VLSI Lab Report Assignment 7

BCSE 4th Year 2nd Semester

Name: ***Priti Shaw***
Roll No. : ***001710501076***
Batch: ***A3***

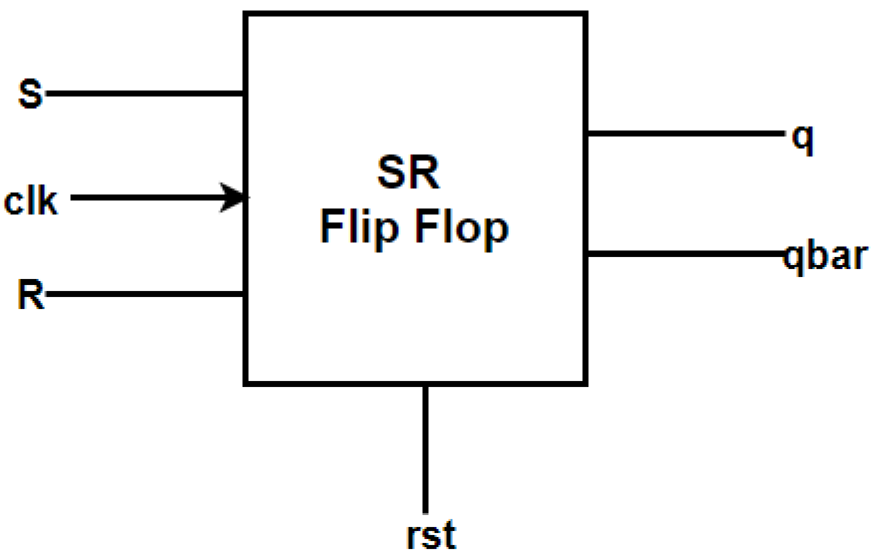
1. SR Flip flop

Description

Implement a SR Flip Flop. Also, write the testbench for it.

SR Flip-Flop is a one-bit memory bistable device that has two inputs, one which will “SET” the device (meaning the output = “1”), and is labelled S and one which will “RESET” the device (meaning the output = “0”), labelled R. Also one Reset input is there to make output (Q) “0”. Otherwise the output can change only in the positive edges of the clock (positive edge triggered) (for negative edge triggered flip flop it will be opposite).

Block Diagram



Entity

```
entity SRFlipflop is
    Port ( rst : in  STD_LOGIC;
           clk : in  STD_LOGIC;
           s : in  STD_LOGIC;
           r : in  STD_LOGIC;
           q : inout  STD_LOGIC;
           qbar : inout  STD_LOGIC);
end SRFlipflop;
```

Truth Table

S	R	Q_n	Q_{n+1}
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	Intermediate
1	1	1	Intermediate

Characteristics Table

S	R	q_{n+1}
0	0	q_n
0	1	0
1	0	1
1	1	Intermediate

Architecture

```
architecture Behavioral of SRFlipflop is

shared variable q1,q1bar:std_logic;

begin
p1:process(clk,rst)

begin
```

```
            if rst='1' then
                  q<='0';
                  qbar<='1';
                  q1:='0';
                  q1bar:='1';
            elsif (clk'event and clk='1') then
                  if (s='0' and r='0') then
                        q<=q1;
                        qbar<=q1bar;
                  elsif (s='0' and r='1') then
                        q<='0';
                        qbar<='1';
                        q1:='0';
                        q1bar:='1';
                  elsif (s='1' and r='0') then
                        q<='1';
                        qbar<='0';
                        q1:='1';
                        q1bar:='0';
                  elsif (s='1' and r='1') then
                        q<='Z';
                        qbar<='Z';
                        q1:='Z';
                        q1bar:='Z';
                  end if;
            end if;
end process;
end Behavioral;
```

TestBench

```
stim_proc: process
   begin
                  rst<='0';
                  wait for 1 ps;

                  rst<='1';
                  wait for 1 ps;

                  rst<='0';

                  loop1:loop
```

```
                               s<='0';
                               r<='0';
                               wait for 1 ps;
                               wait for 1 ps;
                               s<='0';
                               r<='1';
                               wait for 1 ps;
                               wait for 1 ps;
                               s<='1';
                               r<='0';
                               wait for 1 ps;
                               wait for 1 ps;
                               s<='1';
                               r<='1';
                               wait for 1 ps;
                               wait for 1 ps;
                   end loop;
    end process;
```

Timing Diagram

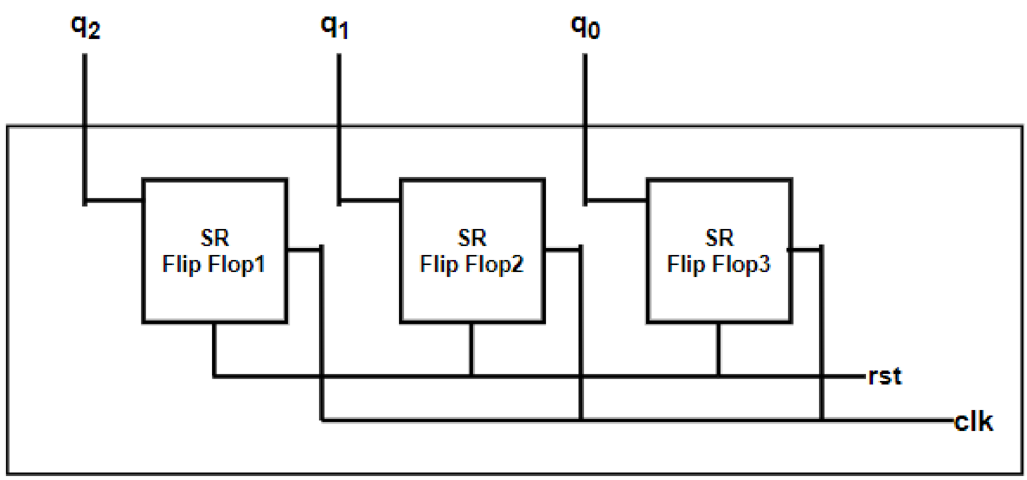
3-bit Up Counter using SR Flip-Flop

Description

Implement a 3-bit Up counter using SR Flip Flop using component instantiation and behavioral modelling . Also, write the testbench for it.

3 SR Flip-Flops are connected to a common clock and some boolean expressions using the outputs (Q(i)) of the flip-flops are provided as inputs to the set and reset bits of the flip-flops. Their reset bit is also the same. The 3 SR flip flop outputs will produce the binary numbers 0 to 7 continuously in order at regular time intervals.

Circuit Diagram



Truth Table

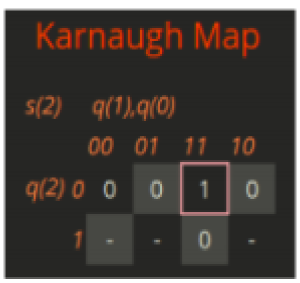
Excitation Table:

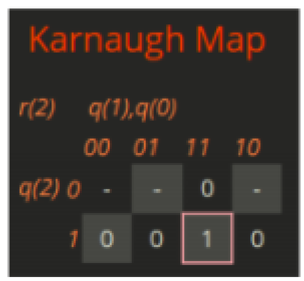
q_n	q_{n+1}	S	R
0	0	0	X
0	1	1	0
1	0	0	1
1	1	X	0

State Table:

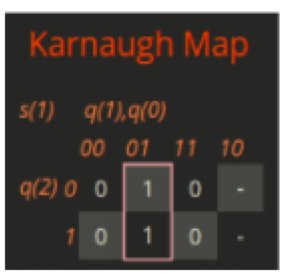
q_2	q_1	q_0	S_2	R_2	S_1	R_1	S_0	R_0
0	0	0	0	X	0	X	1	0
0	0	1	0	X	1	0	0	1
0	1	0	0	X	X	0	1	0
0	1	1	1	0	0	1	0	1
1	0	0	X	0	0	X	1	0
1	0	1	X	0	1	0	0	1
1	1	0	X	0	X	0	1	0
1	1	1	0	1	0	1	0	1

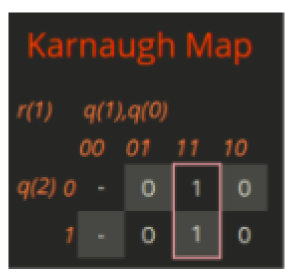
Calculation



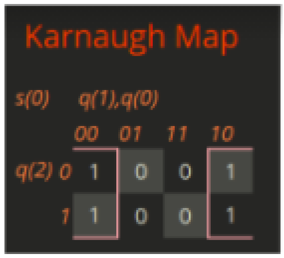


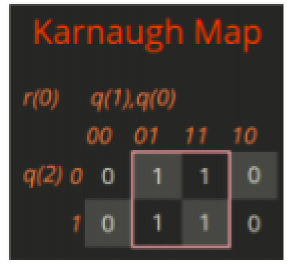
$$s(2) = \overline{q(2)} \cdot q(1) \cdot q(0)$$
$$r(2) = q(2) \cdot q(1) \cdot q(0)$$





$$s(1) = \overline{q(1)} \cdot q(0)$$
$$r(1) = q(1) \cdot q(0)$$





$$s(0) = \overline{q(0)}$$
$$r(0) = q(0)$$

Entity

a) Using component Instantiation

```
entity SRFlipflopCounter is
    Port ( rst : in  STD_LOGIC;
           clk : in  STD_LOGIC;
           qq : inout  STD_LOGIC_VECTOR (2 downto 0);
           qqbar : inout  STD_LOGIC_VECTOR (2 downto 0));
end SRFlipflopCounter;
```

b) Using behavioral modelling

```
entity SRFlipFlopCounterUsingProcedure is
    Port ( rst : in  STD_LOGIC;
           clk : in  STD_LOGIC;
           qqq : inout  STD_LOGIC_VECTOR (2 downto 0);
           qqqbar : inout  STD_LOGIC_VECTOR (2 downto 0));
end SRFlipFlopCounterUsingProcedure;
```

Architecture

a) Using component Instantiation

```
architecture Behavioral of SRFlipflopCounter is

component SRFlipflop is
    Port ( rst : in  STD_LOGIC;
           clk : in  STD_LOGIC;
           s : in  STD_LOGIC;
           r : in  STD_LOGIC;
           q : inout  STD_LOGIC;
           qbar : inout  STD_LOGIC);
end component;
signal ss,rr: std_logic_vector(2 downto 0);
```

```
begin
            ss(2)<=qqbar(2) and qq(1) and qq(0);
            rr(2)<=qq(2) and qq(1) and qq(0);

            ss(1)<=qqbar(1) and qq(0);
            rr(1)<=qq(1) and qq(0);

            ss(0)<=qqbar(0);
            rr(0)<=qq(0);

            gen1: for k in 0 to 2 generate
                              comm:SRFlipflop port
map(rst,clk,ss(k),rr(k),qq(k),qqbar(k));
            end generate;
end Behavioral;
```

b) Using behavioral modelling

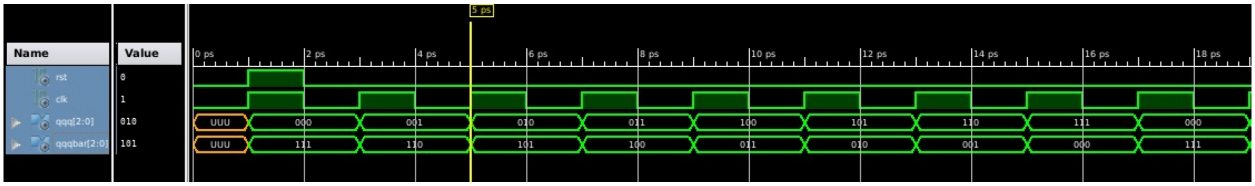
```
architecture Behavioral of SRFlipFlopCounterUsingProcedure is
shared variable qqqqq1, qqqqq1bar:std_logic_vector(2 downto 0);
begin
p1:process(clk, rst)
variable q6,q6bar:std_logic_vector(2 downto 0);
begin
            if rst='1' then
            qqq<="000";
            qqqbar<="111";
            qqqqq1:="000";
            qqqqq1bar:="111";
      elsif (clk'event and clk='1') then

            procc:SRFlipFlopUpCounterProcedure(rst,clk,q6(2 downto
0),q6bar(2 downto 0),qqqqq1(2 downto 0),qqqqq1bar(2 downto 0));

            qqq(2 downto 0)<=q6(2 downto 0);
            qqqbar(2 downto 0)<=q6bar(2 downto 0);
            qqqqq1(2 downto 0) := q6(2 downto 0);
            qqqqq1bar(2 downto 0):=q6bar(2 downto 0);
      end if;
end process;
end Behavioral;
```

SR Flipflop Counter in Package

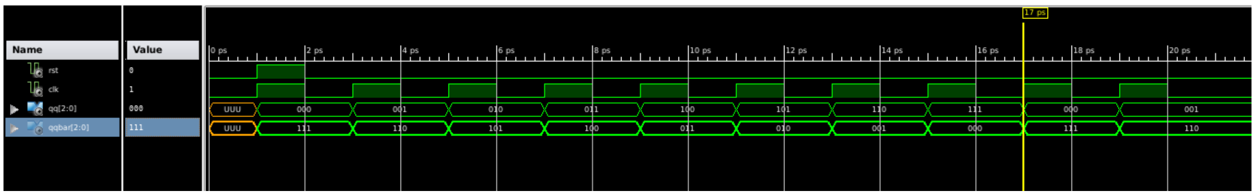
```
procedure SRFlipFlopUpCounterProcedure(rst: in std_logic; clk: in
std_logic; qq: inout std_logic_vector; qqbar: inout std_logic_vector; qqq1:
inout std_logic_vector; qqq1bar: inout std_logic_vector) is
variable ss,rr,qq1,qq1bar:std_logic_vector(2 downto 0);
variable i:integer;
begin
      if rst='1' then
            qq(2 downto 0):="000";
            qqbar(2 downto 0):="111";
            qq1:="000";
            qq1bar:="111";
            qqq1(2 downto 0):="000";
            qqq1bar(2 downto 0):="111";
      elsif (clk='1') then
            qq1(2 downto 0):=qqq1(2 downto 0);
            qq1bar(2 downto 0):=qqq1bar(2 downto 0);

            ss(2):= qq1bar(2) and qq1(1) and qq1(0);
            rr(2):= qq1(2) and qq1(1) and qq1(0);

            ss(1):= qq1bar(1) and qq1(0);
            rr(1):= qq1(1) and qq1(0);

            ss(0):=qq1bar(0);
            rr(0):=qq1(0);

            for i in 0 to 2 loop

prock:SRFlipFlopProcedure(rst,clk,ss(i),rr(i),qq1(i),qq1bar(i),qqq1(i),qqq1
bar(i));
            end loop;
            qq(2 downto 0) := qq1(2 downto 0);
            qqbar(2 downto 0) := qq1bar(2 downto 0);
            qqq1(2 downto 0):=qq1(2 downto 0);
            qqq1bar(2 downto 0):=qq1bar(2 downto 0);
      end if;
end procedure;
```

Timing Diagram

a) Using Component Instantiation



b) Using behavioural modelling



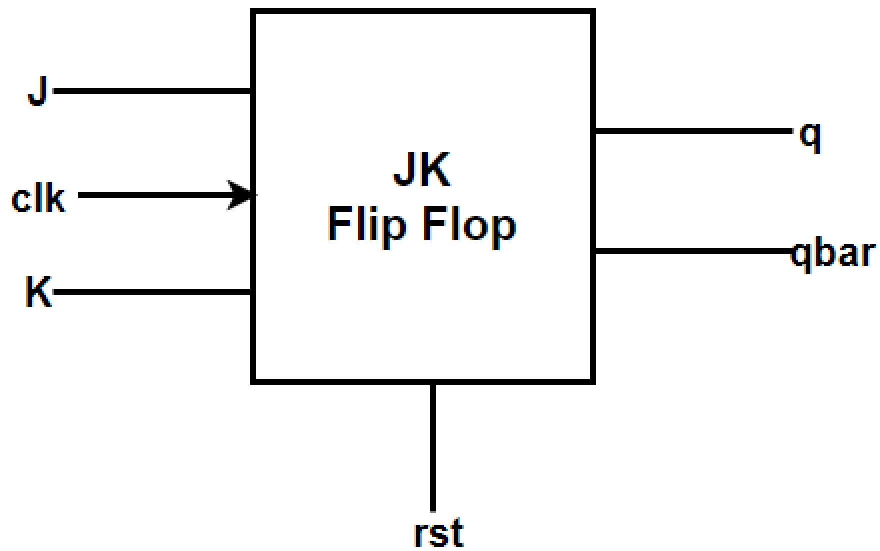
2. JK Flip flop

Description

Implement a JK Flip Flop. Also, write the testbench for it.

JK Flip-Flop acts similar to the SR Flip-Flop. The only difference is that when J and K both inputs are 1, then the output will be toggled.

Block Diagram



Entity

```
entity JKFlipflop is
    Port ( rst : in  STD_LOGIC;
           clk : in  STD_LOGIC;
           j : in  STD_LOGIC;
           k : in  STD_LOGIC;
           q : inout  STD_LOGIC;
           qbar : inout  STD_LOGIC);
end JKFlipflop;
```

Truth Table

J	K	Q_n	Q_{n+1}
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

Characteristics Table

J	K	q_{n+1}
0	0	q_n
0	1	0
1	0	1
1	1	q_n bar

Architecture

```
architecture Behavioral of JKFlipflop is
shared variable qjk,qjkbar:std_logic;
begin
p1:process(clk,rst)

begin
            if rst='1' then
                        q<='0';
                        qbar<='1';
                        qjk:='0';
                        qjkbar:='1';
            elsif (clk'event) then
                  if (j='0' and k='0') then
                              q<=qjk;
                              qbar<=qjkbar;
                  elsif (j='0' and k='1') then
                              q<='0';
                              qbar<='1';
                              qjk:='0';
                              qjkbar:='1';
                  elsif (j='1' and k='0') then
                              q<='1';
                              qbar<='0';
                              qjk:='1';
                              qjkbar:='0';
                  elsif (j='1' and k='1') then
                              q<=qjkbar;
                              qbar<=qjk;
                              qjk:=q;
                              qjkbar:=qbar;
                  end if;
            end if;
end process;

end Behavioral;
```

TestBench

```
stim_proc: process
   begin
                  rst<='0';
                  wait for 1 ps;

                  rst<='1';
                  wait for 1 ps;

                  rst<='0';

                  loop1:loop
                              s<='0';
                              r<='0';
                              wait for 1 ps;
                              wait for 1 ps;
                              s<='0';
                              r<='1';
                              wait for 1 ps;
                              wait for 1 ps;
                              s<='1';
                              r<='0';
                              wait for 1 ps;
                              wait for 1 ps;
                              s<='1';
                              r<='1';
                              wait for 1 ps;
                              wait for 1 ps;
                  end loop;
   end process;
END;
```

Timing Diagram

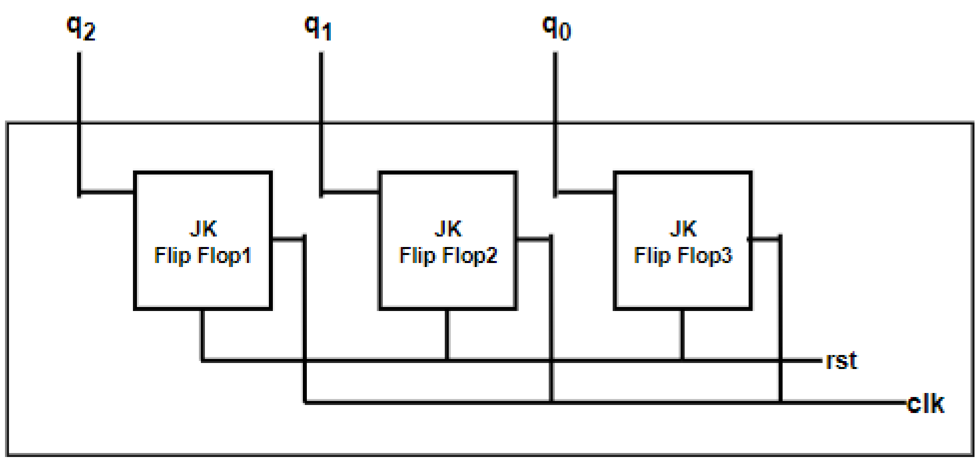
3-bit Up Counter using JK Flip-Flop

Description

Implement a 3-bit Up counter using JK Flip Flop using component instantiation and behavioral modelling . Also, write the testbench for it.

JK Flip-Flop acts similar to the SR Flip-Flop. The only difference is that when J and K both inputs are 1, then the output will be toggled. The mod 8 Up Counter or 3 bit Up Counter setup is similar to that using SR Flip-Flop. The boolean expressions to the J and K inputs of the 3 JK Flip-Flops will be different.

Circuit Diagram



Truth Table

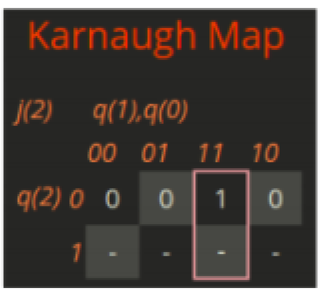
Excitation Table:

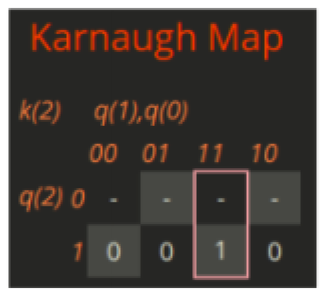
q_n	q_{n+1}	J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

State Table:

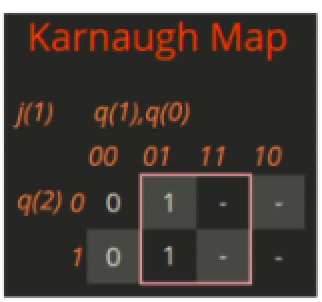
q_2	q_1	q_0	J_2	K_2	J_1	K_1	J_0	K_0
0	0	0	0	X	0	X	1	X
0	0	1	0	X	1	X	X	1
0	1	0	0	X	X	0	1	X
0	1	1	1	X	X	1	X	1
1	0	0	X	0	0	X	1	X
1	0	1	X	0	1	X	X	1
1	1	0	X	0	X	0	1	X
1	1	1	X	1	X	1	X	1

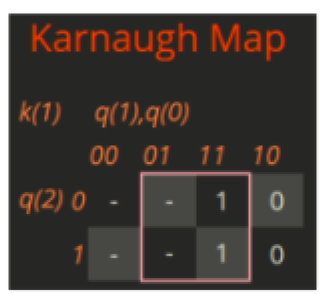
Calculation





$$j(2) = q(1) \cdot q(0)$$
$$k(2) = q(1) \cdot q(0)$$





$$j(1) = q(0)$$
$$k(1) = q(0)$$

$$j(0) = 1$$
$$k(0) = 1$$

Entity

a) Using component Instantiation

```
entity JKFlipflopCounterUsingComponent is
    Port ( rst : in  STD_LOGIC;
           clk : in  STD_LOGIC;
           qq : inout  STD_LOGIC_VECTOR (2 downto 0);
           qqbar : inout  STD_LOGIC_VECTOR (2 downto 0));
end JKFlipflopCounterUsingComponent;
```

b) Using behavioral modelling

```
entity JKFlipflopCounterUsingProcedure is
    Port ( rst : in  STD_LOGIC;
           clk : in  STD_LOGIC;
           qqq : inout  STD_LOGIC_VECTOR (2 downto 0);
           qqqbar : inout  STD_LOGIC_VECTOR (2 downto 0));
end JKFlipflopCounterUsingProcedure;
```

Architecture

a) Using component Instantiation

```
architecture Behavioral of JKFlipflopCounterUsingComponent is
component JKFlipflop is
    Port ( rst : in  STD_LOGIC;
           clk : in  STD_LOGIC;
           j : in  STD_LOGIC;
           k : in  STD_LOGIC;
           q : inout  STD_LOGIC;
           qbar : inout  STD_LOGIC);
end component;
signal jj,kk:std_logic_vector(2 downto 0);
begin
      jj(2)<=qq(1) and qq(0);
```

```
      kk(2)<=qq(1) and qq(0);

      jj(1)<=qq(0);
      kk(1)<=qq(0);

      jj(0)<='1';
      kk(0)<='1';

      gen1: for k in 0 to 2 generate
            comm:JKFlipflop port map(rst,clk,jj(k),kk(k),qq(k),qqbar(k));
      end generate;
end Behavioral;
```

b) Using behavioral modelling

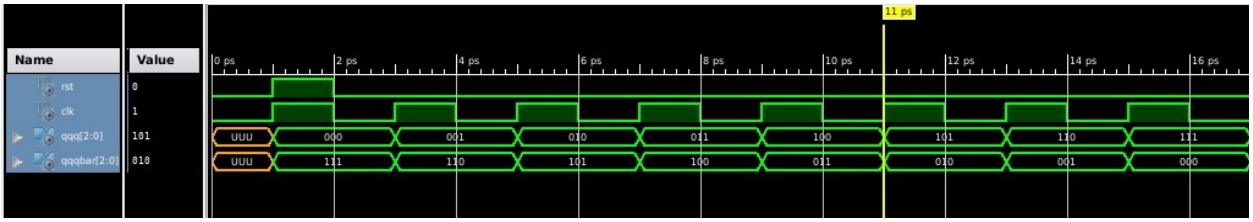
```
architecture Behavioral of JKFlipflopCounterUsingProcedure is

shared variable qqq1, qqq1bar:std_logic_vector(2 downto 0);

begin
p1:process(clk,rst)
variable q4,q4bar:std_logic_vector(2 downto 0);
begin
      if rst='1' then
            qqq<="000";
            qqqbar<="111";
            qqq1:="000";
            qqq1bar:="111";
      elsif (clk'event and clk='1') then

            procc:JKFlipFlopUpCounterProcedure(rst,clk,q4(2 downto
0),q4bar(2 downto 0),qqq1(2 downto 0),qqq1bar(2 downto 0));

            qqq(2 downto 0)<=q4(2 downto 0);
            qqqbar(2 downto 0)<=q4bar(2 downto 0);
            qqq1(2 downto 0) := q4(2 downto 0);
            qqq1bar(2 downto 0):=q4bar(2 downto 0);
      end if;
end process;
end Behavioral;
```

JK Flipflop Counter in Package

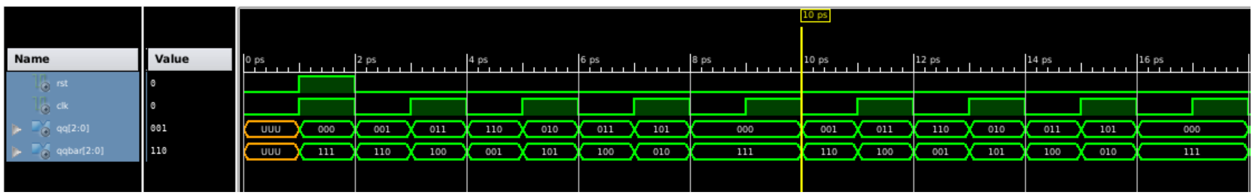
```
procedure JKFlipFlopUpCounterProcedure(rst:in std_logic;clk:in
std_logic;qq:inout std_logic_vector;qqbar:inout std_logic_vector;
qqq1:inout std_logic_vector;qqq1bar:inout std_logic_vector) is
variable jj,kk,qq1,qq1bar:std_logic_vector(2 downto 0);
variable i:integer;
begin
            if rst='1' then
                  qq(2 downto 0):="000";
                  qqbar(2 downto 0):="111";
                  qq1:="000";
                  qq1bar:="111";
                  qqq1(2 downto 0):="000";
                  qqq1bar(2 downto 0):="111";
            elsif (clk='1') then
                  qq1(2 downto 0):=qqq1(2 downto 0);
                  qq1bar(2 downto 0):=qqq1bar(2 downto 0);
                  jj(2):= qq1(1) and qq1(0);
                  kk(2):= qq1(1) and qq1(0);

                  jj(1):= qq1(0);
                  kk(1):= qq1(0);

                  jj(0):='1';
                  kk(0):='1';

                  for i in 0 to 2 loop

prock:JKFlipFlopProcedure(rst,clk,jj(i),kk(i),qq1(i),qq1bar(i),qqq1(i),qqq1
bar(i));
                  end loop;
                  qq(2 downto 0) := qq1(2 downto 0);
                  qqbar(2 downto 0) := qq1bar(2 downto 0);
                  qqq1(2 downto 0):=qq1(2 downto 0);
                  qqq1bar(2 downto 0):=qq1bar(2 downto 0);
            end if;
end procedure;
```

Timing Diagram

a) Using Component Instantiation



b) Using behavioural modelling



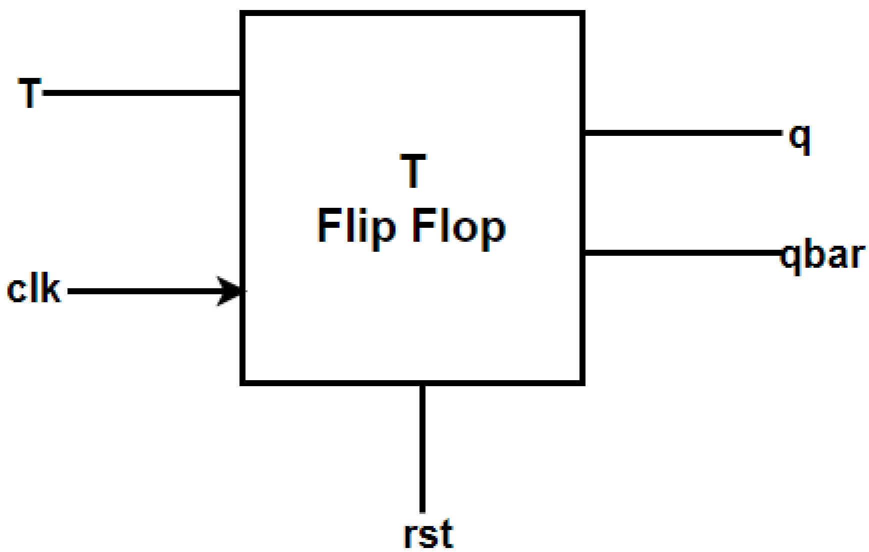
3. T Flip flop

Description

Implement a T Flip Flop. Also, write the testbench for it.

T Flip-Flop takes one input T. If T is 0 then the output remains the same and if T is 1 then the output is toggled.

Block Diagram



Entity

```
entity TFlipflop is
    Port ( rst : in  STD_LOGIC;
           clk : in  STD_LOGIC;
           t : in  STD_LOGIC;
           q : inout  STD_LOGIC;
           qbar : inout  STD_LOGIC);
end TFlipflop;
```

Truth Table

T	Q_n	Q_{n+1}
0	0	0
0	1	1
1	0	1
1	1	0

Characteristics Table

K	q_{n+1}
0	q_n
1	$q_n$bar

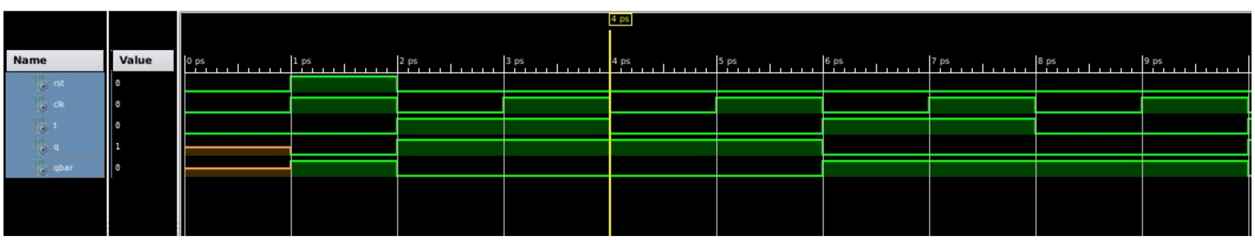
Architecture

```
architecture Behavioral of TFlipflop is
shared variable qt1,qt1bar: std_logic;
begin
p1:process(clk, rst)
begin
            if rst='1' then
                        q<='0';
                        qbar<='1';
                        qt1:='0';
                        qt1bar:='1';
            elsif (clk'event) then
                        if t='1' then
                                    q<=qt1bar;
                                    qbar<=qt1;
                                    qt1:=q;
                                    qt1bar:=qbar;
                        end if;
            end if;
end process;

end Behavioral;
```

TestBench

```
stim_proc: process
   begin
      rst<='0';
            wait for 1 ps;

            rst<='1';
            wait for 1 ps;

            rst<='0';

            loop1:loop
                  t<='1';
                  wait for 1 ps;
                  wait for 1 ps;
                  t<='0';
                  wait for 1 ps;
                  wait for 1 ps;
            end loop;
   end process;
END;
```

Timing Diagram



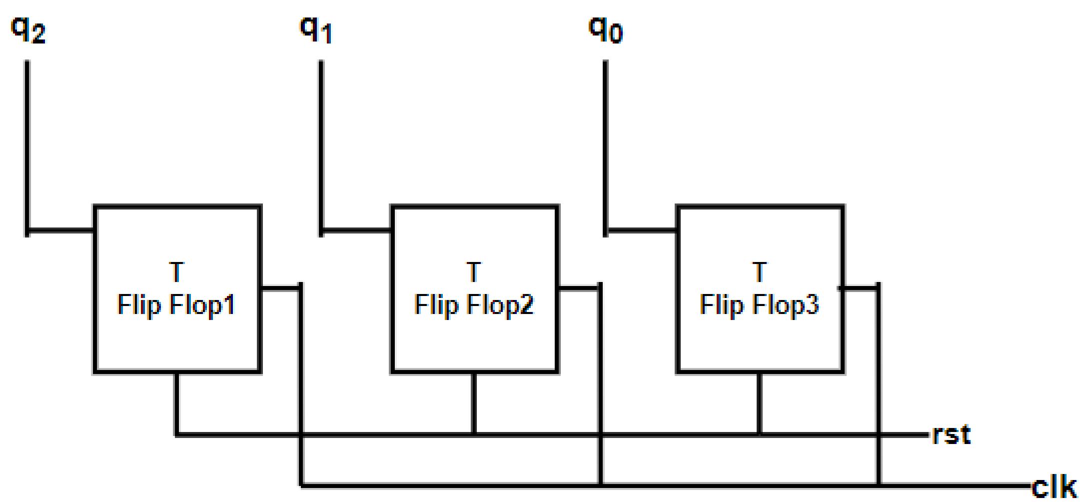
3-bit Up Counter using T Flip-Flop

Description

Implement a 3-bit Up counter using T Flip Flop using component instantiation. Also, write the testbench for it.

T Flip-Flop takes one input T. If T is 0 then the output remains the same and if T is 1 then the output is toggled. The Counter setup is the same as that using SR Flip-Flop. If Q gives the Up Counter output, then $\overline{Q}$will give the Down Counter Output.

Circuit Diagram



Truth Table

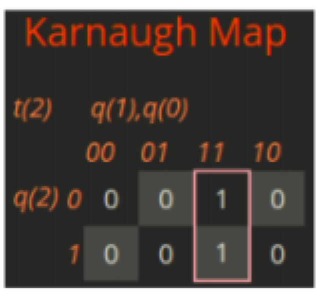
Excitation Table:

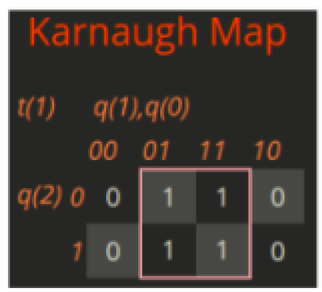
q_n	q_{n+1}	T
0	0	0
0	1	1
1	0	1
1	1	0

State Table:

q_2	q_1	q_0	T_2	T_1	T_0
0	0	0	0	0	1
0	0	1	0	1	1
0	1	0	0	0	1
0	1	1	1	1	1
1	0	0	0	0	1
1	0	1	0	1	1
1	1	0	0	0	1
1	1	1	1	1	1

Calculation





$$t(2) = q(1) \cdot q(0)$$
$$t(1) = q(0)$$
$$t(0) = 1$$

Entity

```
entity TFlipflopCounterUsingComponent is
    Port ( rst : in  STD_LOGIC;
           clk : in  STD_LOGIC;
           qq : inout  STD_LOGIC_VECTOR (2 downto 0);
           qqbar : inout  STD_LOGIC_VECTOR (2 downto 0));
end TFlipflopCounterUsingComponent;
```

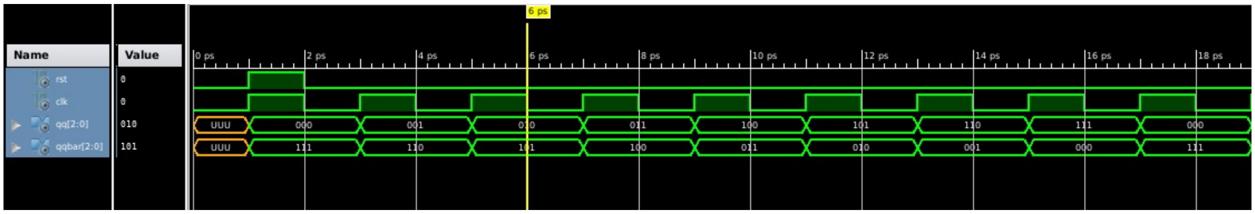
Architecture

```
architecture Behavioral of TFlipflopCounterUsingComponent is
component TFlipflop is
    Port ( rst : in  STD_LOGIC;
           clk : in  STD_LOGIC;
           t : in  STD_LOGIC;
           q : inout  STD_LOGIC;
           qbar : inout  STD_LOGIC);
end component;
signal tt:std_logic_vector(2 downto 0);
begin
            tt(0)<='1';
            tt(1)<=qq(0);
            tt(2)<=qq(1) and qq(0);
            gen2: for k in 0 to 2 generate
                        comm: TFlipflop port map(rst, clk, tt(k), qqbar(k),
qq(k));
            end generate;
end Behavioral;
```

TestBench

```
stim_proc: process
   begin
                  rst<='0';
                  wait for 1 ps;

                  rst<='1';
                  wait for 1 ps;

                  rst<='0';
                  loop
                        wait for 1 ps;
                        wait for 1 ps;
                  end loop;
   end process;
END;
```

Timing Diagram



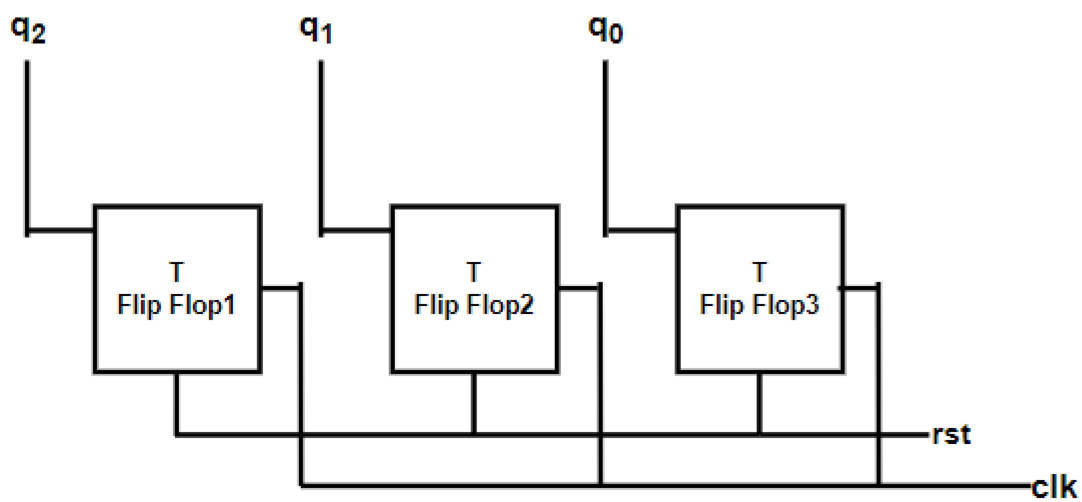
3-bit Down Counter using T Flip-Flop

Description

Implement a 3-bit Down counter using T Flip Flop using behavioral modelling. Also, write the testbench for it.

T Flip-Flop takes one input T. If T is 0 then the output remains the same and if T is 1 then the output is toggled. The Counter setup is the same as that using SR Flip-Flop. If Q gives the Up Counter output, then $\bar{Q}$will give the Down Counter Output.

Circuit Diagram



Truth Table

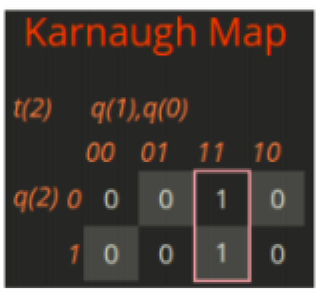
Excitation Table:

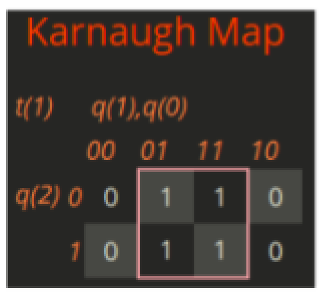
q_n	q_{n+1}	T
0	0	0
0	1	1
1	0	1
1	1	0

State Table:

q_2	q_1	q_0	T_2	T_1	T_0
0	0	0	0	0	1
0	0	1	0	1	1
0	1	0	0	0	1
0	1	1	1	1	1
1	0	0	0	0	1
1	0	1	0	1	1
1	1	0	0	0	1
1	1	1	1	1	1

Calculation





$$t(2) = q(1) \cdot q(0)$$
$$t(1) = q(0)$$
$$t(0) = 1$$

Entity

```
entity TFlipflopDownCounter is
    Port ( rst : in  STD_LOGIC;
           clk : in  STD_LOGIC;
           qqqq : inout  STD_LOGIC_VECTOR (2 downto 0);
           qqqqbar : inout  STD_LOGIC_VECTOR (2 downto 0));
end TFlipflopDownCounter;
```

Architecture

```
architecture Behavioral of TFlipflopDownCounter is
shared variable qqqq1, qqqq1bar:std_logic_vector(2 downto 0);
begin

p1:process(clk,rst)
variable q5,q5bar:std_logic_vector(2 downto 0);

begin
      if rst='1' then
            qqqq<="111";
            qqqqbar<="000";
            qqqq1:="111";
            qqqq1bar:="000";
      elsif (clk'event and clk='1') then

            procc:tFlipFlopDownCounterProcedure(rst,clk,q5(2 downto
0),q5bar(2 downto 0),qqqq1(2 downto 0),qqqq1bar(2 downto 0));

            qqqq(2 downto 0)<=q5(2 downto 0);
            qqqqbar(2 downto 0)<=q5bar(2 downto 0);
            qqqq1(2 downto 0) := q5(2 downto 0);
            qqqq1bar(2 downto 0):=q5bar(2 downto 0);
      end if;

end process;

end Behavioral;
```

T Flipflop Counter in Package

```
procedure TFlipFlopDownCounterProcedure(rst:in std_logic;clk:in
std_logic;qq:inout std_logic_vector;qqbar:inout
std_logic_vector;qqqq1:inout std_logic_vector;qqqq1bar:inout
std_logic_vector) is
variable t,qq1,qq1bar:std_logic_vector(2 downto 0);
variable i:integer;
begin
      if rst='1' then
            qq(2 downto 0):="111";
            qqbar(2 downto 0):="000";
            qq1:="111";
```

```
            qq1bar:="000";
            qqqq1(2 downto 0):="000";
            qqqq1bar(2 downto 0):="111";
      elsif (clk='1') then
            qq1(2 downto 0):=qqqq1(2 downto 0);
            qq1(2 downto 0):=qqqq1(2 downto 0);
            t(0) := '1';
            t(1) := not qq1(0);
            t(2) := (not qq1(1)) and (not qq1(0));
            for i in 0 to 2 loop

prock:TFlipFlopProcedure(rst,clk,t(i),qq1(i),qq1bar(i),qqqq1(i),qqqq1bar(i)
);
            end loop;
            qq(2 downto 0):=qq1(2 downto 0);
            qqbar(2 downto 0):=qq1bar(2 downto 0);
            qqqq1(2 downto 0):=qq1(2 downto 0);
            qqqq1bar(2 downto 0):=qq1bar(2 downto 0);

      end if;
end procedure;
```

TestBench

```
stim_proc: process
   begin
                  rst<='0';
                  wait for 1 ps;

                  rst<='1';
                  wait for 1 ps;

                  rst<='0';
                  loop
                        wait for 1 ps;
                        wait for 1 ps;
                  end loop;
   end process;
END;
```

Timing Diagram