PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO GRANDE DO SUL

FACULDADE DE ENGENHARIA

FACULDADE DE INFORMÁTICA

ENGENHARIA DA COMPUTAÇÃO

Éverton Soares da Cunha

Desenvolvimento de Sistemas Embarcados utilizando Plataformas FPGA com Dispositivos ARM

Orientador: Prof. Dr. Fernando Gehm Moraes

Porto Alegre

2014

Desenvolvimento de Sistemas Embarcados utilizando Plataformas FPGA com Dispositivos ARM

O trabalho de conclusão de curso é apresentado como parte das atividades para obtenção do grau de Engenharia de Computação da Faculdade de Engenharia da Pontifícia Universidade Católica do Rio Grande do Sul.

Orientador: Prof. Dr. Fernando Gehm Moraes

Porto Alegre

2014

Éverton Soares da Cunha

Agradecimentos

Primeiramente agradeço a Deus por chegar até esse momento. Aos meus pais e minha irmã, por todo o suporte que uma família pode dar, sou extremamente agradecido. Em especial, à minha namorada, que teve muita paciência e calma com a minha produção textual e que me apoiou em todos os momentos. Agradeço ao meu orientador Fernando Gehm Moraes, que me trouxe para este mundo do hardware como bolsista do GAPH e desde então minha motivação e interesse por essa área só têm crescido, culminando então neste trabalho de conclusão de curso.

Aos meus colegas e amigos, que de todas as formas, acadêmicas ou não, me auxiliaram até aqui e com os quais espero seguir convivendo e trocando experiências, meu muito obrigado. E finalmente, agradeço a PUCRS, por toda assistência e infraestrutura fornecida.

Resumo

No presente trabalho é abordado o desenvolvimento de aplicações em um FPGA com arquitetura SoC (System on a chip). Este FPGA contém lógica programável, dois processadores ARM, e o barramento AXI que realiza a comunicação entre essas duas áreas. A ferramenta Vivado e o SDK, ambos fornecidos pela Xilinx, são utilizados para desenvolver as aplicações. A ferramenta Vivado é responsável por realizar configurações na área de processamento (ARM) e desenvolver IPs (*intellectual property*) na área da lógica programável. O SDK é um ambiente para desenvolver programas a serem executados na área de processamento. A integração do FPGA com um processador ARM cria a possibilidade de desenvolver aplicações embarcadas para os setores automotivo, médico, industrial, equipamentos de imagem e telecomunicações.

Palavras chaves: FPGA, SoC, Vivado, ARM.

ABSTRACT

This end-of-term work presents the development of applications on an FPGA with a SoC (System on a chip) architecture. This FPGA contains programmable logic, two ARM processors, and an AXI bus that performs the communication between these two areas. The Vivado and the SDK frameworks, both provided by Xilinx, are used to develop the applications. The Vivado framework is responsible for setting the processing area (ARM) and synthesize dedicated hardware IPs (intellectual property) in the programmable logic area. The FPGA integration with an ARM processor creates the possibility of developing embedded applications for the automotive sectors, medical, industrial, image equipment and telecommunications.

Keywords: FPGA, SoC, Vivado, ARM.

Lista de Figuras

Lista de Siglas

ARM - Advanced RISC Machine
ASIC - Application Specific Integrated Circuit
AXI - Advanced Extensible Interface
CLB - Configuration Logical Block
CPU - Central Processing Unit
DSP - Digital Signal Processor
ELF - Executable and Linkable Format file
FPGA - Field Programmable Gate Array
IDE - Integrated Design Environment
IP - Internet Protocol
ISO - International Standards Organization
LUT - Look- Up Tables
OSI - Open Systems Interconnection
PL - Programmable Logic
PS - Processing System
RAM - Random Access Memory
RISC - Reduced Instruction Set Computer
SDK - Software Development Kit
SoC - System on a chip
TCP - Transmission Control Protocol
UDP - User Datagram Protocol
VHDL - VHSIC Hardware Description Language
VHSIC - Very High Speed Integrated Circuits

Sumário

1. INTRODUÇÃO

ASIC, do inglês *Application Specific Integrated Circuit*, é um circuito integrado para uma aplicação específica, tendo por objetivo solucionar demandas do mercado. Entretanto, cada projeto exige alto investimento, requerendo meses ou anos para ser desenvolvido, e fica limitado a uma dada aplicação. O desenvolvimento de ASICs é justificado comercialmente com um alto volume de produção, ou aplicações muito específicas, como em medicina (por exemplo, marca-passo) ou em aplicações militares.

FPGA, do inglês *Field-programmable Gate Array*, é também um dispositivo que permite a implementação de aplicações específicas. Ao contrário de ASICs, a implementação em FPGAs não é permanente, podendo ser configurado de forma barata e fácil (no caso de FPGAs configurados a partir de uma memória). A vantagem do FPGA é sua baixa granularidade, i.e., a unidade de configuração é no nível de porta lógica. Desta forma, a partir de uma descrição HDL pode-se criar praticamente qualquer circuito digital. Assim como um computador, o FPGA executa milhões de operações simultâneas, podendo ser centenas de vezes mais rápido do que os projetos baseados em microprocessadores [DEH08] dada a possibilidade de se explorar o paralelismo espacial oferecido pela grande quantidade de elementos configuráveis presentes nos atuais dispositivos. FPGAs possibilitam também que erros de projetos sejam facilmente corrigidos e que atualizações de funcionalidades não se tornem custosas, tornando-os uma alternativa atraente para muitas aplicações, e particularmente indicado para o projeto de equipamentos eletrônicos com volume de produção não muito elevado.

Gordon Earl Moore, co-fundador da Intel, fez uma previsão que a cada 18 meses o número de transistores de um *chip* dobraria, e como resultado a capacidade de processamento também aumentaria. Esta previsão, conhecida como Lei de Moore [SCA97] vem se confirmando até hoje. Este incremento da capacidade de integração resultou em circuitos integrados (CIs) com milhões de transistores, possibilitando o desenvolvimento de um sistema completo em um único CI, resultando nos SoCs (*System on a Chip*), que contêm processadores, memórias e outros componentes, sendo encontrados em, por exemplo em *tablets*, *smartphones* e *GPS* [BOB07]

SoCs foram inicialmente projetados como ASICs e utilizados como plataformas de desenvolvimento para diversas aplicações. Exemplo de SoCs desta classe é a plataforma OMAP (Texas) [TEX14] Da mesma forma, os FPGAs são hoje SoCs, que embarcam no mesmo dispositivo processadores, periféricos e lógica programável.

Estes FPGAs podem ser utilizados em diversas aplicações de sistemas embarcados, como no setor automotivo, médico, industrial, equipamentos de imagem e telecomunicações, por exemplo. Compete ao meio acadêmico e a indústria realizar estudos e incentivos para utilizar a combinação: lógica programável com processadores embarcados.

1.1 Objetivos

O presente trabalho de conclusão de curso tem por objetivo estratégico, além de aplicar os conhecimentos adquiridos ao longo do curso, o *domínio em projeto de sistemas computacionais em dispositivos FPGAs com estrutura de SoC*.

Para atingir este objetivo estratégico, os objetivos específicos compreendem:

- Domínio da tecnologia FPGA (Xilinx) com processador embarcado (ARM);
- Domínio de ferramentas para o projeto em FPGAs do tipo SoC (Vivado);
- Desenvolvimento de uma aplicação que utilize o processador ARM e a lógica reconfigurável.

1.2 Estrutura do documento

O documento está organizado como segue. O Capítulo 2 contém o referencial teórico, com a finalidade de apresentar os temas necessários para o desenvolvimento do trabalho. Neste capítulo são abordados os seguintes conceitos: (*i*) a estrutura convencional de FPGAs, e os FPGAs modernos dos principais fabricantes, Xilinx e Altera; (*ii*) SoCs; (*iii*) camada OSI e protocolos. O Capítulo 3 apresenta o dispositivo Zynq 7020, hardware que foi utilizado no trabalho, sendo dividido em 3 seções: (*i*) FPGA Artix 7; (*ii*) processador ARM; (*iii*) barramento AXI. O Capítulo 4, descreve de forma sucinta a utilização da ferramenta Vivado. O Capítulo 5 apresenta a biblioteca para pilhas de protocolos LwIP. O Capítulo 6 detalha aplicações desenvolvidas ao longo do TCC. O Capítulo 7 conclui este TCC e aponta direções para trabalhos futuros. O final do texto contém um anexo, complementar ao Capítulo 4, o qual apresenta o desenvolvimento de um projeto com a ferramenta Vivado.

2. REFERENCIAL TEÓRICO

2.1 FPGAs modernos

O dispositivo FPGA foi introduzido no mercado em 1985 pela empresa Xilinx. Conforme a Figura 1, os primeiros FPGAs eram constituídos basicamente por [BOB07]

- CLBs ou *Configurable Logic Blocks* (blocos lógicos configuráveis): contendo elementos de processamento para a realização de operações lógicas, bem como flip-flops para a implementação de lógica sequencial. CLBs são implementadas com memórias e multiplexadores;
- *Interconnections* (interconexões) e *Switch matrix* (matriz de chaveamento): são fios e chaves programáveis para conectar os blocos lógicos entre si e com os blocos de entrada/saída;
- *Input / Output* (entrada/saída): interfaces configuráveis com os circuitos externos ao FPGA.

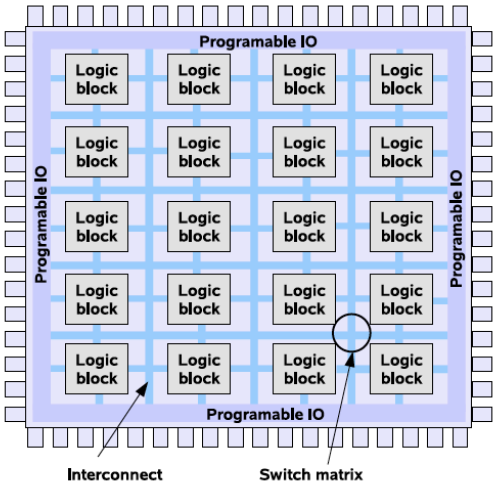


Figura 1 - Arquitetura básica de FPGAs [BOB07]

O primeiro FPGA comercial, XC2064, possuía 64 CLBs e interconexões configuráveis entre os blocos lógicos. Cada CLB continha duas LUTs (*Look-Up Tables*) de 3 entradas. Uma LUT é um bloco de hardware, que contém todos os possíveis resultados de uma dada função para um dado conjunto de valores de entrada. Uma LUT pode ser vista como uma tabela verdade, ou seja, o valor de saída corresponde ao acesso à uma linha da tabela verdade.

Com a evolução da tecnologia de fabricação de semicondutores e na fabricação de FPGAs, foram adicionados novos recursos, como: blocos DSPs (*digital signal processing*), contendo multiplicadores físicos de 18 bits; PLLs (*phase-locked loop*) permitindo realizar multiplicação e divisão da frequência do sinal de relógio, além de minimizar o *skew* do mesmo; DLL (delay-locked loop) opera de forma semelhante ao PLL, porém com implementação digital; *transceivers* de alta velocidade, que permitem ao FPGA receber dados em alta frequência, paralelizando-os para uso pela lógica programável.

Os dois grandes fabricantes de FPGAs são Xilinx e Altera. Segundo uma pesquisa feita pela UBM Electronics sobre embarcados, 64% dos projetos utilizam Xilinx, enquanto 42% utilizam Altera e a terceira colocada é a Lattice com 10% de uso em projetos [UBM13] Estes dados levam em consideração que os usuários de FPGAs podem ter escolhido um ou mais fabricantes para execução dos seus projetos.

2.1.1 Xilinx

As famílias de FPGAs Xilinx compreendem Spartan 6, Artix 7, Kintex 7, Virtex 7, Kintex UltraSCALE e Virtex UltraScale, sendo esta última a mais recente, com transistores FinFET de 16 nm [XIL14a] O último modelo da série Virtex UltraScale tem como recursos: 4.407.408 *logic cells*, 5.037.120 Flip-Flops, 2.518.560 LUTs, 2.880 DSP, interface PCI Express e interface com memória DDR3.

A arquitetura dos FPGAs Virtex UltraScale é organizada em um *layout* de colunas de recursos, onde são combinadas em diferentes proporções para prover uma capacidade otimizada para a densidade do dispositivo, sua aplicação e custo. A Figura 2 mostra uma visão de recursos agrupados.

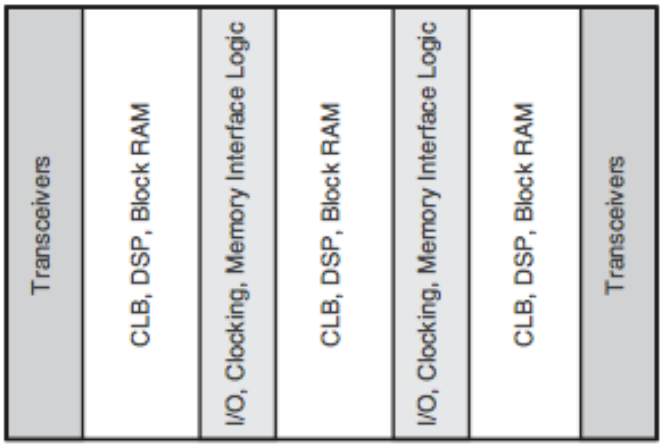


Figura 2 - FPGA com coluna de recursos [XIL14b].

Os recursos no FPGA são divididos em domínios de relógio, conforme apresentado na Figura 3. A altura de um domínio de relógio é de 60 CLBs. Um banco de 52 I/Os (entradas/saídas), 24 DSP *slices*, 12 blocos de RAMs, ou 4 canais de *transceivers* também equivalem como a altura de um domínio de relógio. A largura de um domínio de relógio é a mesma em todos os casos, independentemente do tamanho do dispositivo ou da combinação de recursos na região, permitindo que os resultados de *timing* sejam repetíveis [XIL14b] Esta característica é importante para que um hard-IP (módulo IP já posicionado e roteado) possa ser posicionado em diferentes regiões do FPGA.

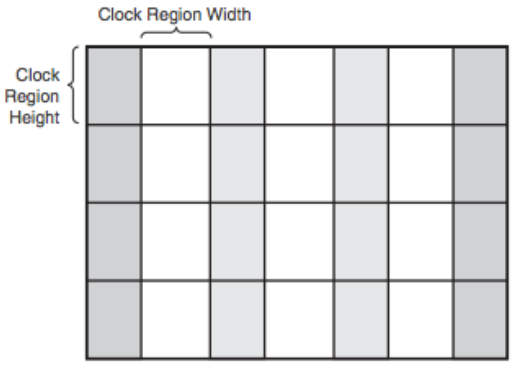


Figura 3 - FPGA dividido em domínios de relógio [XIL14b]

2.1.2 Altera

Dispositivos Altera possuem três famílias, Cyclone, Arria e Stratix. A última possui o modelo Stratix 10 como o mais recente, fabricado com tecnologia de 14 nm *tri-gate* [ALT14a] [ALT14b] semelhante ao FinFET da Xilinx.

A arquitetura Stratix, Figura 4, é constituída por elementos lógicos na vertical, blocos de

memória TriMatrix que são compostos por três tamanhos de blocos de RAM, cada um dos quais pode ser configurado para suportar uma ampla gama de funções. Blocos DSPs, e PLLs que são cercadas por elementos de I/Os, conforme ilustrado na Figura 4. Dispositivos desta série são baseados em *MultiTrack* interconectado com *DirectDrive*, o primeiro consiste em, linhas contínuas de roteamento otimizado para desempenho de diferentes comprimentos utilizados para a comunicação dentro e entre os blocos de projeto distintos. O segundo é uma a tecnologia de roteamento proprietário, que garante o uso de recursos de roteamento idêntico para qualquer função, independentemente do seu posicionamento dentro do dispositivo. Esta tecnologia simplifica a fase de integração de sistemas de projetos baseados em blocos programáveis [ALT14b]

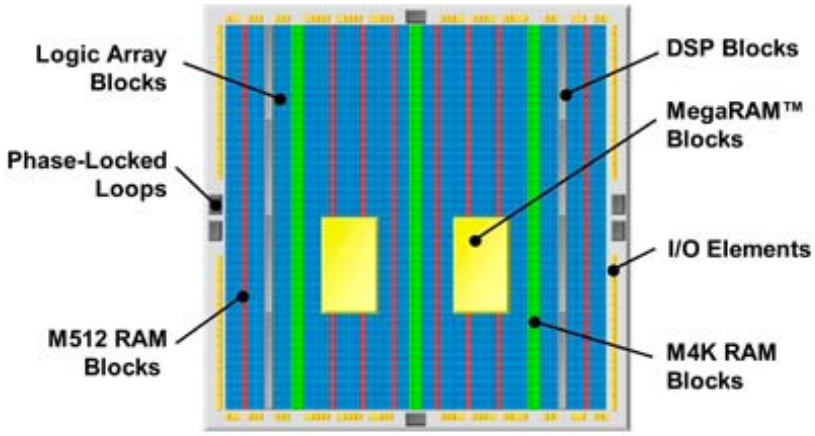


Figura 4 - Arquitetura Stratix Altera [ALT14b]

2.2 SoCs

SoC (*System on a chip*), é onde em um único circuito integrado pode-se ter um sistema computacional completo, incluindo processador, memória, circuitos para áudio e vídeo, além da arquitetura de interconexão. Adicionando lógica programável, cria-se a possibilidade de fazer modificações na estrutura do hardware, como adicionar novos módulos ao SoC. Estes novos módulos podem ser vistos como aceleradores, auxiliando os processadores embarcados em tratamentos que requerem alta capacidade de processamento. É através de um barramento que é feita a comunicação do sistema de processamento com o sistema programável. Um exemplo deste tipo de arquitetura é apresentado na Figura 5.

SoCs possuem blocos lógicos de dados previamente projetados que são utilizados no projeto de uma aplicação, chamados *Intellectual Property* (IP). Os IPs tem o objetivo de facilitar o projeto, já que são portáveis e reutilizáveis. Exemplos de IPs são processadores e barramentos. Os IPs podem ser classificados de três formas [RAJ00]

- *Hard:* módulos já caracterizados, não permitindo alterações por parte dos projetistas. É menos flexível do que os outros dois tipos de *cores* citados abaixo. Normalmente corresponde a um bloco já posicionado e roteado;
- *Firm*: semelhantes ao *hard*, mas são configuráveis para diversas aplicações. Corresponde a um *netlist* (conjunto de portas lógicas e interconexões que compõem um circuito integrado) resultante da síntese logica para uma tecnologia específica;
- *Soft*: é o mais flexível dos três, podendo ser tanto um *netlist* não sintetizado para uma tecnologia específica ou uma descrição em HDL, como Verilog ou VHDL.

As vantagens da utilização de SoCs incluem: (1) redução do tempo de projeto; (2) menor número de componentes eletrônicos no sistema final, reduzindo custo e aumentando a confiabilidade do sistema; (3) maior desempenho, dado que todos os componentes estão dentro do mesmo circuito integrado.

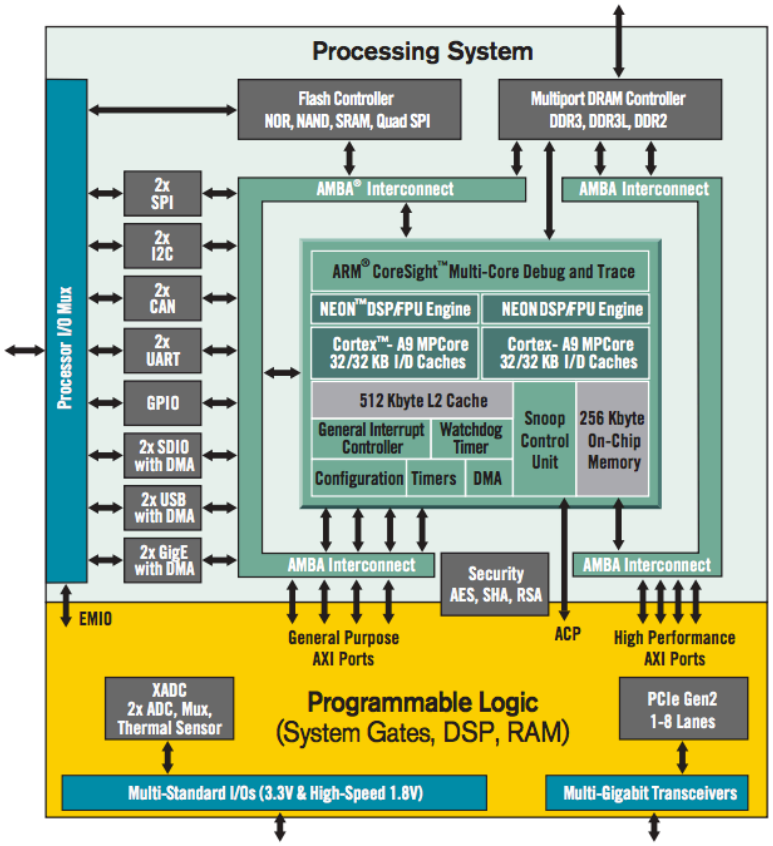


Figura 5 - Integração *Processing System* e *Programmable Logic* [XIL14c]

2.3 Protocolos de Comunicação

Para a execução deste TCC foi necessária a utilização de protocolos de rede, pois são utilizados como estudo de caso duas aplicações que realizam tratamento de pacotes de rede. Portanto, foi feito um estudo da pilha de protocolos tendo por referência o modelo o OSI (*Open Systems Interconnection*).

O modelo OSI, Figura 6, se baseia em uma proposta desenvolvida pela ISO (*International Standards Organization*) com o objetivo de uma padronização internacional dos protocolos empregados nas seguintes camadas [TAN03]

- Física (*Physical*): trata da transmissão de bits por um canal de comunicação, deve garantir que o bit enviado será o mesmo bit recebido pela outra interface;
- Enlace (*Data link*): detecta e corrige erros que possam acontecer na camada anterior. É responsável pela transmissão e recepção de quadros e pelo controle de fluxo;
- Rede (*Network*): responsável por controlar o roteamento dos pacotes da origem ao destino;
- Transporte (*Transport*): receber os dados da camada superior, divide-os em unidades menores caso necessário, repassa para a camada de rede e garante que todos os dados chegarão corretamente a outra extremidade;
- Sessão (*Session*): permite que os usuários de diferentes *hosts* estabeleçam sessões entre eles;
- Apresentação (*Presentation*): relacionada a sintaxe e a semântica das informações transmitidas;
- Aplicação (*Application*): voltada a uma série de protocolos comumente necessários para usuários.

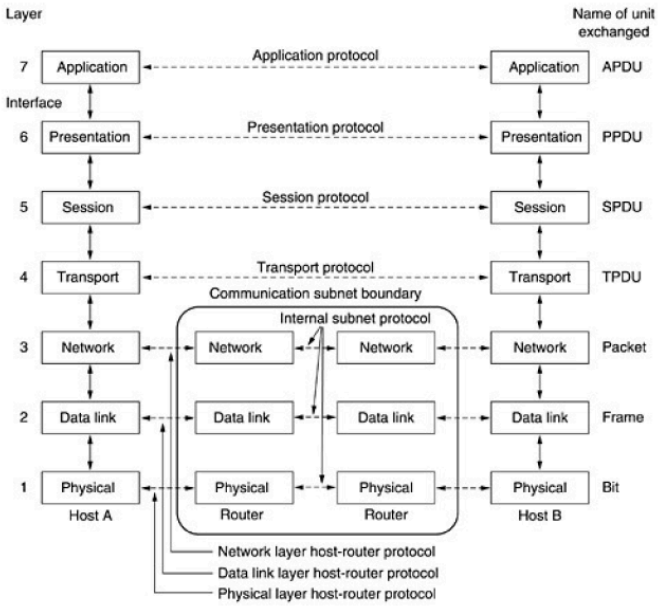


Figura 6 - Modelo OSI [TAN03]

2.3.1 Frame Ethernet

O frame Ethernet trafega na camada de enlace. Este frame é ilustrado na Figura 7. Os campos do frame são:

- *Preamble*: 7 bytes, realiza a sincronização entre o *clock* do receptor e o *clock* do transmissor;
- *Start Frame Delimiter*: delimitador de início de quadro;
- *Destination address* e *Source address*: endereço de origem e destino, respectivamente, formado por 6 bytes cada campo;
- *Length*: indica o tamanho do frame, excluindo-se os campos *preamble* e delimitador;
- *Data:* com até 1500 bytes, valor fixado em 1978 devido a limitações em relação a RAM que um receptor possuía. Neste campo também esta a identificação do que o receptor deve fazer. Existe também um comprimento mínimo de quadro, que é no mínimo 64 bytes, do campo endereço de destino até o campo de verificação. Não havendo este tamanho mínimo, existe o campo PAD, para preencher o quadro até o tamanho mínimo;
- *Check-sum*: responsável por detectar erros.

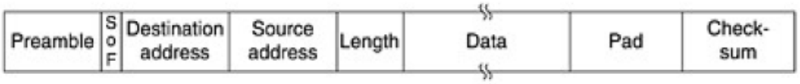


Figura 7 - Frame Ethernet [TAN03]

A descrição feita até aqui, se refere ao padrão estabelecido pelo IEEE, porém já existia um padrão criado pela DIX (DEC, Intel, Xerox). No padrão DIX, o campo *Preamble* possuía 8 bytes, contendo o padrão de bits 10101010. A outra diferença era que o campo *Length* se chamava *Type,* e nele que estava a identificação do que o receptor deveria fazer com o *frame*.

2.3.2 Datagrama IPV4

O pacote IP trafega pela camada de rede. O datagrama é dividido em um *Header* (cabeçalho) de 20 bytes e um campo *Payload* ou *Data*(dados), Figura 8. O Header tem como campos:

- *Version*: informa a versão do protocolo a qual o datagrama pertence;
- *Header Length* (HLEN): informa o tamanho do cabeçalho com o numero de palavras de 32 bits;
- *Service Type*: é destinado a distinguir entre diferentes classes de serviços;
- *Total Length*: define todo o tamanho do datagrama, incluindo cabeçalho e dados, em bytes de oito bits. O datagrama de tamanho mínimo é de vinte bytes e o máximo é 64 Kb;
- *Identifcation*: campo de identificação, este campo é usado principalmente para identificar fragmentos do datagrama IP original;
- *Flags*: o campo de três bits que segue é usado para controlar ou identificar fragmentos. Um bit não é utilizado, e dois campos de 1 bit, DF (*Don't Fragment*) indica para não fragmentar o datagrama, e MF (*More Fragments*) informa que há mais fragmentos;
- *Fragment Offset*: informa a qual ponto do datagrama atual o fragmento pertence;
- *Time to Live* (TTL): contador utilizado para limitar a vida do datagrama, prevenindo que os datagramas persistam (ex. andando aos círculos) numa rede.
- P*rotocol* : informa a que processo de transporte o datagrama deve ser entregue, como ICMP e TCP;
- *Header checksum* : confere apenas o cabeçalho, verificando possíveis erros;
- *Source IP Address* e *Destination IP Address*: informam o número da rede e o número do host, origem e destino respectivamente;
- *Options*: foi projetado para permitir versões posteriores do protocolo que incluam informações inexistentes no projeto original;
- *Padding*: garante que o cabeçalho será um múltiplo de 32 bits, para atender o campo *HLEN*.

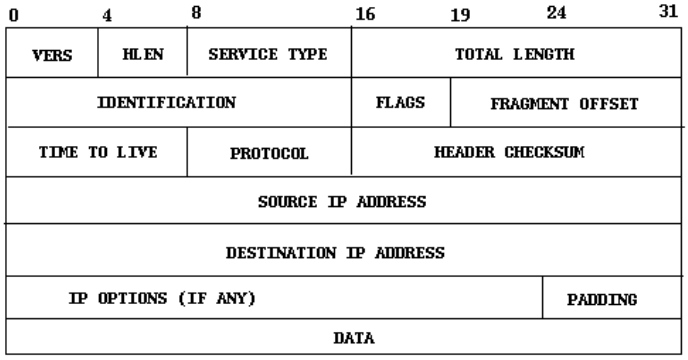


Figura 8 - Datagrama IP [TAN03]

2.3.3 Segmento TCP

O TCP (*Transmission Control Protocol*) foi projetado para oferecer um fluxo de bytes fim a fim confiável em uma rede operando na camada de transporte. A camada de rede não oferece qualquer garantia de que os datagramas serão entregues da forma apropriada, portanto, cabe ao TCP retransmiti-los sempre que necessário. O TCP também tem de reorganizá-los em mensagens na sequência correta [TAN03]

O segmento começa com um *Header* (cabeçalho) de formato fixo de 20 bytes, podendo ser seguido por opções de cabeçalho. Depois das opções, pode haver até 65.535-20-20= 65.495 bytes de dados, onde o primeiro valor de 20 se refere ao *Header* do IP e o segundo ao *Header* do TCP, Figura 9. O cabeçalho, ilustrado na Figura 9, possui os campos *Source port* e *Destination port* que identificam a aplicação de origem e destino, respectivamente; *Sequence Number* que identifica a posição deste segmento no fluxo de dados; *Acknowledgement Number* utilizado para

confirmar o recebimento de segmentos enviados anteriormente e especifica o próximo segmento aguardado. O campo TCP *header length* informa quantas palavras de 32 bits existem no cabeçalho, logo a seguir há o espaço reservado para testes e novas implementações; os próximos 6 bits são referentes a *flags* que possuem informações de conexão. No campo *Window size* é onde feito o controle de fluxo, indicando quantos bytes podem ser enviados a partir do byte confirmado.

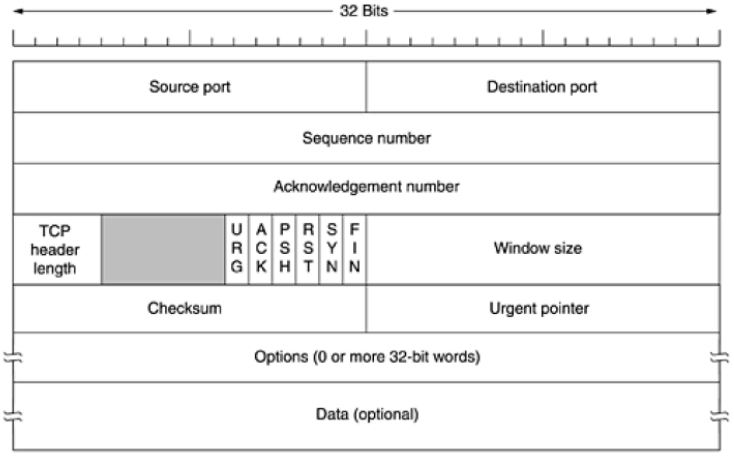


Figura 9 - Segmento TCP [TAN03]

O *Checksum* contém as informações usadas na verificação de erros do protocolo. O campo *Urgent Pointer* é usado pela origem para indicar onde se encontra algum dado urgente dentro do segmento. O campo *Options* foi projetado com uma forma de oferecer recursos extras.

2.3.4 Segmento UDP

UDP (*User Datagram Protocol*), Figura 10, oferece um meio para as aplicações enviarem datagramas IP encapsulados sem que seja necessário estabelecer conexão. Possui dois campos *Source port* e *Destination port*, que são a porta de origem e a de destino, respectivamente. Sem estes campos a camada de transporte não saberia o que fazer com o pacote. O campo UDP *Length* inclui o cabeçalho de 8 bytes e os dados. O UDP *Checksum* é opcional e armazenado como 0 se não for calculado [TAN03]

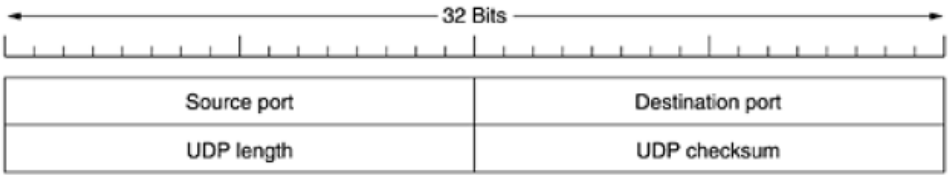


Figura 10 - Segmento UDP [TAN03]

3. DISPOSTIVO ZYNQ 7020

Este dispositivo pertence à família Zynq 7000 da Xilinx, que une um PS (*processing system*), ou sistema de processamento, com PL (*programmable logic*), ou lógica programável, no mesmo dispositivo. O PS possui um dual-core ARM Cortex-A9 MP e o PL utiliza o FPGA Artix-7. A conexão entre eles é feita pelo barramento AXI4 (*Advanced eXtensible Interface*), pertencente à família AMBA (*Advanced Microcontroler Bus Architecture*) [XIL14c]

3.1 ARM - Sistema de Processamento(PS)

O processador ARM é o centro do PS. A arquitetura utilizada neste processador é RISC (*Reduced Instruction Set Computer*), possuindo um grande número de registradores, um conjunto de instruções simples, com ênfase na otimização do *pipeline* de instruções [STA11] Outras características relevantes do processador [XIL13a] [ARM14] são:

- Frequência de operação até 866 MHz;
- Capacidade de operar em um único processador ou utilizando os dois, de modo simétrico ou assimétrico;
- NEON, é um conjunto de instruções do tipo SIMD (*Single Multiple Data*) para acelerar aplicações multimídias e de processamento de sinais;
- THUMB 2, Thumb são instruções ARM codificadas para aumentar o desempenho do processador em certas aplicações, ela estende o limite de 16 bits do modo Thumb com algumas instruções adicionais de 32 bits;
- Cache L1 de 32 Kb;
- MMU (memory management unit) integrado;
- ACP (*Accelerator coherency port*), interface coerente permitindo acessos do PL para o espaço de memória da CPU;
- Cache L2 de 512 Kb;
- 8 canais DMA (*Direct Memory Acess*), sendo 4 canais dedicados ao PL;
- Interrupções e *Timers*.

3.2 Artix 7- Lógica Programável(PL)

A lógica programável adota a arquitetura do FPGA Artix 7, possuindo como principais características [XIL13a]

- 8 LUTs por CLB para implementação de lógica combinacional ou de memória distribuída;
- LUTs podem ser configuradas como blocos de memória 64x1 ou 32X2 bits, registradores de deslocamento (SRL), ou somadores 2x4-bit em cascata para funções aritméticas;
- Cada CLB contém 16 flip-flops;
- Blocos de memória com capacidade de 36 Kb;
- DSP *slices*, contendo um multiplicador 25x18, com um acumulador de 48 bits;
- Tecnologia SelectIO com suporte a DDR3 e interface de até 1.866 Mb/s;
- Conectivade serial de alta velocidade com transceptores multi-gigabit embutidos a partir de 600 Mb/s e de taxas máximas de 6,6 Gb/s até 28,05 Gb/s, oferecendo um modo especial, de baixa potência, otimizado para interface de chip para chip;
- *User Configurable Analog Interface* (XADC), interface analógica configurável;

- Interface PCI Express, para até 8 vias.

3.3 Barramento AXI-Lite

A Xilinx adotou o protocolo AXI (*Advanced eXtensible Interface*) para IPs a partir dos FPGAs Spartan-6 e Virtex-6 e o continua a utilizar para as séries 7 e Zynq 7000. AXI faz parte da família de barramentos para os processadores ARM, sendo a última versão AXI4. Há três tipos de interfaces [XIL12]

- AXI4: para interfaces de memória de alto desempenho;
- AXI4-Lite: barramento de menor desempenho (por exemplo, para controle e status de registradores);
- AXI4-Stream: para alta velocidade de dados *streaming* (por exemplo, para vídeo em alta resolução).

O limite de transferência no AXI4 é de uma rajada de até 256 dados por transação. O AXI4-Lite permite somente 1 dado transferido por transação, e o AXI4-Stream suporta uma rajada com tamanho ilimitado de dados [XIL12]

Neste TCC utilizo a versão AXI4-Lite, os canais disponíveis na arquitetura são [ARM13]

- *Read Address Channel* (canal de leitura de endereço);
- *Write Address Chanel* (canal de escrita de endereço);
- *Read Data Channel* (canal de escrita de dado);
- *Write Data Channel* (Canal de leitura de dado);
- *Write Response Channel* (canal de resposta de escrita).

O *Read Address Channel* é onde o IP envia um sinal informando que deseja iniciar uma transação de leitura. Este canal traz informações de endereçamento e alguns sinais *handshaking*. O *Read Data Channel* contém os dados que são transferidos durante uma operação, juntamente com os sinais de *handshaking* associados (Figura 11).

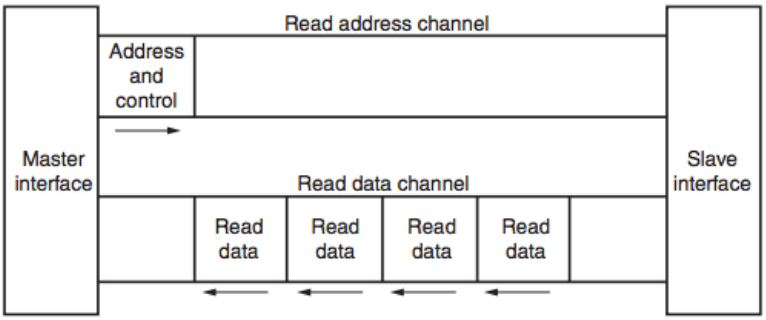


Figura 11 - Arquitetura do canal de leitura do AXI-lite[XIL12]

O *Write Address Chanel* é a via utilizada pelo IP para sinalizar que deseja iniciar uma transação de escrita. Ele é idêntico ao *Read Address Channel*. O *Write Data Channel* executa uma função equivalente ao *Read Data Channel*, exceto que os dados são oriundos do *master*. O *Write Response Channel* é utilizado pelo *slave* para avisar a recepção dos dados, ou para indicar que ocorreu um erro (Figura 12).

Os sinais de *handshaking* presentes em todas as transações de *read* e *write*, são importantes para o controle dos dados que trafegam nos canais disponíveis. Os sinais são baseados em um princípio de *ready* e *valid* ou seja “pronto” e “válido”. O sinal *ready* é usado pelo *slave* para indicar que está pronto para aceitar a transferência de dados ou endereçamento, e *valid* é usado para informar que o dado emitido naquele canal é válido, para que então possa ser utilizado.

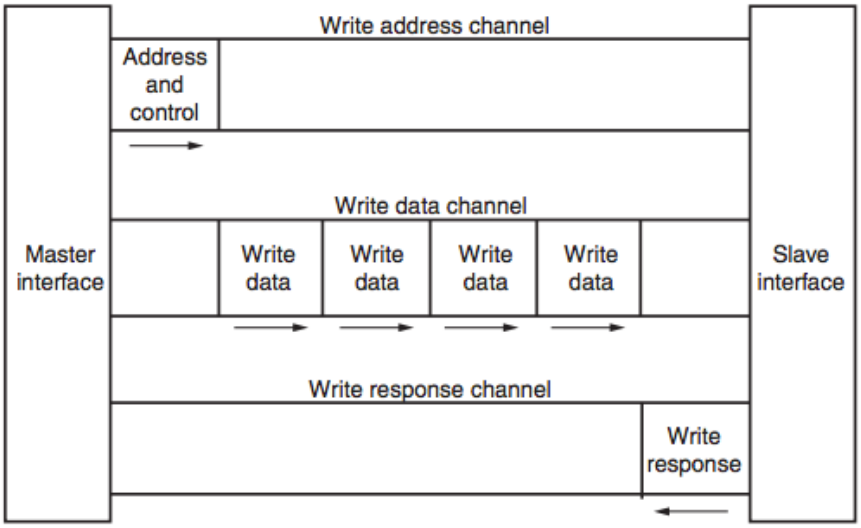


Figura 12 - Arquitetura do canal de escrita AXI-lite[XIL12]

4. FERRAMENTA DE EDIÇÃO DE HARDWARE E SOFTWARE - VIVADO

É o conjunto mais recente de ferramentas oferecidas pela Xilinx. A suíte Vivado possibilita o desenvolvimento de aplicações em nível de hardware e software. O projeto deve ser iniciado pelo Vivado IDE (*Integrated Design Environment*), e então o hardware criado é exportado para o Xilinx SDK (*Software Development Kit*). Esta ferramenta vem para substituir e integrar as funcionalidades do PlanAhead e do Chipscope (utilizado para testes de sinais no próprio hardware que foi prototipado).

A Figura 13 ilustra o fluxo de projeto para implementação de um projeto no Vivado IDE e as informações contidas no arquivo que são exportadas para o Xilinx SDK. A suíte possui uma gama de opções de IP (*Intellectual Property*) e permite a implementação dos próprios IPs.

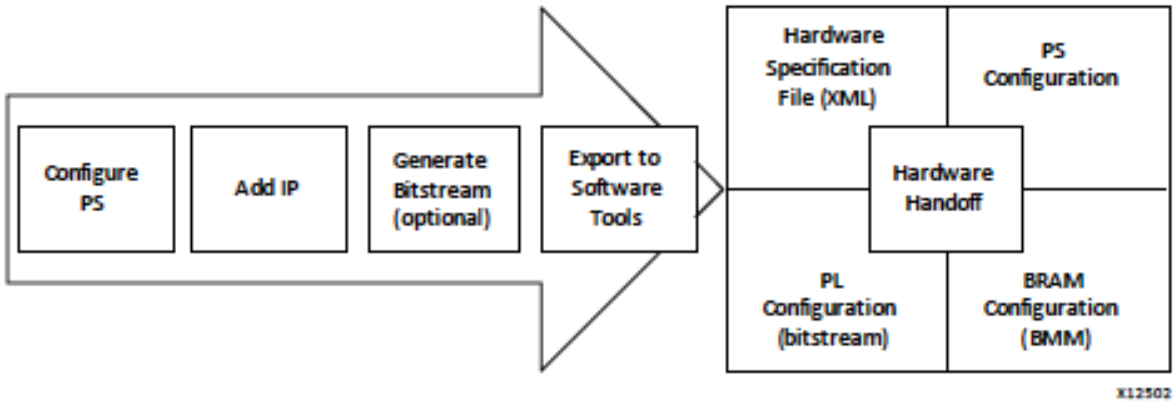


Figura 13 - Fluxo do projeto de Hardware e informações contidas no arquivo exportado para desenvolver o software [XIL13b]

O primeiro passo do fluxo é a configuração do PS no Vivado, onde são configuradas as opções de memória, a frequência que o processador vai operar, os periféricos disponíveis no hardware que vão ser instanciados, as interfaces de comunicação com o barramento AXI e as interrupções. Na etapa de inclusão de IPs, podem ser adicionados IPs disponíveis no catálogo do Vivado ou adicionado um IP desenvolvido especificamente para o projeto. Após validado o projeto é gerado o *bitstream* e então exportado para o SDK.

O SDK possui especificações do *hardware* exportado, como periféricos disponíveis e seus respectivos endereçamentos. Com base neste projeto é desenvolvido um *software* que vai se comunicar com o processador. Após concluído e gerado um arquivo com a extensão .elf para ser enviado para placa juntamente com o arquivo .bit referente à etapa de hardware.

No anexo ao final deste documento há um tutorial que apresenta o ambiente Vivado com suas particularidades, através de um projeto simples de escrita e leitura nos registradores de um IP personalizado.

O domínio do ambiente Vivado, com utilização do PS e de periférico desenvolvido para um dado projeto, é uma importante contribuição deste TCC. O tutorial presente no anexo será uma importante fonte de referência para novos trabalhos utilizando esta tecnologia.

5. BIBLIOTECA LWIP PARA PILHA DE PROTOCOLOS

O objetivo da implementação da biblioteca *Lightweight Internet Protocol* (LwIP) TCP/IP é o de reduzir o uso de recursos de memória. Isso faz com que LwIP seja adequado para uso em sistemas embarcados [SAV14].

Os protocolos suportados pela biblioteca LwIP são: ARP, IP, TCP, UDP, DNS, SNMP, DHCP, ICMP, IGMP, PPP e PPPoE.

São oferecidas três diferentes APIs [LWI14a]:

- Raw: é o núcleo da LwIP. Utiliza *callbacks* para lidar com eventos. Ideal para aplicações sem sistemas operacionais. Alto desempenho e pouco uso de memória.
- NetConn: construída sobre a Raw. Ele permite operação *multi-threaded* e, portanto, requer um sistema operacional. Desempenho mais baixo em comparação com a *raw* e maior consumo de memória.
- BSD socket: construída sobre NetConn. Seu objetivo é a portabilidade.

O protocolo e a API utilizada na aplicação de bloqueio de palavras (Seção 6.2) foram respectivamente o TCP e a *raw*. As etapas de configuração e recebimento de um pacote TCP incluem as funções descritas abaixo [LWI14b].

- Funções para a configuração de uma conexão TCP:
  - **tcp_new**: utilizada para criar uma nova conexão PCB (*Protocol Control Block*). A PCB é uma estrutura utilizada para armazenar o estado de uma dada conexão.
  - **tcp_bind**: vincula o PCB a um número de endereço de rede e uma porta local.
  - **tcp_listen**: habilita uma estrutura PCB para *ouvir* as conexões de entrada.
- Funções para a receber dados através de uma conexão TCP:
  - **tcp_recv**: define a função de callback que será chamada quando novos dados chegam na interface de rede.
  - **tcp_recved**: informa ao núcleo LwIP que o aplicativo processou os dados.

A estrutura utilizada para gerenciar os dados na memória pelo LwIP se chama *pbuf*, sendo uma lista encadeada de buffers projetada para receber um pacote de rede, conforme apresentado na Figura 14.

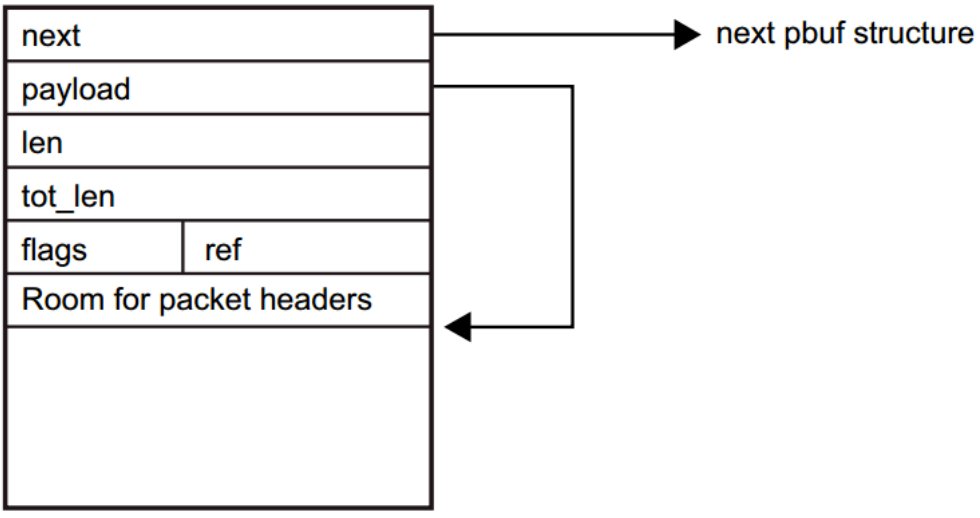


Figura 14 - Estrutura de um *pbuf* [STM14]

Os campos que estão presentes em cada *pbuf* são:

- next: ponteiro para o próximo *pbuf* da lista;

- payload: ponteiro para o dado;
- len: tamanho do payload;
- tot_len: soma do campo len com todos elementos da lista;
- flags: tipo de *pbuf*;
- *ref*: contagem de referência que indica o número de ponteiros, um pbuf só pode ser liberado se a sua contagem é zero.

Os tipos de *pbuf* que podem ser definidos são:

- PBUF_POOL: é o tipo pbuf utilizado para a recepção de pacotes uma vez que fornece a alocação mais rápido.
- PBUF_RAM: pbuf são alocados dinamicamente a partir de uma área de memória continua. A alocação é PBUF_RAM é mais lento do que o PBUF_POOL e pode levar à fragmentação da memória.
- PBUF_ROM: pbuf usado para passar dados constante por referência.

Na aplicação de bloqueio de palavras é utilizado o tipo PBUF_POOL. Segue abaixo um código exemplo que faz a configuração da conexão e o recebimento de pacotes, semelhante ao utilizado no projeto de bloqueio de palavras. Neste código foram omitidos os tratamentos que devem ser feitos em caso de erros no estabelecimento da conexão ou no recebimento de dados.

```
#include "lwip/tcp.h”

struct tcp_pcb *pcb;
struct pbuf *p;
unsigned port = 28;

/*************************** Funções de configuração de conexão **********************/
// iniciando uma estrutura PCB para receber os pacotes TCPs
pcb= tcp_new();

// vincula o PCB a um IP e uma porta
tcp_bind(pcb, IP_ADDR_ANY, port);

//habilitaa estrutura para escutar as conexões
tcp_listen(pcb);

//define a função que deve ser chamada quando houver uma conexão estabelecida
tcp_accept(pcb, tcp_recv);

/*************************** Funções de recebimento de pacote **********************/
//especifica a função que deve ser chamada quando o TCP recebe um dado
tcp_recv(newpcb, tcp_recved);

//função chamado para processar os dados
tcp_recved(tpcb, p->len);
```

6. APLICAÇÕES DESENVOLVIDAS

As etapas iniciais do TCC consistiram em: (i) compreensão da arquitetura do dispositivo Zynq 7200; (ii) domínio das ferramentas de projeto; (iii) compreensão da comunicação entre o processador ARM e o barramento AMBA AXI4-Lite. A etapa seguinte do TCC corresponde ao desenvolvimento de aplicações que exemplificam o uso desta tecnologia, descritas neste capítulo.

Para criar uma aplicação que utilize estes conhecimentos, foi necessário ter o domínio da suíte de desenvolvimento da Xilinx, onde foi utilizada a linguagem de descrição de *hardware* VHDL e a linguagem de programação C. Depois de concluída a aplicação, ela foi prototipada na placa Zedboard Zynq 7020 (Figura 15).

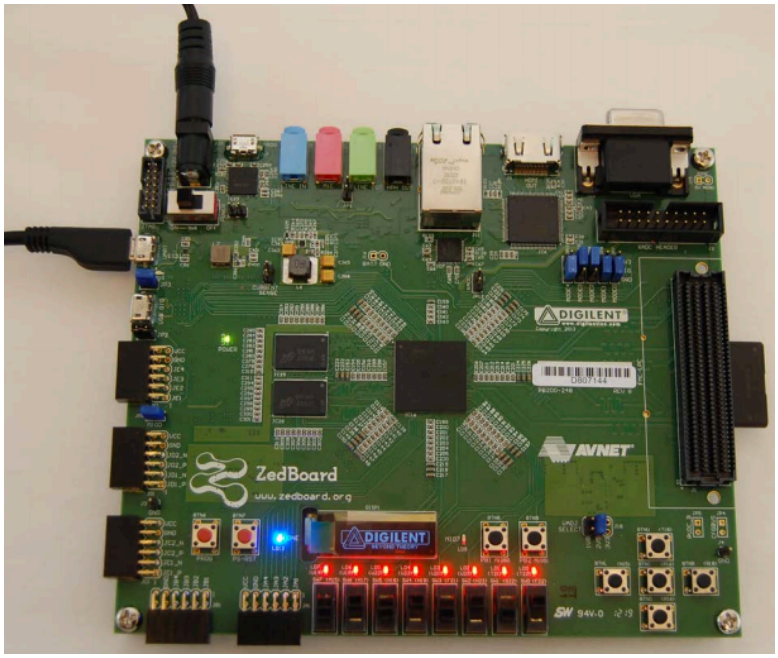


Figura 15 - Placa Zynq-7020 para a prototipação [ZED14].

A Figura 16 ilustra como está configurado o ambiente de desenvolvimento. Um microcomputador (PC) possui instalado a suíte da Xilinx, a qual contém as ferramentas Vivado IDE e o Xilinx SDK. O Vivado IDE envia para a placa o arquivo no formato *bitstream*, que configura o hardware do projeto. O Xilinx SDK envia o arquivo ELF, que é o software que executa no processador ARM. Estes arquivos são gerados depois de implementado o hardware e compilada a aplicação. Também neste ambiente é feita a verificação do comportamento do que foi enviado para a placa através das opções de debug de cada software. No Vivado é verificado o comportamento interno do hardware, através da captura dos valores de sinais internos do IP. No Xilinx SDK verifica-se cada etapa de execução do software desenvolvido em C. O sistema embarcado é autônomo, composto pela placa Zedboard e uma interface com a rede Ethernet, que no ambiente de teste é simulada por um computador que envia pacotes para o *hardware*. Depois de configurada a placa, ela opera sem a necessidade de iterações com o PC.

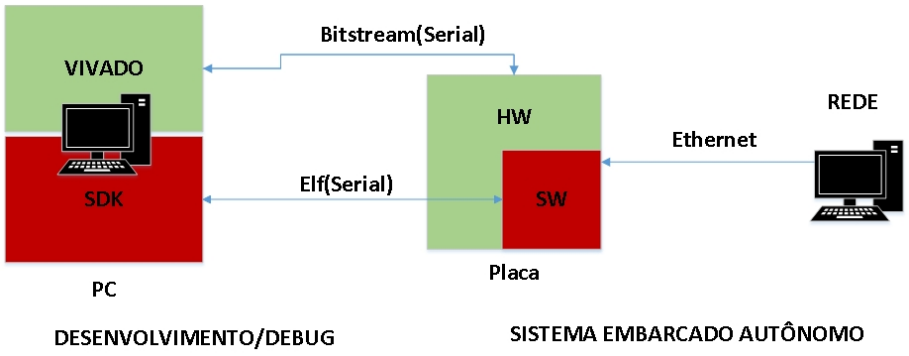


Figura 16 - Ambiente de desenvolvimento e teste.

Nas aplicações desenvolvidas, o barramento AXI4-Lite é utilizado para transportar os dados do processador ARM para os registradores do IP que está localizado na lógica programável. O IP gera um sinal de interrupção para o processador receber por este mesmo barramento os dados do PL.

6.1 Aplicação para redirecionar pacote de rede para o PL

A primeira aplicação desenvolvida recebe pacotes da rede através de um emulador de pacotes, e desvia-os diretamente para o PL fazer um dado tratamento, e posteriormente enviar o resultado deste processamento para o processador. A referência para esta aplicação é um projeto disponível publicamente [XIL13c], o qual faz o redirecionamento de pacotes, mas com pacotes gerados internamente ao FPGA (pelo processador). Este projeto foi desenvolvido originalmente [XIL13c] para a ferramenta PlanAhead, ferramenta da Xilinx anterior ao Vivado,

O primeiro passo do desenvolvimento consistiu em portar a aplicação para a ferramenta Vivado, conforme apresentado na Figura 17.

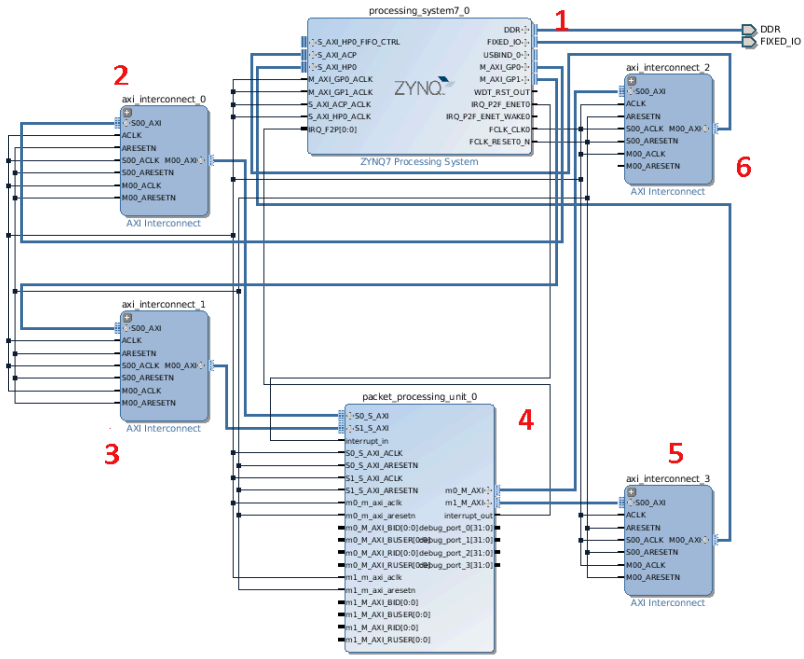


Figura 17 – Arquitetura de blocos do projeto no Vivado.

O projeto é composto por quatro interconexões AXI, duas referentes à comunicações PS (*master*) para PL (*slave*) e duas PL (*master*) para PS (*slave*):

- O módulo 1 é onde está localizada a área de processamento.
- O módulo 2 conecta a porta *master* 0 do PS (módulo 1) com a porta *slave* 0 do *packet processing unit* (módulo 4), este sendo responsável pelo controle de endereçamento, tamanho e status do pacote.
- O módulo 3 conecta a porta *master* 1 do PS (módulo 1) com a porta *slave* 1 do *packet processing unit* (módulo 4), este sendo responsável por inspecionar e visualizar o tipo de protocolo.
- O módulo 4 é o IP localizado na área da lógica programável, sendo responsável por dividir o pacote recebido em duas partes, enviando o cabeçalho para o canal *master* 0 e o restante do pacote para o canal *master* 1.
- O módulo 5 conecta a porta *master* 1 do módulo 4 à porta HP (*High-Performance*) do módulo 1. Esta porta oferece um alto desempenho no tráfego da área logica para a de processamento.
- O módulo 6 conecta o canal *master* 0 do módulo 4 à porta ACP (*accelerator coherency port*) do módulo 1. Esta é uma porta que tem acesso a área de processamento com uma baixa latência.

No IP existem duas interfaces de interrupção, uma de entrada que é utilizada para avisar quando um pacote estiver pronto para ser “processado”, e uma de saída que avisa a área de processamento quando está concluída a verificação no pacote [XIL13c].

Para a arquitetura receber pacotes diretamente da rede Ethernet foi desenvolvido um driver para a interface de rede, onde os pacotes são recebidos e identificados. Este driver faz a configuração da interface e cadastra um MAC (1 na Figura 18), e aguarda pacotes gerados pela rede. São identificados dois tipos de pacotes, um gerado pela rede que é referente ao DHCP e um ARP emulado para verificar se a interface recebe um pacote gerado artificialmente (2 na Figura 18).

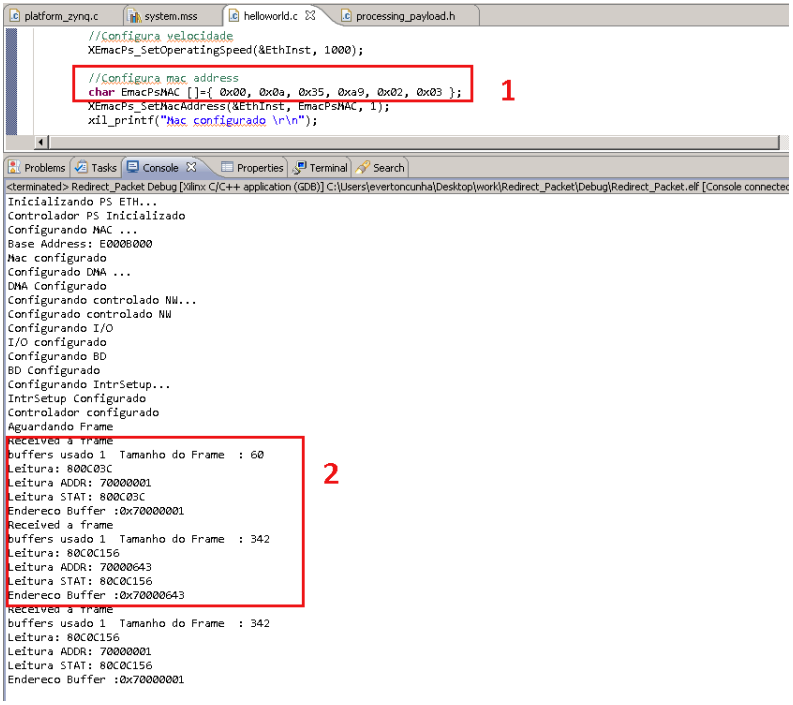


Figura 18 – Recebimento de pacotes ARP visualizados no SDK.

A Figura 19 apresenta uma tela do Wireshark, uma ferramenta de análise de tráfego de rede, onde há confirmação da recepção do pacote ARP, com o filtro habilitado para visualizar somente o MAC referente à interface da placa da Zynq (1 na Figura 19), o pacote (2 na Figura 19) e confirmando o destino, origem e o tipo de pacote (3 na Figura 19).

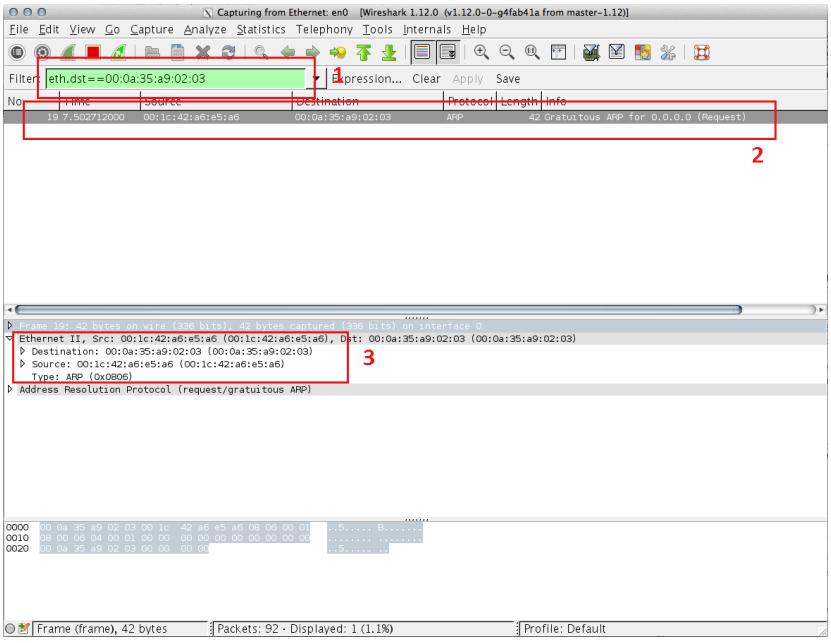


Figura 19 – Visualização pelo Wireshark do pacote transmitido.

O projeto recebe os pacotes da rede externa, os envia para o processador embarcado, e após encaminha para a lógica programável. O objetivo original desta aplicação era fazer o desvio diretamente para a parte lógica, porém este procedimento não logrou êxito na sua execução.

É preciso um estudo maior no desvio de dados para o PL sem a necessidade de passar pelo processador. Este projeto inicial permitiu entretanto compreender as ferramentas Vivado e o desenvolvimento do software no SDK. Esta aplicação pode ter continuidade em trabalhos futuros.

6.2 Aplicação para bloquear palavras

Esta segunda aplicação tem por objetivo utilizar a PL como aceleradora do PS, a partir de dados oriundos da interface de rede. O recebimento das palavras, para o processador encaminhar para o PL é feito através de uma comunicação utilizando a API *raw* e o protocolo TCP da biblioteca LwIP. A comunicação externa é feita por telnet, onde os parâmetros passados são o IP da placa e a porta. Um código semelhante ao utilizado no software foi descrito no Capítulo 5.

Dois tipos de palavras são recebidos pela aplicação. O primeiro tipo de palavra recebido pelo PS corresponde ao que deve ser bloqueado. O segundo tipo de palavras é o que será processado pelo PL, ou seja comparado com as palavras inicialmente recebidas, e se forem iguais serão descartados e o processador será informado via interrupção.

A Figura 20 apresenta a arquitetura da aplicação, na ferramenta Vivado, através de um diagrama de blocos:

- O módulo 1 é a área de processamento (PS), a qual recebe as palavras oriundas da interface de rede e as direciona para a área da lógica programável (PL).
- O módulo 2 faz a conexão da porta *master* do módulo 1 com a porta *slave* do IP, módulo 3.
- O módulo 3 é o IP que realiza a seleção das palavras a serem descartadas. Quando isto ocorrer o processador recebe um sinal através do canal de interrupção (em destaque na Figura 20). O processamento feito pelo IP é explicado mais adiante na Figura 22 .
- O módulo 4 é a conexão do *master* do IP, módulo 3, com o *slave* do PS, módulo 1.

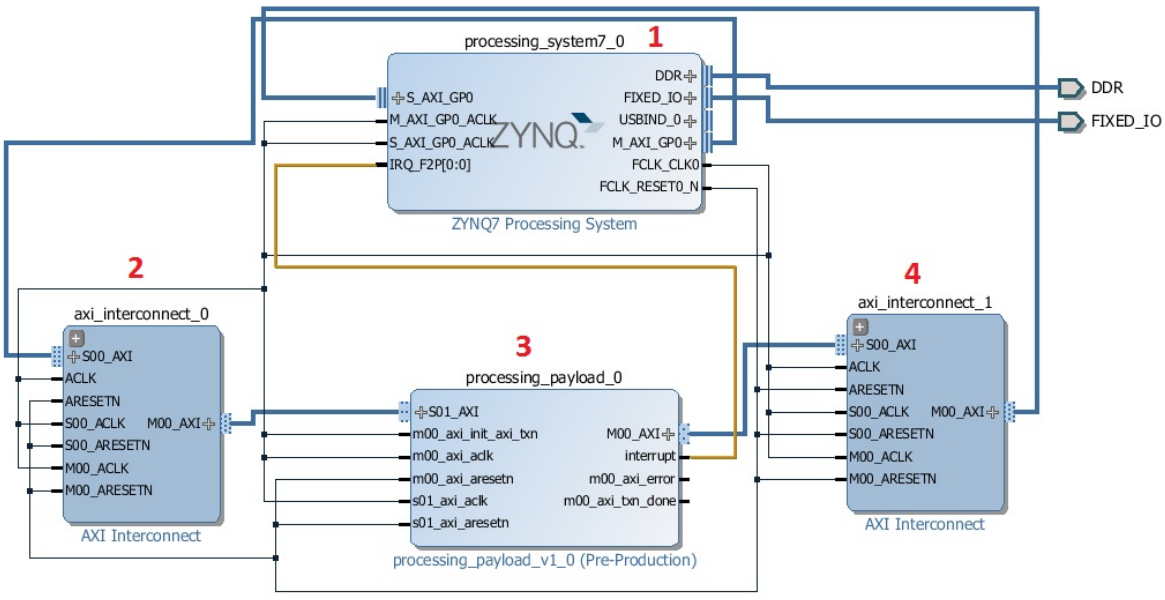


Figura 20 – Arquitetura de blocos do projeto de bloquear palavras no Vivado.

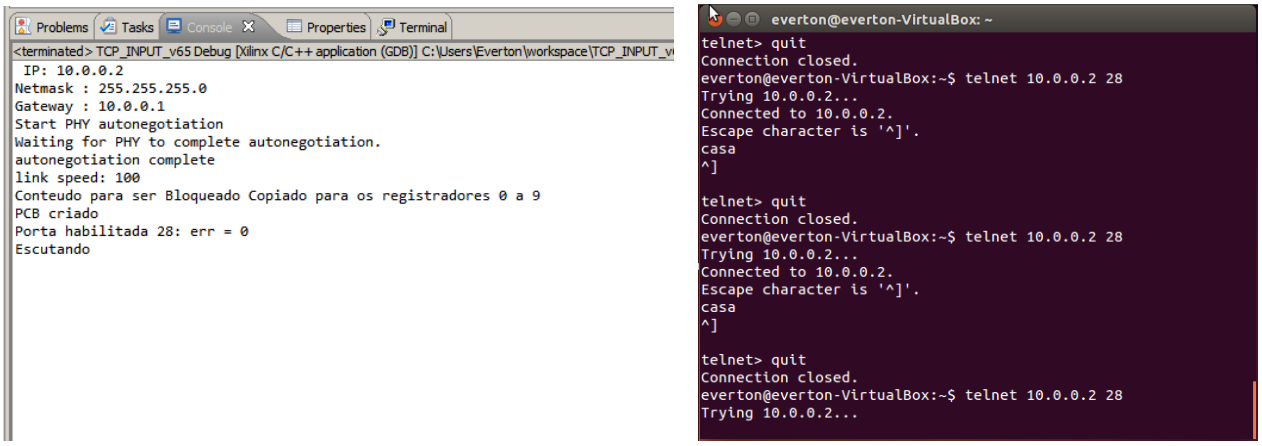
É utilizado como base para o software uma aplicação disponível no SDK que utiliza a biblioteca LwIP para realizar uma conexão com um host remoto através de telnet. Telnet é um protocolo que opera utilizando sobre o TCP.

O software é iniciado invocando as funções referentes à configuração das rotinas de tratamento de interrupção (*interrupt handler*). A função *interrupt handler* contém o código desenvolvido para o PL se comunicar com o PS. Esta função é executada quando a interrupção

for ativada pelo IP, informando o bloqueio da palavra recebida. Após a configuração das rotinas de tratamento de interrupção é realizada a configuração da interface de rede.

O próximo passo a ser executado é a chamada da função responsável por copiar as palavras que devem ser bloqueadas para os registradores do IP. Após, a aplicação fica aguardando palavras enviadas via telnet para serem analisadas.

A Figura 21(a) ilustra a inicialização da aplicação. Após as configurações, a aplicação aguarda o envio de palavras. A Figura 21(b) apresenta a comunicação realizada em um terminal no *host* para o envio das palavras.



(a) (b)

Figura 21 – Inicialização da aplicação e conexão externa com a mesma.

A palavra é recebida no PL, e é copiada para um registrador conforme sua classificação. Os registradores do periférico foram divididos em dois tipos, tipo 0, destinados a palavras que devem ser bloqueadas e tipo 1, correspondendo a palavras recebidas para serem comparadas aos padrões previamente recebidos.

A máquina de estados responsável pelo processamento de pacotes do tipo 1 é ilustrada na Figura 22.

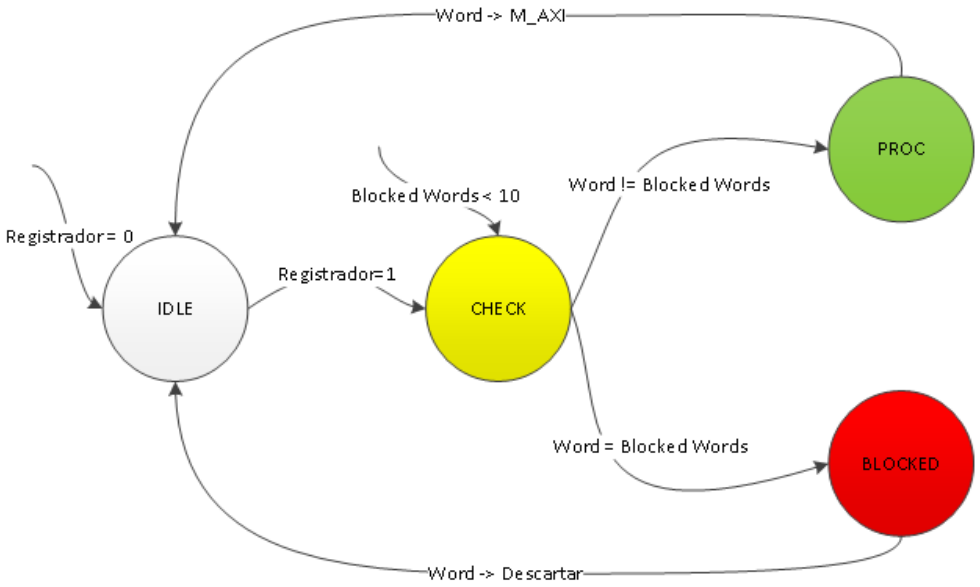


Figura 22 – Máquina de estados referente ao processo de comparação das palavras que devem sem bloqueadas.

No primeiro estado, IDLE, são copiadas para os registradores do periférico as palavras que devem ser bloqueadas (*padrões*), ficando neste estado até a recepção de palavras para registradores do tipo 1. O próximo estado, CHECK, faz uma comparação da palavra que acabou

de ser recebida com os padrões que devem ser bloqueados. Não havendo comparação positiva a máquina de estados segue para o estado PROC, podendo reencaminhar palavra para o processador. No caso de uma comparação positiva a máquina de estados segue para o estado BLOCKED. Neste caso a palavra é igual a algum padrão, sendo descartada. Após estes dois últimos estados, a máquina de estados retorna para o estado IDLE.

A Figura 23 ilustra o caminho percorrido pela palavra no hardware. Após entrar pela interface de rede (1 na Figura 23), ela é enviada através do barramento AXI pela porta *master* do PS (2 na Figura 23), sendo encaminhada para o periférico onde será armazenada nos registradores (3 e 4 na Figura 23). Para voltar para o processador a palavra pode percorrer três caminhos: (*i*) a porta *slave* do PS (5 na Figura 23); (*ii*) através da porta de alto desempenho (7 na Figura 23); (*iii*) através da porta ACP (8 na Figura 23). Quando o pacote deve ser descartado, o processador é informado via interrupção (6 na Figura 23). O retorno da palavra para o periférico não foi implementado até o presente momento. A função do periférico é de informar se a palavra deve ser ou não bloqueada.

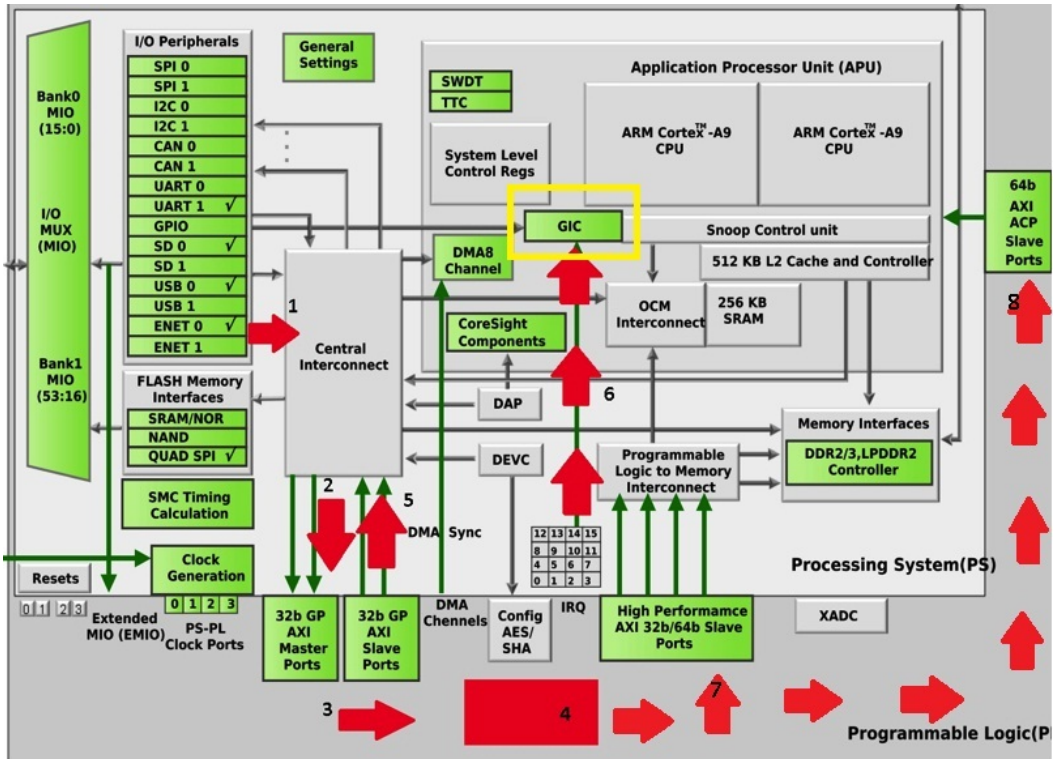


Figura 23 – Caminho percorrido pela palavra no hardware.

O GIC (*Generic Interrupt Controller*), em destaque na Figura 23, pode ser configurado de duas formas, interrupção a partir do PL e/ou interrupção a partir do PS. O controlador garante que toda interrupção será atendida, sempre de acordo com a com a maior prioridade. Toda a fonte de interrupção contém um número de identificação único [XIL13b] A faixa de identificação utilizada para se referir a IPs no FPGA é de 84 a 91.

No nível de software é necessário criar duas rotinas para o tratamento de interrupção. Uma que faça a configuração ou ativação do *driver*. Esta configuração é genérica e serve para diferentes interrupções. A outra rotina refere-se a ação que é tomada quando ocorre a interrupção (*interrupt handler*).

Quando um projeto é exportado do Vivado para o SDK são gerados *headers*. Destes, três são essenciais para implementação do software com suporte a interrupção:

- *Xparameters.h*: este arquivo contém os endereços para o processador acessar o periférico.
- *Xscugic.h*: este arquivo contém as funções utilizadas na configuração da interrupção no software.
- Xil_exception.h: este arquivo contém as exceções do processador.

Segue um código semelhante ao utilizado neste projeto, apenas não realizando os tratamentos em caso de erros.

```
#include "xparameters.h"
#include "xscugic.h"

#define XPAR_PS7_SCUGIC_0_DEVICE_ID 0

//driver de interrupção
XScuGic InterruptController;
static XScuGic_Config *GicConfig;

//inicializando o drive de interrupção
GicConfig = XScuGic_LookupConfig(XPAR_PS7_SCUGIC_0_DEVICE_ID);
XScuGic_CfgInitialize(&InterruptController, GicConfig,GicConfig->CpuBaseAddress);

//conectando hardware no driver de interrupção
XScuGic_Connect(&InterruptController,91,
                    (Xil_ExceptionHandler)INTERRUPT_Handler1,
                    (void *)&InterruptController);
      XScuGic_Enable(&InterruptController, 91);

//habilita interrupção no ARM
XScuGic_Enable(&InterruptController, 91);
```

Para verificar o comportamento no hardware, é utilizada a ferramenta *Vivado Logic Analyzer*. A Figura 24 ilustra o início de operação da máquina de estado apresentada na Figura 22. No estado IDLE armazena-se os padrões a serem verificados nos registradores BLOCKED_reg (1 na Figura 24). O sinal TYPE_REGISTER informa o tipo de palavra, sendo neste caso particular do tipo 0 (2 na Figura 24).

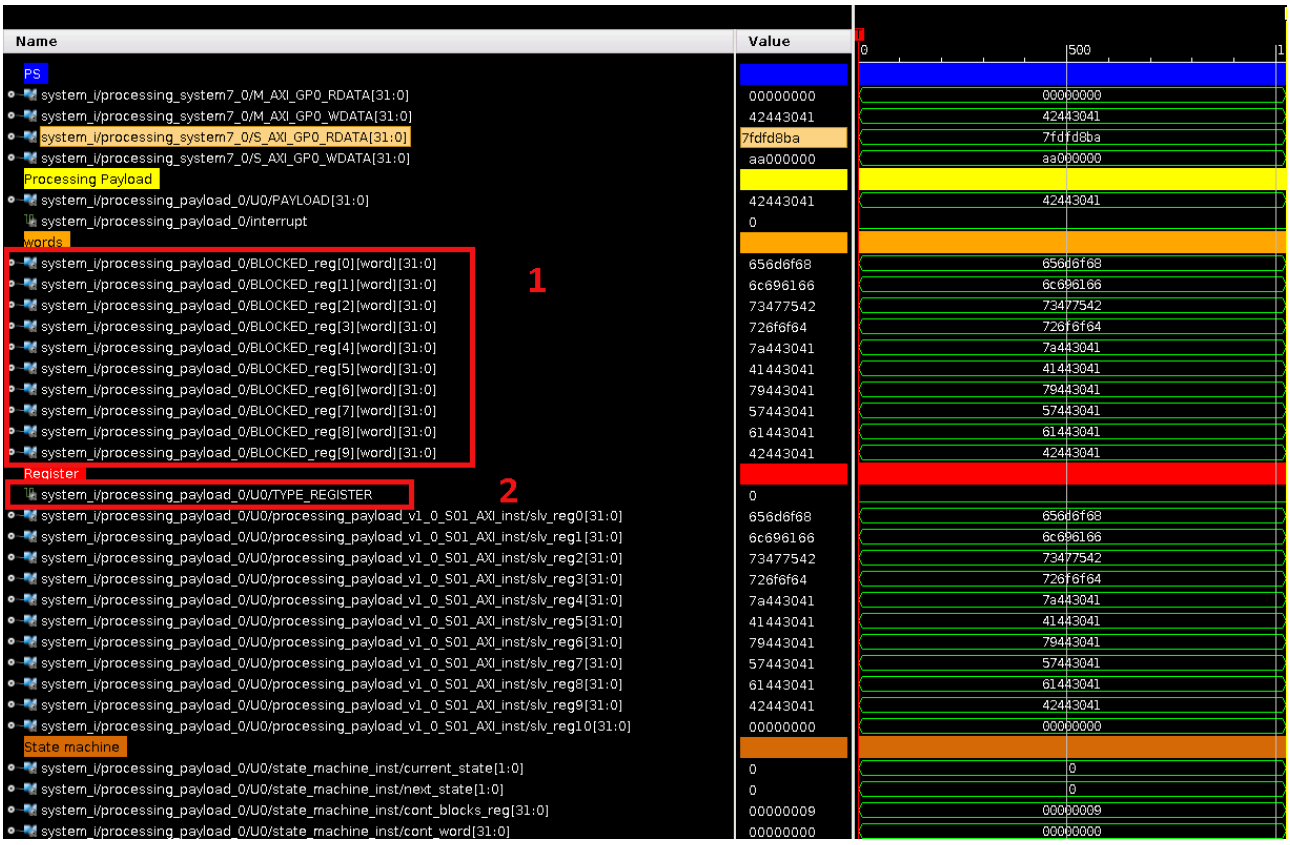


Figura 24 – Estado IDLE no momento que as palavras que devem ser bloqueadas estão sendo armazenadas.

Esta primeira etapa corresponde à inicialização do sistema. Para sair do modo IDLE é necessário que seja identificado que algo foi escrito nos registradores do tipo 1, escrevendo-se uma combinação de caracteres que não consta na lista de palavras a serem bloqueadas, Figura 25.

```
Inicializacao da interrupcao OK 1
Inicializacao da interrupcao OK 2
OK
 IP: 192.168.1.10
Netmask : 255.255.255.0
Gateway : 192.168.1.1
Start PHY autonegotiation
Waiting for PHY to complete autonegotiation.
autonegotiation complete
link speed: 1000
Conteudo para ser Bloqueado Copiado para os registradores 0 a 9
PCB criado
Porta habilitada 28: err = 0
Escutando
Resgistrador 43C00040 com o valor = 74747474
```

```
Evertons-Mac-mini:~ evertoncunha$ telnet 192.168.1.10 28
Trying 192.168.1.10...
Connected to 192.168.1.10.
Escape character is '^]'.
tttt
```

Figura 25 – Momento que a aplicação recebe uma palavra enviada pelo terminal.

O envio dos caracteres é confirmado no Logic Analyzer, Figura 26. No sinal PAYLOAD (1 na Figura 26) é visualizado os caracteres na forma hexadecimal e logo abaixo o sinal de interrupção, 1 na Figura 26. O sinal REGISTER_TYPE indica uma palavra do tipo 1 (2 na Figura 26). Nos sinais da máquina de estado é verificado que esta foi para o estado CHECK (estado 1), e visto que a apalvra não deve ser bloqueada, a máquina de estados passou para o estado PROC (estado 3), 3 na Figura 26.

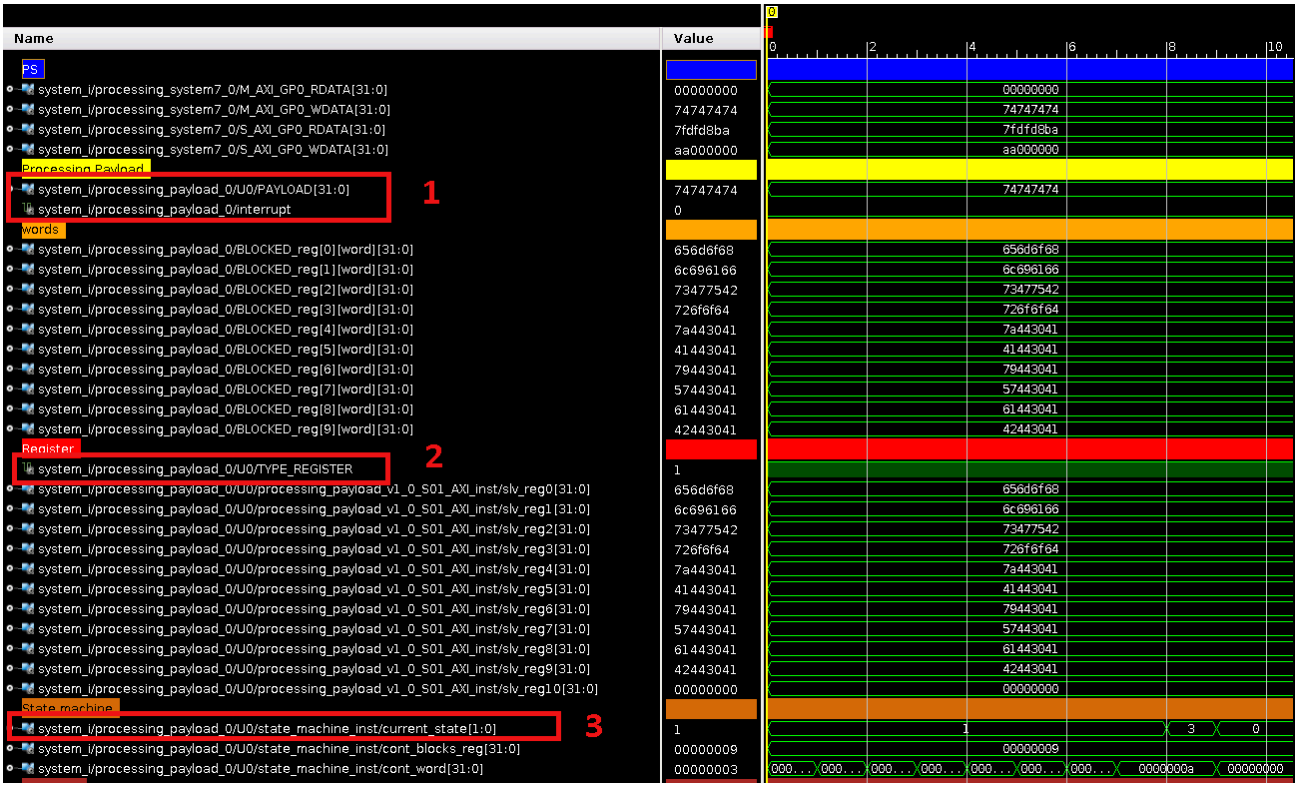
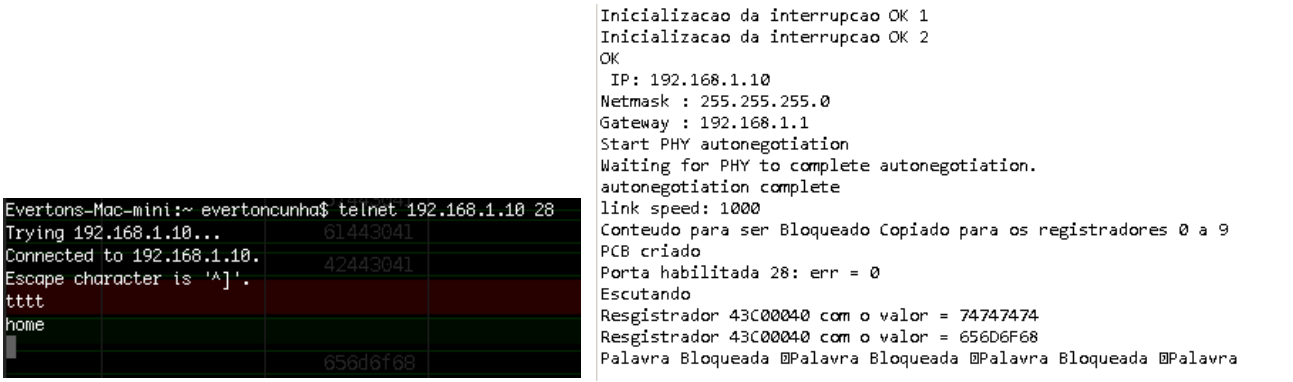


Figura 26 – Escrita no registrador do tipo 1.

Agora exemplifica-se um caso onde uma palavra deve ser bloqueada. A palavra a ser avaliado é "*home*" (em hexadecimal: 0x68 0x6F 0x6D 0x65). A Figura 27(a) apresenta o envio da palavra "home". A Figura 27(b) acusa a recepção da palavra, imprimindo a palavra (0x656D6F68) e o aviso ao processador que este pacote deve ser bloqueado.



(a) (b)

Figura 27 – Envio de palavra que consta na lista de bloqueio.

Na ferramenta Logic Analyzer, Figura 28, o sinal PAYLOAD (4 na Figura 28) contém a palavra enviada e a interrupção habilitada no sinal *interrupt* (1 na Figura 28). No conjunto de palavras que devem ser descartadas pode ser observada uma palavra com conteúdo idêntico ao PAYLOAD (2 na Figura 28). Nos sinais referentes a máquina de estado, após ser verificada a igualdade no estado de CHECK (estado 1), passa-se para o o estado de BLOCK, (estado 2), 3 na Figura 28.

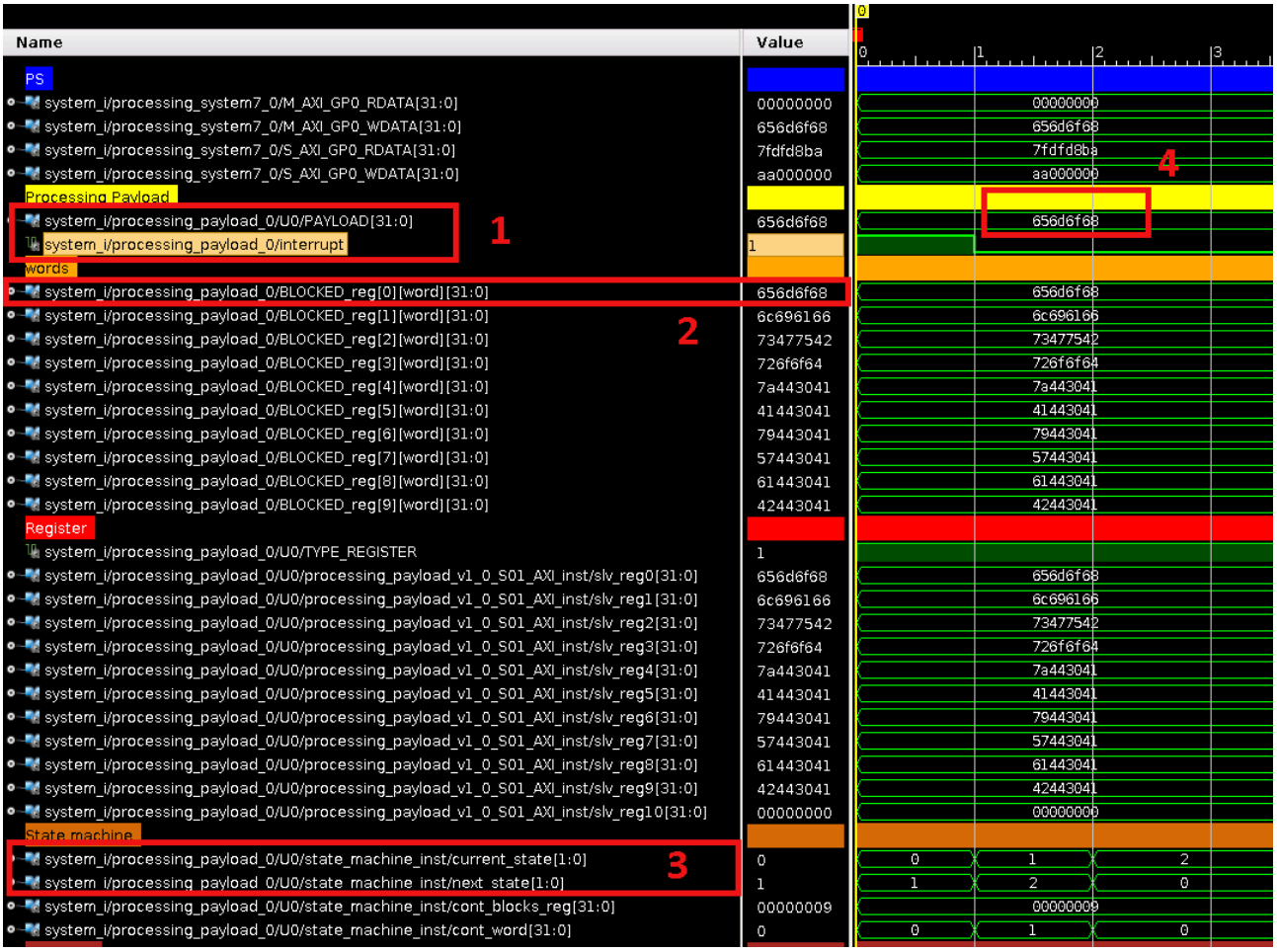


Figura 28 – Palavra bloqueada, com habilitação do sinal de interrupção.

7. CONCLUSÃO E TRABALHOS FUTUROS

Este TCC teve por característica a multidisciplinaridade, requerendo do Autor a aplicação dos conteúdos de arquitetura de computadores, organização de computadores, algoritmos de programação, e redes de computadores. A execução deste TCC permitiu colocar em prática estes conteúdos estudados ao longo do curso, além de aprofundá-los em diversos tópicos, como a utilização de SoCs para o desenvolvimento de sistemas embarcados.

Os objetivos estabelecidos no início deste trabalho foram plenamente atingidos: (*i*) domínio da tecnologia FPGA (Xilinx) com processador embarcado (ARM); (*ii*) domínio de ferramentas para o projeto em FPGAs do tipo SoC (Vivado); (*iii*) desenvolvimento de uma aplicação que utilize o processador ARM e a lógica reconfigurável.

Como trabalhos futuros enumera-se:

- Desenvolver outras formas de enviar dados para a área de processamento utilizando a biblioteca LwIP, realizando a recepção e o tratamento de outros protocolos.
- Desenvolver outras aplicações que utilizam a área de lógica programável para auxiliar a área de processamento. Dentro destas aplicações, concluir o projeto de redirecionamento de pacotes para a área de logica programável sem a necessidade de passar pela área de processamento.
- Explorar a possiblidade de adicionar IPs específicos para novas aplicações com interfaces de entrada e saída utilizando outros modos de barramento AXI, e realizando conexões com às portas ACP e HP da área de processamento.
- Neste trabalho utilizou-se um *core* do processador ARM. Explorar a possibilidade de dividir as tarefas de processamento em dois *cores*, podendo assim executar aplicações distintas em cada *core*.
- Neste trabalho foram desenvolvidas aplicações sem sistema operacional. Já encontra-se em desenvolvimento um trabalho que visa fazer um relatório técnico abordando as etapas de embarcar o Linux com um projeto desenvolvido no Vivado, incluindo o tratamento de dados externos, comunicação com o FPGA e retornado estes dados para o processador.

REFERÊNCIAS

[ALT14a] ALTERA Corporate. “FPGAs Altera”. Capturado em: http://www.altera.com/products/fpga.html, Agosto 2014.

[ALT14b] ALTERA Corporate. “High-Performance Stratix Architecture”. Capturado em: http://www.altera.com/devices/fpga/stratix-fpgas/stratix/stratix/features/stx-architecture.html, Agosto 2014.

[ARM13] ARM Ltd. “AMBA Open Specifications”. Capturado em: http://www.arm.com/products/system-ip/amba/amba-open-specifications.php, Agosto 2014.

[ARM14] ARM Ltd. “Cortex-A9 Processor”. Capturado em: http://www.arm.com/products/processors/cortex-a/cortex-a9.php, Agosto 2014.

[BOB07] C. Bobda. “Introduction to Reconfigurable Computing: Architectures, algorithms and applications”. Dordrecht: Springer, 2007, 375p.

[DEH08] S. Hauck e A. DeHon. “Reconfigurable Computing: The Theory and Practice of FPGA Based Computation”. Burlington: Elseiver, 2008, 945p.

[LWI14a] LwIP “LwIP Wiki”, Capturam em: http://lwip.wikia.com/wiki/LwIP_Wiki, Novembro 2014

[LWI14b] LwIP “RAW/TCP”, Capturado em:http://lwip.wikia.com/wiki/Raw/TCP, Jundo de 2011

[RAJ00] R. Rajsuman. “System-on-a-Chip: Design and Test”. Santa Clara: Artech House, 2000, 277p.

[SAV14] Savannah “LwIP – A Lightweight TCP/IP stack”, Capturado em: http://savannah.nongnu.org/projects/lwip/, Novembro 2014.

[SCA97] R.R. Schaller. “Moore’s law: past, present and future”. Spectrum IEEE, vol. 34-6, Jun 1997, pp 52-59.

[STA11] W. Stallings. “Arquitetura e Organização de Computadores”. São Paulo: Pearson, 2010, 624p.

[STM14] STMicroelectronics “LwIP TCP/IP stack demonstration for STM32F4x7 microcontrollers”, Capturado em: http://www.st.com/st-web-ui/static/active/cn/resource/technical/document/application_note/DM00036052.pdf, Novembro 2014.

[TAN03] A. Tanenbaum. “Redes de computadores”. São Paulo: Elsevier, 2003, 945p.

[TEX14] Texas Instruments Inc. “OMAP™ Mobile Processors: OMAP™ 4 Platform”. Capturado em: http://www.ti.com/general/docs/wtbu/wtbuproductcontent.tsp?contentId=53247&navigationId=12842&templateId=6123, Agosto 2014.

[UBM13] UBM Tech. “2013 Embedded Markets Study”, Capturado em: https://ubmelectronics.app.box.com/s/rma2l5gn0xixxoh2qkvr, Agosto 2014.

[XIL12] XILINX Inc. “AXI Reference Guide”. Capturado em: http://www.xilinx.com/support/documentation/ip_documentation/axi_ref_guide/latest/ug761_axi_reference_guide.pdf, Agosto 2014.

[XIL13a] XILINX Inc. “Zynq-7000 All Programmable SoC Overview”, Capturado em: http://www.xilinx.com/support/documentation/data_sheets/ds190-Zynq-7000-Overview.pdf, Agosto 2014.

[XIL13b] XILINX Inc. “Vivado Design Suite”, Capturado em: http://www.xilinx.com/support/documentation/sw_manuals/xilinx2013_3/ug898-vivado-embedded-design.pdf, Agosto 2013.

[XIL13c] Wiki XILINX “Zynq-7000 AP SoC - Performance - Ethernet Packet Inspection - Bare Metal - Redirecting Headers to PL and Cache Tech Tip”, Capturado em: http://www.wiki.xilinx.com/Zynq-7000+AP+SoC+-+Performance+-

+Ethernet+Packet+Inspection+-+Bare+Metal+-
+Redirecting+Headers+to+PL+and+Cache+Tech+Tip, Junho de 2013.

[XIL14a] XILINX Inc. “FPGAs Xilinx”, Capturado em: http://www.xilinx.com/products/silicon-devices/fpga/index.htm, Agosto 2014.

[XIL14b] XILINX Inc. “UltraScale Architecture and Product Overview”, Capturado em: http://www.xilinx.com/support/documentation/data_sheets/ds890-ultrascale-overview.pdf, Agosto 2014.

[XIL14c] XILINX Inc. “Zynq-7000 Silicon Devices”, Capturado em: http://www.xilinx.com/products/silicon-devices/soc/zynq-7000/silicon-devices/index.htm, Agosto 2014.

[ZED14] ZEDBOARD “ZedBoard Documentation”, Capturado em: http://www.zedboard.org/support/documentation/1521, Agosto 2014.

ANEXO: TUTORIAL VIVADO

Neste tutorial são abordadas as etapas para desenvolver um projeto simples de comunicação entre o *Processing System* (PS) e *Programmable Logic* (PL), onde neste último haverá um periférico personalizado. O principal objetivo é demonstrar a utilização da ferramenta. A versão utilizada neste tutorial é a 2014.2.

Figura 1 - Tela inicial do Vivado.

Para iniciar um novo projeto deve-se clicar em ***Create New Project***, Figura 1, e em seguida clicar em ***Next***, Figura 2. Na janela seguinte é apresentada uma janela para definir o nome e local do projeto, Figura 3.

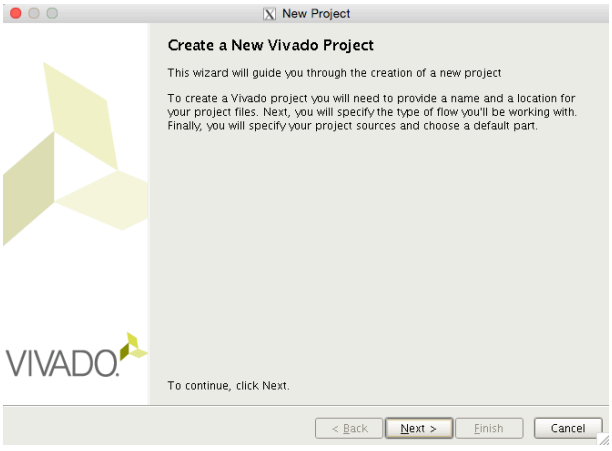


Figura 2 - Início do guia para criar um projeto.

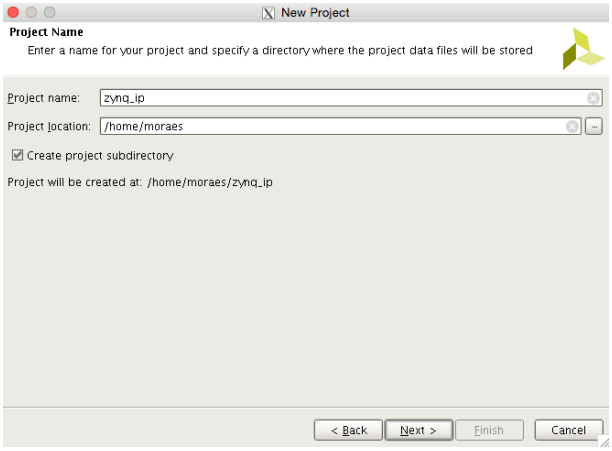


Figura 3 - Definição do nome e local do projeto.

A Figura 4, apresenta a etapa para especificar o tipo do projeto, neste exemplo como não existem fontes implementadas, deve-se marcar a opção ***Do not specify sources at this time***, com esta opção desmarcada na próxima etapa é possível importar VHDLs/Verilogs, *constrains* e IPs previamente implementados. Depois apenas clicar em ***Next***.

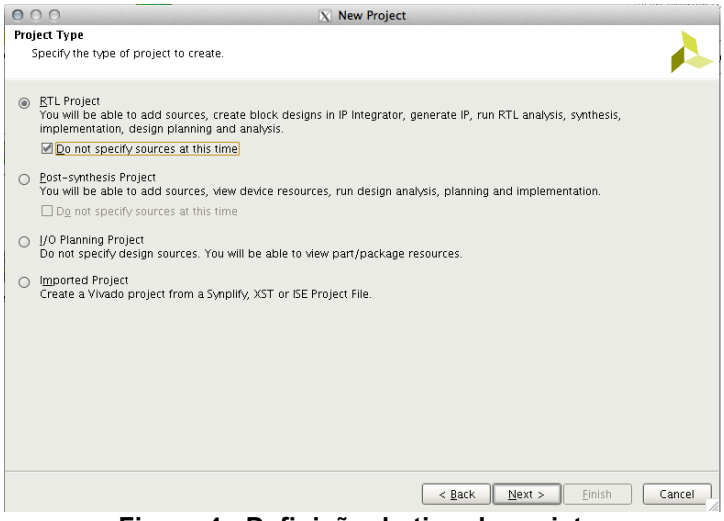


Figura 4 - Definição do tipo de projeto.

Na próxima tela, Figura 5, são definidas as configurações da placa. As especificações devem ser obtidas do manual do fabricante. Observar que os dados do dispositivo são apresentados, como número de LUTs e flip-flops. Clique em ***Next*** após concluir.

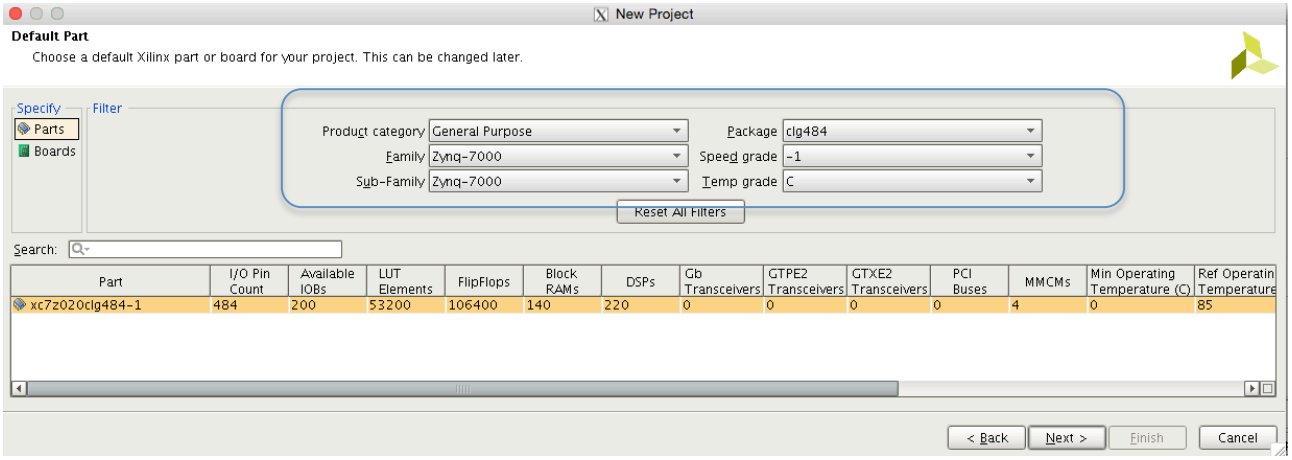


Figura 5 – Definição dos parâmetros da placa Zynq.

Após Clicar em ***Finish***, Figura 6,para concluir a guia de configuração de projeto.

Figura 6 - Finalizada a etapa inicial de configuração de um novo projeto.

Após concluída a etapa de configurações básicas é apresentado o ambiente de desenvolvimento do projeto, Figura 7. Considerações importantes sobre o ambiente de projeto:

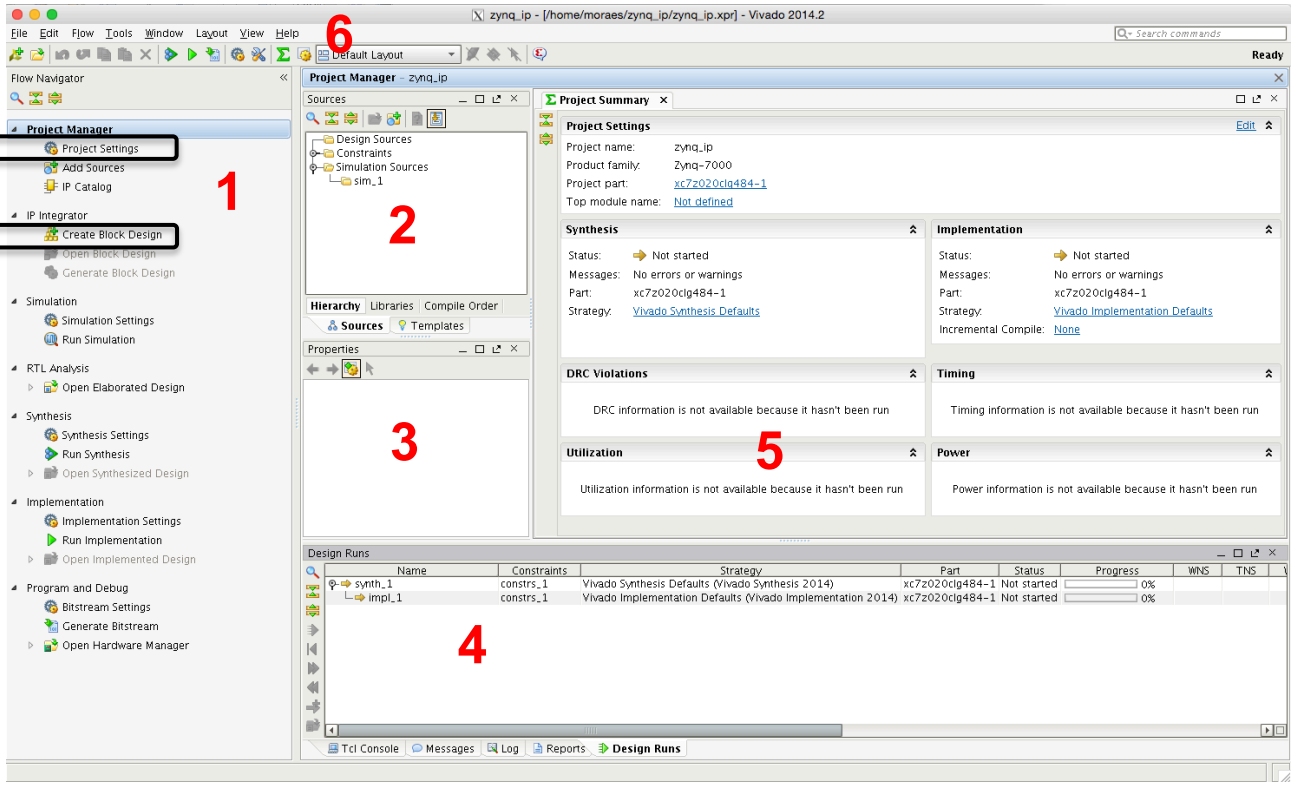


Figura 7 - Tela inicial do ambiente de projeto.

1. Nesta barra existem as opções de configurar o projeto, adicionar fontes e acessar o catálogo de IPs. É possível executar toda etapa do fluxo de projeto, como simulação, síntese, implementação e programação do hardware.
2. Aqui fica a árvore do projeto, algumas só podem ser acessadas se outras etapas anteriores forem concluídas com sucesso.
3. Propriedades de componentes utilizados durante o projeto.
4. Nestas abas pode-se configurar qualquer tipo de saída para ser visualizada, como Debug, mas é sugerível que seja utilizada para visualizar os logs e mensagens de eventos e erros durante o projeto.
5. Aqui em cada etapa do projeto será apresentada uma informação diferente, neste primeiro momento não se tem informações referentes as etapas.
6. Barra com opções que podem ser utilizadas durante o projeto.

Para fazer as configurações no projeto, deve-se clicar em ***Project Settings***, em destaque na Figura 7. Nesta janela, Figura 8, existe a possibilidade de escolher a linguagem, que por default está em Verilog, o nome do *top* e alterar opções para as outras etapas do projeto como a etapa de síntese, por exemplo. Clicar em ***Apply*** e ***OK***, após realizar a alteração para VHDL

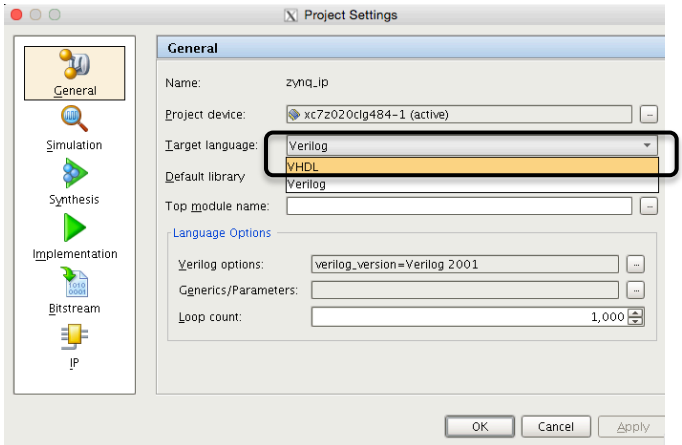


Figura 8 – Configurações do projeto existente.

Após configurado o projeto, é necessário criar um “bloco” ,onde ficarão os blocos IPs. Este bloco funciona como um “integrador”. Clicando no botão que ***Create Block Design*** (em destaque na Figura 7), definir um nome para o bloco, Figura 9, e clicar em ***OK***.

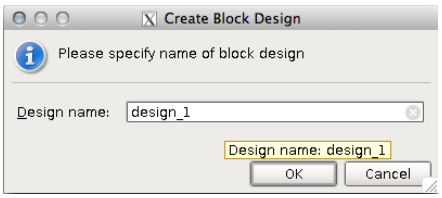


Figura 9 - Definição do nome do bloco.

Com o bloco pronto é necessário adicionar o(s) IP(s) no projeto. Primeiro adiciona-se o bloco referente ao processamento. É preciso adicionar o PS referente a arquitetura configurada. Clicando em ***Add IP***, é visualizado o catálogo de IPs da ferramenta, Figura 10. Procurar *ZYNQ7 Processing System* e clicar duas vezes ou *enter* (cuidar: não selecionar *ZYNQ7 Processing Syste BFM*).

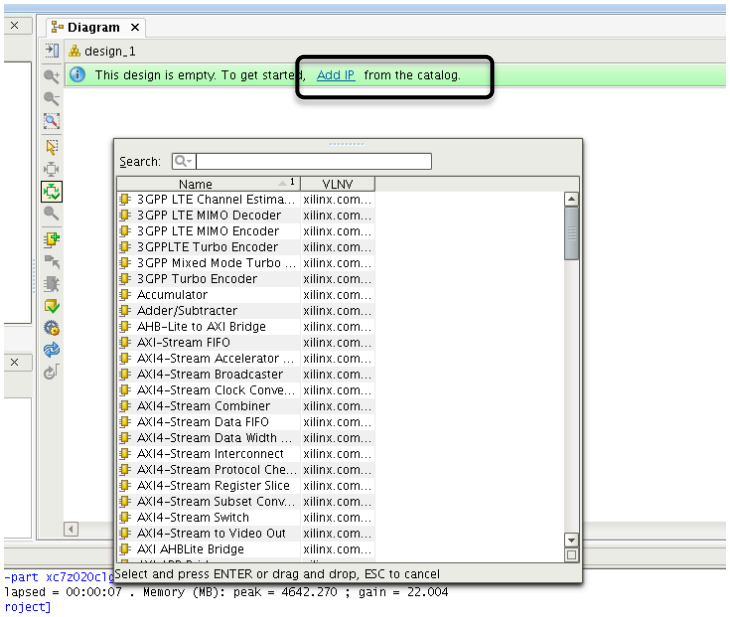


Figura 10 - Inserção do PS no projeto.

O projeto apresenta o IP que é responsável pelo processamento e à esquerda pode-se observar os pinos que por padrão estão configurados. Agora é preciso configurar o bloco para trabalhar como desejado. Clicando duas vezes sobre o bloco ou com o botão direito e na opção ***Customize Block***, Figura 11.

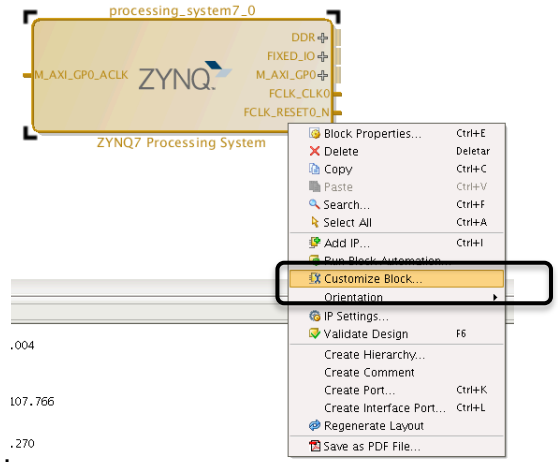


Figura 11 - Edição do *Processing System*.

Com o ambiente de edição do PS aberto, observar os blocos verdes, Figura 12. Estes são os blocos que podem ser configurados no *processing system*.

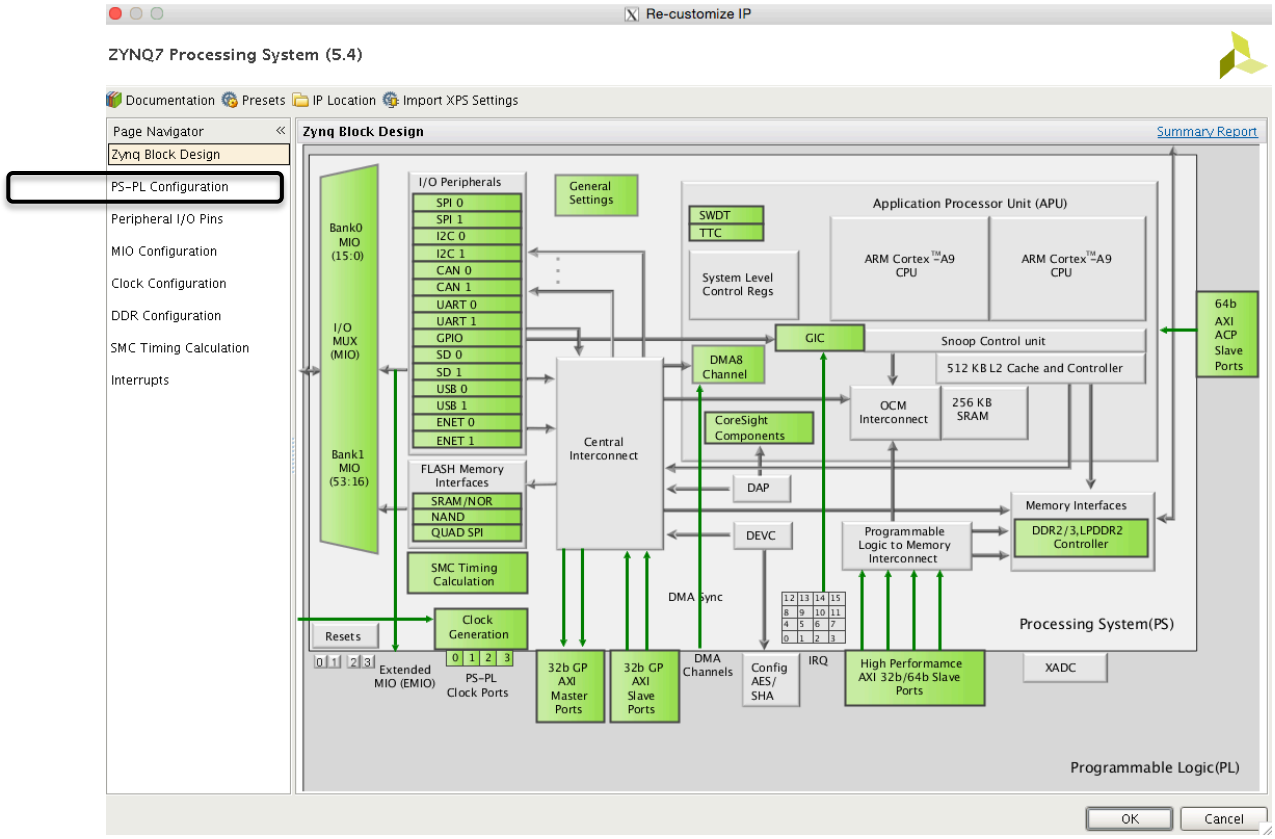


Figura 12 – Ambiente configuração do IP Processing System.

No menu que se encontra à esquerda do editor, há opções para edição divididas em categorias, ou pode-se clicar diretamente nos blocos ativos para configuração. Em ***PS-PL Configuration***, Figura 13, vê-se onde são definidas as opções para comunicação do PL com o PS. Selecionar os campos destacados, que são referentes a habilitação de um reset e de uma interface máster AXI que se comunicará com o IP.

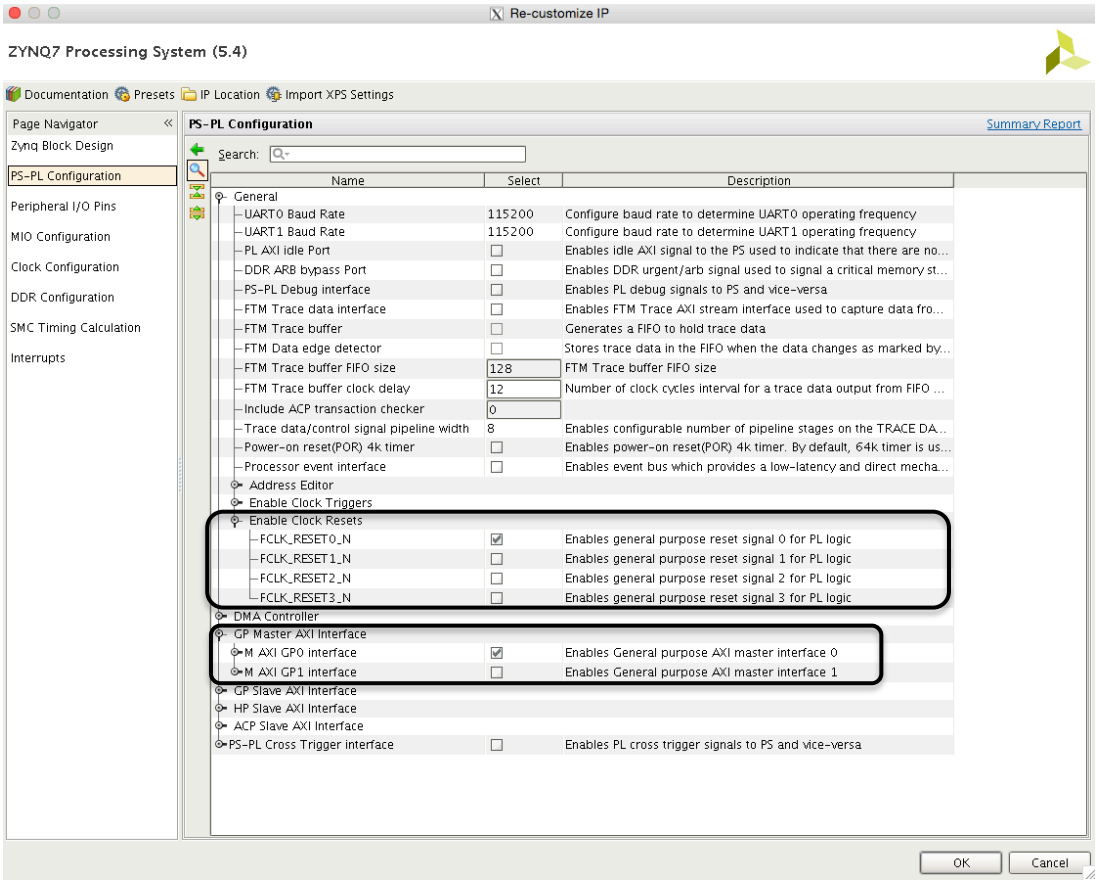


Figura 13 - PS-PL Configuration.

No ***MIO Configuration***, Figura 14, e ***Peripheral I/O Pins***, Figura 15, são configurados os pinos dos periféricos conforme o projeto. Os periféricos configurados estão ilustrados na Figura 16. Na Figura 14 deve-se executar as ações em destaque. Na Figura 15 não há alterações a realizar. Notar que o módulo inserido está em destaque, em verde. Na Figura 16 devem-se selecionar os periféricos.

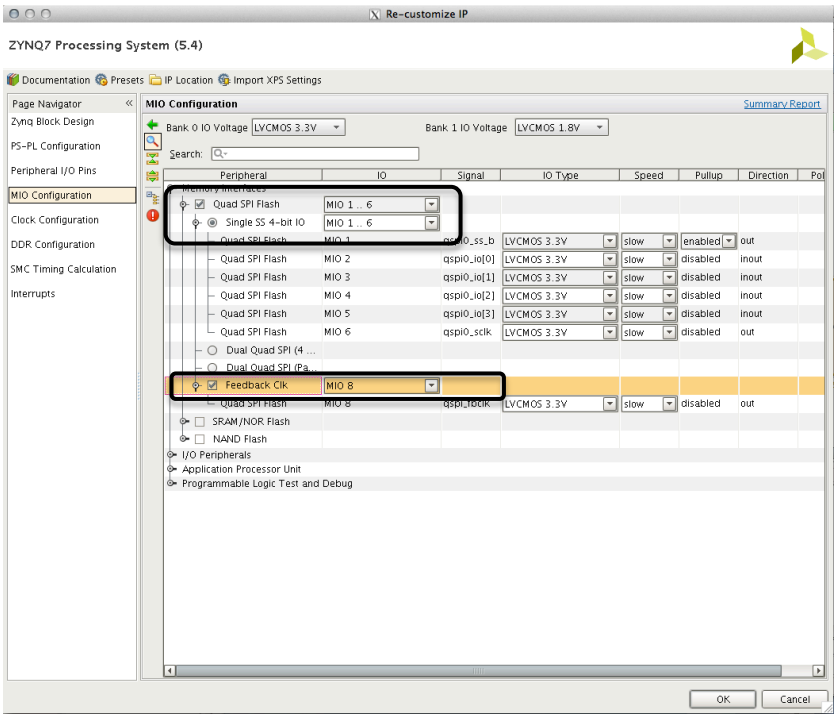


Figura 14 - Configuração da interface de memória.

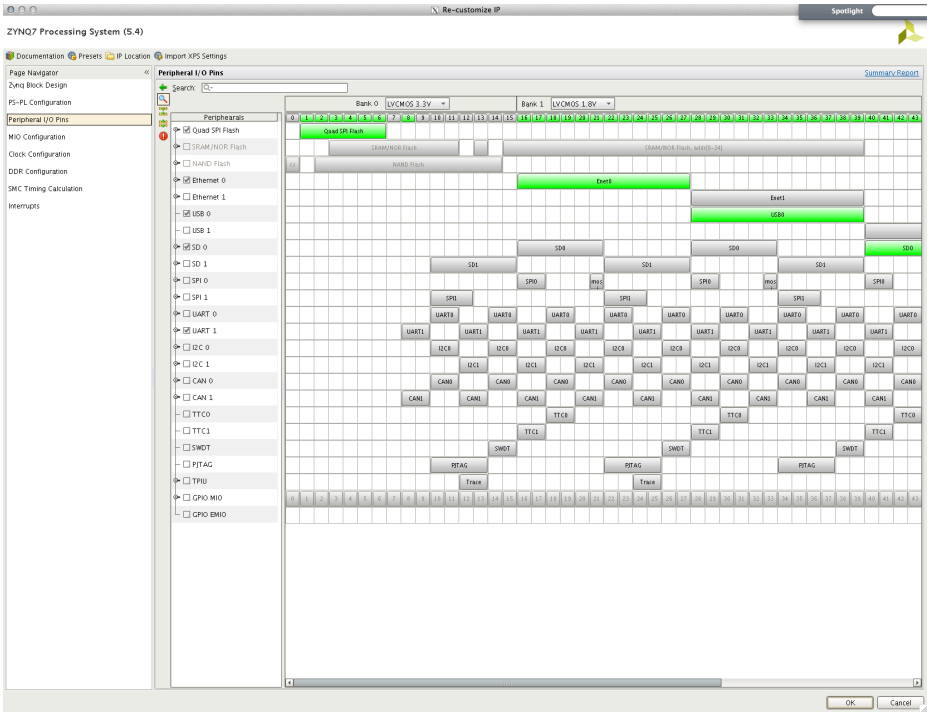


Figura 15 - Visualização da utilização dos periféricos pela interface Peripheral I/O Pins.

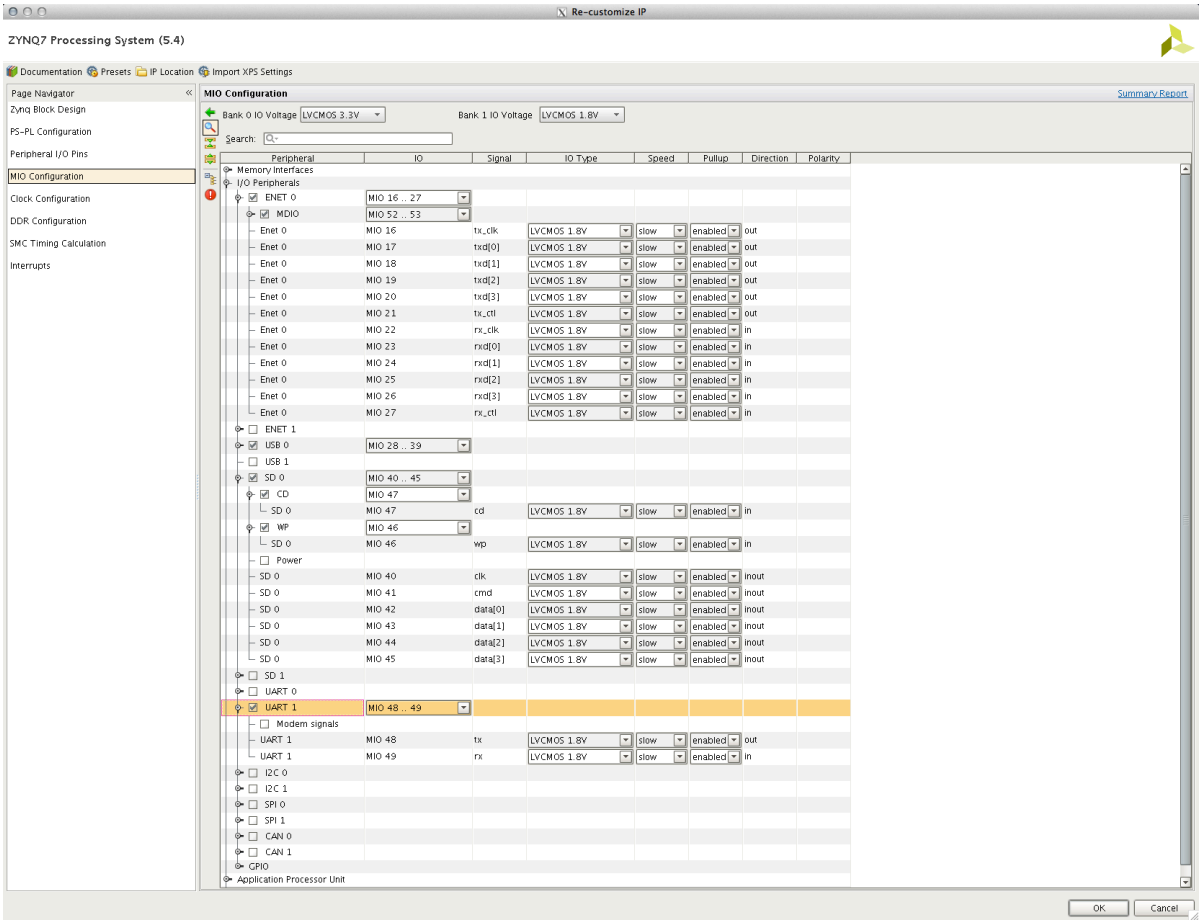


Figura 16 - Configuração dos periféricos.

O ***Clock configuration***, é onde é definido o *clock*. As configurações neste projeto estão na Figura 17.

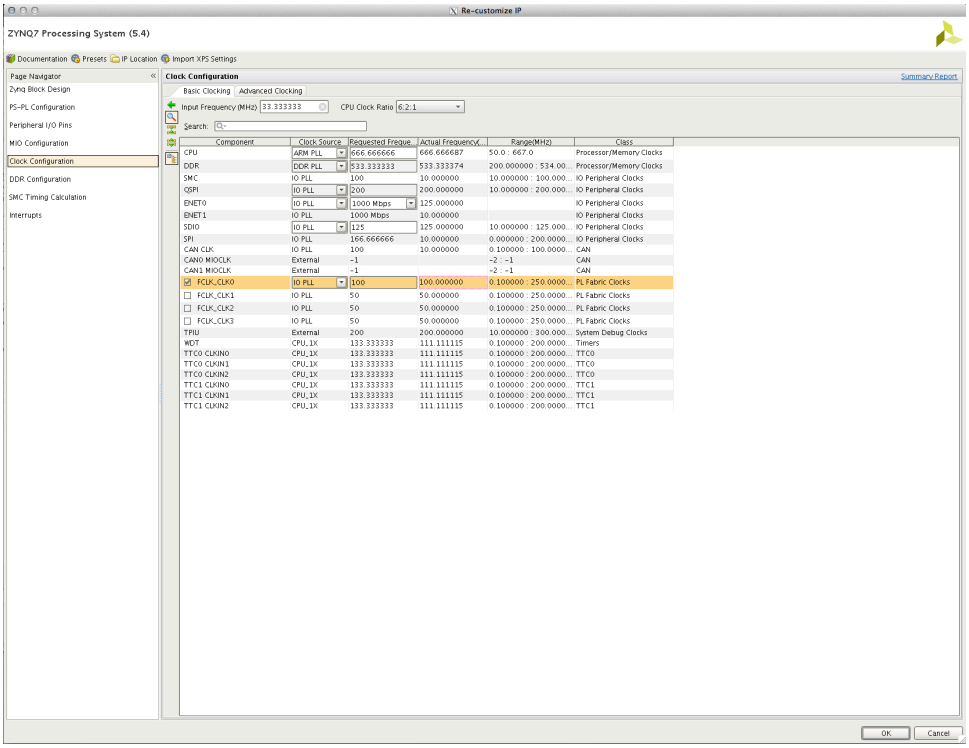


Figura 17 - Configurações no Clock configuration.

DDR Configuration, configuração referente à memória, no projeto é utilizado DDR3. Na Figura 18, esta detalhada as configurações. Nesta tela deve-se configurar como na figura abaixo.

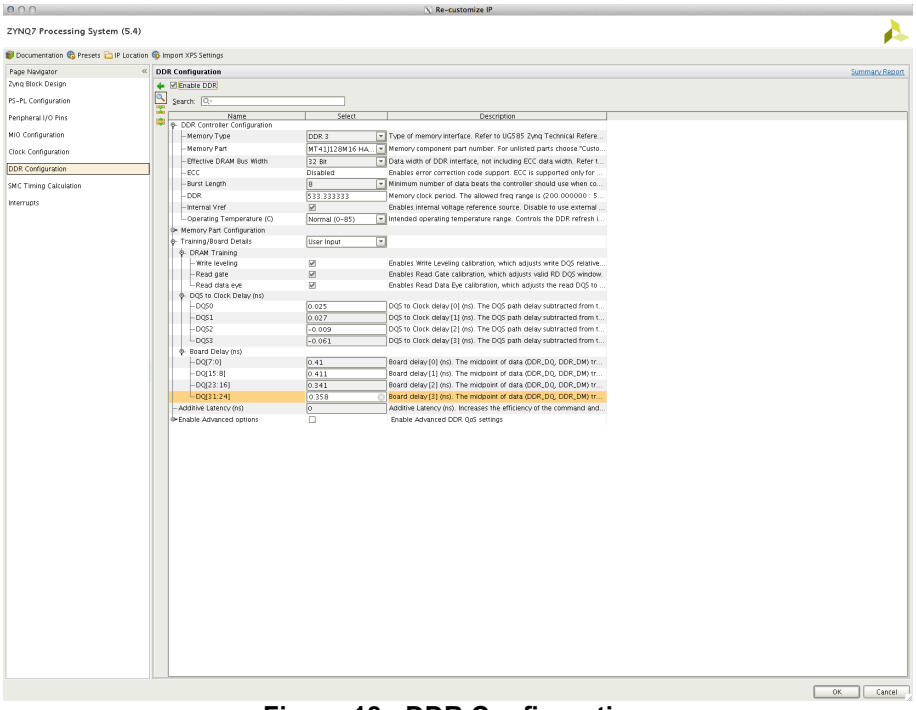


Figura 18 - DDR Configuration.

Em ***Interrupts***, Figura 19, são definidas as interrupções tanto do PS para o PL, como do PL para o PS. Neste projeto não vai ser utilizado a interrupção.

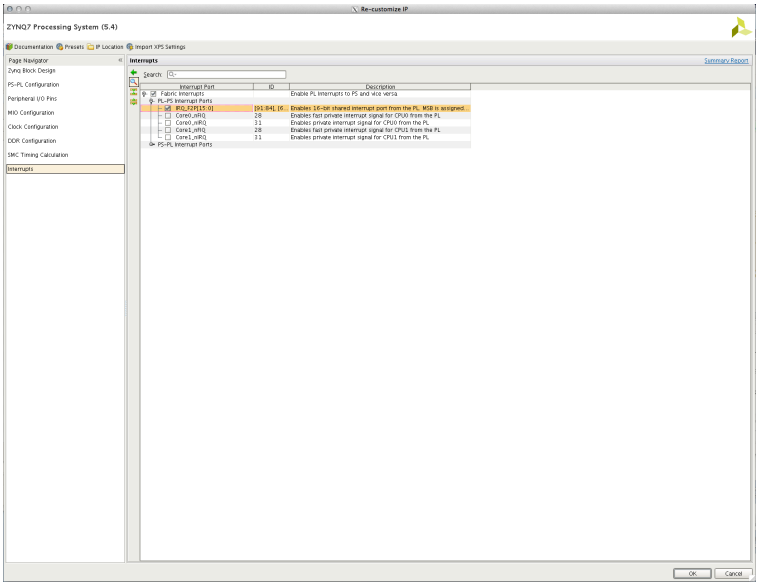


Figura 19 - Configurações do Interrupts.

Depois de todos os blocos desejados, clicar em ***OK***. Observar que os periféricos que foram habilitados estão assinalados como na Figura 20.

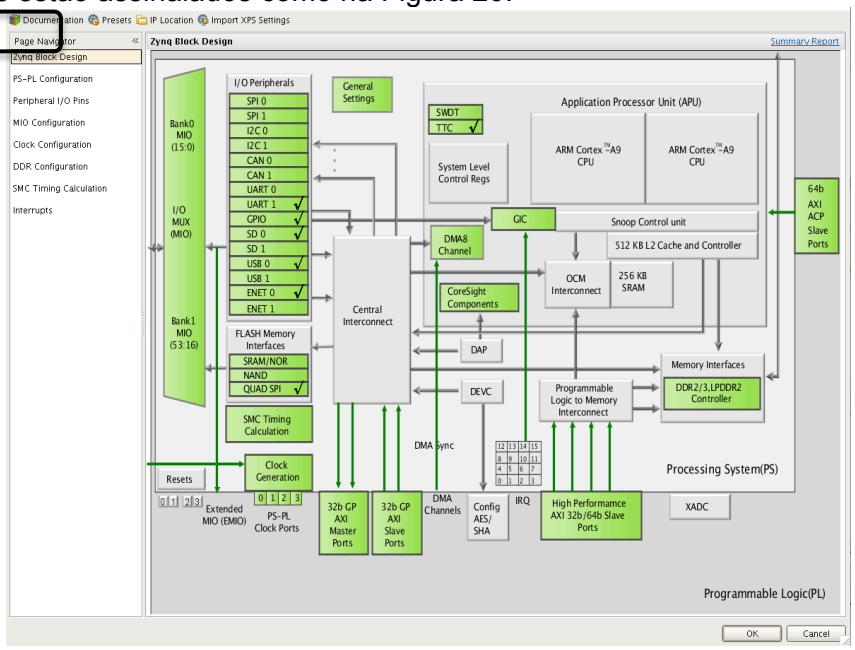


Figura 20 - Blocos após a configuração do IP.

A etapa de configuração do IP que está localizado no PS está concluída, a próxima etapa vai tratar de criar o IP que vai ficar no PL. No projeto corrente é criado um IP que não está no catálogo do Vivado, com o objetivo de mostrar como é possível personalizar um IP.

Para a placa em utilização pode-se fazer esta configuração automaticamente selecionando-se Preset → Zedboard, Figura 21.

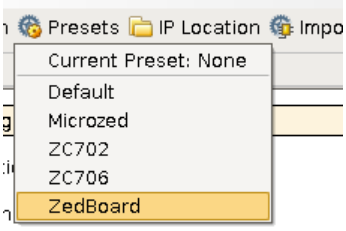


Figura 21 – Configuração padrão para a placa.

Acessando ***Tools*** e ***Create and Package IP***, Figura 22.

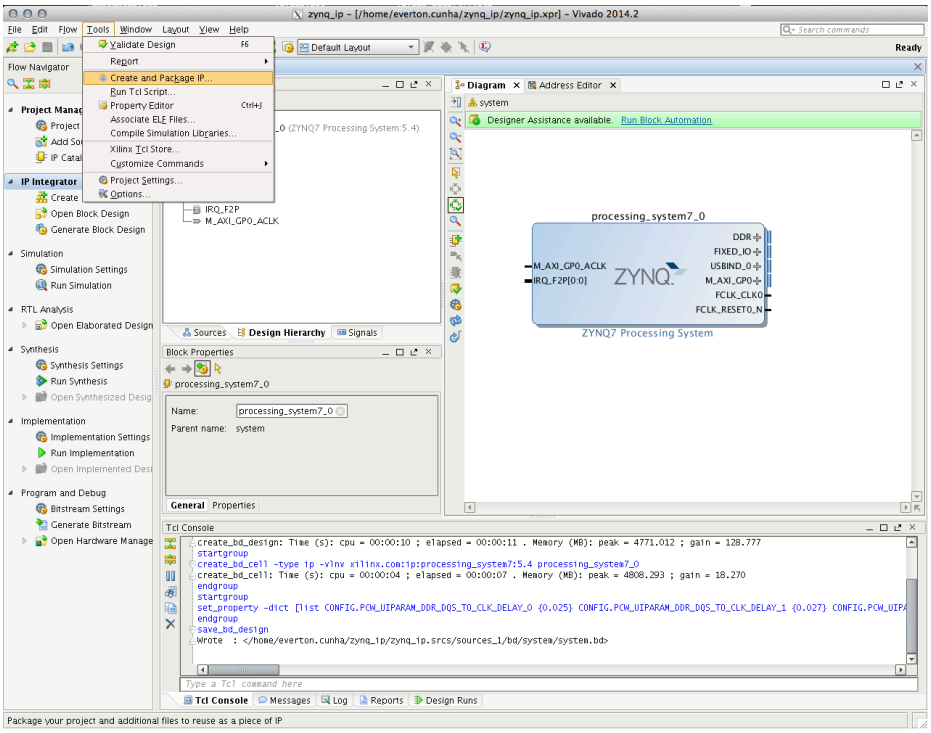


Figura 22 - Adicionar IP personalizado.

É iniciada a configuração do IP, clicando em ***Next***, Figura 23, a próxima etapa é definido o nome do novo periférico que utiliza barramento AXI4, Figura 24.

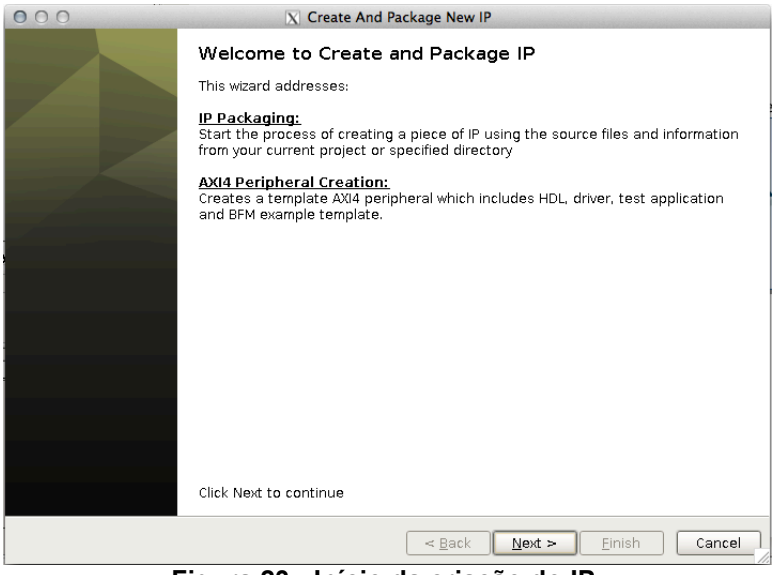


Figura 23 - Início da criação do IP.

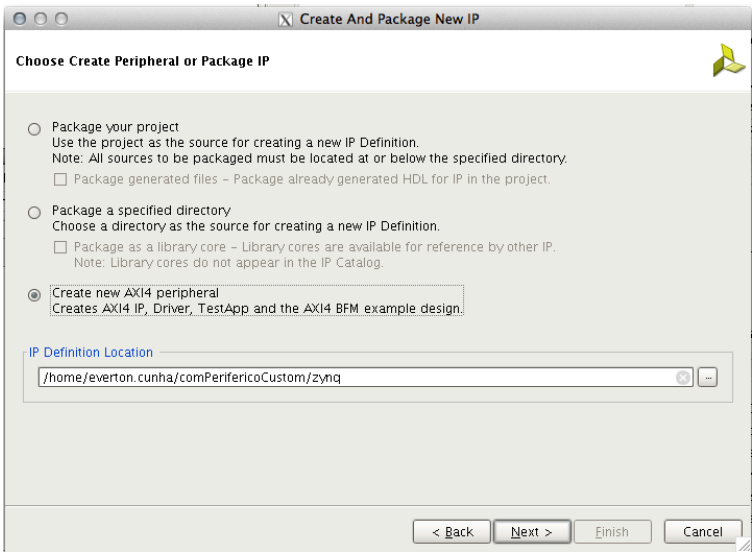


Figura 24 - Definição do caminho do periférico.

Na Figura 25, são definidos: o nome, o versionamento do IP e outras características. Após preencher os campos, clicar em ***Next***. Na próxima etapa, Figura 26, são definidas as interfaces de comunicação com o periférico. Nesta etapa são definidos o nome e o tipo de interface, Figura 27. Configura-se a interface como *master* ou *slave*. Sendo a interface *slave*, é necessário definir o número de registradores que o periférico terá naquele barramento, Figura 28.

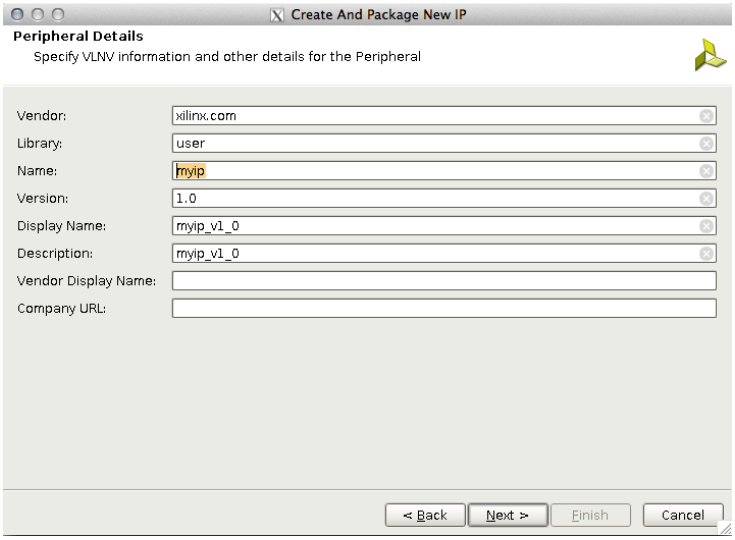


Figura 25 - Definição de informações do periférico.

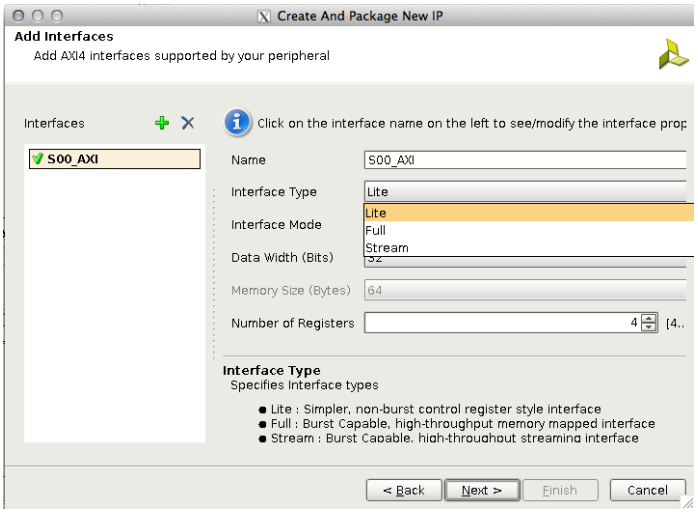


Figura 26 - Configuração de interfaces AXI4 .

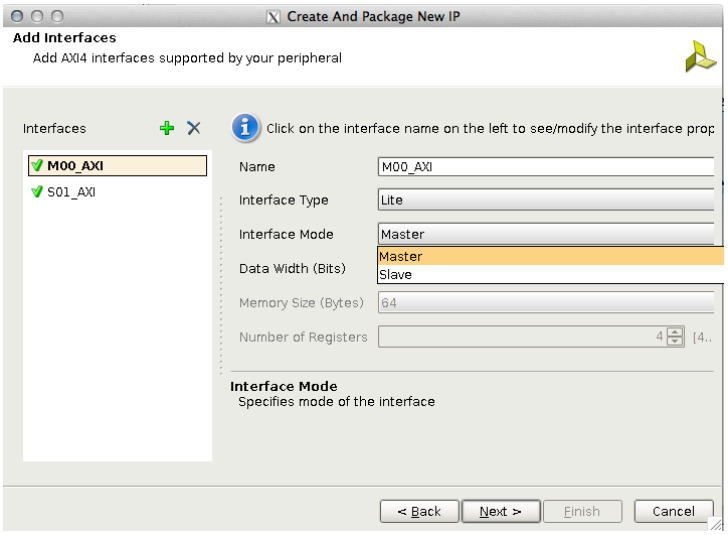


Figura 27 - Definição do modo da interface.

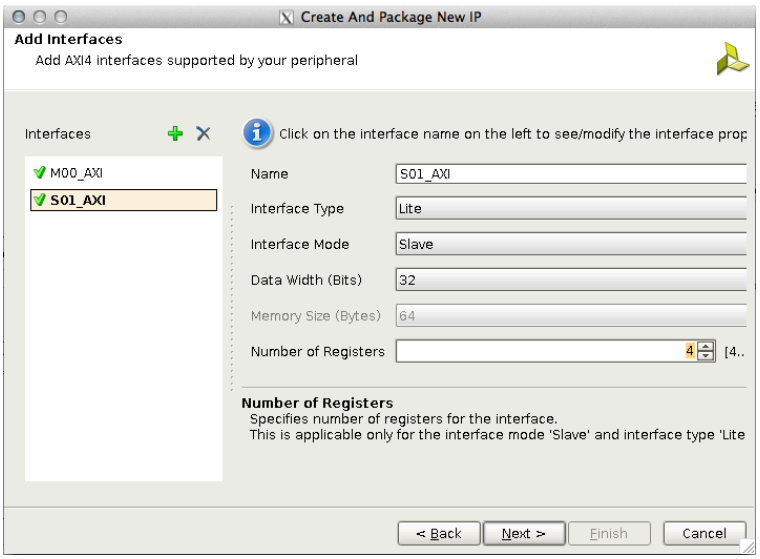


Figura 28 - Número de registradores.

Na Figura 29, selecionando a opção ***Generate Drivers***, é criado o driver para comunicação do software com o hardware. E após clicar em ***Next*** e ***Finish***, Figura 30, o IP será adicionado no catálogo.

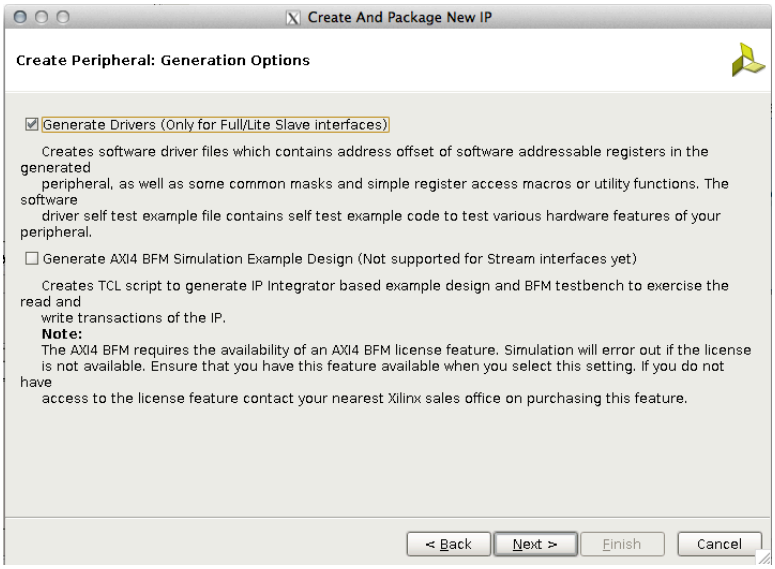


Figura 29 - Habilitando driver para o periférico.

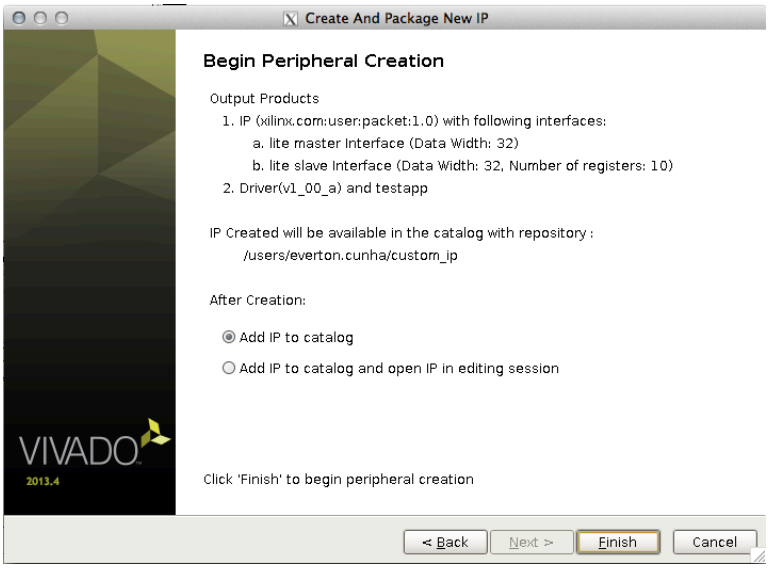


Figura 30 - Finalizando IP.

Neste projeto não é necessário editar o IP criado. Se for necessário editar após concluir a etapa inicial, deve-se marcar a opção ***Add IP to catalog open IP in editing session***, Figura 30. Caso seja necessário realizar edições durante o projeto, apenas clicar com o botão direito em cima do bloco referente ao IP e clicar em ***Edit in IP Packager.***

Clicando no ícone em destaque na Figura 31, é aberto o catálogo e então procura-se o IP criado, que aqui foi definido como ***myip***. O IP ficará disponível para realizar as conexões com os demais componentes do projeto ou o único componente que é referente ao PS. É indicado fazer manualmente esta configuração, por exemplo, seria necessário adicionar um AXI Interconnect para conectar o IP do PS com o IP do PL. Mas para fim de um projeto exemplo, defini utilizar a opção ***Run Block Automation*** que realiza as conexões automaticamente, Figura 31. A Figura 32 apresenta a visualização do layout do atual projeto que está disponível na aba ***Diagram***.

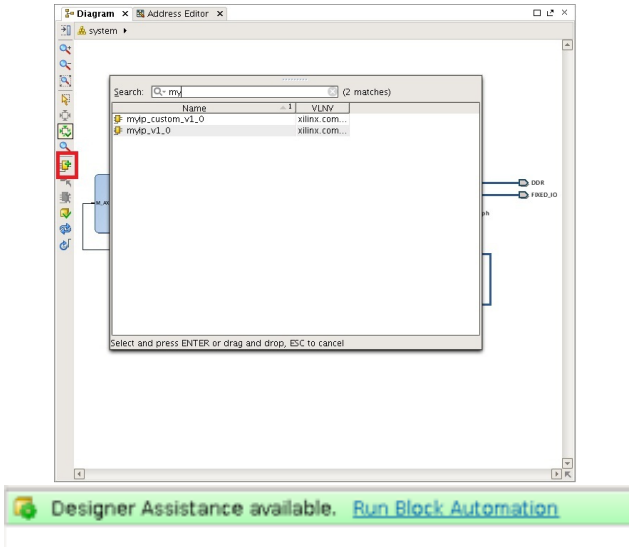


Figura 31 - Adicionando o IP do catálogo e executando conexão automática.

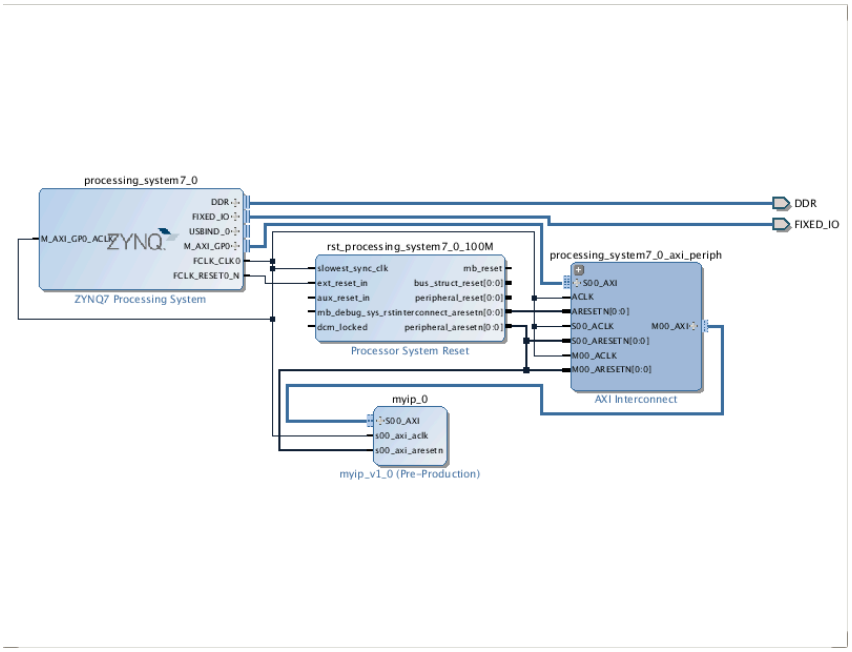


Figura 32 - Layout do Projeto.

Na aba ***Address Editor*** está o endereço referente ao IP, clicar no ícone ***Auto Assign Adress***, Figura 33, para atribuir um endereço, nesta mesma aba pode-se definir o *range* do endereçamento. Após clicar no ícone de validação do projeto, Figura 34, deve retornar um projeto sem erros e uma arquitetura válida.

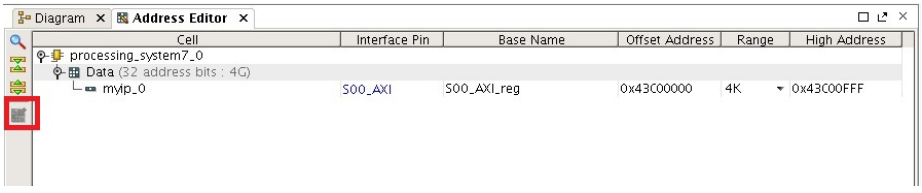


Figura 33 - Endereçamento do periférico.

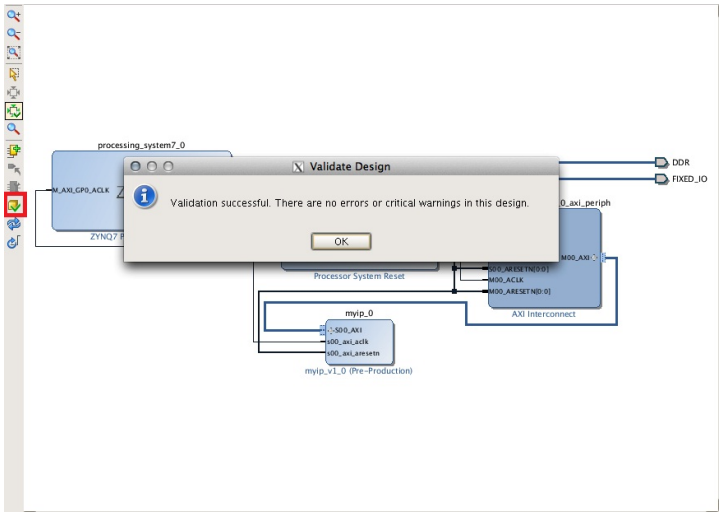


Figura 34 - Validação do projeto.

Completando o projeto com os “blocos” e validando é preciso gerar os arquivos referentes aos IPs criados, clicando com o botão direito no projeto de blocos e em ***Generate Output Products***, Figura 35. Fazendo o mesmo caminho mas agora clicando em ***Create HDL Wrapper*** será criado um VHDL ou Verilog, depende de qual linguagem foi definida no início do projeto, que funcionara com uma camada que envolve o projeto, Figura 36.

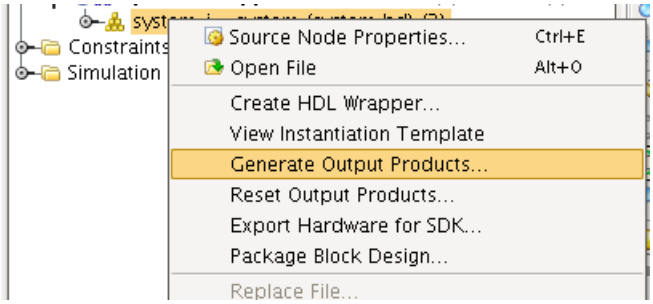


Figura 35 - Gerando Output Products e Create HDL Wrapper.

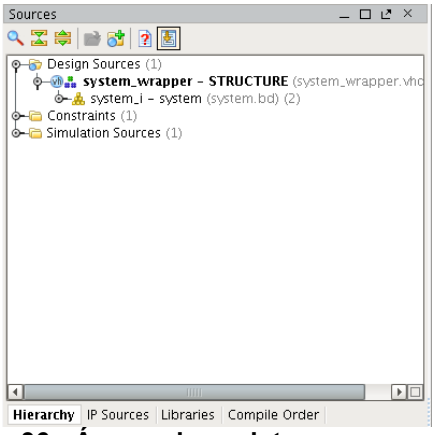


Figura 36 - Árvore do projeto com o *wrapper*.

Agora é feita a etapa de síntese, não ocorrendo erros, clique ***Open Synthesized Design*** onde são definidos os sinais que serão analisados na etapa de *debug*, Figura 37.

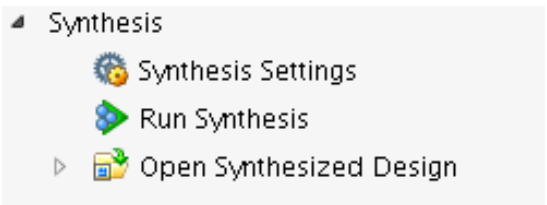


Figura 37 - Etapa de Síntese.

Na Figura 38, está o ambiente da etapa de síntese:
1- O *Netlist*, aqui é visualizado todos os pinos do projeto.
2- Aqui estão os sinais que foram selecionados para serem usados na etapa de *debug*.
3- O esquemático do projeto atual.

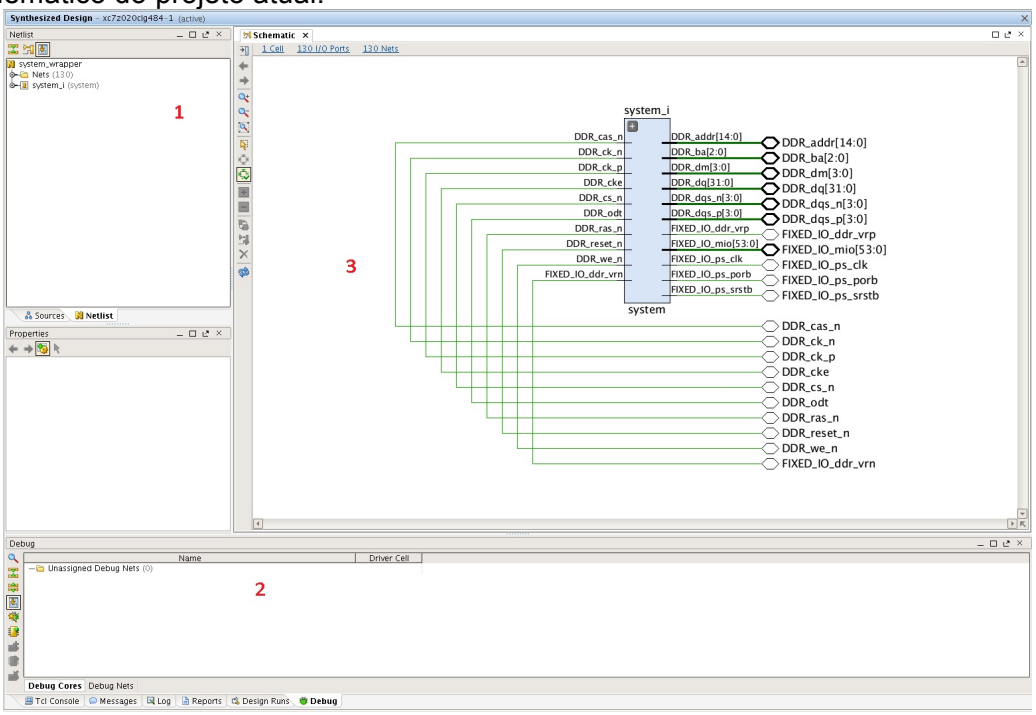


Figura 38 - Visualização do projeto de síntese.

Quando abrir o projeto sintetizado, adicionar os sinais desejados para análise comportamental do hardware , aqui foram selecionados os sinais do barramento AXI, onde pode-se verificar os dados trafegando e os registradores do IP. Para marcar um sinal, clicar com o botão direito e selecionar a opção ***Mark Debug***, Figura 39.

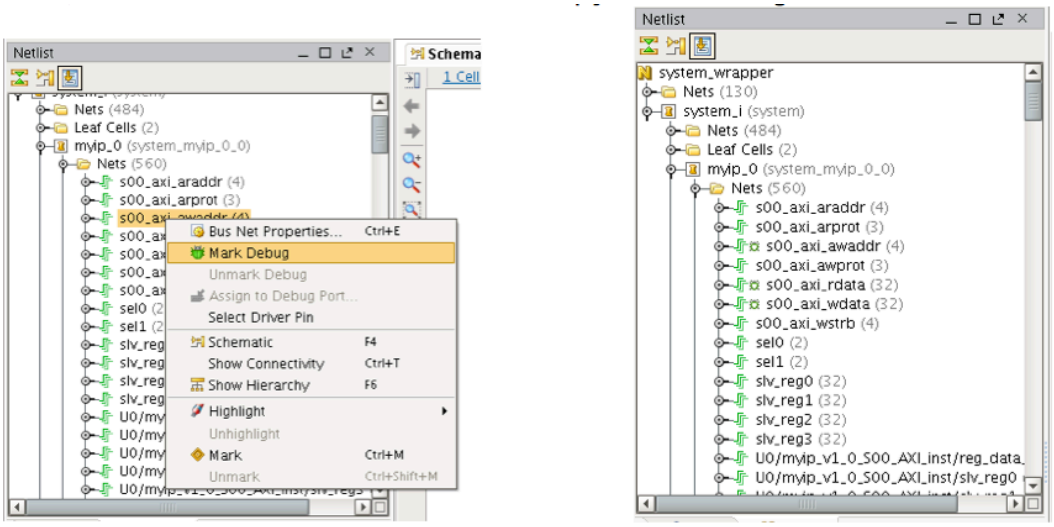


Figura 39 - Marcando sinais para Debug.

Os sinais adicionados serão vistos na parte de baixo na aba ***Debug***, os sinais estarão na pasta ***Unassigned Debug Nets***, Figura 40. Clicando no ícone em destaque que executa a construção do *debug core*.

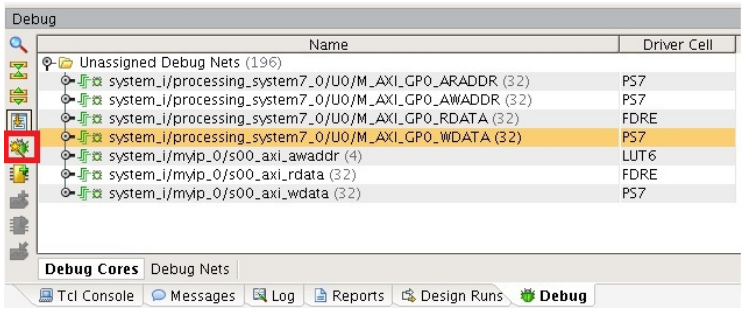


Figura 40 - Sinais definidos como alvo para Debug.

Na Figura 41, é definido um *clock* domínio para os sinais selecionados, e clique em ***Next***.

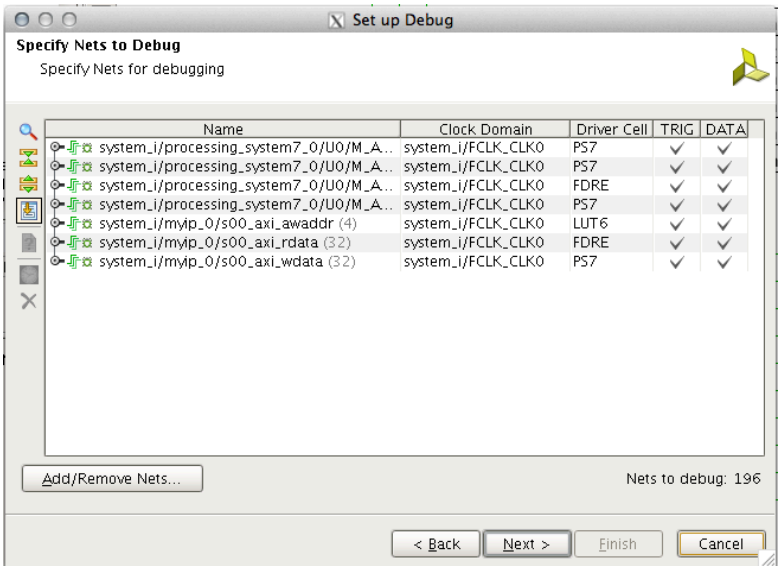


Figura 41- Definição de clock dos sinais.

Habilitar as duas opções de modo de trigger e de captura. Clicar em ***Next*** e em seguida ***Finish***, Figura 42.Na Figura 43, o *debug core* está estabelecido com os sinais escolhidos anteriormente.

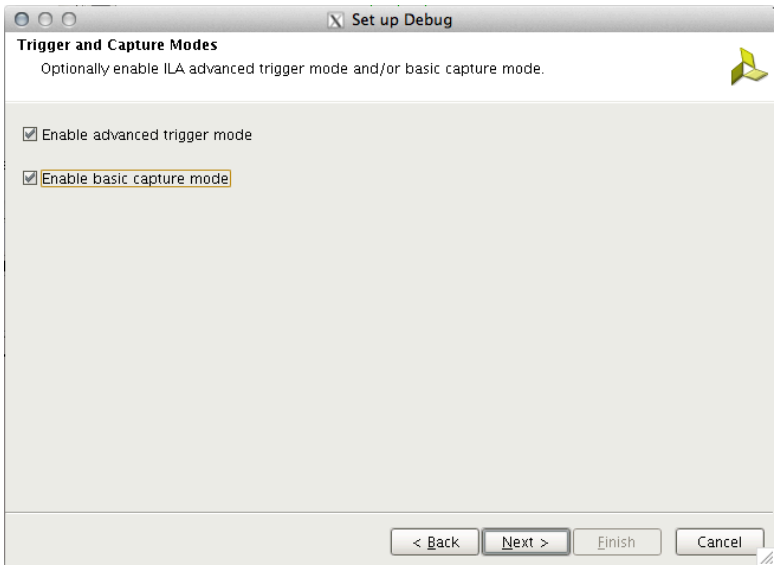


Figura 42 - Habilitando modo de Trigger e modo de captura.

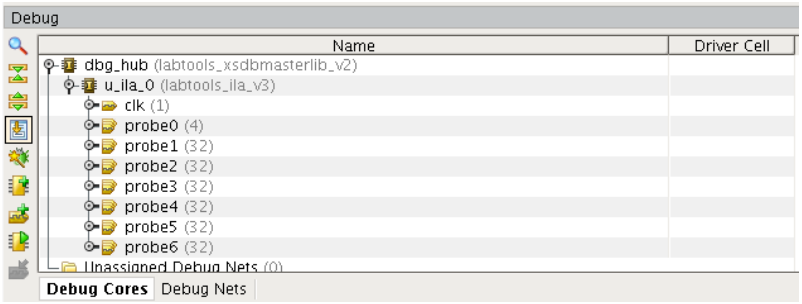


Figura 43 –*Debug Core*.

Para dar continuidade ao projeto, é preciso ir para a etapa de implementação. Mas antes o atual projeto deve ser salvo, surgirá então uma mensagem informando que deve ser criado um arquivo para salvar as *constrains* e que a síntese ficará "desatualizada", Figura 44. Para o primeiro aviso temos a janela, Figura 45, que aparece logo que após clicar em ***OK*** para definir um nome para o arquivo.

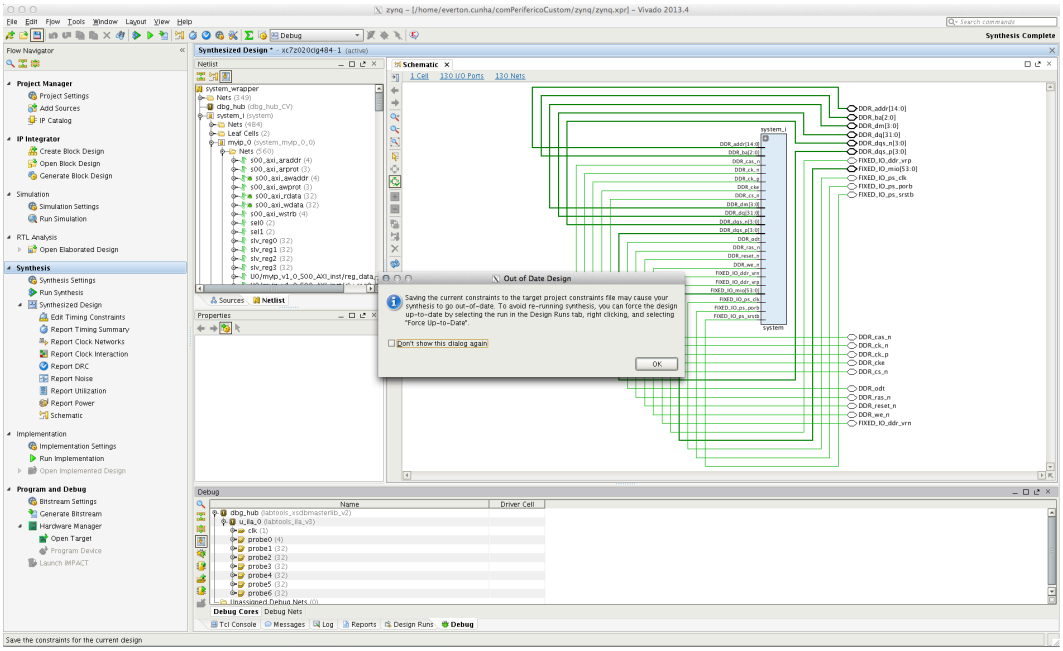


Figura 44 - Informação para criar arquivo de constraints e que síntese ficará desatualizada.

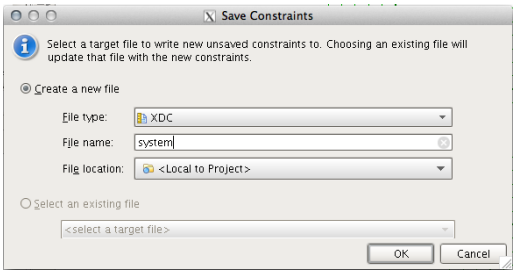


Figura 45 - Nome do arquivo com as constraints.

Para o segundo, pode-se observar na aba ***Design Runs***, como visto na Figura 46, que a etapa de síntese está com um ponto de exclamação, não é necessário realizar a etapa de síntese novamente. Clicando com o botão direito e em ***Force Up-to-Date***, Figura 47. Na Figura 48, já se pode observar que a etapa de síntese está sem o ponto de exclamação.

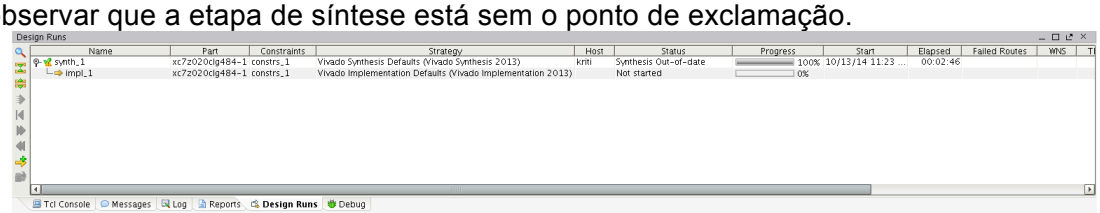


Figura 46 - Informação do andamento das etapas do projeto.

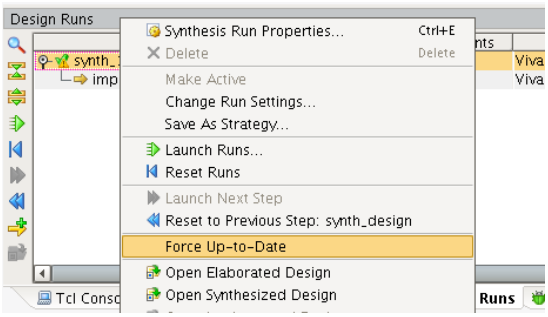


Figura 47 - Informando ao projeto que a etapa de síntese não precisa ser atualizada.

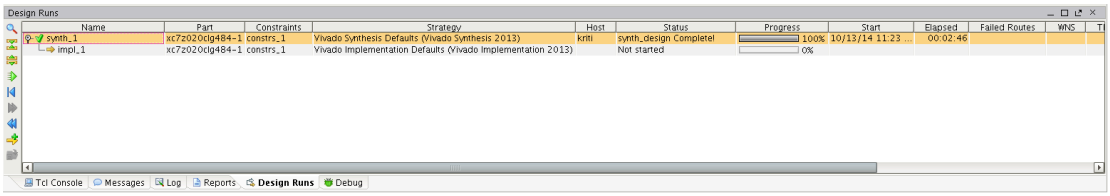


Figura 48 - Informação de síntese atualizada.

A etapa de síntese está concluída, clicando em ***Run implementation*** é iniciada a etapa de implementação, Figura 49.

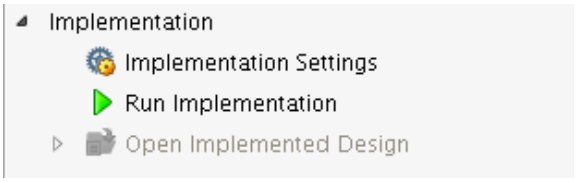


Figura 49 - Etapa de Implementação.

Após concluída a etapa de implementação, pode ser visualizada a ocupação da área lógica(PL), Figura 50. Na Figura 51, as duas etapas aparecem concluídas e sem erros.

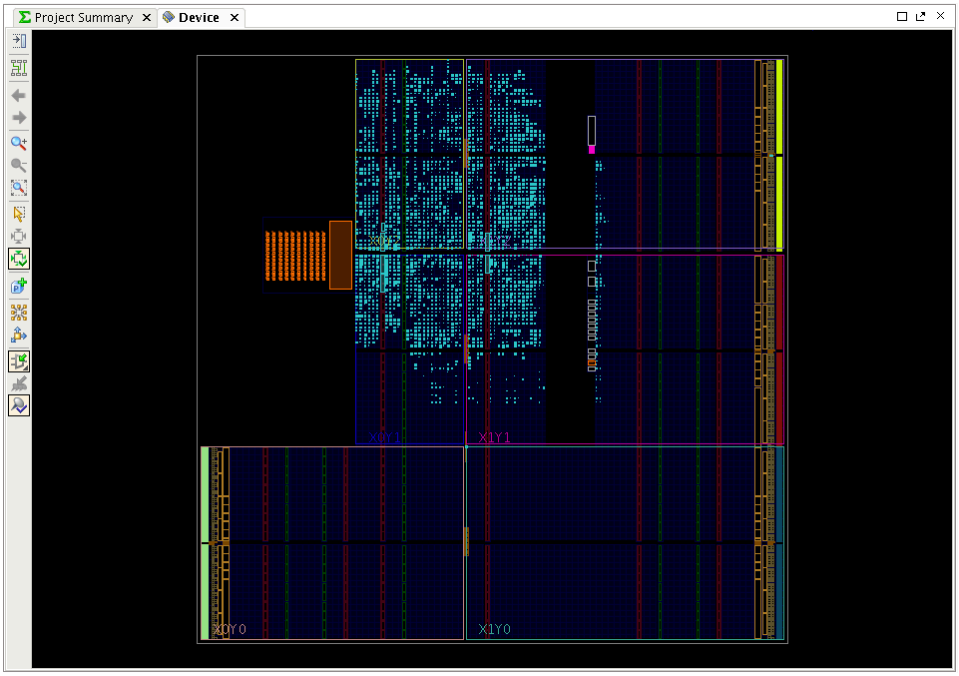


Figura 50 - Projeto de implementação: Visualização da utilização da área da parte lógica.

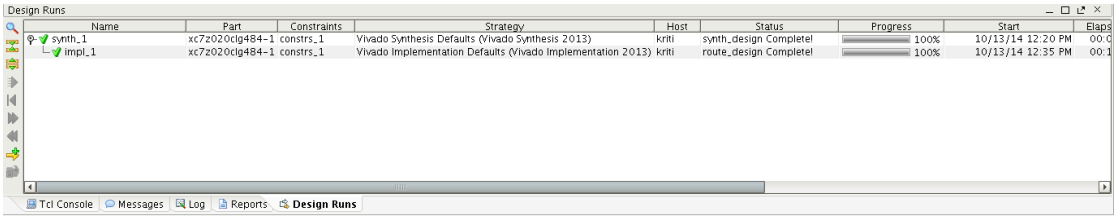


Figura 29 - Etapas de síntese e implementação concluídas.

Agora para realizar a programação da placa será gerado o ***bitstream***. Clicando em ***Generate Bitstream***, Figura 52.

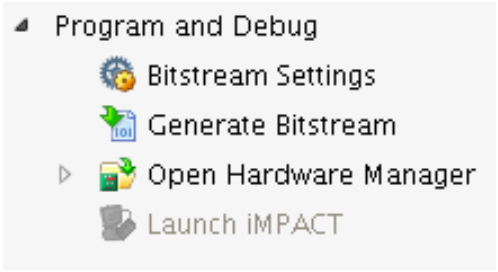


Figura 52 - Etapa de programação e debug.

Na Figura 53, os cabos da jtag(2) e da uart(1) devem estar conectados, pois será utilizado agora e na etapa de execução do software. A disposição dos *jumps* deve ficar como está em destaque(3).

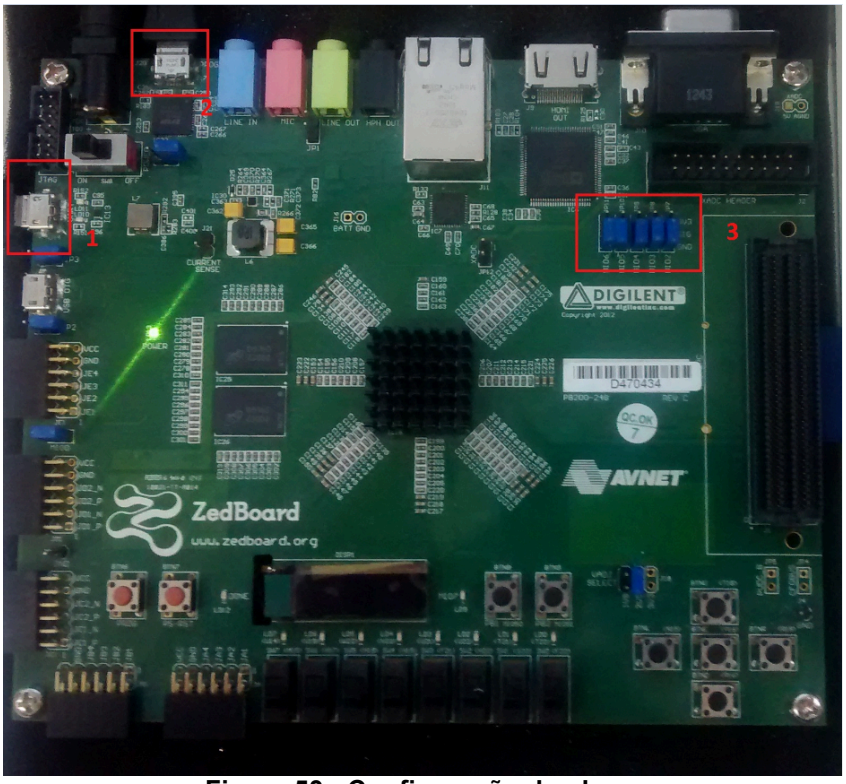


Figura 53 - Configuração da placa.

Clicar em ***Open Hardware Manager***, para iniciar a configuração de programação da placa, Figura 54.

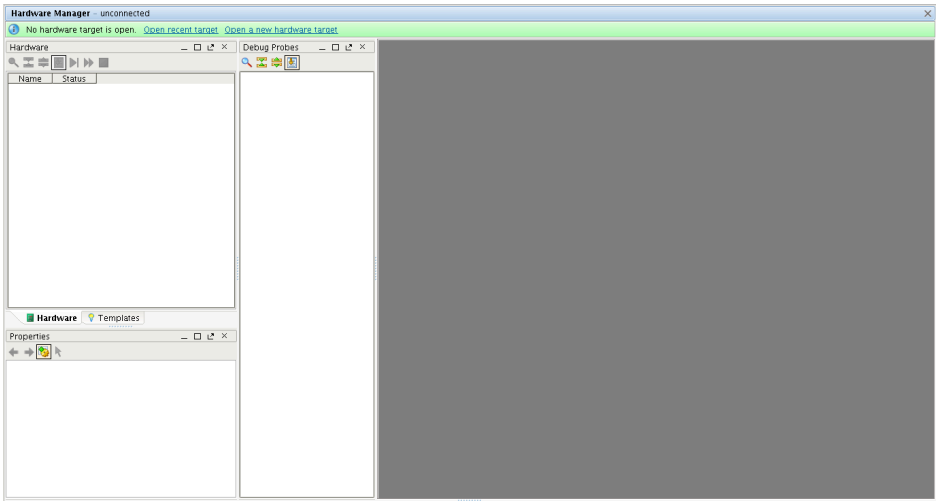


Figura 54 - Gerenciamento do hardware do projeto.

Clicando em ***Open a new hardware target,*** Figura 54, é iniciada a configuração. No projeto, a placa fica em outro computador, caso seja em um computador local, trocar o ip para *localhost*, Figura 56. Na Figura 57, é selecionado o hardware. Na Figura 58, é definida a frequência de comunicação com a placa.

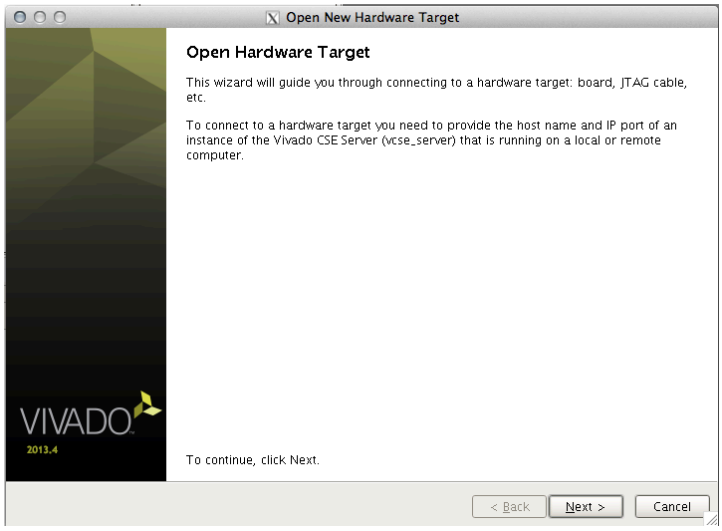


Figura 55 - Início de configuração de comunicação com o hardware.

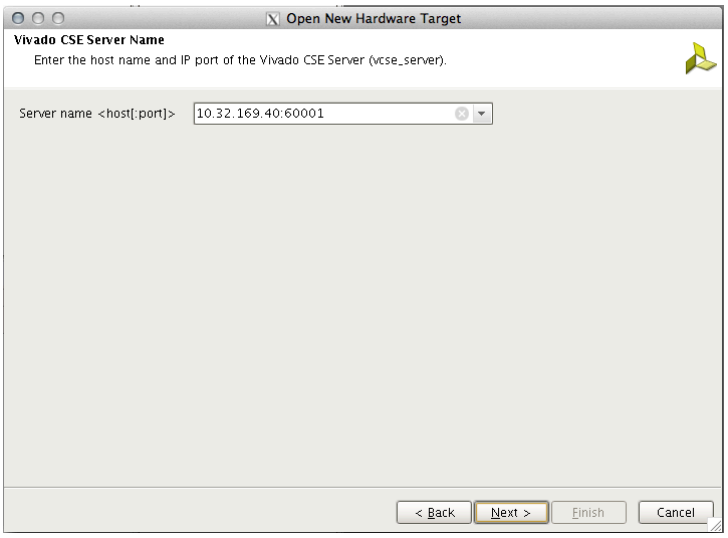


Figura 56 - Definir endereço de rede onde está localizada a placa.

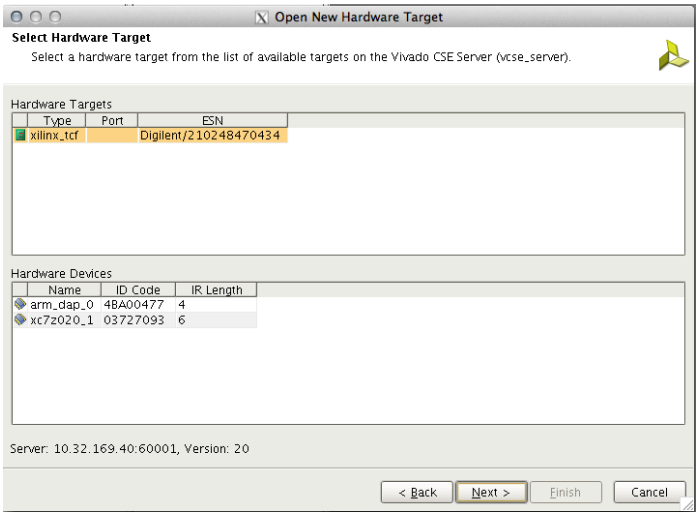


Figura 57 – Hardware Target

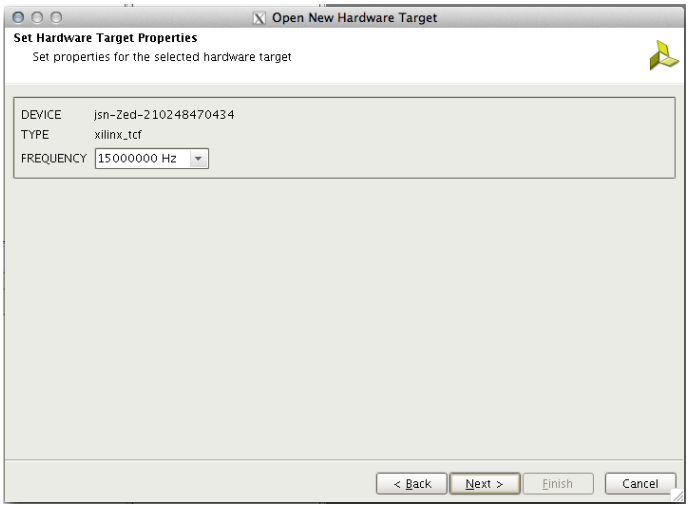


Figura 58 - Definição de frequência de comunicação.

Após clicar em ***Finish***, Figura 59, clicar com o botão direito no dispositivo ou clicar em ***program device***, Figura 60, para enviar o *bitstream*.

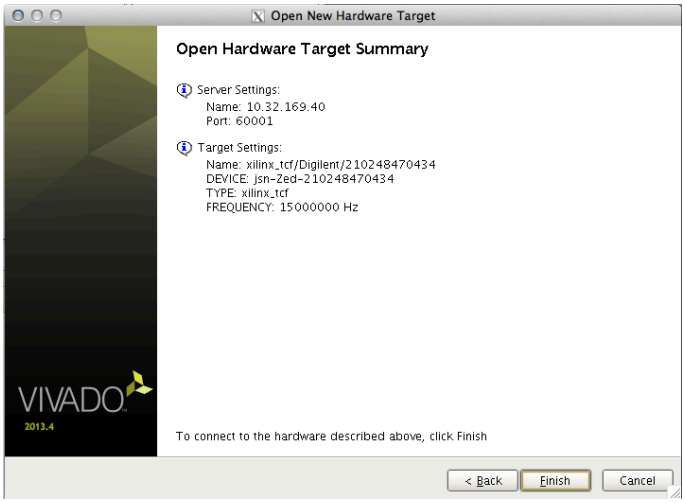


Figura 59 - Informações da comunicação criada.

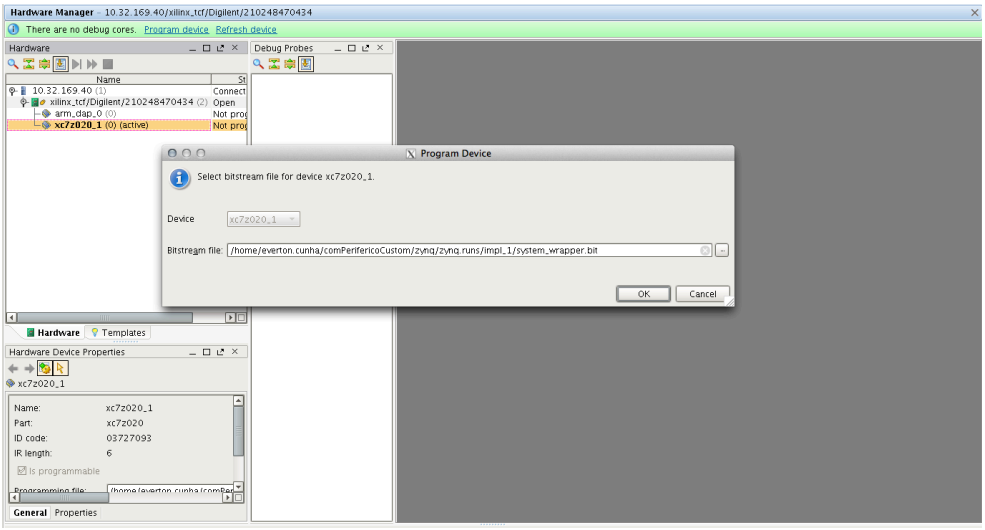


Figura 60 - Enviando o *bitstream* para o hardware.

Na Figura 71, após configurada a placa, o led em destque deve estar aceso; isto informa que a placa recebeu o *bitstream*. Depois de enviado o *bitstream*, uma mensagem de erro aparecerá na tela, Figura 72, informando que nenhum clock foi encontrado. Pois é preciso enviar o .elf para a placa,a aplicação que se encontra no processador irá fazer as configurações para iniciar a aplicação. Para desenvolver a aplicação primeiro deve-se exportar o *hardware* para o SDK.

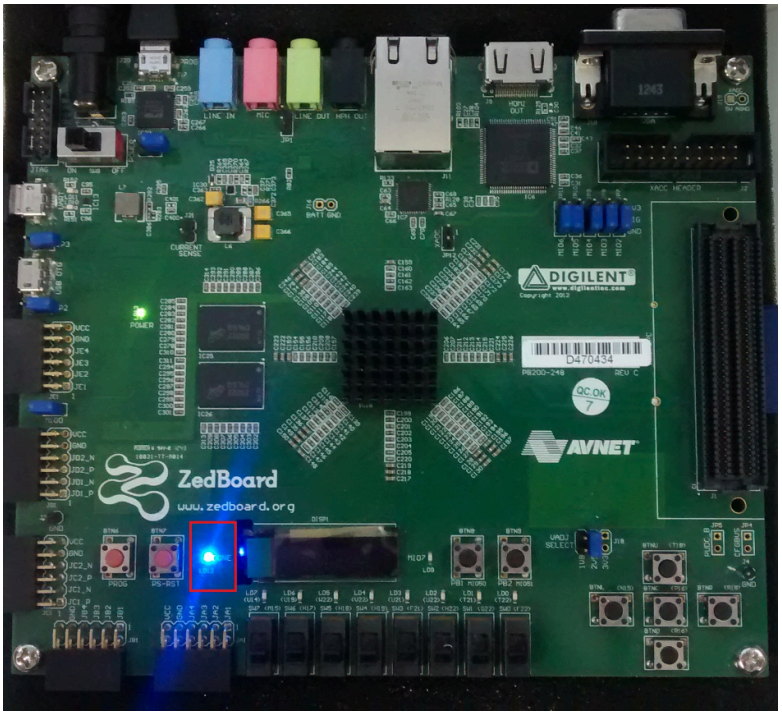


Figura 71 - Placa com o bitstream submetido.

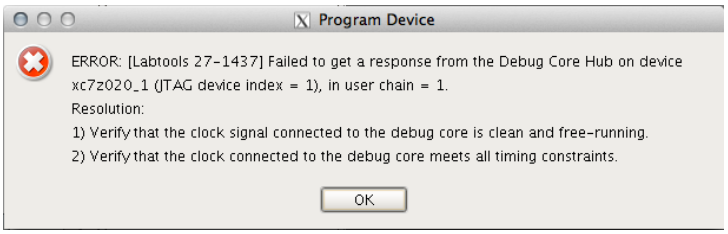


Figura 72 - Mensagem de erro após recebimento do bitstream

Para exportar o projeto, é necessário que o diagrama de blocos esteja aberto, caso contrário ocorrerá um erro, depois de aberto, clicar em ***File***, ***Export*** e ***Export Hardware for SDK***, Figura

73. Selecionar as três opções, Figura 74, e exportar o projeto.

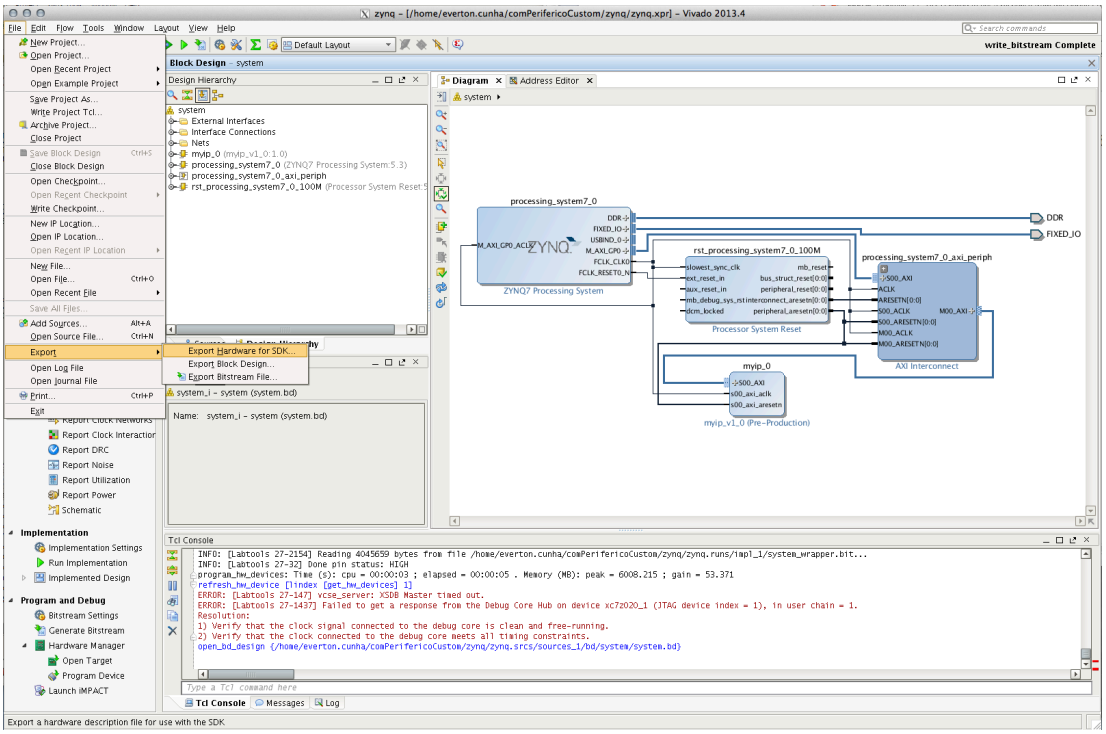


Figura 73 - Exportando hardware para o SDK.

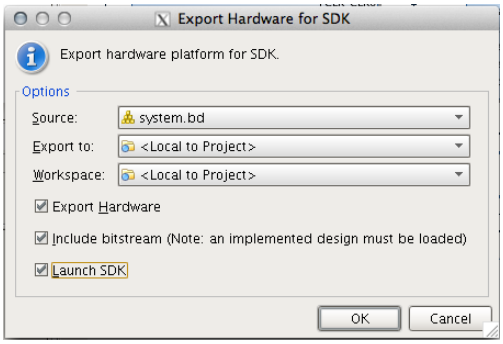


Figura 74 - Definições para exportar hardware.

Após exportar o projeto no Vivado, deve ser implementado o software no SDK, clicando em ***File -> New ->Application Project***. Então são definidos o nome do projeto(1), a plataforma do *hardware*(2) e o nome do suporte para o projeto(3), Figura 75. Escolher a aplicação para teste, Figura 76.

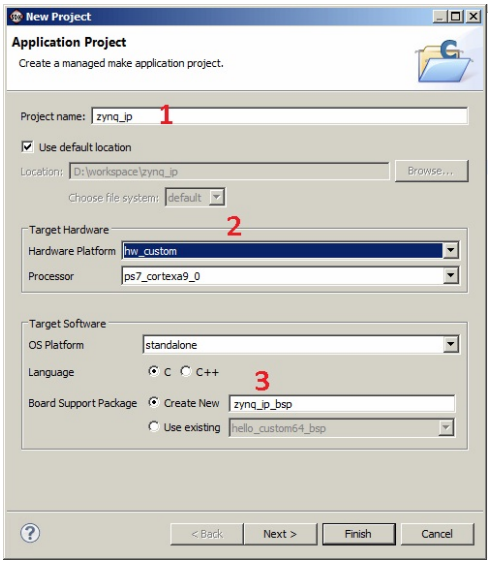


Figura 75 - Definindo parâmetros da aplicação.

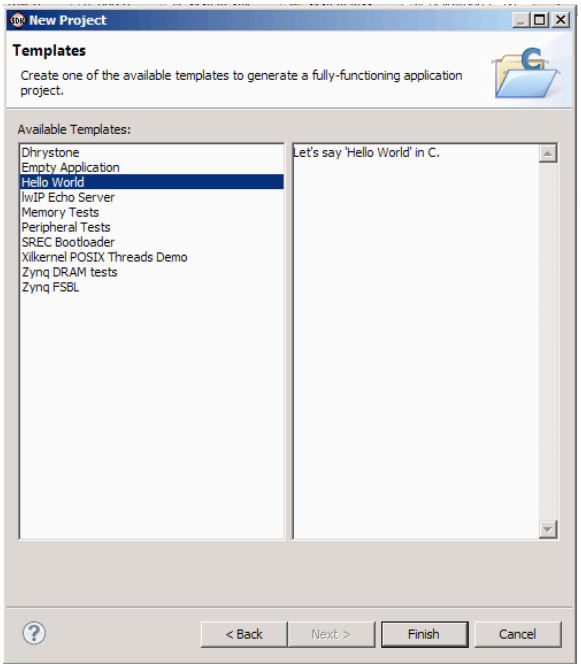


Figura 76 - *Hello World* para testar o projeto criado.

Após clicar em ***Finish***, no ***Project Explorer***, estão os software, pacote de suporte a placa e o projeto de hardware(1). Informações do projeto(2) na barra ***Console***, é mostrado a compilação do projeto pronto. Na Figura 77, é o caminho para iniciar a configuração de execução da aplicação.

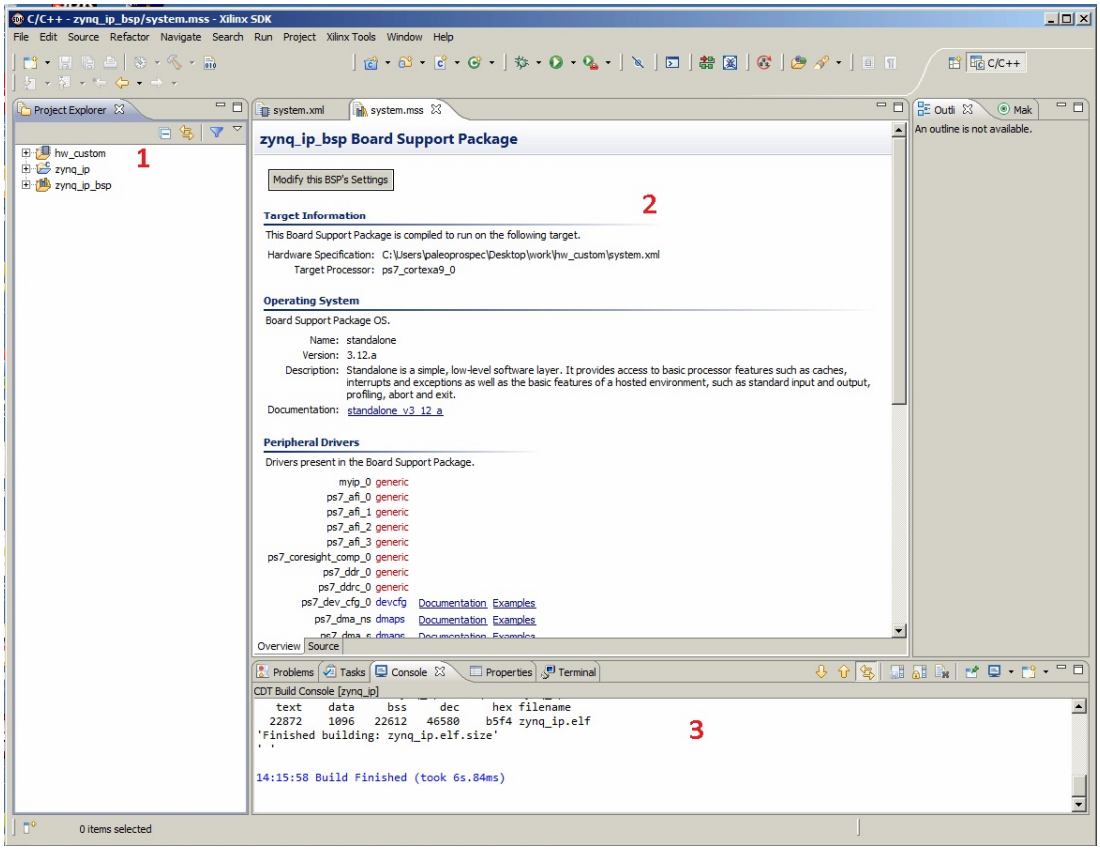


Figura 77 - Ambiente de desenvolvimento do software.

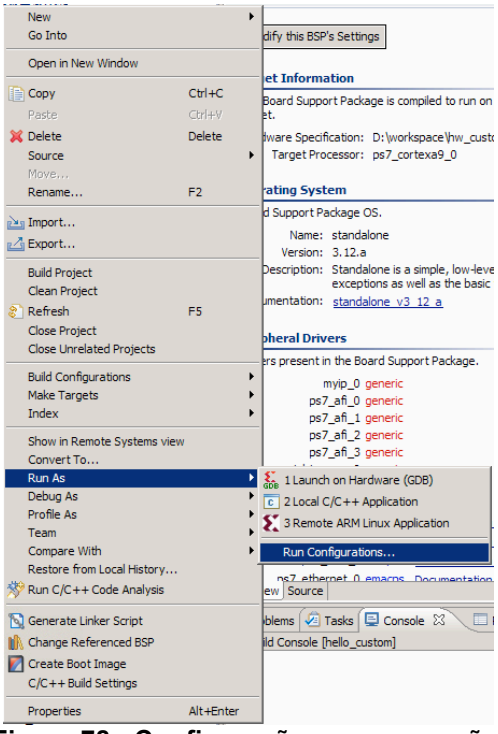


Figura 78 - Configuração para execução.

Seleciona ***Xilinx C/C++ application***(1) e criar novo aplicativo(2), após selecionar o arquivo .elf que foi criado e na aba ***STDIO Connection*** escolher a porta ***Com*** correta e selecionar o ***BAUD Rate*** 115200, Figura 79. Clicar em ***Run***. A saída deve ser algo como na Figura 80.

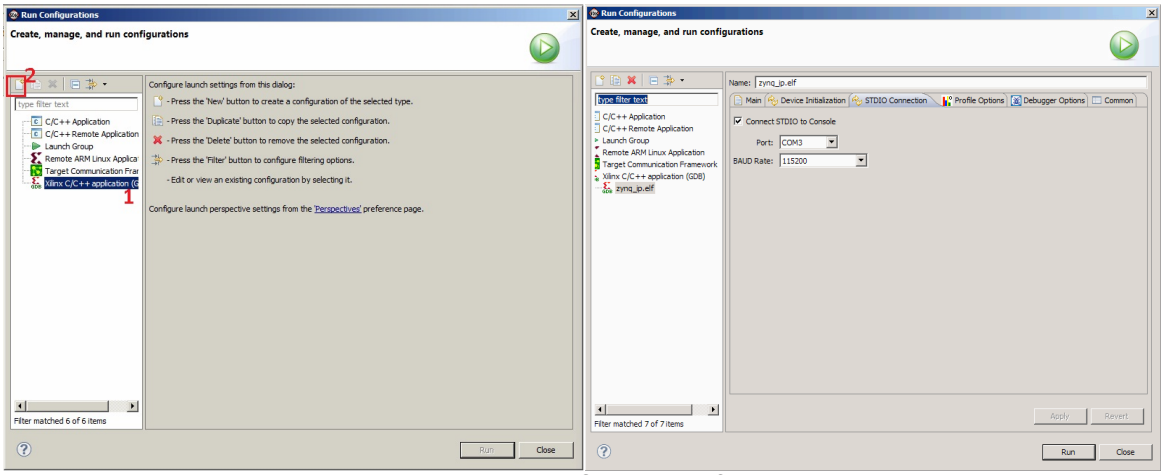


Figura 79 - Criando .elf.

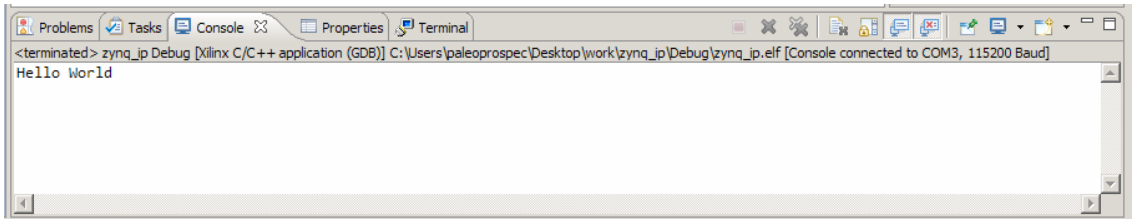


Figura 80 - Hello World executado na placa.

Para implementar um software que se comunique com o IP criado na primeira parte deste tutorial é preciso importar o arquivo referente ao *driver* do dispositivo, que neste projeto encontra-se no diretório “myip_1.0/drivers/myip_v1_00_a/src/”. Na Figura 81 é definido o caminho para importação do *driver*.

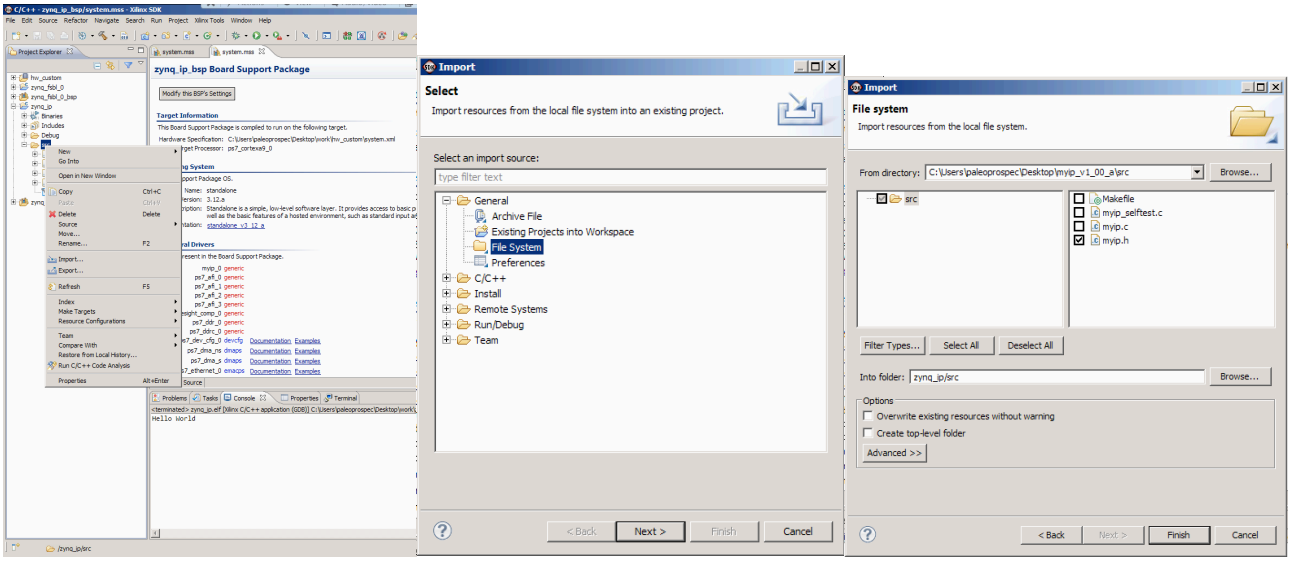


Figura 81 - Importando driver.

O código abaixo é referente ao conteúdo do ao myip.h:

```
#ifndef MYIP_H
#define MYIP_H

/****************** Include Files ******************/
#include "xil_types.h”
#include "xstatus.h"
#include "xil_io.h"
```

```
#define MYIP_S00_AXI_SLV_REG0_OFFSET 0
#define MYIP_S00_AXI_SLV_REG1_OFFSET 4
#define MYIP_S00_AXI_SLV_REG2_OFFSET 8
#define MYIP_S00_AXI_SLV_REG3_OFFSET 12

/**************************** Type Definition ***********************/

#define MYIP_mWriteReg(BaseAddress, RegOffset, Data) \
        Xil_Out32((BaseAddress) + (RegOffset), (Xuint32)(Data))

#define MYIP_mReadReg(BaseAddress, RegOffset) \
   Xil_In32((BaseAddress) + (RegOffset))

XStatus MYIP_Reg_SelfTest(void * baseaddr_p);

#endif // MYIP_H
```

Na aba ***Adress Editor***, localizada no Vivado, é onde estão os endereços referentes aos IPs, localizar o endereço do IP que foi criado, que neste projeto é **0x43c00000**. E que deve ser utilizado no *software* como endereço base, assim como esta no código abaixo. Em seguida tem a saída do programa referente ao código exemplo.

```
#include "myip.h"

#define BASE_ADDRESS 0x43c00000
int main(){
  int valor_atual=0;
  int valor = 0;
  int valor_lido = 0;
  int i;

    init_platform();

    printf("Escrevendo nos registradores \r\n");

    for(i=0;i<=12;i+=4){
            valor_atual = MYIP_mReadReg(BASE_ADDRESS,i);

            printf("Valor no registrador = %i \r\n Entrada:",valor_atual);
            scanf("%i",&valor);
            printf("Escrito %i no registrado\r\n ", valor);

            MYIP_mWriteReg(BASE_ADDRESS,i,valor);
    }
    printf("Lendo os registradores \r\n ");

    for(i=0;i<=12;i+=4){
            valor_lido = MYIP_mReadReg(BASE_ADDRESS,i);
    printf("Valor no registrador = %i \r\n",valor_lido);
    }
    cleanup_platform();

    return 0;
}
```

A saída do programa abaixo:

```
Escrevendo nos registradores
Valor no registrador 1= 0
Entrada:12
Escrito 12 no registrado

Valor no registrador 2= 0
Entrada:18
Escrito 18 no registrado

Valor no registrador 3= 0
Entrada:1986
Escrito 1986 no registrado

Valor no registrador 4= 0
Entrada:2014
Escrito 2014 no registrado
```

Após enviar o arquivo elf, voltar para o Vivado no ***Hardware Manager***, clicar no hardware com o botão direito e em ***Refresh***(1), com isto serão visualizados os sinais do debug core(2) e as propriedades referentes à etapa de captura do debug, Figura 82.

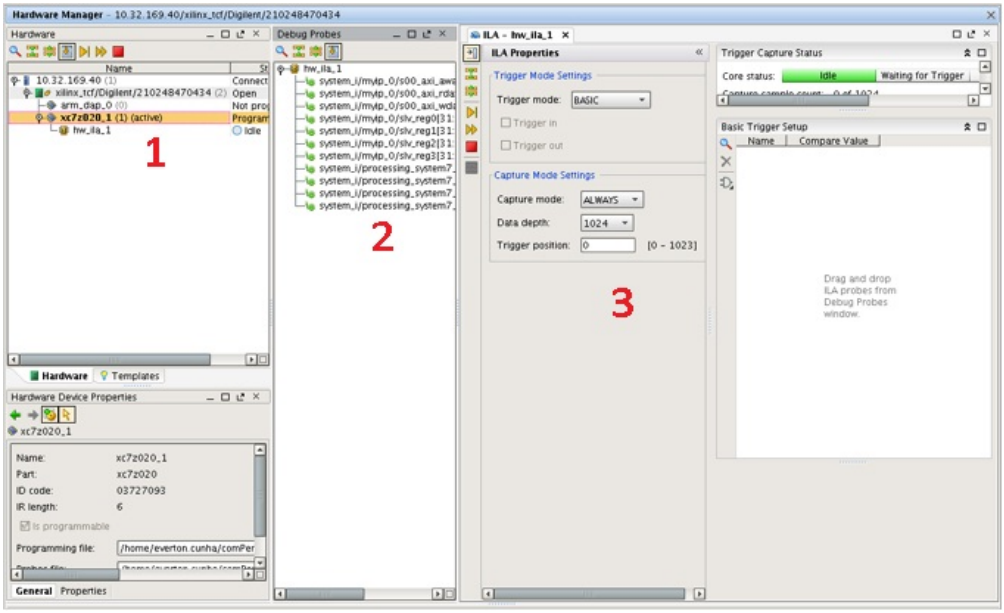


Figura 82 - Ambiente de Debug.

Selecionar em ***Capture Mode Settings***, o modo de captura como básico; com isto é habilitado o campo onde são visualizados os sinais do debug, Figura 83.

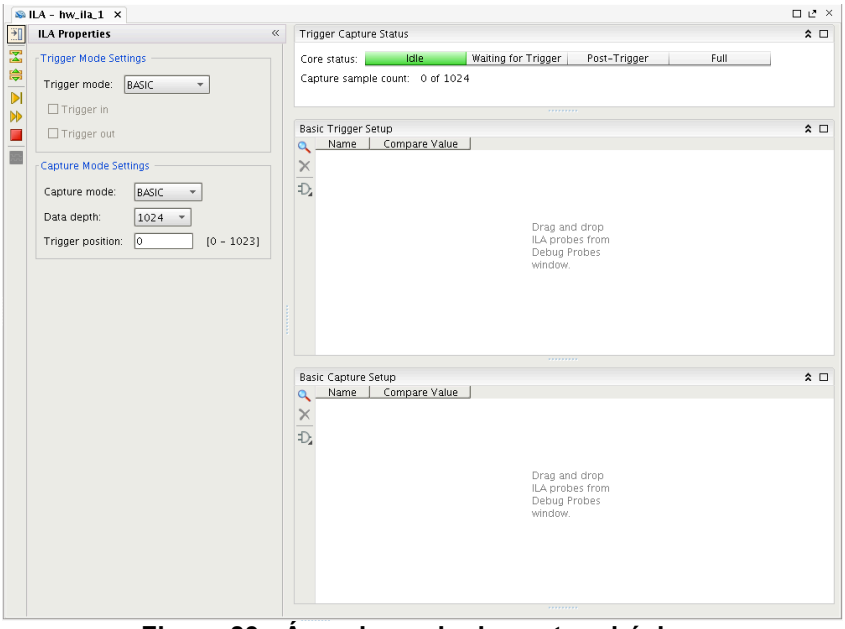


Figura 83 - Área de modo de captura básico.

Selecionar o sinal que deseja ser “monitorado”, quando chegar no valor especificado é mostrado até então o comportamento do sinal no *hardware*, no exemplo o s00_axi_wdata referente ao IP criado, Figura 84.

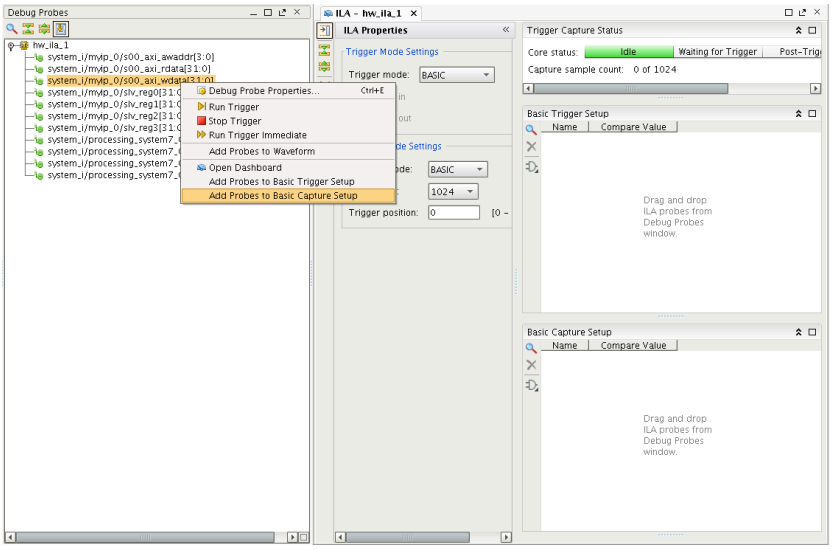


Figura 84 - Adicionando sinais para captura.

Como o último valor que será recebido é 2014 em hexadecimal 0x7DE, Figura 85, é definido que quando este valor for recebido a captura de ondas será mostrada na tela do Vivado.

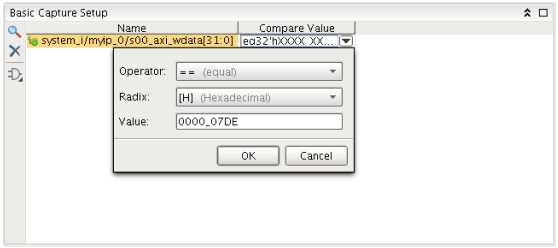


Figura 85 - Definindo que será mostrado o formato de onda dos sinais quando o sinal s00_axi_wdata for igual a 0x7DE.

Clicando no  , ele aguardará a condição ser verdadeira para plotar a forma de onda, clicado em  , é plotado imediatamente a forma de onda. É recomendado utilizar esta opção se não foi estabelecida alguma condição para ser testada. Na Figura 86, no lado esquerdo está o *debug core* aguardando a condição ser verdadeira, e no lado direito já concluída. E com isto é produzida as ondas referentes ao projeto, Figura 87.

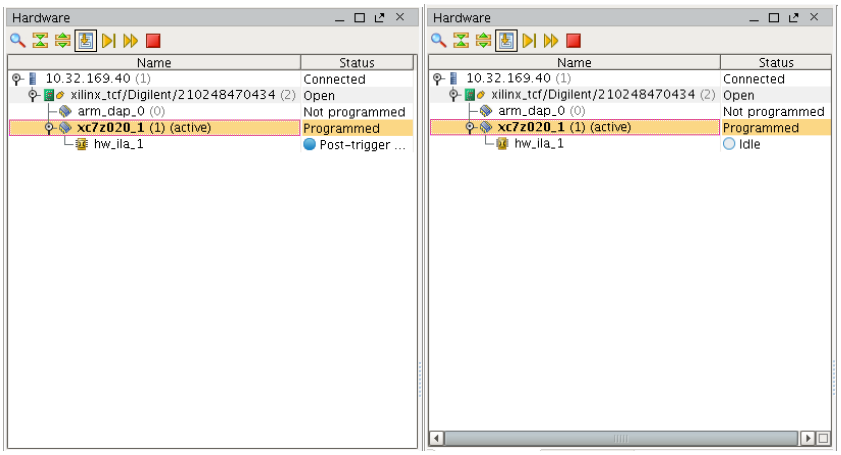


Figura 86- Pré e pós captura.

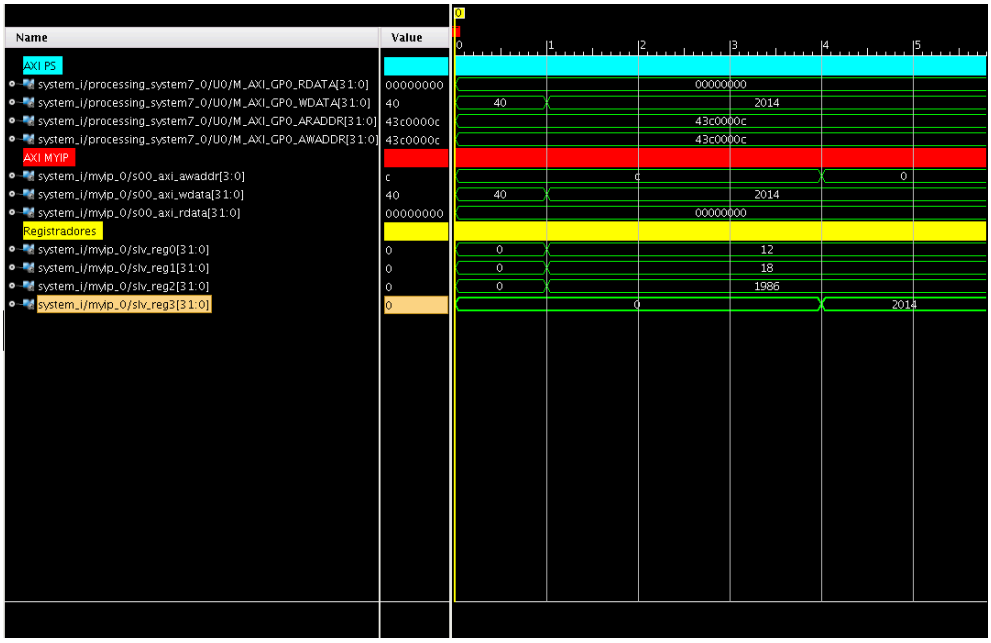


Figura 87 - Forma de onda.