Assignment 2 part 4

Flip-Flop Circuits

Objective:

To construct and study the operations of the following circuits:

(i) RS and Clocked RS Flip-Flop
(ii) D Flip-Flop
(iii) JK and Master-Slave JK Flip-Flop
(iv) T Flip-Flop

Overview:

So far you have encountered with *combinatorial logic*, i.e. circuits for which the output depends only on the inputs. In many instances it is desirable to have the next output depending on the current output. A simple example is a *counter*, where the next number to be output is determined by the current number stored. Circuits that remember their current output or state are often called *sequential logic* circuits. Clearly, sequential logic requires the ability to store the current state. In other words, *memory* is required by sequential logic circuits, which can be created with boolean gates. If you arrange the gates correctly, they will remember an input value. This simple concept is the basis of RAM (random access memory) in computers, and also makes it possible to create a wide variety of other useful circuits.

Memory relies on a concept called **feedback**. That is, the output of a gate is fed back into the input. The simplest possible feedback circuit using two inverters is shown below (Fig.1):

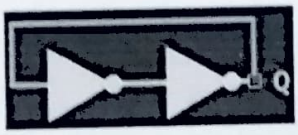


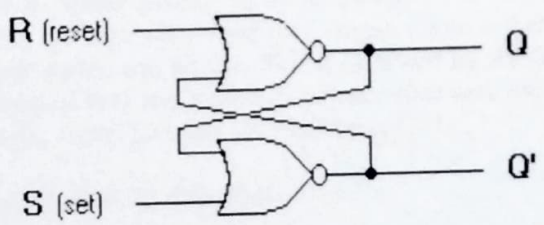
Fig.1: Simplest realization of feedback circuit

If you follow the feedback path, you can see that if Q happens to be 1 (or 0), it will always be 1 (or 0). Since it's nice to be able to control the circuits we create, this one doesn't have much use -- but it does let you see how feedback works. It turns out that in "real" sequential circuits, you can actually use this sort of simple inverter feedback approach. The memory elements in these circuits are called *flip-flops*. A flip-flop circuit has two outputs, one for the normal value and one for the complement value of the stored bit. Binary information can enter a flip-flop in a variety of ways and gives rise to different types of flip-flops.

RS Flip-Flop

RS flip-flop is the simplest possible memory element. It can be constructed from two NAND gates or two NOR gates. Let us understand the operation of the RS flip-flop using NOR gates as shown below using the truth table for 'A NOR B' gate. The inputs R and S are referred to as the Reset and Set inputs, respectively. The outputs Q and Q' are complements of each other and are referred to as the normal and complement outputs, respectively. The binary state of the flip-flop is taken to be the value of the normal output. When Q=1 and Q'=0, it is in the *set state* (or 1-state). When Q=0 and Q'=1, it is in the *reset/clear state* (or 0-state).

Circuit Diagram:



A	B	$\overline{A+B}$
0	0	1
0	1	0
1	0	0
1	1	0

- **S=1 and R=0:** The output of the bottom NOR gate is equal to zero, Q'=0. Hence both inputs to the top NOR gate are equal to 0, thus, Q=1. Hence, the input combination S=1 and R=0 leads to the flip-flop being **set** to Q=1.
- **S=0 and R=1:** Similar to the arguments above, the outputs become Q=0 and Q'=1. We say that the flip-flop is **reset**.
- **S=0 and R=0:** Assume the flip-flop was previously in set (S=1 and R=0) condition. Now changing S to 0 results Q' still at 0 and Q=1. Similarly, when the flip-flop was previously in a reset state (S=0 and R=1), the outputs do not change. Therefore, with inputs S=0 and R=0, the flip-flop holds its state.
- **S=1 and R=1:** This condition violates the fact that both outputs are complements of each other since each of them tries to go to 0, which is not a stable configuration. It is impossible to predict which output will go to 1 and which will stay at 0. In normal operation this condition must be avoided by making sure that 1's are not applied to both inputs simultaneously, thus making it one of the main disadvantages of RS flip-flop.

All the above conditions are summarized in the characteristic table below:

Characteristic Table:

R	S	Q	Q'	Comment
0	0	Q	Q'	Hold state
0	1	1	0	Set
1	0	0	1	Reset
1	1	?	?	Indeterminate

Debounce circuit

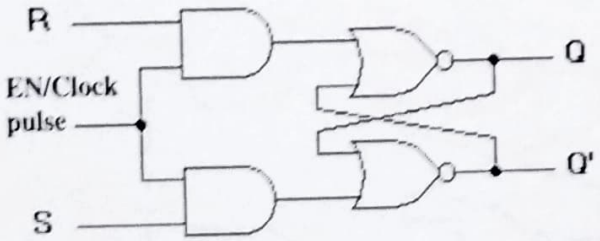
An elementary example using this flip-flop is the debounce circuit. Suppose a piece of electronics is to change state under the action of a mechanical switch. When this switch is moved from position S to R (S=0, R=1), the contacts make and break several times at R before settling to good contact. It is desirable that the electronics should respond to the first contact and then remain stable, rather than switching back and forth as the circuit makes and breaks. This is achieved by RS flip-flop which is reset to Q=0 by the first signal R=1 and remains in a fixed state until the switch is moved back to position S, when the signal S=1 sets the flip-flop to Q=1.

Gated or Clocked RS Flip-Flop

It is sometimes desirable in sequential logic circuits to have a bistable RS flip-flop that only changes state when certain conditions are met regardless of the condition of either the Set or the Reset inputs. By connecting a 2-input AND gate in series with each input terminal of the RS NOR Flip-flop a Gated RS Flip-flop can be created. This extra conditional input is called an "Enable" input and is given the prefix of "EN" as shown below. When the Enable input "EN" = 0, the outputs of the two AND gates are also at logic level 0, (AND Gate principles) regardless of the condition of the two inputs S and R, latching the two outputs Q and Q' into their last known state. When the enable input "EN" = 1, the circuit responds as a normal RS bistable flip-flop with the two AND gates becoming transparent to the Set and Reset signals. This Enable input can also be connected to a clock timing signal adding clock synchronisation to the flip-flop creating what is sometimes called a "Clocked SR Flip-flop".

So a **Gated/Clocked RS Flip-flop** operates as a standard bistable latch but the outputs are only activated when a logic "1" is applied to its EN input and deactivated by a logic "0". The property of this flip-flop is summarized in its characteristic table where Q_n is the logic state of the previous output and Q_{n-1} is that of the next output and the clock input being at logic 1 for all the R and S input combinations.

Circuit Diagram:



Characteristic Table:

Q_n	R	S	Q_{n+1}
0	0	0	0 (Hold)
0	1	0	0
0	0	1	1
0	1	1	Indeterminate
1	0	0	1 (Hold)
1	1	0	0
1	0	1	1
1	1	1	Indeterminate

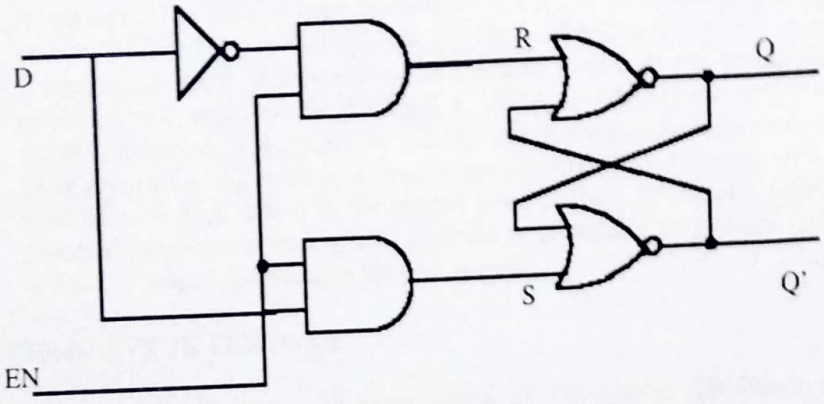
D FLIP-FLOP

An RS flip-flop is rarely used in actual sequential logic because of its undefined outputs for inputs R= S= 1. It can be modified to form a more useful circuit called D flip-flop, where D stands for data. The D flip-flop has only a single data input D as shown in the circuit diagram. That data input is connected to the S input of an RS flip-flop, while the inverse of D is connected to the R input. To allow the flip-flop to be in a holding state, a D-flip flop has a second input called Enable, EN. The Enable-input is AND-ed with the D-input.

- When **EN=0**, irrespective of D-input, the R = S = 0 and the **state is held**.
- When **EN= 1**, the S input of the RS flip-flop equals the D input and R is the inverse of D. Hence, output **Q follows D**, when EN= 1.
- When **EN returns to 0**, the most recent input **D is 'remembered'**.

The circuit operation is summarized in the characteristic table for **EN=1**.

Circuit Diagram:



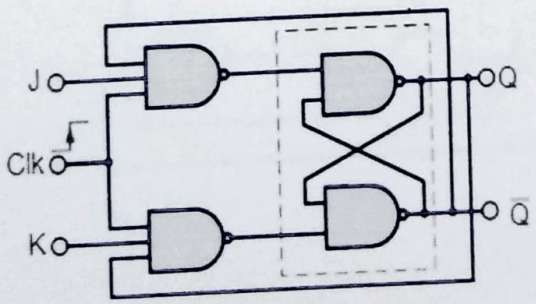
Characteristic Table:

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

JK FLIP-FLOP:

The JK flip flop (JK means Jack Kilby, a Texas instrument engineer, who invented it) is the most versatile flip-flop, and the most commonly used flip flop. Like the RS flip-flop, it has two data inputs, J and K, and an EN/clock pulse input (CP). Note that in the following circuit diagram NAND gates are used instead of NOR gates. It has no undefined states, however. The fundamental difference of this device is the feedback paths to the AND gates of the input, i.e. Q is AND-ed with K and CP and Q' with J and CP.

Circuit Diagram:



Characteristic Table:

Q_n	J	K	Q_{n+1}
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1(Toggle,$\bar{Q}_n$)
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0(Toggle,$\bar{Q}_n$)

The JK flip-flop has the following characteristics:

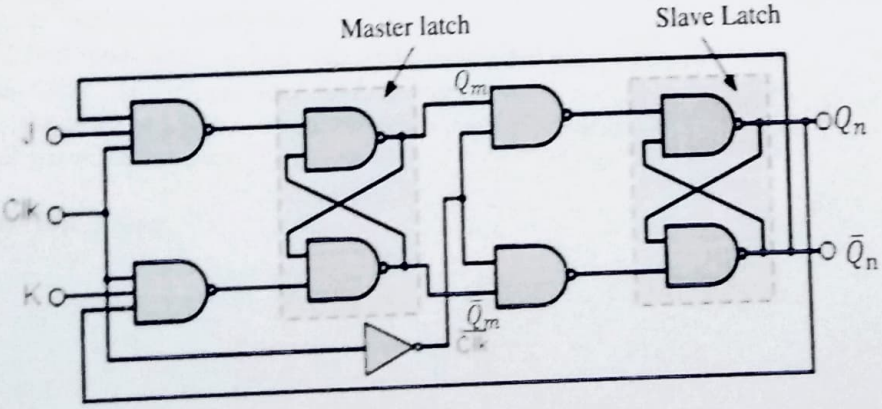
- If one input (J or K) is at logic 0, and the other is at logic 1, then the output is set or reset (by J and K respectively), just like the RS flip-flop.
- If both inputs are 0, then it remains in the same state as it was before the clock pulse occurred; again like the RS flip flop. CP has no effect on the output.
- If both inputs are high, however the flip-flop changes state whenever a clock pulse occurs; i.e., the clock pulse toggles the flip-flop again and again until the CP goes back to 0 as shown in the shaded rows of the characteristic table above. Since this condition is undesirable, it should be eliminated by an improvised form of this flip-flop as discussed in the next section.

MASTER-SLAVE JK FLIP-FLOP:

Although JK flip-flop is an improvement on the clocked SR flip-flop it still suffers from timing problems called "race" if the output Q changes state before the timing pulse of the clock input has time to go "OFF", so the timing pulse period (T) must be kept as short as possible (high frequency). As this is sometimes not possible with modern TTL IC's the much improved Master-Slave J-K Flip-Flop was developed. This eliminates all the timing problems by using two SR flip-flops connected together in series, one for the "Master" circuit, which triggers on the leading edge of the clock pulse and the other, the "Slave" circuit, which triggers on the falling edge of the clock pulse.

The master-slave JK flip flop consists of two flip flops arranged so that when the clock pulse enables the first, or master, it disables the second, or slave. When the clock changes state again (i.e., on its falling edge) the output of the master latch is transferred to the slave latch. Again, toggling is accomplished by the connection of the output with the input AND gates.

Circuit Diagram:



Characteristic Table:

CP	J	K	Q_m	$\bar{Q}_m$	Q_n	$\bar{Q}_n$
0→1	0	0	Hold		Hold	
1→0	0	0	Hold		Hold	
0→1	0	1	0	1	Hold	
1→0	0	1	Hold		0	1
0→1	1	0	1	0	Hold	
1→0	1	0	Hold		1	0
0→1	1	1	Toggle		Hold	
1→0	1	1	Hold		Toggle	

T FLIP-FLOP:

The T flip-flop is a single input version of the JK flip-flop. The T flip-flop is obtained from the JK type if both inputs are tied together.

Circuit Diagram:

Same as Master-Slave JK flip-flop with J=K=1

- The toggle, or T, flip-flop is a bistable device, where the output of the T flip-flop "toggles" with each clock pulse.
- Till CP=0, the output is in hold state (three input AND gate principle).
- When CP=1, for T=0, previous output is memorized by the circuit. When T=1 along with the clock pulse, the output toggles from the previous value as given in the characteristic table below.

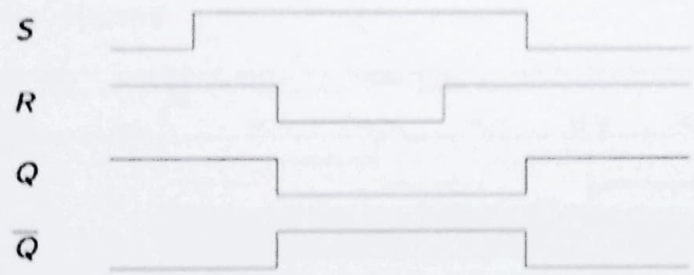
Characteristic Table:

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

VHDL Code for an SR Latch

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

entity sr1 is
   port(r,s:in bit; q,qbar:buffer bit);
end sr1;

architecture virat of sr1 is
   signal s1,r1:bit;
begin
   q<= s nand qbar;
   qbar<= r nand q;
end virat;
```

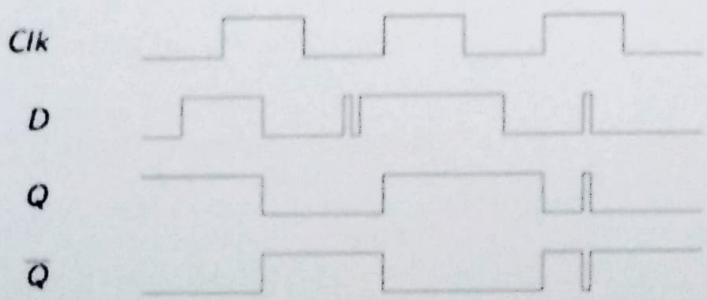
Waveforms



VHDL Code for a D Latch

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

entity D1 is
   port(d:in bit; q,qbar:buffer bit);
end D1;

architecture virat of D1 is
   signal s1,r1:bit;
begin
   q<= d nand qbar;
   qbar<= d nand q;
end virat;
```

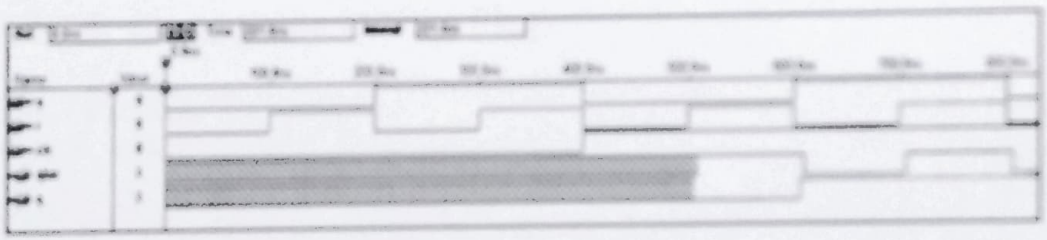
Waveforms



VHDL Code for an SR Flip Flop

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

entity srflip is
   port(r,s,clk:in bit; q,qbar:buffer bit);
end srflip;

architecture virat of srflip is
   signal sl,rl:bit;
begin
   sl<=s nand clk;
   rl<=r nand clk;
   q<= sl nand qbar;
   qbar<= rl nand q;
end virat;
```

Waveforms

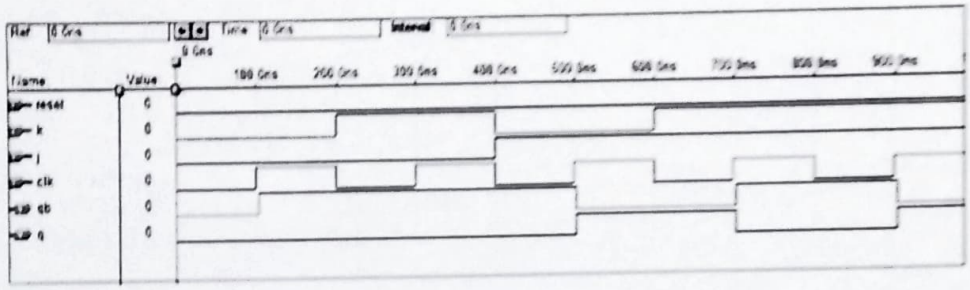


VHDL code for a JK Flip Flop

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity jk is
   port(
      j : in STD_LOGIC;
      k : in STD_LOGIC;
      clk : in STD_LOGIC;
      reset : in STD_LOGIC;
      q : out STD_LOGIC;
      qb : out STD_LOGIC
   );
end jk;

architecture virat of jk is
begin
   jkff : process (j,k,clk,reset) is
   variable m : std_logic := '0';

   begin
      if (reset = '1') then
         m := '0';
      elsif (rising_edge (clk)) then
         if (j/ = k) then
            m := j;
         elsif (j = '1' and k = '1') then
```

```
            m : = not m;
          end if;
        end if;

        q <= m;
        qb <= not m;
    end process jkff;
end virat;
```

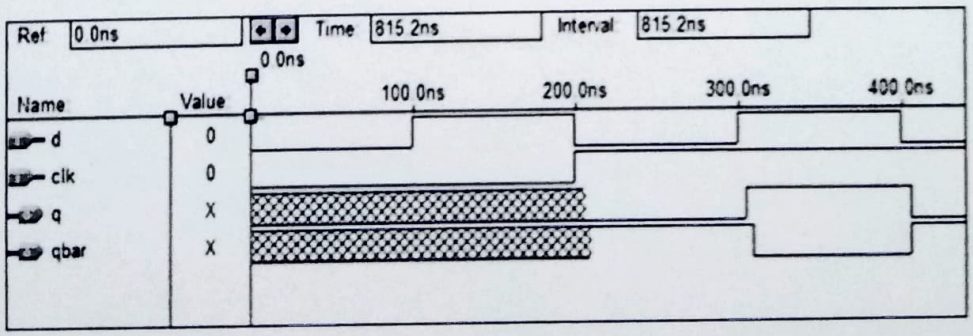
Waveforms



VHDL Code for a D Flip Flop

```
Library ieee;
use ieee.std_logic_1164.all;

entity dflip is
   port(d,clk:in bit; q,qbar:buffer bit);
end dflip;

architecture virat of dflip is
   signal d1,d2:bit;
begin
   d1<=d nand clk;
   d2<=(not d) nand clk;
   q<= d1 nand qbar;
   qbar<= d2 nand q;
end virat;
```

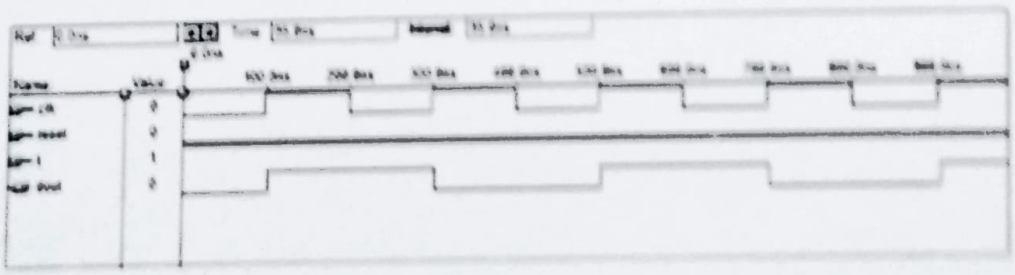
Waveforms



VHDL Code for a T Flip Flop

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity Toggle_flip_flop is
   port(
      t : in STD_LOGIC;
      clk : in STD_LOGIC;
      reset : in STD_LOGIC;
      dout : out STD_LOGIC
   );
end Toggle_flip_flop;

architecture virat of Toggle_flip_flop is
begin
   tff : process (t,clk,reset) is
   variable m : std_logic : = '0';

   begin
      if (reset = '1') then
         m : = '0';
      elsif (rising_edge (clk)) then
         if (t = '1') then
            m : = not m;
         end if;
      end if;
      dout < = m;
   end process tff;
end virat;
```

Waveforms



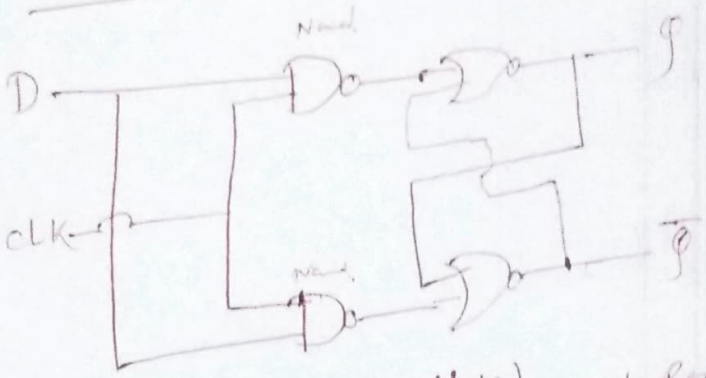
view source

print?

```
1 library ieee;
2 use ieee. std_logic_1164.all;
3 use ieee. std_logic_arith.all;
4 use ieee. std_logic_unsigned.all;
5
6 entity D_FF is
7 PORT( D,CLOCK: in std_logic;
8 Q: out std_logic);
9 end D_FF;
10
11 architecture behavioral of D_FF is
12 begin
13 process(CLOCK)
14 begin
15 if(CLOCK='1' and CLOCK'EVENT) then
16 Q <= D;
17 end if;
18 end process;
19 end behavioral;
```

D Flip Flop

ckt



q	D	q(t+1)
0	0	0
0	1	1
1	0	0
1	1	1

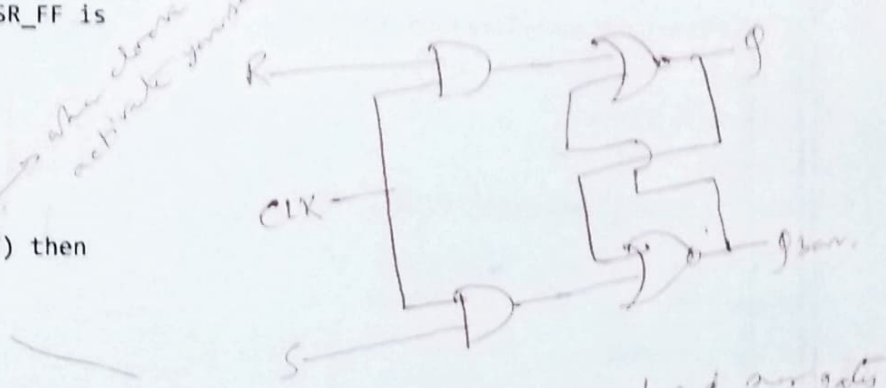
view source

print?

```
1 library ieee;
2 use ieee. std_logic_1164.all;
3 use ieee. std_logic_arith.all;
4 use ieee. std_logic_unsigned.all;
5
6 entity SR_FF is
7 PORT( S,R,CLOCK: in std_logic;
8 Q, QBAR: out std_logic);
9 end SR_FF;
10
11 Architecture behavioral of SR_FF is
12 begin
13 PROCESS(CLOCK)
14 variable tmp: std_logic;
15 begin
16 if(CLOCK='1' and CLOCK'EVENT) then
17 if(S='0' and R='0')then
18 tmp:=tmp;
19 elsif(S='1' and R='1')then
20 tmp:='Z';
21 elsif(S='0' and R='1')then
22 tmp:='0';
23 else
24 tmp:='1';
25 end if;
26 end if;
27 Q <= tmp;
28 QBAR <= not tmp;
29 end PROCESS;
30 end behavioral;
```

SR flip flop

Event is Relates to the signals and it occurs on a signal if the current value of the signal changes. It is a wait process the signal changes the value.



Truth Table

S	R	Q
0	0	0 → indeterminate
0	1	0 Reset
1	0	1 Set
1	1	Unchanged
0	0	1 → indeterminate
0	1	0 Reset
1	0	1 Set
1	1	Unchanged