DIGITAL LOGIC WITH VHDL
(Fall 2013)
Unit 5

✓ *SEQUENTIAL CIRCUITS*

- Asynchronous sequential circuits: Latches
- Synchronous circuits: flip flops, counters, registers.

Instructor: Daniel Llamocca

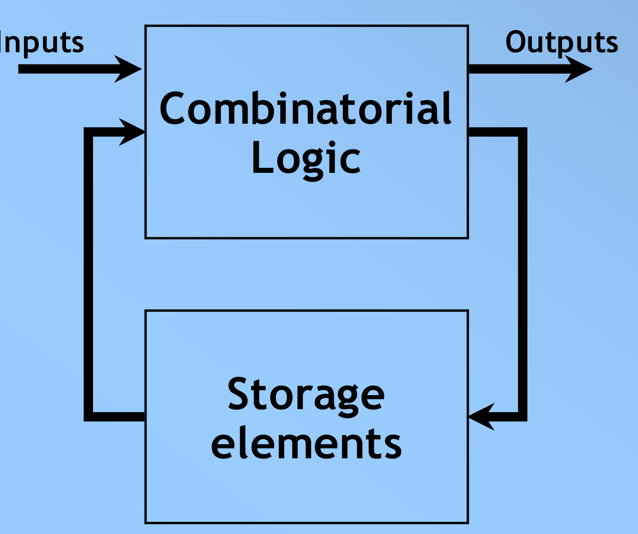
✓ COMBINATORIAL CIRCUITS

- In combinatorial circuits, the output only depends upon the present input values.
- There exist another class of logic circuits whose outputs not only depend on the present input values but also on the past values of inputs, outputs, and/or internal signal. These circuits include storage elements to store those previous values.
- The content of those storage elements represents the ***circuit state***. When the circuit inputs change, it can be that the circuit stays in certain state or changes to a different one. Over time, the circuit goes through a sequence of states as a result of a change in the inputs. The circuits with this behavior are called ***sequential circuits***.

COMBINATORIAL CIRCUIT

Inputs → **Combinatorial Logic** → Outputs

SEQUENTIAL CIRCUIT



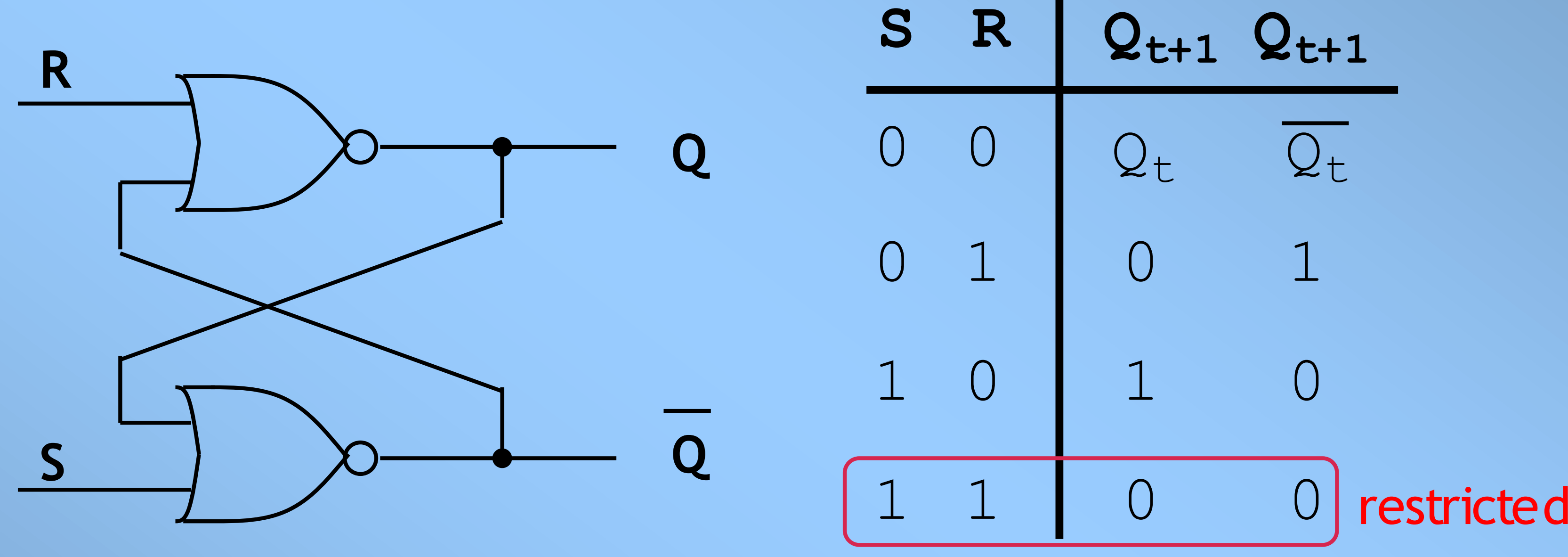
✓ *SEQUENTIAL CIRCUITS*

- Combinatorial circuits can be described with concurrent statement, or behavioral statements.
- Sequential circuits are ***best described*** with sequential statements.
- Sequential circuits can either be synchronous or asynchronous. In VHDL, they are described with asynchronous/synchronous processes.
- Basic asynchronous sequential circuit: Latch
- Basic synchronous sequential circuits: flip flops, counters, and registers.

- We will now go over the VHDL description of sequential circuits.

✓ ASYNCHRONOUS PROCESS (LATCHES)

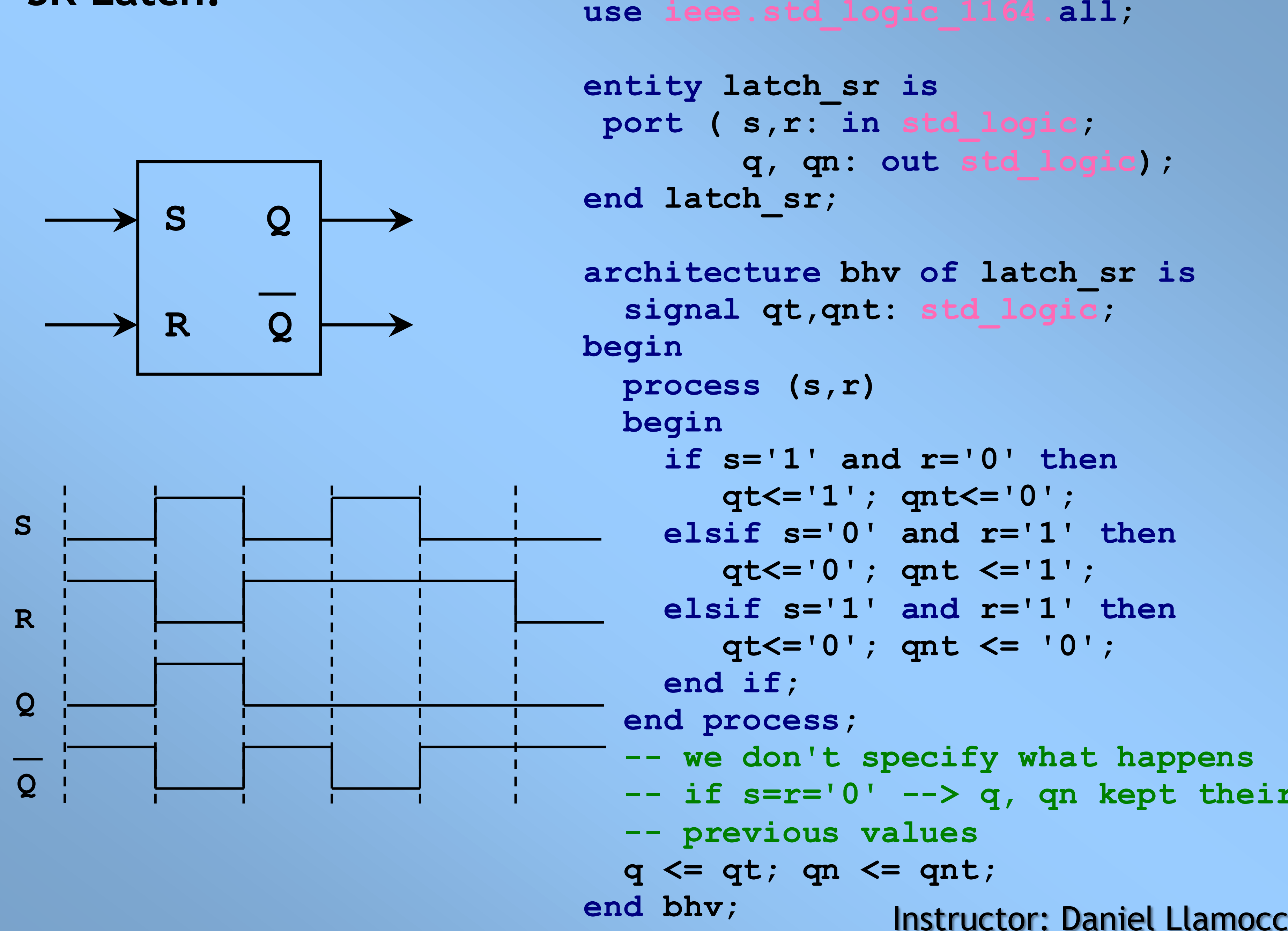
- **SR Latch**
- An SR Latch based on NOR gates::



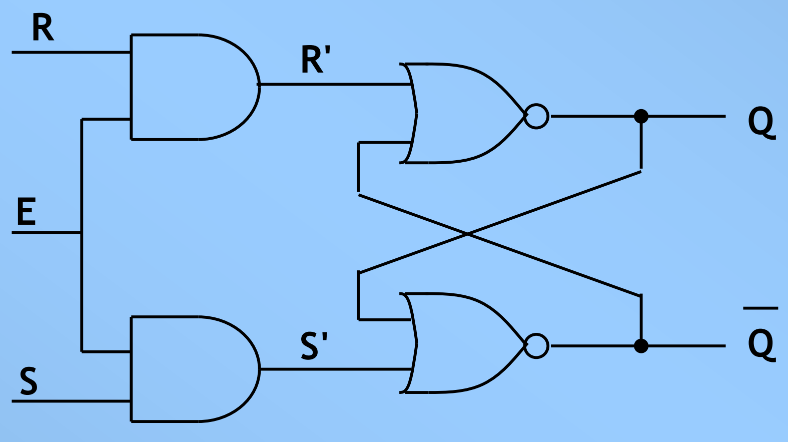
S	R	Q_{t+1}	$\overline{Q}_{t+1}$
0	0	Q_t	$\overline{Q_t}$
0	1	0	1
1	0	1	0
1	1	0	0

- According to its truth table, the output can be assigned to either ‘0’ or ‘1’. This circuit state (‘0’ or ‘1’) is stored in the circuit when S=R=‘0’.

SR Latch:



- SR Latch with enable

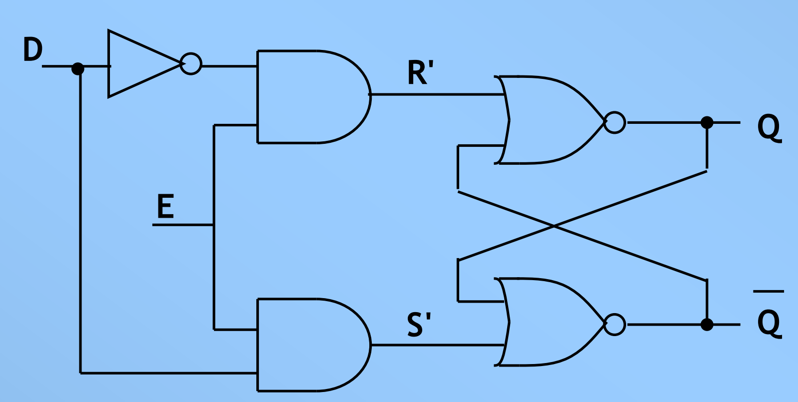


E	S	R	Q_{t+1}	$\overline{Q_{t+1}}$
0	x	x	Q_t	$\overline{Q_t}$
1	0	0	Q_t	$\overline{Q_t}$
1	0	1	0	1
1	1	0	1	0
1	1	1	0	0

- Note: If E = ‘0’, the previous output is kept.

```
library ieee;
use ieee.std_logic_1164.all;

entity latch_sr_E is
 port ( s,r, E: in std_logic;
        q, qn: out std_logic);
end latch_sr_E;

architecture bhv of latch_sr_E is
  signal qt,qnt: std_logic;
begin
  process (s,r)
  begin
    if E = '1' then
      if s='1' and r='0' then
         qt<='1'; qnt<='0';
      elsif s='0' and r='1' then
         qt<='0'; qnt <='1';
      elsif s='1' and r='1' then
         qt<='0'; qnt <= '0';
      end if;
    end if;
  end process;
  q <= qt; qn <= qnt;
end bhv;
```

- D Latch D with enable

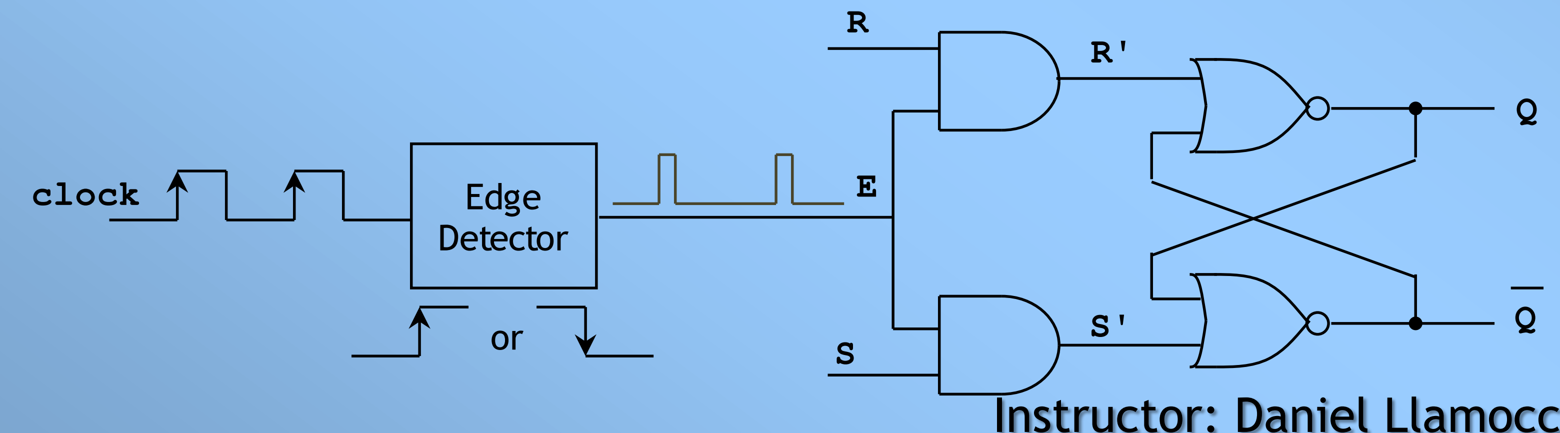


E	D	Q_{t+1}
0	x	Q_t
1	0	0
1	1	1

```
library ieee;
use ieee.std_logic_1164.all;

entity latch_D is
 port ( D, E: in std_logic;
        q, qn: out std_logic);
end latch_D;

architecture bhv of latch_D is
  signal qt: std_logic;
begin
  process (D,E)
  begin
    if E = '1' then
       qt <= d;
    end if;
  end process;
  q <= qt; qn <= not(qt);
end bhv;
```

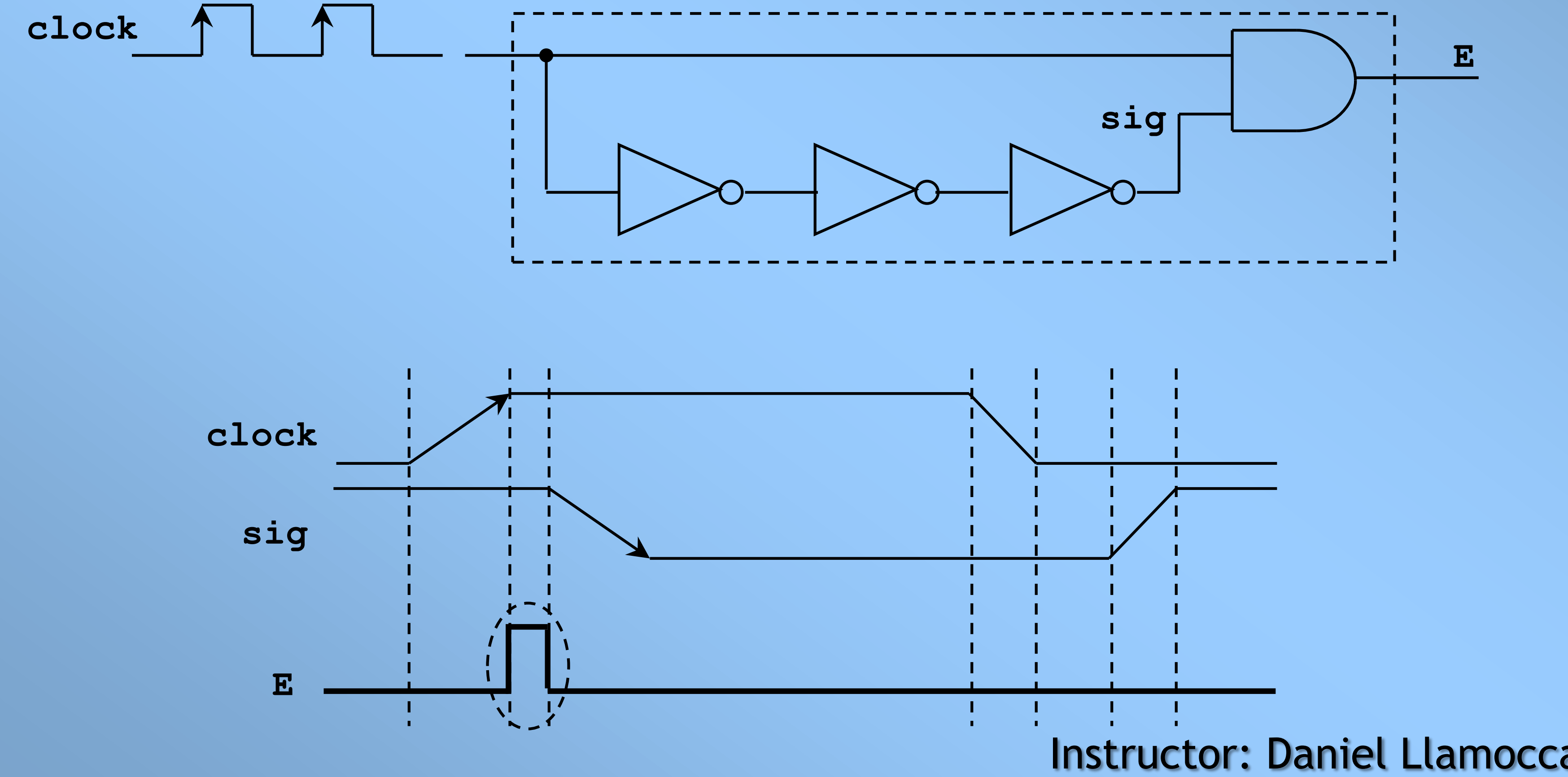
✓ SYNCHRONOUS PROCESSES

- **Flip Flops**
- Unlike a Latch, a flip flop only changes its outputs on the edge (rising or falling) of a signal called ***clock***. A ***clock*** signal is an square wave with a fixed frequency.
- To detect a rising or falling edge, flip flops include an edge detector circuit. Input: a ***clock*** signal, Output: short duration pulses during the rising (or falling) clock edges. These pulses are then connected to the enable input in a Latch.
- For example, an SR flip flop is made out of: a SR Latch with an edge detector circuit. The edge detector generates enable signals during the during the rising (or falling) clock edges.



✓ SYNCHRONOUS PROCESSES

- **Flip Flops:** The edge detector circuit generates E='1' during the edge (rising or falling). We will work with circuits activated by either rising or falling edge. We will not work with circuits activated by both edges.
- An example of a circuit that detects a rising edge is shown below. The redundant NOT gates cause a delay that allows a pulse to be generated during a rising edge (or positive edge).



✓ SYNCHRONOUS PROCESSES

- **SR Flip Flop**

```
library ieee;
use ieee.std_logic_1164.all;

entity ff_sr is
 port ( s,r, clock: in std_logic;
        q, qn: out std_logic);
end ff_sr;

architecture bhv of ff_sr is
  signal qt,qnt: std_logic;
begin
  process (s,r,clock)
  begin
   if (clock'event and clock='1') then
   --if (clock'event and clock='0') then
      if s='1' and r='0' then
         qt<='1'; qnt<='0';
      elsif s='0' and r='1' then
         qt<='0'; qnt <='1';
      elsif s='1' and r='1' then
         qt<='0'; qnt <= '0';
      end if;
   end if;
  end process;
  q <= qt; qn <= qnt;
end bhv;
```

S Q clock R $\overline{Q}$
Positive edge-triggered

S Q clock R $\overline{Q}$
Negative edge-triggered

Positive-edge triggered →

Negative-edge triggered →

R E S clock Edge Detector Q $\overline{Q}$

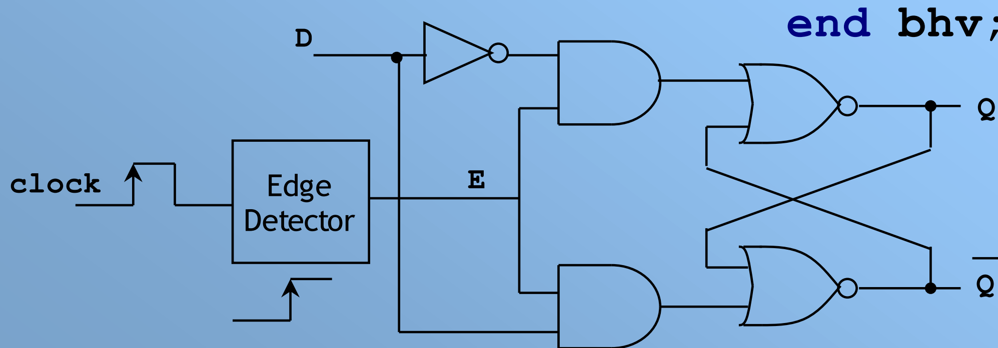
Instructor: Daniel Llamocca

✓ SYNCHRONOUS PROCESSES

- **D Flip Flop**

clock	D	Q_{t+1}
↑	0	0
↑	1	1

```vhdl
library ieee;
use ieee.std_logic_1164.all;

entity ff_d is
 port ( d, clock: in std_logic;
        q, qn: out std_logic);
end ff_d;

architecture bhv of ff_d is
  signal qt,qnt: std_logic;
begin
  process (d,clock)
  begin
    if (clock'event and clock='1') then
       qt<=d;
    end if;
  end process;
  q <= qt; qn <= not(qt);
end bhv;
```



Instructor: Daniel Llamocca

✓ SYNCHRONOUS PROCESSES

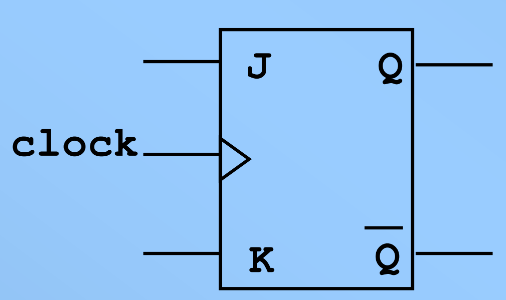
- **T Flip Flop**

clock	T	Q_{t+1}
↑	0	Q_t
↑	1	$\overline{Q_t}$

```
library ieee;
use ieee.std_logic_1164.all;

entity ff_t is
 port ( t, clock: in std_logic;
        q, qn: out std_logic);
end ff_t;

architecture bhv of ff_t is
  signal qt,qnt: std_logic;
begin
  process (t,clock)
  begin
    if (clock'event and clock='1') then
       if t = '1' then
          qt <= not(qt);
       end if;
    end if;
  end process;
  q <= qt; qn <= not(qt);
end bhv;
```

✓ SYNCHRONOUS PROCESSES

- **JK Flip Flop**

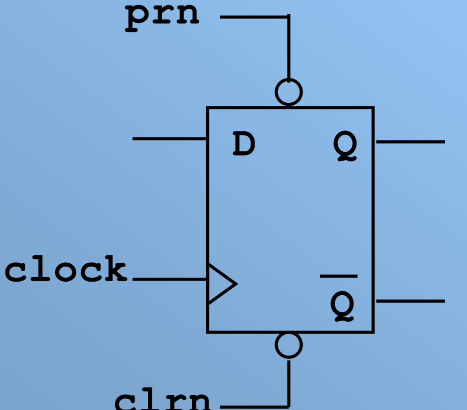


clock	J	K	Q_{t+1}
↑	0	0	Q_t
↑	0	1	0
↑	1	0	1
↑	1	1	$\overline{Q_t}$

```
library ieee;
use ieee.std_logic_1164.all;

entity ff_jk is
 port ( s,r, clock: in std_logic;
        q, qn: out std_logic);
end ff_jk;

architecture bhv of ff_jk is
  signal qt,qnt: std_logic;
begin
  process (j,k,clock)
  begin
    if (clock'event and clock='1') then
      if j='1' and k='1' then
         qt<= not(qt);
      elsif j='1' and k='0' then
         qt<='0';
      elsif j='0' and k='1' then
         qt<='1';
      end if;
    end if;
  end process;
  q <= qt; qn <= qnt;
end bhv;
```

✓ SYNCHRONOUS PROCESSES

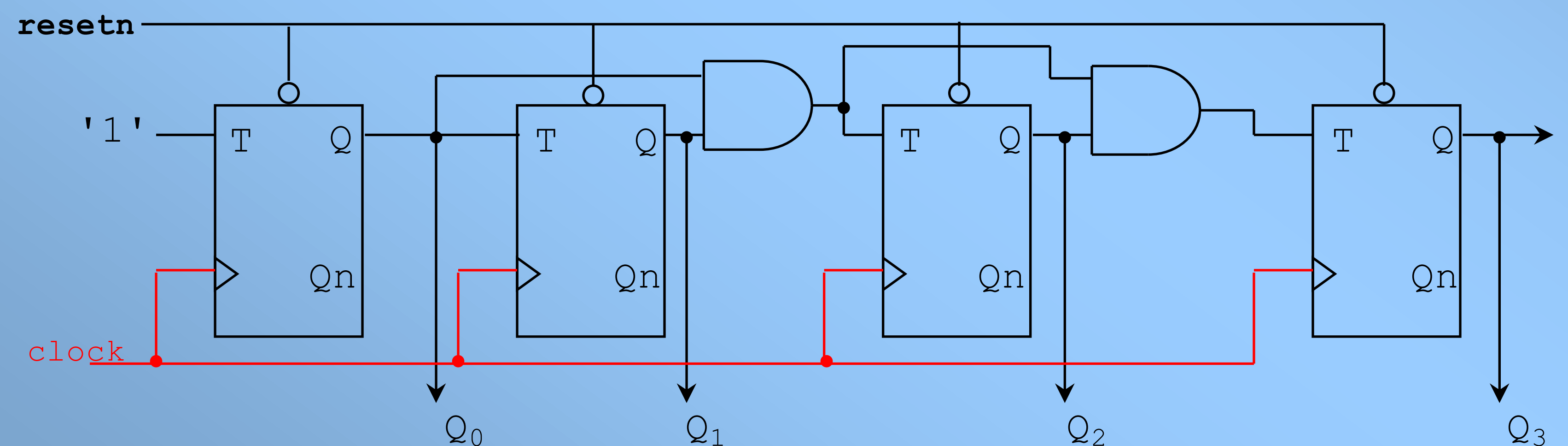
- **D Flip Flop D with asynchronous inputs: ‘clrn’, ‘prn’**
  - clrn = ‘0’ ➔ q = ‘0’
    prn = ‘0’ ➔ q = ‘1’
  - This inputs force the outputs to a value immediately.
  - This is a useful feature if we want to initialize the circuit with no regards to the rising (or falling) clock edge




```
library ieee;
use ieee.std_logic_1164.all;

entity ff_dp is
 port ( d,clrn,prn,clock: in std_logic;
        q, qn: out std_logic);
end ff_dp;

architecture bhv of ff_dp is
  signal qt,qnt: std_logic;
begin
  process (d,clrn,prn,clock)
  begin
    if clrn = '0' then
       qt <= '0';
    elsif prn = '0' then
       qt <= '1';
    elsif (clock'event and clock='1') then
       qt <= d;
    end if;
  end process;
  q <= qt; qn <= not(qt);
end bhv;
```

✓ SYNCHRONOUS PROCESSES

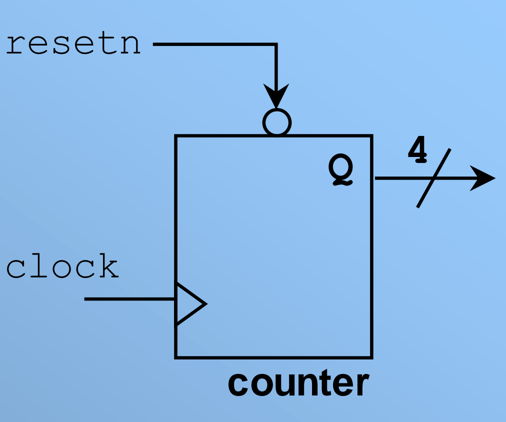
- **Synchronous Counters**
- Counters are very useful in digital systems. They can count the number of occurrences of a certain event, generate time intervals for task control, track elapsed time between two events, etc.
- Synchronous counters change their output on the clock edge (rising or falling). Counters are made of flip flops and combinatorial logic. Every flip flop in a synchronous counter shares the same clock signal. The figure shows a 4-bit synchronous counter (0000 to 1111). A 'resetn' signal is also included to initialize the count.



Instructor: Daniel Llamocca

✓ 4-bit Synchronous counter with asynchronous active-low reset

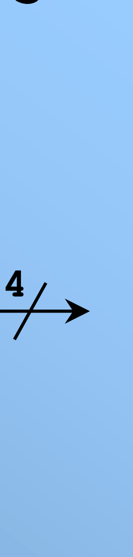
- The 'resetn' signal sets the output to "0000" disregarding the clock edge




```vhdl
library ieee;
use ieee.std_logic_1164.all;

entity my_count4b is
 port ( clock, resetn: in std_logic;
        Q: out integer range 0 to 15);
end my_count4b;

architecture bhv of my_count4b is
  signal Qt: integer range 0 to 15;
begin
  process (resetn,clock)
  begin
    if resetn = '0' then
       Qt <= 0;
    elsif (clock'event and clock='1') then
       Qt <= Qt + 1;
    end if;
  end process;
  Q <= Qt;
end bhv;
```

✓ 4-bit Synchronous counter with asynchronous active-low reset and enable

- Note that the enable signal ‘E’ is synchronous, thus it is only considered on the rising clock edge

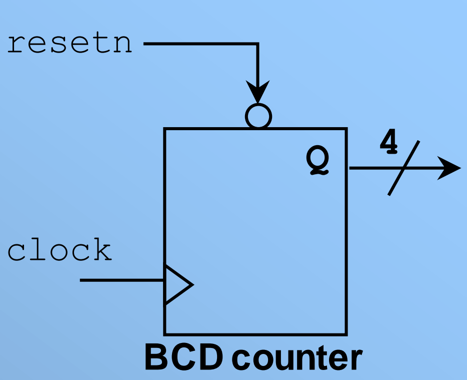



```
library ieee;
use ieee.std_logic_1164.all;

entity my_count4b_E is
 port ( clock, resetn, E: in std_logic;
        Q: out integer range 0 to 15);
end my_count4b_E;

architecture bhv of my_count4b_E is
  signal Qt: integer range 0 to 15;
begin
  process (resetn,clock, E)
  begin
    if resetn = '0' then
       Qt <= 0;
    elsif (clock'event and clock='1') then
       if E = '1' then
          Qt <= Qt + 1;
       end if;
    end if;
  end process;
  Q <= Qt;
end bhv;
```

Instructor: Daniel Llamocca

✓ 4-bit Synchronous BCD counter with asynchronous active-low reset

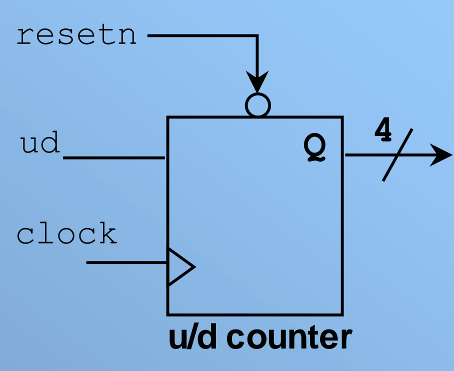
- It counts from 0000 to 1001




```
library ieee;
use ieee.std_logic_1164.all;

entity my_bcd_count is
 port ( clock, resetn: in std_logic;
        Q: out integer range 0 to 15);
end my_bcd_count;

architecture bhv of my_bcd_count is
  signal Qt: integer range 0 to 15;
begin
  process (resetn,clock)
  begin
    if resetn = '0' then
       Qt <= 0;
    elsif (clock'event and clock='1') then
       if Qt = 9 then
          Qt <= 0;
       else
          Qt <= Qt + 1;
       end if;
    end if;
  end process;
  Q <= Qt;
end bhv;
```

✓ 4-bit Synchronous up/down counter with asynchronous active-low reset

- ud = 0 -> down
- ud = 1 -> up
- When `Qt = 0000`, then `Qt <= Qt-1` will result in `Qt = 1111`

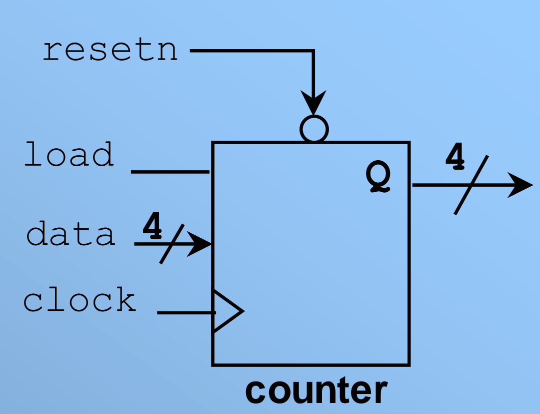



```
library ieee;
use ieee.std_logic_1164.all;

entity my_ud_count is
 port ( clock, resetn,ud: in std_logic;
        Q: out integer range 0 to 15);
end my_ud_count;

architecture bhv of my_ud_count is
  signal Qt: integer range 0 to 15;
begin
  process (resetn,clock,ud)
  begin
    if resetn = '0' then
       Qt <= 0;
    elsif (clock'event and clock='1') then
       if ud = '0' then
          Qt <= Qt - 1;
       else
          Qt <= Qt + 1;
       end if;
    end if;
  end process;
  Q <= Qt;
end bhv;
```

✓ 4-bit Synchronous counter with parallel load

- Here, we use Q as vector.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity my_lcount is
 port ( clock, resetn,load: in std_logic;
        data: in std_logic_vector(3 downto 0);
        Q: out std_logic_vector(3 downto 0));
end my_lcount;

architecture bhv of my_lcount is
  signal Qt: std_logic_vector(3 downto 0);
begin
  process (resetn,clock,load)
  begin
    if resetn = '0' then
      Qt <= "0000";
    elsif (clock'event and clock='1') then
      if load = '1' then
         Qt <= data;
      else
         Qt <= Qt + "0001";
      end if;
    end if;
  end process;
  Q <= Qt;
end bhv;
```



✓ *SYNCHRONOUS PROCESSES*

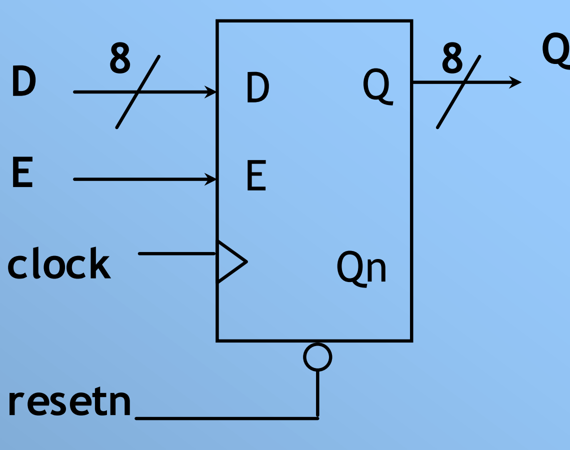
- **Registers**
- These are sequential circuits that store the values of signals. There exist many register types: registers to handle interruptions in a PC, microprocessor registers, etc.
- A register that can hold 'n' bits is a collection of 'n' D-type flip flops
- Register types:
  - Simple Register (with/without enable)
  - Parallel access shift register (parallel output/serial output0.
  - Shift register (with/without enable)

    Serial input, parallel output

    Serial input, serial outputl

✓ PARALLEL LOAD, PARALLEL OUTPUT

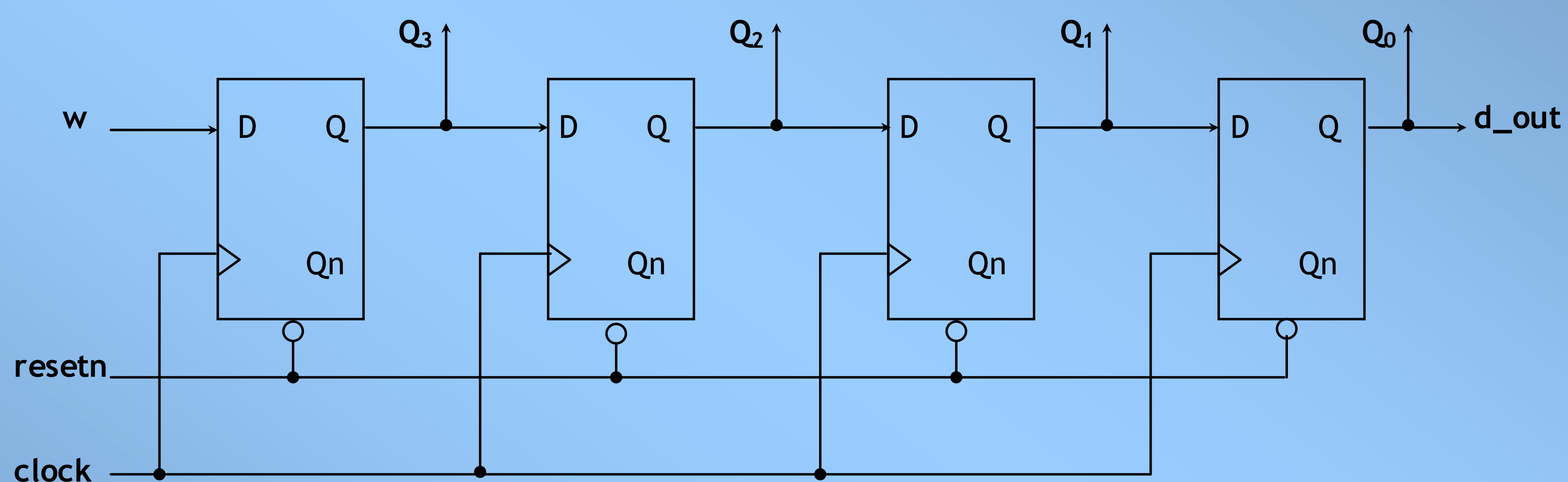
- **8-bit register with enable and asynchronous reset**




```
library ieee;
use ieee.std_logic_1164.all;

entity reg8 is
 port (clock, resetn, E: in std_logic;
  D: in std_logic_vector (7 downto 0);
  Q: out std_logic_vector (7 downto 0));
end reg8;

architecture bhv of reg8 is
begin
 process (resetn,E,clock)
 begin
  if resetn = '0' then
   Q <= (others => '0');
  elsif (clock'event and clock = '1') then
   if E = '1' then
     Q <= D;
   end if;
  end if;
 end process;
end bhv;
```

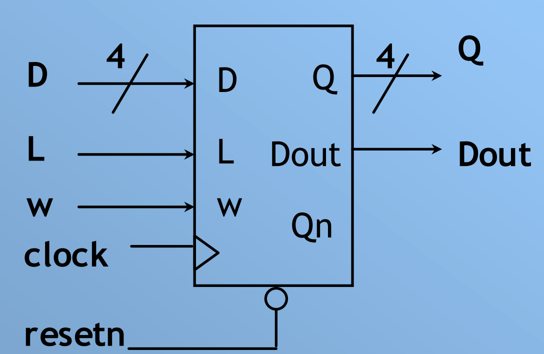
✓ *REGISTER: Serial Input, Serial/Parallel Output*

- **Shifting to the right.**



✓ REGISTER

- **4-bit register:**
  Parallel/serial load
  Parallel/serial output
  Shifting to the right
- The signal 'L' decides whether the load is parallel or serial
- 'Dout': serial output
- 'Q': parallel output




```
library ieee;
use ieee.std_logic_1164.all;

entity reg_t is
 port (clock, resetn, L,w: in std_logic;
  Dout: out std_logic;
  D: in std_logic_vector (3 downto 0);
  Q: out std_logic_vector (3 downto 0));
end reg_t;

architecture bhv of reg_t is
begin
 process (resetn,L,clock)
 begin
  if resetn = '0' then
   Q <= "0000";
  elsif (clock'event and clock = '1') then
   if L = '1' then
     Q <= D;
   else
     Q(0) <= Q(1); Q(1) <= Q(2);
     Q(2) <= Q(3); Q(3) <= w;
   end if;
  end if;
 end process;
Dout <= Q(0);
end bhv;
```

✓ REGISTER

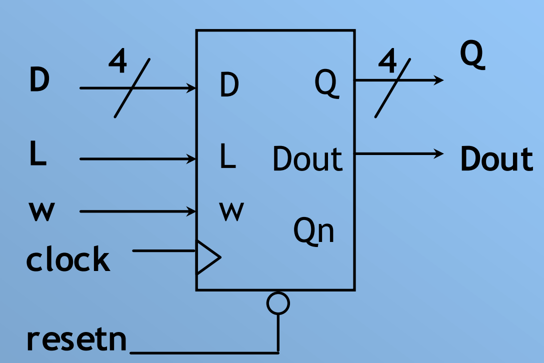
- **Alternative VHDL code:**
- **4-bit register:**
  Parallel/serial load
  Parallel/serial output
  Shifting to the right
- The signal 'L' decides whether the load is parallel or serial
- 'Dout': serial output
- 'Q': parallel output

D
4
L
w
clock
resetn
D
Q
L
Dout
w
Qn
4
Q
Dout

```
library ieee;
use ieee.std_logic_1164.all;

entity reg_t is
 port (clock, resetn, L,w: in std_logic;
  Dout: out std_logic;
  D: in std_logic_vector (3 downto 0);
  Q: out std_logic_vector (3 downto 0));
end reg_t;

architecture bhv of reg_t is
begin
 process (resetn,L,clock)
 begin
  if resetn = '0' then
   Q <= "0000";
  elsif (clock'event and clock = '1') then
   if L = '1' then
     Q <= D;
   else
     gg: for i in 0 to 2 loop
           Q(i) <= Q(i+1);
         end loop;
         Q(3) <= w;
   end if;
  end if;
 end process;
Dout <= Q(0);
end bhv;
```

✓ REGISTER

- **4-bit register: Parallel/serial load Parallel/serial output Shifting to the left**
- The signal 'L' decides whether the load is parallel or serial
- 'Dout': serial output
- 'Q': parallel output




```
library ieee;
use ieee.std_logic_1164.all;

entity reg_i is
 port (clock, resetn, L,w: in std_logic;
  Dout: out std_logic;
  D: in std_logic_vector (3 downto 0);
  Q: out std_logic_vector (3 downto 0));
end reg_i;

architecture bhv of reg_i is
begin
 process (resetn,L,clock)
 begin
  if resetn = '0' then
   Q <= "0000";
  elsif (clock'event and clock = '1') then
   if L = '1' then
     Q <= D;
   else
     Q(3) <= Q(2); Q(2) <= Q(1);
     Q(1) <= Q(0); Q(0) <= w;
   end if;
  end if;
 end process;
Dout <= Q(3);
end bhv;
```

✓ REGISTER

- **Alternative VHDL code:**
- **4-bit register:**
  Parallel/serial load
  Parallel/serial output
  Shifting to the left
- The signal ‘L’ decides whether the load is parallel or serial
- ‘Dout’: serial output
- ‘Q’: parallel output

D 4 D Q 4 Q
L L Dout Dout
w w Qn
clock
resetn

```
library ieee;
use ieee.std_logic_1164.all;

entity reg_i is
 port (clock, resetn, L,w: in std_logic;
   Dout: out std_logic;
   D: in std_logic_vector (3 downto 0);
   Q: out std_logic_vector (3 downto 0));
end reg_i;

architecture bhv of reg_i is
begin
 process (resetn,L,clock)
 begin
  if resetn = '0' then
   Q <= "0000";
  elsif (clock'event and clock = '1') then
   if L = '1' then
     Q <= D;
   else
      gg: for i in 1 to 3 loop
           Q(i) <= Q(i-1);
         end loop;
           Q(0) <= w;
   end if;
  end if;
 end process;
Dout <= Q(3);
end bhv;
```