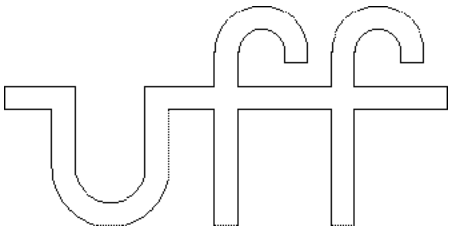
Apostila
com
Códigos de Programas Demonstrativos
usando a linguagem VHDL
para
Circuitos Digitais

(Versão A2020M05D07)



Universidade Federal Fluminense

Alexandre Santos de la Vega

Departamento de Engenharia de Telecomunicações – TET

Escola de Engenharia – TCE

Universidade Federal Fluminense – UFF

Maio – 2020

.

621.3192 (*) D278 (*) 2020	de la Vega, Alexandre Santos Apostila com Códigos de Programas Demonstrativos usando a linguagem VHDL para Circuitos Digitais / Alexandre Santos de la Vega. – Niterói: UFF/TCE/TET, 2020. 137 (sem romanos) ou 151 (com romanos) p. (*) Apostila com Códigos de Programas Demonstrativos – Graduação, Engenharia de Telecomunicações, UFF/TCE/TET, 2020. 1. Circuitos Digitais. 2. Técnicas Digitais. 3. Telecomunicações. I. Título.

(*) OBTER INFO NA BIBLIOTECA E ATUALIZAR !!!

Aos meus alunos.

Prefácio

O trabalho em questão cobre os tópicos abordados na disciplina Circuitos Digitais.

O presente volume apresenta um conteúdo prático, utilizando códigos de programas demonstrativos, baseados na linguagem VHDL. O conteúdo teórico pode ser encontrado no volume entitulado Apostila de Teoria para Circuitos Digitais.

As apostilas foram escritas com o intuito de servir como uma referência rápida para os alunos dos cursos de graduação e de mestrado em Engenharia de Telecomunicações da Universidade Federal Fluminense (UFF).

O material básico utilizado para o conteúdo teórico foram as minhas notas de aula, que, por sua vez, originaram-se em uma coletânea de livros sobre os assuntos abordados.

Os códigos de programas demonstrativos são completamente autorais.

A motivação principal foi a de aumentar o dinamismo das aulas. Portanto, deve ficar bem claro que estas apostilas não pretendem substituir os livros textos ou outros livros de referência. Muito pelo contrário, elas devem ser utilizadas apenas como ponto de partida para estudos mais aprofundados, utilizando-se a literatura existente.

Espero conseguir manter o presente texto em constante atualização e ampliação.

Correções e sugestões são sempre bem-vindas.

Rio de Janeiro, 08 de setembro de 2017.

Alexandre Santos de la Vega
UFF / TCE / TET

Agradecimentos

Aos professores do Departamento de Engenharia de Telecomunicações da Universidade Federal Fluminense (TET/UFF) que colaboraram com críticas e sugestões bastante úteis à finalização deste trabalho.

Aos funcionários e ex-funcionários do TET, Arlei, Carmen Lúcia, Eduardo, Francisco e Jussara, pelo apoio constante.

Aos meus alunos, que, além de servirem de motivação principal, obrigam-me sempre a tentar melhorar, em todos os sentidos.

Mais uma vez, e sempre, aos meus pais, por tudo.

Rio de Janeiro, 08 de setembro de 2017.
Alexandre Santos de la Vega
UFF / TCE / TET

Apresentação

- O presente documento encontra-se em constante atualização.
- Ele consta de listagens de códigos de programas demonstrativos, baseados na linguagem VHDL, desenvolvidos para melhor elucidar os tópicos desenvolvidos em sala de aula.
- Na preparação das aulas foram utilizados os seguintes livros:
  - Livros indicados pela ementa da disciplina: [IC08], [Tau82].
  - Outros livros indicados: [HP81], [Rhy73], [TWM07], [Uye02].
- Este documento aborda os seguintes assuntos:
  - Linguagem de descrição de *hardware* (*Hardware Description Language* ou HDL).
  - Linguagem de descrição de *hardware* VHDL.
  - Circuitos digitais combinacionais.
  - Circuitos digitais sequenciais.

Sumário

Parte I

Introdução

Capítulo 1

Linguagens de descrição de *hardware*

1.1 Introdução

- Desde a implementação do primeiro dispositivo eletrônico em circuito integrado, os avanços tecnológicos têm possibilitado um rápido aumento na quantidade de elementos que podem ser combinados em um único circuito nesse tipo de implementação.
- Naturalmente, com a oferta de uma maior densidade de componentes, a complexidade dos circuitos projetados cresce na mesma taxa.
- Porém, a capacidade de um ser humano em lidar com a idealização, o projeto, a documentação e a manutenção de sistemas com um grande número de componentes é extremamente limitada.
- Dessa forma, torna-se necessário o uso de ferramentas de apoio, adequadas a tal tipo de problema.
- Existem duas técnicas de projeto largamente utilizadas na abordagem de problemas de elevada complexidade:
  - Adotar uma visão hierárquica na elaboração do sistema, de forma que, em cada nível de representação, toda a complexidade dos níveis inferiores seja ocultada.
  - Aumentar o nível de abstração na descrição do sistema, de forma que o foco esteja mais na função desempenhada e menos na implementação propriamente dita.

1.2 Abordagem hierárquica

- Na definição de um sistema de baixa complexidade, pode-se descrever a sua operação de uma forma simples e direta.
- Por outro lado, na definição de sistemas com complexidade elevada, pode-se utilizar o conceito organizacional de hierarquia.
- Em uma abordagem hierárquica, um sistema de complexidade genérica é recursivamente dividido em módulos ou unidades mais simples. O ponto de parada da recursividade é subjetivo e costuma ser escolhido como a descrição comportamental mais simples possível e/ou desejada.

- Nesse sentido, o sistema completo pode ser interpretado como o módulo mais complexo ou mais externo da hierarquia.

- A abordagem hierárquica facilita a descrição, a análise e o projeto dos circuitos, uma vez que cada módulo pode ser tratado como um circuito único, isolado dos demais.

- A descrição hierárquica no sentido do todo para as partes mais simples é chamada de *top-down*.

- Por outro lado, a descrição hierárquica no sentido das partes mais simples para o todo é chamada de *bottom-up*.

- Normalmente, aplica-se um processo *top-down* para a especificação e um processo *bottom-up* para a implementação de sistemas.

1.3 Níveis de abstração

A descrição, a análise e a síntese de circuitos podem ser realizadas em diversos níveis de abstração.

Do ponto de vista do comportamento modelado, os seguintes níveis podem ser considerados:

- Físico-matemático: que adota equações matemáticas para descrever um modelo físico de comportamento. Obviamente, é o modelo mais próximo do comportamento físico do circuito. É tipicamente utilizado na descrição funcional de circuitos analógicos.

- Lógico: que emprega equações lógicas na sua descrição. É naturalmente utilizado na descrição funcional de circuitos digitais.

- Estrutural: que utiliza, intrinsicamente, uma descrição hierárquica do circuito modelado. Inicialmente, são definidos, testados e validados, blocos de baixa complexidade. Em seguida, realizando-se instanciações desses blocos, são definidos blocos de complexidade mais elevada. Tal processo é repetido até que o sistema desejado seja adequadamente definido. Portanto, esse é um modelo que preserva a visão da estrutura física dos circuitos.

- Comportamental: que apresenta um nível de representação mais abstrato e mais distante do sistema físico. Encontra aplicação direta em testes de funcionalidade dos circuitos.

Do ponto de vista da complexidade dos sistemas, os seguintes níveis podem ser considerados:

- Componentes: que representam os elementos básicos de circuitos.

- Células básicas: que são circuitos de baixa complexidade.

- Blocos funcionais: que são circuitos de média complexidade.

- Sistemas: que são circuitos de alta complexidade.

1.4 Linguagens de descrição de *hardware*

- Na área de projeto de circuitos integrados, o uso de uma Linguagem de Descrição de *Hardware* (*Hardware Description Language* ou HDL) tem sido proposto, a fim de permitir uma descrição mais abstrata dos seus elementos constituintes e de possibilitar que estes sejam organizados de forma hierárquica.
- Uma HDL é uma linguagem de modelagem, utilizada para descrever tanto a estrutura quanto a operação de um *hardware* digital.
- Em linguagens de programação comuns, os processos de interpretação e de compilação podem ser modelados como a tradução de uma linguagem entendida por uma máquina virtual para uma outra linguagem asssociada a uma outra máquina virtual.
- No caso de uma HDL, o modelo é um pouco diferente. A partir da descrição apresentada pelo código elaborado, o compilador deve inferir um *hardware* digital equivalente.
- Portanto, por meio de uma HDL, além de um mapeamento lingüístico e/ou matemático, é realizado um mapemento físico.
- De acordo com o seu comportamento funcional, um *hardware* digital pode ser classificado da seguinte forma:
  - Combinacional: sistema instantâneo (ou sem memória), com operação concorrente de eventos.
  - Seqüencial: sistema dinâmico (ou com memória), com operação seqüencial de eventos.
- Logo, uma HDL deve ser capaz de descrever ambos os comportamentos: o concorrente e o seqüencial.
- As aplicações típicas para uma HDL são as seguintes:
  - Documentação de circuitos digitais.
  - Análises de circuitos digitais, tais como: simulações e checagens diversas.
  - Síntese (projeto) de circuitos digitais. Uma vez definida uma implementação alvo, o compilador infere um circuito equivalente à descrição HDL e gera um código adequado para tal implementação. Sínteses típicas são as seguintes: a geração de código para configurar dispositivos lógicos programáveis e a geração de máscaras (*layout*) para fabricação de circuitos integrados.
- Algumas características que levam uma HDL a ser largamente empregada são as seguintes:
  - Apresentar padrões bem estabelecidos e bem documentados.
  - Apresentar características encontradas em outras HDLs.
  - Possuir vasta literatura disponível.
  - A existência de várias ferramentas computacionais para a HDL em questão, tais como: editores, compiladores, simuladores, checadores diversos.
  - A existência de ferramentas computacionais de diversos tipos, tais como:
    - Domínio público ou comercial.

    - Implementada isoladamente ou incluída em ambiente de desenvolvimento integrado (*Integrated Development Environment* ou IDE).
    - Implementada em diversas plataformas computacionais.

- Exemplos de HDL
  - Independentes de tecnologia e de fabricante: VHDL, Verilog, SystemVerilog.
  - Dependentes de fabricante: AHDL (Altera HDL).

- A Tabela 1.1 apresenta uma lista de fabricantes, produtos e funções, que lidam com HDL.

Fabricante	Produto	Função
Altera	Quartus II	Síntese e simulação
Xilinx	ISE	Síntese e simulação
Menthor Graphics	Precision RTL	Síntese
	ModelSim	Simulação
Synopys/Synplicity	Design Compiler Ultra	Síntese
	Synplify Pro/Premier	
	VCS	Simulação
Cadence	NC-Sim	Simulação

Tabela 1.1: Lista de fabricantes, produtos e funções, que lidam com HDL.

Capítulo 2

Introdução à linguagem VHDL

A linguagem VHDL é apresentada a seguir, de forma introdutória. Para que se adquira um conhecimento mais aprofundado sobre a linguagem, é recomendado consultar uma literatura específica (manuais e livros especializados).

2.1 Histórico da linguagem VHDL

- Durante o desenvolvimento do programa *Very High Speed Integrated Circuits* (VHSIC), iniciado em 1980 pelo Departamento de Defesa (DoD) dos Estados Unidos da América, surgiu a necessidade de uma HDL específica para lidar com os tipos de circuitos integrados envolvidos no programa. Em função disso, foi proposta a primeira versão da linguagem VHDL (*VHSIC Hardware Description Language*).

- A linguagem VHDL continuou a ser desenvolvida pelo IEEE (*Institute of Electrical and Electronics Engineers*) e foi a primeira HDL padronizada, por meio dos padrões IEEE Standard 1076 (*Standard VHDL Language Reference Manual* - 1987) e IEEE Standard 1164 (*Standard Multivalue Logic System for VHDL Model Interoperability* - 1993).

- Os padrões IEEE são revisados, pelo menos, a cada cinco anos. Portanto, já foram gerados os padrões VHDL-1987, VHDL-1993, VHDL-2002 e VHDL-2008.

2.2 VHDL como linguagem

Como qualquer linguagem escrita, VHDL utiliza um conjunto específico de símbolos e de regras que definem aspectos de sintaxe e de semântica.

VHDL não é uma linguagem de programação de computadores. Ela é uma linguagem que tem uma finalidade distinta e específica, sendo classificada como Linguagem de Descrição de *Hardware* (*Hardware Description Language* ou HDL). Nesse sentido, VHDL foi desenvolvida para a descrição de circuitos eletrônicos digitais, sendo aplicada na documentação, simulação e síntese automática de tais circuitos.

É fundamental compreender a diferença entre uma linguagem de programação e uma HDL. Enquanto o código de uma linguagem de programação é traduzido em comandos reconhecidos e executáveis por uma máquina computacional, o código de uma HDL é traduzido em um circuito que é utilizado para construir tais máquinas.

Mesmo assim, VHDL apresenta elementos comuns a diversas linguagens de programação modernas, alguns dos quais são discutidos a seguir.

2.2.1 Considerações gerais

- Arquivos contendo código VHDL são formatados em tipo TEXTO.

- O nome do arquivo que contém o código VHDL (`nome.vhd`) deve ser o mesmo nome da entidade mais externa na hierarquia de circuitos descrita pelo arquivo em questão.

- Comentários são iniciados com dois hífens consecutivos (- -).

- VHDL não é *case sensitive*.

  Logo: palavra ≡ PaLaVrA ≡ PALAVRA.

- Um sinal físico, com valores binários, é definido pelo tipo BIT. Por sua vez, um conjunto de tais sinais é definido pelo tipo BIT_VECTOR.

- O valor de um sinal do tipo BIT é indicado com aspas simples (p.ex.: '0' e '1'), enquanto o valor de um sinal do tipo BIT_VECTOR é delimitado por aspas duplas (p.ex.: "0000", "0101" e "1111").

- Duas operações básicas no uso de VHDL são a compilação e a simulação. Em uma forma bastante simplificada, elas podem ser definidas como a seguir. A partir de uma descrição VHDL do circuito digital, armazenada em arquivo do tipo texto, o compilador VHDL infere um circuito equivalente na implementação alvo e armazena tal informação em um outro arquivo do tipo texto. A partir do arquivo que contém informação sobre o circuito compilado e de uma descrição de sinais de teste, armazenada em arquivo do tipo texto, o simulador VHDL calcula os sinais gerados pelas saídas do circuito.

2.2.2 Identificadores

Um identificador é uma cadeia de caracteres (*string*), usada para designar um nome único a um objeto, de forma que ele possa ser adequadamente referenciado.

Por padrão, Verilog aceita identificadores da ordem de 1000 caracteres.

2.2.3 Palavras reservadas

As palavras reservadas (*reserved words* ou *keywords*) são identificadores que possuem um significado especial dentro da linguagem. Assim, seu uso é restrito à sua definição original e, uma vez que não podem ser redefinidas, elas não podem ser empregadas para nenhum outro propósito.

A Figura 2.1 apresenta as palavras reservadas de VHDL.

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

Figura 2.1: Palavras reservadas de VHDL.

2.2.4 Identificadores definidos pelo usuário

Algumas regras básicas para a construção de identificadores são as seguintes:

- Todo identificador é formado por uma seqüência de caracteres (*string*) única, de qualquer comprimento.
- Identificadores podem ser formados apenas com letras minúsculas e/ou maiúsculas (a até z e A até Z), com números de 0 a 9 e com o símbolo "_" (sublinhado ou *underscore*).
- Todo identificador deve começar com uma letra.
- O símbolo de *underscore* não pode ser usado como primeiro nem como último caractere do identificador. Também não é permitido usar dois símbolos de *underscore* consecutivos.
- VHDL não é *case sensitive*. Logo: identificador ≡ IdEnTiFiCaDoR ≡ IDENTIFICADOR.

Os padrões mais recentes permitem o uso de um conjunto expandido de caracteres, incluindo a utilização de hífen e de acentos. Porém, as regras acima são suficientes para garantir a compatibilidade entre os diversos padrões.

2.2.5 Elementos sintáticos

Além das palavras reservadas e dos identificadores definidos pelo usuário, podem-se utilizar símbolos especiais para escrever o código VHDL. Assim como as palavras reservadas, seu uso é restrito à sua definição original.

A Figura 2.2 apresenta símbolos especiais de VHDL.

Símbolo	Significado
−−	Comentário
;	Terminador
(	Parêntese da esquerda
)	Parêntese da direita
:	Separação entre elemento e tipo
<>	Declaração de faixa indefinida (*box*)
#	Notação `base#número#`
.	Notação de ponto
'	Aspas simples ou marca de *tick*
"	Aspas duplas
&	Concatenador
<=	Atribuição a sinal
:=	Atribuição a variável/constante
=>	Atribuição a elemento de conjunto

Símbolo	Significado
|	OR condicional
=>	Símbolo de THEN em CASE
+	Adição ou identidade unária
−	Subtração ou negação unária
*	Multiplicação
/	Divisão (com truncamento)
**	Exponenciação
=	Igual a
/=	Diferente de
<	Menor do que
>	Maior do que
<=	Menor do que ou igual a
>=	Maior do que ou igual a

Figura 2.2: Símbolos especiais de VHDL.

2.3 Conceitos básicos sobre o código VHDL

2.3.1 Elementos básicos

- Alguns elementos básicos de um código VHDL são: constante, variável, sinal e operador.
- Constantes são geralmente empregadas para flexibilização e/ou otimização do código.
- Variável é um elemento abstrato para armazenamento de informação matemática.
- Sinal é o elemento da linguagem associado com elementos físicos de conexão (pino e fio).
- Operadores representam as relações funcionais básicas. Eles são definidos com as palavras reservadas e com os elementos sintáticos, sendo organizados em seis classes: atribuição, lógica, aritmética, comparação, deslocamento e concatenação. A Tabela 2.1 apresenta os operadores de VHDL.

Classe	Operadores
Atribuição	<=, :=, =>.
Lógica	NOT, AND, OR, XOR, NAND, NOR, XNOR.
Aritmética	+, -, *, /, **, ABS, MOD(1), REM(2).
Comparação	=, /=. <, >, <=, >=.
Deslocamento	SLL, SRL, SLA, SRA, ROL, ROR.
Concatenação	& (“,” e OTHERS).

(1) *Module*: resto de a/b, com sinal de b.

(2) *Remainder*: resto de a/b, com sinal de a.

Tabela 2.1: Operadores de VHDL.

2.3.2 Tipos de execução

- De acordo com o tipo de execução, os códigos VHDL são divididos em: concorrente e seqüencial.
- Códigos concorrentes são empregados para descrever circuitos digitais combinacionais. Por sua vez, os códigos seqüenciais são utilizados para descrever tanto circuitos combinacionais quanto circuitos seqüenciais.
- As instruções VHDL são naturalmente executadas de forma concorrente. Assim, embora o código seja organizado em linhas, todas as linhas têm igual precedência.
- Para que um código seja considerado seqüencial, isso deve ser forçado. O mecanismo mais comum para forçar um código a ser seqüencial é denominado de processo, definido pela instrução PROCESS. Um processo é concorrente com qualquer outro comando e com qualquer outro processo. Outras opções para gerar código seqüencial são os subprogramas (FUNCTION e PROCEDURE).

- As constantes podem ser declaradas e usadas em ambos os tipos de código.
- Variáveis só podem ser declaradas e usadas dentro de um código seqüencial.
- Os sinais só podem ser declarados dentro de código concorrente, mas podem ser utilizados em ambos os tipos de código.
- Os operadores podem ser usados para construir ambos os tipos de código.
- As seguintes instruções de controle de fluxo podem ser empregadas apenas em código concorrente: WHEN, SELECT e GENERATE.
- Por outro lado, as seguintes instruções de controle de fluxo podem ser empregadas apenas em código seqüencial, ou seja, dentro de PROCESS, FUNCTION ou PROCEDURE: IF, CASE, LOOP e WAIT.

2.3.3 Mecanismo genérico de simulação

- A simulação de um modelo em VHDL é baseada em eventos.
- A passagem do tempo é simulada em passos discretos, associados à ocorrência de eventos.
- Quando um novo valor é agendado para ser atribuído a um dado sinal, em um tempo futuro, diz-se que ocorreu o agendamento de uma transação sobre tal sinal.
- Ao ocorrer uma atribuição a um sinal, se o novo valor for diferente do valor anterior, diz-se que ocorreu um evento sobre tal sinal.
- Uma simulação é dividida em duas partes: a fase de inicialização e os ciclos de simulação.
- A fase de inicialização é composta pelas seguintes etapas:
  - O tempo da simulação é ajustado para o valor inicial $t = 0$ s.
  - A cada sinal declarado, é atribuído um valor inicial.
  - Para cada um dos processos declarados, é ativada uma instância e seus comandos seqüenciais são executados.
  - Usualmente, um processo contém atribuições a sinais, que agendarão transações sobre eles, em valores futuros de tempo.
  - Cada processo executa até que seja alcançado o comando WAIT, o que causa a sua suspensão.
  - Após a suspensão de todos os processos ativados, a fase de inicialização é terminada, passando-se para a execução dos ciclos de simulação.
- Os ciclos de simulação são formados pelas seguintes etapas:
  - O tempo de simulação é avançado até o próximo valor para o qual foi agendada uma transação sobre um sinal.
  - Todas as transações agendadadas para o tempo corrente são executadas.
  - As execuções das transações podem causar a ocorrência de eventos sobre sinais.
  - Todos os processos sensíveis aos sinais sobres os quais ocorreram eventos são reativados.

  - Os processos reativados executam seus comandos seqüenciais.
  - Possivelmente, os processos reativados agendarão novas transações sobre sinais
  - Cada processo executa até que seja alcançado o comando WAIT, o que causa a sua suspensão.
  - Após a suspensão de todos os processos reativados, o ciclo é repetido.
- Quando não houver mais qualquer transação agendadada, a simulação atinge seu fim.

2.4 Estrutura do código VHDL

Um código VHDL genérico, que contém a descrição de um determinado circuito, apresenta as seguintes partes:

- Declaração de bibliotecas e pacotes: que é um conjunto de declarações sobre as bibliotecas a serem consideradas e sobre os pacotes pertencentes a tais bibliotecas que deverão ser empregados.
- Entidade: que descreve a interface de acesso ao circuito, definindo a sua identificação, as suas entradas e as suas saidas.
- Arquitetura: que descreve a operação do circuito, definindo as relações entre as suas saídas e as suas entradas.

Cada uma dessas partes é discutida a seguir.

2.4.1 Bibliotecas e pacotes

- Em aplicativos de desenvolvimento, é comum que diversos elementos sejam previamente definidos, tais como: identificadores, valores constantes, nomes de variáveis e de estruturas, inicialização de variáveis e de estruturas, macros, funções e objetos.
- O objetivo em se definir previamente tais elementos é facilitar o trabalho do projetista.
- Uma vez definidos, testados e validados, tais elementos podem ser utilizados em quaisquer projetos, sem que seja necessária a sua definição a cada projeto.
- Cada aplicativo possui seus próprios padrões para a organização dos elementos previamente definidos.
- Uma organização simples e bastante utilizada são os arquivos de configuração.
- Por outro lado, uma forma mais estruturada de organização é obtida através do agrupamento de informações em um pacote (*package*) e de pacotes em uma biblioteca (*library*).
- Os compiladores VHDL normalmente consideram a inclusão automática das seguintes bibliotecas: *std* e *work*.
- A biblioteca *std* contém definições sobre os tipos básicos de dados e os correspondentes operadores. Por sua vez, a biblioteca *work* indica o diretório onde estão armazenados os arquivos do projeto.
- Nos circuitos digitais descritos em VHDL, são largamente utilizados a biblioteca padrão *ieee* e os seus pacotes *standard* e *IEEE 1164*, ambos definidos pelo IEEE.

2.4.2 Entidade

Entidade é o termo associado a um módulo de circuito em VHDL. A declaração de uma entidade descreve a interface de acesso ao circuito, definindo a sua identificação, as suas entradas e as suas saidas. As entradas e saídas são associadas aos pontos ou pinos de acesso do circuito físico e são conjuntamente denominadas de portas.

2.4.3 Arquitetura

Aspectos gerais

Arquitetura é o mecanismo utilizado em VHDL para descrever a operação de um circuito. Uma declaração de arquitetura é sempre associada a uma determinada entidade. Na arquitetura, são definidas as relações entre as saídas e as entradas da entidade a ela associada.

Um mesmo circuito digital pode ser descrito por diversas maneiras equivalentes, tais como: tabela verdade, equações genéricas diferentes, equações relacionadas a diferentes decomposições específicas, diferentes decomposições hierárquicas. Assim, dependendo do nível de abstração utilizado, uma mesma entidade pode ter seu funcionamento descrito por diversas arquiteturas diferentes.

Tipos de descrição

De uma forma geral, a descrição das operações de um circuito dentro de uma arquitetura pode assumir duas formas: comportamental ou estrutural. Em uma descrição comportamental, é feita uma descrição explícita das relações entre as saídas e as entradas. Para tal, são usados os operadores e/ou as instruções de controle de fluxo. Por sua vez, em uma descrição estrutural, é utilizado o conceito de hierarquia. Nesse caso, são utilizados módulos mais simples, previamente descritos, bem como são definidas as conexões que os interligam. Dado que uma descrição estrutural é um modelo hierárquico, os módulos que compõem o nível mais baixo e básico da hierarquia devem ser descritos de uma forma comportamental.

Instanciação de módulos

Deve-se notar que a existência de um mecanismo que possibilita a definição de um circuito em forma hierárquica permite a construção de bibliotecas de módulos e a instanciação de módulos a partir de uma determinada biblioteca.

A técnica de instanciação de módulos a partir de uma biblioteca otimiza o trabalho de descrição de circuitos complexos, bem como permite a reusabilidade de código.

Em VHDL, um módulo instanciável é denominado de componente (COMPONENT). Pode-se instanciar um módulo componente de duas formas básicas:

- O componente é declarado em um pacote, que é localizado em uma biblioteca, e é instanciado no código principal.

- O componente é declarado e instanciado no código principal.

2.5 Algumas regras sintáticas de VHDL

Comumente, as regras sintáticas das linguagens de programação são apresentadas com as notações denominadas de BNF (*Backus-Naur Form*) e EBNF (*Extended Backus-Naur Form*). Seguindo esse padrão, alguma regras sintáticas de VHDL são apresentadas a seguir.

2.5.1 Regras para biblioteca

```
library_clause <=
    LIBRARY identifier { , ... } ;

use_clause <=
    USE selected_name { , ... } ;

selected_name <=
    identifier . identifier . ( identifier | ALL )
```

Em `selected_name`, o `identifier` mais à esquerda refere-se à biblioteca, o `identifier` do meio indica o pacote e o `identifier` mais à direita aponta o item a ser utilizado.

2.5.2 Regras para pacote

```
package_declaration <=
    PACKAGE identifier IS
        { package_declarative_item}
    END [ PACKAGE ] [ identifier ] ;

package_body <=
    PACKAGE BODY identifier IS
        { package_body_declarative_item}
    END [ PACKAGE BODY ] [ identifier ] ;
```

2.5.3 Regras para entidade

```
entity_declaration <=
    ENTITY identifier IS
        [ GENERIC ( generic_interface_list ) ; ]
        [ PORT    (    port_interface_list ) ; ]
    END [ ENTITY ] [ identifier ] ;

generic_interface_list <=
    ( identifier { , ... } : subtype_indication [ := expression ] )
    { , ... }

port_interface_list <=
    ( identifier { , ... } : [ mode ] subtype_indication ) { , ... }
```

```
modet <=
    IN | OUT | INOUT
```

2.5.4 Regras para arquitetura

```
architecture_body <=
    ARCHITECTURE identifier OF entity_name IS
        { block_declarative_item }
    BEGIN
        { concurrent_statement }
    END [ ARCHITECTURE ] [ identifier ] ;

signal_declaration <=
    SIGNAL identifier { , ... } : subtype_indication [ := expression ] ;

signal_assignment_statement <=
    name <= ( value_expression [ AFTER time_expression ] ) { , ... } ;

conditional_signal_assignment <=
    name <= { waveform WHEN boolean_expression ELSE }
              waveform [ WHEN boolean_expression ] ;

selected_signal_assignment <=
    WITH expression SELECT
        name <= { waveform WHEN choices , }
                  waveform WHEN choices ;
```

2.5.5 Regras para processo

```
process_statement <=
    process_label:
    PROCESS [ ( signal_name { , ... } ) ] [ IS ]
        { process_declarative_item }
    BEGIN
        { sequential_statement }
    END PROCESS ;

wait_statement <=
    WAIT [ ON    signal_name { , ... } ]
         [ UNTIL boolean_expression    ]
         [ FOR   time_expression       ];
```

2.5.6 Regras para componente

```
component_instantiation_statement <=
    instantiation_label:
    ENTITY entity_name ( architecture_identifier )
        [ GENERIC MAP ( generic_association_list ) ]
        PORT MAP ( port_association_list ) ;

generic_association_list <=
    ( [ generic_name => ] ( expression | OPEN ) ) { , ... } ;

port_association_list <=
    ( [ port_name => ] signal_name ) { , ... } ;
```

2.6 Exemplos de declarações genéricas

2.6.1 Exemplos de biblioteca e de pacote

Um código tipicamente encontrado em arquivos VHDL, para a declaração de bibliotecas e para o uso de pacotes, é o seguinte:

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

2.6.2 Exemplos de entidade

Uma declaração genérica de entidade é a seguinte:

```
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
ENTITY __entity_name IS
  --
  GENERIC(__parameter_name : __type := __default_value;
          __parameter_name : __type := __default_value);
  --
  PORT(__input_name, __input_name   : IN    BIT;
       __input_vector_name          : IN    BIT_VECTOR(__high downto __low);
       __bidir_name,  __bidir_name  : INOUT BIT;
       __output_name, __output_name : OUT   BIT);
  --
END ENTITY __entity_name;
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
```

Deve ser ressaltado que o campo GENERIC é um conteúdo opcional.

2.6.3 Exemplos de arquitetura

Uma declaração genérica de arquitetura é a seguinte:

```
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
ARCHITECTURE __architecture_name OF __entity_name IS
  --  Declarative section
    --  TYPE
    --  CONSTANT
    --  SIGNAL
    --  COMPONENT
    --  VARIABLE
    --  FUNCTION
  --
BEGIN
  --  Code section
    --  Process Statement
    --  Concurrent Procedure Call
    --  Concurrent Signal Assignment
    --  Conditional Signal Assignment
    --  Selected Signal Assignment
    --  Component Instantiation Statement
    --  Generate Statement
  --
END ARCHITECTURE __architecture_name;
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
```

Um exemplo de declaração de arquitetura que emprega componentes (declarados no próprio código) é o seguinte:

```
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
ARCHITECTURE __architecture_name OF __entity_name IS
  --
  COMPONENT C_1 IS
    PORT(x, y : IN  BIT;
         z    : OUT BIT);
  END COMPONENT C_1;
  --
  COMPONENT C_2 IS
    PORT(x, y, z : IN  BIT;
         w       : OUT BIT);
  END COMPONENT C_2;
  --
  SIGNAL s1, s2 : BIT;
  --
BEGIN
  --
  Inst_1 : C_1 PORT MAP(x => a, y => b, z => s1)  --> Syntax #1
  --
```

```
  Inst_2 : C_1 PORT MAP(c,d,s2)  --> Syntax #2
  --
  Inst_3 : C_2 PORT MAP(e,s1,s2,f)
  --
END ARCHITECTURE __architecture_name;
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
```

2.6.4 Exemplos de processo

Uma declaração genérica de processo, comumente usada no caso de circuitos combinacionais, é a seguinte:

```
--
__process_label : PROCESS (__signal_name, __signal_name) IS
  VARIABLE __variable_name : __type;
  VARIABLE __variable_name : __type;
BEGIN
  --  Signal Assignment Statement
  --  Variable Assignment Statement
  --  Procedure Call Statement
  --  If Statement
  --  Case Statement
  --  Loop Statement
END PROCESS __process_label;
--
```

Uma declaração genérica de processo, comumente usada no caso de circuitos seqüenciais, é a seguinte:

```
--
__process_label : PROCESS IS
  VARIABLE __variable_name : __type;
  VARIABLE __variable_name : __type;
BEGIN
  WAIT UNTIL __clk_signal = __valor;
  --  Signal Assignment Statement
  --  Variable Assignment Statement
  --  Procedure Call Statement
  --  If Statement
  --  Case Statement
  --  Loop Statement
END PROCESS __process_label;
--
```

Parte II

Circuitos Combinacionais: Construções básicas

Capítulo 3

Exemplos de construções básicas

3.1 Introdução

Nesse capítulo são apresentados exemplos de construções básicas na HDL em questão. A seguir, são abordados os seguintes itens:

- Códigos para Tabela Verdade.
- Códigos para equivalentes formas de expressão lógica.
- Códigos para equivalentes formas de descrição.

3.2 Códigos para Tabela Verdade

- A Listagem 1 apresenta um exemplo de descrição de Tabela Verdade, com a mesma quantidade de valores '0' e '1'.
- A Listagem 2 apresenta um exemplo de descrição de Tabela Verdade, com quantidades diferentes de valores '0' e '1'.
- A Listagem 3 apresenta um exemplo de descrição de Tabela Verdade, com a criação de um sinal intermediário.

Listagem 1 - Exemplo de descrição de Tabela Verdade ('0' e '1' em mesma quantidade):

```
--
-- ========================================
-- Exemplo de descricao de Tabela Verdade
-- com mesma quantidade de valores '0' e '1'
-- ========================================
--
-- ============================
-- Arquivo: truth_table_00.vhd
-- ============================
--
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
ENTITY truth_table_00 IS
  PORT(a,b,c: IN  BIT;
```

```
      f    : OUT BIT);
END ENTITY truth_table_00;
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
--
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
ARCHITECTURE modelo_condicional OF truth_table_00 IS
--
BEGIN
  --  Conditional Signal Assignment
  f <= '1' WHEN (a = '0' AND b = '1' AND c = '1') ELSE
       '1' WHEN (a = '1' AND b = '0' AND c = '0') ELSE
       '1' WHEN (a = '1' AND b = '1' AND c = '0') ELSE
       '1' WHEN (a = '1' AND b = '1' AND c = '1') ELSE
       '0';
END ARCHITECTURE modelo_condicional;
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
--
-- EOF
--
```

Listagem 2 - Exemplo de descrição de Tabela Verdade (‘0’ e ‘1’ em quantidades diferentes):

```
--
-- ================================================
-- Exemplo de descricao de Tabela Verdade
-- com quantidades diferentes de valores ‘0’ e ‘1’
-- ================================================
--
-- ============================
-- Arquivo: truth_table_01.vhd
-- ============================
--
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
ENTITY truth_table_01 IS
  PORT(a,b,c: IN  BIT;
       f    : OUT BIT);
END ENTITY truth_table_01;
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
--
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
ARCHITECTURE modelo_condicional OF truth_table_01 IS
--
BEGIN
  --  Conditional Signal Assignment
  f <= '0' WHEN (a = '0' AND b = '0' AND c = '0') ELSE
       '0' WHEN (a = '1' AND b = '1' AND c = '0') ELSE
       '1';
```

```
END ARCHITECTURE modelo_condicional;
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
--
-- EOF
--
```

Listagem 3 - Exemplo de descrição de Tabela Verdade (criação de sinal intermediário):

```
--
-- ====================================
-- Exemplo de descricao de Tabela Verdade
-- com criacao de sinal intermediario
-- ====================================
--
-- ===========================
-- Arquivo: truth_table_02.vhd
-- ===========================
--
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
ENTITY truth_table_02 IS
  PORT(a,b,c: IN  BIT;
       f    : OUT BIT);
END ENTITY truth_table_02;
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
--
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
ARCHITECTURE modelo_condicional OF truth_table_02 IS
--
SIGNAL x: BIT;
--
BEGIN
  --  Conditional Signal Assignment
  x <= '0' WHEN (b = c) ELSE
       '1';
  --  Conditional Signal Assignment
  f <= '0' WHEN (a = x) ELSE
       '1';
END ARCHITECTURE modelo_condicional;
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
--
-- EOF
--
```

3.3 Códigos para equivalentes formas de expressão lógica

- As listagens abaixo descrevem a mesma função.
- A Listagem 4 apresenta a descrição da Tabela Verdade, implementando cada valor "1" separadamente.
- A Listagem 5 apresenta a implementação da forma SOP padrão.
- A Listagem 6 apresenta a implementação da forma SOP mínima.
- A Listagem 7 apresenta a implementação da forma mínima global.

Listagem 4 - Exemplo de descrição da Tabela Verdade:

```
--
-- ====================================
-- Exemplo de descricao da Tabela Verdade
-- ====================================
--
-- ============================
-- Arquivo: vhd_tv_mintermos.vhd
-- ============================
--
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
ENTITY vhd_tv_mintermos IS
  PORT(a,b,c,d: IN  BIT;
       f      : OUT BIT);
END ENTITY vhd_tv_mintermos;
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
--
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
ARCHITECTURE modelo_tv_mintermos OF vhd_tv_mintermos IS
--
BEGIN
  --  Conditional Signal Assignment
  f <= '1' WHEN (a = '0' AND b = '1' AND c = '0' AND d = '1') ELSE
       '1' WHEN (a = '0' AND b = '1' AND c = '1' AND d = '0') ELSE
       '1' WHEN (a = '1' AND b = '0' AND c = '0' AND d = '1') ELSE
       '1' WHEN (a = '1' AND b = '0' AND c = '1' AND d = '0') ELSE
       '1' WHEN (a = '1' AND b = '1' AND c = '0' AND d = '1') ELSE
       '1' WHEN (a = '1' AND b = '1' AND c = '1' AND d = '0') ELSE
       '0';
  --
END ARCHITECTURE tv_mintermos;
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
--
-- EOF
--
```

Listagem 5 - Exemplo de descrição da forma SOP padrão:

```
--
-- ======================================
-- Exemplo de descricao da forma SOP padrao
-- ======================================
--
-- ========================
-- Arquivo: vhd_std_sop.vhd
-- ========================
--
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
ENTITY vhd_std_sop IS
  PORT(a,b,c,d          : IN  BIT;
       p1,p2,p3,p4,p5,p6: OUT BIT;
       f                : OUT BIT);
END ENTITY vhd_std_sop;
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
--
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
ARCHITECTURE modelo_std_sop OF vhd_std_sop IS
--
SIGNAL s1,s2,s3,s4,s5,s6: BIT;
--
BEGIN
  --
  s1 <= (NOT a) AND      b  AND (NOT c) AND      d ;
  s2 <= (NOT a) AND      b  AND      c  AND (NOT d);
  s3 <=      a  AND (NOT b) AND (NOT c) AND      d ;
  s4 <=      a  AND (NOT b) AND      c  AND (NOT d);
  s5 <=      a  AND      b  AND (NOT c) AND (NOT d);
  s6 <=      a  AND      b  AND      c  AND (NOT d);
  --
  p1 <= s1;
  p2 <= s2;
  p3 <= s3;
  p4 <= s4;
  p5 <= s5;
  p6 <= s6;
  --
  f <= s1 OR s2 OR s3 OR s4 OR s5 OR s6;
  --
END ARCHITECTURE modelo_std_sop;
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
--
-- EOF
--
```

Listagem 6 - Exemplo de descrição da forma SOP mínima:

```
--
-- ======================================
-- Exemplo de descricao da forma SOP minima
-- ======================================
--
-- ========================
-- Arquivo: vhd_min_sop.vhd
-- ========================
--
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
ENTITY vhd_min_sop IS
  PORT(a,b,c,d    : IN  BIT;
       p1,p2,p3,p4: OUT BIT;
       f          : OUT BIT);
END ENTITY vhd_min_sop;
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
--
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
ARCHITECTURE modelo_min_sop OF vhd_min_sop IS
--
SIGNAL s1,s2,s3,s4: BIT;
--
BEGIN
  --
  s1 <= b AND (NOT c) AND      d ;
  s2 <= b AND      c  AND (NOT d);
  s3 <= a AND (NOT c) AND      d ;
  s4 <= a AND      c  AND (NOT d);
  --
  p1 <= s1;
  p2 <= s2;
  p3 <= s3;
  p4 <= s4;
  --
  f <= s1 OR s2 OR s3 OR s4;
  --
END ARCHITECTURE modelo_min_sop;
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
--
-- EOF
--
```

Listagem 7 - Exemplo de descrição da forma mínima global:

```
--
-- ========================================
-- Exemplo de descricao da forma minima global
-- ========================================
--
-- ==========================
-- Arquivo: vhd_global_min.vhd
-- ==========================
--
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
ENTITY vhd_global_min IS
  PORT(a,b,c,d: IN  BIT;
       p1,p2,p3,p4: OUT BIT;
       f      : OUT BIT);
END ENTITY vhd_global_min;
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
--
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
ARCHITECTURE modelo_global_min OF vhd_global_min IS
--
SIGNAL l1s1,l1s2,l1s3,l2s1: BIT;
--
BEGIN
  --
  l1s1 <= (NOT c) AND      d ;
  l1s2 <=      c  AND (NOT d);
  l1s3 <= a OR b;
  --
  l2s1 <= l1s1 OR l1s2;
  --
  p1 <= l1s1;
  p2 <= l1s2;
  p3 <= l1s3;
  p4 <= l2s1;
  --
  f <= l1s3 AND l2s1;
  --
END ARCHITECTURE modelo_global_min;
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
--
-- EOF
--
```

3.4 Códigos para equivalentes formas de descrição

- As listagens abaixo descrevem a mesma função.
- A Listagem 8 apresenta uma descrição usando comando condicional.
- A Listagem 9 apresenta uma descrição equivalente usando operadores.
- A Listagem 10 apresenta uma descrição equivalente usando componentes instanciados primitivos.
- A Listagem 11 apresenta uma descrição equivalente usando componentes instanciados em um mesmo arquivo.
- A Listagem 12 apresenta uma descrição equivalente usando componentes instanciados em arquivos separados.

Listagem 8 - Exemplo de descrição usando comando condicional:

```
--
-- ================================================
-- Exemplo de descricao usando comando condicional
-- ================================================
--
-- =====================
-- Arquivo: vhd_when.vhd
-- =====================
--
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
ENTITY vhd_when IS
  PORT(a,b,c,d: IN  BIT;
       f      : OUT BIT);
END ENTITY vhd_when;
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
--
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
ARCHITECTURE modelo_condicional OF vhd_when IS
--
BEGIN
  --  Conditional Signal Assignment
  f <= '1' WHEN (a = '1' AND b = '1' AND c = '0' AND d = '0') ELSE
       '1' WHEN (a = '1' AND b = '1' AND c = '0' AND d = '1') ELSE
       '1' WHEN (a = '1' AND b = '1' AND c = '1' AND d = '0') ELSE
       --
       '1' WHEN (a = '0' AND b = '0' AND c = '1' AND d = '1') ELSE
       '1' WHEN (a = '0' AND b = '1' AND c = '1' AND d = '1') ELSE
       '1' WHEN (a = '1' AND b = '0' AND c = '1' AND d = '1') ELSE
       --
       '1' WHEN (a = '1' AND b = '1' AND c = '1' AND d = '1') ELSE
       --
       '0';
```

```
  --
END ARCHITECTURE modelo_condicional;
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
--
-- EOF
--
```

Listagem 9 - Exemplo de descrição usando operadores:

```
--
-- ====================================
-- Exemplo de descricao usando operadores
-- ====================================
--
-- =====================
-- Arquivo: vhd_op.vhd
-- =====================
--
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
ENTITY vhd_op IS
  PORT(a,b,c,d: IN  BIT;
       f      : OUT BIT);
END ENTITY vhd_op;
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
--
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
ARCHITECTURE modelo_operador OF vhd_op IS
--
SIGNAL sab, scd: BIT;
--
BEGIN
  --  Concurrent Signal Assignment
  --
  sab <= a AND b;
  scd <= c AND d;
  --
  f <= sab OR scd;
  --
END ARCHITECTURE modelo_operador;
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
--
-- EOF
--
```

Listagem 10 - Exemplo de descrição usando componentes instanciados primitivos:

```
--
-- ====================================================
-- Exemplo de descricao usando componentes instanciados
-- primitivos
-- ====================================================
--
--
-- ================================
-- Arquivo: vhd_estrut_primitive.vhd
-- ================================
--
ENTITY vhd_estrut_primitive IS
  PORT(a,b,c,d: IN  BIT;
       f      : OUT BIT);
END ENTITY vhd_estrut_primitive;
--
--
ARCHITECTURE modelo_componente OF vhd_estrut_primitive IS
--
SIGNAL sab, scd: BIT;
--
BEGIN
  --  Primitive Component Instantiation Statement
  --
  sab <= a AND b;
  scd <= c AND d;
  --
  f   <= sab OR scd;
  --
END ARCHITECTURE modelo_componente;
--
-- EOF
--
```

Listagem 11 - Exemplo de descrição usando componentes instanciados em um mesmo arquivo:

```
--
-- ================================================
-- Exemplo de descricao usando componentes instanciados
-- em um mesmo arquivo
-- ================================================
--
--
-- ================================
-- Arquivo: vhd_estrut_same_file.vhd
-- ================================
--
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
ENTITY vhd_estrut_same_file IS
  PORT(a,b,c,d: IN  BIT;
       f      : OUT BIT);
END ENTITY vhd_estrut_same_file;
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
--
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
ARCHITECTURE modelo_componente OF vhd_estrut_same_file IS
--
COMPONENT vhd_and2_component IS
  PORT(x,y: IN  BIT;
       z  : OUT BIT);
END COMPONENT vhd_and2_component ;
--
--
COMPONENT vhd_or2_component IS
  PORT(x,y: IN  BIT;
       z  : OUT BIT);
END COMPONENT vhd_or2_component ;
--
--
SIGNAL sab, scd: BIT;
--
BEGIN
  --  Component Instantiation Statement
  --
  AND1: vhd_and2_component PORT MAP(a,b,sab);
  AND2: vhd_and2_component PORT MAP(c,d,scd);
  --
  OR1 : vhd_or2_component PORT MAP(sab,scd,f);
  --
END ARCHITECTURE modelo_componente;
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
--
--
```

```
-- ========================
-- Definicao dos componentes
-- ========================
--
-- ========================
--
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
ENTITY vhd_and2_component IS
  PORT(x,y: IN  BIT;
       z  : OUT BIT);
END ENTITY vhd_and2_component;
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
--
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
ARCHITECTURE modelo_op OF vhd_and2_component IS
--
BEGIN
  --
  z <= x AND y;
  --
END ARCHITECTURE modelo_op;
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
--
-- ========================
--
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
ENTITY vhd_or2_component IS
  PORT(x,y: IN  BIT;
       z  : OUT BIT);
END ENTITY vhd_or2_component;
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
--
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
ARCHITECTURE modelo_op OF vhd_or2_component IS
--
BEGIN
  --
  z <= x OR y;
  --
END ARCHITECTURE modelo_op;
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
--
-- ========================
--
-- EOF
--
```

Listagem 12 - Exemplo de descrição usando componentes instanciados em arquivos separados:

```
--
-- ================================================
-- Exemplo de descricao usando componentes instanciados
-- em arquivos separados
-- ================================================
--
-- ====================================
-- Arquivo: vhd_estrut_not_same_file.vhd
-- ====================================
--
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
ENTITY vhd_estrut_not_same_file IS
  PORT(a,b,c,d: IN  BIT;
       f      : OUT BIT);
END ENTITY vhd_estrut_not_same_file;
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
--
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
ARCHITECTURE modelo_componente OF vhd_estrut_not_same_file IS
--
COMPONENT vhd_and2_component IS
  PORT(x,y: IN  BIT;
       z  : OUT BIT);
END COMPONENT vhd_and2_component ;
--
--
COMPONENT vhd_or2_component IS
  PORT(x,y: IN  BIT;
       z  : OUT BIT);
END COMPONENT vhd_or2_component ;
--
--
SIGNAL sab, scd: BIT;
--
BEGIN
  --  Component Instantiation Statement
  --
  AND1: vhd_and2_component PORT MAP(a,b,sab);
  AND2: vhd_and2_component PORT MAP(c,d,scd);
  --
  OR1 : vhd_or2_component PORT MAP(sab,scd,f);
  --
END ARCHITECTURE modelo_componente;
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
--
-- EOF
--
```

```
--
-- ================================
-- Arquivo: vhd_and2_component.vhd
-- ================================
--
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
ENTITY vhd_and2_component IS
  PORT(x,y: IN  BIT;
       z  : OUT BIT);
END ENTITY vhd_and2_component;
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
--
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
ARCHITECTURE modelo_op OF vhd_and2_component IS
--
BEGIN
  --
  z <= x AND y;
  --
END ARCHITECTURE modelo_op;
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
--
-- EOF
--
--
-- ================================
-- Arquivo: vhd_or2_component.vhd
-- ================================
--
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
ENTITY vhd_or2_component IS
  PORT(x,y: IN  BIT;
       z  : OUT BIT);
END ENTITY vhd_or2_component;
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
--
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
ARCHITECTURE modelo_op OF vhd_or2_component IS
--
BEGIN
  --
  z <= x OR y;
  --
END ARCHITECTURE modelo_op;
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
--
-- EOF
--
```

Capítulo 4

Exemplos de aplicação direta de portas lógicas

4.1 Introdução

Nesse capítulo são apresentados exemplos de circuitos digitais simples, com o emprego direto de uma porta lógica. A seguir, são abordados os seguintes itens:

- Elemento de controle.
- Elemento de paridade.
- Elemento de igualdade.

4.2 Elemento de controle

- A Listagem 1 apresenta um bloco de controle de interrupção.
- A Listagem 2 apresenta um elemento inversor configurável.

Listagem 1 - Bloco de controle de interrupção:

```
--
-- ======================================================
-- Exemplo de descricao de bloco de controle de interrupcao
-- ======================================================
--
-- ================================
-- Arquivo: vhd_4pins_int_ctrl.vhd
-- ================================
--
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
ENTITY vhd_4pins_int_ctrl IS
  PORT(NMI_pin : IN BIT;
       MI1_pin , MI2_pin , MI3_pin : IN BIT;
       --
       IMbit1, IMbit2, IMbit3 : IN BIT;
       --
```

```
       NMI_req : OUT BIT;
       MI1_req , MI2_req , MI3_req : OUT BIT);
END ENTITY vhd_4pins_int_ctrl;
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
--
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
ARCHITECTURE modelo_condicional OF vhd_4pins_int_ctrl IS
--
BEGIN
  --  Signal Assignment Statement
  NMI_req <= NMI_pin;
  MI1_req <= MI1_pin AND IMbit1;
  MI2_req <= MI2_pin AND IMbit2;
  MI3_req <= MI3_pin AND IMbit3;
  --
END ARCHITECTURE modelo_condicional;
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
--
-- EOF
--
```

Listagem 2 - Elemento inversor configurável:

```
--
-- ======================================================
-- Exemplo de descricao de elemento inversor configuravel
-- ======================================================
--
-- ============================
-- Arquivo: vhd_config_not.vhd
-- ============================
--
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
ENTITY vhd_config_not IS
  PORT(cfg_ctrl   : IN  BIT;
       cfg_not_in : IN  BIT;
       cfg_not_out: OUT BIT);
END ENTITY vhd_config_not;
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
--
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
ARCHITECTURE modelo_condicional OF vhd_config_not IS
--
BEGIN
  --  Signal Assignment Statement
  cfg_not_out <= cfg_not_in XOR cfg_ctrl;
  --
```

```
END ARCHITECTURE modelo_condicional;
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
--
-- EOF
--
```

4.3 Elemento de paridade

- A Listagem 3 apresenta a implementação de um gerador de paridade par e ímpar.
- A Listagem 4 apresenta a implementação de um identificador de paridade par e ímpar.

Listagem 3 - Exemplo de descrição de gerador de paridade par e ímpar:

```
--
-- =====================================================
-- Exemplo de descricao de gerador de paridade par e impar
-- =====================================================
--
-- ============================
-- Arquivo: vhd_parity_gen.vhd
-- ============================
--
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
ENTITY vhd_parity_gen IS
  PORT(data_bit1 , data_bit2 : IN  BIT;
       odd_parity_bit , even_parity_bit : OUT BIT);
END ENTITY vhd_parity_gen;
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
--
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
ARCHITECTURE modelo_condicional OF vhd_parity_gen IS
--
BEGIN
  --  Signal Assignment Statement
  --
  odd_parity_bit  <= data_bit1 XNOR data_bit2;
  --
  even_parity_bit <= data_bit1  XOR data_bit2;
  --
END ARCHITECTURE modelo_condicional;
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
--
-- EOF
--
```

Listagem 4 - Exemplo de descrição de identificador de paridade par e ímpar:

```
--
-- ============================================================
-- Exemplo de descricao de identificador de paridade par e impar
-- ============================================================
--
-- ============================
-- Arquivo: vhd_parity_idf.vhd
-- ============================
--
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
ENTITY vhd_parity_idf IS
  PORT(data_bit, parity_bit : IN  BIT;
       odd_parity_bit , even_parity_bit : OUT BIT);
END ENTITY vhd_parity_idf;
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
--
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
ARCHITECTURE modelo_condicional OF vhd_parity_idf IS
--
BEGIN
  --  Signal Assignment Statement
  --
  odd_parity_bit  <= data_bit  XOR parity_bit;
  --
  even_parity_bit <= data_bit XNOR parity_bit;
  --
END ARCHITECTURE modelo_condicional;
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
--
-- EOF
--
```

4.4 Elemento de igualdade

- A Listagem 5 apresenta a implementação de um identificador de igualdade entre padrões binários.

Listagem 5 - Exemplo de descrição de identificador de igualdade:

```
--
-- ================================================
-- Exemplo de descricao de identificador de igualdade
-- ================================================
--
-- ================================
-- Arquivo: vhd_1bit_equal_idf.vhd
-- ================================
--
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
ENTITY vhd_1bit_equal_idf IS
  PORT(data1, data2   : IN  BIT;
       equal_data     : OUT BIT;
       not_equal_data : OUT BIT);
END ENTITY vhd_1bit_equal_idf;
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
--
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
ARCHITECTURE modelo_condicional OF vhd_1bit_equal_idf IS
--
BEGIN
  --  Signal Assignment Statement
  --
  equal_data     <= data1 XNOR data2;
  --
  not_equal_data <= data1  XOR data2;
  --
END ARCHITECTURE modelo_condicional;
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
--
-- EOF
--
```

Parte III

Circuitos Combinacionais: Blocos funcionais

Capítulo 5

Exemplos de blocos funcionais

Nessa parte, são apresentados exemplos de circuitos digitais que implementam funções comumente encontradas em sistemas digitais.

A seguir, são abordados os seguintes itens:

- Detector de igualdade entre dois grupos de *bits*.
- Selecionadores:
  - Multiplexador (MUX).
  - Demultiplexador (DEMUX).
  - Decodificador de endereço (*address decoder*) ou decodificador de linha (*line decoder*).
- Deslocador configurável (*barrel shifter*).
- Decodificador.
- Conversor de códigos.
- Separador e relacionador de *bits* 1 e 0 em palavra de N *bits*.
- Somadores básicos.
- Detector de *overflow* e saturador.

Capítulo 6

Detector de igualdade entre dois grupos de *bits*

6.1 Detector de igualdade para um *bit*

- A Listagem 1 apresenta um detector de igualdade para um *bit*, baseado em Tabela Verdade.
- A Listagem 2 apresenta um detector de igualdade para um *bit*, baseado na comparação direta.
- A Listagem 3 apresenta um detector de igualdade para um *bit*, baseado em operador lógico.

Listagem 1 - Detector de igualdade para um *bit*, baseado em Tabela Verdade:

```
--
-- ============================================================
-- Detector de igualdade para um bit, baseado em Tabela Verdade
-- ============================================================
--
-- ==================================
-- Arquivo: vhd_tt_1bit_x_equal_y.vhd
-- ==================================
--
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
ENTITY vhd_tt_1bit_x_equal_y IS
  PORT(nbr1_k , nbr2_k : IN  BIT;
       eq_inp_k        : IN  BIT;
       eq_out_k        : OUT BIT);
END ENTITY vhd_tt_1bit_x_equal_y;
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
--
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
ARCHITECTURE tt_model OF vhd_tt_1bit_x_equal_y IS
--
BEGIN
  --
```

```
  eq_out_k <=
     '1' WHEN (eq_inp_k = '1' AND nbr1_k = '0' AND nbr2_k = '0') ELSE
     '1' WHEN (eq_inp_k = '1' AND nbr1_k = '1' AND nbr2_k = '1') ELSE
     --
     '0';
  --
END ARCHITECTURE tt_model;
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
--
-- EOF
--
```

Listagem 2 - Detector de igualdade para um *bit*, baseado na comparação direta:

```
--
-- ================================================================
-- Detector de igualdade para um bit, baseado na comparacao direta
-- ================================================================
--
-- ==================================
-- Arquivo: vhd_eq_1bit_x_equal_y.vhd
-- ==================================
--
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
ENTITY vhd_eq_1bit_x_equal_y IS
  PORT(nbr1_k , nbr2_k : IN  BIT;
       eq_inp_k        : IN  BIT;
       eq_out_k        : OUT BIT);
END ENTITY vhd_eq_1bit_x_equal_y;
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
--
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
ARCHITECTURE eq_model OF vhd_eq_1bit_x_equal_y IS
--
BEGIN
  --
  eq_out_k <= '1' WHEN (eq_inp_k = '1' AND nbr1_k = nbr2_k) ELSE
              --
              '0';
  --
END ARCHITECTURE eq_model;
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
--
-- EOF
--
```

Listagem 3 - Detector de igualdade para um *bit*, baseado em operador lógico:

```
--
-- ============================================================
-- Detector de igualdade para um bit, baseado em operador logico
-- ============================================================
--
-- ==================================
-- Arquivo: vhd_op_1bit_x_equal_y.vhd
-- ==================================
--
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
ENTITY vhd_op_1bit_x_equal_y IS
  PORT(nbr1_k , nbr2_k : IN  BIT;
       eq_inp_k        : IN  BIT;
       eq_out_k        : OUT BIT);
END ENTITY vhd_op_1bit_x_equal_y;
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
--
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
ARCHITECTURE eq_model OF vhd_op_1bit_x_equal_y IS
--
BEGIN
  --
  eq_out_k <= eq_inp_k AND (nbr1_k XNOR nbr2_k);
  --
END ARCHITECTURE eq_model;
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
--
-- EOF
--
```

6.2 Detector de igualdade para quatro *bits*

- A Listagem 4 apresenta um detector de igualdade para quatro *bits*, baseado em operador lógico.
- A Listagem 5 apresenta um detector de igualdade para dezesseis *bits*, empregando o detector de quatro *bits* da Listagem 4.

Listagem 4 - Detector de igualdade para quatro *bits*, baseado baseado em operador lógico:

```
--
-- ====================================================================
-- Detector de igualdade para quatro bits, baseado em operador logico
-- ====================================================================
--
-- ================================
-- Arquivo: vhd_4bit_x_equal_y.vhd
-- ================================
--
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
ENTITY vhd_4bit_x_equal_y IS
  PORT(nbr1, nbr2 : IN  BIT_VECTOR(3 DOWNTO 0);
       eq_inp     : IN  BIT;
       eq_out     : OUT BIT);
END ENTITY vhd_4bit_x_equal_y;
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
--
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
ARCHITECTURE eq_model OF vhd_4bit_x_equal_y IS
--
SIGNAL eq0, eq1, eq2, eq3: BIT;
--
BEGIN
  --
  eq3 <= nbr1(3) XNOR nbr2(3);
  eq2 <= nbr1(2) XNOR nbr2(2);
  eq1 <= nbr1(1) XNOR nbr2(1);
  eq0 <= nbr1(0) XNOR nbr2(0);
  --
  eq_out <= eq_inp AND eq0 AND eq1 AND eq2 AND eq3;
  --
END ARCHITECTURE eq_model;
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
--
-- EOF
--
```

Listagem 5 - Detector de igualdade para dezesseis *bits*, empregando o detector de quatro *bits*:

```
--
-- ========================================
-- Detector de igualdade para dezesseis bits,
--    empregando o detector de quatro bits
-- ========================================
--
```

```
-- ================================
-- Arquivo: vhd_4x4bit_x_equal_y.vhd
-- ================================
--
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
ENTITY vhd_4x4bit_x_equal_y IS
  PORT(nbr1, nbr2 : IN  BIT_VECTOR(15 DOWNTO 0);
       eq_inp     : IN  BIT;
       eq_out     : OUT BIT);
END ENTITY vhd_4x4bit_x_equal_y;
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
--
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
ARCHITECTURE eq_component OF vhd_4x4bit_x_equal_y IS
--
COMPONENT vhd_4bit_x_equal_y IS
  PORT(nbr1, nbr2 : IN  BIT_VECTOR(3 DOWNTO 0);
       eq_inp     : IN  BIT;
       eq_out     : OUT BIT);
END COMPONENT vhd_4bit_x_equal_y;
--
SIGNAL eq_1_0,eq_2_1,eq_3_2: BIT;
--
BEGIN
  --
  Eq4bits_3: vhd_4bit_x_equal_y
     PORT MAP(nbr1(15 DOWNTO 12),nbr2(15 DOWNTO 12),eq_inp,eq_3_2);
  --
  Eq4bits_2: vhd_4bit_x_equal_y
     PORT MAP(nbr1(11 DOWNTO  8),nbr2(11 DOWNTO  8),eq_3_2,eq_2_1);
  --
  Eq4bits_1: vhd_4bit_x_equal_y
     PORT MAP(nbr1( 7 DOWNTO  4),nbr2( 7 DOWNTO  4),eq_2_1,eq_1_0);
  --
  Eq4bits_0: vhd_4bit_x_equal_y
     PORT MAP(nbr1( 3 DOWNTO  0),nbr2( 3 DOWNTO  0),eq_1_0,eq_out);
  --
END ARCHITECTURE eq_component;
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
--
-- EOF
--
```

6.3 Detectores de igualdade baseados em detector para um *bit*

- A Listagem 6 apresenta um detector de igualdade para quatro *bits*, empregando o detector de um *bit* da Listagem 3.
- A Listagem 7 apresenta um detector de igualdade para oito *bits*, empregando o detector de um *bit* da Listagem 3.

Listagem 6 - Detector de igualdade para quatro *bits*, empregando o detector de um *bit*:

```
--
-- =====================================
-- Detector de igualdade para quatro bits,
--     empregando o detector de um bit
-- =====================================
--
-- ================================
-- Arquivo: vhd_4x1bit_x_equal_y.vhd
-- ================================
--
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
ENTITY vhd_4x1bit_x_equal_y IS
  PORT(nbr1, nbr2 : IN  BIT_VECTOR(3 DOWNTO 0);
       eq_inp     : IN  BIT;
       eq_out     : OUT BIT);
END ENTITY vhd_4x1bit_x_equal_y;
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
--
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
ARCHITECTURE eq_component OF vhd_4x1bit_x_equal_y IS
--
COMPONENT vhd_op_1bit_x_equal_y IS
  PORT(nbr1_k , nbr2_k : IN  BIT;
       eq_inp_k        : IN  BIT;
       eq_out_k        : OUT BIT);
END COMPONENT vhd_op_1bit_x_equal_y;
--
SIGNAL eq_1_0,eq_2_1,eq_3_2: BIT;
--
BEGIN
  --
  Eq1bit_3: vhd_op_1bit_x_equal_y
     PORT MAP(nbr1(3),nbr2(3),eq_inp,eq_3_2);
  --
  Eq1bit_2: vhd_op_1bit_x_equal_y
     PORT MAP(nbr1(2),nbr2(2),eq_3_2,eq_2_1);
  --
  Eq1bit_1: vhd_op_1bit_x_equal_y
```

```
     PORT MAP(nbr1(1),nbr2(1),eq_2_1,eq_1_0);
  --
  Eq1bit_0: vhd_op_1bit_x_equal_y
     PORT MAP(nbr1(0),nbr2(0),eq_1_0,eq_out);
  --
END ARCHITECTURE eq_component;
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
--
-- EOF
--
```

Listagem 7 - Detector de igualdade para oito *bits*, empregando o detector de um *bit*:

```
--
-- ====================================
-- Detector de igualdade para oito bits,
--    empregando o detector de um bit
-- ====================================
--
-- ====================================
-- Arquivo: vhd_8x1bit_gen_x_equal_y.vhd
-- ====================================
--
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
ENTITY vhd_8x1bit_gen_x_equal_y IS
  GENERIC(nob: INTEGER := 8);   -- Number Of Bits
  --
  PORT(nbr1, nbr2 : IN  BIT_VECTOR((nob-1) DOWNTO 0);
       eq_inp     : IN  BIT;
       eq_out     : OUT BIT);
END ENTITY vhd_8x1bit_gen_x_equal_y;
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
--
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
ARCHITECTURE eq_gen_component OF vhd_8x1bit_gen_x_equal_y IS
--
COMPONENT vhd_op_1bit_x_equal_y IS
  PORT(nbr1_k , nbr2_k : IN  BIT;
       eq_inp_k        : IN  BIT;
       eq_out_k        : OUT BIT);
END COMPONENT vhd_op_1bit_x_equal_y;
--
SIGNAL internal_eq: BIT_VECTOR(nob DOWNTO 0);
--
BEGIN
  --
  internal_eq(nob) <= eq_inp;
```

```
  --
  gen_eq: FOR ind IN (nob-1) DOWNTO 0 GENERATE
          BEGIN
            Eq1bit: vhd_op_1bit_x_equal_y
               PORT MAP(nbr1(ind), nbr2(ind),
                        internal_eq(ind+1), internal_eq(ind));
          END GENERATE;
  --
  eq_out <= internal_eq(0);
  --
END ARCHITECTURE eq_gen_component;
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
--
-- EOF
--
```

Capítulo 7

Selecionadores: MUX, DEMUX e *address decoder*

7.1 Introdução

Nesse capítulo, são apresentados códigos para três blocos funcionais relacionados entre si:

- Multiplexador (MUX).
- Demultiplexador (DEMUX).
- Decodificador de endereço (*address decoder*) ou decodificador de linha (*line decoder*).

Todos eles representam um tipo de selecionador.

7.2 Multiplexador (MUX)

A seguir, são apresentados códigos para multiplexadores com N entradas de B *bits*, denominados de MUX NxB.

7.2.1 MUX 2x1

- A Listagem 1 apresenta um MUX 2x1, baseado em operador lógico.

Listagem 1 - MUX 2x1, baseado em operador lógico:

```
--
-- ====================================
-- MUX 2x1, baseado em operador logico
-- ====================================
--
-- ============================
-- Arquivo: vhd_op_mux_2x1.vhd
-- ============================
--
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
ENTITY vhd_op_mux_2x1 IS
  PORT(mux_in0, mux_in1 : IN  BIT;
```

```
       sel0             : IN  BIT;
       mux_out          : OUT BIT);
END ENTITY vhd_op_mux_2x1;
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
--
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
ARCHITECTURE op_model OF vhd_op_mux_2x1 IS
--
BEGIN
  --
  mux_out <= ( (NOT sel0) AND mux_in0 )  OR
             (      sel0  AND mux_in1 )  ;
  --
END ARCHITECTURE op_model;
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
--
-- EOF
--
```

7.2.2 MUX 4x1

- A Listagem 2 apresenta um MUX 4x1, empregando o MUX 2x1 da Listagem 1.
- A Listagem 3 apresenta um MUX 4x1, baseado em operador lógico.

Listagem 2 - MUX 4x1, empregando o MUX 2x1:

```
--
-- ============================
-- MUX 4x1, empregando o MUX 2x1
-- ============================
--
-- ============================
-- Arquivo: vhd_hier_mux_4x1.vhd
-- ============================
--
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
ENTITY vhd_hier_mux_4x1 IS
  PORT(mux_in0, mux_in1, mux_in2, mux_in3 : IN  BIT;
       sel0,sel1                          : IN  BIT;
       mux_out                            : OUT BIT);
END ENTITY vhd_hier_mux_4x1;
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
--
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
ARCHITECTURE hier_model OF vhd_hier_mux_4x1 IS
--
```

```
COMPONENT vhd_op_mux_2x1 IS
  PORT(mux_in0, mux_in1 : IN  BIT;
       sel0             : IN  BIT;
       mux_out          : OUT BIT);
END COMPONENT vhd_op_mux_2x1;
--
SIGNAL int_out_01,int_out_23: BIT;
--
BEGIN
  --
  mux_23: vhd_op_mux_2x1
          PORT MAP(mux_in2,mux_in3,sel0,int_out_23);
  --
  mux_01: vhd_op_mux_2x1
          PORT MAP(mux_in0,mux_in1,sel0,int_out_01);
  --
  --
  mux_LH: vhd_op_mux_2x1
          PORT MAP(int_out_01,int_out_23,sel1,mux_out);
  --
END ARCHITECTURE hier_model;
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
--
-- EOF
--
```

Listagem 3 - MUX 4x1, baseado em operador lógico:

```
--
-- ====================================
-- MUX 4x1, baseado em operador logico
-- ====================================
--
-- ============================
-- Arquivo: vhd_op_mux_4x1.vhd
-- ============================
--
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
ENTITY vhd_op_mux_4x1 IS
  PORT(mux_in0, mux_in1, mux_in2, mux_in3 : IN  BIT;
       sel0,sel1                          : IN  BIT;
       mux_out                            : OUT BIT);
END ENTITY vhd_op_mux_4x1;
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
--
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
ARCHITECTURE op_model OF vhd_op_mux_4x1 IS
```

```
--
BEGIN
  --
  mux_out <= ( (NOT sel1) AND (NOT sel0) AND mux_in0 )  OR
              ( (NOT sel1) AND      sel0  AND mux_in1 )  OR
              (      sel1  AND (NOT sel0) AND mux_in2 )  OR
              (      sel1  AND      sel0  AND mux_in3 )  ;
  --
END ARCHITECTURE op_model;
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
--
-- EOF
--
```

7.2.3 MUX 8x1

- A Listagem 4 apresenta um MUX 8x1, empregando comando condicional.

Listagem 4 - MUX 8x1, empregando comando condicional:

```
--
-- ================================================
-- MUX 8x1, empregando comando condicional de VHDL
-- ================================================
--
-- ==============================
-- Arquivo: vhd_when_mux_8x1.vhd
-- ==============================
--
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
ENTITY vhd_when_mux_8x1 IS
  GENERIC(-- Number Of Selectors
          nos: INTEGER := 3;
          -- Number Of Inputs
          noi: INTEGER := 8);
  --
  PORT(mux_in : IN  BIT_VECTOR((noi-1) DOWNTO 0);
       sel    : IN  BIT_VECTOR((nos-1) DOWNTO 0);
       mux_out: OUT BIT);
END ENTITY vhd_when_mux_8x1;
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
--
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
ARCHITECTURE when_model OF vhd_when_mux_8x1 IS
--
BEGIN
  --
```

```
  mux_out <= mux_in(0) WHEN sel = "000" ELSE
             mux_in(1) WHEN sel = "001" ELSE
             mux_in(2) WHEN sel = "010" ELSE
             mux_in(3) WHEN sel = "011" ELSE
             mux_in(4) WHEN sel = "100" ELSE
             mux_in(5) WHEN sel = "101" ELSE
             mux_in(6) WHEN sel = "110" ELSE
             mux_in(7) WHEN sel = "111" ;
  --
END ARCHITECTURE when_model;
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
--
-- EOF
--
```

7.3 Demultiplexador (DEMUX)

A seguir, são apresentados códigos para demultiplexadores com N saídas de B *bits*, denominados de DEMUX NxB.

7.3.1 DEMUX 2x1

- A Listagem 5 apresenta um DEMUX 2x1, baseado em operador lógico.

Listagem 5 - DEMUX 2x1, baseado em operador lógico:

```
--
-- ====================================
-- DEMUX 2x1, baseado em operador logico
-- ====================================
--
-- ============================
-- Arquivo: vhd_op_demux_2x1.vhd
-- ============================
--
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
ENTITY vhd_op_demux_2x1 IS
  PORT(demux_in               : IN  BIT;
       sel0                   : IN  BIT;
       demux_out0, demux_out1 : OUT BIT);
END ENTITY vhd_op_demux_2x1;
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
--
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
ARCHITECTURE op_model OF vhd_op_demux_2x1 IS
--
```

```
BEGIN
  --
  demux_out0 <= ( (NOT sel0) AND demux_in );
  demux_out1 <= (      sel0  AND demux_in );
  --
END ARCHITECTURE op_model;
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
--
-- EOF
--
```

7.3.2 DEMUX 4x1

- A Listagem 6 apresenta um DEMUX 4x1, empregando o DEMUX 2x1 da Listagem 5.
- A Listagem 7 apresenta um DEMUX 4x1, baseado em operador lógico.

Listagem 6 - DEMUX 4x1, empregando o DEMUX 2x1:

```
--
-- ================================
-- DEMUX 4x1, empregando o DEMUX 2x1
-- ================================
--
-- ==============================
-- Arquivo: vhd_hier_demux_4x1.vhd
-- ==============================
--
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
ENTITY vhd_hier_demux_4x1 IS
  PORT(demux_in                : IN  BIT;
       sel0, sel1              : IN  BIT;
       demux_out0, demux_out1,
       demux_out2, demux_out3  : OUT BIT);
END ENTITY vhd_hier_demux_4x1;
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
--
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
ARCHITECTURE hier_model OF vhd_hier_demux_4x1 IS
--
COMPONENT vhd_op_demux_2x1 IS
  PORT(demux_in               : IN  BIT;
       sel0                   : IN  BIT;
       demux_out0, demux_out1 : OUT BIT);
END COMPONENT vhd_op_demux_2x1;
--
SIGNAL int_out_01,int_out_23: BIT;
```

```
--
BEGIN
  --
  demux_LH: vhd_op_demux_2x1
            PORT MAP(demux_in,sel1,int_out_01,int_out_23);
  --
  --
  demux_23: vhd_op_demux_2x1
            PORT MAP(int_out_23,sel0,demux_out2,demux_out3);
  --
  demux_01: vhd_op_demux_2x1
            PORT MAP(int_out_01,sel0,demux_out0,demux_out1);
  --
END ARCHITECTURE hier_model;
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
--
-- EOF
--
```

Listagem 7 - DEMUX 4x1, baseado em operador lógico:

```
--
-- ====================================
-- DEMUX 4x1, baseado em operador logico
-- ====================================
--
-- ============================
-- Arquivo: vhd_op_demux_4x1.vhd
-- ============================
--
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
ENTITY vhd_op_demux_4x1 IS
  PORT(demux_in                : IN  BIT;
       sel0, sel1              : IN  BIT;
       demux_out0, demux_out1,
       demux_out2, demux_out3  : OUT BIT);
END ENTITY vhd_op_demux_4x1;
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
--
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
ARCHITECTURE op_model OF vhd_op_demux_4x1 IS
--
BEGIN
  --
  demux_out0 <= ( (NOT sel1) AND (NOT sel0) AND demux_in );
  demux_out1 <= ( (NOT sel1) AND      sel0  AND demux_in );
  demux_out2 <= (      sel1  AND (NOT sel0) AND demux_in );
```

```
  demux_out3 <= (      sel1  AND      sel0  AND demux_in );
  --
END ARCHITECTURE op_model;
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
--
-- EOF
--
```

7.3.3 DEMUX 8x1

- A Listagem 8 apresenta um DEMUX 8x1, empregando comando condicional.

Listagem 8 - DEMUX 8x1, empregando comando condicional:

```
--
-- ================================================
-- DEMUX 8x1, empregando comando condicional de VHDL
-- ================================================
--
-- ================================
-- Arquivo: vhd_when_demux_8x1.vhd
-- ================================
--
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
ENTITY vhd_when_demux_8x1 IS
  GENERIC(-- Number Of Selectors
          nos: INTEGER := 3;
          -- Number Of Outputs
          noo: INTEGER := 8);
  --
  PORT(demux_in : IN  BIT;
       sel      : IN  BIT_VECTOR((nos-1) DOWNTO 0);
       demux_out: OUT BIT_VECTOR((noo-1) DOWNTO 0) );
END ENTITY vhd_when_demux_8x1;
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
--
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
ARCHITECTURE when_model OF vhd_when_demux_8x1 IS
--
BEGIN
  --
  demux_out(0) <= demux_in WHEN sel = "000" ELSE '0';
  demux_out(1) <= demux_in WHEN sel = "001" ELSE '0';
  demux_out(2) <= demux_in WHEN sel = "010" ELSE '0';
  demux_out(3) <= demux_in WHEN sel = "011" ELSE '0';
  demux_out(4) <= demux_in WHEN sel = "100" ELSE '0';
  demux_out(5) <= demux_in WHEN sel = "101" ELSE '0';
```

```
  demux_out(6) <= demux_in WHEN sel = "110" ELSE '0';
  demux_out(7) <= demux_in WHEN sel = "111" ELSE '0';
  --
END ARCHITECTURE when_model;
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
--
-- EOF
--
```

7.4 Decodificador de endereços (*address decoder*)

A seguir, são apresentados códigos para decodificadores de endereços com B *bits* e $N = 2^B$ saídas de 1 *bit*, denominados de *address decoder* BxN ou *line decoder* BxN.

7.4.1 *Address decoder* 1x2

- A Listagem 9 apresenta um *address decoder* 1x2, baseado em operador lógico.
- A Listagem 10 apresenta um *address decoder* 1x2, com *enable*, baseado em operador lógico.

Listagem 9 - *Address decoder* 1x2, baseado em operador lógico:

```
--
-- ============================================
-- Address decoder 1x2, baseado em operador logico
-- ============================================
--
-- ======================================
-- Arquivo: vhd_op_address_decoder_1x2.vhd
-- ======================================
--
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
ENTITY vhd_op_address_decoder_1x2 IS
  PORT(
       sel0                  : IN  BIT;
       address_decoder_out0,
       address_decoder_out1  : OUT BIT);
END ENTITY vhd_op_address_decoder_1x2;
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
--
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
ARCHITECTURE op_model OF vhd_op_address_decoder_1x2 IS
--
BEGIN
  --
  address_decoder_out0 <= (NOT sel0);
```

```
  address_decoder_out1 <=      sel0 ;
  --
END ARCHITECTURE op_model;
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
--
-- EOF
--
```

Listagem 10 - *Address decoder* 1x2, com *enable*, baseado em operador lógico:

```
--
-- ============================================================
-- Address decoder 1x2, com enable, baseado em operador logico
-- ============================================================
--
-- ==========================================
-- Arquivo: vhd_op_address_decoder_1x2_ena.vhd
-- ==========================================
--
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
ENTITY vhd_op_address_decoder_1x2_ena IS
  PORT(
       ena                   : IN  BIT;
       sel0                  : IN  BIT;
       address_decoder_out0,
       address_decoder_out1  : OUT BIT);
END ENTITY vhd_op_address_decoder_1x2_ena;
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
--
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
ARCHITECTURE op_model OF vhd_op_address_decoder_1x2_ena IS
--
BEGIN
  --
  address_decoder_out0 <= (NOT sel0) AND ena;
  address_decoder_out1 <= (    sel0) AND ena;
  --
END ARCHITECTURE op_model;
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
--
-- EOF
--
```

7.4.2 *Address decoder* 2x4

- A Listagem 11 apresenta um *address decoder* 2x4, baseado em operador lógico.
- A Listagem 12 apresenta um *address decoder* 2x4, com *enable*, baseado em operador lógico.

Listagem 11 - *Address decoder* 2x4, baseado em operador lógico:

```
--
-- ============================================
-- Address decoder 2x4, baseado em operador logico
-- ============================================
--
-- ======================================
-- Arquivo: vhd_op_address_decoder_2x4.vhd
-- ======================================
--
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
ENTITY vhd_op_address_decoder_2x4 IS
  PORT(
       sel0, sel1            : IN  BIT;
       address_decoder_out0,
       address_decoder_out1,
       address_decoder_out2,
       address_decoder_out3  : OUT BIT);
END ENTITY vhd_op_address_decoder_2x4;
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
--
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
ARCHITECTURE op_model OF vhd_op_address_decoder_2x4 IS
--
BEGIN
  --
  address_decoder_out0 <= ( (NOT sel1) AND (NOT sel0) );
  address_decoder_out1 <= ( (NOT sel1) AND      sel0  );
  address_decoder_out2 <= (      sel1  AND (NOT sel0) );
  address_decoder_out3 <= (      sel1  AND      sel0  );
  --
END ARCHITECTURE op_model;
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
--
-- EOF
--
```

Listagem 12 - *Address decoder* 2x4, com *enable*, baseado em operador lógico:

```
--
-- ========================================================
-- Address decoder 2x4, com enable, baseado em operador logico
-- ========================================================
--
-- ==========================================
-- Arquivo: vhd_op_address_decoder_2x4_ena.vhd
-- ==========================================
--
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
ENTITY vhd_op_address_decoder_2x4_ena IS
  PORT(
       ena                   : IN  BIT;
       sel0, sel1            : IN  BIT;
       address_decoder_out0,
       address_decoder_out1,
       address_decoder_out2,
       address_decoder_out3  : OUT BIT);
END ENTITY vhd_op_address_decoder_2x4_ena;
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
--
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
ARCHITECTURE op_model OF vhd_op_address_decoder_2x4_ena IS
--
BEGIN
  --
  address_decoder_out0 <= ( (NOT sel1) AND (NOT sel0) ) AND ena;
  address_decoder_out1 <= ( (NOT sel1) AND      sel0  ) AND ena;
  address_decoder_out2 <= (      sel1  AND (NOT sel0) ) AND ena;
  address_decoder_out3 <= (      sel1  AND      sel0  ) AND ena;
  --
END ARCHITECTURE op_model;
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
--
-- EOF
--
```

7.4.3 *Address decoder* 3x8

- A Listagem 13 apresenta um *address decoder* 3x8, empregando comando condicional.
- A Listagem 14 apresenta um *address decoder* 3x8, com *enable*, empregando comando condicional.

Listagem 13 - *Address decoder* 3x8, empregando comando condicional:

```
--
-- ============================================================
-- Address decoder 3x8, empregando comando condicional de VHDL
-- ============================================================
--
-- ========================================
-- Arquivo: vhd_when_address_decoder_3x8.vhd
-- ========================================
--
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
ENTITY vhd_when_address_decoder_3x8 IS
  GENERIC(-- Number Of Selectors
          nos: INTEGER := 3;
          -- Number Of Outputs
          noo: INTEGER := 8);
  --
  PORT(
       sel      : IN  BIT_VECTOR((nos-1) DOWNTO 0);
       address_decoder_out: OUT BIT_VECTOR((noo-1) DOWNTO 0) );
END ENTITY vhd_when_address_decoder_3x8;
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
--
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
ARCHITECTURE when_model OF vhd_when_address_decoder_3x8 IS
--
BEGIN
  --
  address_decoder_out(0) <= '1' WHEN sel = "000" ELSE '0';
  address_decoder_out(1) <= '1' WHEN sel = "001" ELSE '0';
  address_decoder_out(2) <= '1' WHEN sel = "010" ELSE '0';
  address_decoder_out(3) <= '1' WHEN sel = "011" ELSE '0';
  address_decoder_out(4) <= '1' WHEN sel = "100" ELSE '0';
  address_decoder_out(5) <= '1' WHEN sel = "101" ELSE '0';
  address_decoder_out(6) <= '1' WHEN sel = "110" ELSE '0';
  address_decoder_out(7) <= '1' WHEN sel = "111" ELSE '0';
  --
END ARCHITECTURE when_model;
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
--
-- EOF
--
```

Listagem 14 - *Address decoder* 3x8, com *enable*, empregando comando condicional:

```
--
-- ========================================================================
-- Address decoder 3x8, com enable, empregando comando condicional de VHDL
-- ========================================================================
--
-- ===========================================
-- Arquivo: vhd_when_address_decoder_3x8_ena.vhd
-- ===========================================
--
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
ENTITY vhd_when_address_decoder_3x8_ena IS
  GENERIC(-- Number Of Selectors
          nos: INTEGER := 3;
          -- Number Of Outputs
          noo: INTEGER := 8);
  --
  PORT(
       ena      : IN  BIT;
       sel      : IN  BIT_VECTOR((nos-1) DOWNTO 0);
       address_decoder_out: OUT BIT_VECTOR((noo-1) DOWNTO 0) );
END ENTITY vhd_when_address_decoder_3x8_ena;
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
--
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
ARCHITECTURE when_model OF vhd_when_address_decoder_3x8_ena IS
--
BEGIN
  --
  address_decoder_out(0) <= '1' WHEN (sel = "000" AND ena = '1') ELSE '0';
  address_decoder_out(1) <= '1' WHEN (sel = "001" AND ena = '1') ELSE '0';
  address_decoder_out(2) <= '1' WHEN (sel = "010" AND ena = '1') ELSE '0';
  address_decoder_out(3) <= '1' WHEN (sel = "011" AND ena = '1') ELSE '0';
  address_decoder_out(4) <= '1' WHEN (sel = "100" AND ena = '1') ELSE '0';
  address_decoder_out(5) <= '1' WHEN (sel = "101" AND ena = '1') ELSE '0';
  address_decoder_out(6) <= '1' WHEN (sel = "110" AND ena = '1') ELSE '0';
  address_decoder_out(7) <= '1' WHEN (sel = "111" AND ena = '1') ELSE '0';
  --
END ARCHITECTURE when_model;
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
--
-- EOF
--
```

Capítulo 8

Deslocador configurável (*barrel shifter*)

8.1 Introdução

Nesse capítulo, são apresentados códigos para alguns exemplos de deslocador configurável (*barrel shifter*). Serão levadas em consideração as seguintes opções:

- Quantidade de deslocamentos.
- Deslocamento lógico e aritmético.
- Deslocamento para esquerda e para direita.
- Rotação para esquerda e para direita.

8.2 *Barrel shifter* com deslocamento para a direita

- A Listagem 1 apresenta um *barrel shifter*, baseado em uma estrutura que contém níveis de deslocamento, com seleção da quantidade de deslocamentos, com seleção de deslocamento lógico ou aritmético e com deslocamento para a direita.
- A Listagem 2 apresenta o deslocador para direita configurável que representa cada um dos níveis do *barrel shifter* da Listagem 1. Ele é baseado em Multiplexador 2x1 e apresenta seleção da quantidade de deslocamentos, seleção de deslocamento lógico ou aritmético e deslocamento para a direita.

Listagem 1 - *Barrel shifter* baseado em uma estrutura que contém níveis de deslocamento:

```
--
-- ======================================================
-- Barrel shifter,
-- baseado em uma estrutura que contem niveis de deslocamento,
-- com selecao da quantidade de deslocamentos,
-- com selecao de deslocamento logico ou aritmetico
-- e
-- com deslocamento para a direita
-- ======================================================
--
--
```

```
-- ==================================
-- Arquivo: vhd_shr_comb_la_3_lvls.vhd
-- ==================================
--
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
ENTITY vhd_shr_comb_la_3_lvls IS
  --> Info:
  -->   N levels: range [0 to (nol-1)]
  -->   B bits  : range [0 to (nob-1)]
  -->   S shifts: range [0 to (nob-1)]
  GENERIC(
          nol: POSITIVE := 3; --> Nbr Of Levels
          nob: POSITIVE := 8  --> (nob = 2**nol)
         );
  --
  PORT(
       log0_arit1_sel : IN  BIT;  --> 0 = log
                                  --> 1 = arit
       --
       shft_n0_y1_sel : IN  BIT_VECTOR((nol-1) DOWNTO 0);
                        --> each bit: 0 = no  /  1 = yes
       --
       shr_in         : IN  BIT_VECTOR((nob-1) DOWNTO 0);
       shr_out        : OUT BIT_VECTOR((nob-1) DOWNTO 0)
      );
END ENTITY vhd_shr_comb_la_3_lvls ;
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
--
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
ARCHITECTURE estrut_model OF vhd_shr_comb_la_3_lvls IS
--
COMPONENT vhd_shr_comb_la_N_lvls_level IS
  --> Info:
  -->   N levels: range [0 to (nol-1)]
  -->   B bits  : range [0 to (nob-1)]
  -->   S shifts: range [0 to (nob-1)]
  GENERIC(
          nos : POSITIVE := 1;  --> (nod = 2**level)
          nob : POSITIVE := 8   --> (nob = 2**nol  )
         );
  --
  PORT(
       log0_arit1_sel : IN  BIT;  --> 0 = log
                                  --> 1 = arit
       --
       shft_n0_y1_sel : IN  BIT;  --> 0 = no
                                  --> 1 = yes
       --
       shr_in         : IN  BIT_VECTOR((nob-1) DOWNTO 0);
```

```
       shr_out        : OUT BIT_VECTOR((nob-1) DOWNTO 0)
      );
END COMPONENT vhd_shr_comb_la_N_lvls_level ;
--
TYPE nol_array IS
     ARRAY(0 TO nol) OF BIT_VECTOR(shr_in'range);
--
SIGNAL data_into_shft_level: nol_array;
--
BEGIN
  --
  data_into_shft_level(0) <= shr_in;
  --
  --
  --
    shr_lvl_0:
      vhd_shr_comb_la_N_lvls_level
        GENERIC MAP (1,nob)
           PORT MAP (log0_arit1_sel,
                     shft_n0_y1_sel(0),
                     data_into_shft_level(0),
                     data_into_shft_level( 0 +1));
  --
    shr_lvl_1:
      vhd_shr_comb_la_N_lvls_level
        GENERIC MAP (2,nob)
           PORT MAP (log0_arit1_sel,
                     shft_n0_y1_sel(1),
                     data_into_shft_level(1),
                     data_into_shft_level( 1 +1));
  --
    shr_lvl_2:
      vhd_shr_comb_la_N_lvls_level
        GENERIC MAP (4,nob)
           PORT MAP (log0_arit1_sel,
                     shft_n0_y1_sel(2),
                     data_into_shft_level(2),
                     data_into_shft_level( 2 +1));
  --
  --
  --
  shr_out <= data_into_shft_level(nol);
  --
END ARCHITECTURE estrut_model ;
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
--
-- EOF
--
```

Listagem 2 - Deslocador para direita configurável (níveis do *barrel shifter* da Listagem 1):

```
--
-- ====================================================================
-- Deslocador para direita configuravel,
-- que representa cada um dos niveis do barrel shifter da Listagem 1
-- ====================================================================
--
-- ========================================
-- Arquivo: vhd_shr_comb_la_N_lvls_level.vhd
-- ========================================
--
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
ENTITY vhd_shr_comb_la_N_lvls_level IS
  --> Info:
  -->   N levels: range [0 to (nol-1)]
  -->   B bits  : range [0 to (nob-1)]
  -->   S shifts: range [0 to (nob-1)]
  GENERIC(
          nos : POSITIVE := 1;  --> (nod = 2**level)
          nob : POSITIVE := 8   --> (nob = 2**nol  )
         );
  --
  PORT(
       log0_arit1_sel : IN  BIT;  --> 0 = log
                                  --> 1 = arit
       --
       shft_n0_y1_sel : IN  BIT;  --> 0 = no
                                  --> 1 = yes
       --
       shr_in         : IN  BIT_VECTOR((nob-1) DOWNTO 0);
       shr_out        : OUT BIT_VECTOR((nob-1) DOWNTO 0)
      );
END ENTITY vhd_shr_comb_la_N_lvls_level ;
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
--
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
ARCHITECTURE estrut_model OF vhd_shr_comb_la_N_lvls_level IS
--
BEGIN
  --
  proc_lvl_shft_in :
  PROCESS (log0_arit1_sel , shft_n0_y1_sel, shr_in) IS
  --
  VARIABLE shft_msb    : BIT;
  VARIABLE shft_vector : BIT_VECTOR(shr_in'range);
  --
  BEGIN
    --
```

```
    IF (log0_arit1_sel = '0') THEN
      shft_msb := '0';
    ELSE
      shft_msb := shr_in(nob-1);
    END IF ;
    --
    --
    --
    IF (shft_n0_y1_sel= '0') THEN
      shr_out <= shr_in;
    ELSE
      --
      loop_msb:
      FOR k IN (nob-1) DOWNTO ( (nob-1) - (nos-1) ) LOOP
        shft_vector (k) := shft_msb;
      END LOOP ;
      --
      loop_others:
      FOR k IN ( (nob-1) - (nos) ) DOWNTO 0 LOOP
        shft_vector (k) := shr_in( k + (nos) );
      END LOOP ;
      --
      shr_out <= shft_vector ;
    END IF ;
    --
  END PROCESS proc_lvl_shft_in ;
  --
END ARCHITECTURE estrut_model ;
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
--
-- EOF
--
```

Capítulo 9

Decodificador

9.1 Introdução

Nesse capítulo, são apresentados códigos para alguns exemplos de decodificador.

9.2 Decodificador com validação de código

A seguir, são apresentados códigos para um decodificador com validação de código. O decodificador em questão possui um sinal de saída que alerta quando o código de entrada é válido (A = '1') ou não (A = '0'). A especificação do conjunto dos códigos de entrada válidos, bem como do mapeamento entre os códigos de entrada e de saída, pode ser obtida diretamente da Listagem 1.

- A Listagem 1 apresenta um decodificador com validação de código, baseado em código condicional.
- A Listagem 2 apresenta um decodificador com validação de código, baseado em operador lógico.

Listagem 1 - Decodificador com validação de código, baseado em código condicional:

```
--
-- ====================================
-- Decodificador com validacao de codigo,
-- baseado em codigo condicional
-- ====================================
--
-- ========================================
-- Arquivo: vhd_when_decoder_4x8_2017_2_a2.vhd
-- ========================================
--
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
ENTITY vhd_when_decoder_4x8_2017_2_a2 IS
 --
  GENERIC(noib : INTEGER := 4;
          noob : INTEGER := 8);
 --
```

```
  PORT(
       x : IN  BIT_VECTOR( (noib-1) DOWNTO 0);
       y : OUT BIT_VECTOR( (noob-1) DOWNTO 0);
       a : OUT BIT
      );
 --
END ENTITY vhd_when_decoder_4x8_2017_2_a2;
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
--
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
ARCHITECTURE conditional_model OF vhd_when_decoder_4x8_2017_2_a2 IS
--
BEGIN
 --
  a <= '1' WHEN x = "0001" ELSE
       '1' WHEN x = "0011" ELSE
       '1' WHEN x = "0100" ELSE
       '1' WHEN x = "0110" ELSE
       --
       '1' WHEN x = "1001" ELSE
       '1' WHEN x = "1011" ELSE
       '1' WHEN x = "1100" ELSE
       '1' WHEN x = "1110" ELSE
       --
       '0';
 --
  y <= "01000011" WHEN x = "0001" ELSE
       "01100100" WHEN x = "0011" ELSE
       "10010110" WHEN x = "0100" ELSE
       "10111001" WHEN x = "0110" ELSE
       --
       "11001011" WHEN x = "1001" ELSE
       "11101100" WHEN x = "1011" ELSE
       "00011110" WHEN x = "1100" ELSE
       "00110001" WHEN x = "1110" ELSE
       --
       "00000000";
 --
END ARCHITECTURE conditional_model;
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
--
-- EOF
--
```

Listagem 2 - Decodificador com validação de código, baseado em operador lógico:

```
--
-- ====================================
-- Decodificador com validacao de codigo,
-- baseado em operador logico
-- ====================================
--
-- ========================================
-- Arquivo: vhd_op_decoder_4x8_2017_2_a2.vhd
-- ========================================
--
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
ENTITY vhd_op_decoder_4x8_2017_2_a2 IS
  GENERIC(noib : INTEGER := 4;
          noob : INTEGER := 8);
 --
  PORT(
       x : IN  BIT_VECTOR( (noib-1) DOWNTO 0);
       y : OUT BIT_VECTOR( (noob-1) DOWNTO 0);
       a : OUT BIT
      );
END ENTITY vhd_op_decoder_4x8_2017_2_a2;
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
--
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
ARCHITECTURE op_model OF vhd_op_decoder_4x8_2017_2_a2 IS
--
BEGIN
 --
  a <= x(2) XOR x(0);
 --
  y(7) <= x(3) XOR x(2);
  y(6) <= ( NOT x(2) );
  y(5) <= x(1);
  y(4) <= x(2);
  y(3) <= ( ( NOT x(3) ) AND x(2) AND x(1) ) OR
          ( x(3) AND ( NOT x(1) ) ) OR
          ( x(3) AND x(0) );
  y(2) <= x(1) XNOR x(0);
  y(1) <= ( NOT x(1) );
  y(0) <= x(1) XOR x(0);
 --
END ARCHITECTURE op_model;
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
--
-- EOF
--
```

Capítulo 10

Conversor de códigos

10.1 Introdução

Nesse capítulo, são apresentados códigos para alguns exemplos de conversores de códigos. Serão considerados os seguintes códigos: Binário Seqüencial, *One-hot*, Gray, Johnson, BCD (*Binary-Coded Decimal*).

10.2 Conversor Binário-*One-hot*

Um conversor de código Binário Seqüencial para código *One-hot* é um bloco funcional com B *bits* de entrada e $N = 2^B$ *bits* de saída. A saída S_0 representa o LSB, enquanto a saída S_{N-1} representa o MSB. A saída S_k, para $0 \leq k \leq (N-1)$, é equivalente ao mintermo m_k. Portanto, por definição, esse conversor é um *address decoder* BxN.

10.3 Conversor Binário-Gray

- A Listagem 1 apresenta um conversor de código Binário Seqüencial para código Gray, baseado em operadores.

Listagem 1 - Conversor de Binário Seqüencial para Gray, baseado em operadores:

```
--
-- ======================================================
-- Conversor de código Binário Seqüencial para código Gray,
-- baseado em operadores
-- ======================================================
--
-- ============================
-- Arquivo: binary_to_gray_N.vhd
-- ============================
--
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
ENTITY binary_to_gray_N IS
  GENERIC(nob : INTEGER := 4);
  --
  PORT(msb_plus : IN  BIT;
```

```
       data_in  : IN  BIT_VECTOR ( (nob-1) DOWNTO 0);
       data_out : OUT BIT_VECTOR ( (nob-1) DOWNTO 0);
       lsb_less : OUT BIT);
END ENTITY binary_to_gray_N ;
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
--
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
ARCHITECTURE hier_gen_model OF binary_to_gray_N IS
--
SIGNAL internal_data_bus : BIT_VECTOR ( (nob) DOWNTO 0);
--
BEGIN
  --
  internal_data_bus (nob) <= msb_plus ;
  internal_data_bus ( (nob-1) DOWNTO 0 ) <= data_in ;
  --
  bin_2_gray:
  FOR k IN (nob-1) DOWNTO 0 GENERATE
    BEGIN
      bit_k: data_out (k) <= internal_data_bus (k+1)
                             XOR
                             internal_data_bus(K) ;
    END GENERATE;
  --
  lsb_less <= data_in(0) ;
  --
END ARCHITECTURE hier_gen_model ;
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
--
-- EOF
--
```

Capítulo 11

Separador e relacionador de *bits* 1 e 0 em palavra de N *bits*

11.1 Introdução

Nesse capítulo, são apresentados códigos para alguns exemplos de separador e de relacionador de *bits* 1 e 0, em palavra de N *bits*.

11.2 Separador de *bits* 1 e 0 em palavra de N *bits*

A seguir, é apresentado código para um separador de *bits* 1 e 0, em uma palavra de N *bits*. O separador em questão aloca os *bits* 1 da palavra de entrada nos *bits* mais significativos da palavra de saída.

- A Listagem 1 apresenta a célula básica do separador.
- A Listagem 2 apresenta uma coluna genérica do separador, baseada na sua célula básica. Ela é parametrizável, apresentando $N = 4$ como tamanho de palavra original.
- A Listagem 3 apresenta o separador de N *bits*, baseado na sua coluna genérica. Ele é parametrizável, apresentando $N = 6$ como tamanho de palavra original.

Listagem 1 - Célula básica do separador de *bits* 1 e 0, em palavra de N *bits*:

```
--
-- ============================
-- Celula basica
-- do separador de bits 1 e 0,
-- em palavra de N bits
-- ============================
--
-- ============================================
-- Arquivo: vhd_separa_bits_1_0_word_N_basic_cell
-- ============================================
--
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
ENTITY vhd_separa_bits_1_0_word_N_basic_cell IS
  PORT(E_k, Pk_1 : IN  BIT;
```

```
       S_k, P_k  : OUT BIT);
END ENTITY vhd_separa_bits_1_0_word_N_basic_cell ;
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
--
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
ARCHITECTURE log_op_model OF
             vhd_separa_bits_1_0_word_N_basic_cell IS
BEGIN
  --
  S_k <= E_k OR  Pk_1;
  --
  P_k <= E_k AND Pk_1;
  --
END ARCHITECTURE log_op_model ;
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
--
-- EOF
--
```

Listagem 2 - Coluna genérica do separador de *bits* 1 e 0, em palavra de N *bits*:

```
--
-- ============================
-- Coluna generica
-- do separador de bits 1 e 0,
-- em palavra de N bits
-- ============================
--
-- ================================================
-- Arquivo: vhd_separa_bits_1_0_word_N_column_slice
-- ================================================
--
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
ENTITY vhd_separa_bits_1_0_word_N_column_slice IS
  GENERIC(nob : INTEGER := 4);
  --
  PORT(D_k : IN  BIT;
       E   : IN  BIT_VECTOR ( (nob-1) DOWNTO 0);
       S   : OUT BIT_VECTOR ( (nob-1) DOWNTO 0));
END ENTITY vhd_separa_bits_1_0_word_N_column_slice ;
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
--
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
ARCHITECTURE hier_gen_model OF
             vhd_separa_bits_1_0_word_N_column_slice IS
--
COMPONENT vhd_separa_bits_1_0_word_N_basic_cell IS
```

```
  PORT(E_k, Pk_1 : IN  BIT;
       S_k, P_k  : OUT BIT);
END COMPONENT vhd_separa_bits_1_0_word_N_basic_cell ;
--
SIGNAL P_chain: BIT_VECTOR ( nob DOWNTO 0);
--
BEGIN
  --
  P_chain(nob) <= D_k;
  --
  lin_gen: FOR k IN (nob-1) DOWNTO 0 GENERATE
           BEGIN
            bas_cell_k: vhd_separa_bits_1_0_word_N_basic_cell
                        PORT MAP( E(k), P_chain(k+1),
                                  S(k), P_chain(k)   );
           END GENERATE;
  --
END ARCHITECTURE hier_gen_model ;
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
--
-- EOF
--
```

Listagem 3 - Separador de *bits* 1 e 0, em palavra de N *bits*:

```
--
-- ========================
-- Separador de bits 1 e 0,
-- em palavra de N bits
-- ========================
--
-- ==================================
-- Arquivo: vhd_separa_bits_1_0_word_N
-- ==================================
--
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
ENTITY vhd_separa_bits_1_0_word_N IS
  GENERIC(nob : INTEGER := 6);
  --
  PORT(data_in  : IN  BIT_VECTOR ( (nob-1) DOWNTO 0);
       data_out : OUT BIT_VECTOR ( (nob-1) DOWNTO 0));
END ENTITY vhd_separa_bits_1_0_word_N ;
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
--
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
ARCHITECTURE hier_gen_model OF
             vhd_separa_bits_1_0_word_N IS
```

```
--
COMPONENT vhd_separa_bits_1_0_word_N_basic_cell IS
  PORT(E_k, Pk_1 : IN  BIT;
       S_k, P_k  : OUT BIT);
END COMPONENT vhd_separa_bits_1_0_word_N_basic_cell ;
--
COMPONENT vhd_separa_bits_1_0_word_N_column_slice IS
  GENERIC(nob : INTEGER := 4);
  --
  PORT(D_k : IN  BIT;
       E   : IN  BIT_VECTOR ( (nob-1) DOWNTO 0);
       S   : OUT BIT_VECTOR ( (nob-1) DOWNTO 0));
END COMPONENT vhd_separa_bits_1_0_word_N_column_slice ;
--
TYPE m_chain IS ARRAY ( (nob-1) DOWNTO 0)
                OF BIT_VECTOR ( (nob-1) DOWNTO 0);
--
SIGNAL data_chain: m_chain;
--
BEGIN
  --
  init_gen: FOR k IN (nob-1) DOWNTO 0 GENERATE
              data_chain(nob)(k) <= '0';
            END GENERATE;
  --
  col_gen: FOR k IN (nob-1) DOWNTO 0 GENERATE
             col_k: vhd_separa_bits_1_0_word_N_column_slice
                    GENERIC MAP (nob)
                    PORT MAP( data_in(k),
                              data_chain(k+1),
                              data_chain(k)   );
            END GENERATE;
  --
--  data_out ( (nob-1) DOWNTO 0) <= data_chain_0 ( (nob-1) DOWNTO 0);
--  data_out  <= data_chain_0 ( (nob-1) DOWNTO 0);
  data_out  <= data_chain(0);
  --
END ARCHITECTURE hier_gen_model ;
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
--
-- EOF
--
```

11.3 Árbitro da relação entre *bits* 1 e 0 em palavra de N *bits*

A seguir, é apresentado código para um árbitro da relação entre *bits* 1 e 0, em palavra de N *bits*. O árbitro em questão indica se a quantidade de *bits* 1 é menor_que, igual_a ou maior_que os *bits* 0 da palavra. Ele analisa os *bits* medianos de uma palavra, previamente organizada por um separador de *bits* 1 e 0, e toma a decisão.

- A Listagem 4 apresenta um árbitro para *bits* 1 alocados nos *bits* menos significativos da palavra.
- A Listagem 5 apresenta um árbitro para *bits* 1 alocados nos *bits* mais significativos da palavra.
- A Listagem 6 apresenta um árbitro que é independente da ordem com que os dois grupos de *bits* são alocados na palavra.

Listagem 4 - Árbitro para *bits* 1 alocados nos *bits* menos significativos da palavra:

```
--
-- ================================================
-- Arbitro para bits 1
-- alocados nos bits menos significativos da palavra
-- ================================================
--
-- ====================================================
-- Arquivo: vhd_total_1_lt_eq_gt_0_after_separation_lsb
-- ====================================================
--
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
ENTITY vhd_total_1_lt_eq_gt_0_after_separation_lsb IS
  PORT(--> Even0_Odd1 == 0 for even N
       --> Even0_Odd1 == 1 for odd  N
       Even0_Odd1: IN BIT;
       --
       --> Bits at: N/2 and N/2 - 1
       Mid_N, Mid_N_minus_1: IN BIT;
       --
       --> Less, Equal, Greater: 1 versus 0
       L1T0, E1A0, G1T0: OUT BIT);
END ENTITY vhd_total_1_lt_eq_gt_0_after_separation_lsb ;
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
--
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
ARCHITECTURE log_op_model OF
             vhd_total_1_lt_eq_gt_0_after_separation_lsb IS
--
BEGIN
  --
  L1T0 <= (NOT Mid_N_minus_1)
```

```
          OR
          ( (NOT Mid_N)  AND  (Even0_Odd1) );
  --
  --
  G1T0 <= (Mid_N);
  --
  --      E1A0 = (NOT L1T0)  AND  (NOT G1T0);
  E1A0 <= ( (NOT Mid_N)
            AND
            (Mid_N_minus_1)
            AND
            (NOT Even0_Odd1) );
  --
END ARCHITECTURE log_op_model ;
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
--
-- EOF
--
```

Listagem 5 - Árbitro para *bits* 1 alocados nos *bits* mais significativos da palavra:

```
--
-- ================================================
-- Arbitro para bits 1
-- alocados nos bits mais significativos da palavra
-- ================================================
--
-- ====================================================
-- Arquivo: vhd_total_1_lt_eq_gt_0_after_separation_msb
-- ====================================================
--
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
ENTITY vhd_total_1_lt_eq_gt_0_after_separation_msb IS
  PORT(--> Even0_Odd1 == 0 for even N
       --> Even0_Odd1 == 1 for odd  N
       Even0_Odd1: IN BIT;
       --
       --> Bits at: N/2 and N/2 - 1
       Mid_N, Mid_N_minus_1: IN BIT;
       --
       --> Less, Equal, Greater: 1 versus 0
       L1T0, E1A0, G1T0: OUT BIT);
END ENTITY vhd_total_1_lt_eq_gt_0_after_separation_msb ;
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
--
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
ARCHITECTURE log_op_model OF
```

```
              vhd_total_1_lt_eq_gt_0_after_separation_msb IS
--
BEGIN
  --
  L1T0 <= (NOT Mid_N);
  --
  --
  G1T0 <= (Mid_N_minus_1)
          OR
          ( (Mid_N)  AND  (Even0_Odd1) );
  --
  --      E1A0 = (NOT L1T0)  AND  (NOT G1T0);
  E1A0 <= ( (Mid_N)
            AND
            (NOT Mid_N_minus_1)
            AND
            (NOT Even0_Odd1) );
  --
END ARCHITECTURE log_op_model ;
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
--
-- EOF
--
```

Listagem 6 - Árbitro que é independente da ordem de alocação dos dois grupos de *bits*:

```
--
-- ================================================
-- Arbitro que eh independente da ordem de alocacao
-- dos dois grupos de bits
-- ================================================
--
-- ================================================
-- Arquivo: vhd_total_1_lt_eq_gt_0_after_separation
-- ================================================
--
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
ENTITY vhd_total_1_lt_eq_gt_0_after_separation IS
  PORT(--> Even0_Odd1 == 0 for even N
       --> Even0_Odd1 == 1 for odd  N
       Even0_Odd1: IN BIT;
       --
       --> Bits at: N/2 and N/2 - 1
       Mid_N, Mid_N_minus_1: IN BIT;
       --
       --> Less, Equal, Greater: 1 versus 0
       L1T0, E1A0, G1T0: OUT BIT);
```

```
END ENTITY vhd_total_1_lt_eq_gt_0_after_separation ;
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
--
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
ARCHITECTURE log_op_model OF
             vhd_total_1_lt_eq_gt_0_after_separation IS
--
BEGIN
  --
  L1T0 <= ( (NOT Mid_N)  AND  (NOT Mid_N_minus_1) )
          OR
          ( (NOT Mid_N)  AND  (Even0_Odd1)        );
  --
  --
  G1T0 <= ( Mid_N  AND  Mid_N_minus_1 )
          OR
          ( Mid_N  AND  Even0_Odd1    );
  --
  --      E1A0 = (NOT L1T0)  AND  (NOT G1T0);
  E1A0 <= ( (NOT Mid_N)
            AND
            (    Mid_N_minus_1)
            AND
            (NOT Even0_Odd1) )
          OR
          ( (    Mid_N)
            AND
            (NOT Mid_N_minus_1)
            AND
            (NOT Even0_Odd1) );
  --
END ARCHITECTURE log_op_model ;
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
--
-- EOF
--
```

11.4 Relacionador de *bits* 1 e 0 em palavra de N *bits*

A seguir, é apresentado código para um relacionador de *bits* 1 e 0, em uma palavra de N *bits*. O relacionador em questão indica se a quantidade de *bits* 1 é menor_que, igual_a ou maior_que os *bits* 0 da palavra. Inicialmente, ele realiza um agrupamento dos *bits*, utilizando um separador de *bits* apresentado anteriormente. Em seguida, ele analisa os *bits* medianos do agrupamento, utilizando um árbitro apresentado anteriormente.

- A Listagem 7 apresenta o relacionador de *bits* 1 e 0, hierárquico e parametrizável, baseado no separador de *bits* da Listagem 3 e no árbitro da Listagem 6.
- A Listagem 8 apresenta uma instanciação do relacionador de *bits* 1 e 0 da Listagem 7, empregando $N = 7$ *bits*.

Listagem 7 - Relacionador de *bits* 1 e 0, em palavra de N *bits*:

```
--
-- ========================================================
-- Relacionador de bits 1 e 0,
-- hierarquico e parametrizavel,
-- baseado no separador de bits e no arbitro.
-- ========================================================
--
-- ================================================
-- Arquivo: vhd_separa_e_relaciona_bits_1_0_word_N
-- ================================================
--
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
ENTITY vhd_separa_e_relaciona_bits_1_0_word_N IS
  GENERIC(nob : INTEGER := 4);  -- Number Of Bits
  --
  PORT(data_in  : IN  BIT_VECTOR ( (nob-1) DOWNTO 0);
       data_out : OUT BIT_VECTOR ( (nob-1) DOWNTO 0);
       L1T0, E1A0, G1T0: OUT BIT);
END ENTITY vhd_separa_e_relaciona_bits_1_0_word_N ;
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
--
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
ARCHITECTURE hier_gen_model OF
             vhd_separa_e_relaciona_bits_1_0_word_N IS
--
COMPONENT vhd_separa_bits_1_0_word_N IS
  GENERIC(nob : INTEGER := 3);
  --
  PORT(data_in  : IN  BIT_VECTOR ( (nob-1) DOWNTO 0);
       data_out : OUT BIT_VECTOR ( (nob-1) DOWNTO 0));
END COMPONENT vhd_separa_bits_1_0_word_N ;
--
COMPONENT vhd_total_1_lt_eq_gt_0_after_separation IS
  PORT(--> Even0_Odd1 == 0 for even N
```

```
        --> Even0_Odd1 == 1 for odd  N
        Even0_Odd1: IN BIT;
        --
        --> Bits at: N/2 and N/2 - 1
        Mid_N, Mid_N_minus_1: IN BIT;
        --
        --> Less, Equal, Greater: 1 versus 0
        L1T0, E1A0, G1T0: OUT BIT);
END COMPONENT vhd_total_1_lt_eq_gt_0_after_separation ;
--
SIGNAL data_out_bus : BIT_VECTOR ( (nob-1) DOWNTO 0);
--
SIGNAL Even0_Odd1, Mid_N, Mid_N_minus_1 : BIT;
--
BEGIN
  --
  separa: vhd_separa_bits_1_0_word_N
  GENERIC MAP (nob)
          PORT MAP (data_in, data_out_bus);
  --
  --
  data_out <= data_out_bus;
  --
  --
  PROCESS (data_out_bus)
  BEGIN
    IF ( (nob REM 2) = 0 ) THEN
      Even0_Odd1    <= '0';  -- NOB is even...
      Mid_N         <= data_out_bus( ( (nob-1)/2 ) + 1 );
      Mid_N_minus_1 <= data_out_bus( ( (nob-1)/2 )     );
    ELSE
      Even0_Odd1    <= '1';  -- NOB is odd...
      Mid_N         <= data_out_bus(   (nob-1)/2       );
      Mid_N_minus_1 <= data_out_bus( ( (nob-1)/2 ) - 1 );
    END IF;
  END PROCESS;
  --
  --
  relaciona:  vhd_total_1_lt_eq_gt_0_after_separation
              PORT MAP(Even0_Odd1,
                       Mid_N,
                       Mid_N_minus_1,
                       L1T0, E1A0, G1T0);
  --
END ARCHITECTURE hier_gen_model ;
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
--
-- EOF
--
```

Listagem 8 - Instanciação do relacionador de *bits* 1 e 0, com palavra de $N = 7$ *bits*:

```
--
-- ========================================================
-- Instanciação do relacionador de bits 1 e 0,
-- hierarquico e parametrizavel,
-- baseado no separador de bits e no arbitro,
-- empregando N=7 bits
-- ========================================================
--
-- ================================================
-- Arquivo: vhd_separa_e_relaciona_bits_1_0_word_N7
-- ================================================
--
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
ENTITY vhd_separa_e_relaciona_bits_1_0_word_N7 IS
  GENERIC(nob : INTEGER := 7);  -- Number Of Bits
  --
  PORT(data_in  : IN  BIT_VECTOR ( (nob-1) DOWNTO 0);
       data_out : OUT BIT_VECTOR ( (nob-1) DOWNTO 0);
       L1T0, E1A0, G1T0: OUT BIT);
END ENTITY vhd_separa_e_relaciona_bits_1_0_word_N7 ;
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
--
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
ARCHITECTURE hier_gen_model OF
             vhd_separa_e_relaciona_bits_1_0_word_N7 IS
--
COMPONENT vhd_separa_e_relaciona_bits_1_0_word_N IS
  GENERIC(nob : INTEGER := 4);  -- Number Of Bits
  --
  PORT(data_in  : IN  BIT_VECTOR ( (nob-1) DOWNTO 0);
       data_out : OUT BIT_VECTOR ( (nob-1) DOWNTO 0);
       L1T0, E1A0, G1T0: OUT BIT);
END COMPONENT vhd_separa_e_relaciona_bits_1_0_word_N ;
--
--
BEGIN
  --
  separa_e_relaciona:
    vhd_separa_e_relaciona_bits_1_0_word_N
    GENERIC MAP (nob)
    PORT MAP (data_in, data_out, L1T0, E1A0, G1T0);
  --
END ARCHITECTURE hier_gen_model ;
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
--
-- EOF
--
```

Capítulo 12

Somadores básicos

12.1 Introdução

Nesse capítulo, são apresentados códigos para alguns exemplos de somadores básicos.

12.2 Somador de 3 *bits* ou *full adder* (FA)

- A Listagem 1 apresenta um somador de 3 *bits* ou somador completo ou *full adder* (FA), baseado em operador lógico.

Listagem 1 - Somador de 3 *bits* ou *full adder* (FA), baseado em operador lógico:

```
--
-- ================================================
-- Somador de 3 bits ou somador completo ou full adder,
-- baseado em operador lógico
-- ================================================
--
-- ==========================
-- Arquivo: vhd_op_full_adder
-- ==========================
--
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
ENTITY vhd_op_full_adder IS
  PORT(c_in         : IN  BIT;
       op1, op2     : IN  BIT;
       adder_result : OUT BIT;
       c_out        : OUT BIT);
END ENTITY vhd_op_full_adder;
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
--
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
ARCHITECTURE op_model OF vhd_op_full_adder IS
--
BEGIN
  --
  adder_result <= op1 XOR op2 XOR c_in;
```

```
  --
  c_out <= (op1 AND op2) OR (op1 AND c_in) OR (op2 AND c_in);
  --
END ARCHITECTURE op_model;
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- ----
--
-- EOF
--
```

12.3 Somador com propagação de *carry* (CPA ou RCA)

A seguir, são apresentados códigos para um somador com propagação de *carry* ou *Carry Propagate Adder* (CPA) ou *Ripple Carry Adder* (RCA).

- A Listagem 2 apresenta um RCA de 4 *bits*, empregando o *full adder* da Listagem 1.
- A Listagem 3 apresenta um RCA de 8 *bits*, empregando o RCA de 4 *bits* da Listagem 2.
- A Listagem 4 apresenta um RCA de N *bits*, empregando o *full adder* da Listagem 1. Para o exemplo, é utilizado o valor $N = 8$.

Listagem 2 - Somador RCA de 4 *bits*, empregando o *full adder*:

```
--
-- ============================================
-- Somador RCA de 4 bits, empregando o full adder
-- ============================================
--
-- ============================
-- Arquivo: vhd_hier_cpa_4_bits
-- ============================
--
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
ENTITY vhd_hier_cpa_4_bits IS
  GENERIC(nob : INTEGER := 4);   -- Number Of Bits
  --
  PORT(c_in         : IN  BIT;
       op1, op2     : IN  BIT_VECTOR ( (nob-1) DOWNTO 0);
       adder_result : OUT BIT_VECTOR ( (nob-1) DOWNTO 0);
       c_out        : OUT BIT);
END ENTITY vhd_hier_cpa_4_bits;
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
--
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
ARCHITECTURE hier_model OF vhd_hier_cpa_4_bits IS
--
COMPONENT vhd_op_full_adder IS
```

```
  PORT(c_in         : IN  BIT;
       op1, op2     : IN  BIT;
       adder_result : OUT BIT;
       c_out        : OUT BIT);
END COMPONENT vhd_op_full_adder ;
--
SIGNAL carry_chain: BIT_VECTOR ( nob DOWNTO 0);
--
BEGIN
  --
  carry_chain(0) <= c_in;
  --
  FA0: vhd_op_full_adder
       PORT MAP( carry_chain(0), op1(0), op2(0),
                 adder_result(0), carry_chain(1) );
  FA1: vhd_op_full_adder
       PORT MAP( carry_chain(1), op1(1), op2(1),
                 adder_result(1), carry_chain(2) );
  FA2: vhd_op_full_adder
       PORT MAP( carry_chain(2), op1(2), op2(2),
                 adder_result(2), carry_chain(3) );
  FA3: vhd_op_full_adder
       PORT MAP( carry_chain(3), op1(3), op2(3),
                 adder_result(3), carry_chain(4) );
  --
  c_out <= carry_chain(nob);
  --
END ARCHITECTURE hier_model;
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
--
-- EOF
--
```

Listagem 3 - Somador RCA de 8 *bits*, empregando o RCA de 4 *bits*:

```
--
-- ================================================
-- Somador RCA de 8 bits, empregando o RCA de 4 bits
-- ================================================
--
-- ==============================
-- Arquivo: vhd_hier_cpa_2x4_bits
-- ==============================
--
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
ENTITY vhd_hier_cpa_2x4_bits IS
```

```
  GENERIC(nob : INTEGER := 8;    -- Number Of Bits
          noa : INTEGER := 2);   -- Number Of Adders
  --
  PORT(c_in         : IN  BIT;
       op1, op2     : IN  BIT_VECTOR ( (nob-1) DOWNTO 0);
       adder_result : OUT BIT_VECTOR ( (nob-1) DOWNTO 0);
       c_out        : OUT BIT);
END ENTITY vhd_hier_cpa_2x4_bits ;
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
--
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
ARCHITECTURE hier_model OF vhd_hier_cpa_2x4_bits IS
--
COMPONENT vhd_hier_cpa_4_bits IS
  GENERIC(nob : INTEGER := 4);
  --
  PORT(c_in         : IN  BIT;
       op1, op2     : IN  BIT_VECTOR ( (nob-1) DOWNTO 0);
       adder_result : OUT BIT_VECTOR ( (nob-1) DOWNTO 0);
       c_out        : OUT BIT);
END COMPONENT vhd_hier_cpa_4_bits ;
--
SIGNAL carry_chain: BIT_VECTOR ( noa DOWNTO 0);
--
BEGIN
  --
  carry_chain(0) <= c_in;
  --
  CPA40: vhd_hier_cpa_4_bits
         PORT MAP( carry_chain(0), op1(3 DOWNTO 0), op2(3 DOWNTO 0),
                   adder_result(3 DOWNTO 0), carry_chain(1) );
  CPA41: vhd_hier_cpa_4_bits
         PORT MAP( carry_chain(1), op1(7 DOWNTO 4), op2(7 DOWNTO 4),
                   adder_result(7 DOWNTO 4), carry_chain(2) );
  --
  c_out <= carry_chain(2);
  --
END ARCHITECTURE hier_model;
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
--
-- EOF
--
```

Listagem 4 - Somador RCA de N *bits*, empregando o *full adder*:

```
--
-- ============================================
-- Somador RCA de N bits, empregando o full adder
-- ============================================
--
-- ================================
-- Arquivo: vhd_hier_gen_cpa_Nx1_bits
-- ================================
--
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
ENTITY vhd_hier_gen_cpa_Nx1_bits IS
  GENERIC(nob : INTEGER := 8);
  --
  PORT(c_in         : IN  BIT;
       op1, op2     : IN  BIT_VECTOR ( (nob-1) DOWNTO 0);
       adder_result : OUT BIT_VECTOR ( (nob-1) DOWNTO 0);
       c_out        : OUT BIT);
END ENTITY vhd_hier_gen_cpa_Nx1_bits ;
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
--
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
ARCHITECTURE hier_gen_model OF vhd_hier_gen_cpa_Nx1_bits IS
--
COMPONENT vhd_op_full_adder IS
  PORT(c_in         : IN  BIT;
       op1, op2     : IN  BIT;
       adder_result : OUT BIT;
       c_out        : OUT BIT);
END COMPONENT vhd_op_full_adder ;
--
SIGNAL carry_chain: BIT_VECTOR ( nob DOWNTO 0);
--
BEGIN
  --
  carry_chain(0) <= c_in;
  --
  FA_gen: FOR k IN 0 TO (nob-1) GENERATE
          BEGIN
           FA_k: vhd_op_full_adder
                 PORT MAP( carry_chain(k), op1(k), op2(k),
                           adder_result(k), carry_chain(k+1) );
          END GENERATE;
  --
  c_out <= carry_chain(nob);
  --
END ARCHITECTURE hier_gen_model ;
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
```

```
--
-- EOF
--
```

Capítulo 13

Detector de *overflow* e saturador

13.1 Introdução

Nesse capítulo, são apresentados códigos para alguns exemplos de detectores de *overflow* e de saturadores.

13.2 Detector de *overflow*

- A Listagem 1 apresenta um detector de *overflow* em adição de operandos codificados em complemento-a-2.

Listagem 1 - Detector de *overflow* em adição de operandos codificados em complemento-a-2:

```
--
-- ====================================================
-- Detector de \textit{overflow}
-- em adição de operandos codificados em complemento-a-2
-- ====================================================
--
-- ========================================================
-- Arquivo: vhd_equ_detector_overflow_adicao_complemento_2
-- ========================================================
--
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
ENTITY vhd_equ_detector_overflow_adicao_complemento_2 IS
  PORT(op1_sign, op2_sign, res_sign: IN  BIT;
       add_overflow                : OUT BIT);
END ENTITY vhd_equ_detector_overflow_adicao_complemento_2 ;
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
--
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
ARCHITECTURE log_op_model OF
          vhd_equ_detector_overflow_adicao_complemento_2 IS
--
BEGIN
  --
add_overflow <= ( (NOT op1_sign) AND (NOT op2_sign)
```

```
                                AND (    res_sign) )
              OR
              ( (    op1_sign) AND (    op2_sign)
                                AND (NOT res_sign) );
  --
END ARCHITECTURE log_op_model ;
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
--
-- EOF
--
```

13.3 Saturador

A seguir, são apresentados códigos para saturadores. O saturador é um bloco funcional responsável por corrigir um valor proveniente de operação numérica com ocorrência de *overflow*.

- Os saturadores a seguir levam em consideração que os dados numéricos estão codificados em complemento-a-2.
- A Listagem 2 apresenta um saturador para valores codificados em complemento-a-2, configurável e parametrizável.
- A Listagem 3 apresenta um saturador para valores codificados em complemento-a-2, de 5 *bits*, empregando o saturador configurável e parametrizável da Listagem 2.

Listagem 2 - Saturador configurável e parametrizável:

```
--
-- ============================================
-- Saturador
-- para valores codificados em complemento-a-2,
-- configurável e parametrizável
-- ============================================
--
-- ========================================
-- Arquivo: vhd_equ_saturador_complemento_2_N
-- ========================================
--
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
ENTITY vhd_equ_saturador_complemento_2_N IS
  GENERIC(nob : INTEGER := 3);  -- Number Of Bits
  --
  PORT(--> overflow signal
       overflow: IN BIT;
       --
       --> saturation type:
       -->   0 = Max Representable Number
```

```
        -->   1 = Max Representable Module
        sat_0MaxNbr_1MaxMod: IN BIT;
        --
        --> numeric values 2's complement
        inp_value: IN  BIT_VECTOR ( (nob-1) DOWNTO 0);
        out_value: OUT BIT_VECTOR ( (nob-1) DOWNTO 0)
        );
END ENTITY vhd_equ_saturador_complemento_2_N ;
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
--
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
ARCHITECTURE hier_gen_model OF
             vhd_equ_saturador_complemento_2_N IS
--
BEGIN
  --
  -- signal bit = MSB
  out_value(nob-1) <= ( (NOT overflow) AND (    inp_value(nob-1)) )
                      OR
                      ( (    overflow) AND (NOT inp_value(nob-1)) );
  --
  --
  -- mantissa bits, but LSB
  middle_out_bits:
    FOR bit_ind IN (nob-2) DOWNTO 1 GENERATE
      out_value(bit_ind) <= ( (NOT overflow) AND inp_value(bit_ind) )
                            OR
                            (      overflow  AND inp_value(nob-1)   );


    END GENERATE;
  --
  -- mantissa bit = LSB
  out_value(0) <= ( (NOT overflow) AND inp_value(0) )
                  OR
                  (      overflow  AND inp_value(nob-1) )
                  OR
                  (      overflow  AND sat_0MaxNbr_1MaxMod );
  --
END ARCHITECTURE hier_gen_model ;
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
--
-- EOF
--
```

Listagem 3 - Saturador de 5 *bits*, empregando o saturador configurável e parametrizável:

```
--
-- ====================================================
-- Saturador
-- para valores codificados em complemento-a-2,
-- de 5 \textit{bits},
-- empregando o saturador configurável e parametrizável
-- ====================================================
--
-- ==========================================
-- Arquivo: vhd_equ_saturador_complemento_2_N5
-- ==========================================
--
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
ENTITY vhd_equ_saturador_complemento_2_N5 IS
  GENERIC(nob : INTEGER := 5);  -- Number Of Bits
  --
  PORT(--> overflow signal
       overflow: IN BIT;
       --
       --> saturation type:
       -->   0 = Max Representable Number
       -->   1 = Max Representable Module
       sat_0MaxNbr_1MaxMod: IN BIT;
       --
       --> numeric values 2's complement
       inp_value: IN  BIT_VECTOR ( (nob-1) DOWNTO 0);
       out_value: OUT BIT_VECTOR ( (nob-1) DOWNTO 0)
       );
END ENTITY vhd_equ_saturador_complemento_2_N5 ;
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
--
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
ARCHITECTURE hier_gen_model OF
             vhd_equ_saturador_complemento_2_N5 IS
--
COMPONENT vhd_equ_saturador_complemento_2_N IS
  GENERIC(nob : INTEGER := 3);  -- Number Of Bits
  --
  PORT(--> overflow signal
       overflow: IN BIT;
       --
       --> saturation type:
       -->   0 = Max Representable Number
       -->   1 = Max Representable Module
       sat_0MaxNbr_1MaxMod: IN BIT;
       --
       --> numeric values 2's complement
```

```
        inp_value: IN  BIT_VECTOR ( (nob-1) DOWNTO 0);
        out_value: OUT BIT_VECTOR ( (nob-1) DOWNTO 0)
        );
END COMPONENT vhd_equ_saturador_complemento_2_N ;
--
--
BEGIN
  --
  saturador:
    vhd_equ_saturador_complemento_2_N
    GENERIC MAP (nob)
    PORT MAP (overflow, sat_0MaxNbr_1MaxMod, inp_value, out_value);
  --
END ARCHITECTURE hier_gen_model ;
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
--
-- EOF
--
```

Parte IV

Circuitos Sequenciais: Construções básicas

Capítulo 14

Elemento básico de armazenamento: *flip-flop*

14.1 Introdução

Flip-flop é uma designação genérica para um circuito digital capaz de armazenar um *bit*. Portanto, um *flip-flop* pode ser chamado de Elemento Básico de Armazenamento (EBA).

Nesse capítulo, são apresentados códigos para alguns tipos de *flip-flops*.

14.2 *Flip-flops* com saída simples e sem controles extras

- A Listagem 1 apresenta um *flip-flop* do tipo D, sensível à transição positiva, com saída simples, sem controles extras.
- A Listagem 2 apresenta um *flip-flop* do tipo JK, sensível à transição positiva, com saída simples, sem controles extras.
- A Listagem 3 apresenta um *flip-flop* do tipo T, com entrada seletora, sensível à transição positiva, com saída simples, sem controles extras.
- A Listagem 4 apresenta um *flip-flop* do tipo T, sem entrada seletora, sensível à transição positiva, com saída simples, sem controles extras.

Listagem 1 - *Flip-flop* D, sem controles extras:

```
--
-- ================================================
-- Flip-flop do tipo D, sensivel aa transicao positiva,
-- com saida simples, sem controles extras
-- ================================================
--
-- =================
-- Arquivo: vhd_d_ff
-- =================
--
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
ENTITY vhd_d_ff IS
  PORT(d  :  IN BIT;
```

```
       clk:  IN BIT;
       q  : OUT BIT);
END ENTITY vhd_d_ff ;
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
--
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
ARCHITECTURE process_model OF vhd_d_ff IS
--
BEGIN
  PROCESS (clk)
  --
  BEGIN
    IF ( clk'EVENT AND clk='1' )
      THEN
        q <= d;
    END IF;
  END PROCESS ;
END ARCHITECTURE process_model ;
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
--
-- EOF
--
```

Listagem 2 - *Flip-flop* JK, sem controles extras:

```
--
-- ======================================================
-- Flip-flop do tipo JK, sensivel aa transicao positiva,
-- com saida simples, sem controles extras
-- ======================================================
--
-- ==================
-- Arquivo: vhd_jk_ff
-- ==================
--
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
ENTITY vhd_jk_ff IS
  PORT(j,k:  IN BIT;
       clk:  IN BIT;
       q  : OUT BIT);
END ENTITY vhd_jk_ff ;
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
--
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
ARCHITECTURE process_model OF vhd_jk_ff IS
--
SIGNAL s: BIT;
```

```
--
BEGIN
  PROCESS (clk)
  --
  BEGIN
    IF ( clk'EVENT AND clk='1' )
      THEN
        IF    (j='0' AND k='0')
          THEN
            s <= s;
        ELSIF (j='0' AND k='1')
          THEN
            s <= '0';
        ELSIF (j='1' AND k='0')
          THEN
            s <= '1';
        ELSE
            s <= NOT (s);
        END IF;
    END IF;
  END PROCESS ;
  --
  q <= s;
END ARCHITECTURE process_model ;
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
--
-- EOF
--
```

Listagem 3 - *Flip-flop* T, com entrada seletora, sem controles extras:

```
--
-- ========================================
-- Flip-flop do tipo T, com entrada seletora,
-- sensivel aa transicao positiva,
-- com saida simples, sem controles extras
-- ========================================
--
-- =====================
-- Arquivo: vhd_t_sel_ff
-- =====================
--
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
ENTITY vhd_t_sel_ff IS
  PORT(t   :  IN BIT;
       clk:  IN BIT;
       q   : OUT BIT);
```

```
END ENTITY vhd_t_sel_ff ;
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
--
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
ARCHITECTURE process_model OF vhd_t_sel_ff IS
--
SIGNAL s: BIT;
--
BEGIN
  PROCESS (clk)
  --
  BEGIN
    IF ( clk'EVENT AND clk='1' )
      THEN
        IF (t='1')
          THEN
            s <= NOT (s);
        END IF;
    END IF;
  END PROCESS ;
  --
  q <= s;
END ARCHITECTURE process_model ;
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
--
-- EOF
--
```

Listagem 4 - *Flip-flop* T, sem entrada seletora, sem controles extras:

```
--
-- ========================================
-- Flip-flop do tipo T, sem entrada seletora,
-- sensivel aa transicao positiva,
-- com saida simples, sem controles extras
-- ========================================
--
-- ========================
-- Arquivo: vhd_t_no_sel_ff
-- ========================
--
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
ENTITY vhd_t_no_sel_ff IS
  PORT(clk:  IN BIT;
       q  : OUT BIT);
END ENTITY vhd_t_no_sel_ff ;
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
```

```
--
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
ARCHITECTURE process_model OF vhd_t_no_sel_ff IS
--
SIGNAL s: BIT;
--
BEGIN
  PROCESS (clk)
  --
  BEGIN
    IF ( clk'EVENT AND clk='1' )
      THEN
            s <= NOT (s);
    END IF;
  END PROCESS ;
  --
  q <= s;
END ARCHITECTURE process_model ;
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
--
-- EOF
--
```

14.3 *Flip-flops* com saída simples e com controles extras

- A Listagem 5 apresenta um *flip-flop* do tipo D, sensível à transição positiva, com saída simples, com controles extras.
- A Listagem 6 apresenta um *flip-flop* do tipo JK, sensível à transição positiva, com saída simples, com controles extras.
- A Listagem 7 apresenta um *flip-flop* do tipo T, com entrada seletora, sensível à transição positiva, com saída simples, com controles extras.
- A Listagem 8 apresenta um *flip-flop* do tipo T, sem entrada seletora, sensível à transição positiva, com saída simples, com controles extras.

Listagem 5 - *Flip-flop* D, com controles extras:

```
--
-- ================================================
-- Flip-flop do tipo D, sensivel aa transicao positiva,
-- com saida simples, com controles extras
-- ================================================
--
-- ====================
-- Arquivo: vhd_d_ff_cp
```

```
-- ====================
--
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
ENTITY vhd_d_ff_cp IS
  PORT(d      :  IN BIT;
       clr,pre:  IN BIT;
       clk    :  IN BIT;
       q      : OUT BIT);
END ENTITY vhd_d_ff_cp ;
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
--
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
ARCHITECTURE process_model OF vhd_d_ff_cp IS
--
SIGNAL s: BIT;
--
BEGIN
  PROCESS (clk,pre,clr)
  --
  BEGIN
    IF    ( clr='1')
      THEN
        s <= '0';
    ELSIF ( pre='1')
      THEN
        s <= '1';
    ELSIF ( clk'EVENT AND clk='1' )
      THEN
        s <= d;
    END IF;
  END PROCESS ;
  --
  q <= s;
END ARCHITECTURE process_model ;
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
--
-- EOF
--
```

Listagem 6 - *Flip-flop* JK, com controles extras:

```
--
-- ====================================================
-- Flip-flop do tipo JK, sensivel aa transicao positiva,
-- com saida simples, com controles extras
-- ====================================================
--
-- =====================
-- Arquivo: vhd_jk_ff_cp
-- =====================
--
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
ENTITY vhd_jk_ff_cp IS
  PORT(j,k    :  IN BIT;
       clr,pre:  IN BIT;
       clk    :  IN BIT;
       q      : OUT BIT);
END ENTITY vhd_jk_ff_cp ;
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
--
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
ARCHITECTURE process_model OF vhd_jk_ff_cp IS
--
SIGNAL s: BIT;
--
BEGIN
  PROCESS (clk,pre,clr)
  --
  BEGIN
    IF    ( clr='1')
      THEN
        s <= '0';
    ELSIF ( pre='1')
      THEN
        s <= '1';
    ELSIF ( clk'EVENT AND clk='1' )
      THEN
        IF    (j='0' AND k='0')
          THEN
            s <= s;
        ELSIF (j='0' AND k='1')
          THEN
            s <= '0';
        ELSIF (j='1' AND k='0')
          THEN
            s <= '1';
        ELSE
            s <= NOT (s);
```

```
        END IF;
    END IF;
  END PROCESS ;
  --
  q <= s;
END ARCHITECTURE process_model ;
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
--
-- EOF
--
```

Listagem 7 - *Flip-flop* T, com entrada seletora, com controles extras:

```
--
-- ========================================
-- Flip-flop do tipo T, com entrada seletora,
-- sensivel aa transicao positiva,
-- com saida simples, com controles extras
-- ========================================
--
-- ========================
-- Arquivo: vhd_t_sel_ff_cp
-- ========================
--
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
ENTITY vhd_t_sel_ff_cp IS
  PORT(t      :  IN BIT;
       clr,pre:  IN BIT;
       clk    :  IN BIT;
       q      : OUT BIT);
END ENTITY vhd_t_sel_ff_cp ;
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
--
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
ARCHITECTURE process_model OF vhd_t_sel_ff_cp IS
--
SIGNAL s: BIT;
--
BEGIN
  PROCESS (clk,pre,clr)
  --
  BEGIN
    IF    ( clr='1')
      THEN
        s <= '0';
    ELSIF ( pre='1')
      THEN
```

```
        s <= '1';
    ELSIF ( clk'EVENT AND clk='1' )
      THEN
        IF (t='1')
          THEN
            s <= NOT (s);
        END IF;
    END IF;
  END PROCESS ;
  --
  q <= s;
END ARCHITECTURE process_model ;
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
--
-- EOF
--
```

Listagem 8 - *Flip-flop* T, sem entrada seletora, com controles extras:

```
--
-- ========================================
-- Flip-flop do tipo T, sem entrada seletora,
-- sensivel aa transicao positiva,
-- com saida simples, com controles extras
-- ========================================
--
-- ===========================
-- Arquivo: vhd_t_no_sel_ff_cp
-- ===========================
--
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
ENTITY vhd_t_no_sel_ff_cp IS
  PORT(clr,pre:  IN BIT;
       clk    :  IN BIT;
       q      : OUT BIT);
END ENTITY vhd_t_no_sel_ff_cp ;
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
--
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
ARCHITECTURE process_model OF vhd_t_no_sel_ff_cp IS
--
SIGNAL s: BIT;
--
BEGIN
  PROCESS (clk,pre,clr)
  --
  BEGIN
```

```
    IF    ( clr='1')
      THEN
        s <= '0';
    ELSIF ( pre='1')
      THEN
        s <= '1';
    ELSIF ( clk'EVENT AND clk='1' )
      THEN
        s <= NOT (s);
    END IF;
  END PROCESS ;
  --
  q <= s;
END ARCHITECTURE process_model ;
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
--
-- EOF
--
```

Parte V

Circuitos Sequenciais: Blocos funcionais

Capítulo 15

Exemplos de blocos funcionais

Nessa parte, são apresentados exemplos de circuitos digitais que implementam funções comumente encontradas em sistemas digitais.

A seguir, são abordados os seguintes itens:

- Elementos de armazenamento: registradores.
- Divisores de freqüência.
- Máquinas de Estados Finitos (*Finite-State Machines*) simples.

Capítulo 16

Elementos de armazenamento: registradores

16.1 Introdução

Registrador é uma designação genérica para um circuito digital capaz de armazenar um conjunto de N *bits*.

Nesse capítulo, são apresentados códigos para alguns tipos de registradores.

16.2 Tipos básicos de registradores

- Os *flip-flops* e os multiplexadores que serão empregados nos códigos que se seguem foram apresentados anteriormente nesse documento.
- Os registradores que serão codificados a seguir utilizam *flip-flops* do tipo D, sensíveis à transição positiva, com saída simples, sem controles extras.
- A Listagem 1 apresenta um registrador básico do tipo SISO (*Serial-Input Serial-Output*).
- A Listagem 2 apresenta um registrador básico do tipo SIPO (*Serial-Input Parallel-Output*).
- A Listagem 3 apresenta um registrador básico do tipo PISO (*Parallel-Input Serial-Output*), com deslocamento para a esquerda.
- A Listagem 4 apresenta um registrador básico do tipo PISO (*Parallel-Input Serial-Output*), com deslocamento para a direita.
- A Listagem 5 apresenta um registrador básico do tipo PIPO (*Parallel-Input Parallel-Output*).
- A Listagem 6 apresenta um registrador básico do tipo PIPO (*Parallel-Input Parallel-Output*), com deslocamento para a esquerda e para a direita.

Listagem 1 - Registrador SISO básico:

```
--
-- ============================================================
-- Registrador basico do tipo SISO (Serial-Input Serial-Output)
-- ============================================================
--
-- =====================
-- Arquivo: vhd_reg_siso
-- =====================
--
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
ENTITY vhd_reg_siso IS
  GENERIC(nos: NATURAL := 5);  -- Number of sections
  --
  PORT(reg_si:  IN BIT;
       clk   :  IN BIT;
       reg_so: OUT BIT);
END ENTITY vhd_reg_siso ;
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
--
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
ARCHITECTURE struct_model OF vhd_reg_siso IS
--
COMPONENT vhd_d_ff IS
  PORT(d  :  IN BIT;
       clk:  IN BIT;
       q  : OUT BIT);
END COMPONENT vhd_d_ff ;
--
SIGNAL q_vector: BIT_VECTOR ( nos DOWNTO 0 );
--
BEGIN
  --
  q_vector(nos) <= reg_si;
  --
  shift_action:
  FOR i IN (nos-1) DOWNTO 0 GENERATE
  BEGIN
    d_ff_i: vhd_d_ff
            PORT MAP ( q_vector(i+1), clk, q_vector(i) );
  END GENERATE;
  --
  reg_so <= q_vector(0);
  --
END ARCHITECTURE struct_model ;
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
--
-- EOF
```

```
--
```

Listagem 2 - Registrador SIPO básico:

```
--
-- ============================================================
-- Registrador basico do tipo SIPO (Serial-Input Parallel-Output)
-- ============================================================
--
-- =====================
-- Arquivo: vhd_reg_sipo
-- =====================
--
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
ENTITY vhd_reg_sipo IS
  GENERIC(nos: NATURAL := 5);  -- Number of sections
  --
  PORT(reg_si:  IN BIT;
       clk   :  IN BIT;
       reg_po: OUT BIT_VECTOR ( (nos-1) DOWNTO 0 )  );
END ENTITY vhd_reg_sipo ;
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
--
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
ARCHITECTURE struct_model OF vhd_reg_sipo IS
--
COMPONENT vhd_d_ff IS
  PORT(d  :  IN BIT;
       clk:  IN BIT;
       q  : OUT BIT);
END COMPONENT vhd_d_ff ;
--
SIGNAL q_vector: BIT_VECTOR ( nos DOWNTO 0 );
--
BEGIN
  --
  q_vector(nos) <= reg_si;
  --
  shift_action:
  FOR i IN (nos-1) DOWNTO 0 GENERATE
  BEGIN
    d_ff_i: vhd_d_ff
            PORT MAP ( q_vector(i+1), clk, q_vector(i) );
  END GENERATE;
  --
  reg_po <= q_vector( (nos-1) DOWNTO 0 );
  --
```

```
END ARCHITECTURE struct_model ;
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
--
-- EOF
--
```

Listagem 3 - Registrador PISO básico, com deslocamento para a esquerda:

```
--
-- ================================================================
-- Registrador basico do tipo PISO (Parallel-Input Serial-Output),
-- com deslocamento para a esquerda.
-- ================================================================
--
-- ========================
-- Arquivo: vhd_reg_piso_shl
-- ========================
--
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
ENTITY vhd_reg_piso_shl IS
  GENERIC(nos: NATURAL := 5);  -- Number of sections
  --
  PORT(reg_pi      :  IN BIT_VECTOR ( (nos-1) DOWNTO 0 );
       reg_si      :  IN BIT;
       --
       load0_shift1:  IN BIT;
       clk         :  IN BIT;
       --
       reg_so      : OUT BIT
      );
END ENTITY vhd_reg_piso_shl ;
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
--
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
ARCHITECTURE struct_model OF vhd_reg_piso_shl IS
--
COMPONENT vhd_op_mux_2x1 IS
  PORT(mux_in0, mux_in1 : IN  BIT;
       sel0             : IN  BIT;
       mux_out          : OUT BIT);
END COMPONENT vhd_op_mux_2x1;

--
COMPONENT vhd_d_ff IS
  PORT(d  :  IN BIT;
       clk:  IN BIT;
       q  : OUT BIT);
```

```
END COMPONENT vhd_d_ff ;
--
SIGNAL d_vector: BIT_VECTOR ( (nos-1) DOWNTO  0 );
SIGNAL q_vector: BIT_VECTOR ( (nos-1) DOWNTO -1 );
--
BEGIN
  --
  q_vector(-1) <= reg_si;
  --
  ld_shft_action:
  FOR i IN (nos-1) DOWNTO 0 GENERATE
  BEGIN
    mux_i : vhd_op_mux_2x1
            PORT MAP
            (    reg_pi(i), q_vector(i-1),
              load0_shift1,   d_vector(i) );
    --
    d_ff_i: vhd_d_ff
            PORT MAP ( d_vector(i), clk, q_vector(i) );
  END GENERATE;
  --
  reg_so <= q_vector((nos-1));
  --
END ARCHITECTURE struct_model ;
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
--
-- EOF
--
```

Listagem 4 - Registrador PISO básico, com deslocamento para a direita:

```
--
-- ================================================================
-- Registrador basico do tipo PISO (Parallel-Input Serial-Output),
-- com deslocamento para a direita.
-- ================================================================
--
-- ========================
-- Arquivo: vhd_reg_piso_shr
-- ========================
--
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
ENTITY vhd_reg_piso_shr IS
  GENERIC(nos: NATURAL := 5);  -- Number of sections
  --
  PORT(reg_pi      :  IN BIT_VECTOR ( (nos-1) DOWNTO 0 );
       reg_si      :  IN BIT;
```

```
       --
       load0_shift1:  IN BIT;
       clk          :  IN BIT;
       --
       reg_so       : OUT BIT
      );
END ENTITY vhd_reg_piso_shr ;
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
--
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
ARCHITECTURE struct_model OF vhd_reg_piso_shr IS
--
COMPONENT vhd_op_mux_2x1 IS
  PORT(mux_in0, mux_in1 : IN  BIT;
       sel0             : IN  BIT;
       mux_out          : OUT BIT);
END COMPONENT vhd_op_mux_2x1;

--
COMPONENT vhd_d_ff IS
  PORT(d  :  IN BIT;
       clk:  IN BIT;
       q  : OUT BIT);
END COMPONENT vhd_d_ff ;
--
SIGNAL d_vector: BIT_VECTOR ( (nos-1) DOWNTO 0 );
SIGNAL q_vector: BIT_VECTOR (  nos    DOWNTO 0 );
--
BEGIN
  --
  q_vector(nos) <= reg_si;
  --
  ld_shft_action:
  FOR i IN (nos-1) DOWNTO 0 GENERATE
  BEGIN
    mux_i : vhd_op_mux_2x1
            PORT MAP
            (    reg_pi(i), q_vector(i+1),
              load0_shift1,   d_vector(i) );
    --
    d_ff_i: vhd_d_ff
            PORT MAP ( d_vector(i), clk, q_vector(i) );
  END GENERATE;
  --
  reg_so <= q_vector(0);
  --
END ARCHITECTURE struct_model ;
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
--
```

```
-- EOF
--
```

Listagem 5 - Registrador PIPO básico:

```
--
-- ================================================================
-- Registrador basico do tipo PIPO (Parallel-Input Parallel-Output)
-- ================================================================
--
-- =====================
-- Arquivo: vhd_reg_pipo
-- =====================
--
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
ENTITY vhd_reg_pipo IS
  GENERIC(nos: NATURAL := 5);  -- Number of sections
  --
  PORT(reg_pi:  IN BIT_VECTOR ( (nos-1) DOWNTO 0 );
       clk   :  IN BIT;
       reg_po: OUT BIT_VECTOR ( (nos-1) DOWNTO 0 )  );
END ENTITY vhd_reg_pipo ;
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
--
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
ARCHITECTURE struct_model OF vhd_reg_pipo IS
--
COMPONENT vhd_d_ff IS
  PORT(d  :  IN BIT;
       clk:  IN BIT;
       q  : OUT BIT);
END COMPONENT vhd_d_ff ;
--
SIGNAL q_vector: BIT_VECTOR ( (nos-1) DOWNTO 0 );
--
BEGIN
  --
  load_action:
  FOR i IN (nos-1) DOWNTO 0 GENERATE
  BEGIN
    d_ff_i: vhd_d_ff
            PORT MAP ( reg_pi(i), clk, q_vector(i) );
  END GENERATE;
  --
  reg_po <= q_vector( (nos-1) DOWNTO 0 );
  --
END ARCHITECTURE struct_model ;
```

```
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
--
-- EOF
--
```

Listagem 6 - Registrador PIPO básico, com deslocamento (esquerda e direita):

```
--
-- ================================================================
-- Registrador basico do tipo PIPO (Parallel-Input Parallel-Output),
-- com deslocamento (esquerda e direita)
-- ================================================================
--
-- ============================
-- Arquivo: vhd_reg_pipo_shl_shr
-- ============================
--
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
ENTITY vhd_reg_pipo_shl_shr IS
  GENERIC(nos: NATURAL := 5);  -- Number of sections
  --
  PORT(reg_pi      :  IN BIT_VECTOR ( (nos-1) DOWNTO 0 );
       reg_si_msb  :  IN BIT;
       reg_si_lsb  :  IN BIT;
       reg_po      : OUT BIT_VECTOR ( (nos-1) DOWNTO 0 );
       --
       load0_shift1:  IN BIT;
       shl0_shr1   :  IN BIT;
       --
       clk         :  IN BIT
      );
END ENTITY vhd_reg_pipo_shl_shr ;
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
--
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
ARCHITECTURE struct_model OF vhd_reg_pipo_shl_shr IS
--
COMPONENT vhd_op_mux_2x1 IS
  PORT(mux_in0, mux_in1 : IN  BIT;
       sel0             : IN  BIT;
       mux_out          : OUT BIT);
END COMPONENT vhd_op_mux_2x1;

--
COMPONENT vhd_d_ff IS
  PORT(d  :  IN BIT;
       clk:  IN BIT;
```

```
       q  : OUT BIT);
END COMPONENT vhd_d_ff ;
--
SIGNAL mux_sh_out: BIT_VECTOR ( (nos-1) DOWNTO  0 );
SIGNAL d_vector  : BIT_VECTOR ( (nos-1) DOWNTO  0 );
SIGNAL q_vector  : BIT_VECTOR (  nos    DOWNTO -1 );
--
BEGIN
  --
  q_vector(nos) <= reg_si_msb;
  q_vector( -1) <= reg_si_lsb;
  --
  ld_shft_action:
  FOR i IN (nos-1) DOWNTO 0 GENERATE
  BEGIN
    mux_sh_i: vhd_op_mux_2x1
              PORT MAP
              ( q_vector(i-1), q_vector(i+1),
                    shl0_shr1, mux_sh_out(i) );
    --
    mux_in_i: vhd_op_mux_2x1
              PORT MAP
              (     reg_pi(i), mux_sh_out(i),
                 load0_shift1,   d_vector(i) );
    --
    d_ff_i  : vhd_d_ff
              PORT MAP ( d_vector(i), clk, q_vector(i) );
  END GENERATE;
  --
  reg_po <= q_vector((nos-1) DOWNTO  0);
  --
END ARCHITECTURE struct_model ;
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --

--
-- EOF
--
```

Capítulo 17

Divisores de freqüência

17.1 Introdução

Divisor de freqüência é um circuito digital que recebe um sinal períodico com período T_0, ou freqüência $F_0 = 1/T_0$, e gera um ou mais sinais periódicos, com períodos $T_k = kT_0$, ou freqüências $F_k = 1/kT_0 = F_0/k$.

Nesse capítulo, são apresentados códigos para alguns divisores de freqüência.

17.2 Divisores de freqüência

- Os *flip-flops* que serão citados a seguir foram apresentados anteriormente nesse documento.
- A Listagem 1 apresenta um divisor de freqüência que utiliza *flip-flops* do tipo T, sem entrada seletora, sensíveis à transição positiva, com saída simples, sem controles extras.

Listagem 1 - Divisor de freqüência empregando *flip-flop* do tipo T:

```
--
-- ================================================
-- Divisor de frequencia empregando flip-flop do tipo T
-- ================================================
--
-- ==========================
-- Arquivo: vhd_div_freq_t_ff
-- ==========================
--
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
ENTITY vhd_div_freq_t_ff IS
  GENERIC(nos: NATURAL := 4);  -- Numero de secoes
  --
  PORT(clk_in :  IN BIT;
       clk_out: OUT BIT_VECTOR ( (nos-1) DOWNTO 0 )  );
END ENTITY vhd_div_freq_t_ff ;
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
--
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
```

```
ARCHITECTURE process_model OF vhd_div_freq_t_ff IS
--
COMPONENT vhd_t_no_sel_ff IS
  PORT(clk:  IN BIT;
       q  : OUT BIT);
END COMPONENT vhd_t_no_sel_ff ;
--
SIGNAL q_vector: BIT_VECTOR ( (nos) DOWNTO 0 );
--
BEGIN
  --
  q_vector(nos) <= clk_in;
  --
  div_action:
  FOR i IN (nos-1) DOWNTO 0 GENERATE
  BEGIN
    t_ff_i: vhd_t_no_sel_ff
            PORT MAP ( q_vector(i+1) , q_vector(i) );
  END GENERATE;
  --
  clk_out <= q_vector( (nos-1) DOWNTO 0 );
  --
END ARCHITECTURE process_model ;
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
--
-- EOF
--
```

Capítulo 18

Máquinas de Estados Finitos simples

18.1 Introdução

Nesse capítulo, são apresentados códigos para algumas Máquinas de Estados Finitos (*Finite-State Machines*) simples.

18.2 Máquinas de Estados Finitos simples

- A Listagem 1 apresenta um circuito digital sequencial, do tipo Mealy, que implementa um contador, com contagem crescente, de módulo 4, cujas saídas são codificadas em binário puro, que atende a um sinal de inicialização ativo em nível baixo ($\overline{RST}$). Os valores da contagem, em representação decimal, são os seguintes: z = (02/03, 04/07, 10/11, 16/19).

Listagem 1 - Contador, crescente, módulo 4, com *reset* (02/03, 04/07, 10/11, 16/19):

```
--
-- =======================================
-- Contador crescente, modulo 4, com reset
--   ( 02/03 , 04/07 , 10/11 , 16/19 )
-- =======================================
--
-- ====================================
-- vhd_counter_mealy_0203_0407_1011_1619
-- ====================================
--
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
ENTITY vhd_counter_mealy_0203_0407_1011_1619 IS
  PORT(
       clk , rstb :  IN BIT;
       x          :  IN BIT;
       z          : OUT BIT_VECTOR(4 DOWNTO 0)
      );
END ENTITY vhd_counter_mealy_0203_0407_1011_1619 ;
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
--
```

```
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
ARCHITECTURE fsm_model OF
             vhd_counter_mealy_0203_0407_1011_1619 IS
--
TYPE counter_state IS (q0, q1, q2, q3);
--
SIGNAL y , e : counter_state;
--
BEGIN
  --
  -- -- --  G & A  -- -- --
  --
  PROCESS (clk,rstb)
  --
  BEGIN
    --
    IF ( rstb = '0' )
      THEN
        y <= q0;
    --
    ELSIF ( clk'EVENT AND clk='1' )
      THEN
        y <= e;
    END IF;
    --
  END PROCESS ;
  --
  -- -- --  G & A  -- -- --
  --
  -- -- --   C C   -- -- --
  --
  PROCESS (y,x)
  --
  BEGIN
    CASE y IS
      --
      WHEN q0 =>
        IF (x = '0')
          THEN
            z <= "00010";  -- 02;
          ELSE
            z <= "00011";  -- 03;
        END IF;
        --
        e <= q1;
      --
      WHEN q1 =>
        IF (x = '0')
          THEN
```

```
            z <= "00100";  -- 04;
          ELSE
            z <= "00111";  -- 07;
        END IF;
        --
        e <= q2;
      --
      WHEN q2 =>
        IF (x = '0')
          THEN
            z <= "01010";  -- 10;
          ELSE
            z <= "01011";  -- 11;
        END IF;
        --
        e <= q3;
      --
      WHEN q3 =>
        IF (x = '0')
          THEN
            z <= "10000";  -- 16;
          ELSE
            z <= "10011";  -- 19;
        END IF;
        --
        e <= q0;
      --
    END CASE ;
  END PROCESS ;
  --
  -- -- --   C C   -- -- --
  --
END ARCHITECTURE fsm_model ;
-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- ----
--
-- EOF
--
```

Referências Bibliográficas

[HP81] F. J. Hill and G. R. Peterson. *Introduction to Switching Theory and Logical Design*. John Wiley, New York, NY, 3rd edition, 1981.

[IC08] I. V. Idoeta and F. G. Capuano. *Elementos de Eletrônica Digital*. Editora Érica, 40.ª edição edition, 2008.

[Rhy73] V. T. Rhyne. *Fundamentals of Digital Systems Design*. Prentice-Hall, Englewood Cliffs, NJ, 1973.

[Tau82] H. Taub. *Digital Circuits and Microprocessors*. McGraw-Hill, New York, NY, 1982. Em português: McGraw-Hill, Rio de Janeiro, 1984.

[TWM07] R. J. Tocci, N. S. Widmer, and G. L. Moss. *Sistemas Digitais: Princípios e Aplicações*. Prentice Hall, Pearson Education, 10.ª edição edition, 2007.

[Uye02] J. P. Uyemura. *Sistemas Digitais: Uma abordagem integrada*. Thomson Pioneira, São Paulo, SP, 2002.