دانشگاه اصفهان

دانشکده مهندسی کامپیوتر

درس طراحی کامپیوتری سیستمهای دیجیتال

فصل هفتم : SIGNAL and VARIABLE

Signal

- SIGNAL serves to pass values in and out of the circuit, as well as between its internal units
- In other words, a signal represents circuit interconnects (wires)
- For example, all ports of an entity are signals

```
SIGNAL signal_name: signal_type [range] [:= default_value];
```

```
-----------------------------------------
SIGNAL flag: STD_LOGIC := '0';
SIGNAL address: NATURAL RANGE 0 TO 2**N-1;
SIGNAL data: BIT_VECTOR(15 DOWNTO 0);
-----------------------------------------
```

Resolved and Guarded Signal

```
SIGNAL signal_name: resolution_function signal_type [:= def_value];
```

```
SIGNAL x: my_resolution_function my_data_type;
```

```
SIGNAL signal_name: ... signal_type [REGISTER | BUS] [:= def_value];
```

Variable

- VARIABLE is a very valuable object for use in sequential code
- Because it is visible only inside the sequential unit in which it was created, it represents just local information
- The only exception is shared variables

```
VARIABLE variable_name: variable_type [range] [:= default_value];
```

```
-----------------------------------------
VARIABLE flag: STD_LOGIC := '0';
VARIABLE address: NATURAL RANGE 0 TO 2**N-1;
VARIABLE data: BIT_VECTOR(15 DOWNTO 0);
-----------------------------------------
```

Shared Variable

- When declared as shared, a variable can be accessed by more than one sequential code and also by concurrent code
- Only one sequential unit should modify its value
- Additionally, the value of a shared variable can be passed to a signal in an assignment made outside the sequential code
- A shared variable can be declared in ENTITY, ARCHITECTURE, GENERATE, and PACKAGE

Counter with SHARED VARIABLE

- Note that each variable is modified by only one process

- passing of their values to signals can be done outside the processes

```
1   ------------------------------------------------
2   ENTITY counter_with_sharedvar IS
3      PORT (clk: IN BIT;
4            digit1, digit2: OUT INTEGER RANGE 0 TO 9);
5   END ENTITY;
6   ------------------------------------------------
7   ARCHITECTURE counter OF counter_with_sharedvar IS
8      SHARED VARIABLE temp1, temp2: INTEGER RANGE 0 TO 9;
9   BEGIN
10     ----------------------------------
11     proc1: PROCESS (clk)
```

```
12     BEGIN
13        IF (clk'EVENT AND clk='1') THEN
14           IF (temp1=9) THEN
15              temp1 := 0;
16           ELSE
17              temp1 := temp1 + 1;
18           END IF;
19        END IF;
20     END PROCESS proc1;
21     ---------------------------------
22     proc2: PROCESS (clk)
23     BEGIN
24        IF (clk'EVENT AND clk='1') THEN
25           IF (temp1=9) THEN
26              IF (temp2=9) THEN
27                 temp2 := 0;
28              ELSE
29                 temp2 := temp2 + 1;
30              END IF;
31           END IF;
32        END IF;
33     END PROCESS proc2;
34     ---------------------------------
35     digit1 <= temp1;
36     digit2 <= temp2;
37  END ARCHITECTURE;
38  -------------------------------------------------
```

SIGNAL versus VARIABLE

Rule	SIGNAL	VARIABLE
1. Local of declaration	ENTITY, ARCHITECTURE, BLOCK, GENERATE, and PACKAGE (declaration in sequential code is forbidden)	Only in sequential units (PROCESS and subprograms), except shared variables (declared in ENTITY, ARCHITECTURE, BLOCK, GENERATE, PACKAGE)
2. Scope (local of use and of modification)	Can be global (used and modified anywhere in the code)	Local (used and modified only inside its own sequential unit), except shared variables (can be global, but modified by only one sequential unit)
3. Update	New value available only at the end of the current cycle	Updated immediately (new value ready to be used in the next line of code)
4. Assignment operator	Values are assigned using "<=" Example: sig<=5;	Values are assigned using ":=" Example: var:=5;
5. Multiple assignments	Only one assignment is allowed	Multiple assignments are fine (because update is immediate)
6. Inference of registers	Flip-flops are inferred when an assignment to a signal occurs at the transition of another signal	Flip-flops are inferred when an assignment to a variable occurs at the transition of a signal and this variable's value eventually affects a signal's value

SIGNAL versus VARIABLE Usage

```
1  ...
2  ----------------------------------------
3  ARCHITECTURE example OF example IS
4     SIGNAL sig: INTEGER RANGE -8 TO 7;
5  BEGIN
6     PROCESS (clock)
7        VARIABLE var INTEGER RANGE -8 TO 7;
8     BEGIN
9        sig <= 0;
10       var := 0;
11       IF (clock'EVENT AND clock='1') THEN
12          sig <= sig + 1;
13          var := var + 1;
14          IF (sig=a) THEN ...
15          ELSIF (var=b) THEN ...
16          END IF;
17       END IF;
18       ...
19    END PROCESS;
20 END example:
21 ----------------------------------------
```

Rule 1: Was not violated because both sig and var were declared in right places (lines 4 and 7).

Rule 2: Is fine because var was used only inside the process (sig can be used anywhere in the architecture code).

Rule 3: The assignment in line 12 increments sig, but the new value will only be ready at the conclusion of the present process, so line 14 indeed compares a to an outdated value of sig

Rule 4: Was not violated because all assignments to sig and var (lines 9–10 and 12–13) employed the proper operators.

Rule 5: Was violated because lines 9 and 12 together make multiple assignments to sig. However, because this is a sequential code, such repetitions might be accepted by the compiler, but only the last assignment will survive, thus producing an incorrect circuit. Regarding var, no violation occurred because for variables multiple assignments (lines 10, 13) are fine

Counters with SIGNAL and VARIABLE

```
1  ------------------------------------------------
2  ENTITY counter IS
3     PORT (clk: IN BIT;
4           count1, count2: OUT INTEGER RANGE 0 TO 9;
5  END ENTITY;
6  ------------------------------------------------
7  ARCHITECTURE dual_counter OF counter IS
8     SIGNAL temp1: INTEGER RANGE 0 TO 10;
9  BEGIN
10    -----counter 1: with signal:-----
11    with_sig: PROCESS(clk)
12    BEGIN
13       IF (clk'EVENT AND clk='1') THEN
14          temp1 <= temp1 + 1;
15          IF (temp1=10) THEN
16             temp1 <= 0;
17          END IF;
18       END IF;
19       count1 <= temp1;
20    END PROCESS with_sig;
```

```
21      -----counter 2: with variable:-----
22      with_var: PROCESS(clk)
23         VARIABLE temp2: INTEGER RANGE 0 TO 10;
24      BEGIN
25         IF (clk'EVENT AND clk='1') THEN
26            temp2 := temp2 + 1;
27            IF (temp2=10) THEN
28               temp2 := 0;
29            END IF;
30         END IF;
31         count2 <= temp2;
32      END PROCESS with_var;
33  END ARCHITECTURE;
34  ------------------------------------------------
```

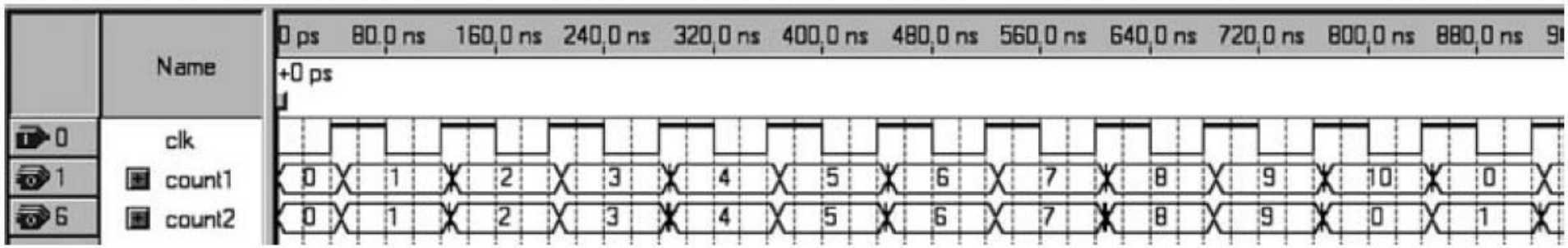


Figure 7.2
Simulation results from the counters of example 7.3.

The Inference of Registers

```
-----------------------------------
SIGNAL clk: BIT;
SIGNAL sig1: BIT_VECTOR(7 DOWNTO 0);
SIGNAL sig2: INTEGER RANGE 0 TO 7;
VARIABLE var: BIT_VECTOR(3 DOWNTO 0);
-----------------------------------
```

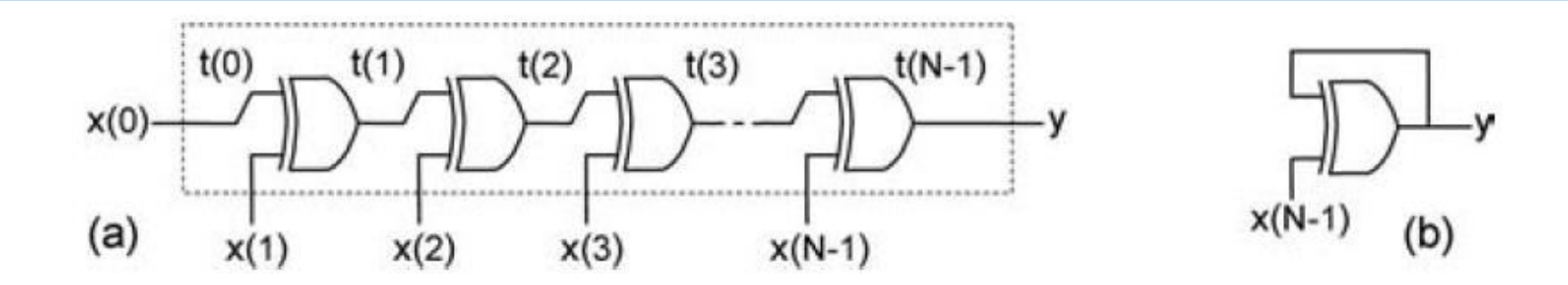
```
-------------------------------
IF (clk'EVENT AND clk='1') THEN
   sig1 <= x;
   sig2 <= y;
   var := z;
END IF;
-------------------------------
```

Then in the code below all three objects would be registered (stored) because assignments are made to all three at the transition of another signal (*clk*). A total of $8 + 3 + 4 = 15$ DFFs would then be inferred.

In the next code, only *sig2* and *var* would be stored ($3 + 4 = 7$ DFFs).

```
---------------------------------
PROCESS (clk)
BEGIN
   IF (clk'EVENT AND clk='1') THEN
       sig2 <= y;
       var := z;
   END IF; ...
   sig1 <= x;
END PROCESS;
---------------------------------
```

Making Multiple Signal Assignments



Parity detector

Making Multiple Signal Assignments

```
1  ----------------------------------------
2  ENTITY parity_det IS
3    GENERIC (N: POSITIVE := 8);
4    PORT (x: IN BIT_VECTOR(N-1 DOWNTO 0);
5          y: OUT BIT);
6  END ENTITY;
7  ----------------------------------------
8  ARCHITECTURE not_ok OF parity_det IS
9     SIGNAL temp: BIT;
10 BEGIN
11    temp <= x(0);
12    gen: FOR i IN 1 TO N-1 GENERATE
13       temp <= temp XOR x(i);
14    END GENERATE;
15    y <= temp;
16 END ARCHITECTURE;
17 ----------------------------------------
```

```
7  ----------------------------
8  ARCHITECTURE not_ok OF ...
9    SIGNAL temp: BIT;
10 BEGIN
11   PROCESS (x)
12   BEGIN
13     temp <= x(0);
14     FOR i IN 1 TO N-1 LOOP
15       temp <= temp XOR x(i);
16     END LOOP;
17     y <= temp;
18   END PROCESS;
19 END ARCHITECTURE;
20 ----------------------------
```

```
 7  ----------------------------------------
 8  ARCHITECTURE ok OF parity_det IS
 9     SIGNAL temp: BIT_VECTOR(N-1 DOWNTO 0);
10  BEGIN
11     temp(0) <= x(0);
12     gen: FOR i IN 1 TO N-1 GENERATE
13        temp(i) <= temp(i-1) XOR x(i);
14     END GENERATE;
15     y <= temp(N-1);
16  END ARCHITECTURE;
17  ----------------------------------------
```

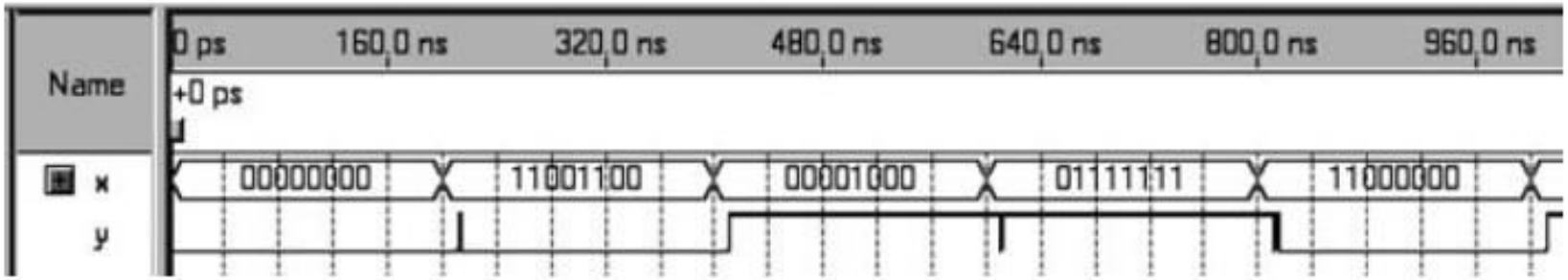


Figure 7.7
Simulation results from the parity detector of figure 7.6.