

## Experimento 4:

### INTRODUÇÃO À PROGRAMAÇÃO DE FPGAs UTILIZANDO A LINGUAGEM VHDL

#### 1 OBJETIVO

É objetivo deste experimento é implementar circuitos combinacionais simples baseados em FPGA, utilizando a linguagem de descrição de hardware VHDL. Todas as etapas do processo são abordadas, desde a elaboração do código VHDL, passando pelo processo de síntese e até a programação propriamente dita, quando o resultado da compilação é transferido para o FPGA.

São estudados ainda o conversor de código binário para código de 7 segmentos e um comparador de palavras binárias.

#### 2 MATERIAL UTILIZADO

- Placa de prototipagem Basys2 da Diligent, Inc., com o FPGA Spartan 3E-100 CP132 (XC3S100E-CP132), da Xilinx.
- Um microcomputador PC.
- Software de desenvolvimento ISE WebPACK, da Xilinx, para síntese do código VHDL.
- Software Adept, da Diligent, para programação do FPGA.

#### 3 INTRODUÇÃO

##### 3.1 A placa de prototipagem Basys2

Neste experimento, será abordada a programação do FPGA Spartan 3E-100 da Xilinx, utilizando a linguagem de descrição de hardware VHDL. A placa Basys2 da Diligent, ilustrada na figura 1, contém um circuito FPGA dessa família, bem como um microcontrolador Atmel AT90USB2 e diversos dispositivos de entrada e saída, como, por exemplo: chaves, botões de pressão, LEDs, displays de 7 segmentos, interfaces PS/2 e VGA, e conectores Pmod. Assim, essa placa é bastante conveniente para a elaboração de protótipos de sistemas digitais baseados em FPGA.

Durante este experimento, será utilizada a placa Basys2, pela conveniência de programação do XC3S100E e de avaliação do sistema final. Alguns pinos do FPGA são conectados a chaves, botões, LEDs e mostradores de 7 segmentos, permitindo a alteração dos bits de entradas e observação dos bits de saída sem a necessidade de um circuito externo à placa.

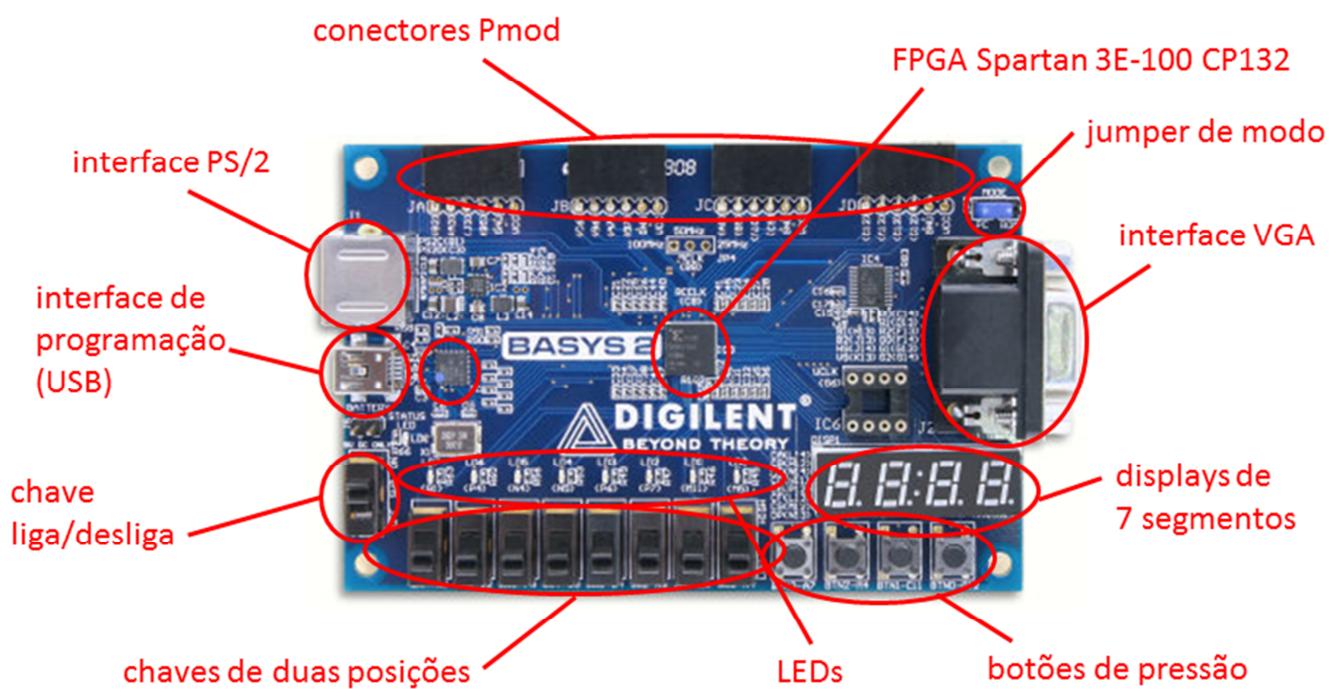


Figura 1 - Placa de prototipagem Basys2, da Digilent, Inc.

Na próxima seção, será mostrado um exemplo de programação do XC3S100E utilizando a placa Basys2. O sistema digital abordado é um circuito combinacional bastante simples, já que o objetivo é apenas ilustrar todas as etapas necessárias.

### 3.2 Exemplo de implementação de uma função combinacional utilizando o FPGA XC3S100E

Nesta seção, será implementado um detector de paridade com 4 bits de entrada com uso do FPGA XC3S100E. O circuito correspondente é apresentado na figura 2. Observe que a saída é 1 se e somente se o número de 1s na entrada é ímpar.

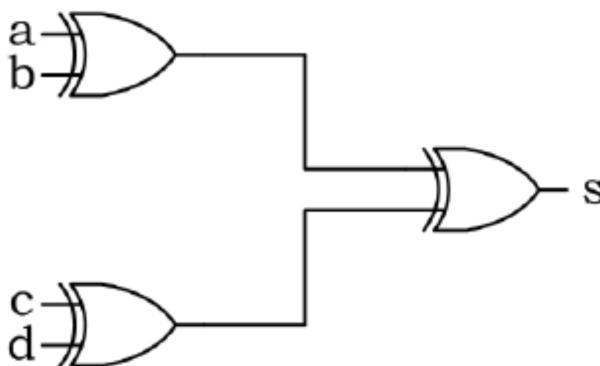


Figura 2 - Detector de paridade com 4 bits de entrada.

Neste desenvolvimento, serão utilizados dois programas: o ISE WebPACK, da Xilinx, e o Adept, da Diligent. Os dois programas podem ser obtidos gratuitamente nas páginas dos fabricantes. Para download do ISE WebPACK (ISE Design Suite), será necessário realizar um cadastro na página da Xilinx. Também será necessário baixar o arquivo de licença, o que é feito gratuitamente.

É importante que todas as etapas do processo sejam reproduzidas, e que todos os resultados intermediários sejam analisados; admite-se que os dois programas encontram-se já instalados e configurados no computador de desenvolvimento.

#### 3.2.1 Elaboração da descrição em VHDL

Inicialmente, será feita a descrição em VHDL do circuito da figura 2.

Para tanto, execute o programa ISE WebPACK (ícone Xilinx ISE Design Suite 13.1, no desktop). Se houverem janelas ou projeto abertos, vá em “Window/Close All” e em “File/Close Project”.

A seguir, clique no botão “New Project”. Crie seu projeto na pasta “C:\alunos”, em uma subpasta com seu número de matrícula (ex: C:\alunos\0123456). Entre “detecpar” como nome do

projeto. No menu “Top-level source type”, escolha HDL e clique em “Next”.

Na janela “Project Settings” use as opções mostradas na figura 3, e clique em “Next”. Verifique as seleções e clique em “Finish”.

Evaluation Development Board	None Specified
Product Category	All
Family	Spartan3E
Device	XC3S100E
Package	CP132
Speed	-4
Top-Level Source Type	HDL
Synthesis Tool	XST (VHDL/Verilog)
Simulator	ISim (VHDL/Verilog)
Preferred Lanquaqe	Verilog
Property Specification in Project File	Store all values
Manual Compile Order	<input type="checkbox"/>
VHDL Source Analysis Standard	VHDL-93
Enable Message Filtering	<input type="checkbox"/>

Figura 3 - Configurações do projeto.

Clique em “Project\New source” para criar um novo arquivo VHDL (ou em “Process\Add source” para adicionar um arquivo VHDL já existente). Na janela “Select Source Type”, escolha “VHDL Module”. O nome do arquivo pode ser detecpar.vhdl, e deve ser salvo na pasta do projeto. Certifique-se de que a opção “Add to project” está marcada e clique em “Next”.

Na janela “Define Module”, você deve dar uma nome para entidade (ex: “detector\_paridade\_4bits”) e para a arquitetura (ex: “detector\_paridade\_4bits\_op”). Crie 4 pinos (“ports”) de entrada (“in”), chamados *a*, *b*, *c* e *d*, e 1 pino de saída (“out”), chamado *s*. Clique em “Next”. Obs.: As opções “Bus”, “MSB” e “LSB” são usados quando se trabalha com “ports” do tipo BIT\_VECTOR ou STD\_LOGIC\_VECTOR, o que veremos na seção 3.3.1.

Verifique suas seleções e clique em “Finish”. Com isso, um template do código VHDL será exibido na tela. Você precisa agora escrever a descrição do circuito a ser implementado (figura 2), conforme apresentado a seguir.



```
library ieee;
use ieee.std_logic_1164.all;
entity detector_paridade_4bits is
    port(a,b,c,d: in STD_LOGIC;
         s: out STD_LOGIC);
end detector_paridade_4bits;

architecture detector_paridade_4bits_op of detector_paridade_4bits is
signal e,f: STD_LOGIC;
begin
    e <= a xor b;
    f <= c xor d;
    s <= e xor f;
end detector_paridade_4bits_op;
```

Observe que a novidade desta implementação em relação às estruturas estudadas no experimento anterior é a utilização do módulo STD\_LOGIC da biblioteca IEEE, declarado nas duas primeiras linhas (a palavra-chave “all”, na segunda linha, faz com que sejam carregadas todas as estruturas presentes no módulo). O objetivo é que se possa utilizar o tipo de sinal/variável STD LOGIC, que se diferencia do tipo BIT antes adotado por apresentar outros valores além de ‘0’ ou ‘1’, como o valor indeterminado e o irrelevante.

Digitada a descrição, salve o arquivo. Utilize então a verificação automática de sintaxe. Na árvore hierárquica do projeto (janela da esquerda), clique no nome da entidade que você criou. Na janela abaixo, clique em “Synthesize - XST” e selecione “Check Syntax” com um clique duplo. Corrigidos os eventuais erros de digitação, salve novamente e repita o processo. Se o processo de síntese for completado com sucesso, feche o editor VHDL.

### 3.2.2 Mapeamento dos sinais de entrada e saída do sistema em pinos do FPGA

Para que o circuito descrito possa ser implementado em FPGA, é necessário associar os sinais de entrada ( $a$ ,  $b$ ,  $c$  e  $d$ ) e de saída ( $s$ ) a pinos desse dispositivo. Em algumas situações, isso pode ser feito de forma automática; entretanto, como se dispõe da placa Basys2, é preferível neste exemplo forçar a associação das entradas a pinos que já estejam conectados a dispositivos de entrada (chaves) e saída (LEDs) da placa. Desta forma, é possível avaliar o funcionamento do sistema implementado sem a necessidade de circuitos ou componentes externos à placa.

Para tanto, clique em “Project\New Source” e escolha “Implementation Constraints File”. O nome do arquivo pode ser o mesmo do projeto (“detecpar.ucf”), e pode ser salvo na mesma pasta

que o código VHDL. Certifique-se de que a opção “Add to project” está marcada e clique em “Next”. Verifique suas seleções e clique em “Finish”.

O arquivo aberto será utilizado para o mapeamento dos pinos. Digite a descrição que se segue:

```
NET "a" LOC = "B4"; # SW3
NET "b" LOC = "K3"; # SW2
NET "c" LOC = "L3"; # SW1
NET "d" LOC = "P11"; # SW0
NET "s" LOC = "M5"; # LD0
```

A estrutura deste arquivo (denominado UCF, ou *universal constraints file*) é bastante simples. O que é feito aqui é a atribuição dos sinais  $a$ ,  $b$ ,  $c$ ,  $d$ ,  $s$  respectivamente aos pinos B4, K3, L3, P11 e M5 do FPGA. Os quatro primeiros estão ligados às chaves SW0, SW1, SW2 e SW3, enquanto que o último é ligado ao led LD0. Na placa, esse led está localizado bem acima da chave LD0, que é a chave mais à direita. A tabela 1 lista os nomes dos pinos correspondentes aos principais dispositivos de entrada e saída da placa Basys2.

Atenção para a sintaxe do código, em especial para o uso das aspas duplas, do igual e do ponto-e-vírgula. O texto após o símbolo # corresponde a um comentário, e é opcional. Salve então o arquivo e feche a janela com o arquivo UCF.

### 3.2.3 Etapa de síntese

A etapa seguinte, denominada síntese, consiste na compilação inicial do projeto, sem que no entanto seja gerado o arquivo com a configuração a ser enviada ao FPGA.

Na árvore hierárquica do projeto (janela da esquerda), clique no nome da entidade que você criou. Na janela abaixo, aplique um clique duplo em “Synthesize - XST”. Verifique se o processo de síntese for completado com sucesso.



**Tabela 1 - Nomes dos pinos correspondentes aos principais dispositivos de entrada e saída da placa Basys2.**

Dispositivos de entrada		Dispositivos de saída	
Chave	Pino	LED	Pino
SW7	N3	LD7	G1
SW6	E2	LD6	P4
SW5	F3	LD5	N4
SW4	G3	LD4	N5
SW3	B4	LD3	P6
SW2	K3	LD2	P7
SW1	L3	LD1	M11
SW0	P11	LD0	M5
Botão	Pino		
BTN3	A7		
BTN2	M4		
BTN1	C11		
BTN0	G12		

### 3.2.4 Etapa de implementação

Na fase de implementação, é gerado o arquivo de extensão “.bit”, com a configuração a ser enviada ao FPGA. Neste exemplo, será gerado o arquivo detector\_paridade\_4bits.bit.

Na árvore hierárquica do projeto (janela da esquerda), clique no nome da entidade que você criou. Na janela abaixo, aplique um clique duplo em “Implement Design”. Verifique se o processo de síntese for completado com sucesso.

A seguir, aplique um clique duplo em “Generate Programming File”. Se o processo for completado com sucesso, será criado o arquivo detector\_paridade\_4bits.bit na pasta do projeto.

### 3.2.5 Programação do FPGA

A programação propriamente dita do FPGA é feita a partir do arquivo “.bit” obtido na fase de implementação. Como neste experimento é utilizada a placa Basys2, o programa a ser utilizado para programação é o Adept, da Diligent, Inc.

Antes de começar, verifique se o *jumper* de modo, azul, localizado no canto direito superior da placa Basys2, está na posição PC. A seguir, verifique se a placa Basys2 está conectada à porta USB do PC e se a chave de liga/desliga está na posição ligada (“on”). Um LED vermelho (*status LED*) acenderá quando a placa está ligada.

Execute então o programa Adept. Ao ser ligado, o programa testa a placa. Se a inicialização falhar, feche o programa, repita as instruções do parágrafo acima e tente novamente. Se a mensagem “Initialization Complete” aparecer, siga em frente. Caso contrário, chame o professor ou monitor.

Na aba “Config”, há dois botões “Browse”. Um está associado ao FPGA e outro à PROM. Clique no botão “Browse” associado ao FPGA e selecione o arquivo detector\_paridade\_4bits.bit na pasta do projeto. Caso apareça um aviso sobre possíveis problemas associado ao clock de startup, ignore esse aviso e clique em “Sim” para continuar. A seguir, clique em “Program” para programar a placa. Se, novamente, o aviso aparecer, clique mais uma vez em “Sim”. Neste momento é realizada a programação do FPGA.

Verifique se a mensagem “Programming Successful” apareceu. Se tudo deu certo, teste agora o funcionamento do detector de paridade na placa. As chaves de entrada são indicadas por SW0, SW1, SW2 e SW3. Se o número de chaves acionadas por ímpar (uma ou três), o led LD0 deve acender. Caso contrário, ele deve apagar. Teste todas as combinações possíveis!

Ao terminar, solicite o visto ao professor o monitor. A seguir, desligue a chave liga/desliga da placa e passe ao próximo item do experimento.

## 3.3 Algumas estruturas em VHDL úteis ao experimento

Duas estruturas em VHDL, além daquelas abordadas no experimento 4, serão de grande utilidade para a realização deste experimento. Uma delas é um tipo de sinal/variável denominado STD\_LOGIC\_VECTOR, que consiste num vetor de bits de dimensão ajustável. A outra estrutura, denominada “with-select”, permite vincular o valor a ser atribuído a um sinal ou variável ao valor de um outro, que pode inclusive ser um vetor.

### 3.3.1 O tipo STD\_LOGIC\_VECTOR

Um sinal ou variável definido como de tipo STD\_LOGIC\_VECTOR pode ser visto como um conjunto de elementos do tipo STD\_LOGIC, acessados por meio de um índice. Considere como exemplos as seguintes declarações:

```
signal A: STD_LOGIC_VECTOR(2 downto 0);  
signal B: STD_LOGIC_VECTOR(0 to 2);
```

Os sinais A e B assim definidos são palavras de 3 bits, com índices 0, 1 e 2. Individualmente, A(2), A(1), A(0), B(0), B(1), B(2) consistem, cada um, num sinal do tipo STD\_LOGIC. Assim, podem ser feitas atribuições do tipo A(2) <= ‘0’, B(1) <= ‘1’, e assim por diante.

No entanto, uma das vantagens do STD\_LOGIC\_VECTOR é permitir realizar uma atribuição de valor para todos os bits de uma



palavra numa única linha de código VHDL. Assim, as linhas

```
A <= "100";  
B <= "100";
```

equivalem a:

```
A(2) <= '1';  
A(1) <= '0';  
A(0) <= '0';  
B(0) <= '1';  
B(1) <= '0';  
B(2) <= '0';
```

É importante observar a diferença entre os valores atribuídos a A e a B. No primeiro caso, como os índices de A foram definidos de 2 a 0 (signal A: STD\_LOGIC\_VECTOR(2 downto 0)), o primeiro bit à esquerda em “100” é atribuído a A(2), o segundo a A(1), o terceiro a A(0). No caso de B, os índices foram definidos em ordem crescente, de 0 a 2 (signal B: STD\_LOGIC\_VECTOR(0 to 2)), de forma que o primeiro bit em “100” é atribuído a B(0), o segundo a B(1) e o terceiro a B(2).

Na janela “Define Module” do ISE WebPACK, as opções “Bus”, “MSB” (*most significant bit*, ou bit mais significativo) e “LSB” (*least significant bit*, ou bit menos significativo) são usados para criar “ports” do tipo BIT\_VECTOR ou STD\_LOGIC\_VECTOR. Por exemplo, a configuração apresentada na figura 4 gera o modelo de código VHDL apresentado na figura 5.

Uma consideração final acerca do tipo STD\_LOGIC\_VECTOR diz respeito à implementação de um circuito digital nos FPGAs Xilinx. Se a descrição em VHDL é utilizada com este fim, é necessário associar cada bit de um vetor de entrada ou saída a um pino do circuito. No arquivo UCF, a sintaxe a ser utilizada é:

```
NET "A<i>" LOC = "Px";
```

onde *i* é o índice no vetor A do bit a ser associado ao pino Px do FPGA. Assim, caso se deseje associar o bit A(2) ao pino P47, por exemplo, deve-se incluir a seguinte linha no arquivo UCF:

```
NET "A<2>" LOC = "P47";
```

Isto será muito útil quando trabalharmos com os mostradores de 7 segmentos.

### Define Module

Specify ports for module.

Entity name

Architecture name

Port Name	Direction	Bus	MSB	LSB
x	in	<input checked="" type="checkbox"/>		
A	in	<input checked="" type="checkbox"/>	3	0
B	in	<input checked="" type="checkbox"/>	1	5
saída	out	<input checked="" type="checkbox"/>		

Figura 4 – Criação de “ports” do tipo STD\_LOGIC\_VECTOR no ISE WebPACK.

```
entity minhaentidade is  
    Port ( x : in STD_LOGIC;  
           A : in STD_LOGIC_VECTOR (3 downto 0);  
           B : in STD_LOGIC_VECTOR (1 downto 5);  
           saída : out STD_LOGIC);  
end minhaentidade;  
  
architecture minhaarquitetura of minhaentidade is  
begin  
  
end minhaarquitetura;
```

Figura 5 – Modelo de código VHDL gerado pelo ISE WebPACK a partir da configuração apresentada na figura 4.

### 3.3.2 A estrutura “with-select”

A estrutura “with-select” permite que a um dado sinal seja atribuído uma de várias expressões booleanas possíveis, escolhida a partir do valor de um segundo sinal. Sua construção mais básica é apresentada a seguir.

Nessa construção, o sinal 1 recebe a expressão booleana 1 se o sinal 2 assume o valor 1, mas recebe a expressão booleana 2 se o sinal 2 assume o valor 2, e assim por diante. Se o sinal 2 não assume nenhum dos valores relacionados entre valor1 e valorn, o sinal 1 recebe a expressão booleana n + 1.

A última linha da construção não é obrigatória; se todos os valores possíveis do sinal 2 estão relacionados entre valor1 e valorn, a última linha é omitida, e a construção se encerra com o sinal “;” colocado após a linha “(expressão booleana n+1) when others”. No entanto quando se trabalha com o tipo STD\_LOGIC (ao invés do tipo BIT), seria necessário cobrir todas as possibilidades possíveis, incluindo o valor indeterminado, irrelevante, etc., e não somente os valores 0 e 1. Assim, quase sempre é necessário incluir a opção “when others”.



```
with sinal2 select
    sinal1 <= (expressão booleana 1) when (valor1),
    (expressão booleana 2) when (valor2),
    (expressão booleana 3) when (valor3),
    ...
    (expressão booleana n) when (valorn),
    (expressão booleana n+1) when others;
```

Como um exemplo, considere esta implementação alternativa da função ou-exclusivo. Suponha que os dois bits de entrada são representados por uma única palavra de dois bits  $A$ . Se  $s$  é o sinal de saída, pode-se escrever:

```
with A select
    s <= '1' when "10",
    '1' when "01",
    '0' when others;
```

Assim,  $s$  recebe 1 se e somente se  $A(1) = 1$  e  $A(0) = 0$ , ou se  $A(1) = 0$  e  $A(0) = 1$ .

Finalmente, observe que se uma mesma expressão booleana deve ser atribuída ao sinal 1 para dois ou mais valores diferentes do sinal 2, como no último exemplo. É possível combinar esses valores numa única linha em VHDL, por meio do conectivo  $|$ , como mostrado a seguir:

```
with A select
    s <= '1' when "10" | "01",
    '0' when others;
```

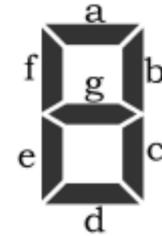
A interpretação neste caso é a seguinte:  $s$  recebe 1 se  $A$  é “10” ou “01”, e recebe 0 para todos os outros valores possíveis de  $A$ .

Este tipo de estrutura é muito útil na implementação de conversores de código.

### 3.4 Conversor de código binário para código de 7 segmentos

Um display ou mostrador de 7 segmentos é um dispositivo composto por 7 LEDs dispostos estratégicamente, de forma a constituir o dígito 8, conforme ilustrado na figura 6. Muitos mostradores de 7 segmentos dispõem ainda de um LED adicional, que implementa o ponto decimal.

A depender de quais LEDs estão acesos e quais estão apagados, o mostrador esquematizado permite representar qualquer dígito decimal, bem como as letras de A a F (algumas maiúsculas, outras minúsculas). Assim, um dígito hexadecimal é associado univocamente a uma configuração dos 7 segmentos, na qual alguns são acesos, outros não.



**Figura 6 - Disposição dos LEDs em um display de 7 segmentos.**

Nesse sentido, um conversor da representação binária para a representação em 7 segmentos do hexadecimal correspondente é um circuito que, dados 4 bits de entrada, retorna um nível lógico para cada segmento, baixo se ele deve ser aceso, alto em caso contrário, de forma a se obter a configuração associada àquele dígito. Trata-se, portanto, de um sistema com 4 entradas e 7 saídas.

A tabela 2 apresenta as possíveis combinações dos 4 bits de entrada, o dígito hexadecimal correspondente e a configuração de saída dos 7 LEDs, fornecida pelo circuito conversor.

A placa Basys2 dispõe de quatro mostradores de 7 segmentos. No entanto, há somente 12 pinos associados aos mostradores:

- 4 pinos para escolher qual(is) mostrador(es) está(ão) ligado(s);
- 7 pinos para acender cada segmento individual; e
- 1 pino para o ponto decimal.

A tabela 3 lista o nome de cada um desses pinos, para criação do arquivo UCF.

Os pinos da placa Basys2 listados na tabela 3 são ativos em nível baixo. Portanto, para acender um dos segmentos, ou para selecionar um dos mostradores, é preciso escrever o bit 0 no mesmo. Uma dica é, ao invés de trabalhar com a lógica invertida, simplesmente usar a operação “not( )” para inverter as saídas correspondentes aos segmentos e ao seletor de mostrador.

Assim, o se os pinos AN0, b e c estão em 0, enquanto os demais pinos listados na tabela 3 estão em 1, o dígito “1” será mostrado no primeiro mostrador. Da mesma forma, se os pinos AN2, a, b e



c estão em 0, enquanto os demais pinos estão em 1, então o dígito “7” aparecerá no terceiro mostrador.

**Tabela 2 - Configuração do display de 7 segmentos.**

Bits de entrada	Dígito hexadecimal	Configuração do display
0000	0	
0001	1	
0010	2	
0011	3	
0100	4	
0101	5	
0110	6	
0111	7	
1000	8	
1001	9	
1010	A	
1011	B	
1100	C	
1101	D	
1110	E	
1111	F	

**Tabela 3 - Nomes dos pinos associados aos mostradores de 7 segmentos.**

Seletor de mostrador	Segmentos		
Mostrador	Pino	Segmento	Pino
AN3	K14	a	L14
AN2	M13	b	H12
AN1	J12	c	N14
AN0	F12	d	N11
		e	P12
		f	L13
		g	M12
		ponto decimal	N13

## 4 PARTE EXPERIMENTAL

### 4.1 Detector de paridade de 4 bits

Execute o procedimento descrito passo-a-passo na seção 3.2.

### 4.2 Conversor de código binário de 4 bits para código de 7 segmentos

Implemente em VHDL um conversor do código binário para o código de 7 segmentos. Os quatro bits de entrada devem ser representados por um único vetor B, com índices de 0 a 3 sendo 3 o mais significativo. Caso ainda desconheça o código de 7 segmentos, consulte a seção 3.4.

Sugestão: represente os 7 bits de saída, a, b, c, d, e, f, g, também por um único vetor, e utilize a estrutura “with-select” para atribuir os valores a estes bits em função do vetor de entrada.

Associe cada sinal de entrada e de saída do conversor a um pino do FPGA (utilize para tanto um arquivo UCF, conforme descrito na seção 3.2.2 e nas tabelas 1 e 3). Para facilitar a avaliação do sistema implementado, utilize para entradas pinos conectados a chaves da placa Basys2 (SW3 a SW0) e para saídas pinos conectados ao mostrador de 7 segmentos AN0. Lembre-se de ativar o mostrador AN0, escrevendo 0 no bit correspondente. Lembre-se ainda que os 7 segmentos do mostrador também são ativos em nível baixo.

Realize as etapas de síntese e implementação do conversor. A seguir, com o programa Adept, programe o FPGA da placa Basys2 para que esta realize a conversão de código descrita em VHDL.

Avalie as saídas do FPGA para todas as 16 combinações possíveis de entrada, usando as chaves.



### 4.3 Comparador de duas palavras de 4 bits

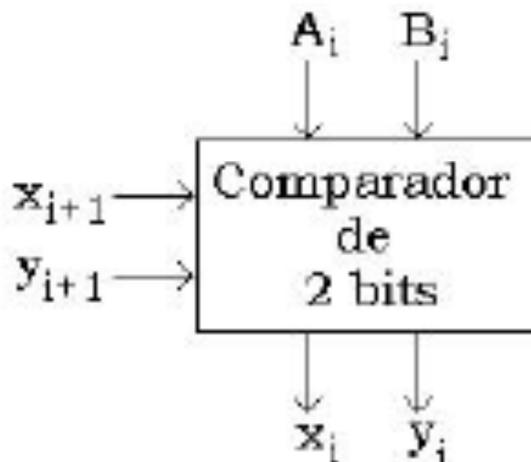
Um comparador de duas palavras A e B de  $n$  bits tem por objetivo determinar se o inteiro representado por A é menor, igual ou maior do que o representado por B. Como há três possibilidades ( $A < B$ ,  $A = B$ ,  $A > B$ ), pelo menos dois bits são necessários para codificar a saída. Uma alternativa é a apresentada na tabela 4.

**Tabela 4 - Codificação de sinais para um comparador bit-a-bit.**

entrada	resultado da comparação		saída
	$x_{i+1}y_{i+1}$	$A_iB_i$	
00	00	$A = B$	00
00	01	$A < B$	01
00	10	$A > B$	10
00	11	$A = B$	00
01	xx	$A < B$	01
10	xx	$A > B$	10
11	xx	entrada proibida	saída irrelevante

Obs.:  $x_iy_i = 11$  não é uma saída possível e, portanto,  $x_{i+1}y_{i+1} = 11$  também não é uma entrada permitida.

Quando comparamos dois números, o primeiro passo é comparar os dígitos mais significativos. Somente se eles forem iguais será necessário olhar o próximo dígito na ordem de significância, e assim por diante. O mesmo vale para números binários (mas temos comparação de bits ao invés de comparação de dígitos). Assim, um comparador de palavras de  $n$  bits pode ser implementado a partir de  $n$  comparadores de duas palavras de 1 bit. Para isso, é necessário que cada comparador fundamental receba como entradas, além dos bits  $A_i$  e  $B_i$  de mesma significância, os bits  $x$  e  $y$  provenientes da comparação dos bits imediatamente mais significativos ( $x_{i+1}$  e  $y_{i+1}$ ). A tabela 4 descreve esse comparador e a figura 7 apresenta um diagrama de blocos do mesmo.



**Figura 7 - Comparador de duas palavras de um bit para implementação de um comparador bit-a-bit de duas palavras de  $n$  bits.**

Procedimento:

- Faça uma descrição em VHDL de um circuito que implementa o comparador da figura 7. Para tal, use a codificação apresentada na tabela 4 (ver também tabela da questão 2 da seção 5.3).
- Apresente o esquemático (diagrama de blocos) de um comparador de duas palavras de 4 bits utilizando somente quatro componentes básicos como o da figura 7 e alguns sinais.
- Faça uma descrição em VHDL do circuito do item anterior, com 8 entradas e 2 saídas. Utilize para tanto a entidade definida no primeiro item como um componente do novo módulo. Use valores constantes nas entradas  $x$  e  $y$  do comparador dos bits mais significativos.
- Implemente o comparador de duas palavras de 4 bits em FPGA, usando como entradas as chaves SW7 a SW0, e como saída os LEDs LD0 e LD1. Associe cada sinal de entrada e de saída a um pino do FPGA (utilize para tanto um arquivo UCF, conforme descrito na seção 3.2.2 e nas tabelas 1 e 3).
- Realize as etapas de síntese e implementação do conversor. A seguir, com o programa Adept, programe o FPGA da placa Basys2 para que esta realize a comparação de palavras descrita em VHDL.
- Avalie as saídas do FPGA para várias combinações possíveis de entrada, usando as chaves.



## 5 INSTRUÇÕES PARA A REALIZAÇÃO DO EXPERIMENTO

### 5.1 Pré-relatório

O projeto a ser apresentado no início da aula deve conter:

- tabela verdade de conversão de código binário/7-segmentos, com 4 bits de entrada;
- tabela verdade do comparador de duas palavras de um bit apresentado na figura 7 e tabela 4. Ver tabela apresentada na questão 2 da seção 5.3.
- diagrama de blocos de um comparador de duas palavras de 4 bits utilizando somente quatro componentes básicos como o da figura 7 e alguns sinais.
- tabelas listando os nomes dos pinos de entrada e saída do FPGA a serem utilizados em cada um dos três vistos;

### 5.2 Vistos

Para conseguir os vistos, o grupo deve realizar os procedimentos descritos na seção 4 deste roteiro. O grupo deve chamar o professor para marcar o visto ao fim dos itens 4.1, 4.2 e 4.3. O primeiro visto vale 2 pontos e os demais valem 3 pontos cada. Os 2 pontos restantes correspondem à nota de pré-relatório. O grupo terá duas aulas para conseguir os vistos.

### 5.3 Relatório

O relatório é individual, deve ser feito à mão, e consiste em responder ao questionário abaixo. Não é necessário entregar um relatório formal, com introdução, metodologia, resultados, etc.

- 1) Apresente o código VHDL do conversor binário - 7 segmentos (incluso comentários explicando cada passo).

- 2) Complete a tabela verdade do comparador de palavras de 1 bit. Use “don’t cares” onde possível.

X <sub>i+1</sub>	Y <sub>i+1</sub>	A <sub>i</sub>	B <sub>i</sub>	X <sub>i</sub>	Y <sub>i</sub>
0	0	0	0		
0	0	0	1		
0	0	1	0		
0	0	1	1		
0	1	0	0		
0	1	0	1		
0	1	1	0		
0	1	1	1		
1	0	0	0		
1	0	0	1		
1	0	1	0		
1	0	1	1		
1	1	0	0		
1	1	0	1		
1	1	1	0		
1	1	1	1		

- 3) Monte os mapas de Karnaugh para X<sub>i</sub> e Y<sub>i</sub> e apresente as equações lógicas.
- 4) Apresente o código VHDL desse comparador utilizando as equações encontradas na questão anterior.
- 5) Usando a entidade criada na questão anterior, implemente um comparador de palavras de 3 bits:
  - a) apresente o diagrama de blocos
  - b) apresente o código VHDL

**Atenção:** Alunos que apresentarem códigos iguais serão reprovados na disciplina e o fato será comunicado à coordenação do curso para que as medidas disciplinares cabíveis sejam tomadas.