DIGITAL SYSTEM DESIGN LABORATORY

REPORT LAB 2

IMPLEMENTATION OF SEQUENTIAL LOGIC CIRCUITS USING VERILOG AND VHDL IN FPGA KIT

NAME: Bùi Gia Bảo

ID: ITITIU22019

I. LAB OBJECTIVES

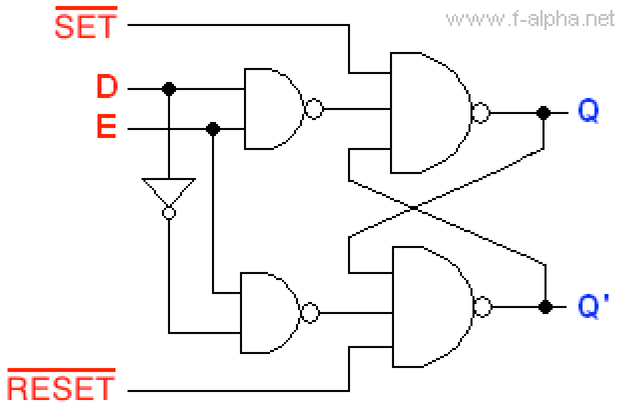
This Lab experiments are intended to implement Basic Sequential Circuits in Verilog. Students are require to write test bench to simulate the given example code and Top level module to implement these codes in DE2-115 FPGA Kit.

II. LAB EXPERIMENT EXERCISES

AIM: WRITE VHDL OR VHDL CODES TO SIMULATE AND IMPLEMENT THE FOLLOWING DIGITAL SEQUENTIAL LOGIC CIRCUITS:

1) DFF with Asynchronous Reset using Verilog
2) DFF with Synchronous Reset using VHDL
3) JK Flip Flop VHDL
4) Asynchronous Counter 4-bit using VHDL structural modeling
5) Asynchronous Counter 8-bit using VHDL behavior modeling
6) Synchronous Counter 4-bit using Verilog structural modeling
7) Synchronous Counter n-bit VHDL behavior modeling
8) Right Shift Register 4-bit using Verilog structural modeling
9) Left Shift Register 4-bit VHDL structural modeling
10) Universal Shift Register N-bit Verilog behavior modeling
11) Universal Shift Register N-bit VHDL behavior modeling

1/ DFF with Asynchronous Reset using Verilog



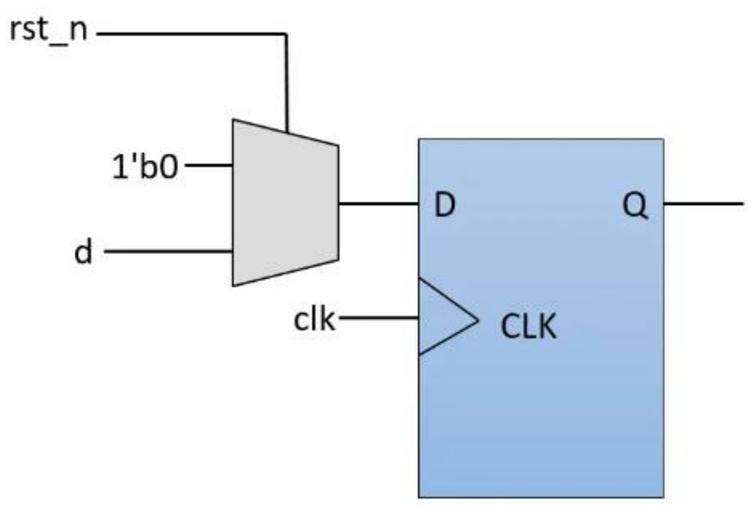
Input			Output	
D	reset	clock	Q	Q'
0	0	0	0	1
0	0	1	0	1
0	1	0	0	1
0	1	1	0	1
1	0	0	0	1
1	0	1	1	0
1	1	0	0	1
1	1	1	0	1

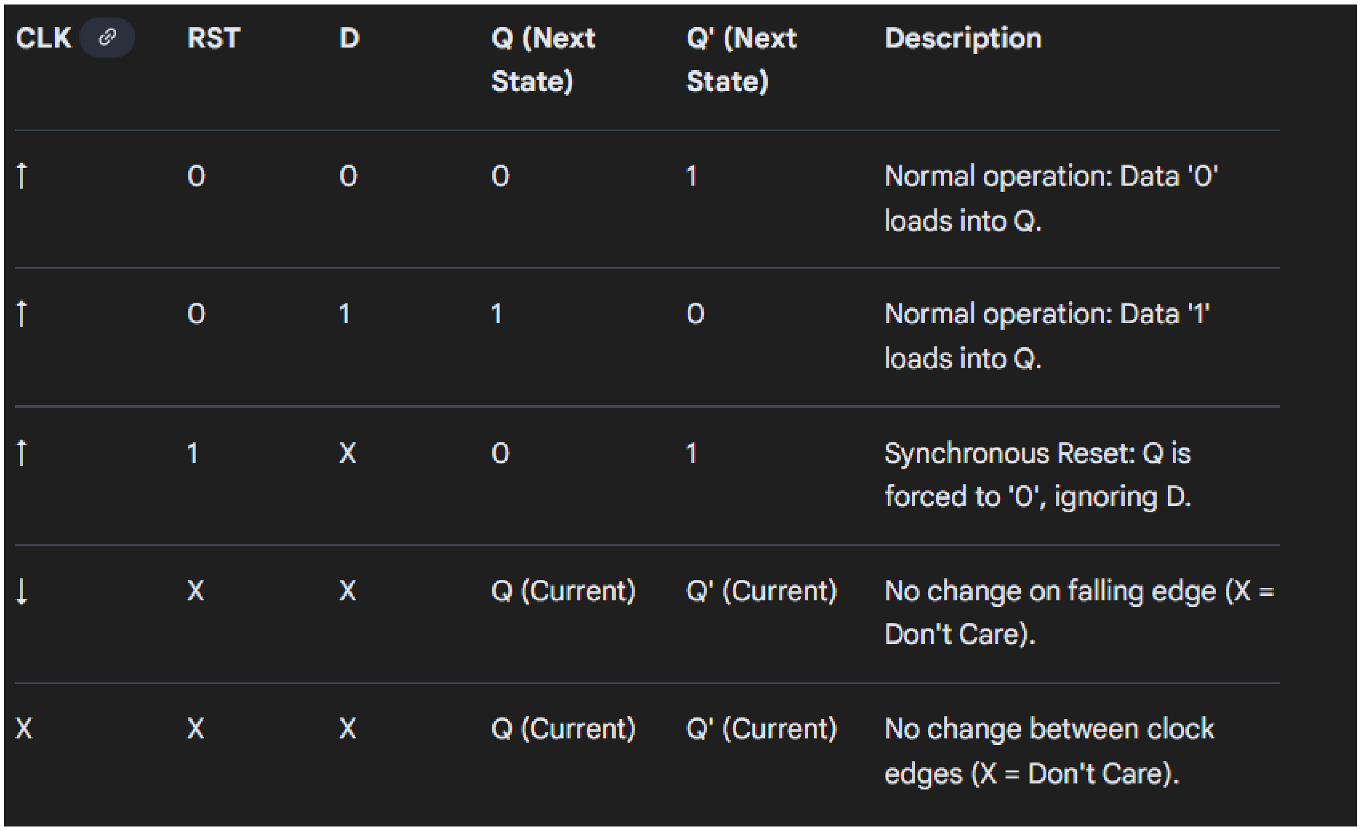
CODE

```
module lab2_ex1(SW,LEDG,LEDR);
        input[2:0] SW;
        output [1:0] LEDG;
        output [2:0] LEDR;
        assign LEDR=SW;

         dff_async_reset test(SW[0],SW[1],SW[2],LEDG[0]);
endmodule

module dff_async_reset (
          input wire data,      // Data input

          input wire reset,    // Asynchronous reset input
          input wire clk,          // Clock input
          output reg q          // Output
        );


          always @ (posedge clk or negedge reset) begin
            // If reset is active (assuming active-low reset)
            if (~reset) begin
              q <= 1'b0;        // Reset the output
            end else begin
              q <= data;        // Capture the data on the clock edge
            end
          end
```

endmodule

2/ DFF with Synchronous Reset using VHDL



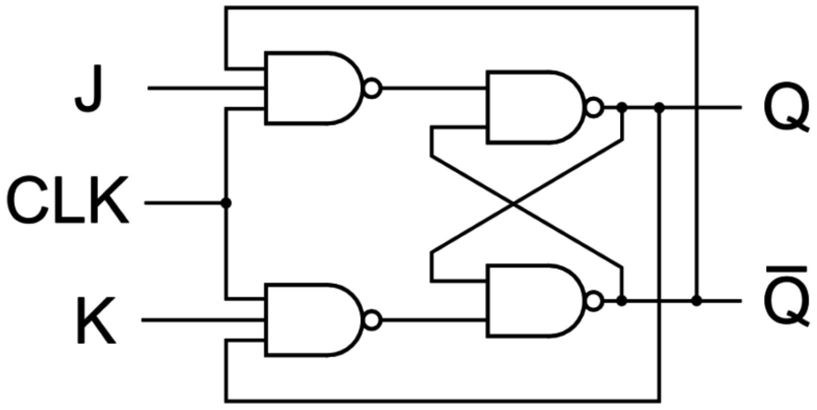


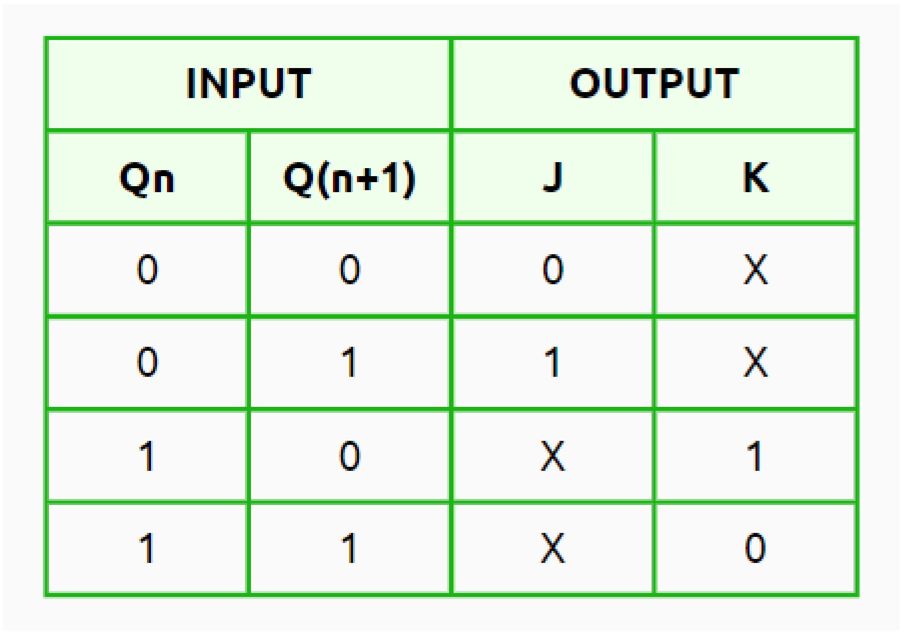
CLK	RST	D	Q (Next State)	Q' (Next State)	Description
↑	0	0	0	1	Normal operation: Data '0' loads into Q.
↑	0	1	1	0	Normal operation: Data '1' loads into Q.
↑	1	X	0	1	Synchronous Reset: Q is forced to '0', ignoring D.
↓	X	X	Q (Current)	Q' (Current)	No change on falling edge (X = Don't Care).
X	X	X	Q (Current)	Q' (Current)	No change between clock edges (X = Don't Care).

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity ex2_vhdl is
    Port (
        SW   : in  STD_LOGIC_VECTOR(2 downto 0);  -- SW(0)=clk, SW(1)=reset, SW(2)=D
        LEDG : out STD_LOGIC_VECTOR(1 downto 0);
        LEDR : out STD_LOGIC_VECTOR(2 downto 0)
    );
end ex2_vhdl;

architecture Structural of ex2_vhdl is

    -- Internal component declaration for DFF
    component dff_sync
        Port (
            clk   : in  STD_LOGIC;
            reset : in  STD_LOGIC;
            d     : in  STD_LOGIC;
            q     : out STD_LOGIC
        );
    end component;

begin
    -- Connect switches to red LEDs
    LEDR <= SW;

    -- Instantiate D Flip-Flop (Synchronous Reset)
    test: dff_sync
        port map (
            clk   => SW(0),
            reset => SW(1),
            d     => SW(2),
            q     => LEDG(0)
        );

    -- Unused green LED output (set to 0)
    LEDG(1) <= '0';

end Structural;


library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity dff_sync is
    Port (
        clk   : in  STD_LOGIC;
        reset : in  STD_LOGIC;
        d     : in  STD_LOGIC;
        q     : out STD_LOGIC
    );
end dff_sync;
```

```
architecture Behavioral of dff_sync is
    signal q_reg : STD_LOGIC := '0';
            begin
                process (clk)
                begin
                    if rising_edge(clk) then
                        if reset = '1' then
                            q_reg <= '0';
                        else
                            q_reg <= d;
                        end if;
                    end if;
                end process;
                q <= q_reg;
end Behavioral;
```

3/ JK Flip Flop VHDL





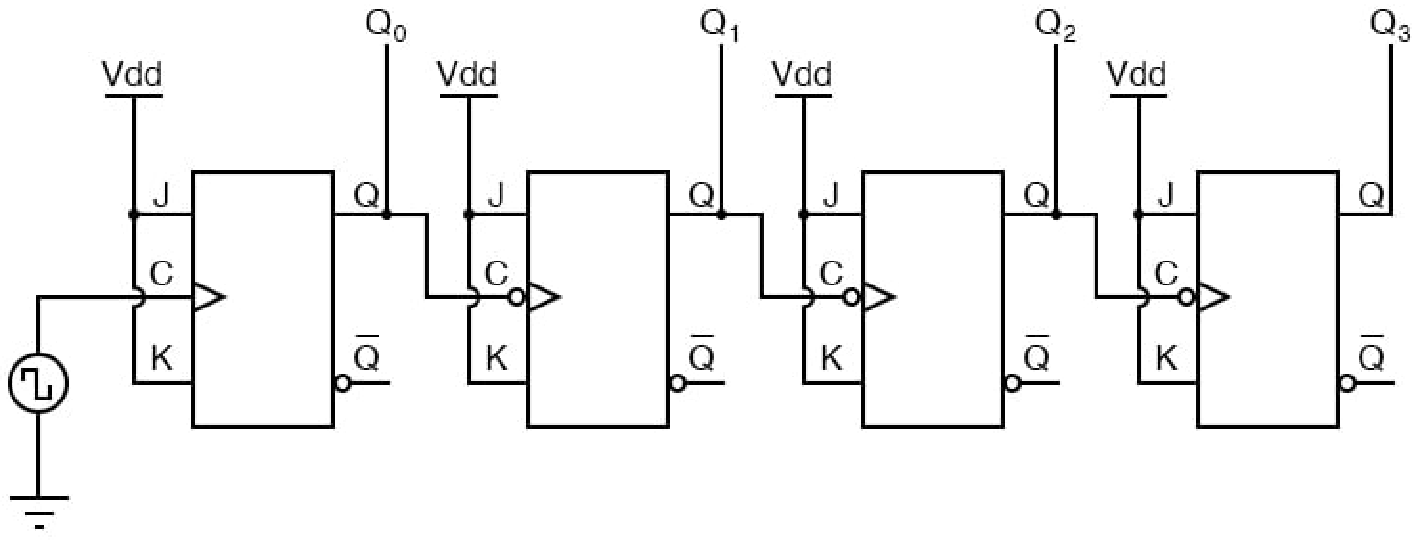
INPUT		OUTPUT	
Qn	Q(n+1)	J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

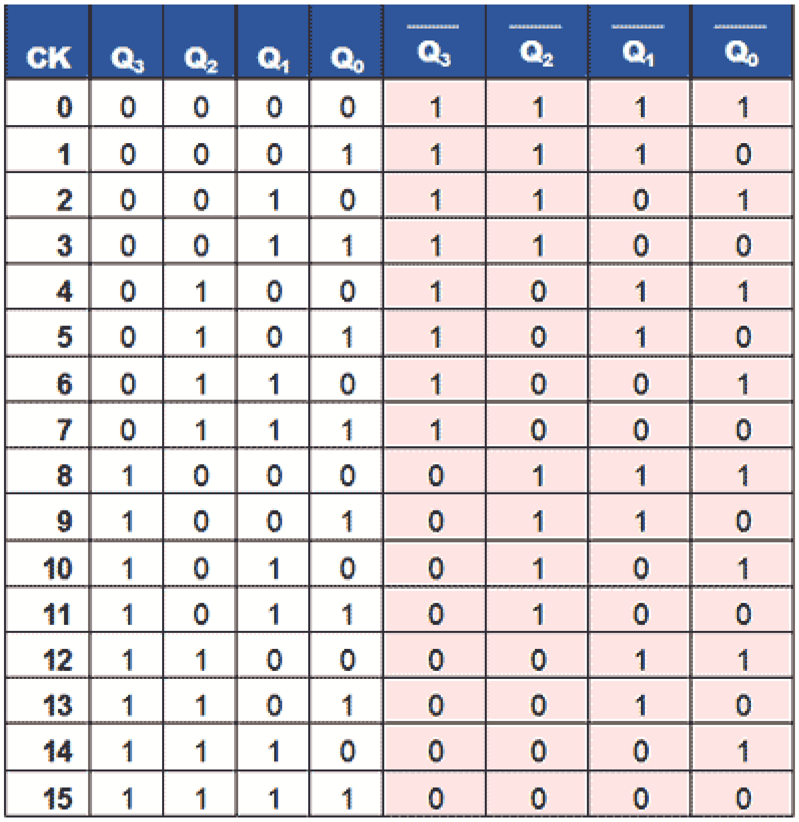
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- ================================================================
-- Top-Level Entity: ex3_vhdl
-- Description: JK Flip-Flop with Asynchronous Reset
-- ================================================================
entity ex3_vhdl is
    Port (
        SW   : in  STD_LOGIC_VECTOR(3 downto 0);  -- SW(0)=CLK, SW(1)=J, SW(2)=K,
SW(3)=RESET
        LEDG : out STD_LOGIC_VECTOR(1 downto 0);  -- LEDG(0)=Q, LEDG(1)=Q'
        LEDR : out STD_LOGIC_VECTOR(3 downto 0)   -- Mirrors SW
    );
end ex3_vhdl;

architecture Structural of ex3_vhdl is
    -- Component declaration for JK Flip-Flop
    component jk_ff
        Port (
            clk   : in  STD_LOGIC;
            j     : in  STD_LOGIC;
            k     : in  STD_LOGIC;
            reset : in  STD_LOGIC;  -- Active-high asynchronous reset
            q     : out STD_LOGIC
        );
    end component;

    signal Q_sig : STD_LOGIC;
```

```
begin
    -- Connect switches to red LEDs
    LEDR <= SW;

    -- Instantiate JK Flip-Flop
    DUT: jk_ff
        port map (
            clk   => SW(0),
            j     => SW(1),
            k     => SW(2),
            reset => SW(3),
            q     => Q_sig
        );

    -- Output Q and its complement
    LEDG(0) <= Q_sig;
    LEDG(1) <= not Q_sig;

end Structural;


-- ============================================================
-- Submodule: JK Flip-Flop (Asynchronous Reset)
-- ============================================================
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity jk_ff is
    Port (
        clk   : in  STD_LOGIC;
        j     : in  STD_LOGIC;
        k     : in  STD_LOGIC;
        reset : in  STD_LOGIC;
        q     : out STD_LOGIC
    );
end jk_ff;

architecture Behavioral of jk_ff is
    signal q_reg : STD_LOGIC := '0';
    signal jk_vec : STD_LOGIC_VECTOR(1 downto 0);
begin
    jk_vec <= j & k;  -- Explicitly make a 2-bit vector

    process (clk, reset)
    begin
        if reset = '1' then
            q_reg <= '0';                -- Asynchronous reset
        elsif rising_edge(clk) then
            case jk_vec is
                when "00" => q_reg <= q_reg; -- No change
                when "01" => q_reg <= '0';   -- Reset
                when "10" => q_reg <= '1';   -- Set
                when "11" => q_reg <= not q_reg; -- Toggle
                when others => null;
            end case;
        end if;
    end process;

    q <= q_reg;
end Behavioral;
```

4/ Asynchronous Counter 4-bit using VHDL structural modeling

A four-bit "up" counter





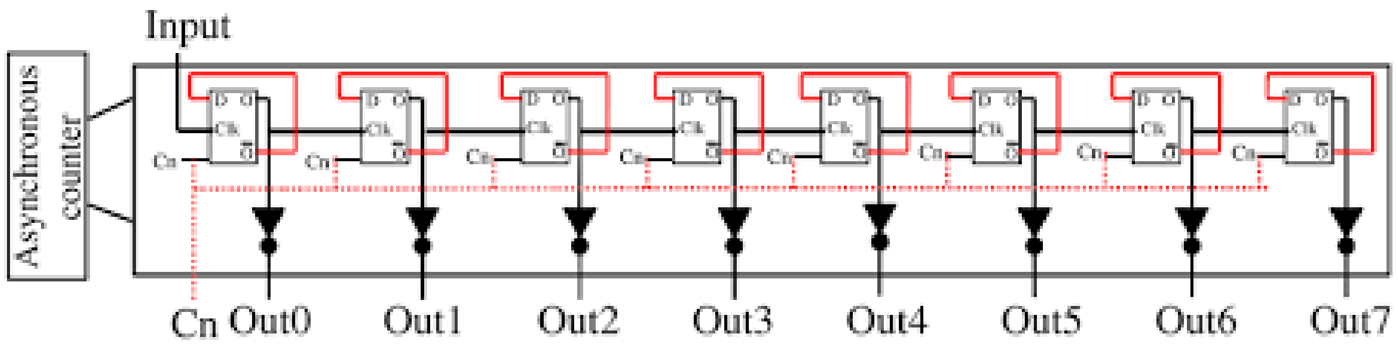
CK	Q_3	Q_2	Q_1	Q_0	$\overline{Q_3}$	$\overline{Q_2}$	$\overline{Q_1}$	$\overline{Q_0}$
0	0	0	0	0	1	1	1	1
1	0	0	0	1	1	1	1	0
2	0	0	1	0	1	1	0	1
3	0	0	1	1	1	1	0	0
4	0	1	0	0	1	0	1	1
5	0	1	0	1	1	0	1	0
6	0	1	1	0	1	0	0	1
7	0	1	1	1	1	0	0	0
8	1	0	0	0	0	1	1	1
9	1	0	0	1	0	1	1	0
10	1	0	1	0	0	1	0	1
11	1	0	1	1	0	1	0	0
12	1	1	0	0	0	0	1	1
13	1	1	0	1	0	0	1	0
14	1	1	1	0	0	0	0	1
15	1	1	1	1	0	0	0	0

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity ex4_vhdl is
   port (
      CLOCK_50 : in  STD_LOGIC;          -- 50 MHz clock input
      SW       : in  STD_LOGIC_VECTOR(0 downto 0);  -- SW(0) = active-low reset
      LEDG     : out STD_LOGIC_VECTOR(3 downto 0);  -- 4-bit counter output
      LEDR     : out STD_LOGIC_VECTOR(0 downto 0)   -- mirror reset switch
   );
end ex4_vhdl;

architecture rtl of ex4_vhdl is

   signal clk_1Hz : STD_LOGIC := '0';
   signal q       : STD_LOGIC_VECTOR(3 downto 0) := (others => '0');

begin

   -- Mirror reset switch on LEDR
   LEDR <= SW;

   -- Instantiate 1 Hz Clock Divider
   Divider: entity work.clock_divider_1Hz
```

```
        port map (
            clk_in  => CLOCK_50,
            reset_n => SW(0),
            clk_out => clk_1Hz
        );

    -- Asynchronous Ripple Counter
    process(clk_1Hz, SW)
    begin
        if SW(0) = '0' then
            q(0) <= '0';
        elsif rising_edge(clk_1Hz) then
            q(0) <= not q(0);
        end if;
    end process;

    process(q(0), SW)
    begin
        if SW(0) = '0' then
            q(1) <= '0';
        elsif falling_edge(q(0)) then
            q(1) <= not q(1);
        end if;
    end process;

    process(q(1), SW)
    begin
        if SW(0) = '0' then
            q(2) <= '0';
        elsif falling_edge(q(1)) then
            q(2) <= not q(2);
        end if;
    end process;

    process(q(2), SW)
    begin
        if SW(0) = '0' then
            q(3) <= '0';
        elsif falling_edge(q(2)) then
            q(3) <= not q(3);
        end if;
    end process;

    LEDG <= q;

end rtl;
```

```
--==================================================================
-- 50 MHz to 1 Hz Clock Divider
--==================================================================
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity clock_divider_1Hz is
   port (
      clk_in  : in  STD_LOGIC;
      reset_n : in  STD_LOGIC;   -- Active-low reset
      clk_out : out STD_LOGIC
   );
end clock_divider_1Hz;

architecture rtl of clock_divider_1Hz is
   signal count   : unsigned(25 downto 0) := (others => '0');
   signal clk_reg : STD_LOGIC := '0';
begin

   process(clk_in, reset_n)
   begin
      if reset_n = '0' then
         count   <= (others => '0');
         clk_reg <= '0';
      elsif rising_edge(clk_in) then
         if count = 24999999 then  -- Toggle every 0.5s for 1Hz period
            count   <= (others => '0');
            clk_reg <= not clk_reg;
         else
            count <= count + 1;
         end if;
      end if;
   end process;

   clk_out <= clk_reg;

end rtl;
```

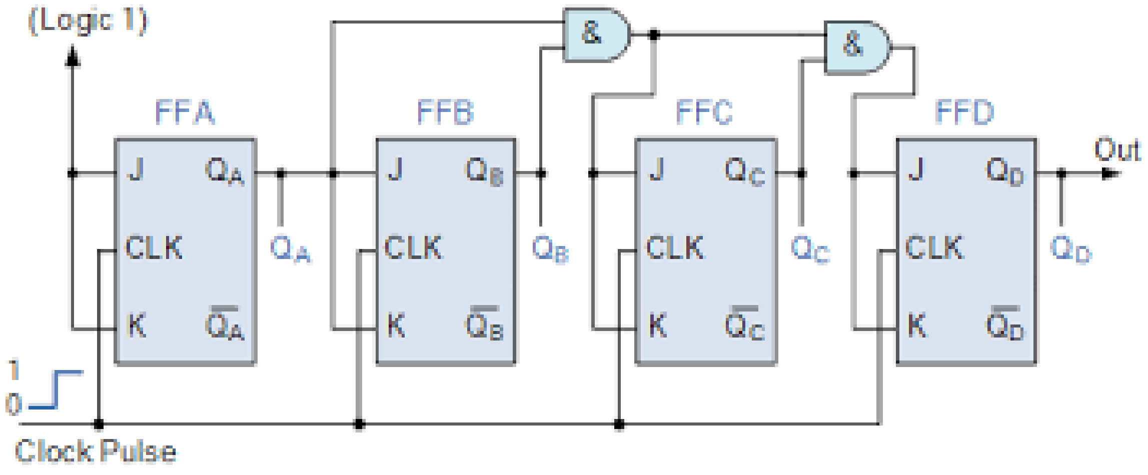
5/ Asynchronous Counter 8-bit using VHDL behavior modeling



CODE

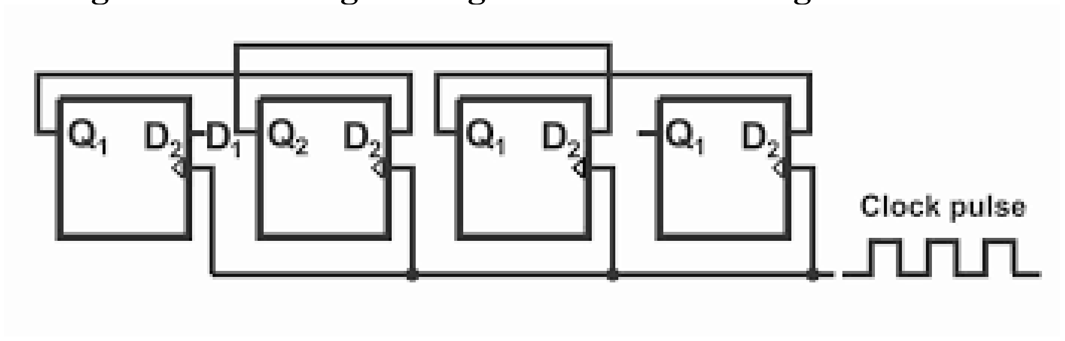
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity ex5_vhdl is
    port (
        CLOCK_50 : in  STD_LOGIC;           -- 50 MHz clock input
        SW       : in  STD_LOGIC_VECTOR(0 downto 0);  -- SW(0) = active-low reset
        LEDG     : out STD_LOGIC_VECTOR(7 downto 0);  -- 4-bit counter output
        LEDR     : out STD_LOGIC_VECTOR(0 downto 0)   -- mirror reset switch
    );
end ex5_vhdl;

architecture rtl of ex5_vhdl is

    signal clk_1Hz : STD_LOGIC := '0';
    signal q       : STD_LOGIC_VECTOR(7 downto 0) := (others => '0');

begin

    -- Mirror reset switch on LEDR
    LEDR <= SW;

    -- Instantiate 1 Hz Clock Divider
    Divider: entity work.clock_divider_1Hz
        port map (
            clk_in  => CLOCK_50,
            reset_n => SW(0),
            clk_out => clk_1Hz
        );

    -- Asynchronous Ripple Counter
    process(clk_1Hz, SW)
    begin
        if SW(0) = '1' then
            q(0) <= '0';
        elsif rising_edge(clk_1Hz) then
            q(0) <= not q(0);
        end if;
    end process;

    process(q(0), SW)
    begin
        if SW(0) = '1' then
            q(1) <= '0';
        elsif falling_edge(q(0)) then
            q(1) <= not q(1);
        end if;
    end process;

    process(q(1), SW)
    begin
        if SW(0) = '1' then
            q(2) <= '0';
        elsif falling_edge(q(1)) then
            q(2) <= not q(2);
        end if;
    end process;

    process(q(2), SW)
    begin
        if SW(0) = '1' then
            q(3) <= '0';
        elsif falling_edge(q(2)) then
            q(3) <= not q(3);
        end if;
    end process;

    process(q(3), SW)
    begin
        if SW(0) = '1' then
            q(4) <= '0';
        elsif falling_edge(q(3)) then
            q(4) <= not q(4);
        end if;
    end process;

    process(q(4), SW)
    begin
        if SW(0) = '1' then
            q(5) <= '0';
        elsif falling_edge(q(4)) then
            q(5) <= not q(5);
        end if;
    end process;

    process(q(5), SW)
    begin
        if SW(0) = '1' then
            q(6) <= '0';
        elsif falling_edge(q(5)) then
            q(6) <= not q(6);
        end if;
    end process;

     process(q(6), SW)
    begin
        if SW(0) = '1' then
            q(7) <= '0';
        elsif falling_edge(q(6)) then
            q(7) <= not q(7);
        end if;
    end process;


    LEDG <= q;

end rtl;


library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity clock_divider_1Hz is
    port (
        clk_in  : in  STD_LOGIC;
        reset_n : in  STD_LOGIC;   -- Active-low reset
        clk_out : out STD_LOGIC
    );
end clock_divider_1Hz;

architecture rtl of clock_divider_1Hz is
    signal count   : unsigned(25 downto 0) := (others => '0');
    signal clk_reg : STD_LOGIC := '0';
begin

    process(clk_in, reset_n)
    begin
        if reset_n = '1' then
            count   <= (others => '0');
            clk_reg <= '0';
        elsif rising_edge(clk_in) then
            if count = 24999999 then  -- Toggle every 0.5s for 1Hz period
                count   <= (others => '0');
                clk_reg <= not clk_reg;
            else
                count <= count + 1;
            end if;
        end if;
    end process;

    clk_out <= clk_reg;

end rtl;
```

6/ Synchronous Counter 4-bit using Verilog structural modeling



7/ Synchronous Counter n-bit VHDL behavior modeling

8/ Right Shift Register 4-bit using Verilog structural modeling

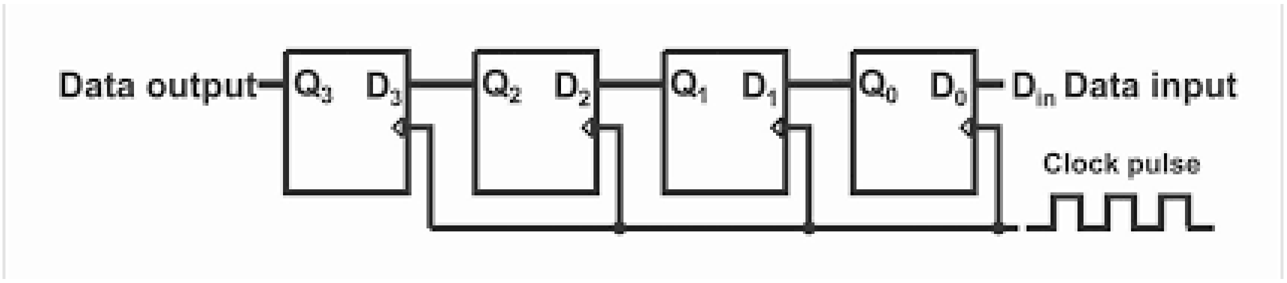


module ex8_verilog(SW, LEDG, LEDR);

```
    input  [3:0] SW;       // SW[0]=CLK, SW[1]=d, SW[2]=input, SW[3]=ASYNC_RESET
    output [3:0] LEDG;     // LEDG[0]=Q, LEDG[1]=Q'
    output [3:0] LEDR;     // Mirrors switch input

    assign LEDR = SW;
    wire q0, q1, q2, q3;
    wire d0, d1, d2, d3;

    assign d3 = SW[2];
    assign d2 = q3;
    assign d1 = q2;
    assign d0 = q1;


    // Instantiate JK Flip-Flop
    jkff_structural JK3 (
        .clk(SW[0]),
        .d(d3),
        .reset(SW[3]),
        .q(q3),
        .qbar(Qbar)
    );
    jkff_structural JK2 (
        .clk(SW[0]),
        .d(d2),
        .reset(SW[3]),
        .q(q2),
        .qbar(Qbar)
    );
    jkff_structural JK1 (
        .clk(SW[0]),
        .d(d1),
        .reset(SW[3]),
        .q(q1),
        .qbar(Qbar)
    );
    jkff_structural JK0 (
        .clk(SW[0]),
        .d(d0),
        .reset(SW[3]),
        .q(q0),
        .qbar(Qbar)
    );


assign LEDG = {q3,q2,q1,q0};

endmodule
```

```
module jkff_structural(
            input clk, d, reset,
            output q, qbar
         );
            wire jn, kn, dnot;
            wire s1, r1, s2, r2;
            wire qm, qmb;  // master latch output

            // Invert reset
            wire nreset;
            not (nreset, reset);
                  not(dnot,d);

            // Input gating for master latch
            nand (s1, d, qmb, clk);
            nand (r1, dnot, qm, clk);

            // Master latch
            nand (qm, s1, qmb, nreset);
            nand (qmb, r1, qm, nreset);

            // Slave latch (triggered on inverted clock)
            nand (s2, qm, ~clk);
            nand (r2, qmb, ~clk);
            nand (q, s2, qbar, nreset);
            nand (qbar, r2, q, nreset);
endmodule
```

9/ Left Shift Register 4-bit VHDL structural modeling



CODE

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

--==========================================================
-- D-type JK-style flip-flop (structural form, no qbar)
--==========================================================
entity jkff_structural is
   port (
      clk   : in  std_logic;
      d     : in  std_logic;
      reset : in  std_logic;
      q     : out std_logic
   );
end jkff_structural;

architecture structural of jkff_structural is
   signal nreset, dnot : std_logic;
   signal s1, r1, s2, r2 : std_logic;
   signal qm, qmb : std_logic;
begin
   nreset <= not reset;
   dnot   <= not d;

   -- Input gating for master latch
   s1 <= not (d and qmb and clk);
   r1 <= not (dnot and qm and clk);

   -- Master latch
   qm  <= not (s1 and qmb and nreset);
   qmb <= not (r1 and qm and nreset);

   -- Slave latch (triggered on inverted clock)
   s2 <= not (qm and (not clk));
   r2 <= not (qmb and (not clk));
   q  <= not (s2 and r2 and nreset);
end structural;


--==================================================
-- Top-level module (equivalent to ex9_verilog)
--==================================================
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity ex9_vhdl is
   port (
      SW   : in  std_logic_vector(3 downto 0);  -- SW[0]=CLK, SW[1]=d, SW[2]=input,
SW[3]=ASYNC_RESET
      LEDG : out std_logic_vector(3 downto 0);  -- Q outputs
      LEDR : out std_logic_vector(3 downto 0)   -- Mirrors switch input
   );
end ex9_vhdl;

architecture behavioral of ex9_vhdl is
```

```
    signal q0, q1, q2, q3 : std_logic;
    signal d0, d1, d2, d3 : std_logic;
begin
    LEDR <= SW;

    d3 <= SW(2);
    d2 <= q3;
    d1 <= q2;
    d0 <= q1;

    -- Instantiate Flip-Flops
    JK3: entity work.jkff_structural
        port map(clk => SW(0), d => d3, reset => SW(3), q => q3);

    JK2: entity work.jkff_structural
        port map(clk => SW(0), d => d2, reset => SW(3), q => q2);

    JK1: entity work.jkff_structural
        port map(clk => SW(0), d => d1, reset => SW(3), q => q1);

    JK0: entity work.jkff_structural
        port map(clk => SW(0), d => d0, reset => SW(3), q => q0);

    -- Output assignment
    LEDG <= (q3 & q2 & q1 & q0);
end behavioral;
```

10/ Universal Shift Register N-bit Verilog behavior modeling

11/ Universal Shift Register N-bit VHDL behavior modeling