Latches

A latch is a level sensitive memory cell that is transparent to signals passing from the D input to Q output when enabled, and holds the value of D on Q at the time when it becomes disabled. The Q-bar output signal is always the inverse of the Q output signal, see Figure 1.

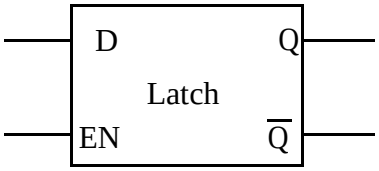

Figure 1 The level sensitive D-type transparent latch

inputs		outputs	
D	EN	Q_+	$\overline{Q}_+$
–	0	Q_-	Q_-
0	1	0	1
1	1	1	0

– =don't care

Table 1 Characteristic table of a D-type transparent latch

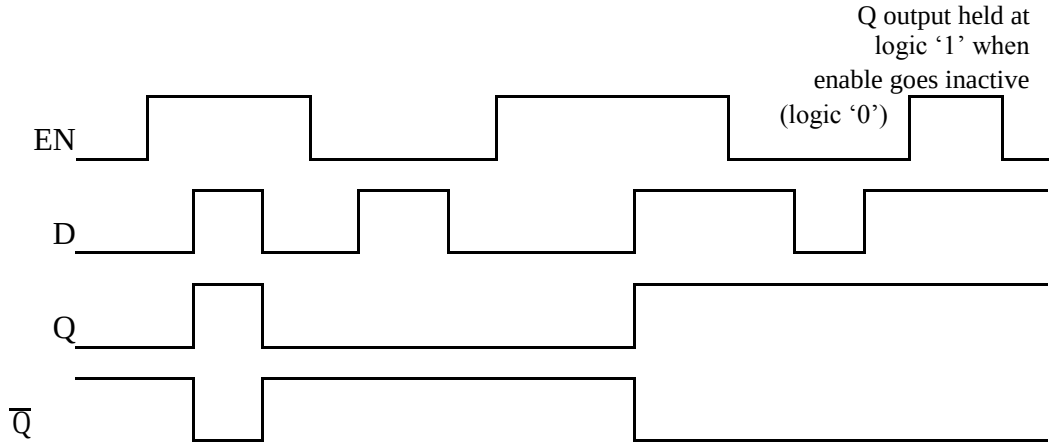

Figure 2 Simulation of the level sensitive D-type transparent latch

Example 1 Modeling of a D-type transparent latch
The **if** statement without **else** clause is the simplest implementation of D-type transparent latch.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
entity d_latch is
port (d, en: in std_logic;
q: out std_logic);
end entity d_latch;
architecture rtl of d_latch is
begin
```

```
process (d, en)
begin
if (en = '1') then
q <= d;
end if;
end process;
end architecture rtl;
```

D-Type Flip-Flop

The D-type flip-flop is an edge triggered memory device that transfers a signal's value on its D input, to its Q output, when an active edge transition occurs on its clock input. The output value is held until the next active clock edge. The Q-bar output signal is always the inverse of the Q output signal, see Figure 3.

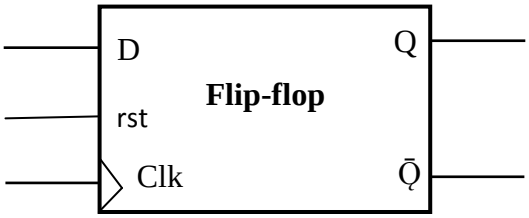

Figure 3 The level sensitive D-type flip-flop

Table 2 Characteristic table of a D-type flip-flop

inputs		outputs	
D	CLK	Q_+	$\overline{Q}_+$
0	↑	Q_-	Q_-
1	↑		
–	0	Q_-	$\bar{Q}_-$
–	1	Q_-	$\bar{Q}_-$

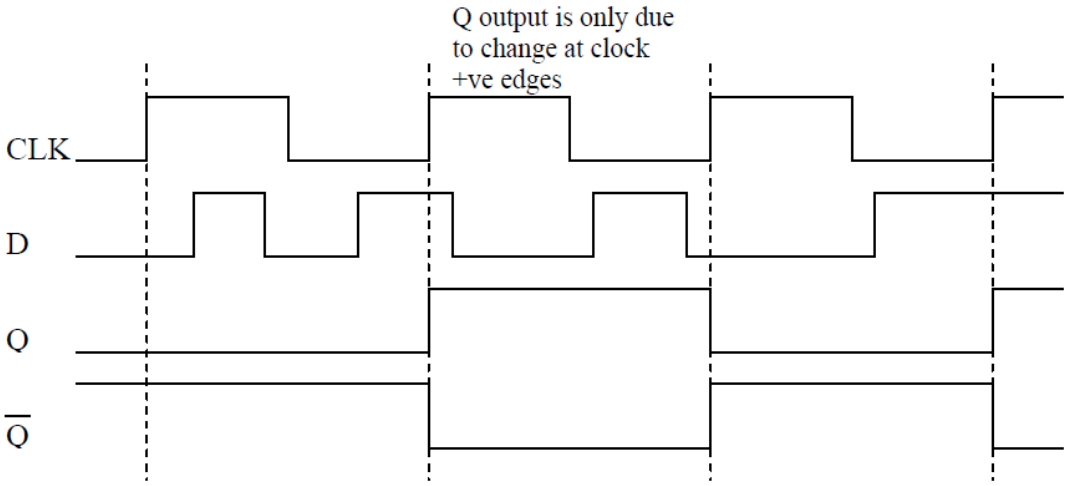

Figure 4 Simulation of a +ve edge sensitive D-type flip-flop

Example 2 Modeling flip-flops with reset

Different flip-flops with asynchronous and synchronous resets are modeled. The first architecture models a D-type flip-flop with an asynchronous reset input, this is the commonly used D-type flip-flop, while the second architecture models D-type flip flop with a synchronous reset as appears from the process sensitivity list and the **if** statements sequence.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
entity d_flip_flop is
port (d, clk, rst: in std_logic;
q: out std_logic);
end entity d_flip_flop;
architecture first of d_flip_flop is
begin
process (rst, clk)
begin
if (rst = '1') then
q <= '0';
elsif (rising_edge(clk)) then
q <= d;
end if;
end process;
end architecture first;
architecture second of d_flip_flop is
begin
process (clk)
begin
if (rising_edge(clk)) then
if (rst = '1') then
q <= '0';
else
q <= d;
end if;
end if;
end process;
end architecture second;
```

Example 3.16 Modeling flip-flop with enable and asynchronous reset

Flip-flop with enable input and asynchronous reset is modeled to illustrate the addition of the enable signal control, which is synchronous, accordingly with the asynchronous reset.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
entity d_flip_flop is
port (d, clk, rst, en: in std_logic;
q: out std_logic);
end entity d_flip_flop;
architecture rtl of d_flip_flop is
begin
process (rst, clk)
```

```
begin
if (rst = '1') then
q <= '0';
elsif (rising_edge(clk)) then
if (en = '1') then
q <= d;
end if;
end if;
end process;
end architecture rtl;
```

Parallel Registers

A parallel register is simply a bank of D-type flip-flops, its implementation has the same structure as for one flip-flop but with extended inputs and outputs, see Figure 3.9.

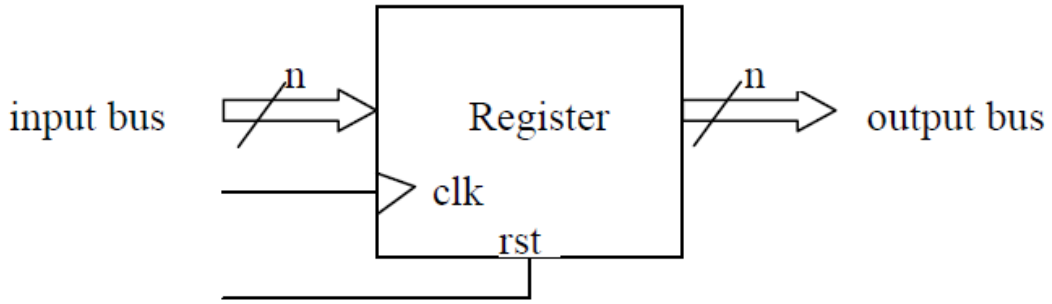

Figure 3.1 Register with n inputs and n outputs

Example 3 Modeling of a parallel 8 bits register

A typical example for a parallel 8 bits register is modeled, based on the flip-flop model shown previously but with inputs and outputs are extended as buses.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
entity reg_par is
port (clk, rst: in std_logic;
reg_in: in std_logic_vector(7 downto 0);
reg_out: out std_logic_vector(7 downto 0));
end entity reg_par;
architecture rtl of reg_par is
begin
process (rst, clk)
begin
if (rst = '1') then
reg_out <= "00000000";
elsif (rising_edge(clk)) then
reg_out <= reg_in;
end if;
```

```
end process;
end architecture rtl;
```

Example 3.18 Modeling of a parallel 8 bits register with enable

The same as the previous example but with an added enable signal to provide control over the register, so that the register only register the value at its inputs when enable is active, otherwise remaining the contents unchanged. Also, an **others** statement in the reset action line is added to fill registers with any size with zeros. This model is also based on the flip-flop model shown in example 3.10.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
entity reg_par is
port (clk, rst, en: in std_logic;
reg_in: in std_logic_vector(7 downto 0);
reg_out: out std_logic_vector(7 downto 0));
end entity reg_par;
architecture rtl of reg_par is
begin
process (rst, clk)
begin
if (rst = '1') then
reg_out <= (others => '0');
elsif (rising_edge(clk)) then
if (en = '1') then
reg_out <= reg_in;
end if;
end if;
end process;
end architecture rtl;
```