DISEÑO Y ANÁLISIS DE CIRCUITOS DIGITALES CON VHDL

ALFONSO URQUÍA, CARLA MARTÍN-VILLALBA

EDITORIAL UNED - 7101201GR01A01

Figuras y Tablas

Material de apoyo a la tutoría de la asignatura
Ingeniería de Computadores III
del Grado en Ingeniería Informática de la UNED

TEMA 1

Fundamentos del diseño del hardware digital

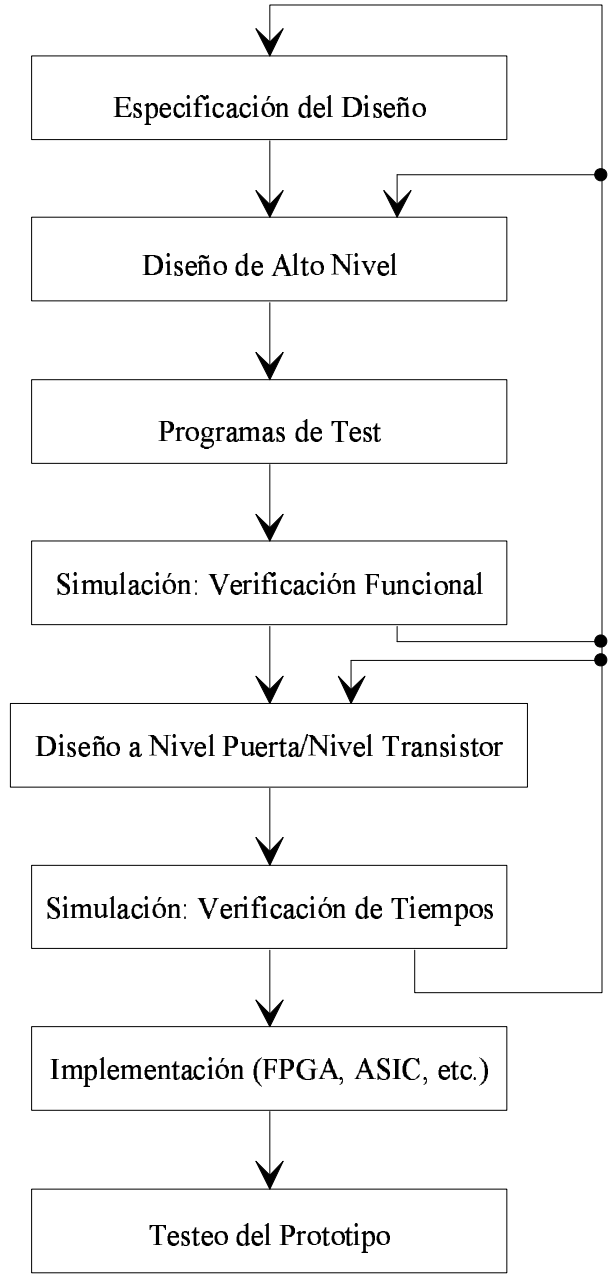

Especificación del Diseño
Diseño de Alto Nivel
Programas de Test
Simulación: Verificación Funcional
Diseño a Nivel Puerta/Nivel Transistor
Simulación: Verificación de Tiempos
Implementación (FPGA, ASIC, etc.)
Testeo del Prototipo

Fig. 1.1

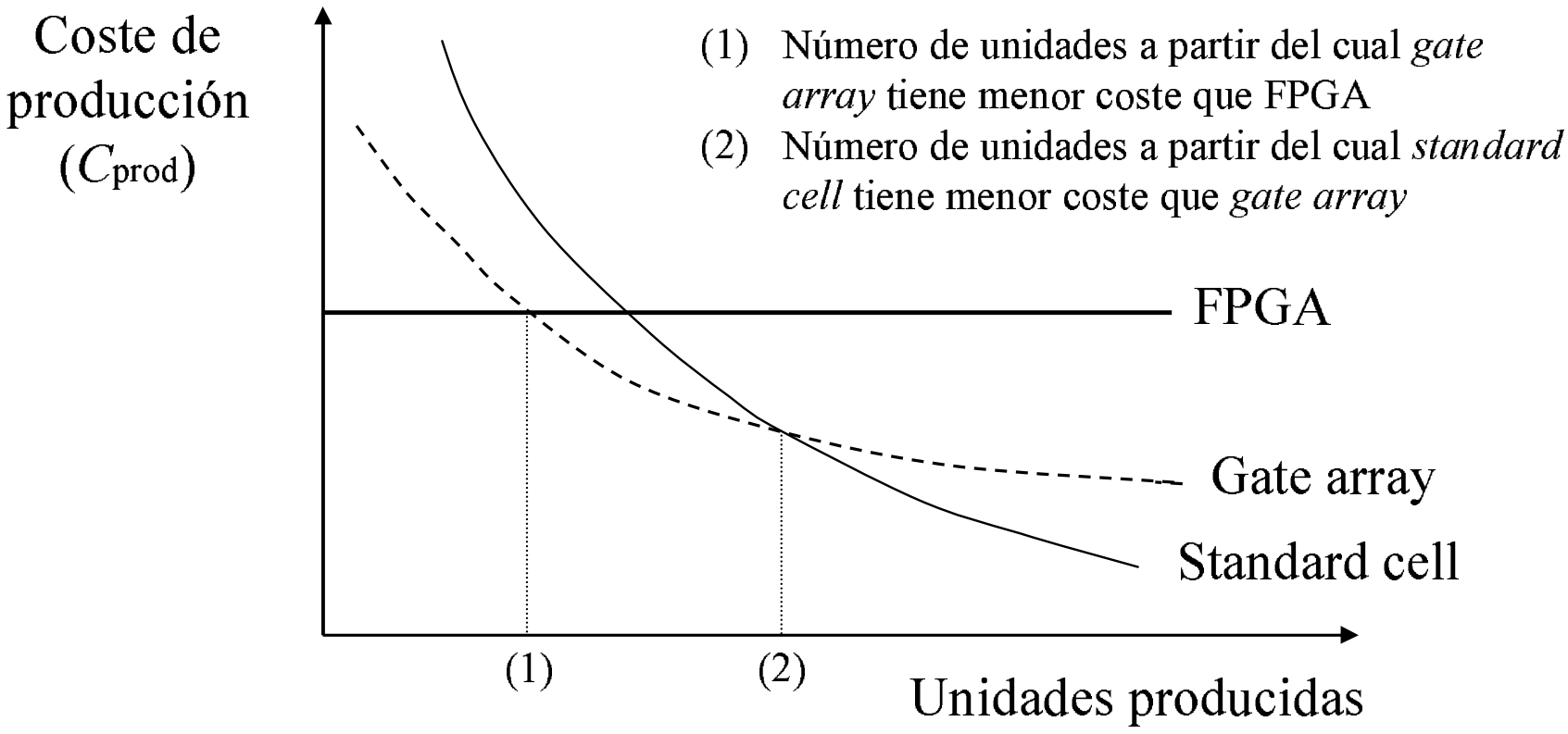

Coste de producción (C_{prod})
(1) Número de unidades a partir del cual *gate array* tiene menor coste que FPGA
(2) Número de unidades a partir del cual *standard cell* tiene menor coste que *gate array*
FPGA
Gate array
Standard cell
(1)
(2)
Unidades producidas

Fig. 1.2

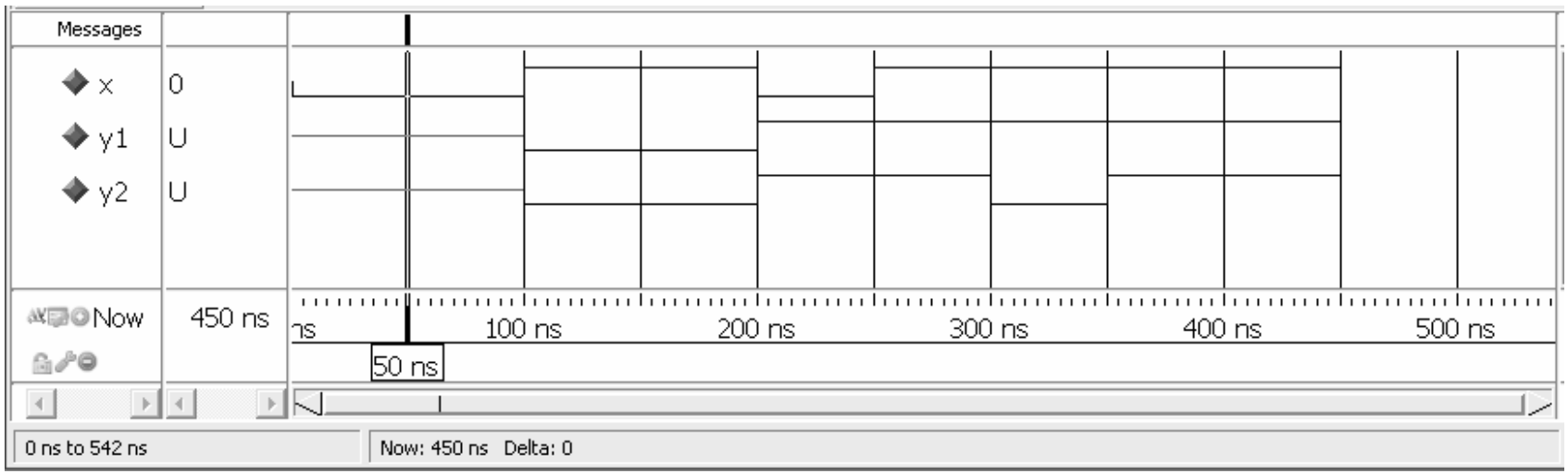

Messages
x
y1
y2
0
U
U
Now
450 ns
ns
100 ns
200 ns
300 ns
400 ns
500 ns
50 ns
0 ns to 542 ns
Now: 450 ns Delta: 0

Fig. 1.3

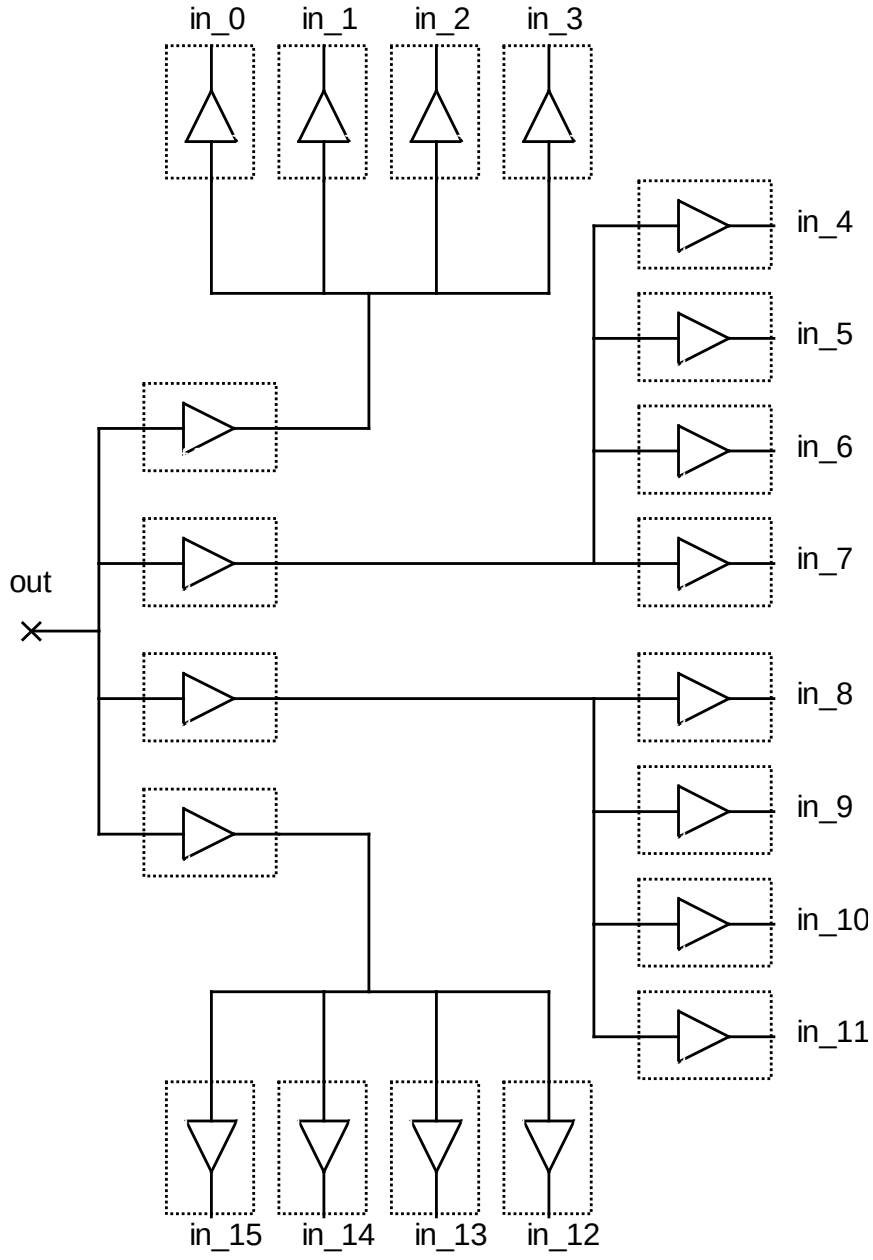

in_0
in_1
in_2
in_3
in_4
in_5
in_6
in_7
out
in_8
in_9
in_10
in_11
in_15
in_14
in_13
in_12

Fig. 1.4

Circuito banco de pruebas

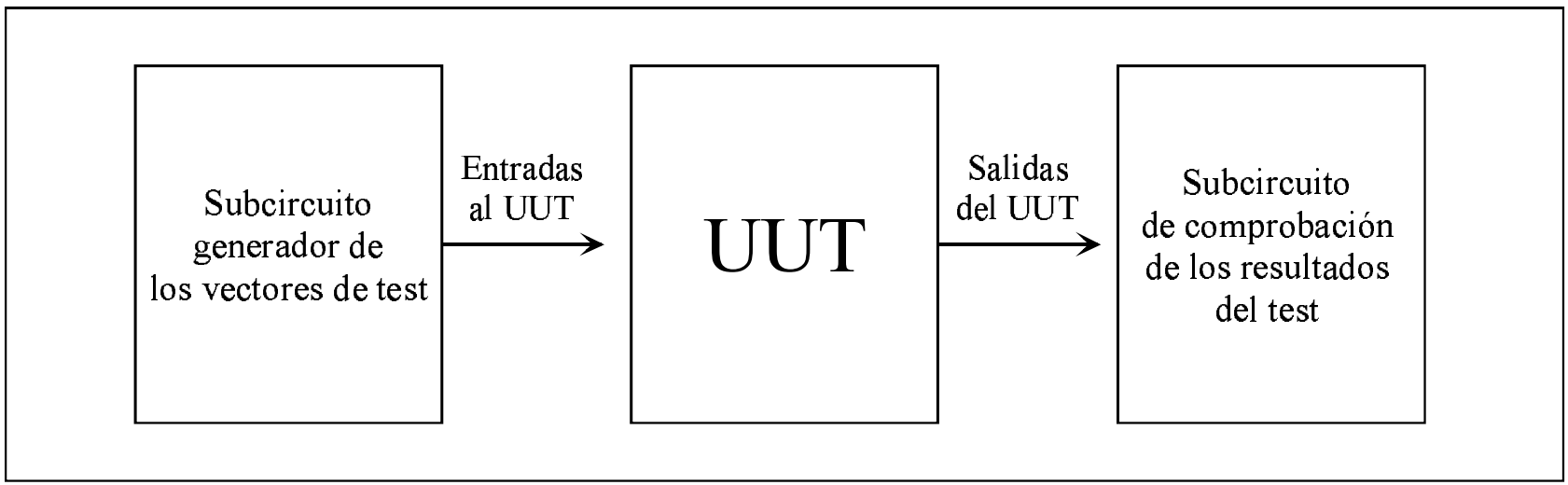


Fig. 1.5

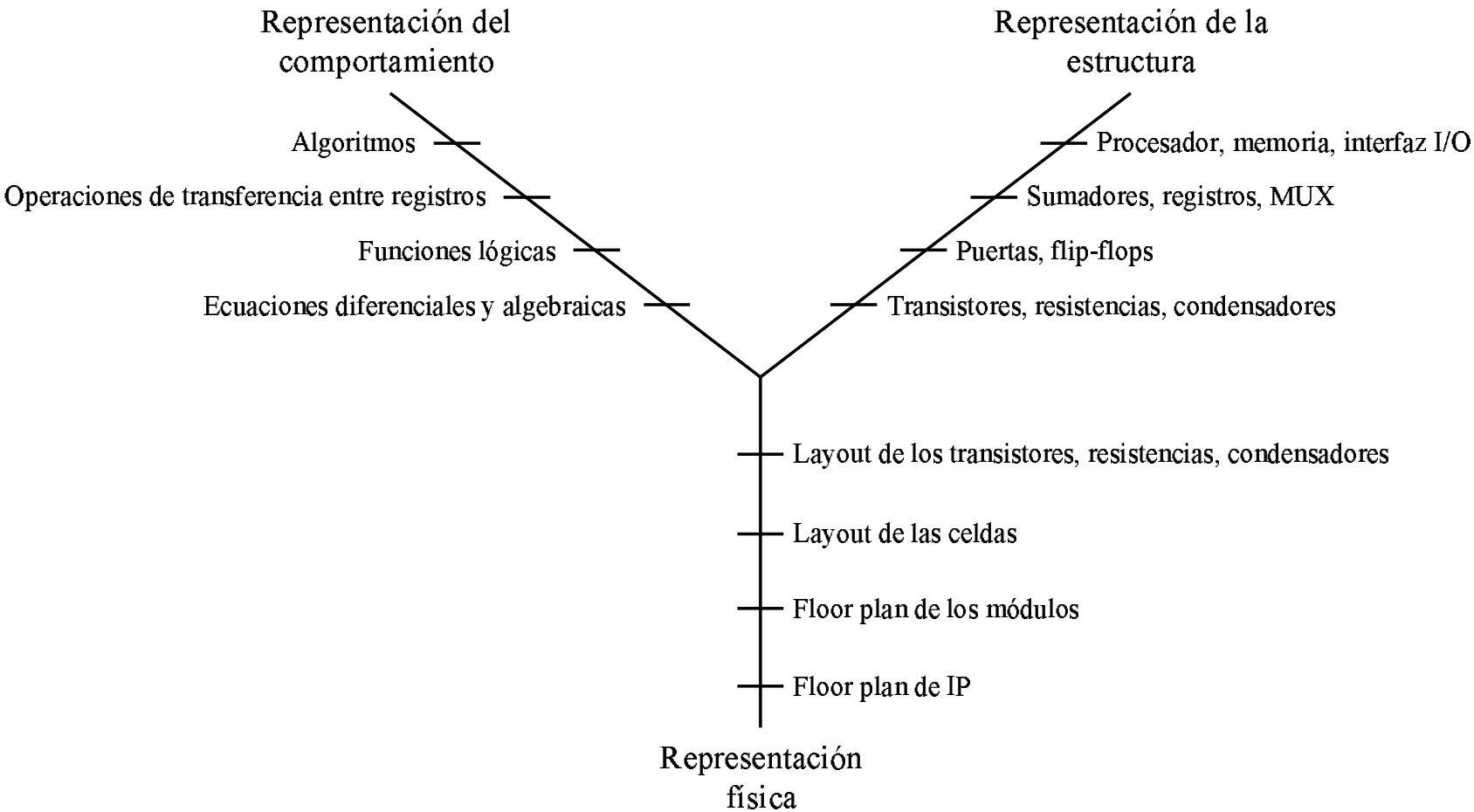

Representación del comportamiento
Representación de la estructura
Algoritmos
Operaciones de transferencia entre registros
Funciones lógicas
Ecuaciones diferenciales y algebraicas
Procesador, memoria, interfaz I/O
Sumadores, registros, MUX
Puertas, flip-flops
Transistores, resistencias, condensadores
Layout de los transistores, resistencias, condensadores
Layout de las celdas
Floor plan de los módulos
Floor plan de IP
Representación física

Fig. 1.6

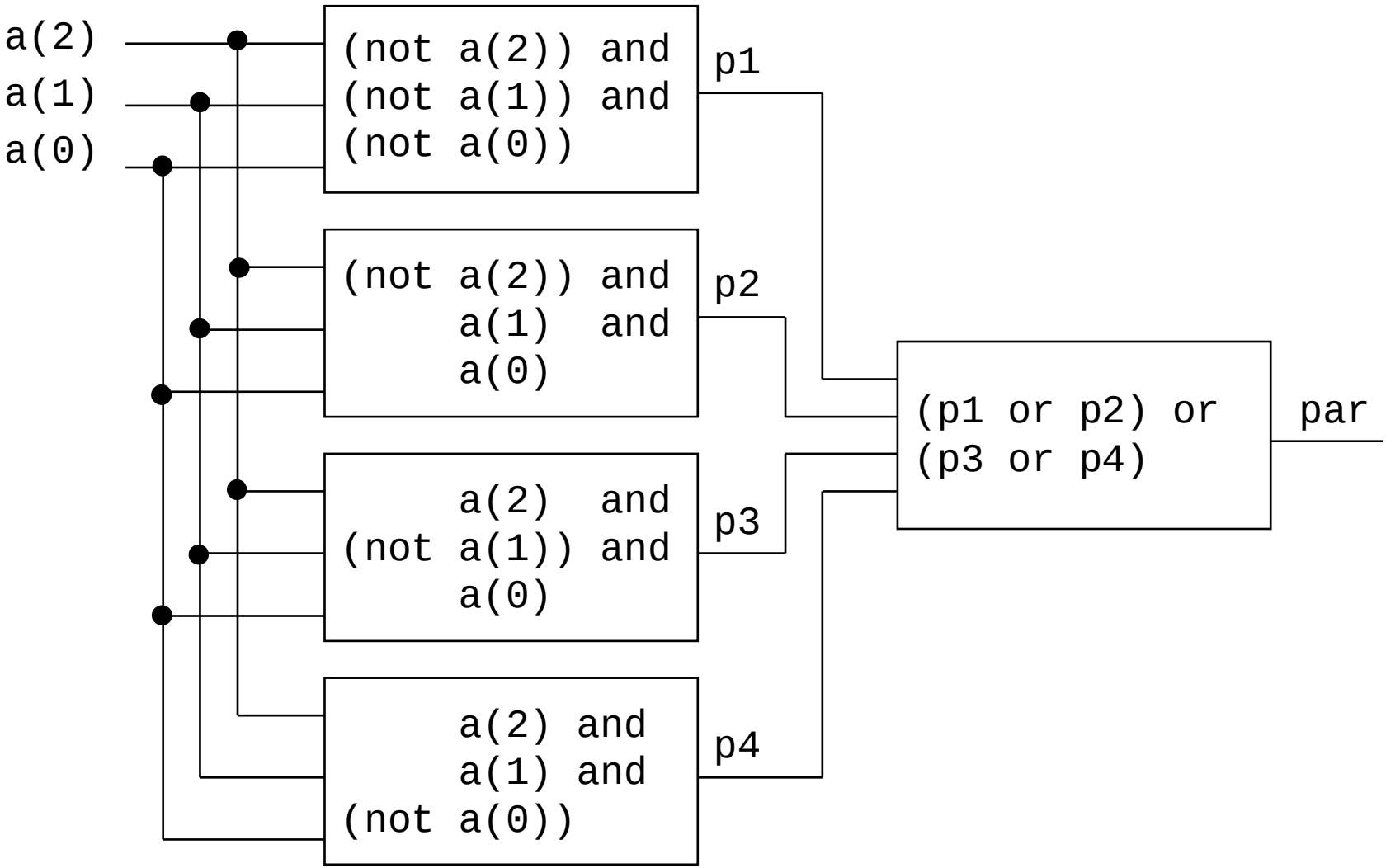

a(2)
a(1)
a(0)
(not a(2)) and
(not a(1)) and
(not a(0))
p1
(not a(2)) and
a(1) and
a(0)
p2
a(2) and
(not a(1)) and
a(0)
p3
a(2) and
a(1) and
(not a(0))
p4
(p1 or p2) or
(p3 or p4)
par

Fig. 1.7

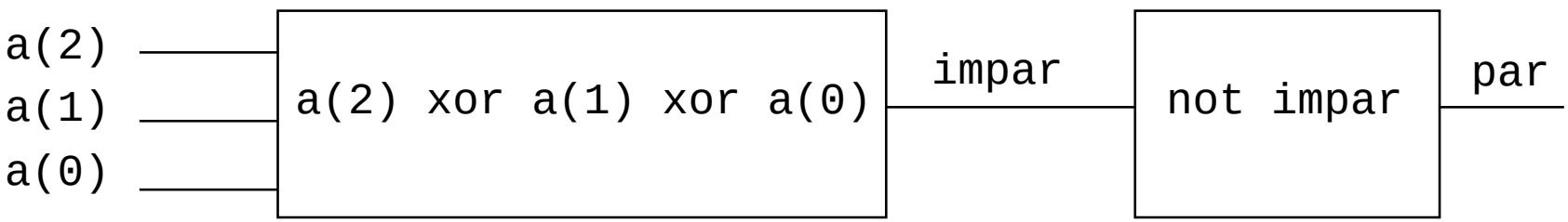


Fig. 1.8

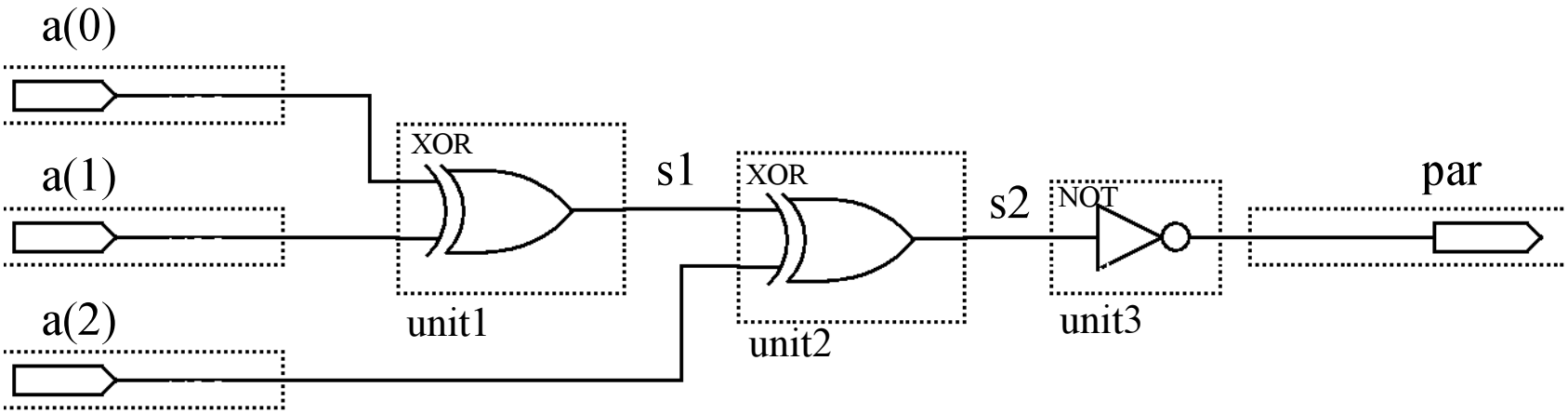


Fig. 1.9

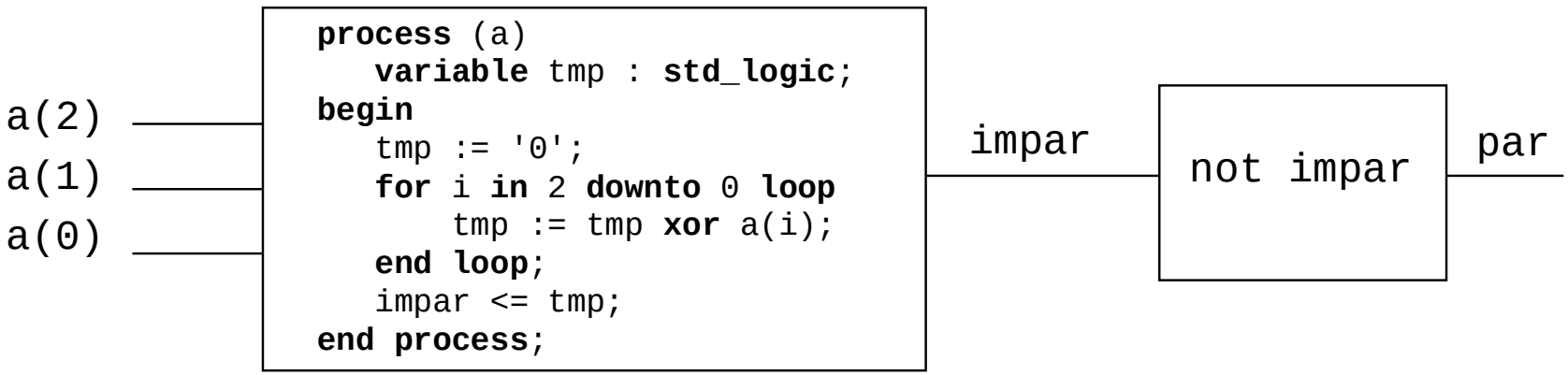

a(2)
a(1)
a(0)
process (a)
variable tmp : std_logic;
begin
tmp := '0';
for i in 2 downto 0 loop
tmp := tmp xor a(i);
end loop;
impar <= tmp;
end process;
impar
not impar
par

Fig. 1.10

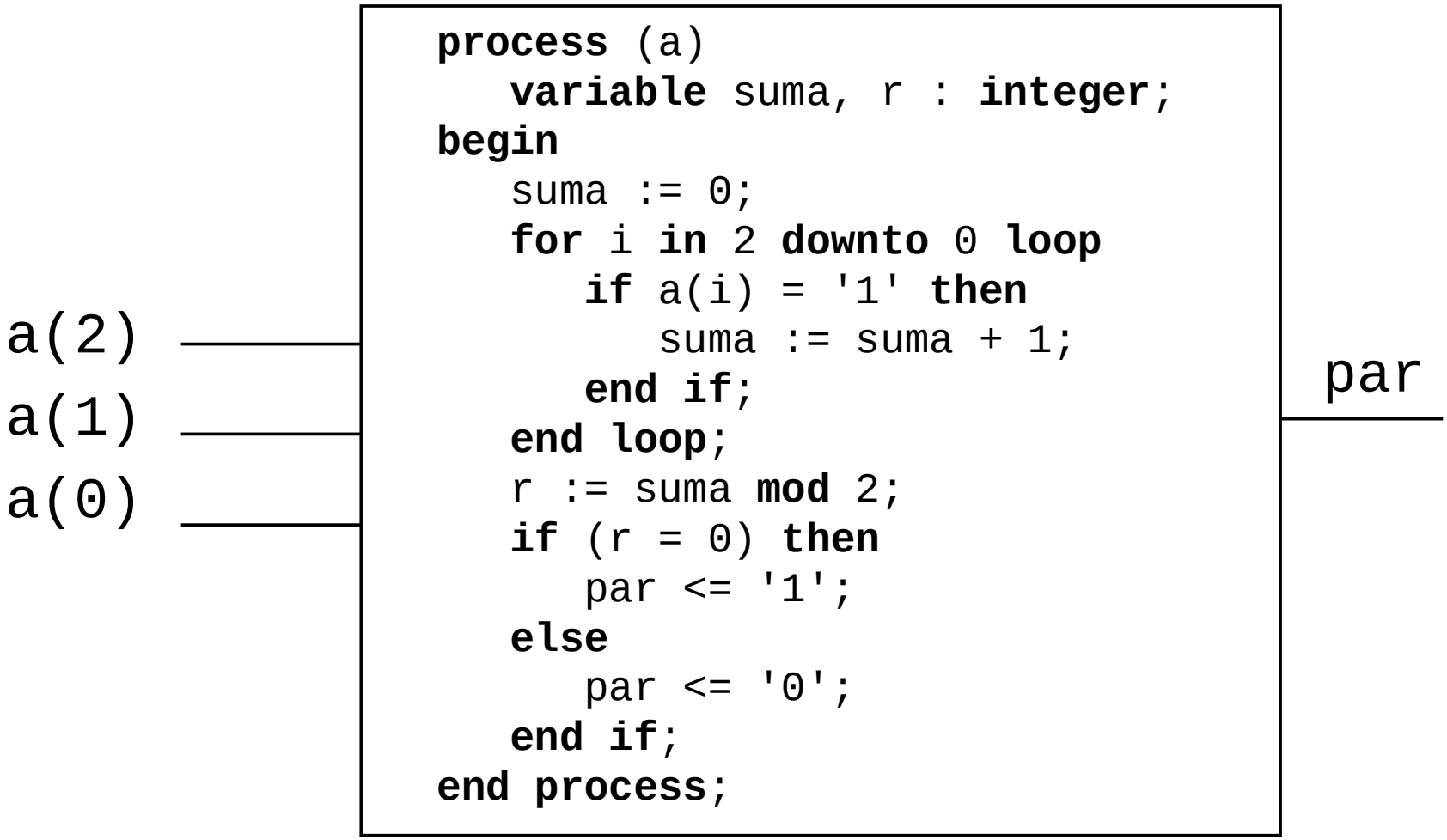
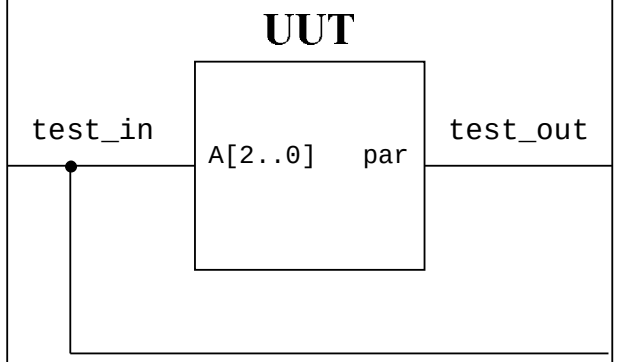

a(2)
a(1)
a(0)
process (a)
variable suma, r : integer;
begin
suma := 0;
for i in 2 downto 0 loop
if a(i) = '1' then
suma := suma + 1;
end if;
end loop;
r := suma mod 2;
if (r = 0) then
par <= '1';
else
par <= '0';
end if;
end process;
par

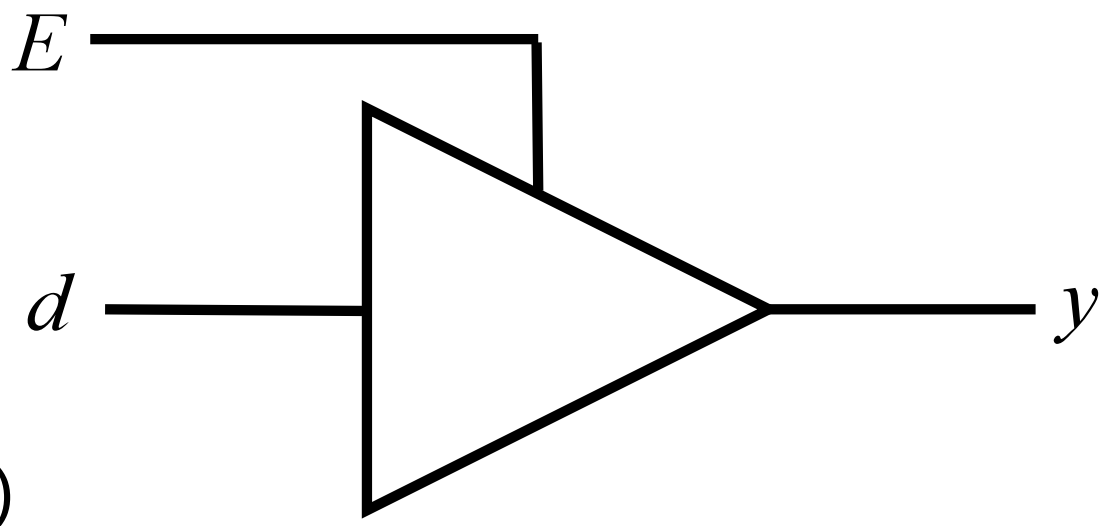
Fig. 1.11

```
vect_test: process
begin
   test_in <= "000";
   wait for 200 ns;
   test_in <= "001";
   wait for 200 ns;
   test_in <= "010";
   wait for 200 ns;
   test_in <= "011";
   wait for 200 ns;
   test_in <= "100";
   wait for 200 ns;
   test_in <= "101";
   wait for 200 ns;
   test_in <= "110";
   wait for 200 ns;
   test_in <= "111";
   wait for 200 ns;
end process vect_test;
```




```
verif : process
   variable error_status : boolean;
begin
   wait on test_in;
   wait for 100 ns;
   if ( ( test_in = "000" and test_out = '1' ) or
        ( test_in = "001" and test_out = '0' ) or
        ( test_in = "010" and test_out = '0' ) or
        ( test_in = "011" and test_out = '1' ) or
        ( test_in = "100" and test_out = '0' ) or
        ( test_in = "101" and test_out = '1' ) or
        ( test_in = "110" and test_out = '1' ) or
        ( test_in = "111" and test_out = '0' ) )
   then
       error_status := false;
   else
       error_status := true;
   end if;
   assert not error_status
               report "Test fallado."
               severity note;
end process verif;
```

Fig. 1.12



a)

E	Y
0	Z
1	d

b)

Fig. 1.13

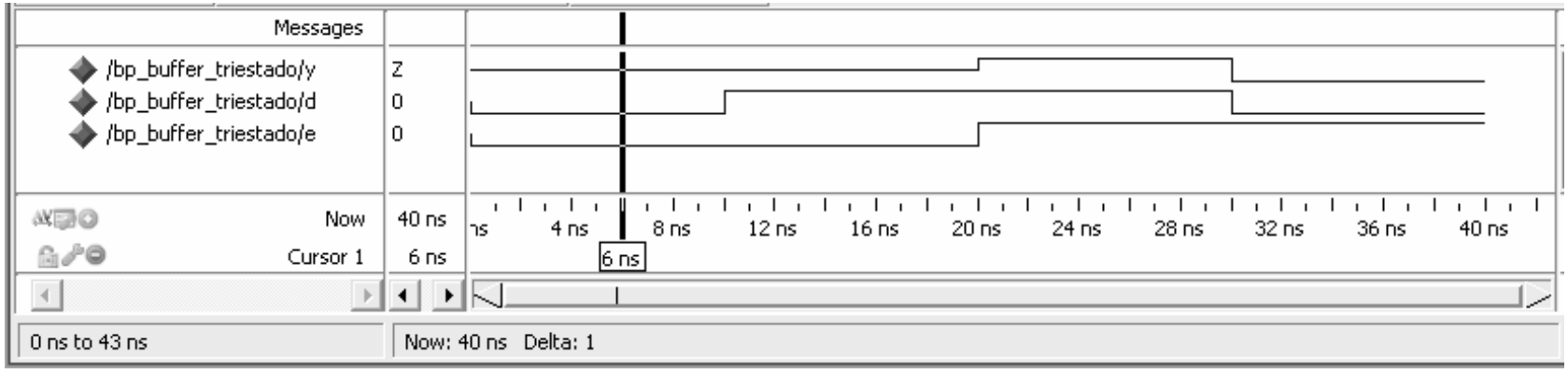

Messages
/bp_buffer_triestado/y
/bp_buffer_triestado/d
/bp_buffer_triestado/e
Z
0
0
Now
40 ns
Cursor 1
6 ns
ns
4 ns
8 ns
12 ns
16 ns
20 ns
24 ns
28 ns
32 ns
36 ns
40 ns
6 ns
0 ns to 43 ns
Now: 40 ns Delta: 1

Fig. 1.14

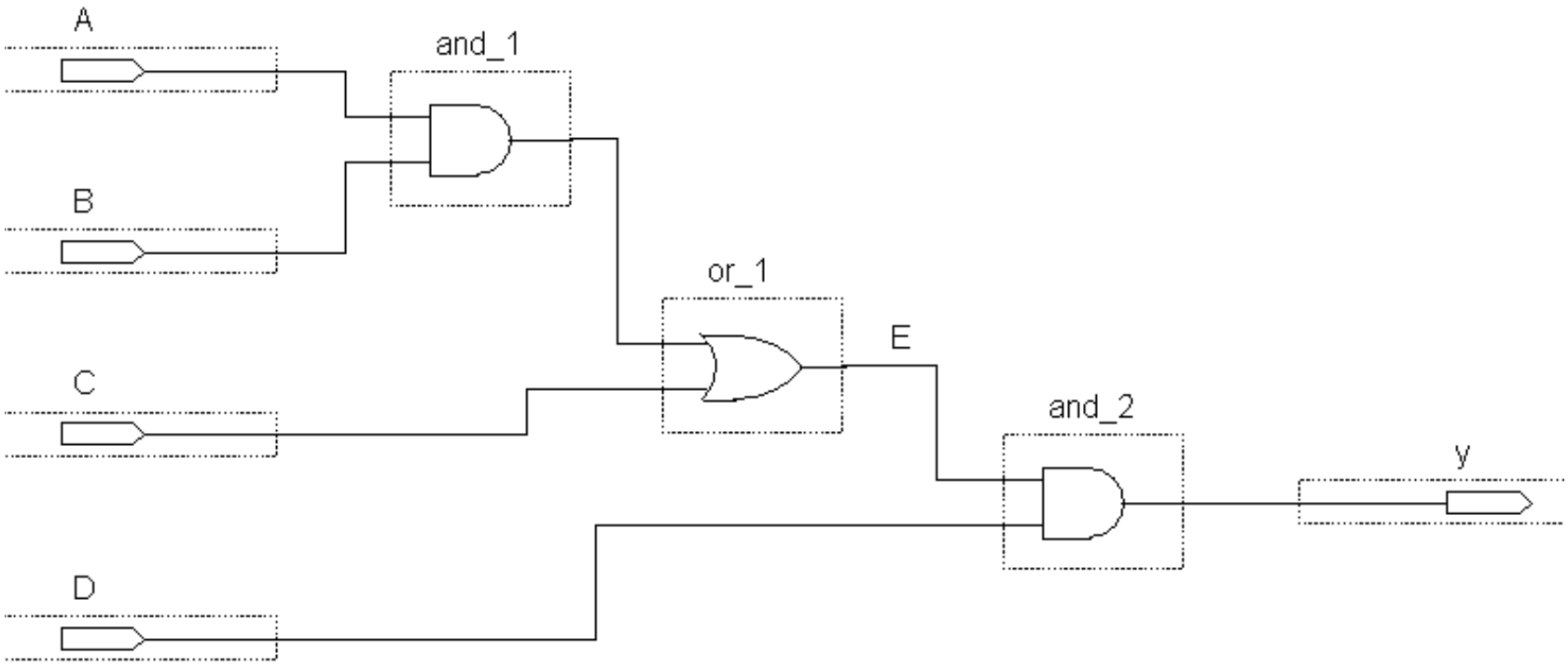


Fig. 1.15

a_2	a_1	a_0	par
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

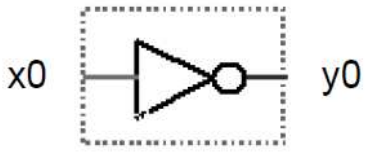
Tabla 1.1

Entradas A B	Salida correcta z	$A/0$ z	$B/0$ z	$z/0$ z	$A/1$ z	$B/1$ z	$z/1$ z
0 0	0	0	0	0	0	0	1
0 1	0	0	0	0	1	0	1
1 0	0	0	0	0	0	1	1
1 1	1	0	0	0	1	1	1

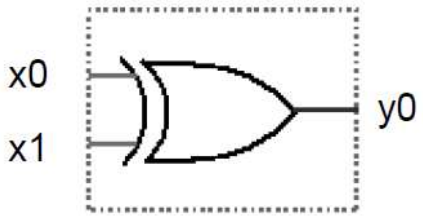
Tabla 1.2

TEMA 2

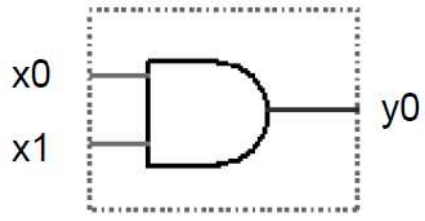
Conceptos básicos de VHDL




```
entity not is
  port ( y0  : out std_logic;
         x0  : in  std_logic );
end entity not;
```




```
entity xor2 is
  port ( y0      : out std_logic;
         x0, x1  : in  std_logic );
end entity xor2;
```




```
entity and2 is
  port ( y0      : out std_logic;
         x0, x1  : in  std_logic );
end entity and2;
```

Fig. 2.1

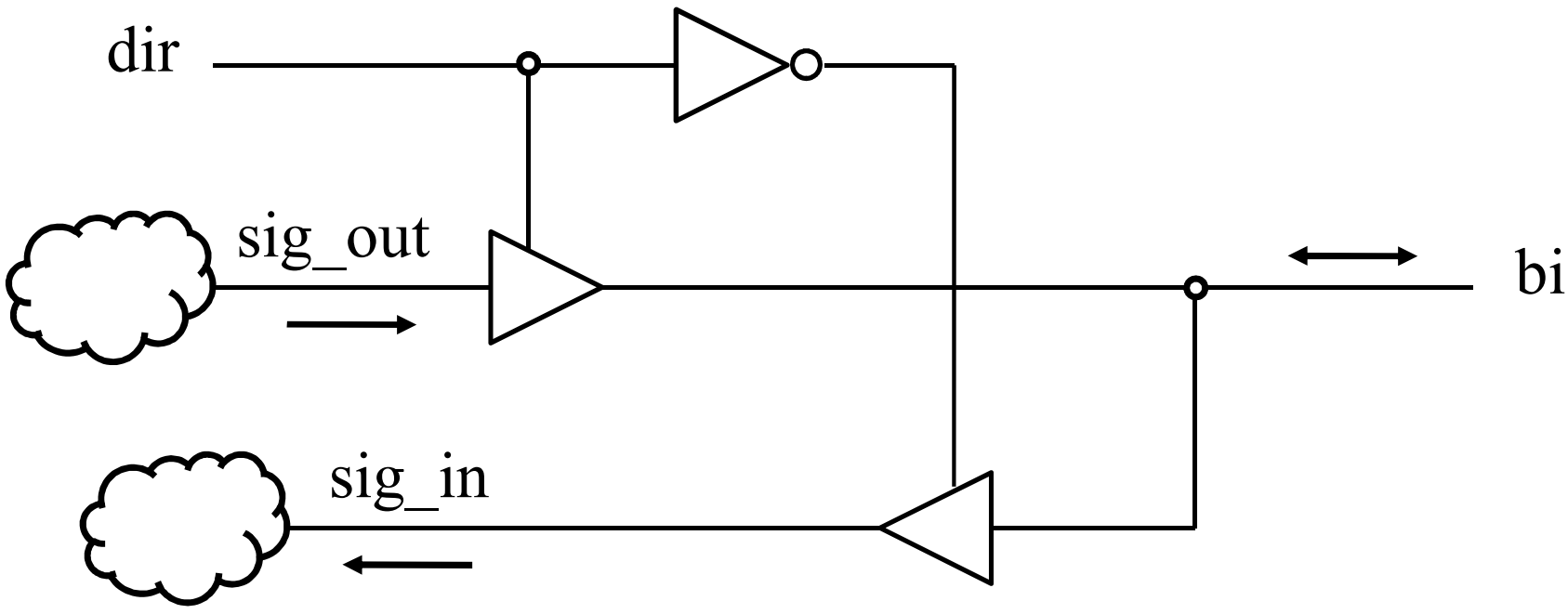


Fig. 2.2

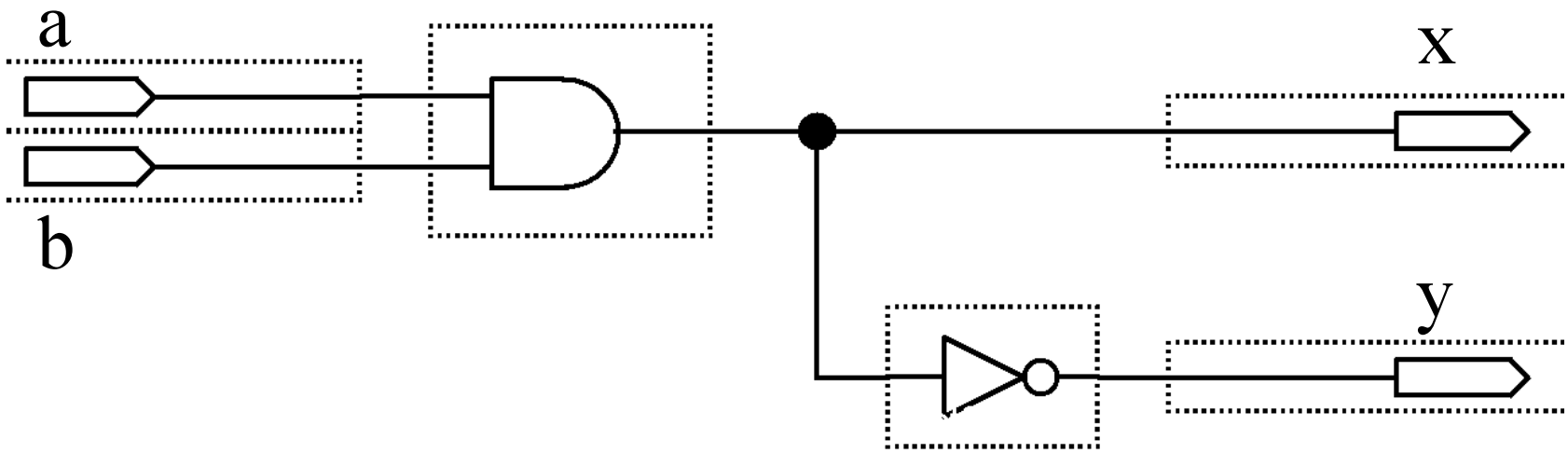


Fig. 2.3

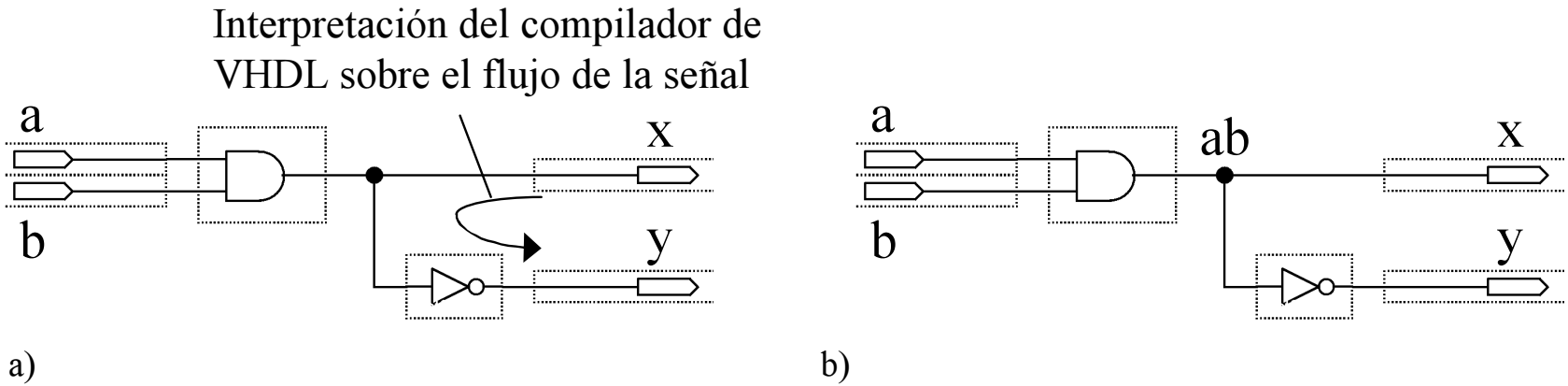
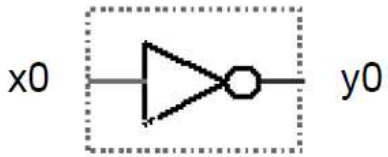
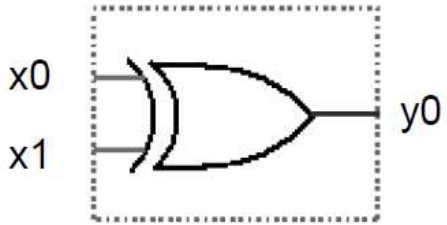

Interpretación del compilador de
VHDL sobre el flujo de la señal
a
b
x
y
a)
a
b
ab
x
y
b)

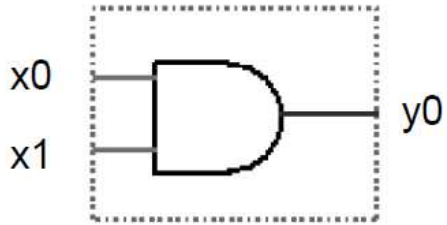
Fig. 2.4




```
architecture not of not is
begin
  y0 <= not x0;
end architecture not;
```




```
architecture xor2 of xor2 is
begin
  y0 <= x0 xor x1;
end architecture xor2;
```




```
architecture and2 of and2 is
begin
  y0 <= x0 and x1;
end architecture and2;
```

Fig. 2.5

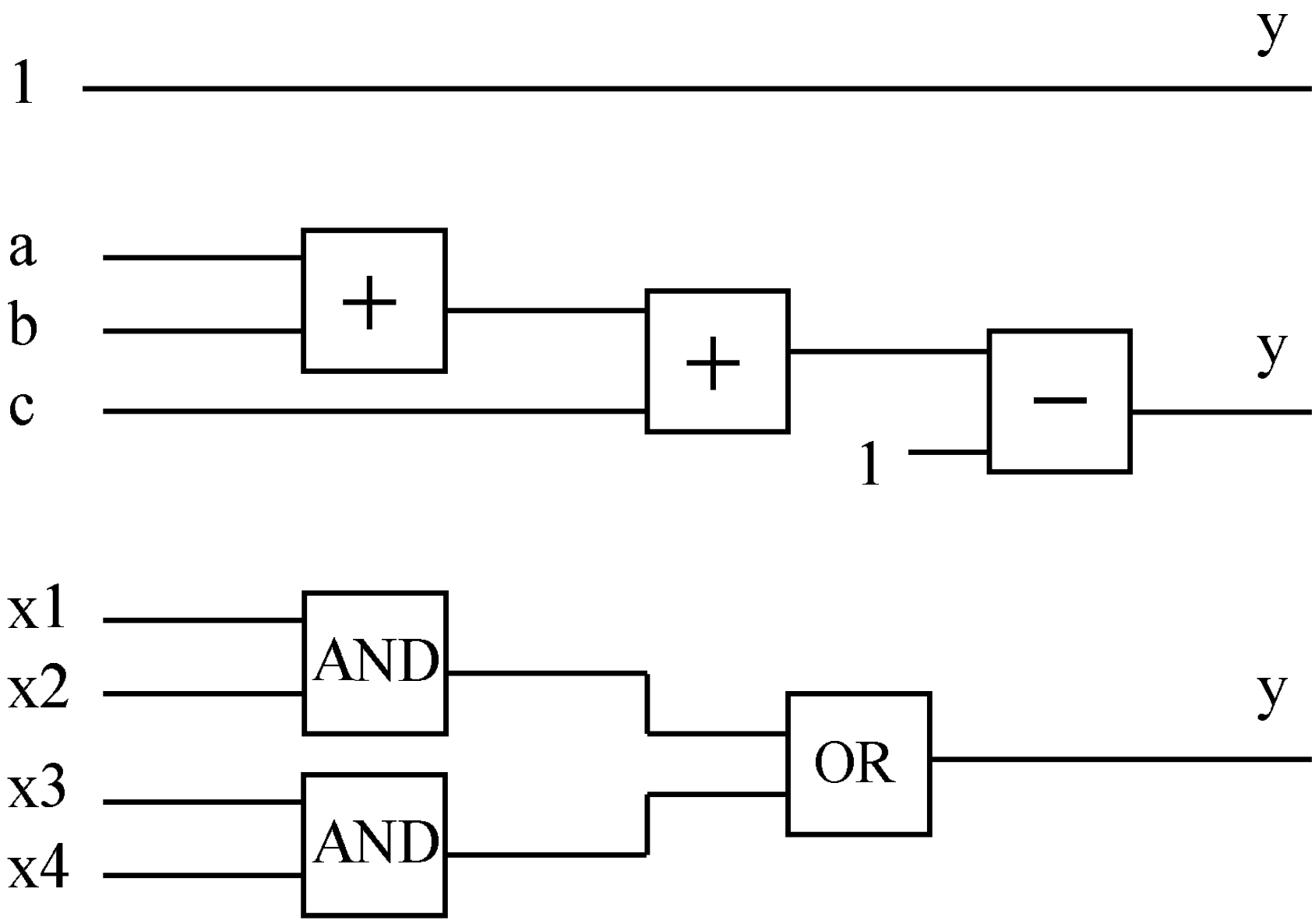


Fig. 2.6

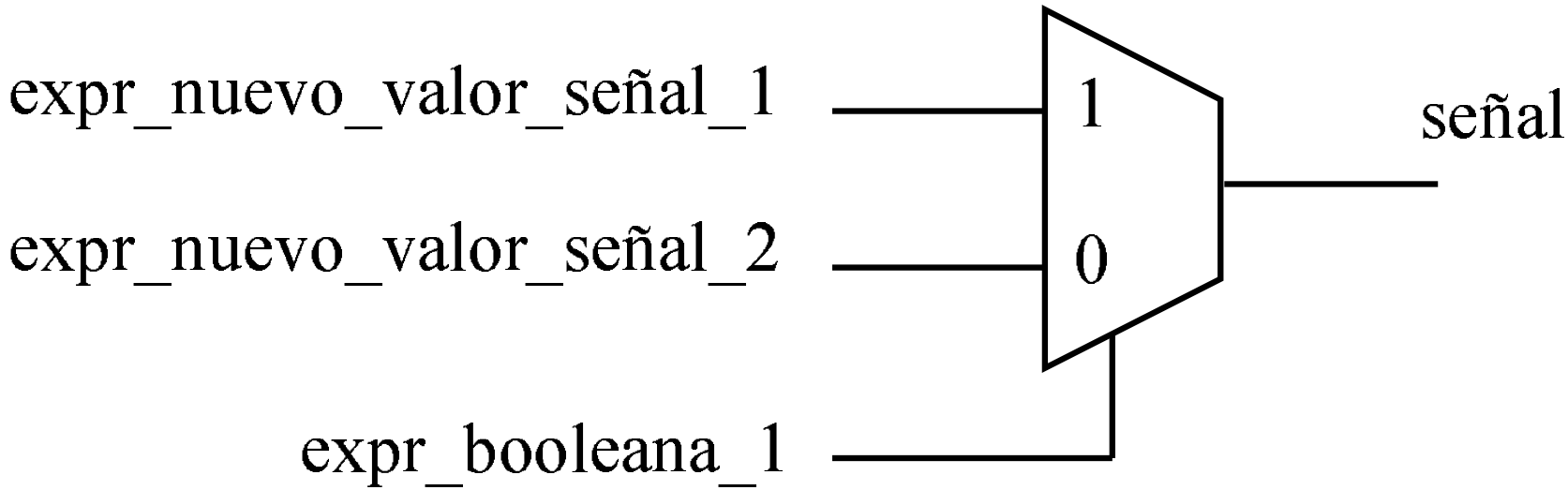

expr_nuevo_valor_señal_1
expr_nuevo_valor_señal_2
expr_booleana_1
1
0
señal

Fig. 2.7

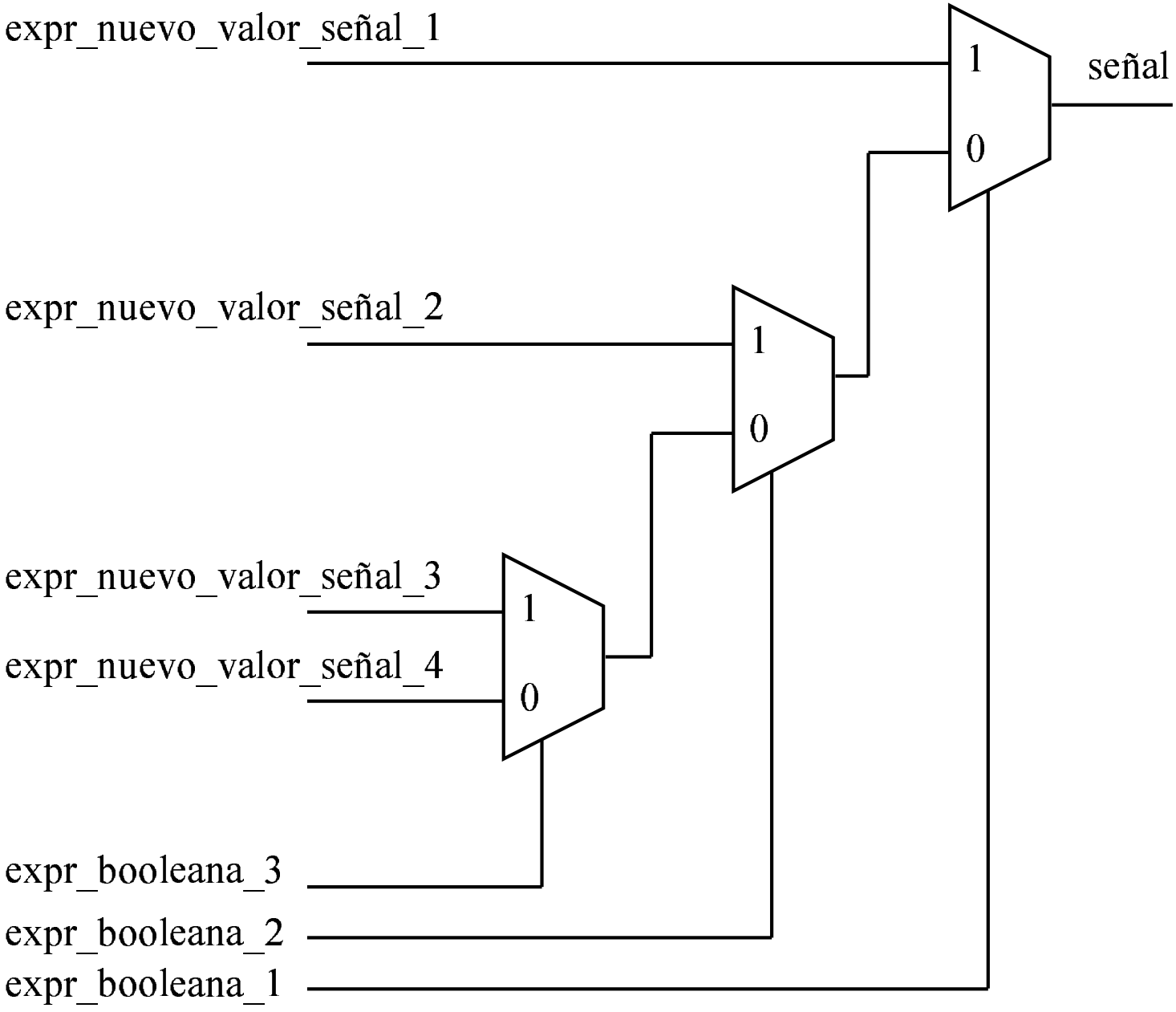

expr_nuevo_valor_señal_1
1
señal
0
expr_nuevo_valor_señal_2
1
0
expr_nuevo_valor_señal_3
1
expr_nuevo_valor_señal_4
0
expr_booleana_3
expr_booleana_2
expr_booleana_1

Fig. 2.8

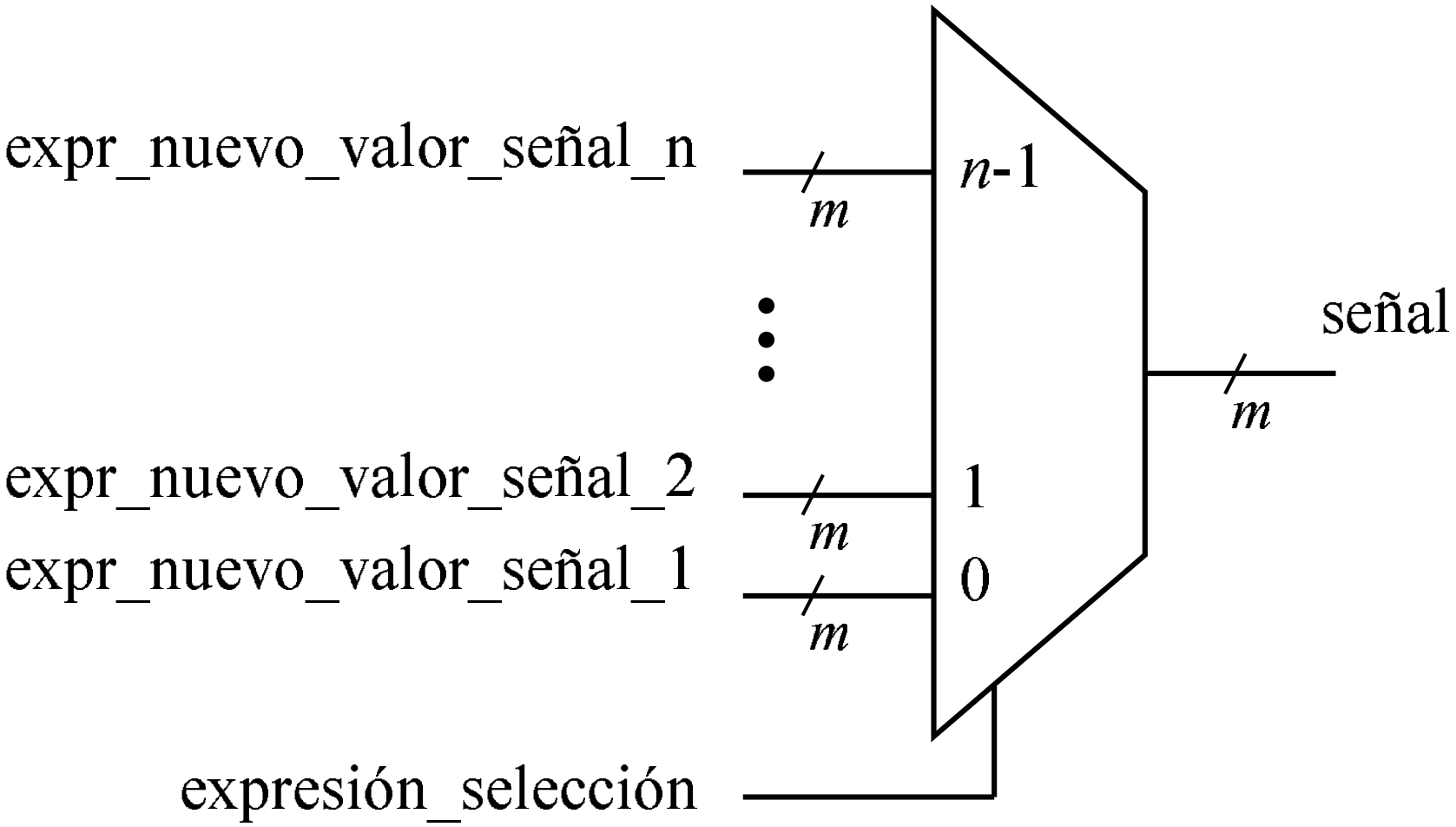

expr_nuevo_valor_señal_n
m
n-1
señal
m
expr_nuevo_valor_señal_2
m
1
expr_nuevo_valor_señal_1
m
0
expresión_selección

Fig. 2.9

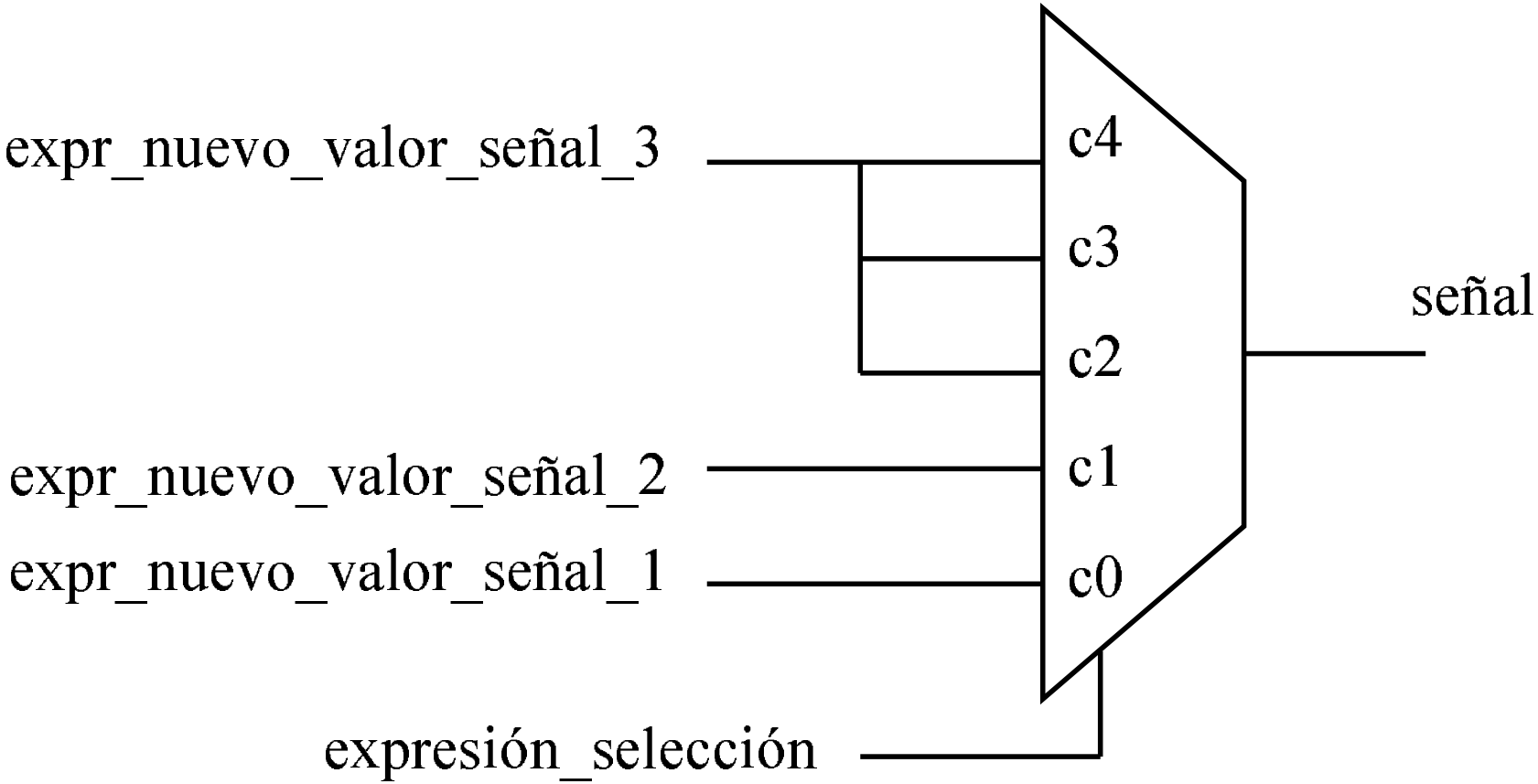
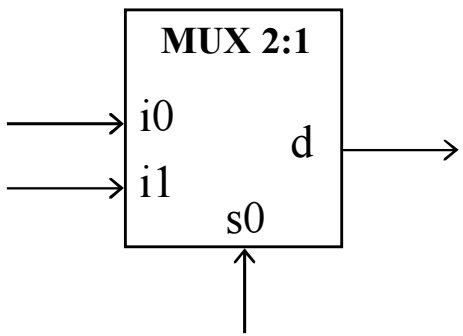
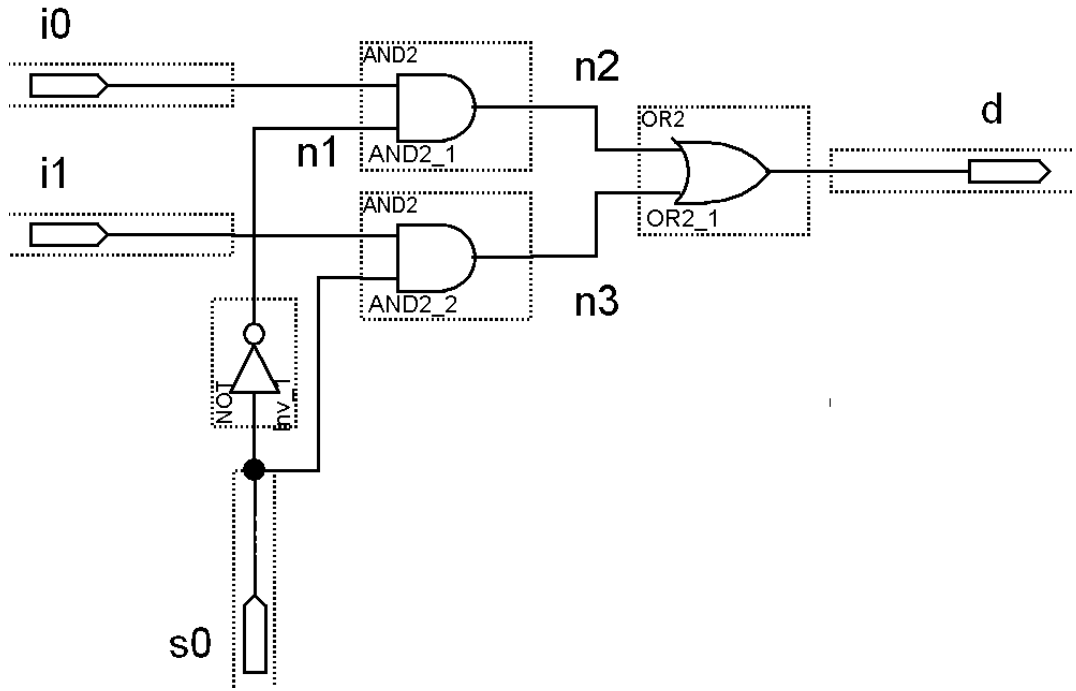

expr_nuevo_valor_señal_3
expr_nuevo_valor_señal_2
expr_nuevo_valor_señal_1
expresión_selección
c4
c3
c2
c1
c0
señal

Fig. 2.10


MUX 2:1
i0
i1
s0
d

a)


i0
i1
s0
n1
n2
n3
d
AND2
AND2_1
AND2
AND2_2
OR2
OR2_1
NOT
inv_1

b)

Fig. 2.11

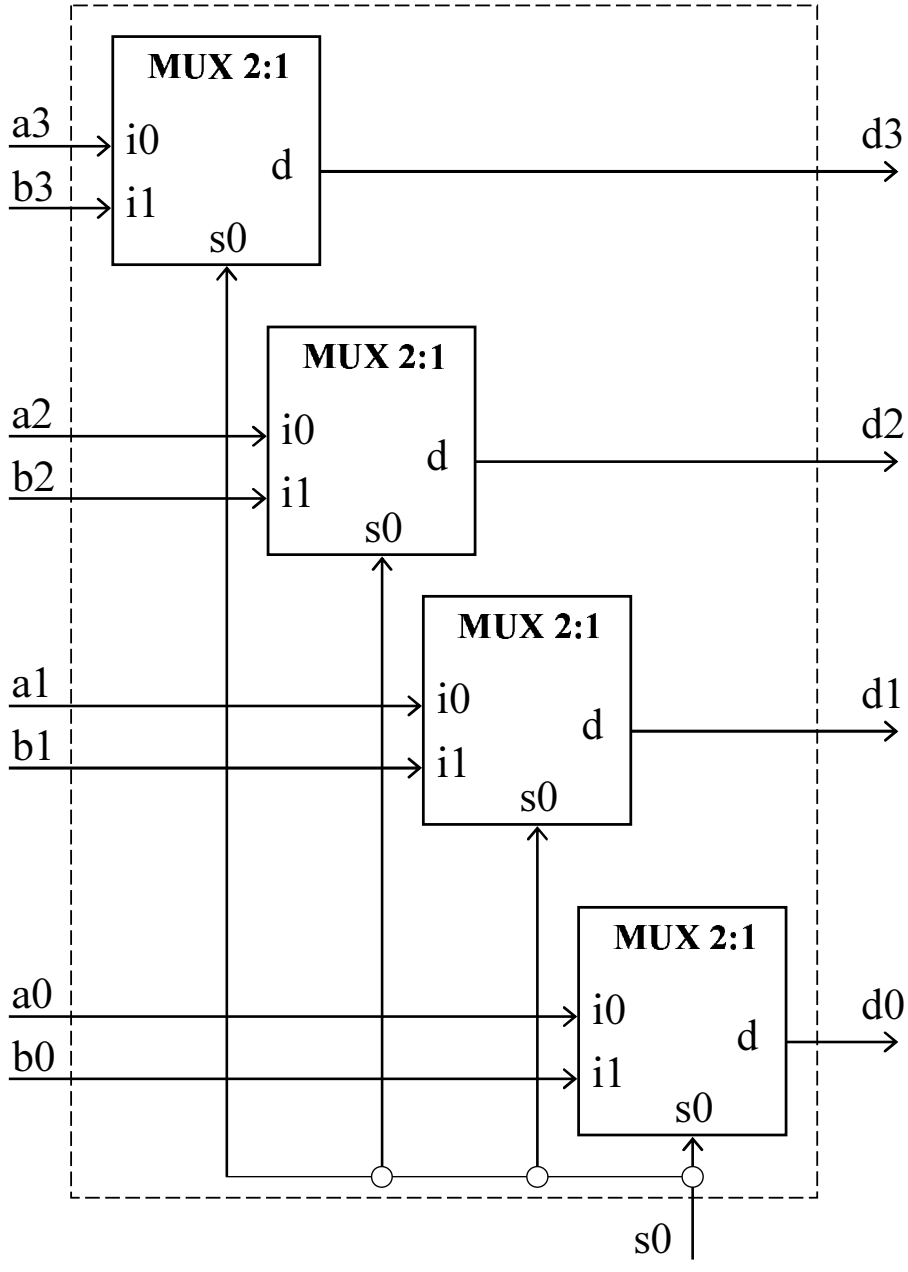

MUX 2:1
a3
b3
i0
i1
d
s0
d3
MUX 2:1
a2
b2
i0
i1
d
s0
d2
MUX 2:1
a1
b1
i0
i1
d
s0
d1
MUX 2:1
a0
b0
i0
i1
d
s0
d0
s0

Fig. 2.12

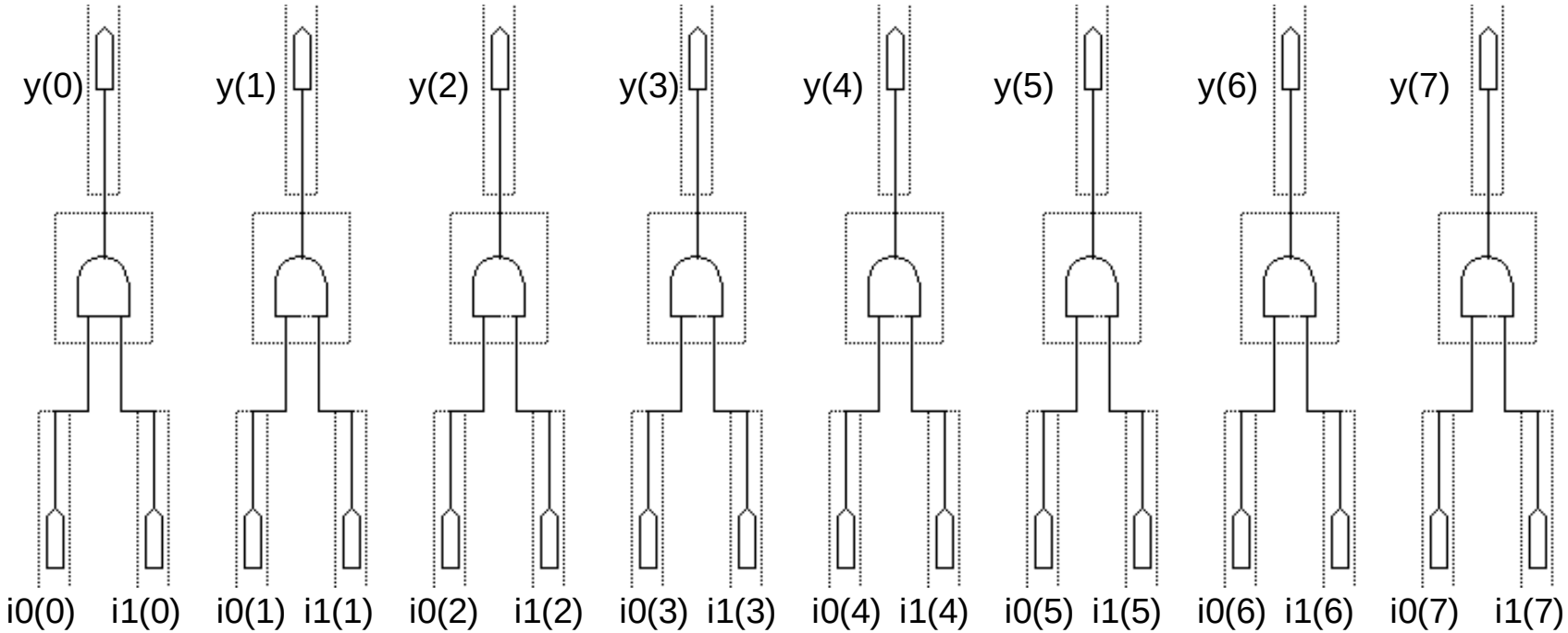

y(0)
y(1)
y(2)
y(3)
y(4)
y(5)
y(6)
y(7)
i0(0)
i1(0)
i0(1)
i1(1)
i0(2)
i1(2)
i0(3)
i1(3)
i0(4)
i1(4)
i0(5)
i1(5)
i0(6)
i1(6)
i0(7)
i1(7)

Fig. 2.13

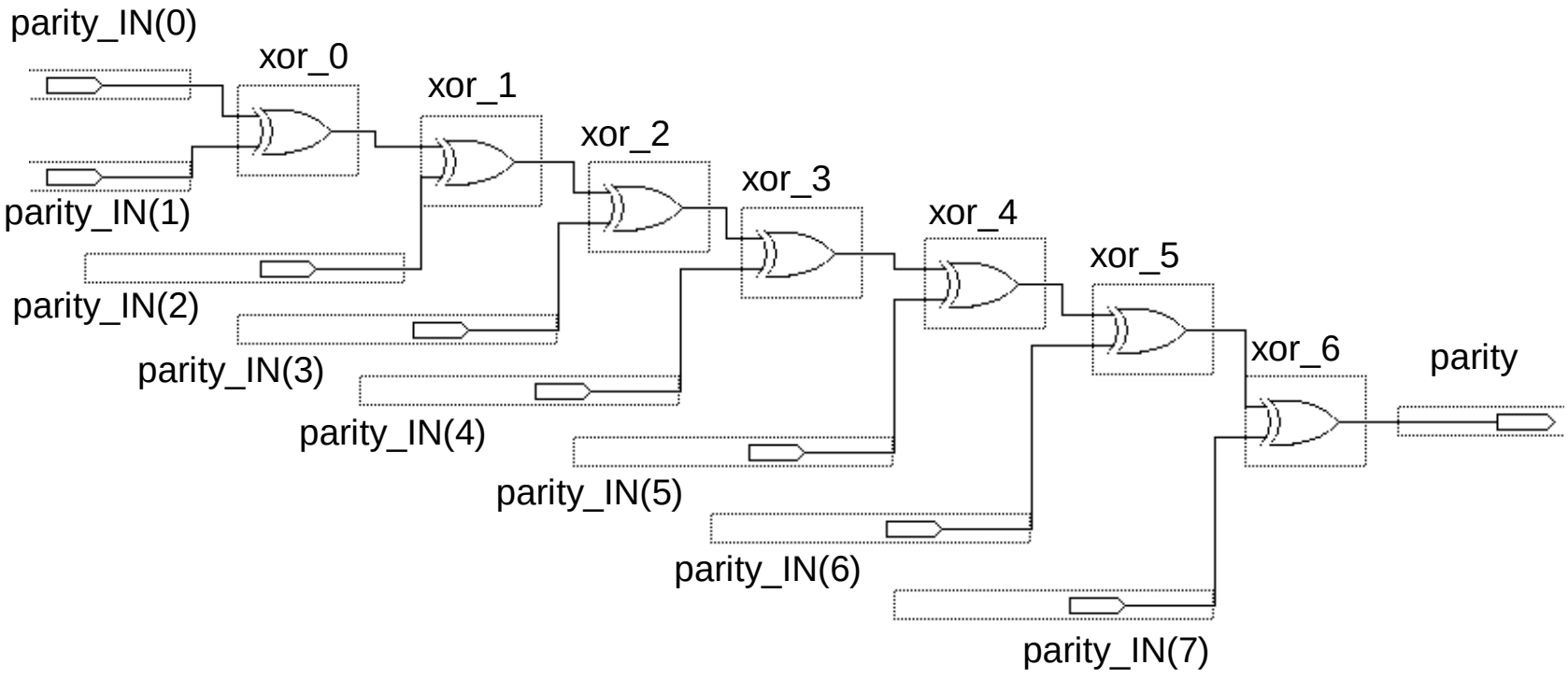

parity_IN(0)
xor_0
xor_1
xor_2
xor_3
xor_4
xor_5
xor_6
parity
parity_IN(1)
parity_IN(2)
parity_IN(3)
parity_IN(4)
parity_IN(5)
parity_IN(6)
parity_IN(7)

Fig. 2.14

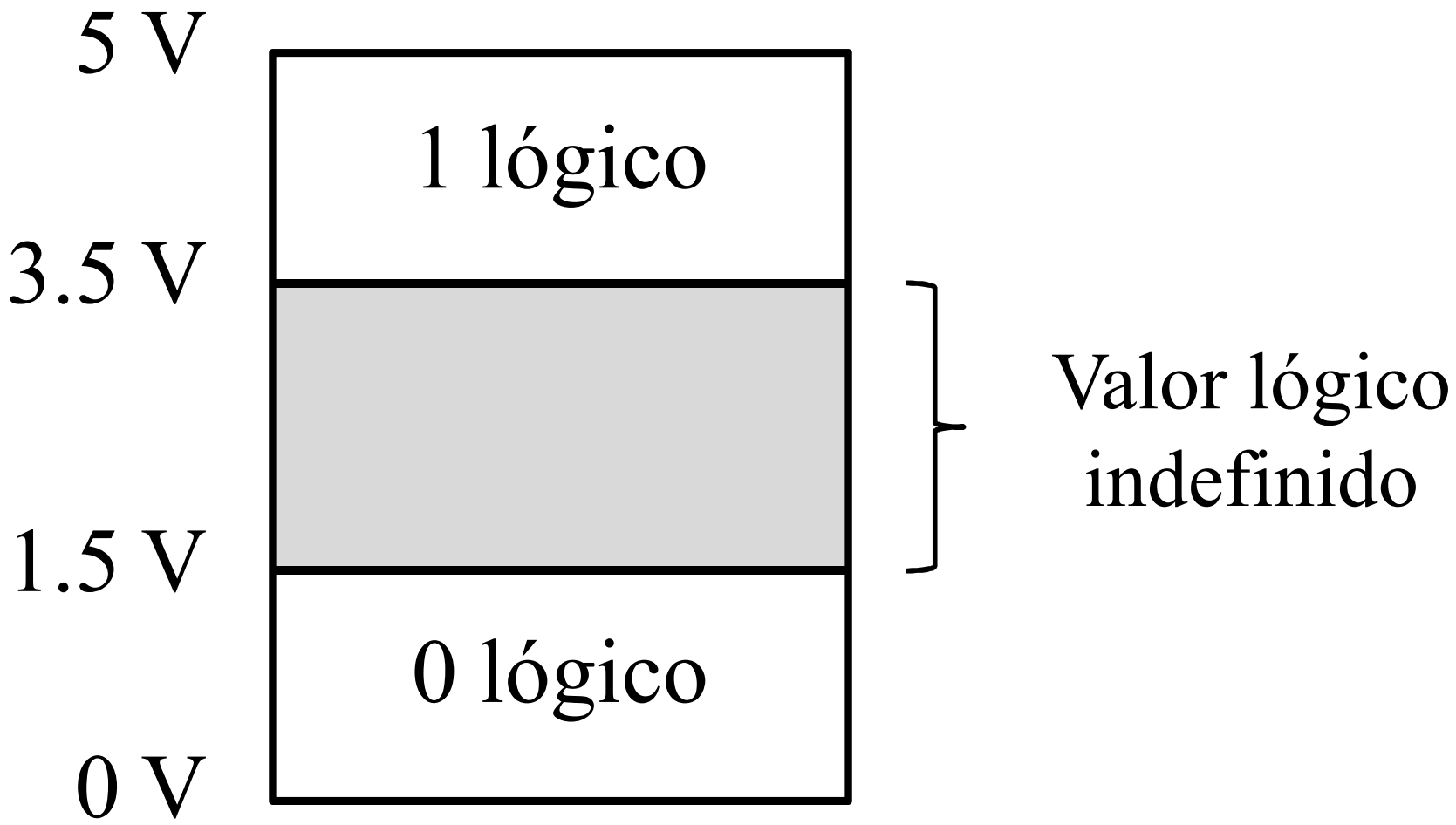


Fig. 2.15

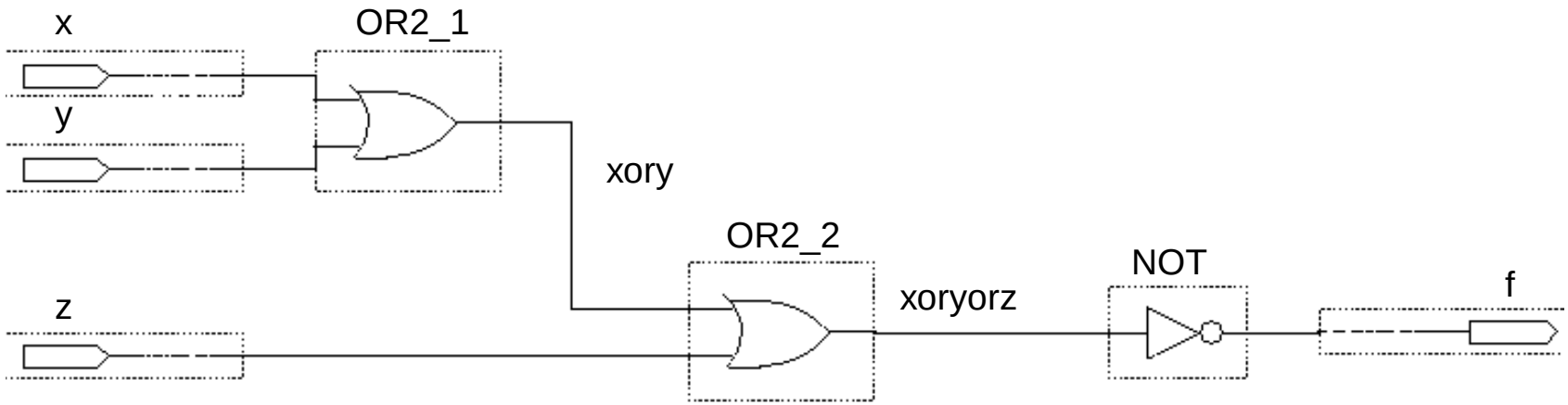


Fig. 2.16

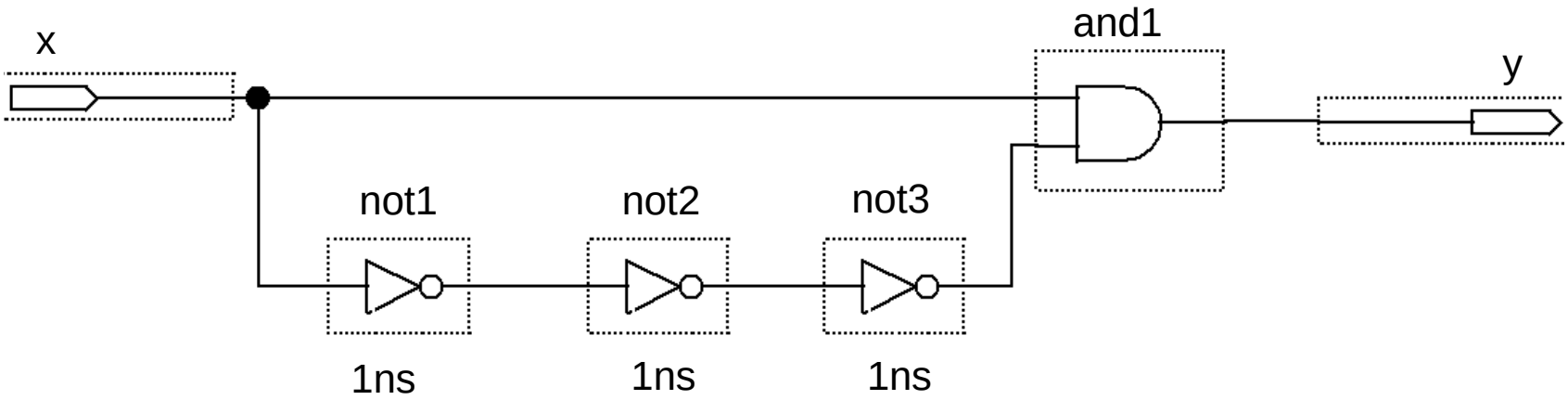
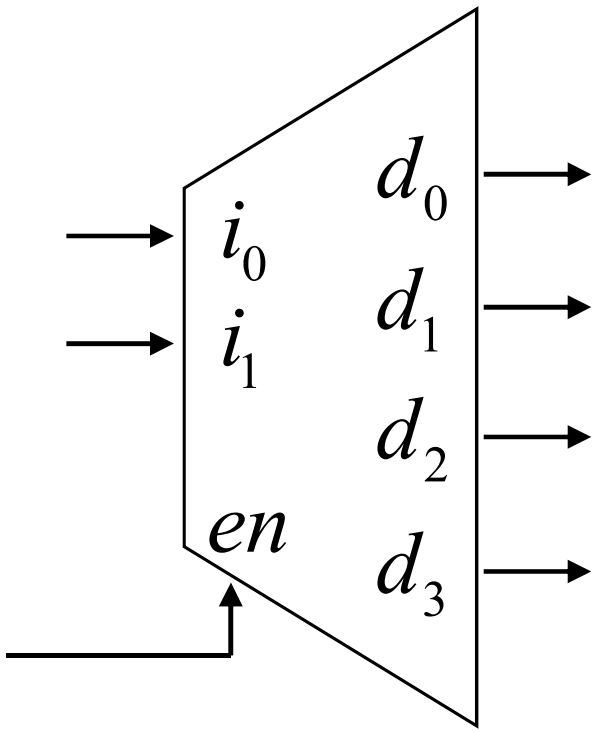


Fig. 2.17



en	i_1	i_0	d_3	d_2	d_1	d_0
0	–	–	0	0	0	0
1	0	0	0	0	0	1
1	0	1	0	0	1	0
1	1	0	0	1	0	0
1	1	1	1	0	0	0

Fig. 2.18

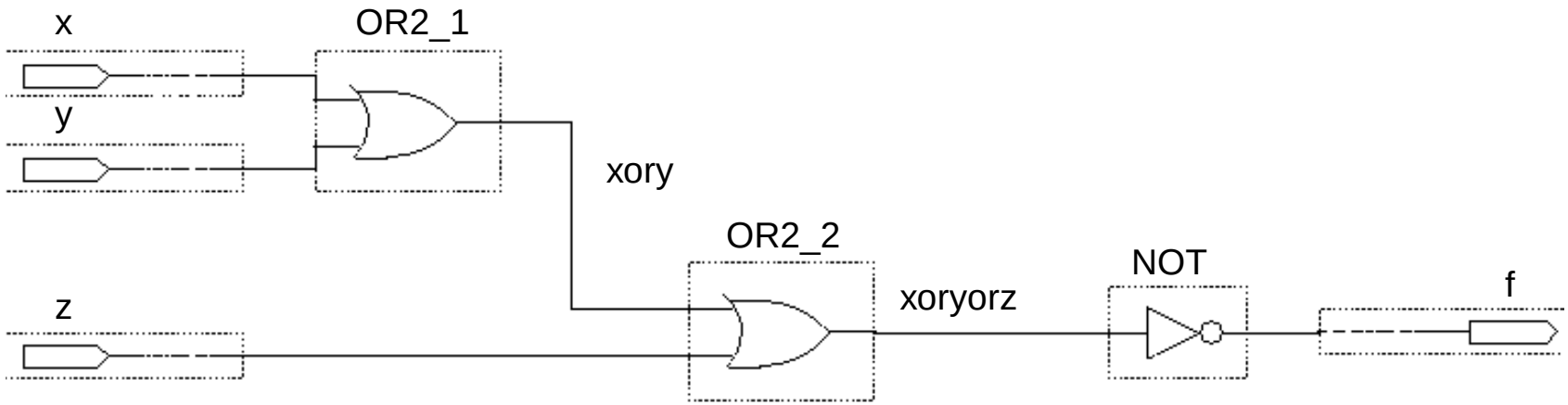


Fig. 2.19

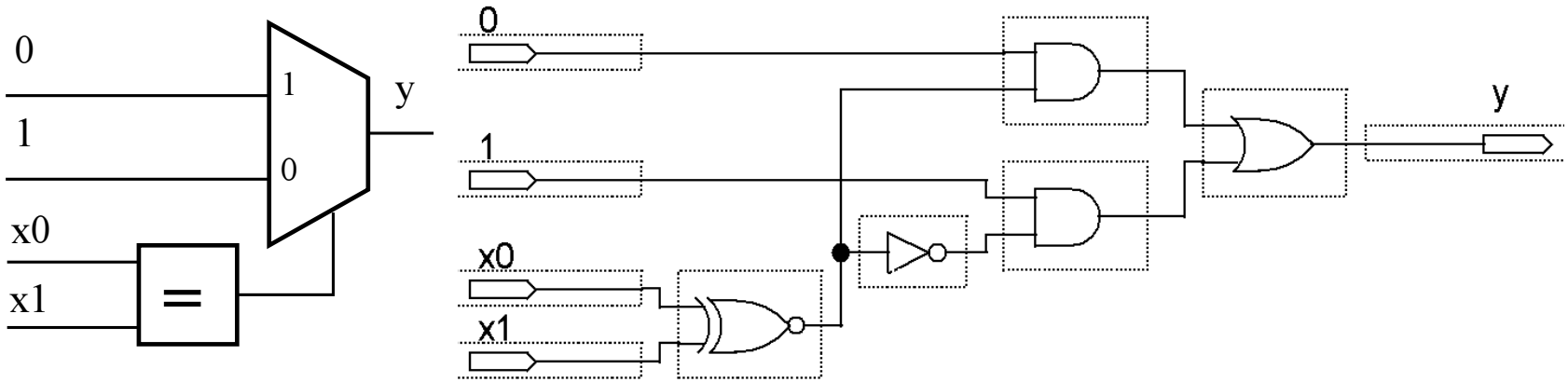


Fig. 2.20

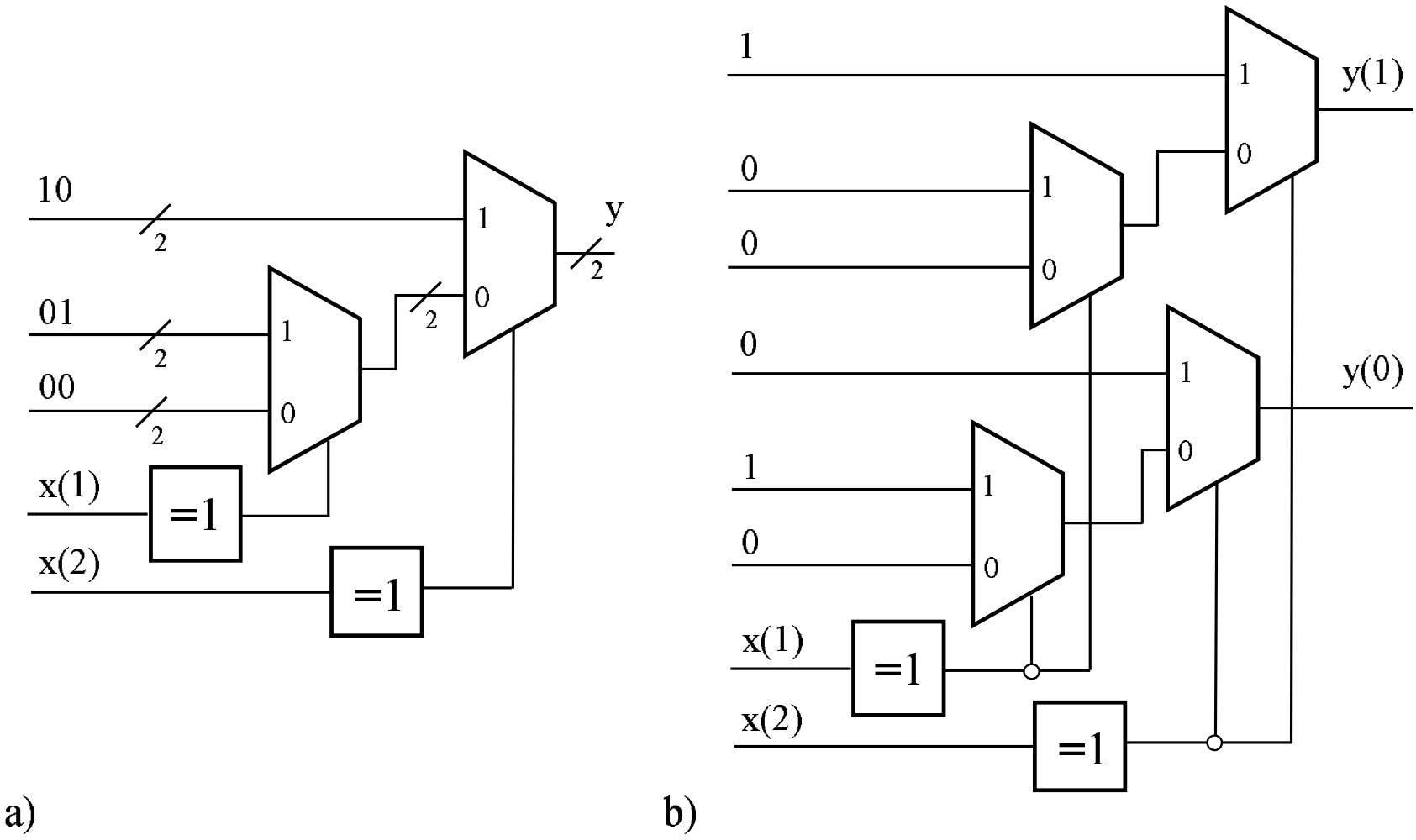


Fig. 2.21

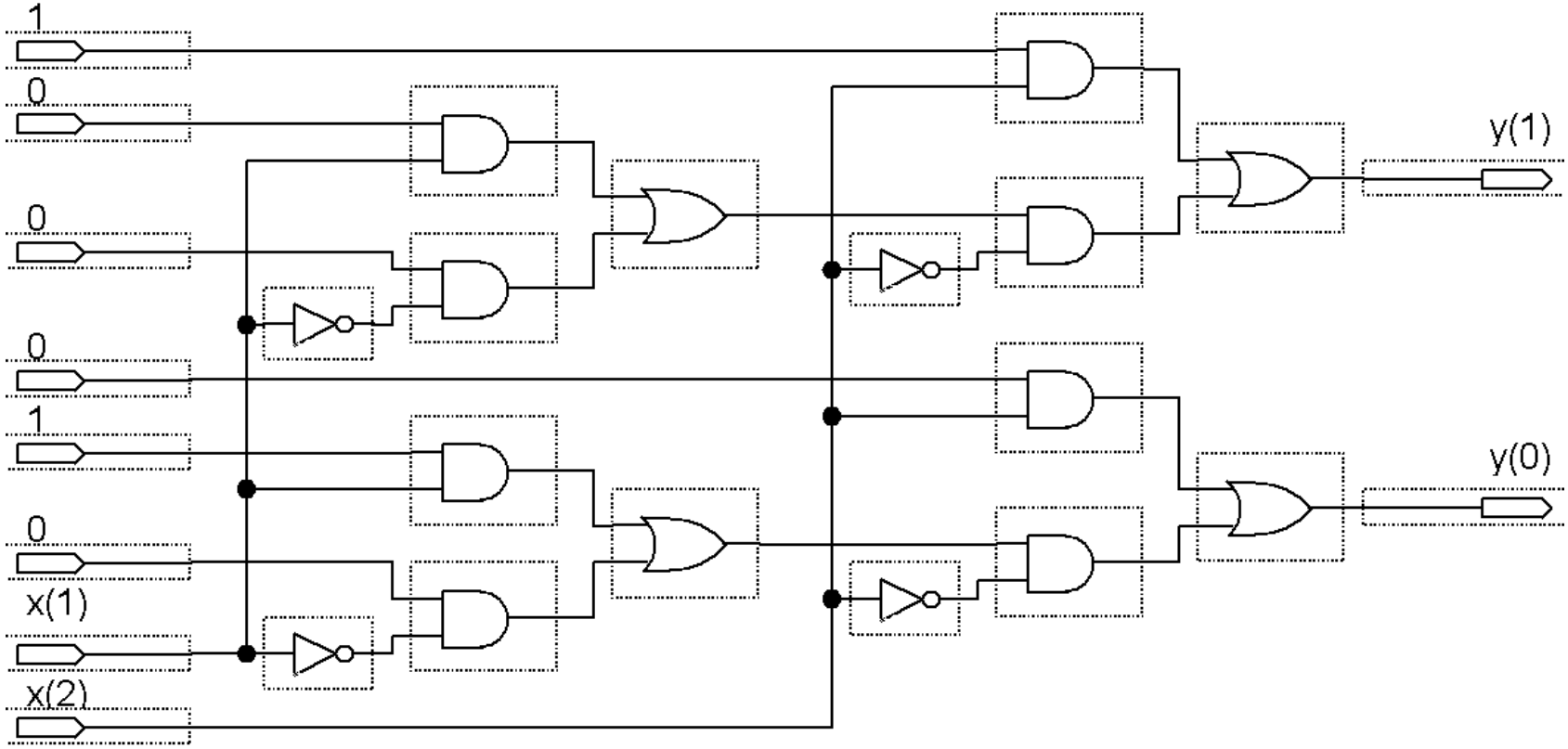


Fig. 2.22

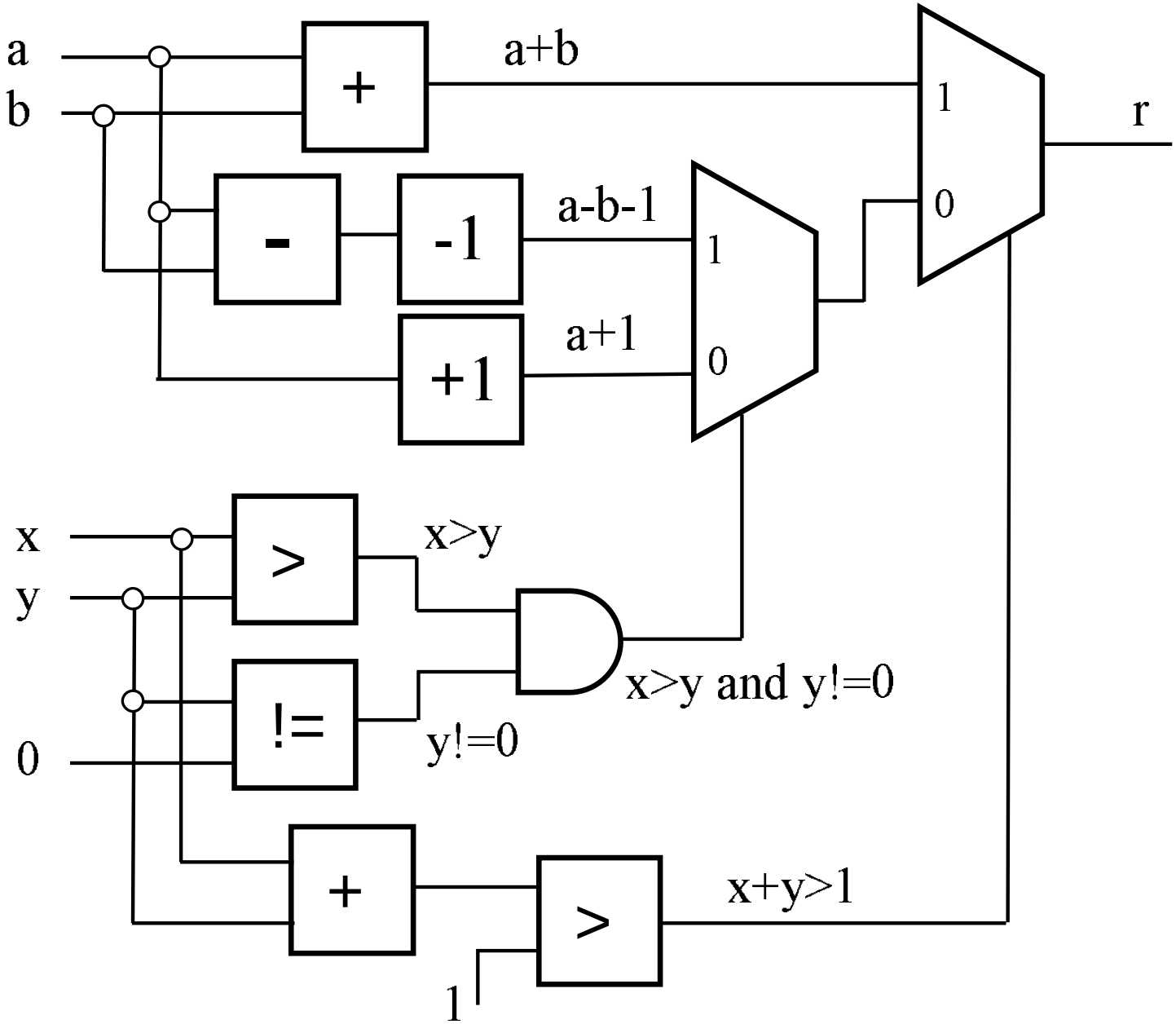


Fig. 2.23

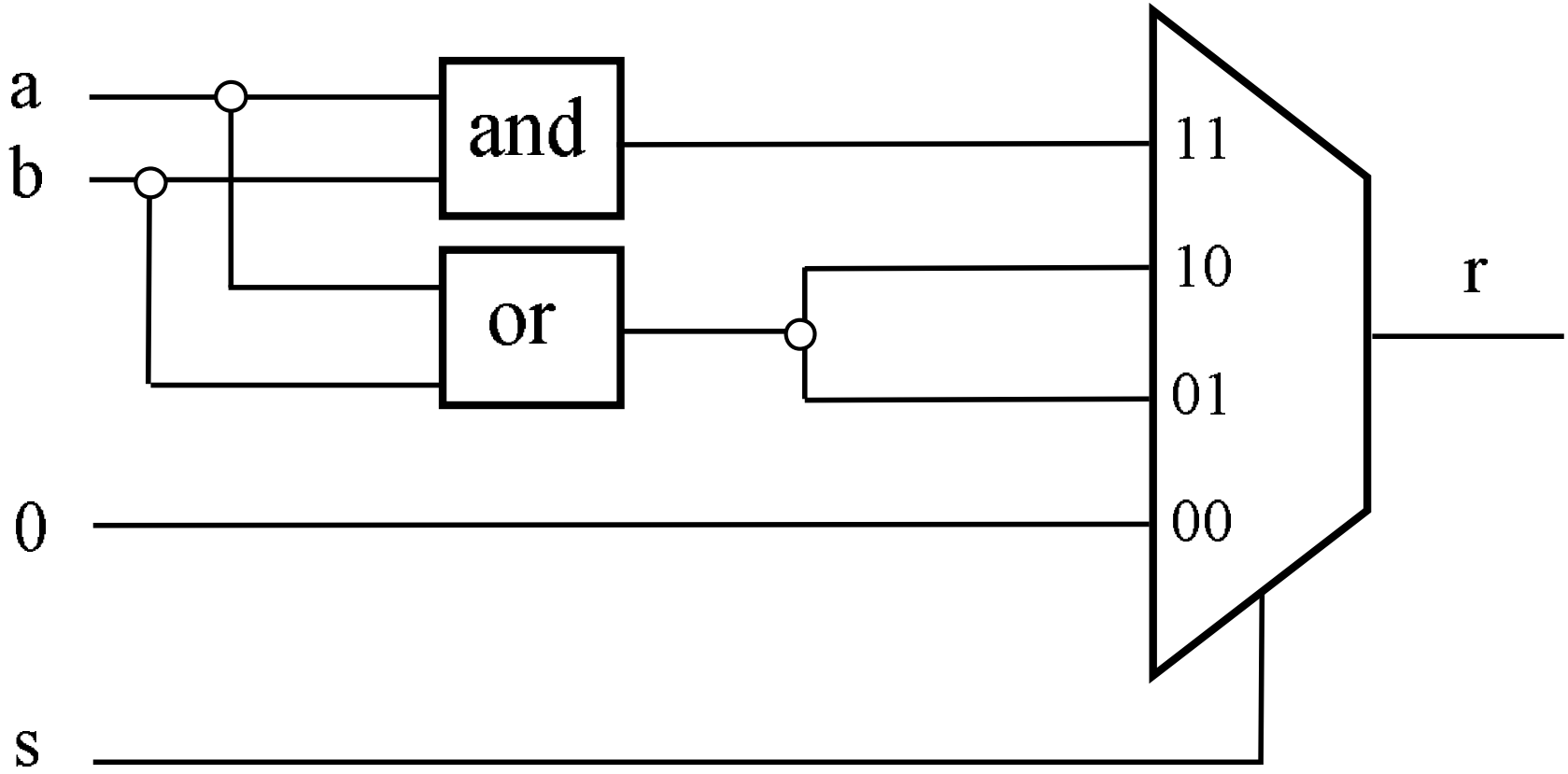


Fig. 2.24

```
if boolean_expr then
    a <= valor_expr_a_1;
    b <= valor_expr_b_1;
else
    a <= valor_expr_a_2;
    b <= valor_expr_b_2;
end if;
```

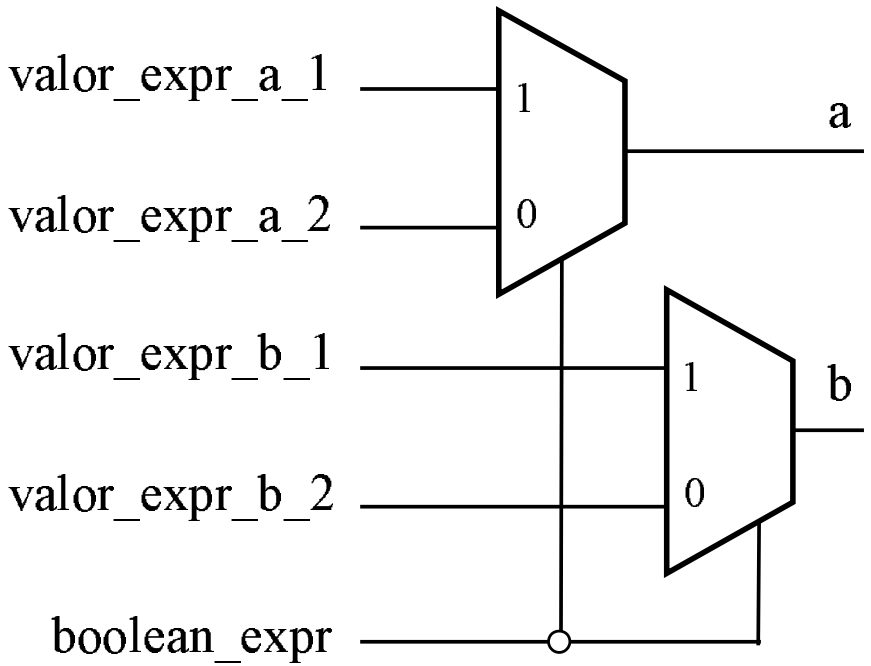


Fig. 2.25

```
if boolean_expr_1 then
    if boolean_expr_2 then
        a <= valor_expr_a_1;
    else
        a <= valor_expr_a_2;
    end if;
else
    if boolean_expr_3 then
        a <= valor_expr_a_3;
    else
        a <= valor_expr_a_4;
    end if;
end if;
```

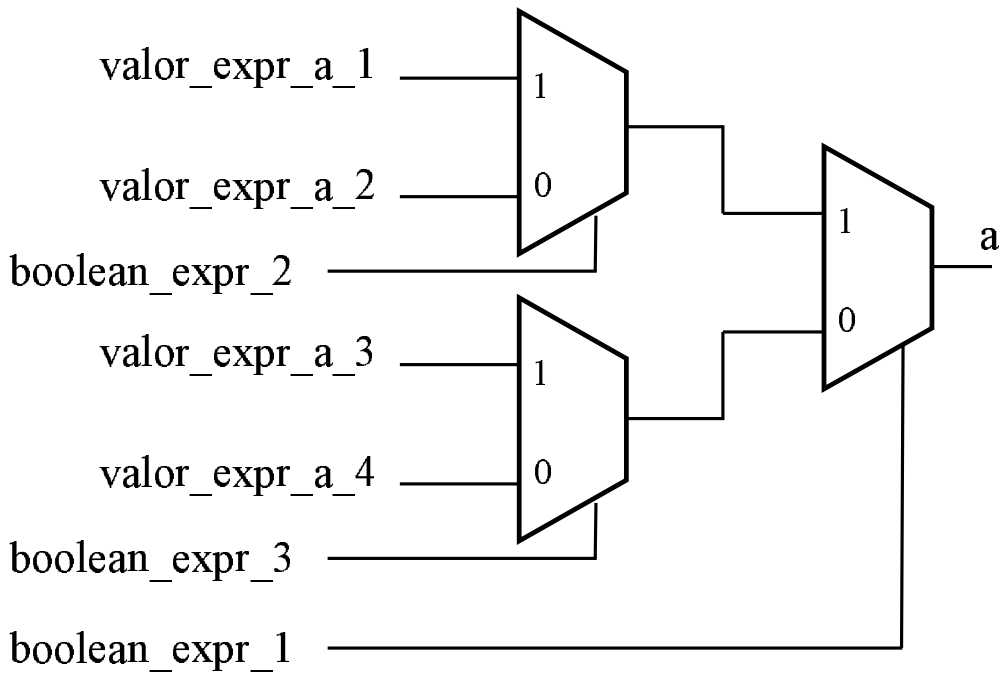


Fig. 2.26

```
case expr_case is
    when c0 =>
      a <= valor_expr_a_0;
      b <= valor_expr_b_0;
    when c1 =>
      a <= valor_expr_a_1;
      b <= valor_expr_b_1;
    when others =>
      a <= valor_expr_a_n;
      b <= valor_expr_b_n;
  end case;
```

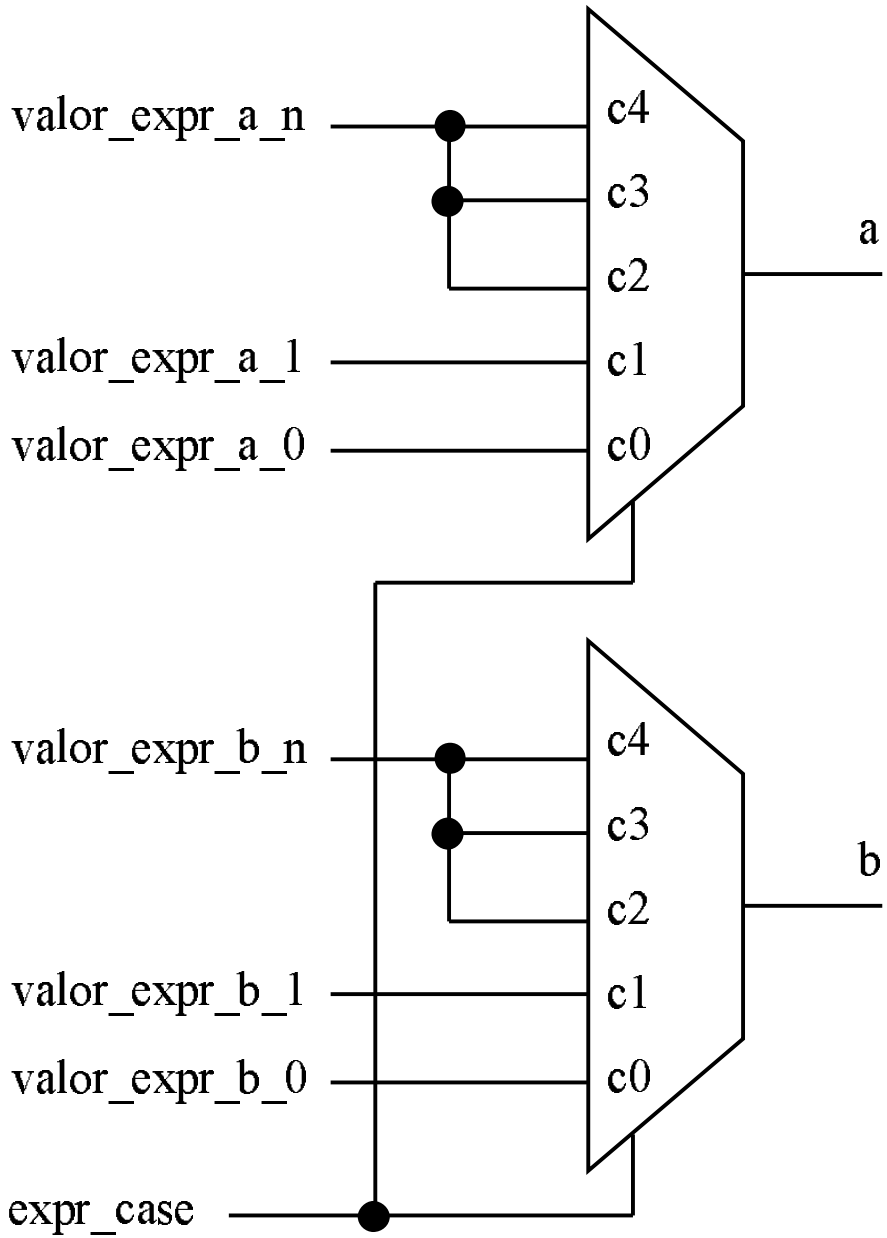


Fig. 2.27

Operador	Significado	Tipo del operando a	Tipo del operando b	Tipo del resultado
a ** b	Exponenciación	integer	integer	integer
abs a	Valor absoluto	integer		integer
not a	NOT lógico	boolean,bit, bit_vector		boolean,bit, bit_vector
a * b a / b a mod b a rem b	Multiplicación División Módulo Resto	integer	integer	integer
+ a - a	Identidad Negación	integer		integer
a + b a - b	Suma Resta	integer	integer	integer
a & b	Concatenación	vector, elemento	vector, elemento	vector
a sll b a srl b a sla b a sra b a rol b a ror b	Desplaz. lógico izqda Desplaz. lógico dcha Desplaz. aritmético izqda Desplaz. aritmético dcha Rotación lógica izqda Rotación lógica dcha	bit_vector	integer	bit_vector
a = b a /= b	Comprueba igualdad Comprueba desigualdad	cualquiera	mismo que a	boolean
a < b a <= b a > b a >= b	Menor que Menor o igual que Mayor que Mayor o igual que	escalar, vector	mismo que a	boolean
a and b a or b a nand b a nor b a xor b a xnor b	AND lógica OR lógica NAND lógica NOR lógica OR exclusiva lógica NOR exclusiva lógica	boolean,bit, bit_vector	mismo que a	mismo que a

Tabla 2.1

Precedencia	Operadores
Mayor	`**` `abs` `not`
	`*` `/` `mod` `rem`
	`+` `-` (identidad y negación)
	`&` `+` `-` (suma y resta)
	`sll` `srl` `sla` `sra` `rol` `ror`
	`=` `/=` `<` `<=` `>` `>=`
Menor	`and` `or` `nand` `nor` `xor` `xnor`

Tabla 2.2

Conversión de:	a:	Función
std_logic	bit	`to_bit(arg)`
bit	std_logic	`to_stdulogic(arg)`
std_logic_vector	bit_vector	`to_bitvector(arg)`
bit_vector	std_logic_vector	`to_stdlogicvector(arg)`

Tabla 2.3

Conversión de:	a:	Función
std_logic_vector, signed	unsigned	`unsigned(arg)`
std_logic_vector, unsigned	signed	`signed(arg)`
unsigned, signed	std_logic_vector	`std_logic_vector(arg)`
unsigned, signed	integer	`to_integer(arg)`
integer, natural	unsigned	`to_unsigned(arg,size)`
integer	signed	`to_signed(arg,size)`
integer	std_logic_vector	integer → unsigned/signed → std_logic_vector
std_logic_vector	integer	std_logic_vector → unsigned/signed → integer

Tabla 2.4

Atributo	Significado
`<tipo>'`	Indica que el argumento es del tipo `<tipo>`. Por ejemplo, **unsigned**`'("00")` establece que `"00"` se usa como **unsigned**, en lugar de como **bit_vector** o **std_logic_vector**.
`<vector1D>'range`	Rango de índices de un vector $1D$
`<vector1D>'reverse_range`	Rango de índices del vector $1D$, en orden inverso
`<vector1D>'length`	Número de componentes del vector $1D$
`<vector1D>'low`	Índice más pequeño del vector
`<vector1D>'high`	Índice más grande del vector
`<vector1D>'left`	Índice del componente más a la izquierda
`<vector1D>'right`	Índice del componente más a la derecha
`<señal>'stable`	*true* mientras no hay cambios en `<señal>`
`<señal>'event`	*true* sólo cuando hay un cambio en `<señal>`

Tabla 2.5

TEMA 3

Simulación del código VHDL

```
library IEEE;
use IEEE.std_logic_1164.all;

entity resF is
end entity resF;

architecture resF of resF is
  signal s : std_logic;
begin
  p1 : process
  begin
    s <= '1';
    wait;
  end process p1;

  p2 : process
  begin
    s <= '0';
    wait;
  end process p2;
end architecture resF;
```

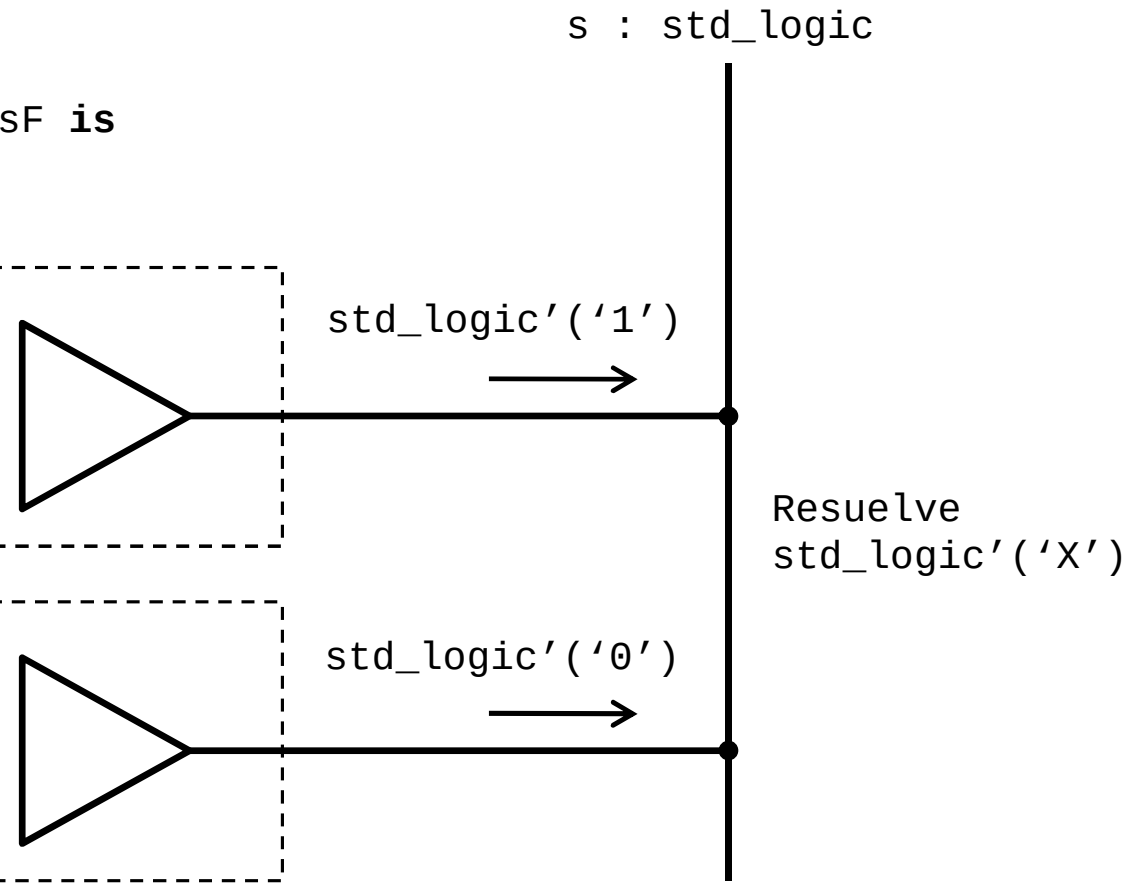


Fig. 3.1

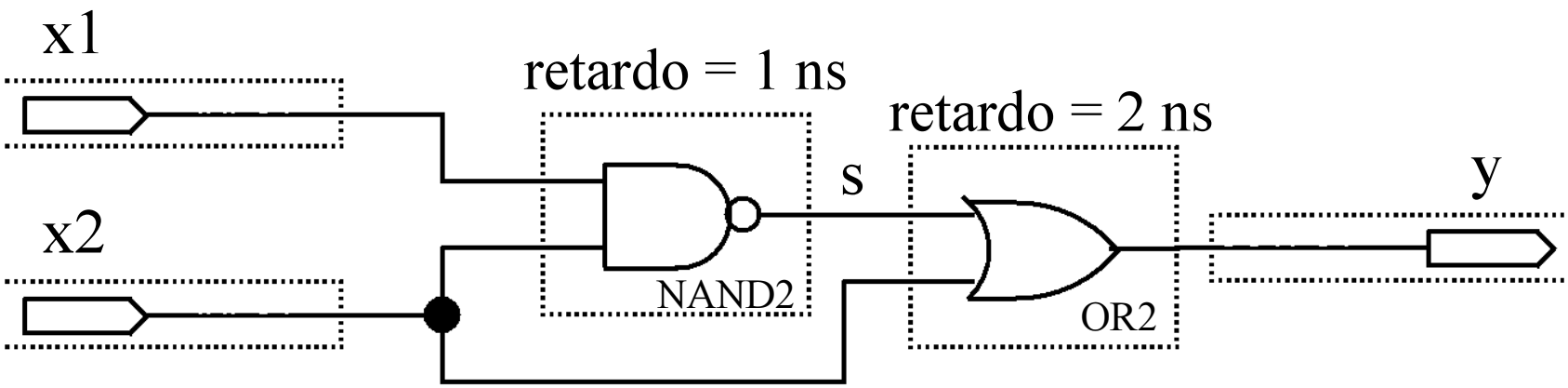


Fig. 3.2

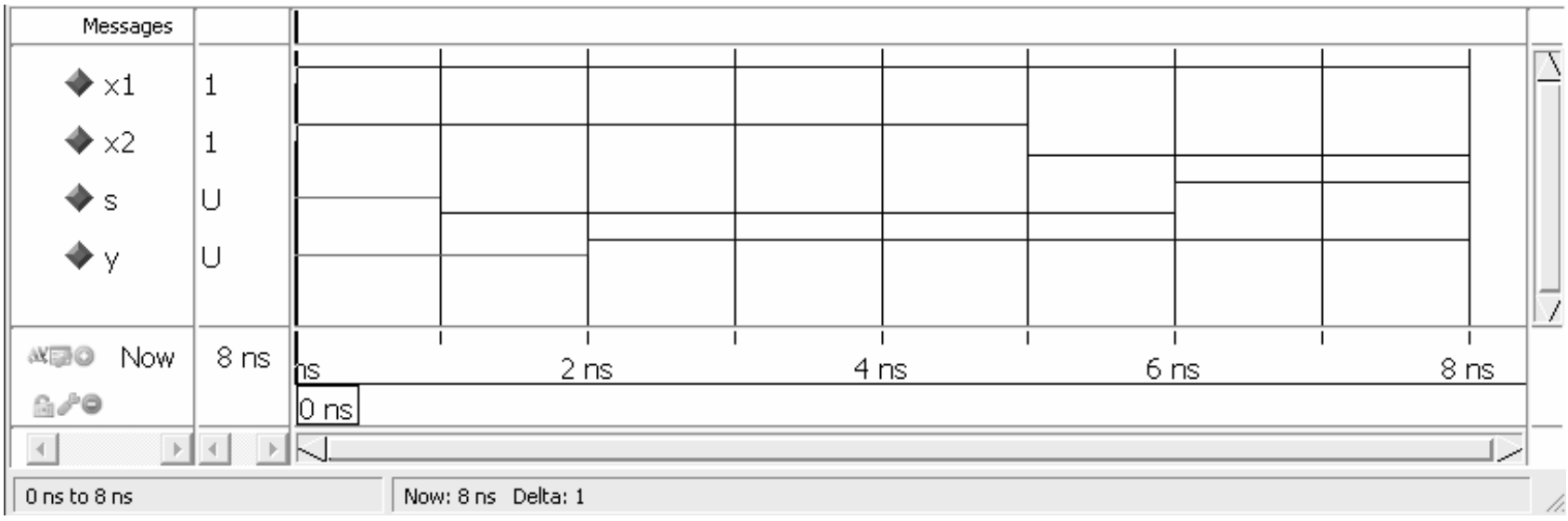

Messages
x1
x2
s
y
1
1
U
U
Now
8 ns
ns
2 ns
4 ns
6 ns
8 ns
0 ns
0 ns to 8 ns
Now: 8 ns Delta: 1

Fig. 3.3

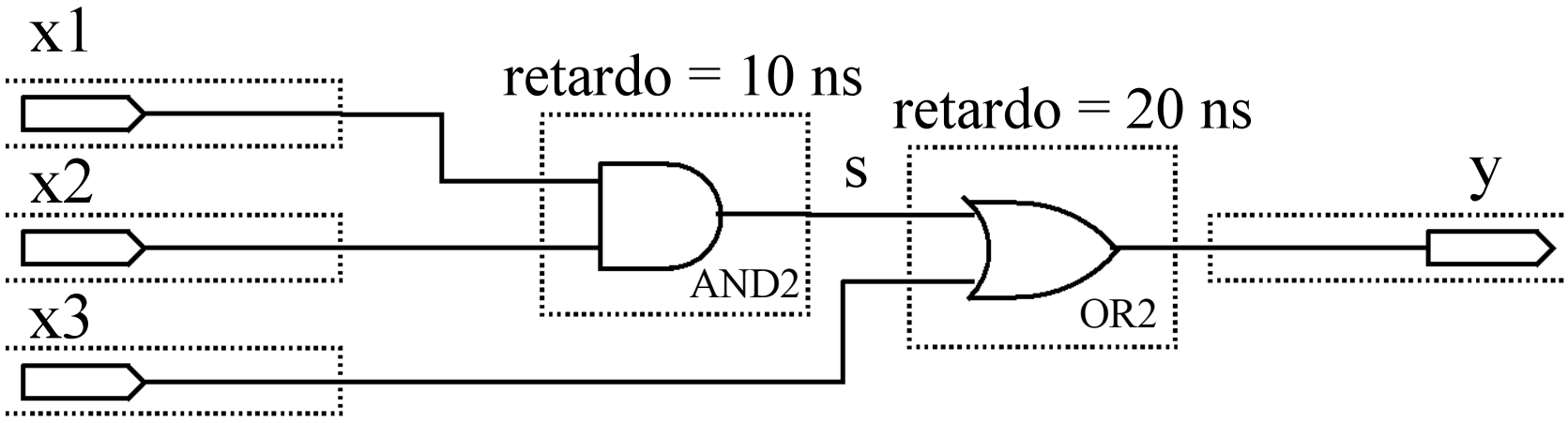


Fig. 3.4

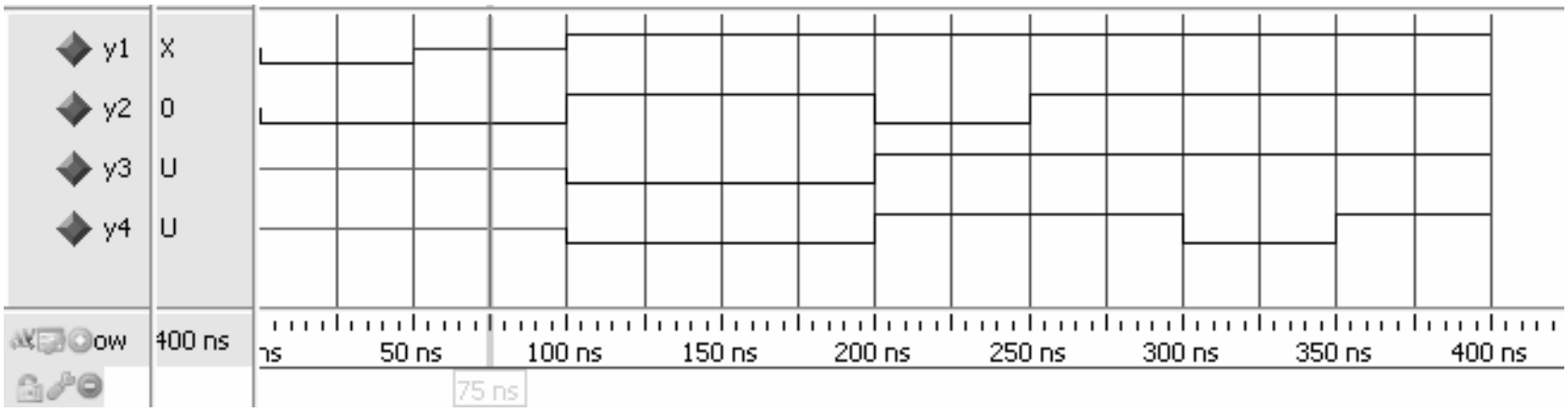

y1
X
y2
0
y3
U
y4
U
ow
400 ns
ns
50 ns
100 ns
150 ns
200 ns
250 ns
300 ns
350 ns
400 ns
75 ns

Fig. 3.5

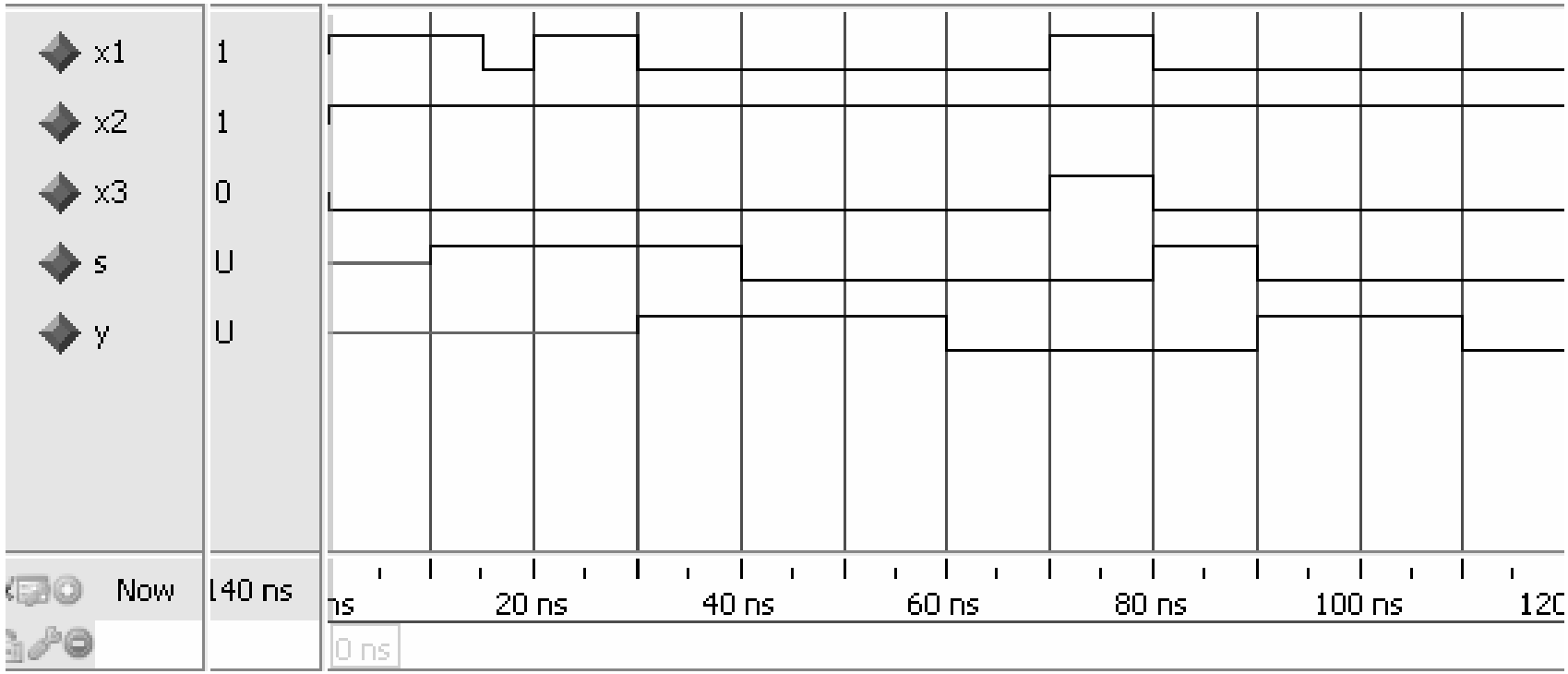

x1
x2
x3
s
y
1
1
0
U
U
Now
140 ns
ns
20 ns
40 ns
60 ns
80 ns
100 ns
120
0 ns

Fig. 3.6

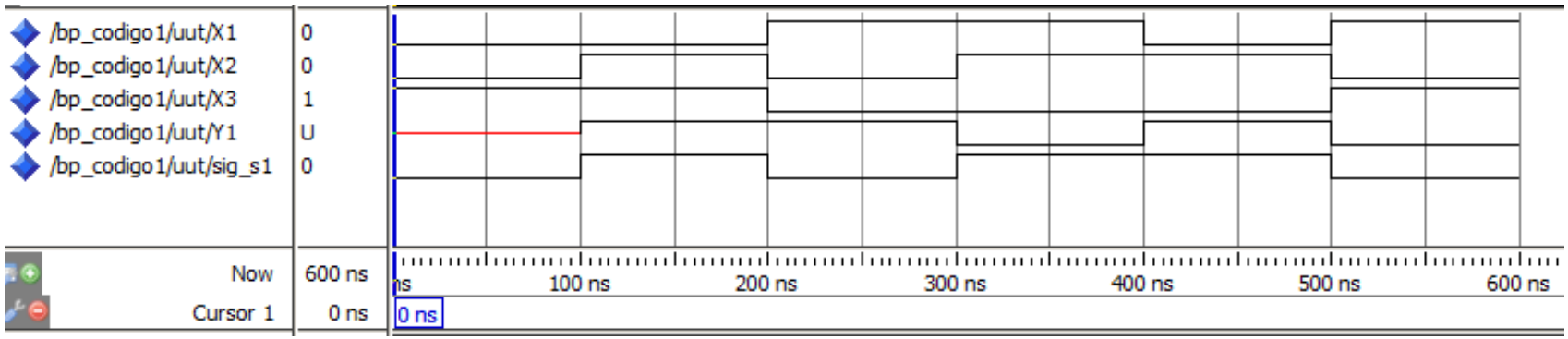

/bp_codigo1/uut/X1
/bp_codigo1/uut/X2
/bp_codigo1/uut/X3
/bp_codigo1/uut/Y1
/bp_codigo1/uut/sig_s1
0
0
1
U
0
Now
600 ns
Cursor 1
0 ns
0 ns
100 ns
200 ns
300 ns
400 ns
500 ns
600 ns

Fig. 3.7

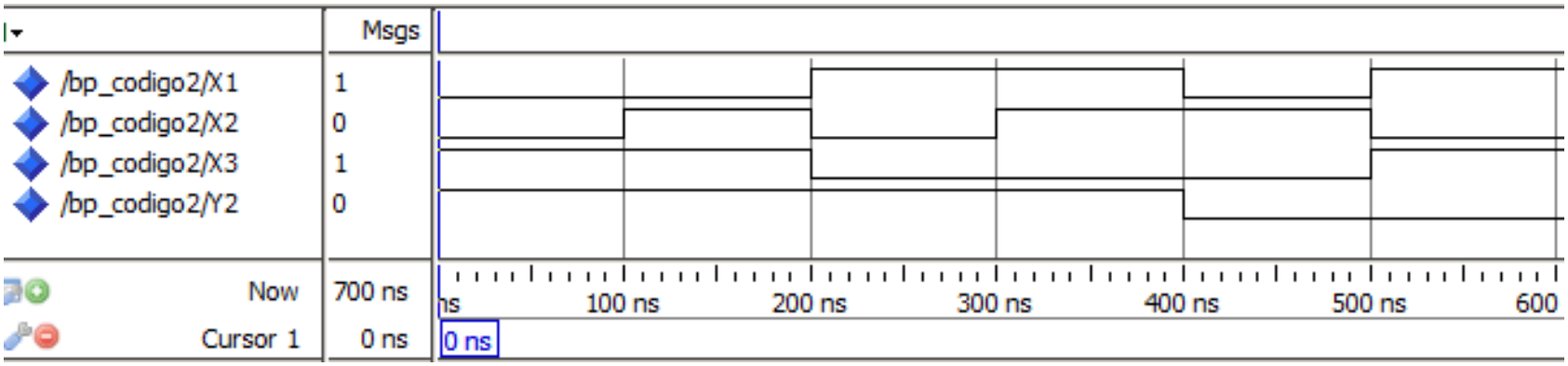

Msgs
/bp_codigo2/X1
/bp_codigo2/X2
/bp_codigo2/X3
/bp_codigo2/Y2
1
0
1
0
Now
700 ns
Cursor 1
0 ns
0 ns
100 ns
200 ns
300 ns
400 ns
500 ns
600

Fig. 3.8

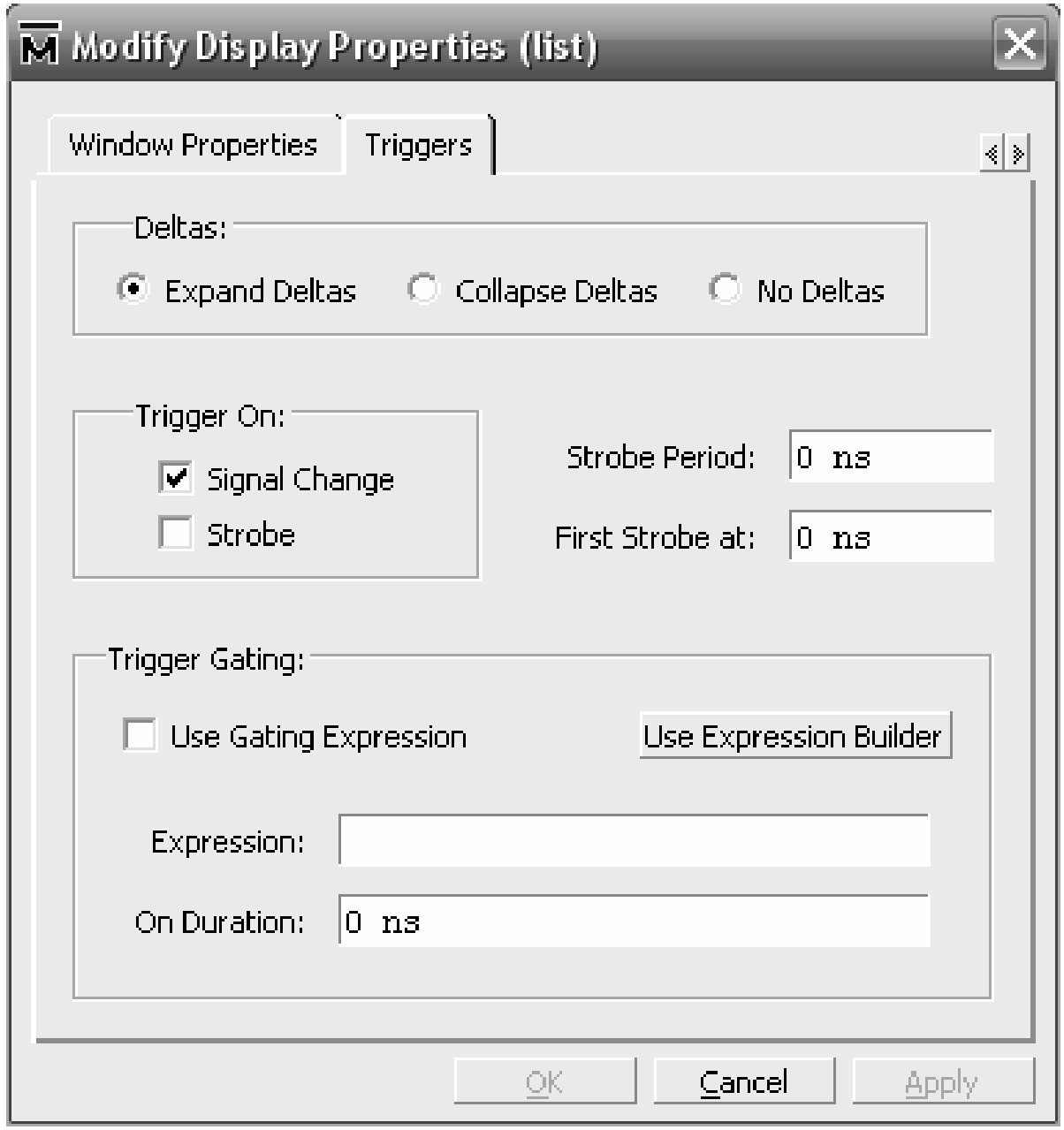

Modify Display Properties (list)
Window Properties
Triggers
Deltas:
Expand Deltas
Collapse Deltas
No Deltas
Trigger On:
Signal Change
Strobe
Strobe Period:
0 ns
First Strobe at:
0 ns
Trigger Gating:
Use Gating Expression
Use Expression Builder
Expression:
On Duration:
0 ns
OK
Cancel
Apply

Fig. 3.9

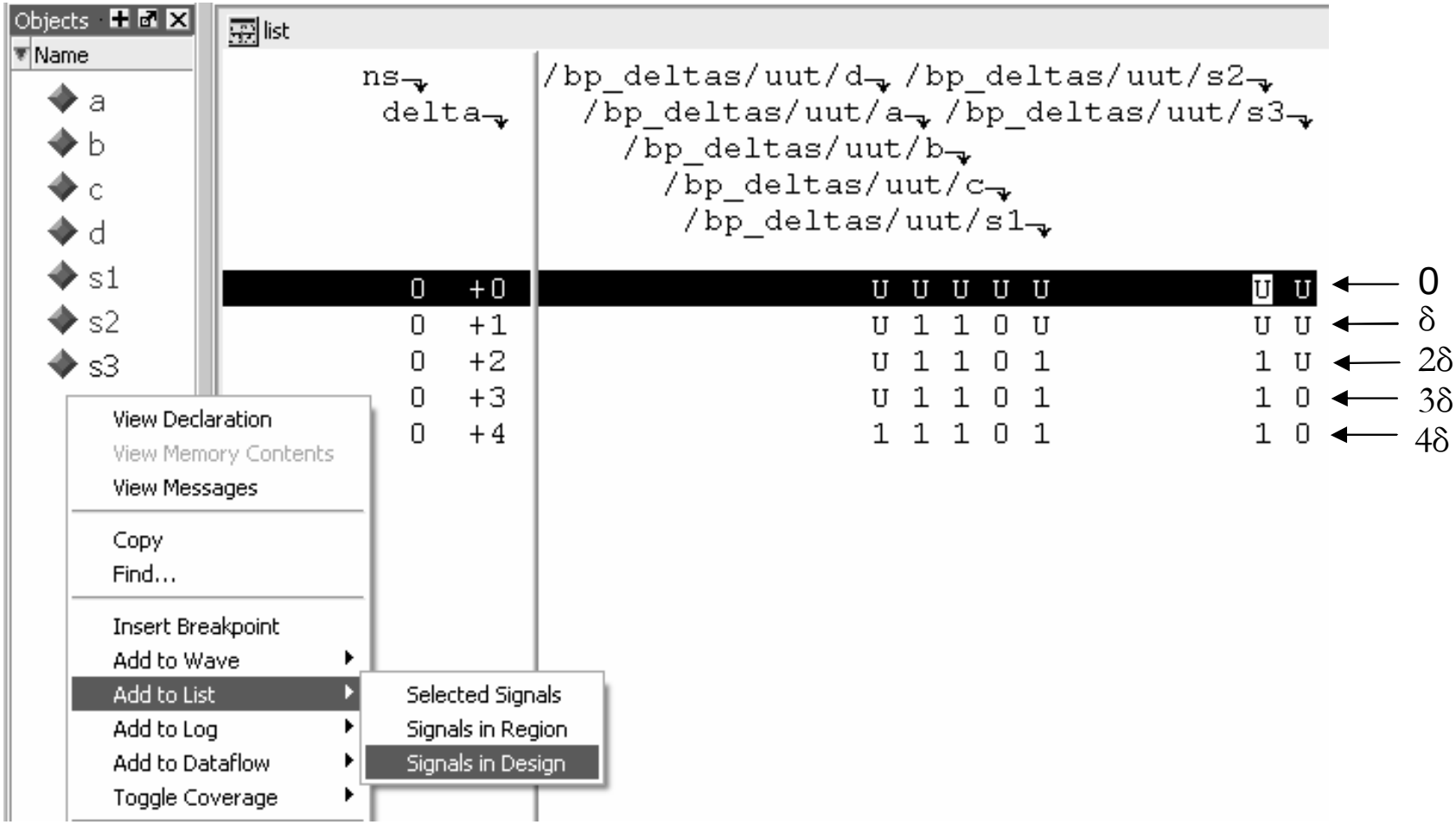

Objects
Name
a
b
c
d
s1
s2
s3
list
ns
delta
/bp_deltas/uut/d
/bp_deltas/uut/s2
/bp_deltas/uut/a
/bp_deltas/uut/s3
/bp_deltas/uut/b
/bp_deltas/uut/c
/bp_deltas/uut/s1
0 +0 U U U U U U U
0 +1 U 1 1 0 U U U
0 +2 U 1 1 0 1 1 U
0 +3 U 1 1 0 1 1 0
0 +4 1 1 1 0 1 1 0
0
δ
2δ
3δ
4δ
View Declaration
View Memory Contents
View Messages
Copy
Find...
Insert Breakpoint
Add to Wave
Add to List
Add to Log
Add to Dataflow
Toggle Coverage
Selected Signals
Signals in Region
Signals in Design

Fig. 3.10

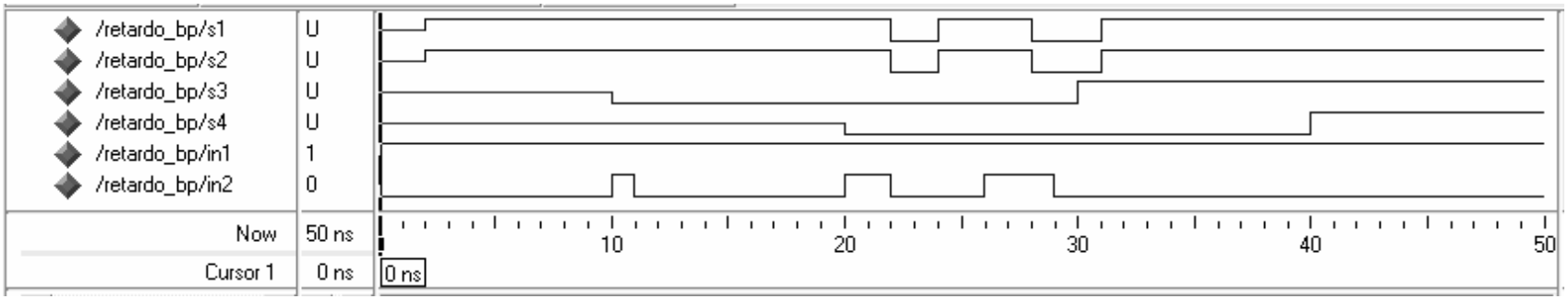

/retardo_bp/s1
/retardo_bp/s2
/retardo_bp/s3
/retardo_bp/s4
/retardo_bp/in1
/retardo_bp/in2
U
U
U
U
1
0
Now
50 ns
Cursor 1
0 ns
0 ns
10
20
30
40
50

Fig. 3.11

	signal	variable
Asignación	<=	:=
Qué representa	Interconexiones en el circuito	Información local
Visibilidad	Puede ser global (visible desde todo el código)	Local (visible sólo desde el correspondiente bloque **process**, **function** o **procedure**)
Comportamiento	El nuevo valor es asignado transcurrido un retardo, que por defecto vale δ.	El nuevo valor es asignado inmediatamente
Uso	En **package**, **entity** y **architecture**. En una **entity**, los puertos son por defecto señales	Sólo en código secuencial. Es decir, dentro de **process**, **function** o **procedure**

Tabla 3.1

	U	**X**	**0**	**1**	**Z**	**W**	**L**	**H**	**–**
U	U	U	U	U	U	U	U	U	U
X	U	X	X	X	X	X	X	X	X
0	U	X	0	X	0	0	0	0	X
1	U	X	X	1	1	1	1	1	X
Z	U	X	0	1	Z	W	L	H	X
W	U	X	0	1	W	W	W	W	X
L	U	X	0	1	L	W	L	W	X
H	U	X	0	1	H	W	W	H	X
–	U	X	X	X	X	X	X	X	X

Tabla 3.2

`s'event`	Función booleana que devuelve *true* cuando se produce un evento en la señal. El resto del tiempo devuelve *false*. Ejemplos:
	`if clk'event then ...`
	En el preciso instante en que `clk` cambia de valor, entonces ...
	`wait until clk'event and clk='1';`
	Suspende el proceso hasta que el valor de `clk` pasa a ser `'1'`

`s'active`	Función booleana que devuelve *true* en el instante en el cual se produce una transacción a la señal. El resto del tiempo devuelve *false*. Ejemplos:
	`if dato'active then ...`
	En el preciso instante en que se asigna valor a `dato`, entonces ...
	`wait until dato'active;`
	Suspende el proceso hasta que se asigna valor a `dato`

`s'transaction`	Crea una señal implícita de tipo **bit**, cuyo valor va cambiando `'0'`, `'1'`, `'0'`, ... cada vez que se produce una transacción a la señal `s`. Ejemplo: `wait on dato'transaction;` Se suspende el proceso hasta que se asigna valor a `dato`
`s'delayed(T)`	Crea una señal implícita del mismo tipo que `s`, cuyo valor es igual al valor de `s` retrasado `T`. Si se omite `T`, el retardo por defecto es 0, con lo cual es el valor de la señal en el instante δ anterior. Por ejemplo, `s'delayed(4 ns)` en el instante 10 ns es el valor que tenía `s` en el instante 6 ns.

Tabla 3.3

`s'stable(T)`	Crea una señal implícita de tipo booleano, cuyo valor es *true* si no se ha producido un evento en la señal `s` durante las anteriores `T` unidades de tiempo. En caso contrario, vale *false*. Ejemplo: `if dato'stable(100 ns) then ...` Si no se ha producido ningún evento en la señal `dato` durante los últimos 100 ns, entonces ...
`s'quiet(T)`	Crea una señal implícita de tipo booleano, cuyo valor es *true* si no se ha producido ninguna transacción a la señal `s` durante las anteriores `T` unidades de tiempo. En caso contrario, vale *false*. Ejemplo: `if dato'quiet(100 ns) then ...` Si no se ha producido ninguna transacción a la señal `dato` durante los últimos 100 ns, entonces ...
`s'last_event`	Función que devuelve un valor del tipo **time**: el tiempo transcurrido desde que se produjo el último evento en la señal `s`. Si no ha habido evento previo, devuelve el valor máximo del tipo de dato **time** (time'high).
`s'last_active`	Función que devuelve un valor del tipo **time**: el tiempo transcurrido desde que se produjo la última transacción a la señal `s`. Si no ha habido ninguna transacción previa, devuelve el valor máximo del tipo de dato **time** (time'high).
`s'last_value`	Función que devuelve un valor del mismo tipo que `s`: el valor de la señal previo a la última transacción a la señal.

Tabla 3.3 (cont.)

TEMA 4

Diseño de lógica combinacional

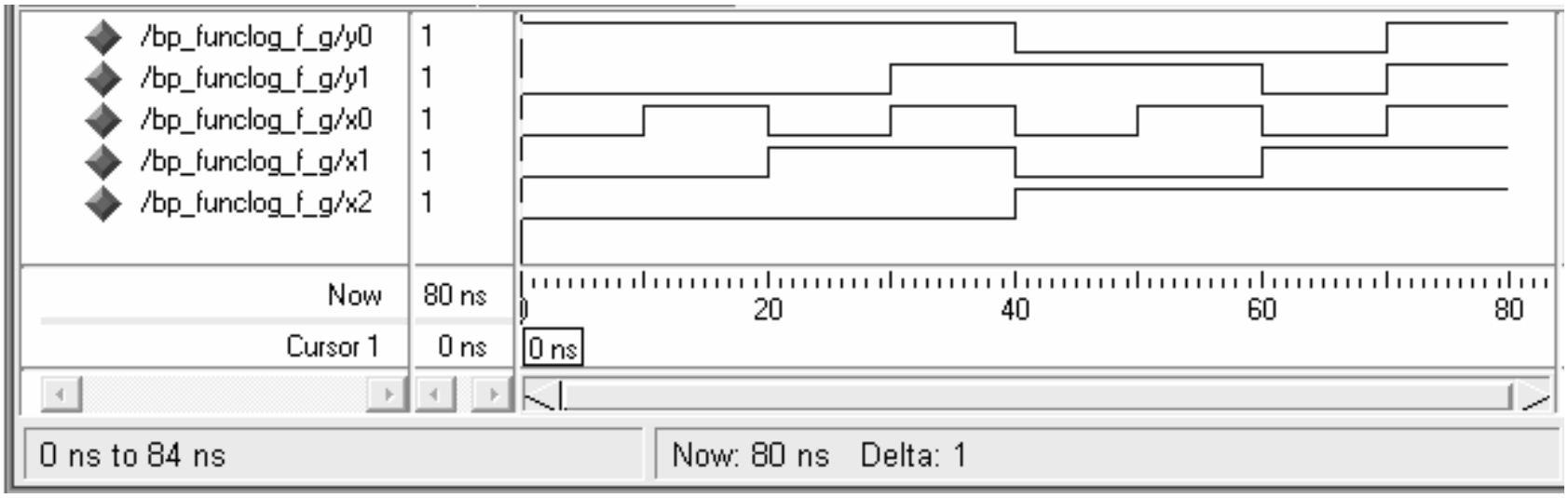

/bp_funclog_f_g/y0
/bp_funclog_f_g/y1
/bp_funclog_f_g/x0
/bp_funclog_f_g/x1
/bp_funclog_f_g/x2
1
1
1
1
1
Now
80 ns
Cursor 1
0 ns
0 ns
0
20
40
60
80
0 ns to 84 ns
Now: 80 ns Delta: 1

Fig. 4.1

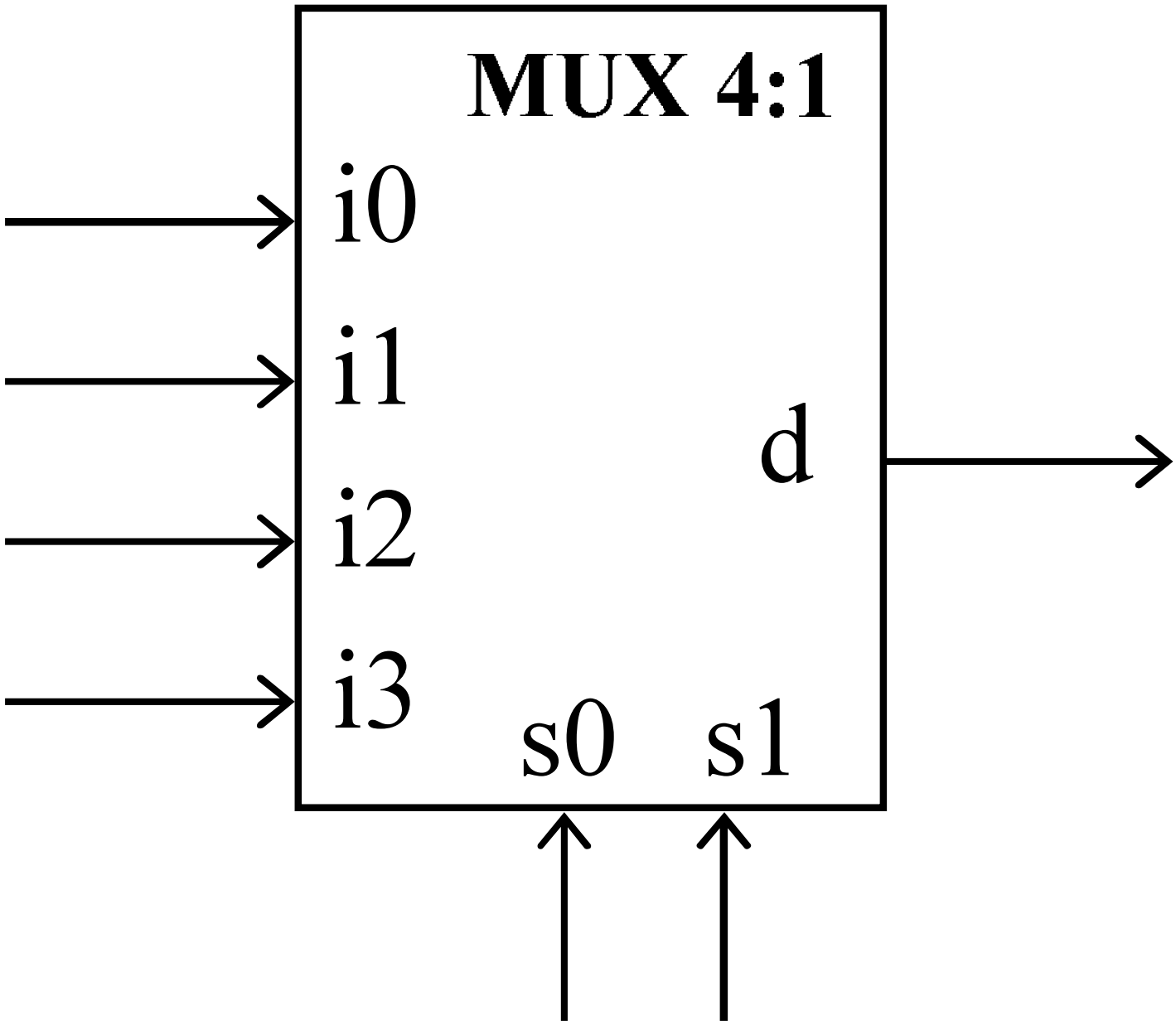
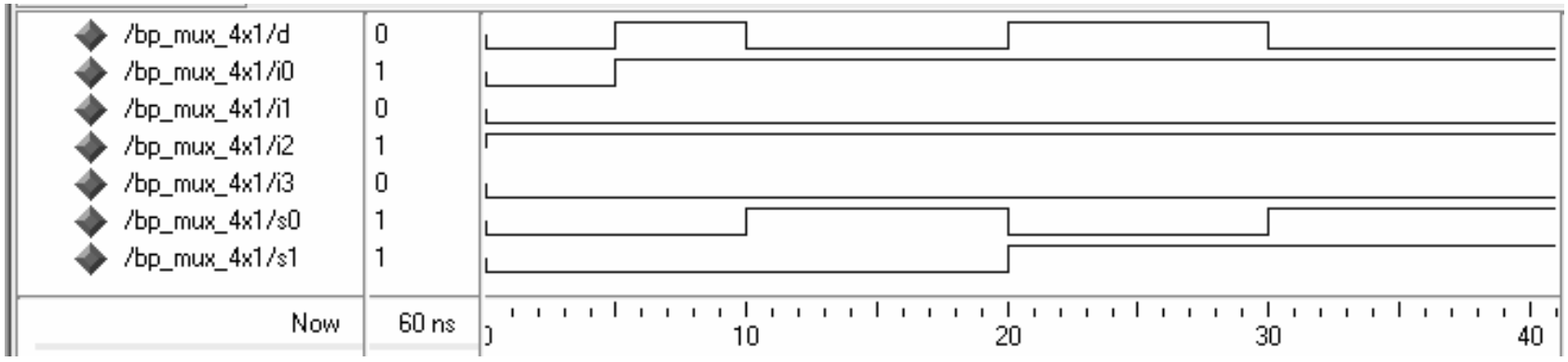
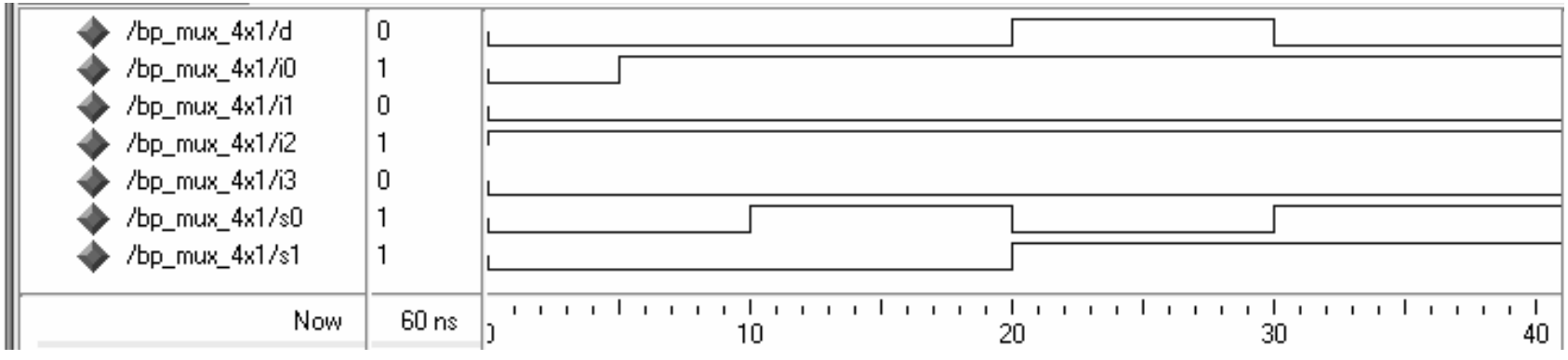


Fig. 4.2


/bp_mux_4x1/d
/bp_mux_4x1/i0
/bp_mux_4x1/i1
/bp_mux_4x1/i2
/bp_mux_4x1/i3
/bp_mux_4x1/s0
/bp_mux_4x1/s1
0
1
0
1
0
1
1
Now
60 ns
10
20
30
40

a)


/bp_mux_4x1/d
/bp_mux_4x1/i0
/bp_mux_4x1/i1
/bp_mux_4x1/i2
/bp_mux_4x1/i3
/bp_mux_4x1/s0
/bp_mux_4x1/s1
0
1
0
1
0
1
1
Now
60 ns
10
20
30
40

b)

Fig. 4.3

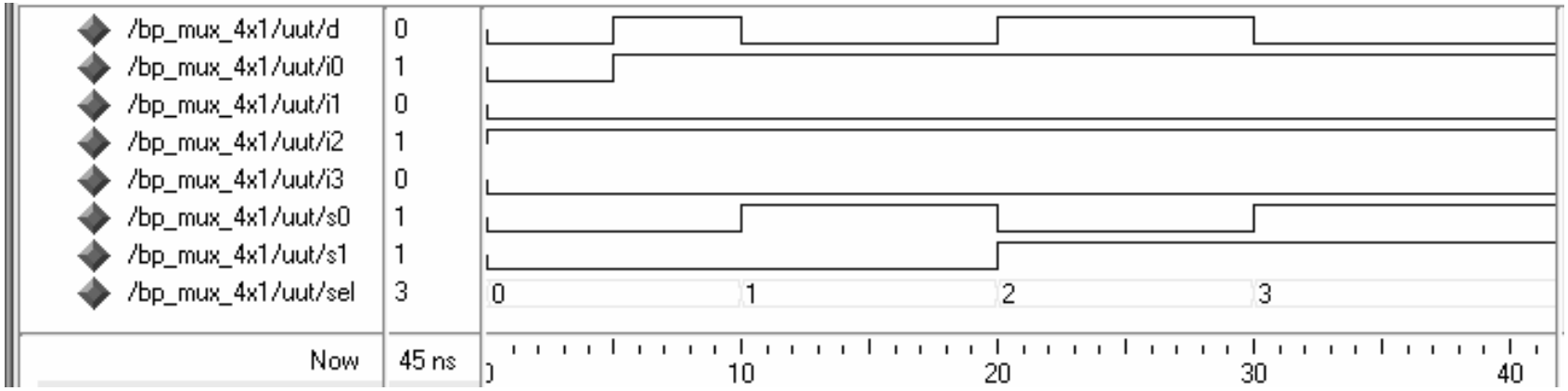

/bp_mux_4x1/uut/d
/bp_mux_4x1/uut/i0
/bp_mux_4x1/uut/i1
/bp_mux_4x1/uut/i2
/bp_mux_4x1/uut/i3
/bp_mux_4x1/uut/s0
/bp_mux_4x1/uut/s1
/bp_mux_4x1/uut/sel
0
1
0
1
0
1
1
3
0
1
2
3
Now
45 ns
0
10
20
30
40

Fig. 4.4

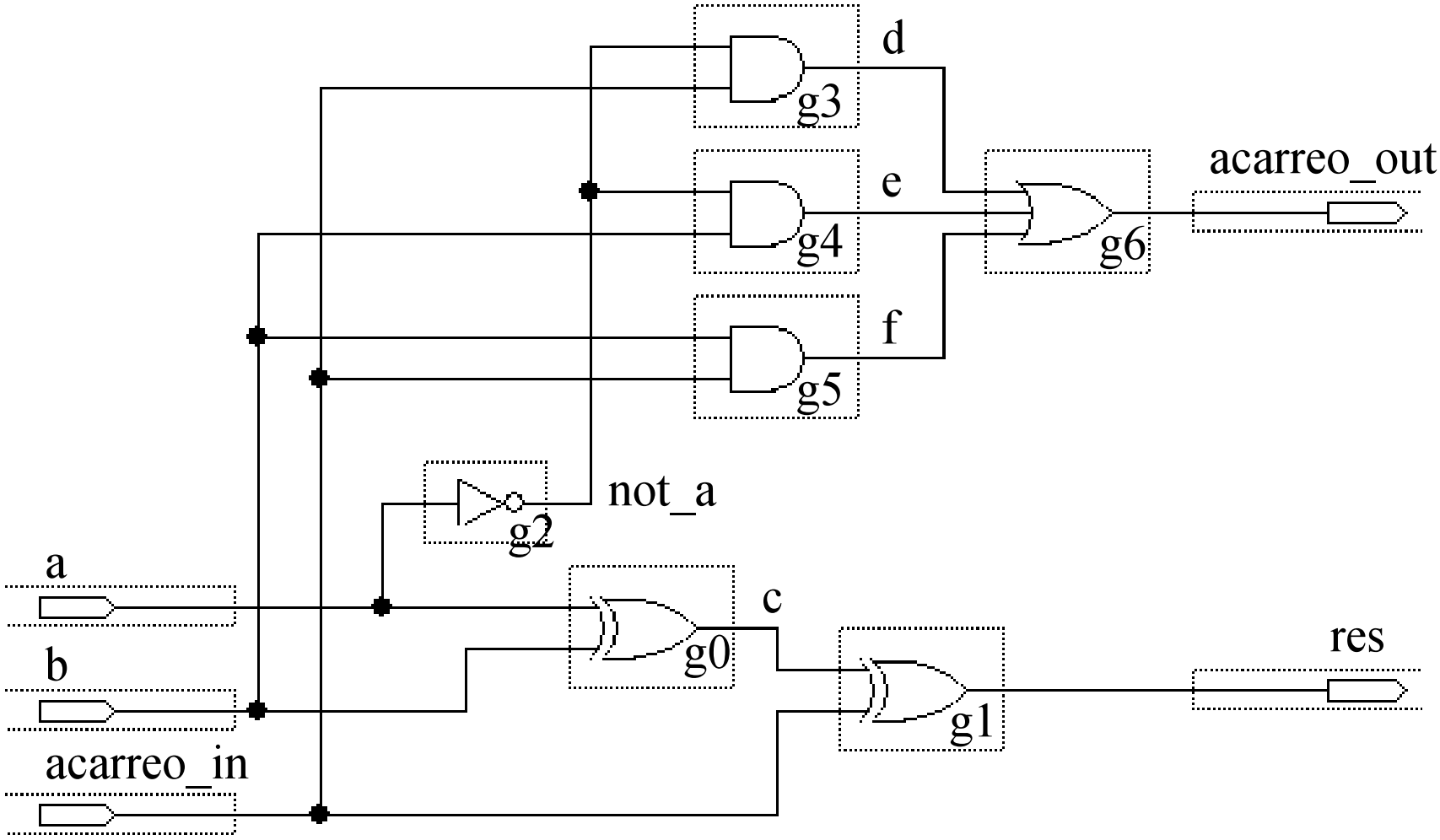


Fig. 4.5

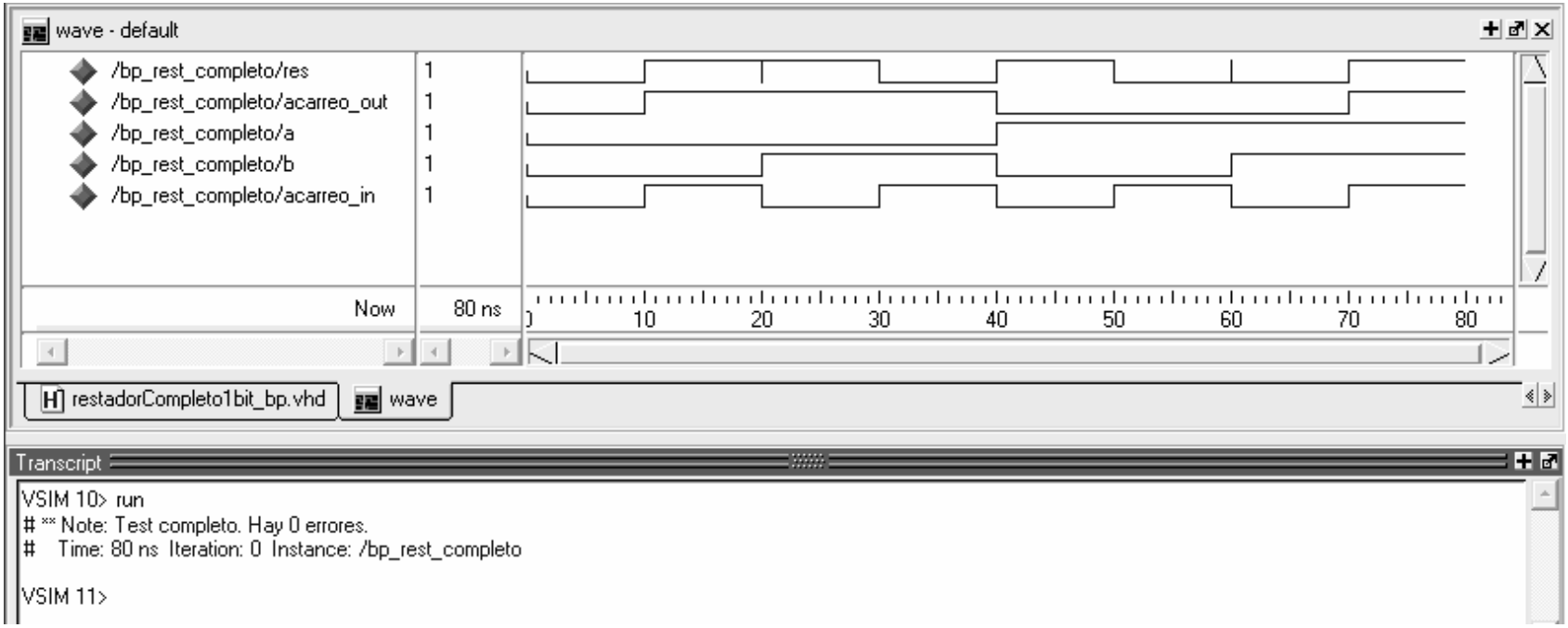

wave - default
/bp_rest_completo/res
/bp_rest_completo/acarreo_out
/bp_rest_completo/a
/bp_rest_completo/b
/bp_rest_completo/acarreo_in
1
1
1
1
1
Now
80 ns
10
20
30
40
50
60
70
80
restadorCompleto1bit_bp.vhd
wave
Transcript
VSIM 10> run
** Note: Test completo. Hay 0 errores.
Time: 80 ns Iteration: 0 Instance: /bp_rest_completo
VSIM 11>

Fig. 4.6

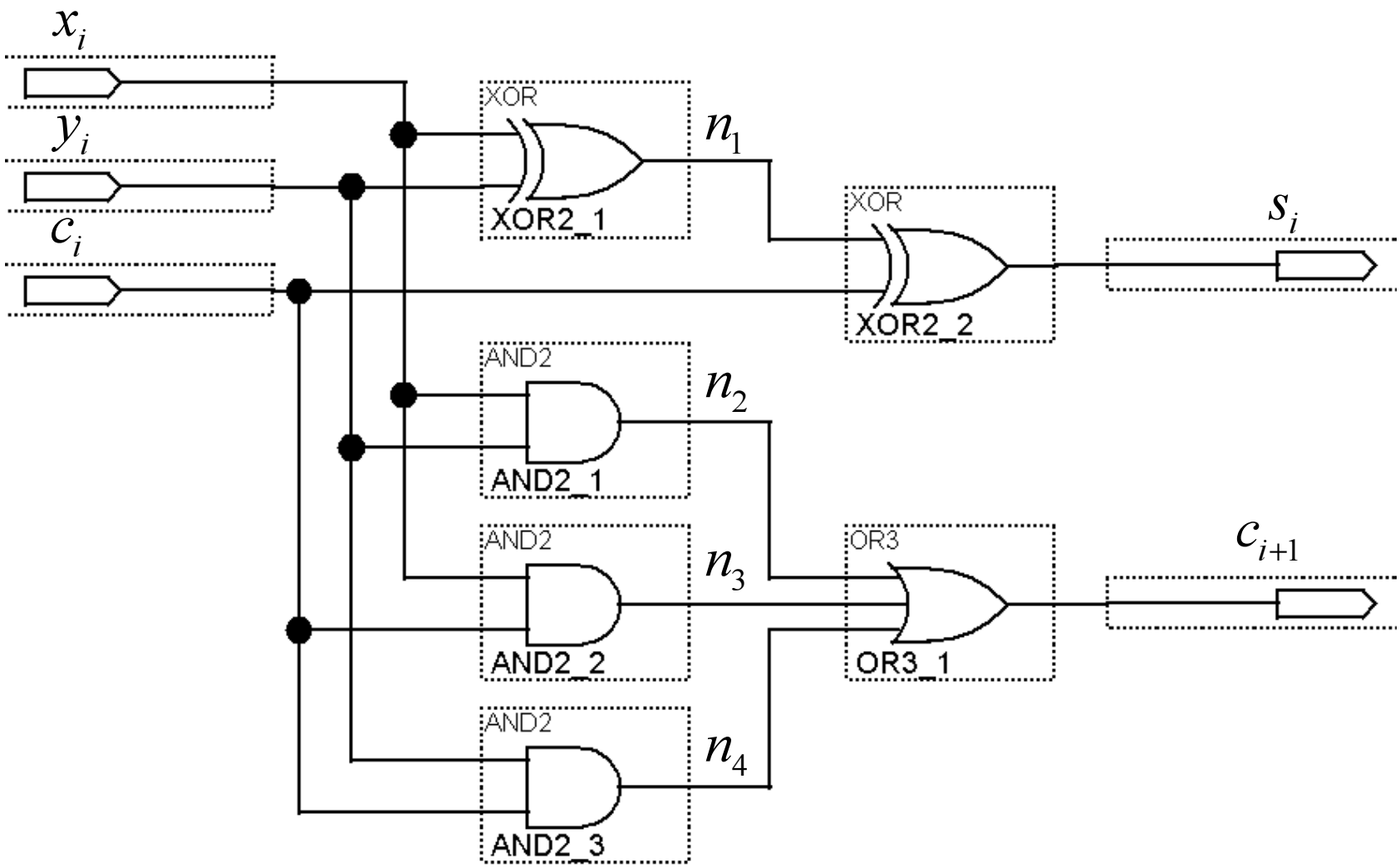

x_i
y_i
c_i
XOR
XOR2_1
n_1
XOR
XOR2_2
s_i
AND2
AND2_1
n_2
AND2
AND2_2
n_3
OR3
OR3_1
c_{i+1}
AND2
AND2_3
n_4

Fig. 4.7

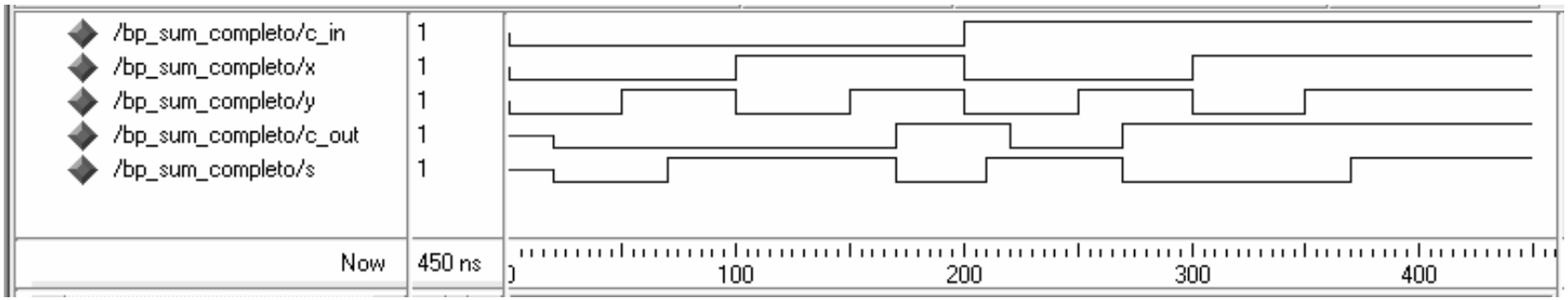

/bp_sum_completo/c_in
/bp_sum_completo/x
/bp_sum_completo/y
/bp_sum_completo/c_out
/bp_sum_completo/s
1
1
1
1
1
Now
450 ns
100
200
300
400

Fig. 4.8

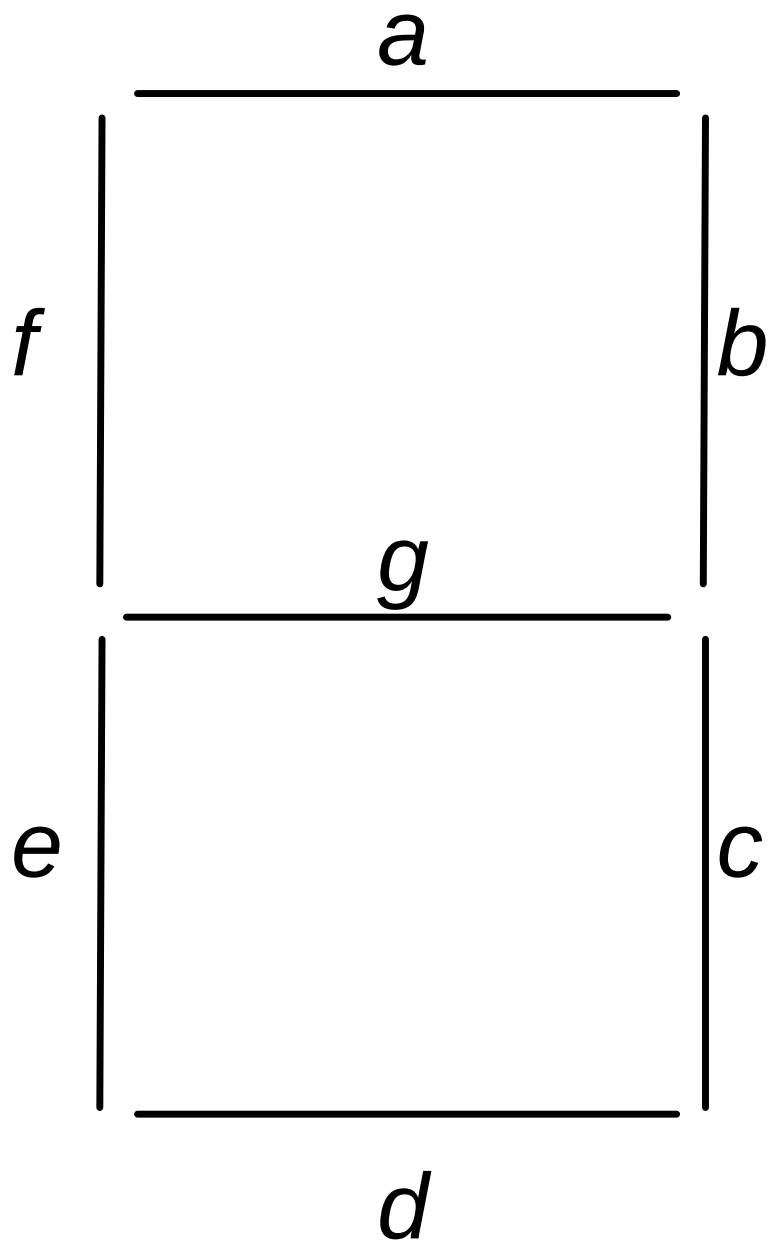


Fig. 4.9

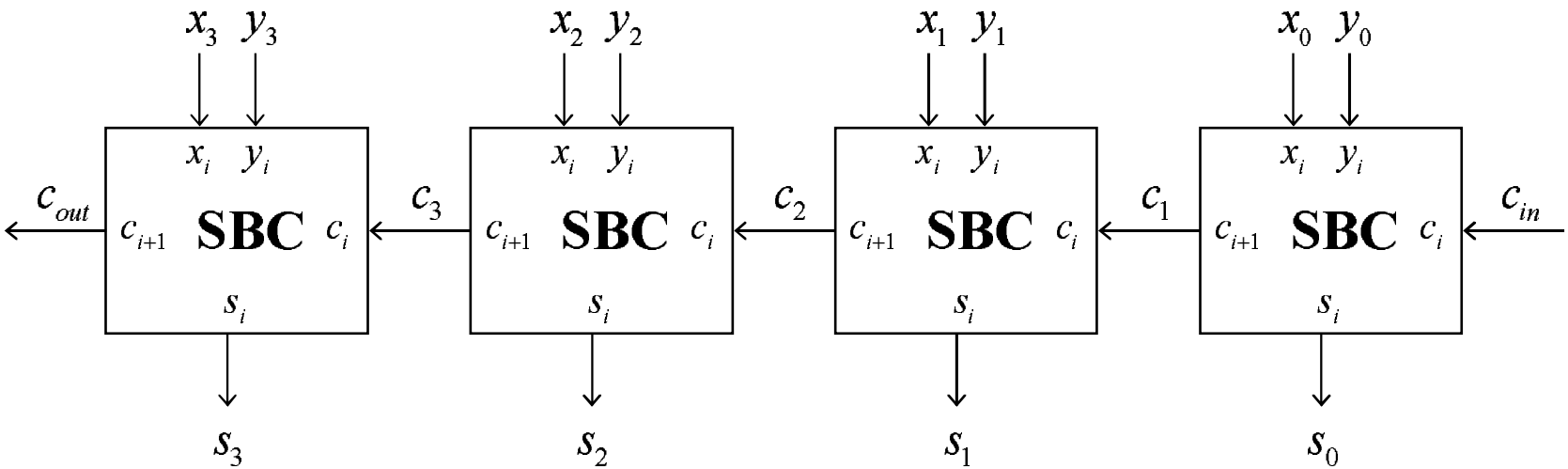

x_3 y_3
x_2 y_2
x_1 y_1
x_0 y_0
x_i y_i
c_{out}
c_{i+1} SBC c_i
c_3
c_2
c_1
c_{in}
s_i
s_3
s_2
s_1
s_0

Fig. 4.10

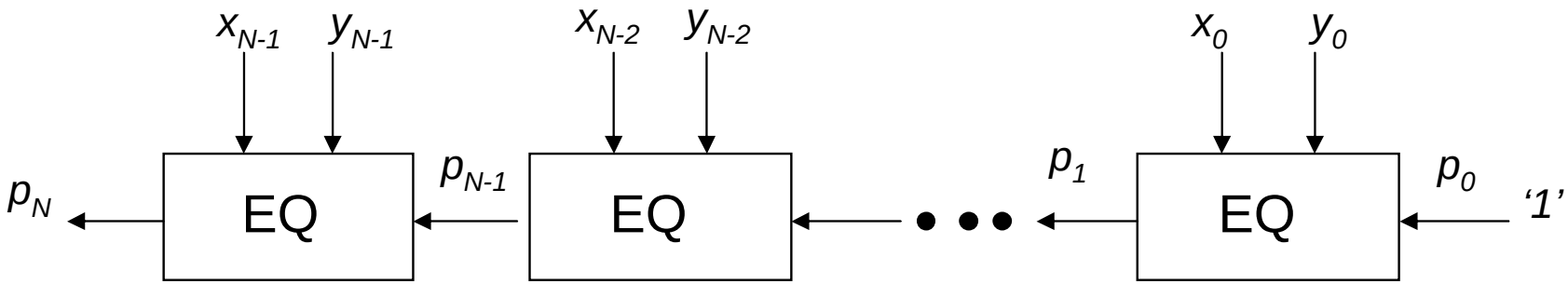


Fig. 4.11

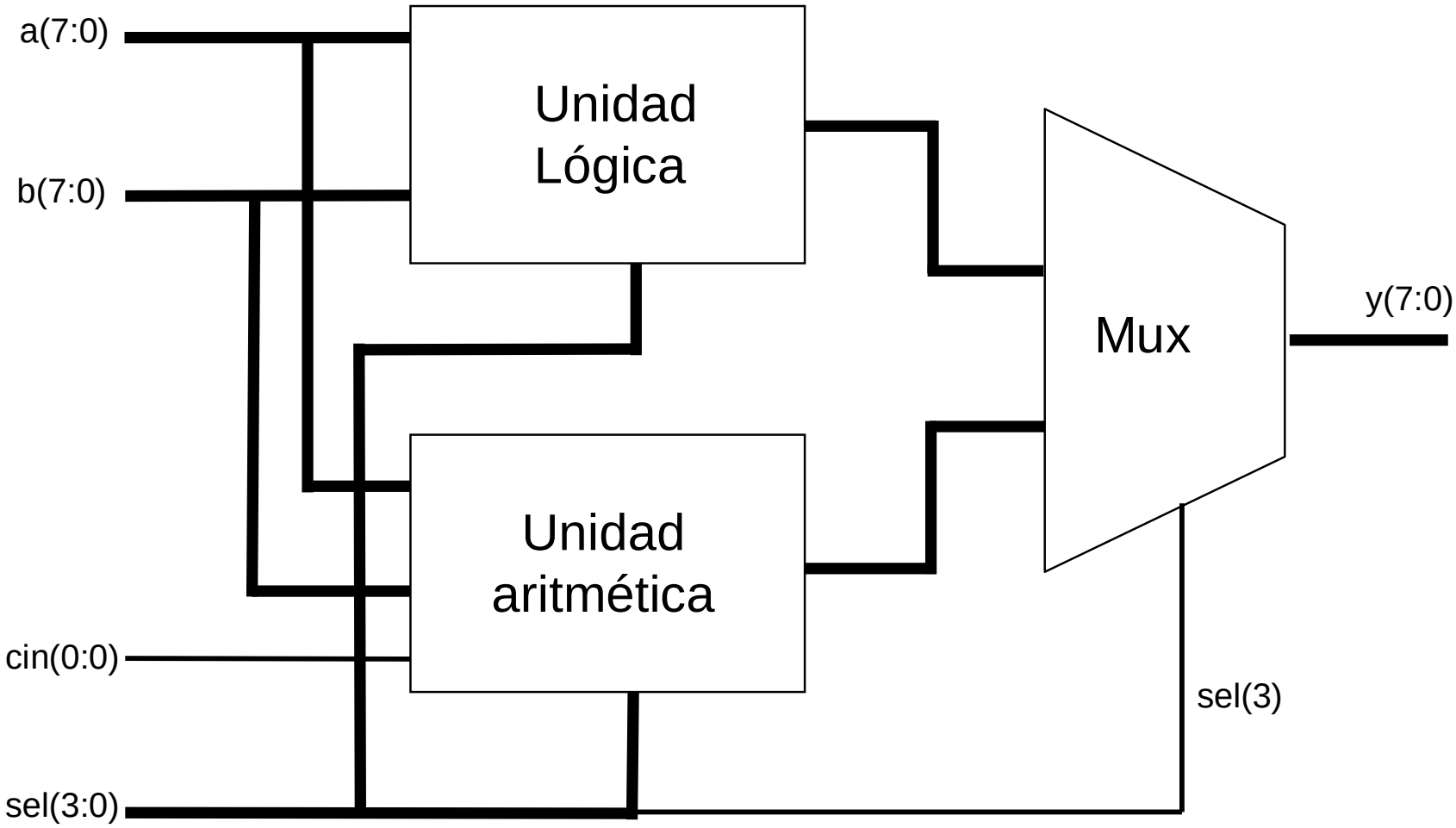
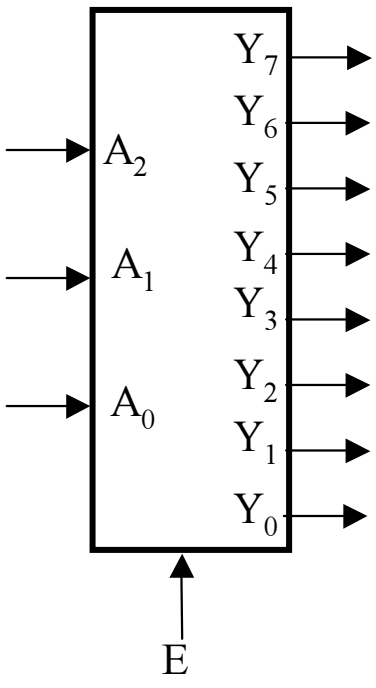

a(7:0)
b(7:0)
Unidad
Lógica
Mux
y(7:0)
Unidad
aritmética
cin(0:0)
sel(3)
sel(3:0)

Fig. 4.12



a)

b)

E	A_2	A_1	A_0	Y_7	Y_6	Y_5	Y_4	Y_3	Y_2	Y_1	Y_0
0	X	X	X	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	1
1	0	0	1	0	0	0	0	0	0	1	0
1	0	1	0	0	0	0	0	0	1	0	0
1	0	1	1	0	0	0	0	1	0	0	0
1	1	0	0	0	0	0	1	0	0	0	0
1	1	0	1	0	0	1	0	0	0	0	0
1	1	1	0	0	1	0	0	0	0	0	0
1	1	1	1	1	0	0	0	0	0	0	0

Fig. 4.13

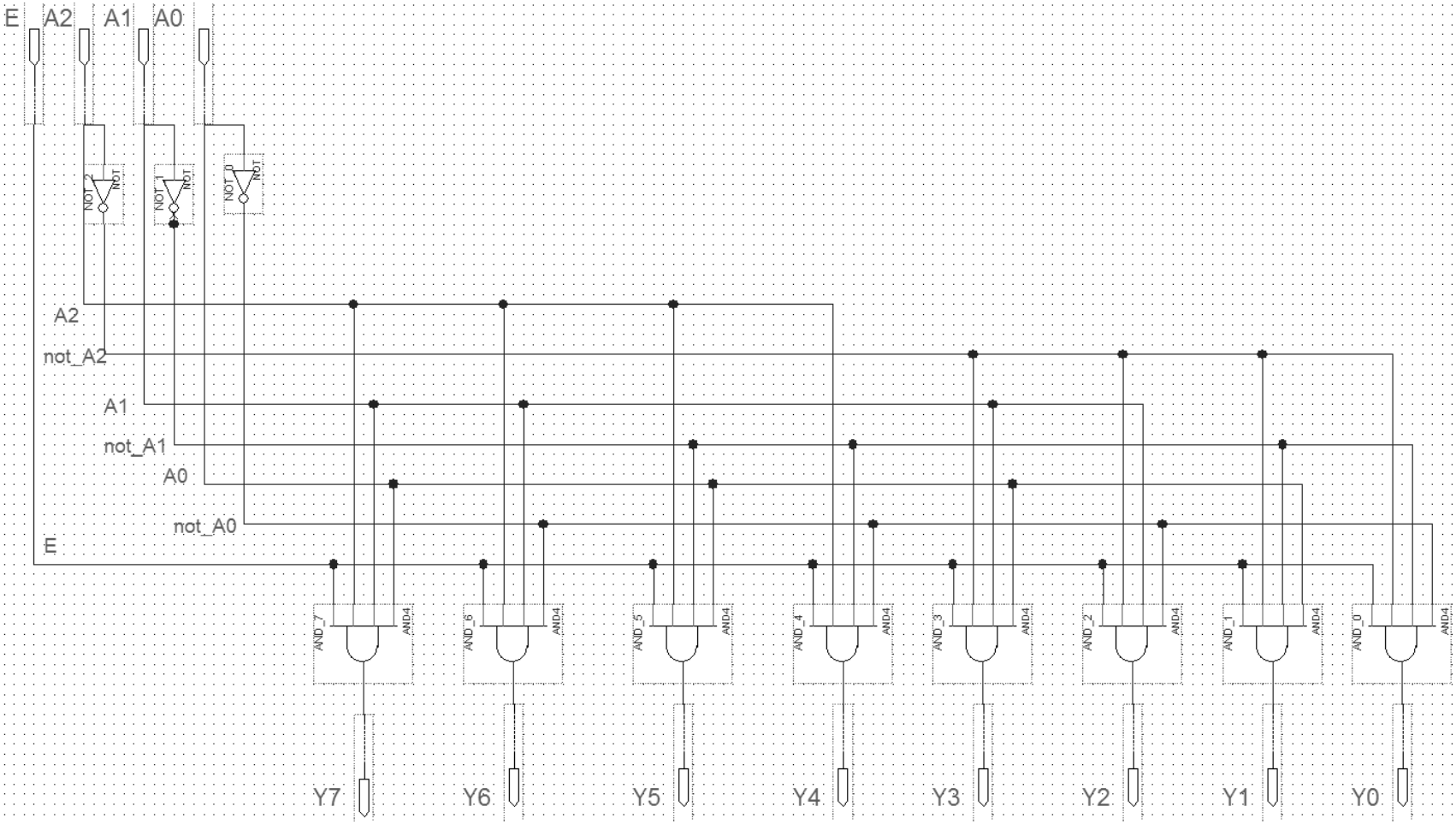

E
A2
A1
A0
NOT_2
NOT_1
NOT_0
NOT
A2
not_A2
A1
not_A1
A0
not_A0
E
AND_7
AND_6
AND_5
AND_4
AND_3
AND_2
AND_1
AND_0
AND4
Y7
Y6
Y5
Y4
Y3
Y2
Y1
Y0

Fig. 4.14

mode	Operación	Descripción
0 0 0	$A * 2$	Multiplicar por 2 (equivale a desplazamiento a la izqda)
0 0 1	$A + B$	Suma
0 1 0	$A - B$	Resta
0 1 1	$-A$	Complemento a dos
1 0 0	A and B	AND lógico
1 0 1	A or B	OR lógico
1 1 0	A xor B	XOR lógico
1 1 1	not A	Complemento de todos los bits

Tabla 4.1

i_3	i_2	i_1	i_0	a	b	c	d	e	f	g
0	0	0	0	1	1	1	1	1	1	0
0	0	0	1	0	1	1	0	0	0	0
0	0	1	0	1	1	0	1	1	0	1
0	0	1	1	1	1	1	1	0	0	1
0	1	0	0	0	1	1	0	0	1	1
0	1	0	1	1	0	1	1	0	1	1
0	1	1	0	1	0	1	1	1	1	1
0	1	1	1	1	1	1	0	0	0	0
1	0	0	0	1	1	1	1	1	1	1
1	0	0	1	1	1	1	0	0	1	1

Tabla 4.2

op	Operación
0 0	Desplaza a la izquierda rellenando con '0'
0 1	Desplaza a la derecha rellenando con '0'
1 0	Rota a la izquierda
1 1	Rota a la derecha

Tabla 4.3

Unidad	`sel`	Operación	Descripción
Unidad Aritmética	0 0 0 0	a	Deja pasar a
	0 0 0 1	$a + 1$	Incrementa a
	0 0 1 0	a - 1	Decrementa a
	0 0 1 1	b	Deja pasar b
	0 1 0 0	b+1	Incrementa b
	0 1 0 1	b-1	Decrementa b
	0 1 1 0	a+b	Suma a y b
	0 1 1 1	a+b+cin	Suma a, b y cin
Unidad Lógica	1 0 0 0	not a	Complemento de a
	1 0 0 1	not b	Complemento de b
	1 0 1 0	a and b	AND lógico
	1 0 1 1	a or b	OR lógico
	1 1 0 0	a nand b	NAND lógico
	1 1 0 1	a nor b	NOR lógico
	1 1 1 0	a xor b	XOR lógico
	1 1 1 1	a xnor b	XNOR lógico

Tabla 4.4

TEMA 5

Registros y memorias

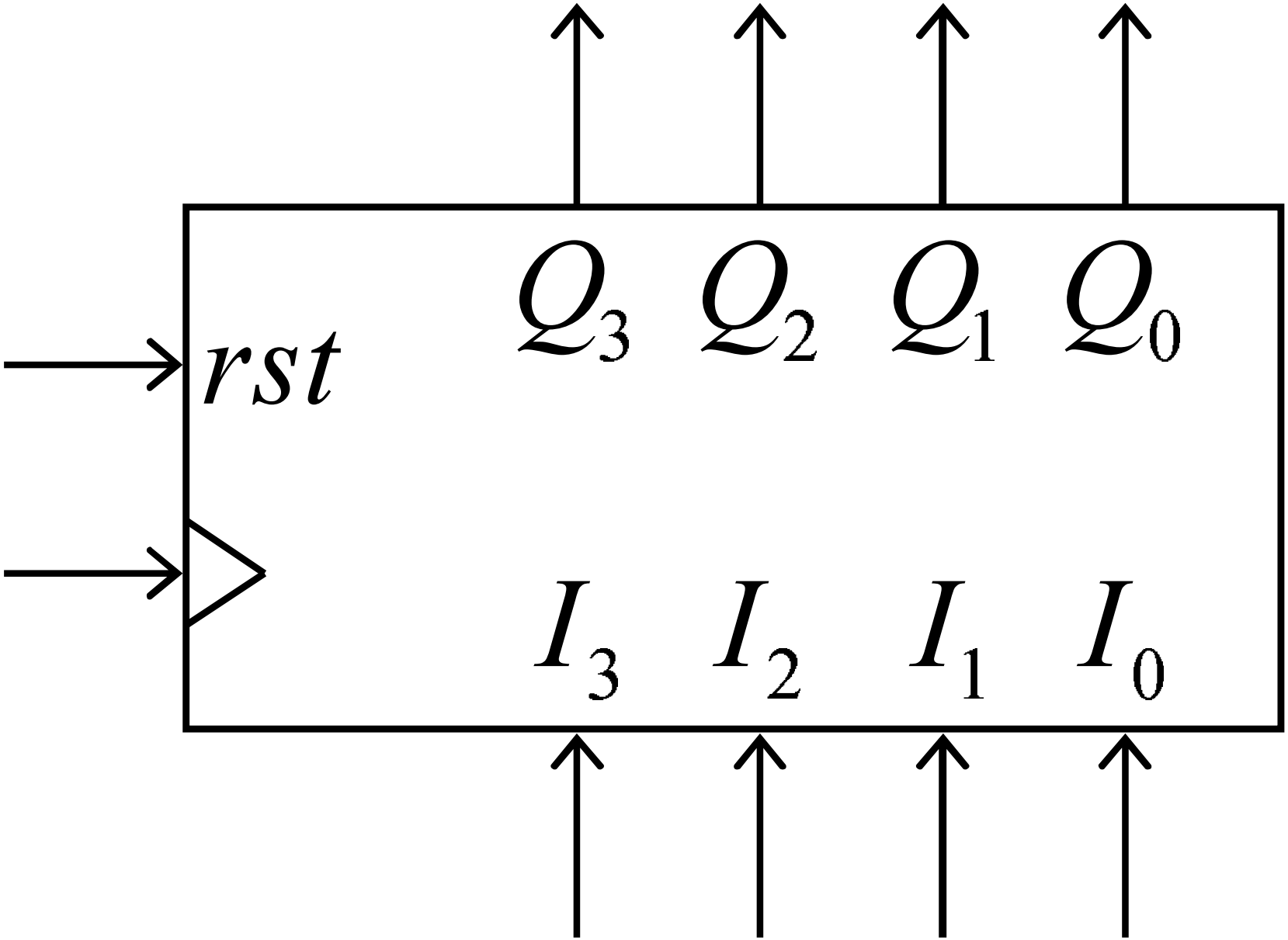


Fig. 5.1

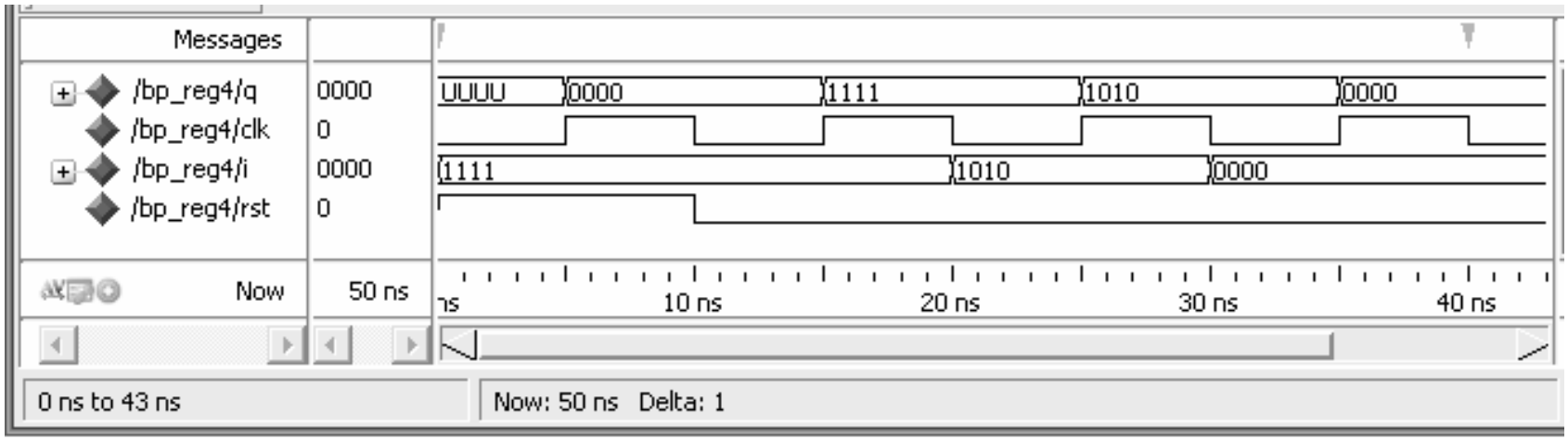
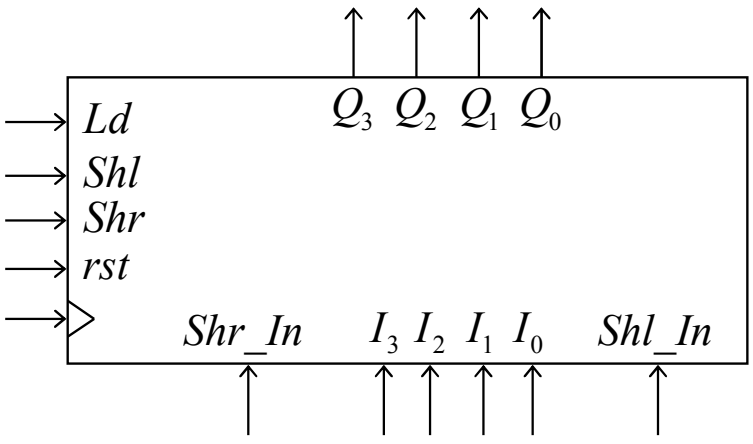

Messages
/bp_reg4/q
/bp_reg4/clk
/bp_reg4/i
/bp_reg4/rst
0000
0
0000
0
UUUU
0000
1111
1010
0000
1111
1010
0000
Now
50 ns
ns
10 ns
20 ns
30 ns
40 ns
0 ns to 43 ns
Now: 50 ns Delta: 1

Fig. 5.2



rst	*Ld*	*Shr*	*Shl*	Operación
0	0	0	0	Mantiene valor
0	0	0	1	Desplaz. Izq.
0	0	1	–	Desplaz. Drcha.
0	1	–	–	Carga paralelo
1	–	–	–	Reset (carga “0000”)

Fig. 5.3

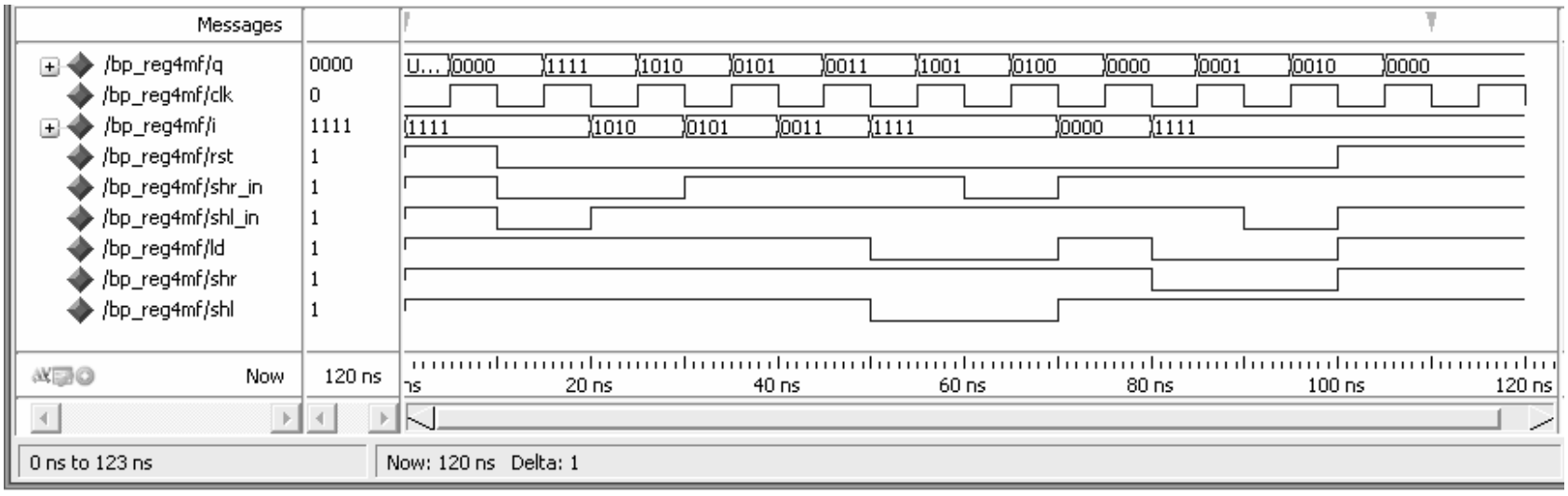
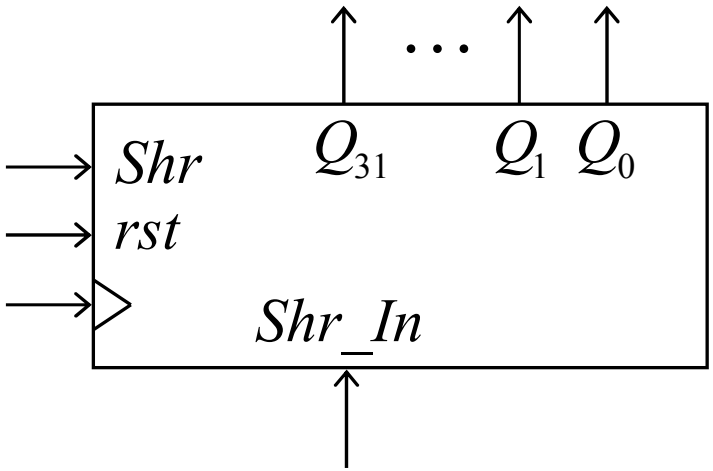

Messages
/bp_reg4mf/q
/bp_reg4mf/clk
/bp_reg4mf/i
/bp_reg4mf/rst
/bp_reg4mf/shr_in
/bp_reg4mf/shl_in
/bp_reg4mf/ld
/bp_reg4mf/shr
/bp_reg4mf/shl
0000
0
1111
1
1
1
1
1
1
U...
0000
1111
1010
0101
0011
1001
0100
0000
0001
0010
0000
1111
1010
0101
0011
1111
0000
1111
Now
120 ns
ns
20 ns
40 ns
60 ns
80 ns
100 ns
120 ns
0 ns to 123 ns
Now: 120 ns Delta: 1

Fig. 5.4



rst	*Shr*	Operación
0	0	Mantiene valor
0	1	Desplaz. Drcha.
1	–	Reset (carga X"00000000")

Fig. 5.5

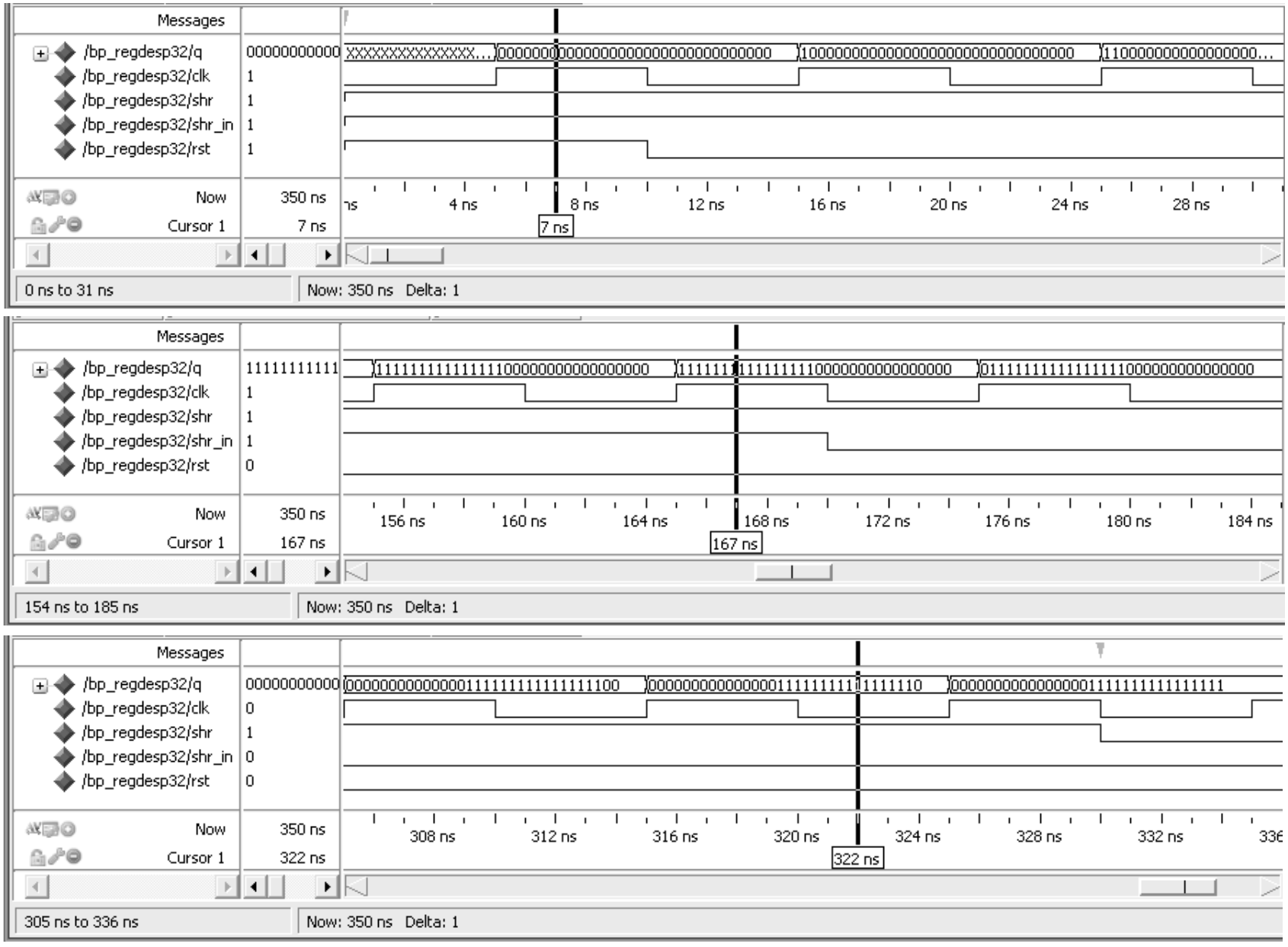

Messages
/bp_regdesp32/q
/bp_regdesp32/clk
/bp_regdesp32/shr
/bp_regdesp32/shr_in
/bp_regdesp32/rst
00000000000
1
1
1
1
Now
350 ns
Cursor 1
7 ns
4 ns
8 ns
12 ns
16 ns
20 ns
24 ns
28 ns
0 ns to 31 ns
Now: 350 ns Delta: 1
Messages
/bp_regdesp32/q
/bp_regdesp32/clk
/bp_regdesp32/shr
/bp_regdesp32/shr_in
/bp_regdesp32/rst
11111111111
1
1
1
0
Now
350 ns
Cursor 1
167 ns
156 ns
160 ns
164 ns
168 ns
172 ns
176 ns
180 ns
184 ns
154 ns to 185 ns
Now: 350 ns Delta: 1
Messages
/bp_regdesp32/q
/bp_regdesp32/clk
/bp_regdesp32/shr
/bp_regdesp32/shr_in
/bp_regdesp32/rst
00000000000
0
1
0
0
Now
350 ns
Cursor 1
322 ns
308 ns
312 ns
316 ns
320 ns
324 ns
328 ns
332 ns
305 ns to 336 ns
Now: 350 ns Delta: 1

Fig. 5.6

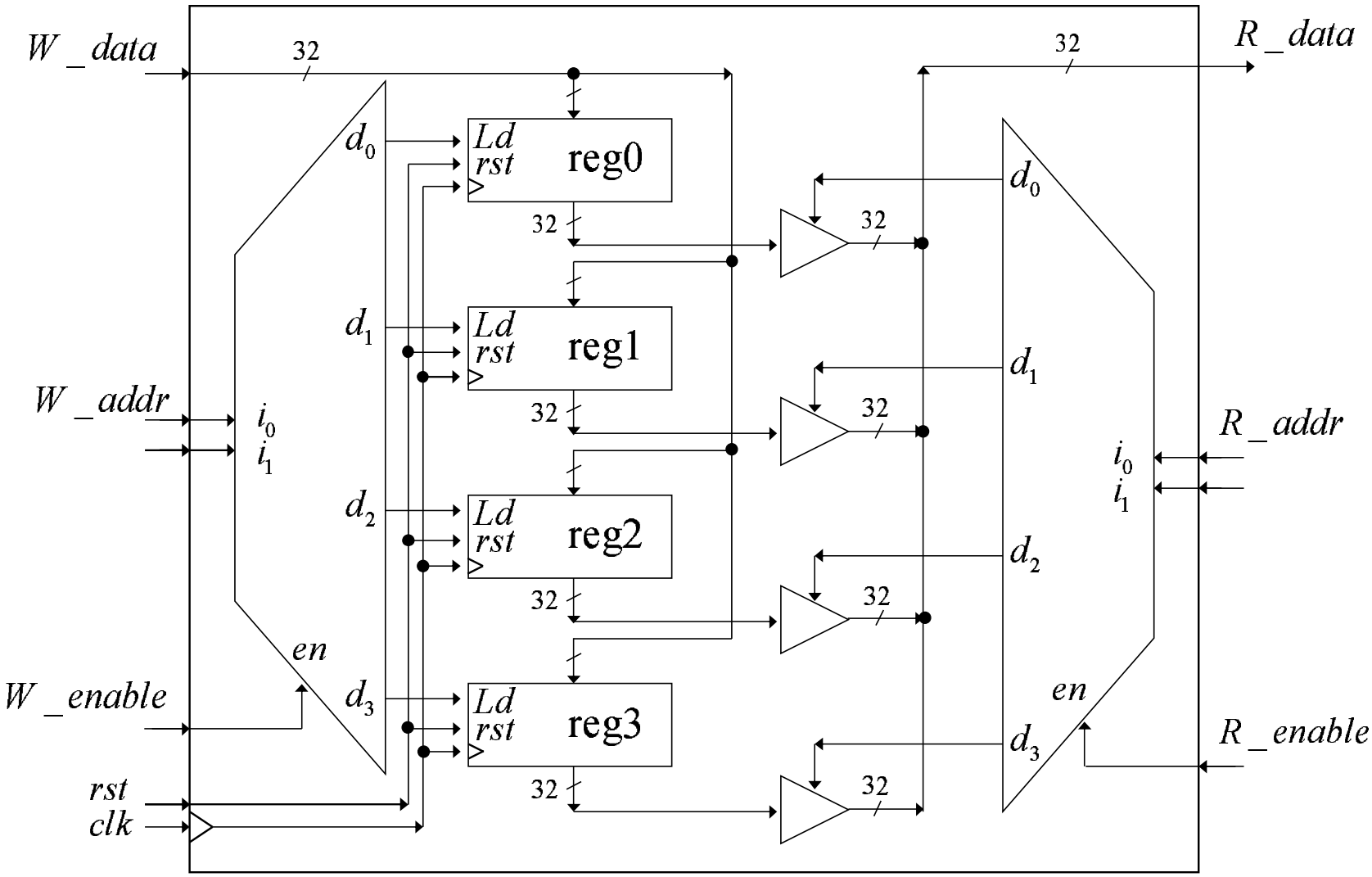

W_data
32
R_data
32
d0
Ld
rst
reg0
32
d1
Ld
rst
reg1
32
W_addr
i0
i1
d2
Ld
rst
reg2
32
en
W_enable
d3
Ld
rst
reg3
32
rst
clk
32
d0
32
d1
32
R_addr
i0
i1
d2
32
en
d3
R_enable
32

Fig. 5.7

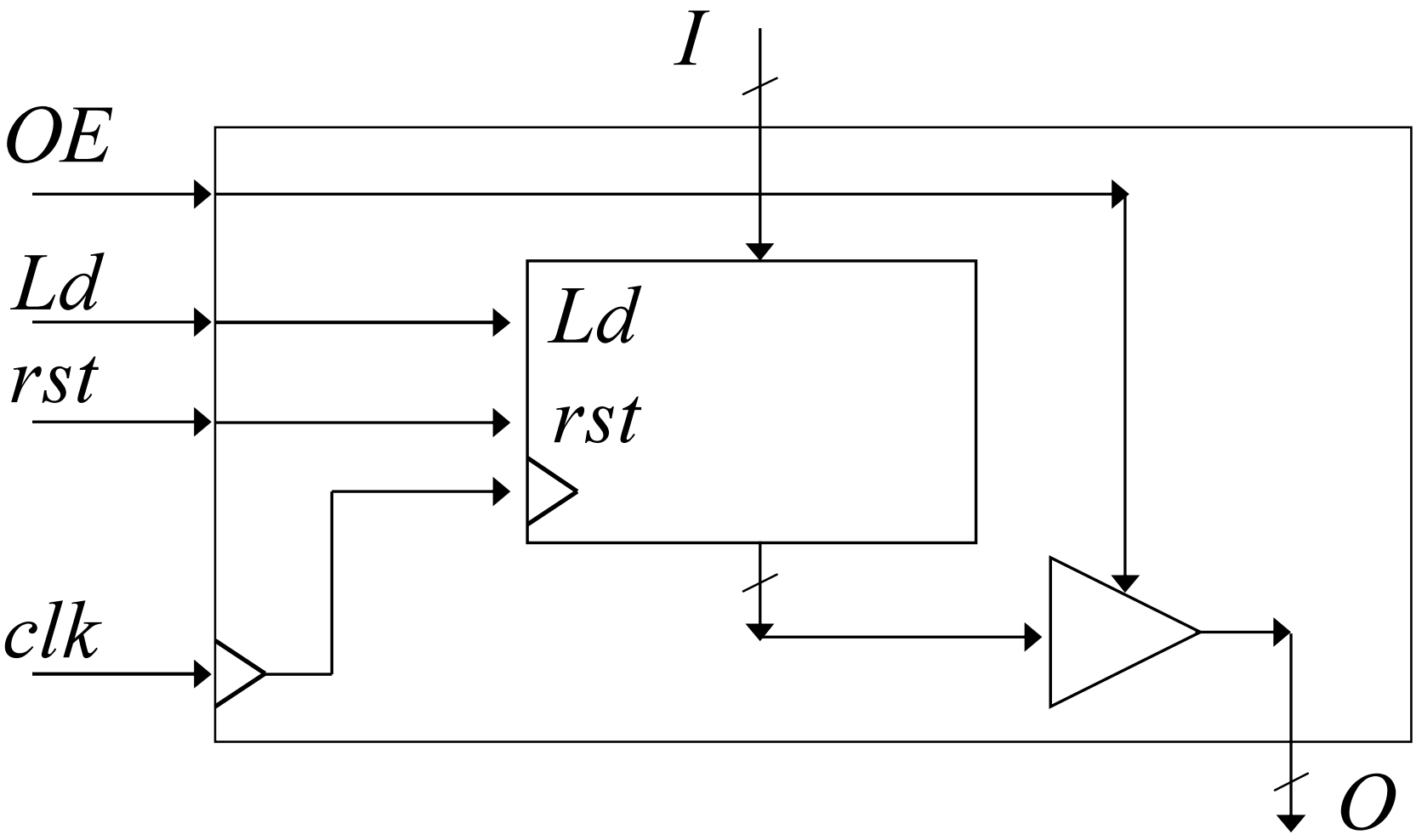


Fig. 5.8

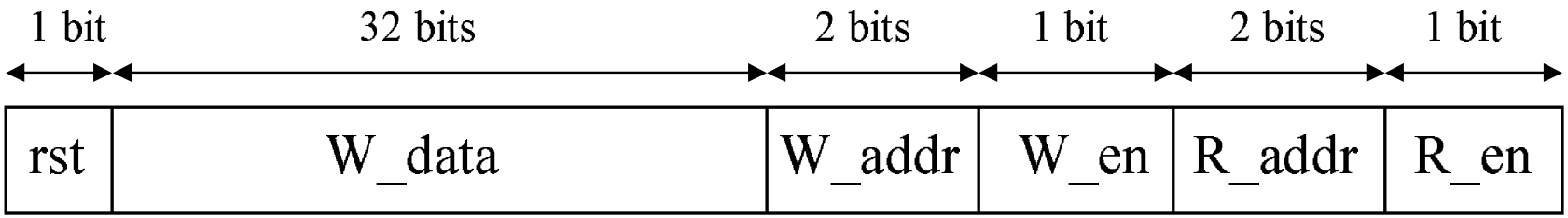

1 bit
32 bits
2 bits
1 bit
2 bits
1 bit
rst
W_data
W_addr
W_en
R_addr
R_en

Fig. 5.9

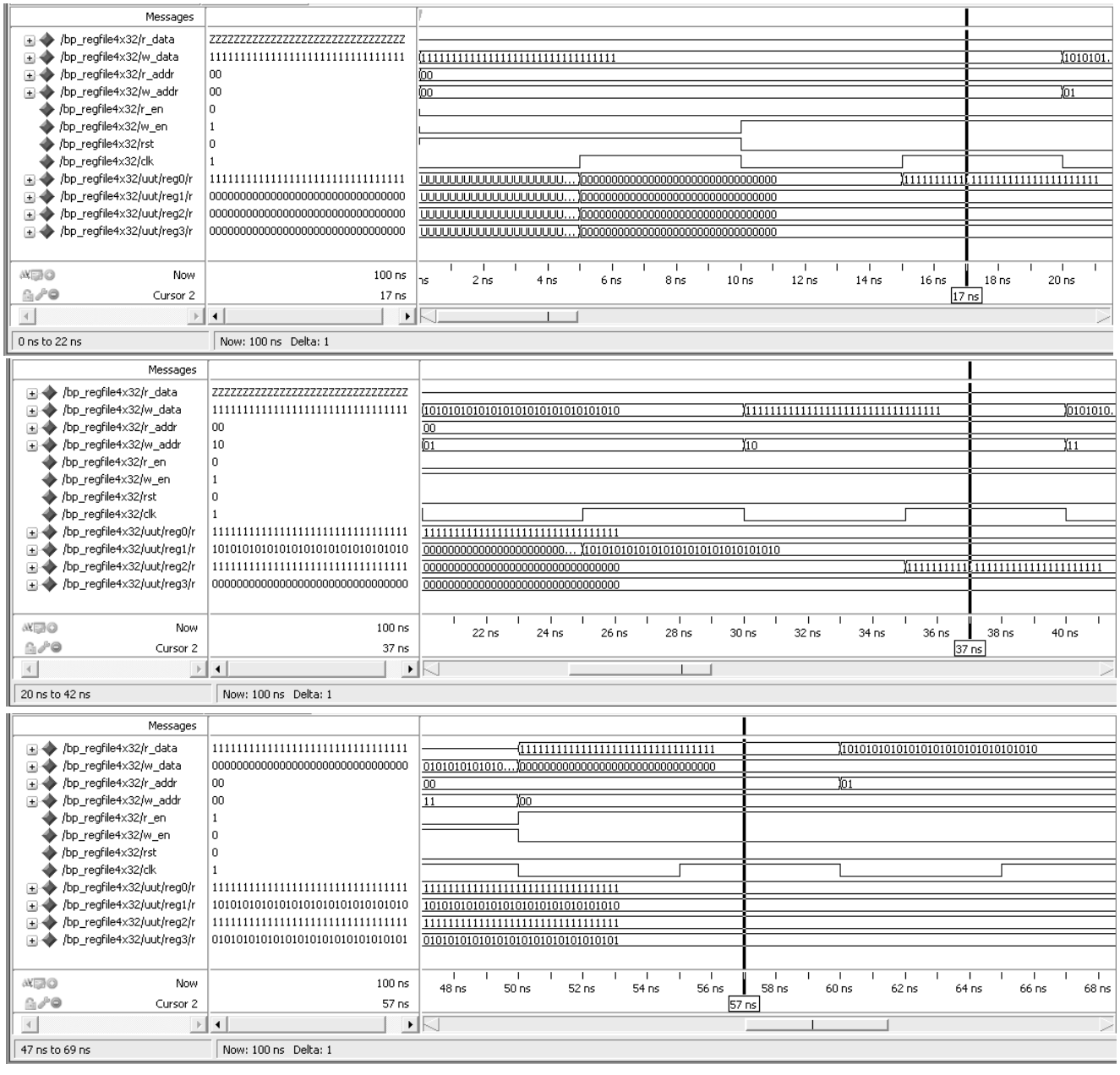

Messages
/bp_regfile4x32/r_data
/bp_regfile4x32/w_data
/bp_regfile4x32/r_addr
/bp_regfile4x32/w_addr
/bp_regfile4x32/r_en
/bp_regfile4x32/w_en
/bp_regfile4x32/rst
/bp_regfile4x32/clk
/bp_regfile4x32/uut/reg0/r
/bp_regfile4x32/uut/reg1/r
/bp_regfile4x32/uut/reg2/r
/bp_regfile4x32/uut/reg3/r
Now
Cursor 2
100 ns
17 ns
2 ns
4 ns
6 ns
8 ns
10 ns
12 ns
14 ns
16 ns
18 ns
20 ns
0 ns to 22 ns
Now: 100 ns Delta: 1
Messages
/bp_regfile4x32/r_data
/bp_regfile4x32/w_data
/bp_regfile4x32/r_addr
/bp_regfile4x32/w_addr
/bp_regfile4x32/r_en
/bp_regfile4x32/w_en
/bp_regfile4x32/rst
/bp_regfile4x32/clk
/bp_regfile4x32/uut/reg0/r
/bp_regfile4x32/uut/reg1/r
/bp_regfile4x32/uut/reg2/r
/bp_regfile4x32/uut/reg3/r
Now
Cursor 2
100 ns
37 ns
22 ns
24 ns
26 ns
28 ns
30 ns
32 ns
34 ns
36 ns
38 ns
40 ns
20 ns to 42 ns
Now: 100 ns Delta: 1
Messages
/bp_regfile4x32/r_data
/bp_regfile4x32/w_data
/bp_regfile4x32/r_addr
/bp_regfile4x32/w_addr
/bp_regfile4x32/r_en
/bp_regfile4x32/w_en
/bp_regfile4x32/rst
/bp_regfile4x32/clk
/bp_regfile4x32/uut/reg0/r
/bp_regfile4x32/uut/reg1/r
/bp_regfile4x32/uut/reg2/r
/bp_regfile4x32/uut/reg3/r
Now
Cursor 2
100 ns
57 ns
48 ns
50 ns
52 ns
54 ns
56 ns
58 ns
60 ns
62 ns
64 ns
66 ns
68 ns
47 ns to 69 ns
Now: 100 ns Delta: 1

Fig. 5.10

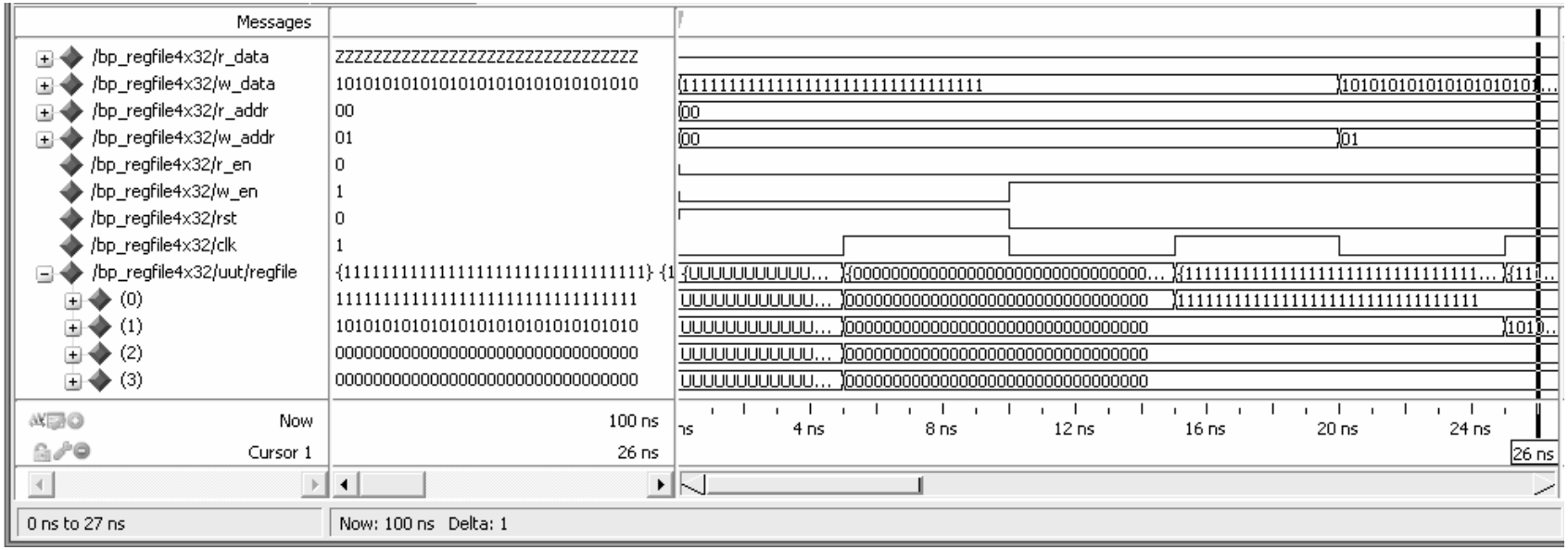

Messages
/bp_regfile4x32/r_data
/bp_regfile4x32/w_data
/bp_regfile4x32/r_addr
/bp_regfile4x32/w_addr
/bp_regfile4x32/r_en
/bp_regfile4x32/w_en
/bp_regfile4x32/rst
/bp_regfile4x32/clk
/bp_regfile4x32/uut/regfile
(0)
(1)
(2)
(3)
zzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzz
10101010101010101010101010101010
00
01
0
1
1
0
1
{11111111111111111111111111111111} {1
11111111111111111111111111111111
10101010101010101010101010101010
00000000000000000000000000000000
00000000000000000000000000000000
Now
100 ns
Cursor 1
26 ns
4 ns
8 ns
12 ns
16 ns
20 ns
24 ns
26 ns
0 ns to 27 ns
Now: 100 ns Delta: 1

Fig. 5.11

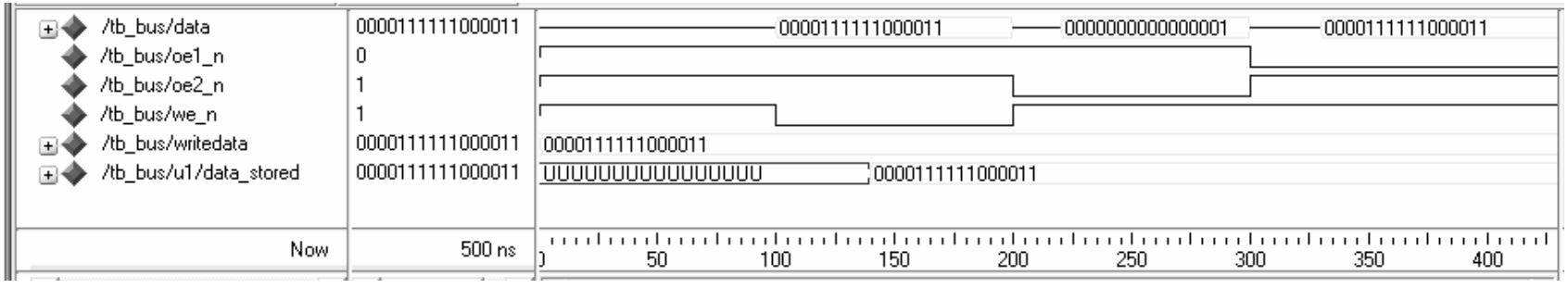

/tb_bus/data
/tb_bus/oe1_n
/tb_bus/oe2_n
/tb_bus/we_n
/tb_bus/writedata
/tb_bus/u1/data_stored
0000111111000011
0
1
1
0000111111000011
0000111111000011
0000111111000011
0000111111000011
0000111111000011
UUUUUUUUUUUUUUUU
Now
500 ns
0
50
100
150
200
250
300
350
400

Fig. 5.12

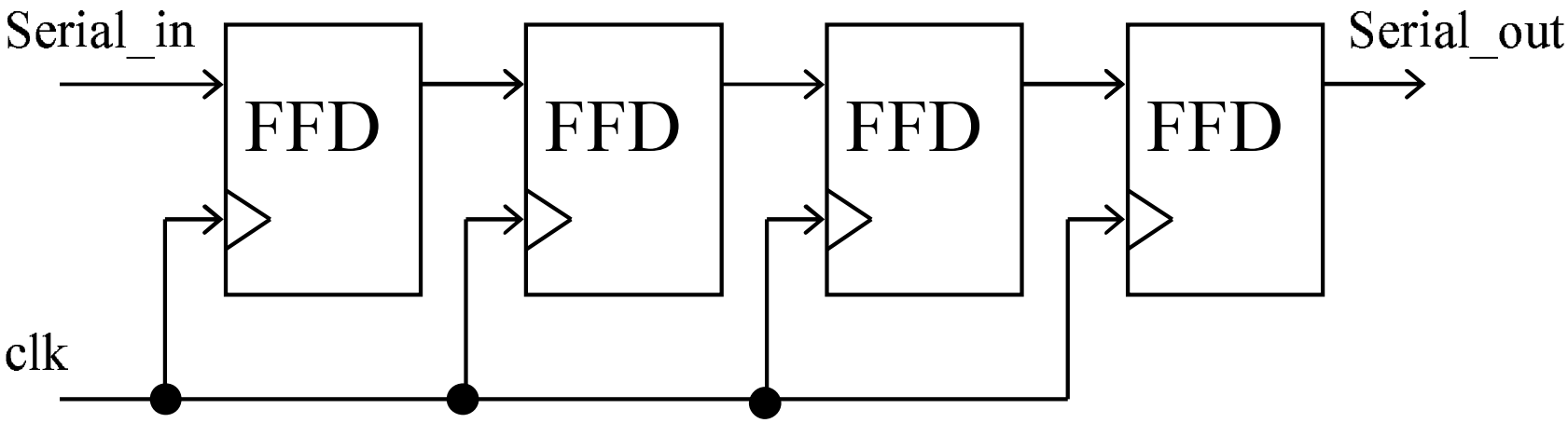


Fig. 5.13

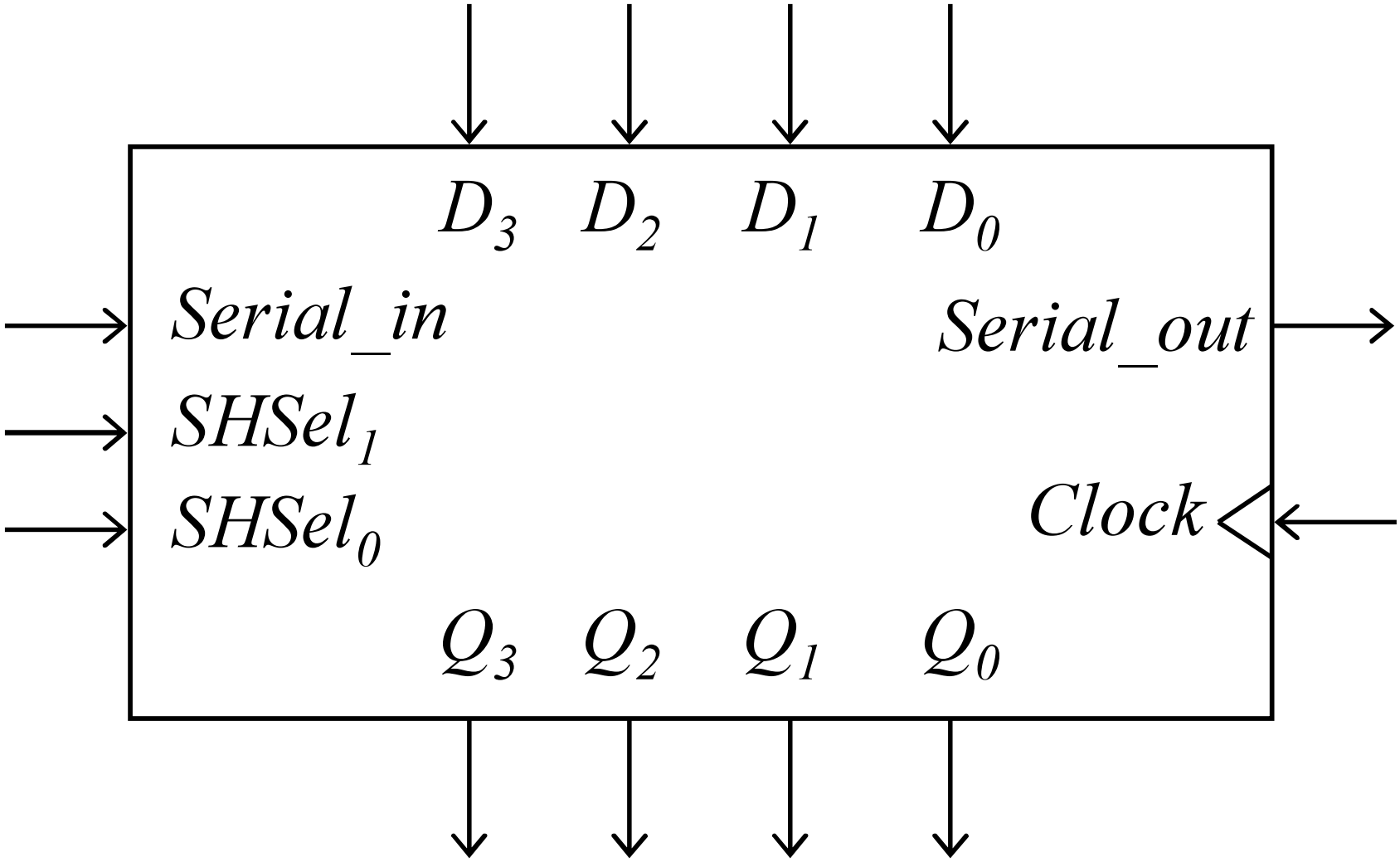

D_3 D_2 D_1 D_0
Serial_in
Serial_out
$SHSel_1$
$SHSel_0$
Clock
Q_3 Q_2 Q_1 Q_0

Fig. 5.14

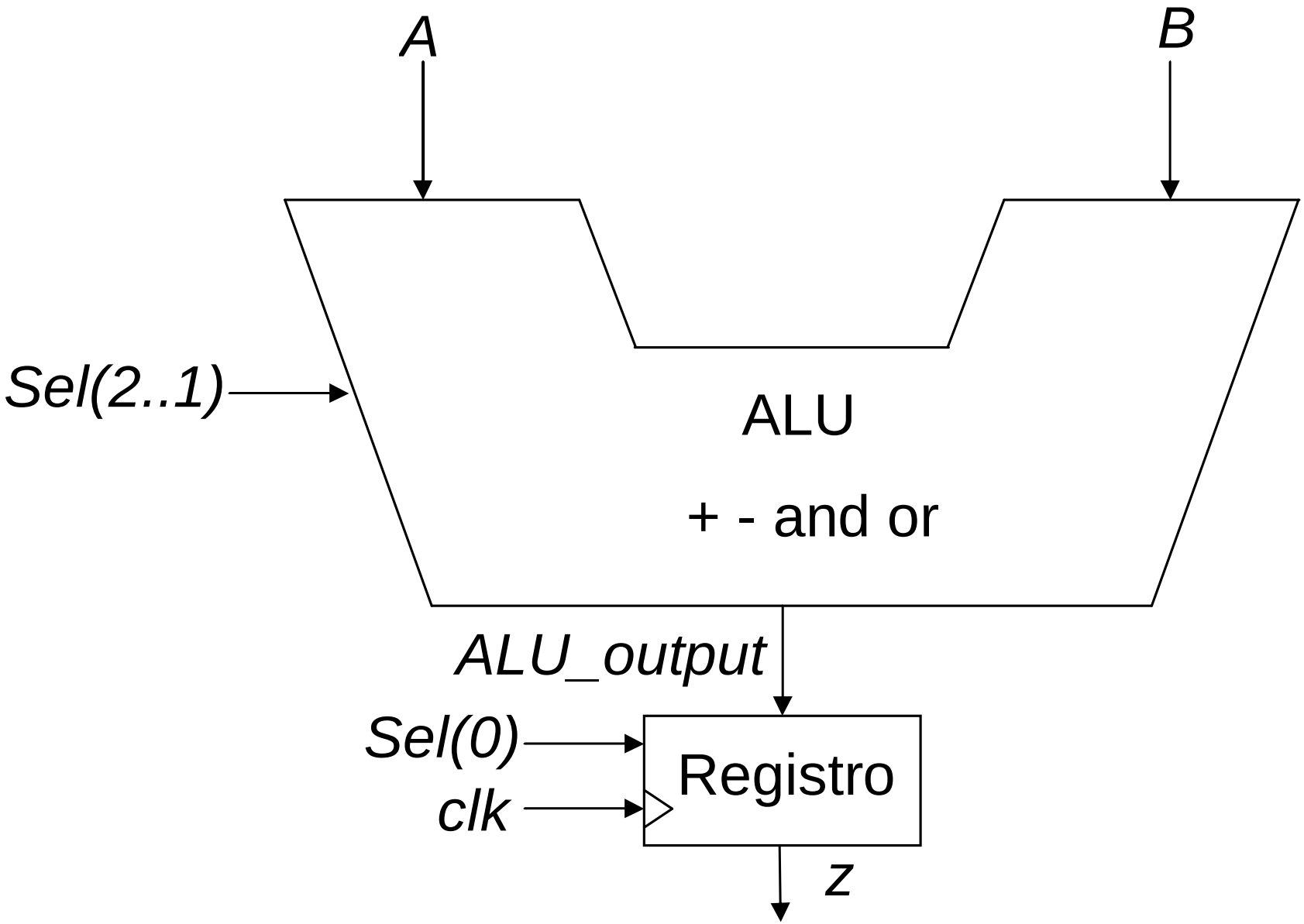


Fig. 5.15

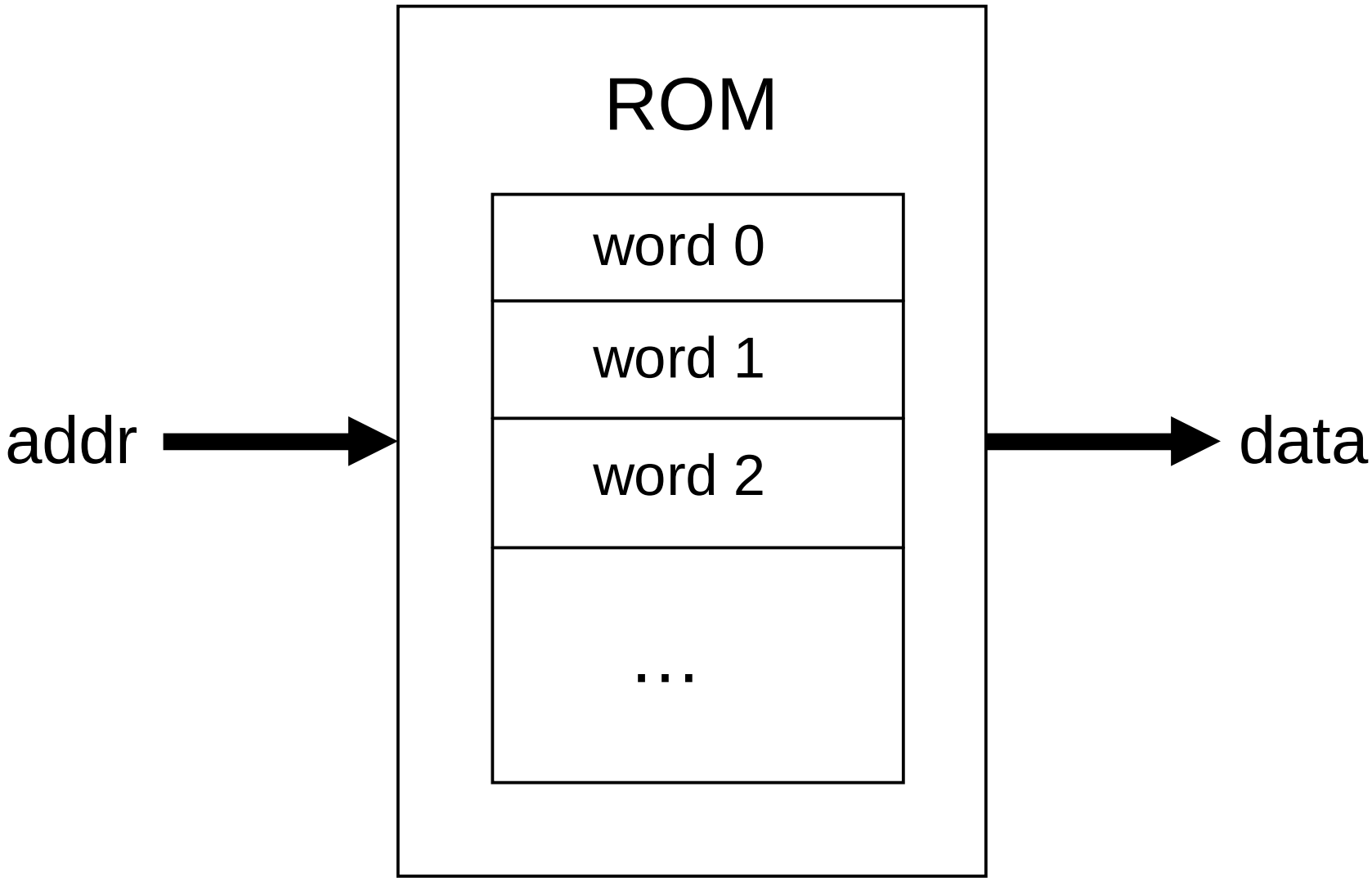


Fig. 5.16

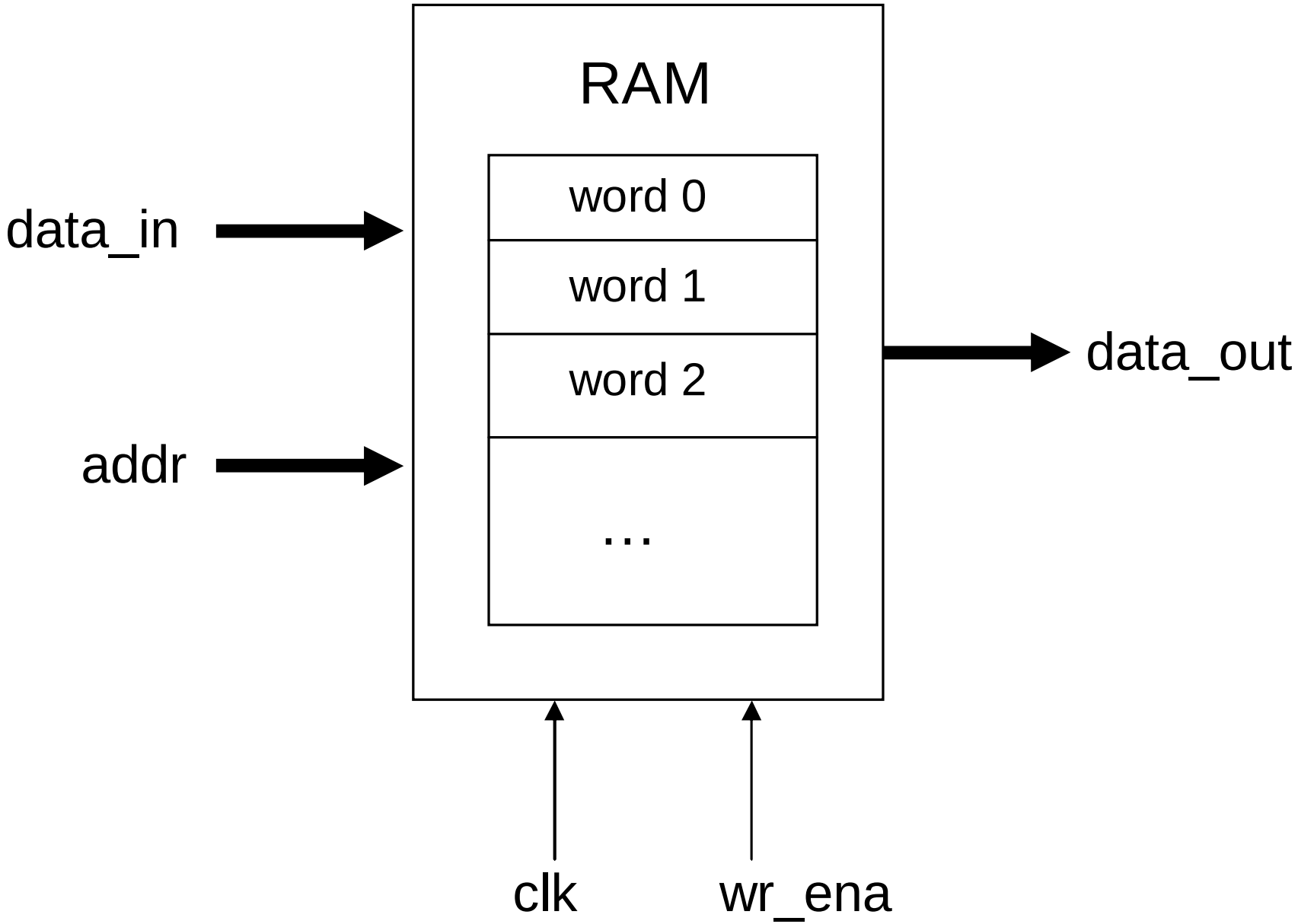


Fig. 5.17

rst	W_data	W_addr	W_en	R_addr	R_en	
1	11111111111111111111111111111111	00	0	00	0	Reset
0	11111111111111111111111111111111	00	1	00	0	Escribe reg0
0	10101010101010101010101010101010	01	1	00	0	Escribe reg1
0	11111111111111111111111111111111	10	1	00	0	Escribe reg2
0	01010101010101010101010101010101	11	1	00	0	Escribe reg3
0	00000000000000000000000000000000	00	0	00	1	Lee reg0
0	00000000000000000000000000000000	00	0	01	1	Lee reg1
0	00000000000000000000000000000000	00	0	10	1	Lee reg2
0	00000000000000000000000000000000	00	0	11	1	Lee reg3

Tabla 5.1

$SHSel_1$	$SHSel_2$	**Operación**
0	0	Mantiene valor
0	1	Carga en paralelo
1	0	Desplaza 1 bit a la derecha el contenido del registro y asigna a Q_3 el valor en $Serial\_in$

Tabla 5.2

Sel(0)	Operación
0	Desplazamiento a la derecha introduciendo un '0' a la izquierda
1	Carga el nuevo valor

Tabla 5.3

Sel(2)	Sel(1)	Operación
0	0	suma
0	1	resta
1	0	and
1	0	or

Tabla 5.4

TEMA 6

Diseño de lógica secuencial

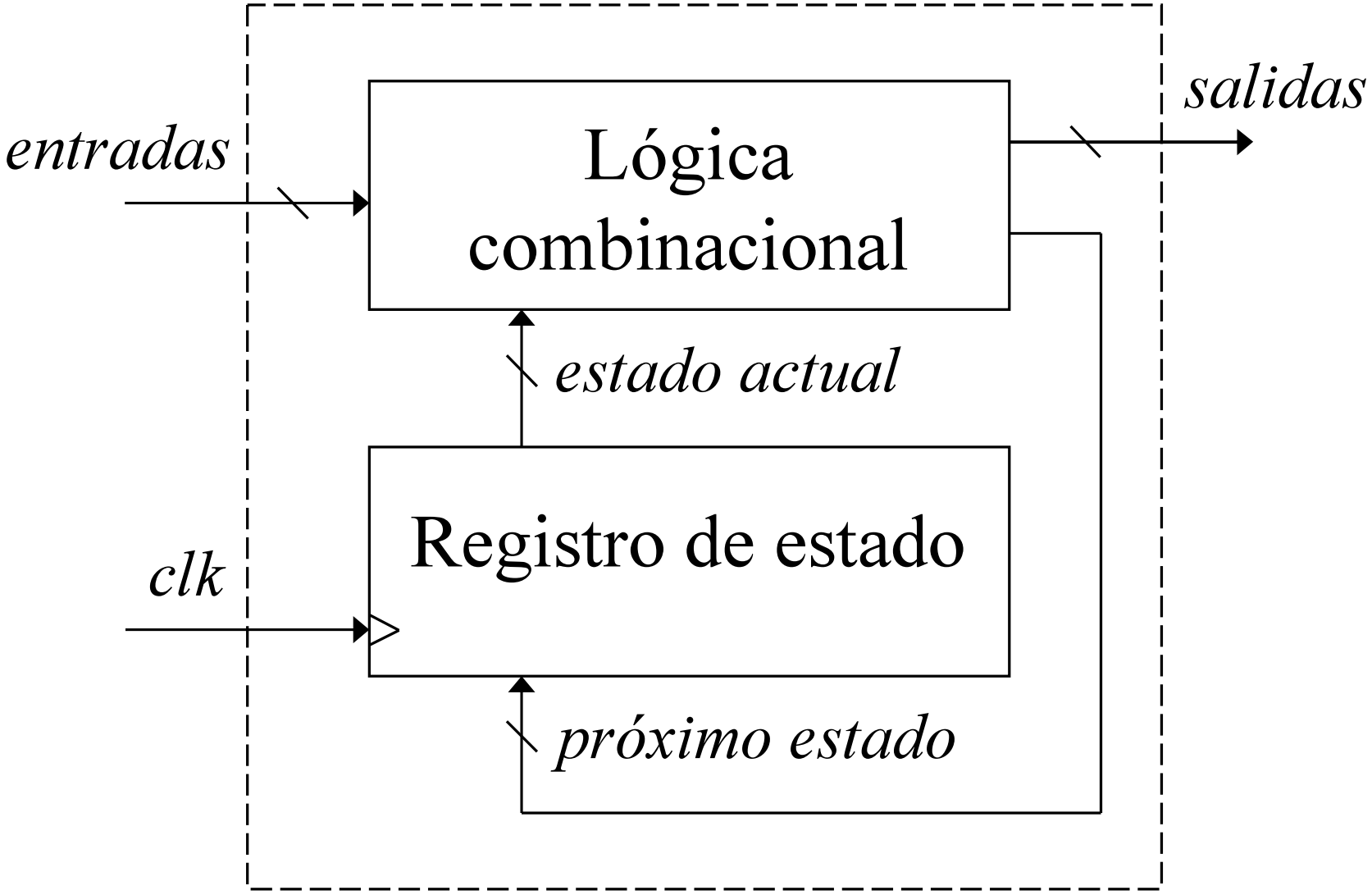

entradas
Lógica
combinacional
salidas
estado actual
Registro de estado
clk
próximo estado

Fig. 6.1

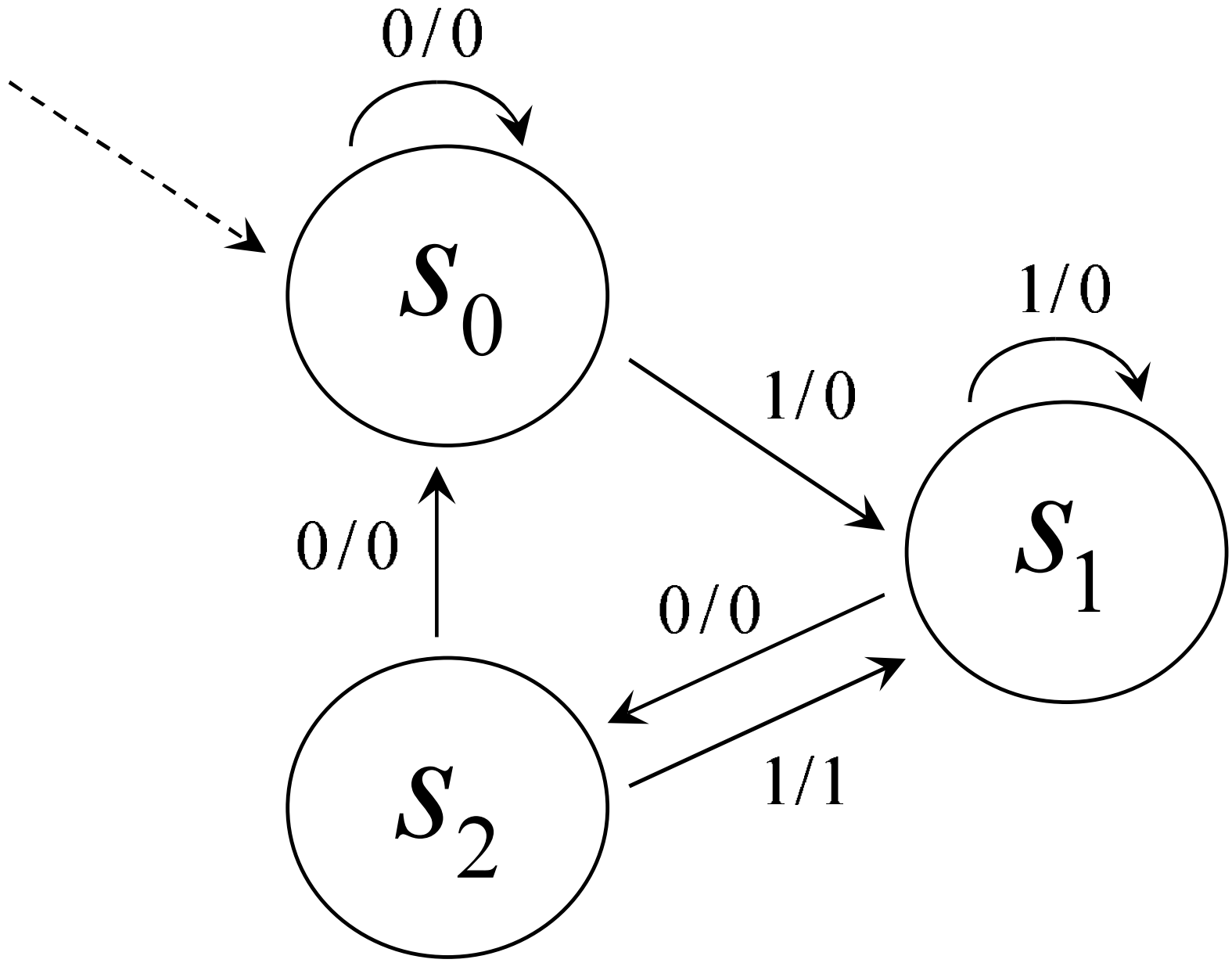


Fig. 6.2

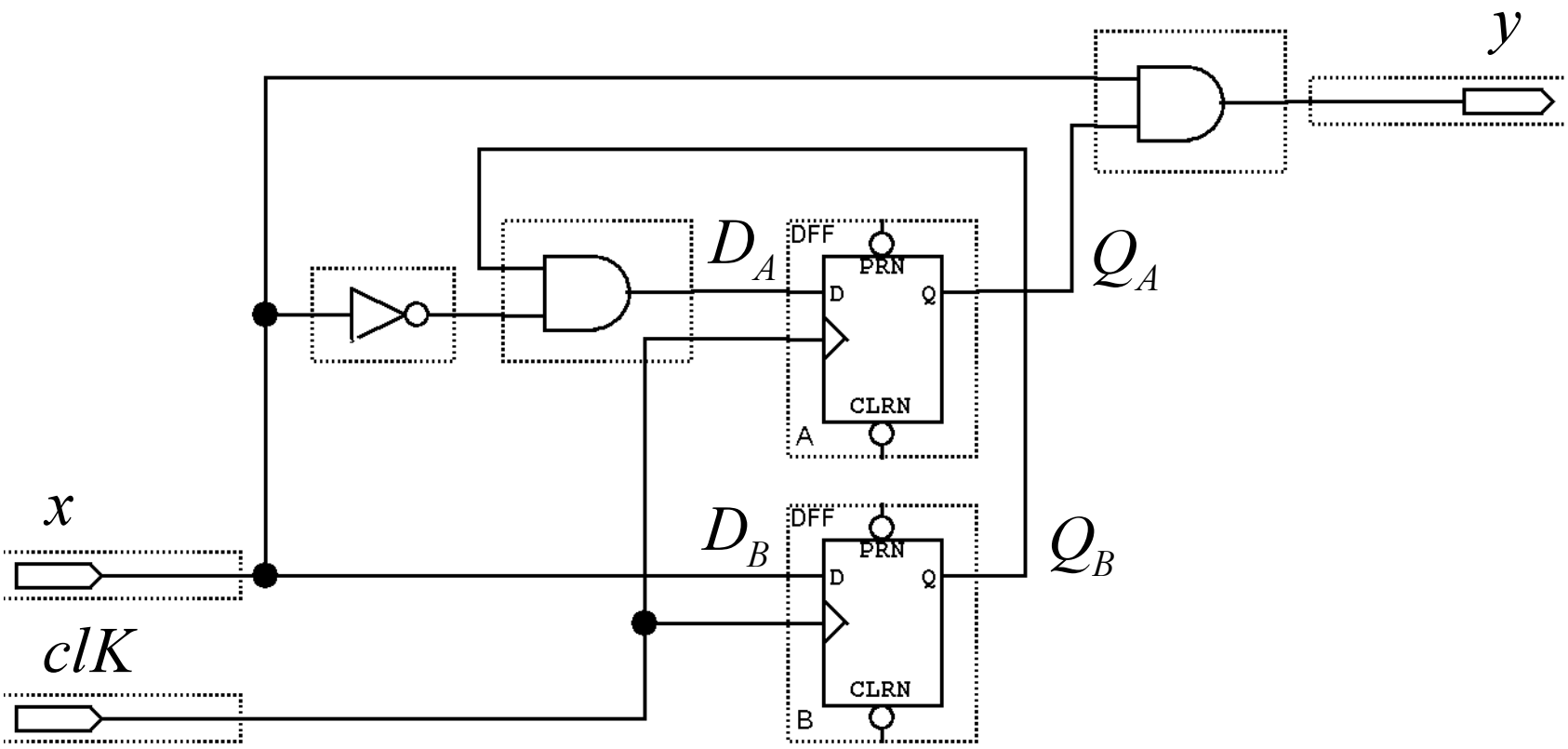


Fig. 6.3

J	***K***	**Nuevo estado**
0	0	$Q_{t+1}=Q_t$
0	1	$Q_{t+1}=0$
1	0	$Q_{t+1}=1$
1	1	$Q_{t+1}=\text{not } Q_t$

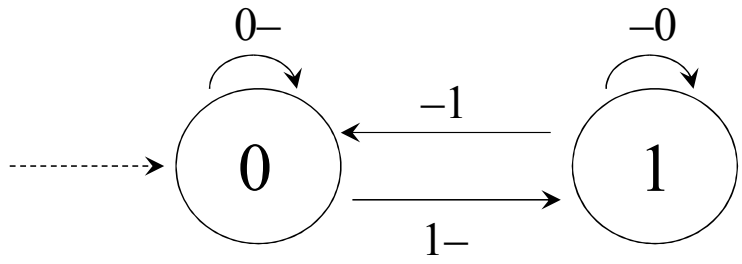


Fig. 6.4

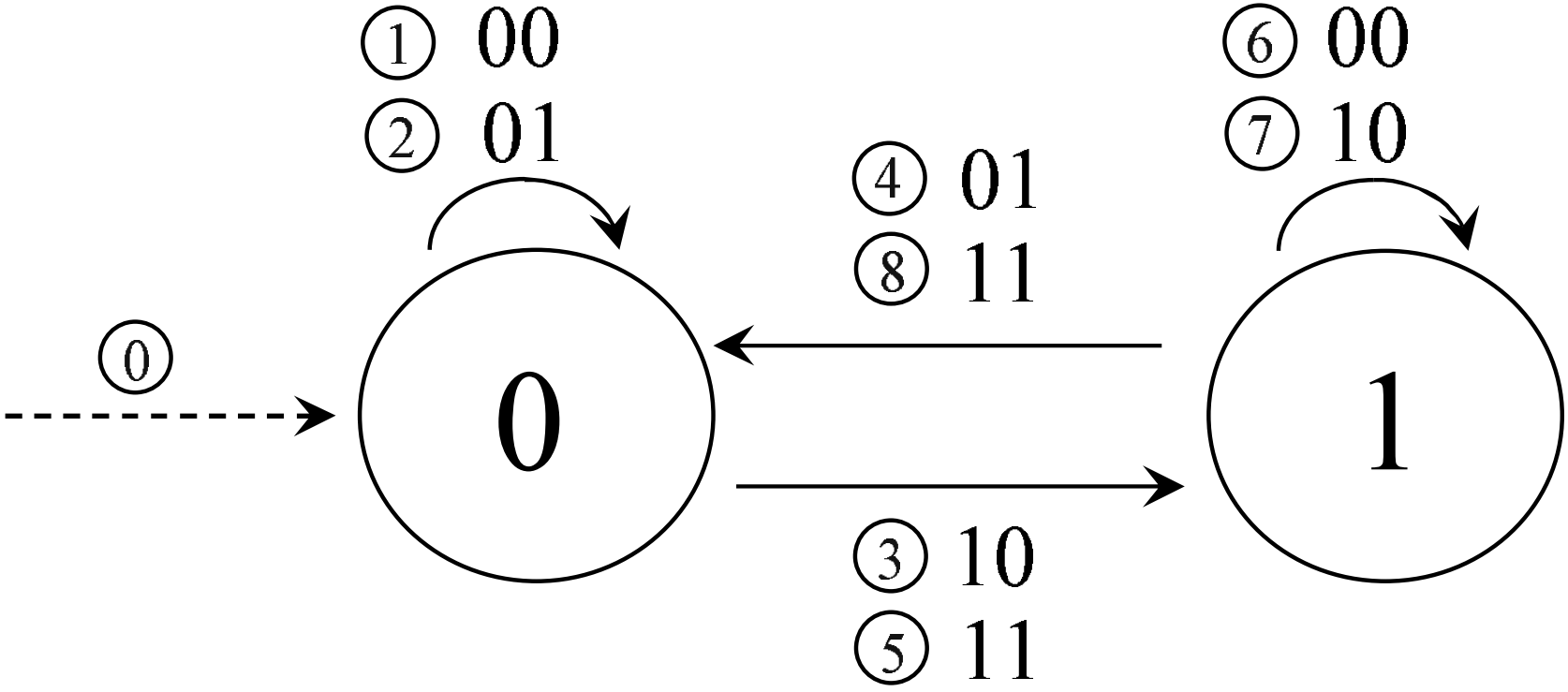


Fig. 6.5

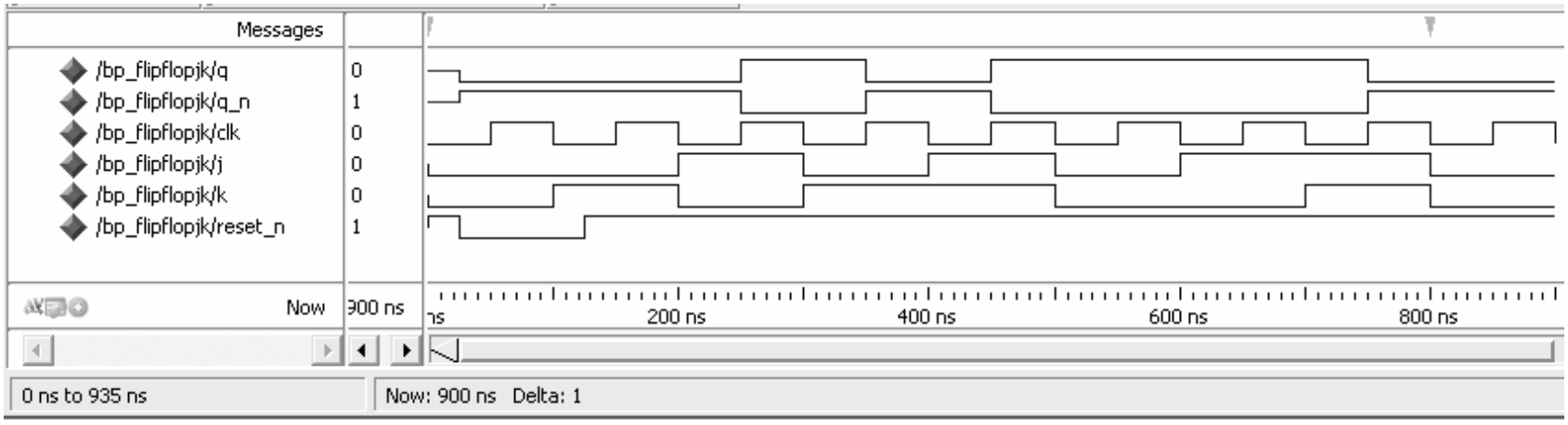

Messages
/bp_flipflopjk/q
/bp_flipflopjk/q_n
/bp_flipflopjk/clk
/bp_flipflopjk/j
/bp_flipflopjk/k
/bp_flipflopjk/reset_n
0
1
0
0
0
1
Now
900 ns
200 ns
400 ns
600 ns
800 ns
0 ns to 935 ns
Now: 900 ns Delta: 1

Fig. 6.6

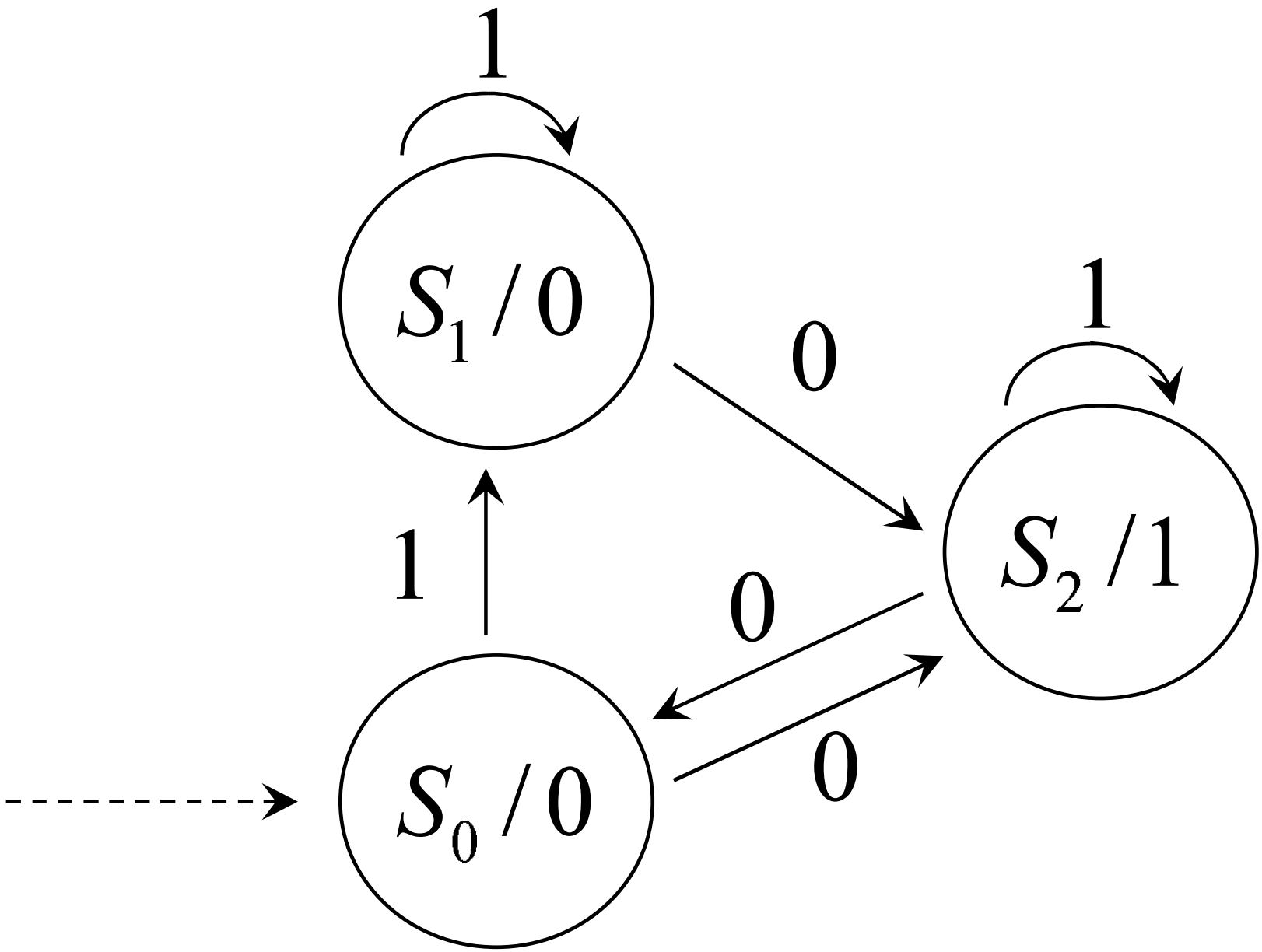


Fig. 6.7

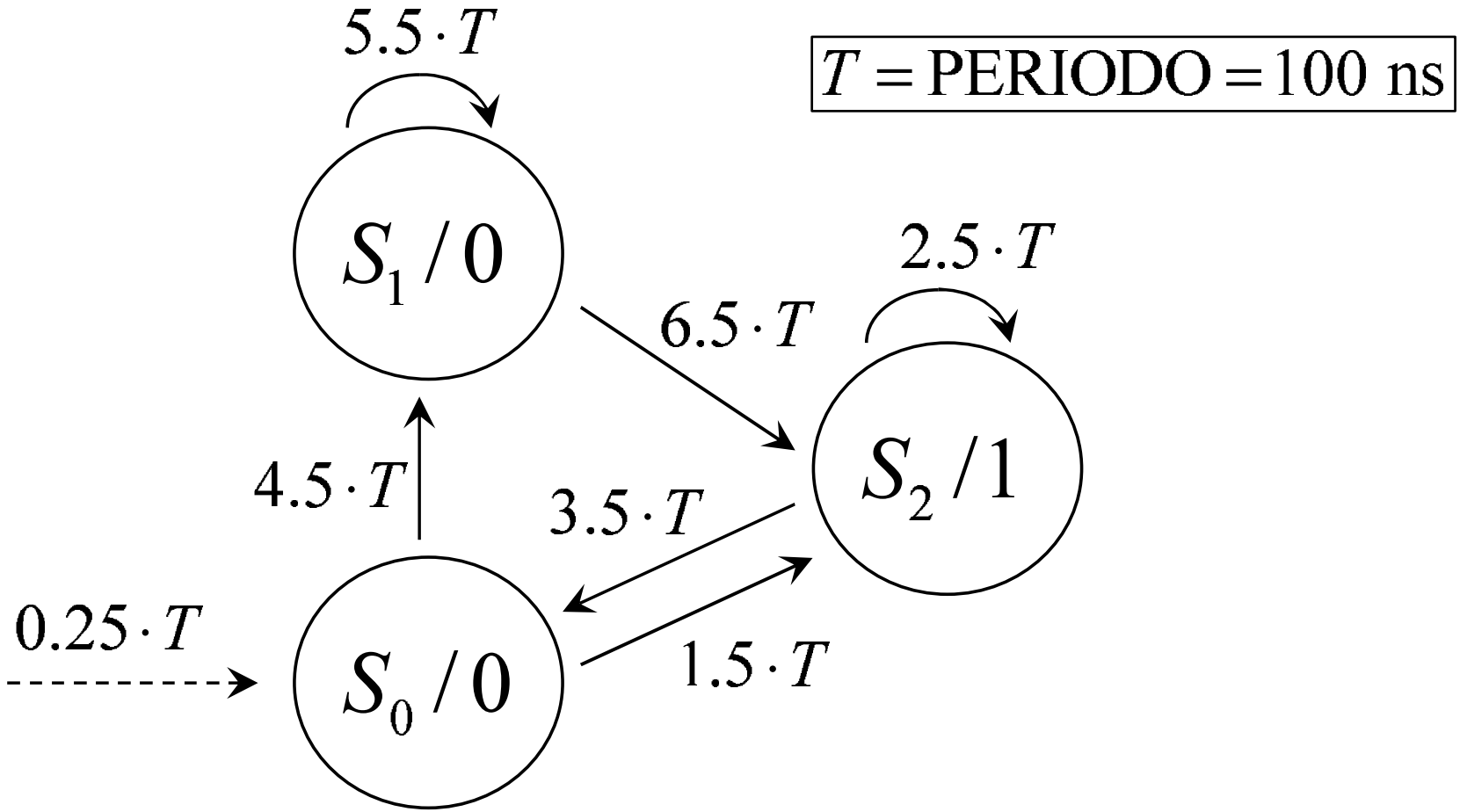

$T = \text{PERIODO} = 100$ ns
$5.5 \cdot T$
$S_1 / 0$
$2.5 \cdot T$
$6.5 \cdot T$
$S_2 / 1$
$4.5 \cdot T$
$3.5 \cdot T$
$0.25 \cdot T$
$S_0 / 0$
$1.5 \cdot T$

Fig. 6.8

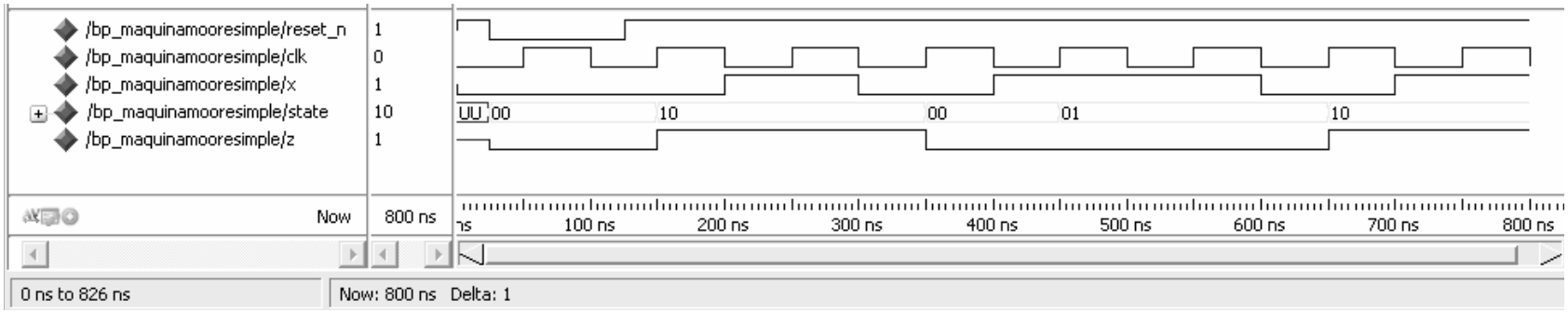

/bp_maquinamooresimple/reset_n
/bp_maquinamooresimple/clk
/bp_maquinamooresimple/x
/bp_maquinamooresimple/state
/bp_maquinamooresimple/z
1
0
1
10
1
UU
00
10
00
01
10
Now
800 ns
ns
100 ns
200 ns
300 ns
400 ns
500 ns
600 ns
700 ns
800 ns
0 ns to 826 ns
Now: 800 ns Delta: 1

Fig. 6.9

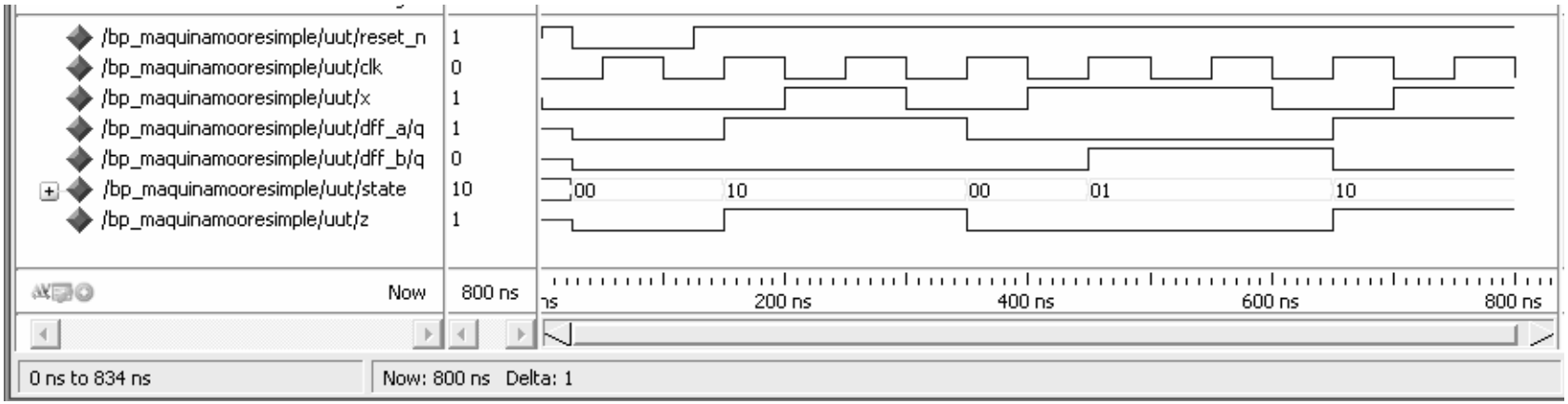

/bp_maquinamooresimple/uut/reset_n
/bp_maquinamooresimple/uut/clk
/bp_maquinamooresimple/uut/x
/bp_maquinamooresimple/uut/dff_a/q
/bp_maquinamooresimple/uut/dff_b/q
/bp_maquinamooresimple/uut/state
/bp_maquinamooresimple/uut/z
1
0
1
1
0
10
1
00
10
00
01
10
Now
800 ns
200 ns
400 ns
600 ns
800 ns
0 ns to 834 ns
Now: 800 ns Delta: 1

Fig. 6.10

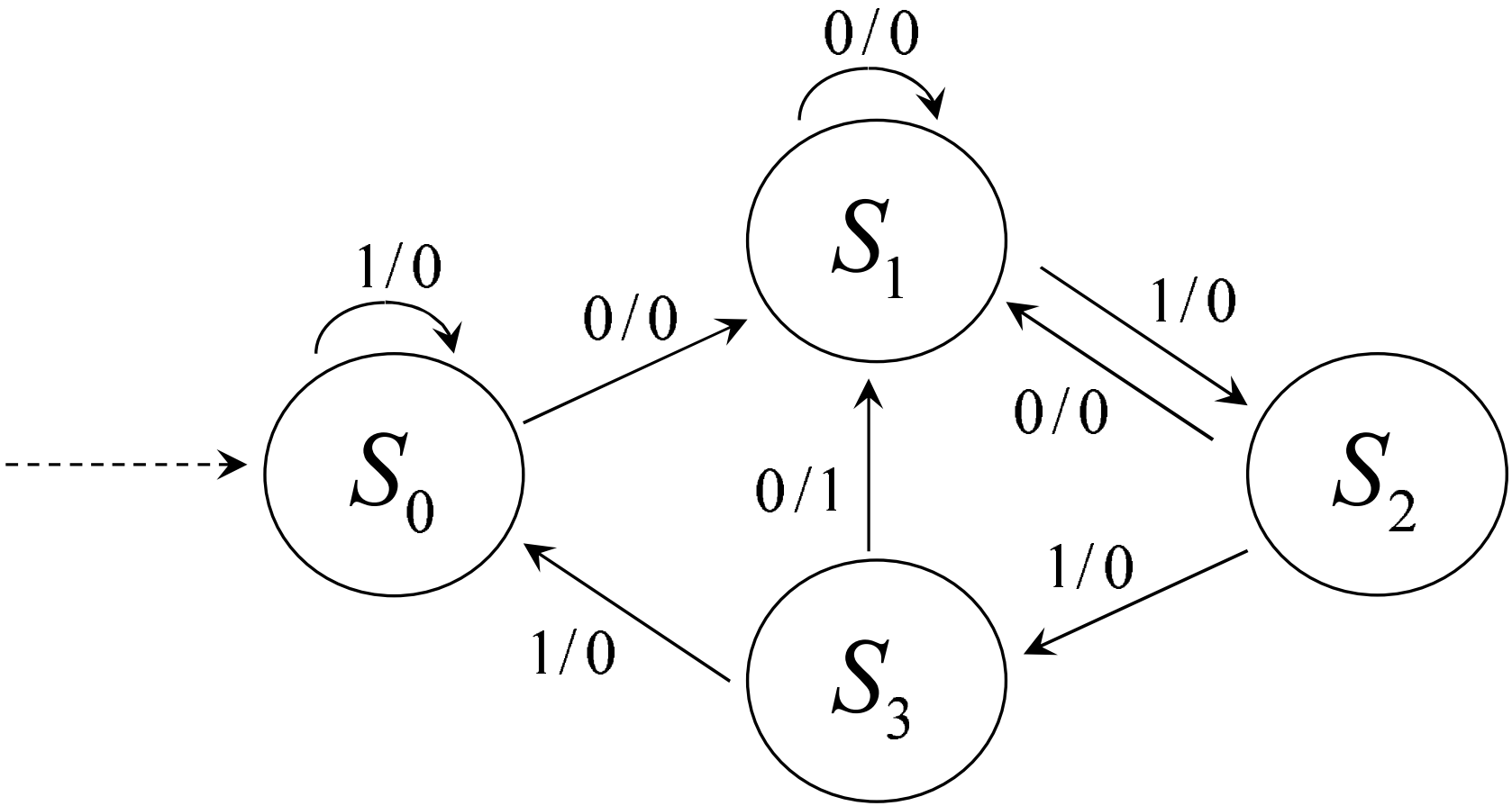


Fig. 6.11

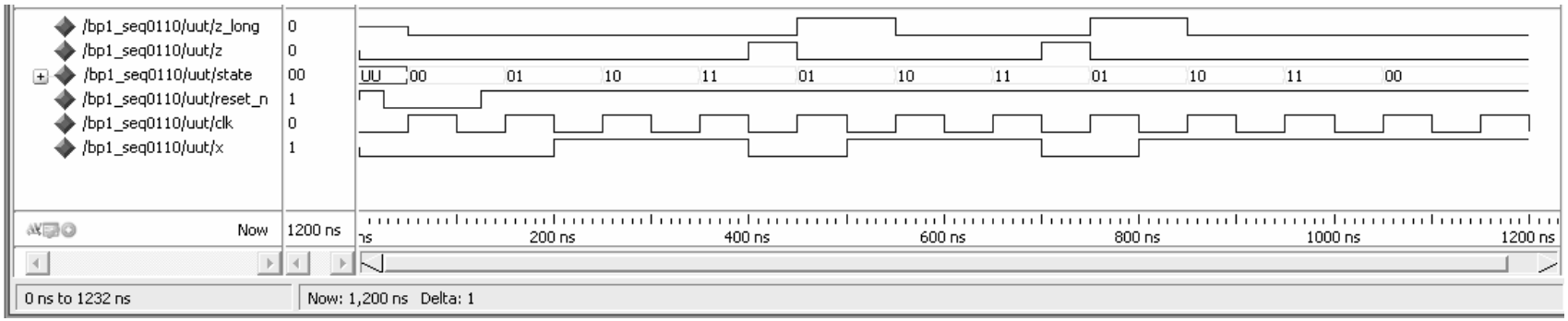

/bp1_seq0110/uut/z_long
/bp1_seq0110/uut/z
/bp1_seq0110/uut/state
/bp1_seq0110/uut/reset_n
/bp1_seq0110/uut/clk
/bp1_seq0110/uut/x
0
0
00
1
0
1
UU
00
01
10
11
01
10
11
01
10
11
00
Now
1200 ns
200 ns
400 ns
600 ns
800 ns
1000 ns
1200 ns
0 ns to 1232 ns
Now: 1,200 ns Delta: 1

Fig. 6.12

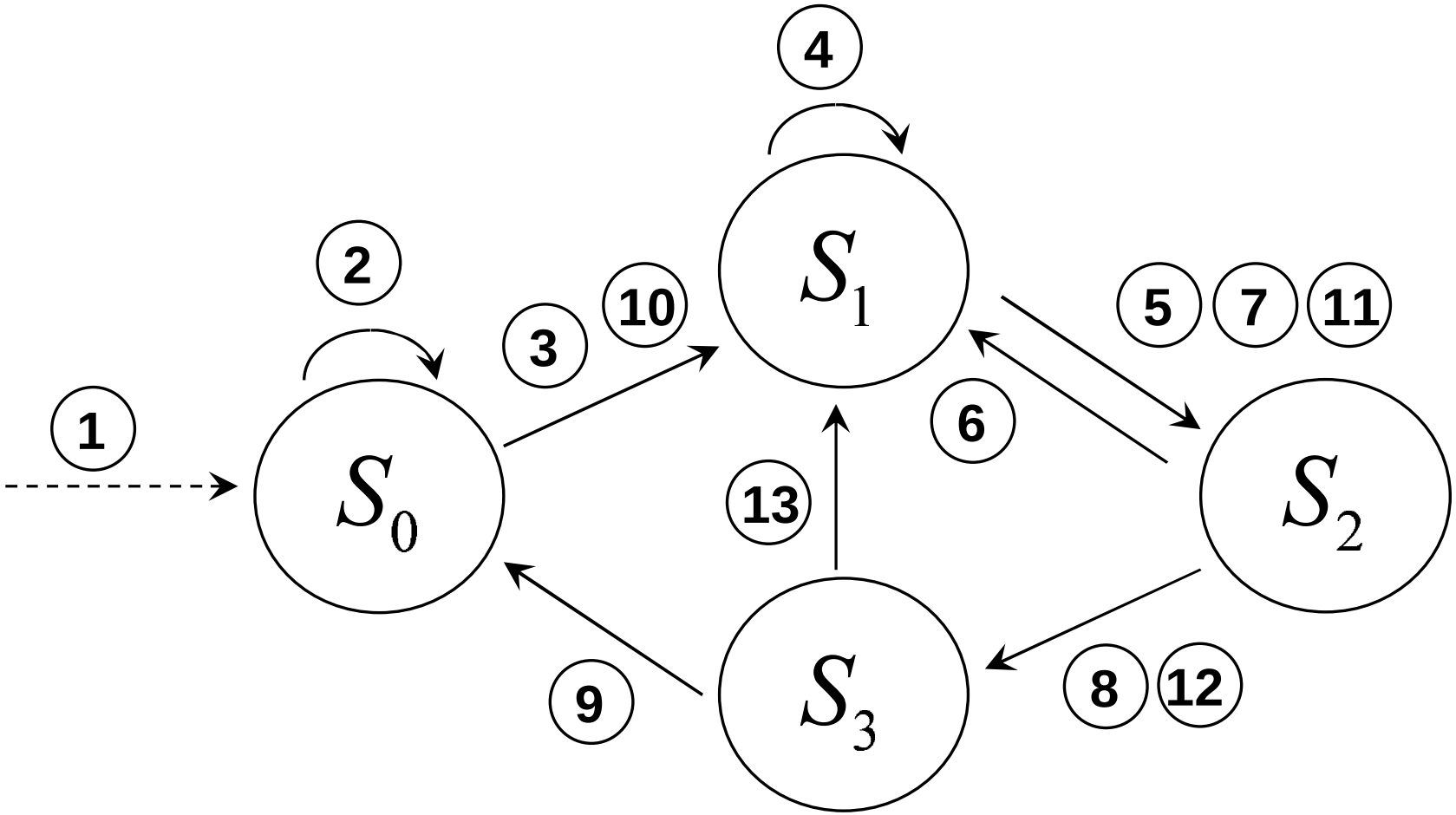


Fig. 6.13

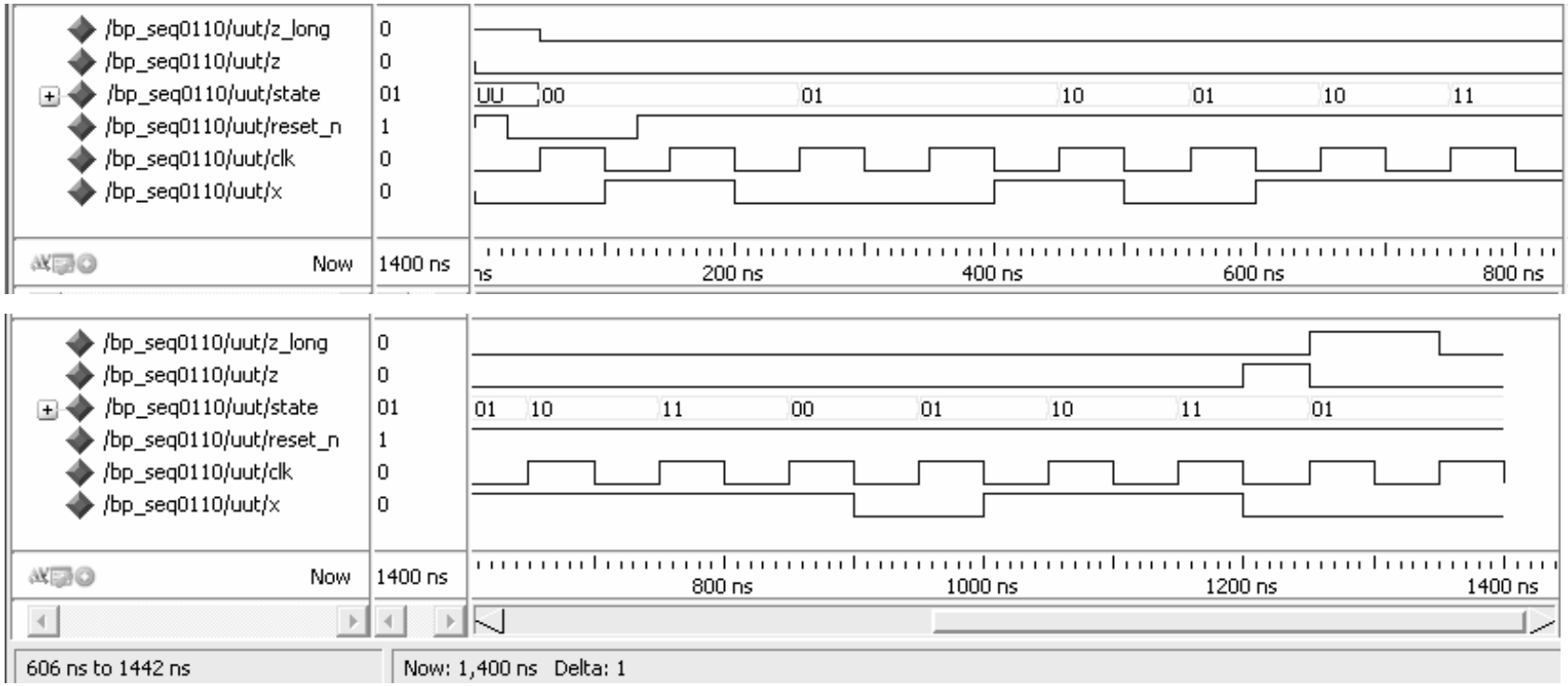

/bp_seq0110/uut/z_long
/bp_seq0110/uut/z
/bp_seq0110/uut/state
/bp_seq0110/uut/reset_n
/bp_seq0110/uut/clk
/bp_seq0110/uut/x
0
0
01
1
0
0
UU
00
01
10
01
10
11
Now
1400 ns
200 ns
400 ns
600 ns
800 ns
/bp_seq0110/uut/z_long
/bp_seq0110/uut/z
/bp_seq0110/uut/state
/bp_seq0110/uut/reset_n
/bp_seq0110/uut/clk
/bp_seq0110/uut/x
0
0
01
1
0
0
01
10
11
00
01
10
11
01
Now
1400 ns
800 ns
1000 ns
1200 ns
1400 ns
606 ns to 1442 ns
Now: 1,400 ns Delta: 1

Fig. 6.14

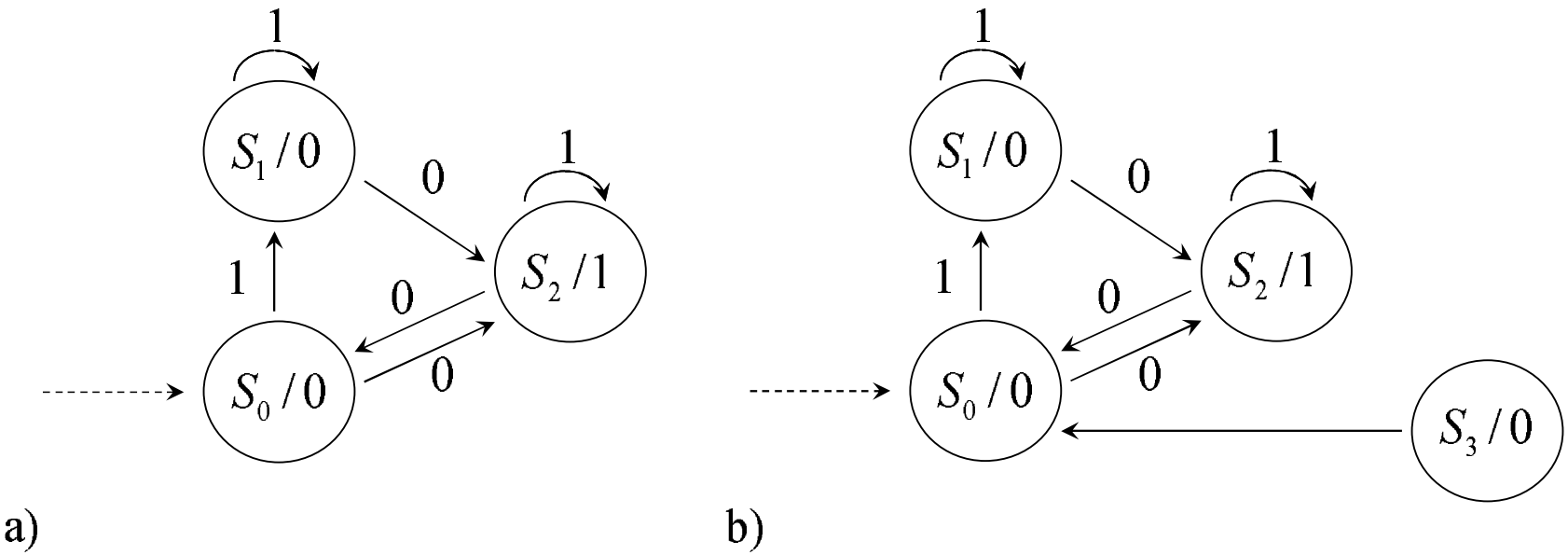


Fig. 6.15

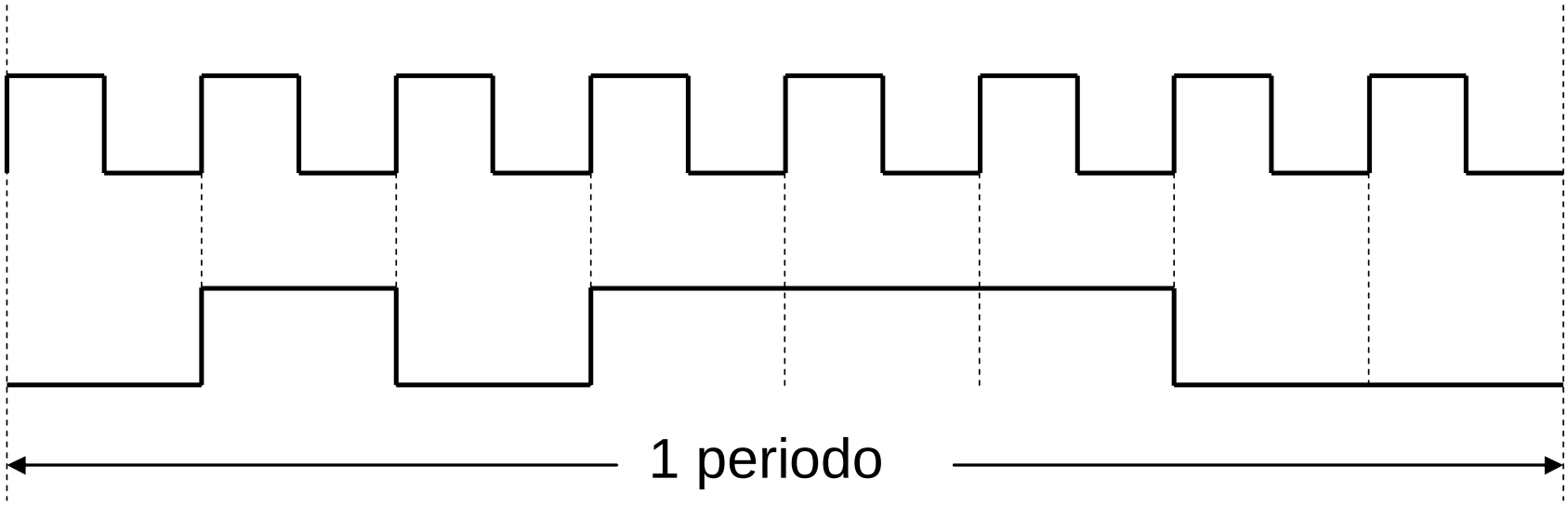


Fig. 6.16

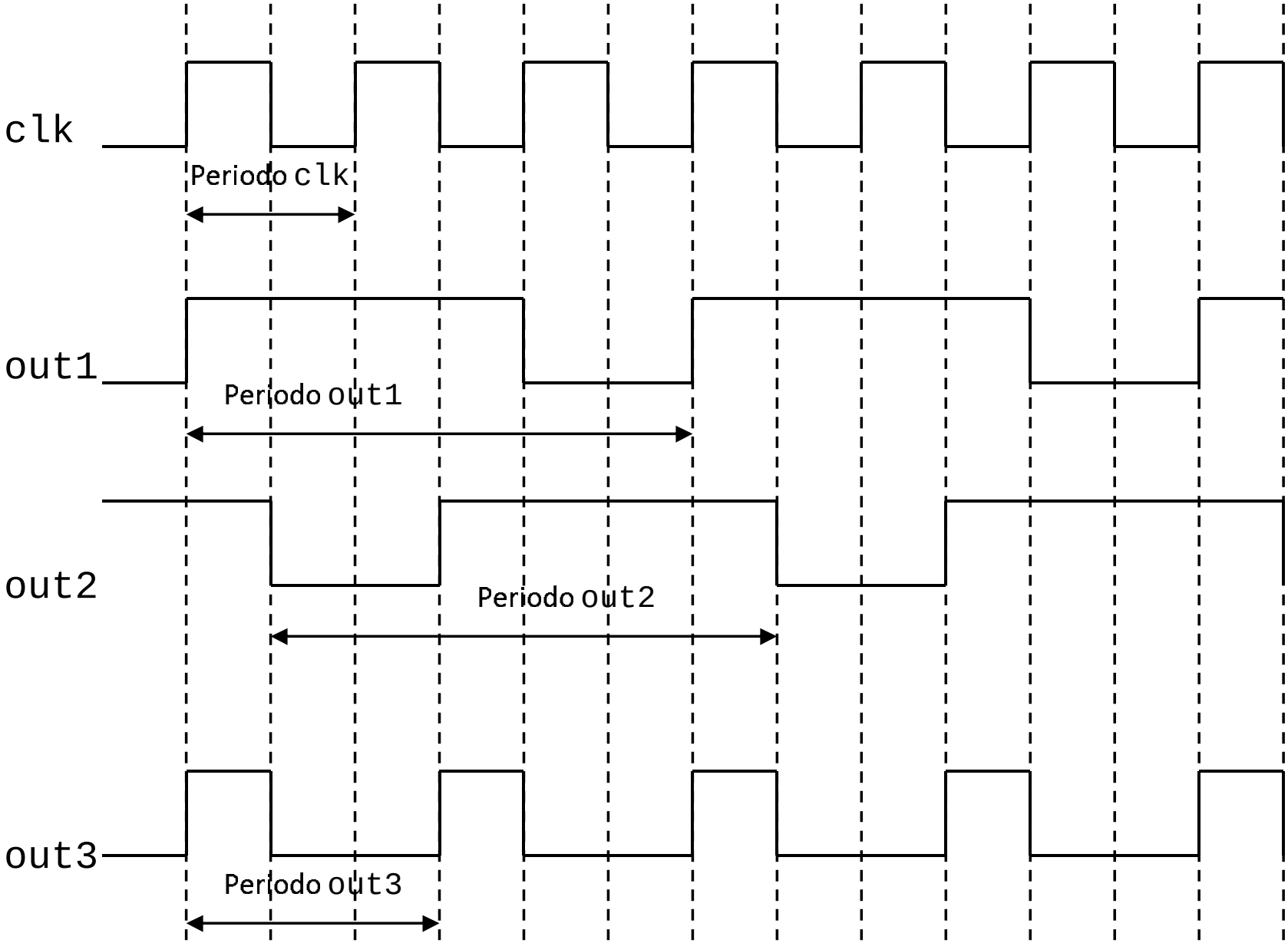

clk
Periodo clk
out1
Periodo out1
out2
Periodo out2
out3
Periodo out3

Fig. 6.17

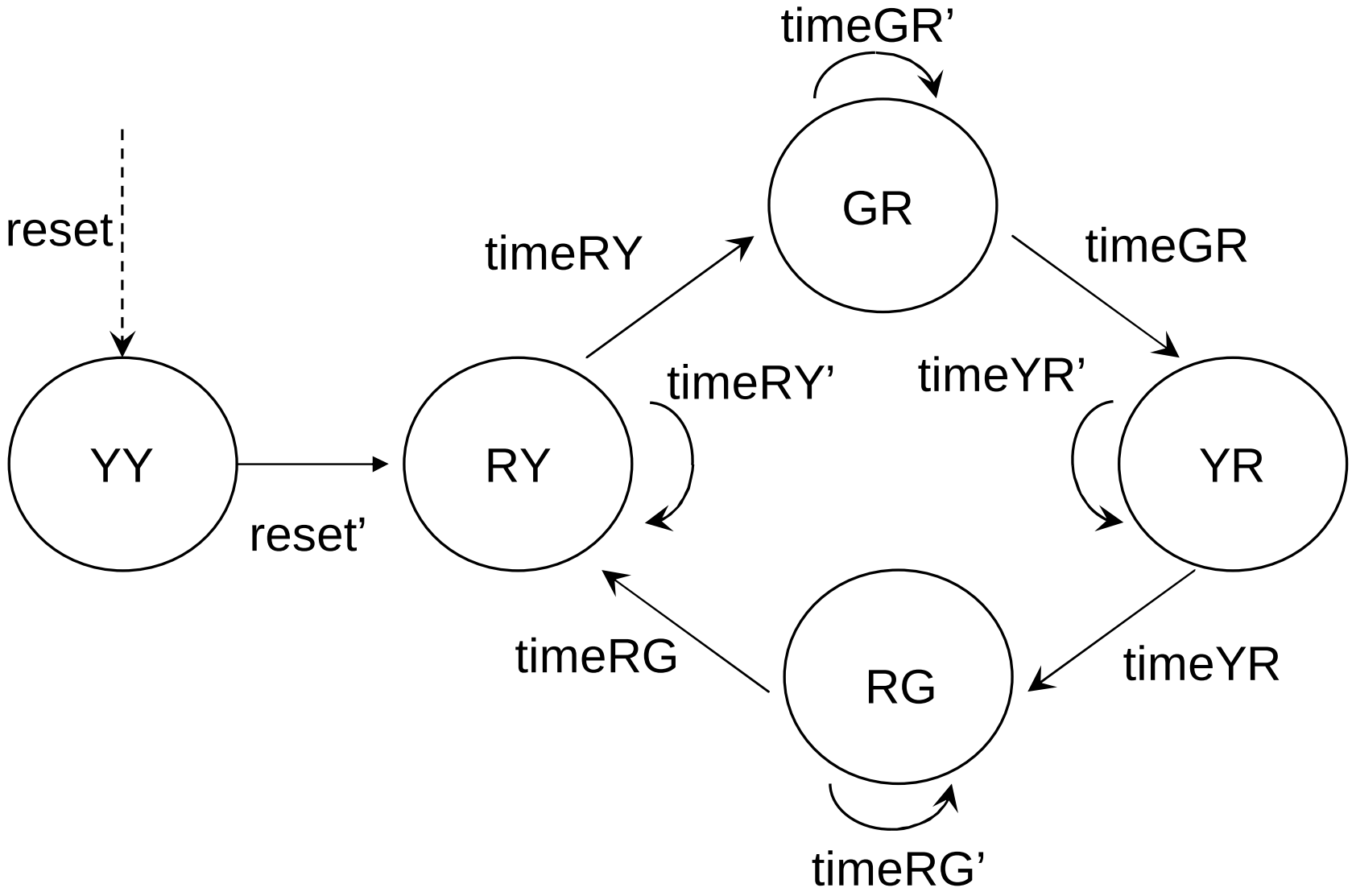

timeGR'
reset
GR
timeRY
timeGR
timeRY'
timeYR'
YY
RY
YR
reset'
timeRG
RG
timeYR
timeRG'

Fig. 6.18

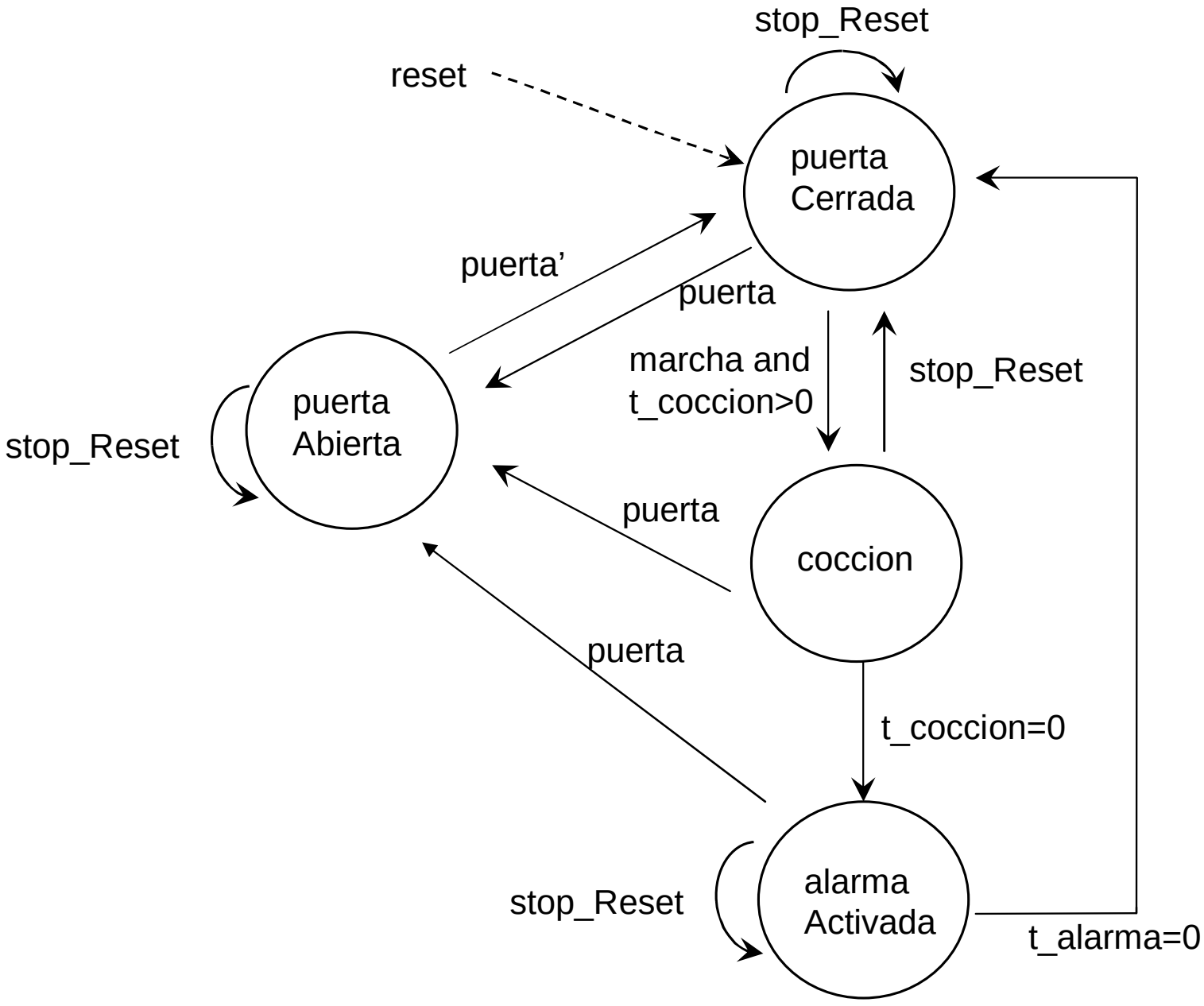

stop_Reset
reset
puerta
Cerrada
puerta'
puerta
marcha and
t_coccion>0
stop_Reset
puerta
Abierta
stop_Reset
puerta
coccion
puerta
t_coccion=0
alarma
Activada
stop_Reset
t_alarma=0

Fig. 6.19

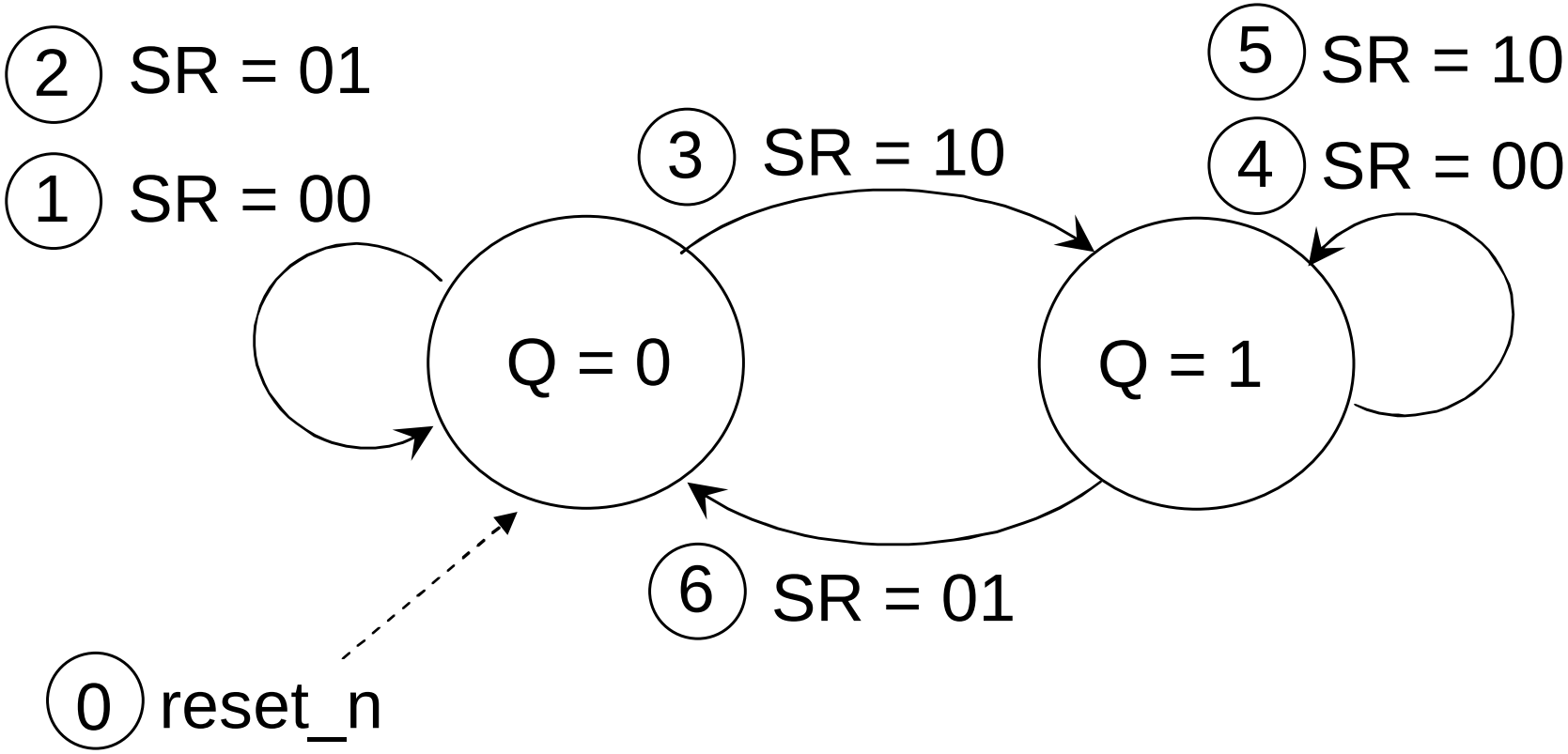


Fig. 6.20

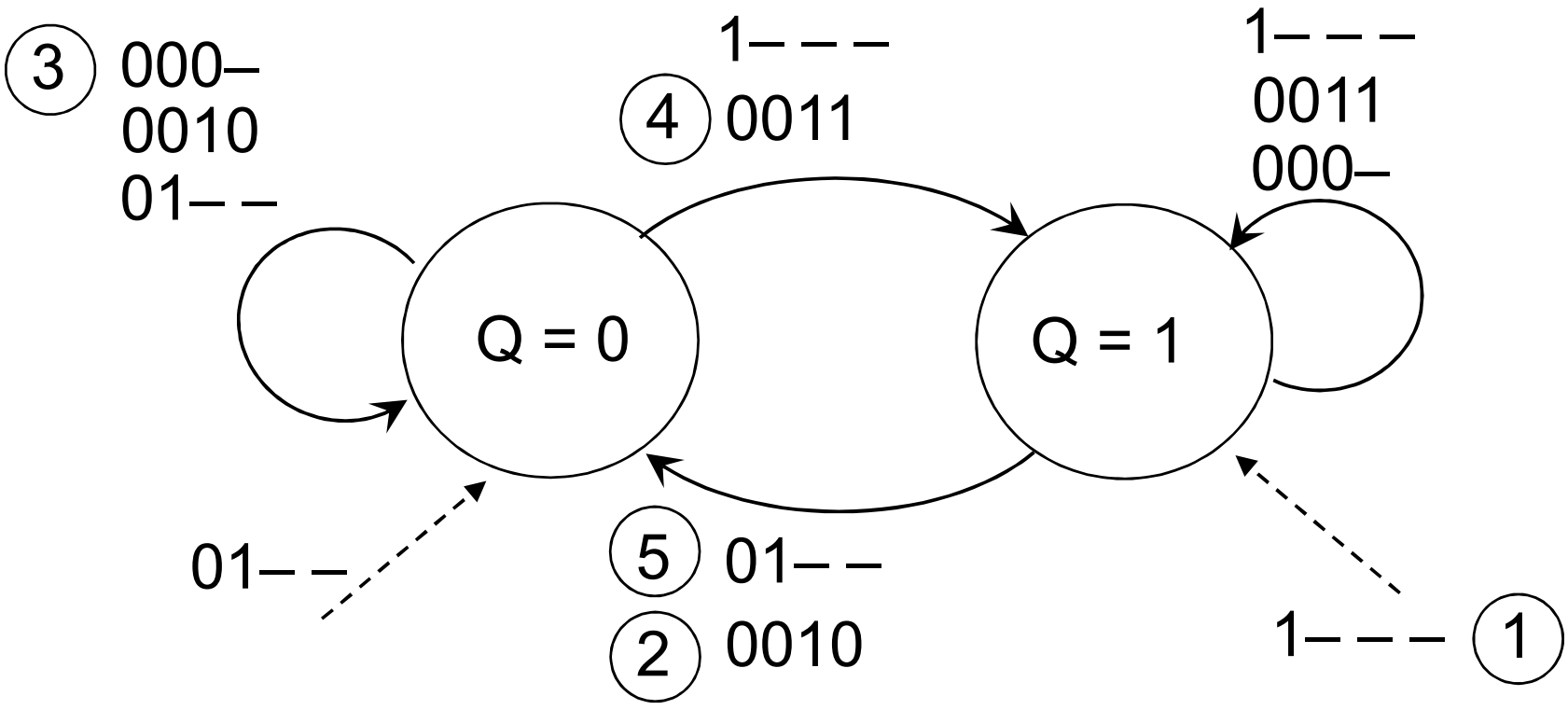


Fig. 6.21

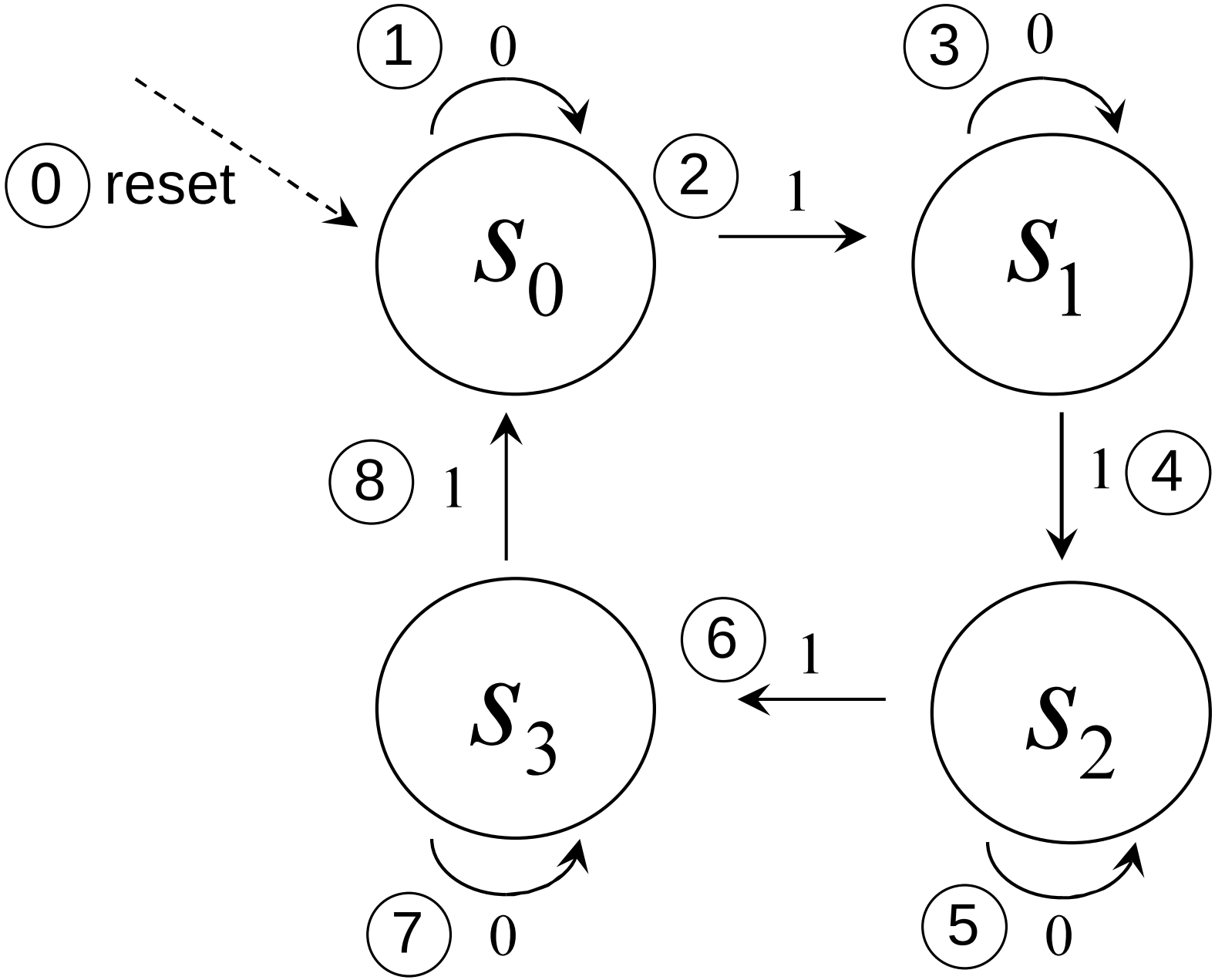


Fig. 6.22

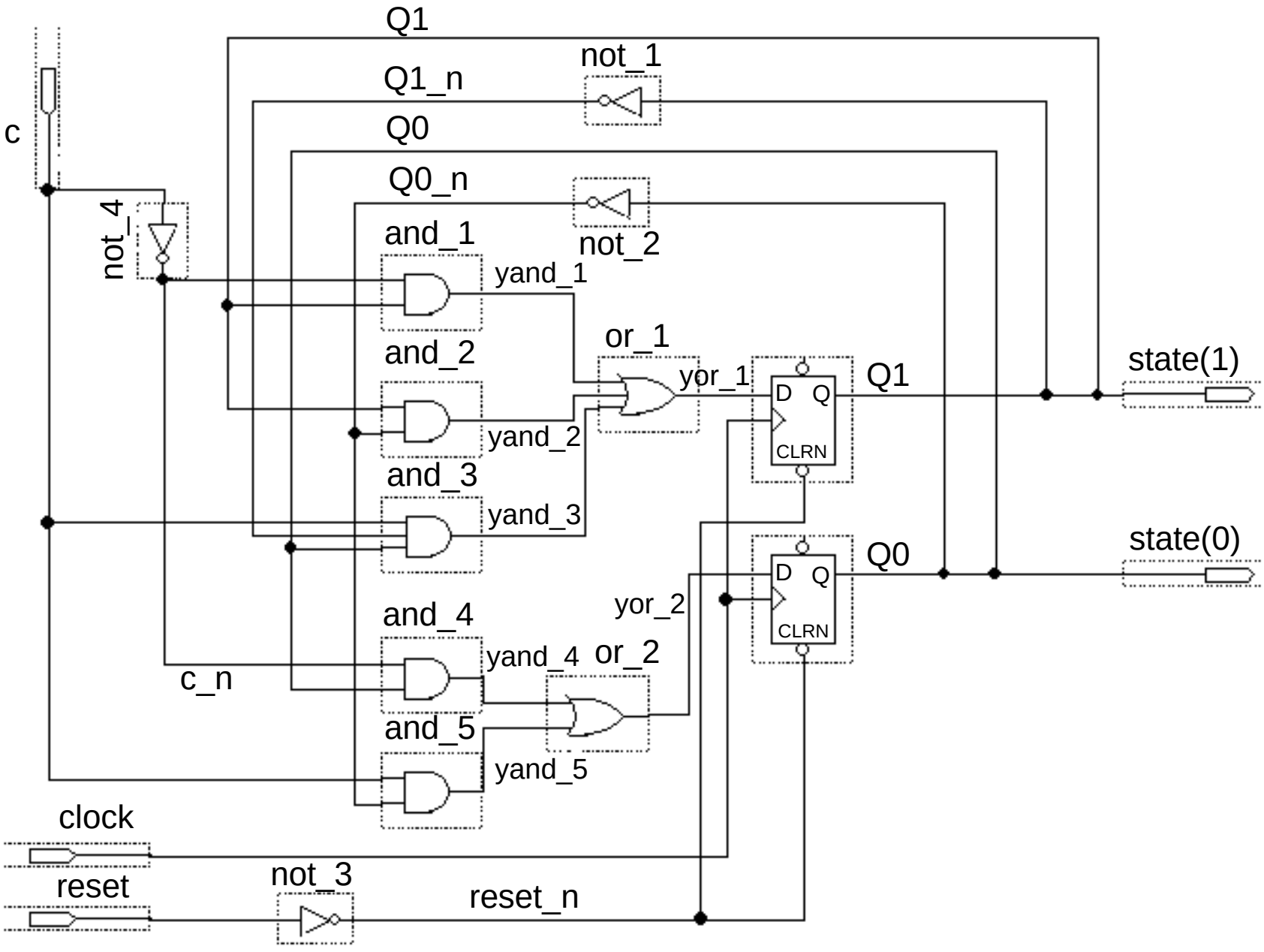

Q1
not_1
Q1_n
c
Q0
Q0_n
not_4
not_2
and_1
yand_1
or_1
and_2
yor_1
Q1
state(1)
D Q
CLRN
yand_2
and_3
yand_3
state(0)
Q0
D Q
CLRN
yor_2
and_4
yand_4
or_2
c_n
and_5
yand_5
clock
reset
not_3
reset_n

Fig. 6.23

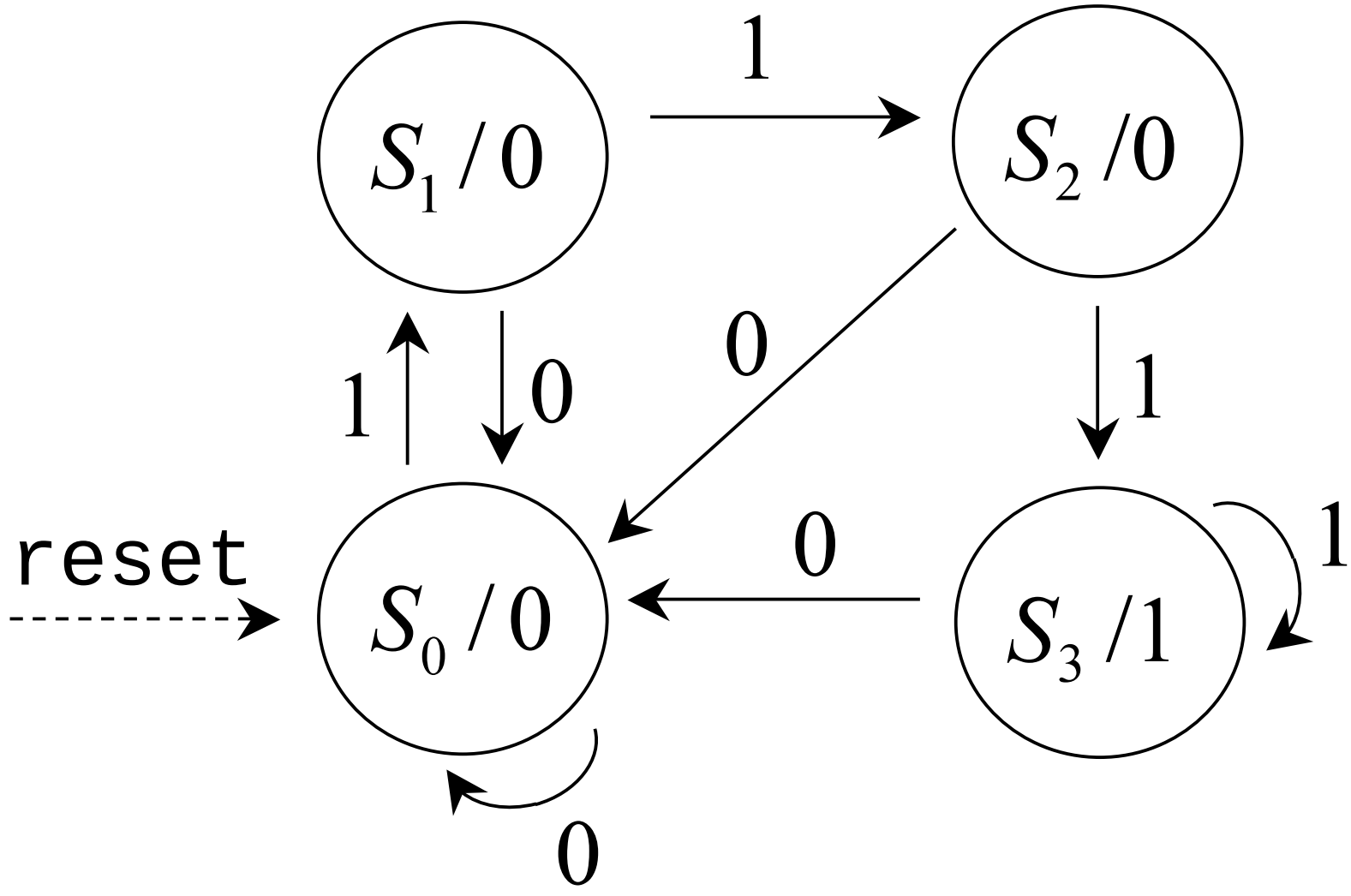


Fig. 6.24

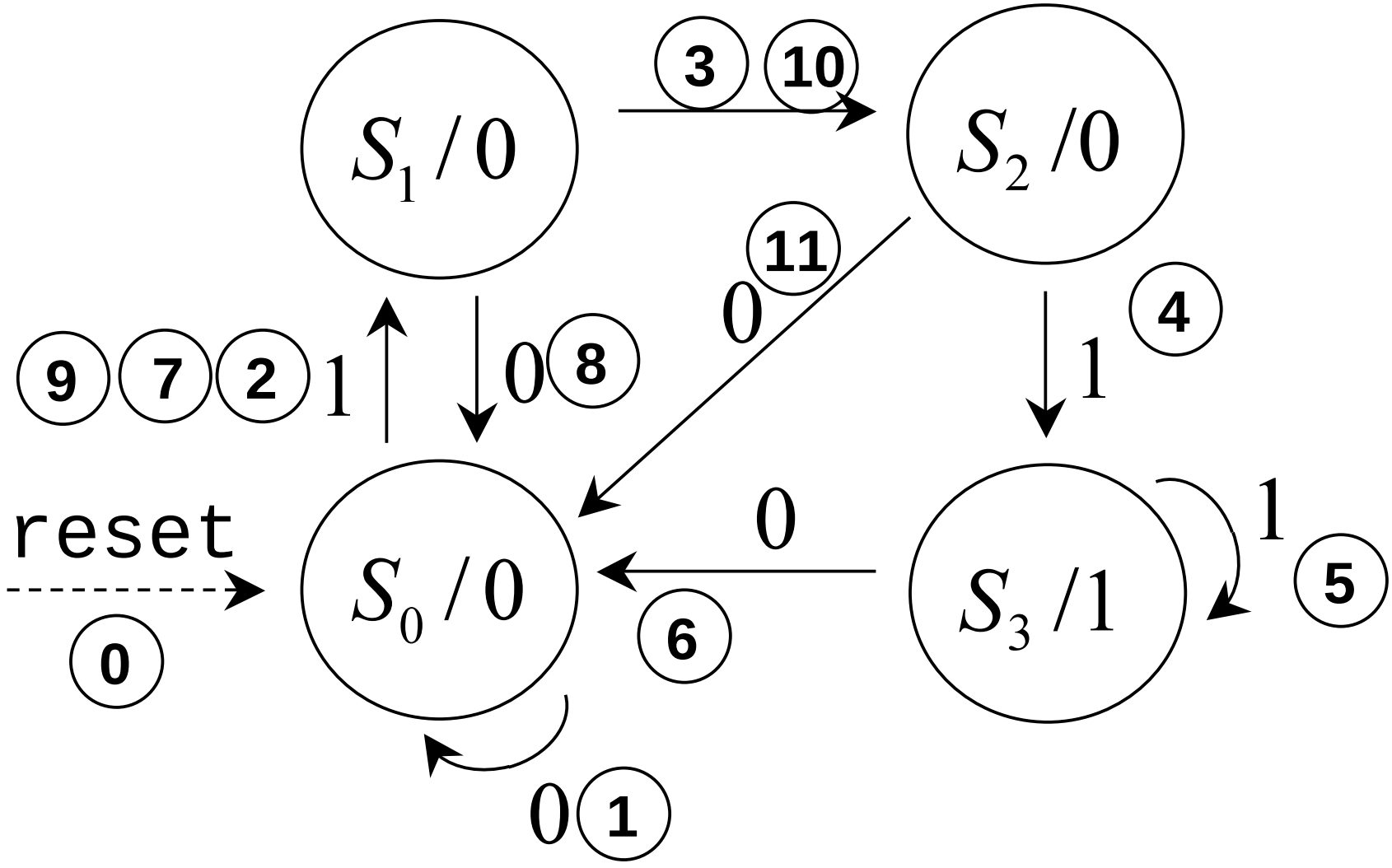


Fig. 6.25

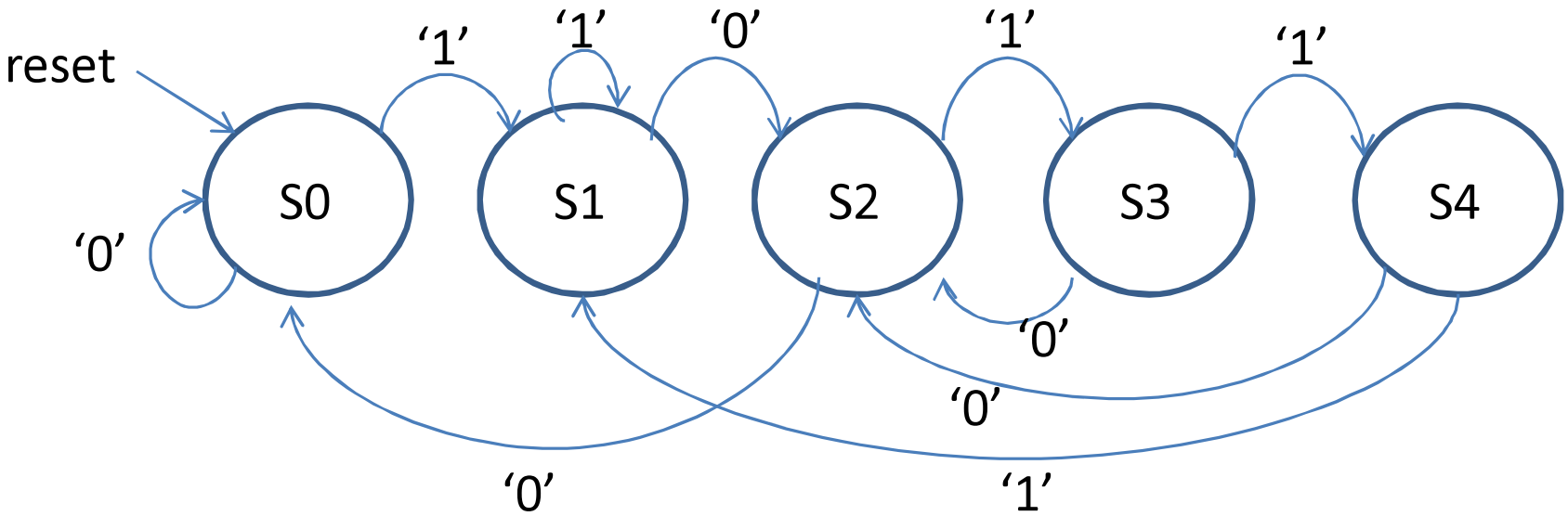


Fig. 6.26

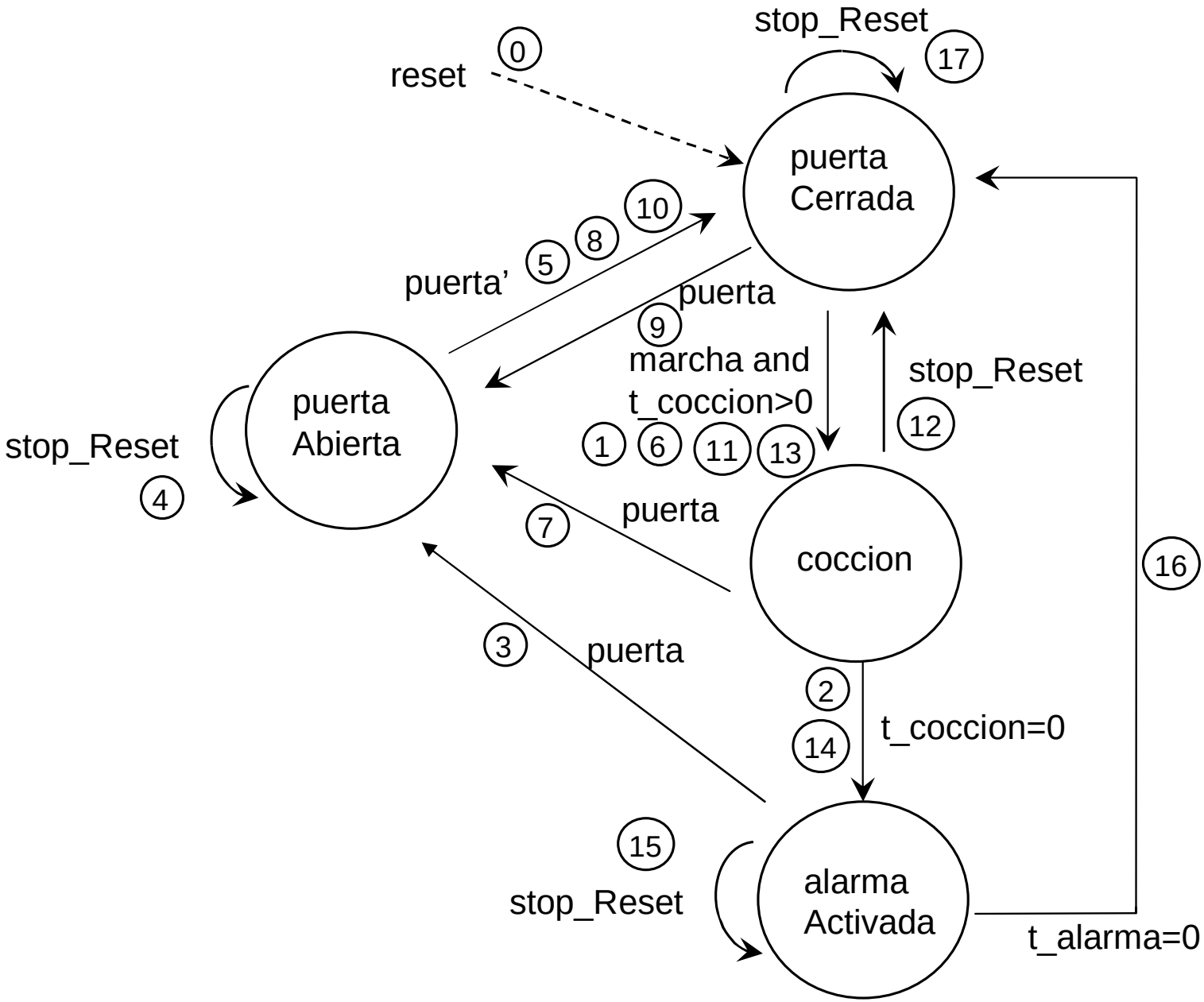

stop_Reset
17
reset
0
puerta
Cerrada
10
8
5
puerta'
puerta
9
marcha and
t_coccion>0
stop_Reset
12
puerta
Abierta
stop_Reset
4
1
6
11
13
puerta
7
coccion
16
3
puerta
2
14
t_coccion=0
15
stop_Reset
alarma
Activada
t_alarma=0

Fig. 6.27

Señal	Significado
r1	Pone el semáforo 1 en rojo
r2	Pone el semáforo 2 en rojo.
g1	Pone el semáforo 1 en verde.
g2	Pone el semáforo 2 en verde.
y1	Pone el semáforo 1 en amarillo.
y2	Pone el semáforo 2 en amarillo.

Tabla 6.1

Estado	Tiempo	Salida
RG	timeRG (30 s)	`r1 = '1', g2 = '1'`
RY	timeRY (5 s)	`r1 = '1', y2 = '1'`
GR	timeGR (45 s)	`g1 = '1', r2 = '1'`
YR	timeYR (5 s)	`y1 = '1', r2 = '1'`
YY	–	`y1 = '1', y2 = '1'`

Tabla 6.2

Entrada	Significado
`minuto`	Incrementa el contador de tiempo de cocción en 60 seg. En cualquier momento se puede apretar este botón para aumentar el periodo de cocción.
`marcha`	Inicia la marcha del horno siempre que el tiempo de cocción sea mayor que 0.
`stop_Reset`	Si el microondas se encuentra en estado de cocción, cuando esta señal está a '1' se detiene la marcha del horno. Si se activa esta señal cuando el horno no está en estado de cocción se resetea el tiempo de cocción.
`puerta`	Según su valor sea '1' ó '0' indica si la puerta del microondas está abierta o cerrada. Abrir la puerta en el periodo de cocción interrumpe la cocción. Abrir la puerta cuando la alarma está activada la desconecta.
`clk`	Reloj de entrada con periodo de 125 ms.
`reset`	Señal asíncrona para resetear el sistema.

Tabla 6.3

Salida	Significado
`segundos(9...0)`	Indican a una pantalla el número de segundos de cocción restantes. La codificación es binaria, por lo que se pueden programar hasta 1023 seg.
`calentar`	Cuando está a '1' el horno calienta.
`luz`	Cuando está a '1' enciende la luz interna del horno.
`alarma`	Cuando está a '1' suena la alarma. La alarma debe sonar cuando el tiempo de cocción llegue a 0 después de que el horno ha estado calentando. La alarma se debe desconectar cuando se abre la puerta del horno.

Tabla 6.4

Estado	Significado
PuertaAbierta	Microondas parado con la puerta abierta. Las señales de salida tienen los siguientes valores: `calentar = 0`, `luz = 1` y `alarma = 0`
PuertaCerrada	Microondas parado con la puerta cerrada. Las señales de salida tienen los siguientes valores: `calentar = 0`, `luz = 0` y `alarma = 0`
Coccion	El microondas está cociendo. Las señales de salida tienen los siguientes valores: `calentar = 1`, `luz = 1` y `alarma = 0`
AlarmaActivada	La alarma se encuentra activa. Las señales de salida tienen los siguientes valores: `calentar = 0`, `luz = 0` y `alarma = 1`.

Tabla 6.5

Estado actual		Entrada	Próximo estado		Salida	
$Q1$	$Q0$	c	$D1$	$D0$	$State(1:0)$	
0	0	0	0	0	0	0
0	0	1	0	1	0	0
0	1	0	0	1	0	1
0	1	1	1	0	0	1
1	0	0	1	0	1	0
1	0	1	1	1	1	0
1	1	0	1	1	1	1
1	1	1	0	0	1	1

Tabla 6.6

TEMA 7

Metodología de transferencia entre registros

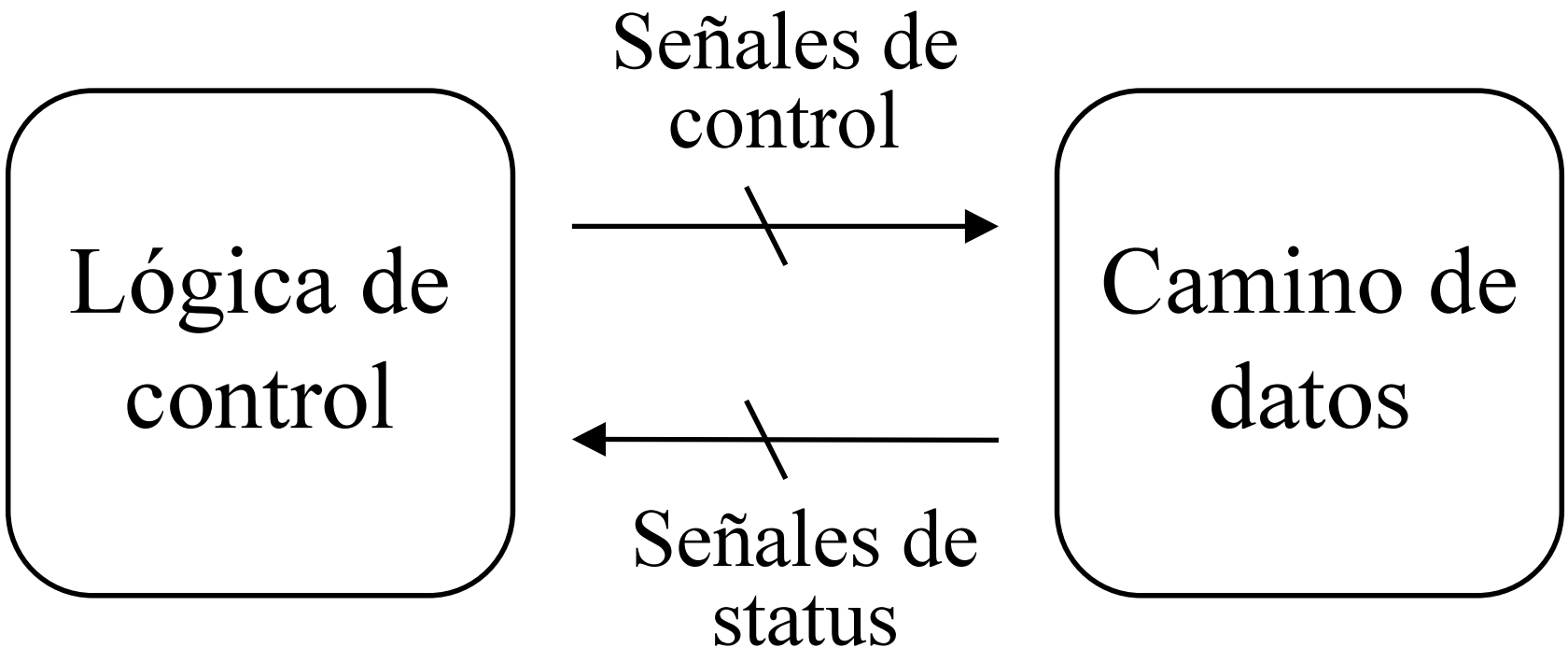

Señales de control
Lógica de control
Camino de datos
Señales de status

Fig. 7.1

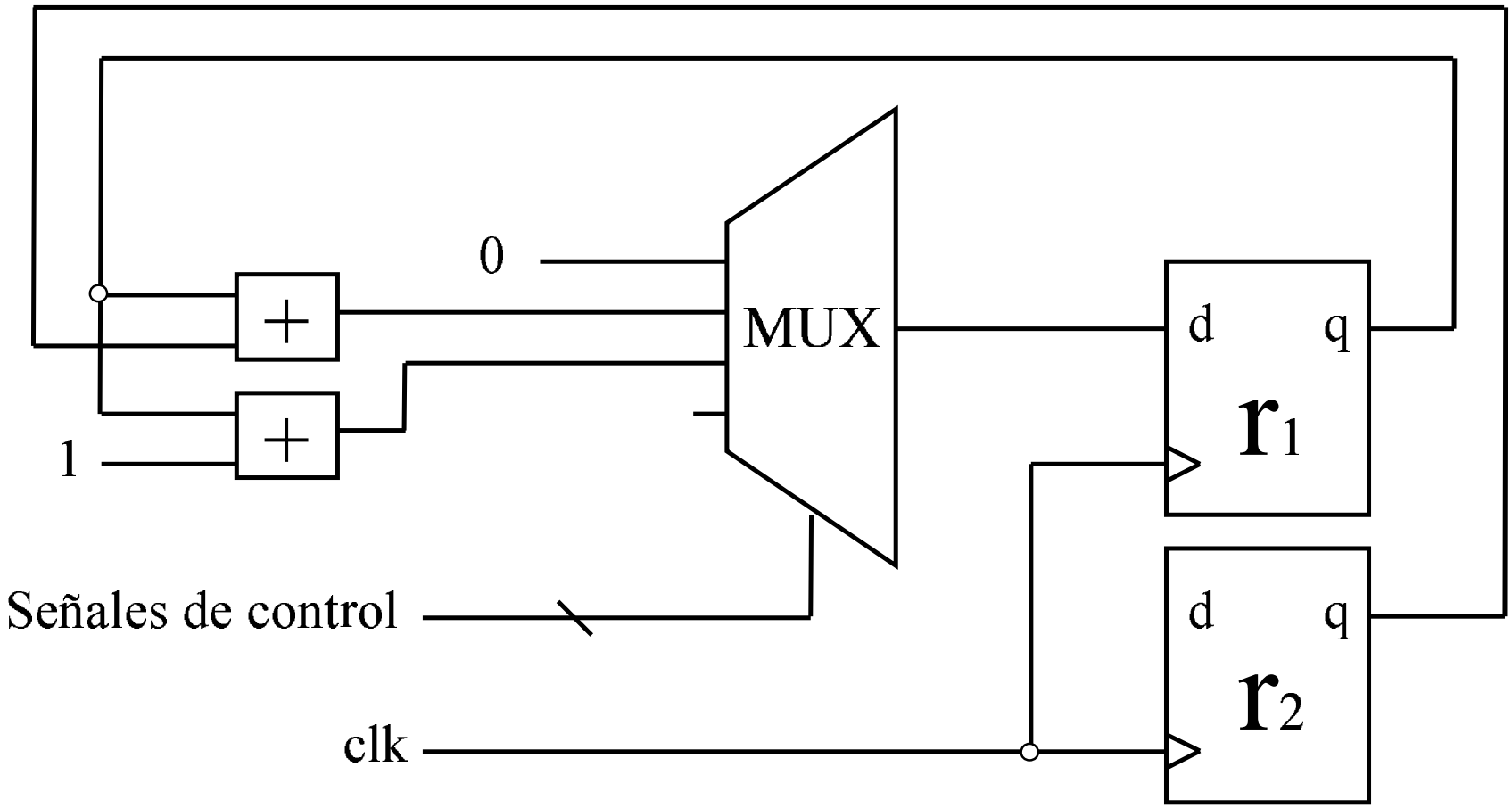


Fig. 7.2

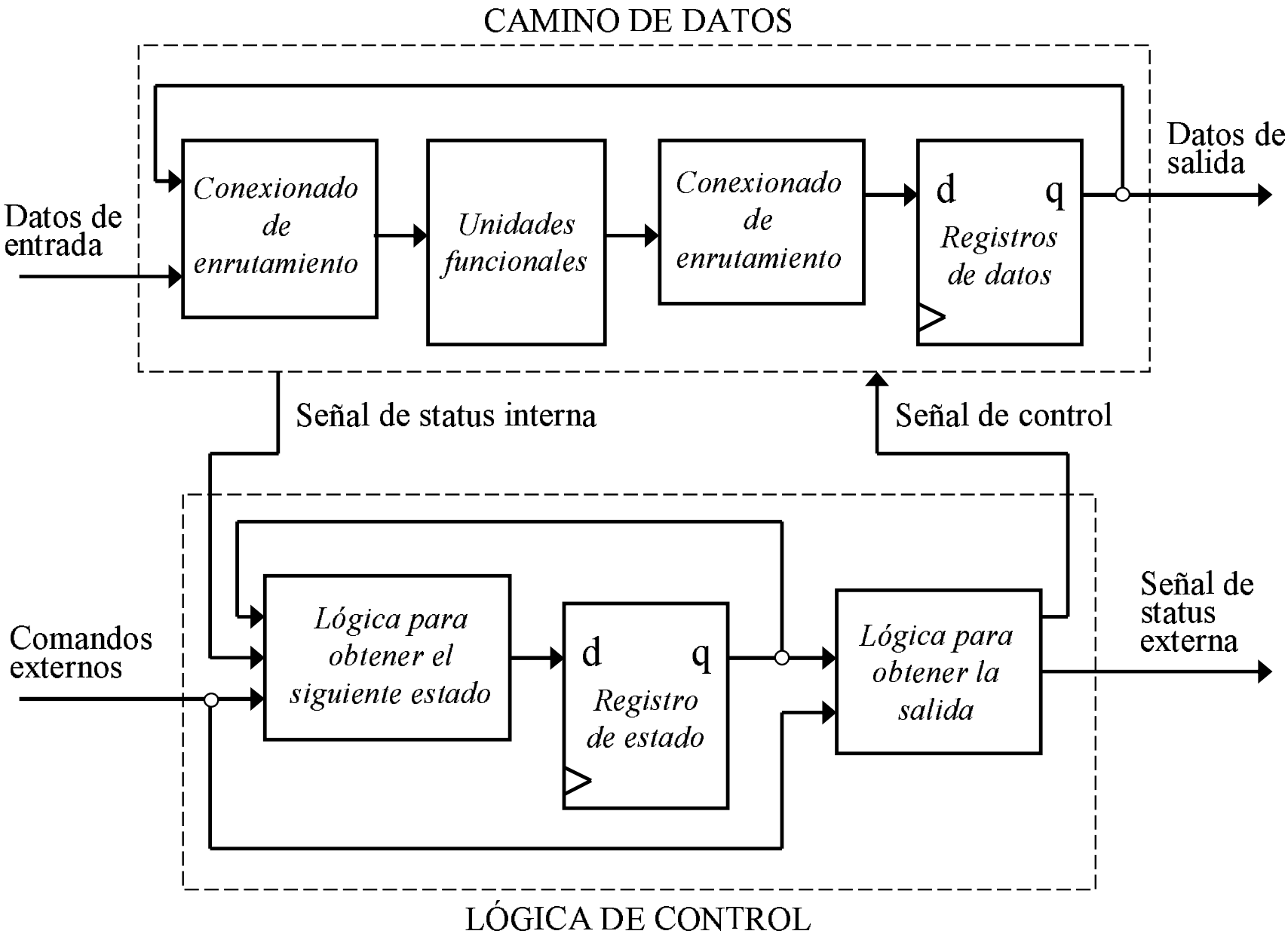

CAMINO DE DATOS
Datos de entrada
Conexionado de enrutamiento
Unidades funcionales
Conexionado de enrutamiento
d
q
Registros de datos
Datos de salida
Señal de status interna
Señal de control
Comandos externos
Lógica para obtener el siguiente estado
d
q
Registro de estado
Lógica para obtener la salida
Señal de status externa
LÓGICA DE CONTROL

Fig. 7.3

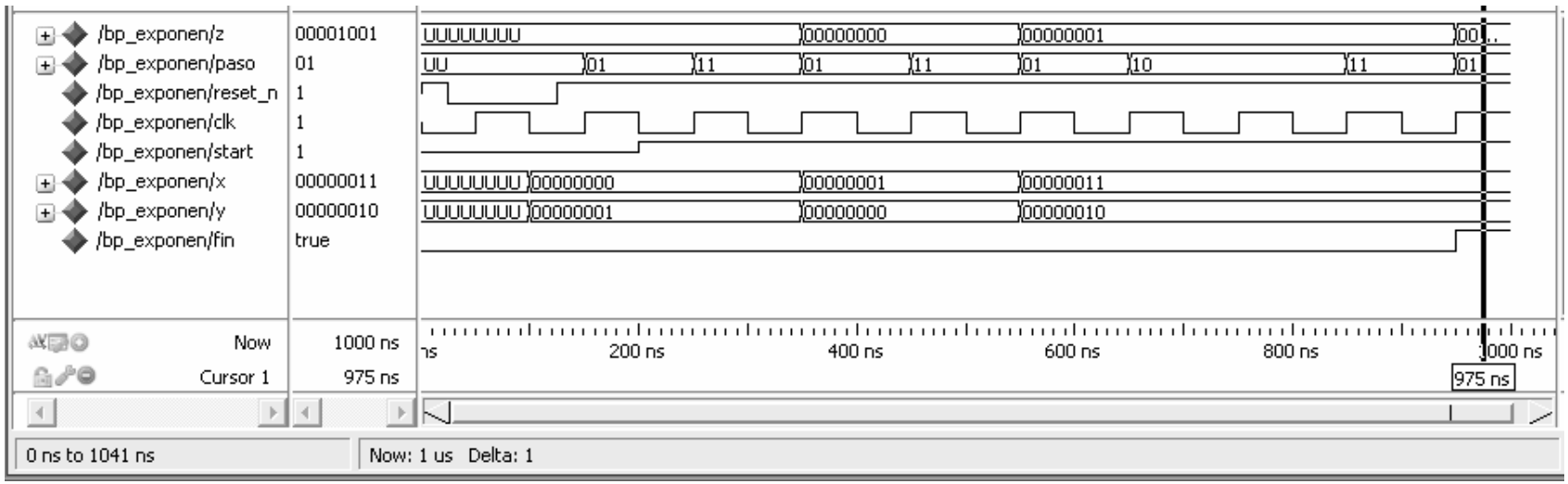

/bp_exponen/z
/bp_exponen/paso
/bp_exponen/reset_n
/bp_exponen/clk
/bp_exponen/start
/bp_exponen/x
/bp_exponen/y
/bp_exponen/fin
00001001
01
1
1
1
00000011
00000010
true
UUUUUUUU
00000000
00000001
UU
01
11
10
UUUUUUUU
Now
1000 ns
Cursor 1
975 ns
200 ns
400 ns
600 ns
800 ns
0 ns to 1041 ns
Now: 1 us Delta: 1

Fig. 7.4

Sentencia VHDL	Operación RT
`X <= Y;`	Transfiere el contenido de Y a X
`X <= 0;`	Guarda en X el valor decimal 0
`X <= (others => '1');`	Pone todos los bits de X al valor '1'
`X <= 1;`	Guarda en X el valor decimal 1
`X <= X slr 1;`	Desplaza el contenido de X un bit hacia la derecha e introduce el valor '0' en el bit más significativo (MSB)
`X <= X ror 4;`	Desplaza el contenido de X cuatro bits hacia la derecha de manera circular
`X <= M( 16#C2# );`	Transfiere a X el contenido del elemento $C2_{16}$ (elemento 194_{10}) de M
`X <= Y or Z;`	$X \leftarrow Y$ or Z (or bit a bit)
`X <= Y and Z;`	$X \leftarrow Y$ and Z (and bit a bit)
`X <= Y xor Z;`	$X \leftarrow Y$ xor Z (xor bit a bit)
`X <= not Y;`	$X \leftarrow$ complemento a 1 de Y
`X <= -Y;`	$X \leftarrow$ complemento a 2 de Y
`X <= Y + Z;`	$X \leftarrow Y + Z$
`X <= Y - Z;`	$X \leftarrow Y - Z$
`X <= (c and Y) or ((not c) and Z);`	$X \leftarrow (c$ and $Y)$ or $(\bar{c}$ and $Z)$
`X <= Y when c else Z;`	Equivalente a la anterior
`if (c) then X <= Y; else X <= Z;`	Equivalente a la anterior

Tabla 7.1

Entrada	Descripción
`m10`	Introducida moneda de 10 céntimos.
`m20`	Introducida moneda de 20 céntimos.
`m50`	Introducida moneda de 50 céntimos.
`devolución`	Solicitud de devolución del dinero.
`selec_agua`	Selección de producto: una botella de agua.
`selec_refresco`	Selección de producto: una lata de refresco.

Tabla 7.2

Señal	Descripción
`total`	Dinero total introducido por el cliente.
`beep`	Se activa cuando el cliente selecciona el producto sin haber introducido dinero suficiente.
`devuelve_moneda`	Se activa para devolver la última moneda introducida, si esta moneda hace que se exceda el dinero total máximo que acepta la máquina.
`dispensa_refresco`	Dispensa una lata de refresco.
`dispensa_agua`	Dispensa una botella de agua.
`devuelve_cambio`	Devuelve el dinero restante.

Tabla 7.3

Entrada	data
m10	"001"
m20	"010"
m50	"101"
devolución	"100"
selec_agua	"011"
selec_refresco	"000"

Tabla 7.4