VERILOG

Monitoria Infra-Estrutura de Hardware 2011.2

Álvaro João – ajss
Húgaro Bernardino – hbb
Fred Rabelo - ferrf
Leonardo Leandro – lsl2
Jéssica de Carvalho – jcb
Silas Pedrosa - sfgp

www.cin.ufpe.br/~if674cc

Histórico

- Antes de Verilog, o design de sistemas digitais era baseado em esquemáticos.
- Para projetos grandes, esta alternativa era inviável.
- Com o surgimento de Verilog, criou-se uma nova forma de "enxergar" circuitos lógicos.
- Conceitos de linguagens de programação tradicionais aplicados à descrição de hardware.

Tipos de dados

- Wire – Utilizado para conectar dois pontos.
- Reg – Utilizado para armazenar valores
  - **Reg não é o mesmo que registrador.**

Vetores e arrays

- Os tipos de dados podem ser agrupados em vetores. A notação para isto é feita através de uso do intervalo da forma [*left*:*right*], onde o bit correspondente ao *left* sempre é o mais significativo.
- wire [7:0] bus; // barramento de 8 bits, bus[7] = bit mais significativo
- reg [63:0] m; // registrador de 64 bits, m[63] = bit mais significativo

- Definindo entradas
  - input clock;
- Definindo saídas
  - output reg saida;
- Sinais bidirecionais
  - inout sinal;
- Sinais de vetores de bits
  - input[7:0] entrada;

Operadores

- Semelhantes aos de linguagens de programação tradicionais.
- Aritméticos
  - +, -, *, /, %
- Lógicos
  - !, &&, ||
- Comparação
  - >, <, >=, <=, ==, !=
- Shift
  - >> ,<<
- Concatenação
  - {}

Module

- Representação de "caixas pretas" (componentes) de um design.
- Composto por entradas, saídas e uma lógica interna.
- Análogo a **funções com retorno** em linguagens de programação.

Uso de module 1/3

```
module nome ( /*<lista de portas>*/ );
// <declaração de portas>
// < lista de parametros>
// <declaração de wires, regs e outras variáveis>
// <descrição de fluxo de dados com o comando assign>
// <instanciação de módulos>
// <blocos de descrição de comportamento>
endmodule
```

- A declaração de portas descreve se os sinais são de entrada, saída ou ambos através das palavras-chave input, output, inout, respectivamente.

Uso de module 2/3

- As seguintes regras de conexão devem ser levadas em consideração:
- 1 - Internamente, todos os sinais de entrada devem ser do tipo **wire**. Externamente, as entradas podem estar conectadas a sinais tipo wire ou reg.
- 2 - Sinais de saída devem ser do tipo reg. Externamente, devem estar associados a uma variável do tipo wire.
- 3 - Sinais do tipo inout devem ser do tipo wire e externamente associados a uma variável do tipo wire.

Exemplo de uso de module 3/3

```
1  module AluSrcA( AluSrcA, in_pc, in_regA, out_AluSrcA);
2
3      input AluSrcA; //seletor
4      input [31:0] in_pc;
5      input [31:0] in_regA;
6      output [31:0] out_AluSrcA;
7
8  endmodule
9
```

If-Else

```
1  if(sinal == 1) begin
2      var = var + 1;
3   end else begin
4      var = 0;
5   end
6
```

Case

```
1  case (AluSrcA)
2              0: out_AluSrcA = in_pc;
3              1: out_AluSrcA = in_regA;
4              default: out_AluSrcA = 0;
5   endcase
6
```

- Ao contrário de C e Java, a instrução case em Verilog não necessita de break. Assim que a condição igualar a um dos valores, as instruções correspondentes são executadas e a execução prossegue a partir do endcase.

While

```
1  while(incrementar) begin
2      total = total + 1;
3  end
4
```

Repeat

```
1  repeat (16) begin
2      $display ("Valor atual de i: %d", i);
3      i = i + 1;
4  end
5
```

For

```
1   for (i = 0; i < 16; i = i +1) begin
2         $display ("Valor atual de i: %d", i);
3     end
4
```

- Em verilog os operadores ++ e -- não são usados, devendo por isso utilizar-se i = i + (ou -) 1;

Always

- Delimita bloco de código que será executado sempre que um certo sinal for ativado.
- Possui uma lista de sensibilidade para determinar quais sinais ativarão a execução do código.

Uso de always

```
1  always @ (posedge clk or posedge rst) begin
2      if (rst) begin
3          count <= 0;
4      end else begin
5          count <= count + 1;
6      end
7  end
```

Assign

- Usado para modelar circuitos combinacionais, onde as saídas mudam quando muda uma das entradas.
- É executado continuamente.
- Não possui lista de sensibilidade.

```
1    assign out = (enable) ? data : 0'd;
2
```

Initial

- Bloco de código executado apenas uma vez, no início da simulação.

```
1   initial begin
2       clk = 0;
3       reset = 0;
4       req_0 = 0;
5       req_1 = 0;
6   end
7
```

Dicas e recomendações

- Para o projeto, é recomendável utilizar sempre output reg.
- É uma boa prática e facilita a máquina de estados usar parameter para definir nomes para números que serão muito usados. Ex.:

- parameter Reset = 6'b000000;