VHDL:

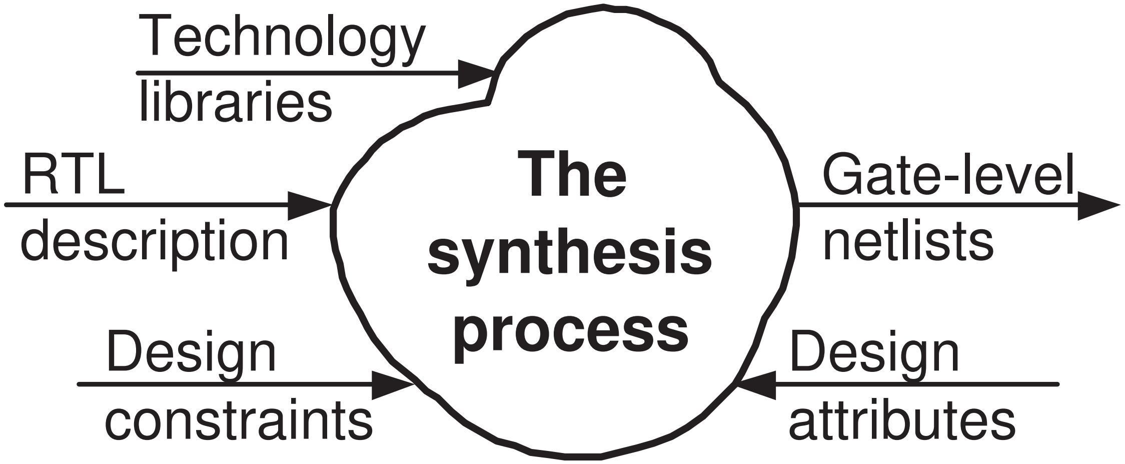
RTL Synthesis Basics

Goals

- To learn the basics of RTL synthesis.
- To be able to synthesize a digital system, given its VHDL model.
- To be able to relate VHDL code to its synthesized output.
- To appreciate the fact that synthesis is very sensitive to how the VHDL code is written!
- To learn how to write VHDL code that is efficient at synthesis.
- To get an idea about the synthesis of integer subtypes.
- To understand the synthesis of combinational logic.
- To understand the use of VHDL in synthesizing sequential logic.
- To get familiar with synthesizable process templates.
- To learn the cases when latches and flip-flops are inferred.
- To know how tristate elements are inferred.

RTL Synthesis

- Synthesis is an automatic method of converting a higher level of abstraction (RTL) to a lower level of abstraction (gate level netlists).
- Synthesis produces technology-specific implementation from technology-independent VHDL description.
- Not all VHDL can be used for synthesis. There are the VHDL subset for synthesis and synthesis style description.
- Synthesis is very sensitive to how the VHDL code is written!
- Gate level netlists are optimized for area, speed, power, or testability.

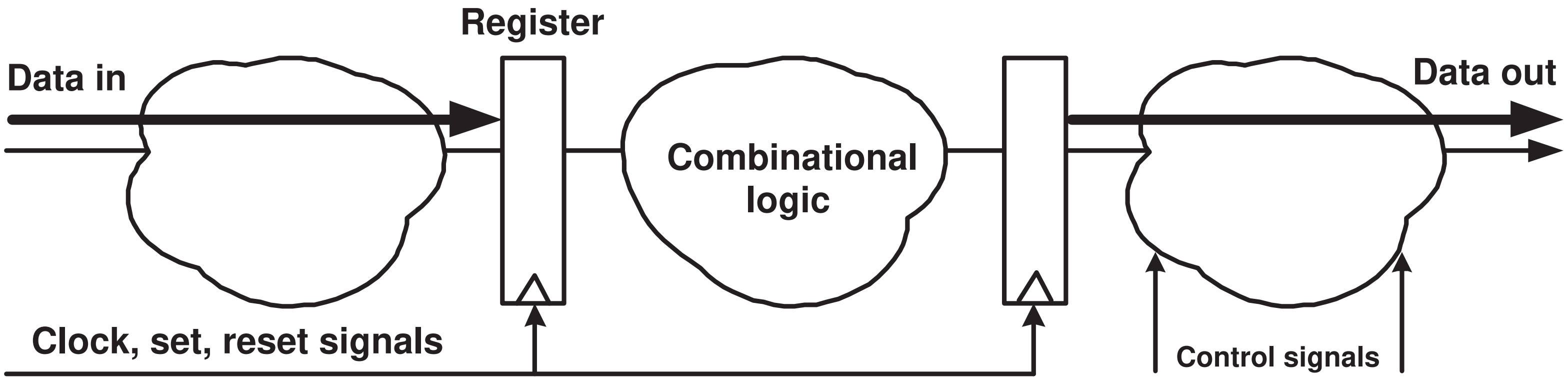


RTL Synthesis (Cont.)

- The essence of RTL is that you define all the registers in the system and the transfers between those registers that occur on each clock cycle (i.e., the combinational logic).
- Registers are described either:
  1. explicitly through component instantiation, or
  2. implicitly through inference (following a few simple rules)
- Combinational logic is described by: sequential control statements, subprograms, or concurrent statements.
- RTL synthesis describes the behavior of system at each clock period.
- RTL synthesis is used for synchronous design.

RTL Synthesis (Cont.)

- An RTL architecture can be divided into a datapath and control:
  - **Datapath:** performs operations on data (i.e., ALU),
  - **Control:** tells datapath, memory, etc. what to do.
- The datapath flows left to right, whereas the control flows bottom up.
- **Critical path:** critical timing path in a logical circuit is the longest synthesizable path from primary inputs to primary outputs.
- The critical path in the circuit limits the maximum clock frequency.
- Synthesis tools examine all signal paths and spot the critical path.



RTL Synthesis (Cont.)

It is recommended to use RTL code whenever possible as it provides the following:

1. Readable code
2. Ability to use same code for synthesis and simulation
3. Portable code for migration to different technologies
4. Resusable code for later designs

Synthesis of Integer Subtypes

- Number of bits in hardware depends on the object type.
- Number of bits required to represent integer subtypes depends on the largest value of the type.
- If an integer has negative values, 2’s complement format is used.

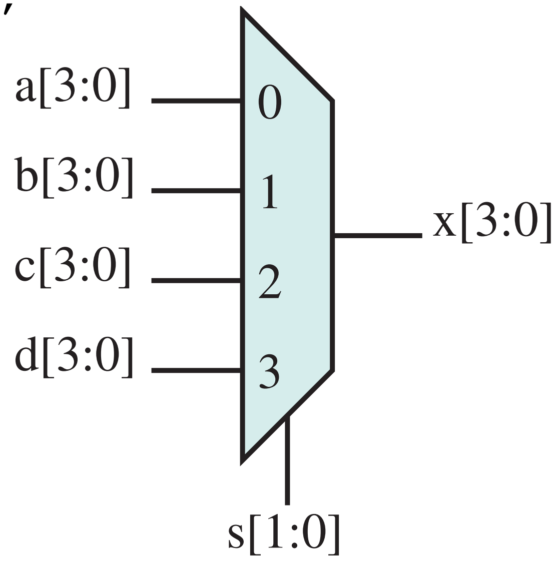
Example: 1 (Class) *Show how each of the types below are synthesized.*

```
SUBTYPE int_1 IS integer RANGE 0 TO 15;
SUBTYPE int_2 IS integer RANGE 10 TO 31;
SUBTYPE int_3 IS positive RANGE 25 DOWNTO 5;
TYPE int_4 IS RANGE -128 TO +15;

SIGNAL a:  int_1;       -- 4 bits unsigned integer
SIGNAL b:  int_2;       -- 5 bits unsigned integer
SIGNAL c:  int_3;       -- 5 bits unsigned integer
SIGNAL d:  int_4;       -- 8 bits two’s complement integer
SIGNAL e:  integer;     -- 32 bits two’s complement integer
```

Synthesis of Signal Assignment Statements

Example: 2 (Class) *Show how the following selective concurrent signal assignment statement is synthesized.*

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
ENTITY mux IS
   PORT( a, b, c, d:  IN std_logic_vector (3 DOWNTO 0);
         s:  IN std_logic_vector (1 DOWNTO 0);
         x:  OUT std_logic_vector (3 DOWNTO 0));
END ENTITY mux;
ARCHITECTURE rtl OF mux IS BEGIN
   WITH s SELECT
      x <=  a             WHEN "00",
            b             WHEN "01",
            c             WHEN "10",
            d             WHEN "11",
            "- - - - "    WHEN OTHERS;
END ARCHITECTURE rtl;
```



Don’t care simplifies synthesized logic.

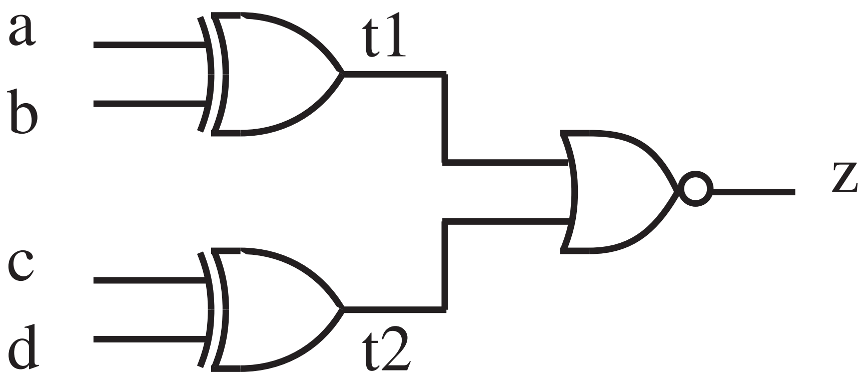
Synthesis of Process Statements

- Process statements can by synthesized to either:
  1. Pure combinational logic, or
  2. A sequential logic with latches or flip-flips inferred from the given VHDL code.
- For pure combinational logic synthesis of process statements, 3 rules must be obeyed:
  1. The sensitivity list must be complete (i.e., contains all signals read by the process).
  2. There must be no incomplete assignments (i.e., all outputs must be assigned for all input conditions).
  3. There must be no feedback (i.e., no signal or variable may be read by the process and then subsequently assigned by the process).

Synthesis of Variables and Signals

Example: 3 (Class) *Show how the assignment statements below are synthesized.*

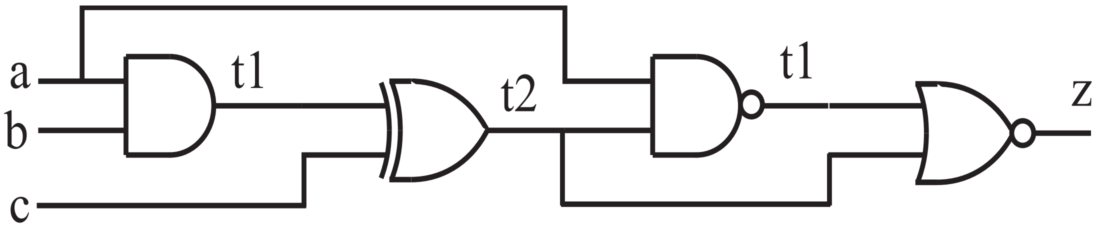
```
ENTITY gate IS
   PORT( a, b, c, d:  IN bit;
               z:  OUT bit);
END ENTITY gate;

ARCHITECTURE behav OF gate IS
BEGIN
   pg:  PROCESS (a, b, c, d) IS
      VARIABLE t1, t2:  bit;
   BEGIN
      t1 := a XOR b;
      t2 := c XOR d;
      z <= t1 NOR t2;
   END PROCESS pg;
END ARCHITECTURE behav;
```



Synthesis of Variables and Signals (Cont.)

Example: 4 (Class) *Show how the following assignment statements are synthesized.*

```
ENTITY gate IS
   PORT( a, b, c:  IN bit;
         z:  OUT bit);
END ENTITY gate;

ARCHITECTURE behav OF gate IS
BEGIN
   pg:  PROCESS (a, b, c) IS
      VARIABLE t1, t2:  bit;
   BEGIN
      t1 := a AND b;
      t2 := t1 XOR c;
      t1 := a NAND t2;
      z <= t1 NOR t2;
   END PROCESS pg;
END ARCHITECTURE behav;
```

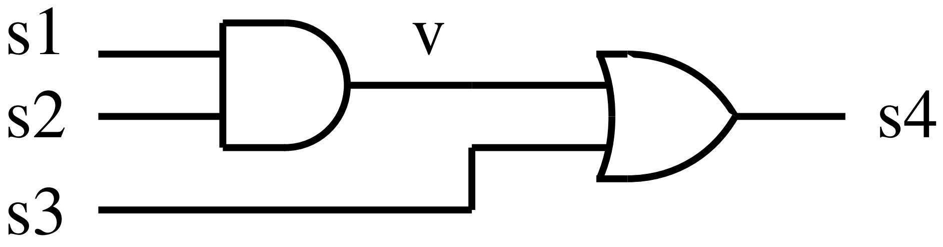


Variable t1 has generated two different wires.

Synthesis of Variables and Signals (Cont.)

Example: 5 (Class) *Show how the following assignment statements are synthesized.*

```
ENTITY gate IS
   PORT( s1, s2, s3:  IN bit;
         s4:  OUT bit);
END ENTITY gate;

ARCHITECTURE behav OF gate IS BEGIN
   p1:  PROCESS (s1, s2, s3) IS
      VARIABLE v:  bit;
   BEGIN
      v := s1 AND s2;
      s4 <= v OR s3;
   END PROCESS p1;
END ARCHITECTURE behav;
```



A latch is not required as v is temporary and need not be saved in a latch.

Synthesis of Variables and Signals (Cont.)

Example: 6 (Class) *Comment on the synthesis output if you reverse the order of execution of the sequential statements in Example 5.*

```
ENTITY gate IS
   PORT( s1, s2, s3:  IN bit;
         s4:  OUT bit);
END ENTITY gate;
ARCHITECTURE behav OF gate IS
BEGIN
   p1:  PROCESS (s1, s2, s3) IS
      VARIABLE v:  bit;
   BEGIN
      s4 <= v OR s3;
      v := s1 AND s2;
   END PROCESS p1;
END ARCHITECTURE behav;
```

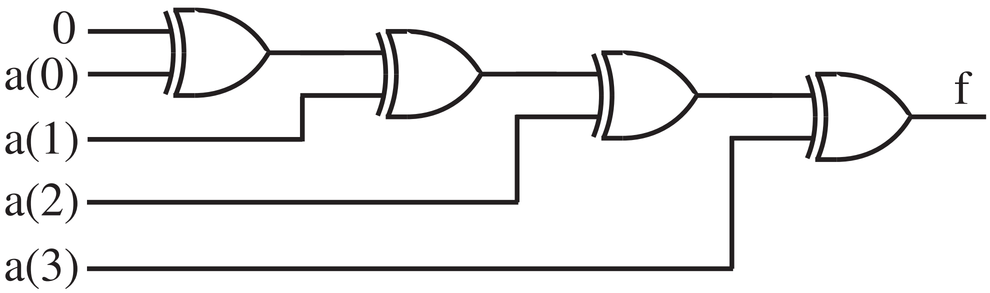
This code contains a feedback. v is read by the process and then assigned by it. v needs to retain its value for the next process run. Compiler might generate an error, produce the previous circuit, or infer a latch.

Synthesis of For-Loop Statements

FOR-LOOPs are the only ones typically supported for synthesis.

Example: 7 (Class) *Show how the following for-loop is synthesized.*

```
ENTITY example IS
   GENERIC( n:  integer := 4);
   PORT( a:  IN bit_vector (0 TO n-1);
         f:  OUT bit);
END ENTITY example;

ARCHITECTURE behav OF example IS
BEGIN
   fl:  PROCESS (a) IS
      VARIABLE v:  bit;
   BEGIN
      v := '0';
      FOR j IN a'range LOOP
         v := v XOR a(j);
      END LOOP;
      f <= v;
   END PROCESS fl;
END ARCHITECTURE behav;
```



Synthesis of For-Loop Statements (Cont.)

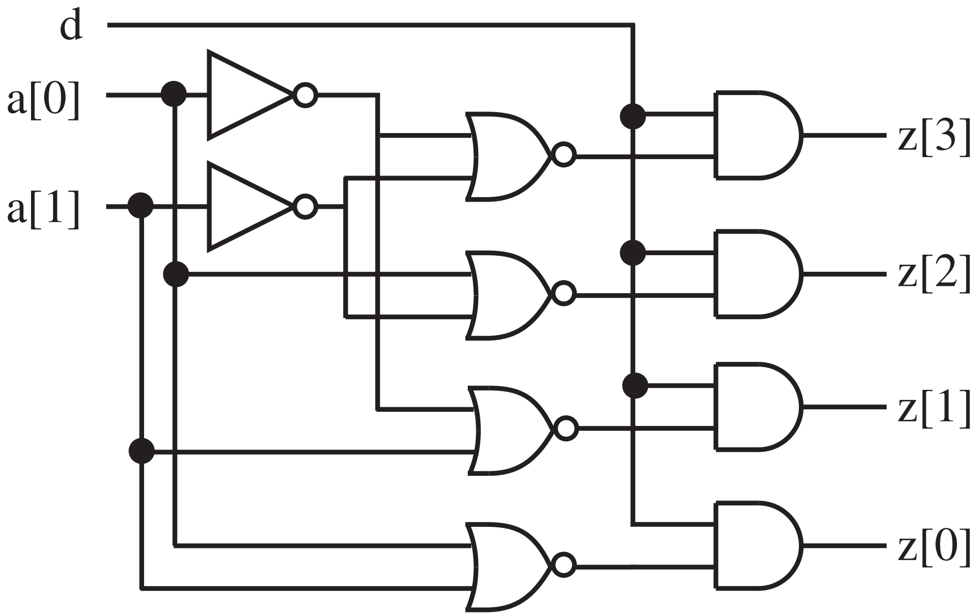
Example: 8 (Class) *Show how the following DEMUX is synthesized.*

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL; -- for the to_integer function

ENTITY demux IS
   PORT( a:  IN unsigned (1 DOWNTO 0);
         d:  IN std_logic;
         z:  OUT unsigned (3 DOWNTO 0));
END ENTITY demux;

ARCHITECTURE rtl OF demux IS
BEGIN
   dm:  PROCESS (a, d) IS
      VARIABLE temp:  integer RANGE 0 TO 3;
   BEGIN
      temp := to_integer (a); -- converts an unsigned into an integer
         ⋮
```

Synthesis of For-Loop Statements (Cont.)

Example 8: (Cont.)

```
          :
      FOR j IN z'range LOOP
         IF temp = j THEN
            z(j) <= d;
         ELSE
            z(j) <= '0';
         END IF;
      END LOOP
   END PROCESS dm;
END ARCHITECTURE rtl;
```



- NOR's are used instead of AND's for area/speed optimizations.
- When the for-loop is expanded, we get the following IF statements:

```
IF temp = 0 THEN z(0) <= d; ELSE z(0) <= '0'; END IF;
IF temp = 1 THEN z(1) <= d; ELSE z(1) <= '0'; END IF;
IF temp = 2 THEN z(2) <= d; ELSE z(2) <= '0'; END IF;
IF temp = 3 THEN z(3) <= d; ELSE z(3) <= '0'; END IF;
```

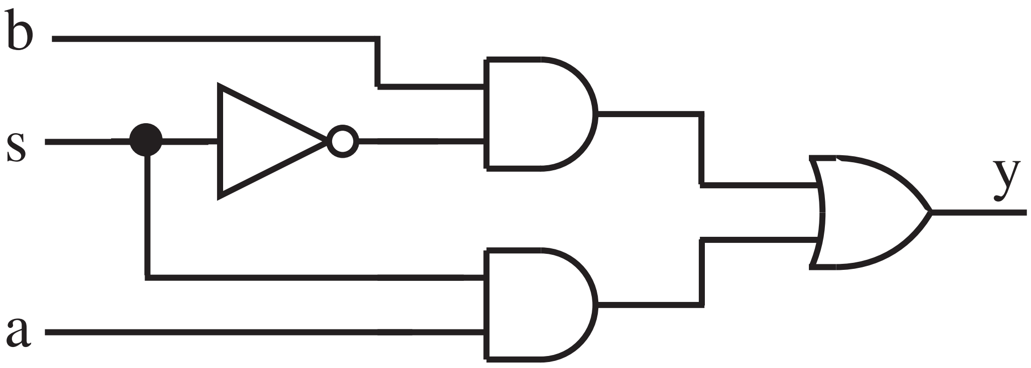
Inferring Latches

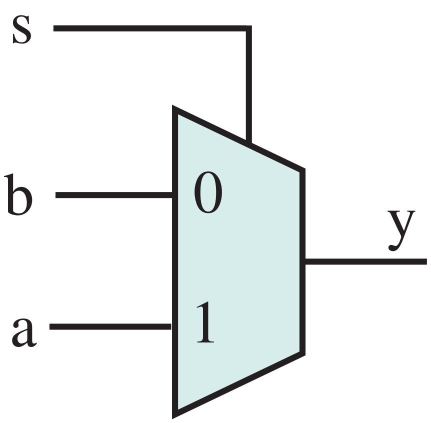
- A transparent latch will be inferred while synthesizing a process in two situations:
  1. Some branches of `IF` or `CASE` statements are incompletely specified and no default output values are defined before these two statements.
  2. A clock level detection condition is checked in `IF` or `WAIT` (or even in signal assignment statement (conditional or selective)).
- In other words, a latch is inferred if a path through the code exists such that a particular variable or signal is not assigned a new value.
- The latch will hold the value of that variable or signal.
- On the other hand, a latch will not be generated if a variable or signal is assigned a value in all branches of the IF or CASE statements.

Synthesis of IF Statements

IF statements synthesize to either elementary gates or muxs.

```
IF s THEN
   y <= a;
ELSE
   y <= b;
END IF;
```

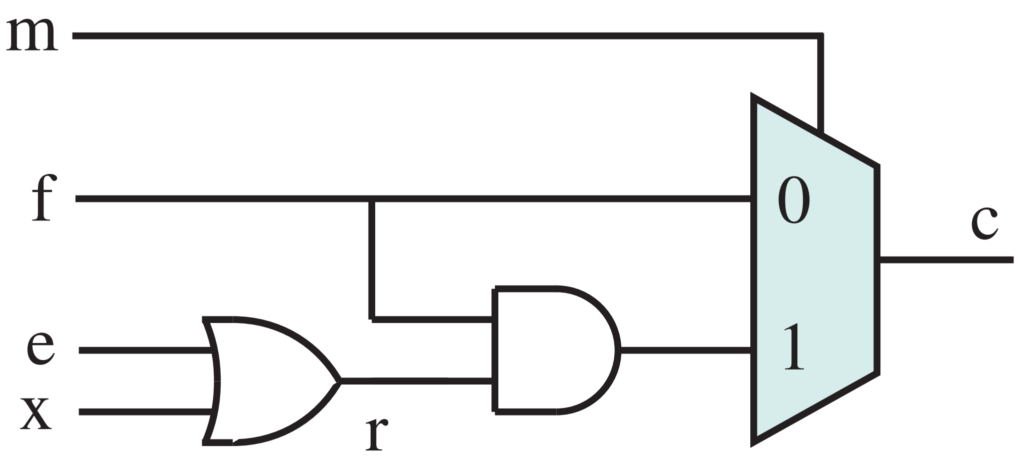




Synthesis of IF Statements (Cont.)

Example: 9 (Class) *Show how the IF statement below is synthesized.*

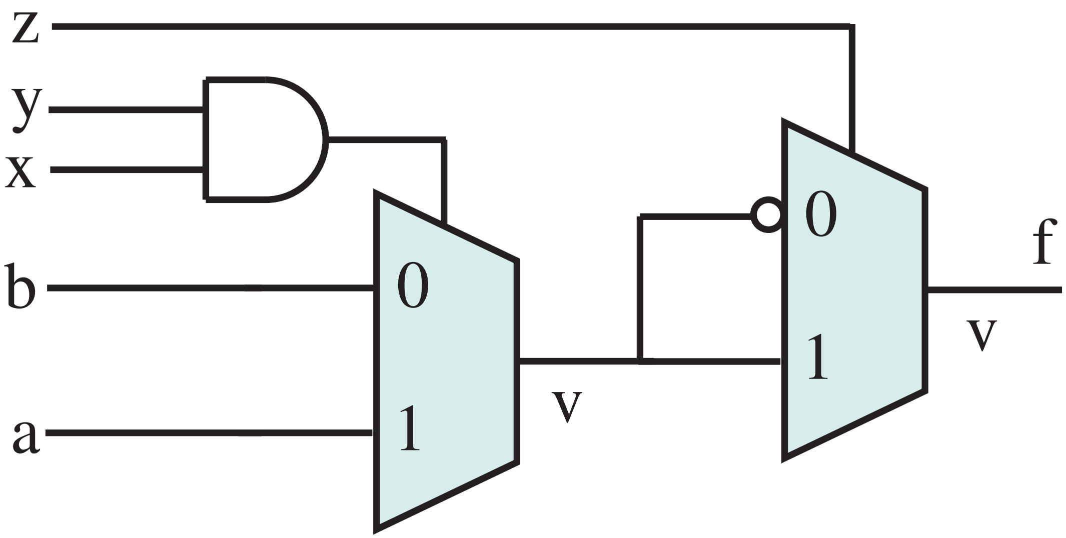
```
ENTITY sequence IS
   PORT( x, e, f:  IN bit;
         m:  IN boolean;
         c:  OUT bit);
END ENTITY sequence;

ARCHITECTURE comb OF sequence IS
   SIGNAL r:  bit;
BEGIN
   ps:  PROCESS (x, e, f, m, r) IS
   BEGIN
      r <= e OR x;
      IF m THEN
         c <= r AND f;
      ELSE
         c <= f;
      END IF;
   END PROCESS ps;
END ARCHITECTURE comb;
```



Synthesis of IF Statements (Cont.)

Example: 10 (Class) *Show how the IF statement below is synthesized.*

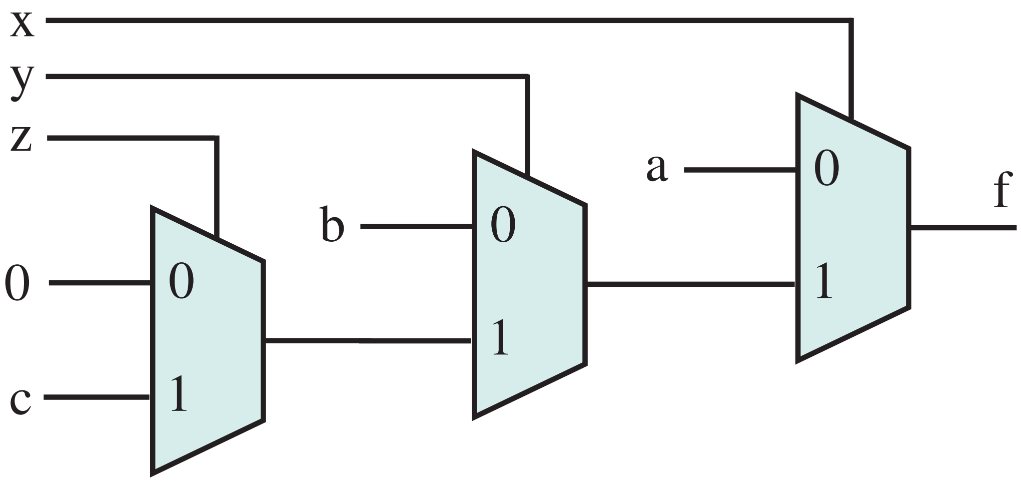
```
ENTITY circuit IS
   PORT( a, b, x, y, z:  IN bit;
         f:  OUT bit);
END ENTITY circuit;
ARCHITECTURE model OF circuit IS BEGIN
   p1:  PROCESS (a, b, x, y, z) IS
      VARIABLE v:  bit;
   BEGIN
      IF x = '1' AND y = '1' THEN
         v := a;
      ELSE
         v := b;
      END IF;
      IF z = '0' THEN
         v := NOT v;
      END IF;
      f <= v;
   END PROCESS p1;
END ARCHITECTURE model;
```



Synthesis of IF Statements (Cont.)

Example: 11 (Class) *Show how the IF statement below is synthesized.*

```
ENTITY ct IS
   PORT( a, b, c, x, y, z:  IN bit;
         f:  OUT bit);
END ENTITY ct;

ARCHITECTURE model OF ct IS
BEGIN
   p2:  PROCESS (a, b, c, x, y, z) IS
   BEGIN
      f <= '0';
      IF x = '0' THEN
         f <= a;
      ELSIF y = '0' THEN
         f <= b;
      ELSIF z = '1' THEN
         f <= c;
      END IF;
   END PROCESS p2;
END ARCHITECTURE model;
```

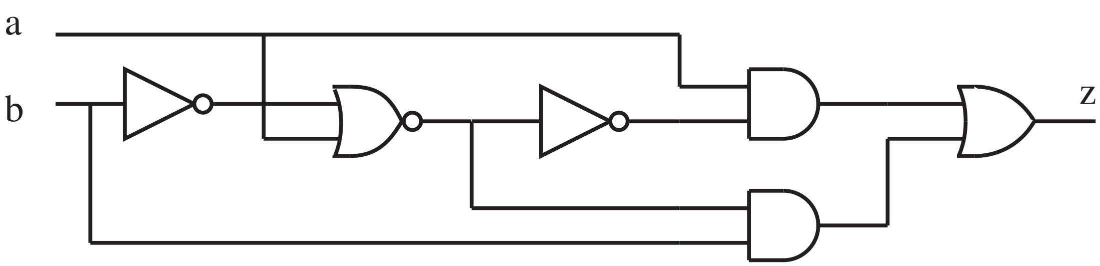


Synthesis of IF Statements (Cont.)

Example: 12 (Home) *Build a circuit that takes two single-bit inputs and asserts its output when the second input is greater than the first one.*

Example: 13 (Class) *Show how the IF statement below is synthesized.*

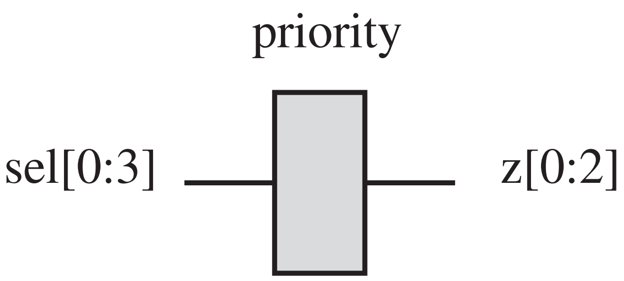
```
ENTITY compare IS
   PORT( a, b:   IN bit;
         z:   OUT bit);
END ENTITY compare;

ARCHITECTURE test OF compare IS
BEGIN
   pc:  PROCESS (a, b) IS
   BEGIN
      IF b > a THEN
         z <= b;
      ELSE
         z <= a;
      END IF;
   END PROCESS pc;
END ARCHITECTURE test;
```

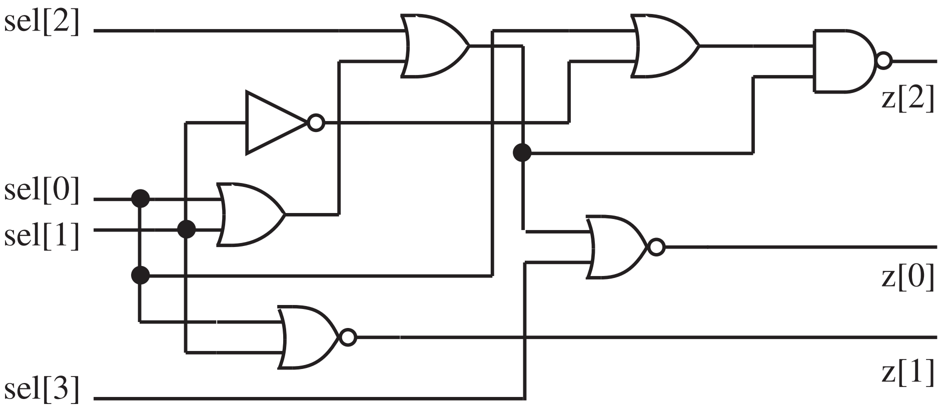


Synthesis of IF Statements (Cont.)

Example: 14 (Class) *Show how this priority encoder is synthesized.*

```
ENTITY priority IS
   PORT( sel:  IN bit_vector (0 TO 3);
         z:  OUT bit_vector (0 TO 2));
END ENTITY priority;
ARCHITECTURE behav OF priority IS BEGIN
   pp:  PROCESS (sel) IS BEGIN
      IF sel(0) = '1' THEN
         z <= "000";
      ELSIF sel(1) = '1' THEN
         z <= "001";
      ELSIF sel(2) = '1' THEN
         z <= "010";
      ELSIF sel(3) = '1' THEN
         z <= "011";
      ELSE
         z <= "111";
      END IF;
   END PROCESS pp;
END ARCHITECTURE behav;
```

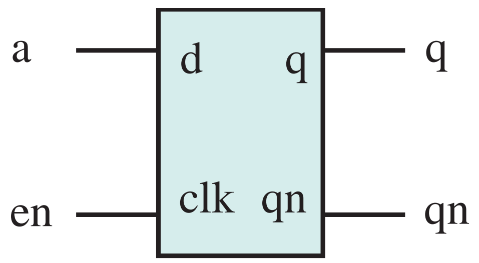




Synthesis of IF Statements (Cont.)

Example: 15 (Class) *Show how the following code for a latch will be synthesized.*

```
ENTITY latch_delay IS
   GENERIC( delay:  time := 1 ns);
   PORT( a, en:  IN bit;
         q, qn:  OUT bit);
END ENTITY latch_delay;
ARCHITECTURE behav OF latch_delay IS BEGIN
   pl:  PROCESS (a, en) IS BEGIN
      IF en = '1' THEN
         q <= a AFTER delay;
         qn <= NOT (a) AFTER delay;
      END IF;
   END PROCESS pl;
END ARCHITECTURE behav;
```

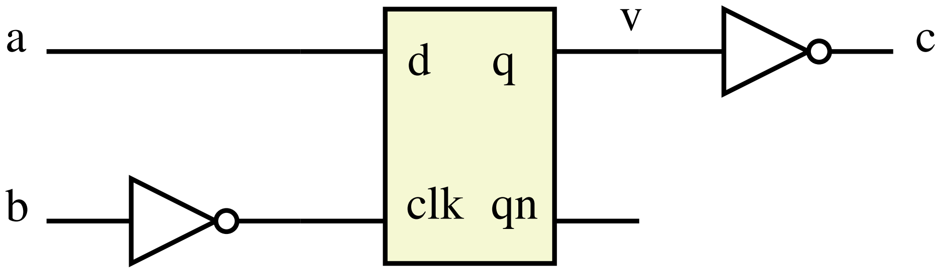


- Optimization uses the qn output instead of installing another latch.
- **Rule:** Synthesis ignores inertial and transport delays.

Synthesis of IF Statements (Cont.)

Example: 16 (Class) *Show how the IF statement below is synthesized.*

```
ENTITY gate IS
   PORT( a, b:   IN bit;
         c:   OUT bit);
END ENTITY gate;
ARCHITECTURE behav OF gate IS
BEGIN
   pl:  PROCESS (a, b) IS
      VARIABLE v:  bit;
   BEGIN
      IF b = '0' THEN
         v := a;
      END IF;
      c <= NOT (v);
   END PROCESS pl;
END ARCHITECTURE behav;
```

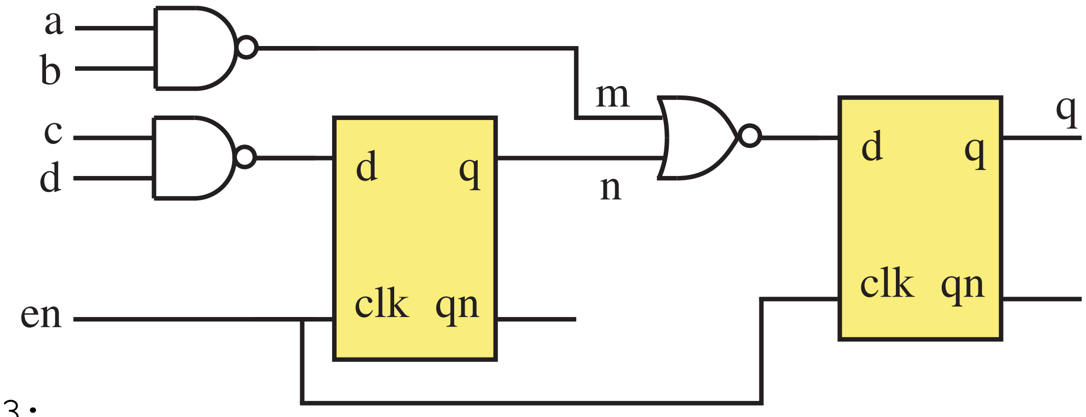


- v is not assigned in the ELSE branch of IF statement. This implies a latch to keep the value of v when `b = '1'`.
- If c was inside the IF statement, it would have been connected to qn.

Synthesis of IF Statements (Cont.)

Example: 17 (Class) *Show how the IF statement below is synthesized.*

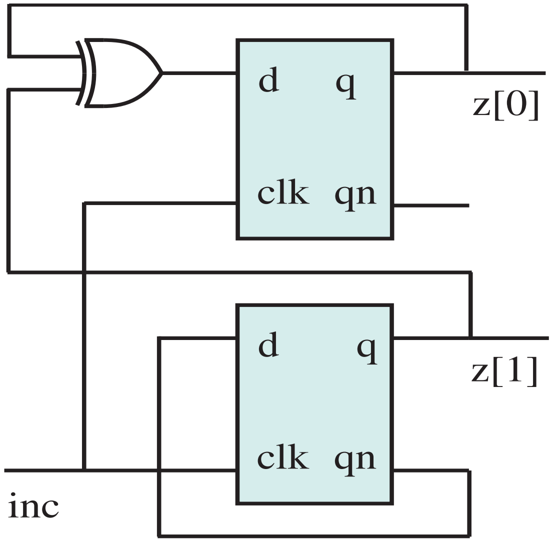
```
ENTITY latch_3 IS
   PORT( a, b, c, d, en:  IN bit;
         q:  OUT bit);
END ENTITY latch_3;
ARCHITECTURE behave_3 OF latch_3 IS
   SIGNAL n:  bit;
BEGIN
   pl:  PROCESS (a, b, c, d, en, n) IS
      VARIABLE m:  bit;
   BEGIN
      IF en = '1' THEN
         m := a NAND b;
         n <= c NAND d;
         q <= m NOR n;
      END IF;
   END PROCESS pl;
END ARCHITECTURE behave_3;
```



- No need to latch m since the variable is used after it is defined.
- A latch is needed for n since its old value is used every process run.

Synthesis of IF Statements (Cont.)

Example: 18 (Class) *Show how the increment-by-one circuit below is synthesized.*



```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;
ENTITY increment IS
   PORT( inc:  IN bit;
         z:  OUT unsigned (0 TO 1));
END ENTITY increment;
ARCHITECTURE behav OF increment IS
BEGIN
   pi:  PROCESS (inc) IS
      VARIABLE temp:  unsigned (0 TO 1) := (OTHERS => '0');
   BEGIN
      IF inc = '1' THEN
         temp := temp + 1;
      END IF;
      z <= temp;
   END PROCESS pi;
END ARCHITECTURE behav;
```

Synthesis of CASE Statements

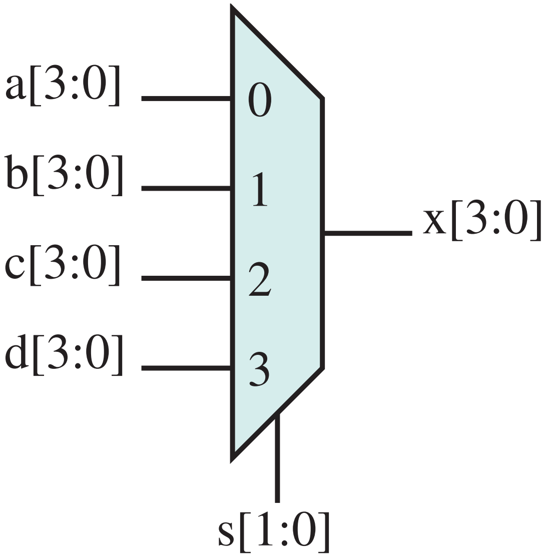
- CASE statements synthesize to MUXs or elementary gates.
- A latch is inferred if some branches of CASE statement are unspecified.
- If NULL is used in any of the branches, a latch is inferred.
- In other words, a latch is inferred if a path through the code exists such that a particular variable or signal is not assigned a new value.
- The latch will hold the value of the signal.
- If we have a CASE statement and do not want a latch, then we must use OTHERS and must assign a value to the output in every branch.
- **Examples:**

```
WHEN OTHERS => y := "0000_0000";
WHEN OTHERS => y := "- - - -_- - - - ";
```

Synthesis of CASE Statements (Cont.)

Example: 19 (Class) *Show how the CASE statement below is synthesized.*

```
ENTITY mux IS
   PORT( a, b, c, d:  IN bit_vector (3 DOWNTO 0);
         s:  IN bit_vector (1 DOWNTO 0);
         x:  OUT bit_vector (3 DOWNTO 0));
END ENTITY mux;

ARCHITECTURE rtl OF mux IS BEGIN
   sl:  PROCESS (a, b, c, d, s) IS
   BEGIN
      CASE s IS
         WHEN "00" => x <= a;
         WHEN "01" => x <= b;
         WHEN "10" => x <= c;
         WHEN "11" => x <= d;
      END CASE;
   END PROCESS sl;
END ARCHITECTURE rtl;
```



Synthesis of CASE Statements (Cont.)

Example: 20 (Class) *Show how the following 3-to-6 decoder with an enable is synthesized.*

inputs				outputs					
En	a2	a1	a0	y5	y4	y3	y2	y1	y0
0	x	x	x	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	1
1	0	0	1	0	0	0	0	1	0
1	0	1	0	0	0	0	1	0	0
1	0	1	1	0	0	1	0	0	0
1	1	0	0	0	1	0	0	0	0
1	1	0	1	1	0	0	0	0	0

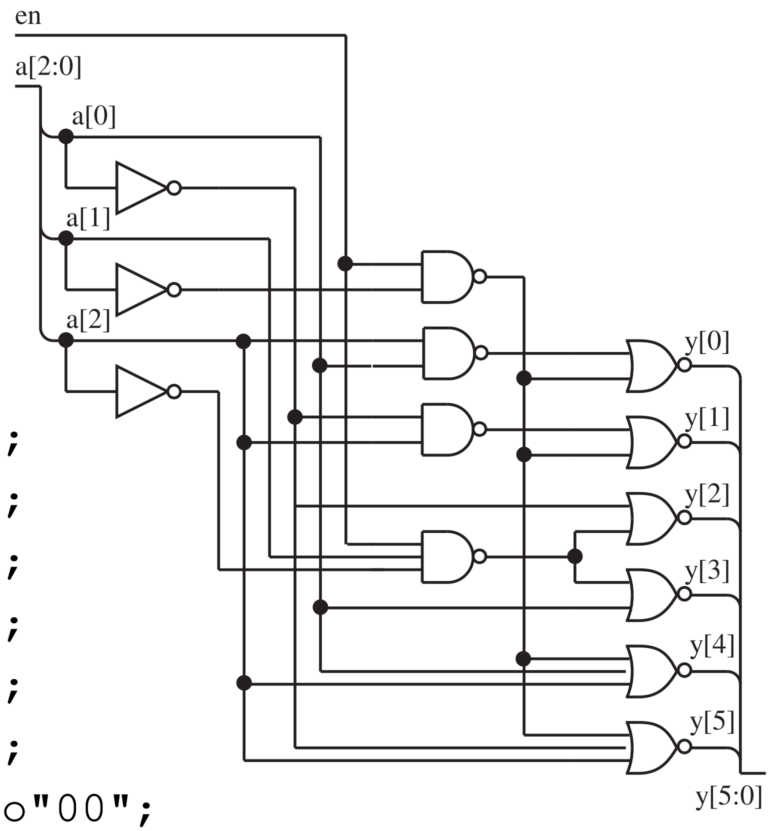
Synthesis of CASE Statements (Cont.)

Example 20: (Cont.)

```
LIBRARY ieee;
USE ieee.numeric_bit.ALL;

ENTITY decoder IS
   PORT( a:   IN integer RANGE 0 TO 7;
         en:   IN bit;
         y:   OUT unsigned (5 DOWNTO 0));
END ENTITY decoder;

ARCHITECTURE design OF decoder IS
BEGIN
   pd:  PROCESS (a, en) IS
   BEGIN
      IF en = ’0’ THEN
         y <= o"00";
      ELSE
         ⋮
```

Synthesis of CASE Statements (Cont.)

Example 20: (Cont.)



```
            :
            CASE a IS
               WHEN 0 => y <= o"01";
               WHEN 1 => y <= o"02";
               WHEN 2 => y <= o"04";
               WHEN 3 => y <= o"10";
               WHEN 4 => y <= o"20";
               WHEN 5 => y <= o"40";
               WHEN OTHERS => y <= o"00";
               END CASE;
            END IF;
         END PROCESS pd;
END ARCHITECTURE design;
```

Synthesis of CASE Statements (Cont.)

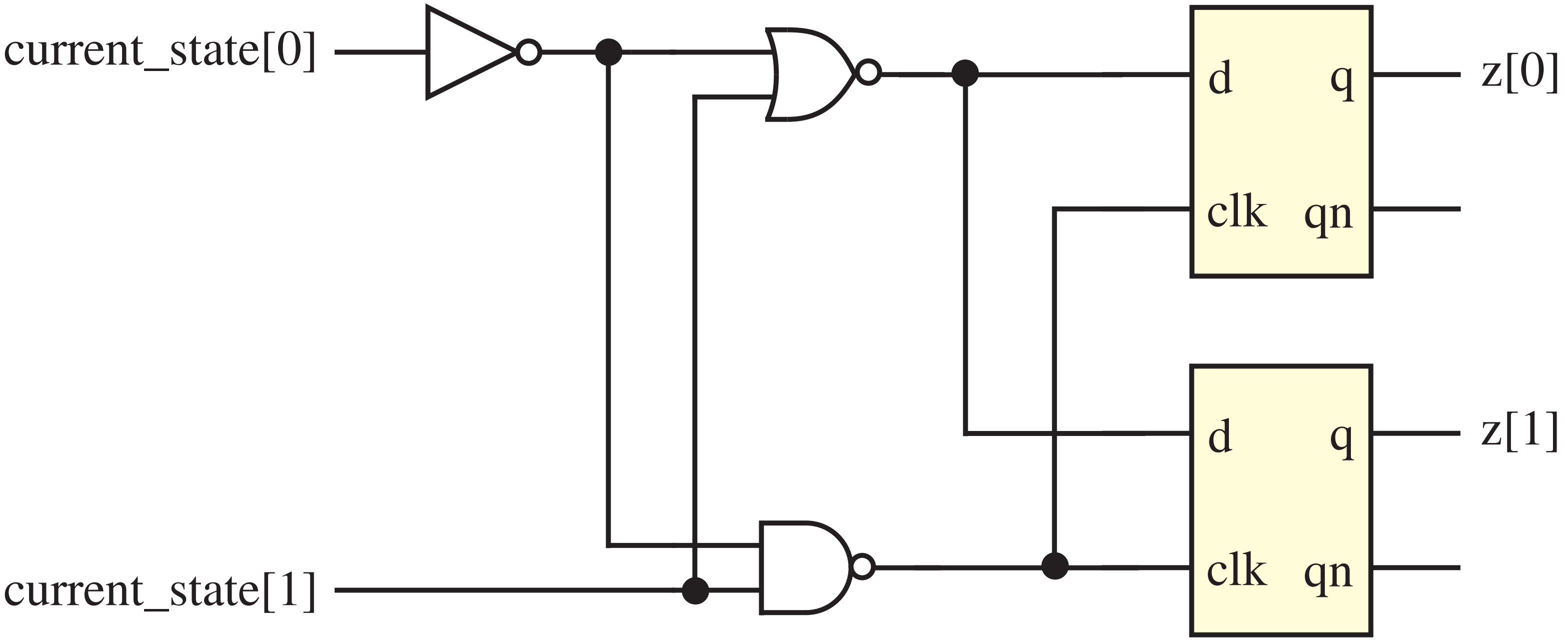
Example: 21 (Class) *Show how this CASE statement is synthesized.*

```
PACKAGE collect IS
   TYPE state IS (s0, s1, s2, s3);
END PACKAGE collect;

USE WORK.collect.ALL;
ENTITY state_update IS
   PORT( current_state:  IN state;
         z:  OUT integer RANGE 0 TO 3);
END ENTITY state_update;
ARCHITECTURE update OF state_update IS
BEGIN
   cs:  PROCESS (current_state) IS
   BEGIN
      CASE current_state IS
         WHEN s0 | s3        => z <= 0;
         WHEN s1             => z <= 3;
         WHEN OTHERS         => NULL;
      END CASE;
   END PROCESS cs;
END ARCHITECTURE update;
```

Synthesis of CASE Statements (Cont.)

Example 21: (Cont.)

A latch is inferred since z is not assigned a value in the OTHERS statement even though z is defined in every other branch of CASE statement.



Synthesis of CASE Statements (Cont.)

Example: 22 (Home) *Show how the encoder written using the CASE statement below is synthesized.*

```
ENTITY latch_case IS
   PORT (a:  IN integer RANGE 0 TO 15;
         y:  OUT integer RANGE 0 TO 4);
END ENTITY latch_case;
ARCHITECTURE rtl OF latch_case IS BEGIN
   p1:  PROCESS (a) BEGIN
      CASE a IS
         WHEN 0 TO 3 | 5            => y <= 1;
         WHEN 6 TO 8 | 10 TO 13     => y <= 2;
         WHEN 4 | 9                 => y <= 3;
         WHEN 15                    => y <= 4;
         WHEN OTHERS                => NULL;
      END CASE;
   END PROCESS p1;
END ARCHITECTURE rtl;
```

Synthesis of CASE Statements (Cont.)

Example: 23 (Home) *Show how this CASE statement is synthesized.*

```
PACKAGE types IS
   TYPE primecolor IS (red, green, blue);
END PACKAGE types;

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;
USE WORK.types.ALL;

ENTITY latch_nested IS
   PORT( screencolor:  IN primecolor;
         number:  IN unsigned (1 DOWNTO 0);
         a:  IN unsigned (3 DOWNTO 0) := "0000";
         y:  OUT unsigned (3 DOWNTO 0));
END ENTITY latch_nested;

ARCHITECTURE rtl OF latch_nested IS
BEGIN
         ⋮
```

Synthesis of CASE Statements (Cont.)

Example 23: (Cont.)

```
         ⋮
   pc:  PROCESS (screencolor, number, a) IS
      VARIABLE y_var:  unsigned (3 DOWNTO 0) := "0000";
   BEGIN
      CASE screencolor IS
         WHEN red     =>   y_var := a;
         WHEN green   =>   y_var := a + 1;
         WHEN blue    =>   CASE number IS
                              WHEN "00"    => y_var := a;
                              WHEN "01"    => y_var := a + 1;
                              WHEN "10"    => y_var := a + 2;
                              WHEN OTHERS  => NULL;
                           END CASE;
         WHEN OTHERS  =>   y_var:= a + 1;
      END CASE;
      y <= y_var;
   END PROCESS pc;
END ARCHITECTURE rtl;
```

Rule: Synthesis ignores default and initial values.

Inferring Flip-Flops

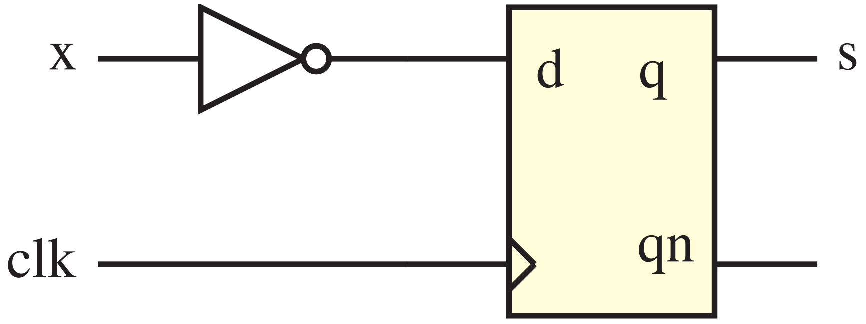
- A flip-flop is inferred in a process if the following conditions are all satisfied:
  - A clock edge detection condition is used in IF or WAIT statements.
  - The process sensitivity list includes the clock and nothing else except set and reset signals.
  - No other statements should be written before or after the IF/WAIT statement in the process, except reset and set statements.
- Multiple IF statements will selectively infer flip-flops.
- Examples of using clock edge detection conditions:

```
WAIT UNTIL clock = ’1’ AND clock’event;
WAIT UNTIL rising_edge (clock);
IF clock = ’0’ AND clock’event THEN ···
IF falling_edge (clock) THEN ···
```

Inferring Flip-Flops (Cont.)

Example: 24 (Class) *Show how the following IF statement is synthesized.*

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY ff IS
   PORT( x, clk:  IN std_logic;
         s:  OUT std_logic);
END ENTITY ff;

ARCHITECTURE ff OF ff IS
BEGIN
   f:  PROCESS (clk) IS
   BEGIN
      IF rising_edge (clk) THEN
         s <= NOT x;
      END IF;
   END PROCESS f;
END ARCHITECTURE ff;
```



Synthesis of WAIT Statements

- Statements after WAIT execute synchronously with clock edge.
- Hence the WAIT statement is synthesized as an edge-triggered flip-flop.
- Synthesis tools allow only one `WAIT` statement in the whole process. It is not possible to model combinational logic in that process.
- On the other hand, `IF` statements allow defining combinational logic and inferred latches and flip-flops in the same process.
- Sequential logic is synthesized using a process statement with a special form:

```
pw:  PROCESS IS BEGIN
   WAIT UNTIL clock_condition;
   -- Synchronous logic described which is
   -- executed when clock level or edge occurs.
END PROCESS pw;
```

Synthesis of WAIT Statements (Cont.)

- `WAIT UNTIL` is supported for synthesis under two conditions:
  1. WAIT statement must be the first statement in the process.
  2. The boolean condition in `WAIT UNTIL` statement must have one of the following forms to indicate falling or rising clock edge. Clock of type std_logic.

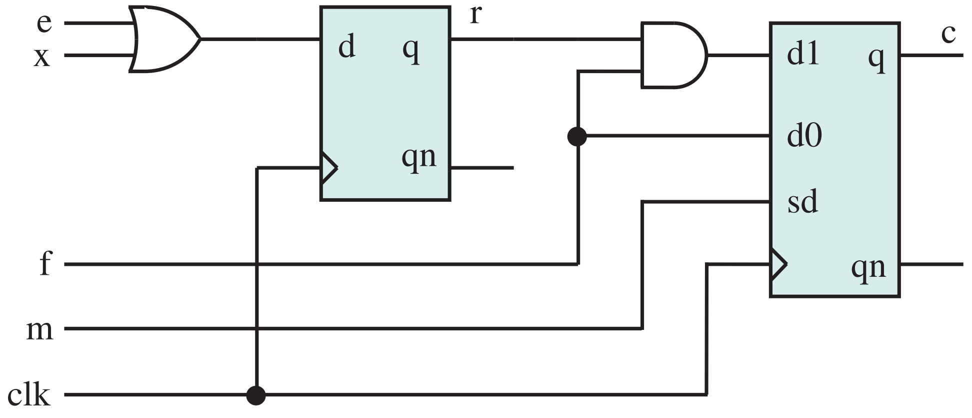
```
WAIT UNTIL clock = clock_value;
WAIT UNTIL clock = clock_value AND clock'event;
WAIT UNTIL rising_edge (clock);
WAIT UNTIL falling_edge (clock);
```

- The `WAIT` statement delays the execution of the whole process until its expression becomes true.
- This means that all other variable/signal assignment statements in the process will infer flip-flops.

Synthesis of WAIT Statements (Cont.)

Example: 25 (Class) *Show how Example 9 synchronized is synthesized.*

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
ENTITY sequence IS
   PORT( e, x, f, clk:  IN std_logic;
         m:  IN boolean;
         c:  OUT std_logic); -- c uses a flip-flop with input select
END ENTITY sequence;
ARCHITECTURE test OF sequence IS
   SIGNAL r:  std_logic;
BEGIN
   ps:  PROCESS IS BEGIN
      WAIT UNTIL rising_edge (clk);
      r <= e OR x;
      IF m THEN
         c <= r AND f;
      ELSE
         c <= f;
      END IF;
   END PROCESS ps;
END ARCHITECTURE test;
```



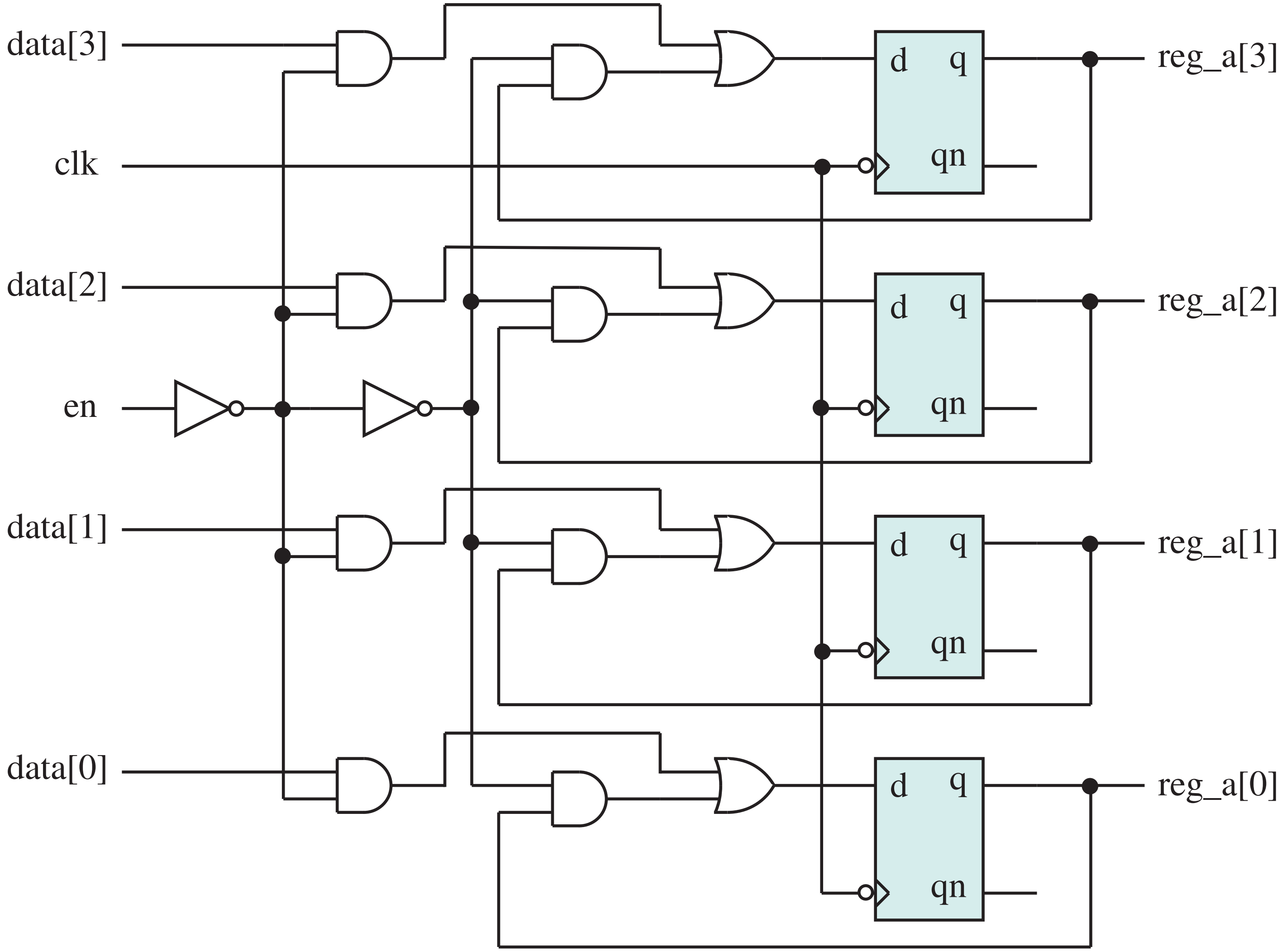
Synthesis of WAIT Statements (Cont.)

Example: 26 (Class) *Show how this WAIT statement is synthesized.*

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY reg IS
   PORT( clk, en:  IN std_logic;
         data:  IN std_logic_vector (0 TO 3);
         reg_q:  OUT std_logic_vector (0 To 3));
END ENTITY reg;

ARCHITECTURE behav OF reg IS
BEGIN
   sy:  PROCESS IS
   BEGIN
      WAIT UNTIL falling_edge (clk);
      IF en = ’0’ THEN
         reg_q <= data;
      END IF
   END PROCESS sy;
END ARCHITECTURE behav;
```

Synthesis of WAIT Statements (Cont.)

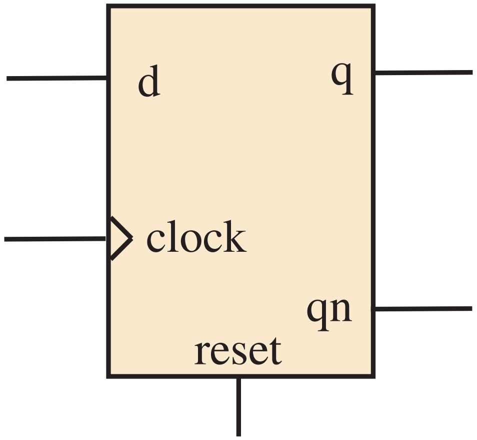
Example 26: (Cont.)



Synthesis of Asynchronous Set or Reset

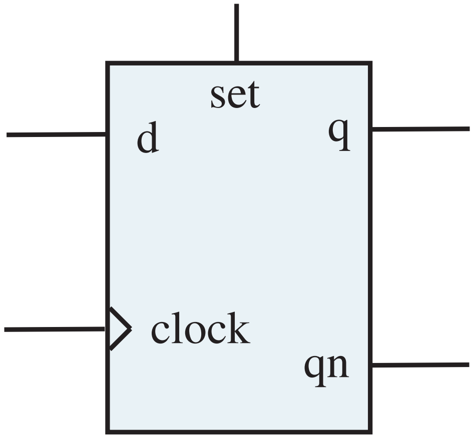
Example: 27 (Class) *Show how asynchronous reset is synthesized.*

```
rs:  PROCESS (clock, reset)
BEGIN
   IF reset = '1' THEN
      q <= '0';
   ELSIF rising_edge (clock) THEN
      q <= d;
   END IF;
END PROCESS rs;
```



Example: 28 (Class) *Show how asynchronous set is synthesized.*

```
st:  PROCESS (clock, set)
BEGIN
   IF set = '1' THEN
      q <= '1';
   ELSIF rising_edge (clock) THEN
      q <= d;
   END IF;
END PROCESS st;
```



Synthesis of Asynchronous Set or Reset (Cont.)

- Clocked logic with asynchronous reset or set is synthesized by testing the level of the asynchronous control signal before testing for the clock edge.
- In fact, the form of the process must be exactly as shown before.
- Both the clock and the asynchronous reset or set must be in the sensitivity list of the process with nothing else.
- The process must contain one IF statement, with no other statements before or after that IF statement.
- The IF statement must contain exactly one ELSIF part with no ELSE part.
- The first condition of the IF statement must test the level of the asynchronous set or reset and nothing else.

Synthesis of Asynchronous Set or Reset (Cont.)

- The statements in the first branch of the IF statement must contain assignments to reset or set all flip-flops inferred by the process.
- The second condition of the IF statement (the ELSIF part) must test for the clock edge and nothing else.
- The second branch of the IF statement may contain any number and combination of synthesizable sequential statements.
- The reset, set, and clock signals must not be used anywhere in the process except in the sensitivity list and IF conditions as described above.
- All flip-flops inferred from the second branch of the IF statement must be reset or set in the first branch, and all flip-flops reset or set in the first branch must be inferred in the second branch.

Synthesis of Procedures and Functions

- Synthesis tools require that procedures and functions:
  - represent blocks of combinational logic,
  - do not detect edges or contain WAIT statements.
- Out and inout parameters may be synthesized as registers if the procedure or function is called from a clocked process.

Example: 29 (Home) *Apply above VHDL rules on synthesizing procedures and functions on the code below.*

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;
ENTITY counter IS
   PORT( clock, reset, load:  IN bit;
         start, stop:  IN unsigned (3 DOWNTO 0);
         count:  INOUT unsigned (3 DOWNTO 0));
END ENTITY counter;
ARCHITECTURE rtl OF counter IS
         ⋮
```

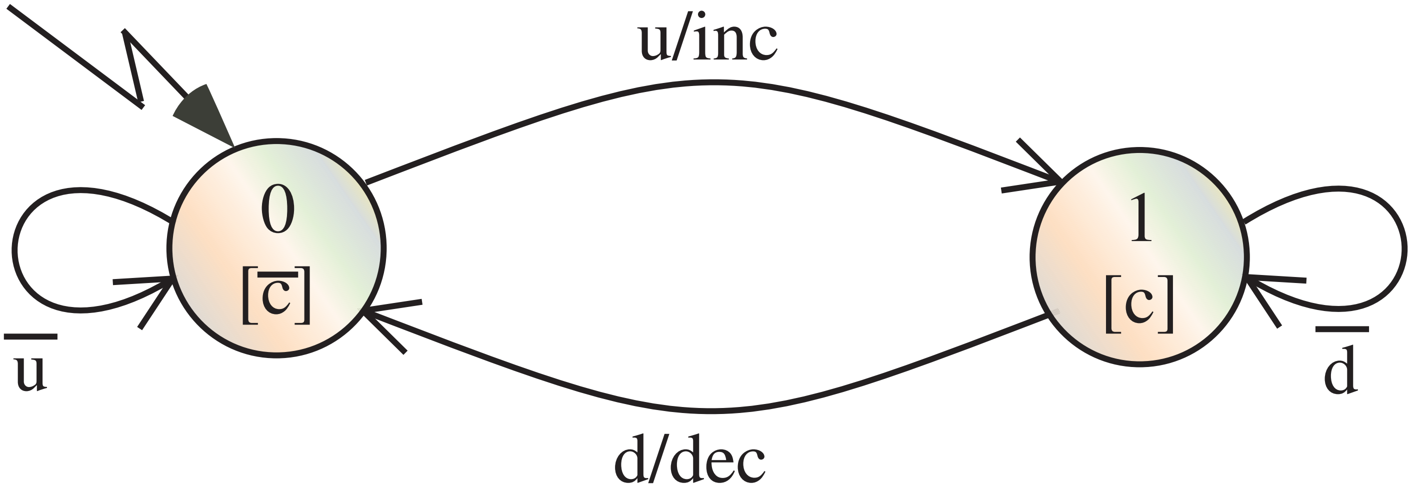
Synthesis of Procedures and Functions (Cont.)

Example 29: (Cont.)

```
          ⋮
   PROCEDURE counting( SIGNAL load:  bit; SIGNAL start, stop:  unsigned;
                       SIGNAL count:  INOUT unsigned) IS
   BEGIN
      IF load = ’1’ THEN
         count <= start;
      ELSIF count < stop THEN
         count <= count + 1;
      ELSE
         count <= count + 0;
      END IF;
   END PROCEDURE counting;
BEGIN
   pc:  PROCESS (clock, reset) IS BEGIN
      IF reset = ’1’ THEN
         count <= (OTHERS => ’0’);
      ELSIF clock = ’1’ and clock’event THEN
         counting (load, start, stop, count);
      END IF;
   END PROCESS pc;
END ARCHITECTURE rtl;
```

Synthesis of Finite-State Machines (FSM)

Example: 30 (Class) *Model the following FSM using a two-process architecture. Represent states using integers. (Hint: the shown FSM has Moore and Mealy outputs.) Show how the modelled FSM is synthesized. Assume that IF statements are synthesized using muxs.*



Notes:

- When an input is not specified on an FSM, it is a don't care.
- When an output is not specified on an FSM, it is a zero value.

Synthesis of Finite-State Machines (FSM) (Cont.)

Example 30: (Cont.)

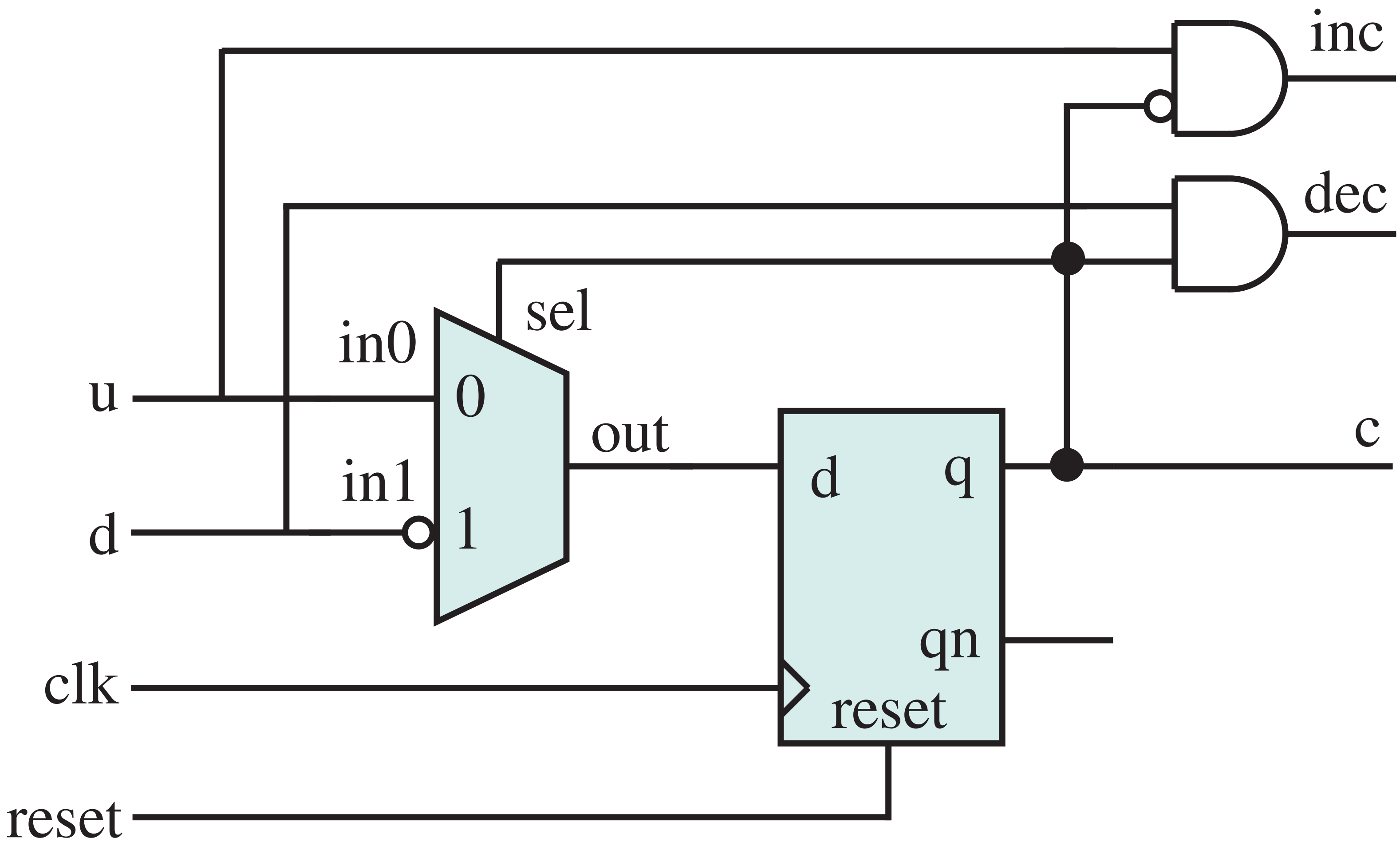
```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
ENTITY source IS
   PORT( reset, u, d:  IN bit;
         clk:  IN std_logic;
         inc, dec, c:  OUT bit);
END ENTITY source;
ARCHITECTURE fsm OF source IS
   SIGNAL current_state, next_state:  integer RANGE 0 TO 1;
BEGIN
   ps:  PROCESS (clk, reset) IS BEGIN
      IF reset = '1' THEN
         current_state <= 0;
      ELSIF rising_edge (clk) THEN
         current_state <= next_state;
      END IF;
   END PROCESS ps;
            ⋮
```

Synthesis of Finite-State Machines (FSM) (Cont.)

Example 30: (Cont.)

```
            ⋮
   ns:  PROCESS (current_state, u, d) IS BEGIN
      CASE current_state IS
         WHEN 0 =>
            c <= ’0’; -- Moore output
            IF u = ’1’ THEN
               inc <= ’1’; dec <= ’0’; next_state <= 1;
            ELSE
               inc <= ’0’; dec <= ’0’; next_state <= 0;
            END IF;
         WHEN 1 =>
            c <= ’1’;
            IF d = ’1’ THEN
               inc <= ’0’; dec <= ’1’; next_state <= 0;
            ELSE
               inc <= ’0’; dec <= ’0’; next_state <= 1;
            END IF;
      END CASE;
   END PROCESS ns;
END ARCHITECTURE fsm;
```

Synthesis of Finite-State Machines (FSM) (Cont.)



Inferring Tristate Elements

- To create tristate logic we must use `std_logic` type since it has `'Z'` as one of its possible values.
- We get tristate elements when the value `'Z'` is assigned to a target value.
- It is possible to combine inferring tristate gates with memory elements (flip-flops and latches).

Inferring Tristate Elements (Cont.)

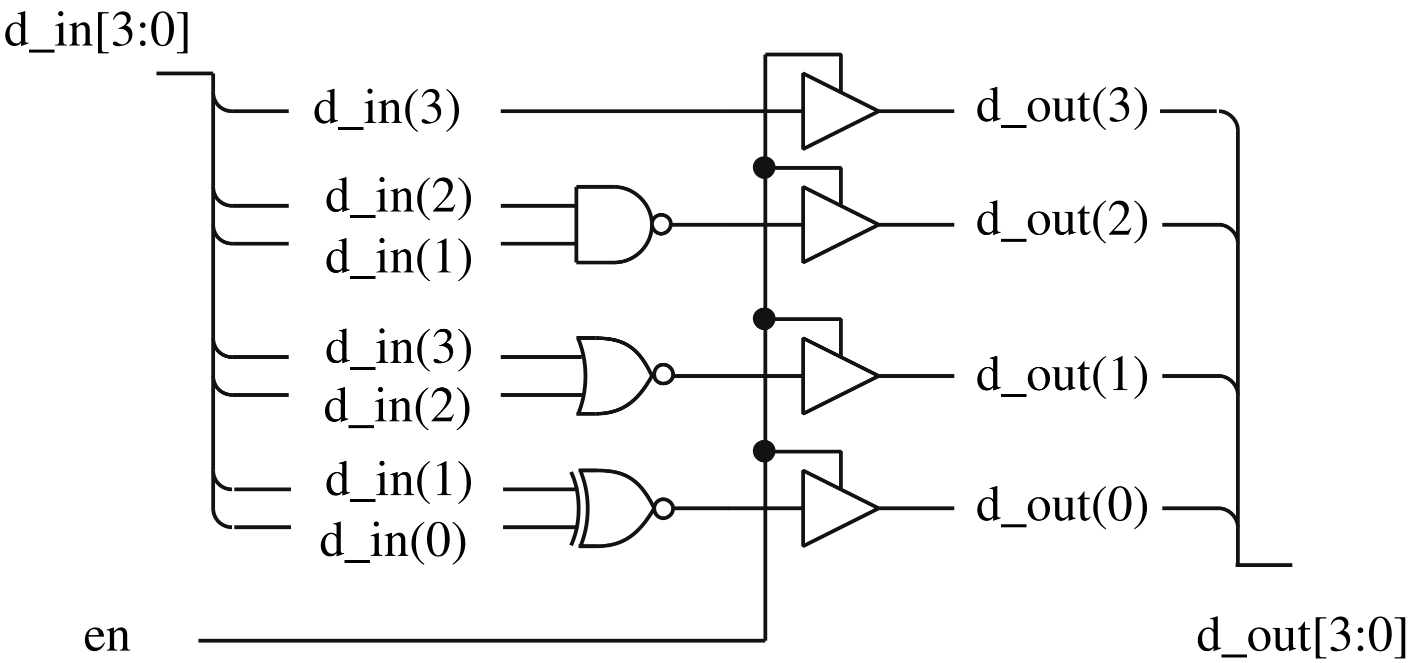
Example: 31 (Class) *Show how the code below is synthesized.*

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
ENTITY state3 IS
   PORT( en:  IN std_logic;
         d_in:  IN std_logic_vector (3 DOWNTO 0);
         d_out:  OUT std_logic_vector (3 DOWNTO 0));
END ENTITY state3;
ARCHITECTURE rtl OF state3 IS BEGIN
   p1:  PROCESS (en, d_in) IS BEGIN
      IF en = ’1’ THEN
         d_out (0) <= d_in (0) XNOR d_in (1);
      ELSE
         d_out (0) <= ’Z’;
      END IF;
      CASE en IS
         WHEN ’1’ => d_out (1) <= d_in (2) NOR d_in (3);
         WHEN OTHERS => d_out (1) <= ’Z’;
      END CASE;
   END PROCESS p1;
         ⋮
```

Inferring Tristate Elements (Cont.)

Example 31: (Cont.)

```
          ⋮
    d_out (2) <=d_in (1) NAND d_in (2) WHEN en = '1' ELSE 'Z';
    WITH en SELECT
       d_out (3) <=d_in (3) WHEN '1',
                   'Z' WHEN OTHERS;
END ARCHITECTURE rtl;
```



Inferring Tristate Elements (Cont.)

Example: 32 (Home) *Show how the code below is synthesized.*

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY prebus IS
   PORT( my_in:  IN std_logic_vector (7 DOWNTO 0);
         sel:  IN std_logic;
         my_out:  OUT std_logic_vector (7 DOWNTO 0));
END ENTITY prebus;

ARCHITECTURE maxpld OF prebus IS
BEGIN
   my_out <= my_in WHEN sel = ’1’ ELSE
            (OTHERS => ’Z’);
END ARCHITECTURE maxpld;
```

Inferring Tristate Elements (Cont.)

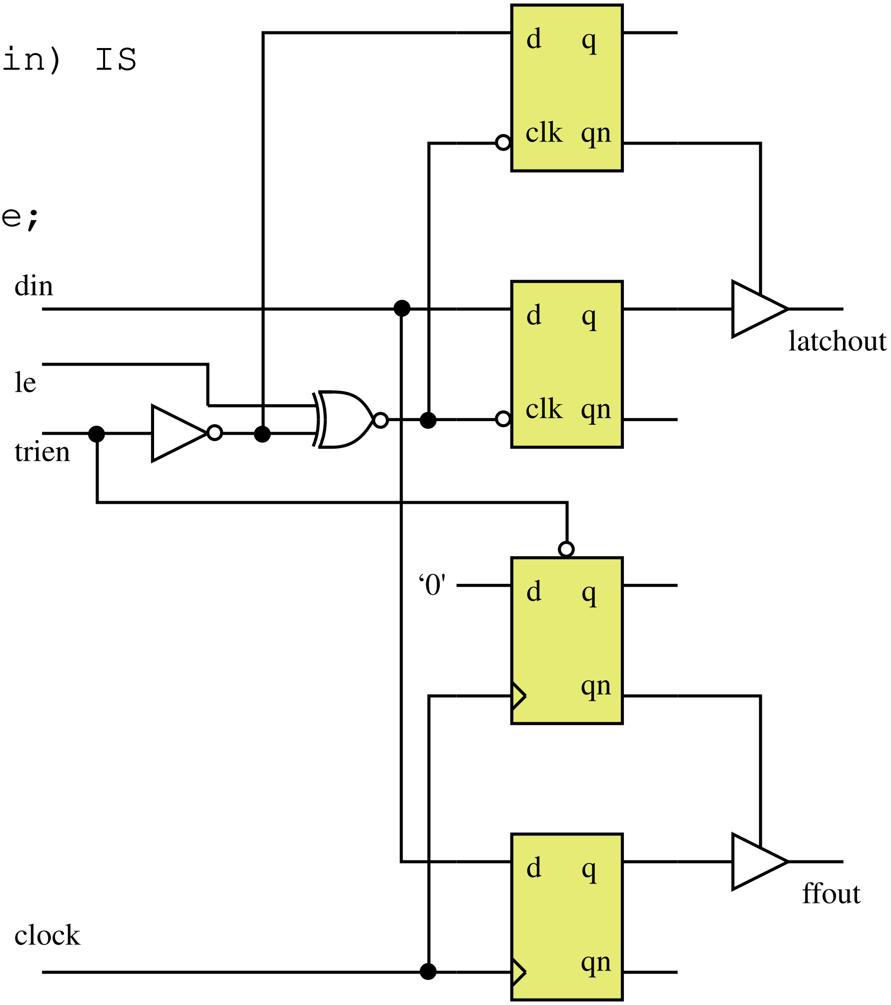
Example: 33 (Class) *Show how the code below is synthesized.*

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY trireg IS
   PORT( trien, clock, le, din:  IN std_logic;
         ffout, latchout:  OUT std_logic);
END ENTITY trireg;
ARCHITECTURE mixed OF trireg IS
BEGIN
   ff:  PROCESS (trien, clock) IS
   BEGIN
      IF trien = ’0’ THEN
         ffout <= ’Z’;
      ELSIF rising_edge (clock) THEN
         ffout <= din;
      END IF;
   END PROCESS ff;
         ⋮
```

Inferring Tristate Elements (Cont.)

Example 33: (Cont.)

```
          .
          .
          .
   latch:  PROCESS (trien, le, din) IS
      variable temp:  std_logic;
   BEGIN
      temp := NOT (trien) XNOR le;
      IF temp = '0' THEN
         IF trien = '0' THEN
            latchout <= 'Z';
         ELSE
            latchout <= din;
         END IF;
      END IF;
   END PROCESS latch;
END ARCHITECTURE mixed;
```



d
q
clk
qn
din
d
q
latchout
le
clk
qn
trien
'0'
d
q
qn
d
q
ffout
clock
qn