

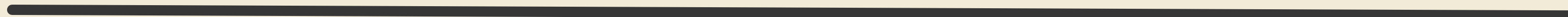


# Introdução ao **VERILOG COMPORTAMENTAL** parte um



# PRAZER, CAMILA

Graduanda em Engenharia de  
Computação pela UEFS e ex-monitora de  
Cálculo II.





# SUMÁRIO

## da oficina (parte um)

- Revisão do Quartus
- Conceitos básicos
- Operadores
- Blocos de procedimentos
- Hierarquia
- Estrutura de decisão
  - if-else-if

# CONCEITOS BÁSICOS

- Verilog é uma **linguagem de descrição de hardware** que permite a simulação e síntese lógica e física de um sistema
- É um padrão IEEE 1364 – 2005
- O único hardware presente na FPGA é o que está descrito no Verilog
- Não há chamadas de sistema operacional!!
- Não há sequencialidade em Verilog. Tudo é executado em paralelo
- Todo o tempo do hardware é baseado no clock

# CONCEITOS BÁSICOS

- MODELOS:
  - **estrutural**: relaciona as entradas e saídas de um sistema por meio de barramentos e fios
  - **comportamental**: descreve o funcionamento do circuito. O papel de criação lógica é feita pelas ferramentas de síntese
- OPINIÃO: **SEJAM PARANOICOS**

# TIPOS DE DADOS

- ***nets***: representam uma conexão física entre estruturas e módulos.
  - *wires*
- ***parameters*** (parâmetros): usa constantes para parametrização dos módulos.
  - definidos globalmente por meio da palavra *parameter*
  - localmente (interno ao módulo) por meio da palavra *localparam*.
- ***reg*** (registradores/registros): unidade básica. Tamanho varia de acordo com a declaração
  - conjunto de flip flops que guardam valores

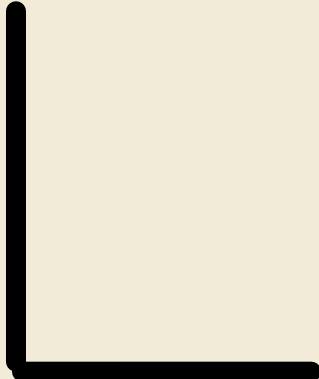
# TIPOS DE DADOS

Tipos de reg	Descrição
reg	Sem sinal
reg signed	Com sinal
integer	Sem sinal. Tamanho de 32 bits
real	Ponto flutuante com precisão dupla
time	Sem sinal. Tamanho de 64 bits

CUIDADO COM A DECLARAÇÃO DE REGISTRADORES COM SINAL. O DEFAULT É SEM SINAL



# OPERADORES

- 
- 1 Aritméticos
  - 2 Bit-wise, redução e deslocamento
  - 3 Relacional
  - 4 Condicional
  - 5 Igualdade
  - 6 Concatenação e replicação



# OPERADORES

Tipo	Operadores	
Concatenação e replicação	{ } { }	Prioridade alta
Inversão (unária)	! ~ + -	
Aritmético	** * + - / %	
Deslocamento	<< >> <<< >>>	
Relacional	< <= > >=	
Igualdade	= != === !==	
Bit a bit / redução	& ~& ^ ^~ ^~   ~	

Tipo	Operadores	
Lógicos	&&	
Condicional	ternário	Prioridade baixa

# OPERADORES

- Operador ternário
  - `assign out = {sel} ? value_if_true : value_if_false`
- Operador de replicação
  - `{12{1'b1}}` = `1'b_1111_1111_1111`
- Operador de concatenação
  - exemplo no quartus
- Cuidado do **deslocamento de valores com sinal**

# HIERARQUIA NO VERILOG

- **Módulos** são pedaços de hardware reutilizáveis
  - a cada instanciação, o compilador realizará uma cópia do hardware daquele módulo e gravará na FPGA
  - as instâncias de um módulo são idênticas, completas e concorrentes
- Módulos trocam informações entre si por meio de suas listas de portas
  - wires são necessários para realizar a interconexão entre os módulos
- Assemelham-se a funções:
  - possuem entradas, operação sobre estas entradas e geram saídas

# DECLARAÇÃO BÁSICA

```
module <nome_do_modulo> (<lista_de_portas>);  
  
<declaração_dos_tipos_das_portas> // portas INPUT/OUTPUT/INOUT  
  
<declaração_de_tipo> // WIRE,REG, INTEGER, etc  
  
<funcionalidade_do_circuito> // always/assign  
  
<especificação_de_tempo> // usado em simulação  
  
endmodule
```

[Exemplo em código aqui](#)

# TIPOS DE PORTAS

- **Input:** entrada de dados no módulo
- **Output:** saída de dados no módulo
  - pode ser do tipo reg
- **Inout:** canal bidirecional
  - não pode ser realizada a leitura e escrita simultaneamente
  - não devem ser do tipo registro
  - Uma abordagem recomendada é colocar um sinal de ENABLE para a porta de saída e alta impedância para a porta de entrada de dados

# CONEXÃO ENTRE MÓDULOS

Exemplo no Quartus: somador de 5 bits

# BLOCOS

- **initial** (processual): são usados para iniciar variáveis em simulação. Não são sintetizados e são executados apenas uma vez
- **always** (processual): executa todas as chamadas internas em loop
  - múltiplos blocos são executados concorrentemente
  - se um sinal recebe uma atribuição em um bloco, ele não pode receber outro assign em outro bloco
  - nos blocos always, os assigns devem ser feitos com o tipo reg. Não podem ser feitos com wire
- **begin...end** (sequencial): agrupam ações a serem executadas sequencialmente

# ATRIBUIÇÃO CONTÍNUA

- para lógica combinacional
- atribuição direta ao fio
  - `wire [7:0] sum_out = data_a + data_b;`
- com assign
  - `assign sum_out = sum_out + mult_out + data_out;`
  - variável à esquerda deve ser do tipo wire
  - variável à direita podem ser reg, wire ou chamadas de funções

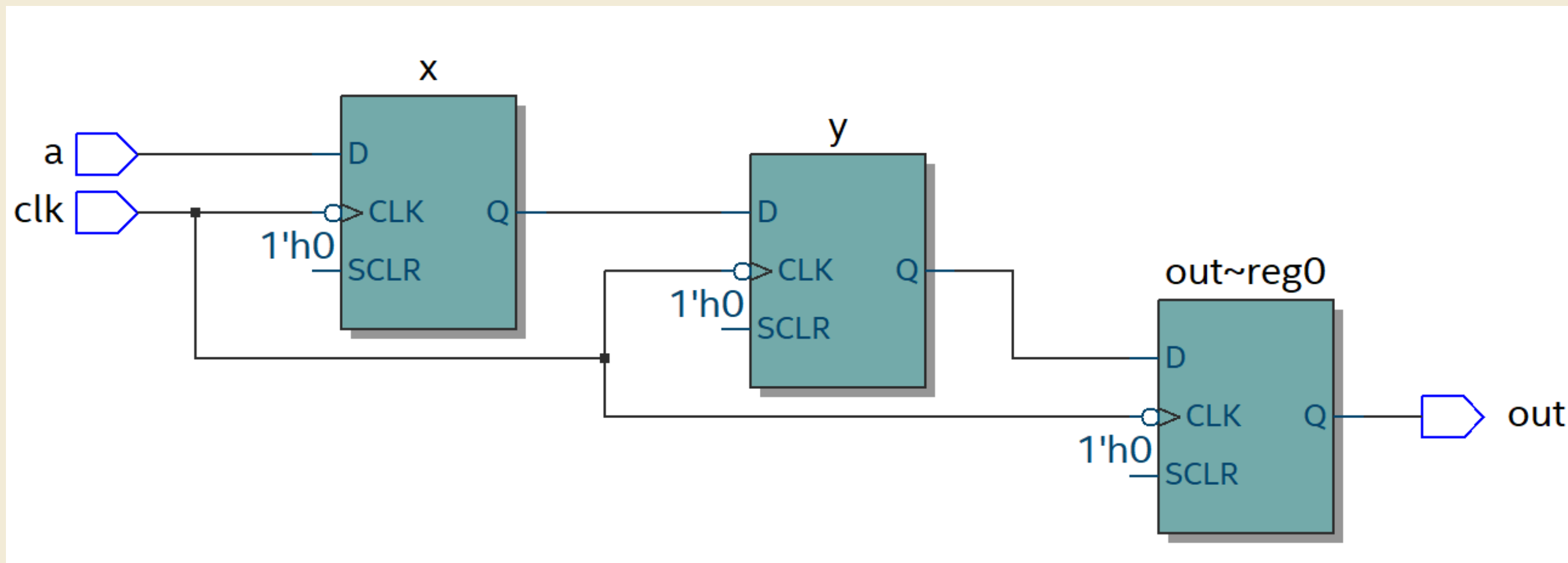


# ATRIBUIÇÕES BLOQUEANTES E NÃO BLOQUEANTES

- **REGRA:**
  - em implementações combinacionais, use atribuições bloqueantes
    - Exemplo: `sel = x;`
  - em implementações sequenciais, use atribuições não bloqueantes. Nestes casos, a ordem dos statements importa.
    - Exemplo: `sel <= x;`
- Qual a diferença? Visualização com a atividade prática 1

# ATIVIDADE PRÁTICA 1

Implemente o circuito da imagem abaixo utilizando a atribuição bloqueante e depois a atribuição não-bloqueante. Compare os RTLs



[Implementação aqui](#)

# IF-ELSE-IF

```
if (condicao) begin
    <declaracao unica>
end else if (condicao2) begin
    <declaracoes>
end else begin
    <declaracoes>
end
```

# ATIVIDADE PRÁTICA 2.1

Altere o código da atividade prática para adicionar um sinal de reset (borda de subida)

Implementação aqui

## ATIVIDADE PRÁTICA 2.2

Implemente em Verilog um módulo que execute as operações de soma (0) e subtração (1), tendo operandos com 8 bits. Verifique também casos de overflow.

Implementação aqui



# Introdução ao **VERILOG COMPORTAMENTAL** parte dois



# SUMÁRIO

## da oficina (parte dois)

- Revisão do dia um
- Estruturas de decisão
  - case
- Máquinas de estado finitas
  - Moore
  - Mealy
- Estruturas de repetição

# CASE

```
case [<expressao>]
  case_item1 :
    <declaracao unica>
  case_item2 :
    begin
      <declaracoes multiplas>
    end
  default:
    <declaracao unica>
endcase
```

**Cuidado com if's e case's incompletos:** o Verilog pode implementar estes casos como LATCHES, atrapalhando a temporização e o funcionamento do seu sistema

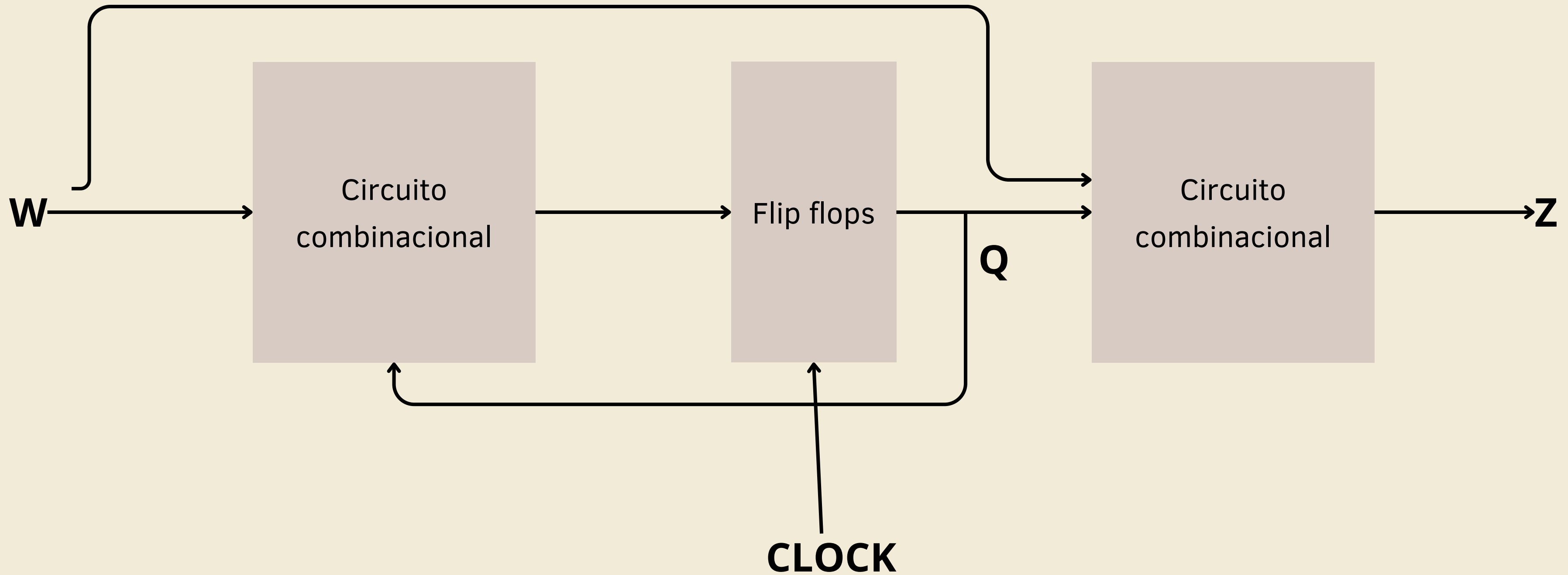


# ATIVIDADE PRÁTICA 3

Implemente um multiplexador 6 para 1 utilizando a estrutura case

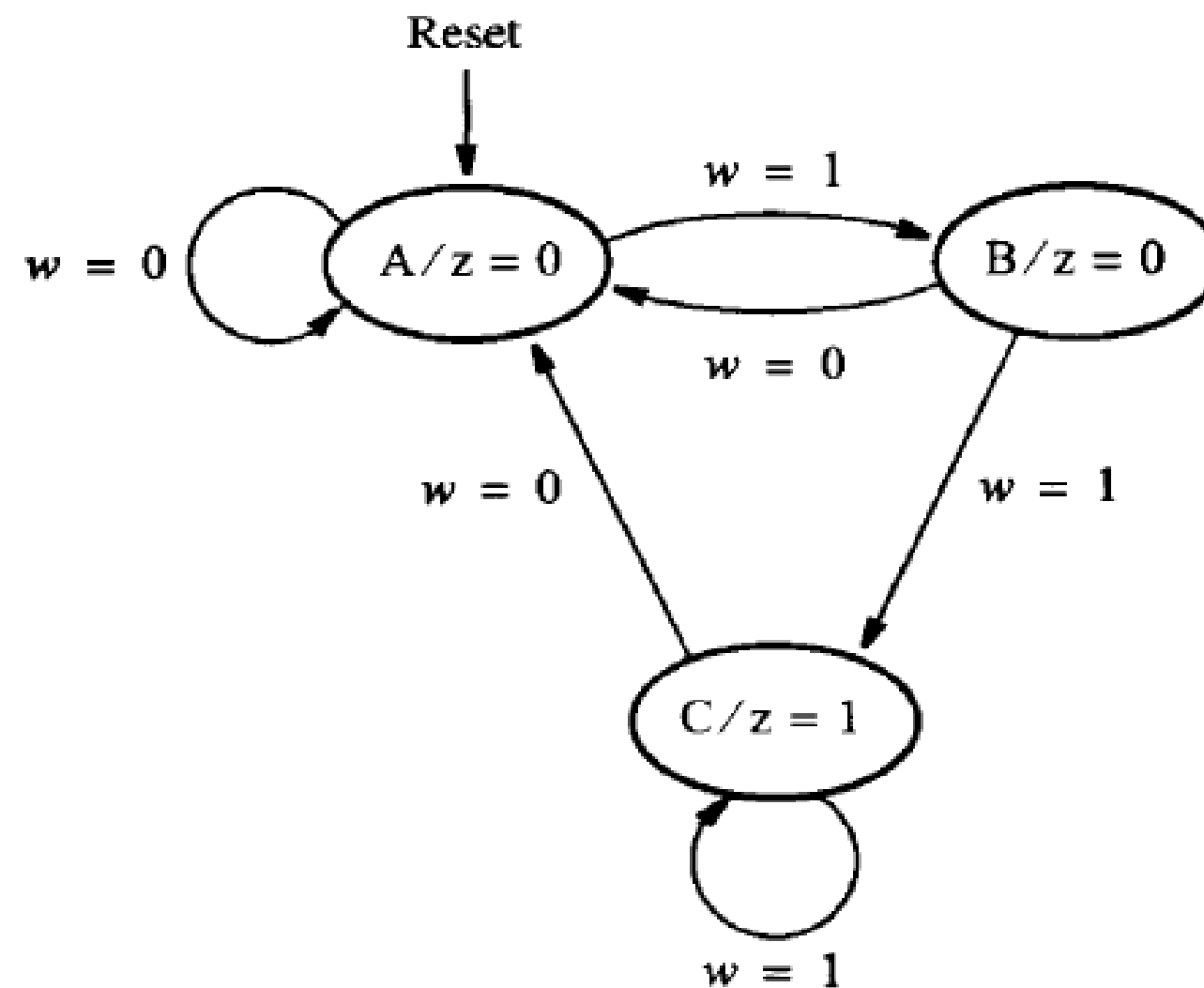
Implementação aqui

# MÁQUINAS DE ESTADOS FINITAS



# MÁQUINAS DE ESTADOS FINITAS

- **Tipo Moore:** saídas depende apenas dos estados

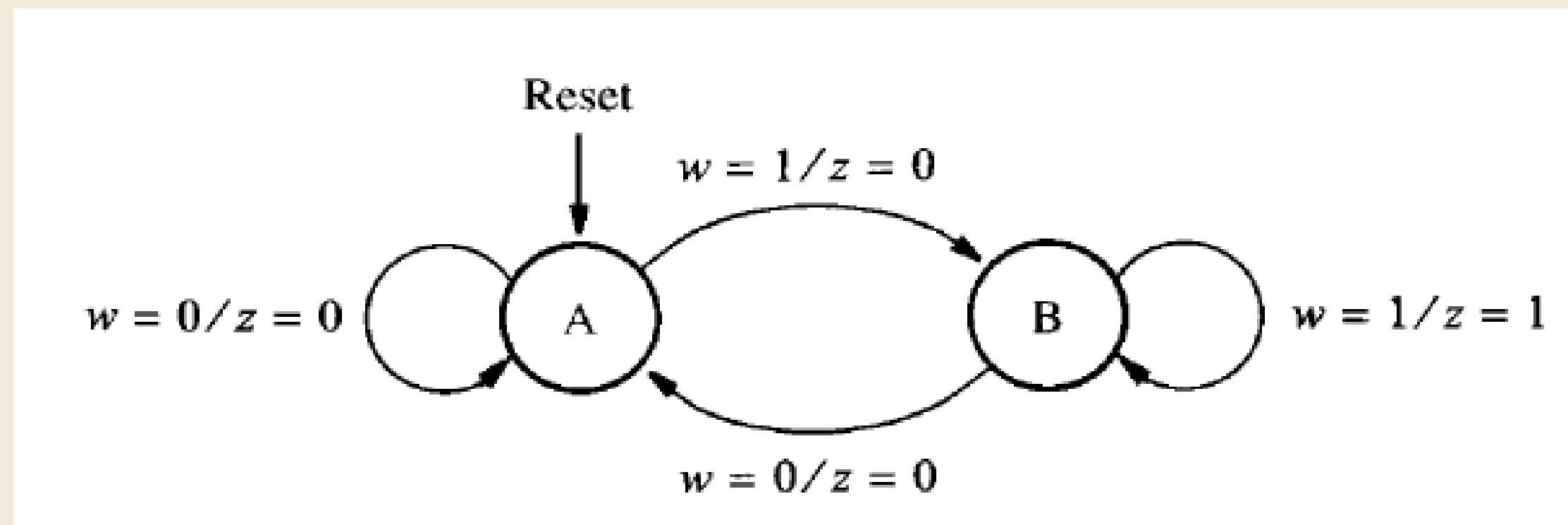


Fonte: Wakerly  
(2007, p.482)

Implementação aqui

# MÁQUINAS DE ESTADOS FINITAS

- **Tipo Mealy:** saídas dependem dos estados e das entradas



Fonte: Wakerly (2007, p.497)

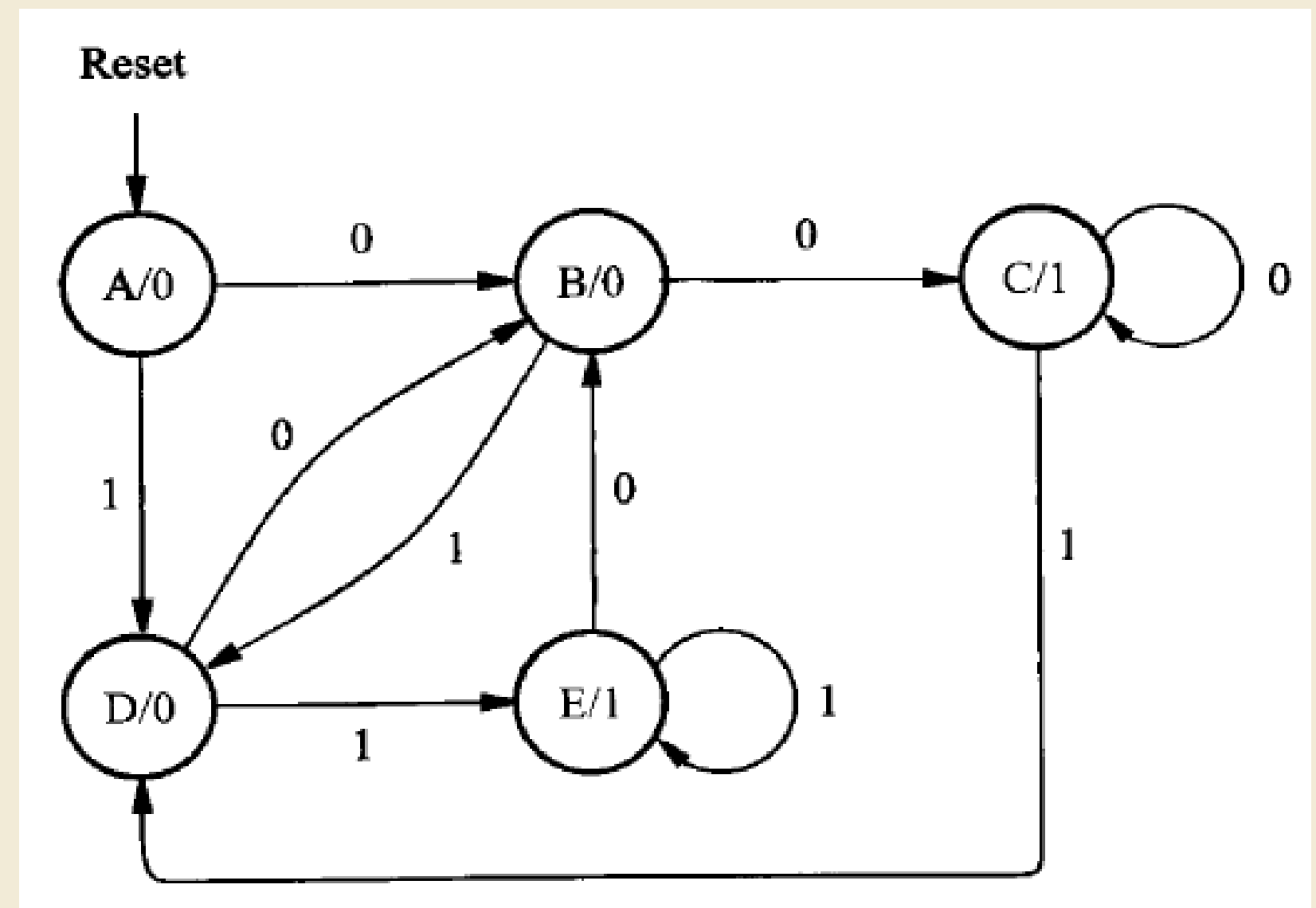
Implementação aqui

# IMPLEMENTAÇÃO DE MEFS COM VERILOG COMPORTAMENTAL

- Utiliza-se usualmente as estruturas case e if-else-se
  - **sempre cuidado com os casos definidos**
- Parametrização pode auxiliar na implementação
- Exemplo no Quartus

# ATIVIDADE PRÁTICA 4

Implemente um MEF do tipo Moore que tenha um input  $w$  e um output  $z$ . A máquina é um detector de sequência que produz  $z=1$  quando os dois últimos valores de  $w$  forem 00 ou 11. Caso contrário produz  $z=0$ .



Fonte: Wakerly (2007, p.562)

[Implementação aqui](#)

# REFERÊNCIAS

Universidade Federal de Pernambuco. Monitoria de SD: Linguagem Verilog. 2011. Disponível em: <<https://www.cin.ufpe.br/~voo/sd/Aula6>>

LIMA, Thiago. Tutorial de Verilog - Operadores. Embarcados, 2015. Disponível em: <<https://embarcados.com.br/tutorial-de-verilog-operadores/>>

PEREIRA, Rodrigo. PROCESSADORES PROGRAMÁVEIS – como projetar um processador em VERILOG – Codificação – parte 3. Embarcados, 2015. Disponível em: <<https://embarcados.com.br/processador-em-verilog-3/#Operadores-em-VERILOG>>

NANDLAND. Introduction to Verilog. [s.d.]. Disponível em: <[https://nandland.com/introduction-to-verilog-for-beginners-with-code-examples/#google\\_vignette](https://nandland.com/introduction-to-verilog-for-beginners-with-code-examples/#google_vignette)>

BROWN, S.; VRANESIC, Z. Fundamentals of Digital Logic with Verilog Design. 2. ed. Canada:McGraw Hill Higher Education, 2007.