes e “data” é o valor armazenado no registrador.

Pode-se ver a estrutura padrão para os datapoints criados na Figura 9.10, que exibe a estrutura do datapoint criado para o PACIFIC, embora ainda não tenha sido divulgada a relação de registradores que o chip terá, estes são representados por 2 registradores de teste.

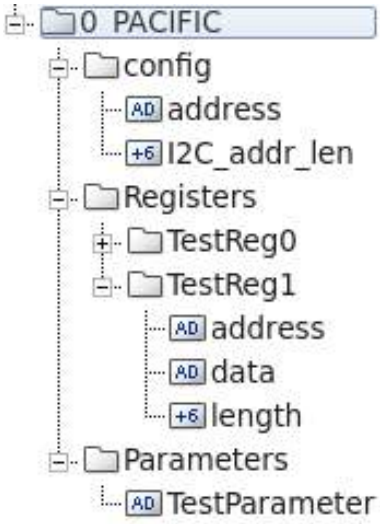

Figura 9.10: Modelo genérico de datapoint utilizado para os dispositivos da *front-end*.

Em conformidade com a Figura 9.9, os datapoints criados para o projeto do sistema de controle foram os seguintes:

- 0_PACIFIC representa o chip PACIFIC.
- 0_FPGA_*cluster* representando os FPGAs responsáveis pela clusterização dos dados.
- 0_SCA representando os chips GBT SCA.
- 0_GBTx_old representando os chips GBTx.
- 0_HK_FPGA representando o FPGA HouseKeeping, responsável pelo gerenciamento da Master Board.
- 2_Cluster_Board representa em conjunto uma Clusterization Board e uma PACIFIC Board, contendo todos os seus componentes.
- 1_Master_Board representa uma Master Board, contendo todos os seus componentes.

Da lista de datapoints acima, os que representam os chips PACIFIC, *cluster* FPGA e HouseKeeping FPGA não tiveram seus registradores listados pois no momento deste trabalho no sistema de controle estes dispositivos ainda não haviam suas especificações de registradores divulgadas. Para estes foi criado um tipo genérico conforme exibido na Figura 9.10, ainda a ser preenchido com os registradores correspondentes uma vez que estes sejam divulgados.

O datapoint do GBTx segue o mesmo modelo genérico. Tem um grupo chamado "config" que guarda seu endereço e largura do endereço, e um grupo chamado "registers" que guarda todos os registradores com seus respectivos efndereços, comprimentos e valores, conforme exemplo da Figura 9.10.

Para o GBTx, foram criados 363 registradores. Foi utilizado um script para ler a tabela de registradores contida na seção 17.2 do Manual do GBTx [22] e listá-los em formato de arquivo de texto. Após isto foi criado uma função no WinCC para interpretar o arquivo de texto com os registradores e criar um tipo de datapoint correspondente, e outra função para preencher tais datapoints com os valores de endereço, dados e etc. Os registradores utilizados são os de endereço de 0 a 365, com exceção dos 362, 363 e 364 que são registradores vazios e não foram adicionados.

Para o GBT SCA, foram criados 75 registradores. Diferente do realizado no GBTx, os registradores para o GBT SCA foram selecionados baseados no interesse para o sistema de teste. Não havia uma lista completa de registradores e portanto estes foram filtrados do Manual do GBT SCA [21] e digitados a mão em uma planilha eletrônica, a partir da qual se pôde transformar a lista em um conjunto de comandos de criação de datapoints, a ser copiado e executado no WinCC.

A estrutura do datapoint do GBT SCA é um pouco diferente. Ao invés de uma única grande lista de registradores, este é dividido em "channels" ou canais referentes aos blocos internos do chip. Cada canal tem seu próprio endereço e registradores. Os canais existentes podem ser vistos na Figura 9.11. Note que existem 16 canais de I2C, mas por economia de espaço, foi suprimido da Figura 9.11 os canais de 1 ao 14. Na imagem original existe uma lista com 16 canais I2C.

A forma de escrita e leitura nos registradores do GBT SCA também difere dos demais. Não há um endereço para os registradores no qual se pode escrever ou ler. Ao invés, o GBT SCA apenas interpreta comandos. Existem comandos diferentes e únicos para escrever e ler em cada registrador. Por consequência disto não se especifica a operação

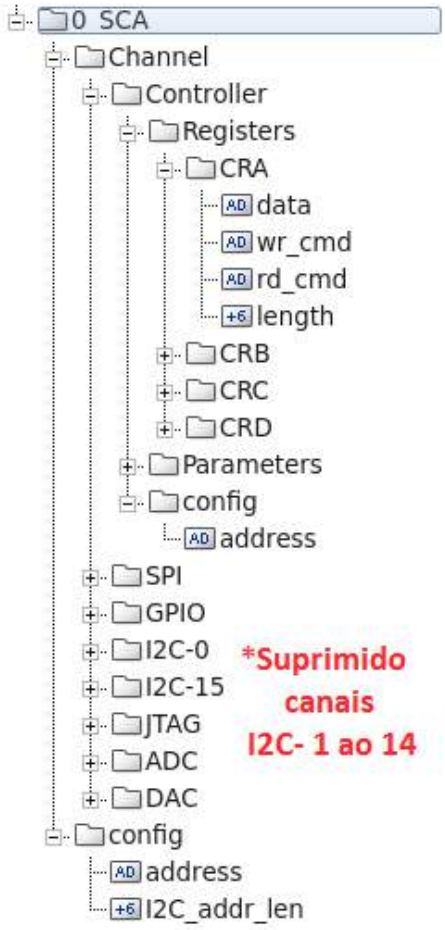

Figura 9.11: Estrutura de datapoints do chip GBT SCA.

(leitura ou escrita) e o endereço. Apenas o comando.

Por esta razão na Figura 9.11 não se vê um campo de endereço mas, para cada registrador, um campo chamado "wr_cmd" (write command) e "rd_cmd" (read command), representando os comandos de escrita e leitura respectivamente para o dado registrador.

Registradores que são só de escrita ou só de leitura simplesmente não terão o comando equivalente à função complementar.

Além dos datapoints que representam os dispositivos, também há os que representam as placas. Estes contém internamente diversos datapoints representando os dispositivos encontrados na placa. Desta forma é possível organizar o sistema de forma hierárquica. Isto também permite ao sistema ser escalável e modular.

Quando criamos um elemento do tipo datapoint 1_Master_Board, por exemplo, cri-

amos um conjunto de datapoints que representa uma *front-end* completa. Basta alterar um único datapoint, o endereço do GBT Master desta *front-end*, e então estaremos definindo de forma individual todos os datapoints subordinados a este GBT Master.

Na Figura 9.12 podemos ver a estrutura dos datapoints que representam as placas da *front-end*, utilizando os datapoints dos dispositivos já descritos nesta seção.

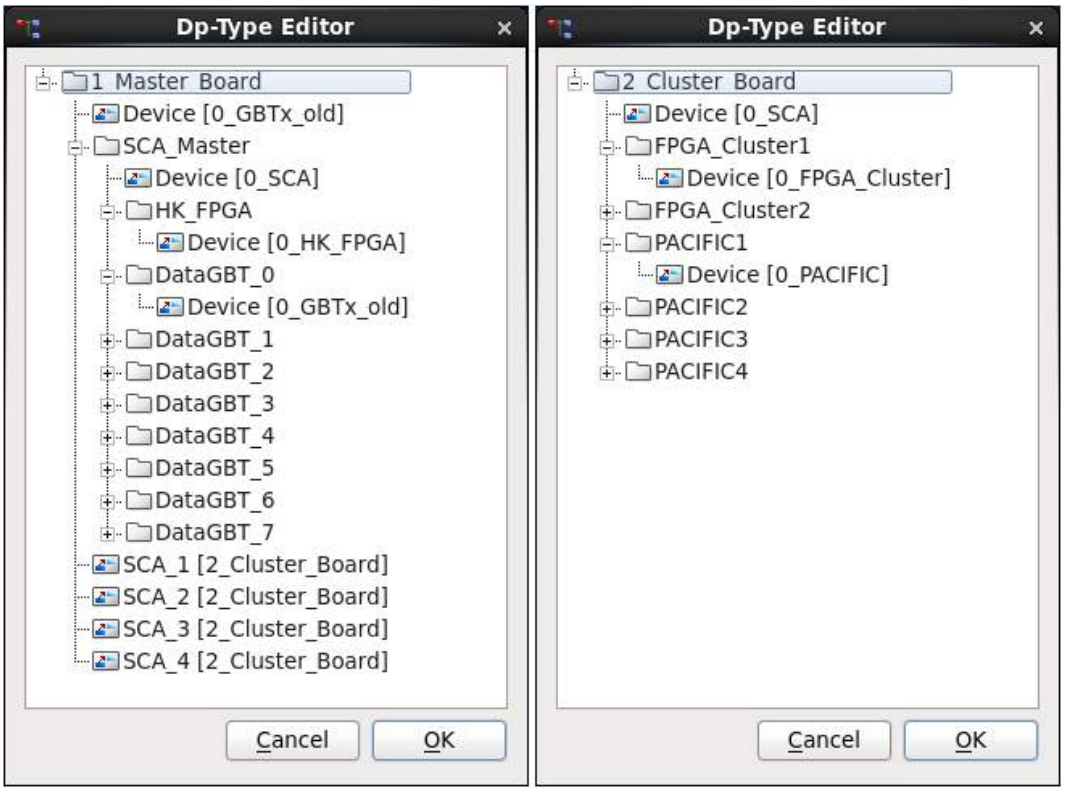

Figura 9.12: Janela de edição de tipo de datapoint. À esquerda o datapoint Master_Board e à direita o Cluster_Board.

9.8 Biblioteca de controle fwlbSCIFI.

Vimos nas seções anteriores que o sistema de controle é construído em cima de uma estrutura de datapoints, em que estes por sua vez, devem representar os valores encontrados no hardware do sistema.

Vimos também que o Framework LHCb fornece funções que podem ser utilizadas para acesso aos registradores do firmware do MiniDAQ, mas não ao registradores da *front-end*, que é o objetivo final do sistema de controle.

A biblioteca fwlbSCIFI foi criada para fornecer ao usuário funções de acesso aos registradores que se encontram na *front-end*. Ela utiliza o Framework LHCb para acessar os registradores do firmware do MiniDAQ e através destes enviar comandos ao bloco SOL40_SCA, a partir do qual é possível controlar os registradores da *front-end* através do link GBT conectado à *front-end*.

O objetivo final da biblioteca fwlbSCIFI é provêr funções para abstrair o procedimento de envio de comandos para a *front-end*, de modo que o usuário final precise apenas ler e escrever nos datapoints do sistema, com as funções e controles gráficos nativos do WinCC. Um manager utilizando a biblioteca fwlbSCIFI se encarregaria da interface com o bloco SOL40_SCA.

Até a data de elaboração deste documento a biblioteca fwlbSCIFI compreendia 9 funções, das quais cinco delas foram criadas com o intuito de serem usadas internamente pelas funções da API, que ficam disponíveis ao usuário. Estas cinco funções internas são:

- SOL40SCA_SetCommand
- SOL40SCA_Go
- SOL40SCA_SetReplyAddress
- SOL40SCA_NewReply
- SOL40SCA_GetReply

9.8.1 Funções de acesso direto ao bloco SOL40_SCA.

As funções com o prefixo "SOL40SCA" são as funções que atuam diretamente nos registradores do bloco SOL40_SCA. Estas funções, a princípio, não precisam ser utilizadas pelo usuário final, uma vez que estas são chamadas pelas demais funções.

A função SOL40SCA_SetCommand recebe como argumento uma string que deve ser hexadecimal. A função retornará um aviso caso contenha um caractere não hexadecimal e será interrompida. Esta função apenas verifica os dados, os divide em 6 pedaços de 32 bits e os escreve nos 6 registradores de comando do bloco SOL40_SCA. São os registradores representados na Figura 9.7.

A função SOL40SCA_Go é a função responsável por dizer ao bloco SOL40_SCA que há um novo comando a ser executado. Na prática ela escreve '1' no bit 0 do registrador de controle do bloco SOL40_SCA, que é um bit apenas de escrita. Como se pode ver na Figura 9.8, escrever neste bit engatilha o envio do comando ECS. Para cada novo comando a ser enviado, deve-se escrevê-los nos registradores e após isto acionar o envio com a escrita de '1' no registrador de controle.

Para cada comando enviado ao bloco SOL40_SCA é gerada uma resposta. Esta resposta é armazenada em uma memória e o formato é semelhante à palavra de comando. Os primeiros 32 bits se referem ao endereço do comando, onde contém informação do número do Master GBT, SCA e canal I2C para o qual foi enviado o comando. A resposta é lida a partir de 6 registradores de 32 bits, mas a resposta não fica imediatamente disponível. Imagine que foi enviado dois comandos para dispositivos I2C diferentes, com tempos de resposta diferentes. É possível selecionar qual resposta será vizualizada primeiro, filtrando o valor de saída da memória onde as respostas são guardadas. Portanto, caso seja enviado múltiplos comandos, pode-se esperar a resposta de um dispositivo específico cujo o endereço corresponda ao comando enviado.

Para selecionar de qual dispositivo se deseja receber a resposta, utiliza-se a função SOL40SCA_SetReplyAddress. Ela seleciona o endereço de onde provém a resposta a ser lida pela memória. Isto ocorre escrevendo o valor desejado no registrador de endereço da memória. Esta função recebe uma string hexadecimal de 32 bits, interpreta e escreve no registrador mencionado.

Após escrever o endereço do comando cuja resposta se deseja ler, esta não se torna imediatamente disponível nos 6 registradores de onde se lê a resposta. É necessário solicitar uma nova resposta ao bloco SOL40_SCA. Isto é feito escrevendo '1' no bit 1 do registrador de controle. O bit 1 é um bit somente de escrita e é responsável por atualizar o valor dos 6 registradores com o último comando recebido correspondente ao endereço do registrador de endereço.

A função SOL40SCA_NewReply escreve '1' no bit 1 do registrador de controle a fim de solicitar nova resposta. Quando há uma nova resposta disponível, o bit 2 do registrador de controle fica ativado. Solicitar nova resposta nesta ocasião resetará o referido bit, indicando que a resposta do comando foi lida e não há outra disponível.

Após verificar a chegada de uma resposta e solicitá-la com a função SOL40SCA_NewReply, pode-se ler a resposta utilizando a função SOL40SCA_GetReply. É possível ler a resposta

completa acessando diretamente os registradores onde a resposta é armazenada, mas para fins de simplificação, a função SOL40SCA_GetReply faz isto pelo usuário. Ela lê a resposta e retorna uma string hexadecimal correspondente ao valor da resposta.

Estas são as funções internas da biblioteca fwlbSCIFI. Estas funções não foram criadas com o intuito de serem chamadas diretamente pelo usuário, mas são disponibilizadas da mesma forma que as outras para fins de desenvolvimento e teste.

9.8.2 Funções de envio de comandos à eletrônica de *front-end*.

As quatro funções restantes são as funções que foram criadas com o intuito de criar uma camada de abstração do processo de envio de comandos para a *front-end*, visando simplificar a utilização do sistema de controle. As referidas funções são listadas a seguir:

- read_from_SCA
- write_to_SCA
- read_I2C_device
- write_I2C_device

As funções de leitura, read_from_SCA e read_I2C_device, são utilizadas para ler um registrador em um chip GBT SCA ou em um dos dispositivos conectados aos canais I2C do GBT SCA respectivamente. As funções write_to_SCA e write_I2C_device são utilizadas para escrever em algum dos registradores ou enviar comandos para os mesmos chips.

Funções de acesso ao SCA. Para ler ou escrever nos registradores do GBT SCA ou enviar comandos, as funções read_from_SCA e write_to_SCA sao utilizadas. Ambas recebem dois parâmetros em comum, que é o número do link GBT o qual o bloco SOL40 deve utilizar para comunicação com a *front-end*, e o datapoint referente ao registrador que se deseja escrever.

A função de escrita recebe um parâmetro adicional, que são os dados a serem enviados, no formato de string hexadecimal. A função de leitura, por sua vez, retorna os dados solicitados, também no formato de uma string hexadecimal.

Ambas as funções verificam os endereços dos dispositivos a serem utilizados, e a partir destes criam a string de comando a ser passada para o bloco SOL40_SCA. A exemplo, suponhamos que o registrador que se deseja acessar seja o "Control Register A" (CRA) do canal Controller do chip SCA Master. O datapoint correspondente seria:

```
GBT_Master0.SCA_Master.Device.Channel.Controller.Registers.CRA
```

O endereço do GBT Master será o valor do datapoint:

```
GBT_Master0.Device.config.address
```

O endereço do GBT SCA será o valor do datapoint:

```
GBT_Master0.SCA_Master.Device.config.address
```

O número do canal utilizado do GBT SCA será:

```
GBT_Master0.SCA_Master.Device.Channel.Controller.config.address
```

Após separar cada um destes valores, a função monta a palavra de comando baseado no formato descrito na seção 9.6, conforme se vê na Figura 9.7, e envia o comando ao bloco SOL40_SCA utilizando as funções SOL40SCA_SetCommand e SOL40SCA_Go.

Caso seja a função de leitura, após a função "Go"disparar o envio do comando, esta ainda realiza a leitura da resposta do comando, executando as funções "SetReplyAddress", "NewReply" e "GetReply".

Funções de acesso aos dispositivos I2C. Para ler ou escrever nos registradores dos dispositivos conectado em uma das portas I2C do GBT SCA, utiliza-se as funções read_I2C_device e write_I2C_device.

Assim como as funções para acesso ao SCA, ambam também recebem os parâmetros de link GBT o qual o bloco SOL40 deve utilizar para comunicação com a *front-end*, e o datapoint referente ao registrador que se deseja escrever.

A função de escrita recebe um parâmetro adicional, que são os dados a serem enviados, no formato de string hexadecimal. A função de leitura, por sua vez, retorna os dados solicitados, também no formato de uma string hexadecimal.

De forma semelhante às funções de acesso ao GBT SCA, estas também verificam os endereços dos dispositivos a serem utilizados, mas adicionando também o endereço I2C do registrador final a qual o comando é direcionado.

9.9 Painéis de controle para a FE

Após a criação dos datapoints que representam o sistema e as funções que fazem a interface entre os datapoints e os registradores reais, é necessário haver uma interface homem-máquina para que o operador do sistema possa monitorar e controlar o sistema de forma prática, intuitiva e eficiente. Para esta interface foram criados painéis com o WinCC.

Os painéis são processos (*Managers* UIM) que fornecem uma interface gráfica a ser utilizada pelo operador. O WinCC fornece uma ferramenta chamada GEDI (Graphical Editor) com a qual se pode fazer painéis de maneira simplificada.

É possível associar elementos dos painéis a datapoints, de forma a reagirem a mudanças nos valores dos datapoints. É possível, por exemplo, adicionar um componente do tipo barra de progresso, cujo preenchimento varie de acordo com o preenchimento de um buffer de dados no hardware e assim ter um feedback visual de um componente do sistema.

De maneira preliminar, para o sistema de controle da *front-end* do SciFi, foram criados apenas painéis representando os chips utilizados na *front-end*. Desta forma a maioria dos painéis são compostos de campos de texto que exprimem os valores reais dos registradores correspondentes.

Na Figura 9.13 se pode encontrar um painel criado para suporte à utilização e teste do bloco SOL40_SCA. Ele utiliza as funções de acesso direto ao bloco SOL40_SCA para envio de comando e monitoramento das respostas.

Na parte superior se pode ver um campo longo de texto chamado “To be sent:”, que é onde se escreve o comando a ser enviado. Ao clicar no botão “Set Command”, a função SOL40SCA_SetCommand é chamada com o parâmetro do campo de texto. A função irá interpretar a string e escrever nos 6 registradores de comando do bloco SOL40_SCA.

O seis campos mais curtos de texto, logo abaixo do botão “Set Command”, são os valores reais dos registradores de comando do bloco SOL40_SCA.

O botão “Go” chama a função de mesmo nome, da biblioteca fwlbSCIFI, que envia o comando à *front-end*.

A resposta do comando se encontrará então disponível para leitura nos campos de texto da parte inferior do painel.

Há uma caixa de texto no centro do painel, onde se vê escrito “Reply ready!” com fundo amarelo, que corresponde ao bit que indica se há uma nova resposta referente ao comando contido no registrador de endereço do bloco. Pode-se definir o valor do registrador de endereço do SOL40_SCA através dos pequenos campos de texto marcados pela borda com o texto “Reply Address”.

Este painel foi essencial durante o desenvolvimento do sistema e também do teste do firmware SOL40_SCA, e é muito util para depuração uma vez que o usuário tem acesso direto ao comando enviado ao bloco SOL40_SCA. No entanto, este não é um painel para uso do usuário final.

Os painéis a seguir são os painéis a serem utilizados pelo operador. Neles não se tem acesso aos registradores do firmware no MiniDAQ e no bloco SOL40_SCA, mas ao invés, toda esta parte é abstraída. Os comandos são criados, enviados e as respostas exibidas de forma automática.

A Figura 9.14 é o painel principal de acesso à *front-end*. Este painel funciona como um índice da placa de *front-end*. Nele há uma grande imagem do diagrama de blocos da *front-end* completa, onde se pode clicar nos componentes individuais para abrir o respectivo painel de acesso aos registradores do componente clicado.

Na Figura 9.15 podemos ver o painel de acesso aos registradores do GBTx. Os registradores foram agrupados em diferentes abas de acordo com o prefixo do nome do registrador. Os onze possíveis prefixos são listados nas abas.

O botão “Get All” solicita o valor de todos os registradores com o prefixo da aba selecionada, atualiza os datapoints correspondentes e os imprime na coluna “Read” da tabela. O botão “Set All” escreve os valores contidos na coluna “Write” da aba selecionada em todos os registradores correspondentes e então faz a leitura conforme realizada pelo botão “Get All”.

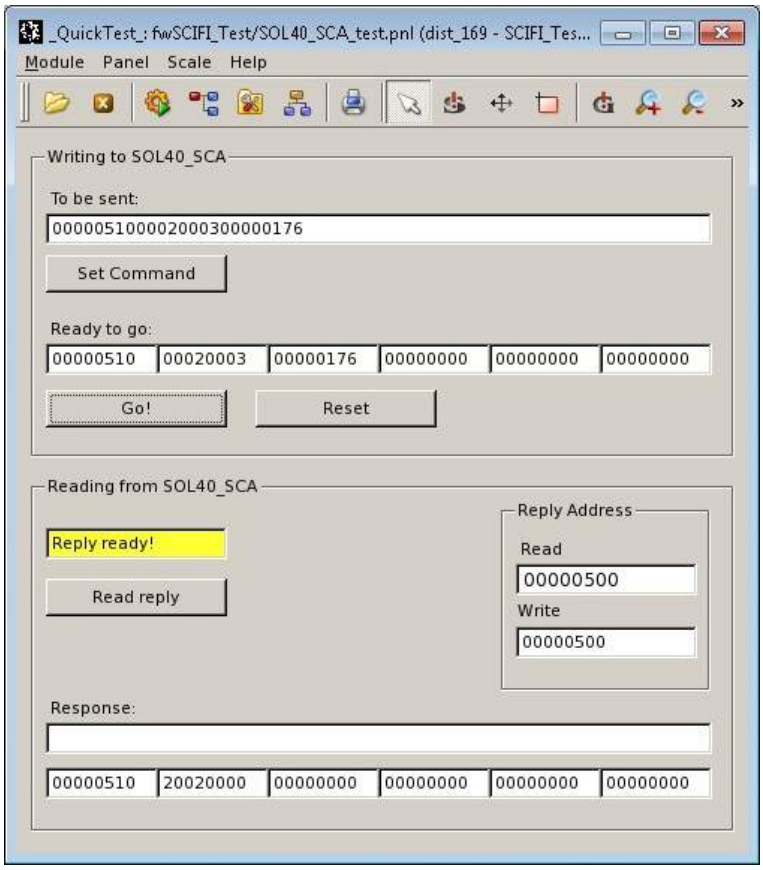

Figura 9.13: Painel desenvolvido para envio de comandos ao bloco SOL40_SCA.

O botão "Set Default Config" está fora do escopo das abas. Este escreve em todos os registradores do GBT os valores padrões de configuração.

Na Figura 9.16 se vê o painel de controle dos registradores do chip GBT SCA. Para abrir o painel, basta clicar em qualquer um dos chips GBT SCA no painel de controle da *front-end* que uma janela correspondente ao chip GBT SCA clicado será aberta.

Diferente do painel GBTx, os registradores são divididos por canais, conforme também organizado nos datapoints. Para cada canal existe uma aba contendo uma tabela com seus respectivos registradores. Dentro da aba do I2C existem 16 abas internas, criadas desta forma para facilitar a organização.

O botão "Write All" lê todos os valores da coluna "Write" e escreve nos registradores correspondentes na *front-end*. Após o ciclo de escrita a função realiza então a leitura dos mesmos registradores e atualiza os valores tanto nos datapoints como na tabela.

As células com cor de fundo cinza são para indicar os registradores apenas de escrita

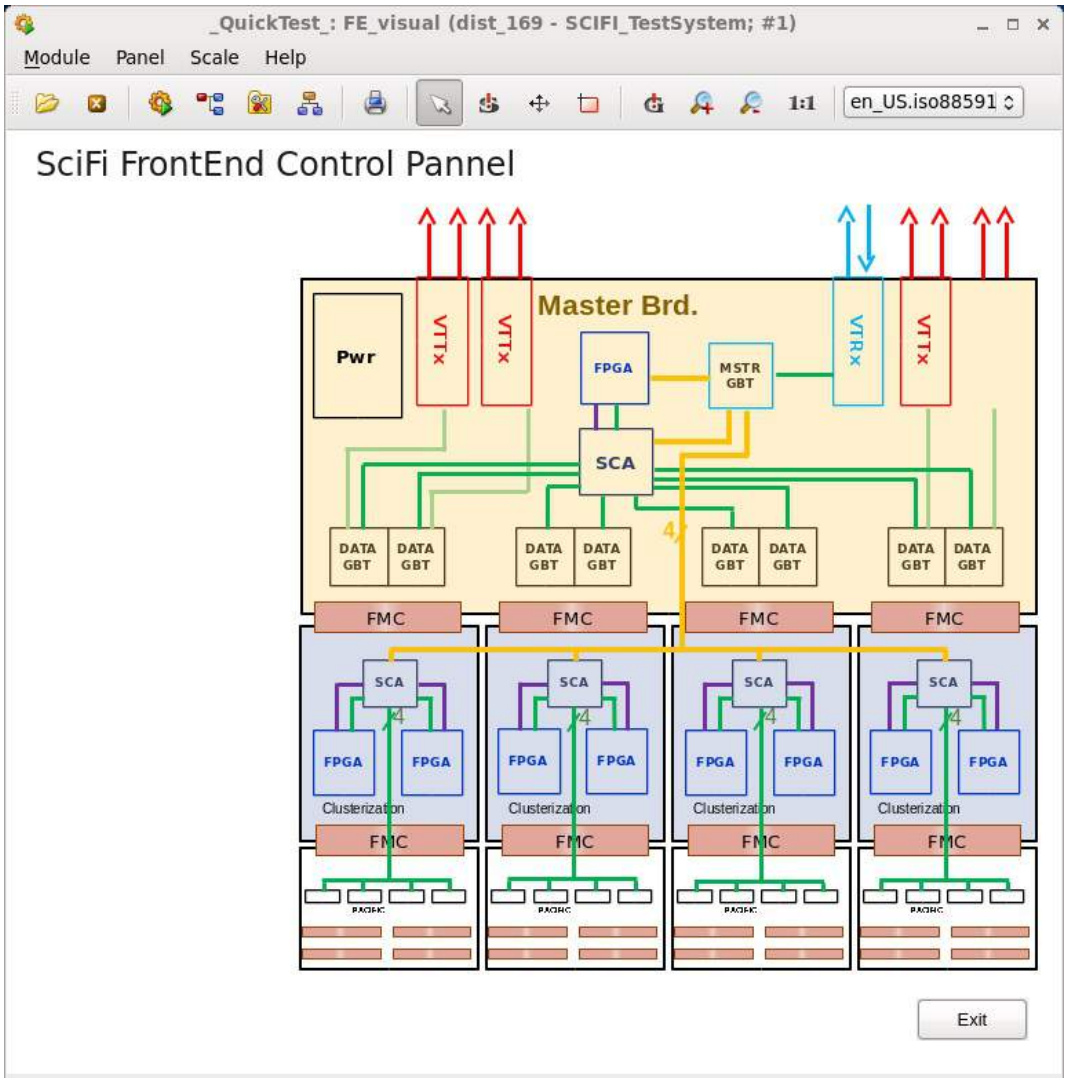

Figura 9.14: Painel desenvolvido para acesso aos paineis dos dispositivos da Front-end.

ou apenas de leitura. Caso um registrador seja apenas leitura, a célula da coluna “Write” estará cinza e desabilitada, e o mesmo ocorre para os somente escrita.

Estes quatro painéis foram os painéis criados inicialmente para validação do sistema de controle e uso preliminar para teste da *front-end*. Eles contem toda a funcionalidade necessária para configurar quaisquer dos chips mencionados, no entanto espera-se que no futuro as configurações mais usuais sejam aplicadas automaticamente por meio de scripts ou funções acionadas por botões.

Os dispositivos HouseKeeping FPGA, Clusterization FPGA e PACIFIC também terão seus respectivos painéis no futuro, no entanto, até a realização deste trabalho, ainda

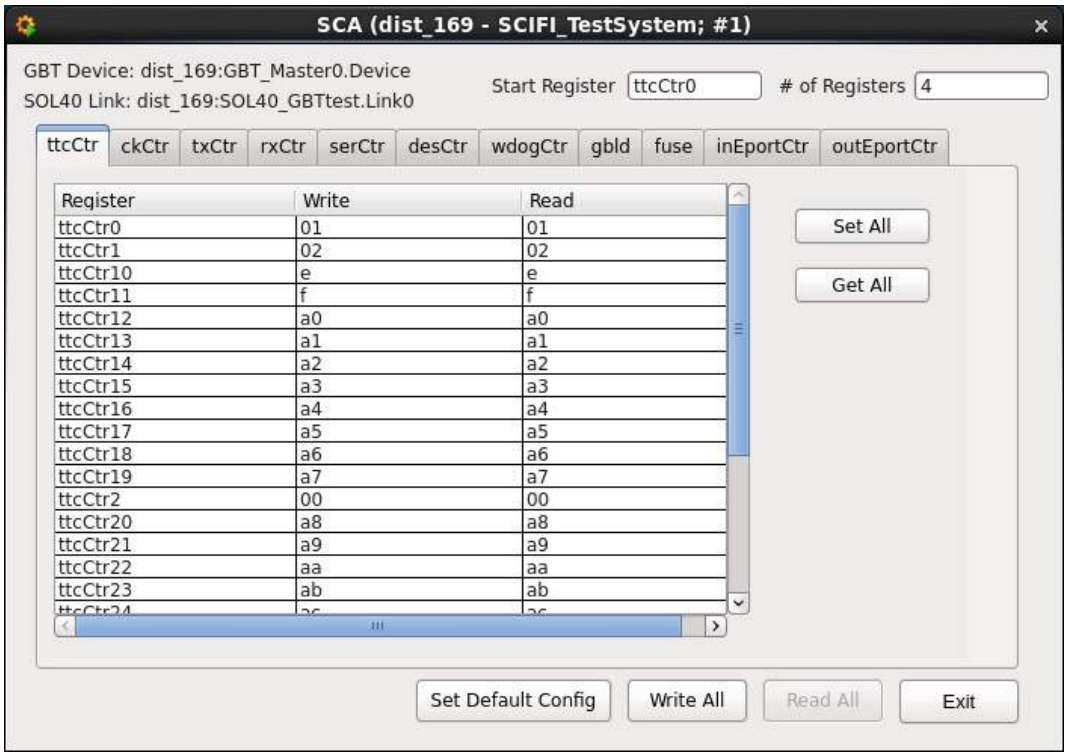

Figura 9.15: Painel desenvolvido para leitura e escrita nos registradores do GBTx.

não se encontravam prontas as informações necessárias para elaboração da estrutura de datapoints e consequentemente os painéis. Painéis genéricos foram criados em branco apenas como representação.

9.10 Configuração inicial do GBT utilizando um Arduino

Nas seções anteriores foram expostos detalhes sobre a comunicação do sistema de controle com a *front-end* e pudemos ver que toda esta comunicação utiliza o link GBT, que é conectado ao GBTx Master. Antes de utilizar o GBTx para tal função, é necessário configurá-lo. Configurar significa escrever determinados valores em diversos registradores.

O usuário que desejar utilizar um GBT para comunicação através do link óptico tem duas alternativas: configurar o GBTx através da porta I2C ou utilizar um GBT que já tenha os fuses queimados para a configuração desejada.

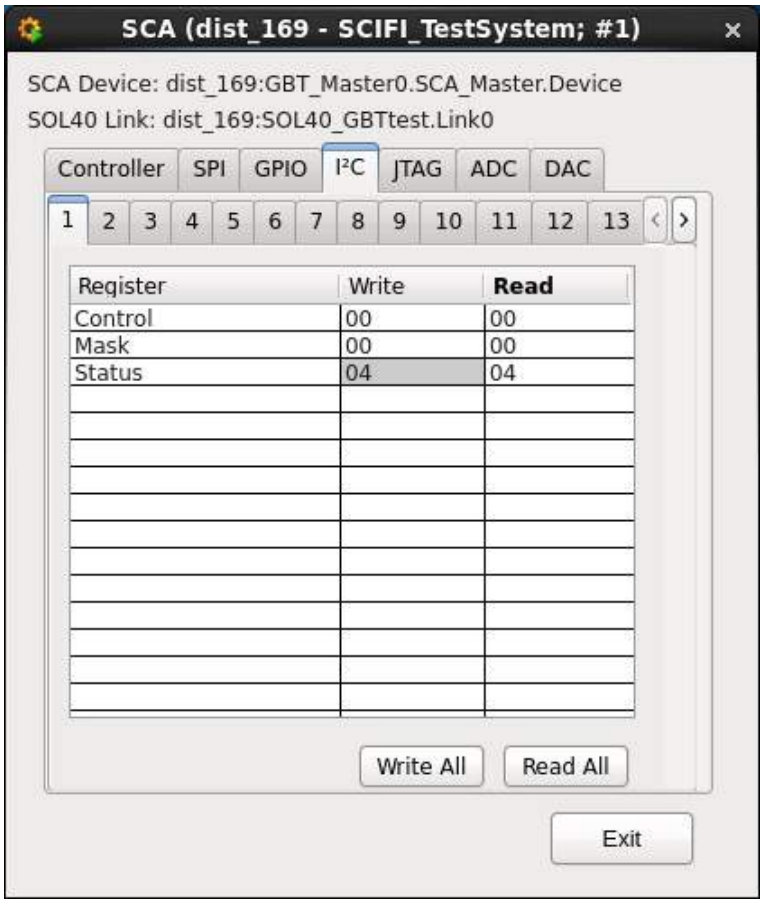

Figura 9.16: Painel desenvolvido para leitura e escrita nos registradores do GBT SCA.

O GBT contido na placa VDLB que foi utilizada para o desenvolvimento do sistema de teste já continha, na forma de fuses, a configuração mínima necessária para comunicação óptica. Quando alimentado, este já iniciava de uma forma que era possível escrever em seus registradores utilizando o miniDAQ, através do link GBT.

A forma alternativa era utilizar um dispositivo I2C para configurar o GBTx Master toda vez que o mesmo fosse reiniciado. No CERN, foram encomendados diversos módulos conversores USB-I2C os quais eram utilizados para tal configuração quando o usuário não queria queimar os fuses de sua placa.

Ao término do desenvolvimento do sistema de controle, no mesmo período em que se concluía a produção do protótipo da primeira Master Board, era necessário testar todo o sistema na *front-end* real, ao invés de utilizar somente a placa VLDB. No entanto, neste mesmo período, devido a problemas técnicos nos conversores USB-I2C disponíveis e pela

dificuldade de se encontrar solução comercial, nâo seria possível configurar o GBTx Master da recém-chegada Master Board para testar o sistema de controle.

Visando evitar a necessidade de esperar a chegada de novos conversores USB-I2C, foi desenvolvida uma solução alternativa para configuração do GBTx utilizando uma placa Arduino.

A solução compreendia uma placa Arduino Mega 2560, contendo um microcontrolador ATmega2560 e um módulo conversor de nível de 5V para 3.3V. A Figura 9.17 mostra o Arduino e o conversor de nível ligados à placa VLDB.

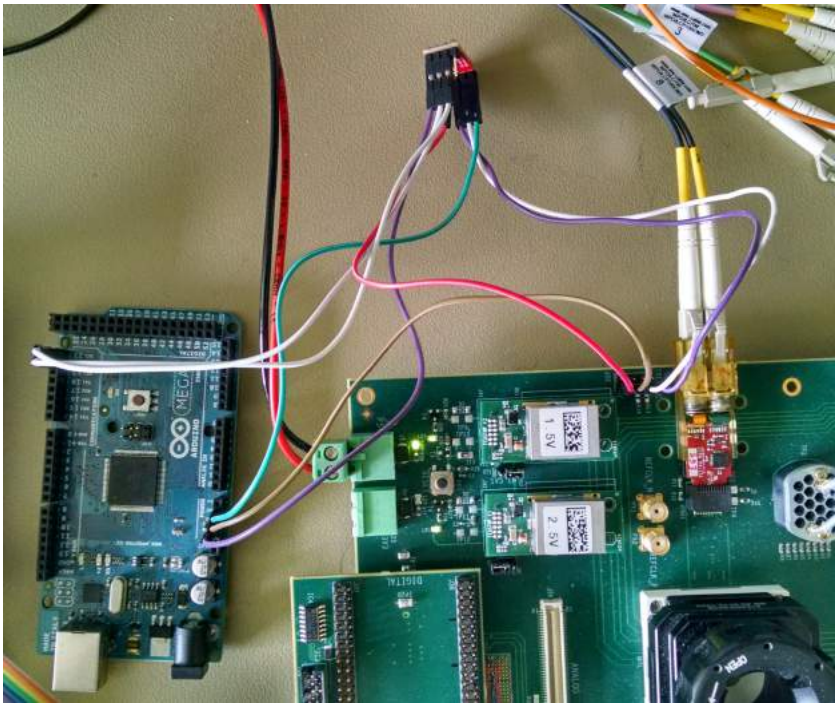

Figura 9.17: Arduino ligado à interface I2C da placa VLDB.

O Arduino contém integrado um conversor USB-UART, o qual era ligado à porta USB do computador. Através da IDE do Arduino, era possível enviar e receber bytes ao Arduino pela interface mencionada.

Foi criado um firmware para o Arduino que recebia comandos pela porta serial e os interpretava em comandos de leitura ou escrita na porta I2C do Arduino. Com isto era possível, através do computador e do Arduino, enviar comandos I2C para o chip GBTx e assim configura-lo.

O firmware aceita 4 comandos, listados abaixo:

`write <address> <value>`

Escreve o valor <value>no registrador <address>.

`read <address>`

Lê o registrador de endereço <address>e retorna o valor lido no formato "<address>: 0x<value>".

`rw <address> <value>`

Escreve do valor <value>no registrador <address>e executa uma leitura. Se o valor lido for o mesmo escrito, retorna " ok". Caso contrário retorna " <-XXX".

`read all`

Faz a leitura de todos os registradores dos endereços 0 a 365 e retorna os valores no mesmo formato do comando read.

Para configurar o GBTx, enviava-se ao Arduino um arquivo de texto com os comandos de escrita e leitura para todos os registradores de interesse. O firmware ignorava linhas iniciadas com o caractere "#" a fim de possibilitar comentários. Um exemplo da lista de comandos, contendo só os registradores 100, 101 e 102 seria desta forma:

```
rw 100 0xA1
#registradores 101 a 102 (comentario):
rw 101 0xB2
rw 102 0xC3
```

E após envio, teríamos a seguinte resposta numa situação em que houvesse erro na leitura do resgistrador 102:

```
rw 100 0xA1 ok
rw 101 0xB2 ok
rw 102 0xC3 <-XXX
```

O código completo do firmware do Arduino pode ser visto no Apêndice B. Todos os testes iniciais realizados na Master Board utilizaram a solução apresentada nesta seção.

Capítulo 10

Resultados

Ao longo do processo descrito nesta dissertação foram realizados diversos testes, principalmente durante a etapa de validação do MiniDAQ e depuração dos erros que ocorriam. Na seçâo 8.3 foram descritos alguns destes testes, com carater funcional, de validar tais funções para que fosse possível avançar para a etapa seguinte. Os respectivos resultados e conclusões destes testes podem ser encontrados em suas respectivas subseções e uma síntese destes testes se encontra nos items a seguir:

- O teste da camada física do MiniDAQ (Seção 8.3.1) visava testar os *transceivers* do FPGA e os componentes ópticos da AMC40 e por resultado pudemos concluir que o hardware estava 100% funcional e os problemas encontrados em testes posteriores eram puramente de software.

- O teste de injeção de dados em loopback (Seção 8.3.4) visava validar o acesso à AMC40 através do CCPC e o caminho de dados da AMC40 até o PC de aquisição. Como resultado os dados não chegavam de forma íntegra ao PC de destino. O resultado do teste anterior ajudou a limitar as possíveis causas do erro e foi concluído que era um problema no software que interpretava os dados no PC. O resultado final foi que a falha foi reportada e corrigida.

- Os testes de comunicação entre o MiniDAQ e um chip GBT real estão documentados na Seção 8.3.7. Trata-se de uma série de testes que visavam validar o uso do MiniDAQ para comunicação com um GBT. Após vários resultados negativos e um trabalho de depuração, foi concluído que os bits do GBT emulado no MiniDAQ estavam

invertidos, e que uma diferença de 1% na frequência do clock entre os módulos já era o suficiente para impedir que os seus receptores obtessem *lock*.

Nas seções a seguir veremos resultados adicionais de testes específicos que visam quantificar a performance do sinal óptico utilizado no hardware do sistema de teste.

10.1 Sinal óptico do MiniDAQ

O sinal óptico do MiniDAQ provém dos MiniPODs, que recebem o sinal elétrico do FPGA Stratix V contido na AMC40 e o traduz em óptico utilizando um emissor laser.

Os sinais foram analizados de três formas diferentes: utilizando osciloscópio do LAFEX no CBPF, utilizando osciloscópio de Nikhef e utilizando a ferramenta de transceiver integrada no FPGA Stratix V.

Adicionalmente, também foram realizados testes estatísticos para verificar se a taxa de erro de bits seria adequada ao projeto do sistema de testes.

10.1.1 Medida com osciloscópio de 4GHz

O LAFEX, no CBPF, possui a disposição um osciloscópio da Tektronix modelo DPO70404C, com largura de banda de 4GHz e taxa de amostragem de 25GS/s.

Antes de nos atermos ao resultado, no entanto, é importante analizarmos o resultado esperado para a configuração citada.

A entrada de um osciloscópio funciona como um filtro passa-baixa cujo sinal senoidal será atenuado em 3db quando sua frequência corresponder à largura de banda do osciloscópio, que é equivalente à frequência de corte do filtro. Pode-se encontrar mais detalhes sobre o assunto em [23]. Consideremos o sinal que se deseja medir, um sinal digital de 4.8GBps, como uma onda quadrada de 2.4GHz. Conforme explicado em [23], um sinal de 2.4GHz, tendo 60% da largura de banda do instrumento de medida (4GHz), sofreria atenuação de 12%. No entanto, isto se refere a sinais senoidais apenas. Se decompormos a onda quadrada de 2.4GHz em seus harmônicos, veremos que o segundo harmônico já estará bem além da frequência de corte do filtro de 4GHz. Por consequência, ao medir uma onda quadrada de 2.4GHz em um instrumento de 4GHz, espera-se medir algo bem próximo ao primeiro harmônico do sinal, que é uma senóide de 2.4GHz.

Na Figura 10.1 podemos ver o resultado da medida de gráfico de olho do link GBT do MiniDAQ transmitindo a 4.8Gbps, sendo medido a partir de uma placa conversora SFP-SMA, que recebe o sinal óptico do MiniDAQ.

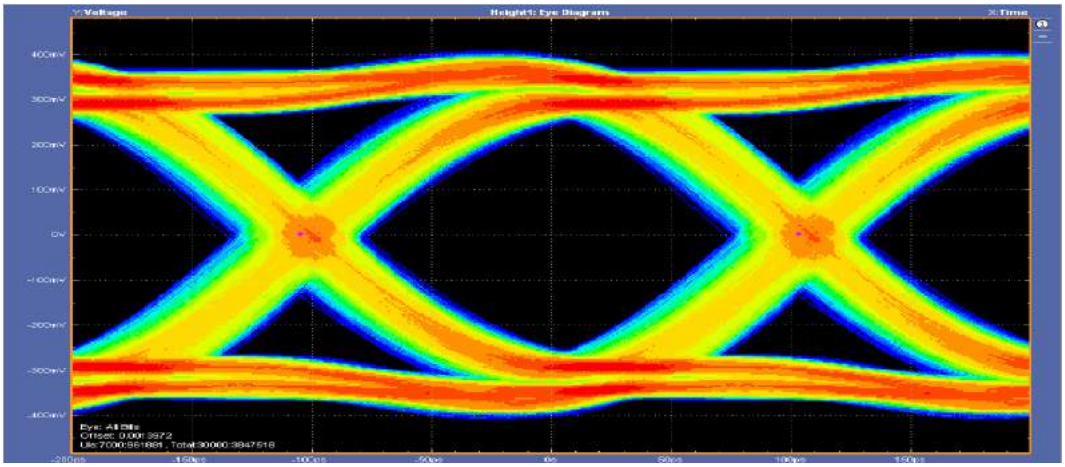

Figura 10.1: Gráfico de olho do sinal óptico do MiniDAQ a 4.8Gbps, medido a partir de instrumento de 4GHz.

Como esperado, o resultado foi bem próximo a uma senoide. Espera-se que o sinal real tenha um tempo de subida inferior ao resultado encontrado, uma vez que se houver harmônicos responsáveis pelo tempo de subida, estes certamente foram atenuados pelo filtro passa-baixa.

A Figura 10.2 mostra a medição do jitter pelo método de erro de intervalo de tempo (TIE) do sinal apresentado na Figura 10.1.

A Figura 10.3 mostra a medição do gráfico de Bathtub do sinal do MiniDAQ, apresentando a taxa de erro de bit por intervalo de medição. O resultado aponta que uma região acima de 60% do gráfico apresentaria taxa de erro de bit abaixo de 10^{-12}.

10.1.2 Medida alternativa com osciloscópio de 20GHz

O laboratório do instituto de Nikhef possui um osciloscópio com largura de banda superior ao utilizado anteriormente e que foi usado para gerar o gráfico de olho do setup do MiniDAQ que lá se encontra instalado. O gráfico de olho citado se encontra na Figura 10.4.

A partir deste resultado foi possível confirmar as deduções realizadas com o instrumento de 4GHz.

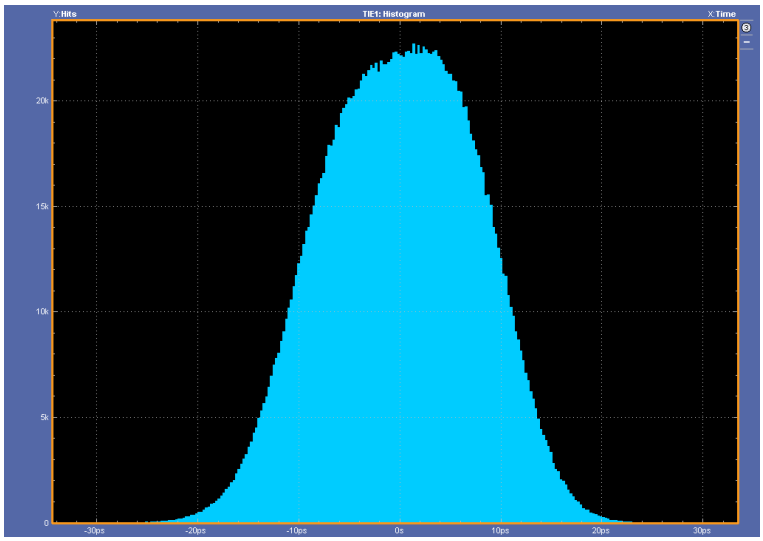

Figura 10.2: Medição do jitter por erro de intervalo de tempo.

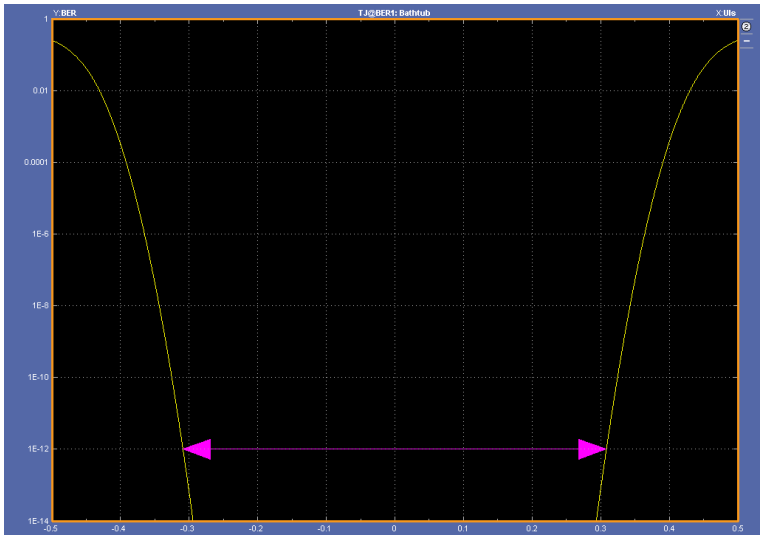

Figura 10.3: Medição do gráfico de bathtube do sinal do MiniDAQ. O eixo vertical representa a taxa de erro de bit e o eixo horizontal representa o ponto de amostragem em referência ao mesmo eixo horizontal do gráfico de olho.

10.1.3 Validação do Sinal com o Altera Transceiver Toolkit

O Transceiver Toolkit é uma ferramenta da Altera com diversas funções de disgnóstico dos transceptores integrados em algumas famílias de FPGA da Altera. A utilização se dá através do instanciamento de um bloco próprio da Altera e acesso ao FPGA via JTAG.

Utilizando o Altera Transceiver Toolkit, um dos canais foi configurado para gerar

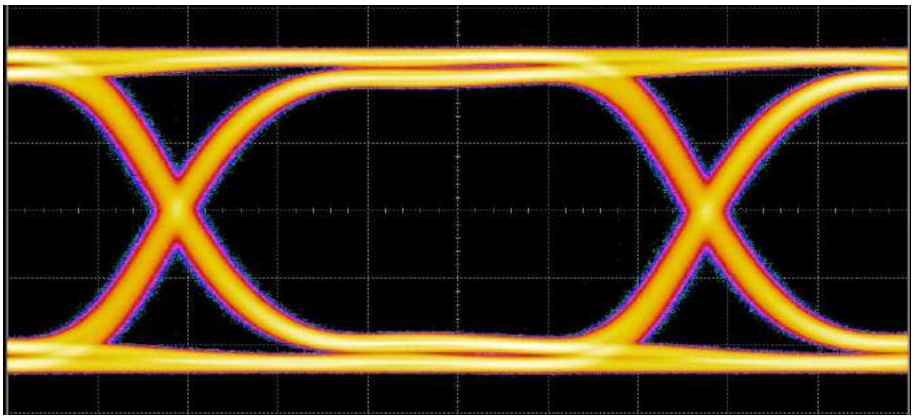

Figura 10.4: Medida do gráfico de olho do link óptico do MiniDAQ situado em Nikhef a partir de osciloscópio de largura de banda de 20GHz.

um padrão de bits pseudo-aleatório utilizando o algoritmo PRBS7, e este foi ligado em loopback de volta na AMC40. O teste foi executado por 30 minutos, tendo transmitido $8,64 \times 10^{12}$ bits sem ocorrência de erro.

Ainda utilizando o Altera Transceiver Toolkit, também foi gerado um gráfico de olho a partir da mesma transmissão de dados pseudo-aleatórios PRBS7 do qual a partir da largura abertura se pode concluir que o sinal é de qualidade suficiente para a aplicação proposta. Na Figura 10.5 é possível ver o resultado.

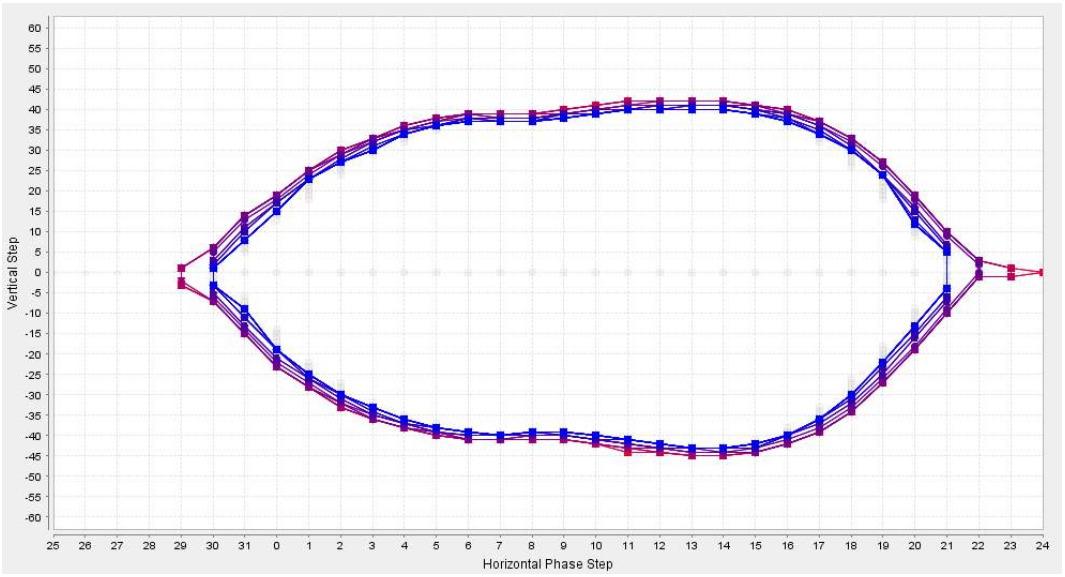

Figura 10.5: Gráfico de olho gerado pelos transceivers da AMC40 a partir da transmissão de uma sequência PRBS7 pseudo-aleatória em loopback.

10.2 Biblioteca e Painéis WinCC

O sistema de controle, junto com a biblioteca fwlbSCIFI e os painéis criados foram testados com um protótipo da Master Board logo após a fase de desenvolvimento. Os testes tiveram carater funcional, tendo como principal objetivo a verificação do funcionamento dos protótipos da Master Board. O grupo do CBPF até o presente momento ainda não tem uma placa com um GBT real para realização de testes mais extensos.

Durante o período de algumas horas em que se sucederam os testes das Master Boards, o sistema de contole foi utilizado para configuração dos GBTx de dados e teste do SCA Master. Em nenhum momento se percebeu perda do travamento do header no GBT, e todos os comandos de leitura e escrita nos registradores dos GBTx e GBT SCA da Master Board funcionaram corretamente.

O tempo de leitura e escrita nos registradores I2C pelo GBT SCA, no entanto, se mostrou elevado. Foi necessário de 4 a 5 minutos para configuração de todos os 365 registradores de cada GBTx de dados. Isto ocorreu porque a biblioteca precisa, até o presente momento, enviar um comando de escrita e leitura para cada registrador individual.

A operação de escrita e leitura nos 365 registradores do GBTx Master utilizando a ferramenta com Arduino descrita na Seção 9.10 levou um tempo inferior a 10s.

Todas as funcionalidades implementadas no sistema do controle funcionaram corretamente.

Capítulo 11

Conclusão

A presente dissertação apresentou o processo de validação do hardware MiniDAQ para utilização no sistema de teste do detector de fibras cintilantes do LHCb, assim como o desenvolvimento de um sistema de controle supervisório SCADA para configuração da *front-end* a ser testada.

O grupo responsável pelo sistema de teste do SciFi foi o primeiro grupo da colaboração a utilizar o MiniDAQ e por isto exerceu um importante trabalho de depuração que, em conjunto com diferentes desenvolvedores do hardware, software e firmware do MiniDAQ, pôde contribuir com um feedback prático a fim de ajudá-los a chegar no lançamento de uma versão estável do sistema. Esta contribuição se estende aos novos módulos de aquisição PCIe40, uma vez que o mesmo compartilhará partes do firmware e software da TELL40 e MiniDAQ.

Os testes de comunicação entre o MiniDAQ e um chip GBTx real permitiram verificar o problema que impedia a comunicação entre os dispositivos, e o relatório gerado permitiu à colaboração definir a versão oficial a ser usada para emulação do GBT nos módulos de aquisição do experimento.

A máquina virtual criada para boot do MiniDAQ se mostrou estável ao longo dos dois anos de uso, tendo sido adotada por diferentes grupos da colaboração até a presente data.

Com os resultados dos testes realizados no MiniDAQ, este se mostrou uma plataforma confiável tanto para aquisição de dados quanto para controle da *front-end* a ser testada. Também foi possível concluir que o MiniDAQ atende aos requisitos do sistema de

teste.

O desenvolvimento do sistema de controle em WinCC serviu não somente como base para o controle do sistema de teste do SciFi, que era o objetivo primário, mas também como ponto de partida para o sistema de controle da *front-end* a ser utilizado no experimento. Este sistema foi crucial para os testes dos protótipos da Master Board e era de fato a única ferramenta capaz de se comunicar com todos os dispositivos presentes na mesma. A biblioteca fwlbSCIFI e os painéis criados para o sistema de controle permitiram ao grupo de Nikhef, responsável pelo desenvolvimento da Master Board, a realização dos testes de validação dos primeiros protótipos da Master Board.

O objetivo principal do presente trabalho era o estabelecimento de um setup apto a ser utilizado no sistema de teste do SciFi assim como meios que permitissem o controle do mesmo via software. Este objetivo foi concluído com sucesso e permitirá ao futuro desenvolvedor do sistema de teste do SciFi realizar a conclusão de um sistema de controle sólido quando todos os componentes de hardware que irão compor o sistema estiverem finalizados.

Referências Bibliográficas

[1] L. Evans and P. Bryant. Lhc machine. *JINST*, 3:S08001, 2008.

[2] ATLAS Collaboration, G. Asd, et al. The atlas experiment at the cern large hadron collider. *JINST*, 3:S08003, 2008.

[3] CMS Collaboration, S. Chatrchyan, et al. The cms experiment at the cern lhc. *JINST*, 3:S08004, 2008.

[4] LHCb Collaboration, A. Augusto Alves Jr, et al. The lhcb detector at the lhc. *JINST*, 3:S08005, 2008.

[5] ALICE Collaboration, K. Aamodt, et al. The alice experiment at the cern lhc. *JINST*, 3:S08002, 2008.

[6] TOTEM Collaboration, G. Anelli, et al. The totem experiment at the cern large hadron collider. *JINST*, 3:S08007, 2008.

[7] LHCf Collaboration, O. Adriani, et al. The lhcf detector at the cern large hadron collider. *JINST*, 3:S08006, 2008.

[8] LHCb Collaboration et al. Lhcb rich. Technical report, European Organization for Nuclear Research - CERN, 2000.

[9] LHCb Collaboration et al. Lhcb calorimeters. Technical report, European Organization for Nuclear Research - CERN, 2000.

[10] LHCb Collaboration et al. Lhcb muon system. Technical report, European Organization for Nuclear Research - CERN, 2001.

[11] Cian O'Luanaigh. New technologies for the High-Luminosity LHC. Nov 2015.

[12] Letter of Intent for the LHCb Upgrade. Technical Report CERN-LHCC-2011-001. LHCC-I-018, CERN, Geneva, Mar 2011.

[13] I Bediaga, J M De Miranda, F Ferreira Rodrigues, J Magnin, A Massafferri, I Nasteva, A C dos Reis, S Amato, K Carvalho Akiba, L De Paula, O Francisco, M Gandelman, A Gomes, J H Lopes, J M Otalora Goicochea, E Polycarpo, M S Rangel, B Souza De Paula, D Vieira, C Gobel, J Molina Rodriguez, and et al. Framework TDR for the LHCb Upgrade: Technical Design Report. Technical Report CERN-LHCC-2012-007. LHCb-TDR-12, CERN, Geneva, Apr 2012.

[14] LHCb Trigger and Online Upgrade Technical Design Report. Technical Report CERN-LHCC-2014-016. LHCB-TDR-016, CERN, Geneva, May 2014.

[15] Neus Lopez March and Matthias Karacson. Radiation studies for the LHCb tracker upgrade. Technical Report LHCb-PUB-2014-022. CERN-LHCb-PUB-2014-022. LHCb-INT-2013-003, CERN, Geneva, Feb 2014.

[16] Gbt project home page. Disponivel em: https://espace.cern.ch/GBT-Project/default.aspx. [Acessado em 13/Maio/2016].

[17] Jose Mazorra De Cos. PACIFIC : The readout ASIC for the SciFi Tracker planned for the upgrade of the LHCb detector. Sep 2015.

[18] A Affolder, B Allongue, G Blanchot, F Faccio, C Fuentes, A Greenall, and S Michelis. Dc-dc converters with reduced mass for trackers at the hl-lhc. *Journal of Instrumentation*, 6(11):C11035, 2011.

[19] G Vouters, F Alessio, J Cachemiche, S Cap, C Drancourt, P Durante, P Duval, L Fournier, M Jevaud, F Hachon, J Mendez, F Rethore, and S. T'Jampens. LHCb upgrade MiniDAQ Handbook. Technical report, CERN, Geneva, Nov 2013.

[20] Cairo Caplan, Andre Massafferri, Federico Alessio, Clara Gaspar, Richard Jacobsson, and Ken Wyllie. Desenvolvimento e validacao do bloco logico programavel generico para comunicacao entre os ASIC GBT-SCA da eletronica de Front-End para o upgrade do experimento LHCb. Technical Report CBPF-NT-002/16, CBPF, Rio de Janeiro, Jan 2016.

[21] A Gabrielli, S Bonacini, K Kloukinas, A Marchioro, P Moreira, A Ranieri, and D De Robertis. The GBT-SCA, a radiation tolerant ASIC for detector control applications in SLHC experiments. 2009.

[22] Giulia Papotti. *Architectural Studies of a Radiation-hard Transceiver ASIC in 0.13 um CMOS for Digital Optical Links in High Energy Physics Applications*. PhD thesis, Universita Degli Studi di Parma, 2007.

[23] Tektronix. Understanding Oscilloscope Bandwidth, Rise Time and Signal Fidelity. Technical Report 55W-18024-3.

Appendices

Apêndice A

Código da Biblioteca fwlbSCIFI

```
// ==========================================================================
// ============================== WRITE TO SCA ==============================

// Writes to a device controlled by GBT-SCA I2C port

int write_to_SCA(string dpSOL40link, string dp, string data) {

  string rw;
  string gbt_name, sca_name, ch_name;
  string gbt_num, sca_num, ch_num, reg_address;
  int reg_len, addr_len;
  dyn_string command;
  dyn_string split;
  string length, datafield, command_string;
  int query;

 // GBT_Master0.SCA_Master.Device.Channel.Controller.Registers.CRA.data

  dp = dpSubStr( dp, DPSUB_SYS_DP_EL);
```

```
split = strsplit(dp, ".");

  gbt_name = split[1];
  sca_name = gbt_name +"."+ split[2];
  ch_name =  sca_name +"."+ split[3] +"."+ split[4] +"."+ split[5];

  dp = substr(dp, 0, strpos(dp,".data", 0)); // cut off the ".data"

  if (!dpExists(dp + ".wr_cmd")) {
    DebugTN("Read Only DP: "+dp);
    return -1;
  }

  query = dpGet(gbt_name+".Device.config.address", gbt_num,
                sca_name+".Device.config.address", sca_num,
                ch_name +".config.address", ch_num,
                //ch_name +".Device.config.I2C_addr_len", addr_len,
                dp      +".wr_cmd", reg_address,
                dp      +".length",  reg_len);
  if (query!= 0 ) {
    DebugTN("Error on the dpGet of write_I2C_device() function");
    return query;
  }

  // Check and correct if length of the data is wrong
  if ( (strlen(data)/2) != reg_len) {
    if ( (strlen(data)/2) > reg_len) {
                                          //DebugTN("reg_len: " + reg_len);
                                          //DebugTN("data: " + data);
      data = substr(data, strlen(data)-reg_len*2, reg_len*2);
                                          //DebugTN("data: " + data);
    } else {
      while (strlen(data)<reg_len) data = "0" + data;
    }
```

```
}

// set total address
sprintf( length, "%02X" , (reg_len + addr_len) );


rw = "10"; // "10" = write command, "11" = read command

command[1] = gbt_num + sca_num + ch_num + reg_address;

command[2] = "00" + length + "0000";

datafield = data;

bool need4 = false;

if (strlen(datafield)> 8) {
  command[3] = substr(datafield, strlen(datafield)-8, 8);
  command[4] = substr(datafield, 0, strlen(datafield)-8);
  need4 = true;
} else{
  command[3] = datafield;
}

//padding with left zeroes
while (strlen(command[3])<8) command[3] = "0" + command[3];

command_string = command[1]+command[2]+command[3];

if ( need4 ) {
  while (strlen(command[4])<8) command[4] = "0" + command[4];
  command_string = command_string+command[4];
}
```

```
 // DebugTN(command_string);

  SOL40SCA_SetCommand(dpSOL40link, command_string );
  SOL40SCA_Go(dpSOL40link);

  return 0;
}
// ============================================================================
// ============================== READ FROM SCA ===============================

// Writes to a device controlled by GBT-SCA I2C port

int read_from_SCA(string dpSOL40link, string dp) {

  string databack;
  string rw;
  string gbt_name, sca_name, ch_name;
  string gbt_num, sca_num, ch_num, reg_address;
  int reg_len, addr_len;
  dyn_string command;
  dyn_string split;
  string length, datafield, command_string;
  int query;

 // GBT_Master0.SCA_Master.Device.Channel.Controller.Registers.CRA.data

  dp = dpSubStr( dp, DPSUB_SYS_DP_EL);

  split = strsplit(dp, ".");

  gbt_name = split[1];
  sca_name = gbt_name +"."+ split[2];
  ch_name =  sca_name +"."+ split[3] +"."+ split[4] +"."+ split[5];
```

```
 dp = substr(dp, 0, strpos(dp,".data", 0)); // cut off the ".data"

 if (!dpExists(dp + ".rd_cmd")) {
   DebugTN("Write Only DP: "+dp);
   return -1;
 }

 query = dpGet(gbt_name+".Device.config.address", gbt_num,
               sca_name+".Device.config.address", sca_num,
               ch_name +".config.address", ch_num,
               //ch_name +".Device.config.I2C_addr_len", addr_len,
               dp      +".rd_cmd", reg_address,
               dp      +".length",  reg_len);
 if (query!= 0 ) {
   DebugTN("Error on the dpGet of write_I2C_device() function");
   return query;
 }

 // set total address
 sprintf( length, "%02X" , reg_len );

 //rw = "10"; // "10" = write command, "11" = read command

 command[1] = gbt_num + sca_num + ch_num + reg_address;

 command[2] = "00" + length + "0000";

 command_string = command[1] + command[2];

// DebugTN(command_string);

 SOL40SCA_SetCommand(dpSOL40link, command_string );
 SOL40SCA_Go(dpSOL40link);
 SOL40SCA_SetReplyAddress(dpSOL40link, command[1]);
```

```
  SOL40SCA_NewReply(dpSOL40link);

  databack = SOL40SCA_GetReply(dpSOL40link);

  databack = substr(databack, strlen(databack)-reg_len*2, reg_len*2);

  dpSetWait( dp+".data" , databack );
  return 0;
}
// ============================================================================
// ============================ WRITE TO I2C DEVICE ============================

// Writes to a device controlled by GBT-SCA I2C port

write_I2C_device(string dpSOL40link, string dp, string data) {

  string rw;
  string gbt_name, sca_name, ch_name;
  string gbt_num, sca_num, ch_num, reg_address;
  int reg_len, addr_len;
  dyn_string command;
  dyn_string split;
  string length, datafield, command_string;
  int query;

  dp = dpSubStr( dp, DPSUB_SYS_DP_EL);

  split = strsplit(dp, ".");

  gbt_name = split[1];
  sca_name = gbt_name +"."+ split[2];
  ch_name =  sca_name +"."+ split[3];

  dp = substr(dp, 0, strpos(dp,".data", 0)); // cut off the ".data"
```

```
query = dpGet(gbt_name+".Device.config.address", gbt_num,
              sca_name+".Device.config.address", sca_num,
              ch_name +".Device.config.address", ch_num,
              ch_name +".Device.config.address_len", addr_len, //length of internal
              dp      +".address", reg_address,
              dp      +".length",  reg_len);
if (query!= 0 ) {
  DebugTN("Error on the dpGet of write_I2C_device() function");
  return query;
}

// Check and correct if length of the address is wrong
if ( strlen(reg_address) != (addr_len*2)) {
  if ( strlen(reg_address) > (addr_len*2)) {
    reg_address = substr(reg_address, strlen(reg_address)-addr_len-1, addr_len*2);
  } else {
    while (strlen(reg_address)<(addr_len*2)) reg_address = "0" + reg_address;
  }
}

// Check and correct if length of the data is wrong
if ( (strlen(data)/2) != reg_len) {
  if ( (strlen(data)/2) > reg_len) {
                                          //DebugTN("reg_len: " + reg_len);
                                          //DebugTN("data: " + data);
    data = substr(data, strlen(data)-reg_len*2, reg_len*2);
                                          //DebugTN("data: " + data);
  } else {
    while (strlen(data)<reg_len) data = "0" + data;
  }
}

// set total length
```

```
 sprintf( length, "%02X" , (reg_len + addr_len) );
// DebugTN();

 rw = "90"; // "90" = write command, "91" = read command

 command[1] = gbt_num + sca_num + ch_num + rw;

 command[2] = "00" + length + "0000";

 datafield = data + reg_address;
                                        //DebugTN("data: " + data);
                                        //DebugTN("reg_address: " + reg_address);
                                        //DebugTN("datafield: " + datafield);
 bool need4 = false;

 if (strlen(datafield)> 8) {
   command[3] = substr(datafield, strlen(datafield)-8, 8);
   command[4] = substr(datafield, 0, strlen(datafield)-8);
   need4 = true;
 } else{
   command[3] = datafield;
 }

 //padding with left zeroes
 while (strlen(command[3])<8) command[3] = "0" + command[3];

 command_string = command[1]+command[2]+command[3];

 if ( need4 ) {
   while (strlen(command[4])<8) command[4] = "0" + command[4];
   command_string = command_string+command[4];
 }

 SOL40SCA_SetCommand(dpSOL40link, command_string );
```

```
  SOL40SCA_Go(dpSOL40link);

}

// ================================================================================
// ============================ READ FROM I2C DEVICE ==========================

// Reads from a device controlled by GBT-SCA I2C port

read_I2C_device(string dpSOL40link, string dp) {

  string databack;
  string rw;
  string gbt_name, sca_name, ch_name;
  string gbt_num, sca_num, ch_num, reg_address;
  int reg_len, addr_len;
  dyn_string command;
  dyn_string split;
  string length, datafield, command_string;
  int query;

  dp = dpSubStr( dp, DPSUB_SYS_DP_EL);

  split = strsplit(dp, ".");

  gbt_name = split[1];
  sca_name = gbt_name +"."+ split[2];
  ch_name =  sca_name +"."+ split[3];

  dp = substr(dp, 0, strpos(dp,".data", 0)); // cut off the ".data"
  query = dpGet(gbt_name+".Device.config.address", gbt_num,
                sca_name+".Device.config.address", sca_num,
                ch_name +".Device.config.address", ch_num,
                ch_name +".Device.config.address_len", addr_len, //length of internal
```

```
              dp      +".address", reg_address,
              dp      +".length",  reg_len);
if (query!= 0 ) {
  DebugTN("Error on the dpGet of write_I2C_device() function");
  return query;
}

sprintf( length, "%02X" , addr_len );

// Check and correct if length of the address is wrong
if ( (strlen(reg_address)/2) != addr_len) {
  if ( (strlen(reg_address)/2) > addr_len) {
    reg_address = substr(reg_address, strlen(reg_address)-addr_len-1, addr_len*2);
  } else {
    while (strlen(reg_address)<addr_len) reg_address = "0" + reg_address;
  }
}

rw = "90"; // "90" = write command, "91" = read command

sprintf( length, "%02X" , addr_len );

command[1] = gbt_num + sca_num + ch_num + rw;

command[2] = "00" + length + "0000";

command[3] = reg_address;
while (strlen(command[3])<8) command[3] = "0" + command[3];

command_string = command[1]+command[2]+command[3];

SOL40SCA_SetCommand(dpSOL40link, command_string );
SOL40SCA_Go(dpSOL40link);
```

```
  rw = "91"; // "90" = write command, "91" = read command

  sprintf( length, "%02X" , reg_len );

  command[1] = gbt_num + sca_num + ch_num + rw;

  command[2] = "00" + length + "0000";

  command_string = command[1]+command[2];

  SOL40SCA_SetCommand(dpSOL40link, command_string );
  SOL40SCA_Go(dpSOL40link);
  SOL40SCA_SetReplyAddress(dpSOL40link, command[1]);
  SOL40SCA_NewReply(dpSOL40link);

  databack = SOL40SCA_GetReply(dpSOL40link);

  databack = substr(databack, strlen(databack)-reg_len*2, reg_len*2);

  dpSetWait( dp+".data" , databack );
}
// ============================================================================
// ======================== SET REPLY ADDRESS SOL40-SCA =======================

// Send get reply command to SOL40-SCA (set bit 2 of control register)
SOL40SCA_SetReplyAddress(string dpSOL40link, string address) {

  string reg;
  anytype data, mask;
  dyn_dyn_anytype wdata, rdata, mask;
  dyn_int rstatus;

  reg = dpSOL40link+".Registers_SOL40.sca_reply_addr";
```

```
  data = address;
 // mask = "FFFFFFFF";

  wdata[1] = (dyn_anytype) fwHw_convertHexToByte(data);
  //mask[1] = (dyn_anytype) fwHw_convertHexToByte(mask);
  if (isFunctionDefined("fwCcpc_writeRead")) {
    fwCcpc_writeRead(makeDynString(reg), wdata, mask, rdata, rstatus);
  } else {
    DebugTN("funcao fwCcpc_writeRead not defined.");
  }
 // DebugTN("SetReplyAddress: "+address);

 // DebugTN("Set Reply Address: " + address);
}

// ============================================================================
// ======================== READ REPLY FROM SOL40-SCA =======================

// Send get reply command to SOL40-SCA (set bit 2 of control register)
SOL40SCA_NewReply(string dpSOL40link) {

  string reg;
  anytype data, mask;
  dyn_dyn_anytype wdata, rdata, mask;
  dyn_int rstatus;

  reg = dpSOL40link+".Registers_SOL40.sca_control";

  data = "00000002";
  mask = "00000002";

  wdata[1] = (dyn_anytype) fwHw_convertHexToByte(data);
  mask[1] = (dyn_anytype) fwHw_convertHexToByte(mask);
  if (isFunctionDefined("fwCcpc_writeRead")) {
```

```
    fwCcpc_writeRead(makeDynString(reg), wdata, mask, rdata, rstatus);
  } else {
    DebugTN("funcao fwCcpc_writeRead not defined.");
  }
  //DebugTN("New Reply");
}

// ============================================================================
// ========================== GET REPLY FROM SOL40-SCA ========================

// Send GO command to SOL40-SCA (set bit 1 of control register)
string SOL40SCA_GetReply(string dpSOL40link) {

 // string result;
  dyn_dyn_char data;
  dyn_string   registers, result;
  dyn_int      status;
  int          i;

  dynAppend(registers, dpSOL40link+".Registers_SOL40.sca_reply_hdr0");
  dynAppend(registers, dpSOL40link+".Registers_SOL40.sca_reply_hdr1");
  dynAppend(registers, dpSOL40link+".Registers_SOL40.sca_reply_data0");

  if (fwCcpc_read(registers, data, status))   // read data from registers
     for(i = 1; i <= dynlen(registers); i++)
     {
       result[i] = fwCcpc_convertByteToHex(data[i]);
       // setValue("", "text", fwCcpc_convertByteToHex(data[i]) );
      //  DebugTN("polling sempre?!");
     }
  else
     DebugTN("Error found!");
  DebugTN("Get Reply: " + result[1] +" "+ result[2] +" "+ result[3]);
  return result[3];
```

```
}

// ============================================================================
// ========================== SEND "GO" TO SOL40-SCA ==========================

// Send GO command to SOL40-SCA (set bit 1 of control register)
SOL40SCA_Go(string dpSOL40link) {

  string reg;
  anytype data, mask;
  dyn_dyn_anytype wdata, rdata, mask;
  dyn_int rstatus;

  reg = dpSOL40link+".Registers_SOL40.sca_control";

  data = "00000001";
  mask = "00000001";

    wdata[1] = (dyn_anytype) fwHw_convertHexToByte(data);
    mask[1] = (dyn_anytype) fwHw_convertHexToByte(mask);
    if (isFunctionDefined("fwCcpc_writeRead")) {
      fwCcpc_writeRead(makeDynString(reg), wdata, mask, rdata, rstatus);
    } else {
      DebugTN("funcao fwCcpc_writeRead not defined.");
    }
 // DebugTN("Go");
}

// ============================================================================
// ======================== SET COMMAND TO SOL40-SCA ==========================
SOL40SCA_SetCommand(string dpSOL40link, string command) {

  int k, i; // only for the loop
  string reg_data; //temp
```

```
dyn_string   registers;
dyn_dyn_char data, rdata, mask;
dyn_int      status;
string debug_string;

//Set the 6 registers from where the SOL40-SCA block pulls the command
dynAppend(registers, dpSOL40link+".Registers_SOL40.sca_cmd_hdr0");
dynAppend(registers, dpSOL40link+".Registers_SOL40.sca_cmd_hdr1");
dynAppend(registers, dpSOL40link+".Registers_SOL40.sca_cmd_data0");
dynAppend(registers, dpSOL40link+".Registers_SOL40.sca_cmd_data1");
dynAppend(registers, dpSOL40link+".Registers_SOL40.sca_cmd_data2");
dynAppend(registers, dpSOL40link+".Registers_SOL40.sca_cmd_data3");

//remove white spaces and ()
command = strltrim(command, " ()");

//padding with zeroes:
while (strlen(command)<48) command = command + "0";

debug_string = "SetCommand: ";

// set the 6 data values from the command string
for (i=0 ; i<=5 ; i++) {
  reg_data = "";
  for (k=(0+i*8) ; k<=(7+i*8) ; k++ ) {
    reg_data = reg_data + command[k];
  }
  if (fwHw_isHex(reg_data))
    dynAppend(data,fwCcpc_convertHexToByte(reg_data));
  else
    DebugTN("Value to "+i+"° register is not a valid HEX.");
  debug_string += " "+reg_data;
}
```

```
  DebugTN(debug_string);

  // Write to the 6 command registers
  if (isFunctionDefined("fwCcpc_writeRead")) {
    if (fwCcpc_writeRead(registers, data, mask, rdata, status))
       for(i = 1; i <= dynlen(status); i++){
        //  DebugTN("Status: " + status[i]);
        //  DebugTN("Data: " + fwCcpc_convertByteToHex(rdata[i]));
       }
    else
       DebugTN("Error found!");
  }
  else {
    DebugTN("Function fwCcpc_writeRead not defined.");
  }

}
// ============================================================================
```

Apêndice B

Firmware do Conversor USB-I2C para Arduino

```
// Mauricio Féo (m.feo@cern.ch)
// Code to write to GBT's registers using an Arduino
// (Remember to use a level converter!)
//
// Usage:
// Open a serial terminal, connect to Arduino COM's port and type
//
// write <address> <value>
// read <address>
//
// 0x in front of the value will make it hexadecimal
// The read returns:
// <address>: 0x<value>

#include <Wire.h>

#define GBT_I2C_address 0x1

void setup() {
  Serial.begin(9600);
```

```
  Wire.begin();
}

int reg_address;
byte reg_data;
String cmd;

//=======================================
void loop() {
  if (Serial.available()) {
    cmd = Serial.readStringUntil(' ');

    if (cmd.charAt(0)=="#") {
      cmd = Serial.readStringUntil('\n');
      Serial.println("#");
    }else
    if (cmd=="rw") {
      cmd = Serial.readStringUntil(' ');
      if (cmd.substring(0,2)=="0x") reg_address = HexToInt(cmd);
        else reg_address = cmd.toInt();
      cmd = Serial.readStringUntil('\n');
      if (cmd.substring(0,2)=="0x") reg_data = HexToInt(cmd);
        else reg_data = cmd.toInt();
      GBT_read_write(reg_address, reg_data);

    }else

    if (cmd=="write") {
      cmd = Serial.readStringUntil(' ');
      if (cmd.substring(0,2)=="0x") reg_address = HexToInt(cmd);
        else reg_address = cmd.toInt();
      cmd = Serial.readStringUntil('\n');
      if (cmd.substring(0,2)=="0x") reg_data = HexToInt(cmd);
        else reg_data = cmd.toInt();
```

```
      GBT_write(reg_address, reg_data);

    }else

      if (cmd=="read") {
      cmd = Serial.readStringUntil('\n');
      if (cmd.substring(0,3)=="all") GBT_readAll();
      else {
        if (cmd.substring(0,2)=="0x") reg_address = HexToInt(cmd);
        else reg_address = cmd.toInt();
        reg_data = GBT_read(reg_address);
        Serial.print(reg_address);
        Serial.print(": 0x");
        Serial.println(reg_data, HEX);
      }
    }
  }
}

//=======================================
int GBT_readAll()
{
  byte data;
  int k;
  for (k=0; k<=365; k++) {
    data = GBT_read(k);

    Serial.print(k);
    Serial.print(": 0x");
    Serial.println(data, HEX);
  }
}

//=======================================
```

```
int HexToInt(String str)
{
  char chr[6];
  str.substring(2).toCharArray(chr, 5);
  return (int) strtol(chr, 0, 16);
}
//=======================================

void GBT_read_write(int address, byte data) {
  byte data_read;

  GBT_write(address, data);

  data_read = GBT_read(address);

  if (data == data_read) {
    Serial.print(address);
    Serial.println(" ok");
  } else {
    Serial.print(address);
    Serial.println(" <-XXX");
  }
}
//=======================================

byte GBT_read(int address) {
  byte data;
  Wire.beginTransmission(GBT_I2C_address);
  Wire.write(lowByte(address));
  Wire.write(highByte(address));
  Wire.endTransmission(false);
  Wire.requestFrom(GBT_I2C_address, 1); //read 1 byte
  data = Wire.read();
  return data;
```

```
}
//=========================================

void GBT_write(int address, byte data) {
  Wire.beginTransmission(GBT_I2C_address);
  Wire.write(lowByte(address));
  Wire.write(highByte(address));
  Wire.write(data);
  Wire.endTransmission();

}
```

Apêndice C

Histórico de Modificações Desde a Criação da Máquina Virtual de Boot

```
Maurício Féo Rivello - m.feo@cern.ch
This is a log of everything done in the Virtual Machine created to serve
 as a boot server for the AMC40's CCPC.

================ CREATING VIRTUAL MACHINE ================================

Create Virtual Machine
OS: Linux 2.6/2.x (32bit)
Name: BootServer
1GB RAM
Disk: BootServer.vmdk (VMDK, Dynamically allocated, 20GB max)

-In the VM Settings -> System -> Processor: mark Enable PAE/NX

================ INSTALLING LINUX =========================================

Install SL6.2 from SL-62-i386-2012-02-16-LiveCD.iso

hostname: BootServer
root pass: root123
```

```
username: user
password: user123

=================== CONFIGURING VITUAL MACHINE ========================

-Add user to sudoers group: as su:
echo 'user ALL=(ALL) ALL' >> /etc/sudoers

-On VM Menu: Devices -> Insert Guest Additions CD
-Install prerequisites:
sudo yum install kernel-devel-2.6.32-220.4.1.el6.i686 gcc make
-Install Guest Additions from CD's autorun

sudo usermod -aG vboxsf user

-Shutdown the VM. Go to settings:
-General->advanced: Shared Clipboard: Bidirectional
-Shared folder: Add a new. We will call it "vmshared". Mark "auto-mount".
-Network->Adapter 1-> Attached to-> Bridge Adapter: Ethernet card. (for booting)
-Network->Adapter 2-> Attached to-> NAT (for internet, etc.)
-Turn on the VM.

*vmshared will be mounted now on /media/sf_vmshared at statup. Useful for data file tran
*Eth0 will be connected to your PC ethernet card, should use it for the boot server.
*Eth1 will receive from the network your pc is connected to.
*Shared Clipboard, it means you can copy/paste from you VM to the pc now!

================== SETTING UP NETWORK ==================================

-Config eth0 (boot), HWADDR must be the one from eth0.
sudo gedit /etc/sysconfig/network-scripts/ifcfg-eth0
DEVICE=eth0
HWADDR="08:00:27:FC:7D:1C"
BOOTPROTO=none
```

```
ONBOOT=yes
NETWORK=192.168.1.1
NETMASK=255.255.255.0
IPADDR=192.168.1.1
USERCTL=no

sudo ifup eth0

-Config eth1 (used for internet, etc.), HWADDR must be the one from eth1:
sudo gedit /etc/sysconfig/network-scripts/ifcfg-eth1
DEVICE="eth1"
HWADDR="08:00:27:72:7A:2A"
NM_CONTROLLED="yes"
ONBOOT="no"

sudo ifup eth1

-Disable SELinux:
sudo gedit /etc/selinux/config
-set SELINUX=disabled

-Set firewall to allow everything from eth0
sudo iptables -I INPUT -i eth0 -j ACCEPT
sudo /sbin/service iptables save

-(choose connections "system eth0" and "system eth1")

================== SETTING UP BOOT SERVER ===================================

-Download https://lbredmine.cern.ch/attachments/download/37/SL62.tarz and
- http://marwww.in2p3.fr/~duval/SL62-CCPC.tar.bz2 to the shared folder.

-Installing TFTP, DHCP and NFS servers
sudo yum install syslinux tftp-server dhcp nfs-utils nfs-utils-lib nfs4-acl-tools
```

```
-Config DHCP
sudo gedit /etc/dhcp/dhcpd.conf
-Add the following, but change the MAC to yours CCPC (find in comments):
#================ dhcpd.conf start =================#
#
# DHCP Server Configuration file.
# see /usr/share/doc/dhcp*/dhcpd.conf.sample
# see 'man 5 dhcpd.conf'
#
not authoritative;
ddns-update-style none;
deny unknown-clients;
option ip-forwarding false; # No IP forwarding
option mask-supplier false; # Don't respond to ICMP Mask req

option vendor-encapsulated-options 01:04:00:00:00:00:ff;
option vendor-class-identifier "PXEClient";

# Use this to send dhcp log messages to a different log file (you also
# have to hack syslog.conf to complete the redirection).
log-facility local6;
allow booting;
allow bootp;
class "pxeclients" {
match if substring(option vendor-class-identifier, 0, 9) = "PXEClient";
next-server 192.168.1.1;
filename "linux-install/pxelinux.0";
}

# thats the private network NIC
shared-network eth0 {
subnet 192.168.1.0 netmask 255.255.255.0 {
```

```
# set the DHCP address allocation parameters
default-lease-time 21600;
max-lease-time 43200;
next-server 192.168.1.1; # myself
# the CCPC IP address allocation and pxe boot code
host CCPC10 {
  hardware ethernet 00:E0:4B:43:7A:E0; # must use your CCPC MAC
fixed-address 192.168.1.51;
option host-name "CCPC10";
filename "linux-install/pxelinux.0";
}
} # subnet
} # eth0
#================ dhcpd.conf end =================#

- restrict DHCP daemon to only listen to interface eth0 (:
sudo gedit /etc/sysconfig/dhcpd
-set DHCPDARGS="eth0"


-Config TFTP:

sudo gedit /etc/xinetd.d/tftp
-set server_args = -s /tftpboot
-set disable = no

sudo mkdir -p /tftpboot/linux-install
sudo cp /usr/share/syslinux/pxelinux.0 /tftpboot/linux-install

sudo tar zxvf /media/sf_vmshared/SL62.tarz -C /tftpboot/linux-install/

sudo mkdir -p /tftpboot/linux-install/pxelinux.cfg/
sudo gedit /tftpboot/linux-install/pxelinux.cfg/default
-Add the following:
```

```
default sl62
label sl62
kernel sl62/vmlinuz
append initrd=sl62/initramfs root=nfs:192.168.1.1:/diskless/i386/SL62/root rdshell rw

sudo mkdir -p /diskless/i386/SL62
sudo tar xvjf /media/sf_vmshared/SL62-CCPC.tar.bz2 -C /diskless/i386/SL62/

sudo gedit /etc/exports
-add the following:
/diskless/i386/SL62 *(rw,sync,no_root_squash,insecure)

sudo /sbin/chkconfig --level 345 xinetd on
sudo /sbin/chkconfig --level 345 tftp on
sudo /sbin/chkconfig --level 345 nfs on

-reboot

-Start all services:

sudo /etc/init.d/xinetd restart
sudo service dhcpd restart
sudo service nfs restart

-Monitor with:
sudo tail -f /var/log/messages

-Plug the board and power it up!

-Wait for around a minute and try:
ssh -X upgrade@192.168.1.51
-password: upgrade4CCPC

-It worked! Logged in into the CCPC!
```