UNIVERSIDADE FEDERAL DO PARÁ
INSTITUTO DE CIÊNCIAS EXATAS E NATURAIS
FACULDADE DE COMPUTAÇÃO
CURSO DE BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

Carla Cristina da Cruz Araujo
Karlyanne Paoola Silva Braga

CONCEITOS DE COMPUTAÇÃO RECONFIGURÁVEL COM APLICAÇÃO PRÁTICA EM *HARDWARE* DIDÁTICO

Belém - PA
2017

Carla Cristina da Cruz Araujo
Karlyanne Paoola Silva Braga

CONCEITOS DE COMPUTAÇÃO RECONFIGURÁVEL COM APLICAÇÕES PRÁTICAS EM *HARDWARE* DIDÁTICO

Trabalho de conclusão de curso de graduação apresentado para obtenção do título de Bacharel em Ciência da Computação. Instituto de Ciências Exatas e Naturais. Faculdade de Computação. Universidade Federal do Pará.
Orientador Prof. Dr. Dionne Cavalcante Monteiro
Co-orientador Msc. Leonardo Sarraff Nunes de Moraes

Belém - PA
2017

CONCEITOS DE COMPUTAÇÃO RECONFIGURÁVEL COM APLICAÇÕES PRÁTICAS EM *HARDWARE* DIDÁTICO

Carla Cristina da Cruz Araujo
Karlyanne Paoola Silva Braga

Trabalho de conclusão de curso apresentado ao Instituto de Ciências Exatas e Naturais da Universidade Federal do Pará como requisito para a obtenção do título de Bacharel em Ciência da Computação. Julgado em ___ de abril de 2017, com o seguinte conceito _____.

_____

Prof. Dr. Dionne Cavalcante Monteiro
ORIENTADOR – ICEN – UFPA

_____

Msc. Leonardo Sarraff Nunes de Moraes
CO-ORIENTADOR

_____

Profa. Dra. Regiane Silva Kawasaki Frances
CONVIDADA – ICEN – UFPA

_____

Prof. Dr. Adalbery Rodrigues Castro
CONVIDADO – ITEC – UFPA

Belém - PA
2017

AGRADECIMENTOS CARLA

Primeiramente, quero agradecer a Deus, por ter me dado à força que precisei durante o período todo esse período.

Aos meus pais José Carlos e Maria do Socorro, que me incentivaram e me apoiaram em todas as minhas decisões, em especial a minha rainha Maria que sem a força e o apoio incondicional dela não teria chego até aqui, por ter aguentado meu stress e consolado meus choros nos momentos de aflição. A minha amada avó Firmina Almeida que ao lado dos meus pais, foi uma das minhas maiores incentivadoras.

Ao meu irmão José Carlos, mesmo com suas brincadeiras que eu iria me forma depois do meu filho, me incentivou a ir até o fim.

Agradecer aos amigos irmãos que o curso me deu, Carlos Eduardo, Adilson Souza, Lemuell, Wellson Sérgio, Aden e Bruna, por estarem sempre comigo e me ajudarem sempre que precisei.

Aos meus amigos do LAAI, Yvan e Anderson pelos momentos de descontração e conversas, e claro almoços no RU.

A Karlyanne Paoola que além de parceira do TCC e amiga, se tornou uma irmã, estando ao meu lado para o que precisei, e dividindo as angústias e aflições que o TCC nos trouxe.

Ao professor e meu orientador Dionne Monteiro, pela confiança no trabalho proposto, e pela ajuda durante todo o processo.

Ao meu companheiro Anderson Pinheiro, que me ajudou e me incentivou sempre. E ao meu maior motivo de lutar pela minha formação e ir em busca do melhor sempre, meu Filho Enzo.

“Não é sobre chegar no topo do mundo
E saber que venceu
É sobre escalar e sentir
Que o caminho te fortaleceu
É sobre ser abrigo
E também ter morada em outros corações
E assim ter amigos contigo
Em todas as situações

A gente não pode ter tudo
Qual seria a graça do mundo se fosse assim?
Por isso, eu prefiro sorrisos
E os presentes que a vida trouxe
Pra perto de mim.”

Ana Vilela

AGRADECIMENTOS KARLYANNE

Agradecer a Deus, primeiramente, que tem me dado força e sabedoria sempre que necessito.

Dediquei este trabalho "*in memorian*" ao meu avô materno (Basílio), meu primo (André) e amigos de graduação (Ariane, Bruno, Renato e Thamirys).

A minha amada mãezinha, Ilsa Braga, por me motivar sempre, e por ser o maior exemplo de sabedoria, força e dedicação para mim, por todo o seu suporte financeiro, me ajudando sempre com muito amor e carinho. Ao meu tio Manoel Silva que sempre foi minha referência paterna, com seu apoio incondicional em tudo na minha vida, além de um pai é um grande amigo. Ao meu irmão Anderson e minha prima/irmã Cristiane que sempre me apoiaram, ajudaram e incentivaram. As minhas tias Maria, Fátima e Cenilda que são como mães me ajudando em tudo que podem sempre e nunca deixando de acredita em mim. A minha avó Raimunda Ferreira, que sempre será minha maior referência de mulher forte e guerreira, que ora por mim todos os dias. Agraço a Deus por suas vidas.

A todos meus outros familiares. Meus tios Rui Braga, Raul Braga, Carlos Silva e Evair Silva. Minhas tias Socorro Braga, Sebastiana Azevedo, Cristina Rodrigues e Ana Souza (quase tia). Meus primos Alexandre, Adriele, Wanessa, Arthur, Andrey e Andrew. E por fim meus irmãos Paulo, Pablo, Peterson e Pâmela que estiveram comigo desde o início dessa caminhada, com muito apoio e carinho.

Ao meu prezado professor e orientador, Dionne Cavalcante Monteiro, por me ceder espaço em seu projeto e por toda a confiança, conhecimento e apoio dados a mim durante o último ano. Em especial, por sua disponibilidade e seriedade na correção deste trabalho. Um grande exemplo a ser seguindo não apenas como profissional, mas como pessoa.

A professora e convidada especial, Regiane Kawasaki, por todo conhecimento passado nas matérias de sistema de computação, sistemas operacionais e programação II, sempre ministrando suas disciplinas com seriedade e dedicação no aprendizado dos alunos, minha maior referência feminina na faculdade de computação.

A todos meus professores e colegas de curso pela troca de conhecimento. Aos amigos que este curso me deu, Diego Souza, Marjorie Marques, Bárbara Souza, Rodrigo Silva, Elder Ferreira, os meus amigos Maratoneiros, todos os Computeiros por todos os momentos que passamos juntos, de alegrias e tristezas. Amigos que irei levar para vida inteira.

A Carla Araujo que além de parceira do TCC, se tornou uma grande amiga e irmã, estando ao meu lado para o que precisei, dando forças sempre que pensei em desistir,

acreditando em mim sempre. Obrigada pela confiança, pela união, companheirismo e por me aturar em todos os momentos.

Aos meus grandes amigos, Wlad Silva, Sylmara Vales, Luana Ribeiro, Stephanie Rodrigues, Carlos Braga, Helen Ferreira, Jéssyca Melo, Suanne Santos e Danilo Andrade por me ajudarem quando deles precisei, e sempre estarem comigo em todos os momentos.

Por fim, não menos importante, a minha amada Elizabeth Branches, por todo suporte, incentivo, companheirismo, amor e carinho nesses últimos anos. Agradeço a Deus por sua vida e pelo nosso amor.

A todos que direta ou indiretamente fizeram parte da minha formação, o meu muito obrigada.

RESUMO

A computação reconfigurável é como uma solução intermediária entre as soluções em *hardware* e *software*, com objetivos, metas e motivações relacionados à melhoria de desempenho, flexibilidade, generalidade, eficiência, custo e outros. Este trabalho tem como objetivo a criação de um tutorial da linguagem de descrição paralela VHDL com aplicações práticas em *hardware* didático, introduzindo o conceito do modelo de computação reconfigurável. O trabalho apresenta as características, conceitos básicos e peculiaridades da linguagem de descrição paralela VHDL, colocando em prática em laboratórios que aumentam o grau de dificuldade conforme o aluno vai se adaptando à linguagem, ao final o aluno deverá desenvolver uma aplicação que consiste em um jogo tradicional da década de setenta chamado *Pong*, o desenvolvimento dessa aplicação é feito utilizando os conhecimentos adquiridos ao decorrer dos laboratórios práticos do trabalho.

Palavras-chaves: *Computação reconfigurável; linguagem VHDL; programação paralela; tutorial.*

ABSTRACT

The computation reconfigurable is as an intermediate solution among the solutions in *hardware* and *software*, with objectives, goals and motivations related with the acting improvement, flexibility, generality, efficiency, cost and other. This work has as the objective the creation of a tutorial of the parallel description language VHDL with practical applications in didactic *hardware*, introducing the concept of the model of computation Reconfigurable. The work presents the characteristics, basic concepts and peculiarities of the parallel description language VHDL, putting in practice with laboratories that increase the degree of difficulty as the student goes if adapting the language, at the end the student should develop an application that consists of a traditional game of the decade of seventy call Pong, the development of this application is done using the knowledge acquired over the course of practical labs work.

Key words: Reconfigurable Computing; VHDL language; parallel programming; tutorial.

LISTA DE FIGURAS

Figura 2.1 Modulo lógico genérico. 29
Figura 2.2 Estrutura de um programa em VHDL 30
Figura 2.3 Entidade de Projeto da Linguagem VHDL. 31
Figura 2.4 Exemplo de entidade de Projeto do bloco Modulo. 33
Figura 2.5 Modos de operação de uma PORT. 34
Figura 2.6 Exemplo de modelo estrutural 36
Figura 2.7 Declaração VHDL onde estão indicadas algumas das estruturas e comandos. 44
Figura 3.1 Placa DE0 com indicadores de seus componentes. 52
Figura 3.2 Tela inicial do QuartusII. 53
Figura 3.3 Tela de criação do projeto. 54
Figura 3.4 Adição do nome e local do projeto. 54
Figura 3.5 Adição de Componente.. 55
Figura 3.6 Seleção da FPGA 55
Figura 3.7 Tela 4 de criação de Projeto. 56
Figura 3.8 Tela 5 Sumário 56
Figura 3.9 Criação do arquivo. 57
Figura 3.10 Código em VHDL 57
Figura 3.11 Iniciar Compilação. 58
Figura 3.12 Salvar arquivo á ser compilado. 58
Figura 3.13 Janelas de antes e depois da compilação. 58
Figura 3.14 Entrar na janela de adição dos pinos. 59
Figura 3.15 Inserção dos pinos. 60
Figura 3.16 Placa DE0. 60
Figura 3.17 Abrir a janela de programação. 60
Figura 3.18 Seleção do Hardware da placa. 61
Figura 3.19 Programação do dispositivo. 61
Figura 3.20 Resultado na placa DE0. 62
Figura 3.21 Código em VHDl no Quartus II. 63
Figura 3.22 Conexão dos pinos da FPGA aos sinais lógicos no VHDL. 64
Figura 3.23 Resultado na FPGA DE0 da Conexão das chaves aos LEDS 64
Figura 3.24 Multiplexer 2-to-1: a) Circuito; b) Tabela verdade; c) Símbolo 65
Figura 3.25 Resultado na FPGA DE0 do multiplexador 2-to-1 66

Figura 3.26 Multiplexação de vetor digital com 3 elementos. 67
Figura 3.27 Resultado na FPGA multiplexador 2-to-1 com entrada e saída de vetores..... 68
Figura 3.28 Resultado na FPGA multiplexador 2-to-1 com entrada e saída de vetores com o BUTTON pressionado..... 69
Figura 3.29 Multiplexador 3-to-1 utilizando dois multiplexadores 2-to-1. 69
Figura 3.30 Resultado na FPGA multiplexador 2-to-1 com entrada e saída de vetores com o BUTTON pressionado. 71
Figura 3.31 Multiplexador de 2 bits 3-to-1. 71
Figura 3.32 Decodificador 2 bit para 7 segmentos. 72
Figura 3.33 Esquema de ligação entre o primeiro display de 7 segmentos e o FPGA. 72
Figura 3.34 Display de 7 segmentos e suas respectivas numerações de LEDs. 73
Figura 3.35 Resultado conversor 2 bits para 7 segmentos. 75
Figura 3.36 Resultado na FPGA do conversor 2 bit para 7 segmentos com WHEN. 76
Figura 3.37 Circuito selecionador de caracteres para display de 7 segmentos..... 76
Figura 3.38 Resultado na FPGA do Multiplexador de caracteres com saída para display de 7 segmentos. 79
Figura 3.39 Resultado na FPGA do decodificador binário decimal. 81
Figura 3.40 Circuito parcial que converte binário em decimal. 82
Figura 3.41 Resultado na FPGA do circuito parcial do decodificar binário para decimal com 2 dígitos. 83
Figura 3.42 Somador completo de 2 bits. 84
Figura 3.43 Entradas e saídas do somador completo de 2 bits. 84
Figura 3.44 Somador completo de 4 bits utilizando o componente somador completo de 2 bits. 85
Figura 3.45 Inclusão de arquivos existentes em um novo projeto VHDL. 86
Figura 3.46 Resultado Componente que reutiliza o somador completo de 2 bits. 88
Figura 3.47 Resultado na FPGA do conversor BCD de 4 bits. 90
Figura 3.48 Latch RS. 91
Figura 3.49 Resultado latch RS na placa DE0. 92
Figura 3.49 Latch D a partir do latch RS. 92
Figura 3.51 Resultado na FPGA do lach D. 94
Figura 3.52 Circuito Integrado 74LS74 que contém 2 flip-flops D. 94
Figura 3.53 Resultado na DE0 do Flip-floptipo D completo. 96
Figura 3.54 Registrador de 4 bits paralelo utilizando flip-flopD. 96

Figura 3.55 Resultado na placa DE0 do Registrador paralelo de 4 bits com flip-floptipo D...98
Figura 3.56 Registrador de deslocamento de 4 bits.....99
Figura 3.57 Resultado na placa DE0 do Registrador de deslocamento de 4 bits. 100
Figura 3.58 Contador de 4 bits com flip flop D. 102
Figura 3.59 Resultado do Contador Up de 4 bits com flip flop D.....104
Figura 3.60 Resultado na placa DE0 do código do Contador Up de 4 bits com flip flop D alternativo. 106
Figura 3.61 Resultado na placa DE0 do Contador de N bits. 107
Figura 3.62 Distribuição dos sinais de clock na placa DE0. 108
Figura 3.63 Resultado na placa DE0 do clock automático. 108
Figura 3.64 Conexão VGA com a FPGA DE0.....110
Figura 3.65 Localização do Qsys no Quartus II. 113
Figura 3.66 Tela inicial o Qsys.....113
Figura 3.67 Remove o componente padrão. 114
Figura 3.68 Adição do componente Avalon ALTPLL. 114
Figura 3.69 Ajuste do clock.....115
Figura 3.70 Limpar as entradas/saídas desnecessárias. 115
Figura 3.71 Ajuste da saída do clock.....116
Figura 3.72 Componente padrão customizado com os valores selecionados.....116
Figura 3.73 Alterar nome das entradas / saídas do componente. 117
Figura 3.74 Visualizar o código em VHDL 117
Figura 3.75 Código em VHDL do componente PLL gerado. 118
Figura 3.76 Gerar o componente PLL. 118
Figura 3.77 Clicar no botão Generate para gravar os arquivos no código. 119
Figura 3.78 Adição do arquivo ao projeto.....119
Figura 3.79 Seleção do arquivo correto. 120
Figura 3.80 Verifica se o arquivo foi adicionado corretamente. 120
Figura 3.81 Inicio da configuração.....122
Figura 3.82 Seleção das opções de dispositivo e pinos 123
Figura 3.83 Configuração da dupla finalidade dos pinos 124
Figura 3.84 Configuração de entrada e saída regular dos pinos.....125
Figura 3.85 Resultado na placa DE0 125
Figura 3.86 Resultado na placa do código VGA.vhd do quadrado 129
Figura 3.87 Resultado do jogo simples na placa 135

Figura 4.1 Resultado na tela do desenho da bola 146
Figura 4.2 Resultado na tela do desenho da barra 147
Figura 4.3 Resultado na tela da bola batendo na parte superior 148
Figura 4.4 Resultado na tela da bola batendo na parte inferior 149
Figura 4.5 Resultado na tela da bola batendo na parte direita. 149
Figura 4.6 Resultado na tela da bola batendo na barra 150
Figura 4.7 Resultado na tela da bola da redução da barra 151
Figura 4.8 Resultado na tela da pontuação 152
Figura 4.9 Resultado na tela da bola batendo na barra. 153
Figura 4.10 Resultado na placa da atualização dos leds. 155
Figura 4.13 Resultado final da aplicação 156

LISTA DE QUADROS

Quadro 2.1 Declaração do uso de pacotes e bibliotecas 32
Quadro 2.2 Uma declaração genérica para o módulo da Figura 2.1. 33
Quadro 2.3 Declaração genérica de uma entidade. 35
Quadro 2.4 Exemplo de declaração de arquitetura. 35
Quadro 2.5 Descrição estrutural da Unidade_AOI. 37
Quadro 2.6 Declaração de componente 37
Quadro 2.7 Declaração de componente AND2 38
Quadro 2.8 Declaração de componente OR2. 38
Quadro 2.9 Descrição de sistemas sequenciais cujo comando fundamental é o “PROCESS”.39
Quadro 2.10 Ligação da arquitetura á entidade. 41
Quadro 2.11 Exemplo retirado do livro d‟Amore. 42
Quadro 2.12 Exemplo VHDL não sensível a Espaços em Branco. 45
Quadro 2.13 Exemplo VHDL de comentário. 46
Quadro 2.14 Exemplos de identificadores. 46
Quadro 2.15 Comando IF 47
Quadro 2.16 Exemplo -Comando IF then 47
Quadro 2.17 Estrutura comando If then Else 48
Quadro 2.18 Exemplo - Comando If then Else 48
Quadro 2.19 Estrutura comando case 49
Quadro 2.20 Exemplo de Comando Case 49
Quadro 2.21 Exemplo de Comando While Loop 50
Quadro 2.22 Estrutura For loop 51
Quadro 2.23 Exemplo de Comando For Loop 51
Quadro 3.1 Código 01- conexão das chaves aos LEDs na placa DE0 63
Quadro 3.2 Código 02 – Multiplexador 2-to-1 em VHDL. 65
Quadro 3.3 Código 03 – Multiplexador 2-to-1 com entrada e saída de vetores. 67
Quadro 3.4 Código 04– Multiplexador de 2 bits 3-to-1 69
Quadro 3.5 Código 05 – Conversor 2 bits para 7 segmentos. 73
Quadro 3.6 Código06 – Conversor 2 bit para 7 segmentos com WHEN. 75
Quadro 3.7 Código 07 – Multiplexador de caracteres com saída para display de 7 segmentos. 76

Quadro 3.8 Código 08 – Decodificador Binário Decimal. 79
Quadro 3.9 Código 09 – Circuito parcial do decodificar binário para decimal com 2 dígitos. 82
Quadro 3.10 Código 10 – Somador completo de 2 dígitos 84
Quadro 3.11 Código 11 – Componente que reutiliza o somador completo de 2 bits. 86
Quadro 3.12 Código 12 – Conversor BCD de 4 bits. 88
Quadro 3.13 Código 13 – Conversor BCD de 4 bits. 91
Quadro 3.14 Código 14 – Conversor BCD de 4 bits com Latch D. 93
Quadro 3.15 Código 15 – Flip-floptipo D completo. 94
Quadro 3.16 Código 16 – Registrador paralelo de 4 bits com flip-floptipo D. 97
Quadro 3.17 Código 17 – Registrador de deslocamento de 4 bits com flip-flop tipo D. 99
Quadro 3.18 Código 18 – Registrador de deslocamento de 8 bits. 100
Quadro 3.19 Código 19 – Flip flop D 102
Quadro 3.20 Código 20 – Contador Up de 4 bits com Flip flop D. 103
Quadro 3.21 Código 21 – Contador Up de 4 bits com flip-flop D alternativo. 105
Quadro 3.22 Código 22 – Contador de N bits. 106
Quadro 3.23 Código 23 – SYNC.vhd linha cruzada 111
Quadro 3.24 Código 24 – VGA.vhd. linha cruzada 120
Quadro 3.25 Código 25 – pcg.vhd. quadrado 126
Quadro 3.26 Código 26 – SYNC.vhd. quadrado 127
Quadro 3.27 Código 27 – VGA.VHD do quadrado. 128
Quadro 3.28 Código 28 – PCG.vhd jogo simples 129
Quadro 3.29 Código 27 – SYNC.vhd do jogo simples 130
Quadro 3.30 Código VGA jogo simples 134
Quadro 4.1 código PCG.vhd jogo pong 136
Quadro 4.2 Parâmetros de entrada e saída da bola. 138
Quadro 4.3 Procedimento bola. 138
Quadro 4.4 Parâmetros de entrada e saída da barra. 139
Quadro 4.5 Procedimento barra 139
Quadro 4.6 Código Sync.vhd jogo pong 139
Quadro 4.7 Declarações de entrada e saída 144
Quadro 4.8 Declarações de entrada e saída 144
Quadro 4.9 Sinais atribuídos as funcionalidades da bola 144
Quadro 4.10 Sinais atribuídos as funcionalidades da barra 145
Quadro 4.11 Sinais atribuídos para marcação dos pontos 145

Quadro 4.12 Desenha a bola 146
Quadro 4.13 Desenha barra 146
Quadro 4.11 Limite da bola 147
Quadro 4.15 Caso a bola bata na parte superior 148
Quadro 4.16 Caso a bola bata na parte inferior 148
Quadro 4.17 Caso a bola bata na lateral direita 149
Quadro 4.18 Caso a bola bata na barra 150
Quadro 4.19 Redução da barra 150
Quadro 4.20 Caso a bola bata na lateral esquerda 151
Quadro 4.21 Controle da barra 152
Quadro 4.22 Movimento barra 152
Quadro 4.23 Controle de outros objetos 153
Quadro 4.24 Atualização dos LEDs 154
Quadro 4.25 Declaração de entrada e saídas do código VGA.vhd. 155
Quadro 4.26 Declaração dos componentes. 155
Quadro 4.24 Interligação das entradas e saídas 156

LISTA DE TABELAS

Tabela 2.1 A estrutura de uma descrição VHDL com os seus blocos 30
Tabela 2.2 Pacotes mais utilizados 31
Tabela 2.3 Tipos de PORT mais utilizados. 34
Tabela 2.4 Palavras reservadas em VHDL. 44
Tabela 2.5 Símbolos definidos em VHDL.. 44
Tabela 2.6 Conexão dos pinos para o Exemplo- Comando If then 47
Tabela 2.7 Conexão dos pinos para o Exemplo- Comando If then Else 49
Tabela 3.1 Conexão entre os sinais VHDL e os pinos da FGPA para o Multiplexador 2-to-1. 66
Tabela 3.2 Conexão entre os sinais VHDL e os pinos do FPGA para o multiplexador 2-to-1 com entrada e saída de vetores. 68
Tabela 3.3 Conexão entre os sinais VHDL e os pinos do FPGA para o Multiplexador de 2 bits 3-to-1. 70
Tabela 3.4 Tabela verdade a ser ilustrada.....71
Tabela 3.5 Tabela verdade.....72
Tabela 3.6 Conexões dos sinais em VHDL do Conversor 2 bits para 7 segmentos.....74
Tabela 3.7 Conexões dos sinais em VHDL do Conversor 2 bits para 7 segmentos.....78
Tabela 3.8 Tabela-verdade da Tarefa 3. 79
Tabela 3.9 Conexões dos sinais em VHDL do Decodificador Binário Decimal. 80
Tabela 3.10 Sinais em binário e seus respectivos em decimal. 81
Tabela 3.11 Conexões dos sinais em VHDL do Circuito parcial do decodificar binário para decimal com 2 dígitos.....83
Tabela 3.12 Tabela-verdade do somador completo de 2 bits. 84
Tabela 3.13 Pin Planner do somador completo de 4 bits. 87
Tabela 3.14 Pin Planner do conversor BCD de 4 bits. 89
Tabela 3.15 Configuração dos pinos do latch RS.....91
Tabela 3.16 Tabela verdade para análise. 91
Tabela 3.17 Tabela verdade para análise do latch D. 93
Tabela 3.18 Configuração dos pinos do Flip-floptipo D completo. 95
Tabela 3.19 Configuração dos pinos do Registrador paralelo de 4 bits com flip-floptipo D...97
Tabela 3.20 Conexões para o Registrador de deslocamento de 4 bits.....100
Tabela 3.21 Conexões para o Contador Up de 4 bits com flip flop D.....104

Tabela 3.22 Configuração dos pinos do Contador Up de 4 bits com flip flop D alternativo. 105
Tabela 3.23 General timing 109
Tabela 3.24 Horizontal timing (line) Polarity of horizontal sync pulse is positive. 109
Tabela 3.25 Vertical timing (frame) Polarity of vertical sync pulse is positive. 109
Tabela 3.26 Conexões VGA com os pinos da placa 121
Tabela 3.27 Conexão dos pinos VGA. 134

LISTA DE ABREVIATURAS

CPU	Central processing uni
ILP	Paralelismo em nível de instrução
ASIC	Application specific integrated circuit
FGPAS	Field programmable gate arrays
VHDL	Vhsic *hardware* description language
VHSIC	Very high speed integrated circuit
DOD	*Hardware* description language
RTL	*Register* transfer level
VGA	Video graphics array
BCD	Binary code decimal
HDL	*Hardware* description language
IHM	Interface homem-máquina

SUMÁRIO

AGRADECIMENTOS CARLA 3

AGRADECIMENTOS KARLYANNE 4

RESUMO 6

ABSTRACT 7

LISTA DE FIGURAS 8

LISTA DE QUADROS 12

LISTA DE TABELAS 15

LISTA DE ABREVIATURAS 17

SUMÁRIO 18

1. INTRODUÇÃO 22

1.1 MOTIVAÇÃO E JUSTIFICATIVA 25

1.2 OBJETIVOS 25

1.2.1 Objetivo Geral: 25

1.2.2 Objetivos Específicos: 25

1.3 ORGANIZAÇÃO DO TRABALHO 26

2. LINGUAGEM VHDL 27

2.1 CARACTERISTICAS IMPORTANTES DO VHDL 28

2.2 CONCEITOS BÁSICOS DA LINGUAGEM VHDL 29

2.2.1 Declaração de *Package* ou *Library* 31

2.2.2 Declaração de Entidade 32

2.2.3 Declaração da Arquitetura 35

2.2.4 Descrição Estrutural 36

2.2.5 Modelo de Descrição Comportamental 39

2.2.6 Configuração 41

2.3 ELEMENTOS BÁSICOS DA SITAXE DE VHDL 43

2.3.1 Espaço em Branco 45
2.3.2 Comentários 45
2.3.3 Parênteses 46
2.4 COMANDOS IF, CASE E LOOP 46
2.4.1 Comando If Then 47
2.4.2 Comandos *If Then Else* 48
2.4.3. Comando *CASE*. 49
2.4.4. Comando *WHILE LOOP*. 50
2.4.5. Comando *FOR LOOP* 51
3. LABORATÓRIOS 52
3.1 LABORATORIO 1. 62
3.1.1 Parte I - Switches, Luzes 62
3.1.2 Parte II - Multiplexador 2-to-1 64
3.1.3 Parte III - Multiplexador 3-to-1 69
3.1.4 Parte IV - Conversor 2 bits para 7 segmentos 71
3.1.5 Parte V - Multiplexador com saída para display 76
3.2 LABORATORIO 2. 79
3.2.1 Parte I - Decodificador Binário Decimal 79
3.2.2 Parte II - Decodificador binário para decimal com 2 dígitos. 81
3.2.3 Parte III - Somador completo 84
3.2.4 Parte IV - Conversor BCD de 4 bits. 88
3.3 LABORATORIO 3. 90
3.3.1 Parte I - Latch RS 90
3.3.2 Parte II - Latch D. 92
3.3.3 Parte III - *Flip-flop*tipo D 94
3.3.4 Parte IV - Registrador paralelo de 4 bits com *flip-flop*tipo D 96
3.1.5 Parte V - Registrador de deslocamento de 4 bits com *flip-flop* tipo D. 98

3.4 LABORATORIO 4. 101
3.4.1 Parte I - Contador de 4 bits com Flip flop D 102
3.4.2 Parte II - Contador *Up* de 4 bits com *flip-flop* D alternativo 105
3.4.3 Parte III - Contador de N bits. 106
3.3.4 Parte IV - Circuito sequencial de forma automática 108
3.5 LABORATORIO 5. 109
3.5.1 Parte I – Padrão VGA 109
3.5.2 Parte II - VGA – Linhas cruzadas 110
3.5.3 Parte III - VGA – Quadrado 126
3.5.4 Parte IV- Jogo simples 129
4. APLICAÇÃO 136
4.1 DESENVOLVIMENTO DA APLICAÇÃO. 136
4.1.1 Módulo PCG 136
4.1.2 Módulo SYNC 139
4.1.3 Módulo VGA 155
5. CONCLUSÃO 157
REFERências 159
APÊNDICE A 161
A.1- CÓDIGOS USADOS NO LABORATÓRIO 1 161
A.2- CÓDIGOS USADOS NO LABORATÓRIO 2 165
A.3- CÓDIGOS USADOS NO LABORATÓRIO 3 167
A.4- CÓDIGOS USADOS NO LABORATÓRIO 4 172
A.5- CÓDIGOS USADOS NO LABORATÓRIO 5 175
APÊNDICE B 186
B.1- CÓDIGO MODULO PCG 186
B.2- CÓDIGO MODULO SYNC 187
B.3 - CÓDIGO MODULO VGA 195

B.4- PINOS PARA CONEXÃO COM A PLACA 196
APÊNDICE C 197
C.1- CONEXÃO ENTRE AS CHAVES DA PLACA DE0 E OS PINOS DO FPGA. 197
C.2- CONEXÃO ENTRE AS LEDS DA PLACA DE0 E OS PINOS DO FPGA. 197
C.3- CONEXÃO ENTRE BOTÕES DA PLACA DE0 E OS PINOS DO FPGA. 197
C.4- CONEXÃO ENTRE OS DISPLAYS DE 7 SEGMENTOS DA PLACA DE0 E OS PINOS DO FPGA. 197

1. INTRODUÇÃO

Durante seu desenvolvimento histórico o homem sempre teve necessidade de contar e calcular, e à medida que esses cálculos foram se tornando mais frequentes e trabalhosos, o ser humano passou a criar instrumentos para realizar esses cálculos. A história dos computadores começou no momento em que o homem sentiu a precisão de realizar cálculos complexos com maior precisão e rapidez. A ideia de que a informação podia ser codificada, para em seguida ser processada independentemente do sentido das mensagens, foi se formando aos poucos: estava aberto o caminho para que o cálculo se tornasse progressivamente uma questão de máquinas e não mais de instrumentos (BRETON, 1987).

Os computadores convencionais podem ser utilizados para uma ampla gama de aplicações, esses computadores baseiam-se no modelo idealizado por Von Neumann que seria a interpretação técnica mais bem aceita do conceito de máquina de Turing devido a sua capacidade de processamento aberta. Logo, as observações sobre o modelo de Von Neumann também se aplicariam em geral ao modelo de Turing.

O modelo de Von Neumann consiste em um programa que é armazenado na memória e executado sequencialmente segundo um controle exercido por um registrador denominado de contador de programa contido em uma CPU. Esse registrador recebe endereços sequenciais, ou resultados de instruções do *branch*[1], que redireciona o fluxo de controle de programa (TANENBAUM, 2013). Esta característica de busca de instruções na memória da mesma forma das buscas de dados, que podem estar armazenados no mesmo espaço e endereçamento das instruções ou em espaço de memória independente, foi o diferencial do modelo de Von Neumann em relação aos modelos anteriores.

Ultimamente pode-se observar um grande número de aplicações que exigem cada vez mais uma grande quantidade de processamento de dados como as aplicações de mapeamento genético, a computação gráfica, as previsões meteorológicas e até mesmo programas que exigem um grande número de variáveis de entrada, com isso a computação toma uma nova vertente não convencional com uma área da computação preocupada com a criação de condições para que processamentos dessa elevada carga computacional possam ser executados em intervalos de tempo factíveis, permitindo que programas que levariam anos para fornecerem resultados em uma máquina comum possam fazê-lo em bem menos tempo (horas ou dias).

---

[1] São as responsáveis por tratar das informações que estão sendo processadas, ou que serão processadas, ou que já foram processadas (que passam pelo pipeline).

Quando uma aplicação ultrapassa a capacidade dos computadores convencionais, recorre-se a diferentes abordagens destinadas a criar sistemas computacionais de alto desempenho. Mesmo esse modelo de Von Neumann sendo fundamentalmente baseado em execuções sequenciais, ele permite explorar a técnica da computação paralela que ao decompor um problema em tarefas menores, processa essas tarefas em paralelo, alcançando bons resultados. Em máquinas de núcleo simples, com um único contador de programa, o paralelismo pode ser explorado por meio de técnicas de ILP (*Instruction Level Parallelism*/ Paralelismo em Nível de Instrução). Em caso de multiprocessadores ou multicomputadores, por sua vez, podem ser definidas diferentes *threads* a serem executadas por cada núcleo de processamento, sendo que cada linha utiliza um valor próprio no contador de programa para controlar sua sequência de execução (LINHARES, 2015). Contudo a aplicação deve possuir estrutura compatível com o modelo. Para muitas aplicações da computação paralela oferece pouca aceleração para cada unidade de processamento adicional. Sistemas desse tipo sofrem também *overhead* (sobrecarga) de comunicação entre processadores.

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 (um ASIC - *Application Specific Integrated Circuit*) 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*.

No entanto os custos de projeto e implementação de tal sistema são muito elevados, apenas se justificando se a produção for de alto volume. Outra dificuldade nessa abordagem orientada a aplicação é o tempo de desenvolvimento longo enquanto o desempenho de computadores convencionais aumenta rapidamente. Devido a isso o *hardware* dedicado pode ficar obsoleto em pouco tempo. É de salientar também que as soluções baseadas em ASIC são completamente inflexíveis dado que sua funcionalidade não pode ser modificada após a fabricação.

Levando em consideração essas dificuldades de custo, e curta duração de vida útil desses *hardwares* dedicados, esses acabam não representando uma abordagem suficientemente atrativa. Contudo se for possível diminuir os custos e o tempo de desenvolvimento, essa técnica torna-se bastante viável.

A computação reconfigurável combina o desempenho do *hardware* dedicado (ASIC) com graus de flexibilidade alcançados nos *softwares.* Entretanto, enquanto que os componentes

de *software* estão limitados às arquiteturas dos microprocessadores, a computação reconfigurável permite ter arquiteturas adaptadas às aplicações. A computação reconfigurável consiste em um processamento computacional de alta velocidade e muito flexível, que possui como característica suas "auto- mudanças" de caminho de dados e o controle do fluxo.

Esse paradigma baseia-se em dispositivos lógicos reprogramáveis que podem atingir um desempenho elevado, e ao mesmo tempo, fornecer flexibilidade da programação de portas lógicas. Um dispositivo de *hardware* típico utilizado em computação reconfigurável são os FGPAs (*Field-Programmable Gate Arrays*), cuja configuração é realizada pelo projetista. A prototipação rápida dos FGPAs permite que um projeto seja testado em um *hardware* real antes da produção final em um circuito ASIC, assim, os erros podem ser corrigidos. Isso permite uma redução significativa do *time-to-market*[2] (REDAELLI; SANTAMBROGI; SCIUTO, 2008).

A configuração FPGA é geralmente especificada usando uma linguagem de descrição de *hardware* (HDL), O projetista cria um arquivo de texto, seguindo certo conjunto de regras, conhecido como sintaxe da linguagem, e usa um compilador para criar dados de programação do dispositivo lógico programável (ou PLD). Esta descrição de *hardware* pode ser usada para gerar projetos hierárquicos, ou seja, um componente definido em uma descrição pode ser usado para gerar um *hardware* específico ou ser usado como parte de outro projeto.

As HDLs têm uma grande semelhança às linguagens de programação, mas são especificamente orientadas à descrição das estruturas e do comportamento do *hardware*. As mais utilizadas nos dias de hoje são a linguagem VERILOG e a VHDL.

Atualmente a velocidade e os recursos disponíveis em FGPAs são comparáveis com os dos ASICs, com a vantagem de flexibilidade inerente às aplicações de *software*s (IOULIIA SKLIAROVA, 2003).

Segundo (ATHANAS; SILVERMAN, 1993) a principal característica da computação reconfigurável é a presença de um *hardware* que pode ser reconfigurado para implementar uma funcionalidade específica mais apropriada e sob medida. Unindo microprocessadores e *hardwares* programáveis com finalidade de combinar o potencial do *hardware* e do *software* a ser utilizado em aplicações que vão desde sistemas embarcados a sistemas computacionais de alta performance.

Com o aumento da importância da desse novo modelo computacional, cresce cada vez mais a necessidade de introduzir uma disciplina específica de computação reconfigurável nos

---

[2] O tempo requerido para desenvolver um sistema até um ponto em que ele possa ser disponibilizado para o usuário.

currículos acadêmico (HARTENSTEIN, 2006). Interessante também adicionar tópicos de conceitos de computação reconfigurável no ensino de arquitetura de computadores como uma preparação ao ensino desse modelo, assim como a disciplina de introdução à programação precisará, introduzir conceitos de programação paralela devido à difusão dos modernos processadores *multicore*.

Esse trabalho visa à criação de um material inicial para o ensino acadêmico da computação reconfigurável através da linguagem VHDL com intuito de oferecer aos alunos de graduação um conhecimento básico na prática desse modelo computacional, e despertar o interesse desses alunos aos modelos de computação não convencionais.

1.1 MOTIVAÇÃO E JUSTIFICATIVA

Vendo a necessidade por parte dos acadêmicos de computação em aprenderem os novos modelos computacionais como a computação reconfigurável e a escassez de matérias de ensino tanto para profissionais ministrarem aulas como para alunos estudarem a cerca do assunto, surgiu a proposta desse trabalho que consiste em desenvolver um material teórico e prático de ensino da linguagem VHDL, para uma disciplina introdutora á computação reconfigurável. Mostrando a alunos e professores interessados nesse novo modelo computacional por onde começar a desenvolver algo nessa área que está em grande expansão na computação moderna.

A dificuldade de materiais disponíveis para facilitar o aprendizado infelizmente é um fator que afasta muitos acadêmicos de diversos assuntos computacionais pelo os quais possam se interessar durante sua vida acadêmica e profissional.

Tendo em vista a grande importância da computação reconfigurável e a grande escassez de matérias relacionados ao aprendizado da mesma, esse trabalho se justifica em facilitar o ensino e aprendizagem desse paradigma através da linguagem VHDL.

1.2 OBJETIVOS

1.2.1 Objetivo Geral:

O objetivo geral do trabalho é a criação de um material didático e de fácil compreensão da computação reconfigurável, por meio do ensino teórico e prático da linguagem VHDL.

1.2.2 Objetivos Específicos:

- Desenvolver de material de serventia para alunos e professores interessados na computação reconfigurável.

- Desenvolver ao final do material uma pequena aplicação utilizando a linguagem VHDL.
- Disponibilizar esse material para acadêmicos e docentes interessados no aprendizado e ensino da computação reconfigurável.

1.3 ORGANIZAÇÃO DO TRABALHO

O trabalho segue organizado da seguinte forma: o capítulo faz uma abordagem sobre a linguagem VHDL, com suas características e seus elementos básicos de Sintaxe. O terceiro capítulo oferece cinco laboratórios para que seja posto em prática os ensinamentos da linguagem VHDL, sendo que cada laboratório aborda uma temática e contém tarefas para fixação. O quarto capítulo descreve uma aplicação mais ampla que consiste em uma variação de um jogo tradicional da década de setenta conhecido como *Pong*. Por fim o quinto capítulo exibe as conclusões e sugestões para trabalhos futuros.

2. LINGUAGEM VHDL

As primeiras linguagens de descrição de *hardware* foram desenvolvidas no final da década de 60, como alternativa às linguagens de programação para descrever e simular dispositivos *hardwares*. Durante dez anos, inúmeras linguagens foram desenvolvidas com sintaxe e semânticas incompatíveis, permitindo descrições a diferentes níveis de modelização (DÉHARBE, 1998).

No fim dos anos 70, o Departamento de Defesa dos Estados Unidos (DoD/ Department of Defense) definiu um programa chamado VHSIC (*Very High Speed Integrated Circuit*) que visava à descrição técnica e projeto de uma nova linha de circuitos integrados. Com o avanço acelerado dos dispositivos eletrônicos, entretanto este programa apresentou-se ineficiente, principalmente na representação de grandes e complexos projetos.

O projeto VHSIC era de alta prioridade militar e havia dezenas de fornecedores envolvidos, o DoD() estava preocupado principalmente com as questões de portabilidade, documentação e compreensibilidade dos projetos. Cada um destes fornecedores atuava desenvolvendo partes dos projetos ou mesmo fornecendo componentes que viriam a se encaixar em outros sistemas maiores. Desta forma o DoD optou por buscar desenvolver uma linguagem que servisse como base para troca de informações sobre estes componentes e projetos. Uma linguagem que, independente do formato original do circuito, pudesse servir como uma descrição e documentação eficientes do circuito, possibilitando os mais diferentes fornecedores e participantes a entender o funcionamento das outras partes, padronizando a comunicação (SOUZA, 2008).

Em 1981, aprimorando-se as ideias do VHSIC, foi proposta uma linguagem de descrição de *hardware* mais genérica e flexível, a linguagem chamada VHDL (VHSIC *Hardware Description Language*). A linguagem VHDL que foi originalmente desenvolvida sob o comando do DoD para documentar o comportamento de ASICs que compunham os equipamentos vendidos às Forças Armadas americanas. Isto quer dizer que a linguagem VHDL foi desenvolvida para substituir os complexos manuais que descreviam o funcionamento dos ASICs (D'AMORE, 2012). Até aquele momento, a única metodologia largamente utilizada no projeto de circuitos era a criação através de diagramas esquemáticos. O problema com esta metodologia é o fato de que desenho tem menor portabilidade, são mais complexos para compreensão e são extremamente dependentes da ferramenta utilizada para produzi-los.

O desenvolvimento da VHDL serviu inicialmente aos propósitos de documentação do projeto VHSIC. Entretanto, naquela época buscava-se uma linguagem que facilitasse o projeto

de um circuito, ou seja, a partir de uma descrição textual, um algoritmo, desenvolver o circuito, sem necessidade de especificar explicitamente as ligações entre componentes. A VHDL presta-se adequadamente a tais propósitos, podendo ser utilizada para as tarefas de documentação, descrição, síntese, simulação, teste, verificação formal e ainda compilação de *software*, em alguns casos (PERRY, 2002).

Diferente das linguagens de programação convencionais que são baseadas execução sequencial das instruções, a linguagem VHDL opera baseada em execução paralela podendo descrever ou modelar as operações em *Hardware* Digital, já que esses dispositivos também operam em paralelo. Lembrando também que em VHDL, as variáveis mudam sem atraso e os sinais mudam com um pequeno atraso.

2.1 CARACTERISTICAS IMPORTANTES DO VHDL

Nesta sessão serão mostradas principais caraterísticas do VHDL, para a melhor compreensão da linguagem, dentre elas estão:

- Suporta projetos com múltiplos níveis de hierarquias. Portanto, a descrição geral do circuito pode consistir na interligação de outras descrições menores.
- Favorece projeto "*top-down*", onde projetos complexos partem de um nível de especificação mais elevado para um mais baixo.
- Permite, através de simulação, verificar o comportamento do sistema digital;
- Permite descrever *hardware* em diversos níveis de abstração, por exemplo:
  - Algorítmico ou comportamental;
  - Transferência entre registradores (RTL);
  - Nível de portas lógicas (*Gate Level*);
- Pode mesclar diferentes níveis de abstração em um mesmo código.
- Comandos executados concorrentemente (com exceção de regiões específicas no código), assim como os elementos de um sistema digital executam tarefas simultaneamente:
- Ordem dos comandos é irrelevante;
- Mudança de valor em um sinal acarreta a execução de todos os comandos envolvidos;
- Possibilita delimitar regiões de código sequencial (subprogramas e processos) onde a execução dos comandos segue a ordem de sua apresentação no código.
- Possibilita a definição de Biblioteca e Pacote ("*Library*" e "*Package*", respectivamente).

- Pacotes(“*Package*”,): armazenam subprogramas, constantes ou novos tipos definidos, evitando a repetição de uma definição em todas as descrições.
- Bibliotecas (“*Library*”): armazenam informações compiladas, sendo a biblioteca corrente denominada “*Work*”.

2.2 CONCEITOS BÁSICOS DA LINGUAGEM VHDL

Devido ao seu potencial, a linguagem VHDL é complexa, e por vezes e difícil entendimento, devido às inúmeras opções para modelar o comportamento de um circuito. Contudo, o entendimento de uma pequena quantidade de comandos, suficiente para a modelagem de estruturas medianamente complexas, pode ser rapidamente atingido. A necessidade de projetos mais complexos encaminha à procura por novos comandos levando a uma maior compreensão das opções da linguagem.

Considerando o bloco de função lógica da Figura 2.1, ele é composto de portas de entradas as quais são as variáveis lógicas I0, I1 e I2 e de duas portas de saída para a função f(I0, I1, I2) que são S1 e S2. Uma Linguagem de Descrição de *Hardware*, como a VHDL, possibilita que a operação lógica que descreve internamente o bloco, por exemplo, da Figura 2.1, seja descrita usando enunciados bem definidos tal como uma linguagem de programação de alto nível para computadores.

Figura 2.1 *Modulo lógico genérico.*

Fonte: *D’Amore(2012)* .

Para utilizar a linguagem VHDL, inicialmente faz-se a descrição da unidade lógica usando a sintaxe e formato específicos da linguagem. Essas informações são as entradas de um arquivo de texto que será utilizado como entrada de um compilador VHDL. O arquivo de saída é então utilizado na etapa de simulação gráfica, para verificação de resultados. Estando o projeto funcionando como esperado, é só programar o chip com o projeto (PERRY, 2002).

A estrutura de um programa em VHDL baseia-se em quatro blocos:

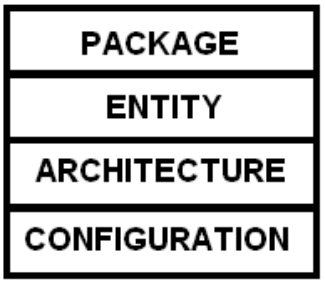


Figura 2.2 *Estrutura de um programa em VHDL .*

Fonte: *D'Amore(2012) .*

- *PACKAGE* ou *LIBRARY*: conjunto de subprogramas que descrevem elementos e componentes já programados para serem reutilizados.
- *ENTITY* (Entidade): define as portas de entradas e saídas dos circuitos na descrição.
- *ARCHITECTURE* (Arquitetura): implementações do projeto, descreve as relações entre as portas.
- *CONFIGURATION* (Configuração): define as arquiteturas que serão utilizadas.

A estrutura de uma descrição VHDL com os seus blocos definidos é mostrado a seguir:

Tabela 2.1 *A estrutura de uma descrição VHDL com os seus blocos .*

LIBRARY IEEE; ***USE IEEE.STD_LOGIC_1164.all;*** ***USE IEEE.STD_LOGIC_UNSIGNED.all;***	PACKAGE (BIBLIOTECAS)
ENTITY *exemplo* ***IS*** *PORT (* *<descrição dos pinos de I/O>* *);* ***END*** *exemplo;*	ENTITY (PINOS DE I/O)
ARCHITECTURE *teste* ***OF*** *exemplo* ***IS*** ***BEGIN*** *.....* ***END*** *teste;*	ARCHITECTURE (ARQUITETURA)

A descrição em VHDL de qualquer módulo lógico é representada pela **Entidade de Projeto,** figura 2.3 a qual deve ser composta de ao menos duas das estruturas, que consistem na declaração da **entidade (“*entity*”)** e na **arquitetura (“*architecture*”).** Para o módulo da Figura 2.1.

- *ENTITY*: No caso da entidade de Projeto da Figura 2.1 seria:
  I0, I1, I2, S0 e S1
- *ARCHITECTURE*: No caso da entidade de Projeto da Figura 2.1 seria o circuito interno ao bloco referente ao módulo.

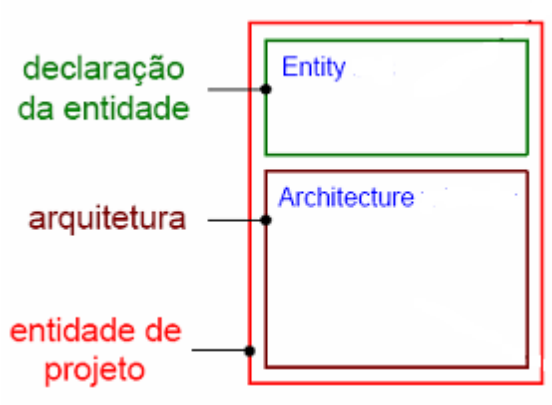


Figura 2.3 *Entidade de Projeto da Linguagem VHDL.*

Fonte: *D'Amore(2012) .*

2.2.1 Declaração de *Package* ou *Library*

O *PACKAGE* ou *LIBRARY* são conjuntos de declarações usadas para um determinado fim. Pode ser um conjunto de subprogramas (contendo bibliotecas ou funções) que operem com um tipo de dado assim como pode ser um conjunto de declarações necessárias para modelar um determinado projeto. Pacotes mais utilizados estão citados na Tabela 2.1.

Tabela 2.2 *Pacotes mais utilizados.*

Biblioteca	**Pacote**	**Tipos de dados**	**Descrição**
IEEE	STD_LOGIC_1164	STD_LOGIC e STD_LOGIC_VECTOR	Define o padrão para descrever a interconexão entre tipos de dados usados na linguagem VHDL
IEEE	STD_LOGIC_ARITH	Especifica tipos de dados sinalizados e não sinalizados	Funções de conversão, operações aritméticas e comparações para uso de dados sinalizados e não sinalizados
IEEE	STD_LOGIC_UNSIGNED	STD_LOGIC_VECTOR	Define funções que permitem usar tipos de dados STD_LOGIC_VECTOR, como se fossem tipo de dado não sinalizado
IEEE	STD_LOGIC_SIGNED	STD_LOGIC_VECTOR	Define funções que permitem usar tipos de dados STD_LOGIC_VECTOR, como se fossem tipo de dado sinalizado
IEEE	NUMERIC_STD		Define operações aritméticas seguindo o padrão IEEE.

			Substitui os pacotes usados juntos STD_LOGIC_ARITH, STD_LOGIC_UNSIGNED e STD_LOGIC_SIGNED
STD	STANDARD	BIT(0 ou 1) BIT_VECTOR(3 Downto 0); BIT_VECTOR(0 to 3); BOOLEAN(falso ou verdadeiro); INTEIRO(positivo ou negativo)	
STD	TEXTIO		Define os arquivos de operações
WORK	<definido pelo usuário>		Biblioteca corrente de Trabalho

OBS: As bibliotecas STD e *WORK* não necessitam ser referenciada no projeto VHDL, pois são assumidas automaticamente pela linguagem.

A declaração do uso de pacotes e bibliotecas é mostrada no quadro 2.1.

Quadro 2.1 *Declaração do uso de pacotes e bibliotecas* .

```
LIBRARY < nome_da biblioteca>
        USE < nome_do_pacote_biblioteca> .ALL
```

OBS: O uso de .all implica que todos os elementos da biblioteca podem ser usados. Sempre que for utilizar alguma biblioteca sempre user .ALL

2.2.2 Declaração de Entidade

Todo componente VHDL tem que ser definido como uma entidade ("*entity"*), o que nada mais é do que uma representação formal de uma simples porta lógica até um sistema lógico completo. Na declaração de uma entidade, descreve-se o conjunto de entradas e saídas que constituem o projeto, é equivalente ao símbolo de um bloco em captura esquemática, como está esquematizado na Figura 2.4.

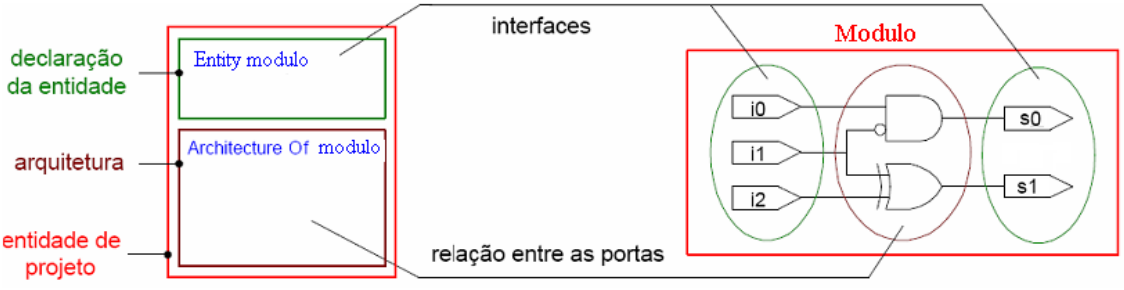


Figura 2.4 *Exemplo de entidade de Projeto do bloco Modulo.*

Fonte: *D'Amore(2012)* .

Para o módulo lógico da Figura 2.4, a declaração da entidade é que vai declarar que este módulo contém I0, I1 e I3 como entradas, e s1 e s2 como saída. Uma declaração genérica para o módulo da Figura 2.4 é mostrada no Quadro 2.2.

Quadro 2.2 *Uma declaração genérica para o módulo da Figura 2.1.*

```
ENTITY modulo IS
PORT (I0,I1, I2:modo_1 tipo_a; --entradas
      S0,S1 :modo_2 tipo_b); --saída
END modulo;
```

A declaração de uma entidade deve conter três cláusulas como explicado a seguir:

ENTITY: palavra reservada "*ENTITY*" inicia a declaração seguida do nome que a identifica e da palavra reservada IS.

```
ENTITY modulo IS
```

PORT: lista as entradas e as saídas da entidade de projeto. Define o modo e tipo das portas.

Modos de operação de uma *PORT*: existem quatro modos, representados na Figura 2.5, os quais são:

- Modo ***IN*** : entrada.
- Modo ***OUT***: saída.
- Modo ***BUFFER***: saída que pode realimentar o circuito.
- Modo ***INOUT***: bidirecional.

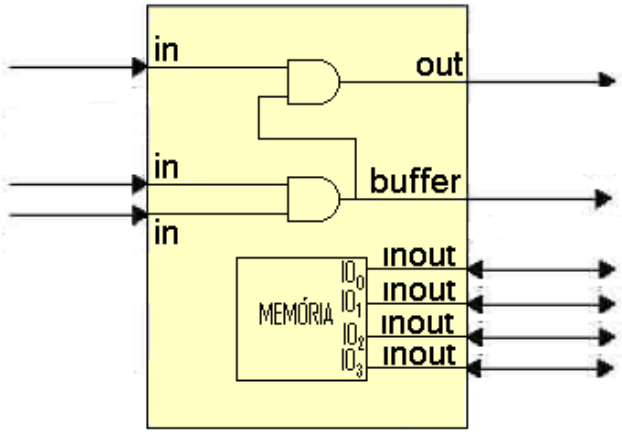


Figura 2.5 *Modos de operação de uma PORT.*

Fonte: *D'Amore(2012)* .

A declaração inicia com a palavra reservada *PORT* e em seguida abre-se um parêntesis "(" seguido do nome das entradas separado por vírgula "," seguido de dois pontos ":" do modo, por exemplo: ***IN*** e do tipo das entradas listadas na linha e de ponto e vírgula(;), quando trata-se de entradas. A última declaração da porta é finalizada com o parêntesis antes do ponto e vírgula (;).

Tipos de uma *PORT*: Serão explicados em um item posterior. Mas os tipos de *PORT* mais utilizados estão descritos na Tabela 2.3 a seguir.

Tabela 2.3 *Tipos de PORT mais utilizados.*

Bit	Assume valores '0' ou '1' **x: in bit;**
bit_vector	Vetor de bits **x: in bit_vector(7 downto 0);** **x: in bit_vector(0 to 7);**
std_logic	**x: in std_logic;**
std_logic_vector	**x: in std_logic_vector(7 downto 0);** **x: in std_logic_vector(0 to 7);**
Boolean	Assume valores TRUE ou FALSE
Real	

END: termina a declaração. Usa-se a palavra reservada *END* seguida do nome da Entidade de Projeto e de ponto e vírgula (;).

Uma declaração genérica de uma entidade é mostrada a seguir:

Quadro 2.3 *Declaração genérica de uma entidade.*

```
ENTITY nome IS
PORT ( I0, I1,...In : IN tipo_a; --entradas
        S0, S1 : OUT tipo_b ); -- saídas
        Y0 : BUFFER tipo_c); -- saída
        Z0, Z1 : INOUT tipo_d ); -- entrada/saída
END nome;
```

Normalmente as portas de entrada de uma entidade são declaradas antes das de saída. Cada declaração de interface é seguida por ponto e vírgula ';'. Também é necessário colocar-se ponto e vírgula no final da definição de porta.

2.2.3 Declaração da Arquitetura

A declaração da arquitetura ("*architecture*") descreve o comportamento da entidade, define o seu funcionamento interno, isto é, como as entradas e saídas influem no funcionamento e como se relacionam com outros sinais internos. Para tal, utiliza-se uma série de comandos de operação. A declaração de uma arquitetura pode conter comandos concorrentes ou sequenciais. Sua organização pode conter declaração de sinais, constante, componentes, operadores lógicos, etc, assim como comandos (ex: *BEGIN*, *END*). A linguagem VHDL permite ter mais de uma *architecture* para a mesma entidade. Uma Arquitetura consiste de duas partes:

- A seção de declaração da arquitetura.
- O corpo da arquitetura.

Um exemplo de declaração de arquitetura é mostrado:

Quadro 2.4 *Exemplo de declaração de arquitetura.*

```
ARCHITECTURE behavioral OF ent IS
SIGNAL c_internal: small INT;
BEGIN
    c_internal<= a0 + b0;
   c0<= c_internal + a1 + b1;
END behavioral;
```

A seção de declaração da arquitetura ("*architecture*") é a área entre a chave *architecture* e a chave *begin*. Nesse espaço podem-se declarar objetos que são localizados na arquitetura.

Após a seção de declaração tem-se o corpo da arquitetura o qual especifica o comportamento da arquitetura. A arquitetura de uma entidade pode ser descrita de três formas distintas de abstração, mas que, em geral, conduzem a uma mesma implementação.

- Descrição estrutural
- Descrição comportamental
- Descrição física

2.2.4 Descrição Estrutural

Descreve todos os componentes e suas interconexões onde as atribuições de sinais são feitas através do mapeamento de entradas e saídas de componentes. Ou seja, é como se fosse uma lista de ligações entre componentes básicos pré-definidos, onde:

- *Component*: é exatamente a descrição de um componente
- *port map*: é um mapeamento deste componente em um sistema maior.

Exemplo 1: Um exemplo de arquitetura estrutural pode ser visto tomando o circuito da Figura 2.6.

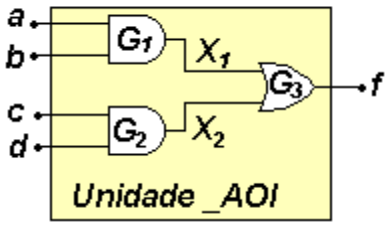


Figura 2.6 *Exemplo de descrição estrutural.*

Fonte: *D'Amore(2012).*

As três equações para X1, X2 e f fornecem a estrutura básica do circuito lógico, pois fornecem a informação necessária para reconstruir o diagrama exatamente como ele for originalmente apresentado. Modelos estruturais são escritos usando o mesmo conceito.

No caso do exemplo da Figura 2.6, os sinais X1 e X2 são sinais internos, portanto não foram declarados na *"ENTITY",* então devem ser declarados no corpo da arquitetura antes entre as declarações "*ARCHITECTURE*" e "*BEGIN*". Modelos estruturais em VHDL são criados utilizando blocos construtivos chamados de componentes. Um componente é um módulo lógico definido em uma listagem normal de uma entidade com uma arquitetura, sendo utilizado para criar outros módulos lógicos. Um componente pode ser visto como uma simples porta lógica, ou representar um bloco mais complicado. Um componente será analisado como um simples bloco construtivo, independente da sua complexidade interna. Para criar um modelo estrutural, mostrado no quadro a seguir do circuito da Figura 2.6, primeiro se escolhe os componentes e

depois se descreve como eles são interconectados. A conexão é especificada por uma listagem formal que mapeia as portas dos módulos. O mapeamento de portas (*port map*) especifica que portas da entidade são conectadas para quais sinais do sistema que se está montando.

Quadro 2.5 *Descrição estrutural da Unidade_AOI.*

```
ENTITY Unidade_AOI IS
        Port(a,b,c,d :IN bit;
             f: OUT bit);
END  Unidade_AOI;
--A segunda listagem constrói a unidade usando componentes
ARCHITECTURE Estrutural of Unidade_AOI IS
--Definir a porta AND como um componente
COMPONENTE AND2
        PORT (x,y : IN bit;
                z: OUT bit);
END COMPONENTE;
-- Define a porta OR como um componente
COMPONENT OR2
        PORT(x, y : IN bit;
        z : OUT bit);
END COMPONENT;
--A próxima linha declara os seguintes sinais internos ao módulo
SIGNAL X1, X2: bit;
--O mapeamento das portas especifica a conexão interna
BEGIN
        G1: AND2 PORT MAP(a,b, X1);
        G2: AND2 PORT MAP(c,d, X2);
        G3 : OR2 PORT MAP(X1, X2, f);
END ESTRUTURAL;
```

Um dos conceitos novos apresentados no Quadro 2.5 foi à declaração de um componente a qual genericamente é mostrada no Quadro 2.6 abaixo:

Quadro 2.6 *Declaração de componente .*

```
COMPONENT AND2
        PORT(x, y : IN bit;
                z : OUT bit);
END COMPONENT;
```

Para utilizar uma unidade como se fosse um componente, um módulo mais complexo, ele deve ser previamente definido como uma entidade que tem uma determinada arquitetura. Isto significa que já haviam sido feitas na listagem completa do programa em VHDL, as seguintes declarações:

Quadro 2.7 *Declaração de componente AND2 .*

```
ENTITY AND2 IS
        PORT(u, v, : IN bit;
             q : OUT bit);
END AND2;
ARCHITECTURE Logica OF AND2 IS
BEGIN
        q <= u AND v;
END Logica;
```

Quadro 2.8 *Declaração de componente OR2.*

```
ENTITY OR2 IS
        PORT(u, v, : IN bit;
             q : OUT bit);
END OR2;
ARCHITECTURE Logica of OR2 IS
BEGIN
        q <= u or v;
END Logica;
```

O conceito de componentes pode ser entendido usando o conceito de biblioteca do projeto, que é uma coleção de diferentes módulos, cada um definido por uma declaração de entidade e arquitetura. Uma vez que as células tenham sido definidas na biblioteca, podem-se usar cópias das células no projeto em questão usando o comando *component*. Isto é chamado de instanciação da célula e o componente é chamado de instância do original. Instanciar uma célula significa copiar a célula que está na biblioteca. Uma célula pode ser instanciada quantas vezes forem necessárias.

Na linha ”*signal: X1, X2 : bit;*” do quadro 2.5,

A palavra *SIGNAL* é utilizada para definir os identificadores (variáveis) internos X1 e X2 que definem as saídas das portas G1 e G2 do diagrama lógico original. A definição de sinal é diferente da definição de identificadores utilizados na declaração de uma porta (*port*) que existe dentro do módulo. Identificadores internos, são, portanto utilizados para conectar portas , formando as conexões do módulo.

Depois dos componentes declarados, o circuito é descrito com a utilização das palavras chave *port map*, as quais fornecem a informação de como os sinais (ligações internas) dos componentes interagem.

A linha “*G1:AND2 port map (a, b, X1);” do quadro 2.5*

É o primeiro mapeamento e define que a porta G1 é um componente AND2 e que está conectado aos identificadores de porta a, b e ao sinal X1. A ordem dos sinais na declaração do

componente AND2 foi dada como (u, v, q) que corresponde a primeira entrada, segunda entrada e saída, respectivamente. O mapeamento da porta (a, b, X1) mantém essa mesma ordem. Portanto, a variável de entrada a da Unidade_AOI é mapeada na variável de entrada u da entidade AND2; a variável de entrada b da Unidade_AOI é mapeada na variável de entrada v da entidade AND2; a variável de entrada X1 da Unidade_AOI é mapeada na variável de entrada q da entidade AND2; O mesmo raciocínio serve para as outras linhas de mapeamento.

2.2.5 Descrição Comportamental

A Linguagem de descrição de *hardware* VHDL permite a utilização de comandos condicionais para construir uma declaração de arquitetura baseada no comportamento do módulo. Esse tipo de descrição, representado no Quadro 2.9, é utilizado na descrição de sistemas sequenciais cujo comando fundamental é o "*PROCESS*" o qual pode ser, opcionalmente precedido de um rótulo ("*label*") e seguido de uma lista de sensibilidade que indica quais são as variáveis e sinais cuja alteração deve levar à reavaliação da saída. No simulador funcional, quando uma variável da lista é modificada, o processo é simulado novamente. Basicamente, a diferença entre arquitetura com descrição comportamental e por fluxo de dados está no uso da declaração *PROCESS* que define os processos concorrentes.

Obs: Sinais com inicialização assíncrona devem obrigatoriamente estar na lista de sensibilidade. Caso não conste nesta lista, a ferramenta de síntese irá colocar uma mensagem alertando a ausência, mas irá construir o circuito, podendo este não corresponder com o que se deseja implementar. Quando a operação do sinal é síncrona, não há necessidade de incluir o sinal na lista de sensibilidade.

Quadro 2.9 *Descrição de sistemas sequenciais cujo comando fundamental é o "PROCESS".*

```
process_name: PROCESS( sensitivity_list_signal_1, ... )
BEGIN
-- comandos do processo
END PROCESS process_name;
```

Esta é a forma mais flexível e poderosa de descrição. O corpo da arquitetura comportamental é montado descrevendo sua função a partir de uma ou várias sentenças de processo (*PROCESS*), que são conjuntos de ações a serem executadas. Os tipos de ações que podem ser realizadas incluem expressões de avaliação, atribuição de valores a variáveis, execução condicional, expressões repetitivas e chamadas de subprogramas. Dentro dos processos os comandos são executados sequencialmente, mas se em uma arquitetura tiver mais de um processo eles serão executados concorrentemente entre si.

2.2.6 Descrição Física

Na representação física o sistema é representado do ponto de vista físico, onde estão os pinos, onde estão os componentes no *chip*, quais as larguras dos *pitches*, tamanho do componente. A visão física do sistema provê a informação mais detalhada. É a especificação final para a fabricação do *hardware*.

No exemplo de descrição física utiliza-se o circuito somador de 1 bit e estender para um circuito somador de 4 bits, tendo como entrada as chaves da placa DE0 e como saída os LEDs desta mesma placa.

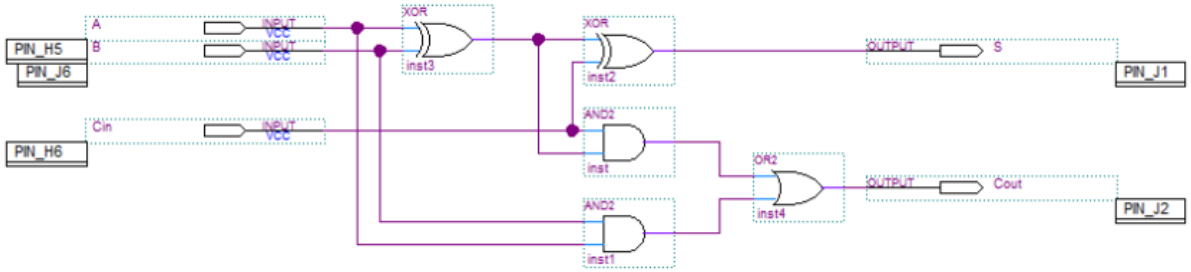


Figura 2.7 *Circuito combinacional somador de 1 bit.*

Observe que no circuito acima a entrada nomeada como a letra A está conectada ao pino PIN_H5, a entrada B está conectada ao pino PIN_J6, a entrada Cin está conectada ao pino PIN_H6, a saída S está conectada ao pino PIN_J1 e a saída Cout está conectada ao pino PIN_J2. Os pinos H6, H5 e J6 correspondem as chaves SW2, SW1 e SW0 e os pinos J2 e J1 correspondem aos LEDs LEDG[1] e LEDG[0], respectivamente.

Utilizando o bloco somador de 1 bit desenvolvido anteriormente e empregando boas práticas de desenvolvimento de circuitos lógicos, simplificando o circuito final e acelerando o seu desenvolvimento. Com isto, o circuito lógico combinacional que representa um somador completo de 4 bits, é mostrado na figura 2.8.

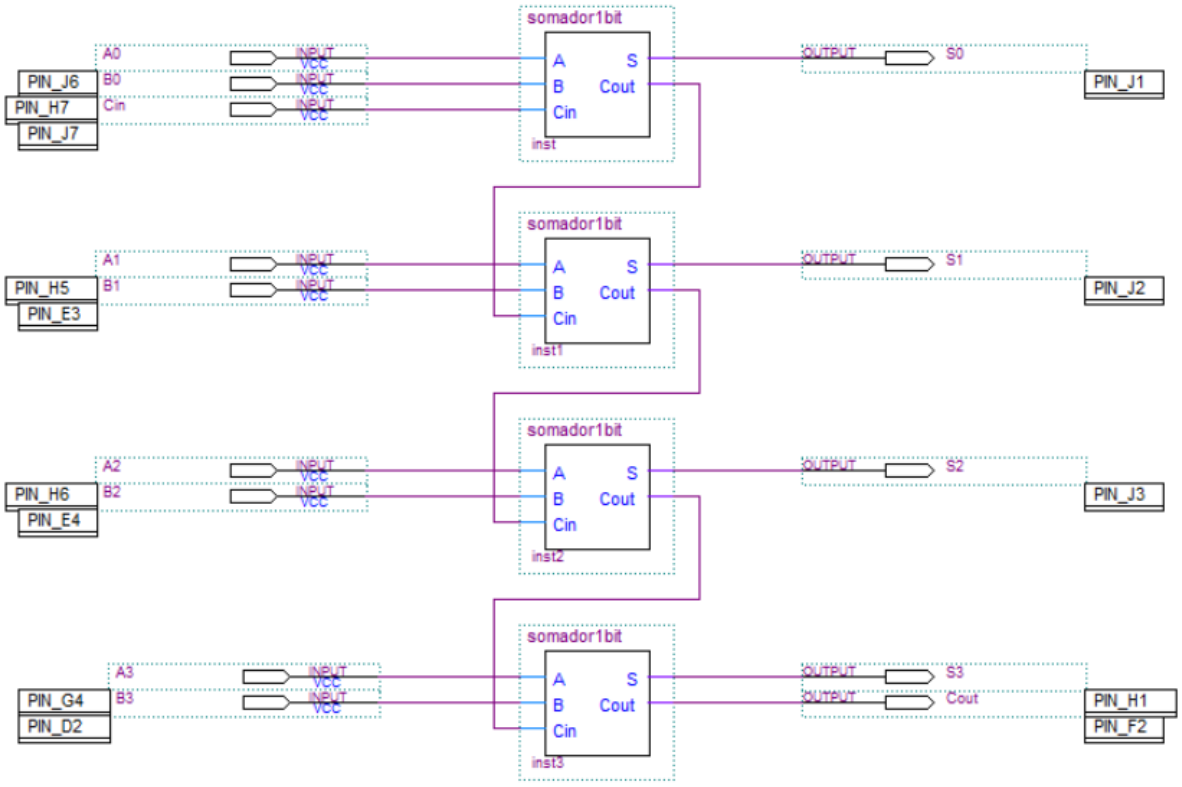


Figura 2.8 *Circuito combinacional somador de 4 bit.*

Para testar este circuito, será conectado os números binários A3A2A1A0 e B3B2B1B0 aos pinos G4, H6, H5, J6, D2, E4, E3 e H7 que representam as chaves SW3, SW2, SW1, SW0, SW9, SW8, SW7 e SW6, respectivamente. A entrada Cin deverá ser conectada ao pino J7, SW5. As saídas S3S2S1S0 são conectadas aos pinos H1, J3, j2 e J1 e a saída Cout ao pino F2, que representam os LEDs LEDG3, LEDG2, LEDG1, LEDG0 e LEDG4.

2.2.7 Configuração

A configuração estabelece o elo entre uma declaração de componente e uma entidade de projeto, ela define as arquiteturas que serão utilizadas. Uma mesma entidade pode ter várias arquiteturas. Para ligar a arquitetura à entidade é usada a seguinte declaração:

Quadro 2.10 *Ligação da arquitetura á entidade.*

For lista_de_rótulos: nome_componente (Especificação do componente) *USE ENTITY* *WORK.nome_entidade(arquitetura)* (indicação de elo)

Como por exemplo, o quadro 2.11 abaixo mostra a declaração da entidade ”soma” com suas três arquiteturas que são utilizadas na forma de componentes. Na entidade “somadores1”

são declarados dois componentes “add” e “adx”. Na declaração do componente “add” o nome das portas coincidem com o das portas da entidade “soma”.

A especificação da configuração é definida nas linhas onde iniciam com declarações ***“FOR u1”, “FOR u2” e “FOR u3”.*** Na solicitação “u1” que emprega a declaração do componente “add” é estabelecido um elo com “soma”. Como não foi especificada a arquitetura, é empregada a última arquitetura compilada. A solicitação “u2” emprega também a declaração do componente “add”, para essa solicitação é estabelecido o elo com a arquitetura “lenta” da entidade “soma”. A solicitação “u3” emprega a declaração do componente “adx”, e para essa solicitação é estabelecido o elo com arquitetura “típica” de “soma”. Como nesse último caso os nomes das portas são diferentes, é necessário definir a correspondência entre nomes através de “*PORT MAP*”.

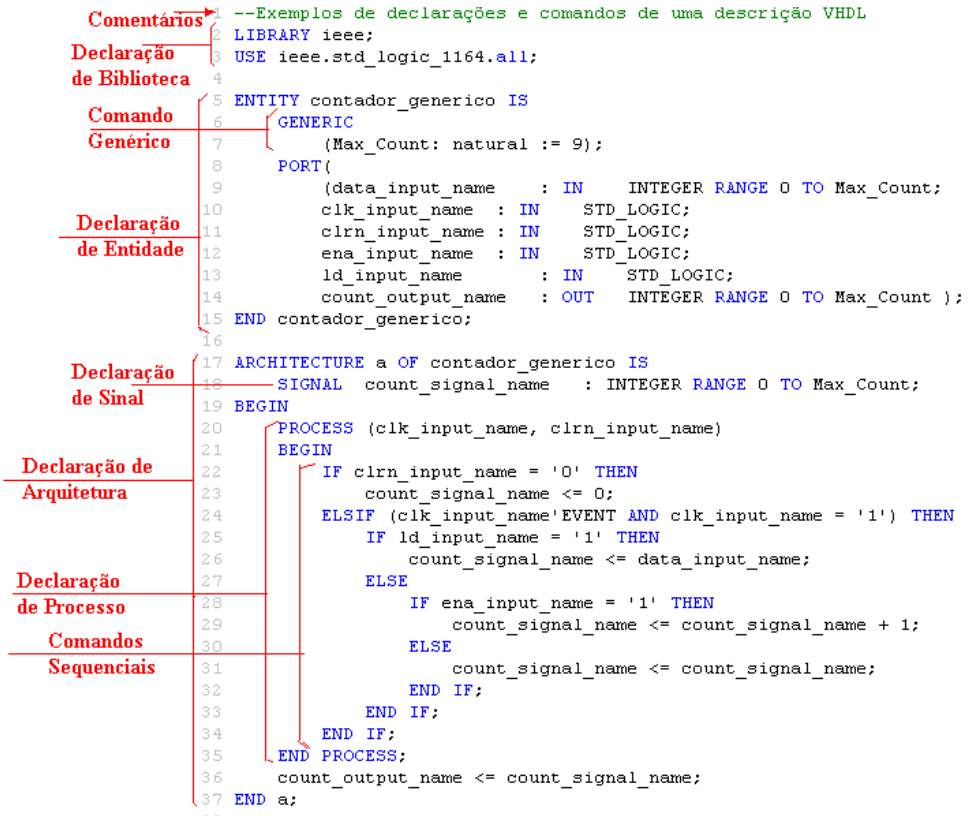
Quadro 2.11 *Exemplo retirado do livro d"Amore.*

```
ENTITY soma IS
        PORT (a, b : IN INTEGER 0 TO 7;
                s : OUT INTEGER 0 TO 15);
END soma;

ACHITECTURE lenta OF soma IS
BEGIN
        s<= a + b AFTER 50ns;
END lenta;

ACHITECTURE típica OF soma IS
BEGIN
         s<= a + b AFTER 25ns
END típica;

ACHITECTURE rapida OF soma IS
BEGIN
        s<= a + b AFTER 10ns;
END rapida;

-- Especificação da configuração
ENTITY somadores1 IS
        PORT (x,y : IN INTEGER 0 TO 7;
                sl, st, sr : OUT INTEGER 0 TO 15);
END somadores1;

ACHITECTURE teste OF somadores1 IS
--“add”: nome_local componente
--portas com mesma designação
COMPONENT add PORT (a, b : IN INTEGER 0 TO 7;
                        s : OUT INTEGER 0 TO 15);
END COMPONENT;
```

```
--"adx": nome_local componente
-- portas com designação diferente
COMPONENT adx PORT (k, l : IN INTEGER 0 TO 7;
                    m : OUT INTEGER 0 TO 15);
END COMPONENT;

-- associação entre "add", "adx" e as entidades de projeto "soma"
--arquitetura de u1 é a ultima compilada
FOR u1: add USE ENTITY WORK.soma;
FOR u2: add USE ENTITY WORK.soma(lenta);
FOR u3: adx USE ENTITY WORK.soma(tipica) PORT MAP(a=>k, b=>l, s=>m);
BEGIN
        u1 : add PORT MAP (x, y, Sr)
        u2 : add PORT MAP (x, y, Sl)
         u3 : adx PORT MAP (x, y, St)
END teste;
```

2.3 ELEMENTOS BÁSICOS DA SITAXE DE VHDL

Nesta sessão, serão discutidas as regras básicas de sintaxe de cada comando de uma descrição VHDL. A figura 2.9 mostra uma declaração VHDL onde estão indicadas algumas das estruturas e comandos que podem aparecer neste tipo de descrição.




```
1  --Exemplos de declarações e comandos de uma descrição VHDL
2  LIBRARY ieee;
3  USE ieee.std_logic_1164.all;
4
5  ENTITY contador_generico IS
6      GENERIC
7          (Max_Count: natural := 9);
8      PORT(
9          (data_input_name    : IN    INTEGER RANGE 0 TO Max_Count;
10         clk_input_name  : IN    STD_LOGIC;
11         clrn_input_name : IN    STD_LOGIC;
12         ena_input_name  : IN    STD_LOGIC;
13         ld_input_name       : IN    STD_LOGIC;
14         count_output_name   : OUT   INTEGER RANGE 0 TO Max_Count );
15 END contador_generico;
16
17 ARCHITECTURE a OF contador_generico IS
18     SIGNAL  count_signal_name   : INTEGER RANGE 0 TO Max_Count;
19 BEGIN
20     PROCESS (clk_input_name, clrn_input_name)
21     BEGIN
22         IF clrn_input_name = '0' THEN
23             count_signal_name <= 0;
24         ELSIF (clk_input_name'EVENT AND clk_input_name = '1') THEN
25             IF ld_input_name = '1' THEN
26                 count_signal_name <= data_input_name;
27             ELSE
28                 IF ena_input_name = '1' THEN
29                     count_signal_name <= count_signal_name + 1;
30                 ELSE
31                     count_signal_name <= count_signal_name;
32                 END IF;
33             END IF;
34         END IF;
35     END PROCESS;
36     count_output_name <= count_signal_name;
37 END a;
```

Figura 2.9 *Declaração VHDL onde estão indicadas algumas das estruturas e comandos.*

Como em qualquer linguagem de descrição, a VHDL utiliza um conjunto bem definido de regras, as quais devem ser seguidas e que definem palavras-chave e a sintaxe da linguagem que se refere ao uso de uma palavra e a ordem que deve ser obedecida para escrever os comandos. A linguagem não é *case-sensitive* (ou seja, não faz diferença entre letras maiúsculas e minúsculas), mas frequentemente são usadas maiúsculas para as palavras reservadas. As palavras-chave, mostradas na tabela 2.4, tem um significado reservado na linguagem e não podem ser usadas para outro propósito. A sintaxe refere-se ao uso de uma palavra e a ordem que deve ser obedecida para escrever os comandos. A tabela 2.5 mostra uma lista dos símbolos especiais e suas sintaxes.

Tabela 2.4 *Palavras reservadas em VHDL.*

abs	downto	library	postponed	srl
access	else	linkage	procedure	subtype
after	elsif	literal	process	then
alias	end	loop	pure	to
all	entity	map	range	transport
and	exit	mod	record	type
architecture	file	nand	*register*	unaffected
array	for	new	reject	units
assert	function	next	rem	until
attribute	*Generate*	nor	report	use
begin	generic	not	return	variable
block	group	null	rol	wait
body	guarded	of	ror	when
buffer	if	on	select	while
bus	impure	open	severity	with
case	in	or	signal	xnor
component	inertial	others	shared	xor
configuration	inout	out	sla	
constant	is	package	sll	
disconnect	label	port	sra	

Tabela 2.5 *Símbolos definidos em VHDL..*

Símbolo	Significado	Símbolo	Significado
+	Adição ou número Positivo	:	Separação entre uma variável e o tipo
-	Subtração ou	“	Aspas dupla

	número negativo		
	Divisão	‘	Aspas simples ou marca de tick
=	Igualdade	**	Exponenciação
<	Menor do que	=>	Seta indicando “então”
>	Maior do que	=>	Seta indicando “recebe”
&	Concatenador	:=	Associação de valor para variáveis
|	Barra vertical	/=	Desigualdade
;	Terminador	>=	Maior do que ou igual a
#	Literal incluído	<=	Menor do que ou igual a
(	Parêntese da Esquerda	<=	Associação de valor para sinais
)	Parêntese da Direita	<>	Caixa
.	Notação de Ponto	--	Comentário

2.3.1 Espaço em Branco

A linguagem VHDL não é sensível a espaços em branco. No exemplo, é mostrado os dois casos, que para o compilador vão ter o mesmo significado.

Quadro 2.12 *Exemplo VHDL não sensível a Espaços em Branco.*

```
BEGIN
    R_g <= (NOT D) AND Clk;
    S_g <= D AND Clk;
    Qa <= NOT (R_g OR Qb);
    Qb <= NOT (S_g OR Qa);
    Q <= Qa;
END Estrutura;
```

```
BEGIN
    R_g <=       (NOT D) AND Clk;
    S_g     <= D AND Clk;
    Qa <= NOT (R_g  OR Qb);
    Qb <= NOT (S_g OR Qa);
    Q <= Qa  ;
END Estrutura;
```

2.3.2 Comentários

Os comentários na linguagem iniciam com dois hifens ”-- ” e vão até o final a área de compilação do código, a cada linha comentário deve-se iniciar com dois hifens, esses comentários serão ignorados pelo compilador.

Quadro 2.13 *Exemplo VHDL de comentário.*

```
-- Módulo que conecta as chaves SW as luzes LEDG
ENTITY Lab1Parte1 IS
        PORT( SW : IN STD_LOGIC_VECTOR(9 DOWNTO 0);
                LEDG : OUT STD_LOGIC_VECTOR(9 DOWNTO 0) );
END Lab1Parte1;

ARCHITECTURE Behavior OF Lab1Parte1 IS
BEGIN
         LEDG <= SW;
END Behavior;
```

2.3.3 Identificadores

Os identificadores são usados para se atribuir nomes à sinais ou processos, devem ser escolhidos de tal maneira que sejam fáceis de serem lembrados e que tenham algum significado no projeto lógico.

- Pode conter apenas caracteres alfanuméricos (A a Z, A a Z, 09) e o caractere sublinhado (_).
- O primeiro caractere deve ser uma letra e a última não pode ser um sublinhado.
- Um identificador não pode incluir dois sublinhados consecutivos.
- Um identificador é não *case sensitive*.
- Um identificador pode ser de qualquer comprimento.

Quadro 2.14 *Exemplos de identificadores.*

VÁLIDOS	INVÁLIDOS
rs_clk	rs_clk
ab08B	sinal#1
A_1023	A__1023

2.4 COMANDOS IF, CASE E LOOP

Abaixo será descrito os comando sequências, usados na linguagem VHDL.

2.4.1 Comando If Then

Algumas operações são executadas somente se certas condições são satisfeitas. Estas operações são chamadas de condicionais e pode-se especifica-las como:

Quadro 2.15 *Comando IF*

```
IF condição THEN operação
END IF;
```

Se a condição for verdadeira, então a lista de operações é executada. As operações desta lista devem estar separadas por ponto-e-vírgula. Ao término destas operações deve ser colocado o *end if*.

Quadro 2.16 *Exemplo -Comando IF then*

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
ENTITY cont4bitsUp IS
        PORT ( rst,clk : IN std_logic;
               q,qb : INOUT std_logic_vector(3 DOWNTO 0));
END cont4bitsUp;

ARCHITECTURE Behavioral OF cont4bitsUp IS
BEGIN
        PROCESS (clk, rst)
        BEGIN
                IF( rst = '1' ) THEN
                        q <= "0000";
                ELSE
                     IF( clk = '1' and clk'event) THEN
                             q <= q + 1;
                     END IF;
                END IF;
         END PROCESS;
         qb <= NOT q;
END behavioral;
```

Abaixo representamos os sinais conectados as entradas e saídas na placa DE0 para teste do código do quadro 2.16, que representa um contador *Up* de 4 bits com *flip-flop* tipo D alternativo.

Tabela 2.6 *Conexão dos pinos para o Exemplo- Comando If then*

Sinal em VHDL	Sinal na Placa DE0	Pinos na placa DE0
CLK	BUTTON[2]	PIN_F1
RST	SW[0]	PIN_J6

Q(0)	LEDG[0]	PIN_J1
Q(1)	LEDG[1]	PIN_J2
Q(2)	LEDG[2]	PIN_J3
Q(3)	LEDG[3]	PIN_H1

2.4.2 Comandos *If Then Else*

O comando *if then* é bem simples e torna-se necessário incorporar alguma modificação para se obter maior flexibilidade. Este comando pode ser modificado para:

Quadro 2.17 *Estrutura comando If then Else*

```
IF condição THEN
   operação_1 ;
ELSE operação_2;
```

Assim se a condição for verdadeira, a operação_1 é executada, caso contrário, a operação_2 é executada. Na verdade, operação_1 e operação_2 podem ser várias operações separadas por ponto-e-vírgula.

Para uma maior flexibilidade ainda, este comando pode incluir novos testes através do *elsif.*

Quadro 2.18 *Exemplo - Comando If then Else*

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY FFD IS
  PORT( clk, rst, pre, ce, d : IN std_logic;
        q, nq : OUT std_logic );
END ENTITY ffd;
ARCHITECTURE behavior OF ffd IS
SIGNAL temp : std_logic;
BEGIN
   PROCESS (clk) IS
     bEGIN
        -- IF rising_edge(clk) THEN -- borda de subida
       IF falling_edge(clk) THEN -- borda de descida
          IF (rst='1') THEN
             q <= '0';
                                  nq <= '1';
          ELSIF (pre='1') THEN
             q <= '1';              nq <= '0';
          ELSIF (ce='1') THEN
             q <= d;
                                   nq <= NOT d;
          END IF;
```

```
    END IF;
  END PROCESS;
END ARCHITECTURE behavior;
```

Abaixo representamos os sinais conectados as entradas e saídas na placa DE0 para teste do exemplo 1.2, que representa um *flip-flop*tipo D ativado na borda de subida, e caso queira ativar a borda de subida basta usar a função *rising_edge* comentada no código no lugar de *falling_edge*.

Tabela 2.7 *Conexão dos pinos para o Exemplo- Comando If then Else*

Sinal em VHDL	Sinal na Placa DE0	Pinos na placa DE0
D	SW[0]	PIN_J6
CLK	BUTON[2]	PIN_F1
PRE	SW[1]	PIN_H5
RST	SW[2]	PIN_H6
CE	SW[9]	PIN_D2
Q	LEDG[0]	PIN_J1
nQ	LEDG[1]	PIN_J2

2.4.3. Comando *CASE*.

O uso do comando *If then Elsif* serve para selecionar uma ramificação dentre as várias possíveis, e que pode ficar complicado se o número de opções se tornar maior que três. Para esta situação utiliza-se o comando *case*. Ao invés de se avaliar uma expressão booleana, o comando case verifica as condições de uma expressão discreta ou um *array*. Cada alternativa é composta por:

Quadro 2.19 *Estrutura comando case*

```
when alternativa => operação
```

Quadro 2.20 *Exemplo de Comando Case*

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
-----------------------------------------------
ENTITY DECODER IS
PORT(I:       IN std_logic_vector(1 DOWNTO 0);
     O:       OUT std_logic_vector(3 DOWNTO 0)
);
END DECODER;

ARCHITECTURE behv OF DECODER IS
BEGIN
  -- PROCESS statement
```

```
   PROCESS (I)
   BEGIN
   -- use case statement

      CASE I IS
            WHEN "00" => O <= "0001";
            WHEN "01" => O <= "0010";
            WHEN "10" => O <= "0100";
            WHEN "11" => O <= "1000";
            WHEN others => O <= "XXXX";
        END CASE;

   END PROCESS;
END behv;
ARCHITECTURE when_else OF DECODER IS
BEGIN
   -- USE WHEN..ELSE statement
   O <=        "0001" when I = "00" ELSE
               "0010" when I = "01" ELSE
               "0100" when I = "10" ELSE
               "1000" when I = "11" ELSE
               "XXXX";
END WHEN_ELSE;
```

2.4.4. Comando *WHILE LOOP*.

O comando *while loop* condicional funciona de forma similar ao comando *If then*. Ele inicia com uma condição lógica, porém tem na última linha um *jump* para o início do *loop*. O *loop* é executado enquanto a condição presente no seu início for válida. A condição é verificada e ser for satisfeita, os comandos presentes no *loop* são executados, caso contrário, o *loop* é considerado completo e o comando passa para a instrução seguinte ao *loop*.

Quadro 2.21 *Exemplo de Comando While Loop*

```
TYPE REGISTER IS bit_vector(7 DOWNTO 0);
TYPE REG_ARRAY IS ARRAY(4 DOWNTO 0) OF REGISTER;
SIGNAL fifo: reg_array;
PROCESS (reset)
VARIABLE i: integer := 0;
BEGIN
IF reset = '1' THEN
   WHILE i <= 4 LOOP
IF i /= 2 THEN
   fifo(i) <= (others => '0');
END IF;
i := i + 1;
   END LOOP;
END IF;
```

```
END PROCESS;
```

2.4.5. Comando *FOR LOOP*

Comando *while loop* repete-se enquanto a condição apresentada no seu início for satisfeita. Algumas vezes pode ser necessário repetir o *loop* por um número específico de vezes. Isto pode ser feito de forma mais conveniente com o comando *for loop*. Este comando não usa nenhuma expressão booleana, mas sim um contador, e desde que o valor do contador esteja em certa faixa, o *loop* é executado. Ao término de cada execução do *loop*, o contador recebe o novo valor. O contador não precisa ser declarado e é tratado como uma constante e só existe dentro do *loop*. A estrutura do comando for *loop* é dada por:

Quadro 2.22 *Estrutura For loop*

```
[nome :] FOR variável in faixa
LOOP
comandos do loop
END LOOP [nome];
```

Quadro 2.23 *Exemplo de Comando For Loop*

```
TYPE REGISTER IS bit_vector(7 DOWNTO 0);
TYPE reg_array is array(4 DOWNTO 0) OF REGISTER;
SIGNAL fifo: reg_array;
PROCESS (reset)
BEGIN
IF reset = '1' THEN
   FOR i IN 4 DOWNTO 0 LOOP
IF i = 2 THEN
      NEXT;
ELSE
       fifo(i) <= (others => '0');
END IF;
     END LOOP;
END IF;
END PROCESS;
```

3. LABORATÓRIOS

Este capítulo tem o intuito de colocar em prática a teoria aprendida do capítulo anterior sobre a linguagem VHDL, ele foi distribuído em 5 laboratórios onde cada um aborda uma temática da lógica computacional descrita na linguagem referida, com tarefas para teste de aprendizagem ao final de cada tópico. Para a implementação foram utilizados a placa DE0 e o *software QuartusII* versão 13.1, produtos da *Terasic* e Altera, que são empresas fabricantes de dispositivos lógicos programáveis.

Uma fotografia da placa DE0 é mostrada na figura. Descreve o layout da placa e indica a localização dos conectores e dos principais componentes (TERASIC TECHNOLOGIES, 2011).

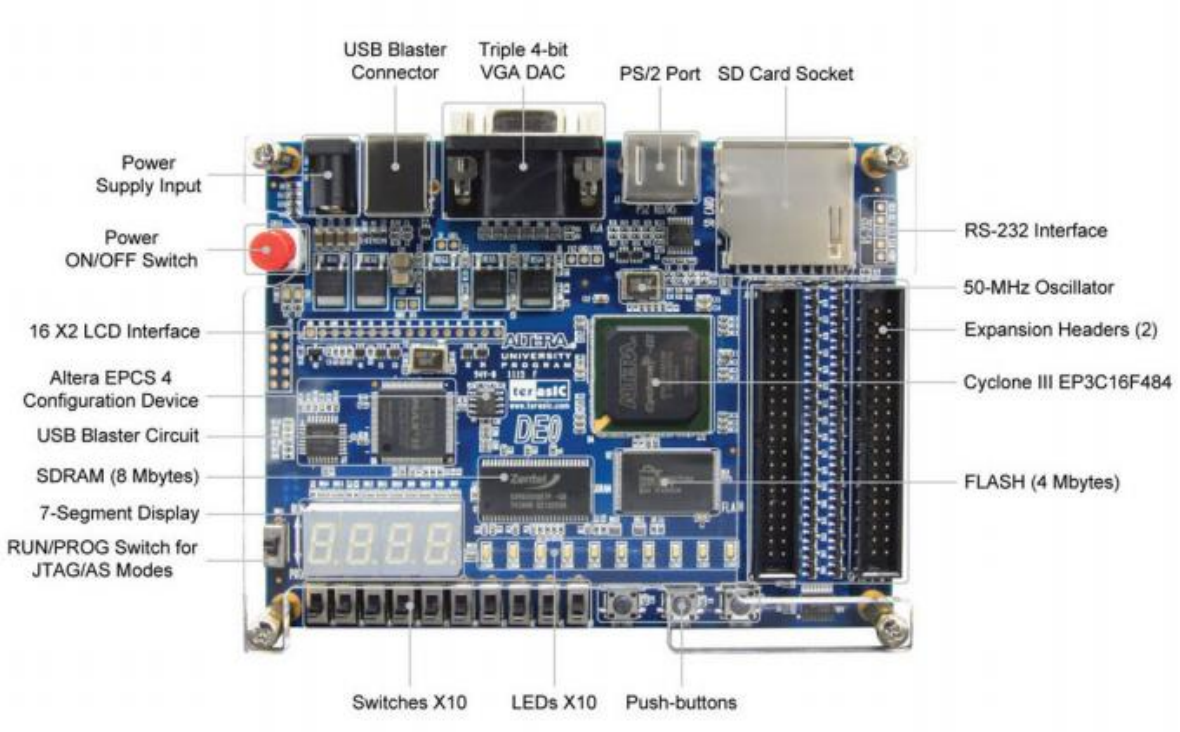


Figura 3.1 *Placa DE0 com indicadores de seus componentes.*

Fonte: *TERASIC TECHNOLOGIES(2011).*

O laboratório 1 tem o propósito de ensinar como conectar entradas e saídas em uma DE0 através da implementação em linguagem VHDL. O laboratório 2 Objetiva ensinar a conversão de números binários para decimais. Enquanto que o laboratório 3, busca mostrar a implementação dos componentes básicos de memória, como: *latches*, *flip-flops* e registradores. Já o laboratório 4 mostra como criar contadores através dos componentes de memória. E por fim o laboratório 5 ensina a utilização do *hardware* ligado direto ao conector VGA.

Serão mostradas a seguir figuras que mostram telas do *software* de criação *QuartusII*, os passos a seguir mostram como criar um projeto para a implementação dos códigos mostrados nos laboratórios.

Passo 1- Ao abrir *Quartus* será visto uma tela parecida com a da figura 3.2, nesta tela clique em *New Project Wizard*, ou em *File* > *New Project Wizard*, para a criação de um novo projeto.

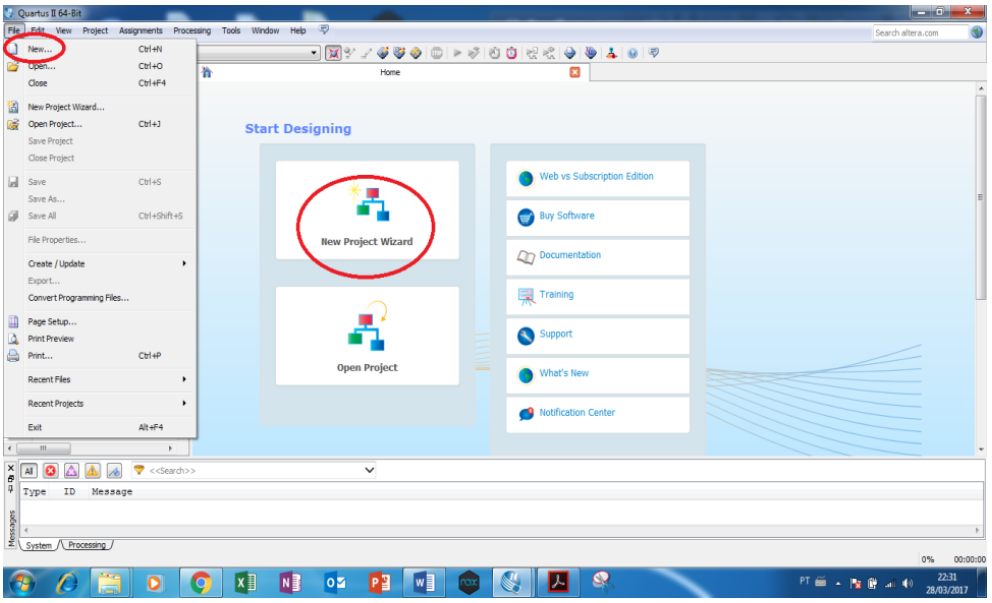


Figura 3.2 *Tela inicial do QuartusII.*

Passo 2- Depois de clicar em *New Project Wizard*, como mostra o passo 1. Clique em *Next*, como mostra a figura 3.3.

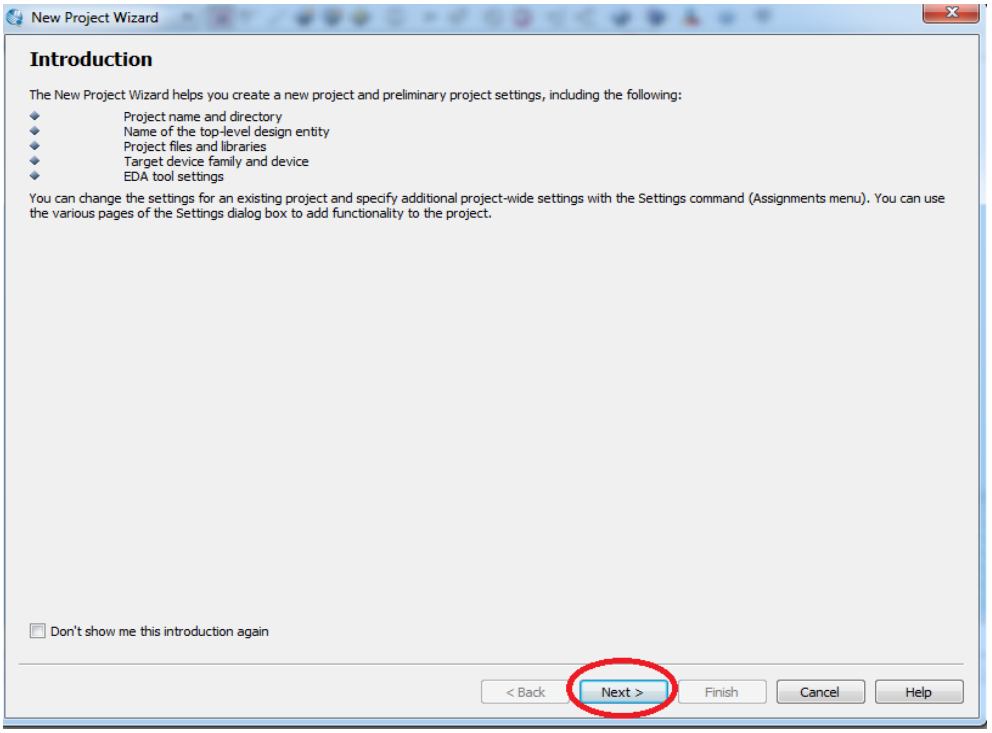


Figura 3.3 *Tela de criação do projeto.*

Passo 3- como mostra a figura 3.4 a seguir, clique no ao indicador 1, que identifica o local ao qual o projeto será salvo, e depois clique no identificador 2 e digite o nome do projeto, depois clique no identificador 3 *Next*.

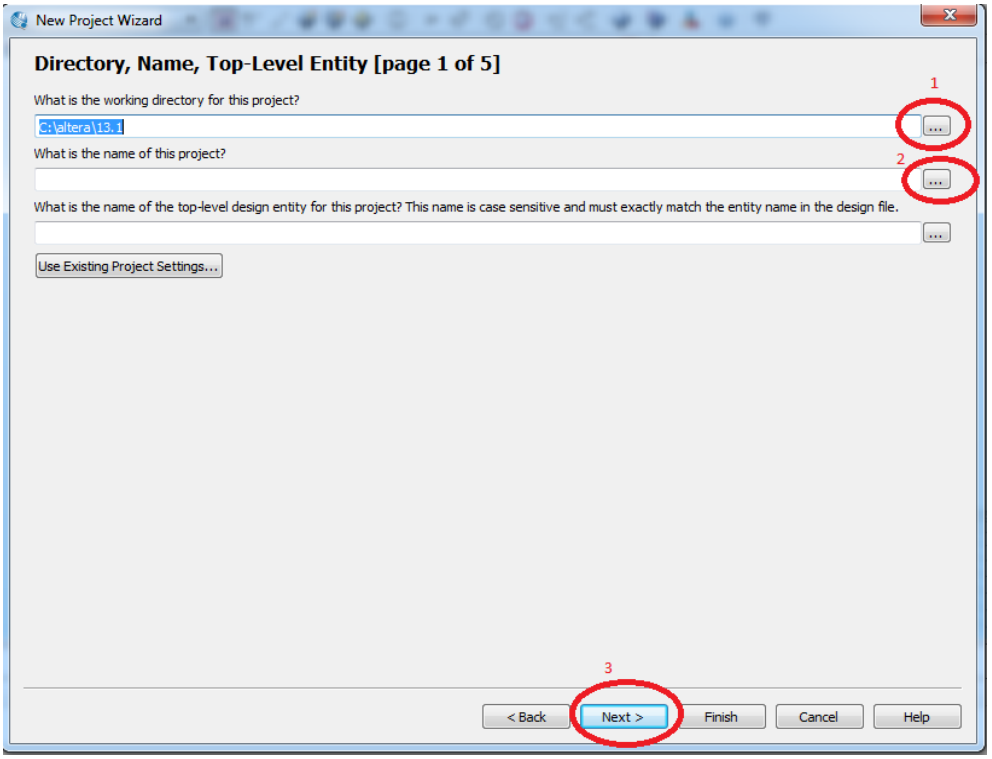


Figura 3.4 *Adição do nome e local do projeto.*

Passo 4- Nesta tela da figura, clicar em *Next* sem alterar nada inicialmente.

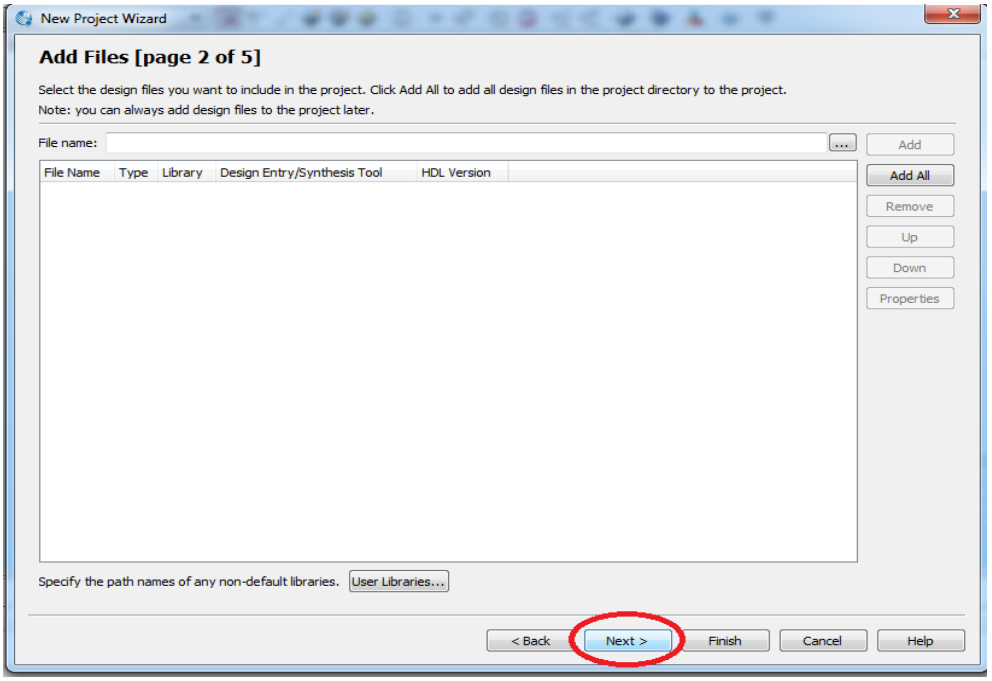


Figura 3.5 *Adição de Componente.*

Passo 5- Clique no identificador 1 e selecione a FPGA *ciclone III* em seguida selecione *EP3C16F484C6* que são os dados da FPGA DE0 que vamos usar como mostra o identificador 2, depois *Next*.

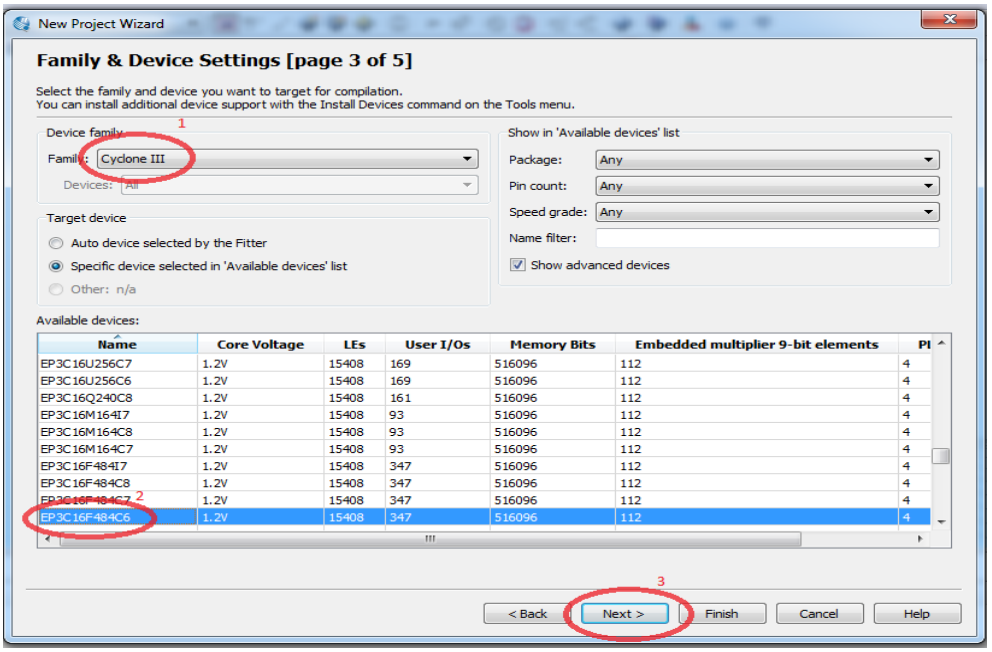


Figura 3.6 *Seleção da FPGA.*

Passo 6- A próxima tela click em *Next* sem alterar nada.

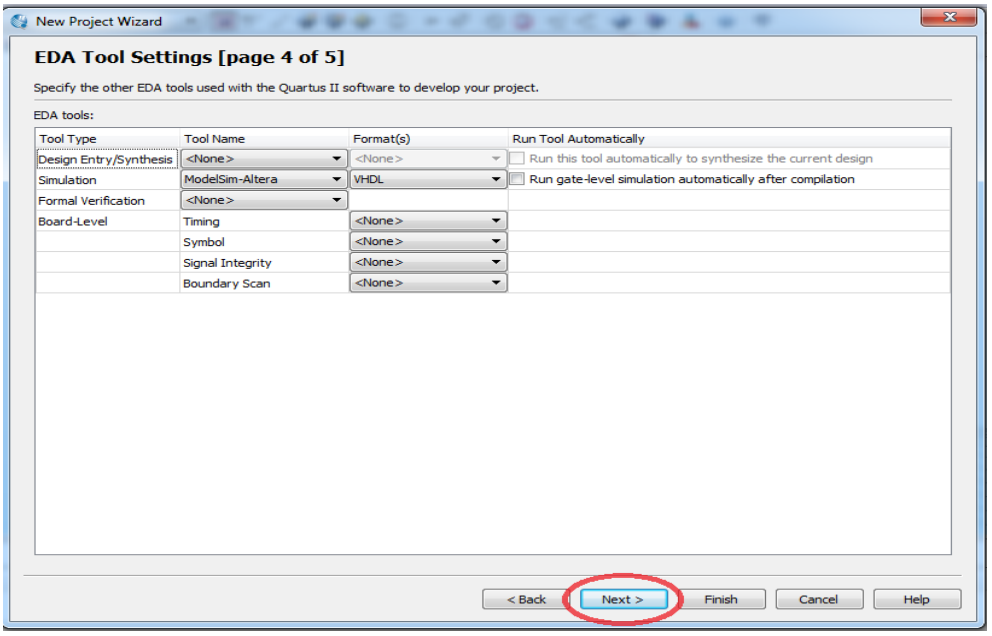


Figura 3.7 *Tela 4 de criação de Projeto.*

Passo 7- Na tela mostrada na figura 3.8, é mostrado um sumário com as configurações feitas, clique em *FINISH*.

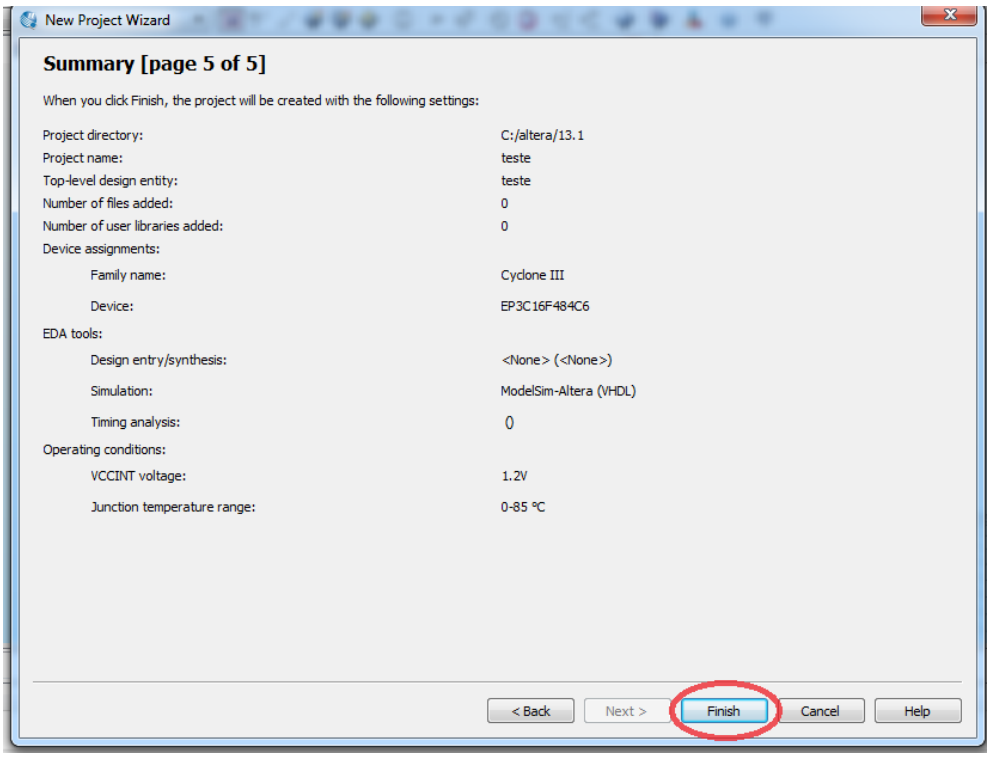


Figura 3.8 *Tela 5 Sumário.*

Passo 8- Na tela mostrada na figura 3.9 será criado um arquivo VHDL, aonde vamos escrever o código a ser implementado, basta clicar em *NEW >VHDL > OK*, como mostra os identificadores da figura.

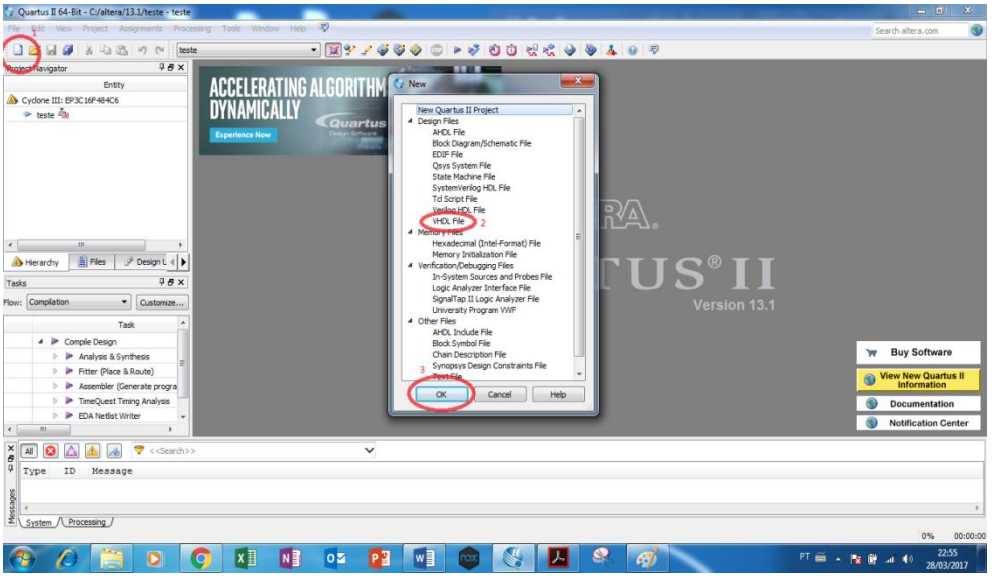


Figura 3.9 *Criação do arquivo.*

Passo 9- Na tela seguinte é mostrada a inserção do código no arquivo criado, lembrando que o nome da entidade do código é o mesmo que a do projeto como mostra a figura 3.10, se os nomes forem diferentes dará erro.

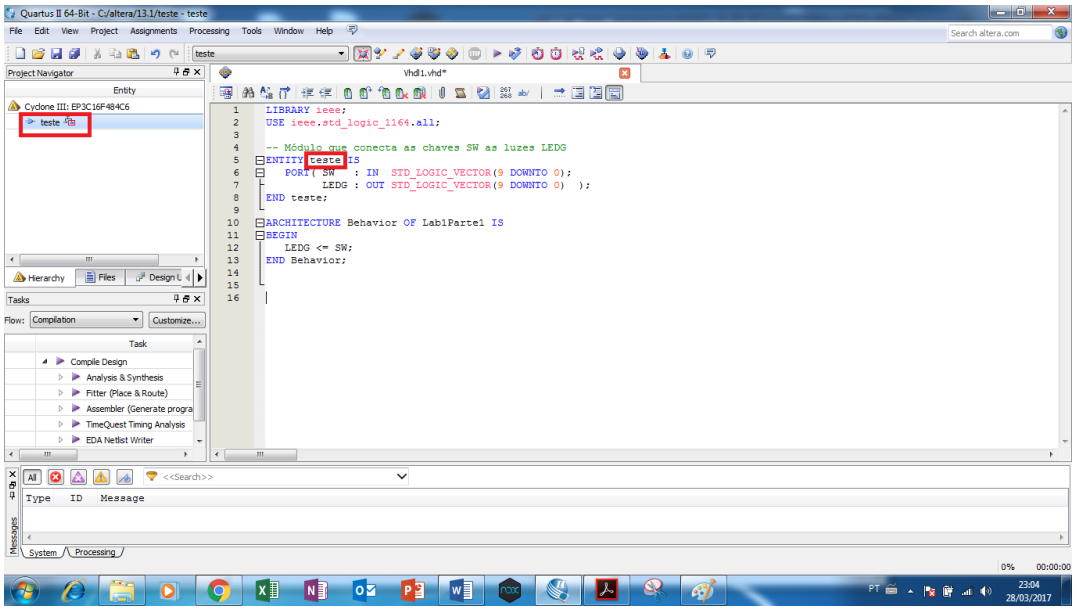


Figura 3.10 *Código em VHDL.*

O arquivo VHDL criado, é processado por várias ferramentas *Quartus II* que analisam o arquivo, sintetizam o código e geram uma implementação dele para o chip de destino. Essas ferramentas são controladas pelo programa aplicativo chamado *Compiler*, encontrado no *Quartus II*. Como mostra o passo seguinte.

Passo 10- Execute o Compilador selecionando *Processing* > Start *Compilation*, ou clicando no ícone da barra de ferramentas que se parece com um triângulo roxo.

Figura 3.11 *Iniciar Compilação.*

Passo 11- Ao clicar em compilar, pedira para salvar o arquivo, coloque o nome da entidade do código e salve no mesmo diretório do projeto.

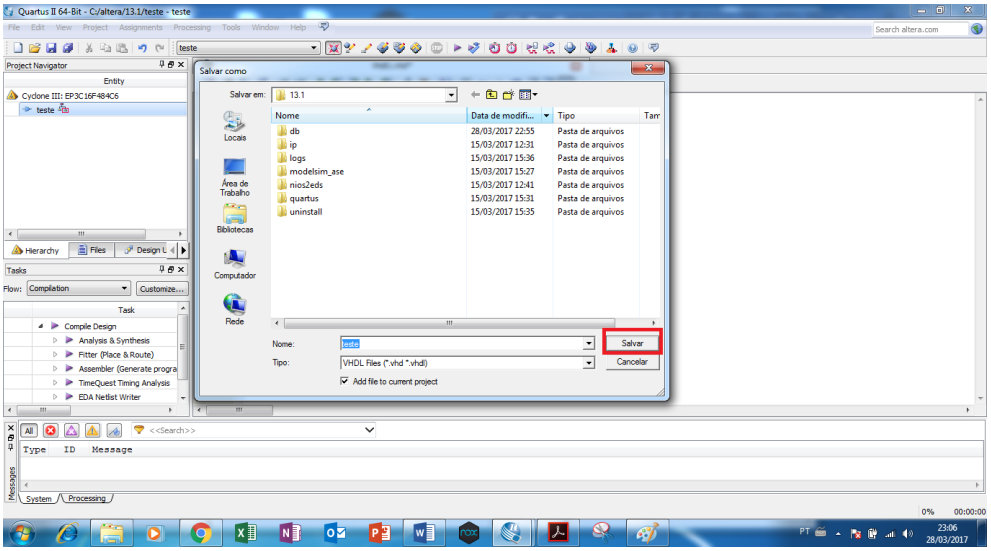


Figura 3.12 *Salvar arquivo á ser compilado.*

Passo 12- Compilando o código em VHDL escrito.

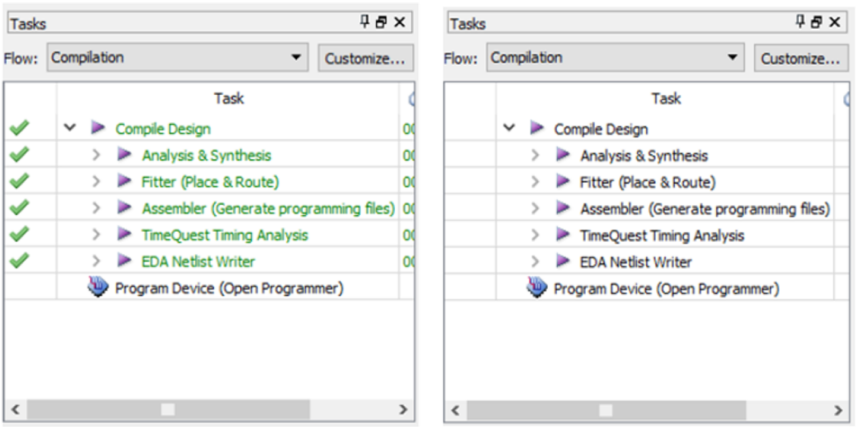


Figura 3.13 *Janelas de antes e depois da compilação.*

Durante a compilação acima, o *Quartus II Compiler* foi livre para escolher quaisquer pinos no FPGA selecionado para servir como entradas e saídas. No entanto, a placa da série

DE0 tem conexões de *hardware* entre os pinos FPGA e os outros componentes na placa. As atribuições de pinos são feitas usando o Editor de Atribuição. Passo 13 - Selecione *Assignments* > *Assignment* Editor, ou clique no ícone selecionado como mostra a figura abaixo .

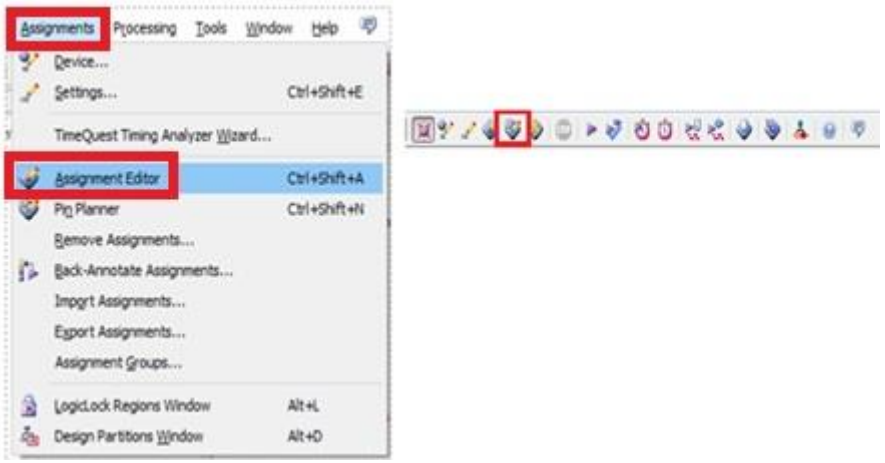


Figura 3.14 *Entrar na janela de adição dos pinos.*

Usaremos duas entradas, rotuladas SW0 e SW1, para fornecer as entradas externas, SW[1] e SW[2] ao nosso arquivo VHDL de exemplo. Estes interruptores são conectados aos pinos FPGA. Iremos conectar as saídas LEDG[0] e LEDG[1] do código aos diodos emissores de luz verde rotulado LEDG0 e LEDG0, como mostra a figura abaixo.

Passo 14- Em *LOCATION* local aonde serão colocados os pinos para associação com a placa, digite PIN_ J1 para associar o LEDG0 a placa, PIN_ J2 para o LEDG1, PIN_J6 e PIN_ H5, para associar as chaves SW0 e SW1 respectivamente.

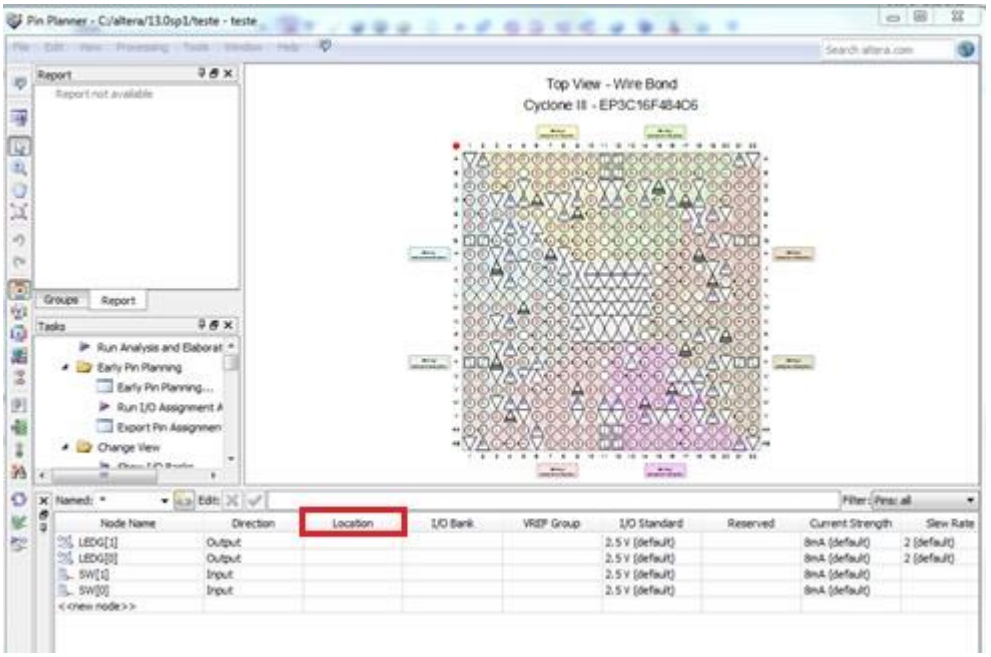


Figura 3.15 *Inserção dos pinos.*

Após associar todos os pinos, compile seu programa novamente para testar seu programa na placa DE0.

Passo 15- Ligue a placa apertando no botão vermelho e com a chave no modo RUN.

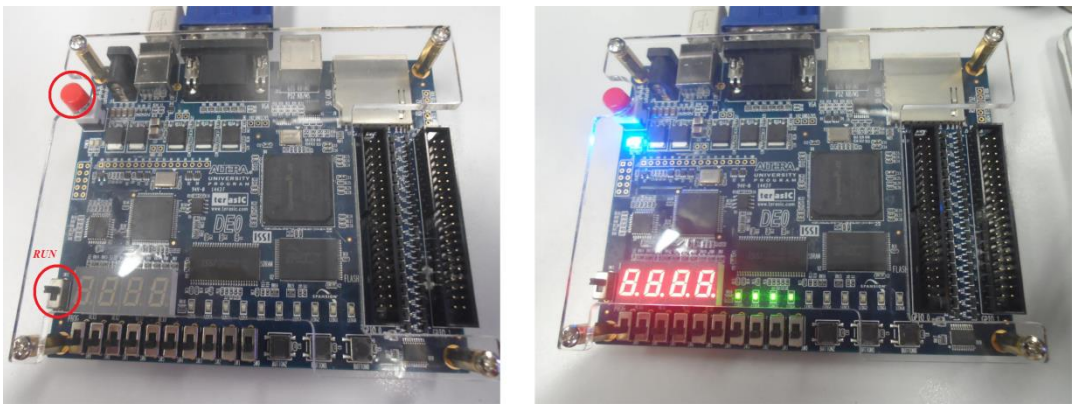


Figura 3.16 *Placa DE0.*

Passo 16- Clique em *Tools* > *Programmer*, para abrir a janela de programação e configuração do dispositivo ou clique no ícone demarcado na figura 3.17.

Figura 3.17 *Abrir a janela de programação.*

Passo 17- Em *Hardware* Setup selecione *USB-BLASTER*, para a configuração da placa.

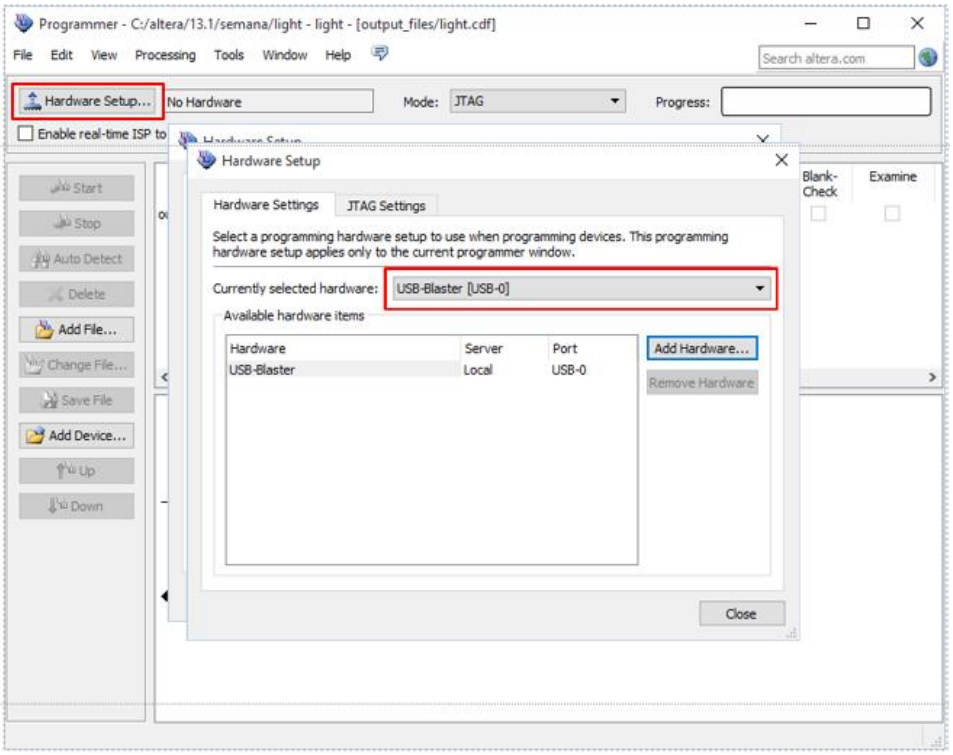


Figura 3.18 *Seleção do Hardware da placa.*

Passo 18- Clique em *Start*, o programa será transferido para sua placa e poderá ser testado.

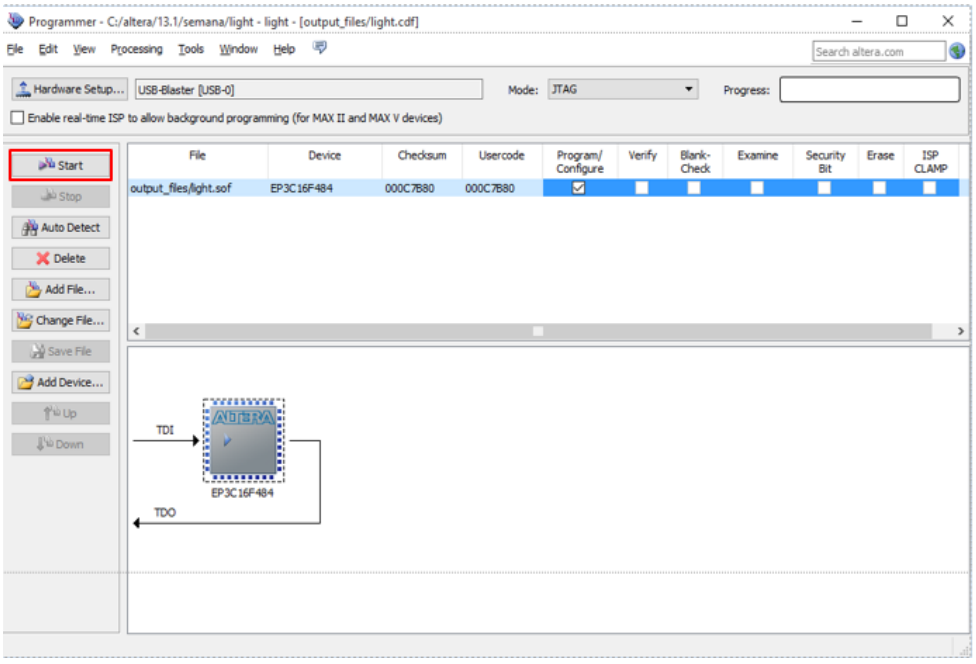


Figura 3.19 *Programação do dispositivo.*

Selecione as Chaves SW0 e SW1 na placa e observe o LEDG0 E LEDG1, para ver o resultado na placa.

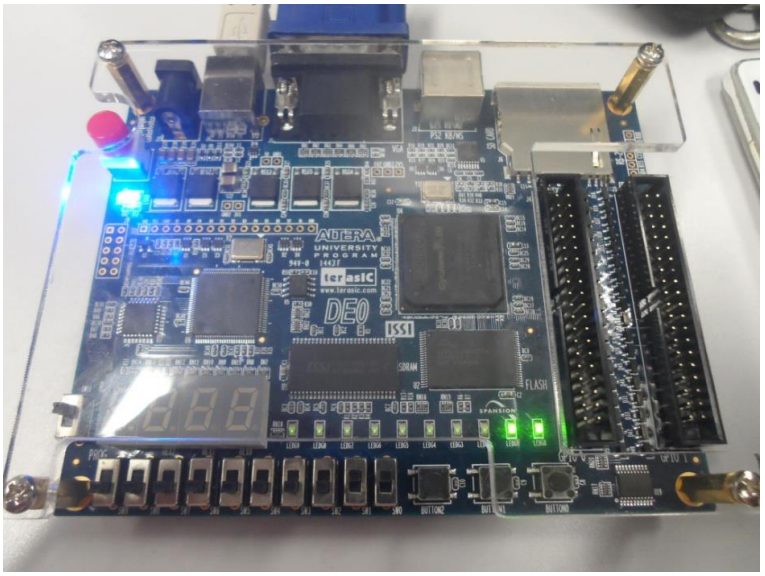


Figura 3.20 *Resultado na placa DE0.*

3.1 LABORATORIO 1.

Switches, Luzes e Multiplexadores.

O propósito desta experiência é aprender como conectar as entradas e saídas dos dispositivos em um FPGA e implementar um circuito com esses dispositivos através da linguagem VHDL. Serão usados os switches SW0 a SW9 como entradas e LEDS e o display de 7 segmentos como saída (ALTERA, 2006).

3.1.1 Parte I - Switches, Luzes

A placa DE0 possui 10 chaves denominadas de SW0 a SW9 que podem ser usadas como entradas digitais e 10 LEDs, LEDG0 a LEDG9, que podem ser usadas como saídas digitais. Como existem 10 chaves e LEDs, é mais conveniente representá-los como como vetores em VHDL.

```
LEDG(9) <= SW(9);
LEDG(8) <= SW(8);
...
LEDG(0) <= SW(0);
```

A placa DE0 possui conexões entre os pinos do FPGA e as chaves e LEDs e para que as chaves e LEDs sejam usados é necessário indicar no projeto do *Quartus II* a quais pinos do FPGA as entradas digitais estão conectadas.

Abaixo o código em VHDL que conecta as chaves aos LEDs.

```
1   LIBRARY ieee;
2   USE ieee.std_logic_1164.all;
3
4   -- Módulo que conecta as chaves SW as luzes LEDG
5   ENTITY Lab1Parte1 IS
6      PORT( SW   : IN  STD_LOGIC_VECTOR(9 DOWNTO 0);
7            LEDG : OUT STD_LOGIC_VECTOR(9 DOWNTO 0)  );
8   END Lab1Parte1;
9
10  ARCHITECTURE Behavior OF Lab1Parte1 IS
11  BEGIN
12     LEDG <= SW;
13  END Behavior;
```

Figura 3.21 *Código em VHDl no Quartus II.*

Fonte: *ALTERA (2006).*

Quadro 3.1 *Código 01- conexão das chaves aos LEDs na placa DE0*

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

-- Módulo que conecta as chaves SW as luzes LEDG
ENTITY Lab1Parte1 IS
        PORT( SW   : IN  std_logic_vector(9 DOWNTO 0);
              LEDG : OUT std_logic_vector(9 DOWNTO 0)  );
END Lab1Parte1;

ARCHITECTURE Behavior OF Lab1Parte1 IS
BEGIN
        LEDG <= SW;
END Behavior;
```

Para que o código acima seja aceito pela placa DE0 é necessário que os pinos do FPGA sejam conectados aos sinais definidos no arquivo em VHDL. No nosso exemplo acima os sinais de entrada são as chaves denominadas de SW0 a SW9 e os sinais de saída são os LEDs denominados de LEDG0 a LEDG9. Na figura abaixo se observa a conexão entre os sinais SW e LEDG e os pinos da placa FPGA.

	Status	From	To	Assignment Name	Value	Enabled	Entity	Comment	Tag
1	✓		out LEDG[0]	Location	PIN_J1	Yes			
2	✓		out LEDG[1]	Location	PIN_J2	Yes			
3	✓		out LEDG[2]	Location	PIN_J3	Yes			
4	✓		out LEDG[3]	Location	PIN_H1	Yes			
5	✓		out LEDG[4]	Location	PIN_F2	Yes			
6	✓		out LEDG[5]	Location	PIN_E1	Yes			
7	✓		out LEDG[6]	Location	PIN_C1	Yes			
8	✓		out LEDG[7]	Location	PIN_C2	Yes			
9	✓		out LEDG[8]	Location	PIN_B2	Yes			
10	✓		out LEDG[9]	Location	PIN_B1	Yes			
11	✓		in SW[0]	Location	PIN_J6	Yes			
12	✓		in SW[1]	Location	PIN_H5	Yes			
13	✓		in SW[2]	Location	PIN_H6	Yes			
14	✓		in SW[3]	Location	PIN_G4	Yes			
15	✓		in SW[4]	Location	PIN_G5	Yes			
16	✓		in SW[5]	Location	PIN_J7	Yes			
17	✓		in SW[6]	Location	PIN_H7	Yes			
18	✓		in SW[7]	Location	PIN_E3	Yes			
19	✓		in SW[8]	Location	PIN_E4	Yes			
20	✓		in SW[9]	Location	PIN_D2	Yes			
21		<<new>>	<<new>>	<<new>>					

Figura 3.22 *Conexão dos pinos da FPGA aos sinais lógicos no VHDL.*

Fonte: *ALTERA (2006).*

No apêndice estão ilustradas todas as entradas e saídas da placa DE0 e em quais pinos do FPGA estas entradas e saídas estão conectadas.

Abaixo a imagem ilustra o resultado do código da conexão das chaves aos LEDS na placa DE0 utilizando VHDL na FPGA DE0, foi adicionada às entradas, os bits 1111100000.

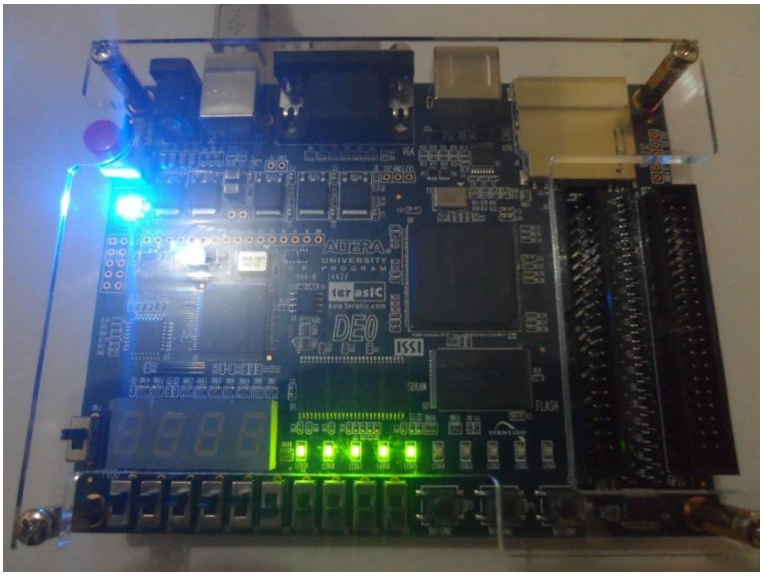
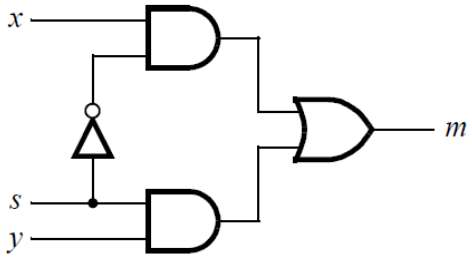


Figura 3.23 *Resultado na FPGA DE0 da Conexão das chaves aos LEDS*

3.1.2 Parte II - Multiplexador 2-to-1

Uma das estruturas lógicas mais utilizadas em linguagens de descrição de *hardware* é o multiplexador. Na figura 3.24 é mostrado um circuito de um multiplexador com 2 entradas e 1

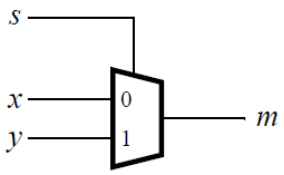
saída, 2-to-1 *multiplexer*. O sinal *s* seleciona qual entrada, *x* ou *y*, vai ser transmitida para a saída *m*.



a) Circuit

s	m
0	x
1	y

b) Truth table



c) Symbol

Figura 3.24 *Multiplexer 2-to-1: a) Circuito; b) Tabela verdade; c) Símbolo*

Fonte: *ALTERA (2006).*

Quando s = 0 o sinal x será enviado para a saída m. Caso s = 1, o sinal y será enviado para a saída m. A lógica de programação em VHDL que descreve um multiplexador 2-to-1 pode ser descrita como:

m <= (NOT (s) AND x) OR (s AND y);

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*. Na listagem abaixo é ilustrado o código em VHDL utilizado para implementar o multiplexador 2-to-1.

Quadro 3.2 *Código 02 – Multiplexador 2-to-1 em VHDL.*

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

-- Módulo que descreve um multiplexador 2-to-1
ENTITY Lab1Parte1 IS
        PORT( x, y, s : IN STD_LOGIC; m : OUT STD_LOGIC );
END Lab1Parte1;

ARCHITECTURE Behavior OF Lab1Parte1 IS
BEGIN
        m <= (NOT (s) AND x) OR (s AND y);
END Behavior;
```

Para implementar o quadro 3.2 descrito acima na placa DE-0 vamos usar o seguinte esquema de conexão entre os pinos do FPGA e os sinais de entrada e saída em VHDL.

Tabela 3.1 *Conexão entre os sinais VHDL e os pinos da FGPA para o Multiplexador 2-to-1.*

Sinal em VHDL	**Sinal na Placa DE0**	**Pinos na placa DE0**
X	SW[0]	PIN_J6
Y	SW[1]	PIN_H5
S	BUTTON[2]	PIN_F1
M	LEDG[0]	PIN_J1

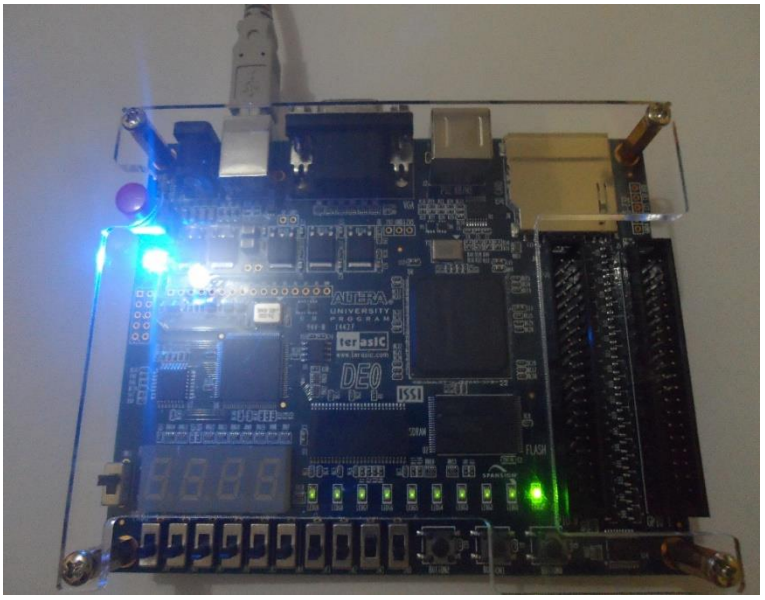


Figura 3.25 *Resultado na FPGA DE0 do multiplexador 2-to-1*

O circuito criado acima pode ser expandido para que um conjunto de entradas seja repassado para a saída, ou seja, um vetor de sinais digitais será repassado para uma saída de acordo com o valor da entrada de seleção. Um esquema desta expansão pode ser observado na figura 3.26.

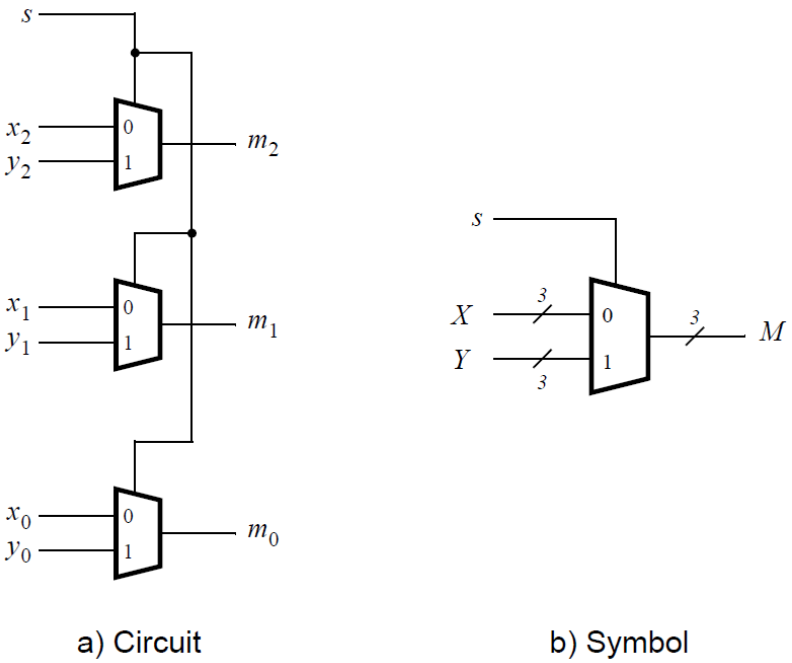


Figura 3.26 *Multiplexação de vetor digital com 3 elementos.*

Fonte: *ALTERA (2006).*

As entradas x0, x1 e x2 serão representadas por um vetor X, assim como as entradas y0, y1 e y2 serão representadas pelo vetor Y e as saídas m0, m1 e m2 serão representadas pelo vetor M. A lógica de programação que utilizada estes vetores pode descrita como:

m(0) <= (NOT (s) AND x(0)) OR (s AND y(0));

m(1) <= (NOT (s) AND x(1)) OR (s AND y(1));

m(2) <= (NOT (s) AND x(2)) OR (s AND y(2));

Quadro 3.3 *Código 03 – Multiplexador 2-to-1 com entrada e saída de vetores.*

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

-- Módulo que descreve um multiplexador 2-to-1 para um vetor de entrada
ENTITY Lab1Parte2 IS
        PORT( X, Y : IN  STD_LOGIC_VECTOR(2 DOWNTO 0);
                    s : IN  STD_LOGIC;
                   M : OUT STD_LOGIC_VECTOR(2 DOWNTO 0) );
END Lab1Parte2;

ARCHITECTURE Behavior OF Lab1Parte2 IS
BEGIN
        M(0) <= (NOT (s) AND X(0)) OR (s AND Y(0));
        M(1) <= (NOT (s) AND X(1)) OR (s AND Y(1));
        M(2) <= (NOT (s) AND X(2)) OR (s AND Y(2));
END Behavior;
```

Para sintetizar o quadro 3.3 foram atribuídos Conexões entre os sinais VHDL e os pinos do FPGA para o multiplexador 2-to-1 com entrada e saída de vetores mostrados na tabela 3.2.

Tabela 3.2 *Conexão entre os sinais VHDL e os pinos do FPGA para o multiplexador 2-to-1 com entrada e saída de vetores.*

Sinal em VHDL	Sinal na Placa DE0	Pinos na placa DE0
X(0)	SW[0]	PIN_J6
X(1)	SW[1]	PIN_H5
X(2)	SW[2]	PIN_H6
Y(0)	SW[3]	PIN_G4
Y(1)	SW[4]	PIN_G5
Y(2)	SW[5]	PIN_J7
S	BUTTON[2]	PIN_F1
M(0)	LEDG[0]	PIN_J1
M(1)	LEDG[1]	PIN_J2
M(2)	LEDG[2]	PIN_J3

Abaixo ilustrado na figura 3.27 e na figura 3.28 o resultado na FPGA de como ficou a entrada dos bits 111000 sem e com o *BUTTON* pressionado respectivamente.

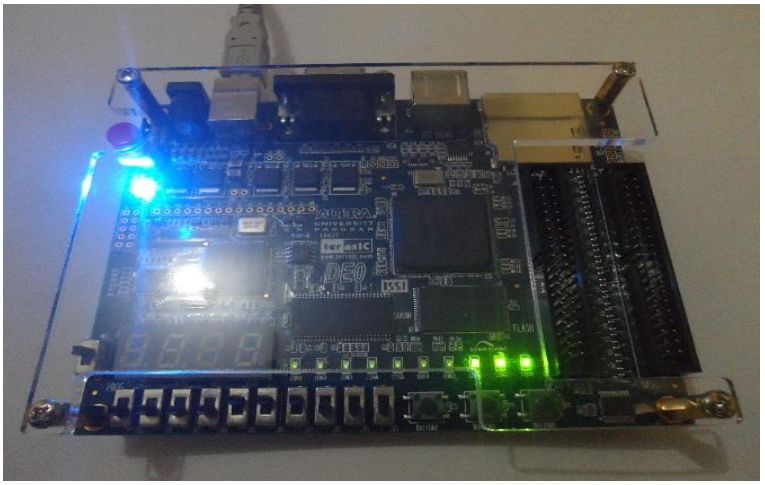


Figura 3.27 *Resultado na FPGA multiplexador 2-to-1 com entrada e saída de vetores.*

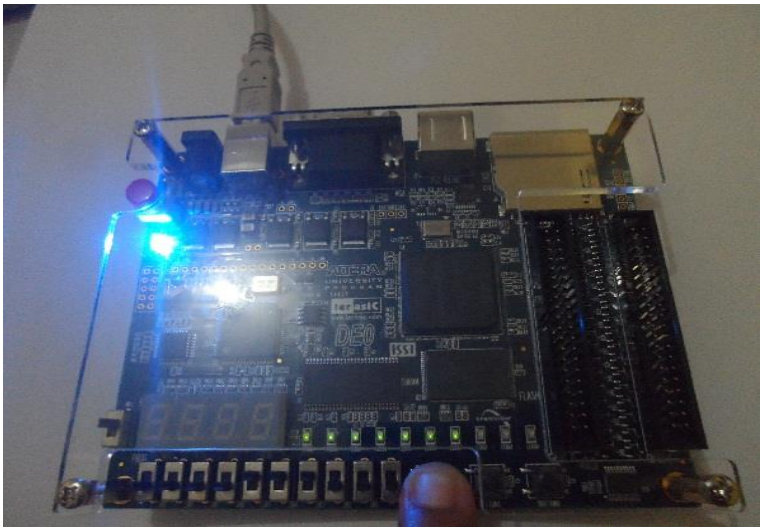
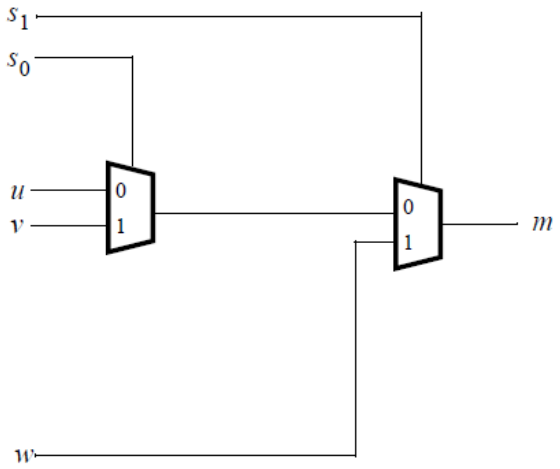


Figura 3.28 *Resultado na FPGA multiplexador 2-to-1 com entrada e saída de vetores com o BUTTON pressionado*

3.1.3 Parte III - Multiplexador 3-to-1

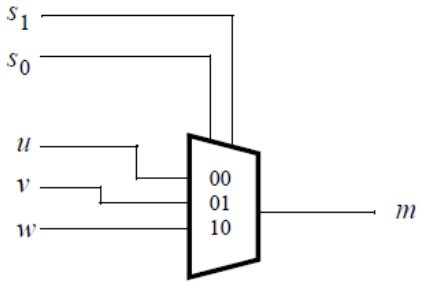
A figura 3.29 mostra uma ilustração de como um multiplexador 3-to-1 pode ser implementado a partir de multiplexadores 2-to-1. O circuito utiliza duas entradas de seleção s1 e s0 e implementa a tabela verdade.



a) Circuit

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

b) Truth table



c) Symbol

Figura 3.29 *Multiplexador 3-to-1 utilizando dois multiplexadores 2-to-1.*

Fonte: *ALTERA (2006).*

O código que sintetiza o esquema mostrado na figura 3.29 está disponível na lista abaixo.

Quadro 3.4 *Código 04– Multiplexador de 2 bits 3-to-1*

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

```
-- Módulo que descreve um multiplexador 3-to-1
ENTITY Lab1Parte3 IS
        PORT( u, v, w, s0, s1 : IN STD_LOGIC; m : OUT STD_LOGIC );
END Lab1Parte3;

ARCHITECTURE Behavior OF Lab1Parte3 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 Behavior;
```

Para que o código 04 possa ser sintetizável, os sinais em VHDL foram atribuídos aos seguintes pinos no FPGA, que são usados para conectar os sinais s0, s1, u, v e w as chaves SW[0-1,7-9] e a saída m ao LED LEDG[0]

Tabela 3.3 *Conexão entre os sinais VHDL e os pinos do FPGA para o Multiplexador de 2 bits 3-to-1.*

Sinal em VHDL	**Sinal na Placa DE0**	**Pinos na placa DE0**
s0	SW[0]	PIN_J6
s1	SW[1]	PIN_H5
u	SW[7]	PIN_E3
v	SW[8]	PIN_E4
w	SW[9]	PIN_D2
m	LEDG[0]	PIN_J1

O resultado do código do Multiplexador de 2 bits 3-to-1 pode ser verificado na Figura 3.30, resultado esse obtido na FPGA DE0.

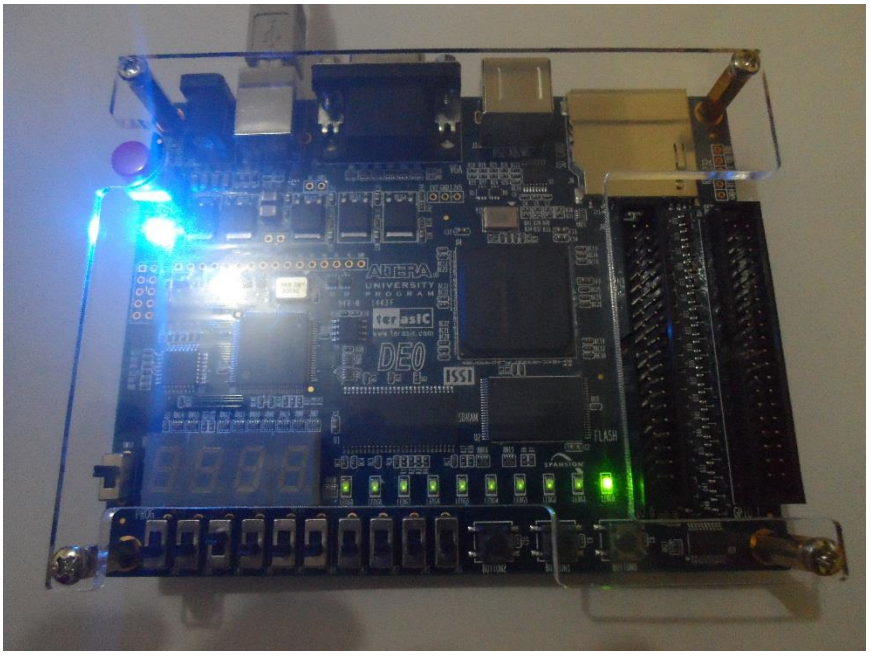


Figura 3.30 *Resultado na FPGA multiplexador 2-to-1 com entrada e saída de vetores com o BUTTON pressionado.*

Tarefa 01: Como devemos proceder para que um quarto sinal z possa ser utilizado de tal forma que a tabela verdade seja a ilustrada no quadro abaixo?

Tabela 3.4 *Tabela verdade a ser ilustrada.*

s1	s0	M
0	0	U
0	1	V
1	0	W
1	1	Z

Tarefa 02: O multiplexador acima envia somente um bit para a saída. Caso se deseje com que mais de um bit seja enviado para a saída, devemos realizar um procedimento semelhante ao mostrado na figura 3.29. A figura 3.31 mostra o símbolo de um multiplexador de 2 bits 3-to-1. Sintetize este circuito na placa DE0.

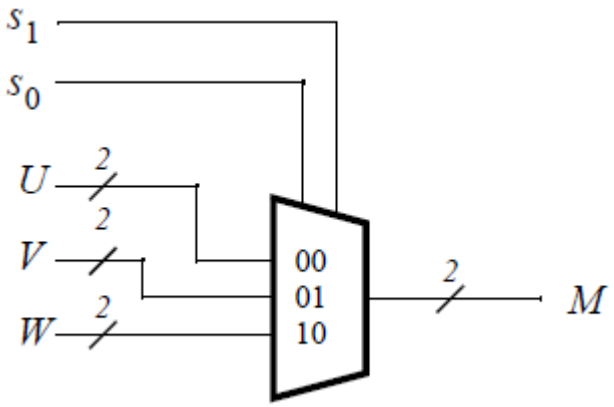


Figura 3.31 *Multiplexador de 2 bits 3-to-1.*

Fonte: *ALTERA (2006).*

3.1.4 Parte IV - Conversor 2 bits para 7 segmentos

A figura 3.32 ilustra um módulo decodificador de 7 segmentos que possui 2 bits c1 e c2de entrada. Este decodificador fornece 7 saídas que são conectados a um display de 7 segmentos na placa DE0. O quadro 05 mostra a tabela verdade que deve ser implementada e seus caracteres: ‘d’, ‘E’, ‘0’ e ‘ ‘.

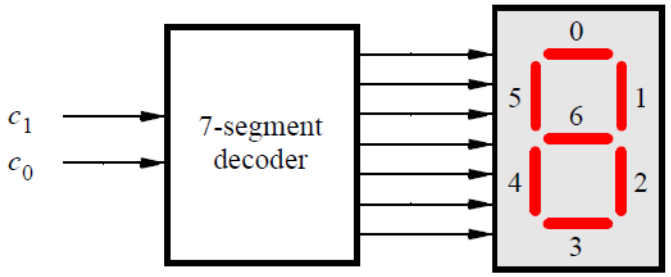


Figura 3.32 *Decodificador 2 bit para 7 segmentos.*

Fonte: *ALTERA (2006).*

Tabela 3.5 *Tabela verdade.*

c1	c0	Caractere
0	0	d
0	1	E
1	0	0
1	1	

O display de 7 segmentos da figura 3.31 está conectado a um pino do FPGA. Cada segmento é iluminado quando o nível lógico 0 (baixo) é aplicado na saída do FPGA conectada ao segmento. As figuras 3.33 e 3.34 ilustram a conexão entre o FPGA e o display de 7 segmentos.

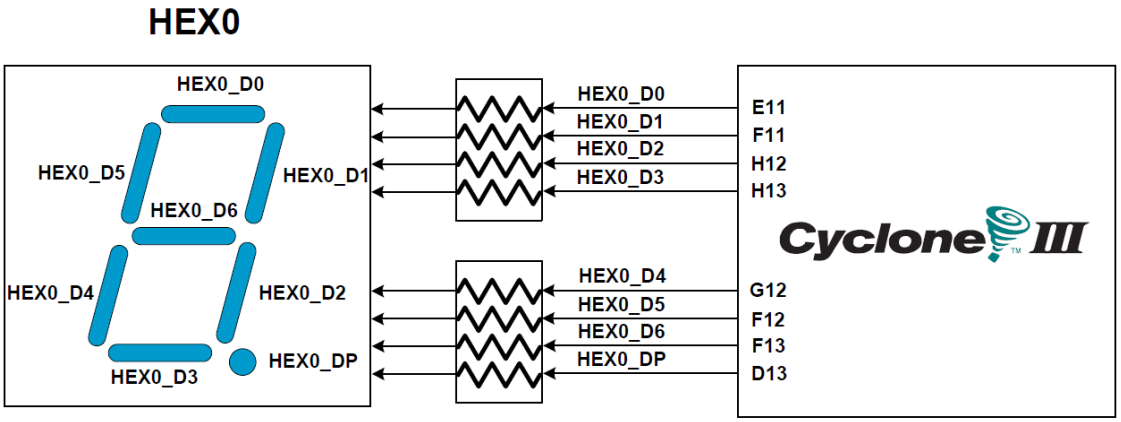


Figura 3.33 *Esquema de ligação entre o primeiro display de 7 segmentos e o FPGA.*

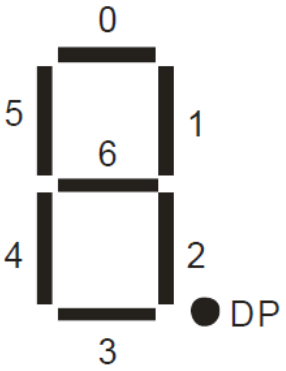


Figura 3.34 *Display de 7 segmentos e suas respectivas numerações de LEDs.*

O código que implementa o conversor 2 bits para 7 segmentos da figura 3.32 está mostrado na lista abaixo.

Quadro 3.5 *Código 05 – Conversor 2 bits para 7 segmentos.*

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

-- Módulo decodificador 2 bits para 7 segmentos
ENTITY Lab1Parte4 IS
	PORT( c0, c1 : IN STD_LOGIC;
		D : OUT STD_LOGIC_VECTOR(6 DOWNTO 0) );
END Lab1Parte4;

ARCHITECTURE Behavior OF Lab1Parte4 IS
BEGIN
	proc1 : PROCESS( c0, c1) IS
	BEGIN
		IF (c1 = '0' AND c0 = '0') THEN
			D(0) <= '1';
			D(1) <= '0';
			D(2) <= '0';
			D(3) <= '0';
			D(4) <= '0';
			D(5) <= '1';
			D(6) <= '0';
		ELSIF (c1 = '0' AND c0 = '1') THEN
			D(0) <= '0';
			D(1) <= '1';
			D(2) <= '1';
			D(3) <= '0';
			D(4) <= '0';
			D(5) <= '0';
			D(6) <= '0';
		ELSIF (c1 = '1' AND c0 = '0') THEn
			D(0) <= '0';
			D(1) <= '0';
			D(2) <= '0';
```

```
                D(3) <= '0';
                D(4) <= '0';
                D(5) <= '0';
                D(6) <= '1';
        ELSE
                D(0) <= '1';
                D(1) <= '1';
                D(2) <= '1';
                D(3) <= '1';
                D(4) <= '1';
                D(5) <= '1';
                D(6) <= '1';
        END IF;
    END PROCESS proc1;
END Behavior;
```

As conexões dos sinais em VHDL deste decodificador são mostrados na tabela 3.6.

Tabela 3.6 *Conexões dos sinais em VHDL do Conversor 2 bits para 7 segmentos*

Sinal em VHDL	**Sinal na Placa DE0**	**Pinos na placa DE0**
c0	SW[0]	PIN_J6
c1	SW[1]	PIN_H5
D(0)	HEX0_D[0]	PIN_E11
D(1)	HEX0_D[1]	PIN_F11
D(2)	HEX0_D[2]	PIN_H12
D(3)	HEX0_D[3]	PIN_H13
D(4)	HEX0_D[4]	PIN_G12
D(5)	HEX0_D[5]	PIN_F12
D(6)	HEX0_D[6]	PIN_F13

A figura 3.35 ilustra como fica o resultado na FPGA do decodificador, para o resultado foram usados os dados do quadro 3.5.

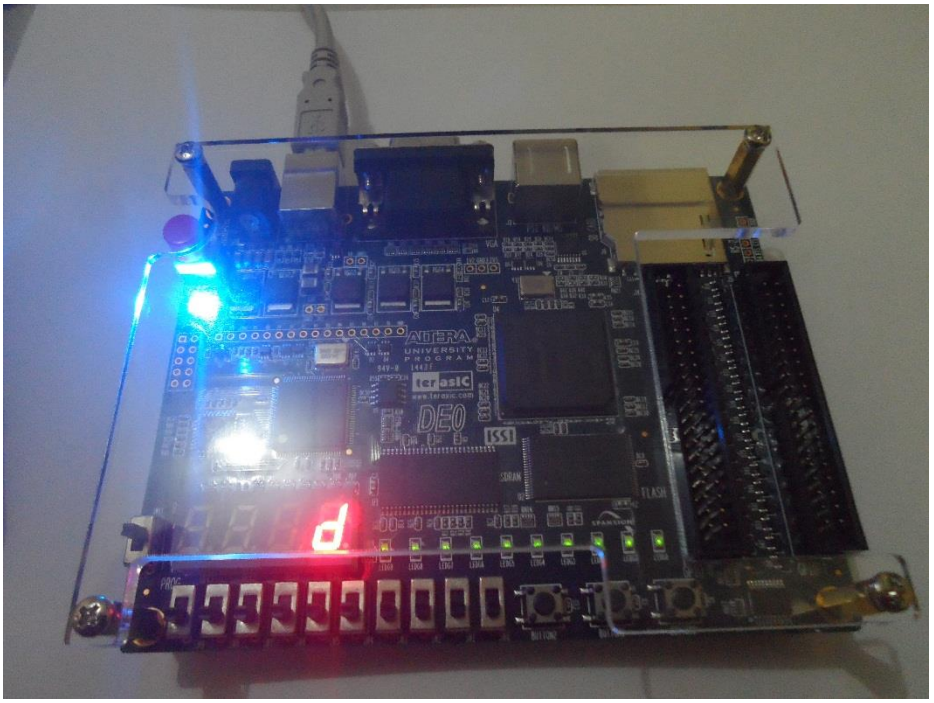


Figura 3.35 *Resultado conversor 2 bits para 7 segmentos.*

Um código alternativo usando o atribuidor de sinais condicional WHEN é ilustrado no quadro. Neste código são utilizados vetores para que o código seja o mais compacto possível. Observe que a ordem em que os vetores são criados, 6 DOWNTO 0, deve ser a mesma ordem em que os bits devem ser representados na atribuição e comparação.

Quadro 3.6 *Código06 – Conversor 2 bit para 7 segmentos com WHEN.*

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

-- Módulo decodificador 2 bits para 7 segmentos
ENTITY Lab1Parte4 IS
        PORT( C : IN std_logic_vector(1 DOWNTO 0);
                D : OUT std_logic_vector(6 DOWNTO 0) );
END Lab1Parte4;

ARCHITECTURE Behavior OF Lab1Parte4 IS
BEGIN
        D <=  "0100001" WHEN C="00" ELSE
                "0000110" WHEN C="01" ELSE
                "1000000" WHEN C="10" ELSE
                "1111111" WHEN C="11";
END Behavior;
```

Utilizando as mesmas conexões do quadro 3.6 e os mesmos dados do quadro 3.5, o resultado na FPGA do conversor 2 bit para 7 segmentos com WHEN está ilustrado na figura 3.36.

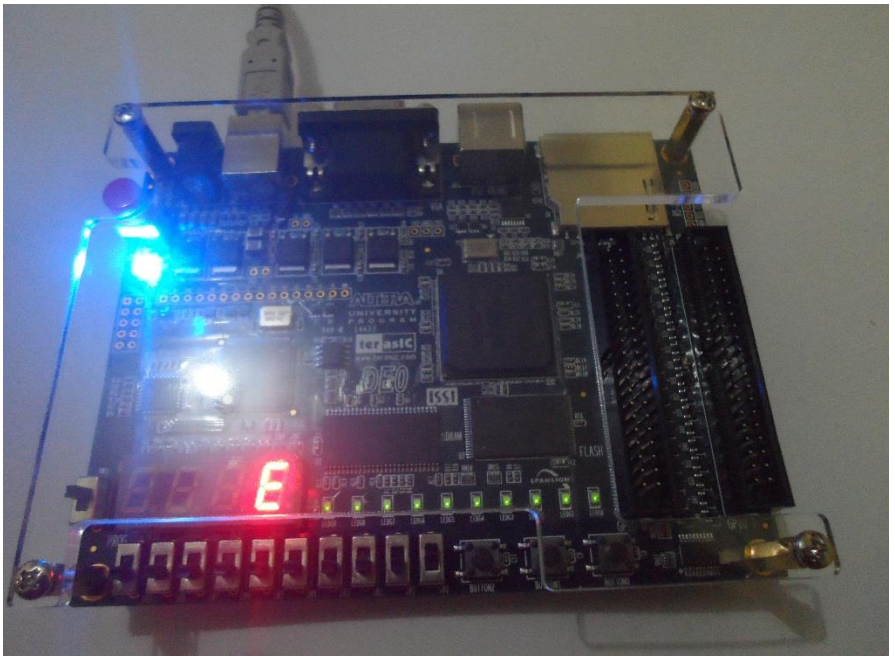


Figura 3.36 *Resultado na FPGA do conversor 2 bit para 7 segmentos com WHEN.*

3.1.5 Parte V - Multiplexador com saída para display

A figura 3.37 ilustra um circuito que possui 2 componentes que foram discutidos anteriormente: multiplexador de 2 bits com 3 entradas e display de 7 segmentos. O multiplexador é responsável por selecionar o conjunto de 2 bits que representa os caracteres representados no quadro 3.5 da Parte IV. Após esta escolha, os bits selecionados serão enviados como entrada para o conversor 2 bits para 7 segmentos que irá acionar o display de 7 segmentos com os caracteres que correspondem aos bits de entrada. O código em VHDL que implementa este circuito é exibido no quadro 3.7.

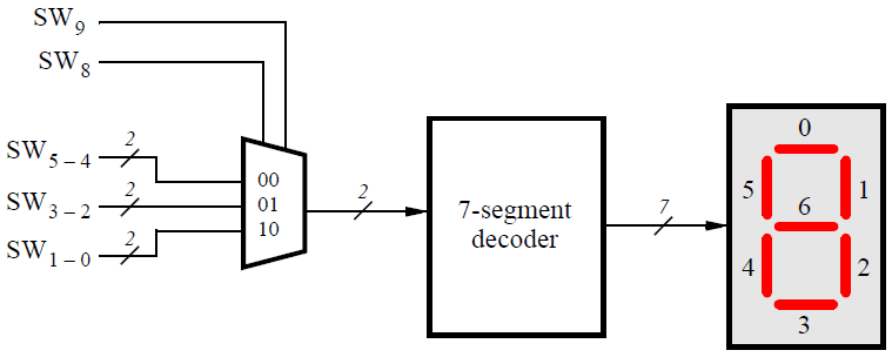


Figura 3.37 *Circuito selecionador de caracteres para display de 7 segmentos.*

Fonte: *ALTERA (2006).*

Quadro 3.7 *Código 07 – Multiplexador de caracteres com saída para display de 7 segmentos.*

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

```
ENTITY Lab1Parte5 IS
	PORT( SW  : IN STD_LOGIC_VECTOR(9 DOWNTO 0);
		HEX0 : OUT STD_LOGIC_VECTOR(0 to 6));
END Lab1Parte5;

ARCHITECTURE Behavior OF Lab1Parte5 IS
	COMPONENT mux_2bit_3to1
		PORT( S, U, V, W : IN STD_LOGIC_VECTOR(1 DOWNTO 0);
			M	: OUT STD_LOGIC_VECTOR( 1 DOWNTO 0));
	END COMPONENT;

	COMPONENT char_7seg
		PORT( C  : IN STD_LOGIC_VECTOR(1 DOWNTO 0);
			Display : OUT STD_LOGIC_VECTOR(0 TO 6));
	END COMPONENT;
	SIGNAL M : STD_LOGIC_VECTOR(1 DOWNTO 0);
BEGIN
	M0 : mux_2bit_3to1 PORT MAP( SW(9 DOWNTO 8), SW(5 DOWNTO 4),
SW(3 DOWNTO 2), SW(1 DOWNTO 0), M);
	H0 : char_7seg PORT MAP( M, HEX0);
END Behavior;

LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY mux_2bit_3to1 IS
	PORT ( S, U, V, W : IN STD_LOGIC_VECTOR(1 DOWNTO 0);
		M : OUT STD_LOGIC_VECTOR(1 DOWNTO 0));
END mux_2bit_3to1;

ARCHITECTURE Behavior OF mux_2bit_3to1 IS
BEGIN
	M <= U    WHEN S = "00" ELSE
	     V    WHEN S = "01" ELSE
	     W    WHEN S = "10" ELSE
	    "11" WHEN S = "11";
END Behavior;

LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY char_7seg IS
	PORT( C	: IN STD_LOGIC_VECTOR(1 DOWNTO 0);
		Display : OUT STD_LOGIC_VECTOR(0 TO 6));
END char_7seg;

ARCHITECTURE Behavior OF char_7seg IS
BEGIN
	Display <= "1000010" WHEN C = "00" ELSE
		   "0110000" WHEN C = "01" ELSE
```

```
                    "0000001" WHEN C = "10" ELSE
                    "1111111" WHEN C = "11";
END Behavior;
```

Este código utiliza componentes para implementar o multiplexador e o conversor 2 bits para 7 segmentos. Na implementação da arquitetura do circuito é usada a palavra reserva *COMPONENT* para destacar o uso de um componente que será futuramente declarado, como pode ser visto em *mux_2bit_3to1* e *char7seg*. Diz-se que o componente principal, Lab1ParteV, está instanciando os outros componentes, *mux_2bit_3to1* e *char7seg*. Para a correta utilização dos componentes, o mapeamento entre as portas dos componentes principal e secundários tem que ser realizado para que as entradas e saídas intermediárias sejam corretamente conectadas. Este mapeamento pode ser observado ao lado das *labels M0* e *H0*.

Para testar o código acima se utiliza as conexões de sinal VHD e sinal da placa DE0, mostrados na tabela 3.7.

Tabela 3.7 *Conexões dos sinais em VHDL do Conversor 2 bits para 7 segmentos.*

Sinal em VHDL	**Sinal na Placa DE0**	**Pinos na placa DE0**
SW(0)	SW[0]	PIN_J6
SW(1)	SW[1]	PIN_H5
SW(2)	SW[2]	PIN_H6
SW(3)	SW[3]	PIN_G4
SW(4)	SW[4]	PIN_G5
SW(5)	SW[5]	PIN_J7
SW(6)	SW[6]	PIN_H7
SW(7)	SW[7]	PIN_E3
SW(8)	SW[8]	PIN_E4
SW(9)	SW[9]	PIN_D2
HEX0(0)	HEX0[0]	PIN_E11
HEX0(1)	HEX0[1]	PIN_F11
HEX0(2)	HEX0[2]	PIN_H12
HEX0(3)	HEX0[3]	PIN_H13
HEX0(4)	HEX0[4]	PIN_G12
HEX0(5)	HEX0[5]	PIN_F12
HEX0(6)	HEX0[6]	PIN_F13

O resultado do Multiplexador de caracteres com saída para display de 7 segmentos mostrado na FPGA, ilustrado na imagem 3.38 abaixo, utilizou os dados da tabela 3.7.

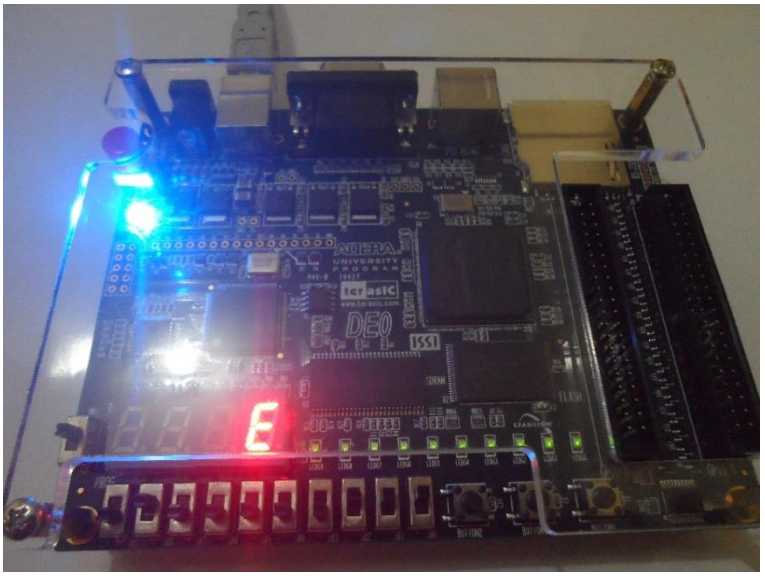


Figura 3.38 *Resultado na FPGA do Multiplexador de caracteres com saída para display de 7 segmentos.*

Tarefa 3: Crie um código em VHDL que implemente a tabela-verdade abaixo. Aqui devem ser usados todos os 4 displays de 7 segmentos da placa DE0.

Tabela 3.8 *Tabela-verdade da Tarefa 3.*

SW9	SW8	HEX3	HEX2	HEX1	HEX0
0	0		D	E	0
0	1	d	E	0	
1	0	E	0		D
1	1	0		d	E

3.2 LABORATORIO 2.

Este tópico aborda a construção de circuitos combinacionais que implementam conversão de números binários para decimal e adição em BCD (binary-code-decimal) (ALTERA, 2008).

3.2.1 Parte I - Decodificador Binário Decimal

Vamos utilizar os displays de 7 segmentos HEX1 e HEX0 para exibirem os números decimais que correspondem aos valores binários representados pelas chaves SW0-9. O display HEX1 vai representar em decimal o número binário representado pelo conjunto de chaves SW7-4, enquanto que o display HEX0 vai representar o conjunto de chaves SW3-0. Cada display deverá representar os números de 0 a 9. Os valores binários de 1010 (10 em decimal) a 1111 (15em decimal) não deverão ser exibidos

Quadro 3.8 *Código 08 – Decodificador Binário Decimal.*

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

```
ENTITY Lab2Parte1 IS
        PORT( SW : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
        HEX0 : OUT STD_LOGIC_VECTOR(6 DOWNTO 0));
END Lab2Parte1;

ARCHITECTURE Behavior OF Lab2Parte1 IS
BEGIN
      HEX0 <= "1000000" WHEN SW = "0000" ELSE
                  "1111001" WHEN SW = "0001" ELSE
                  "0100100" WHEN SW = "0010" ELSE
                  "0110000" WHEN SW = "0011" ELSE
                  "0011001" WHEN SW = "0100" ELSE
                  "0010010" WHEN SW = "0101" ELSE
                  "0000010" WHEN SW = "0110" ELSE
                  "1111000" WHEN SW = "0111" ELSE
                  "0000000" WHEN SW = "1000" ELSE
                  "0010000" WHEN SW = "1001" ELSE
                 "1111111";
END Behavior;
```

Tabela 3.9 *Conexões dos sinais em VHDL do Decodificador Binário Decimal.*

Sinal em VHDL	**Sinal na Placa DE0**	**Pinos na placa DE0**
HEX0(0)	HEX0[0]	PIN_E11
HEX0(1)	HEX0[1]	PIN_F11
HEX0(2)	HEX0[2]	PIN_H12
HEX0(3)	HEX0[3]	PIN_H13
HEX0(4)	HEX0[4]	PIN_G12
HEX0(5)	HEX0[5]	PIN_F12
HEX0(6)	HEX0[6]	PIN_F13

O resultado do código do decodificador binário decimal na FPGA pode ser visualizado na Figura 3.39 abaixo.

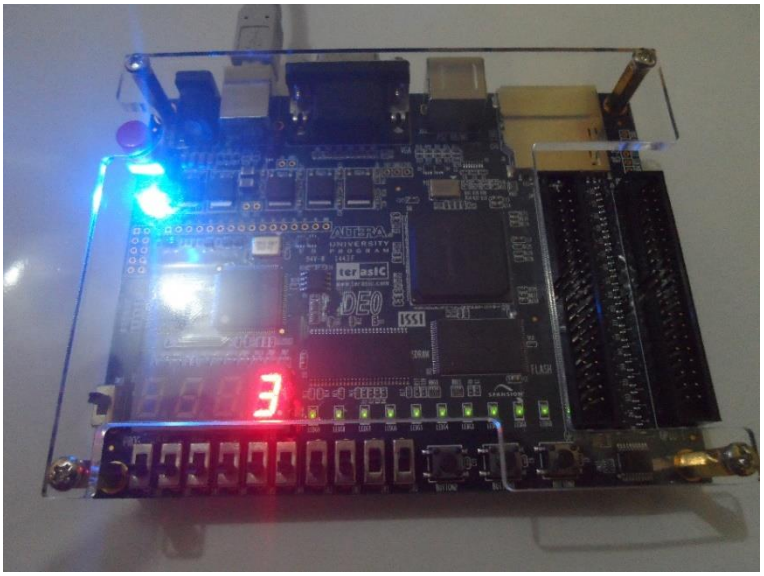


Figura 3.39 *Resultado na FPGA do decodificador binário decimal.*

3.2.2 Parte II - Decodificador binário para decimal com 2 dígitos.

Nesta parte vamos projetar um circuito que converte um número binário com 4 bits V = v3v2v1v0 em seu equivalente em decimal com dois dígitos D = d1d0. A tabela abaixo mostra as entradas e saídas equivalentes.

Tabela 3.10 *Sinais em binário e seus respectivos em decimal.*

Valor binário				Valor decimal	
v_3	v_2	v_1	v_0	d_1	d_2
0	0	0	0	0	0
0	0	0	1	0	1
0	0	1	0	0	2
0	0	1	1	0	3
0	1	0	0	0	4
0	1	0	1	0	5
0	1	1	0	0	6
0	1	1	1	0	7
1	0	0	0	0	8
1	0	0	1	0	9
1	0	1	0	1	0
1	0	1	1	1	1
1	1	0	0	1	2
1	1	0	1	1	3
1	1	1	0	1	4
1	1	1	1	1	5

A Figura 3.39 mostra um desenho parcial do deste circuito, que inclui um comparador para controlar o display de 7 segmentos. Vamos completar o projeto deste circuito criando uma

entidade que inclui comparadores, multiplexadores e o circuito A. Esta entidade deverá ter como entrada os quatro bits em V, a saída M com quatro bits e a saída z. Para esta tarefa não vamos utilizar nenhum comando condicional, somente a Álgebra de *Boole*.

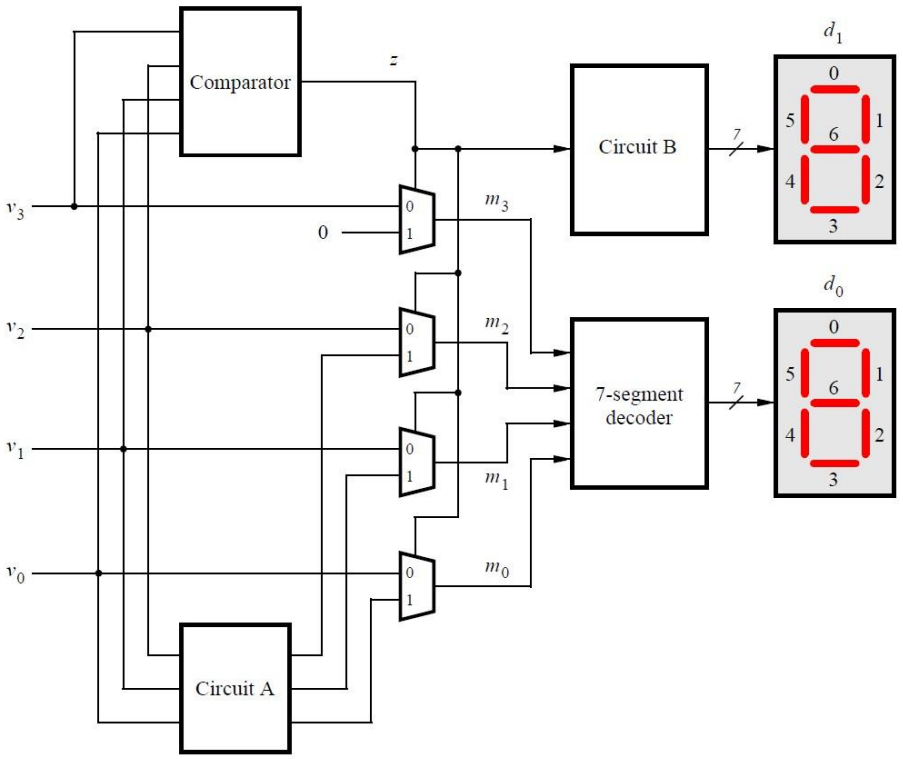


Figura 3.40 *Circuito parcial que converte binário em decimal.*

Fonte: *ALTERA (2008)* .

Abaixo o quadro 3.9 representa a implementação em VHDL do circuito parcial do decodificar binário para decimal com 2 dígitos.

Quadro 3.9 *Código 09 – Circuito parcial do decodificar binário para decimal com 2 dígitos*

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

ENTITY Lab2Parte2 IS
        PORT( V : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
                M : OUT STD_LOGIC_VECTOR(3 DOWNTO 0);
                z : OUT STD_LOGIC );
END Lab2Parte2;

ARCHITECTURE Behavior OF Lab2Parte2 IS
SIGNAL C : STD_LOGIC_VECTOR(2 DOWNTO 0);
SIGNAL temp : STD_LOGIC;
BEGIN
        temp <= V(3) AND (V(2) OR V(1));
        C(2) <= V(2) AND V(1);
        C(1) <= V(2) AND (NOT V(1));
        C(0) <= V(0) AND (V(2) OR V(1));
```

```
        M(0) <= ((NOT (temp) AND V(0)) OR (temp AND C(0)));
        M(1) <= ((NOT (temp) AND V(1)) OR (temp AND C(1)));
        M(2) <= ((NOT (temp) AND V(2)) OR (temp AND C(2)));
        M(3) <= ((NOT (temp) AND V(3)) OR (temp AND '0'));
        z <= temp;
END Behavior;
```

Para testar a entidade criada acima vamos utilizar as chaves e LEDs indicados na tabela:

Tabela 3.11 *Conexões dos sinais em VHDL do Circuito parcial do decodificar binário para decimal com 2 dígitos.*

Sinal em VHDL	Sinal na Placa DE0	Pinos na placa DE0
V(0)	SW[0]	PIN_J6
V(1)	SW[1]	PIN_H5
V(2)	SW[2]	PIN_H6
V(3)	SW[3]	PIN_G4
M(0)	LEDG[0]	PIN_J1
M(1)	LEDG[1]	PIN_J2
M(2)	LEDG[2]	PIN_J3
M(3)	LEDG[3]	PIN_H1
Z	LEDG[4]	PIN_F2

A imagem 3.41 mostra como ficou o resultado na FPGA do Circuito parcial do decodificar binário para decimal com 2 dígitos, o resultado é referente aos bits 1101.

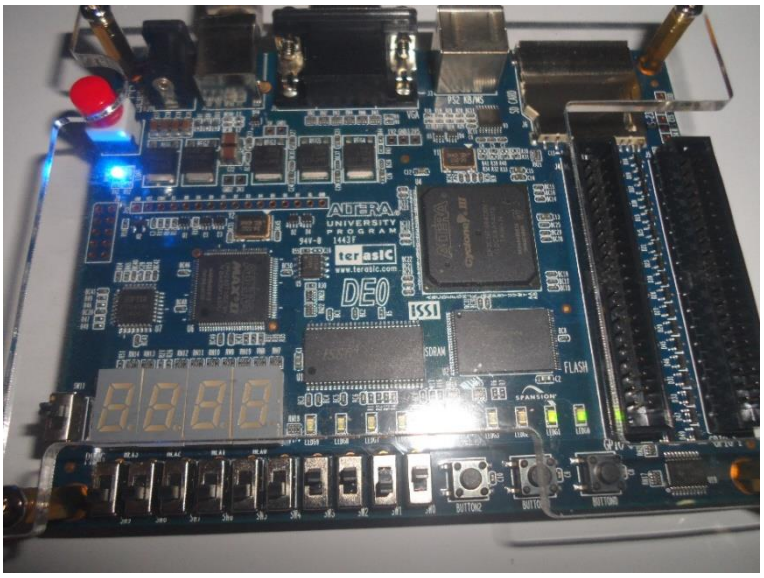


Figura 3.41 *Resultado na FPGA do circuito parcial do decodificar binário para decimal com 2 dígitos.*

Tarefa 4: Inclua o circuito B da figura 3.40 e adicione os dois displays de 7 segmentos para de tal forma que a saída do circuito B seja colocada no display HEX1 e a saída do

decodificador de 7 segmentos seja colocada no display de 7 segmentos HEX0, ou seja: d1 = HEX1 e d2 = HEX2.

3.2.3 Parte III - Somador completo

A figura 3.42 mostra um somador completo utilizando portas lógicas básicas (XOR) e um multiplexador 2-to-1, cuja tabela-verdade é mostrada na tabela 3.12. A figura 3.43 ilustra este somador como um componente tendo como entrada a, b e ci e saída s e co, ou seja, cos = a + b + ci.

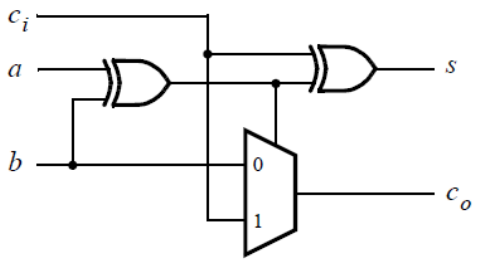


Figura 3.42 *Somador completo de 2 bits.*

Fonte: *ALTERA (2008).*

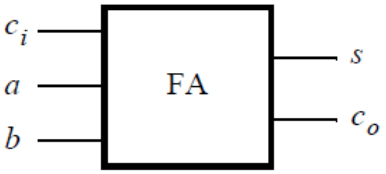


Figura 3.43 *Entradas e saídas do somador completo de 2 bits.*

Fonte: *ALTERA (2008).*

Tabela 3.12 *Tabela-verdade do somador completo de 2 bits.*

Entrada			**Saída**	
b	**a**	**ci**	**co**	**s**
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

A implementação em VHDL do somador completo mostrado acima está descrita no quadro 3.10, exibo abaixo.

Quadro 3.10 *Código 10 – Somador completo de 2 dígitos*

```
LIBRARY IEEE;
```

```
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY FullAdder IS
        PORT( a, b, ci : IN STD_LOGIC;
                s, co : OUT STD_LOGIC );
END FullAdder;

ARCHITECTURE Behavior OF FullAdder IS
BEGIN
        s <= a XOR b XOR ci;
        co <= (a AND b) OR (ci and (a XOR b));
END Behavior;
```

Com este somador completo é possível executar a soma de dois números binários de qualquer comprimento de bits bastando para isso colocá-lo em cascata conforme exemplificado na figura 3.44 onde é mostrado um somador de 4 bits utilizando 4 somadores de 2 bits.

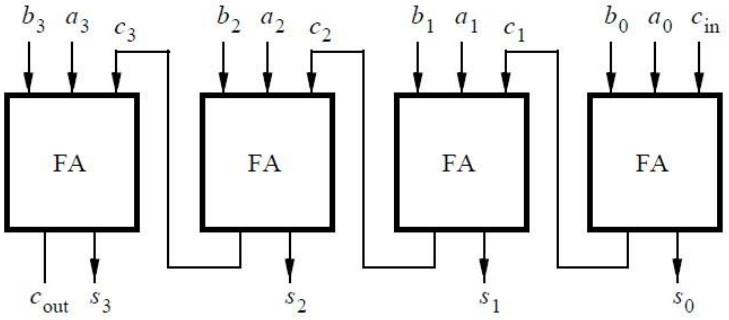


Figura 3.44 *Somador completo de 4 bits utilizando o componente somador completo de 2 bits.*

Fonte: *ALTERA (2008).*

Para se fazer a reutilização de código no VHDL é preciso fazer com que o código do somador completo de 2 bits faça parte do projeto principal do somador completo de 4 bits. Para isso é necessário que na segunda tela do *Project Wizard* seja incluído o arquivo do somador completo de 2 bits, conforme ilustrado na figura 3.45.

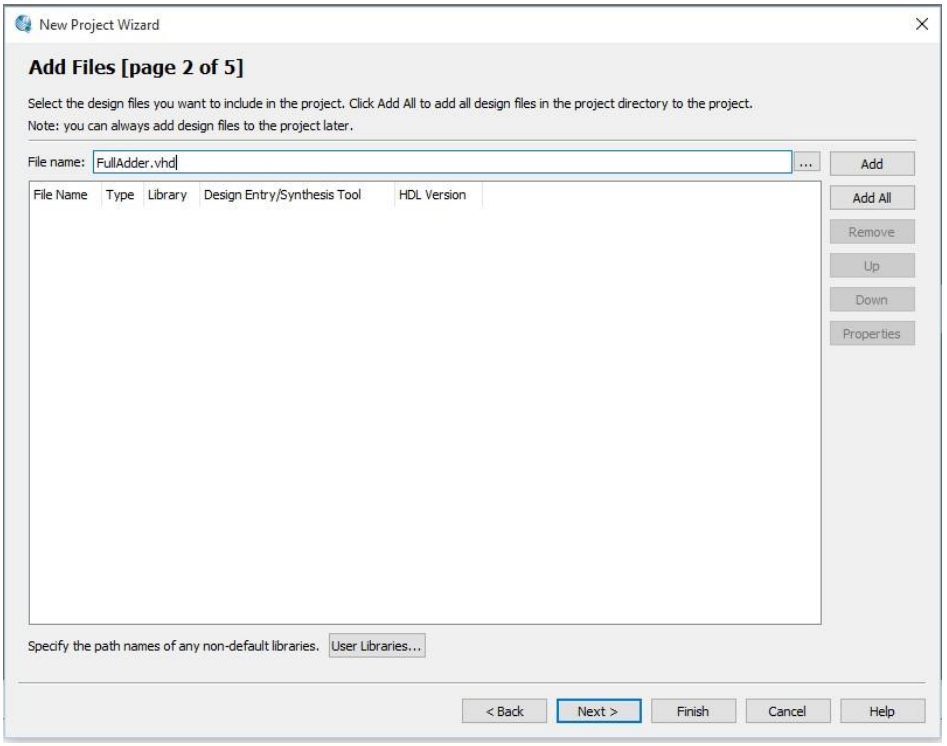


Figura 3.45 *Inclusão de arquivos existentes em um novo projeto VHDL.*

Fonte: *ALTERA (2008).*

Para a reutilização do componente somador completo de 2 bits é necessário criar um componente conforme ilustra o quadro 3.11. Para a correta utilização ainda é necessário que as portas de cada componente sejam mapeadas para as entradas e saídas do componente principal através de *PORT MAP*. Para cada componente é necessário a criação de um *label* denominado no quadro 3.11 por FA0, FA1, FA2 e FA3. Neste caso estamos utilizando o mapeamento direto, onde cada porta de entrada/saída é conectada explicitamente ao componente principal ou a sinais de transição devidamente especificados (co0, co1, co2 e co3).

Quadro 3.11 *Código 11 – Componente que reutiliza o somador completo de 2 bits.*

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

ENTITY FBRC IS
        PORT( AF, BF : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
                    SF : OUT STD_LOGIC_VECTOR(3 DOWNTO 0);
                    cfo : OUT STD_LOGIC );
END FBRC;

ARCHITECTURE Behavior OF FBRC IS
COMPONENT FullAdder
        PORT( a, b, ci : IN STD_LOGIC;
                  s, co : OUT STD_LOGIC );
END COMPONENT;
SIGNAL co0, co1, co2, co3 : STD_LOGIC := '0';
```

```
BEGIN
        FA0 : FullAdder PORT MAP ( a => AF(0), b => BF(0), ci => '0', s => SF(0),
        co => co0);
        FA1 : FullAdder PORT MAP ( a => AF(1), b => BF(1), ci => co0, s => SF(1),
        co => co1);
        FA2 : FullAdder PORT MAP ( a => AF(2), b => BF(2), ci => co1, s => SF(2),
        co => co2);
        FA3 : FullAdder PORT MAP ( a => AF(3), b => BF(3), ci => co2, s => SF(3),
        co => co3);
END Behavior;
```

Na tabela 3.13 são mostradas as conexões aos pinos da placa DE0 para que possam ser efetuados os testes deste somador completo de 4 bits. Os bits correspondentes ao número binário de entrada **A = a3a2a1a0** são conectados as chaves SW[3~0] e os bits correspondentes ao número binário **B = b3b2b1b0** estão conectados as chaves SW[7~4]. O resultado da soma, **cf0SF = A + B**, é mostrado através de LEDs, ou seja, o resultado da soma em binário **cfoSF = cf0sf3sf2sf1sf0** está conectado aos LEDS LEDG[4~0].

Tabela 3.13 *Pin Planner do somador completo de 4 bits.*

Sinal em VHDL	**Sinal na Placa DE0**	**Pinos na placa DE0**
A(0)	SW[0]	PIN_J6
A(1)	SW[1]	PIN_H5
A(2)	SW[2]	PIN_H6
A(3)	SW[3]	PIN_G4
B(0)	SW[4]	PIN_G5
B(1)	SW[5]	PIN_J7
B(2)	SW[6]	PIN_H7
B(3)	SW[7]	PIN_E3
SF(0)	LEDG[0]	PIN_J1
SF(1)	LEDG[1]	PIN_J2
SF(2)	LEDG[2]	PIN_J3
SF(3)	LEDG[3]	PIN_H1
cfo	LEDG[4]	PIN_F2

Na figura 3.46 abaixo está o resultado do código do componente que reutiliza o somador completo de 2 bits. Foram utilizados como entrada a soma 0010 + 0001.

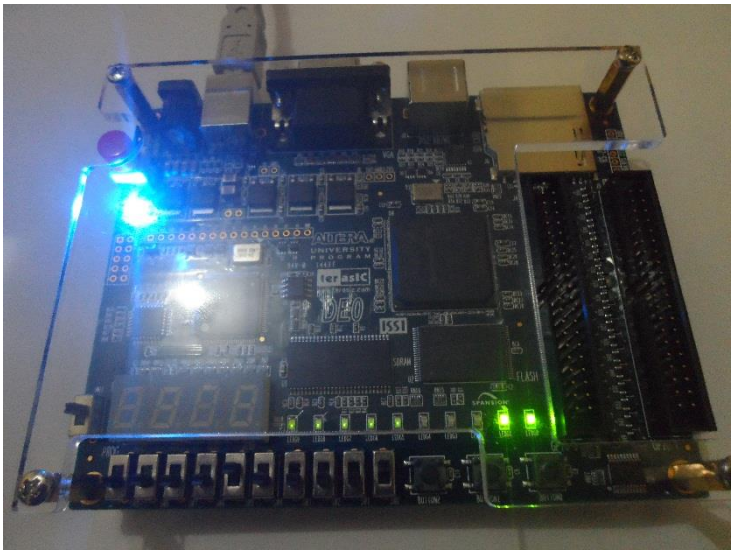


Figura 3.46 *Resultado Componente que reutiliza o somador completo de 2 bits.*

3.2.4 Parte IV - Conversor BCD de 4 bits.

Na parte II do Laboratório 02 foi discutida a conversão de números binários para decimal. Algumas vezes é usual ter circuitos que utilizam este método para representar números decimais onde cada dígito decimal é representado através de 4 bits. Este esquema é conhecido como representação *Binary Coded Decimal* (BCD). O código em VHDL que executa este circuito está representado no quadro 3.12, onde somente os números decimais de 0 a 15 são mostrados através da combinação de 4 bits.

Quadro 3.12 *Código 12 – Conversor BCD de 4 bits.*

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

ENTITY BinTo7Seg IS
        PORT( V   : IN  STD_LOGIC_VECTOR(3 DOWNTO 0);
                HEX0, HEX1 : OUT STD_LOGIC_VECTOR(6 DOWNTO 0));
END BinTo7Seg;

ARCHITECTURE Behavior OF BinTo7Seg IS
SIGNAL C    : STD_LOGIC_VECTOR(2 DOWNTO 0);
SIGNAL temp : STD_LOGIC;
SIGNAL M    : STD_LOGIC_VECTOR(3 DOWNTO 0); SIGNAL z    : STD_LOGIC ;
BEGIN
        temp <= V(3) AND (V(2) OR V(1));
        C(2) <= V(2) AND V(1);
        C(1) <= V(2) AND (NOT V(1));
        C(0) <= V(0) AND (V(2) OR V(1));
        M(0) <= ((NOT (temp) AND V(0)) OR (temp AND C(0)));
        M(1) <= ((NOT (temp) AND V(1)) OR (temp AND C(1)));
        M(2) <= ((NOT (temp) AND V(2)) OR (temp AND C(2)));
```

```
        M(3) <= ((NOT (temp) AND V(3)) OR (temp AND '0'));

        z <= temp;

        HEX0 <= "1000000" WHEN M = "0000" ELSE
                "1111001" WHEN M = "0001" ELSE
                "0100100" WHEN M = "0010" ELSE
                "0110000" WHEN M = "0011" ELSE
                "0011001" WHEN M = "0100" ELSE
                "0010010" WHEN M = "0101" ELSE
                "0000010" WHEN M = "0110" ELSE
                "1111000" WHEN M = "0111" ELSE
                "0000000" WHEN M = "1000" ELSE
                "0010000" WHEN M = "1001" ELSE
               "1111111";
        HEX1 <= "1111001" WHEN z = '1' ELSE
                "1000000";
END Behavior;
```

Para testar o código acima as conexões mostradas na tabela 3.14 foram utilizadas.

Tabela 3.14 *Pin Planner do conversor BCD de 4 bits.*

Sinal em VHDL	**Sinal na Placa**	**Pinos na placa**
V(0)	SW[0]	PIN_J6
V(1)	SW[1]	PIN_H5
V(2)	SW[2]	PIN_H6
V(3)	SW[3]	PIN_G4
HEX0(0)	HEX0[0]	PIN_E11
HEX0(2)	HEX0[1]	PIN_F11
HEX0(3)	HEX0[2]	PIN_H12
HEX0(4)	HEX0[3]	PIN_H13
HEX0(5)	HEX0[4]	PIN_G12
HEX0(6)	HEX0[5]	PIN_F12
HEX0(7)	HEX0[6]	PIN_F13
HEX1(0)	HEX1[0]	PIN_A13
HEX1(1)	HEX1[1]	PIN_B13
HEX1(2)	HEX1[2]	PIN_C13
HEX1(3)	HEX1[3]	PIN_A14
HEX1(4)	HEX1[4]	PIN_B14
HEX1(5)	HEX1[5]	PIN_E14
HEX1(6)	HEX1[6]	PIN_A15

O resultado do conversor BCD de 4 bits na FPGA é ilustrado na figura 3.47.

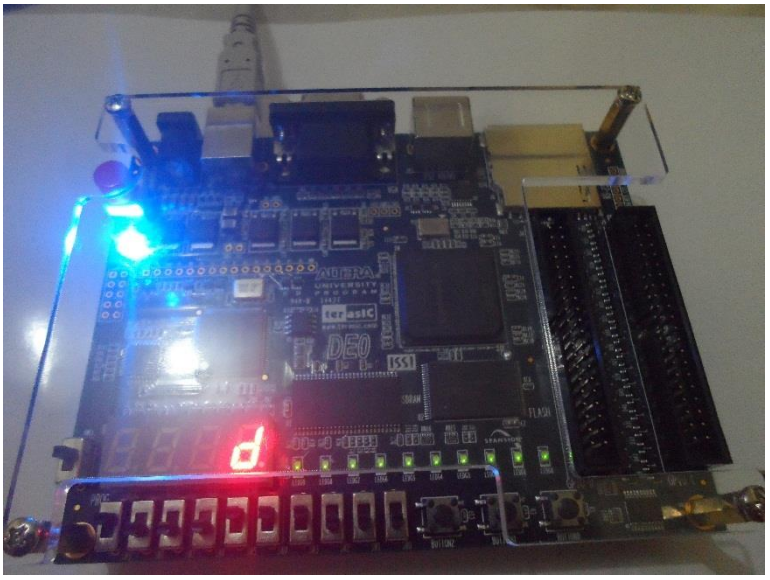


Figura 3.47 *Resultado na FPGA do conversor BCD de 4 bits.*

Tarefa 4: Projete um circuito em VHDL para ser executado na placa DE0 que some dois números binários com 4 bits cada e represente o seu resultado no formato BCD. Observe que o resultado da soma poderá ser um número de até 5 bits. Com isto o quadro 3.12 deverá ser alterado para representar este resultado.

Tarefa 5: Projeto um circuito combinacional que converta um número binário de 6 bits em um número decimal representado por 2 dígitos na representação BCD. Utilize as chaves SW[5~0] como entrada do número binário e os displays de 7 segmentos HEX1 e HEX0 para representar o número decimal em BCD. Implemente este circuito na placa DE0.

3.3 LABORATORIO 3.

Latches, flip-flops **e registradores**

O objetivo deste laboratório é estudar os componentes básicos de memória denominados de *latches, flip-flops* e registradores (ALTERA, 2010).

3.3.1 Parte I - *Latch RS*

Os circuitos integrados FPGA da Altera incluem em sua construção flip-flops para a elaboração de circuitos pelo usuário. Nesta etapa iremos verificar como estes elementos destinados ao armazenamento de bits podem ser criados de forma a não utilizar os disponíveis no circuito integrado FPGA.

A figura 3.48 ilustra um circuito do tipo *latch RS* e o código em VHDL que sintetiza este *latch* é mostrado no quadro 3.13.

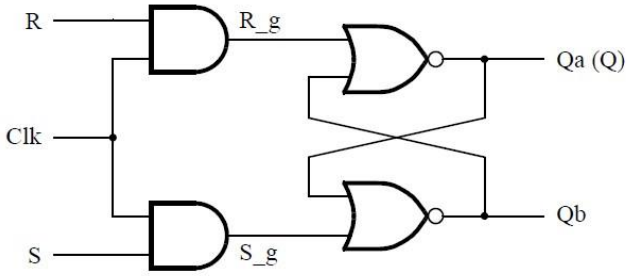


Figura 3.48 *Latch RS.*

Fonte: *ALTERA (2010).*

Quadro 3.13 *Código 13 – Conversor BCD de 4 bits.*

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

ENTITY Lab3Parte1 IS
        PORT( Clk, R, S : IN STD_LOGIC;
                Q   : OUT STD_LOGIC);
END Lab3Parte1;

ARCHITECTURE Estrutura OF  Lab3Parte1 IS
SIGNAL R_g, S_g, Qa, Qb : STD_LOGIC;
ATTRIBUTE keep : BOOLEAN;
ATTRIBUTE keep OF R_g, S_g, Qa, Qb :
SIGNAL IS true;


BEGIN
        R_g <= R AND Clk; S_g <= S AND Clk;
        Qa <= NOT (R_g OR Qb);
        Qb <= NOT (S_g OR Qa);
        Q <= Qa;
END Estrutura;
```

Para testar o código acima utilize a configuração dos pinos conforme mostrado na tabela 3.15.

Tabela 3.15 *Configuração dos pinos do latch RS.*

Sinal em VHDL	Sinal na Placa DE0	Pinos na placa DE0
R	SW[0]	PIN_J6
S	SW[1]	PIN_H5
Clk	BUTON[2]	PIN_F1
Q	LEDG[0]	PIN_J1

Para analisar o comportamento deste circuito complete a tabela-verdade abaixo:

Tabela 3.16 *Tabela verdade para análise.*

Clk	R	S	Q

0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

O Resultado do quadro 3.13 na placa DE0 é mostrado na figura 3.49.

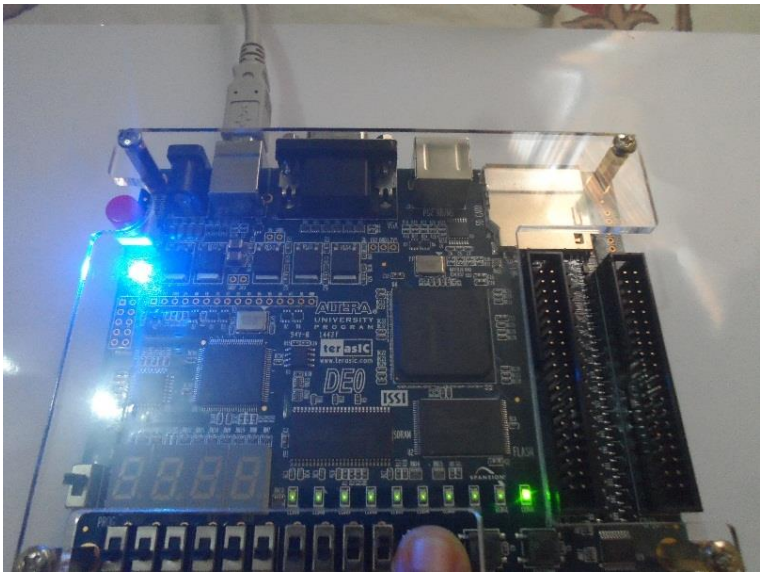


Figura 3.49 *Resultado latch RS na placa DE0.*

Tarefa 6: O que acontece quando as entradas R e S são modificadas? Quando o estado da saída Q não é alterado? Existe algum efeito memória?

3.3.2 Parte II - *Latch* D.

Como uma evolução do *latch* RS, a figura 16 mostra o *latch* D. Partindo do quadro 3.13, o quadro 3.14 implementa o *latch* D.

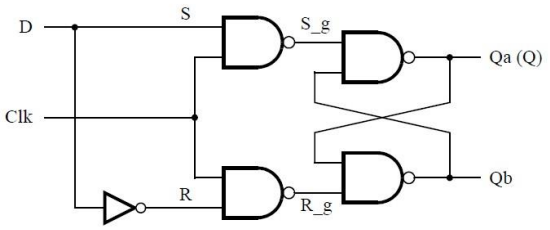


Figura 3.50 *Latch D a partir do latch RS.*

Fonte: *ALTERA (2010).*

Quadro 3.14 *Código 14 – Conversor BCD de 4 bits com Latch D.*

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

ENTITY Lab3Parte2 IS
        PORT(Clk D: IN STD_LOGIC;
                Q : OUT STD_LOGIC);
END Lab3Parte2;

ARCHITECTURE Estrutura OF Lab3Parte2 IS
SIGNAL R_g, S_g, Qa, Qb : STD_LOGIC;
ATTRIBUTE keep : BOOLEAN;
ATTRIBUTE keep OF R_g, S_g, Qa, Qb :
SIGNAL IS true;
 BEGIN
      R_g <= (NOT D) AND Clk;
      S_g <= D AND Clk;
      Qa <= NOT (R_g OR Qb);
      Qb <= NOT (S_g OR Qa);
      Q <= Qa;
END Estrutura;
```

Tabela 3.17 *Tabela verdade para análise do latch D.*

Clk	D	Q
0	0	
0	1	
1	0	
1	1	
↑	0	
↑	1	
↓	0	
↓	1	

O símbolo ↑ (transição positiva) na tabela-verdade indica que quando o BUTTON1 é pressionado, ou seja, uma transição do estado 0 para o estado 1. Da mesma forma o símbolo ↓ (transição negativa) indica a transição do estado 1 para o estado 0.

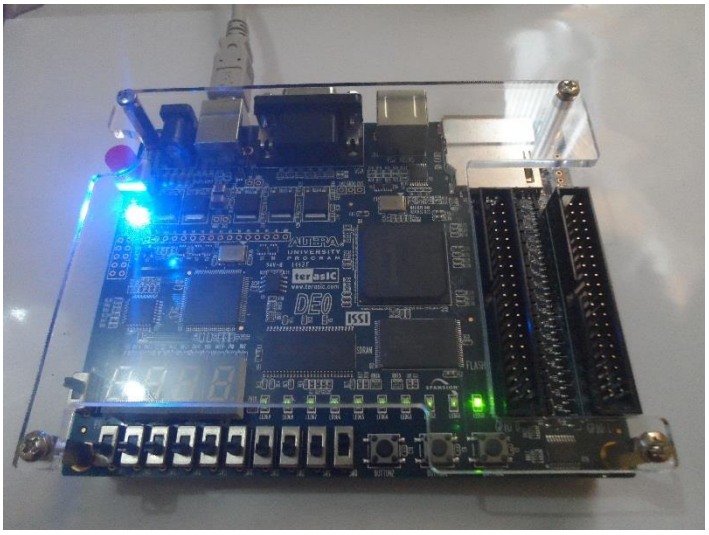


Figura 3.51 *Resultado na FPGA do lach D.*

Tarefa 7: Qual a diferença entre o *latch* RS e o *latch* D? Como as transições positivas e negativas são usadas para ativar o efeito memória no *latch* D?

3.3.3 Parte III - *Flip-flop*tipo D

O código abaixo representa um *Flip-flop*tipo D ativado na borda de descida que é representado pelo circuito integrado 7474 da figura 3.52. Caso se queira usar a ativação deste *Flip-flop*na borda de subida basta usar a função *rising_edge* no lugar da função *falling_edge*, conforme detalhado no código do componente.

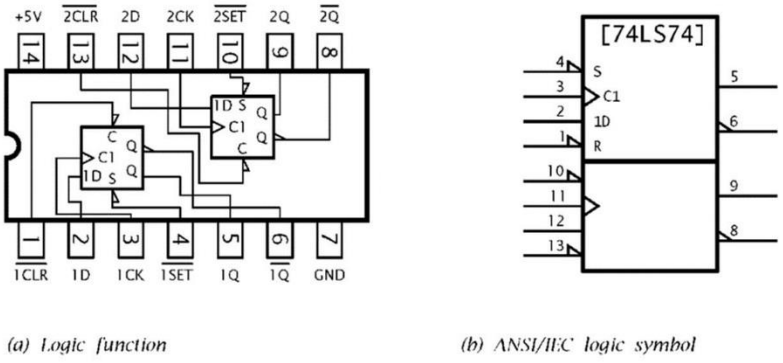


Figura 3.52 *Circuito Integrado 74LS74 que contém 2 flip-flops D.*
Fonte: *ALTERA (2010).*

Quadro 3.15 *Código 15 – Flip-floptipo D completo.*

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

ENTITY ffD IS
  port(  CLK, RST, PRE, CE, D : IN STD_LOGIC;
```

```
                    Q, nQ : OUT STD_LOGIC );
  END ENTITY ffD;

  ARCHITECTURE Behavior OF ffD IS SIGNAL temp : STD_LOGIC;
  BEGIN
PROCESS
(CLK) IS BEGIN
      -- IF rising_edge(CLK) THEN     -- Borda de subida
        IF falling_edge(CLK) THEN    -- Borda de descida
          IF (RST='1') THEN
             Q <= '0';
              nQ <= ‘1’;

           ELSIF (PRE = ‘1’) THEN
              Q <= '1';
              nQ <= '0';
           ELSIF (CE='1') THEN
              Q <= D;
            nQ <= NOT D:
           END IF;
         END IF;
END PROCESS;
END Behavior;
```

Os sinais RST (*reset*) e PRE (*preset*) controlam a saída do *Flip-flop*e quando estão ativos levam a saída Q do *flip-flop*D para 0 ou 1, respectivamente.

Um *PROCESS* em VHDL permite que o seu conteúdo seja executado de forma sequencial, já que no VHDL o processamento paralelo é nativo. O *PROCESS* é necessário para que as condições sejam testadas e as saídas do *flip-flop*tipo D sejam corretamente assinaladas, ou seja, tenham o seu valor corretamente atribuído.

A entrada CE (*chip enable*) vai habilitar o circuito para que ele responda as entradas. Caso o valor do CE seja 0 a saída não irá responde as alterações em D, mas somente as alterações em PRE e RST. Para que as saídas sejam alteradas é necessário que um pulso na entrada CLK seja aplicado.

Abaixo representamos os sinais conectados as entradas e saídas na placa DE0.

Tabela 3.18 *Configuração dos pinos do Flip-floptipo D completo.*

Sinal em VHDI	**Sinal na Placa DE0**	**Pinos na placa DE0**
D	SW[0]	PIN_J6
CLK	BUTON[2]	PIN_F1
PRE	SW[1]	PIN_H5
RST	SW[2]	PIN_H6
CE	SW[9]	PIN_D2
Q	LEDG[0]	PIN_J1

nQ	LEDG[1]	PIN_J2

Teste o circuito para todos os valores possíveis e verifique o comportamento do *flip-flop*D.

O resultado de um dos testes sugerido acima na placa DE0 é mostrado na figura 3.53.

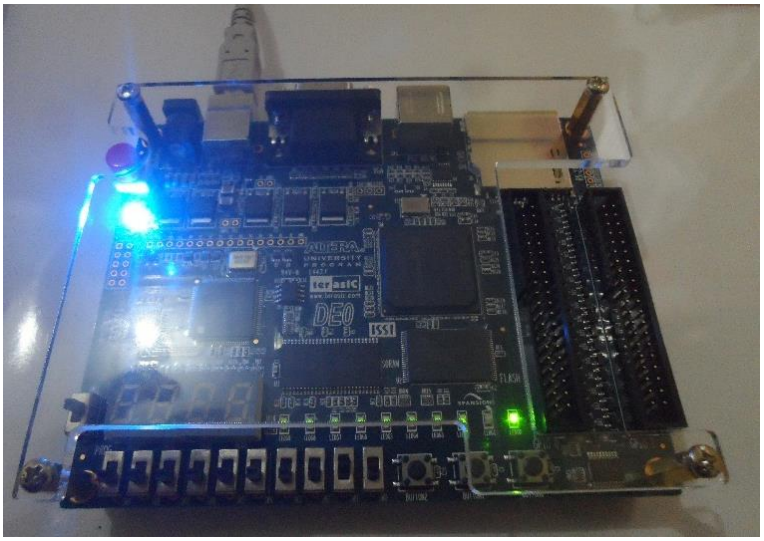


Figura 3.53 *Resultado na DE0 do Flip-floptipo D completo.*

3.3.4 Parte IV - Registrador paralelo de 4 bits com *flip-flop*tipo D

O *flip-flop* pode ser utilizado para executar várias funções que estão presentes dentro de uma CPU. Uma destas funções é armazenar dados para que instruções sejam guardados os dados a serem processados ou já processados possam ser disponibilizados.

A figura 3.54 representa um registrador de 4 bits utilizando *flip-flop*s D. Note que o armazenamento dos valores é de forma paralela. Como a entrada é paralela apenas um pulso de *clock* é necessário para que todos os bits sejam armazenados.

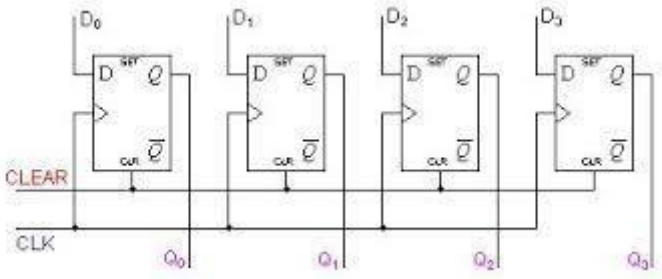


Figura 3.54 *Registrador de 4 bits paralelo utilizando flip-flopD.*

Fonte: *ALTERA (2010).*

O quadro 3.16 mostra o código em VHDL do registrador paralelo de 4 bits utilizando o componente *flip-flop*D completo que implementamos no item anterior.

Quadro 3.16 *Código 16 – Registrador paralelo de 4 bits com flip-floptipo D.*

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY reg4bit IS
        PORT (  regCLK, regRST, regCE : IN STD_LOGIC;
                Dreg : IN  STD_LOGIC_VECTOR(3 DOWNTO 0);
                Qreg : OUT STD_LOGIC_VECTOR(3 DOWNTO 0) );
END reg4bit;

ARCHITECTURE Behavior OF reg4bit IS
COMPONENT ffD
        PORT(   CLK, RST, PRE, CE, D : IN STD_LOGIC;
                Q, nQ : OUT STD_LOGIC );
END COMPONENT;
BEGIN
         FFD0 : ffD PORT MAP(D => Dreg(0), CLK => regCLK, RST => regRST, PRE
=> '0',
         Q => Qreg(0), CE => regCE );
        FFD1 : ffD PORT MAP(D => Dreg(1), CLK => regCLK, RST => regRST, PRE
  => '0',
         Q => Qreg(1), CE => regCE );
        FFD2 : ffD PORT MAP(D => Dreg(2), CLK => regCLK, RST => regRST, PRE
  => '0',
         Q => Qreg(2), CE => regCE );
        FFD3 : ffD PORT MAP(D => Dreg(3), CLK => regCLK, RST => regRST, PRE =>
        '0',
        Q =>Qreg(3), CE => regCE );
END Behavior;
```

Para o quadro 3.16, a configuração de pinos utilizados na placa DE-0 são:

Tabela 3.19 *Configuração dos pinos do Registrador paralelo de 4 bits com flip-floptipo D.*

Sinal em VHDI	**Sinal na Placa**	**Pinos na placa**
Dreg(0)	SW[0]	PIN_J
Dreg (1)	SW[1]	PIN_H5
Dreg (2)	SW[2]	PIN_H6
Dreg (3)	SW[3]	PIN_G4
regCLK	BUTTON[2]	PIN_F1
regRST	SW[8]	PIN_E4
regCE	SW[9]	PIN_D2
Qreg (0)	LEDG[0]	PIN_J
Qreg (1)	LEDG[1]	PIN_J
Qreg (2)	LEDG[2]	PIN_J
Qreg (3)	LEDG[3]	PIN_H1

Na figura 3.55 é mostrado o resultado do código do Registrador paralelo de 4 bits com *flip-flop*tipo D, na placa DE0.

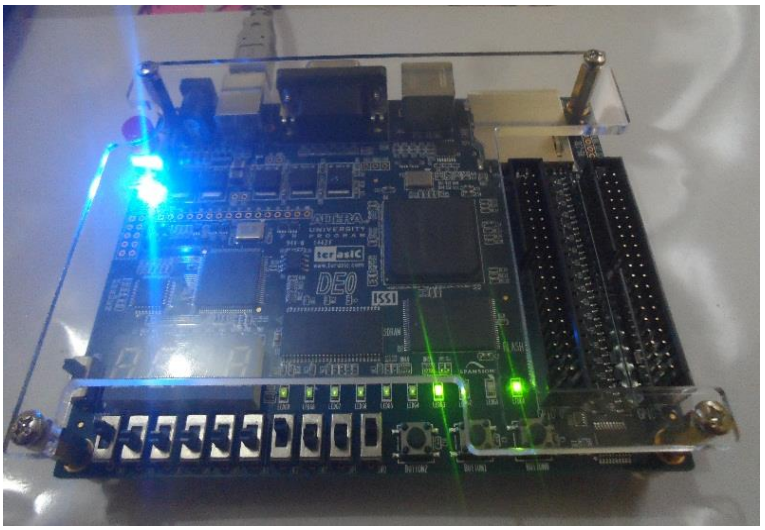


Figura 3.55 *Resultado na placa DE0 do Registrador paralelo de 4 bits com flip-floptipo D.*

Tarefa 8: Construa um registrador de 8 bits utilizando o *flip-flop*D.

Tarefa 9: Sem utilizar os *flip-flops* D, construa um circuito que tenha a mesma funcionalidade do registrador paralelo de 4 bits descrito nesta sessão.

3.1.5 Parte V - Registrador de deslocamento de 4 bits com *flip-flop* tipo D.

O registrador discutido anteriormente utiliza 4 linhas de entrada e 4 linhas de saída. Para o processamento interno ele é ideal, pois somente com um pulso de *Clock* podemos armazenar todos os bits de uma única vez.

Quando desejamos fazer conexões externas com outros periféricos, quanto mais linha de dados mais complexa e custosa se torna efetuar a conexão com registradores paralelos. Neste caso é comum utilizar os registradores com entrada serial onde são necessários n pulsos de *clock* para armazenar n bits. Tais registradores são conhecidos como *shift register* ou registradores de deslocamento, onde para cada pulso de *clock* os bits são deslocados para frente. A figura 3.56 ilustra um registrador de deslocamento de 4 bits.

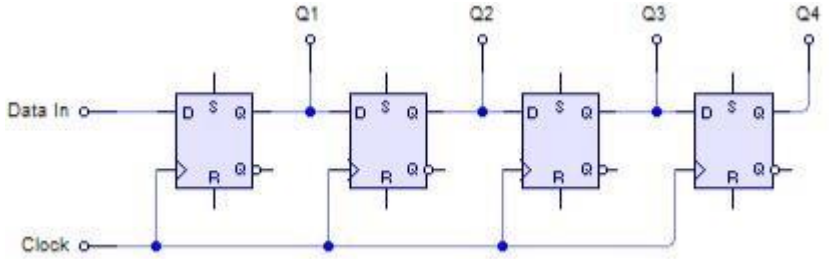


Figura 3.56 *Registrador de deslocamento de 4 bits.*

Fonte: *ALTERA (2010).*

O quadro 3.17 abaixo mostra o registrador de deslocamento em VHDL a partir do *flip-flop* D.

Quadro 3.17 *Código 17 – Registrador de deslocamento de 4 bits com flip-flop tipo D.*

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

ENTITY shift4bit IS
        PORT(regCLK, regRST, regCE, Dreg : IN STD_LOGIC;
                Qreg : OUT STD_LOGIC_VECTOR(3 DOWNTO 0) );
END ENTITY shift4bit;

ARCHITECTURE Behavior OF shift4bit IS
COMPONENT ffD
        PORT( CLK, RST, PRE, CE, D : IN STD_LOGIC;
                Q, nQ : OUT STD_LOGIC );
END COMPONENT;
SIGNAL t0, t1, t2, t3 : STD_LOGIC := '0';
  BEGIN
        FFD0 : ffD PORT MAP(D => Dreg, CLK => regCLK, RST => regRST, PRE
 => '0',
         Q => t0, CE => regCE );
        FFD1 : ffD PORT MAP(D => t0,  CLK => regCLK, RST => regRST, PRE =>
        '0',
        Q => t1, CE => regCE );
        FFD2 : ffD PORT MAP(D => t1,  CLK => regCLK, RST => regRST, PRE =>
        '0',
        Q => t2, CE => regCE );
        FFD3 : ffD PORT MAP(D => t2,  CLK => regCLK, RST => regRST, PRE =>
        '0',
        Q => t3, CE=> regCE );

        Qreg(0) <= t0;
        Qreg(1)     <=
        t1;
        Qreg(2) <= t2;
        Qreg(3)     <=
        t3;
  END Behavior;
```

Para implementar o quadro 3.17 as entradas e saídas utilizadas na placa DE-0 são:

Tabela 3.20 *Conexões para o Registrador de deslocamento de 4 bits.*

Sinal em	Sinal na Placa	Pinos na placa
Dreg(0)	SW[0]	PIN_J
regCLK	BUTTON[2]	PIN_F1
regRST	SW[8]	PIN_E4
regCE	SW[9]	PIN_D2
Qreg (0)	LEDG[0]	PIN_J
Qreg (1)	LEDG[1]	PIN_J
Qreg (2)	LEDG[2]	PIN_J
Qreg (3)	LEDG[3]	PIN_H1

O resultado da saída do Registrador de deslocamento de 4 bits é mostrado na figura 3.57 abaixo, nela todos os bits já foram deslocados.

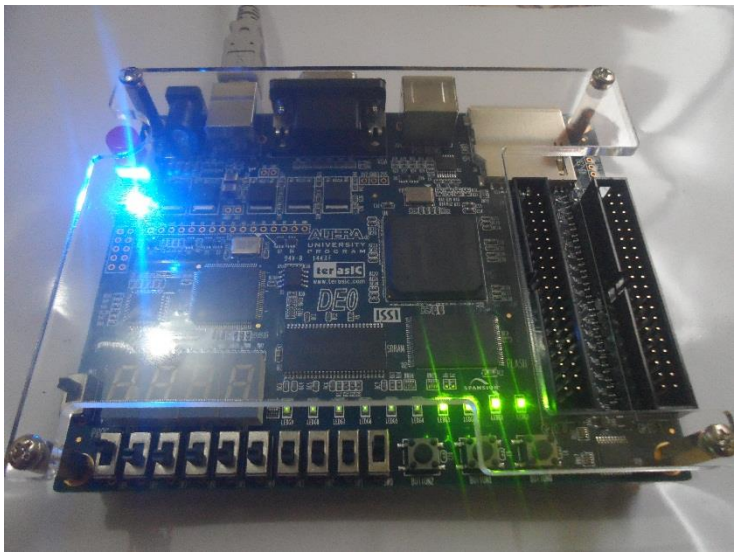


Figura 3.57 *Resultado na placa DE0 do Registrador de deslocamento de 4 bits.*

Tarefa 10: O código abaixo foi retirado de um site na internet1 e faz uma outra abordagem sobre como implementar um registrador de deslocamento. Quais as diferenças na estrutura do programa em VHDL destas duas formar de implementar o registrador de deslocamento?

Quadro 3.18 *Código 18 – Registrador de deslocamento de 8 bits.*

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

ENTITY shift_register_top IS
        PORT ( clk : IN  std_logic; d   : IN  std_logic;
                led : OUT std_logic_vector(7 downto 0));
END shift_register_top;
```

```
ARCHITECTURE behavioral OF shift_register_top IS
SIGNAL clock_div : std_logic_vector(4 downto 0);
SIGNAL shift_reg : std_logic_vector(7 downto 0) := x"00";
BEGIN

-- clock divider

PROCESS (clk)

BEGIN
      IF (clk'event AND clk = '1') THEN
          clock_div <= clock_div + '1';
      END IF;
END PROCESS;

-- shift register
PROCESS (clock_div(4))
BEGIN
      IF (clock_div(4)'event and clock_div(4) = '1')   THEN

               shift_reg(7)<= D;
              shift_reg(6) <= shift_reg(7);
              shift_reg(5) <= shift_reg(6);
              shift_reg(4) <= shift_reg(5);
              shift_reg(3) <= shift_reg(4);
              shift_reg(2) <= shift_reg(3);
              shift_reg(1) <= shift_reg(2);
             shift_reg(0)  <=   shift_reg(1);
     END IF;

END PROCESS;

-- hook up the shift register bits to the LEDs

   led <= shift_reg;

END Behavioral;
```

3.4 LABORATORIO 4.

Um dos componentes mais utilizados em circuitos sequencia são os contadores. Com eles podemos realizar diversas tarefas tais como o incremento e decremento para o controle de laço.

Neste laboratório iremos criar contadores baseados em *flip flops* D e utilizando somente códigos VHDL de nível mais alto. Também será visto a utilização do sinal de *clock* nativo da placa educacional DE0.

3.4.1 Parte I - Contador de 4 bits com *Flip flop* D

Existem diversos tipos de contadores que podem ser construídos com os mais variados tipos de *flip flops* (RS, T, D, JK, etc.). Na figura 3.58 é mostrado um contador de 4 bits utilizando *flip flops* tipo D. Observe que as saídas A0, A1, A2 e A3 são as saídas em binário relativa aos pulsos de *clock* da entrada CLK.

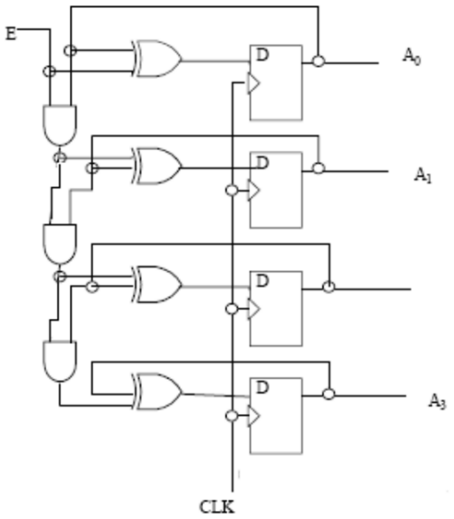


Figura 3.58 *Contador de 4 bits com flip flop D.*

O quadro 3.19 ilustra o código em VHDL para criar a entidade que representa o *flip-flop* D, aqui denominada de *flip-flop* D. Observe na figura 3.49 que as saídas dos *flip-flops* D são usadas também como entrada, já que existe uma retroalimentação para algumas portas lógicas. Para que possamos usar corretamente é necessário declarar estas portas como entrada e saída ao mesmo tempo utilizando para isto *INOUT*.

Quadro 3.19 *Código 19 – Flip flop D*

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

ENTITY flipflopd IS
        PORT( d, rst, clk : IN std_logic;
              q, qb : INOUT std_logic);

END flipflopd;

ARCHITECTURE behavioral OF flipflopd IS
BEGIN
    PROCESS (clk, rst)
    BEGIN
        IF (rst='0') THEN
            q<='0';
        ELSE
```

```
            IF(clk='1' and clk' event) THEN
                IF (d='0') THEN
                    q<='0';
                ELSE
                    q<='1';
                END IF;
            END IF;
        END PROCESS;
qb<=not q;
END behavioral;
```

O quadro 3.20 mostra o código em VHDL para o contador de 4 bits que utiliza como base *flip-flops* tipo D, implementado no quadro 3.17, e segue o esquema de ligação da figura 3.59. Neste código observa-se que os pinos dos *flip-flops* tipo D são mapeados de forma indireta seguindo a ordem de declaração do componente, ou seja, este tipo de mapeamento é implícito e é obrigatório seguir a ordem de declaração das portas efetuada no componente que será reutilizado. Caso não deseje efetuar nesta ordem é necessário utilizar => para que o pino seja direcionado corretamente.

Quadro 3.20 *Código 20 – Contador Up de 4 bits com Flip flop D.*

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

   ENTITY updff IS
          PORT ( rst,clk : IN std_logic;
                 q,qb    : INOUT std_logic_vector(3 downto 0));
END updff;

ARCHITECTURE Behavioral OF updff IS
COMPONENT flipflopD IS
          PORT( d,rst,clk: in std_logic;
                q,qb: inout std_logic);
END COMPONENT;
SIGNAL a,b,c,d,e,f,g,h,i,j : std_logic;
BEGIN
      a<=not q(0);
      D1 : flipflopD port map(a,rst,clk,q(0),qb(0);

      b<=(q(0) xor q(1));

      D2 : flipflopD port map(b,rst,clk,q(1),qb(1));
      c<= q(0) and q(1)and qb(2);
      d<= qb(0) or qb(1);
      e<= q(2) and d ;
```

```
        j<= c or e;

        D3 : flipflopD port map(j,rst,clk,q(2),qb(2));
        f<= qb(0) or qb(1) or qb(2);
        g<= f and q(3);
         h<= q(0) and q(1) and q(2) and qb(3);
          i<=g or h;

         D4 : flipflopD port map(i,rst,clk,q(3),qb(3));

END Behavioral;
```

Para testar o quadro 3.19 é necessário colocá-lo na placa DE0 com a configuração do quadro abaixo.

Tabela 3.21 *Conexões para o Contador Up de 4 bits com flip flop D*

Sinal em VHDL	Sinal na Placa DE0	Pinos na placa DE0
CLK	BUTTON[2]	PIN_F1
RST	SW[0]	PIN_J6
Q(0)	LEDG[0]	PIN_J1
Q(1)	LEDG[1]	PIN_J2
Q(2)	LEDG[2]	PIN_J3
Q(3)	LEDG[3]	PIN_H1

Para verificar o funcionamento do contador basta pressionar o botão BUTTON[2] para que os valores em binário sejam colocados nas saídas correspondentes, ou seja, LEDG[3] LEDG[2] LEDG[1] LEDG[0] <= Q(3) Q(2) Q(1) Q(0).

O resultado da verificação do código do Contador *Up* de 4 bits com *flip-flop* D está sendo mostrado na imagem 3.59 abaixo.

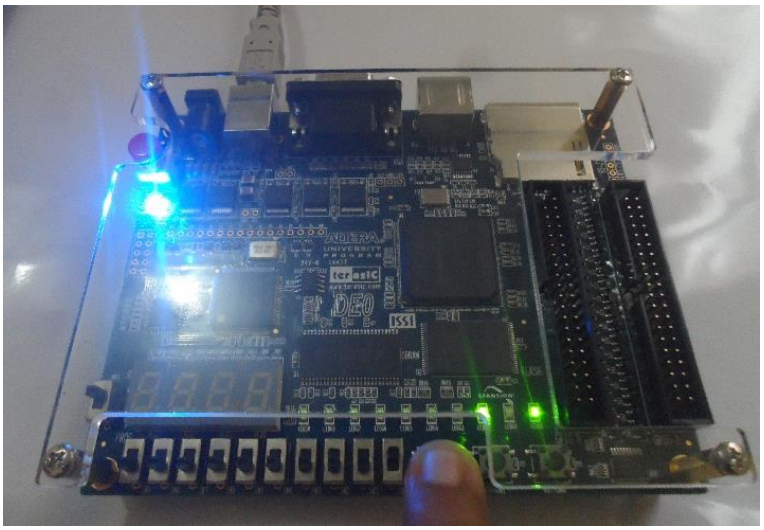


Figura 3.59 *Resultado do Contador Up de 4 bits com flip flop D.*

3.4.2 Parte II - Contador *Up* de 4 bits com *flip-flop* D alternativo

Outra forma de implementar circuitos sequencias é utilizar a própria estrutura da linguagem VHDL para construir componentes. No quadro 3.21 implementamos o mesmo contador de 4 bits da figura 3.59, porém aqui só se utiliza o código em VHDL e não mais o componente *flip-flop* tipo D.

Observe que a porta de saída Q é incrementa normalmente como uma linguagem de alto nível (Q <= Q + 1), apesar de esta estar declarada como um vetor de bits. Neste caso o próprio compilador irá fazer o incremento correto para construir o circuito digital sequencial equivalente.

Quadro 3.21 *Código 21 – Contador Up de 4 bits com flip-flop D alternativo.*

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

ENTITY cont4bitsup IS
        PORT ( rst,clk : IN std_logic;
               q,qb    : INOUT std_logic_vector(3 downto 0));
END cont4bitsup;

ARCHITECTURE behavioral OF cont4bitsup IS
BEGIN
        PROCESS (clk, rst)
        BEGIN
               IF ( rst = '1' ) THEN
                    q <= "0000";

               ELSE
                    IF( clk = '1' and clk'event) THEN
                         q <= q + 1;
                     END IF;
              END IF;
             END PROCESS;
             qb <= not q;
END behavioral;
```

Para testar o quadro 3.21 foram utilizadas as conexões mostradas no quadro abaixo.

Tabela 3.22 *Configuração dos pinos do Contador Up de 4 bits com flip flop D alternativo.*

Sinal em VHDL	Sinal na Placa DE0	Pinos na placa DE0
CLK	BUTTON[2]	PIN_F1
RST	SW[0]	PIN_J6
Q(0)	LEDG[0]	PIN_J1
Q(1)	LEDG[1]	PIN_J2

Q(2)	LEDG[2]	PIN_J3
Q(3)	LEDG[3]	PIN_H1

Com as conexões acima temos o resultado na placa DE0 mostrado na figura 3.60.

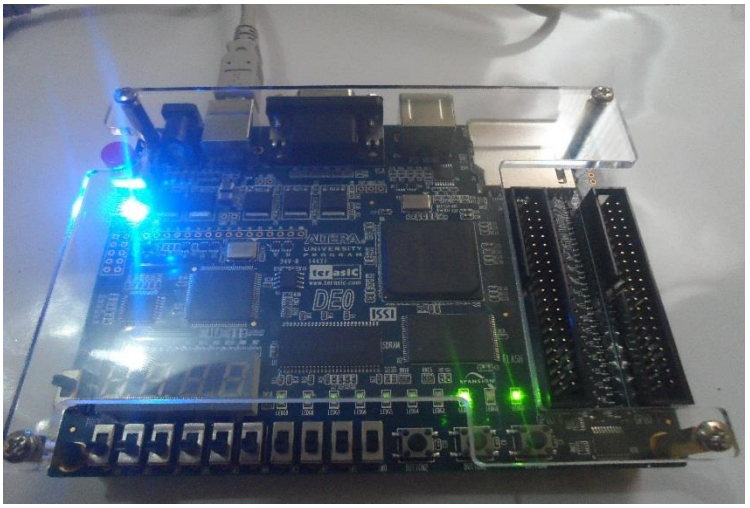


Figura 3.60 *Resultado na placa DE0 do código do Contador Up de 4 bits com flip flop D alternativo.*

Tarefa 11: Criar um contador de 8 bits que possa incrementar ou decrementar a saída.

Tarefa 12: Conectar o contador anterior a 3 displays de 7 segmentos.

3.4.3 Parte III - Contador de N bits.

A linguagem VHDL permite a generalização de alguns códigos. Uma necessidade é construir um componente que pode ser reutilizado e se necessário customizado. No caso de contadores é comum necessitar de contadores com saídas diversas: 4 bits, 8 bits, 12, bits, 16, bits, 24 bits, 32 bits, etc.

Um exemplo de um código VHDL "genérico" é ilustrado no quadro 3.22. Juntamente com a entidade é declarado um número n do tipo natural que pode ter qualquer valor. Neste código usa-se o valor 4 para n mantendo-se a compatibilidade com os códigos anteriores. Observe que a saída Q é declarada em função do valor de n. Para testar este código, utilize as mesmas conexões do código anterior.

Quadro 3.22 *Código 22 – Contador de N bits.*

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

ENTITY counterN IS
        GENERIC( n : natural := 4 );
         PORT ( clock     : IN std_logic;
```

```
                reset_n : IN std_logic;
                Q       : OUT std_logic_vector (n-1 downto 0) );
END ENTITY;
ARCHITECTURE rtl OF counterN IS
SIGNAL value : std_logic_vector(n-1 downto 0);
BEGIN
        PROCESS(clock, reset_n)
        BEGIN
              IF (reset_n = '0') THEN
                      value <= (OTHERS => '0');
           ELSIF ((clock'event) AND (clock = '1')) THEN
                    value <= value + 1;
    END IF;
         END PROCESS;
         Q <= value;
END rtl;
```

Para testar o quadro 3.22 foram usadas as mesmas conexões dos sinais VHDL com os sinais dos pinos da placa DE0, da tabela 3.22.

O resultado do código do contador de N bits está sendo mostrado na figura 3.61.

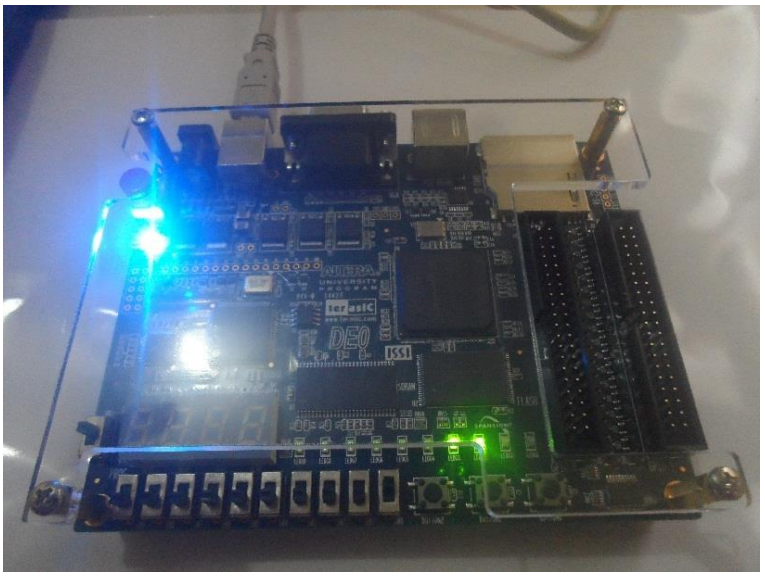


Figura 3.61 *Resultado na placa DE0 do Contador de N bits.*

3.3.4 Parte IV - Circuito sequencial de forma automática

Para que um circuito sequencial seja utilizado de forma automática é necessária a utilização de um sinal de *clock* com uma determinada frequência. No caso da placa DE0 existem dois pinos ligados ao FPGA que fornecem um sinal de *clock* de 50 MHz para que possamos utilizar em nossos projetos que abordam circuitos sequenciais. Estes sinais podem ser obtidos nos pinos B12 (CLOCK_50_2) ou G21 (CLOCK_50), conforme pode ser observado a partir da figura 3.62. Para testar um circuito automático pode-se utilizar o quadro 3.22, alterando a tabela 3.22 o pino da DE0 associado ao sinal VHDL do *clock* para B12 ou G21.

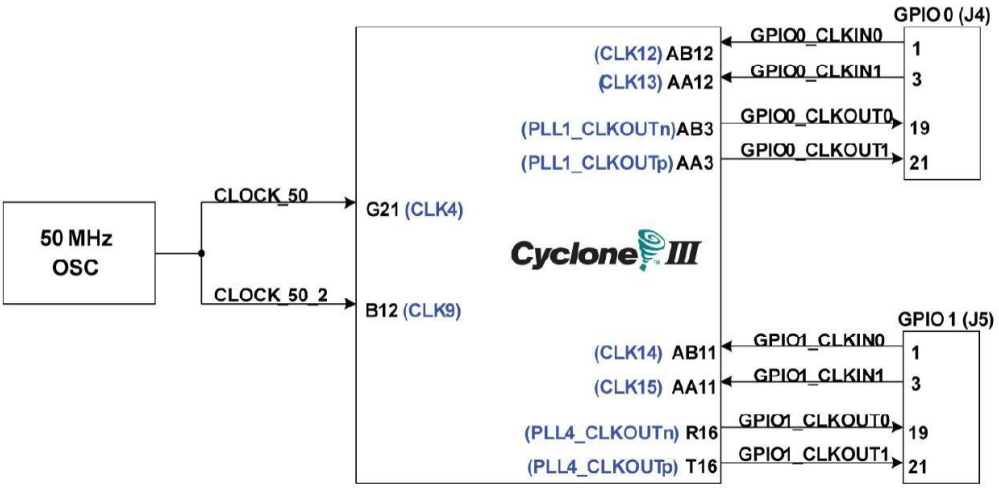


Figura 3.62 *Distribuição dos sinais de clock na placa DE0.*

Fonte: *TERASIC TECHNOLOGIES(2011).*

O resultado na placa DE0 utilizando o quadro 3.22 e o pino G21 da placa é mostrado na figura 3.63.

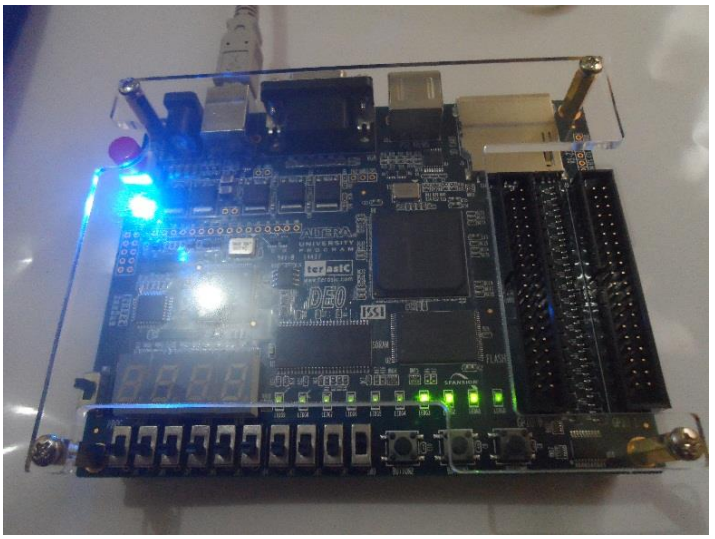


Figura 3.63 *Resultado na placa DE0 do clock automático.*

Tarefa 13: Crie um contador de 10 bits que possui como entrada o sinal de *CLOCK*_50 (PIN_G21) e como saídas os LEDs da placa DE0, LEDG[0~9]. Observe como as saídas se comportam.

Tarefa 14: Aumente o número de bits do contador acima para 28 e coloque as saídas Q(27) a Q(18) nos LEDs da placa DE0, LEDG[0~9]. Observe como as saídas se comportam. Qual a frequência de iluminação dos últimos LEDs?

3.5 LABORATORIO 5.

Nesta etapa vamos aprender a manusear diretamente o *hardware* ligado ao conector VGA de tal forma que possamos controlar a exibição na tela de um monitor.

3.5.1 Parte I – Padrão VGA

VESA Signal* 1280 x 1024 @ 60 Hz *timing

Vamos usar o padrão VGA para definição de 1280 x 1024 pixels. Abaixo são colocadas algumas referências sobre a temporização para obter o sincronismo adequado para esta configuração. Observe que a frequência dos *pixels* tem que ser de 108 MHz. Como não temos este padrão de *clock* iremos utilizar um componente chamado PLL para obter este sinal de *clock* a partir do sinal de *clock* nativo da placa DE0 que é de 50 MHz.

Tabela 3.23 *General timing*

Screen refresh rate	60 Hz
Vertical refresh	63.981042654028 kHz
Pixel freq.	108.0 MHz

Tabela 3.24 *Horizontal timing (line) Polarity of horizontal sync pulse is positive.*

Scanline part	Pixels	Time [µs]
Visible area	1280	11.851851851852
Front porch	48	0.44444444444444
Sync pulse	112	1.037037037037
Back porch	248	2.2962962962963
Whole line	1688	15.62962962963

Tabela 3.25 *Vertical timing (frame) Polarity of vertical sync pulse is positive.*

Frame part	Lines	Time [ms]
Visible area	1024	16.004740740741
Front porch	1	0.01562962962963

Sync pulse	3	0.046888888888889
Back porch	38	0.59392592592593
Whole frame	1066	16.661185185185

3.5.2 Parte II - VGA – Linhas cruzadas

Neste tópico vamos criar os componentes necessários para desenhar duas linhas na tela do monitor com a resolução anteriormente estipulada.

No quadro 3.23 possui o componente principal que será interligado aos pinos do nosso FPGA que será conectado ao adaptador VGA para ligação com um monitor. A figura 3.64 ilustra esta conexão.

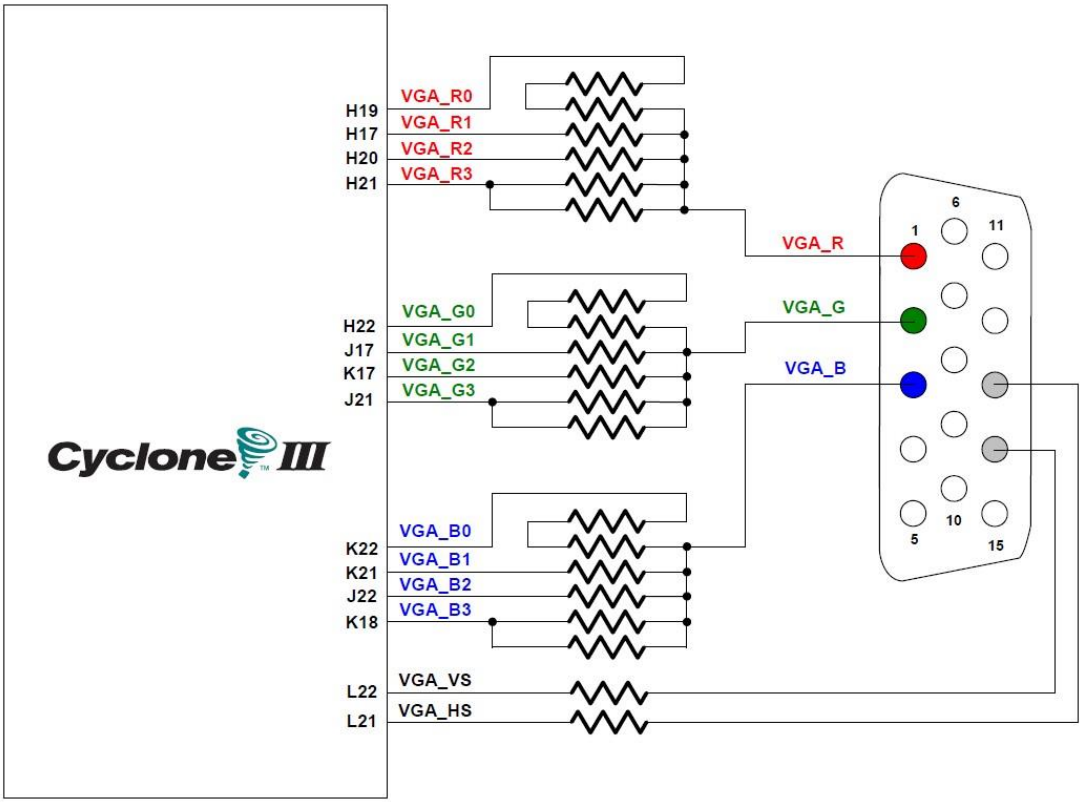


Figura 3.64 *Conexão VGA com a FPGA DE0*

Fonte: *TERASIC TECHNOLOGIES(2011).*

No quadro 3.24 usamos dois componentes: SYNC e PLL1. O componente SYNC está descrito no quadro 3.23 e possui a função de efetivamente desenhar os pixels na tela do monitor na área útil.

Observe que cada cor (*Red, Green e Blue*) são descritas por um *string* de 4 bits, sendo que a *string* "0000" indica ausência da cor e a *string* "1111" indica a presença total da cor. Com isto podemos fazer a combinação das cores primárias para obter cores diferentes.

Quadro 3.23 *Código 23 – SYNC.vhd linha cruzada*

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

ENTITY sync IS
        PORT ( clk : IN std_logic;
                hsync, vsync : OUt std_logic;
                r, g, b : OUT std_logic_vector(3 downto 0)
        );

END sync;

ARCHITECTURE main OF sync IS

SIGNAL hpos : integer RANGE 0 TO 1688 := 0;

SIGNAL vpos : integer RANGE 0 TO 1066 := 0;

BEGIN
    PROCESS (clk)
    BEGIN
        IF( clk'event and clk='1' ) THEN
            IF( hpos=1042 or vpos=554 ) THEN -- desenha o pixels.

                r <= (others=>'1');
                g <= (others=>'1');
                b <= (others=>'1');
            ELSE
                r <= (others=>'0');
                g <= (others=>'0');
                b <= (others=>'0');
            END IF;

            IF( hpos < 1688 ) THEN
                hpos <= hpos + 1;
            ELSE
                hpos <= 0;

                IF( vpos < 1066 ) THEN
                        vpos <= vpos + 1;
                ELSE
                        vpos <= 0;
                END IF;
            END IF;

            IF( hpos > 48 and hpos < 160 ) then
                hsync <= '0';
```

```
                        ELSE
                            hsync <= '1';
                        END IF;

                        IF( vpos > 0  AND vpos < 4 ) THEN
                            vsync <= '0';
                        ELSE
                            vsync <= '1';
                        END IF;

                        IF( (hpos > 0 AND hpos < 408) OR (vpos > 0 AND vpos
< 42) ) THEN
                            r <= (OTHERS=>'0');
                            g <= (OTHERS=>'0');
                            b <= (OTHERS=>'0');
                        END IF;
                END IF;
        END PROCESS;
END main;
```

O outro componente utilizado no quadro 3.24 é o PLL1. Este componente é criado através do aplicativo *Qsys* que faz parte da instalação do *Quartus* II. O PLL é um *Phase-Locked Loop* que neste caso é utilizado para obter o *clock* de 108 MHz, utilizado na sincronização do sinal de VGA, a partir do *clock* disponível na placa DE0 de 50 MHz, ou seja, iremos multiplicar o valor da frequência para obter um *clock* de valor mais alto que o original.

Para facilitar a utilização de alguns componentes, o *Qsys* apresenta um conjunto de componentes genéricos que podem ser adaptados para as necessidades da aplicação. Abaixo serão mostrados os passos para gerar o PLL e as principais telas do *Qsys* a cada passo.

Passo 1- No *Quartus* II vá em *Tools* > *Qsys*, para ter acesso a tela do *Qsys* e começar o processo de geração do PLL.

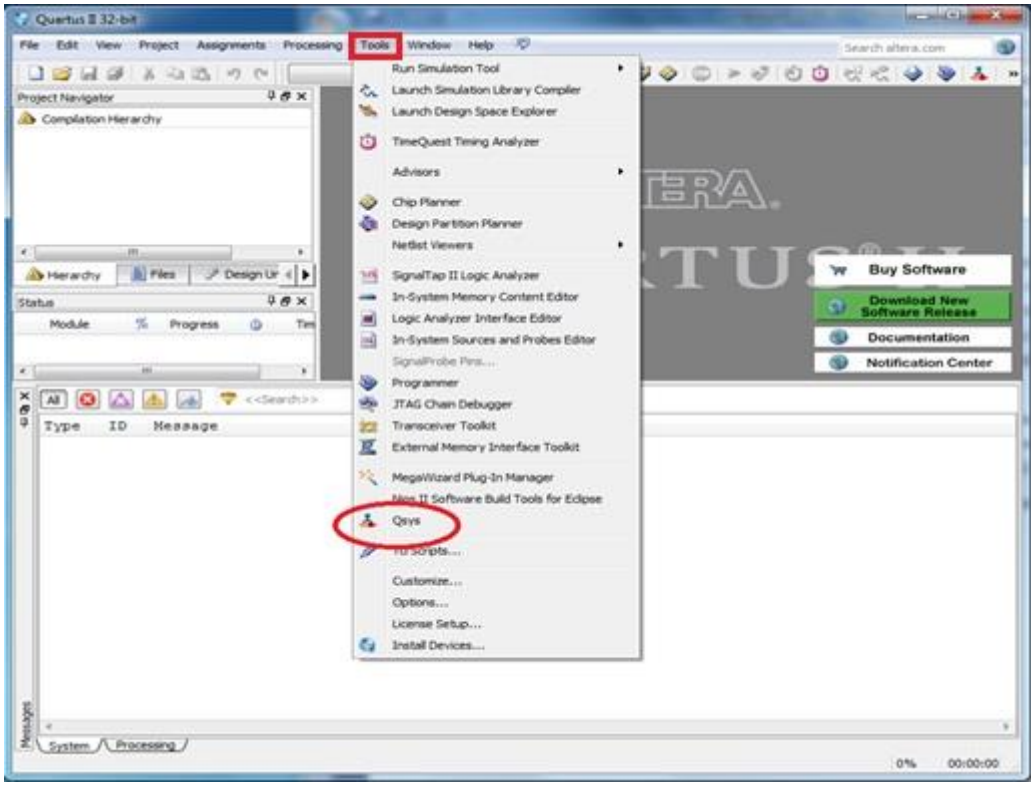


Figura 3.65 *Localização do Qsys no Quartus II.*

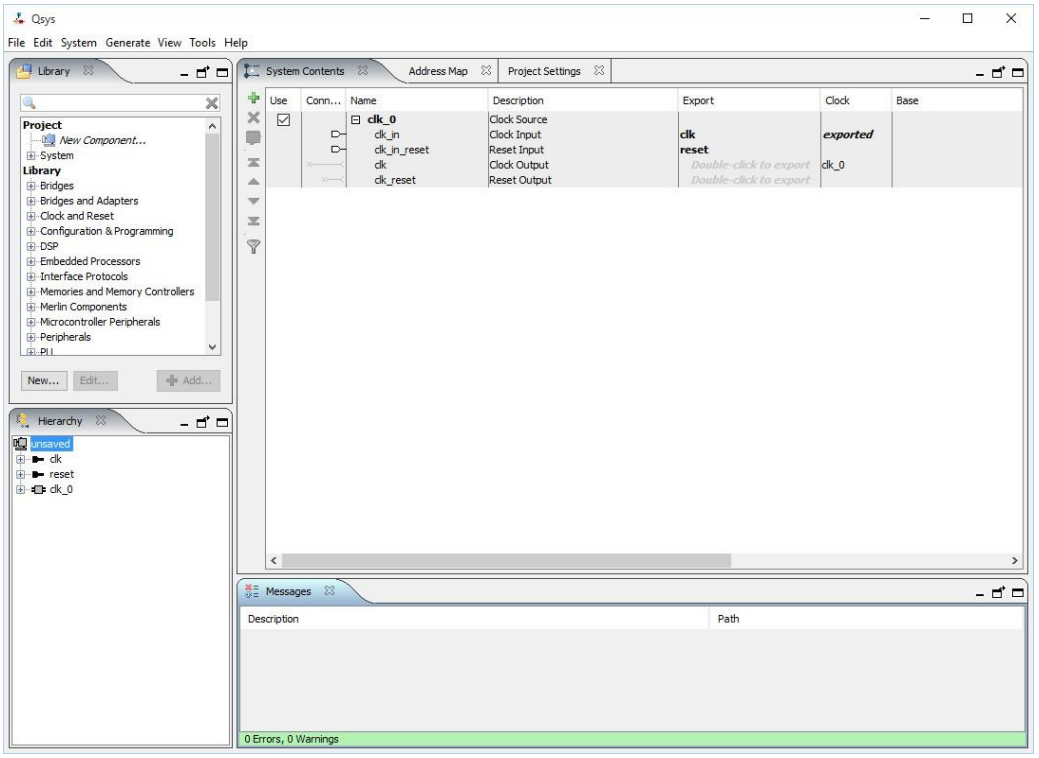


Figura 3.66 *Tela inicial o Qsys.*

Passo 2- Aparecerá a tela inicial do *Qsys*, nela remova os componentes padrão que aparece.

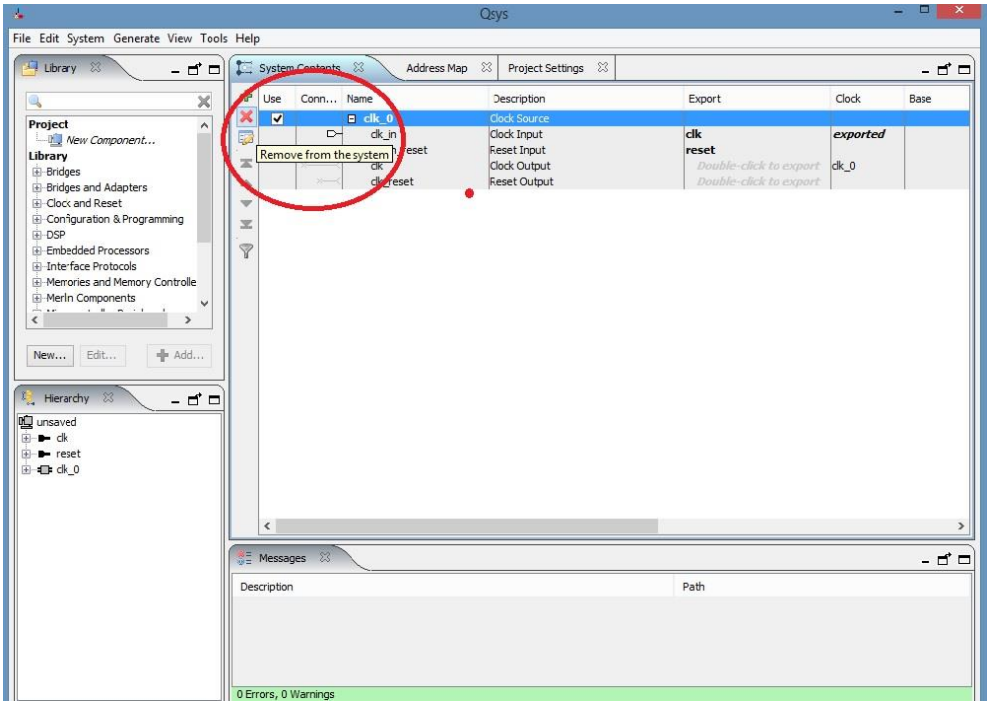


Figura 3.67 *Remove o componente padrão.*

Passo 3 - Adicione o componente *Avalon ALTPLL*. Vá em *PLL* > *Avalon ALTPLL* >*Add.*

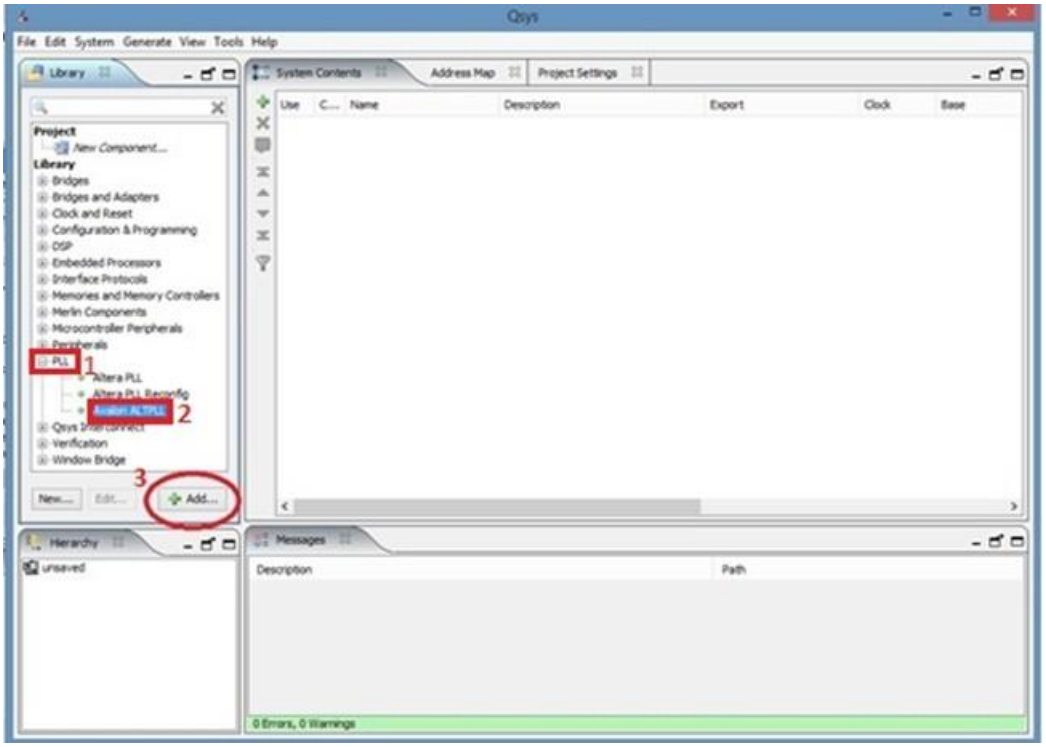


Figura 3.68 *Adição do componente Avalon ALTPLL.*

Passo 4- Ajuste o valor do *clock* para 50 MHz, que é o *clock* disponível da placa DE0.

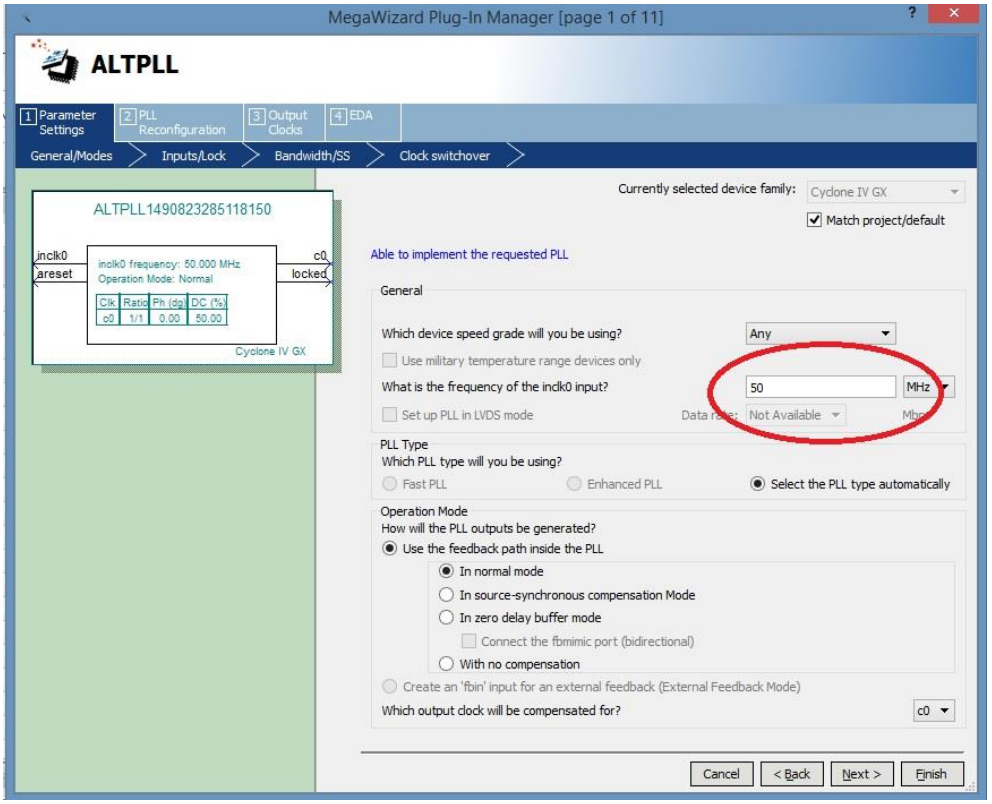


Figura 3.69 *Ajuste do clock.*

Passo 5- Ao acessa a tela retire a seleção da área mostrada na figura 3.70, para limpar as entradas e saídas desnecessárias.

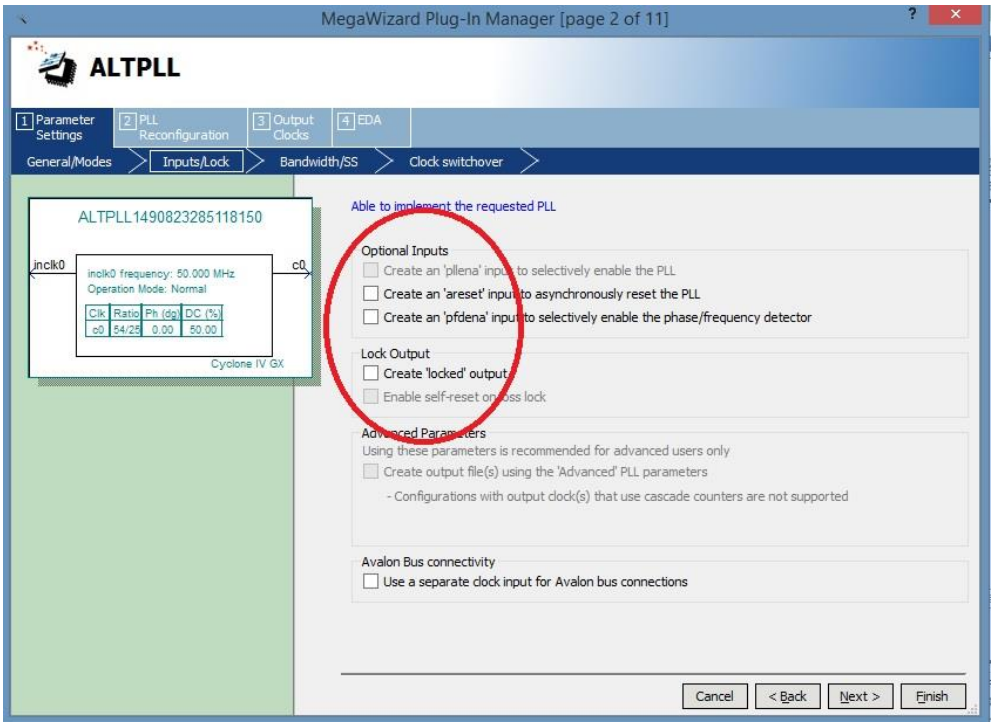


Figura 3.70 *Limpar as entradas/saídas desnecessárias.*

Passo 6 - Ajuste a saída do *clock* para 108 MHz, e depois finalize o *ALTPLL*.

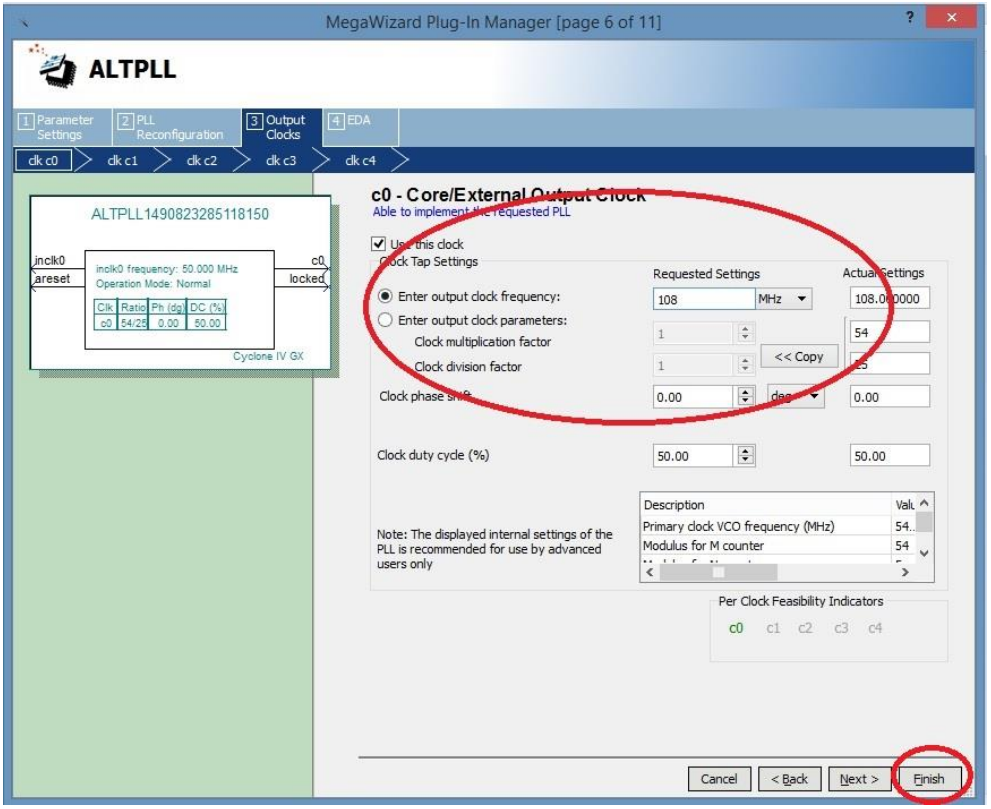


Figura 3.71 *Ajuste da saída do clock.*

Passo 7- depois de finalizar o *ALTPLL*, aparecerá novamente a tela inicial do *QSYS* (Figura 3.71), altere o nome das entradas e saídas para *clk_in*, reset e *clk_out*, respectivamente como mostrado na figura 3.72.

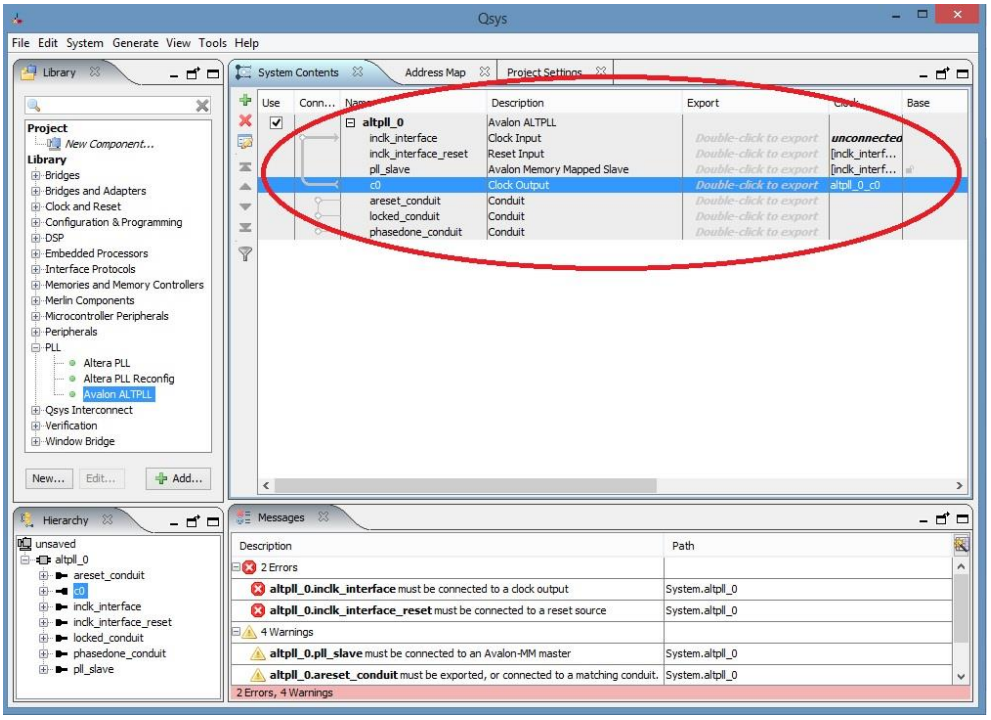


Figura 3.72 *Componente padrão customizado com os valores selecionados.*

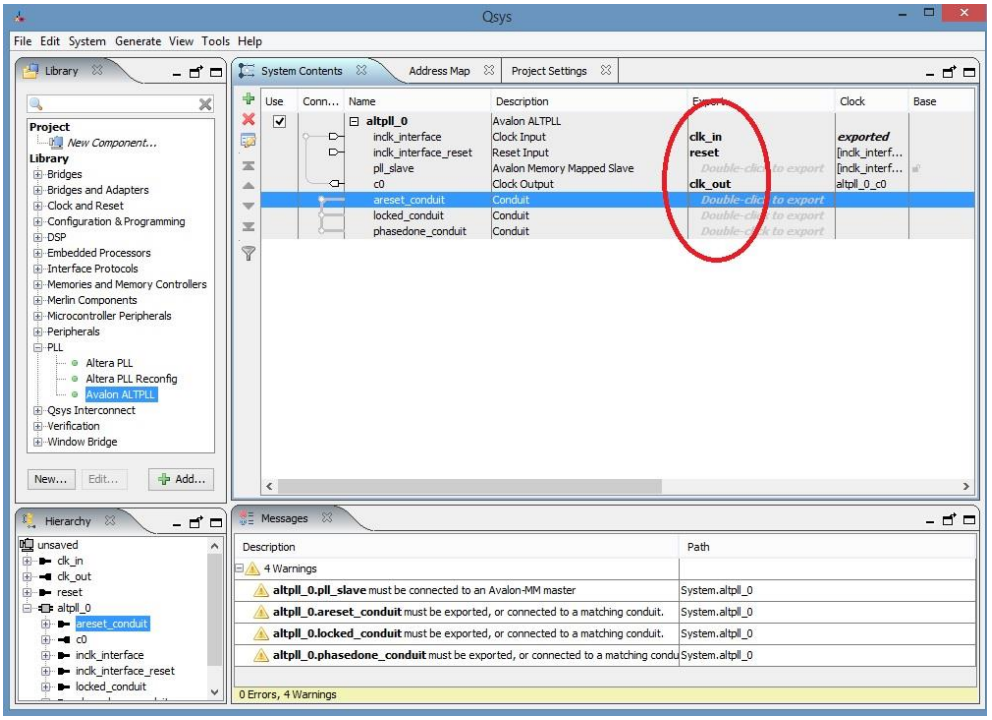


Figura 3.73 *Alterar nome das entradas / saídas do componente.*

Passo 8- Para visualizar o código do PLL em VHDL, clique em *Generate* > HDL *Example*.

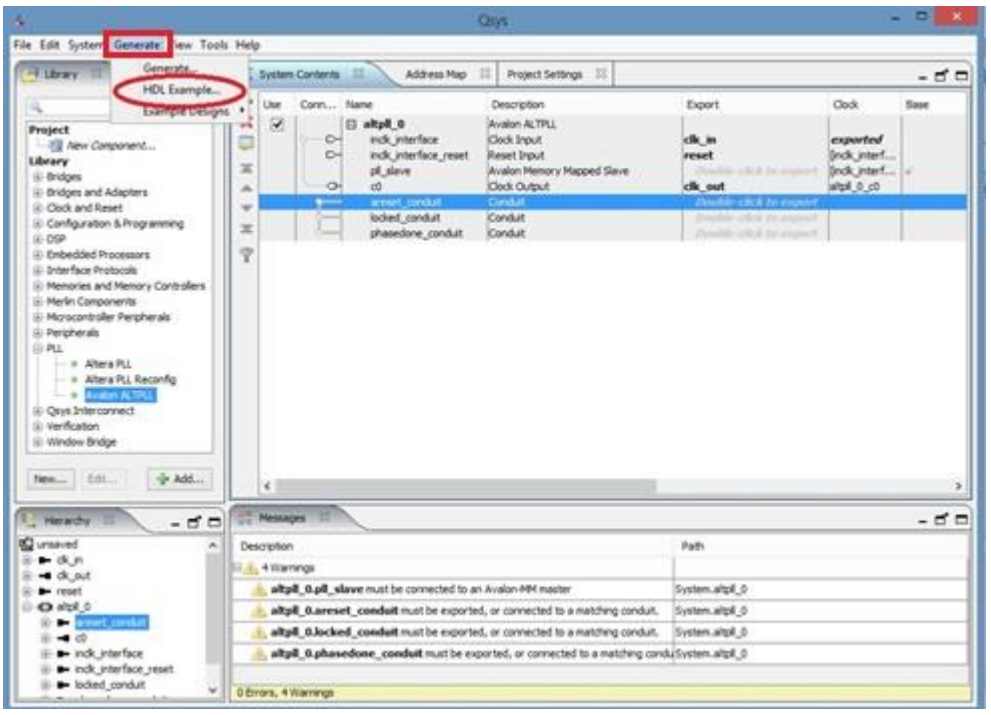


Figura 3.74 *Visualizar o código em VHDL*

Passo 9 – Nessa tela como mostra a figura 3.75, em HDL *Language* e selecione VHDL, e o código será mostrado.

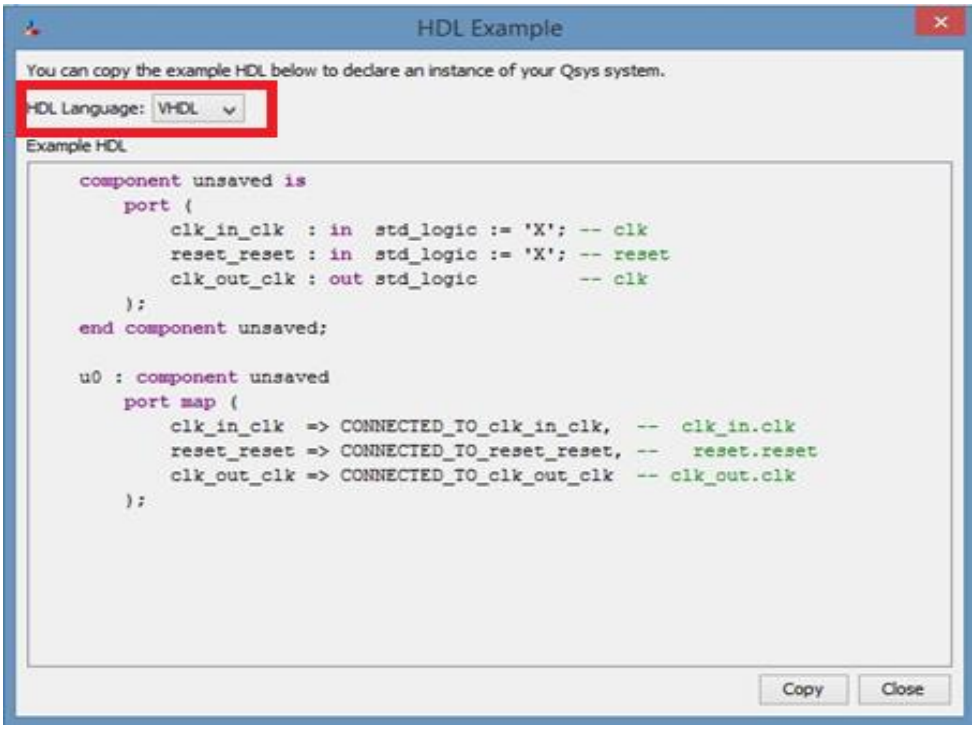


Figura 3.75 *Código em VHDL do componente PLL gerado.*

Após os passos que foram demonstrados nas figuras anteriores, vamos salvar o arquivo com o nome de PLL1 e gerar o componente para ser utilizado no projeto, como mostra as figuras 3.76 e 3.77 dos passos seguintes.

Passo 10 - Clique novamente em *Generate*, só que agora clique em *Generate*.

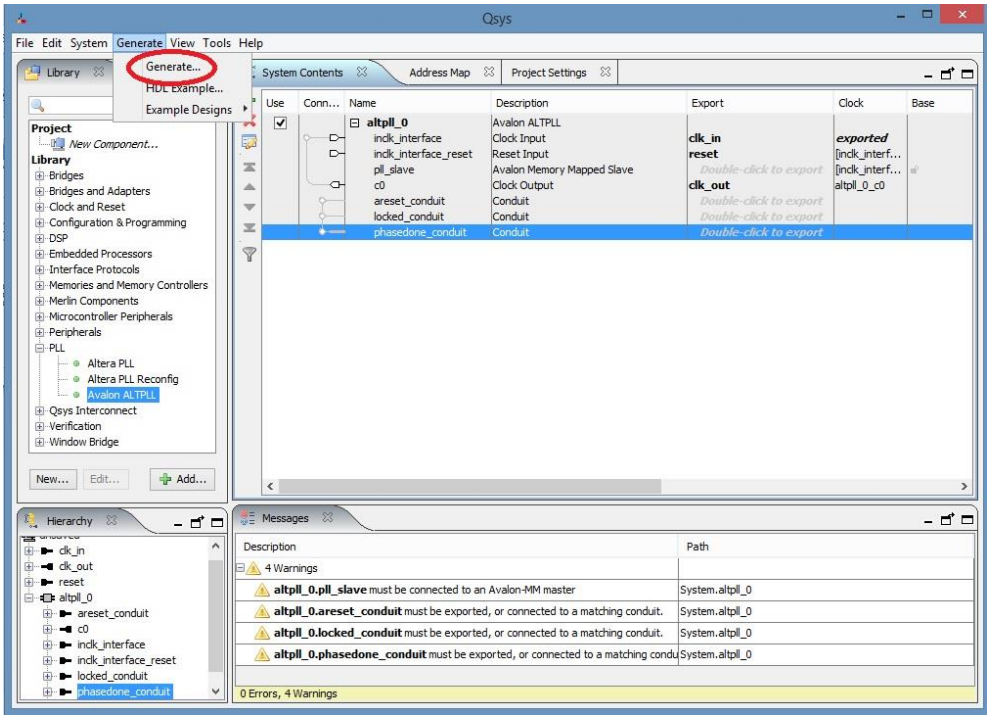


Figura 3.76 *Gerar o componente PLL.*

Passo 11 – Salve o código PLL gerado, clicando em *Generate*.

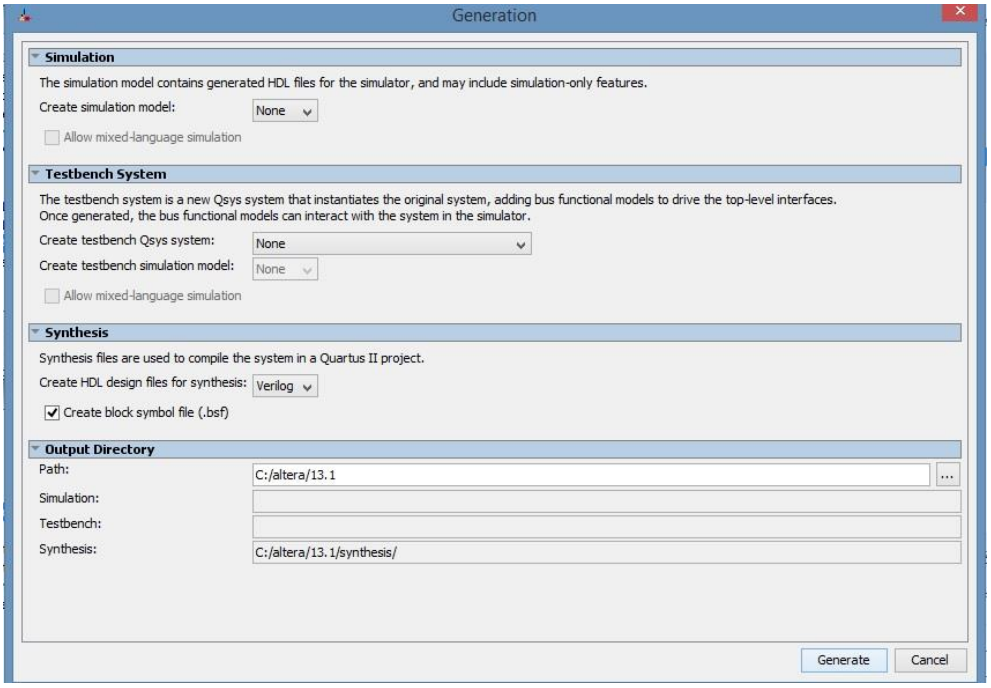


Figura 3.77 *Clicar no botão Generate para gravar os arquivos no código*.

Nos passos a seguir mostrados pelas figuras 3.78, 3.79 e 3.80, mostramos adição do componente ao projeto do VGA.vhd, a seleção do arquivo correto: PLL1.qip, observar a pasta onde o arquivo está armazenado, e depois verificar se ele foi adicionado ao projeto corretamente.

Passo 12- Para adicionar o arquivo ao projeto, clique com o botão direito do mouse encima de file, e depois com o botão esquerdo em *Add/Remove Files in Project.*

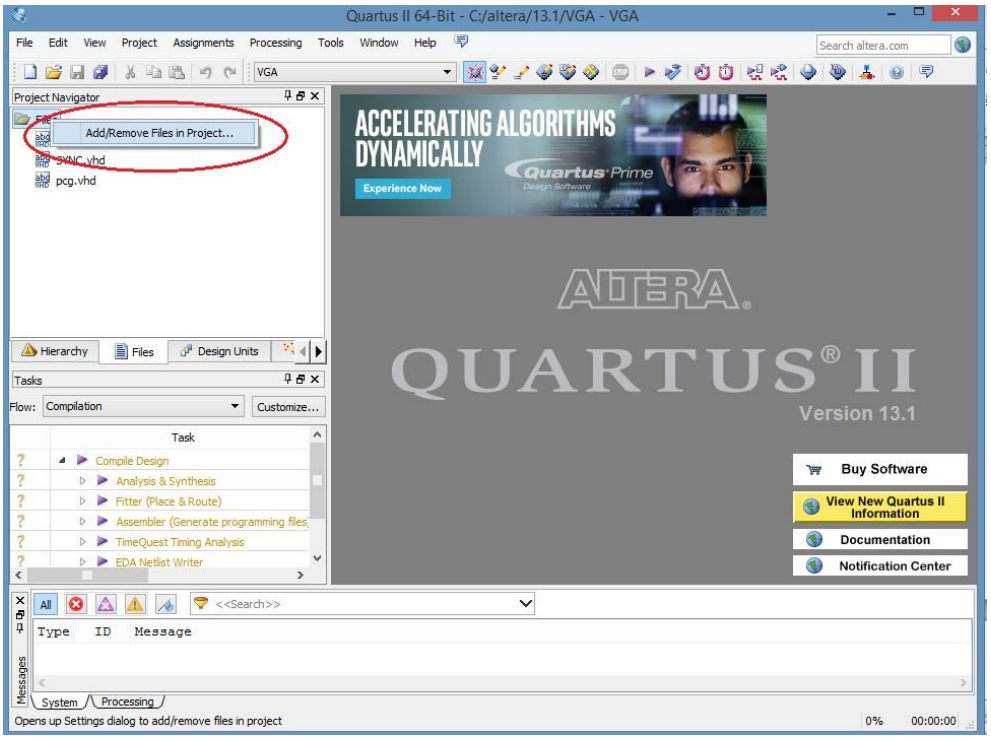


Figura 3.78 *Adição do arquivo ao projeto*

Passo 13- Agora com a pasta localizada, selecione o arquivo PLL1.qip.

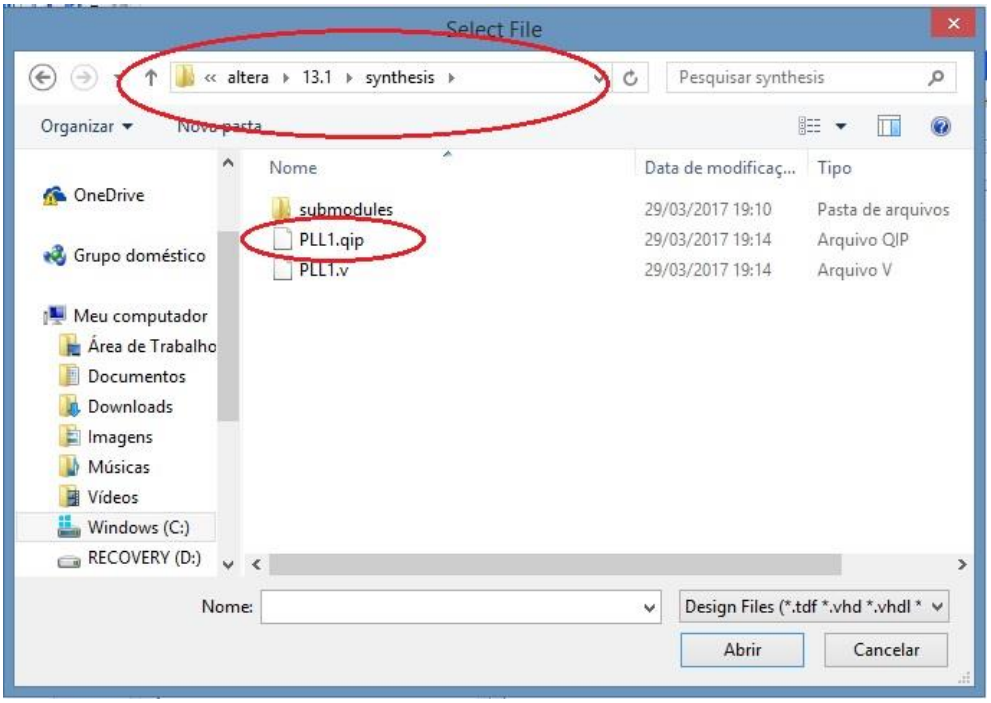


Figura 3.79 *Seleção do arquivo correto.*

Passo 14- verifique se o arquivo certo foi inserido

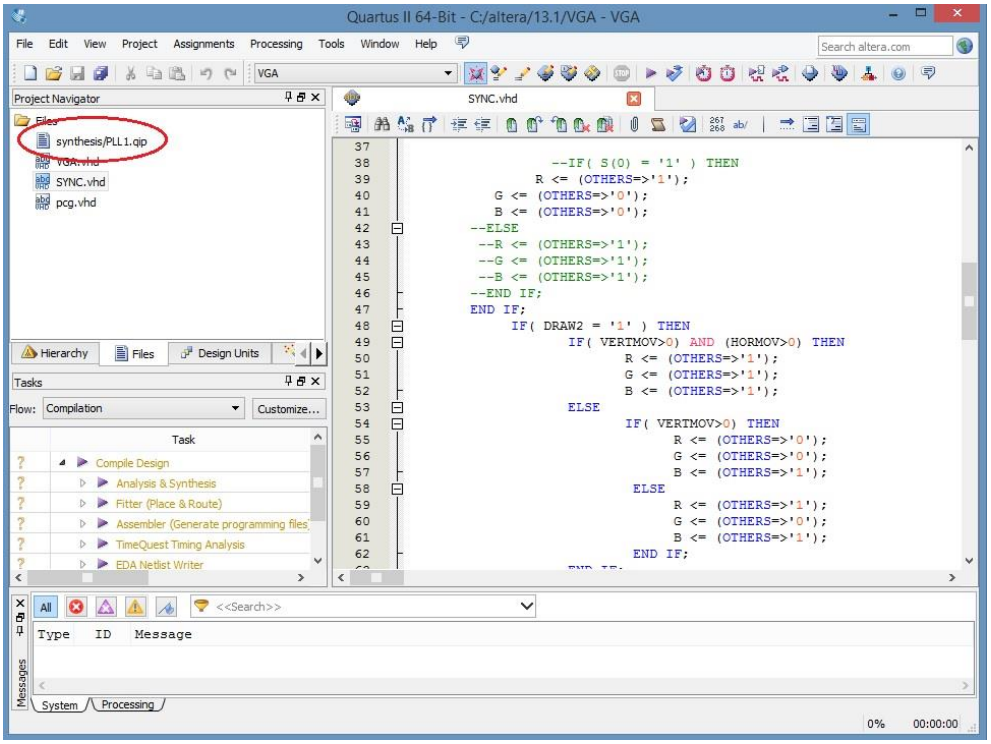


Figura 3.80 *Verifica se o arquivo foi adicionado corretamente.*

A seguir temos o quadro 3.24 já com os componentes adicionados.

Quadro 3.24 *Código 24 – VGA.vhd. linha cruzada*

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

```
USE IEEE.NUMERIC_STD.ALL;

ENTITY vga IS
        PORT (clock_24 : IN std_logic_vector(1 downto 0);
                vga_hs, vga_vs : OUT std_logic;
                vga_r, vga_g, vga_b : OUT std_logic_vector(3 downto 0));
END vga;

ARCHITECTURE main OF vga IS
SIGNAL vgaclk, reset : std_logic:='0';
COMPONENT pll1 IS
        PORT(clk_in_clk : IN std_logic := 'x'; -- clk
                reset_reset : IN std_logic := 'x'; -- reset
                clk_out_clk : OUT std_logic -- clk
        );
END COMPONENT pll1;

COMPONENT sync IS
         PORT( clk : IN  std_logic;
                hsync, vsync : OUT std_logic;
                r, g, b : OUT std_logic_vector(3 downto 0)
          );
END COMPONENT sync;
BEGIN
    c1 : sync PORT MAP( vgaclk, vga_hs, vga_vs, vga_r, vga_g, vga_b);
   c2 : pll1 PORT MAP(clock_24(0), reset, vgaclk);
END main;
```

Para o teste do código VGA utilizamos as conexões dos sinais do código com os pinos da DE0 descritos na tabela 3.26.

Tabela 3.26 *Conexões VGA com os pinos da placa*

Sinal VHDL	**Pino**
*CLOCK*_24[0]	PIN_G21
VGA_B[3]	PIN_K18
VGA_B[2]	PIN_J22
VGA_B[1]	PIN_K21
VGA_B[0]	PIN_K22
VGA_G[3]	PIN_J21
VGA_G[2]	PIN_K17
VGA_G[1]	PIN_J17
VGA_G[0]	PIN_H22
VGA_R[3]	PIN_H21
VGA_R[2]	PIN_H20
VGA_R[1]	PIN_H17
VGA_R[0]	PIN_H19
VGA_HS	PIN_L21
VGA_VS	PIN_L22

Ao adicionarmos os pinos de conexão com a FPGA, e recompilarmos o código, vamos nos deparar com um erro no pino K22. Isso acontece porque na placa DE0 o pino K22 está programado para atuar como uma porta especial e quando usamos os pinos que estão conectados a porta VGA é necessário configurá-los para que eles sejam pinos de entrada e saída (Regular I/O).

Para isso os passos abaixo descritos nas figuras devem ser efetuados para que haja a solução do problema.

Passo 1- Clique em *Assignments* > *Device*. Para começar as configurações dos pinos.

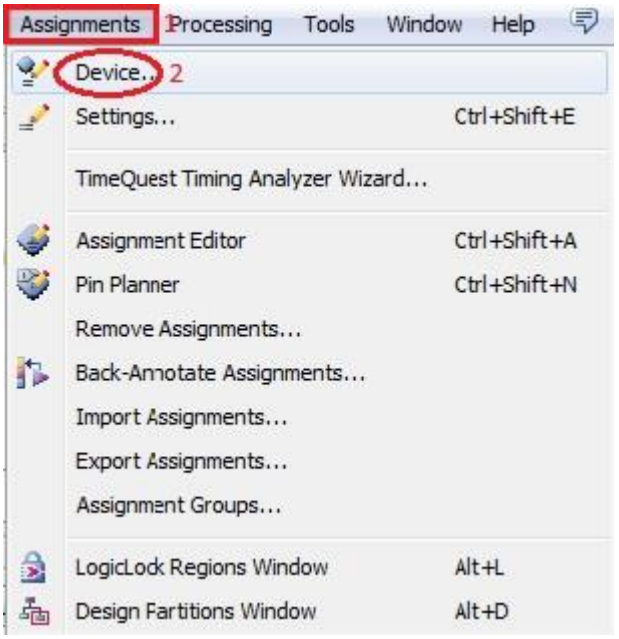


Figura 3.81 *Inicio da configuração*

Passo 2 – Dentro da tela do *Device*, clique em *Device and Pin Options*, para selecionar as opções de dispositivo e pinos.

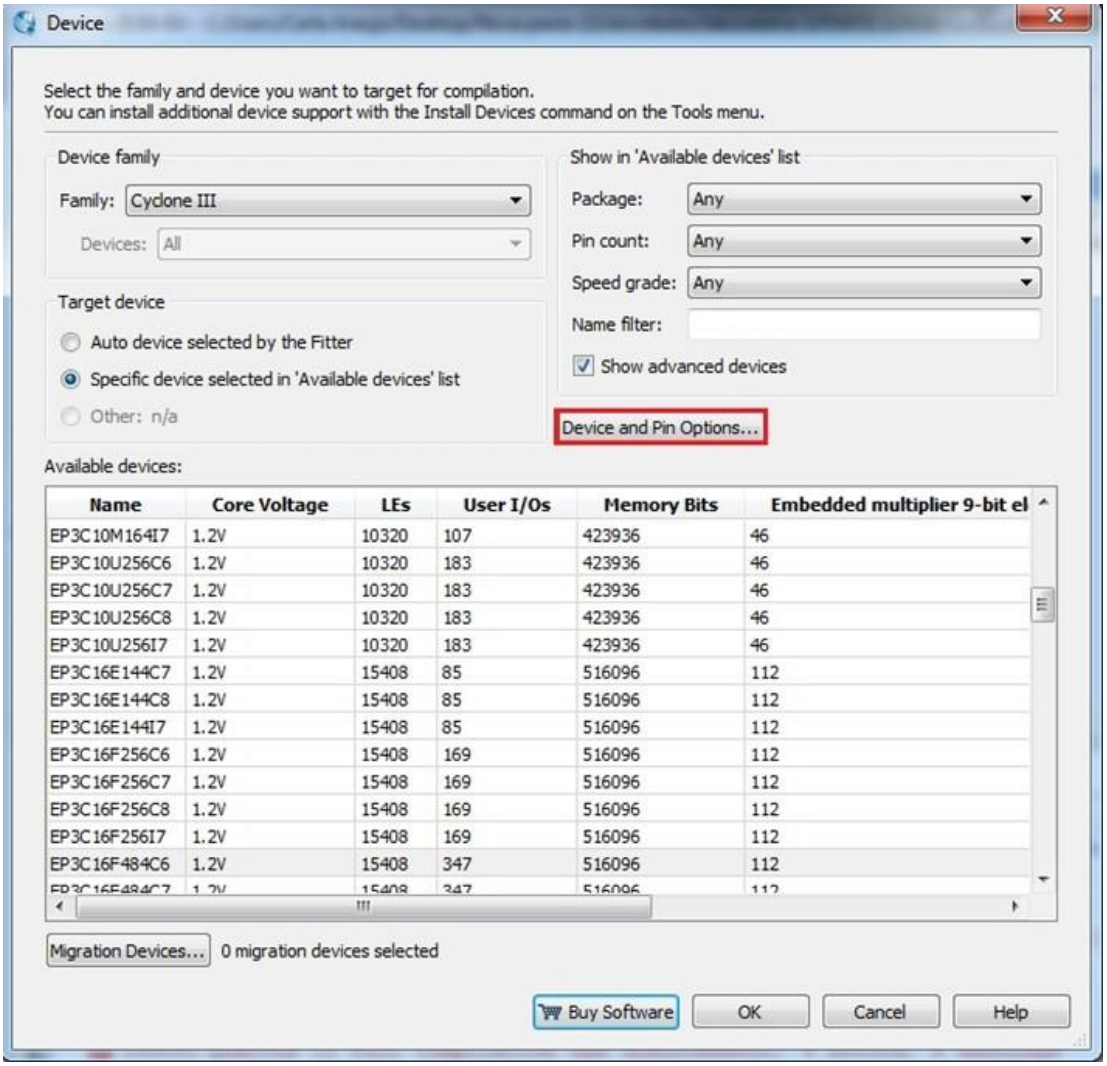


Figura 3.82 *Seleção das opções de dispositivo e pinos*

Passo 3 – Ao fazer o passo 2, vai ser mostrado uma tela aonde será selecionado uma categoria de configuração, no nosso caso vamos selecionar *Dual-purpose pins*, onde será configurado a dupla finalidade do pino.

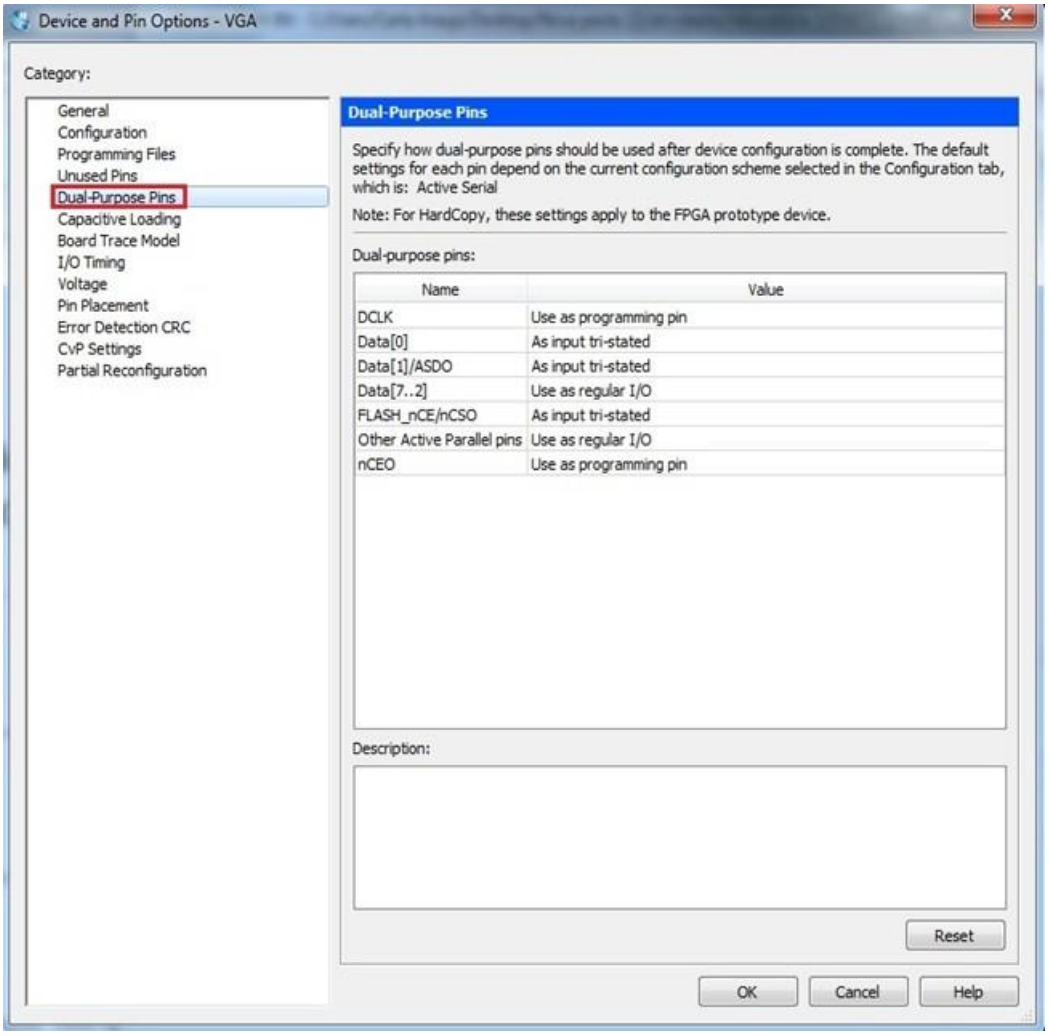


Figura 3.83 *Configuração da dupla finalidade dos pinos*

Passo 4- Depois de fazer o passo 3, com *Dual-Purpose* Pins já selecionado, clique em *nCEO > Use as regular I /O > OK*, e o pino estará configurado e agora o K22 será visto como uma entrada/saída regular, idêntico aos outros pinos.

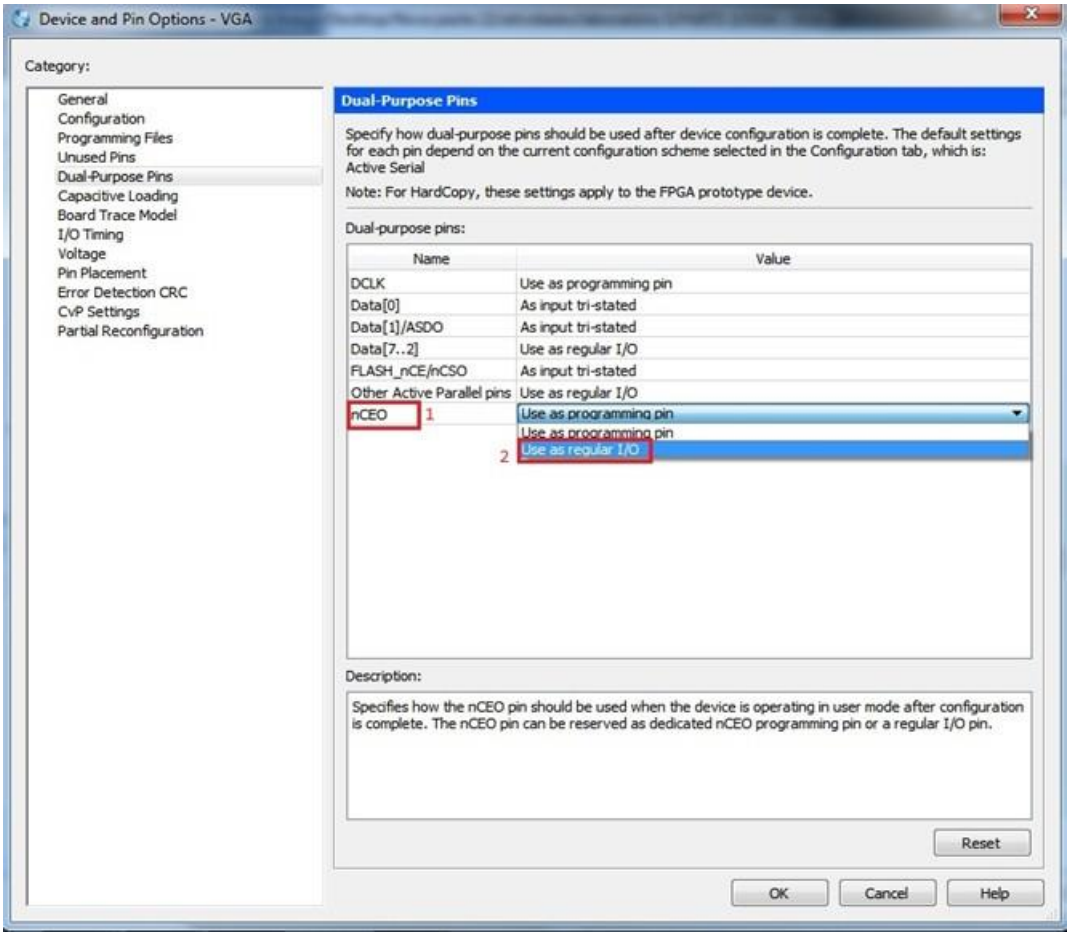


Figura 3.84 *Configuração de entrada e saída regular dos pinos*

O teste na placa DE0 com os pinos mostrados anteriormente foi feito, e é mostrado seu resultado na figura 3.85.

Figura 3.85 *Resultado na placa DE0*

Tarefa 14: Altere a cor da linha de branco para amarelo.

3.5.3 Parte III - VGA – Quadrado

Os códigos abaixo foram criados para desenhar um quadrado branco na tela do computador. Observe que estamos utilizando um pacote que faz a seleção das cores no padrão *RGB* que deverá ser desenhado, quadro 3.25. No quadro 3.26 *SYNC.vhd* está sendo utilizado somente para desenhar o quadrado, sem alterar diretamente as cores do quadrado. Os códigos que desenhavam as linhas foram mantidos para comparação, mas foram colocados como comentário.

Quadro 3.25 *Código 25 – pcg.vhd. quadrado*

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

PACKAGE my IS
        PROCEDURE sq(
                                SIGNAL xcur, ycur, xpos, ypos : IN integer;
                                SIGNAL rgb : OUT std_logic_vector(3 downto 0);
                               SIGNAL draw : OUt std_logic
        );
END my;
PACKAGE BODY my IS
        PROCEDURE sq(
                                SIGNAL xcur, ycur, xpos, ypos : IN integer;
                                SIGNAL rgb : OUT std_logic_vector(3 downto 0);
                               SIGNAL draw : OUT std_logic
        BEGIN
                              IF( xcur > xpos AND xcur < (xpos+100) AND ycur > ypos AND ycur
<(ypos+100) ) THEN
                rgb<=”1111”;
                        draw<=’1’;
                ELSE
                        draw<=’0’;
                END IF;
        END sq;
END my;
```

Quadro 3.26 *Código 26 – SYNC.vhd. quadrado*

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

ENTITY  sync IS  PORT(  clk : IN std_logic;
                        hsync , vsync : OUt std_logic;
                       r, g, b : OUT std_logic_vector(3 downto 0));

END sync;

ARCHITECTURE main OF sync IS
SIGNAL rgb : std_logic_vector(3 downto 0);
SIGNAL draw : std_logic;
SIGNAL sq_x1, sq_y1 : integer RANGE 0 to 1688 := 0;
SIGNAL hpos : integer RANGE 0 to 1688 := 0;
SIGNAL vpos : integer RANGE 0 to 1066 := 0;
BEGIN
        sq_x1 <= 1048;
        sq_y1 <= 554;
        sq( hpos, vpos, sq_x1, sq_y1, rgb, draw);
        PROCESS (clk)
        BEGIN
                IF( clk'event and clk='1' ) THEN
                        -- desenha um quadrado no meio da tela
                                IF( draw = '1' ) THEN

                                                r <= rgb;
                                                g <= rgb;
                                                b<=  rgb;
                                        ELSE
                                                r <= (others=>'0');
                                                g<=  (others=>'0');
                                                b<=  (others=>'0');

                                        END IF;
                                -- desenha uma linha vertical e outra horizontal no meio da tela
                                 --  IF( hpos=1042 or vpos=554 ) THEN
                                 --          r <= (others=>'1');
                                 --          g <= (others=>'1');
                                 --          b <= (others=>'1');
                                 --  ELSE
                                 --          r <= (others=>'0');
                                 --          g <= (others=>'0');
                                --          b <= (others=>'0');

                                 -- END IF;

                                  IF( hpos < 1688 ) THEN
```

```
                        hpos <= hpos + 1;
                    ELSE
                        hpos <= 0;

                        IF( vpos < 1066 ) THEN
                                vpos <= vpos + 1;
                        ELSE
                                vpos <= 0;
                        END IF;
                    END IF;

                    IF( hpos > 48 AND hpos < 160 ) THEN
                        hsync <= '0';
                    ELSE
                        hsync <= '1';
                    END IF;

                    IF( vpos > 0  AND vpos < 4 ) THEN

                        vsync <= '0';
                    ELSE
                        vsync <= '1';
                    END IF;

                    IF( (hpos > 0 AND hpos < 408) OR (vpos > 0 AND vpos <
42) ) THEN
                        r <= (OTHERS=>'0');
                        g <= (OTHERS=>'0');
                        b <= (OTHERS=>'0');
                    END IF;
                END IF;
            END PROCESS;
END main;
```

Quadro 3.27 *Código 27 – VGA.VHD do quadrado.*

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

ENTITY vga is
        PORT (clock_24 : IN std_logic_vector(1 downto 0);
                vga_hs, vga_vs : OUT std_logic;
                vga_r, vga_g, vga_b : OUT std_logic_vector(3 downto 0));
END vga;

ARCHITECTURE main OF vga IS
SIGNAL vgaclk, reset : std_logic:='0';
COMPONENT pll1 is
        PORT(clk_in_clk : IN std_logic := 'x'; -- clk
                reset_reset : IN std_logic := 'x'; -- reset
```

```
                clk_out_clk : OUT std_logic -- clk
        );
END COMPONENT pll1;

COMPONENT sync IS
        PORT( clk : IN  std_logic;
                hsync, vsync : OUT std_logic;
                r, g, b : OUT std_logic_vector(3 downto 0)
         );
END    COMPONENT    sync;
BEGIN
    c1 : sync PORT MAP( vgaclk, vga_hs, vga_vs, vga_r, vga_g, vga_b);
   c2 : pll1 PORT MAP(clock_24(0), reset, vgaclk);
END main;
```

Para verificar a criação do quadrado na tela, utilizamos as conexões dos sinais do código VGA dos pinos da placa DE0 da tabela 3.27 mostrado na parte I deste laboratório 5.

O resultado da implementação na placa pode ser verificado na figura 3.86.

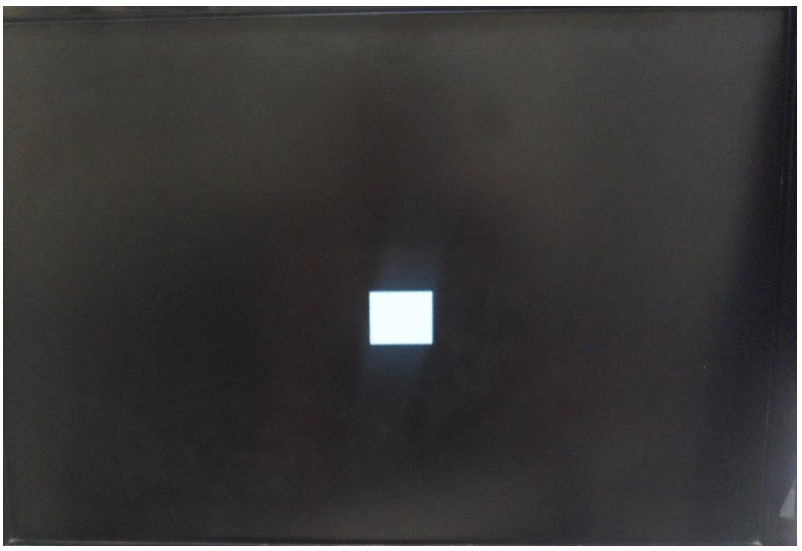


Figura 3.86 *Resultado na placa do código VGA.vhd do quadrado*

3.5.4 Parte IV- Jogo simples

Com os conhecimentos aprendidos nos tópicos anteriores, vamos criar um jogo simples, utilizando dois quadrados. Nos códigos abaixo são utilizadas as entradas *Button*[0-2] e SW[0] para mover os quadrados e as chaves SW[8] e SW[9] para selecionar os quadrados que serão movidos. No caso do nosso jogo somente os quadrados em vermelho serão movidos.

No código do componente PCG, são adicionados os sinais utilizados, assim como o tamanho dos quadrados.

Quadro 3.28 *Código 28 – PCG.vhd jogo simples*

```
LIBRARY IEEE;
```

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

PACKAGE my IS
        PROCEDURE sq (SIGNAL xcur, ycur, xpos, ypos: IN integer;
                      SIGNAL rgb: OUT std_logic_vector (3 downto 0);
                      SIGNAL draw : OUT std_logic
                     );
        END my;
PACKAGE BODY my IS

        PROCEDURE sq ( SIGNAL xcur, ycur, xpos, ypos : IN integer;
                       SIGNAL rgb : OUT std_logic_vector(3 downto 0);
                       SIGNAL draw : OUT std_logic ) is
         BEGIN
              IF ( xcur > xpos AND xcur < (xpos+100) AND ycur > ypos AND ycur <
ypos+100) ) tHEN
                  rgb <= "1111";
                  draw <= '1';
               ELSE
                   draw <= '0';
               END IF;
         END sq;
    END my;
```

Abaixo o código *SYNC* aonde será feita a sincronização para a criação dos quadrados, tão quando seus movimentos e cores.

Quadro 3.29 *Código 27 – SYNC.vhd do jogo simples*

```
LIBRARY IEE E;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.NUMERIC_STD.ALL;
USE WORK.MY.ALL;
ENTITY sync IS
      PORT (  clk : IN std_logic;
              hsync , vsync : OUT std_logic;
              r, g, b : OUT std_logic_vector(3 downto 0);
             keys : IN std_logic_vector(3 downto 0);
            s : IN std_logic_vector(1 downto 0)
           );
END sync;
```

```
ARCHITECTURE main OF sync IS
```

```
SIGNAL rgb: std_logic_vector (3 downto 0);
SIGNAL draw1, draw2: std_logic;
SIGNAL sq_x1, sq_y1: integer RANGE 0 TO 1688: = 600;
SIGNAL sq_x2, sq_y2: integer RANGe 0 TO 1688: = 500;
SIGNAL hpos: integer RANGE 0 to 1688: = 0;
SIGNAL vpos: integer RANGE 0 to 1066: = 0;
BEGIN
      --sq_x1 <= 1048;
      --sq_y1 <= 554;
       sq (hpos, vpos, sq_x1, sq_y1, rgb, draw1);
       sq (hpos, vpos, sq_x2, sq_y2, rgb, draw2);
PROCESS (clk)
BEGIN
      IF (clk'event AND clk='1' ) THEN
           IF (draw1 = '1' ) THEN
                IF(s(0) = '1' ) THEN
                        r <= (OTHERS=>'1');
                        g <= (OTHERS=>'0');
                        b <= (OTHERS=>'0');
               ELSE
                        r <= (OTHERS=>'1');
                        g <= (OTHERS=>'1');
                        b <= (OTHERS=>'1');
               END IF;
           END IF;

          IF ( draw2 = '1' ) THEN
               IF ( s(1) = '1' ) THEN
                      r <= (OTHERS=>'1');
                      g <= (OTHERS=>'0');
                      b <= (OTHERS=>'0');
               ELSE
                      r <= (OTHERS=>'1');
                     g  <= (OTHERS=>'1');
```

```
                b <= (OTHERS=>'1');
            END IF;
        END IF;

        IF( draw1 = '0' AND draw2 = '0' ) THEN
            r <= (OTHERS=>'0');
            g <= (oTHERS=>'0');
            b <= (OTHERS=>'0');
        END IF;
        IF( hpos < 1688 ) THEN
            hpos <= hpos + 1;
        ELSE
            hpos <= 0;
        IF( vpos < 1066 ) THEN
            vpos <= vpos + 1
        ELSE
            IF( s(0) = '1' ) THEN
                IF( keys(0) = '0' ) THEN
                    sq_x1 <= sq_x1 + 5;
                END IF;
                IF( keys(1) = '0' ) THEN
                    sq_x1 <= sq_x1 - 5;
                END IF;
                IF( keys(2) = '0' ) THEN
                    sq_y1 <= sq_y1 + 5;
                END IF;
                IF( keys(3) = '0' ) THEN
                    sq_y1 <= sq_y1 - 5;
                END IF;
            END IF;
            IF( s(1) = '1' ) THEN
                IF( keys(0) = '0' ) THEN
                    sq_x2 <= sq_x2 + 5;
                END IF;
```

```
                IF( keys(1) = '0' ) THEN
                    sq_x2 <= sq_x2 - 5;
                END IF;
                IF( keys(2) = '0' ) THEN
                    sq_y2 <= sq_y2 + 5;
                END IF;
                IF( keys(3) = '0' ) THEN
                    sq_y2 <= sq_y2 - 5;
                END IF;
            END IF;
            vpos <= 0;
        END IF;
    END IF;
    IF( hpos > 48 AND hpos < 160 ) THEN
        hsync <= '0';
    ELSE
        hsync <= '1';
    END IF;
        IF( vpos > 0 and vpos < 4 ) THEN
            vsync <= '0';
        ELSE
            vsync <= '1';
        END IF;
            IF( (hpos > 0 AND hpos < 408) OR (vpos > 0 AND vpos < 42) ) THEN
                r <= (OTHERS=>'0');
                g <= (OTHERS=>'0');
                b <= (OTHERS=>'0');
            END IF;
    END IF;
  END PROCESS;
END main;
```

No código VGA, vão ser integrados como componentes os códigos criados acima, e o PLL, aonde a forma de gerar seu código foi ensinada na parte II desse laboratório.

Quadro 3.30 *Código VGA jogo simples*

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

ENTITY vga IS
        PORT ( clock_24 : IN std_logic_vector(1 downto 0);
               vga_hs, vga_vs : OUT std_logic;
               vga_r, vga_g, vga_b : OUT std_logic_vector(3 downto 0);
               key : IN std_logic_vector(3 downto 0);
               sw : IN std_logic_vector(1 downto 0)
               );
END vga;

ARCHITECTURE main OF vga IS
SIGNAL vgaclk, reset : std_logic:='0';

COMPONENT pll1 IS
        PORT ( clk_in_clk : IN std_logic := 'x'; -- clk
               reset_reset : IN std_logic := 'x'; -- reset
               clk_out_clk : OUT std_logic -- clk
              );
END COMPONENT pll1;

COMPONENT sync IS
        PORT ( clk : IN std_logic;
               hsync, vsync : OUT std_logic;
               r, g, b : OUt std_logic_vector(3 downto 0);
               keys : IN std_logic_vector(3 downto 0);
               s : IN std_logic_vector(1 downto 0)
             );
END COMPONENT sync;
BEGIN
    c1 : sync PORT MAP( vgaclk, vga_hs, vga_vs, vga_r, vga_g, vga_b, key, sw);
    c2 : pll1 PORT MAP(clock_24(0), reset, vgaclk);
END main;
```

Tabela 3.27 *Conexão dos pinos VGA.*

Sinal VHDL	**Pino**
*CLOCK*_24[1]	------
*CLOCK*_24[0]	PIN_G21
KEY[3]	PIN_J6
KEY[2]	PIN_F1
KEY[1]	PIN_G3
KEY[0]	PIN_H2
SW[1]	PIN_D2
SW[0]	PIN_E4
VGA_B[3]	PIN_K18
VGA_B[2]	PIN_J22

VGA_B[1]	PIN_K21
VGA_B[0]	PIN_K22
VGA_G[3]	PIN_J21
VGA_G[2]	PIN_K17
VGA_G[1]	PIN_J17
VGA_G[0]	PIN_H22
VGA_R[3]	PIN_H21
VGA_R[2]	PIN_H20
VGA_R[1]	PIN_H17
VGA_R[0]	PIN_H19
VGA_HS	PIN_L21
VGA_VS	PIN_L22

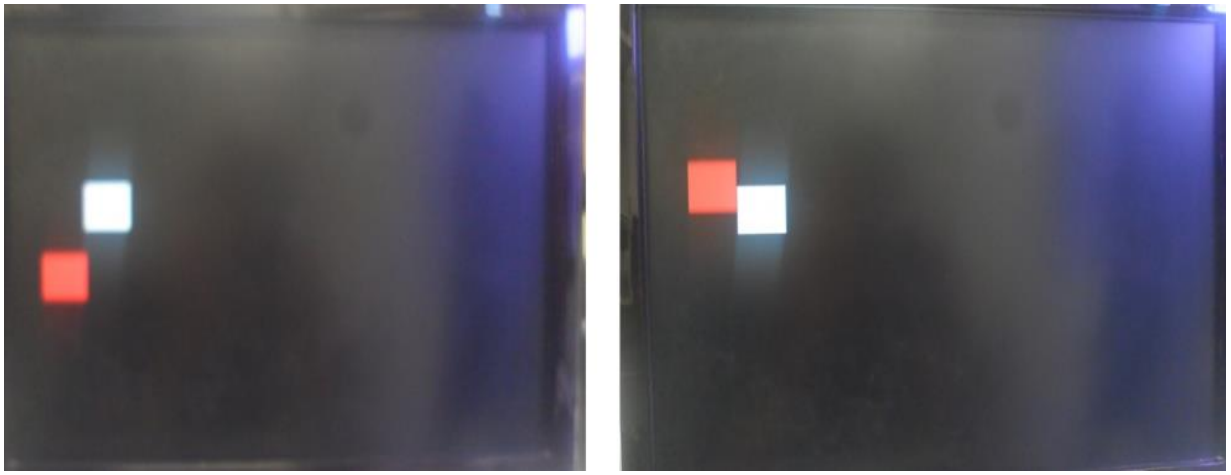


Figura 3.87 *Resultado do jogo simples na placa*

4. APLICAÇÃO

Um dos jogos mais tradicionais e o primeiro jogo lucrativo da história, o jogo *Pong* era formado apenas por um monitor e o console de jogo (portanto o jogo era um videogame). Quanto à funcionalidade, baseava-se em duas barras (denominadas *paddle*), controladas por dois jogadores diferentes, que "rebatiam" uma bola para impedir que a mesma alcançasse suas respectivas *dead zones*. Muito similar aos populares fliperamas da época. O objetivo deste capítulo é criar uma variação single player deste jogo e utilizá-lo como exemplo de aplicação didática de desenvolvimento de projetos em VHDL, utilizando o padrão VGA e botões da placa DE0 como IHM (*Interface Homem-Máquina*) do jogo.

Em vez de fazer com que a bola vá para a *dead zone* do adversário, o jogador deverá apenas brincar com ela, impedindo-a de cair em sua própria *dead zone*. Se a bola entrar em contato com sua *dead zone*, então a mesma volta à posição de início e é acrescentado um ponto ao contador que será visualizado no display de sete segmentos da placa DE0. O movimento da barra será controlado pelo comando de dois botões, que o moverão para cima e para baixo, e a "bola" terá seu movimento orientado pela batida nas laterais do jogo e na barra.

4.1 DESENVOLVIMENTO DA APLICAÇÃO.

Para o desenvolvimento dessa aplicação utiliza-se a sessão anterior 3.5 como base. Pois a aplicação irá fará uso de dois quadrados, como os criados na sessão citada, sendo um definido como a bola, e o outro como a barra do jogo.

4.1.1 Módulo PCG

Neste módulo são definidos os pacotes que contém os procedimentos usados pela bola e pela barra, sendo estes semelhantes, diferenciando-se apenas com os parâmetros de tamanho horizontal e vertical, que são incluídos como procedimentos da barra.

Abaixo encontrasse o código *PCG.vhd*.

Quadro 4.1 *código PCG.vhd jogo pong*

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.NUMERIC_STD.ALL;
PACKAGE my IS
PROCEDURE bola(
        SIGNAL xcur, ycur, xpos, ypos : IN integer;
        SIGNAL rgb : OUT std_logic_vector(3 downto 0);
        SIGNAL draw : OUT std_logic
);
-- o procedimento para a barra é praticamente igual ao procedimento para a bola, com a
```

```
-- única diferença sendo que o procedimento da barra possui dois parâmetros adicionais,
-- sendo eles o tamanho vertical e horizontal.

PROCEDURE barra(
        SIGNAL xcur, ycur, xpos, ypos, tamanho_horizontal, tamanho_vertical : IN integer;
        SIGNAL rgb : OUT std_logic_vector(3 downto 0);
        SIGNAL draw : OUT std_logic
);
END my;
-- pacotes que definem as dimensões da bola e da barra
PACKAGE body my IS
        PROCEDURE bola(
            SIGNAL xcur, ycur, xpos, ypos : IN integer;
            SIGNAL rgb : OUT std_logic_vector(3 downto 0);
            SIGNAL draw : OUT std_logic ) is

            BEGIN
                IF( xcur > xpos AND xcur < (xpos+25) AND ycur > ypos AND ycur <
                (ypos+25) ) THEN
                    rgb <= "1111";
                    draw <= '1';
                ELSE
                    draw <= '0';
                END IF;
        END bola;
        PROCEDURE barra(
            SIGNAL xcur, ycur, xpos, ypos, tamanho_horizontal, tamanho_vertical : IN
integer;
            SIGNAL rgb : OUT std_logic_vector(3 downto 0);
            SIGNAL draw : OUT std_logic ) IS
            BEGIN
                IF(xcur > xpos AND xcur < (xpos+tamanho_horizontal) AND ycur >
ypos AND ycur < (ypos+tamanho_vertical) ) THEN
                    rgb <= "1111";
                    draw <= '1';
                ELSE
                    draw <= '0';
                END IF;
        END barra;
END my;
```

- PROCEDIMENTO BOLA

Até o momento, o objeto a ser rebatido pela barra, vem sendo tratando erroneamente como “bola”. Entretanto, este não possui a forma de uma bola. Propôs-se que a figura tivesse, apesar de ser comumente uma bola, o formato de um quadrado, para o procedimento da bola, estão sendo passados como Parâmetros de entrada bola *Xcur, Ycur, Xpos, Ypos*, e Parâmetros de saída *RGB* e *DRAW.*

Quadro 4.2 *Parâmetros de entrada e saída da bola.*

```
PROCEDURE bola(
        SIGNAL xcur, ycur, xpos, ypos : IN integer;
        SIGNAL rgb : OUT std_logic_vector(3 downto 0);
        SIGNAL draw : OUT std_logic
);
```

Os parâmetros de entrada *Xcur* e *Ycur* recebem a posição do pixel que está sendo desenhado. Se estiver dentro da área definida pelo quadrado com canto superior esquerdo no ponto (*Xpos, Ypos*) e com largura igual a 25 e altura igual a 25 a variável *DRAWBOLA* que é o parâmetro de saída usada para desenhar o pixel na tela, é ativada, indicando para o arquivo *SYNC.vhd* que o pixels indicado por *Xcur* e *Ycur* pode ser desenhado, a cor que o pixel vai receber é determinada pelo parâmetro de saída *RGB*, inicialmente será desenhado na cor branca, mais pode ter sua cor alterada no módulo *SYNC* como será visto adiante.

Quadro 4.3 *Procedimento bola.*

```
PACKAGE body my IS
        PROCEDURE bola(

                SIGNAL xcur, ycur, xpos, ypos : IN integer;
                SIGNAL rgb : OUT std_logic_vector(3 downto 0);
                SIGNAL drawbola : OUT std_logic ) is

        BEGIN

                IF( xcur > xpos AND xcur < (xpos+25) AND ycur > ypos AND ycur <
                  (ypos+25) ) THEN
                                rgb <= "1111";
                                drawbola <= '1';
                        ELSE
                                drawbola <= '0';
                        END IF;
END BOLA;
```

- PROCEDIMENTO BARRA

Como foi mencionado, os procedimentos para a bola e para barra são semelhantes, porém no procedimento da barra são acrescidos dos parâmetros dimensionais TAMANHO_HORIZONTAL, TAMANHO_VERTICAL.

São definidos como parâmetros de entrada da barra *Xcur, Ycur, Xpos, Ypos*, *TAMANHO_HORIZONTAL*, *TAMANHO_VERTICAL*, e como saídas os valores RGB, e *DRAWBARRA.*

Quadro 4.4 *Parâmetros de entrada e saída da barra.*

```
PROCEDURE barra(
        SIGNAL xcur, ycur, xpos, ypos, tamanho_horizontal, tamanho_vertical :IN integer;
        SIGNAL rgb : OUT std_logic_vector(3 downto 0);
        SIGNAL draw : OUT std_logic );
```

Assim como foi mostrado no procedimento da bola, os parâmetros de entrada *Xcur* e *Ycur* recebem a posição do pixel que está sendo desenhado e se estiver dentro da área definida pela barra com canto superior esquerdo no ponto *Xpos* e *Ypos* e com largura igual a *TAMANHO_HORIZONTAL* e altura igual a *TAMANHO_VERTICAL* a variável *DRAWBARRA*, é ativada e o pixels indicado por *Xcur* e *Ycur* pode ser desenhado, a cor que o pixel vai receber é determinada pelo parâmetro de saída *RGB*, inicialmente será desenhado na cor branca, mais poderá também ter sua cor alterada no módulo *SYNC*.

Quadro 4.5 *Procedimento barra*

```
PROCEDURE barra(
                SIGNAL xcur, ycur, xpos, ypos, tamanho_horizontal, tamanho_vertical : IN
integer;
                SIGNAL rgb : OUT std_logic_vector(3 downto 0);
                SIGNAL drawbarra : OUT std_logic ) is

                BEGIN
                        IF( xcur > xpos AND xcur < (xpos+tamanho_horizontal) AND ycur >
ypos AND ycur <
                        (ypos+tamanho_vertical) ) THEN
                                rgb <= "1111";
                                drawbarra <= '1';
                        ELSE
                                drawbola <= '0';
                        END IF;
        END barra;
END my;
```

4.1.2 Módulo SYNC

Neste módulo, foi criado o componente de sincronização *SYNC.vhd*, para exibir imagens no monitor. Basicamente, este trata *HSYNC, VSYNC* como indicadores de pixels de cores de coordenadas XY, atribuindo ao sinal *RGB* o valor da cor do respectivo ponto.

Quadro 4.6 *Código Sync.vhd jogo pong*

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

```
USE IEEE.NUMERIC_STD.ALL;
USE WORK.MY.ALL;
ENTITY sync IS
	PORT(
		clk : IN std_logic;
		hsync , vsync : OUT std_logic;
		r, g, b : OUT std_logic_vector(3 downto 0);
		keys : IN std_logic_vector(1 downto 0);
		led: OUT std_logic_vector(6 downto 0)
	);
END sync;
ARCHITECTURE main OF sync IS
-- y = horizontal, x= vertical
-- sinais atribuidos para as funcionalidades da bola
-- movimento, espaço aonde ela deve bater.
	SIGNAL rgb : std_logic_vector(3 downto 0);
	SIGNAL drawbola : std_logic;
	SIGNAL x_bola : integer range 0 TO 1688 := 844;
	SIGNAL y_bola : integer range 0 TO 1688 := 400;
	SIGNAL hpos : integer range 0 TO 1688 := 0;
	SIGNAL vpos : integer range 0 TO 1066 := 0;
	SIGNAL hbola : integer range 0 TO 1688 := 0;
	SIGNAL vbola : integer range 0 TO 1066 := 0;
	SIGNAL my : integer range -10 TO 10 := 1;-- movimento da bola
	SIGNAL mx : integer range -10 TO 10 := 1;-- movimento da bola

-- sinais atribuidos para as funções da barra
	SIGNAL rgb_barra : std_logic_vector(3 downto 0);
	SIGNAL x_barra, y_barra : integer RANGE 0 TO 1670 := 440; -- determina o espaço
que a barra vai ficar na tela, direito ou esquerdo (no caso do jogo o esquerdo)
	SIGNAL drawbarra : std_logic;
	SIGNAL hbarra : integer RANGE 0 TO 1688 := 0;
	SIGNAL vbarra : integer RANGE 0 TO 1066 := 0;

-- sinais atribuidos para marcação dos pontos e diminuição da barra.
	SIGNAL points : integer RANGE 0 TO 100 := 0;
	SIGNAL tamanho_horizontal : integer RANGE 0 to 100 := 20;
	SIGNAL tamanho_vertical : integer RANGE 0 TO 200 := 190;
	SIGNAL num_vezes_toque_barra : integer RANGE 0 TO 2000 := 0;
	SIGNAL velocidade_bola : integer RANGE 0 TO 50 := 10;

BEGIN
bola( hbola, vbola, x_bola, y_bola, rgb, drawbola);
barra(hbarra, vbarra, x_barra, y_barra, tamanho_horizontal, tamanho_vertical, rgb_barra,
drawbarra);
PROCESS (clk)
BEGIN

	IF( clk'event and clk='1') THEN
		-- desenha um quadrado no meio da tela representando a bola do jogo
```

```
IF( drawbola = '1') THEN
        r <= (OTHERS=>'1');-- cor da bola(amarela)
        g <= (oTHERS=>'1');
        b <= (OTHERS=>'0');
ELSE
        r <= (OTHERS=>'0');
        g <= (OTHERS=>'0');
        b <= (OTHERS=>'0');
END IF;

-- desenha a barra

IF( drawbarra='1' ) THEN
        r <= (OTHERS => '1');  -- cor da barra(rosa)
        g <= (OTHERS =>'0');
        b <= (OTHERS =>'1');
END IF;


-- controla a bola
IF( hbola < 1688 ) THEN
        hbola <= hbola + 1;
ELSE
        hbola <= 0;

        IF( vbola < 1066 ) THEN
                vbola <= vbola + 1;
        ELSE
                -- caso a bola toque na parte superior da tela
                IF( y_bola > (1066-50) ) THEN
                        my <= - velocidade_bola;

                        y_bola <= y_bola + my;
                        x_bola <= x_bola + mx;

                -- caso a bola toque na parte inferior da tela
                ELSIF(y_bola < 50) THEN
                        my <= velocidade_bola;

                        y_bola <= y_bola + my;
                        x_bola <= x_bola + mx;

                -- caso a bola toque na lateral direita da tela
                ELSIF (x_bola > 1638) THEN
                        mx <= - velocidade_bola;

                        y_bola <= y_bola + my;
                        x_bola <= x_bola + mx;

                -- caso a bola bata na barra
```

```
                                ELSIF (x_bola < 440 + tamanho_horizontal AND y_bola >=
y_barra - 50 AND y_bola <= y_barra + tamanho_vertical) THEN
                        num_vezes_toque_barra <= um_vezes_toque_barra + 1;

                -- a cada 5 toques da bola na barra, aumentamos a velocidade em 1 unidade.
                                        IF (num_vezes_toque_barra > 5) THEN
                                                num_vezes_toque_barra <= 0;
                                                velocidade_bola <= velocidade_bola + 1;
                                        END IF;

                                        mx <= velocidade_bola;

                                        y_bola <= y_bola + my;
                                        x_bola <= x_bola + mx;

                                        -- a cada toque na bola, reduzimos o tamanho da barra
                                        -- por 3 unidades, até o tamanho mínimo de 60.
                                        -- e quando chegar ao tamanho mínimo o jogo para.
                                        IF (tamanho_vertical > 60) THEN
                                                tamanho_vertical <= tamanho_vertical - 3;

                                                IF( drawbarra='1' ) THEN
                                                   r <= (OTHERS=>'1');
                                                   g <= (OTHERS=>'1');
                                                   b <= (oTHERS=>'1');
                                                END IF;
                                        END IF;

                        -- caso a bola toque a lateral esquerda da tela, caracterizando um ponto.

                                ELSIF (x_bola < 400) THEN
                                        mx <= velocidade_bola;
                                        my <= velocidade_bola;
                                        points <= points + 1;

                                        y_bola <= 550 + my;
                                        x_bola <= 1000 + mx;

                                ELSE
                                        y_bola <= y_bola + my;
                                        x_bola <= x_bola + mx;
                                END IF;

                                        vbola <= 0;
                        END IF;
                END IF;
-- controla a barra
        IF( hbarra < 1688 ) THEN
                        hbarra <= hbarra + 1;
        ELSE
```

```
                        hbarra <= 0;

                        IF( vbarra < 1066 ) THEN
                                vbarra <= vbarra + 1;
                        ELSE
                                -- comandos para movimentar a barra para cima e para baixo.
                            IF((y_barra)>=42 AND (y_barra+190)<=1066) THEN -- limite da
barra  na tela
                            IF( keys(0) = '0' ) then-- movimento pra cima(button 2 da placa)
                                        y_barra <= y_barra + 5;
                             END IF;

                             IF( keys(1) = '0' ) THEN -- movimento para baixo(button 1 da
placa)
                                        y_barra <= y_barra - 5;
                            END IF;
                            ELSE
                                IF (y_barra < 42) THEN
                                        y_barra <= y_barra + 5;
                                ELSE
                                        y_barra <= y_barra - 5;
                                END IF;
                            END IF;
                               vbarra <= 0;
                            END IF;
        END IF;

                -- controla outros objetos

                IF( hpos < 1688 ) THEN
                        hpos <= hpos + 1;
                ELSE
                        hpos <= 0;

                        IF( vpos < 1066 ) THEN
                                vpos <= vpos + 1;
                        ELSE
                                vpos <= 0;
                        END IF;
                END IF;

                IF( hpos > 48 AND hpos < 160 ) THEN
                        hsync <= '0';
                ELSE
                        hsync <= '1';
                END IF;

                IF( vpos > 0  AND vpos < 4 ) THEN
                        vsync <= '0';
                ELSE
```

```
                vsync <= '1';
        END IF;

        IF( (hpos > 0 AND hpos < 408) OR (vpos > 0 AND vpos < 42) ) THEN
                r <= (OTHERS =>'0');
                g <= (OTHERS =>'0');
                b <= (OTHERS =>'0');
        END IF;
```

- ENTIDADE DO CÓDIGO

Na entidade do código *SYNC.vhd*, foram declaradas como entrada *CLK*, e *KEYS* e como saídas *HSYNC, VSYNC, R,G,B e LED*.

Quadro 4.7 *Declarações de entrada e saída*

```
ENTITY sync IS
        PORT(
                clk : IN std_logic;
                hsync , vsync : OUT std_logic;
                r, g, b : OUT std_logic_vector(3 downto 0);
                keys : IN std_logic_vector(1 downto 0);
                led: OUT std_logic_vector(6 downto 0)      );
END sync;
```

- ENTIDADE DO CÓDIGO

Na entidade do código *SYNC.vhd*, foram declaradas como entrada o *CLK*, este Gerado pelo *PLL*, e *KEYS*, que serão os *BUTTON* 1 e 2 da placa DE0 e como saídas para a placa *HSYNC, VSYNC, R,G,B* e *LED*.

Quadro 4.8 *Declarações de entrada e saída*

```
ENTITY sync IS
        PORT(
                clk : IN std_logic;
                hsync , vsync : OUT std_logic;
                r, g, b : OUT std_logic_vector(3 downto 0);
                keys : IN std_logic_vector(1 downto 0);
                led: OUT std_logic_vector(6 downto 0)      );
END sync;
```

- SINAIS ATRIBUÍDOS

No código serão atribuídos sinais para os movimentos, espaço onde a bola e a barra se encontram na tela, espaço aonde a bola deve bater, e sinais que serão atribuídos para fazer a marcação da pontuação do jogo.

Quadro 4.9 *Sinais atribuídos as funcionalidades da bola*

```
ARCHITECTURE main OF sync IS
```

```
-- x = horizontal, y = vertical
-- sinais atribuidos para as funcionalidades da bola
-- movimento, espaço aonde ela deve bater.

    SIGNAL rgb : std_logic_vector(3 downto 0);
    SIGNAL drawbola : std_logic;
    SIGNAL x_bola, y_bola : integer RANGE 0 TO 1688 := 600;
    SIGNAL hpos : integer RANGE 0 TO 1688 := 0;
    SIGNAL vpos : integer RANGE 0 TO 1066 := 0;
    SIGNAL hbola : integer RANGE 0 TO 1688 := 0;
    SIGNAL vbola : integer RANGE 0 TO 1066 := 0;
    SIGNAL my : integer RANGE -10 TO 10 := 5;-- movimento da bola
    SIGNAL mx : integer RANGE -10 TO 10 := 5;-- movimento da bola
```

Quadro 4.10 *Sinais atribuídos as funcionalidades da barra*

```
-- Sinais atribuídos para as funções da barra
        SIGNAL RGB_BARRA : std_logic_vector(3 downto 0);
        SIGNAL x_barra, y_barra : integer RANGE 0 TO 1670 := 440; -- Determina o --
espaço que a barra vai ficar na tela, direito ou esquerdo (no caso do jogo o esquerdo)
        SIGNAL drawbarra : std_logic;
        SIGNAL hbarra : integer RANGE 0 TO 1688 := 0;
        SIGNAL vbarra : integer RANGE 0 TO 1066 := 0;
```

Quadro 4.11 *Sinais atribuídos para marcação dos pontos*

```
--Sinais atribuídos para marcação dos pontos e diminuição da barra.
        SIGNAL points : integer RANGE 0 TO 100 := 0;
        SIGNAL tamanho_horizontal : integer RANGE 0 TO 100 := 20;
        SIGNAL tamanho_vertical : integer RANGE 0 TO 200 := 190;
        SIGNAL num_vezes_toque_barra : integer RANGE 0 TO 2000 := 0;
        SIGNAL velocidade_bola : integer RANGE 0 TO 50 := 10;
```

Ainda no módulo SYNC serão definidos os passos de criação do jogo.

- DESENHA A BOLA (QUADRADO)

Como já foi mencionado, não será de fato uma bola mais sim um quadrado, que já teve suas dimensões e cores especificadas no módulo PCG, no módulo SYNC ela será desenhada e pode ter sua cor alterada. A bola será desenhada na tela na cor amarela.

Quadro 4.12 *Desenha a bola*

```
bola ( hbola, vbola, x_bola, y_bola, rgb, drawbola);
PROCESS (clk)
BEGIN
        IF( clk'event AND clk='1') THEN
                -- desenha um quadrado no meio da tela representando a bola do jogo
                IF( drawbola = '1') THEN
                        r <= (OTHERS=>'1');-- cor da bola(amarela)
                        g <= (OTHERS=>'1');
                        b <= (OTHERS=>'0');
```

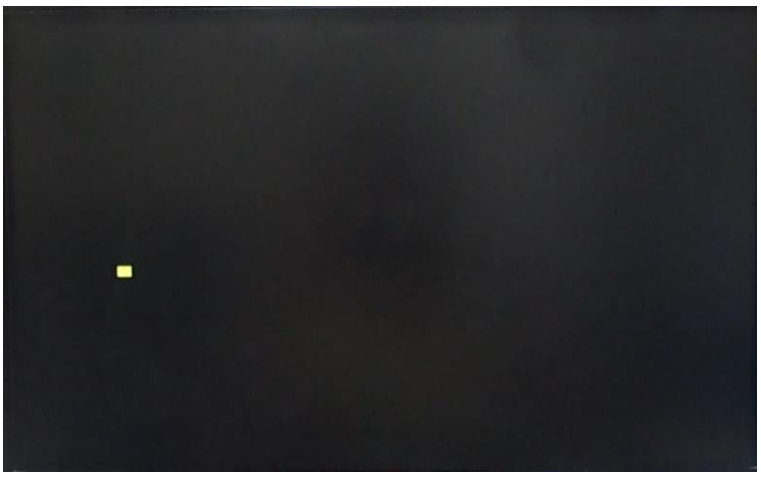


Figura 4.1 *Resultado na tela do desenho da bola*

- DESENHA BARRA

A barra também tem suas dimensões e cor especificadas no módulo PCG, e será desenhada no módulo SYNC podendo ter sua cor alterada, assim como no procedimento de desenhar a bola. No caso do jogo a barra será desenhada no canto esquerdo da tela e na cor rosa.

Quadro 4.13 *Desenha barra*

```
barra (hbarra, vbarra, x_barra, y_barra, tamanho_horizontal, tamanho_vertical, rgb_barra,
drawbarra);

PROCESS (clk)
BEGIN
-- desenha a barra no canto esquerdo da tela
                IF( drawbarra='1' ) THEN
                        r <= (OTHERS=>'1'); -- cor da barra (rosa)
```

```
                g <= (OTHERS=>'0');
                b <= (OTHERS=>'1');
            END IF;
```

Figura 4.2 *Resultado na tela do desenho da barra*

Depois de desenhar a bola e a barra, define-se o controle, as dimensões da tela ao qual a bola e barra poderão bater e a velocidade dos movimentos.

- CONTROLE BOLA (QUADRADO)

Estipulou-se o limite da bola dentro do espaço da tela, de acordo com as dimensões da tela.

Quadro 4.14 *Limite da bola*

```
-- Controla a bola
        IF( hbola < 1688 ) THEN
    hbola <= hbola + 1;
        ELSE
    hbola <= 0;
        IF( vbola < 1066 ) THEN
    vbola <= vbola + 1;
```

A bola não deverá simplesmente avançar e ultrapassar os limites do monitor. Quando a bola bater na barra ou com um dos limites da tela, para permanecer dentro da área permitida, ela será “rebatida”. Para o rebatimento da bola em qualquer situação foram estipulados casos, observe que todas as situações se dão em função da velocidade da bola.

- Caso a bola bata na parte superior

Quando a bola bate na parte superior da tela o movimento vertical (MY) da velocidade deverá ser invertido.

Quadro 4.15 *Caso a bola bata na parte superior*

```
-- Caso a bola toque na parte superior da tela
        IF( y_bola > (1066-50) ) THEN
            my <= - velocidade_bola;
            y_bola <= y_bola + my;
            x_bola <= x_bola + mx;
```

Figura 4.3 *Resultado na tela da bola batendo na parte superior*

- Caso a bola bata na parte inferior

Quando a bola bate na parte inferior da tela movimento vertical (MY) da velocidade deverá se o mantido.

Quadro 4.16 *Caso a bola bata na parte inferior*

```
-- Caso a bola toque na parte inferior da tela
        ELSIF (y_bola < 50) THEN
            my <= velocidade_bola;
            y_bola <= y_bola + my;
            x_bola <= x_bola + mx;
```

Figura 4.4 *Resultado na tela da bola batendo na parte inferior*

- Caso a bola bata na lateral direita

Quando a bola bate na lateral direita da tela o movimento horizontal (MX) da velocidade deverá ser invertido.

Quadro 4.17 *Caso a bola bata na lateral direita*

```
-- Caso a bola toque na lateral direita da tela
        ELSIF (x_bola > 1638) THEN
          mx <= - velocidade_bola;
         y_bola <= y_bola + my;
        x_bola <= x_bola + mx;
```

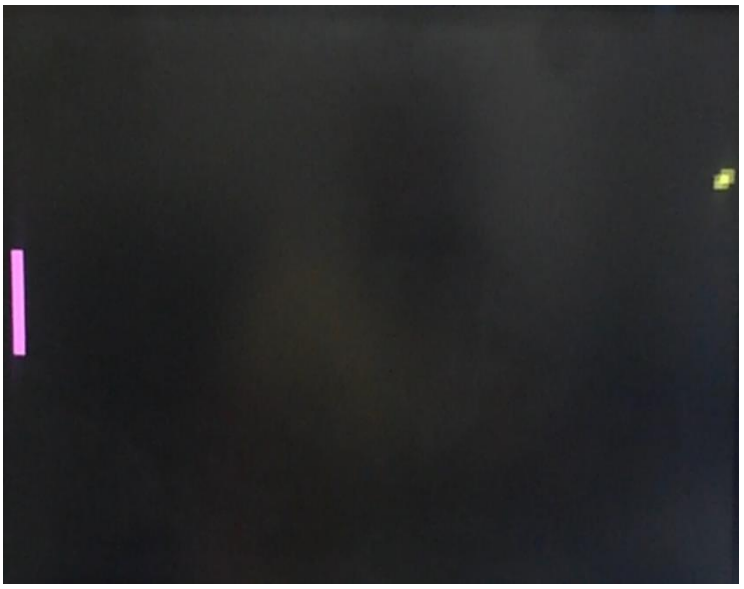


Figura 4.5 *Resultado na tela da bola batendo na parte direita.*

- Caso a bola bata na barra

Quando a bola bate na barra irá contar o número de vezes que a bola toca na barra, e a cada 5 toques a velocidade da bola aumenta em 1 unidade, e o movimento horizontal (MX) da velocidade deverá ser mantido.

Quadro 4.18 *Caso a bola bata na barra*

```
-- Caso a bola bata na barra
            ELSIF (x_bola < 440 + tamanho_horizontal AND y_bola >= y_barra - 50 AND
y_bola <= y_barra + tamanho_vertical) then

                    num_vezes_toque_barra <= num_vezes_toque_barra + 1;

-- a cada 5 toques da bola na barra, aumentamos a velocidade em 1 unidade.
                    IF (num_vezes_toque_barra > 5) THEN
                        num_vezes_toque_barra <= 0;
                        velocidade_bola <= velocidade_bola + 1;
                    END IF;

                    mx <= velocidade_bola;
                    y_bola <= y_bola + my;
                    x_bola <= x_bola + mx;
```

Figura 4.6 *Resultado na tela da bola batendo na barra*

- Redução barra

A cada toque da bola na barra, a barra deverá ser reduzida em 3 unidades, até chegar ao tamanho mínimo vertical de 60. Quando a barra atingir o tamanho mínimo vertical o jogo para.

Quadro 4.19 *Redução da barra*

```
-- A cada toque na bola, reduzimos o tamanho da barra
-- por 3 unidades, até o tamanho mínimo de 60.
```

```
-- e quando chegar ao tamanho mínimo o jogo para.

        IF (tamanho_vertical > 60) THEN
                tamanho_vertical <= tamanho_vertical - 3;

                IF( drawbarra='1' ) THEN
                    r <= (OTHERS=>'1');
                    g <= (OTHERS=>'1');
                    b <= (OTHERS=>'1');
                END IF;
        END IF;
```

Figura 4.7 *Resultado na tela da bola da redução da barra*

- Caso a bola bata na lateral esquerda

Se a bola bater na lateral esquerda, que é a área de *dead zone* do jogo, tanto o movimento horizontal (MX) quanto o movimento vertical (MY) vão receber a velocidade da bola, e a cada batida caracterizará um ponto. A pontuação é contabilizada pelo contador *POINTS*.

Quadro 4.20 *Caso a bola bata na lateral esquerda*

```
-- caso a bola toque a lateral esquerda da tela, caracterizando um ponto.
         ELSIF (x_bola < 400) THEN
                mx <= velocidade_bola;
                my <= velocidade_bola;
                points <= points + 1;

                y_bola <= 550 + my;
                x_bola <= 1000 + mx;

                ELSE
                y_bola <= y_bola + my;
                x_bola <= x_bola + mx;
        END IF;

        vbola <= 0;
        END IF;
    END IF;
```

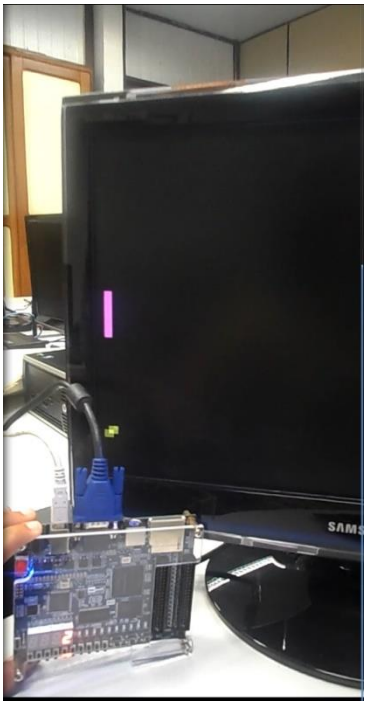


Figura 4.8 *Resultado na tela da pontuação*

- CONTROLE BARRA

A barra será posicionada no lado esquerdo, como já foi mencionado, e seu limite será as partes superior e inferior da tela.

Quadro 4.21 *Controle da barra*

```
-- Controla a barra
        IF (hbarra < 1688) THEN
                        hbarra <= hbarra + 1;
        ELSE
                        hbarra <= 0;

                        IF (vbarra < 1066) THEN
                                vbarra <= vbarra + 1;
```

A barra terá dois movimentos no sentido vertical que são para cima e para baixo, esses movimentos serão feitos através de dois botões da placa DE0 *BUTTON1* e *BUTTON2*.

Quadro 4.22 *Movimento barra*

```
                        ELSE
                                IF (y_barra < 42) THEN
                                    y_barra <= y_barra + 3;
                                ELSE
                                    y_barra <= y_barra - 3;
                                END IF;
                        END IF;
                                vbarra <= 0;
                END IF;
```

```
END IF;
```

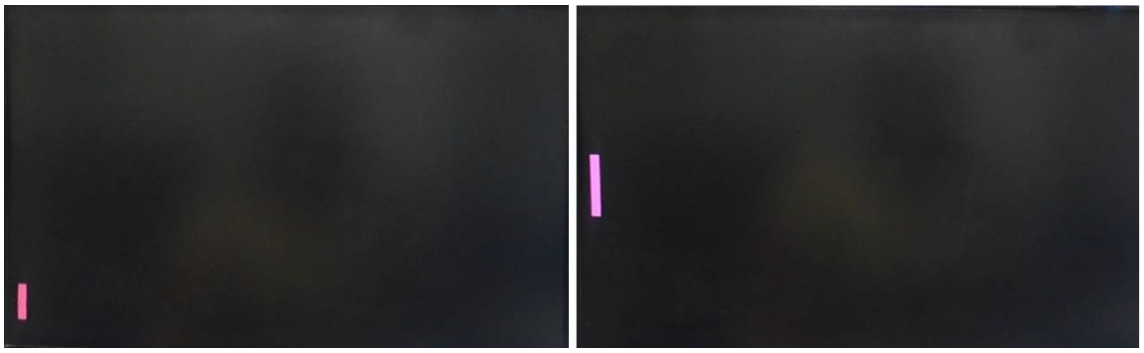


Figura 4.9 *Resultado na tela da bola batendo na barra.*

- CONTROLE DE OUTROS OBJETOS

No trecho do código no quadro abaixo, é feito o controle na tela de qualquer outro objeto que não seja a bola ou a barra.

Quadro 4.23 *Controle de outros objetos*

```
-- controla outros objetos
        IF( hpos < 1688 ) THEN
            hpos <= hpos + 1;
        ELSE
            hpos <= 0;

            IF( vpos < 1066 ) THEN
                vpos <= vpos + 1;
            ELSE
                vpos <= 0;
        END IF;
          END IF;

        IF( hpos > 48 AND hpos < 160 ) THEN
            hsync <= '0';
        ELSE        hsync <= '1';
        END IF;

        IF( vpos > 0  and vpos < 4 ) THEN
            vsync <= '0';
        ELSE
            vsync <= '1';
        END IF;

        IF( (hpos > 0 AND hpos < 408) OR (vpos > 0 AND vpos < 42) ) THEN
                r <= (OTHERS=>'0');
                g <= (OTHERS=>'0');
```

```
            b <= (OTHERS=>'0');
        END IF;
```

- ATUALIZAÇÃO DOS PONTOS

Para visualizar a contagem dos pontos feitos quando a bola bate na dead zone, foi inserido ao código os LEDs do display HEX0 de 7 segmentos da placa. Eles atualizaram a cada ponto feito, e quando o contador contar 9 pontos, o display zera novamente.

Quadro 4.24 *Atualização dos LEDs*

```
-- Atualiza o LED da contagm dos pontos, se a pontuação chegar a 9,o display zera e reinicia
a contagem.
                IF (points = 0) THEN
                        led<= "1000000";
                ELSIF (points = 1) THEN
                        led <= "1111001";
                ELSIF (points = 2) THEN
                        led<= "0100100";
                ELSIF (points = 3) THEN
                        led<= "0110000";
                ELSIF (points = 4) THEN
                        led <= "0011000";
                ELSIF (points = 5) THEN
                        led <= "0010010";
                ELSIF (points = 6) THEN
                        led <= "0000010";
                ELSIF (points = 7) THEN
                        led <= "1111001";
                ELSIF (points = 8) THEN
                        led <= "0000000";
                ELSIF (points = 9) THEN
                        led <= "0010000";
                        points <= 0;
                ELSE
                        led <= "1111111";
                END IF;
            END IF;
END PROCESS;
END MAIN;
```

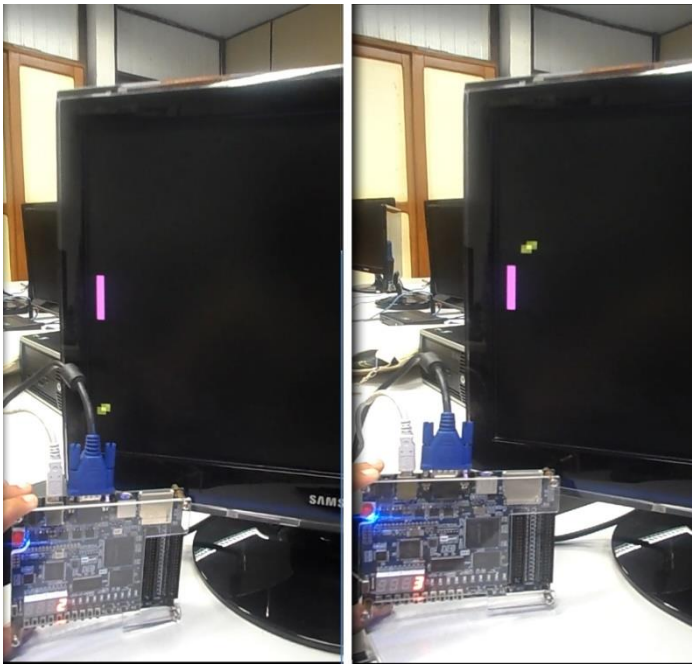


Figura 4.10 *Resultado na placa da atualização dos leds.*

4.1.3 Módulo VGA

No módulo VGA vamos fazer a conexão dos outros módulos, mais o código PLL que no capitulo 3 foi mostrado como gerar. Neste módulo declaramos entradas e saídas, que serão as saídas para monitor, às entradas e saídas do código VGA serão sincronizadas com as entradas e saídas do Módulo *SYNC* e do Código PLL gerado.

Quadro 4.25 *Declaração de entrada e saídas do código VGA.vhd.*

```
ENTITY vga IS
	PORT(
		clock_24 : IN std_logic_vector(1 downto 0);
		vga_hs, vga_vs : OUT std_logic;
		vga_r, vga_g, vga_b : OUT std_logic_vector(3 downto 0);
		key : IN std_logic_vector(1 downto 0);
		led: OUT std_logic_vector(6 downto 0)
	);
END vga;
```

No capítulo 3 foi mostrado como adicionar os componentes ao código VGA.

Quadro 4.26 *Declaração dos componentes.*

```
ARCHITECTURE main OF vga IS
	SIGNAL vgaclk, reset : std_logic := '0';
	-----------------------------------------------------
	COMPONENT pll1 IS
		PORT(
			clk_in_clk : IN std_logic := 'x'; -- clk
```

```
                reset_reset : IN std_logic := 'x'; -- reset
                clk_out_clk : OUT std_logic -- clk
        );
    END COMPONENT pll1;
    ------------------------------------------------
    COMPONENT sync IS
        PORT (
                clk: IN std_logic;
                hsync, vsync: OUT std_logic;
                r, g, b: OUT std_logic_vector (3 downto 0);
                keys: IN std_logic_vector (1 downto 0);
                led: OUT std_logic_vector (6 downto 0)
        );
    END COMPONENT sync;
```

Tanto o componente *SYNC* quanto o componente *PLL* terão suas entradas e saídas interligadas com as saídas e entradas declaradas no Módulo *VGA*.

Quadro 4.27 *Interligação das entradas e saídas*

```
BEGIN
    c1 : sync PORT MAP( vgaclk, vga_hs, vga_vs, vga_r, vga_g, vga_b, key, led);
   c2 : pll1 PORT MAP(clock_24(0), reset, vgaclk);
END main;
```

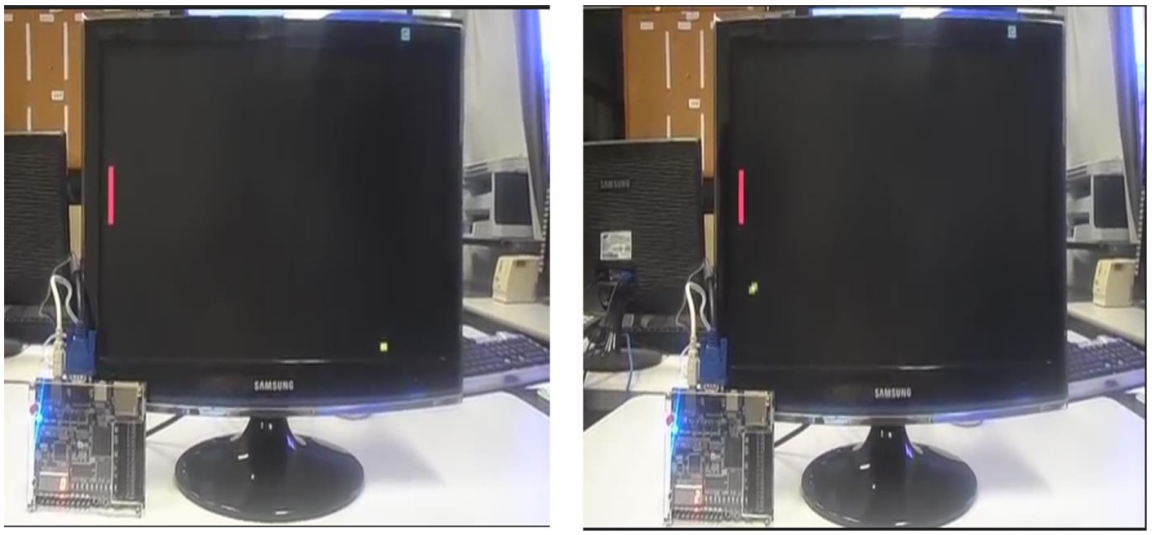


Figura 4.11 *Resultado final da aplicação*

Os códigos completos do módulo PCG, módulo SYNC e Módulo VGA e as conexões com os pinos, estão todos no Apêndice B.

A visualização da aplicação em funcionamento, pode ser feita pelo link: https://www.youtube.com/watch?v=V02AYNgLaLw&feature=youtu.be

5. CONCLUSÃO

O presente trabalho possibilitou a criação de um material da linguagem VHDL para o ensino de graduandos dos cursos de computação, com o intuito de introduzir nos currículos acadêmicos um novo modelo de computação, conhecido como computação reconfigurável. O material já foi utilizado em uma disciplina optativa do curso de bacharelado em Ciência da Computação na Universidade Federal do Pará. A disciplina foi recebida com muito entusiasmo pelos alunos, que demostraram bastante interesse em conhecer e entender melhor esse novo paradigma.

Com o decorrer da elaboração desse trabalho, foi percebida a carência do estudo desse e de outros modelos computacionais, conhecidos como não convencionais. O que de certa forma é bastante preocupando, pois, esses modelos não são mais o "futuro da computação" e sim nosso presente. Segundo (FIGUEIREDO *et al.*, 2008) "O grande desafio observado está no ponto em que esses modelos não convencionais passam a ser convencionais e se mostram presentes no nosso cotidiano. Se tratando de cursos inovadores e de conceito na área de computação no cenário nacional, vem a questão de quando os modelos não convencionais devem ser apresentados aos graduandos".

A computação evolui a cada dia, e o ensino dela na graduação tem a necessidade de evoluir em conjunto. Ensinar a alunos de graduação as novas vertentes da computação moderna, como esses modelos não convencionais são de suma importância na qualificação desses futuros profissionais.

A utilização da computação reconfigurável vem ganhando cada vez mais espaço. Com o aumento da utilização de dispositivos reconfiguráveis para a realização de operações específicas em sistemas embarcados, operações como o processamento de sinais, a realização de cálculos de algoritmos de criptografia, processamento de imagens, a codificação e decodificação de vídeos e até na utilização de processamento de áudio entre outras inúmeras possibilidades de utilização.

Além do ensino da programação paralela, que já é uma realidade não apenas nas universidades, como no mercado também por conta do avanço dos multinúcleos que há pouco tempo atrás não estava próximo à realidade computacional e atualmente cada vez mais as pessoas possuem processadores com mais de um núcleo, que caracteriza a não convenção do modelo paralelo avançando para os meios convencionais.

Por fim, esse trabalho não consiste apenas em um tutorial para o ensino do VHDL, ou uma introdução aos conceitos da computação reconfigurável. Esse trabalho deseja mostrar aos

alunos de graduação, essas novas vertentes da computação, com intuito de despertá-los para esse futuro, já tão presente no nosso cotidiano.

Ao final do trabalho foi implementado uma aplicação que tomou como base os conhecimentos adquiridos nos laboratórios, essa aplicação trata-se do jogo pong que muito conhecido na década de 70. Como trabalhos futuros pretende-se implementar o teclado PS2, para que tenha seu funcionamento reconhecido pela placa DE0, e fazer com que o *pong* possa ser jogado por dois jogadores, adicionando outra barra, e delimitando outra *dead zone*.

REFERÊNCIAS

ALTERA. Sobre a Empresa: Altera Corporation. **Laboratory Exercise 1 Switches Lights and Multiplexers**, 2006. Disponivel em: <ftp://ftp.altera.com/up/pub/Altera_Material/9.0/Digital_Logic/DE1/Laboratory_Exercises/VHDL/lab1_VHDL.pdf>. Acesso em: 20 Dezembro 2016.

ALTERA. Sobre a empresa: Altera Corporation. **Laboratory Exercise 2 - Numbers and Display**, 2008. Disponivel em: <ftp://ftp.altera.com/up/pub/Altera_Material/9.0/Digital_Logic/DE2-70/Laboratory_Exercises/VHDL/lab2_VHDL.pdf>. Acesso em: 20 Desembro 2016.

ALTERA. Sobre a Empresa: Altera Corporation. **Latches, Flip-flops, and Registers**, 2010. Disponivel em: <ftp://ftp.altera.com/up/pub/Altera_Material/10.0/Laboratory_Exercises/Digital_Logic/DE2-Series/Verilog/lab3_Verilog.pdf>. Acesso em: 20 Dezembro 2016.

ATHANAS, P. M.; SILVERMAN, H. F. Processor reconfiguration through instruction-set metamorphosis. **IEEE**, v. 26, n. 3, 1993.

BRETON, P. **Histoire de L'informatique**. Paris: La Découverte, 1987.

D'AMORE, R. **VHDL Descrição e Síntese de Circuitos Digitais**. 2ª. ed. [S.l.]: LTC, 2012.

DÉHARBE, D. **VHDL**. Universidade Federal do Rio Grande do Norte. [S.l.]. 1998.

FIGUEIREDO, C. et al. **Computação Não Convencional: aprendizado de modelos e arquiteturas não convencionais na graduação**. Pontifícia Universidade Católica de Minas Gerais. Minas Gerais. 2008.

HARTENSTEIN, R. **Why we need Reconfigurable Computing Education:** Introduction. 1st International Workshop on Reconfigurable Computing Education. [S.l.]: [s.n.]. 2006.

IOULIIA SKLIAROVA, A. B. F. Introdução à computação reconfigurável. **REVISTA DO DETUA**, v. IV, n. 1, setembro 2003.

LINHARES, R. R. **Comtribuição para o desenvolvimento de uma arquitetura de computação própria ao paradigma orientado a notificações**. Universidade Tecnológica Federal do Paraná. Curitiba, p. 354. 2015.

PERRY, D. L. **VHDL:** Programming by Example. 4ª. ed. [S.l.]: McGraw-Hill, 2002.

REDAELLI, F.; SANTAMBROGI, M. D.; SCIUTO, D. **Task scheduling with configuration perfetching an anti-fragmentation techniques on dynamically reconfigurable systems**. Proceeding of the conference on Design, automation and test in Europe. New York: USA. ACM. 2008. p. 519-522.

SOUZA, A. R. **Desenvolvimento e Implementação em FPGA de um sistema portátil para a aquisição e compressão sem perdas de eletrocardiogramas**. Universidade Federal da Paraíba. João Pessoa, p. 180. 2008.

TANENBAUM, A. S. **Organização estruturada de computadores**. 6ª. ed. São Paulo: Pearson, 2013.

TERASIC TECHNOLOGIES. Sobre a Empresa: Terasic Technologies. **DE2 Development and Education Board User Manual**, 2011. Disponivel em: <https://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&CategoryNo=56&No=364&PartNo=4>. Acesso em: 20 Dezembro 2016.

APÊNDICE A

A.1- Códigos usados no laboratório 1

Código 01 – Conexão das chaves aos LEDS na placa DE0..

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

-- Módulo que conecta as chaves SW as luzes LEDG
ENTITY Lab1Parte1 IS
        PORT( SW   : IN  std_logic_vector(9 DOWNTO 0);
              LEDG : OUT std_logic_vector(9 DOWNTO 0)  );
END Lab1Parte1;

ARCHITECTURE Behavior OF Lab1Parte1 IS
BEGIN
        LEDG <= SW;
END Behavior;
```

Código 02 – Multiplexador 2-to-1 em VHDL.

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
-- Módulo que descreve um multiplexador 2-to-1
ENTITY Lab1Parte1 IS
        PORT( x, y, s : IN std_logic; m : OUT std_logic );
END Lab1Parte1;
ARCHITECTURE Behavior OF Lab1Parte1 IS
BEGIN
        m <= (NOT (s) AND x) OR (s AND y);
END Behavior;
```

Código 03 – Multiplexador 2-to-1 com entrada e saída de vetores.

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
-- Módulo que descreve um multiplexador 2-to-1 para um vetor de entrada
ENTITY Lab1Parte2 IS
        PORT( X, Y : IN  std_logic_vector(2 DOWNTO 0);
                    s : IN  std_logic;
                   M : OUT std_logic_vector(2 DOWNTO 0) );
END Lab1Parte2;
ARCHITECTURE Behavior OF Lab1Parte2 IS
BEGIN
        M(0) <= (NOT (s) AND X(0)) OR (s AND Y(0));
        M(1) <= (NOT (s) AND X(1)) OR (s AND Y(1));
        M(2) <= (NOT (s) AND X(2)) OR (s AND Y(2));
END Behavior;
```

Código 04– Multiplexador de 2 bits 3-to-1

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
-- Módulo que descreve um multiplexador 3-to-1
ENTITY Lab1Parte3 IS
        PORT( u, v, w, s0, s1 : IN STD_LOGIC; m : OUT STD_LOGIC );
END Lab1Parte3;
ARCHITECTURE Behavior OF Lab1Parte3 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 Behavior;
```

Código 05 – Conversor 2 bits para 7 segmentos.

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

-- Módulo decodificador 2 bits para 7 segmentos
ENTITY Lab1Parte4 IS
        PORT(  c0, c1 : IN STD_LOGIC;
                D : OUT STD_LOGIC_VECTOR(6 DOWNTO 0) );
```

```
END Lab1Parte4;

ARCHITECTURE Behavior OF Lab1Parte4 IS
BEGIN
	proc1 : PROCESS( c0, c1) IS
	BEGIN
		IF (c1 = '0' AND c0 = '0') THEN
			D(0) <= '1';
			D(1) <= '0';
			D(2) <= '0';
			D(3) <= '0';
			D(4) <= '0';
			D(5) <= '1';
			D(6) <= '0';
		ELSIF (c1 = '0' AND c0 = '1') THEN
			D(0) <= '0';
			D(1) <= '1';
			D(2) <= '1';
			D(3) <= '0';
			D(4) <= '0';
			D(5) <= '0';
			D(6) <= '0';
		ELSIF (c1 = '1' AND c0 = '0') THEN
			D(0) <= '0';
			D(1) <= '0';
			D(2) <= '0';
			D(3) <= '0';
			D(4) <= '0';
			D(5) <= '0';
			D(6) <= '1';
		ELSE
			D(0) <= '1';
			D(1) <= '1';
			D(2) <= '1';
			D(3) <= '1';
			D(4) <= '1';
			D(5) <= '1';
			D(6) <= '1';
		END IF;
	END PROCESS proc1;
END Behavior;
```

Código06 – Conversor 2 bit para 7 segmentos com WHEN.

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

-- Módulo decodificador 2 bits para 7 segmentos
ENTITY Lab1Parte4 IS
	PORT(  C : IN STD_LOGIC_VECTOR(1 DOWNto 0);
		D : OUT STD_LOGIC_VECTOR(6 DOWNTO 0) );
END Lab1Parte4;
ARCHITECTURE Behavior OF Lab1Parte4 IS
BEGIN
	D <=    "0100001" WHEN C="00" ELSE
```

```
                "0000110" WHEN C="01" ELSE
                "1000000" WHEN C="10" ELSE
                "1111111" WHEN C="11";
END Behavior;
```

Código 07 – Multiplexador de caracteres com saída para display de 7 segmentos.

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

ENTITY Lab1Parte5 IS
        PORT( SW  : IN STD_LOGIC_VECTOR(9 DOWNTO 0);
                HEX0 : OUT STD_LOGIC_VECTOR(0 to 6));
END Lab1Parte5;
ARCHITECTURE Behavior OF Lab1Parte5 IS
        COMPONENT mux_2bit_3to1
                PORT( S, U, V, W : IN STD_LOGIC_VECTOR(1 DOWNTO 0);
                        M        : OUT STD_LOGIC_VECTOR( 1 DOWNTO 0));
        END COMPONENT;
        COMPONENT char_7seg
                PORT( C  : IN STD_LOGIC_VECTOR(1 DOWNTO 0);
                        Display : OUT STD_LOGIC_VECTOR(0 TO 6));
        END COMPONENT;
        SIGNAL M : STD_LOGIC_VECTOR(1 DOWNTO 0);
BEGIN
        M0 : mux_2bit_3to1 PORT MAP( SW(9 DOWNTO 8), SW(5 DOWNTO 4), SW(3 DOWNTO 2),
SW(1 DOWNTO 0), M);
        H0 : char_7seg PORT MAP( M, HEX0);
END Behavior;
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY mux_2bit_3to1 IS
        PORT ( S, U, V, W : IN STD_LOGIC_VECTOR(1 DOWNTO 0);
                M : OUT STD_LOGIC_VECTOR(1 DOWNTO 0));
END mux_2bit_3to1;
ARCHITECTURE Behavior OF mux_2bit_3to1 IS
BEGIN
        M <= U    WHEN S = "00" ELSE
                V    WHEN S = "01" ELSE
```

```
        W    WHEN S = "10" ELSE
       "11" WHEN S = "11";
END Behavior;
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY char_7seg IS
    PORT( C       : IN STD_LOGIC_VECTOR(1 DOWNTO 0);
          Display : OUT STD_LOGIC_VECTOR(0 TO 6));
END char_7seg;
ARCHITECTURE Behavior OF char_7seg IS
BEGIN
    Display <= "1000010" WHEN C = "00" ELSE
               "0110000" WHEN C = "01" ELSE
               "0000001" WHEN C = "10" ELSE
               "1111111" WHEN C = "11";
END Behavior;
```

A.2- Códigos usados no laboratório 2

Código 08 – Decodificador Binário Decimal.

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

ENTITY Lab2Parte1 IS
    PORT( SW : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
    HEX0 : OUT STD_LOGIC_VECTOR(6 DOWNTO 0));
END Lab2Parte1;

ARCHITECTURE Behavior OF Lab2Parte1 IS
BEGIN
    HEX0 <= "1000000" WHEN SW = "0000" ELSE
            "1111001" WHEN SW = "0001" ELSE
            "0100100" WHEN SW = "0010" ELSE
            "0110000" WHEN SW = "0011" ELSE
            "0011001" WHEN SW = "0100" ELSE
            "0010010" WHEN SW = "0101" ELSE
            "0000010" WHEN SW = "0110" ELSE
            "1111000" WHEN SW = "0111" ELSE
            "0000000" WHEN SW = "1000" ELSE
            "0010000" WHEN SW = "1001" ELSE
            "1111111";
END Behavior;
```

Código 09 – Circuito parcial do decodificar binário para decimal com 2 dígitos

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

ENTITY Lab2Parte2 IS
        PORT( V : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
                M : OUT STD_LOGIC_VECTOR(3 DOWNTO 0);
                z : OUT STD_LOGIC );
END Lab2Parte2;

ARCHITECTURE Behavior OF Lab2Parte2 IS
SIGNAL C : STD_LOGIC_VECTOR(2 DOWNTO 0);
SIGNAL temp : STD_LOGIC;
BEGIN
        temp <= V(3) AND (V(2) OR V(1));
        C(2) <= V(2) AND V(1);
        C(1) <= V(2) AND (NOT V(1));
        C(0) <= V(0) AND (V(2) OR V(1));
        M(0) <= ((NOT (temp) AND V(0)) OR (temp AND C(0)));
        M(1) <= ((NOT (temp) AND V(1)) OR (temp AND C(1)));
        M(2) <= ((NOT (temp) AND V(2)) OR (temp AND C(2)));
        M(3) <= ((NOT (temp) AND V(3)) OR (temp AND '0'));
        z <= temp;
                                                END Behavior;
```

Código 10 – Somador completo de 2 dígitos

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

ENTITY FullAdder IS
         PORT( a, b, ci : IN STD_LOGIC;
                  s, co : OUT STD_LOGIC );
END FullAdder;

ARCHITECTURE Behavior OF FullAdder IS
BEGIN
         s <= a XOR b XOR ci;
         co <= (a AND b) OR (ci and (a XOR b));

END Behavior;
```

Código 11 – Componente que reutiliza o somador completo de 2 bits.

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

ENTITY FBRC IS
        PORT( AF, BF : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
                 SF : OUT STD_LOGIC_VECTOR(3 DOWNTO 0);
                 cfo : OUT STD_LOGIC );
END FBRC;
ARCHITECTURE Behavior OF FBRC IS
COMPONENT FullAdder
        PORT( a, b, ci : IN STD_LOGIC;
                s, co : OUT STD_LOGIC );
END COMPONENT;
```

```
SIGNAL co0, co1, co2, co3 : STD_LOGIC := '0';
BEGIN
        FA0 : FullAdder PORT MAP ( a => AF(0), b => BF(0), ci => '0', s => SF(0), co => co0);
        FA1 : FullAdder PORT MAP ( a => AF(1), b => BF(1), ci => co0, s => SF(1), co => co1);
        FA2 : FullAdder PORT MAP ( a => AF(2), b => BF(2), ci => co1, s => SF(2), co => co2);
        FA3 : FullAdder PORT MAP ( a => AF(3), b => BF(3), ci => co2, s => SF(3), co => co3);
END Behavior;
```

Código 12 – Conversor BCD de 4 bits.

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

ENTITY BinTo7Seg IS
        PORT( V   : IN  STD_LOGIC_VECTOR(3 DOWNTO 0);
                HEX0, HEX1 : OUT STD_LOGIC_VECTOR(6 DOWNTO 0));
END BinTo7Seg;

ARCHITECTURE Behavior OF BinTo7Seg IS
SIGNAL C    : STD_LOGIC_VECTOR(2 DOWNTO 0);
SIGNAL temp : STD_LOGIC;
SIGNAL M    : STD_LOGIC_VECTOR(3 DOWNTO 0); SIGNAL z    : STD_LOGIC ;
BEGIN
        temp <= V(3) AND (V(2) OR V(1));
        C(2) <= V(2) AND V(1);
        C(1) <= V(2) AND (NOT V(1));
        C(0) <= V(0) AND (V(2) OR V(1));
        M(0) <= ((NOT (temp) AND V(0)) OR (temp AND C(0)));
        M(1) <= ((NOT (temp) AND V(1)) OR (temp AND C(1)));
        M(2) <= ((NOT (temp) AND V(2)) OR (temp AND C(2)));
        M(3) <= ((NOT (temp) AND V(3)) OR (temp AND '0'));

        z <= temp;

        HEX0 <= "1000000" WHEN M = "0000" ELSE
                "1111001" WHEN M = "0001" ELSE
                "0100100" WHEN M = "0010" ELSE
                "0110000" WHEN M = "0011" ELSE
                "0011001" WHEN M = "0100" ELSE
                "0010010" WHEN M = "0101" ELSE
                "0000010" WHEN M = "0110" ELSE
                "1111000" WHEN M = "0111" ELSE
                "0000000" WHEN M = "1000" ELSE
                "0010000" WHEN M = "1001" ELSE
                "1111111";
        HEX1 <= "1111001" WHEN z = '1' ELSE
                "1000000";
END Behavior;
```

A.3- Códigos usados no laboratório 3

Código 13 – Conversor BCD de 4 bits.

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

ENTITY Lab3Parte1 IS
        PORT( Clk, R, S : IN STD_LOGIC;
                Q   : OUT STD_LOGIC);
END Lab3Parte1;

ARCHITECTURE Estrutura OF  Lab3Parte1 IS
SIGNAL R_g, S_g, Qa, Qb : STD_LOGIC;
ATTRIBUTE keep : BOOLEAN;
ATTRIBUTE keep OF R_g, S_g, Qa, Qb :
SIGNAL IS true;


BEGIN
        R_g <= R AND Clk; S_g <= S AND Clk;
        Qa <= NOT (R_g OR Qb);
        Qb <= NOT (S_g OR Qa);
        Q <= Qa;
END Estrutura;
```

Código 14 – Conversor BCD de 4 bits com Latch D.

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY Lab3Parte2 IS
        PORT(Clk D: IN STD_LOGIC;
                Q : OUT STD_LOGIC);
END Lab3Parte2;
ARCHITECTURE Estrutura OF Lab3Parte2 IS
SIGNAL R_g, S_g, Qa, Qb : STD_LOGIC;
ATTRIBUTE keep : BOOLEAN;
ATTRIBUTE keep OF R_g, S_g, Qa, Qb :
SIGNAL IS true;
BEGIN
        R_g <= (NOT D) AND Clk; S_g <= D AND
        Clk;
        Qa <= NOT (R_g OR Qb); Qb <= NOT
        (S_g OR Qa);
        Q <= Qa;
END Estrutura;
```

Código 15 – Flip-floptipo D completo.

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

ENTITY ffD IS
  port(  CLK, RST, PRE, CE, D : IN STD_LOGIC;
                        Q, nQ : OUT STD_LOGIC );
END ENTITY ffD;

ARCHITECTURE Behavior OF ffD IS SIGNAL temp : STD_LOGIC;
BEGIN
PROCESS
(CLK) IS BEGIN
    -- IF rising_edge(CLK) THEN     -- Borda de subida
      IF falling_edge(CLK) THEN    -- Borda de descida
        IF (RST='1') THEN
          Q <= '0';
           nQ <= ‘1’;

        ELSIF (PRE = ‘1’) THEN
           Q <= '1';
           nQ <= '0';
        ELSIF (CE='1') THEN
           Q <= D;
         nQ <= NOT D:
        END IF;
      END IF;
END PROCESS;
END Behavior;
```

Código 16 – Registrador paralelo de 4 bits com flip-floptipo D.

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY reg4bit IS
        PORT (  regCLK, regRST, regCE : IN STD_LOGIC;
                Dreg : IN  STD_LOGIC_VECTOR(3 DOWNTO 0);
                Qreg : OUT STD_LOGIC_VECTOR(3 DOWNTO 0) );
END reg4bit;
ARCHITECTURE Behavior OF reg4bit IS
```

```
COMPONENT ffD
        PORT(   CLK, RST, PRE, CE, D : IN STD_LOGIC;
                Q, nQ : OUT STD_LOGIC );
END COMPONENT;
BEGIN
        FFD0 : ffD PORT MAP(D => Dreg(0), CLK => regCLK, RST => regRST, PRE => '0',
        Q => Qreg(0), CE => regCE );
        FFD1 : ffD PORT MAP(D => Dreg(1), CLK => regCLK, RST => regRST, PRE => '0',
        Q => Qreg(1), CE => regCE );
        FFD2 : ffD PORT MAP(D => Dreg(2), CLK => regCLK, RST => regRST, PRE => '0',
        Q => Qreg(2), CE => regCE );
        FFD3 : ffD PORT MAP(D => Dreg(3), CLK => regCLK, RST => regRST, PRE => '0',
        Q =>Qreg(3), CE => regCE );
                                        END Behavior;
```

Código 17 – Registrador de deslocamento de 4 bits com flip-floptipo D.

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

ENTITY shift4bit IS
        PORT(regCLK, regRST, regCE, Dreg : IN STD_LOGIC;
                Qreg : OUT STD_LOGIC_VECTOR(3 DOWNTO 0) );
END ENTITY shift4bit;

ARCHITECTURE Behavior OF shift4bit IS COMPONENT
ffD
        PORT(   CLK, RST, PRE, CE, D : IN  STD_LOGIC;
                Q, nQ : OUT STD_LOGIC );
END COMPONENT;
SIGNAL t0, t1, t2, t3 : STD_LOGIC := '0';
   BEGIN
        FFD0 : ffD PORT MAP(D => Dreg, CLK => regCLK, RST => regRST, PRE => '0',
        Q => t0, CE => regCE );
        FFD1 : ffD PORT MAP(D => t0,   CLK => regCLK, RST => regRST, PRE => '0',
        Q => t1, CE => regCE );
        FFD2 : ffD PORT MAP(D => t1,   CLK => regCLK, RST => regRST, PRE => '0',
        Q => t2, CE => regCE );
        FFD3 : ffD PORT MAP(D => t2,   CLK => regCLK, RST => regRST, PRE => '0',
        Q => t3, CE=> regCE );
```

```
        Qreg(0)   <=   t0;
        Qreg(1)   <=   t1;
        Qreg(2)   <=   t2;
        Qreg(3)   <=   t3;
   END Behavior;
```

Código 18 – Registrador de deslocamento de 8 bits.

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

ENTITY shift_register_top IS
      PORT ( clk : IN  std_logic; d   : IN  std_logic;
            led : OUT std_logic_vector(7 DOWNTO 0));
END shift_register_top;

ARCHITECTURE behavioral OF shift_register_top IS
SIGNAL clock_div : std_logic_vector(4 DOWNTO 0);
SIGNAL shift_reg : std_logic_vector(7 DOWNTO 0) := x"00";
BEGIN

-- clock divider
PROCESS (clk)
BEGIN
    IF (clk'event and clk = '1') THEN
       clock_div <= clock_div + '1';
    END IF;
END PROCESS;

-- shift register
PROCESS (clock_div(4))
BEGIN
    IF (clock_div(4)'event and clock_div(4) = '1')   THEN

        shift_reg(7)   <= d;
        shift_reg(6)       <=       shift_reg(7);
        shift_reg(5)       <=       shift_reg(6);
        shift_reg(4)       <=       shift_reg(5);
```

```
            shift_reg(3)        <=        shift_reg(4);
            shift_reg(2)        <=        shift_reg(3);
            shift_reg(1)        <=        shift_reg(2);
            shift_reg(0)   <=   shift_reg(1);
    END IF;
END PROCESS;
-- hook up the shift register bits to the leds
   led <= shift_reg;
END BEHAVIORAL;
```

A.4- Códigos usados no laboratório 4

Código 19 – Flip flop D

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

ENTITY flipflopD IS
        PORT( d, rst, clk : IN std_logic;
              q, qb : INOUT std_logic);
END flipflopD;
ARCHITECTURE Behavioral OF flipflopD IS
BEGIN
    PROCESS (clk, rst)
    BEGIN
        IF (rst='0') THEN
           q<='0';
        ELSE
        IF(clk='1' and clk' event) THEN
            IF (d='0') THEN
                q<='0';
            ELSE
                q<='1';
            END IF;
        END IF;
     END PROCESS;
qb<=NOT q;
```

```
END Behavioral;
```

Código 20 – Contador Up de 4 bits com Flip flop D.

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

   ENTITY updff IS
         PORT ( rst,clk : IN std_logic;
               q,qb    : INOUT std_logic_vector(3 downto 0));
END updff;

ARCHITECTURE Behavioral OF updff IS COMPONENT
flipflopD IS
         PORT( d,rst,clk: IN std_logic;
               q,qb: INOUT std_logic);
END COMPONENT;
SIGNAL a,b,c,d,e,f,g,h,i,j : std_logic;
BEGIN
      a<=NOT q(0);
      D1 : flipflopD PORT MAP(a,rst,clk,q(0),qb(0);

      b<=(q(0) XOR q(1));

      D2 : flipflopD PORT MAP(b,rst,clk,q(1),qb(1));
      c<= q(0) AND q(1)AND qb(2);
      d<= qb(0) OR qb(1);
      e<= q(2) AND d ;
       j<= c OR e;

       D3 : flipflopD PORT MAP(j,rst,clk,q(2),qb(2));
       f<= qb(0) OR qb(1) OR qb(2);
       g<= f AND q(3);
        h<= q(0) AND q(1) AND q(2) AND qb(3);
        i<=g OR h;

       D4 : flipflopD PORT MAP(i,rst,clk,q(3),qb(3));

END Behavioral;
```

Código 21 – Contador Up de 4 bits com flip flop D alternativo.

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
ENTITY cont4bitsUp IS
        PORT ( rst,clk : IN std_logic;
               q,qb    : INOUT std_logic_vector(3 DOWNTO 0));
END cont4bitsUp;

ARCHITECTURE Behavioral OF cont4bitsUp IS
BEGIN
        PROCESS (clk, rst)
        BEGIN
            IF ( rst = '1' ) THEN
                q <= "0000";
            ELSE
                IF( clk = '1' and clk'event) THEN
                    q <= q + 1;
                END IF;
            END IF;
        END PROCESS;
        qb <= NOT q;
END Behavioral;
```

Código 22 – Contador de N bits.

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
USE IEEE.STD_LOGIC_SIGNED.ALL;
ENTITY counterN IS
    GENERIC( n : natural := 4 );
     PORT ( clock     : IN STD_LOGIC;
            reset_n : IN STD_LOGIC;
            Q         : OUT STD_LOGIC_VECTOR(n-1 DOWNTO 0) );
END ENTITY;
ARCHITECTURE rtl OF counterN IS
SIGNAL VALUE : std_logic_vector(n-1 DOWNTO 0);
BEGIN
    PROCESS(clock, reset_n)
    BEGIN
        IF (reset_n = '0') THEN
                value <= (OTHERS => '0');
       ELSIF ((clock'event) AND (clock = '1')) THEN
```

```
                value <= value + 1;
    END IF;
        END PROCESS;
        Q <= value;
END rtl;
```

A.5- Códigos usados no laboratório 5

Código 23 – SYNC.vhd

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.NUMERIC_STD.ALL;
ENTITY SYNC IS
        PORT ( CLK : IN std_logic;
                HSYNC, VSYNC : OUT std_logic;
                R, G, B : OUT std_logic_vector(3 DOWNTO 0)
        );
END SYNC;
ARCHITECTURE MAIN OF SYNC IS
SIGNAL HPOS : integer range 0 TO 1688 := 0;
SIGNAL VPOS : integer range 0 TO 1066 := 0;
BEGIN
PROCESS (CLK) BEGIN
        IF( clk'event and clk='1' ) THEN
                IF( HPOS=1042 OR VPOS=554 ) THEN -- desenha o pixels.
R <= (OTHERS=>'1');
G <= (OTHERS=>'1');
B <= (OTHERS=>'1');
                ELSE
                        R <= (OTHERS=>'0');
                        G <= (OTHERS=>'0');
                        B <= (OTHERS=>'0');
                END IF;

                IF( HPOS < 1688 ) THEN
    HPOS <= HPOS + 1;
                ELSE
    HPOS <= 0;
    IF( VPOS < 1066 ) THEN
```

```
                VPOS <= VPOS + 1;
        ELSE
                VPOS <= 0;
        END IF;
                        END IF;
                        IF( HPOS > 48 AND HPOS < 160 ) THEN
                                HSYNC <= '0';
                        ELSE
                                HSYNC <= '1';
                        END IF;

                        IF( VPOS > 0  AND VPOS < 4 ) THEN
                                VSYNC <= '0';
                        ELSE
                                VSYNC <= '1';
                        END IF;

                        IF( (HPOS > 0 AND HPOS < 408) OR (VPOS > 0 AND VPOS < 42) )
THEN
R <= (OTHERS=>'0');
G <= (OTHERS=>'0');
B <= (OTHERS=>'0');
                        END IF;
                END IF;
        END PROCESS;
END MAIN;
```

Código 24 – VGA.vhd.

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

ENTITY VGA IS
        PORT (CLOCK_24 : IN std_logic_vector(1 downto 0);
                VGA_HS, VGA_VS : OUT std_logic;
                VGA_R, VGA_G, VGA_B : OUT std_logic_vector(3 DOWNTO 0));
END VGA;
ARCHITECTURE MAIN OF VGA IS
SIGNAL vgaclk, reset : std_logic:='0';
COMPONENT PLL1 IS
        PORT(CLK_IN_CLK : IN std_logic := 'X'; -- CLK
```

```
                RESET_RESET : IN std_logic := 'X'; -- RESET
                CLK_OUT_CLK : OUT std_logic -- CLK
        );
END COMPONENT PLL1;
COMPONENT SYNC IS
        PORT( CLK : IN  std_logic;
                HSYNC, VSYNC : OUT std_logic;
                R, G, B : OUT std_logic_vector(3 DOWNTO 0)
        );
END COMPONENT SYNC; BEGIN
    C1 : SYNC PORT MAP( VGACLK, VGA_HS, VGA_VS, VGA_R, VGA_G, VGA_B);    C2 : PLL1
    PORT MAP(CLOCK_24(0), RESET, VGACLK);
END MAIN;
```

Código 25 – pcg.vhd.

```
LIBRARY IEEE;
USE IEEE STD_LOGIC_1164.ALL;
USE IEEE NUMERIC_STD.ALL;
PACKAGE MY IS
        PROCEDURE SQ(
                                SIGNAL XCUR, YCUR, XPOS, YPOS : IN INTEGER;
                                SIGNAL RGB : OUT STD_LOGIC_VECTOR(3 DOWNTO 0);
                               SIGNAL DRAW : OUT STD_LOGIC
        );
END MY;
PACKAGE BODY MY IS
        PROCEDURE SQ(
                                SIGNAL XCUR, YCUR, XPOS, YPOS : IN INTEGER;
                                SIGNAL RGB : OUT STD_LOGIC_VECTOR(3 DOWNTO 0);
                               SIGNAL DRAW : OUT STD_LOGIC
        BEGIN
                              IF( XCUR > XPOS AND XCUR < (XPOS+100) AND YCUR > YPOS AND
YCUR <(YPOS+100) ) THEN
                RGB<=”1111”;
                        DRAW<=’1’;
                ELSE
                        DRAW<=’0’;
                END IF;
        END SQ;
END MY;
```

Código 26 – SYNC.vhd.

```
LIBRARY IEEE;

USE IEEE.STD_LOGIC_1164.ALL;

USE IEEE.NUMERIC_STD.ALL;

USE WORK.MY.ALL;


ENTITY  SYNC IS  PORT(

                                CLK : IN STD_LOGIC;

                                HSYNC , VSYNC : OUT STD_LOGIC;

                               R, G, B : OUT STD_LOGIC_VECTOR(3 DOWNTO 0));


END SYNC;
```

```
ARCHITECTURE MAIN OF SYNC IS
SIGNAL RGB : STD_LOGIC_VECTOR(3 DOWNTO 0);
SIGNAL DRAW : STD_LOGIC;
SIGNAL SQ_X1, SQ_Y1 : INTEGER RANGE 0 TO 1688 := 0;
SIGNAL HPOS : INTEGER RANGE 0 TO 1688 := 0;
SIGNAL VPOS : INTEGER RANGE 0 TO 1066 := 0; BEGIN
        SQ_X1 <= 1048;
        SQ_Y1 <= 554;
        SQ( HPOS, VPOS, SQ_X1, SQ_Y1, RGB, DRAW);
        PROCESS (CLK)
        BEGIN
                IF( CLK'EVENT AND CLK='1' ) THEN
                        -- Desenha um quadrado no meio da tela
                                IF( DRAW = '1' ) THEN

                                        R <= RGB;
                                        G <= RGB;
                                        B<=  RGB;
                                ELSE
                                        R <= (OTHERS=>'0');
                                        G<=  (OTHERS=>'0');
                                        B<=  (OTHERS=>'0');
                                END IF;
                        -- Desenha uma linha vertical e outra horizontal no meio da tela
                        --  IF( HPOS=1042 OR VPOS=554 ) THEN
                        --          R <= (OTHERS=>'1');
                        --          G <= (OTHERS=>'1');
                        --          B <= (OTHERS=>'1');
                        --  ELSE
                        --          R <= (OTHERS=>'0');
                        --          G <= (OTHERS=>'0');
                        --          B <= (OTHERS=>'0');
                        -- END IF;

                        IF( HPOS < 1688 ) THEN
                                HPOS <= HPOS + 1;
                        ELSE
```

```
                    HPOS <= 0;

                    IF( VPOS < 1066 ) THEN
                        VPOS <= VPOS + 1;
                    ELSE
                        VPOS <= 0;
                    END IF;
                END IF;

                IF( HPOS > 48 AND HPOS < 160 ) THEN
                    HSYNC <= '0';
                ELSE
                    HSYNC <= '1';
                END IF;

                IF( VPOS > 0  AND VPOS < 4 ) THEN
                    VSYNC <= '0';
                ELSE
                    VSYNC <= '1';
                END IF;

                IF( (HPOS > 0 AND HPOS < 408) OR (VPOS > 0 AND VPOS < 42) )
THEN
                    R <= (OTHERS=>'0');
                    G <= (OTHERS=>'0');
                    B <= (OTHERS=>'0');
                END IF;
            END IF;
        END PROCESS;
                                END MAIN;
```

Código 27 – VGA.VHD do quadrado.

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

ENTITY VGA IS
    PORT (CLOCK_24 : IN STD_LOGIC_VECTOR(1 downto 0);
          VGA_HS, VGA_VS : OUT STD_LOGIC;
          VGA_R, VGA_G, VGA_B : OUT STD_LOGIC_VECTOR(3 DOWNTO 0));
END VGA;

ARCHITECTURE MAIN OF VGA IS
```

```
SIGNAL VGACLK, RESET : STD_LOGIC:='0';
COMPONENT PLL1 IS
PORT(CLK_IN_CLK : IN STD_LOGIC := 'X'; -- CLK
        RESET_RESET : IN STD_LOGIC := 'X'; -- RESET
        CLK_OUT_CLK : OUT STD_LOGIC -- CLK
);
END COMPONENT PLL1;

COMPONENT SYNC IS
            PORT( CLK : IN  STD_LOGIC;
                    HSYNC, VSYNC : OUT STD_LOGIC;
                    R, G, B : OUT STD_LOGIC_VECTOR(3 DOWNTO 0)
                );
END COMPONENT SYNC; BEGIN
  C1 : SYNC PORT MAP( VGACLK, VGA_HS, VGA_VS, VGA_R, VGA_G, VGA_B);
 C2 : PLL1 PORT MAP(CLOCK_24(0), RESET, VGACLK);
END MAIN;
```

Código 28 – PCG.vhd

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

PACKAGE MY IS
            PROCEDURE SQ (SIGNAL Xcur, Ycur, Xpos, Ypos: IN INTEGER;
                                SIGNAL RGB: OUT STD_LOGIC_VECTOR (3 DOWNTO 0);
                                SIGNAL DRAW : OUT STD_LOGIC
                               );
            END MY;
PACKAGE BODY MY IS

            PROCEDURE SQ ( SIGNAL Xcur, Ycur, Xpos, Ypos : IN INTEGER;
                                SIGNAL RGB : OUT STD_LOGIC_VECTOR(3 DOWNTO 0);
                                SIGNAL DRAW : OUT STD_LOGIC ) IS
             BEGIN
                  IF ( Xcur > Xpos AND Xcur < (Xpos+100) AND Ycur > Ypos AND Ycur <
Ypos+100) ) THEN
                        RGB <= "1111";
                        DRAW <= '1';
                    ELSE
                         DRAW <= '0';
                    END IF;
              END SQ;
         END MY;
```

Código 29 – SYNC.vhd do jogo simples

```
LIBRARY IEEE;

USE IEEE.STD_LOGIC_1164.ALL;

USE IEEE.NUMERIC_STD.ALL;

USE WORK.MY.ALL;
```

```
ENTITY SYNC IS
        PORT (  CLK : IN STD_LOGIC;
                HSYNC , VSYNC : OUT STD_LOGIC;
                R, G, B : OUT STD_LOGIC_VECTOR(3 DOWNTO 0);
                KEYS : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
                S : IN STD_LOGIC_VECTOR(1 DOWNTO 0)
              );
END SYNC;
ARCHITECTURE MAIN OF SYNC IS
SIGNAL RGB: STD_LOGIC_VECTOR (3 DOWNTO 0);
SIGNAL DRAW1, DRAW2: STD_LOGIC;
SIGNAL SQ_X1, SQ_Y1: INTEGER RANGE 0 TO 1688: = 600;
SIGNAL SQ_X2, SQ_Y2: INTEGER RANGE 0 TO 1688: = 500;
SIGNAL HPOS: INTEGER RANGE 0 TO 1688: = 0;
SIGNAL VPOS: INTEGER RANGE 0 TO 1066: = 0;
BEGIN
    --SQ_X1 <= 1048;
    --SQ_Y1 <= 554;
     SQ (HPOS, VPOS, SQ_X1, SQ_Y1, RGB, DRAW1);
     SQ (HPOS, VPOS, SQ_X2, SQ_Y2, RGB, DRAW2);
PROCESS (CLK)
BEGIN
    IF (CLK'EVENT AND CLK='1' ) THEN
        IF (DRAW1 = '1' ) THEN
            IF(S(0) = '1' ) THEN
                R <= (OTHERS=>'1');
                G <= (OTHERS=>'0');
                B <= (OTHERS=>'0');
            ELSE
                R <= (OTHERS=>'1');
                G <= (OTHERS=>'1');
                B <= (OTHERS=>'1');
            END IF;
        END IF;

        IF ( DRAW2 = '1' ) THEN
```

```
      IF ( S(1) = '1' ) THEN
            R <= (OTHERS=>'1');
            G <= (OTHERS=>'0');
            B <= (OTHERS=>'0');
      ELSE
            R <= (OTHERS=>'1');
           G  <= (OTHERS=>'1');
            B <= (OTHERS=>'1');
      END IF;
 END IF;

 IF( DRAW1 = '0' AND DRAW2 = '0' ) THEN
      R <= (OTHERS=>'0');
      G <= (OTHERS=>'0');
      B <= (OTHERS=>'0');
 END IF;
 IF( HPOS < 1688 ) THEN
     HPOS <= HPOS + 1;
 ELSE
     HPOS <= 0;
IF( VPOS < 1066 ) THEN
    VPOS <= VPOS + 1
ELSE
    IF( S(0) = '1' ) THEN
        IF( KEYS(0) = '0' ) THEN
            SQ_X1 <= SQ_X1 + 5;
        END IF;
        IF( KEYS(1) = '0' ) THEN
            SQ_X1 <= SQ_X1 - 5;
        END IF;
        IF( KEYS(2) = '0' ) THEN
           SQ_Y1 <= SQ_Y1 + 5;
        END IF;
        IF( KEYS(3) = '0' ) THEN
             SQ_Y1 <= SQ_Y1 - 5;
        END IF;
   END IF;
```

```
            IF( S(1) = '1' ) THEN
                IF( KEYS(0) = '0' ) THEN
                    SQ_X2 <= SQ_X2 + 5;
                END IF;
                IF( KEYS(1) = '0' ) THEN
                    SQ_X2 <= SQ_X2 - 5;
                END IF;
                IF( KEYS(2) = '0' ) THEN
                    SQ_Y2 <= SQ_Y2 + 5;
                END IF;
                IF( KEYS(3) = '0' ) THEN
                    SQ_Y2 <= SQ_Y2 - 5;
                END IF;
            END IF;
            VPOS <= 0;
        END IF;
    END IF;
    IF( HPOS > 48 AND HPOS < 160 ) THEN
        HSYNC <= '0';
    ELSE
        HSYNC <= '1';
    END IF;
        IF( VPOS > 0 AND VPOS < 4 ) THEN
            VSYNC <= '0';
        ELSE
            VSYNC <= '1';
        END IF;
            IF( (HPOS > 0 AND HPOS < 408) OR (VPOS > 0 AND VPOS < 42) ) THEN
                R <= (OTHERS=>'0');
                G <= (OTHERS=>'0');
                B <= (OTHERS=>'0');
            END IF;
    END IF;
  END PROCESS;
END MAIN;
```

Código 30 – VGA.vhd jogo simples

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

ENTITY VGA IS
        PORT ( CLOCK_24 : IN STD_LOGIC_VECTOR(1 downto 0);
               VGA_HS, VGA_VS : OUT STD_LOGIC;
               VGA_R, VGA_G, VGA_B : OUT STD_LOGIC_VECTOR(3 DOWNTO 0);
               KEY : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
               SW : IN STD_LOGIC_VECTOR(1 DOWNTO 0)
                );
END VGA;

ARCHITECTURE MAIN OF VGA IS
SIGNAL VGACLK, RESET : STD_LOGIC:='0';

COMPONENT PLL1 IS
        PORT ( CLK_IN_CLK : IN STD_LOGIC := 'X'; -- CLK
               RESET_RESET : IN STD_LOGIC := 'X'; -- RESET
               CLK_OUT_CLK : OUT STD_LOGIC -- CLK
              );
END COMPONENT PLL1;

COMPONENT SYNC IS
        PORT ( CLK : IN STD_LOGIC;
               HSYNC, VSYNC : OUT STD_LOGIC;
               R, G, B : OUT STD_LOGIC_VECTOR(3 DOWNTO 0);
               KEYS : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
               S : IN STD_LOGIC_VECTOR(1 DOWNTO 0)
             );
END COMPONENT SYNC;
BEGIN
    C1 : SYNC PORT MAP( VGACLK, VGA_HS, VGA_VS, VGA_R, VGA_G, VGA_B, KEY, SW);
    C2 : PLL1 PORT MAP(CLOCK_24(0), RESET, VGACLK);
END MAIN;
```

APÊNDICE B

CÓDIGOS COMPLETOS E PINOS USADOS PARA A CONEXÃO DA APLICAÇÃO.

B.1- Código modulo PCG

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
PACKAGE MY IS
        PROCEDURE BOLA(
            SIGNAL Xcur, Ycur, Xpos, Ypos : IN INTEGER;
            SIGNAL RGB : OUT STD_LOGIC_VECTOR(3 DOWNTO 0);
            SIGNAL DRAW : OUT STD_LOGIC
);

-- O procedimento para a barra é praticamente igual ao procedimento para a bola, com a
-- única diferença sendo que o procedimento da barra possui dois parâmetros adicionais,
-- sendo eles o tamanho vertical e horizontal.

        PROCEDURE BARRA(
             SIGNAL Xcur, Ycur, Xpos, Ypos, TAMANHO_HORIZONTAL,
TAMANHO_VERTICAL : IN INTEGER;
             SIGNAL RGB : OUT STD_LOGIC_VECTOR(3 DOWNTO 0);
             SIGNAL DRAW : OUT STD_LOGIC
);

END MY;
-- PACOTES QUE DEFINEM AS DIMENSÕES DA BOLA E DA BARRA
PACKAGE BODY MY IS
         PROCEDURE BOLA(
                  SIGNAL Xcur, Ycur, Xpos, Ypos : IN INTEGER;
                  SIGNAL RGB : OUT STD_LOGIC_VECTOR(3 DOWNTO 0);
                  SIGNAL DRAW : OUT STD_LOGIC ) IS

                  BEGIN
                           IF( Xcur > Xpos AND Xcur < (Xpos+25) AND Ycur > Ypos AND Ycur <
                           (Ypos+25) ) THEN
                                    RGB <= "1111";
                                    DRAW <= '1';
                           ELSE
                                    DRAW <= '0';
                           END IF;
         END BOLA;

         PROCEDURE BARRA(
                  SIGNAL Xcur, Ycur, Xpos, Ypos, TAMANHO_HORIZONTAL,
TAMANHO_VERTICAL : IN INTEGER;
                  SIGNAL RGB : OUT STD_LOGIC_VECTOR(3 DOWNTO 0);
                  SIGNAL DRAW : OUT STD_LOGIC ) IS

                  BEGIN
                           IF( Xcur > Xpos AND Xcur < (Xpos+TAMANHO_HORIZONTAL) AND
Ycur > Ypos AND Ycur <
```

```
                        (Ypos+TAMANHO_VERTICAL) ) THEN
                                RGB <= "1111";
                                DRAW <= '1';
                        ELSE
                                DRAW <= '0';
                        END IF;
        END BARRA;
END MY;
```

B.2- Código modulo SYNC

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.NUMERIC_STD.ALL;
USE WORK.MY.ALL;
ENTITY SYNC IS
        PORT(
                CLK : IN STD_LOGIC;
                HSYNC , VSYNC : OUT STD_LOGIC;
                R, G, B : OUT STD_LOGIC_VECTOR(3 DOWNTO 0);
                KEYS : IN STD_LOGIC_VECTOR(1 DOWNTO 0);
                LED: OUT STD_LOGIC_VECTOR(6 DOWNTO 0)
        );
END SYNC;
ARCHITECTURE MAIN OF SYNC IS
-- Y = HORIZONTAL, X= VERTICAL
-- SINAIS ATRIBUIDOS PARA AS FUNCIONALIDADES DA BOLA
-- MOVIMENTO, ESPAÇO AONDE ELA DEVE BATER.
        SIGNAL RGB : STD_LOGIC_VECTOR(3 DOWNTO 0);
        SIGNAL DRAWBOLA : STD_LOGIC;
        SIGNAL X_BOLA, Y_BOLA : INTEGER RANGE 0 TO 1688 := 600;
        SIGNAL HPOS : INTEGER RANGE 0 TO 1688 := 0;
        SIGNAL VPOS : INTEGER RANGE 0 TO 1066 := 0;
        SIGNAL HBOLA : INTEGER RANGE 0 TO 1688 := 0;
        SIGNAL VBOLA : INTEGER RANGE 0 TO 1066 := 0;
        SIGNAL MY : INTEGER RANGE -10 TO 10 := 5;-- MOVIMENTO DA BOLA
        SIGNAL MX : INTEGER RANGE -10 TO 10 := 5;-- MOVIMENTO DA BOLA
-- SINAIS ATRIBUIDOS PARA AS FUNÇÕES DA BARRA
        SIGNAL RGB_BARRA : STD_LOGIC_VECTOR(3 DOWNTO 0);
```

```
        SIGNAL X_BARRA, Y_BARRA : INTEGER RANGE 0 TO 1670 := 440; -- DETERMINA
O ESPAÇO QUE A BARRA VAI FICAR NA TELA, DIREITO OU ESQUERDO (NO CASO DO
JOGO O ESQUERDO)
        SIGNAL DRAWBARRA : STD_LOGIC;
        SIGNAL HBARRA : INTEGER RANGE 0 TO 1688 := 0;
        SIGNAL VBARRA : INTEGER RANGE 0 TO 1066 := 0;
-- SINAIS ATRIBUIDOS PARA MARCAÇÃO DOS PONTOS E DIMINUIÇÃO DA BARRA.
        SIGNAL POINTS : INTEGER RANGE 0 TO 100 := 0;
        SIGNAL TAMANHO_HORIZONTAL : INTEGER RANGE 0 TO 100 := 20;
        SIGNAL TAMANHO_VERTICAL : INTEGER RANGE 0 TO 200 := 190;
        SIGNAL NUM_VEZES_TOQUE_BARRA : INTEGER RANGE 0 TO 2000 := 0;
        SIGNAL VELOCIDADE_BOLA : INTEGER RANGE 0 TO 50 := 10;
BEGIN
BOLA( HBOLA, VBOLA, X_BOLA, Y_BOLA, RGB, DRAWBOLA);
BARRA(HBARRA, VBARRA, X_BARRA, Y_BARRA, TAMANHO_HORIZONTAL,
TAMANHO_VERTICAL, RGB_BARRA, DRAWBARRA);
PROCESS (CLK)
BEGIN
        IF( CLK'EVENT AND CLK='1') THEN
                -- Desenha um quadrado no meio da tela representando a BOLA do jogo
                IF( DRAWBOLA = '1') THEN
                        R <= (OTHERS=>'1');-- COR DA BOLA(AMARELA)
                        G <= (OTHERS=>'1');
                        B <= (OTHERS=>'1');
                ELSE
                        R <= (OTHERS=>'0');
                        G <= (OTHERS=>'0');
                        B <= (OTHERS=>'0');
                END IF;
                -- Desenha a barra
                IF( DRAWBARRA='1' ) THEN
                  R <= (OTHERS=>'1');-- COR DA BARRA(ROSA)
                        G <= (OTHERS=>'0');
                        B <= (OTHERS=>'1');
                END IF;
                -- Controla a BOLA
                IF( HBOLA < 1688 ) THEN
```

```
                HBOLA <= HBOLA + 1;
        ELSE
                HBOLA <= 0;

                IF( VBOLA < 1066 ) THEN
                        VBOLA <= VBOLA + 1;
                ELSE
                        -- Caso a bola toque na parte superior da tela
                        IF( Y_BOLA > (1066-50) ) THEN
                                MY <= - VELOCIDADE_BOLA;

                                Y_BOLA <= Y_BOLA + MY;
                                X_BOLA <= X_BOLA + MX:
                        -- Caso a bola toque na parte inferior da tela
                        ELSIF(Y_BOLA < 50) THEN
                                MY <= VELOCIDADE_BOLA;

                                Y_BOLA <= Y_BOLA + MY;
                                X_BOLA <= X_BOLA + MX;
                        -- Caso a bola toque na lateral direita da tela
                        ELSIF (X_BOLA > 1638) THEN
                                MX <= - VELOCIDADE_BOLA;

                                Y_BOLA <= Y_BOLA + MY;
                                X_BOLA <= X_BOLA + MX;
                                -- Caso a bola bata na barra
                        ELSIF (X_BOLA < 440 + TAMANHO_HORIZONTAL AND
Y_BOLA >= Y_BARRA - 50 AND Y_BOLA <= Y_BARRA + TAMANHO_VERTICAL) THEN
                                NUM_VEZES_TOQUE_BARRA <=
NUM_VEZES_TOQUE_BARRA + 1;

                                -- A cada 5 toques da bola na barra, aumentamos a velocidade
em 1 unidade.
                                IF (NUM_VEZES_TOQUE_BARRA > 5) THEN
                                        NUM_VEZES_TOQUE_BARRA <= 0;
                                        VELOCIDADE_BOLA <= VELOCIDADE_BOLA +
1;
```

```
                END IF;

                MX <= VELOCIDADE_BOLA;

                Y_BOLA <= Y_BOLA + MY;
                X_BOLA <= X_BOLA + MX;

                -- A cada toque na bola, reduzimos o tamanho da barra
                -- por 3 unidades, até o tamanho mínimo de 60.
                -- E QUANDO CHEGAR AO TAMANHO MÍNIMO o jogo
para.
                IF (TAMANHO_VERTICAL > 60) THEN
                TAMANHO_VERTICAL <= TAMANHO_VERTICAL - 3;
                                                        IF(
DRAWBARRA='1' ) THEN
                                        R <= (OTHERS=>'1');
                        G <= (OTHERS=>'1');
                        B <= (OTHERS=>'1');
                                END IF;
                END IF;

                -- Caso a bola toque a lateral esquerda da tela, caracterizando um
ponto.
                ELSIF (X_BOLA < 400) THEN
                        MX <= VELOCIDADE_BOLA;
                        MY <= VELOCIDADE_BOLA;
                        POINTS <= POINTS + 1;

                        Y_BOLA <= 550 + MY;
                        X_BOLA <= 1000 + MX;

                ELSE
                        Y_BOLA <= Y_BOLA + MY;
                        X_BOLA <= X_BOLA + MX;
                END IF;
                VBOLA <= 0;
        END IF;
```

```
                END IF;
-- CONTROLA A BARRA
        IF( HBARRA < 1688 ) THEN
                        HBARRA <= HBARRA + 1;
        ELSE
                        HBARRA <= 0;
                        IF( VBARRA < 1066 ) THEN
                                VBARRA <= VBARRA + 1;
                        ELSE
                                -- Comandos para movimentar a barra para cima e para baixo
                    IF((Y_BARRA)>=42 AND (Y_BARRA+190)<=1066) THEN -- limite da barra
na tela
                                  IF( KEYS(0) = '0' ) THEN-- movimento pra cima(button 2 da placa)
                                        Y_BARRA <= Y_BARRA + 10;
                                  END IF;
                                  IF( KEYS(1) = '0' ) THEN -- movimento para baixo(button 1 da
placa)
                                        Y_BARRA <= Y_BARRA - 10;
                                  END IF;
                                ELSE
                                  IF (Y_BARRA < 42) THEN
                                         Y_BARRA <= Y_BARRA + 10;
                                   ELSE
                                         Y_BARRA <= Y_BARRA - 10;
                                  END IF;
                                END IF;
                                VBARRA <= 0;
                        END IF;
        END IF;
                -- Controla outros objetos
                IF( HPOS < 1688 ) THEN
                        HPOS <= HPOS + 1;
                ELSE
                        HPOS <= 0;
                        IF( VPOS < 1066 ) THEN
                                VPOS <= VPOS + 1;
                        ELSE
```

```
                    VPOS <= 0;
                END IF;
            END IF;
            IF( HPOS > 48 AND HPOS < 160 ) THEN
                HSYNC <= '0';
            ELSE
                HSYNC <= '1';
            END IF;
            IF( VPOS > 0  AND VPOS < 4 ) THEN
                VSYNC <= '0';
            ELSE
                VSYNC <= '1';
            END IF;
            IF( (HPOS > 0 AND HPOS < 408) OR (VPOS > 0 AND VPOS < 42) ) THEN
                R <= (OTHERS=>'0');
                G <= (OTHERS=>'0');
                B <= (OTHERS=>'0');
            END IF;
        -- Atualiza o LED da contagm dos pontos, se a pontuação chegar a 9,
        --      o display zera e reinicia a contagem.
            IF (POINTS = 0) THEN
                LED(0) <= '0';
                LED(1) <= '0';
                LED(2) <= '0';
                LED(3) <= '0';
                LED(4) <= '0';
                LED(5) <= '0';
                LED(6) <= '1';
            ELSIF (POINTS = 1) THEN
                LED(0) <= '1';
                LED(1) <= '0';
                LED(2) <= '0';
                LED(3) <= '1';
                LED(4) <= '1';
                LED(5) <= '1';
                LED(6) <= '1';
            ELSIF (POINTS = 2) THEN
```

```
            LED(0) <= '0';
            LED(1) <= '0';
            LED(2) <= '1';
            LED(3) <= '0';
            LED(4) <= '0';
            LED(5) <= '1';
            LED(6) <= '0';
        ELSIF (POINTS = 3) THEN
            LED(0) <= '0';
            LED(1) <= '0';
            LED(2) <= '0';
            LED(3) <= '0';
            LED(4) <= '1';
            LED(5) <= '1';
            LED(6) <= '0';
        ELSIF (POINTS = 4) THEN
            LED(0) <= '1';
            LED(1) <= '0';
            LED(2) <= '0';
            LED(3) <= '1';
            LED(4) <= '1';
            LED(5) <= '0';
            LED(6) <= '0';
        ELSIF (POINTS = 5) THEN
            LED(0) <= '0';
            LED(1) <= '1';
            LED(2) <= '0';
            LED(3) <= '0';
            LED(4) <= '1';
            LED(5) <= '0';
            LED(6) <= '0';
        ELSIF (POINTS = 6) THEN
            LED(0) <= '0';
            LED(1) <= '1';
            LED(2) <= '0';
            LED(3) <= '0';
            LED(4) <= '0';
```

```
                LED(5) <= '0';
                LED(6) <= '0';
        ELSIF (POINTS = 7) THEN
                LED(0) <= '0';
                LED(1) <= '0';
                LED(2) <= '0';
                LED(3) <= '1';
                LED(4) <= '1';
                LED(5) <= '1';
                LED(6) <= '1';
        ELSIF (POINTS = 8) THEN
                LED(0) <= '0';
                LED(1) <= '0';
                LED(2) <= '0';
                LED(3) <= '0';
                LED(4) <= '0';
                LED(5) <= '0';
                LED(6) <= '0';
        ELSIF (POINTS = 9) THEN
                LED(0) <= '0';
                LED(1) <= '0';
                LED(2) <= '0';
                LED(3) <= '0';
                LED(4) <= '1';
                LED(5) <= '0';
                LED(6) <= '0';
                POINTS <= 0;
        ELSE
                LED(0) <= '1';
                LED(1) <= '1';
                LED(2) <= '1';
                LED(3) <= '1';
                LED(4) <= '1';
                LED(5) <= '1';
                LED(6) <= '1';
        END IF;
END IF;
```

```
END PROCESS;
END MAIN;
```

B.3 - Código modulo VGA

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

--componente principal, aonde interliga os outros pra saida na tela.
ENTITY VGA IS
	PORT(
		CLOCK_24 : IN STD_LOGIC_VECTOR(1 downto 0);
		VGA_HS, VGA_VS : OUT STD_LOGIC;
		VGA_R, VGA_G, VGA_B : OUT STD_LOGIC_VECTOR(3 DOWNTO 0);
		KEY : IN STD_LOGIC_VECTOR(1 DOWNTO 0);
		LED: OUT STD_LOGIC_VECTOR(6 DOWNTO 0)
	);
END VGA;

ARCHITECTURE MAIN OF VGA IS
	SIGNAL VGACLK, RESET : STD_LOGIC := '0';
	-----------------------------------------------------
	COMPONENT PLL1 IS
		PORT(
			CLK_IN_CLK : IN STD_LOGIC := 'X'; -- CLK
			RESET_RESET : IN STD_LOGIC := 'X'; -- RESET
			CLK_OUT_CLK : OUT STD_LOGIC -- CLK
		);
	END COMPONENT PLL1;
	-----------------------------------------------------
	COMPONENT SYNC IS
		PORT(
			CLK : IN STD_LOGIC;
			HSYNC, VSYNC : OUT STD_LOGIC;
			R, G, B : OUT STD_LOGIC_VECTOR(3 DOWNTO 0);
			KEYS : IN STD_LOGIC_VECTOR(1 DOWNTO 0);
			LED: OUT STD_LOGIC_VECTOR(6 DOWNTO 0)
		);
	END COMPONENT SYNC;

	BEGIN
	C1 : SYNC PORT MAP( VGACLK, VGA_HS, VGA_VS, VGA_R, VGA_G, VGA_B, KEY,
LED);
	C2 : PLL1 PORT MAP(CLOCK_24(0), RESET, VGACLK);
END MAIN;
```

B.4- Pinos para conexão com a placa

Sinal VHDL	**Pino**
*CLOCK*_24[0]	PIN_G21
VGA_B[3]	PIN_K15
VGA_B[2]	PIN_J22
VGA_B[1]	PIN_K21
VGA_G[3]	PIN_J21
VGA_G[2]	PIN_K17
VGA_G[1]	PIN_J17
VGA_G[0]	PIN_H22
VGA_HS	PIN_L21
VGA_R[3]	PIN_H21
VGA_R[2]	PIN_H20
VGA_R[1]	PIN_H17
VGA_R[0]	PIN_H19
VGA_VS	PIN_L22
KEY[1]	PIN_F1
KEY[0]	PIN_G3
LED[0]	PIN_E11
LED[1]	PIN_F11
LED[2]	PIN_H12
LED[3]	PIN_H13
LED[4]	PIN_G12
LED[5]	PIN_F12
LED[6]	PIN_F13

APÊNDICE C

PINOS DA PLACA DE0

C.1- Conexão entre as chaves da placa DE0 e os pinos do FPGA.

SIGNAL NAME	FPGA PIN No.	DESCRIPTION
SW[0]	PIN_J6	SLIDE SWITCH[0]
SW[1]	PIN_H5	SLIDE SWITCH[1]
SW[2]	PIN_H6	SLIDE SWITCH[2]
SW[3]	PIN_G4	SLIDE SWITCH[3]
SW[4]	PIN_G5	SLIDE SWITCH[4]
SW[5]	PIN_J7	SLIDE SWITCH[5]
SW[6]	PIN_H7	SLIDE SWITCH[6]
SW[7]	PIN_E3	SLIDE SWITCH[7]
SW[8]	PIN_E4	SLIDE SWITCH[8]
SW[9]	PIN_D2	SLIDE SWITCH[9]

C.2- Conexão entre as leds da placa DE0 e os pinos do FPGA.

SIGNAL NAME	FPGA PIN No.	DESCRIPTION
LEDG[0]	PIN_J1	LED GREEN[0]
LEDG[1]	PIN_J2	LED GREEN[1]
LEDG[2]	PIN_J3	LED GREEN[2]
LEDG[3]	PIN_H1	LED GREEN[3]
LEDG[4]	PIN_F2	LED GREEN[4]
LEDG[5]	PIN_E1	LED GREEN[5]
LEDG[6]	PIN_C1	LED GREEN[6]
LEDG[7]	PIN_C2	LED GREEN[7]
LEDG[8]	PIN_B2	LED GREEN[8]
LEDG[9]	PIN_B1	LED GREEN[9]

C.3- Conexão entre botões da placa DE0 e os pinos do FPGA.

SIGNAL NAME	FPGA PIN No.	DESCRIPTION
BUTTON[0]	PIN_H2	PUSHBUTTON[0]
BUTTON[1]	PIN_G3	PUSHBUTTON[1]
BUTTON[2]	PIN_F1	PUSHBUTTON[2]

C.4- Conexão entre os displays de 7 segmentos da placa DE0 e os pinos do FPGA.

SIGNAL NAME	FPGA PIN No.	DESCRIPTION
HEX0_D[0]	PIN_E11	SEVEN SEG. DIG. 0[0]
HEX0_D[1]	PIN_F11	SEVEN SEG. DIG. 0[1]
HEX0_D[2]	PIN_H12	SEVEN SEG. DIG. 0[2]
HEX0_D[3]	PIN_H13	SEVEN SEG. DIG. 0[3]
HEX0_D[4]	PIN_G12	SEVEN SEG. DIG. 0[4]

HEX0_D[5]	PIN_F12	SEVEN SEG. DIG. 0[5]
HEX0_D[6]	PIN_F13	SEVEN SEG. DIG. 0[6]
HEX0_DP	PIN_D13	SEVEN SEG. DEC POINT 0
HEX1_D[0]	PIN_A13	SEVEN SEG. DIG. 1[0]
HEX1_D[1]	PIN_B13	SEVEN SEG. DIG. 1[1]
HEX1_D[2]	PIN_C13	SEVEN SEG. DIG. 1[2]
HEX1_D[3]	PIN_A14	SEVEN SEG. DIG. 1[3]
HEX1_D[4]	PIN_B14	SEVEN SEG. DIG. 1[4]
HEX1_D[5]	PIN_E14	SEVEN SEG. DIG. 1[5]
HEX1_D[6]	PIN_A15	SEVEN SEG. DIG. 1[6]
HEX1_DP	PIN_B15	SEVEN SEG. DIC. POINT 1
HEX2_D[0]	PIN_D15	SEVEN SEG. DIG. 2[0]
HEX2_D[1]	PIN_A16	SEVEN SEG. DIG. 2[1]
HEX2_D[2]	PIN_B16	SEVEN SEG. DIG. 2[2]
HEX2_D[3]	PIN_E15	SEVEN SEG. DIG. 2[3]
HEX2_D[4]	PIN_A17	SEVEN SEG. DIG. 2[4]
HEX2_D[5]	PIN_B17	SEVEN SEG. DIG. 2[5]
HEX2_D[6]	PIN_F14	SEVEN SEG. DIG. 2[6]
HEX2_DP	PIN_A18	SEVEN SEG. DIC. POINT 2
HEX3_D[0]	PIN_B18	SEVEN SEG. DIG. 3[0]
HEX3_D[1]	PIN_F15	SEVEN SEG. DIG. 3[1]
HEX3_D[2]	PIN_A19	SEVEN SEG. DIG. 3[2]
HEX3_D[3]	PIN_B19	SEVEN SEG. DIG. 3[3]
HEX3_D[4]	PIN_C19	SEVEN SEG. DIG. 3[4]
HEX3_D[5]	PIN_D19	SEVEN SEG. DIG. 3[5]
HEX3_D[6]	PIN_G15	SEVEN SEG. DIG. 3[6]
HEX3_DP	PIN_G16	SEVEN SEG. DIC. POINT 3