UNIVERSIDADE FEDERAL DO MARANHÃO
CENTRO DE CIÊNCIAS EXATAS E TECNOLÓGICAS
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

JOEL CESARIO DE OLIVEIRA NETO

ANÁLISE DO PROTOCOLO DE COMUNICAÇÃO ADVANCED EXTENSIBLE INTERFACE UTILIZADO EM SISTEMAS DE UM *CHIP*

SÃO LUÍS- MA
2017

JOEL CESARIO DE OLIVEIRA NETO

ANÁLISE DO PROTOCOLO DE COMUNICAÇÃO ADVANCED EXTENSIBLE INTERFACE UTILIZADO EM SISTEMAS DE UM *CHIP*

Monografia apresentada ao curso de Engenharia Elétrica da Universidade Federal do Maranhão, como requisito para obtenção do título de Bacharel em Engenharia Elétrica.

Orientador: Prof. Dr. André Borges Cavalcante

Co-orientador Prof. Dr. Manuel Leonel da Costa Neto.

SÃO LUÍS- MA

2017

Oliviera Neto, Joel Cesario de.
ANÁLISE DO PROTOCOLO DE COMUNICAÇÃO ADVANCED EXTENSIBLE INTERFACE UTILIZADO EM SISTEMAS DE UM CHIP / Joel Cesario de Oliviera Neto. - 2017.
73 f.

Coorientador(a): Manuel Leonel da Costa Neto.
Orientador(a): André Borges Cavalcante.
Monografia (Graduação) - Curso de Engenharia Elétrica, Universidade Federal do Maranhão, CCET UFMA sala 303 bloco 03 LMD, 2017.

1. AXI DMA. 2. AXI Master. 3. AXI Slave. 4. Protocolo de Comunicação AXI. 5. Zynq-7000. I. Cavalcante, André Borges. II. Costa Neto, Manuel Leonel da. III. Título.

JOEL CESARIO DE OLIVEIRA NETO

ANÁLISE DO PROTOCOLO DE COMUNICAÇÃO ADVANCED EXTENSIBLE INTERFACE UTILIZADO EM SISTEMAS DE UM CHIP

Monografia apresentada ao curso de Engenharia Elétrica da Universidade Federal do Maranhão, como requisito para obtenção do título de Bacharel em Engenharia Elétrica.

Orientador: Prof. Dr. André Borges Cavalcante

Co-orientador Prof. Dr. Manuel Leonel da Costa Neto.

Aprovado em: 18/07/2017

BANCA EXAMINADORA

Prof. Dr. André Borges Cavalcante
Universidade Federal do Maranhão
(Orientador)

Prof. Dr. Manuel Leonel da Costa Neto
Universidade Federal do Maranhão
(1° Examinador)

Prof. Esp. José Roberto Quezada Pena
Universidade Federal do Maranhão
(2° Examinador)

Dedico este trabalho para minha família,

especialmente aos meus pais, Joel e Edinalva..

AGRADECIMENTOS

Agradeço primeiramente a Deus, por sempre estar ao meu lado, principalmente nas horas de reflexões, nos momentos de fraquezas, dor e sofrimento.

Agradeço imensamente a minha família, em especial meus pais Joel Cesario de Oliveira Filho e Edinalva Aquino de Oliveira, que apostaram e acreditaram em mim e sempre ajudaram e me apoiaram em tudo que precisei nesta caminhada, e com certeza sem eles não teria conseguido chegar até aqui.

Agradeço ao meu irmão Symon Sirano Aquino de Oliveira, que sempre me ajudou e apoio durante os meus estudos e fora dele.

Agradeço aos amigos que conquistei no curso, em especial aos meus amigos George Vagner e Rodrigo Macedo, por termos compartilhado nossas vitórias, derrotas, alegrias, festas e estudos ao longo da jornada acadêmica.

Ao meu orientador Prof. Dr. André Borges Cavalcante, que me deu toda assistência e atenção para que eu pudesse desenvolver este trabalho, apesar de todos os meus defeitos, sempre manteve toda paciência e dedicou o que pôde do seu tempo em função da realização desta monografia.

Aos professores Manuel Leonel da Costa Neto e José Roberto Quezada Pena, que gentilmente aceitaram fazer parte da minha banca e também colaboraram para o meu sucesso, me acolhendo no laboratório de eletrônica aplicada e oferecendo todo o suporte necessário durante o processo de desenvolvimento deste trabalho.

Enfim, a todas as pessoas que mesmo indiretamente contribuíram para me moldar e me tornar um engenheiro eletricista. Meu muito obrigado!

“ Penso noventa e nove vezes e nada descubro; deixo de pensar, mergulho em profundo silêncio - e eis que a verdade se me revela.”

(Albert Einstein)

RESUMO

Atualmente novas tecnologias vêm sendo usadas na indústria e é necessário acompanhar o funcionamento e a aplicação da mesma no meio acadêmico. Dentre essas um destaque é feito para o protocolo de comunicação AXI, usado para fazer a comunicação em sistemas de um *chip*. Neste trabalho temos como objetivo realizar um estudo do protocolo de comunicação AXI. Este estudo foi dividido em duas etapas, na primeira é feita uma abordagem teórica do protocolo de comunicação AXI, onde é explicado o seu princípio de funcionamento, sua arquitetura, como ocorrem às transações e as variações do protocolo AXI. Na segunda etapa é feita uma abordagem pratica do mesmo, onde foram realizados dois experimentos do protocolo de comunicação AXI, utilizando o kit de desenvolvimento Zedboard, que tem como núcleo um SoC Zynq-7000, fabricado pela Xilinx. O Zynq-7000 é composto por duas camadas, onde a primeira é composta por um sistema de lógica programável e a segunda é composta por um sistema de processamento. Juntamente com o Zedboard, foi utilizado o ambiente de desenvolvimento Vivado, que é composto pelos *softwares* Vivado e o SDK, fornecidos pela Xilinx.

Palavras-chaves: Protocolo de Comunicação AXI. AXI *Master*. AXI *Slave*. AXI DMA. Zynq-7000.

ABSTRACT

Currently new technologies are being used in the industry and it is necessary to follow the operation and the application in the academic environment. Among these, a highlight is made for the AXI communication protocol used to communicate on one-chip systems. In this work we aim to carry out a study of the AXI communication protocol. The study was divided in two stages, in the first stage was made a theoretical approach of the communication protocol AXI, where it is explained its principle of operation, its architecture, how they occur to the transactions and variations of the AXI protocol. In the second stage, a practical approach of it was made. Two AXI communication protocol experiments were realized, we used the Zedboard development kit, which has as core a Zynq-7000 SoC, manufactured by Xilinx. The Zynq-7000 consists of two layers, where the first is composed of a *Programmable Logic* system and the second is composed of a *Processing System*. Along with Zedboard, we used the Vivado development environment, which is comprised of Vivado *software* and the SDK, provided by Xilinx.

Keywords: AXI Communication Protocol. AXI *Master*. AXI *Slave*. AXI DMA. Zynq-7000.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE SIGLAS

SoC	System on a *Chip*
PL	Lógica Programável
PS	Sistema de Processamento
FPGA	Field-programmable Gate Array
HDL	*Hardware* Description Language
AMBA	Advanced Microcontroller Bus Architecture
AXI	Advanced eXtensible Interface
ARM	Advanced RISC Machine
DMA	Data Memory Acess

1. INTRODUÇÃO

O primeiro computador eletromecânico foi construído pelo engenheiro alemão Konrad Zuse em 1936, este computador era capaz somente de realizar operações matemáticas básicas, porém ocupava um grande espaço físico[1]. Desde então a tecnologia de construção de computadores vem evoluindo de forma espantosa, permitindo que atualmente sistemas inteiros de computação posam estar embarcados em um único *chip*. Com o advento destes sistemas mais complexos, em um único *chip*, foi necessária a criação de protocolos de comunicação mais avançados para realizar a comunicação entre os componentes internos, bem como, com os periféricos do sistema. Entre os vários protocolos de comunicação, destaca-se o protocolo de comunicação AXI. Na Figura 1.1 tem-se o diagrama de blocos do processador ARM1176JZF-S, que utilizar o protocolo de comunicação AXI[2].

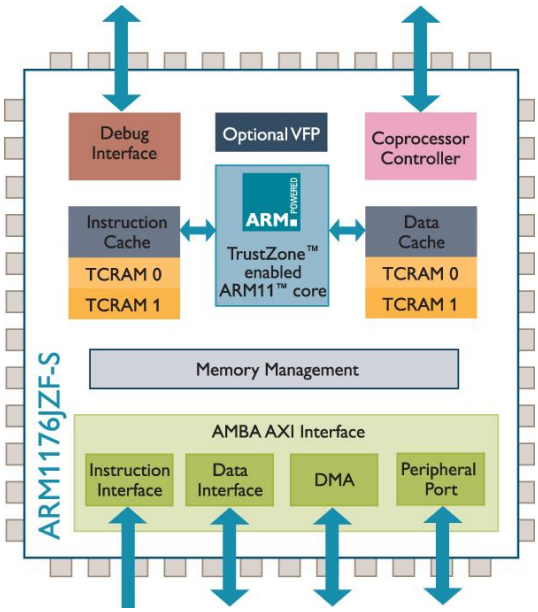

Figura 1.1: DIAGRAMA DE BLOCOS DO PROCESSADOR ARM1176JZF-S
Fonte: www.embeddedinsights.com/epd/arm/arm-arm1176jz-s-arm1176jzf-s.php

O protocolo de comunicação AXI, do inglês *Advanced extensible interface*, é um protocolo de comunicação pertencente à família ARM AMBA, que é uma família de barramentos e protocolos de comunicação introduzida pela ARM em 1996[3]. Este protocolo é utilizado para realizar a comunicação e a interconexão entre os vários módulos dentro do sistema computacional, e com os seus periféricos, como por exemplo, a CPU com a memória. O protocolo AXI é amplamente utilizando em sistemas que necessitam de um alto desempenho e uma alta frequência na transmissão de dados. Este protocolo divide o sistema basicamente em

duas classes, *Slave* e *Master*, onde AXI *Master* é o módulo do sistema que inicia a transação de dados e AXI *Slave* é o módulo do sistema que atua como alvo da transação de dados, recebendo e respondendo a transação de dados. O protocolo AXI é dividido em duas categorias:

1. Memória Mapeada;
2. *Stream*.

No protocolo AXI de Memória Mapeada os dados são enviados separadamente, leitura e escrita, e com tamanho limitado, já no AXI *Stream* os dados são enviados em um fluxo unidirecional e de forma ilimitada. Atualmente é utilizada a segunda versão do protocolo de comunicação AXI, lançada em 2010, esta versão é denominada AXI4[3]. O protocolo de comunicação AXI é amplamente utilizado na indústria, como por exemplo, nos SOC (*system-on-chip*) da Xilinx[4]. Na Figura 1.2 tem-se um diagrama simplificado do SOC ZYNQ-7000, que utilizar o protocolo de comunicação AXI, para realizar a comunicação entre seus componentes e com seus periféricos.

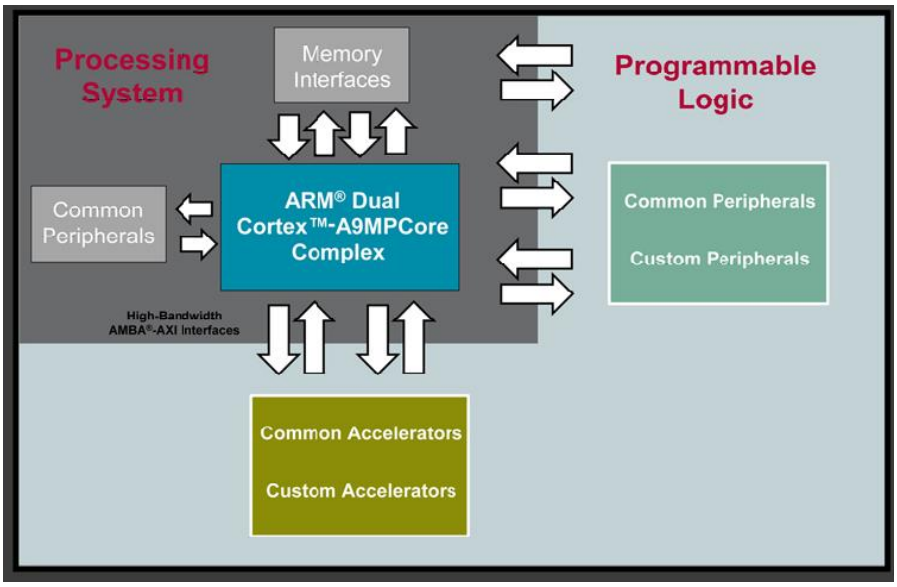

Figura 1.2: DIAGRAMA DO SOC ZYNQ-7000
Fonte: blog.csdn.net/g_salamander/article/details/48161975

Para o estudo do protocolo de comunicação AXI, foi utilizado neste trabalho o SOC da Xilinx, o ZYNQ 7000, que consiste em um dispositivo dividido basicamente em duas partes:

1. PS (*Processing System*);
2. PL (*Programmable Logic*).

A PS, que pode ser traduzida como sistema de processamento, essa unidade é basicamente constituída por uma unidade de processamento de dados, onde o coração é um processador ARM Cortex-A9, com arquitetura de 32 bits, uma boa velocidade de

processamento e um baixo consumo de energia. A PL pode ser traduzido como lógica programável, esta unidade é basicamente constituída por um FPGA (*Field Programmable Gate Array*) que são dispositivos programáveis que permitem a implementação de circuitos digitais para aplicações lógicas[4]. Essas duas unidades são interconectadas através do protocolo de comunicação AXI4.

No entanto, este protocolo pode ser demasiadamente complexo devido ao grande número de canais ou sinais para envio de dados e controle. Apesar disto, existe uma carência de materiais em português para o estudo básico do protocolo de comunicação AXI. Este trabalho apresenta uma introdução ao protocolo AXI de forma simplificada, focando nos principais grupos de sinais e removendo detalhes desnecessários para o entendimento. Além disso, este trabalho apresentará duas implementações, sendo, uma interface AXI *Slave Lite* que representa uma forma simplificada do procolo AXI de Memoria Mapeada com uma comunicação extremamente rápida e de pequeno custo em termos de área em silício, e uma interface AXI DMA que realiza a comunicação entre as duas categorias do protocolo de comunicação AXI, ou seja, ele realiza a interface de comunicação entre o protocolo AXI de Memoria Mapeada e o Protocolo de AXI *Stream*.

1.1 OBJETIVO

Neste trabalho tem-se como objetivo geral analisar o funcionamento do protocolo de comunicação AXI, utilizado em sistemas embarcados de um único *chip* e demonstrar duas implementações, sendo, uma implementação de uma interface AXI *Slave Lite* e uma interface AXI DMA.

Para contemplar este objetivo foram realizados os seguintes objetivos específicos:

- Apresentar a Arquitetura do protocolo de comunicação AXI.
- Apresentar o funcionamento da transmissão, leitura e escrita de dados no protocolo de comunicação AXI.
- Apresentar as variações do protocolo de comunicação AXI.
- Apresentar o funcionamento do protocolo de comunicação AXI de Memória Mapeada, por meio de um experimento.
- Apresentar o funcionamento em conjunto das duas variações do protocolo AXI, por meio de um experimento da interface AXI DMA.

1.2 MOTIVAÇÃO

O protocolo AXI é de fundamental importância no desenvolvimento de sistemas embarcados, sistemas em um *chip* e em processadores para vários fabricantes. Como por exemplo a Hyundai, que utiliza um dispositivo ZYNQ, como núcleo central de processamento, que se comunica através do protocolo AXI para a fabricação de um exoesqueleto, utilizado para auxiliar a mobilidade de pessoas portadoras de nercessidades espciais ou idosos[5]. Na Figura 1.3 tem-se um foto do exoesqueleto fabricado pela Hyundai.

Figura 1.3: EXOESQUELETO DA HYUNDAI
Fonte:https://forums.xilinx.com/t5/Xcell-Daily-Blog/Hyundai-s-wearable-exoskeleton-restores-personal-mobility-to-the/ba-p/647031

Outro fabricante que utiliza dispositivos com o protocolo de comunicação AXI é a Samsung. Que utiliza em seus smartphones com processadores baseados na arquitetura ARM, qué se comunicam internamente através desse protocolo. Na Figura 1.4 tem-se a imagem de um processador Exynos 4 Quad, utilizado em smartphones da linha Galaxy, que é baseado no processador ARM Cortex-A9[6].

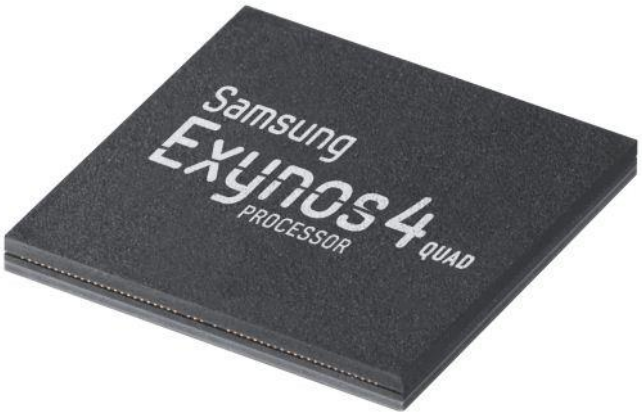

Figura 1.4: PROCESSADOR EXYNOS 4 QUAD
Fonte: https://www.xda-developers.com/no-odin-root-exploit-found-for-exynos-4412-and-4210/

Além da Samsung, varias outras fabricantes de SOC e smartphones utilizam processadores que se comunicam através do protocolo de comunicação AXI, dentre elas podemos citar a Apple, que utiliza em seu smartphone Iphone 5 o processador Apple A6X, que é baseado no processador ARM Cortex-A9[6]. Na Figura 1.5 tem-se a imagem desse processador.

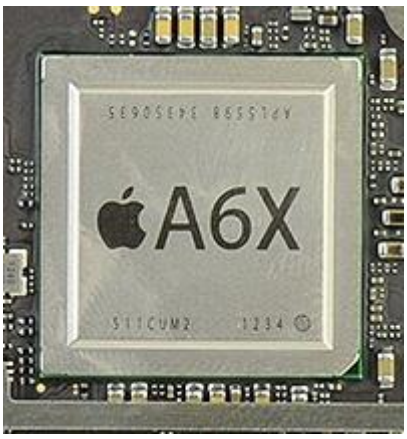

Figura 1.5: PROCESSADOR APPLE A6X
Fonte: http://www.embedded.com/print/4401455

1.3 METODOLOGIA

A seguir são citados os principais passos da metodologia a ser utilizada para o desenvolvimento deste trabalho de monografia.

- Levantamento bibliográfico sobre o protocolo ARM AXI utilizando no dispositivo ZYNQ 7000.
- Estudo das categorias de interface AXI
- Implementação da interface AXI *Slave Lite* em VHDL.
- Implementação da interface AXI *Slave Lite* no Vivado e SDK
- Implementação da interface AXI DMA no Vivado e SDK
- Escrita da monografia, formatação e correção de texto junto ao orientador.
- Documentação final e apresentação do trabalho de monografia.

1.4 ESTRUTURA DO TRABALHO

A seguir é apresentada uma síntese dos assuntos que serão abordados nas próximas seções deste trabalho:

SEÇÃO 02 – FUNDAMENTAÇÃO TEÓRICA: é mostrado toda a base teórica necessária para a compreensão do protocolo de comunicação AXI, onde são apresentados os principais conceitos e as principais tecnologias necessárias para o entendimento deste protocolo, abrangendo um estudo sobre os protocolos de comunicação utilizados em sistemas de informática e sobre o SOC ZYNQ-7000 da Xilinx.

SEÇÃO 03 – PROTOCOLO DE COMUNICAÇÃO AXI4: é abordado todo o funcionamento do protocolo de comunicação AXI, onde são abordadas todas as suas subdivisões, seus principais sinais e suas variações, como também é apresentado o seu funcionamento na transmissão de dados.

SEÇÃO 04 – EXPERIMENTOS E RESULTADOS: é apresentada toda a tecnologia utilizada para a realização de experimentos, utilizando o protocolo de comunicação AXI. Além disso, são demonstradas duas aplicações do protocolo de comunicação AXI, o *software* e o *hardware*, desenvolvido em VHDL, como também o funcionamento das mesmas.

SEÇÃO 05 – CONCLUSÃO: é apresentada uma conclusão sobre o estudo do protocolo AXI, onde é feita uma breve análise sobre o mesmo e demonstrar de forma resumida os experimentos.

2. FUDAMENTAÇÃO TEÓRICA

Para o compreendimento do protocolo de comunicação AXI, além de se realizar o estudo do mesmo, é necessário ter uma base teórica. Neste capitulo são introduzidos os principais fundamentos teóricos envolvidos no estudo do protocolo de comunicação AXI e a tecnologia adotada para as aplicações desenvolvidas com o mesmo.

2.1 MICROPROCESSADORES

Os microprocessadores, geralmente também chamados de processadores ou simplesmente CPU, do inglês *Central Processing Unit*, que pode ser traduzido como UCP, Unidade Central de Processamento. São circuitos integrados que podem ser programados para executar tarefas predefinidas, manipulando e processando dados, ou seja, um processador é um circuito integrado, geralmente fabricado de silício, capaz de realizar processamento de dados e resolver cálculos matemáticos, obedecendo a um conjunto predeterminado de instruções. Sua função é basicamente receber um determinado dado e realizar o seu processamento, conforme uma programação pré-definida, e desenvolver o resultado[7]. Na Figura 2.1 tem-se uma imagem do primeiro processador comercial, o 4004, desenvolvido pela Intel em 1971 para ser utilizado em uma calculadora portátil.

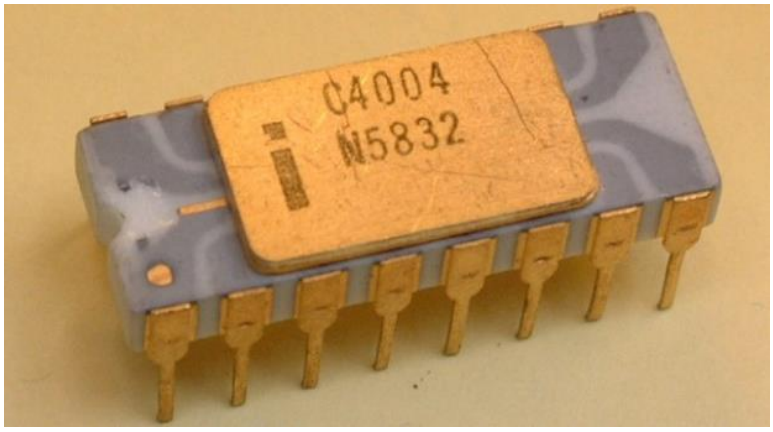

Figura 2.1: PROCESSADOR INTEL 4004
Fonte:http://www.techtudo.com.br/artigos/noticia/2011/11/intel-4004-o-primeiro-processador-da-historia-comemora-40-anos-de-idade.html

Os processadores são compostos por vários componentes básicos, onde cada componente pode realizar uma determinada função especifica no processamento de dados. Na Figura 2.2 tem-se um diagrama de blocos dos componentes básicos de um processador.

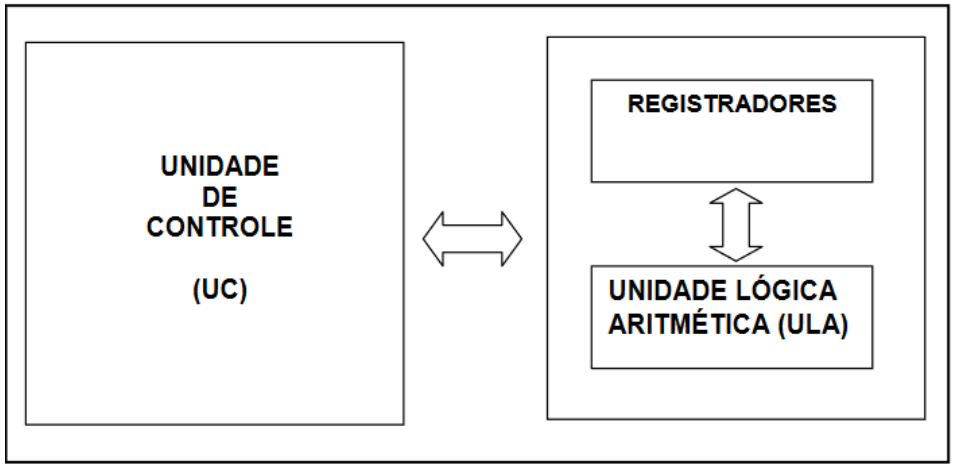

Figura 2.2: COMPONENTES BÁSICOS DE UM PROCESSADOR.
Fonte: Elaborado pelo Autor.

Os principais componentes de um processador são[7]:

- ULA (Unidade Lógica Aritmética) - é um circuito digital implementado dentro dos processadores que é responsável em realizar todas as operações lógicas e aritméticas. Assim, na ULA são executadas efetivamente as instruções dos programas, exceto as instruções de controle e movimentação de dados que são executadas na UC;
- UC (Unidade de Controle) - é a parte dentro dos processadores responsável por realizar o controle de todos os sinais que irão controlar as operações externas do processador, como também fornece todas as instruções para se ter um correto funcionamento interno do processador. Podemos resumir que a UC é a parte do processador responsável pelo controle de todas as ações do mesmo, logo a UC é responsável por realizar o comando de todos os outros componentes;
- Registradores são circuitos digitais capazes de armazenar dados. Os registradores são os meios mais rápidos para o armazenamento de dados. Existem vários tipos de registradores, os mais importantes são:

1. Apontador de instruções (PC) - este registrador guarda o endereço da próxima instrução a ser executada pelo processador;
2. Registrador de instrução (RI) - este registrador armazena a instrução que esta sendo executada pelo processador;
3. Apontador de pilha (SP) - este registrador é responsável por guarda o endereço da pilha de execução do programa que esta sendo executado no processador.

Existem dois tipos de arquitetura de um processador, sendo elas[8]:

1. Arquitetura CISC (*Complex Instruction Set Computer)*, que pode ser traduzido como “computador com um conjunto completo de instruções” - é uma arquitetura para processadores que é capaz de executar um conjunto de diferentes instruções complexas, assim tornando o sistema versátil. Uma importante característica dos processadores com arquitetura CISC é a presença de um microcódigo no processador. Este microcódigo é uma parte do processador que contém uma microprogramação, onde um conjunto de instruções são previamente gravadas no processador. Isto permite que o processador receba as instruções dos programas e as execute utilizando as instruções contida na sua microprogramação, ou seja, com a utilização da microprogramação pode gerar uma quebra nas instruções, que já são em baixo nível, em diversas instruções mais próximas do *hardware*;
2. Arquitetura RISC (*Reduced Instruction Set Computer*), que pode ser traduzido como ”Computador com um conjunto reduzido de instruções” - é uma arquitetura para processadores que se baseia em um conjunto de instruções de menor tamanho e com menor complexidade, onde estas instruções serão executadas basicamente na mesma quantidade de tempo. Nos processadores baseados na arquitetura RISC as instruções não passam pela microprogramação, e serão executadas diretamente no *hardware*. A principal característica da arquitetura RISC é que suas instruções são em quantidades reduzidas e tem um baixo nível de complexidade.

Na Figura 2.3 tem-se um gráfico com as etapas que a instrução passar nos processadores com arquitetura RISC e nos com arquitetura CISC.

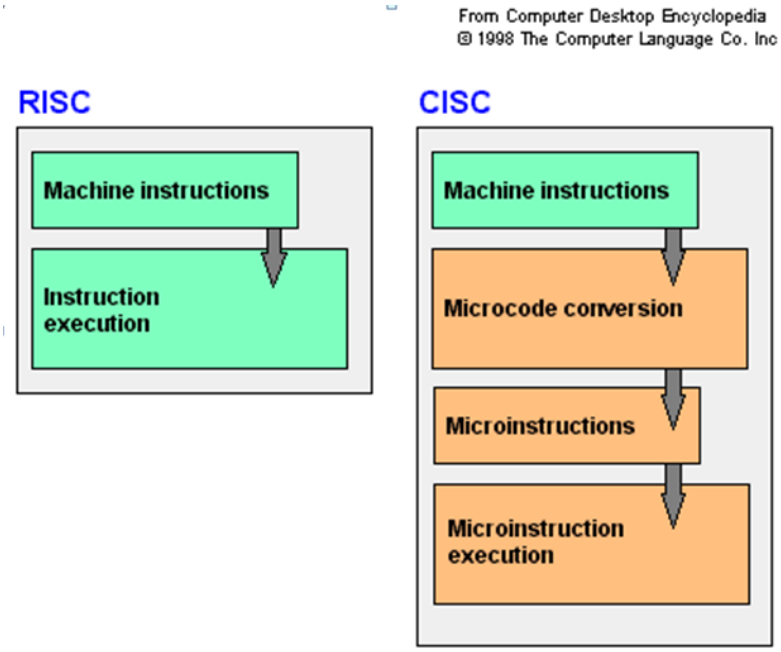

Figura 2.3: COMPARATIVO ENTRE AS ETAPAS DE INSTRUÇÃO RISC VS CISC.
Fonte: electel.blogspot.com.br/2015/07/understanding-processor-architecture-cisc-vs-risc.html.

Nos processadores com arquitetura RISC tem-se menos etapas para se executar uma instrução, gerando assim, um ganho em desempenho em relação aos processadores CISC. Mas, em contrapartida, os processadores CISC possuem microprogramação e tem a vantagem de reduzir o tamanho do código a ser executável no processador, com isso tem-se uma relação entre o tamanho do código versus o desempenho.

2.2 *FIELD PROGRAMMABLE GATE ARRAYS*

O dispositivo FPGA (*Field Programmable Gate Arrays*) é um circuito integrado, feito de silício, formado por semicondutores que são arranjados em matrizes de Blocos Lógicos Configuráveis (CLBs), sendo eles conectados por várias conexões reconfiguráveis. Na Figura 2.4 é ilustrada a estrutura geral de um dispositivo FPGA. Esta conFiguração em blocos permite que os FPGAS possam ser remodelados de acordo com a aplicação desejada.

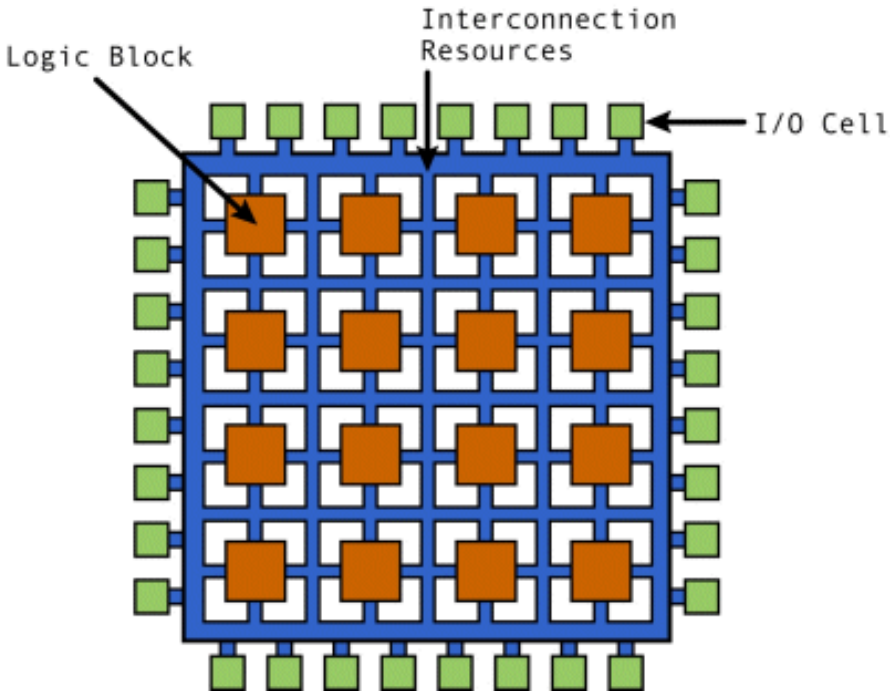

Figura 2.4: ESTRUTURA BASICA DE UM FPGA
Fonte: eetimes.com/document.asp?doc_id=1274496.

Os blocos lógicos configuráveis são os elementos principais que compõem um FPGA, eles são basicamente formados por[9]:

- Multiplexadores, são circuitos selecionadores, onde eles recebem várias entradas e retornam uma única saída;
- *Flip – Flops*, registradores de deslocamentos binários, que são utilizados para sincronizar lógicas e registrar estados lógicos entres os ciclos de *clock* dentro de um dispositivo FPGA;
- LUTs, *Look up Tables*, é composto por memórias SRAM, que são memorias de acesso aleatório que mantém os dados armazenados desde que seja mantida a alimentação. A função do LUT é armazenar tabelas verdades para as funções lógicas que definiram o *hardware* a ser implementado no FPGA.

Dentro do FPGA os vários blocos lógicos se conectam, sendo esta conexão volátil e reconfigurável, assim se ajustando de acordo com a sua aplicação. Com o avanço das ferramentas de síntese, as regras que regem as ligações entre os blocos lógicos configuráveis passaram a ser geradas de forma automática, simplificando a criação de novos projetos. Além dos blocos lógicos programáveis os FPGAs mais modernos estão sendo embutidos, também, com blocos de funções especificas que complementam os blocos lógicos genéricos, estes blocos

normalmente são bancos de registradores de I/O, multiplicadores, memória em *chip* e circuitos gerenciadores de *clock*.

A implementação e a reconFiguração de dispositivos FPGA é feita pelo meio da linguagem de descrição de *hardware* (HDL), do inglês *Hardware Description Language*, que são linguagens textuais utilizadas para descrever e programar o *hardware*. Existem vários tipos de linguagem HDL sendo as HDL's mais utilizadas o VHDL (*VHSIC Hardware Language*) e o *Verilog.* As linguagens HDL são semelhantes às linguagens de programação de *softwares*, porém suas funções são destinadas a descrição de estrutura e comportamento de *hardware*.

2.3 PROTOCOLO DE COMUNICAÇÃO

Protocolo de comunicação é um conjunto regras e normas que realizam o controle e possibilita uma comunicação, transferência de dados e conexão entre um ou mais sistemas computacionais, ou seja, um protocolo pode ser definido de maneira mais simplificada como todas as regras que comandam a sintaxe, semântica e a sincronização da comunicação dentro de um sistema computacional[10]. Pode-se definir sintaxe nos protocolos de comunicação como o formato dos dados e a ordem em que são apresentados. A semântica é definida como o significado que cada conjunto sintático dará sentido à mensagem. Os protocolos de comunicação são implementados no *software*, *hardware* ou por uma combinação de *software* e *hardware.*

Existe uma gama de variedades de protocolo de comunicação, sendo que, eles variam muito de propósito e sofisticação, mas a maioria dos protocolos de comunicação realizam as seguintes funções[10]:

- Detecção da conexão física;
- Endereçamento, onde é especificado o destino da mensagem;
- Handshanking, processo de estabelecimento de ligação;
- Como iniciar, formatar e finalizar uma mensagem;
- Controle de erros, onde é realizada a identificação e correção dos erros na mensagem e na comunicação;
- Como detectar e corrigir a perda inesperada da conexão;
- Realizar a retransmissão e a confirmação de recebimento da mensagem;
- Realizar a conversão do código e indicar o termino da sessão ou conexão.

2.4 SISTEMAS EM UM CHIP

SOC, que pode ser traduzido em português para sistema em um *chip*, é um termo utilizado para definir um tipo de arquitetura de computadores em que todos os componentes de um sistema eletrônico estão em um único circuito integrado (*chip*). Sendo que estes podem conter tanto sinais digitais quanto analógicos, estes sistemas são muito utilizados atualmente, sendo vistos principalmente em sistemas embarcados e em smartphones. Na Figura 2.5 tem-se um exemplo de um diagrama de blocos de uma arquitetura SOC da Xilinx, o ZYNQ-7000, que é composto por uma unidade de processamento, que contém um processador dual core ARM córtex-A9, controladores, memórias, unidade lógica programável, que contém um FPGA Artix-7 e vários outros periféricos[11].

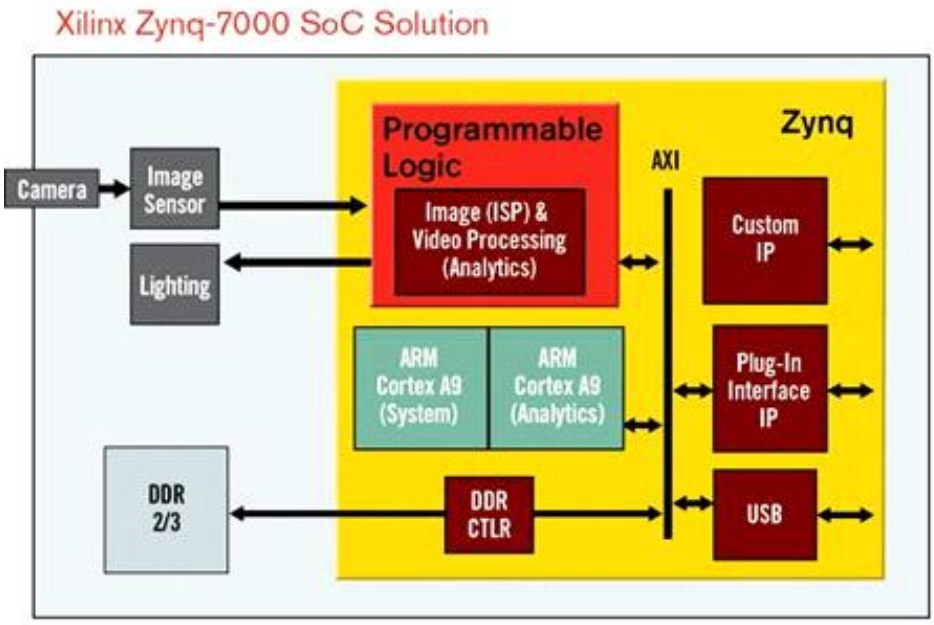

Figura 2.5: DIAGRAMA DE BLOCOS BASICO DO ZYNQ-7000.
Fonte: The ZYNQ book ebook.

Com o aperfeiçoamento dos SOCs, tornou-se possível o desenvolvimento de aplicações mais complexas, envolvendo uma vasta combinação de *hardware*s e *softwares* para solução de problemas específicos, com isso, houve o desenvolvimento de novas tecnologias e metodologias para facilitar a implementação de projetos nas plataformas SOC atuais, sendo que, umas das tecnologias mais utilizadas é a *Intellectual property* (IP) que consiste em utilizar blocos pré-projetados e pré-testados, que são obtidos de terceiros ou de fabricação própria para a combinação interna do SOC, sendo estes IP's reutilizáveis em projetos posteriores. Logo, para

realização do projeto de um SOC pode-se reutilizar diversos blocos de IP interconectados por um barramento que atenda aos requisitos do projeto. Os dispositivos SOC abordados nesse trabalho são divido em duas camadas, um Sistema de Processamento de *software*, PS, e uma estrutura de *hardware* com Lógica Programável, PL.

2.4.1 SISTEMA DE PROCESSAMENTO (PS)

O sistema de processamento em dispositivos SOC é basicamente composto por processadores que são produzidos para realizar funções gerais. Onde eles são projetados para que tenham um grande poder de processamento, consumindo a menor quantidade de energia possível e ocupando uma pequena área física. Nos SOC's Xilinx são utilizados processadores da ARM[12]. ARM, do inglês *Advanced* RISC *Machine*, é um tipo de arquitetura de processadores desenvolvida pela empresa britânica ARM *Holdings*. Que possui os direitos autorais sobre a arquitetura, e licencia o direito de produção da mesma para empresas que desejam produzir um processador com a arquitetura e o conjunto de instruções da ARM.

Os processadores ARM são do tipo RISC, normalmente de 32 bits, desenvolvidos para ter uma arquitetura simples, ocupar uma pequena área e ter um baixo consumo de energia[12]. Uma das principais vantagens dos processadores ARM é que eles possuem um conjunto de instruções muito mais complexo do que outros processadores da família RISC, graças ao seu circuito simples, um pipeline curto e um baixo consumo de energia, sempre que o processador não estiver operando, o processador consegue gera um desempenho muito satisfatório comparado a processadores da família CISC, como o Pentium por exemplo. Com isso, juntamente ao fato de ocupar uma pequena área e o baixo consumo de energia, esta família de processadores é largamente utilizada em aplicações de alta complexidade que necessitam de um baixo consumo de energia, por exemplo, em sistemas de um *chip*, como o ZYNQ-7000, da Xilinx. Uma das versões mais atuais dos processadores ARM é o processador ARM Cortex-A9, que é largamente utilizado em SOC e smartphones. Na Figura 2.6 tem-se a imagem de um processador baseado no ARM Cortex-A9, utilizado em um Smartphone da Apple.

Figura 2.6: PROCESSADOR A6 DA APPLE.
Fonte: Redmondpie.com/apple-a6-processor-goes-into-production-being-manufactured-by-tsmc-not-samsung-report/.

O ARM Cortex-A9 é um processador baseado na arquitetura ARMv7 de 32 bits, que opera em um *clock* que varia de 0.8GHZ até 2.0GHZ. Através desta frequência variada de *clock* os sistemas operacionais compatíveis com este processador podem aumentar o *clock* de acordo com que a demanda de trabalho. Assim, o processador normalmente trabalha com uma frequência de 0.8GHZ, mas caso haja uma aplicação que necessite de um maior poder de processamento a frequência aumentara para 2.0GHZ, com essa variação de frequência de operação há uma diminuição no consumo de energia, quando o sistema tiver em baixa operação[12].

Outro motivo que justificar a grande aplicação desse processador em SOC, é a utilização de coprocessadores. No córtex A9 é utilizado o coprocessador NEOM™, com o propósito de estender o conjunto de instruções da arquitetura ARMv7. Os coprocessadores consistem em pequenos processadores com propósitos específicos, com isso há uma melhora no desempenho de um grupo determinado de instruções.

O NEOM™ é um coprocessador utilizado nos processadores da série ARM Cortex-A, sendo composto por uma extensão do conjunto de instruções SIMD (Sigle Instruction Multiple Data) de 128 bits, que recebe o nome de advanced SIMD, onde as principais funções são[12]:

- Estrutura SIMD de instrução Load/Store;
- Transferências entre o processador e os registradores;
- Instruções de processamentos com os valores em ponto flutuantes de precisão simples.

Na Figura 2.7 tem-se o diagrama de blocos de um processador ARM Cortex A9.

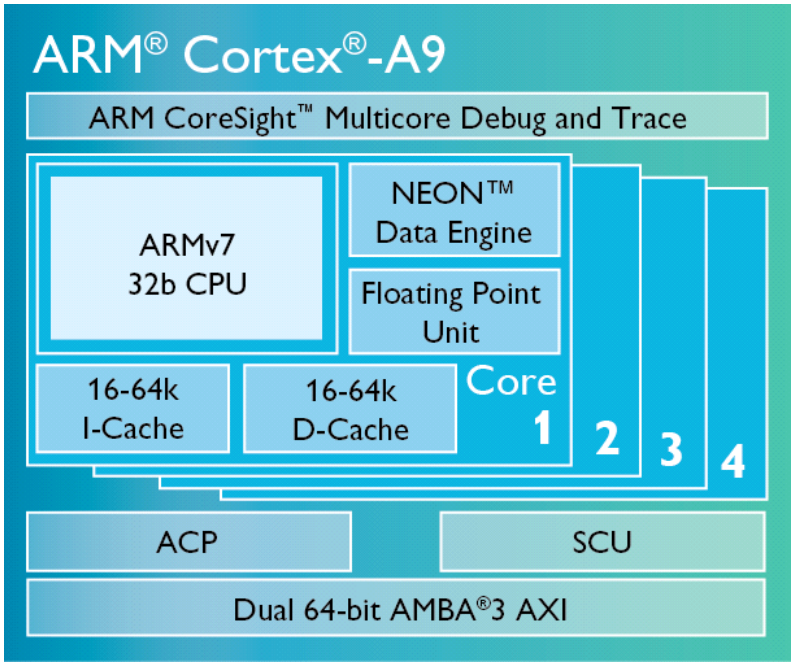

Figura 2.7: DIAGRAMA DE BLOCOS DO PROCESSADOR ARM CORTEX-A9.
Fonte: arm.com/products/processors/cortex-m/cortex-a9.php

Na Figura 2.7 são demonstrados todos os componentes por núcleo do processador córtex–A9, onde temos:

- A CPU baseada na arquitetura ARMv7;
- Os coprocessadores NEOM™;
- Uma unidade de ponto flutuante;
- Duas unidades de memória cache.

Vale ressaltar que este processador se comunica internamente pelo protocolo de comunicação AXI e que ele tem as unidades ACP e SCU para realizar a mediação e verificar os status energia gasta pelo processador.

2.4.2 LÓGICA PROGRAMÁVEL (PL)

O bloco lógico programável, em dispositivos SOC, é o *hardware* que pode ser implementado e reconFigurado para várias aplicações. Este bloco é basicamente um dispositivo FPGA, nos SOC mais recentes da Xilinx são utilizados os FPGA da série Xilinx®7, fabricados pela própria Xilinx[13].

Os FPGA da série Xilinx®7, são um conjunto de quatro famílias de FPGA, que são projetados visando um menor consumo de energia, potência gasta em watts, e um melhor desempenho por área de silício para uma mesma aplicação em relação a outros FPGA semelhantes. Uma das principais características dessa série de FPGA é que o mesmo projeto

pode ser implementado em qualquer uma dessas quatros famílias de FPGA sem alteração no projeto. As quatro famílias de FPGA pertencentes a essa série são[13]:

- O FPGA Spartan-7, que tem como ponto principal um baixo custo com um desempenho muito satisfatório em relação a outros de mesma faixa de preço, sendo este o FPGA de entrada da série Xilinx®7. Ele possui 102 mil CBL, uma memória RAM de 4.2 MB e uma interface para conexão DDR3 de até 800MB/s;
- O FPGA Artix-7, que visa um ótimo desempenho por um baixo consumo de energia em watts, em relação aos outros FPGA dessa série, e pouca variação no consumo de energia em watts em relação ao volume de operações realizadas. Ele possui 215 mil CBL, uma memória RAM de 13 MB e uma interface para conexão DDR3 de até 1.066MB/s;
- O FPGA Kintex-7, que tem como característica a melhor relação entre preço e desempenho dessa série de FPGA, onde apresenta um desempenho um pouco inferior em relação ao Virtex-7 com um custo um pouco superior ao Artix-7. Ele possui 478 mil CBL, uma memória RAM de 34 MB e uma interface para conexão DDR3 de até 1.866MB/s;
- O FPGA Virtex-7, que é o mais mordendo desta série, onde ele apresenta um ótimo desempenho e uma maior capacidade de implementação de *hardware*, em contrapartida, ele é o mais carro desta série. Ele possui 1.955 mil CBL, uma memória RAM de 68 MB e uma interface para conexão DDR3 de até 1.866MB/s.

Na Figura 2.8 tem-se um FPGA Artix-7, este FPGA é largamente utilizado nas PL dos SOC's da Xilinx, como por exemplo, o Zynq-7000.

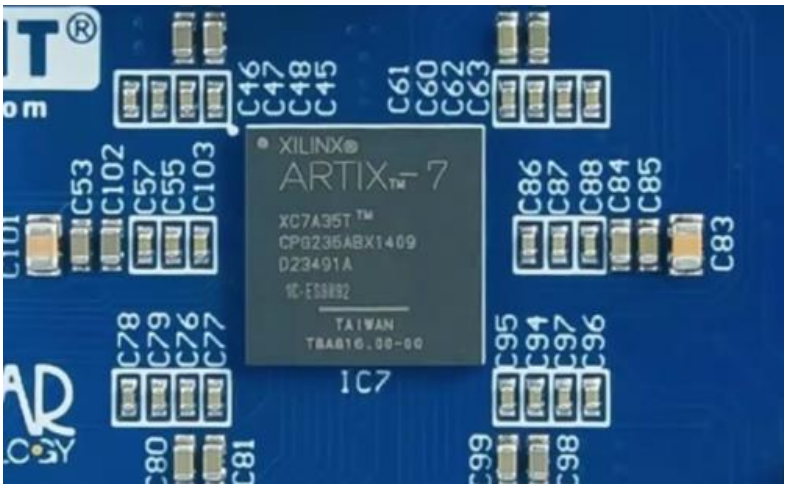

Figura 2.8: FPGA ARTIX-7..
Fonte: reference.digilentinc.com/learn/programmable-logic/tutorials/basys-3-getting-started/start

O Artix-7 é um FPGA da série Xilinx®7, sendo ele atualmente um dos FPGA mais utilizados nos SOC’s da Xilinx. Isso se deve porque ele oferece um alto poder de processamento em relação ao baixo consumo de energia, sendo que, ele pode chegar a consumir cerca de 50% menos energia ao realizar a mesma operação em comparação com o Spartan-7. Ele pode transmitir dados em até 211GB/s e possui uma interface para memória DDR3 com uma conexão de até 1.066MB/s.

Outro fator que justificar o grande uso dos FPGA Artix-7 é a sua conFiguração. Sendo as principais características dessa configuração[13]:

- 8 LUTs por CLB para implementação;
- CLB contendo 16 flip-flops;
- LUTs possuem duas conFigurações como blocos de memórias 64x1 bits ou 32x2 bits;
- Possuem registradores de deslocamento ou somadores 2x4-bits em cascata para funções aritméticas;
- Blocos de memória de 36kb de capacidade;
- Possui tecnologia SelectIO com suporte a DDR3 e interface de até 1.866 Mb/s;
- Possui conectividade serial de alta velocidade;
- Possui interface PCI Express, para até 8 vias.

3. PROTOCOLO AXI

O protocolo de comunicação AXI pertence à família de protocolos ARM AMBA, composta por um conjunto de especificações para interconexões de sistemas em um *chip*. O protocolo AXI é direcionado para projetos com alta frequência de transmissão de dados. Ele é adequado como um protocolo de comunicação de alta velocidade, pois apresenta as seguintes características[3]:

- As informações de endereço e controle são enviadas separadamente;
- Suporta transferências de dados não sequenciais usando os bits de estouro;
- Transações rápidas somente com o endereço inicial fornecido;
- Transações com dados de tamanho variado, variando de 8 bits a 1024 bits;
- Transações em rajadas de dados, com a quantidade de dados variando de 1 a 256 dados por rajada;
- Separa os canais de leitura e escrita, permitindo assim um acesso direto a memória;
- Pode emitir o endereço pendente, após o início da transmissão de dados;
- Pode concluir as transações fora de ordem e
- Pode adicionar etapas, após o início da transmissão de dados.

3.1. ARQUITETURA AXI4

O protocolo AXI, que atualmente está na sua segunda versão, que é denominada AXI4, possui transações baseadas em rajadas de dados. Pode-se definir transação como a transferência de dados de um ponto do SOC para outro ponto do mesmo, sendo que cada transação contém os seguintes canais independentes[14]:

- Endereço de leitura;
- Leitura de dados;
- Endereço de escrita;
- Escrita de dados e
- Resposta de Escrita.

Uma definição importante no protocolo AXI, é a definição de AXI *Master* e AXI *Slave*, onde AXI *Master* é o módulo dentro do SOC que inicia a transação e AXI *Slave* é o módulo dentro do SOC que é o alvo da transação, isto é, que recebe e responde a transação. Um

AXI *Master* se interliga e transfere dados para um AXI *Slave* através de um módulo de interconexão AXI. Vale ressaltar que vários AXI *Master*s podem ser conectados a vários AXI *Slave*s através da interconexão AXI. Na Figura 3.1 tem-se um exemplo de um diagrama de blocos que representa um AXI *Master* ligado a dois AXI *Slave*s através de uma interconexão AXI.

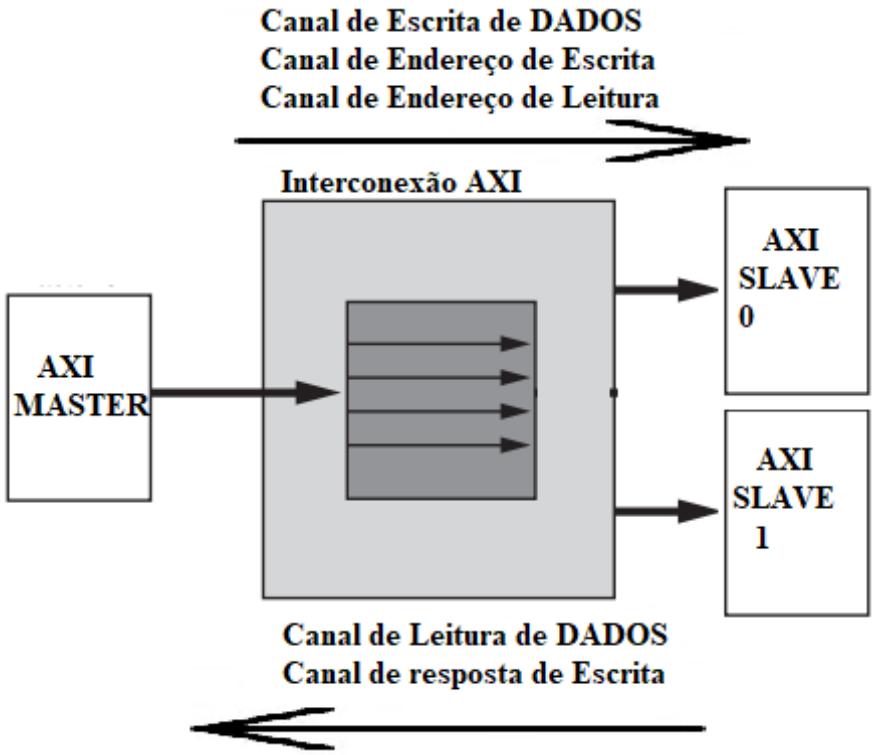

Figura 3.1: CONEXÃO ATRAVÉS DO PROTOCOLO AXI.
Fonte: Elaborado pelo Autor.

No protocolo AXI. um canal de endereço transporta as informações de controle juntamente com a descrição da natureza do dado que irá ser transmitido. O dado é transmitido entre o AXI *Master* e o AXI *Slave* das seguintes formas[14]:

- O AXI *Master* utiliza o canal de escrita de dados para transferir dados para o AXI *Slave*, sendo que em uma transação de escrita o AXI *Slave* utiliza o canal de resposta de escrita para informar a conclusão da transferência de dados para o AXI *Master*.
- O AXI *Slave* utilizar o canal de leitura de dados para transferir dados para o AXI *Master*.

É importante ressaltar que o protocolo AXI permite que as informações de endereço e controle sejam transmitidas antes da real transferência de dados, por meio do canal de

endereço, assim permitindo que as transações sejam concluídas fora da ordem que foram iniciadas.

3.1.1. CANAIS DO PROTOCOLO AXI4

Os canais do protocolo AXI são independentes entre si, consistindo em um conjunto de sinais de controle, dados e endereços. Para a transmissão de dados é realizado um mecanismo de handshake bidirecional, através dos sinais VALID e READY. Handshake pode ser definido como o processo onde duas máquinas afirmam que se reconheceram e estão prontas para iniciar comunicação entre si. Onde na primeira etapa deste Handshake é[14]:

1. A fonte da informação utiliza o sinal VALID para informa quando informações válidas de sinais de dados, controle ou endereço estão disponíveis no canal;
2. Em seguida o destino utiliza o sinal READY para informa a fonte quando ele está pronto para aceitar as informações.
3. Em caso de transmissão no canal de dados de escrita ou no canal de dados de leitura existe também o sinal LAST que indica quando ocorre a transferência do último dado na transação.

Nas Figuras 3.2 e 3.3 estão demonstrados a comunicação entre um AXI *Master* com um AXI *Slave*, sendo que a interconexão AXI está omitida da demonstração nas duas Figuras. Na Figura 3.2 tem-se uma transação de leitura entre o AXI *Master* e o AXI *Slave*, onde se utiliza o canal de endereço de leitura e o canal de leitura de dados. Na subseção 3.2 é demonstrado como ocorre a transação de dados através desses canais.

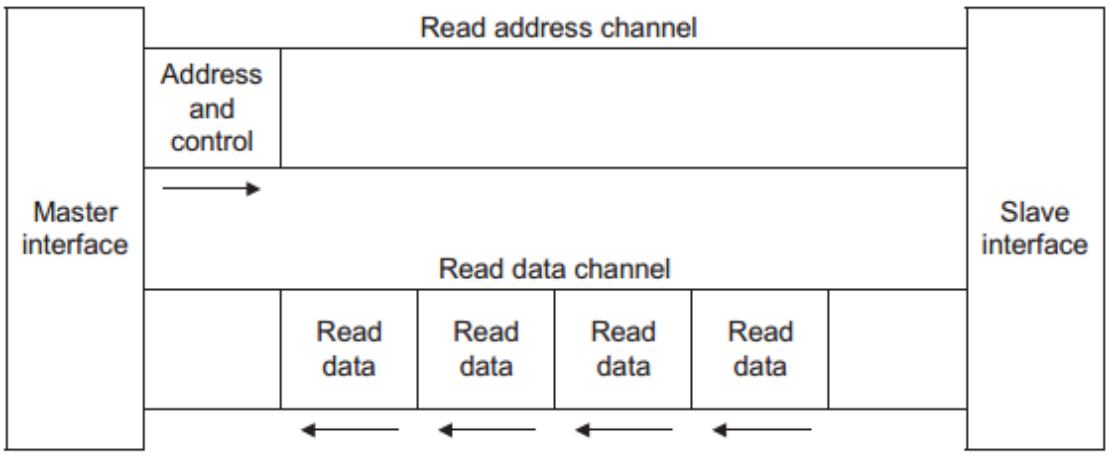

Figura 3.2: TRANSAÇÃO DE LEITURA.
Fonte: AMBA AXI Protocol Specification.

3.1.1.1 CANAL DE ENDEREÇO DE LEITURA

O canal de endereço de leitura é separado do canal de endereço de escrita, sendo que no canal de endereço de leitura são transportadas todas as informações de endereço e controle necessárias para uma transação. As informações contidas no endereço ditam o funcionamento dos seguintes mecanismos no protocolo AXI[14]:

- É informado a quantidade de dados que serão enviados na rajada de dados, onde pode variar a transferência de 1 a 256 dados por rajada.
- É informado o tamanho do dado que irá ser transferido na rajada de dados, esse tamanho pode variar de 8 bits a 1024 bits.
- O nível de cachê no sistema e o controle de buffer, onde é realizado o controle dos registradores e dos dados armazenados nele.

3.1.1.2 CANAL DE LEITURA DE DADOS

O canal de leitura de dados transporta os dados do AXI *Slave* para o AXI *Master*, esses dados podem ter tamanho de 8 a 1024 bits. Juntamente com os dados é enviado um sinal de resposta que indica o status de conclusão da transação de leitura para o AXI *Master*, informando o status no final da transmissão de cada dado na rajada de dado. Os status indicados por esse sinal são[14]:

- OKAY indica o acesso ao AXI *Slave* foi bem-sucedido;
- EXOKAY indica que o acesso exclusivo ao AXI *Slave* foi bem-sucedido;

- SLVERR indica erro ao acessar o AXI *Slave* e
- DECERR indica problema na interconexão.

Na Figura 3.3 tem-se uma transação de escrita entre o AXI *Master* e o AXI *Slave*, onde se utiliza o canal de endereço de escrita, o canal de leitura de dados e a escrita da resposta.

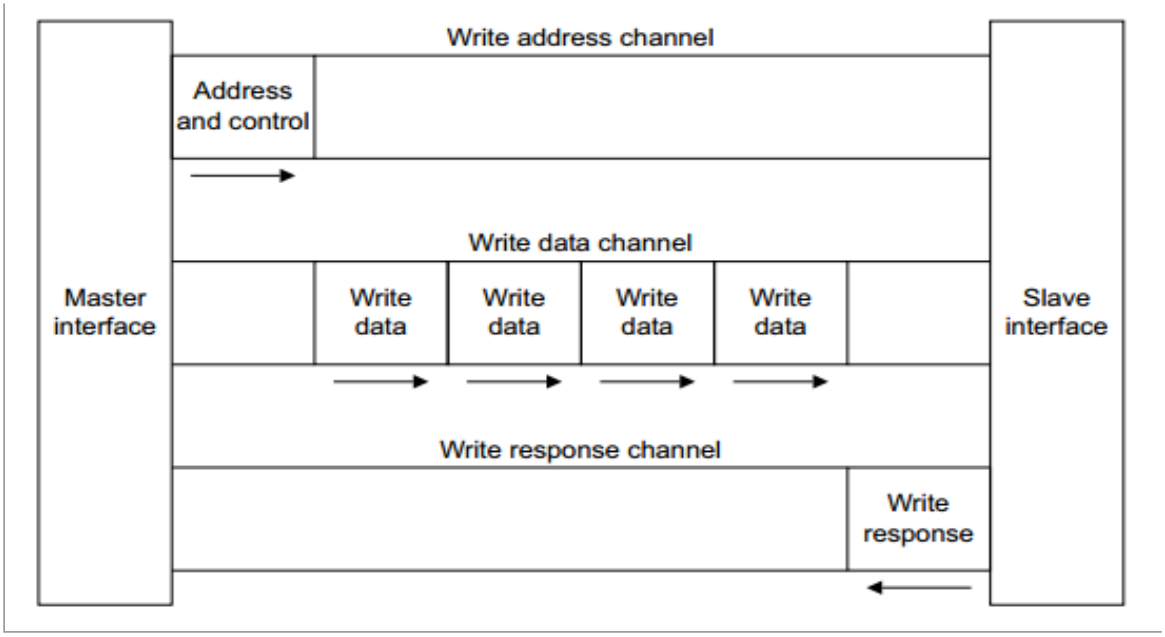

Figura 3.3: TRANSAÇÃO DE ESCRITA.
Fonte: AMBA AXI Protocol Specification.

3.1.1.3 CANAL DE ENDEREÇO DE ESCRITA

O canal de endereço de escrita funciona de forma semelhante ao canal de endereço de leitura, vale ressaltar que os dois são separados e podem funcionar paralelamente e de forma independente entre si.

3.1.1.4 CANAL DE ESCRITA DE DADOS

O canal de escrita de dados transmite os dados do AXI *Master* para serem gravados no AXI *Slave*. Os dados no canal de escrita de dados podem ter de 8 a 1024 bits de tamanho e há um byte de estouro para cada oito bits de dados, indicando quais os bytes, no canal de escrita de dados, serão válidos. As informações no canal de escrita de dados são sempre tratadas como se estivessem armazenadas em buffer, assim o AXI *Master* pode executar todas as transações de escrita sem que o AXI *Slave* tome conhecimento das transações gravadas anteriormente[14].

3.1.1.5 CANAL DE RESPOSTA DE ESCRITA

O canal de resposta de escrita fornece uma maneira para o AXI *Slave* responder as transações de escrita do AXI *Master*, permitindo assim o AXI *Slave* informar o status de conclusão da transação para o AXI *Master*. O canal de resposta só irá informa ao AXI *Master* no final da rajada de dados, e não ao final do envio de cada dado dentro da rajada. Logo ele só irá fornecer uma resposta ao final da transação de escrita, indicado o recebimento ou se ocorreu algum erro na transmissão de dados. Os status indicados por esse sinal são[14]:

- OKAY indica o acesso ao AXI *Slave* foi bem-sucedido;
- EXOKAY indica que o acesso exclusivo ao AXI *Slave* foi bem-sucedido;
- SLVERR indica erro ao acessar o AXI *Slave* e
- DECERR indica problema na interconexão.

3.1.2 INTERFACE E INTERCONEXÃO AXI

Em um sistema típico há um número variado de módulos AXI *Master* ligados a um número variado de módulos de AXI *Slave*s por meio de uma interface, juntamente com uma interconexão AXI como é demonstrado na Figura 3.4.

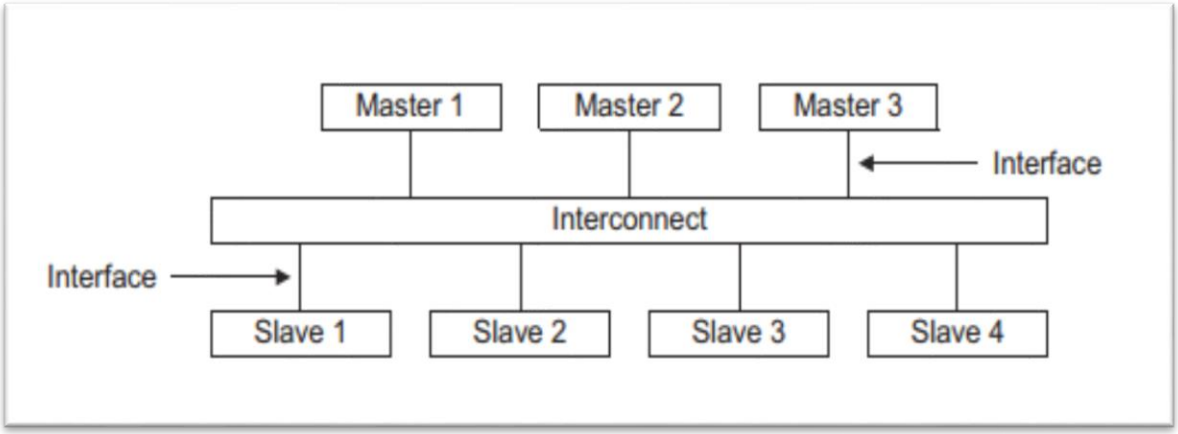

Figura 3.4: **CONEXÃO ATRAVÉS DE UMA INTERCONEXÃO AXI.**
Fonte: AMBA AXI Protocol Specification.

Podemos definir interface como um dispositivo que proporciona uma ligação física ou lógica entre dois sistemas ou parte de um sistema que não podem ser conectados diretamente. No protocolo AXI podemos ter três tipos de interfaces[14]:

- Entre um AXI *Master* e um AXI *Slave*;
- Entre um AXI *Master* e uma Interconexão e

- Entre um AXI *Slave* e uma interconexão.

Com esses três tipos de interfaces pode-se ter uma gama de diferentes tipos de implementações da interconexão AXI. Na Figura 3.5 tem-se um diagrama de bloco detalhado da interconexão AXI, onde podemos constatar que a interconexão é dividida em dois hemisférios, o hemisfério Sl, que está ligado ao AXI *Master* e o hemisfério Ml que está ligado ao AXI *Slave*, essa divisão em hemisférios permite que o AXI *Master* "veja" a interconexão como um AXI *Slave* enquanto o AXI *Slave* vê a mesma interconexão como um AXI *Master*, os hemisférios são simétricos. Quando ocorre a transação, se tomamos como referencial ela ocorrendo do AXI *Master* para o AXI *Slave*[14]:

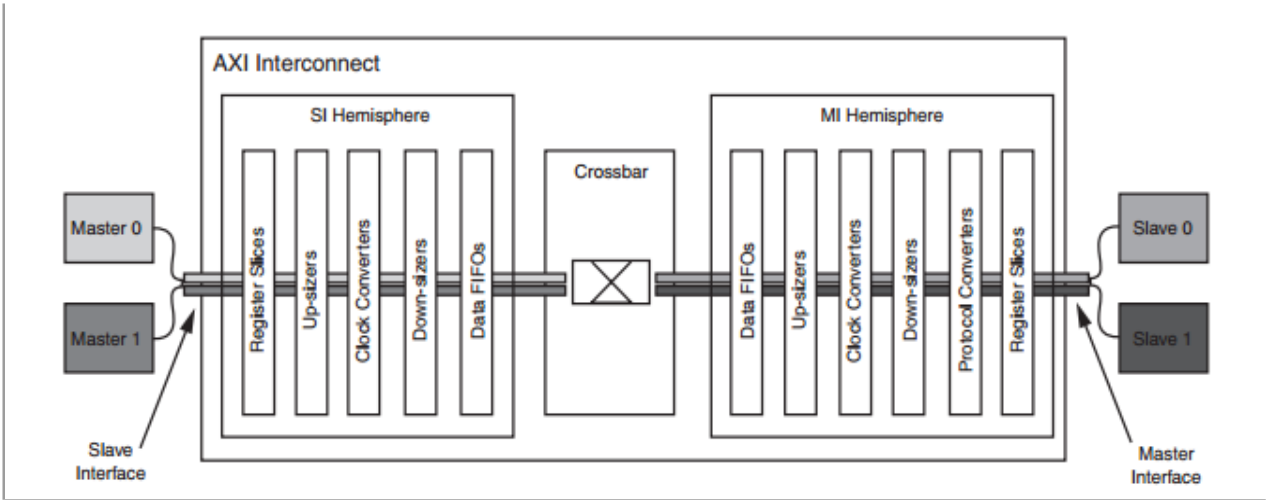

Figura 3.5: DIAGRAMA DE BLOCOS DETALHADO DE UMA INTERCONEXÃO AXI.
Fonte: AMBA AXI Protocol Specification.

- Primeiramente o dado passa por um registrador que serve como buffer para tornar as transações mais robustas.
- Em seguida ela passa por um módulo *up-sizers*, que tem como função, realizar o empacotamento de dados, caso necessário, reduzindo o número de bits de estouro, fazendo assim que a transação tenha um número menor de dados, os *up-sizers* têm funções diferentes para leitura, onde ocorre o processo de serialização dos dados e na escrita, ocorre a compactação dos dados.
- A transação passa em seguida por um conversor de *clock* que tem como objetivo converter a frequência de operação, nos casos em que elas apresentem valores de operação diferentes.
- O penúltimo estagio é o *Down-sizers*, que tem como função, em caso de a capacidade de recebimento de dados na transação no lado de um hemisfério seja

maior que no outro lado, ocorre um aumento no número de bits de estouro aumentando assim o número de rajadas de dados, na escrita ocorre a serialização dos dados já na leitura ocorre a compactação dos dados.

- A transação então passa por um módulo data FIFO (*first in first out*) que tem como objetivo evitar a perda de dados, funcionando como uma “fila”.

Saindo do hemisfério SI a transação passa por uma interface que conecta todos os AXI *Master* ligados a essa interconexão com todos os AXI *Slave*s ligada a mesma, para então entrar no hemisfério MI, onde a transação realiza as operações de forma inversa às realizadas no outro hemisfério.

Como pode ser visto na Figura 3.5, uma interconexão AXI permite que vários AXI *Master*s se interconectem com vários AXI *Slave*s, sendo que uma interligação entre vários AXI *Master* e vários AXI *Slave* pode ser descrita como um módulo equivalente, que possui várias portas *Master*s e *Slave*s simétricas, assim esses módulos podem ser ligados com outros AXI *Master*, AXI *Slave* ou outras interconexões AXI, respeitando uma hierarquia pré-definida no projeto. Na Figura 3.6 tem-se um exemplo destas interconexões.

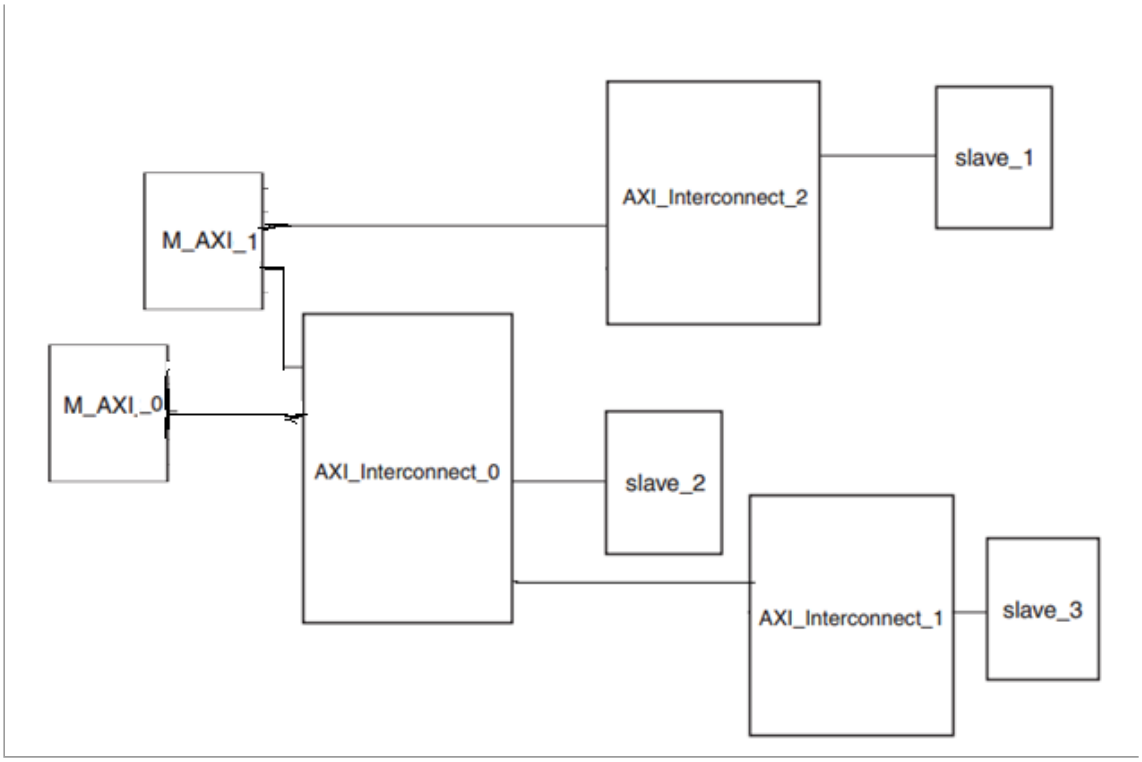

Figura 3.6: DIAGRAMA DE BLOCOS DE UMA LIGAÇÃO COM VARIOS AXI *MASTER* LIGADOS A VARIOS AXI *SLAVE*.
Fonte: Elaborado pelo Autor.

Na Figura 3.6 tem-se uma interligação com três interconexões AXI, três AXI *Slave*s e dois AXI *Master*s. O AXI *Master* M_AXI_1 está conectado a todos os AXI *Slave*s por meio das interconexões AXI, já o AXI *Master* M_AXI_0 não está conectado a interconexão AXI_Interconnect_2 e logo não está conectado ao AXI *Slave* 1.

3.1.3 REGISTRADORES *SLICES*

Os registrados *Slices* são importantes buffers que tornam as transações no protocolo AXI mais robustas. São registradores que podem ser colocados em qualquer canal ao custo somente da adição de um ciclo de *clock* de latência em relação a transação de dados. Isso se deve porque os canais do protocolo AXI transferem informações somente em uma direção e não há correlação entre eles. Também é possível colocar os registrados *Slices* em qualquer ponto dentro de uma interconexão[14].

3.2 TRANSAÇÕES NO PROTOCOLO AXI4

Nesta subseção iremos demonstra por meio de exemplos o funcionamento das transações no protocolo AXI.

3.2.1 TRANSAÇÃO DE ESCRITA

Na Figura 3.7 tem-se um exemplo de uma transação de escrita. A transação é iniciada quando o AXI *Master*, através do sinal AWVALID, indica que está pronto para enviar as informações de endereço e controle para o AXI *Slave*, através do canal de endereço de escrita. O AXI *Slave* responde com o sinal AWREADY para indicar que ele está pronto para receber as informações de endereço e controle. Quando os dois sinais estão em faixa alta, o AXI *Master* envia através do sinal AWADDR as informações de endereço, juntamente com as informações de endereço são enviadas informações de controle através de outros sinais.

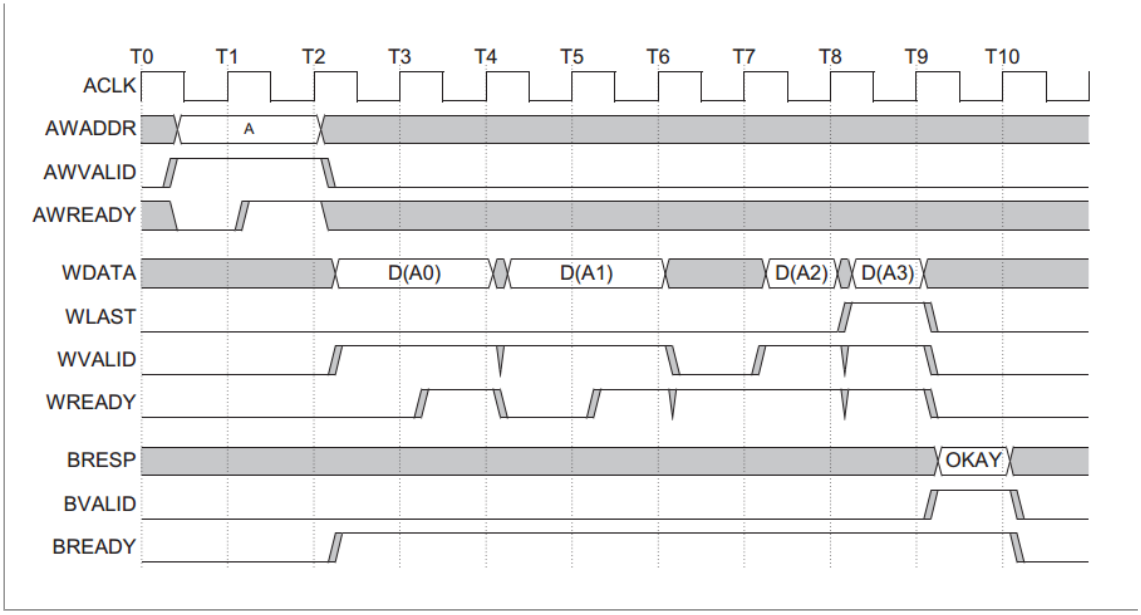

Figura 3.7: TRANSAÇÃO DE ESCRITA.
Fonte: AMBA AXI Protocol Specification.

Após o envio das informações de endereço e controle pelo canal de endereço de escrita, o AXI *Master* sinaliza que está pronto para enviar dados pelo canal de escrita de dados, através do sinal WVALID em faixa alta. O AXI *Slave* sinaliza que está pronto para receber a transação de escrita em rajada de dados, através do sinal WREADY em faixa alta. Quando ambos os sinais estão em faixa alta, o AXI *Master* envia os dados através do sinal WDATA. O AXI *Master* sinaliza para o AXI *Slave* com o sinal WLAST informando o envio do último dado da transação.

Após o envio dos dados através do canal de escrita de dados. O AXI *Slave* sinaliza que está pronto para enviar uma resposta a transação de dados pelo do canal de resposta de escrita, através do sinal BVALID em faixa alta. O AXI *Master* sinaliza que está pronto para receber está resposta, através do sinal BREADY em faixa alta. Com ambos os sinais em faixa alta o AXI *Slave* envia uma resposta, através do sinal BRESP indicando o status da conclusão da transação de escrita.

Dentre os demais sinais envolvidos na transação de escrita, podemos destacar[14]:

1. No canal de endereço de escrita:
    - O sinal AWID que informa o tag de identificação do AXI *Master*.
    - O sinal AWLEN que informa o número de dados na transação de escrita, pode variar de 1 dado até 256 dados.
    - O sinal AWSIZE que informa o tamanho do dado na transação de escrita, pode variar de 8 bits até 1024 bits.
2. No canal de escrita de dados:
    - O sinal RID que informa o tag de identificação do AXI *Master*.

- O sinal WSTRB auxiliar o preenchimento de dados na memória. Onde a cada 8 bits enviados através do canal de escrita de dados é enviado um bit de estouro.

3. No canal de resposta de escrita:
    - O sinal BID que informa o tag de identificação do AXI *Master*.

3.2.2 TRANSAÇÃO DE LEITURA

Na Figura 3.8 tem-se um exemplo de uma transação de leitura. A transação é iniciada quando o AXI *Master*, através do sinal ARVALID em faixa alta, indica que está pronto para enviar as informações de endereço e controle para o AXI *Slave*, através do canal de endereço de leitura. O AXI *Slave* responde com o sinal ARREADY em faixa alta, para indicar que ele está pronto para receber as informações de endereço e controle. Quando os dois sinais estão em faixa alta, o AXI *Master* envia através do sinal ARADDR as informações de endereço, juntamente com as informações de endereço são enviadas informações de controle através de outros sinais.

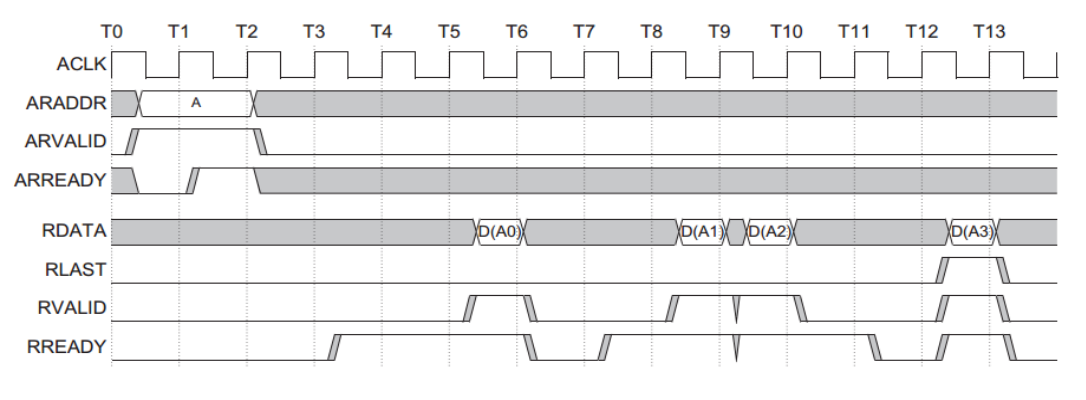

Figura 3.8: TRANSAÇÂO DE LEITURA.
Fonte: AMBA AXI Protocol Specification.

Após o envio das informações de endereço e controle pelo canal de endereço de leitura. O AXI *Slave* sinaliza que está pronto para enviar dados pelo canal de leitura de dados, através do sinal RVALID em faixa alta. O AXI *Master* sinaliza que está pronto para receber a transação de leitura em rajada de dados, através do sinal RREADY em faixa alta. Quando ambos os sinais estão em faixa alta, o AXI *Slave* envia os dados através do sinal RDATA. O AXI *Slave* sinaliza para o AXI *Master* com o sinal RLAST, informando o envio do último dado da transação.

Dentre os demais sinais envolvidos na transação de leitura, podemos destacar[14]:

1. No canal de endereço de leitura:

- O sinal ARID que informa o tag de identificação do AXI *Master*.
- O sinal ARLEN que informa o número de dados na transação de leitura, pode variar de 1 dado até 256 dados.
- O sinal ARSIZE que informa o tamanho do dado na transação de leitura, ele pode variar de 8 bits até 1024 bits.

2. No canal de leitura de dados:
   - O sinal RID que informa o tag de identificação do AXI *Master*.
   - O sinal RRESP que informa para o AXI *Master* o status da conclusão da transação de dados.

3.2.3 ORDEM DA TRANSAÇÃO

O protocolo de comunicação AXI permite que a transação seja concluída fora da ordem. Isso ocorre por que ele fornece uma tag de identificação para cada transação na interface. O protocolo AXI somente exige que as transações marcadas com a mesma tag de identificação sejam concluídas em ordem, enquanto as que estão marcadas com tags diferentes podem ser concluídas fora da mesma. Utilizar transações fora de ordem melhoram o desempenho do sistema em duas maneiras[14]:

- Com o auxílio da interconexão AXI, os AXI *Slave*s com resposta mais rápida podem ter suas transações concluídas antes das transações solicitadas anteriormente por AXI *Slave*s com resposta mais lenta.
- Os AXI *Slave*s podem retorna os dados lidos fora da ordem, como por exemplo, um item de dados resultante de uma transação posterior pode estar disponível a partir de um registrador interno, antes que os dados de endereço estejam disponíveis.

É importante ressaltar que se um AXI *Master* requer que todas as suas transações sejam concluídas na mesma ordem que elas foram emitidas, então todas as transações devem ter a mesma tag de identificação. Em um sistema com mais de um AXI *Master*, a interconexão AXI fica responsável por separar as tags de identificação de cada AXI *Master*, a fim de garantir que cada AXI *Master* tenha sua tag de informação exclusiva, a tag de identificação é semelhante ao número de identificação do AXI *Master*.

3.3 VARIAÇÕES DO PROTOCOLO AXI4

Podemos dividir o protocolo AXI4 em duas categorias, o protocolo AXI4 de Memória Mapeada e o protocolo AXI4-*Stream*.

3.3.1 PROTOCOLO AXI4 DE MEMÓRIA MAPEADA

O protocolo de comunicação AXI4 de Memória Mapeada fornece conexões separadas entre o canal de dados e o de endereço, tanto para operação de leitura quanto de escrita de dados, ou seja, toda vez que ocorrer uma transação na interface, será passado um endereço onde o dado será lindo ou escrito, juntamente com suas informações de controle, independente se a transação é de leitura ou escrita de dados. Isso permite que a transferência de dados seja bidirecional e simultânea.

No protocolo de Memória Mapeada em uma única transação é transmitido o endereço, o controle e até 256 dados de informação, onde cada dado tem tamanho variado de 8 a 1024 bits. Este protocolo é compatível com uma gama de opções que visam obter um melhor rendimento na transmissão de dado, entre elas podemos citar os *UP-sizers*, os *Down-sizers* e o processamento da transação fora da ordem.

Em nível de *hardware* é importante ressaltar que o protocolo de comunicação AXI4 de Memória Mapeada permite que haja uma interconexão entre AXI *Master* e AXI *Slave*s com frequência de *clock* diferentes, além disso, há também registradores *Slices* que auxiliam no sincronismo da transação.

3.3.1.1 PROTOCOLO AXI4 *LITE*

O protocolo AXI4 *Lite* é uma variação mais leve do protocolo AXI4 de Memória Mapeada, usada para transmissões de dados mais simples. Na AXI4 *Lite* a transação ocorre com a transmissão de um único dado, seu tamanho varia de 32 ou 64 bits, diferente da AXI4 onde eram rajadas de até 256 dados, juntamente com o endereço e o controle. Como na transação é enviado um único dado, o protocolo AXI4 *Lite* despensa o uso de bits de estouro. Um dos principais benefícios de utilizar o protocolo AXI4 *Lite* é que ele permite uma lógica menor com uma interface mais simples[14].

3.3.2 PROTOCOLO AXI4-*STREAM*

O protocolo AXI4-*Stream* não possui nenhuma fase de endereçamento de dados, isso significa que quando ocorrer a transação o dado irá fluir diretamente do AXI *Master* para o AXI *Slave*, de forma contínua sem precisar de sinais de endereçamento e com uma quantidade

ilimitados de dados nas transações. Na Figura 3.9, onde é apresentado o diagrama de blocos do canal de transferência de dados na ligação entre um AXI *Master* com um AXI *Slave*.

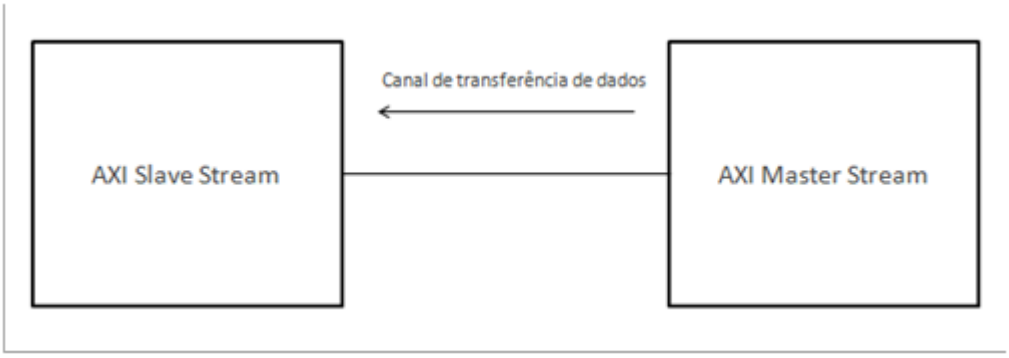

Figura 3.9: CONEXÃO ATRAVÉS DO PROTOCOLO AXI *STREAM*.
Fonte: Elaborado pelo Autor.

O protocolo AXI-*Stream* realiza a transmissão de dados através de um handshake, onde o AXI *Slave* informa para o AXI *Master* que está pronto para receber os dados pelo sinal TREADY e o AXI *Master* informa para o AXI *Slave* que irá começar a enviar os dados através do sinal TVALID[15]. Na Figura 3.10 tem-se um diagrama de sinais no momento de transmissão de dados.

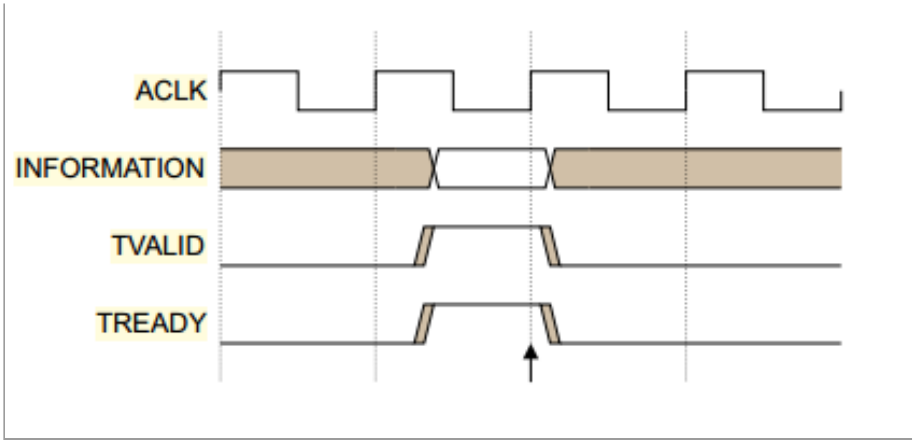

Figura 3.10: TRANSAÇÃO NO PROTOCOLO AXI *STREAM*.
Fonte: AMBA 4AXI4-*Stream* Protocol Specification.

Os dados só serão enviados do AXI *Master* para o AXI *Slave* se ambos os sinais TVALID e TREADY estiverem em faixa alta. É importante ressaltar que os dados só são enviados de AXI *Master* para AXI *Slave*, logo se forem necessárias transferências bidirecionais ambos os periféricos tem que ser AXI *Master* e AXI *Slave*[15].

3.3.3 PRINCIPAIS DIFERENCIAS ENTRE OS PROTOCOLOS AXI4-*STREAM* E O AXI4 DE MEMÓRIA MAPEADA

Algumas das diferenças fundamentais entre o funcionamento da interface AXI4-*Stream* e da interface AXI 4 de Memória Mapeada são:

- No AXI4-*Stream* não são passadas informações de controle e endereço;
- A rajada de dados na interface AXI4-*Stream* tem quantidade ilimitada de dados;
- A interface AXI4–*Stream* não permite a transferência de dados fora de ordem e
- A interface AXI4–*Stream* permite que o tamanho do dado na rajada de dados posa ser qualquer número inteiro de bytes.

3.4 INTERCONEXÕES COM OS PROTOCOLOS AXI4-*STREAM* E O AXI4 DE MEMÓRIA MAPEADA

Em muitos casos tem-se que utilizar sistemas que consistem na combinação de módulos de interface AXI4-*Stream* com módulos de AXI4 de Memória Mapeada. Esses sistemas funcionam em conjunto graças aos módulos AXI DMA, do inglês *Data Memory Access*. Os módulos AXI DMA servem para transformar dados vindos da interface AXI4 de Memória Mapeada para dados que podem ser transmitidos em interface AXI4- *Stream*, e vice-versa. Na Figura 3.11 tem-se o diagrama de bloco de um módulo DMA utilizado para transforma os dados vindo de uma interface AXI4 – *Stream* para dados que possam ser transportados por uma interface AXI4 de Memória Mapeada.

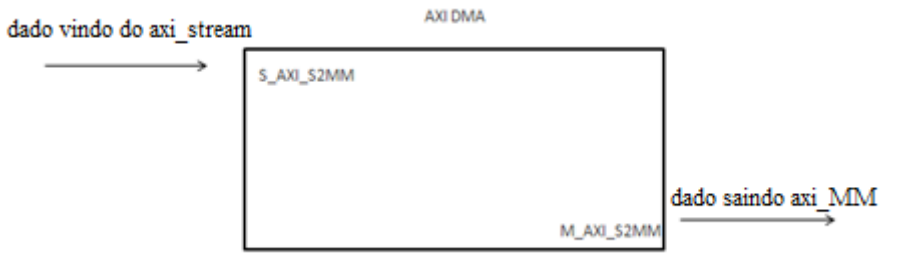

Figura 3.11: MÓDULO AXI DMA.
Fonte: Elaborado pelo Autor.

Na transformação de dados vindos do protocolo AXI4-*Stream* para dados do protocolo AXI4 de Memória Mapeada, o protocolo AXI4-*Stream* trata a entrada do AXI DMA como uma porta AXI *Slave*, através do competente S_AXI_S2MM (*Slave* AXI-*Stream* to *Memory Map*), enquanto no protocolo AXI4 de Memória Mapeada trata a saída do mesmo

como um AXI *Master*, através do componente M_AXI_S2MM (*Master* AXI *Stream* to *Memory Map*). Já na transformação de dados vindos do protocolo AXI4 de Memória Mapeada para dados de protocolo AXI4-*Stream*, o AXI DMA será tratado como um AXI *Master* pelo protocolo AXI4-*Stream* e como um AXI *Slave* pelo protocolo AXI4 de Memória Mapeada.

Na Figura 3.12 tem-se um exemplo de um diagrama de blocos de uma aplicação básica que utiliza o conjunto das interfaces AXI4-*Stream* e a interface AXI4 de Memória Mapeada para realizar processamento de imagem.

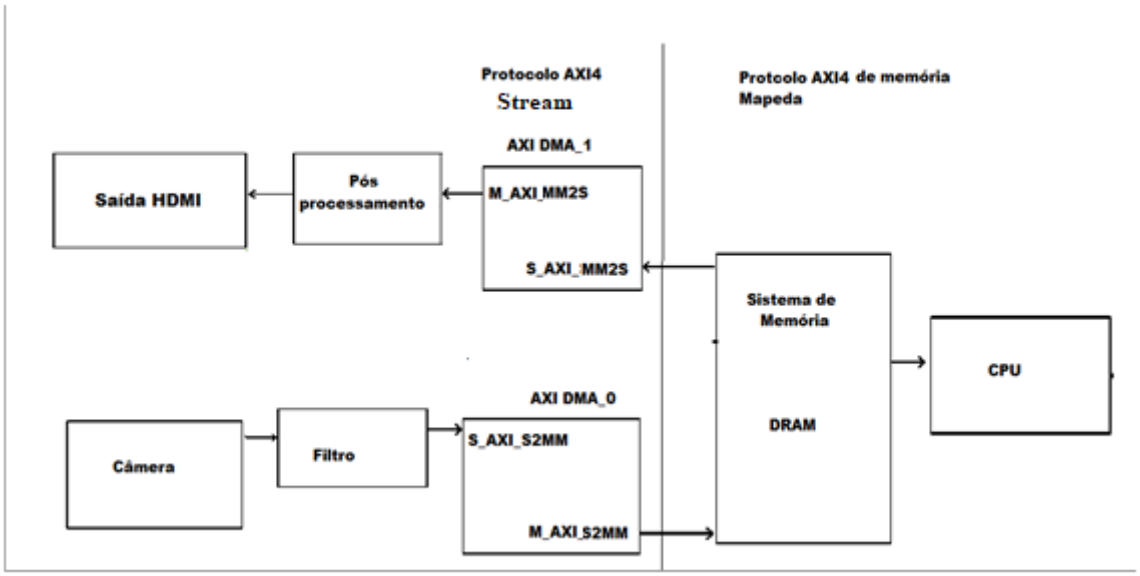

Figura 3.12: EXEMPLO DO USO DO MÓDULO AXI DMA.
Fonte: Elaborado pelo Autor.

No diagrama de blocos da Figura 3.12, observa-se que o dado é transmitido da câmera até o filtro por meio de protocolo de comunicação AXI *Stream*, do filtro o dado é transmitido para o AXI DMA, o AXI DMA_0 funciona como um AXI *Slave* para a interface AXI4-*Stream* e como AXI *Master* para a interface AXI4 de Memória Mapeada, assim o dado será escrito pelo AXI DMA_0 na memória DRAM onde ele pode ser acessado pelo CPU. O dado em seguida irá seguir para o AXI DMA_1, onde ele funcionará com um AXI *Slave* para o protocolo AXI4 de Memória Mapeada, e irá fazer as leitura dos dados na memória DRAM. Estes dados serão transformados em dados que podem ser transportados pelo protocolo AXI4-*Stream*, o AXI DMA_1 funciona como um AXI *Master* para o protocolo de comunicação AXI4-*Stream*. Em seguida os dados passarão pelo módulo de pós processamento até a saída HDMI.

4. EXPERIMENTOS E RESULTADOS

Nesta seção são demonstrados o *software* e o *hardware*, utilizados nos dois experimentos do protocolo de comunicação AXI4, com o intuito de explicar o funcionamento do protocolo de comunicação AXI4 de uma forma prática e simplificada. Os experimentos são feitos no ambiente de desenvolvimento Vivado, utilizando o kit de desenvolvimento Zedboard, que é composto por um SOC Zynq-7000.

No primeiro experimento é demonstrado o funcionamento do protocolo de comunicação AXI4 de memória mapeada. Este experimento é dividido em quatro etapas, onde primeiro foi criado um módulo de IP contendo um multiplicador. Este módulo funciona como um AXI *SLAVE LITE*. Em seguida é criado um *hardware*, implementado no SOC. Para então ser criado e implementado o *software* que realiza o controle, envio e recebimento dos dados da aplicação através da PS. E por último são demonstrados os resultados.

No segundo experimento é demonstrado o funcionamento do AXI DMA. O termo DMA é a abreviação do inglês, *Direct memmory access.* O DMA permite que dispositivos de *hardware* em um sistema computacional acessem a memória do sistema para leitura ou escrita independentemente do CPU. No protocolo AXI, o AXI DMA consiste em um módulo capaz de interconectar em um mesmo sistema as duas variações do protocolo de comunicação AXI4. Este experimento é dividido em três etapas, onde primeiro é criado um *hardware*, que contem uma comunicação entre o protocolo AXI4 de Memória Mapeada e o protocolo AXI4 *Stream*. Então é implementado um *software* que realiza o controle, envio e recebimento dos dados da aplicação, através da PS. E por último são demonstrados os resultados.

4.1 *SOFTWARE* E *HARDWARE* UTILIZADOS

Nesta subseção é apresentado o *software* e o *hardware* utilizado para a elaboração do estudo prático do protocolo de comunicação AXI. Onde o *software* utilizado neste trabalho é o ambiente de desenvolvimento Vivado e o *hardware* é o kit de desenvolvimento Zedboard, que contém como seu núcleo o SOC ZYNQ-7000 da Xilinx.

4.1.1 *SOFTWARE* PARA EDIÇÃO DE *HARDWARE* E *SOFTWARE* VIVADO/SDK

O ambiente de desenvolvimento Vivado *design suite* consiste em um conjunto de *softwares* criado pela Xilinx, para desenvolvimento de aplicações em FPGA e SOC. Este ambiente de desenvolvimento nos possibilita diversas aplicações, onde pode-se criar, analisar e realizar a síntese de códigos em VHDL ou *Verilog*, criação de projetos através de blocos pré desenvolvidos, IP's, criação de projetos em linguagem C e também a conversão de códigos em linguagem C para a lógica programável. O ambiente de desenvolvimento Vivado design suite é composto pelos seguintes *softwares*[16]:

- Vivado HLS(*High-Level Synthesis*), é um *software* que funciona basicamente como um compilador de alto nível que trabalha com as linguagens C, C++ e systemC. Este *software* oferece a possibilidade de criar aplicações diretamente para o SOC, sem a necessidade da criação prévia de um *hardware* para o mesmo, visando assim acelera a produtividade de novos projetos. Outra utilidade desde *software* é a conversão de códigos criados no OpenCL (*Open Computing Language*) para blocos de IP que podem ser utilizados nos outros *softwares* do ambiente de desenvolvimento Vivado.
- Vivado é um *software* capaz de realizar inúmeras aplicações. Basicamente ele é o responsável pela criação do *hardware* do SOC, ou do FPGA, sendo que para isso ele conta com vários sistemas, sendo um deles o sistema de criação de novos scripts em VHDL ou Verilog, o TLC(*Tool Command Language*) Store, que permite a criação de novas funções, IP's, ou complementar as funções previamente desenvolvidas para o Vivado. Outro sistema importante do *software* vivado é o IP integrator. Ele é o responsável por integrar todos os diferentes blocos de IP, tanto os criados pelo usuário, quanto aqueles fornecidos pela Xilinx. O *software* Vivado conta com um potente compilador, que suporta os scripts criados em TLC, os blocos de IP's e uma mistura de linguagem de programação em *hardware*, como o VHDL e o Verilog por exemplo.
- SDK (*Software Development Kit*) - o *software* SDK é basicamente o responsável em realizar a conFiguração da PS do SOC, onde nele criamos um programa em C ou em C++, que será responsável por realizar o controle do *hardware* criado no Vivado.

A implementação de um projeto em um SOC na suíte vivado é dividida em duas partes. Na primeira parte tem-se que criar o *hardware* no Vivado, para depois exporta-lo para o SDK onde é criado o *software* e em seguida implementamos ambos no SOC. Na Figura 4.1 tem-se uma ilustração de um fluxo de projeto para a implementação de um SOC no ambiente de desenvolvimento VIVADO.

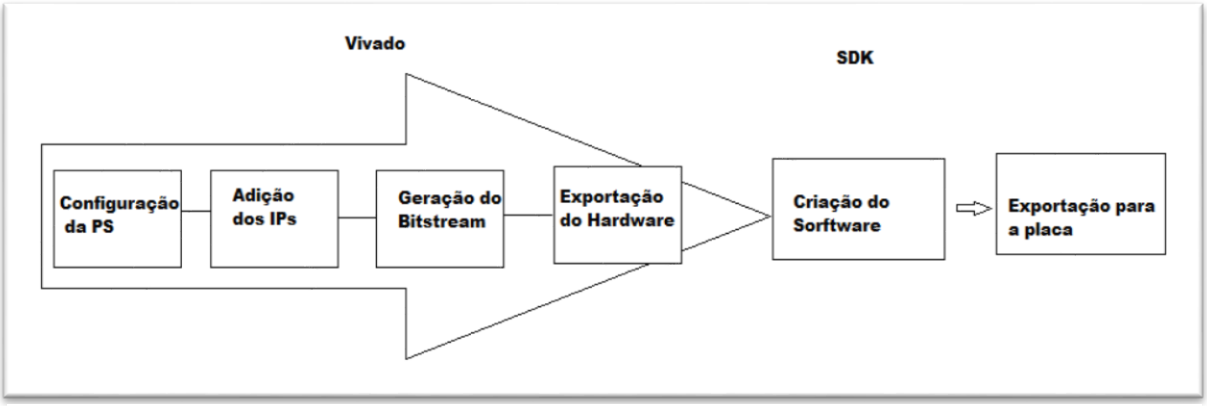

Figura 4.1: FLUXO DE PROJETO DE UM SOC NO AMBIENTE VIVADO.
Fonte: Elaborado pelo autor.

Como pode ser visto através da Figura 4.1, inicialmente tem-se que configurar a PS no vivado, selecionado as opções de memória, a frequência do *clock* que o processador irá operar, os periféricos que estão disponíveis em *hardware* que serão utilizados, as interfaces de comunicação com o barramento AXI e as interrupções. Em seguida tem-se que incluir os IPs que são utilizados no projeto. Vale ressaltar que o vivado possui uma gama de opções de IPs disponíveis em catálogo para a utilização. Assim, pode-se adicionar uma IP disponível do próprio Vivado ou criar uma IP no Vivado especialmente para ser utilizado no projeto. Em seguida valida-se o projeto e gera-se o código VHDL, para então produzir o seu bit*stream* e exporta o mesmo para o SDK.

No SDK tem-se todas as especificações do projeto exportado, juntamente com os periféricos disponíveis e seus endereços, sendo que, com base no projeto do *hardware* que foi desenvolvido. É criado um programa em C para gerar o *software* de controle para a aplicação. O arquivo gerado terá a extensão .elf para ser implementado no SOC em conjunto com o arquivo gerado no Vivado que tem a extensão .bit, referente ao *hardware*(PL).

4.1.2 ZYNQ 7000

O ZYNQ 7000 é um dispositivo fabricado pela Xilinx, feito em arquitetura SOC[17]. Nele tem-se basicamente duas unidades, a PL, unidade Lógica Programável, composta por um FPGA artix-7 e a PS, sistema de processamento, que é a parte do SOC responsável pelo processamento de dados, onde tem-se um processador dual core ARM córtex-A9 juntamente com outros componentes, sendo que a conexão interna dos blocos e entre esses dois blocos é feita por meio do protocolo de comunicação AXI4. Na Figura 4.2 tem-se o digrama de blocos detalhado de um SOC ZYNQ 7000.

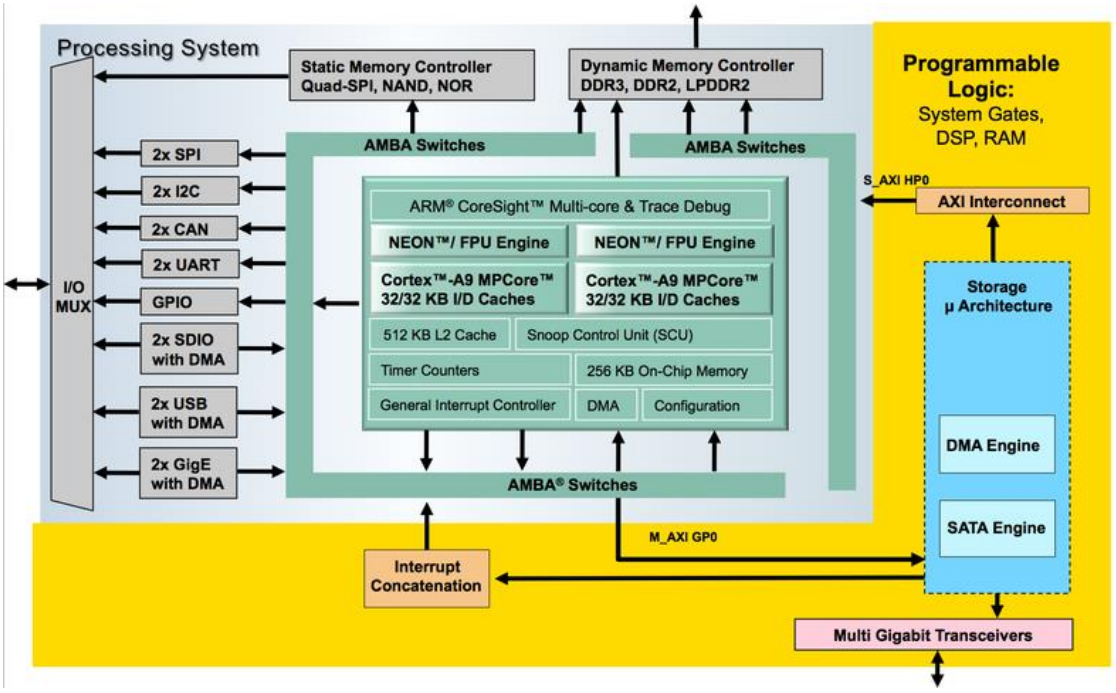

Figura 4.2: DIAGRAMA DE BLOCOS DO ZYNQ-7000.
Fonte: The ZYNQ book ebook.

Na Figura 4.2 pode-se observar os principais blocos que compõem um SOC ZYNQ 7000, onde tem-se os seus componentes detalhados listado a seguir[17]:

VISÃO GERAL DO ZYNQ 7000

1. Sistema de processamento

   Processador ARM Cortex-A9

   - 2.5 DMIPS/MHz por núcleo

- *Clock* de até 2GHz em operação

Cache L1

- Instrução de 32KB e cachê de dados de 32KB.
- Comprimento de linha do cachê 32 bytes.
- Suporta paridade.

Cache L2

- 512KB de cachê do controlador de alto desempenho.
- Suporte de paridade
- Interface AXI mestre para o controlador DDR
- Interface AXI mestre para todos os dispositivos da PL e da PS.
- Interface AXI escravo para a unidade de controle Snoop(SCU).

Unidade de controle Snoop (SCU).

A SCU conecta os dois processadores córtex-A9, a memória e os periféricos do sistema via a interface AXI.

- Fornece a prioridade de acesso do cachê L1 aos processadores e a porta de coerência aceleradora (ACP).
- Fornece a prioridade de acesso entre os processadores córtex-A9 e o cachê L2
- Fornece acesso a memória ROM e a memória RAM.
- Gerencia a ACP.

Acelerando do *software* com *Hardware*

A porta aceleradora de coerência tem as seguintes características:

- É uma porta de 64 bits.
- Ligada a PL por um AXI escravo juntamente com a SCU
- É usada para acelera a PL.
- Usado para troca de dados entre os processadores com o *hardware*.

Coprocessadores

Os processadores baseados em arquitetura ARM utilizam coprocessadores com o intuito de estender o conjunto de instruções do processador ARM:

- O NEOM^{TM} é um coprocessador para auxiliar em aplicações de multimídia.
- As instruções têm que ter menos ciclos de *clock* do que o processador ARM.
- Ativado via *software*.

O mecanismo de ponto flutuante é um coprocessador utilizado como uma extensão do NEOM™, sendo suas principais características:

- Registradores em conjunto com os do NEOM™.
- 2 MFLOPS/ performance por MHz.

2. Lógica programável

Na unidade lógica programável tem-se um FPGA Artix 7, sendo suas principais características:

- 8 LUTs por CLB para implementação;
- CLB contendo 16 flip-flops;
- LUTs possuem duas conFigurações como blocos de memórias 64x1 bits ou 32x2 bits
- Possuem registradores de deslocamento ou somadores 2x4-bits em cascata para funções aritméticas;
- Blocos de memória de 36kb de capacidade;
- Possui tecnologia SelectIO com suporte a DDR3 e interface de até 1.866 Mb/s;
- Possui conectividade serial de alta velocidade;
- Possui interface PCI Express, para até 8 vias;

3. Periféricos de entrada e saída.

O Zynq-7000 é composto por um conjunto de periféricos de comunicação:

- General Purpose Input/Output (GPIO);
- Gigabit Ethernet Controllers;
- Controladores USB: Um para comunicação com o Host, Dispositivos gerais e um para a OTG;
- Controladores SD / SDIO;
- Controladores SPI: *Master* ou *Slave*;
- Controladores CAN;
- Controladores UART;
- Controladores I2C;
- PS MIO I/Os.

O hardware utilizado neste trabalho foi o Zedboard, que é uma plataforma de desenvolvimento da Digilent, baseado no SOC ZYNQ-7000 da Xilinx, que contém várias interfaces de entrada e saída. Na Figura 4.3 tem-se a ilustração dos principais componentes e recursos de uma plataforma Zedboard[18].

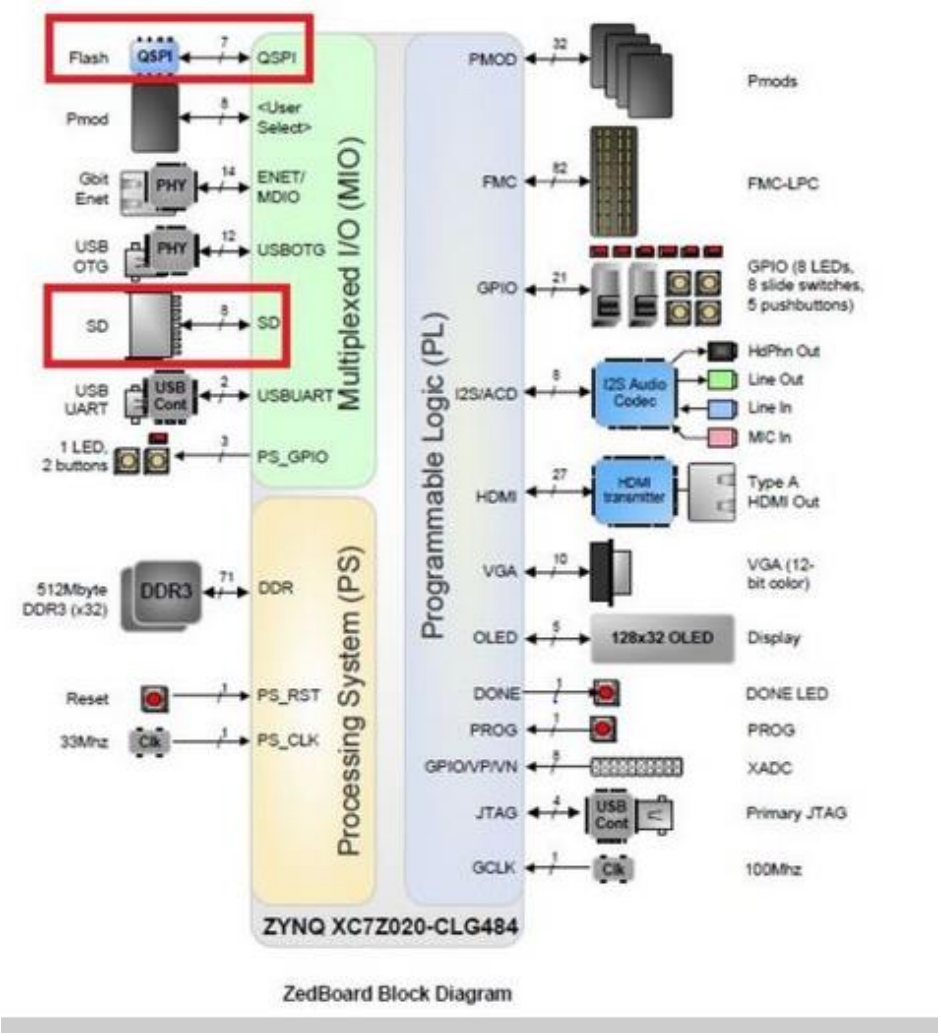

Figura 4.3: DIAGRAMA DO ZEDBOARD.
Fonte: Zedboard *hardware* user's Guide.

Os principais recursos fornecidos pela Zedboard são[18]:

- Xilinx XC7Z020-1CGL484CES Zynq-7000 AP SOC.
- Memória de 512 MB DDR3
- Memória Flash de 256 MB Quad SPI
- Cartão SD de 4GB
- Conectividade

1. Ethernet 10/100/1000
2. Entrada USB-JTAG Programação
3. Entrada de cartão SD
4. Entrada USB 2.0 FS Ponte USB-UART
5. Cinco Digilent Pmod headers
6. Conector FMC (Low Pin Count)
7. Entrada USB OTG 2.0 (Dispositivo / Host / OTG)
8. Dois botões de reinicialização (1 PS, 1 PL)
9. Sete botões de pressão (2 PS, 5 PL)
10. Oito interruptores (PL)
11. Nove LEDs (1 PS, 8 PL)
12. Porta de acesso de depuração ARM (DAP)
13. Xilinx XADC headers

- Osciladores na placa
  1. 33 333 MHz (PS)
  2. 100 MHz (PL)
- Áudio/Vídeo
  1. Saída HDMI
  2. VGA (cor de 12 bits)
  3. Display OLED 128x32
  4. Saída de Áudio Line-in, Line-out, fone de ouvido e microfone.
- Entrada de 12V CC @ 3,0 A (máx.)
- Dimensões de Comprimento 16 cm, largura 13,5 cm.

4.2 PRIMEIRO EXPERIMENTO

Nesta subseção iremos apresentar o primeiro experimento realizado com o intuito de explicar o funcionamento do protocolo de comunicação AXI4 *Lite* de Memória Mapeada, que consiste em uma versão simplificada do protocolo AXI4 de Memória Mapeada. Sendo que a principal diferença entre eles é que no AXI4 *Lite*, durante a transação de dados só ocorre uma rajada de dados e seus dados tem tamanho de 32 ou 64 bits, nas transações de leitura ou de escrita.

4.2.1 DESENVOLVIMENTO DO MÓDULO IP PARA A APLICAÇÃO

A primeira parte do experimento consiste no desenvolvimento de um *hardware* capaz de realizar multiplicações. Este *hardware* foi desenvolvido em VHDL. Na Figura 4.4 tem-se o código em VHDL, que foi utilizado para este experimento.

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.std_logic_arith.all;
4  use ieee.std_logic_unsigned.all;
5
6  entity multiplier is
7    port(
8      clk : in std_logic;
9      a   : in std_logic_vector(15 downto 0);
10     b   : in std_logic_vector(15 downto 0);
11     p   : out std_logic_vector(31 downto 0)
12   );
13 end multiplier;
14
15 architecture IMP of multiplier is
16
17 begin
18   process (clk)
19   begin
20     if clk'event and clk = '1' then
21       p <= a * b;
22     end if;
23   end process;
24 end IMP;
```

Figura 4.4: CÓDIGO EM VHDL DO PRIMEIRO EXPERIMENTO.
Fonte: Elaborado pelo autor.

O *hardware* criado através do código acima realiza operações com tamanho de dados de 16 bits e gera uma resposta de até 32 bits, funcionando sempre que o *clock* tiver em 1, realizando multiplicações entre as variáveis a e b de 16 bits. Com ele é criado no Vivado um módulo IP que se comunicar através do protocolo AXI4. Na Figura 4.5 tem-se o módulo IP que foi criado para este experimento.

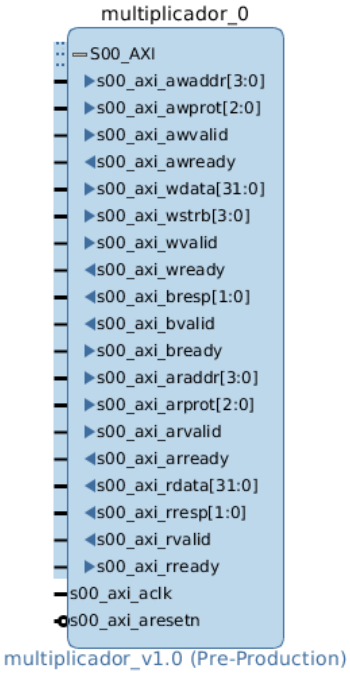

Figura 4.5: MÓDULO IP DO PRIMEIRO EXPERIMENTO.
Fonte: Elaborado pelo autor

Este IP funciona como um AXI4 *Slave*, com tamanho de dados de 32 bits, e irá se comunicar com o sistema criado através do protocolo de comunicação AXI4 *Lite* de Memória Mapeada, através dos vários barramentos e sinais de comunicação, leitura e escrita de dados. Separando esses barramentos e sinais pelos canais de comunicação tem-se:

1. Canal de endereço de escrita
    - O barramento s00_axi_awvalid, através desse barramento é enviado o sinal awvalid pelo AXI *Master* que indica que existem informações de endereço e controle a serem enviadas para o AXI *Slave*.
    - O barramento s00_axi_awready, através desse barramento é enviado o sinal awready pelo AXI *Slave* que indica para o AXI *Master*, que o AXI *Slave* está pronto para aceitar as informações de endereço e controle.
    - Os barramentos s00_axi_awaddr e s00_axi_awprot, consistem nos barramentos que enviam as informações de endereço e controle para que ocorra a transação de dados de escrita.

2. Canal de escrita de dados

- O barramento s00_axi_wvalid, através desse barramento é enviado o sinal wvalid pelo AXI *Master* que indica que existe uma transação de dados de escrita a ser enviada para o AXI *Slave*.
- O barramento s00_axi_wready, através desse barramento é enviado o sinal wready pelo AXI *Slave* que indica para o AXI *Master*, que o AXI *Slave* está pronto para aceitar a transação de dados de escrita enviado pelo AXI *Master*.
- O barramento s00_axi_wdata, que consiste no barramento de escrita de dados.
- O barramento s00_axi_wstrb, através desse barramento é enviado o sinal wstrb, que consiste em um sinal que auxilia o preenchimento de dados na memória. Onde a cada 8 bits enviados através do barramento de dados é enviado um bit de estouro através desse barramento.

3. Canal de resposta de escrita
    - O barramento s00_axi_bvalid, através desse barramento é enviado o sinal bvalid pelo AXI *Slave* que indica que existem informações de resposta a transação de dados a serem enviados para o AXI *Master*.
    - O barramento s00_axi_bready, através desse barramento é enviado o sinal bready pelo AXI *Master* que indica para o AXI *Slave*, que o AXI *Master* está pronto para aceitar as informações de respostas.
    - O barramento s00_axi_bresp, consiste no barramento que envia para o AXI *Master* um sinal emitido pelo AXI *Slave* de resposta para a escrita de dados. Onde ele sinaliza o status da transação e o fim da mesma.
4. Canal de endereço de leitura
    - O barramento s00_axi_arvalid, através desse barramento é enviado o sinal arvalid pelo AXI *Master* que indica que existem informações de endereço e controle a serem enviados para o AXI *Slave*.
    - O barramento s00_axi_arready, através desse barramento é enviado o sinal arready pelo AXI *Slave* que indica para o AXI *Master*, que o AXI *Slave* está pronto para aceitar as informações de endereço e controle.
    - Os barramentos s00_axi_araddr, s00_axi_arprot, que consistem nos barramentos que enviam as informações de endereço e controle para que ocorra a transação de dados de leitura.

5. Canal de leitura de dados
    - O barramento s00_axi_rvalid, através desse barramento é enviado o sinal rvalid pelo AXI *Slave* que indica que existe uma transação de dados de leitura a ser enviada para o AXI *Master*.
    - O barramento s00_axi_rready, através desse barramento é enviado o sinal rready pelo AXI *Master* que indica para o AXI *Slave*, que o AXI *Master* está pronto para aceitar a transação de dados de leitura do AXI *Slave*.
    - O barramento s00_axi_rdata consiste no barramento onde ocorre a transação de dados de leitura.
    - O barramento s00_axi_rresp, através desse barramento é enviado o sinal rresp pelo AXI *Slave* que sinaliza para o AXI *Master* um status da transação de dados de leitura.

Juntamente com esses canais tem-se os barramentos:

- O barramento s00_axi_aclk, que fornece o mesmo *clock* para o sistema e
- O barramento s00_axi_aresetn, que fornece um reset para o sistema.

O ‘s’ antes do nome dos barramentos indica que estamos visualizando o protocolo de comunicação pelo lado do AXI *Slave*. Se visualizamos o protocolo pelo AXI *Master* tem-se um ‘m’ antes do nome dos barramentos e sinais.

4.2.2 *HARDWARE*

Através do *software* Vivado é feito um *hardware* para atender aos quesitos utilizados no primeiro experimento. Na Figura 4.6 tem-se o diagrama de blocos deste *hardware*.

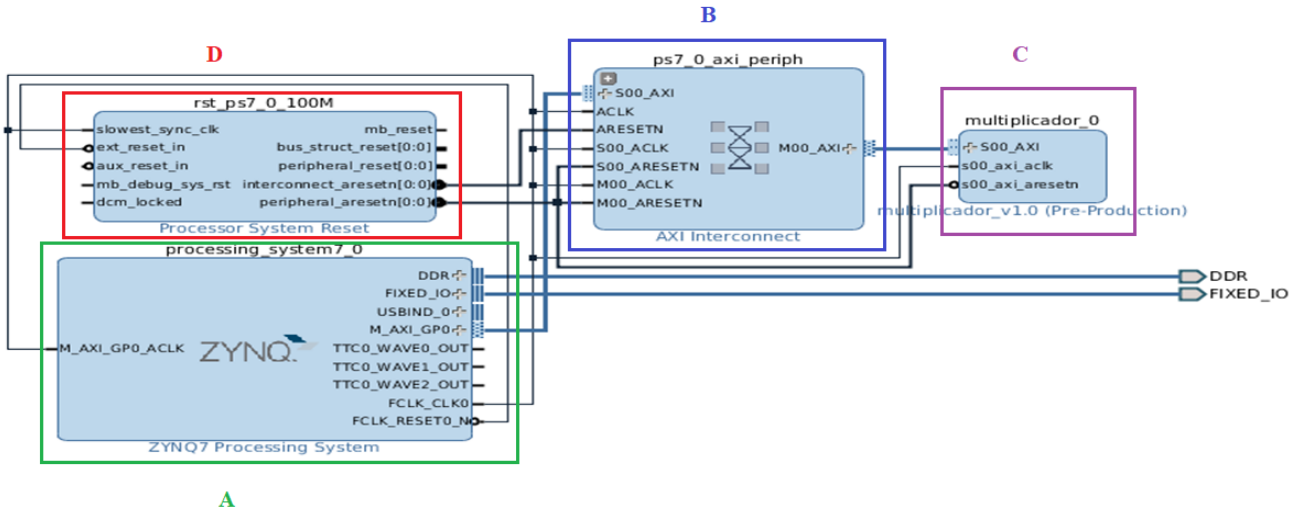

Figura 4.6: *HARDWARE* SINTETIZADO NO VIVADO DO PRIMEIRO EXPERIMENTO.
Fonte: Elaborado pelo autor.

No diagrama de blocos da Figura 4.6 é mostrado o *hardware* criado para o primeiro experimento. Este *hardware* é composto por módulos de IP's disponíveis na biblioteca da Xilinx e pelo o módulo IP criado previamente. Onde os seus componentes são:

- ZYNQ7 *Processing System (*A*)* é o sistema de processamento do SOC (PS). É responsável por realizar as operações de controle e transmissão de dados para a PL e os outros periféricos do projeto. Está conectado a ele, a memória DDR e as entradas e saídas fixas (FIXED_IO), que são responsáveis em auxiliar no processamento e transmissão de dados;

- AXI *Interconnect* (B) é o módulo IP que realiza a interface e controle das interconexões entre os vários módulos de IP do sistema, como por exemplo, a ligação entre o ZYNQ *Processing System* e o multiplicador_V1.0;

- multiplicador_V1.0 (C) é o módulo IP que foi criado para este experimento. Ele é capaz de realizar operações de 16 bits, com resultado em 32 bits. Onde um dado será transmitido para ele em 32 bits, e este dado será dividido em duas partes, onde a primeira metade será multiplicado pela segunda, e o resultado será salvo em um registrador e

- *Processor System Reset* (D) é um módulo IP instalado automaticamente durante a síntese do projeto, ele é responsável pelo controle de reinicializarão do sistema, ele está ligado a todos os módulos IP do sistema.

O ZYNQ7 *Processing System* (PS), que funciona como um AXI *Master*, estando conectado com o nosso módulo de IP, multiplicador_V1.0, por meio de uma interconexão AXI que realiza toda a interface de conexão entre eles. Funcionando como AXI *Slave* para a PS e como AXI *Master* para o módulo de IP. Nesta aplicação serão realizadas tanto operações de escrita, quando a PS transmite o dado que irá ser multiplicado para o bloco de IP, quanto de leitura quando o bloco de IP transmite a resposta para a PS. Todos os outros três componentes do sistema estão conectados com um Processor System Reset, que nos fornece um sistema de reinicialização, caso necessário.

4.2.3 *SOFTWARE*

Para a realização deste experimento foi feito um programa na linguagem C. Na Figura 4.7 tem-se o código em C criado para este experimento.

```
#include "platform.h"
#include "xbasic_types.h"
#include "xparameters.h"

Xuint32 *baseaddr_p = (Xuint32 *)XPAR_MULTIPLICADOR_0_S00_AXI_BASEADDR;

int main()
{
init_platform();

xil_printf("Teste do IP, Multiplicador\n\r");

*(baseaddr_p+0) = 0x00040002;
xil_printf("Escrita: 0x%08x \n\r", *(baseaddr_p+0));

xil_printf("Leitura : 0x%08x \n\r", *(baseaddr_p+1));

xil_printf("Fim do Teste.\n\n\r");

return 0;
}
```

Figura 4.7: CÓDIGO EM C DO PRIMEIRO EXPERIMENTO.
Fonte: Elaborado pelo autor.

Este programa é implementado na PS, por meio de uma conexão serial, através do *software* SDK. Através deste programa é possível entrar com os dados na PS, para que então eles sejam transmitidos para a PL, onde serão tratados e então retornarão para a PS, onde são demonstrados na tela do computador.

4.2.4 RESULTADOS

Os resultados da transação podem ser vistos no computador por meio de uma conexão serial. Na Figura 4.8 tem-se uma imagem do resultado gerado pelo primeiro experimento.

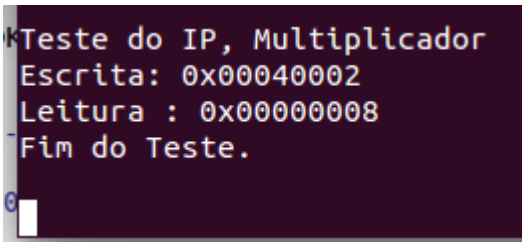

Figura 4.8: RESULTADO DO PRIMEIRO EXPERIMENTO.
Fonte: Elaborado pelo autor.

Primeiramente foi realizada uma transação de escrita, onde o AXI *Master* (PS) escreveu através do canal de escrita de endereço, as informações de endereço e controle que serão necessárias para a transação de dados. Em seguida através do canal de escrita de dados, o AXI *Master* transmite o dado, que está em hexadecimal, 0x00040002 para um registrador dentro do módulo IP, que funcionar como um AXI *Slave*. Após o fim da transmissão de dados o AXI *Slave*, envia um sinal de resposta para o AXI *Master*, sinalizando o recebimento da transmissão de dados. No módulo IP o dado foi dividido em duas partes, onde os primeiros 16 bits, 0004, foram multiplicados pelos últimos 16 bits, 0002, gerando assim o resultado 0x00000008, que é "salvo" em outro registrador. Este resultado é transmitido por uma transação de leitura. Onde através do canal de endereço de leitura, o AXI *Master* envia para o AXI *Slave* as informações de endereço e controle. Então o AXI *Slave* irá enviar para o AXI *Master* os dados. Juntamente com o envio dos dados é sinalizado, pelo AXI *Slave*, o fim da transação de dados. O resultado das transações é demonstrado no computador através de uma conexão por meio de uma porta serial.

4.3 SEGUNDO EXPERIMENTO

Neste tópico é apresentado o segundo experimento realizado com o intuito de explicar o funcionamento do módulo AXI DMA, que é o módulo IP capaz de realizar a interface de conexão entre o protocolo AXI4 de Memória Mapeada e o protocolo AXI4 *Stream*. Na Figura 4.9 tem-se um diagrama de bloco que ilustrar o funcionamento básico deste experimento.

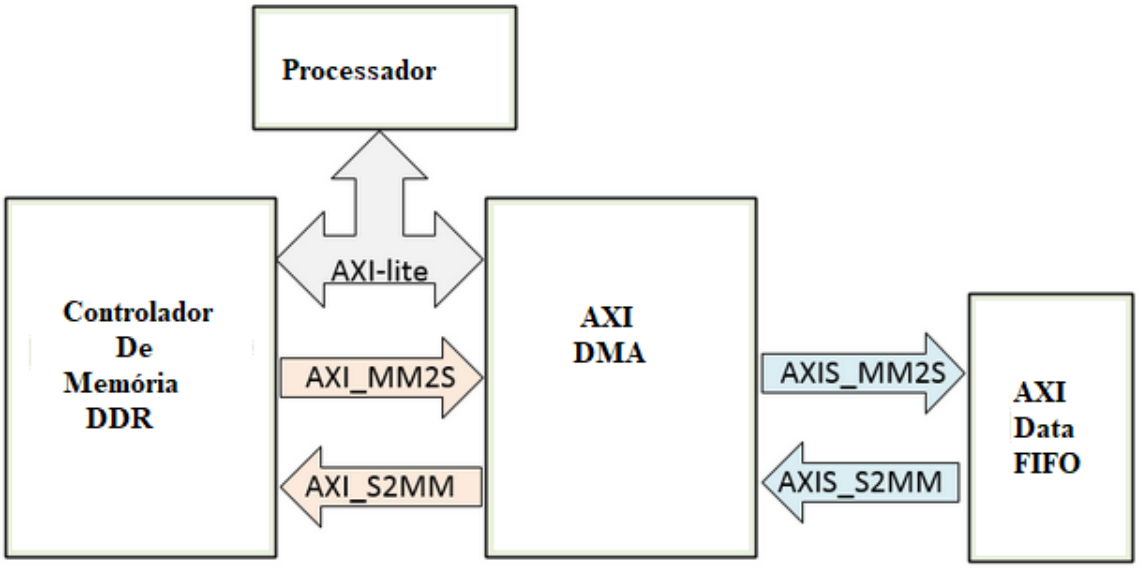

Figura 4.9: DIAGRAMA DE BLOCOS DO SEGUNDO EXPERIMENTO.
Fonte: Elaborado pelo autor.

O processador e o controlador de memória DDR estão embutidos na PS do SOC. O AXI DMA e o AXI Data FIFO, que se comunicam através do protocolo AXI4 *Stream*, serão implementados na PL do SOC. O barramento AXI-*lite* permite que o processador se comunique com o AXI DMA para realizar a configuração, controle e a transferência de dados. Os barramentos AXI_MM2S e AXI_S2MM se comunicam com o sistema através do protocolo AXI4 de memória mapeada e fornecem ao AXI DMA acesso a memória DDR. Os barramentos AXIS_S2MM e AXIS_MM2S se comunicam com o sistema através do protocolo AXI4 *Stream* e fazem uma transmissão de dados entre o AXI DMA e o AXI Data FIFO. O experimento consiste em inserirmos um dado na PS, este dado é transmitido por todo o sistema e retorna para a PS, gerando um loop. Para então ser testado e comprovar o funcionamento do AXI DMA.

4.3.1 *HARDWARE*

Através do *software* Vivado foi desenvolvido um *hardware* para atender aos quesitos utilizados no segundo experimento. Na Figura 4.10 tem-se o diagrama de blocos do *hardware* desenvolvido.

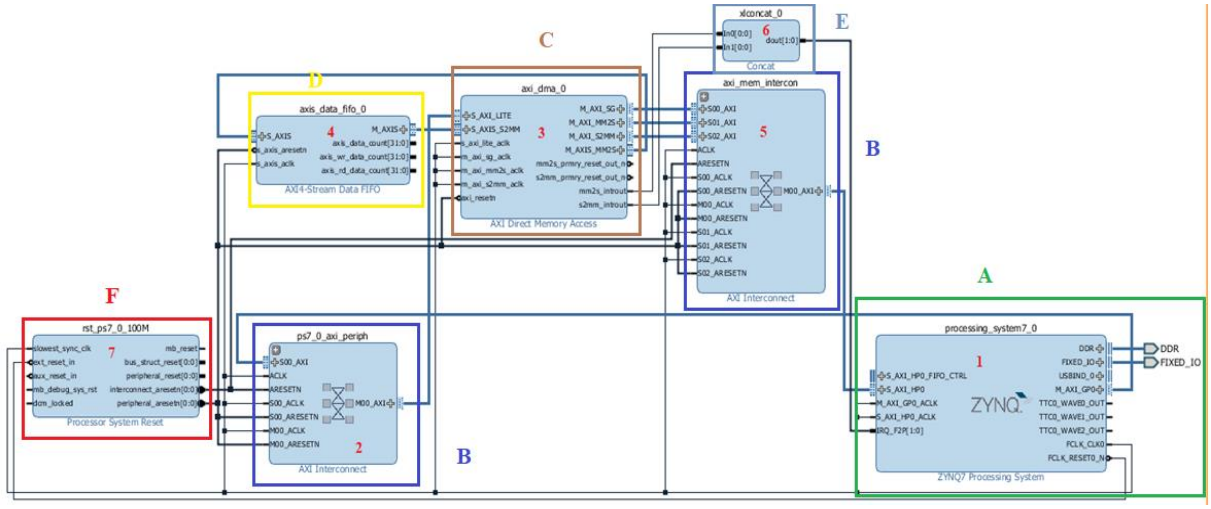

Figura 4.10: *HARDWARE* SINTETIZADO NO VIVADO DO SEGUNDO EXPERIMENTO.
Fonte: Elaborado pelo autor.

No diagrama de blocos da Figura 4.10 é mostrado o *hardware* desenvolvido para o segundo experimento. Este *hardware* é composto por módulos de IP's disponíveis na biblioteca da Xilinx. Onde os seus componentes são:

- ZYNQ7 *Processing System*, A, é o sistema de processamento do SOC (PS). É responsável por realizar as operações de controle e transmissão de dados para a PL e os outros periféricos do projeto. Está conectado a ele, a memória DDR e as entradas e saídas fixas (FIXED_IO), que são responsáveis em auxiliar no processamento e transmissão de dados;

- AXI *Interconnect*, B, é o módulo IP que realiza a interface e controle das interconexões entre os vários módulos de IP do sistema, como por exemplo, a ligação entre o ZYNQ *Processing System* e o AXI *Direct Memory Access*;

- AXI *Direct Memory* Access, C, é o módulo IP que realizar a interface de ligação entre os protocolos AXI4 de Memória Mapeada e AXI4 *Stream*;

- AXI4 *Stream* Data FIFO, D, é o módulo IP que realiza uma rápida e continua transmissão de dados, pois ele dispensa a etapa de endereçamento. Funciona como um FIFO, First In First Out, onde o primeiro dado que foi recebido será o primeiro dado a ser transmitido. Ele se comunica com o sistema através do protocolo AXI4 de *Stream*;

- *Concat*, E, é um módulo IP responsável por realizar a concatenação de barramentos com barramentos de dados de tamanhos diferentes e

- *Processor System Reset*, F, é um módulo IP instalado automaticamente durante a síntese do projeto, ele é responsável pelo controle de reinicializarão do sistema, ele está ligado a todos os módulos IP do sistema.

O ZYNQ7 *Processing System*, A, (PS) está conectado com o AXI DMA, C, por meio do AXI *Interconnect*, B, que funciona como AXI *Master* para o AXI DMA, C, e como AXI *Slave* para o ZYNQ7 *Processing System*. Eles se comunicam através do protocolo AXI4 de Memória Mapeada. Na Figura 4.11 tem-se a ligação entre esses módulos IP. Os barramentos e sinais presentes nas transações do protocolo AXI4 de Memória Mapeada já foram explicados no primeiro experimento.

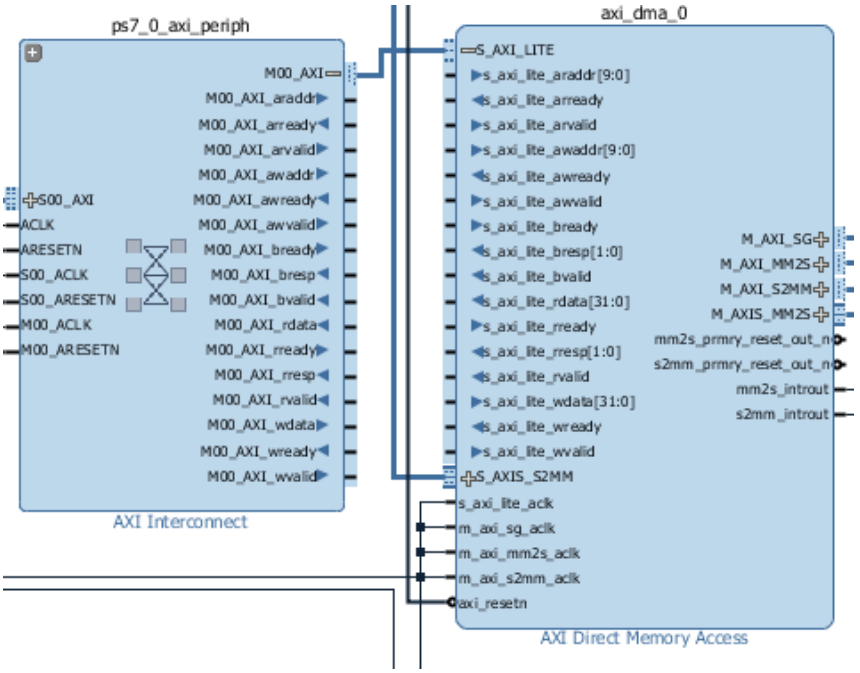

Figura 4.11: LIGAÇÃO ENTRE A INTERCONEXÃO AXI E O AXI DMA.
Fonte: Elaborado pelo autor.

O AXI DMA, C, está conectado diretamente com AXI4 *Stream* data FIFO, D. Eles se comunicam pelo protocolo AXI4 *Stream*. Como eles estão conectados em *loop*, ambos alternam seu funcionam entre AXI *Master* e AXI *Slave*. Na Figura 4.12 tem-se a ligação entre esses módulos IP e seus barramentos.

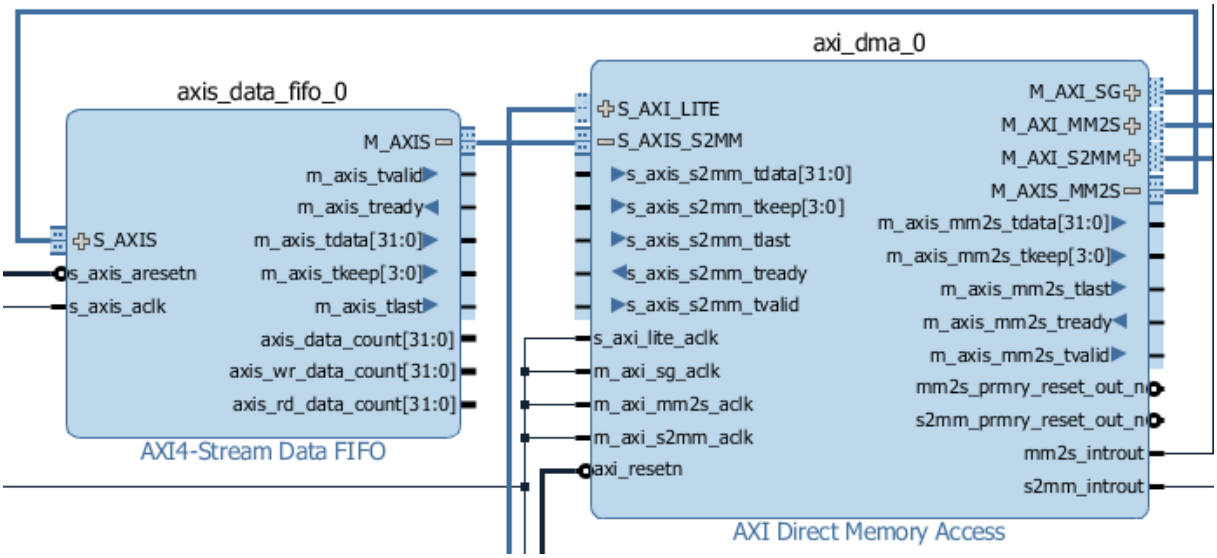

Figura 4.12: LIGAÇÃO ENTRE O AXI DMA E O AXI4-*STREAM* DATA FIFO.
Fonte: Elaborado pelo autor.

Na Figura acima tem-se um caso onde o AXI4 *Stream* Data FIFO funciona como AXI *Master* e o AXI DMA funciona como AXI *Slave*. Os barramentos envolvidos na transação de dados são:

- O barramento m_axis_tvalid que está interligado ao barramento s_axis_s2mm_tvalid. Onde por eles é enviado o sinal tvalid que indica que o AXI *Master* está pronto para enviar dados;

- O barramento m_axis_tready que está interligado ao barramento s_axis_s2mm_tready. Onde por eles é enviado o sinal tready que indica que o AXI *Slave* está pronto para receber dados;

- O barramento m_axis_tdata que está interligado ao barramento s_axis_s2mm_tdata. Onde por eles é enviado os dados do AXI *Master* para o AXI *Slave*, vale ressalta que no protocolo AXI4 *Stream* o envio de dados é unidirecional;

- O barramento m_axis_tkeep que está interligado ao barramento s_axis_s2mm_tkeep. Onde por eles é enviado o sinal tkeep que indica quais

são os bytes nulos, que podem ser removidos do fluxo de dados na transação e

- O barramento m_axis_tlast que está interligado ao barramento s_axis_s2mm_tlast. Onde por eles é enviado o sinal tlast que indica o fim da transação de dados.

O AXI DMA, C, que está funcionando como um AXI *Master*, também está conectado ao módulo ZYNQ7 *Processing System*, A, por meio de outra AXI Interconnect, B-5, que realiza toda a interface de conexão entre eles. Funcionando como AXI *Slave* para o AXI DMA e como AXI *Master* para a PS. Nesta aplicação serão realizadas somente operações de escrita, onde o AXI *Master* irá realizar envio de dados para o AXI *Slave*. O Concat está conectado ao AXI DMA e ao ZYNQ7 *Processing System*, para auxiliar no controle dos barramentos de dados, que tem largura de dados de tamanhos variados. Todos os componentes do sistema estão conectados com um Processor System Reset, que nos fornece um sistema de reinicialização, caso necessário.

4.3.2 *SOFTWARE*

Para a realização deste experimento foi utilizado um programa na linguagem C da biblioteca de exemplos do *software* SDK. Na Figura 4.13 tem-se a primeira parte do código em C que foi utilizado para este experimento.

```
int main(void)
{
        int Status;
        XAxiDma_Config *Config;

#if defined(XPAR_UARTNS550_0_BASEADDR)

        Uart550_Setup();

#endif

        xil_printf("\r\n--- Entering main() --- \r\n");

#ifdef __aarch64__
        Xil_SetTlbAttributes(TX_BD_SPACE_BASE, MARK_UNCACHEABLE);
        Xil_SetTlbAttributes(RX_BD_SPACE_BASE, MARK_UNCACHEABLE);
```

Figura 4.13: CÓDIGO EM C DE INICIALIZAÇÃO DO AXI DMA.
Fonte: Elaborado pelo autor.

Este programa é implementado na PS, por meio de uma conexão serial, através do *software* SDK. Onde é conFigurado e iniciado o AXI DMA, para então é inserido um dado no mesmo através da PS, este dado será enviado para todo o sistema e voltará para a PS, gerando um loop. Na Figura 4.14 tem-se a parte do programa em C que realiza os testes na transmissão de dados no sistema.

```
/* Initialize DMA engine */
Status = XAxiDma_CfgInitialize(&AxiDma, Config);
if (Status != XST_SUCCESS) {
        xil_printf("Initialization failed %d\r\n", Status);
        return XST_FAILURE;
}

if(!XAxiDma_HasSg(&AxiDma)) {
        xil_printf("Device configured as Simple mode \r\n");

        return XST_FAILURE;
}

Status = TxSetup(&AxiDma);
if (Status != XST_SUCCESS) {
        return XST_FAILURE;
}

Status = RxSetup(&AxiDma);
if (Status != XST_SUCCESS) {
        return XST_FAILURE;
}

/* Send a packet */
Status = SendPacket(&AxiDma);
if (Status != XST_SUCCESS) {
        return XST_FAILURE;
}

/* Check DMA transfer result */
Status = CheckDmaResult(&AxiDma);

xil_printf("AXI DMA SG Polling Test %s\r\n",
        (Status == XST_SUCCESS)? "passed":"failed");

xil_printf("--- Exiting main() --- \r\n");
```

Figura 4.14: CÓDIGO EM C DOS TESTES NA TRANSMISSÃO DO AXI DMA.
Fonte: Elaborado pelo autor.

Durante o a transação de dados são realizados vários testes no sistema, onde é testada a inicialização do AXI DMA, o envio e a transferência do pacote de dados através do AXI DMA. O resultado é demonstrado na tela do computador.

4.3.3 RESULTADOS

Os resultados do teste do AXI DMA podem ser vistos no computador por meio de uma conexão serial. Na Figura 4.15 tem-se uma imagem do resultado gerado pelo segundo experimento.

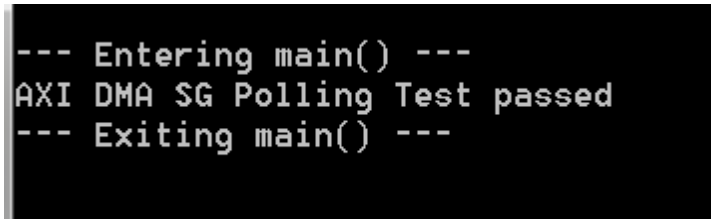

Figura 4.15: RESULTADO DO SEGUNDO EXPERIMENTO.
Fonte: Elaborado pelo autor

Primeiramente é realizada uma transação de escrita de dados, onde o ZYNQ7 *Processing System,* funcionando como AXI *Master,* iniciou uma transação do protocolo AXI4 Memória Mapeada, onde ele enviou as informações de controle e endereços e os dados pelo barramento M_AXI_GPO para o barramento S_AXI_*Lite* do AXI DMA através do AXI Interconnect, onde ele funciona como AXI *Master* para o AXI DMA e como AXI *Slave* para o ZYNQ7 *Processing System*. O AXI DMA através do barramento M_AXIS_MM2S realiza uma transação de dados do protocolo AXI4 *Stream* para o barramento S_AXIS do AXI *Stream* data FIFO, onde o AXI DMA funciona como AXI *Master* e o AXI *Stream* DATA FIFO como AXI *Slave*. O AXI *Stream* Data FIFO através do barramento M_AXIS transmite o dado para o barramento S_AXIS_S2MM do AXI DMA, onde o AXI *Stream* data FIFO funciona como AXI *Master* e o AXI DMA como AXI *Slave*. Em seguida o AXI DMA, através do barramento M_AXI_S2MM, inicia uma transação de dados do protocolo AXI4 de Memória Mapeada para o barramento S_AXI_HP0 do ZYNQ7 *Processing System* por meio do AXI Interconnect, onde ele funciona como AXI *Slave* Para o AXI DMA e como AXI *Master* para o ZYNQ7 *Processing System*. O resultado das transações é demonstrado no computador através de uma conexão por meio de uma porta serial.

5. CONCLUSÃO

O desenvolvimento do estudo do protocolo de comunicação AXI é de grande importância para a compreensão do funcionamento de sistemas em um *chip*. Neste trabalho foi apresentado todo o seu funcionamento partindo da sua arquitetura, onde foram mostrados o seu princípio de funcionamento e os seus principais componentes, bem como, suas interconexões. Em um segundo momento foi mostrado como ocorrem às transações, de leitura e de escrita, as variações do protocolo AXI e como essas variações se interligam em um sistema. Para fim de uma demonstração pratica do protocolo de comunicação AXI, foram realizados dois experimentos. Eles foram feitos no ambiente de desenvolvimento Vivado e no kit de desenvolvimento Zedboard, que é composto por um SoC Zynq-7000.

No primeiro experimento foi criado, no vivado, um módulo IP que se comunica através do protocolo AXI *Lite* de memória mapeada, este IP foi integrado a um *hardware*, composto pela PS, uma interconexão AXI e um sistema de reset. Esse *hardware* foi implementado no SoC Zynq-7000 juntamente com um *software*, um programa em linguagem C para realizar o controle e fornece os dados para a aplicação. Com esse experimento foi possível demonstrar, de uma forma prática, o funcionamento do protocolo AXI4 de memória mapeada. Nele foram demonstrados todos os sinais envolvidos nas transações e como eles ocorrem no sistema.

No segundo experimento foi mostrado o funcionamento do módulo AXI DMA, onde foi feito no Vivado, um *hardware* que possui as duas variações do protocolo de comunicação AXI, comunicando entre si através desse módulo. Esse *hardware* foi implementado no SoC, juntamente com um *software* em linguagem C, que está presente na biblioteca da Xilinx. Com esse experimento além de explicar o funcionamento do módulo AXI DMA, foi demonstrado de uma forma prática como funciona a transação de dados no protocolo AXI4 *Stream* e os sinais presentes na transação de dados.

Com o estudo do protocolo e os dois experimentos realizados, foi possível realizar uma explicação sobre o funcionamento do protocolo de comunicação AXI de uma forma simplificada e prática eliminando vários detalhes menores desnecessários para o entendimento básico do mesmo.

REFERÊNCIAS BIBLIOGRÁFICAS

[1]CONTI, FATIMA. **Historia da informática e da internet 1900 a 1939.** Disponível em < http://www.ufpa.br/dicas/net1/int-h190.htm > Acesso em 02/06/2017

[2]EMBEDDED-INSIGHTS. **ARM1176JZ-S/ARM1176JZF-S**. Disponível em: <http://www.embeddedinsights.com/epd/arm/arm-arm1176jz-s-arm1176jzf-s.php> Acesso em 29/05/2017

[3]XILINX, ***AXI Reference Guide,*** V13.1 de 7 de março de 2011.

[4]XILINX, ***ZYNQ-7000 All Programmable SoC Software Developers Guide***, V12.0 de 30 de setembro de 2015.

[5]XILINX. ***Hyundai's wearable exoskeleton restores personal mobility.*** Disponível em <https://forums.xilinx.com/t5/Xcell-Daily-Blog/Hyundai-s-wearable-exoskeleton-restores-personal-mobility-to-the/ba-p/647031> Acesso em 15/06/2017

[6]CIPOLI, PEDRO. **Cortex-A9, Cortex-A15...Conheça os diferentes tipos de processadores móveis.** Disponível em <https://canaltech.com.br/produtos/cortex-a9-cortex-a15conheca-os-diferentes-tipos-de-processadores-moveis-6738/> Acesso em 05/06/2017

[7]STALLINGS, WILLIAM, ***Arquitetura E Organização de Computadores***, 8ª Edição, editora: Prentice Hall, Brasil, 2010.

[8]ELECTEL. ***Understanding Processor Architecture: RISC versus CISC.*** Disponível em < electel.blogspot.com.br/2015/07/understanding-processor-architecture-cisc-vs-risc.html> Acesso em 19/06/2017

[9]ZEIDMAN, BOB. ***EETimes All about FPGAS.*** Disponível em < eetimes.com/document.asp?doc_id=1274496> Acesso em 15/06/2017

[10]JUNCAS. **Protocolo de Comunicação de Dados.** disponível em < http://soalexjuncal.blogspot.com.br/2010/02/protocolo-de-comunicacao-de-dados.html> Acesso em 09/05/2017

[11]XILINX, ***ZYNQ-7000 All Programmable SoC Overview, Product Specification,*** V1.10 de 27 de setembro de 2016.

[12]ARM. ***Cortex-A9 Processor.*** Disponível em <arm.com/products/processors/cortex-m/cortex-a9.php> Acesso em 25/06/2017

[13]XILINX. **7 Series FPGAs Data Sheet: Overview, Product Specification.** V2.4 28 de março de 2017

[14]ARM, ***AMBA AXI Protocol Specification,*** V2.0 de 03 de março de 2010.

[15]ARM, ***AMBA 4 AXI4-Stream Protocol Specification,*** V1.0 de 03 de março de 2010.

[16]XILINX, ***Vivado Design Suite, User*** *Guide* de 18 de novembro de 2015.

[17]CROCKETT, Louise h. ELLOT, Ross A. ENDERWITZ, Martin A. e STEWART, Robert W. ***The ZYNQ Book***. Strathclyde Academic Media, 1ª edição, Scotland, UK: 2014.

[18]AVNET. **ZedBoard (Zynq Evaluation and Development Hardware User's Guide).** V2.2 de 27 de janeiro de 2014

[19]SYMES, Dominic, Wright, Chris e Rayfield, John. ***ARM System Developer's Guides*.** Editora Elsevier, 4ª edição,San Francisco: 2004.

[20]XDA. ***No-Odin Root Exploit Found for Exynos 4412 and 4210.*** Disponível em <https://www.xda-developers.com/no-odin-root-exploit-found-for-exynos-4412-and-4210/> Acesso em 12/06/2017

[21]YOGASINGAM, ALAM. ***Apple A6X uncovered.*** Disponível em < http://www.embedded.com/print/4401455> Acesso em 12/06/2017

[22]TECHTUDO. ***Intel 4004 o primeiro processador da historia faz 40 anos de idade.*** Disponível em <http://www.techtudo.com.br/artigos/noticia/2011/11/intel-4004-o-primeiro-processador-da-historia-comemora-40-anos-de-idade.html> Acesso em 17/06/2017

[23]JOHNSON, JEFF. ***Creating a Custom IP block in Vivado.*** Disponível em < http://www.fpgadeveloper.com/2014/08/creating-a-custom-ip-block-in-vivado.html> Acesso em 10/04/2017

[24]JOHNSON, JEFF. ***Using the AXI DMA in Vivado.*** Disponível em < http://www.fpgadeveloper.com/2014/08/using-the-axi-dma-in-vivado.html> Acesso em 25/05/2017

[25]IMRAN, AWAIS. ***Apple A6 Processor Goes Into Production; Being Manufactured By TSMC, Not Samsung.*** Disponível em < Redmondpie.com/apple-a6-processor-goes-into-production-being-manufactured-by-tsmc-not-samsung-report> Acesso em 24/06/2017

[26]DIGILENT. ***Getting Started with the Basys 3.*** Disponível em <reference.digilentinc.com/learn/programmable-logic/tutorials/basys-3-getting-started/start> Acesso em 27/06/2017