ISSN 0102-745X

Notas Técnicas

CBPF-NT-004/25
setembro 2025

Comunicação entre processador HPS e FPGA em um chip SoC Cyclone V

Danilo dos Santos Cardoso e Herman Pessoa Lima Jnior

MINISTÉRIO DA
CIÊNCIA, TECNOLOGIA
E INOVAÇÃO

dx.doi.org/10.7437/NT2236-7640/2025.03.004
Notas Técnicas, v. 15, n. 3, p. 1–18, 2025

Comunicação entre processador HPS e FPGA em um chip SoC Cyclone V

Communication between HPS processor and FPGA in a Cyclone V SoC chip

Danilo dos Santos Cardoso
dsantoscardoso@outlook.com
Prof. Herman Pessoa Lima Júnior
Centro Brasileiro de Pesquisas Físicas - CBPF
Rua Dr. Xavier Sigaud, 150
Urca - Rio de Janeiro - RJ - Brasil - CEP:22290-180
Submetido em 21 de novembro de 2023 *Aceito em 25 de novembro de 2023*

Resumo: Esta nota apresenta um resumo que inclui conceitos básicos e avançados no que se refere à interação entre HPS (*Hard Processor System*) e FPGA *(Field Programmable Gate Array*) em um único chip (SoC – *System-on-a-chip*), mais precisamente um Cyclone V do fabricante Intel. São abordados conceitos das arquiteturas individuais, mapeamento em memória, sistema operacional Linux embarcado, ferramentas computacionais, criação de componentes IP no FPGA, e os comandos para executar a comunicação entre os dois dispositivos.

Palavras-Chave: Interconexão, Mapeamento em Memória, Linux embarcado, componentes IP.

Abstract: This note presents a summary that includes basic and advanced concepts regarding the interaction between HPS and FPGA on a single chip (SoC), more precisely an Intel Cyclone V. Concepts of the individual architectures, Memory-Mapping, Embedded Linux operating system, computational tools, creation of IP components on the FPGA, and the commands to execute the communication between these two devices.

Keywords: Interconnect, Memory-Mapping Components, Embedded Linux, IP components.

1. INTRODUÇÃO

Avançar na criação de chips eletrônicos menores e mais sofisticados sempre foi um dos alvos das grandes indústrias de microeletrônica. Segundo Gordon Moore, conseguir 'amontoar' [3] cada vez mais componentes nos circuitos integrados começa a chegar a um limite [2]. No caso de processadores, de acordo com [4], a utilização de mais de um núcleo simultaneamente pode estender a lei de Moore no futuro.

Combinar funcionalidades também se tornou um produto desta indústria. Os sistemas-em-um-chip, ou SoC (*System-on-a-chip*) são combinações de funcionalidades eletrônicas em um único chip formando um sistema, como por exemplo: microcontroladores com *wi-fi* (PIC32MZ-W1), CPU com GPU (*Apple* M1), microcontrolador com sensores e *bluetooth low-energy* (*Intel Curie*), e o conjunto FPGA e Processador, também chamado de FPSoC [7].

Este trabalho aborda a tecnologia FPSoC, que está disponível há pouco mais de uma década. A Xilinx, criadora do FPGA, e comprada pela AMD, lançou o Zynq-7000; e a Altera, posteriormente comprada pela Intel, lançou o Cyclone V SoC. Ambas apresentaram os produtos ao mercado em meados de 2012 [1] [6].

Conforme [5], FPGAs SoC fornecem performances maiores do que soluções baseadas em FPGA tradicional e processadores como chips separados. O motivo principal é que a combinação de processador e FPGA em um único chip fornece interconexões dedicadas para a interligação destes diferentes sistemas, que podem utilizar desde simples sinais de controle, a completos barramentos. E esta é uma das grandes vantagens deste dispositivo SoC, sua construção em um único chip consegue fornecer baixa latência e alta taxa de transferência nos barramentos entres seus componentes [5].

Neste trabalho utilizaremos um SoC da Intel, o Cyclone V SE. Ele é composto por um FPGA e um HPS (*Hard Processor System*). Este HPS por si só já é um sistema completo, sendo composto basicamente por um processador Dual-Core ARM[1] Cortex-A9 de 925 MHz, periféricos (GPIO, SPI, I2C, ethernet e outros) e interfaces de memória (controladores de memórias SDRAM e Flash).

O escopo deste trabalho é o de utilizar esta interação entre os componentes através das 'pontes' entre HPS e FPGA. O SoC mencionado é parte integrante da placa eletrônica de desenvolvimento da *Terasic* modelo DE10-Nano. Nela será utilizado o sistema operacional (SO) Linux executado pelo processador ARM. A interface de programação será o *Visual Studio Code Insiders* em um computador com sistema operacional *Windows*, comunicando-se com a placa de desenvolvimento via ethernet. A principal linguagem de programação do processador para criação de programas no nível do usuário será o *Python*.

[1] *ARM é uma família de processadores com arquitetura RISC desenvolvida pela empresa britânica ARM Holdings.*

2. AS ARQUITETURAS DE BARRAMENTOS DO FPGA E DO PROCESSADOR

Tanto o FPGA quanto o HPS possuem individualmente barramentos de comunicação entre seus blocos de funções internos. Por serem componentes distintos, de fabricantes e desenvolvedores diferentes, naturalmente possuem protocolos proprietários, e consequentemente implementações particulares.

A arquitetura AMBA (*Advanced Microcontroller Bus Architecture*) foi criada pela ARM *Holdings* e é direcionada para uso em SoC, com padrões abertos e gratuitos, visando a conexão e o gerenciamento de blocos funcionais. A AMBA possui algumas subdivisões, sendo as utilizadas neste SoC os seguintes protocolos: *Advanced eXtensible Interface* (AXI), *Advanced High-Performance Bus* (AHB), e a *Advanced Peripheral Bus* (APB) [10]. Neste SoC, entre estes três, somente o AXI é utilizado para conexões entre o HPS e o FPGA, os demais são utilizados somente no HPS.

A Avalon é no nome dado à família de interfaces[2] do FPGA Intel. Ela é dedicada a transmissões de altas taxas de dados, leitura e escrita de registros e memórias, além de controle de componentes *off-chip* [19]. Devido ao fato de o FPGA ser um componente onde se projeta *hardware* via HDLs (*Hardware Description Languages*), a interface Avalon pode ser utilizada para comunicação entre os diferentes componentes criados. Por exemplo, pode-se criar uma estrutura no FPGA com um processador Nios II, *on-chip* RAM, *Display Port* e DSPs, e interligá-los via interfaces Avalon. Este exemplo utilizou IPs (*Intelectual Property*), que são alguns componentes de *firmware* prontos fornecidos pela Altera (nem todos gratuitos); porém, o usuário pode utilizar esta interface para fazer a comunicação entre os mais diversos componentes criados. Assim como a AMBA, a Avalon é um padrão aberto, sem custo, e possui suas subdivisões. As interfaces que compõem a família Avalon serão mais trabalhadas nesta abordagem, e são as seguintes:

- *Streaming Interface* (Avalon-ST): para maiores quantidades de dados e transmissões multiplexadas;
- *Memory Mapped Interface* (Avalon-MM): usada em leituras e escritas típicas para conexões Mestre-Escravo (*Host-Agent*)[3], neste caso utiliza-se de endereço numérico similar ao de acesso a memórias;
- *Tri-State Interface* (Avalon-TC): indicadas para periféricos *off-chip*, permite por exemplo o compartilhamento de pinos entre periféricos para se utilizar da multiplexação;
- *Conduit Interface*: interface que acondiciona sinais individuais ou grupos de sinais que não se enquadram nos demais tipos de interfaces.

As demais interfaces são autoexplicativas: *Interrupt Interface*, *Clock Interface* e *Reset Interface*.

Estas nomenclaturas serão novamente utilizadas na geração das interconexões no *software Quartus*.

3. OS BARRAMENTOS DE INTERCONEXÃO

Para se utilizar um SoC de alta performance em velocidade, conhecer sua estrutura de interconexão é primordial. No caso do Cyclone V SE, as interconexões entre FPGA e o HPS (Avalon e AXI, respectivamente) são realizadas por barramentos denominados 'pontes'. Como cada componente possui seu protocolo específico, a 'tradução' é feita pelo sistema de interconexão, não sendo necessária a intervenção do usuário.

A Figura 1 apresenta, de forma simplificada, as conexões internas do HPS, e as pontes entre HPS e FPGA. As setas indicam a relação entre os blocos, partindo do mestre para o escravo. O diagrama é simplificado, alguns blocos não estão presentes.

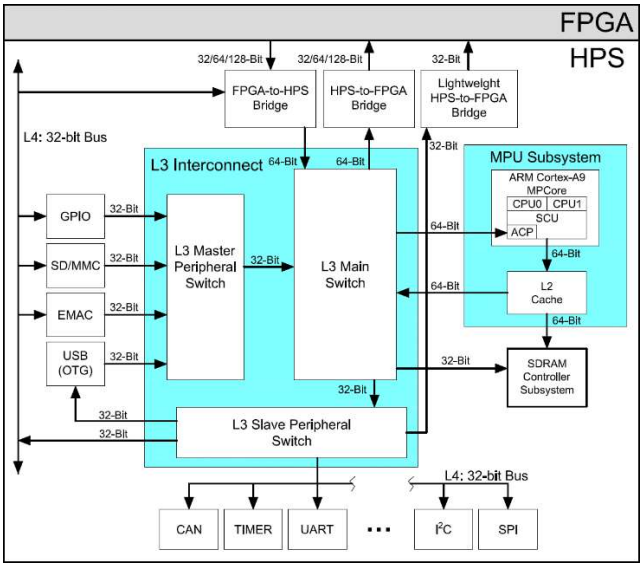

Figura 1: Diagrama de blocos simplificado do dispositivo Cyclone V com os barramentos de interconexão.
Fonte: Adaptada de [9] p.8-2.

O HPS possui uma unidade de processamento (MPU) com dois núcleos ARM Cortex-A9 de 32 bits. O *Snoop*

[2] Interface: é a descrição usada pela Intel quando se trata de Avalon.

[3] A versão utilizada do *software Quartus* da Intel foi a de 2018. Sua documentação ainda trazia os termos 'Mestre' e 'Escravo' para relacionar dois componentes que se comunicam. Alguns documentos da Intel mais recentes que foram pesquisados substituíram estes termos por '*Host*' e '*Agent*', respectivamente. Outras *big techs* também estão fazendo alterações semelhantes. Para evitar incompatibilidade de versões, os documentos utilizados foram os de 2018 e, portanto, a terminologia 'Mestre-Escravo' ainda foi utilizada neste trabalho.

Control Unit (SCU) garante a coerência de conteúdo entre as memórias caches L1, L2 e a SDRAM [7]. A porta ACP (*Accelerator Coherency Port*) é uma interface que permite que periféricos mestres com níveis superiores a L2 (incluindo o FPGA), acessem as memórias cache L1 e L2 mantendo coerência de dados com os processadores e o SCU [9] p.10-27.

A Interconexão do Sistema, chamada de *System Interconnect*, é a estrutura composta pela L3 *Interconnect* e pelo barramento de nível 4 L4, presentes na Figura 1. Ela implementa uma arquitetura que suporta múltiplas transações simultâneas entre mestres e escravos com alta taxa de transferência, compreendendo os periféricos e a MPU [8].

O barramento L4 é o responsável pelo acesso aos periféricos do HPS, como SPI, I2C, UART, EMAC (ethernet), entre outros. Há ainda uma interface que não está presente na Figura 1, a *FPGA-to-SDRAM*, que permite ao FPGA acessar diretamente a memória SDRAM sem a coerência de memória [7].

3.1. As pontes de interconexão

Como retratado na Figura 1, os acessos entre HPS e FPGA ocorrem via L3 *Main Switch* e *L3 Slave Peripheral Switch*. Existem três interfaces para este fim, duas de alta velocidade e uma de menor performance; são as pontes previamente comentadas e são mapeadas em memória [8]:

- *FPGA-to-HPS bridge* (F2H): barramento de alta performance com largura de dados configurável entre 32, 64 ou 128 bits, possibilitando ao FPGA comandar transações aos escravos no HPS e acessar a interface de coerência de memória (ACP).
- *HPS-to-FPGA bridge* (H2F): idêntica à ponte FPGA-*to*-HPS, mas com o comando no sentido inverso, permitindo ao HPS comandar escravos no FPGA.
- *Lightweight HPS-to-FPGA bridge* (LWB): interface fixa de 32 bits, que também permite ao HPS comandar escravos no FPGA. Possui menor performance perante as outras duas, sendo útil para acessar registros de controle e *status* de periféricos lógicos.

Como comparativo, segundo [7], a interface H2F configurada com a largura de 128 bits de dados e o FPGA sendo executado com um *clock* de 150 MHz, consegue alcançar taxa de transferência próxima de 1100 MB/s. Este valor foi alcançado em determinadas configurações e em aplicação específica tipo *baremetal*. Nas mesmas condições, a LWB (que possui a largura de dados fixa em 32 bits) chega a 18% deste valor.

4. MAPEAMENTO EM MEMÓRIA

As pontes entre os componentes são acessadas através do conceito de Mapeamento-em-Memória (*Memory-Mapping* (MM)). Este método permite que os periféricos ou as próprias pontes, sejam acessadas pelo mesmo barramento de acesso à memória [12] (em determinados níveis entre L1 e L4). Assim, para o processador, as instruções de acesso à memória e as de acesso aos dispositivos com estas características são as mesmas. Esta simplificação permite a criação de um *hardware* mais simples e rápido. Para acessar algum periférico, o processador realiza uma operação de leitura ou escrita no endereço em questão.

A Tabela 1 fornece alguns endereços dos periféricos do dispositivo Cyclone V utilizado do ponto de vista do HPS. Foram suprimidos vários outros endereços. O primeiro endereço, 0xC0000000, representa o endereço 3 Giga, abaixo deste valor estão os endereços utilizados pelo HPS para memória SDRAM e *Boot*. Portanto, para acessar algum periférico no FPGA via LWB, o endereço inicial é o 0xFF200000. Este endereço representa o primeiro periférico configurado da LWB, os endereços dos vários outros periféricos - se criados - são *offset* deste endereço, e dependem de cada componente criado. Da mesma maneira, acessa-se a *On-chip RAM* a partir do endereço 0xFFFF0000, e a H2F a partir do endereço 0xC0000000. O endereço de cada periférico contemplado em alguma das pontes pode ser visualizado ou determinado no *software Quartus*, via programa do usuário. As pontes direcionadas nos sentidos do FPGA para o HPS (F2H), e do FPGA para a SDRAM não serão tratadas neste trabalho.

Tabela 1: Mapa parcial de endereços do Cyclone V. Fonte: [9] p2-15 a p2-20.

Interface	Nome	End. Inicial	Tamanho
FPGA SLAVES	*FPGA Slaves Accessed via H2F*	0xC0000000	960 MB
LWFPGA SLAVES	*FPGA slaves accessed with LWB*	0xFF200000	2 MB
EMAC0	*Ethernet MAC 0*	0xFF700000	8 KB
SDMMC	*SD/MMC*	0xFF704000	4 KB
USB0	*USB 2.0 OTG 0 ctrl registers*	0xFFB00000	256 KB
UART0	*UART 0*	0xFFC02000	4 KB
OCRAM	*On-chip RAM*	0xFFFF0000	64 KB

5. SISTEMA OPERACIONAL DO HPS E SUAS FERRAMENTAS COMPUTACIONAIS

A plataforma eletrônica de desenvolvimento utilizada neste trabalho é a DE10-Nano, da empresa *Terasic*, que possui um SoC FPGA Cyclone V. A Figura 2 contém um diagrama com os principais recursos utilizados neste

trabalho, evidenciando o componente do SoC ao qual estão ligados diretamente.

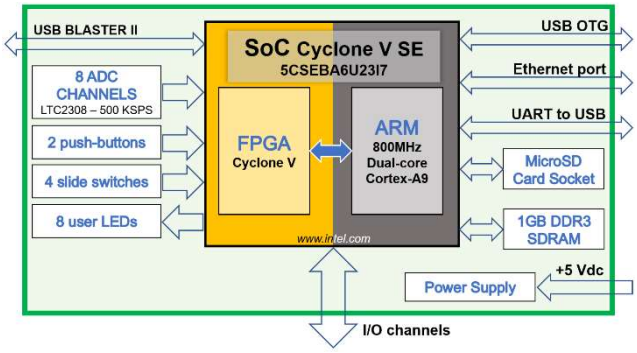

Figura 2: Diagrama da placa eletrônica DE10-Nano da *Terasic*, destacando os principais blocos utilizados neste trabalho. Fonte: Adaptado de [22].

Criar programas exclusivos a partir do zero para uma plataforma como esta, não é vantajoso quando se pretende utilizar os diversos recursos presentes na placa eletrônica, como a conexão ethernet, por exemplo, que possui complexos protocolos e *firmware* muito específicos.

Contudo, o processador ARM incorporado ao SoC é capaz de executar um sistema operacional (SO) Linux. Entre as vantagens no uso de um sistema operacional embarcado pode-se destacar: o gerenciamento das conexões seriais, do cartão de memória e da memória RAM; depuração *online;* possibilidade de execução de diversos *softwares* de terceiros compatíveis com o SO; e a existência de bibliotecas de códigos e documentações. Porém, uma das dificuldades encontradas no uso de um sistema operacional, é a de lidar com as características não-determinísticas, como o uso de memória, as interrupções, e principalmente quanto ao gerenciamento de tempo de um processador relativamente lento, já que não se pode definir categoricamente seu tempo de execução.

A Intel e a ARM *Holdings* são colaboradoras de plataformas e distribuições Linux como a *Armstrong Distribution* e o projeto *Yocto*[4] [14].

5.1. A distribuição Linux rsYocto

Com o projeto *Yocto* é possível criar distribuições Linux customizadas para sistema embarcados. Uma distribuição gratuita personalizada para a placa DE10-Nano é a chamada *rsYocto*[5]. Esta distribuição Linux é direcionada para uso com os SoCs da Cyclone V e Arria 10 [13], ambos da Intel. A rsYocto implementa um kernel Linux com uma variedade de componentes que simplificam o processo de interação entre o FPGA e o HPS.

Um dos destaques da rsYocto é a acessibilidade aos componentes da placa DE10-Nano através de um navegador de internet. O servidor *Apache* está incluso na distribuição, e assim é possível utilizar-se de uma interface de usuário mais amigável através de computadores ou *smartphones*. Para este tipo de aplicação o desenvolvedor já deixou pré-instalado o *framework Django*, que possibilita a utilização da linguagem *Python* na programação do HPS quando se utiliza uma aplicação *web*. A versão do rsYocto utilizada durante os testes foi a 1.041.

5.2. Instalando o sistema operacional

Abaixo é fornecido um resumo dos programas, cabos e equipamentos necessários para a instalação da placa e do sistema operacional (SO):

- Kit de desenvolvimento DE10-Nano, da *Terasic*, ou outro modelo similar disponível em [15];
- Cartão de memória *microSD* com capacidade entre 2 e 8 GB (máximo), e respectivo adaptador para inserir no computador;
- Cabos ethernet;
- Roteador com a função DHCP habilitada;
- Computador;
- *Software* Terminal, como o *Tera Term* ou similar.
- *Software* para gerar a imagem no cartão de memória, como o *Win32 Disk Imager* ou *Rufus*;

O roteiro mais detalhado para a instalação do sistema operacional está em [16], mas os principais passos estão descritos a seguir. Foi utilizado um computador com sistema operacional *Windows 10*.

As versões do rsYocto estão disponibilizadas no *GitHub* em [15], deve-se procurar pelo nome da placa utilizada da *Terasic* no sufixo das opções e executar o *download* do arquivo *.zip* correspondente. É necessário um *software* para gerar a imagem no cartão de memória *microSD* que conterá o sistema operacional. Foi utilizado o *Win32 Disk Imager*. Com ele é possível gerar a imagem no cartão a partir do arquivo *.img* extraído do arquivo baixado. Após a criação da imagem (processo que dura cerca de 40 min em um computador com processador Intel *Core* i7-7500U a 2.70 GHz e 20 GB de memória RAM), o cartão está pronto para ser inserido no *slot* da placa ainda desenergizada.

Antes de energizar a placa é importante conferir o modo de configuração do FPGA e ligar os cabos de comunicação

[4] O Yocto Project é um projeto de colaboração de código aberto que ajuda os desenvolvedores a criarem sistemas personalizados baseados em Linux, independente da arquitetura de hardware. Acessível em: https://www.yoctoproject.org

[5] RSYocto: as iniciais RS representam o nome do criador desta distribuição Linux para Cyclone V, seu nome é Robin Sebastian. Disponível em [13].

necessários. O modo de configuração do FPGA deve ser selecionado através do seletor MSEL, que se encontra no lado superior da placa, denominado SW10. Olhando o seletor de frente (com a ordem numérica crescente) as chaves devem ficar nas posições 10100X (MSEL [0:4]). Os seletores para cima significam '0', e para baixo significam '1'. Há uma pequena incoerência neste ponto, pois a identificação no componente SW10 indica 'ON' para cima, ao mesmo tempo que o texto impresso na placa indica '0' para cima. O correto é a indicação impressa na superfície da placa, ou seja, o '0'; a posição da última chave é indiferente. Esta configuração permite que o *u-boot* (responsável pela inicialização) e o Linux, alterem a configuração do FPGA; sendo assim, o arquivo de configuração do FPGA que estiver no cartão de memória será utilizado durante a inicialização para programar o FPGA.

Neste ponto, os cabos ethernet e USB / UART (conector J4 na placa) devem estar conectados. Esta conexão USB será utilizada apenas para configurar a conexão ethernet, a autenticação SSH[6], e para fazer o login inicial no Linux. É interessante que a rede esteja conectada à internet para aproveitar algumas funcionalidades do Linux, como atualizar o horário do SO, por exemplo. Neste trabalho, a placa foi conectada via cabo a um roteador que estava conectado à internet e ao computador, conforme ilustrado na Figura 3. Esta figura também contém a interligação do J13 com o computador, que futuramente pode ser utilizada para programar o FPGA.

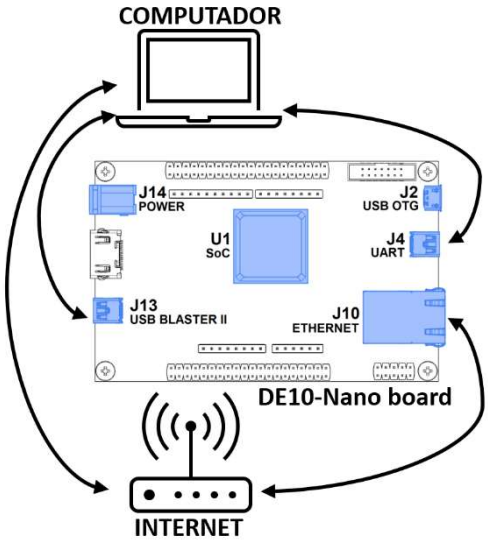

Figura 3: Interligações recomendadas.

Deve-se configurar a porta serial no *software* Terminal para os valores: 115200 N8 (ASCII) com CR/LF. Neste momento a placa pode ser energizada. Como resultado, alguns LEDs verdes na placa, ao lado do conector ethernet, irão acender, e o *software* terminal deverá mostrar as etapas da inicialização. Quando não houver mais movimentação no *software* terminal, o seguinte texto deverá aparecer: 'cyclone5 login:', deve-se digitar a palavra de *login*: 'root', e em seguida o *password*: 'eit'. O texto 'root@clyclone5: ~ #' deverá ser exibido indicando que o *login* foi executado corretamente e o terminal Linux está pronto para receber novos comandos. O sistema operacional está instalado e pronto para ser utilizado. A partir deste ponto já é possível utilizar os comandos tipo *Shell* descritos na seção 7.1.

5.3. Configurando o acesso via Ethernet

Para obter acesso à placa via ethernet é preciso descobrir o endereço IPv4 designado a ela pelo servidor DHCP do roteador. Considerando que o *boot* foi realizado utilizando o *software* terminal, pode-se utilizar o comando Linux *ifconfig* ou buscar no histórico do *boot*, pois o endereço é visível durante o processo de inicialização. Com o comando *ifconfig* do Linux é possível alterar o endereço IPv4, da seguinte forma (onde os valores reais devem substituir os campos < >):

```
ifconfig <nome_da_interface>
<endereço_ip_da_placa> netmask
<máscara_de_sub_rede>
```

Como exemplo, para alterar somente o IPv4 deve-se executar:

```
ifconfig eth0 192.168.100.33
```

O nome 'eth0' da interface foi verificado nos dados fornecidos pelo comando *ifconfig* digitado previamente. Para outras alterações, como deixar o IPv4 estático, basta utilizar os comandos próprios do Linux.

Para acessar o sistema operacional da placa via ethernet, a partir de um computador com *Windows* (ou Linux), é necessário liberar a conexão ethernet com a criação da autenticação SSH para o endereço IPv4 adotado. O Linux (neste caso, o SO embarcado na placa) exige este tipo de criptografia para a conexão via ethernet. O próximo passo não é obrigatório, mas com ele se consegue o acesso ao Linux da placa via comandos tipo *Shell*, que é uma interface muito simples. Para isto utiliza-se o *prompt de comando do Windows* (botão Iniciar e em seguida o comando 'cmd'). De forma semelhante ao comando de alteração do endereço IP, basta digitar o comando abaixo, confirmando em seguinda com 'yes', e após a solicitação, o *password* 'eit'.

```
ssh root@<endereço_ip_da_placa>
```

Neste ponto é possível utilizar os comandos *Shell* descritos na seção 7 através do *software* terminal ou o *Command* do *Windows* via conexão ethernet.

[6] SSH: *Secure Socket Shell* é um protocolo de segurança que utiliza criptografia para troca de arquivos entre cliente e servidor.

5.4. Instalando o Visual Studio Code Insiders

É possível obter um acesso mais dinâmico ao sistema da placa como um todo, englobando o sistema operacional e suas funcionalidades, programas do usuário para o HPS, realizar depurações online, entre outras facilidades. Para isto utiliza-se o *Visual Studio Code Insiders* (VSCI) *online* com a placa, via ethernet, e diretamente do *Windows*. De acordo com SEBASTIAN [16] somente o VSCI permite o acesso remoto via SSH para dispositivos com ARMv7A rodando Linux. Isto foi relatado em 2020 e, portanto, pode ser que novas versões do *Visual Studio* (diferentes da *Insiders)* sejam capazes de fornecer esta interface. Neste trabalho utilizamos o VSCI, como recomendado por SEBASTIAN.

O VSCI permite o acesso ao Linux apenas com chave gerada pelo *SSH-Keygen*; devendo a chave ser renovada a cada troca de placa, cartão *microSD* ou endereço IP [16]. A sequência de comandos abaixo deve ser inserida no *prompt de Comando* do *Windows* 10:

```
ssh-keygen -t rsa -b 4096
```

Confirma-se com *Enter* as duas solicitações seguintes sem digitar nada. O *Windows* armazena a chave no diretório: C:\Users\<nome_do_usuário>\.ssh. Feito isto, é preciso utilizar o endereço IPv4 da placa para indicar 'quem' utilizará esta chave gerada:

```
SET REMOTEHOST=root@<endereço_ip_da_placa>
```

Na sequência copia-se para o Linux da placa a chave SSH gerada, digitando todo o texto abaixo. Não é necessária nenhuma modificação:

```
scp %USERPROFILE%\.ssh\id_rsa.pub
%REMOTEHOST%:~/tmp.pub
```

A senha é: 'eit'. Após isto, é necessário autenticar e ativar a chave SSH gerada inserindo todo o texto abaixo. Mais uma vez não é necessária nenhuma modificação. A senha será solicitada novamente:

```
ssh %REMOTEHOST% "mkdir -p ~/.ssh && chmod 700
~/.ssh && cat ~/tmp.pub >>
~/.ssh/authorized_keys && chmod 600
~/.ssh/authorized_keys && rm -f ~/tmp.pub"
```

5.5. Preparando o Visual Studio Code Insiders

O *Visual Studio Code Insiders* (VSCI) é a ferramenta utilizada para o acesso remoto à placa, sendo possível visualizar desde o sistema operacional até os programas específicos do usuário no que tange ao HPS; o programa do FPGA ainda é o próprio da Intel, o *Quartus Prime*.

Mesmo com o VSCI já instalado no computador, ainda é necessário instalar algumas extensões. Na barra lateral esquerda da interface do VSCI, em 'Extensões', pesquisa-se pela extensão '*Remote Development*' da *Microsoft*. Após sua instalação deve-se reiniciar o VSCI. Após este processo, ainda na barra lateral esquerda, clica-se no novo ícone 'Explorador Remoto', próximo a 'SSH TARGETS'. Haverá um ícone de engrenagem. Ao clicar nele, uma lista suspensa, tipo *dropdown*, irá aparecer, e esta contém o endereço do computador onde foi criada a chave SSH seguida do arquivo 'config':

```
C:\Users\<nome_do_usuário>\.ssh\config
```

Selecionando esta linha da lista, o arquivo será aberto e nele deve-se configurar o endereço IPv4 da placa, que deverá ficar semelhante a:

```
Host rsYocto
    ForwardAgent yes
    HostName <endereço_ip_da_placa>
    User root
    IdentityFile ~/.ssh/id_rsa
```

Salva-se as alterações. O nome do *host* rsYocto irá aparecer no menu à esquerda. Clica-se neste nome com o botão direito do *mouse* e seleciona-se '*Connect Host in current Window*'. O VSCI irá instalar seu próprio servidor na placa.

Em casos de atualizações do *software* VSCI, pode ocorrer algum erro durante esta instalação do servidor. Um procedimento inicial que pode resolver o problema é o de apagar a pasta já instalada no diretório do Linux da placa, que foi criada pelo próprio VSCI para este fim. Também é necessário atualizar todas as extensões no VSCI para corresponder à nova versão instalada. Para apagar a pasta utiliza-se o comando abaixo em um terminal e a porta UART da placa.

```
root@cyclone5:~# rm -fr .vscode-server-insiders/
```

Finalizada a instalação do VSCI, na barra à esquerda, na aba do ícone 'Explorador', haverá o botão '*Open Folder*'; com ele é possível definir algumas pastas que ficarão com o acesso mais fácil, através de uma lista de diretórios que estará visível em 'Explorador Remoto'. Após selecionado o ícone, aparecerá uma lista suspensa onde a primeira opção será o diretório '/*home*/*root*/', está é a pasta principal da aplicação, onde estão os programas de exemplo e é o local que deverá ser utilizado pelo usuário. Para acessar pastas anteriores a este diretório é preciso escolher o texto '..' na lista suspensa. Desta forma, consegue-se chegar até o diretório raiz do Linux embarcado. Como já salientado, todas as pastas escolhidas neste processo ('*Open Folder*') ficarão acessíveis de forma mais direta no ícone 'Explorador Remoto', e a pasta selecionada no processo será a raiz da seleção. Somente será possível acessar as pastas subsequentes. Para acessar uma pasta anterior à selecionada, todo o processo deste parágrafo deverá de ser repetido buscando a pasta desejada.

5.6. A ferramenta de desenvolvimento remoto para linguagem Python

Outra extensão deve ser instalada para facilitar a depuração de códigos Python que serão executados pelo HPS. Esta extensão fornece recursos visuais para enriquecer o desenvolvimento de códigos nesta linguagem, como o *IntelliSense* (sugestão durante a digitação), navegação e formatação de código, comandos e variáveis com cores diferentes, entre outros, além da funcionalidade de depuração de código.

De forma semelhante à instalação do '*Remote Development*', deve-se buscar pela extensão Python da *Microsoft* e instalá-la utilizando o botão '*Install in SSH: rsYocto*'. Neste ponto, todas as ferramentas computacionais necessárias, visando o HPS, estão instaladas e funcionais.

6. CONFIGURANDO O FPGA COM OS PROGRAMAS QUARTUS PRIME E PLATFORM DESIGNER

Além do HPS e dos recursos computacionais associados, também é preciso que se programe o FPGA. Para o FPGA Cyclone V existe a versão de *software* gratuito, porém com menos recursos, que é o *Quartus Prime Lite Edition* (doravante somente *Quartus*), disponível no website da Intel. A versão utilizada neste projeto foi a 18.1.

Utilizando placas de desenvolvimento de empresas parceiras da Intel, como a *Terasic*, pode-se aproveitar de algumas facilidades. Como o DE10-Nano trata-se de um kit de desenvolvimento produzido por uma empresa em parceria com a Intel, existem diversos tutoriais, aplicações e configurações prontas. Um *site* de auxílio ao desenvolvedor de produtos FPGA Intel é o *www.rocketboards.org*. Também no *site* da própria *Terasic* é possível fazer o download do conteúdo do CD-ROM antigamente fornecido junto com o kit, que possui a documentação de suporte. Nesta documentação encontra-se a pasta de projeto chamada DE10_NANO_SoC_GHRD. O GHRD (*Golden Hardware Reference Design*) é um projeto *premium* do *software Quartus*, fornecido pela *Terasic*, considerado um projeto de referência. Ele é composto de um conjunto de componentes essenciais de *hardware* e *software* que servem de ponto inicial para diversos projetos que utilizem o kit. Pode-se citar, por exemplo, as designações dos pinos do FPGA, com seus níveis de tensões, nomes e funções, que já estão quase totalmente definidos e acessíveis em um arquivo Verilog. Outra funcionalidade importante presente, e de maior interesse deste trabalho, são alguns *Soft-IPs* dos componentes do FPGA, que terão interação com o HPS e que estão presentes no kit.

Por outro lado, como este trabalho utiliza o *rsYocto*, o criador desta distribuição Linux também disponibiliza um projeto para o FPGA, com várias funcionalidades prontas e com mais interação do que foi criado para o HPS no GHRD. O *download* da versão do projeto *Quartus* para o FPGA com a versão rsYocto-1.041 pode ser feito pelo endereço em [17]. Caso a versão de *boot* utilizada na seção 5.2 seja diferente, a versão selecionada no *site* deve ser reconsiderada. O arquivo em questão é o 'DE10NANOrsyocto.qar', que deve ser descompactado utilizando a função *'Restore Archived Project'*, em *'Project',* no *Quartus*.

Nas seções seguintes não serão tratadas a linguagem ou a forma como programar um FPGA, mas sim a interligação entre os diversos arquivos e componentes criados no *Quartus* e suas ferramentas ramificadas. Serão abordados apenas os pinos, funções e alguns conceitos de programação no que tangem à criação de componentes IP e à comunicação entre o HPS e o FPGA. Neste kit de desenvolvimento existem LED's e chaves ligadas ao FPGA que serão controlados pelo HPS, e estes demonstrarão os exemplos. Para abrir o projeto, após sua descompactação, deve-se utilizar o arquivo de extensão '.qpf'. Pode haver alguns avisos de diferenças entre versões do *Quartus* em que foi criado o projeto com a que está sendo utilizada. Para resolver as incompatibilidades basta confirmar todas as solicitações caso a versão utilizada seja mais nova. Versões mais antigas que a 18.1 não foram testadas.

Para configurar os componentes que farão parte da interconexão entre o FPGA e o HPS deve-se utilizar a ferramenta *Platform Designer*, dentro do *Quartus*. A versão fornecida em [17] já possui os componentes configurados e serão utilizados como exemplo.

6.1. Ferramenta de integração *Platform Designer*

O *Platform Designer* (PD) é uma ferramenta de integração de sistemas, que gera automaticamente a lógica de interconexão entre funções de propriedade intelectual (IP) e os subsistemas do FPGA [18]. O nome anterior desta ferramenta era *Qsys*, utilizado nas versões mais antigas do *Quartus Prime* e *Quartus II*.

Com o projeto geral descompactado e aberto no *Quartus*, abre-se o PD em: '*Tools/Platform Designer*'. Será solicitado o arquivo de extensão '.qsys', que deve estar na pasta raiz do projeto.

A Figura 4 é a visualização do *software Platform Designer* com um arquivo em edição. No canto superior esquerdo encontra-se o catálogo de componentes prontos, chamado de '*IP Catalog*'. Logo abaixo, em '*Hierarchy*', estão todos os pontos relevantes criados e interconectados entre os diversos componentes já configurados em '*System Contents*'. Esta é a área onde estão os componentes selecionados pelo desenvolvedor e onde deve se executar suas interligações. Em '*Parameters'* podem ser visualizados e ajustados os parâmetros de algum

componente selecionado em '*System Contents*', ou em '*Hierarchy*'.

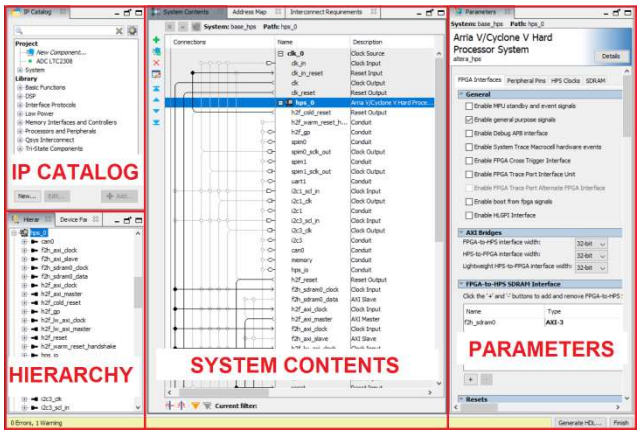

Figura 4: Interface gráfica do *Platform Designer*.

A interface do usuário com o PD visa facilitar o desenvolvimento do código; as conexões entre os componentes são selecionadas clicando nos nós com os círculos vazios. Os nós preenchidos representam as interconexões já estabelecidas (Figura 5).

A Figura 5 mostra resumidamente o circuito criado na versão rsYocto-1.041. O nome do arquivo é 'base_hps.qsys', sendo importante memorizá-lo pois irá aparecer em vários locais. Muitas informações foram suprimidas nesta figura, e os endereços da interconexão foram deslocados da coluna à direita (que foi omitida) para a esquerda para melhor na visualização. Portanto, os endereços representam o escravo em questão (destacados em vermelho).

Na sequência serão tratados os pontos mais importantes para exemplificar a interconexão entre HPS e FPGA. O primeiro componente é o '*Clock Source'*, que possui uma saída de *clock* direcionada para outros componentes. Há a possibilidade de utilizar *PLL's* para alterar sua frequência. Nota-se, ainda, na Figura 5, que o sinal de entrada de *clock* (clk_in) foi exportado. Isto é feito com clique duplo na junção da linha em questão com a coluna '*Export'*. Desta forma, associa-se este sinal a algum pino físico do FPGA. Isto será novamente abordado durante a explicação do código. Conclui-se que este componente possui uma entrada que é um pino físico, e uma saída que será distribuída pelo *hardware* do FPGA utilizando esta plataforma, conforme a necessidade do usuário.

O segundo componente é o *hps_0*, que é uma representação do HPS do SoC. Ele possui diversas funcionalidades e parâmetros para serem configurados, mas que não serão abordados neste trabalho. Basicamente, aqui se parametrizam as interfaces com o FPGA. Seleciona-se o proprietário de alguns periféricos (se é o HPS ou o FPGA), interrupções, DMA, diversas configurações relacionadas à memória SDRAM, e o mais importante a ser tratado aqui: a configuração das pontes (ver seção 3.1).

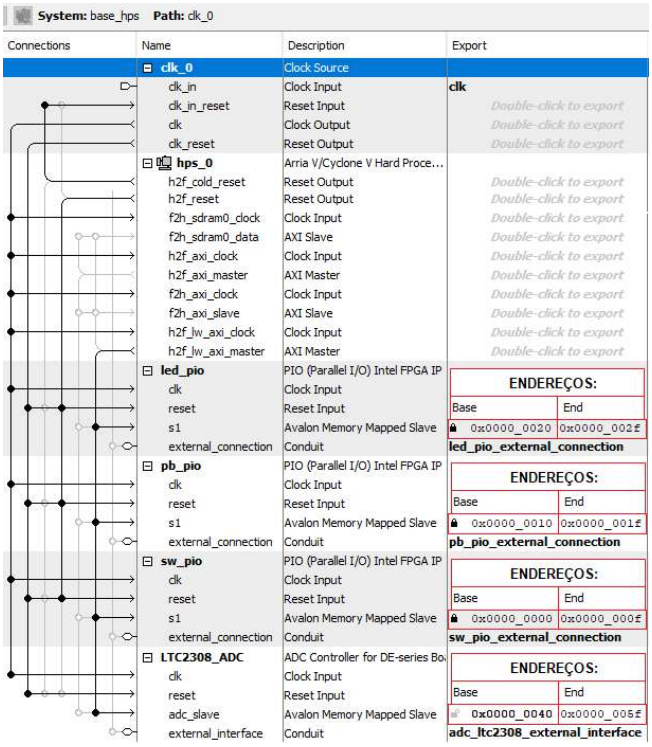

Figura 5: Interligação entre os componentes e os endereços de acesso aos componentes escravos, no *Platform Designer*.

Para acessar todas estas configurações basta clicar duas vezes na primeira linha do componente. Na lateral direita do PD, em '*Parameters*', desce-se até 'AXI *Bridges*' (ver seção 2), ali estão as opções de largura do barramento entre 32 e 128 bits para H2F e F2H, ou a opção para desabilitá-las. A LWB possui largura fixa de 32 bits. Todas as pontes estão configuradas com larguras de 32 bits e estão habilitadas. Desta forma, é possível visualizar suas conexões no componente *hps_0* da Figura 5.

Os próximos três componentes contêm 'pio' em seus nomes, porque trata-se de PIOs (*Parallel Input / Output*), que é um tipo de IP associado a entradas e saídas digitais do FPGA utilizando Avalon-MM (ver seção 2). Nas parametrizações de cada um destes componentes estão as configurações de entradas ou saídas: em 'pb_pio' estão configuradas as duas entradas de botão KEY0 e KEY1; em 'sw_pio' estão configuradas como entradas as chaves SW [3:0]; e o 'led_pio' representa as saídas digitais dos oito LEDs LED [7:0]; todos estes componentes eletrônicos estão instalados na placa de desenvolvimento e ligados aos pinos do FPGA. Observa-se que, de forma semelhante ao primeiro componente (*clk_0*), estes três também possuem subcomponentes exportados ('*external_connection*'). São interfaces Avalon do tipo *Conduit* (ver seção 2), que serão usadas no decorrer do código no *Quartus* para associá-los aos pinos físicos do FPGA.

O último componente é o 'LTC2308_ADC', referente ao ADC LTC2308, integrante da placa de desenvolvimento. Há duas possibilidades de abordagem: utilizar o IP genérico do catálogo de IP's, com nome *ADC Controller for DE-*

series Boards, da Intel, ou criar manualmente o componente. Utilizando a primeira opção não foi possível atingir a taxa de amostragem do ADC, de 500 kHz. Portanto, para esse objetivo, deve ser criado um componente específico e o associar a um programa HDL desenvolvido de forma exclusiva. Esta etapa será abordada posteriormente. Nota-se que há também a exportação de subcomponentes ('*external_interface*'), que são os pinos do FPGA interligados fisicamente ao ADC.

Cabe agora salientar a interligação das pontes entre o HPS e os demais componentes. Ainda na Figura 5, nota-se que os quatro últimos componentes são escravos da LWB do *hps_0*. No componente HPS tem-se o mestre '*h2f_lw_axi_master'* (AXI *Master*) interligado aos componentes escravos PIO em 's1', e em 'adc*_slave'* no componente do ADC, todos descritos como *Avalon Memory Mapped Slave*. Ainda nos escravos, na coluna mais à direita, encontram-se os *offsets* dos endereços de cada componente, relativos ao endereço inicial da LWB, que é 0xFF200000 (Tabela 1). Durante a inserção do componente escravo no diagrama de '*System Contents*', os endereços são automaticamente calculados, mas o endereço inicial pode ser alterado pelo usuário desbloqueando o símbolo de cadeado ao lado do endereço. Estes são os endereços que deverão ser utilizados no programa de usuário do HPS. Vale lembrar que a conversão entre os protocolos AXI e Avalon é feita automaticamente sem intervenção do usuário.

6.2. Criando um componente com interface Avalon Memory-Mapped Slave no Platform Designer

Como discutido na seção anterior, pode-se criar manualmente um componente para se obter melhores resultados do ADC presente na placa de desenvolvimento. O componente a ser criado é um escravo do HPS que utiliza a ponte LWB. Este componente, assim como os demais que já estão disponíveis no *Platform Designer*, são chamados de IP (*Intellectual Property*).

Será abordada uma maneira de criar um componente de forma resumida; há muitas possibilidades, diversos tipos de componentes, variações de sinais e interfaces. O exemplo abordado foi efetivamente criado para um outro projeto com uma perspectiva particular.

Inicialmente, foi criado um arquivo Verilog com a declaração do Módulo 'avalon_adc_ltc2308'. O nome de declaração também deve ser o nome do arquivo. O arquivo pode estar na pasta raiz do projeto, por exemplo. Para este componente escravo são necessários alguns sinais (registros ou binários) para uma interface Avalon [19] p.14:

- *address*: é utilizado para direcionar a qual variável está se querendo ler ou escrever. É necessário definir quantos bits. Foi considerado 3 bits, conseguindo assim oito possibilidades de endereços (2^3);
- *read*: sinal binário que indica que a operação em questão é a leitura de um dado;
- *readdata*: registro de 32 bits que conterá o valor retornado ao mestre de acordo com sua solicitação;
- *waitrequest*: o componente escravo pode declarar que está ocupado mantendo este bit ativo, mantendo a comunicação em *stand-by*;
- *write*: sinal binário que indica que a operação em questão é a escrita de um dado;
- *writedata*: registro de 32 bits que contém o dado que está sendo enviado do mestre para o escravo.

Para aprofundar o entendimento, a aplicação destes sinais e registros pode ser exemplificada por uma interação direta entre uma CPU e uma memória. O mestre (CPU) que vai ler um dado da memória, informa ao escravo (memória) o endereço que se quer ler no registro *address* e em seguida ativa o *read*, então a memória disponibiliza o valor solicitado em *readdata*. Se o mestre vai escrever algum valor na memória, ele insere este valor em *writedata*, insere o endereço desejado da memória em *address*, e após isto ele ativa o *write*, a memória então registra o valor no endereço solicitado.

Também pode-se exemplificar a aplicação destes sinais e registros voltando para o uso do ADC – lembrando que ele é externo ao FPGA, e neste caso especificamente, o FPGA envia os comandos e recebe leituras do ADC via interface SPI. Como exemplo, pode-se pensar em um processo que contém um comando para iniciar a conversão AD e outro comando para enviar uma configuração ao ADC. Ambos os processos são de escrita do mestre para o escravo, mas possuem funcionalidades diferentes. Para distinguir entre estas funções utiliza-se o registro *address,* – apesar de parecer um pouco incoerente em razão do nome do registro. Por exemplo: com o *write* ativo e com *address* igual a 0, o código HDL deve interpretar que o FPGA precisa enviar o comando de iniciar a conversão ao ADC, ao passo que se *address* = 1, o código HDL deve interpretar que o FPGA precisa enviar um comando de configuração ao ADC com o valor presente em *writedata*, definido previamente pelo HPS. A seguir serão ilustradas as duas opções, e neste momento não será abordada a linha de código no HPS para enviar os comandos para o FPGA.

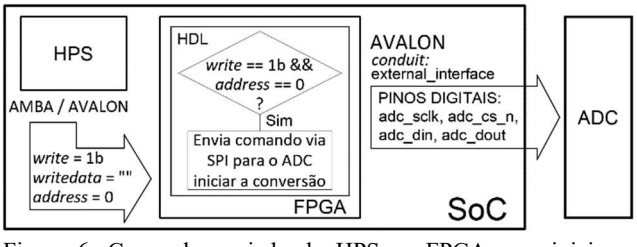

Figura 6: Comando enviado do HPS ao FPGA para iniciar a conversão do ADC.

Na Figura 6 está representado o trajeto de uma solicitação ao FPGA feita pelo HPS, com o intuito de se fazer uma leitura analógica atualizada do ADC. Utilizando alguma ponte (LWB ou H2F), o HPS inicia o envio de sua solicitação ao FPGA. O *System Interconnect* (ver seção 3) realiza a conversão entre as arquiteturas AMBA e Avalon. O código HDL presente no FPGA, ao detectar a condição necessária (*write* = 1 e *address* = 0), inicia o envio do comando pré-definido ao *chip* do ADC para que este faça uma aquisição analógica. Este comando é enviado via SPI, utilizando os pinos digitais do FPGA definidos previamente na interface Avalon do tipo *Conduit* (ver seção 2).

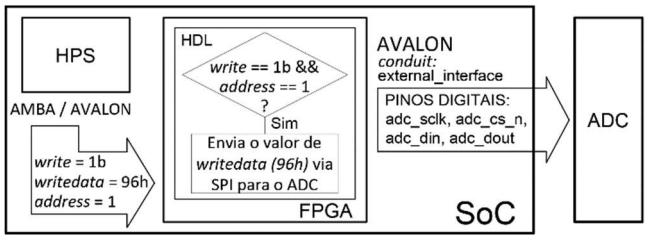

Figura 7: Comando enviado do HPS ao FPGA para enviar um código específico ao ADC.

Na Figura 7 tem-se o exemplo de um processo em que o HPS precisa enviar um determinado código ao ADC através do FPGA. A diferença neste caso, em relação ao exemplo anterior, é que há um valor presente em *writedata* que vai ser enviado ao ADC. Para diferenciar este processo com o do exemplo anterior, faz-se o valor do registro *address* igual a 1, desta forma, através deste registro, consegue-se distinguir os diferentes processos dentro do código HDL.

Foi exemplificada a interface do tipo *Conduit*, que representa a interface do componente IP com pinos físicos do FPGA. Para aprofundar mais sobre a utilização deste tipo de interface, é preciso destacar que alguns sinais são específicos para este componente em particular, pois trata-se de um controle de um *chip* específico. Assim sendo, este ADC precisa dos sinais do protocolo SPI: sclk (*clock*), din (*data in*), dout (*data out*) e cs_n (*chip select* negativo). Como todos estes sinais são pinos do FPGA, eles têm de ser configurados como entrada/saída e ter os tamanhos definidos em uma interface *Conduit*.

Além da interface *Conduit,* é necessário configurar interfaces obrigatórias para este componente, a saber, as responsáveis por gerar os sinais de *clock* e *reset*. Estes sinais são utilizados amplamente no HDL. O *clock* é o sincronizador dos processos e o *reset* tem de ser utilizado para reiniciar as diversas variáveis e registros dentro dos processos quando o HPS envia este sinal.

Abaixo está apresentado o arquivo Verilog preparado especificamente para este componente IP. Ele é considerado um arquivo de síntese, que será criado durante a geração do HDL no *Platform Designer* (ver '*Generate HDL*' na seção 6.4) e incorporado ao projeto do *Quartus*. Ele não é obrigatório, mas facilita a criação do componente, tendo a função inicial de fornecer os parâmetros e sinais que serão usados ao gerar o novo modelo de componente IP, e posteriormente será anexado ao programa principal quando for usado em algum projeto. O arquivo ficará armazenado no próprio componente criado, e é considerado como seu arquivo do tipo *Top-Level*. Para promover um melhor entendimento no decorrer desta seção, este arquivo será chamado de *Núcleo*. Uma observação importante: este arquivo *Núcleo* – que irá aparecer no projeto do *Quartus* após a geração do HDL no PD – não deve ser alterado, pois ele será renovado a cada geração do HDL no PD. Uma maneira usual para fazer uso do componente, a partir de sua inserção ao projeto do *Quartus*, é utilizando de instâncias, ou seja, haverá este arquivo fixo *Núcleo*, e outro onde será desenvolvida a programação que faz uso do componente, doravante chamado *Célula*, que pode sofrer modificações pelo desenvolvedor pois não irá interferir no arquivo *Núcleo*. Para isso há duas opções: instanciar o arquivo *Célula* dentro do arquivo *Núcleo* antes da criação do componente, ou o oposto, onde o arquivo *Célula* instancia o arquivo *Núcleo*; esta opção pode ser realizada mesmo após a criação do componente. A única diferença observada na escolha das duas opções, durante a preparação desta Nota Técnica, é a localização dos arquivos dentro do projeto do *Quartus* quando se utiliza a aba *Project Navigator*. Na primeira opção, o arquivo *Célula* estará localizado na sequência do arquivo *Núcleo*, quando se utiliza a visualização em lista Hierárquica. Na segunda opção apenas o arquivo *Núcleo* permanece na hierarquia do projeto, o arquivo *Célula* somente é encontrado na opção de visualização em Arquivos (*Files*) da mesma aba, o que parece dificultar a navegação entre os arquivos. Neste exemplo foi utilizada a primeira opção.

No texto abaixo, o ADC2RAM_control u0 é a instanciação no arquivo original do componente (*Núcleo*). O nome do módulo ('*module*') deve ser o mesmo nome do arquivo (terceiro parágrafo desta seção):

```
module avalon_adc_ltc2308 (clock, reset, read, write,
  readdata, writedata, address, waitrequest, adc_sclk,
  adc_cs_n, adc_din, adc_dout );

 //Declaração das portas:
  input clock, reset, read, write;
  input [31:0] writedata;
  input [2:0] address;
  output reg [31:0] readdata;
  output reg waitrequest;
  input adc_dout;
  output adc_sclk, adc_cs_n, adc_din;

//Instanciação:
ADC2RAM_control u0 (
   .clock     (clock),          .address     (address ),
   .reset     (reset),          .waitrequest(waitrequest),
   .read      (read),           .adc_sclk    (adc_sclk),
   .write     (write),          .adc_cs_n    (adc_cs_n),
   .readdata  (readdata),       .adc_din     (adc_din),
   .writedata(writedata),       .adc_dout    (adc_dout)
); endmodule
```

Toda instância deve possuir seu arquivo HDL com o mesmo nome declarado no código. No código acima, a

instanciação ADC2RAM_control u0 transferirá todos os registros e sinais declarados neste trecho do código para sua instância ADC2RAM_control (arquivo *Célula*) diretamente, que neste caso especificamente, deverá conter toda a caracterização para a interação com o *chip* ADC.

Para entender melhor a necessidade deste arquivo *Top-Level* deve-se partir para a criação de um componente. Para tal, clica-se em '*File/New Component...*' no PD. Será aberta uma nova janela nomeada '*Component Editor*' como na Figura 8. Na aba '*Component Type*' tem-se as informações principais do componente que serão visualizadas no '*IP Catalog*' da área de trabalho do PD. Na aba '*Files*' está o local de inserção do arquivo *Top-Level*, que deve ser inserido pelo botão '*Add File...*' em '*Synthesis Files*'; em seguida o arquivo deve ser analisado através do botão '*Analyse Synthesis Files*'. Após a análise, a aba '*Signal & Interfaces*' possuirá os sinais, interfaces e registros previamente organizados, mas ainda precisa-se de alguma parametrização.

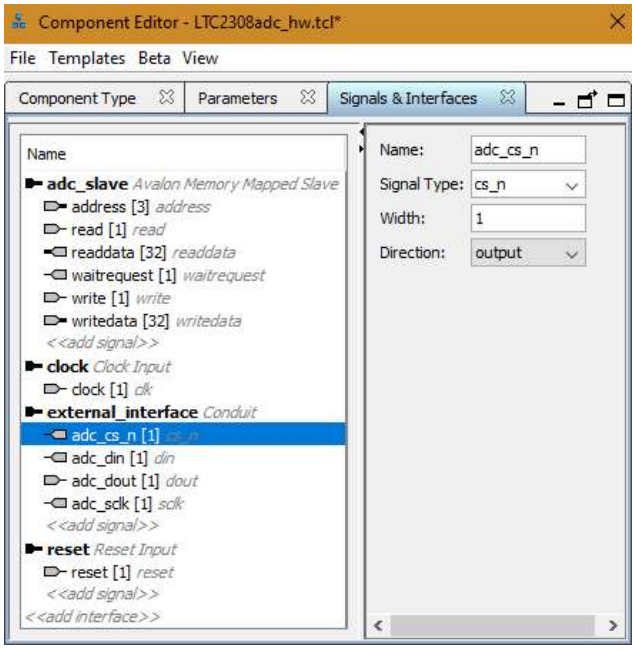

Figura 8: Criando um componente IP no Platform Designer

Todos os *sinais* estão reunidos nas diferentes *interfaces* detectadas na análise. Os sinais da interface Avalon ficam corretos, mas os sinais que serão exportados estão na interface errada e precisam de ajustes. Para isto clica-se em '*<<add interface>>*' e depois em *'Conduit*'. A interface nova aparecerá junto às outras com o nome '*conduit_end*'. Todos os quatro sinais do ADC podem ser selecionados e arrastados para a área da nova interface criada. Estes sinais ainda precisam de ajustes individuais no campo '*Signal Type*', ao selecionar algum destes, aparecerão suas configurações ao lado direito. Neste campo em específico, sugere-se colocar a função de cada um como p. ex. 'cs_n' para o "adc_cs_n'. Deve-se confirmar se a interface *clock* está configurada como do tipo '*Clock Input*' e com nome adequado; e se o seu sinal '*clock'* desta interface está com o '*Signal Type*' em 'clk'. Da mesma forma, a interface de *reset* deverá ser configurada como tipo '*Reset Input*', e seu sinal deverá estar com o '*Signal Type*' em 'reset'.

Os nomes dos sinais não devem ser alterados, pois já estão no código analisado. O mesmo não se aplica aos nomes das interfaces, que podem ser alterados, sendo estes os nomes que aparecerão em '*System Contents*' quando o componente for inserido. Para se obter o que está descrito na Figura 5, os nomes devem ser *adc_slave*, *clock*, *external_interface* e *reset*. Após isto, finaliza-se a criação do componente clicando em '*Finish...*'. O componente gerado poderá ser localizado em '*IP Catalog*' com o nome definido no início do processo.

Caso não se esteja utilizando um arquivo como foi exemplificado, pode-se utilizar da opção de *Templates*, selecionando para este caso, o modelo '*Add Avalon-MM Simple Slave*', isto facilita na criação das interfaces e sinais.

Torna-se pertinente ressaltar que esta versão IP do ADC não é integrante do diretório do rsYocto. A versão original utiliza o IP da própria Intel, e os exemplos de comandos que serão explicados abaixo são relativos ao original e não a este que foi aqui construído.

6.3. Funcionamento do componente IP do ADC LTC-2308 original da Intel

Este componente IP, diferentemente dos outros IPs PIO que foram utilizados, possui mais de um endereço e também aceita operações de leitura e escrita. Entender seu funcionamento ajuda a reforçar as funções de cada interface Avalon. Um outro ponto a se destacar é que as informações abaixo estão relacionadas ao componente IP original da Intel, o mesmo que vem na versão rsYocto 1.041. O componente criado e explicado na seção 6.2 contém funções diferentes em seus endereços, porque isto é particularmente definido no HDL do componente.

Este IP contém algumas configurações que podem ser ajustadas no *Platform Designer*: seleção do modelo da placa de desenvolvimento da *Terasic*, seleção do *clock* do ADC, e seleção da quantidade de canais. Esta última configuração é importante se ajustar antes da compilação, porque quanto maior a quantidade de canais habilitados menor é a quantidade de amostragens por canal. Isto acontece porque o funcionamento do componente inicia quando se ativa a variável *Update*. Com isto, todos os canais habilitados serão lidos sequencialmente. Se o intuito é utilizar somente o primeiro canal, e os oito estão habilitados, será feita uma varredura desde o primeiro até o último canal para depois retornar ao primeiro, ou seja, a taxa de amostragem é oito vezes menor.

Um manual específico deste componente acompanha a instalação do *Quartus*, e está disponível em [21]. Os dados

da Tabela 2 foram retirados deste manual e possuem as funções associadas aos endereços e às operações.

De acordo com a tabela, se o endereço 0 (ou seja, o primeiro endereço do componente) é selecionado, mas for uma operação de leitura, o HDL do componente irá interpretar este comando e fornecer como resposta o valor do canal Analógico 0. Se por outro lado, ainda no endereço 0, for enviado um comando Avalon para escrita, o valor a ser escrito será interpretado pelo HDL do componente e realizará a lógica que lhe for atribuída. Portanto, neste componente deve haver uma operação de escrita do valor 1 no endereço 0 para que se atualize os valores convertidos dos canais habilitados.

Tabela 2: Endereços e suas funções para leitura e escrita do ADC LTC 2308 na comunicação Avalon (usando IP da Intel).

Offset (bytes)	Nome do Registro	*Read* ou *Write*	Função
0	CH_0	R	Valor do canal 0
	Update	W	Atualiza os valores convertidos
4	CH_1	R	Valor do canal 1
	Auto_update	W	Hab / Desabilita *auto-updating*
8	CH_2	R	Valor do canal 2
12	CH_3	R	Valor do canal 3
16	CH_4	R	Valor do canal 4
20	CH_5	R	Valor do canal 5
24	CH_6	R	Valor do canal 6
28	CH_7	R	Valor do canal 7

6.4. Finalizando a estrutura no Platform Designer

Para completar o entendimento de como associar a estrutura criada no *Platform Designer* (PD) ao projeto do *Quartus* é necessário observar os nomes gerados nas exportações na coluna '*Export*'. Todos os pinos físicos devem ser exportados com clique duplo na linha correspondente dentro desta coluna, onde os nomes também podem ser editados. Estes nomes deverão ser declarados na entidade de nível superior (*Top-level Entity*) do projeto *Quartus* e associados aos pinos usando as devidas atribuições no *Pin Planner*.

Finalizadas as configurações e interligações no PD, gera-se o arquivo HDL no *menu* de opções '*Generate / Generate HDL*'. Se ainda não houver, será criada uma pasta no diretório raiz do projeto *Quartus* com o nome do arquivo do PD, que neste exemplo foi 'base_hps', e dentro desta pasta haverá o diretório '*synthesis \ submodules'* que conterá os arquivos de síntese de todos os componentes para o projeto. Mas para que o *Quartus* adicione estes arquivos ao projeto é necessário abrir a aba '*Project*' e ir em '*Add/Remove Files in Project...*'; abre-se uma nova janela com, pelo menos, o arquivo DE10NANOrsyocto.v (que é o *Top-level File*). Deve-se localizar o arquivo '*base_hps.qip*' dentro da pasta '*synthesis'* citada anteriormente; após a inserção, este deverá ser colocado como o segundo arquivo na lista.

6.5. Finalizando o projeto no Quartus

A entidade de nível superior deste projeto é o DE10_NANOrsyocto.v, um arquivo Verilog que possui as declarações dos pinos e instanciações de todo o projeto, inclusive pinos da saída de vídeo HDMI, por exemplo. Destacando as informações relativas a este trabalho tem-se:

```
module DE10_NANOrsyocto (
//declarações dos pinos conforme
//os nomes definidos no Pin Planner
 output  ADC_CONVST,      //pino do ADC
 output  ADC_SCK, //pino do ADC
 output  ADC_SDI, //pino do ADC
 input   ADC_SDO, //pino do ADC
 input    [1:0]    KEY,    //os 2 botões físicos
 output   [9:0]   LED,     //8 LEDs da placa
 input   [3:0]    SW,      //as 4 chaves físicas
 input   FPGA_CLK1_50     //entrada de um clock de 50
MHz
);

base_hps u0 (//'base_hps' é o nome do arquivo do PD
.clk_clk  (FPGA_CLK1_50), //associa a fonte de clock ao
clk_0
//associa os pinos externos às interfaces Conduit:
 .adc_ltc2308_external_interface_cs_n   (ADC_CONVST),
 .adc_ltc2308_external_interface_sclk    (ADC_SCK),
 .adc_ltc2308_external_interface_din     (ADC_SDI),
 .adc_ltc2308_external_interface_dout   (ADC_SDO),
 .led_pio_external_connection_export   (LED),
 .pb_pio_external_connection_export    (KEY),
 .sw_pio_external_connection_export    (SW)
);
endmodule
```

A Figura 9 contém os mesmos dados da Figura 5. O *Platform Designer* (PD) adiciona alguns sufixos aos nomes. O nome "exportado" do IP ltc2308adc é 'adc_ltc2308_external_interface', mas o PD adicionou a função do pino ao final do nome para diferenciar cada um. Aos nomes exportados dos PIOs foi adicionado '*_export*' e ao pino de entrada de *clock* adicionou-se o sufixo *'_clk'*.

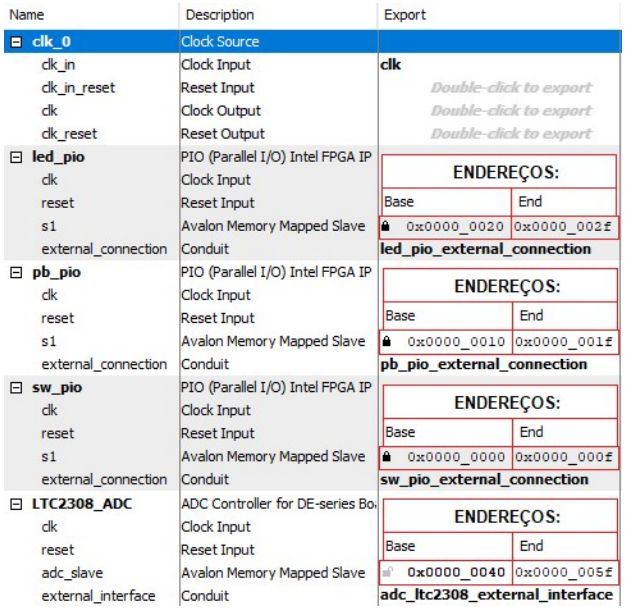

Figura 9: Os quatro componentes IP usados neste projeto que possuem conexão Avalon com o HPS e o *clock*.

Estas diferenças nos nomes podem ser verificadas no arquivo *base_hps.v* que se encontra no diretório '*base_hps \ synthesis*'. São estes os nomes que devem ser utilizados nos arquivos de nível superior.

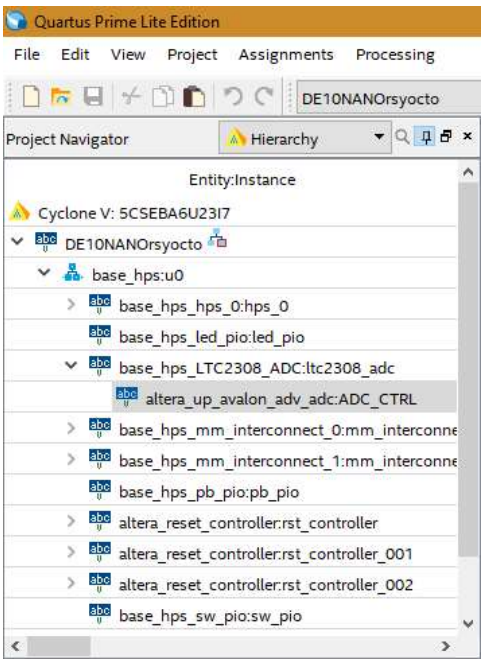

Figura 10: Estrutura do projeto *rsYocto*-1.041 compilado.

A hierarquia dos vários códigos HDL sintetizados pode ser visualizada no *Quartus* conforme a Figura 10. Nota-se o arquivo *Top-Level* 'DE10NANOrsYocto', seguido da entidade base_hps que foi gerada pelo PD. Na sequência, dentro desta entidade, estão seus arquivos HDL, alguns relacionados aos componentes IP presentes na estrutura, e outros, por exemplo, relacionados às interconexões. Utilizando o LTC2308_ADC' da Figura 9 como modelo, observa-se, na Figura 10, que este componente irá gerar dois arquivos após o processo de síntese no PD: o primeiro é o 'base_hps_LTC2308_ADC' (nome gerado pelo PD com prefixo e sufixo evidentes), e o segundo arquivo, o 'altera_up_avalon_adv_adc'. A hierarquia entre os dois arquivos se deve ao fato de que o primeiro arquivo instancia o segundo em determinada parte de seu programa. Neste caso em específico, por se tratar de um componente IP da Intel / Altera – e estabelecido por ela desta forma –, o segundo arquivo contém o HDL com o programa de funcionamento mais diretamente associado ao ADC, ou seja, controla a ativação e desativação dos respectivos pinos do FPGA, avalia o nível lógico dos sinais enviados pelo ADC via SPI, envia configurações ao ADC via SPI, etc. O primeiro arquivo ficou responsável por aguardar o resultado do ADC, interpretado pela instância inferior (o segundo arquivo), e disponibilizá-lo para as instâncias superiores. Também na Figura 10 nota-se as entidades relacionadas aos LEDs (led_pio), aos botões (pb_pio) e às chaves (sw_pio). Abaixo, uma parte do HDL referente aos LEDs gerado pelo componente do PD led_pio.

```
module base_hps_led_pio (address, chipselect, clk,
reset_n, write_n, writedata, out_port, readdata);

output [7: 0] out_port;      input        write_n;

input  [1: 0] address;       input  [31:0] writedata;
input         chipselect;    reg    [7:0]  data_out;
input         clk;           wire   [7:0]  out_port;
input         reset_n;

  always @(posedge clk or negedge reset_n)
    begin
      if (reset_n == 0)
          data_out <= 85;
      else if (chipselect && ~write_n && (address == 0))
          data_out <= writedata[7 : 0];
    end
  assign out_port = data_out;
endmodule
```

Observa-se, no código, que este componente ainda possui mais um sinal na interface *Avalon Memory-mapped Slave*, que é o '*chipselect'*. Este sinal não é necessário visto que a própria interconexão faz a tarefa de direcionar os dados para o componente desejado. Este sinal já é considerado descontinuado pela Intel [19].

Tomando como exemplo o HDL do componente IP dos LEDs, há uma finalidade diferente à do ADC: a saída 'out_port' é simplesmente convertida na descrição que foi utilizada no *Top-Level File*, não possui nenhuma outra lógica combinacional antes, ou seja, o valor carregado em 'out_port' é transferido diretamente para os LEDs. Esta 'conversão ocorre no arquivo base_hps.v da seguinte forma:

```
base_hps_led_pio led_pio ( [...]
    .out_port (led_pio_external_connection_export) );
```

O projeto pode ser compilado e transferido para o FPGA via USB-Blaster II (J13 da placa) para testes. O arquivo gerado pela compilação pode ser alocado no cartão após a geração da imagem; uma das partições é acessível via *Windows*, e esta contém o arquivo que é carregado no FPGA naturalmente durante o *boot*. O nome deve ser 'socfpga.rbf'. Para gerar o arquivo com esta extensão é necessário utilizar um recurso do *Quartus*, acessível em '*File / Convert Programming Files...*'. Após aberto o conversor, seleciona-se o tipo '*Raw Binary File (.rbf)*', e o modo '*Passive Parallel x16*'. Em '*Input files to convert*', seleciona-se o arquivo já presente na janela e clica-se em '*Add File...*' que abrirá uma nova janela; deve-se percorrer até a pasta do projeto e selecionar o arquivo '.sof' que possui o nome do projeto; clica-se em '*Generate*'.

7. COMANDOS PARA USO DAS PONTES

Serão abordadas duas maneiras para realizar o acesso aos IPs do FPGA via HPS. Ambas são feitas via Linux, seja por comandos tipo *Shell* ou usando sintaxe em linguagem *Python*. Outro modo de realizar este tipo de operação é através do NIOS II, um processador implementável no FPGA via PD, mas não será tratado neste trabalho.

7.1. Utilizando comandos Shell

O criador da distribuição rsYocto criou ferramentas para deixar a interação com os IPs do FPGA mais fácil, chamado de *rstools*. O mais aconselhável é utilizar os comandos criados por ele. Utilizando algum *software* Terminal (pela UART ou ethernet), após o *login* digita-se FPGA seguido de *TAB* (3x) para se visualizar todos os comandos possíveis. São eles: FPGA-gpiRead, FPGA-gpoWrite, FPGA-readBridge, FPGA-readMSEL, FPGA-resetFabric, FPGA-status, FPGA-writeBridge, FPGA-writeConfig. O sufixo '-h' após o comando fornece informações mais detalhadas.

Os endereços de cada componente IP na Figura 9 serão utilizados nos comandos. Todos os IPs nesta figura utilizaram a LWB e, portanto, utiliza-se o sufixo -lw no comando. Desta forma, o endereço selecionado será automaticamente somado ao endereço inicial da LWB que é o 0xFF200000 (ver Tabela 1):

- Para apagar todos os LEDs (endereço 20h):
  `FPGA-writeBridge -lw 20 0`
- Para acender todos os LEDs:
  `FPGA-writeBridge -lw 20 255` *(final em decimal)*
  `FPGA-writeBridge -lw 20 -h ff` *(final em hex)*
- Para controlar somente um LED:
  `FPGA-writeBridge -lw 20 -b 4 0` *(desliga LED 4)*
  `FPGA-writeBridge -lw 20 -b 7 1` *(liga LED 7)*
- Para ler o estado das chaves SW (endereço 0):
  `FPGA-readBridge -lw 0`
- Para ler o estado dos botões KEY (endereço 10h)
  `FPGA-readBridge -lw 10`

A resposta destes comandos traz várias informações, inclusive gráficas. Isto pode exigir certa manipulação para extrair o que realmente se precisa. Todos estes comandos, além de outros, estão disponíveis em [20].

7.2. Utilizando comandos Python

Como já citado, foi utilizado o *Visual Studio Code Insiders* com a extensão *Python* instalada, para execução e depuração de códigos *python* (.py). Para executar arquivos *python* no *Windows* é necessário antes instalar o interpretador *Python* para *Windows* utilizando a *Windows Store*. Com ele é possível utilizar a função 'pip' para fazer instalações de pacotes relativos à programação *Python*, inclusive no *Visual Studio*. Os passos para a instalação podem ser visualizados em [11].

Com o VSCI conectado com a placa (seção 5.5) nota-se que há diversos programas de exemplo com a extensão .py no diretório raiz do usuário (/*home*/*root*), que são rotinas que utilizam código *Python*. Para executá-los pelo processador do SoC basta digitar no *software* terminal:

```
root@cyclone5:/# python3 <nome_do_arquivo>.py
```

Este mesmo diretório possui o arquivo 'devmem.py', que será utilizado em todos os programas ou rotinas *Python* que precisarem utilizar as *bridges* entre HPS e FPGA. Este arquivo não deve ser alterado, e os arquivos que o utilizarem devem estar preferencialmente no mesmo diretório ou será necessário incluir o diretório completo na função de importação. Abaixo o código de acesso ao ADC LTC-2308 com o IP da Intel:

```
1. import devmem
2. de = devmem.DevMem(0xFF200000,0x40 + 32, "/dev/mem")
3. de.write (0x40,[1])
4. raw_value = (de.read (0x40,1)) [0]
5. de.write (0x40,[0])
```

Estas linhas representam o mínimo necessário para acessar o componente. As linhas estão numeradas e alguns números estão à vista para facilitar o entendimento. O código deve estar em um arquivo com extensão .py.

Na linha 1 é importada a rotina 'devmem', como citado anteriormente (todas as demais linhas deste código utilizam a rotina). A linha 2 realiza um processo chamado de *montagem*, ela faz com que a variável nomeada 'de' represente um trecho de endereços de memória, remetendo ao uso de interfaces tipo *Memory-Mapped.* Neste caso, como trata-se da LWB, o endereço é 0xFF200000 (Tabela 1). Na Figura 9 consegue-se o valor *offset* do endereço da LWB e a quantidade de endereços utilizados (0x5F – 0x40 = 31**d.** 31 + 1 = 32 endereços). Estes são os valores inseridos após o endereço inicial e indicarão ao processo de montagem a quantidade de endereços a serem mapeados. O final do comando "/dev/mem" formaliza que a montagem será dos endereços de memória, já que o Linux utiliza este diretório para ter acesso à memória física.

O endereço inicial do componente ADC é um *offset* de '0x40' (Figuras 5 e 9) em relação ao endereço da LWB (0xFF200000). De forma similar, este endereço inicial é considerado como valor de *offset* igual a '0' para a Tabela 2. Na sequência, o endereço '0x44' é considerado como valor '4', e assim por diante. Isto vale para todas as ações de leitura e escrita.

Na linha 3 há o comando de escrita (W) no *offset* do ADC (0x40). O valor '1' à direita é automaticamente inserido no sinal *writedata* da interface Avalon. Como citado no parágrafo anterior, o valor referente ao *offset* de '0x40' é igual a '0', e este valor é inserido no sinal *address* da interface Avalon. Para acessar outros canais, basta adicionar o *offset* relativo ao canal. O canal 1 (CH_1), por exemplo, será acessado substituindo o valor da esquerda (0x40) da linha 3 por '0x44', pois possui o *offset* igual a '4'; desta forma o valor de *address* será '4'. Toda a lógica de ativar ('1') ou desativar ('0') os demais sinais da interface – como o *write* ou *chipselect* – é abstraída, o que também acontece na operação de leitura. Portanto, ainda de acordo com a Tabela 2, escrever (W) o valor '1' com o *address* = 0

(*Update*) irá iniciar a função de atualização dos valores convertidos do ADC.

Na linha 4 ocorre a operação de leitura. Novamente, acessando a Tabela 2, observa-se que em uma operação de leitura (R) no *address* = 0, haverá o retorno do último valor convertido do canal 0 do ADC (CH_0). Este valor é transportado pelo sinal *readdata* da interface Avalon, e guardado na variável '*raw_value*'.

A última linha do código desativa a atualização dos valores do ADC. Nesta linha, o valor '0' é escrito (W) no *address* = 0. O uso de números decimais, hexadecimais ou binários em quaisquer partes do código é irrelevante.

Quanto aos LEDs, a forma é semelhante, basta se atentar aos endereços e sempre enviar os estados dos oito LEDs:

```
de = devmem.DevMem(0xFF200000, 0x20 + 16, "/dev/mem")
de.write (32,[7])// 0x20 = 32d //Estado dos LEDs: [7]
00000111b
raw_value = (de.read (32,1)) [0] //Para ler o estado
atual
de.write (32,[6])// 0x20 = 32d //Estado dos LEDs: [6]
00000110b
```

Para as chaves SW:

```
de = devmem.DevMem(0xFF200000, 0x0 + 16, "/dev/mem")
raw_value = (de.read (0,1)) [0] //Para ler o estado
atual
```

Para os *push-buttons*:

```
de = devmem.DevMem(0xFF200000, 0x10 + 16, "/dev/mem")
raw_value = (de.read (0x10,1)) [0] //Para ler o estado
atual
```

Estes exemplos representam instâncias de programas individuais. Se, por exemplo, for necessário utilizar em um único programa os LEDs, botões e as chaves, basta utilizar apenas um comando de mapeamento da memória englobando o último endereço de todos os componentes; neste caso deve-se utilizar o comando dos LEDs:

```
de = devmem.DevMem(0xFF200000, 0x20 + 16, "/dev/mem")
```

Desta forma, '*de*' incluirá desde o endereço 0xFF200000 até o 0xFF20002F. Se no comando de leitura for utilizado o endereço '0', será lido o estado atual dos *switches* normalmente:

```
raw_value = (de.read (0,1)) [0] //Para ler o estado
atual das SW
```

De forma análoga, ao se utilizar o endereço do IP ADC, que neste caso é o último, pode se utilizar todos os IPs em apenas um mapeamento de memória. De qualquer forma, todos os acessos deverão ser realizados em um mesmo arquivo *Python*.

7.3. Utilizando a ponte H2F

A ponte *HPS-to-FPGA* (H2F) também pode ser testada utilizando esta versão do *rsYocto*, e a forma de trabalho é a mesma que a LWB, demonstrada nas seções anteriores. A única ressalva é que a configuração da H2F foi testada apenas com 32 bits, e esta ponte tem a possibilidade de fornecer o barramento com 32, 64 ou 128 bits, o que pode ser configurado via *Platform Designer* (PD).

O componente IP que foi utilizado para estes testes é um '*System ID Peripheral Intel FPGA IP*', que simplesmente fornece um número pré-definido quando lhe é pedido. Sua identificação no PD é '*sysid2*' e está ligado ao HPS via '*h2f_axi_master*', ou seja, a ponte H2F. O *offset* no endereço referente à H2F (0xC0000000 (Tabela 1)) é 0x00 e o endereço final é 0x07. Para acessar este componente utilizando um terminal basta digitar o comando:

```
FPGA-readBridge -hf 0
```

Para realizar o acesso com código *Python*, deve-se utilizar as mesmas condições da LWB da seção anterior (utilizando *devmen.py*, etc):

```
de = devmem.DevMem(0xC0000000, 0x0 + 7, "/dev/mem")
raw_value = (de.read (0x0, 1)) [0]
```

8. CONCLUSÃO

Neste trabalho foi apresentada uma introdução ao conjunto de interconexões, presentes em um SoC com FPGA + HPS, e como utilizá-lo para acesso aos recursos do SoC. Inicialmente foram abordadas as arquiteturas de interconexão de ambos os dispositivos, pois são de fabricantes distintos. Abordou-se conceitos relacionados à Interface Avalon, Arquitetura AMBA, *Memory-Mapping* (Mapeamento em Memória) e às *Bridges*, que servem de conexão entre HPS e FPGA. Foram apresentadas a instalação e o uso de um sistema operacional Linux, que inclui as funcionalidades da interconexão em um sistema embarcado com processador ARMv7, além de suas ferramentas computacionais para programação, depuração e manuseio dos arquivos e diretórios. Quanto ao FPGA, foram abordados os conceitos de criação, funcionamento e integração de componentes IP sintetizáveis, que se associam a um sistema com interface Avalon para a comunicação com o HPS. Por fim, foram mostrados os códigos e comandos para estabelecer comunicação entre os dispositivos, tanto via comandos *Shell* quanto comandos em linguagem *Python*.

Foram abordadas somente as comunicações onde o HPS é o gerenciador. Há outras possibilidades onde o FPGA é o controlador do HPS ou da memória SDRAM. Outro ponto em aberto é a inicialização automática de arquivos ou comandos, ou a execução de algum arquivo *Python* de forma arbitrária, visto que todos os comandos foram iniciados manualmente via terminal. A inicialização automática deixaria a placa mais autônoma.

Diante da grande quantidade de possibilidades de configuração, roteamento e programação, este trabalho apresenta somente uma primeira abordagem de como iniciar um projeto perante as infinitas e possíveis aplicações

utilizando essa poderosa ferramenta que combina FPGA e um processador ARM em um único *chip*.

9. REFERÊNCIAS

[1] XILINX. Moving a Generation ahead. 2012. Acessado em 01/05/2023. https://www.amd.com/system/files/documents/xilinx-overview.pdf.

[2] ISA White Paper. Fifty Years of Microprocessor Technology Advancements: 1965 to 2015. *Itanium Solutions Alliance.* Acessado em 14/05/2023. https://www.ic.unicamp.br/~cortes/mo601/artigos_referencias/intel_50years_microprocessor_to_2025_white paper5.pdf.

[3] Gordon E. Moore. Cramming More Components onto Integrated Circuits, *ELECTRONICS, Volume 38 Apr. 19, 1965*. Acessado em 20/04/2023. Disponível em: https://hasler.ece.gatech.edu/Published_papers/Technology_overview/gordon_moore_1965_article.pdf.

[4] Geoff Koch. Discovering Multi-Core: Extending the Benefits of Moore's Law. *Technology@Intel Magazine*. Jul/2005.

[5] ALTERA Corporation. SoC FPGA Product Overview. *Advance Information Brief AIB-01017-1.3*. 2012.

[6] McCONNEL, Toni. Altera ships its first Cyclone V SoC devices. Acessado 03/05/23. https://www.embedded.com/altera-ships-its-first-cyclone-v-soc-devices/.

[7] MOLANES, Roberto Fernandez. et al. Performance Characterization and Design Guidelines for Efficient Processor–FPGA Communication in Cyclone V FPSoCs. *IEEE Transactions on Industrial Electronics, VOL. 65, No. 5, MAY 2018.*

[8] INTEL. Cyclone V Device Overview. CV-51001. ID: 683694. Versão: 07/05/2018.

[9] ALTERA Corporation. Cyclone V Hard Processor System Technical Reference Manual. cv_5v4. 28/02/2020.

[10] AMBA, ARM Developer. Acessado em 22/04/23 https://developer.arm.com/Architectures/AMBA.

[11] MICROSOFT. Uso de Python para scripts e automação. *Microsoft Learn*. Acessado em 14/05/23. https://learn.microsoft.com/pt-br/windows/python/scripting.

[12] MELO, Rodrigo A. et al. Memory-mapped I/O over Dual Port BRAM on FPGA. *Instituto Nacional de Tecnología Industrial*, Argentina. 2012.

[13] SEBASTIAN, Robin. RSYocto Linux Distribution for Intel SoC-FPGA. Acessado em: 26/04/2023. https://github.com/robseb/rsyocto/tree/rsYocto-1.042.

[14] INTEL Learning. Treinamento online com o tema: Getting Started with Linux OS for Intel® SoC FPGAs. Acessado em: 25/04/2023. https://learning.intel.com/Developer/learn/course/795/getting-started-with-linux-os-for-intelr-soc-fpgas.

[15] SEBASTIAN, Robin. RSYocto Releases. Acessado em: 02/05/2023. https://github.com/robseb/rsyocto/releases.

[16] SEBASTIAN, Robin. Debugging Python applications remotely. Acessado em 05/05/2023. https://github.com/robseb/rsyocto/blob/rsYocto-1.041/doc/guides/4_Python.md.

[17] SEBASTIAN, Robin. RSYocto Linux Distribution for Intel SoC-FPGA. Acessado em: 30/04/2023. https://github.com/robseb/rsyocto/tree/rsYocto-1.041/fpga.

[18] INTEL. Platform Designer. Acessado em 10/05/23. https://www.intel.com.br/content/www/br/pt/software/programmable/quartus-prime/qts-platform-designer.html.

[19] INTEL. Avalon Interface Specifications. MNL-AVABUSREF. 2020.01.03. Acessado em: 09/05/2023. https://www.intel.com/content/www/us/en/docs/programmable/683091/18-1/introduction.html.

[20] SEBASTIAN, Robin. Use of Hard IP, FPGA-IP and configuration of the FPGA fabric. Acessado: 10/05/2023. https://github.com/robseb/rsyocto/blob/rsYocto-1.041/doc/guides/2_FPGA_HARDIP.md.

[21] INTEL. ADC LTC2308 COMPONENT IP. <local_de_instalação_Quartus>\intelFPGA_lite\<versão_do_Quartus>\ip\altera\university_program\input_output\altera_up_avalon_adc\doc.

[22] TERASIC. DE10-Nano User Manual. DE10-Nano_User_manual.pdf.

Notas Técnicas é uma publicação de trabalhos técnicos relevantes, das diferentes áreas da física e afins, e áreas interdisciplinares tais como: Química, Computação, Matemática Aplicada, Biblioteconomia, Eletrônica e Mecânica entre outras.

Cópias desta publicação podem ser obtidas diretamente na página web http://revistas.cbpf.br/index.php/nt ou por correspondência ao:

Centro Brasileiro de Pesquisas Físicas
Área de Publicações
Rua Dr. Xavier Sigaud, 150 – 4º andar
22290-180 – Rio de Janeiro, RJ
Brasil
E-mail: alinecd@cbpf.br/valeria@cbpf.br
http://portal.cbpf.br/publicacoes-do-cbpf

Notas Técnicas is a publication of relevant technical papers, from different areas of physics and related fields, and interdisciplinary areas such as Chemistry, Computer Science, Applied Mathematics, Library Science, Electronics and Mechanical Engineering among others.

Copies of these reports can be downloaded directly from the website http://notastecnicas.cbpf.br or requested by regular mail to:

Centro Brasileiro de Pesquisas Físicas
Área de Publicações
Rua Dr. Xavier Sigaud, 150 – 4º andar
22290-180 – Rio de Janeiro, RJ
Brazil
E-mail: alinecd@cbpf.br/valeria@cbpf.br
http://portal.cbpf.br/publicacoes-do-cbpf