SEQUENTIAL CIRCUITS

- Asynchronous sequential circuits: Latches
- Synchronous circuits: flip flops, counters, registers.
- Testbench: Generating clock stimulus
COMBINATIONAL CIRCUITS

- In combinational 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.
SEQUENTIAL CIRCUITS

- Combinational circuits can be described with concurrent statements 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 circuits: Latches
- Basic synchronous sequential circuits: flip flops, counters, and registers.

- We will now go over the VHDL description of sequential circuits.
**ASYNCHRONOUS PROCESS:**

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

```
<table>
<thead>
<tr>
<th>S</th>
<th>R</th>
<th>Q_{t+1}</th>
<th>\overline{Q}_{t+1}</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>Q_t</td>
<td>\overline{Q_t}</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
```

- 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’.
- FPGAs usually have trouble implementing these circuits as FPGAs are synchronous circuits.
library ieee;
use ieee.std_logic_1164.all;

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

architecture bhv of latch_sr is
signal qt,qnt: std_logic;
begin
process (s,r)
begin
if s='1' and r='0' then
qt<='1'; qnt<='0';
elif s='0' and r='1' then
qt<='0'; qnt <= '1';
elif s='1' and r='1' then
qt<='0'; qnt <= '0';
end if;
end process;
-- we don't specify what happens
-- if s=r='0' --> q, qn kept their
-- previous values
q <= qt; qn <= qnt;
end bhv;

SR Latch: VHDL code

<table>
<thead>
<tr>
<th>S</th>
<th>Q</th>
<th>S</th>
<th>Q</th>
<th>S</th>
<th>Q</th>
<th>S</th>
<th>Q</th>
</tr>
</thead>
<tbody>
<tr>
<td>R</td>
<td>Q</td>
<td>R</td>
<td>Q</td>
<td>R</td>
<td>Q</td>
<td>R</td>
<td>Q</td>
</tr>
</tbody>
</table>

---[Daniel Llamocca]---
- SR Latch with enable

<table>
<thead>
<tr>
<th>E</th>
<th>S</th>
<th>R</th>
<th>Q_{t+1}</th>
<th>Q_{t+1}</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>x</td>
<td>x</td>
<td>Q_{t}</td>
<td>Q_{t}</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>Q_{t}</td>
<td>Q_{t}</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

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

```vhdl
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;
```
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;

<table>
<thead>
<tr>
<th>E</th>
<th>D</th>
<th>Q_{t+1}</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>x</td>
<td>Q_{t}</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
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 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

```vhdl
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 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;
```

Positive edge-triggered
Negative edge-triggered
SYNCHRONOUS PROCESSES

- D Flip Flop

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;

<table>
<thead>
<tr>
<th>clock</th>
<th>D</th>
<th>Q_{t+1}</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
SYNCHRONOUS PROCESSES

- T Flip Flop

```
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

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;

<table>
<thead>
<tr>
<th>clock</th>
<th>J</th>
<th>K</th>
<th>Q_{t+1}</th>
</tr>
</thead>
<tbody>
<tr>
<td>~</td>
<td>0</td>
<td>0</td>
<td>Q_t</td>
</tr>
<tr>
<td>~</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>~</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>~</td>
<td>1</td>
<td>1</td>
<td>Q_t</td>
</tr>
</tbody>
</table>
**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

```vhdl
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

- 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, pipelining registers, etc.

  - n-bit Register: Storage element that can hold ‘n’ bits. It is a collection of ‘n’ D-type flip flops

- Register types:
  - Simple Register (with/without enable)
  - Shift register (with/without enable)
    - Serial input, parallel output
    - Serial input, serial output
  - Parallel access shift register (parallel/serial input, parallel/serial output)
PARALLEL LOAD, PARALLEL OUTPUT

- 8-bit register with enable and asynchronous reset

```vhdl
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;
```
PARALLEL LOAD, PARALLEL OUTPUT

- n-bit register with enable, sclr and asynchronous reset
- sclr: only works if $E = '1'$

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

entity my_rege is
  generic (N: INTEGER:= 4);
  port ( clock, resetn: in std_logic;
         E, sclr: in std_logic;
         D: in std_logic_vector (N-1 downto 0);
         Q: out std_logic_vector (N-1 downto 0));
  end my_rege;

architecture Behavioral of my_rege is
  signal Qt: std_logic_vector (N-1 downto 0);
  begin
    process (resetn, clock)
      begin
        if resetn = '0' then Qt <= (others =>'0');
        elsif (clock'event and clock = '1') then
          if E = '1' then
            if sclr='1' then Qt <= (others =>'0');
            else Qt <= D;
            end if;
          else Qt <= D;
          end if;
        end if;
      end process;
    Q <= Qt;
  end Behavioral;
end my_rege;

my_rege.zip:
  my_rege.vhd,
  tb_my_rege.vhd
```

Daniel Llamocca
TESTBENCH

- Generating clock stimulus

A **clock** signal is an square wave with a fixed frequency. The Duty Cycle is usually 50%.

The example shows a code snippet of the testbench for my_reg3.vhd: An independent process is needed just to create the clock signal

```vhdl
architecture bhv of tb_my_rege is
...
constant T: time:= 10 ns;
constant DC: real:= 0.5;
begin
  uut: my_rege port map (clock=>clock,E=>E,
resetn=>resetn,sclr=>sclr,D=>D,Q=>Q);
  clock_process: process
  begin
    clock <='0'; wait for (T - T*DC);
    clock <='1'; wait for T*DC;
  end process;
  stim_process: process
  begin
    wait for 100 ns;
    resetn <= '1'; wait for 2*T;
    --
  end process;
end bhv;
```

[Diagram of clock and reset signals with Duty Cycle (DC) percentage]
✓ **REGISTER: Example**

- **3-state buffers and 6-to-6 LUT**
  - LUT6-to-6: built by grouping six LUT6-to-1 in parallel. LUT6-to-1: made out of LUT4-to-1.
  - Note that the port DATA can be input or output at different times. In VHDL, we use the **INOUT** data type to specify this.
  - LUT6-to-6 contents: Any function of 6 input bits and 6 output bits can be pre-computed and stored in the LUT6-to-6. In the example, the function is \( OLUT = [ILUT^{0.95}] \)

---

![Diagram of 3-state buffers and 6-to-6 LUT]

Daniel Llamocca
**REGISTER: Example**

- **3-state buffers and 6-to-6 LUT**
- Data: 64 rows of 6 bits. Or 6 columns of 64 bits. The figure shows the entity VHDL portion of the system.

```vhdl
entity sysLUT6to6 is
    generic(
        data5: std_logic_vector(63 downto 0) := "ffffffc000000000";
        data4: std_logic_vector(63 downto 0) := "fc00003ffffc0000";
        data3: std_logic_vector(63 downto 0) := "03ff003ff003ff00";
        data2: std_logic_vector(63 downto 0) := "83e0f83e0f83e0f0";
        data1: std_logic_vector(63 downto 0) := "639ce739ce739ce7398cc";
        data0: std_logic_vector(63 downto 0) := "5a5296b5ad6a56aa";
    );
    port (clk, resetn, OE: in std_logic;
          data: inout std_logic_vector (5 downto 0));
end sysLUT6to6;
```

- **Testbench:** The code shows that when DATA is output (OE=0), it MUST be assigned the value ‘Z’.

```vhdle
resetn<='0'; DATA<="ZZZZZZ"; wait for 100 ns;
resetn<='1'; wait for T;
OE<='1'; DATA<="110001"; wait for 2*T;
OE<='0'; DATA<="ZZZZZZ"; wait for 2*T;
OE<='1'; DATA<="100111"; wait for 2*T;
OE<='0'; DATA<="ZZZZZZ"; wait for 2*T;
```

Daniel Llamocca


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

- **n-bit right shift register:**

  ![n-bit right shift register diagram]

  - resetn
  - shiftin
  - clk

- **n-bit left shift register:**

  ![n-bit left shift register diagram]

  - resetn
  - shiftin
  - clk

- **my_shiftreg.zip**: Generic n-bit left/right Shift Register
  my_shiftreg.vhd,
  tb_my_shiftreg.vhd
PARALLEL ACCESS SHIFT REGISTER

- 4-bit right parallel access shift register with enable:

- 4-bit left parallel Access shift register with enable:

- my_pashiftreg.zip: Generic n-bit left/right Parallel Access Shift Register
  my_pashiftreg.vhd, tb_my_pashiftreg.vhd

Daniel Llamocca
PARALLEL ACCESS
SHIFT REGISTER

- Parallel/serial load
- Parallel/serial output
- Shift to the right, 4 bits

- $s_l=1$ -> Parallel load
- $s_l=0$ -> Serial load

- ‘din’: serial input
- ‘D’: parallel input
- ‘dout’: serial output
- ‘Q’: parallel output

```
library ieee;
use ieee.std_logic_1164.all;
entity pashreg4_right is
  port (clock, resetn: in std_logic;
        E, s_l, din: in std_logic;
        dout: out std_logic;
        D: in std_logic_vector (3 downto 0);
        Q: out std_logic_vector (3 downto 0));
end pashreg4_right;

architecture bhv of pashreg4_right is
  signal Qt: std_logic_vector (3 downto 0);
begin
  process (resetn, clock, s_l, E)
  begin
    if resetn = '0' then Qt <= "0000";
    elsif (clock'event and clock = '1') then
      if E = '1' then
        if s_l='1' then Qt <= D;
        else
          Qt(0) <= Qt(1); Qt(1) <= Qt(2);
          Qt(2) <= Qt(3); Qt(3) <= din;
        end if;
      end if;
    end if;
  end process;
  Q <= Qt; dout <= Qt(0);
end bhv;
```
**PARALLEL ACCESS SHIFT REGISTER**

- **Parallel/serial load**
  - Parallel/serial output
  - Shift to the **right**, 4 bits
- **Use of VHDL for loop**
  - \( s_l=1 \) -> Parallel load
  - \( s_l=0 \) -> Serial load
- ‘\( \text{din} \)': serial input
- ‘\( D \)': parallel input
- ‘\( \text{dout} \)': serial output
- ‘\( Q \)': parallel output

```vhdl
library ieee;
use ieee.std_logic_1164.all;
entity pashreg4_right is
  port (clock, resetn: in std_logic;
        E, s_l, din: in std_logic;
        dout: out std_logic;
        D: in std_logic_vector (3 downto 0);
        Q: out std_logic_vector (3 downto 0));
end pashreg4_right;

architecture bhv of pashreg4_right is
begin
  signal Qt: std_logic_vector (3 downto 0);
  process (resetn, clock, s_l, E)
  begin
    if resetn = '0' then
      Qt <= "0000";
    elsif (clock'event and clock = '1') then
      if E = '1' then
        if s_l = '1' then
          Qt <= D;
        else
          gg: for i in 0 to 2 loop
            Qt(i) <= Qt(i+1);
          end loop;
          Qt(3) <= din;
        end if;
      end if;
    end if;
  end process;
  Q <= Qt; dout <= Qt(0);
end bhv;
```
PARALLEL ACCESS
SHIFT REGISTER

- Parallel/serial load
- Parallel/serial output
- Shift to the left, 4 bits

- \( s_l = 1 \) -> Parallel load
- \( s_l = 0 \) -> Serial load

- ‘din’: serial input
- ‘D’: parallel input
- ‘dout’: serial output
- ‘Q’: parallel output

```vhdl
library ieee;
use ieee.std_logic_1164.all;
entity pashreg4_left is
  port (clock, resetn: in std_logic;
        E, s_l, din: in std_logic;
        dout: out std_logic;
        D: in std_logic_vector (3 downto 0);
        Q: out std_logic_vector (3 downto 0));
end pashreg4_left;

architecture bhv of pashreg4_left is
  signal Qt: std_logic_vector (3 downto 0);
begin
  process (resetn, clock, s_l, E)
  begin
    if resetn = '0' then Qt <= "0000";
    elsif (clock'event and clock = '1') then
      if E = '1' then
        if s_l='1' then Qt <= D;
        else
          Qt(3) <= Qt(2); Qt(2) <= Qt(1);
          Qt(1) <= Qt(0); Qt(0) <= din;
        end if;
      end if;
    end if;
  end process;
  Q <= Qt; dout <= Qt(3);
end bhv;
```
PARALLEL ACCESS
SHIFT REGISTER

- Parallel/serial load
- Parallel/serial output
- Shift to the left, 4 bits

- Use of VHDL for loop
- \( s_1 = 1 \) -> Parallel load
- \( s_1 = 0 \) -> Serial load
- ‘din’: serial input
- ‘D’: parallel input
- ‘dout’: serial output
- ‘Q’: parallel output

```vhdl
library ieee;
use ieee.std_logic_1164.all;
entity pashreg4_left is
  port (clock, resetn: in std_logic;
    E, s_l, din: in std_logic;
    dout: out std_logic;
    D: in std_logic_vector (3 downto 0);
    Q: out std_logic_vector (3 downto 0));
end pashreg4_left;

architecture bhv of pashreg4_left is
  signal Qt: std_logic_vector (3 downto 0);
begin
  process (resetn, clock, s_l, E)
  begin
    if resetn = '0' then
      Qt <= "0000";
    elsif (clock'event and clock = '1') then
      if E = '1' then
        if s_l='1' then
          Qt <= D;
        else
          gg:
            for i in 1 to 3 loop
              Qt(i) <= Qt(i-1);
            end loop;
            Qt(0) <= din;
        end if;
      end if;
    end if;
  end process;
  Q <= Qt; dout <= Qt(3);
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 binary counter (0000 to 1111). A resetn signal is also included to initialize the count.
4-bit binary counter with asynchronous active-low reset

- Count: 0 to $2^4 - 1$
- resetn: active-low signal that sets Q to 0 as soon as it is 0, with no regard to the clock
- VHDL code: The behavioral style is preferred for counters (instead of the structural description).
- VHDL code: integer is used instead of std_logic_vector. The number of bits (4) is automatically computed.

```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 binary counter with enable and asynchronous active-low reset

- Note that the enable signal ‘E’ is synchronous, thus it is only considered on the rising clock edge.
- When Qt = 1111, then Qt ← Qt+1 will result in Qt = 0000. Thus, this is also a counter modulo-15.

```vhdl
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;
```

![Diagram of a 4-bit binary counter with enable and asynchronous active-low reset](image)

- Note that the enable signal ‘E’ is synchronous, thus it is only considered on the rising clock edge.
- When Qt = 1111, then Qt ← Qt+1 will result in Qt = 0000. Thus, this is also a counter modulo-15.
4-bit binary counter with enable, asynchronous active-low reset and synchronous clear

- The signals ‘E’ and ‘sclr’ are synchronous, thus they are only considered on the rising clock edge.
- If E=sclr=1 then Qt is set to 0.
- When Qt = 15, then Qt ← Qt+1 will result in Qt = 0.

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

entity my_count4b_E_sclr is
  port ( clock, resetn, E, sclr: in std_logic;
        Q: out integer range 0 to 15);
end my_count4b_E_sclr;

architecture bhv of my_count4b_E_sclr 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
        if sclr = '1' then Qt <= 0;
        else
          Qt <= Qt + 1;
        end if;
      end if;
      end if;
    end process;
  Q <= Qt;
end bhv;
```
4-bit BCD counter with asynchronous active-low reset

- Count: 0 to 9

```vhdl
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 modulo-13 counter with enable and asynchronous active-low reset

- Count: 0 to 12
- Output ‘z’: Asserted when count is 12.

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

entity my_mod13count is
  port ( clock, resetn, E: in std_logic;
         Q: out std_logic_vector(3 downto 0);
         z: out std_logic);
end my_mod13count;

architecture bhv of my_mod13count is
  signal Qt: integer range 0 to 12;
begin
  process (resetn,clock,E)
  begin
    if resetn = '0' then Qt <= 0;
    elsif (clock'event and clock='1') then
      if E = '1' then
        if Qt=12 then Qt <= 0;
        else Qt <= Qt + 1;
        end if;
      end if;
    end if;
  end process;
  Q <= conv_std_logic_vector(Qt,4);
  z <= '1' when Qt = 12 else '0';
end bhv;
```

- my_mod13count.zip:
  my_mod13count.vhd,
tb_my_mod13count.vhd.
4-bit synchronous up/down counter with asynchronous active-low reset

- $ud = 0 \rightarrow$ down
- $ud = 1 \rightarrow$ up
- When $Qt = 0000$, then $Qt \leftarrow Qt-1$ will result in $Qt = 1111$

library ieee;
use ieee.std_logic_1164.all;

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

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

mybcd_udcount.zip:
mybcd_udcount.vhd,
tb_mybcd_udcount.vhd
mybcd_udcount.ucf
4-bit Synchronous counter with parallel load

- Here, we use Q as a std_logic_vector.

```vhdl
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;
```

Diagram:

- `resetn`
- `load`
- `data`
- `clock`
- `counter`