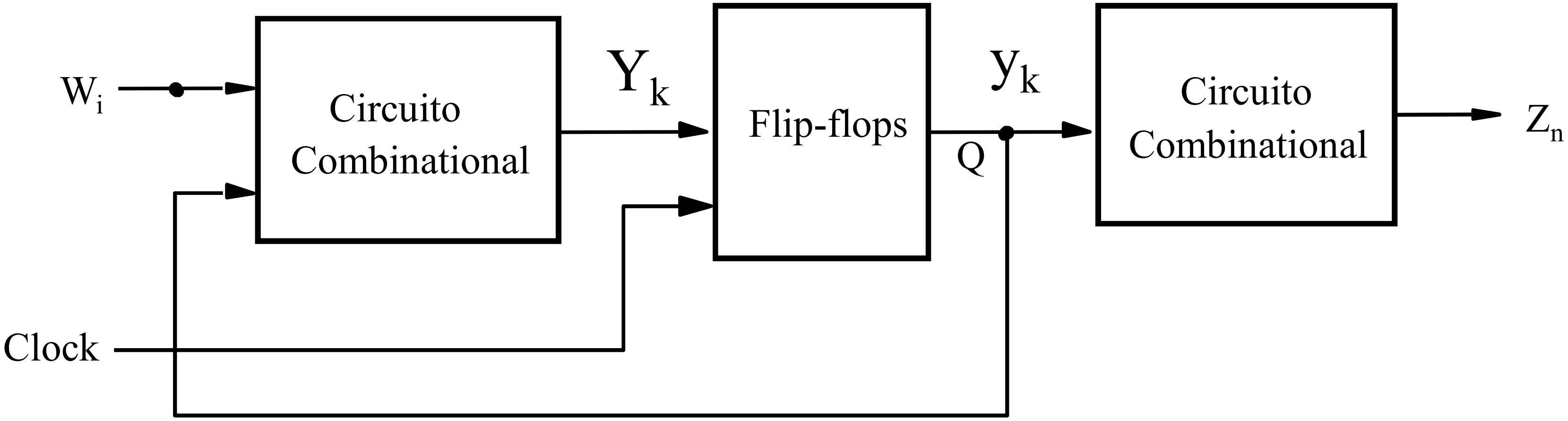
Circuitos Lógicos e Organização de Computadores

Capítulo 8 –Circuitos Sequenciais Síncronos

Ricardo Pannain

pannain@unicamp.br

Forma geral de um circuito sequencial - MOORE



Máquina de estados finitos (FSM – Finite State Machine) → o comportamento do circuito pode ser representado usando um número finito de estados.

Máquina de MOORE → saídas dependem apenas do estado do circuito.

Etapas Básicas de Projeto

Exemplo:

Supondo um circuito que tenha uma entrada *w* e uma saída *z*.

- Todas as mudanças ocorrerão na subida do clock.
- A saída *z* é igual a 1 se durante dois ciclos consecutivos imediatamente precedente a entrada *w* for igual a 1. Caso contrário o valor é zero.

A tabela abaixo ilustra o funcionamento para um padrão de w qualquer.

Clockcycle:	t_0	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}
w:	0	1	0	1	1	0	1	1	1	0	1
z:	0	0	0	0	0	1	0	0	1	1	0

Etapas Básicas de Projeto

PRIMEIRO PASSO → Diagrama de estados

Determinar quantos estados são necessários e quais transições são possíveis de um estado para outro.

No exemplo:

Supondo o estado inicial A (com saída z = 0) → enquanto w = 0 ele permanece neste estado.
Se w = 1, na subida do clock ele passará para o estado B (com z = 0).
Uma vez no estado B, se w = 0 , ele passará para o estado A (z = 0) na subida do clock.
Se w = 1, passará para um terceiro estado C (z = 1).
No estado C, se w = 0, ele passará para o estado A (z = 0) na subida do clock. Se w = 1,
Continuará no estado C (z = 1)

Diagrama de estados do exemplo anterior – Máquina de MOORE

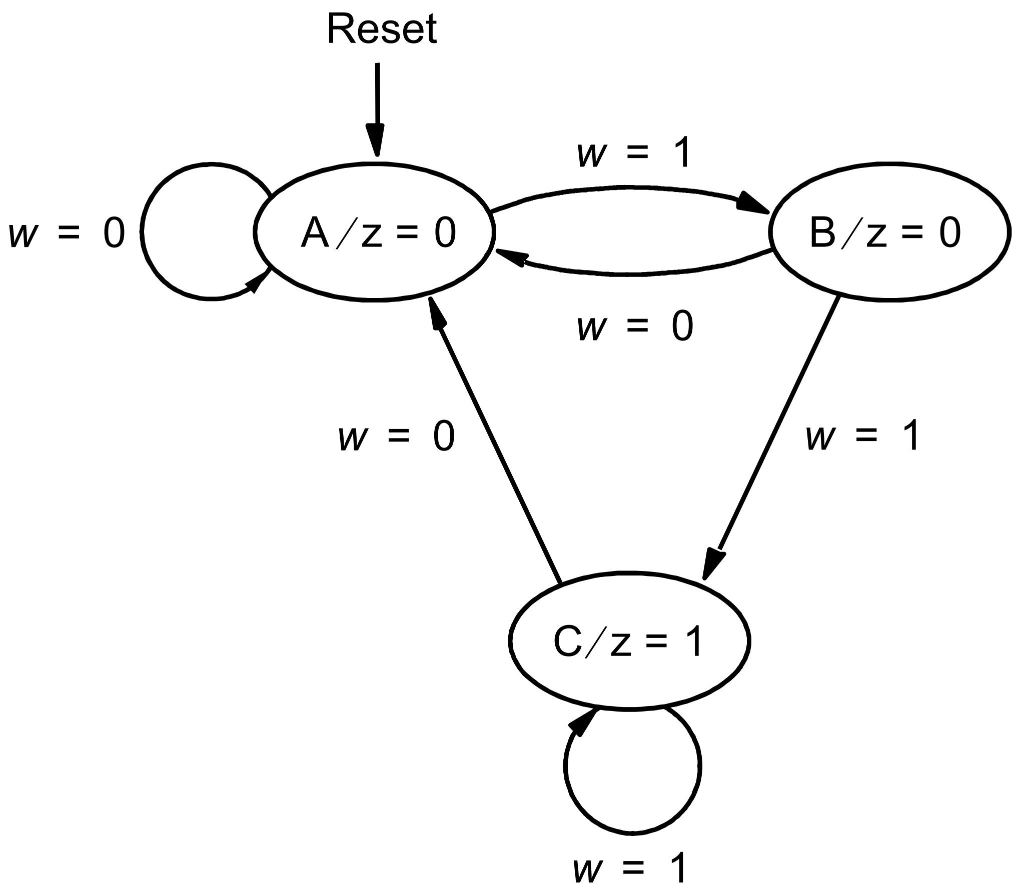
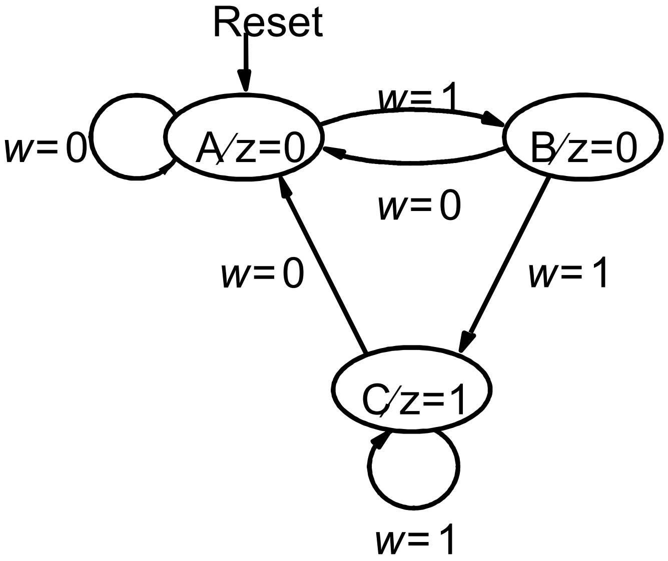


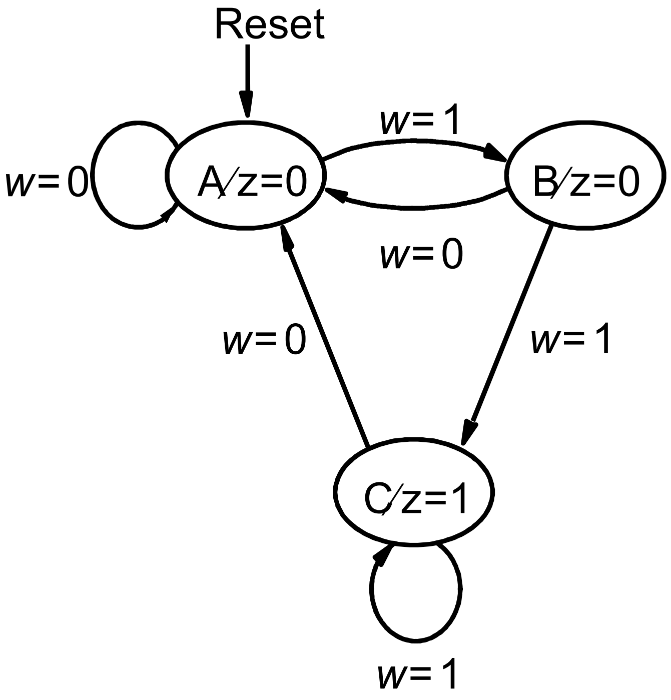
Diagrama de estados do exemplo anterior – Máquina de MOORE



Clockcycle:	t_0	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}
w:	0	1	0	1	1	0	1	1	1	0	1
z:	0	0	0	0	0	1	0	0	1	1	0

Tabela de estados

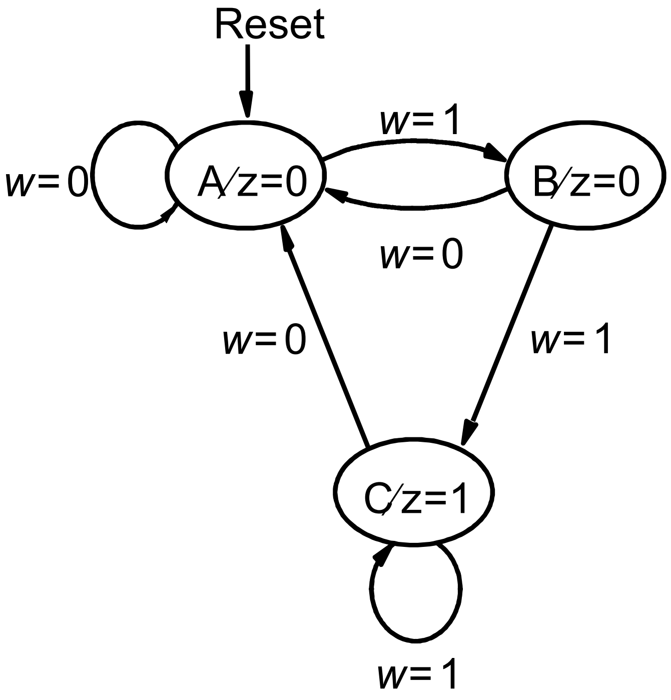
SEGUNDO PASSO → Tabela de Estados



Present state	Next state		Output
	$w = 0$	$w = 1$	z

Tabela de estados

SEGUNDO PASSO → Tabela de Estados



Present state	Next state		Output
	$w = 0$	$w = 1$	z
A	A	B	0
B	A	C	0
C	A	C	1

Circuito seqüencial geral

Como temos 3 estados, precisaremos de 2 Flip-Flops e o circuito ficaria:

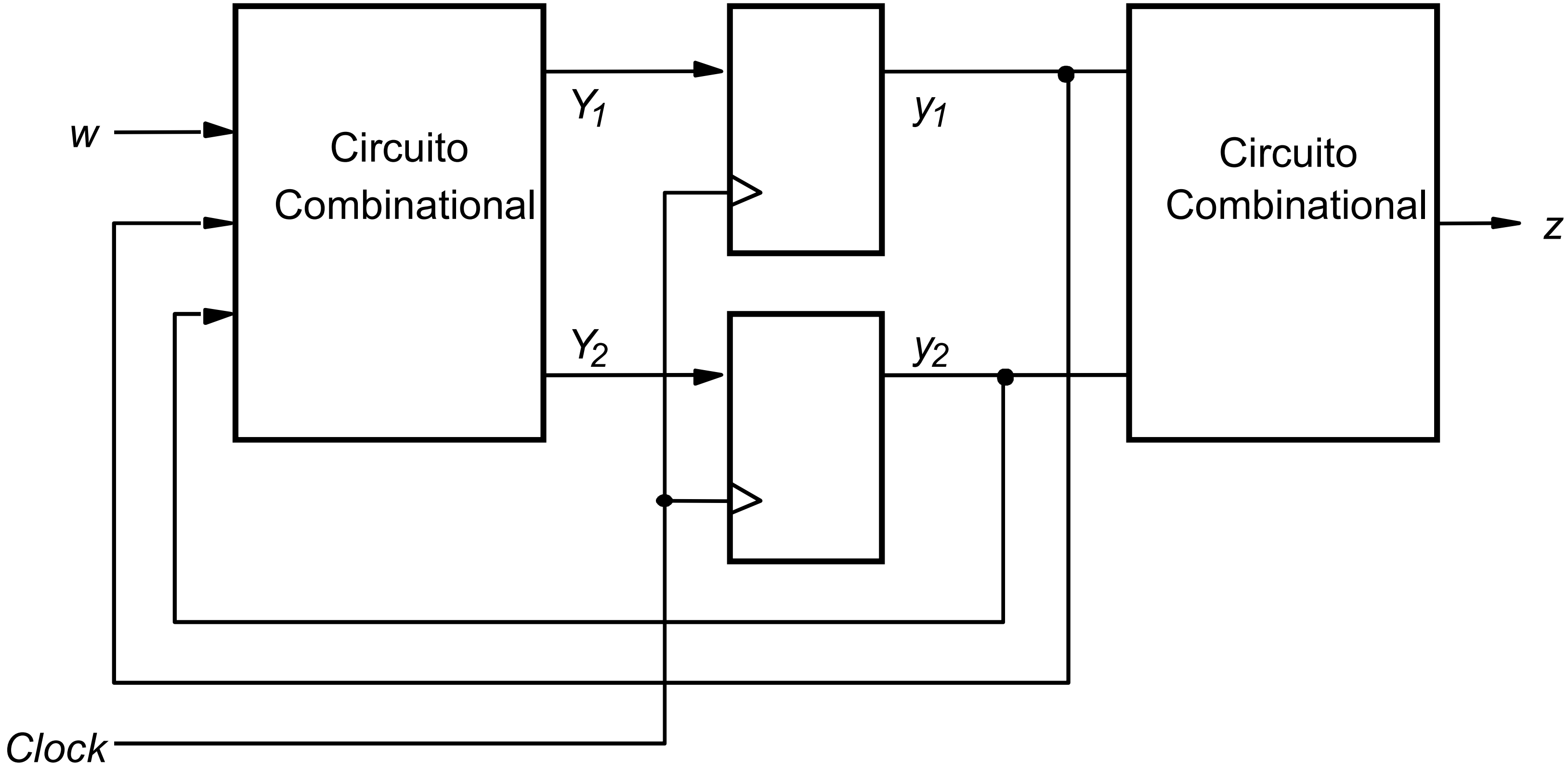


Tabela de estados

Present state	Next state		Output z
	$w = 0$	$w = 1$	
A	A	B	0
B	A	C	0
C	A	C	1

Tabela de estados assinalados

	Present state	Next state		Output
		$w = 0$	$w = 1$	z
	$y_2 y_1$	$Y_2 Y_1$	$Y_2 Y_1$	
A	00	00	01	0
B	01	00	10	0
C	10	00	10	1
	11	*dd*	*dd*	*d*

Tabela de estados assinalados

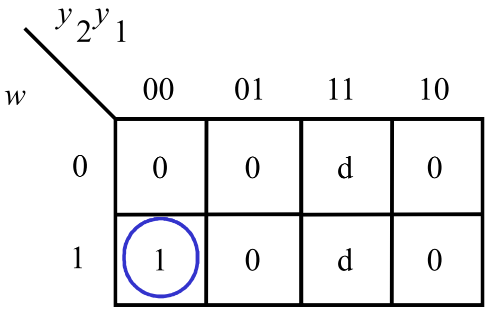
	Present state	Next state		Output
		$w = 0$	$w = 1$	z
	y_2y_1	Y_2Y_1	Y_2Y_1	
A	00	00	01	0
B	01	00	10	0
C	10	00	10	1
	11	*dd*	*dd*	*d*

w	y2	y1	Y2	Y1
0	0	0	0	0
0	0	1	0	0
0	1	0	0	0
0	1	1	d	d
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	d	d

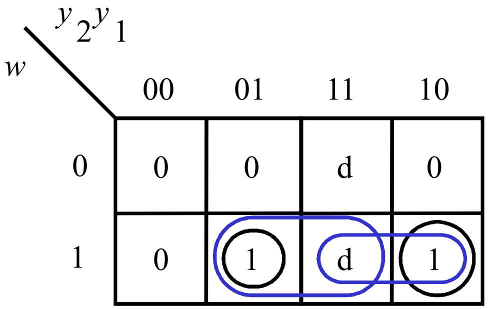
y2	y1	z
0	0	0
0	1	0
1	0	1
1	1	d

Tabelas Verdade

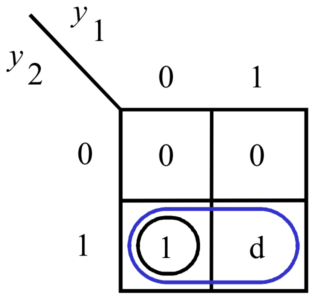
Derivando as expressões lógicas dos próximos estados e da saída



w \ y_2y_1	00	01	11	10
0	0	0	d	0
1	1	0	d	0



w \ y_2y_1	00	01	11	10
0	0	0	d	0
1	0	1	d	1



y_2 \ y_1	0	1
0	0	0
1	1	d

Ignorando don't cares	Usando don't cares
$Y_1 = w\bar{y}_1\bar{y}_2$	$Y_1 = w\bar{y}_1\bar{y}_2$
$Y_2 = wy_1\bar{y}_2 + w\bar{y}_1y_2$	$Y_2 = wy_1 + wy_2$ $= w(y_1 + y_2)$
$z = \bar{y}_1y_2$	$z = y_2$

Circuito Seqüencial do exemplo anterior

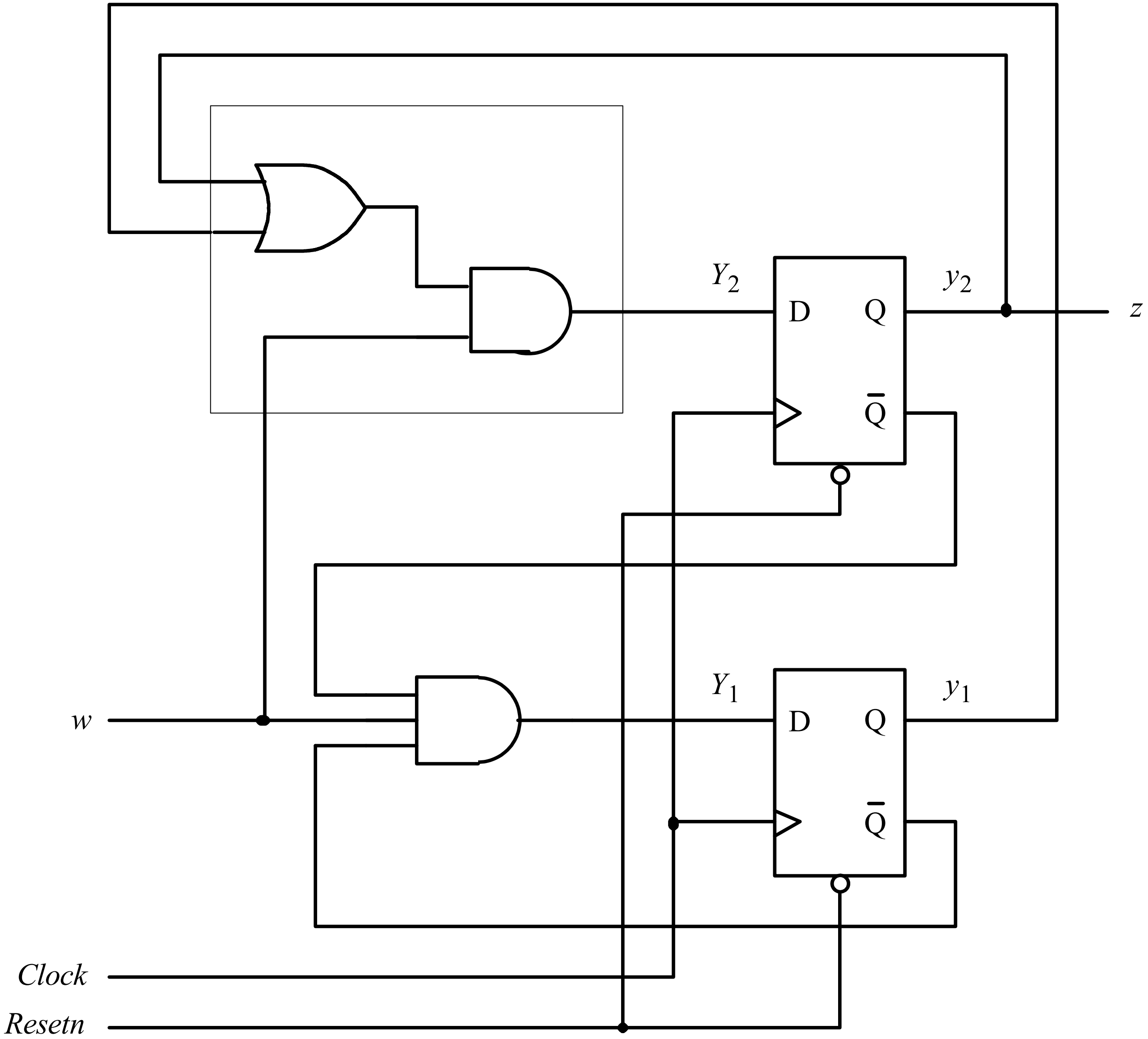
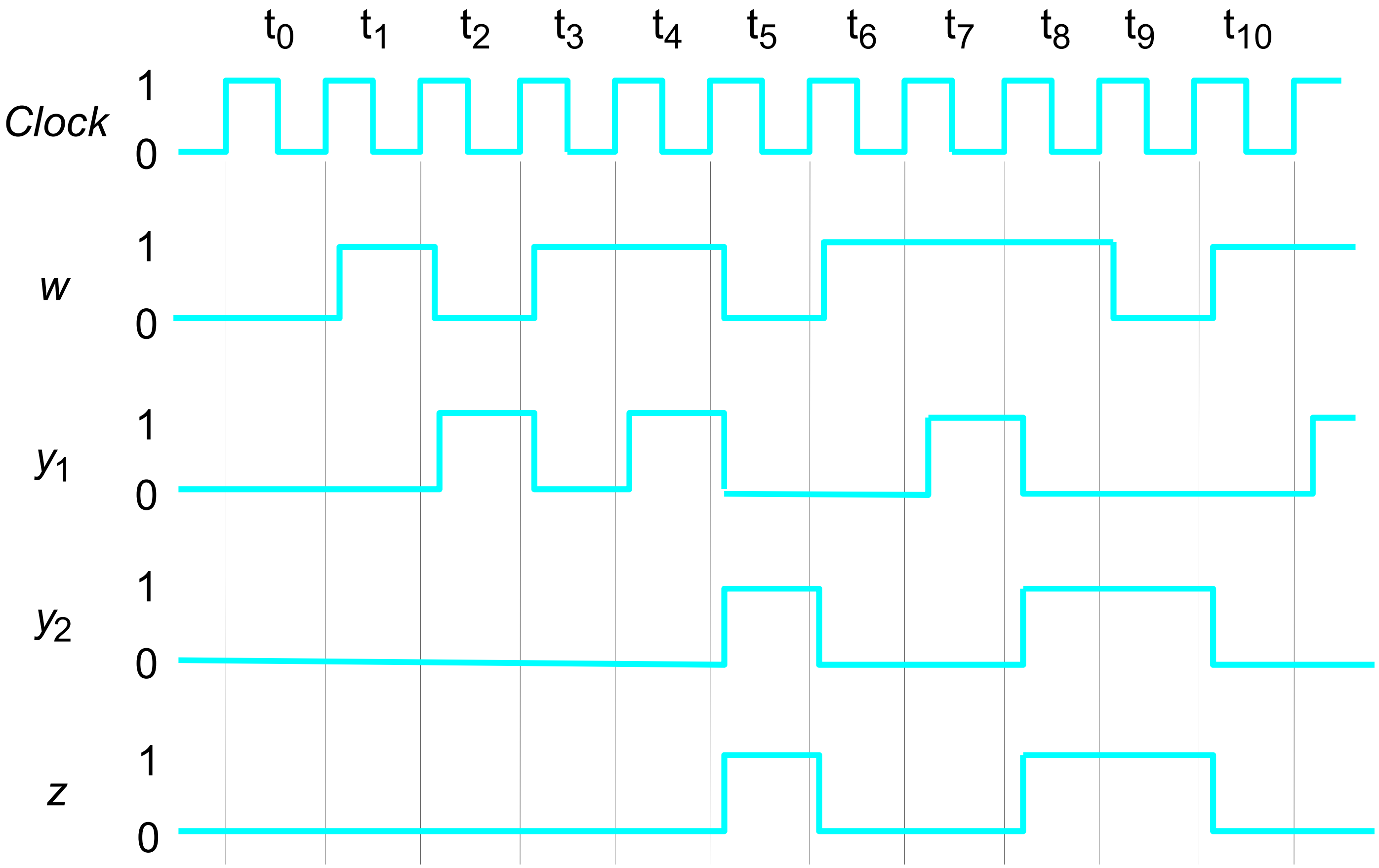
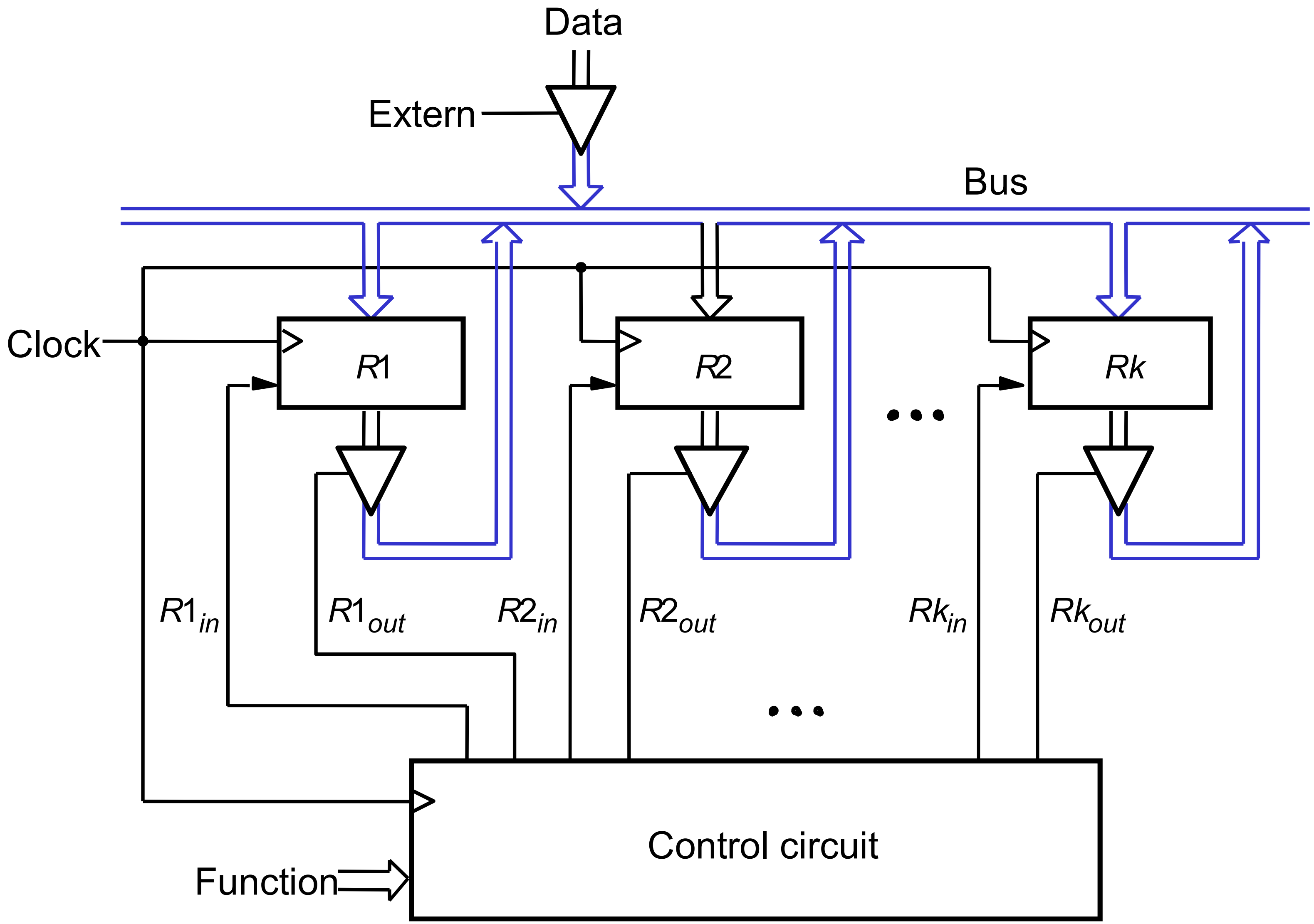


Diagrama de tempo do exemplo anterior



Clockcycle:	t_0	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}
w:	0	1	0	1	1	0	1	1	1	0	1
z:	0	0	0	0	0	1	0	0	1	1	0

Sistema digital com *k* registradores



Exemplo – Troca de conteúdo entre dois registradores MOORE

Trocar o conteúdo dos registradores R1 e R2, utilizando o registador R3 com auxiliar.
A troca se inicia quando w=1. A operação só termina quando a troca for feita, independente de w (gera done).

R3 ← R2
R2 ← R1
R1 ← R3

Sinais necessários

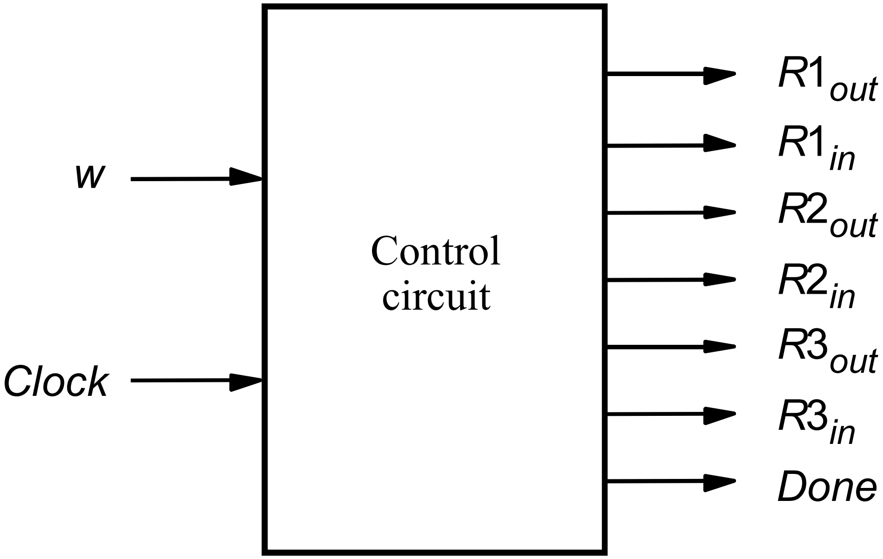
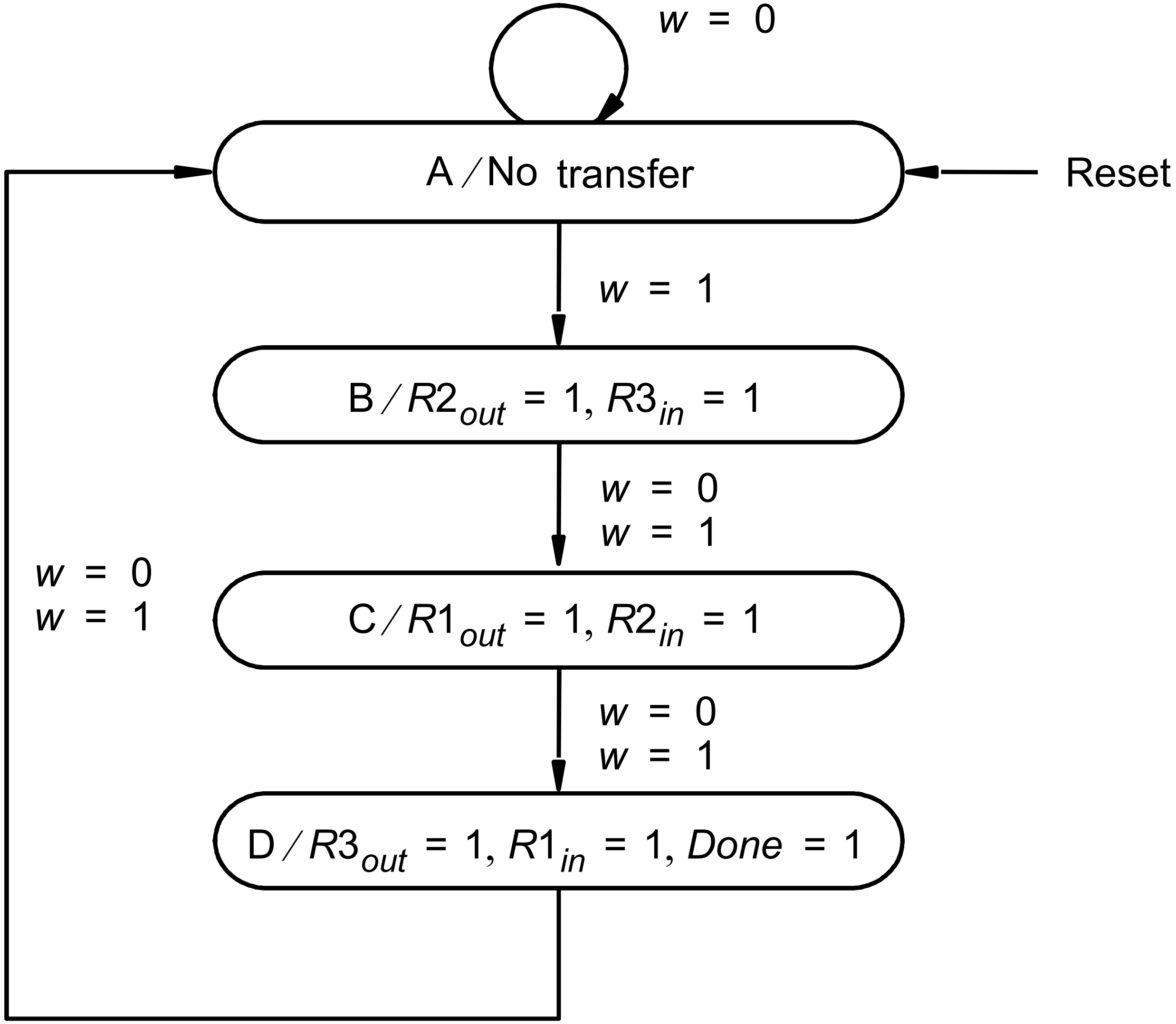
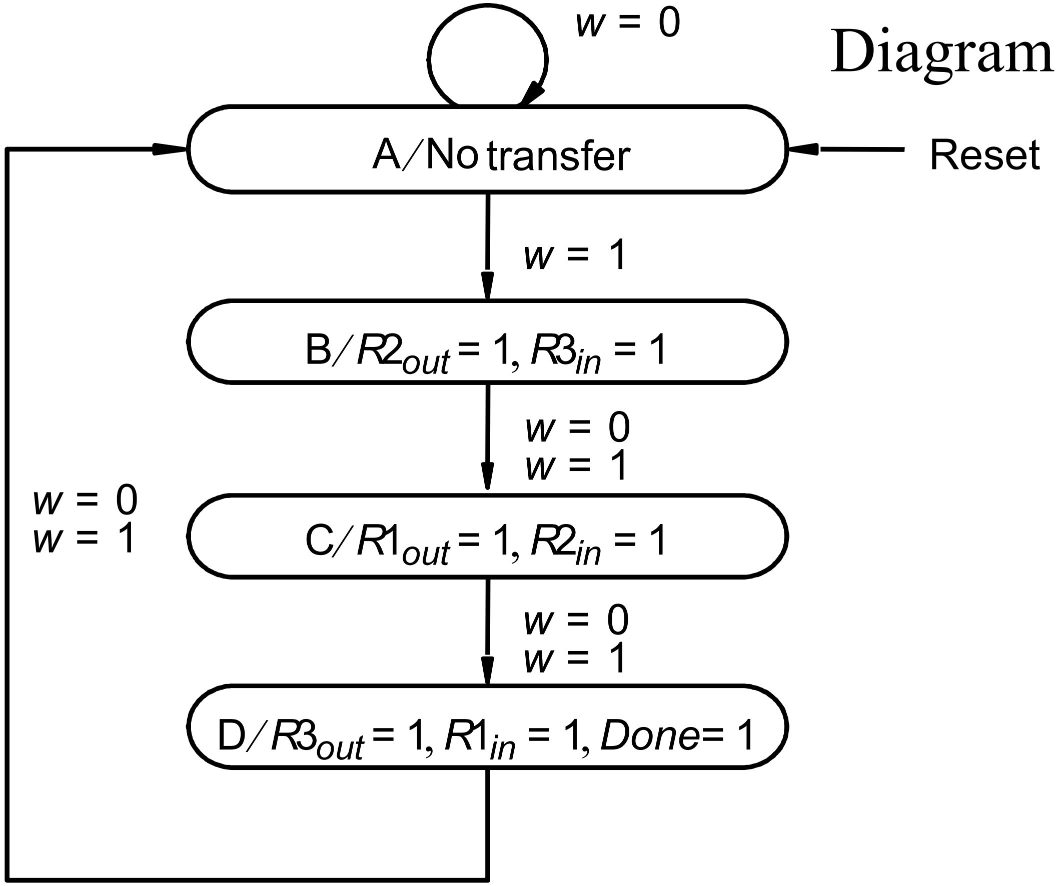


Diagrama de estados



$w = 0$
A / No transfer
Reset
$w = 1$
B / $R2_{out} = 1$, $R3_{in} = 1$
$w = 0$
$w = 1$
$w = 0$
$w = 1$
C / $R1_{out} = 1$, $R2_{in} = 1$
$w = 0$
$w = 1$
D / $R3_{out} = 1$, $R1_{in} = 1$, $Done = 1$

Diagrama de estados


w = 0
A/No transfer
Reset
w = 1
B/$R2_{out}$ = 1, $R3_{in}$ = 1
w = 0
w = 1
C/$R1_{out}$ = 1, $R2_{in}$ = 1
w = 0
w = 1
D/$R3_{out}$ = 1, $R1_{in}$ = 1, Done = 1
w = 0
w = 1

Present state	Next state		Outputs						
	$w = 0$	$w = 1$	$R1_{out}$	$R1_{in}$	$R2_{out}$	$R2_{in}$	$R3_{out}$	$R3_{in}$	$Done$
A	A	B	0	0	0	0	0	0	0
B	C	C	0	0	1	0	0	1	0
C	D	D	1	0	0	1	0	0	0
D	A	A	0	1	0	0	1	0	1

Tabela de estados

Present state	Next state		Outputs						
	w = 0	*w* = 1	$R1_{out}$	$R1_{in}$	$R2_{out}$	$R2_{in}$	$R3_{out}$	$R3_{in}$	$Done$
A	A	B	0	0	0	0	0	0	0
B	C	C	0	0	1	0	0	1	0
C	D	D	1	0	0	1	0	0	0
D	A	A	0	1	0	0	1	0	1

Tabela de estados assinalados

	Present state	Nextstate		Outputs						
		$w = 0$	$w = 1$							
	y_2y_1	Y_2Y_1	Y_2Y_1	$R1_{out}$	$R1_{in}$	$R2_{out}$	$R2_{in}$	$R3_{out}$	$R3_{in}$	$Done$
A	00	00	0 1	0	0	0	0	0	0	0
B	01	10	1 0	0	0	1	0	0	1	0
C	10	11	1 1	1	0	0	1	0	0	0
D	11	00	0 0	0	1	0	0	1	0	1

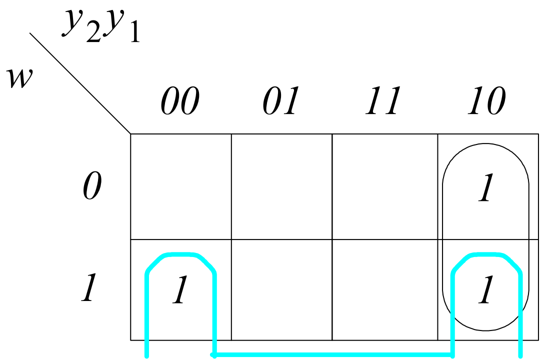
Tabela de estados assinalados

	Present state	Nextstate		Outputs						
		$w = 0$	$w = 1$							
	y_2y_1	Y_2Y_1	Y_2Y_1	$R1_{out}$	$R1_{in}$	$R2_{out}$	$R2_{in}$	$R3_{out}$	$R3_{in}$	$Done$
A	00	00	0 1	0	0	0	0	0	0	0
B	01	10	1 0	0	0	1	0	0	1	0
C	10	11	1 1	1	0	0	1	0	0	0
D	11	00	0 0	0	1	0	0	1	0	1

w	y2	y1	Y2	Y1
0	0	0	0	0
0	0	1	1	0
0	1	0	1	1
0	1	1	0	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	1
1	1	1	0	0

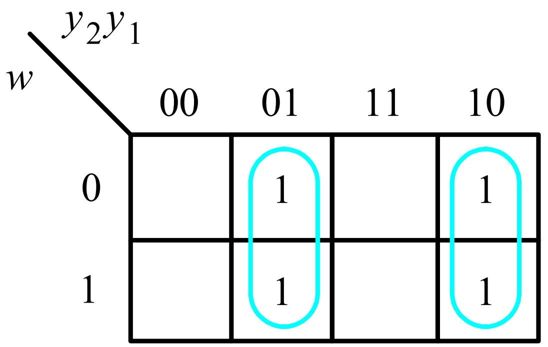
Derivação da expressão do próximo estado

w	y2	y1	Y2	Y1
0	0	0	0	0
0	0	1	1	0
0	1	0	1	1
0	1	1	0	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	1
1	1	1	0	0



$$\begin{array}{c|cccc} w \backslash y_2y_1 & 00 & 01 & 11 & 10 \\ \hline 0 & & & & 1 \\ 1 & 1 & & & 1 \end{array}$$

$$Y_1 = w\bar{y}_1 + \bar{y}_1 y_2$$



$$\begin{array}{c|cccc} w \backslash y_2y_1 & 00 & 01 & 11 & 10 \\ \hline 0 & & 1 & & 1 \\ 1 & & 1 & & 1 \end{array}$$

$$Y_2 = y_1\bar{y}_2 + \bar{y}_1 y_2$$

Tabela de estados assinalados

	Present state	Nextstate		Outputs						
		$w = 0$	$w = 1$							
	$y_2 y_1$	$Y_2 Y_1$	$Y_2 Y_1$	$R1_{out}$	$R1_{in}$	$R2_{out}$	$R2_{in}$	$R3_{out}$	$R3_{in}$	$Done$
A	00	00	0 1	0	0	0	0	0	0	0
B	01	10	1 0	0	0	1	0	0	1	0
C	10	11	1 1	1	0	0	1	0	0	0
D	11	00	0 0	0	1	0	0	1	0	1

y2	y1	R1 out	R1 in	R2 out	R2 in	R3 out	R3 in	Done
0	0	0	0	0	0	0	0	0
0	1	0	0	1	0	0	1	0
1	0	1	0	0	1	0	0	0
1	1	0	1	0	0	1	0	1

Derivação das expressões das saídas

y2 **y1**	**0**	**1**
0	0	1
1	0	0

R2out,R3in =y’1y2

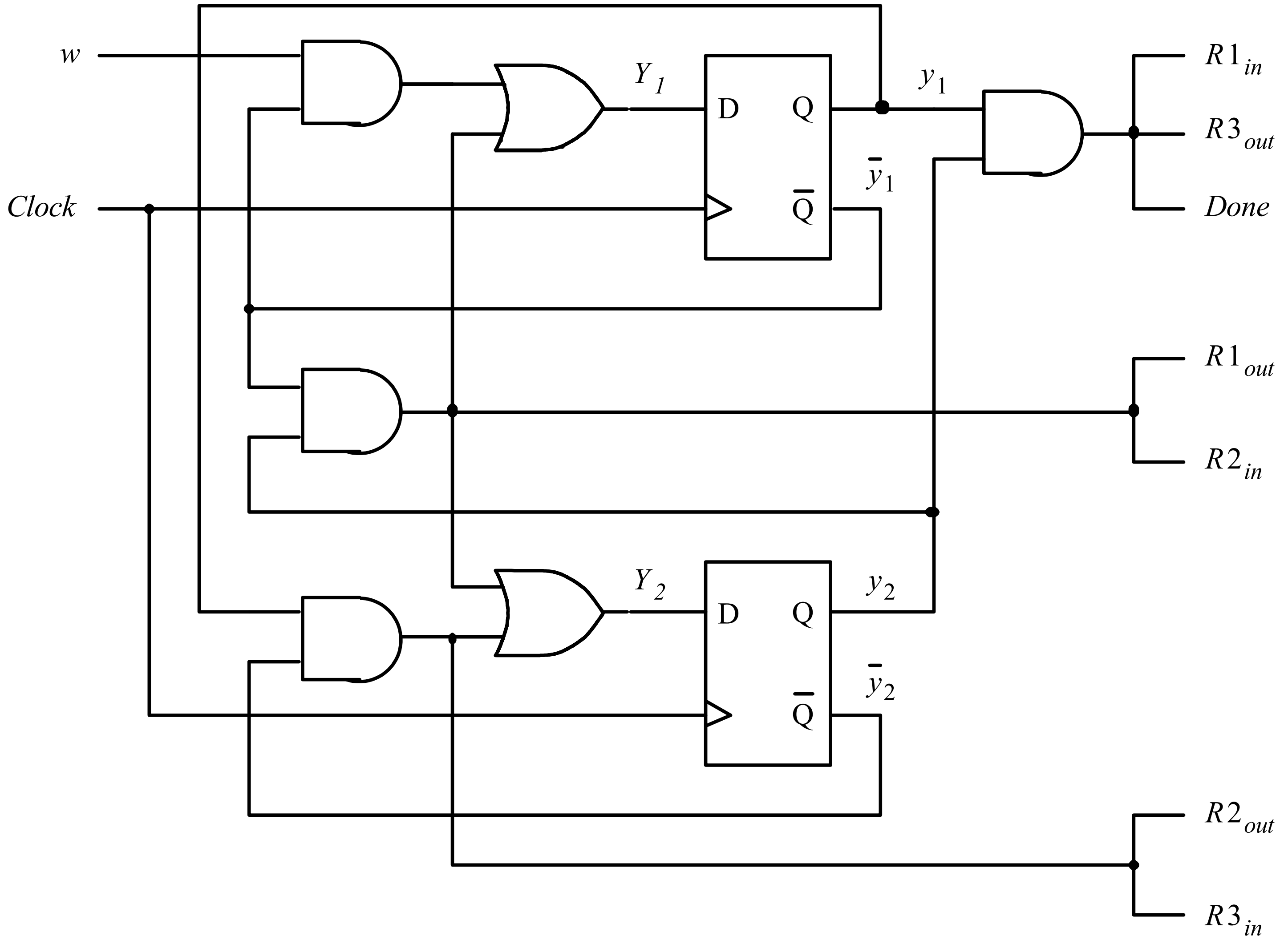
y2 **y1**	**0**	**1**
0	0	0
1	1	0

R1out,R2in = y1y’2

y2 **y1**	**0**	**1**
0	0	0
1	0	1

R3out,R1in, Done = y1y2

Circuito sequencial correspondente



Circuito seqüencial correspondente

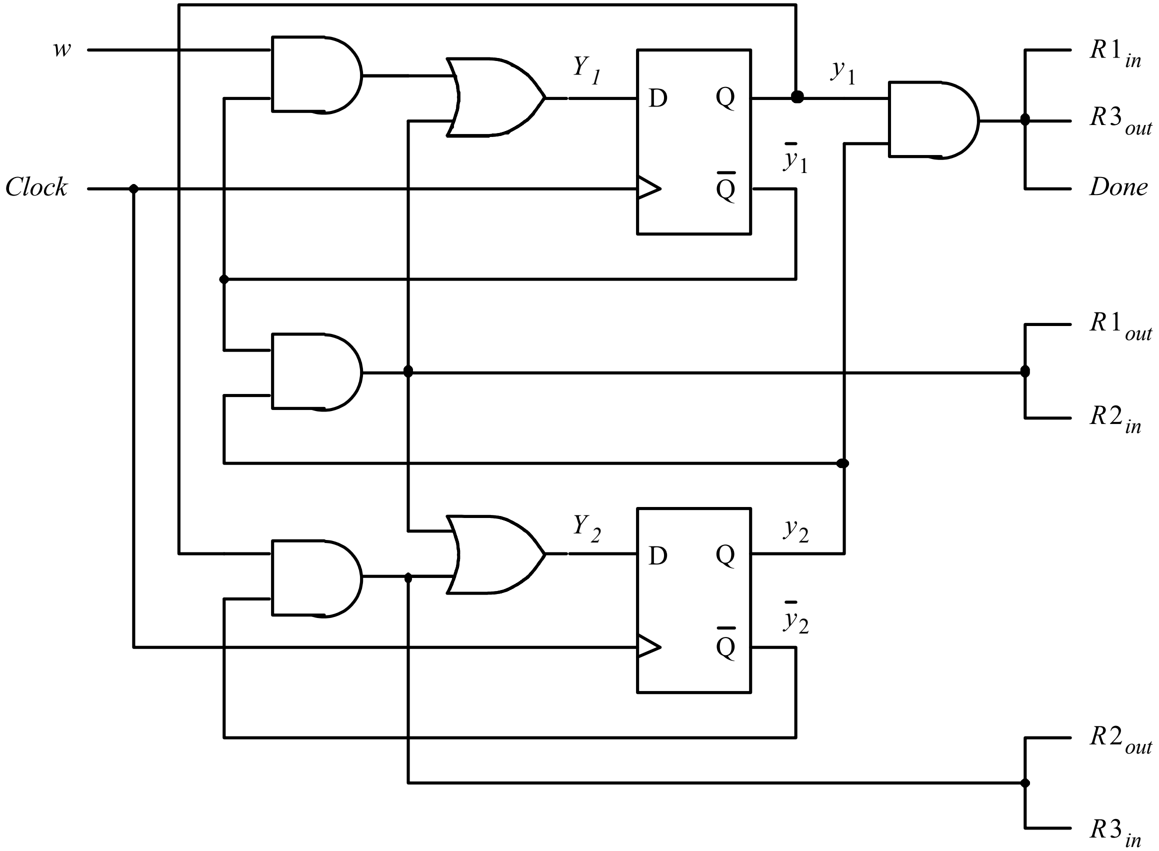
R2out,R3in =y’1y2

R1out,R2in = y1y’2

R3out,R1in, Done = y1y2

Y1=wy1’+y1’y2

Y2 = y1y2’+y1’y2



Problemas de assinalamento de estados

Suponha que, no primeiro exemplo, ao assinalar valores para os estados presentes, tenhamos escolhido conforma a tabela abaixo:

	Present state	Next state		Output
		$w = 0$	$w = 1$	
	y_2y_1	Y_2Y_1	Y_2Y_1	z
A	00	00	01	0
B	01	00	11	0
C	11	00	11	1
	10	*dd*	*dd*	*d*

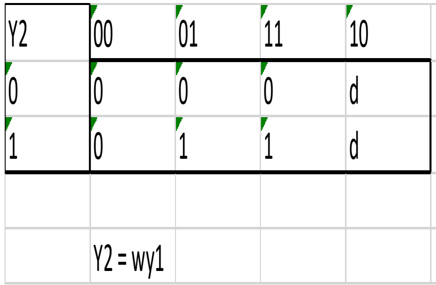
Problemas de assinalamento de estados

Suponha que, no primeiro exemplo, ao assinalar valores para os estados presentes, tenhamos escolhido conforma a tabela abaixo:

$Y_1 = D_1 = w$
$Y_2 = D_2 = w\ y_1$
$z = y_2$

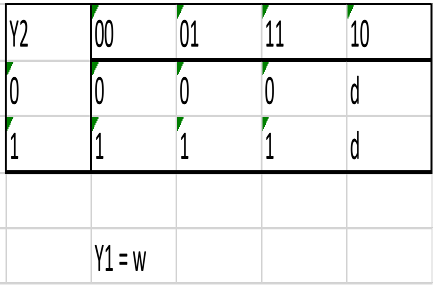
w	y2	y1	Y2	Y1
0	0	0	0	0
0	0	1	0	0
0	1	0	d	d
0	1	1	0	0
1	0	0	0	1
1	0	1	1	1
1	1	0	d	d
1	1	1	1	1

	Present state	Next state		Output
		$w = 0$	$w = 1$	
	$y_2 y_1$	$Y_2 Y_1$	$Y_2 Y_1$	z
A	00	00	01	0
B	01	00	11	0
C	11	00	11	1
	10	*dd*	*dd*	*d*



Y2	00	01	11	10
0	0	0	0	d
1	0	1	1	d

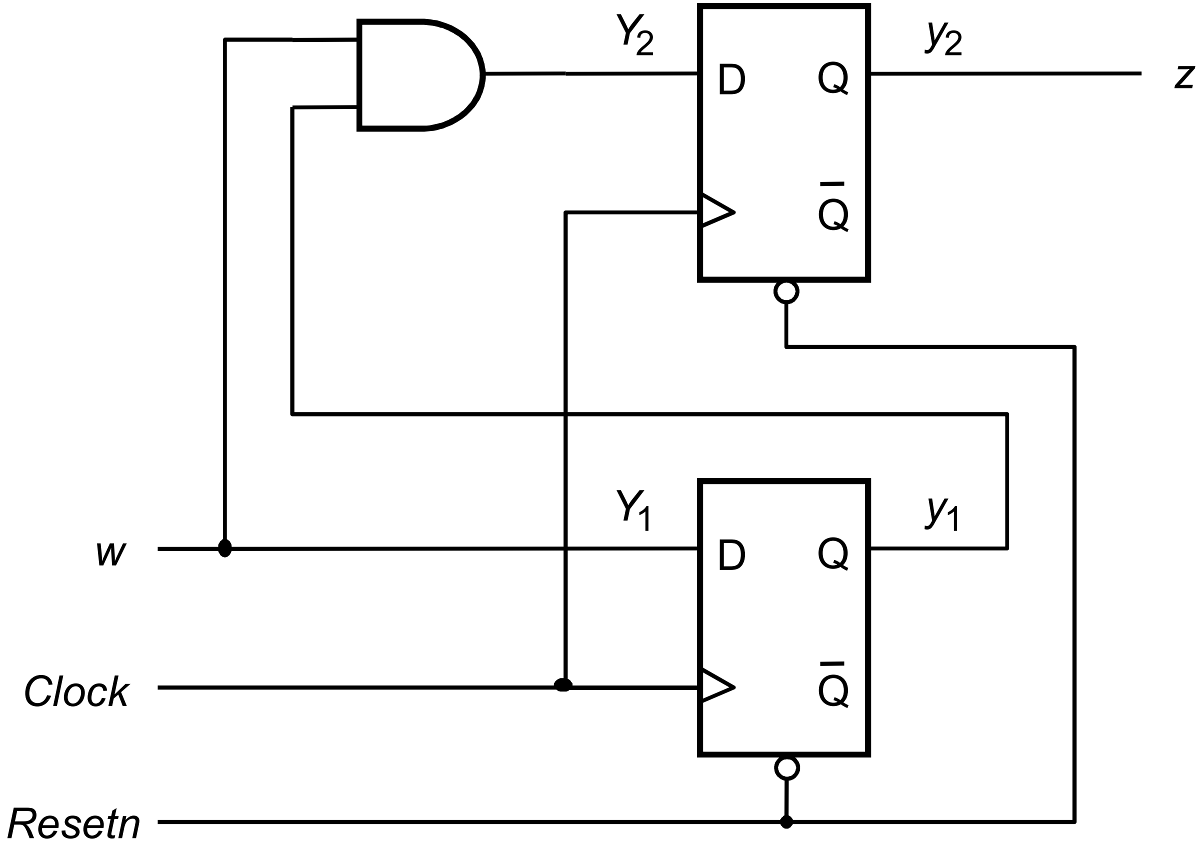
Y2 = wy1



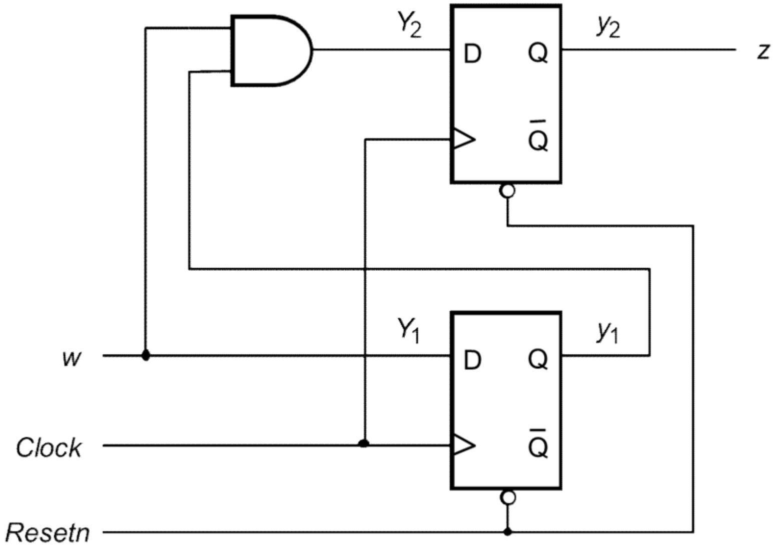
Y2	00	01	11	10
0	0	0	0	d
1	1	1	1	d

Y1 = w

Circuito com a melhoria no assinalamento



Circuito com a melhoria no assinalamento



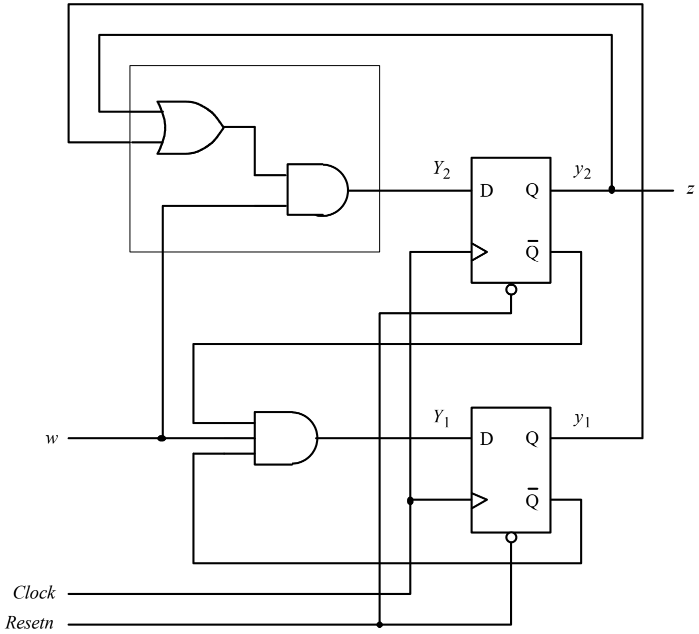
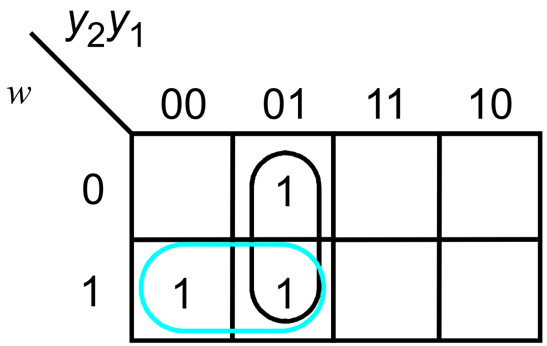


Tabela de estados com melhoria de assnalamento – exemplo 2

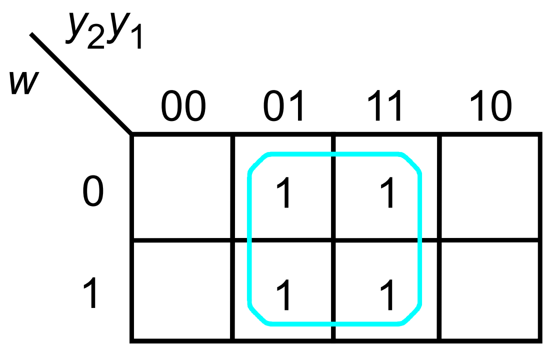
	Present state	Nextstate		Outputs						
		$w = 0$	$w = 1$							
	$y_2 y_1$	$Y_2 Y_1$	$Y_2 Y_1$	$R1_{out}$	$R1_{in}$	$R2_{out}$	$R2_{in}$	$R3_{out}$	$R3_{in}$	$Done$
A	00	0 0	01	0	0	0	0	0	0	0
B	01	1 1	11	0	0	1	0	0	1	0
C	11	1 0	10	1	0	0	1	0	0	0
D	10	0 0	00	0	1	0	0	1	0	1

Derivação da expressão do próximo estado



w \ y_2y_1	00	01	11	10
0		1		
1	1	1		

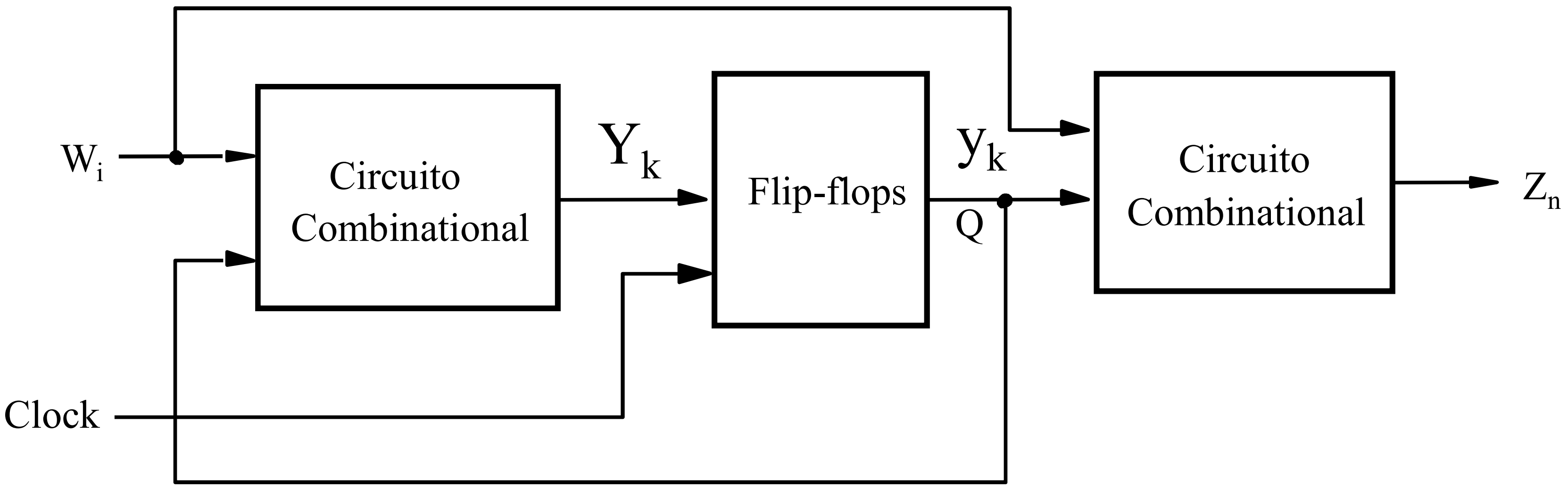
$$Y_1 = w\bar{y}_2 + y_1\bar{y}_2$$



w \ y_2y_1	00	01	11	10
0		1	1	
1		1	1	

$$Y_2 = y_1$$

Forma geral de um circuito sequencial - MEALY



Máquina de estados finitos (FSM – Finite State Machine) → o comportamento do circuito pode ser representado usando um número finito de estados.

Máquina de MEALY → saídas dependem do estado atual do circuito e das entradas.

Máquina de Mealy

Nos exemplos anteriores, os circuitos sequenciais onde cada estado tem valores dos sinais de saídas associados a eles, são chamadas **Máquinas de MOORE**.

Os circuitos sequenciais onde os sinais das saídas estão associados tanto aos estados como às entradas são chamadas de **Máquina de MEALY**

Tomando o primeiro exemplo visto anteriormente, que gerava z = 1 quando a ocorrência de w = 1 era detectada em dois períodos consecutivos de clock. Supondo agora que queiramos que z seja 1 no segundo ciclo que w = 1 seja detectado, como exemplifica a tabela abaixo:

Clock cycle:	t_0	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}
w:	0	1	0	1	1	0	1	1	1	0	1
z:	0	0	0	0	1	0	0	1	1	0	0

Diagrama de estado

No exemplo:

Supondo o estado inicial A → enquanto w = 0 ele permanece neste estado e produz a saída *z* = 0. Se w = 1 (*z* = 0), na subida do clock, ele passará para o estado B.
Uma vez no estado B, se w = 0 (*z* = 0) , ele passará para o estado A na subida do clock.
Se w = 1 produzirá a saída *z* = 1 e permanecerá neste estado..

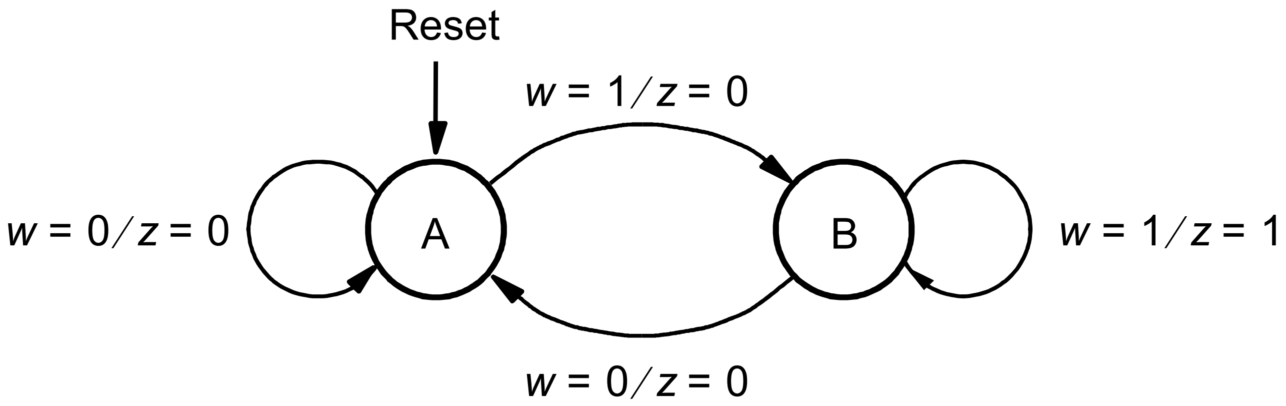
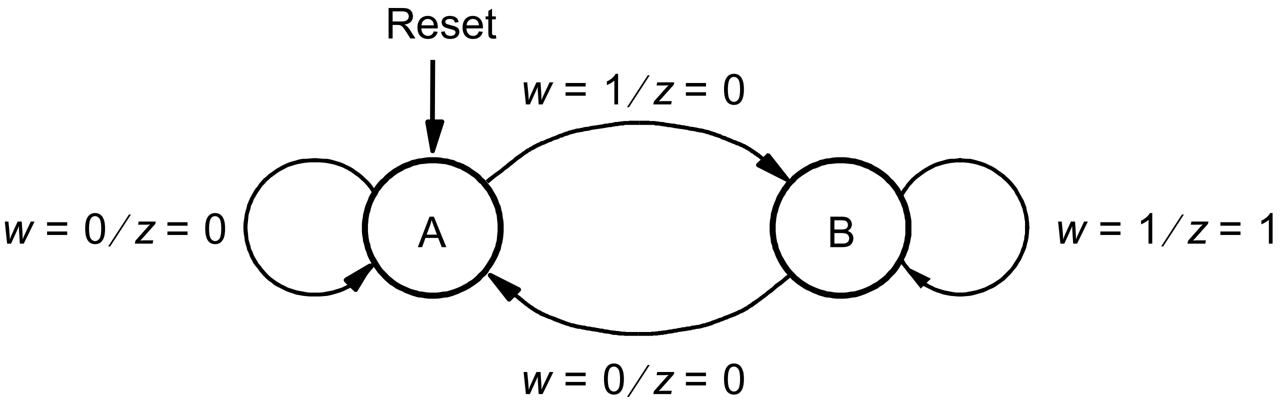


Tabela de estados



Present state	Next state		Output z	
	$w = 0$	$w = 1$	$w = 0$	$w = 1$
A	A	B	0	0
B	A	B	0	1

Tabela de estados assinalados

	Present state	Next state		Output	
		$w = 0$	$w = 1$	$w = 0$	$w = 1$
	y	Y	Y	z	z
A	0	0	1	0	0
B	1	0	1	0	1

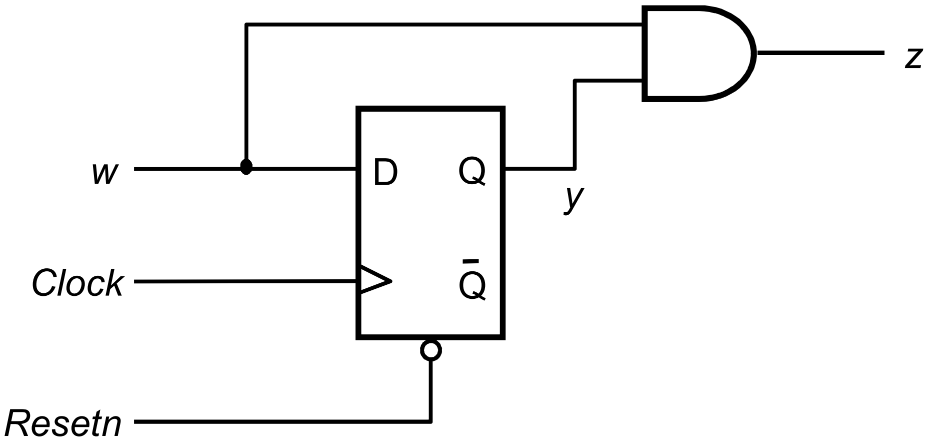
w	y	Y
0	0	0
0	1	0
1	0	1
1	1	1

w	y	z
0	0	0
0	1	0
1	0	0
1	1	1

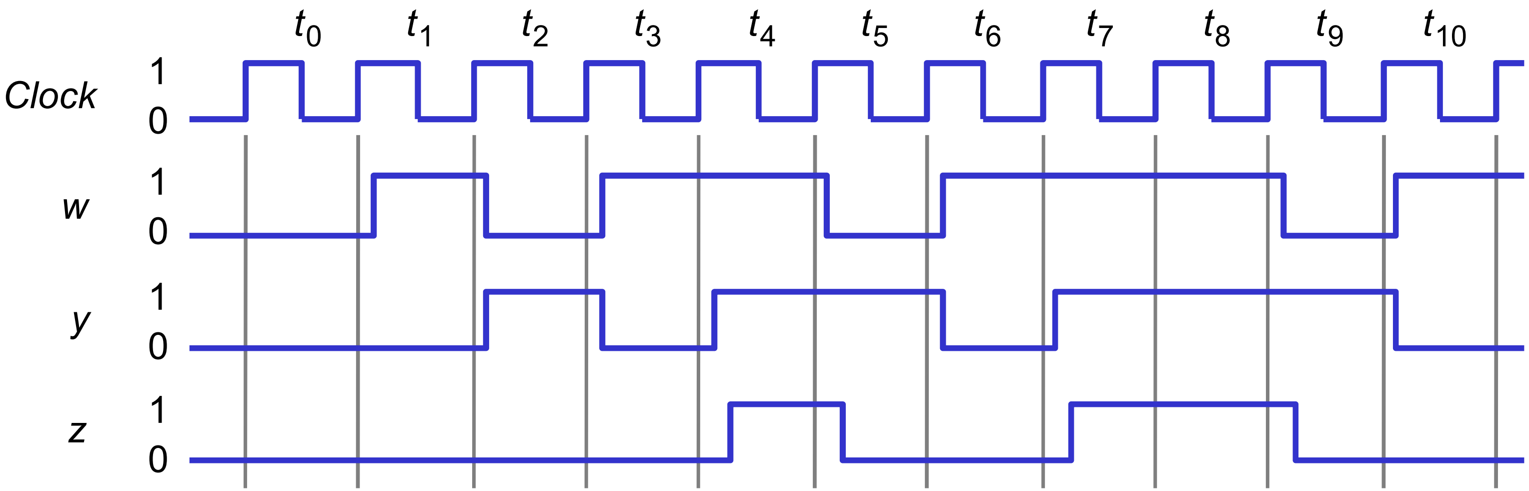
$Y = wy' + wy = w$

$z = wy$

Implementação da FSM

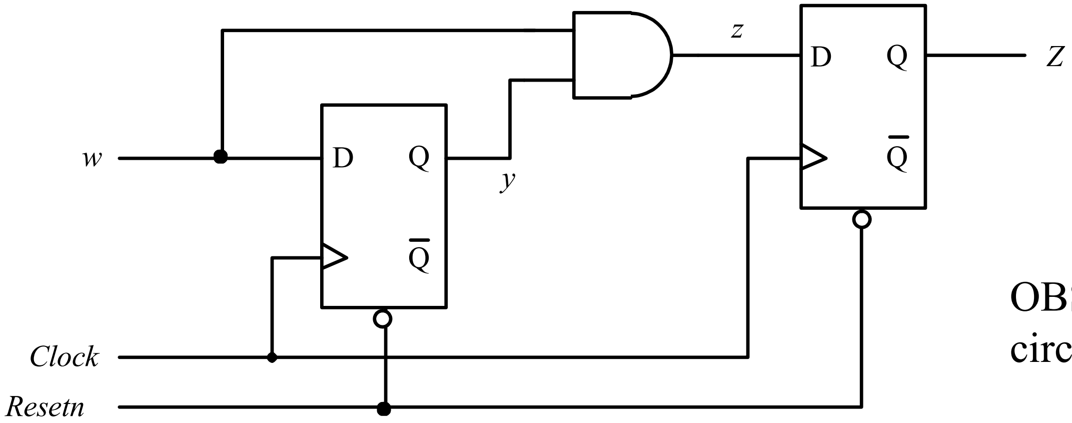


(a) Circuito



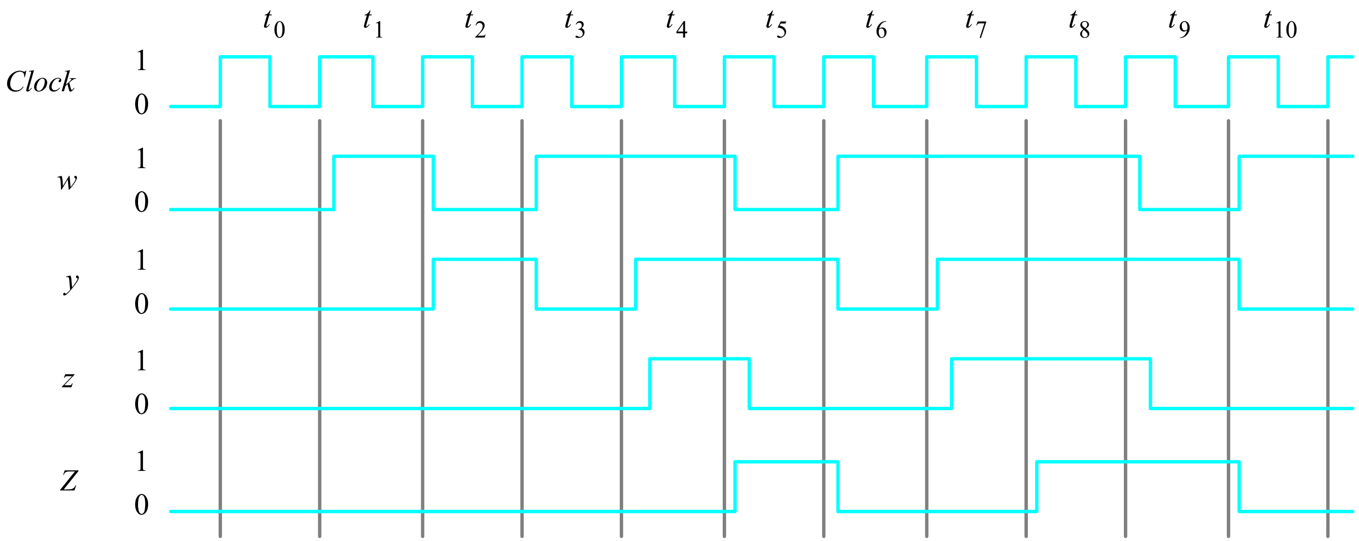
(b) Diagrama de tempo

Implementação da FSM incluíndo um atraso na saída



OBS – Mesmo circuito do slide 29

(a) Circuit



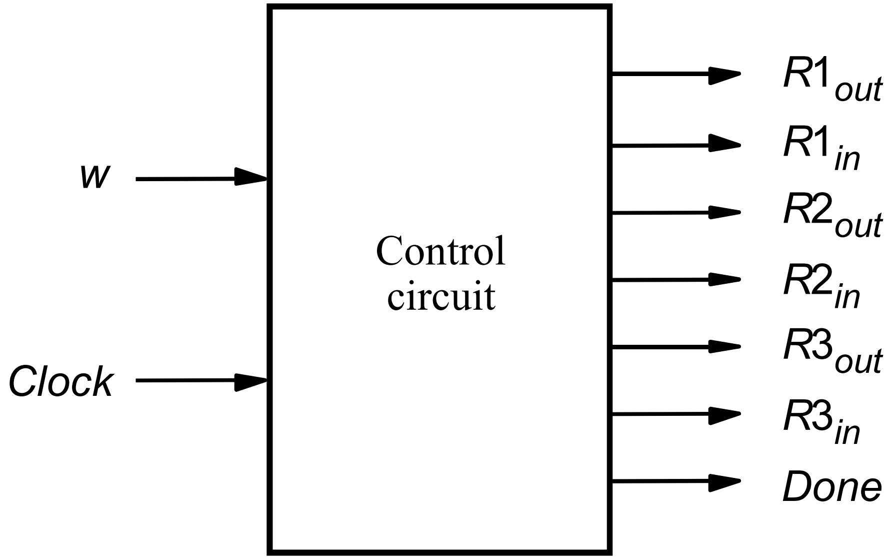
(b) Timing diagram

Exemplo – Troca de conteúdo entre dois registradores - Mealy

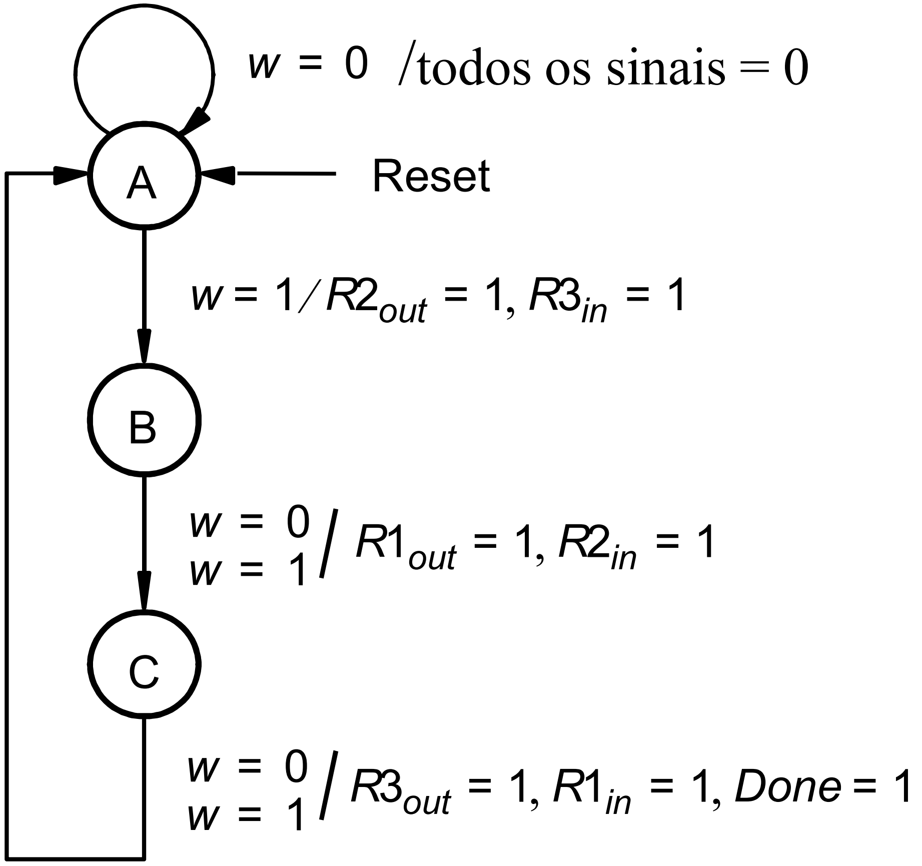
Trocar o conteúdo dos registradores R1 e R2, utilizando o registador R3 com auxiliar

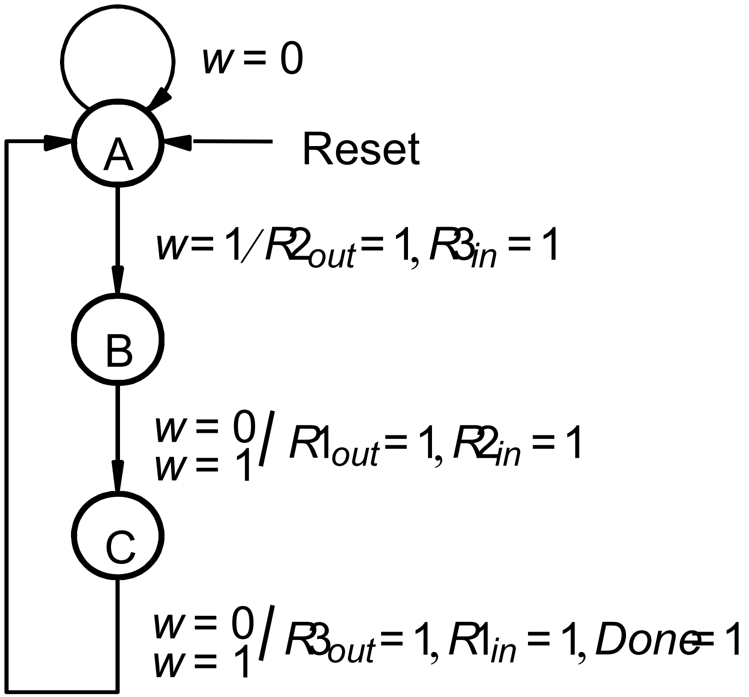
R3 ← R2
R2 ← R1
R1 ← R3

Sinais necessários

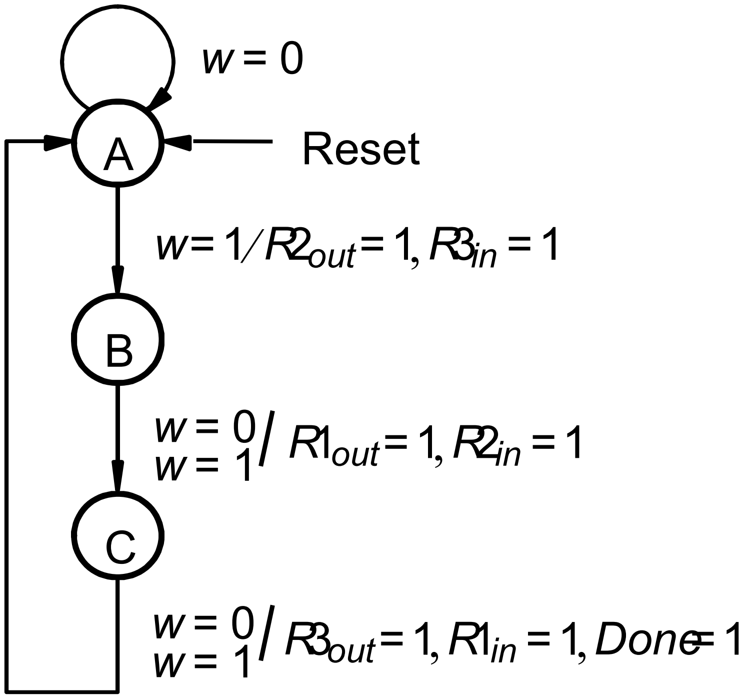


Exemplo – troca de conteúdo de registradores


$w = 0$ /todos os sinais = 0
Reset
A
$w = 1 / R2_{out} = 1, R3_{in} = 1$
B
$w = 0$
$w = 1$
/ $R1_{out} = 1, R2_{in} = 1$
C
$w = 0$
$w = 1$
/ $R3_{out} = 1, R1_{in} = 1, Done = 1$


w = 0
Reset
A
w = 1 / R2out = 1, R3in = 1
B
w = 0
w = 1 / R1out = 1, R2in = 1
C
w = 0
w = 1 / R3out = 1, R1in = 1, Done = 1

estado atual	Próximo Estado		saídas														
y2y1	Y2	Y1	R1out		R1in		R2out		R2in		R3out		R3in		done	
	w = 0	w = 1	w = 0	w = 1	w = 0	w = 1	w = 0	w = 1	w = 0	w = 1	w = 0	w = 1	w = 0	w = 1	w = 0	w = 1
	y2y1	y2y1														
A	A	B	0	0	0	0	0	1	0	0	0	0	0	1	0	0
B	C	C	1	1	0	0	0	0	1	1	0	0	0	0	0	0
C	A	A	0	0	1	1	0	0	0	0	1	1	0	0	1	1


$w = 0$
A
Reset
$w = 1 / R2_{out} = 1, R3_{in} = 1$
B
$w = 0$
$w = 1$
$/ R1_{out} = 1, R2_{in} = 1$
C
$w = 0$
$w = 1$
$/ R3_{out} = 1, R1_{in} = 1, Done = 1$

estado atual	Próximo Estado		saídas													
y2y1	Y2	Y1	R1out		R1in		R2out		R2in		R3out		R3in		done	
	w = 0	w = 1	w = 0	w = 1	w = 0	w = 1	w = 0	w = 1	w = 0	w = 1	w = 0	w = 1	w = 0	w = 1	w = 0	w = 1
	y2y1	y2y1														
A 00	A 00	B 01	0	0	0	0	0	1	0	0	0	0	0	1	0	0
B 01	C 10	C 10	1	1	0	0	0	0	1	1	0	0	0	0	0	0
C 10	A 00	A 00	0	0	1	1	0	0	0	0	1	1	0	0	1	1
11	dd	dd	d	d	d	d	d	d	d	d	d	d	d	d	d	d

estado atual	Próximo Estado		saídas													
y2y1	Y2	Y1	R1out		R1in		R2out		R2in		R3out		R3in		done	
	w = 0	w = 1	w = 0	w = 1	w = 0	w = 1	w = 0	w = 1	w = 0	w = 1	w = 0	w = 1	w = 0	w = 1	w = 0	w = 1
	y2y1	y2y1														
A 00	A 00	B 01	0	0	0	0	0	1	0	0	0	0	0	1	0	0
B 01	C 10	C 10	1	1	0	0	0	0	1	1	0	0	0	0	0	0
C 10	A 00	A 00	0	0	1	1	0	0	0	0	1	1	0	0	1	1
11	dd	dd	d	d	d	d	d	d	d	d	d	d	d	d	d	d

Y2 = y1

y2y1	00	01	11	10
w				
0	0	1	d	0
1	0	1	d	0

Y1 = y2’y1’w

y2y1	00	01	11	10
w				
0	0	0	d	0
1	1	0	d	0

R2out,R3in =y2’y1’w

y2y1	00	01	11	10
w				
0	0	0	d	0
1	1	0	d	0

R1out,R2in = y1

y2y1	00	01	11	10
w				
0	0	1	d	0
1	0	1	d	o

R3out,R2in,done = y2

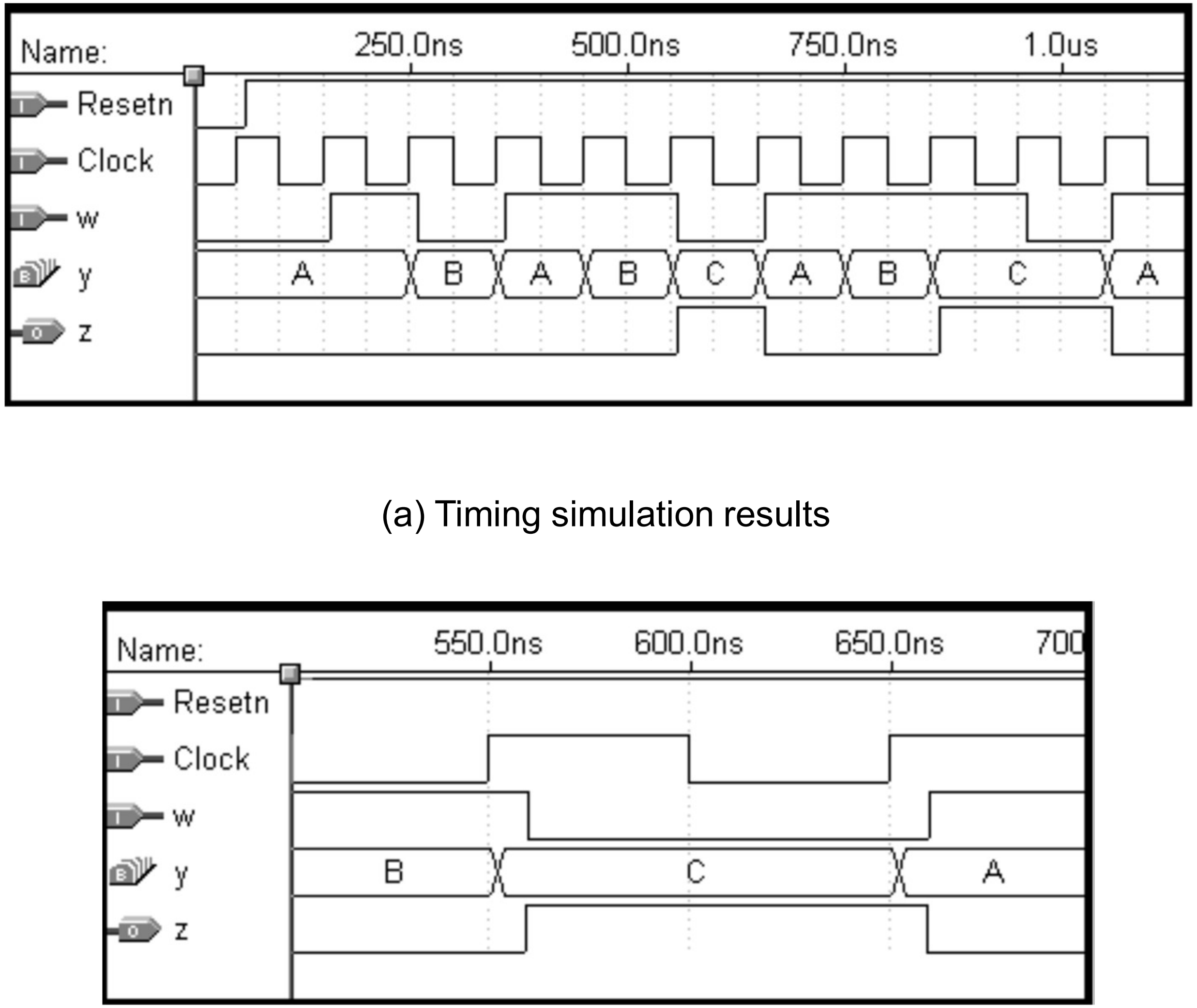
y2y1	00	01	11	10
w				
0	0	0	d	1
1	0	0	d	1

```
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 : State_type ;
BEGIN
    PROCESS ( Resetn, Clock )
    BEGIN
        IF Resetn = '0' THEN
            y <= A ;
        ELSIF (Clock'EVENT AND Clock = '1') THEN

con't ...
```

Figure 8.29a   VHDL code for a simple FSM

```
            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 ;
```

Figure 8.29b VHDL code for a simple FSM (con’t)



(a) Timing simulation results

(b) Magnified simulation results, showing timing details

Figure 8.32 Simulation results

```
(ENTITY declaration not shown)

ARCHITECTURE Behavior OF simple IS
        TYPE State_type IS (A, B, C) ;
        SIGNAL y_present, y_next : State_type ;
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 ;
```

Figure 8.33a Alternative style of code for an FSM

```
            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 ;
```

Figure 8.33b Alternative style of code for an FSM (con’t)

(ENTITY declaration not shown)

```
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 ...

Figure 8.34 A user-defined attribute for manual state assignment

```
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
```

Figure 8.35a Using constants for manual state assignment

```
            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 ;
```

Figure 8.35b Using constants for manual state assignment (cont’)

```
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
```

Figure 8.36 VHDL code for a Mealy machine

```
                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 ;
```

Figure 8.36b VHDL code for a Mealy machine (con’t)

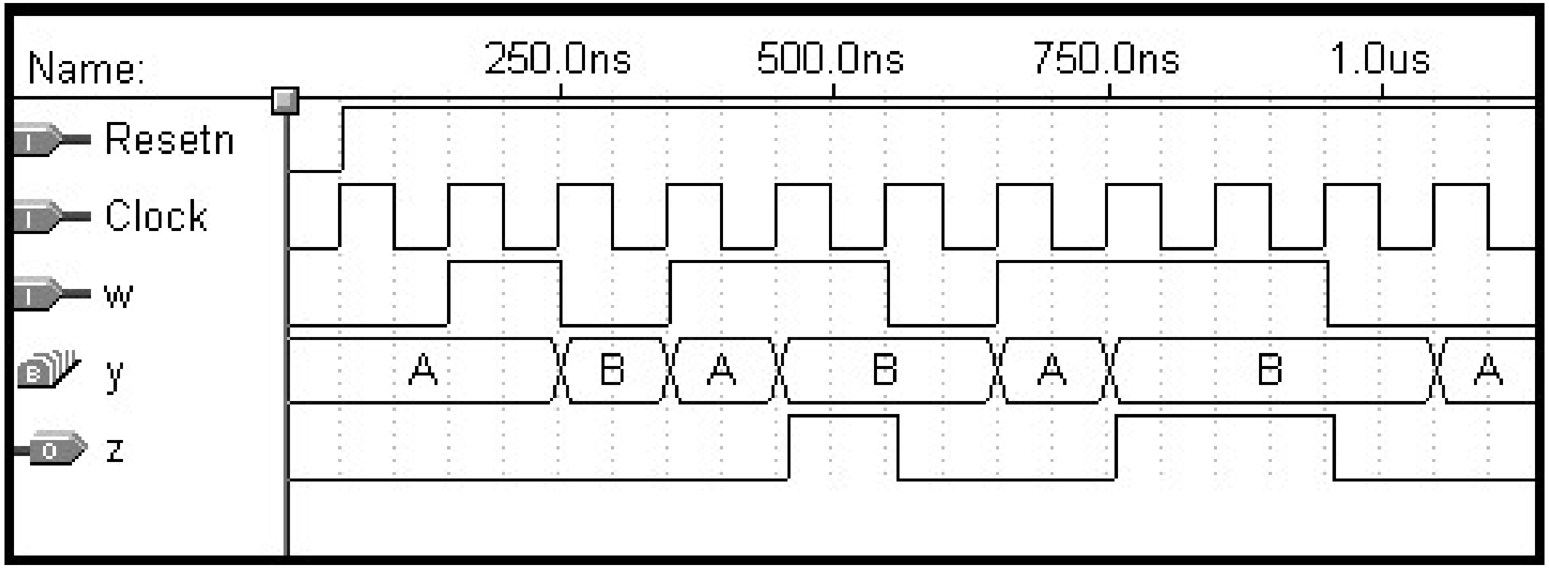


Figure 8.37 Simulation results for the Mealy machine

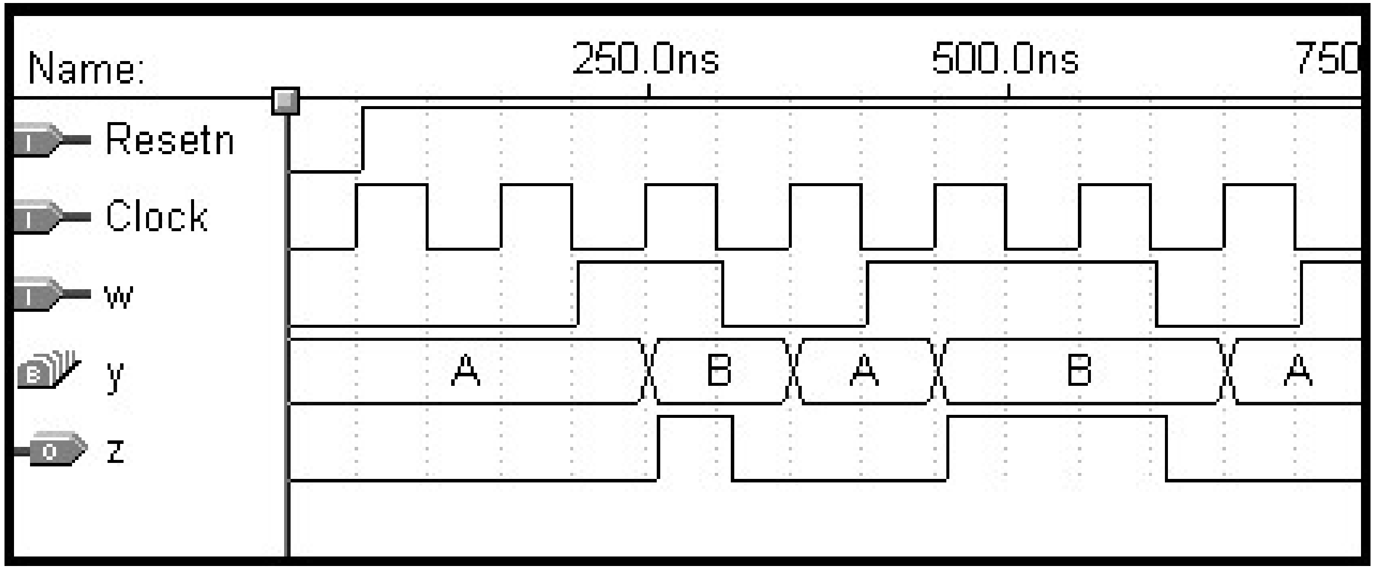
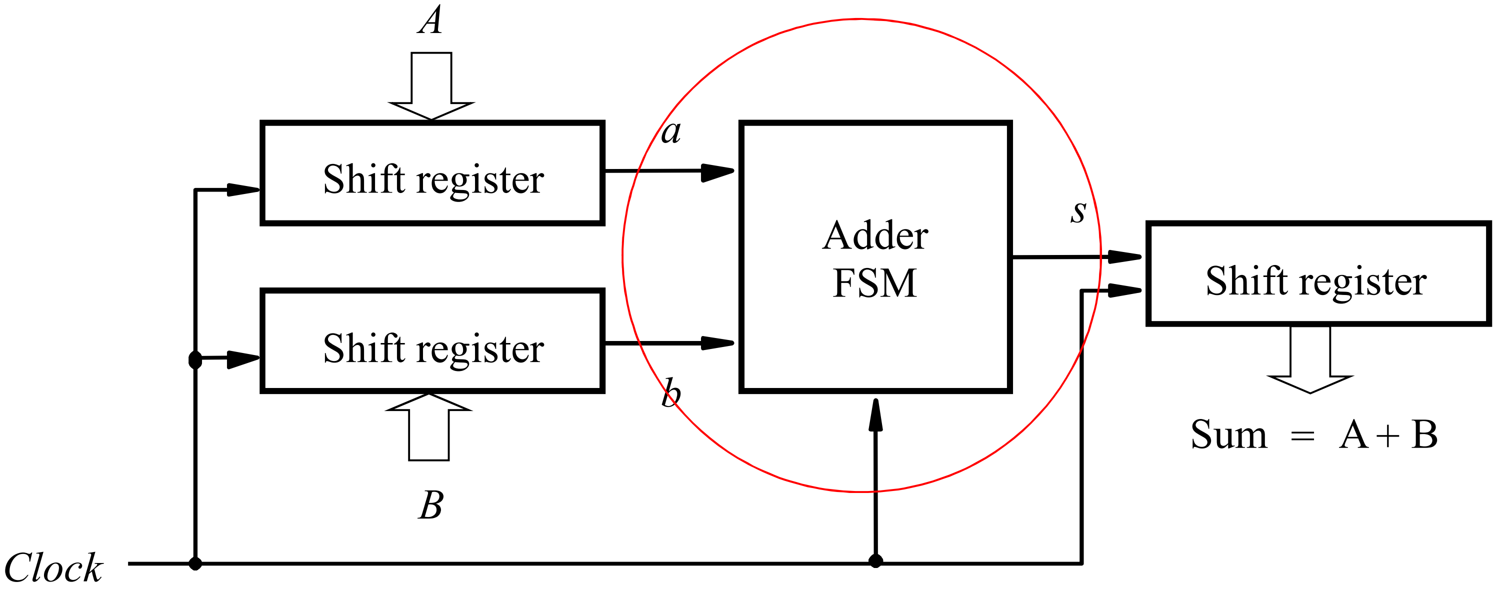
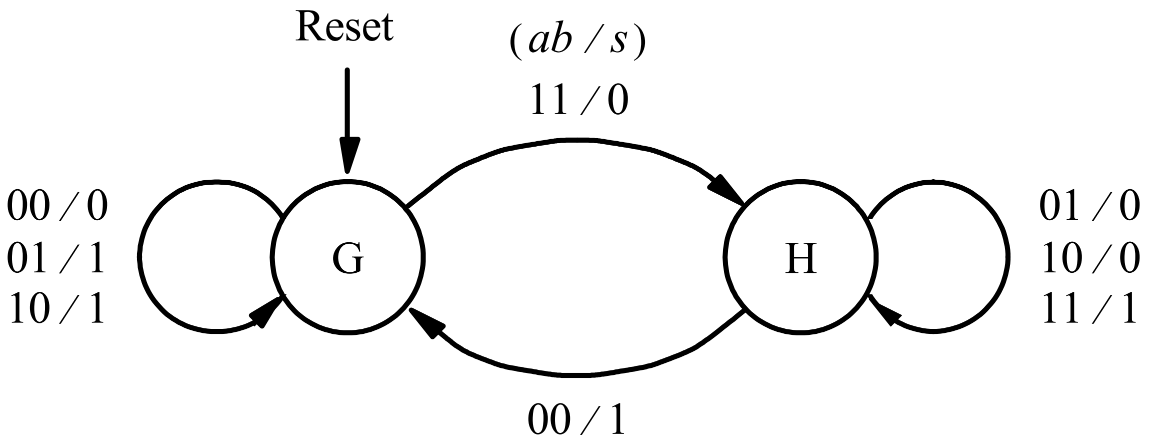


Figure 8.38 Potential problem with asynchronous inputs to a Mealy FSM

Exemplo – somador serial - MEALY



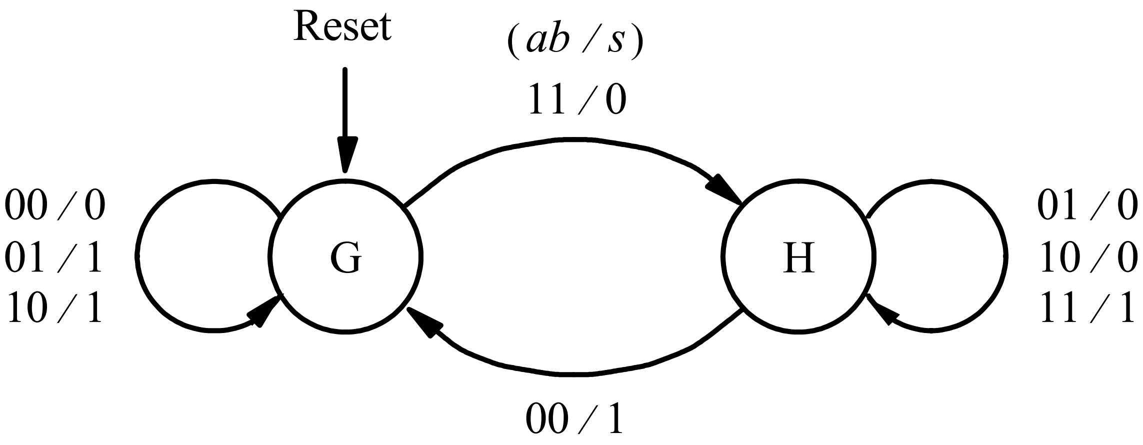
Somador serial – diagrama de estados - Mealy



G: carry-in = 0

H: carry-in = 1

Somador serial – Tabela de estados - Mealy



G: carry-in = 0

H: carry-in = 1

Present state	Next state				Output s			
	ab =00	01	10	11	00	01	10	11
G	G	G	G	H	0	1	1	0
H	G	H	H	H	1	0	0	1

Somador serial – Tabela de estados assinalados - Mealy

Present state	Next state				Output			
	ab =00	01	10	11	00	01	10	11
y	*Y*				*s*			
0	0	0	0	1	0	1	1	0
1	0	1	1	1	1	0	0	1

ab	00	01	11	10
y				
0	0	0	1	0
1	0	1	1	1

ab	00	01	11	10
y				
0	0	1	0	1
1	1	0	1	0

Y = ab + ay + by

s= a’b’y + a’by’ + aby + ab’y’ =
= a xor b xor y

Somador serial – Tabela de estados assinalados - Mealy

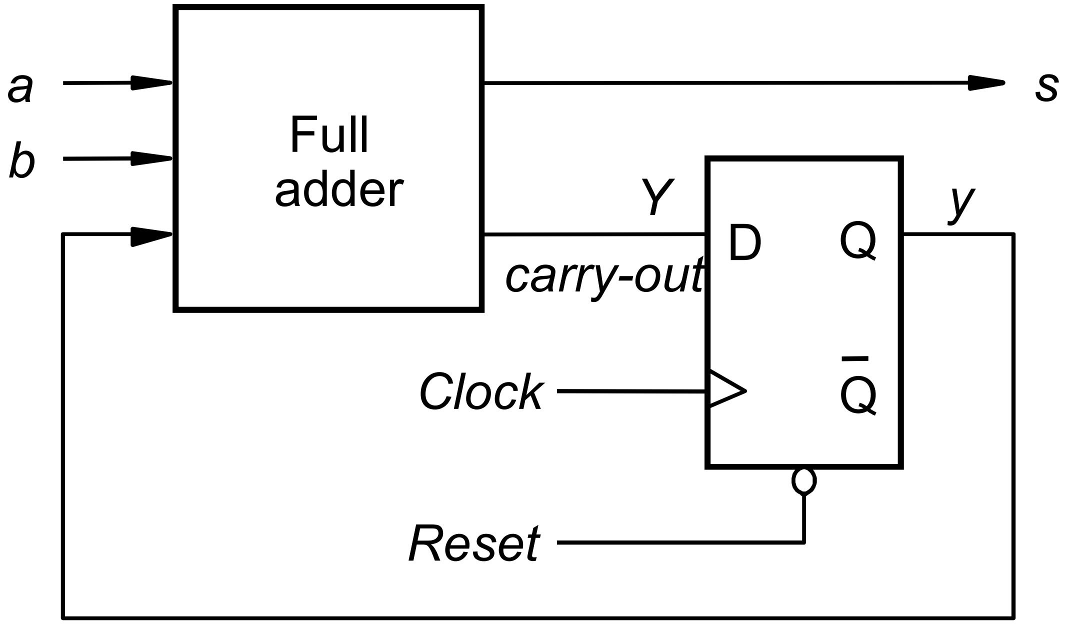
Present state	Next state				Output			
	ab =00	01	10	11	00	01	10	11
y	Y				s			
0	0	0	0	1	0	1	1	0
1	0	1	1	1	1	0	0	1

carry out

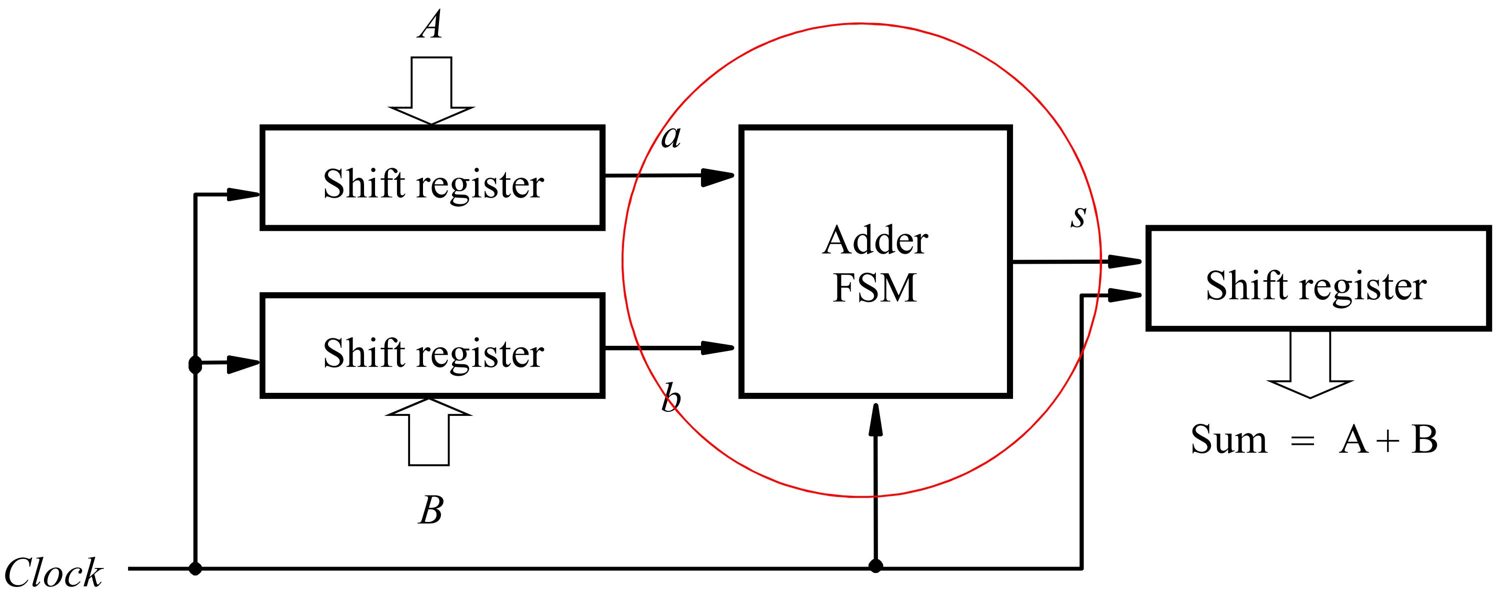
$$\left.\begin{array}{l} Y = a\,b + a\,y + b\,y \\ \\ s = a \oplus b \oplus y \end{array}\right\} \Rightarrow \text{Full adder}$$

soma

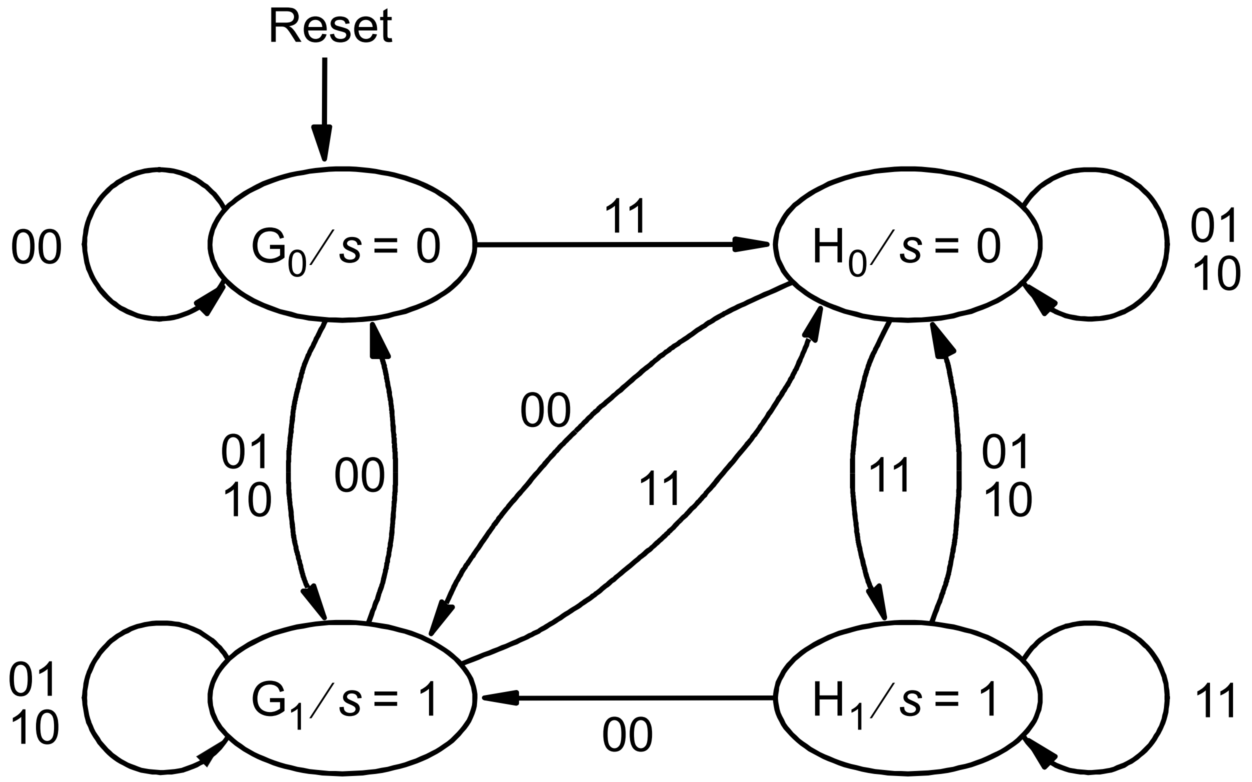
Somador serial – Mealy FSM



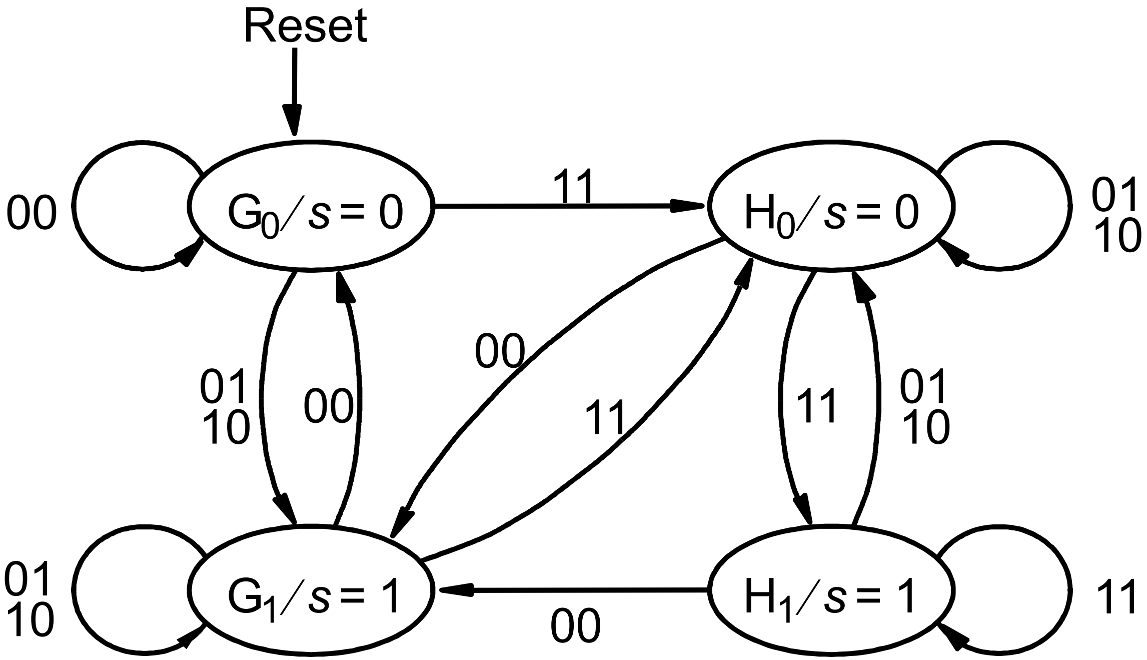
Exemplo – somador serial - MOORE



Somador serial – diagrama de estados - Moore



Somador serial – tabela de estados - Moore



Present state	Nextstate				Output
	ab =00	01	10	11	*s*
G_0	G_0	G_1	G_1	H_0	0
G_1	G_0	G_1	G_1	H_0	1
H_0	G_1	H_0	H_0	H_1	0
H_1	G_1	H_0	H_0	H_1	1

Somador serial – tabela de estados assinalados - Moore

Present	Nextstate				Output
state	ab =00	01	10	11	
y_2y_1	Y_2Y_1				s
00	0 0	01	0 1	10	0
01	0 0	01	0 1	10	1
10	0 1	10	1 0	11	0
11	0 1	10	1 0	11	1

y2y1	00	01	11	10
ab				
00	0	0	1	1
01	1	1	0	0
11	0	0	1	1
10	1	1	0	0

y2y1	00	01	11	10
ab				
00	0	0	0	0
01	0	0	1	1
11	1	1	1	1
10	0	0	1	1

y1 y2	0	1
0	0	1
1	0	1

s= y1

Y1 = ab’y’2 + a’by’2 + a’b’y2 +aby2

Y2 = ab + by2 + ay2

Somador serial – tabela de estados assinalados - Moore

Present state y_2y_1	Nextstate				Output s
	ab =00	01	10	11	
	Y_2Y_1				
00	0 0	01	0 1	10	0
01	0 0	01	0 1	10	1
10	0 1	10	1 0	11	0
11	0 1	10	1 0	11	1

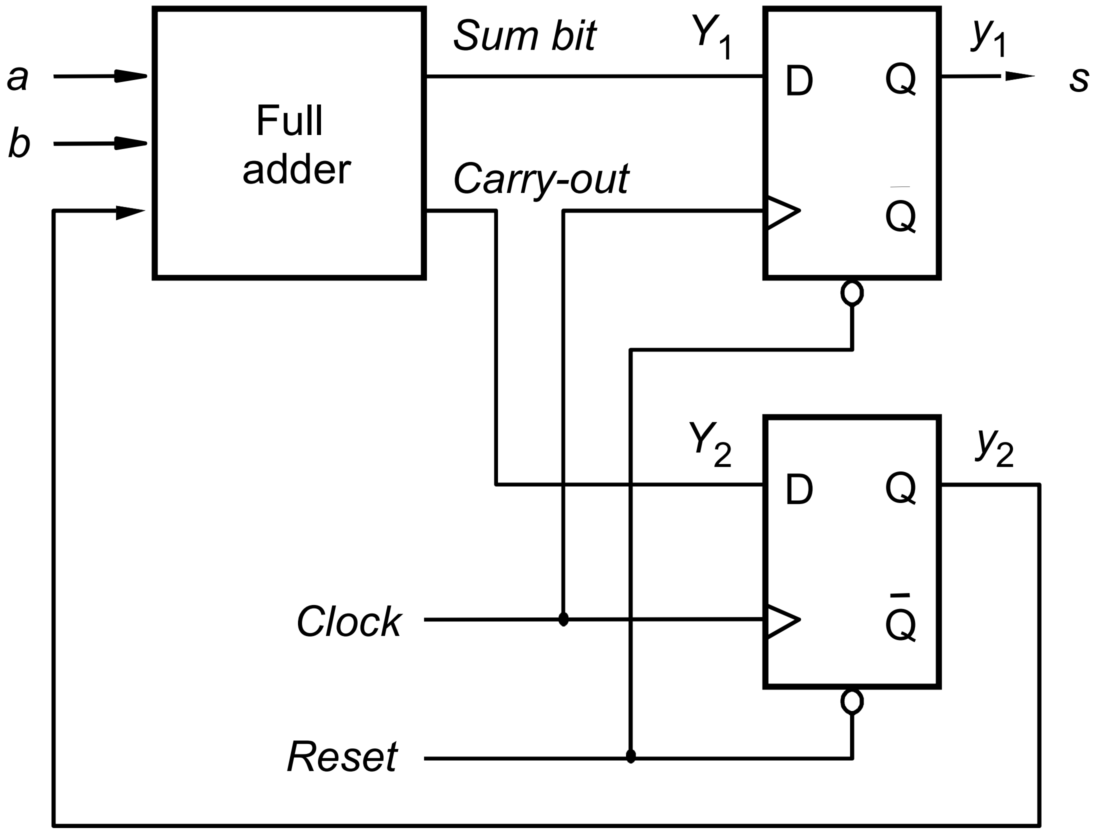
soma

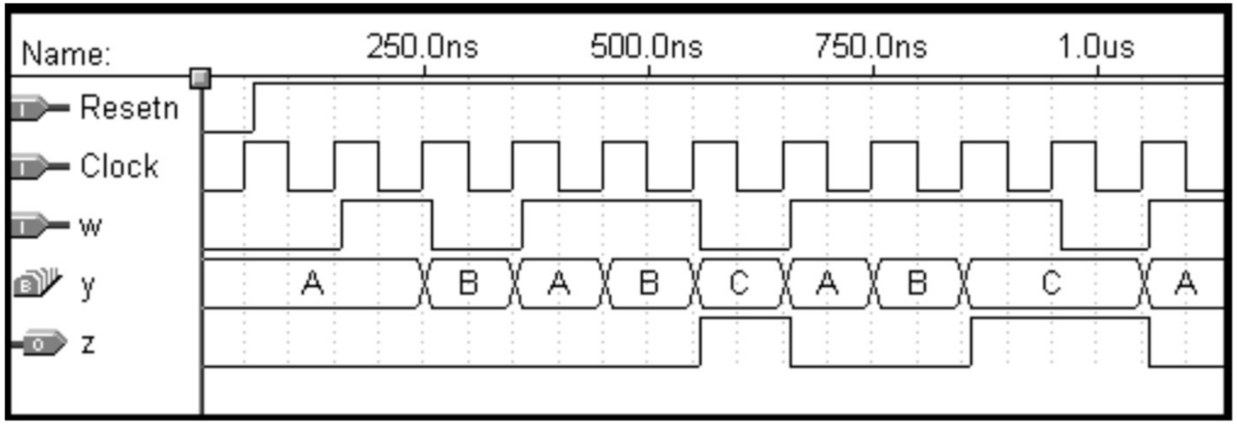
$$\left.\begin{array}{l} Y1 = a \oplus b \oplus y_2 \\ \\ Y_2 = a\,b + a\,y_2 + b\,y_2 \end{array}\right\} \longrightarrow \text{Full adder}$$

$s = y_1$

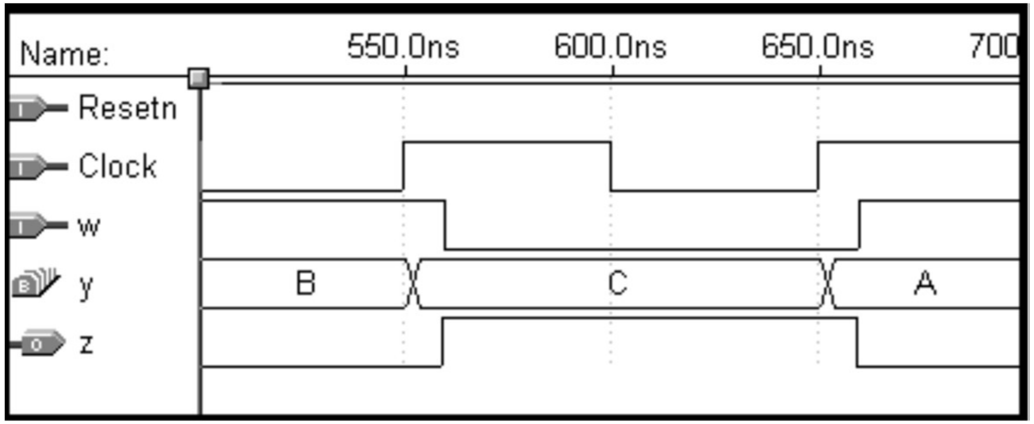
carry out

Somador serial –Moore FSM





(a) Timing simulation results



(b) Magnified simulation results, showing timing details

Figure 8.32 Simulation results

(ENTITY declaration not shown)

```
ARCHITECTURE Behavior OF simple IS
    TYPE State_type IS (A, B, C) ;
    SIGNAL y_present, y_next : State_type ;
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 ;
```

Figure 8.33a Alternative style of code for an FSM

```
            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 ;
```

Figure 8.33b Alternative style of code for an FSM (con’t)

(ENTITY declaration not shown)

```
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 ...

Figure 8.34 A user-defined attribute for manual state assignment

```
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
```

Figure 8.35a Using constants for manual state assignment

```
                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 ;
```

Figure 8.35b Using constants for manual state assignment (cont’)

```
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
```

Figure 8.36 VHDL code for a Mealy machine

```
                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 ;
```

Figure 8.36b VHDL code for a Mealy machine (con’t)

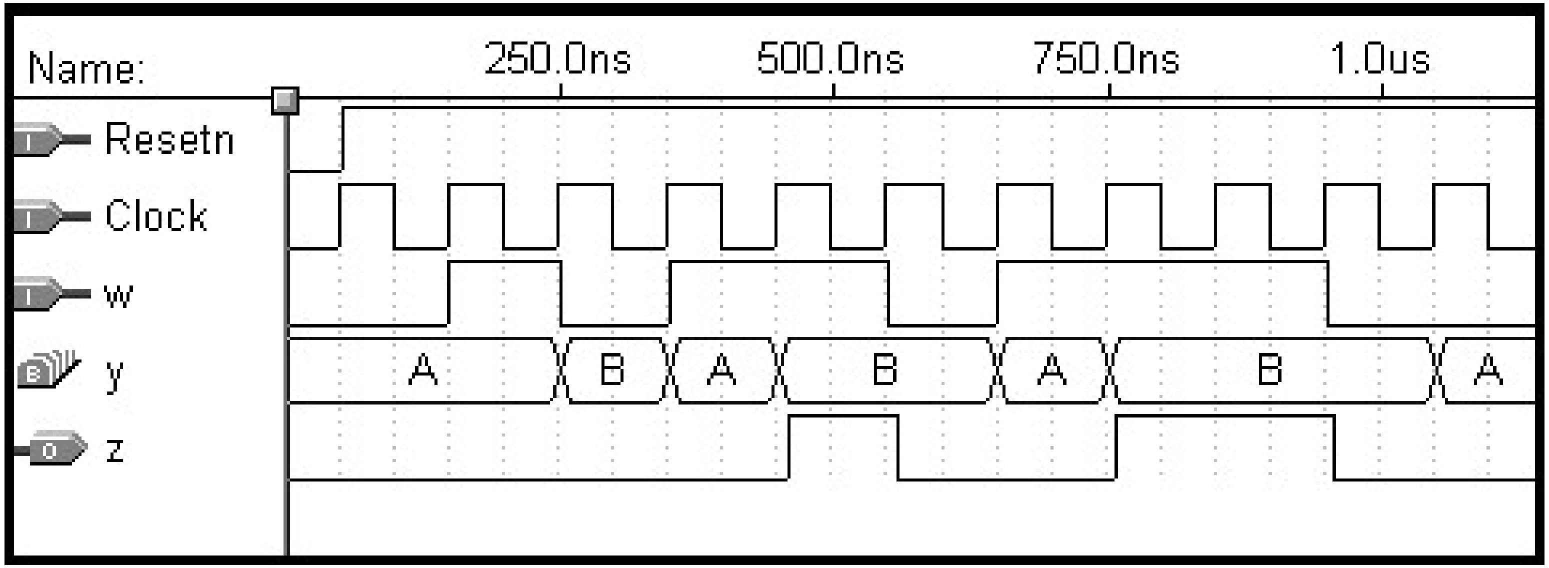


Figure 8.37 Simulation results for the Mealy machine

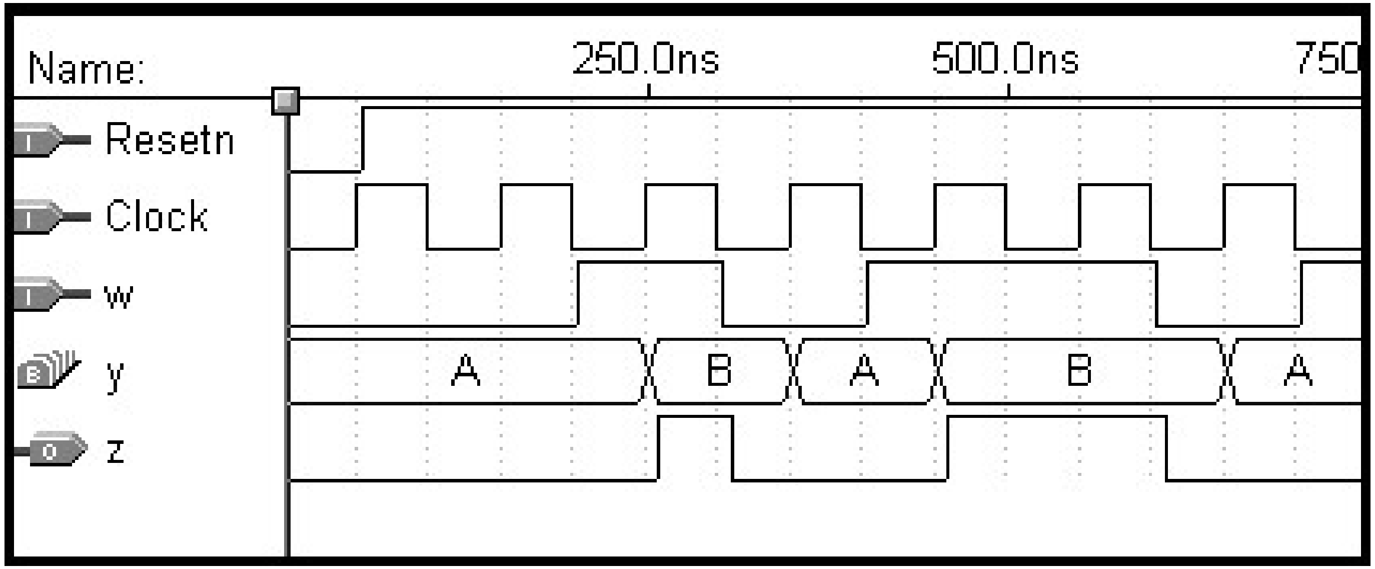


Figure 8.38 Potential problem with asynchronous inputs to a Mealy FSM

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;
-- left-to-right shift register with parallel load and enable
ENTITY shiftrne IS
        GENERIC ( N : INTEGER := 4 ) ;
        PORT (  R           : IN            STD_LOGIC_VECTOR(N-1 DOWNTO 0) ;
                L, E, w     : IN            STD_LOGIC ;
                Clock       : IN            STD_LOGIC ;
                Q           : BUFFER        STD_LOGIC_VECTOR(N-1 DOWNTO 0) ) ;
END shiftrne ;
ARCHITECTURE Behavior OF shiftrne IS
BEGIN
        PROCESS
        BEGIN

… con't
```

Figure 8.48a Code for a left-to-right shift register with an enable input

```
        WAIT UNTIL Clock'EVENT AND Clock = '1' ;
        IF E = '1' THEN
            IF L = '1' THEN
                Q <= R ;
            ELSE
                Genbits: FOR i IN 0 TO N-2 LOOP
                    Q(i) <= Q(i+1) ;
                END LOOP ;
                Q(N-1) <= w ;
            END IF ;
        END IF ;
    END PROCESS ;
END Behavior ;
```

Figure 8.48b Code for a left-to-right shift register with an enable input (con’t)

```
LIBRARY ieee ;
USE ieee.std_logic_1164.all ;
ENTITY serial IS
        GENERIC ( length : INTEGER := 8 ) ;
        PORT (  Clock     : IN          STD_LOGIC ;
                Reset     : IN          STD_LOGIC ;
                A, B      : IN          STD_LOGIC_VECTOR(length-1 DOWNTO 0) ;
                Sum       : BUFFER      STD_LOGIC_VECTOR(length-1 DOWNTO 0) );
END serial ;

ARCHITECTURE Behavior OF serial IS
        COMPONENT shiftrne
              GENERIC ( N : INTEGER := 4 ) ;
              PORT (   R          : IN          STD_LOGIC_VECTOR(N-1 DOWNTO 0) ;
                       L, E, w     : IN         STD_LOGIC ;
                       Clock      : IN          STD_LOGIC ;
                       Q          : BUFFER      STD_LOGIC_VECTOR(N-1 DOWNTO 0) ) ;
        END COMPONENT ;
        SIGNAL QA, QB, Null_in : STD_LOGIC_VECTOR(length-1 DOWNTO 0) ;
        SIGNAL s, Low, High, Run : STD_LOGIC ;
        SIGNAL Count : INTEGER RANGE 0 TO length ;
        TYPE State_type IS (G, H) ;
        SIGNAL y : State_type ;

… con’t
```

Figure 8.49a VHDL code for the serial adder

```
BEGIN
    Low <= '0' ; High <= '1' ;
    ShiftA: shiftrne GENERIC MAP (N => length)
        PORT MAP ( A, Reset, High, Low, Clock, QA ) ;
    ShiftB: shiftrne GENERIC MAP (N => length)
        PORT MAP ( B, Reset, High, Low, Clock, QB ) ;
    AdderFSM: PROCESS ( Reset, Clock )
    BEGIN
        IF Reset = '1' THEN
            y <= G ;
        ELSIF Clock'EVENT AND Clock = '1' THEN
            CASE y IS
                WHEN G =>
                    IF QA(0) = '1' AND QB(0) = '1' THEN y <= H ;
                    ELSE y <= G ;
                    END IF ;
                WHEN H =>
                    IF QA(0) = '0' AND QB(0) = '0' THEN y <= G ;
                    ELSE y <= H ;
                    END IF ;
            END CASE ;
        END IF ;
    END PROCESS AdderFSM ;

… con’t
```

Figure 8.49b VHDL code for the serial adder (con’t)

```
    WITH y SELECT
        s <=  QA(0) XOR QB(0) WHEN G,
              NOT ( QA(0) XOR QB(0) ) WHEN H ;
    Null_in <= (OTHERS => '0') ;
    ShiftSum: shiftrne GENERIC MAP ( N => length )
        PORT MAP ( Null_in, Reset, Run, s, Clock, Sum ) ;
    Stop: PROCESS
    BEGIN
        WAIT UNTIL (Clock'EVENT AND Clock = '1') ;
        IF Reset = '1' THEN
            Count <= length ;
        ELSIF Run = '1' THEN
            Count <= Count -1 ;
        END IF ;
    END PROCESS ;
    Run <= '0' WHEN Count = 0 ELSE '1' ;  -- stops counter and ShiftSum
END Behavior ;
```

Figure 8.49c VHDL code for the serial adder (con’t)

Minimização de estados

Definição 1 – Sejam 2 estados Si e Sj. Eles são ditos equivalentes se somente se para todas as possíveis sequências de entradas, a mesma sequência de saída é produzida, Levando para o mesmo estado ou para estados equivalentes.
OBS – É mais fácil mostrar os estados que não são equivalentes.

Procedimento de Particionamento

1. Separar em blocos, onde os estados com mesma saída devam pertencer ao bloco
2. Verificar os sucessores para cada combinação de entrada. Se estes sucessores pertencerem a um determinado subconjunto, deixá-lo no subconjunto, senão, criar outro subconjunto com estes estados.
3. Repetir até conseguir um resultado igual ao anterior.

Minimização de estados

Tabela de Estados

Present state	Next state		Output z
	$w = 0$	$w = 1$	
A	B	C	1
B	D	F	1
C	F	E	0
D	B	G	1
E	F	C	0
F	E	D	0
G	F	G	0

Minimização de estados

Tabela de Estados

Present state	Next state		Output
	$w = 0$	$w = 1$	z
A	B	C	1
B	D	F	1
C	F	E	0
D	B	G	1
E	F	C	0
F	E	D	0
G	F	G	0

P1 = (ABCDEFG)

P2 = (ABD) (CEFG) ➔ próximos estados para w = 0 (BDB) (FFEF)
➔ próximos estados para w = 1 (CFG) (ECDG)

P3 = (ABD) (CEG) (F) ➔ próximos estados para w = 0 (BDB) (FFF) (E)
➔ próximos estados para w = 1 (CFG) (ECG) (D)

P4 = (AD) (B) (CEG) (F) ➔ próximos estados para w = 0 (BB) (D) (FFF) (E)
➔ próximos estados para w = 1 (CG) (F) (ECG) (D)

P5= (AD) (B) (CEG) (F) ➔ FIM

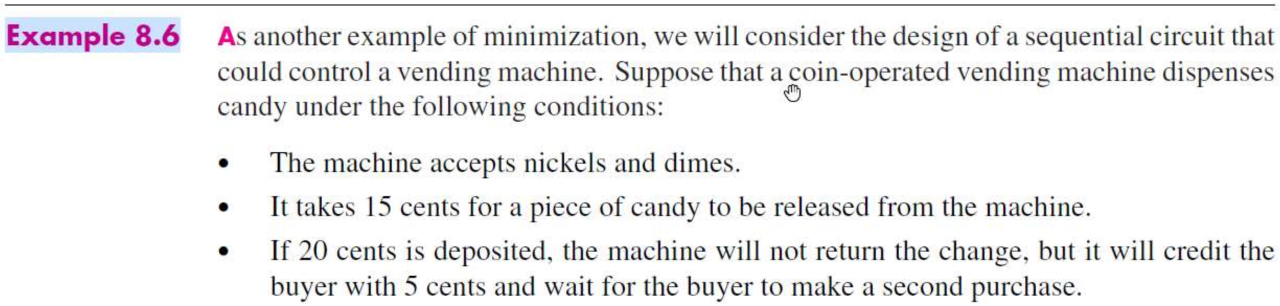
Minimização de estados

Tabela de Estados

Present state	Next state		Output z
	$w = 0$	$w = 1$	
A	B	C	1
B	D	F	1
C	F	E	0
D	B	G	1
E	F	C	0
F	E	D	0
G	F	G	0

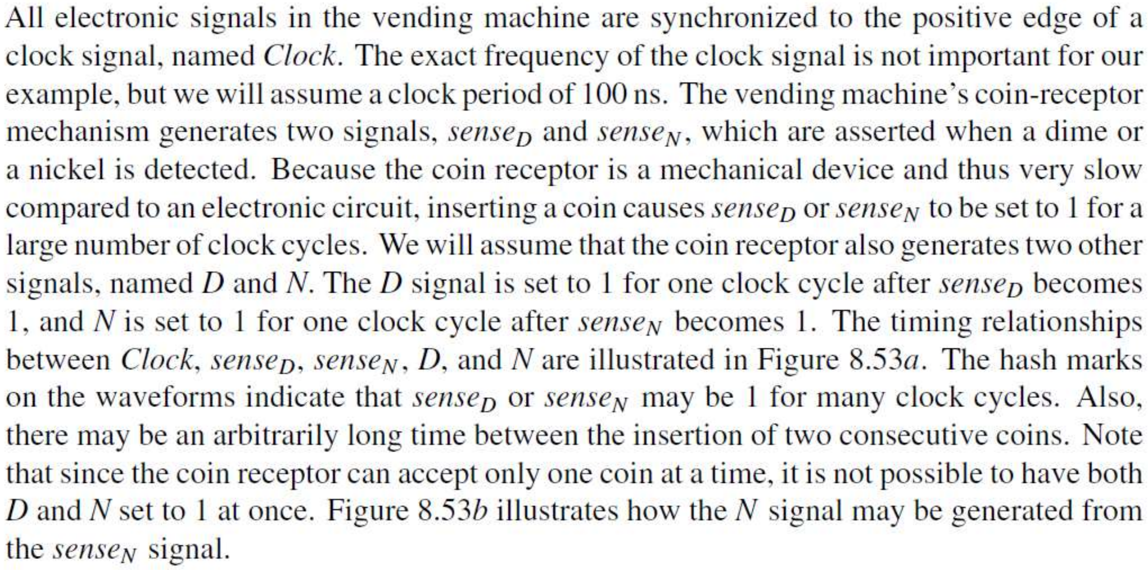
Present state	Next state		Output z
	w = 0	w = 1	
A	B	C	1
B	A	F	1
C	F	C	0
F	C	A	0

P1 = (ABCDEFG)		
P2 = (ABD) (CEFG)	➔ próximos estados para w = 0	(BDB) (FFEF)
	➔ próximos estados para w = 1	(CFG) (ECDG) - D não pertence ao grupo de C E G
P3 = (ABD) (CEG) (F)	➔ próximos estados para w = 0	(BDB) (FFF) (E)
	➔ próximos estados para w = 1	(CFG) (ECG) (D) - F não pertence ao grupo C G
P4 = (AD) (B) (CEG) (F)	➔ próximos estados para w = 0	(BB) (D) (FFF) (D)
	➔ próximos estados para w = 1	(CG) (F) (ECG) (D)
P5= (AD) (B) (CEG) (F)	➔ FIM	



Example 8.6 As another example of minimization, we will consider the design of a sequential circuit that could control a vending machine. Suppose that a coin-operated vending machine dispenses candy under the following conditions:

- The machine accepts nickels and dimes.
- It takes 15 cents for a piece of candy to be released from the machine.
- If 20 cents is deposited, the machine will not return the change, but it will credit the buyer with 5 cents and wait for the buyer to make a second purchase.



All electronic signals in the vending machine are synchronized to the positive edge of a clock signal, named *Clock*. The exact frequency of the clock signal is not important for our example, but we will assume a clock period of 100 ns. The vending machine's coin-receptor mechanism generates two signals, $sense_D$ and $sense_N$, which are asserted when a dime or a nickel is detected. Because the coin receptor is a mechanical device and thus very slow compared to an electronic circuit, inserting a coin causes $sense_D$ or $sense_N$ to be set to 1 for a large number of clock cycles. We will assume that the coin receptor also generates two other signals, named D and N. The D signal is set to 1 for one clock cycle after $sense_D$ becomes 1, and N is set to 1 for one clock cycle after $sense_N$ becomes 1. The timing relationships between *Clock*, $sense_D$, $sense_N$, D, and N are illustrated in Figure 8.53*a*. The hash marks on the waveforms indicate that $sense_D$ or $sense_N$ may be 1 for many clock cycles. Also, there may be an arbitrarily long time between the insertion of two consecutive coins. Note that since the coin receptor can accept only one coin at a time, it is not possible to have both D and N set to 1 at once. Figure 8.53*b* illustrates how the N signal may be generated from the $sense_N$ signal.

Máquina de Moore, cujas entradas são N(5) e D(10), saída z que, quando = 1, libera o produto.

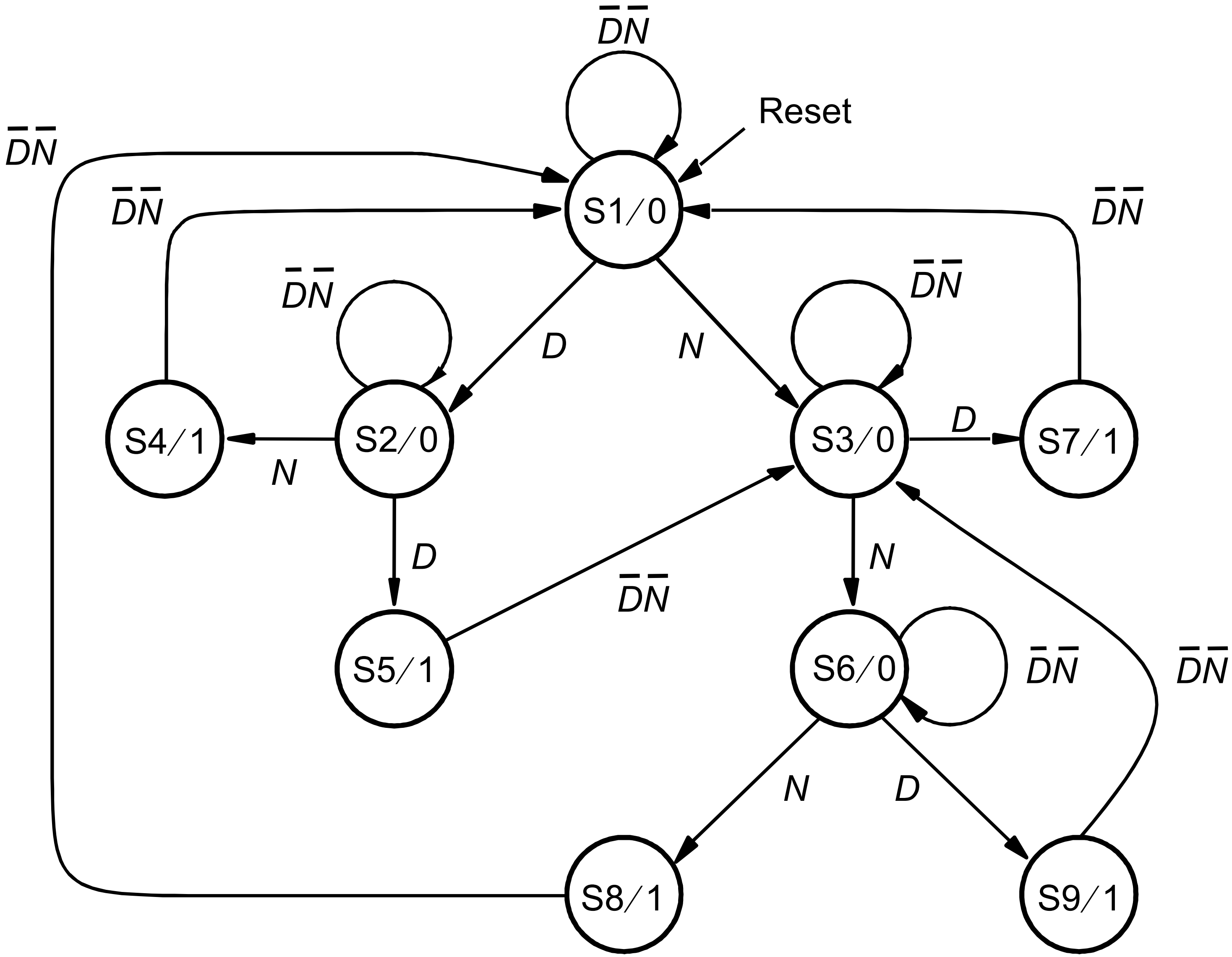
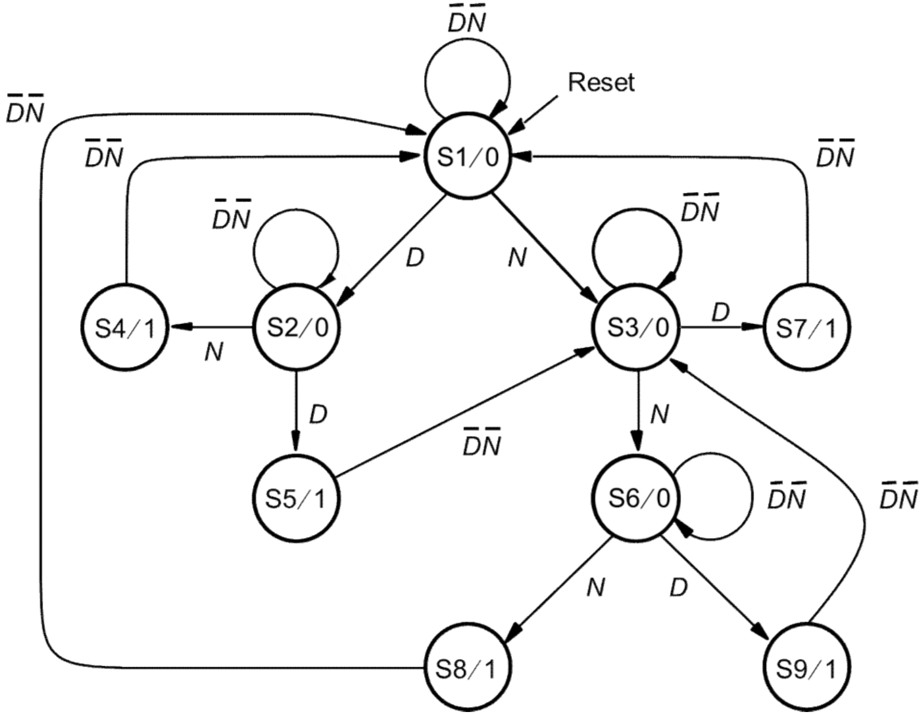


Figure 8.54 State diagram for Example 8.6

Present state	Next state				Output
	DN =00	01	10	11	*z*
S1	S1	S3	S2	–	0
S2	S2	S4	S5	–	0
S3	S3	S6	S7	–	0
S4	S1	–	–	–	1
S5	S3	–	–	–	1
S6	S6	S8	S9	–	0
S7	S1	–	–	–	1
S8	S1	–	–	–	1
S9	S3	–	–	–	1

Figura 8.55 Tabela de Estados -
Exemplo 8.6

Present state	Next state				Output z
	DN =00	01	10	11	
S1	S1	S3	S2	–	0
S2	S2	S4	S5	–	0
S3	S3	S6	S7	–	0
S4	S1	–	–	–	1
S5	S3	–	–	–	1
S6	S6	S8	S9	–	0
S7	S1	–	–	–	1
S8	S1	–	–	–	1
S9	S3	–	–	–	1



Present state	Next state				Output z
	DN =00	01	10	11	
S1	S1	S3	S2	–	0
S2	S2	S4	S5	–	0
S3	S3	S6	S7	–	0
S4	S1	–	–	–	1
S5	S3	–	–	–	1
S6	S6	S8	S9	–	0
S7	S1	–	–	–	1
S8	S1	–	–	–	1
S9	S3	–	–	–	1

P1 = (S1S2S3S4S5S6S7S8S9)
P2 = (S1 S2 S3 S6) (S4 S5 S7 S8 S9) ➔ SUCESSORES 00 (S1 S2 S3 S6) (S1 S3 S1 S1 S3)
➔ SUCESSORES 01 (S3 S4 S6 S8) (- - - - -)
➔ SUCESSORES 10 (S2 S5 S7 S9) (- - - - -)
➔ SUCESSORES 11 (- - - -) (- - - - -)
P3 = (S1 S3) (S2 S6) (S4 S5 S7 S8 S9) ➔ SUCESSORES 00 (S1 S3) (S2 S6) (S1 S3 S1 S1 S3)
➔ SUCESSORES 01 (S3 S6) (S4 S8) (- - - - -)
➔ SUCESSORES 10 (S2 S7) (S5 S9) (- - - - -)
➔ SUCESSORES 11 (- -) (- -) (- - - - -)
P4 = (S1) (S3) (S2 S6) (S4 S5 S7 S8 S9) ➔ SUCESSORES 00 (S1) (S3) (S2 S6) (S1 S3 S1 S1 S3)
➔ SUCESSORES 01 (S3) (S6) (S4 S8) (- - - - -)
➔ SUCESSORES 10 (S2) (S7) (S5 S9) (- - - - -)
➔ SUCESSORES 11 (-) (-) (- -) (- - - - -)
P5 = (S1) (S3) (S2 S6) (S4 S7 S8) (S5 S9) ➔ SUCESSORES 00 (S1) (S3) (S2 S6) (S1 S1 S1) (S3 S3)
➔ SUCESSORES 01 (S3) (S6) (S4 S8) (- - - - -)
➔ SUCESSORES 10 (S2) (S7) (S5 S9) (- - - - -)
➔ SUCESSORES 11 (-) (-) (- -) (- - - - -)
P6 = (S1) (S3) (S2 S6) (S4 S7 S8) (S5 S9)

Present state	Next state				Output
	DN =00	01	10	11	*z*
S1	S1	S3	S2	–	0
S2	S2	S4	S5	–	0
S3	S3	S6	S7	–	0
S4	S1	–	–	–	1
S5	S3	–	–	–	1
S6	S6	S8	S9	–	0
S7	S1	–	–	–	1
S8	S1	–	–	–	1
S9	S3	–	–	–	1

Present state	Next state Y3Y2Y1				Output
	DN				
y3y2y1	00	01	10	11	*z*
S1 000	S1 000	S3 010	S2 001	–	0
S2 001	S2 001	S4 011	S5 100	–	0
S3 010	S3 010	S2 001	S4 011	–	0
S4 011	S1 000	–	–	–	1
S5 100	S3 010	–	–	–	1

Figure 8.56 Minimized state table for Example 8.6

Present state y3y2y1	Next state Y3Y2Y1				Output z
	DN				
	00	01	10	11	
S1 000	S1 000	S3 010	S2 001	–	0
S2 001	S2 001	S4 011	S5 100	–	0
S3 010	S3 010	S2 001	S4 011	–	0
S4 011	S1 000	–	–	–	1
S5 100	S3 010	–	–	–	1

$$Y3 = y1.D$$

y3 y2	00	01	11	10
y1 D				
00	0	0	d	0
01	0	0	d	d
11	1	d	d	d
10	0	0	d	d

N=0

y3 y2	00	01	11	10
y1 D				
00	0	0	d	d
01	d	d	d	d
11	d	d	d	d
10	0	d	d	d

N=1

Figure 8.56 Minimized state table for Example 8.6

Present state	Next state Y3Y2Y1				Output
	DN				
y3y2y1	00	01	10	11	*z*
S1 000	S1 000	S3 010	S2 001	–	0
S2 001	S2 001	S4 011	S5 100	–	0
S3 010	S3 010	S2 001	S4 011	–	0
S4 011	S1 000	–	–	–	1
S5 100	S3 010	–	–	–	1

Y2 = N y’2 + y3y’2 +Ny2y’1

y3 y2	00	01	11	10
y1 D				
00	0	1	d	1
01	0	1	d	d
11	0	d	d	d
10	0	0	d	d

N=0

y3 y2	00	01	11	10
y1 D				
00	1	0	d	d
01	d	d	d	d
11	d	d	d	d
10	1	d	d	d

N=1

Figure 8.56 Minimized state table for Example 8.6

Present state y3y2y1	Next state Y3Y2Y1				Output *z*
	DN 00	01	10	11	
S1 000	S1 000	S3 010	S2 001	–	0
S2 001	S2 001	S4 011	S5 100	–	0
S3 010	S3 010	S2 001	S4 011	–	0
S4 011	S1 000	–	–	–	1
S5 100	S3 010	–	–	–	1

Y1 = y’3y2 +y3 N + y´2y1 N’

y3 y2	00	01	11	10
y1 D				
00	0	0	d	0
01	1	1	d	d
11	0	d	d	d
10	1	0	d	d

N=0

y3 y2	00	01	11	10
y1 D				
00	0	1	d	d
01	d	d	d	d
11	d	d	d	d
10	1	d	d	d

N=1

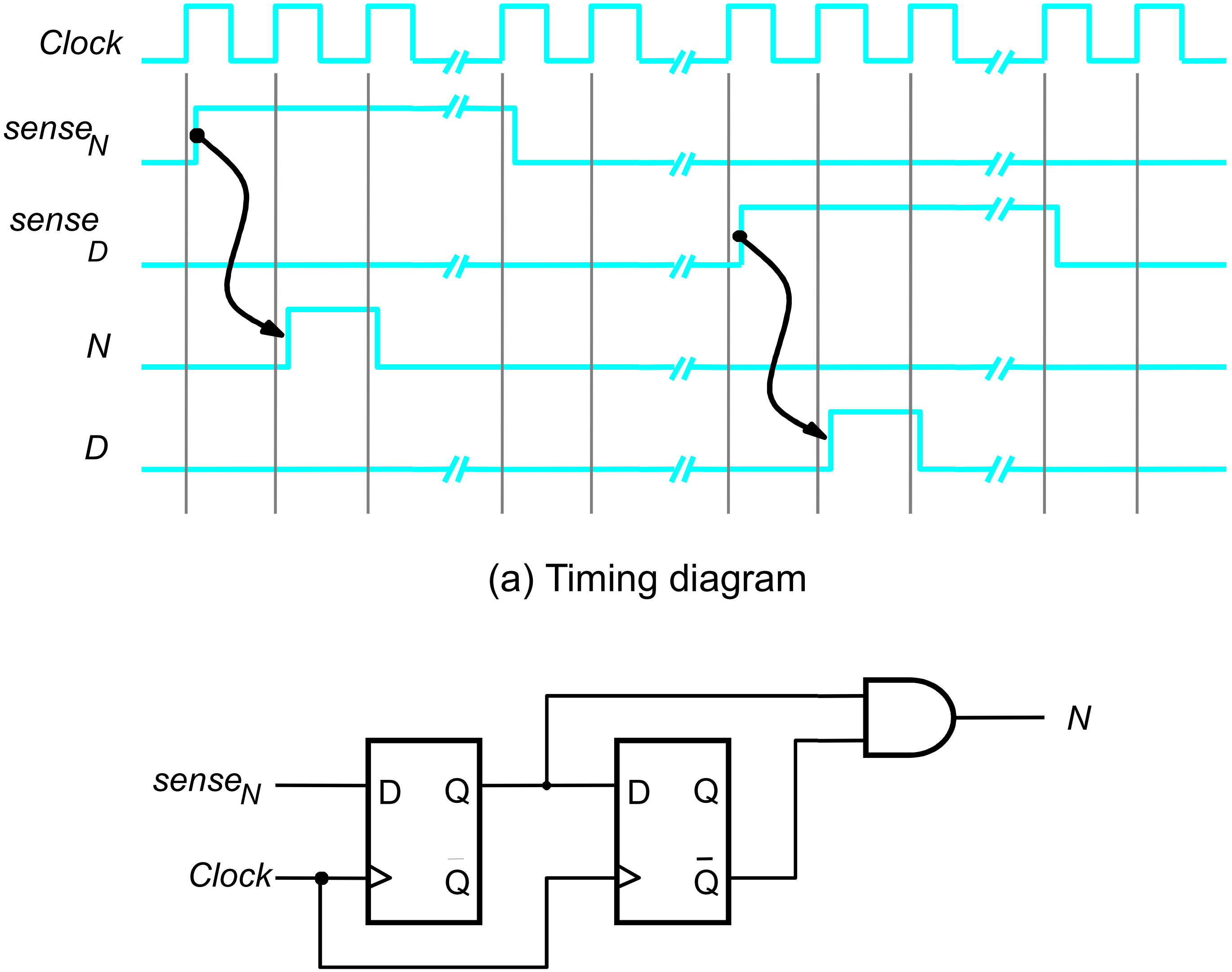
Figure 8.56 Minimized state table for Example 8.6

Present state	Next state Y3Y2Y1				Output
	DN				
y3y2y1	00	01	10	11	*z*
S1 000	S1 000	S3 010	S2 001	–	0
S2 001	S2 001	S4 011	S5 100	–	0
S3 010	S3 010	S2 001	S4 011	–	0
S4 011	S1 000	–	–	–	1
S5 100	S3 010	–	–	–	1

z = y3

y3 y2	00	01	11	10
y1				
0	0	0	d	1
1	0	0	d	d

Figure 8.56 Minimized state table for Example 8.6



(a) Timing diagram

(b) Circuit that generates *N*

Figure 8.53 Signals for the vending machine

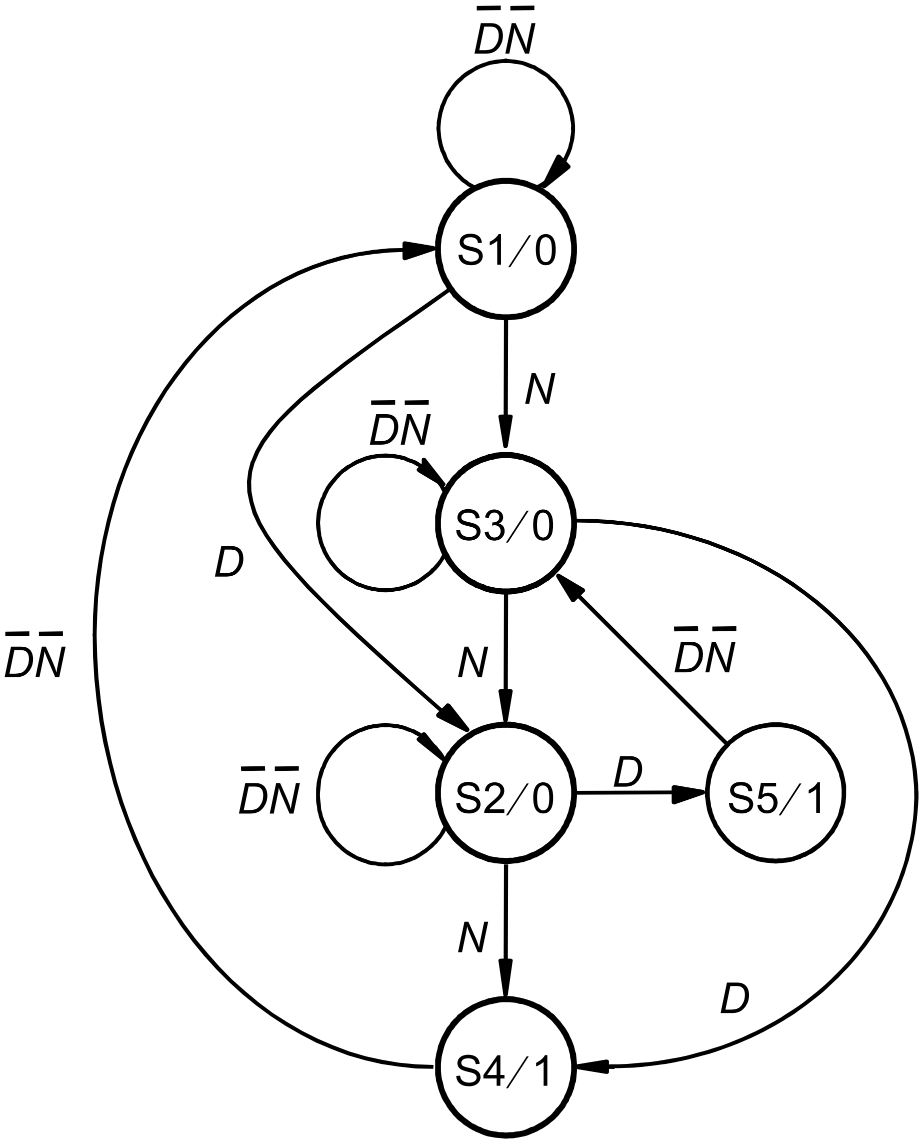
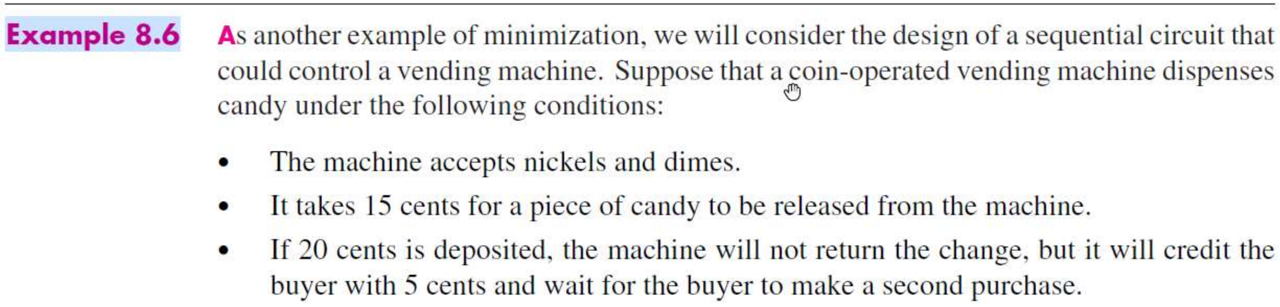
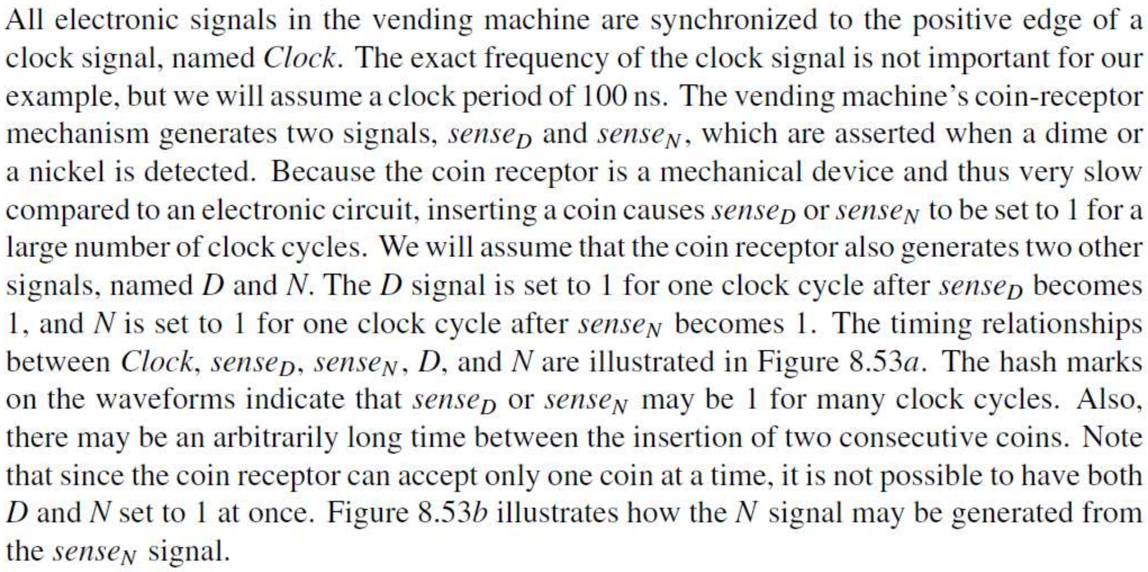


Figure 8.57 Minimized state diagram for Example 8.6



Example 8.6 As another example of minimization, we will consider the design of a sequential circuit that could control a vending machine. Suppose that a coin-operated vending machine dispenses candy under the following conditions:

- The machine accepts nickels and dimes.
- It takes 15 cents for a piece of candy to be released from the machine.
- If 20 cents is deposited, the machine will not return the change, but it will credit the buyer with 5 cents and wait for the buyer to make a second purchase.



All electronic signals in the vending machine are synchronized to the positive edge of a clock signal, named *Clock*. The exact frequency of the clock signal is not important for our example, but we will assume a clock period of 100 ns. The vending machine's coin-receptor mechanism generates two signals, $sense_D$ and $sense_N$, which are asserted when a dime or a nickel is detected. Because the coin receptor is a mechanical device and thus very slow compared to an electronic circuit, inserting a coin causes $sense_D$ or $sense_N$ to be set to 1 for a large number of clock cycles. We will assume that the coin receptor also generates two other signals, named D and N. The D signal is set to 1 for one clock cycle after $sense_D$ becomes 1, and N is set to 1 for one clock cycle after $sense_N$ becomes 1. The timing relationships between *Clock*, $sense_D$, $sense_N$, D, and N are illustrated in Figure 8.53*a*. The hash marks on the waveforms indicate that $sense_D$ or $sense_N$ may be 1 for many clock cycles. Also, there may be an arbitrarily long time between the insertion of two consecutive coins. Note that since the coin receptor can accept only one coin at a time, it is not possible to have both D and N set to 1 at once. Figure 8.53*b* illustrates how the N signal may be generated from the $sense_N$ signal.

Máquina de Mealy, cujas entradas são N(5) e D(10), saída z que, quando = 1, libera o produto.

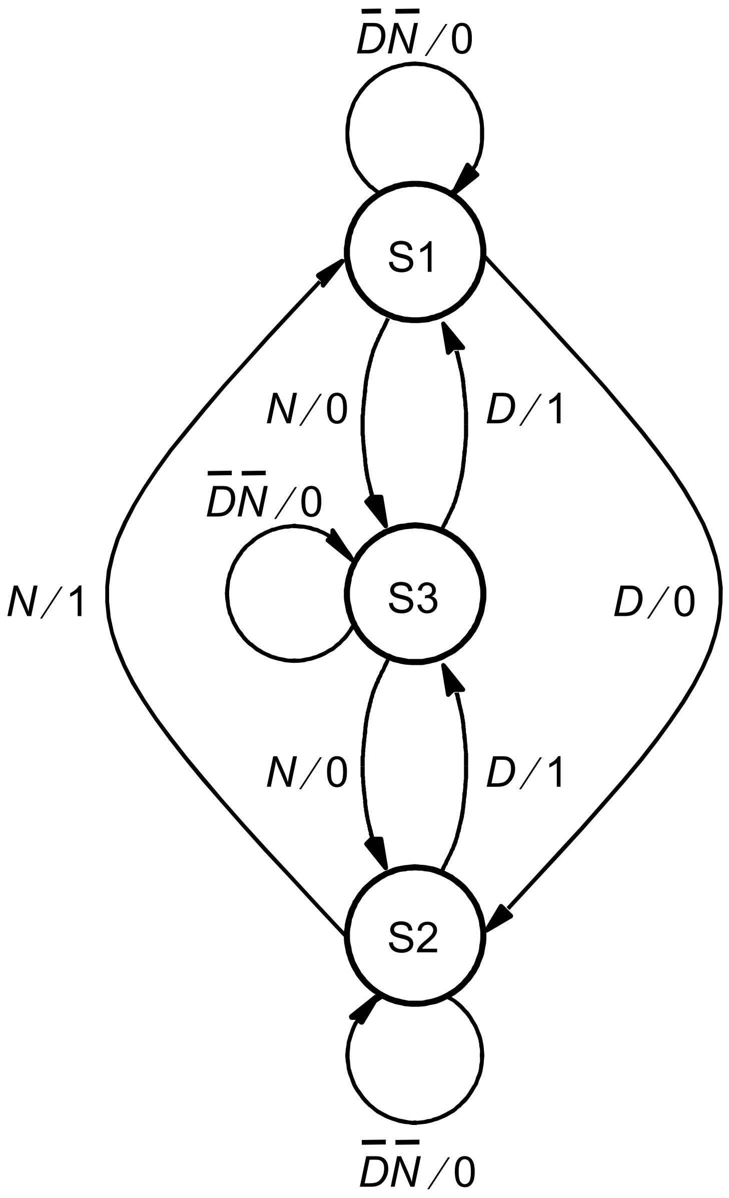


Figure 8.58 Mealy-type FSM for Example 8.6

ESTADO ATUAL	PRÓXIMO ESTADO - Y1Y2				SAÍDAS - z			
y1y2	D'N'	D'N	DN'	DN	D'N'	D'N	DN'	DN
S1 00	S1 0 0	S3 1 0	S2 0 1	NN	0	0	0	NN
S2 01	S2 0 1	S1 0 0	S3 1 0	NN	0	1	1	NN
S3 10	S3 1 0	S2 0 1	S1 0 0	NN	0	0	1	NN

Y1 = y1D’N’+y1’y2’N+y2D

y1y2/ DN	00	01	11	10
00	0	0	d	1
01	1	0	d	0
11	d	d	d	d
10	0	1	d	0

Y2 = y1’y2’D+y2D’N’+y1N

y1y2/ DN	00	01	11	10
00	0	1	d	0
01	0	0	d	1
11	d	d	d	d
10	1	0	d	0

z = y2N +y2D+y1D

y1y2/ DN	00	01	11	10
00	0	0	d	0
01	0	1	d	0
11	d	d	d	d
10	0	1	d	1

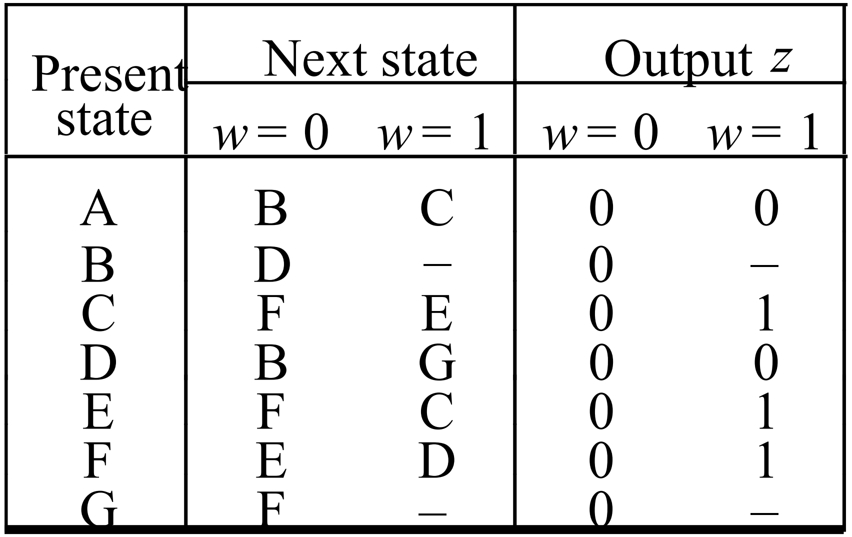
Mealy-type FSM for Example 8.6

Tabela com estados não definidos

Present state	Next state		Output z	
	$w = 0$	$w = 1$	$w = 0$	$w = 1$
A	B	C	0	0
B	D	–	0	–
C	F	E	0	1
D	B	G	0	0
E	F	C	0	1
F	E	D	0	1
G	F	–	0	–

Figure 8.59 Incompletely specified state table for Example 8.7

Tabela com estados não definidos



Present state	Next state		Output *z*	
	w = 0	*w* = 1	*w* = 0	*w* = 1
A	B	C	0	0
B	D	–	0	–
C	F	E	0	1
D	B	G	0	0
E	F	C	0	1
F	E	D	0	1
G	F	–	0	–

P1 = (ABCDEFG)

P2 = (AD) (BG) (CEF) ➔ PRÓXIMO ESTADO 0 (BB) (DF) (FFE)
➔ PRÓXIMO ESTADO 1 (CG) (--) (ECD)

P3 = (A) (D) (BG) (CE) (F) ➔ PRÓXIMO ESTADO 0 (B) (B) (DF) (FF) (E)
➔ PRÓXIMO ESTADO 1 (C) (G) (- -) (EC) (D)

P4 = (A) (D) (B) (G) (CE) (F) ➔ PRÓXIMO ESTADO 0 (B) (B) (D) (F) (FF) (E)
➔ PRÓXIMO ESTADO 1 (C) (G) (-) (-) (EC) (D)

P5 = (A) (D) (B) (G) (CE) (F) ➔ FIM

Figure 8.59 Incompletely specified state table for Example 8.7

Contador

Projete um contador módulo 8, síncrono que conta na borda de subida, usando FF tipo D, que tenha uma entrada w, que controla a contagem (w = 0 congela e w = 1 permite a contagem). Para reiniciar a contagem, um sinal de reset faz com que ele volte a zero. MÁQUINA MOORE

Contador

Projete um contador módulo 8, síncrono que conta na borda de subida, usando FF tipo D, que tenha uma entrada w, que controla a contagem (w = 0 congela e w = 1 permite a contagem). Para reiniciar a contagem, um sinal de reset faz com que ele volte a zero. MOORE

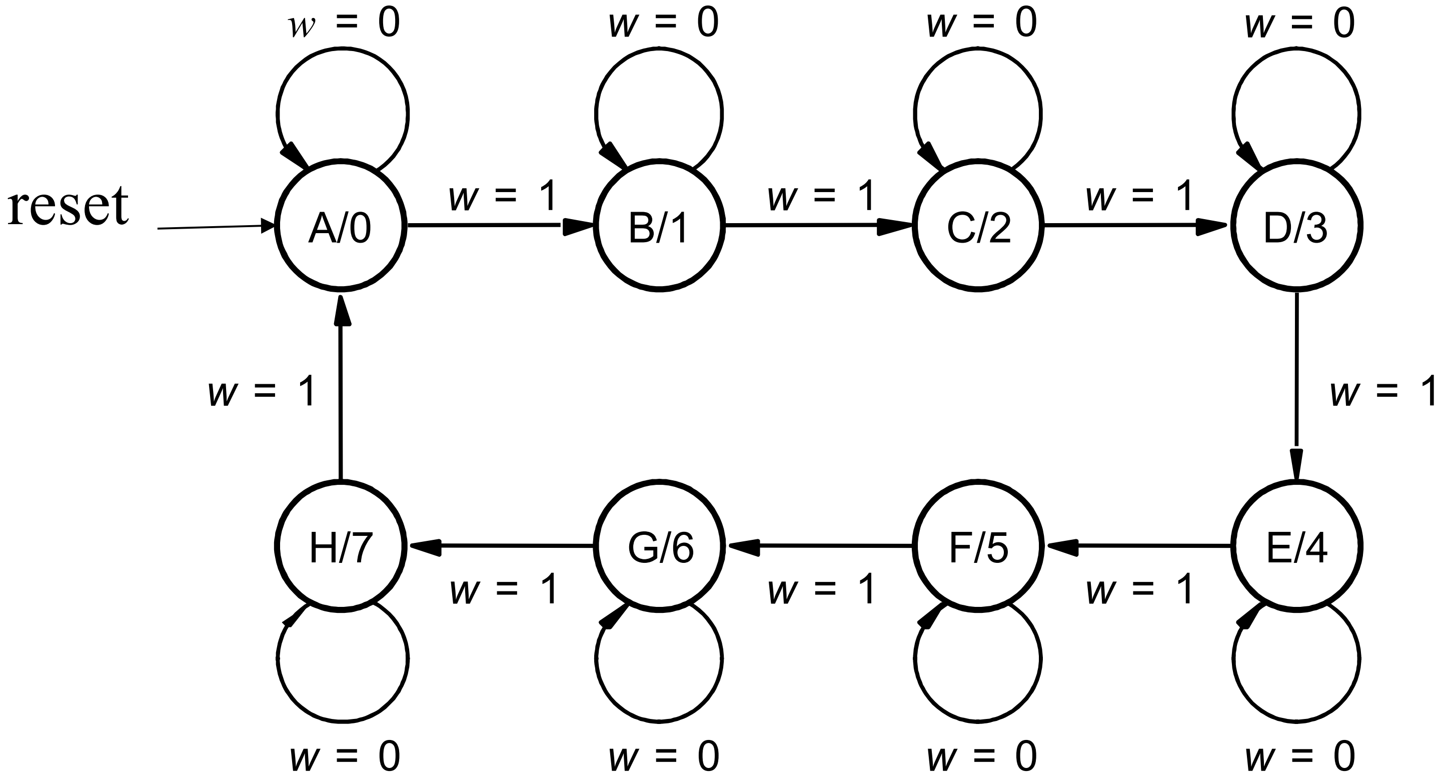


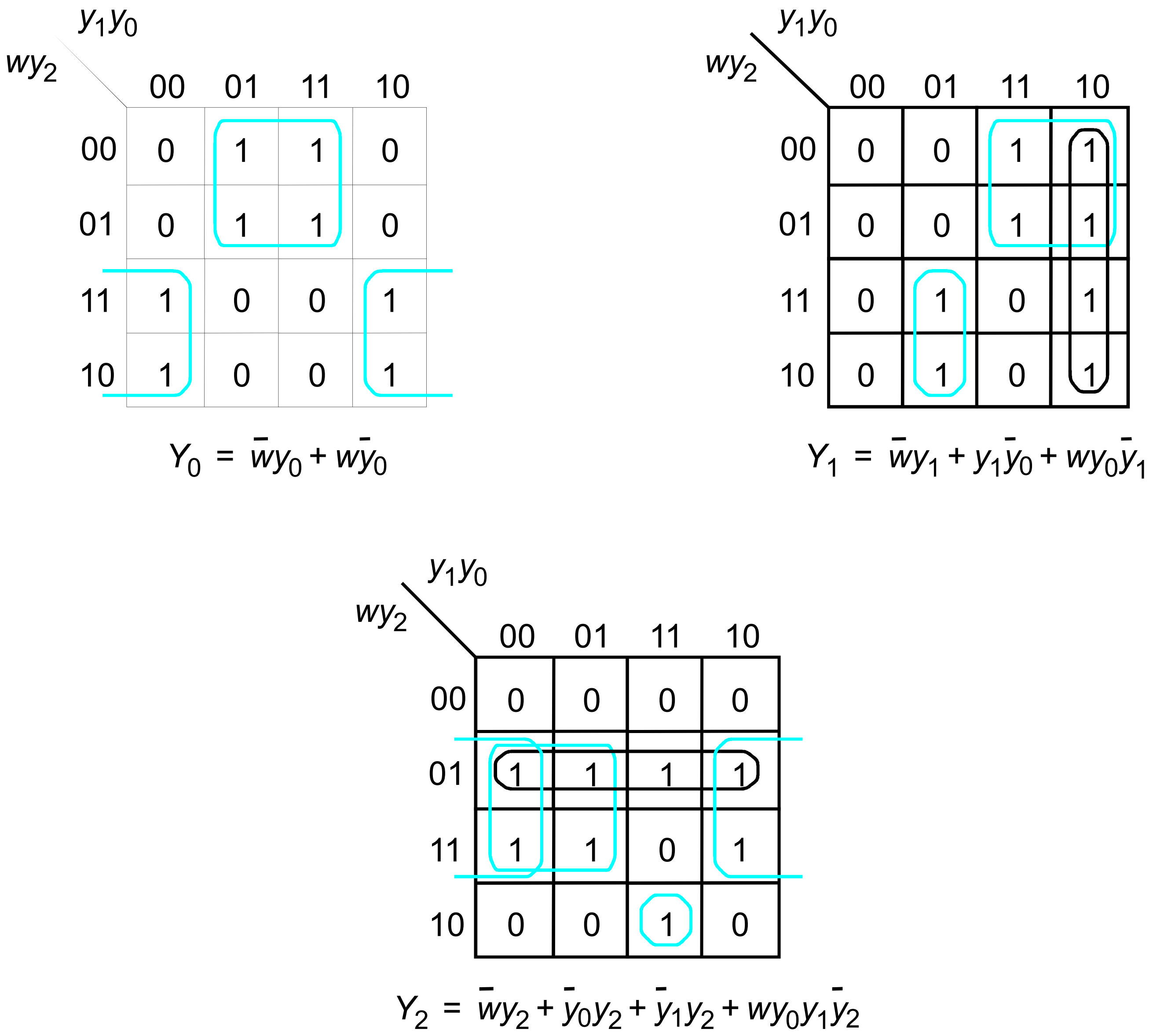
Figure 8.60 State diagram for a counter

Present state	Next state		Output
	$w = 0$	$w = 1$	
A	A	B	0
B	B	C	1
C	C	D	2
D	D	E	3
E	E	F	4
F	F	G	5
G	G	H	6
H	H	A	7

Figure 8.61 State table for the counter

	Present state	Next state		Count
		$w = 0$	$w = 1$	
	$y_2y_1y_0$	$Y_2Y_1Y_0$	$Y_2Y_1Y_0$	$z_2z_1z_0$
A	000	000	001	000
B	001	001	010	001
C	010	010	011	010
D	011	011	100	011
E	100	100	101	100
F	101	101	110	101
G	110	110	111	110
H	111	111	000	111

Figure 8.62 State-assigned table for the counter



wy_2 \ y_1y_0	00	01	11	10
00	0	1	1	0
01	0	1	1	0
11	1	0	0	1
10	1	0	0	1

$Y_0 = \bar{w}y_0 + w\bar{y}_0$

wy_2 \ y_1y_0	00	01	11	10
00	0	0	1	1
01	0	0	1	1
11	0	1	0	1
10	0	1	0	1

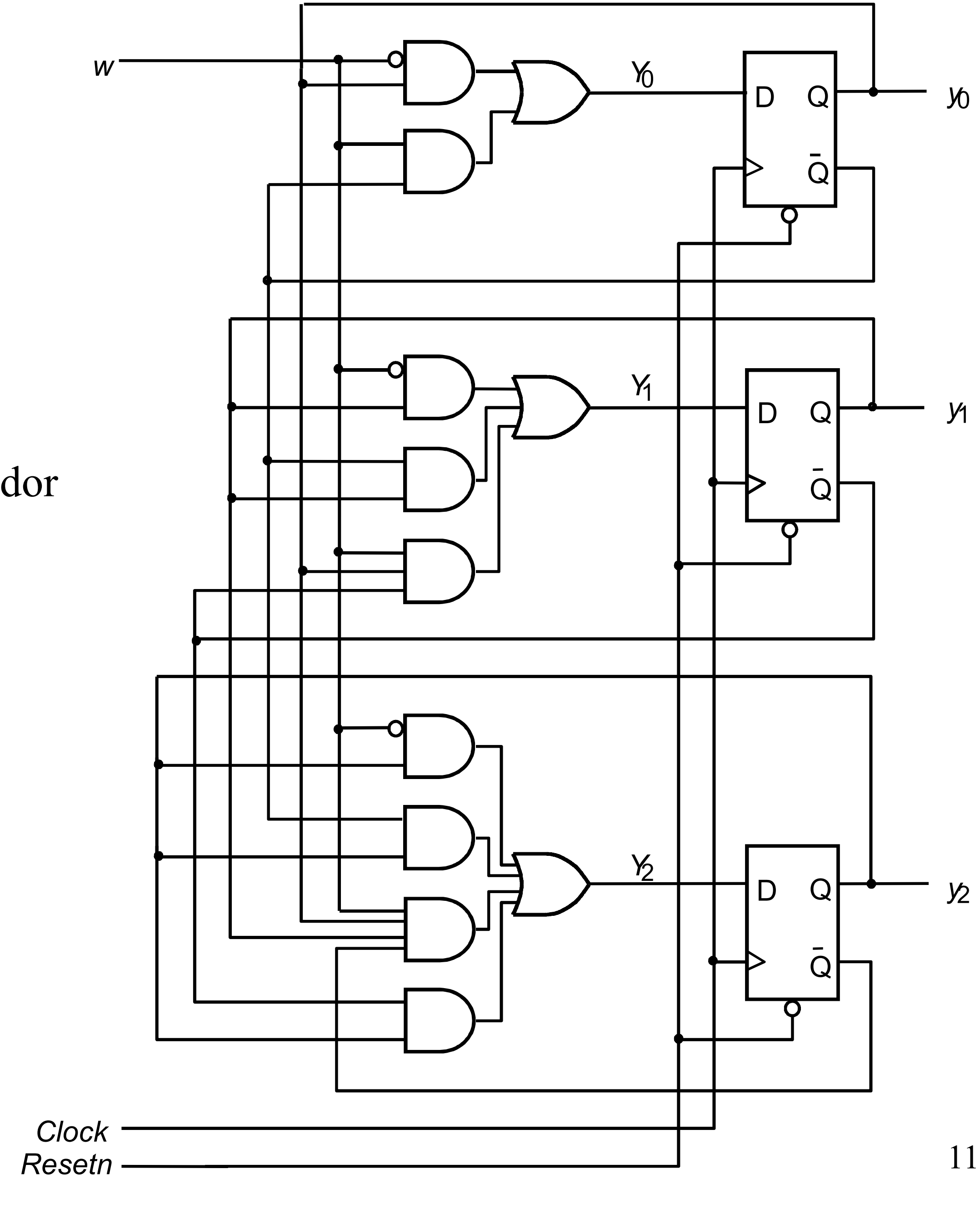
$Y_1 = \bar{w}y_1 + y_1\bar{y}_0 + wy_0\bar{y}_1$

wy_2 \ y_1y_0	00	01	11	10
00	0	0	0	0
01	1	1	1	1
11	1	1	0	1
10	0	0	1	0

$Y_2 = \bar{w}y_2 + \bar{y}_0y_2 + \bar{y}_1y_2 + wy_0y_1\bar{y}_2$

Figure 8.63 Karnaugh maps for D flip-flops for the counter

Diagrama do circuito do contador

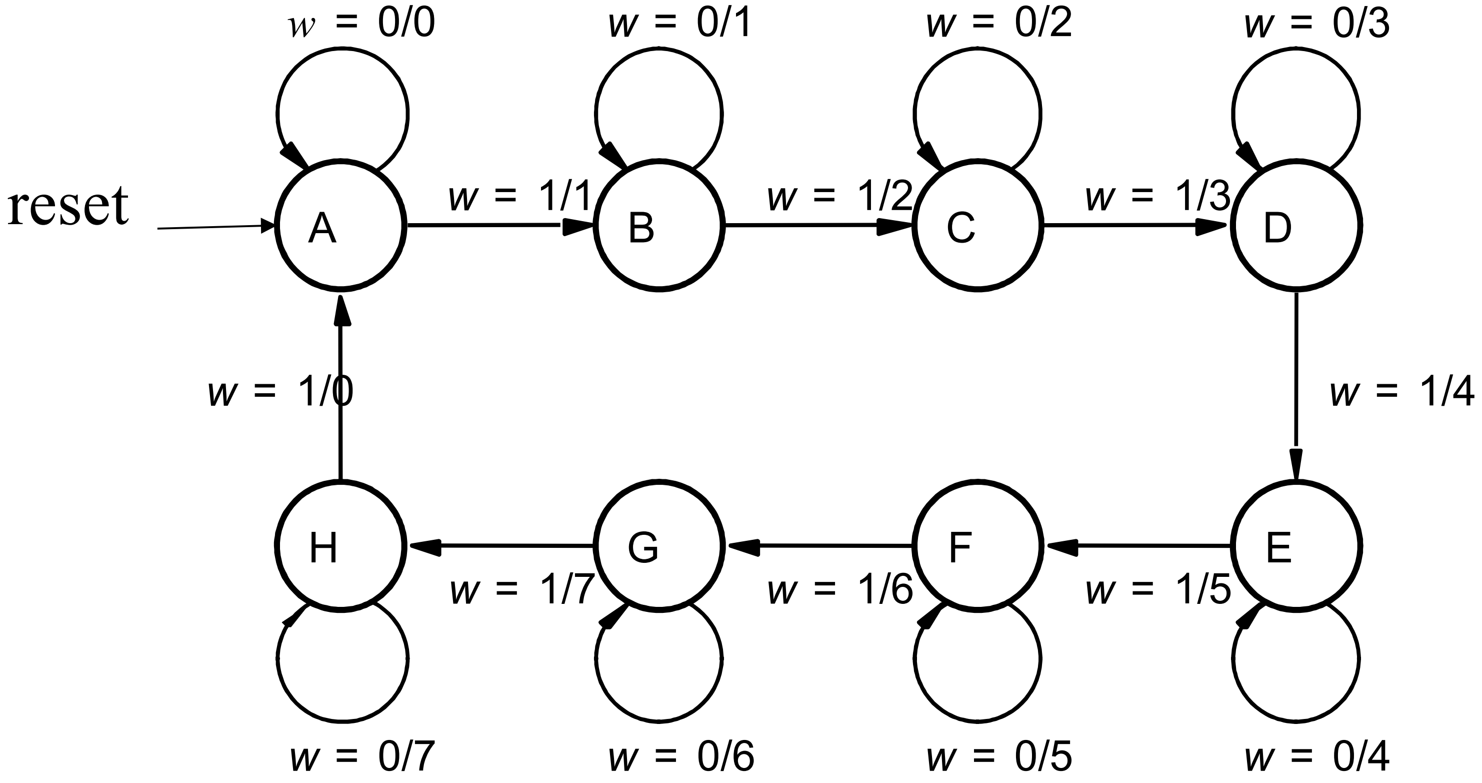


Contador

Projete um contador módulo 8, síncrono que conta na borda de subida, usando FF tipo D, que tenha uma entrada w, que controla a contagem (w = 0 congela e w = 1 permite a contagem). Para reiniciar a contagem, um sinal de reset faz com que ele volte a zero. MEALY

Contador

Projete um contador módulo 8, síncrono que conta na borda de subida, usando FF tipo D, que tenha uma entrada w, que controla a contagem (w = 0 congela e w = 1 permite a contagem). Para reiniciar a contagem, um sinal de reset faz com que ele volte a zero. MEALY



	Present state $y_2y_1y_0$	Next state	
		$w = 0$	$w = 1$
		$Y_2Y_1Y_0$	$Y_2Y_1Y_0$
A	000	000	001
B	001	001	010
C	010	010	011
D	011	011	100
E	100	100	101
F	101	101	110
G	110	110	111
H	111	111	000

MEALY

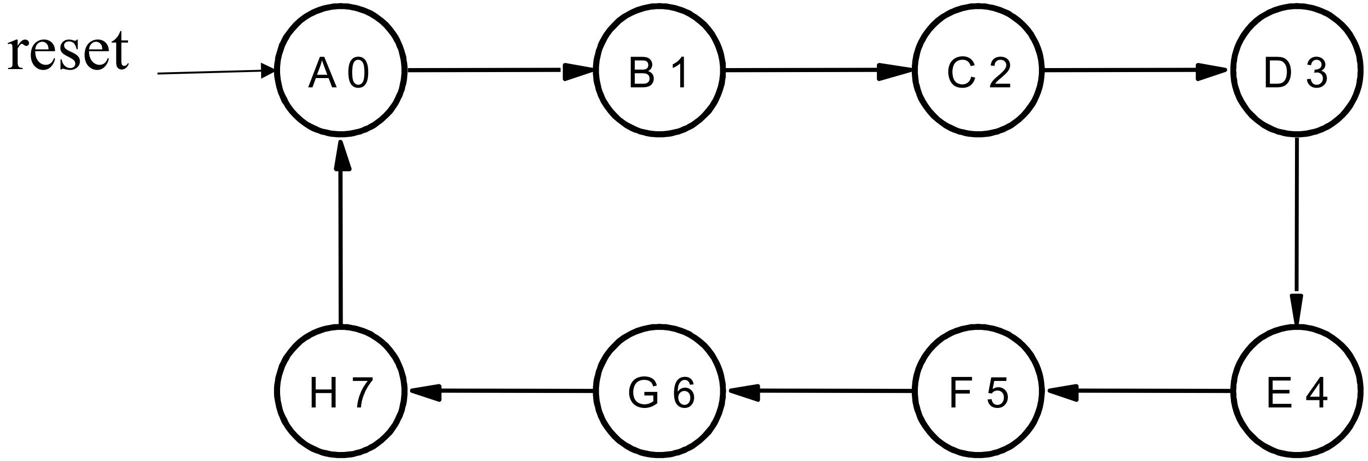
z2z1z0

w=0			w=1		
z2	**z1**	**z0**	**z2**	**z1**	**z0**
0	0	0	0	0	1
0	0	1	0	1	0
0	1	0	0	1	1
0	1	1	1	0	0
1	0	0	1	0	1
1	0	1	1	1	0
1	1	0	1	1	1
1	1	1	0	0	0

Figure 8.62 State-assigned table for the counter

Contador

Projete um contador módulo 8, síncrono que conta na borda de subida, usando FF tipo D. Para reiniciar a contagem, um sinal de reset faz com que ele volte a zero. MOORE e MEALY



Projete um contador módulo 8, síncrono que conta na borda de subida, usando FF tipo D. Para reiniciar a contagem, um sinal de reset faz com que ele volte a zero. MOORE e MEALY

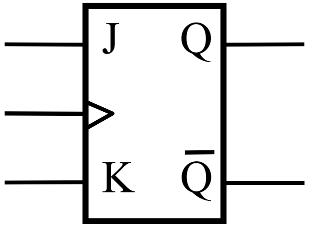
	Present state $y_2y_1y_0$	Next state	Count $z_2z_1z_0$
		$Y_2Y_1Y_0$	
A	000	001	000
B	001	010	001
C	010	011	010
D	011	100	011
E	100	101	100
F	101	110	101
G	110	111	110
H	111	000	111

FF JK

Quando J= 0, K= d e o estado é 0 → mantém estado em 0
Quando J = 1, K = d e o estado é 0 → muda estado para 1
Quando J = d, K = 0 e o estado é 1 → mantém estado para 1
Quando J = d, K = 1 e o estado é 1 → muda estado para 0

J	K	Q (t+1)
0	0	Q (t)
0	1	0
1	0	1
1	1	$\overline{Q}$ (t)

(b) Tabela Verdade



(c) Símbolo Gráfico

	Present state	Flip-flop inputs								Count
		$w = 0$				$w = 1$				
	$y_2y_1y_0$	$Y_2Y_1Y_0$	J_2K_2	J_1K_1	J_0K_0	$Y_2Y_1Y_0$	J_2K_2	J_1K_1	J_0K_0	$z_2z_1z_0$
A	000	000	0d	0d	0d	001	0d	0d	1d	000
B	001	001	0d	0d	d0	010	0d	1d	d1	001
C	010	010	0d	d0	0d	011	0d	d0	1d	010
D	011	011	0d	d0	d0	100	1d	d1	d1	011
E	100	100	d0	0d	0d	101	d0	0d	1d	100
F	101	101	d0	0d	d0	110	d0	1d	d1	101
G	110	110	d0	d0	0d	111	d0	d0	1d	110
H	111	111	d0	d0	d0	000	d1	d1	d1	111

Figure 8.65 Excitation table for the counter with JK flip-flops

Mapa Karnaugh - contador

wy_2 \ y_1y_0	00	01	11	10
00	0	d	d	0
01	0	d	d	0
11	1	d	d	1
10	1	d	d	1

$J_0 = w$

wy_2 \ y_1y_0	00	01	11	10
00	d	0	0	d
01	d	0	0	d
11	d	1	1	d
10	d	1	1	d

$K_0 = w$

wy_2 \ y_1y_0	00	01	11	10
00	0	0	0	0
01	d	d	d	d
11	d	d	d	d
10	0	0	1	0

$J_2 = wy_0y_1$

wy_2 \ y_1y_0	00	01	11	10
00	0	0	d	d
01	0	0	d	d
11	0	1	d	d
10	0	1	d	d

$J_1 = wy_0$

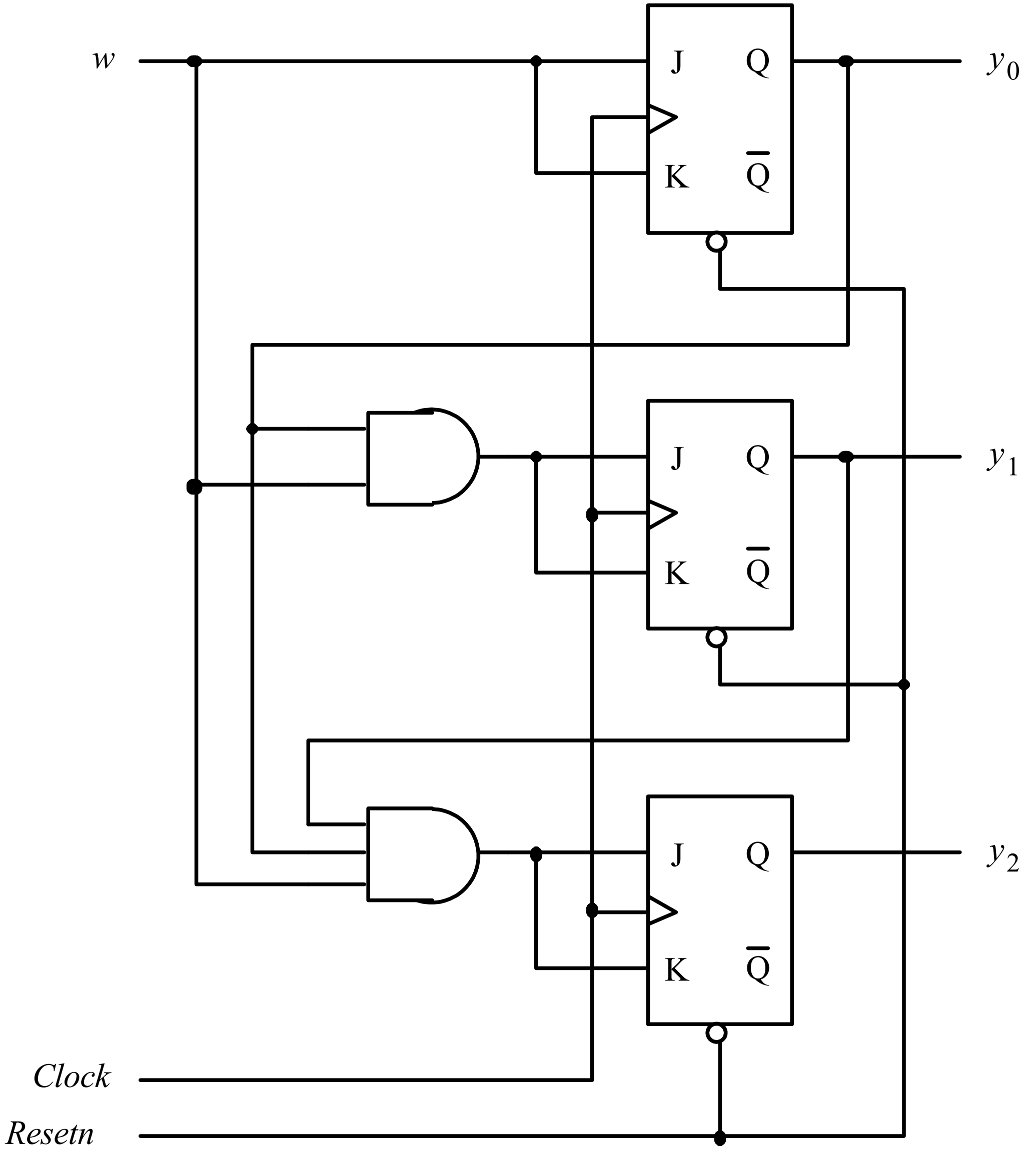
wy_2 \ y_1y_0	00	01	11	10
00	d	d	0	0
01	d	d	0	0
11	d	d	1	0
10	d	d	1	0

$K_1 = wy_0$

wy_2 \ y_1y_0	00	01	11	10
00	d	d	d	d
01	0	0	0	0
11	0	0	1	0
10	d	d	d	d

$K_2 = wy_0y_1$

Diagrama do circuito usando flip-flop JK



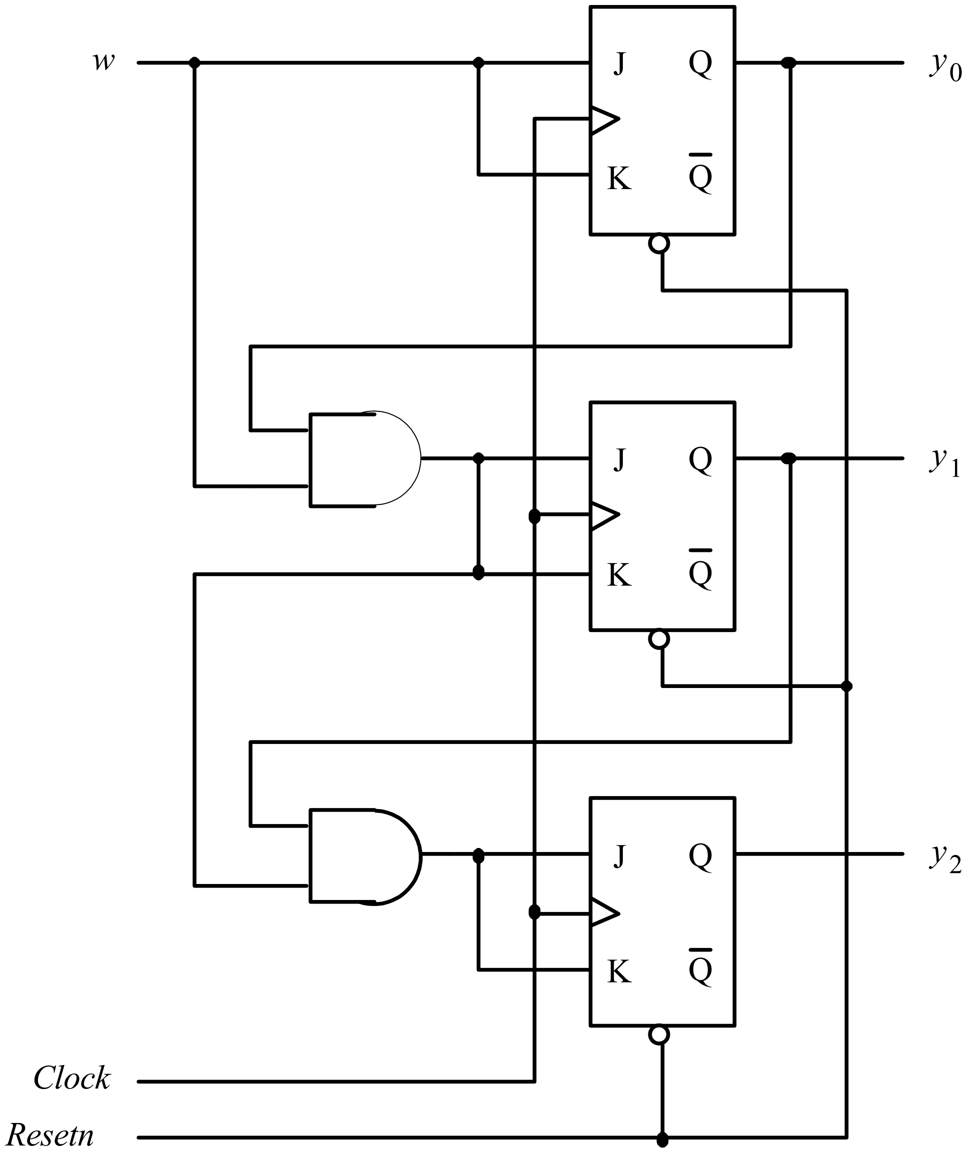


Figure 8.68 Factored-form implementation of the counter

Present state	Next state	Output $z_2z_1z_0$
A	B	000
B	C	100
C	D	010
D	E	110
E	F	001
F	G	101
G	H	011
H	A	111

Figure 8.69 State table for the counterlike example

Present state $y_2y_1y_0$	Next state $Y_2Y_1Y_0$	Output $z_2z_1z_0$
000	100	000
100	010	100
010	110	010
110	001	110
001	101	001
101	011	101
011	111	011
111	000	111

Figure 8.70 State-assigned table

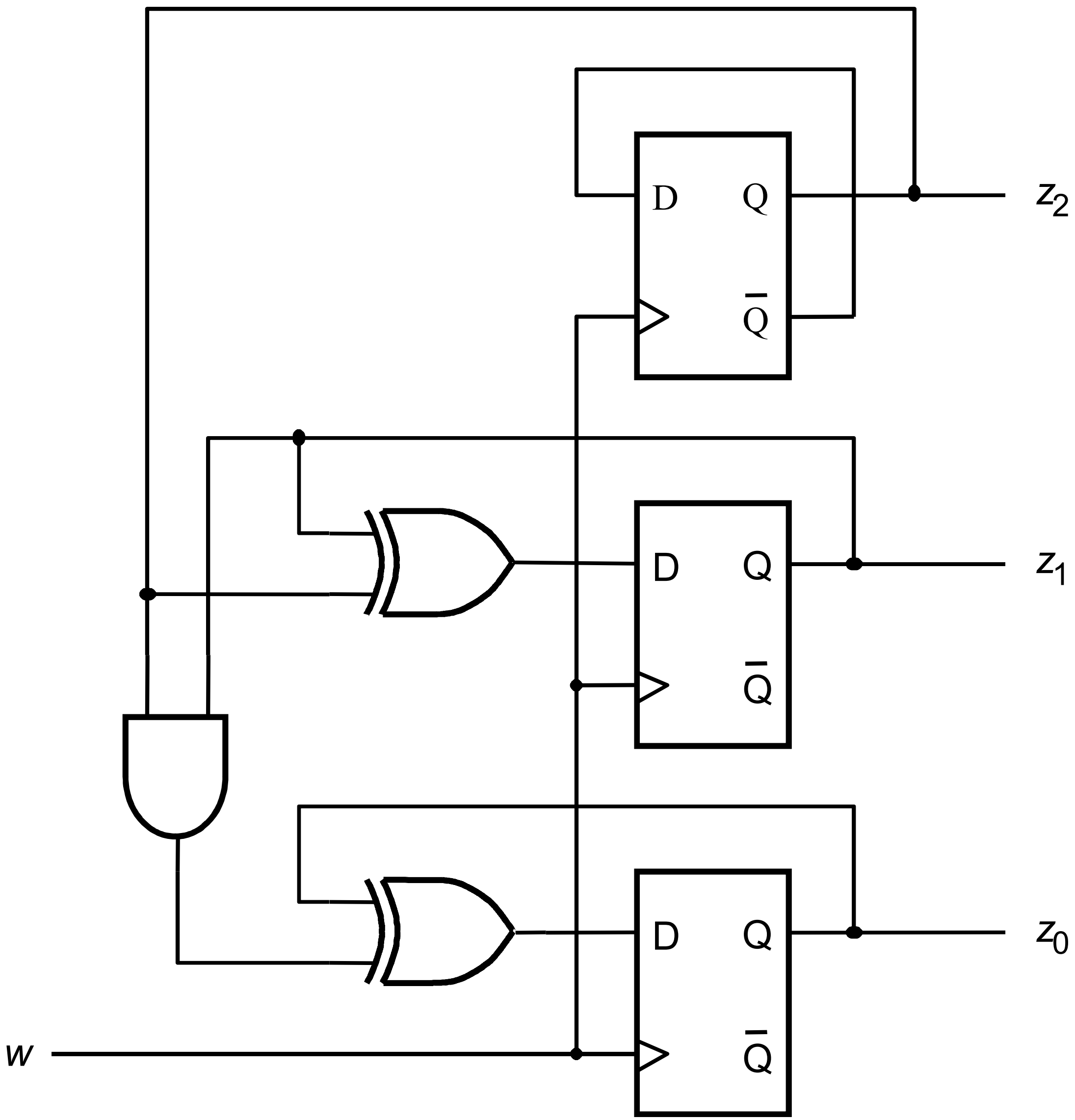


Figure 8.71 Circuit for the counterlike example

Projete um circuito que gerencie o acesso a um recurso de um computador. São 3 dispositivos, d1, d2 e d3, sendo que d1 tem maior prioridade que d2 e d3 e d2 tem maior prioridade que d3

MOORE

Entradas - requisições r1r2r3

Saídas garantia de uso g1g2g3

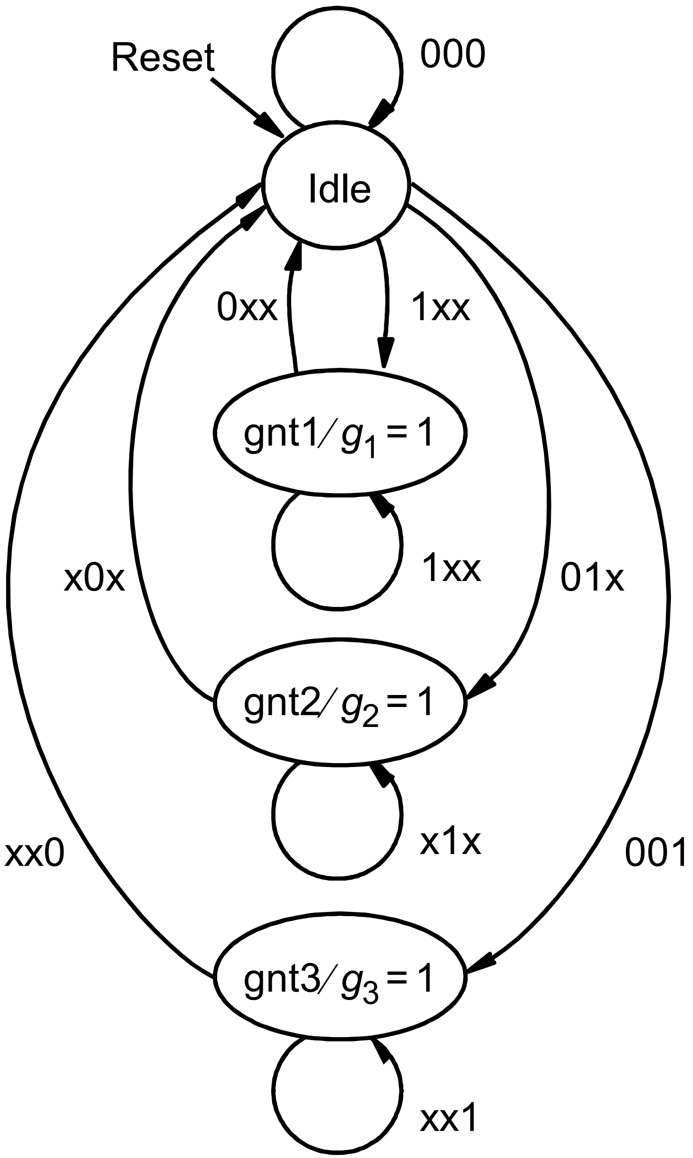


Figure 8.72 State diagram for the arbiter

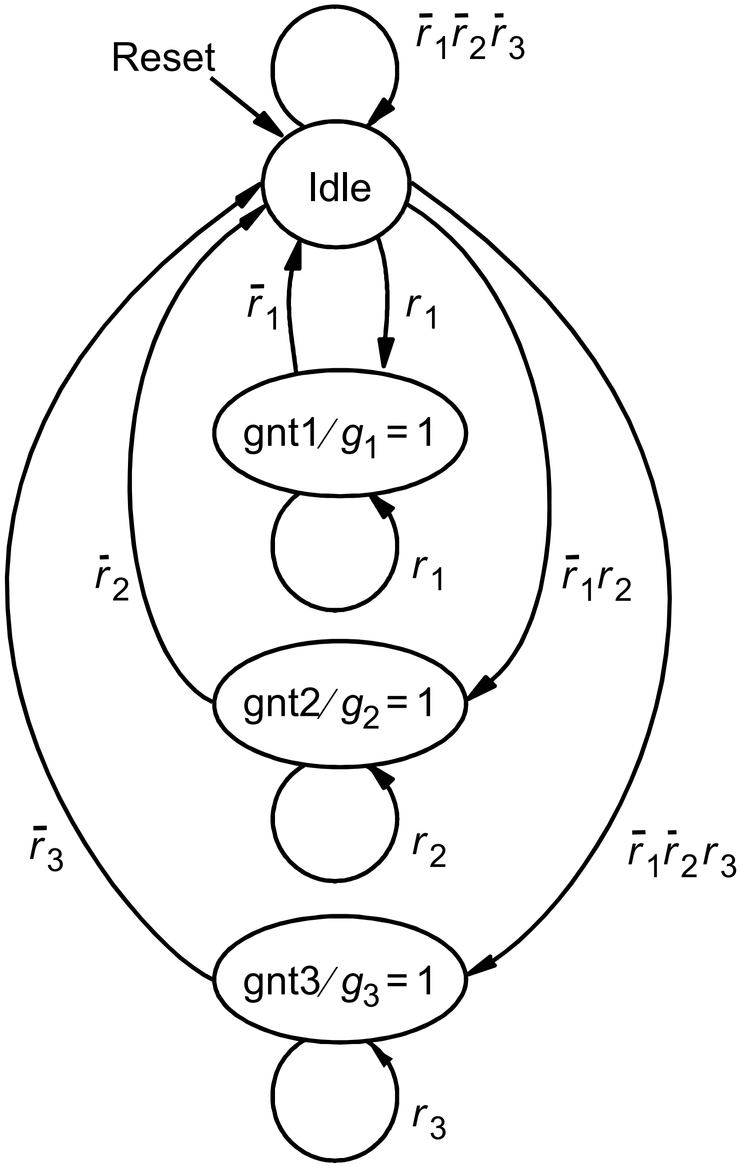


Figure 8.73 Alternative style of state diagram for the arbiter

Projete um circuito
que gerencie o acesso
a um recurso de um computador.
São 3 dispositivos, d1, d2 e d3,
sendo que d1 tem maior
prioridade que d2 e d3 e d2
tem maior prioridade que d3
MEALY

Entradas - requisições
r1r2r3

Saídas garantia de uso
g1g2g3

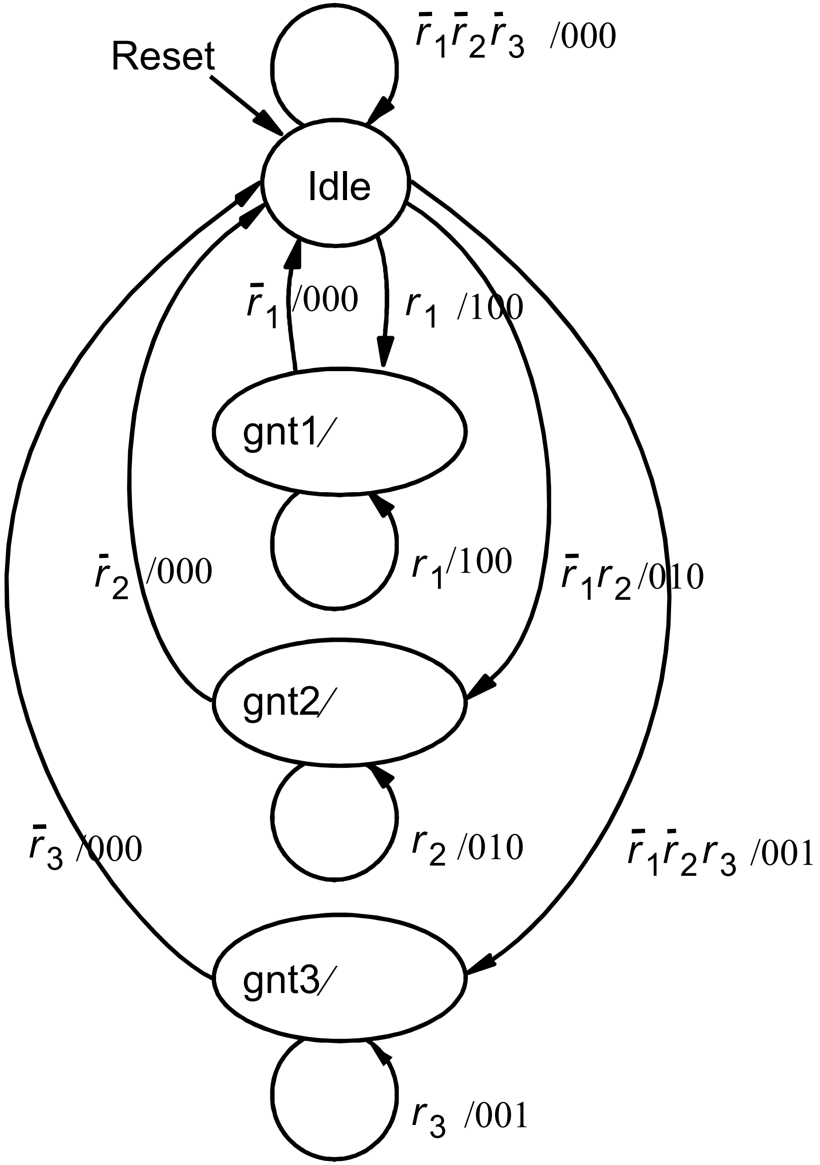


Figure 8.73 Alternative style of state diagram for the arbiter

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

ENTITY arbiter IS
    PORT (  Clock, Resetn      : IN       STD_LOGIC ;
            r                  : IN       STD_LOGIC_VECTOR(1 TO 3) ;
            g                  : OUT      STD_LOGIC_VECTOR(1 TO 3) ) ;
END arbiter ;

ARCHITECTURE Behavior OF arbiter IS
    TYPE State_type IS (Idle, gnt1, gnt2, gnt3) ;
    SIGNAL y : State_type ;
BEGIN
    PROCESS ( Resetn, Clock )
    BEGIN
        IF Resetn = '0' THEN y <= Idle ;
        ELSIF (Clock'EVENT AND Clock = '1') THEN
            CASE y IS
                WHEN Idle =>
                    IF r(1) = '1' THEN y <= gnt1 ;
                    ELSIF r(2) = '1' THEN y <= gnt2 ;
                    ELSIF r(3) = '1' THEN y <= gnt3 ;
                    ELSE y <= Idle ;
                    END IF ;
… con’t
```

Figure 8.74a VHDL code for the arbiter

```
            WHEN gnt1 =>
               IF r(1) = '1' THEN y <= gnt1 ;
               ELSE y <= Idle ;
               END IF ;
            WHEN gnt2 =>
               IF r(2) = '1' THEN y <= gnt2 ;
               ELSE y <= Idle ;
               END IF ;
            WHEN gnt3 =>
               IF r(3) = '1' THEN y <= gnt3 ;
               ELSE y <= Idle ;
               END IF ;
         END CASE ;
      END IF ;
   END PROCESS ;
   g(1) <= '1' WHEN y = gnt1 ELSE '0' ;
   g(2) <= '1' WHEN y = gnt2 ELSE '0' ;
   g(3) <= '1' WHEN y = gnt3 ELSE '0' ;
END Behavior ;
```

Figure 8.74b VHDL code for the arbiter (con’t)

```
    .

    .

    .
    PROCESS( y )
    BEGIN
        IF y = gnt1 THEN g(1) <= '1' ;
        ELSIF y = gnt2 THEN g(2) <= '1' ;
        ELSIF y = gnt3 THEN g(3) <= '1' ;
        END IF ;
    END PROCESS ;
END Behavior ;
```

Figure 8.75 Incorrect VHDL code for the grant signals

```
    .
    .
    .
    PROCESS( y )
    BEGIN
        g(1) <= '0' ;
        g(2) <= '0' ;
        g(3) <= '0' ;
        IF y = gnt1 THEN g(1) <= '1' ;
        ELSIF y = gnt2 THEN g(2) <= '1' ;
        ELSIF y = gnt3 THEN g(3) <= '1' ;
        END IF ;
    END PROCESS ;
END Behavior ;
```

Figure 8.76 Correct VHDL code for the grant signals

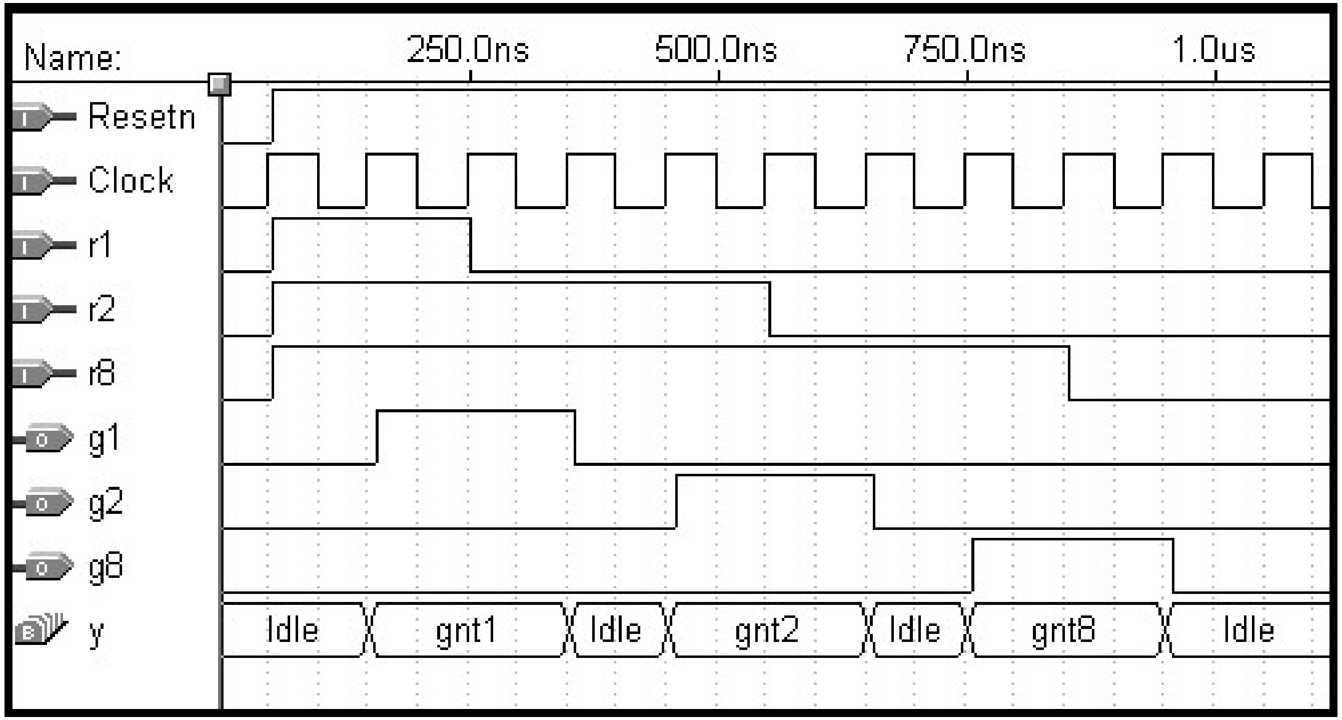
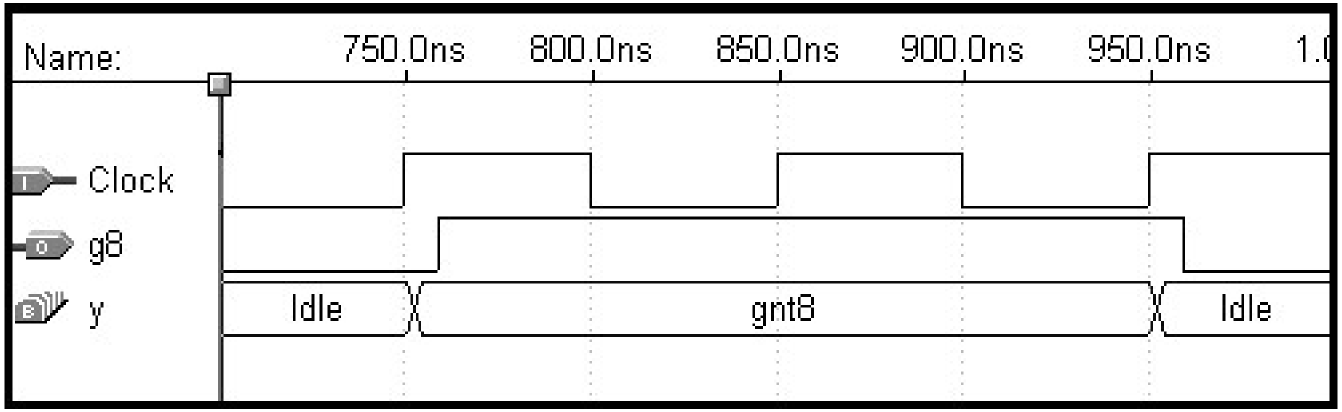
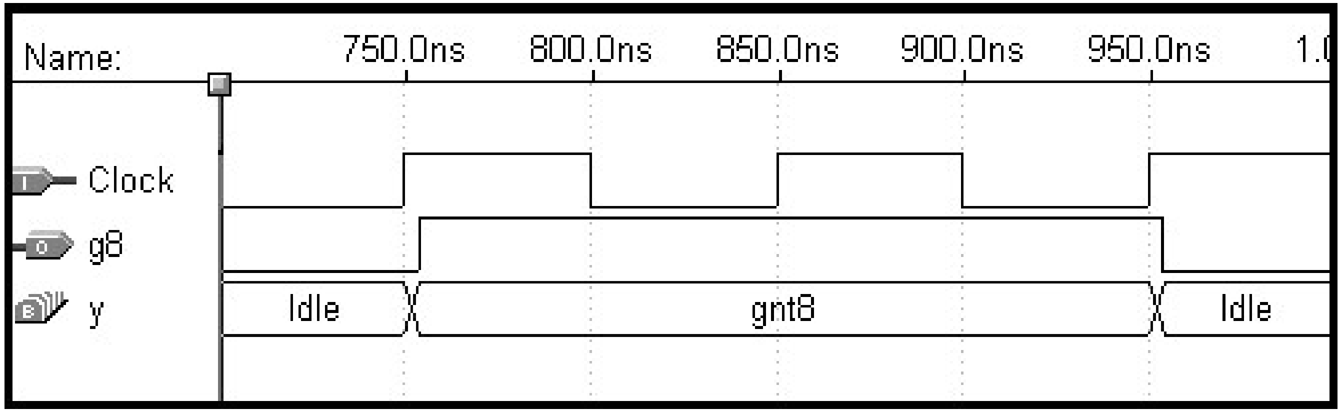


Figure 8.77 Simulation results for the arbiter circuit



a) Output delays using binary encoding



b) Output delays using one-hot encoding

Figure 8.78 Output delays in the arbiter circuit

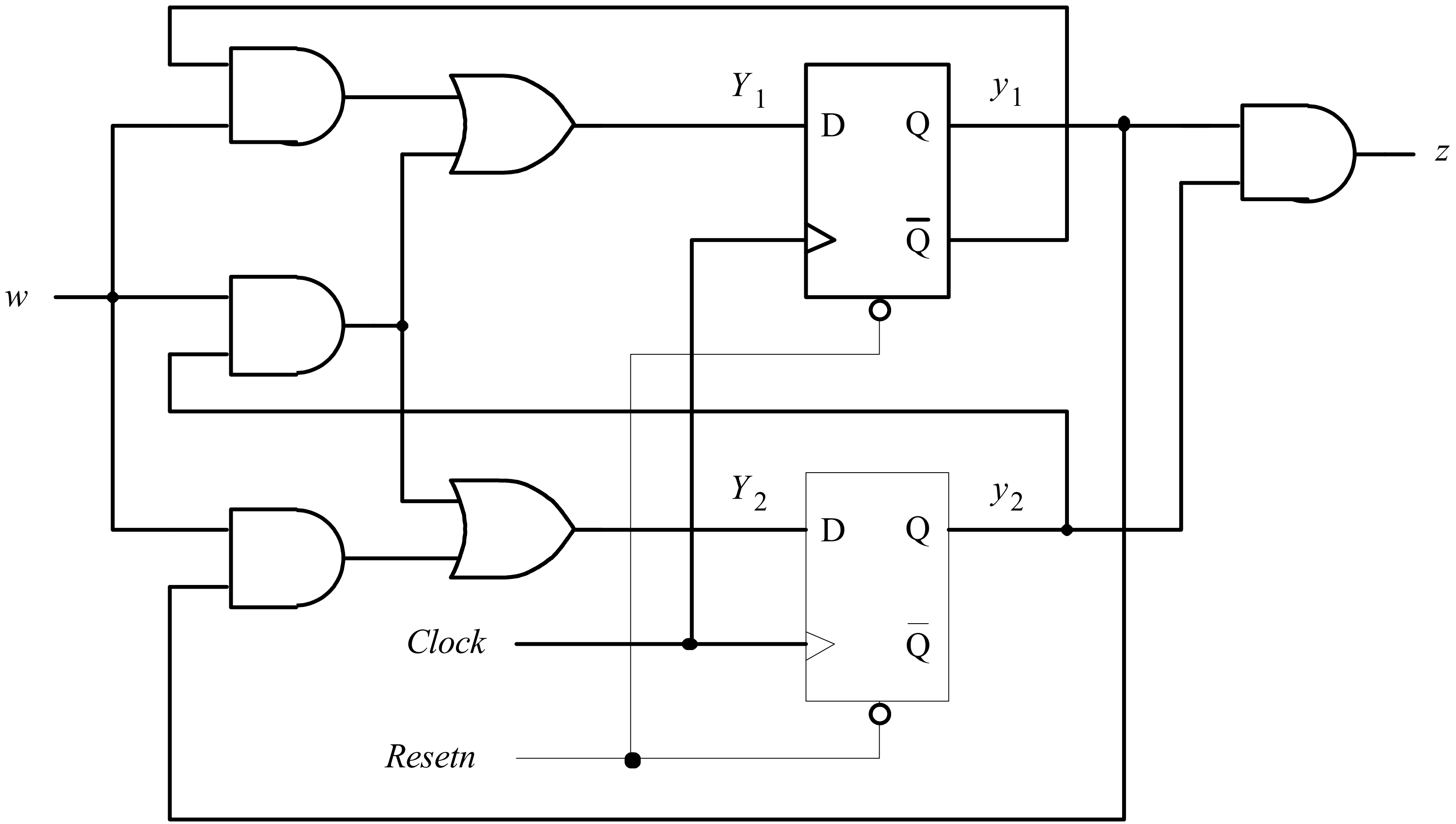


Figure 8.80 Circuit for Example 8.8

Present state y_2y_1	Next State		Output z
	w = 0	w = 1	
	Y_2Y_1	Y_2Y_1	
0 0	0 0	01	0
0 1	0 0	10	0
1 0	0 0	11	0
1 1	0 0	11	1

(a)State-assigned table

Present state	Next state		Output z
	w = 0	w = 1	
A	A	B	0
B	A	C	0
C	A	D	0
D	A	D	1

(b)State table

Figure 8.81 Tables for the circuit in Example 8.8

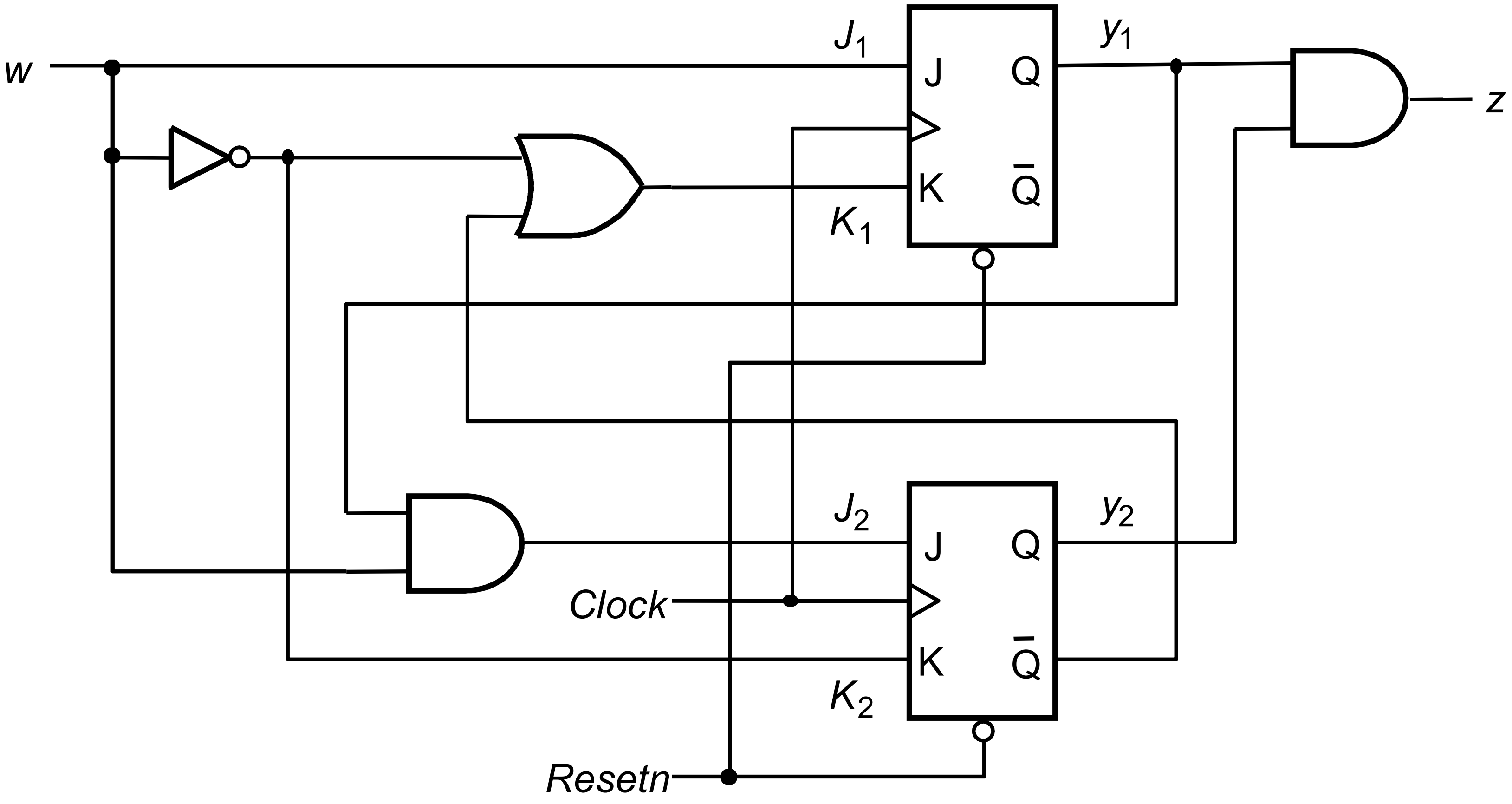


Figure 8.82 Circuit for Example 8.9

Present state	Flip-flop inputs				Output
	$w = 0$		$w = 1$		
y_2y_1	J_2K_2	J_1K_1	J_2K_2	J_1K_1	z
00	01	0 1	0 0	1 1	0
01	01	0 1	1 0	1 1	0
10	01	0 1	0 0	1 0	0
11	01	0 1	1 0	1 0	1

Figure 8.83 Excitation table

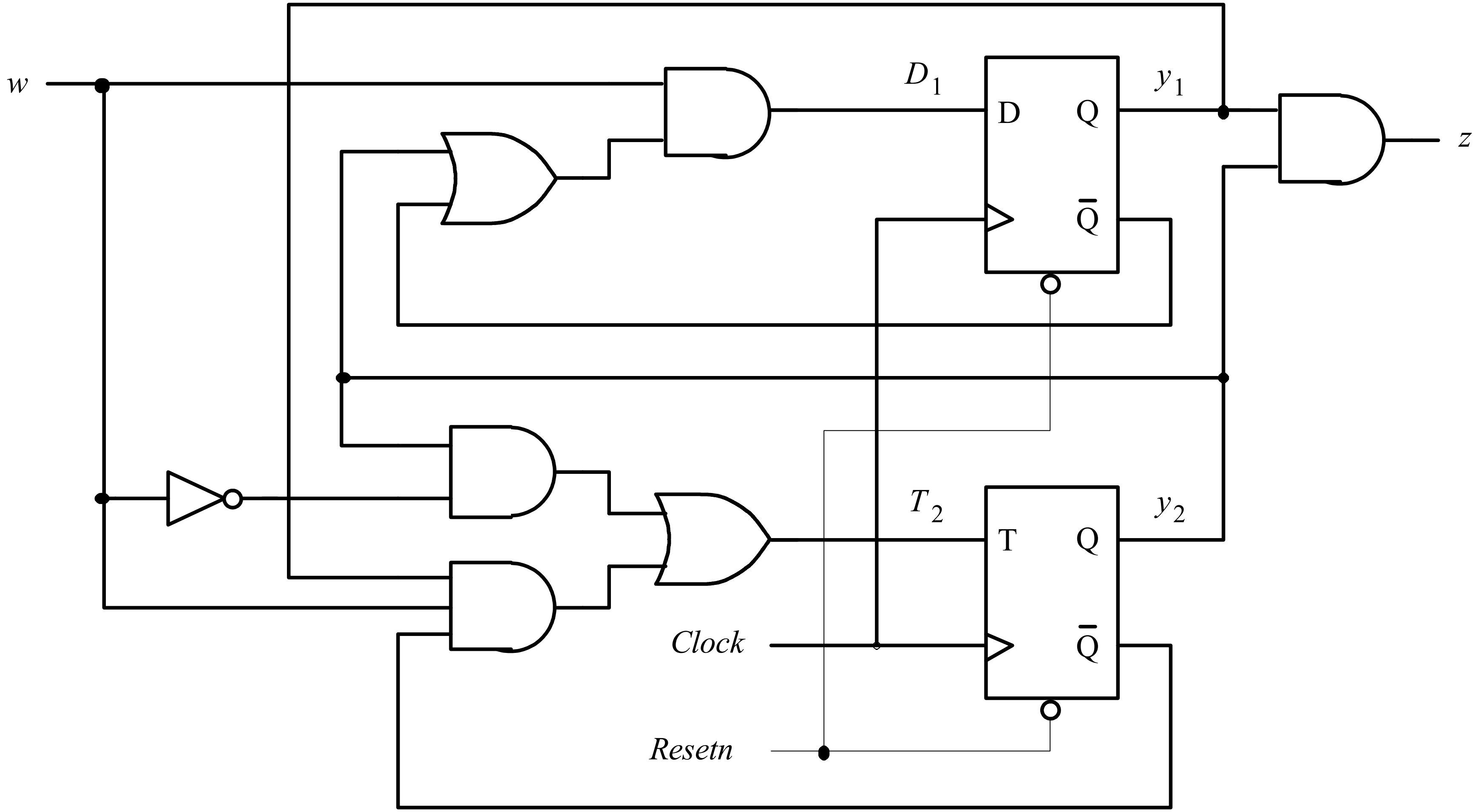
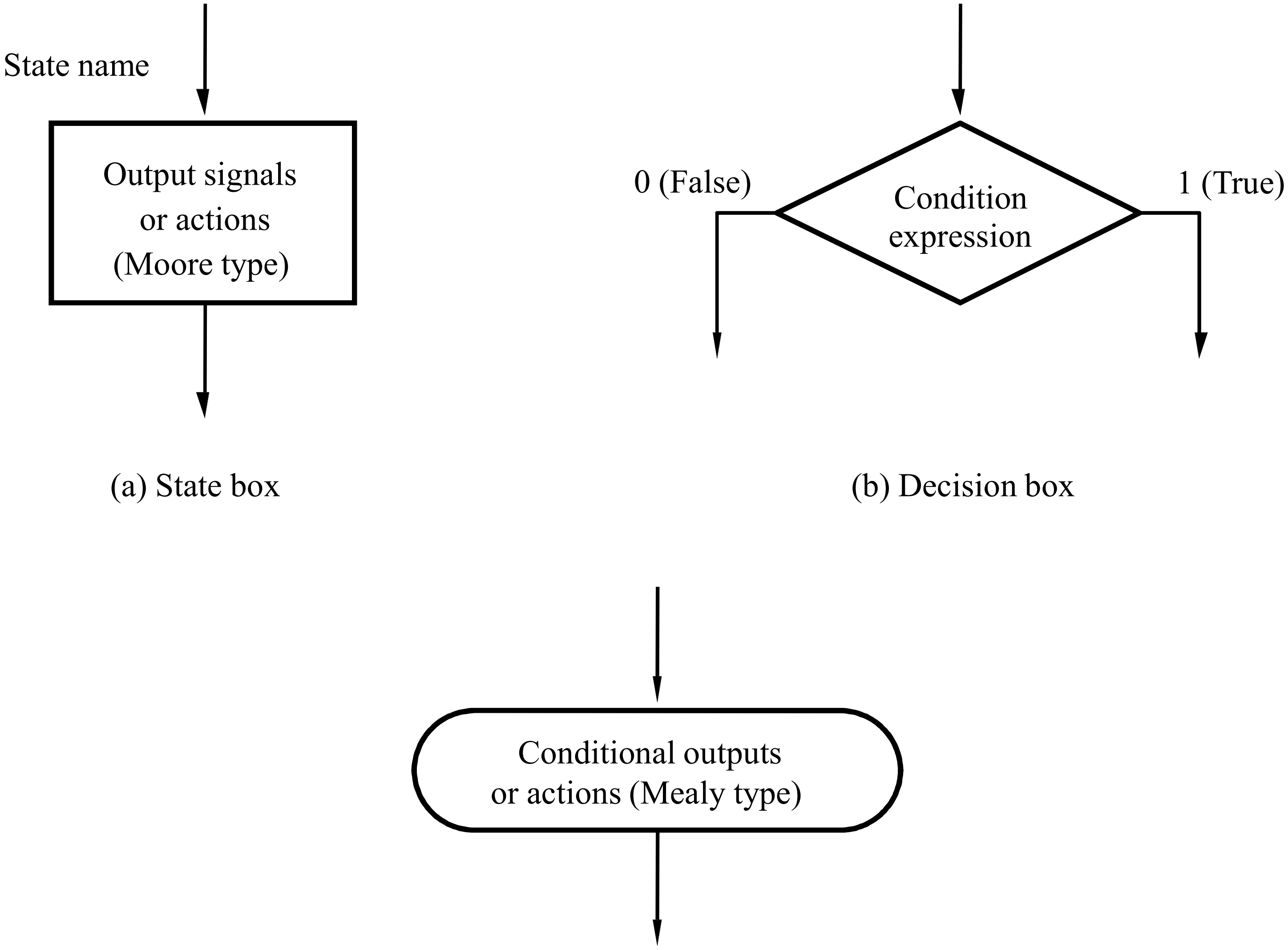


Figure 8.84 Circuit for Example 8.10

Present state	Flip-flop inputs		Output
	$w = 0$	$w = 1$	
y_2y_1	T_2D_1	T_2D_1	z
0 0	0 0	01	0
0 1	0 0	10	0
1 0	1 0	01	0
1 1	1 0	01	1

Figure 8.85 Excitation table



(a) State box

(b) Decision box

(c) Conditional output box

Figure 8.86 Elements used in ASM charts

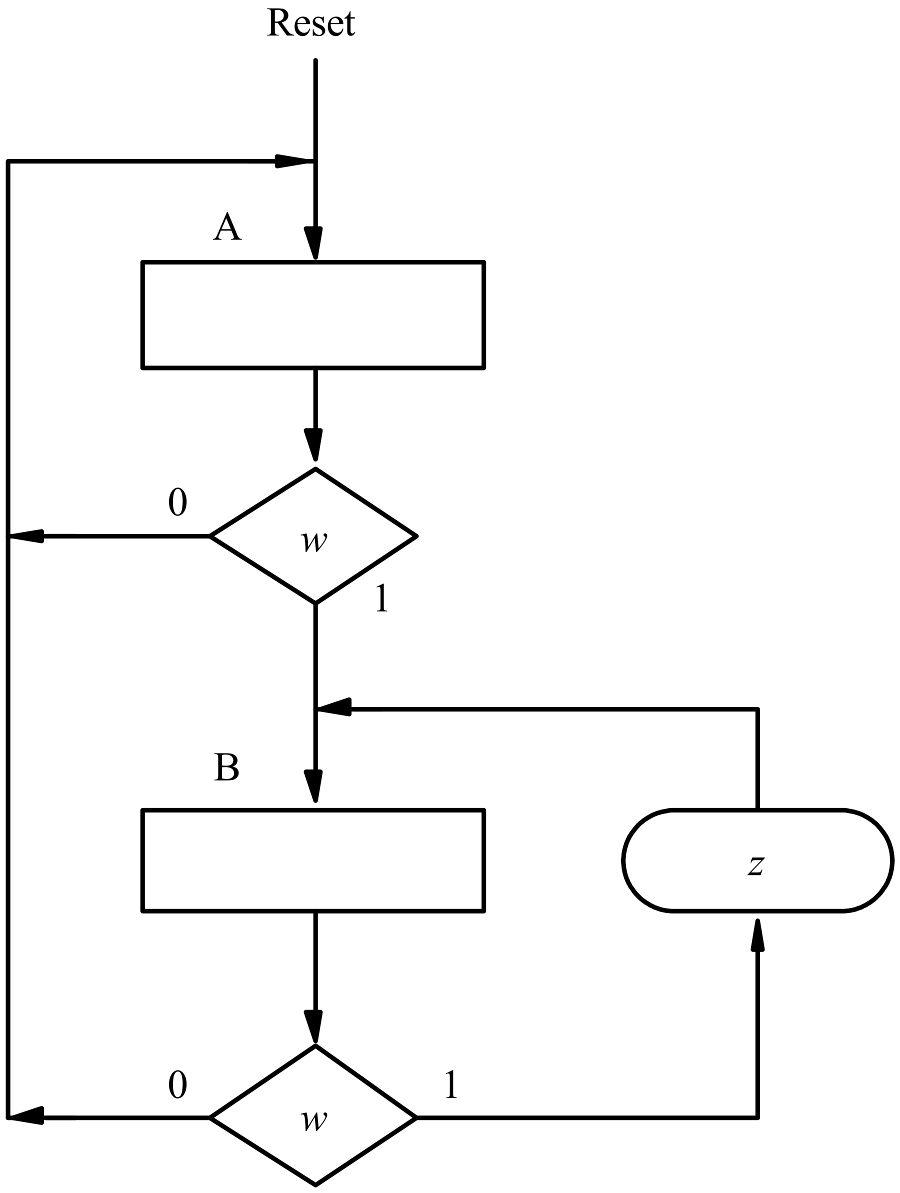


Figure 8.88 ASM chart for the FSM in Figure 8.23

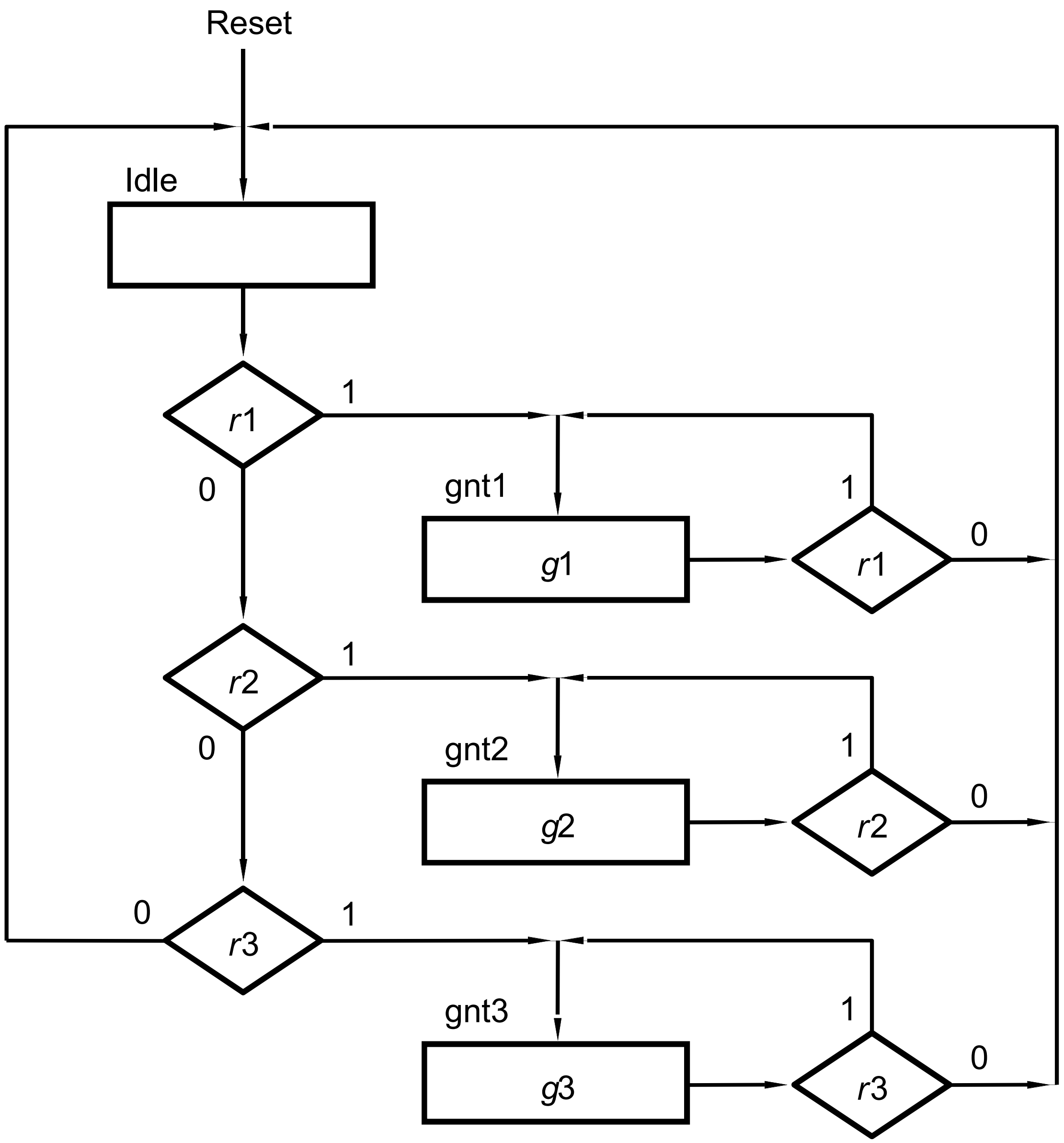


Figure 8.89 ASM chart for the arbiter

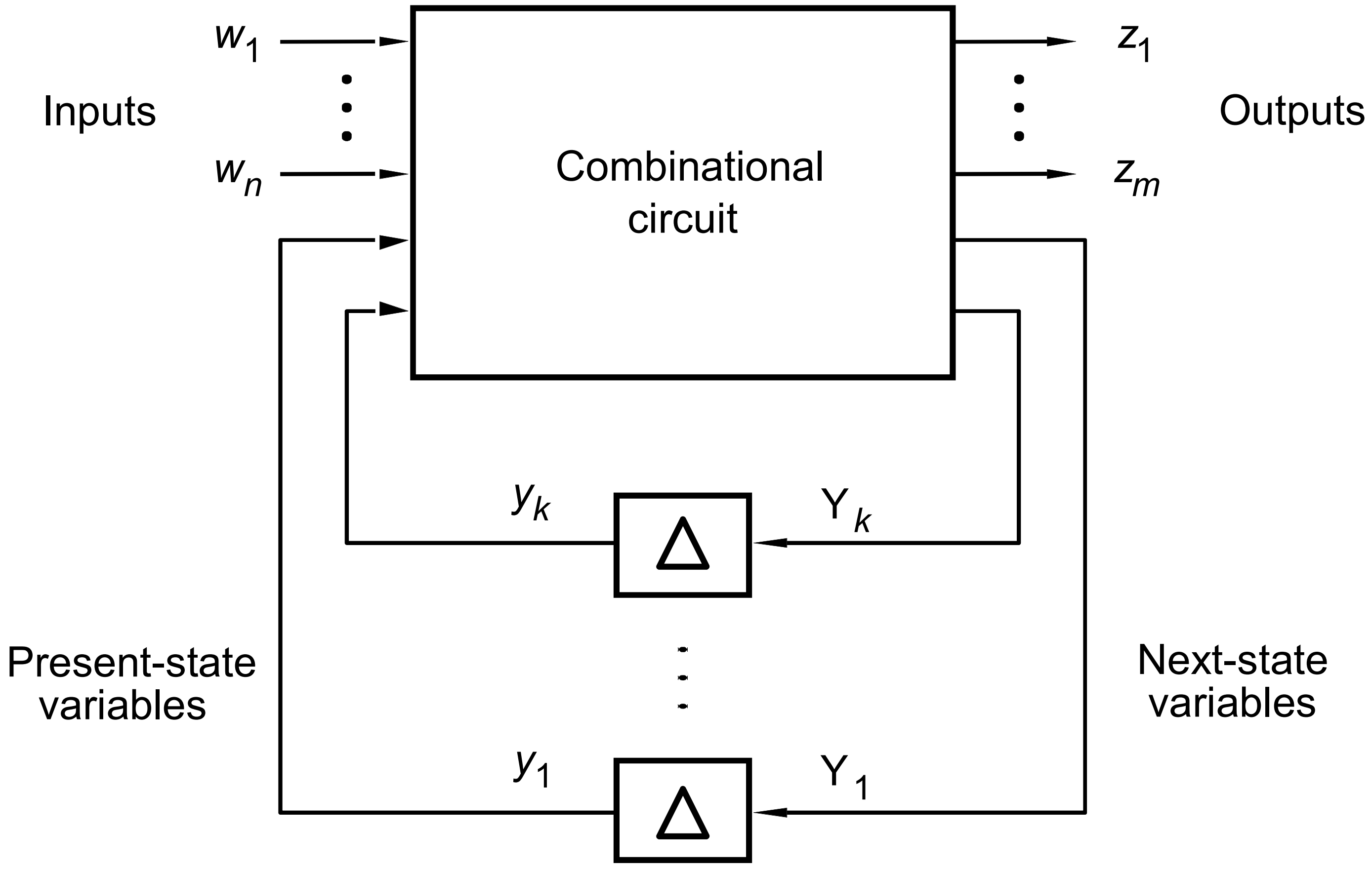


Figure 8.90 The general model for a sequential circuit

Present state	Next state		Output
	$w = 0$	$w = 1$	
y_2y_1	Y_2Y_1	Y_2Y_1	z
0 0	1 0	1 1	0
0 1	0 1	0 0	0
1 0	1 1	0 0	0
1 1	1 0	0 1	1

Figure P8.1 State-assigned table for problems 8.1 and 8.2

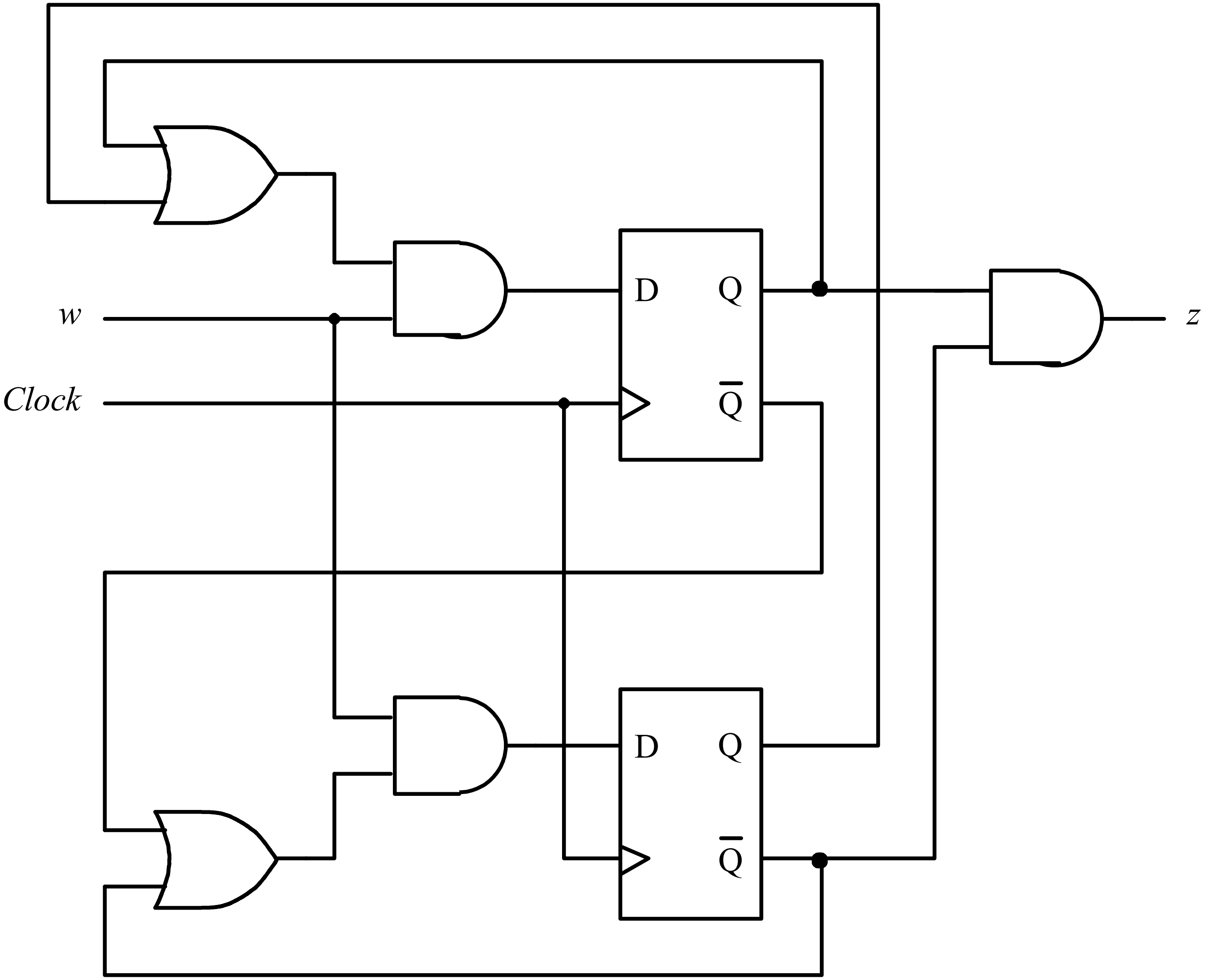
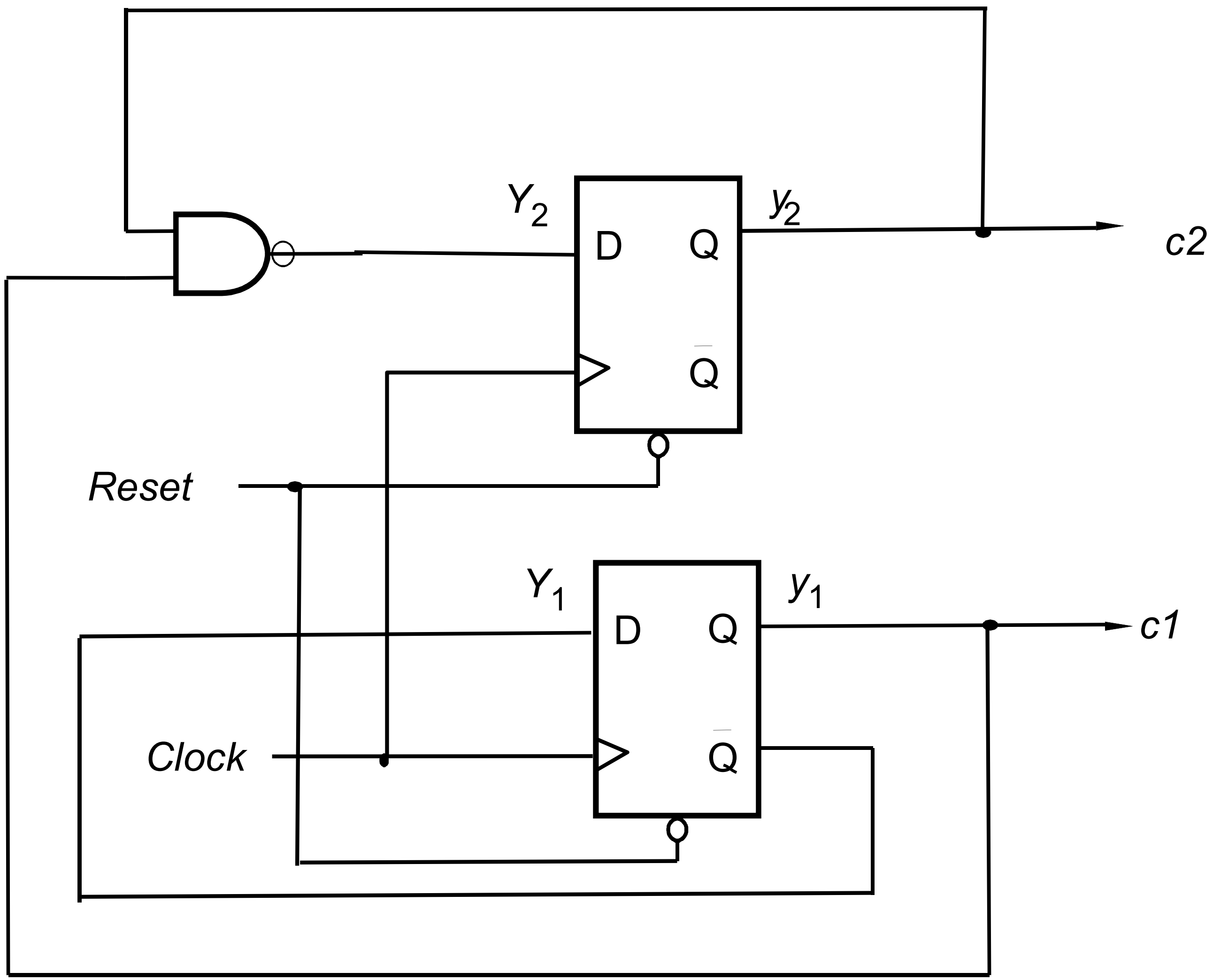


Figure P8.2 Circuit for problem 8.29


Y_2
D
Q
$\overline{Q}$
y_2
c2
Reset
Y_1
D
Q
$\overline{Q}$
y_1
c1
Clock