

```

/* Testbench for ex41.sv
*/

module ex41_tb0();
    logic a, b, c, y;

    ex41 dut(a, b, c, y);

    initial begin
        $display("a b c  y");
        $display("-----");
        $monitor("%b %b %b | %b", a,b,c,y);
        a=0; b=0; c=0; #10;
            c=1; #10;
            b=1; c=0; #10;
            c=1; #10;
        a=1; b=0; c=0; #10;
            c=1; #10;
            b=1; c=0; #10;
            c=1; #10;
        $display("-----");
    end
endmodule

```

HARDWARE DESCRIPTION
LANGUAGES

SYSTEMVERILOG

TESTING HDL MODULES

- ▶ Modules are tested using simulators able to interpret the HDL code
- ▶ “A **testbench** is an HDL module that is used to test another module, called the **device under test** (DUT). The testbench contains statements to **apply inputs** to the DUT and, ideally, **to check** that the **correct outputs** are produced. The **input** and **desired output patterns** are called **test vectors**.” (Harris-Harris)
- ▶ In particular, the testbench contains code to **instantiate** the DUT
- ▶ A test is performed processing both the DUT module and the testbench module with HDL tools

HDL TOOLS: ICARUS VERILOG

- ▶ Icarus Verilog is an open-source HDL toolset which supports parts of the Verilog and SystemVerilog standards
 - ▶ *“Icarus Verilog is a Verilog simulation and synthesis tool. It operates as a compiler, compiling source code written in Verilog (IEEE-1364) into some target format. For **batch simulation**, the compiler can generate an intermediate form called vvp assembly. This intermediate form is executed by the vvp command. For synthesis, the compiler generates netlists in the desired format.”*
- ▶ From <http://iverilog.icarus.com/home>

EXAMPLE 4.1

- ▶ The SystemVerilog code in the figure defines a combinational module with:
 - ▶ 3 single-bit inputs (a, b, c)
 - ▶ logic \rightarrow bit value(s)
 - ▶ 1 single-bit value output (y) defined as:
 - ▶ $y = \bar{a}\bar{b}\bar{c} + \bar{a}b\bar{c} + a\bar{b}c$
- ▶ To test the module a *test bench* is required

```
/* Example 4.1 */  
  
module ex41(input logic a, b, c, output  
logic y);  
    assign y = ~a & ~b & ~c |  
               a & ~b & ~c |  
               a & ~b & c;  
endmodule
```

EXAMPLE 4.1

- ▶ The code in the figure defines a test bench (ex41_tb0) module instantiating the module ex41 as a device under test (DUT)
- ▶ In the `initial` block the dut is tested against all the eight possible 3-bit inputs, one at a time with a delay of 10 time units after each input
 - ▶ The `$display` statements simply output messages to the console
 - ▶ The `$monitor` statement is used to output the input value along with the corresponding output every time the input changes

```
/* Testbench for ex41.sv
*/

module ex41_tb0();
    logic a, b, c, y;

    ex41 dut(a, b, c, y);

    initial begin
        $display("a b c   y");
        $display("-----");
        $monitor("%b %b %b | %b", a,b,c,y);
        a=0; b=0; c=0; #10;
                c=1; #10;
                b=1; c=0; #10;
                c=1; #10;
        a=1; b=0; c=0; #10;
                c=1; #10;
                b=1; c=0; #10;
                c=1; #10;
        $display("-----");
    end
endmodule
```

EXAMPLE 4.1

- ▶ Test of the `ex41` module code with test bench 0 (`ex41_tb0`)
 - ▶ Initial directory content

```
→ 0-HH-example-4.1-sillyfunction ls  
README.txt  ex41.sv    ex41_tb0.sv
```

DUT
CODE

TESTBENCH
CODE

EXAMPLE 4.1

- ▶ Test of the ex41 module code with test bench 0 (ex41_tb0)
- ▶ Execution of the SystemVerilog to vvp compiler (iverilog)

```
→ 0-HH-example-4.1-sillyfunction ls  
README.txt ex41.sv ex41_tb0.sv  
→ 0-HH-example-4.1-sillyfunction iverilog -o ex41_tb0 -g2012 ex41.sv ex41_tb0.sv
```

DUT
CODE

VVP CODE
FILENAME

TESTBENCH
CODE

EXAMPLE 4.1

- ▶ Test of the `ex41` module code with test bench 0 (`ex41_tb0`)
 - ▶ Directory content after execution of the SystemVerilog to vvp compiler

```
→ 0-HH-example-4.1-sillyfunction ls
README.txt  ex41.sv  ex41_tb0.sv
→ 0-HH-example-4.1-sillyfunction iverilog -o ex41_tb0 -g2012 ex41.sv ex41_tb0.sv
→ 0-HH-example-4.1-sillyfunction ls
README.txt  ex41_tb0  ex41.sv  ex41_tb0.sv
```

DUT
CODE

VVP
CODE

VVP CODE
FILENAME

TESTBENCH
CODE

EXAMPLE 4.1

- ▶ Test of the `ex41` module code with test bench 0 (`ex41_tb0`)
 - ▶ A simulation is carried on by executing the `vvp` code with the `vvp` command

```
→ 0-HH-example-4.1-sillyfunction ls
README.txt ex41.sv ex41_tb0.sv
→ 0-HH-example-4.1-sillyfunction iverilog -o ex41_tb0 -g2012 ex41.sv ex41_tb0.sv
→ 0-HH-example-4.1-sillyfunction ls
README.txt ex41_tb0 ex41.sv ex41_tb0.sv
→ 0-HH-example-4.1-sillyfunction vvp ex41_tb0
```

a	b	c	y
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	0

DUT CODE

VVP CODE

TESTBENCH CODE

EXAMPLE 4.1

- ▶ Test of the `ex41` module code with test bench 0 (`ex41_tb0`)
 - ▶ Simulation data is produced as output according to the test bench 0 code

```
→ 0-HH-example-4.1-sillyfunction ls  
README.txt ex41.sv ex41_tb0.sv  
→ 0-HH-example-4.1-sillyfunction iverilog -o ex41_tb0 -g2012 ex41.sv ex41_tb0.sv  
→ 0-HH-example-4.1-sillyfunction ls  
README.txt ex41_tb0 ex41.sv ex41_tb0.sv  
→ 0-HH-example-4.1-sillyfunction vvp ex41_tb0
```

a	b	c	y
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	0

DUT
CODEVVP
CODETESTBENCH
CODESIMULATION
OUTPUT

EXAMPLE 4.1 (TB 1)

- ▶ Test of the ex41 module code with test bench 1 (ex41_tb1)

```
module ex41_tb1();
  logic a, b, c, y;

  ex41 dut(a, b, c, y);

  initial begin
    a=0;    b=0;    c=0;    #10;
    assert (y==1) else $error("000 failed");
    c=1;    #10;
    assert (y==0) else $error("001 failed");
    b=1;    c=0;    #10;
    assert (y==0) else $error("010 failed");
    c=1;    #10;
    assert (y==0) else $error("011 failed");
    a=1;    b=0;    c=0;    #10;
    assert (y==1) else $error("100 failed");
    c=1;    #10;
    assert (y==0) else $error("111 failed");
  end
endmodule
```

EXAMPLE 4.1 (TB 2)

```

module ex41_tb2();
  logic clk, reset;
  logic a, b, c, y, yexpected;
  logic [31:0] vectornum, errors;
  logic [3:0] testvectors[0:7];

  ex41 dut(a, b, c, y);

  always
  begin
    clk=1; #5; clk=0; #5;
  end
  initial
  begin
    $dumpfile("ex41_tb2.vcd");
    $dumpvars(0, ex41_tb2);
    $readmemb("example.tv",
testvectors);
    vectornum=0; errors=0;
    reset=1; #27; reset=0;
  end
  always @(posedge clk)
  begin
    #1; {a, b, c, yexpected}
    =testvectors[vectornum];
  end

```

```

    always @(negedge clk)
    if(~reset)
    begin
      if(y!=yexpected) begin
        $display("Error: inputs=%b",
{a, b, c});
        $display(" outputs=%b (%b
expected)", y, yexpected);
        errors=errors+1;
      end
      vectornum=vectornum+1;
      if(testvectors[vectornum]===4'bx)
    begin
      $display("%d tests completed
with %d errors", vectornum, errors);
      $finish;
    end
    end
  endmodule

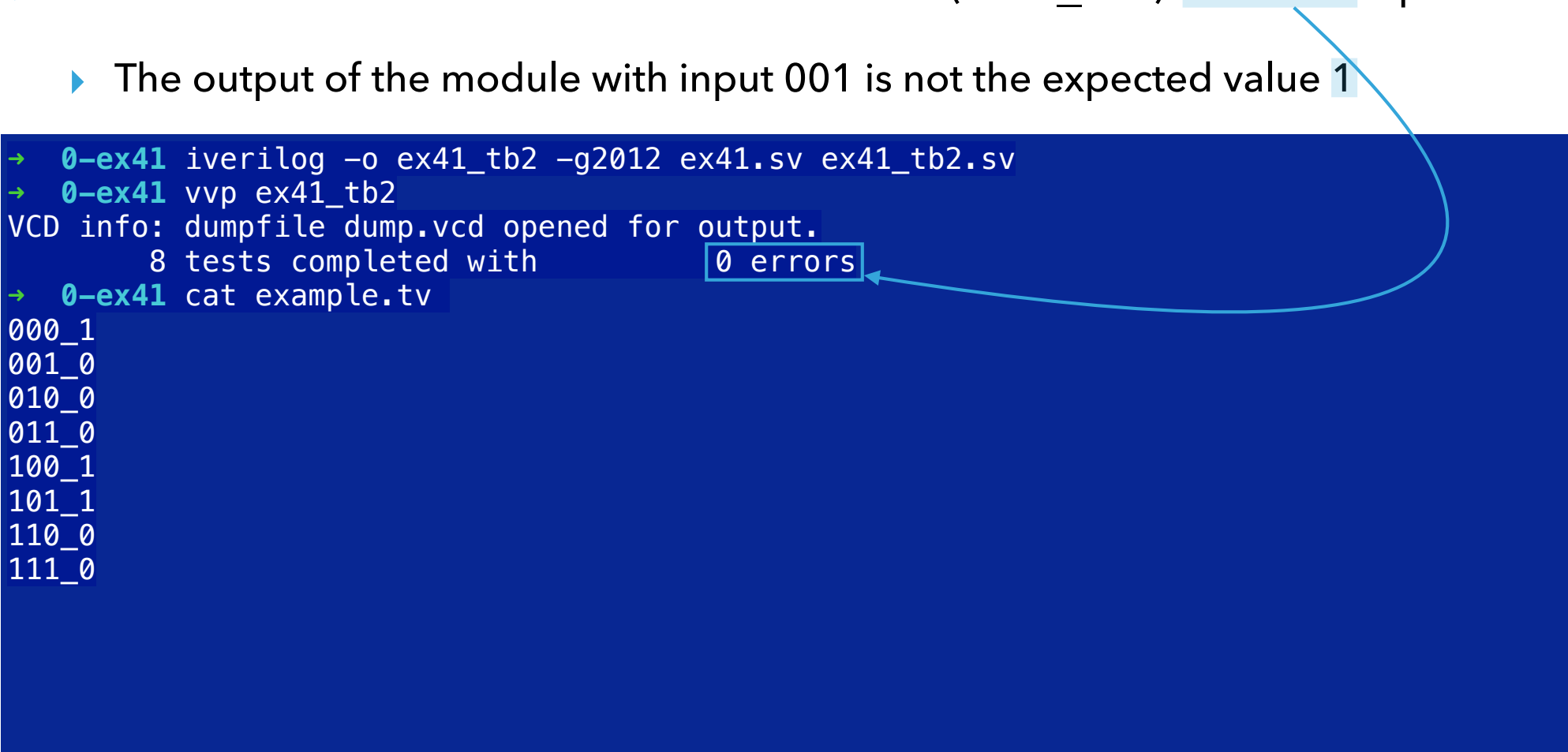
```

**VARIABLE
DUMP
STATEMENTS**

EXAMPLE 4.1 (TB 2)

- ▶ Test of the ex41 module code with test bench 2 (ex41_tb2): no errors reported
 - ▶ The output of the module with input 001 is not the expected value 1

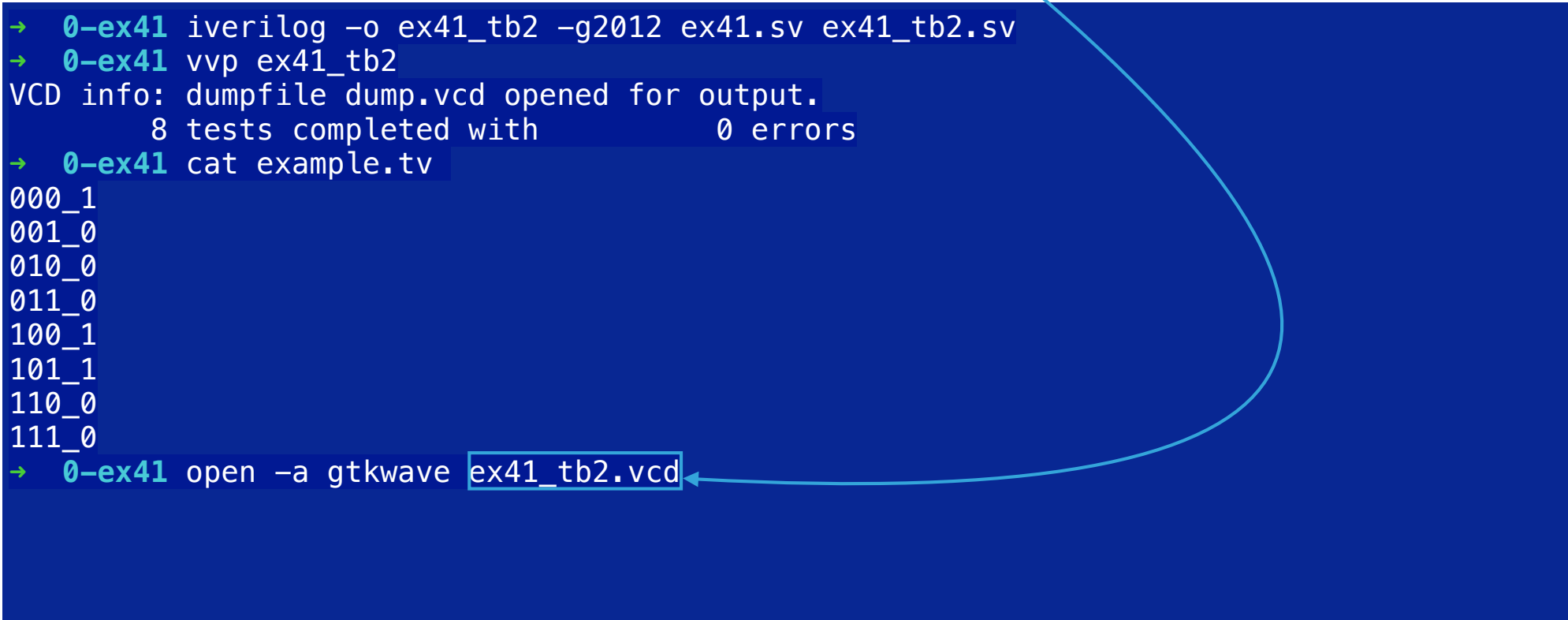
```
→ 0-ex41 iverilog -o ex41_tb2 -g2012 ex41.sv ex41_tb2.sv
→ 0-ex41 vvp ex41_tb2
VCD info: dumpfile dump.vcd opened for output.
      8 tests completed with 0 errors
→ 0-ex41 cat example.tv
000_1
001_0
010_0
011_0
100_1
101_1
110_0
111_0
```



EXAMPLE 4.1 (TB 2)

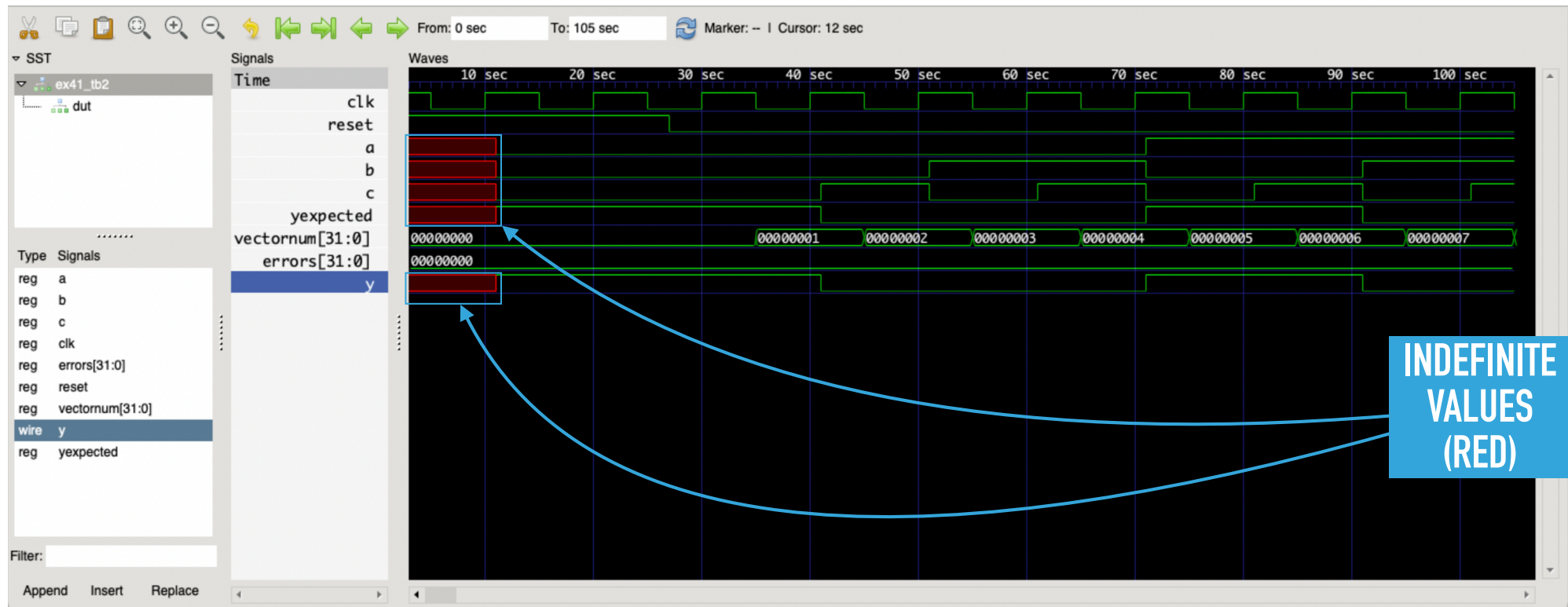
- ▶ Test of the `ex41` module code with test bench 2 (`ex41_tb2`)
 - ▶ The statement `$dumpvars(0, ex41_tb2);` in the test bench specifies that the values of all the variables will be recorded (dumped) into the file specified with `$dumpfile("ex41_tb2.vcd");` then opened with `GTKWave`

```
→ 0-ex41 iverilog -o ex41_tb2 -g2012 ex41.sv ex41_tb2.sv
→ 0-ex41 vvp ex41_tb2
VCD info: dumpfile dump.vcd opened for output.
          8 tests completed with          0 errors
→ 0-ex41 cat example.tv
000_1
001_0
010_0
011_0
100_1
101_1
110_0
111_0
→ 0-ex41 open -a gtkwave ex41_tb2.vcd
```



EXAMPLE 4.1 (TB 2)

- ▶ The dump file can be viewed using GTKWave
 - ▶ The output value is always the expected one ($y=y_{\text{expected}}$)



Visualization through GTKWave of the VCD file generated by test bench 2. Variable values before the first assignment are indefinite so are shown in red

EXAMPLE 4.1 (TB 2 WITH ERROR)

```

module ex41_tb2();
  logic clk, reset;
  logic a, b, c, y, yexpected;
  logic [31:0] vectornum, errors;
  logic [3:0] testvectors[0:7];

  ex41 dut(a, b, c, y);

  always
  begin
    clk=1; #5; clk=0; #5;
  end
  initial
  begin
    $dumpfile("ex41_tb2.vcd");
    $dumpvars(0, ex41_tb2);
    $readmemb("example_error.tv",
testvectors);
    vectornum=0; errors=0;
    reset=1; #27; reset=0;
  end
  always @(posedge clk)
  begin
    #1; {a, b, c, yexpected}
=testvectors[vectornum];
  end

```

```

always @(negedge clk)
  if(~reset)
  begin
    if(y!=yexpected) begin
      $display("Error: inputs=%b",
{a, b, c});
      $display(" outputs=%b (%b
expected)", y, yexpected);
      errors=errors+1;
    end
    vectornum=vectornum+1;
    if(testvectors[vectornum]===4'bx)
  begin
    $display("%d tests completed
with %d errors", vectornum, errors);
    $finish;
  end
  end
endmodule

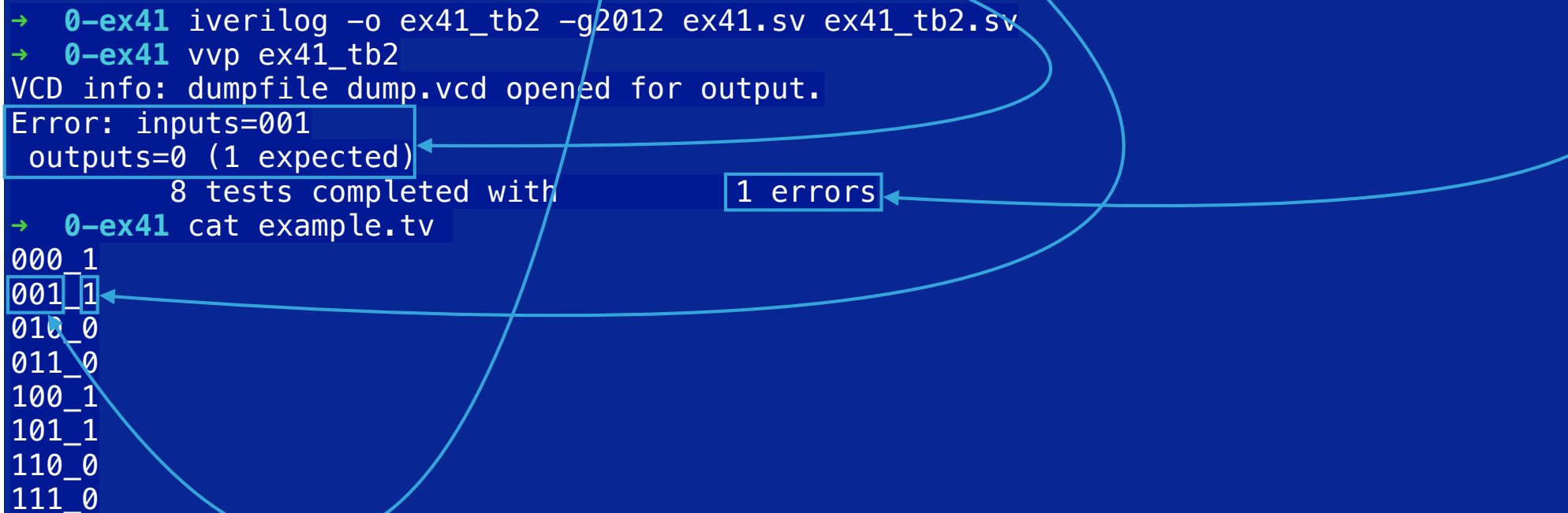
```

VECTOR FILE
WITH ERROR-
INDUCING VALUE

EXAMPLE 4.1 (TB 2 WITH ERROR)

- ▶ Test of the ex41 module code with test bench 2 loading an error-inducing vector (ex41_tb2_error): one error reported
 - ▶ The output of the module with input 001 is not the expected value 1

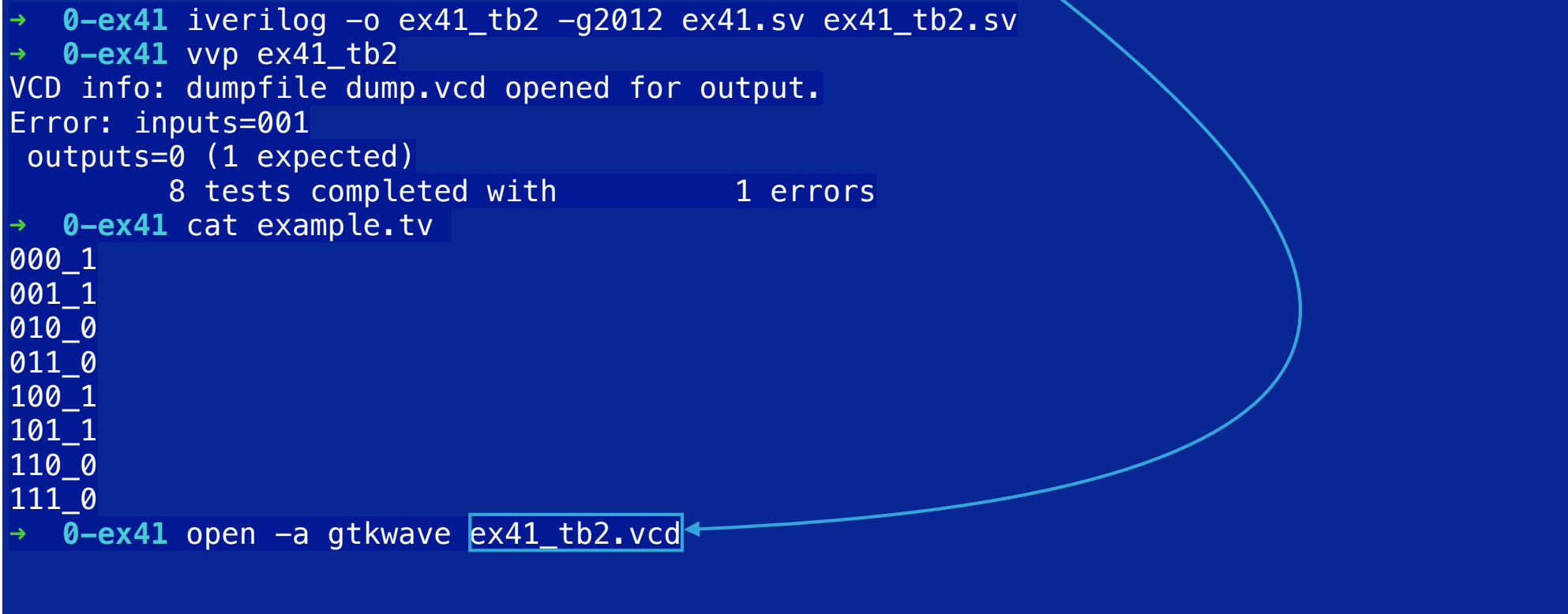
```
→ 0-ex41 iverilog -o ex41_tb2 -g2012 ex41.sv ex41_tb2.sv
→ 0-ex41 vvp ex41_tb2
VCD info: dumpfile dump.vcd opened for output.
Error: inputs=001
      outputs=0 (1 expected)
      8 tests completed with 1 errors
→ 0-ex41 cat example.tv
000_1
001_1
010_0
011_0
100_1
101_1
110_0
111_0
```



EXAMPLE 4.1 (TB 2 WITH ERROR)

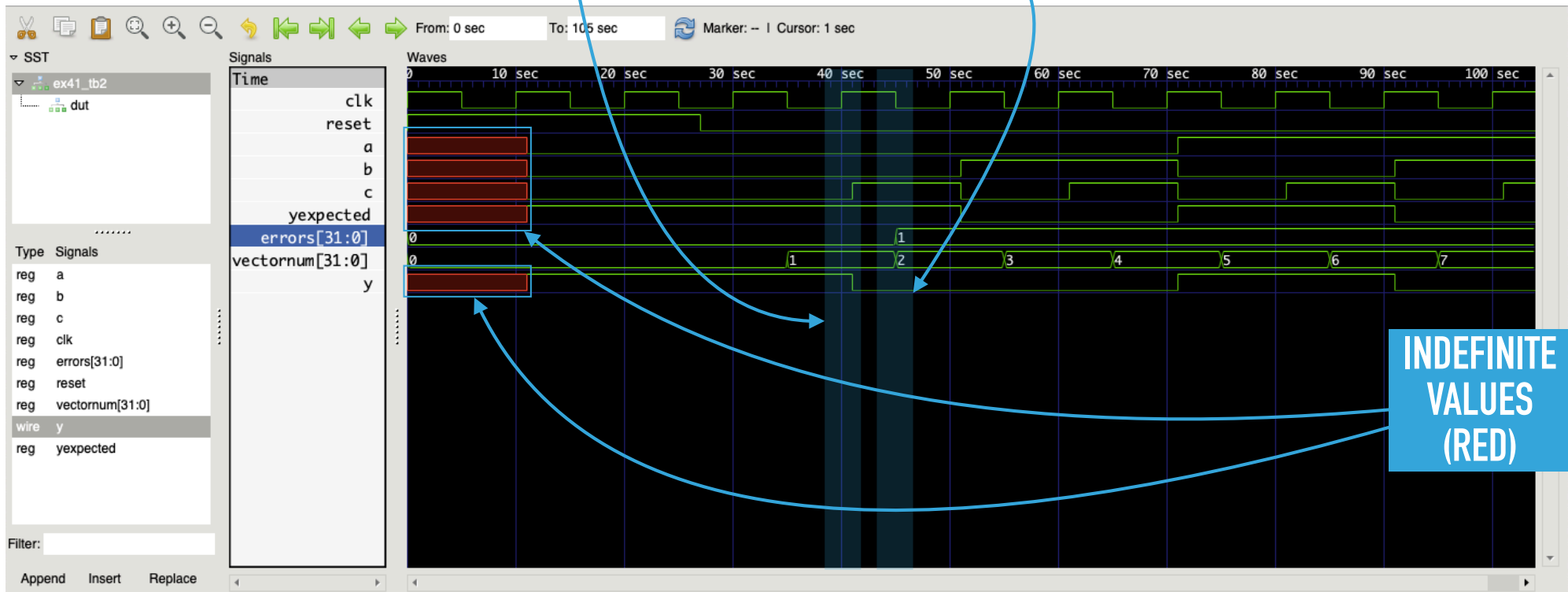
- ▶ Test of the `ex41` module code with test bench 2 (`ex41_tb2`)
 - ▶ The statement `$dumpvars(0, ex41_tb2);` in the test bench specifies that the values of all the variables will be recorded (dumped) into the file specified with `$dumpfile("ex41_tb2.vcd");` then opened with `GTKWave`

```
→ 0-ex41 iverilog -o ex41_tb2 -g2012 ex41.sv ex41_tb2.sv
→ 0-ex41 vvp ex41_tb2
VCD info: dumpfile dump.vcd opened for output.
Error: inputs=001
      outputs=0 (1 expected)
           8 tests completed with           1 errors
→ 0-ex41 cat example.tv
000_1
001_1
010_0
011_0
100_1
101_1
110_0
111_0
→ 0-ex41 open -a gtkwave ex41_tb2.vcd
```



EXAMPLE 4.1 (TB 2 WITH ERROR)

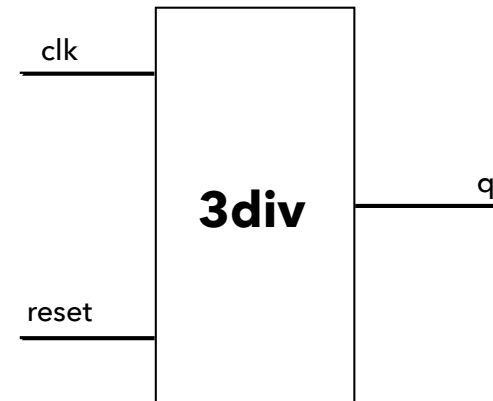
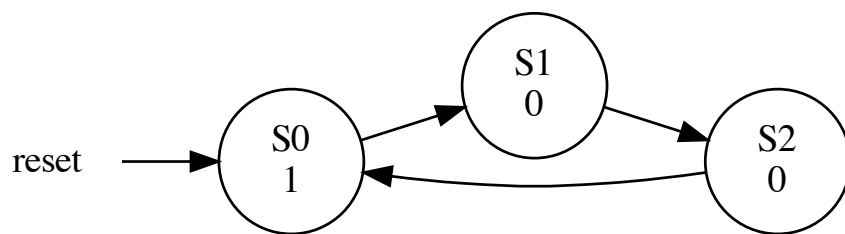
- ▶ The dump file can be viewed using GTKWave
 - ▶ The error can be easily spotted at raising edge #4 of clk as yexpected and y differ
 - ▶ The test bench reacts to the error incrementing the errors variable at falling edge #4 of clk



Visualization through GTKWave of the VCD file generated by test bench 2. Variable values before the first assignment are indefinite so are shown in red

EXAMPLE 4.30

- ▶ The ex430 module implements a divide-by-three counter
 - ▶ The module has no inputs (except clock and reset) and one output
 - ▶ The output divides the frequency of the clock by 3



EXAMPLE 4.30

```

module divideby3FSM (input  logic clk,
                    input  logic reset,
                    output logic q);

    //typedef enum logic [1:0] {S0, S1, S2}
    statetype;
    //statetype [1:0] state, nextstate;
    enum logic [1:0] {S0, S1, S2}
    statetype;
    logic [1:0] state, nextstate;

    // state register
    always_ff @ (posedge clk, posedge
reset)
        if (reset) state <= S0;
        else      state <= nextstate;

    // next state logic
    always_comb
        case (state)
            S0:    nextstate = S1;
            S1:    nextstate = S2;
            S2:    nextstate = S0;
            default: nextstate = S0;
        endcase

```

```

always @(negedge clk)
    if(~reset)
        begin
            if(y!=yexpected) begin
                $display("Error: inputs=%b",
{a, b, c});
                $display(" outputs=%b (%b
expected)", y, yexpected);
                errors=errors+1;
            end
            vectornum=vectornum+1;
            if(testvectors[vectornum]===4'bx)
begin
                $display("%d tests completed
with %d errors", vectornum, errors);
                $finish;
            end
        end
    endmodule

```

DEFINITIONS
UNSUPPORTED BY
ICARUS VERILOG

EXAMPLE 4.30

- ▶ The testbench (ex430_tb) for the ex430 module includes a `timescale` definition
 - ▶ The unit time is set to 10 ns with 1 ns precision
 - ▶ The test runs for $10 \times (1 + 100)$ ns
- ▶ All the variables are dumped to the VCD file `ex430_tb.vcd`
 - ▶ The `$monitor` statement outputs 4-digit time values every time the input changes
 - ▶ The `$finish` statement ends the test

```
`timescale 10ns/1ns

module divideby3FSM_tb();
    logic clk, reset, q;

    divideby3FSM dut(clk, reset, q);

    always begin
        clk=0; #1; clk=1; #1;
    end
    initial begin
        $dumpfile("ex430_tb.vcd");
        $dumpvars(0, divideby3FSM_tb);
        $monitor("[%04t] reset: %b, clk: %b, q: %b", $time, reset, clk, q);
        reset=1;
        #1;
        reset=0;
        #100;
        $finish;
    end
endmodule
```

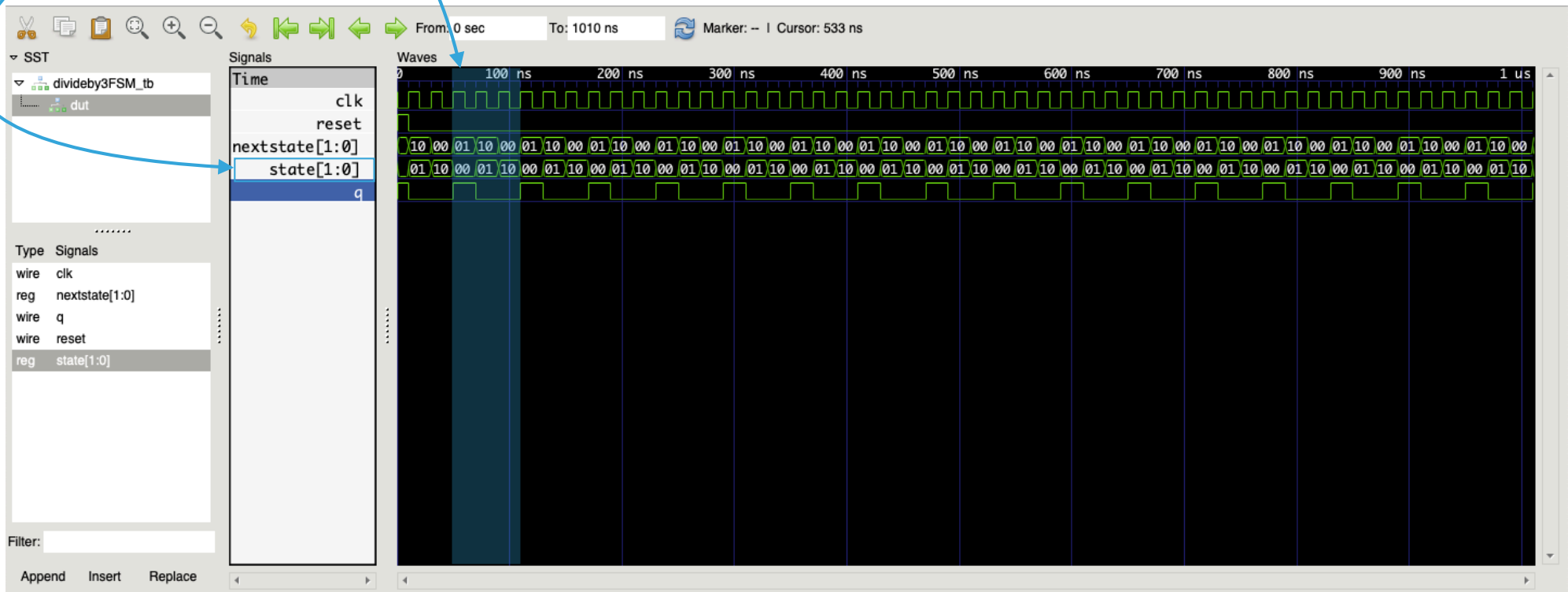
EXAMPLE 4.30

- ▶ Test of the `ex430` module code with a test bench (`ex430_tb`)
 - ▶ The test bench runs for 1010 ns. Clock changes every 10 ns. The time T between two rising edges of `clk` is 20 ns
 - ▶ The output `q` is high for 20 ns then low for 40 ns. The time T_q between two rising edges of `q` is thus 60 ns: $T_q = T/3$

```
→ 1-ex430 iverilog -o ex430_tb -g2012 ex430.sv ex430_tb.sv
→ 1-ex430 vvp ex430_tb
VCD info: dumpfile ex430_tb.vcd opened for output.
[0000] reset: 1, clk: 0, q: 1
[0010] reset: 0, clk: 1, q: 0
[0020] reset: 0, clk: 0, q: 0
[0030] reset: 0, clk: 1, q: 0
[0040] reset: 0, clk: 0, q: 0
[0050] reset: 0, clk: 1, q: 1
[0060] reset: 0, clk: 0, q: 1
[0070] reset: 0, clk: 1, q: 0
[0080] reset: 0, clk: 0, q: 0
[0090] reset: 0, clk: 1, q: 0
[0100] reset: 0, clk: 0, q: 0
.
.
[1010] reset: 0, clk: 1, q: 1
→ 1-ex430 open -a gtkwave ex430_tb.vcd
```

EXAMPLE 4.30

- ▶ The dump file can be viewed using GTKWave
 - ▶ State changes at each raising edge of the clock through values 2'b00, 2'b01, 2'b10
 - ▶ The output q is high for one clock cycle out of every three



Visualization through GTKWave of the VCD file generated by the test bench.

RETI LOGICHE

ESERCITAZIONE 2021-12-10

Q07 NO. IMPLICANTI FUNZIONE BOOLEANA

- ▶ Indicare gli implicant, individuando quelli primi e quelli essenziali, della funzione:

$$f(A, B, C, D) = \sum 1, 2, 3, 4, 6, 7, 9, 12, 14$$

Q07. NO. IMPLICANTI FUNZIONE BOOLEANA

- Q. Indicare gli implicant, individuando quelli primi e quelli essenziali, della funzione:

► $f(A, B, C, D) = \sum 1, 2, 3, 4, 6, 7, 9, 12, 14$

► R.

- Implicant primari: $\bar{A}C, B\bar{D}, \bar{A}\bar{B}D, \bar{B}\bar{C}D$
- Implicant essenziali: $\bar{A}C, B\bar{D}, \bar{B}\bar{C}D$

		<i>CD</i>			
		00	01	11	10
<i>AB</i>	00		1	1	1
	01	1		1	1
	11	1			1
	10		1		

RETI LOGICHE

ESERCITAZIONE 2021-12-23

Q01 NUMERO FLIP-FLOP

- ▶ Scegliendo di utilizzare flip-flop D, quanti di questi elementi sono sufficienti per costruire una rete sequenziale caratterizzata da 4 stati?

Q01 NUMERO FLIP-FLOP

- ▶ Scegliendo di utilizzare flip-flop D, quanti di questi elementi sono sufficienti per costruire una rete sequenziale caratterizzata da 4 stati?
- ▶ R. 2

Q2. MINIMIZZAZIONE ALGEBRICA

- ▶ Q. Si esprima la funzione booleana indicata in basso in forma minima tramite **manipolazioni algebriche**, specificando le identità adoperate. Si confronti inoltre la forma minima con la k-map dell'espressione originale.

- ▶ $F(X, Y, Z) = \bar{X}YZ + \bar{X}Y\bar{Z} + XZ$

Q2. MINIMIZZAZIONE ALGEBRICA

- ▶ Q. Si esprima la funzione booleana indicata in basso in forma minima tramite manipolazioni algebriche, specificando le identità adoperate. Si confronti inoltre la forma minima con la k-map dell'espressione originale.

- ▶ $F(X, Y, Z) = \bar{X}YZ + \bar{X}Y\bar{Z} + XZ$

- ▶ R.

- ▶ $F(X, Y, Z) = \bar{X}YZ + \bar{X}Y\bar{Z} + XZ =$

- ▶ $F(X, Y, Z) = \bar{X}Y + XZ =$ (T. della combinazione)

- ▶ $\bar{X}Y + XZ$

		YZ			
		00	01	11	10
X	0			1	1
	1		1	1	

Q3. MINIMIZZAZIONE CON K-MAP

- ▶ Q. Minimizzare in forma SOP la funzione indicata usando k-map e tenendo conto delle non-specificazioni (indifferenze).

- ▶ $F(A, B, C, D) = \sum (1, 2, 5, 8, 9, 10)$

- ▶ $d(A, B, C, D) = \sum (0, 11, 13)$

Q3. MINIMIZZAZIONE CON K-MAP

- ▶ Q. Minimizzare in forma SOP la funzione indicata usando k-map e tenendo conto delle non-specificazioni (indifferenze).

- ▶ $F(A, B, C, D) = \sum (1, 2, 5, 8, 9, 10)$

- ▶ $d(A, B, C, D) = \sum (0, 11, 13)$

- ▶ R. $F(A, B, C, D) = \overline{B}\overline{D} + \overline{C}D$

		<i>CD</i>			
		00	01	11	10
<i>AB</i>	00	X	1		1
	01		1		
	11		X		
	10	1	1	X	1

K-map con risoluzione delle indifferenze

		<i>CD</i>			
		00	01	11	10
<i>AB</i>	00	1	1		1
	01		1		
	11		1		
	10	1	1		1

Q4. TEMPORIZZAZIONI

- ▶ Q. Lo sfasamento di clock
 - a. aumenta il tempo a disposizione dei circuiti combinatori per produrre segnali stabili tra un fronte di clock e il successivo
 - b. riduce il tempo a disposizione dei circuiti combinatori per produrre segnali stabili tra un fronte di clock e il successivo
 - c. riduce il tempo di hold.

Q4. TEMPORIZZAZIONI

- ▶ Q. Lo sfasamento di clock
 - a. aumenta il tempo a disposizione dei circuiti combinatori per produrre segnali stabili tra un fronte di clock e il successivo
 - b. riduce il tempo a disposizione dei circuiti combinatori per produrre segnali stabili tra un fronte di clock e il successivo**
 - c. riduce il tempo di hold

Q5. MTBF

- ▶ Q. Inserire il tempo medio tra errori (**MTBF**), in **secondi**, arrotondato alle unità, per un sincronizzatore caratterizzato, oltre che dai parametri [1], dalla frequenza di lavoro di 0,5 GHz, considerando che i segnali in ingresso ai flip-flop devono essere stabili almeno 100 ps prima del fronte di sincronizzazione e che il segnale asincrono di ingresso cambia in media una volta ogni 10 secondi.
[1] $\tau = 200 \text{ ps}$, $T_0 = 150 \text{ ps}$, $t_{hold} = 5 \text{ ps}$, $t_{ccq} = 10 \text{ ps}$

- ▶ R.

Q5. MTBF

- Q. Inserire il tempo medio tra errori (**MTBF**), in **secondi**, arrotondato alle unità, per un sincronizzatore caratterizzato, oltre che dai parametri [1], dalla frequenza di lavoro di 0,5 GHz, considerando che i segnali in ingresso ai flip-flop devono essere stabili almeno 100 ps prima del fronte di sincronizzazione e che il segnale asincrono di ingresso cambia in media una volta ogni 10 secondi.

$$[1] \quad \tau = 200 \text{ ps}, T_0 = 150 \text{ ps}, t_{hold} = 5 \text{ ps}, t_{ccq} = 10 \text{ ps}$$

- R.

$$T_c = \frac{1}{0,5 \text{ GHz}} = \frac{1}{\frac{1}{2} \cdot 10^9 \text{ s}^{-1}} = 2 \cdot 10^{-9} \text{ s} = 2 \text{ ns} = 2000 \text{ ps}$$

$$MTBF = \frac{T_c}{NT_0} e^{\frac{T_c - t_{setup}}{\tau}} = \frac{2000 \text{ ps}}{0,1 \text{ s}^{-1} \cdot 150 \text{ ps}} e^{\frac{2000 \text{ ps} - 100 \text{ ps}}{200 \text{ ps}}} = 133,3 \cdot e^{\frac{1900 \text{ ps}}{200 \text{ ps}}} \text{ s} =$$

$$133,3 \cdot e^{9,5} \text{ s} \approx 1781297$$

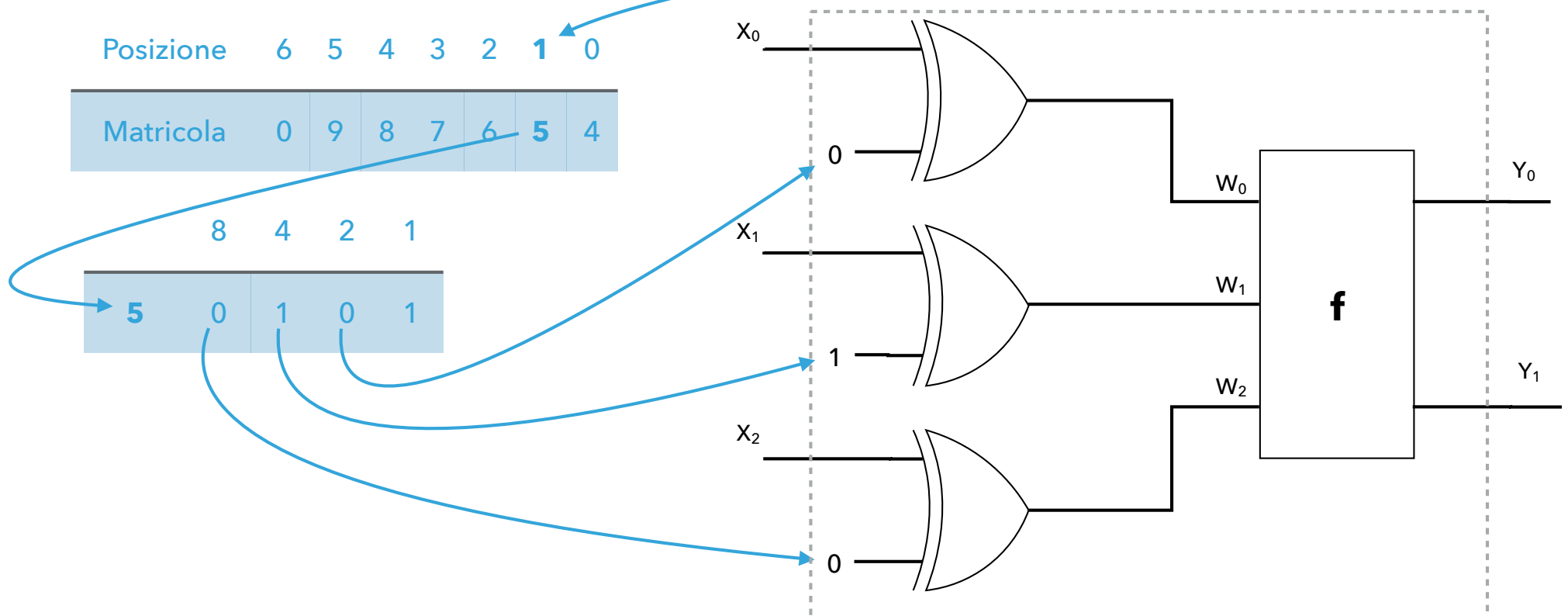
Q6. SINTESI RETE COMBINATORIA

- Fornire la descrizione algebrica minima e la relativa rappresentazione circuitale di una RL la cui uscita Y riporti il numero di bit a 0 del valore ottenuto operando lo XOR tra il suo ingresso X a 3 bit e i tre bit più significativi della rappresentazione binaria della cifra in **posizione 1** del proprio numero di **matricola**.



Q6. SINTESI RETE COMBINATORIA

- Fornire la descrizione algebrica minima e la relativa rappresentazione circuitale di una RL la cui uscita Y riporti il numero di bit a 0 del valore ottenuto operando lo XOR tra il suo ingresso X a 3 bit e i tre bit più significativi della rappresentazione binaria della cifra in **posizione 1** del proprio numero di **matricola**.



La rete combinatoria è decomposta in un modulo f che calcola il numero dei bit a zero nei suoi ingressi W e in un livello di porte XOR

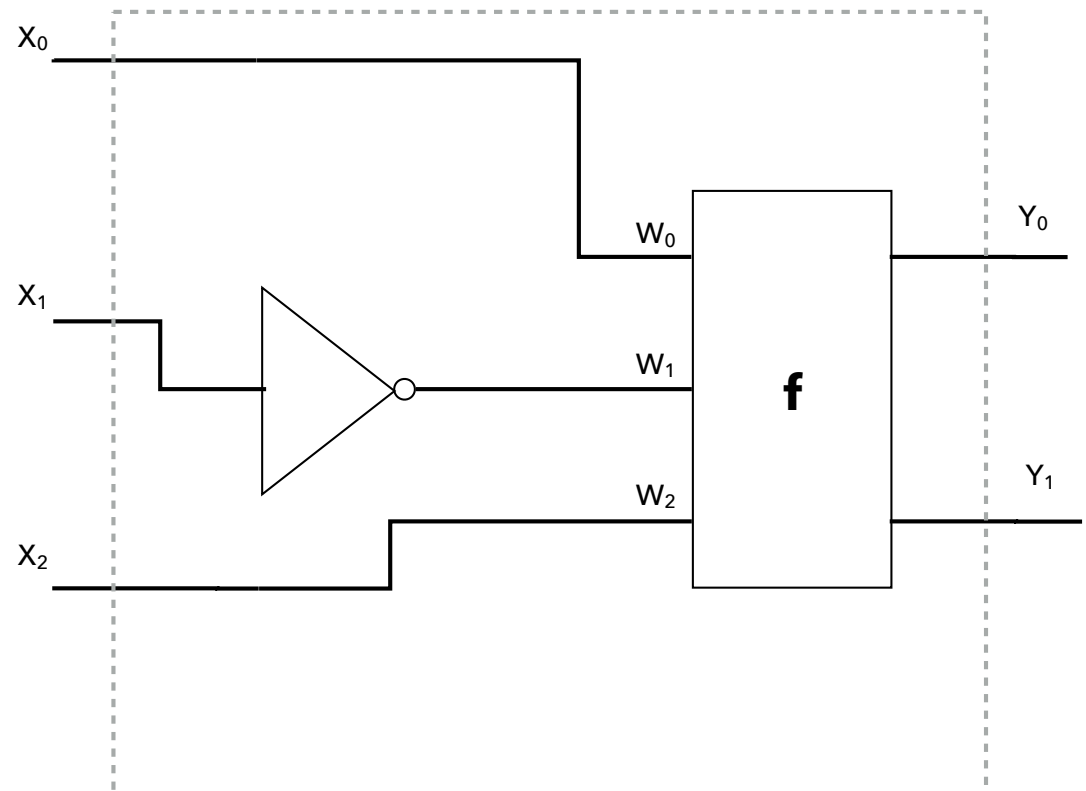
Q6. SINTESI RETE COMBINATORIA

- Fornire la descrizione algebrica minima e la relativa rappresentazione circuitale di una RL la cui uscita Y riporti il numero di bit a 0 del valore ottenuto operando lo XOR tra il suo ingresso X a 3 bit e i tre bit più significativi della rappresentazione binaria della cifra in posizione 1 del proprio numero di matricola.

$$W_0 = X_0 \oplus 0 = X_0$$

$$W_1 = X_1 \oplus 1 = \overline{X_1}$$

$$W_2 = X_2 \oplus 0 = X_2$$



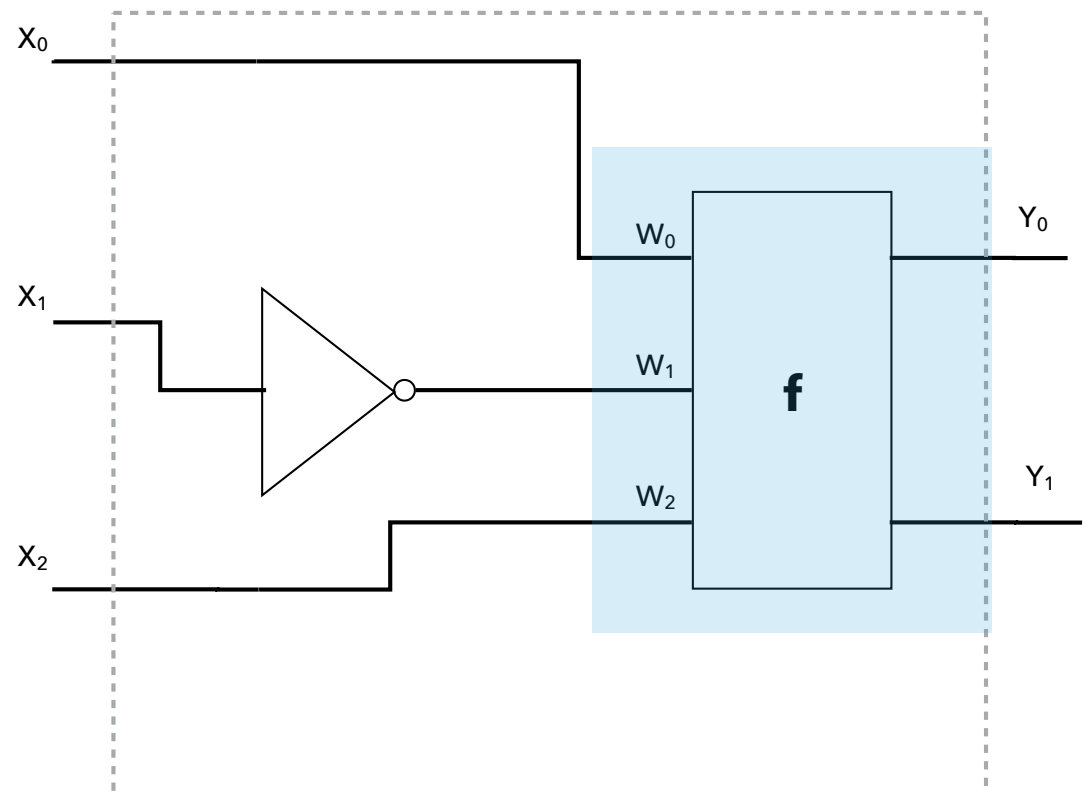
Le porte XOR a due ingressi, di cui uno costante, possono essere trasformate in NOT o eliminate

Q6. SINTESI RETE COMBINATORIA

- Fornire la descrizione algebrica minima e la relativa rappresentazione circuitale di una RL la cui uscita Y riporti il numero di bit a 0 del valore ottenuto operando lo XOR tra il suo ingresso X a 3 bit e i tre bit più significativi della rappresentazione binaria della cifra in posizione 1 del proprio numero di matricola.

W_2	W_1	W_0	Y_1	Y_0
0	0	0	1	1
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	0	0

Tabella di verità di f



Il blocco f può essere sintetizzato con decoder, multiplexer o utilizzando espressioni semplificate con manipolazioni o k-map

Q6. SINTESI RETE COMBINATORIA

- Fornire la descrizione algebrica minima e la relativa rappresentazione circuitale di una RL la cui uscita Y riporti il numero di bit a 0 del valore ottenuto operando lo XOR tra il suo ingresso X a 3 bit e i tre bit più significativi della rappresentazione binaria della cifra in posizione 1 del proprio numero di matricola.

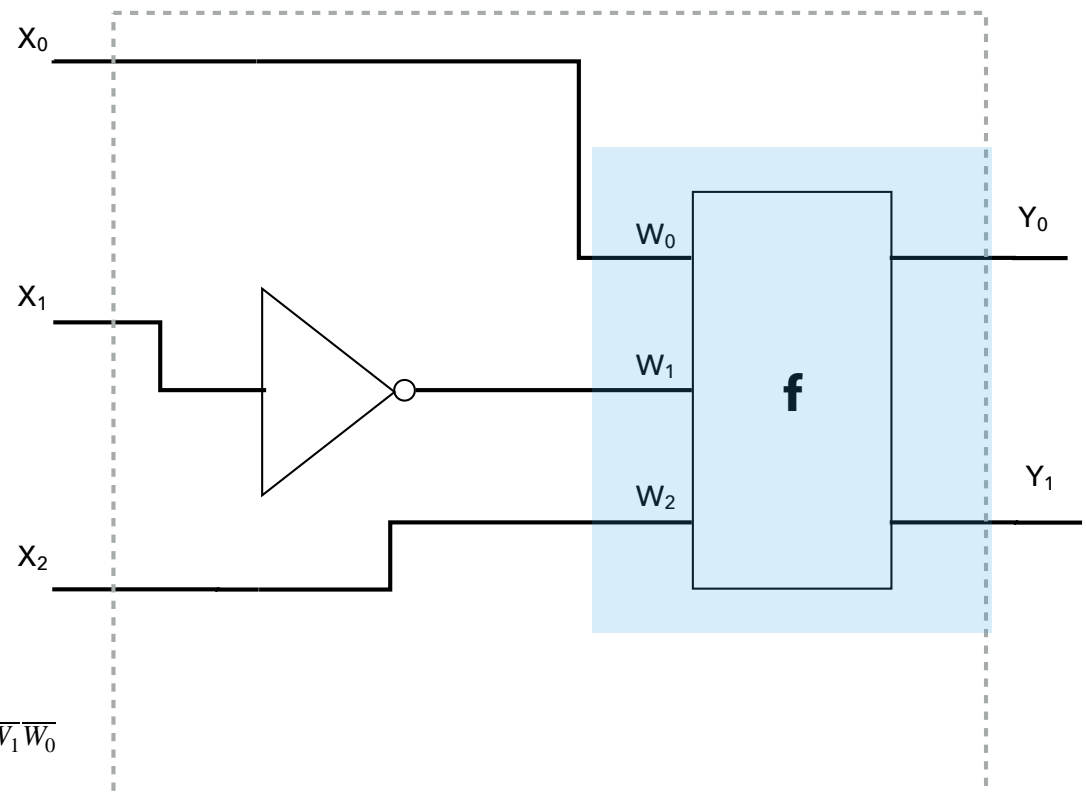
		$W_1 W_0$			
		00	01	11	10
W_2	0	1		1	
	1		1		1

$$Y_0 = \overline{W_2} \overline{W_1} \overline{W_0} + \overline{W_2} W_1 W_0 + W_2 \overline{W_1} W_0 + W_2 W_1 \overline{W_0} = \overline{W_2} \oplus W_1 \oplus W_0$$

		$W_1 W_0$			
		00	01	11	10
W_2	0	1	1		1
	1	1			

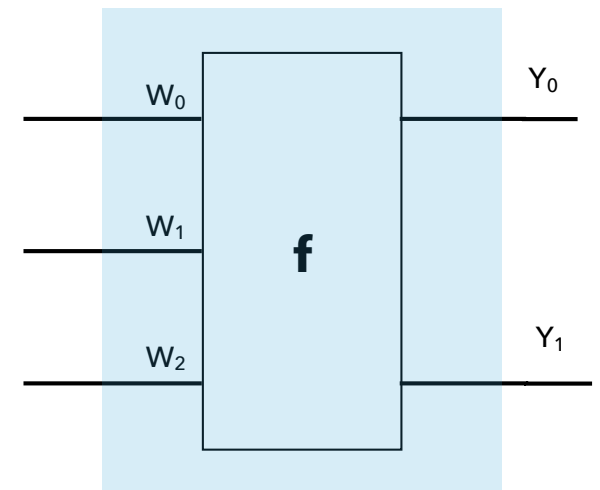
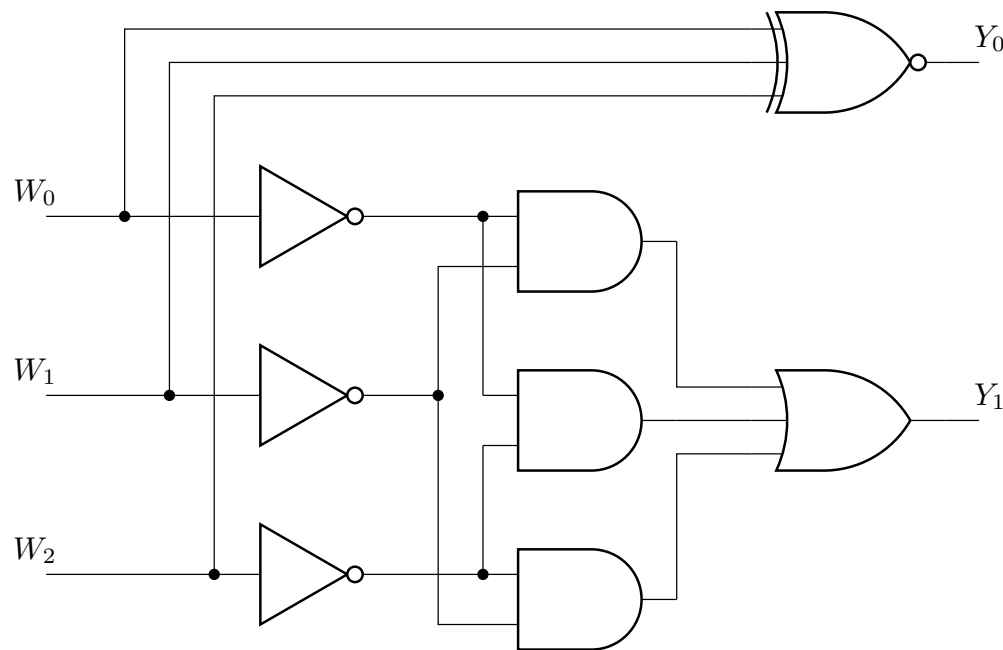
$$Y_1 = \overline{W_2} \overline{W_1} \overline{W_0} + \overline{W_2} \overline{W_1} X_0 + \overline{W_2} W_1 \overline{W_0} + W_2 \overline{W_1} \overline{W_0} = \overline{W_2} \overline{W_1} + \overline{W_2} \overline{W_0} + \overline{W_1} \overline{W_0}$$

Semplificazione con k-map delle funzioni che definiscono il blocco f



Q6. SINTESI RETE COMBINATORIA

- Fornire la descrizione algebrica minima e la relativa rappresentazione circuitale di una RL la cui uscita Y riporti il numero di bit a 0 del valore ottenuto operando lo XOR tra il suo ingresso X a 3 bit e i tre bit più significativi della rappresentazione binaria della cifra in posizione 1 del proprio numero di matricola.



Q6. SINTESI RETE COMBINATORIA

- Fornire la descrizione algebrica minima e la relativa rappresentazione circuitale di una RL la cui uscita Y riporti il numero di bit a 0 del valore ottenuto operando lo XOR tra il suo ingresso X a 3 bit e i tre bit più significativi della rappresentazione binaria della cifra in posizione 1 del proprio numero di matricola.

$$Y_0 = \overline{W_2 \oplus W_1 \oplus W_0}$$

$$Y_1 = \overline{W_2 W_1} + \overline{W_2 W_0} + \overline{W_1 W_0}$$

$$W_0 = X_0 \oplus 0 = X_0$$

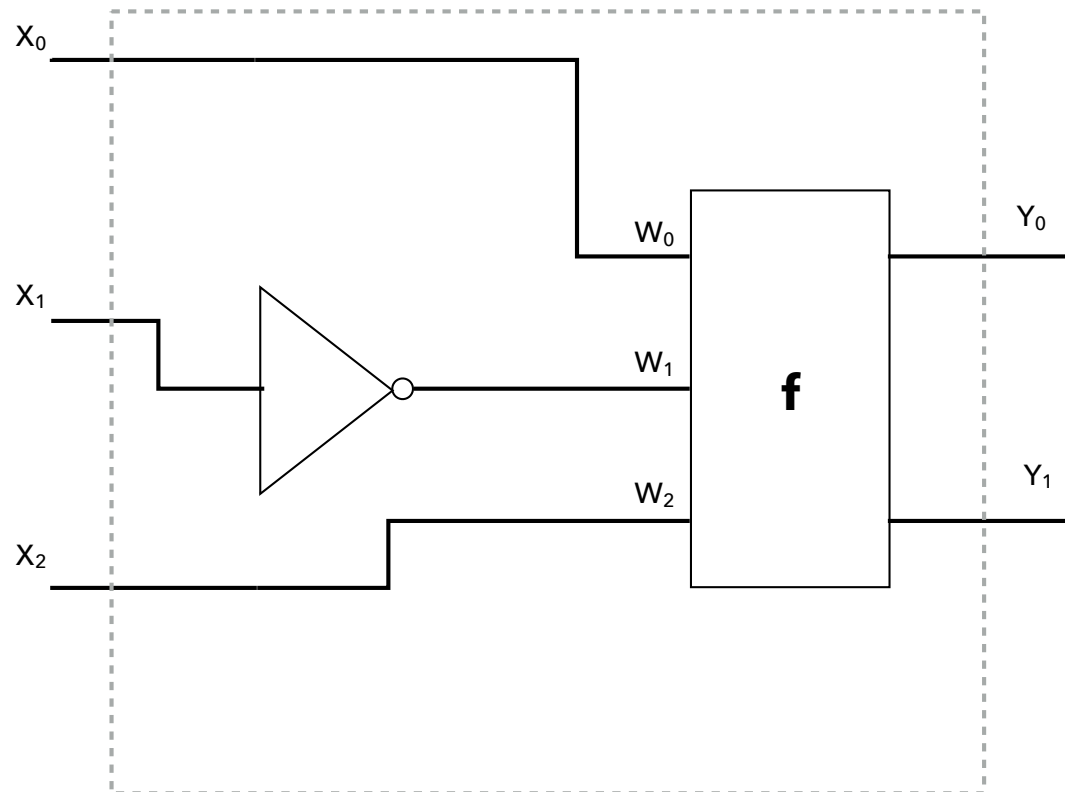
$$W_1 = X_1 \oplus 1 = \overline{X_1}$$

$$W_2 = X_2 \oplus 0 = X_2$$



$$Y_0 = \overline{X_2 \oplus \overline{X_1} \oplus X_0}$$

$$Y_1 = \overline{X_2} X_1 + \overline{X_2} \overline{X_0} + X_1 \overline{X_0}$$

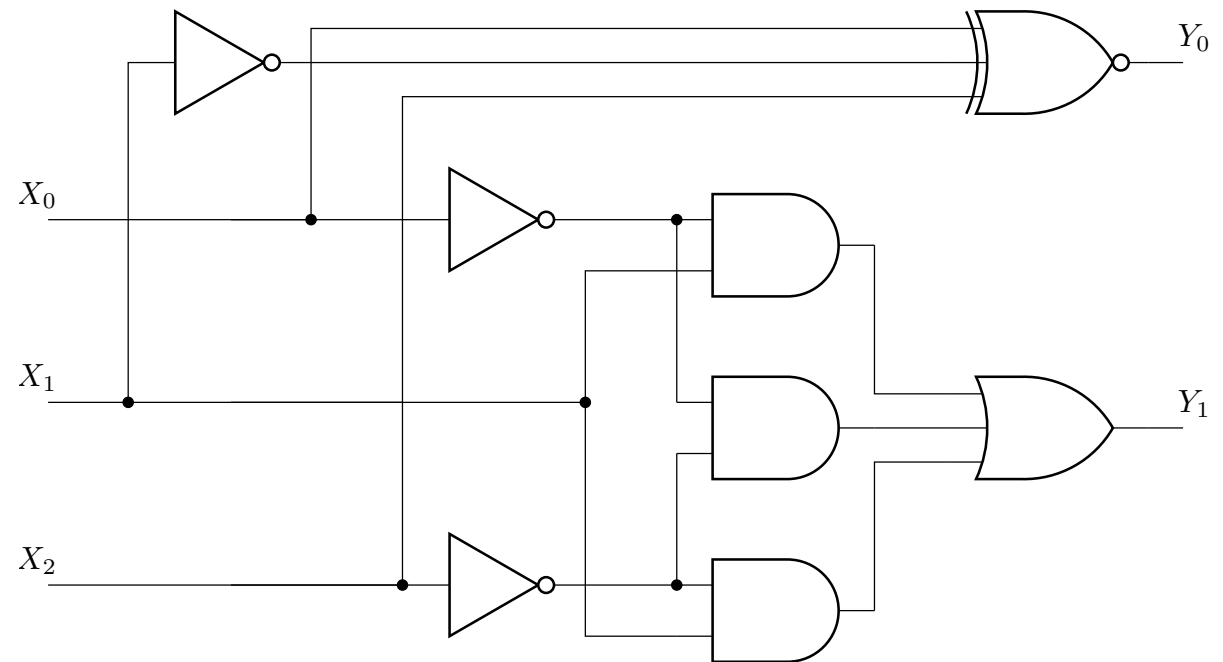
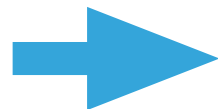


Q6. SINTESI RETE COMBINATORIA

- Fornire la descrizione algebrica minima e la relativa rappresentazione circuitale di una RL la cui uscita Y riporti il numero di bit a 0 del valore ottenuto operando lo XOR tra il suo ingresso X a 3 bit e i tre bit più significativi della rappresentazione binaria della cifra in posizione 1 del proprio numero di matricola.

$$Y_0 = \overline{X_2 \oplus \overline{X_1} \oplus X_0}$$

$$Y_1 = \overline{X_2}X_1 + \overline{X_2}\overline{X_0} + X_1\overline{X_0}$$



Q6. SINTESI RETE COMBINATORIA

- ▶ Implementazione in SystemVerilog
(non richiesta)
- ▶ Modulo

```
module q6(input logic [2:0] x, output
logic [1:0] y);

/* studentid: 0123456
 * 0123456 5=4'b0101, 3 msb: 3'b010
 *      ^
 */

logic [2:0] studentidmask= 3'b010;
logic [2:0] w;

assign w=x^studentidmask;
/*
assign w[2]=x[2]^studentidmask[2];
assign w[1]=x[1]^studentidmask[1];
assign w[0]=x[0]^studentidmask[0];
*/
assign y[0]=~(^w);
assign y[1]=~w[2]&~w[1] | ~w[2]&~w[0]
| ~w[1]&~w[0];
endmodule
```

Q6. SINTESI RETE COMBINATORIA

- Implementazione in SystemVerilog (non richiesta)
 - Test bench

```
module q6_tb();  
    logic [2:0] x;  
    logic [1:0] y;  
  
    q6 dut(x, y);  
  
    initial begin  
        $display("x2 x1 x0 | y1 y0");  
        $display("-----");  
        $monitor(" %b %b %b | %b %b",  
x[2], x[1], x[0], y[1], y[0]);  
        x=3'b000; #1;  
        x=3'b001; #1;  
        x=3'b010; #1;  
        x=3'b011; #1;  
        x=3'b100; #1;  
        x=3'b101; #1;  
        x=3'b110; #1;  
        x=3'b111; #1;  
        $finish;  
    end  
endmodule
```


Q6. SINTESI RETE COMBINATORIA

- ▶ Implementazione in SystemVerilog
(non richiesta)
- ▶ Output del test bench

```

→ 06 iverilog -g2012 -o q6.vvp q6.sv
q6_tb.sv
→ 06 vvp q6.vvp
x2 x1 x0 | y1 y0
-----
0  0  0 | 1  0
0  0  1 | 0  1
0  1  0 | 1  1
0  1  1 | 1  0
1  0  0 | 0  1
1  0  1 | 0  0
1  1  0 | 1  0
1  1  1 | 0  1
→ 06

```

Q7. SINTESI RETE SEQUENZIALE

- ▶ Progettare una rete sequenziale con ingresso X a 1 bit la cui uscita Y assuma il valore 1 finché non riconosca negli ultimi tre valori presentati all'ingresso un numero pari di bit con valore 1. Dopo il riconoscimento, a meno che il bit successivo in ingresso determini un ulteriore riconoscimento, l'uscita della rete deve tornare al valore 1. Si producano il **diagramma degli stati**, la tabella di transizione e il diagramma circuitale.

Q7. SINTESI RETE SEQUENZIALE

- ▶ Progettare una rete sequenziale con ingresso X a 1 bit la cui uscita Y assuma il valore 1 finché non riconosca negli ultimi tre valori presentati all'ingresso un numero pari di bit con valore 1. Dopo il riconoscimento, a meno che il bit successivo in ingresso determini un ulteriore riconoscimento, l'uscita della rete deve tornare al valore 1. Si producano il **diagramma degli stati**, la tabella di transizione e il diagramma circuitale.
- ▶ In questo caso la codifica binaria degli stati permette di semplificare la progettazione

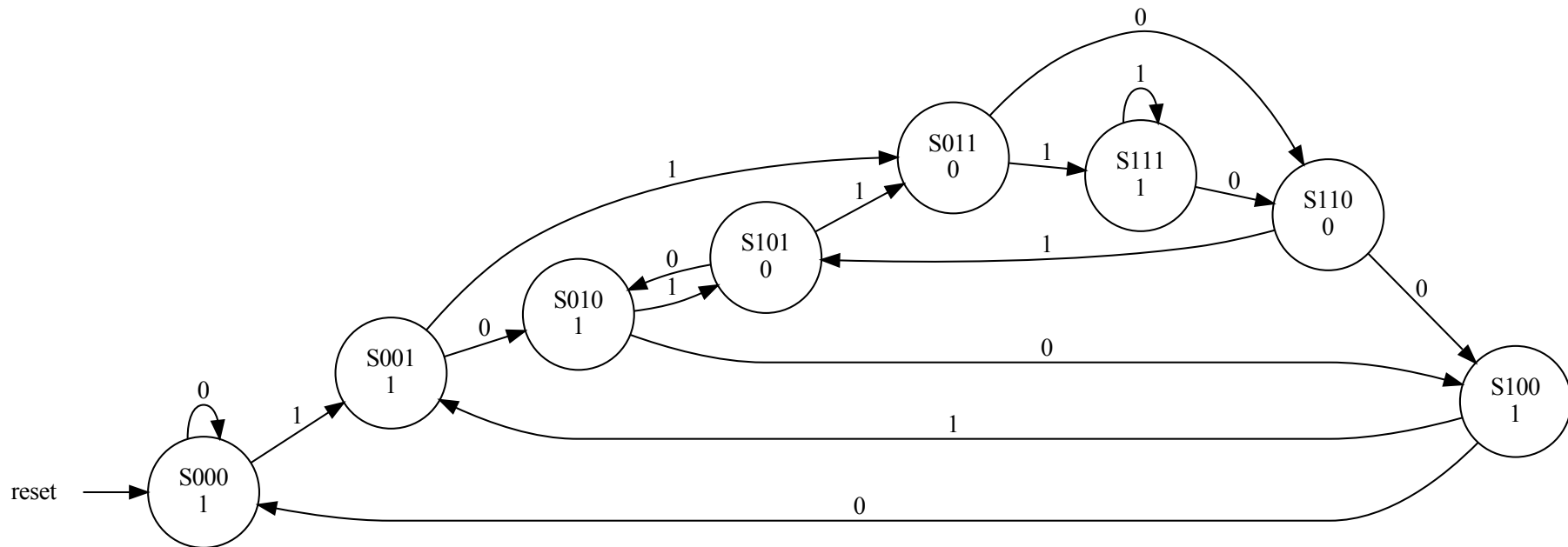


Diagramma degli stati (Moore)

Q7. SINTESI RETE SEQUENZIALE

- ▶ Progettare una rete sequenziale con ingresso X a 1 bit la cui uscita Y assuma il valore 1 finché non riconosca negli ultimi tre valori presentati all'ingresso un numero pari di bit con valore 1. Dopo il riconoscimento, a meno che il bit successivo in ingresso determini un ulteriore riconoscimento, l'uscita della rete deve tornare al valore 1. Si producano il **diagramma degli stati**, la **tabella di transizione** e il diagramma circuitale.
- ▶ In questo caso la codifica binaria degli stati permette di semplificare la progettazione

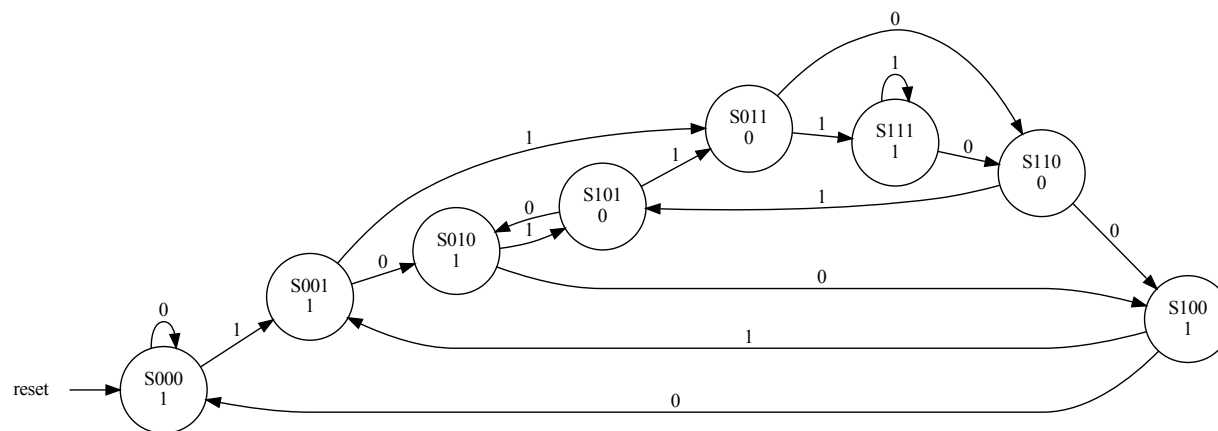


Diagramma degli stati (Moore)

Current State: S	Input: X	Next State: S'
S000	0	S000
S000	1	S001
S001	0	S010
S001	1	S011
S010	0	S100
S010	1	S101
S011	0	S110
S011	1	S111
S100	0	S000
S100	1	S001
S101	0	S010
S101	1	S011
S110	0	S100
S110	1	S101
S111	0	S110
S111	1	S111

Tabella di transizione

Q7. SINTESI RETE SEQUENZIALE

- ▶ Progettare una rete sequenziale con ingresso X a 1 bit la cui uscita Y assuma il valore 1 finché non riconosca negli ultimi tre valori presentati all'ingresso un numero pari di bit con valore 1. Dopo il riconoscimento, a meno che il bit successivo in ingresso determini un ulteriore riconoscimento, l'uscita della rete deve tornare al valore 1. Si producano il **diagramma degli stati**, la **tabella di transizione** e il diagramma circuitale.
- ▶ In questo caso usare direttamente codifica binaria nelle etichette degli stati, che evidenzia la sequenza degli ultimi tre bit in ingresso, permette di semplificare la progettazione
- ▶ Analizzando la tabella di transizione si nota che la logica combinatoria di stato prossimo deve manipolare i segnali di stato e di ingresso in maniera tale che il registro contenga i valori degli ultimi tre bit presentati all'ingresso in ordine cronologico. A ogni impulso di clock i due bit meno significativi del registro scorrono verso sinistra, sovrascrivendo il valore del bit più significativo e lasciando posto al valore del bit di ingresso (left shift).

$$S'_2 = S_1$$

$$S'_1 = S_0$$

$$S'_0 = X$$

Current State: S	Input: X	Next State: S'
S000	0	S000
S000	1	S001
S001	0	S010
S001	1	S011
S010	0	S100
S010	1	S101
S011	0	S110
S011	1	S111
S100	0	S000
S100	1	S001
S101	0	S010
S101	1	S011
S110	0	S100
S110	1	S101
S111	0	S110
S111	1	S111

Tabella di transizione

Q7. SINTESI RETE SEQUENZIALE

- ▶ Progettare una rete sequenziale con ingresso X a 1 bit la cui uscita Y assuma il valore 1 finché non riconosca negli ultimi tre valori presentati all'ingresso un numero pari di bit con valore 1. Dopo il riconoscimento, a meno che il bit successivo in ingresso determini un ulteriore riconoscimento, l'uscita della rete deve tornare al valore 1. Si producano il **diagramma degli stati**, la **tabella di transizione** e il diagramma circuitale.
- ▶ La codifica binaria nelle etichette degli stati permette di semplificare anche la progettazione della rete di uscita, che esprime la somma della **parità** del valore a tre bit memorizzato (XOR) e del mintermine 0.

$$S'_2 = S_1$$

$$S'_1 = S_0$$

$$S'_0 = X$$

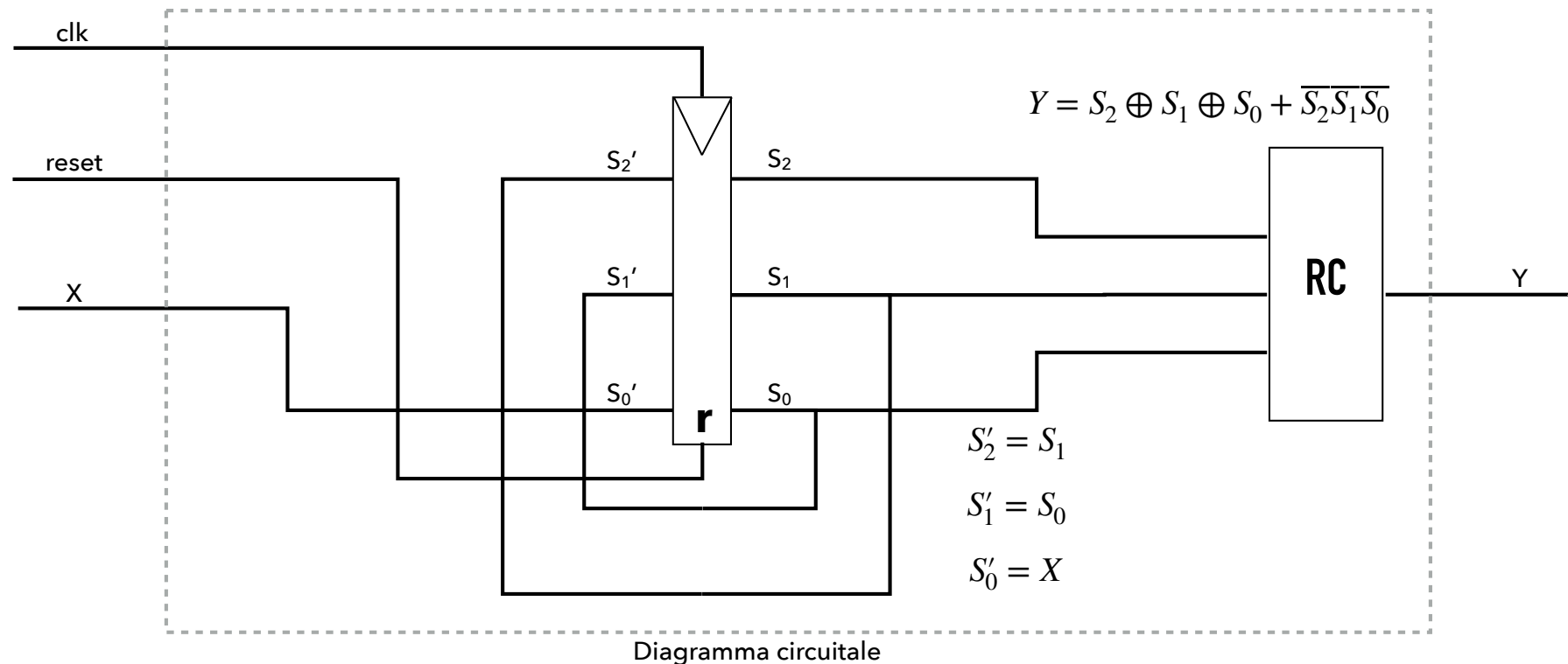
$$Y = S_2 \oplus S_1 \oplus S_0 + \overline{S_2} \overline{S_1} \overline{S_0}$$

Current State: S	Output: Y
S000	1
S001	1
S010	1
S011	0
S100	1
S101	0
S110	0
S111	1

Tabella delle uscite

Q7. SINTESI RETE SEQUENZIALE

- Progettare una rete sequenziale con ingresso X a 1 bit la cui uscita Y assuma il valore 1 finché non riconosca negli ultimi tre valori presentati all'ingresso un numero pari di bit con valore 1. Dopo il riconoscimento, a meno che il bit successivo in ingresso determini un ulteriore riconoscimento, l'uscita della rete deve tornare al valore 1. Si producano il diagramma degli stati, la tabella di transizione e il **diagramma circuitale**



RETI LOGICHE

ESERCITAZIONE 2022-01-10

Q1. HDL RETE COMBINATORIA

- ▶ Indicare il codice di una possibile implementazione in SystemVerilog del **test bench** che mostri la tabella di verità di una rete combinatoria (usare rc come identificatore) con tre ingressi (a, b, c) e una uscita (y)

```
module
```

```
endmodule
```

Q1. HDL RETE COMBINATORIA

- ▶ Indicare il codice di una possibile implementazione in SystemVerilog del **test bench** che mostri la tabella di verità di una rete combinatoria (usare rc come identificatore) con tre ingressi (a, b, c) e una uscita (y)
- ▶ Nell'implementazione con il costrutto **for** è necessario usare un numero di bit sufficienti (4) per contare fino al valore limite (8) in modo che la condizione sia verificabile e il ciclo non continui indefinitamente

```

module q1_tb();

    logic a,b,c,y;
    logic [3:0] counter;
    rc dut(a,b,c,y);

    initial begin
        $display("a b c | y");
        $display("-----");
        $monitor("%b %b %b | %b",
a,b,c,y);
        /*      a=0; b=0; c=0; #1;
                  c=1; #1;
                  b=1; c=0; #1;
                  c=1; #1;
                  a=1; b=0; c=0; #1;
                  c=1; #1;
                  b=1; c=0; #1;
                  c=1; #1; */

        counter=0;
        for(counter=0; counter<8;
counter=counter+1) begin
            /* a=counter[2];
b=counter[1]; c=counter[0]; #1; */
            {a,b,c}=counter[2:0]; #1;
        end
    end
endmodule

```

Q2. HDL RETE SEQUENZIALE

- ▶ Indicare il **codice di una possibile implementazione in SystemVerilog** di una rete sequenziale con ingresso X a 1 bit la cui uscita Y assuma il valore 1 finché non riconosca negli ultimi tre valori presentati all'ingresso un numero pari di bit con valore 1. Dopo il riconoscimento, a meno che il bit successivo in ingresso termini un ulteriore riconoscimento, l'uscita della rete deve tornare al valore 1 (Esercitazione 2021-12-23 - Q7)
- ▶ Includere un test bench per descrivere in maniera completa il funzionamento della rete

```
module
```

```
endmodule
```

Q2. HDL RETE SEQUENZIALE

- ▶ Indicare il **codice di una possibile implementazione in SystemVerilog** di una rete sequenziale con ingresso X a 1 bit la cui uscita Y assuma il valore 1 finché non riconosca negli ultimi tre valori presentati all'ingresso un numero pari di bit con valore 1. Dopo il riconoscimento, a meno che il bit successivo in ingresso termini un ulteriore riconoscimento, l'uscita della rete deve tornare al valore 1 (Esercitazione 2021-12-23 - Q7)
- ▶ Includere un test bench per descrivere in maniera completa il funzionamento della rete

```
module q7(input logic clk, input logic
reset, input logic x, output logic y);

    logic [2:0] state, nextstate;

    always_ff @(posedge clk, posedge
reset) begin
        if (reset) begin
            state <= 3'b000;
        end
        else state <= nextstate;
    end

    assign nextstate= {state[1:0], x};
    assign y=(^state)|(state==3'b000);

endmodule
```

Q2. HDL RETE SEQUENZIALE

- ▶ Indicare il codice di una possibile implementazione in SystemVerilog di una rete sequenziale con ingresso X a 1 bit la cui uscita Y assuma il valore 1 finché non riconosca negli ultimi tre valori presentati all'ingresso un numero pari di bit con valore 1. Dopo il riconoscimento, a meno che il bit successivo in ingresso termini un ulteriore riconoscimento, l'uscita della rete deve tornare al valore 1 (Esercitazione 2021-12-23 - Q7)
- ▶ Includere un **test bench** per descrivere in maniera completa il funzionamento della rete

```
`timescale 1ns/1ns

module q7_tb();
    logic clk, reset, x, y;

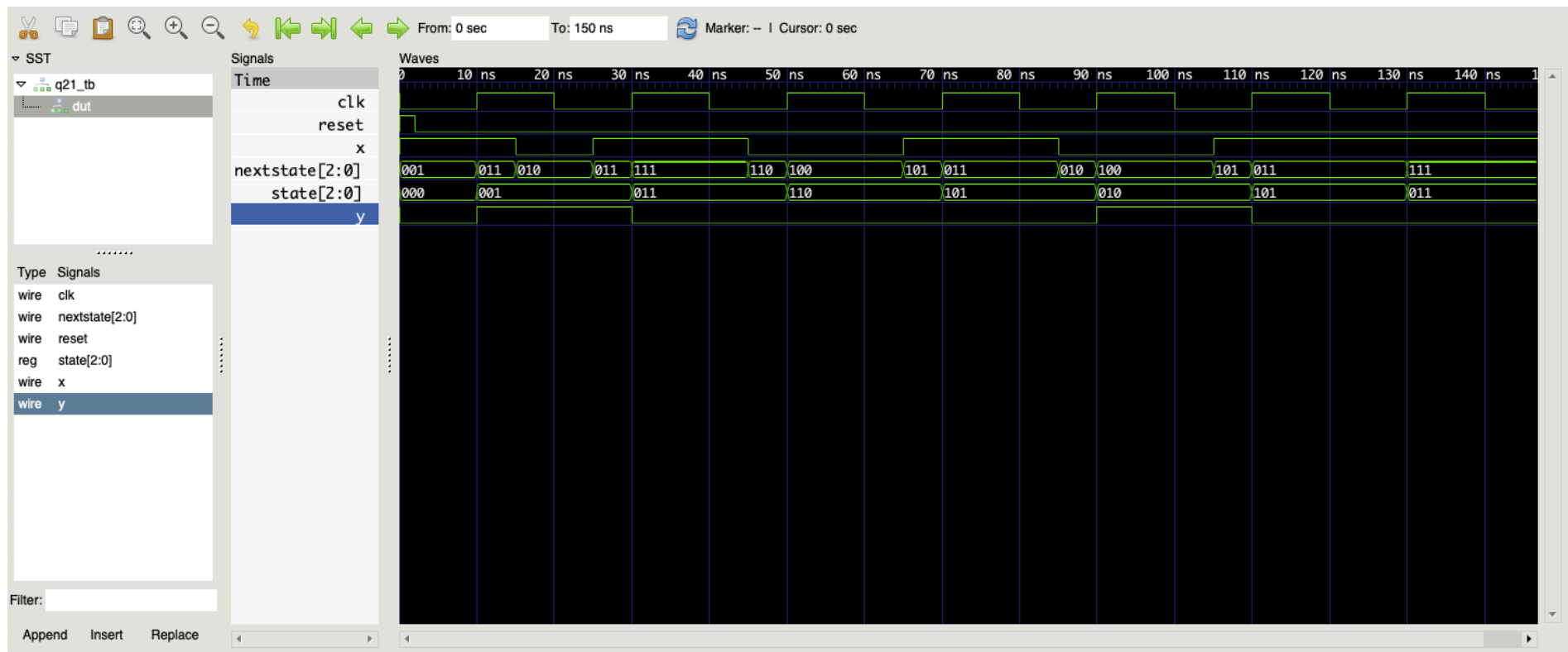
    q7 dut(clk, reset, x, y);

    always begin
        clk=0; #10; clk=1; #10;
    end

    initial begin
        $dumpfile("q7_tb.vcd");
        $dumpvars(0, q7_tb);
        x=1;
            reset=1; #2;
            reset=0; #2;
                #11;
        x=0;                #10;
        x=1;                #20;
        x=0;                #20;
        x=1;                #20;
        x=0;                #20;
        x=1;                #20;
        x=1;                #40;
        $finish;
    end
endmodule
```

Q2. HDL RETE SEQUENZIALE

- ▶ Indicare il **codice di una possibile implementazione in SystemVerilog** di una rete sequenziale con ingresso X a 1 bit la cui uscita Y assuma il valore 1 finché non riconosca negli ultimi tre valori presentati all'ingresso un numero pari di bit con valore 1. Dopo il riconoscimento, a meno che il bit successivo in ingresso termini un ulteriore riconoscimento, l'uscita della rete deve tornare al valore 1 (Esercitazione 2021-12-23 - Q7)
- ▶ Includere un test bench per descrivere in maniera completa il funzionamento della rete



Visualizzazione con GTKWave del file VCD generato dal test bench

RETI LOGICHE

ESERCITAZIONE 2022-01-12

Q17. SINTESI RETE COMBINATORIA

- Fornire la descrizione algebrica minima e la relativa rappresentazione circuitale di una RL con ingresso X a 2 bit, la cui uscita Y riporti il valore definito dalla [1], con n pari al valore ottenuto interpretando come numero intero senza segno i due bit meno significativi della rappresentazione binaria della cifra in **posizione 1** del proprio numero di matricola. Indicare tutti i valori coinvolti.

Posizione	6	5	4	3	2	1	0
Matricola	0	9	8	7	6	4	5

	8	4	2	1
5	0	1	0	0

$$[1] Y = X + k, k = \max \{n, 1\}$$

$$[2] 0 \leq X \leq 3 \rightarrow 0 \leq Y \leq 3 + k$$

$$n = 0, k = \max \{n, 1\} = 1$$

$$[3] Y = X + 1, 0 \leq Y \leq 4$$

Y richiede tre bit perché somma di due numeri a due bit

Q17. SINTESI RETE COMBINATORIA

- Fornire la descrizione algebrica minima e la relativa rappresentazione circuitale di una RL con ingresso X a 2 bit, la cui uscita Y riporti il valore definito dalla [3].

X_1	X_0	Y_2	Y_1	Y_0
0	0	0	0	1
0	1	0	1	0
1	0	0	1	1
1	1	1	0	0

Tabella di verità



[3] $Y = X + 1, 0 \leq Y \leq 4$

Q17. SINTESI RETE COMBINATORIA

- Fornire la descrizione algebrica minima e la relativa rappresentazione circuitale di una RL con ingresso X a 2 bit, la cui uscita Y riporti il valore definito dalla [3].

[3] $Y = X + 1, 0 \leq Y \leq 4$

X_1	X_0	Y_2	Y_1	Y_0
0	0	0	0	1
0	1	0	1	0
1	0	0	1	1
1	1	1	0	0

Tabella di verità

$$Y_0 = \overline{X_1}\overline{X_0} + X_1\overline{X_0} = \overline{X_0}$$

$$Y_1 = \overline{X_1}X_0 + X_1\overline{X_0} = X_1 \oplus X_0$$

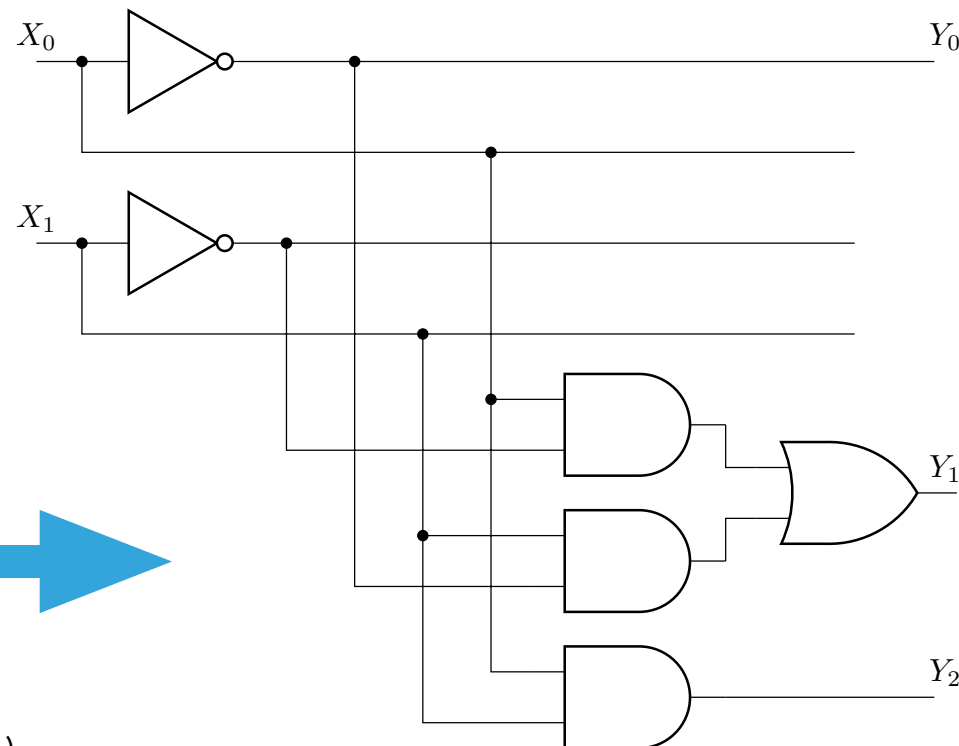
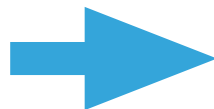
$$Y_2 = X_1X_0$$

Q17. SINTESI RETE COMBINATORIA

- Fornire la descrizione algebrica minima e la relativa rappresentazione circuitale di una RL con ingresso X a 2 bit, la cui uscita Y riporti il valore definito dalla [3].

$$[3] Y = X + 1, 0 \leq Y \leq 4$$

$$\begin{aligned} Y_0 &= \overline{X_1}\overline{X_0} + X_1\overline{X_0} = \overline{X_0} \\ Y_1 &= \overline{X_1}X_0 + X_1\overline{X_0} = X_1 \oplus X_0 \\ Y_2 &= X_1X_0 \end{aligned}$$



Rappresentazione circuitale della rete combinatoria (1)

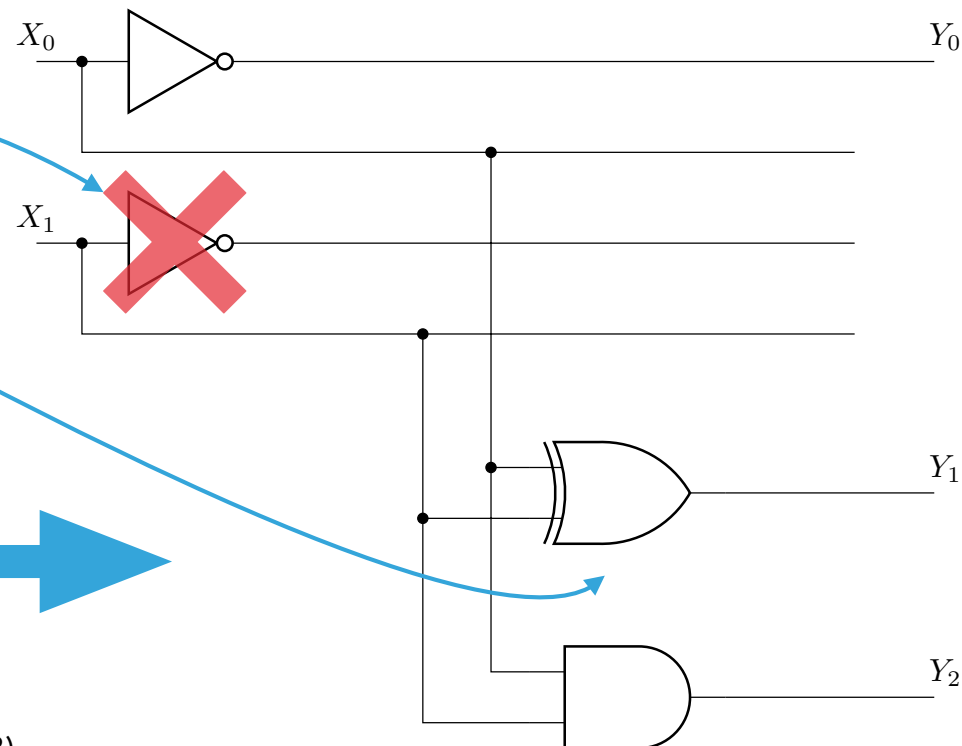
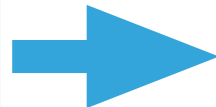
Q17. SINTESI RETE COMBINATORIA

- Fornire la descrizione algebrica minima e la relativa rappresentazione circuitale di una RL con ingresso X a 2 bit, la cui uscita Y riporti il valore definito dalla [3].

$$[3] Y = X + 1, 0 \leq Y \leq 4$$

DUE PORTE AND E UNA PORTA NOT
SONO "ASSORBITE" DALLA PORTA XOR

$$\begin{aligned} Y_0 &= \overline{X_1}\overline{X_0} + X_1\overline{X_0} = \overline{X_0} \\ Y_1 &= \overline{X_1}X_0 + X_1\overline{X_0} = X_1 \oplus X_0 \\ Y_2 &= X_1X_0 \end{aligned}$$



Rappresentazione circuitale della rete combinatoria (2)

Q17. HDL RETE COMBINATORIA

- ▶ Implementazione in SystemVerilog
(non richiesta)
- ▶ Modulo

```
module q17(input logic [1:0] x, output
logic [2:0] y);

    /* studentid: 0987645
    * 0987645 4=4'b0100
    *      ^      ^^
    *              n
    *
    * n=2'b00 k=max(2'b00, 1)=1
    * y=x+2'b01
    */

    /*
    assign y[0]=~x[0];
    assign y[1]= x[1]^x[0];
    assign y[2]= x[1]&x[0];
    */
    assign y=x+2'b01;
endmodule
```

Q17. HDL RETE COMBINATORIA

- Implementazione in SystemVerilog (non richiesta)
- Test bench

```
module q17_tb();
    logic [1:0] x;
    logic [2:0] y;
    logic [2:0] counter;

    q17 dut(x, y);

    initial begin
        $display("x1 x0 | y2 y1 y0");
        $display("-----");
        $monitor(" %b %b | %b %b %b",
x[1], x[0], y[2], y[1], y[0]);
        /* x=2'b00; #1;
           x=2'b01; #1;
           x=2'b10; #1;
           x=2'b11; #1;
        */
        for(counter=0; counter<4;
counter=counter+1) begin
            x=counter[1:0]; #1;
        end
        $finish;
    end
endmodule
```

Q17. HDL RETE COMBINATORIA

- Implementazione in SystemVerilog
(non richiesta)
- Output del test bench

```
→ 17 iverilog -o q17.vvp -g2012 q17.sv
q17_tb.sv
→ 17 vvp q17.vvp
```

x1	x0		y2	y1	y0
0	0		0	0	1
0	1		0	1	0
1	0		0	1	1
1	1		1	0	0

Q18. SINTESI RETE SEQUENZIALE

- ▶ Progettare una rete sequenziale sincrona con ingresso X a 1 bit la cui uscita Y assuma il valore 0 quando riconosca la sequenza di bit 0, 1, 1 negli ultimi valori presentati all'ingresso. L'uscita Y deve assumere il valore 1 per tutte le altre sequenze di ingresso. Si producano il **diagramma degli stati**, la tabella di transizione e il diagramma circuitale.

Q18. SINTESI RETE SEQUENZIALE

- Progettare una rete sequenziale sincrona con ingresso X a 1 bit la cui uscita Y assuma il valore 0 quando riconosca la sequenza di bit 0, 1, 1 negli ultimi valori presentati all'ingresso. L'uscita Y deve assumere il valore 1 per tutte le altre sequenze di ingresso. Si producano il **diagramma degli stati**, la tabella di transizione e il diagramma circuitale.

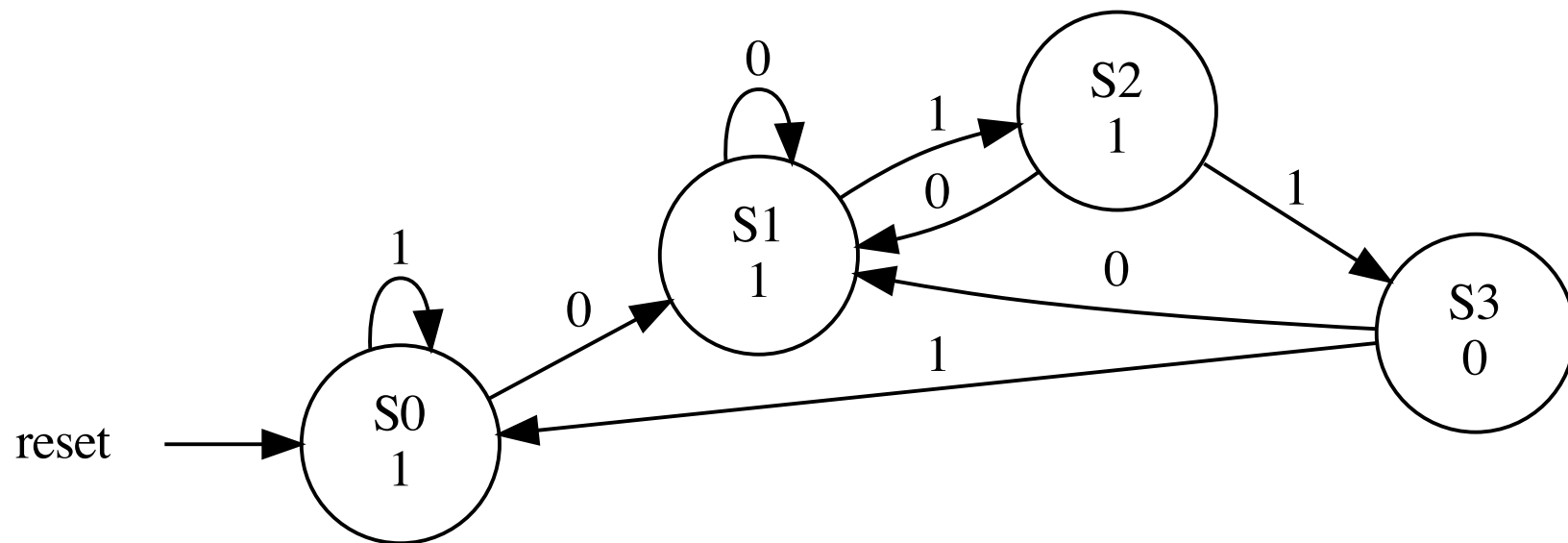


Diagramma degli stati

Q18. SINTESI RETE SEQUENZIALE

- Progettare una rete sequenziale sincrona con ingresso X a 1 bit la cui uscita Y assuma il valore 0 quando riconosca la sequenza di bit 0, 1, 1 negli ultimi valori presentati all'ingresso. L'uscita Y deve assumere il valore 1 per tutte le altre sequenze di ingresso. Si producano il **diagramma degli stati**, la **tabella di transizione** e il diagramma circuitale.

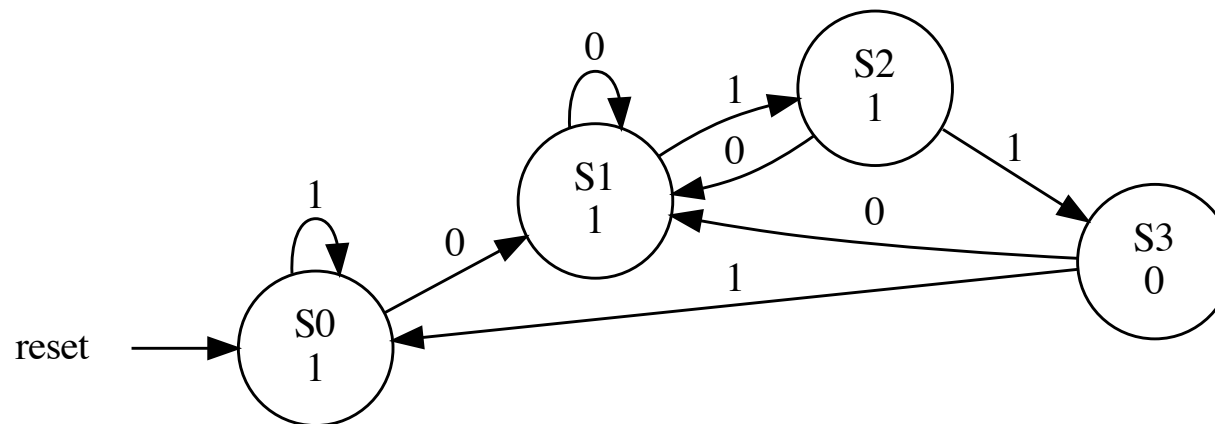


Diagramma degli stati

Current State: S	Input: X	Next State: S'
S0	0	S1
S0	1	S0
S1	0	S1
S1	1	S2
S2	0	S1
S2	1	S3
S3	0	S1
S3	1	S0

Tabella di transizione

Q18. SINTESI RETE SEQUENZIALE

- Progettare una rete sequenziale sincrona con ingresso X a 1 bit la cui uscita Y assuma il valore 0 quando riconosca la sequenza di bit 0, 1, 1 negli ultimi valori presentati all'ingresso. L'uscita Y deve assumere il valore 1 per tutte le altre sequenze di ingresso. Si producano il **diagramma degli stati**, la **tabella di transizione** e il diagramma circuitale.

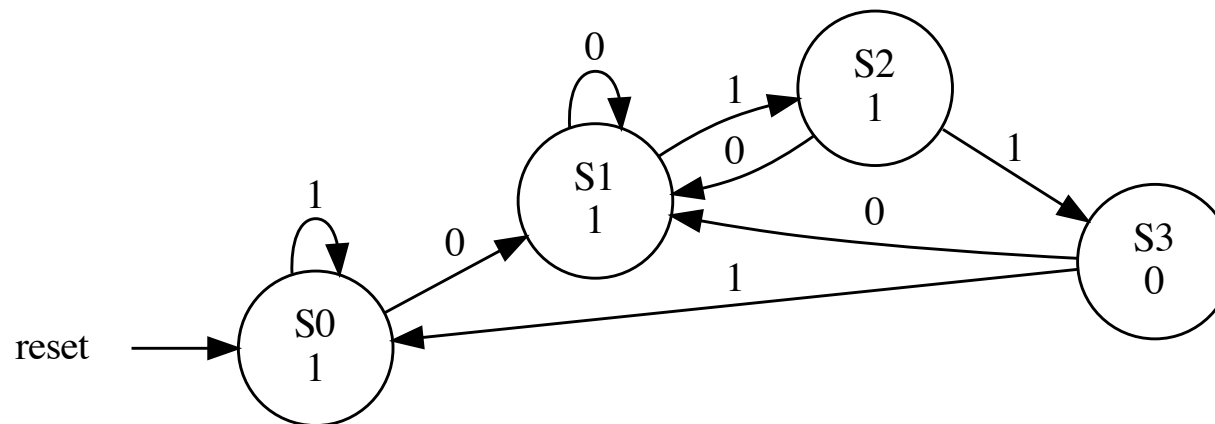


Diagramma degli stati

Current State: S	Output: Y
S0	1
S1	1
S2	1
S3	0

Tabella di uscita

Q18. SINTESI RETE SEQUENZIALE

- Progettare una rete sequenziale sincrona con ingresso X a 1 bit la cui uscita Y assuma il valore 0 quando riconosca la sequenza di bit 0, 1, 1 negli ultimi valori presentati all'ingresso. L'uscita Y deve assumere il valore 1 per tutte le altre sequenze di ingresso. Si producano il **diagramma degli stati**, la **tabella di transizione** e il diagramma circuitale.

State	S_1S_0
S0	00
S1	01
S2	10
S3	11

Tabella di codifica degli stati

Current State: S_1S_0	Output: Y
00	1
01	1
10	1
11	0

Tabella di uscita (codificata)

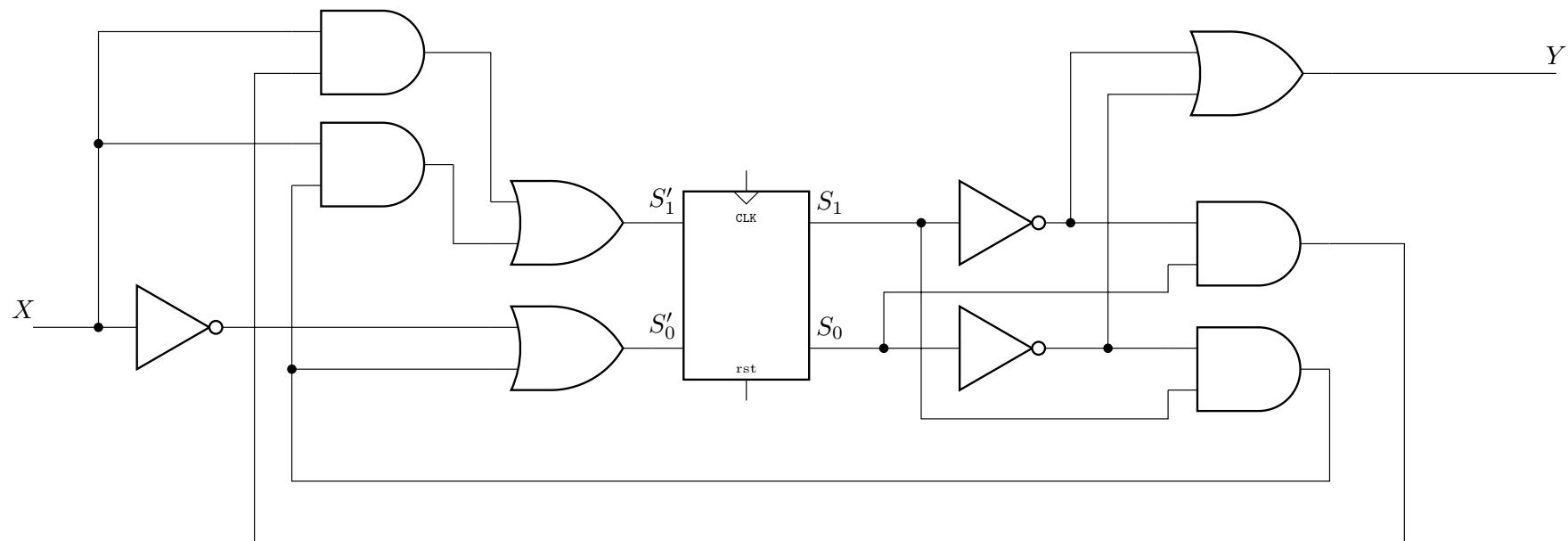
$$\begin{aligned}
 S'_1 &= \overline{S_1}S_0X + S_1\overline{S_0}X \\
 S'_0 &= S_1\overline{S_0} + \overline{X} \\
 Y &= \overline{S_1} + \overline{S_0}
 \end{aligned}$$

Current State: S_1S_0	Input: X	Next State: $S'_1S'_0$
00	0	01
00	1	00
01	0	01
01	1	10
10	0	01
10	1	11
11	0	01
11	1	00

Tabella di transizione (codificata)

Q18. SINTESI RETE SEQUENZIALE

- Progettare una rete sequenziale sincrona con ingresso X a 1 bit la cui uscita Y assuma il valore 0 quando riconosca la sequenza di bit 0, 1, 1 negli ultimi valori presentati all'ingresso. L'uscita Y deve assumere il valore 1 per tutte le altre sequenze di ingresso. Si producano il **diagramma degli stati**, la **tabella di transizione** e il diagramma circuitale.



$$S'_1 = \overline{S_1}S_0X + S_1\overline{S_0}X$$

$$S'_0 = S_1\overline{S_0} + \overline{X}$$

$$Y = \overline{S_1} + \overline{S_0}$$

Q19. HDL RETE SEQUENZIALE

- Indicare il **codice di una possibile implementazione** in SystemVerilog della rete sequenziale descritta nel quesito 19. Includere un test bench per descrivere in maniera completa il funzionamento della rete.

```
module q19(input logic clk, reset, x,
output logic y);

    logic [1:0] state, nextstate;
    enum logic [1:0] {S0, S1, S2, S3}
statevals;

    always_ff @(posedge clk, posedge
reset) begin
        if(reset) begin
            state=S0;
        end else begin
            state<=nextstate;
        end
    end

    always_comb begin
        case(state)
            S0 : nextstate= (x ? S0 : S1);
            S1 : nextstate= (x ? S2 : S1);
            S2 : nextstate= (x ? S3 : S1);
            S3 : nextstate= (x ? S0 : S1);
            default: nextstate= S0;
        endcase
    end
    assign y= ~(state==S3);

endmodule
```

Q19. HDL RETE SEQUENZIALE

- Indicare il codice di una possibile implementazione in SystemVerilog della rete sequenziale descritta nel quesito 19. Includere un **test bench** per descrivere in maniera completa il funzionamento della rete.

```
`timescale 1ns/1ns

module q19_tb();
    logic clk, reset, x, y;

    q19 dut(clk, reset, x, y);

    always begin
        clk=0; #10; clk=1; #10;
    end

    initial begin
        $dumpfile("q21_tb.vcd");
        $dumpvars(0, q21_tb);
        x=1;
            reset=1; #2;
            reset=0; #2;
                #11;
        x=0;                #10;
        x=1;                #20;
        x=0;                #20;
        x=1;                #20;
        x=0;                #20;
        x=1;                #20;
        x=1;                #25;
        $finish;
    end
endmodule
```

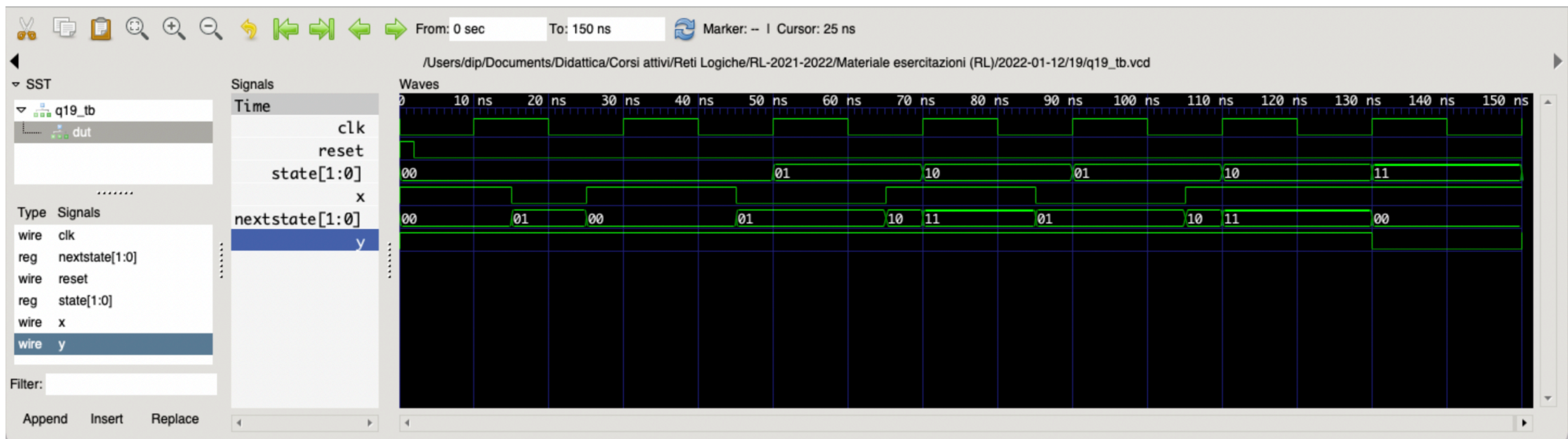
Q19. HDL RETE SEQUENZIALE

- Indicare il **codice di una possibile implementazione** in SystemVerilog della rete sequenziale descritta nel quesito 19. Includere un test bench per descrivere in maniera completa il funzionamento della rete.

```
module q19(input logic clk, reset, x, output logic y);  
  
    logic [1:0] state, nextstate;  
    enum logic [1:0] {S0, S1, S2, S3} statevals;  
  
    always_ff @(posedge clk, posedge reset) begin  
        if(reset) begin  
            state=S0;  
        end else begin  
            state<=nextstate;  
        end  
    end  
  
    assign nextstate[1]= (~state[1]&state[0]&x) | (state[1]&~state[0]&x);  
    assign nextstate[0]= (state[1]&~state[0]) | ~x;  
    assign y= ~state[1]|~state[0];  
endmodule
```


Q19. HDL RETE SEQUENZIALE

- Indicare il codice di una possibile implementazione in SystemVerilog della rete sequenziale descritta nel quesito 19. Includere un test bench per descrivere in maniera completa il funzionamento della rete.



RETI LOGICHE

ESERCITAZIONE 2022-01-13

Q14. MINIMIZZAZIONE CON K-MAP

- ▶ Q. Minimizzare in forma SOP la funzione indicata usando k-map e tenendo conto delle non-specificazioni (indifferenze).

- ▶ $F(A, B, C, D) = \prod (1, 7, 9, 13)$

- ▶ $d(A, B, C, D) = \sum (5, 15)$

- ▶ R. $F(A, B, C, D) =$

Q14. MINIMIZZAZIONE CON K-MAP

- ▶ Q. Minimizzare in forma SOP la funzione indicata usando k-map e tenendo conto delle non-specificazioni (indifferenze).

- ▶ $F(A, B, C, D) = \prod (1, 7, 9, 13)$

- ▶ $d(A, B, C, D) = \sum (5, 15)$

- ▶ La funzione è in forma POS:

- ▶ le celle 1, 7, 9 e 13 hanno valore 0

- ▶ Le indifferenze sono date in forma SOP:

- ▶ le celle 5 e 15 hanno valore X

- ▶ Le celle rimanenti hanno valore 1

- ▶ R. $F(A, B, C, D) =$

		<i>CD</i>			
		00	01	11	10
<i>AB</i>	00	1	0	1	1
	01	1	X	0	1
	11	1	0	X	1
	10	1	0	1	1

K-map con indifferenze

Q14. MINIMIZZAZIONE CON K-MAP

- ▶ Q. Minimizzare in forma SOP la funzione indicata usando k-map e tenendo conto delle non-specificazioni (indifferenze).

- ▶ $F(A, B, C, D) = \prod (1, 7, 9, 13)$

- ▶ $d(A, B, C, D) = \sum (5, 15)$

- ▶ La funzione è in forma POS:

- ▶ le celle 1, 7, 9 e 13 hanno valore 0

- ▶ Le indifferenze sono date in forma SOP:

- ▶ le celle 5 e 15 hanno valore X

- ▶ Le celle rimanenti hanno valore 1

- ▶ R. $F(A, B, C, D) = \overline{D} + \overline{B}C$

		<i>CD</i>			
		00	01	11	10
<i>AB</i>	00	1	0	1	1
	01	1	X	0	1
	11	1	0	X	1
	10	1	0	1	1

K-map con risoluzione delle indifferenze

		<i>CD</i>			
		00	01	11	10
<i>AB</i>	00	1	0	1	1
	01	1	X	0	1
	11	1	0	X	1
	10	1	0	1	1

Q17. SINTESI RETE COMBINATORIA

- Fornire la descrizione algebrica minima e la relativa rappresentazione circuitale di una RL, con ingresso X a due bit, la cui uscita Y riporti il valore definito dalla [1], con k pari al valore ottenuto interpretando come numero intero senza segno i due bit meno significativi della rappresentazione binaria della cifra in **posizione 2** del proprio numero di matricola, da indicare nella risposta, più uno.

Posizione	6	5	4	3	2	1	0
Matricola	0	9	8	7	6	4	5

8	4	2	1
6	0	1	1 0

$$[1] Y = X \cdot 2^k + 1, k = n + 1$$

$$[2] 0 \leq X \leq 3 \rightarrow 0 \leq Y \leq 3 \cdot 2^k$$

$$n = 2, k = n + 1 = 2 + 1 = 3$$

$$[3] Y = X \cdot 2^3 + 1, 0 \leq Y \leq 3 \cdot 2^3 + 1$$

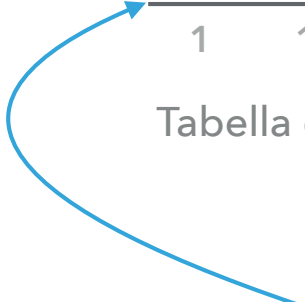
Y richiede cinque bit, Y[2:0]=3'b001

Q17. SINTESI RETE COMBINATORIA

- Fornire la descrizione algebrica minima e la relativa rappresentazione circuitale di una RL, con ingresso X a due bit, la cui uscita Y riporti il valore definito dalla [3]

X_1	X_0	Y_4	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0	0	1
0	1	0	1	0	0	1
1	0	1	0	0	0	1
1	1	1	1	0	0	1

Tabella di verità



[3] $Y = X \cdot 2^3 + 1$

Q17. SINTESI RETE COMBINATORIA

- Fornire la descrizione algebrica minima e la relativa rappresentazione circuitale di una RL, con ingresso X a due bit, la cui uscita Y riporti il valore definito dalla [3]

[3] $Y = X \cdot 2^3 + 1$

X_1	X_0	Y_4	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0	0	1
0	1	0	1	0	0	1
1	0	1	0	0	0	1
1	1	1	1	0	0	1

Tabella di verità

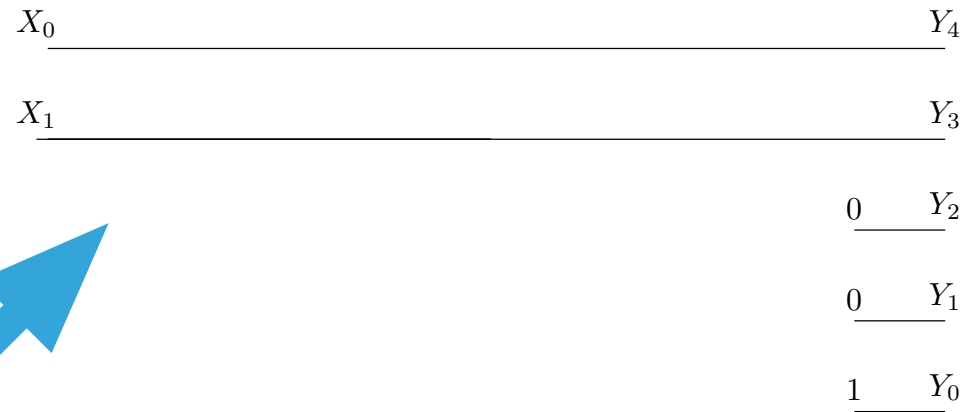
$$\begin{aligned} Y_0 &= 1 \\ Y_2 &= Y_1 = 0 \\ Y_3 &= X_0 \\ Y_4 &= X_1 \end{aligned}$$

Q17. SINTESI RETE COMBINATORIA

- Fornire la descrizione algebrica minima e la relativa rappresentazione circuitale di una RL, con ingresso X a due bit, la cui uscita Y riporti il valore definito dalla [3]

$$[3] Y = X \cdot 2^3 + 1$$

$$\begin{aligned} Y_0 &= 1 \\ Y_2 &= Y_1 = 0 \\ Y_3 &= X_0 \\ Y_4 &= X_1 \end{aligned}$$

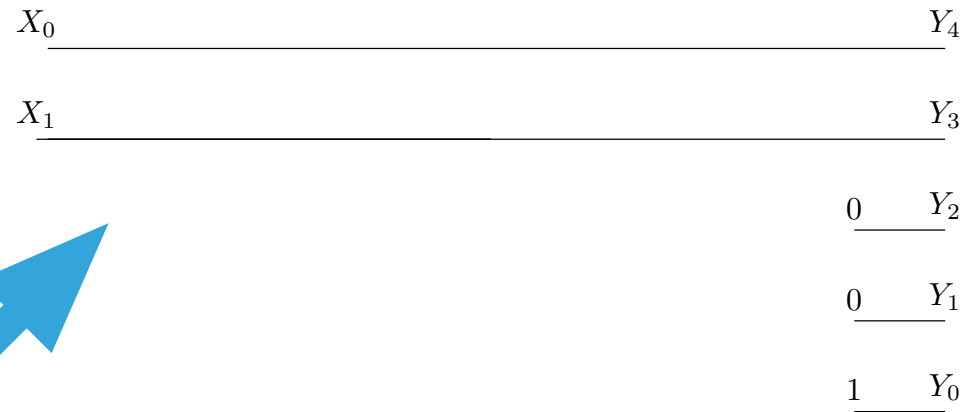


Q17. SINTESI RETE COMBINATORIA

- Fornire la descrizione algebrica minima e la relativa rappresentazione circuitale di una RL, con ingresso X a due bit, la cui uscita Y riporti il valore definito dalla [3]

$$[3] Y = X \cdot 2^3 + 1$$

$$\begin{aligned} Y_0 &= 1 \\ Y_2 &= Y_1 = 0 \\ Y_3 &= X_0 \\ Y_4 &= X_1 \end{aligned}$$

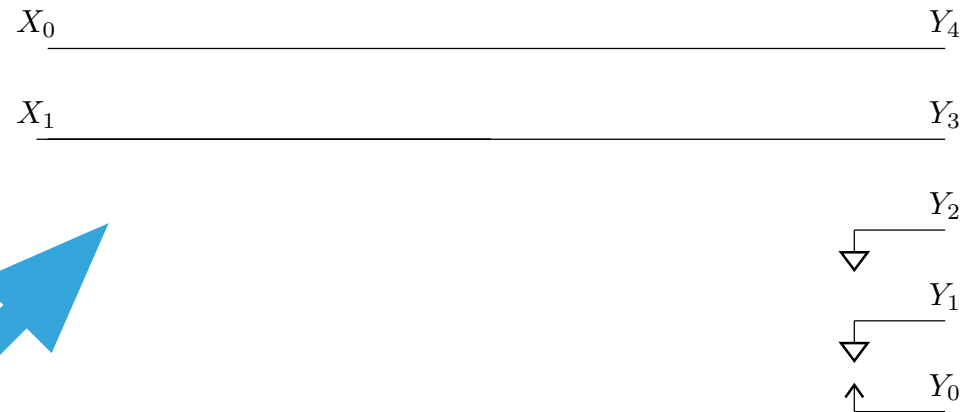


Q17. SINTESI RETE COMBINATORIA

- Fornire la descrizione algebrica minima e la relativa rappresentazione circuitale di una RL, con ingresso X a due bit, la cui uscita Y riporti il valore definito dalla [3]

$$[3] Y = X \cdot 2^3 + 1$$

$$\begin{aligned} Y_0 &= 1 \\ Y_2 &= Y_1 = 0 \\ Y_3 &= X_0 \\ Y_4 &= X_1 \end{aligned}$$



Q17. SINTESI RETE COMBINATORIA

- ▶ Implementazione in SystemVerilog
(non richiesta)
 - ▶ Modulo

```
module q17(input logic [1:0] x, output
logic [4:0] y);

    /* studentid: 0987645
    * 0987645 6=4'b0110
    *      ^      ^^
    *              n
    *
    * n=2'b10 k=2+1
    * y=(x*2^^3)+1
    */

    /*
    assign y={x, 3'b001};
    */
    assign y=(x<<3)+1;
endmodule
```

Q17. SINTESI RETE COMBINATORIA

- Implementazione in SystemVerilog (non richiesta)
 - Test bench

```
module q17_tb();
    logic [1:0] x;
    logic [4:0] y;
    logic [2:0] counter;

    q17 dut(x, y);

    initial begin
        $display("y=(x*2^^3)+1=x*8+1");
        $display("x1 x0 | y4 y3 y2 y1 y0 ||
x   y");

        $display("-----
--");

        $monitor(" %b %b | %b %b %b %b
%b ||%2d %3d", x[1], x[0], y[4], y[3],
y[2], y[1], y[0], x, y);
        /* x=2'b00; #1;
           x=2'b01; #1;
           x=2'b10; #1;
           x=2'b11; #1;
        */
        for(counter=0; counter<4;
counter=counter+1) begin
            x=counter[1:0]; #1;
        end
        $finish;
    end
end
```

Q17. SINTESI RETE COMBINATORIA

- ▶ Implementazione in SystemVerilog (non richiesta)
 - ▶ Output del test bench

```
→ 17 iverilog -g2012 -o q17.vvp q17.sv
q17_tb.sv
→ 17 vvp q17.vvp
y=(x*2^^3)+1=x*8+1
x1 x0 | y4 y3 y2 y1 y0 || x  y
-----
0  0 | 0  0  0  0  1 || 0  1
0  1 | 0  1  0  0  1 || 1  9
1  0 | 1  0  0  0  1 || 2 17
1  1 | 1  1  0  0  1 || 3 25
→ 17
```

Q18. SINTESI RETE SEQUENZIALE

- ▶ Progettare una rete sequenziale con un ingresso a un bit la cui uscita riporti in maniera continua (sia trasparente) per tutta la durata del ciclo i -esimo il valore dell'ingresso se quest'ultimo ha assunto il valore 1 in corrispondenza al fronte di salita del clock del ciclo $i-1$ -esimo. In caso contrario l'uscita deve essere 0.

Q18. SINTESI RETE SEQUENZIALE

- Progettare una rete sequenziale con un ingresso a un bit la cui uscita riporti in maniera continua (sia trasparente) per tutta la durata del ciclo i -esimo il valore dell'ingresso se quest'ultimo ha assunto il valore 1 in corrispondenza al fronte di salita del clock del ciclo $i-1$ -esimo. In caso contrario l'uscita deve essere 0.

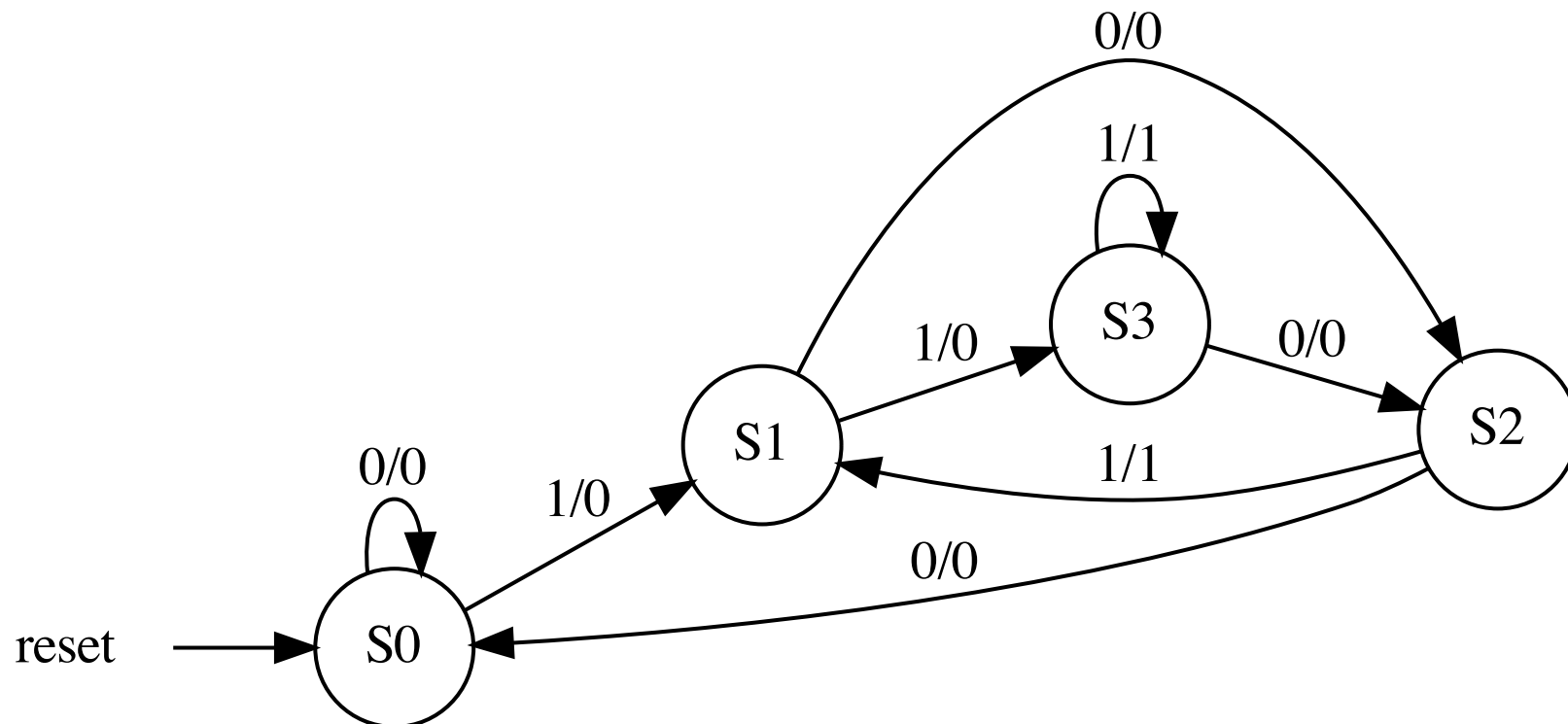


Diagramma degli stati

Q18. SINTESI RETE SEQUENZIALE

- Progettare una rete sequenziale con un ingresso a un bit la cui uscita riporti in maniera continua (sia trasparente) per tutta la durata del ciclo i -esimo il valore dell'ingresso se quest'ultimo ha assunto il valore 1 in corrispondenza al fronte di salita del clock del ciclo $i-1$ -esimo. In caso contrario l'uscita deve essere 0.

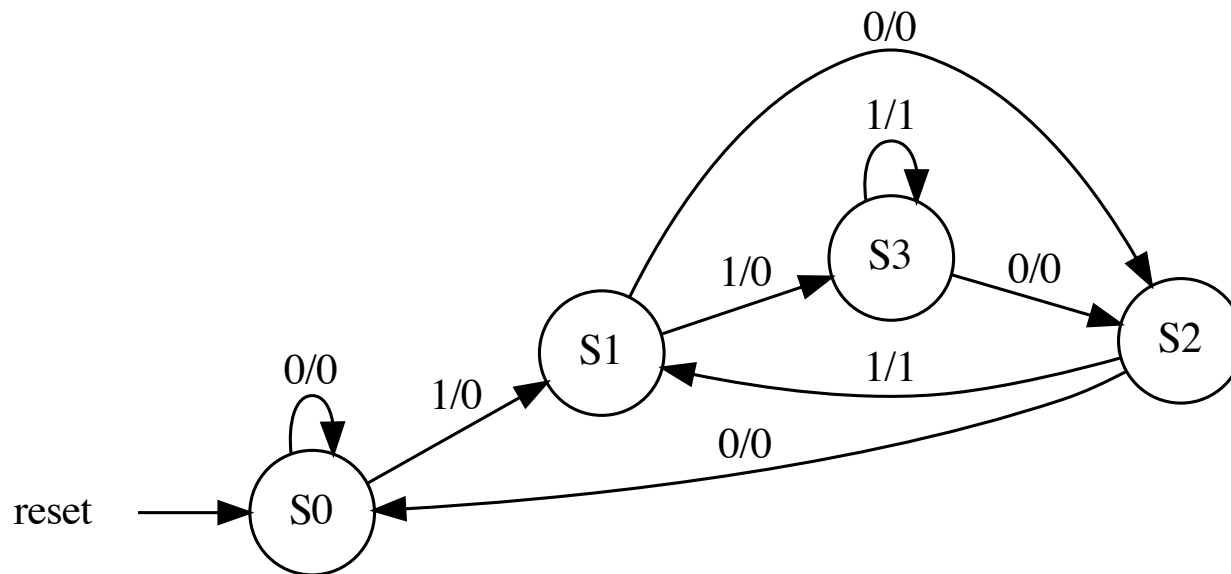


Diagramma degli stati

Current State: S	Input: P	Next State:	Output: Y
S0	0	S0	0
S0	1	S1	0
S1	0	S2	0
S1	1	S3	0
S2	0	S0	0
S2	1	S1	1
S3	0	S2	0
S3	1	S3	1

Tabella di transizione e uscita

Q18. SINTESI RETE SEQUENZIALE

- Progettare una rete sequenziale con un ingresso a un bit la cui uscita riporti in maniera continua (sia trasparente) per tutta la durata del ciclo i -esimo il valore dell'ingresso se quest'ultimo ha assunto il valore 1 in corrispondenza al fronte di salita del clock del ciclo $i-1$ -esimo. In caso contrario l'uscita deve essere 0.

State	S_1S_0
S0	00
S1	01
S2	10
S3	11

Tabella di codifica degli stati

$$\begin{aligned}
 S'_1 &= S_0 \\
 S'_0 &= P \\
 Y &= S_1P
 \end{aligned}$$

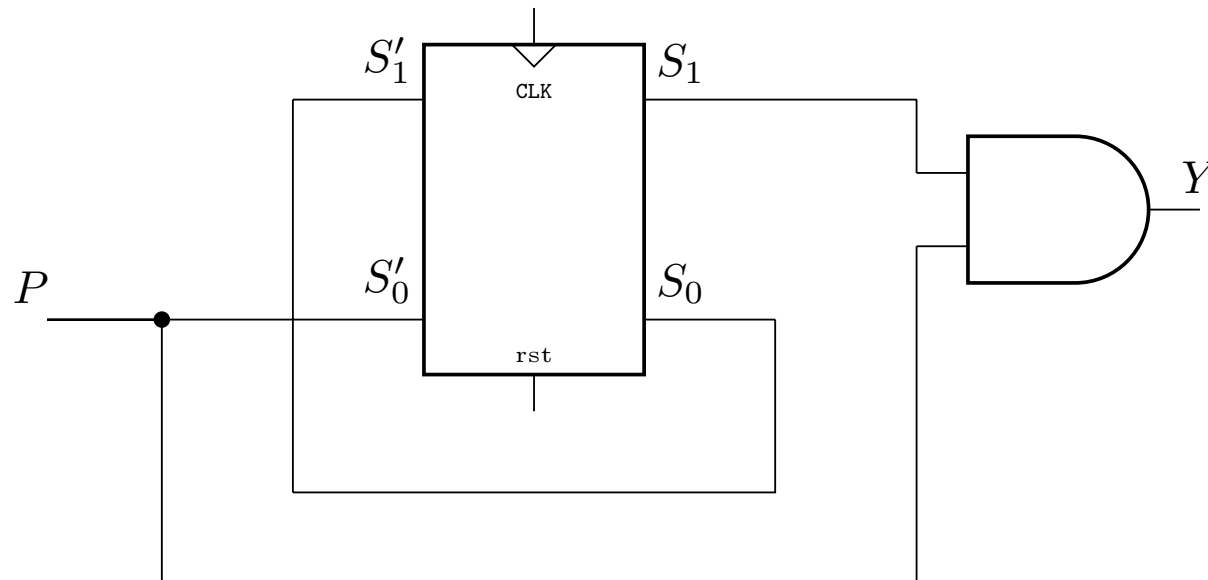
Current State: S_1S_0	Input: P	Next State: $S'_1S'_0$	Output: Y
00	0	00	0
00	1	01	0
01	0	10	0
01	1	11	0
10	0	00	0
10	1	01	1
11	0	10	0
11	1	11	1

Tabella di transizione e uscita (codificata)

Q18. SINTESI RETE SEQUENZIALE

- Progettare una rete sequenziale con un ingresso a un bit la cui uscita riporti in maniera continua (sia trasparente) per tutta la durata del ciclo i-esimo il valore dell'ingresso se quest'ultimo ha assunto il valore 1 in corrispondenza al fronte di salita del clock del ciclo i-1-esimo. In caso contrario l'uscita deve essere 0.

$$\begin{aligned}S'_1 &= S_0 \\ S'_0 &= P \\ Y &= S_1 P\end{aligned}$$



Q19. HDL RETE SEQUENZIALE

- ▶ Indicare il codice di una **possibile implementazione** in SystemVerilog della rete sequenziale descritta nel quesito 18. Includere un test bench per descrivere in maniera completa il funzionamento della rete.
- ▶ Esempio di soluzione (1)
 - ▶ Traduzione della FSM in SystemVerilog

```
module q19(input logic clk, reset, p,
output logic y);

    enum logic [1:0] {S0, S1, S2, S3}
statevals;
    logic [1:0] state, nextstate;

    always_ff @(posedge clk, posedge
reset) begin
        if(reset) begin
            state<=S0;
        end else begin
            state<=nextstate;
        end
    end

    always_comb begin
        case(state)
            S0 : nextstate= (p ? S1 : S0);
            S1 : nextstate= (p ? S3 : S2);
            S2 : nextstate= (p ? S1 : S0);
            S3 : nextstate= (p ? S3 : S2);
            default: nextstate= S0;
        endcase
    end

    assign y=state[1]&p;
endmodule
```

Q19. HDL RETE SEQUENZIALE

- ▶ Indicare il codice di una **possibile implementazione** in SystemVerilog della rete sequenziale descritta nel quesito 18. Includere un test bench per descrivere in maniera completa il funzionamento della rete.
- ▶ Esempio di soluzione (2)
 - ▶ Implementazione con registro a scorrimento

```
module q19(input logic clk, reset, p,  
output logic y);  
  
    logic [1:0] state, nextstate;  
  
    always_ff @(posedge clk, posedge  
reset) begin  
        if(reset) begin  
            state<=2'b00;  
        end else begin  
            state<=nextstate;  
        end  
    end  
  
    assign nextstate[1]=state[0];  
    assign nextstate[0]=p;  
    assign y=state[1]&p;  
  
endmodule
```

Q19. HDL RETE SEQUENZIALE

- ▶ Indicare il codice di una possibile implementazione in SystemVerilog della rete sequenziale descritta nel quesito 18. Includere un **test bench** per descrivere in maniera completa il funzionamento della rete
- ▶ I due blocchi `initial` vengono eseguiti in parallelo da `t=0`

```
`timescale 1ns/1ns

module q19_tb();
    logic clk, reset, p, y;

    q19 dut(clk, reset, p, y);

    always begin
        clk=0; #10; clk=1; #10;
    end

    initial begin
        $dumpfile("q19_tb.vcd");
        $dumpvars(0, q19_tb);
        reset=1; #2;
        reset=0;
    end

    initial begin
        p=1; #3;
        p=0; #4;
        p=1; #8;
        p=0; #10;
        p=1; #20;
        p=0; #20;
        p=1; #20;
        p=0; #20;
        p=1; #20;
        p=1; #25;
        $finish;
    end
end
endmodule
```

Q19. HDL RETE SEQUENZIALE

- Indicare il codice di una possibile implementazione in SystemVerilog della rete sequenziale descritta nel quesito 18. Includere un test bench per descrivere in maniera completa il funzionamento della rete.

