Filename="ch10.doc"

10.0 LOOPING Constructs

10.1 LOOP Statement

There are three types of loops: FOR, WHILE, and LOOP-EXIT. Autologic VHDL only supports theFOR loop construct.

10.1.1 FOR Loop

- Format:
  [label:] FOR loop_parameter IN discrete_range
  LOOP
  --sequential_statements
  END LOOP [label];
- Sequential statements are executed once for each value in loop parameter's range.
- Loop parameter is implicitly declared and may not be modified within loop or used outside loop.
- Implicitly declared means there does not have to be a variable declaration. The discrete_range can be subrange of integer or enumeration type.

Example:

```
PROCESS(signal_a)
BEGIN
label_1: FOR index IN 0 TO 7 -- index is implicitly declared
      LOOP
        ray_out(index) <= ray_in(index);
      END LOOP label_1;
END PROCESS;
```

10.1.2 WHILE Loop

- Format:
  [label]: WHILE boolean_expression
  LOOP
  -- sequential_statements
  END LOOP [label];
- boolean_expression is evaluated before each repetition of loop.

Example:

```
P1: PROCESS(signal_a)
    VARIABLE index:INTEGER:=0;
  BEGIN
    while_loop:
    WHILE index < 8 LOOP
      ray_out(index) <= ray_in(index);
      index:=index+1;
    END LOOP while_loop;
  END PROCESS P1;
```

- Loop parameter is explicitly declared and can be any scalar type.
- If the loop parameter index in the above example is of type real, the corresponding boolean expression would be Index < 8.0 and the increment would have to be index:=index +1.0;

10.1.3 LOOP-EXIT Construct

- Format:
```
LOOP
  -- sequential_statements
  -- including decision_to_quit
END LOOP;
```
- Does not contain an iteration scheme specifying how loop terminates.
- A decision to leave loop must be made from within loop.
- Can be terminated from within by: EXIT, NEXT

10.1.4 EXIT statement

- Used to break out of loops
- Only occurs whithin loops
- Format:
```
EXIT;
EXIT loop_label;
EXIT WHEN boolean_expression;
EXIT loop_label WHEN boolean_expression
```
- EXIT command takes effect as soon as it is executed. Statements which appear after EXIT are not executed.

Example:

```
show_loops:
PROCESS(s)
  VARIABLE sum,cnt:INTEGER :=0;
BEGIN
  sum:=0; cnt:=0;
  first:LOOP
        cnt:=cnt+1;
        sum:=sum+cnt;
        EXIT WHEN sum > 100;
       END LOOP;
  second:LOOP
         cnt:=cnt+1;
         sum:=sum+cnt;
         IF cnt>100 THEN EXIT;
         END IF;
        END LOOP;
END PROCESS show_loops;
```

NOTE: It is necessary to reset the variable sum and cnt to zero, even if their initial value is zero. Since, the initial value is not a reset value. It occurs only the first time the loop executed. Thereafter, each time the process is entered the value of the variable will be equal to the value assigned to the variable the last time the process was exited.

10.1.5 NEXT statement

- Causes completion of one of the iterations of an enclosing loop.
- Format:
  NEXT;
  NEXT loop_label;
  NEXT WHEN boolean_expression;
  NEXT loop_label WHEN boolean_expression;

Example:

```
next_xmpl:
PROCESS(s)
  VARIABLE temp,j:INTEGER:=0;
BEGIN
  temp:=0;j:=0;
  a_loop:FOR j IN 0 TO 7 LOOP
         j:=j+1;
         IF j>5 THEN
           NEXT a_loop;
         END IF;
         temp:=temp+1;
       END LOOP a_loop;
 END PROCESS next_xmpl;
```

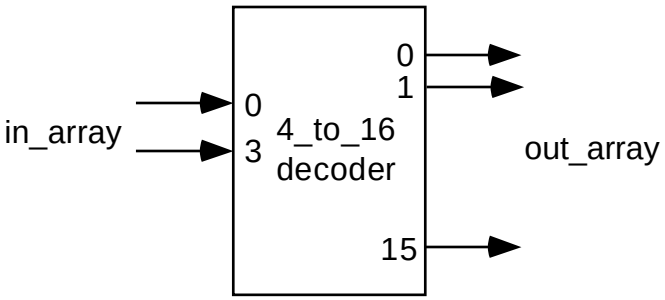
- NEXT without loop label causes the next iteration of the loop it is contained in to be executed. Control still remains within the loop's iteration scheme.
- Loop label can refer to the label on any loop enclosing the NEXT statement.
- The NEXT statement causes completion of the current iteration. Statements which follow the NEXT are not executed. Control is transferred to the beginning of the loop and the next iteration is started.

Example: a four-to-sixteen decoder can easily be modeled and synthesize using the FOR loop.

```
1       PACKAGE my_intgr IS
2               SUBTYPE my_int IS INTEGER RANGE 0 TO 1;
3       END my_intgr;
4
5       LIBRARY IEEE, ARITHMETIC;
6       USE IEEE.STD_LOGIC_1164.ALL;
7       USE ARITHMETIC.STD_LOGIC_ARITH.ALL;
8       USE WORK.my_intgr.ALL;
9
10      ENTITY dec4_16 IS
11              PORT(in_array: IN ARRAY(0 TO 3) OF my_int;
12                      out_array: OUT ARRAY(0 TO 15) OF my_int);
13      END dec4_16;
14
15      ARCHITECTURE archdec4_16 OF dec4_16 IS
16      BEGIN
17              PROCESS(in_array)
18                      VARIABLE index: INTEGER RANGE 0 TO 15;
19              BEGIN
20                      index :=0;
21                      FOR  i IN in_array'RANGE LOOP
22                              index := index + (2**i)*in_array(i);
```

```
23                      END LOOP;
24
25                      FOR j IN 0 TO 15 LOOP
26                              out_array(j) <=0;
27                      END LOOP;
28                      out_array(index) <= 1;
29              END PROCESS;
30      END archdec4_16;
```

NOTE: the use in_array'RANGE attribute to determine the index range.
in_array is an array of type integer with range 0 to 1.



10.2.1 Variable Use With Conditionals

When assigning to variables with conditional assignments, you must ensure that the variable is always assigned to, otherwise AutoLogic VHDL stops during compilation with an unsynthesizeable internal state error. AutoLogic VHDL does not build transparent latches for variables.

```
1       LIBRARY IEEE;
2       USE IEEE.STD_LOGIC_1164.ALL;
3       ENTITY varx IS
4               PORT(in :IN STD_LOGIC;
5                       out : OUT STD_LOGIC);
6       END varx;
7       ARCHITECTURE archvarxOF varx IS
8       BEGIN
9               PROCESS(in)
10                      VARIABLE x:STD_LOGIC;
11              BEGIN
12                      IF(in='1') THEN
13                              x:='1';
14                      ELSIF (in='0') THEN
15                              x:='0';
16                      END IF;
17                      out <=x;
16              END PROCESS;
17      END archvarx;
```

Note x is a STD_LOGIC variable, which has nine possible values. Seven values were not accounted for. Since AutoLogic VHDL does not build transparent latches for variables. The above code cannot be synthesized

```
1   LIBRARY IEEE;
2   USE IEEE.STD_LOGIC_1164.ALL;
3   ENTITY varx IS
4       PORT(in :IN STD_LOGIC;
5           out : OUT STD_LOGIC);
6   END varx;
7   ARCHITECTURE archvarxOF varx IS
8   BEGIN
9       PROCESS(in)
10          VARIABLE x:STD_LOGIC;
11      BEGIN
12          IF(in='1') THEN
13              x:='1';
14          ELSE
15              x:='0';
16          END IF;
17          out <=x;
18      END PROCESS;
19  END archvarx;
```

NOTE x is now always defined. This code is now synthesizeable.

If replaced variable x to signal x .

```
1   LIBRARY IEEE;
2   USE IEEE.STD_LOGIC_1164.ALL;
3   ENTITY sigx IS
4       PORT(in :IN STD_LOGIC;
5           out : OUT STD_LOGIC);
6   END sigx;
7   ARCHITECTURE archsigxOF sigx IS
8       SIGNAL x : STD_LOGIC;
9   BEGIN
10      PROCESS(in)
11      BEGIN
12          IF(in='1') THEN
13              x<='1';
14          ELSE
15              x<='0';
16          END IF;
17          out <=x;
18      END PROCESS;
19  END archvarx;
```

In line 17 if x is a signal, it must also appears in the process sensitivity list, otherwise the new value of x will not be assigned to out. NOTE a variable cannot appears in the process sensitivity list. The above code is still not synthesizeable.

```
1       LIBRARY IEEE;
2       USE IEEE.STD_LOGIC_1164.ALL;
3       ENTITY sigx IS
4               PORT(in :IN STD_LOGIC;
5                       out : OUT STD_LOGIC);
6       END sigx;
9       ARCHITECTURE archvarxOF varx IS
10              SIGNAL x : STD_LOGIC;
9       BEGIN
10              PROCESS(in, x)
11              BEGIN
12                      IF(in='1') THEN
13                              x<='1';
14                      ELSE
15                              x<='0';
16                      END IF;
17                      out <=x;
18              END PROCESS;
19      END archvarx;
```

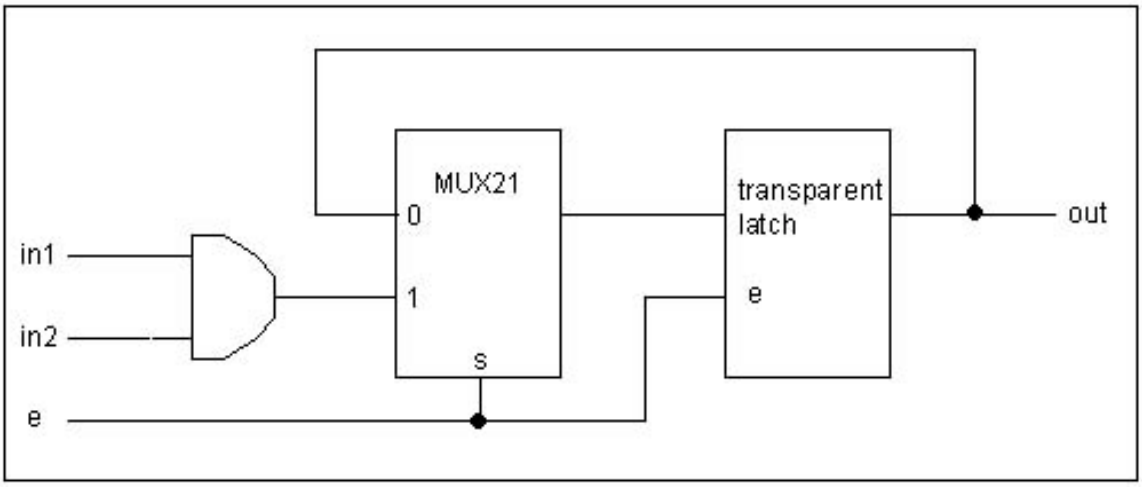
This is now synthesizeable.

10.2.2 Signal Use with Conditionals

AutoLogic VHDL does not support processes where a signal requiring a latch is used on the right-hand-side of an expression or assignment. This restriction is removed if the signal appears in the sensitivity list.

```
1       LIBRARY IEEE;
2       USE IEEE.STD_LOGIC_1164.ALL;
3       ENTITY signalx IS
4               PORT(e, in1, in2 :IN STD_LOGIC;
5                       out : OUT STD_LOGIC);
6       END signalx;
7       ARCHITECTURE archsignalx OF signalx IS
8               SIGNAL temp: STD_LOGIC;
9       BEGIN
10              PROCESS(e, in1, in2, temp)
11                      VARIABLE x:STD_LOGIC;
12              BEGIN
13                      IF(e='1') THEN
14                              temp<= in1 AND in2;
15                      END IF;
16                      out <= temp;
17              END PROCESS;
18      END archsignalx;
```

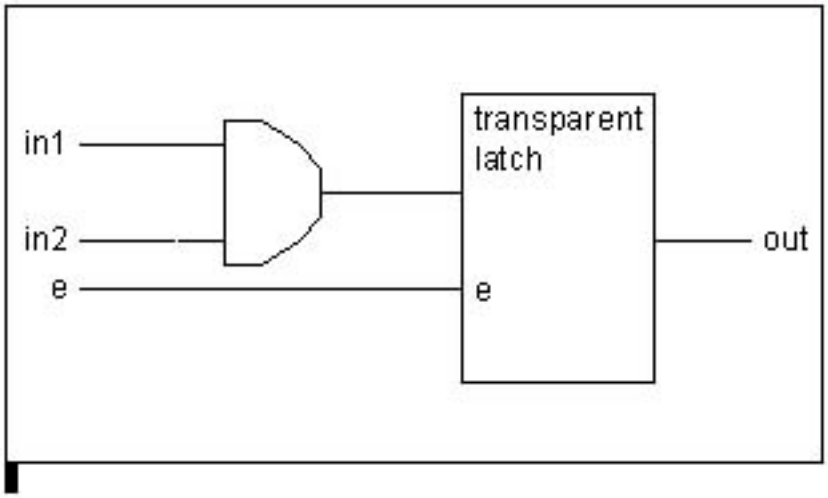
The value of temp is only updated whenever e='1', implies that the signal out in line 16 must remenber the previous value of temp. This code will synthesize temp with feedback through a tarnsparent latch.



If the feedback is not desired, and what you really want is to latch the result of the and operation, then the assignment to the signal out should be done outside the process using a concurrent signal assignment. The modified code is given below:

```
1       LIBRARY IEEE;
2       USE IEEE.STD_LOGIC_1164.ALL;
3       ENTITY signalx IS
4               PORT(e, in1, in2 :IN STD_LOGIC;
5                       out : OUT STD_LOGIC);
6       END signalx;
7       ARCHITECTURE archsignalx OF signalx IS
8               SIGNAL temp: STD_LOGIC;
9       BEGIN
10              PROCESS(e, in1, in2, temp)
11                      VARIABLE x:STD_LOGIC;
12              BEGIN
13                      IF(e='1') THEN
14                              temp<= in1 AND in2;
15                      END IF;
16              END PROCESS;
17              out <= temp;
18      END archsignalx;
```

The synthesized result is :


in1
in2
e
transparent
latch
e
out