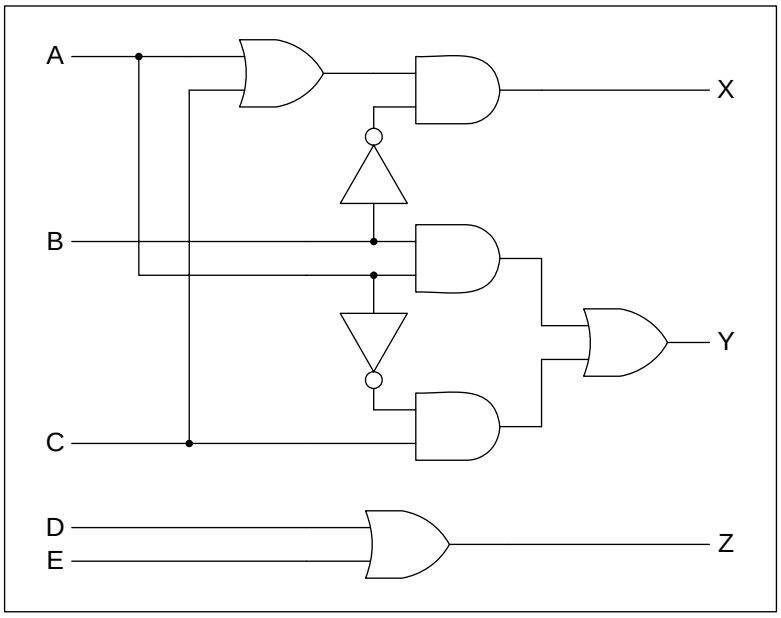
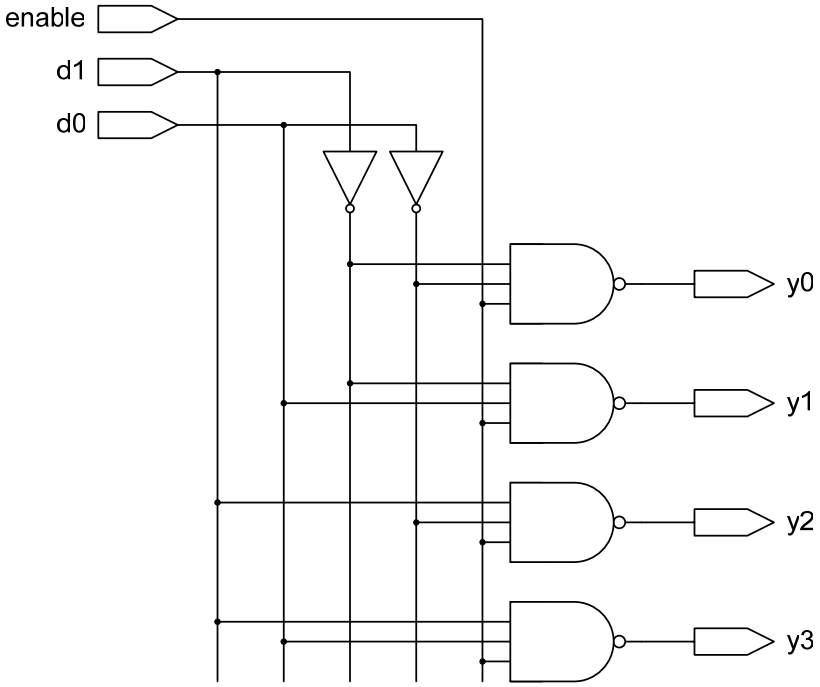
“Simple”




```
ENTITY  simple  IS
PORT (
          a, b, c, d, e            :IN BIT;
          x, y, z                  :OUT BIT);
END  simple;

ARCHITECTURE  example  OF  simple  IS
BEGIN
     --  signal assignment statements (Boolean)
     x  <=  (a OR c) AND NOT b;
     y  <=  (a AND b) OR (NOT a AND c);
     z  <=  d OR e;
END  example;
```

2-input Decoder



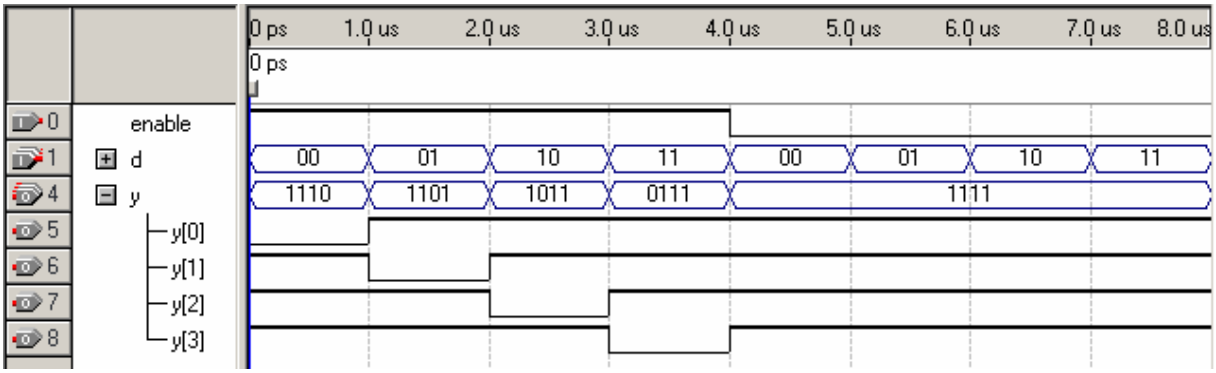

```
ENTITY decoder2x4 IS
PORT (
     d          :IN BIT_VECTOR (1 DOWNTO 0);
     enable     :IN BIT;
     y          :OUT BIT_VECTOR (0 TO 3)
);
END decoder2x4;


ARCHITECTURE boolean OF decoder2x4 IS
SIGNAL d1bar, d0bar      :BIT;     -- declare "buried" nodes

BEGIN
     -- write Boolean expressions for outputs
     -- using concurrent signal assignments
y(3) <= NOT (d(1) AND d(0) AND enable);
y(2) <= NOT (d(1) AND d0bar AND enable);
y(1) <= NOT (d1bar AND d(0) AND enable);
y(0) <= NOT (d1bar AND d0bar AND enable);
     -- decoder with active-low outputs
     -- order of statements does not matter
d1bar <= NOT d(1);
d0bar <= NOT d(0);

END boolean;
```

Simulation Results



```
ARCHITECTURE conditional OF decoder2x4 IS
BEGIN
-- conditional signal assignment statement
     y <= "0111"    WHEN (d = "00" AND enable = '1') ELSE
          "1011"    WHEN (d = "01" AND enable = '1') ELSE
          "1101"    WHEN (d = "10" AND enable = '1') ELSE
          "1110"    WHEN (d = "11" AND enable = '1') ELSE
          "1111";
END conditional;
```

```
ARCHITECTURE selected OF decoder2x4 IS
SIGNAL    inputs    :BIT_VECTOR (2 DOWNTO 0);
     -- define new 3-bit signal array “buried” within entity
BEGIN
     inputs <= enable & d;
     -- concatenate bits together for new array
     -- selected signal assignment statement
     WITH inputs SELECT
          y <= "0111"    WHEN "100",
               "1011"    WHEN "101",
               "1101"    WHEN "110",
               "1110"    WHEN "111",
               "1111"    WHEN OTHERS;        -- default
END selected;
```

```
ARCHITECTURE behavior OF decoder2x4 IS
BEGIN
PROCESS (d, enable)
     -- sensitivity list, invoke process if listed inputs change
BEGIN
     IF enable = '1'     THEN
          CASE d IS
               WHEN "00" =>        y <= "0111";
               WHEN "01" =>        y <= "1011";
               WHEN "10" =>        y <= "1101";
               WHEN "11" =>        y <= "1110";
          END CASE;
     ELSE                          y <= "1111";
     END IF;

END PROCESS;
END behavior;
```

entity *entity-name* **is**
    port (*list-of-interface-ports*);
end *entity-name*;
-- this is a comment line

architecture *architecture-name* **of** *entity-name* **is**
    [architecture-item-declarations]
begin
    concurrent-statements:
        process-statement
        block-statement
        concurrent-procedure-call-statement
        concurrent-assertion-statement
        concurrent-signal-assignment-statement
        component-instantiation-statement
        generate-statement
end *architecture-name* ;

[*process-label*:] **process** [(*sensitivity-list*)] [**is**]
    [process-item-declarations]
begin
    sequential-statements:
        variable-assignment-statement
        signal-assignment-statement
        wait-statement
        if-statement
        case-statement
        loop-statement
        null-statement
        exit-statement
        next-statement
        assertion-statement
        report-statement
        procedure-call-statement
        return-statement
end process [*process-label*] ;

```
     -- 2-channel multiplexer
ENTITY mux2to1 IS
PORT (a, b          :IN BIT;
      s             :IN BIT;
      y             :OUT BIT);
END mux2to1;

ARCHITECTURE ex1 OF mux2to1 IS
BEGIN
     y <= a WHEN s = '0' ELSE b;
     -- conditional signal assignment
END ex1;
```

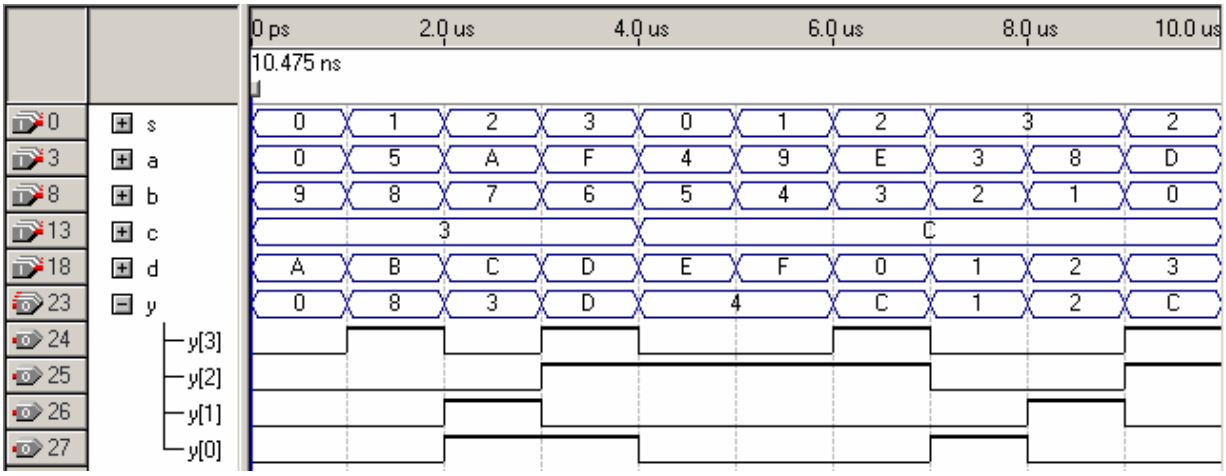
```
ARCHITECTURE ex2 OF mux2to1 IS
BEGIN
PROCESS (a, b, s)
BEGIN
     IF s = '0'     THEN y <= a;        -- IF statement
     ELSE                y <= b;
     END IF;
END PROCESS;
END ex2;
```

```
     -- 2-channel, 4-bit multiplexer
ENTITY mux2to1 IS
PORT (a, b          :IN BIT_VECTOR (3 DOWNTO 0);
      s             :IN BIT;
      y             :OUT BIT_VECTOR (3 DOWNTO 0));
END mux2to1;
```

```
ENTITY mux4to1 IS          -- 4-channel, 4-bit MUX
PORT (a, b           :IN BIT_VECTOR (3 DOWNTO 0);
      c, d           :IN BIT_VECTOR (3 DOWNTO 0);
      s              :IN BIT_VECTOR (1 DOWNTO 0);
      y              :OUT BIT_VECTOR (3 DOWNTO 0));
END mux4to1;

ARCHITECTURE ex3 OF mux4to1 IS
BEGIN
PROCESS (a, b, c, d, s)
BEGIN
     CASE s IS
          WHEN “00”      =>        y <= a;
          WHEN “01”      =>        y <= b;
          WHEN “10”      =>        y <= c;
          WHEN “11”      =>        y <= d;
     END CASE;
END PROCESS;
END ex3;
```

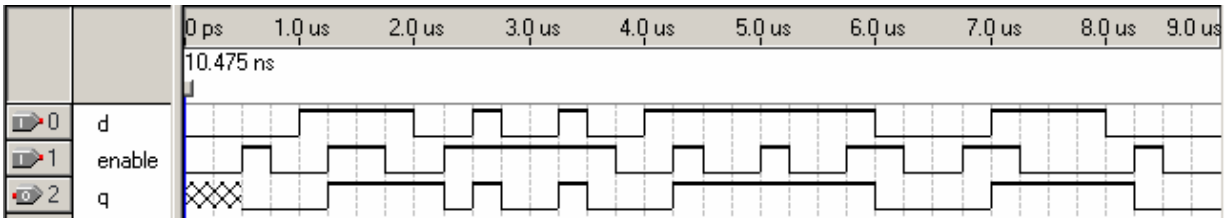
4-channel, 4-bit MUX Simulation



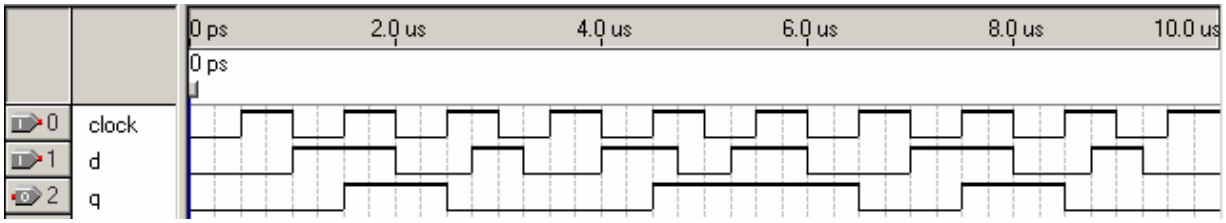
```
     -- transparent latch
ENTITY d_latch IS
PORT (d, enable          :IN BIT;
      q                  :OUT BIT);
END d_latch;
ARCHITECTURE level_enabled OF d_latch IS
BEGIN
PROCESS (d, enable)
BEGIN
          -- enabled with high logic level
     IF enable = '1'     THEN      q <= d;
     END IF;
          -- memory is implied, no else statement
END PROCESS;
END level_enabled;
```

```
     -- clocked D flip-flop
ENTITY dflipflop IS
PORT (d, clock           :IN BIT;
      q                  :OUT BIT);
END dflipflop;
ARCHITECTURE edge_triggered OF dflipflop IS
BEGIN
PROCESS (clock)
BEGIN
          -- triggers on rising edge
     IF (clock = '1' AND clock'EVENT)   THEN      q <= d;
     END IF;                  -- 'EVENT signal attribute
END PROCESS;
END edge_triggered;
```

Transparent D Latch Simulation



Clocked D Flip-flop Simulation



```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
     -- using Standard Logic

ENTITY dflipflop IS
PORT (d, clock           :IN STD_LOGIC;
      q                  :OUT STD_LOGIC);
END dflipflop;

ARCHITECTURE edge_trig OF dflipflop IS
BEGIN
PROCESS (clock)
BEGIN
IF (RISING_EDGE(clock))  THEN      q <= d;
     -- also FALLING_EDGE() detection
END IF;
END PROCESS;
END edge_trig;
```

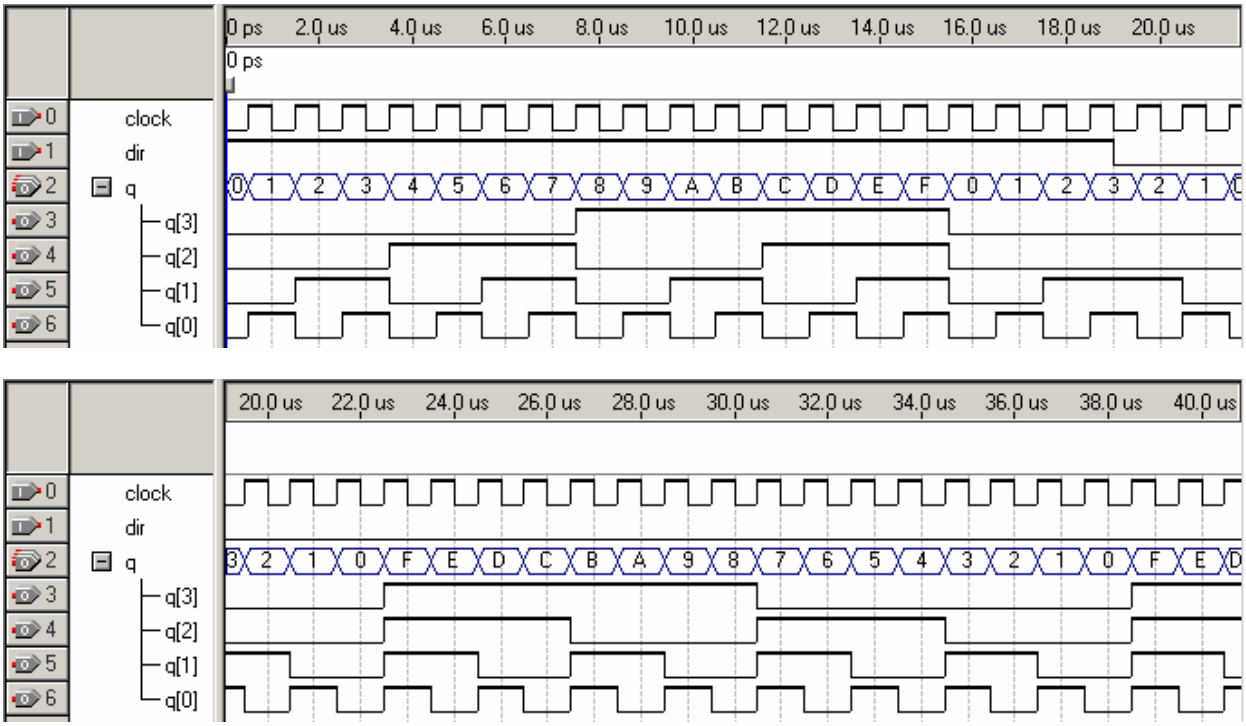
```
ENTITY d_ff IS
PORT (data, clock, preset, clear   :IN BIT;
      q_out                        :OUT BIT);
END d_ff;

ARCHITECTURE asynchronous OF d_ff IS
BEGIN
PROCESS (clock, preset, clear)
BEGIN
     -- priority order: clear, preset, clock
     IF (clear = '0')     THEN     q_out <= '0';
     ELSIF (preset = '0') THEN     q_out <= '1';
     ELSIF (clock'EVENT AND clock = '0') THEN
                                   q_out <= data;
     END IF;
END PROCESS;
END asynchronous;
```

```
     -- mod-16 up/down binary counter
ENTITY updncntr IS
PORT (
     clock, dir     :IN BIT;
     q              :OUT INTEGER RANGE 0 TO 15
                -- output q will need 4 bits
);
END updncntr;

ARCHITECTURE binary OF updncntr IS
BEGIN
PROCESS (clock)          -- clock change?
     VARIABLE count      :INTEGER RANGE 0 TO 15;
     -- create a variable for counter value
BEGIN
     IF (clock'EVENT AND clock = '1')   THEN
          IF  (dir = '1')  THEN    -- count up
                    count := count + 1;
          ELSE                     -- count down
                    count := count - 1;
          END IF;
     END IF;             -- count holds with no clock
     q <= count;         -- send count value to output port
END PROCESS;
END binary;
```

Mod-16 Up/Down Binary Counter Simulation



```
ENTITY mod10 IS
PORT (
     clock, enable       :IN BIT;
     load, clear         :IN BIT;
     d                   :IN INTEGER RANGE 0 TO 15;
     q                   :OUT INTEGER RANGE 0 TO 15;
     rco                 :OUT BIT
);
END mod10;

ARCHITECTURE bcd OF mod10 IS
BEGIN

PROCESS (clock, clear, enable)     -- asynchronous clear
     VARIABLE counter    :INTEGER RANGE 0 TO 15;
BEGIN
     IF (clear = '1') THEN         counter := 0;
               -- asynchronous clear has priority
     ELSIF (clock'EVENT AND clock = '1') THEN
          IF (load = '1') THEN     counter := d;
                              -- synchronous load
          ELSIF  (enable = '1') THEN
               IF (counter = 9) THEN    -- recycle
                                   counter := 0;
               ELSE                counter := counter + 1;
               END IF;        -- hold count behavior is implied
          END IF;
     END IF;

     -- rco detects terminal count when enabled
     IF ((counter = 9) AND (enable = '1')) THEN
                         rco <= '1';
     ELSE           rco <= '0';
     END IF;

q <= counter;            -- output counter to ports
END PROCESS;
END bcd;
```

BCD Counter Simulation