Universidade Federal de Uberlândia
UFU
Curso de Ciência da Computação
AOC2

Relatório de HDL

Aluno: Antonio Carlos Neto
Professor: Alexandro

Setembro
2017

Conteúdo

1 Introdução

Linguagem de Descrição de Hardware(LDH ou HDL), é um tipo possível de linguagem no qual descreve como é o funcionamento de um circuito e verificar seu funcionamento por meio de simulações.LDHs são usados para descrever especificações de um sistema executáveis de algum pedaço de hardware programável, como FPGA (Field Programmable Gate Array) ou um dispositivo ASIC (Aplication Specific Integrated Circuit). Na atualidade existe várias linguagens LDHs, como: VHDL(Very Hardware Description Language), Verilog, SystemC, HardwareC, ABEL(Advanced Boolean Equation Language), Silage, EDIF, SystemVeriLog.

O surgimento do Very High Speed Integrated Circuits(VHSIC), em torno dos anos 1980, foi implementado em torno KARL ABL e por um consórcio internacional financiado pela Comissão da União Europeia. Com esse surgimento conseguimos algumas vantagens, a principal é o time-to-market, ou seja, um produto leva muito menos tempo para ser produzido.

2 VHDL

Very Hardware Description Language(VHDL) é uma linguagem de descrição de hardware com ênfase em circuitos integrados de altíssima velocidade(VHSIC), tendo como característica seus comandos ocorrendo simultaneamente com exceção de processos. VHDL permite simular e verificar o comportamento de sistemas digitais e descrever hardware em diversos níveis de abstração, como Comportamental e Transferência entre registradores (RTL).VHDL provê mecanismos para modelar a concorrência e sincronização que ocorrem a nível físico no hardware. Algumas vantagens são:

•Time-To-Market: Se há dez anos atrás um produto demorava 6 meses para ser desenvolvido, mas permanecia no mercado por 2 anos, hoje um produto não permanece mais de 18 meses, logo o seu desenvolvimento deve levar bem menos tempo.

•Menor Ciclo e Custo de Desenvolvimento: devido à eliminação de geração, manutenção de esquemáticos e pela diminuição de erros de desenvolvimento pelo uso de simulação nos ciclos iniciais do projeto.

•Aumento de Qualidade no Desenvolvimento: VHDL facilita o rápido experimento com diferentes arquiteturas e técnicas de implementação, e pela capacidade das ferramentas de síntese otimizarem um projeto tanto para área mínima quanto para velocidade máxima;

•Gerenciamento do Projeto: Projetos em VHDL facilitam a estruturação de componentes (topdown), facilitam a documentação e são necessárias menos pessoas para desenvolver e verificar, sendo também mais simples modificar o projeto.

VHDL também possui desvantagens, pois a cultura deve ser alterada para acomodarmos com essa nova tecnologia, possibilitando uma dificuldade de aprendizado; A otimização em comparação com um circuito esquemático, é menos eficiente.

Um projeto pode ser dividido em quatro partes, Package(constantes e bibliotecas), Entity(pinos de entrada e saída), Architecture(implementações do projeto), Configuration(define as arquiteturas que serão utilizadas). Packade é usado quando precisa-se usar um comando que não existe nas bibliotecas padrão. Deve ser definido antes do inicio da entity. Para usar a package é necessário usar duas declarações: library use. O package mais conhecido é o STD_LOGIC_1164 da IEEE por conter a maioria dos comandos adicionais usados na linguagem. Entity é a parte principal do projeto, é a interface do Sistema que descreve as entradas e saídas. Composta de duas partes: parameters e connections. Parameters refere-se aos parâmetros, exemplo largura de barramento, são declarados como generics. Connections por sua vez, refere-se como ocorre a transferência de informações, são declarados como

ports. Architecture é o corpo do sistema, onde são feitas as atribuições, operações, comparações, dentre outros, sendo que pode existir varias arquiteturas para uma mesma entity. Process é uma diretiva usada quando se quer fazer uma lista de operações a serem executadas, implementada dentro de architecture possuindo uma forma estruturada.

Operadores e Expressões:

- Operadores Lógicos: Existe os operadores and, or, nand, nor, xor e xnor que necessitam dois operandos, enquanto operador not precisa de apenas um operando.
- Deslocamento: Operação feita somente em vetores, composta dois operandos, um sendo o array e o outro um integer, que é o número de posições a serem deslocadas. Operações possíveis:
    1. shift left logical (deslocamento lógico a esquerda);
    2. shift right logical (deslocamento lógico a direita);
    3. shift left arithmetic (deslocamento aritmético a esquerda);
    4. shift right arithmetic (deslocamento aritmético a direita);
    5. rotate left logical (rotacionamento lógico a esquerda);
    6. rotate right logical (rotacionamento lógico a direita).
- Operadores aritméticos: Usados para realizar alguma operação matemática(aritmética), necessitando de dois operandos. Operações possíveis:
    1. + : soma ou identidade;
    2. - : subtração ou negação;
    3. * : multiplicação;
    4. / : divisão;
    5. mod : módulo;
    6. rem : resto da divisão;
    7. abs : valor absoluto;
    8. ** : exponenciação.
- Comparações: Utilizados para fazer uma comparação entre dois operandos resultando um valor booleano.
    1. = : igual;
    2. /= : diferente;

3. < : menor;
4. <= : menor ou igual;
5. > : maior;
6. >= : maior ou igual;

Tipo de dados mais utilizados:

- bit: Assume valores { "0"e "1"}. bit_vector: tipo que designa conjunto de bits. Exemplo: "10001100"ou x"8C"
- std_logic: Semelhante ao tipo bit, mas permite assumir mais valores que permitem melhor análise na simulação. Assume valores { "0", "1", "X", "L", "l", "H", "h", "Z", "U"}. std_logic_vector: tipo que designa um conjunto de bits std_logic.
- boolean: Assume valores { true, false }. Útil apenas para descrições abstratas, onde um sinal só pode assumir dois valores.
- Physical: Representam uma medida: voltagem, capacitância, tempo. Tipos pré-definidos: fs, ps, ns, um, ms, sec, min, hr.
- Real: Utilizado durante desenvolvimento da especificação. Exemplos: -1.0 / +2.35 / 37.0 / -1.5E+23.
- Inteiros: Exemplos: +1 / 1232 / -1234. Não é possível realizar operações lógicas sobre inteiros (deve-se realizar a conversão explícita).
- Character: VHDL não é "case sensitive", exceto para caracteres. Valor entre aspas simples: 'a', 'x', '0', '1'. String: tipo que designa um conjunto de caracteres exemplo: "vhdl".

Exemplos de Código em VHDL:

- Decodificador BCD para 7 segmentos:

```
entity decodificador_7seg is
port(
bcd : IN BIT_VECTOR(3 DOWNTO 0);
segmentos: OUT BIT_VECTOR(6 DOWNTO 0));
end decodificador_7seg;

architecture teste_4 of decodificador_7seg is
begin
```

```
WITH bcd SELECT
segmentos ¡= ”1111110”WHEN ”0000”,
”0110000”WHEN ”0001”,
”1101101”WHEN ”0010”,
”1111001”WHEN ”0011”,
”0110011”WHEN ”0100”,
”1011011”WHEN ”0101”,
”1011111”WHEN ”0110”,
”1110000”WHEN ”0111”,
”1111111”WHEN ”1000”,
”1111011”WHEN ”1001”,
”1111110”WHEN OTHERS;
end teste_4;
```

- Somador de 4 bits completo utilizando portas lógicas:

```
–libraries to be used are specified here
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

–entity declaration with port definitions
entity rc_adder is
port(
num1 : in std_logic_vector(3 downto 0); –4 bit input 1
num2 : in std_logic_vector(3 downto 0); – 4 bit input 2
sum : out std_logic_vector(3 downto 0); – 4 bit sum
carry: out std_logic – carry out.
);
end rc_adder;

–architecture of entity
architecture Behavioral of rc_adder is
–temporary signal declarations(for intermediate carry's).
signal c0,c1,c2,c3 : std_logic := '0';
begin

–first full adder
sum(0) ¡= num1(0) xor num2(0); –sum calculation
c0 ¡= num1(0) and num2(0); –carry calculation

–second full adder
```

sum(1) ¡= num1(1) xor num2(1) xor c0;
c1 ¡= (num1(1) and num2(1)) or (num1(1) and c0) or (num2(1) and c0);

–third full adder
sum(2) ¡= num1(2) xor num2(2) xor c1;
c2 ¡= (num1(2) and num2(2)) or (num1(2) and c1) or (num2(2) and c1);

–fourth(final) full adder
sum(3) ¡= num1(3) xor num2(3) xor c2;
c3 ¡= (num1(3) and num2(3)) or (num1(3) and c2) or (num2(3) and c2);

–final carry assignment
carry ¡= c3;

end Behavioral;

3 VeriLog

Verilog HDL is a hardware description language used to design and document electronic systems. Verilog HDL allows designers to design at various levels of abstraction. It is the most widely used HDL with a user community of more than 50,000 active designers.

Verilog foi criado em meados da década de 1980, o tamanho típico do projeto era da ordem de 5 a 10 mil portões, o método típico de criação de design era o uso de ferramentas gráficas, e as ferramentas de simulação eram o início da verificação do nível do portão. A Verilog incluiu capacidades que permitiram uma nova geração de tecnologias EDA (Electronic Design Automation) que envolveram a RTL (Register Tranfer Level) para o sistema. Consequentemente, Verilog tornou-se o idioma preferido para os designers IC.

Durante a década de 1990, a linguagem da Verilog continuou sua evolução, mas foi até 2001 que o IEEE criou o padrão com novas extensões em linguagem. Algumas das novas capacidades do padrão eram, por exemplo, matrizes multidimensionais, variáveis automáticas e geração de indicação. Hoje, a maioria das ferramentas de EDA suportam o padrão de 2001.

Em 2005, o IEEE cria um novo padrão que apresenta pequenas mudanças, mas nesse mesmo ano é criado o padrão System Verilog que é usado como linguagem de design, mas também como uma linguagem de verificação e o conceito de Descrição de Hardware e Idioma de Verificação) uma definição ambígua deste idioma é a combinação de Verilog com C ++.

Notações da Linguagem:

- Estados do Bit:
    1. 1 significa Um logico;
    2. 0 significa Zero logico;
    3. Z significa Alta Impedância, flutuando, não conectado.
    4. X significa Valor lógico desconhecido ou indefinido. Todo estado X deve estar em 0, 1, Z ou em transição entre esses estados. Não podemos setar um bit para o estado X.
- Operadores unitários:
    1. Operador soma "+", a = b + c (adição binaria);
    2. Operador subtração "−", a = b - c (subtração binaria);
- Operadores de todos os bits:

1. Operador OR "| ", a = |b;
2. Operador AND "&", a = &b;
3. Operador XOR "OR", a = ORb;
4. Operador NOR "∼|", a = ∼|b;
5. Operador NAND "∼&", a = ∼&b;
6. Operador XNOR "∼OR—", a = ∼ORb;

- Operadores de bit-wise:

1. Operador NOT "∼", a = ∼b;
2. Operador OR "|", a = b|c;
3. Operador AND "&", a = b&c;
4. Operador XOR "OR", a = bORc;
5. Operador NOR "∼|", a = b∼|c;
6. Operador NAND "∼&", a = b∼&c;
7. Operador XNOR "∼OR", a = b∼ORc;

- Operadores de igualdade:

1. Operador de atribuição "=", a = b;
2. Operador de comparação de igualdade "==", a == b;
3. Operador de comparação de igualdade e inclui os estados Z e X "==="a===b;
4. Operador de comparação de diferença "!=", a !=b;
5. Operador de comparação de diferença "e inclui os estados Z e X !==", a !==b;