Universidade de Brasília

Prática de Eletrônica Digital I

Apostila do laboratório - 2016.2

PROFESSORES

LEONARDO AGUAYO
HENRIQUE MARRA TAIRA MENEGAZ
LOURDES MATTOS BRASIL
MARCUS CHAFFIM
MARIANA COSTA BERNARDES MATIAS

UnB - FGA
Agosto de 2016

Conteúdo

Parte I

Apresentação e Regras Gerais

AMBIENTAÇÃO 1

USO DO LABORATÓRIO E NORMAS DE DOCUMENTAÇÃO

1.1 Considerações Gerais

O laboratório é um complemento essencial das aulas teóricas. Assim, os experimentos estarão sincronizados, na medida do possível, com os tópicos vistos previamente em sala de aula.

A critério do professor, os experimentos serão realizados individualmente ou por uma dupla. No caso da opção por dupla, esta permanecerá a mesma durante todo o semestre. Caso haja desistência ou trancamento da disciplina por um aluno da dupla, haverá uma reordenação de dupla (caso dois alunos se encontrem na mesma situação), ou o aluno continuará a realizar os experimentos só. **Não será permitida a formação de trios.**

Para a realização dos experimentos, os alunos deverão apresentar um pré-relatório correspondente à prática que será realizada. O aluno (ou dupla) que não apresentar o pré-relatório - ou apresentá-lo incompleto - não poderá realizar o experimento, obtendo consequentemente nota zero na pratica em questão.

No início de uma aula típica, os alunos entregarão ao professor dois documentos: (I) o pré-relatório correspondente ao experimento do dia e (II) o relatório do experimento anterior. Tipicamente, o prazo de entrega do relatório é de **uma semana** a partir da data da realização do experimento.

Atenção: não será admitido plágio de qualquer espécie. Caso detectado, será punido com nota zero.

1.2 Normas para Elaboração de Pré-relatórios

O tempo de aula em laboratório é um tempo que não deve ser desperdiçado. É a chance que o aluno tem de enfrentar dificuldades inesperadas, aprimorar as habilida-

des de depuração e uso do raciocínio lógico para resolver problemas na presença do professor.

É de grande importância, portanto, o planejamento prévio e a utilização de uma documentação adequada. O pré-relatório é um documento direcionado para a **execução** do experimento. Em geral, no pré-relatório o aluno deverá realizar simulações dos circuitos presentes no experimento e responder questões referentes à pratica a ser realizada.

O documento do pré-relatório é de formato livre. Entretanto, deve conter:

1. **Cabeçalho** com identificação completa do documento, contendo:
   - Nome e código da disciplina;
   - Número e título do experimento;
   - Turma de laboratório;
   - Nome, assinatura e matrícula do autor;
   - Local e data.
2. **Respostas** às perguntas do roteiro, devidamente identificadas;
3. **Tabelas e diagramas** devidamente identificados, incluindo as tabelas de conexão;
4. **Diagramas esquemáticos** dos circuitos simulados, seguindo as mesmas normas dos especificados para o relatório.

1.3 Normas para Elaboração de Relatórios

As normas a seguir são válidas para relatórios, sejam eles escritos à mão ou em formato digital.

1.3.1 Estrutura do Relatório

O relatório é do aluno ou da **dupla que realizou o experimento**. As páginas devem ser numeradas e todas as figuras, gráficos e tabelas devem ter título e numeração. **Exemplo**: “Figura 2.1 - Diagrama lógico do circuito somador”. O relatório deve ter a seguinte estrutura:

1. **Capa**, contendo:
   - Nome e código da disciplina;
   - Número e título do experimento;
   - Turma de laboratório;
   - Nome, assinatura e matrícula dos autores;
   - Local e data.
2. **Sumário**, apresentando as partes constituintes do relatório com as respectivas paginações.

3. **Introdução**, indicando a delimitação do tema, apresentando a justificativa descrevendo o propósito do relatório.

4. **Parte Experimental**, descrevendo os passos realizados, dificuldades e soluções para os problemas encontrados. Aqui, deve-se apresentar uma descrição dos resultados encontrados em forma de figuras, gráficos e tabelas.

5. **Discussão** sobre os resultados encontrados, comentando **detalhadamente** as medições realizadas e dando a devida interpretação destas, informando se os objetivos da experimento foram alcançados. Esta é uma das partes mais importantes do relatório: aqui, há oportunidade para expressar os conhecimentos adquiridos na prática e fazer a interrelação com os fundamentos teóricos.

6. **Conclusões**, mostrando os êxitos e eventuais problemas encontrados na realização do experimento, indicando as limitações, apresentando recomendações e/ou sugestões.

7. **Referencias Bibliográficas**, relacionadas e citadas de acordo com as normas da ABNT.

8. **Diagramas Esquemáticos. Todos** os diagramas devem ser inseridos **ao final do relatório em páginas separadas do texto**, indicando a identificação do circuito, autor, revisor, versão e datas relevantes.

   Para o tamanho dos diagramas, há apenas **duas** opções: dois diagramas por página (orientação retrato, para circuitos mais simples) ou um único diagrama por página (orientação paisagem, para circuitos mais complexos).

   Os diagramas devem conter a pinagem e identificação de **todos** os componentes, como mostrado na Figura 1.1.

   Cabe aqui listar alguns princípios orientadores para desenhar diagramas esquemáticos:

   - As diferentes funções desempenhadas pelo circuito devem se localizar em regiões distintas. Use este princípio sempre, mesmo que ao custo de deixar algumas áreas em branco para separar visualmente os grupos funcionais. Se necessário, divida o seu projeto em mais de um esquemático. Use uma seta (por exemplo ⇨) para indicar sinais que vão de um diagrama a outro.
   - Use um ponto para indicar conexões entre fios.
   - Sempre que possível, alinhe os componentes na horizontal ou vertical.

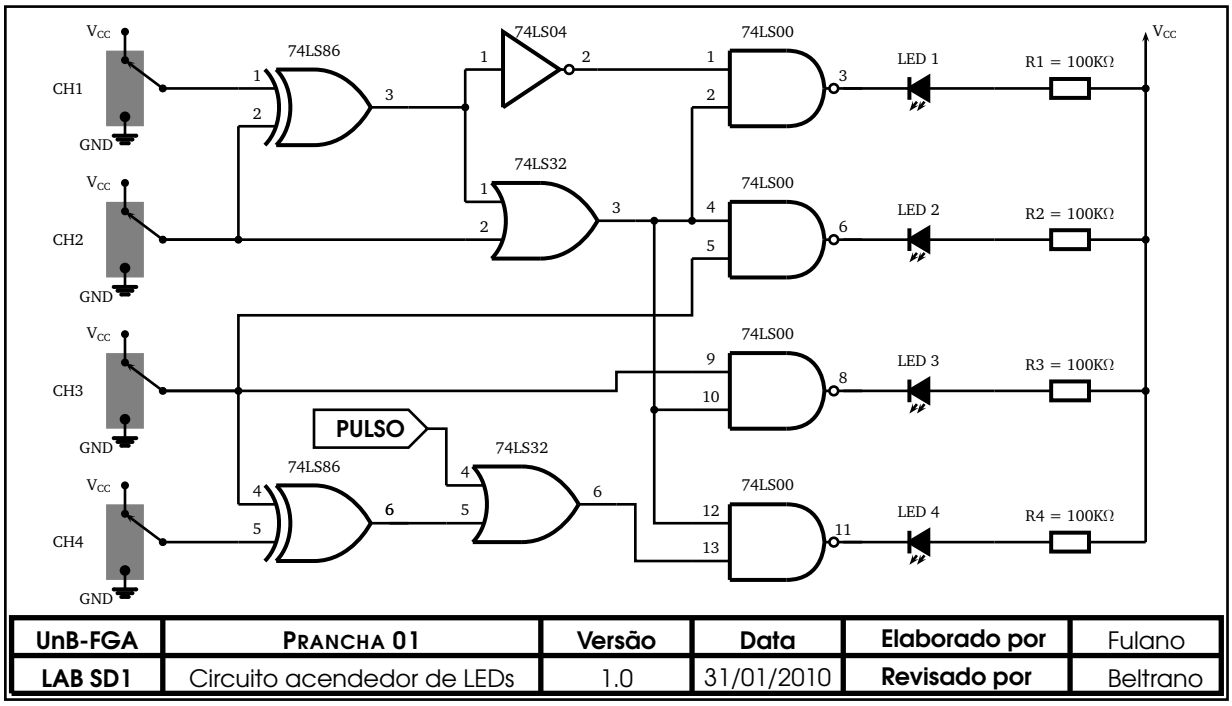


Figura 1.1: Exemplo de diagrama esquemático.

Parte II

Experimentos

EXPERIMENTO 1

PORTAS LÓGICAS

1.1 Objetivos

Apresentar ao aluno as portas lógicas básicas e o ambiente de simulação do laboratório.

1.2 Álgebra de Boole e Circuitos de Chaveamento

Em 1847, o matemático, filósofo e lógico britânico George Boole publicou seu livro "*The Mathematical Analysis of Logic*", onde apresentou uma maneira de quantificar proposições lógicas que são classificadas em duas classes: verdadeiro (**V**) ou falso (**F**). Ele mostrou que sentenças lógicas (por exemplo, 'todo X é Y') podem ser representadas por equações algébricas, e por consequência, a manipulação das proposições lógicas podem ser feitas pela manipulação de equações algébricas - daí o nome de álgebra booleana.

Algum tempo depois, em 1904, O matemático americano Edward Vermilye Huntington formalizou matematicamente a álgebra booleana do ponto de vista axiomático e apresentou os seguintes postulados, aqui já apresentados considerando que há apenas dois valores possíveis, 0 e 1, para as variáveis $\mathbf{X}, \mathbf{Y}, \mathbf{Z}$:

P1. $\mathbf{X} + 0 = \mathbf{X}$

P2. $\mathbf{X} \cdot 1 = \mathbf{X}$

P3. $\mathbf{X} + \mathbf{Y} = \mathbf{Y} + \mathbf{X}$

P4. $\mathbf{X} \cdot \mathbf{Y} = \mathbf{Y} \cdot \mathbf{X}$

P5. $\mathbf{X} + \overline{\mathbf{X}} = 1$

P6. $\mathbf{X} \cdot \overline{\mathbf{X}} = 0$

P7. $\mathbf{X} + (\mathbf{Y} \cdot \mathbf{Z}) = (\mathbf{X} + \mathbf{Y}) \cdot (\mathbf{X} + \mathbf{Z})$

P8. $\mathbf{X} \cdot (\mathbf{Y} + \mathbf{Z}) = (\mathbf{X} \cdot \mathbf{Y}) + (\mathbf{X} \cdot \mathbf{Z})$.

Nos postulados de Huntington, a operação $+$ é chamada de **OU**, a operação $\cdot$ de **E** e a operação **NÃO** (ou complemento) é representada por uma barra: $\overline{0} = 1$ e $\overline{1} = 0$.

As operações têm relação direta com os conectivos lógicos do cálculo proposicional encontrados em lógica: $\vee$, $\wedge$ e $\neg$.

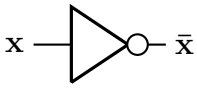
Além estes postulados, para analisar circuitos lógicos também são úteis as seguintes expressões:

P9. $\mathbf{X + X = X}$ **P11.** $\overline{\mathbf{X + Y}} = \overline{\mathbf{X}} \cdot \overline{\mathbf{Y}}$

P10. $\mathbf{X \cdot X = X}$ **P12.** $\overline{\mathbf{X \cdot Y}} = \overline{\mathbf{X}} + \overline{\mathbf{Y}}$

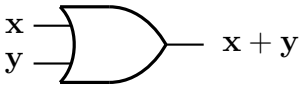
As três operações são os blocos básicos da eletrônica digital: circuitos complexos são criados a partir da combinação destas três operações, de modo a prover uma certa relação entre entradas e saídas. Esta relação entrada-saída é chamada de **função booleana**, e pode ser representada por uma **tabela-verdade**. No contexto desta disciplina, a função booleana é implementada por um circuito eletrônico. Não entraremos nos detalhes do seu funcionamento interno, mas no Apêndice 2 apresentamos alguns circuitos comuns.

Os três blocos básicos mencionados também são funções booleanas elementares, com as seguintes tabelas-verdade e símbolos gráficos:



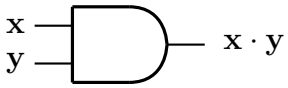
Porta **NÃO (NOT)**

x	$\bar{x}$
0	1
1	0



Porta **OU (OR)**

x	y	x + y
0	0	0
0	1	1
1	0	1
1	1	1



Porta **E (AND)**

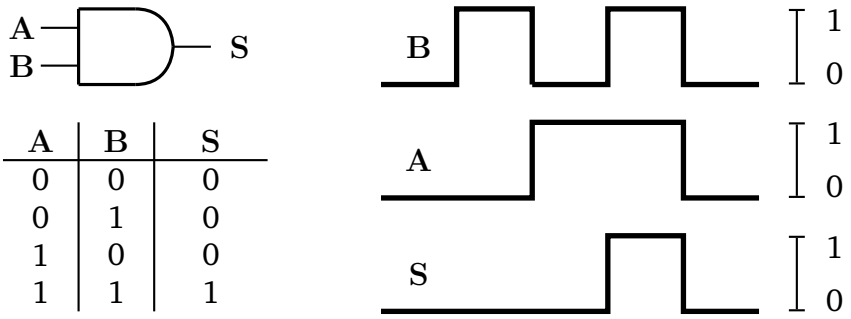
x	y	x · y
0	0	0
0	1	0
1	0	0
1	1	1

Figura 1.1: Funções booleanas elementares e seus símbolos.

Em 1938, o matemático, engenheiro elétrico e criptografista americano Claude Elwood Shannon mostrou em sua dissertação de mestrado que a álgebra booleana pode ser utilizada para descrever o comportamento de circuitos elétricos operados com chaves (ou relés) - daí o nome 'circuitos de chaveamento'. Este trabalho explicitou o vínculo entre os fundamentos matemáticos presentes na álgebra booleana e circuitos elétricos, permitindo a criação de sistemas digitais complexos sobre uma base teórica sólida.

Resumindo, do ponto de vista matemático funções booleanas mapeiam variáveis de entrada definidas no conjunto $\{\mathbf{0}, \mathbf{1}\}$ em saídas também definidas no conjunto $\{\mathbf{0}, \mathbf{1}\}$. Circuitos complexos têm múltiplas entradas e múltiplas saídas, e cada saída é caracterizada por uma função booleana específica das entradas.

A maneira de representar a relação entre entradas e saídas pode ser feita de uma segunda maneira, além da tabela-verdade. Podemos imaginar as entradas mudando dinamicamente no tempo na forma de **sinais de entrada**, de modo a produzir um **sinal de saída**. Para caracterizar o circuito, os valores lógicos dos sinais de entrada devem contemplar todas as possibilidades descritas na tabela-verdade.



A	B	S
0	0	0
0	1	0
1	0	0
1	1	1

Figura 1.2: Usando formas de onda para caracterizar circuitos lógicos.

A Figura 1.2 mostra como podemos usar sinais para caracterizar o circuito. A entrada **B** oscila no tempo entre os níveis lógicos 0 e 1, gerando uma forma de onda quadrada. Simultaneamente, a entrada **A** também o faz, mas não com o mesmo formato: note que o período desta forma de onda é o dobro do período da outra entrada[1].

Cabe aqui apresentar uma maneira de se implementar portas lógicas em circuitos integrados usada extensivamente no século passado durante as décadas de 70 e 80. A Figura 1.3 mostra a identificação dos terminais do CI 74LS32 que contém quatro portas OR de duas entradas. Observe nesta figura que os pinos 14 (VCC) e 7 (GND) devem ser ligados à fonte de alimentação nos terminais positivo e negativo, respectivamente. As entradas e saídas são especificadas com valores de tensão de aproximadamente 5 V para nível lógico **1** e de 0 V para nível lógico **0**, em consonância com os valores de tensão para VCC e GND.

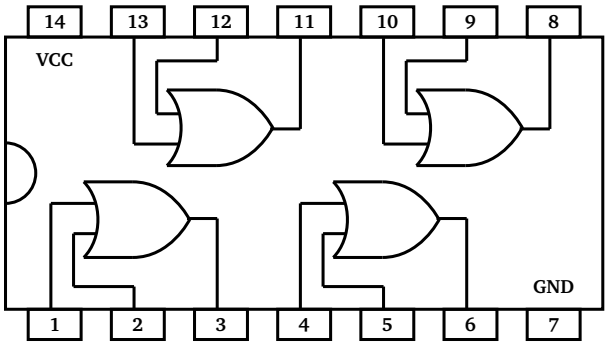
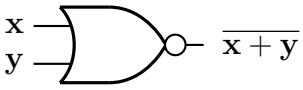


Figura 1.3: Identificação dos terminais do CI 74LS32 (vista superior).

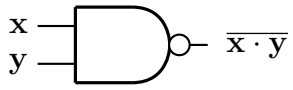
Outras portas básicas e úteis na síntese de circuitos digitais estão mostradas na Figura 1.4 com suas respectivas tabelas-verdade. Observe o uso de um círculo à saída da porta para indicar a inversão do nível lógico. No jargão, dizemos que as saídas das portas NOR e NAND estão "barradas".

[1]Se aplicássemos a mesma forma de onda nas duas entradas, a saída correspondente contemplaria apenas os casos da tabela-verdade quando $\mathbf{A} = \mathbf{B}$.



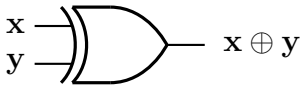
Porta **NOR**

x	y	$\overline{x+y}$
0	0	1
0	1	0
1	0	0
1	1	0



Porta **NAND**

x	y	$\overline{x \cdot y}$
0	0	1
0	1	1
1	0	1
1	1	0



Porta **XOR**

x	y	$x \oplus y$
0	0	0
0	1	1
1	0	1
1	1	0

Figura 1.4: Funções booleanas auxiliares NOR (NÃO-OU), NAND (NÃO-E) e XOR (OU-EXCLUSIVO) com seus símbolos e tabelas-verdade.

A porta NAND tem uma caraterística especial: a partir dela, é possível implementar todas as outras portas básicas, pois:

- $\overline{X \cdot X} = \overline{X}$ (porta NÃO);
- $\overline{X \cdot Y} = \overline{X} + \overline{Y}$ (porta OU com as entradas barradas).

A mesma propriedade pode ser encontrada para portas NOR, já que pelas expressões **P11** e **P12**, chamados de teoremas de De Morgan, NAND e NOR são intercambiáveis.

Se tiver curiosidade, procure os CIs correspondentes às portas lógicas mencionadas até o momento e observe a pinagem das portas. Todos os CIs tem pinagens iguais, em termos de entradas e saídas?

1.3 Usando VHDL

VHDL é uma linguagem que possibilita descrever e simular um circuito eletrônico digital qualquer. Em Inglês, significa *VHSIC Hardware Description Language*. VHSIC, por sua vez, significa *Very High Speed Integrated Circuits*.

A partir da descrição do circuito digital usando VHDL em um arquivo texto, é possível implementá-lo fisicamente em um tipo especial de *chip* após realizar um processo de compilação e gravação. Este tipo *hardware* é reconfigurável, com duas grandes vertentes no mercado: CPLDs (*Complex Programmable Logic Devices*) e FPGAs (*Field Programmable Gate Arrays*). Por ser padronizada, a linguagem é portável, isto é, pode ser usada em mais de um tipo de hardware reconfigurável. No momento, não entraremos em qualquer detalhe sobre como este tipo de *chip* funciona; por hora, basta saber que é possível descrever um circuito lógico em um arquivo texto e fazer com que o *chip* CPLD ou FPGA execute a função determinada pelo arquivo texto.

Em um certo sentido, vocês já fazem isto ao programar um computador para realizar uma determinada função. Mas aqui, ao usar VHDL, há uma diferença essencial. Com VHDL, o que fazemos é alterar as conexões internas no *hardware* de um CPLD ou FPGA para realizar uma função; em contraste, ao compilar um programa, fazemos com que um *hardware* já pronto - o microprocessador - realize a função desejada. Este ponto é

muito importante, pois ao descrever um circuito usando VHDL devemos estar atentos a certos aspectos que simplesmente não aparecem ao usarmos uma linguagem de programação como **C** ou Java. Daí sempre ter em mente da linguagem VHDL ter a função de **descrever** um *hardware*, e não de servir de plataforma para *programação*.

Em um ponto, VHDL é radicalmente diferente das outas linguagens de programação: as instruções em VHDL que descrevem o comporatemnto de um circuito são executadas **simultaneamente**, a menos que explicitemos qual porção de código será executada sequencialmente.Trata-se portanto de um código concorrente, adequado para processamento paralelo de dados.

Um outro aspecto importante: a linguagem permite tanto a realização da **síntese** do circuito quanto a sua **simulação**. Em um primeiro momento, usaremos VHDL para descrever e simular os circuitos projetados; em um segundo, vamos sintetizá-los nos *kits* disponíveis no laboratório.

1.3.1 Um *"Hello World"* em VHDL

Eis um exemplo básico de código VHDL, apresentado aqui apenas para ilustrar como um circuito simples pode ser descrito.

```
------------------------------------------------------------
-- Declaração das bibliotecas -- isto é um comentário
------------------------------------------------------------
LIBRARY ieee; -- o IEEE padronizou a linguagem
USE ieee.std_logic_1164.all; -- uso toda a biblioteca (.all)
LIBRARY std;
USE std.standard.all;
------------------------------------------------------------
-- Especificação das entradas e saídas
------------------------------------------------------------
ENTITY Hello_world IS
   PORT(
     x, y  : IN  BIT; -- x e y são entradas do tipo BIT
     z     : OUT BIT  -- z é saída, do tipo BIT
   );
END Hello_world;
------------------------------------------------------------
-- Descrição do funcionamento ou comportamento
------------------------------------------------------------
ARCHITECTURE Minha_arq OF Hello_world IS
BEGIN
     z <= x AND y; -- isto é uma porta 'E'
END Minha_arq;
```

O texto tem três seções: na primeira, declaram-se as bibliotecas que serão usadas. Tipicamente, as bibliotecas necessárias são **ieee** e **std**. Elas permitem o reuso de componentes e definições, e é possível criar bibliotecas específicas para uso dentro de projetos.

Na segunda seção, iniciada pela diretiva **ENTITY**, há uma especificação das entradas e saídas do circuito, e qual o tipo da porta (entrada, saída, etc.).

Finalmente, há a seção onde o funcionamento do circuito é descrito, indicada pela diretiva **ARCHITECTURE**. Dentro do contexto desta disciplina, esta é a seção que demanda a realização de um projeto de circuito **antes** da escrita do código, já que a linguagem VHDL apenas descreve o circuito. Enfatizamos novamente (i): a relação entre entradas e saídas do circuito é determinada por um circuito digital que implementa uma função booleana, e (ii): em um primeiro momento, é de fundamental importância saber **projetar** o circuito, para então descrevê-lo. Aqui, o circuito vem antes do código.

Para circuitos combinacionais simples, o "esqueleto" de código apresentado é suficiente para simular seu comportamento. As técnicas necessárias para projetar os circuitos digitais e para realizar uma eventual otimização serão apresentadas ao longo do curso.

No Apêndice X encontra-se uma breve introdução à linguagem VHDL, que não substitui material específico e completo que pode ser encontrado nas referências.

1.4 O Ambiente de Simulação

A verificação do funcionamento de um circuito pode ser realizada simulando em um software adequado a relação entre suas entradas e saídas.

Há muitas opções de simuladores para simulação de circuitos digitais. Usaremos neste curso dois simuladores diferentes: um introdutório (QUCS), para fixar os conceitos mais básicos, e um segundo e mais complexo (ISE ou Vivado[2]) para não apenas simular, mas também sintetizar os circuitos em um CI FPGA. O aluno é encorajado a aprender **sozinho** o uso de outras ferramentas (há dezenas de tutoriais na Internet), e identificar as limitações das mesmas quando comparadas às de uso profissional.

1.4.1 Simulando uma Porta Lógica no QUCS

O *software* livre QUCS permite a simulação de circuitos analógicos e digitais. Para iniciar uma simulação de um circuito digital, basta criar um novo projeto e selecionar `components` dentro da aba à esquerda, e então escolher a porta lógica desejada em `digital components`.

O QUCS apresenta resultados de duas maneiras: pela tabela-verdade, ou pelas formas de onda das entradas e saídas. Para circuitos com muitas entradas e saídas, em geral é mais conveniente e prático analisar as formas de onda.

No exemplo da Figura 1.5, está a simulação do comportamento de uma porta **E** de duas entradas. Note que há um gerador de sinal aplicado a cada entrada, e todos os pontos de interesse (A, B e C) têm um nome (definidos pelo botão `Wire Label`). A nomeação destes pontos é importante, pois sem isto não é possível construir a tabela-verdade ou as formas de onda.

Para simular o circuito, deve-se inicialmente estipular qual o tipo de simulação. Esta opção é dada dentro da caixa `digital simulation`, que pode ser encontrada ao final da lista em `digital components`.

[2]Ambos da fabricante de FPGA Xilinx. Existe um similar da fabricante Altera, o simulador Quartus. As versões básicas são todas grátis.

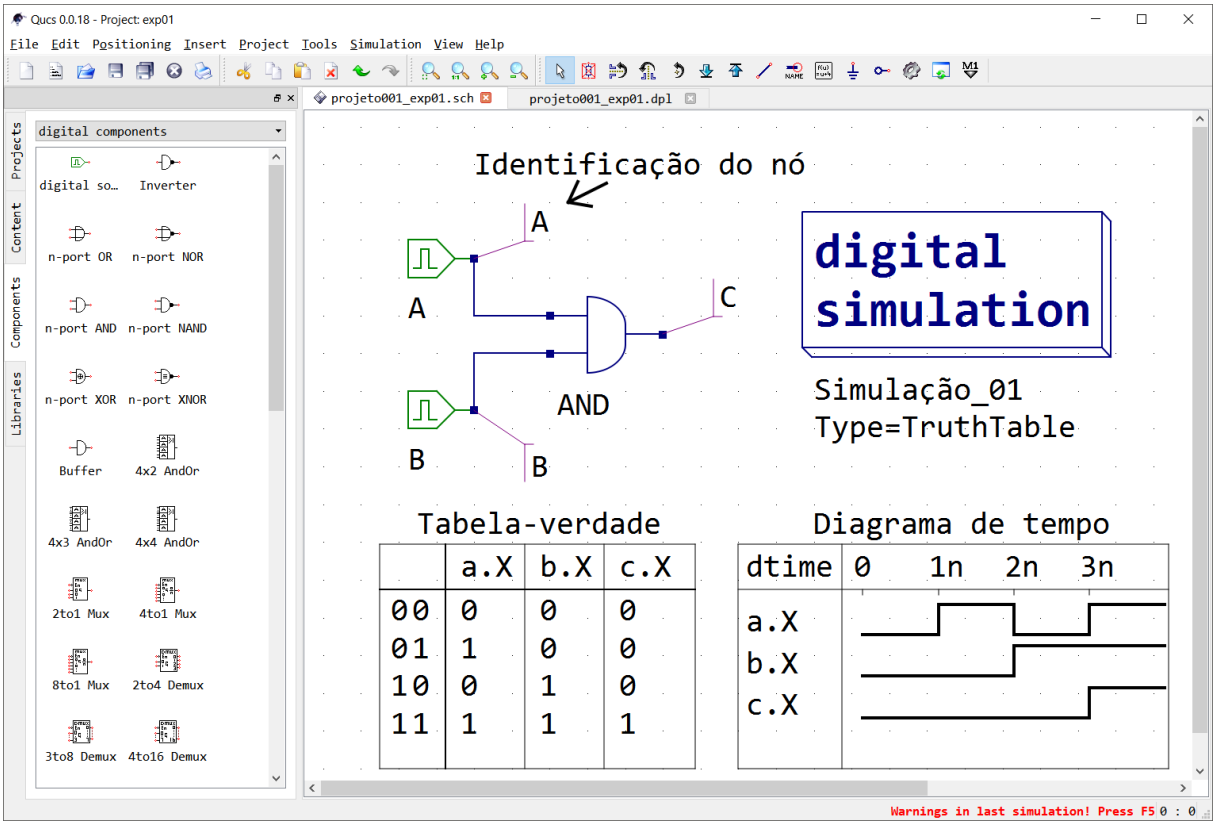


Figura 1.5: Tela do simulador QUCS com a tabela-verdade e formas de onda de interesse para uma porta **E**.

1.4.2 Usando QUCS para Simular VHDL

Para circuitos simples, é possível usar o QUCS para simular um circuito descrito por código VHDL. Basta arrastar para a tela de esquemático o bloco `VHDL` e vincular ao mesmo um arquivo previamente editado com extensão `.vhd`. O número de entradas e saídas do bloco é determinado pelas definições da diretiva **`PORT`** do código VHDL. Neste caso, o arquivo foi o seguinte:

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

entity minha_xor is
   port
   (x, y: in BIT; z: out BIT);
end minha_xor;

architecture minha_xor_arch of minha_xor is
begin
z <= x xor y;
end minha_xor_arch;
```

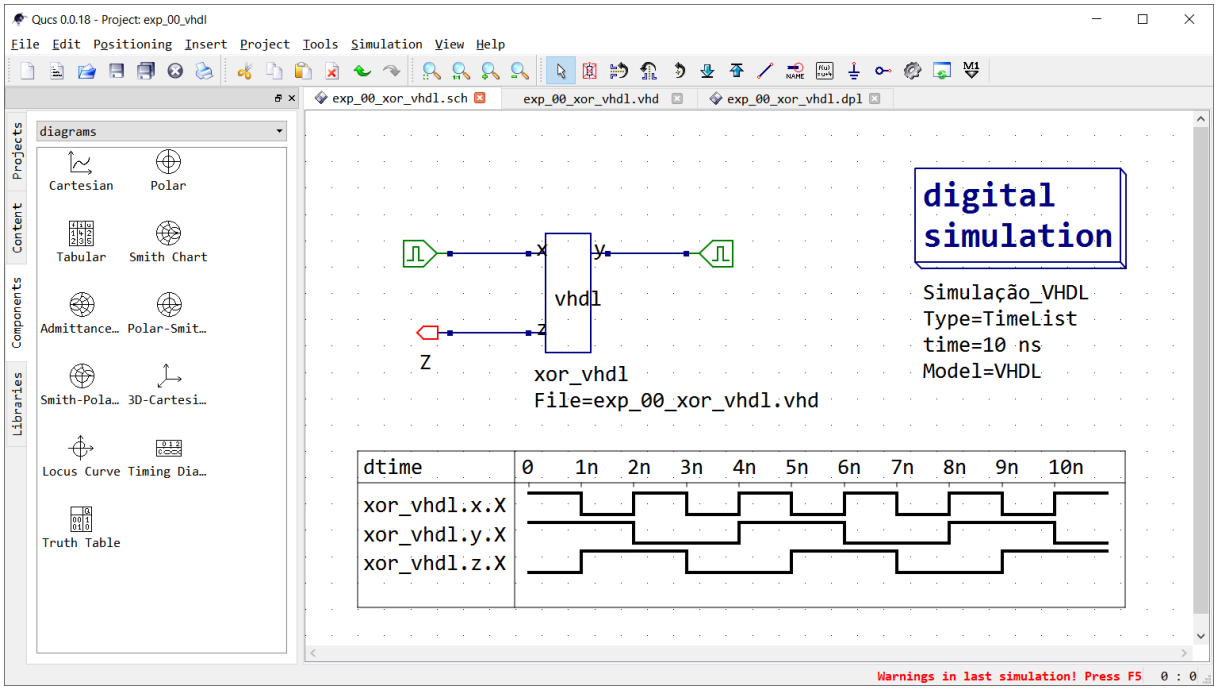


Figura 1.6: Tela do simulador QUCS com simulação VHDL.

1.5 Atividades em Laboratório

Siga as orientações do professor para ativar o ambiente de simulação em seu computador ou no disponível em sua bancada. Para todas as atividades abaixo, capture a tela de seu computador como comprovante da realização de sua atividade.

Atividade 1. Simule, utilizando a ferramenta disponível, o circuito correspondente a uma porta NAND. Apresente as formas de onda à entrada e saída do circuito.

Atividade 2. Apresente a tabela-verdade e as formas de onda à entrada e saída de uma porta XNOR.

Atividade 3. Apresente o código VHDL de uma porta NAND e de uma XNOR. Usando o simulador indicado pelo professor, gere as formas de onda à entrada e saída destas duas portas e compare com as obtidas anteriormente.

Atividade 4. Projete um circuito que realiza a função lógica AND de **três** variáveis booleanas. Apresente a tabela-verdade, simule seu circuito e mostre as formas de onda nas entradas e na saída.

1.6 Relatório

Em seu relatório de atividades, apresente:

1. Os passos seguidos para configurar o ambiente de simulação;

2. As telas capturadas durante as atividades no ambiente de laboratório;
3. Uma comparação entre os métodos de descrição das portas lógicas (circuito usando símbolos lógicos, tabela-verdade, expressão booleana, código VHDL);
4. Simulação do **circuito** da Atividade 4 em um simulador **diferente** do usado em sala, apresentando as formas de onda correspondentes. Compare os dois simuladores levando em conta os seguintes aspectos:
    - Facilidade de uso;
    - Possibilidade de uso de VHDL;
    - Flexibilidade na apresentação dos resultados desejados.

EXPERIMENTO 2

CIRCUITOS LÓGICOS COMBINACIONAIS

2.1 Objetivos

Aprimorar a visão do aluno quanto à metodologia e implementação de um Circuito Lógico Combinacional e às implicações das decisões de projeto.

2.2 Introdução

Os circuitos lógicos combinacionais são aqueles onde o nível lógico da(s) saída(s), em qualquer instante de tempo, depende única e exclusivamente, dos níveis lógicos presentes nas entradas naquele instante. Em outras palavras, são circuitos que não possuem a característica de memória: valores passados das entradas não afetam as saídas, apenas os valores presentes o fazem.

O estudo dos circuitos combinacionais é importante para compreender o funcionamento de circuitos utilizados na construção de computadores e em vários outros sistemas digitais, como por exemplo, os circuitos somadores, subtratores, codificadores, entre outros. De forma geral, os circuitos combinacionais podem ser usados para solucionar problemas em que se necessita de uma resposta a condições específicas representadas pelas variáveis de entrada. A relação entre entradas e saídas descreve o funcionamento do circuito, e esta relação pode ser expressa de diversas formas:

- Por uma descrição textual;
- Pela tabela-verdade dos valores de cada saída, expressos em função dos valores das entradas;
- Por meio de um conjunto de equações booleanas, com regras e propriedades dadas pela álgebra de Boole;
- Por um diagrama chamado de **Mapa de Karnaugh**.

2.3 Projeto de Circuitos Combinacionais

A Figura 2.1 ilustra uma possível sequência do processo para o projeto de circuitos digitais combinacionais.

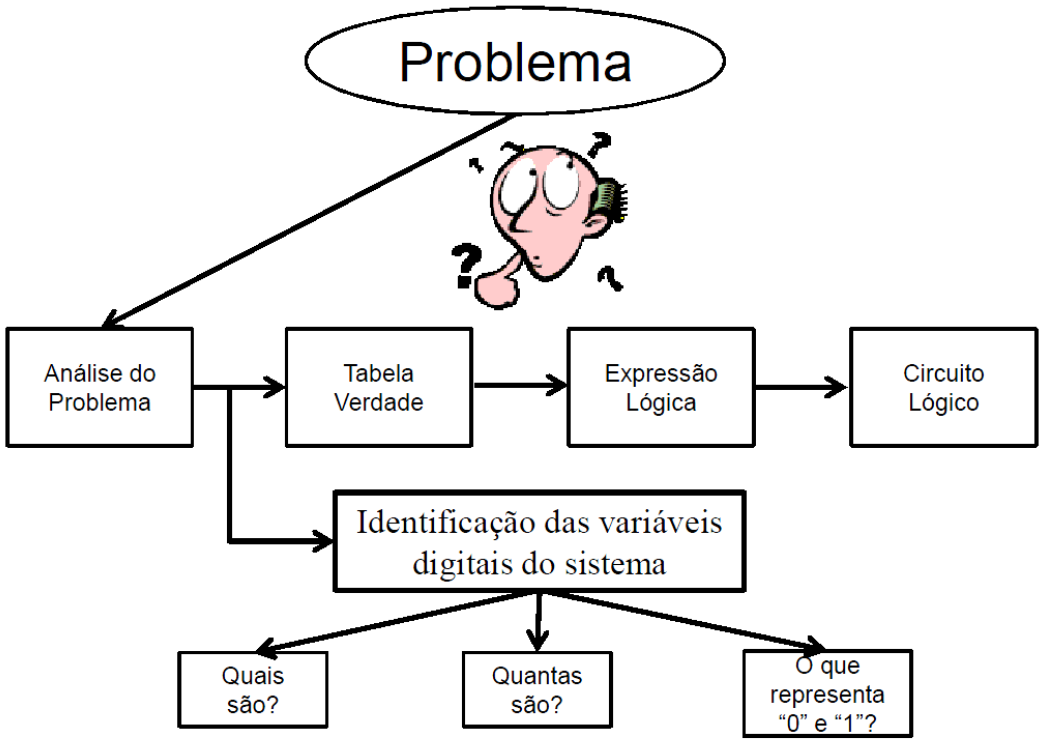


Figura 2.1: Ilustração da metodologia a ser seguida para o projeto de circuitos combinacionais.

O processo se inicia com uma análise detalhada e profunda do problema que deve ser resolvido. Esta análise deve permitir a descrição do problema de forma completa, por exemplo na forma de uma **tabela-verdade**. Esta é a etapa mais difícil e requer experiência do projetista, pois não é possível torná-la um processo algorítmico, passível de uma metodização absoluta devido, sobretudo, às nuances e características de cada problema. O importante neste ponto é ter segurança de que o problema está bem especificado. Caso contrário, corre-se o risco de "resolver certo o problema errado".

Apesar de não haver uma sequencia de passos determinada para projetar, como uma "receita de bolo", em geral a identificação correta das variáveis de entrada e saída é um passo que auxilia na construção da especificação. Nesse processo, o projetista procura responder algumas perguntas, como por exemplo:

- Quantas são as variáveis de entrada e saída?
- Quais são?
- O que representa os níveis lógicos ZERO e UM?

Após essas definições deve-se realizar as combinações das variáveis de entrada e montar, por exemplo, a tabela-verdade para cada saída.

Após a obtenção da tabela-verdade, o próximo passo é gerar a expressão booleana de cada saída. Uma forma de realizar esse procedimento é escrever o termo AND (produto) para cada caso em que a saída esteja em nível lógico alto e depois escrever a expressão de soma de produtos para a saída.

A obtenção da expressão lógica do problema já permite a implementação do circuito. No entanto, é altamente recomendável simplificar a expressão lógica obtida de forma a obter um circuito mais simples e, consequentemente, mais barato. O processo de simplificação pode ser realizado através da álgebra de Boole ou através da utilização de Mapas de Karnaugh.

Antes de implementar em *protoboard* ou em FPGA o circuito lógico obtido da expressão final simplificada, recomenda-se simular o circuito em software apropriado, de forma a validar o projeto e assim evitar perda de tempo na montagem de circuitos errados.

2.3.1 Um Exemplo Simples de Projeto de Circuito Combinacional

Considere o projeto de um circuito combinacional que realize a seguinte função: por intermédio de uma chave seletora, apresenta em sua saída uma de duas entradas, **A** ou **B**. Se a posição da chave for fechada, a saída é a entrada **A**. Caso contrário, a saída é a entrada **B**.

Vamos construir a tabela-verdade do circuito. Notem que há três entradas: **A**, **B** e a seleção (que chamaremos de **C**). A saída será denominada de **S**. Também devemos atribuir valores lógicos para a variável **C**. Neste caso, adotamos: **C** fechada $\rightarrow$ **0** e **C** aberta $\rightarrow$ **1**. Com estas primeiras identificações, podemos construir a tabela-verdade:

C	**A**	**B**	**S**
0	0	0	0
0	0	1	0
0	1	0	**1**
0	1	1	**1**
1	0	0	0
1	0	1	**1**
1	1	0	0
1	1	1	**1**

Tabela 2.1: Tabela-verdade de circuito combinacional.

Uma observação com relação à construção da tabela: as colunas foram especificadas de maneira sistemática, de modo a permitir que todas as possibilidades de valores para as variáveis de entradas estejam contempladas.

A partir da tabela, podemos verificar para quais valores de saída a função booleana é **1**: são os casos indicados em negrito. Tomemos um dos casos, para **C**=0, **A**=1 e **B**=0. Chamemos este caso de $\mathbf{S}_2$ (se iniciarmos a contagem das linhas em 0, esta será a

segunda linha, onde **S** vale 1). Para que $\mathbf{S}_2$ seja 1, todas as variáveis **C**, **A** e **B** devem ter estes valores específicos (0, 1 e 0 respectivamente). Se tomarmos então

$$\mathbf{S}_2 = \overline{\mathbf{C}} \text{ AND } \mathbf{A} \text{ AND } \overline{\mathbf{B}} = \overline{\mathbf{C}}\mathbf{A}\overline{\mathbf{B}}, \tag{2.1}$$

esta condição estará automaticamente satisfeita.

A partir da tabela-verdade, é possível observar que em qualquer um dos quatro casos indicados em negrito, a função **S** apresenta nível lógico 1. Portanto, podemos escrever para a saída **S**:

$$\mathbf{S} = \mathbf{S}_2 + \mathbf{S}_3 + \mathbf{S}_5 + \mathbf{S}_6 = \overline{\mathbf{C}}\mathbf{A}\overline{\mathbf{B}} + \overline{\mathbf{C}}\mathbf{A}\mathbf{B} + \mathbf{C}\overline{\mathbf{A}}\mathbf{B} + \mathbf{C}\mathbf{A}\mathbf{B}. \tag{2.2}$$

Como a função booleana ou tem valor 0 ou valor 1, a especificação está completa, pois nos outros casos (por exemplo, $\mathbf{S}_0$) ela fica automaticamente definida com o valor 0. Basta, então, especificar as condições para quando **S** vale 1, chamada convenção de **lógica positiva**.

Um próximo passo é a simplificação da função booleana, usado essencialmente para reduzir custos. Um critério possível para direcionar a otimização é minimizar o número de portas lógicas utilizadas. Para tanto, vamos simplificar a expressão de **S** usando propriedades de álgebra booleana:

$$\overline{\mathbf{C}}\mathbf{A}\overline{\mathbf{B}} + \overline{\mathbf{C}}\mathbf{A}\mathbf{B} + \mathbf{C}\overline{\mathbf{A}}\mathbf{B} + \mathbf{C}\mathbf{A}\mathbf{B} = \overline{\mathbf{C}}\mathbf{A}\underbrace{\left(\overline{\mathbf{B}} + \mathbf{B}\right)}_{1} + \mathbf{C}\mathbf{B}\underbrace{\left(\overline{\mathbf{A}} + \mathbf{A}\right)}_{1} =$$

$$\overline{\mathbf{C}}\mathbf{A} \cdot \mathbf{1} + \mathbf{C}\mathbf{B} \cdot \mathbf{1} = \overline{\mathbf{C}}\mathbf{A} + \mathbf{C}\mathbf{B}. \tag{2.3}$$

O circuito final está representado na Figura 2.2. Quando **C**=0, a porta AND inferior tem saída nula, e a superior tem saída **A**. O sinal à saída da porta OR é, portanto, **A**+0 = **A**. Quando **C**=1, na AND superior a saída é nula, na AND inferior a saída é **B** e a saída da OR é **B**.

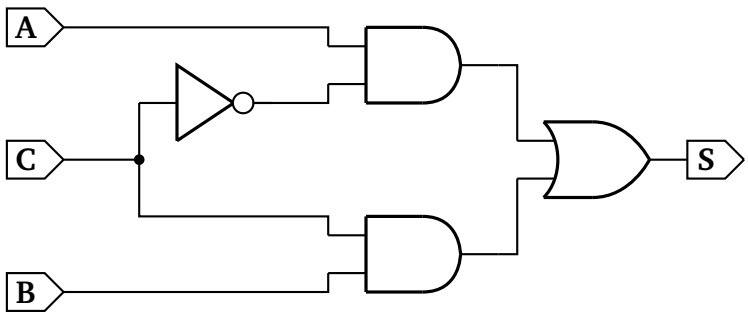


Figura 2.2: Circuito final simplificado após manipulação algébrica da função booleana.

Para expressões booleanas simples, a técnica de simplificação usando álgebra booleana é possível; porém, quando o número de variáveis de entrada e de saída aumenta, tende a ser trabalhosa e passível de erro quando realizada por uma pessoa. Veremos em outros experimentos técnicas passíveis de serem automatizadas usando algoritmos com fins de otimização de custo de circuitos.

2.4 Pré-Relatório - Experimento 2

Para o pré-relatório do Experimento 2, o aluno deve realizar e documentar as atividades das próximas subseções, seguindo as orientações do professor em sala.

2.4.1 Análise de um Circuito Digital

Observe o circuito da Figura 2.3 e responda:

- Quantas entradas e saídas tem o circuito?
- O circuito apresenta uma porta **E** de três entradas. Como é possível obtê-la a partir de portas **E** de duas entradas? Justifique.
- Apresente a tabela-verdade do circuito;
- Explique que este circuito faz, i.e., apresente uma possível função para o mesmo.

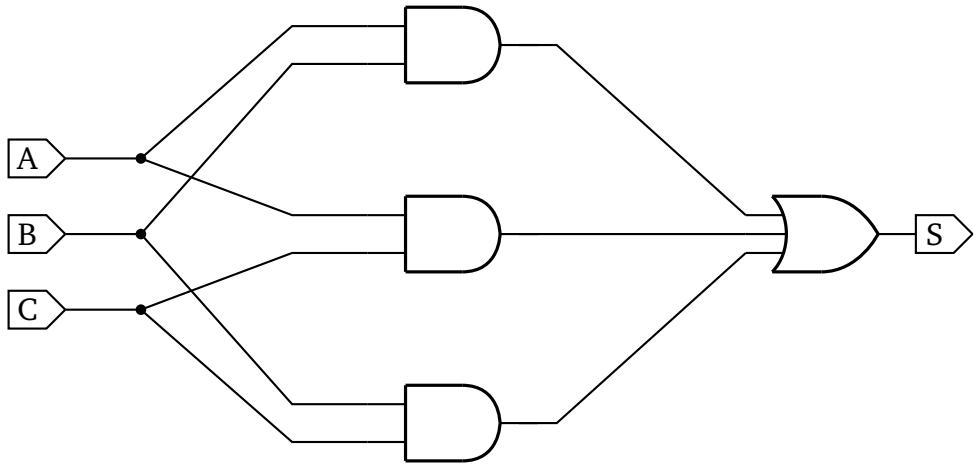


Figura 2.3: Circuito a ser analisado.

2.4.2 Projetos e Simulações

Para os projetos a seguir, apresente:

- O diagrama do circuito, seguindo as diretrizes da Figura 1.1;
- A tabela-verdade do circuito;
- A forma de onda na saída, considerando todas as possibilidades de ocorrência dos sinais de entrada.

Como referência para sua documentação, siga o modelo de resultado apresentado na Figura 1.5.

Projeto 1 . Considere um circuito de alarme de automóvel usado para detectar uma determinada condição indesejada. Sinais provenientes de sensores são usadas para indicar, respectivamente, o estado da porta do motorista (aberto/fechado), o estado da ignição (ligada/desligada) e o estado dos faróis (aceso/apagado). O alarme deve ser ativado em dois casos: (i) sempre que os faróis estão acesos **e** a ignição está desligada **ou** (ii) a porta do veículo está aberta **e** a ignição está ligada. Projete, usando portas lógicas básicas, um circuito que gere um sinal para ativar o alarme deste automóvel.

Projeto 2 . Um circuito detector de paridade realiza a seguinte função: para um conjunto de k bits, apresenta o resultado **1** se a quantidade de bits **1** for ímpar e apresenta o resultado **0** se a quantidade de bits **1** for par. Projete um detector de paridade para um bloco de $k = 4$ bits. Sugestão: use portas **XOR**.

2.4.3 Simulações Computacionais

Simule o circuito da Figura 2.3 e os circuitos dos Projetos 1 e 2. Para cada circuito, apresente as formas de onda das entradas e da saída.

2.4.4 Projeto Usando VHDL

Apresente o código VHDL de todos os circuitos desta Seção. Use o QUCS, ModelSim (Altera), Vivado ou ISE (ambos Xilinx) para gerar as formas de onda correspondentes às entradas e saídas.

EXPERIMENTO 3

CIRCUITOS SOMADORES E SUBTRATORES

3.1 Objetivos

Familiarização com a aritmética binária e com a implementação de circuitos somadores binários.

3.2 Introdução

Uma função essencial da maioria dos computadores e calculadoras é a realização de operações aritméticas. Essas operações são realizadas em uma parte específica do hardware conhecida como Unidade Lógica e Aritmética (ULA). Esta unidade é formada por portas lógicas e flip-flops que combinados permitem a realização de somas, subtrações, multiplicações e divisões de números binários. Esses circuitos realizam essas operações em uma velocidade considerada humanamente impossível. Normalmente, uma operação de adição demora menos que 100 ns [TWM07].

A estrutura básica de uma ULA está mostrada na Figura 3.1. O objetivo básico de uma ULA é receber dados binários armazenados na memória e executar operações aritméticas e lógicas sobre esses dados, de acordo com instruções provenientes da unidade de controle. Assim, uma sequência de operações típica de uma ULA pode ocorrer conforme se segue:

1. A unidade de controle recebe uma instrução determinando que um determinado valor na memória deve ser somado ao valor do acumulador;

2. O valor é transferido da memória para o registrador B;

3. Os valores do acumulador e do registrador B são apresentados à lógica de adição que executa a soma e armazena o resultado no acumulador;

4. O resultado pode ser mantido no acumulador para operações subsequentes ou ser transferido para a memória.

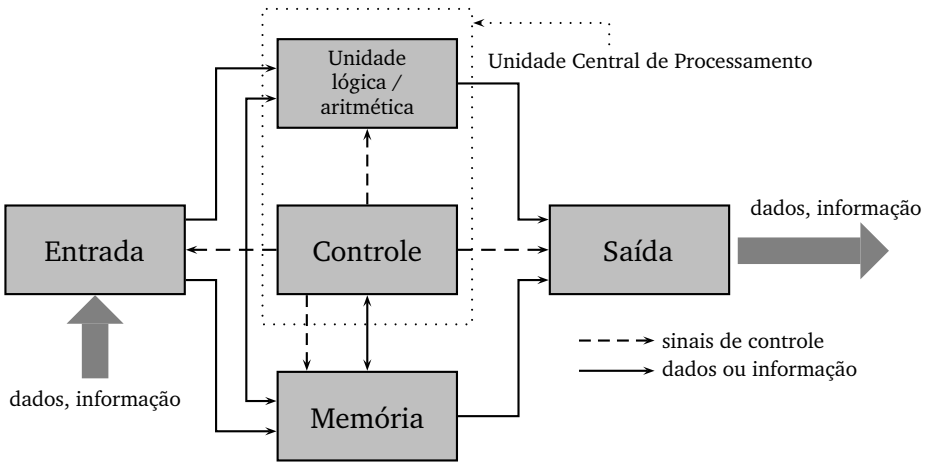


Figura 3.1: Blocos funcionais de uma ULA. Figura adaptada de [TWM07].

A complexidade dos blocos funcionais da ULA mostrados na Figura 3.1 é proporcional à complexidade do sistema em que será utilizada; assim, sistemas simples permitem o uso de ULAs simples e sistemas sofisticados exigem ULAs sofisticadas. Uma vez estabelecido o porte do sistema, existe também o compromisso entre velocidade e preço. Por exemplo, as calculadoras eletrônicas exigem ULAs que permitem operações complexas, porém com velocidade de operação baixa, reduzindo-se o custo; já os computadores de grande porte exigem velocidade de operação elevada, aumentando o custo da ULA.

Neste momento do curso, no entanto, não estamos interessados no estudo detalhado de todos os blocos funcionais que formam uma ULA. Nosso foco, então, será apenas nos circuitos lógicos que realizam as operações aritméticas de soma e subtração.

3.3 Representação de Números Binários

Como a maioria dos computadores e das calculadoras digitais realiza operações tanto com números positivos e negativos, é necessário representar, de alguma forma, o sinal do número ($+$ ou $-$). Existem várias formas de obter essa representação. Uma das maneiras é a representação denominada de sistema **sinal-magnitude**, ou sinal e amplitude. Nesta forma de representação, simplesmente adiciona-se ao número um outro bit denominado de **bit de sinal**. Em geral, a convenção comum é utilizar o bit 0 para número positivos e o bit 1 para números negativos.

Como exemplo, vamos representar os números 27_{10} e -55_{10} na base 2 usando a representação sinal-magnitude:

$$+27_{10} = \underbrace{0}_{+}\ \underbrace{1\,1\,0\,1\,0\,1}_{\text{magnitude}}$$

$$-55_{10} = \underbrace{1}_{-}\ \underbrace{1\,1\,0\,1\,0\,0\,1}_{\text{magnitude}}$$

Notem que, para o primeiro caso, foram necessários 7 bits para representar o número, e no segundo, 8 bits. A quantidade de bits necessária para representar os números em uma sistema digital (por exemplo, a ULA) em geral é fixa. Assim, nos referenciamos a uma ULA de 8 bits, 16 bits, e assim por diante.

Uma outra observação refere-se ao uso do ponto decimal: assim como na base 10, números à direita do ponto decimal representam potências negativas de 2. Assim, em sinal-magnitude o número $0101.11_2 = +5 + 2^{-1} + 2^{-2} = 5.75_{10}$.

Embora o sistema sinal-magnitude seja uma representação direta, os computadores e calculadores normalmente não o utilizam, porque esse sistema requer a implementação de circuitos mais complexos quando comparados a outras alternativas. Um exemplo de complicador é o número 0: ele pode ter duas representações. Por exemplo, usando 1 bit de sinal e 4 de magnitude há duas possibilidades: (0 0000) ou (1 0000).

A maioria dos sistemas modernos usa o **sistema de complemento de 2** para representar números negativos. O complemento de 2 de um número negativo é obtido tomando o complemento de 1 do número (substituição de todos os 0s por 1s e 1s por 0s) e somando 1 na posição do bit menos significativo.

O sistema de complemento de 2 para representação de números com sinal funciona da seguinte forma:

- Se o número for **positivo**, a magnitude é representada por na sua forma direta, e um bit 0 representando o sinal é colocado ao lado do bit mais significativo.
- Se o número for **negativo**, a magnitude é representada na sua forma de complemento de 2, e um bit 1 é colocado ao lado do bit mais significativo.

Um exemplo de representação em complemento de 2: o número -97_{10} = (11100001) em sinal-magnitude é representado em complemento de 2 como (10011111), pois tomando a magnitude (1100001) resulta

$$\text{CP}_2[1100001] = \text{CP}_1[1100001] + 1 = 0011110 + 1 = 0011111.$$

Esse sistema é o mais utilizado para representar números com sinal porque permite realizar a operação de subtração efetuando, na verdade, uma operação de adição. Desta forma, o sistema digital pode usar o mesmo circuito tanto na adição quando na subtração, desse modo poupando hardware. Observe que se realizarmos a operação de complemento de 2 duas vezes, encontramos a magnitude do número binário.

Para visualizar como esse procedimento funciona, basta lembrar que realizar a subtração de $(5 - 4)$ é equivalente a realizar a seguinte operação de adição $(5 + (-4))$. Portanto, para realizar a operação de soma ou subtração que envolva números negativos, basta determinar o complemento de 2 dos números negativos envolvidos e realizar a operação de adição. O procedimento descrito abaixo ajuda na tarefa de realizar operações no sistema de complemento de 2.

1. Represente os números envolvidos em binário puro;
2. Verifique a quantidade de bits da representação, se necessário complete a sequência de bits com zeros à esquerda de acordo com a quantidade de bits definida para a realização da operação;

3. Identifique os números negativos e determine o seu complemento de 2;

4. Realize a soma binária;

5. Verifique o bit de sinal do resultado. Se for 0, o resultado é positivo; se for 1, o resultado é negativo;

6. Em caso de resultado negativo, se quiser visualizar o resultado, recomenda-se representar o número na forma de sinal-magnitude, assim determine o complemento de 2 do resultado para determinar a magnitude do número negativo obtido (lembrando ao final deste processo que trata-se de um número negativo);

Um procedimento manual rápido para encontrar o complemento de 2 de um número binário é partindo de sua representação positiva: (i) inicie copiando os bits da direita para a esquerda, até encontrar o primeiro bit 1; (ii) copie este bit 1; e (iii) inverta todos os outros bits à esquerda, incluindo o de sinal (que estava com 0). Em outras palavras, inverta todos os bits à esquerda do 1 menos significativo. Verifique.

3.3.1 *Overflow* Aritmético

Ocorre sempre que uma operação aritmética produz um número que necessita ser expresso em mais bits de magnitude do que está disponível. Por exemplo, considere um sistema digital que trabalha com números de 4 bits de magnitude e um bit de sinal. Considere que seja necessário realizar a adição de $+9$ (01001) com $+8$ (01000). Neste caso, tem-se como resultado o número (10001), que representaria o decimal -1, enquanto que a resposta deveria ser $+17$, indicando obviamente um erro no cálculo. Isso ocorre porque para representar a magnitude 17 é necessário mais do que os quatro bits disponíveis, portanto ocorre o transbordamento do "vai-um" ou *overflow* ou transbordamento.

A condição de transbordamento pode ocorrer apenas quando dois números positivos ou dois números negativos são somados, e isso sempre produz um resultado errado. Desta forma, o transbordamento pode ser detectado verificando se o bit correspondente à casa de sinal do resultado tem o mesmo valor dos bits de sinal dos números que estão sendo somados.

Como exemplo de uma condição onde há transbordamento, vamos realizar a operação -18_{10} - 22_{10}, usando 6 bits, incluindo o de sinal. Notem que os 5 bits permitem a representação máxima de -31. Já expressando dos números negativos em complemento de 2, vem:

0	1	1	1			"vai-um"
1	0	1	1	1	0	(-18)
1	0	1	0	1	0	(-22)+
0	1	1	0	0	0	(+24)

3.3.2 Circuitos Somadores e Subtratores

Existem diversos circuitos diferentes para implementar a operação aritmética de soma. Faremos aqui uma bastante usada, a do somador completo. Para realizar uma soma de dois números, vamos imaginar a soma ocorrendo da direita para a esquerda na i-ésima posição, com a eventual presença do "vai-um". Para computar o resultado na posição i, precisamos dos bits dos dois números: A_i e B_i; e também do eventual "vem-um", C_i. São, portanto, três entradas. O resultado em si são dois números: o resultado da soma naquela posição S_i e um eventual "vai-um" C_{i+1} a ser propagado para cada à esquerda. O bit mais à direita, chamado de nenos significativo (LSB - *least significant bit*) não tem "vem um". Como já foi observado, no caso de soma de dois números de mesmo sinal, a presença do "vai-um" na soma mais à esquerda (MSB - *most significant bit*) é um indicador de *overflow*.

	1	1				1		"vai-um"
0	0	1	1	0	0	0	1	(25)
0	0	0	1	0	0	0	1	(09)+
0	1	0	0	0	0	1	0	(34)
+	MSB						LSB	

Consideremos então a seguinte tabela:

A_i	B_i	C_i	S_i	C_{i+1}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Tabela 3.1: Tabela-verdade do somador completo.

A saída S_i nada mais é do que um XOR das três entradas (verifique):

$$S_i = A_i \oplus B_i \oplus C_i.$$

O "vai-um" aparece quando pelo menos duas das três entradas tem valor 1. Portanto,

$$C_{i+1} = A_i\, B_i + A_i\, C_i + B_i\, C_i.$$

O circuito pode então ser implementado com portas lógicas na forma mostrada na Figura 3.2. Na mesma figura, encontra-se um diagrama de blocos do circuito apresentando apenas as entradas e saídas.

O circuito apresentado é capaz de somar corretamente apenas um bit. Para realizar uma adição de N bits, basta concatenar N blocos, como mostrado na Figura 3.3 para N=4.

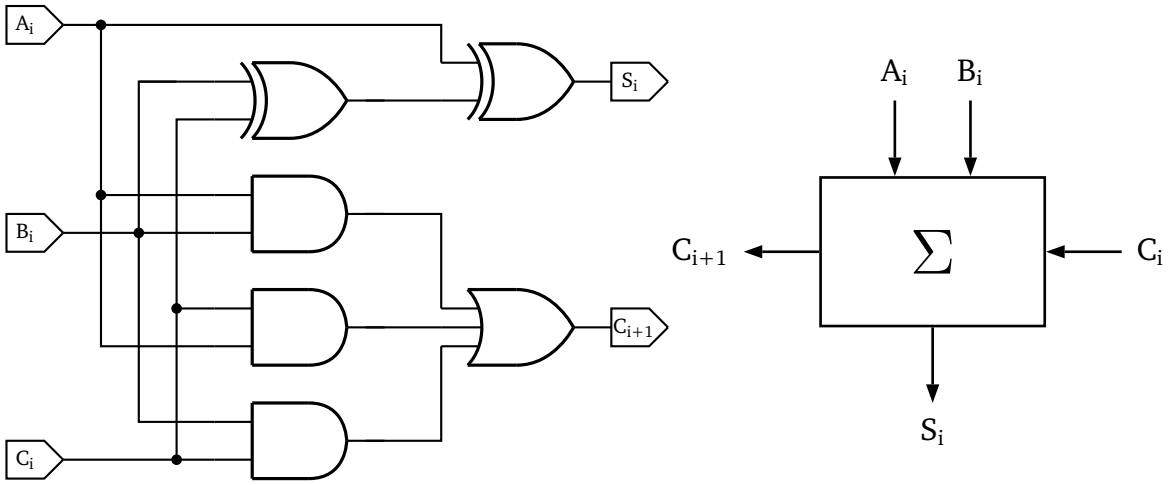


Figura 3.2: Circuito somador completo: implementação com portas lógicas e representação entrada-saída.

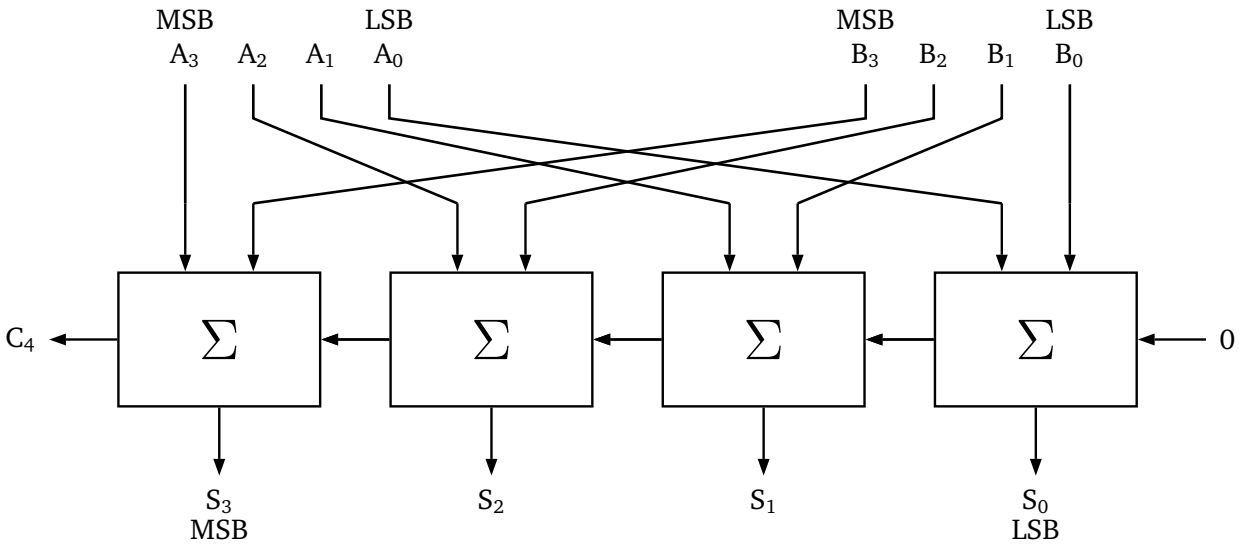


Figura 3.3: Circuito somador para entradas de 4 bits usando somadores completos.

3.4 Pré-Relatório

3.4.1 Pesquisa bibliográfica

Realize uma pesquisa bibliográfica sobre as diferentes configurações de circuitos somadores existentes. Faça uma breve explicação sobre a teoria envolvida em cada um deles, além de uma comparação entre essas diferentes configurações, destacando em cada caso as vantagens e desvantagens. DICA: Pesquisar sobre um circuito denominado **"*carry* antecipado"** (*look-ahead carry*).

3.4.2 Projetos e Simulações

Nesta seção são descritos os circuitos que devem ser projetados e simulados. Na etapa de simulação o aluno pode utilizar o software de sua preferência, desde que forneça as formas de de onda às entradas e saída. Sugerimos utilizar softwares com funcionalidade similar à encontrada na sala SS.

Em seu pré-relatório, devem ser apresentados:

- O nome do software utilizado (fabricante, versão, sistema operacional);
- Os diagramas de blocos e esquemáticos dos circuitos projetados;
- Um plano de validação contendo quais as saídas esperadas para as entradas escolhidas;
- O código VHDL de cada circuito projetado.

Nos projetos os alunos devem apresentar todas as informações adicionais que forem necessárias para perfeita compreensão do projeto realizado.

3.4.2.1 Projeto e Simulação 1

Projetar e simular um circuito que permita realizar o complemento de 1 de um número de 3 bits (incluindo o bit de sinal). Esse circuito deve possuir ainda uma entrada seletora (SEL) que permita especificar quando se deve realizar essa operação de complemento.

Desta forma, quando a operação de complemento não for desejada, o circuito deve fornecer na saída exatamente o número de entrada (ver Figura 3.4). Assim, a função realizada por esse circuito depende do valor da entrada de seleção SEL:

- se SEL = 0, a função selecionada é a IGUALDADE e Z = A;
- se SEL = 1, a função selecionada é o COMPLEMENTO DE 1 e Z = $CP_1[A]$.

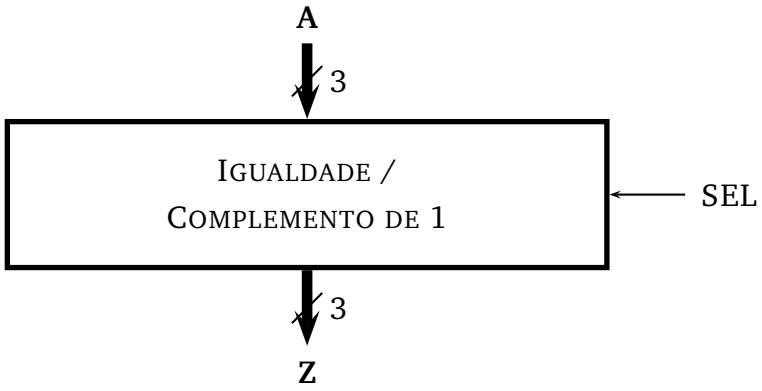


Figura 3.4: Circuito seletor de Igualdade/Complemento de 1.

3.4.2.2 Projeto e Simulação 2

Projetar e simular um circuito que seja capaz de detectar uma condição de *overflow* (estouro de capacidade) para ser usado com um circuito somador de números com sinal, de três bits, codificados na forma de complemento de 2.

3.4.2.3 Projeto e Simulação 3

Projetar e simular um circuito SOMADOR/SUBTRATOR de três bits (incluindo o bit de sinal) para números com sinal, codificados na forma de complemento de 2, dado pelo bloco funcional mostrado na Figura 3.5. O circuito deve ter ainda uma saída (E) que indica as situações em que ocorreu um estouro de capacidade (*overflow*). Desta forma, a função realizada por este circuito, que depende do valor da entrada de seleção SEL, pode ser resumida da seguinte forma:

- se SEL = 0, a função selecionada é a SOMA e S = A + B;
- se SEL = 1, a função selecionada é a SUBTRAÇÃO e S = A + CP_2[B].
- A saída E indica a condição de Overflow.
- A saída C_3 é o *carry* da operação, que pode ser utilizado para expandir a capacidade da operação através da associação com outros blocos somadores.

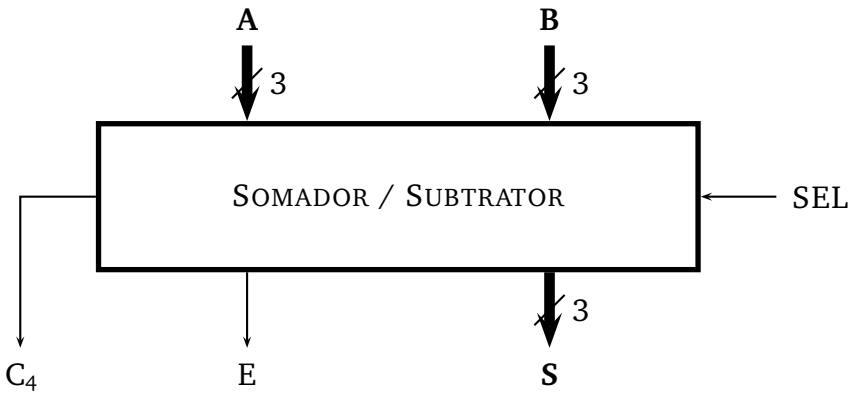


Figura 3.5: Circuito Somador/Subtrator de três bits.

DICAS:

- O bloco lógico funcional desse circuito deve ser, primeiramente, detalhado na forma de um circuito lógico utilizando grandes blocos funcionais. Em seguida, esse circuito lógico deve ser detalhado usando os blocos funcionais disponíveis individualmente na forma de portas lógicas.
- Utilize os circuitos projetados nas Seções 3.4.2.1 e 3.4.2.2 como parte do seu projeto.

- Cuidado na etapa de verificação do circuito. Prepare tabelas contendo os valores de entrada e saída e siga-as, verificando se o circuito fornece a saída correta em todos os casos.

OPCIONAL: O circuito projetado irá fornecer na saída os valores negativos em sua forma de complemento de 2. Altere o seu projeto de forma que na saída os números negativos não estejam nesta forma, mas representados pela notação sinal-amplitude.

3.5 Roteiro Experimental

Implemente em ambiente de simulação os circuitos projetados nas Seções 3.4.2.1, 3.4.2.2 e 3.4.2.3 seguindo as orientações do professor em sala. Em todos os projetos deve-se realizar a seguinte sequência:

1. Apresente os diagramas de blocos de seu sistema completo, em termos de entradas e saídas.
2. Se possível, refine o detalhamento do diagrama para conter sub-blocos funcionais, também representados em termos de entradas e saídas.
3. Repita o processo anterior até chegar ao nível de descrição em termos de portas lógicas.
4. Implemente o código VHDL do circuito todo.
5. Apresente formas de onda que validem o funcionamento de seu projeto.
6. Observe, também, a ocorrência de resultados inválidos. Caso existam, em que condições eles ocorrem?
7. Responda as seguintes perguntas:
    - Quais as diferenças entre o circuito projetado na Seção 3.4.2.3 e uma ULA (Unidade Lógica Arimética) encontrada nos principais processadores?
    - Como o circuito projetado na Seção 3.4.2.3 poderia ser modificado para poder apresentar resultados com 3 bits de magnitude?

3.6 Usando VHDL

A linguagem VHDL permite o tratamento de vários tipos de dados. Nos experimentos anteriores, encontramos essencialmente apenas o tipo `BIT`. Vamos tratar agora de vetores de bits, especificados por `BIT_VECTOR`. Por exemplo, se escrevemos

```
y:  BIT_VECTOR (4 DOWNTO 0);
x:  BIT_VECTOR (0 TO 6);
```

Estamos essencialmente criando um vetor `y` com 5 bits, sendo o MSB o mais à esquerda. O vetor `x` tem 7 bits, com o MSB à direita. Usualmente, usamos a notação posicional com o MSB à esquerda. Para estes casos, podemos escrever

```
y <= "10000"; -- MSB é 1
x <= "1010010"; -- MSB é 0
```

Como exemplo, podemos definir uma porta AND de 8 entradas e 8 saídas assim

```
-------------------------------------------------------
LIBRARY ieee; USE ieee.std_logic_1164.all;
LIBRARY std;  USE std.standard.all;
-------------------------------------------------------
ENTITY minha_and8 IS
   PORT( x, y  : IN  BIT_VECTOR (7 DOWNTO 0);
         z     : OUT BIT_VECTOR (7 DOWNTO 0)  );
END minha_and8;
-------------------------------------------------------
ARCHITECTURE minha_arq OF minha_and8 IS
BEGIN
     z <= x AND y; -- 8 bits nas entradas x,y e saída z
END minha_arq;
-------------------------------------------------------
```

A linguagem também permite usar os tipos `INTEGER`, `NATURAL` e `REAL`, bastante úteis para trabalhar com operações aritméticas. Como exemplo, se `z` for um `INTEGER`, é válido atribuir `z <= 34;`.

Um outro elemento que pode ser útil neste experimento é o conceito de **sinais** em VHDL. A diretiva `SIGNAL` é usada para nomear uma interconexão dentro de um circuito que não esteja diretamente ligada às entradas e saídas definidas definidas pela diretiva `PORT` dentro da seção `ENTITY`.

Ao descrevermos as entradas e saídas dentro da diretiva `PORT`, todos os elementos internos são, por default, sinais. A atribuição de valores para um `SIGNAL` é dada pelo símbolo <=. Assim, ao escrever `z <= a XOR b;` estamos dizendo "o sinal z recebe o valor de a XOR b".

Dentro do código VHDL, a definição de sinais ajuda a estruturar a descrição do circuito. Por exemplo, para o circuito da Figura 3.2 poderíamos escrever para as operações XOR este trecho de código

```
-------------------------------------------------------
...
ARCHITECTURE minha_arq OF meu_somador IS

SIGNAL bxc: BIT;

BEGIN
bxc <= b XOR c; -- esta linha de código é executada
s <= bxc XOR a; -- simultaneamente com esta
...
END minha_arq;
-------------------------------------------------------
```

Observe que a declaração de `bxc` é feita antes de `BEGIN`, e que não faz sentido especificar se o sinal tem modo `IN` ou `OUT`, pois é uma conexão interna do circuito. Note ainda, como assinalado pelos comentários, que as duas linhas de código são executadas **em paralelo**: não se trata de um código sequencial. Exatamente o mesmo resultado seria obtido se invertêssemos a posição das duas linhas.

Ao escrever os códigos VHDL deste experimento, use estes novos conceitos para simplificar a descrição do circuito.

EXPERIMENTO 4

CIRCUITOS CODIFICADORES

4.1 Objetivos

Apresentar uma técnica de minimização de funções booleanas; compreender o funcionamento e familiarizar o aluno com o projeto de circuitos codificadores.

4.2 Simplificação de Funções Booleanas

Ao realizar a síntese de um circuito digital, uma das etapas do processo é a obtenção da função booleana que apresenta a relação entre entradas e saídas. Nem sempre a expressão obtida é a que permite a implementação mais barata do circuito em termos do número de portas lógicas necessárias para realizar a função obtida.

Neste contexto, torna-se relevante considerar técnicas de simplificação de circuitos. Os exemplos apresentados são para exemplos simples, mas a aplicação das técnicas torna-se tão mais relevante quanto mais complexo for o circuito. Atualmente, há circuitos integrados em FPGA com bilhões de transistores, o que em primeira análise dispensaria a necessidade de minimização de circuitos simples. Entretanto, o projetista sempre terá limitação de *hardware* para trabalhar e é com esta tônica que apresentamos as técnicas de minimização: como metodologia de projeto, buscando otimizar a eficiência do projeto face a uma determinada restrição.

Um outro argumento para manter o tema em pauta é olhar a minimização de funções booleanas do ponto de vista de consumo energético. Com o advento de dispositivos eletrônicos portáteis, como *smartphones* e *tablets*, a taxa de consumo de energia da bateria é crucial. E, com menos *hardware*, menos energia é consumida.

4.2.1 Simplificação por manipulação de função booleana

Com o circuito descrito por sua função booleana, esta técnica consiste na aplicação dos teoremas de álgebra booleana para simplificar ao máximo a expressão obtida em

termos da quantidade de operações lógicas necessárias. Um exemplo: simplificar a expressão

$$\mathbf{S} = \overline{\mathbf{A}}\,\overline{\mathbf{B}}\,\overline{\mathbf{C}} + \overline{\mathbf{A}}\,\overline{\mathbf{B}}\mathbf{C} + \mathbf{A}\overline{\mathbf{B}}\mathbf{C} + \mathbf{A}\mathbf{B}\mathbf{C}.$$

Aplicando o fato de que $\mathbf{X} + \overline{\mathbf{X}} = \mathbf{1}$, vem

$$\mathbf{S} = \overline{\mathbf{A}}\,\overline{\mathbf{B}}\,\overline{\mathbf{C}} + \overline{\mathbf{A}}\,\overline{\mathbf{B}}\mathbf{C} + \mathbf{A}\overline{\mathbf{B}}\mathbf{C} + \mathbf{A}\mathbf{B}\mathbf{C} = \overline{\mathbf{A}}\,\overline{\mathbf{B}} + \mathbf{A}\mathbf{C}.$$

Este técnica, se feita manualmente, exige atenção e aplicação dos teoremas de álgebra booleana. Como desvantagem desta técnica, podemos citar:

- Depende da inspiração e experiência do projetista;
- Nem sempre é obvio qual teorema utilizar;
- Não é fácil dizer se uma expressão está na sua forma mais simples ou se ainda pode ser mais simplificada.

4.2.2 Simplificação usando Mapas de Karnaugh

Uma função booleana pode ser expressa em uma forma padronizada, chamada de **forma canônica**. Há duas formulações de formas canônicas: soma de produtos e produto de somas. Tomemos então uma função booleana f de n variáveis expressa por $f(x_1, x_2, \cdots, x_i, \cdots, x_n)$. A mesma função pode ser decomposta na forma

$$f(x_1, x_2, \cdots x_i, \cdots, x_n) = \underbrace{x_1 \cdot f(1, x_2, \cdots, x_i, \cdots x_n)}_{1 \text{ se } x_1=1} + \underbrace{\overline{x_1} \cdot f(0, x_2, \cdots, x_i, \cdots x_n)}_{1 \text{ se } x_1=0}.$$

Vamos tomar um exemplo com duas variáveis x_1 e x_2. Repetindo o processo, vem

$$f(x_1, x_2) = x_1 \cdot f(1, x_2) + \overline{x_1} \cdot f(0, x_2) = \\ x_1 \cdot x_2 \cdot f(1,1) + x_1 \cdot \overline{x_2} \cdot f(1,0) + \overline{x_1} \cdot x_2 \cdot f(0,1) + \overline{x_1} \cdot \overline{x_2} \cdot f(0,0). \quad (4.1)$$

As quatro parcelas acima estão na forma soma de produtos (SP). Notem que basta selecionar quais parcelas têm valor lógico 1 para definir a função f. Por exemplo, a função elementar AND pode ser expressa por $f = x_1 \cdot x_2$. As outras parcelas automaticamente expressam as condições nas quais f tem valor 0. As parcelas de uma SP são chamadas de **mintermos**, e indexados de acordo com a convenção apresentada na Tabela 4.1 para três variáveis. A extensão para n variáveis é imediata.

Uma vez definidos os mintermos de uma função booleana, i.e., as combinações de entrada onde ela vale 1, podemos descrevê-la como uma soma de mintermos. Por exemplo, se

$$f(x_1, x_2, x_3) = \overline{x_1}\,\overline{x_2}\,\overline{x_3} + \overline{x_1}\,x_2\,x_3 + x_1\,x_2\,x_3,$$

podemos reescrevê-la como

$$f = \sum m(0, 3, 7).$$

O Mapa de Karnaugh (ou Mapa-K) usa uma disposição matricial inteligente para apresentar os mintermos de uma função booleana, para em seguida permitir utilizar a regra

MINTERMO	LITERAL	DECIMAL	NOTAÇÃO
$\overline{x_1}\ \overline{x_2}\ \overline{x_3}$	0 0 0	0	m_0
$\overline{x_1}\ \overline{x_2}\ x_3$	0 0 1	1	m_1
$\overline{x_1}\ x_2\ \overline{x_3}$	0 1 0	2	m_2
$\overline{x_1}\ x_2\ x_3$	0 1 1	3	m_3
$x_1\ \overline{x_2}\ \overline{x_3}$	1 0 0	4	m_4
$x_1\ \overline{x_2}\ x_3$	1 0 1	5	m_5
$x_1\ x_2\ \overline{x_3}$	1 1 0	6	m_6
$x_1\ x_2\ x_3$	1 1 1	7	m_7

Tabela 4.1: Convenção para nomear mintermos.

de simplificação $\mathbf{X} + \overline{\mathbf{X}} = \mathbf{1}$. Em um Mapa K, células adjacentes diferem em um bit na representação literal de seus mintermos.

A Figura 4.1 mostra a disposição dos mintermos para uma função de 4 variáveis e um exemplo com f definida por

$$f = \sum m(0, 4, 6, 11, 12, 14, 15).$$

No diagrama à esquerda, também destacamos as regiões onde as variáveis têm valores específicos. Note que, nos extremos $0, 2, 10$ e 8, $x_2 = 0$. Também observe que a disposição apresentada não é única: é possível trabalhar com Mapas K que têm linhas e colunas trocadas quando comparadas à do exemplo.

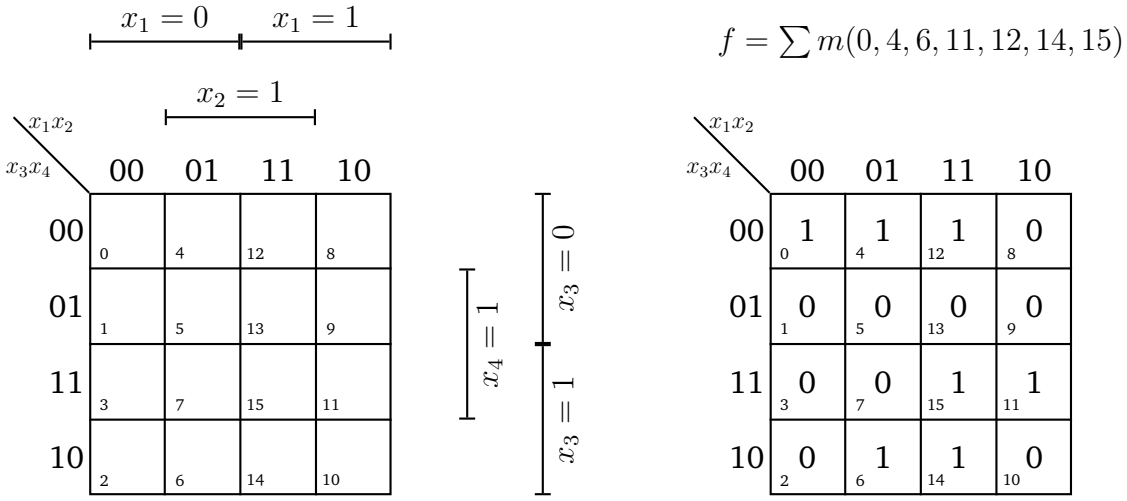


Figura 4.1: Mapa K para 4 variáveis, com exemplo de inserção dos mintermos.

Vamos agora usar o Mapa K para simplificar uma função booleana. Considere a

função

$$f = \sum m(0,4,8,10,11,12,14,15) = \\ \overline{x_1}\,\overline{x_2}\,\overline{x_3}\,\overline{x_4} + \overline{x_1}\,x_2\,\overline{x_3}\,\overline{x_4} + x_1\,\overline{x_2}\,\overline{x_3}\,\overline{x_4} + x_1\,\overline{x_2}\,x_3\,\overline{x_4} + \\ x_1\,\overline{x_2}\,x_3\,x_4 + x_1\,x_2\,\overline{x_3}\,\overline{x_4} + x_1\,x_2\,x_3\,\overline{x_4} + x_1\,x_2\,x_3\,x_4. \quad (4.2)$$

Ao desenhar o Mapa K, mostrado na Figura 4.2, notem que a primeira linha correspondente a $x_3 = 0$ e $x_4 = 0$ ficou preenchida com 1's. Isto significa que, nesta linha, f não depende as entradas x_1 e x_2, pois para qualquer combinação delas o resultado é o mesmo. Assim, para esta região, vale $f = \overline{x_3} \cdot \overline{x_4}$.

Na região delimitada pelos mintermos $10, 11, 14, 15$, $x_3 = 1$ e, simultaneamente, $x_1 = 1$. Não importa o valor de x_2 (pode ser 0 ou 1) e nem o de x_4. Nesta região, vale portanto $f = x_1 \cdot x_3$.

$$f = \sum m(0,4,8,10,11,12,14,15)$$

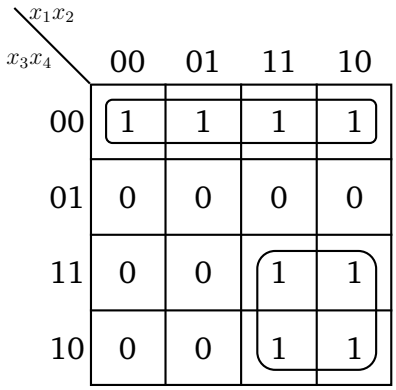


Figura 4.2: Exemplo de uso do Mapa K para simplificação de função booleana.

Estes dois agrupamentos cobrem todas as regiões onde $f = 1$. Podemos então escrever

$$f = \overline{x_3} \cdot \overline{x_4} + x_1 \cdot x_3,$$

bem mais simples do que a expressão original.

Observem que, neste exemplo com 4 variáveis, podemos afirmar que:

- Um único quadrado no Mapa K é definido por quatro valores específicos das variáveis (por construção);
- Um agrupamento de dois 1's (chamados de cubos-2) vizinhos na vertical ou horizontal elimina uma variável, pois quadrados vizinhos diferem de 1 bit - e, portanto, na região de agrupamento f não depende da variável que teve o bit alterado;
- Um agrupamento de 4 bits (cubos-4) elimina duas variáveis;
- Um agrupamento de 8 bits (cubos-8) elimina três variáveis.

Observem que não é qualquer formato de agrupamento de 4 ou 8 bits que permite a eliminação de variáveis. Dois exemplos mostrados na Figura 4.3: notem que as colunas laterais à esquerda e direita, assim como as linas inferior e superior, também são vizinhas. Você também pode imaginar o Mapa K como sendo um toróide para determinar quem são os vizinhos nas bordas.

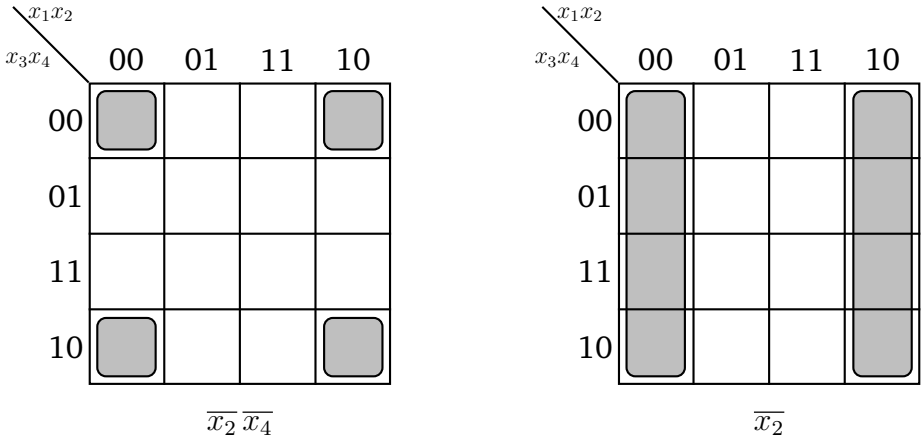


Figura 4.3: Dois agrupamentos em Mapa K de 4 variáveis.

Em resumo, o uso do Mapa k permite a simplificação de uma função booleana ao determinar o menor conjunto de cubos que cobre os mintermos da função. Maiores detalhes serão vistos na teoria.

Vamos agora usar o Mapa K para uma situação comum ao minimizar uma função booleana. Às vezes, a função não precisa ser definida para todos os possíveis mintermos: dizemos que a função booleana não está completamente especificada. Esta situação ocorre quando temos a certeza de que uma determinada entrada nunca vai ocorrer - portanto, o valor da saída é irrelevante[1] para o funcionamento geral do sistema. Quando isto ocorre, não importa se as entradas correspondentes são 0 ou 1.

No Mapa K, assinalamos os mintermos do tipo *don't care* com um $\times$ representando 0 ou 1, de modo a construir os maiores agrupamentos possíveis. A Figura 4.4 mostra um exemplo do uso das condições "dont´t care" para uma função booleana de quatro variáveis.

Observem que a solução mais simples deixa o "dont´t care" 5 no valor 1 (formando um cubo-4 $(0, 1, 5, 4)$ e os restantes como zero.

4.3 Displays

O *display* é um elemento importante no projeto de sistemas eletrônicos, dada a sua função de apresentar informações inteligíveis a um ser humano. A gama de utilização dos *displays* é bastante diversa, indo desde aplicações no setor industrial até a incorporação em produtos eletrônicos de consumo de massa. Um exemplo recente são os *displays*

[1] Em Inglês, *don't care* - não importa.

$$f = \sum m(0,1,4) + \sum d(5,6,12,13,14)$$

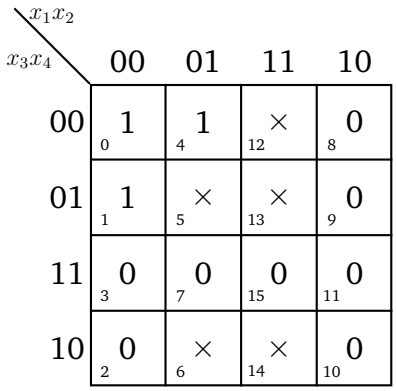


Figura 4.4: Uso do Mapa K com elementos "dont´t care".

sensíveis ao toque usados em produtos portáteis. Assim, encontra-se no mercado uma grande variedade de opções de formatos, especificações e complexidade de *displays*.

Todos os *displays* pedem algum tipo de circuito para controlar seus elementos internos. Por exemplo, para um *display* SXGA (*Super Extended Graphics Array*) de 1280 x 1024 pixels usado em notebooks, torna-se necessária a presença de circuitos destinados a controlar cada um de seus *pixels*, de modo a poder mostrar dinamicamente as imagens que se deseja visualizar.

Neste experimento, será utilizado um *display* LED de 7 segmentos. Este tipo de *display* é usado para visualizar informações numéricas, podendo ser usado em relógios, instrumentos de medição, painéis de preço e calculadoras, dentre outras aplicações. Como os *displays* necessitam de controladores, também será utilizado um conversor de dígitos hexadecimais para 7 segmentos, projetado especificamente para realizar interface com o *display* mencionado.

4.3.1 O Display LED de 7 Segmentos

Como o nome indica, o arranjo deste *display* consiste em uma matriz de LEDs formando sete segmentos, referenciados pelas letras A até G. Além das letras, é comum existir um ponto (DP − decimal point), prevendo aplicações numéricas. Notem que 7 segmentos permitem representar números de 0 a 15 em hexadecimal (verifiquem).

O kit FPGA a ser utilizado futuramente tem 4 *displays*, e cada um deve ser configurado para exibir uma determinada informação. Neste experimento, o projeto consiste em implementar um decodificador que transforme uma informação binária de 4 bits em uma sequência específica de quais segmentos estarão ativos para representar a informação em hexadecimal. A Tabela 4.2 mostra como o decodificador deve funcionar.

As entradas são uma palavra binária de 4 bits e, para cada uma delas, um grupo específico de segmentos deve estar ativo (nível lógico 1) para que a representação correspondente seja mostrada no *display*.

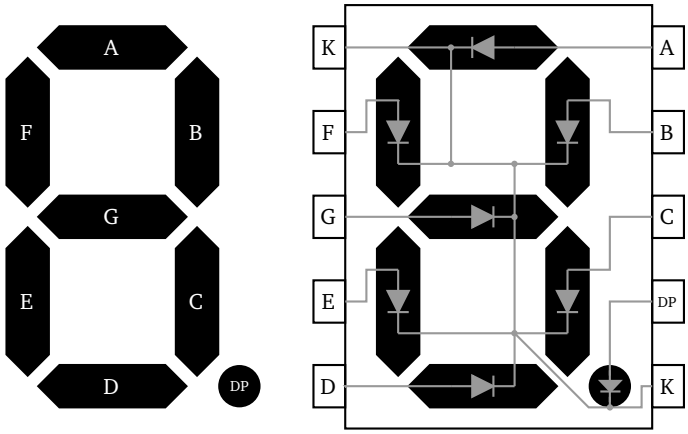


Figura 4.5: Display de 7 segmentos. Esquerda: disposição dos segmentos; direita: configuração catodo comum.

4.4 Pré-Relatório

- Usando minimização com Mapa K, construa a partir da Tabela 4.2 sete funções booleanas (uma para cada segmento) de quatro variáveis. Apresente o esquemático dos circuitos correspondentes, simule no ambiente de sua preferência e apresente as formas de onda correspondentes.

- Repita o projeto, usando VHDL. Use vetores para representar as variáveis de interesse.

Entrada	HEX	A B C D E F G
0 0 0 0	0	1 1 1 1 1 1 0
0 0 0 1	1	0 1 1 0 0 0 0
0 0 1 0	2	1 1 0 1 1 0 1
0 0 1 1	3	1 1 1 1 0 0 1
0 1 0 0	4	0 1 1 0 0 1 1
0 1 0 1	5	1 0 1 1 0 1 1
0 1 1 0	6	1 0 1 1 1 1 1
0 1 1 1	7	1 1 1 0 0 0 0
1 0 0 0	8	1 1 1 1 1 1 1
1 0 0 1	9	1 1 1 1 0 1 1
1 0 1 0	A	1 1 1 0 1 1 1
1 0 1 1	B	0 0 1 1 1 1 1
1 1 0 0	C	1 0 0 1 1 1 0
1 1 0 1	D	0 1 1 1 1 0 1
1 1 1 0	E	1 0 0 1 1 1 1
1 1 1 1	F	1 0 0 0 1 1 1

Tabela 4.2: Segmentos ativos em display de 7 segmentos para entradas de 4 bits.

4.5 Usando VHDL

Uma funcionalidade da linguagem VHDL são os comandos `WITH/SELECT/WHEN` e `WHEN/ELSE`. Um exemplo é a implementação de um MUX 4×1 usando `WHEN/ELSE`:

```
-------------------------------------------------------
LIBRARY ieee;
USE ieee.std_logic_1164.all;
-------------------------------------------------------
ENTITY meu_mux41 IS
    PORT( x0, x1, x2, x3: IN STD_LOGIC;
          seletor: IN STD_LOGIC_VECTOR (1 downto 0);
          s: OUT STD_LOGIC);
END meu_mux41;
-------------------------------------------------------
ARCHITECTURE minha_arq OF meu_mux41 IS
BEGIN
     s <= x0 WHEN seletor="00" ELSE
          x1 WHEN seletor="01" ELSE
          x2 WHEN seletor="10" ELSE
          x3;
END minha_arq;
-------------------------------------------------------
```

O mesmo circuito, usando `WITH/SELECT`. Observe os detalhes do uso de vírgulas após o `ELSE` e a atribuição `OTHERS` para `x3`.

```
-------------------------------------------------------
LIBRARY ieee;
USE ieee.std_logic_1164.all;
-------------------------------------------------------
ENTITY meu_mux41 IS
PORT( x0, x1, x2, x3: IN STD_LOGIC;
seletor: IN STD_LOGIC_VECTOR (1 downto 0);
s: OUT STD_LOGIC);
END meu_mux41;
-------------------------------------------------------
ARCHITECTURE minha_arq OF meu_mux41 IS
BEGIN
    WITH seletor SELECT
    s <=  x0 WHEN "00",
          x1 WHEN "01",
          x2 WHEN "10",
          x3 WHEN OTHERS;
END minha_arq;
-------------------------------------------------------
```

EXPERIMENTO 5

CIRCUITOS MULTIPLEXADORES E DEMULTIPLEXADORES

5.1 Objetivos

Familiarização com os conceitos de multiplexação e demultiplexação, bem como sua utilização para implementação de funções lógicas [IC07, SS07, Uye02, TWM07].

5.2 Circuitos Multiplexadores

5.2.1 Introdução

Multiplexar significa selecionar dados dentre diversas fontes. A Figura 5.1 mostra o esquema funcional generalizado de um multiplexador lógico. Nesse dispositivo, os terminais de seleção determinam o terminal de entrada de dados que terá seu conteúdo transferido para a saída. A operação inversa é denominada demultiplexação.

A Figura 5.2 mostra o esquema funcional de um demultiplexador. Como será mostrado adiante, o demultiplexador lógico é quase equivalente a um decodificador. As operações de multiplexação e demultiplexação são realizadas quando diversas fontes de dados compartilham uma mesma unidade de processamento ou canal de transmissão.

5.2.2 Uso de multiplexadores para implementar funções lógicas

A implementação de funções lógicas pode exigir circuitos combinacionais complexos. Ao projetar esses circuitos, o projetista pode utilizar alguns componentes padronizados que normalmente proporcionam uma significativa redução do número de componentes do projeto. Esses componentes são classificados de acordo com os seus graus de integração (MSI - *Medium Scale Integration* e LSI - *Large Scale Integration*). Ao escolher o

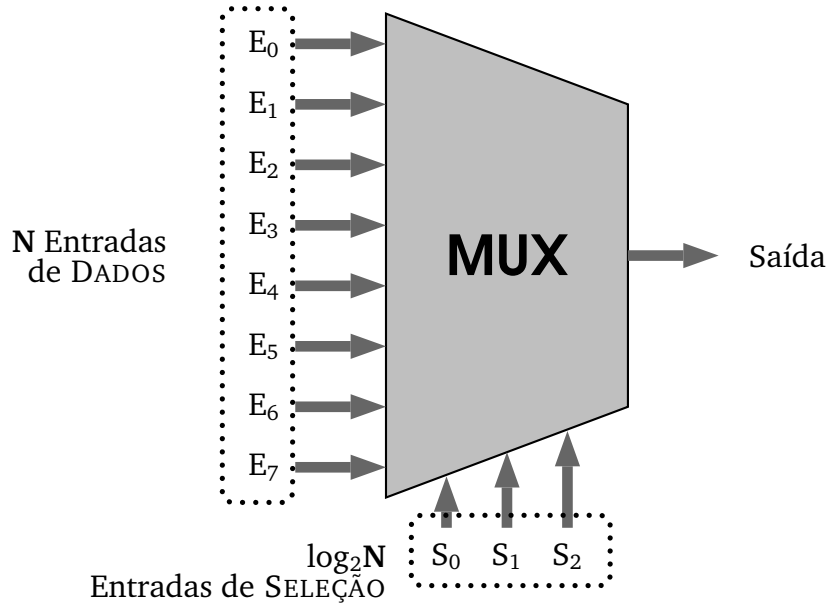


Figura 5.1: Representação de um multiplexador 8/1.

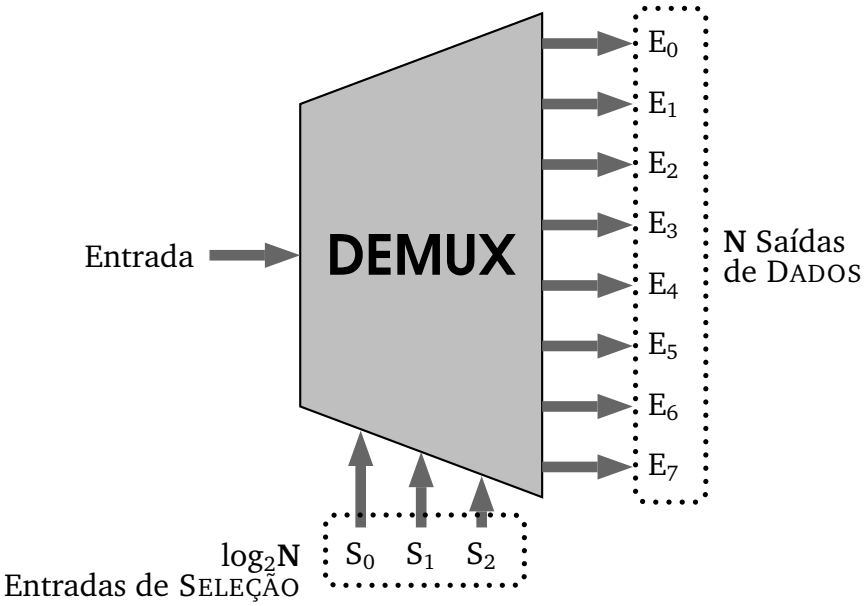


Figura 5.2: Representação de um demultiplexador 1/8.

projeto final, o projetista deve levar em conta, também, que um circuito composto por esses componentes de integração são, geralmente, mais caros que um circuito composto apenas por portas lógicas [SM06].

Exemplos desses componentes que facilitam a implementação de circuitos complexos são os multiplexadores e de demultiplexadores. De fato, em comparação com portas lógicas, o uso de multiplexadores e demultiplexadores resulta em projetos mais simples, compactos e flexíveis.

Para entender como funciona o processo de utilização dos multiplexadores para implementar uma função lógica, considere o seguinte exemplo:

Exemplo 1: Suponha que estejamos interessados em projetar o circuito representado pela tabela verdade mostrada na Figura 5.3. Neste caso, o multiplexador pode ser usado para gerar a função lógica diretamente da tabela verdade sem simplificação.

Observe o esquemático da Figura 5.3. As variáveis lógicas de entrada do *projeto* (A, B e C) são conectadas às entradas de *seleção do multiplexador* (S_0, S_1 e S_2). Cada entrada de *dados do multiplexador* (I_0, ..., I_7) é colocada em nível lógico ALTO ou BAIXO de acordo com a função booleana desejada para a saída ($Z = A\bar{B}\bar{C} + \bar{A}B\bar{C} + ABC$; note que essa função booleana produz a tabela de verdade desejada). Finalmente, a saída do circuito é a saída do multiplexador (Z).

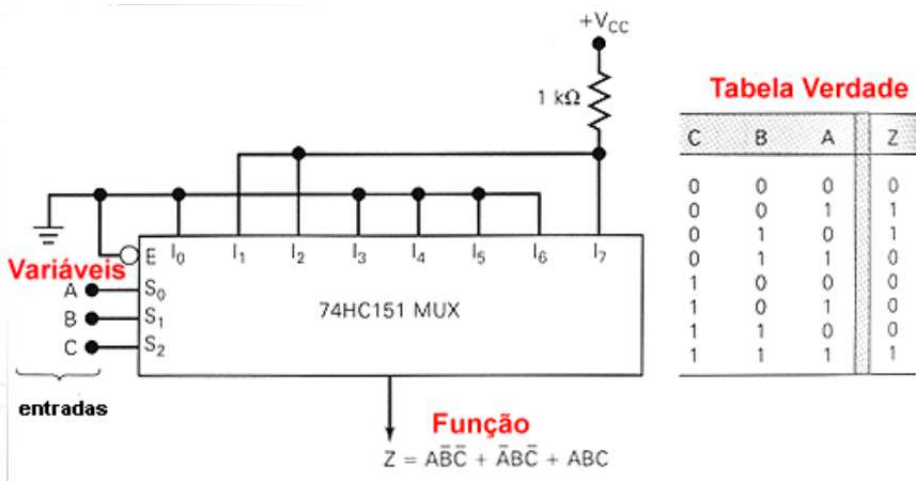


Figura 5.3: Tabela verdade e esquemático do exemplo1.

Exemplo 2: No exemplo 1, o problema analisado consistia de 3 variáveis de entrada e foi utilizado um Mux 8×1 – Mux $N \times M$ significa um multiplexador de N entradas, cada uma composta de M bits. O projeto foi relativamente simples, mas há situações um pouco mais complexas.

Suponha que um problema seja composto por mais variáveis de entrada do que a capacidade dos multiplexadores que um projetista tenha disponível. Por exemplo, suponha que queiramos implementar a função descrita na tabela verdade da Figura 5.4, mas que dispomos de multiplexadores de 8 canais. Para implementar essa função com um multiplexador apenas, precisaríamos de um multiplexador de 16 canais, visto que temos 4 variáveis de entrada.

Uma solução para esse problema é a seguinte: escolhemos três das quatro variáveis de entrada do projeto (A, B, C, D) para acionar os terminais de seleção (E_1, E_2 e E_3); vamos escolher A, B e C (cf. Figura 5.5).

5.2.2.1 Técnica geral

Com essa escolha podemos simplificar nova tabela de verdade inicial. Observe a Figura 5.4; versão simplificada tabela de verdade definida em função da entrada D de acordo com os agrupamentos tracejados (cada dupla de linhas foi agrupada em

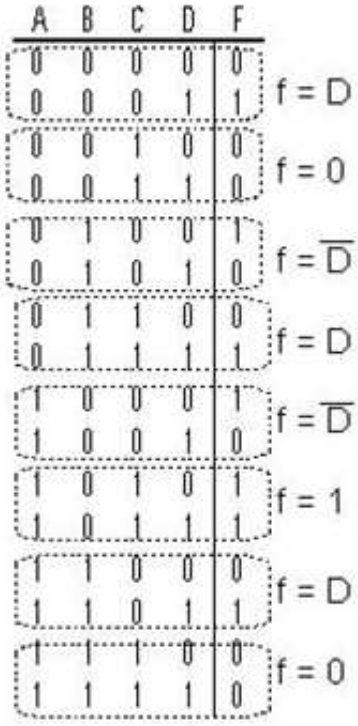


Figura 5.4: Tabela verdade do exemplo2.

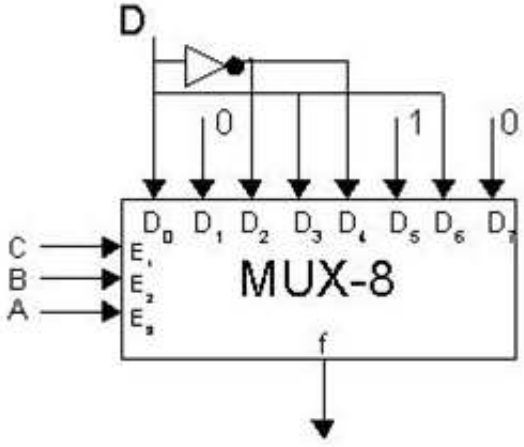


Figura 5.5: Implementação da função f com um MUX-8.

uma nova linha e colocada em função da entrada D). A tabela inicial tinha 16 saídas diferentes; nova tem 8.

Portanto, agora, conseguimos implementar o projeto com um multiplexador de 8 entradas. Para isso, basta conectar à *cada entrada de dados do multiplexador* a função descrita na nova versão da tabela de verdade, a saber: $D_0 = D$, $D_1 = 0$, $D_2 = \bar{D}$, $D_3 = D$, $D_4 = \bar{D}$, $D_5 = 1$, $D_6 = D$, $D_7 = 0$ (cf. Figuras 5.4 e 5.5). Finalmente, a saída do circuito é a saída do multiplexador. A Figura 5.5 apresenta o esquemático final desse

circuito.

****

A técnica de implementação introduzida no Exemplo 2 pode ser generalizada para uma função de n variáveis. Considere um multiplexador de 8 terminais de dados. A expressão booleana da saída S é

$$S = \bar{E}_1\bar{E}_2\bar{E}_3D_0 + E_1\bar{E}_2\bar{E}_3D_1 + \ldots + E_1E_2E3D_7. \tag{5.1}$$

Qualquer função de $n > 3$ variáveis pode ser colocada na forma seguinte:

$$f(A, B, C, D, E, \ldots) = \bar{A}\bar{B}\bar{C}F_0(D, E, \ldots) + A\bar{B}\bar{C}F_1(D, E, \ldots) + \ldots + ABCF_7(D, E, \ldots), \tag{5.2}$$

em que A, B e C são variáveis selecionadas *arbitrariamente* dentre as n disponíveis. Os termos F_0, F_1, ..., F_7 são funções das $(n - 3)$ variáveis restantes; portanto são funções mais simples (de menos variáveis) que a função original f.

No caso particular em que $n = 4$ (mesmo caso do Exemplo 2), as funções F_0, F_1, ..., F_7 serão funções da única variável restante. Nesse caso, haverá apenas 4 possibilidades (1, 0, D, $\bar{D}$ no Exemplo 2). Também se $n = 3$, a função f estará na própria forma canônica de mintermos e, portanto, os fatores F_0, F_1, ..., F_7 só podem ser identicamente iguais a 0 ou identicamente iguais a 1.

Naturalmente, essa técnica pode ser estendida a outros multiplexadores. Um multiplexador de 16 entradas de dados, por exemplo, pode implementar qualquer uma das 2^{32} funções diferentes de 5 variáveis; para isso, bastará uma única porta inversora adicional.

5.3 Pré-relatório

Nesta seção, são descritos os circuitos que devem ser projetados e simulados. Na etapa de simulação o aluno pode utilizar o software de sua preferência, desde que forneça as formas de onda às entradas e saída. Sugerimos utilizar softwares com funcionalidade similar à encontrada na sala SS.

Em seu pré-relatório, devem ser apresentados:

- O nome do software utilizado (fabricante, versão, sistema operacional);
- Os diagramas de blocos e esquemáticos dos circuitos projetados;
- Um plano de validação contendo quais as saídas esperadas para as entradas escolhidas;
- O código VHDL de cada circuito projetado.

Nos projetos, os alunos devem apresentar todas as informações adicionais que forem necessárias para perfeita compreensão do projeto realizado.

5.3.1 Projeto e Simulação 1

Projete e simule um circuito que implemente a função dada abaixo usando um multiplexador de 4 entradas de dados e um decodificador de 4 entradas.

$$f(A,B,C,D) = \bar{A}\bar{B}\bar{C}\bar{D} + A\bar{B}\bar{C}D + \bar{A}\bar{B}C\bar{D} + \bar{A}BCD \tag{5.3}$$

Orientação: O demultiplexador também é um circuito útil na implementação de funções complicadas, pois cada uma de suas saídas constitui um dos mintermos das variáveis de entrada. Observe, por exemplo, que qualquer função de 4 variáveis pode ser implementada com um DECOD-4 e mais algumas portas OU. Portanto, use o decodificador como um gerador de mintermos para o multiplexador.

5.3.2 Projeto e Simulação 2

A economia do consumo de energia e de componentes é sempre um fator importante a ser considerado em projetos de sistemas digitais. Particularmente, em projetos que necessitem de acionamento de vários *displays*, é comum a utilização de uma técnica conhecida como *multiplexação de displays* [SM06].

Essa técnica permite que apenas *um* decodificador de *displays* possa controlar uma *série* de *displays*. Para isso, é explorado o fato de que os *displays* de LED são capazes de operar em tempos da ordem de nanosegundos. Isto significa que eles podem operar num baixo duty cycle (ou fator de forma) com uma alta taxa de amostragem; isto é, o sistema pode ativar os *displays* ciclicamente (acendendo e apagando), aproveitando a característica do olho humano de detectar apenas os "picos"de brilho [Hig].

Desta forma, no projeto do circuito, os *displays* são ciclicamente acesos e apagados numa frequência conveniente de tal forma que, para o olho humano, todos os *displays* permanecem durante todo o tempo acesos. Existem diversas estratégias para realizar essa tarefa, a escolha da melhor depende das particularidades cada problema; um exemplo está apresentado na Figura 5.6.

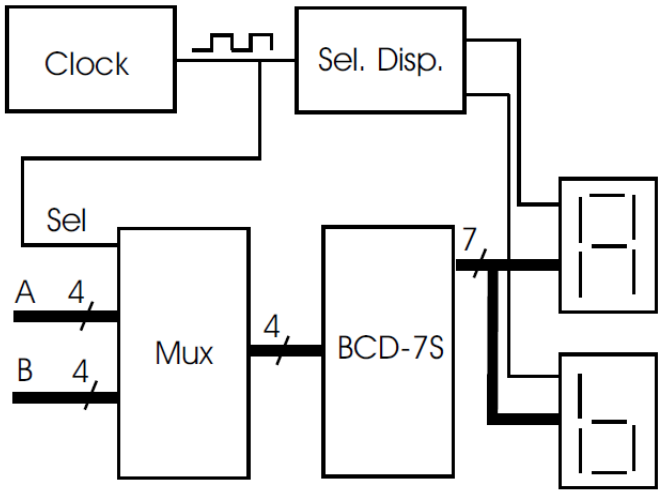


Figura 5.6: Diagrama de blocos do projeto. Figura extraída de [Hig].

Para o pré-relatório o aluno deve implementar um circuito que tem o objetivo de representar um número decimal de dois dígitos (N) em dois *displays* de sete segmentos; o dígito de unidade em um *display* e o dígito de dezena em outro. Portanto, o circuito deverá ter *duas* entradas (A e B); cada entrada será um dígito BCD (um número decimal de 0 a 9). A entrada A será o dígito da dezena de N e B o da unidade (portanto $N = AB$; por exemplo, para $N = 32$, $A = 3$ e $B = 2$).

Esse circuito deverá ter *um*, e apenas um, multiplexador com *um* conversor BCD-7 segmentos para acionar os *dois displays* de sete segmentos alternadamente. Tenha em mente que, conforme exposto acima, os *displays* não precisam estar ativos todo o tempo; eles podem ser ligados e desligados com uma certa frequência (acima de 100 Hz), de modo que o olho humano não consiga notar esse chaveamento [Hig].

Um sinal de onda quadrada deverá escolher quando cada dígito é mostrado em seu respectivo *display* da seguinte forma: quando a onda quadrada está em nível lógico 1, então

- o *display* de A é ligado,
- A é representado nesse *display*, e
- o *display* de B é desligado;

quando a onda quadrada está em nível lógico 0, então

- o *display* de B é ligado,
- B é representado nesse *display*, e
- o *display* de A é desligado.

Note que a onda quadrada deve agir sobre o multiplexador *e* sobre os *displays* de maneira simultânea e sincronizada [Hig].

Enfim, faça o seguinte:

- Projete o circuito que implemente essa função a partir do diagrama de blocos da Figura 5.6.
- Simule o circuito projetado por meio de esquemáticos, sem utilizar programação em VHDL. Para isso, faça com que a onda quadrada seja proveniente de um gerador de funções.
- Simule o circuito projetado utilizando programação em VHDL. Para isso, Recomendamos i) usar vetores de variáveis binárias (vide Seção 4.5), ii) usar as funções WHEN/ELSE ou WITH/SELECT/WHEN (vide Seção 4.5).

Depois, apresente o seguinte no pré-relatório:

- as funções booleanas *minimizadas* de cada segmento; são 14 segmentos (7 de cada *display*), portanto serão 14 funções. Recomendamos construir as tabelas de verdade e utilizar mapas de Karnaugh para obter essas funções.
- o esquemático do circuito projetado.

- os códigos em VHDL utilizados.

EXPERIMENTO 6

INTRODUÇÃO AO PROJETO EM FPGA

6.1 Objetivos

Compreender os conceitos e se familiarizar com as etapas de projeto de circuitos digitais em FPGA utilizando programação VHDL.

6.2 Projeto em FPGA

6.2.1 Introdução

Um FPGA (*Field Programmable Gate Array* ou, em português, Arranjo de Portas Programáveis por Campo) é um tipo de dispositivo de lógica programável, ou seja, um *hardware* que pode ser configurado através de programação para realizar funções lógicas especificadas pelo usuário. A principal vantagem da lógica programável é a possibilidade de alterar projetos com facilidade sem alterações físicas no *hardware* ou substituição de componentes, tornando-a uma excelente ferramenta para a prototipagem e teste de circuitos digitais.

6.2.2 Estrutura do FPGA

O FPGA é um chip que suporta a implementação de circuitos lógicos relativamente grandes. Consiste de um grande arranjo de células lógicas ou blocos lógicos configuráveis contidos em um único circuito integrado.

Os três elementos básicos do FPGA são o bloco lógico configurável, as interconexões programáveis e os blocos de entrada e saída (I/O – in/out) como ilustrado no Fig. 6.1. Os blocos lógicos formam uma matriz bidimensional, e as interconexões programáveis são organizadas como canais de roteamento horizontal e vertical entre as linhas e colunas dos blocos lógicos. Os canais de roteamento possuem chaves de interliga-

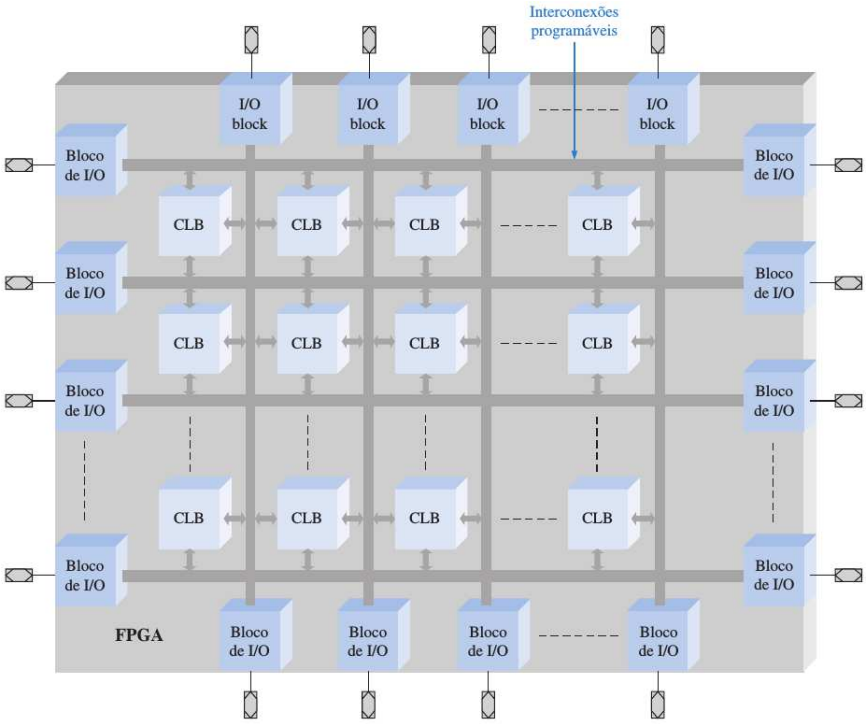


Figura 6.1: Estrutura básica de um FPGA. O bloco lógico configurável é o CLB.

ção programáveis que permitem conectar os blocos lógicos de maneira conveniente, em função das necessidades de cada projeto.

Os blocos de I/O estão nas bordas da estrutura e proporcionam acesso individualmente selecionável de entrada, saída ou bidirecional ao mundo externo. A matriz de interconexões programáveis distribuídas provê as interconexões entre os blocos lógicos e as conexões para as entradas e saídas. FPGAs de grande capacidade podem ter dezenas de centenas de blocos lógicos, além de memória e outros recursos.

6.2.3 Etapas de projeto em FPGA

Conforme discutido, um FPGA pode ser visto como um "quadro branco" no qual podemos implementar um determinado circuito ou sistema projetado fazendo uso de um certo processo de desenvolvimento. Esse processo necessita de um *software* instalado em um computador para implementar um projeto de circuito em um chip programável. O computador tem que ser interfaceado por meio de uma placa de desenvolvimento contendo o chip programável.

Vários passos, chamados de fluxo de projeto, estão envolvidos no processo de implementação de um projeto lógico em um FPGA:

Entrada do Projeto

Esse é o primeiro passo para implementação de um circuito em FPGA. O projeto do circuito ou sistema tem que ser inserido no *software* de desenvolvimento através de entrada textual, gráfica (desenho esquemático) ou descrição em diagrama de estado.

A inserção textual é realizada com uma linguagem de descrição de *hardware* (HDL - *hardware description language*), tal como VHDL, Verilog, AHDL ou ABEL. A inserção gráfica (esquemático) permite que funções lógicas pré-armazenadas sejam selecionadas a partir de um biblioteca, apresentadas na tela e então interconectadas para criar o projeto lógico. A inserção por diagrama de estados requer a especificação dos estados pelos quais um circuito lógico seqüencial passa e as condições que provocam a mudança de cada estado.

Uma vez que o projeto é inserido no *software*, ele é compilado. Um compilador é um programa que traduz o código-fonte em código-objeto num formato que pode ser testado logicamente.

Simulação Funcional

O projeto compilado é simulado por *software* para confirmar se os circuitos lógicos funcionam conforme esperado. A simulação irá verificar se as saídas adequadas são geradas para um conjunto de entradas especificadas. Qualquer falha demonstrada pela simulação deve ser corrigida voltando-se à etapa de inserção do projeto e realizando as alterações apropriadas.

Síntese

A fase de síntese ocorre quando o projeto é traduzido em uma lista (*netlist*), a qual tem uma forma padronizada e é independente do dispositivo. A *netlist* é uma rede de portas, registradores e conexões que implementa as funções descritas pelo HDL.

Implementação

A implementação é onde a estrutura lógica descrita pela *netlist* é mapeada na estrutura real do dispositivo a ser programado. O processo de implementação é denominado *fitting* ou *place and route* e resulta em uma saída denominada sequência de bits (*bitstream*), a qual depende do dispositivo usado.

Simulação de Temporização

Esse passo ocorre após o projeto ser mapeado no dispositivo especificado. A simulação de temporização é basicamente usada para confirmar que não existem falhas no projeto ou problemas de temporização em função dos atrasos de propagação.

Download

Uma vez gerada uma seqüência de bits para um dispositivo programável específico, ele tem que ser transferido (operação de *download*) ao dispositivo para implementar o projeto no *hardware*. Alguns dispositivos são voláteis, o que significa que eles perdem o *bitstream* armazenado quando sofrem uma operação de *reset* ou quando a alimentação é desligada. Nesse caso, o *bitstream* tem que ser armazenado numa memória e recarregado no dispositivo após cada *reset*. Além disso, o conteúdo de um dispositivo ISP pode ser manipulado ou atualizado enquanto estiver operando no sistema.

6.3 Placa Digilent Basys3 Xilinx Artix-7

No Laboratório SS, estão disponíveis para uso na disciplina de Prática de Eletrônica Digital 1 um conjunto de placas da Digilent modelo Basys 3 (Fig. 6.2), que é uma plataforma de desenvolvimento de circuito completa para o chip FPGA Artix-7 da Xilinx. Estas placas serão usadas no restante do curso para programação do FPGA, em conjunto com o *software* de desenvolvimento Vivado Design Suite.

A Basys 3 uma coleção de portas e periféricos incluindo 16 chaves, 4 displays de 7 segmentos, 16 LEDs, 5 *pushbuttons*, saída VGA de 12 bits, flash serial, além de portas USB-JTAG para programação e comunicação da plataforma e etc.

A placa trabalha com o *software* Vivado Design Suite que inclui muitas ferramentas para facilitar o projeto do circuito a ser implementado no FPGA.

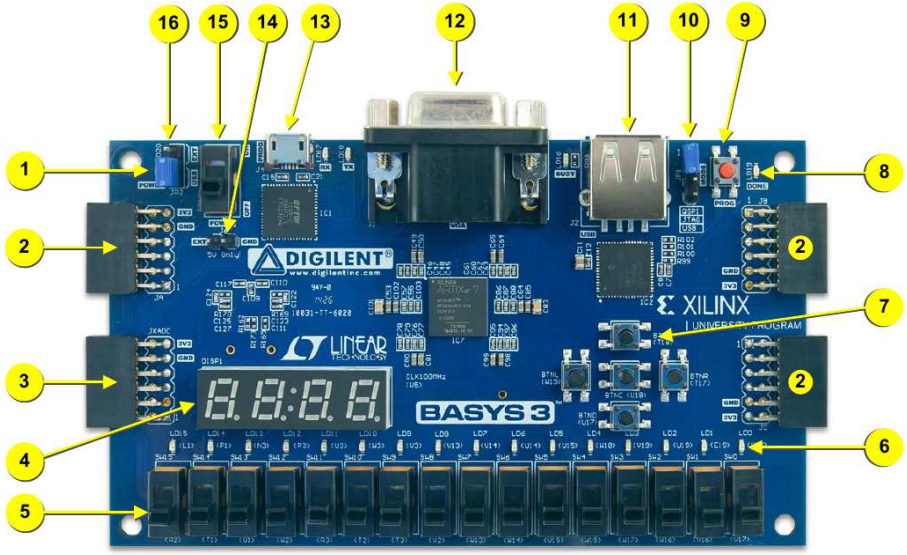


Figura 6.2: Placa Basys 3 da Digilent.

	Descrição do componente		*Descrição do componente*
1	LED da alimentação	**9**	Botão de reset da FPGA
2	Conectores Pmod	**10**	Jumper de modo de programação
3	Conector Pmod sinal analógico (XADC)	**11**	Conector host USB
4	Displays 7-segmentos (4)	**12**	Conector VGA
5	Chaves (16)	**13**	Porta partilhada UART/JTAG
6	LEDs (16)	**14**	Conector para alimentação externa
7	*Pushbuttons* (5)	**15**	Chave de alimentação
8	LED de programação completa	**16**	Jumper de seleção de alimentação

6.4 Pré-Relatório

6.4.1 Pesquisa bibliográfica

1) Leia o manual da placa Basys 3 (*Basys 3 FPGA Board Reference Manual*) e determine:

a) As formas de alimentação suportadas pela placa, os níveis de tensão e corrente envolvidos e como esta configuração é feita;

b) As formas de programação suportadas pela placa e como elas podem ser configuradas;

c) As formas de armazenamento do bitstream na placa (tipos de memória);

d) A quantidade e frequência dos clocks disponíveis;

e) A identificação (nome) dos pinos usados para as entradas da placa (chaves e *pushbuttons*) e para as saídas (LEDs e displays de 7-segmentos);

f) O tipo de display de 7-segmentos (anodo comum ou catodo comum) e os níveis lógicos dos sinais que devem ser aplicados nos pinos de anodo e catodo para que os segmentos se acendam (observe que esta placa usa transistores no pino comum).

6.4.2 Projetos e Simulações

Para a etapa de projetos, será necessário instalar o *software* do Vivado Design Suite (Vivado HL WebPACK Edition - versão 2016.2).

Você pode encontrar pacotes de instalação e instruções em: `https://www.xilinx.com/support/download.html`

6.4.2.1 Projeto e Simulação 1

Projete um sistema em que 7 chaves da placa ($SW0$ à $SW7$) serão usadas para ativar 7 LEDs ($LD0$ à $LD1$), tal que:

$LD0 = \overline{SW0}$
$LD1 = SW1 \,.\, \overline{SW2}$
$LD2 = LD1 + LD3$
$LD3 = SW2 \,.\, SW3$
$LD4 = SW4$
$LD5 = SW5$
$LD6 = SW6$
$LD7 = SW7$

Crie um projeto para implementar este sistema na placa Basys 3 utilizando o *software* Vivado. Você deverá realizar todas as etapas de projeto, deixando apenas a fase final de *download* para ser realizada em sala durante a parte prática deste roteiro.

Para auxiliá-lo na familiarização com o ambiente de desenvolvimento do Vivado, bem como as etapas para criação e configuração do projeto, use como referência o tutorial disponibilizado (*Vivado Tutorial*).

6.4.2.2 Projeto e Simulação 2

Adapte o código VHDL para multiplexação de dois display de 7-segmentos desenvolvido no Experimento 5. Assim como no experimento anterior, o sistema projetado terá duas entradas (A e B), cada uma representando um dígito BCD para a dezena e para

a unidade, respectivamente. Uma terceira entrada, servirá como chave seletora para alternar entre os displays.

Com a chave em 0:

- o display das unidades é ligado,
- o valor de B é representado no display, e
- o display das dezenas é desligado.

Com a chave em 1:

- o display das dezenas é ligado,
- o valor de A é representado no display, e
- o display das unidades é desligado.

Produza um projeto completo no software Vivado para implementar o sistema especificado acima e realize todas as etapas de desenvolvimento deixando apenas a fase final de *download* para ser realizada em sala de aula.

Dica: algumas funcionalidades de VHDL serão bastante úteis para realizar este projeto. Uma delas é o uso de componentes para reaproveitamento de código e interconexão de blocos.

Por exemplo, o código VHDL a seguir representa o circuito ilustrado na Fig. 6.3:

```
entity my_circuit is
   port (x,y:  in std_vetor;
         z,w:  out std_vetor);
end latch;
architecture structure of my_circuit is
   component my_component
      port (a,b:  in std_vetor;
            c:  out std_vetor);
   end component;
begin
   n1:  nor_gate port map (y,w,z);
   n2:  nor_gate port map (x,z,w);
end structure;
```

Algorithm 1: Exemplo do uso de *component* no VHDL.

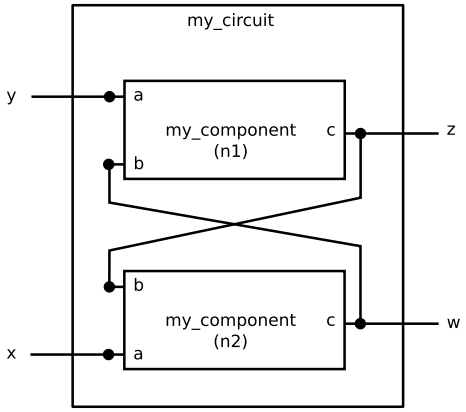


Figura 6.3: Circuito resultante do código apresentado no Algoritmo 1.

Outro conceito novo de VHDL que será explorado em sala durante este experimento é o uso de processos.

Até o momento, estivemos codificando os circuitos em VHDL de forma estrutural, em que componentes são interligados entre si e operam de forma concorrente (ao mesmo tempo). Para isso, utilizamos sentenças concorrentes para atribuição de um sinal através de equações booleanas, *with-select-when* ou *when-else*.

Um processo também se interliga com outros processos e elementos do circuito de forma concorrente. Entretanto, internamente, as sentenças de um processo são executadas de forma sequencial. Pesquise sobre o uso de *process* no VHDL e como ele pode ser usado para gerar um sinal de *clock* em seu projeto.

EXPERIMENTO 7

CIRCUITOS CONTADORES SÍNCRONOS E ASSÍNCRONOS

7.1 Objetivos

Familiarização com projeto e montagem de circuitos contadores síncronos e assíncronos.

7.2 Circuitos Contadores

7.2.1 Introdução

Um contador eletrônico é provavelmente um dos mais úteis e versáteis subsistemas num sistema digital. Graças às diversas versões disponíveis podem ser utilizados, por exemplo, para contagens diversas, divisão de frequência, medição de intervalo de tempo e frequência, geração de formas de onda, e, até mesmo, para converter informações analógicas em digitais. Um contador digital é um circuito sequencial, configurado de tal modo que para cada estado presente nas saídas dos *flip-flops*, existe um estado seguinte bem definido. Durante a operação de contagem, o contador desloca-se de um estado para o outro de acordo com uma sequência especificada.

É possível identificar uma característica que classifica os contadores, de forma ampla, em duas categorias: síncronos e assíncronos. Há, entretanto, vários outros aspectos a serem considerados. Assim sendo, dentro de cada uma das duas categorias, é ainda possível classificar os contadores em função do número de estados (módulo), do número de saídas (*bits*), do tipo de sequência gerada (binária, decimal, código de Gray, *etc*.) ou do tipo de operação: fixa ou selecionável.

7.2.2 Contador Assíncrono

A Figura 7.1 mostra o circuito de um contador binário de quatro *bits*. A seguir é descrito, resumidamente, o funcionamento deste circuito [TWM07]:

- Os pulsos de *clock* são aplicados apenas na entrada CLK do FF. Assim, o FF A comutará cada vez que ocorrer uma borda de descida no pulso de *clock*. Observe que $J = K = 1$ para todos os FFs.

- A saída normal do FF A funciona como clock de entrada para o FF B, sendo que este FF comuta a cada vez que a saída A muda de 1 para 0. Da mesma forma, o FF C comuta quando B muda de 1 para 0, e o FF D muda de 1 para 0.

- As saídas do FFs D, C, B e A representam um número binário de quatro *bits*, sendo D o MSB. Considerando que todos os FFs tenham sido "resetados" para o estado 0, as formas de onda mostradas na Figura 7.5 mostram que uma contagem binária sequencial de 0000 a 1111 é seguida à medida que os pulsos de *clock* são aplicados continuamente.

- Após a borda de descida do décimo quinto pulso de *clock*, os FFs do contador estão na condição 1111. Na décima sexta borda de descida do *clock*, o FF A muda de 1 para 0, fazendo o FF B mude de 1 para 0, e assim por diante até que o contador chegue ao estado 0000. Em outras palavras, o contador realizou um ciclo completo de contagem (de 0000 a 1111) e retornou ao estado 0000, a partir de onde começará um novo ciclo de contagem à medida que os pulsos subsequentes de *clock* forem aplicados.

Conforme descrito, neste contador, a saída de cada FF aciona a entrada de *clock* do FF seguinte. Esse tipo de contador é chamado de contador assíncrono porque os FFs não mudam de estado exatamente com o mesmo sincronismo com que os pulsos de *clock* são aplicados, apenas o FF A responde aos pulsos de *clock*. Este tipo de contador também é denominado de **contador ondulante** (*ripple counter*) devido à maneira de os FFs responderem um após o outro como um tipo de efeito de ondulação.

Os contadores ondulantes são o tipo mais simples de contadores binários, visto que requerem poucos componentes para produzir a operação de contagem desejada. Entretanto, eles têm uma grande desvantagem, causada pelo seu princípio básico de operação. Na seção 7.3.1 será solicitado ao aluno para pesquisar e descrever sobre essa desvantagem.

7.2.3 Contador Síncrono

Conforme mencionado, os contadores assíncronos apresentam alguns problemas decorrentes do seu princípio básico de funcionamento. As limitações observadas nesses circuitos podem ser superadas com o uso de contadores síncronos ou paralelos nos quais os FFs são disparados simultaneamente (em paralelo) pelos pulsos de *clock* de entrada. No entanto, visto que os pulsos de clock de entrada são aplicados em todos os FFs, algum recurso precisa ser utilizado para controlar o momento que um FF deve alterar

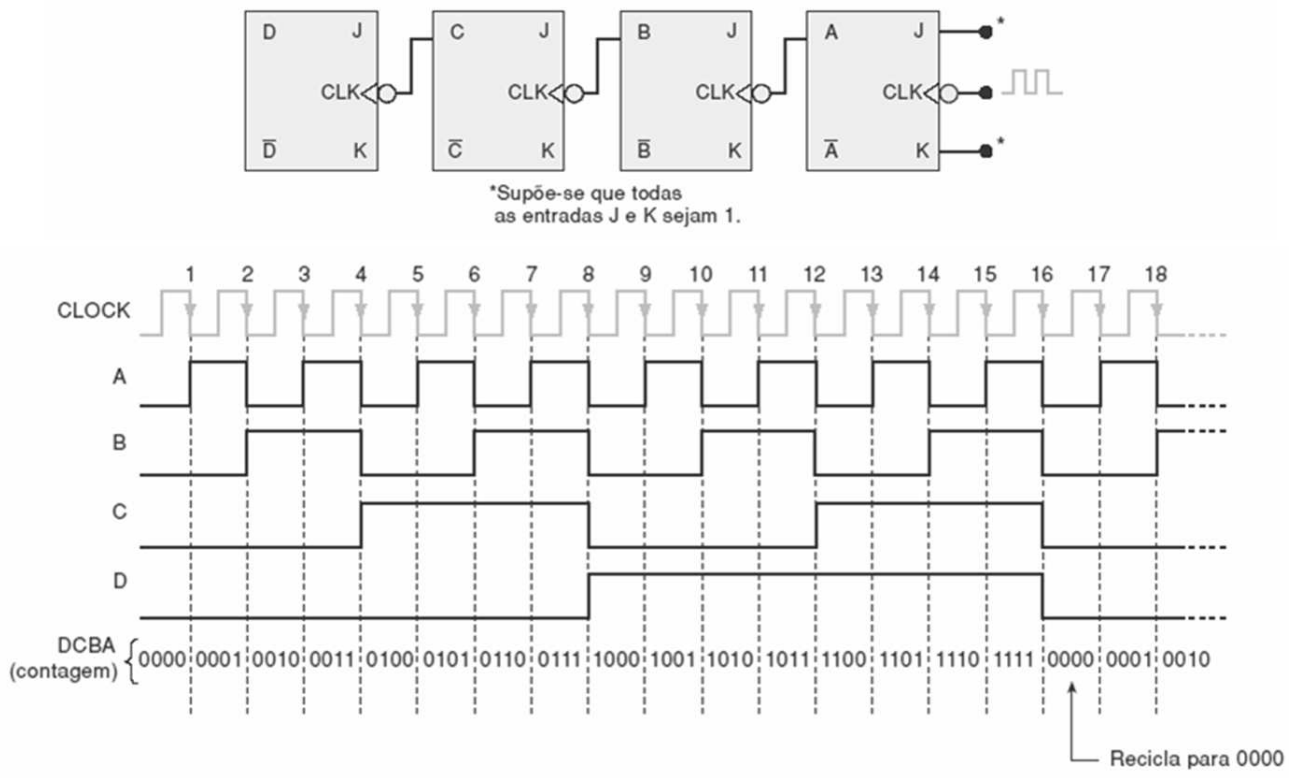


Figura 7.1: Contador assíncrono de quatro *bits*.

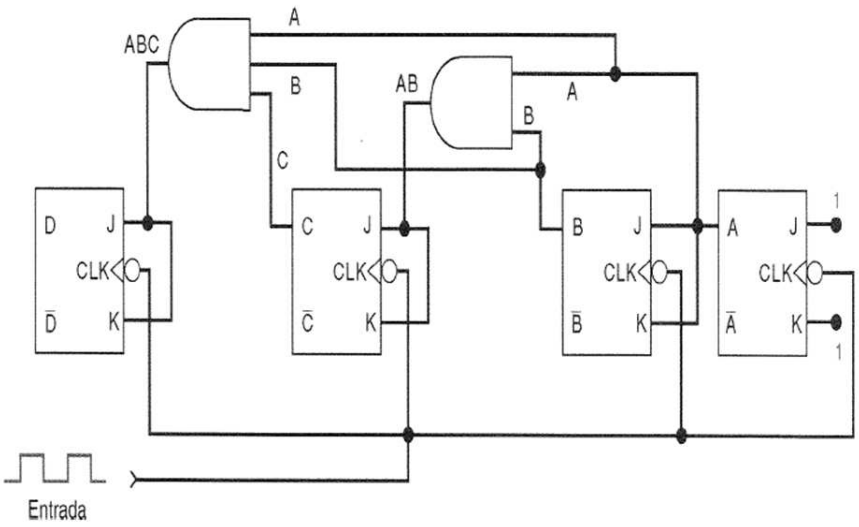
estado lógico. Desta forma, em um contador síncrono existe uma parte combinacional nas entradas de cada FF para controlar os momentos de transição.

A Figura 7.2 mostra um exemplo de contador síncrono. Se compararmos essa configuração com o seu correspondente assíncrono da Figura 7.1 temos as seguintes diferenças [TWM07]:

- As entradas CLK de todos os FFs estão conectadas juntas, de modo que o sinal de clock é aplicado simultaneamente em cada FF.
- Apenas o FF A tem suas entradas J e K ligadas de forma permanente em nível lógico alto.
- O contador síncrono requer um circuito maior do que o contador assíncrono.

Uma análise detalhada do circuito da Figura 7.2 nos permite concluir que o princípio básico para a construção de um contador síncrono pode ser enunciado da seguinte forma: **Cada FF deve ter suas entradas J e K conectadas de modo que elas estejam no nível lógico ALTO apenas quando as saídas de todos os FFs de mais baixa ordem estiveram no estado ALTO.**

Em quase todas as aplicações de contadores, as variáveis de estado, ou simplesmente estados, são consideradas como saídas. As saídas do contador podem ser codificadas de



Contagem	D	C	B	A
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1
10	1	0	1	0
11	1	0	1	1
12	1	1	0	0
13	1	1	0	1
14	1	1	1	0
15	1	1	1	1
0	0	0	0	0
.	.	.	.	.
.	.	.	.	.
.	.	etc.	.	.

Figura 7.2: Contador síncrono de módulo 16.

várias formas: binária, BCD, Gray, *etc*. Esse código e o módulo do contador determinam como serão os blocos combinatórios do próximo estado e da saída, bem como o número de *flip-flops* a ser utilizado.

7.2.4 Módulo Contador

O valor do módulo contador indica o número de estados da sequência de contagem, isto é, o número de estados que o contador percorre em cada ciclo completo de contagem até retornar ao estado inicial. Por exemplo, no circuito contador mostrado na Figura 7.2 tem 16 estados distintos (de 0000 a 1111). Assim, ele é um contador ondulante de módulo 16. Em outras palavras, o contador está limitado ao valor do módulo que é igual a 2^N, onde N é o número de FF. Esse valor é o valor máximo do módulo que pode ser obtido usando N flip-flops. O contador básico pode ser modificado para gerar um módulo menor que 2^N, fazendo com que o contador pule estados que normalmente são parte da sequência de contagem. Um dos métodos mais comuns para se fazer isso é ilustrado na Figura 7.3 em que um contador de três *bits* é mostrado. Pode-se verificar que o contador é um contador binário de módulo 8 que conta em sequência de 000 a 111. Entretanto, a presença da porta NAND altera essa sequência da seguinte forma:

- A saída da porta NAND está conectada nas entradas assíncronas da entrada CLEAR de cada FF. Enquanto a saída da porta NAND estiver em nível ALTO, não terá efeito sobre o contador. Entretanto, quando ela vai para o nível BAIXO, ocorre um sinal de CLEAR em todos os FF, logo, o contador vai imediatamente para o estado 000.
- As entradas da porta NAND são as saídas dos flip-flops B e C e, portanto, a saída da porta NAND irá para nível BAIXO sempre que B = C =1. Essa condição ocorre quando o contador passa do estado 101 para o estado 110 na borda de descida do pulso 6. O nível BAIXO na saída da porta NAND resetará imediatamente o

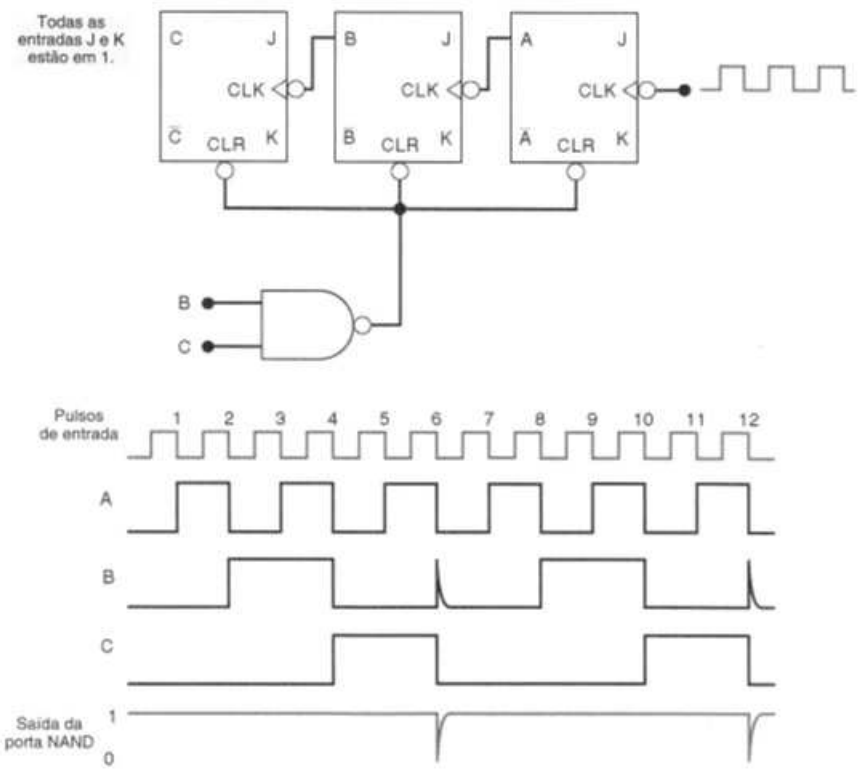


Figura 7.3: Contador de módulo 6 produzido resetando-se um contador de módulo 8 quando a contagem seis (110) ocorre.

contador para o estado 000. Uma vez que os FF foram resetados, a saída da porta NAND retorna para o nível ALTO, visto que a condição B = C =1 não existe mais.

- A sequência da contagem é, portanto, conforme ilustrada na Figura 7.4.

Lembrar, ainda, que um contador de módulo 10 é também conhecido como contador decádico. Na realidade, um contador decádico é qualquer contador que tenha 10 estados distintos, não importando a sequência. Para resumir, qualquer contador de módulo 10 é um contador decádico, e qualquer contador decádico que conte em binário de 0000 a 1001 é um contador BCD.

7.2.5 Diagrama de transição de estados

É uma forma de mostrar como os estados dos FFs mudam a cada pulso de *clock* aplicado.

As setas que conectam um círculo ao outro mostram como ocorre a mudança de um estado para outro, conforme os pulsos de *clock* são aplicados. Observando um estado de um círculo em particular, vemos qual é o estado anterior e o posterior. Um exemplo é o que se encontra ilustrada na Figura conforme ilustrada na Figura 7.5.

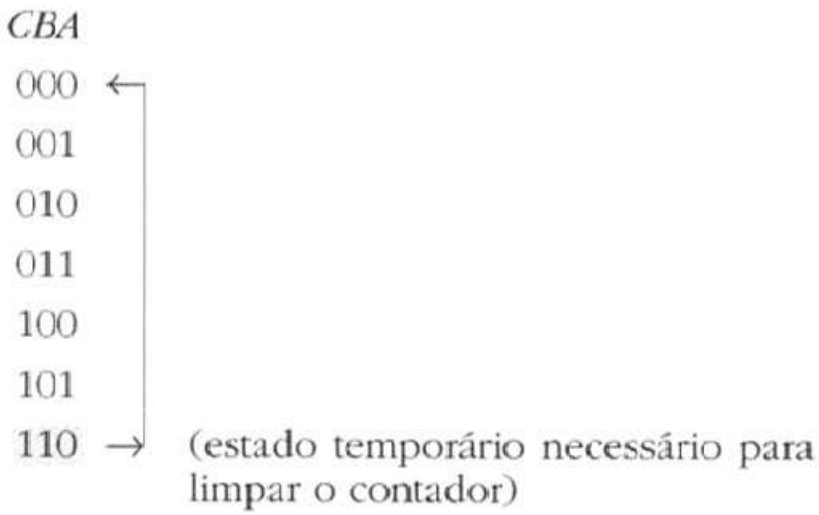


Figura 7.4: Sequência do contador de módulo 6 com *reset*.

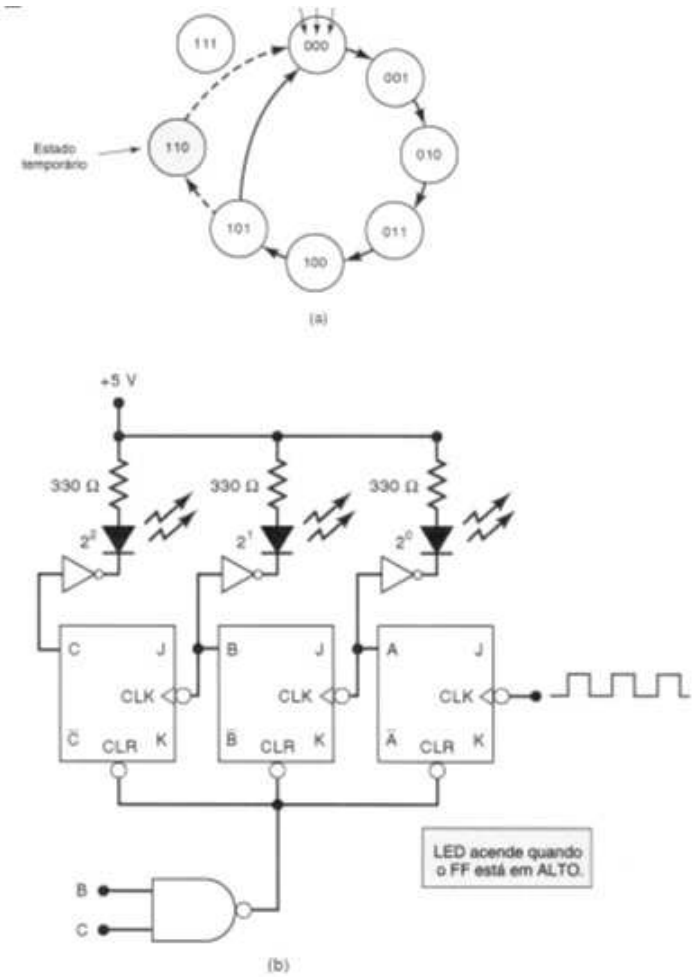


Figura 7.5: Diagrama de transição de estados.

7.2.6 Contadores em VHDL

Um contador é um circuito sequencial muito utilizado em projetos digitais. Apesar de ser simples, é um circuito interessante para mostrar recursos importantes da linguagem de descrição de hardware VHDL. O trecho de código, apresentado na Figura 7.6, mostra um contador simples, onde a saída segue uma contagem crescente.

```
1  -- contador crescente em VHDL
2  library ieee;
3  use ieee.std_logic_1164.all;
4  use ieee.std_logic_arith.all;
5
6  entity contador is
7    port (clock, reset, enable : in std_logic;
8          q : out std_logic_vector (3 downto 0));
9  end contador;
10
11 architecture cont_arch of contador is
12 begin
13   process (clock, reset)
14   variable contagem : integer range 0 to 15;
15   begin
16     if reset = '1' then
17       contagem := 0;
18     -- na borda de subida de clock...
19     elsif clock'event and clock='1' then
20       if enable='1' then
21         contagem := contagem + 1;
22       end if;
23     end if;
24     -- conversão de integer para std_logic_vector com 4 bits
25     q <= conv_std_logic_vector(contagem, 4);
26   end process;
27
28 end cont_arch;
```

Figura 7.6: Descrição VHDL de um contador.

O contador da descrição da Figura 7.6 é um contador de 4 bits com *reset* assíncrono e sinal de *enable* da contagem. A implementação em VHDL usa um processo que utiliza uma variável contagem do tipo *integer*. Na ativação do sinal *reset*, a contagem é zerada e, na borda de subida do *clock*, se *enable* estiver ativado, a contagem é incrementada. Como a saída *q* tem tipo *std_logic_vector*, é usada a função de conversão de tipos *conv_std_logic_vector* para converter a contagem inteira em um vetor de *bits*. Na linha 15, a definição da variável contagem faz com que seja criado internamente um elemento com 4 *bits* para armazenar o valor da contagem atual. Para uso da função de conversão de tipos é necessário a inclusão da biblioteca *ieee.std_logic_arith*.

Uma alternativa para a descrição de contadores é através de uma máquina de estados. Para tanto, as Figuras 7.7, 7.8 e 7.9 apresentam a descrição VHDL do contador binário com uma máquina de estados, usando três processos: o primeiro processo verifica o sinal *reset* e muda o estado na borda de subida do *clock*. O segundo processo identifica o próximo estado e o terceiro especifica a saída do circuito para cada estado (máquina de Moore).

Embora esta descrição seja mais longa que a descrição da Figura 7.6, o uso de uma máquina de estados permite uma flexibilidade maior: por exemplo, com a mesma

sequência de estados, é possível mudar os valores apresentados na saída do circuito. Para isto, basta modificar as linhas de código referentes ao processo saidas. Por exemplo, com poucas modificações é possível desenvolver a descrição de um contador módulo 16 com saídas em código Gray.

```
1  -- contador binario com ME (máquina de estados)
2  library ieee;
3  use ieee.std_logic_1164.all;
4
5  entity contbinME is
6    port (clock, reset, enable : in std_logic;
7          saida : out std_logic_vector (3 downto 0));
8  end contbinME;
9
10 architecture contbinME_arch of contbinME is
11   type tipo_estado is (E0, E1, E2, E3, E4, E5, E6, E7,
12     E8, E9, E10, E11, E12, E13, E14, E15);
13   signal estado, prox_estado: tipo_estado;
14
15 begin
16   -- processo mudaestado
17   mudaestado: process (clock, reset)
18   begin
19     if reset = '1' then
20       estado <= E0;
21     elsif clock'event and clock='1' then
22       if enable = '1' then
23         estado <= prox_estado;
24       end if;
25     end if;
26   end process;
```

Figura 7.7: Contador binário descrito com uma máquina de estados - Parte 1.

```
-- processo transicao
--   mudanca de estados segundo a maquina de estados
--   lista de sensibilidade = sinais clock, estado e reset
transicao: process (estado, enable)
begin
  if enable = '1' then
    case estado is
      when E0  => prox_estado <= E1;
      when E1  => prox_estado <= E2;
      when E2  => prox_estado <= E3;
      when E3  => prox_estado <= E4;
      when E4  => prox_estado <= E5;
      when E5  => prox_estado <= E6;
      when E6  => prox_estado <= E7;
      when E7  => prox_estado <= E8;
      when E8  => prox_estado <= E9;
      when E9  => prox_estado <= E10;
      when E10 => prox_estado <= E11;
      when E11 => prox_estado <= E12;
      when E12 => prox_estado <= E13;
      when E13 => prox_estado <= E14;
      when E14 => prox_estado <= E15;
      when E15 => prox_estado <= E0;
    end case;
  end if;
end process;
```

Figura 7.8: Contador binário descrito com uma máquina de estados - Parte 2.

```
55  -- processo saidas
56  --    determina saida do circuito
57  saidas: process (estado)
58  begin
59    case estado is
60      when E0  => saida <= "0000";
61      when E1  => saida <= "0001";
62      when E2  => saida <= "0010";
63      when E3  => saida <= "0011";
64      when E4  => saida <= "0100";
65      when E5  => saida <= "0101";
66      when E6  => saida <= "0110";
67      when E7  => saida <= "0111";
68      when E8  => saida <= "1000";
69      when E9  => saida <= "1001";
70      when E10 => saida <= "1010";
71      when E11 => saida <= "1011";
72      when E12 => saida <= "1100";
73      when E13 => saida <= "1101";
74      when E14 => saida <= "1110";
75      when E15 => saida <= "1111";
76    end case;
77  end process;
78 end contbinME_arch;
```

Figura 7.9: Contador binário descrito com uma máquina de estados - Parte 3.

7.3 Pré-Relatório

7.3.1 Pesquisa Bibliográfica

Realize pesquisas bibliográficas e procure responder as questões a seguir:

8. Quais são os principais problemas existentes nos contadores assíncronos? Descreva cada um deles de forma resumida.

9. Cite as vantagens dos contadores síncronos em relação aos contadores assíncronos.

10. Realize um resumo do procedimento para a realização de projeto de circuitos contadores síncronos e assíncronos. Procure deixar bem claro as diferenças no procedimento nos dois tipos de circuitos.

11. Como os circuitos contadores se inserem nos processadores existentes, principalmente, nos computadores e calculadoras?

12. Descreva aplicações práticas que utilizam os circuitos contadores.

7.3.2 Projetos e Simulações

Nesta seção são descritos os circuitos que devem ser projetados e/ou simulados. Na etapa de simulação, o aluno deverá utilizar o software Vivado.

Nos projetos os alunos devem apresentar todas as etapas do desenvolvimento, incluindo as tabelas verdades, os diagramas esquemáticos dos circuitos, a forma de onda na saída, considerando todas as possibilidades de ocorrência dos sinais de entrada, o código VHDL de cada circuito projetado e todas as informações adicionais que julgar necessárias para perfeita compreensão do projeto realizado.

7.3.2.1 Projeto e Simulação 1

Projete e simule um contador síncrono crescente/decrescente de módulo 10. Uma variável de entrada adicional UP/Down deve ser adicionada ao circuito para permitir selecionar o modo de contagem. Considere que se UP/Down = 1 o contador realiza a contagem crescente e, se UP/Down = 0, o contador realiza a contagem decrescente. As saídas do contador devem ser visualizadas em um *display* de 7 segmentos. Em seguida, altere seu projeto substituindo o contador síncrono por um contador assíncrono. Faça uma análise comparativa entre os dois projetos, destacando as vantagens e desvantagens de cada um deles.

7.3.2.2 Projeto e Simulação 2

Projete e simule um contador síncrono usando FFs J-K que tenha a seguinte sequência: 000, 010, 101, 110 e repete. Os estados indesejáveis (não usados) devem levar o contador sempre para o estado 000 no próximo pulso de *clock*. Além disso, seu circuito deve conter uma lógica que permite definir o estado inicial da contagem, ou seja, deve permitir selecionar um dos estados 000, 010, 101 ou 110 como estado inicial ao energizar o circuito. As saídas do contador devem ser visualizadas em um *display* de 7 segmentos.

Parte III

Projetos Finais

I Regras Gerais

I.1 Introdução

Esse documento descreve os projetos finais que devem ser realizados pelos alunos de laboratório da disciplina de Prática de Eletrônica Digital I. O objetivo do trabalho final é o de fomentar a integração dos conhecimentos adquiridos ao longo do curso e propiciar maior experiência na elaboração de projetos na área de eletrônica digital.

Os projetos devem ser realizados em duplas e entregues na data especificada pelo professor. Nas próximas Seções deste documento serão descritas as especificações dos temas propostos. Cada dupla deverá escolher e realizar apenas um dos projetos. O trabalho somente pode ser realizado pela dupla de alunos formada no início do semestre, ou por um único integrante: **é vedada a participação de trios**. Caso um integrante da dupla desista do curso ao longo do semestre, o aluno remanescente deve procurar um outro colega que esteja na mesma situação ou realizar o trabalho sozinho. Se este for o seu caso, avise o professor de sua turma de laboratório.

Os projetos deverão ser implementados em FPGA e todas as especificações descritas devem ser cumpridas. A falta de simulação ou implementação de quaisquer itens da especificação implicará na perda de pontos na nota final do trabalho. Na data da apresentação, o `bitstream` será carregado na placa e o funcionamento do circuito será apresentado ao professor.

Adicionalmente, os alunos devem preparar um relatório descrevendo, em detalhes, o processo de desenvolvimento do projeto. Neste documento devem estar descritos todos os passos relevantes para implementação do projeto. As etapas mínimas do desenvolvimento do projeto envolvem:

- Descrição do funcionamento da solução em diagrama de blocos, destacando as variáveis de entrada e saída. Se necessário, usar mais de um nível de detalhamento dos blocos;

- Especificação das entradas e saídas utilizadas no kit `BASYS3` e o correspondente arquivo de restrição;
- Código VHDL comentado. *Atenção*: plágio, se detectado, implicará em nota zero em todo o trabalho final;
- Processo de simplificação do circuito, quando aplicável;
- Diagrama esquemático do circuito;
- Simulação do(s) circuito(s) em software adequado, apresentando todas as formas de onda necessárias para se obter o completo entendimento de seu funcionamento;
- Metodologia de testes, operação e depuração de erros;
- Discussões;
- Conclusões.

Por fim, a critério do professor da turma, os alunos podem preparar uma apresentação de no máximo 5 minutos, apresentando os pontos principais do projeto e as maiores dificuldades encontradas. Após o tempo de apresentação o professor (ou uma banca de professores) terá 5 minutos para realizar perguntas a respeito dos projetos.

Qualquer dúvida sobre os temas propostos e/ou sobre a realização dos projetos devem ser sanadas, antecipadamente, com o seu professor de laboratório.

Neste semestre, a dupla poderá optar por um dos temas apresentados nesta apostila.Alguns projetos apresentam itens de desafio, que se corretamente implementados serão premiados com pontos adicionais.

ATENÇÃO
O professor de sua turma de laboratório poderá apresentar critérios adicionais com relação à escolha dos temas ou outras restrições que julgar necessárias.

I.2 Documentos Esperados

O projeto escolhido deve estar documentado da seguinte forma:

- Um relatório impresso do projeto, com os itens elencados na Seção “Introdução”. Neste documento, todos os diagramas de blocos e esquemáticos devem estar presentes e devidamente identificados. Não serão aceitos documentos manuscritos.
- Todos os códigos VHDL e de restrição necessários para simular e sintetizar a solução proposta. Se for utilizar a plataforma Vivado, compactar a pasta do projeto;
- Quando aplicável, um conjunto de no máximo 15 slides, descrevendo o trabalho realizado. Não é preciso imprimir os slides.
- Todos os arquivos eletrônicos correspondentes os esquemáticos e simulações simulações do projeto.

O conjunto dos documentos deve ser compactado e enviado ao e-mail do professor ou plataforma Moodle, no mais tardar, dois dias antes da data da apresentação do projeto. O arquivo será identificado da seguinte forma: `TXX_PYY_aluno1_aluno2.zip`, onde XX é a turma, YY a opção de tema e o restante são os primeiros nomes dos alunos da dupla.

Exemplo: Fulano e Sicrano da Turma A optaram pela opção de projeto 1. O arquivo a ser enviado será `T01_P01_fulano_sicrano.zip`.

PROJETO 1

ULA PROGRAMÁVEL

1.1 Introdução

Nesta opção, o projeto é de uma Unidade Lógico-Aritmética (ULA) que possa executar um conjunto de instruções armazenados em memória. Uma ULA é um circuito responsável por realizar operações aritméticas e lógicas em um sistema digital, onde a operação que deve ser executada é determinada por sinais de controle externos. Uma representação simplificada de uma ULA está apresentada na Figura 1.1, onde **A** e **B** são os dados (ou operandos) sobre os quais será realizada uma operação. Como há várias opções para as operações, há uma entrada **S** para realizar a seleção de qual deve ser a operação a ser apresentada na saída da ULA.

1.2 Projeto Básico

O sistema deve possuir as seguintes características:

Comprimento dos operandos A e B: 4 bits $(A_0\,A_1\,A_2\,A_3)$ e $(B_0\,B_1\,B_2\,B_3)$.

Tipos de operação: há dois tipos de operação - com um ou com dois operandos, sendo as operações lógicas ou aritméticas. As operações são selecionáveis por um comando **S** de 4 bits e o resultado **F** é calculado de acordo com Tabela 1.1.

Se operações resultarem em *overflow* ou *underflow* aritmético, o *display* deve apresentar a palavra “Erro”.

Saída de Dados: o resultado das operações deve ser apresentado nos quatro *displays* de sete segmentos. Há quatro modos de apresentação, selecionáveis por duas chaves $A_1\,A_0$: decimal (00), hexadecimal (01), binário (11) e octal (11).

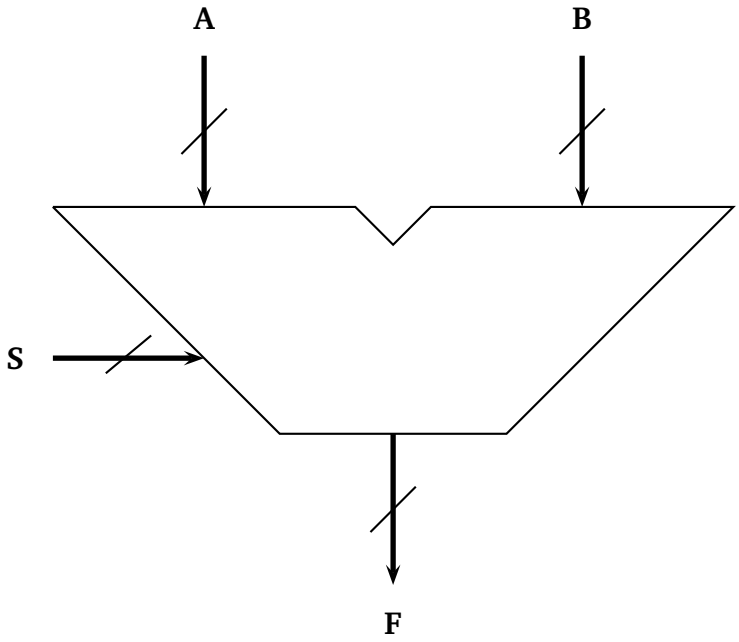


Figura 1.1: Esquema simplificado da ULA Programável.

Tabela 1.1: Operações a serem realizadas pela ULA.

Mnem.	$S_3\,S_2\,S_1\,S_0$	$F_3\,F_2\,F_1\,F_0$	Mnem.	$S_3\,S_2\,S_1\,S_0$	$F_3\,F_2\,F_1\,F_0$
ZERA	0 0 0 0	0 0 0 0	SOMA(A,B)	1 0 0 0	A+B
TUDOUM	0 0 0 1	1 1 1 1	SUBT(A,B)	1 0 0 1	A-B
OPA	0 0 1 0	$A_0A_1A_2A_3$	MULT(A,B)	1 0 1 0	A.B
OPB	0 0 1 1	$B_0B_1B_2B_3$	DIV(A,B)	1 0 1 1	A/B
OR(A,B)	0 1 0 0	OR bit a bit	MOD(A,B)	1 1 0 0	A mod B
AND(A,B)	0 1 0 1	AND bit a bit	SQA(A)	1 1 0 1	A^2
XOR(A,B)	0 1 1 0	XOR bit a bit	NEGA	1 1 1 0	-A
NOTA	0 1 1 1	NOT bit a bit	ADD1	1 1 1 1	A+1

Modos de Operação: há dois modos de uso do sistema, selecionados por uma chave. No modo "Calculadora" (CALC), os operandos A e B, assim como o seletor de operações **S**, são inseridos por chaves.

No modo "Programa" (PROG), uma sequência de operações é armazenada em uma memória e o resultado da operação corrente é armazenado em um registrador chamado de "Acumulador" (ACC). O valor do acumulador é visualizado no *display*.

O procedimento para inserção das operações na memória é simples: cada operação é definida por chaves e inserida em posições sequencias de memória, por intermédio de um botão "Enter".

Cada vez que o botão "Enter" é pressionado, um contador chamado "Endereço de Memória" (EDM) incrementa de uma unidade o valor de sua saída, de modo a

indicar a posição de memória onde a operação será armazenada. A memória tem 1024 posições (10 bits), e cada uma armazena uma operação de 4 bits. A posição inicial de memória é sempre zero. A cada passo, a posição de memória corrente (seu endereço) é apresentada em hexadecimal nos três primeiros *displays*, e o código da operação a ser inserida no quarto *display* (também em hexadecimal).

Após inserida a última operação, o botão "Run" inicia a execução do programa.

No seu projeto, deixe claro quais os resultados para várias possibilidades de entrada e saída (isto é, elabore um manual de instruções de uso do seu circuito).

1.3 Desafios Adicionais

- Aprimore a operação da ULA no modo "Calculadora " para contemplar operandos com 8 bits (+1 ponto).
- Aprimore seu projeto para que o conteúdo do ACC seja inserido na memória (+1 ponto);

PROJETO 2

VALOR NUMÉRICO DE POLINÔMIOS

Implemente um calculador que forneça no *display* o resultado de $P(x)$ ou de sua derivada $P'(x)$, onde $P(x)$ é um polinômio de grau n, com coeficientes $(c_0, c_1, c_2, \cdots, c_n)$. Os valores de x e dos coeficientes são inteiros com sinal.

2.1 Projeto Básico

O circuito tem dois modos de operação, habilitados por uma chave.

- Modo "valor numérico". Neste modo:
  - O usuário cadastra o grau do polinômio usando as chaves. O valor deve aparecer no *display*;
  - O usuário pressiona o botão "Enter", e inicia a inserção dos coeficientes em notação sinal-e-amplitude. Cada coeficiente também é inserido pelo botão "Enter" e mostrado no *display* (*display* mais à esquerda mostra o sinal);
  - Após o último coeficiente ser inserido, o *display* apresenta a informação "OK";
  - O usuário apresenta o valor de x usando as chaves (também sinal-e-amplitude), e o valor de $P(x)$ aparece no *display*;
  - Quando o usuário pressiona a chave "derivada", o valor de $P'(x)$ aparece no *display*;
- Modo "operações com polinômios". Neste modo:
  - O usuário cadastra o grau do primeiro polinômio usando as chaves. O valor deve aparecer no *display*;

- O usuário pressiona o botão “Enter”, e inicia a inserção dos coeficientes em notação sinal-e-amplitude. Cada coeficiente também é inserido pelo botão “Enter” e mostrado no *display* (*display* mais à esquerda mostra o sinal);
- Após o último coeficiente ser inserido, o *display* apresenta “OKP1”;
- O usuário cadastra o segundo polinômio realizando os mesmos passos. Após o último coeficiente ser inserido, o *display* apresenta novamente “OKP2”.
- Se o usuário pressiona o botão (produto), no display aparece:
  - “Prod” durante 2 segundos;
  - A sequência dos coeficientes de $p_1(x).p_2(x)$, do menor grau para maior grau, durante 2 segundos cada;
  - “End”
- A mesma funcionalidade é esperada para os botões (+) e (-), correspondendo à soma e subtração de $p_1(x)$ e $p2(x)$. No display deve aparecer “Add” ou “Sub”.

2.2 Desafio (+2 pontos)

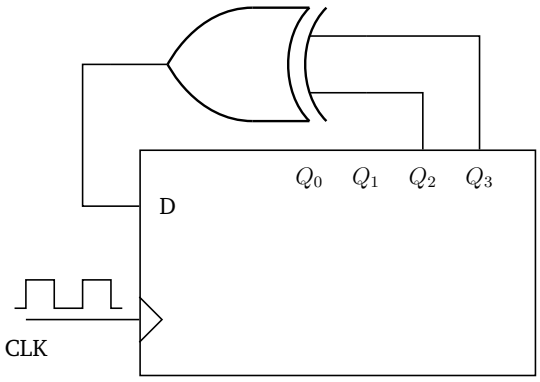
Adapte o funcionamento para calcular uma raiz do polinômio cadastrado (modo “valor numérico”), no intervalo [a, b], onde a e b também são inseridos.

PROJETO 3

GERADOR DE NÚMEROS ALEATÓRIOS

É possível implementar uma sequência pseudo-aleatória de números utilizando um registrador de deslocamento de saídas $(Q_0\, Q_1 \cdots Q_n, \cdots, Q_m)$ e uma porta XOR.

A porta XOR tem duas entradas: o bit mais significativo do registrador (m) e o n-ésimo bit (n) das saídas disponíveis. Um exemplo está na figura abaixo, com $m = 4$ e $n = 3$.



É possível mostrar que, se o polinômio $1 + x^n + x^m$ for irredutível e primo sobre o corpo de Galois, a sequência de estados gerada é máxima, isto é, passa por todos os 2^m estados, exceto o **0**.

Um detalhe da implementação é que o estado correspondente à todas as saídas $Q_k = 0$ fica "travado". Portanto, este estado é proibido e não aparece na sequência de estados. Assim, sua implementação deve prever a escolha de um estado inicial.

A Tabela a seguir apresenta algumas escolhas para n e m. O comprimento da sequência corresponde ao número total de estados.

m	*n*	*Length*
3	2	7
4	3	15
5	3	31
6	5	63
7	6	127
9	5	511
10	7	1023
11	9	2047
15	14	32767
17	14	131071
18	11	262143
20	17	1048575
21	19	2097151
22	21	4194303
23	18	8388607
25	22	33554431
28	25	268435455
29	27	536870911
31	28	2147483647
33	20	8589934591
35	33	34359738367
36	25	68719476735
39	35	549755813887

3.1 Projeto Básico

Implemente um gerador de números aleatórios de $2^m - 1$ estados. Tome um grupo de k dos m bits do registrador de deslocamento e apresente em decimal no *display* (até 8192) e nos leds (até $65.535 = 2^{16} - 1$). Os valores de m, n e k são inseridos por chaves.

O circuito tem dois modos de funcionamento: "Gera" e "Valida", selecionados por uma chave.

Modo "Gera". Neste modo o *display* exibe sequencialmente os números decimais correspondentes ao grupo de k bits. A velocidade de apresentação é variável, de acordo com 4 opções também selecionadas por chaves: 1, 2, 5 e 10 Hz. A qualquer momento, o botão "Para/Continua" pode ser pressionado para visualizar o resultado.

Modo "Valida". Neste modo:

- O usuário apresenta um grupo de **S** de 4 bits usando chaves;
- O usuário pressiona o botão "Run" e o gerador passa por todos os estados possíveis;
- Ao retornar ao estado inicial, para e apresenta "OK" no *display*;
- O usuário pressiona o botão "Estatística";
- O sistema apresenta quantas vezes **S** aparece nos últimos 4 bits do registrador durante a passagem por todos os estados.

3.2 Desafios Adicionais

- Aprimore seu projeto para **S** ser de tamanho variável (+1 ponto);

- Use cada segmento dos *displays* (incluindo o ponto decimal) para representar um bit - portanto, $4 \times 8 = 32$ bits disponíveis. Neste caso, é possível visualizar na Basys3 até 48 bits em binário, fazendo uso desta estratégia e complementando com os 16 bits dos leds acima das chaves (+1 ponto).

PROJETO 4

MULTIPLICAÇÃO DE MATRIZES

Implemente um multiplicador de duas matrizes $\mathbf{A}_{m\times n}$ e $\mathbf{B}_{n\times k}$. As dimensões e os elementos da matriz resultante $\mathbf{C}_{m\times k} = \mathbf{A}\cdot\mathbf{B}$ devem ser visualizados no *display*.

4.1 Projeto Básico

As dimensões e elementos de $\mathbf{A}$ e $\mathbf{B}$ devem ser inseridos da seguinte forma:

- Inserem-se pelas chaves as dimensões da primeira matriz, sequencialmente (linhas e colunas);
- Pressiona-se o botão "Enter";
- Inicia-se o processo de inserção sequencial, também pelas chaves, dos elementos da matriz (todos os elementos da primeira linha, depois todos os da segunda, e assim sucessivamente). Após a inserção da cada elemento, pressiona-se o botão "Enter";
- Após o último elemento da matriz ser inserido, os *displays* apresentam o nome da matriz e suas dimensões (use 3 dos 4 *displays*);
- Repete-se o processo para a segunda matriz;
- O *display* apresenta `C n m`, onde `m` e `n` são as dimensões da matriz resultante;
- Ao pressionar o botão "Mostra", o *display* apresenta sequencialmente os elementos da matriz $\mathbf{C}$ (linhas e colunas), cada elemento por 1 segundo.

Use o ponto decimal do *display* mais à esquerda para indicar o sinal negativo do resultado. Caso algum elemento tenha módulo superior a 9999, os *displays* apresentam a mensagem "erro".

4.2 Desafio Adicional (+2 pontos)

Refaça o projeto para contemplar matrizes complexas.

PROJETO 5

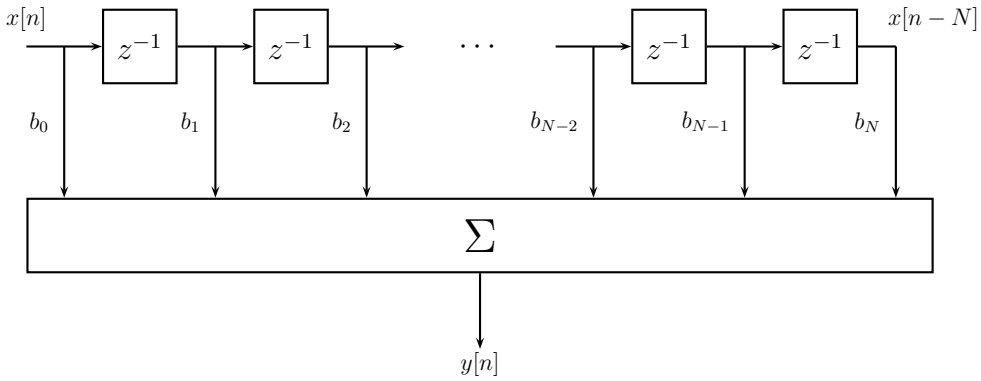
FILTROS DIGITAIS

Um filtro digital FIR (*Finite Impluse Response*) de ordem N, com entrada $x[n]$ e saída $y[n]$ pode ser descrito pela seguinte expressão:

$$y[n] = b_0x[n]+b_1x[n-1]+\cdots+b_{N-1}x[n-(N-1)]+b_Nx[n-N] = \sum_{k=0}^{N} b_k \cdot x[n-k]. \quad (5.1)$$

Os valores de b_k, $k = 0, 1, \cdots, N$, são chamados de **coeficientes** do filtro. Os números inteiros n são índices que indicam posições dos vetores y e x e, no caso de aplicações no domínio do tempo, representam um instante de tempo n.

A fórmula acima pode ser interpretada como: "no instante n, a saída $y[n]$ é uma combinação linear das amostras presente ($x[n]$) e passadas ($x[n-1], x[n-2], \cdots$) do sinal de entrada". A figura abaixo apresenta um diagrama esquemático de um filtro FIR.



5.1 Projeto Básico

Implemente um filtro digital de ordem máxima $N = 64$. Os coeficientes b_k e a sequência de entrada são compostos de valores inteiros, sendo $x[n]$ com comprimento

máximo de 256 posições. Ambas as sequencias devem ser armazenadas em uma ROM.

A saída deve ser apresentada nos *displays* em formato decimal, a uma taxa de 1 Hz. Para a sua sequência de entrada, assim como para os coeficientes estipulados, escreva em seu relatório qual deve ser a saída esperada. Sugestão: use o Matlab para calcular os valores da saída.

Use o ponto decimal do *display* mais à esquerda para indicar o sinal negativo do resultado. Caso algum elemento tenha módulo superior a 9999, os *displays* apresentam a mensagem "erro".

5.2 Desafio Adicional (+2 pontos)

Um filtro digital IIR (*Infinite Impluse Response*) com entrada $x[n]$ e saída $y[n]$ pode ser descrito pela seguinte expressão:

$$\begin{aligned} y[n] = b_0x[n] + b_1x[n-1] + \cdots + b_{M-1}x[n-(M-1)] + b_Mx[n-M] \\ - (a_1y[n-1] + a_2y[n-2] + \cdots + a_{N-1}y[n-(N-1)] + a_Ny[n-N]) = \\ \sum_{k=0}^{M} b_k \cdot x[n-k] - \sum_{k=0}^{N} a_k \cdot y[n-k]. \end{aligned} \quad (5.2)$$

Implemente um filtro IIR com $M+1$ coeficientes b_k e N coeficientes a_k. Os valores máximos para o comprimento dos vetores de coeficientes é de 64. Armazene 256 amostras dos sinais de entrada em uma ROM e 512 amostras do sinal de saída em uma RAM. Todos os elementos mencionados assumem valores inteiros.

Ao pressionar o botão "Filtra", a saída deve ser apresentada nos *displays* em formato decimal, a uma taxa de 1 Hz. Escreva em seu relatório qual deve ser a saída esperada para os sinais que você definiu.

Assim como no projeto básico, use o ponto decimal do *display* mais à esquerda para indicar o sinal negativo do resultado. Caso algum elemento tenha módulo superior a 9999, os *displays* apresentam a mensagem "erro".

BIBLIOGRAFIA

[Hig] Ricardo Tokio Higuti. *Apostila de Circuitos Digitais I - Experiência 3 - Multiplexadores e Decodicadores*. Universidade Estadual Paulista Júlio de Mesquita Filho - Campus Ilha Solteira - Departamento de Engenharia Elétrica.

[IC07] Ivan Valeije Idoeta and Francisco Gabriel Capuano. *Elementos de Eletrônica Digital*. Érica, 40ª edition, 2007.

[Ped04] Volnei A. Pedroni. *Circuit Design with VHDL*. MIT Press, 1 edition, 2004.

[SM06] Edison Spina and Edson T. Midorikawa. *Multiplexação de Displays - Apostila do Curso de Laboratório de Sistemas Digitais da USP (EPUSP)*. Escola Politécnica da USP - Departamento de Engenharia de Computação e Sistemas Digitais, São Paulo, 2006.

[SS07] Adel S. Sedra and Kenneth C. Smith. *Microeletrônica*. Pearson Prentice Hall, 5ª edition, 2007.

[TWM07] Ronald J. Tocci, Neal S. Widmer, and Gregory L. Moss. *Sistemas Digitais - Princípios e Aplicações*. Prentice-Hall, 10ª edition, 2007.

[Uye02] John P. Uyemura. *Sistemas Digitais - Uma abordagem integrada*. Pioneira Thomson Learning, 1 edition, 2002.