



FUNDAÇÃO EDSON QUEIROZ
UNIVERSIDADE DE FORTALEZA
ENSINANDO E APRENDENDO

T566 –SISTEMAS DIGITAIS AVANÇADOS

Aula 10 - Verilog

Prof. Danilo Reis



O que é?

- Linguagem de descrição de Hardware para modelar circuitos;
- Modela comportamento: O componente é descrito como entrada e saídas;
- Modela a estrutura: O componente é descrito como um low-level de conexões de componentes primitivos;



Histórico

- **Verilog** foi criada por Phil Moorby and Prabhu Goel na empresa Gateway Design Automation em 1984;
- Em 1990 a Cadence comprou a Gateway Design Automation;
- Em 1995 a Cadence decidiu tornar publica a linguagem para torna-se um padrão submetida para Open Verilog e IEEE.
- Padronizada em 1995 pela IEEE (Verilog 95);
- Revisão da IEEE em 2001;
- Revisão da IEEE em 2005 (Verilog 2005)
- Em 2009 submetida o superset da Verilog 2005 chamada System Verilog;



Objetivo da Linguagem

```
always @(a, b, c, d, sel)
```

```
  case (sel)
```

```
    2'b00: mux_out = a;
```

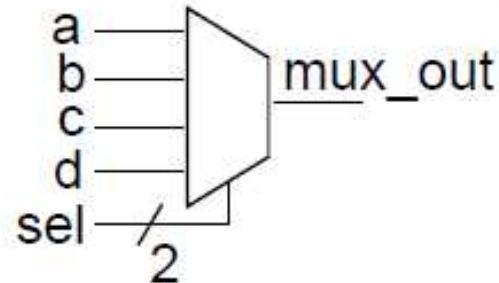
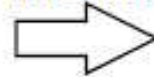
```
    2'b01: mux_out = b;
```

```
    2'b10: mux_out = c;
```

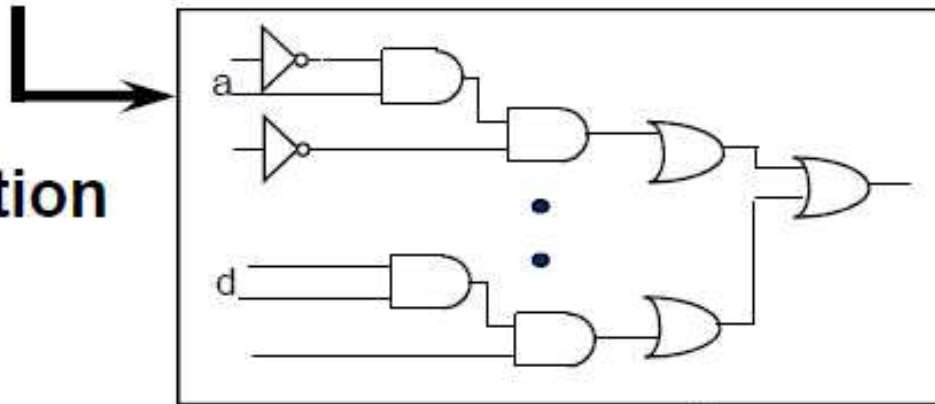
```
    2'b11: mux_out = d;
```

```
  endcase
```

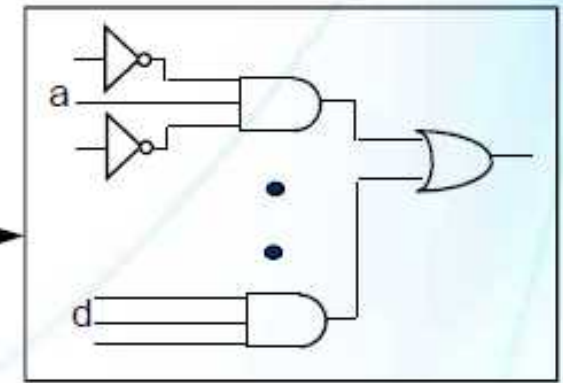
inferred



Translation

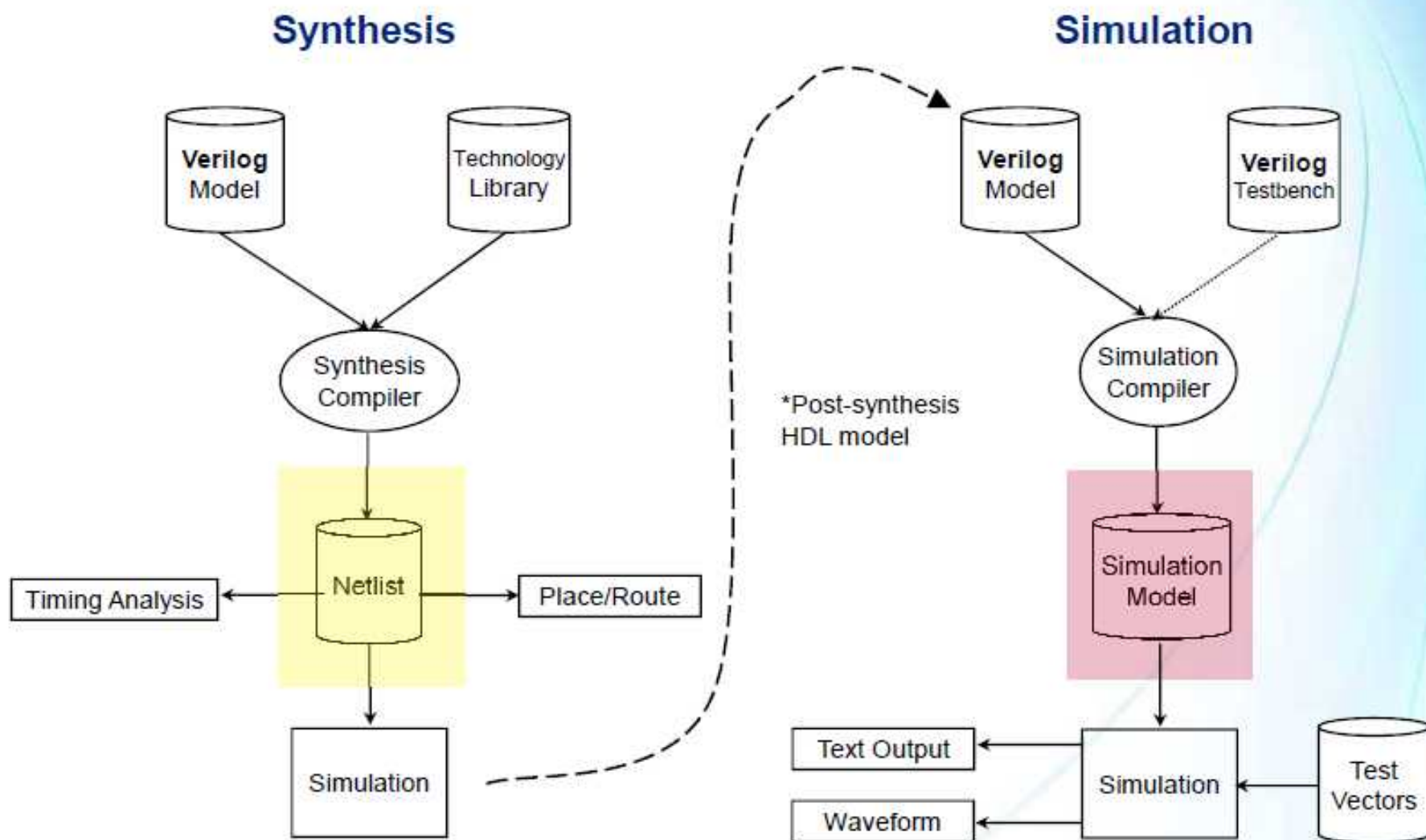


Optimization





Typical RTL Synthesis & RTL Simulation Flows





Estrutura Básica

```
module module_name (port_list);
```

```
port declarations
```

```
data type declarations
```

```
circuit functionality
```

```
timing specifications
```

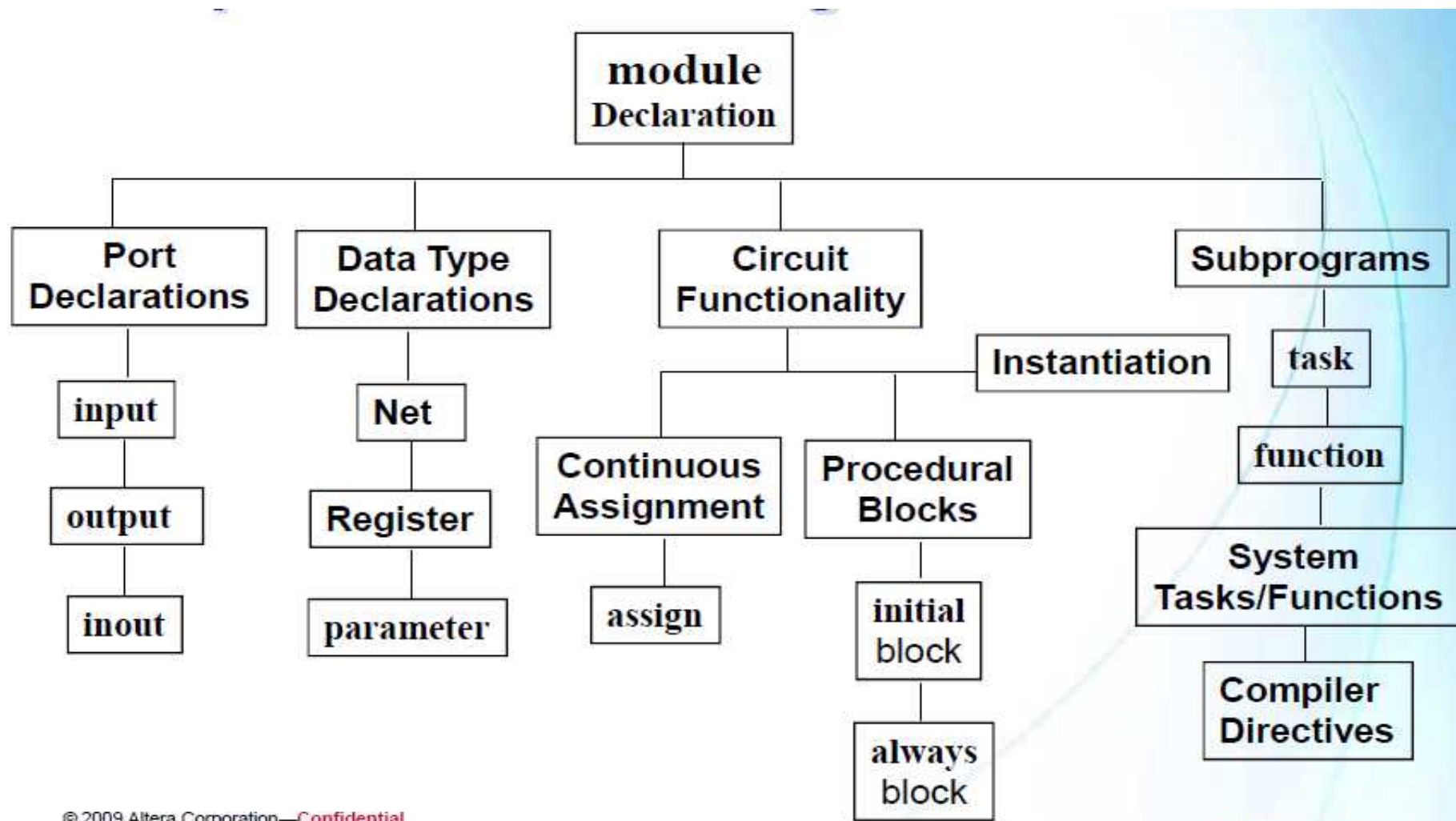
```
endmodule
```

O verilog possui as seguintes características:

- Cada módulo começa com a palavra **module** e termina com a palavra **endmodule**;
- É case-sensitive;
- Todas as palavras reservadas são minúsculas;
- Ponto e vírgula(';') é o terminador de instruções;
- Comentários são como em C, i.e., // para comentário de linha e /* */ para comentários mutli-linha;
- Especificação de timing é utilizada para simulação;
- Na linguagem nem todos os comandos são sintetizáveis;

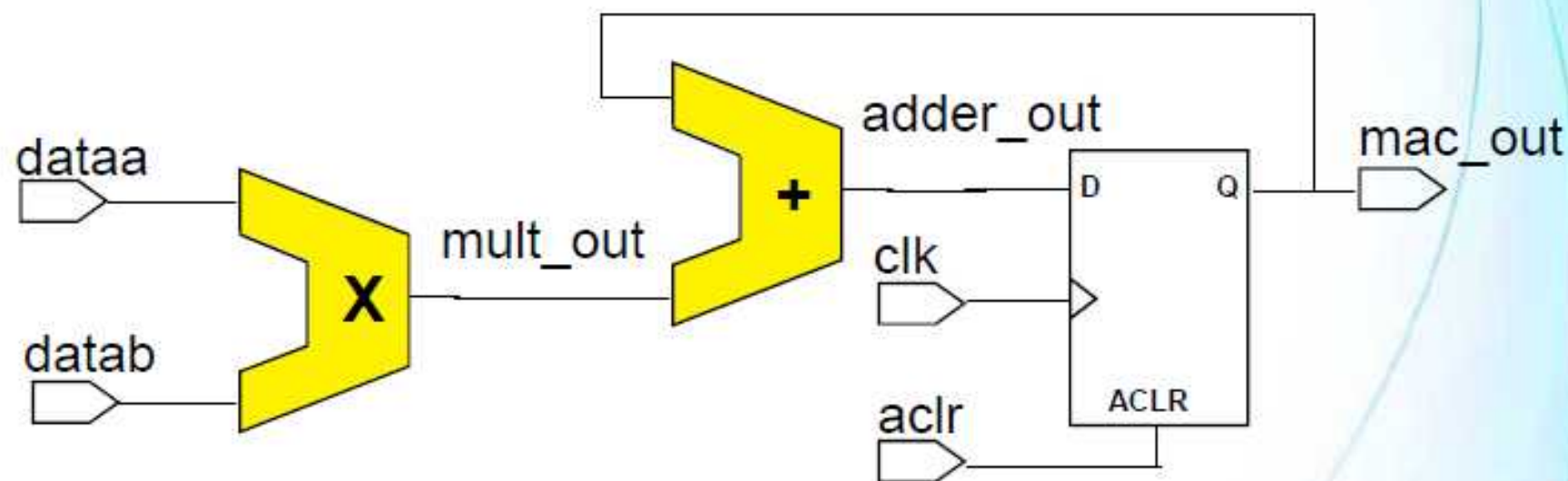


Componentes de um módulo Verilog





Exemplo circuito Multiplicador Acumulador





Verilog Model: Multiplier-Accumulator (MAC)

```
`timescale 1 ns/ 10 ps

module mult_acc (
    input [7:0] dataa, datab,
    input clk, aclr,
    output reg [15:0] mac_out
);

    wire [15:0] mult_out, adder_out;

    parameter mult_size = 8;

    assign adder_out = mult_out + mac_out;

    always @ (posedge clk, posedge aclr) begin
        if (aclr)
            mac_out <= 16'h0000;
        else
            mac_out <= adder_out;
        end

    mult_a #(.width_in(mult_size))
        u1 (.in_a(dataa), .in_b(datab), .mult_out(mult_out));

endmodule
```



Declaração do módulo

- Começa com palavra **module**
- Prover um bloco Verilog (module)
- Inclui o port list;

```
module mult_acc (mac_out, dataa, datab, clk, aclr);
```



Declaração dos Port

- Define o nome, tamanho, tipos e direção de todas as portas;
- Formato

```
<port_type> port_name;
```

- Exemplo

```
input [7:0] dataa, datab;  
input clk, aclr;  
output [15:0] mac_out;
```

- Port types
 - input \Rightarrow input port
 - output \Rightarrow output port
 - inout \Rightarrow bidirectional port



Declaração Module/Port (Verilog 2001)

- Na versão Verilog 2001 a declaração pode ser combinada
- Exemplo

```
module mult_acc (  
    input [7:0] dataa, datab,  
    input clk, aclr,  
    output [15:0] mac_out  
);
```



Declaração Module/Port no MAC

```
`timescale 1 ns/ 10 ps

module mult_acc (
    input [7:0] dataa, datab,
    input clk, aclr,
    output reg [15:0] mac_out
);

    wire [15:0] mult_out, adder_out;

    parameter mult_size = 8;

    assign adder_out = mult_out + mac_out;

    always @ (posedge clk, posedge aclr) begin
        if (aclr)
            mac_out <= 16'h0000;
        else
            mac_out <= adder_out;
        end

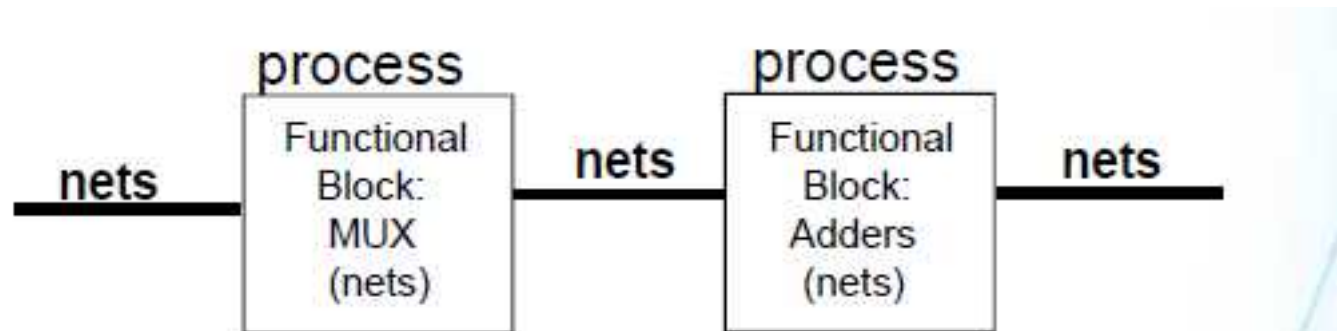
    mult_a #(.width_in(mult_size))
        u1 (.in_a(dataa), .in_b(datab), .mult_out(mult_out));

endmodule
```

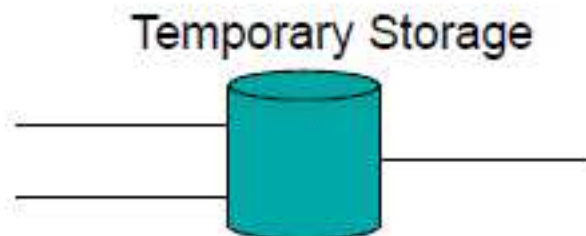


Data Types

- Net data type - representa a interconexão física das estruturas



- Variable data type - representa elementos para armazenar dados temporários





Net Data Types & Net Arrays

- **wire** \Rightarrow represents a node
- **tri** \Rightarrow represents a tri-state node
- Bus Declarations:
 - **<data_type>** [MSB : LSB] <signal name> ;
 - **<data_type>** [LSB : MSB] <signal name> ;
- Exemplo:
 - **wire** <signal name> ;
 - **wire** [15:0] mult_out, adder_out;



Net Data Types

Net Data Types	Functionality	Synthesis Support?
wire	Used for interconnect	Y
tri		Y
supply0	Represents constant value (i.e. power supply)	Y
supply1		Y
wand	Represents wired logic	Y
triand		Y
wor		Y
trior		Y
tri0	Tri-state node with pull-up/pull-down	Y
tri1		Y
triereg	Stores last value when not driven	N

Note: There is no functional difference between wire & tri; wand & triand; wor & trior



Variable Data Types & Variable Arrays

- **reg**(1) - unsigned variable de qualquer tamanho em bit
 - Use **reg signed** for a signed implementation(2)
- **integer** - signed variable (usually 32 bits)
- Bus Declarations:
 - **<data_type>** [MSB : LSB] **<signal name>** ;
 - **<data_type>** [LSB : MSB] **<signal name>** ;
- Examples:
 - **reg** **<signal name>** ;
 - **reg** [7 : 0] **out** ;

Notas:

- 1) Type reg não se refere a registrador físico;
- 2) A representação signed também é suportada por net data types



Variable Data Types

Net Data Types	Functionality	Synthesis Support?
reg	Unsigned variable (by default) Use <code>reg signed</code> for signed representation	Y
integer	Signed variable (usually 32 bits)	Y
time	Unsigned integers (usually 64 bits) used for storing and manipulating simulation time	N
real	Double precision floating point variable	N
realtime	Double-precision floating point variable used with time	N



Memory

- Multi-dimensional variable array
- Não pode ser um **net type**
- Exemplo:

```
reg [31:0] mem[0:1023]; // 1Kx32
reg [31:0] instr;

instr = mem[2];
mem [1000][5:0] = instr[5:0]; // Unsupported by synthesis
```

- Não é permitido escrever múltiplos elementos em um assignment

```
mem = 32'd0; Illegal!!!
```



Verilog 2001 Module/Port/Parameter

- Declaração combinada de Module, port e parameter
 - Illegal para parameters locais

```
module mult_acc
  #(parameter size = 8)
  (
    input [size-1:0] dataa, datab,
    input clk, clr,
    output [(size*2)-1:0] mac_out
  );
```



Data Type

- Todos sinal (incluindo os ports) deve ter um assigned data type;
- Data types para sinais devem ser explicitamente declarados na declaração do módulo;
- Ports são wire (net) data types por default se não for explicitamente declaradas



MAC (Data Type Declarations)

```
`timescale 1 ns/ 10 ps

module mult_acc (
    input [7:0] dataa, datab,
    input clk, aclr,
    output reg [15:0] mac_out
);

    wire [15:0] mult_out, adder_out;

    parameter mult_size = 8;

    assign adder_out = mult_out + mac_out;

    always @ (posedge clk, posedge aclr) begin
        if (aclr)
            mac_out <= 16'h0000;
        else
            mac_out <= adder_out;
        end

    multa #(.width_in(mult_size))
        u1 (.in_a(dataa), .in_b(datab), .mult_out(mult_out));

endmodule
```




Atribuindo valores - Números

- São sized ou unsized: `<size>'<base format><number>`
 - Exemplo sized: `3'b010` = Número binário de 3-bit
 - O prefixo (3) indica o tamanho do número
 - Exemplo unsized: `123` = número decimal de 32-bit por default.
 - Defaults
 - Não especificado `<base format>` defaults to decimal
 - Não especificado `<size>` defaults de número com 32-bit.
- Base Formats
 - Decimal ('d or 'D) `16'd255` = número decimal de 16-bit
 - Hexadecimal ('h or 'H) `8'h9a` = número hexadecimal de 8-bit.
 - Binary ('b or 'B) `'b1010` = número binário de 32-bit.
 - Octal ('o or 'O) `'o21` = número octal de 32-bit
 - Signed ('s' or 'S') `16'shFA` = número hex signed de 16-bit.



Números

- **Números negativos** - especificados com sinal '-' antes de <size>
 - Legal: **-8'd3** = Número negativo de 8-bit(armazenado como complemento de 2)
 - Illegal: **4'd-2** = **ERROR!!**
- **Caracteres especiais com números**
 - '_' (underscore): usado para facilitar a leitura
 - Exemplo: **32'h21_65_bc_fe** = 32-bit número hexadecimal
 - 'x' or 'X' (valor desconhecido)
 - Exemplo: **12'h12x** = 12-bit hexadecimal number; LSBs desconhecido
 - 'z' or 'Z' (valor de high impedance)
 - Exemplo: **1'bz** = número de 1-bit high impedance



Numeros Extensoes

- Se MSB é 0, x, or z, extensão de número preenche os MSBs com 0, x, or z, respectivamente
 - Exemplos
 - `3'b01` é igual a `3'b001`
 - `3'bx1` é igual a `3'bxx1`
 - `3'bz` é igual a `3'bzzz`
-
- Se MSB é 1, extensão de número preenche os MSBs com 0
 -
 - Exemplo
 - `3'b1` é igual a `3'b001`



Operadores

- Aritméticos
- Bitwise
- Redução
- Relacionais
- Igualdade
- Lógicos
- Deslocamentos
- Miscelâneos



Operadores Aritmeticos

Operator Symbol	Functionality	Examples ain = 5 ; bin = 10 ; cin = 2'b01 ; din = 2'b0z		
+	Add, Positive	$\text{bin} + \text{cin} \Rightarrow 11$	$+\text{bin} \Rightarrow 10$	$\text{ain} + \text{din} \Rightarrow \text{x}$
-	Subtract, Negate	$\text{bin} - \text{cin} \Rightarrow 9$	$-\text{bin} \Rightarrow -10$	$\text{ain} - \text{din} \Rightarrow \text{x}$
*	Multiply	$\text{ain} * \text{bin} \Rightarrow 50$		
/	Divide*	$\text{bin} / \text{ain} \Rightarrow 2$		
%	Modulus	$\text{bin} \% \text{ain} \Rightarrow 0$		
**	Exponent*	$\text{ain} ** 2 \Rightarrow 25$		



Operadores Bitwise

Operator Symbol	Functionality	Examples $ain = 3'b101$; $bin = 3'b110$; $cin = 3'b01x$	
\sim	Invert each bit	$\sim ain \Rightarrow 3'b'010$	$\sim cin \Rightarrow 3'b10x$
$\&$	AND each bit	$ain \& bin \Rightarrow 3'b100$	$bin \& cin \Rightarrow 3'b010$
$ $	OR each bit	$ain bin \Rightarrow 3'b111$	$bin cin \Rightarrow 3'b11x$
\wedge	XOR each bit	$ain \wedge bin \Rightarrow 3'b011$	$bin \wedge cin \Rightarrow 3'b10x$
$\wedge \sim$ or $\sim \wedge$	XNOR each bit	$ain \wedge \sim bin \Rightarrow 3'b100$	$bin \sim \wedge cin \Rightarrow 3'b01x$



Operadores Redução

Operator Symbol	Functionality	Examples $ain = 4'b1010$; $bin = 4'b10xz$; $cin = 4'b111z$		
$\&$	AND all bits	$\&ain \Rightarrow 1'b0$	$\&bin \Rightarrow 1'b0$	$\&cin \Rightarrow 1'bx$
$\sim\&$	NAND all bits	$\sim\&ain \Rightarrow 1'b1$	$\sim\&bin \Rightarrow 1'b1$	$\sim\&cin \Rightarrow 1'bx$
$ $	OR all bits	$ ain \Rightarrow 1'b1$	$ bin \Rightarrow 1'b1$	$ cin \Rightarrow 1'b1$
$\sim $	NOR all bits	$\sim ain \Rightarrow 1'b0$	$\sim bin \Rightarrow 1'b0$	$\sim cin \Rightarrow 1'b0$
\wedge	XOR all bits	$\wedge ain \Rightarrow 1'b0$	$\wedge bin \Rightarrow 1'bx$	$\wedge cin \Rightarrow 1'bx$
$\wedge\sim$ or $\sim\wedge$	XNOR all bits	$\sim\wedge ain \Rightarrow 1'b1$	$\sim\wedge bin \Rightarrow 1'bx$	$\sim\wedge cin \Rightarrow 1'bx$



Operadores Relacionais

Operator Symbol	Functionality	Examples	
		ain = 3'b101 ; bin = 3'b110 ; cin = 3'b01x	
>	Greater than	ain > bin \Rightarrow 1'b0	bin > cin \Rightarrow 1'bx
<	Less than	ain < bin \Rightarrow 1'b1	bin < cin \Rightarrow 1'bx
>=	Greater than or equal to	ain >= bin \Rightarrow 1'b0	bin >= cin \Rightarrow 1'bx
<=	Less than or equal to	ain <= bin \Rightarrow 1'b1	bin <= cin \Rightarrow 1'bx



Operadores Igualdade

Operator Symbol	Functionality	Examples <code>ain = 3'b101 ; bin = 3'b110 ; cin = 3'b01x</code>	
<code>==</code>	Equality	<code>ain == bin ⇒ 1'b0</code>	<code>cin == cin ⇒ 1'bx</code>
<code>!=</code>	Inequality	<code>ain != bin ⇒ 1'b1</code>	<code>cin != cin ⇒ 1'bx</code>
<code>===</code>	Case equality	<code>ain === bin ⇒ 1'b0</code>	<code>cin === cin ⇒ 1'b1</code>
<code>!==</code>	Case inequality	<code>ain <= bin ⇒ 1'b1</code>	<code>cin !== cin ⇒ 1'b0</code>



Operadores Logicos

Operator Symbol	Functionality	Examples <code>ain = 3'b101 ; bin = 3'b000 ; cin = 3'b01x</code>		
!	Expression not true	<code>!ain ⇒ 1'b0</code>	<code>!bin ⇒ 1'b1</code>	<code>!cin ⇒ 1'bx</code>
&&	AND of two expressions	<code>ain && bin ⇒ 1'b0</code>	<code>bin && cin ⇒ 1'bx</code>	
 	OR of two expressions	<code>ain bin ⇒ 1'b1</code>	<code>bin cin ⇒ 1'bx</code>	



Operadores Deslocamentos

Operator Symbol	Functionality	Examples $ain = 3'b101$; $bin = 3'b01x$	
\ll	Logical shift left	$ain \ll 2 \Rightarrow 3'b100$	$bin \ll 2 \Rightarrow 3'bx00$
\gg	Logical shift right	$ain \gg 2 \Rightarrow 3'b001$	$bin \gg 2 \Rightarrow 3'b000$
\lll	Arithmetic shift left	$ain \lll 2 \Rightarrow 3'b100$	$bin \lll \Rightarrow 3'bx00$
\ggg	Arithmetic shift right	$ain \ggg 2 \Rightarrow 3'b111$ (signed)	$bin \ggg 2 \Rightarrow 3'b000$ (signed)

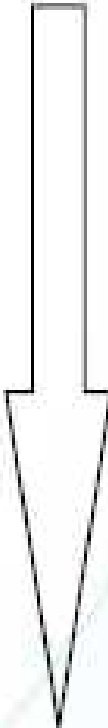


Operadores Miscelaneos

Operator Symbol	Functionality	Format & Examples
<code>?:</code>	Conditional test	<code>(condition) ? true_value : false_value</code> <code>sig_out = (sel == 2'b01) ? a : b</code>
<code>{}</code>	Concatenate	<code>ain = 3'b010 ; bin = 3'110</code> <code>{ain,bin} ⇒ 6'b010110</code>
<code>{ {} }</code>	Replicate	<code>{3 {3'b101}} ⇒ 9'b101101101</code>



Precedencia

Operator(s)	Priority
+ - ! ~ & ~& etc. (unary* operators)	 <p>High</p> <p>Low</p>
**	
* / %	
+ - (binary operators)	
<< >> <<< >>>	
< > <= >=	
== != === !==	
& (binary operator)	
^ ~^ ^~ (binary operators)	
(binary operator)	
&&	
?:	
{ } { { } }	



Continuous Assignments

O modelo de comportamento de lógica combinacional usando expressões e operadores o continuous assignment pode ser feito de duas maneiras:

- Continuous assignments declarado na net
 - **wire [15:0] adder_out = mult_out + out;**
- Utilizando instrução assign
 - **wire [15:0] adder_out;**
 - **assign adder_out = mult_out + out;**



Continuous Assignments características

1. O lado esquerdo do assignment (LHS) deve ser do tipo net data type;
2. Sempre ativo: Quando um dos operandos do lado direito muda automaticamente a expressão é calculada e atualizada no lado esquerdo;
3. RHS pode ser uma expressão contendo net data type, variable data type ou uma function call (ou combinação delas);
4. Valores de Delay podem ser atribuídos ao modelo de portas



Referências

- Curso oficial da Altera "Introduction to Verilog";