Arquitetura de Computadores

Noções Gerais de HDL (Verilog); Representações de dados; Operações sobre os dados.

UNIVERSIDADE FEDERAL DE SERGIPE

Professor
Calebe Conceição

Na aula de hoje, você irá:

Revisar (ver) os conteúdos de Fundamentos de Sistemas Digitais

Aprofundar-se no domínio de uma linguagem profissional de descrição de hardware

Ambientar-se com o uso de ferramentas de projeto de circuitos digitais

Ter a oportunidade de estudar uma linguagem nova

Aprofunde-se!

Consulte o manual da linguagem

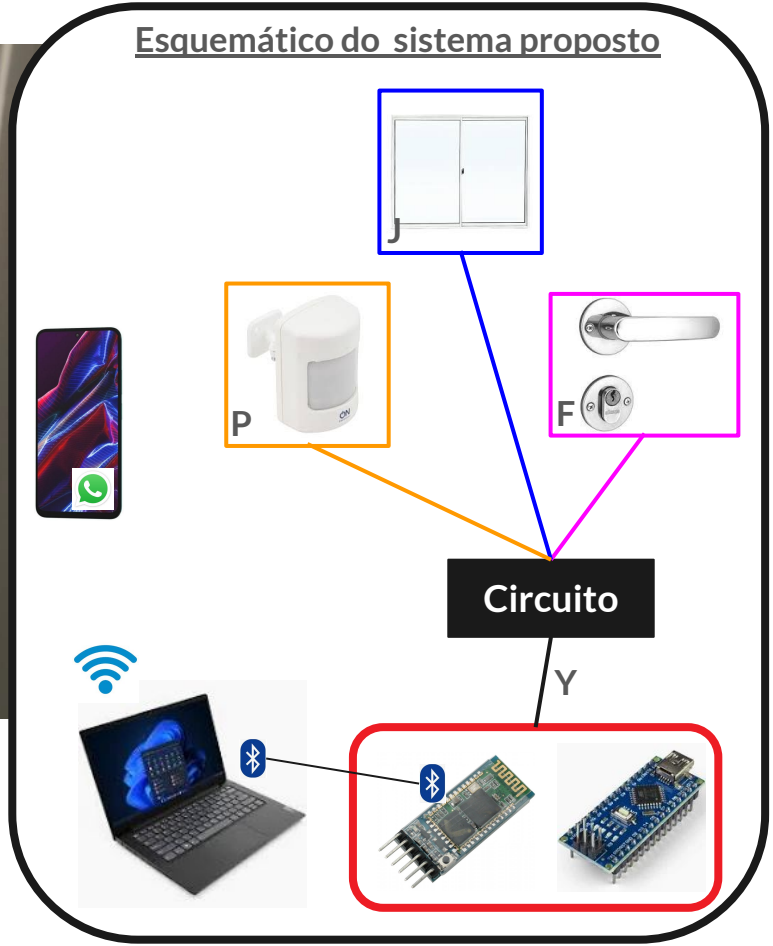
Disponível no Classroom

Acessível via portal de periódicos da CAPES.

Contextualização

Revisão: Análise e Projeto Digital (Compreensão)

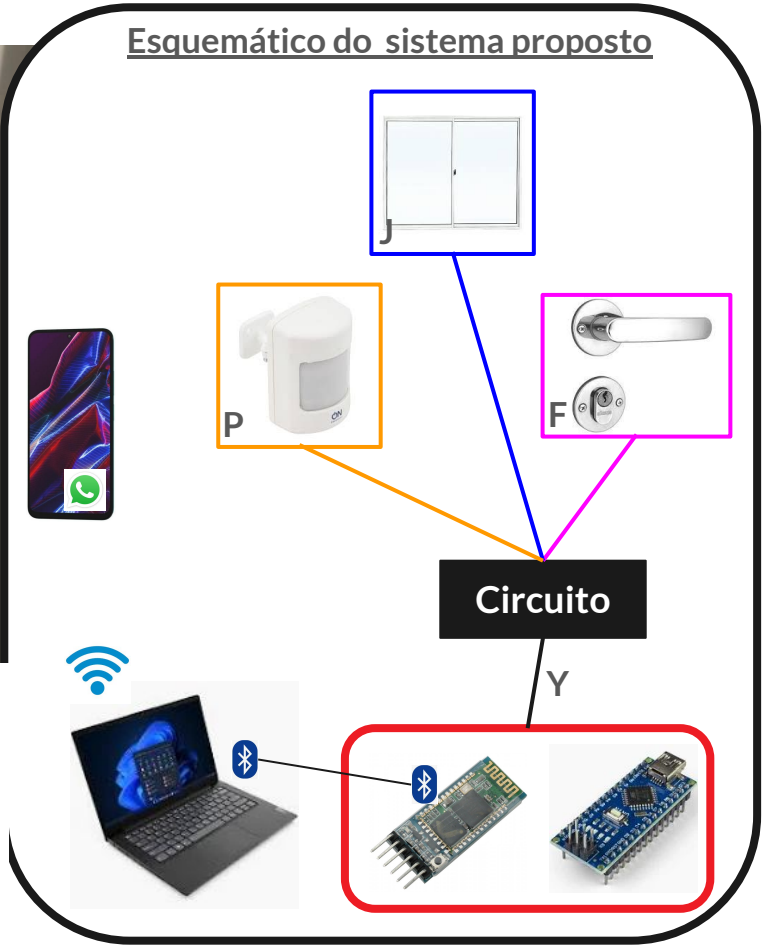
Revisão: Análise e Projeto Digital (Especificação)



UNIVERSIDADE FEDERAL DE SERGIPE

Revisão: Análise e Projeto Digital (Especificação)

- Quero ser avisado quando:
  - Algum movimento é detectado e está tudo fechado
  - Quando não há movimento e algo está aberto
  - Quando só a janela está aberta e há movimento

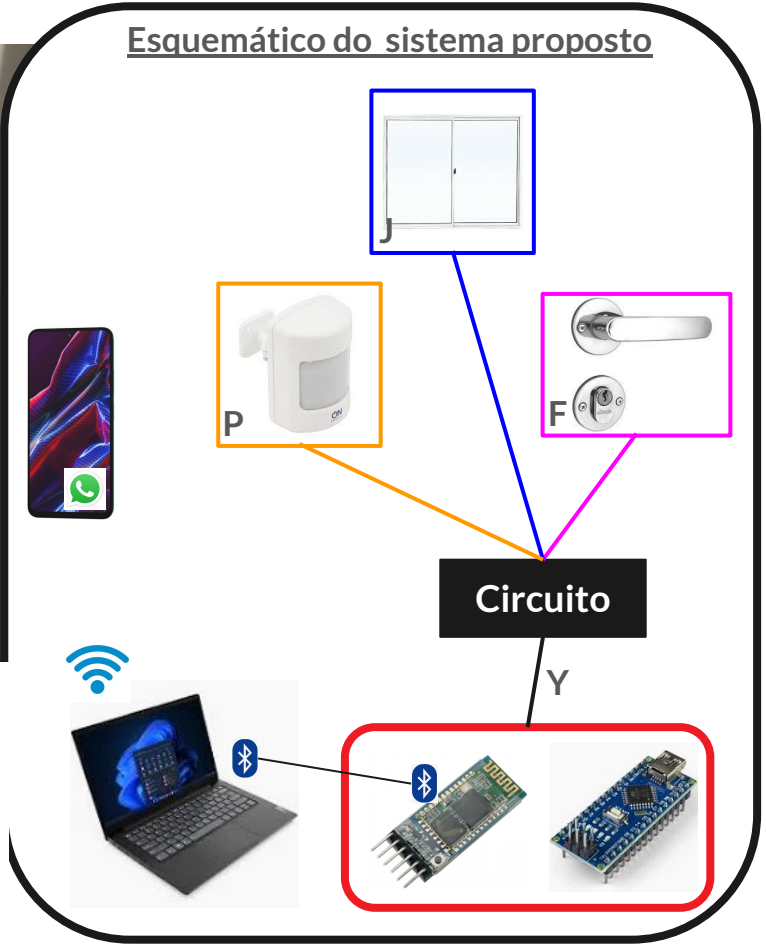


UNIVERSIDADE FEDERAL DE SERGIPE

Revisão: Análise e Projeto Digital (Detalhamento)

- Quero ser avisado quando:
  - Algum movimento é detectado e está tudo fechado
  - Quando não há movimento e algo está aberto
  - Quando só a janela está aberta e há movimento

Podemos descrever as mesmas regras de outra forma, como tabela-verdade

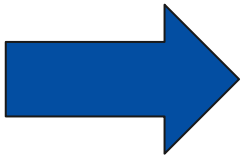


Revisão: Análise e Projeto Digital (Refino)

- Quero ser avisado quando:
  - Algum movimento é detectado e está tudo fechado
  - Quando não há movimento e algo está aberto
  - Quando só a janela está aberta e há movimento

Podemos descrever as mesmas regras de outra forma, como tabela-verdade

Descrição	**J**anela	**F**echadura	**M**ovimento	**Y**(alarme)
Tudo certo, tudo seguro.	fechada	trancada	sem	desliga
Como? Arrombaram a porta.	fechada	trancada	com	liga
Esqueci aberto. Volta pra fechar.	fechada	aberta	sem	liga
Tudo certo, sou eu na sala	fechada	aberta	com	desliga
Esqueci aberto. Volta pra fechar.	aberta	trancada	sem	liga
Como? Arrombaram a janela	aberta	trancada	com	liga
Esqueci aberto. Volta pra fechar.	aberta	aberta	sem	liga
Sou eu na sala tomando um ar	aberta	aberta	com	desliga



J	F	M	Y
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

Revisão: Análise e Projeto Digital (Refino)

- Quero ser avisado quando:
  - Algum movimento é detectado e está tudo fechado
  - Quando não há movimento e algo está aberto
  - Quando só a janela está aberta e há movimento

Note:
Saímos disso

Para isso

Podemos descrever as mesmas regras de outra forma, como tabela-verdade

Descrição	**J**anela	**F**echadura	**M**ovimento	**Y**(alarme)
Tudo certo, tudo seguro.	fechada	trancada	sem	desliga
Como? Arrombaram a porta.	fechada	trancada	com	liga
Esqueci aberto. Volta pra fechar.	fechada	aberta	sem	liga
Tudo certo, sou eu na sala	fechada	aberta	com	desliga
Esqueci aberto. Volta pra fechar.	aberta	trancada	sem	liga
Como? Arrombaram a janela	aberta	trancada	com	liga
Esqueci aberto. Volta pra fechar.	aberta	aberta	sem	liga
Sou eu na sala tomando um ar	aberta	aberta	com	desliga

J	F	M	Y
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

Outras representações para a função Y

J	F	M	Y
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

Forma canônica
Permite representar uma função de forma única e consistente

Algumas formas mais enxutas de representar a tabela-verdade:

Y(J,F,M) = (0,1,1,0,1,1,1,0)

Y(J,F,M) = (1,2,4,5,6)

Também podemos representar Y em suas formas canônicas

- Soma De Produtos (SDP/*SOP*) - (*Forma Normal Disjuntiva*)
  - Olho as saídas 1 (**mintermos**) e escrevo a expressão na forma SDP

Y = ~J.~F.M + ~J.F.~M + J.~F.~M + J.~F.M + J.F.~M

- Produtos De Somas (PDS/*POS*) - (*Forma Normal Conjuntiva*)
  - Olho as saídas 0 (**maxtermos**) e escrevo a expressão na forma PDS

Y = (J+F+M).(J+~F+~M).(~J+~F+~M)

Importante!!! para estar na forma canônica (SDP ou PDS), é preciso que todas as variáveis apareçam em todos os termos.

O trabalho está só começando...

J	F	M	Y
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

Dependendo da tecnologia-alvo:

Mantemos as expressões canônicas:

Ex.: ROM, EPROM

Ou minimizamos

Ex.: , PLA, PAL, lógica aleatória

Para minimizar:

Manualmente, usando os mapas de Karnaugh

Manualmente, usando minimizações algébricas

Usamos ferramentas CAD, como o alg. de Quine-McCluskey

J\FM	00	01	11	10
0	0	1	0	1
1	1	1	0	1

J\FM	00	01	11	10
0	0	1	0	1
1	1	1	0	1

J\FM	00	01	11	10
0	0	1	0	1
1	1	1	0	1

A expressão final minimizada é :

Y = ~F.M + F.~M + J.~M

O trabalho está só começando...

J	F	M	Y
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

Dependendo da tecnologia-alvo:

Mantemos as expressões canônicas:

Ex.: ROM, EPROM

Ou minimizamos

Ex.: , PLA, PAL, lógica aleatória

Para minimizar:

Manualmente, usando os mapas de Karnaugh

Manualmente, usando minimizações algébricas

Usamos ferramentas CAD, como o alg. de Quine-McCluskey

J\FM	00	01	11	10
0	0	1	0	1
1	1	1	0	1

J\FM	00	01	11	10
0	0	1	0	1
1	1	1	0	1

J\FM	00	01	11	10
0	0	1	0	1
1	1	1	0	1

A expressão final minimizada é :

Y = ~F.M + F.~M + J.~M

Note:
Saímos disso

Para isso

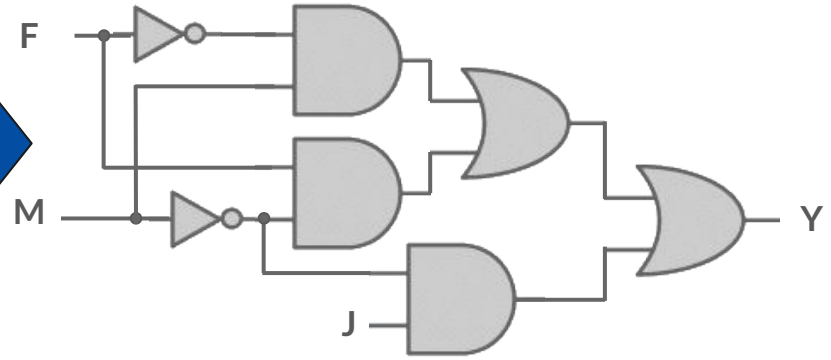
A expressão final minimizada é :

$$Y = \sim F.M + F\sim M + J.\sim M$$

Será que dá pra fazer menor?

J	F	M	Y
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

E chegamos nisso



E onde entra uma HDL como a Verilog?

Conceito de HDL

As linguagens que permitem a descrição textual de circuitos são chamadas HDL (*Hardware Description Language*).
Exemplos:

SystemVerilog VHDL Verilog®

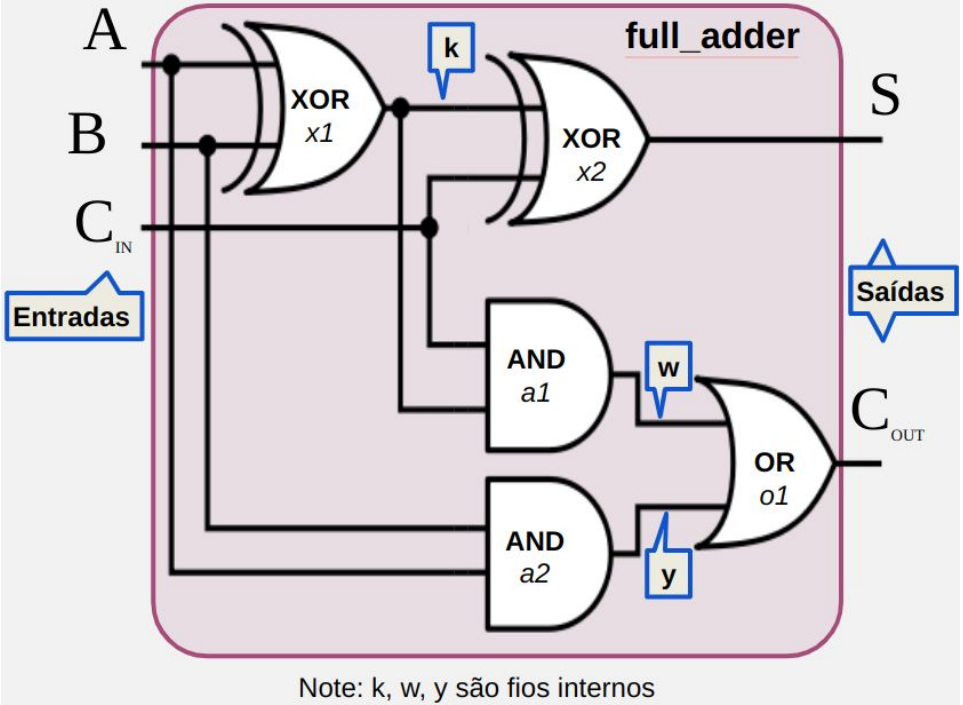
É inviável desenvolver os modernos circuitos integrados utilizando apenas modelagem esquemática (desenho de portas e interconexões).

Pentium IV (ano 2000) → $4x10^7$ portas

A descrição textual facilita a modelagem do projeto.

Representação de circuitos!

Modelagem estrutural



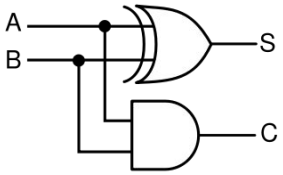
Note: k, w, y são fios internos

```
// declaramos o módulo e listamos todos os seus pinos
module full_adder (S, Cout, A, B, Cin);

  // informamos o sentido de cada pino
  input A;     // A é uma entrada
  input B;     // B é uma entrada
  input Cin;   // Cin é uma entrada
  output S;    // S é uma saída
  output Cout; // Cout é uma saída

  // declaramos os fios internos
  wire k, w, y;

  // instanciamos os elementos, ligando pinos e fios
  xor x1 (k, A, B);    // k ← A xor B
  xor x2 (S, k, Cin);  // S ← k xor Cin
  and a1 (w, k, Cin);  // w ← k and Cin
  and a2 (y, A, B);    // y ← A and B
  or  o1 (Cout, w, y); // Cout ← w and y
endmodule
```

Netlist: portas e ligações

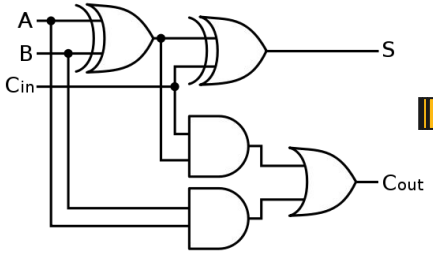
Representação de circuitos!

Modelagem dataflow



Meio Somador

```
module half_adder(
        output S, C,
        input A, B );
    assign S = A ^ B;
    assign C = A & B;
endmodule
```



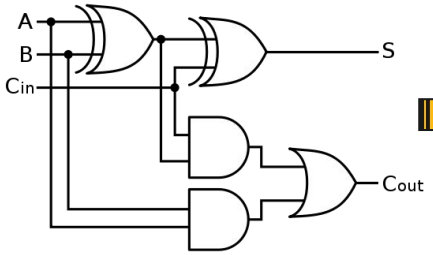
Somador Completo

```
module full_adder(
        output S, Cout,
        input A, B, Cin);
    assign S = A ^ B ^ Cin;
    assign C = ((A ^ B) & Cin) | (A & B);
endmodule
```

Expressões lógicas

Representação de circuitos!

Modelagem Comportamental



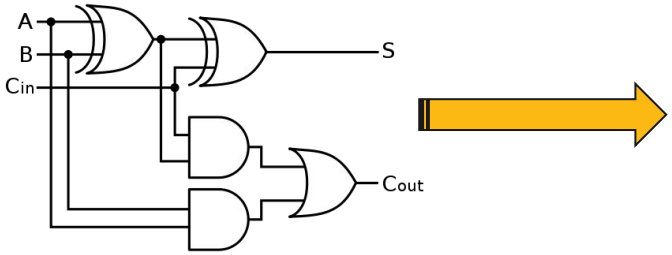
Somador Completo

```
module full_adder(
    input A, B, Cin,
    output reg S, Cout);
    always @(*) begin
        case ({A, B, Cin})
            3'b000: begin S = 0; Cout = 0; end
            3'b001: begin S = 1; Cout = 0; end
            3'b010: begin S = 1; Cout = 0; end
            3'b011: begin S = 0; Cout = 1; end
            3'b100: begin S = 1; Cout = 0; end
            3'b101: begin S = 0; Cout = 1; end
            3'b110: begin S = 0; Cout = 1; end
            3'b111: begin S = 1; Cout = 1; end
            default: begin S = 0; Cout = 0; end
        endcase
    end
endmodule
```

A tabela-verdade

Representação de circuitos!

Modelagem Comportamental

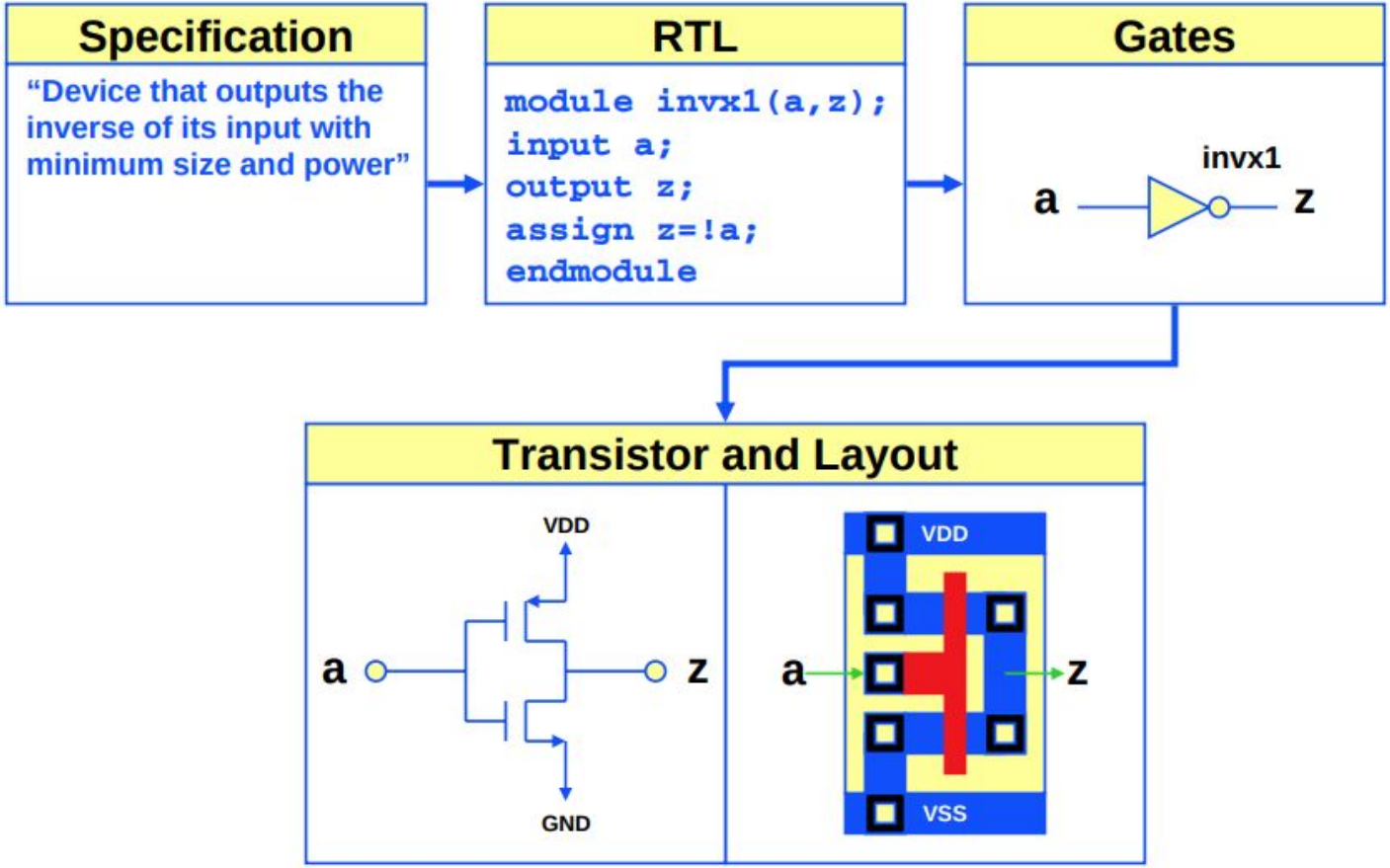


Somador Completo

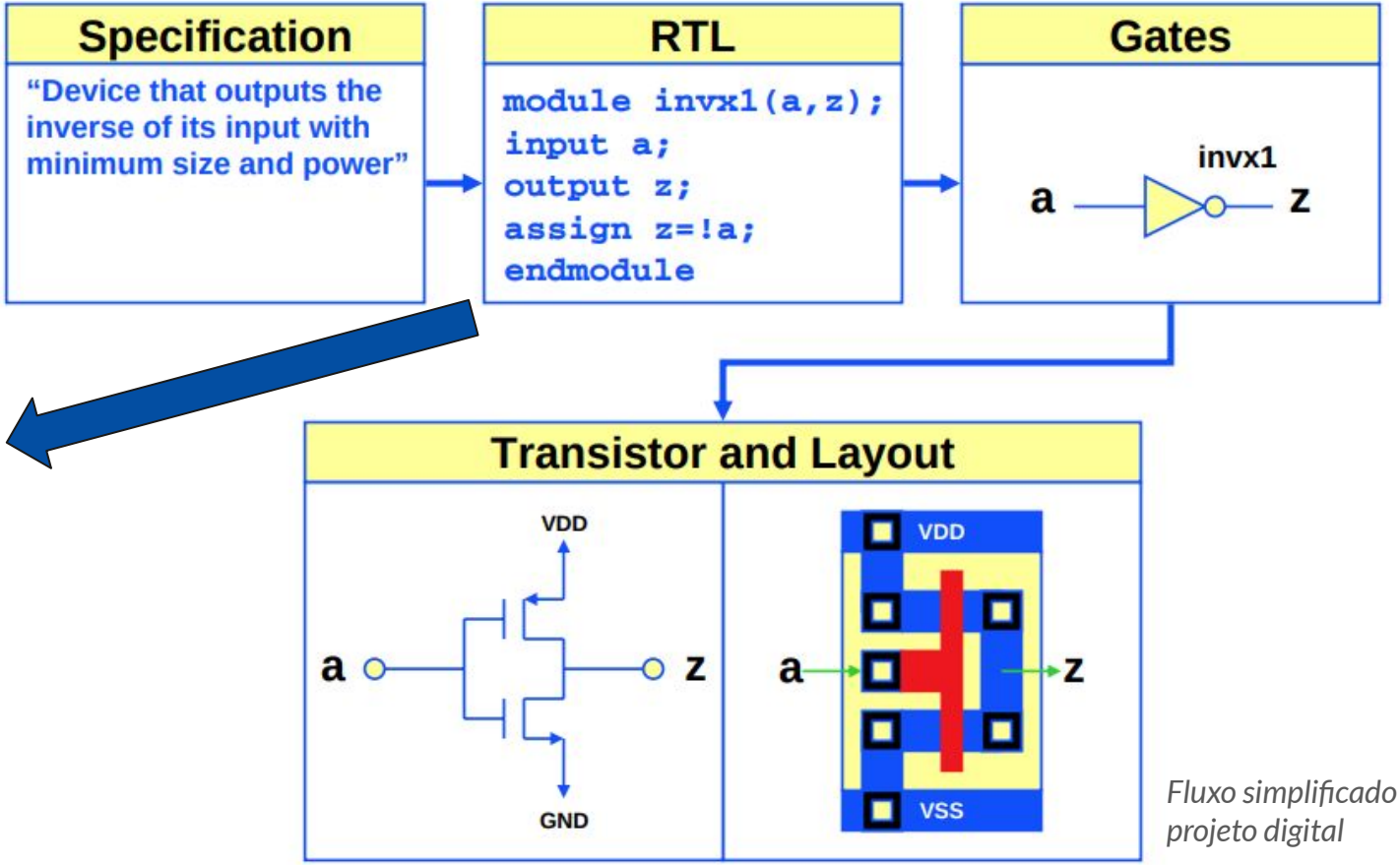
```
module full_adder(
    input A, B, Cin,
     output reg S, Cout);
     always @(*) begin
          {Cout, S} = A + B + Cin;
    end
endmodule
```

O algoritmo

Para que serve uma HDL?



Para que serve uma HDL?



Código (HDL) Verilog
Captura a especificação de maneira formal. Serve como entrada para ferramentas de automação do projeto.

Fluxo simplificado do projeto digital

Representação de dados e operações

Convenções Léxicas

Espaço em branco

Espaços em branco são ignorados em Verilog, exceto quando separam tokens, ou em Strings.

Comentários

Podem ser de uma linha (//) ou de múltiplas linhas(/* ... */), ao estilo de comentários em C, Java.

Operadores

São de três tipos: unário, binário e ternário. Operadores unários precedem os operandos, operadores binários aparecem entre os operandos. O operador ternário têm dois símbolos que ficam entre os operandos.

Especificação de números

Há dois tipos de especificação de números: dimensionado (*sized*) e não dimensionado (*unsized*).

Números dimensionados (*sized*)

São representados no formato *<size>'<base format> <number>*

Ex.: 4'b1111, 12'habc, 16'd255

Números dimensionados (*sized*)

Números especificados sem a descrição de tamanho possuem o número de bits definidos pelo simulador ou máquina. (pelo menos 32 bits)

Ex.: 23456, 'hc3, 'o21

Convenções Léxicas

Valores Z ou X

Verilog tem dois símbolos para desconhecido (X) e alta impedância (Z). São muito importantes para representar circuitos reais. Representam 4 bits se aparecem em uma especificação hexadecimal, 3 se na especificação octal, ou 1 bit se na especificação binária.

Ex.: 12'h13x, 16'bz

Números negativos

Números negativos podem ser especificados colocando um sinal de menos (-) antes da dimensão de uma constante. O número que especifica o tamanho é sempre positivo. É ilegal ter um sinal de menos entre o *<base format>* e o *<numero>*

Ex.: -6d3, 4'd-2 (ilegal)

Caractere underscore (_)

É permitido em qualquer lugar da especificação de um número, exceto no começo. Usado para ajudar na legibilidade, e são ignorados.

Ex.: 12'b1111_0000_1010

Identificadores e palavras reservadas

Verilog é *case sensitive*. Todas as palavras reservadas são escritas em minúsculo. Consulte o manual.

Tipos de Dados

Conjunto de valores

Verilog suporta 4 valores para modelar hardware real: 0, 1, x , z; respectivamente: 0 lógico (false), 1 lógico (true), valor desconhecido, e alta impedância.

Ex.: 12'h13x, 16'bz

Net

Representam conexões entre elementos de hardware. Assume o valor do seu driver. São declaradas usando a palavra-chave ***wire***.

Registrador

Representam elementos de armazenamento de dados. Mantém um valor até que outro valor seja atribuído. São declarados usando a palavra-chave ***reg***.

Vetores

Tipos de dados net ou reg podem ser declarados como vetores (com comprimento de múltiplos bits). Se a largura do vetor não é declarada (usando colchetes), o default é ter tamanho 1.

Ex.: wire [7:0] bus; // barramento de 8 bits.

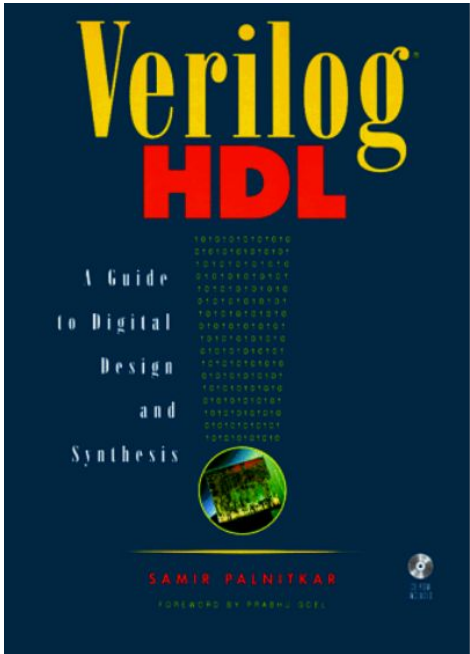
Tipos de Dados

Inteiro, Real e Registro de Tempo

Os tipos de dados integer, real e time são tipos de dados suportados em Verilog, muito úteis na simulação. Possuem o tamanho atrelado ao tamanho da palavra da máquina host, mas são de pelo menos 32 bits. O tipo time é usado para guardar o tempo de simulação. A diretiva de sistema $time deve ser chamada para retornar o tempo de simulação.

Consulte no livro detalhes sobre Arrays, Memórias, Parâmetros, e Strings (Capítulo 3)

Consulte também sobre Tarefas de Sistema e Diretivas de Compilação (Capítulo 3)



Executando
(por simulação)

A partir da descrição Verilog...

Qual devo usar?
O que te deixar mais confortável

Estilos distintos de descrever o mesmo hardware

Descrição dataflow

```
1   module mux2x1_1bit(
2       output y,
3       input sel,
4       input a,
5       input b
6   );
7
8       assign  y = ~sel&a | sel&b;
9
10  endmodule
```

Descrição estrutural

```
1   module mux2x1 (
2       input A,    // Entrada A
3       input B,    // Entrada B
4       input sel,  // Entrada de seleção
5       output Y    // Saída
6   );
7       wire not_sel;                 // Fio para a saída do NOT da seleção
8       wire and1_out;                // Fio para a saída da primeira porta AND
9       wire and2_out;                // Fio para a saída da segunda porta AND
10      not u0 (not_sel, sel);        // NOT da entrada de seleção
11      and u1 (and1_out, A, not_sel);// Primeira porta AND (A e ~sel)
12      and u2 (and2_out, B, sel);    // Segunda porta AND (B e sel)
13      or u3 (Y, and1_out, and2_out);// Porta OR para combinar as saídas das portas AND
14  endmodule
```

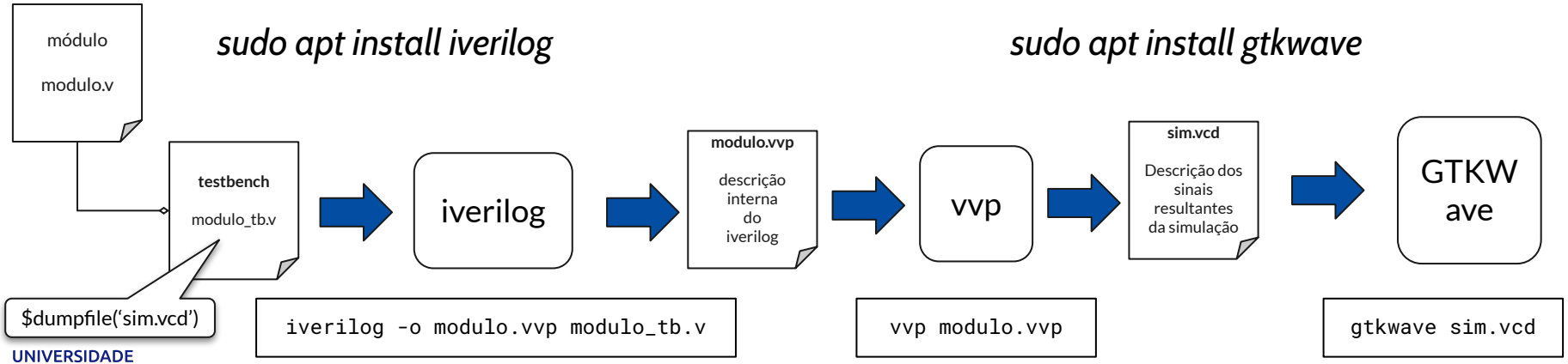
Com as ferramentas devidamente instaladas...

- **Icarus Verilog** → free compiler implementation for the IEEE-1364 Verilog hardware description language. Icarus Verilog is intended to compile ALL of the Verilog HDL, as described in the IEEE-1364 standard. Of course, it's not quite there yet.
  - https://github.com/steveicarus/iverilog

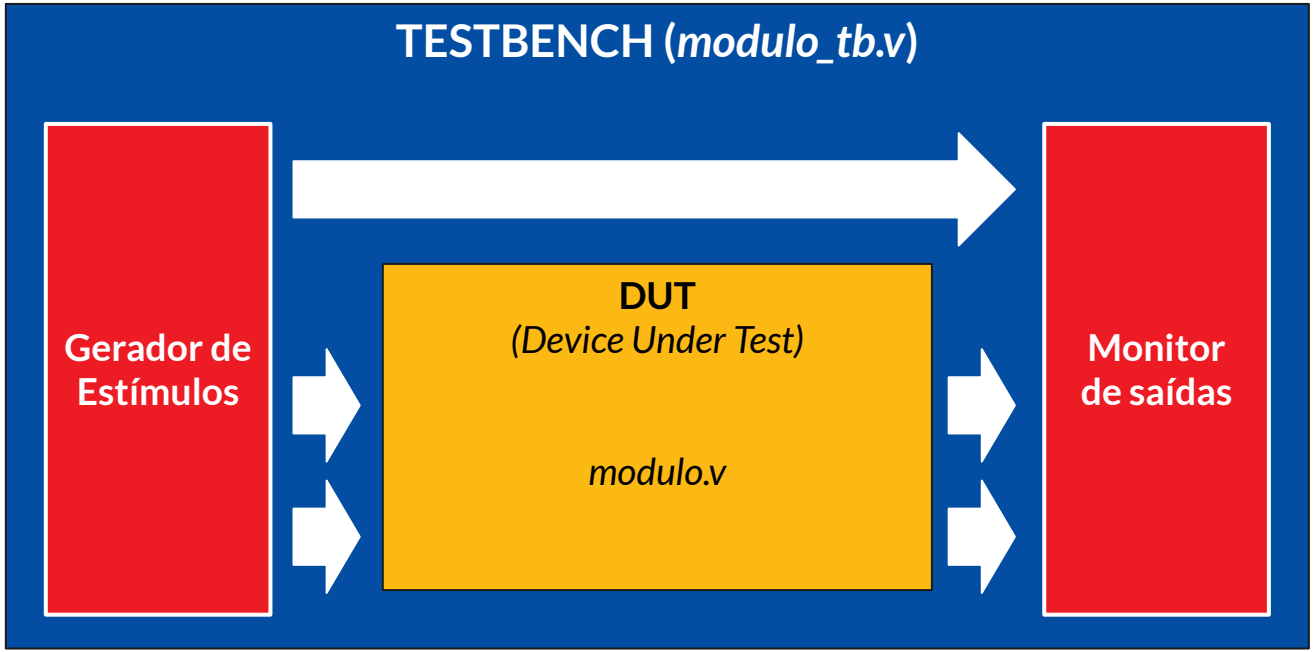
- **GTKWave** → a fully featured GTK+ based wave viewer for Unix, Win32, and Mac OSX which reads LXT, LXT2, VZT, FST, and GHW files as well as standard Verilog VCD/EVCD files and allows their viewing.
  - https://gtkwave.sourceforge.net/

sudo apt install iverilog

sudo apt install gtkwave



Criar o testbench simplificado a seguir



Gerador de Estímulos: gera entradas para o DUT e as repassa para o Monitor

Monitor: Calcula a saída correta, e a compara as saídas gerada pelo DUT.

Dever de casa

Implemente um módulo somador de 4 bits em Verilog, implemente seu testbench, e realize a simulação usando Icarus Verilog e Gtkwave para visualizar as formas de onda.

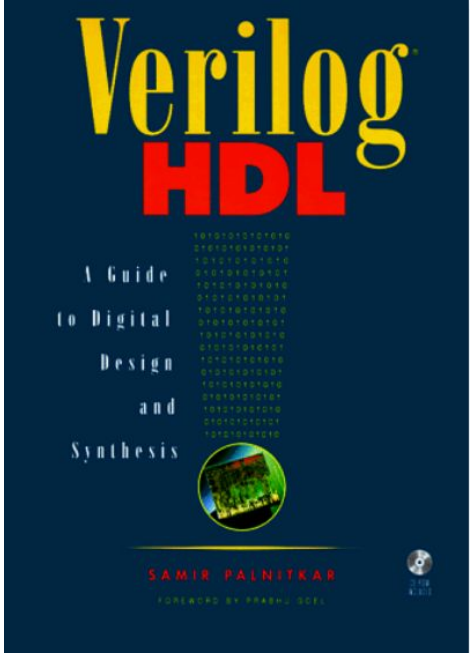
1) Realize a simulação exaustiva (todos as combinações de valores de A e B). Quanto tempo levou a simulação?
2) Realize a simulação randomizada. Como garantir que a implementação está correta?
3) Pesquise sobre SystemVerilog e SystemVerilog *assertions*. Pense sobre como as *assertions* poderia ser usada para automatizar a verificação funcional.

Referências

PALNITKAR, Samir. Verilog HDL: a guide to digital design and synthesis. Prentice Hall Professional, 2003.

IEEE. IEEE Standard Verilog Hardware Description Language, 2001

Hora-Trabalho de Hoje



Leia o capítulo 3

Dúvidas?

Na próxima aula...

Visão geral de Arquitetura de Computadores: Aula 1

Não falte! 😉

Obrigado pela atenção

UNIVERSIDADE
FEDERAL DE
SERGIPE