MC542

Organização de Computadores
Teoria e Prática

2007

Prof. Paulo Cesar Centoducatte

ducatte@ic.unicamp.br

www.ic.unicamp.br/~ducatte

MC542

Circuitos Lógicos

VHDL
Flip-Flops, Registradores, Máquinas de Estados

Título do Capítulo Abordado
Sumário

- Instanciação de um FF D da Biblioteca
- Gated D latch
- Flip-Flop D
- Flip-Flop D Usando Wait Until
- Flip-Flop D com Reset Assíncrono
- Flip-Flop D com Reset Síncrono
- Módulo lpm_shiftreg
- Registrador de 8 bits com Clear Assíncrono
- Registrador de N bits com Clear Assíncrono
- Flip-flop D com um mux 2:1 na entrada D
- Shift Register Usando muxdff como Componente
- Shift Register Código Alternativo
- Registrador N bits com Saída Tri-state
- Código VHDL para FSM de Moore
- Código VHDL para FSM de Mealy

Instanciação de um FF D da Biblioteca

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
LIBRARY altera;
USE altera.maxplus2.all;

ENTITY flipflop IS
    PORT ( D, Clock         : IN    STD_LOGIC;
           Resetn, Presetn  : IN    STD_LOGIC;
           Q                : OUT   STD_LOGIC );
END flipflop;

ARCHITECTURE Structure OF flipflop IS
BEGIN
    dff_instance: dff PORT MAP ( D, Clock, Resetn, Presetn, Q );
END Structure;
```

Gated D latch

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY latch IS
    PORT ( D, Clk   : IN     STD_LOGIC;
              Q     : OUT    STD_LOGIC);
END latch;

ARCHITECTURE Behavior OF latch IS
BEGIN
    PROCESS ( D, Clk )
    BEGIN
        IF Clk = '1' THEN
            Q <= D;
        END IF;
    END PROCESS;
END Behavior;
```

Flip-Flop D

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY flipflop IS
    PORT ( D, Clock    : IN     STD_LOGIC;
               Q       : OUT    STD_LOGIC);
END flipflop;

ARCHITECTURE Behavior OF flipflop IS
BEGIN
    PROCESS ( Clock )
    BEGIN
        IF Clock'EVENT AND Clock = '1' THEN
            Q <= D;
        END IF;
    END PROCESS;
END Behavior;
```

Flip-Flop D Usando Wait Until

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY flipflop IS
    PORT (D, Clock   : IN    STD_LOGIC;
          Q          : OUT   STD_LOGIC );
END flipflop;

ARCHITECTURE Behavior OF flipflop IS
BEGIN
    PROCESS
    BEGIN
        WAIT UNTIL Clock'EVENT AND Clock = '1';
        Q <= D;
    END PROCESS;
END Behavior;
```

Flip-Flop D com Reset Assíncrono

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY flipflop IS
    PORT ( D, Resetn, Clock  : IN     STD_LOGIC;
           Q                 : OUT STD_LOGIC);
END flipflop;

ARCHITECTURE Behavior OF flipflop IS
BEGIN
    PROCESS ( Resetn, Clock )
    BEGIN
        IF Resetn = '0' THEN
            Q <= '0';
        ELSIF Clock'EVENT AND Clock = '1' THEN
            Q <= D;
        END IF;
    END PROCESS;
END Behavior;
```

Flip-Flop D com Reset Síncrono

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY flipflop IS
    PORT (D, Resetn, Clock : IN     STD_LOGIC;
              Q            : OUT    STD_LOGIC);
END flipflop;

ARCHITECTURE Behavior OF flipflop IS
BEGIN
    PROCESS
    BEGIN
        WAIT UNTIL Clock'EVENT AND Clock = '1';
        IF Resetn = '0' THEN
            Q <= '0';
        ELSE
            Q <= D;
        END IF;
    END PROCESS;
END Behavior;
```

Módulo lpm_shiftreg

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
LIBRARY lpm;
USE lpm.lpm_components.all;

ENTITY shift IS
    PORT (  Clock          : IN     STD_LOGIC;
            Reset          : IN     STD_LOGIC;
            Shiftin, Load  : IN     STD_LOGIC;
            R              : IN     STD_LOGIC_VECTOR(3 DOWNTO 0);
            Q              : OUT    STD_LOGIC_VECTOR(3 DOWNTO 0) );
END shift;

ARCHITECTURE Structure OF shift IS
BEGIN
    instance: lpm_shiftreg
        GENERIC MAP (LPM_WIDTH => 4, LPM_DIRECTION => "RIGHT")
        PORT MAP (data => R, clock => Clock, aclr => Reset,
            load => Load, shiftin => Shiftin, q => Q );
END Structure;
```

Registrador de 8 bits com Clear Assíncrono

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY reg8 IS
    PORT (  D               : IN    STD_LOGIC_VECTOR(7 DOWNTO 0);
            Resetn, Clock   : IN    STD_LOGIC;
            Q               : OUT   STD_LOGIC_VECTOR(7 DOWNTO 0) );
END reg8;

ARCHITECTURE Behavior OF reg8 IS
BEGIN
    PROCESS ( Resetn, Clock )
    BEGIN
        IF Resetn = '0' THEN
            Q <= "00000000";
        ELSIF Clock'EVENT AND Clock = '1' THEN
            Q <= D;
        END IF;
    END PROCESS;
END Behavior;
```

Registrador de N bits com Clear Assíncrono

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY regn IS
   GENERIC ( N : INTEGER := 16 );
   PORT (D              : IN  STD_LOGIC_VECTOR(N-1 DOWNTO 0);
         Resetn, Clock : IN  STD_LOGIC;
         Q             : OUT  STD_LOGIC_VECTOR(N-1 DOWNTO 0) );
END regn;

ARCHITECTURE Behavior OF regn IS
BEGIN
    PROCESS ( Resetn, Clock )
    BEGIN
        IF Resetn = '0' THEN
            Q <= (OTHERS => '0');
        ELSIF Clock'EVENT AND Clock = '1' THEN
            Q <= D;
        END IF;
    END PROCESS;
END Behavior;
```

Flip-flop D com um mux 2:1 na entrada D

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY muxdff IS
    PORT ( D0, D1, Sel, Clock  : IN STD_LOGIC;
           Q                   : OUT   STD_LOGIC );
END muxdff;

ARCHITECTURE Behavior OF muxdff IS
BEGIN
    PROCESS
    BEGIN
        WAIT UNTIL Clock'EVENT AND Clock = '1';
        IF Sel = '0' THEN
            Q <= D0;
        ELSE
            Q <= D1;
        END IF;
    END PROCESS;
END Behavior;
```

Shift Register
Usando muxdff como Componente

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY shift4 IS
    PORT ( R            : IN       STD_LOGIC_VECTOR(3 DOWNTO 0);
           L, w, Clock  : IN       STD_LOGIC;
           Q            : BUFFER   STD_LOGIC_VECTOR(3 DOWNTO 0));
END shift4;

ARCHITECTURE Structure OF shift4 IS
    COMPONENT muxdff
        PORT ( D0, D1, Sel, Clock  : IN    STD_LOGIC;
               Q                   : OUT   STD_LOGIC );
    END COMPONENT;
BEGIN
    Stage3: muxdff PORT MAP ( w, R(3), L, Clock, Q(3) );
    Stage2: muxdff PORT MAP ( Q(3), R(2), L, Clock, Q(2) );
    Stage1: muxdff PORT MAP ( Q(2), R(1), L, Clock, Q(1) );
    Stage0: muxdff PORT MAP ( Q(1), R(0), L, Clock, Q(0) );
END Structure;
```

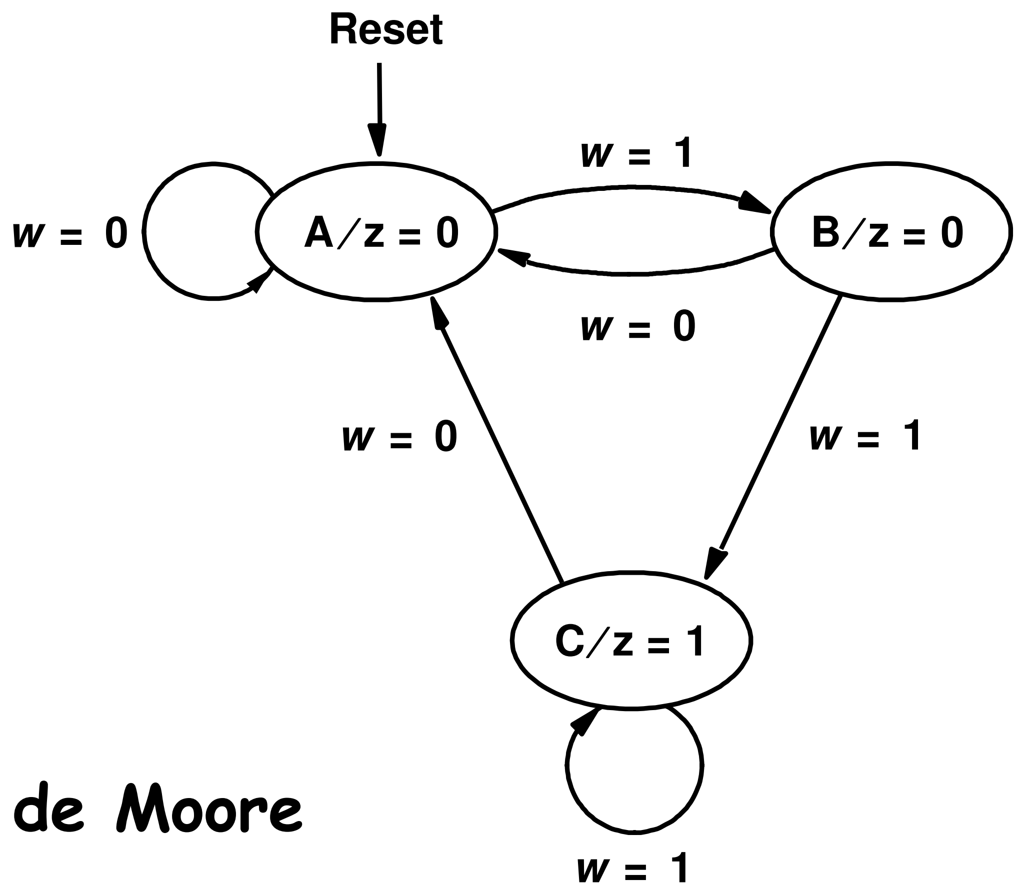
Shift Register
Código Alternativo

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY shift4 IS
    PORT ( R        : IN       STD_LOGIC_VECTOR(3 DOWNTO 0);
           Clock    : IN       STD_LOGIC;
           L, w     : IN       STD_LOGIC;
           Q        : BUFFER   STD_LOGIC_VECTOR(3 DOWNTO 0) );
END shift4;

ARCHITECTURE Behavior OF shift4 IS
BEGIN
    PROCESS
    BEGIN
        WAIT UNTIL Clock'EVENT AND Clock = '1';
        IF L = '1' THEN
            Q <= R;
        ELSE
            Q(0) <= Q(1);
            Q(1) <= Q(2);
            Q(2) <= Q(3);
            Q(3) <= w;
        END IF;
    END PROCESS;
END Behavior;
```

Registrador N bits
com Saída Tri-state

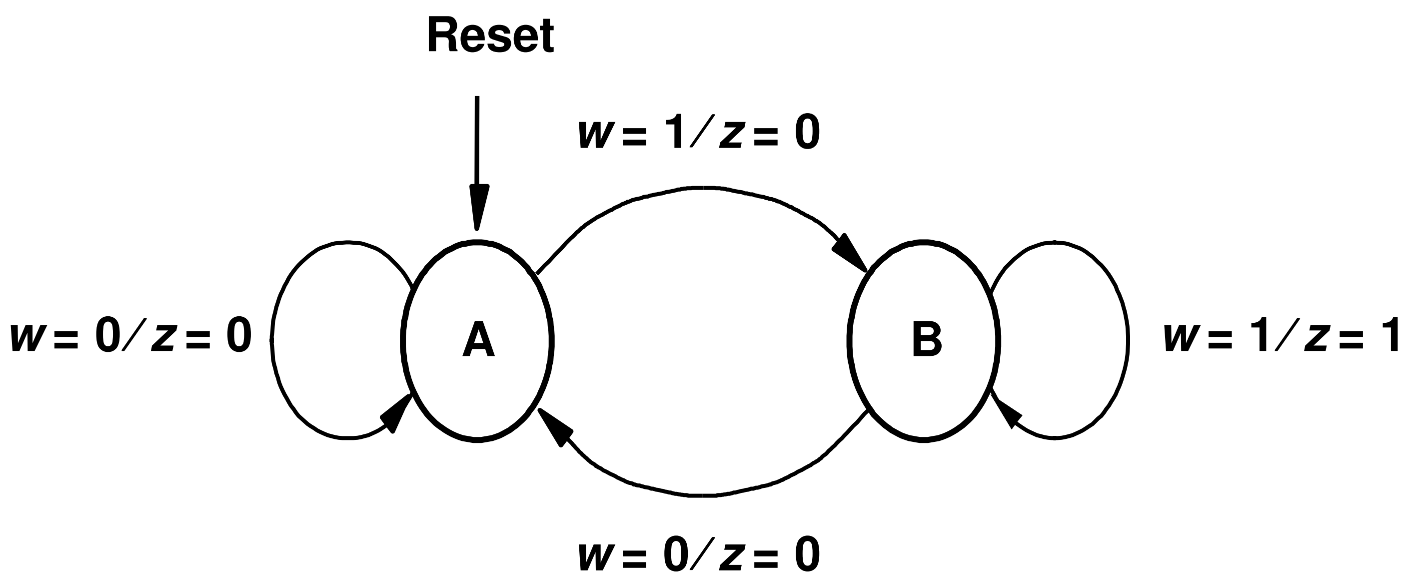
```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY trin IS
    GENERIC ( N : INTEGER := 8 );
    PORT ( X   : IN   STD_LOGIC_VECTOR(N-1 DOWNTO 0);
           E   : IN   STD_LOGIC;
           F   : OUT  STD_LOGIC_VECTOR(N-1 DOWNTO 0) );
END trin;

ARCHITECTURE Behavior OF trin IS
BEGIN
    F <= (OTHERS => 'Z') WHEN E = '0' ELSE X;
END Behavior;
```

Projeto de Máquina de Estados Exemplo:



Máquina de Moore

Projeto de Máquina de Estados
Exemplo:



Máquina de Mealy

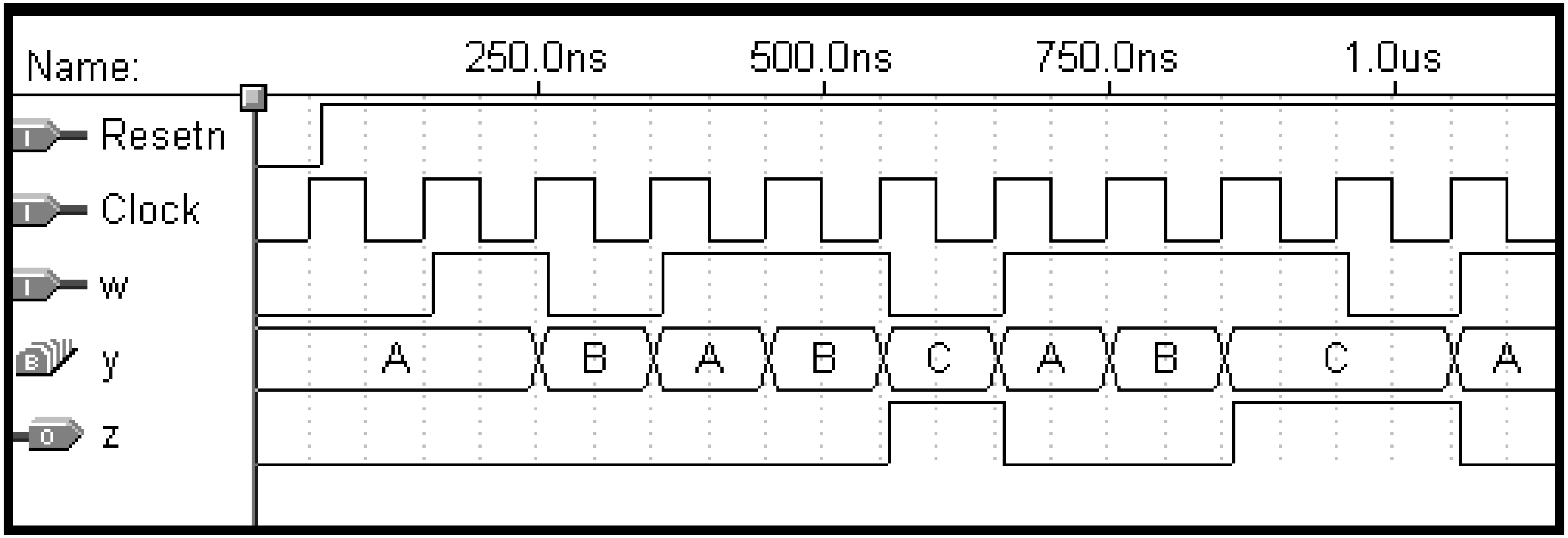
FSM de Moore

```
USE ieee.std_logic_1164.all;

ENTITY simple IS
   PORT (Clock, Resetn, w : IN     STD_LOGIC;
         z                : OUT    STD_LOGIC );
END simple;

ARCHITECTURE Behavior OF simple IS
   TYPE State_type IS (A, B, C);  -- Tipo Enumerado para
                                  -- definir os  Estados
   SIGNAL y : State_type;
BEGIN
   PROCESS ( Resetn, Clock )
   BEGIN
        IF Resetn = '0' THEN  -- A é o estado inicial
              y <= A;
        ELSIF (Clock'EVENT AND Clock = '1') THEN

con't ...
```

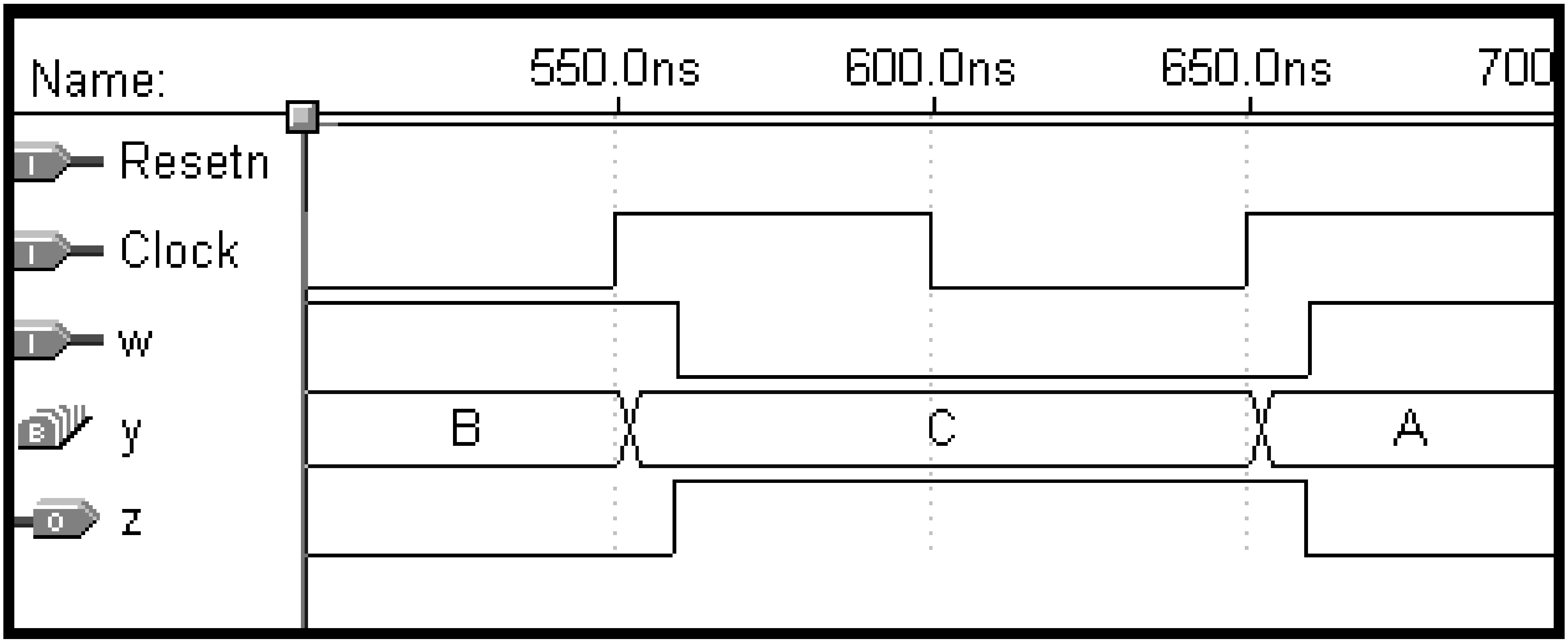
FSM de Moore

```
        CASE y IS
                WHEN A =>
                    IF w = '0'
                      THEN  y <= A;
                      ELSE  y <= B;
                    END IF;
                WHEN B =>
                    IF w = '0'
                      THEN  y <= A;
                      ELSE  y <= C;
                    END IF;
                WHEN C =>
                    IF w = '0'
                      THEN   y <= A;
                      ELSE   y <= C;
                    END IF;
                END CASE;
        END IF;
END PROCESS;
   z <= '1' WHEN y = C ELSE '0';
END Behavior;
```

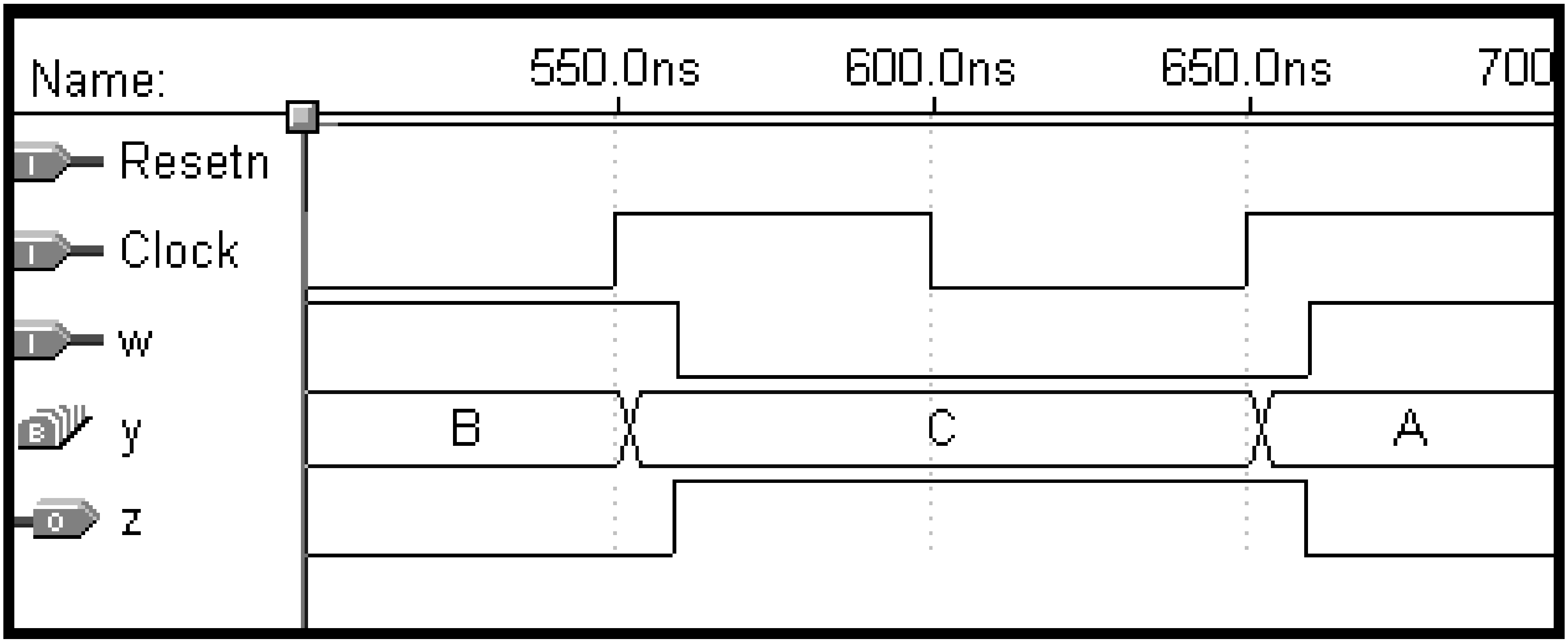
FSM de Moore - Simulação



FSM de Moore - Simulação



FSM de Moore
Simulação



FSM de Moore
Codificação Alternativa (2 processos)

```
USE ieee.std_logic_1164.all;

ENTITY simple IS
   PORT (Clock, Resetn, w : IN    STD_LOGIC;
         z                : OUT   STD_LOGIC );
END simple;

ARCHITECTURE Behavior OF simple IS
   TYPE State_type IS (A, B, C);

   SIGNAL y_present, y_next : State_type;
```

FSM de Moore
Codificação Alternativa (2 processos)

```
BEGIN
   PROCESS ( w, y_present )
   BEGIN
        CASE y_present IS
             WHEN A =>
                  IF w = '0' THEN
                      y_next <= A;
                  ELSE
                      y_next <= B;
                  END IF;
             WHEN B =>
                  IF w = '0' THEN
                      y_next <= A;
                  ELSE
                      y_next <= C;
                  END IF;
```

FSM de Moore
Codificação Alternativa (2 processos)

```
            WHEN C =>
                IF w = '0' THEN
                    y_next <= A;
                ELSE
                    y_next <= C;
                END IF;
        END CASE;
    END PROCESS;

    PROCESS (Clock, Resetn)
    BEGIN
        IF Resetn = '0' THEN
            y_present <= A;
        ELSIF (Clock'EVENT AND Clock = '1') THEN
            y_present <= y_next;
        END IF;
    END PROCESS;

    z <= '1' WHEN y_present = C ELSE '0';
END Behavior;
```

FSM
O Usuário Especificando a Atribuição de Estados

```
ARCHITECTURE Behavior OF simple IS

  TYPE State_TYPE IS (A, B, C);

  ATTRIBUTE ENUM_ENCODING : STRING;

  ATTRIBUTE ENUM_ENCODING OF State_type : TYPE IS "00 01 11";

  SIGNAL y_present, y_next : State_type;
 BEGIN

  con't ...
```

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY simple IS
	PORT ( Clock, Resetn, w : IN    STD_LOGIC;
	       z                : OUT   STD_LOGIC );
END simple;

ARCHITECTURE Behavior OF simple IS
	SIGNAL y_present, y_next : STD_LOGIC_VECTOR(1 DOWNTO 0);
	CONSTANT A  : STD_LOGIC_VECTOR(1 DOWNTO 0) := "00";
	CONSTANT B  : STD_LOGIC_VECTOR(1 DOWNTO 0) := "01";
	CONSTANT C  : STD_LOGIC_VECTOR(1 DOWNTO 0) := "11";
BEGIN
	PROCESS ( w, y_present )
	BEGIN
		CASE y_present IS
			WHEN A =>
				IF w = '0' THEN y_next <= A;
				ELSE y_next <= B;
				END IF;

… con't
```

```
            WHEN B =>
                IF w = '0' THEN y_next <= A;
                ELSE y_next <= C;
                END IF;
            WHEN C =>
                IF w = '0' THEN y_next <= A;
                ELSE y_next <= C;
                END IF;
            WHEN OTHERS =>
                y_next <= A;
        END CASE;
    END PROCESS;

    PROCESS ( Clock, Resetn )
    BEGIN
        IF Resetn = '0' THEN
            y_present <= A;
        ELSIF (Clock'EVENT AND Clock = '1') THEN
            y_present <= y_next;
        END IF;
    END PROCESS;
    z <= '1' WHEN y_present = C ELSE '0';
END Behavior;
```

FSM de Mealy

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY mealy IS
        PORT (Clock, Resetn, w : IN     STD_LOGIC;
                 z               : OUT  STD_LOGIC );
END mealy;

ARCHITECTURE Behavior OF mealy IS
        TYPE State_type IS (A, B);
        SIGNAL y : State_type;
BEGIN
        PROCESS ( Resetn, Clock )
        BEGIN
                IF Resetn = '0' THEN
                        y <= A;
                ELSIF (Clock'EVENT AND Clock = '1') THEN
                        CASE y IS
                                WHEN A =>
                                        IF w = '0' THEN y <= A;
                                        ELSE y <= B;
                                        END IF;
… con't
```

FSM de Mealy

```
                        WHEN B =>
                                IF w = '0' THEN y <= A;
                                ELSE y <= B;
                                END IF;
                END CASE;
        END IF;
    END PROCESS;

    PROCESS ( y, w )
    BEGIN
        CASE y IS
            WHEN A =>
                z <= '0';
            WHEN B =>
                z <= w;
        END CASE;
    END PROCESS;
END Behavior;
```