UNIVERSIDADE FEDERAL DO PARÁ
CAMPUS UNIVERSITÁRIO DE TUCURUÍ
FACULDADE DE ENGENHARIA ELÉTRICA

RICARDO FREITAS LISBOA

ENSINO DE AUTOMAÇÃO UTILIZANDO KIT FPGA DE2 MODELO 2C35 DA ALTERA

TUCURUÍ

2023

RICARDO FREITAS LISBOA

ENSINO DE AUTOMAÇÃO UTILIZANDO KIT FPGA DE2 MODELO 2C35 DA ALTERA

Trabalho de Conclusão de Curso, apresentado como requisito parcial para a obtenção de grau de Bacharel em Engenharia Elétrica, pela Universidade Federal do Pará.

Orientador:
Prof. Dr. André Felipe Souza da Cruz.

TUCURUÍ

2023

Dados Internacionais de Catalogação na Publicação (CIP) de acordo com ISBD
Sistema de Bibliotecas da Universidade Federal do Pará
Gerada automaticamente pelo módulo Ficat, mediante os dados fornecidos pelo(a) autor(a)

L769e LISBOA, RICARDO FREITAS.
ENSINO DE AUTOMAÇÃO UTILIZANDO KIT FPGA DE2 MODELO 2C35 DA ALTERA / RICARDO FREITAS LISBOA. — 2023.
115 f. : il. color.

Orientador(a): Prof. Dr. André Felipe Souza da Cruz
Trabalho de Conclusão de Curso (Graduação) - Universidade Federal do Pará, Campus Universitário de Tucuruí, Faculdade de Engenharia Elétrica, Tucuruí, 2023.

1. Ensino de Engenharia . 2. Sistemas digitais. 3. VHDL. 4. FPGA . I. Título.

CDD 620.007

RICARDO FREITAS LISBOA

ENSINO DE AUTOMAÇÃO UTILIZANDO KIT FPGA DE2 MODELO 2C35 DA ALTERA

Trabalho de Conclusão de Curso, apresentado como requisito parcial para a obtenção de grau de Bacharel em Engenharia Elétrica, pela Universidade Federal do Pará.

Data de aprovação: 06/07/2023

Banca Examinadora:

Prof. Dr. André Felipe Souza da Cruz
Orientador - FEE/CAMTUC/UFPA

Prof. Dr. Rafael Suzuki Bayma
Avaliador Interno - FEE/CAMTUC/UFPA

Prof. Dr. Washington Cesar B. Sousa
Avaliador Interno - FEE/CAMTUC/UFPA

TUCURUÍ

2023

MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL DO PARÁ
CAMPUS UNIVERSITARIO DE TUCURUI

ATA DE DEFESA DE TRABALHO DE CONCLUSÃO DE CURSO (TCC) Nº 9 / 2023 - CTUCU (11.16)

Nº do Protocolo: 23073.048854/2023-13

Tucuruí-PA, 06 de julho de 2023.

TÍTULO: ENSINO DE AUTOMAÇÃO UTILIZANDO KIT FPGA DE2 MODELO 2C35 DA ALTERA		
DISCENTE: RICARDO FREITAS LISBOA		
MATRÍCULA: 201733940038		
#	**BANCA EXAMINADORA**	**CONDIÇÃO**
1	*Prof. Dr. André Felipe Souza da Cruz*	Orientador
2	*Prof. Dr. Rafael Suzuki Bayma*	Membro
3	*Prof. Dr. Washington Cesar Braga de Sousa*	Membro

Data da Defesa: 06/07/2023	**Hora Início:** 9:05	**Hora Término:** 9:44

Trabalho Escrito (0 a 10 pontos por critério)	**Examinador 1**	**Examinador 2**	**Examinador 3**
Formatação	9,75	10,00	9,80
Linguagem (gramática e semântica)	9,50	9,50	9,50
Conteúdo técnico	9,00	9,50	9,50

Defesa Oral (0 a 10 pontos por critério)	**Examinador 1**	**Examinador 2**	**Examinador 3**
Sequência lógica de apresentação	9,50	10,00	10,00
Administração do tempo	10,00	10,00	10,00
Expressão oral	9,75	10,00	10,00
Domínio do tema	9,50	9,00	9,50

Média por examinador	9,57	9,71	9,76
Média Final	9,68		
Conceito Final	EXCELENTE		

(Assinado digitalmente em 08/07/2023 21:20)
ANDRE FELIPE SOUZA DA CRUZ
PROFESSOR DO MAGISTERIO SUPERIOR
CTUCU (11.16)
Matrícula: ###531#4

(Assinado digitalmente em 06/07/2023 13:46)
RAFAEL SUZUKI BAYMA
PROFESSOR DO MAGISTERIO SUPERIOR
NDAE (11.16.16)
Matrícula: ###466#0

(Assinado digitalmente em 08/07/2023 17:08)
WASHINGTON CESAR BRAGA DE SOUSA
PROFESSOR DO MAGISTERIO SUPERIOR
CTUCU (11.16)
Matrícula: ###185#4

Para verificar a autenticidade deste documento entre em **https://sipac.ufpa.br/public/documentos/index.jsp** informando seu número: **9**, ano: **2023**, tipo: **ATA DE DEFESA DE TRABALHO DE CONCLUSÃO DE CURSO (TCC)**, data de emissão: **06/07/2023** e o código de verificação: **6e7850b41d**

Dedico este trabalho a Deus por nortear minha vida. A meu pai Antônio, pelo suporte e criação. E finalmente aos meus amigos pela convivência, apoio e atenção nos momentos alegres e tristes.

AGRADECIMENTOS

Em primeiro lugar, a Deus, que fez com que meus objetivos fossem alcançados, durante todos os meus anos de estudos.

Gostaria de expressar minha profunda gratidão a cada um de vocês pelo apoio, orientação e amizade ao longo dos anos do nosso curso. Nossa jornada acadêmica foi repleta de desafios, mas também de muitas conquistas e aprendizados, e isso não teria sido possível sem a contribuição valiosa de cada um de vocês.

Ao Professor André, meu orientador de TCC, meu sincero agradecimento por sua dedicação e orientação ao longo do processo de pesquisa e desenvolvimento do meu trabalho. Suas sugestões, feedbacks e conhecimentos foram fundamentais para o sucesso do meu projeto.

Ao Professor Jefferson, que ministrou diversas disciplinas ao longo do curso, agradeço por compartilhar seu conhecimento e expertise conosco. Suas aulas foram enriquecedoras e contribuíram para o meu crescimento acadêmico.

Aos colegas de classe Mateus Gonzaga, Alexandre Montibeller, José Antônio Moraes, Victor Sérgio, Nílton Gomes, Walder Júnior e Geisielen Seth, agradeço pela amizade, colaboração e apoio mútuo ao longo desses anos. Juntos, enfrentamos desafios acadêmicos e compartilhamos experiências que certamente levarei para a vida toda.

Ao longo do curso, pude aprender e desenvolver habilidades técnicas, de pesquisa, trabalho em equipe e superação de desafios. As trocas de conhecimento e experiências com cada um de vocês foram enriquecedoras e contribuíram para o meu crescimento pessoal e profissional.

Novamente, a todos vocês, meu sincero agradecimento por fazerem parte da minha jornada acadêmica. Sou grato pela oportunidade de ter estudado e aprendido com pessoas tão talentosas e dedicadas. O conhecimento e as experiências adquiridas durante esse período serão para sempre lembrados e valorizados.

"Enquanto eu estiver vivo, minhas possibilidades são infinitas!"
(Eiichiro Oda, mangaká criador de One Piece).

RESUMO

Neste trabalho é apresentado um conjunto de roteiros práticos para ensino de Engenharia em Automação e Controle, utilizando Kits Dispositivos Lógicos Programáveis (PLD - *Programable Logic Devices*), do tipo FPGA - *Field Programmable Gate Array.* A proposta metodológica se baseia no uso de ferramentas de simulação e design assistido por computador para ensino de automação no curso de Engenharia Elétrica, envolvendo projetos interdisciplinares e práticas de aprendizagem ativa e colaborativa. Primeiramente, é apresentado um estudo de conceitos básicos da linguagem VHDL (VHSIC *Hardware Description Language*, onde VHSIC – *Very High Speed Integrated Circuit*) e das características básicas dos dispositivos FPGA. Essa abordagem visa proporcionar aos estudantes uma compreensão teórica sólida e a capacidade de projetar e implementar circuitos digitais utilizando essa linguagem e tecnologia. Os roteiros práticos fornecidos no trabalho são uma ferramenta valiosa para auxiliar no ensino efetivo e prático da Engenharia em Automação e Controle, capacitando os alunos a adquirir habilidades relevantes para a área e promovendo uma aprendizagem mais envolvente e interativa.

Palavras-chave: Ensino de engenharia; Sistemas digitais; VHDL; FGPA.

ABSTRACT

This work presents a set of practical guides for teaching Automation and Control Engineering using Programmable Logic Devices (PLDs), specifically Field Programmable Gate Arrays (FPGAs). The methodological proposal is based on the use of simulation tools and computer-aided design for teaching automation in the Electrical Engineering course, involving interdisciplinary projects and active and collaborative learning practices. Firstly, a study of the basic concepts of the VHDL language (VHSIC Hardware Description Language, where VHSIC stands for Very High Speed Integrated Circuit) and the basic characteristics of FPGA devices is presented. This approach aims to provide students with a solid theoretical understanding and the ability to design and implement digital circuits using this language and technology. The practical guides provided in this work are a valuable tool to assist in the effective and practical teaching of Automation and Control Engineering, empowering students to acquire the necessary skills in this field.

Keywords: Engineering education; Digital systems; VHDL; FPGA.

LISTA DE ILUSTRAÇÕES

Figura 1 – Arquitetura básica de CPLDs 21
Figura 2 – Representação dos elementos básicos de um FPGA 22
Figura 3 – Blocos lógicos configuráveis básicos (CLBs) dentro de interconexões programáveis globais em linha/coluna. 23
Figura 4 – Modulo Lógico. 24
Figura 5 – Sintaxe do algoritmo em VHDL utilizando a biblioteca IEEE 27
Figura 6 – Comparador de 1 bit simplificado. 35
Figura 7 – Estrutura física desenvolvida no *Software* Quartus II 35
Figura 8 – ALTERA DE2-2C35 38
Figura 9 – Diagrama de Blocos da Placa DE2-2C35 39
Figura 10 – Tela inicial do QuartusII 41
Figura 11 – Nomeação do projeto e local onde o mesmo será salvo 41
Figura 12 – Seleção da Familia Cyclone II 42
Figura 13 – Seleção do FPGA 42
Figura 14 – Diretório do projeto 42
Figura 15 – Criação de Arquio VHDL 43
Figura 16 – Código VHDL do exemplo 2.2.3 43
Figura 17 – Iniciar Compilação do arquivo 44
Figura 18 – Ambiente Virtual Waveform 44
Figura 19 – Insert Node or Bus 45
Figura 20 – Lista de E/S 45
Figura 21 – Ambiente de simulação já com as E/S 46
Figura 22 – Definição dos *clocks* de Entrada 46
Figura 23 – Entradas com *clock* definidos 47
Figura 24 – Execução da simulação 47
Figura 25 – Simulação do comparador de 1 bit simplificado 47
Figura 26 – Ícone para acessar o ambiente de mapeamento de pinos 48
Figura 27 – Ambiente de mapeamento de pinos 48
Figura 28 – Mapeamento de pinos do comparador de 1 bit 48
Figura 29 – Icpne para acesso ao ambiente Programmer 49
Figura 30 – Conexão do FPGA ao computador com o Quartus II 49
Figura 31 – Verificações finais do ambiente de programação 49
Figura 32 – Acompanhamento do processo de programação 50
Figura 33 – Teste de hardware 50
Figura 34 – Multiplexador 2 para 1 53
Figura 35 – Multiplexador 3 para 1 54

Figura 36 – Decodificador de 7 segmentos 54
Figura 37 – Representação do flip-fLop tipo D 55
Figura 38 – Registrador paralelo de 4 bits 56
Figura 39 – Registrador de deslocamento de 4 bits 56
Figura 40 – Simbologia de um contador 57
Figura 41 – Diagrama lógico simplificado para um relógio digital de 12 horas 58
Figura 42 – Pré-simulação acionamento de LEDs 60
Figura 43 – Simulação acionamento de LEDS 60
Figura 44 – Teste de Hardware para o acionamento dos LEDs de saída 61
Figura 45 – Simulação do algoritmo de portas lógicas básicas 63
Figura 46 – Teste de *hardware* para duas entradas 63
Figura 47 – Simulação Multiplexador 3 para 1 65
Figura 48 – Multiplexador 3 para 1 66
Figura 49 – Simulação decodificador para 7 segmentos 67
Figura 50 – Teste de *hardware* do decodificador de 2 bits para 7 segmentos. 68
Figura 51 – Simulação registrador de deslocamento 70
Figura 52 – Teste de *hardware* do Registrador de deslocamento de 4 bits 71
Figura 53 – Simulação contador 74
Figura 54 – Teste de *hardware* do contador 74
Figura 55 – Adicionar/ remover arquivo 77
Figura 56 – Hierarquia de projeto 78
Figura 57 – Teste de Hardware do relógio digital 78

LISTA DE QUADROS

Quadro 1 – Palavras reservadas em VHDL 26
Quadro 2 – Sintaxe de biblioteca 28
Quadro 3 – Pacotes mais utilizados (Biblioteca IEEE) 29
Quadro 4 – Sintaxe de entidade 30
Quadro 5 – Tipos de portas mais utilizadas 30
Quadro 6 – Sintaxe de arquitetura 31
Quadro 7 – Sintaxe de componentes 32
Quadro 8 – Sintaxe de sinais 32
Quadro 9 – Sintaxe atribuições 32
Quadro 10 – Sintaxe de Estruturas de Controle em VHDL 33
Quadro 11 – Sintaxe de processos 34
Quadro 12 – Sintaxe configuração 34
Quadro 13 – Representação Comportamental do comparador de 1 bit 36
Quadro 14 – Representação Estrutural do comparador de 1 bit 36

Quadro 15 – Código para acionamento dos LEDs da placa DE2-C235 59
Quadro 16 – Mapeamento dos pinos para a conexão das chaves (SW) aos LEDs . . 61
Quadro 17 – Porta AND, OR e XOR em VHDL 62
Quadro 18 – Mapeamento dos pinos para as portas AND, OR e XOR 63
Quadro 19 – Entidade do elemento Mux 3:1 64
Quadro 20 – Arquitetura do elemento Mux 3:1 64
Quadro 21 – Conexão entre os sinais VHDL e os pinos do FPGA para o Mux 3-to-1 65
Quadro 22 – Entidade decodificador 2 bits para 7 segmentos 67
Quadro 23 – Conexão entre os sinais VHDL e os pinos do FPGA para o Mux 3-to-1 68
Quadro 24 – Código para implementação de FF-D 69
Quadro 25 – Descrição da entidade do Registrador de deslocamento de 4 bits . . . 69
Quadro 26 – Descrição da arquitetura do Registrador de deslocamento de 4 bits . . 70
Quadro 27 – Conexão entre os sinais VHDL e os pinos do FPGA para o Registrador 71
Quadro 28 – Entidade de um Contador descrito em VHDL 72
Quadro 29 – Arquitetura de um Contador descrito em VHDL 73
Quadro 30 – Conexão entre os sinais VHDL e os pinos do FPGA para o Contador . 74
Quadro 31 – Entidade do relógio digital 75
Quadro 32 – Arquitetura do relógio digital 76

LISTA DE ABREVIATURAS E SIGLAS

CPU	*Central Processing Unit*
PLD	*Programmable Logic Device*
SPLD	*Simple Programmable Logic Device*
CPLD	*Complex Programmable Logic Device*
ASIC	*Application Specific Integrated Circuit*
FPGA	*Field Programmable Gate Arrays*
VHDL	*VHSIC Hardware Description Language*
VHSIC	*Very High Speed Integrated Circuit*
HDL	*Hardware Description Language*

SUMÁRIO

1 INTRODUÇÃO 17
1.1 Justificativa 18
1.2 Objetivos 18
1.3 Estrutura da Monografia 19

2 DISPOSITIVOS LÓGICOS E LÓGICA PROGRAMÁVEL . . 20
2.1 Introdução aos Dispositivos Lógicos Programáveis 20
2.1.1 SPLD e CPLD 20
2.1.2 FPGA 21
2.1.2.1 Blocos Lógicos Configuráveis (CLB) 23
2.1.2.2 Caixa de Conexões (SB) 24
2.1.2.3 Blocos de Entrada e Saída (IOB) 24
2.2 Linguagem de Descrição de *Hardware* - HDL 25
2.2.1 Elementos Básicos da Sintaxe VHDL 25
2.2.2 Conceitos Fundamentais do VHDL 27
2.2.2.1 Declaração de biblioteca e Pacotes 28
2.2.2.2 Declaração de Entidade 29
2.2.2.3 Declaração da Arquitetura 31
2.2.2.4 Configuração 34
2.3 Exemplos de funções lógicas implementadas em VHDL 34
2.3.1 Descrição Física 35
2.3.2 Descrição Comportamental 35
2.3.3 Descrição Estrutural 36

3 ALTERA FPGA DE2-2C35 38
3.1 Diagrama de blocos da placa DE2-2C35 39
3.2 Aplicações do DE2-2C35 40
3.3 Quartus II 40
3.3.1 Criação de novo projeto 40
3.3.2 Criação do arquivo VHDL 43
3.3.3 Compilador 44
3.3.4 Simulação 44
3.3.5 Mapeamento de Pinos 47
3.3.6 Programmer 48

4 ROTEIROS EXPERIMENTAIS 51

4.1 Roteiro 1 - Entradas, saídas e portas lógicas básicas 52
4.1.1 Entradas e saídas 52
4.1.2 Portas lógicas básicas 52
4.1.3 Procedimentos Experimentais 52
4.2 Roteiro 2 - Circuitos Combinacionais 53
4.2.1 Multiplexadores 53
4.2.2 Decodificadores 54
4.2.3 Procedimentos Experimentais 55
4.3 Roteiro 3 - Circuitos Sequenciais 55
4.3.1 Flip-Flop e Registrador 55
4.3.2 Contador 57
4.3.3 Procedimentos Experimentais 57
4.4 Roteiro 4 - Relógio digital 58
4.4.0.1 Definição do problema 58

5 RESULTADOS 59
5.1 Roteiro 1 - Entradas, saídas e portas lógicas básicas 59
5.1.1 Entradas e saídas 59
5.1.2 Portas Lógicas Básicas 62
5.2 Circuito Combinacional 64
5.2.1 Multiplexadores 64
5.2.2 Decodificadores 67
5.3 Circuito Sequencial 69
5.3.1 Flip-Flop e Registrador 69
5.3.2 Contador 72
5.4 Relógio digital 75
5.4.1 Entidade 75
5.4.2 Arquitetura 76
5.4.3 Mapeamento dos pinos 78

6 CONSIDERAÇÕES FINAIS 79

Referências 81

APÊNDICE A – DESCRIÇÃO VHDL PARA O RELÓGIO DIGITAL DO ROTEIRO 4 83

APÊNDICE B – PROCESSADOR 86

APÊNDICE C – ROM 89

APÊNDICE D – DECODIFICADOR 91

APÊNDICE E – CONVERSOR HEXADECIMAL PARA 7 SEGMENTOS 92

APÊNDICE F – BOTOES 94

APÊNDICE G – DIVISOR GENÉRICO 95

APÊNDICE H – CONTADOR DE PROGRAMA (PC) 97

APÊNDICE I – MULTIPLEXADOR 3:1 98

APÊNDICE J – UNIDADE LOGICA ARITMÉTICA (ULA) 99

APÊNDICE K – SOMADOR GENÉRICO 101

APÊNDICE L – MULTIPLEXADOR 2:1 102

APÊNDICE M – BANCO REGISTRADOR 103

APÊNDICE N – UNIDADE DE CONTROLE 105

ANEXO A – MAPEAMENTO DE PINOS 107

1 INTRODUÇÃO

Recentemente, tem-se notado um aumento significativo no número de aplicações que demandam cada vez mais processamento de dados, tais como mapeamento genético, computação gráfica, previsões meteorológicas e programas com grande quantidade de variáveis de entrada. Dowd e Severance (2010) descrevem que a computação de alto desempenho é uma área que se preocupa em criar condições para executar processamentos com carga computacional elevada em intervalos de tempo viáveis.

Quando uma aplicação exige mais capacidade de processamento do que os computadores convencionais podem oferecer, é necessário buscar alternativas para criar sistemas de alto desempenho. Essa busca por soluções de processamento de alta performance é essencial para possibilitar a execução de programas complexos em um intervalo de tempo aceitável, e vem impulsionando o desenvolvimento de novas tecnologias de hardware e software. O uso de técnicas de processamento paralelo, computação distribuída e processamento em nuvem são exemplos de abordagens que visam a criação de sistemas computacionais de alto desempenho (DOWD; SEVERANCE, 2010). Essas tecnologias permitem que programas que levariam anos para fornecerem resultados em máquinas comuns possam ser executados em muito menos tempo, em questão de horas ou dias.

Uma abordagem diferente é a que consiste em construir circuitos orientados a aplicação, que são capazes de fornecer um bom desempenho para essa aplicação específica, ou seja, construir um hardware dedicado a essa aplicação. Por exemplo, é possível projetar e fabricar um circuito integrado específico com o controle e as unidades funcionais personalizadas e otimizadas para uma dada aplicação. Com essa técnica de hardware dedicado pode-se atingir resultados muito bons com menos recursos de *hardware* (HAGER; WELTIN, 2017). O ASIC é um circuito integrado para uma aplicação específica, tendo por objetivo solucionar demandas do mercado.

Outra opção de dispositivo reconfigurável, é o FPGA (do inglês *Field-programmable Gate Array*). Este é um dispositivo que permite a implementação de aplicações específicas. E contrastando com o ASICs, a implementação em FPGAs não é permanente, podendo ser reconfigurada de forma barata e fácil. Como vantagem, este apresenta baixa granularidade, permitindo a configuração em nível de porta lógica. Isso significa que é possível criar virtualmente qualquer circuito digital a partir de uma descrição em HDL (do inglês, *Hardware Description Language*). Assim como um computador, o FPGA é capaz de executar milhares de operações simultâneas, tornando-o potencialmente centenas de vezes mais rápido do que projetos baseados em microprocessadores, devido à possibilidade de explorar o paralelismo espacial oferecido pelos elementos configuráveis presentes nos dispositivos atuais (D'AMORE, 2012).

FPGAs podem ser utilizados em diversas aplicações de sistemas embarcados, como no setor automotivo, médico, industrial, equipamentos de imagem e telecomunicações, por exemplo. Compete ao meio acadêmico e a indústria realizar estudos e incentivos para utilizar a combinação: lógica programável com processadores embarcados. Dado o crescente destaque deste campo de atuação, é interessante que conceitos de programação reconfigurável sejam introduzidos ao ensino de automação e eletrônica (D'AMORE, 2012). Desta forma, este trabalho busca contribuir com o ensino de Engenharia, utilizando o kit FPGA DE2-2C35 da Altera, por meio de uma abordagem pratica obtida pela criação e execução de roteiros aplicáveis em sala de aula. O DE2-2C35 é uma placa de desenvolvimento que incorpora uma FPGA da Altera, juntamente com uma variedade de periféricos e interfaces, tornando-o ideal para projetos de automação. Isso permite que os estudantes aprendam conceitos importantes, como programação em VHDL (VHSIC *Hardware Description Language*), design de sistemas digitais, interfaceamento com sensores e atuadores, e implementação de lógica de controle em hardware. Os discentes podem projetar e implementar sistemas de automação em tempo real, utilizando a plataforma FPGA para desenvolver circuitos personalizados e reconfiguráveis.

Através do ensino de automação utilizando o DE2-2C35, os estudantes podem adquirir habilidades práticas e relevantes para a indústria, como design de sistemas embarcados, programação de dispositivos reconfiguráveis, desenvolvimento de algoritmos de controle em hardware e integração de sistemas de automação com o mundo físico. Além disso, o uso do DE2-2C35 pode incentivar a criatividade e a inovação, permitindo que os estudantes projetem soluções personalizadas para problemas específicos de automação.

1.1 Justificativa

O uso de *hardware* reconfigurável permite a operação dedicada a um objetivo específico, sem influência de outros processos em concorrência, o que resulta em ganho de desempenho e processamento dedicado para a execução de algoritmos. É essencial que os profissionais de engenharia elétrica sejam introduzidos a ferramentas de programação digital, especialmente para análise e aplicação em campos como telecomunicações. No entanto, muitos estudantes enfrentam dificuldades nesses temas de estudo devido à escassez de materiais de ensino disponíveis. Portanto, esse trabalho propõe o desenvolvimento de material de ensino da linguagem VHDL, visando sua implementação em componentes curriculares de eletrônica digital e telecomunicações.

1.2 Objetivos

Este trabalho tem como objetivo, propor a criação de material didático, e de fácil compreensão, para ensino de automação a partir de Dispositivos Lógicos Programáveis,

utilizando um Kit de Aprendizado e Desenvolvimento FPGA (DE2-2C35), programado via VHDL.

Para alcançar o objetivo proposto, foram definidas seguintes metas:

a) Realizar uma revisão bibliográfica sobre os fundamentos de sistemas digitais e lógica de programação;

b) Estudar a implementação de códigos em VHDL;

c) Estudar o funcionamento do FPGA DE2-2C35 da Altera;

d) Propor roteiros experimentais para estudo de lógica combinacional e sequencial;

e) Documentar os resultados obtidos, contribuindo com o ensino de sistemas digitais e eletrônica reconfigurável."

1.3 Estrutura da Monografia

Este trabalho está estruturado em seis capítulos.

O capítulo 1 é introdutório, no qual são apresentados a justificativa, os objetivos e a estrutura do trabalho desenvolvido.

No capítulo 2, é apresentada a fundamentação teórica sobre sistemas digitais, lógica programável e dispositivos lógicos programáveis.

O capítulo 3 reúne as principais partes e o funcionamento do kit FPGA DE2-2C35 da Altera.

No capítulo 4, são apresentados os roteiros experimentais que podem ser aplicados para o ensino de automação utilizando sistemas lógicos programáveis.

O capítulo 5 apresenta os resultados obtidos a partir da execução dos experimentos descritos no capítulo 4.

Por fim, são apresentadas as considerações finais e os anexos contendo o mapeamento de pinos do FPGA DE2 modelo 2C35.

2 DISPOSITIVOS LÓGICOS E LÓGICA PROGRAMÁVEL

A lógica programável é um campo que se baseia em princípios e conceitos, que permitem a criação de soluções para problemas reais por meio do uso de dispositivos programáveis. Esses dispositivos são capazes de executar funções lógicas específicas, as quais podem ser definidas pelo fabricante ou pelo usuário, utilizando tanto *software* quanto *hardware*. Além disso, a lógica programável tem sido amplamente utilizada em diversos campos, como em sistemas embarcados (MISHRA; SARMA, 2018), comunicações digitais (PROAKIS; MANOLAKIS, 2006), processamento de sinais (JUNIOR et al., 2022), entre outros. Esses avanços têm sido possíveis graças aos esforços contínuos de pesquisa e desenvolvimento em tecnologia de sistemas e programas. De acordo com Brown e Rose (2013), os dispositivos programáveis podem ser classificados em várias categorias, dependendo do nível de abstração e da tecnologia utilizada.

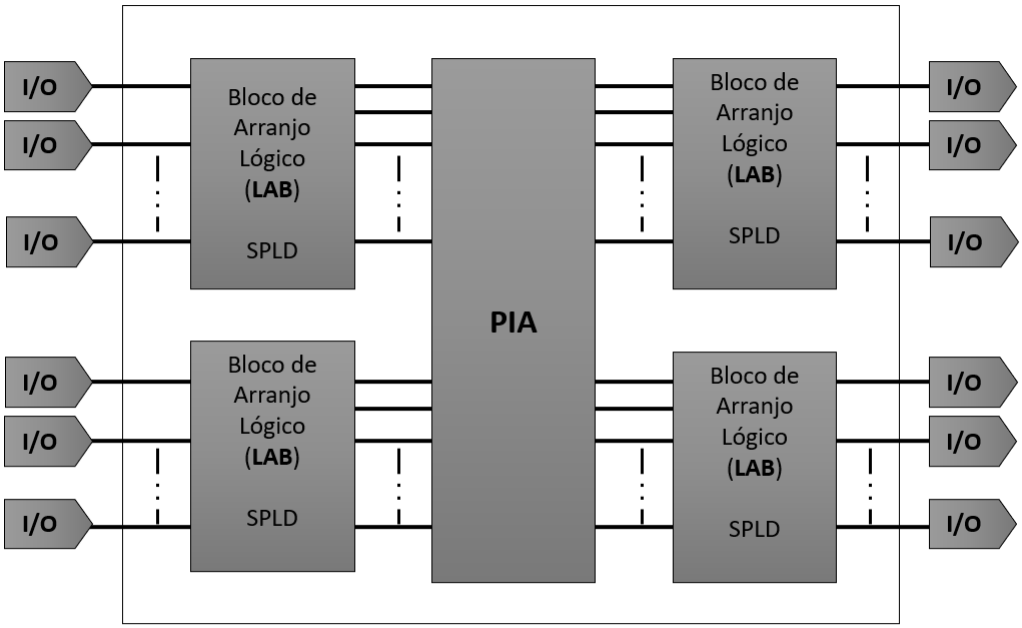
2.1 Introdução aos Dispositivos Lógicos Programáveis

Na década de 1970, surgiram no mercado os dispositivos lógicos programáveis (PLD), que revolucionaram a área de circuitos lógicos combinacionais. Um PLD é basicamente um tipo de chip para uso geral que possui *hardware* configurável e/ou reconfigurável, em contraste com os microprocessadores, que executam programas por meio de *hardware* fixo (PEDRONI, 2010). Dentro da categoria de PLDs, há uma ampla variedade de chips disponíveis, sendo a maioria deles agrupados sob o termo Dispositivos Lógicos Programáveis Simples (SPLD). No entanto, os mais relevantes e conhecidos são os Dispositivos Lógicos Programáveis Complexos (CPLD), os Arranjos de Portas Programáveis em Campo (FPGA) e os Circuitos Integrados de Aplicação Específica (ASIC) (MOORE; WILSON, 2017).

2.1.1 SPLD e CPLD

Geralmente, um SPLD é utilizado em aplicações que exigem menos complexidade, como projetos de pequeno porte. Existem vários tipos de SPLDs disponíveis no mercado, incluindo PALs (*Programmable Array Logic*) e GALs (*Generic Array Logic*). Cada um desses tipos tem suas próprias características e vantagens, tornando-os adequados para diferentes aplicações (FLOYD, 2009). Já o CPLD é um dispositivo eletrônico programável que permite a criação de circuitos lógicos complexos em um único componente. Eles são compostos por vários SPLDs interconectados por meio de um arranjo complexo e programável. A figura 1 ilustra a estrutura de um CPLD convencional (PEDRONI, 2010). Essa estrutura possibilita uma maior capacidade para projetar circuitos lógicos maiores e mais complexos do que os que poderiam ser criados com apenas um SPLD.

Figura 1 – Arquitetura básica de CPLDs



Fonte: Adaptado de Floyd (2009).

O CPLD pode ser reprogramado a qualquer momento, permitindo que o circuito seja atualizado e adaptado às necessidades do projeto. Cada arranjo SPLD em um CPLD é chamado de Bloco de Arranjo Lógico (LAB), que é uma unidade lógica programável que pode executar funções lógicas complexas. Cada LAB é interconectado por linhas de interconexão programáveis, chamadas de Arranjo de Interconexões Programáveis (PIA). Essas interconexões programáveis permitem que os blocos lógicos se comuniquem entre si e formem circuitos digitais complexos e customizados (FLOYD, 2009).

Com esta classe de PLD, é possível implementar funções digitais de média e alta complexidade, incluindo sistemas sequenciais e combinacionais, como registradores de deslocamento, contadores, multiplexadores e decodificadores. Além disso, eles são frequentemente utilizados em aplicações de alta velocidade, como em sistemas de processamento de sinais digitais, processamento de imagens e em sistemas de controle de automação industrial (PEDRONI, 2010) .

2.1.2 FPGA

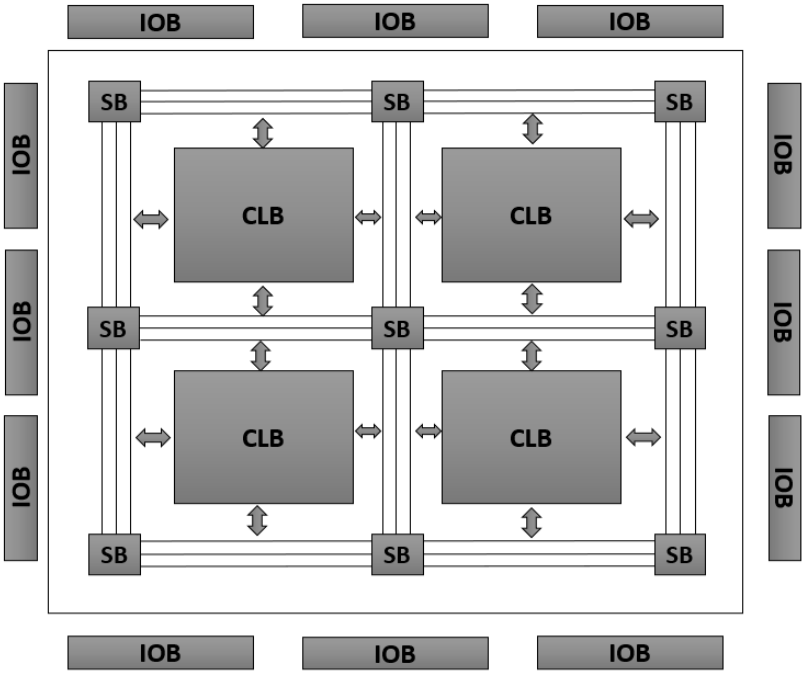
Um FPGA, em tradução, um Arranjo de Portas Programáveis em Campo, é um dispositivo lógico que possui uma estrutura interna ainda mais complexa que um CPLD. Ordonez et al. (2003) afirmam que a estrutura fundamental de um FPGA pode variar entre fabricantes, famílias de dispositivos e até mesmo dentro de uma mesma família, mas existem alguns elementos essenciais que são mantidos, como é mostrado na figura 2, os três elementos básicos que constituem um FPGA são :

a) CLB (*Configurable Logic Block*): Blocos Lógicos Configuráveis, unidade lógica de um FPGA;

b) SB (*Switch Box*): Caixa de Conexão, responsável pela conexão entre os CLBs, através de canais de roteamento;

c) IOB (*In/Out Block*): Blocos de Entrada/Saída, responsáveis pela interface com o ambiente.

Figura 2 – Representação dos elementos básicos de um FPGA



Fonte: Adaptado de Floyd (2009).

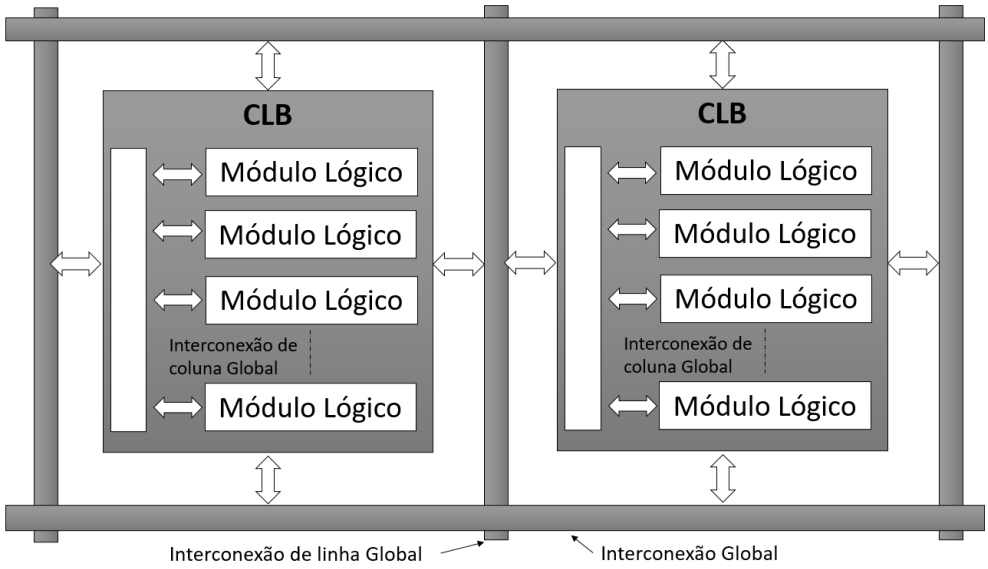
A característica programável em um FPGA é semelhante à interconexão de chaves em uma placa de prototipagem que pode ser alterada pelo projetista. No FPGA, essa interconexão pode ser reprogramada pelo usuário ou pelo designer no laboratório ou em campo. Isso é possível porque o FPGA é composto por uma matriz de células lógicas, que podem ser programadas para executar funções específicas, como portas lógicas, registradores, memórias, entre outras. A programação é feita através de um processo de configuração que pode ser realizado sem a necessidade de retirar o dispositivo do circuito, o que significa que é possível atualizar ou reconfigurar o FPGA sem ter que substituir o *hardware* (ROTH, 2004). Por isso, essa matriz de portas lógicas (*Gate Array*) é chamada de programável em campo (*Field Programmable*). Os tipos de portas lógicas programáveis incluem todas as portas básicas para atender a qualquer necessidade (SMITH; FRANZON, 2008).

2.1.2.1 Blocos Lógicos Configuráveis (CLB)

Os blocos lógicos configuráveis são uma das principais unidades funcionais dos FPGA. Cada CLB consiste em uma rede de células lógicas com a lógica desejada, além de possuir uma variedade de elementos de armazenamento, tais como *flip-flops*, *latches* e registradores. A arquitetura interna dos CLBs varia entre os fabricantes de FPGA, mas em geral eles possuem recursos de roteamento programáveis que permitem a interconexão entre os CLBs e outros blocos funcionais do FPGA. A figura 3 ilustra a disposição dos CLBs essenciais nas interconexões programáveis globais em linha/coluna. Essas interconexões têm a função de estabelecer as conexões necessárias para interligar os blocos lógicos entre si. (FLOYD, 2009).

Os CLBs possuem grande flexibilidade, permitindo que os designers de *hardware* personalizem a lógica interna de acordo com as necessidades de seus projetos. Isso é possível através da programação do FPGA por meio de linguagens de descrição de *hardware* (HDL), como o VHDL, que permite a descrição da lógica desejada em um nível de abstração alto e portável entre diferentes fabricantes de FPGA (SILVA; ASSUNÇÃO, 2013).

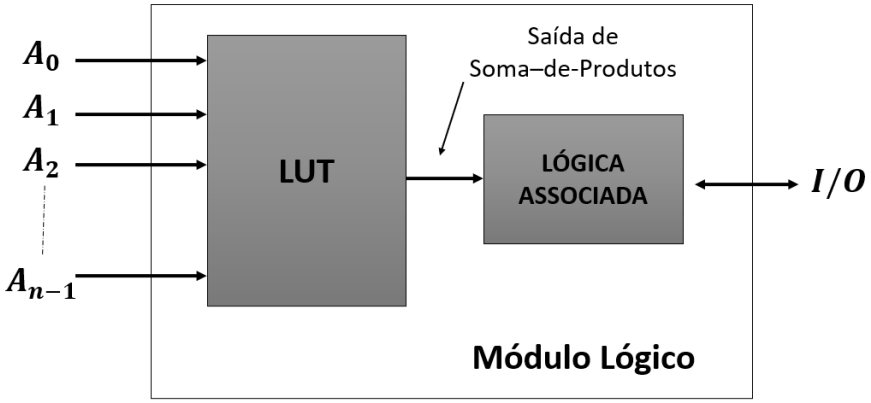
Figura 3 – Blocos lógicos configuráveis básicos (CLBs) dentro de interconexões programáveis globais em linha/coluna.



Fonte: Adaptado de Floyd (2009).

Cada CLB em um FPGA é composto por vários módulos lógicos menores e uma interconexão programável local que permite a conexão entre esses módulos dentro do CLB. Esses módulos lógicos podem ser configurados como lógica combinacional, lógica registrada ou uma combinação de ambas. A figura 4 ilustra o diagrama em bloco de um módulo lógico típico baseado em LUT (*Look Up Table*). Uma LUT, ou tabela de busca, é uma forma de memória programável utilizada para gerar funções lógicas combinacionais em formato de soma-de-produtos (FLOYD, 2009).

Figura 4 – Modulo Lógico.



Fonte: Adaptado de Floyd (2009).

2.1.2.2 Caixa de Conexões (SB)

As Caixas de Conexões servem para unir as interconexões programáveis, sendo estas responsáveis por estabelecer a conexão entre os CLBs e os blocos de entrada/saída (IOB) para implementar a lógica digital desejada. Essas interconexões são constituídas por um conjunto de elementos programáveis, como chaves, multiplexadores e *buffers*, que permitem a configuração do caminho de sinal entre os blocos (BROWN et al., 1992).

Além disso, a disposição das interconexões também é importante para o desempenho e eficiência do FPGA. Por exemplo, a utilização de canais de roteamento que permitem a comunicação vertical e horizontal entre os CLBs, além de reduzir a latência e minimizar a interferência entre os sinais, também possibilita a implementação de circuitos mais complexos (WONG; BRICAUD, 2007).

2.1.2.3 Blocos de Entrada e Saída (IOB)

Os blocos de entrada e saída são responsáveis por fornecer uma interface entre o circuito digital do FPGA e o mundo externo. Eles permitem com que haja a comunicação entre dispositivos, como sensores, atuadores, processadores e outros circuitos digitais.Estes são projetados para serem configuráveis e suportar vários padrões de comunicação, como interfaces seriais, paralelas e diferenças, entre outras. Eles são compostos por circuitos específicos, como registradores, *buffers*, multiplexadores e transceptores, que são configuráveis para atender às necessidades do projeto (BROWN; ROSE, 2013; MAXFIELD, 2004).

A organização dos blocos de entrada e saída varia de acordo com a arquitetura do dispositivo e a família do FPGA. Por exemplo, alguns FPGA possuem blocos de entrada e saída dedicados que são alocados em áreas específicas do dispositivo, enquanto outros possuem blocos de entrada e saída espalhados por toda a matriz de lógica do dispositivo (Xilinx Inc., 2021).

2.2 Linguagem de Descrição de *Hardware* - HDL

Dispositivos lógicos programáveis, como os FPGAs, requerem o uso de uma linguagem específica para implementar circuitos físicos em seu próprio *hardware*. Essa linguagem é conhecida como Linguagem de Descrição de *Hardware* (HDL) (ORDONEZ et al., 2003).

As HDLs são estruturadas de forma a permitir a descrição abstrata do comportamento de *hardware*, possibilitando a modelagem e representação da estrutura e funcionamento de circuitos em diferentes níveis de abstração durante o processo de projeto (ORDONEZ et al., 2003). Isso oferece aos projetistas maior flexibilidade na descrição de circuitos complexos. Anteriormente, essas linguagens eram comumente empregadas para a verificação de lógica. No entanto, com o surgimento dos algoritmos de síntese lógica, tornou-se viável descrever o fluxo de dados dos circuitos em níveis de transferência de registradores (RTL), simplificando a descrição de circuitos digitais. Com isso, detalhes como portas lógicas e interconexões passaram a ser processados automaticamente pelos algoritmos de síntese lógica. (COSTA, 2014).

As linguagens de descrição de *hardware* oferecem várias vantagens em relação aos projetos tradicionais baseados em esquemas elétricos. Permitem a descrição de circuitos digitais de forma independente da tecnologia de fabricação, graças ao nível de abstração dessas linguagens. Além disso, permitem a verificação funcional dos circuitos em estágios iniciais do projeto e facilitam a análise de falhas e erros, uma vez que as HDLs são linguagens textuais semelhantes às linguagens de programação convencionais, como C, C++, Java, entre outras (COSTA, 2014).

Existem várias HDLs disponíveis, porém, as mais conhecidas e amplamente utilizadas são a linguagem Verilog e a linguagem VHDL (VHSIC *HDL,* onde VHSIC - *Very High Speed Integrated Circuit*) (ORDONEZ et al., 2003).

2.2.1 Elementos Básicos da Sintaxe VHDL

A VHDL é uma linguagem de descrição que segue um conjunto de regras bem definidas. Ela requer o uso de palavras-chave e uma sintaxe específica, onde todas as variáveis, operadores e valores devem ser escritos com letras e símbolos do alfabeto. A seguir os elementos sintáticos mais comuns:

a) Comentários: Qualquer linha precedida de ”–” não será compilada.

b) Símbolos especiais: + - / * ( ) . : ;, ‘ “ < >, = |

c) Identificadores: São usados para nomear objetos da linguagem como variáveis, sinais, rotinas, etc. Composto por letras, números e o símbolo. Nunca pode conter símbolos especiais ou coincidir com palavras reservadas.

d) Case-sensitive: Maiúsculas e minúsculas são consideradas iguais, assim VHDL, vhdl e VhDl são possibilidades que representam o mesmo objeto.

e) Números: Qualquer número que se encontra é considerado na base 10. Admite-se a notação científica convencional para números com ponto flutuante. É possível alterar a base de um número usando o simbolo # . Exemplo: 2#00001111, 16#0F ambos representam o número 15, respectivamente na base 2 e 16.

f) Caracteres: Qualquer caractere deve estar entre aspas simples: '1' 'A'.

g) Cadeia de bits: Cadeia ou vetor de bits é uma sequencia de bits em uma determinada base Ex.: B "00001111", X"0F", onde B é binário e X Hexadecimal.

h) Operadores lógicos: NOT, AND, NAND, OR, NOR, XOR ou XNOR atuam sobre os tipos bit,bit_vector e *boolean*. No caso de operações com vetores, a operação é realizada bit a bit.

i) Palavras reservadas: São palavras que possuem um significado especial, são instruções e elementos que permitem definir sentenças. É relevante destacar que o VHDl não é *case sensitive*, ou seja não faz diferenciação entre maiúsculas e minusculas. A seguir, as palavras reservadas mais utilizadas são apresentados no quadro 1.

Quadro 1 – Palavras reservadas em VHDL

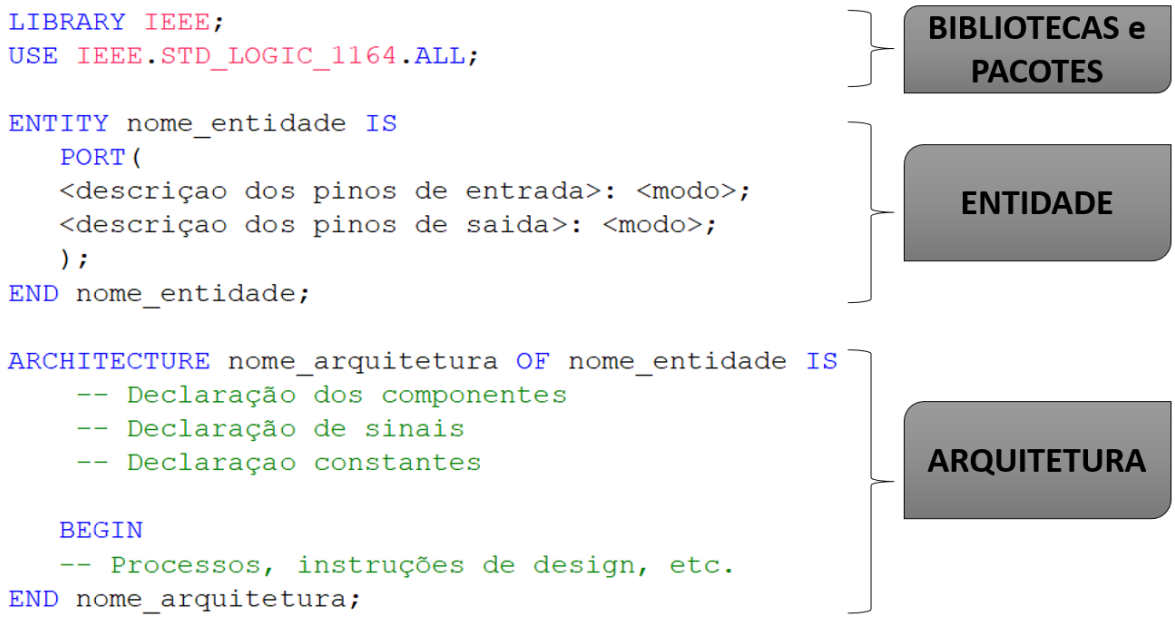
ABS	CONFIGURATION	INOUT	OR	THEN
ACCESS	CONSTANT	IS	OTHERS	TO
AFTER	DISCONNECT	LABEL	OUT	TRANSPORT
ALIAS	DOWNTO	LIBRARY	PACKAGE	TYPE
ALL	ELSE	LINKAGE	PORT	UNITS
AND	ELSIF	LOOP	PROCEDURE	UNTIL
ARCHITECTURE	END	MAP	PROCESS	USE
ARRAY	ENTITY	MOD	RANGE	VARIABLE
ASSERT	EXIT	NAND	RECORD	WAIT
ATTRIBUTE	FILE	NEW	REGISTER	WHEN
BEGIN	FOR	NEXT	REM	WHILE
BLOCK	FUNCTION	NOR	REPORT	WITH
BODY	GENERATE	NOT	RETURN	XOR
BUFFER	GENERIC	NULL	SELECT	XNOR
BUS	GUARDED	OF	SEVERITY	SLL
CASE	IF	ON	SIGNAL	SLA
COMPONENT	IN	OPEN	SUBTYPE	ROL

Fonte: (ORDONEZ et al., 2003).

2.2.2 Conceitos Fundamentais do VHDL

A linguagem VHDL é baseada em uma estrutura hierárquica, onde os componentes são definidos em níveis de abstração, desde o nível mais alto até o nível mais baixo. Isso permite a criação de modelos complexos, divididos em blocos menores e interconectados. No VHDL, a estrutura básica de um design é composta por entidades e arquiteturas (figura 5). A entidade descreve a interface do componente, definindo as portas de entrada e saída, bem como quaisquer parâmetros genéricos. A arquitetura, por sua vez, contém a descrição do comportamento interno do componente, especificando as operações lógicas e a lógica de controle. Dentro desses blocos do design, um circuito lógico ou função pode ser facilmente descrito (HARRIS; HARRIS, 2015; SOUSA, 2017).

Figura 5 – Sintaxe do algoritmo em VHDL utilizando a biblioteca IEEE



Fonte: Autoria própria.

Um design em VHDL começa com um bloco de bibliotecas e pacotes, que fornecem funções uteis ao projeto proposto, seguindo para o bloco de entidade que descreve a interface para o design. A interface define os sinais lógicos de entrada e saída do circuito projetado. O bloco de arquitetura descreve a operação interna do design. Dentro desses blocos, existem vários outros blocos funcionais usados para construir os elementos de design do circuito lógico. A figura 5 exemplifica um código em VHDL, usou-se a biblioteca desenvolvida pelo Instituto de Engenheiros Eletricistas e Eletrônicos (IEEE) muito comum em projetos de descrição de *hardware*, e deste comumente faz-se uso do pacote std_logic_1164. Nesta estrutura a entidade é denominada "nome_entidade", ressalta-se que o nome da entidade deve ser o mesmo nome dado ao projeto desenvolvido na IDE escolhida (este assunto será retomado mais a frente). A arquitetura pode ser nomeada a critério do projetista, para fins didáticos, aqui a arquitetura recebeu o nome de "nome_arquitetura"(ORDONEZ et al., 2003).

2.2.2.1 Declaração de biblioteca e Pacotes

As bibliotecas e os pacotes são conjuntos de declarações usadas para um determinado fim. Pode ser um conjunto de subprogramas, ou funções com um tipo de dado, assim como pode ser um conjunto de declarações necessárias para modelar um determinado projeto. Uma biblioteca é usada para organizar e gerenciar diferentes unidades de design em um projeto VHDL. É comum criar diferentes bibliotecas para organizar e armazenar unidades de design relacionadas. Por exemplo, uma biblioteca pode ser criada para armazenar unidades de design relacionadas a interface serial, enquanto outra biblioteca pode ser criada para armazenar unidades de design relacionadas a processamento de sinais (ROTH, 2004). Os pacotes são usados para definir e compartilhar constantes, tipos de dados, funções, procedimentos e outras construções que podem ser usadas em várias unidades de design. A sintaxe da declaração para o uso de pacotes e bibliotecas é mostrada no quadro 2. Neste pode ser visto o uso de .ALL, que é usada para importar todos os elementos de um pacote específico (ORDONEZ et al., 2003).

Quadro 2 – Sintaxe de biblioteca

```
LIBRARY < nome da biblioteca >
    USE < nome do pacote >.ALL
```

Fonte: Adaptado de Ordonez et al. (2003).

Devido à sua abrangência e funcionalidade, a biblioteca padronizada pelo IEEE é amplamente adotado na indústria e utilizado em uma variedade de projetos de descrição de *hardware*. O quadro 3 apresenta os principais pacotes desta, e conforme mencionado anteriormente, o pacote IEEE.std_logic_1164 é um dos mais utilizado em projetos de descrição de *hardware* em VHDL, isso se dá devido à sua eficiência na definição do tipo de dados std_logic e na disponibilização de um conjunto abrangente de operações lógicas para trabalhar com sinais lógicos. O tipo de dados std_logic é essencial em projetos de *hardware*, pois representa os sinais digitais que são a base do funcionamento dos circuitos. Ele possui valores como '0' (nível lógico baixo), '1' (nível lógico alto) e 'Z' (alta impedância), entre outros. Além disso, o tipo std_logic também inclui operações lógicas, como AND, OR e NOT, que permitem realizar manipulações e cálculos lógicos em sinais . Ao utilizar esse pacote, os projetistas podem se beneficiar de um conjunto consistente de operações lógicas pré-definidas, evitando a necessidade de implementá-las manualmente em seus projetos (EDWARDS, 2011).

Quadro 3 – Pacotes mais utilizados (Biblioteca IEEE)

Pacote	**Tipo de Dados**	**Descrição**
STD_LOGIC_1164	STD_LOGIC e operações lógicas relacionadas	O tipo std_logic é usado para representar sinais digitais. Este também inclui funções para realizar operações lógicas, como AND, OR e NOT, em sinais STD_LOGIC.
NUMERIC_STD	signed e unsigned	Define tipos de dados como INTEGER, SIGNED e UNSIGNED, que são usados para representar números inteiros com e sem sinal. Também oferece operadores aritméticos, como adição, subtração, multiplicação e divisão, que podem ser aplicados a esses tipos de dados.
STD_LOGIC_unsigned e STD_LOGIC_signed	STD_LOGIC_vector	Esses pacotes são variantes especializadas do pacote numeric_std. Incluem funções para operações aritméticas, como adição, subtração e multiplicação, em vetores de sinais std_logic.
STD_LOGIC_tectio	STD_LOGIC	Fornece funções de entrada e saída textual para sinais do tipo STD_LOGIC.

Fonte: Adaptado de Ordonez et al. (2003).

2.2.2.2 Declaração de Entidade

Ao criar um componente em VHDL, é necessário defini-lo como uma entidade. Essa entidade é uma representação formal do componente, que pode variar desde uma porta lógica simples até um sistema lógico completo. Ao declarar a entidade, é preciso descrever todas as entradas e saídas do projeto, o que é similar ao símbolo de um bloco utilizado em um esquema de circuito (D'AMORE, 2012).

Inicia-se com o termo *ENTITY* seguida de um nome definido pelo usuário, prosseguido da palavra reservada *IS* para indicar que a entidade está começando. Dentro da entidade é descrito a seção *PORT* para que as portas da entidade sejam apresentadas. Desta forma, será declarada uma lista das portas de entrada e saída do circuito. Para finalizar a declaração da entidade, usa-se a palavra reservada *END* seguida do nome da entidade e de ponto e vírgula (;) (ORDONEZ et al., 2003). A sintaxe é apresentada no quadro 4.

Quadro 4 – Sintaxe de entidade

```
ENTITY < nome da entidade > IS
    PORT(
        A,B : < modo das entradas >;
        S   : < modo das saidas >
        );
END < nome da entidade>;
```

Fonte: Adaptado de Ordonez et al. (2003).

A declaração das portas começa com a palavra reservada *PORT* seguida por um parêntese "("e em seguida, os nomes das entradas/saídas separados por vírgula (,) seguidos por dois pontos (:) para indicar o modo, finalizando cada porta com ponto e vírgula (;). O comando PORT é concluída com fechamento do parêntese ")"e ponto e vírgula (;). A seguir são apresentados os quatro modos de operação do elemento *PORT*.

a) Modo *IN* : Entrada.

b) Modo *OUT*: Saída.

c) Modo *BUFFER*: Saída que pode realimentar o circuito.

d) Modo *INOUT*: Bidirecional.

Como boa prática, é comum as portas de entrada serem declaradas antes das portas de saída. Ressalta-se que cada declaração da interface é seguida por ponto e virgula (;). O quadro 5 expressa os tipos de portas mais utilizados em projetos programados em VHDL.

Quadro 5 – Tipos de portas mais utilizadas

Tipo de Dados	**Sintaxe**
BIT	x: IN BIT;
BIT_VECTOR	x: IN BIT_VECTOR(7 DOWNTO 0);
STD_LOGIC	x: IN STD_LOGIC;
STD_LOGIC_VECTOR	x: IN STD_LOGIC_VECTOR(7 DOWNTO 0);
BOOLEAN	x: IN BOOLEAN;
INTEGER	x: IN INTEGER RANGE 0 TO 10;

Fonte: Adaptado de D'Amore (2012).

As portas citadas anteriormente foram exemplificadas como entrada (*IN*), mas o mesmo é valido para os demais modos (*OUT, BUFFER, INOUT*).

2.2.2.3 Declaração da Arquitetura

A declaração da arquitetura em VHDL é responsável por descrever o comportamento da entidade, ou seja, como ela funciona internamente, levando em conta as suas entradas, saídas e sinais internos. Essa descrição é feita por meio de uma série de comandos que podem ser concorrentes ou sequenciais, e que incluem declarações de sinais, constantes, componentes, operadores lógicos, entre outros. A declaração é importante porque permite que sejam definidas as operações que a entidade deve realizar, como ela deve se comportar em determinadas situações, e como deve interagir com outros sinais e componentes do circuito (D'AMORE, 2012). A linguagem VHDL permite ter mais de uma arquitetura para a mesma entidade. A arquitetura é definida usando a seguinte sintaxe do quadro 7.

Quadro 6 – Sintaxe de arquitetura

```
ARCHITECTURE < nome da arquitetura > OF < nome da entidade > IS
-- Declarações de sinais, variaveis e constantes
    BEGIN
--Declarações da lógica do circuito
END < nome da arquitetura >;
```

Fonte: Adaptado de Ordonez et al. (2003).

A área entre a chave *ARCHITECTURE* e a chave *BEGIN* é chamada de seção de declaração da arquitetura, após esta tem-se o corpo da arquitetura, o qual especifica seu comportamento. Em VHDL, existem três formas de descrever o comportamento de circuitos digitais: a Estrutural, a Física e a Comportamental. As descrições Estrutural e Física envolvem identificar os componentes individuais que compõem o circuito e suas interconexões. Isso permite especificar um circuito e entender como ele funciona (ALMEIDA; CARAVALHO, 2020). Por outro lado, a descrição Comportamental consiste em descrever o circuito com base no seu comportamento e funcionamento, sem se preocupar com a sua estrutura interna. Essa abordagem é útil quando a estrutura interna não está disponível, mas o comportamento e funcionamento do circuito podem ser interpretados (SILVA, 2019). Ao descrever um circuito utilizando a linguagem VHDL, é comum combinar trechos implementados de forma comportamental e estrutural.(ALMEIDA; CARAVALHO, 2020). A seguir são apresentados elementos úteis a descrição de uma arquitetura.

a) Componente (*COMPONENT*): É uma descrição abstrata e reutilizável de um bloco de *hardware* que define apenas a interface deste. O quadro 7 demonstra a estruturação de sua sintaxe. Como pode ser notado, esta se assemelha a descrição de uma entidade, a diferença esta no fato de que uma entidade é uma descrição completa e detalhada de um bloco de *hardware*, incluindo a interface e a implementação interna.

Quadro 7 – Sintaxe de componentes

```
ARCHITECTURE < nome da arquitetura > OF < nome da entidade > IS
    COMPONENT < nome do coponente > IS
-- Declarações das portas;
        PORT( A : modos        );
    END COMPONENT < nome do componente >;
--Sinais, etc;
    BEGIN
--Declaração da lógica do circuito

END < nome da arquitetura >;
```

Fonte: Adaptado de Ordonez et al. (2003)

b) Sinal (*SIGNAL*): Os sinais são variáveis utilizadas para armazenar e transmitir dados entre as diferentes partes da arquitetura. Dentro de uma arquitetura, a declaração de um sinal pode ocorrer em qualquer lugar do bloco, mas geralmente é colocada no início, logo após a declaração dos componentes. São expressos de acordo com a seguinte sintaxe (quadro 8);

Quadro 8 – Sintaxe de sinais

```
ARCHITECTURE < nome da arquitetura > OF < nome da entidade > IS

        SIGNAL sinal1 : tipo_dados;
        SIGNAL sinal2 : tipo_dados;
BEGIN
--Declaração da lógica do circuito

END < nome da arquitetura >;
```

Fonte: Adaptado de Edwards (2011).

c) Atribuições : A atribuição é usada para associar um valor a um sinal ou variável. O operador de atribuição em VHDL é '<='. O quadro 9 expressa sua sintaxe.

Quadro 9 – Sintaxe atribuições

```
sinal <= valor;
```

Fonte: ALTERA (2011)

d) Estrutura de controle: fornece estruturas que permitem a tomada de decisões com base em condições e a seleção de diferentes ações com base em valores específicos. A seguir podemos exemplificar por meio do quadro 10 as respectivas sintaxes.

Quadro 10 – Sintaxe de Estruturas de Controle em VHDL

```
ARCHITECTURE < nome da arquitetura > OF < nome da entidade > IS

--IF
IF < condição > THEN
-- executado se a condição for verdadeira
ELSE
-- executado se a condição for falsa
END IF;

--CASE
CASE < expressão > IS
    WHEN valor1 =>
-- executado se a expressão for igual ao valor1
    WHEN OTHERS =>
-- nenhum dos valores anteriores
END CASE;

-- FOR LOOP
FOR variável IN RANGE LOOP
-- bloco de código a ser repetido
END LOOP;

--WHILE LOOP
WHILE < condição > LOOP
-- repedido enquanto a condição for verdadeira
END LOOP;

END < nome da arquitetura >;
```

Fonte: Adaptado de D'Amore (2012).

e) Processo (*PROCESS*): São blocos de código que descrevem o comportamento sequencial do componente (quadro 11). A estrutura da arquitetura comportamental é elaborada ao descrever sua função usando uma ou várias sentenças de processo, que consistem em conjuntos de ações a serem executadas. As ações incluem expressões de avaliação, atribuição de valores a variáveis, execução condicional, repetição de expressões e chamadas de subprogramas. Dentro dos processos, os comandos são executados em sequência, mas se houver mais de um processo na arquitetura, eles serão executados simultaneamente (D'AMORE, 2012).

Quadro 11 – Sintaxe de processos

```
ARCHITECTURE < nome da arquitetura > OF < nome da entidade> IS
BEGIN
-- DEclarações e conexões de sinais e componentes
    PROCESS ( < lista de sinais >)
        BEGIN
--Atribuições, comandos de controle, etc.
    END PROCESS;
END < NOME DA ARQUITETURA>
```

Fonte: Adaptado de D'Amore (2012).

2.2.2.4 Configuração

A configuração estabelece a conexão entre a declaração de um componente e a entidade correspondente do projeto, definindo as arquiteturas que serão utilizadas. Uma mesma entidade pode ter várias arquiteturas associadas a ela, cada uma representando uma implementação ou comportamento específico. Ao utilizar configurações, é possível adaptar o projeto às necessidades específicas, selecionando as arquiteturas mais adequadas para cada situação. Isso permite uma maior modularidade e facilita a manutenção e evolução do projeto ao longo do tempo (ROTH, 2004).Para ligar a arquitetura à entidade é usada a seguinte declaração (quadro 12).

Quadro 12 – Sintaxe configuração

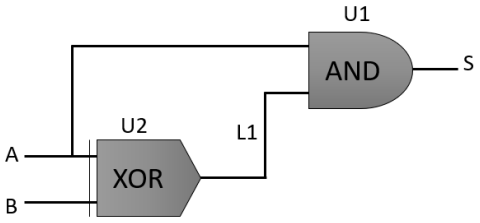
```
CONFIGURATION < nome da configuração > OF < nome da entidade > IS
FOR < nome da arquitetura >
-- Declarações e atribuições de parâmetros
END CONFIGURATION;
```

Fonte: Adaptado de D'Amore (2012).

2.3 Exemplos de funções lógicas implementadas em VHDL

A figura 6 (a) representa o modelo simplificado de um comparador de 1 bit, descrito em VHDL nas formas física (2.3.1), comportamental (2.3.2) e estrutural (2.3.3). Nesta representação, U1 e U2 são os componentes do circuito, L1 é uma linha de conexão entre os componentes, e A, B e S são as portas de entrada e saída do circuito respectivamente.

Figura 6 – Comparador de 1 bit simplificado.



A	B	L1	S
0	0	0	**0**
0	1	1	**0**
1	0	1	**1**
1	1	0	**0**

(a) Circuito lógico (b) Tabela verdade

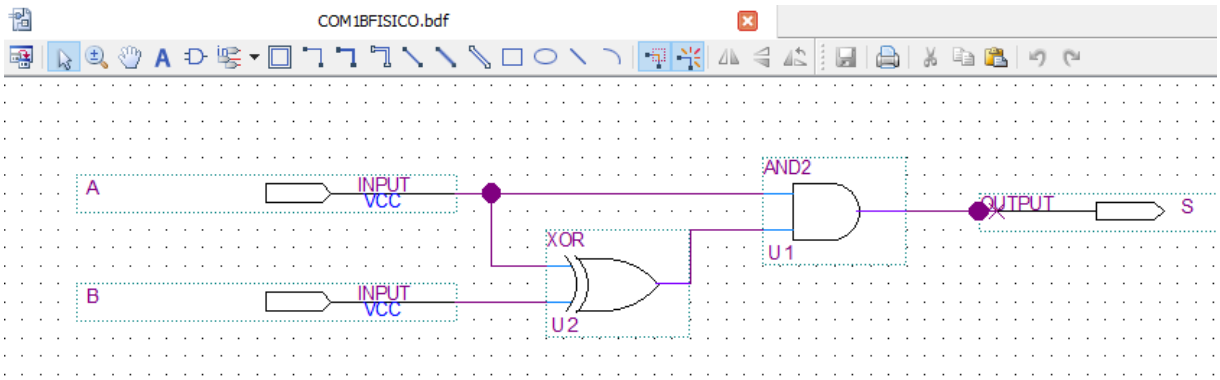
Fonte: Adaptado de Ordonez et al. (2003).

O funcionamento do comparador de 1 bit é simplificado, neste os valores lógicos das portas de entrada A e B são comparados. Após comparação, só é obtido nível lógico alto quando A e B forem diferentes, e A maior do que B. Isto é verificável dada tabela verdade presente na figura 6 (b).

2.3.1 Descrição Física

Através da estrutura física em VHDL (figura 7), cada módulo é descrito em detalhes, incluindo seu comportamento, sinais de entrada e saída, e conexões com outros módulos. Isso proporciona uma representação estruturada do circuito digital, permitindo a simulação, síntese e implementação do projeto em dispositivos reais.

Figura 7 – Estrutura física desenvolvida no *Software* Quartus II



Fonte: Autoria própria.

2.3.2 Descrição Comportamental

A descrição algorítmica apresenta um conjunto de passos que descreve de forma comportamental o circuito digital projetado. Primeiro é descrito a entidade, onde constam as portas de entrada e saída. Em seguida é descrita a arquitetura, responsável pela representação do circuito, neste caso, a arquitetura do comparador de 1 bit (quadro 13) é descrita na sequência, de maneira comportamental, aplicando o princípio algorítmico.

Quadro 13 – Representação Comportamental do comparador de 1 bit

```
ENTITY Comp1Bit_compt IS
    PORT (
        A, B: IN STD_LOGIC;
        S: OUT STD_LOGIC);
END Comp1Bit_compt;

ARCHITECTURE Arq_Comp1Bit_compt OF Comp1Bit_compt IS
BEGIN
    PROCESS (A, B)
    BEGIN
        IF A='1' and B ='0' THEN
            S <= '1';
        ELSE
            S <= '0';
        END IF;
    END PROCESS;
END Arq_Comp1Bit_compt;
```

Fonte: Adaptado de Ordonez et al. (2003).

A comparação foi descrita de forma comportamental utilizando o comando sequencial *IF-THEN-ELSE*, desprezando a lógica composta pelas portas lógicas *XOR* e *AND*.

2.3.3 Descrição Estrutural

Para que um circuito seja descrito na forma estrutural, é necessário conhecer seus componentes e interconexões. Em sistemas onde não é possível visualizar a estrutura interna de maneira detalhada, torna-se difícil a descrição utilizando a metodologia estrutural. Para o exemplo proposto o quadro 14 expressa este tipo de descrição. Perceba que a alteração só acontece no interior da arquitetura.

Quadro 14 – Representação Estrutural do comparador de 1 bit

```
ARCHITECTURE Arq_Comp1Bit_Estr OF Comp1Bit_Estr IS
    SIGAL  L1: STD_LOGIC;
BEGIN
    L1 <= A XOR B;
    S  <= AAND L1;
END Arq_Comp1Bit_compt;
```

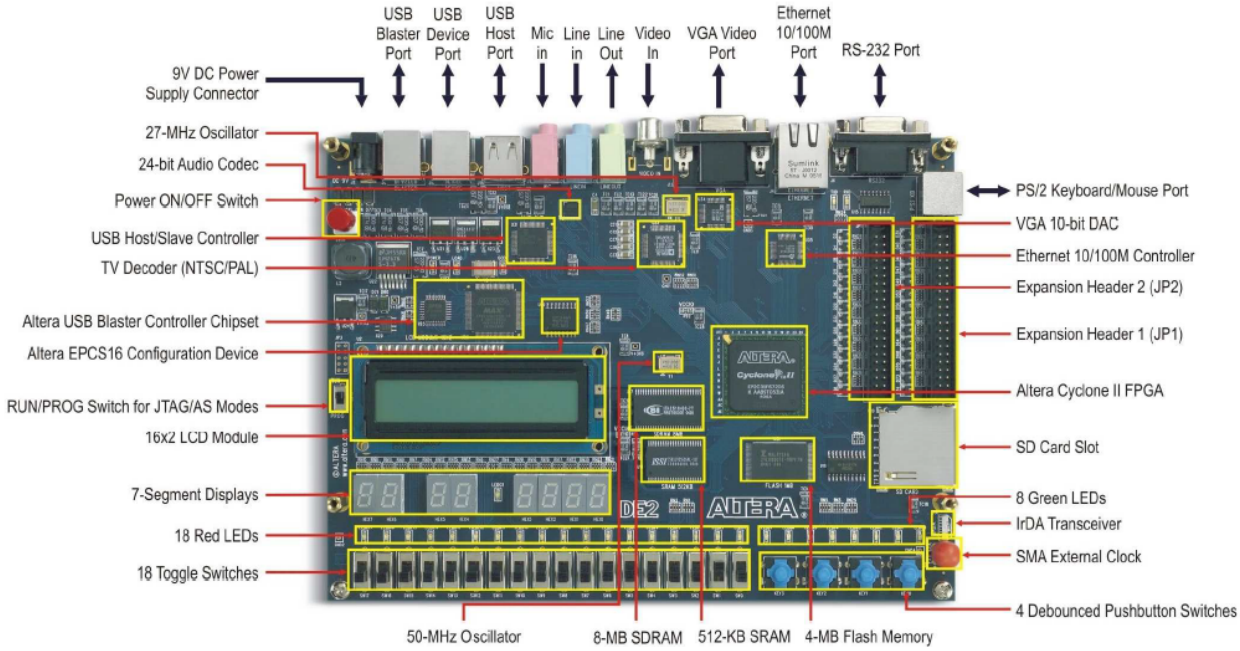
Fonte: Adaptado de Ordonez et al. (2003).

Nesta representação, se **A** for maior do que **B**, a saída **S** recebe o valor lógico **1**; caso contrário, **0**. Além disso, é interessante observar que, sem a correta visualização dos componentes e as respectivas interconexões do comparador de 1 bit, conforme a descrição da figura 6, pode dificultar sua descrição estrutural em VHDL.

3 ALTERA FPGA DE2-2C35

Neste trabalho, utilizou-se a placa de Desenvolvimento e Educação Altera DE2-2C35, ilustrada na figura 8. Esta é equipada com o FPGA Altera Cyclone II EP2C35F672C6, que possui 33.216 elementos lógicos. Além disso, a placa de desenvolvimento oferece uma variedade de recursos, incluindo botões de pressão (pushbuttons), LEDs, clocks de 50 MHz e 27 MHz, decodificador de vídeo e áudio, e pinos de expansão de uso geral (GPIO) para possibilitar a conexão de periféricos externos.

Figura 8 – ALTERA DE2-2C35



Fonte: Altera (2006)

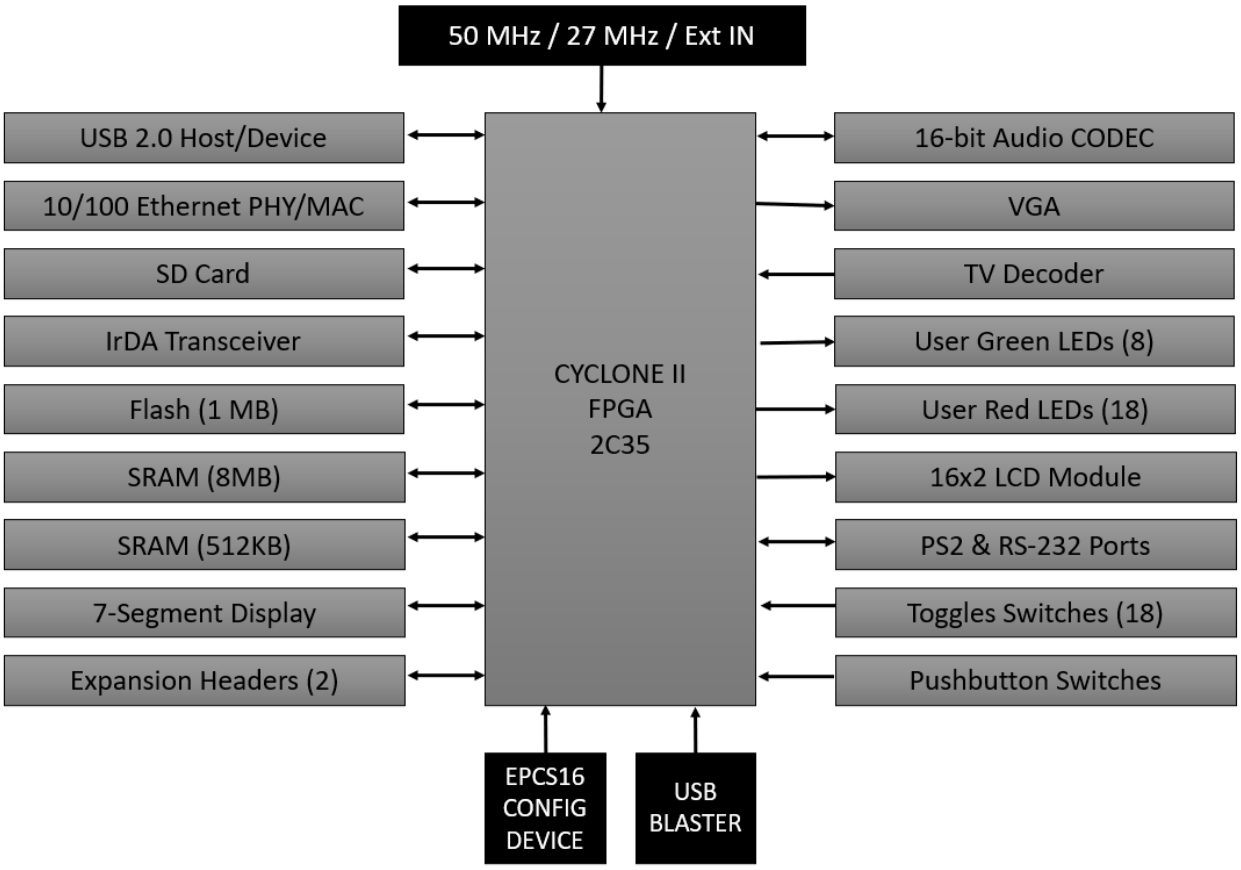
A placa DE2-2C35 pode desempenhar um papel de grande relevância no campo da automação, oferecendo recursos e componentes ideais para o desenvolvimento de projetos nesta área. Com ela, é possível implementar sistemas digitais que permitem o controle e o monitoramento de processos de forma eficiente. Com suas portas GPIO, conectores VGA, conectores de áudio, porta Ethernet, USB, RS-232 e PS/2, a placa oferece uma ampla variedade de opções de conectividade, possibilitando a integração com diversos dispositivos externos. Além das interfaces mencionadas, esta também possui recursos visuais essenciais para facilitar o controle e fornecer feedback visual em tempo real. O display LCD, os botões de pressão, as chaves de alternância e os LEDs são elementos interativos que permitem uma interação direta e intuitiva com o sistema automatizado. Essas interfaces visuais facilitam a supervisão e o controle das operações, proporcionando uma experiência mais eficiente (ALTERA, 2011).

A capacidade de programação flexível do FPGA Cyclone II EP2C35F672C6N permite a implementação de algoritmos e lógicas de controle personalizadas, adaptadas às necessidades específicas de cada sistema. Dessa forma, o kit de Desenvolvimento e Educação da Altera se destaca como uma ferramenta versátil e completa para o desenvolvimento de projetos e ensino. Sua aplicabilidade vai além do ensino acadêmico, abrangendo também a indústria e a pesquisa no campo da automação e controle de processos (ALTERA, 2011; ALTERA, 2006).

3.1 Diagrama de blocos da placa DE2-2C35

A Figura 9 apresenta o diagrama de blocos da placa DE2-2C35. NEsta, todas as conexões são feitas através do dispositivo Cyclone II FPGA. Logo, ele é responsável por fornecer uma plataforma de *hardware* programável e flexível, onde os diferentes componentes da placa podem ser conectados e configurados de acordo com as necessidades do projeto. (ALTERA, 2006).

Figura 9 – Diagrama de Blocos da Placa DE2-2C35



Fonte: Adaptado de Altera (2006)

3.2 Aplicações do DE2-2C35

O DE2-2C35 tem uma ampla gama de aplicações, sendo utilizado em diversos campos, como educação, pesquisa e desenvolvimento de sistemas digitais. Em ambientes acadêmicos, a placa é frequentemente empregada como uma ferramenta de ensino para a compreensão e experimentação prática de conceitos relacionados a sistemas digitais. Ela fornece recursos e componentes essenciais para a implementação de projetos, permitindo que os alunos explorem e apliquem os conhecimentos teóricos em um ambiente real (MATURANA, 2010).

Além disso, o DE2-2C35 também encontra aplicação na indústria, especialmente na área de automação e controle de processos. Sua capacidade de processamento digital de sinal e suas interfaces de entrada e saída versáteis permitem a construção de sistemas de monitoramento e controle eficientes. A placa pode ser integrada a sensores, atuadores e outros dispositivos para criar soluções personalizadas de automação, atendendo às necessidades específicas de cada aplicação. Para utilizá-lo, o usuário deve estar familiarizado com o Software Quartus II, que é utilizado para a programação e configuração do dispositivo. O Quartus II é uma ferramenta poderosa que permite ao usuário projetar, simular e implementar circuitos lógicos no DE2-2C35. É necessário entender os conceitos de design digital e ter conhecimento das linguagens de descrição de hardware, como VHDL ou Verilog, para programar o dispositivo com sucesso.(ALTERA, 2011).

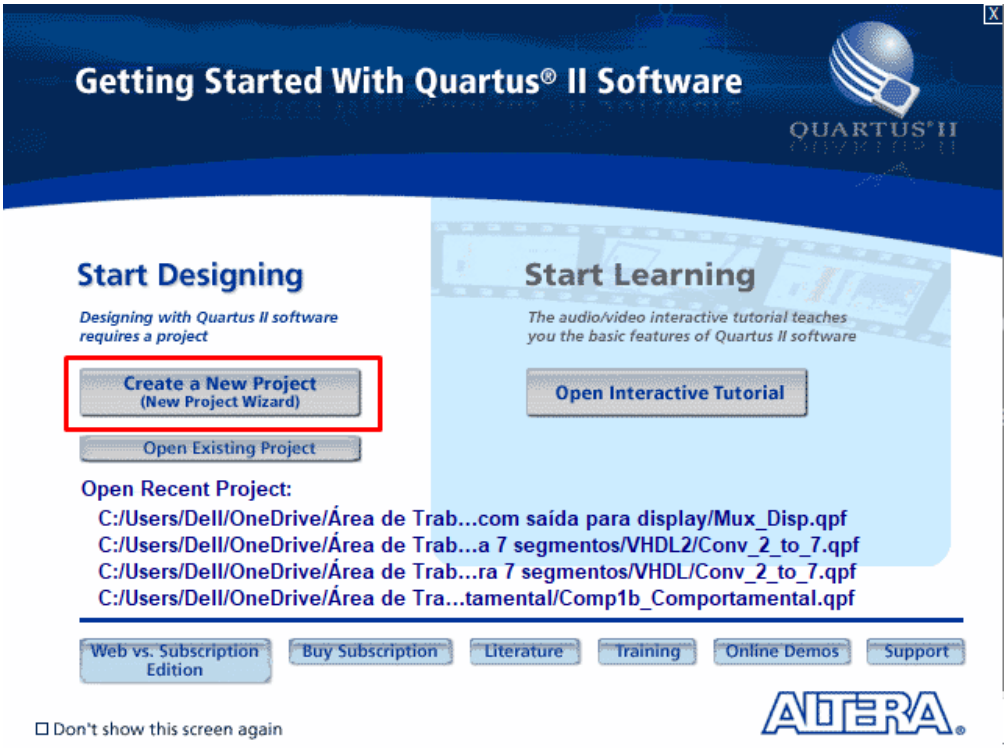
3.3 Quartus II

O software fornecido com a placa DE2 inclui o sistema de projeto assistido por computador (CAD) Quartus® II Web Edition e o processador integrado Nios® II. Também estão incluídos vários auxílios para ajudar estudantes e profissionais a experimentar os recursos do quadro, como tutoriais e aplicativos de exemplo. Diferente das placas DE2, os demais fabricantes de placas FPGA educacionais têm fornecido uma variedade de recursos de *hardware* e ferramentas de *software* CAD necessárias para implementar projetos nessas placas, mas muito pouco material foi oferecido que pudesse ser usado diretamente para fins de ensino (ALTERA, 2006).

3.3.1 Criação de novo projeto

Ao abrir Quartus II, a tela da figura 10 poderá ser visualizada. A opção "Open Project"e permite a edição e visualização dos projetos em andamento na área de trabalho principal. Além disso, o Quartus II oferece uma variedade de recursos e ferramentas para o desenvolvimento de projetos. Após a criação de um novo projeto, o usuário pode começar a projetar e implementar circuitos digitais utilizando a linguagem de descrição de hardware (HDL) de sua escolha.

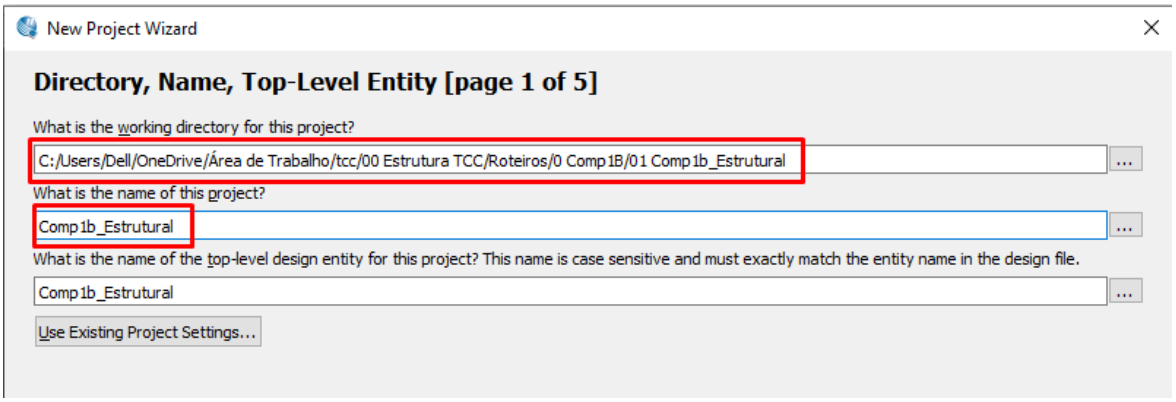
Figura 10 – Tela inicial do QuartusII



Fonte: Autoria própria.

Depois de *New Project Wizard*, a figura 11 pode ser visto o nome e local onde o projeto será salvo. Ao inserir o nome do projeto esta etapa é finalizada.

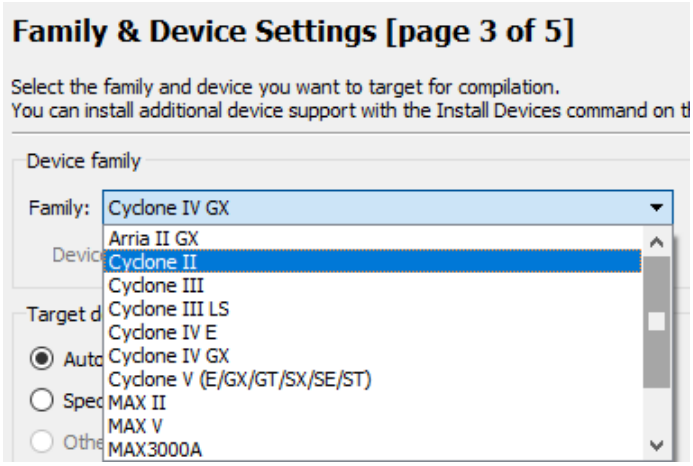
Figura 11 – Nomeação do projeto e local onde o mesmo será salvo



Fonte: Autoria própria.

Na tela 2 do processo de iniciação de projeto no Quartus, encontra-se a opção de inserir arquivos externos. Essa opção permite adicionar arquivos adicionais, como bibliotecas ou módulos pré-existentes, ao projeto em andamento. A pagina 3 é o local onde se escolhe a família do FPGA que será utilizado no projeto. Essa escolha é essencial, pois diferentes famílias de FPGA possuem características e recursos específicos, como número de pinos, capacidade de processamento, recursos de memória, entre outros (figura 12).

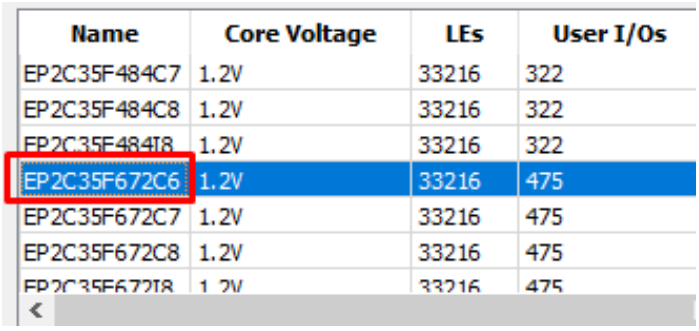
Figura 12 – Seleção da Familia Cyclone II



Fonte: Autoria própria.

Ainda na pagina 3 é selecionado o dispositivo específico (EP2C35F672C6) da família (figura 13).

Figura 13 – Seleção do FPGA

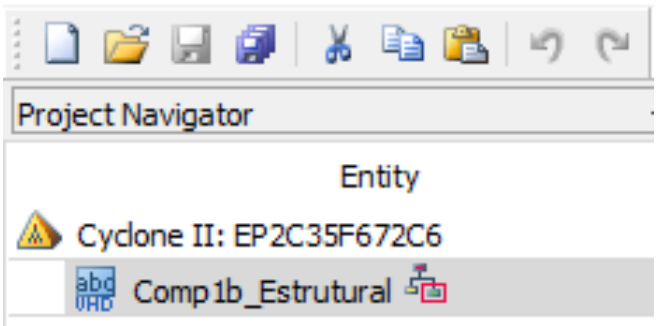


Name	Core Voltage	LEs	User I/Os
EP2C35F484C7	1.2V	33216	322
EP2C35F484C8	1.2V	33216	322
EP2C35F484I8	1.2V	33216	322
EP2C35F672C6	1.2V	33216	475
EP2C35F672C7	1.2V	33216	475
EP2C35F672C8	1.2V	33216	475
EP2C35F672I8	1.2V	33216	475

Fonte: Autoria própria.

Na página final pôde ser exibido o sumário com as especificações. Na lateral esquerda será fornecido o diretório do projeto, já com o dispositivo e arquivo de projeto (figura 14).

Figura 14 – Diretório do projeto

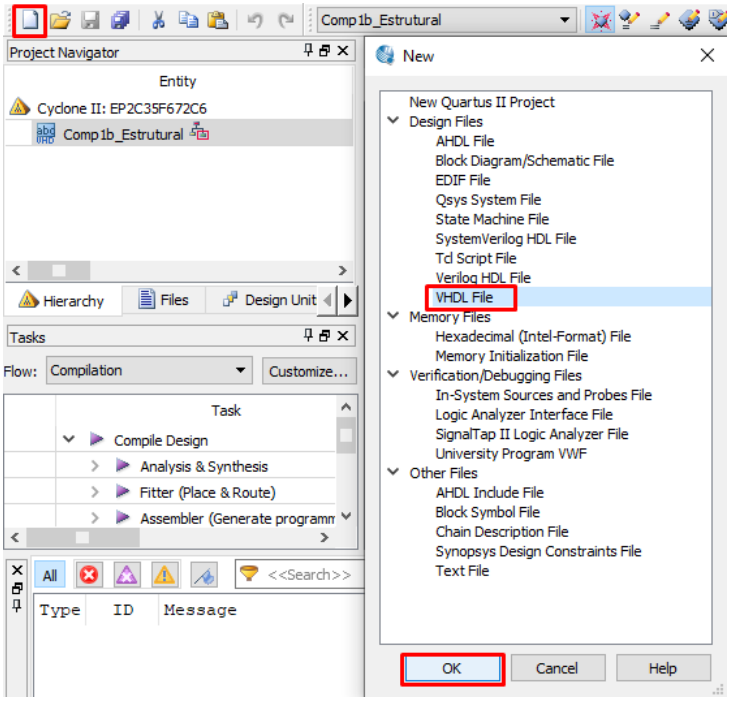


Fonte: Autoria própria.

3.3.2 Criação do arquivo VHDL

Na tela mostrada na figura 15 será criado um arquivo VHDL, onde será escrito o código a ser implementado, basta clicar em NEW >VHDL File > OK, como mostram os identificadores da figura.

Figura 15 – Criação de Arquio VHDL



Fonte: Autoria própria.

A entidade e a arquitetura são expressas na figura 16.

Figura 16 – Código VHDL do exemplo 2.2.3

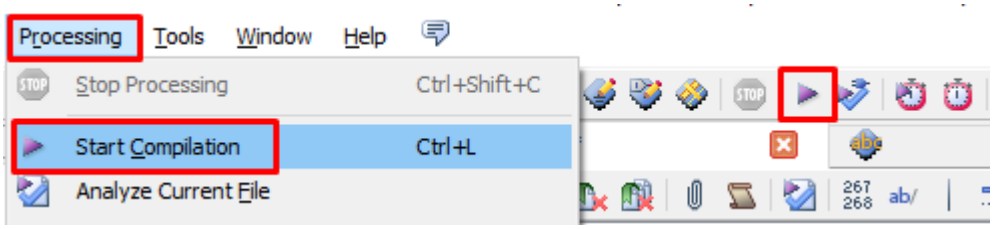
```
--Comparador1Bit_Estrurural

LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY Comp1b_Estrutural IS
    PORT (
        A, B : IN STD_LOGIC;
        S    : OUT STD_LOGIC
    );
END ENTITY Comp1b_Estrutural;

ARCHITECTURE Arq_Comp1b_Estrutural OF Comp1b_Estrutural IS
    SIGNAL L1 : STD_LOGIC;
BEGIN
    L1 <= A XOR B;
    S <= A AND L1;

END ARCHITECTURE Arq_Comp1b_Estrutural;
```

Fonte: Adaptado de Ordonez et al. (2003)

3.3.3 Compilador

Para iniciar o compilador, será necessário selecionar a opção *Processing* e, em seguida, escolher *Start Compilation*. Alternativamente, selecione o ícone indicado (figura 17). Assim será iniciado o processo de compilação do código VHDL, onde o Quartus II executa todas as análises, síntese e gerará a implementação para o chip específico. Essa etapa é crucial, pois é durante a compilação que o Quartus II verifica e otimiza o código VHDL, além de gerar os arquivos necessários para programar o chip de destino. Toda vez que for feita uma ação ou mudança a etapa de compilação deve ser executada.

Figura 17 – Iniciar Compilação do arquivo

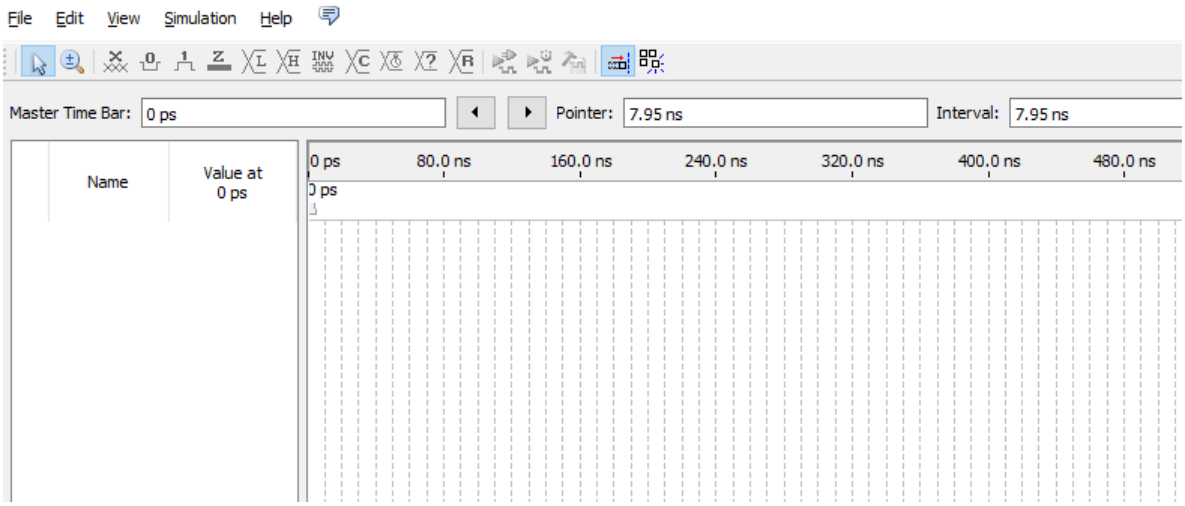


Fonte: Autoria própria.

3.3.4 Simulação

Para realizar a simulação de um projeto, proceda da seguinte forma: abra o software Quartus II e clique em *New file*. Em seguida, selecione a opção *University Program VWF* (Virtual Waveform). Essa escolha permitirá o acesso ao ambiente de simulação do Quartus II. Após iniciado, o usuário será apresentado a tela da figura 18, nesta são inseridos os componentes de Entrada e Saída (E/S) do projeto.

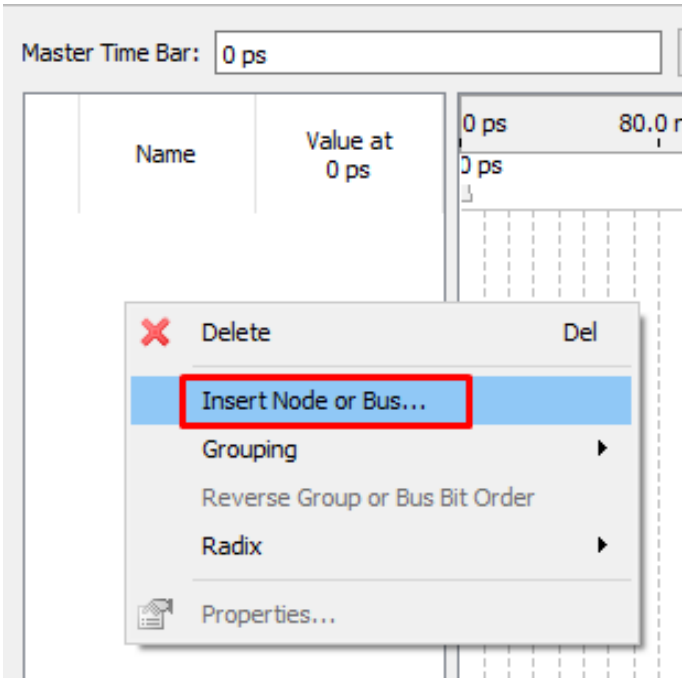
Figura 18 – Ambiente Virtual Waveform



Fonte: Autoria própria.

Para inserir as entradas e saídas, inicialmente clique com o botão direito do mouse na área indicada na figura 19, e selecione a opção *Insert Node or Bus*.

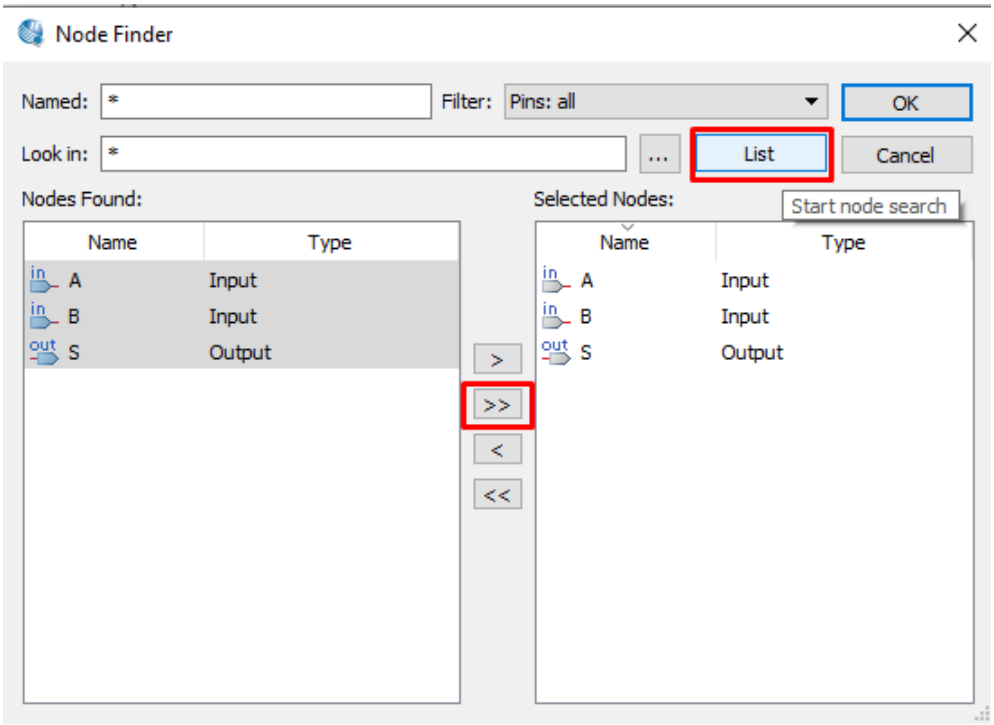
Figura 19 – Insert Node or Bus



Fonte: Autoria própria.

A figura20) expressa a função node finder, útil para exibir as entradas e saídas do projeto, por fim clique no simbolo ">>"e finalize com OK.

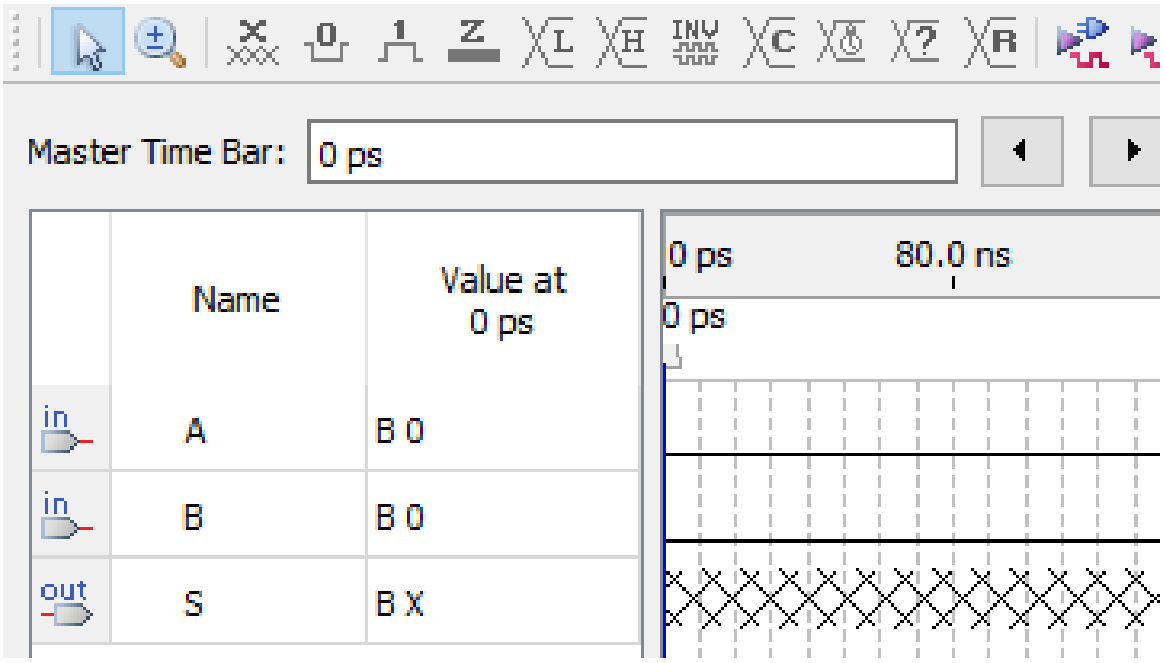
Figura 20 – Lista de E/S



Fonte: Autoria própria.

Após inserir as E/S o ambiente apresentará seguinte aparência (figura 21).

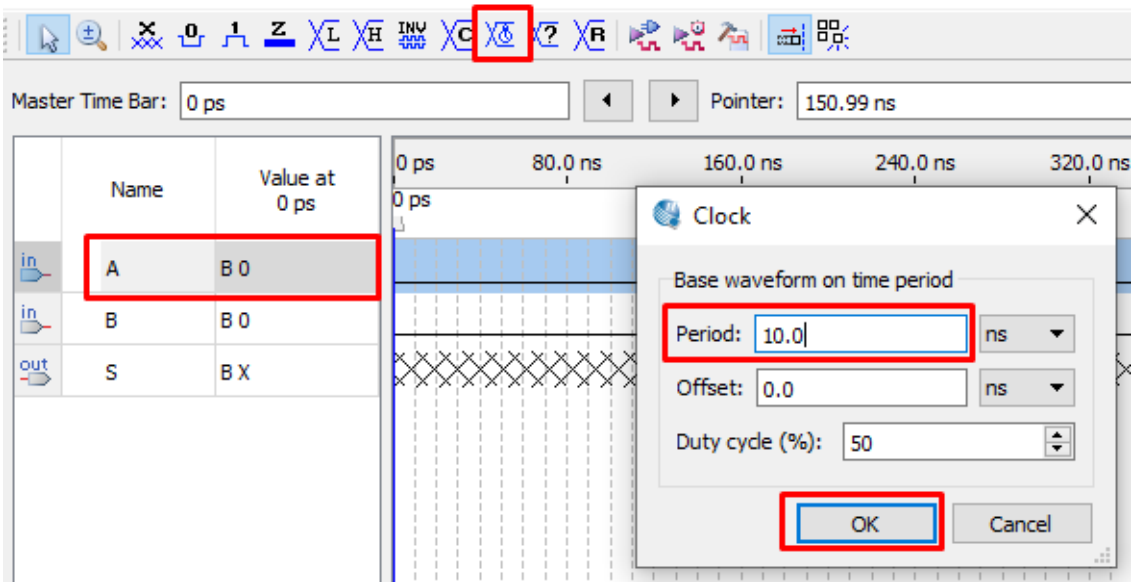
Figura 21 – Ambiente de simulação já com as E/S



Fonte: Autoria própria.

Arbitrariamente será escolhido um *Clock* de 10 ns para a entrada A, para isto basta clicar na área da primeira entrada, clicar no simbolo de relógio e definir o período desejado (figura 22). O mesmo procedimento deve ser aplicado a entrada B com diferença de que nesse fora aplicado *clock* de 30 ns.

Figura 22 – Definição dos *clocks* de Entrada

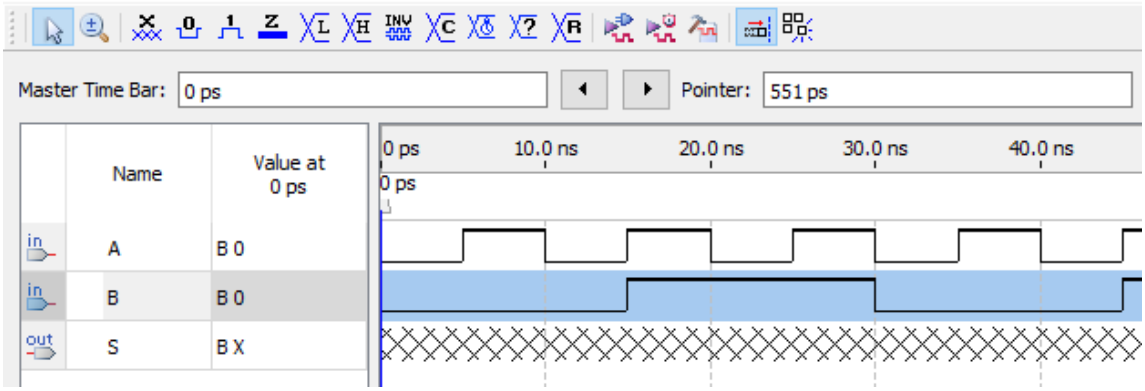


Fonte: Autoria própria.

Com os *clocks* inseridos, o ambiente ficará como mostrado abaixo (figura 23).

Neste passo a simulação será executada, bastando clicar sobre o simbolo indicado na figura 24.

Figura 23 – Entradas com *clock* definidos



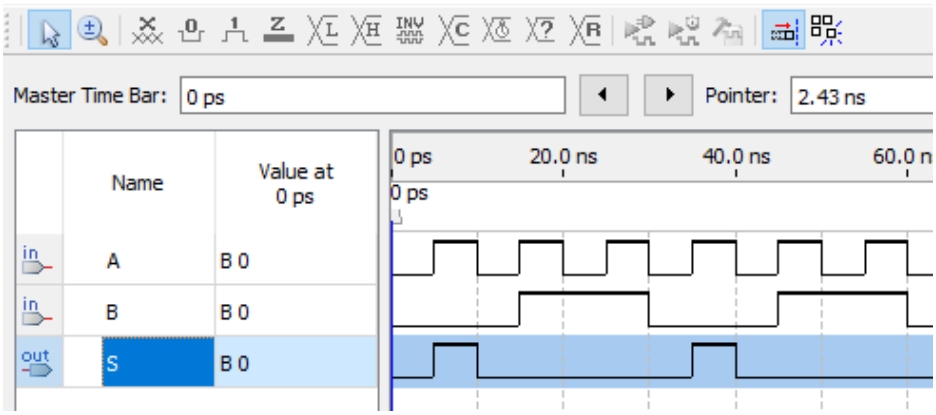
Fonte: Autoria própria.

Figura 24 – Execução da simulação

Fonte: Autoria própria.

A etapa final (figura 25) da simulação do circuito comparador de 1 bit simplificado é vista a seguir.

Figura 25 – Simulação do comparador de 1 bit simplificado



Fonte: Autoria própria.

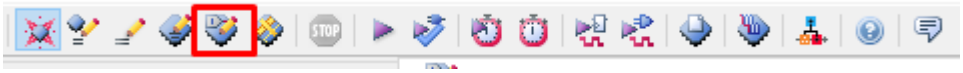
Nesse circuito, espera-se que a saída S esteja em nível lógico alto apenas quando as entradas A e B forem diferentes, e quando o valor de A for maior do que o de B. Essa condição é observada na etapa final da simulação, em que a saída S é ativada em nível lógico alto, somente quando as condições mencionadas são satisfeitas. Através dessa representação gráfica, é possível visualizar e verificar o correto funcionamento do circuito comparador de 1 bit simplificado durante a simulação.

3.3.5 Mapeamento de Pinos

A figura 26 indica o ícone para acesso ao ambiente de mapeamento de pinos, ou seja, onde as entradas e saídas terão sua correspondência com os pinos físicos do FPGA. A

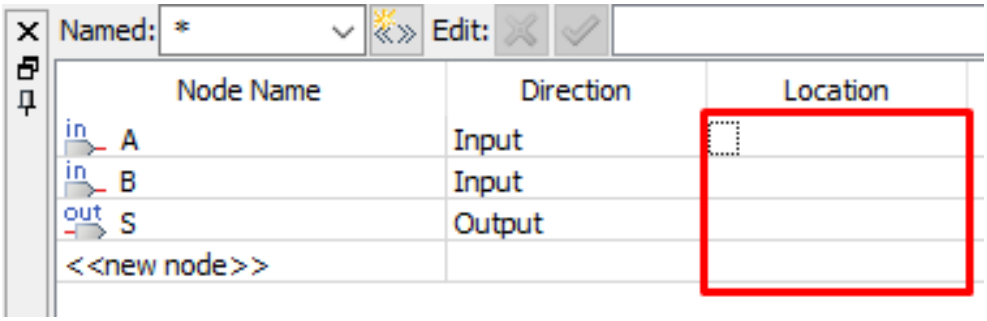
área na figura 27 indicada no quadrado vermelho, mostra o local de inserção de cada pino.

Figura 26 – Ícone para acessar o ambiente de mapeamento de pinos



Fonte: Autoria própria.

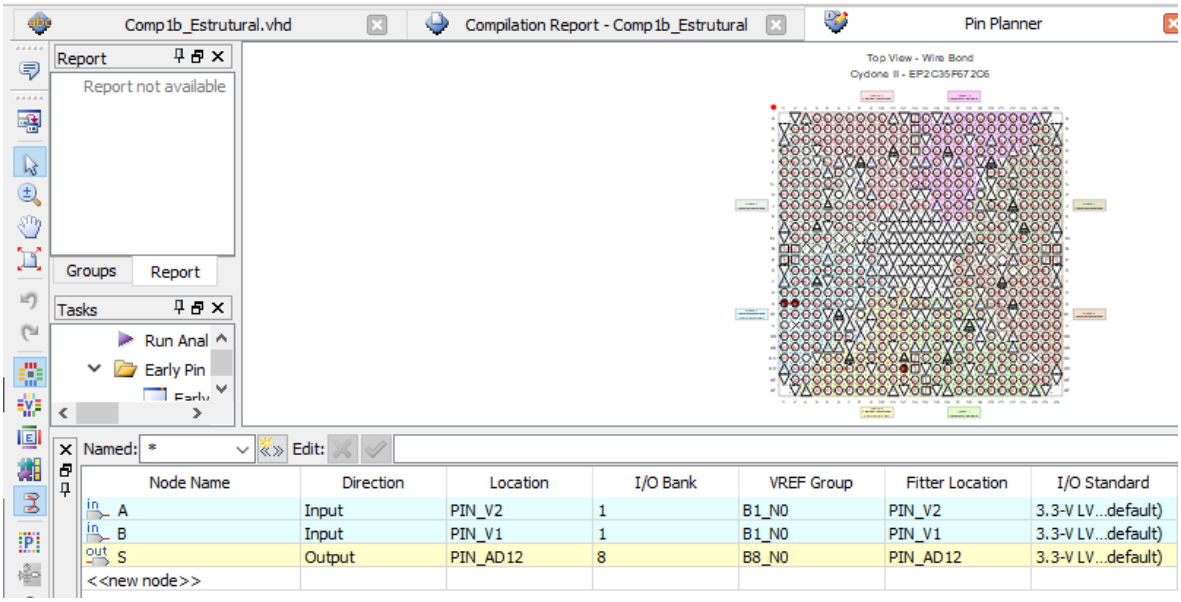
Figura 27 – Ambiente de mapeamento de pinos



Fonte: Autoria própria.

O anexo A fornece a correspondência entre os pinos e as respectivas funções, a figura 28 apresenta-nos a finalização deste passo.

Figura 28 – Mapeamento de pinos do comparador de 1 bit

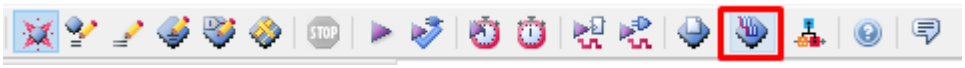


Fonte: Autoria própria.

3.3.6 Programmer

Depois que o design é sintetizado e transformado em uma representação lógica, o arquivo de configuração resultante (bitstream) precisa ser carregado no dispositivo alvo para que o circuito projetado seja executado. Para isto faz-se necessário a utilização do ambiente *programmer* que é iniciado através do ícone destacado na figura 29.

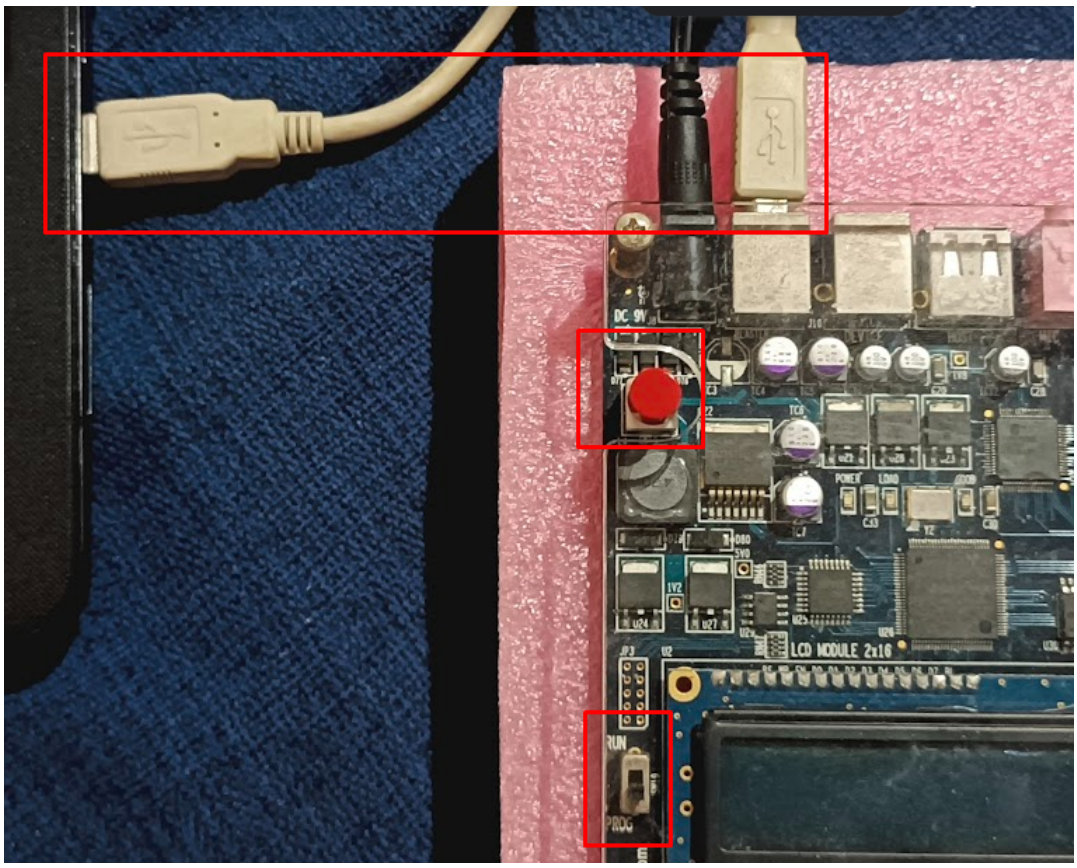
Figura 29 – Icpne para acesso ao ambiente Programmer



Fonte: Autoria própria.

A conexão do dispositivo lógico programável é executada usando a interface JTAG disponibilizada com o Kit DE2-2C35. O cabo de alimentação do dispositivo é ligado através do botão vermelho indicado na figura 30. A chave de seleção da figura 30 deve estar selecionada em *"RUN"* para receber a programação.

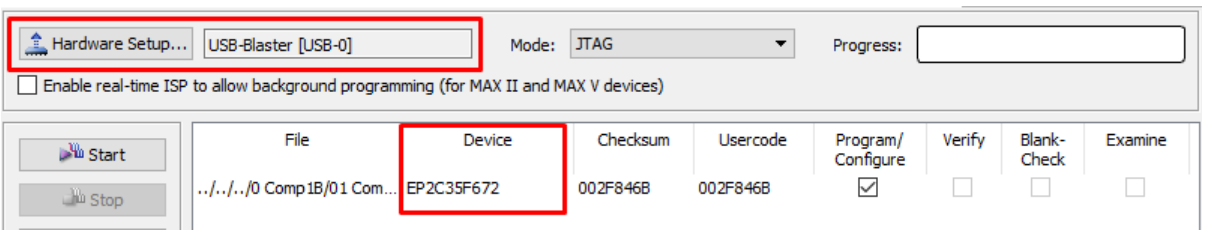
Figura 30 – Conexão do FPGA ao computador com o Quartus II



Fonte: Autoria própria.

No Quartus II, o *Hardware Setup* é indicado como *USB Blaster* [USB-0] (figura 31). Deverá ser indicado corretamente o dispositivo (EP2C35F672).

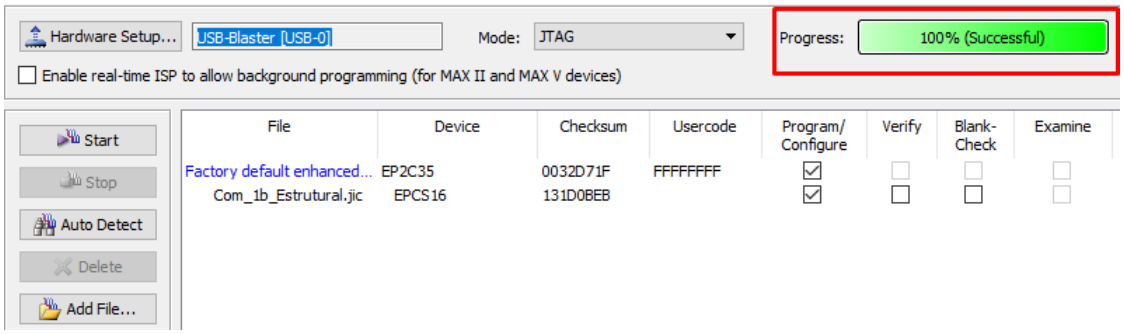
Figura 31 – Verificações finais do ambiente de programação



Fonte: Autoria própria.

Feitas as verificações, o processo é iniciado clicando em *START*, o andamento na barra *Progress*. A seguinte mensagem será informada após finalizado o processo (figura 32).

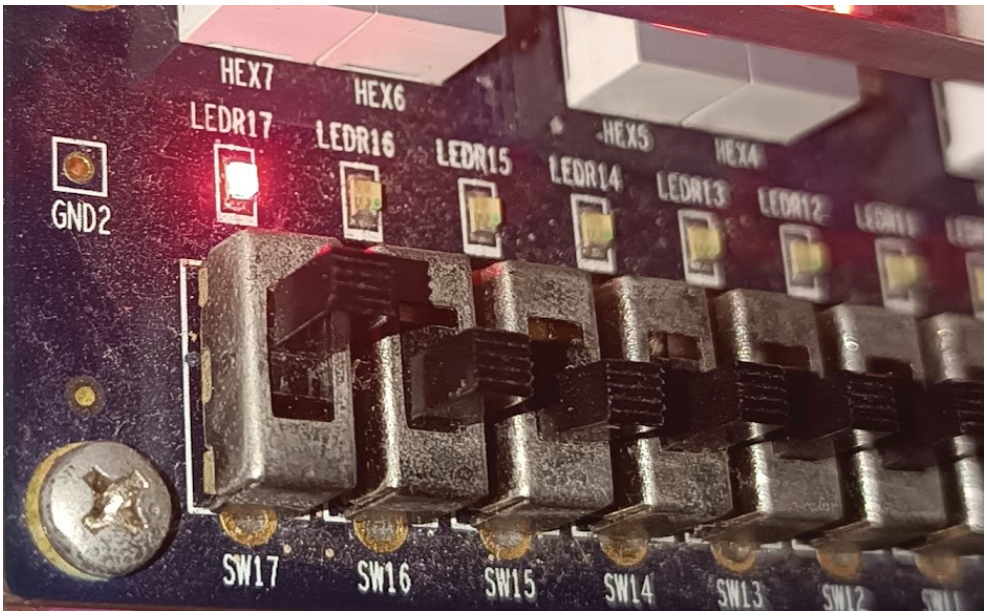
Figura 32 – Acompanhamento do processo de programação



Fonte: Autoria própria.

Com o DE2-2C35 desligado, o cabo JTAG é descontado, e em seguida o dispositivo ligado para teste de hardware programado (figura 33).

Figura 33 – Teste de hardware



Fonte: Autoria própria.

4 ROTEIROS EXPERIMENTAIS

O objetivo principal deste capítulo é aplicar a teoria da linguagem de descrição de hardware, VHDL, apresentada nos capítulos anteriores, com o intuito de promover o aprendizado da automação. Para atingir esse objetivo, foram desenvolvidos quatro roteiros que abrangem diferentes aspectos da lógica digital.

Ao longo do roteiro 1, o discente é motivado a realizar testes nas entradas e saídas do kit FPGA, bem como desenvolver e implementar portas lógicas fundamentais. O objetivo deste é familiarização do aluno com o funcionamento e as capacidades do FPGA, permitindo que ele compreenda como utilizar as interfaces e conexões básicas disponíveis.

O Roteiro 2, objetiva o desenvolvimento e implementação de circuitos lógicos combinacionais. Isso envolve a criação de circuitos que realizam operações lógicas simples, como combinações de portas lógicas. Esses circuitos não possuem elementos de memória e o resultado depende apenas das entradas atuais.

O Roteiro 3, aborda o desenvolvimento e implementação de circuitos lógicos sequenciais. Diferente dos circuitos combinacionais, os circuitos sequenciais possuem elementos de memória, como flip-flops, e o resultado depende tanto das entradas atuais quanto do estado anterior do circuito. Isso permite a criação de circuitos que armazenam informações e realizam operações sequenciais.

Com uma base sólida estabelecida, o roteiro 4 tem como objetivo final o desenvolvimento de uma aplicação em automação, aprimorando os elementos abordados nos roteiros anteriores. Essa proposta surge com a intenção de fornecer um guia prático para o desenvolvimento de projetos que utilizem o kit FPGA DE2-2C35. Vele ressaltar que cada roteiro, por sua vez, explora diferentes aspectos e etapas do processo de aprendizado e experimentação com o FPGA.

Para a execução de todos os roteiros propostos, serão necessários os seguintes itens:

a) Placa de desenvolvimento FPGA DE2-2C35;

b) Computador com o software Quartus II instalado, para programação e configuração;

c) Cabo de alimentação para a placa DE2-2C35.

d) Cabo JTAG para a sintetização do projeto no FPGA;

e) Anexo A para mapeamento dos pinos físicos.

4.1 Roteiro 1 - Entradas, saídas e portas lógicas básicas

É fundamental ter conhecimento sobre os periféricos de interface de um dispositivo, pois eles desempenham um papel essencial na interação e no funcionamento adequado do mesmo. Ao compreendê-los, é possível realizar corretamente a conexão de dispositivos externos, configurar e personalizar as opções de entrada e saída, solucionar problemas, explorar recursos avançados e garantir a segurança e a privacidade do dispositivo. Esta etapa é particularmente relevante em aplicações em que o FPGA precisa se comunicar com outros componentes, como sensores, atuadores ou outros sistemas digitais.

Dito isto, o foco deste primeiro roteiro é fornecer um primeiro contato entre o discente e o dispositivo através de entradas por chaves de alternância e saídas obtidas por LEDs. A elaboração de portas lógicas básicas, encontra-se como segundo passo na construção deste conhecimento. Pois estas são os blocos de construção para a implementação de uma infinidade de projetos.

4.1.1 Entradas e saídas

A placa DE2 é equipada com 18 chaves de alternância, também conhecidas como toggle switches, que são designadas como SW(0) a SW(17). Essas chaves podem ser utilizadas como entradas digitais. Além disso, a placa possui 18 LEDs vermelhos que são designados como LEDR(0) a LEDR(17) e podem ser utilizados como saídas digitais (ALTERA, 2011).

4.1.2 Portas lógicas básicas

Os CLBs (Blocos Lógicos Configuráveis) presentes no dispositivo FPGA são compostos principalmente por portas lógicas. Essa configuração permite a criação de projetos de diferentes complexidades, uma vez que as portas lógicas desempenham operações fundamentais, permitindo a combinação e manipulação de sinais digitais (ROTH, 2004)

4.1.3 Procedimentos Experimentais

Após os comentários acerca dos elementos bases da DE2-C35, pede-se que para as entradas/saídas e para as portas AND, OR e XOR, seja executado:

a) Exemplo de elaboração do algoritmo em VHDL contendo a entidade e a arquitetura que os descreva;

b) Simulação através do ambiente Virtual *Waveform* do Quartus II;

c) Mapeamento dos pinos físicos do FPGA;

d) Teste de *hardware* do projeto.

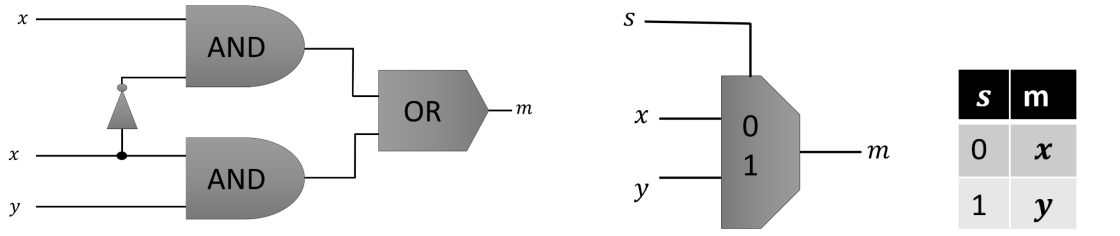
4.2 Roteiro 2 - Circuitos Combinacionais

Um circuito combinacional é composto por um conjunto de portas lógicas que determinam os valores das saídas diretamente, com base nos valores atuais das entradas. Esse tipo de circuito realiza uma operação de processamento de informações, que pode ser especificada por meio de um conjunto de equações booleanas. Cada combinação de valores de entrada pode ser vista como uma informação distinta, e cada conjunto de valores de saída representa o resultado da operação. A seguir serão apresentados três exemplos de circuitos combinacionais.

4.2.1 Multiplexadores

Uma das estruturas lógicas mais utilizadas em linguagens de descrição de *hardware* é o multiplexador (Mux). Um multiplexador é um componente digital muito útil, pois nos permite escolher entre diferentes sinais de entrada e direcionar o selecionado para a saída. A forma mais básica apresenta duas entradas, uma chave de seleção e uma saída. Na figura 34-a é mostrado um circuito de um multiplexador com 2 entradas e 1 saída (Mux 2:1). A representação simbólica é vista no item b da figura 34.

Figura 34 – Multiplexador 2 para 1



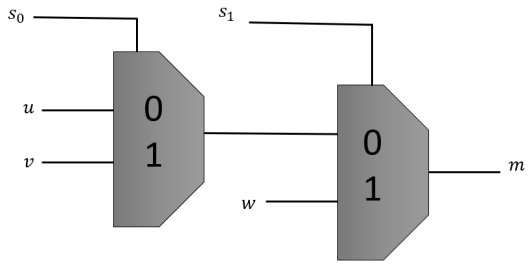
(a) Circuito lógico (b) Simbologia (c) Tabela verdade

Fonte: Autoria própria

Neste tipo de circuito, o sinal **s** seleciona qual entrada será transmitida para a saída **m**. Quando **s = 0** o sinal **x** será enviado para a saída **m**. Caso **s = 1**, o sinal **y** será enviado para a saída **m**. Para codificar o multiplexador em VHDL vamos usar duas chaves como as entradas **x** e **y**, um LED como saída **m** e um botão como chave seletora **s**.

Alternativamente um Multiplexador 3 para 1 (figura 35) pode ser implementado a partir de dois Multiplexador 2 para 1. O circuito utiliza duas entradas de seleção, sendo estas **s1** e **s0**. Perceba que as saídas **u,v,w** são obtidas conforme combinação das chaves de seleção **s1** e **s0**. Vale enfatizar que a configuração do elemento da figura 35-a, tem como base dois elementos já apresentados (figura 34-b), esta informação será importante para a descrição da arquitetura do elemento.

Figura 35 – Multiplexador 3 para 1



(a) Simbologia

s_1	s_0	m
0	0	u
0	1	v
1	0	w
1	1	w

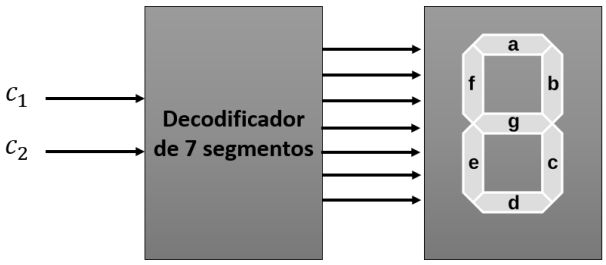
(b) Tabela Verdade

Fonte: Autoria própria

4.2.2 Decodificadores

Os decodificadores são circuitos lógicos utilizados para converter um código de entrada em um código de saída correspondente. Um dos tipos mais comuns é o decodificador de 7 segmentos. Esse decodificador é amplamente utilizado em displays de sete segmentos encontrados em equipamentos eletrônicos, como calculadoras, relógios digitais e painéis de instrumentos. A figura 36 ilustra um módulo decodificador de 7 segmentos que possui 2 bits c1 e c2 de entrada. Este decodificador fornece 7 saídas que são conectados a um display de 7 segmentos na placa DE2-C235. Cada segmento é iluminado quando o nível lógico 0 (baixo) é aplicado na saída do FPGA conectada ao segmento (FLOYD, 2009).

Figura 36 – Decodificador de 7 segmentos



(a) Simbologia

c1	c2	caractere
0	0	**d**
0	1	**E**
1	0	**0**
1	1	

(b) Tabela Verdade

Fonte: Adaptado de Floyd (2009)

A arquitetura de um decodificador de 7 segmentos depende principalmente da quantidade de segmentos que o display possui. Geralmente, um display de 7 segmentos é composto por 7 segmentos individuais (a, b, c, d, e, f, g) que podem ser ativados ou desativados para formar números, caracteres especiais ou letras. Seguindo a tabela verdade da figura 36, ao ser aplicado determinada entrada, o decodificador acionará os segmentos necessários para exibir o símbolo previamente definido na arquitetura, para o casa onde a entrada for '00' a saida corresponde sera a letra 'd', por exemplo. (FLOYD, 2009).

4.2.3 Procedimentos Experimentais

Para o Multiplexador e para o decodificador 2 bits para 7 segmentos, siga os seguintes passos:

a) Elaboração do algoritmo em VHDL contendo a entidade e a arquitetura que descreva a forma mais fundamental do componente;

b) Simulação através do ambiente Virtual *Waveform* do Quartus II;

c) Mapeamento dos pinos físicos do FPGA;

d) Teste de *hardware* do projeto.

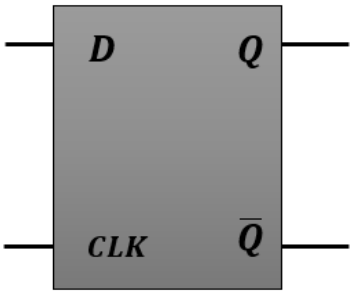
4.3 Roteiro 3 - Circuitos Sequenciais

Os circuitos sequenciais são componentes essenciais em sistemas digitais, pois possuem um estado interno que influencia a saída do circuito. Eles utilizam elementos de memória para armazenar informações, permitindo que a saída seja determinada não apenas pelas entradas presentes, mas também pelo estado atual do circuito. Entre os principais circuitos sequenciais, destacam-se os flip-flops, registradores e contadores (TOCCI; WIDMER; MOSS, 2010).

4.3.1 Flip-Flop e Registrador

A maioria dos sistemas digitais possuem elementos de memoria, dos quais o mais importante é o flip-flop (FF). Um flip-flop é um circuito eletrônico que pode armazenar um bit de informação. A forma mais simples possui uma entrada de dados (D), uma entrada de clock (CLK), e uma saída (Q), que representa o valor armazenado. A saída Q muda apenas quando ocorre uma borda de subida (ou descida, dependendo do tipo de FF) na entrada CLK. Existem diversos tipos de FF, dentre os quais o tipo D (figura 37) é um dos mais comuns .

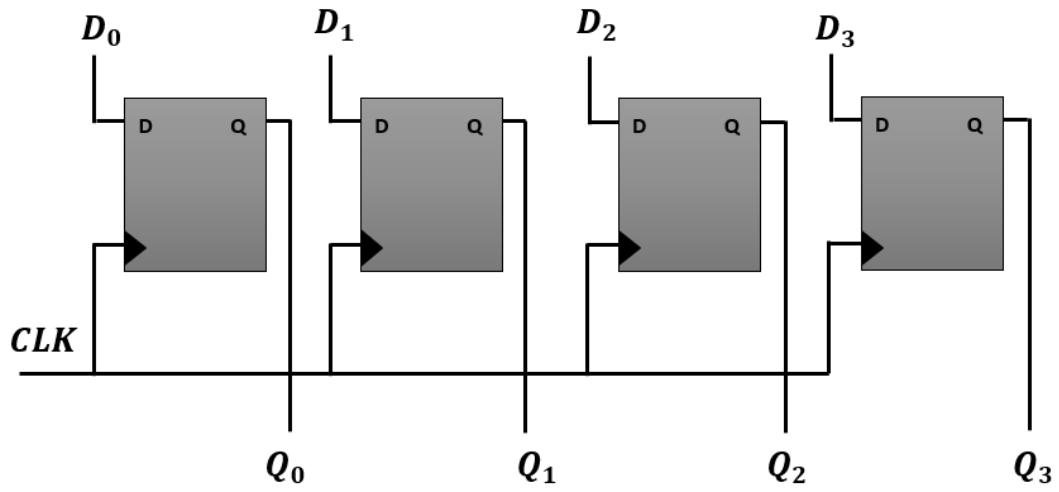
Figura 37 – Representação do flip-fLop tipo D



Fonte: Adaptado de Ordonez et al. (2003)

Os flip-flops são componentes essenciais na construção de registradores. Uma operação comum realizada em registradores é a transferência de dados entre eles, permitindo o fluxo de informações de um registrador para outro. O Registrador de 4 bits representado na figura 38 utiliza flip-flops do tipo D para armazenar os valores. Uma característica importante desse registrador é que o armazenamento dos dados ocorre de forma paralela. Isso significa que todos os bits são capturados simultaneamente (ORDONEZ et al., 2003).

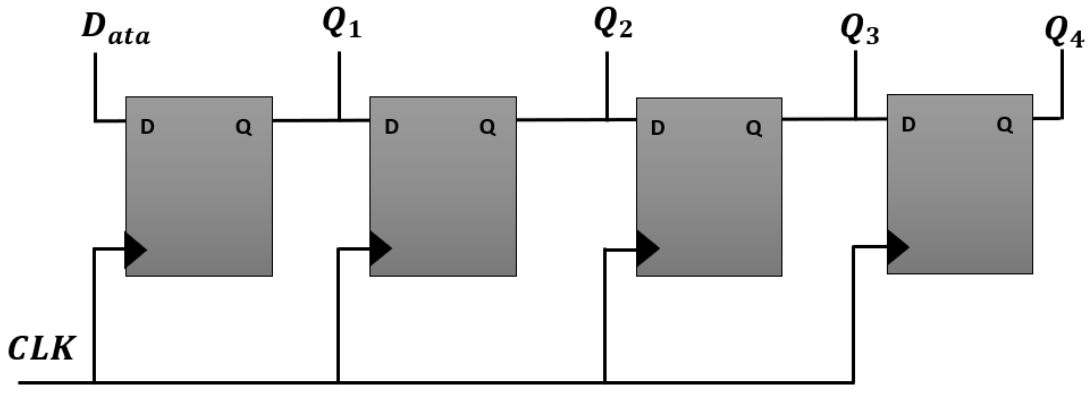
Figura 38 – Registrador paralelo de 4 bits



Fonte: Adaptado de ALTERA (2011)

Quando temos a intenção de estabelecer conexões externas com outros dispositivos periféricos, torna-se mais complicado e dispendioso realizar a conexão com registradores paralelos à medida que aumentamos o número de linhas de dados. Nesses casos, é comum recorrer aos registradores com entrada serial, nos quais são necessários n pulsos de *clock* para armazenar n bits. Esses registradores são conhecidos como *shift registers* ou registradores de deslocamento, pois a cada pulso de *clock*, os bits são deslocados para a frente. A figura 39 representa um exemplo de um registrador de deslocamento de 4 bits (TOCCI; WIDMER; MOSS, 2010).

Figura 39 – Registrador de deslocamento de 4 bits

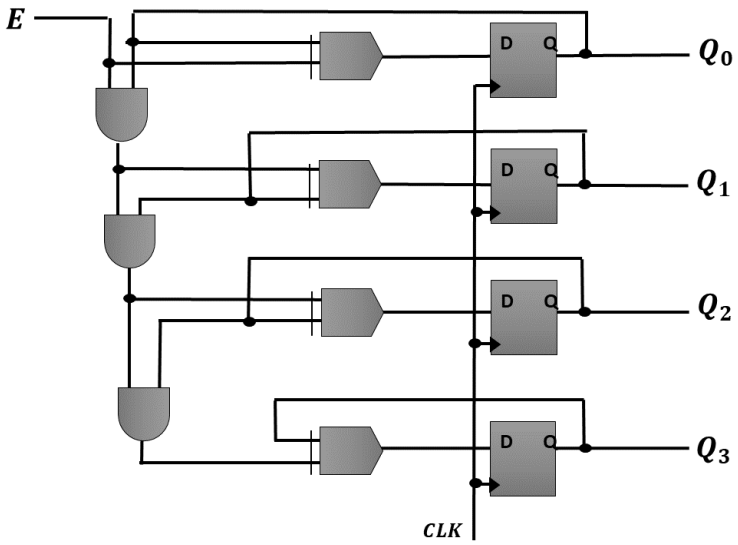


Fonte: Adaptado de ALTERA (2011)

4.3.2 Contador

Entre os circuitos sequenciais, os contadores são circuitos programáveis que geram sequências de valores predefinidos. Eles são usados em uma variedade de aplicações, como cronômetros, contadores de frequência e controladores de estado. A flexibilidade do FPGA permite que os contadores sejam personalizados e adaptados conforme necessário. Isso proporciona vantagens como redução de custos, tempo de desenvolvimento mais curto e a capacidade de atualização ou reprogramação dos contadores. Em resumo, os contadores em FPGA são componentes versáteis e poderosos para sistemas digitais, fornecendo funcionalidades de contagem e sequenciamento. A figura 40 apresenta a simbologia de um contador simplificado para 4 bits.

Figura 40 – Simbologia de um contador



Fonte: Adaptado de (FLOYD, 2009)

4.3.3 Procedimentos Experimentais

Para o registrador de deslocamento de 4 bits e para o contador síncrono de 4 bit, siga os seguintes passos:

a) Elabore o algoritmo em VHDL contendo a entidade e a arquitetura que descreva a forma mais fundamental do componente;

b) Execute simulação através do ambiente Virtual *Waveform* do Quartus II;

c) Forneça o mapeamento dos pinos físicos do FPGA;

d) Execute o teste de *hardware* do projeto.

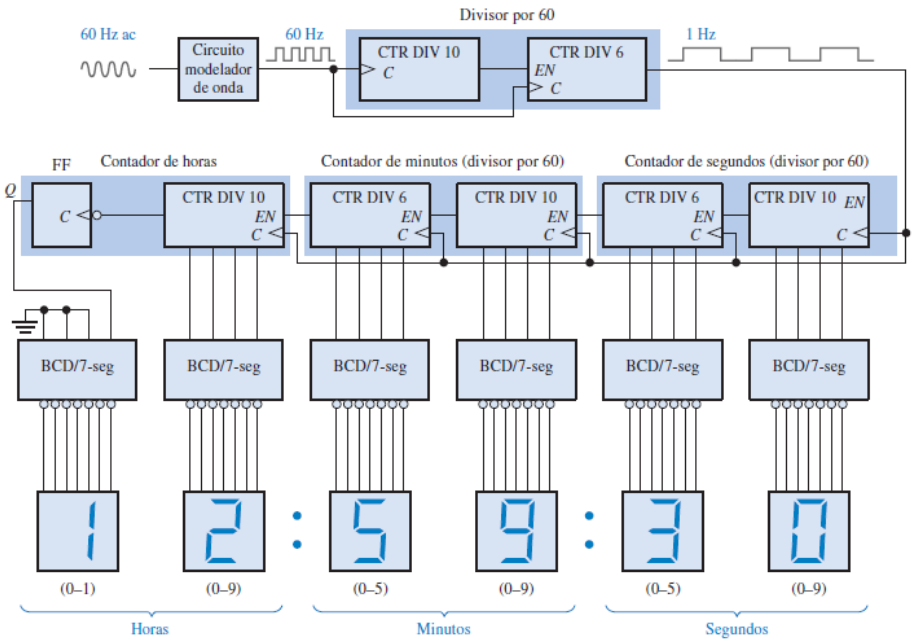
4.4 Roteiro 4 - Relógio digital

Projetos em HDL oferecem a opção de serem desenvolvidos em módulos separados, o que facilita a decomposição do problema e a organização do código. Essa abordagem modular permite dividir o projeto em partes menores e independentes, conhecidas como módulos ou componentes, cada um com uma função específica. Cada módulo pode ser projetado e testado individualmente, o que facilita a depuração e a manutenção do código. Além disso, essa abordagem promove a reutilização de código, uma vez que os módulos podem ser combinados de diferentes maneiras para construir sistemas mais complexos. Dito isto, este roteiro tem como premissa instigar o aluno a desenvolver projetos em blocos, e uni-los para a resolução de problemas maiores.

4.4.0.1 Definição do problema

Na pagina 481 do livro "Sistemas digitais - Fundamentos e aplicações", Floyd (2009) apresenta o exemplo de um relógio digital (figura 41). Logo, como prática de aprendizado, este roteiro propõe o desenvolvimento e implementação em hardware deste relógio. Ao desenvolve-lo, é possível utilizar recursos de temporização e sincronização disponíveis no FPGA para garantir a precisão e a estabilidade. É mister ressaltar que componentes já implementados nos três últimos roteiros podem e devem ser utilizados, pois componentes como contadores, decodificadores e multiplexadores são utilizados para controlar e exibir os dígitos das horas, minutos e segundos. Por fim, o relógio deverá mostrar as horas, minutos e segundos, e o usuário deverá ser capaz de realizar pequenos comandos manuais.

Figura 41 – Diagrama lógico simplificado para um relógio digital de 12 horas



Fonte: Floyd (2009)

5 RESULTADOS

5.1 Roteiro 1 - Entradas, saídas e portas lógicas básicas

5.1.1 Entradas e saídas

Para que cada chave fosse utilizada para controlar um LED específico, era conveniente representá-los como vetores em VHDL. Para fins de simplicidade, foi realizado o acionamento de 5 chaves (SW) correspondentes a 5 LEDs (LEDR). No entanto, é importante ressaltar que o processo de acionamento para os 18 elementos era semelhante. O quadro 15 apresentava o código desenvolvido em VHDL.

Quadro 15 – Código para acionamento dos LEDs da placa DE2-C235

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY SW_LEDR IS
    PORT(
     SW : IN std_logic_vector(4 DOWNTO 0);
   LEDR : OUT std_logic_vector(4 DOWNTO 0)
        );
END SW_LEDR;

ARCHITECTURE ARQ_SW_LEDR OF SW_LEDR IS
   BEGIN
      LEDR <= SW;
END ARQ_SW_LEDR;
```

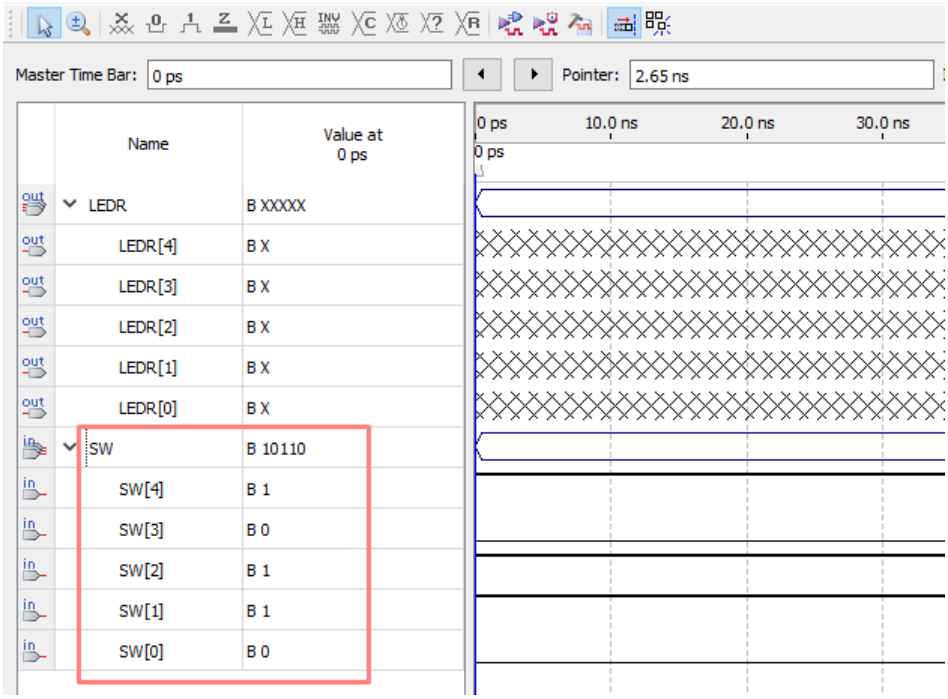
Fonte: Autoria própria

A entidade deste algoritmo informa que SW é um vetor que representa 5 portas lógicas de entrada. Enquanto que LEDR representa um vetor com 5 portas lógicas de saída. A arquitetura descreve o funcionamento destas portas atribuindo a cada entrada, uma saída. Esta correspondência é vista abaixo:

$$\begin{aligned} &\text{LED}(0) <= \text{SW}(0) \\ &\quad\ldots \\ &\text{LED}(4) <= \text{SW}(4) \end{aligned}$$

Uma vez que a entidade e a arquitetura estejam definidas, o próximo passo foi realizar a simulação por meio do ambiente Virtual *Waveform*. Nesse estágio, um valor de entrada arbitrário foi atribuído, por exemplo, "10110", indicando que apenas SW(4), SW(2) e SW(1) estavam com nível lógico alto (figura 42). Isso permitia verificar o comportamento do circuito e observar como as saídas eram afetadas pela escolha de valores de entrada.

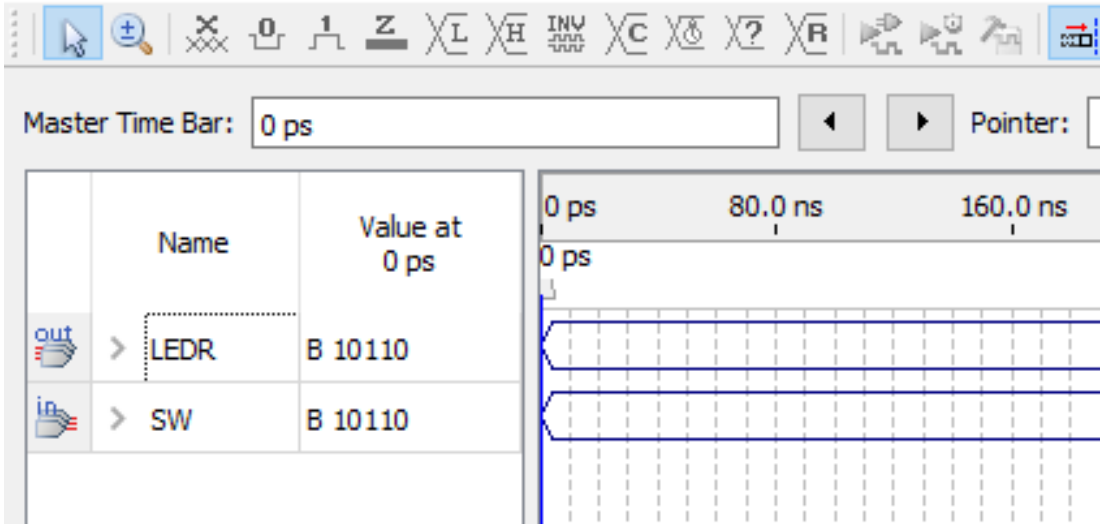
Figura 42 – Pré-simulação acionamento de LEDs



Fonte: Autoria própria

Isso resulta em uma saída idêntica a entrada (figura 43), que neste caso era de se esperar, uma vez que cada entrada controla uma saída respectiva.

Figura 43 – Simulação acionamento de LEDS



Fonte: Autoria própria

5.2 Circuito Combinacional

5.2.1 Multiplexadores

O circuito do multiplexador 3 para 1 proposto na figura 35-a, apresentou tres entradas de sinal (**u, v, w,**), duas entradas de controle (**s0** e **s1**), e uma saída (**m**). Estes elementos são listados nas portas da entidade apresentada no quadro 19.

Quadro 19 – Entidade do elemento Mux 3:1

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY mux_3to1 IS
    PORT(
        u, v, w : IN std_logic;
         s0, s1 : IN std_logic;
              m : OUT std_logic
         );
END mux_3to1;
```

Fonte: Autoria própria

O passo seguinte foi a definição de uma arquitetura. Por meio da estrutura do elemento Mux 3:1 obtém-se a saída final que é composta pela seleção entre a entrada **w** e a seleção que vem do primeiro multiplexador, logo o quadro 20 apresenta a estrutura da arquitetura.

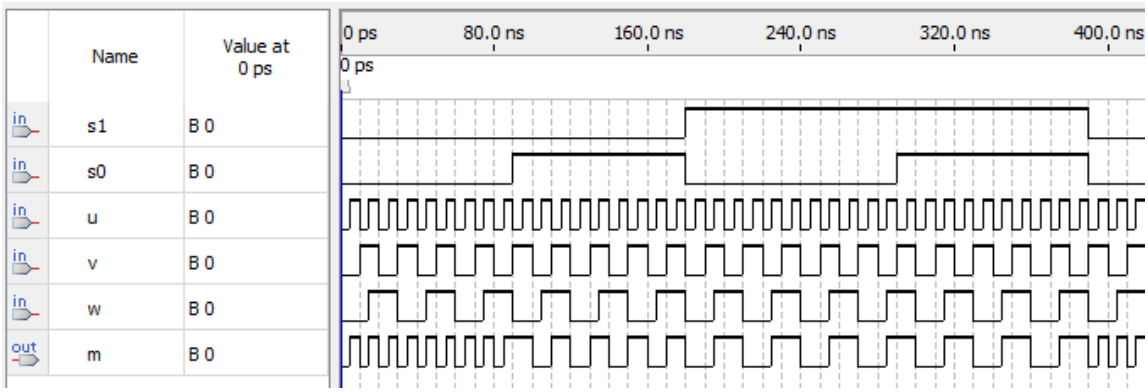
Quadro 20 – Arquitetura do elemento Mux 3:1

```
ARCHITECTURE Arq_ mux_3to1 OF mux_3to1 IS
   SIGNAL m1 : STD_LOGIC;
       BEGIN
          m1 <= ((NOT (s0) AND u) OR (s0 AND v));
           m <= (NOT (s1) AND m1) OR (s1 AND w);
END Arq_mux_3to1r;
```

Fonte: Autoria própria

A figura 47 apresenta a simulação do elemento proposto. Quando as chaves seletoras **s0** e **s1** são configuradas como "00", a saída selecionada corresponde à entrada **u**. Da mesma forma, quando as chaves estão configuradas como "01", "10"ou "11", as saídas selecionadas correspondem às entradas **v** e **w**, respectivamente.

Figura 47 – Simulação Multiplexador 3 para 1



Fonte: Autoria própria

Com os testes finalizados via simulação, a etapa seguinte foi a realização do mapeamento dos pinos, também conhecido como *pinout*, a fim de programar o código fornecido nos quadros 19 e 20. Para facilitar esse processo, o quadro 21 é fornecido, detalhando essa atividade. O *pinout* fornece informações sobre a disposição dos pinos físicos do FPGA frente aos elementos lógicos do circuito programado, permitindo uma correspondência adequada entre as entradas e saídas do código e as conexões físicas do dispositivo.

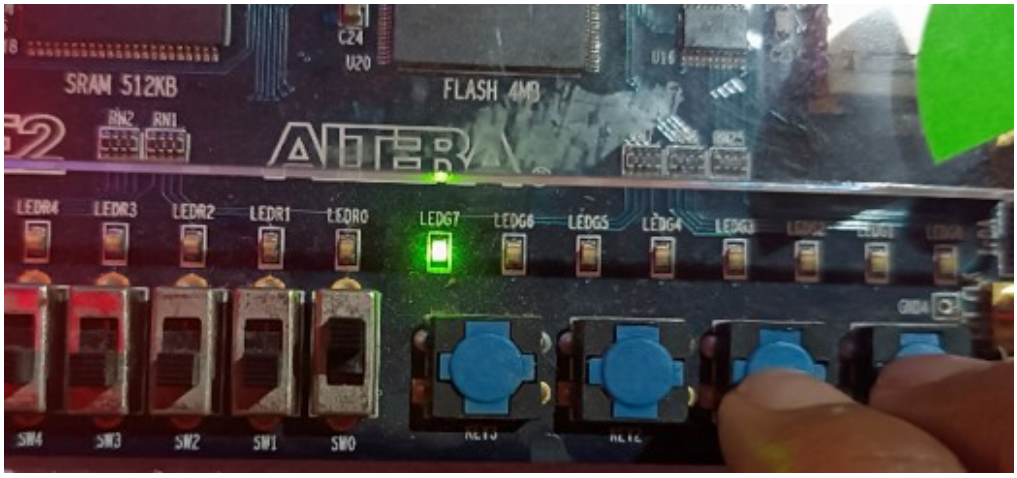
Quadro 21 – Conexão entre os sinais VHDL e os pinos do FPGA para o Mux 3-to-1

Variável VHDL	**Correspondente Físico**	**Pinos na Placa**
s0	KEY [0]	PIN_G26
s1	KEY [1]	PIN_N23
u	SW [0]	PIN_N25
v	SW [1]	PIN_N26
w	SW [2]	PIN_P25
m	LEDG [7]	PIN_Y18

Fonte: Autoria própria

Em seu estado natural, as chaves *pushbuttons* possuem comportamento normalmente fechado (NF), logo estes encontram-se em nível lógico alto quando não solicitadas. Dessa forma, a figura 48 apresenta o teste de hardware para o multiplexador desenvolvido. Quando as chaves KEY(0) e KEY(1) são solicitadas (figura 48-a), o comando de controle "00"é informado, selecionando assim o sinal **u**. Ao pressionar KEY(1), o controle correspondente a "01"é fornecido, selecionando a entrada **v** (figura 48-b). Os chaveamento "10"e "11"selecionam a entrada **w** (figura 48-c). Esse teste permite verificar a implementação do Multiplexador 3 para 1, garantindo que a seleção de sinais de entrada esteja ocorrendo conforme o esperado de acordo com as chaves de controle pressionadas.

Figura 48 – Multiplexador 3 para 1



(a) Seleção da entrada u

(b) Seleção da entrada v

(c) Seleção da entrada w

Fonte: Autoria própria

5.2.2 Decodificadores

Uma possibilidade de descrição para a entidade do decodificador de 2 bits para 7 segmentos, conforme ilustrado na figura 36-b, é apresentado no quadro 22. As portas lógicas utilizadas nessa entidade foram desenvolvidas como vetores, onde **C** corresponde a um vetor de entrada de dois estágios, e a saída **D** é um vetor de 7 estágios. Essa abordagem de vetorização permite um processamento eficiente dos sinais e facilita a manipulação dos segmentos do decodificador.

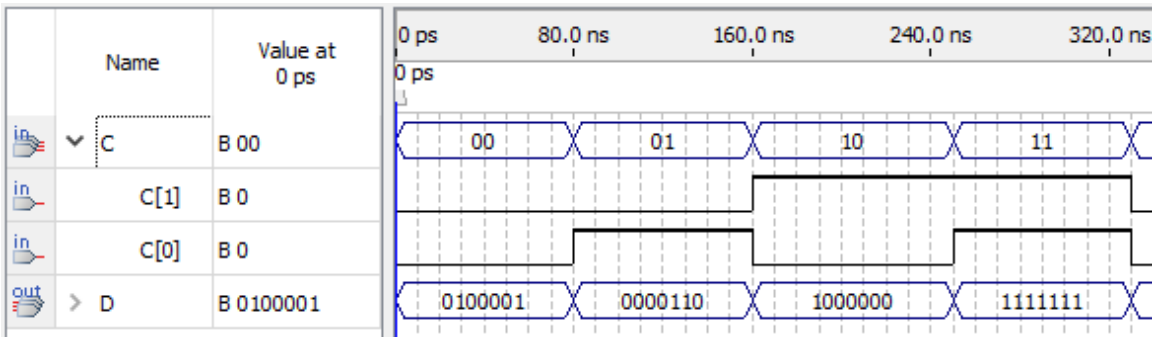
Quadro 22 – Entidade decodificador 2 bits para 7 segmentos

```
ENTITY deco_2_to_7 IS
    PORT(
C : IN std_logic_vector(1 DOWNTO 0);
D : OUT std_logic_vector(6 DOWNTO 0));
END deco_2_to_7;

ARCHITECTURE Arqt_deco_2_to_7 OF deco_2_to_7 IS
BEGIN
    D <= "0100001" WHEN C="00" ELSE
         "0000110" WHEN C="01" ELSE
         "1000000" WHEN C="10" ELSE
         "1111111" WHEN C="11";
END Arqt_Conv_2_to_7;
```

Fonte: Autoria própria

A descrição comportamental do circuito desempenha um papel fundamental na definição de sua arquitetura. É a partir dessa descrição que o projeto do circuito pode ser guiado, permitindo a criação de uma estrutura que atenda aos requisitos de funcionalidade. Podemos fazer esta verificação de comportamento ao longo do tempo por meio da simulação proposta na figura 49.

Figura 49 – Simulação decodificador para 7 segmentos



Fonte: Autoria própria

Após obter os resultados da simulação, o passo seguinte foi realizar o mapeamento dos pinos (quadro 23). Por ser tratar de um vetor, D possui 7 correspondências, se uma destas for ligada incorretamente, o display não mostrará o resultado esperado.

Quadro 23 – Conexão entre os sinais VHDL e os pinos do FPGA para o Mux 3-to-1

Variável VHDL	**Correspondente Físico**	**Pinos na Placa**
C0	SW[16]	PIN_V1
C1	SW[17]	PIN_V2
D[6]	HEX6[6]	PIN_M4
D[5]	HEX6[5]	PIN_M5
D[4]	HEX6[4]	PIN_M3
D[3]	HEX6[3]	PIN_M2
D[2]	HEX6[2]	PIN_P3
D[1]	HEX6[1]	PIN_P4
D[0]	HEX6[0]	PIN_R2

Fonte: Autoria própria

Com a simulação e o mapeamento dos pinos concluídos, o próximo passo foi a realização do teste de hardware. Essa etapa é crucial para verificar o funcionamento efetivo do decodificador de 2 bits para 7 segmentos no FPGA. Através dos testes realizados, é possível validar o circuito implementado e garantir sua correta operação. A figuras 50, apresenta o resultado dos testes realizados, mostrando o comportamento esperado do decodificador para diferentes combinações de entradas. Esses testes são essenciais para confirmar a eficiência e confiabilidade do circuito em situações reais de aplicação.

Figura 50 – Teste de *hardware* do decodificador de 2 bits para 7 segmentos.

Fonte: Autoria própria

5.3 Circuito Sequencial

5.3.1 Flip-Flop e Registrador

O quadro 24 apresenta uma possibilidade de algoritmo VHDL para a descrição do flip-flop tipo D. A entidade denominada **FF_D** é composta pelas portas de entrada de dados (D) e *clock* (clk), e pela saída (Q) todas do tipo de dados STD_LOGIC. A arquitetura contem o processo que faz uso da metodologia de desenvolvimento comportamental.

Quadro 24 – Código para implementação de FF-D

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY FF_D IS
 PORT(
    D, CLK : IN std_logic;
    Q : OUT std_logic
    );
END FF_D;

ARCHITECTURE ARQ_FF_D OF FF_D IS
    BEGIN
        PROCESS (CLK)
        BEGIN
            IF (CLK'EVENT AND CLK ='1') THEN
            Q <= D;
            END IF;
        END PROCESS;
END ARQ_FF_D;
```

Fonte: Adaptado de Roth (2004)

O valor da entrada **D** é armazenado e atualizado na saída **Q** quando ocorre uma borda de subida do sinal de *clock* **CLK**. Isso significa que o valor presente na entrada **D** só é transferido para a saída **Q** quando ocorre uma transição do sinal de *clock* de nível baixo para nível alto. Esta abordagem foi aproveitada no processo de descrição do registrador de deslocamento de 4 bits. O quadro 25 expressa a entidade desenvolvida.

Quadro 25 – Descrição da entidade do Registrador de deslocamento de 4 bits

```
ENTITY SHIFT_REG IS
    PORT(
     D,CLK, RTS :IN std_logic;
      LED       :OUT std_logic_vector(9 DOWNTO 0)
      );
END SHIFT_REG ;
```

Fonte: Autoria própria

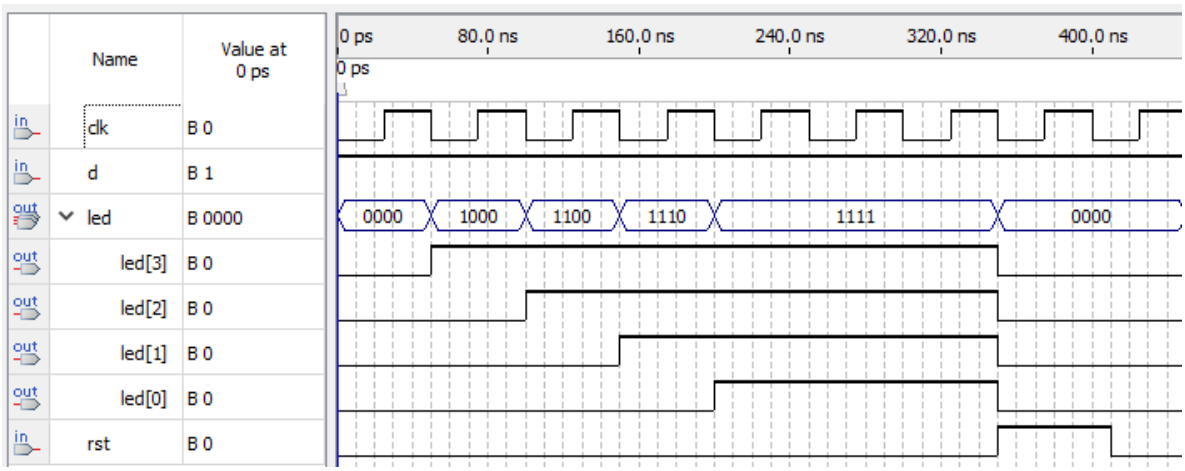
Quadro 26 – Descrição da arquitetura do Registrador de deslocamento de 4 bits

```
ARCHITECTURE ARQ_SHIFT_REG OF SHIFT_REG IS
BEGIN
  PROCESS(D, CLK, RST)
    VARIABLE AUX : STD_LOGIC_VECTOR(3 DOWNTO 0) := (OTHERS => '0');
        BEGIN
            IF RST = '1' THEN
                AUX := (OTHERS => '0');
            ELSIF FALLING_EDGE(CLK) THEN
                AUX(2 DOWNTO 0) := AUX(3 DOWNTO 1);
                AUX(3) := D;
            END IF;
            LED <= AUX;
        END PROCESS;
END ARCHITECTURE;
```

Fonte: Autoria própria

A arquitetura do quadro 26 implementa o comportamento do registrador proposto. A variável 'AUX' é usada como uma variável interna para armazenar os valores do registrador. Dentro do processo, há uma condição 'IF' para verificar se o sinal de *reset* (RST) está ativo. Se sim, a variável 'AUX' é redefinida para '0'. Caso contrário, há uma verificação de borda de descida do sinal de *clock* (FALLING_EDGE(CLK)). Se a borda de descida for detectada, os bits são deslocados para a direita e o bit de entrada (D) é armazenado no bit mais significativo. Por fim, o valor armazenado em 'AUX' é atribuído à saída LED, representando o estado atual do registrador. Estas definições podem ser verificada por meio da simulação apresentada na figura 51.

Figura 51 – Simulação registrador de deslocamento



Fonte: Autoria própria

Após a conclusão da simulação e verificação do comportamento correto do circuito, a próxima etapa é a transição para a parte de pinout, que envolve o mapeamento dos sinais do design para os pinos físicos do dispositivo alvo (quadro 27).

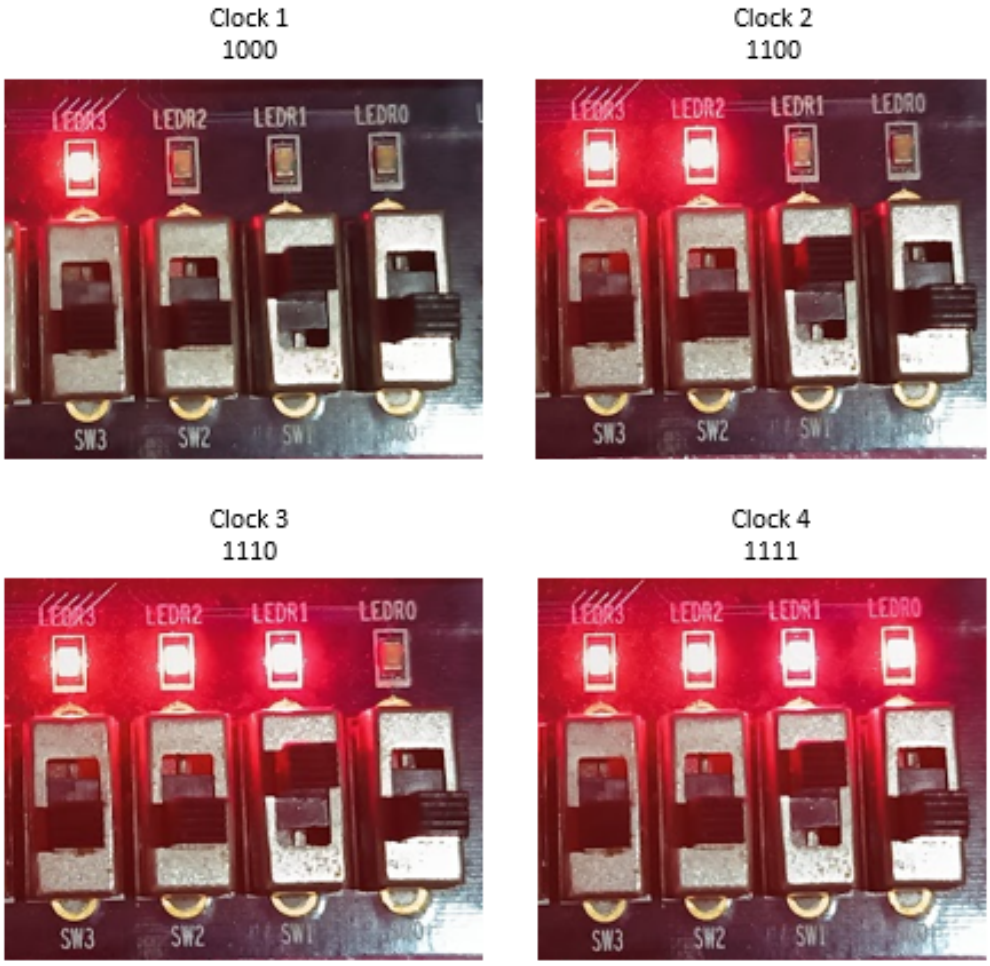
Quadro 27 – Conexão entre os sinais VHDL e os pinos do FPGA para o Registrador

Variável VHDL	**Correspondente Físico**	**Pinos na Placa**
CLK	KEY[0]	PIN_G26
RST	SW [0]	PIN_N25
D	SW [1]	PIN_N26
LED [3]	LEDR [3]	PIN_AC22
LED [2]	LEDR [2]	PIN_AB21
LED [1]	LEDR [2]	PIN_AF23
LED [0]	LEDR [0]	PIN_AE23

Fonte: Autoria própria

Uma vez que os pinos apropriados foram identificados, é necessário realizar a conexão física do circuito, seja por meio de soldagem, encaixe ou outro método de conexão apropriado para o dispositivo alvo. Como a DE2-2C35 já apresenta os itens necessários pode ser feito o teste de *hardware* (figura 52).

Figura 52 – Teste de *hardware* do Registrador de deslocamento de 4 bits



Fonte: Autoria própria

5.3.2 Contador

A entidade presente no quadro 28, foi definida com o propósito de criar um componente personalizável, onde foram utilizados tanto *generics* quanto *ports*. Os *generics* permitiram ajustar as características do componente, enquanto os *ports* serviram como interfaces.

Quadro 28 – Entidade de um Contador descrito em VHDL

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.NUMERIC_STD.ALL;

ENTITY contador IS

    GENERIC (
-- Numero de bit do contador
        n         : INTEGER := 4;
-- Periodo de contagem.
        delay     : INTEGER := 25e6);

    PORT(
        clk, rst : IN  STD_LOGIC;
        q         : OUT UNSIGNED (n-1 DOWNTO 0)
        );

END contador;
```

Fonte: Autoria própria

O comando "n", determina o número de bits do contador, e "*delay*", estabelece o período de contagem em ciclos de *clock*. Dentre as portas declaradas na entidade, "clk"é responsável por sincronizar as operações do componente, "rst"é utilizado para reinicializar o contador e "q"representa o sinal de saída que armazena o valor atual do contador. O valor de 25e6 (25Mhz) foi *inputado* para *delay* para que a contagem aconteça a um *clock* de 2HZ, tendo em vista que foi utilizado o pino de 50 Hz presente no DE2-2C35.

O sinal "habilita"presente na arquitetura (quadro 29), é utilizado para habilitar a contagem a cada ciclo de *delay*. Concorrente a este sinal foram criados dois processos, o primeiro chamado "*PROCESS* (clk)"utilizado para gerar o ciclo de atraso. Neste foi criada a variável "j"do tipo *INTEGER* para contar os ciclos de atraso. A cada borda de subida do sinal de *clock* (clk), a variável "j"foi incrementada em 1. Além disso, a sinal de habilitação (habilita) foi configurado como FALSE. Quando a variável "j"atingiu o valor de delay - 1, indicando a conclusão do ciclo de atraso, "j"foi *resetada* para 0 e a sinal de habilitação foi configurada como *TRUE*, indicando que a contagem poderia ser realizada.

Quadro 29 – Arquitetura de um Contador descrito em VHDL

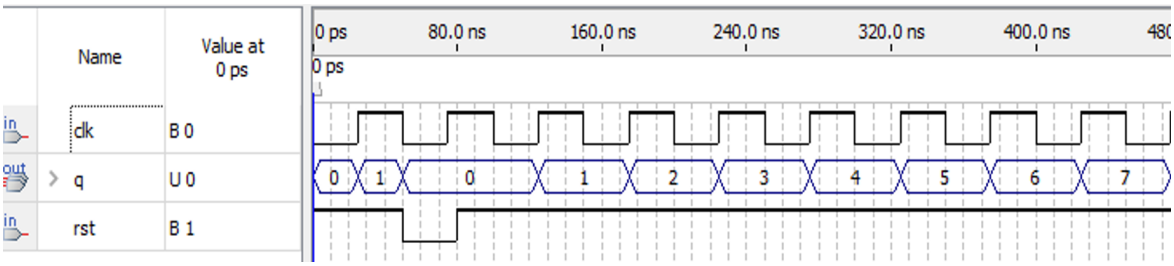
```
ARCHITECTURE Arq_contador OF contador IS
--Habilita a contagem a cada ciclo de delay
    SIGNAL  habilita : BOOLEAN;
BEGIN
--Processo para gerar p ciclo de Delay
    PROCESS(clk) IS
        VARIABLE j   : INTEGER RANGE 0 TO delay;
    BEGIN
        IF RISING_EDGE(clk) THEN
            j := j + 1;
            habilita <= FALSE;
            IF j = delay -1 THEN
                j:=0;
                habilita <= TRUE;
            END IF;
        END IF;
    END PROCESS;
    PROCESS(clk,rst) IS
        VARIABLE i : UNSIGNED(n-1 DOWNTO 0);
        BEGIN
        IF rst = '0' THEN
            i:=(OTHERS => '0');
--Contagem acontece uma vez a cada ciclo de delay
        ELSIF habilita = TRUE THEN
            IF RISING_EDGE(clk) THEN
                i := i + 1;
            END IF;
        END IF;
        q <= i;
    END PROCESS;
END Arq_contador;
```

Fonte: Autoria própria

O segundo processo chamado foi responsável pela contagem em si. Neste foi utilizado uma variável chamada "i"do tipo *UNSIGNED* para armazenar o valor do contador. Quando o sinal de *reset* (rst) estava em nível baixo ('0'), o contador era resetado para zero. Caso contrário, se a sinal de habilitação (habilita) estava em nível alto (*TRUE*) e ocorria uma borda de subida do sinal de *clock* (clk), a variável "i"era incrementada em 1. O valor atual do contador "i"era atribuído à saída "q".

Nesse sentido, procedeu-se à simulação do algoritmo, a fim de examinar o comportamento do contador em diversos cenários, garantindo, dessa forma, a validação de seu funcionamento.

Figura 53 – Simulação contador



Fonte: Autoria própria

Neste experimento, o mapeamento de pinos foi configurado utilizando o clock interno do FPGA. A tabela 30 apresenta o mapeamento de pinos utilizado. Diferentemente do circuito registrador, para o contador, foi escolhido o pino que corresponde ao clock interno de 50 MHz do DE2-2C35 como a fonte de clock.

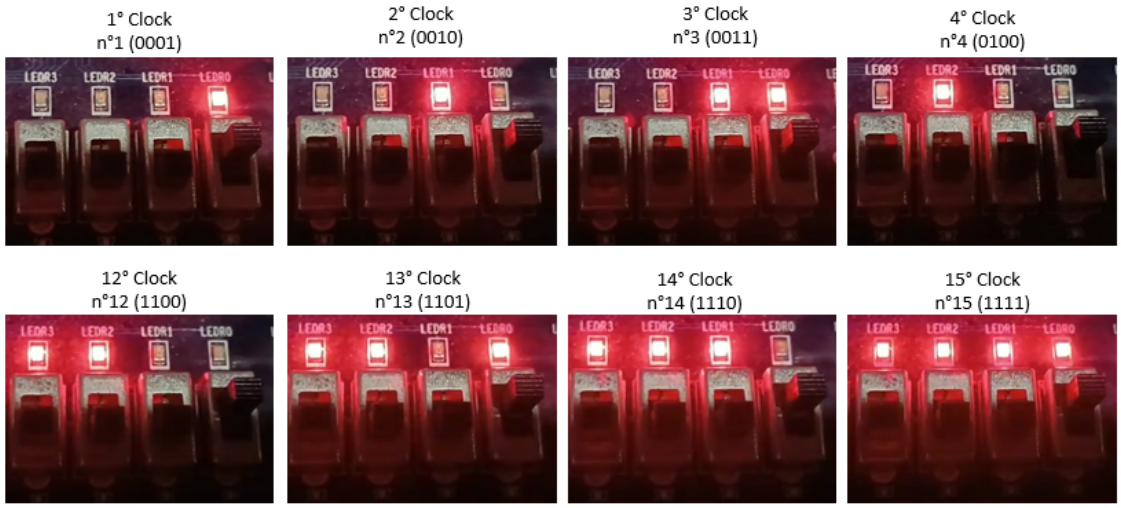
Quadro 30 – Conexão entre os sinais VHDL e os pinos do FPGA para o Contador

Variável VHDL	**Correspondente Físico**	**Pinos na Placa**
clk	CLOCK_50	PIN_N2
rst	SW[0]	PIN_25
q (3)	LEDR[3]	PIN_AC22
q (2)	LEDR[2]	PIN_AB21
q (1)	LEDR[1]	PIN_AF23
q (0)	LEDR[0]	PIN_AE23

Fonte: Autoria própria

No teste de hardware ilustrado na figura 54, são exibidos os resultados dos primeiros quatro ciclos de *clock* da contagem, que vão de 1 a 4. Além disso, também são mostrados os últimos quatro ciclos de *clock*, correspondendo à contagem de 12 a 15.

Figura 54 – Teste de *hardware* do contador



Fonte: Autoria própria

5.4 Relógio digital

O projeto proposto envolve várias partes e elementos que trabalham em conjunto para garantir o seu funcionamento adequado. Sua implementação é complexa devido à diversidade de blocos e elementos envolvidos, sendo necessário utilizar diversos componentes para alcançar as funcionalidades desejadas. O projeto envolve contadores, Divisores de frequência, decodificadores, multiplexadores, entre outros. Outra parte importante é a exibição dos dígitos do relógio. Isso pode ser feito por meio de displays de sete segmentos, que mostram os números de 0 a 9 em formato digital. Para controlar a exibição correta dos dígitos, são utilizados multiplexadores e decodificadores para selecionar e acionar os segmentos apropriados em cada instante de tempo. O algoritmo completo é apresentado no apêndice A.

5.4.1 Entidade

A entidade "relógio"possui três parâmetros dentro de *GENERIC*, que desempenham um papel importante na configuração do relógio. O parâmetro "*word*"determina o tamanho da palavra de instrução utilizada, com um valor padrão de 21 bits. O parâmetro "dados"define o tamanho dos dados manipulados pelo relógio, com um valor padrão de 8 bits. Além disso, o parâmetro "addr_width"especifica o tamanho da largura de endereço, com um valor padrão de 8 bits. Esses parâmetros permitem uma maior flexibilidade e adaptabilidade ao projeto do relógio digital, permitindo que o mesmo seja ajustado de acordo com as necessidades específicas. O algorítimo completo é apresentado no Anexo A. O quadro 31 é apresentado a estruturação da entidade.

Quadro 31 – Entidade do relógio digital

```
ENTITY relogio IS
    GENERIC(
        word       : natural := 21;
        dados      : natural := 8;
        addr_width : natural := 8);

    PORT (
        SW               : in std_logic_vector(17 downto 0);
        CLOCK_50         : in std_logic;
        KEY              : in  std_logic_vector(3 downto 0);
        HEX0, HEX1, HEX2 : OUT STD_LOGIC_VECTOR(6 downto 0)
        HEX3, HEX4, HEX5 : OUT STD_LOGIC_VECTOR(6 downto 0));
);
END ENTITY ;
```

Fonte: Autoria própria

A porta "SW"é uma entrada para receber dados externos, "CLOCK_50"é uma entrada para o sinal interno da placa, KEY"é uma entrada para comandos e as portas "HEX"são saídas para exibir os dígitos em um display. Essas portas são essenciais para o funcionamento do relógio digital e permitem a entrada de dados, sincronização do tempo, controle por comandos e exibição das informações. Os detalhes específicos da implementação do relógio, como o comportamento interno, a lógica de controle e a interação com o usuário, estão presentes na descrição da arquitetura.

5.4.2 Arquitetura

Na arquitetura, foram declarados sinais dos tipos *standard* (std), utilizados para representar diferentes elementos e conexões dentro do relógio. Por exemplo, os sinais "habilitasu", "habilitasd", "habilitamu"e "habilitamd"são responsáveis por habilitar as unidades de segundos, minutos, horas e dia do relógio, respectivamente. O sinal "basetempo"representa a base de tempo utilizada para a contagem do relógio. Outros sinais estão envolvidos na comunicação de dados e na interação com os botões do relógio. Esses sinais desempenham um papel crucial no correto funcionamento do relógio digital, garantindo que as informações sejam processadas adequadamente e exibidas de acordo com as funcionalidades desejadas. o quadro 32 apresenta a estrutura resumida para a arquitetura.

Quadro 32 – Arquitetura do relógio digital

```
ARCHITECTURE  relogio_top OF relogio IS

    SIGNAL ...

    BEGIN

    processador : entity work.processador
    --[...] Corpo do bloco processador
    rom         : entity work.rom
    --[...] Corpo do bloco  rom
    decoficador : entity work.decodificador
    --[...] Corpo do bloco  decodificador
    hex7seg     : entity work.hex7seg_top
    --[...] Corpo do bloco  hexa7 segmentos
    botoes      : entity work.botoes
    --[...] Corpo do bloco  botões
    base_tempo  : entity work.divisorGenerico
    --[...] Corpo do bloco  divisor genérico
END relogio_top;
```

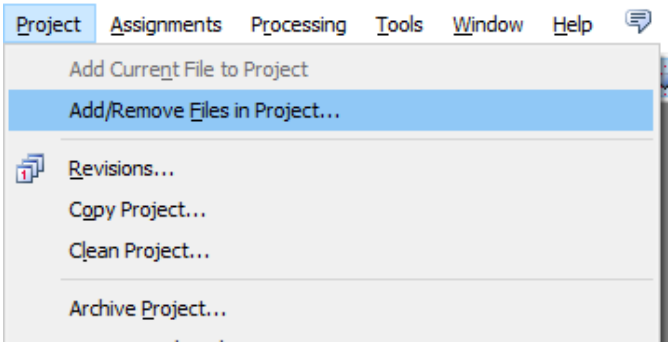
Fonte: Autoria própria

Os sinais indicados com ".work"são blocos que estão sendo referenciados no código principal (Apêndice A), a estrutura desses códigos são descritas nos anexos deste trabalho. A função destes blocos no projeto principal é descrita a seguir.

- Processador: Este bloco contém a lógica e os componentes necessários para realizar operações de processamento de dados(Apêndice B).
- ROM: Representa uma memória somente leitura (ROM). É usada para armazenar dados que não podem ser alterados durante a execução do sistema. (Apêndice C)
- Decodificador: Circuito lógico que converte um código de entrada em um conjunto correspondente de saídas (Apêndice D).
- Decodificador de sete segmentos hexadecimal: Usado para exibir números hexadecimais em displays de sete segmentos (Apêndice E).
- Botoes: Representa botões ou interruptores de entrada. São usados para receber entrada do usuário ou controlar o sistema (Apêndice F).
- Divisor de tempo genérico: Pode ser usado para dividir um sinal de relógio em frequências menores ou controlar o tempo em um sistema (Apêndice G).

Para que os blocos fossem adicionados ao *Top-Level* ou entidade principal, bastou que fossem inseridos na opção Project > Add/remove file, como observado na figura 55-a.

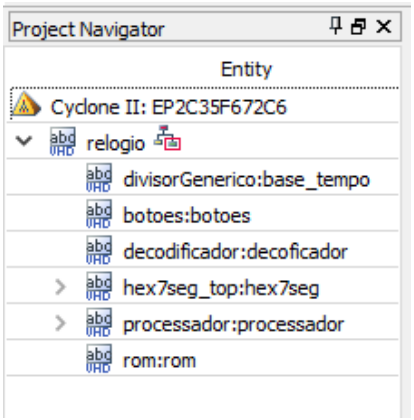
Figura 55 – Adicionar/ remover arquivo



Fonte: Autoria própria

Logo, o diretório do projeto, apresenta a seguinte característica da figura 56. Onde hierarquicamente os demais blocos estão inseridos no projeto principal. Como dito anteriormente, isso torna o desenvolvimento mais organizado e efetivo, haja vista que cada bloco pode ser testado antes da inserção no bloco principal.

Figura 56 – Hierarquia de projeto

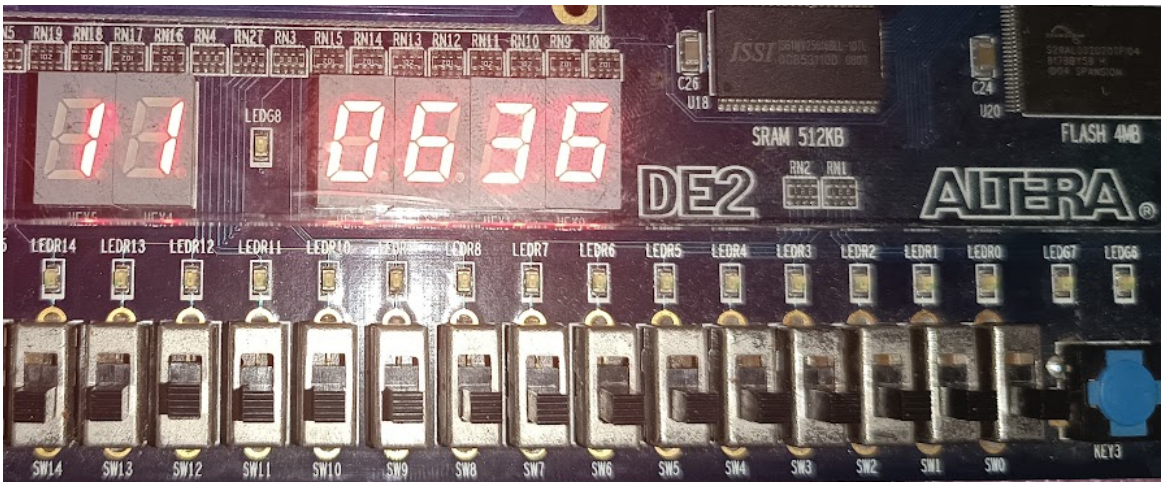


Fonte: Autoria própria

5.4.3 Mapeamento dos pinos

O mapeamento dos pinos contou com o uso dos seguintes periféricos e elementos: *Clock* de 50 Hz, seis displays de 7 segmentos (HEX), três *pushbuttons* (KEY), e as três chaves de alternância (SW). A correspondência destes pode ser verificada no anexo A.

Figura 57 – Teste de Hardware do relógio digital



Fonte: Autoria própria

Por fim, a figura 57 apresenta o relógio digital implementado na placa DE2-2C35. Entre as funções principais esta conta com o botão de Reset (KEY3), adicionar horas (KEY2),adicionar minutos (KEY1), adicionar segundos (KEY0), Acelerar os segundos (SW0) e acelerar os minutos (SW1).

6 Considerações finais

Inicialmente, a pesquisa de referência enfrentou dificuldades em encontrar material atualizado e de qualidade relacionado especificamente ao uso do kit FPGA DE2 da Altera. A disponibilidade de literatura especializada e estudos de caso relevantes pode ser limitada, o que exigiu uma busca ampla e criteriosa por fontes confiáveis. Essa etapa foi fundamental para embasar os roteiros de ensino desenvolvidos.

Desenvolver uma pesquisa em FPGA e gerar roteiros de ensino pode apresentar desafios significativos devido à complexidade e à natureza altamente técnica desse campo. A pesquisa em FPGA requer conhecimento em design digital, linguagens de descrição de hardware, ferramentas de desenvolvimento FPGA, além de uma compreensão aprofundada dos conceitos de eletrônica digital. Espera-se que o discente que for apresentado aos experimentos possua uma solida base em conceitos de eletrônica digital.

A flexibilidade dos FPGA também permite a experimentação e a prototipagem rápida de projetos, possibilitando a exploração de ideias inovadoras e o desenvolvimento de soluções customizadas. Além disso, a pesquisa em FPGA contribui para o avanço científico e tecnológico, impulsionando a criação de produtos inovadores de alta tecnologia.

Este trabalho permite inferir que a utilização de kits FPGA, como o DE2 modelo 2C35, traz benefícios significativos para o ensino de automação. Esses dispositivos reconfiguráveis permitem a implementação e simulação de circuitos combinacionais, sequenciais e a integração desses dois tipos. Dessa forma, os estudantes podem adquirir experiência prática e desenvolver habilidades necessárias para lidar com desafios reais da automação. O uso desses kits proporciona um ambiente de aprendizado imersivo, em que os alunos podem explorar conceitos teóricos e aplicá-los na prática, preparando-se melhor para as demandas do mercado de trabalho.

Em conclusão, o ensino em FPGA utilizando o kit DE2 modelo 2C35 da Altera representa uma abordagem promissora no campo da educação em engenharia. Através desse dispositivo reconfigurável, os estudantes têm a oportunidade de aprender de forma prática e hands-on, explorando conceitos de design digital, circuitos combinacionais e sequenciais. O kit DE2 modelo 2C35 oferece uma plataforma versátil e completa, permitindo a implementação de projetos complexos e a simulação de circuitos em tempo real, preparando os alunos para os desafios do mercado de trabalho.

É importante ressaltar que este trabalho representou apenas uma introdução a esses conceitos e aplicações. Existem muitas áreas adicionais para exploração e aprimoramento no campo dos sistemas embarcados e FPGAs. Alguns exemplos de trabalhos futuros incluem:

- Exploração de técnicas avançadas de design com VHDL: Pesquisar e desenvolver roteiros que abordem conceitos mais complexos de VHDL, como design hierárquico, testbench, simulação de tempo real, técnicas de otimização, entre outros.
- Implementação de sistemas embarcados: Utilizar o kit FPGA DE2-2C35 para desenvolver projetos de sistemas embarcados, envolvendo microcontroladores, periféricos e comunicação com outros dispositivos.
- Aplicação de processamento de sinal digital (DSP): Explorar o uso do kit FPGA para projetos de processamento de sinais digitais, como filtros digitais, transformada de Fourier, processamento de imagens, entre outros.
- Desenvolvimento de interfaces com o mundo real: Integração do kit FPGA com sensores, atuadores e dispositivos externos para a implementação de projetos práticos, como controle de motores, automação residencial, sistemas de segurança, entre outros.
- Estudo de técnicas de otimização de desempenho: Investigar e implementar técnicas de otimização para melhorar o desempenho e a eficiência dos projetos desenvolvidos no kit FPGA DE2-2C35.
- Expansão do escopo de ensino para outros kits e plataformas: Adaptar os roteiros desenvolvidos para outros kits e plataformas FPGA, permitindo uma abordagem mais ampla e diversificada no ensino de automação.

Referências

ALMEIDA, T. d. S.; CARAVALHO, R. L. d. Do vhdl ao vhdl-ams em 1 dia. 2020. Citado na página 31.

ALTERA. *DE2 Development and Education Board, user manual*. [S.l.], 2006. Disponível em: <http://www.altera.com>. Citado 4 vezes nas páginas 38, 39, 40 e 61.

ALTERA. *Introduction to Quartus II Software*. 2011. Disponível em: <http://www.altera.com/literature/manual/quartus2-introduction.pdf>. Citado 6 vezes nas páginas 32, 38, 39, 40, 52 e 56.

BROWN, S. D. et al. *Field-programmable gate arrays*. [S.l.]: Springer Science & Business Media, 1992. v. 180. Citado na página 24.

BROWN, S. D.; ROSE, J. *Field-Programmable Gate Arrays*. [S.l.]: Springer US, 2013. Citado 2 vezes nas páginas 20 e 24.

COSTA, C. A. *Desenvolvimento de hardware de criptografia RSA em linguagem Verilog*. 123 p. Dissertação (Dissertação (Mestrado)) — Universidade Federal de Itajubá, 2014. Disponível em: <https://repositorio.unifei.edu.br/xmlui/handle/123456789/319>. Citado na página 25.

D'AMORE, R. *VHDL Descrição e Síntese de Circuitos Digitais*. 2. ed. [S.l.]: LTC, 2012. Citado 7 vezes nas páginas 17, 18, 29, 30, 31, 33 e 34.

DOWD, K.; SEVERANCE, C. High performance computing. Rice University, 2010. Citado na página 17.

EDWARDS, S. A. Writing vhdl for rtl synthesis. *Columbia University*, v. 1214, 2011. Citado 2 vezes nas páginas 28 e 32.

FLOYD, T. *Sistemas digitais: fundamentos e aplicações*. [S.l.]: Bookman Editora, 2009. Citado 8 vezes nas páginas 20, 21, 22, 23, 24, 54, 57 e 58.

HAGER, R.; WELTIN, H. *ASIC Design Methodologies and Tools*. [S.l.]: Springer, 2017. Citado na página 17.

HARRIS, S. L.; HARRIS, D. *Digital design and computer architecture*. [S.l.]: Morgan Kaufmann, 2015. Citado na página 27.

JUNIOR, E. V. et al. Sistema baseado em lógica fuzzy em fpga para processamento e análise de sinais eletrocardiográficos patológicos fuzzy logic based system in fpga for processing and analysis of pathological electrocardiographic signals. *Brazilian Journal of Development*, v. 8, n. 7, p. 51294–51302, 2022. Citado na página 20.

MATURANA, P. S. Algoritmos de detecção de bordas implementados em fpga. Universidade Estadual Paulista (Unesp), 2010. Citado na página 40.

MAXFIELD, C. *The Design Warrior's Guide to FPGAs: Devices, Tools and Flows*. [S.l.]: Elsevier, 2004. Citado na página 24.

MISHRA, A.; SARMA, N. Fpga based embedded systems design: A review. *Microprocessors and Microsystems*, Elsevier, v. 59, p. 53–73, 2018. Citado na página 20.

MOORE, A.; WILSON, R. *FPGAs for dummies*. 2. ed. New Jersey: John Wiley & Sons, Inc, 2017. 46 p. Disponível em: <https://www.altera.com/content/dam/altera-www/global/en_US/pdfs/literature/misc/fpgas_for_dummies_ebook.pdf>. Citado na página 20.

ORDONEZ, E. D. M. et al. *Projeto, desempenho e aplicações de sistemas digitais em circuitos programáveis (FPGAs)*. Pompéia: Bless, 2003. 300 p. Citado 14 vezes nas páginas 21, 25, 26, 27, 28, 29, 30, 31, 32, 35, 36, 43, 55 e 56.

PEDRONI, V. A. Eletrônica digital moderna e vhdl. *Rio de Janeiro, RJ: Elsevier*, 2010. Citado 2 vezes nas páginas 20 e 21.

PROAKIS, J. G.; MANOLAKIS, D. G. *Digital Signal Processing: Principles, Algorithms, and Applications*. [S.l.]: Prentice Hall, 2006. Citado na página 20.

ROTH, H. *Circuit design with VHDL*. [S.l.]: Cambridge, MA: MIT Press, 2004. Citado 5 vezes nas páginas 22, 28, 34, 52 e 69.

SILVA, A. M.; ASSUNÇÃO, W. K. *Introdução à Lógica Digital*. [S.l.]: Elsevier, 2013. Citado na página 23.

SILVA, G. A. P. d. *Implementação de redes neurais artificiais utilizando VHDL para aplicações em reconhecimento e classificação de padrões*. Dissertação (B.S. thesis) — Universidade Tecnológica Federal do Paraná, 2019. Citado na página 31.

SMITH, R.; FRANZON, P. D. Field-programmable gate array (fpga) technology. In: LIU, W. (Ed.). *Comprehensive Microsystems*. [S.l.]: Elsevier, 2008. p. 45–74. Citado na página 22.

SOUSA, J. P. A. C. d. Projeto e síntese em alto nível de circuitos digitais. 2017. Citado na página 27.

TOCCI, R. J.; WIDMER, N. S.; MOSS, G. L. *Sistemas digitais*. [S.l.]: Pearson Educación, 2010. Citado 2 vezes nas páginas 55 e 56.

WONG, K. C.; BRICAUD, P. *Introduction to Programmable Logic Controllers*. 3. ed. [S.l.]: Wiley, 2007. Citado na página 24.

Xilinx Inc. *UltraScale Architecture and Product Data Sheet: Overview*. 2021. Retrieved from <https://www.xilinx.com/support/documentation/data_sheets/ds890-ultrascale-overview.pdf>. Citado na página 24.

APÊNDICE A – Descrição VHDL para o relógio digital do Roteiro 4

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;

ENTITY relogio IS
    GENERIC(
        word : natural := 21;
        dados : natural := 8;
        addr_width : natural := 8 );

    PORT (
        SW : in std_logic_vector(17 downto 0);
        CLOCK_50 : in std_logic;
        KEY : in  std_logic_vector(3 downto 0);
        HEX0, HEX1, HEX2, HEX3, HEX4, HEX5 : OUT STD_LOGIC_VECTOR(6 downto 0)
);
END ENTITY ;

ARCHITECTURE  relogio_top OF relogio IS

    SIGNAL habilitasu, habilitasd, habilitamu,
    habilitamd, habilitahu, habilitahd,
    habilitabot, basetempo, datain_unprocessed_basetempo: std_logic;
    SIGNAL rw : std_logic_vector(1 downto 0);
    SIGNAL rom_data: std_logic_vector(word-1 downto 0);
    SIGNAL addr,rom_addr: std_logic_vector(addr_width-1 downto 0);
    SIGNAL datain_unprocessed_botoes: std_logic_vector(3 downto 0);
    SIGNAL datain, dataout : std_logic_vector(dados-1 downto 0);

    BEGIN

    processador: entity work.processador
    port map(
        datain => ("000" & (datain_unprocessed_basetempo &
        datain_unprocessed_botoes)),
        rom_in => rom_data,
        clk => CLOCK_50,
```

```
    rom_addr => rom_addr,
    rw => rw,
    dataout => dataout,
    addr => addr
    );

rom: entity work.rom
port map(
    q => rom_data,
    addr => rom_addr
);

decoficador: entity work.decodificador
port map(
    entrada => addr,
    rw => rw,
    habilitasu => habilitasu,
    habilitasd => habilitasd,
    habilitamu => habilitamu,
    habilitamd => habilitamd,
    habilitahu => habilitahu,
    habilitahd => habilitahd,
    habilitabot => habilitabot,
    basetempo => basetempo
    );

hex7seg: entity work.hex7seg_top
port map(
    clk =>  CLOCK_50,
    dado_reg0 => dataout(3 downto 0),
    dado_reg1 => dataout(3 downto 0),
    dado_reg2 => dataout(3 downto 0),
    dado_reg3 => dataout(3 downto 0),
    dado_reg4 => dataout(3 downto 0),
    dado_reg5 => dataout(3 downto 0),
    habilita0 => habilitasu,
    habilita1 => habilitasd,
    habilita2 => habilitamu,
    habilita3 => habilitamd,
```

```
        habilita4 => habilitahu,
        habilita5 => habilitahd,
        HEX0 => HEX0,
        HEX1 => HEX1,
        HEX2 => HEX2,
        HEX3 => HEX3,
        HEX4 => HEX4,
        HEX5 => HEX5
        );

    botoes: entity work.botoes
    port map(
        clk => CLOCK_50,
        KEY => KEY,
        habilita_botoes => habilitabot,
        saida => datain_unprocessed_botoes
        );

    base_tempo: entity work.divisorGenerico
    port map(
        sw_in_0 => SW(0),
        sw_in_1 => SW(1),
        reset => basetempo,
        clk =>  CLOCK_50,
        saida_clk => datain_unprocessed_basetempo
        );

END relogio_top;
```

APÊNDICE B – Processador

```
library ieee;
use ieee.std_logic_1164.all;

entity processador is
   generic (
ROM_DATA_WIDTH : natural := 21;
ADDR_WIDTH : natural := 8;
DATA_WIDTH : natural := 8;
ROM_ADDR_W : natural := 8 );

-- Processador
port(
datain :      in std_logic_vector(DATA_WIDTH-1 downto 0);
rom_in :      in std_logic_vector(ROM_DATA_WIDTH-1 downto 0);
clk :         in std_logic;
rom_addr :    out std_logic_vector(ROM_ADDR_W-1 downto 0);
rw :          out std_logic_vector(1 downto 0);
dataout :     out std_logic_vector(DATA_WIDTH-1 downto 0);
addr :        out std_logic_vector(ADDR_WIDTH-1 downto 0));

end entity;

--  PALAVRA DE INSTUCAO 21 BITS
--OPCODE|REGS1|REGS2|ENDW|IMEDIATO
-- XXXX | XXX | XXX | XXX|XXXXXXXX

architecture description of processador is

signal flag_uc, select_mux1, we_uc, muxDOutSel  :std_logic;
signal ula_instr ,select_mux2 : std_logic_vector(1 downto 0);
signal mux1_out,mux2_out,addr_out: std_logic_vector(DATA_WIDTH-1 downto 0);
signal s1,s2,ula_out, muxDOut_out: std_logic_vector(DATA_WIDTH-1 downto 0);
    signal pc_out : std_logic_vector(ROM_ADDR_W-1 downto 0);

begin
```

```
uc: entity work.uc
port map(
  op => rom_in(ROM_DATA_WIDTH-1 downto 17),--instrucao opcode
        mux1 => select_mux1,--instrucao para a ULA
  muxDataOut=> muxDOutSel,
  opr=>ula_instr,
  mux2 =>select_mux2,
        we =>we_uc,   --Write enable no banco de registradores
  rw =>rw,--Write ou read
  flag => flag_uc
);

ula: entity work.ula
port map(
clk=>clk,
IN_mux=>mux2_out,
IN_banco=>s2,
OP=>ula_instr,
S=>ula_out,
FLAG=>flag_uc
);

mux2: entity work.mux2
port map(
A=>addr_out,
B=>rom_in(7 downto 0),
Sel=>select_mux1,
Y=>mux1_out
);


somadorGenerico: entity work.somadorGenerico
port map(
entradaA=>pc_out,
saida=>addr_out
);

mux3: entity work.mux3
port map(
```

```
  a1=>s1,
  a0=>datain,
  a2=>rom_in(7 downto 0),
  sel=>select_mux2,
  b=>mux2_out
);

muxDataOut: entity work.mux2
port map(
A=>ula_out,
B=>s2,
Sel=>muxDOutSel,
Y=>dataout
);
bancoRegistradores: entity work.bancoRegistradores
port map(
clk=>clk,
enderecoA=>rom_in(16 downto 14),
enderecoB=>rom_in(13 downto 11),
enderecoC=>rom_in(10 downto 8),
dadoEscritaC=>ula_out,
escreveC=>we_uc,
saidaA=>s1,
saidaB=>s2
  );

pc: entity work.pc
port map(
pc_i=>mux1_out,
clk=>clk,
pc_o=>pc_out
);


rom_addr <= pc_out;
--dataout <= muxDOut_out;
addr <= rom_in(7 downto 0);

end description;
```

APÊNDICE C – ROM

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity rom is
generic (
DATA_WIDTH : natural := 21;
ADDR_WIDTH : natural := 8);

port (
addr : in std_logic_vector (ADDR_WIDTH - 1 downto 0);
q : out std_logic_vector((DATA_WIDTH -1) downto 0));
end entity;

architecture rtl of rom is
-- Build a 2-D array type for the ROM
subtype word_t is std_logic_vector((DATA_WIDTH-1) downto 0);
type memory_t is array(2**ADDR_WIDTH-1 downto 0) of word_t;

function init_rom
return memory_t is
variable tmp : memory_t := (others => (others => '0'));

begin
        -- Inicializa os endereços:
tmp(0) := b"000000000010100000000"; tmp(1) := b"000000000010000000000";
tmp(2) := b"000000000001100000000"; tmp(3) := b"000000000001000000000";
tmp(4) := b"000000000000100000000"; tmp(5) := b"000000000000000000000";
tmp(6) := b"110000010110100000101"; tmp(7) := b"110000010010000000100";
tmp(8) := b"110000001101100000011"; tmp(9) := b"110000001001000000010";
tmp(10) := b"110000000100100000001"; tmp(11) := b"110000000000000000000";
tmp(12) := b"010000000000001001110"; tmp(13) := b"000100000000000000001";
tmp(14) := b"110000000000000000000"; tmp(15) := b"001000000000000001010";
tmp(16) := b"001100000000000110101"; tmp(17) := b"010100000011000000110";
tmp(18) := b"001000011011000000001"; tmp(19) := b"001100000000000010101";
tmp(20) := b"010000000000000011001"; tmp(21) := b"000100000000000000001";
```

```
tmp(22) := b"11000000000000000000"; tmp(23) := b"00100000000000001010";
tmp(24) := b"00110000000000110101"; tmp(25) := b"01010000011000000110";
tmp(26) := b"00100001101100000010"; tmp(27) := b"00110000000000111100";
tmp(28) := b"01010000011000000110"; tmp(29) := b"00100001101100000100";
tmp(30) := b"00110000000001000110"; tmp(31) := b"01010000011000000110";
tmp(32) := b"00100001101100001000"; tmp(33) := b"00110000000000000000";
tmp(34) := b"00100000000000001010"; tmp(35) := b"00110000000000110101";
tmp(36) := b"00100000100100000110"; tmp(37) := b"00110000000000111010";
tmp(38) := b"00100000100100001010"; tmp(39) := b"00110000000000111111";
tmp(40) := b"00100000110110000110"; tmp(41) := b"00110000000001000100";
tmp(42) := b"00100001001000001010"; tmp(43) := b"00110000000001001001";
tmp(44) := b"01100001001100000100"; tmp(45) := b"01100001011110000010";
tmp(46) := b"00100001101100000000"; tmp(47) := b"00110000000001001110";
tmp(48) := b"00100001111110000000"; tmp(49) := b"00110000000001001110";
tmp(50) := b"11101101110000000000"; tmp(51) := b"00110000000000000000";
tmp(52) := b"01000000000001001110"; tmp(53) := b"00000000000000000000";
tmp(54) := b"11000000000000000000"; tmp(55) := b"00010000100100000001";
tmp(56) := b"11000000100100000001"; tmp(57) := b"01000000000000100100";
tmp(58) := b"00000000000100000000"; tmp(59) := b"11000000100100000001";
tmp(60) := b"00010000100100000001"; tmp(61) := b"11000000100100000010";
tmp(62) := b"01000000000000100110"; tmp(63) := b"00000000001000000000";
tmp(64) := b"11000000100100000010"; tmp(65) := b"00010000110110000001";
tmp(66) := b"11000000110110000011"; tmp(67) := b"01000000000000101000";
tmp(68) := b"00000000001100000000"; tmp(69) := b"11000000110110000011";
tmp(70) := b"00010001001000000001"; tmp(71) := b"11000001001000000100";
tmp(72) := b"01000000000000101010"; tmp(73) := b"00000000010000000000";
tmp(74) := b"11000001001000000100"; tmp(75) := b"00010001011010000001";
tmp(76) := b"11000001011010000101"; tmp(77) := b"01000000000000101100";
tmp(78) := b"01010000011000010000"; tmp(79) := b"00100001101100010000";
tmp(80) := b"00110000000000001101"; tmp(81) := b"01000000000001001110";
        return tmp;
end init_rom;

-- Declare the ROM signal and specify a default value. Quartus Prime
-- will create a memory initialization file (.mif) based on the
signal rom : memory_t := init_rom;
begin
q <= rom(to_integer(unsigned(addr)));
end rtl;
```

APÊNDICE D – Decodificador

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.numeric_std.ALL;

ENTITY decodificador IS
GENERIC (
largura : NATURAL  := 8);
PORT(
   entrada     : IN STD_LOGIC_VECTOR((largura-1) DOWNTO  0);
rw          : IN STD_LOGIC_VECTOR(1 DOWNTO 0);
habilitasu, habilitasd : OUT std_logic;
habilitamu, habilitamd : OUT std_logic;
habilitahu, habilitahd : OUT std_logic;
habilitabot, basetempo: OUT std_logic );
END ENTITY ;

ARCHITECTURE su OF decodificador IS
    BEGIN
--Escrever no display unidade de segundos
  habilitasu <= '1' WHEN entrada = "00000000" and rw = "01" ELSE '0';
--Escrever no display dezena de segundos
  habilitasd <= '1' WHEN entrada = "00000001" and rw = "01" ELSE '0';
--Escrever no display unidade de minutos
  habilitamu <= '1' WHEN entrada = "00000010" and rw = "01" ELSE '0';
--Escrever no display dezena de minutos
  habilitamd <= '1' WHEN entrada = "00000011" and rw = "01" ELSE '0';
--Escrever no display unidade de horas
  habilitahu <= '1' WHEN entrada = "00000100" and rw = "01" ELSE '0';
--Escrever no display dezena de horas
  habilitahd <= '1' WHEN entrada = "00000101" and rw = "01" ELSE '0';
--Habilitar leitura dos botoes
  habilitabot <= '1'WHEN entrada = "00000110"and rw = "10"  ELSE '0';
--Habilitar leitura da base de tempo
  basetempo <= '1' when entrada = "00010000" and rw = "10"  ELSE '0';
END ARCHITECTURE ;
```

APÊNDICE E – Conversor Hexadecimal para 7 segmentos

```
library IEEE;
use ieee.std_logic_1164.all;

entity conversorHex7Seg is
    port
    (
        -- Input ports
        dadoHex : in  std_logic_vector(3 downto 0);
        apaga   : in  std_logic := '0';
        negativo : in  std_logic := '0';
        overFlow : in  std_logic := '0';
        -- Output ports
        saida7seg : out std_logic_vector(6 downto 0)  -- := (others => '1')
    );
end entity;

architecture comportamento of conversorHex7Seg is
   --
   --       0
   --      ---
   --     |   |
   --    5|   |1
   --     | 6 |
   --      ---
   --     |   |
   --    4|   |2
   --     |   |
   --      ---
   --       3
   --

  signal rascSaida7seg: std_logic_vector(6 downto 0);

begin
    rascSaida7seg <=    "1000000" when dadoHex="0000" else ---0
                            "1111001" when dadoHex="0001" else ---1
```

```
                        "0100100" when dadoHex="0010" else ---2
                        "0110000" when dadoHex="0011" else ---3
                        "0011001" when dadoHex="0100" else ---4
                        "0010010" when dadoHex="0101" else ---5
                        "0000010" when dadoHex="0110" else ---6
                        "1111000" when dadoHex="0111" else ---7
                        "0000000" when dadoHex="1000" else ---8
                        "0010000" when dadoHex="1001" else ---9
                        "0001000" when dadoHex="1010" else ---A
                        "0000011" when dadoHex="1011" else ---B
                        "1000110" when dadoHex="1100" else ---C
                        "0100001" when dadoHex="1101" else ---D
                        "0000110" when dadoHex="1110" else ---E
                        "0001110" when dadoHex="1111" else ---F
                        "1111111"; -- Apaga todos segmentos.

    saida7seg <=     "1100010" when (overFlow='1') else
                        "1111111" when (apaga='1' and negativo='0') else
                        "0111111" when (apaga='0' and negativo='1') else
                        rascSaida7seg;
end architecture;
```

APÊNDICE F – Botoes

```
library IEEE;
use ieee.std_logic_1164.all;

entity botoes is
    port
    (
        -- Input ports
  clk: std_logic;
        KEY : in  std_logic_vector(3 downto 0);
  habilita_botoes : std_logic;
        -- Output ports
  saida : out std_logic_vector(3 downto 0)
    );
end entity;

architecture comportamento of botoes is
signal saidaQ : std_logic_vector(3 downto 0);

begin
process(clk)
begin
if rising_edge(clk) then
saidaQ <= KEY;
end if;
end process;

--MANDA STATUS DO BOTAO QUANDO ESTIVER HABILITADO APENAS
saida <= not saidaQ when (habilita_botoes = '1') else "0000";
end architecture;
```

APÊNDICE G – Divisor Genérico

```
LIBRARY ieee;
   USE ieee.std_logic_1164.ALL;
   use ieee.numeric_std.all;

   entity divisorGenerico is
    generic
    (v_normal : natural := 50000000);
       port(
  sw_in_0 : in std_logic;
  sw_in_1 : in std_logic;
  reset : in std_logic;
           clk : in std_logic;
           saida_clk : out std_logic
           );
   end entity;

architecture divInteiro of divisorGenerico is
        signal tick : std_logic := '0';
        signal contador : integer range 0 to v_normal + 1 := 0;
  signal divisor : natural := 50000000;
  signal concat : std_logic_vector(1 downto 0);
   begin
concat <= sw_in_0 & sw_in_1;
with concat select
divisor <= v_normal/100 when "10", --SELECAO DE VELOCIDADE
  v_normal/500 when "01",
           v_normal/5000 when "11",
  v_normal when others;

        process(clk)
        begin
if rising_edge(clk) then
if tick = '1' then
if reset = '1' then
contador <= 0;
tick <= '0';
```

```
end if;

elsif contador >= divisor then
tick <= '1';

else
contador <= contador + 1;
end if;
end if;
        end process;
    saida_clk <= tick;
    end architecture divInteiro;
```

APÊNDICE H – Contador de Programa (PC)

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;

ENTITY PC IS
GENERIC(
        addr : natural := 8);
PORT(
-- Input ports
pc_i : IN  std_logic_vector(addr-1 DOWNTO 0);
clk  : IN  std_logic;
-- Output ports

pc_o : OUT std_logic_vector(addr-1 DOWNTO 0) );
END ENTITY PC;

-- Update the register output on the clock's rising edge
ARCHITECTURE pc1 OF PC ISis
BEGIN
PROCESS(clk) IS
BEGIN
IF (rising_edge(clk)) THEN
pc_o <= pc_i;
END IF ;
END PROCESS;
END ARCHITECTURE;
```

APÊNDICE I – Multiplexador 3:1

```
library IEEE;
use IEEE.std_logic_1164.ALL;

entity mux3 is
port(
  a0      : in  std_logic_vector(7 downto 0);
  a1      : in  std_logic_vector(7 downto 0);
  a2      : in  std_logic_vector(7 downto 0);
  sel     : in  std_logic_vector(1 downto 0);
  b       : out std_logic_vector(7 downto 0));
end mux3;

architecture rtl of mux3 is
  -- declarative part: empty
begin
  with sel select
    b <= a0 when "00",
       a1 when "01",
       a2 when others;
end rtl;
```

APÊNDICE J – Unidade Logica Aritmética (ULA)

```
library IEEE;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
entity ULA is
generic  (
  addr : natural := 8;
  op_code : natural:= 2);
port(
clk: in std_logic;
IN_mux, IN_banco: in std_logic_vector (addr-1 downto 0);
OP: in std_logic_vector  (op_code-1 downto 0);

S: out std_logic_vector (addr-1 downto 0);
FLAG: out std_logic
);
end entity ULA;

architecture ula1 of ULA is

signal resultadoMov, resultadoAdd: std_logic_vector(addr-1 downto 0);
    signal resultadoCOmp : std_logic_vector(addr-1 downto 0);
    signal resultadoFlag : boolean;

begin
resultadoMov <= std_logic_vector(IN_mux);
resultadoAdd <= std_logic_vector(unsigned(IN_mux) + unsigned(IN_banco));
resultadoComp <= "11111111" when IN_mux = IN_banco else "00000000";

with OP select
S <= resultadoMov when "00", --MOV
resultadoAdd when "01", --ADD
resultadoComp when "11", --COMP
resultadoComp when "10", --COMPI e COMPR
"00000000" when others;

resultadoFlag <= (IN_mux = IN_banco)
```

```
and (OP = "10" --SETA A FLAG DE COMPARACAO
or OP = "11");

process (clk) is
begin
if (rising_edge(clk)) then
if(resultadoFlag) then
FLAG <= '1';
else
FLAG <='0';
end if;
end if;
end process;

end architecture;
```

APÊNDICE K – Somador Genérico

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;           --Soma (esta biblioteca =ieee)

entity somadorGenerico is
    generic(
        larguraDados : natural := 8  );
    port  (
        entradaA: in STD_LOGIC_VECTOR((larguraDados-1) downto 0);
        saida:  out STD_LOGIC_VECTOR((larguraDados-1) downto 0)   );
end entity;

architecture comportamento of somadorGenerico is
    begin
        saida <= STD_LOGIC_VECTOR(unsigned(entradaA) + "1");
end architecture;
```

APÊNDICE L – Multiplexador 2:1

```
library ieee;
use ieee.std_logic_1164.all;

entity mux2 is
generic (DATA_WIDTH : natural := 8);
port(
A : in std_logic_vector(DATA_WIDTH-1 downto 0);
B : in  std_logic_vector(DATA_WIDTH-1 downto 0);
Sel : in std_logic;

Y : out std_logi  c_vector(DATA_WIDTH-1 downto 0)
);
end entity;

architecture muxSelect of mux2 is
begin
Y <= A when Sel = '0' else B;
end architecture;
```

APÊNDICE M – Banco Registrador

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity bancoRegistradores is
    generic  (
        larguraDados        : natural := 8;
        larguraEndBancoRegs : natural := 3 );
-- Leitura de 2 registradores e escrita em 1 registrador simultaneamente.
    port (
        clk: in std_logic;

--Entrada A escolhe qual registrador vai sair para a saida A
        enderecoA: in std_logic_vector((larguraEndBancoRegs-1) downto 0);

--Entrada B Entrada B escolhe qual registrador vai sair para a saida B
        enderecoB: in std_logic_vector((larguraEndBancoRegs-1) downto 0);

--Entrada do end escrita C
        enderecoC: in std_logic_vector((larguraEndBancoRegs-1) downto 0);

--Entrada do dado de escrita C
        dadoEscritaC: in std_logic_vector((larguraDados-1) downto 0);

--Enable
        escreveC : in std_logic := '1';
        saidaA   : out std_logic_vector((larguraDados -1) downto 0);
        saidaB   : out std_logic_vector((larguraDados -1) downto 0)
    );
end entity;

architecture comportamento of bancoRegistradores is

    subtype palavra_t is std_logic_vector((larguraDados-1) downto 0);
    type memoria_t is array(2**larguraEndBancoRegs-1 downto 0) of palavra_t;
```

```
    -- Declaracao dos registradores:
    shared variable registrador : memoria_t;
begin
    process(clk) is
    begin
        if (rising_edge(clk)) then
            if (escreveC = '1') then

--Escreve o dado da entrada C no registrado
                registrador(to_integer(unsigned(enderecoC))) := dadoEscritaC; r
            end if;
        end if;
    end process;
 --Escolhe qual a saida desejada
saidaA <= registrador(to_integer(unsigned(enderecoA)));
saidaB <= registrador(to_integer(unsigned(enderecoB)));

end architecture;
```

APÊNDICE N – Unidade de Controle

```
library ieee;
use ieee.std_logic_1164.all;

entity uc is
    port(
        flag             :in std_logic;
        op               :in std_logic_vector(3 downto 0);
        mux1, muxDataOut :out std_logic;
--instrucao para a ULA -- seletor mux2
        opr,mux2,rw      :out std_logic_vector(1 downto 0);
--enable no banco de registradores
        we               :out std_logic);
end entity;

architecture description of uc is
signal resultadoMux1 : boolean;

constant instr_add : std_logic_vector(3 downto 0) := "0001";

begin
resultadoMux1 <= (flag = '1') and (op = "0011"); --JE QUANDO COMP RETORNA FLAG=1

mux1 <= '1' when resultadoMux1 or op = "0100" else '0';

muxDataOut <= '1' when op = "1100" else '0';

with op select
mux2 <= "10" when "0000", -- MOV
"10" when instr_add, --ADD
"10" when "0010", --COMP
"10" when "0110", --COMPI
"00" when "0101", --READ
"00" when "0100",  --JMP
"00" when "0011", --JE
"01" when others; --COMPR e WRITE
```

```
with op select
we <= '1' when "0101", --READ
'1' when "0110", --COMPI
'1' when "0001",  --ADD
'1' when "0000",  -- MOV
'0' when others;

with op select
rw <= "10" when "0101", --READ
"01" when "1100", --WRITE
"00" when others;

with op select
opr <= "01" when "0001", --ADD,
"10" when "0010", --COMP
"10" when "1110", --COMPR
"10" when "0110", --COMPI
"00" when others;


end description;
```

ANEXO A – Mapeamento de Pinos

O Anexo a seguir apresenta a correspondência entre os sinais e os pinos físicos do FPGA.

Altera DE2 Board Pin Table

Signal Name	FPGA Pin No.	Description
SW[0]	PIN_N25	Toggle Switch[0]
SW[1]	PIN_N26	Toggle Switch[1]
SW[2]	PIN_P25	Toggle Switch[2]
SW[3]	PIN_AE14	Toggle Switch[3]
SW[4]	PIN_AF14	Toggle Switch[4]
SW[5]	PIN_AD13	Toggle Switch[5]
SW[6]	PIN_AC13	Toggle Switch[6]
SW[7]	PIN_C13	Toggle Switch[7]
SW[8]	PIN_B13	Toggle Switch[8]
SW[9]	PIN_A13	Toggle Switch[9]
SW[10]	PIN_N1	Toggle Switch[10]
SW[11]	PIN_P1	Toggle Switch[11]
SW[12]	PIN_P2	Toggle Switch[12]
SW[13]	PIN_T7	Toggle Switch[13]
SW[14]	PIN_U3	Toggle Switch[14]
SW[15]	PIN_U4	Toggle Switch[15]
SW[16]	PIN_V1	Toggle Switch[16]
SW[17]	PIN_V2	Toggle Switch[17]
Signal Name	**FPGA Pin No.**	**Description**
DRAM_ADDR[0]	PIN_T6	SDRAM Address[0]
DRAM_ADDR[1]	PIN_V4	SDRAM Address[1]
DRAM_ADDR[2]	PIN_V3	SDRAM Address[2]
DRAM_ADDR[3]	PIN_W2	SDRAM Address[3]
DRAM_ADDR[4]	PIN_W1	SDRAM Address[4]
DRAM_ADDR[5]	PIN_U6	SDRAM Address[5]
DRAM_ADDR[6]	PIN_U7	SDRAM Address[6]
DRAM_ADDR[7]	PIN_U5	SDRAM Address[7]
DRAM_ADDR[8]	PIN_W4	SDRAM Address[8]
DRAM_ADDR[9]	PIN_W3	SDRAM Address[9]
DRAM_ADDR[10]	PIN_Y1	SDRAM Address[10]
DRAM_ADDR[11]	PIN_V5	SDRAM Address[11]
DRAM_DQ[0]	PIN_V6	SDRAM Data[0]
DRAM_DQ[1]	PIN_AA2	SDRAM Data[1]
DRAM_DQ[2]	PIN_AA1	SDRAM Data[2]
DRAM_DQ[3]	PIN_Y3	SDRAM Data[3]
DRAM_DQ[4]	PIN_Y4	SDRAM Data[4]
DRAM_DQ[5]	PIN_R8	SDRAM Data[5]
DRAM_DQ[6]	PIN_T8	SDRAM Data[6]
DRAM_DQ[7]	PIN_V7	SDRAM Data[7]
DRAM_DQ[8]	PIN_W6	SDRAM Data[8]
DRAM_DQ[9]	PIN_AB2	SDRAM Data[9]
DRAM_DQ[10]	PIN_AB1	SDRAM Data[10]
DRAM_DQ[11]	PIN_AA4	SDRAM Data[11]
DRAM_DQ[12]	PIN_AA3	SDRAM Data[12]
DRAM_DQ[13]	PIN_AC2	SDRAM Data[13]
DRAM_DQ[14]	PIN_AC1	SDRAM Data[14]
DRAM_DQ[15]	PIN_AA5	SDRAM Data[15]
DRAM_BA_0	PIN_AE2	SDRAM Bank Address[0]
DRAM_BA_1	PIN_AE3	SDRAM Bank Address[1]
DRAM_LDQM	PIN_AD2	SDRAM Low-byte Data Mask
DRAM_UDQM	PIN_Y5	SDRAM High-byte Data Mask
DRAM_RAS_N	PIN_AB4	SDRAM Row Address Strobe
DRAM_CAS_N	PIN_AB3	SDRAM Column Address Strobe
DRAM_CKE	PIN_AA6	SDRAM Clock Enable
DRAM_CLK	PIN_AA7	SDRAM Clock
DRAM_WE_N	PIN_AD3	SDRAM Write Enable
DRAM_CS_N	PIN_AC3	SDRAM Chip Select
Signal Name	**FPGA Pin No.**	**Description**
FL_ADDR[0]	PIN_AC18	FLASH Address[0]
FL_ADDR[1]	PIN_AB18	FLASH Address[1]
FL_ADDR[2]	PIN_AE19	FLASH Address[2]
FL_ADDR[3]	PIN_AF19	FLASH Address[3]
FL_ADDR[4]	PIN_AE18	FLASH Address[4]

FL_ADDR[5]	PIN_AF18	FLASH Address[5]
FL_ADDR[6]	PIN_Y16	FLASH Address[6]
FL_ADDR[7]	PIN_AA16	FLASH Address[7]
FL_ADDR[8]	PIN_AD17	FLASH Address[8]
FL_ADDR[9]	PIN_AC17	FLASH Address[9]
FL_ADDR[10]	PIN_AE17	FLASH Address[10]
FL_ADDR[11]	PIN_AF17	FLASH Address[11]
FL_ADDR[12]	PIN_W16	FLASH Address[12]
FL_ADDR[13]	PIN_W15	FLASH Address[13]
FL_ADDR[14]	PIN_AC16	FLASH Address[14]
FL_ADDR[15]	PIN_AD16	FLASH Address[15]
FL_ADDR[16]	PIN_AE16	FLASH Address[16]
FL_ADDR[17]	PIN_AC15	FLASH Address[17]
FL_ADDR[18]	PIN_AB15	FLASH Address[18]
FL_ADDR[19]	PIN_AA15	FLASH Address[19]
FL_ADDR[20]	PIN_Y15	FLASH Address[20]
FL_ADDR[21]	PIN_Y14	FLASH Address[21]
FL_DQ[0]	PIN_AD19	FLASH Data[0]
FL_DQ[1]	PIN_AC19	FLASH Data[1]
FL_DQ[2]	PIN_AF20	FLASH Data[2]
FL_DQ[3]	PIN_AE20	FLASH Data[3]
FL_DQ[4]	PIN_AB20	FLASH Data[4]
FL_DQ[5]	PIN_AC20	FLASH Data[5]
FL_DQ[6]	PIN_AF21	FLASH Data[6]
FL_DQ[7]	PIN_AE21	FLASH Data[7]
FL_CE_N	PIN_V17	FLASH Chip Enable
FL_OE_N	PIN_W17	FLASH Output Enable
FL_RST_N	PIN_AA18	FLASH Reset
FL_WE_N	PIN_AA17	FLASH Write Enable
Signal Name	**FPGA Pin No.**	**Description**
SRAM_ADDR[0]	PIN_AE4	SRAM Address[0]
SRAM_ADDR[1]	PIN_AF4	SRAM Address[1]
SRAM_ADDR[2]	PIN_AC5	SRAM Address[2]
SRAM_ADDR[3]	PIN_AC6	SRAM Address[3]
SRAM_ADDR[4]	PIN_AD4	SRAM Address[4]
SRAM_ADDR[5]	PIN_AD5	SRAM Address[5]
SRAM_ADDR[6]	PIN_AE5	SRAM Address[6]
SRAM_ADDR[7]	PIN_AF5	SRAM Address[7]
SRAM_ADDR[8]	PIN_AD6	SRAM Address[8]
SRAM_ADDR[9]	PIN_AD7	SRAM Address[9]
SRAM_ADDR[10]	PIN_V10	SRAM Address[10]
SRAM_ADDR[11]	PIN_V9	SRAM Address[11]
SRAM_ADDR[12]	PIN_AC7	SRAM Address[12]
SRAM_ADDR[13]	PIN_W8	SRAM Address[13]
SRAM_ADDR[14]	PIN_W10	SRAM Address[14]
SRAM_ADDR[15]	PIN_Y10	SRAM Address[15]
SRAM_ADDR[16]	PIN_AB8	SRAM Address[16]
SRAM_ADDR[17]	PIN_AC8	SRAM Address[17]
SRAM_DQ[0]	PIN_AD8	SRAM Data[0]
SRAM_DQ[1]	PIN_AE6	SRAM Data[1]
SRAM_DQ[2]	PIN_AF6	SRAM Data[2]
SRAM_DQ[3]	PIN_AA9	SRAM Data[3]
SRAM_DQ[4]	PIN_AA10	SRAM Data[4]
SRAM_DQ[5]	PIN_AB10	SRAM Data[5]
SRAM_DQ[6]	PIN_AA11	SRAM Data[6]
SRAM_DQ[7]	PIN_Y11	SRAM Data[7]
SRAM_DQ[8]	PIN_AE7	SRAM Data[8]
SRAM_DQ[9]	PIN_AF7	SRAM Data[9]
SRAM_DQ[10]	PIN_AE8	SRAM Data[10]
SRAM_DQ[11]	PIN_AF8	SRAM Data[11]
SRAM_DQ[12]	PIN_W11	SRAM Data[12]
SRAM_DQ[13]	PIN_W12	SRAM Data[13]
SRAM_DQ[14]	PIN_AC9	SRAM Data[14]
SRAM_DQ[15]	PIN_AC10	SRAM Data[15]

SRAM_WE_N	PIN_AE10	SRAM Write Enable
SRAM_OE_N	PIN_AD10	SRAM Output Enable
SRAM_UB_N	PIN_AF9	SRAM High-byte Data Mask
SRAM_LB_N	PIN_AE9	SRAM Low-byte Data Mask
SRAM_CE_N	PIN_AC11	SRAM Chip Enable
Signal Name	**FPGA Pin No.**	**Description**
OTG_ADDR[0]	PIN_K7	ISP1362 Address[0]
OTG_ADDR[1]	PIN_F2	ISP1362 Address[1]
OTG_DATA[0]	PIN_F4	ISP1362 Data[0]
OTG_DATA[1]	PIN_D2	ISP1362 Data[1]
OTG_DATA[2]	PIN_D1	ISP1362 Data[2]
OTG_DATA[3]	PIN_F7	ISP1362 Data[3]
OTG_DATA[4]	PIN_J5	ISP1362 Data[4]
OTG_DATA[5]	PIN_J8	ISP1362 Data[5]
OTG_DATA[6]	PIN_J7	ISP1362 Data[6]
OTG_DATA[7]	PIN_H6	ISP1362 Data[7]
OTG_DATA[8]	PIN_E2	ISP1362 Data[8]
OTG_DATA[9]	PIN_E1	ISP1362 Data[9]
OTG_DATA[10]	PIN_K6	ISP1362 Data[10]
OTG_DATA[11]	PIN_K5	ISP1362 Data[11]
OTG_DATA[12]	PIN_G4	ISP1362 Data[12]
OTG_DATA[13]	PIN_G3	ISP1362 Data[13]
OTG_DATA[14]	PIN_J6	ISP1362 Data[14]
OTG_DATA[15]	PIN_K8	ISP1362 Data[15]
OTG_CS_N	PIN_F1	ISP1362 Chip Select
OTG_RD_N	PIN_G2	ISP1362 Read
OTG_WR_N	PIN_G1	ISP1362 Write
OTG_RST_N	PIN_G5	ISP1362 Reset
OTG_INT0	PIN_B3	ISP1362 Interrupt 0
OTG_INT1	PIN_C3	ISP1362 Interrupt 1
OTG_DACK0_N	PIN_C2	ISP1362 DMA Acknowledge 0
OTG_DACK1_N	PIN_B2	ISP1362 DMA Acknowledge 1
OTG_DREQ0	PIN_F6	ISP1362 DMA Request 0
OTG_DREQ1	PIN_E5	ISP1362 DMA Request 1
OTG_FSPEED	PIN_F3	USB Full Speed, 0 = Enable, Z = Disable
OTG_LSPEED	PIN_G6	USB Low Speed, 0 = Enable, Z = Disable
Signal Name	**FPGA Pin No.**	**Description**
LCD_DATA[0]	PIN_J1	LCD Data[0]
LCD_DATA[1]	PIN_J2	LCD Data[1]
LCD_DATA[2]	PIN_H1	LCD Data[2]
LCD_DATA[3]	PIN_H2	LCD Data[3]
LCD_DATA[4]	PIN_J4	LCD Data[4]
LCD_DATA[5]	PIN_J3	LCD Data[5]
LCD_DATA[6]	PIN_H4	LCD Data[6]
LCD_DATA[7]	PIN_H3	LCD Data[7]
LCD_RW	PIN_K4	LCD Read/Write Select, 0 = Write, 1 = Read
LCD_EN	PIN_K3	LCD Enable
LCD_RS	PIN_K1	LCD Command/Data Select, 0 = Command, 1 = Data
LCD_ON	PIN_L4	LCD Power ON/OFF
LCD_BLON	PIN_K2	LCD Back Light ON/OFF
Signal Name	**FPGA Pin No.**	**Description**
SD_DAT	PIN_AD24	SD Card Data
SD_DAT3	PIN_AC23	SD Card Data 3
SD_CMD	PIN_Y21	SD Card Command Signal
SD_CLK	PIN_AD25	SD Card Clock
Signal Name	**FPGA Pin No.**	**Description**
TDI	PIN_B14	CPLD -> FPGA (Data in)
TCS	PIN_A14	CPLD -> FPGA (CS)
TCK	PIN_D14	CPLD -> FPGA (Clock)
TDO	PIN_F14	FPGA -> CPLD (Data out)
Signal Name	**FPGA Pin No.**	**Description**
IRDA_TXD	PIN_AE24	IRDA Transmitter
IRDA_RXD	PIN_AE25	IRDA Receiver
Signal Name	**FPGA Pin No.**	**Description**

Altera DE2 Board Pin Table

HEX0[0]	PIN_AF10	Seven Segment Digit 0[0]
HEX0[1]	PIN_AB12	Seven Segment Digit 0[1]
HEX0[2]	PIN_AC12	Seven Segment Digit 0[2]
HEX0[3]	PIN_AD11	Seven Segment Digit 0[3]
HEX0[4]	PIN_AE11	Seven Segment Digit 0[4]
HEX0[5]	PIN_V14	Seven Segment Digit 0[5]
HEX0[6]	PIN_V13	Seven Segment Digit 0[6]
HEX1[0]	PIN_V20	Seven Segment Digit 1[0]
HEX1[1]	PIN_V21	Seven Segment Digit 1[1]
HEX1[2]	PIN_W21	Seven Segment Digit 1[2]
HEX1[3]	PIN_Y22	Seven Segment Digit 1[3]
HEX1[4]	PIN_AA24	Seven Segment Digit 1[4]
HEX1[5]	PIN_AA23	Seven Segment Digit 1[5]
HEX1[6]	PIN_AB24	Seven Segment Digit 1[6]
HEX2[0]	PIN_AB23	Seven Segment Digit 2[0]
HEX2[1]	PIN_V22	Seven Segment Digit 2[1]
HEX2[2]	PIN_AC25	Seven Segment Digit 2[2]
HEX2[3]	PIN_AC26	Seven Segment Digit 2[3]
HEX2[4]	PIN_AB26	Seven Segment Digit 2[4]
HEX2[5]	PIN_AB25	Seven Segment Digit 2[5]
HEX2[6]	PIN_Y24	Seven Segment Digit 2[6]
HEX3[0]	PIN_Y23	Seven Segment Digit 3[0]
HEX3[1]	PIN_AA25	Seven Segment Digit 3[1]
HEX3[2]	PIN_AA26	Seven Segment Digit 3[2]
HEX3[3]	PIN_Y26	Seven Segment Digit 3[3]
HEX3[4]	PIN_Y25	Seven Segment Digit 3[4]
HEX3[5]	PIN_U22	Seven Segment Digit 3[5]
HEX3[6]	PIN_W24	Seven Segment Digit 3[6]
HEX4[0]	PIN_U9	Seven Segment Digit 4[0]
HEX4[1]	PIN_U1	Seven Segment Digit 4[1]
HEX4[2]	PIN_U2	Seven Segment Digit 4[2]
HEX4[3]	PIN_T4	Seven Segment Digit 4[3]
HEX4[4]	PIN_R7	Seven Segment Digit 4[4]
HEX4[5]	PIN_R6	Seven Segment Digit 4[5]
HEX4[6]	PIN_T3	Seven Segment Digit 4[6]
HEX5[0]	PIN_T2	Seven Segment Digit 5[0]
HEX5[1]	PIN_P6	Seven Segment Digit 5[1]
HEX5[2]	PIN_P7	Seven Segment Digit 5[2]
HEX5[3]	PIN_T9	Seven Segment Digit 5[3]
HEX5[4]	PIN_R5	Seven Segment Digit 5[4]
HEX5[5]	PIN_R4	Seven Segment Digit 5[5]
HEX5[6]	PIN_R3	Seven Segment Digit 5[6]
HEX6[0]	PIN_R2	Seven Segment Digit 6[0]
HEX6[1]	PIN_P4	Seven Segment Digit 6[1]
HEX6[2]	PIN_P3	Seven Segment Digit 6[2]
HEX6[3]	PIN_M2	Seven Segment Digit 6[3]
HEX6[4]	PIN_M3	Seven Segment Digit 6[4]
HEX6[5]	PIN_M5	Seven Segment Digit 6[5]
HEX6[6]	PIN_M4	Seven Segment Digit 6[6]
HEX7[0]	PIN_L3	Seven Segment Digit 7[0]
HEX7[1]	PIN_L2	Seven Segment Digit 7[1]
HEX7[2]	PIN_L9	Seven Segment Digit 7[2]
HEX7[3]	PIN_L6	Seven Segment Digit 7[3]
HEX7[4]	PIN_L7	Seven Segment Digit 7[4]
HEX7[5]	PIN_P9	Seven Segment Digit 7[5]
HEX7[6]	PIN_N9	Seven Segment Digit 7[6]
Signal Name	**FPGA Pin No.**	**Description**
KEY[0]	PIN_G26	Pushbutton[0]
KEY[1]	PIN_N23	Pushbutton[1]
KEY[2]	PIN_P23	Pushbutton[2]
KEY[3]	PIN_W26	Pushbutton[3]
Signal Name	**FPGA Pin No.**	**Description**
LEDR[0]	PIN_AE23	LED Red[0]
LEDR[1]	PIN_AF23	LED Red[1]

Altera DE2 Board Pin Table

LEDR[2]	PIN_AB21	LED Red[2]
LEDR[3]	PIN_AC22	LED Red[3]
LEDR[4]	PIN_AD22	LED Red[4]
LEDR[5]	PIN_AD23	LED Red[5]
LEDR[6]	PIN_AD21	LED Red[6]
LEDR[7]	PIN_AC21	LED Red[7]
LEDR[8]	PIN_AA14	LED Red[8]
LEDR[9]	PIN_Y13	LED Red[9]
LEDR[10]	PIN_AA13	LED Red[10]
LEDR[11]	PIN_AC14	LED Red[11]
LEDR[12]	PIN_AD15	LED Red[12]
LEDR[13]	PIN_AE15	LED Red[13]
LEDR[14]	PIN_AF13	LED Red[14]
LEDR[15]	PIN_AE13	LED Red[15]
LEDR[16]	PIN_AE12	LED Red[16]
LEDR[17]	PIN_AD12	LED Red[17]
LEDG[0]	PIN_AE22	LED Green[0]
LEDG[1]	PIN_AF22	LED Green[1]
LEDG[2]	PIN_W19	LED Green[2]
LEDG[3]	PIN_V18	LED Green[3]
LEDG[4]	PIN_U18	LED Green[4]
LEDG[5]	PIN_U17	LED Green[5]
LEDG[6]	PIN_AA20	LED Green[6]
LEDG[7]	PIN_Y18	LED Green[7]
LEDG[8]	PIN_Y12	LED Green[8]
Signal Name	**FPGA Pin No.**	**Description**
CLOCK_27	PIN_D13	On Board 27 MHz
CLOCK_50	PIN_N2	On Board 50 MHz
EXT_CLOCK	PIN_P26	External Clock
Signal Name	**FPGA Pin No.**	**Description**
UART_RXD	PIN_C25	UART Receiver
UART_TXD	PIN_B25	UART Transmitter
Signal Name	**FPGA Pin No.**	**Description**
PS2_CLK	PIN_D26	PS2 Data
PS2_DAT	PIN_C24	PS2 Clock
Signal Name	**FPGA Pin No.**	**Description**
I2C_SCLK	PIN_A6	I2C Data
I2C_SDAT	PIN_B6	I2C Clock
Signal Name	**FPGA Pin No.**	**Description**
TD_DATA[0]	PIN_J9	TV Decoder Data[0]
TD_DATA[1]	PIN_E8	TV Decoder Data[1]
TD_DATA[2]	PIN_H8	TV Decoder Data[2]
TD_DATA[3]	PIN_H10	TV Decoder Data[3]
TD_DATA[4]	PIN_G9	TV Decoder Data[4]
TD_DATA[5]	PIN_F9	TV Decoder Data[5]
TD_DATA[6]	PIN_D7	TV Decoder Data[6]
TD_DATA[7]	PIN_C7	TV Decoder Data[7]
TD_HS	PIN_D5	TV Decoder H_SYNC
TD_VS	PIN_K9	TV Decoder V_SYNC
TD_RESET	PIN_C4	TV Decoder Reset
Signal Name	**FPGA Pin No.**	**Description**
VGA_R[0]	PIN_C8	VGA Red[0]
VGA_R[1]	PIN_F10	VGA Red[1]
VGA_R[2]	PIN_G10	VGA Red[2]
VGA_R[3]	PIN_D9	VGA Red[3]
VGA_R[4]	PIN_C9	VGA Red[4]
VGA_R[5]	PIN_A8	VGA Red[5]
VGA_R[6]	PIN_H11	VGA Red[6]
VGA_R[7]	PIN_H12	VGA Red[7]
VGA_R[8]	PIN_F11	VGA Red[8]
VGA_R[9]	PIN_E10	VGA Red[9]
VGA_G[0]	PIN_B9	VGA Green[0]
VGA_G[1]	PIN_A9	VGA Green[1]
VGA_G[2]	PIN_C10	VGA Green[2]

VGA_G[3]	PIN_D10	VGA Green[3]
VGA_G[4]	PIN_B10	VGA Green[4]
VGA_G[5]	PIN_A10	VGA Green[5]
VGA_G[6]	PIN_G11	VGA Green[6]
VGA_G[7]	PIN_D11	VGA Green[7]
VGA_G[8]	PIN_E12	VGA Green[8]
VGA_G[9]	PIN_D12	VGA Green[9]
VGA_B[0]	PIN_J13	VGA Blue[0]
VGA_B[1]	PIN_J14	VGA Blue[1]
VGA_B[2]	PIN_F12	VGA Blue[2]
VGA_B[3]	PIN_G12	VGA Blue[3]
VGA_B[4]	PIN_J10	VGA Blue[4]
VGA_B[5]	PIN_J11	VGA Blue[5]
VGA_B[6]	PIN_C11	VGA Blue[6]
VGA_B[7]	PIN_B11	VGA Blue[7]
VGA_B[8]	PIN_C12	VGA Blue[8]
VGA_B[9]	PIN_B12	VGA Blue[9]
VGA_CLK	PIN_B8	VGA Clock
VGA_BLANK	PIN_D6	VGA BLANK
VGA_HS	PIN_A7	VGA H_SYNC
VGA_VS	PIN_D8	VGA V_SYNC
VGA_SYNC	PIN_B7	VGA SYNC
Signal Name	**FPGA Pin No.**	**Description**
AUD_ADCLRCK	PIN_C5	Audio CODEC ADC LR Clock
AUD_ADCDAT	PIN_B5	Audio CODEC ADC Data
AUD_DACLRCK	PIN_C6	Audio CODEC DAC LR Clock
AUD_DACDAT	PIN_A4	Audio CODEC DAC Data
AUD_XCK	PIN_A5	Audio CODEC Chip Clock
AUD_BCLK	PIN_B4	Audio CODEC Bit-Stream Clock
Signal Name	**FPGA Pin No.**	**Description**
ENET_DATA[0]	PIN_D17	DM9000A DATA[0]
ENET_DATA[1]	PIN_C17	DM9000A DATA[1]
ENET_DATA[2]	PIN_B18	DM9000A DATA[2]
ENET_DATA[3]	PIN_A18	DM9000A DATA[3]
ENET_DATA[4]	PIN_B17	DM9000A DATA[4]
ENET_DATA[5]	PIN_A17	DM9000A DATA[5]
ENET_DATA[6]	PIN_B16	DM9000A DATA[6]
ENET_DATA[7]	PIN_B15	DM9000A DATA[7]
ENET_DATA[8]	PIN_B20	DM9000A DATA[8]
ENET_DATA[9]	PIN_A20	DM9000A DATA[9]
ENET_DATA[10]	PIN_C19	DM9000A DATA[10]
ENET_DATA[11]	PIN_D19	DM9000A DATA[11]
ENET_DATA[12]	PIN_B19	DM9000A DATA[12]
ENET_DATA[13]	PIN_A19	DM9000A DATA[13]
ENET_DATA[14]	PIN_E18	DM9000A DATA[14]
ENET_DATA[15]	PIN_D18	DM9000A DATA[15]
ENET_CLK	PIN_B24	DM9000A Clock 25 MHz
ENET_CMD	PIN_A21	DM9000A Command/Data Select, 0 = Command, 1 = Data
ENET_CS_N	PIN_A23	DM9000A Chip Select
ENET_INT	PIN_B21	DM9000A Interrupt
ENET_RD_N	PIN_A22	DM9000A Read
ENET_WR_N	PIN_B22	DM9000A Write
ENET_RST_N	PIN_B23	DM9000A Reset
Signal Name	**FPGA Pin No.**	**Description**
GPIO_0[0]	PIN_D25	GPIO Connection 0[0]
GPIO_0[1]	PIN_J22	GPIO Connection 0[1]
GPIO_0[2]	PIN_E26	GPIO Connection 0[2]
GPIO_0[3]	PIN_E25	GPIO Connection 0[3]
GPIO_0[4]	PIN_F24	GPIO Connection 0[4]
GPIO_0[5]	PIN_F23	GPIO Connection 0[5]
GPIO_0[6]	PIN_J21	GPIO Connection 0[6]
GPIO_0[7]	PIN_J20	GPIO Connection 0[7]
GPIO_0[8]	PIN_F25	GPIO Connection 0[8]
GPIO_0[9]	PIN_F26	GPIO Connection 0[9]

GPIO_0[10]	PIN_N18	GPIO Connection 0[10]
GPIO_0[11]	PIN_P18	GPIO Connection 0[11]
GPIO_0[12]	PIN_G23	GPIO Connection 0[12]
GPIO_0[13]	PIN_G24	GPIO Connection 0[13]
GPIO_0[14]	PIN_K22	GPIO Connection 0[14]
GPIO_0[15]	PIN_G25	GPIO Connection 0[15]
GPIO_0[16]	PIN_H23	GPIO Connection 0[16]
GPIO_0[17]	PIN_H24	GPIO Connection 0[17]
GPIO_0[18]	PIN_J23	GPIO Connection 0[18]
GPIO_0[19]	PIN_J24	GPIO Connection 0[19]
GPIO_0[20]	PIN_H25	GPIO Connection 0[20]
GPIO_0[21]	PIN_H26	GPIO Connection 0[21]
GPIO_0[22]	PIN_H19	GPIO Connection 0[22]
GPIO_0[23]	PIN_K18	GPIO Connection 0[23]
GPIO_0[24]	PIN_K19	GPIO Connection 0[24]
GPIO_0[25]	PIN_K21	GPIO Connection 0[25]
GPIO_0[26]	PIN_K23	GPIO Connection 0[26]
GPIO_0[27]	PIN_K24	GPIO Connection 0[27]
GPIO_0[28]	PIN_L21	GPIO Connection 0[28]
GPIO_0[29]	PIN_L20	GPIO Connection 0[29]
GPIO_0[30]	PIN_J25	GPIO Connection 0[30]
GPIO_0[31]	PIN_J26	GPIO Connection 0[31]
GPIO_0[32]	PIN_L23	GPIO Connection 0[32]
GPIO_0[33]	PIN_L24	GPIO Connection 0[33]
GPIO_0[34]	PIN_L25	GPIO Connection 0[34]
GPIO_0[35]	PIN_L19	GPIO Connection 0[35]
GPIO_1[0]	PIN_K25	GPIO Connection 1[0]
GPIO_1[1]	PIN_K26	GPIO Connection 1[1]
GPIO_1[2]	PIN_M22	GPIO Connection 1[2]
GPIO_1[3]	PIN_M23	GPIO Connection 1[3]
GPIO_1[4]	PIN_M19	GPIO Connection 1[4]
GPIO_1[5]	PIN_M20	GPIO Connection 1[5]
GPIO_1[6]	PIN_N20	GPIO Connection 1[6]
GPIO_1[7]	PIN_M21	GPIO Connection 1[7]
GPIO_1[8]	PIN_M24	GPIO Connection 1[8]
GPIO_1[9]	PIN_M25	GPIO Connection 1[9]
GPIO_1[10]	PIN_N24	GPIO Connection 1[10]
GPIO_1[11]	PIN_P24	GPIO Connection 1[11]
GPIO_1[12]	PIN_R25	GPIO Connection 1[12]
GPIO_1[13]	PIN_R24	GPIO Connection 1[13]
GPIO_1[14]	PIN_R20	GPIO Connection 1[14]
GPIO_1[15]	PIN_T22	GPIO Connection 1[15]
GPIO_1[16]	PIN_T23	GPIO Connection 1[16]
GPIO_1[17]	PIN_T24	GPIO Connection 1[17]
GPIO_1[18]	PIN_T25	GPIO Connection 1[18]
GPIO_1[19]	PIN_T18	GPIO Connection 1[19]
GPIO_1[20]	PIN_T21	GPIO Connection 1[20]
GPIO_1[21]	PIN_T20	GPIO Connection 1[21]
GPIO_1[22]	PIN_U26	GPIO Connection 1[22]
GPIO_1[23]	PIN_U25	GPIO Connection 1[23]
GPIO_1[24]	PIN_U23	GPIO Connection 1[24]
GPIO_1[25]	PIN_U24	GPIO Connection 1[25]
GPIO_1[26]	PIN_R19	GPIO Connection 1[26]
GPIO_1[27]	PIN_T19	GPIO Connection 1[27]
GPIO_1[28]	PIN_U20	GPIO Connection 1[28]
GPIO_1[29]	PIN_U21	GPIO Connection 1[29]
GPIO_1[30]	PIN_V26	GPIO Connection 1[30]
GPIO_1[31]	PIN_V25	GPIO Connection 1[31]
GPIO_1[32]	PIN_V24	GPIO Connection 1[32]
GPIO_1[33]	PIN_V23	GPIO Connection 1[33]
GPIO_1[34]	PIN_W25	GPIO Connection 1[34]
GPIO_1[35]	PIN_W23	GPIO Connection 1[35]