D Latches and D Flip-Flops

Two important digital circuits are the D latch and the D flip-flop, both of which are used to store data. The main difference between the circuits is the conditions under which the data are stored. The symbols for these circuits are shown in **Figure 1**.

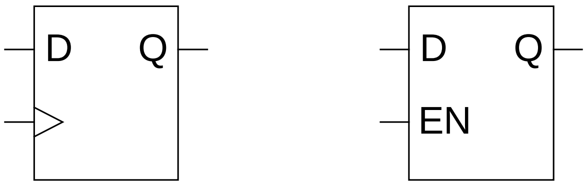

Figure 1 D Flip-Flop and D Latch

In a D latch (or "transparent latch"), the output Q takes on the value of the input D only when an enable input (EN) is HIGH. (We say, "Q follows D".) If D changes while EN = 1, Q will follow immediately. If EN = 0, Q retains its previous value and does not change with D.

In a flip-flop, Q also follows D, but only when there is a transition on an enabling input called the clock (CLK). Typically, the clock input is shown as a triangle, indicating that it is a dynamic input. For a positive edge-triggered D flip-flop, Q follows D when the clock makes a transition from LOW to HIGH. Otherwise, Q will not change.

Latches and flip-flops can also have multiple inputs and outputs, as shown in **Figure 2**. For these devices, all Qs follow all Ds on the appropriate enable condition. For example, if $D_3D_2D_1D_0$ = 0101, then $Q_3Q_2Q_1Q_0$ will equal 0101 after the first positive clock edge for the flip-flop or as soon as EN = 1 for the latch.

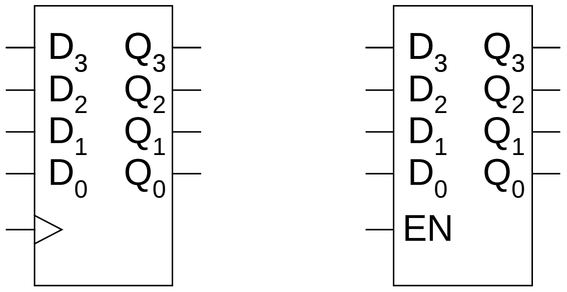

Figure 2 4-bit D Flip-Flop and Latch

VHDL implementation of Latches and Flip-Flops

There are a number of different ways to implement multi-bit latches and flip-flops in VHDL. Several methods for implementing latches are detailed in Chapter 8 of *Digital Design with CPLD Applications and VHDL, 2/e*.

For a behavioral description of a D latch, an IF statement in a PROCESS is written to indicate the “Q follows D” property of the latch:

```
PROCESS(en, d)
BEGIN
   IF(en = '1')THEN
       q <= d;
   END IF;
END PROCESS;
```

The PROCESS statement tests for a change in either **en** or **d**, since a change in either one of these alone can make **q** change.

A D flip-flop can be implemented in a similar way:

```
  PROCESS(clk)
  BEGIN
     IF(clk'EVENT and clk='1')THEN
         q <= d;
     END IF;
END PROCESS;
```

The construct `clk'EVENT` (pronounced “clock tick event”) indicates that a change has just happened on the clock. This, combined with the test for a logic HIGH on the clock, indicates that a positive clock edge has just occurred. The PROCESS statement tests only for a change in **clk**, since no change of **q** is possible without a positive clock edge. Input **d** is not tested because it cannot make **q** change on its own.

In both these structures, the ports Q and D can be single-variable ports (type BIT or STD_LOGIC) or multiple-bit ports (type BIT_VECTOR or STD_LOGIC_VECTOR).

A D latch description can be altered to include an active-LOW asynchronous reset function, as follows:

```
PROCESS (en, d, reset)
BEGIN
   IF (reset = '0') THEN
       q <= (others => '0');
   ELSIF (en = '1') THEN
       q <= d;
   END IF;
END PROCESS;
```

Since the reset input is checked first, it takes priority over the enable input, which is only checked if the reset is not active. The construct `q <= (others => '0');`is a way to set all bits of a vector to 0, regardless of the vector size. This allows us to clear an entire vector without knowing its size. A reset can be added to a flip-flop in a similar way.

VHDL syntax for IF statement:

```
if(condition1)then
   statements
elsif(condition2)then
   statements
elsif(condition3)then
   statements
else
   statements
end if;
```

VHDL syntax for Minimal IF statement:

```
if(condition1)then
   statements
end if;
```

Write IF statements to ensure only one possible outcome

```
-- Example 1: Not good!
-- If reset=0 and clock edge present, reset, then transfer d to q
-- (Not really resetting, is it?)
process(reset, clock)
begin
   if(reset = '0')then
      q <= (others => '0');
   end if;
   if(clock'event and clock = '1')then
      q <= d;
   end if;
end process;
```

```
-- Example 2: only one possible outcome of process
-- Test for reset. If no reset, then test for clock edge.
process(reset, clock)
begin
   if(reset = '0')then
      q <= (others => '0');
   elsif(clock'event and clock = '1')then
      q <= d;
   end if;
end process;
```