Circuit Design with VHDL

Chapter 12

Instructor: Ali Jahanian

Outline

- I-CIRCUIT DESIGN
  - 6 Sequential Code
    - 6.1 PROCESS
    - 6.2 Signals and Variables
    - 6.3 IF
    - 6.4 WAIT
    - 6.5 CASE
    - 6.6 LOOP
    - 6.7 CASE versus IF
    - 6.8 CASE versus WHEN
    - 6.9 Bad Clocking
    - 6.10 Using Sequential Code to Design Combinational Circuits
    - 6.11 Problems

VHDL code

- VHDL code can be:
  - concurrent (parallel)
    - Chapter 5
  - Sequential
    - **This chapter**

Sequential Code

PROCESSES,

FUNCTIONS,

PROCEDURES

are the only sections of code that are executed sequentially.

However, as a whole, any of these blocks is still concurrent with any other statements placed outside it.

Sequential Code …

- One important aspect of sequential code is that it is not limited to sequential logic.
- Indeed, with it we can build sequential circuits as well as combinational circuits.
- *Sequential* code is also called *behavioral* code.

Sequential Code …

- The statements discussed in this section are all *sequential*. They are:

  IF, *WAIT*, *CASE*, and *LOOP*.

- VARIABLES are also restricted to be used in sequential code only. Thus, contrary to a SIGNAL, a VARIABLE can never be *global*, so its value can not be *passed out directly*.

PROCESS

- A PROCESS is a sequential section of VHDL code.

```
[label:] PROCESS (sensitivity list)
  [VARIABLE name type [range] [:= initial_value;]]
BEGIN
  (sequential code)
END PROCESS [label];
```

PROCESS (Example: DFF)

```
1  ------------------------------------
2  LIBRARY ieee;
3  USE ieee.std_logic_1164.all;
4  ------------------------------------
5  ENTITY dff IS
6     PORT (d, clk, rst: IN STD_LOGIC;
7           q: OUT STD_LOGIC);
8  END dff;
9  ------------------------------------
10 ARCHITECTURE behavior OF dff IS
11 BEGIN
12    PROCESS (clk, rst)
13    BEGIN
14       IF (rst='1') THEN
15          q <= '0';
16       ELSIF (clk'EVENT AND clk='1') THEN
17          q <= d;
18    END IF;
19    END PROCESS;
20 END behavior;
21 ------------------------------------
```

PROCESS (Example: DFF) …

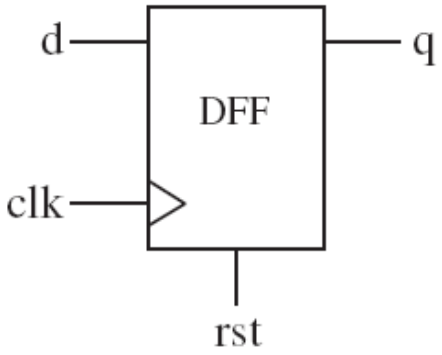


Figure 6.1
DFF with asynchronous reset of example 6.1.

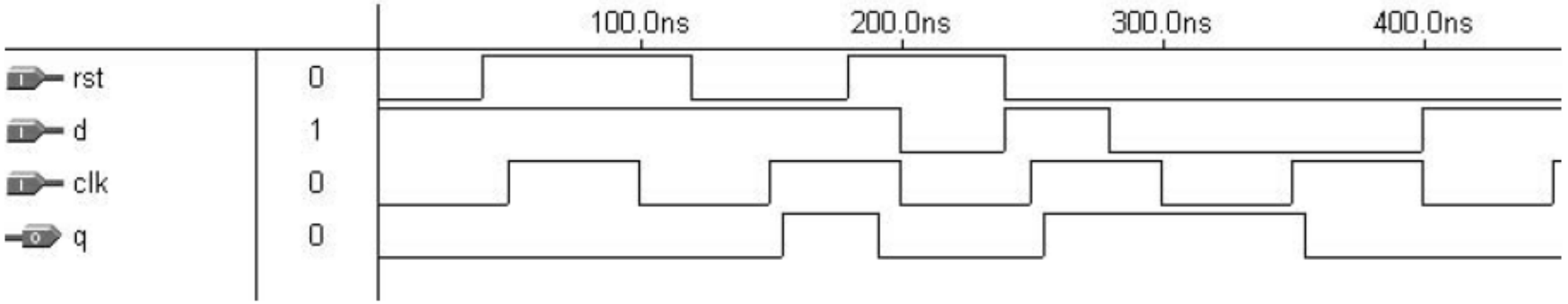


Figure 6.2
Simulation results of example 6.1.

Signals and Variables

- Signals and variables will be studied in detail in the chapter 7.
- A SIGNAL can be declared in a
  - PACKAGE,
  - ENTITY,
  - ARCHITECTURE (in its declarative part)

Global

- A VARIABLE can
  - only be declared inside a piece of sequential code (in a PROCESS, for example).

Local

Signals and Variables …

- The value of a VARIABLE can never be passed out of the PROCESS directly.
- On the other hand, the update of a VARIABLE is immediate, that is, we can promptly count on its new value in the next line of code. That is not the case with a SIGNAL (when used in a PROCESS), for its new value is generally only guaranteed to be available after the onclusion of the present run of the PROCESS.

Signals and Variables …

- The assignment operator
  - for a SIGNAL is “<=”
    - (example: sig <=5),
  - while for a VARIABLE it is “:=”
    - (example: var := 5).

IF

```
IF conditions THEN assignments;
ELSIF conditions THEN assignments;
...
ELSE assignments;
END IF;
```

Example:

```
IF (x<y) THEN temp:="11111111";
ELSIF (x=y AND w='0') THEN temp:="11110000";
ELSE temp:=(OTHERS =>'0');
```

IF (Example: One-digit Counter) . . .

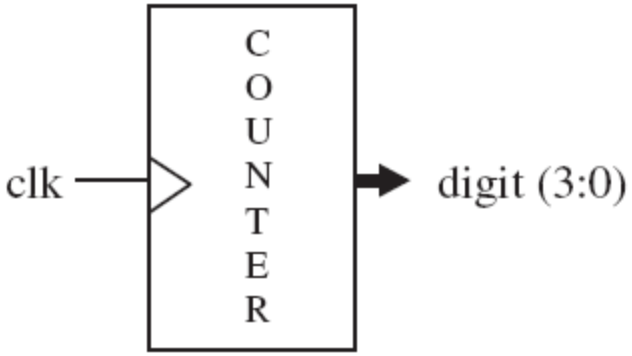


Figure 6.3
Counter of example 6.2.

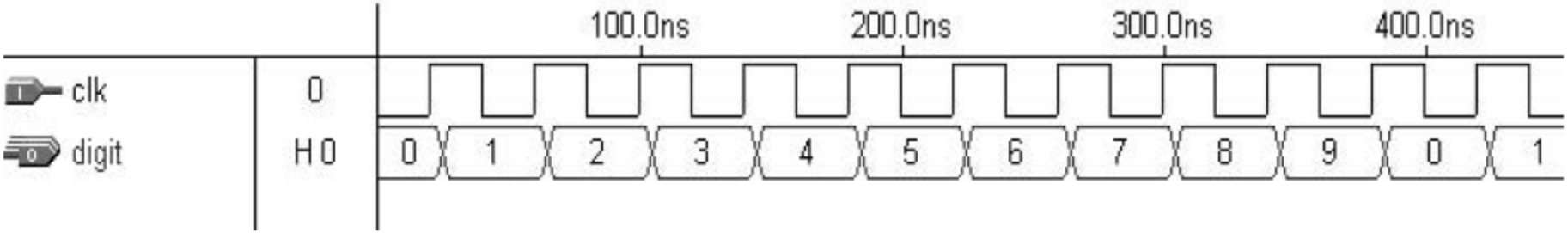


Figure 6.4
Simulation results of example 6.2.

IF (Example: One-digit Counter) . . .

```
1  ------------------------------------------
2  LIBRARY ieee;
3  USE ieee.std_logic_1164.all;
4  ------------------------------------------
5  ENTITY counter IS
6     PORT (clk : IN STD_LOGIC;
7           digit : OUT INTEGER RANGE 0 TO 9);
8  END counter;
9  ------------------------------------------
10 ARCHITECTURE counter OF counter IS
11 BEGIN
12    count: PROCESS(clk)
13       VARIABLE temp : INTEGER RANGE 0 TO 10;
14    BEGIN
15       IF (clk'EVENT AND clk='1') THEN
16          temp := temp + 1;
17          IF (temp=10) THEN temp := 0;
18          END IF;
19       END IF;
20       digit <= temp;
21    END PROCESS count;
22 END counter;
23 ------------------------------------------
```

IF (Example: Shift Register) . . .

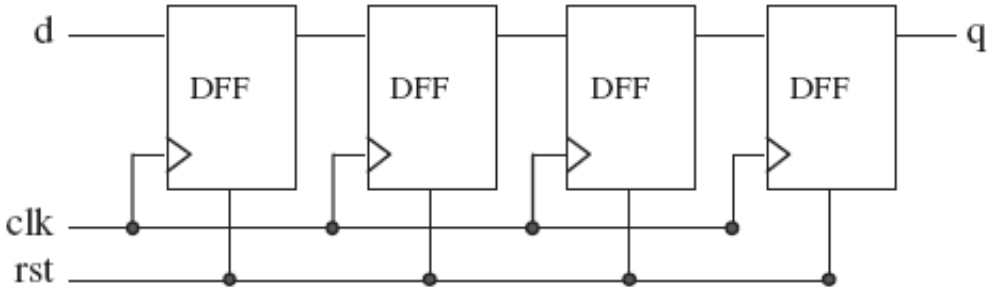


Figure 6.5
Shift register of example 6.3.

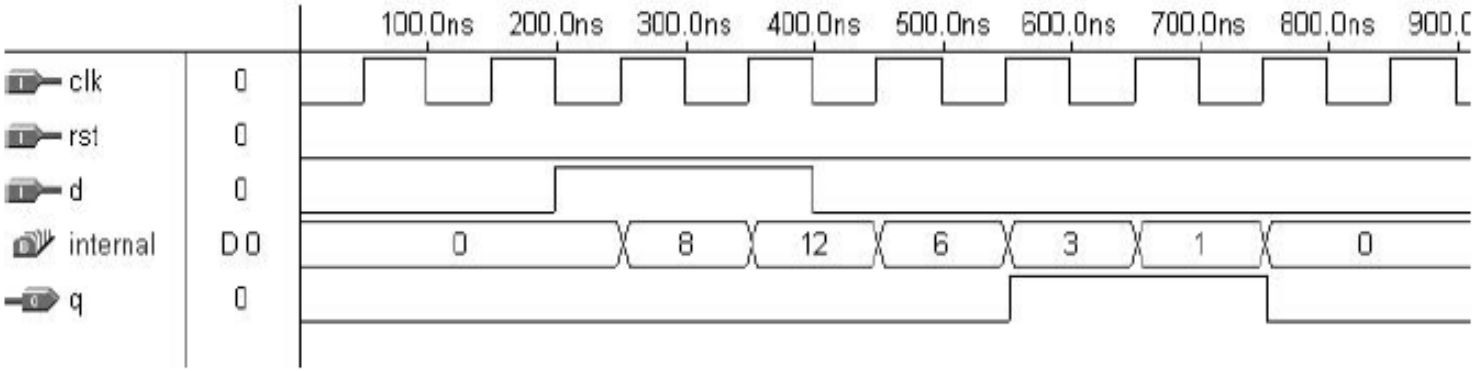


Figure 6.6
Simulation results of example 6.3.

IF (Example: Shift Register) . . .

```
1  ----------------------------------------------------
2  LIBRARY ieee;
3  USE ieee.std_logic_1164.all;
4  ----------------------------------------------------
5  ENTITY shiftreg IS
6     GENERIC (n: INTEGER := 4);     -- # of stages
7     PORT (d, clk, rst: IN STD_LOGIC;
8           q: OUT STD_LOGIC);
9  END shiftreg;
10 ----------------------------------------------------
11 ARCHITECTURE behavior OF shiftreg IS
12    SIGNAL internal: STD_LOGIC_VECTOR (n-1 DOWNTO 0);
13 BEGIN
14    PROCESS (clk, rst)
15    BEGIN
16       IF (rst='1') THEN
17          internal <= (OTHERS => '0');
18       ELSIF (clk'EVENT AND clk='1') THEN
19          internal <= d & internal(internal'LEFT DOWNTO 1);
20       END IF;
21    END PROCESS;
22    q <= internal(0);
23 END behavior;
24 ----------------------------------------------------
```

WAIT

- the PROCESS cannot have a sensitivity list when WAIT is employed

```
WAIT UNTIL signal_condition;
```

```
WAIT ON signal1 [, signal2, ... ];
```

```
WAIT FOR time;
```

WAIT (Example WAIT UNTIL)

Example: 8-bit register with synchronous reset.

```
PROCESS          -- no sensitivity list
BEGIN
   WAIT UNTIL (clk'EVENT AND clk='1');
   IF (rst='1') THEN
      output <= "00000000";
   ELSIF (clk'EVENT AND clk='1') THEN
      output <= input;
   END IF;
END PROCESS;
```

WAIT (Example WAIT ON)

Example: 8-bit register with asynchronous reset.

```
PROCESS
BEGIN
   WAIT ON clk, rst;
   IF (rst='1') THEN
      output <= "00000000";
   ELSIF (clk'EVENT AND clk='1') THEN
      output <= input;
   END IF;
END PROCESS;
```

WAIT (Example WAIT FOR)

- WAIT FOR is intended for simulation only (waveform generation for testbenches).
  - Example: WAIT FOR 5ns

WAIT (Example: DFF #2)

```
1  ---------------------------------------
2  LIBRARY ieee;
3  USE ieee.std_logic_1164.all;
4  ---------------------------------------
5  ENTITY dff IS
6     PORT (d, clk, rst: IN STD_LOGIC;
7           q: OUT STD_LOGIC);
8  END dff;
9  ---------------------------------------
10 ARCHITECTURE dff OF dff IS
11 BEGIN
12    PROCESS
13    BEGIN
14       WAIT ON rst, clk;
15       IF (rst='1') THEN
16          q <= '0';
17       ELSIF (clk'EVENT AND clk='1') THEN
18          q <= d;
19       END IF;
20    END PROCESS;
21 END dff;
22 ---------------------------------------
```

WAIT (Example: One-digit Counter #2)

```
1  ------------------------------------------
2  LIBRARY ieee;
3  USE ieee.std_logic_1164.all;
4  ------------------------------------------
5  ENTITY counter IS
6     PORT (clk : IN STD_LOGIC;
7           digit : OUT INTEGER RANGE 0 TO 9);
8  END counter;
9  ------------------------------------------
10 ARCHITECTURE counter OF counter IS
11 BEGIN
12    PROCESS          -- no sensitivity list
13       VARIABLE temp : INTEGER RANGE 0 TO 10;
14    BEGIN
15       WAIT UNTIL (clk'EVENT AND clk='1');
16       temp := temp + 1;
17       IF (temp=10) THEN temp := 0;
18       END IF;
19       digit <= temp;
20    END PROCESS;
21 END counter;
22 ------------------------------------------
```

CASE

```
CASE identifier IS
    WHEN value => assignments;
    WHEN value => assignments;
    ...
END CASE;
```

Example:

```
CASE control IS
   WHEN "00" => x<=a; y<=b;
   WHEN "01" => x<=b; y<=c;
   WHEN OTHERS => x<="0000"; y<="ZZZZ";
END CASE;
```

```
WHEN value                    -- single value
WHEN value1 to value2         -- range, for enumerated data types
                              -- only
WHEN value1 | value2 |...     -- value1 or value2 or ...
```

CASE (Example: DFF #3)

```
1  ------------------------------------------
2  ENTITY dff IS
3     PORT (d, clk, rst: IN BIT;
4           q: OUT BIT);
5  END dff;
6  ------------------------------------------
7  ARCHITECTURE dff3 OF dff IS
8  BEGIN
9     PROCESS (clk, rst)
10    BEGIN
11       CASE rst IS
12          WHEN '1' => q<='0';
13          WHEN '0' =>
14             IF (clk'EVENT AND clk='1') THEN
15                q <= d;
16             END IF;
17          WHEN OTHERS => NULL;     -- Unnecessary, rst is of type
18                                   -- BIT
19       END CASE;
20    END PROCESS;
21 END dff3;
22 ------------------------------------------
```

CASE (Two-digit Counter with SSD Output)

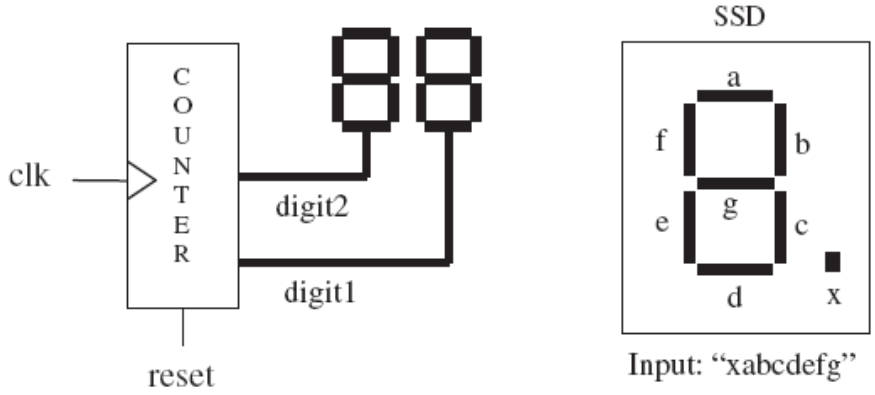


Figure 6.7
2-digit counter of example 6.7.

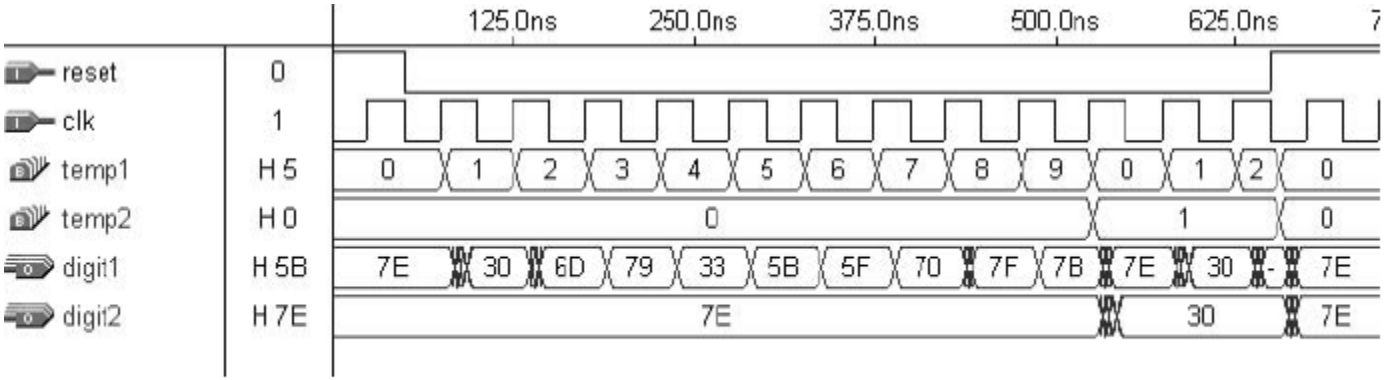


Figure 6.8
Simulation results of example 6.7.

CASE (Two-digit Counter with SSD Output) . . .

```
1  ----------------------------------------------------
2  LIBRARY ieee;
3  USE ieee.std_logic_1164.all;
4  ----------------------------------------------------
5  ENTITY counter IS
6     PORT (clk, reset : IN STD_LOGIC;
7           digit1, digit2 : OUT STD_LOGIC_VECTOR (6 DOWNTO 0));
8  END counter;
9  ----------------------------------------------------
```

CASE (Two-digit Counter with SSD Output) . . .

```
10 ARCHITECTURE counter OF counter IS
11 BEGIN
12    PROCESS(clk, reset)
13       VARIABLE temp1: INTEGER RANGE 0 TO 10;
14       VARIABLE temp2: INTEGER RANGE 0 TO 10;
15    BEGIN
16    ---- counter: ----------------------
17       IF (reset='1') THEN
18          temp1 := 0;
19          temp2 := 0;
20       ELSIF (clk'EVENT AND clk='1') THEN
21          temp1 := temp1 + 1;
22          IF (temp1=10) THEN
23             temp1 := 0;
24             temp2 := temp2 + 1;
25             IF (temp2=10) THEN
26                temp2 := 0;
27             END IF;
28          END IF;
29       END IF;
```

CASE (Two-digit Counter with SSD Output) . . .

```
30        ---- BCD to SSD conversion: --------
31        CASE temp1 IS
32           WHEN 0 => digit1 <= "1111110";    --7E
33           WHEN 1 => digit1 <= "0110000";    --30
34           WHEN 2 => digit1 <= "1101101";    --6D
35           WHEN 3 => digit1 <= "1111001";    --79
36           WHEN 4 => digit1 <= "0110011";    --33
37           WHEN 5 => digit1 <= "1011011";    --5B
38           WHEN 6 => digit1 <= "1011111";    --5F
39           WHEN 7 => digit1 <= "1110000";    --70
40           WHEN 8 => digit1 <= "1111111";    --7F
41           WHEN 9 => digit1 <= "1111011";    --7B
42           WHEN OTHERS => NULL;
43        END CASE;
```

CASE (Two-digit Counter with SSD Output) . . .

```
44        CASE temp2 IS
45           WHEN 0 => digit2 <= "1111110";     --7E
46           WHEN 1 => digit2 <= "0110000";     --30
47           WHEN 2 => digit2 <= "1101101";     --6D
48           WHEN 3 => digit2 <= "1111001";     --79
49           WHEN 4 => digit2 <= "0110011";     --33
50           WHEN 5 => digit2 <= "1011011";     --5B
51           WHEN 6 => digit2 <= "1011111";     --5F
52           WHEN 7 => digit2 <= "1110000";     --70
53           WHEN 8 => digit2 <= "1111111";     --7F
54           WHEN 9 => digit2 <= "1111011";     --7B
55           WHEN OTHERS => NULL;
56        END CASE;
57     END PROCESS;
58 END counter;
59 -------------------------------------------------
```

LOOP

- FOR / LOOP: The loop is repeated a fixed number of times.

```
[label:] FOR identifier IN range LOOP
  (sequential statements)
END LOOP [label];
```

Example of FOR / LOOP:

```
FOR i IN 0 TO 5 LOOP
   x(i) <= enable AND w(i+2);
   y(0, i) <= w(i);
END LOOP;
```

- WHILE / LOOP: The loop is repeated until a condition no longer holds.

```
[label:] WHILE condition LOOP
  (sequential statements)
END LOOP [label];
```

```
WHILE (i < 10) LOOP
   WAIT UNTIL clk'EVENT AND clk='1';
   (other statements)
END LOOP;
```

LOOP …

- EXIT: Used for ending the loop.

```
[label:] EXIT [label] [WHEN condition];
```

```
FOR i IN data'RANGE LOOP
   CASE data(i) IS
      WHEN '0' => count:=count+1;
      WHEN OTHERS => EXIT;
   END CASE;
END LOOP;
```

- NEXT: Used for skipping loop steps.

```
[label:] NEXT [loop_label] [WHEN condition];
```

```
FOR i IN 0 TO 15 LOOP
   NEXT WHEN i=skip;     -- jumps to next iteration
      (...)
END LOOP;
```

LOOP (Example: Carry Ripple Adder)

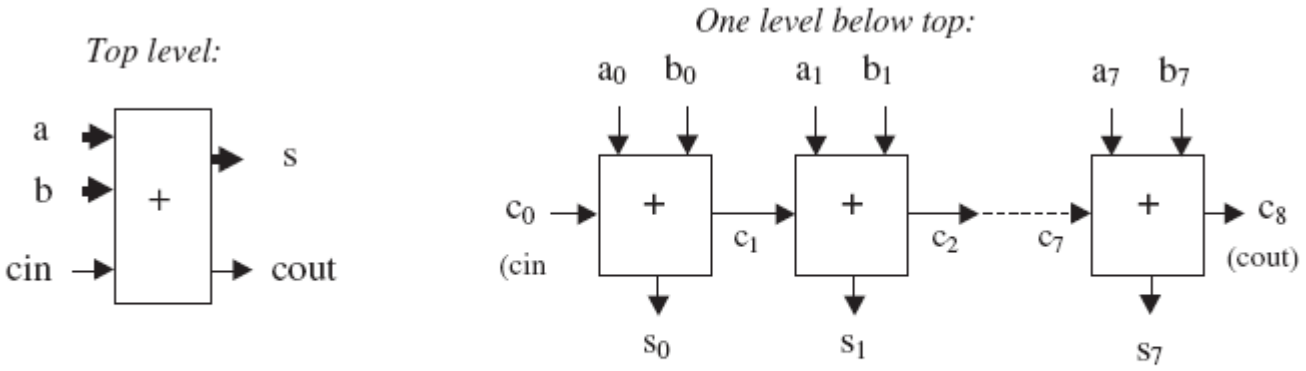


Figure 6.9
8-bit carry ripple adder of example 6.8

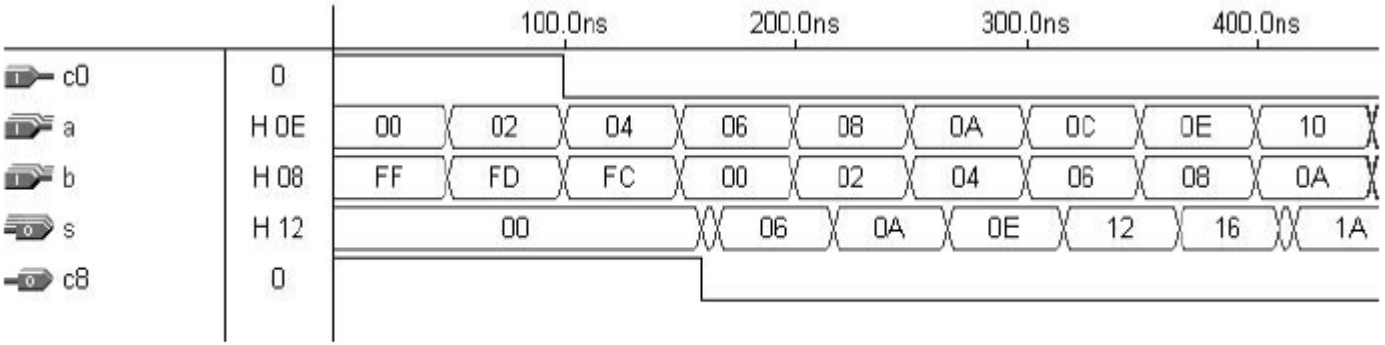


Figure 6.10
Simulation results of example 6.8.

LOOP (Example: Carry Ripple Adder) . . .

```
1  ----- Solution 1: Generic, with VECTORS --------
2  LIBRARY ieee;
3  USE ieee.std_logic_1164.all;
4  ---------------------------------------------
5  ENTITY adder IS
6     GENERIC (length : INTEGER := 8);
7     PORT ( a, b: IN STD_LOGIC_VECTOR (length-1 DOWNTO 0);
8            cin: IN STD_LOGIC;
9            s: OUT STD_LOGIC_VECTOR (length-1 DOWNTO 0);
10           cout: OUT STD_LOGIC);
11 END adder;
12 ---------------------------------------------
```

LOOP (Example: Carry Ripple Adder) . . .

```
13 ARCHITECTURE adder OF adder IS
14 BEGIN
15    PROCESS (a, b, cin)
16       VARIABLE carry : STD_LOGIC_VECTOR (length DOWNTO 0);
17    BEGIN
18       carry(0) := cin;
19       FOR i IN 0 TO length-1 LOOP
20          s(i) <= a(i) XOR b(i) XOR carry(i);
21          carry(i+1) := (a(i) AND b(i)) OR (a(i) AND
22                        carry(i)) OR (b(i) AND carry(i));
23       END LOOP;
24       cout <= carry(length);
25    END PROCESS;
26 END adder;
27 -----------------------------------------------
```

LOOP (Example: Carry Ripple Adder) . . .

```
1  ---- Solution 2: non-generic, with INTEGERS ----
2  LIBRARY ieee;
3  USE ieee.std_logic_1164.all;
4  ---------------------------------------------
5  ENTITY adder IS
6     PORT ( a, b: IN INTEGER RANGE 0 TO 255;
7            c0: IN STD_LOGIC;
8            s: OUT INTEGER RANGE 0 TO 255;
9            c8: OUT STD_LOGIC);
10 END adder;
11 ---------------------------------------------
```

LOOP (Example: Carry Ripple Adder) . . .

```
11 ------------------------------------------------
12 ARCHITECTURE adder OF adder IS
13 BEGIN
14    PROCESS (a, b, c0)
15       VARIABLE temp : INTEGER RANGE 0 TO 511;
16    BEGIN
17       IF (c0='1') THEN temp:=1;
18       ELSE temp:=0;
19       END IF;
20       temp := a + b + temp;
21       IF (temp > 255) THEN
22          c8 <= '1';
23          temp := temp---256;
24       ELSE c8 <= '0';
25       END IF;
26       s <= temp;
27    END PROCESS;
28 END adder;
29 ------------------------------------------------
```

LOOP (Simple Barrel Shifter)

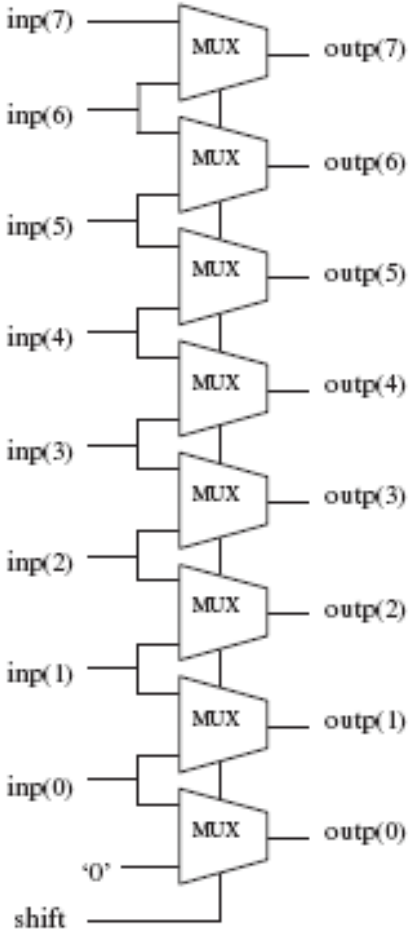


Figure 6.11
Simple barrel shifter of example 6.9.

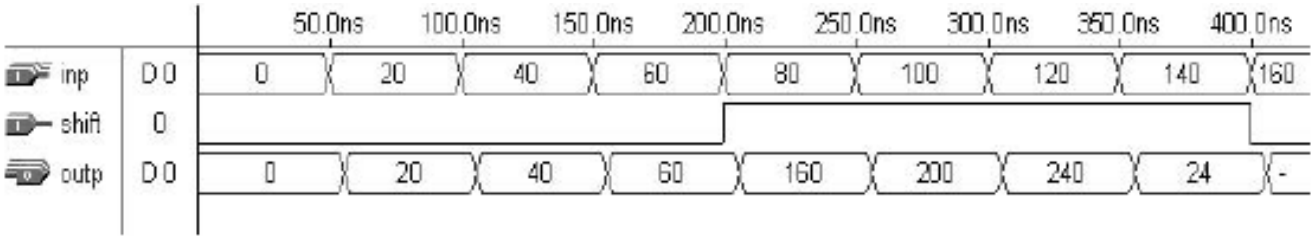


Figure 6.12
Simulation results of example 6.9.

LOOP (Simple Barrel Shifter) . . .

```
1  ----------------------------------------
2  LIBRARY ieee;
3  USE ieee.std_logic_1164.all;
4  ----------------------------------------
5  ENTITY barrel IS
6     GENERIC (n: INTEGER := 8);
7     PORT ( inp: IN STD_LOGIC_VECTOR (n-1 DOWNTO 0);
8            shift: IN INTEGER RANGE 0 TO 1;
9            outp: OUT STD_LOGIC_VECTOR (n-1 DOWNTO 0));
10 END barrel;
11 ----------------------------------------
12 ARCHITECTURE RTL OF barrel IS
13 BEGIN
14    PROCESS (inp, shift)
15    BEGIN
16       IF (shift=0) THEN
17          outp <= inp;
18       ELSE
19          outp(0) <= '0';
20          FOR i IN 1 TO inp'HIGH LOOP
21            outp(i) <= inp(i-1);
22          END LOOP;
23       END IF;
24    END PROCESS;
25 END RTL;
26 ----------------------------------------
```

LOOP (Example: Leading Zeros)

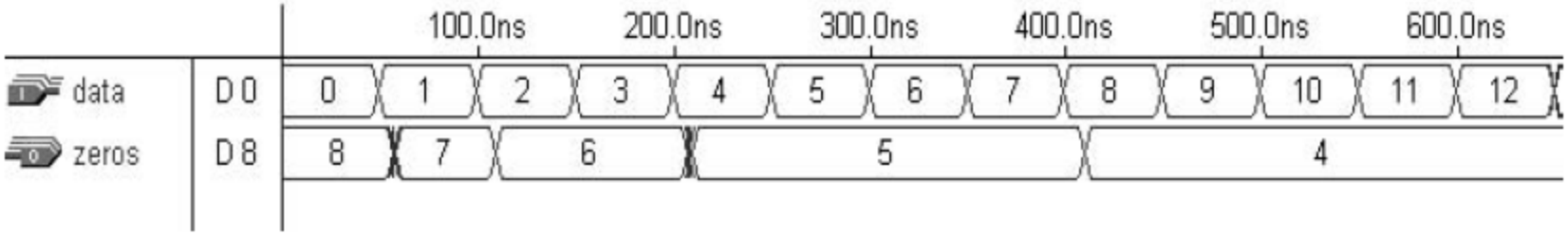


Figure 6.13
Simulation results of example 6.10.

```
1  ------------------------------------------
2  LIBRARY ieee;
3  USE ieee.std_logic_1164.all;
4  ------------------------------------------
5  ENTITY LeadingZeros IS
6     PORT ( data: IN STD_LOGIC_VECTOR (7 DOWNTO 0);
7            zeros: OUT INTEGER RANGE 0 TO 8);
8  END LeadingZeros;
9  ------------------------------------------
```

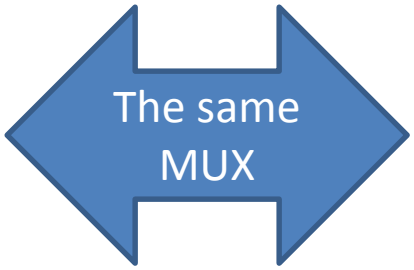
LOOP (Example: Leading Zeros) • • •

```
9  ----------------------------------------
10 ARCHITECTURE behavior OF LeadingZeros IS
11 BEGIN
12    PROCESS (data)
13       VARIABLE count: INTEGER RANGE 0 TO 8;
14    BEGIN
15       count := 0;
16       FOR i IN data'RANGE LOOP
17          CASE data(i) IS
18             WHEN '0' => count := count + 1;
19             WHEN OTHERS => EXIT;
20          END CASE;
21       END LOOP;
22       zeros <= count;
23    END PROCESS;
24 END behavior;
25 ----------------------------------------
```

CASE versus IF

- After optimization, the general tendency is for a circuit synthesized from a VHDL code based on IF not to differ from that based on CASE.

```
---- With IF: --------------
IF (sel="00") THEN x<=a;
ELSIF (sel="01") THEN x<=b;
ELSIF (sel="10") THEN x<=c;
ELSE x<=d;
```




```
---- With CASE: ------------
CASE sel IS
   WHEN "00" => x<=a;
   WHEN "01" => x<=b;
   WHEN "10" => x<=c;
   WHEN OTHERS => x<=d;
END CASE;
```

CASE versus WHEN

Table 6.1
Comparison between WHEN and CASE.

	WHEN	CASE
Statement type	Concurrent	Sequential
Usage	Only outside PROCESSES, FUNCTIONS, or PROCEDURES	Only inside PROCESSES, FUNCTIONS, or PROCEDURES
All permutations must be tested	Yes for WITH/SELECT/WHEN	Yes
Max. # of assignments per test	1	Any
No-action keyword	UNAFFECTED	NULL

```
---- With WHEN: ---------------
WITH sel SELECT
   x <=    a WHEN "000",
           b WHEN "001",
           c WHEN "010",
           UNAFFECTED WHEN OTHERS;
```

```
---- With CASE: ---------------
CASE sel IS
    WHEN "000" => x<=a;
    WHEN "001" => x<=b;
    WHEN "010" => x<=c;
    WHEN OTHERS => NULL;
END CASE;
```

Bad Clocking

```
PROCESS (clk)
BEGIN
   IF(clk'EVENT AND clk='1') THEN
      counter <= counter + 1;
   ELSIF(clk'EVENT AND clk='0') THEN
      counter <= counter + 1;
   END IF;
   ...
END PROCESS;
```

```
PROCESS (clk)
BEGIN
   IF(clk'EVENT) THEN
      counter := counter + 1;
   END IF;
   ...
END PROCESS;
```

```
PROCESS (clk)
BEGIN
   counter := counter + 1;
   ...
END PROCESS;
```

Example : RAM

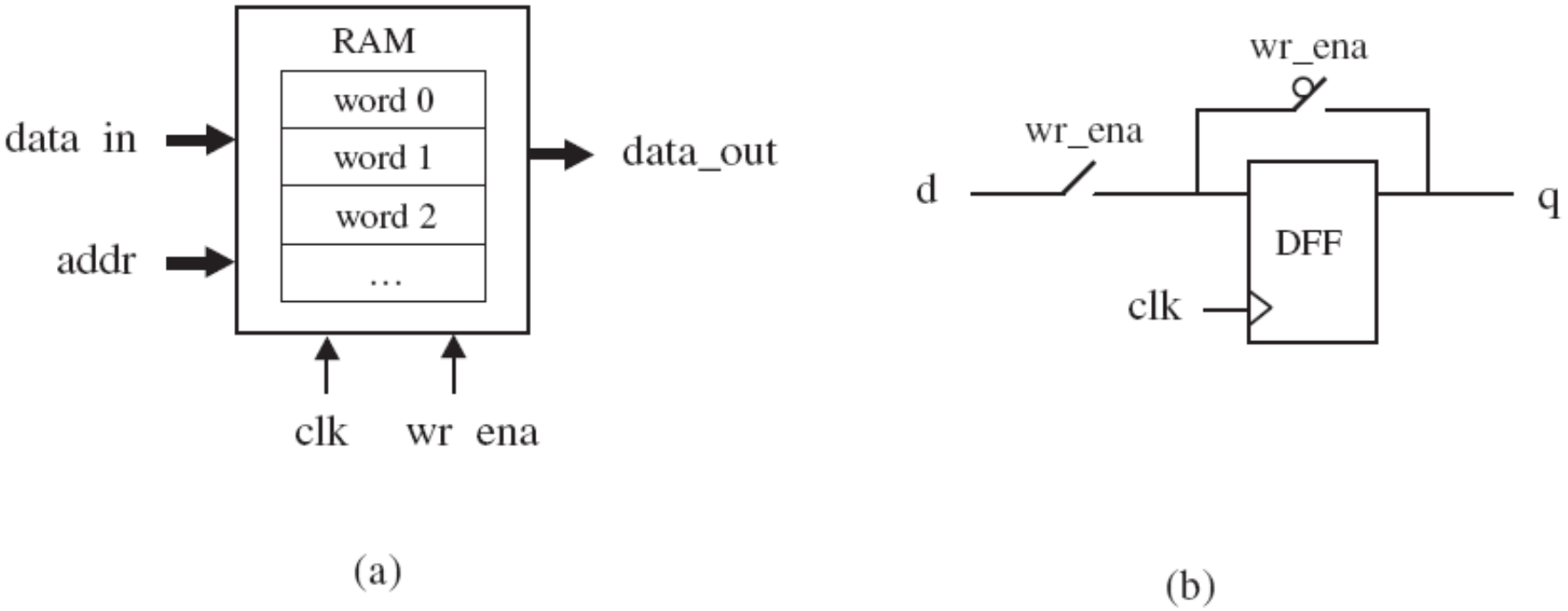


Figure 6.14
RAM circuit of example 6.11.

Example : RAM …

```
1  ------------------------------------------------
2  LIBRARY ieee;
3  USE ieee.std_logic_1164.all;
4  ------------------------------------------------
5  ENTITY ram IS
6  GENERIC ( bits: INTEGER := 8;      -- # of bits per word
7            words: INTEGER := 16);   -- # of words in the memory
8     PORT ( wr_ena, clk: IN STD_LOGIC;
9            addr: IN INTEGER RANGE 0 TO words-1;
10           data_in: IN STD_LOGIC_VECTOR (bits-1 DOWNTO 0);
11           data_out: OUT STD_LOGIC_VECTOR (bits-1 DOWNTO 0));
12 END ram;
13 ------------------------------------------------
```

Example : RAM …

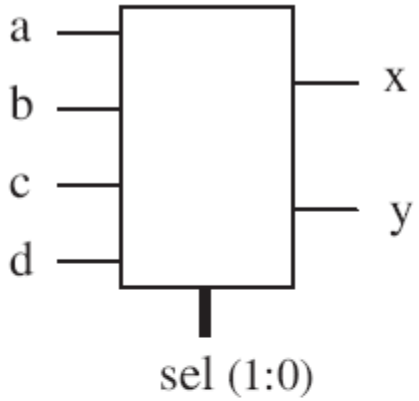
```
14 ARCHITECTURE ram OF ram IS
15    TYPE vector_array IS ARRAY (0 TO words-1) OF
16       STD_LOGIC_VECTOR (bits-1 DOWNTO 0);
17    SIGNAL memory: vector_array;
18 BEGIN
19    PROCESS (clk, wr_ena)
20    BEGIN
21       IF (wr_ena='1') THEN
22          IF (clk'EVENT AND clk='1') THEN
23             memory(addr) <= data_in;
24          END IF;
25       END IF;
26    END PROCESS;
27    data_out <= memory(addr);
28 END ram;
29 ---------------------------------------------------
```

Using Sequential Code to Design Combinational Circuits

- Rule 1: Make sure that all input signals used (read) in the PROCESS appear in its sensitivity list.

- Rule 2: Make sure that all combinations of the input/output signals are included in the code.

Example : Bad Combinational Design



sel	x	y
00	a	0
01	b	1
10	c	
11	d	

sel	x	y
00	a	0
01	b	1
10	c	y
11	d	y

sel	x	y
00	a	0
01	b	1
10	c	X
11	d	X

(a) (b) (c) (d)

Figure 6.16
Circuit of example 6.12: (a) top-level diagram, (b) specifications provided, (c) implemented truth-table, and (d) the right approach.

Example : Bad Combinational Design

```
1  ----------------------------------------
2  LIBRARY ieee;
3  USE ieee.std_logic_1164.all;
4  ----------------------------------------
5  ENTITY example IS
6     PORT (a, b, c, d: IN STD_LOGIC;
7        sel: IN INTEGER RANGE 0 TO 3;
8        x, y: OUT STD_LOGIC);
9  END example;
10 ----------------------------------------
```

Example : Bad Combinational Design

```
11 ARCHITECTURE example OF example IS
12 BEGIN
13     PROCESS (a, b, c, d, sel)
14     BEGIN
15         IF (sel=0) THEN
16             x<=a;
17             y<='0';
18         ELSIF (sel=1) THEN
19             x<=b;
20             y<='1';
21         ELSIF (sel=2) THEN
22             x<=c;
23         ELSE
24             x<=d;
25         END IF;
26     END PROCESS;
27 END example;
28 ---------------------------------------
```

Thanks for your attention

- Don’t forget

Problems!!!

Next session

7 Signals and Variables

- 7.1 CONSTANT
- 7.2 SIGNAL
- 7.3 VARIABLE
- 7.4 SIGNAL versus VARIABLE
- 7.5 Number of Registers
- 7.6 Problems