UNIVERSIDADE CATÓLICA DE PELOTAS

TUTORIAL PARA PROTOTIPAÇÃO EM FPGA

PLACA DIDÁTICA DE0 DA ALTERA

LAMIPS – LABORATÓRIO DE MICROELETRÔNICA E PROCESSAMENTO DE SINAIS
Gustavo Ponzi Seibel

Orientador: Eduardo A. C. da Costa

Este tutorial apresenta exercícios e os passos necessários para realizar prototipações em FPGA utilizando o software QUARTUS II e a placa DE0 da fabricante Altera.

Conteúdo

Lista de Figuras

Lista de Tabelas

1. PLACA DE0

Figura 1 - Kit didático DE0

A placa didática DE0 (Figura 1) possui as ferramentas essenciais para usuários novatos ganharem conhecimento nas áreas de lógica digital e prototipação em FPGA. É equipado com o chip FPGA Altera Cyclone III da família de dispositivos 3C16, que oferece 15.408 LES (unidades lógicas), 346 pinos de I / O, e as demais características apresentadas na Tabela 1.

Tabela 1- Principais características da placa DE0

- **FPGA**
  - Cyclone III FPGA 3C16;
  - 15.408 Unidades lógicas;
  - 4 PLLs;
  - 346 pinos de I / O;
- **Memória**
  - 8 Mbyte memória RAM;
  - 4 Mbyte de memória FLASH;
  - Soquete para cartão SD;
- **Comunicação**
  - Comunicação USB (Driver USB-*BLASTER*);
- **Chaves/botões**
  - 3 botões de pressão (OBS: Quando pressionado nível lógico 0);
  - 10 chaves (OBS: Nível lógico 0 quando desl. e 1 quando ligado);
- **Interface com usuário**
  - 10 LEDs de cor verde (OBS: Ativo em nível lógico 1);
  - 4 displays de sete segmentos (OBS: Segmentos ativos em nível lógico 0);
- **Relógio (*CLOCK*)**
  - Oscilador de 50 MHz;
- **Saída VGA**
  - DAC de 4-bit;
  - Suporta resolução de 1280x1024 com freqüência de atualização de 60 Hz;
- **Portas seriais**

- Uma porta RS-232 (sem o conector);
- Uma porta PS / 2 (permite conexão de teclado e mouse);

➢ **Pinos de expansão**

- 72 pinos de I / O com 8 linhas de alimentação e terra (dois conectores de expansão de 40 pinos);
- Conexão de 40 pinos por cabo de fita utilizado em discos rígidos IDE.

Na figura 2 são identificados todos os periféricos do kit DE0 descritos na tabela1. Destaca-se que além da facilidade de interação que os interruptores, LEDs e displays proporcionam entre o usuário e a placa, é possível utilizar um LCD 16x2 segmentos na conexão indicada (OBS: O LCD não está incluso no kit DE0).

O ANEXO I apresenta as informações para a utilização dos periféricos da placa.

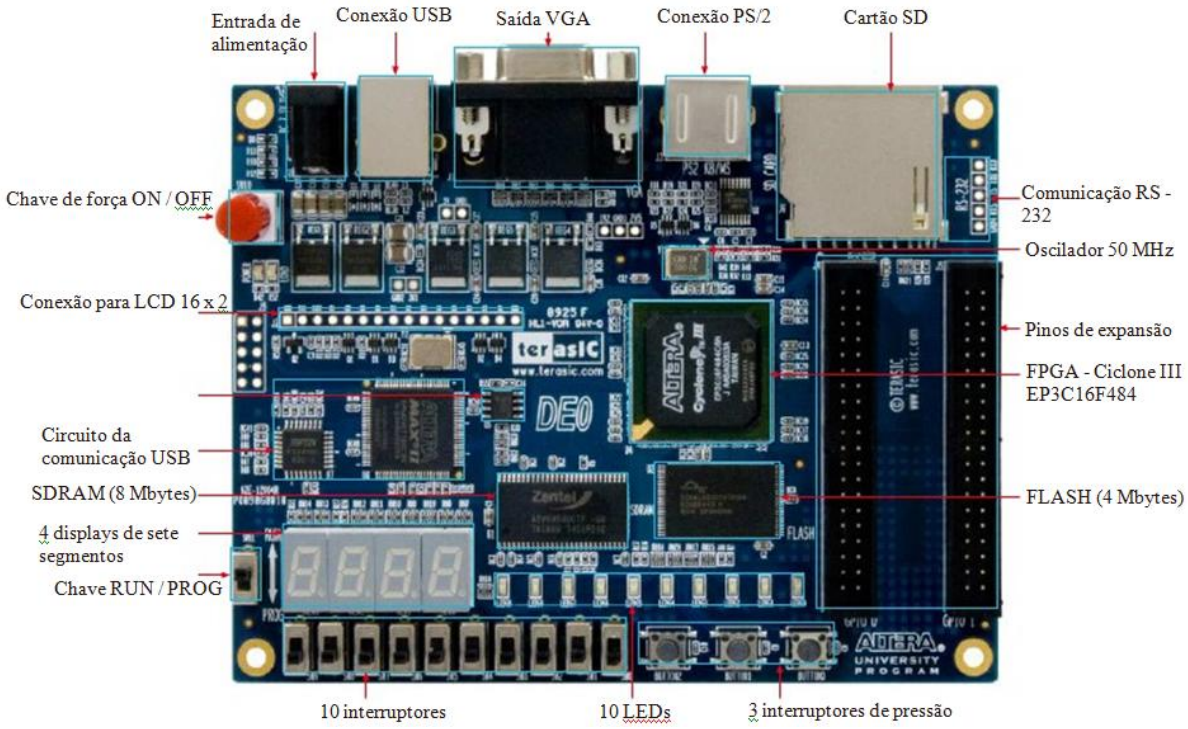

Figura 2 – Identificação dos periféricos na placa DE0

1.1. ITENS INCLUSOS NA CAIXA DE0

A placa DE0 é armazenada em uma caixa que inclui todas as peças necessárias para sua operação.

- ✓ **Placa DE0;**
- ✓ **Proteção de plástico para os circuitos da placa;**
- ✓ **Cabo USB para programação e controle;**
- ✓ **Cabo de alimentação (Entrada 110~220V automático, saída 7.5V DC);**

2. INSTALAÇÃO

O kit DE0 já vem com programação de fábrica que demonstra alguns recursos.

Após conectar o cabo de alimentação, colocar a chave RUN/PROG na posição RUN e pressionar o botão vermelho, os LEDs devem ficar alternando-se e os *displays* devem estar em contagem hexadecimal de 0 à F.

OBS: A comunicação com a placa DE0 apresenta problemas quando é utilizado o software Quartus II com versão inferior a 9.0, é recomendado a versão 9.0.

2.1. COMUNICAÇÃO USB

Para realizar a comunicação entre o kit e o computador é necessário instalar o *driver* USB *Blaster* da Altera. O *driver* encontra-se na pasta do software Quartus II instalado no computador.

Abaixo são apresentados os passos necessários para realizar a instalação do *driver*.

- **1° Passo:** Conectar o cabo USB na placa DE0 e no computador;
- **2° Passo:** Ligar a placa DE0;
- **3° Passo:** Esperar o computador reconhecer o novo *hardware*;
- **4° Passo:** Ao surgir a janela da figura 3 selecione a opção não procurar o *software* no *Windows Update* e clique em avançar;

Figura 3 - Janela do Windows (Novo *hardware* encontrado)

- **5° Passo:** Ao surgir a janela da figura 4 selecione a opção instalar a partir de um local específico e clique em avançar;

Figura 4 - Janela do Windows (Novo *hardware* encontrado)

- **6° Passo:** Ao abrir a janela da figura 5 selecione a opção incluir local na procura, busque no arquivo o endereço "**C:\altera\90\quartus\drivers\usb-blaster"** e clique em avançar (O endereço pode variar pois depende da versão do software Quartus II e do diretório que está instalado);

Figura 5 - Janela do Windows (Instalação do *driver*)

- **7° Passo:** O Windows irá emitir o aviso da figura 6, clique em continuar mesmo assim;

Figura 6 - Aviso do Windows

- **8° Passo:** Ao surgir a janela da figura 7 clique finalizar. A partir deste momento o *driver USB-Blaster* está instalado e a comunicação e programação da placa DE0 podem ser realizadas.

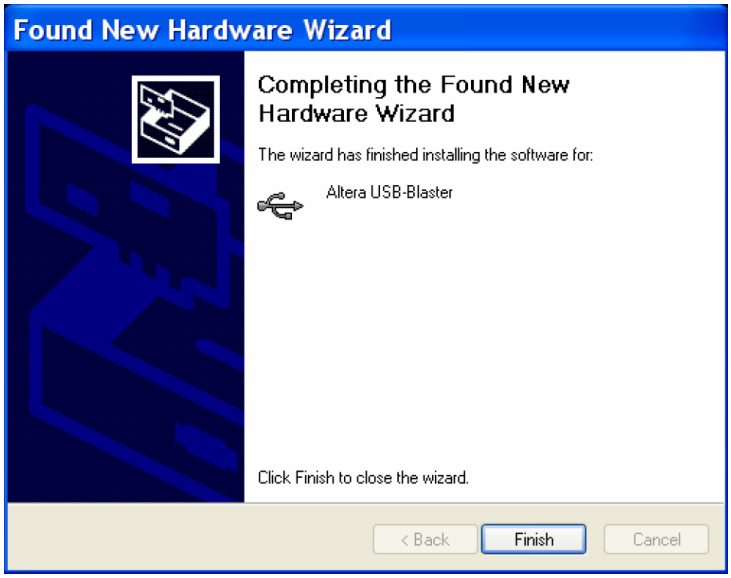

Figura 7 - Janela do Windows (Instalação competa)

Conhecido as características da placa DE0 e com o *driver* de comunicação USB instalado, pode-se começar a praticar a prototipação no FPGA da Altera.

3. PASSOS PARA A PROGRAMAÇÃO DO FPGA

Para realizar os exercícios e programar o FPGA são necessários os seguintes passos:

➔ CRIAR UM NOVO PROJETO NO SOFTWARE QUARTUS II E SELECIONAR A FAMÍLIA DO DISPOSITIVO FPGA UTILIZADO;
➔ IMPORTAR O ARQUIVO PARA ATRIBUIR OS SINAIS UTILIZADOS NOS RESPECTIVOS ENDEREÇOS DOS PINOS DO FPGA;
➔ DESCREVER O PROJETO E COMPILAR;
➔ PROGRAMAR O FPGA SELECIONADO.

Abaixo serão apresentados os passos para realizar estas configurações.

3.1. NOVO PROJETO - Selecionar o FPGA

- **1° Passo:** Inicialmente deve-se criar um novo projeto no Quartus II (Figura 8);

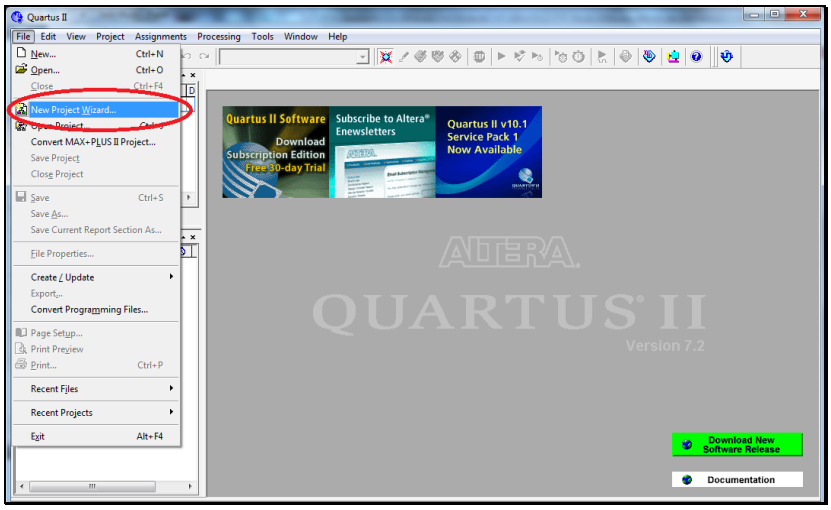

Figura 8 - Criando um novo projeto no Quartus II

- **2° Passo:** Ao abrir a janela da figura 9 selecionar o botão próximo;

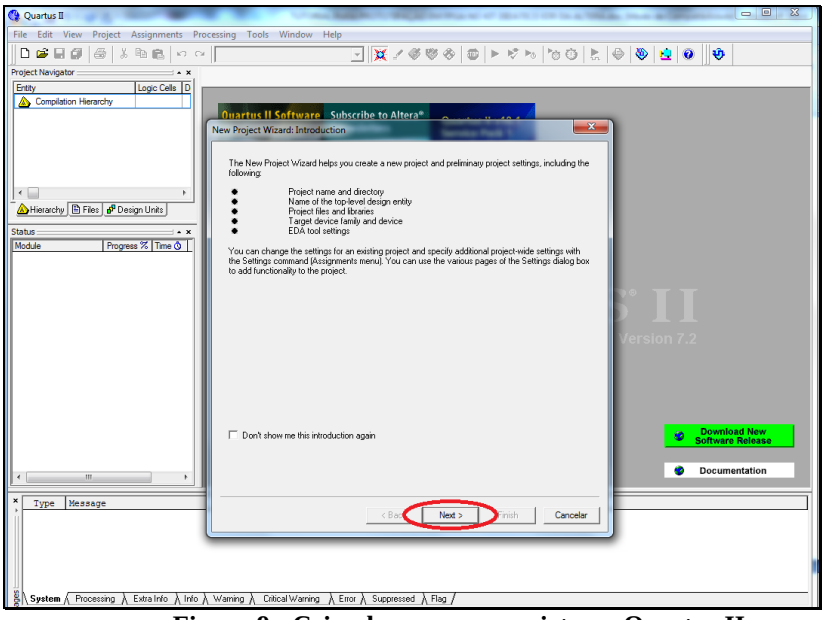

Figura 9 - Criando um novo projeto no Quartus II

- **3° Passo:** Nesta etapa é necessário selecionar o diretório para salvar e nomear o projeto, após clicar em próximo (Figura 10);

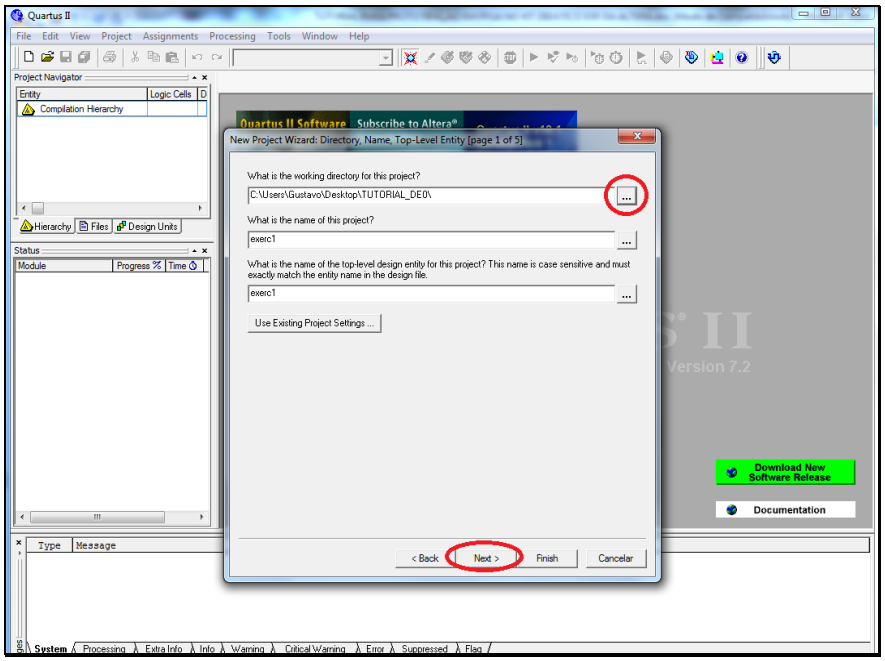

Figura 10 – Selecionar o diretório para salvar o projeto

- **4° Passo:** Ao abrir a janela da figura 11 apenas clicar em próximo;

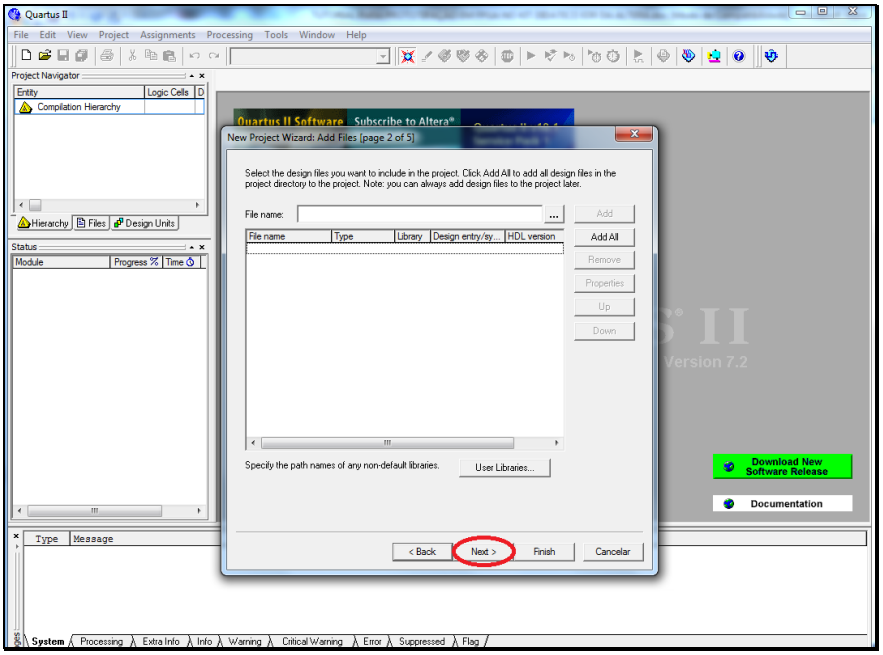

Figura 11 - Criando um novo projeto no Quartus II

5° Passo: Agora é necessário selecionar a família do FPGA (Cyclone III), escolher o dispositivo (EP3C16F484C6), e clicar em próximo como mostra a figura 12;

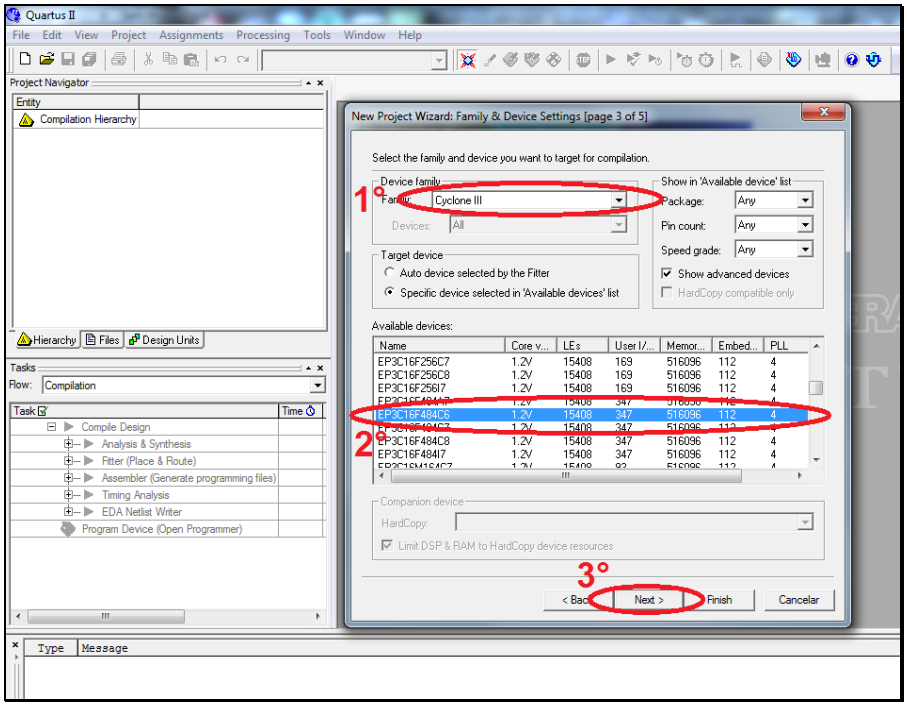

Figura 12 - Selecionar a família do FPGA e o dispositivo

- **6° Passo:** Ao abrir a janela da figura 13 clicar em final;

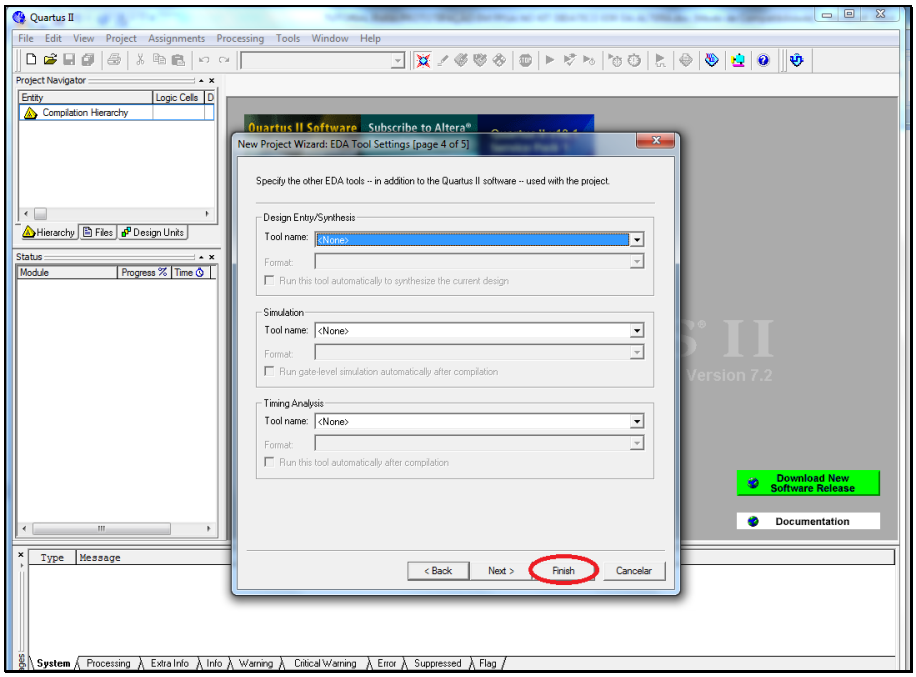

Figura 13 - Finalizando a criação de um novo projeto

3.2. IMPORTAR O ARQUIVO DE ATRIBUIÇÃO DOS PINOS

- **7° Passo:** Após a criação do novo projeto e especificação do FPGA a ser utilizado é necessário importar o arquivo que atribui os sinais utilizados nos respectivos endereços dos pinos do FPGA(Estes sinais são descritos no ANEXO I). Clique em "*Assignments*" e "*Import Assignments*" como mostra a figura 14;

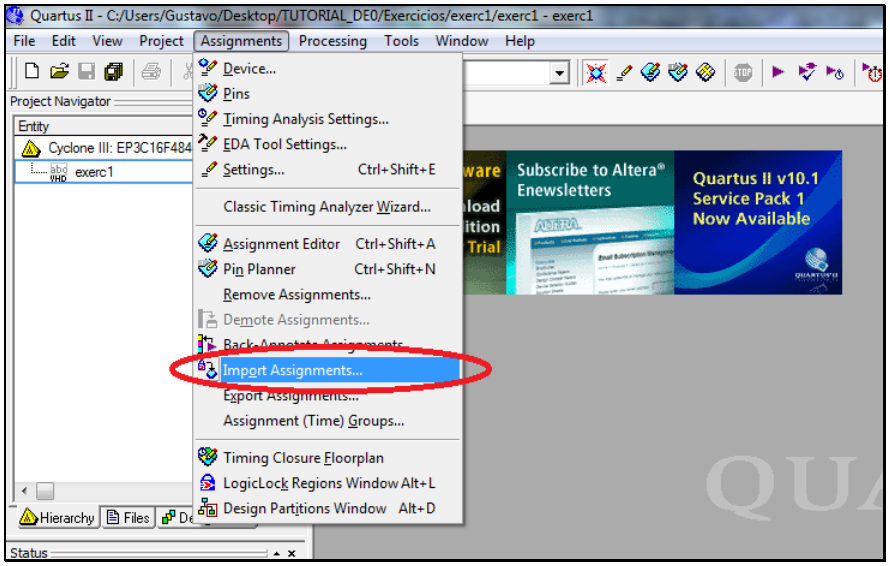

Figura 14 - Importação do arquivo de atribuição dos endereços dos pinos

- **8° Passo:** Ao abrir a janela "*Import Assignments*" da figura 15, clique para selecionar o diretório onde se encontra o arquivo DE0_pin_assignments.qsf (C:\DE0\ DE0_pin_assignments.qsf), após dar um duplo clique no arquivo selecionado, clicar em "OK";

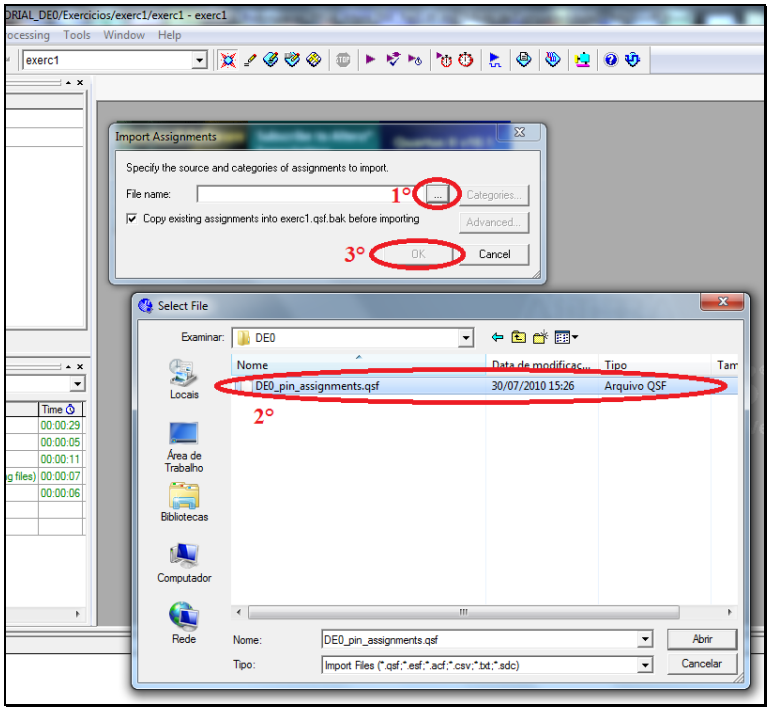

Figura 15 – Selecionar o arquivo de atribuição dos endereços dos pinos

3.3. DESCREVER O PROJETO E COMPILAR

- **9° Passo:** Ao descrever o hardware em VHDL, compilar e verificar se não há nenhum erro na compilação, pode-se iniciar a programação do FPGA da placa DE0. Sempre consultar o ANEXO I para utilizar corretamente os periféricos e seus respectivos sinais.

3.4. PROGRAMAR O FPGA

Após compilar o projeto pode-se programar a placa DE0, é importante certificar-se que o cabo USB está conectado, o botão RUN/PROG está na posição RUN (Consultar tipos de programação no ANEXO I), e o KIT ligado.

- **10° Passo:** Para programar o FPGA da placa DE0 é necessário clicar no botão "*Programmer*" em destaque na figura 16.

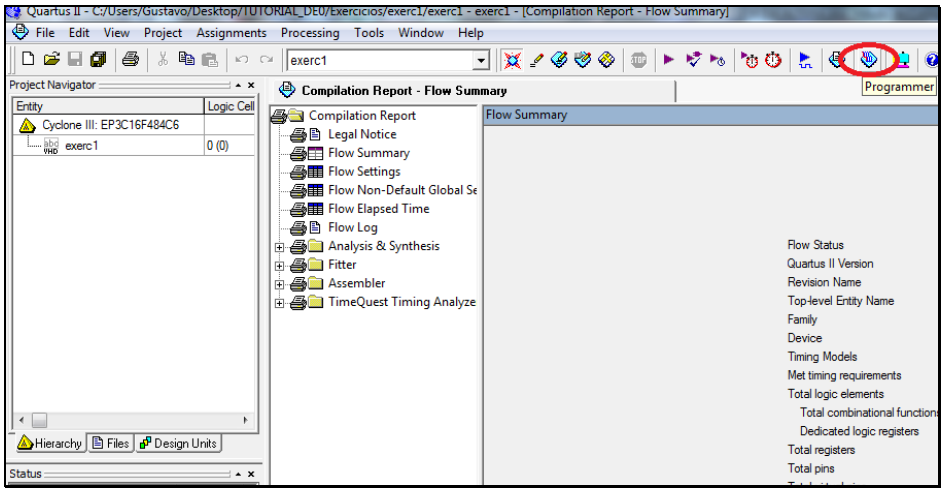

Figura 16 - Programando o FPGA

- **11° Passo:** Para selecionar a placa Altera conectada ao computador, clique no botão "Hardware Setup" em destaque na figura 17.

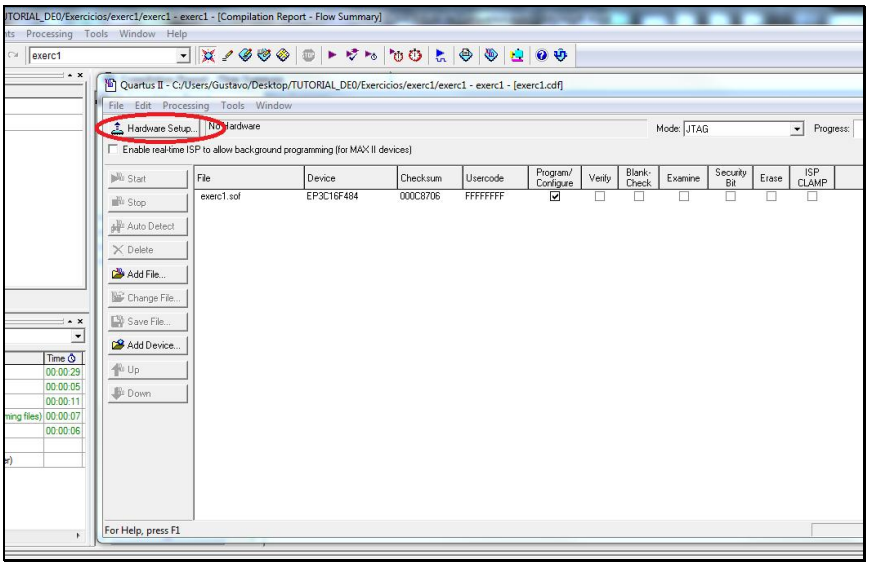

Figura 17 - Selecionando o hardware

- **12° Passo:** Se o *driver USB Blaster* estiver corretamente instalado e a placa DE0 ativa e conectada ao computador, deve surgir, na janela mostrada na figura 18, o *hardware* "*USB-Blaster*". É necessário selecioná-lo com duplo clique e fechar a janela.

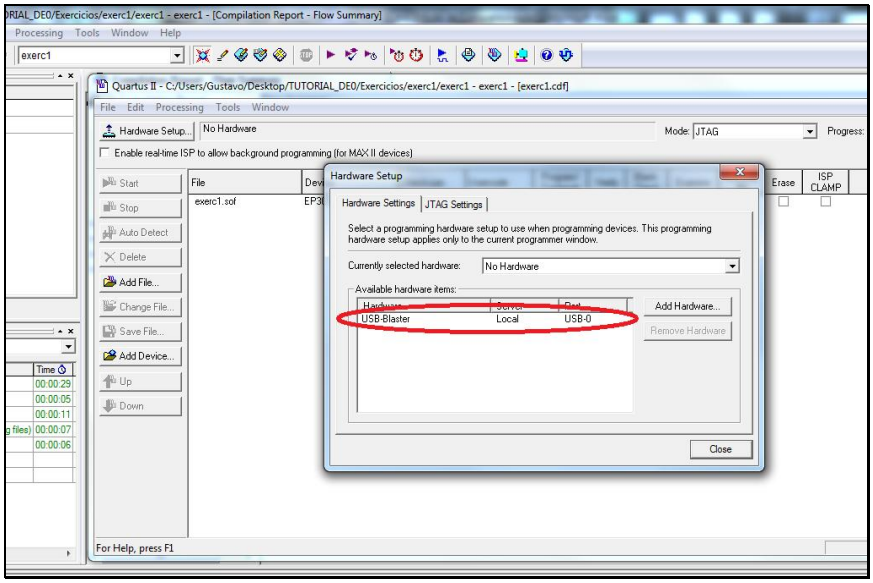

Figura 18 - Selecionando o hardware ativo

- **13° Passo:** Agora já se pode clicar no botão "Start" (Figura 19) para programar o FPGA da placa DE0.

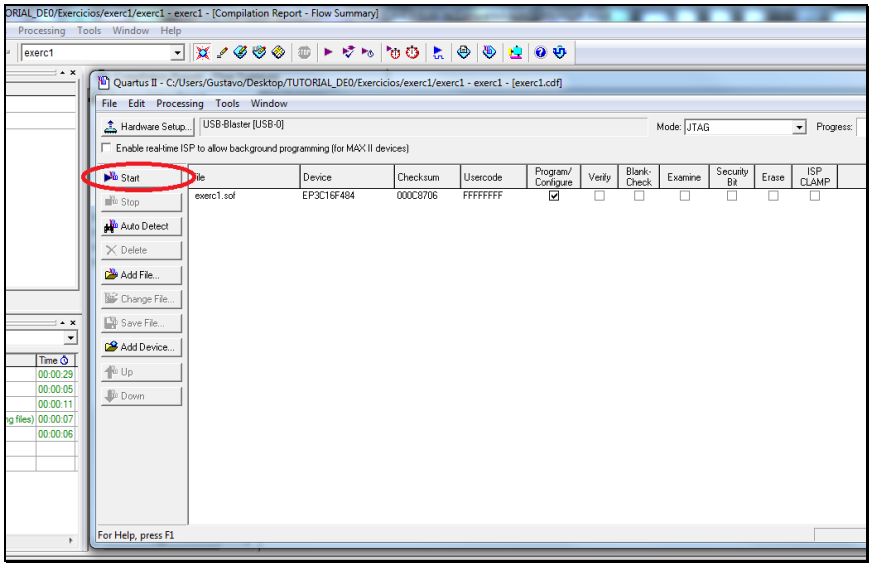

Figura 19 - Programando o FPGA

Com as etapas de instalação do *driver* de comunicação USB concluído e conhecido o processo de programação do FPGA é possível iniciar os exercícios propostos.

4. EXERCÍCIOS

Para realizar os exercícios é necessário ter terminado a etapa de instalação do *driver* de comunicação USB, ter conhecimento básico de programação VHDL e da ferramenta utilizada – o *software* Quartus II da Altera – e saber as características dos periféricos da placa no ANEXO I.

4.1. EXERCÍO 1 (Interruptor e LED)

O objetivo deste exercício é aprender como programar o *chip* FPGA utilizando periféricos da placa DE0 como entradas e saídas.

A placa DE0 possui 10 chaves denominadas SW_{9-0} que podem ser utilizados como entradas, e 10 emissores de luz ou LEDs, denominados $LEDG_{9-0}$, que podem ser utilizados como saídas.

Proposta: Propõe-se a conexão das entradas (chaves) nas saídas (LEDs), ou seja, quando selecionado o interruptor 4 o LED 4 deve acender, o mesmo para as outras chaves e LEDs.

OBS: Devem-se lembrar as características elétricas dos periféricos na tabela 1 ou no ANEXO I:
(chave ligada ➔ nível lógico 1 ➔ nível lógico 1 ➔ LED ligado).

Visto as características dos periféricos a serem utilizados necessita-se apenas direcionar o sinal do vetor das chaves para o sinal do vetor dos LEDs, o direcionamento dos vetores pode ser realizado bit a bit ou de forma direta como mostra a tabela 2.

Tabela 2 - Atribuição de vetores bit a bit ou direta

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

A tabela 3 mostra o código VHDL completo do exercício proposto. Para protipar o circuito descrito execute os passos apresentados no Capitulo 3.

Tabela 3 - Descrição em VHDL das conexões de interruptores e LEDs

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

-- Conexão dos interruptores nos LEDs
ENTITY exerc1 IS
PORT ( SW : IN STD_LOGIC_VECTOR(9 DOWNTO 0); --chaves
       LEDG : OUT STD_LOGIC_VECTOR(9 DOWNTO 0)); --LEDs
END exerc1;

ARCHITECTURE Behavior OF exerc1 IS
BEGIN
LEDG <= SW; -- direciona as chaves para os leds
END Behavior;
```

Lembre-se que as entradas e saídas declaradas na "*Top-level entity*" devem ter os mesmos nomes dos sinais endereçados na tabela DE0_pin_assignments.qsf, o ANEXO I contém mais detalhes sobre os sinais dos periféricos da placa DE0.

4.2. EXERCÍCIO 2 (Demultiplexador)

A proposta agora é utilizar a chave 0 para controlar os LEDs 0 a 7 para que fiquem alternando-se em seqüencia. Quando a chave 0 estiver na posição UP os LEDs 0 a 7 devem seguir uma seqüência crescente, e na posição DOWN, uma seqüência decrescente. O LED 9 deve estar ligado quando a seqüência for crescente e desligado quando for decrescente. Deve-se utilizar o CLOCK_50 para criar um sinal de clock interno com freqüência de 1Hz, ou seja, para que os LEDS fiquem se alternando a cada 1 segundo.

DICAS:

Pode-se utilizar um demultiplexador para ativar os LEDs 0 a 7 e um contador para endereçar o nível lógico alto para a saída do demultiplexador como mostra a figura 20.

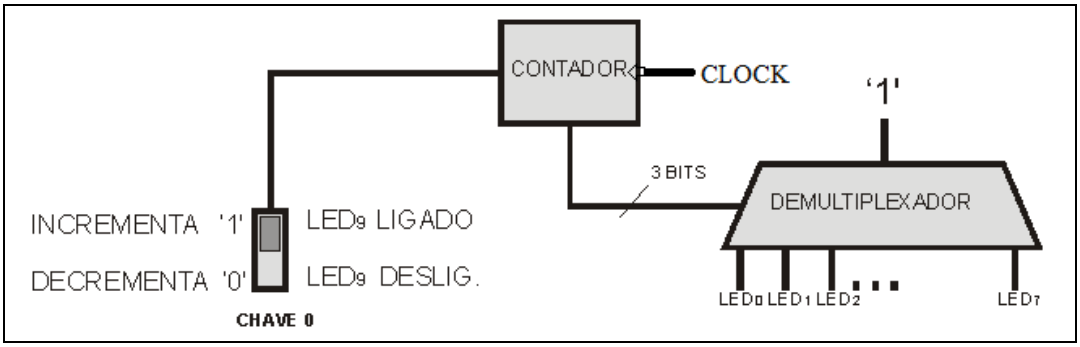

Figura 20 - Diagrama do exercício 2

4.3. EXERCÍCIO 3 (Display)

A figura 21 mostra um dos quatro módulos display de 7 segmentos que a placa DE0 possui, os segmentos destes displays são acionados com sinais de 7 bits. A proposta do exercício é escrever os caracteres 0 1 2 3 nos displays HEX3, HEX2, HEX1 e HEX0 respectivamente. A tabela 4 apresenta os valores para cada bit dos sinais dos displays para que os caracteres sejam representados corretamente. Os segmentos acendem em nível lógico '0'.

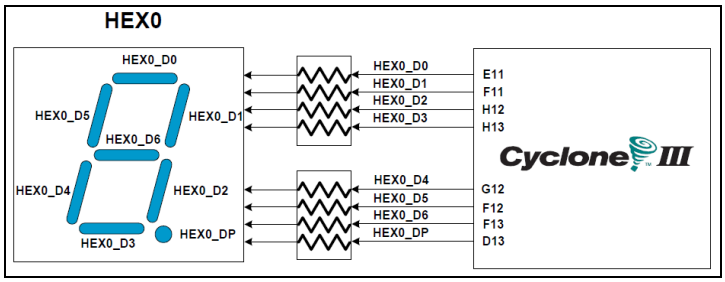

Figura 21 - Sinais do display HEX0

Tabela 4 - Valor dos sinais para representação dos caracteres

Sinais dos displays	BITS (6) (5) (4) (3) (2) (1) (0)	Caractere
HEX3_D	1 0 0 0 0 0 0	0
HEX2_D	1 1 1 1 0 0 1	1
HEX1_D	0 1 0 0 1 0 0	2
HEX0_D	0 1 1 0 0 0 0	3

Tabela 5 - Descrição em VHDL da escrita nos displays hexadecimais

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

-- Displays
ENTITY exerc3 IS
PORT (
        HEX3_D : OUT STD_LOGIC_VECTOR(6 DOWNTO 0);
        HEX2_D: OUT STD_LOGIC_VECTOR(6 DOWNTO 0);
        HEX1_D: OUT STD_LOGIC_VECTOR(6 DOWNTO 0);
        HEX0_D: OUT STD_LOGIC_VECTOR(6 DOWNTO 0));
END exerc3;

ARCHITECTURE Behavior OF exerc3 IS
BEGIN

HEX3_D <= “1000000”; -- caractere 0
HEX2_D <= “1111001”; -- caractere 1
HEX1_D <= “0100100”; -- caractere 2
HEX0_D <= “0110000”; -- caractere 3

END Behavior;
```

4.4. EXERCÍCIO 4 (Displays)

Após a realização do exercício 3 a proposta é modificar o código VHDL para que a contagem que era de 0 a 3 nos displays passe a contar até 9. Para isto os displays terão que alternar-se da direita para esquerda ao mostrar a contagem como demonstra os passos da figura 22. Deve-se utilizar o CLOCK_50 para criar um sinal de clock interno com freqüência de 1Hz, ou seja, para que os displays mudem de caracteres a cada 1 segundo. Ao terminar a contagem os displays devem voltar para o 1° passo.

DICAS: No ANEXO I estão os sinais e as características do CLOCK.

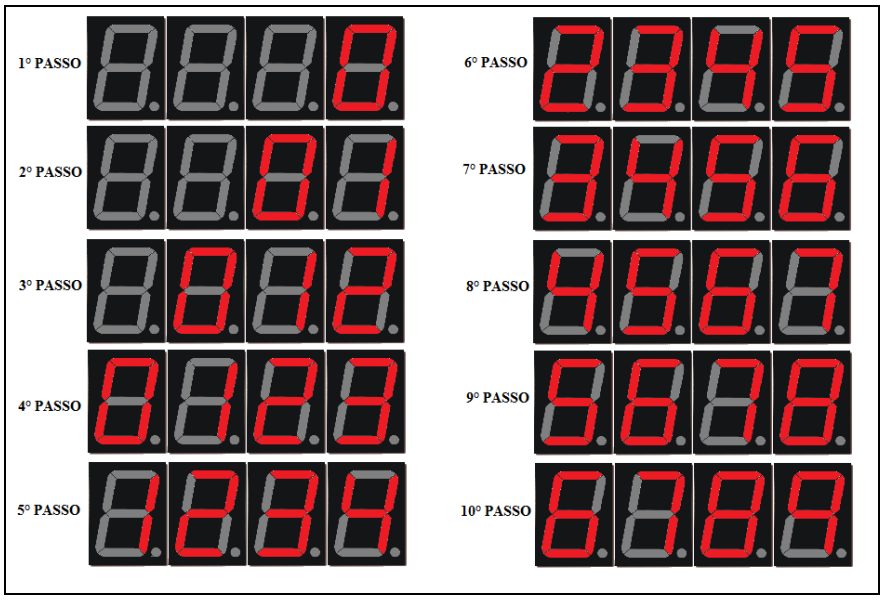

Figura 22 - Contagem seqüencial nos displays

4.5. EXERCÍCIO 5 (MEMÓRIA)

A proposta agora é ler valores da memória Flash e apresentar no display. No ANEXO I estão os sinais de acesso a memória Flash.

Deve-se utilizar as chaves SW9-5 para gerar o endereço de 5 bits da leitura na memória, considerando SW(9) o bit mais significativo do endereço, ou seja, pode-se ler do endereço "00000" ao "11111" (0 ao 31). Os valores que estão nestes endereços de memória também possuem valores no máximo de 5 bits.

Sempre que o BUTTON1 for pressionado os displays HEX1 e HEX0 devem apresentar o valor do local da memória que está endereçado pelas chaves SW9-5.

Na tabela 6 é apresentada a descrição em VHDL para a leitura da memória.

OBS: Ao copiar o código diretamente para o software Quartus II, alguns caracteres são lidos de forma errada, como as aspas (" ") das atribuições de valores binários. Exemplo: HEX1<= "0000001";

Tabela 6 - Código para a leitura de valores na memória FLASH

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

-- Ler na memória
ENTITY ler_memoria_flash IS
PORT (
	CLOCK_50: IN STD_LOGIC;--Entrada do sinal de clock 50MHz

	--Interruptores
	SW: IN STD_LOGIC_VECTOR(9 DOWNTO 0);
	BUTTON: IN STD_LOGIC_VECTOR(2 DOWNTO 0);

	--Leds
	LEDG: OUT STD_LOGIC_VECTOR(9 DOWNTO 0);

	--Memória FLASH
	FL_DQ:INOUT STD_LOGIC_VECTOR(14 DOWNTO 0);     --FLASH Data bus 15 Bits
	FL_DQ15_AM1: INOUT STD_LOGIC;                  --FLASH Data bus Bit 15 or Address A-1
	FL_ADDR: OUT STD_LOGIC_VECTOR(21 DOWNTO 0);  --FLASH Address bus 22 Bits
	FL_WE_N: OUT STD_LOGIC;                        --FLASH Write Enable
	FL_RST_N: OUT STD_LOGIC;                       --FLASH Reset
	FL_OE_N: OUT STD_LOGIC;                       --FLASH Output Enable
	FL_CE_N: OUT STD_LOGIC;                        --FLASH Chip Enable
	FL_WP_N: OUT STD_LOGIC;                        --FLASH Hardware Write Protect
	FL_BYTE_N: OUT STD_LOGIC;                      --FLASH Selects 8/16-bit mode
	FL_RY: IN STD_LOGIC;                           --FLASH Ready(1)/Busy(0)

	--Display hexadecimal
	HEX3_D: OUT STD_LOGIC_VECTOR(6 DOWNTO 0);
	HEX2_D: OUT STD_LOGIC_VECTOR(6 DOWNTO 0);
	HEX1_D: OUT STD_LOGIC_VECTOR(6 DOWNTO 0);
	HEX0_D: OUT STD_LOGIC_VECTOR(6 DOWNTO 0));

END ler_memoria_flash;

ARCHITECTURE Behavior OF ler_memoria_flash IS

--Sinais para controle
signal RESET, LER: std_logic;
signal ENDERECO: std_logic_vector(21 downto 0);
signal DADOS_LIDOS: std_logic_vector(7 downto 0);
signal DADOS_DISPLAY: std_logic_vector(4 downto 0);

-- Estados necessários para a leitura
type state_type is (s_RESET, s_WAIT_COMMAND, s_READ, s_ACKNOWLEDGE);

-- Registradores de estado
signal present_state    : state_type;
signal next_state       : state_type;
```

```
-- Sinais para o sistema de leitura da memória
signal data_from_flash              : std_logic_vector(7 downto 0);
signal cycle_counter  : std_logic_vector(4 downto 0);
signal flash_ce_n, flash_oe_n, flash_we_n, flash_reset_n: std_logic;
signal flash_address   : STD_LOGIC_VECTOR(21 downto 0);
signal i_clock : STD_LOGIC;
signal i_reset_n : STD_LOGIC;
signal i_address : STD_LOGIC_VECTOR(21 downto 0);
signal b_flash_data: STD_LOGIC_VECTOR(7 downto 0);
signal i_read : STD_LOGIC;
signal o_done : STD_LOGIC;

BEGIN

process(CLOCK_50) -- É utilizado o clock de 50MHz da placa DE0 para acesso a memória Flash
            begin
            if rising_edge(CLOCK_50) then
-------------------------------------------------------------------------------------------
--LER DADOS DA MEMÓRIA FLASH-----------------------------------------------

                        if  BUTTON(1) = '0' then --Teste do botão 1
                              if o_done = '0' then --Teste para saber se a memória está livre para acesso
                                    RESET <= '1';
                                    FL_BYTE_N <= '1';--Modo 16 bits
                                    LER <= '1'; --Ativa a leitura
                                    ENDERECO <= "0000000000000000"&SW(9 downto 5);--endereço pelas chaves SW9 a SW5
                                   DADOS_DISPLAY <= DADOS_LIDOS(4 DOWNTO 0); --Dados lidos direto no display
                               end if;
                        else LER <= '0';
                        end if;

-----DECODIFICADOR PARA OS DISPLAYS HEXADECIMAL PARA REPRESETAREM VALORES DE 0 A 31------

            HEX3_D <= "1111111";--limpa display HEX3
            HEX2_D <= "1111111";--limpa display HEX2

            CASE DADOS_DISPLAY IS
             WHEN "00000" => HEX1_D <= "1000000"; HEX0_D <= "1000000";--0
             WHEN "00001" => HEX1_D <= "1000000"; HEX0_D <= "1111001";--1
             WHEN "00010" => HEX1_D <= "1000000"; HEX0_D <= "0100100";--2
             WHEN "00011" => HEX1_D <= "1000000"; HEX0_D <= "0110000";--3
             WHEN "00100" => HEX1_D <= "1000000"; HEX0_D <= "0011001";--4
             WHEN "00101" => HEX1_D <= "1000000"; HEX0_D <= "0010010";--5
             WHEN "00110" => HEX1_D <= "1000000"; HEX0_D <= "0000010";--6
             WHEN "00111" => HEX1_D <= "1000000"; HEX0_D <= "1111000";--7
             WHEN "01000" => HEX1_D <= "1000000"; HEX0_D <= "0000000";--8
             WHEN "01001" => HEX1_D <= "1000000"; HEX0_D <= "0010000";--9
             WHEN "01010" => HEX1_D <= "1111001"; HEX0_D <= "1000000";--10
             WHEN "01011" => HEX1_D <= "1111001"; HEX0_D <= "1111001";--11
             WHEN "01100" => HEX1_D <= "1111001"; HEX0_D <= "0100100";--12
             WHEN "01101" => HEX1_D <= "1111001"; HEX0_D <= "0110000";--13
             WHEN "01110" => HEX1_D <= "1111001"; HEX0_D <= "0011001";--14
             WHEN "01111" => HEX1_D <= "1111001"; HEX0_D <= "0010010";--15
             WHEN "10000" => HEX1_D <= "1111001"; HEX0_D <= "0000010";--16
             WHEN "10001" => HEX1_D <= "1111001"; HEX0_D <= "1111000";--17
             WHEN "10010" => HEX1_D <= "1111001"; HEX0_D <= "0000000";--18
             WHEN "10011" => HEX1_D <= "1111001"; HEX0_D <= "0010000";--19
             WHEN "10100" => HEX1_D <= "0100100"; HEX0_D <= "1000000";--20
             WHEN "10101" => HEX1_D <= "0100100"; HEX0_D <= "1111001";--21
             WHEN "10110" => HEX1_D <= "0100100"; HEX0_D <= "0100100";--22
             WHEN "10111" => HEX1_D <= "0100100"; HEX0_D <= "0110000";--23
             WHEN "11000" => HEX1_D <= "0100100"; HEX0_D <= "0011001";--24
             WHEN "11001" => HEX1_D <= "0100100"; HEX0_D <= "0010010";--25
             WHEN "11010" => HEX1_D <= "0100100"; HEX0_D <= "0000010";--26
             WHEN "11011" => HEX1_D <= "0100100"; HEX0_D <= "1111000";--27
             WHEN "11100" => HEX1_D <= "0100100"; HEX0_D <= "0000000";--28
             WHEN "11101" => HEX1_D <= "0100100"; HEX0_D <= "0010000";--29
             WHEN "11110" => HEX1_D <= "0110000"; HEX0_D <= "1000000";--30
             WHEN "11111" => HEX1_D <= "0110000"; HEX0_D <= "1111001";--31
             WHEN OTHERS  => HEX1_D <= "1111111"; HEX0_D <= "1111111";
            END CASE;


                        end if;
end process;
```

```
--------------------------------------------------------------------------------
----PROCESSO DE LEITURA DA MEMÓRIA FLASH-----------------------------------------
--------------------------------------------------------------------------------
--A leitura da memória FLASH deve seguir os tempos determinados na pág. 49 do
--datasheet da fabricante.
--LINK do datasheet: http://www.spansion.com/Support/Datasheets/S29AL032D.pdf
--------------------------------------------------------------------------------

	PROCESS(i_read, cycle_counter, i_address, present_state)
	BEGIN
		case present_state is
			when s_RESET =>
				if (cycle_counter = "11011") then
					next_state <= s_WAIT_COMMAND;
				else
					next_state <= s_RESET;
				end if;

			when s_WAIT_COMMAND =>
				if (i_read = '1') then
					next_state <= s_READ;
				else
					next_state <= s_WAIT_COMMAND;
				end if;

			when s_READ =>
				if (cycle_counter = "100") then
					next_state <= s_ACKNOWLEDGE;
				else
					next_state <= s_READ;
				end if;

			when s_ACKNOWLEDGE =>
				if (i_read = '1') then
					next_state <= s_ACKNOWLEDGE;
				else
					next_state <= s_WAIT_COMMAND;
				end if;

			when others =>
				next_state <= s_RESET;

		end case;

	END PROCESS;

	-- DEFINIÇÃO DOS ESTADOS----------
	PROCESS(i_clock, i_reset_n)
	BEGIN
		if (i_reset_n = '0') then
			present_state <= s_RESET;
		elsif (rising_edge(i_clock)) then
			present_state <= next_state;
		end if;
	END PROCESS;

	-- DEFINIÇÃO DOS SINAIS DE CONTROLE---
	PROCESS(i_clock, i_reset_n)
	BEGIN
		if (i_reset_n = '0') then
			cycle_counter <= (OTHERS => '0');
			flash_ce_n <= '1';
			flash_oe_n <= '1';
			flash_we_n <= '1';
			flash_reset_n <= '0';
			flash_address <= (OTHERS => '1');
		elsif (rising_edge(i_clock)) then

			-- Reset ---
			if ((present_state = s_RESET) and (cycle_counter < "11001")) then
				flash_reset_n <= '0';
			else
				flash_reset_n <= '1';
			end if;
```

```
                                -- Habilitação da FLASH
                                if ((present_state = s_RESET) or (present_state = s_WAIT_COMMAND) or (present_state =
s_ACKNOWLEDGE)) then
                                        flash_ce_n <= '1';
                                else
                                        flash_ce_n <= '0';
                                end if;

                                -- Contador para o delay dos sinais
                                if (present_state = next_state) then
                                        cycle_counter <= cycle_counter + '1';
                                else
                                        cycle_counter <= (OTHERS => '0');
                                end if;

                                -- Habilitação da saída após o ciclo da leitura.
                                if ((present_state = s_READ) and (cycle_counter > "00000")) then
                                        flash_oe_n <= '0';
                                else
                                        flash_oe_n <= '1';
                                end if;

                                -- Dados no sinal de saída
                                if (present_state = s_READ) then
                                        data_from_flash <= b_flash_data;
                                end if;

                                -- Endereços para a leitura
                                flash_address <= i_address;
                        end if;
        END PROCESS;
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
        -- Sinais de saída para a memória FLASH
        o_done <= '1' when (present_state = s_ACKNOWLEDGE) else '0';
        DADOS_LIDOS <= data_from_flash;
        FL_CE_N <= flash_ce_n;
        FL_OE_N <= flash_oe_n;
        FL_WE_N <= flash_we_n;
        FL_RST_N <= flash_reset_n;
        FL_ADDR <= flash_address;
        b_flash_data <= FL_DQ(7 downto 0);

   --Sinais atribuídos para controle
        i_address<= ENDERECO;
        i_read   <= LER;
        i_reset_n<= RESET;
        i_clock  <= CLOCK_50;

end Behavior;
```

4.6. EXERCÍCIO 6

Após realizar o exercício 5, podemos considerar que os valores da memória correspondem a temperaturas em diferentes ambientes. A proposta agora é determinar qual a maior temperatura e em que ambiente ocorre.

Devem-se considerar temperaturas na faixa de 0 a 31, já armazenadas na memória flash, e 32 ambientes (ambiente 0 ao 31).

Após o BUTTON1 ser acionado, o sistema deve varrer as 32 posições (ambientes) e buscar a maior temperatura entre eles. O ambiente/posição que está o maior valor deve ser apresentado nos displays HEX3 e HEX2, e o valor da temperatura mais alta nos displays HEX1 E HEX0.

Na tabela 7 pode-se analisar o algoritmo para realizar o processo, e na figura 23 é apresentado um diagrama da estrutura proposta no exercício.

Tabela 7 - Algoritmo do processo

```
PRIMEIRO = ENTRADA[0];
FOR(LOOP=0; LOOP<32; LOOP++)
{
 IF ENTRADA [LOOP] > PRIMEIRO
   {
     PRIMEIRO = ENTRADA[LOOP];
     POSIÇÃO = LOOP;
    }
 }
```

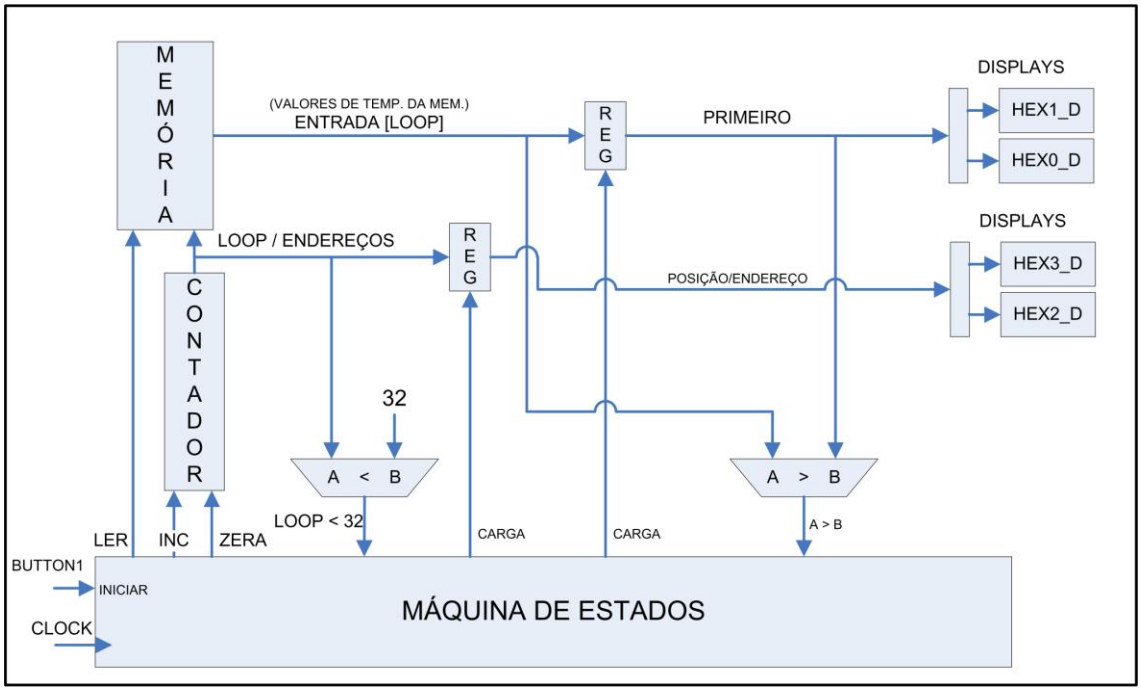

Figura 23 - Diagrama da estrutura para leitura da maior temperatura entre os ambientes

5. ANEXO I

Neste anexo são apresentadas as características dos periféricos da placa DE0 e os pinos do FPGA atribuídos aos sinais pelo arquivo “DE0_pin_assignments.qsf”.

5.1. INTERRUPTORES

A placa DE0 possui 13 interruptores, sendo 10 chaves e 3 botões.

As 10 chaves quando estão na posição *UP* apresentam nível lógico “1” e na posição *DOWN* nível lógico “0” como mostra a figura 24.

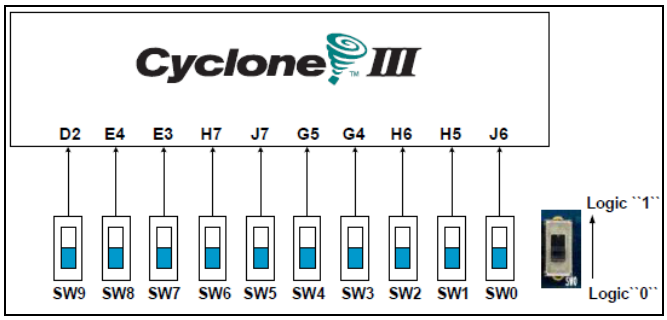

Figura 24 - As chaves presentes na placa DE0

Já os 3 botões, quando pressionados, apresentam nível lógico “0” como mostra a figura 25.

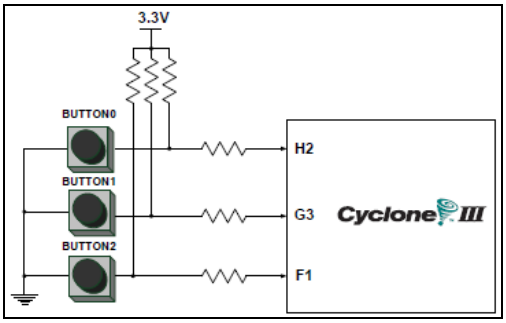

Figura 25 - Botões presentes na placa DE0

Nas tabelas 8 e 9 são apresentados os nomes dos sinais e os pinos do FPGA das chaves e botões, respectivamente.

Tabela 8 - Sinais das chaves e as atribuições aos 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]

Tabela 9 - Sinais dos botões e as atribuições aos 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]

5.2. LEDs

A placa DE0 possui 10 LEDs acionados por nível lógico "1" como mostra a figura 26.

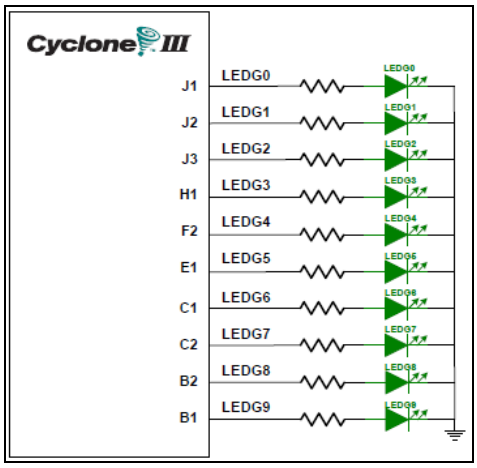

Figura 26 - LEDs presentes na placa DE0

Na tabela 10 são apresentados os sinais dos LEDs e os pinos do FPGA na qual estão endereçados pelo arquivo "DE0_pin_assignments.qsf".

Tabela 10 - Sinais dos LEDs e as atribuições aos 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]

5.3. DISPLAYS

A placa DE0 possui quatro displays com sete segmentos, estes segmentos são ativos em nível lógico "0" e podem representar caracteres hexadecimais de 0 a F.

Na figura 27 observam-se os sinais de controle dos segmentos do display HEX0.

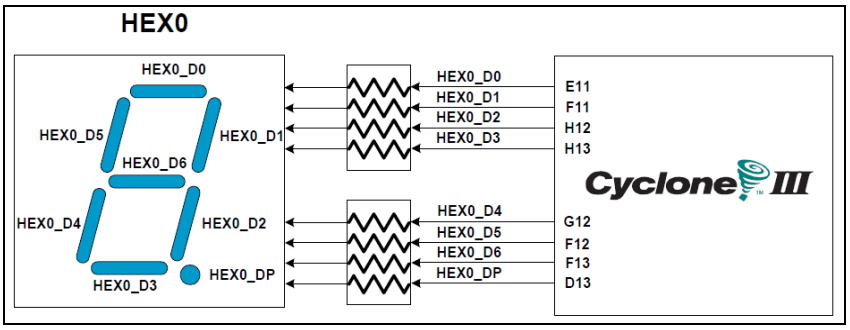

Figura 27 - Display hexadecimal

A tabela 11 apresenta todos os sinais de controle dos displays da placa DE0.

Tabela 11 - Sinais de controle dos segmentos dos displays

Signal Name	FPGA Pin No.	Description
HEX0_D[0]	PIN_E11	Seven Segment Digit 0[0]
HEX0_D[1]	PIN_F11	Seven Segment Digit 0[1]
HEX0_D[2]	PIN_H12	Seven Segment Digit 0[2]
HEX0_D[3]	PIN_H13	Seven Segment Digit 0[3]
HEX0_D[4]	PIN_G12	Seven Segment Digit 0[4]
HEX0_D[5]	PIN_F12	Seven Segment Digit 0[5]
HEX0_D[6]	PIN_F13	Seven Segment Digit 0[6]
HEX0_DP	PIN_D13	Seven Segment Decimal Point 0
HEX1_D[0]	PIN_A13	Seven Segment Digit 1[0]
HEX1_D[1]	PIN_B13	Seven Segment Digit 1[1]
HEX1_D[2]	PIN_C13	Seven Segment Digit 1[2]
HEX1_D[3]	PIN_A14	Seven Segment Digit 1[3]
HEX1_D[4]	PIN_B14	Seven Segment Digit 1[4]
HEX1_D[5]	PIN_E14	Seven Segment Digit 1[5]
HEX1_D[6]	PIN_A15	Seven Segment Digit 1[6]
HEX1_DP	PIN_B15	Seven Segment Decimal Point 1

HEX2_D[0]	PIN_D15	Seven Segment Digit 2[0]
HEX2_D[1]	PIN_A16	Seven Segment Digit 2[1]
HEX2_D[2]	PIN_B16	Seven Segment Digit 2[2]
HEX2_D[3]	PIN_E15	Seven Segment Digit 2[3]
HEX2_D[4]	PIN_A17	Seven Segment Digit 2[4]
HEX2_D[5]	PIN_B17	Seven Segment Digit 2[5]
HEX2_D[6]	PIN_F14	Seven Segment Digit 2[6]
HEX2_DP	PIN_A18	Seven Segment Decimal Point 2
HEX3_D[0]	PIN_B18	Seven Segment Digit 3[0]
HEX3_D[1]	PIN_F15	Seven Segment Digit 3[1]
HEX3_D[2]	PIN_A19	Seven Segment Digit 3[2]
HEX3_D[3]	PIN_B19	Seven Segment Digit 3[3]
HEX3_D[4]	PIN_C19	Seven Segment Digit 3[4]
HEX3_D[5]	PIN_D19	Seven Segment Digit 3[5]
HEX3_D[6]	PIN_G15	Seven Segment Digit 3[6]
HEX3_DP	PIN_G16	Seven Segment Decimal Point 3

5.4. PORTAS DE EXPANSÃO

A placa DE0 possui duas conexões de 40 pinos para cabos de fita utilizados em discos rígidos IDE. Cada conexão possui 36 pinos I/O (entrada e saída) diretas com o FPGA. Estas conexões também apresentam pinos de alimentação 5V, 3.3V e GND como observa-se na figura 28.

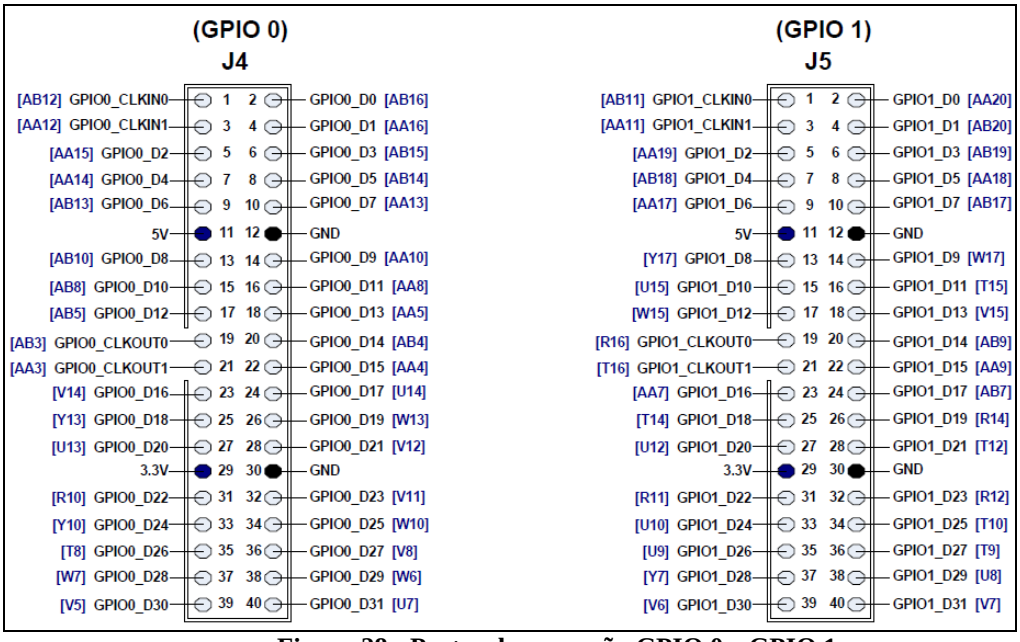

Figura 28 - Portas de expansão GPIO 0 e GPIO 1

A tabela 12 apresenta os sinais das duas portas de expansão e os pinos do FPGA que estão conectadas.

Uma boa aplicação para os sinais dos pinos de expansão é, por exemplo, para verificação de clock e sinais internos dos circuitos prototipados no FPGA, pois ao

direcionar um sinal do projeto que está sendo prototipado para um pino de expansão, pode-se utilizar um osciloscópio para aferição do mesmo.

Tabela 12 - Sinais dos pinos de expansão

Signal Name	FPGA Pin No.	Description
GPIO0_D[0]	PIN_AB16	GPIO Connection 0 IO[0]
GPIO0_D[1]	PIN_AA16	GPIO Connection 0 IO[1]
GPIO0_D[2]	PIN_AA15	GPIO Connection 0 IO[2]
GPIO0_D[3]	PIN_AB15	GPIO Connection 0 IO[3]
GPIO0_D[4]	PIN_AA14	GPIO Connection 0 IO[4]
GPIO0_D[5]	PIN_AB14	GPIO Connection 0 IO[5]
GPIO0_D[6]	PIN_AB13	GPIO Connection 0 IO[6]
GPIO0_D[7]	PIN_AA13	GPIO Connection 0 IO[7]
GPIO0_D[8]	PIN_AB10	GPIO Connection 0 IO[8]
GPIO0_D[9]	PIN_AA10	GPIO Connection 0 IO[9]
GPIO0_D[10]	PIN_AB8	GPIO Connection 0 IO[10]
GPIO0_D[11]	PIN_AA8	GPIO Connection 0 IO[11]
GPIO0_D[12]	PIN_AB5	GPIO Connection 0 IO[12]
GPIO0_D[13]	PIN_AA5	GPIO Connection 0 IO[13]
GPIO0_D[14]	PIN_AB4	GPIO Connection 0 IO[14]
GPIO0_D[15]	PIN_AA4	GPIO Connection 0 IO[15]
GPIO0_D[16]	PIN_V14	GPIO Connection 0 IO[16]
GPIO0_D[17]	PIN_U14	GPIO Connection 0 IO[17]
GPIO0_D[18]	PIN_Y13	GPIO Connection 0 IO[18]
GPIO0_D[19]	PIN_W13	GPIO Connection 0 IO[19]
GPIO0_D[20]	PIN_U13	GPIO Connection 0 IO[20]
GPIO0_D[21]	PIN_V12	GPIO Connection 0 IO[21]
GPIO0_D[22]	PIN_R10	GPIO Connection 0 IO[22]
GPIO0_D[23]	PIN_V11	GPIO Connection 0 IO[23]
GPIO0_D[24]	PIN_Y10	GPIO Connection 0 IO[24]
GPIO0_D[25]	PIN_W10	GPIO Connection 0 IO[25]
GPIO0_D[26]	PIN_T8	GPIO Connection 0 IO[26]
GPIO0_D[27]	PIN_V8	GPIO Connection 0 IO[27]
GPIO0_D[28]	PIN_W7	GPIO Connection 0 IO[28]
GPIO0_D[29]	PIN_W6	GPIO Connection 0 IO[29]
GPIO0_D[30]	PIN_V5	GPIO Connection 0 IO[30]
GPIO0_D[31]	PIN_U7	GPIO Connection 0 IO[31]
GPIO0_CLKIN[0]	PIN_AB12	GPIO Connection 0 PLL In
GPIO0_CLKIN[1]	PIN_AA12	GPIO Connection 0 PLL In
GPIO0_CLKOUT[0]	PIN_AB3	GPIO Connection 0 PLL Out
GPIO0_CLKOUT[1]	PIN_AA3	GPIO Connection 0 PLL Out
GPIO1_D[0]	PIN_AA20	GPIO Connection 1 IO[0]
GPIO1_D[1]	PIN_AB20	GPIO Connection 1 IO[1]
GPIO1_D[2]	PIN_AA19	GPIO Connection 1 IO[2]
GPIO1_D[3]	PIN_AB19	GPIO Connection 1 IO[3]
GPIO1_D[4]	PIN_AB18	GPIO Connection 1 IO[4]
GPIO1_D[5]	PIN_AA18	GPIO Connection 1 IO[5]
GPIO1_D[6]	PIN_AA17	GPIO Connection 1 IO[6]
GPIO1_D[7]	PIN_AB17	GPIO Connection 1 IO[7]

GPIO1_D[8]	PIN_Y17	GPIO Connection 1 IO[8]
GPIO1_D[9]	PIN_W17	GPIO Connection 1 IO[9]
GPIO1_D[10]	PIN_U15	GPIO Connection 1 IO[10]
GPIO1_D[11]	PIN_T15	GPIO Connection 1 IO[11]
GPIO1_D[12]	PIN_W15	GPIO Connection 1 IO[12]
GPIO1_D[13]	PIN_V15	GPIO Connection 1 IO[13]
GPIO1_D[14]	PIN_AB9	GPIO Connection 1 IO[14]
GPIO1_D[15]	PIN_AA9	GPIO Connection 1 IO[15]
GPIO1_D[16]	PIN_AA7	GPIO Connection 1 IO[16]
GPIO1_D[17]	PIN_AB7	GPIO Connection 1 IO[17]
GPIO1_D[18]	PIN_T14	GPIO Connection 1 IO[18]
GPIO1_D[19]	PIN_R14	GPIO Connection 1 IO[19]
GPIO1_D[20]	PIN_U12	GPIO Connection 1 IO[20]
GPIO1_D[21]	PIN_T12	GPIO Connection 1 IO[21]
GPIO1_D[22]	PIN_R11	GPIO Connection 1 IO[22]
GPIO1_D[23]	PIN_R12	GPIO Connection 1 IO[23]
GPIO1_D[24]	PIN_U10	GPIO Connection 1 IO[24]
GPIO1_D[25]	PIN_T10	GPIO Connection 1 IO[25]
GPIO1_D[26]	PIN_U9	GPIO Connection 1 IO[26]
GPIO1_D[27]	PIN_T9	GPIO Connection 1 IO[27]
GPIO1_D[28]	PIN_Y7	GPIO Connection 1 IO[28]
GPIO1_D[29]	PIN_U8	GPIO Connection 1 IO[29]
GPIO1_D[30]	PIN_V6	GPIO Connection 1 IO[30]
GPIO1_D[31]	PIN_V7	GPIO Connection 1 IO[31]
GPIO1_CLKIN[0]	PIN_AB11	GPIO Connection 1 PLL In
GPIO1_CLKIN[1]	PIN_AA11	GPIO Connection 1 PLL In
GPIO1_CLKOUT[0]	PIN_R16	GPIO Connection 1 PLL Out
GPIO1_CLKOUT[1]	PIN_T16	GPIO Connection 1 PLL Out

5.5. CLOCK

A placa DE0 possui um gerador de CLOCK de 50 MHz. Este CLOCK pode ser utilizado através do sinal CLOCK_50 como mostra a tabela 13.

Para conseguir sinais de CLOCK com freqüências menores pode-se utilizar PLL´s da biblioteca MEGAFUNCTIONS do software QUARTUS II, ou apenas implementar um exemplo de código divisor de CLOCK em VHDL apresentado na tabela 14.

Tabela 13 - Sinais do CLOCK 50 MHz

Signal Name	FPGA Pin No.	Description
CLOCK_50	PIN_G21	50 MHz clock input

Tabela 14 - Exemplo de um divisor de clock em VHDL

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

ENTITY exemplo IS
PORT (		CLOCK_50: IN STD_LOGIC);
END exemplo;

ARCHITECTURE Behavior OF exemplo IS
--Declaração dos sinais utilizados pelo divisor de clock
SIGNAL bitClockCounter: INTEGER RANGE 0 TO 2;
SIGNAL CLK_INTERNO: STD_LOGIC; -- Sinal de clock resultante da divisão do clock principal

BEGIN
--exemplo para se obter um clock interno de 25MHz, ou seja, 50MHz / 2
process(CLOCK_50)
	begin
		if CLOCK_50'event and CLOCK_50 = '1' then
			if bitClockCounter < 1 then
				CLK_INTERNO <= '0';
				bitClockCounter <= bitClockCounter + 1;
			elsif bitClockCounter >= 1 and bitClockCounter < 2 then
				CLK_INTERNO <= '1';

				bitClockCounter <= bitClockCounter + 1;
			else
				CLK_INTERNO <= '0';
				bitClockCounter <= 0;
			end if;
		end if;
end process;
END Behavior;
```

5.6. MODOS DE PROGRAMAÇÃO DO FPGA

Existem duas maneiras de programar o FPGA da placa DE0, o modo RUN e PROG. No modo RUN o código compilado pelo programa Quartus II é prototipado em modo temporário, ou seja, se a placa DE0 for desligada e ligada novamente a prototipação vai ser apagada. Já no modo PROG o código VHDL compilado é programado de maneira definitiva na placa até que outro código seja programado da mesma forma.

A seleção dos modos é realizada pela chave presente na placa DE0 (Figura 29).

Para programar o FPGA no modo RUN é necessário seguir os passos descritos no subcapitulo 3.4, já no modo PROG é preciso carregar o arquivo .pof na ferramenta "PROGRAMMER", do software Quartus II da Altera, selecionar o modo "Active Serial Programmer" e clicar em "Start" para programar o FPGA.

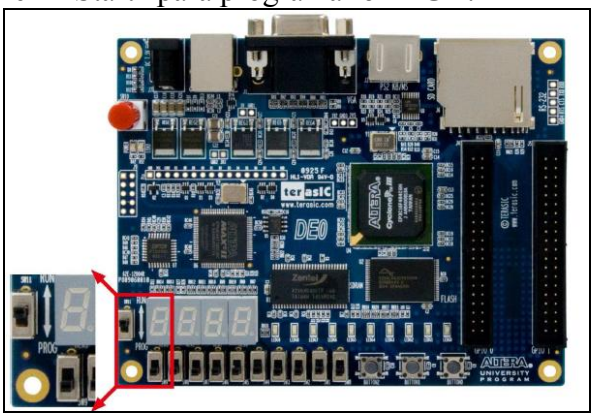

Figura 29 - Chave de seleção de modo de programação

5.7. MEMÓRIA FLASH

A placa DE0 possui um chip de memória Flash de 4 Mbyte conectada ao FPGA como mostra a figura 30. Os pinos de atribuição são listados na tabela 15.

O datasheet da memória Flash pode ser acessado pelo link do site do fabricante Spansion: http://www.spansion.com/Support/Datasheets/S29AL032D.pdf

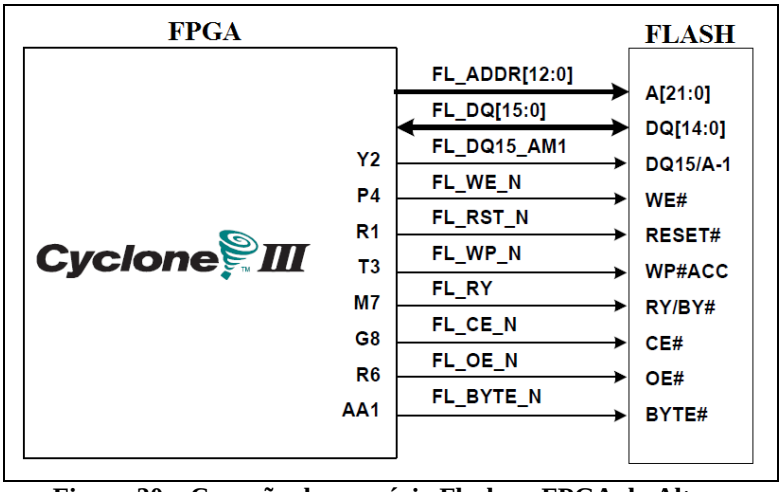

Figura 30 – Conexão da memória Flash ao FPGA da Altera

Tabela 15 – Sinais de acesso a memória Flash

Signal Name	FPGA Pin No.	Description
FL_ADDR[0]	PIN_P7	FLASH Address[0]
FL_ADDR[1]	PIN_P5	FLASH Address[1]
FL_ADDR[2]	PIN_P6	FLASH Address[2]
FL_ADDR[3]	PIN_N7	FLASH Address[3]
FL_ADDR[4]	PIN_N5	FLASH Address[4]
FL_ADDR[5]	PIN_N6	FLASH Address[5]
FL_ADDR[6]	PIN_M8	FLASH Address[6]
FL_ADDR[7]	PIN_M4	FLASH Address[7]
FL_ADDR[8]	PIN_P2	FLASH Address[8]
FL_ADDR[9]	PIN_N2	FLASH Address[9]
FL_ADDR[10]	PIN_N1	FLASH Address[10]
FL_ADDR[11]	PIN_M3	FLASH Address[11]
FL_ADDR[12]	PIN_M2	FLASH Address[12]
FL_ADDR[13]	PIN_M1	FLASH Address[13]
FL_ADDR[14]	PIN_L7	FLASH Address[14]
FL_ADDR[15]	PIN_L6	FLASH Address[15]
FL_ADDR[16]	PIN_AA2	FLASH Address[16]
FL_ADDR[17]	PIN_M5	FLASH Address[17]
FL_ADDR[18]	PIN_M6	FLASH Address[18]
FL_ADDR[19]	PIN_P1	FLASH Address[19]
FL_ADDR[20]	PIN_P3	FLASH Address[20]
FL_ADDR[21]	PIN_R2	FLASH Address[21]
FL_DQ[0]	PIN_R7	FLASH Data[0]
FL_DQ[1]	PIN_P8	FLASH Data[1]
FL_DQ[2]	PIN_R8	FLASH Data[2]
FL_DQ[3]	PIN_U1	FLASH Data[3]
FL_DQ[4]	PIN_V2	FLASH Data[4]
FL_DQ[5]	PIN_V3	FLASH Data[5]
FL_DQ[6]	PIN_W1	FLASH Data[6]

FL_DQ[7]	PIN_Y1	FLASH Data[7]
FL_DQ[8]	PIN_T5	FLASH Data[8]
FL_DQ[9]	PIN_T7	FLASH Data[9]
FL_DQ[10]	PIN_T4	FLASH Data[10]
FL_DQ[11]	PIN_U2	FLASH Data[11]
FL_DQ[12]	PIN_V1	FLASH Data[12]
FL_DQ[13]	PIN_V4	FLASH Data[13]
FL_DQ[14]	PIN_W2	FLASH Data[14]
FL_DQ15_AM1	PIN_Y2	FLASH Data[15]
FL_BYTE_N	PIN_AA1	FLASH Byte/Word Mode Configuration
FL_CE_N	PIN_N8	FLASH Chip Enable
FL_OE_N	PIN_R6	FLASH Output Enable
FL_RST_N	PIN_R1	FLASH Reset
FL_RY	PIN_M7	LASH Ready/Busy output
FL_WE_N	PIN_P4	FLASH Write Enable
FL_WP_N	PIN_T3	FLASH Write Protect /Programming Acceleration