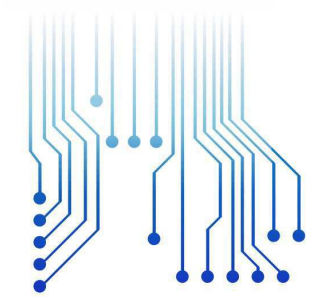


CURSO DE ENGENHARIA ELÉTRICA

THIAGO MORAIS DE OLIVEIRA

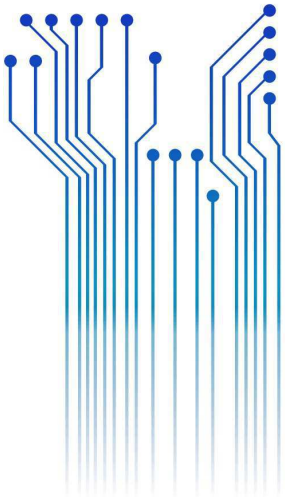
Universidade Federal
de Campina Grande

Centro de Engenharia
Elétrica e Informática

RELATÓRIO DE ESTAGIO SUPERVISIONADO

LABORATÓRIO EMBEDDED

Departamento de
Engenharia Elétrica



CAMPINA GRANDE
DEZEMBRO DE 2019

THIAGO MORAIS DE OLIVEIRA

LABORATÓRIO EMBEDDED

RELATÓRIO DE ESTAGIO SUPERVISIONADO submetido à Unidade Acadêmica de Engenharia Elétrica da Universidade Federal de Campina Grande, como parte dos requisitos necessários para a obtenção do grau de Bacharel em Ciências no Domínio da Engenharia Elétrica.

Área de concentração: Microeletrônica

Orientador:
Professor Marcos Ricardo Alcântara Morais, D.Sc.

CAMPINA GRANDE
DEZEMBRO DE 2019

THIAGO MORAIS DE OLIVEIRA

LABORATÓRIO EMBEDDED

RELATÓRIO DE ESTAGIO SUPERVISIONADO submetido à Unidade Acadêmica de Engenharia Elétrica da Universidade Federal de Campina Grande, como parte dos requisitos necessários para a obtenção do grau de Bacharel em Ciências no Domínio da Engenharia Elétrica.

Área de concentração: Microeletrônica

Aprovado em: / /

Professor Gutemberg Gonçalves dos Santos Júnior, D.Sc.
Universidade Federal de Campina Grande
Avaliador

Professor Marcos Ricardo Alcântara Morais, D.Sc.
Orientador, UFCG

CAMPINA GRANDE
DEZEMBRO DE 2019

Agradecimentos

Agradeço primeiramente a Deus pela minha vida e à imaculada virgem Maria por interceder por mim durante as tribulações. Seguidamente, aos meus queridos pais, Tony Marcus Lima de Oliveira e Fancinete Patrícia de Morais por todo amor e por acreditarem no meu potencial, até quando nem mesmo eu fui capaz de fazê-lo. Aos meus avós Antônio Neno, Nizete, Francisco Aprígio e Maria Joana por todas as histórias durante os cafés da tarde, valores e por me ensinarem o verdadeiro significado de família.

Agradeço a Yara Synthia, por estar ao meu lado nos melhores e nos piores momentos, por todo apoio, carinho e cumplicidade.

A Ariel, Yanca, Yana Priscila e Sérgio Júnior por todo apoio, descontrações e por serem minha segunda família longe de casa, a minha irmã Isabely pelas discretas demonstrações de afeto e aos meus estimados amigos Marco Aurélio e Edylla, pelo companheirismo, passeios de fim de tarde e boas rizadas.

Agradeço aos meus colegas de curso e amigos Jesney, Matheus Andrade, Mylena e Ianca pela sinceridade, assistência e pela honrra de tê-los ao meu lado nessa jornada, seja durante as várias madrugadas de estudo ou nos felizes dias de confraternização.

Agradeço ao laboratório X-MEN, que me trouxe grande satisfação e engrandecimento profissional, em especial ao professor Gutemberg, Antônio Agripino e a Bruno Silva por acreditarem no meu trabalho, por todas as lições e dedicação.

Aos meus professores, por todo o empenho e responsabilidade, em especial ao professor Marcos Morais pela paciência e compromisso em me orientar nesta etapa final da minha formação.

"It's not about how hard you hit. It's about how hard you can get hit and keep moving forward. How much you can take and keep moving forward"

Sylvester Stallone, Rocky Balboa

Resumo

Este relatório apresenta as atividades realizadas pelo aluno graduando Thiago Morais de Oliveira durante o período de estágio supervisionado exercido no Laboratório Embedded, localizado no Departamento de Engenharia Elétrica da Universidade Federal de Campina Grande. Os trabalhos tiveram como objetivo principal integrar IPs utilizando o padrão IP-XACT através a ferramenta Kactus2, possibilitando o reúso e modularização das instâncias. Para isso, foram feitas pesquisas bibliográficas acerca padrão IEEE 1685 assim como implementações e experimentos com o *software* escolhido. Além da geração de um componente através da HDL, a realização dos testes envolveu a implementação de um protocolo de comunicação SPI e a integração dos componentes básicos de uma CPU. A metodologia de integração da ferramenta foi capaz de gerar um verilog sintetizável das instâncias corretamente, segundo os esquemas XML dos componentes.

Palavras-chave: IP-XACT, Kactus2, Reúso, Integração.

Abstract

This report presents the activities performed by the student Thiago Morais de Oliveira during the supervised internship at the Embedded Laboratory, located in the Department of Electrical Engineering of the Federal University of Campina Grande. The main objective of the works was IP integration using the IP-XACT standard through the Kactus2 tool, enabling the instances reuse and modularization. To this end, we conducted bibliographic research on the IEEE 1685 standard as well as implementations and experiments with the chosen software. In addition to the generation of a component through HDL, the tests involved implementing an SPI communication protocol and a CPU basic components integrating. The tool integration methodology was able to generate a synthesizable verilog of the instances correctly, according to the XML schemas of the components.

Keywords: IP-XACT, Kactus2, Reuse, Integration.

Lista de tabelas

Lista de ilustrações

Lista de abreviaturas e siglas

ASIC	*Application Specific Integrated Circuit*
CPU	*Central Processing Unit*
EDA	*Electronic Design Automation*
HDL	*Hardware Description Language*
I2S	*Inter-IC Sound*
IEEE	*Institute of Electrical and Electronics Engineers*
IP	*Intellectual Property*
RTL	*Register-transfer level*
SoC	*System on Chip*
SPI	*Serial Peripheral Interface*
ALU	*Arithmetic Logic Unit*
XML	*Extensible Markup Language*

Sumário

1 Introdução

Este relatório apresenta as atividades realizadas pelo aluno graduando Thiago Morais de Oliveira durante o período de estágio supervisionado exercido no Laboratório Embedded sob a orientação do professor Marcos Ricardo Alcântara Morais e a supervisão do professor Gutemberg Gonçalves dos Santos Júnior. O referido estágio teve vigência entre os dias 23 de Setembro de 2019 até 11 de Dezembro do mesmo ano, tendo como objetivo o estudo e modelagem de *hardware* utilizando linguagem de alto nível.

Um dos grandes focos de atenção nas indústrias de microeletrônica tem sido acerca da modularidade e reúso de IP, isto é, a possibilidade de utlização de blocos previamente concebidos para a implementação de novas funções. Esta prática, quando bem executada, traz benefícios ao grupo de projetistas, reduzindo tempo e esforço computacional e humano, fazendo valer as célebres palavras de Andy Hunt e Dave Thomas: "*Don't repeat yourself*", ou "Não se repita", ou seja, o trabalho do *designer* deve compreender a solução de problemas, ao passo que projetos passados devem ser automatizados e reutilizados em diferentes situações.

A comunicação entre os provedores e consumidores de IP, de maneira geral, é estabelecida por meio de documentos como *user guide*, *datasheets*, RAKs, notas de implementação além do RTL, modelos de simulação, *constraints*, entre outros arquivos. A assimilação de um produto requer a análise humana destes arquivos, dispendiando tempo na etapa de integração.

Com base nestas situações, foi criado o padrão IP-XACT (IEEE 1685-2009) que pode ser visto como uma documentação eletrônica que pode ser interpretada por humanos e programas, de modo a descrever IPs, permitindo a integração dos mesmos por meio de *software*.

1.1 Objetivos

1.1.1 Objetivos Gerais

Os trabalhos realizados tiveram como objetivo a integração de IPs utilizando o padrão IP-XACT através a ferramenta Kactus2.

1.1.2 Objetivos específicos

- Revisão bibliográfica sobre o padrão IP-XACT e da ferramenta Kactus2;
- Descrição segundo o padrão adotado de um componente através de seu RTL;
- Comunicação de alto nível entre módulos;
- Geração de HDL a partir do modelo de integração.

1.2 Metodologia

A realização desse trabalho envolveu a pesquisa bibliográfica do padrão IEEE 1685 e como a ferramenta escolhida opera segundo o mesmo. A escolha do Kactus2 como a melhor opção se deu através de estudos de ferramentas de mercado em desenvolvimento que ralizam a modelagem e integração de IP através de *software*.

Foram escolhidos esquemas XML e suas respectivas HDLs disponibilizados pela comunidade de desenvolvedores do Kactus2 como base de estudos práticos da mesma.

1.3 O laboratório X-MEN

O estágio foi realizado no Laboratório de Excelência em Microeletrônica do Nordeste (X-MEN), parte do laboratório Embedded que desenvolve projetos de pesquisa, desenvolvimento e inovação aplicadas na área de microeletrônica digital em parceria com o VIRTUS.

2 Fundamentação Teórica

Dada uma especificação por um contratante, os passos do fluxo de desenvolvimento ASIC consistem no particionamento dos blocos em unidades de processamento e comunicação menores para a execução de um SoC. Muitas vezes, em um projeto, não são desenvolvidos todos os blocos, sendo integrados IPs de outros fabricantes (como memórias, por exemplo). É tarefa do engenheiro de integração, unir cada parte do *design* obedecendo os protocolos de comunicação particulares de cada bloco.

A tarefa do integrador não é fácil e depende diretamente da análise de documentos, arquivos e *constraints* particulares de cada unidade. O reúso de IPs, isto é, o aproveitamento de blocos feitos previamente ou importados de outros fabricantes, é de extrema importância para garantir a agilidade e otimização do fluxo de projeto.

2.1 IP-XACT

IP-XACT foi criado inicialmente em 2004 a partir de uma associação de companias (SPIRIT) para desenvolvimento de um padrão para integração de um IP em um SoC. Em 2010, o IP-XACT foi tido como padrão IEEE fornecendo uma descrição de um IP, ou de um sistema de IPs interconectados, interpretável por máquina. Essa descrição utiliza o formato XML para guardar informações de modo que as ferramentas de alto nível são utilizadas para edição dos componentes, comunicação entre instâncias segundo protocolos e geração de *hardware*.

Segundo o padrão IEEE 1685-2014, os elementos básicos da linguagem são: componentes (*component*), *design*, configuração de *design*, definição de barramento (*bus definition*), definição de abstração (*abstraction definition*), abstrator (*abstractor*), *generator chain* e catálogo (*catalog*).

O documento **componente** trata da instanciação da interface do IP sem a informação de operação do bloco. Em outras palavras, esse documento descreve a forma que o IP

é visto por outros blocos conforme o seu RTL, abarcando características como parâmetros, registradores, entradas, saídas, etc.

O ***design*** descreve a estrutura de conexão entre IPs, permitindo a comunicação entre componentes. A combinação entre componentes e projetos permite o reúso de um IP a partir do encapsulamento do protocolo de comunicação, agora lido e instanciado pela ferramenta. São descritos nesse documento as instâncias e os parâmetros dos componentes internos e as conexões entre eles.

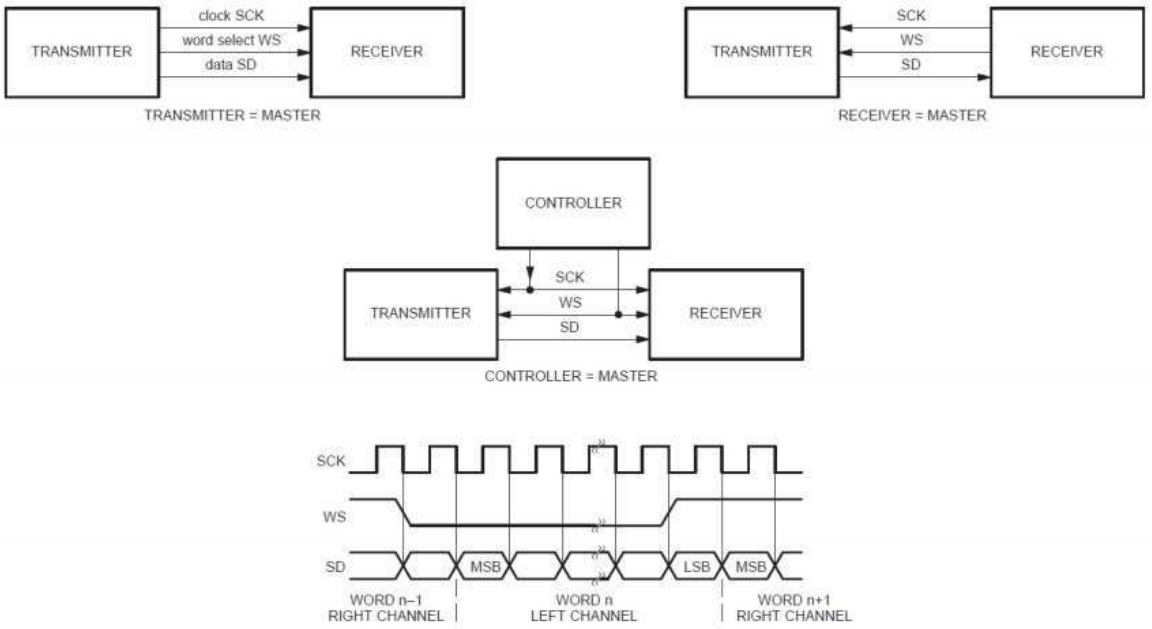
O documento de **configuração de *design*** compreende a abstração da comunicação entre as instâncias do projeto e os modos de operação do *design*.

O protocolo de comunicação utilizado é descrito no documento de **definição de barramento**, onde são elencados os componentes que utilizam tais protocolos. A correta conexão entre componentes por meio de interfaces de barramento só será feita se ambos referenciarem a mesma definição de barramento.

As definições supracitadas são suficientes para entender o conceito de modularidade e reúso de IP via padrão IP-XACT. Um exemplo de uso desse padrão é ilustrado a seguir.

Consideremos uma comunicação entre um controlador, um transmissor e um receptor por meio do protocolo I2S conforme esquematizado na Figura 1.

Figura 1 – Topologias e comunicação I2S.



Fonte: Accellera Systems Initiative, 2018.

Por definição, qualquer um dos blocos pode enviar os sinais de *clock* (SCK) e dado

(SD), exercendo a função de *master*. As situações possíveis são:

- Transmissor no modo *master*;
- Transmissor no modo *slave*;
- Receptor no modo *master*;
- Receptor no modo *slave*;
- Controlador *master*.

Cada uma das situações descritas acima consiste na mudança do RTL de cada bloco, devido o comportamento de cada pino. A padronização consiste em visualizar as instâncias tratadas como **componentes**, cuja funcionalidade será descrita pelo seu RTL para cada operação. O ***design*** corresponde a interconexão destes componentes por meio do barramento I2S, cujo protocolo é indicado no documento de **definição de barramento**.

Uma vez colocados seguindo o padrão IP-XACT, a geração do código HDL desta integração pode ser completamente automatizado por meio de ferramentas computacionais, como o Kactus2, descrito na seção seguinte.

2.2 Kactus2

Kactus2 é uma ferramenta EDA de código aberto para projeto de SoCs utilizando o padrão IP-XACT. A integração dos IPs é feita de modo gráfico e intuitivo favorecendo o reúso, aumentando a produtividade na integração e desenvolvimento de *hardware*.

Por ser uma ferramena gráfica, o Kactus2 pode ser incorporado no fluxo de desenvolvimento do ASIC, se encaixando entre a especificação e a codificação RTL, onde são definidas as interfaces dos blocos. Uma vez modelados em componentes IP-XACT, as interfaces dos IPs podem ser geradas automaticamente.

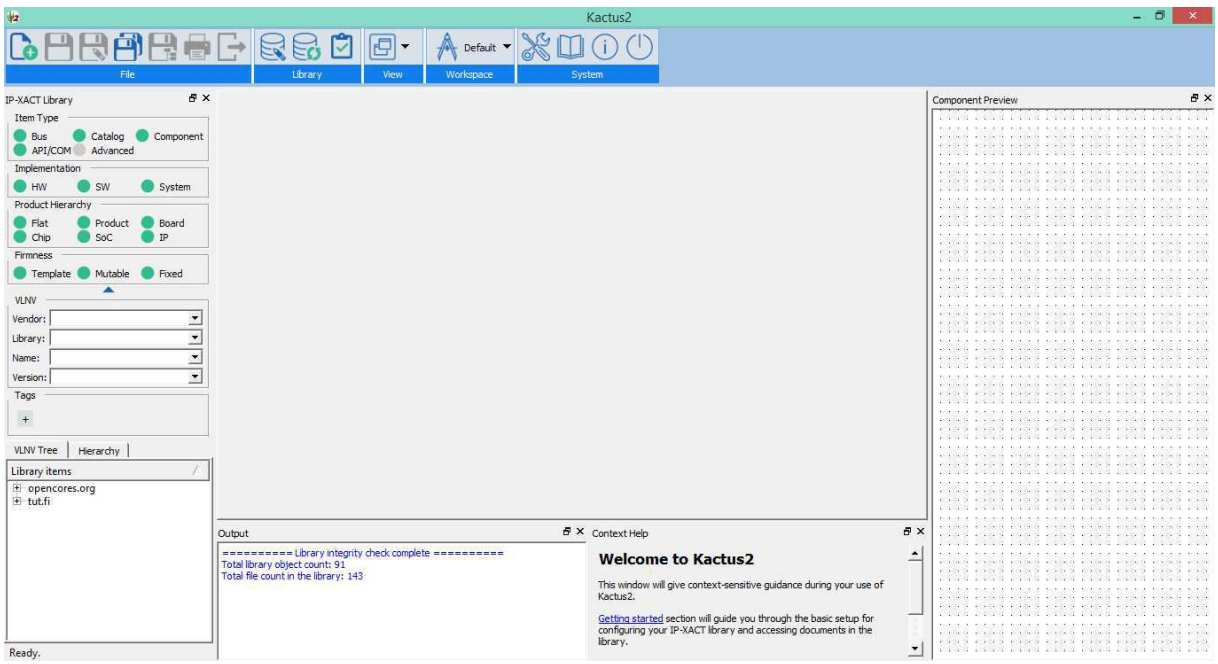
A troca ou reúso de componentes em um *design* é encapsulada pela ferramenta, que é capaz de fazer a integração dos blocos, gerando a HDL segundo o protocolo estabelecido.

3 Testes realizados

Para os testes, foram utilizados esquemas XML e protocolos já no formato IP-XACT disponibilizados pela própria ferramenta Kactus2 com o objetivo de gerar os componentes e estabelecer a conexão entre eles em um *design* hierárquico.

A interface gráfica da ferramenta segue conforme Ilustrado na Figura 2. Será explanado a seguir como se dá a criação de um componente no ambiente Kactus2, partindo da adição de uma HDL, gerando o topo da interface da mesma.

Figura 2 – GUI da ferramenta Kactus2



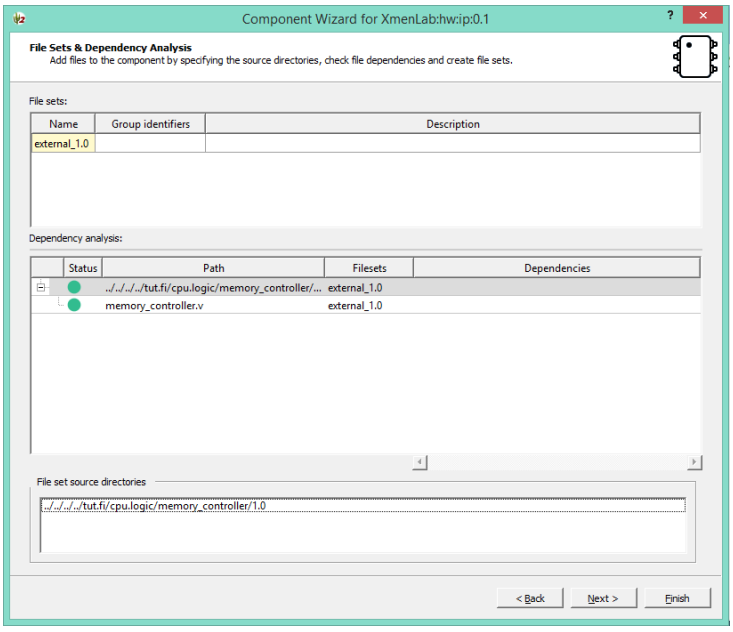
Fonte: O autor.

Ao importar as devidas bibliotecas de produtos, podemos criar um novo componente a partir da ferramenta “Component Wizard” (Figura 3), onde o diretório do HDL do projeto é informado. No próximo passo (Figura 4), a ferramenta interpreta a linguagem verilog

identificando parâmetros e sinais de entrada e saída, sendo possível a inserção de outros pinos. A ferramenta tem suporte a operações matemáticas de parâmetros, implicando em um RTL mais modular.

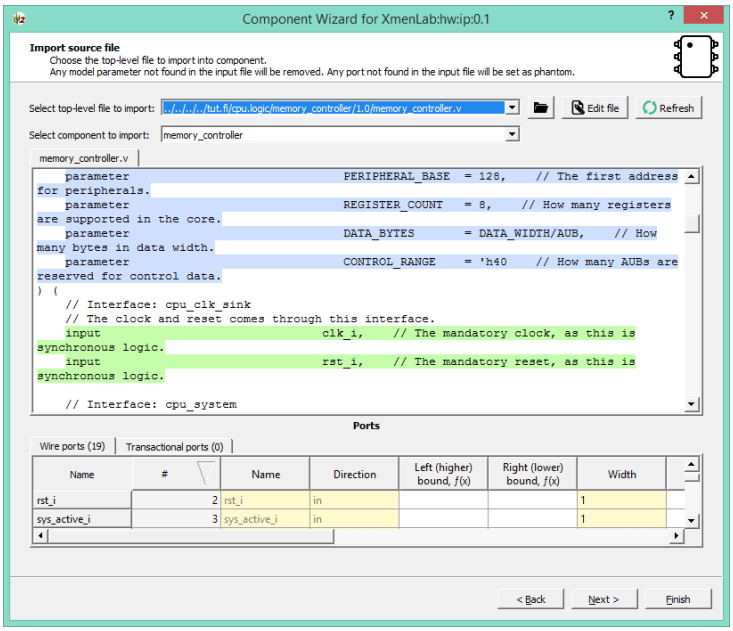
O componente criado pode ser visto na Figura 5 seguido de uma listagem de seus principais atributos. Na Figura 6 é mostrada novamente a interface, com um sumário dos documentos gerados no padrão IP-XACT, segundo esquemas XML. Nesse ponto, é possível a geração da HDL que descreve a interface do componente.

Figura 3 – Editor grafico de componente



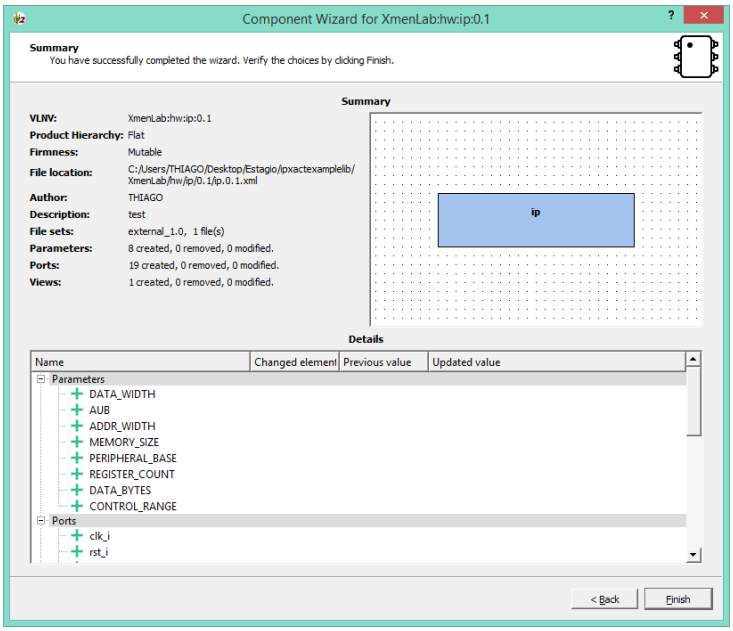
Fonte: O autor.

Figura 4 – Adição e interpretação de arquivos HDL.



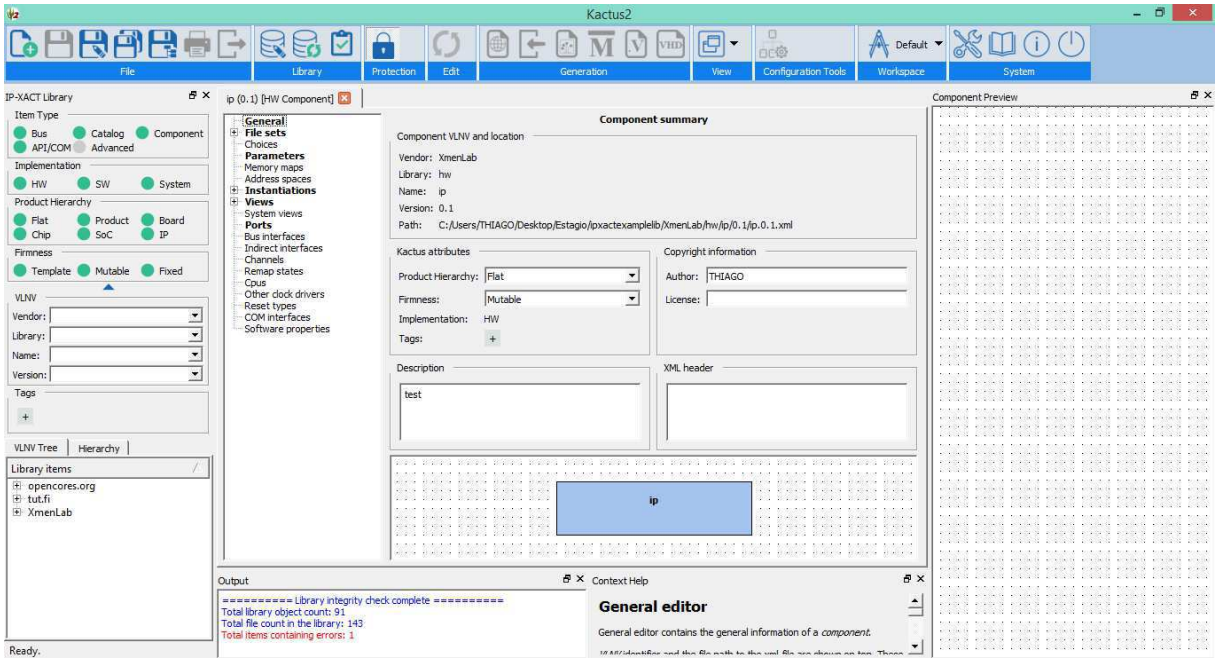
Fonte: O autor.

Figura 5 – Listagem de parâmetros e pinos do componente criado.



Fonte: O autor.

Figura 6 – Componente criado, com sua interface, parâmetros e pinos.

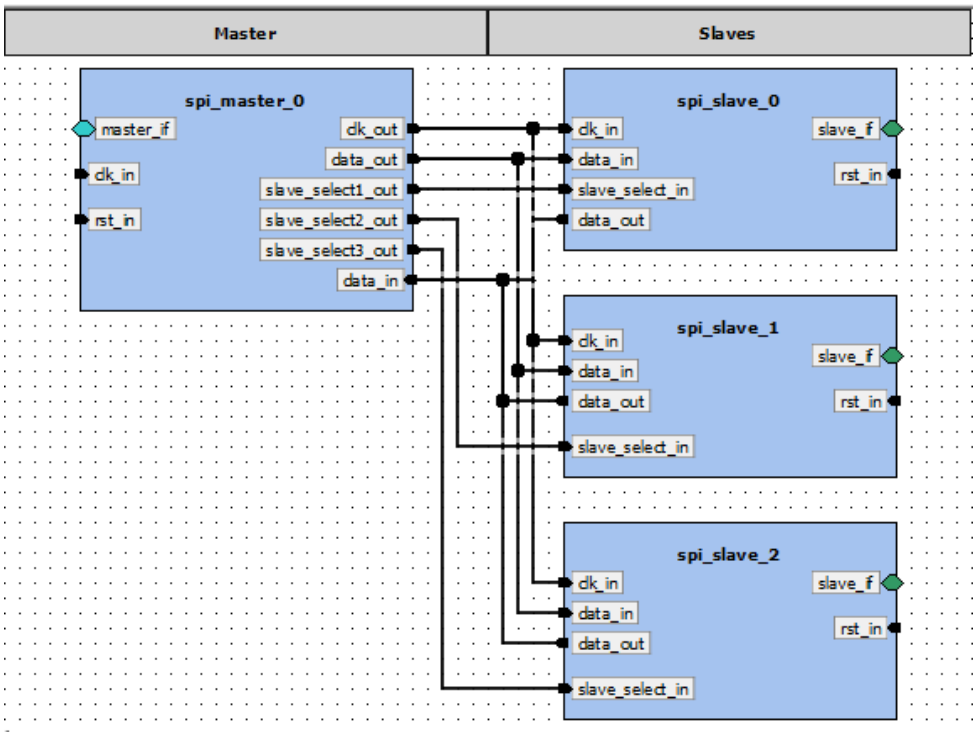


Fonte: O autor.

3.0.1 Protocolo SPI

Seguindo o procedimento de geração de componentes, foi feita a comunicação por protocolo SPI (*Serial Peripheral Interface*) entre um bloco mestre e três escravos em um mesmo *design*. O esquema está ilustrado na Figura 7 e o verilog gerado encontra-se em anexo.

Figura 7 – Protocolo de comunicação SPI.



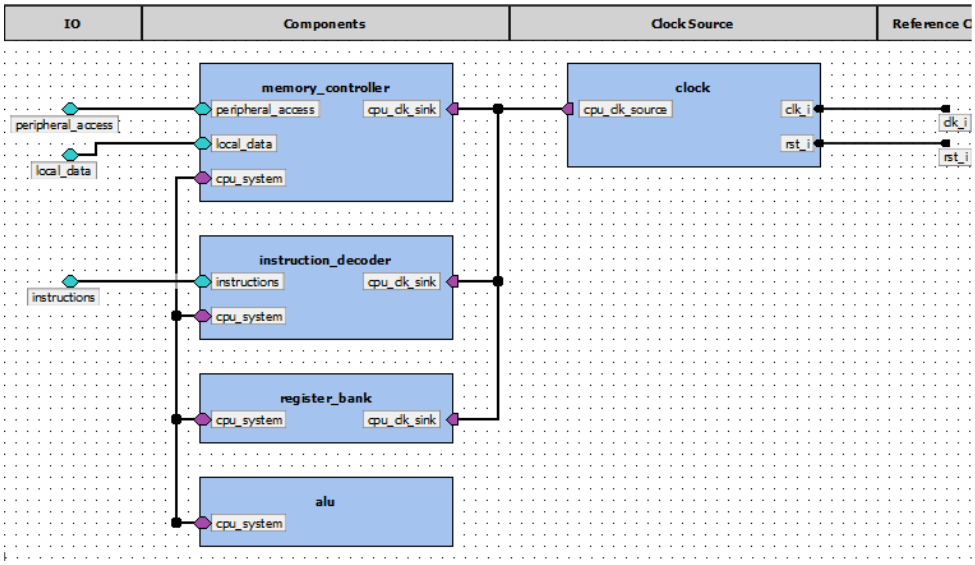
Fonte: O autor.

3.0.2 CPU

Também foi realizada a integração dos componentes de uma CPU básica, tais como decodificador de instrução, ALU, controlador de memória e banco de registradores. É ilustrado na Figura 8 a interconexão entre os componentes e o varilog da integração encontra-se em anexo.

Nesse caso não foi estabelecido um protoloco de comunicação tal como a seção anterior, sendo assim, a definição do barramento foi feita segundo o *handshake* dos componentes.

Figura 8 – *design* de uma CPU básica.



Fonte: O autor.

4 Conclusões

A realização deste trabalho constitui uma forma de reuso dos IPs desenvolvidos através do padrão IP-XACT, pelo uso da ferramenta Kactus2. A metodologia de projeto com a ferramenta pode ser incorporada ao fluxo de desenvolvimento de IP no laboratório XMEN de modo a otimizar a etapa de integração impactando positivamente o tempo de projeto e a integração de blocos desenvolvidos internamente ou adquiridos de fornecedores, uma vez que se trata de um padrão adotado pela indústria.

Seguindo os passos de modelagem descritos neste relatório é possível, de posse dos esquemas XML que descrevem os barramentos e interfaces dos blocos, a descrição da HDL em componentes IP-XACT. Cada implementação realizada pode ser sistematicamente guardada, sendo facilmente adaptável a um novo projeto quando necessária.

5 Anexos

5.1 Anexo A: Código verilog da interface do componente gerado no capítulo 3

```
1
// ------------------------------------------------------------------------------

3 // File          : .v
// Creation date : 06.12.2019
5 // Creation time : 12:02:23
// Description   : test
7 // Created by    :
// Tool : Kactus2 3.8.0 64-bit
9 // Plugin : Verilog generator 2.2
// This file was generated based on IP-XACT component XmenLab:hw:ip:0.1
11 // whose XML file is C:/Users/THIAGO/Desktop/Estagio/ipxactexamplelib/
   XmenLab/hw/ip/0.1/ip.0.1.xml
// ------------------------------------------------------------------------------

13
module memory_controller #(
15     parameter                               ADDR_WIDTH        = ADDR_WIDTH,
           // Width of the addresses.
       parameter                               AUB               = AUB,      //
          Addressable unit bits, size of byte.
17     parameter                               CONTROL_RANGE     = CONTROL_RANGE
          ,     // How many AUBs are reserved for control data.
       parameter                               DATA_BYTES        = DATA_BYTES,
           // How many bytes in data width.
```

```
19      parameter                                   DATA_WIDTH       = DATA_WIDTH,
            // Width for data in registers and instructions.
        parameter                                   MEMORY_SIZE      = MEMORY_SIZE,
            // How many bytes are in memory at total.
21      parameter                                   PERIPHERAL_BASE  =
            PERIPHERAL_BASE,     // The first address for peripherals.
        parameter                                   REGISTER_COUNT   =
            REGISTER_COUNT     // How many registers are supported in the core.
23  ) (
        // These ports are not in any interface
25      input                                       clk_i,     // The mandatory clock,
            as this is synchronous logic.
        input                   [DATA_WIDTH-1:0]    local_read_data,
27      input                   [DATA_WIDTH-1:0]    periph_data_i,
        input                                       periph_slave_rdy,
29      input                                       rst_i,     // The mandatory reset,
            as this is synchronous logic.
        input                                       sys_active_i,
31      input                   [ADDR_WIDTH-1:0]    sys_address_i,
        input                   [DATA_WIDTH-1:0]    sys_data_i,
33      input                                       sys_we_i,
        output                  [ADDR_WIDTH-1:0]    local_address_o,
35      output                  [DATA_WIDTH-1:0]    local_write_data,
        output                                      local_write_o,
37      output reg              [ADDR_WIDTH-1:0]    periph_address_o,
        output reg              [DATA_WIDTH-1:0]    periph_data_o,
39      output reg                                  periph_master_rdy,
        output reg                                  periph_we_o,
41      output                  [DATA_WIDTH-1:0]    sys_data_o,
        output                                      sys_rdy_o,
43      output                                      sys_read_rdy_o
    );
45
    // WARNING: EVERYTHING ON AND ABOVE THIS LINE MAY BE OVERWRITTEN BY KACTUS2
        !!!
47  endmodule
```

5.2 Anexo B: Código verilog da integração do protocolo SPI da Figura 7

```
//
    ------------------------------------------------------------------

// File          : spi.v
// Creation date : 01.12.2019
// Creation time : 13:03:14
// Description   :
// Created by    :
// Tool : Kactus2 3.8.0 64-bit
// Plugin : Verilog generator 2.2
// This file was generated based on IP-XACT component tut.fi:other.
    subsystem:spi_example:1.0
// whose XML file is C:/Users/THIAGO/Desktop/Estagio/ipxactexamplelib/tut.
    fi/other.subsystem/spi_example/1.0/spi_example.1.0.xml
//
    ------------------------------------------------------------------


module spi_example();

    // Ad-hoc wires:
    wire        spi_master_0_clk_out_to_spi_slave_0_clk_in;
    wire
        spi_master_0_slave_select1_out_to_spi_slave_0_slave_select_in;
    wire        spi_master_0_data_out_to_spi_slave_0_data_in;
    wire        spi_master_0_data_out_to_spi_slave_1_data_in;
    wire        spi_master_0_data_in_to_spi_slave_0_data_out;
    wire        spi_slave_1_clk_in_to_spi_master_0_clk_out;
    wire        spi_master_0_data_out_to_spi_slave_2_data_in;
    wire        spi_master_0_data_in_to_spi_slave_1_data_out;
    wire        spi_master_0_data_in_to_spi_slave_2_data_out;
    wire        spi_slave_2_clk_in_to_spi_master_0_clk_out;
    wire
        spi_slave_1_slave_select_in_to_spi_master_0_slave_select2_out;
    wire
        spi_slave_2_slave_select_in_to_spi_master_0_slave_select3_out;

    // spi_master_0 port wires:
    wire        spi_master_0_clk_out;
    wire        spi_master_0_data_in;
```

```
32      wire        spi_master_0_data_out;
        wire        spi_master_0_slave_select1_out;
34      wire        spi_master_0_slave_select2_out;
        wire        spi_master_0_slave_select3_out;
36      // spi_slave_0 port wires:
        wire        spi_slave_0_clk_in;
38      wire        spi_slave_0_data_in;
        wire        spi_slave_0_data_out;
40      wire        spi_slave_0_slave_select_in;
        // spi_slave_1 port wires:
42      wire        spi_slave_1_clk_in;
        wire        spi_slave_1_data_in;
44      wire        spi_slave_1_data_out;
        wire        spi_slave_1_slave_select_in;
46      // spi_slave_2 port wires:
        wire        spi_slave_2_clk_in;
48      wire        spi_slave_2_data_in;
        wire        spi_slave_2_data_out;
50      wire        spi_slave_2_slave_select_in;

52      // spi_master_0 assignments:
        assign spi_master_0_clk_out_to_spi_slave_0_clk_in =
           spi_master_0_clk_out;
54      assign spi_slave_1_clk_in_to_spi_master_0_clk_out =
           spi_master_0_clk_out;
        assign spi_slave_2_clk_in_to_spi_master_0_clk_out =
           spi_master_0_clk_out;
56      assign spi_master_0_data_in =
           spi_master_0_data_in_to_spi_slave_0_data_out;
        assign spi_master_0_data_in =
           spi_master_0_data_in_to_spi_slave_1_data_out;
58      assign spi_master_0_data_in =
           spi_master_0_data_in_to_spi_slave_2_data_out;
        assign spi_master_0_data_out_to_spi_slave_0_data_in =
           spi_master_0_data_out;
60      assign spi_master_0_data_out_to_spi_slave_1_data_in =
           spi_master_0_data_out;
        assign spi_master_0_data_out_to_spi_slave_2_data_in =
           spi_master_0_data_out;
62      assign spi_master_0_slave_select1_out_to_spi_slave_0_slave_select_in =
           spi_master_0_slave_select1_out;
        assign spi_slave_1_slave_select_in_to_spi_master_0_slave_select2_out =
           spi_master_0_slave_select2_out;
64      assign spi_slave_2_slave_select_in_to_spi_master_0_slave_select3_out =
           spi_master_0_slave_select3_out;
```

```
    // spi_slave_0 assignments:
66  assign spi_slave_0_clk_in = spi_master_0_clk_out_to_spi_slave_0_clk_in;
    assign spi_slave_0_data_in =
       spi_master_0_data_out_to_spi_slave_0_data_in;
68  assign spi_master_0_data_in_to_spi_slave_0_data_out =
       spi_slave_0_data_out;
    assign spi_slave_0_slave_select_in =
       spi_master_0_slave_select1_out_to_spi_slave_0_slave_select_in;
70  // spi_slave_1 assignments:
    assign spi_slave_1_clk_in = spi_slave_1_clk_in_to_spi_master_0_clk_out;
72  assign spi_slave_1_data_in =
       spi_master_0_data_out_to_spi_slave_1_data_in;
    assign spi_master_0_data_in_to_spi_slave_1_data_out =
       spi_slave_1_data_out;
74  assign spi_slave_1_slave_select_in =
       spi_slave_1_slave_select_in_to_spi_master_0_slave_select2_out;
    // spi_slave_2 assignments:
76  assign spi_slave_2_clk_in = spi_slave_2_clk_in_to_spi_master_0_clk_out;
    assign spi_slave_2_data_in =
       spi_master_0_data_out_to_spi_slave_2_data_in;
78  assign spi_master_0_data_in_to_spi_slave_2_data_out =
       spi_slave_2_data_out;
    assign spi_slave_2_slave_select_in =
       spi_slave_2_slave_select_in_to_spi_master_0_slave_select3_out;
80
    // IP-XACT VLNV: tut.fi:communication.template:spi_master:1.0
82  spi_master spi_master_0(
        // Interface: master_if
84      .data_in              (spi_master_0_data_in),
        .clk_out              (spi_master_0_clk_out),
86      .data_out             (spi_master_0_data_out),
        .slave_select1_out    (spi_master_0_slave_select1_out),
88      .slave_select2_out    (spi_master_0_slave_select2_out),
        .slave_select3_out    (spi_master_0_slave_select3_out),
90      // These ports are not in any interface
        .clk_in               (),
92      .rst_in               ());

94  // IP-XACT VLNV: tut.fi:communication.template:spi_slave:1.0
    spi_slave #(
96      .SLAVE_ID             (0))
    spi_slave_0(
98      // Interface: slave_if
        .clk_in               (spi_slave_0_clk_in),
100     .data_in              (spi_slave_0_data_in),
```

```
        .slave_select_in     (spi_slave_0_slave_select_in),
        .data_out            (spi_slave_0_data_out),
        // These ports are not in any interface
        .rst_in              ());

    // IP-XACT VLNV: tut.fi:communication.template:spi_slave:1.0
    spi_slave #(
        .SLAVE_ID            (0))
    spi_slave_1(
        // Interface: slave_if
        .clk_in              (spi_slave_1_clk_in),
        .data_in             (spi_slave_1_data_in),
        .slave_select_in     (spi_slave_1_slave_select_in),
        .data_out            (spi_slave_1_data_out),
        // These ports are not in any interface
        .rst_in              ());

    // IP-XACT VLNV: tut.fi:communication.template:spi_slave:1.0
    spi_slave #(
        .SLAVE_ID            (0))
    spi_slave_2(
        // Interface: slave_if
        .clk_in              (spi_slave_2_clk_in),
        .data_in             (spi_slave_2_data_in),
        .slave_select_in     (spi_slave_2_slave_select_in),
        .data_out            (spi_slave_2_data_out),
        // These ports are not in any interface
        .rst_in              ());


endmodule
```

5.3 Anexo B: Código verilog da integração da CPU da Figura 8

```
1 //
    ------------------------------------------------------------------------------

  // File          : cpu.v
3 // Creation date : 28.11.2019
  // Creation time : 04:47:51
5 // Description   : Test arrangement for the example CPU with data memory,
    instuction memory, clock source, and SPI slave.
  // Created by    :
7 // Tool : Kactus2 3.8.0 64-bit
  // Plugin : Verilog generator 2.2
9 // This file was generated based on IP-XACT component tut.fi:cpu.subsystem:
    core_example:1.0
  // whose XML file is C:/Users/THIAGO/Desktop/Estagio/ipxactexamplelib/tut.
    fi/cpu.subsystem/core_example/1.0/core_example.1.0.xml
11 //
    ------------------------------------------------------------------------------


13 module core_example #(
      parameter                                ADDR_WIDTH        = 9,     //
         Width of the addresses.
15    parameter                                DATA_WIDTH        = 32,    //
         Width for data in registers and instructions.
      parameter                                INSTRUCTION_ADDRESS_WIDTH = 8,
           // Width of an instruction address.
17    parameter                                INSTRUCTION_WIDTH = 28,    //
         Total width of an instruction
      parameter                                OP_CODE_WIDTH     = 4,     // Bits
          reserved for operation identifiers.
19    parameter                                PERIPHERAL_BASE   = 128,    //
         The first address for peripherals.
      parameter                                REGISTER_COUNT    = 8,     // How
         many registers are supported in the core.
21    parameter                                REGISTER_ID_WIDTH = 3,     //
         Bits reserved for identification a single register.
      parameter                                SUPPORTED_MEMORY = 512     // How
          much the system supports memory in total.
23 ) (
      // Interface: instructions
```

```
25      input                   [27:0]          instruction_feed,
        output                  [7:0]           iaddr_o,
27
        // Interface: local_data
29      input                   [31:0]          local_read_data,
        output                  [8:0]           local_address_o,
31      output                  [31:0]          local_write_data,
        output                                  local_write_o,
33
        // Interface: peripheral_access
35      input                   [31:0]          mem_data_i,
        input                                   mem_slave_rdy,
37      output                  [8:0]           mem_address_o,
        output                  [31:0]          mem_data_o,
39      output                                  mem_master_rdy,
        output                                  mem_we_o,
41
        // These ports are not in any interface
43      input                                   clk_i,     // The mandatory clock,
           as this is synchronous logic.
        input                                   rst_i     // The mandatory reset, as
            this is synchronous logic.
45  );

47      // memory_controller_cpu_system_to_alu_cpu_system wires:
        wire [31:0] memory_controller_cpu_system_to_alu_cpu_system_address;
49      wire        memory_controller_cpu_system_to_alu_cpu_system_alu_active;
        wire [2:0] memory_controller_cpu_system_to_alu_cpu_system_alu_operation
           ;
51      wire [31:0] memory_controller_cpu_system_to_alu_cpu_system_alu_result;
        wire [31:0] memory_controller_cpu_system_to_alu_cpu_system_alu_status;
53      wire [2:0]
           memory_controller_cpu_system_to_alu_cpu_system_choose_register_1;
        wire [2:0]
           memory_controller_cpu_system_to_alu_cpu_system_choose_register_2;
55      wire [31:0] memory_controller_cpu_system_to_alu_cpu_system_load_value;
        wire        memory_controller_cpu_system_to_alu_cpu_system_mem_active;
57      wire        memory_controller_cpu_system_to_alu_cpu_system_mem_rdy;
        wire        memory_controller_cpu_system_to_alu_cpu_system_mem_read_rdy;
59      wire        memory_controller_cpu_system_to_alu_cpu_system_mem_we;
        wire
           memory_controller_cpu_system_to_alu_cpu_system_register_active;
61      wire [31:0]
           memory_controller_cpu_system_to_alu_cpu_system_register_input;
```

```
    wire [31:0]
        memory_controller_cpu_system_to_alu_cpu_system_register_output_1;
63  wire [31:0]
        memory_controller_cpu_system_to_alu_cpu_system_register_output_2;
    // clock_cpu_clk_source_to_register_bank_cpu_clk_sink wires:
65  wire        clock_cpu_clk_source_to_register_bank_cpu_clk_sink_clk;
    wire        clock_cpu_clk_source_to_register_bank_cpu_clk_sink_rst;
67  // memory_controller_peripheral_access_to_peripheral_access wires:
    wire [8:0]
        memory_controller_peripheral_access_to_peripheral_access_address;
69  wire [31:0]
        memory_controller_peripheral_access_to_peripheral_access_data_ms;
    wire [31:0]
        memory_controller_peripheral_access_to_peripheral_access_data_sm;
71  wire
        memory_controller_peripheral_access_to_peripheral_access_master_rdy;
    wire
        memory_controller_peripheral_access_to_peripheral_access_slave_rdy;
73  wire        memory_controller_peripheral_access_to_peripheral_access_we;
    // instruction_decoder_instruction_feed_to_instructions wires:
75  wire [7:0] instruction_decoder_instruction_feed_to_instructions_address
        ;
    wire [27:0]
        instruction_decoder_instruction_feed_to_instructions_read_data;
77  // memory_controller_local_data_to_local_data wires:
    wire [8:0] memory_controller_local_data_to_local_data_address;
79  wire [31:0] memory_controller_local_data_to_local_data_read_data;
    wire        memory_controller_local_data_to_local_data_write;
81  wire [31:0] memory_controller_local_data_to_local_data_write_data;

83  // Ad-hoc wires:
    wire        clock_clk_i_to_clk_i;
85  wire        clock_rst_i_to_rst_i;

87  // alu port wires:
    wire [2:0] alu_alu_op_i;
89  wire [31:0] alu_alu_result_o;
    wire [31:0] alu_alu_status_o;
91  wire [31:0] alu_register_value_i1;
    wire [31:0] alu_register_value_i2;
93  // clock port wires:
    wire        clock_clk_i;
95  wire        clock_clk_o;
    wire        clock_rst_i;
97  wire        clock_rst_o;
```

```
    // instruction_decoder port wires:
99  wire        instruction_decoder_alu_active_o;
    wire [2:0]  instruction_decoder_alu_op_o;
101 wire [31:0] instruction_decoder_alu_status_i;
    wire [2:0]  instruction_decoder_choose_reg1_o;
103 wire [2:0]  instruction_decoder_choose_reg2_o;
    wire        instruction_decoder_clk_i;
105 wire [7:0]  instruction_decoder_iaddr_o;
    wire [27:0] instruction_decoder_instruction_feed;
107 wire [31:0] instruction_decoder_load_value_i;
    wire        instruction_decoder_mem_active_o;
109 wire        instruction_decoder_mem_rdy_i;
    wire        instruction_decoder_register_active_o;
111 wire [31:0] instruction_decoder_register_value_o;
    wire        instruction_decoder_rst_i;
113 wire        instruction_decoder_we_o;
    // memory_controller port wires:
115 wire        memory_controller_clk_i;
    wire [8:0]  memory_controller_local_address_o;
117 wire [31:0] memory_controller_local_read_data;
    wire [31:0] memory_controller_local_write_data;
119 wire        memory_controller_local_write_o;
    wire [8:0]  memory_controller_periph_address_o;
121 wire [31:0] memory_controller_periph_data_i;
    wire [31:0] memory_controller_periph_data_o;
123 wire        memory_controller_periph_master_rdy;
    wire        memory_controller_periph_slave_rdy;
125 wire        memory_controller_periph_we_o;
    wire        memory_controller_rst_i;
127 wire        memory_controller_sys_active_i;
    wire [8:0]  memory_controller_sys_address_i;
129 wire [31:0] memory_controller_sys_data_i;
    wire [31:0] memory_controller_sys_data_o;
131 wire        memory_controller_sys_rdy_o;
    wire        memory_controller_sys_read_rdy_o;
133 wire        memory_controller_sys_we_i;
    // register_bank port wires:
135 wire        register_bank_alu_active_i;
    wire [31:0] register_bank_alu_result_i;
137 wire [2:0]  register_bank_choose_register_i1;
    wire [2:0]  register_bank_choose_register_i2;
139 wire        register_bank_clk_i;
    wire [31:0] register_bank_load_value_i;
141 wire        register_bank_mem_read_rdy_i;
    wire        register_bank_register_active_i;
```

```
143     wire [31:0] register_bank_register_input;
        wire [31:0] register_bank_register_output1;
145     wire [31:0] register_bank_register_output2;
        wire        register_bank_rst_i;
147
        // Assignments for the ports of the encompassing component:
149     assign clock_clk_i_to_clk_i = clk_i;
        assign iaddr_o =
           instruction_decoder_instruction_feed_to_instructions_address;
151     assign instruction_decoder_instruction_feed_to_instructions_read_data =
            instruction_feed;
        assign local_address_o =
           memory_controller_local_data_to_local_data_address;
153     assign memory_controller_local_data_to_local_data_read_data =
           local_read_data;
        assign local_write_data =
           memory_controller_local_data_to_local_data_write_data;
155     assign local_write_o = memory_controller_local_data_to_local_data_write
           ;
        assign mem_address_o =
           memory_controller_peripheral_access_to_peripheral_access_address;
157     assign memory_controller_peripheral_access_to_peripheral_access_data_sm
            = mem_data_i;
        assign mem_data_o =
           memory_controller_peripheral_access_to_peripheral_access_data_ms;
159     assign mem_master_rdy =
           memory_controller_peripheral_access_to_peripheral_access_master_rdy;
        assign
           memory_controller_peripheral_access_to_peripheral_access_slave_rdy =
            mem_slave_rdy;
161     assign mem_we_o =
           memory_controller_peripheral_access_to_peripheral_access_we;
        assign clock_rst_i_to_rst_i = rst_i;
163
        // alu assignments:
165     assign alu_alu_op_i =
           memory_controller_cpu_system_to_alu_cpu_system_alu_operation;
        assign memory_controller_cpu_system_to_alu_cpu_system_alu_result =
           alu_alu_result_o;
167     assign memory_controller_cpu_system_to_alu_cpu_system_alu_status =
           alu_alu_status_o;
        assign alu_register_value_i1 =
           memory_controller_cpu_system_to_alu_cpu_system_register_output_1;
169     assign alu_register_value_i2 =
           memory_controller_cpu_system_to_alu_cpu_system_register_output_2;
```

```
    // clock assignments:
171 assign clock_clk_i = clock_clk_i_to_clk_i;
    assign clock_cpu_clk_source_to_register_bank_cpu_clk_sink_clk =
        clock_clk_o;
173 assign clock_rst_i = clock_rst_i_to_rst_i;
    assign clock_cpu_clk_source_to_register_bank_cpu_clk_sink_rst =
        clock_rst_o;
175 // instruction_decoder assignments:
    assign memory_controller_cpu_system_to_alu_cpu_system_alu_active =
        instruction_decoder_alu_active_o;
177 assign memory_controller_cpu_system_to_alu_cpu_system_alu_operation =
        instruction_decoder_alu_op_o;
    assign instruction_decoder_alu_status_i =
        memory_controller_cpu_system_to_alu_cpu_system_alu_status;
179 assign memory_controller_cpu_system_to_alu_cpu_system_choose_register_1
         = instruction_decoder_choose_reg1_o;
    assign memory_controller_cpu_system_to_alu_cpu_system_choose_register_2
         = instruction_decoder_choose_reg2_o;
181 assign instruction_decoder_clk_i =
        clock_cpu_clk_source_to_register_bank_cpu_clk_sink_clk;
    assign instruction_decoder_instruction_feed_to_instructions_address =
        instruction_decoder_iaddr_o;
183 assign instruction_decoder_instruction_feed =
        instruction_decoder_instruction_feed_to_instructions_read_data;
    assign instruction_decoder_load_value_i =
        memory_controller_cpu_system_to_alu_cpu_system_load_value;
185 assign memory_controller_cpu_system_to_alu_cpu_system_mem_active =
        instruction_decoder_mem_active_o;
    assign instruction_decoder_mem_rdy_i =
        memory_controller_cpu_system_to_alu_cpu_system_mem_rdy;
187 assign memory_controller_cpu_system_to_alu_cpu_system_register_active =
         instruction_decoder_register_active_o;
    assign memory_controller_cpu_system_to_alu_cpu_system_register_input =
        instruction_decoder_register_value_o;
189 assign instruction_decoder_rst_i =
        clock_cpu_clk_source_to_register_bank_cpu_clk_sink_rst;
    assign memory_controller_cpu_system_to_alu_cpu_system_mem_we =
        instruction_decoder_we_o;
191 // memory_controller assignments:
    assign memory_controller_clk_i =
        clock_cpu_clk_source_to_register_bank_cpu_clk_sink_clk;
193 assign memory_controller_local_data_to_local_data_address =
        memory_controller_local_address_o;
    assign memory_controller_local_read_data =
        memory_controller_local_data_to_local_data_read_data;
```

```
195     assign memory_controller_local_data_to_local_data_write_data =
            memory_controller_local_write_data;
        assign memory_controller_local_data_to_local_data_write =
            memory_controller_local_write_o;
197     assign memory_controller_peripheral_access_to_peripheral_access_address
             = memory_controller_periph_address_o;
        assign memory_controller_periph_data_i =
            memory_controller_peripheral_access_to_peripheral_access_data_sm;
199     assign memory_controller_peripheral_access_to_peripheral_access_data_ms
             = memory_controller_periph_data_o;
        assign
            memory_controller_peripheral_access_to_peripheral_access_master_rdy
            = memory_controller_periph_master_rdy;
201     assign memory_controller_periph_slave_rdy =
            memory_controller_peripheral_access_to_peripheral_access_slave_rdy;
        assign memory_controller_peripheral_access_to_peripheral_access_we =
            memory_controller_periph_we_o;
203     assign memory_controller_rst_i =
            clock_cpu_clk_source_to_register_bank_cpu_clk_sink_rst;
        assign memory_controller_sys_active_i =
            memory_controller_cpu_system_to_alu_cpu_system_mem_active;
205     assign memory_controller_sys_address_i =
            memory_controller_cpu_system_to_alu_cpu_system_address[8:0];
        assign memory_controller_sys_data_i =
            memory_controller_cpu_system_to_alu_cpu_system_register_output_1;
207     assign memory_controller_cpu_system_to_alu_cpu_system_load_value =
            memory_controller_sys_data_o;
        assign memory_controller_cpu_system_to_alu_cpu_system_mem_rdy =
            memory_controller_sys_rdy_o;
209     assign memory_controller_cpu_system_to_alu_cpu_system_mem_read_rdy =
            memory_controller_sys_read_rdy_o;
        assign memory_controller_sys_we_i =
            memory_controller_cpu_system_to_alu_cpu_system_mem_we;
211     // register_bank assignments:
        assign register_bank_alu_active_i =
            memory_controller_cpu_system_to_alu_cpu_system_alu_active;
213     assign register_bank_alu_result_i =
            memory_controller_cpu_system_to_alu_cpu_system_alu_result;
        assign register_bank_choose_register_i1 =
            memory_controller_cpu_system_to_alu_cpu_system_choose_register_1;
215     assign register_bank_choose_register_i2 =
            memory_controller_cpu_system_to_alu_cpu_system_choose_register_2;
        assign register_bank_clk_i =
            clock_cpu_clk_source_to_register_bank_cpu_clk_sink_clk;
```

```
217     assign register_bank_load_value_i =
            memory_controller_cpu_system_to_alu_cpu_system_load_value;
        assign register_bank_mem_read_rdy_i =
            memory_controller_cpu_system_to_alu_cpu_system_mem_read_rdy;
219     assign register_bank_register_active_i =
            memory_controller_cpu_system_to_alu_cpu_system_register_active;
        assign register_bank_register_input =
            memory_controller_cpu_system_to_alu_cpu_system_register_input;
221     assign memory_controller_cpu_system_to_alu_cpu_system_register_output_1
             = register_bank_register_output1;
        assign memory_controller_cpu_system_to_alu_cpu_system_address =
            register_bank_register_output2;
223     assign memory_controller_cpu_system_to_alu_cpu_system_register_output_2
             = register_bank_register_output2;
        assign register_bank_rst_i =
            clock_cpu_clk_source_to_register_bank_cpu_clk_sink_rst;
225
        // IP-XACT VLNV: tut.fi:cpu.logic:alu:1.0
227     alu #(
            .DATA_WIDTH             (32))
229     alu(
            // Interface: cpu_system
231         .alu_op_i               (alu_alu_op_i),
            .register_value_i1      (alu_register_value_i1),
233         .register_value_i2      (alu_register_value_i2),
            .alu_result_o           (alu_alu_result_o),
235         .alu_status_o           (alu_alu_status_o));

237     // IP-XACT VLNV: tut.fi:cpu.logic:clock:1.0
        clock clock(
239         // Interface: cpu_clk_source
            .clk_o                  (clock_clk_o),
241         .rst_o                  (clock_rst_o),
            // These ports are not in any interface
243         .clk_i                  (clock_clk_i),
            .rst_i                  (clock_rst_i));
245
        // IP-XACT VLNV: tut.fi:cpu.logic:instruction_decoder:1.0
247     instruction_decoder #(
            .REGISTER_ID_WIDTH      (3),
249         .INSTRUCTION_WIDTH      (28),
            .DATA_WIDTH             (32),
251         .INSTRUCTION_ADDRESS_WIDTH(8))
        instruction_decoder(
253         // Interface: cpu_clk_sink
```

```
        .clk_i                  (instruction_decoder_clk_i),
255     .rst_i                  (instruction_decoder_rst_i),
        // Interface: cpu_system
257     .alu_status_i           (instruction_decoder_alu_status_i),
        .load_value_i           (instruction_decoder_load_value_i),
259     .mem_rdy_i              (instruction_decoder_mem_rdy_i),
        .alu_active_o           (instruction_decoder_alu_active_o),
261     .alu_op_o               (instruction_decoder_alu_op_o),
        .choose_reg1_o          (instruction_decoder_choose_reg1_o),
263     .choose_reg2_o          (instruction_decoder_choose_reg2_o),
        .mem_active_o           (instruction_decoder_mem_active_o),
265     .register_active_o      (instruction_decoder_register_active_o),
        .register_value_o       (instruction_decoder_register_value_o),
267     .we_o                   (instruction_decoder_we_o),
        // Interface: instructions
269     .instruction_feed       (instruction_decoder_instruction_feed),
        .iaddr_o                (instruction_decoder_iaddr_o));
271
    // IP-XACT VLNV: tut.fi:cpu.logic:memory_controller:1.0
273 memory_controller #(
        .DATA_WIDTH             (32),
275     .ADDR_WIDTH             (9),
        .MEMORY_SIZE            (512),
277     .PERIPHERAL_BASE        (128),
        .REGISTER_COUNT         (8))
279 memory_controller(
        // Interface: cpu_clk_sink
281     .clk_i                  (memory_controller_clk_i),
        .rst_i                  (memory_controller_rst_i),
283     // Interface: cpu_system
        .sys_active_i           (memory_controller_sys_active_i),
285     .sys_address_i          (memory_controller_sys_address_i),
        .sys_data_i             (memory_controller_sys_data_i),
287     .sys_we_i               (memory_controller_sys_we_i),
        .sys_data_o             (memory_controller_sys_data_o),
289     .sys_rdy_o              (memory_controller_sys_rdy_o),
        .sys_read_rdy_o         (memory_controller_sys_read_rdy_o),
291     // Interface: local_data
        .local_read_data        (memory_controller_local_read_data),
293     .local_address_o        (memory_controller_local_address_o),
        .local_write_data       (memory_controller_local_write_data),
295     .local_write_o          (memory_controller_local_write_o),
        // Interface: peripheral_access
297     .periph_data_i          (memory_controller_periph_data_i),
        .periph_slave_rdy       (memory_controller_periph_slave_rdy),
```

```
299         .periph_address_o     (memory_controller_periph_address_o),
            .periph_data_o        (memory_controller_periph_data_o),
301         .periph_master_rdy    (memory_controller_periph_master_rdy),
            .periph_we_o          (memory_controller_periph_we_o));
303
    // IP-XACT VLNV: tut.fi:cpu.logic:register_bank:1.0
305     register_bank #(
            .DATA_WIDTH           (32),
307         .REGISTER_ID_WIDTH    (3),
            .REGISTER_COUNT       (8))
309     register_bank(
            // Interface: cpu_clk_sink
311         .clk_i                (register_bank_clk_i),
            .rst_i                (register_bank_rst_i),
313         // Interface: cpu_system
            .alu_active_i         (register_bank_alu_active_i),
315         .alu_result_i         (register_bank_alu_result_i),
            .choose_register_i1   (register_bank_choose_register_i1),
317         .choose_register_i2   (register_bank_choose_register_i2),
            .load_value_i         (register_bank_load_value_i),
319         .mem_read_rdy_i       (register_bank_mem_read_rdy_i),
            .register_active_i    (register_bank_register_active_i),
321         .register_input       (register_bank_register_input),
            .register_output1     (register_bank_register_output1),
323         .register_output2     (register_bank_register_output2));


325
endmodule
```

Referências

Accellera Systems Initiative Inc. *IP-XACT User Guide*. Accellera Systems Initiative, 8698 Elk Grove Blvd. Suite 1, Elk Grove, CA 95624, USA: 2018.

KRUIJTZER, Wido et al. *Industrial IP integration flows based on IP-XACT standards*. In: 2008 Design, Automation and Test in Europe. IEEE, 2008. p. 32-37.

LEENS, Frédéric. An introduction to I 2 C and SPI protocols. IEEE Instrumentation & Measurement Magazine, v. 12, n. 1, p. 8-13, 2009.

SIMM, Uwe et al. *Accellera Standards Technical Update*. In: 2015 Design and Verification Conference and Exhibition. 2015.