

# Uma Introdução às Linguagens de Descrição de Hardware<sup>†</sup>

Edson T. Midorikawa (2001)  
revisado em 03/2007

*Este texto tem como objetivo apresentar de forma bem sucinta as Linguagens de Descrição de Hardware, mostrando suas características principais e introduzindo as linguagens VHDL e Verilog. É descrito o projeto de um circuito multiplicador binário para ilustrar os conceitos apresentados.*

## 1. Linguagens de Descrição de Hardware

Uma alternativa à *entrada esquemática* de um circuito digital em um sistema de projeto auxiliado por computador é utilizar a técnica de projeto de dispositivos lógicos programáveis (PLDs) com uma ferramenta de projeto baseado em texto ou *linguagem de descrição de hardware* (HDL). Exemplos de HDLs são o AHDL (*Altera Hardware Description Language*) e os padrões VHDL e Verilog.

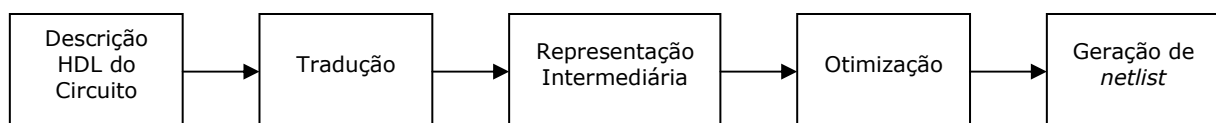
O projetista cria um arquivo de texto, seguindo certo conjunto de regras, conhecido como **sintaxe** da linguagem, e usa um compilador para criar dados de programação do dispositivo lógico programável (ou PLD). Esta descrição de hardware pode ser usada para gerar projetos hierárquicos, ou seja, um componente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

As HDLs têm uma grande semelhança às linguagens de programação, mas são especificamente orientadas à descrição da **estruturas** e do **comportamento** do hardware. Uma grande vantagem das HDLs em relação à entrada esquemática é que elas podem representar diretamente equações booleanas, tabelas verdade e operações complexas (p.ex. operações aritméticas).

Uma **descrição estrutural** descreve a interconexão entre os componentes que fazem parte do circuito. Esta descrição é usada como entrada para uma simulação lógica da mesma forma que uma entrada esquemática. Uma **descrição comportamental** descreve o funcionamento de cada um dos componentes do circuito.

Uma HDL pode ser usada na descrição em vários níveis do circuito em desenvolvimento. Partindo de uma descrição de alto nível, pode ser usada para refinar e particionar esta descrição em outras de nível mais baixo durante o processo de desenvolvimento. A descrição final deve conter componentes primitivos e blocos funcionais.

Uma grande razão para o uso de HDLs é a **síntese lógica**. Uma descrição em HDL em conjunto com uma biblioteca de componentes é usada por uma ferramenta de síntese para a geração automática de um circuito digital. Além disto, estas ferramentas incluem uma etapa de otimização da lógica interna do circuito gerado, antes da geração das estruturas internas de armazenamento, da lógica combinatória e da estrutura de conexão dos componentes (*netlist*). A figura 1 abaixo mostra um diagrama mostrando as etapas principais de síntese lógica.



**Figura 1 - Fluxo das etapas de alto nível da Síntese Lógica.**

<sup>†</sup> Texto anteriormente escrito para a experiência "Lógica Programável II" para a disciplina Laboratório Digital II.

Atualmente, as HDLs mais utilizadas são o VHDL e o Verilog. Ambas as linguagens são hoje padrões aprovados e publicados pelo IEEE (Instituto dos Engenheiros Elétricos e Eletrônicos), tendo, portanto várias ferramentas comerciais disponíveis. Esta padronização leva a uma grande vantagem no desenvolvimento de circuitos usando HDLs: a **portabilidade**. Visto que as ferramentas devem implementar as características padronizadas, fica muito fácil usar sistemas de desenvolvimento de fabricantes diferentes.

## 2. As Linguagens de Descrição VHDL e Verilog

Apresentamos nesta seção as duas linguagens de descrição de hardware mais usadas atualmente, VHDL e Verilog. Após um breve histórico, apresentamos as principais características de cada uma das HDLs.

### 2.1. VHDL

O nome VHDL é um acrônimo de *VHSIC Hardware Description Language*. Já o termo VHSIC é o acrônimo de *Very High Speed Integrated Circuit*. Assim podemos traduzir livremente o nome VHDL como "*linguagem de descrição de hardware para circuitos integrados de velocidade muito alta*".

A linguagem VHDL foi originalmente desenvolvida por empresas contratadas pelo governo americano e agora é um padrão requerido por todos os ASICs (*Application Specific Integrated Circuits* – circuitos integrados específicos da aplicação) projetados para o exército americano. Ele foi padronizado pelo IEEE em 1987 (Padrão 1076-1987 ou VHDL 87) e foi atualizado em 1993 (Padrão 1076-1993 ou VHDL 93). Os trabalhos do IEEE continuam e uma nova revisão está em desenvolvimento (VHDL 200x). Em 1999, foi lançado o padrão IEEE 1076.1, conhecido como VHDL-AMS, que oferece extensões para circuitos com sinais analógicos e digitais.

Todo arquivo VHDL requer ao menos duas estruturas: uma declaração de **entidade** (entity) e uma **arquitetura** (architecture). A declaração de entidade define os aspectos *externos* da função VHDL, isto é, os nomes das entradas e saídas e o nome da função. A arquitetura define os aspectos *internos*, isto é, como as entradas e saídas influem no funcionamento e como se relacionam com outros sinais internos.

Um exemplo de uma descrição VHDL é mostrado na figura 2 a seguir.

```
-- somador de 4 bits: descrição comportamental VHDL
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity somador_4bits is
  port(B, A : in std_logic_vector(3 downto 0);
        C0 : in std_logic;
        S : out std_logic_vector(3 downto 0);
        C4 : out std_logic);
end somador_4bits;

architecture comportamental of somador_4bits is
  signal soma : std_logic_vector(4 downto 0);
begin
  soma <= ('0' & A) + ('0' & B) + ("0000" & C0);
  C4 <= soma(4);
  S <= soma(3 downto 0);
end comportamental;
```

**Figura 2 – Exemplo de uma descrição VHDL.**

Os sinais de entrada e saída são do tipo `std_logic` e `std_logic_vector`. O tipo `std_logic` é definido no pacote `ieee.std_logic_1164` e pode assumir os valores 'U', 'X', '0', '1', 'Z', 'W', 'L', 'H' ou '-'. O tipo `std_logic_vector` representa um vetor de bits. O somador possui 3 sinais de entrada: A, B e C0, e 2 sinais de saída: S e C4. São usados dois operadores: + representa uma adição e &

representa uma concatenação de bits. Assim, '0' & A representa um vetor de 5 bits contendo '0' A(3) A(2) A(1) A(0).

Após a realização de uma operação de adição com os sinais de entrada, a descrição atribui o bit mais significativo de soma a C4 e os outros bits em S.

## 2.2. Verilog

A linguagem Verilog foi introduzida em 1985 pela Gateway Design Automation. Em 1989, a Gateway foi comprada pela empresa Cadence Design Systems, que tornou a linguagem de domínio público em maio de 1990 com a formação da Open Verilog International (OVI). Hoje o Verilog é um padrão IEEE, já tendo duas extensões ou modificações: Verilog-95 (padrão IEEE 1364-1995), Verilog 2001 (IEEE 1364-2001) e Verilog 2005 (IEEE 1364-2005), mas novos trabalhos continuam em andamento com o desenvolvimento do Verilog-AMS com suporte a sinais analógicos e digitais. O Verilog tem uma grande semelhança com a linguagem de programação C.

Uma característica interessante do Verilog é que, ao contrário do VHDL, há a diferenciação entre o uso de maiúsculas e minúsculas. Os identificadores podem conter qualquer seqüência de letras, dígitos, do símbolo '\$' e do símbolo '\_', onde o primeiro caractere deve ser uma letra ou o símbolo '\_'.

O Verilog oferece ao projetista os meios para descrever um sistema digital em vários níveis de abstração, e também suporta ferramentas de projeto para síntese lógica. Projetistas de hardware podem expressar suas idéias com **construções comportamentais**, deixando os detalhes para fases posteriores do projeto. Uma representação abstrata pode ser usada para explorar alternativas arquiteturais através de **simulações** e para detectar restrições de projeto antes do projeto detalhado. Com o detalhamento do projeto, são criadas descrições com **construções estruturais**.

A construção Verilog básica é o **module**. Os sinais de entrada e saída de um módulo (a sua interface) são designados usando **ports**. Cada sinal de um port pode ser declarado como **input**, **output** ou **inout**.

Um procedimento Verilog é sempre um comando **always** ou **initial**, uma tarefa (**task**) ou função (**function**). Os comandos dentro de um bloco seqüencial (comandos que aparecem entre um **begin** e um **end**) que é parte de um procedimento (**procedure**) são executados seqüencialmente na ordem em que aparecem. Mas os procedimentos são executados concorrentemente com outros procedimentos.

Um bloco seqüencial pode aparecer em um comando **always**, no caso em que o bloco deve ser executado repetidamente. Por outro lado, um comando **initial** especifica um bloco seqüencial que é executado apenas uma vez, no início de uma simulação.

Uma **task** é um tipo de procedimento, chamado por outro procedimento, que tem entradas e saídas, mas não tem um valor de retorno. Uma tarefa pode chamar outras tarefas e funções. Uma **function** é um procedimento usado em qualquer expressão, tem ao menos uma entrada, não tem saída e retorna um único valor. Uma função não pode chamar uma tarefa.

Verilog possui um tipo de controle de temporização que atrasa uma atribuição até que um evento específico ocorra. Um controle de evento é especificado por um @. Por exemplo, @(posedge clk) indica um evento referente a uma borda de subida do sinal clk. Um exemplo de uma descrição Verilog é mostrado na figura 3.

No módulo Verilog da figura 3, a lógica de adição é descrita por um único comando usando os operadores + e {}. O operador + representa a adição e {} representa uma operação chamada concatenação. A operação de adição deve ser efetuada sobre tipos de dados sem sinal. A concatenação combina dois sinais em um único sinal com um número de bits igual à soma do número de bits de cada sinal original. Por exemplo, {C4, S} representa o vetor de bits contendo C4 S[3] S[2] S[1] S[0] com  $1 + 4 = 5$  bits. Note que C4 é o quinto bit ou bit mais significativo da concatenação.

```
// somador de 4 bits: descrição comportamental Verilog

module somador_4bits(A, B, C0, S, C4)
  input[3:0] A, B;
  input C0;
  output[3:0];
  output C4;

  assign {C4, S} = A + B + C0;
endmodule
```

**Figura 3 – Exemplo de uma descrição Verilog.**

No módulo Verilog acima, a lógica de adição é descrita por um único comando usando os operadores + e {}. O operador + representa a adição e {} representa uma operação chamada concatenação. A operação de adição deve ser efetuada sobre tipos de dados sem sinal. A concatenação combina dois sinais em um único sinal com um número de bits igual à soma do número de bits de cada sinal original. Por exemplo, {C4, S} representa o vetor de bits contendo C4 S[3] S[2] S[1] S[0] com  $1 + 4 = 5$  bits. Note que C4 é o quinto bit ou bit mais significativo da concatenação.

### 3. Algoritmo para Multiplicação Binária

Para ilustrar a aplicação das linguagens de descrição de hardware, usaremos a operação de multiplicação binária. Nesta seção descrevemos dois algoritmos para efetuar esta operação e depois, na próxima seção, introduzimos o projeto de um circuito multiplicador.

Uma das operações mais custosas em um computador digital é a multiplicação binária. Normalmente pode ser implementada usando-se o algoritmo tradicional de multiplicação decimal que é ensinada nas aulas de Ensino Fundamental, também conhecido como “algoritmo de lápis e papel”. Apresentamos aqui os princípios fundamentais de operação do algoritmo tradicional e de um algoritmo melhorado.

#### 3.1. Algoritmo Tradicional

O algoritmo tradicional de multiplicação de dois números binários sem sinal é composto por sucessivos deslocamentos do multiplicando à esquerda (que constituem as parcelas do cálculo dos produtos parciais) e uma soma. Vamos executar este algoritmo com os números binários 1101 ( $13_{10}$ ) e 1011 ( $11_{10}$ ), como ilustrado na figura 4 abaixo.

13	1101	multiplicando
<u>11</u>	<u>1011</u>	multiplicador
	1101	
	1101	} produtos parciais
	0000	
	<u>1101</u>	
143	10001111	produto

**Figura 4 – Exemplo do algoritmo tradicional de multiplicação binária.**

A execução do algoritmo leva em consideração um bit do multiplicador de cada vez, com o bit menos significativo em primeiro lugar. Se o bit do multiplicador for 1, o multiplicando é copiado para ser somado posteriormente. Em caso contrário, o bit do multiplicador for 0, um valor nulo é copiado em seu lugar. Os números copiados em linhas sucessivas são deslocados à esquerda de uma posição em relação à linha anterior. Finalmente, os números são somados gerando o produto final.

Um circuito digital que implementa este algoritmo deve executar esta soma em etapas. Assim, quando um novo número for copiado, duas parcelas devem ser somadas gerando uma soma parcial. Ilustramos abaixo esta modificação no mesmo exemplo acima na figura 5.

Este algoritmo tradicional tem um requisito muito forte em relação à soma dos produtos parciais. Como são somadas  $n$  produtos parciais de  $n$  bits cada uma, é necessário um circuito somador com  $2n$  bits. Isto representa uma questão que deve ser levado em conta durante o projeto de um circuito de multiplicação binária.

13	1101	multiplicando
<u>11</u>	<u>1011</u>	multiplicador
	1101	} soma dos produtos parciais 2 a 2
	<u>1101</u>	
	100111	
	<u>0000</u>	
	100111	
143	<u>1101</u>	} produto
	10001111	

**Figura 5 – Exemplo do processamento do algoritmo para multiplicação binária.**

### 3.2. Algoritmo Melhorado

A deficiência do algoritmo tradicional de necessitar de um somador com  $2n$  bits pode ser resolvida com um algoritmo levemente diferente. O algoritmo melhorado, que apresentamos aqui, necessita apenas de um somador de  $n$  bits e de operações de deslocamento para a direita do produto parcial.

O algoritmo segue os seguintes passos:

1. Inicialmente o produto parcial é ajustado para 0 (zero);
2. Um bit do multiplicador é processado de cada vez, começando pelo bit menos significativo;
  - a. Se o bit sendo processado for 1 (um), o multiplicando é somado ao produto parcial e depois é realizado um deslocamento pela direita do produto parcial;
  - b. Se o bit sendo processado for 0 (zero), o produto parcial é apenas deslocado para a direita;
3. O bit de vai-um ("carry") do somador é armazenado em um flip-flop que deve estar conectado ao registrador deslocador contendo o produto parcial;
4. A soma deve ser realizada apenas nos  $n$  bits mais significativos de produto parcial.

Um exemplo de aplicação deste algoritmo é ilustrado abaixo na figura 6.

O diagrama de blocos de um circuito que implementa tal algoritmo será apresentado na seção 4 adiante, mais especificamente na figura 9.

13	1101	multiplicando
<u>11</u>	<u>1011</u>	multiplicador
	0000	valor inicial do produto parcial
	<u>1101</u>	soma multiplicando, bit do multiplicador é 1
	1101	
	0110 1	desloca para a direita
	<u>1101</u>	soma multiplicando, bit do multiplicador é 1
1	0011 1	
	1001 11	desloca para a direita
	0100 111	só desloca para a direita, bit do multiplicador é 0
	<u>1101</u>	soma multiplicando, bit do multiplicador é 1
1	0001 111	
	1000 1111	desloca para a direita
	}	
143	10001111	produto

Figura 6 – Exemplo do algoritmo melhorado para multiplicação binária.

## 4. Circuito Multiplicador Binário

Nesta seção apresentamos o projeto de um circuito multiplicador binário que implementa o algoritmo melhorado descrito acima.

### 4.1. Especificação do Multiplicador Binário

O Multiplicador Binário (MB) é responsável pela realização de uma multiplicação de dois números binários sem sinal de 4 bits, introduzidos separadamente no circuito através de uma única via de dados (CH0 a CH3). A operação é iniciada com o acionamento do sinal INICIAR (botão B1), e o resultado da operação com 8 bits (OUT) deve ser conectado a dois displays de saída. Os operandos da multiplicação são especificados para o MB pelos sinais ENTRA\_MULTIPLICANDO (CH6) e ENTRA\_MULTIPLICADOR (CH7).

Os sinais de entrada e saída do Multiplicador Binário são os seguintes:

- IN - via de dados de entrada, com quatro bits;
- OUT - via de dados de saída, com oito bits;
- INICIAR - sinal de controle utilizado iniciar a multiplicação;
- ENTRA\_MULTIPLICANDO - sinal de controle utilizado para especificar o multiplicando da operação;
- ENTRA\_MULTIPLICADOR - sinal de controle utilizado para especificar o multiplicador da operação.

A figura 7 abaixo mostra o MB com os sinais descritos anteriormente:

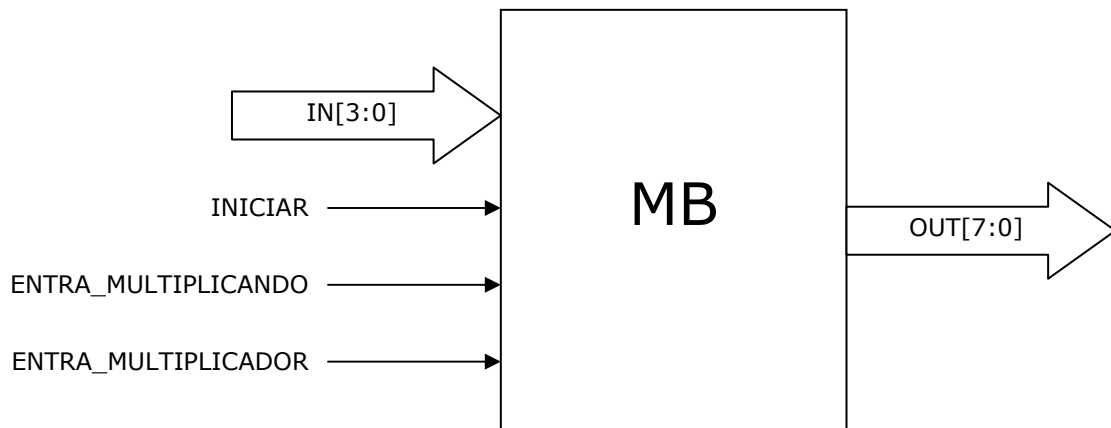


Figura 7 – Sinais de entrada e de saída do Multiplicador Binário.

A operação do circuito deve seguir os seguintes passos:

1. Acertar um valor binário na vida de dados de entrada (IN);
2. Ativar o sinal ENTRA\_MULTIPLICANDO;
3. Colocar outro valor na via de dados de entrada;
4. Ativar o sinal ENTRA\_MULTIPLICADOR;
5. Acionar o botão INICIAR para a execução da multiplicação binária;
6. Verificar resultado na via de dados de saída (OUT).

Caso se deseje repetir um valor anteriormente ajustado no circuito do MB, os passos 1 e 2 ou os passos 3 e 4 podem ser ignorados, visto que um registrador interno deve manter os valores do multiplicador e do multiplicando da operação anterior.

## 4.2. Projeto do MB com Circuitos Digitais

O Sistema Digital (SD) especificado na seção 4.1 pode ser projetado de diversas maneiras. Uma delas é baseada no particionamento proposto na metodologia apresentada na experiência "*Lógica Programável I*" da disciplina Laboratório Digital I. Portanto, vamos dividir o SD em dois módulos: o Fluxo de Dados (FD) e a Unidade de Controle (UC).

Como o FD deve conter os elementos de transformação e/ou armazenamento do SD, os sinais IN e OUT devem ser conectados a ele. Já os sinais de ajuste de valores dos operandos e de início de operação devem ser ligados à UC, pois estes sinais estão relacionados com o controle da operação do circuito. Alguns sinais de controle e de estado devem ser conectados entre o FD e a UC. A figura 8 abaixo ilustra a estrutura interna do SD do MB.

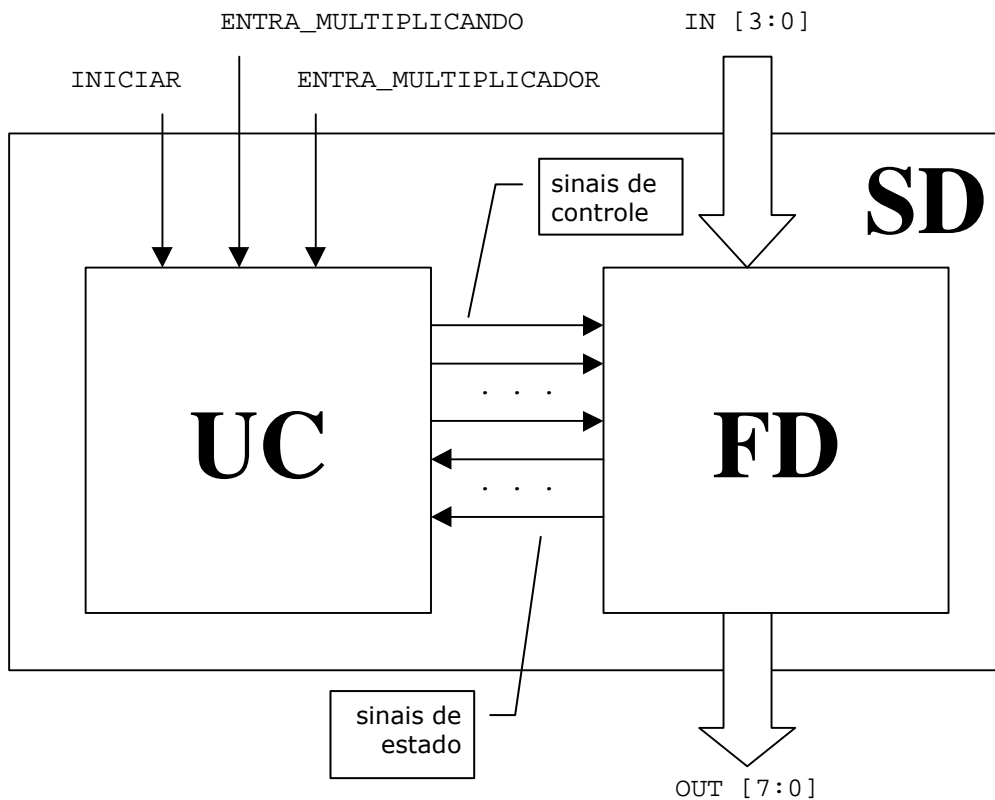


Figura 8 – Estruturação do Sistema Digital do Projeto do Multiplicador Binário.

### 4.2.1 Projeto do Fluxo de Dados

O fluxo de dados deve conter todos os elementos discutidos na seção 3. O diagrama lógico que representa o FD do MB é gerado a partir de um Diagrama de Blocos, como o da figura 9.

As etapas seguintes consistem da escolha dos componentes que realizam as funções dos blocos, da descrição do diagrama lógico.

Para o projeto do FD usando componentes discretos, podemos adotar os componentes da biblioteca da família 74XX:

- Contador P 74193
- Flip-flop C 7474
- Registrador B 74173/175
- Registrador deslocador A 74195
- Registrador deslocador Q 74194/195
- Somador binário 7483/283
- Detetor de zero 7402 (NOR)

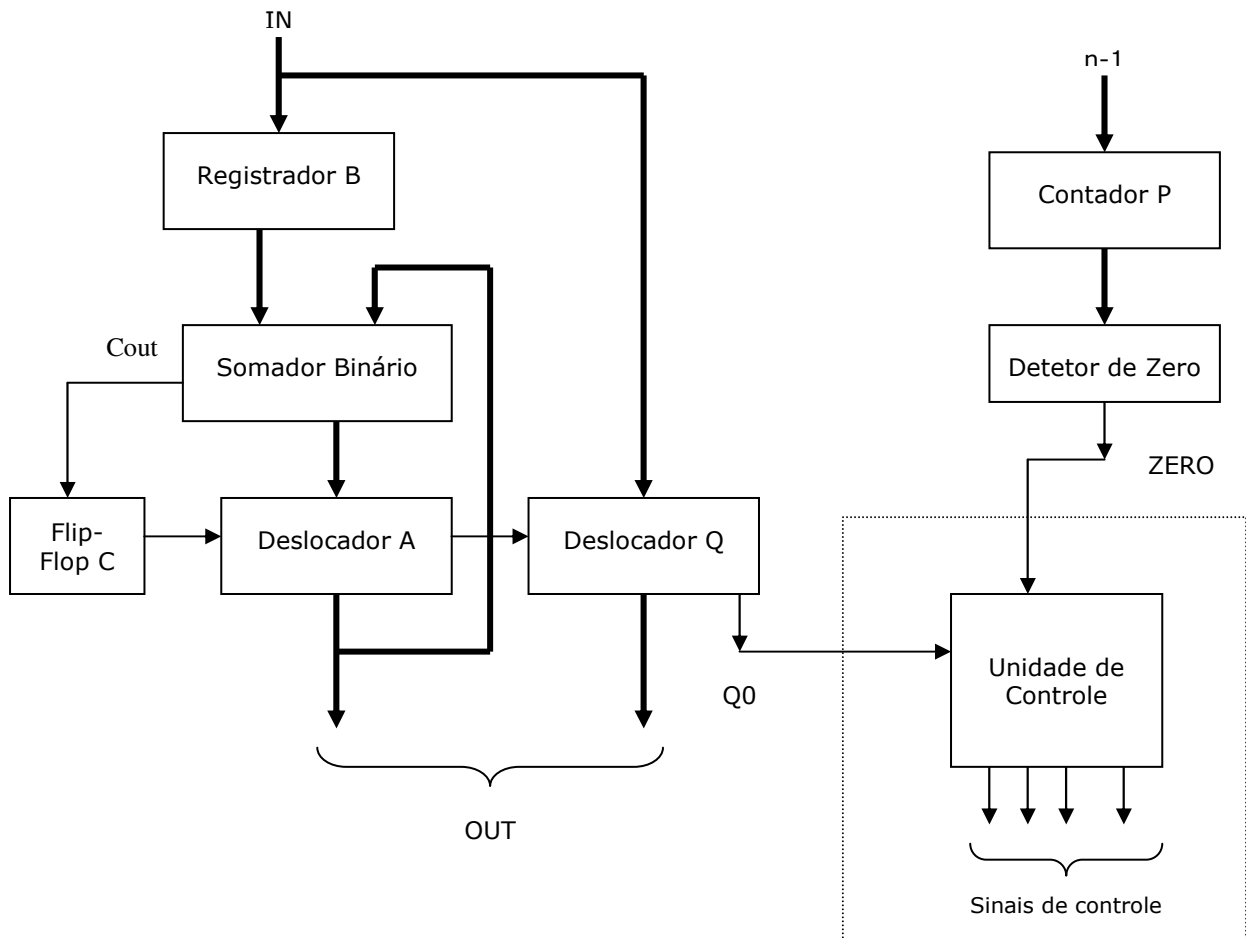


Figura 9 - Diagrama de Blocos do Multiplicador Binário.

#### 4.2.2. Projeto da Unidade de Controle

A unidade de controle pode ser projetada de vários modos. Adotaremos aqui a estratégia usada nas experiências de Laboratório Digital que usa o Diagrama ASM.

Para o MB, uma das soluções para a unidade de controle pode ser vista na figura 10. O diagrama ASM desta figura descreve o algoritmo de operação dos componentes do multiplicador, a partir dos sinais de controle externos do sistema digital.

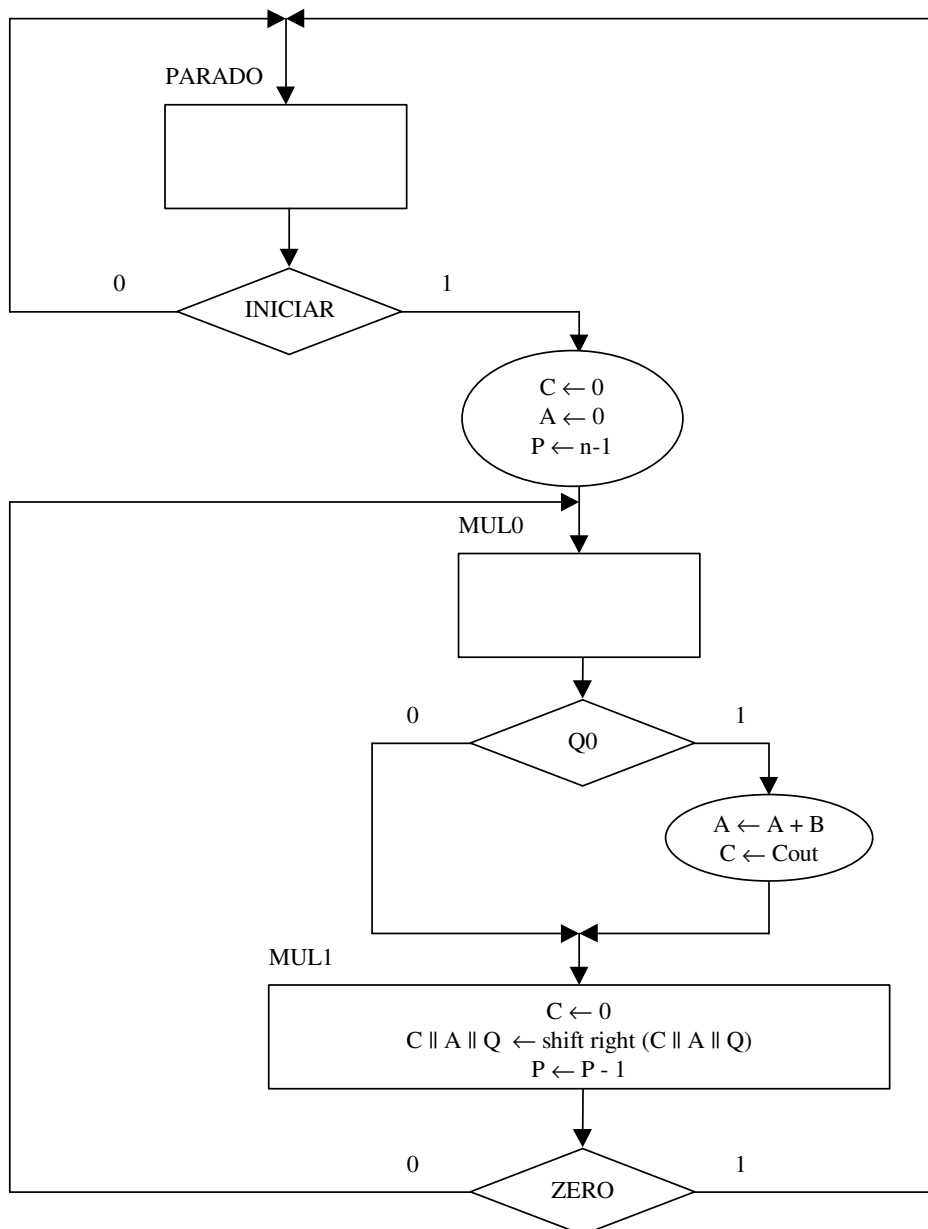


Figura 10 – Diagrama ASM da UC do Projeto do MB.

### 4.3. Projeto do MB com VHDL

Aqui vamos discutir a implementação do circuito do Multiplicador Binário usando-se a linguagem de descrição VHDL.

#### 4.3.1. Descrição VHDL do MB

O **Multiplicador Binário** pode ser representado durante o processo de projeto como uma descrição comportamental VHDL. Tal descrição apresentada abaixo representa o diagrama de blocos do SD apresentado na figura 9 e o diagrama ASM da figura 10. O código VHDL está contido na entidade `multiplicador_binario` e na arquitetura `comportamento_mult4`. A arquitetura contém duas atribuições e três processos. Cada um dos processos tem uma função distinta, mas podem interagir para a realização da operação de multiplicação.

No início da entidade, são definidas as entradas e as saídas do multiplicador. A seguir, no início da arquitetura são definidos os três estados de funcionamento do circuito pela declaração de tipo. Em seguida, sinais internos que gerarão registradores são definidos. Entre os sinais estão estado e prox\_estado para o controle, os registradores A, B, P e Q e o flip-flop C. Um sinal intermediário Z é declarado por conveniência.

Em seguida, o comando de atribuição força o sinal Z a ter valor 1 quando P contiver valor 0. O segundo comando de atribuição atribui a saída concatenada dos registradores A e Q para a saída do multiplicador MULT\_OUT. Isto é necessário para fazer com que os sinais A e Q possam ser usados internamente no circuito.

```

-- Multiplicador binário com n=4: descrição VHDL
-- adaptado de Mano & Kime "Logic and Computer Design Fundamentals"
-- Prentice-Hall, 2nd edition, 2000
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
entity multiplicador_binario is
  port(CLK, RESET, INICIAR, ENTRA_MULTIPLICANDO, ENTRA_MULTIPLICADOR: in std_logic;
        MULT_IN: in std_logic_vector(3 downto 0);
        MULT_OUT: out std_logic_vector(7 downto 0) );
end multiplicador_binario;

architecture comportamento_mult4 of multiplicador_binario is
  type tipo_estado is (PARADO, MUL0, MUL1);
  signal estado, prox_estado : tipo_estado;
  signal A, B, Q: std_logic_vector(3 downto 0);
  signal P: std_logic_vector(1 downto 0);
  signal C, ZERO: std_logic;
begin
  ZERO <= P(1) NOR P(0);
  MULT_OUT <= A & Q;

  registra_estado: process (CLK, RESET)
  begin
    if (RESET = '1') then
      estado <= PARADO;
    elsif (CLK'event and CLK = '1') then
      estado <= prox_estado;
    end if;
  end process;

  func_prox_estado: process (INICIAR, ZERO, estado)
  begin
    case estado is
      when PARADO =>
        if INICIAR = '1' then
          prox_estado <= MUL0;
        else
          prox_estado <= PARADO;
        end if;
      when MUL0 =>
        prox_estado <= MUL1;
      when MUL1 =>
        if ZERO = '1' then
          prox_estado <= PARADO;
        else
          prox_estado <= MUL0;
        end if;
    end case;
  end process;
end behavior;

```

**Figura 11 – Descrição VHDL do Multiplicador Binário.**

O restante da descrição envolve três **processos**. O *primeiro processo* descreve o funcionamento do registrador de estado e inclui o processamento de um sinal RESET e de clock. O *segundo processo* descreve a determinação do próximo estado do circuito, segundo interpretação do diagrama ASM da figura 4.4. Note que o tratamento dos sinais RESET e de clock não são mostrados neste ponto da descrição. Como pode ser visto, os sinais que afetam a determinação do próximo estado do sistema são INICIAR, ZERO e estado. O *terceiro processo* descreve a função do

fluxo de dados. Como as condições de funcionamento do multiplicador são determinadas pelo estado do sistema e pelos sinais de entrada, este processo está relacionado ao diagrama de blocos da figura 4.3. Os registradores do fluxo de dados são todos atribuídos sob controle do sinal CLK. Como o conteúdo dos registradores deve ser acertado antes da operação de multiplicação, não é necessário um comando de reset para estes registradores. O primeiro comando *if* controla a carga do multiplicando no registrador B a partir da entrada. O segundo comando *if* controla a carga do multiplicador no registrador Q.

```

func_fluxo_dados: process (CLK)
variable CA: std_logic_vector(4 downto 0);
begin
  if (CLK'event and CLK = '1') then
    if ENTRA_MULTIPLICANDO = '1' then
      B <= MULT_IN;
    end if;
    if ENTRA_MULTIPLICADOR = '1' then
      Q <= MULT_IN;
    end if;
    case estado is
      when PARADO =>
        if INICIAR = '1' then
          C <= '0';
          A <= "0000";
          P <= "11";
        end if;
      when MUL0 =>
        if Q(0) = '1' then
          CA := ('0' & A) + ('0' & B);
        else
          CA := C & A;
        end if;
        C <= CA(4);
        A <= CA(3 downto 0);
      when MUL1 =>
        C <= '0';
        A <= C & A(3 downto 1);
        Q <= A(0) & Q(3 downto 1);
        P <= P - "01";
      end case;
    end if;
  end process;
end comportamento_mult4;

```

**Figura 12 – Continuação da descrição VHDL do Multiplicador Binário.**

A operação do multiplicador envolve uma série de transferências de registradores a cada estado. Estas transferências são descritas através de um comando *case*, dependendo do estado do sistema, da entrada INICIAR e dos sinais internos Q(0) e ZERO. O comando de adição no estado MUL0 requer algum esforço. Primeiro, para realizar uma adição em vetores *std\_logic*, um comando **use** para o pacote *ieee.std\_logic\_unsigned.all* aparece antes da declaração da entidade. Para transferir o bit de vai-um ("carry") Cout do somador, é executada uma soma de 5 bits com 0's acrescentados à esquerda de A e B e o resultado é atribuído à variável CA de 5 bits. O operador & usado em C&A designa uma operação de concatenação de registradores. No estado MUL1, o deslocamento para a direita é realizado usando-se a concatenação dos bits adequados dos registradores. No final, P é decrementado com a operação de subtração com uma constante de dois bits de valor 1.

## 4.4. Projeto do MB com Verilog

Aqui vamos discutir a implementação do circuito do Multiplicador Binário usando-se a linguagem de descrição Verilog.

### 4.4.1. Descrição Verilog do MB

O **Multiplicador Binário** pode ser representado durante o processo de projeto como uma descrição comportamental Verilog. Tal descrição apresentada abaixo representa o diagrama de blocos do SD apresentado na figura 9 e o diagrama ASM da figura 10. O código Verilog está contido

no módulo multiplicador\_binario. A descrição contém duas atribuições e três processos. Cada um dos processos tem uma função distinta, mas podem interagir para a realização da operação de multiplicação.

No início da descrição, são definidas as entradas e as saídas do multiplicador. A seguir, a declaração de parâmetro define os três estados de funcionamento do circuito e seus códigos binários. Em seguida, sinais internos de tipo registrador são definidos. Entre os sinais estão estado e prox\_estado para o controle, os registradores A, B, P e Q e o flip-flop C. As especificações do circuito de relógio definem que estes componentes serão ativados na borda de subida do sinal de clock.

```
// Multiplicador binário com n=4: descrição Verilog
// adaptado de Mano & Kime "Logic and Computer Design Fundamentals"
// Prentice-Hall, 2nd edition, 2000

module multiplicador_binario (CLK, RESET, INICIAR, ENTRA_MULTIPLICANDO,
    ENTRA_MULTIPLICADOR,
    MULT_IN, MULT_OUT);
    input CLK, RESET, INICIAR, ENTRA_MULTIPLICANDO, ENTRA_MULTIPLICADOR;
    input [3:0] MULT_IN;
    output [7:0] MULT_OUT;
    reg [1:0] estado, prox_estado, P;
    parameter PARADO = 2'b00, MUL0 = 2'b01, MUL1 = 2'b10;
    reg [3:0] A, B, Q;
    reg C;
    wire ZERO;

    assign ZERO = ~| P;
    assign MULT_OUT = {A,Q};

    // registra estado
    always@(posedge CLK or posedge RESET)
    begin
        if (RESET == 1)
            estado <= PARADO;
        else
            estado <= prox_estado;
    end

    // função próximo estado
    always@(INICIAR or ZERO or estado)
    begin
        case (estado)
            PARADO:
                if (INICIAR == 1)
                    prox_estado <= MUL0;
                else
                    prox_estado <= PARADO;
            MUL0:
                prox_estado <= MUL1;
            MUL1:
                if (ZERO == 1)
                    prox_estado <= PARADO;
                else
                    prox_estado <= MUL0;
        endcase
    end
end
```

**Figura 13 – Descrição Verilog do Multiplicador Binário.**

Um sinal intermediário Z do tipo wire é declarado por conveniência. Em seguida, o comando de atribuição força o sinal Z a ter valor 1 quando P contiver valor 0. Esta atribuição usa o operador OR (|) como uma operação de redução. Redução é a aplicação de uma operação em um sinal do tipo wire ou registrador que combina bits individuais. Neste caso, a aplicação da operação OR sobre P faz com que todos os bits de P sejam combinados. Como a operação OR é precedida por ~, a operação definida pelo comando é NOR. O segundo comando de atribuição atribui a saída concatenada dos registradores A e Q para a saída MULT\_OUT. É uma conveniência para fazer com que a saída do circuito seja composta por uma única estrutura.

O restante da descrição envolve três **processos**. O *primeiro processo* descreve o funcionamento do registrador de estado e inclui o processamento de um sinal RESET e de clock. O *segundo processo* descreve a determinação do próximo estado do circuito, segundo interpretação do diagrama ASM da figura 4.4. Note que o tratamento dos sinais RESET e de clock não são mostrados neste ponto da descrição. Como pode ser visto, os sinais que afetam a determinação do próximo estado do sistema são INICIAR, ZERO e estado. O *terceiro processo* descreve a função do fluxo de dados. Como as condições de funcionamento do multiplicador são determinadas pelo estado do sistema e pelos sinais de entrada, este processo está relacionado ao diagrama de blocos da figura 4.3. O primeiro comando *if* controla a carga do multiplicando no registrador B a partir da entrada. O segundo comando *if* controla a carga do multiplicador no registrador Q.

```
// função fluxo de dados
always@(posedge CLK)
begin
  if (ENTRA_MULTIPLICANDO == 1)
    B <= MULT_IN;
  if (ENTRA_MULTIPLICADOR == 1)
    Q <= MULT_IN;
  case (estado)
    PARADO:
      if (INICIAR == 1)
        begin
          C <= 0;
          A <= 4'b0000;
          P <= 2'b11;
        end
    MUL0:
      if (Q[0] == 1)
        {C, A} = A + B;
    MUL1:
      begin
        C <= 1'b0;
        A <= {C, A[3:1]};
        Q <= {A[0], Q[3:1]};
        P <= P - 2'b01;
      end
  endcase
end
endmodule
```

**Figura 14 – Continuação da descrição Verilog do Multiplicador Binário.**

A operação do multiplicador envolve uma série de transferências de registradores a cada estado. Estas transferências são descritas através de um comando *case*, dependendo do estado do sistema, da entrada INICIAR e dos sinais internos Q(0) e ZERO. O comando de adição no estado MUL0 usa a concatenação de C e A para que o bit de vai-um ("carry") Cout do somador seja carregado em C. A linguagem Verilog permite o uso de dois operandos de 4 bits e um resultado de 5 bits. As operações de deslocamento para a direita no estado MUL1 são especificados usando-se a operação de concatenação dos bits adequados de A e Q. No final, P é decrementado com a operação de subtração com uma constante de dois bits de valor 1.

## 5. BIBLIOGRAFIA

1. ALTERA. Manuais diversos da Altera.
2. ASHENDEN, P. J. **The VHDL cookbook**. Dept. Computer Science, University of Adelaide, South Australia. July 1990. (pode ser encontrado na Internet)
3. ASHENDEN, P. J. **The designer's guide to VHDL**. 2nd ed., Morgan Kaufmann, 2002.
4. DUECK, R. K. **Digital Design with CPLD Applications and VHDL**. Delmar, 2001.
5. HYDE, D. C. **Handbook on Verilog HDL**. Computer Science Department, Bucknell University. August 1995.
6. MANO, M. M.; KIME, C. R. **Logic and computer design fundamentals**. 3<sup>rd</sup> edition, New Jersey: Prentice-Hall, 2004.
7. MANO, M. M. **Digital design**. 3<sup>rd</sup> edition, New Jersey: Prentice-Hall, 2002.

8. QUALIS. **Cartões de referência de VHDL e Verilog**. Qualis Design Co., 2000.
9. TOCCI, R. J.; WIDMER, N. S.; MOSS, G. L. **Digital Systems**: principles and applications. 10<sup>th</sup> ed., Prentice-Hall, 2007.
10. WAKERLY, J. F. **Digital design**: principles and practices. 4<sup>th</sup> ed., New Jersey: Prentice-Hall, 2006.