SystemVerilog para Projetos

Curso Presencial IFPB/UFAL

Marcos Morais – Professor DEE
Projetos de Excelência em Microeletrônica – PEM
Centro de Engenharia Elétrica e Informática
Universidade Federal de Campina Grande

Sumário

- Níveis de abstração;
- Projetos bottom-up e top-down;
- Diagrama de blocos;
- Separação entre caminho de dados (datapath) e controle;
- Linguagens de descrição de hardware;
- Fluxos de projeto em microeletrônica e FPGA;
- Subconjunto sintetizável;
- Descrição comportamental e nível RTL;
- Módulos;
- Tipos de dados e valores literais;
- Operadores e expressões;
- Sentenças procedurais e de controle;
- Atribuição de variáveis;
- Procedimentos e processos;
- Tarefas e funções;
- Interfaces;
- Simulação e síntese lógica;
- Aplicações e exemplos;
- Laboratório - Projeto do flux-capacitor (vai e vem de LEDs);
- Laboratório - Projeto de máximo divisor comum

Sistemas Digitais Complexos com SystemVerilog HDL

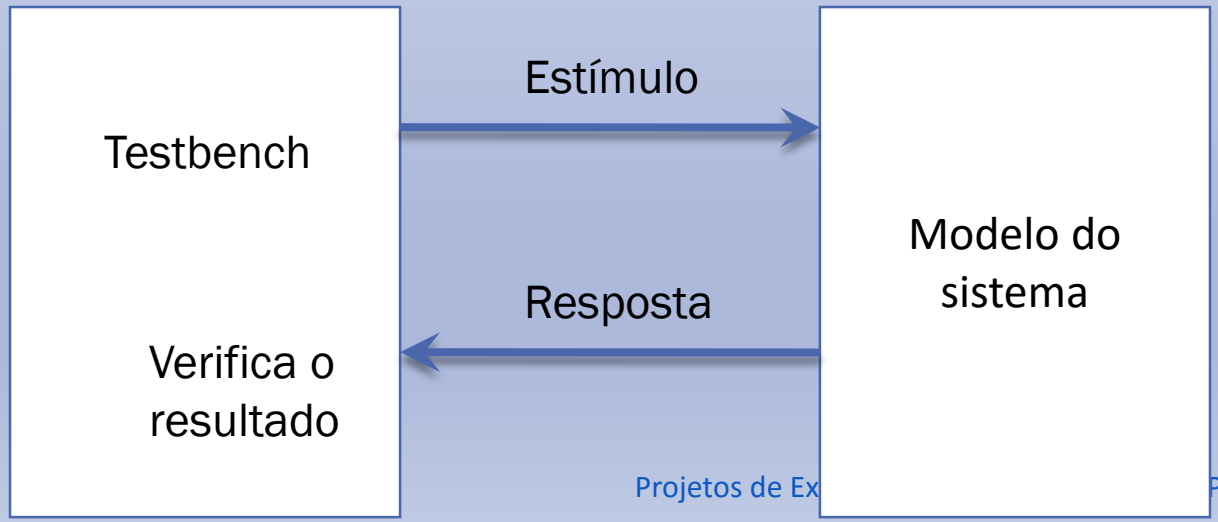
Nível COMPORTAMENTAL e RTL

Projetos de Excelência em Microeletrônica – PEM
Centro de Engenharia Elétrica e Informática
Universidade Federal de Campina Grande

Modelagem COMPORTAMENTAL

Usando Verilog para Simulação

- O testbench gera estímulos e verifica a resposta
- Acoplado ao modelo do sistema
- O par executa simultaneamente



Simulação de Eventos Discretos

- Idéia básica: somente trabalhe quando algo muda
- Centrado sobre uma fila de eventos que contêm eventos marcados com o tempo de simulação em que devem ser executados
- Paradigma básico da simulação
  - Executa todo evento para o tempo simulado atual
  - Fazendo isto muda o estado do sistema e pode escalonar eventos no futuro
  - Quando não há eventos restantes no tempo atual, avança o tempo simulado para o tempo de evento mais cedo da fila

Modelagem Comportamental

- Maneira muito mais fácil de escrever testbenches
- Também pode ser usado para fazer modelos abstratos dos circuitos
  - Mais fácil de escrever
  - Simula mais rápido
- Mais flexível
- Provê sequenciamento
- Verilog obteve sucesso em parte porque permitiu que o modelo e o testbench fossem descritos conjuntamente
- Aula posterior específica sobre simulação com verilog e verificação funcional

Dois componentes principais de Verilog: Comportamental

- Processos concorrentes, disparados por eventos (comportamental)
- Blocos initial e always
- Código imperativo que pode realizar tarefas padrão de manipulação de dados (atribuição, if-then, case)
- Processos ou estão executando (no caso eles podem produzir atrasos) ou estão esperando por um evento de disparo

Blocos Initial e Always

```
initial
   begin
      // sentenças
      // imperativas
end
```

- Roda quando a simulação começa
- Termina quando o controle atinge o end
- Bom para prover estímulos

```
always
   begin
      // sentenças
      // imperativas
end
```

- Roda quando a simulação começa
- Restarta quando o controle atinge o end
- Bom para modelagem ou para especificar hardware

Always e Initial

✖ Executam até encontrar um atraso

```
initial begin
  #10 a = 1; b = 0;
  #10 a = 0; b = 1;
end
```

✖ Ou uma espera por um evento

```
always @(posedge clk) q = d;

always begin
  wait(i);
  a = 0;
  wait(~i);
  a = 1;
end
```

Operadores de atribuição

- Atribuição bloqueante: =
- Atribuição não-bloqueante: <=

```
module assignment_test();
  reg [4:0] a,b,c,e;
  reg [7:0] d;
  initial
        begin
              a<=4'd2;
              $display ("A is %d", a);
              $display ("B is %d", b);
              $display ("D is %d", d);
              $display ("\n\n");
              b<=4'd3;
              $display ("A is %d", a);
              $display ("B is %d", b);
              $display ("D is %d", d);
              $display ("\n\n");
              d<=(a+b);
              $display ("A is %d", a);
              $display ("B is %d", b);
              $display ("D is %d", d);
              $display ("\n\n");
        end
endmodule
```

Resultado

Saída do compilador (atribuição bloqueante)	Saída do compilador (atribuição não-bloqueante)
A is 2 B is x D is x	A is x B is x D is x
A is 2 B is 3 D is x	A is x B is x D is x
A is 2 B is 3 D is 5	A is x B is x D is x

- Atribuição bloqueante: execução seqüencial
- Atribuição não-bloqueante: "simultânea" apenas no final

Como usar

- Basicamente:
  - Use não-bloqueante em lógica sequencial
  - Use bloqueante em lógica combinacional

Funções e Tarefas

Projetos de Excelência em Microeletrônica – PEM
Centro de Engenharia Elétrica e Informática
Universidade Federal de Campina Grande

Função

- pode ser declarada dentro de um module.
- pode ser usado dentro de um always_comb ou always_ff.
- Exemplo de declaração:

```
function logic [3:0] add (logic [3:0] a,b);
  if (a > 0) add = a + b;
  else add = 0;
endfunction
```

- Exemplo

```
always_comb
  if(p1==p2) s <= 15;
  else       s <= add(p1,p2);
```

Tarefa

- pode ser declarada dentro de um module.
- Pode incluir tempo: #atraso, posedge, wait...
- Pode ter várias entradas e saídas
- Variáveis locais (declarada no interior) e globais
- Pode chamar outras tarefas e funções
- Pode ser usado para modelar lógica combinacional e sequencial, mas geralmente apenas para simulação
- Tem que ser chamada como uma sentença

Tarefa

- Exemplo usando variável local

```
module simple_task();
task convert;
input [7:0] temp_in;
output [7:0] temp_out;
begin
 temp_out = (9/5) *( temp_in + 32)
end
endtask
endmodule
```

Tarefa

- Exemplo usando variável global

```
module task_global();
logic [7:0] temp_out;
logic [7:0] temp_in;
task convert;
begin
   temp_out = (9/5) *( temp_in + 32);
end
endtask
endmodule
```

Tarefa

- Usando uma tarefa

```
module task_calling (temp_a, temp_b, temp_c, temp_d);
input logic [7:0] temp_a, temp_c;
output logic [7:0] temp_b, temp_d;
logic [7:0] temp_b, temp_d;
`include "mytask.v"
always @ (temp_a)
begin
  convert (temp_a, temp_b);
end
always @ (temp_c)
begin
 convert (temp_c, temp_d);
end
endmodule
```

Parâmetros

Projetos de Excelência em Microeletrônica – PEM
Centro de Engenharia Elétrica e Informática
Universidade Federal de Campina Grande

Parametrização

- Evitar números mágicos

```
module sum #(parameter
param1=value1,param2=value2)
    (input ...
```

- Exemplo:

```
module sum #(parameter WIDTH=8)
            (input logic [WIDTH-1:0] A,B,
             output logic [WIDTH-1:0] Y);
   always_comb Y <= A+B;
endmodule
module top;
  ...
sum #(16) (.*);
```

Tipos de dados - signed

- Uma vetor de bits definido com logic representa por default um número sem sinal.
- Representação com sinal em complemento de 2:

```
logic signed [ msb : lsb ] nome;
```

- Exemplo

```
logic signed [7:0] saldo; // de -128 a 127
```

Tipos de dados compostos

- Definição de um novo tipo de dado

```
typedef struct {
    logic ...;
    logic ...; } nome_do_tipo;
```

- Exemplo de declaração do novo tipo

```
typedef struct {
  logic a;
  logic [3:0] b; } abba_t;
```

- Exemplo de declaração de barramento usando o novo tipo

```
abba_t mamamia;
```

Novos tipos de dados: inteiros

Tipos inteiros de dois estados: 0 e 1

TIPO	Descrição	Exemplo
bit	Tamanho variável	bit [3:0] um_nibble;
byte	8 bits, com sinal	byte a, b;
shortint	16 bits, com sinal	shortint c, d;
int	32 bits, com sinal	int i,j;
longint	64 bits, com sinal	longint lword;

Inteiros: continuação

Tipos inteiros de quatro estados: 0, 1, X e Z

TYPE	Descrição	Exemplo
reg	Tamanho variável	reg [7:0] a_byte;
logic	Idêntico ao reg	logic [7:0] a_byte;
integer	32 bits, com sinal	integer i, j, k;

Tipos não inteiros

TIPO	Descrição	Exemplo
time	64-bit sem sinal	time now;
shortreal	Como o float em C	shortreal f;
real	Como double em C	double g;
realtime	Idêntico a real	realtime now;

Arrays

- Podem ser *packed* ou *unpacked*
- Exemplo: `reg [3:0][7:0] register [0:9];`
- Dimensões packed (compactadas): [3:0] e [7:0]
- Dimensão unpacked (solta): [0:9]
- Pode-se ler e escrever e testar por igualdade a matriz inteira, fatias de elementos, elementos da matriz
- Há também matrizes dinâmicas ...

Packed	Unpacked
• Contínuo na memória •Pode ser copiado para outro objeto packed •Pode ser repartido •Restrito aos tipos “bit” (bit, logic, ...)	•Na memória como o simulador quiser •Pode ser copiado para outro array do mesmo tipo

Typedef

- Nomes novos para os tipos
- Muito interessante para arrays e structs
- Exemplos:

Com typedef	É o mesmo que

`typedef reg [7:0] octeto;`
 	
`octeto b;`
	`reg [7:0] b;`

`typedef octeto [3:0] quadOcteto;`
 	
`quadOcteto qBytes [1:10];`
	`reg [3:0][7:0] qBytes [1:10];`

Enumeração: enum

- Tipo de dados onde os valores são nomes
- Útil para valores de estados, opcodes, etc
- `enum { circle, ellipse, freeform } c;`
- Usado em conjunto com typedef
- ```
  typedef enum { circle, ellipse, freeform }
  ClosedCurve;
  ClosedCurve c;
  ```
  ```

enum

- Valores default são inteiros, começando do 0
- É fortemente tipado–não se mistura facilmente
- Só com cast (molde)
- ```
  c = 2;                    // ERRO
  c = ClosedCurve'(2); // Moldagem ok
  ```
- Mas quando usado como valor, pode ser comparado com inteiro
  ```

Struct e Union

- Similares a C
- Structs podem ser packed e unpacked
- Struct reserva espaço para todos os itens
- Union usa o mesmo espaço para todos os itens

```
struct {
  int x, y;
} p;
```

- Para selecionar usa a sintaxe .nome

```
p.x = 1;
```

Mais structs e unions

- Expressões com a struct inteira usando {}

```
p = {1,2};
```

- Typedef usado para criar e usar tipos struct

```
typedef struct packed {
 int x, y;
} Point;
Point p;
```

- Exemplo com union: registrador guarda tipo inteiro ou em ponto-flutuante

RTL - Register Transfer Level

- Novos operadores
- Novas formas de laço
- Labels (marcadores)
- Regras de Atribuição
- Conexões .nome e .*
- Formações para síntese
- Unique e Priority

Novos operadores

- Vários de C:
- Incremento e decremento `++`, `--`
- E atribuições `+=`, `-=`, `*=` , etc
- Ampla igualdade e desigualdade: `===` e `!==`
  - X e Z valem “don’t care” = “não importa”
- Igual ao teste de `casex (nunca usar casex!)`

Novas formas de laço

- Também de C:
- do ... while
- break e continue
- foreach para variáveis tipo array
- for melhorado, por exemplo:

```
for (int i = 15, logic j = 0 ; i > 0 ; i--, j = ~j)
  ...
```

Labels (marcadores)

- Em verilog, pode-se colocar labels em sentenças `begin` e `fork`

```
begin : a_label
```

- Em SystemVerilog, pode-se repetir no final `end` e `join`

```
end : a_label
```

- Também em módulos, funções e tarefas

```
module MyModule ...
...
endmodule : MyModule
```

- Também pode dar nomes a laços, permitindo desabilitar (`disable`) e habilitar (`enable`)

```
loop : for (int i=0; ...
```

Regras de Atribuição

- Variáveis (logic) em SystemVerilog podem ser atribuídas usando atribuições procedurais, atribuições contínuas e ser conectadas a saídas de módulos
- wire ainda tem uso quando se quer ligar mais de uma saída – onde se usa as regras de resolução -- para tri-state, wired-and , wired-or, chaves, transistores, etc

Unique e Priority

- Em verilog, usava-se `pragma` para indicar ao sintetizador o que se queria em um `case`
- SystemVerilog inseriu `priority` e `unique`
- Servem para `if` também
- `unique` = completude e unicidade
  - Exatamente um ramo da condição é usado por vez
  - Implica em lógica combinacional, podendo ficar sem `default` e `else`
- `priority` = ao menos um ramo é usado
  - Deixa o sintetizador usar lógica de prioridade que quiser

Interfaces

Separação entre o que e como

Interfaces

- Encapsulam a comunicação entre blocos
- São estruturas hierárquicas, podendo conter outras interfaces
- Vantagens
- Encapsulam conectividade
  - um só item substitui um grupo de nomes
- Encapsulam funcionalidade
  - Isola dos módulos conectados pela interface
  - Nível de abstração e granularidade do protocolo de comunicação refinado independente dos módulos

Vantagens de Interfaces - cont

- Podem conter parâmetros, constantes, variáveis, funções e tarefas, processos (always) e atribuições contínuas
- Ajudam na construção de aplicações como gravação e relatório de cobertura funcional, verificação de protocolo e assertivas
- Usado para acesso sem portas – instanciando a interface como um objeto estático no módulo
  - Permite acesso a informações chamadas em lugares diferentes para compartilhar informações

Vantagens de Interfaces - cont

- Flexibilidade, uma interface pode ser parametrizada
  - Também há a interface genérica, que pode ser usada na definição do módulo e só ser definida na instanciação
- A interface mais simples é um conjunto de fios, similar a uma struct

Definição da Interface

```
// Definição da Interface
interface Bus;
  logic [7:0] Addr, Data;
  logic RWn;
endinterface
```

Módulo definido com interface

```
module RAM (Bus MemBus);
  logic [7:0] mem [256];

  always @*
    if (MemBus.RWn)
      MemBus.Data = mem[MemBus.Addr];
    else
      mem[MemBus.Addr] = MemBus.Data;
endmodule
```

Uso

```
module TestRAM; // Usando
// Instanciando a interface
  Bus TheBus();
  logic [7:0] mem [0:7];
// Conecta
  RAM TheRAM (.MemBus(TheBus));

  initial
  begin
    // Aciona e monitora o Bus
    TheBus.RWn = 0;
    TheBus.Addr = 0;
    for (int I=0; I < 7; I++)
      TheBus.Addr = TheBus.Addr + 1;
    TheBus.RWn = 1;
    TheBus.Data  = mem[0];
  end
endmodule
```

Exemplo de interface simples

```
// Definição da Interface
interface Bus;
  logic [7:0] Addr, Data;
  logic RWn;
endinterface
```

```
module RAM (Bus MemBus);
  logic [7:0] mem [0:255];

  always @*
    if (MemBus.RWn)
      MemBus.Data = mem[MemBus.Addr];
    else
      mem[MemBus.Addr] = MemBus.Data;
endmodule
```

```
// Usando a interface
module TestRAM;
// Instanciando a interface
 Bus TheBus();
 logic[7:0] mem[0:7];
  // Conecta
  RAM TheRAM (.MemBus(TheBus));

  initial
  begin
    // Aciona e monitora o Bus
    TheBus.RWn = 0;
    TheBus.Addr = 0;
    for (int I=0; I<7; I++)
      TheBus.Addr = TheBus.Addr + 1;
    TheBus.RWn = 1;
    TheBus.Data  = mem[0];
  end
endmodule
```

Portas nas interfaces

- Uma interface pode ter portas de entrada, saída e entrada/saída (inout)
- Para conexão extra, conectado externamente
- Forma habitual
- Serve para sinais compartilhados entre interfaces

Igual a anterior, mas com clock

```
interface ClockedBus (input Clk);
    logic[7:0] Addr, Data;
    logic RWn;
endinterface
```

Igual a anterior, mas com clock

```
module RAM (ClockedBus Bus);
  always @(posedge Bus.Clk)
    if (Bus.RWn)
      Bus.Data = mem[Bus.Addr];
    else
      mem[Bus.Addr] = Bus.Data;
endmodule
```

Usando, observe a porta

```
// Usando a interface
module Top;
  reg Clock;

  // Instancia a interface com uma conexão
 de entrada com nome
  ClockedBus TheBus (.Clk(Clock));
  RAM TheRAM (.Bus(TheBus));
  ...
endmodule
```

Interfaces Parametrizadas

```
interface Channel #(parameter N = 0)
    (input bit Clock, bit Ack, bit Sig);
  bit Buff[N-1:0];
  initial
    for (int i = 0; i < N; i++)
      Buff[i] = 0;
  always @ (posedge Clock)
   if(Ack = 1)
     Sig = Buff[N-1];
   else
     Sig = 0;
endinterface
```

Usando Interface Parametrizada

```
// Usando a interface
module Top;
  bit Clock, Ack, Sig;
// Instancia a interface. O parâmetro
// N é posto em 7 usando conexão com
// nome e as portas conectadas usando
// conexão implícita
  Channel #(.N(7)) TheCh (.*);
  TX TheTx (.Ch(TheCh));
  ...
endmodule
```

ModPorts nas Interfaces

- `modport` fornece informação de direção
- As direções são como visto pelos módulos
- No exemplo seguinte, as direções são as vistas pelo módulo na qual o modport é conectado – no caso, a RAM
- Addr – entrada
- Data – inout
- Pode-se ter mais de um `modport`

Exemplo de ModPort (modport)

```
interface MSBus (input Clk);
  logic [7:0] Addr, Data;
  logic RWn;
  modport Slave (input Addr, inout Data);
endinterface
```

Observe que foi usado pela RAM

```
module TestRAM;
   logic Clk;
   MSBus TheBus(.Clk(Clk));
   RAM TheRAM (.MemBus(TheBus.Slave));
   ...
endmodule
```

Observe que foi usado pela RAM

```
module TestRAM;
  logic Clk;
  MSBus TheBus(.Clk(Clk));
  RAM TheRAM (.MemBus(TheBus.Slave));
  ...
endmodule

module RAM (MSBus.Slave MemBus);
  // MemBus.Addr é uma entrada da RAM
endmodule
```

Tarefas e Funções em Interfaces

- Quando mais apropriado, tarefas e funções podem ser definidas em interfaces
- Usado para modelar funcionalidades dos barramentos
- Próximo exemplo mostra duas tarefas

Tarefas e Funções em Interfaces

```
interface MSBus (input Clk);
  logic [7:0] Addr, Data;
  logic RWn;

  task MasterWrite (input logic [7:0] waddr,
                    input logic [7:0] wdata);
    Addr = waddr;
    Data = wdata;
    RWn = 0;
    #10ns RWn = 1;
    Data = 'z;
  endtask

  task MasterRead (input  logic [7:0] raddr,
                   output logic [7:0] rdata);
    Addr = raddr;
    RWn = 1;
    #10ns rdata = Data;
  endtask
endinterface
```

Tarefas e Funções em Interfaces

```
module TestRAM;
  logic Clk;
  logic [7:0] data;
  MSBus TheBus(.Clk(Clk));
  RAM TheRAM (.MemBus(TheBus));
  initial
  begin
    // Escreve na RAM
    for (int i = 0; i<256; i++)
      TheBus.MasterWrite(i[7:0],i[7:0]);

    // Lê da RAM
    for (int i = 0; i<256; i++)
    begin
      TheBus.MasterRead(i[7:0],data);
      ReadCheck : assert (data === i[7:0])
        else $error(“erro de leitura na memória");
    end
  end
endmodule
```

Circuitos sequenciais e controle

PRÓXIMA AULA

Contato

Marcos Morais
Professor DEE
morais@dee.ufcg.edu.br

Linguagem de Descrição de Hardware

Prof. Marcos Morais
DEE/UFCG

Projetos de Excelência em Microeletrônica – PEM
Centro de Engenharia Elétrica e Informática
Universidade Federal de Campina Grande