Weifan Lin
Csc343
03/20/2015

Static Random-Access Memory (SRAM)

Objectives:

The goal of this lab was to create a static random-access memory chip using D-Latches. With an SRAM we were able to store multiple bits at various address locations. And we want to gain knowledge on how to create a memory device that can hold 16-bit wide values and display in the 7-segment HEX displays.

Specifications:

SR-Latch: The SR latch has two inputs S and R, and outputs Q and notQ.

Gated SR-Latch: The gated SR latch has an extra input clock and same outputs as the SR-latch.

D-Latch: The D latch has two inputs D and Clk which is the clock, and outputs Q and notQ.

D-FlipFlop: The D-flipflop has two inputs D and Clk, and two outputs Q and notQ.

T-FlipFlop: The T-flipflop has two inputs T and Clk, and output Q.

JK-FlipFlop: The JK-flipflop has three inputs J, K and Clk, and output Q.

Functionality:

<u>Latches</u>

The SR latch is the most fundamental latch. It takes two inputs, set and reset, and has two outputs, Q and notQ. It has a set state when S is held high and R is low, a reset state when

S is held low and R is held high, a hold state is when both S and R are held low and it remembers the previous state, and an unstable state is when S and R are both held high. The gated SR latch is nearly identical to the standard SR latch, the only difference is that it has an extra input called the enable or control. Depending on the value of this input, it controls whether the output Q will be changed or not. When the enable input is held high for example, signals can pass through the input gates and output is produced. When the enable is held low, however, the latch becomes closed and retains the state that was last left when enable was high. Similar to the SR latch, the gated SR latch still has an undefined state when both R and S are set to 1.

The D latch is a solution to the unstable state of the SR latch, because it prevents the both S and R inputs to be equaled and thus removes the possibility of having the unstable state when S and R were both set to 1. It has only two inputs, the data and the clock. While the clock is high, the output is value of input data. While the clock is low, the output stays the same and input data has no effect on the output.

Flip-Flops

The master-slave D flip-flop is built using two D latches in series, with one of their clock inputs being inverted. It is known as a master-slave because the second (slave) latch responds to changes in the first (master) latch. Our master-slave D flip-flop in this case was negative-edge triggered, meaning that captured input would only be output on a falling edge, when the Clock value went from 1 to 0.

The positive-edge D flip-flop was constructed using three SR NAND latches. The two latches on the left, which are both responsible for input, process the data and clock inputs

for the output, which is represented by the single latch on the right. Since the flip-flop is positive-edged, the output is responsive to the rising edge (from 0 to 1) of the clock input. The T flip-flop was created by modifying a positive-edge D flip-flop and connecting it to a T input through a XOR gate. The input T of this flip-flop determines whether to hold or change the output whenever the clock pulses. For example, if T is high while current output is 0, the next output will be 1. If T is low while current output is 0, the next output will remain 0.

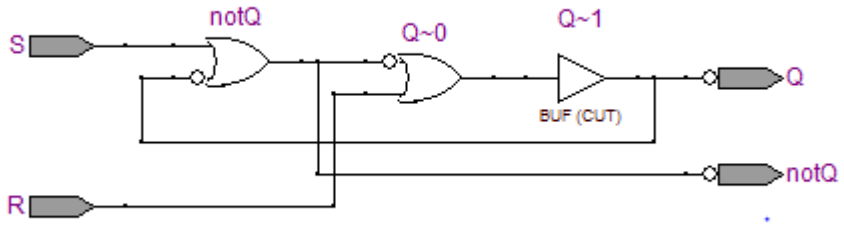
The JK flip-flop was constructed using the positive-edged D flip-flop, which takes in inputs J, K, and Clock. The JK flip-flop is similar to the SR latch in that it has a set state and a reset state when J = 1, K= 0 and J = 0, K = 1 respectively. However, unlike the SR latch, but functionally similar to the T flip-flop, when both J and K inputs are set to 1, the flip-flop is toggled. This results in the next output being the complement of the current output. The JK flip-flop is actually a universal flip-flop as it can be configured to work like a D-flip flop, T flip-flop, or an SR latch/flip-flop.

Design:

SR-Latch VHDL:

```
1   LIBRARY IEEE;
2   use IEEE.STD_LOGIC_1164.ALL;
3
4   entity SR_latch is
5       port( S, R    : in STD_LOGIC;
6             Q, notQ : buffer STD_LOGIC );
7   end SR_latch;
8
9   architecture behav of SR_latch is
10  begin
11      Q <= (R NOR notQ);
12      notQ <= (S NOR Q);
13  end behav;
```

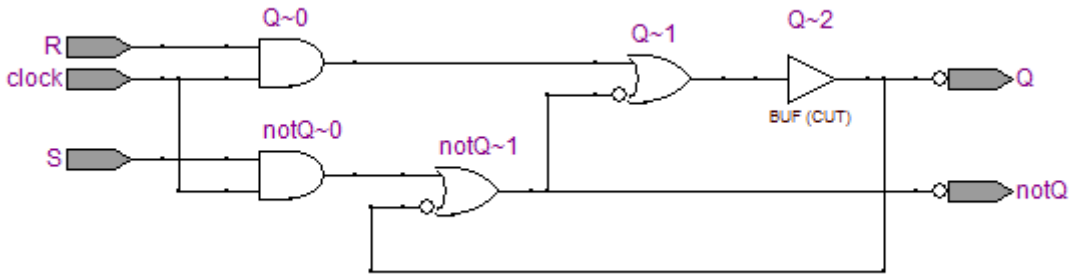
SR-Latch RTL Viewer:



Gated SR-Latch VHDL:

```
1   library IEEE;
2   use IEEE.STD_LOGIC_1164.ALL;
3
4   entity Gated_SR_Latch is
5       Port ( S,R : in STD_LOGIC;
6              clock : in STD_LOGIC;
7              Q, notQ : buffer STD_LOGIC );
8   end Gated_SR_Latch;
9
10  architecture behav of Gated_SR_Latch is
11
12  begin
13      Q <= (clock AND R) NOR notQ;
14      notQ <= (clock AND S) NOR Q;
15  end behav;
```

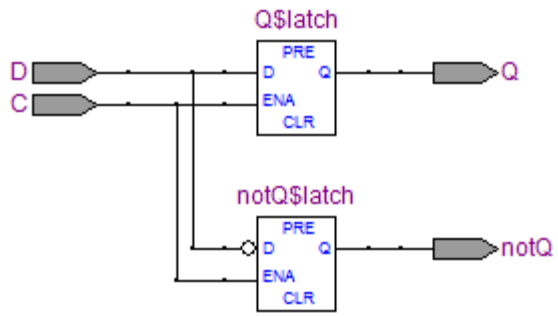
Gated SR-Latch RTL Viewer:



D-Latch VHDL:

```
1   LIBRARY IEEE;
2   use IEEE.STD_LOGIC_1164.ALL;
3
4   entity SR_latch is
5       port( S, R    : in STD_LOGIC;
6             Q, notQ : buffer STD_LOGIC );
7   end SR_latch;
8
9   architecture behav of SR_latch is
10  begin
11      Q <= (R NOR notQ);
12      notQ <= (S NOR Q);
13  end behav;
```

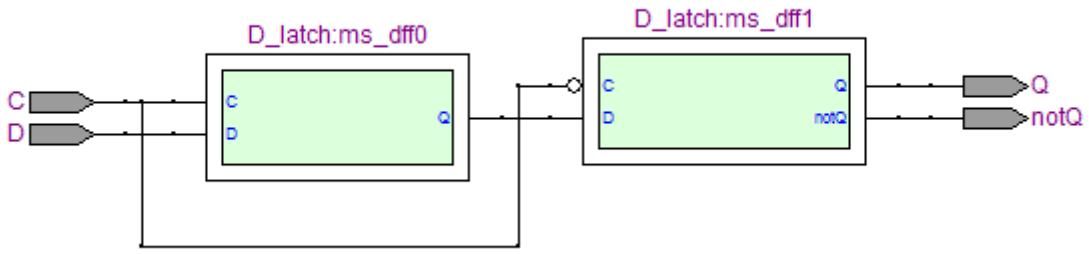
D-Latch Viewer:



Master-slave D Flip-Flop VHDL:

```vhdl
LIBRARY IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity Master_Slave_Dff is
    port( D, C    : in STD_LOGIC;
          Q, notQ : buffer STD_LOGIC );
end Master_Slave_Dff;

architecture behav of Master_Slave_Dff is
component D_latch is
  port( D, C    : in STD_LOGIC;
          Q, notQ : buffer STD_LOGIC );
end component;
   signal Cm, Cs, Qm, Qs, NQm, NQs : std_logic;

begin
    Cm <= C;
    Cs <= not C;

    ms_dff0: D_latch port map (D, Cm, Qm, NQm);
    ms_dff1: D_latch port map (Qm, Cs, Qs, NQs);

    Q <= Qs;
    notQ <= NQs;

end behav;

```

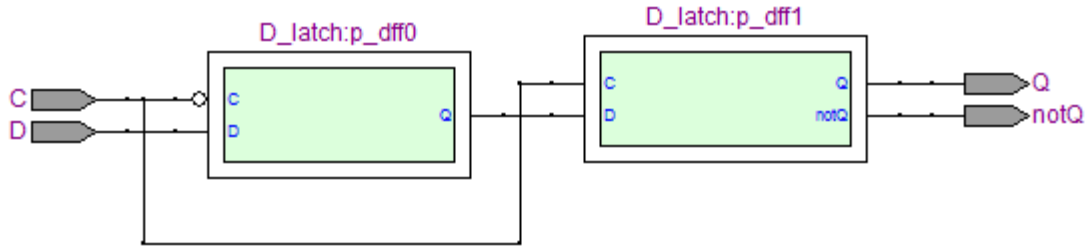
Master-slave D Flip-Flop RTL Viewer:



Positive D Flip-Flop VHDL:

```vhdl
LIBRARY IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity Positive_Dff is
    port( D, C    : in STD_LOGIC;
          Q, notQ : buffer STD_LOGIC );
end Positive_Dff;

architecture behav of Positive_Dff is
component D_latch is
  port( D, C    : in STD_LOGIC;
          Q, notQ : buffer STD_LOGIC );
end component;
   signal C1, C2, Q1, Q2, NQ1, NQ2 : std_logic;

begin
    C1 <= not C;
    C2 <= C;

    p_dff0: D_latch port map (D, C1, Q1, NQ1);
    p_dff1: D_latch port map (Q1, C2, Q2, NQ2);

    Q <= Q2;
    notQ <= NQ2;

end behav;
```

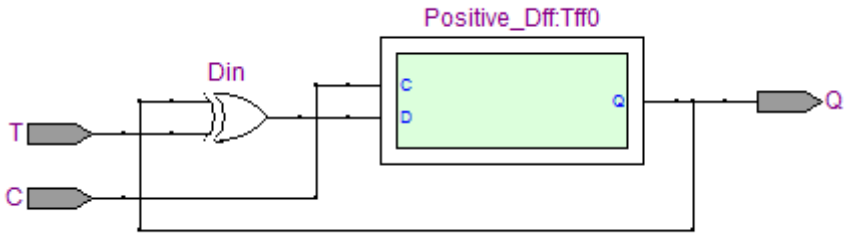
Positive D Flip-Flop RTL Viewer:



T Flip-Flop VHDL:

```vhdl
LIBRARY IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity T_Flip_Flop is
    port( T, C    : in STD_LOGIC;
          Q : buffer STD_LOGIC );
end T_Flip_Flop ;

architecture behav of T_Flip_Flop  is
component Positive_Dff is
  port( D, C    : in STD_LOGIC;
          Q, notQ : buffer STD_LOGIC );
end component;

   signal Din, Q1 : std_logic;

begin
    Din <= T xor Q;

    Tff0: Positive_Dff port map (Din, C, Q1);

    Q <= Q1;

end behav;
```

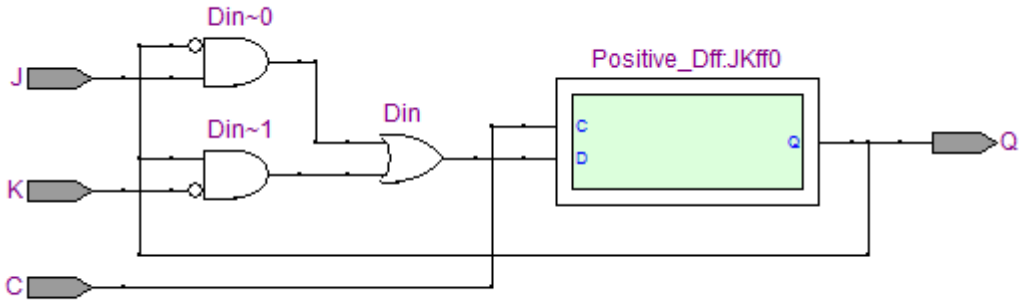
T Flip-Flop RTL Viewer:



JK Flip-Flop VHDL:

```vhdl
LIBRARY IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity JK_Flip_Flop is
    port( J, K, C    : in STD_LOGIC;
          Q : buffer STD_LOGIC );
end JK_Flip_Flop ;

architecture behav of JK_Flip_Flop  is
component Positive_Dff is
  port( D, C    : in STD_LOGIC;
          Q, notQ : buffer STD_LOGIC );
end component;

   signal Din, Q1 : std_logic;

begin
    Din <= (J and not Q) or (not K and Q);

    JKff0: Positive_Dff port map (Din, C, Q1);

    Q <= Q1;

end behav;
```

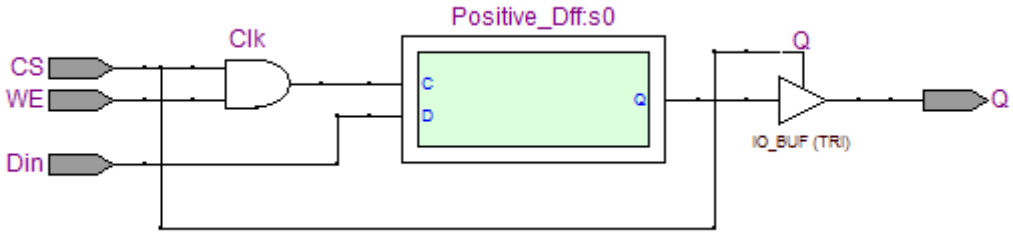
JK Flip-Flop RTL Viewer:



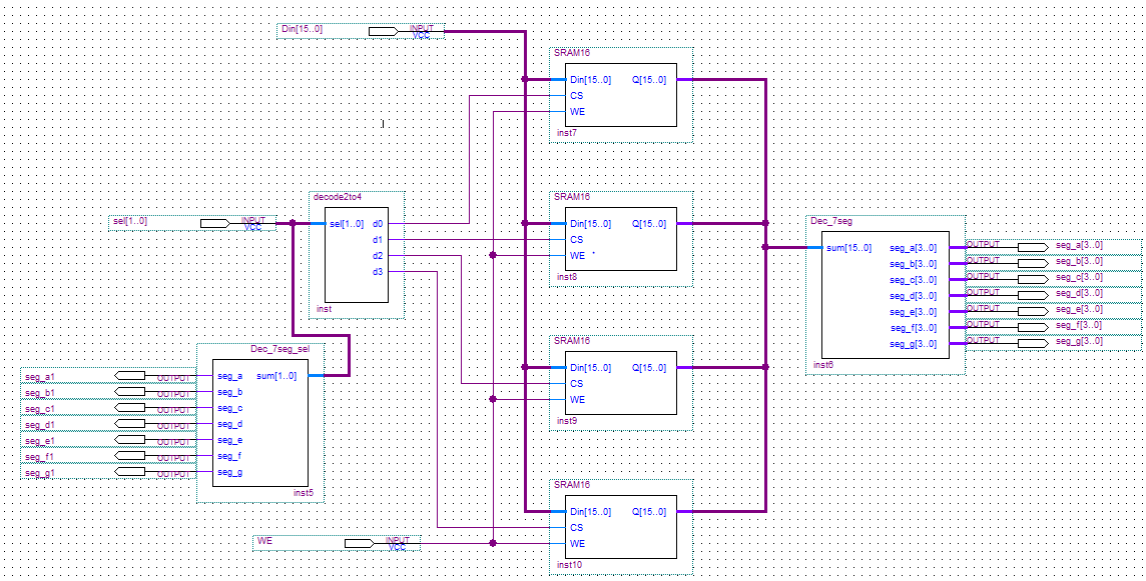
SRAM VHDL:

```vhdl
LIBRARY IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity SRAM is
    port( Din, CS, WE    : in STD_LOGIC;
          Q : out STD_LOGIC );
end SRAM ;

architecture behav of SRAM  is
component Positive_Dff is
        port( D, C    : in STD_LOGIC;
          Q, notQ : buffer STD_LOGIC );
end component;

   signal Clk, Q0 : std_logic;

begin
    Clk <= CS and WE;

    s0: Positive_Dff port map (Din, Clk, Q0);

    Q <= Q0 when (CS = '1') else 'Z';

end behav;
```

SRAM RTL Viewer:

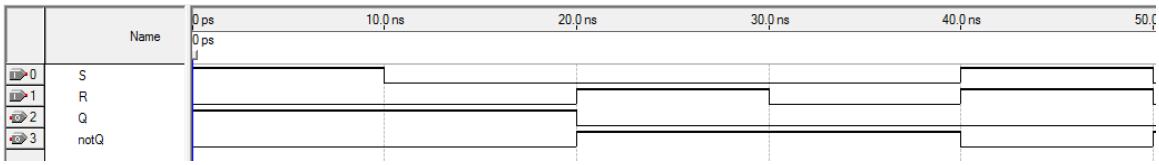


SRAM 4x16:



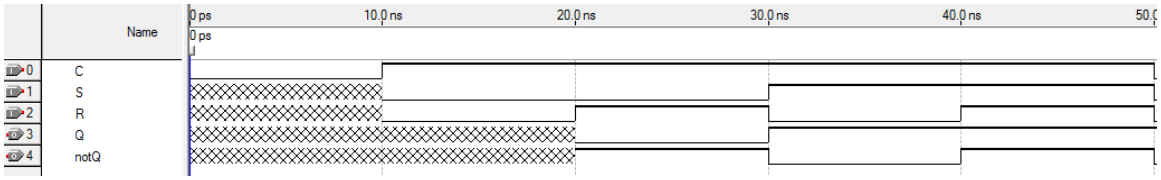
Simulation:

SR Latch Waveform



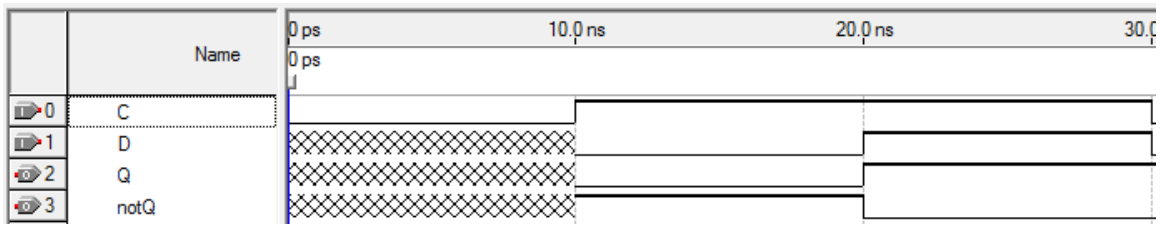
For the standard SR Latch, when the Set input is held high Reset is low, the output of Q is 1, this is called set state as we can see in the first interval. When Set and Reset are held low, the previous output is maintained and thus in this case remains as 1, seen in the second interval. When Reset is held high and Set is held low, this is known as the reset state and the output is set back to 0, as we can see in the third interval. In the fifth interval we have the undefined state of when Reset and Set are both held high and we can notice that Q and notQ both have a value of 0, which should not be possible since they are complements of each other.

Gated SR Latch Waveform



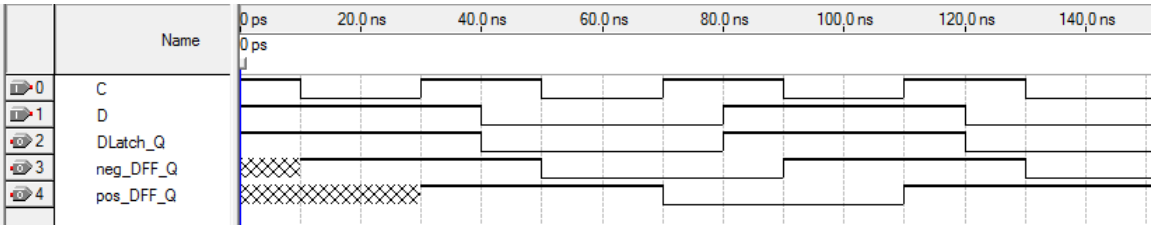
In the gated SR Latch, the circuit will only respond when the Clock (input C) is held high. In the first interval, C is held low, then no matter what inputs are Q has no output. In the second interval we have a hold state, but there is still no output even though the clock is high because there was no output in the previous state. In the third interval, C is held high, set is high and reset is low, thus we have Q with output value of 0. In the fourth interval, C is held high and Set is high, thus we have Q with an output value of 1. In the fifth interval, we again have an undefined state where Q = notQ which should be impossible.

D Latch Waveform



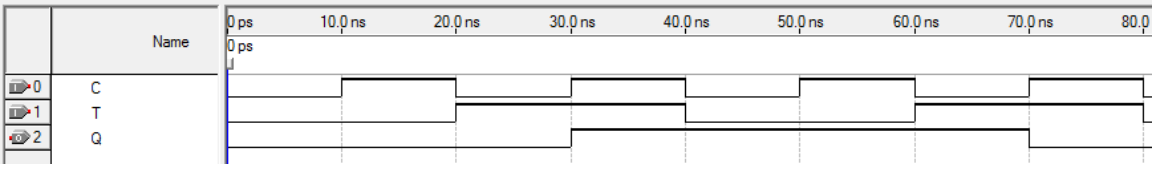
For the D latch, when the Clock is held low, the input of D has no effect on the output as we can see in the first interval. When the Clock is held high, the output value is exactly what the input value is. In the second interval, D has a value of 0 and so does Q. In the third interval, D has a value of 1 and so does Q.

D Latch, Positive DFF, and Negative Master Slave DFF

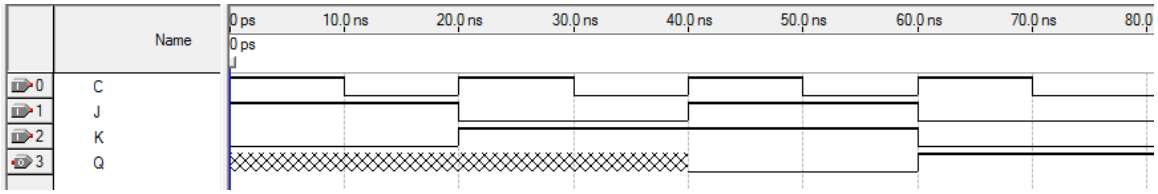


In this waveform, we can see the different behaviors of the D Latch, Positive D Flip-flop, and the Negative Master-Slave D Flip-flop. For the D Latch's Q output, we can see that it follows whatever input D is when Clock is high, and will remember the output from when Clock was last high whenever Clock goes low. For the negative master-slave D flip-flop, an output is not displayed until the Clock meets a falling edge (when it goes from high to low). Its output remains at 1 until the Clock's next falling edge at ~50 nanoseconds, which occurs when D has an input value of 0, and thus the master-slave's output falls to 0 at that moment as well. For the positive D flip-flop, an output is not triggered until the Clock meets a rising edge (when it goes from low to high), which can be seen happening at ~30 nanoseconds. At this moment, input D value is 1, and thus the positive DFF has an output value of 1 until the next rising edge occurs at ~70 nanoseconds. Here, D has an input value of 0 and the positive DFF's output changes to 0 as well.

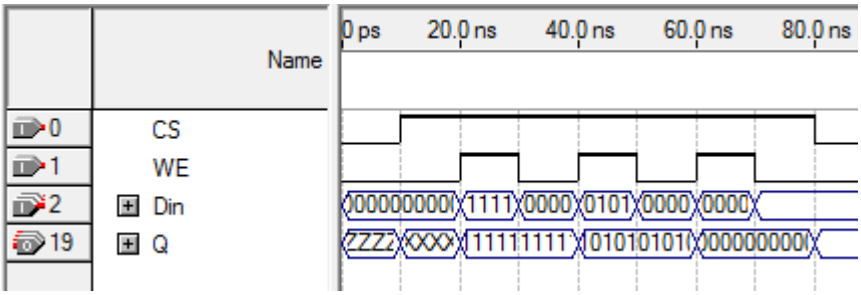
T Flip-Flop Waveform



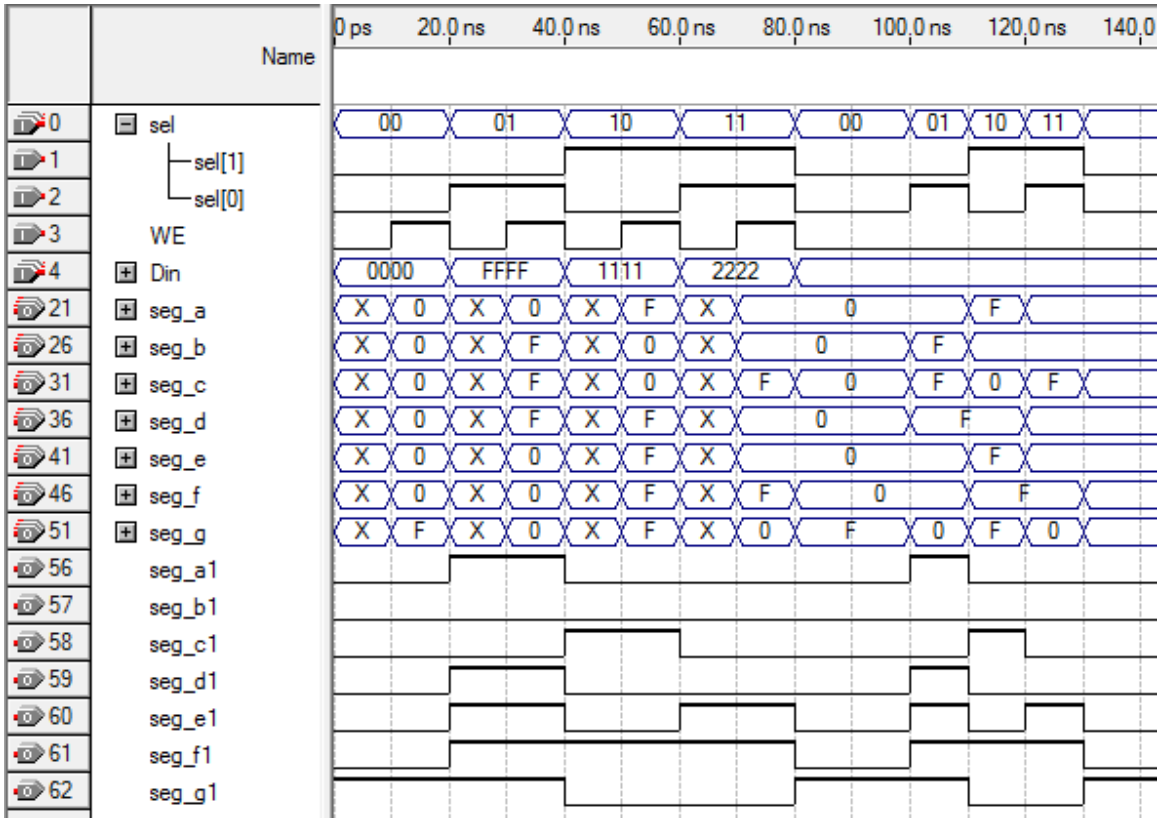
JK Flip-Flop Waveform



SRAM Waveform



SRAM 4x16 Waveform



Conclusion:

Latches, flip-flops, and registers are all basic storage elements. Comparing to normal combinational logic circuits, they have memory and can remember states from a previous time. Latches can propagate input to their outputs as long as the Clock is held high, whereas registers only transfer input to the output during the Clock's rising or falling edges, depending on whether it is positive-edged triggered or negative-edged triggered. SRAM units are more effective than registers at storing data because specific addresses can be chosen to be written to, and they won't be destructed unless overwritten.

Appendix:

Pin Assignments:

to, location
Din[0], PIN_N25
Din[1], PIN_N26
Din[2], PIN_P25
Din[3], PIN_AE14
Din[4], PIN_AF14
Din[5], PIN_AD13
Din[6], PIN_AC13
Din[7], PIN_C13
Din[8], PIN_B13
Din[9], PIN_A13
Din[10], PIN_N1
Din[11], PIN_P1
Din[12], PIN_P2
Din[13], PIN_T7
Din[14], PIN_U3
Din[15], PIN_U4

sel[0], PIN_V1
sel[1], PIN_V2

WE, PIN_G26

seg_a[0], PIN_AF10
seg_b[0], PIN_AB12
seg_c[0], PIN_AC12
seg_d[0], PIN_AD11
seg_e[0], PIN_AE11
seg_f[0], PIN_V14
seg_g[0], PIN_V13
seg_a[1], PIN_V20
seg_b[1], PIN_V21
seg_c[1], PIN_W21
seg_d[1], PIN_Y22
seg_e[1], PIN_AA24
seg_f[1], PIN_AA23
seg_g[1], PIN_AB24

seg_a[2], PIN_AB23
seg_b[2], PIN_V22
seg_c[2], PIN_AC25
seg_d[2], PIN_AC26
seg_e[2], PIN_AB26
seg_f[2], PIN_AB25
seg_g[2], PIN_Y24
seg_a[3], PIN_Y23
seg_b[3], PIN_AA25
seg_c[3], PIN_AA26
seg_d[3], PIN_Y26
seg_e[3], PIN_Y25
seg_f[3], PIN_U22
seg_g[3], PIN_W24

seg_a1, PIN_R2
seg_b1, PIN_P4
seg_c1, PIN_P3
seg_d1, PIN_M2
seg_e1, PIN_M3
seg_f1, PIN_M5
seg_g1, PIN_M4

SR Latch:

```
LIBRARY IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity SR_latch is
        port( S, R    : in STD_LOGIC;
              Q, notQ : buffer STD_LOGIC );
end SR_latch;

architecture behav of SR_latch is
begin
        Q <= (R NOR notQ);
        notQ <= (S NOR Q);
end behav;
```

Gated SR Latch:

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

entity Gated_SR_Latch is
    Port ( S,R : in STD_LOGIC;
           clock : in STD_LOGIC;
           Q, notQ : buffer STD_LOGIC );
end Gated_SR_Latch;

architecture behav of Gated_SR_Latch is

begin
          Q <= (clock AND R) NOR notQ;
          notQ <= (clock AND S) NOR Q;
end behav;
```

D Latch:

```
LIBRARY IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity D_latch is
          port( D, C    : in STD_LOGIC;
                Q, notQ : buffer STD_LOGIC );
end D_latch;

architecture behav of D_latch is
begin
          process(D,C)
          begin
                    if C='1' then
                    Q <= D;
                    notQ <= not D;
                    end if;
          end process;
end behav;
```

Master Slave DFF:

```
LIBRARY IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity Master_Slave_Dff is
        port( D, C    : in STD_LOGIC;
              Q, notQ : buffer STD_LOGIC );
end Master_Slave_Dff;

architecture behav of Master_Slave_Dff is
component D_latch is
  port( D, C    : in STD_LOGIC;
              Q, notQ : buffer STD_LOGIC );
end component;
   signal Cm, Cs, Qm, Qs, NQm, NQs : std_logic;

begin
        Cm <= C;
        Cs <= not C;

        ms_dff0: D_latch port map (D, Cm, Qm, NQm);
        ms_dff1: D_latch port map (Qm, Cs, Qs, NQs);

        Q <= Qs;
        notQ <= NQs;

end behav;
```

Positive DFF:

```
LIBRARY IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity Positive_Dff is
        port( D, C    : in STD_LOGIC;
              Q, notQ : buffer STD_LOGIC );
end Positive_Dff;

architecture behav of Positive_Dff is
```

```
component D_latch is
  port( D, C    : in STD_LOGIC;
              Q, notQ : buffer STD_LOGIC );
end component;
   signal C1, C2, Q1, Q2, NQ1, NQ2 : std_logic;

begin
          C1 <= not C;
          C2 <= C;

          p_dff0: D_latch port map (D, C1, Q1, NQ1);
          p_dff1: D_latch port map (Q1, C2, Q2, NQ2);

          Q <= Q2;
          notQ <= NQ2;

end behav;
```

JK FF:

```
LIBRARY IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity JK_Flip_Flop is
          port( J, K, C    : in STD_LOGIC;
               Q : buffer STD_LOGIC );
end JK_Flip_Flop ;

architecture behav of JK_Flip_Flop  is
component Positive_Dff is
  port( D, C    : in STD_LOGIC;
              Q, notQ : buffer STD_LOGIC );
end component;

   signal Din, Q1 : std_logic;

begin
          Din <= (J and not Q) or (not K and Q);
```

```
        JKff0: Positive_Dff port map (Din, C, Q1);

        Q <= Q1;

end behav;
```

SRAM:

```
LIBRARY IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity SRAM is
        port( Din, CS, WE    : in STD_LOGIC;
            Q : out STD_LOGIC );
end SRAM ;

architecture behav of SRAM  is
component Positive_Dff is
                port( D, C    : in STD_LOGIC;
            Q, notQ : buffer STD_LOGIC );
end component;

  signal Clk, Q0 : std_logic;

begin
        Clk <= CS and WE;

        s0: Positive_Dff port map (Din, Clk, Q0);

        Q <= Q0 when (CS = '1') else 'Z';

end behav;
```

SRAM16:

```
LIBRARY IEEE;
use IEEE.STD_LOGIC_1164.ALL;
```

```
entity SRAM16 is
        port( Din : in STD_LOGIC_VECTOR(15 downto 0);
            CS, WE    : in STD_LOGIC;
            Q : out STD_LOGIC_VECTOR(15 downto 0) );
end SRAM16 ;

architecture behav of SRAM16  is
component SRAM is
        port( Din, CS, WE    : in STD_LOGIC;
            Q : out STD_LOGIC );
end component;

  signal ans : std_logic_vector(15 downto 0);

begin


s16_s0: SRAM port map (Din(0), CS, WE, ans(0));
        s16_s1: SRAM port map (Din(1), CS, WE, ans(1));
        s16_s2: SRAM port map (Din(2), CS, WE, ans(2));
        s16_s3: SRAM port map (Din(3), CS, WE, ans(3));
        s16_s4: SRAM port map (Din(4), CS, WE, ans(4));
        s16_s5: SRAM port map (Din(5), CS, WE, ans(5));
        s16_s6: SRAM port map (Din(6), CS, WE, ans(6));
        s16_s7: SRAM port map (Din(7), CS, WE, ans(7));
        s16_s8: SRAM port map (Din(8), CS, WE, ans(8));
        s16_s9: SRAM port map (Din(9), CS, WE, ans(9));
        s16_s10: SRAM port map (Din(10), CS, WE, ans(10));
        s16_s11: SRAM port map (Din(11), CS, WE, ans(11));
        s16_s12: SRAM port map (Din(12), CS, WE, ans(12));
        s16_s13: SRAM port map (Din(13), CS, WE, ans(13));
        s16_s14: SRAM port map (Din(14), CS, WE, ans(14));
        s16_s15: SRAM port map (Din(15), CS, WE, ans(15));

        Q(15) <= ans(15) when (CS = '1') else 'Z';
        Q(14) <= ans(14) when (CS = '1') else 'Z';
        Q(13) <= ans(13) when (CS = '1') else 'Z';
        Q(12) <= ans(12) when (CS = '1') else 'Z';
        Q(11) <= ans(11) when (CS = '1') else 'Z';
        Q(10) <= ans(10) when (CS = '1') else 'Z';
```

```
    Q(9) <= ans(9) when (CS = '1') else 'Z';
    Q(8) <= ans(8) when (CS = '1') else 'Z';
    Q(7) <= ans(7) when (CS = '1') else 'Z';
    Q(6) <= ans(6) when (CS = '1') else 'Z';
    Q(5) <= ans(5) when (CS = '1') else 'Z';
    Q(4) <= ans(4) when (CS = '1') else 'Z';
    Q(3) <= ans(3) when (CS = '1') else 'Z';
    Q(2) <= ans(2) when (CS = '1') else 'Z';
    Q(1) <= ans(1) when (CS = '1') else 'Z';
    Q(0) <= ans(0) when (CS = '1') else 'Z';

end behav;
```