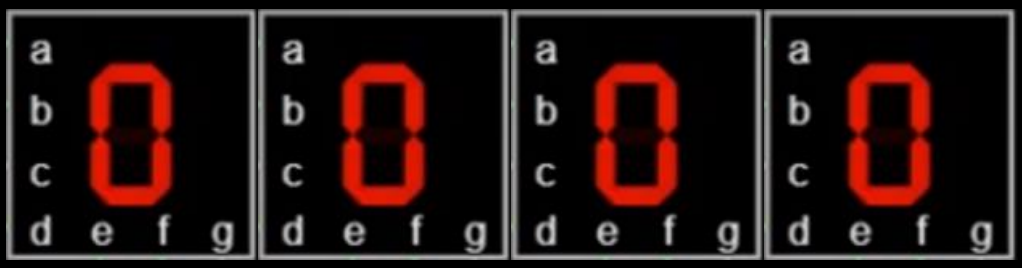
Digital Stopwatch

M1 M0 S1 S0



Team members:

Ahmed Bassam Abdelaal 202101074
- Mohamed Elsayed Ibrahim 2021004384
- Khaled Ashraf Mousa 202100751

Table of Contents

1- Design Goals

We have been asked to design a stopwatch and count down in a single circuit.

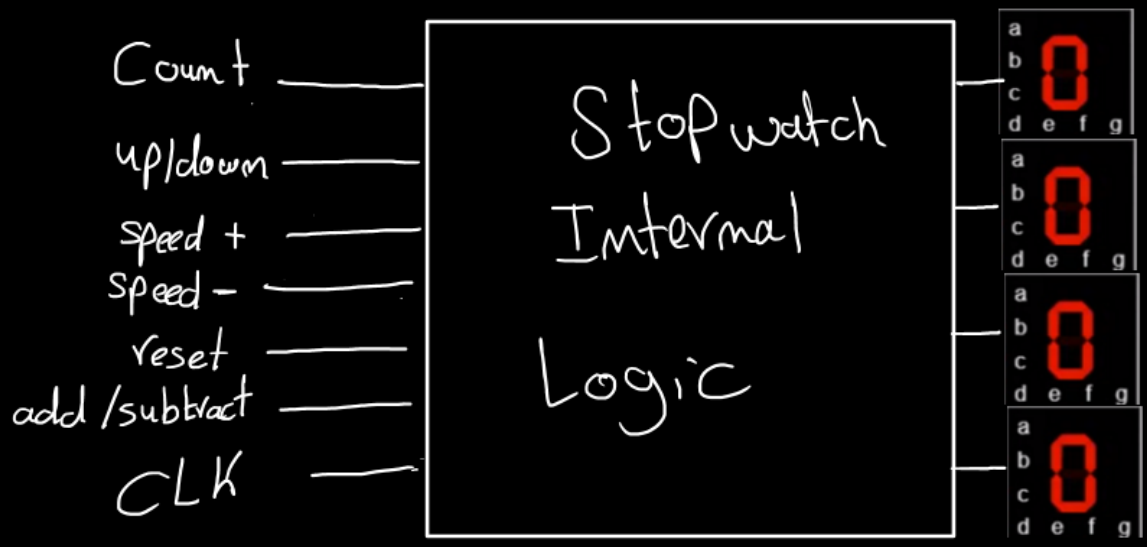
The features are:

- Count up to a specific second then stop
- Count down to a specific second then stop
- You can stop counting whenever you want
- While counting up you can add 2 minutes
- While counting down you can subtract 2 minutes
- Timing rate can be 0.5x or 2x
- The output should be visible on a seven-segment display

Based on these features that we want to do, we thought that for this project we need some blocks to do these features:

- Block for counting
- Block for specifying counting mode up/down
- Block for addition and subtraction
- Block for changing the speed of Counting
- Block for stopping the counting
- Block for displaying the output

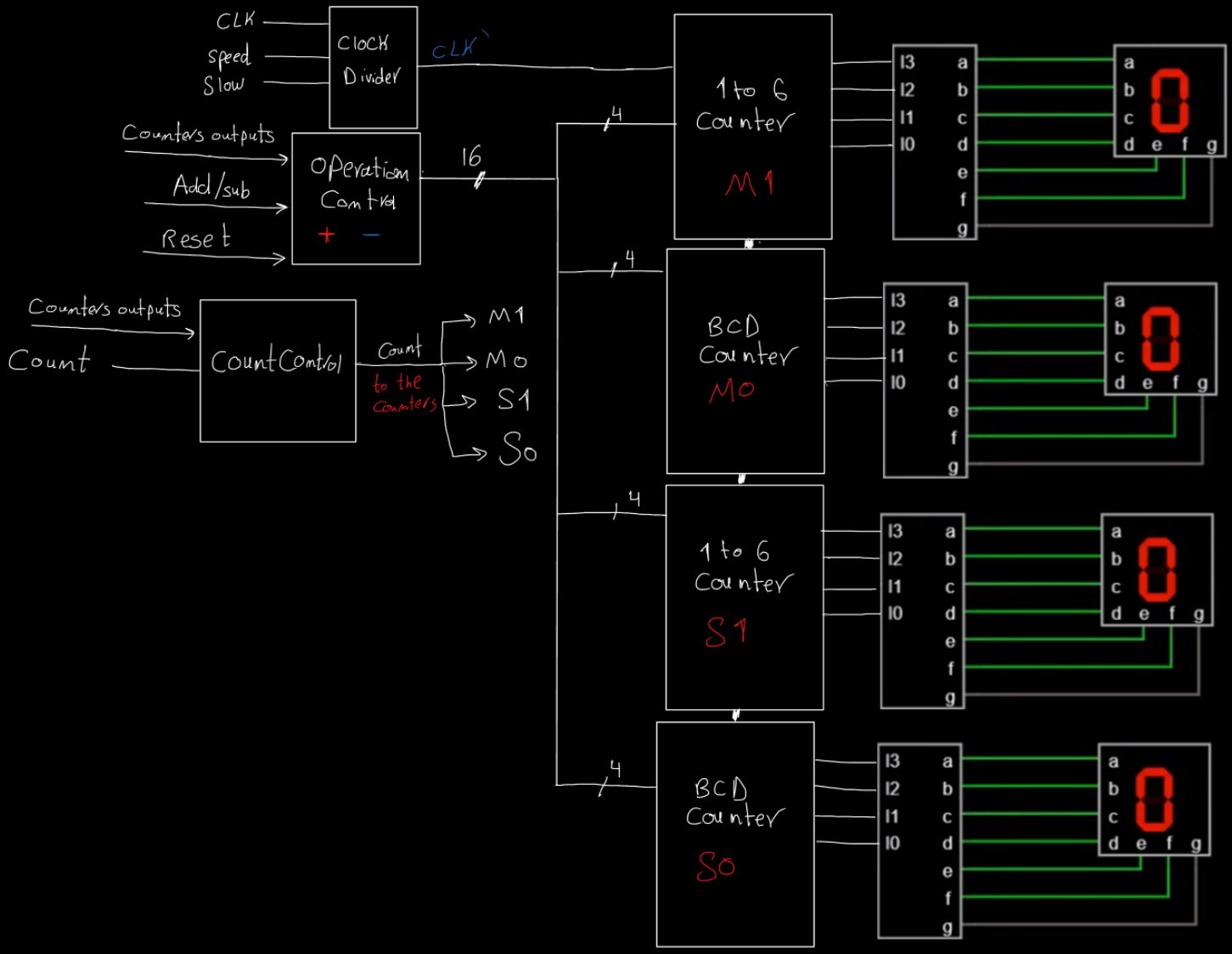
So as a start we want to make this:



In the next section we gathered the different methods thought of to implement such a thing

2- Brainstorming & chosen Approaches

2.1 Stopwatch with synchronous Counter



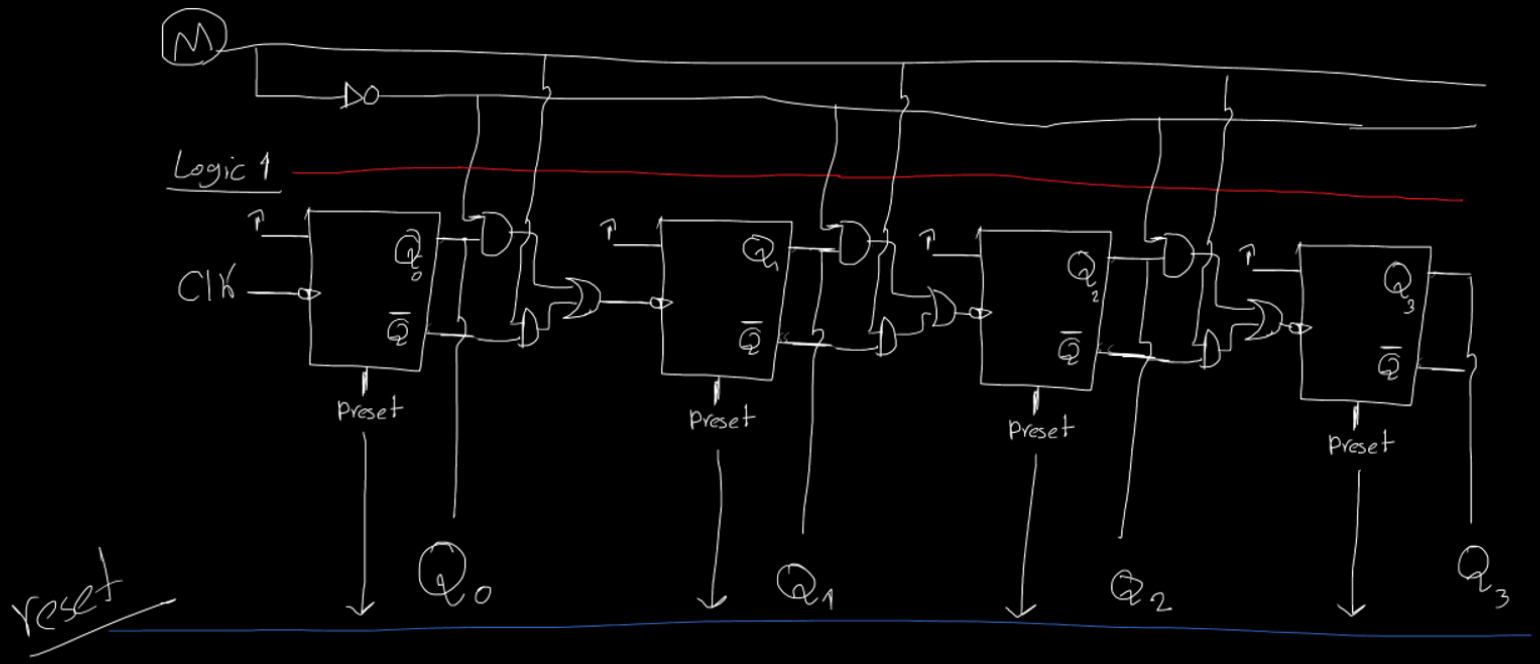
You can see that:

- We used both BCD synchronous counters and 0to5 synchronous counters
- Also the output of the counters is a feedback to the operation control and count control blocks.
- Count control is the block that stops the counter on max or min or when user asks to stop.
- Operation Control is used to add or subtract 2 minutes but if the addition or subtraction can’t exceed the maximum or minimum values.

2.2 Stopwatch with asynchronous Counter

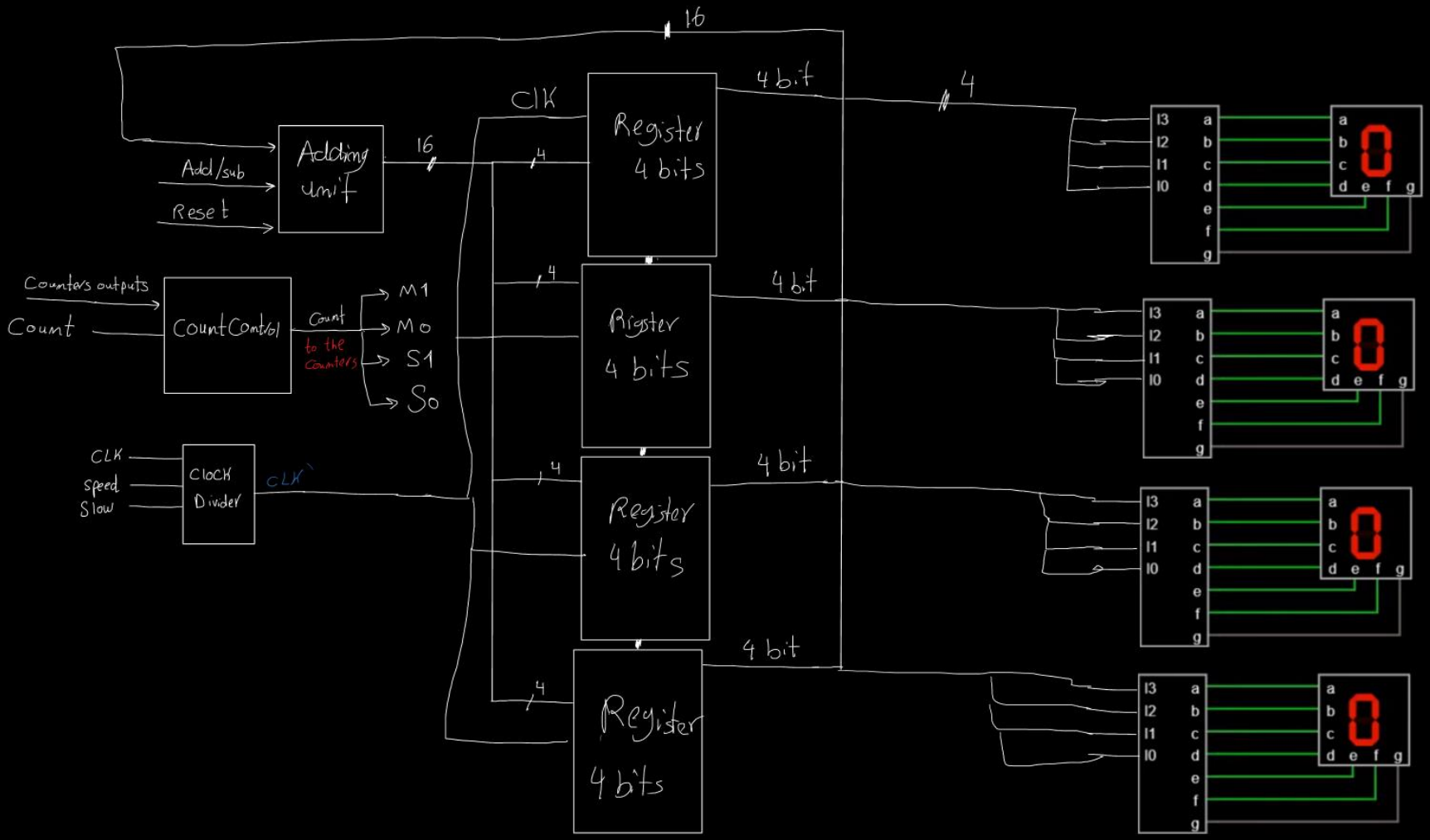
This design is the same as the previous in terms of blocks in the design but the counter we thought of was asynchronous.

We tried to draw it and we have done its simulation on Flastad website



Then all counters will be connected to each other asynchronously.

2.3 Stopwatch with Registers and Adders



In this design we thought of storing the value in a register and then display it and also take the output as a feedback to an adding unit that has some adders and subtractors to count up or down based on the inputs used.

- The clock divider will do the same job as the registers require clock
- The count control here will be little bit tricky as I have to stop the clock to stop the register from loading

Now the next sub section will discuss the design chosen and reasons of that choice.

2.4 Rating Design Approaches

let’s first see the disadvantages of each design:

asynchronous counter design:

The main problem is that the asynchronous counter can have unexpected behavior and it will be complex to control each value because there will be a different logic for each bit.

Register and adder design:

The main problem of this design is the adding unit it will be complicated and why making it add 1 every clock cycle when I have a counter that does the same thing by simple gates between the JK flip flops

Also if we used this logic there will be higher delay in adding ones to the register because the signal goes through more components

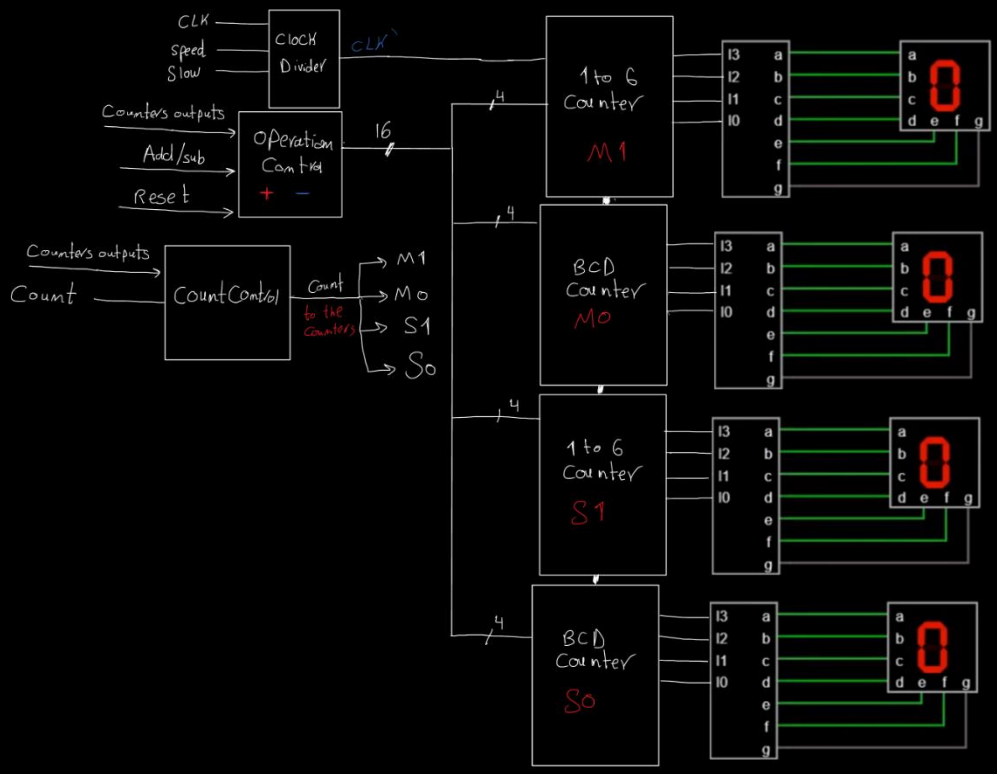
synchronous counter design:

we can conclude that using this method is the best in our opinion because the logic of each bit will be the same unlike the asynchronous counter. Also the synchronous counter will act as a register to load the new values that the user enters through adding two minutes.

3- Chosen Design

Logic & Implementation

as previously seen in the block design of the synchronous counter we need to make the logic and code of the following components:



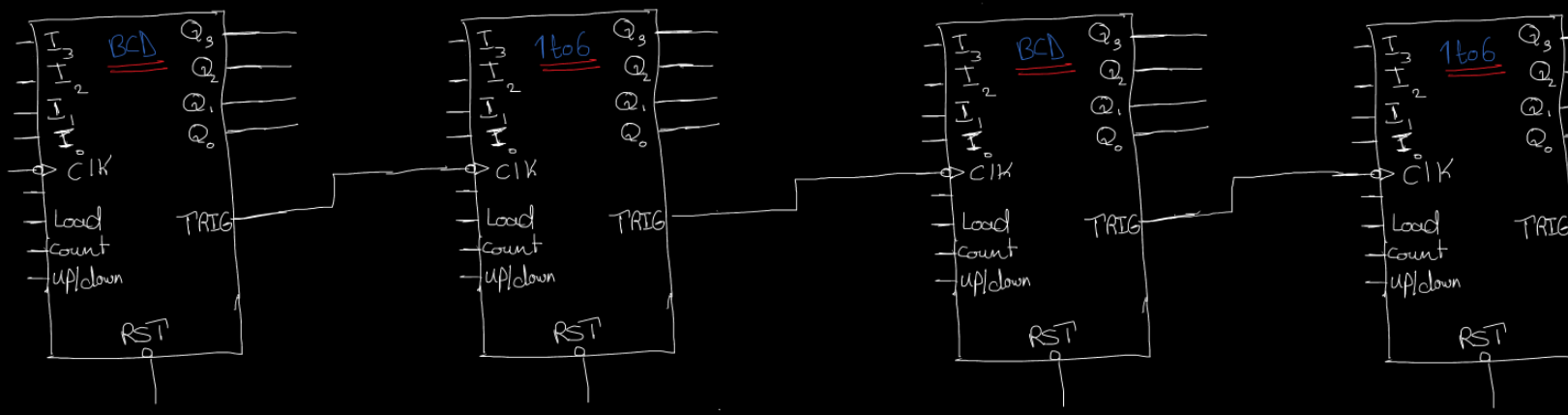
1. Timer (4 counters)
    - 1.2 BCD Up & Down Counter
    - 1.3 0To6 Counter
2. Clock Divider
3. Operation Control Unit
    - 3.1 BCD adder
    - 3.2 BCD subtractor
    - 3.3 BCD Comparator
    - 3.4 Checker
4. Count Control Unit
5. Output Display Unit

3.1 Counters

In this Logic we need 4 counters (two BCD for the first digit of seconds and minutes) (two 0 to 5 counters for the second digit of the seconds and minutes) As the maximum in natural stopwatches is 59:59

We are aware that the maximum count for this project is different. We will consider that in the Count Control Unit.

As Seen the first counter has the clock given and when it reaches its maximum value it motivates the next counter to count.



We need in each counter the following inputs:

- I3, I2, I1, I0 is the inputs through which we can change the value when the user adds or subtracts or resets.
- CLK for the general clock of the flip flops inside the counter
- Load input to stop counting and displays the inputs I3, I2, I1, I0
- Count input to start and stop counting
- Up/Down input to specify the mode of counting up or down

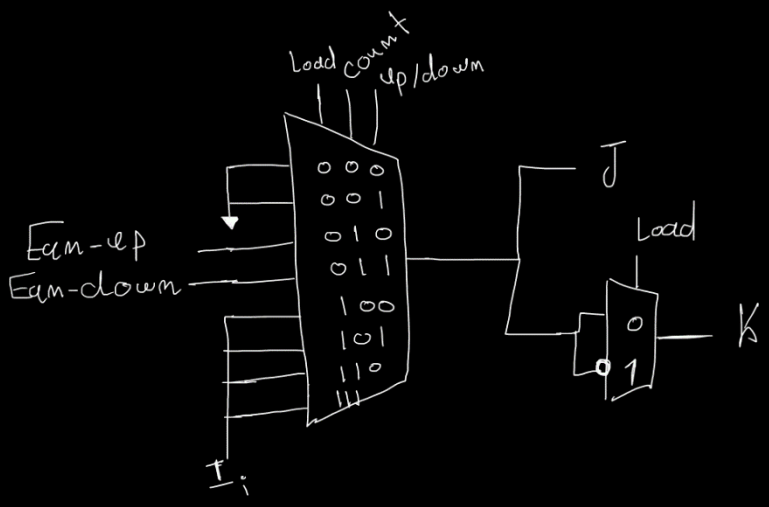
In the next section and the following one we will discuss how we build BCD counter with aforementioned inputs, features.

3.1.1 BCD Up/Down Counter

Here is the function table for the counter

Load	Count	Up/Down	Function	J	K
1	x	x	Load Inputs	Input	Input’
0	0	X	Pause	0	0
0	1	1	Count Up	Eqn Up	Eqn Up
0	1	0	Count Down	Eqn Down	Eqn Down

To implement this function table, we will put MUX8 between every JK flip fop to determine the inputs.



As you can see, we also used mux 2

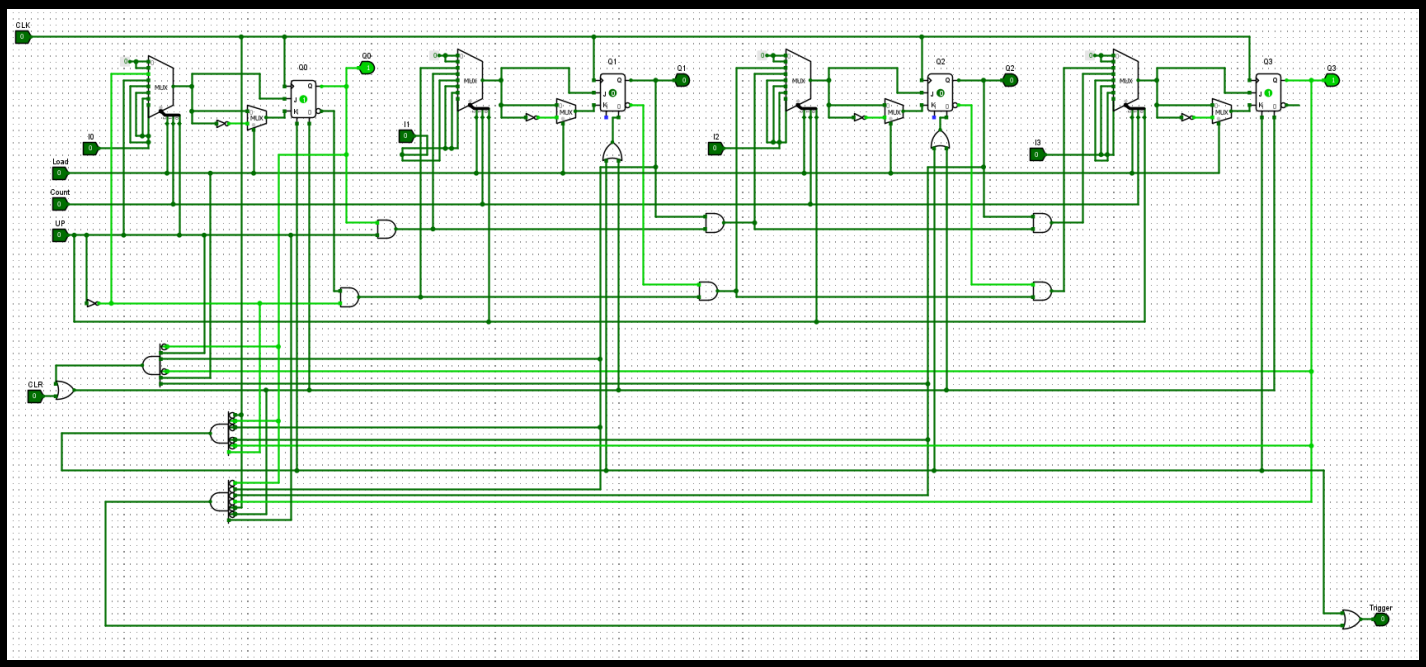
and an inverter to make the k input is the I’ while loading only.

The next step is that we want to make the counter reset its value when it reaches 1010 = 10 when counting up. And to reset its value to 9 when it reaches 1111 (the bits after 0000 when counting downs).

Finally, we need a trigger to activate the next counter and make it count

We will make this trigger go to the clock of the following counter and this trigger will get activated when the counter reset him self in the case of counting up and down.

BCD up/down Counter Schematic



Components:

. JK Flip Flop with asynchronous Reset and Preset

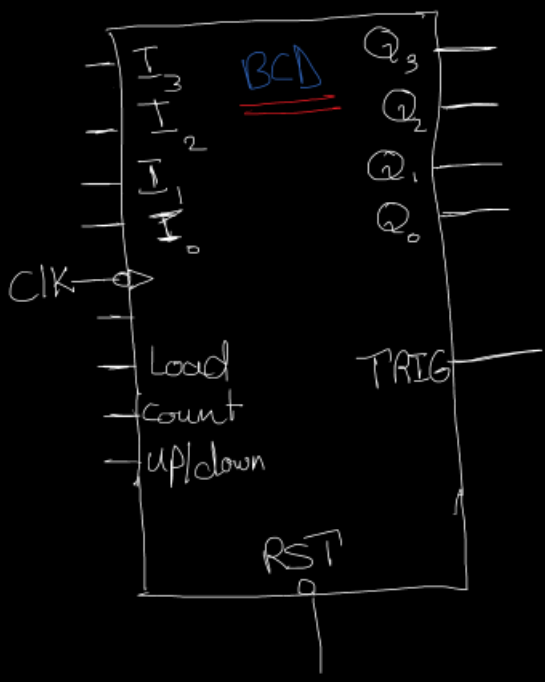
. MUX 8

. MUX 2

. AND Gate

. OR Gate

. Not Gate



Code Implementation

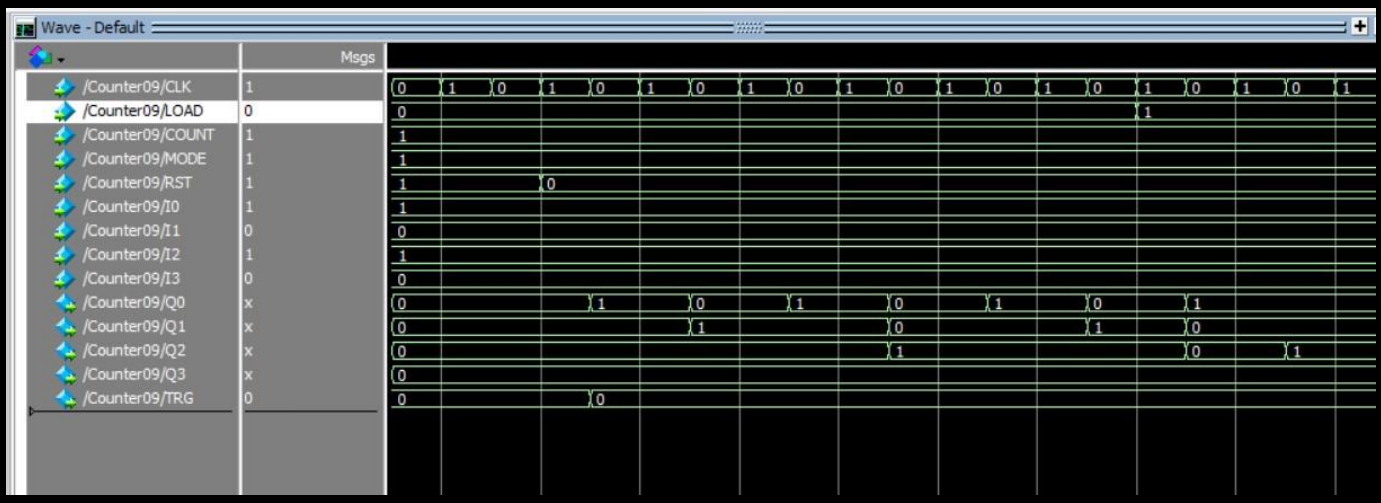
```
module  Counter09 (
    Q0,Q1,Q2,Q3, TRG, CLK, LOAD, COUNT, MODE, RST, I0,I
1,I2,I3
);
    input CLK,LOAD,COUNT,MODE,RST;
    input I0,I1,I2,I3; //Our inputs
    output Q0,Q1,Q2,Q3;
    reg Q0,Q1,Q2,Q3;
    output TRG;  //Our outputs
    wire
 W1,W2,W4,W5,W6,W7,W8,W9,W10,W11,W12,W13,W14,W15,W16,W1
7,W18,W19,W20,W21,W22,W23,W24,W25,W26,W27,W28,W29,W30,
NQ0,NQ1,NQ2,NLOD, W31,W32;
```

```
and(W4,NQ0,W1);
and(W5,Q0,MODE);
and(W6,Q1,W5);
and(W7,Q2,W6);
and(W8,NQ1,W4);
and(W9,NQ2,W8);
and(W10,~Q0,~Q1,~Q2,~Q3,W1);
or(W12,W10,W2);
```

```
Mux81 M8_1(.Y(W14),.I0(0),.I1(0),.I2(W1),.I3(MODE),.I4
(I0),.I5(I0),.I6(I0),.I7(I0),.S0(MODE),.S1(COUNT),.S2
(LOAD));
not(W15,W14);
Mux21 M2_1(.Y(W16),.D0(W14),.D1(W15),.S(LOAD));
Mux81 M8_2(.Y(W17),.I0(0),.I1(0),.I2(W4),.I3(W5),.I4
(I1),.I5(I1),.I6(I1),.I7(I1),.S0(MODE),.S1(COUNT),.S2
(LOAD));
not(W18,W17);
Mux21 M2_2(.Y(W19),.D0(W17),.D1(W18),.S(LOAD));
Mux81 M8_3(.Y(W20),.I0(0),.I1(0),.I2(W8),.I3(W6),.I4
(I2),.I5(I2),.I6(I2),.I7(I2),.S0(MODE),.S1(COUNT),.S2
(LOAD));
not(W21,W20);
Mux21 M2_3(.Y(W22),.D0(W20),.D1(W21),.S(LOAD));
Mux81 M8_4(.Y(W23),.I0(0),.I1(0),.I2(W9),.I3(W7),.I4
(I3),.I5(I3),.I6(I3),.I7(I3),.S0(MODE),.S1(COUNT),.S2
(LOAD));
not(W24,W23);
Mux21 M2_4(.Y(W25),.D0(W23),.D1(W24),.S(LOAD));
```

```
   and(W31,~Q3,~Q2,~Q1,~Q0,MODE,~RST,~CLK);
     SRJKFF SRJKFF1(.Q1(Q0),.Q2(W27),.J(W14),.K(W16),.
CLK(CLK),.RST(W2),.PRST(0));  // J K CLK RST PRST
     SRJKFF SRJKFF2(.Q1(Q1),.Q2(W28),.J(W17),.K(W19),.
CLK(CLK),.RST(W12),.PRST(0));
     SRJKFF SRJKFF3(.Q1(Q2),.Q2(W29),.J(W20),.K(W22),.
CLK(CLK),.RST(W12),.PRST(0));
     SRJKFF SRJKFF4(.Q1(Q3),.Q2(W30),.J(W23),.K(W25),.
CLK(CLK),.RST(W2),.PRST(0)); // issue here

     or(TRG,W10,W31);
endmodule
```

The Simulation



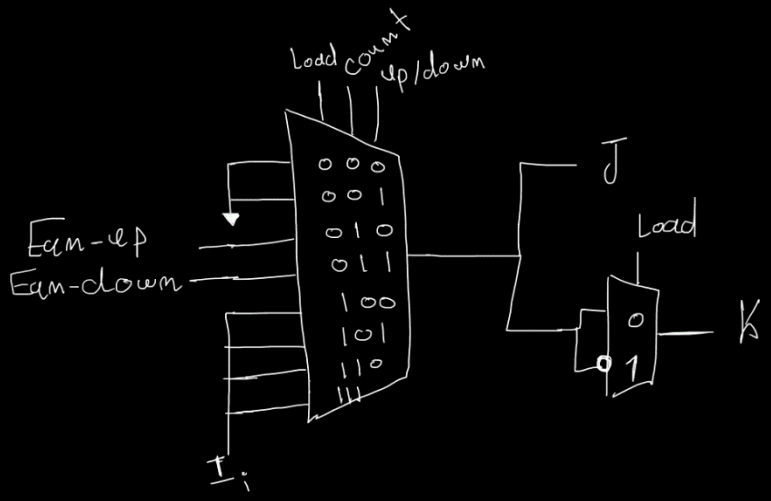
3.1.2 0 to 5 Up & Down Counter

This component is similar to the BCD up and down counter but the reset logic and the trigger logic will be different because you want to reset at 5 which is 1010 in counting up mode and also return to the five again when zero is reached.

Function table is the same as the BCD up and down counter

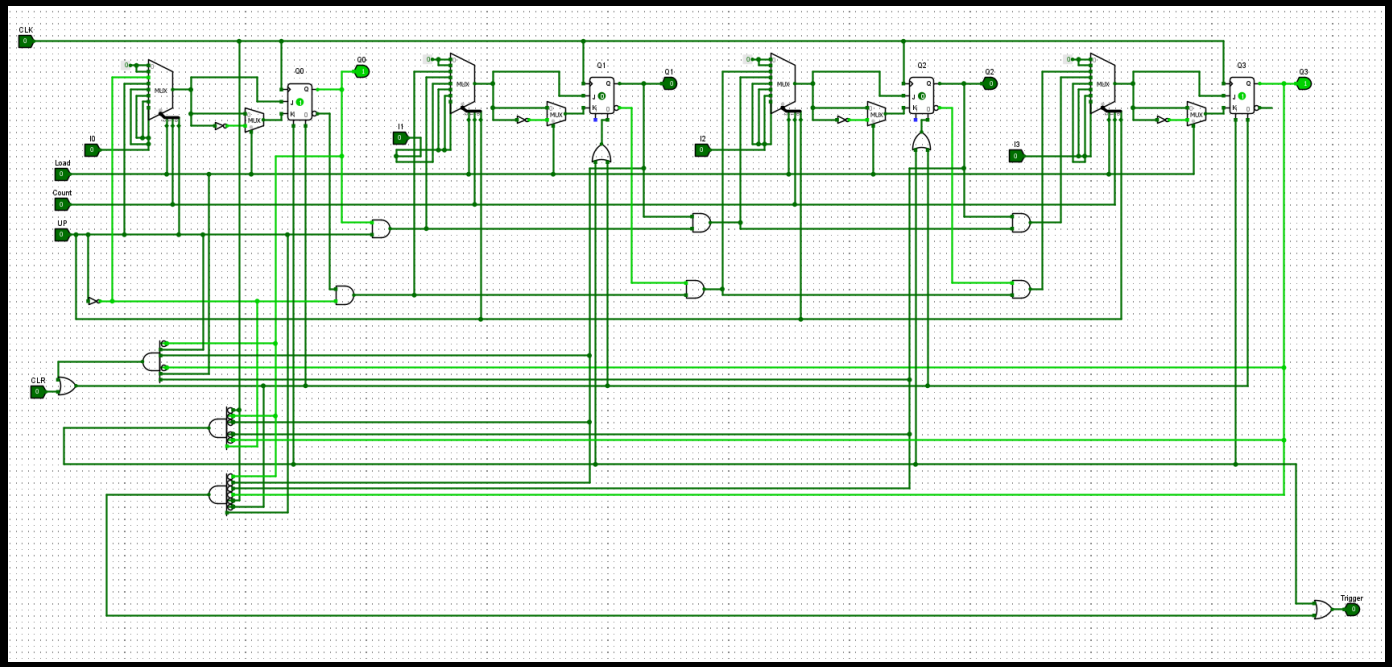
Load	Count	Up/Down	Function	J	K
1	x	x	Load Inputs	Input	Input'
0	0	X	Pause	0	0
0	1	1	Count Up	Eqn Up	Eqn Up
0	1	0	Count Down	Eqn Down	Eqn Down

The same combination of MUX 8 and MUX2 is used to determine the input to the JK flip flops.



The resetting and trigger logic is the same as the BCD counter but when counting up we reset when the counter reaches 6 (0110).

0 to 5 up and down Counter Schematic



Components:

. JK Flip Flop with asynchronous Reset and Preset

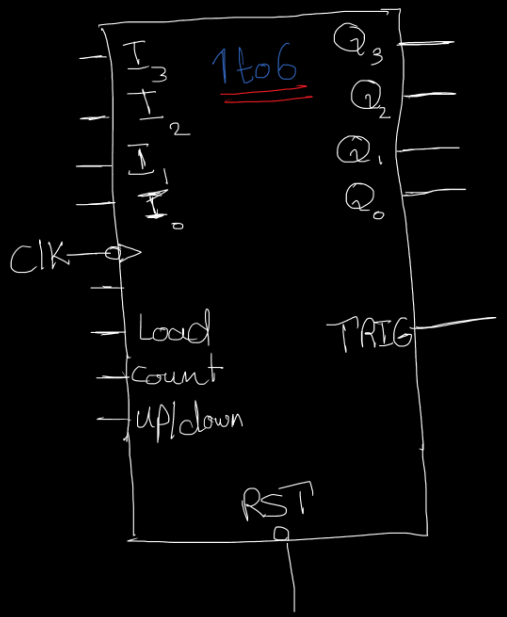
. MUX 8

. MUX 2

. AND Gate

. OR Gate

. Not Gate



Code Implementation

```
module  Counter05 (
    Q0,Q1,Q2,Q3, TRG, CLK, LOAD, COUNT, MODE, RST, I0,I1,I2,I3
);
    input CLK,LOAD,COUNT,MODE,RST;
    input I0,I1,I2,I3; //Our inputs
    output Q0,Q1,Q2,Q3;
    reg Q0,Q1,Q2,Q3;
    output TRG;  //Our outputs
    wire W1,W2,W4,W5,W6,W7,W8,W9,W10,W11,W12,W13,W14,W15
        ,W16,W17,W18,W19,W20,W21,W22,W23,W24,W25,W26,W27,
        W28,W29,W30, NQ0,NQ1,NQ2,NQ3,NLOD, W31,W32;
```

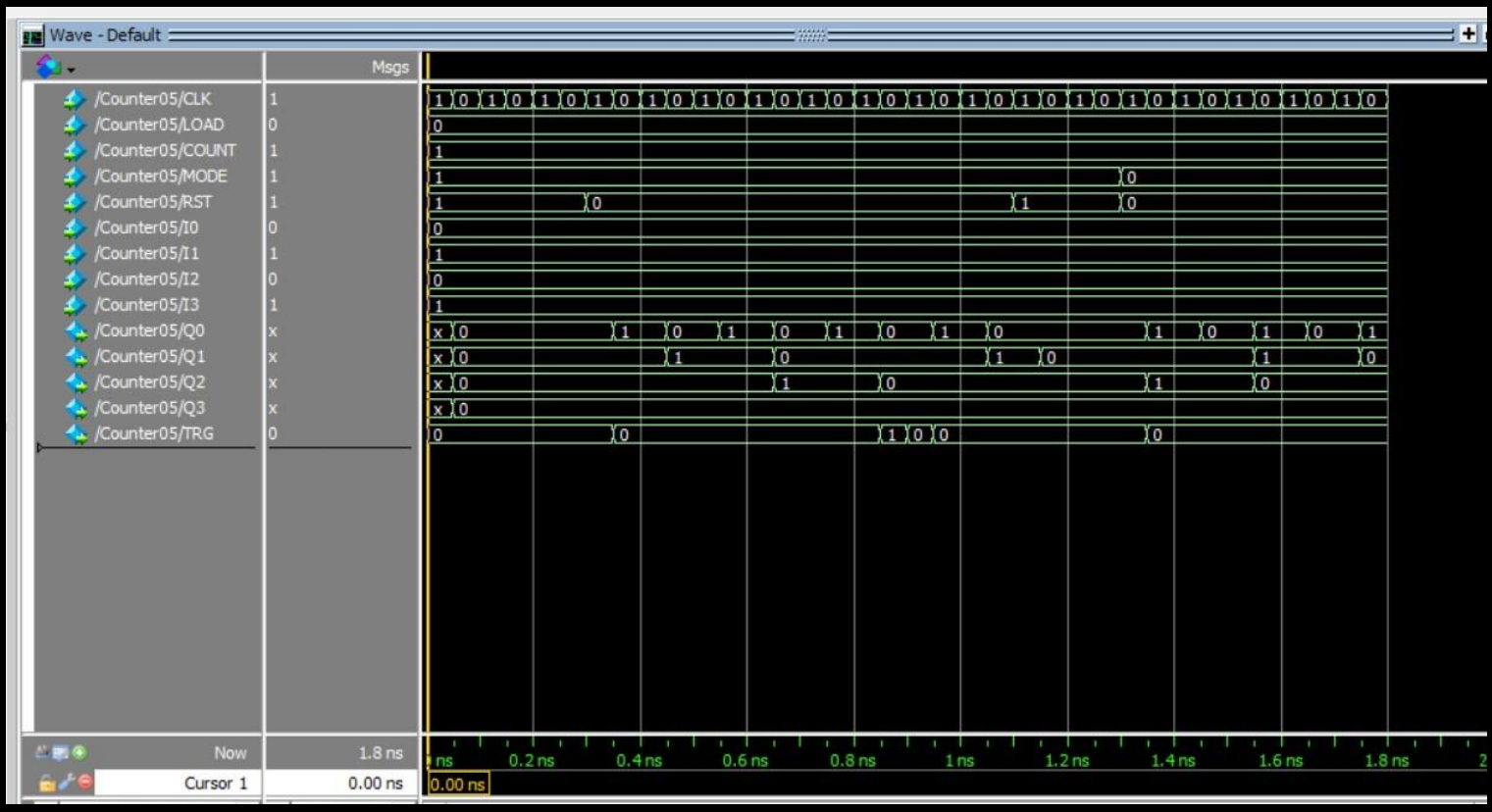
```
not(W1,MODE);
     or(W2,W26,RST);
     not(NQ0,Q0);
     not(NQ1,Q1);
     not(NQ2,Q2);
     not(NQ3,Q3);
     not(NLOD,LOAD);
     and(W26,MODE,Q1,Q2,NQ0,NQ3,NLOD);
```

```
and(W4,NQ0,W1);
and(W5,Q0,MODE);
and(W6,Q1,W5);
and(W7,Q2,W6);
and(W8,NQ1,W4);
and(W9,NQ2,W8);
and(W10,~Q0,~Q1,~Q2,~Q3,W1,~CLK);
or(W12,W10,W2);
```

```
Mux81 M8_1(.Y(W14),.I0(0),.I1(0
),.I2(W1),.I3(MODE),.I4(I0),.I5(I0),.I6(I0),.I7(I0),.S0(MODE),.S1(COUNT),.S2(LOAD));
not(W15,W14);
Mux21 M2_1(.Y(W16),.D0(W14),.D1(W15),.S(LOAD));

Mux81 M8_2(.Y(W17),.I0(0),.I1(0
),.I2(W4),.I3(W5),.I4(I1),.I5(I1),.I6(I1),.I7(I1),.S0(MODE),.S1(COUNT),.S2(LOAD));
not(W18,W17);
Mux21 M2_2(.Y(W19),.D0(W17),.D1(W18),.S(LOAD));
Mux81 M8_3(.Y(W20),.I0(0),.I1(0
),.I2(W8),.I3(W6),.I4(I2),.I5(I2),.I6(I2),.I7(I2),.S0(MODE),.S1(COUNT),.S2(LOAD));
not(W21,W20);
Mux21 M2_3(.Y(W22),.D0(W20),.D1(W21),.S(LOAD));
Mux81 M8_4(.Y(W23),.I0(0),.I1(0
),.I2(W9),.I3(W7),.I4(I3),.I5(I3),.I6(I3),.I7(I3),.S0(MODE),.S1(COUNT),.S2(LOAD));
not(W24,W23);
Mux21 M2_4(.Y(W25),.D0(W23),.D1(W24),.S(LOAD));
```

```
    and(W31,~Q3,~Q2,~Q1,~Q0,MODE,~RST,~CLK);
    //The other one in W10

     SRJKFF SRJKFF1(.Q1(Q0),.Q2(W27),.J(W14),.K(W16),.CLK(CLK),.RST(W2),.PRST(0));
// J K CLK RST PRST
     SRJKFF SRJKFF2(.Q1(Q1),.Q2(W28),.J(W17),.K(W19),.CLK(CLK),.RST(W12),.PRST(0));
     SRJKFF SRJKFF3(.Q1(Q2),.Q2(W29),.J(W20),.K(W22),.CLK(CLK),.RST(W2),.PRST(0));
     SRJKFF SRJKFF4(.Q1(Q3),.Q2(W30),.J(W23),.K(W25),.CLK(CLK),.RST(W12),.PRST(0));
// issue here

     or(TRG,W10,W31);


endmodule
```

The Simulation



3.2 Clock Divider

For speeding up and down we wanted to change the clock that enters the counters so its frequency is divided by 2 or 4. The first clock is the 2x clock (fastest clock) and when it is divided by 2 gives the normal speed and when it is divided by 4 it will give 0.5x speed.

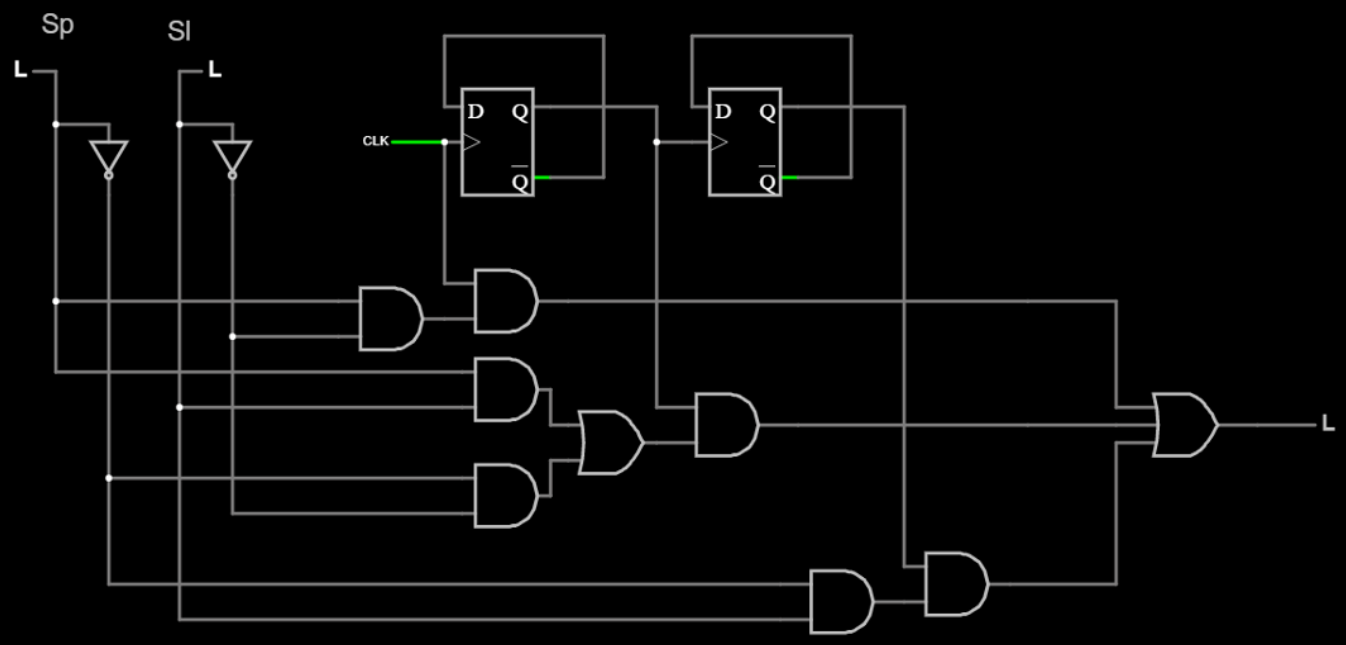
Using D Flip Flops, we can implement this logic

Speed	Slow	CLK(new)
0	0	Q0
1	0	CLK
0	1	Q1
1	1	Q0

$$Q_0 = \overline{S_p}\,\overline{S_L} + S_p S_L$$

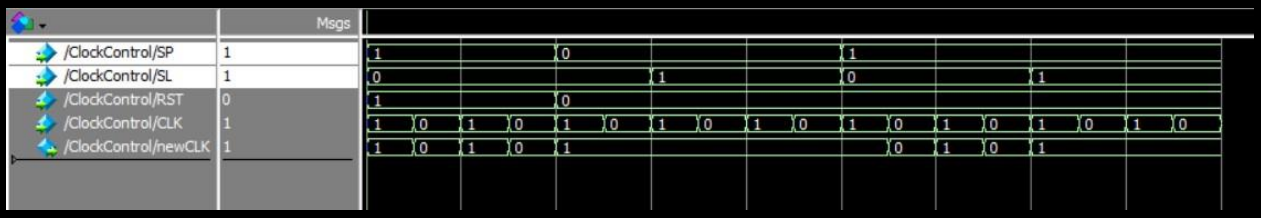
$$CLK = S_p \overline{S_L}$$

$$Q_1 = \overline{S_p} S_L$$



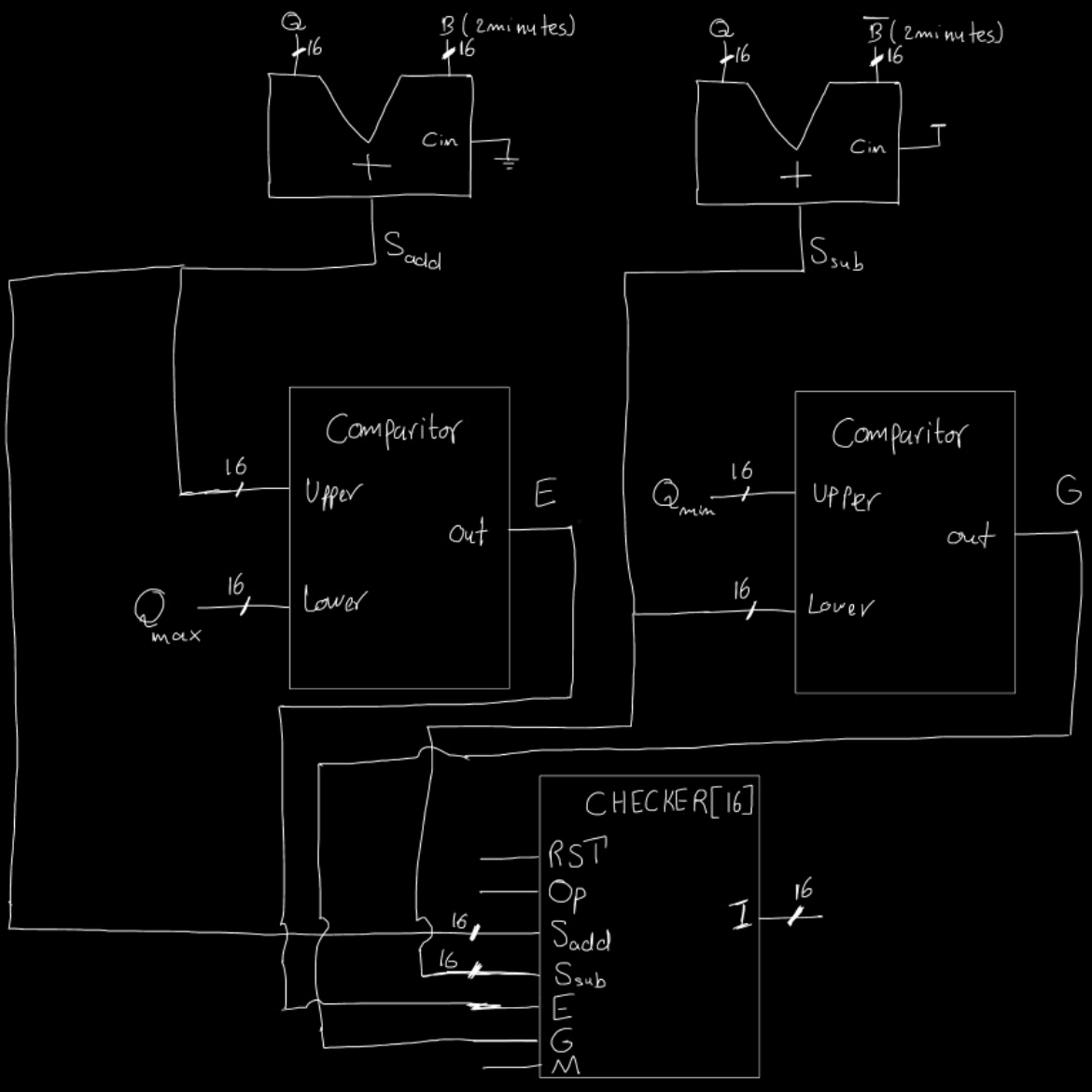
```
module  ClockControl (
    SP, SL, CLK, RST, newCLK // The CLK here should be the fastest one
);
    input SP, SL, RST, CLK; //Our inputs
    output newCLK;  //Our outputs
    wire W1,W2,W3,W4,W5,W6,W7,W8,W9,W10,W11,W12,W13,W14; //Our wires

     // Assigning the values according to our schemitic
     not(W1,SP);
     not(W2,SL);
     and(W3, SP,W2);
     and(W4, SP,SL);
     and(W5, CLK,W3);
     and(W6, W1,W2);
     or(W7, W4,W6);
     and(W8, W1,SL);

    // DFF DFF1 (W11,W13,W13,CLK); //is it ok to put the D in that way??
    // DFF DFF2 (W12,W14,W14,W11);  //??
    DFFR DFF1 (W11,W13,W13,RST,CLK); //is it ok to put the D in that way??
    DFFR DFF2 (W12,W14,W14,RST,W11);  //??

     and(W9, W11,W7);
     and(W10, W12,W8);
     or(newCLK, W5,W9,W10); //removed w10

endmodule
```

The Simulation



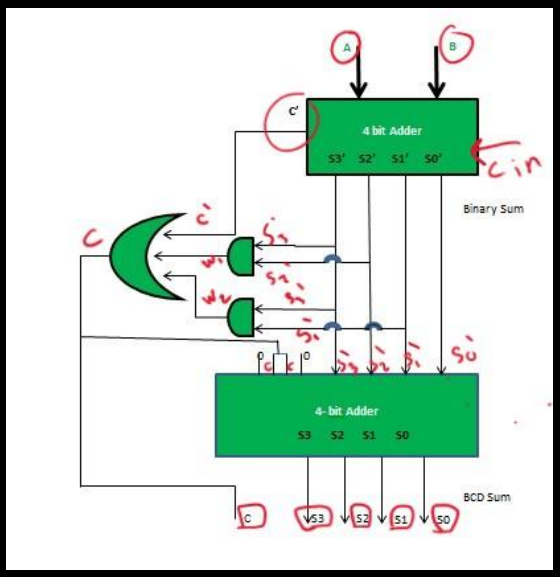
3.3 Operation Control Unit

The Operation Control unit adds, subtracts, and resets the values of Q's. It do not add/subtract directly it guarantees first the new value of Q's will be in range of 10:20 and 49:30. It consists of BCD Adder, BCD Subtractor, BCD comparator and a checker.



3.3.1 BCD Adder

At the beginning we used the traditional Full adder but we faced some problems while adding because we are dealing with digits so we used the BCD adder. The following is the schematic of the 4 bit BCD adder, it consists of two 4bit full adders and some logic gates. Using this 4 bit BCD adder, we implemented a 16 bit BCD adder by taking the carry (C) of the 4 bit BCD Adder and passing it to the next BCD adder as a input carry.

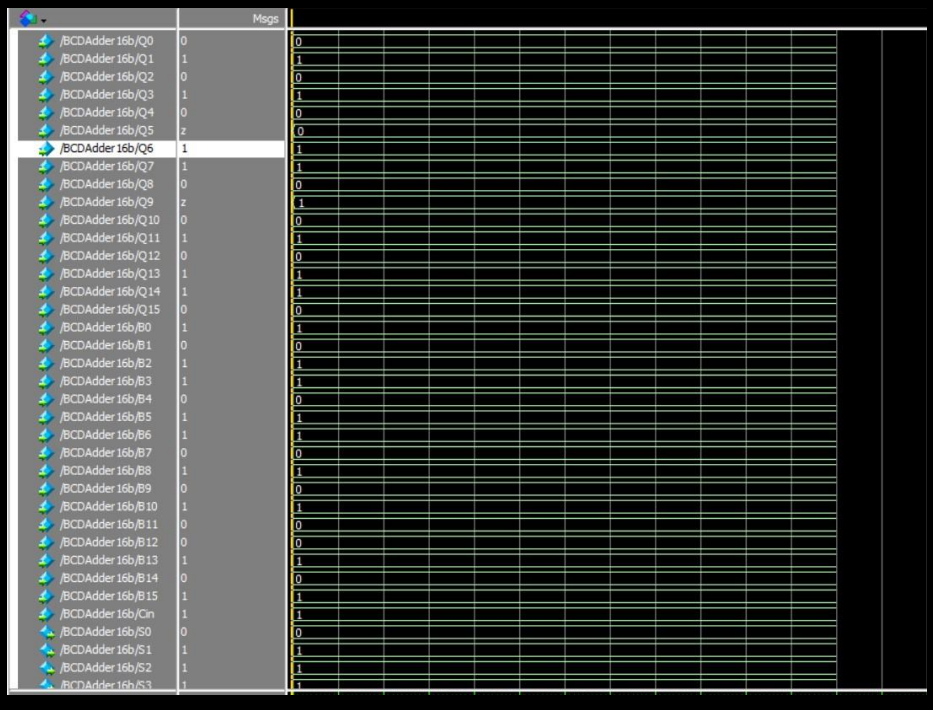


The Code

```
module BCDAdder4b (
    S0,S1,S2,S3,
    Cout,
    Q0,Q1,Q2,Q3,
    B0,B1,B2,B3,
    Cin
);
    input Q0,Q1,Q2,Q3;
    input B0,B1,B2,B3;
    input Cin;
    output S0,S1,S2,S3;
    output Cout;

    wire S00,S11,S22,S33,CCout,W1,W2,Junk;

    FullAdder4b A1(S00,S11,S22,S33,
                   CCout,
                   Q0,Q1,Q2,Q3,
                   B0,B1,B2,B3,
                   Cin);

    and(W1,S33,S22);
    and(W2,S33,S11);
    or(Cout,CCout,W1,W2);

    FullAdder4b A2(S0,S1,S2,S3,
                   Junk,
                   S00,S11,S22,S33,
                   0,Cout,Cout,0,
                   0);

endmodule
```

```
module BCDAdder16b (
    S0,S1,S2,S3,S4,S5,S6,S7,S8,S9,S10,S11,S12,S13,S14,S15,
    Cout,
    Q0,Q1,Q2,Q3,Q4,Q5,Q6,Q7,Q8,Q9,Q10,Q11,Q12,Q13,Q14,Q15,
    B0,B1,B2,B3,B4,B5,B6,B7,B8,B9,B10,B11,B12,B13,B14,B15,
    Cin
);
    input Q0,Q1,Q2,Q3,Q4,Q5,Q6,Q7,Q8,Q9,Q10,Q11,Q12,Q13,Q14,Q15;
    input B0,B1,B2,B3,B4,B5,B6,B7,B8,B9,B10,B11,B12,B13,B14,B15;
    input Cin;
    output S0,S1,S2,S3,S4,S5,S6,S7,S8,S9,S10,S11,S12,S13,S14,S15;
    output Cout;

    wire wCout[2:0];

    BCDAdder4b A0(S0,S1,S2,S3,
                  wCout[0],
                  Q0,Q1,Q2,Q3,
                  B0,B1,B2,B3,
                  Cin
    );

    BCDAdder4b A1(S4,S5,S6,S7,
                  wCout[1],
                  Q4,Q5,Q6,Q7,
                  B4,B5,B6,B7,
                  wCout[0]
    );

    BCDAdder4b A2(S8,S9,S10,S11,
                  wCout[2],
                  Q8,Q9,Q10,Q11,
                  B8,B9,B10,B11,
                  wCout[1]
    );

    BCDAdder4b A3(S12,S13,S14,S15,
                  Cout,
                  Q12,Q13,Q14,Q15,
                  B12,B13,B14,B15,
                  wCout[2]
    );

endmodule
```

The Simulation



3.1.2 BCD Subtractor

We faced the same that we faced in the adder, so we decided to use the BCD Subtractor. The logic behind the BCD Subtractor is first taking the 10's complement of the negative number (which will be subtracted). Then we add it to the first BCD number using the 4 bit BCD Adder. Finally, we check the carry if it is 0 we will take the 10's complement of the answer. We built the 8 BCD Subtractor directly using the same logic. (We did not build 16 bit subtractor because in out case (subtracting 2 minutes) the first 8 bits should be zero so they will have no effect).

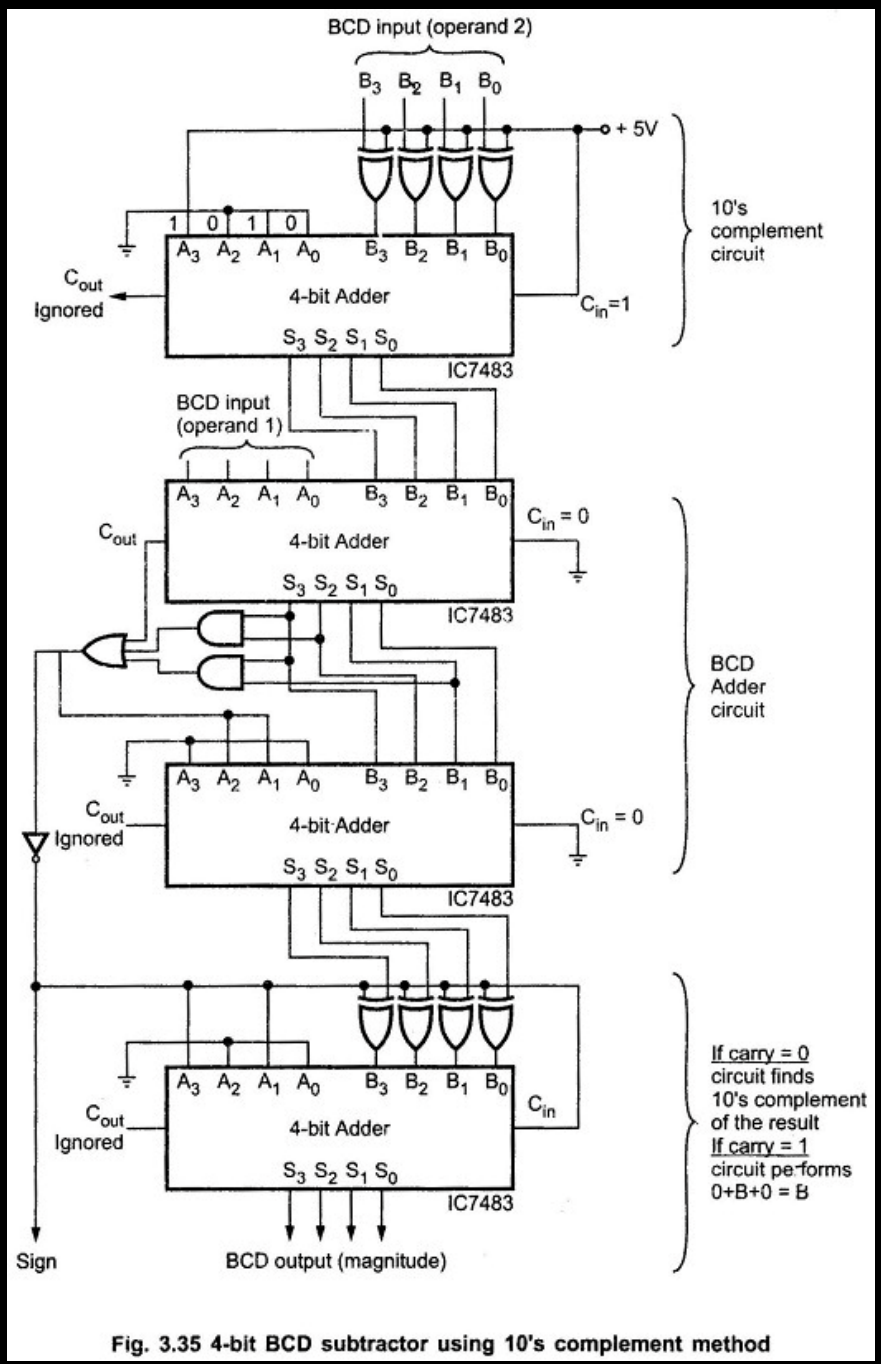


Fig. 3.35 4-bit BCD subtractor using 10's complement method

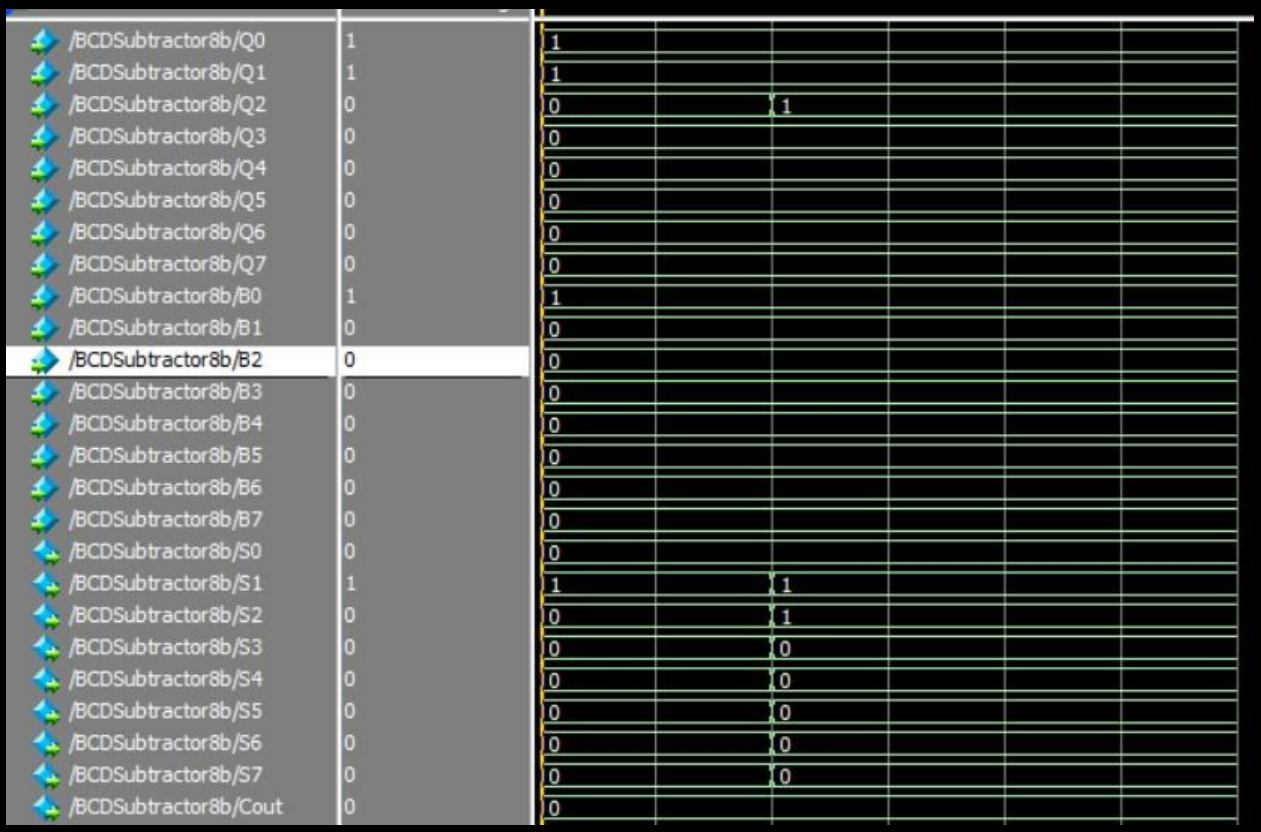
The Code

```
module BCDSubtractor8b (
    S0,S1,S2,S3,S4,S5,S6,S7,
    Cout,
    Q0,Q1,Q2,Q3,Q4,Q5,Q6,Q7,
    B0,B1,B2,B3,B4,B5,B6,B7
);
    input Q0,Q1,Q2,Q3,Q4,Q5,Q6,Q7; //Q - B
    input B0,B1,B2,B3,B4,B5,B6,B7;
    output S0,S1,S2,S3,S4,S5,S6,S7;
    output Cout;

    wire W1,W2,W3,W4,W5,W6,W7,W8,W9,W10,W11,W12,W13,W14,W15,W16,W17,W18,W19,W20,W21,W22,W23,W24,W25,W26,W27,W28,W29,W30,W31,W32,W33,W34,W35,W36,CCout, nCCout, Junk1,Junk2, Junk3, Junk4, Out1,Out2,nOut2,Out3

// getting 10's complement for the first 4 bits
    xor(W1,1,B0);
    xor(W2,1,B1);
    xor(W3,1,B2);
    xor(W4,1,B3);

    FullAdder4b fulladd4b1(
        W5,W6,W7,W8,
        Junk1,
        0,1,0,1,
        W1,W2,W3,W4,
        1
    );

// getting 10's complement for the second 4 bits

    xor(W9,1,B4);
    xor(W10,1,B5);
    xor(W11,1,B6);
    xor(W12,1,B7);

    FullAdder4b fulladd4b2(
        W13,W14,W15,W16,
        Junk2,
        0,1,0,1,
        W9,W10,W11,W12,
        1
    );
```

```
// Adding our input with the 10's complement
BCDAdder4b BCDAdder4b1(
    W17,W18,W19,W20,
    Out1,
    W5,W6,W7,W8,
    Q0,Q1,Q2,Q3,
    0
);

BCDAdder4b BCDAdder4b2(
    W21,W22,W23,W24,
    Out2,
    W13,W14,W15,W16,
    Q4,Q5,Q6,Q7,
    Out1
);

now doing the correction to each 4 bit seperately
not(nOut2,Out2);

xor(W25,W17,nOut2);
xor(W26,W18,nOut2);
xor(W27,W19,nOut2);
xor(W28,W20,nOut2);

xor(W29,W21,nOut2);
xor(W30,W22,nOut2);
xor(W31,W23,nOut2);
xor(W32,W24,nOut2);


FullAdder4b fulladd4b3(
    S0,S1,S2,S3,
    Out3,
    0,nOut2,0,nOut2,
    W25,W26,W27,W28,
    nOut2
);
```

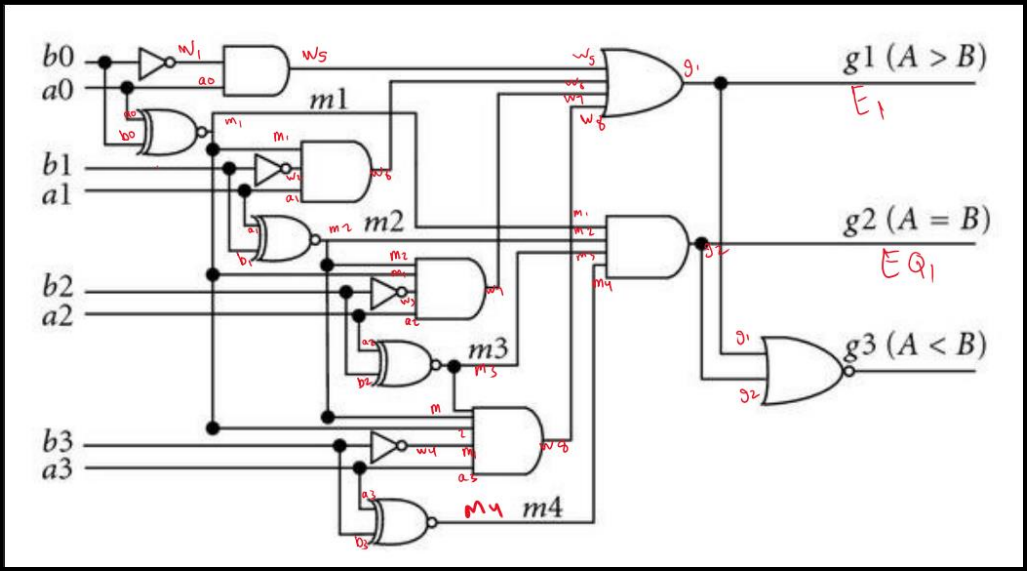
```
    FullAdder4b fulladd4b4(
        W33,W34,W35,W36,
        Cout,
        0,nOut2,0,nOut2,
        W29,W30,W31,W32,
        nOut2
    );

    BCDSubtractor4b BCDSubtractorr4b1
    (
        S4,S5,S6,S7,
        Junk4,
        W33,W34,W35,W36,
        1,0,0,0
    );


endmodule
```

The Simulation

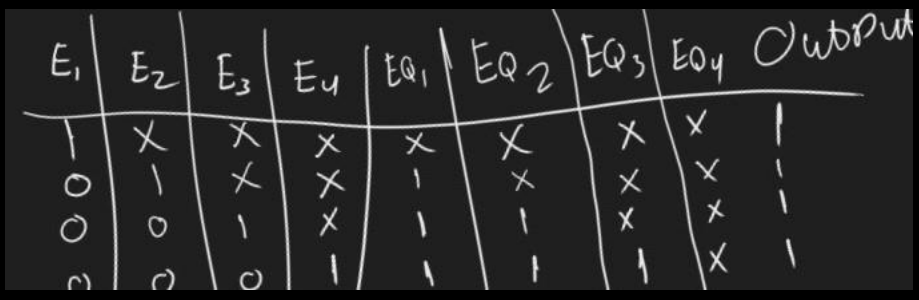


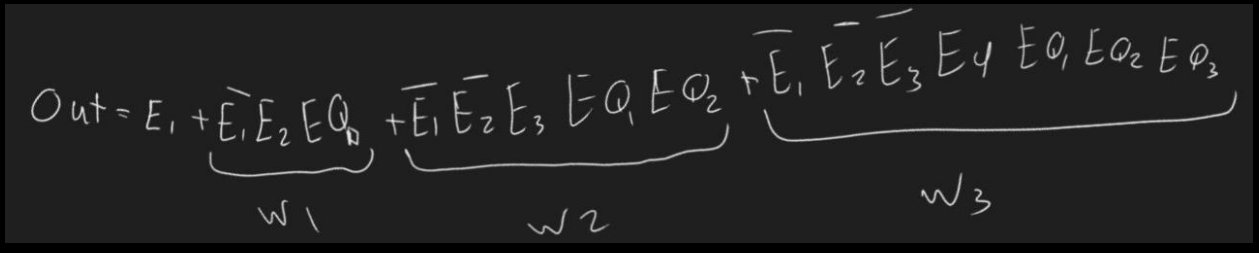
3.1.3 BCD Comparator

First, We wanted to build a 4 bit BCD comparator which compare the Values of 2 BCD inputs and if the First BCD > Second BCD we get E1 = 1, and if they are equal we get EQ1 = 1.



Then we wanted to use this 4 bit BCD comparator to build a 16 bit BCD comparator. The logic behind it was comparing the most significant (most left) digit of the two number, if the most left digit of the first number greater than the most left digit of the second number we get true. Else, we will check the second significant digit but the first digit of the two numbers should be equal and so on.



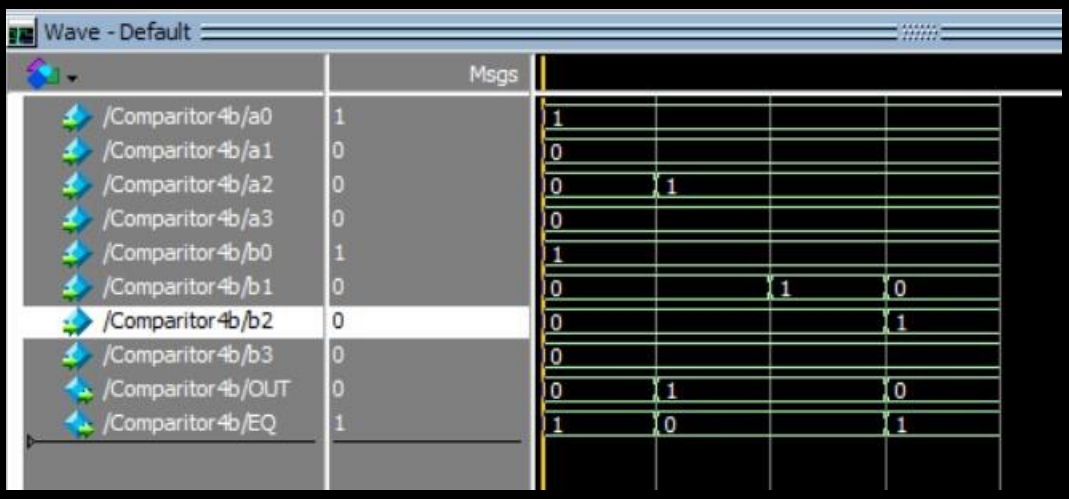


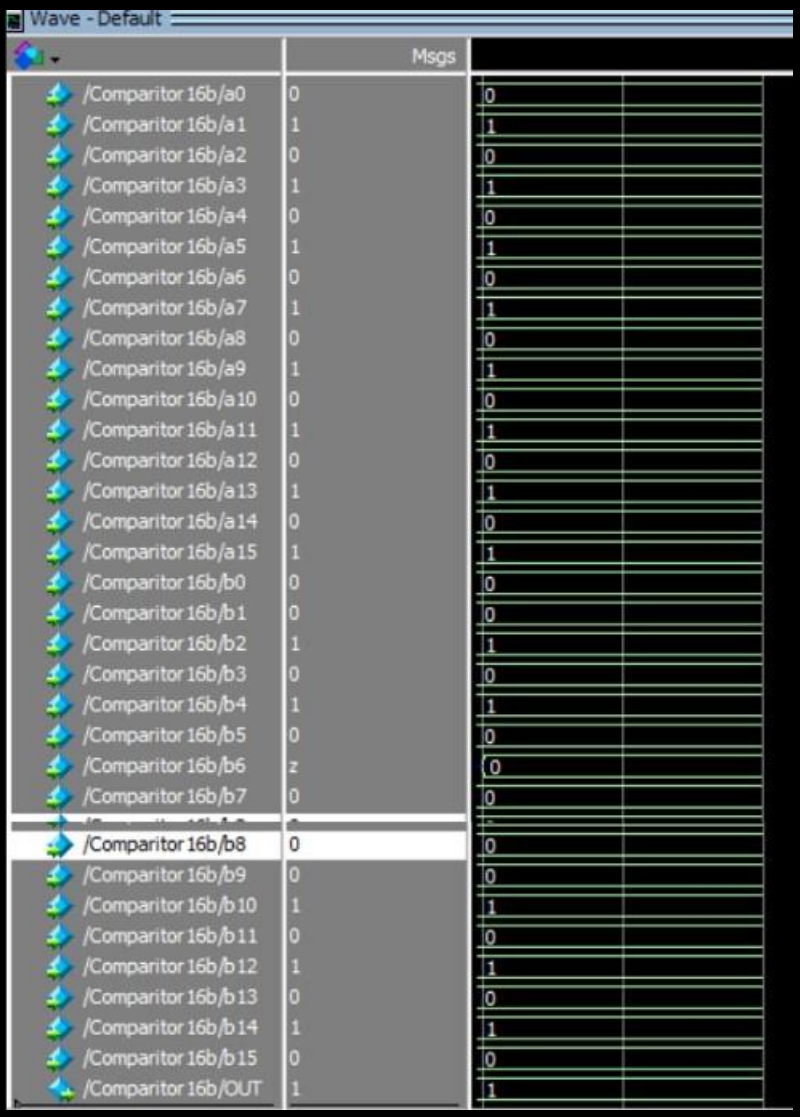
The Code

```
module Comparitor4b (
    OUT, EQ,
    a0,a1,a2,a3,
    b0,b1,b2,b3
); // Trure if A >= B
    input a0,a1,a2,a3;
    input b0,b1,b2,b3;
    output OUT,EQ;
    wire W1,W2,W3,W4,W5,W6,W7,W8,m1,m2,m3,m4,g1,g2,g3;

    not(W1,b3);
    not(W2,b2);
    not(W3,b1);
    not(W4,b0);

    xnor(m1,a3,b3);
    xnor(m2,a2,b2);
    xnor(m3,a1,b1);
    xnor(m4,a0,b0);

    and(W5,W1,a3);
    and(W6,m1,W2,a2);
    and(W7,m2,m1,W3,a1);
    and(W8,m3,m2,m1,W4,a0);

    or(OUT,W5,W6,W7,W8);
    and(EQ,m1,m2,m3,m4);
    nor(g3,g1,g2);

    // or(OUT,g1,g2);


endmodule
```

```
module Comparitor16b (
    OUT,
    a0,a1,a2,a3,a4,a5,a6,a7,a8,a9,a10,a11,a12,a13,a14,a15,
    b0,b1,b2,b3,b4,b5,b6,b7,b8,b9,b10,b11,b12,b13,b14,b15
); // Trure if A >= B
    input a0,a1,a2,a3,a4,a5,a6,a7,a8,a9,a10,a11,a12,a13,a14,a15;
    input b0,b1,b2,b3,b4,b5,b6,b7,b8,b9,b10,b11,b12,b13,b14,b15;
    output OUT;
    wire E1,E2,E3,E4,EQ1,EQ2,EQ3,EQ4,W1,W2,W3;

    Comparitor4b Comparitor4b1 (
        E1,EQ1,
        a12,a13,a14,a15,
        b12,b13,b14,b15
    );

    Comparitor4b Comparitor4b2 (
        E2,EQ2,
        a8,a9,a10,a11,
        b8,b9,b10,b11
    );

    Comparitor4b Comparitor4b3 (
        E3,EQ3,
        a4,a5,a6,a7,
        b4,b5,b6,b7
    );

    Comparitor4b Comparitor4b4 (
        E4,EQ4,
        a0,a1,a2,a3,
        b0,b1,b2,b3
    );


    and(W1,~E1,E2,EQ1);
    and(W2,~E1,~E2,E3,EQ1,EQ2);
    and(W3,~E1,~E2,~E3,E4,EQ1,EQ2,EQ3);

    or(OUT,E1,W1,W2,W3);


endmodule
```

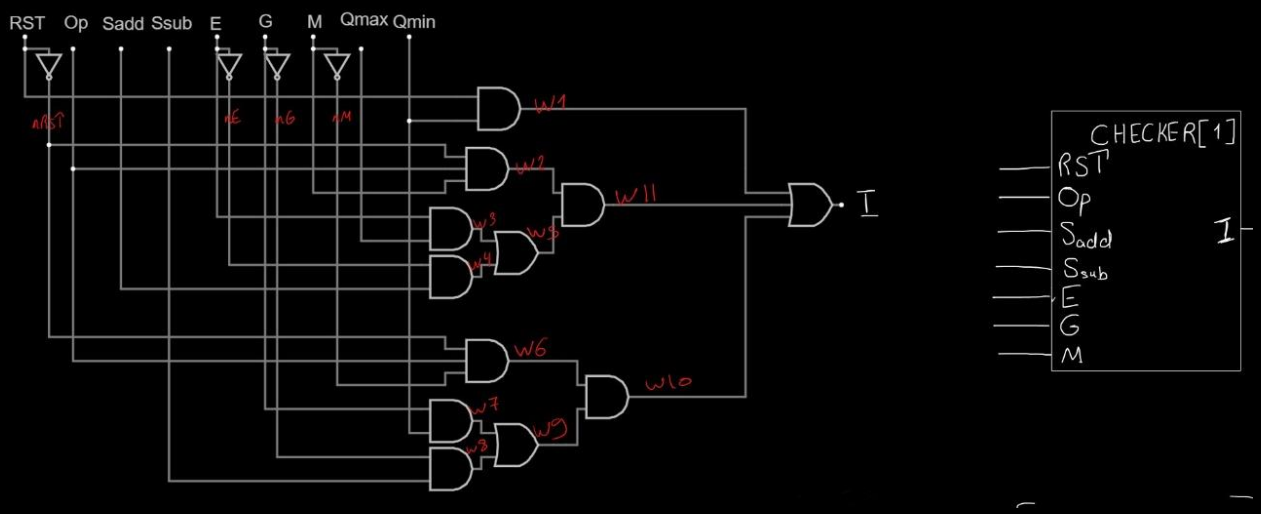
The Simulation





3.1.4 Checker

The goal of the Checker is assigning the values of I's (which will be loaded to the Q's when the user click RST or Add/Subtract buttons). The checker priority is the RST, if it is clicked it assign the 10:20 to the I's and directly it is loaded to the Q's. If the RST button is not clicked but the operation button (add/subtract button) clicked it checks the MODE, if counting up it will add the current Q's with 2 minutes if it exceeds the limit so the Q max will be assigned to the I's if not the addition result will be assigned to the I's. But if the Mode is counting down it will try to subtract the 2 from the current Q's if it is below the minimum the Q min will be assigned to the I's, if it is not below the Q min the result of the subtraction will be assigned to the I's. E should be the output of the comparator which compares the addition result with the Q max. G should be the output of the comparator which compares the addition result with the Q max.

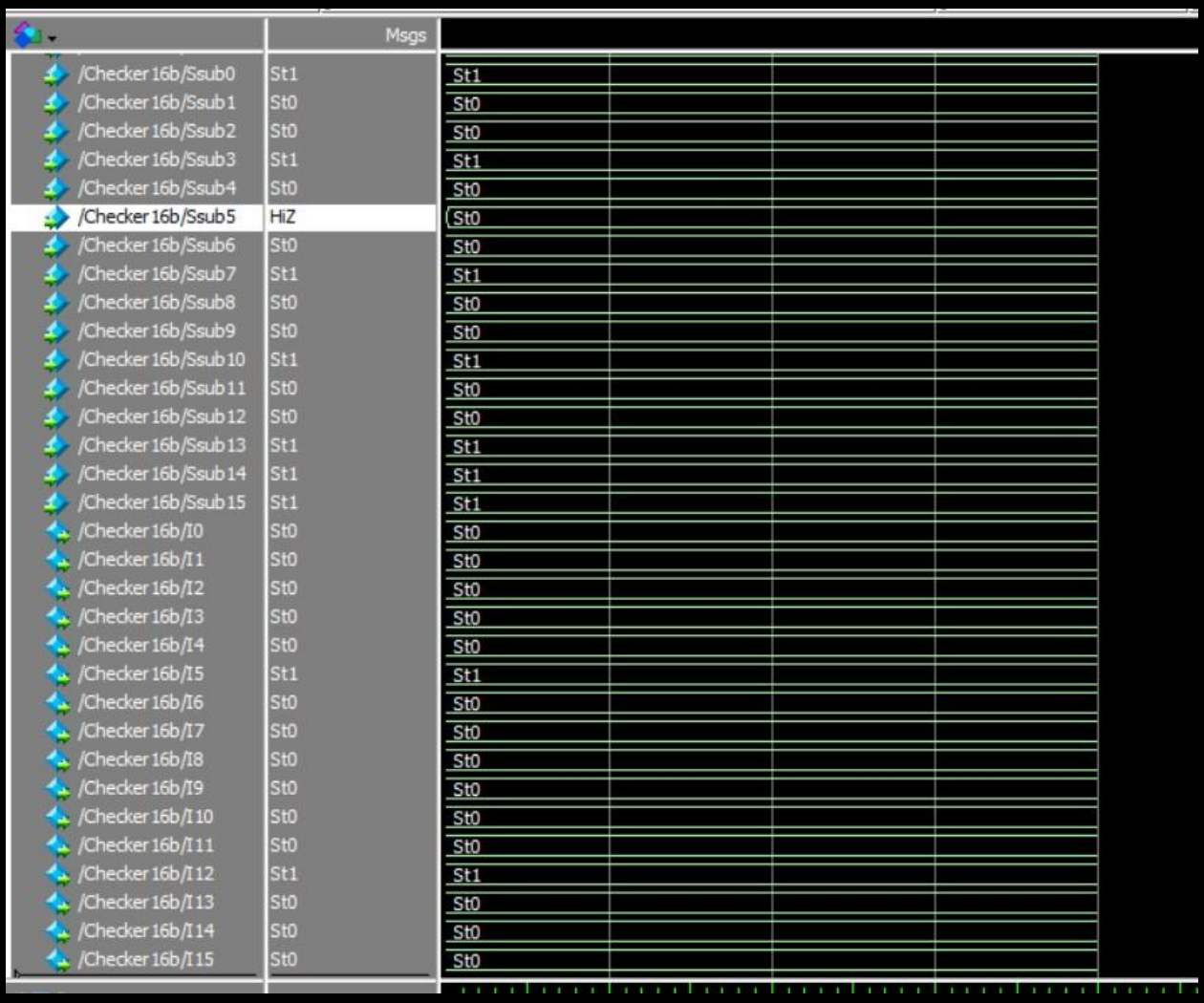


The Code

```
module Checker1b (
    I,RST,Op,Sadd,Ssub,E,G,M,Qmax,Qmin
);
    input RST,Op,Sadd,Ssub,E,G,M,Qmax,Qmin;
    output I;
    wire nRST,nE,nG,nM;
    wire w1,w2,w3,w4,w5,w6,w7,w8,w9,w10,w11;
    not(nRST,RST);
    not(nE,E);
    not(nG,G);
    not(nM,M);

    and(w1,RST,Qmin);
    and(w2,nRST,Op,M);
    and(w3,E,Qmax);
    and(w4,nE,Sadd);
    or(w5,w3,w4);
    and(w6,nRST,Op,nM);
    and(w7,G,Qmin);
    and(w8,nG,Ssub);
    or(w9,w7,w8);
    and(w10,w9,w6);
    and(w11,w2,w5);
    or(I,w1,w10,w11);
endmodule
```

```
module Checker16b (
    I0,I1,I2,I3,I4,I5,I6,I7,I8,I9,I10,I11,I12,I13,I14,I15,
    RST,Op,
    Sadd0,Sadd1,Sadd2,Sadd3,Sadd4,Sadd5,Sadd6,Sadd7,Sadd8,Sadd9,Sadd10,Sadd11,Sadd12,Sadd13,Sadd14,Sadd15,
    Ssub0,Ssub1,Ssub2,Ssub3,Ssub4,Ssub5,Ssub6,Ssub7,Ssub8,Ssub9,Ssub10,Ssub11,Ssub12,Ssub13,Ssub14,Ssub15,
    E,G,M
);
    input  RST,Op,E,G,M; //E is if the upper >= lower in adding // G is if the upper >= lower in subtracting
    input  Sadd0,Sadd1,Sadd2,Sadd3,Sadd4,Sadd5,Sadd6,Sadd7,Sadd8,Sadd9,Sadd10,Sadd11,Sadd12,Sadd13,Sadd14,Sadd15;
    input  Ssub0,Ssub1,Ssub2,Ssub3,Ssub4,Ssub5,Ssub6,Ssub7,Ssub8,Ssub9,Ssub10,Ssub11,Ssub12,Ssub13,Ssub14,Ssub15;
    output I0,I1,I2,I3,I4,I5,I6,I7,I8,I9,I10,I11,I12,I13,I14,I15;

    Checker1b checker1(I0,RST,Op,Sadd0,Ssub0,E,G,M,0,0);
    Checker1b checker2(I1,RST,Op,Sadd1,Ssub1,E,G,M,0,0);
    Checker1b checker3(I2,RST,Op,Sadd2,Ssub2,E,G,M,0,0);
    Checker1b checker4(I3,RST,Op,Sadd3,Ssub3,E,G,M,0,0);
    Checker1b checker5(I4,RST,Op,Sadd4,Ssub4,E,G,M,1,0);
    Checker1b checker6(I5,RST,Op,Sadd5,Ssub5,E,G,M,1,1);
    Checker1b checker7(I6,RST,Op,Sadd6,Ssub6,E,G,M,0,0);
    Checker1b checker8(I7,RST,Op,Sadd7,Ssub7,E,G,M,0,0);
    Checker1b checker9(I8,RST,Op,Sadd8,Ssub8,E,G,M,1,0);
    Checker1b checker10(I9,RST,Op,Sadd9,Ssub9,E,G,M,0,0);
    Checker1b checker11(I10,RST,Op,Sadd10,Ssub10,E,G,M,0,0);
    Checker1b checker12(I11,RST,Op,Sadd11,Ssub11,E,G,M,1,0);
    Checker1b checker13(I12,RST,Op,Sadd12,Ssub12,E,G,M,0,1);
    Checker1b checker14(I13,RST,Op,Sadd13,Ssub13,E,G,M,0,0);
    Checker1b checker15(I14,RST,Op,Sadd14,Ssub14,E,G,M,1,0);
    Checker1b checker16(I15,RST,Op,Sadd15,Ssub15,E,G,M,0,0);
endmodule
```

The Simulation



3.4 Count Control Unit

This Block is responsible for stopping at maximum value when counting up and stopping at minimum value when counting down.

The maximum Value for Timer is 49:30 which can be translated to:

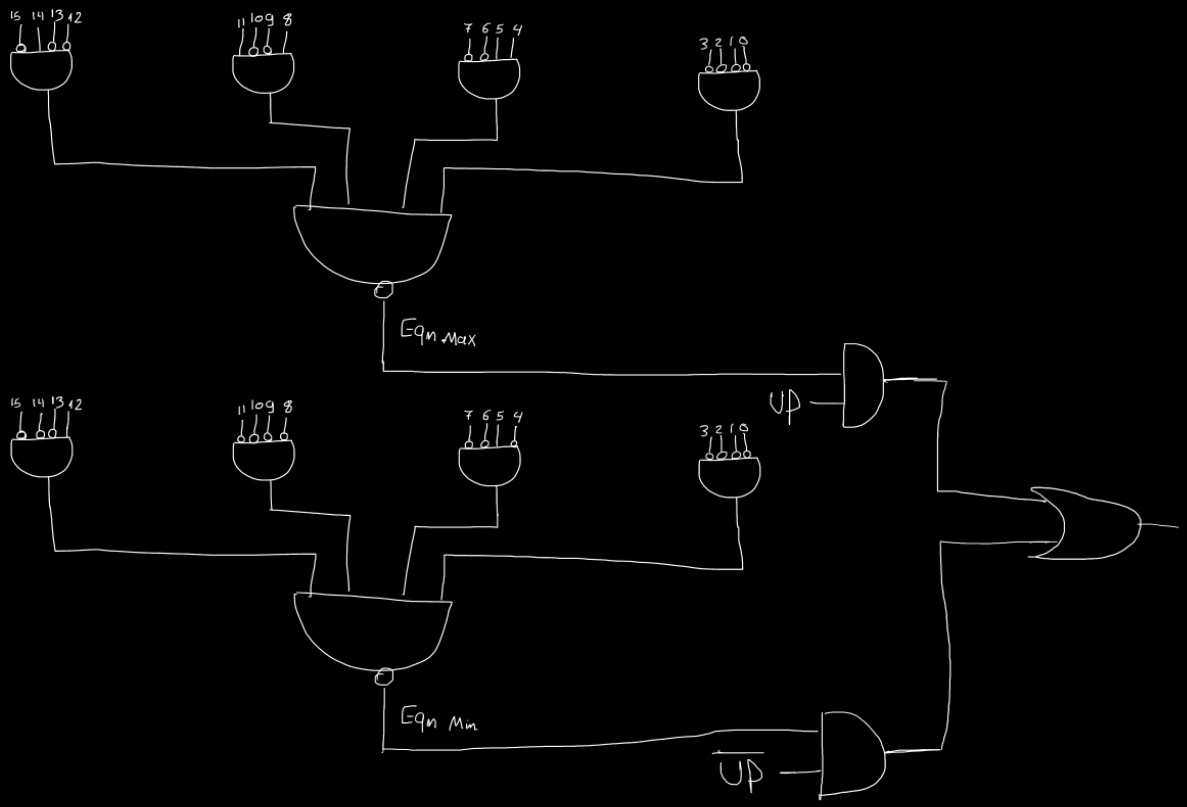
Eqn Max $\overline{Q15}\ Q14\ \overline{Q13}\ \overline{Q12}\ Q11\ \overline{Q10}\ \overline{Q9}\ Q8\ \overline{Q7}\ \overline{Q6}\ Q5\ Q4\ \overline{Q3}\ \overline{Q2}\ \overline{Q1}\ \overline{Q0}$

The minimum Value for Timer is 10:20 which can be translated to:

Eqn Min $\overline{Q15}\ \overline{Q14}\ \overline{Q13}\ Q12\ \overline{Q11}\ \overline{Q10}\ \overline{Q9}\ \overline{Q8}\ \overline{Q7}\ \overline{Q6}\ Q5\ \overline{Q4}\ \overline{Q3}\ \overline{Q2}\ \overline{Q1}\ \overline{Q0}$

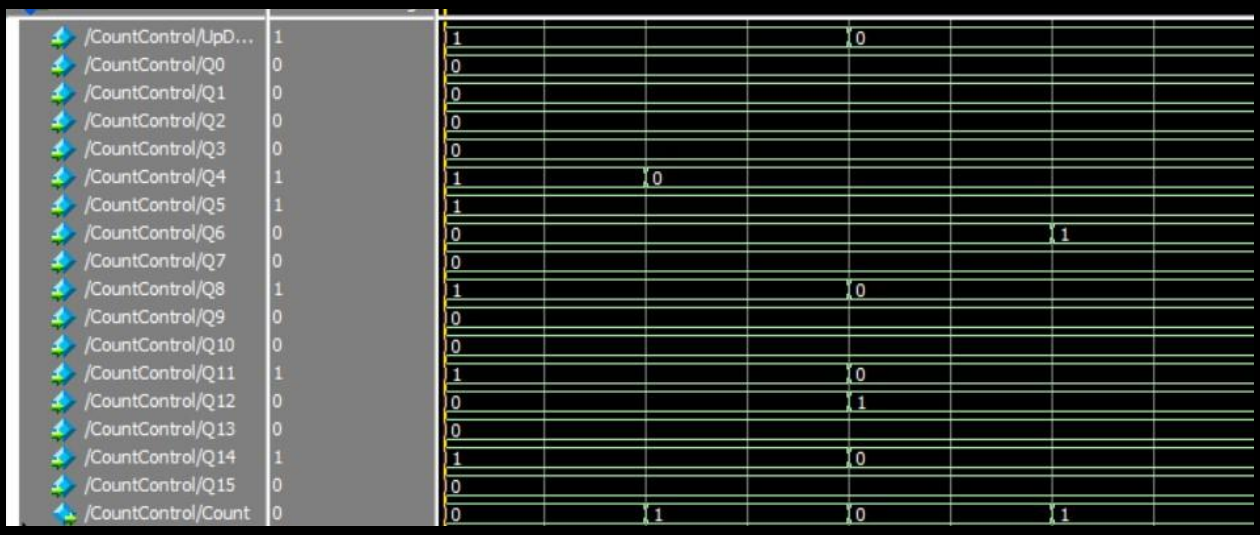
Up	Q15 to Q0	Count
1	Eqn Max	0
0	Eqn Min	0

So the equation will be: Count = Up $\overline{(\text{Eqn Max})}$ + $\overline{\text{Up}}$ $\overline{(\text{Eqn Min})}$



```
module CountControl (
    Count,UpDown,
    Q0,Q1,Q2,Q3,Q4,Q5,Q6,Q7,Q8,Q9,Q10,Q11,Q12,Q13,Q14,Q15
);
    input UpDown;
    input Q0,Q1,Q2,Q3,Q4,Q5,Q6,Q7,Q8,Q9,Q10,Q11,Q12,Q13,Q14,Q15;
    output Count;
    wire EqnMax;
nand(EqnMax,~Q15,Q14,~Q13,~Q12,Q11,~Q10,~Q9,Q8,~Q7,~Q6,Q5,Q4,~Q3,~Q2,~Q1,~Q0);
wire upMode;
and(upMode,EqnMax,UpDown);

nand(EqnMin,~Q15,~Q14,~Q13,Q12,~Q11,~Q10,~Q9,~Q8,~Q7,~Q6,Q5,~Q4,~Q3,~Q2,~Q1,~Q0);
wire DownMode;
and(DownMode,EqnMin,~UpDown);
or(Count,DownMode,upMode);
```

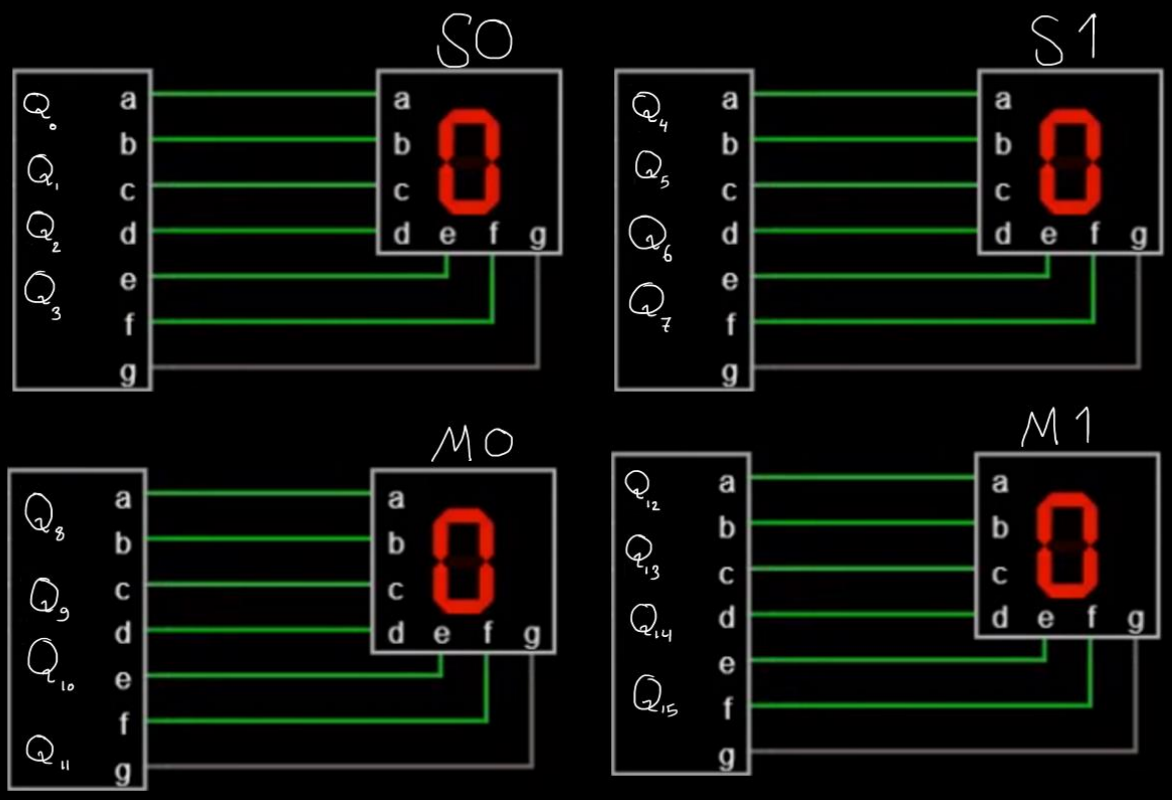
Simulation:



As You can see in the simulation count equals 0 only if the user reached maximum or reached minimum while counting up or down respectively.

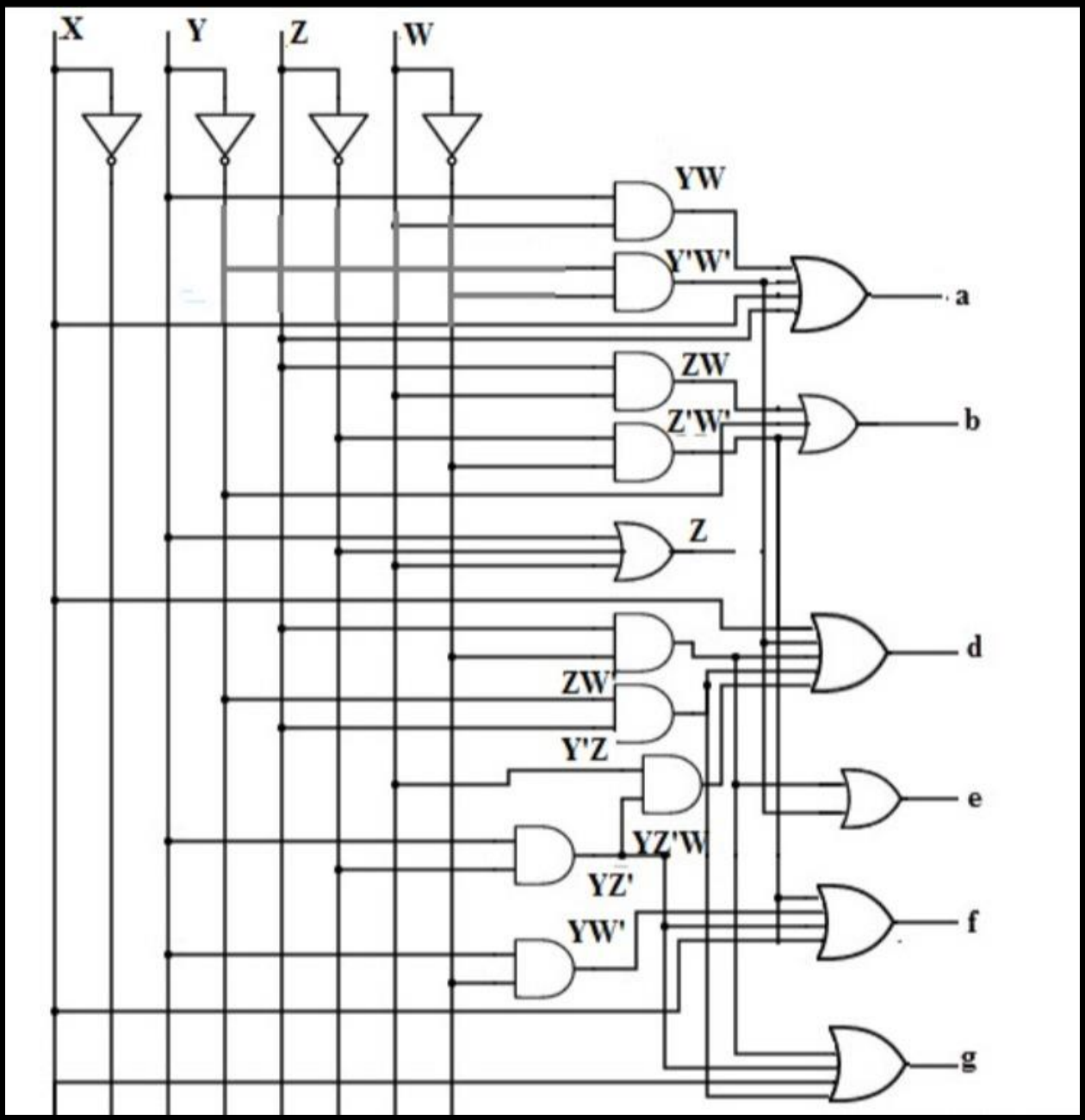
3.5 Output Display Unit

In this unit we just need to take the Q15:Q0 resulting from the counter and enter each four bits in a 7 segment decoder and connect the output to 4 seven segments for M1, M0, S1, S0.



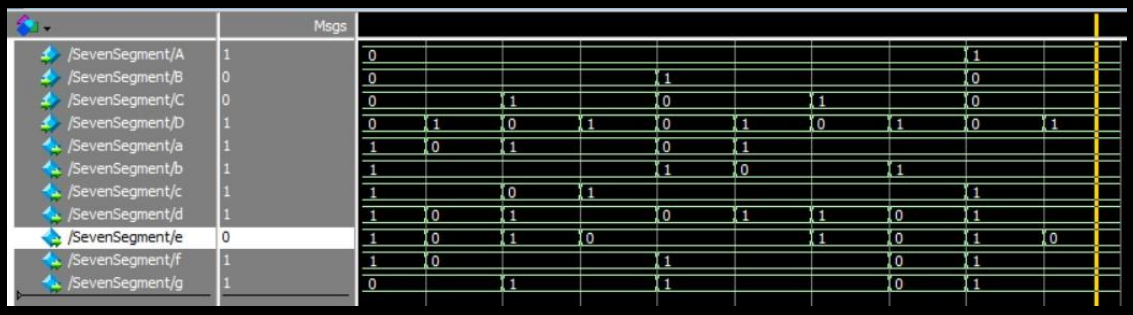
As you can see the 7 Segment Decoder is connected with the 4bit output from each counter of the Timer

Code & Schematic of 7 Segment Decoder



```
module SevenSegment(A, B, C, D, a, b, c, d, e, f, g);
    input A, B, C, D;
    output a, b, c, d, e, f, g;
    wire Bc, Cc, Dc, w1, w2, w22, w3, w4, w5, w6, w7, w8, w9, w10, w11, w12, w13, w14, w15, w16;

    // prepare complimint
    not(Bc, B);
    not(Cc, C);
    not(Dc, D);

    //get a
    and(w1, B, D);
    and(w2, Bc, Dc);
    or(a, A, C, w1, w2);

    //get b
    and(w22, Cc,Dc);//w22 as i name it w2 but i forgot that i used it before
    and(w3, C, D);
    or(b, Bc, w22, w3);

    //get c
    or(c, B, Cc, D);

    //get d
    and(w4, Bc, Dc);
    and(w5, C, Dc);
    and(w6, B, Cc, D);
    and(w7, Bc, C);
    or(d, w4, w5, w6, w7, A);

    //get e
    and(w8, Bc, Dc);
    and(w9, C, Dc);
    or(e, w8, w9);

    //get f
    and(w10, Cc, Dc);
    and(w11, B, Cc);
    and(w12, B, Dc);
    or(f, A, w10, w11, w12);

    //get g
    and(w13, Bc, C);
    and(w14, C, Dc);
    and(w15, B, Cc);
    and(w16, B, Cc);
    or(g, w13, w14, w15, w16, A);

endmodule
```

The Simulation



The truth table

7 Segment Decoder

A	B	C	D	a	b	c	d	e	f	g
0	0	0	0	1	1	1	1	1	1	0
0	0	0	1	0	1	1	0	0	0	0
0	0	1	0	1	1	0	1	1	0	1
0	0	1	1	1	1	1	1	0	0	1
0	1	0	0	0	1	1	0	0	1	1
0	1	0	1	1	0	1	1	0	1	1
0	1	1	0	1	0	1	1	1	1	1
0	1	1	1	1	1	1	0	0	0	0
1	0	0	0	1	1	1	1	1	1	1
1	0	0	1	1	1	1	1	0	1	1

4. Used Elements

In this Chapter we will introduce the sub components that we used and include the truth table, Schematic and the Structural code in system Verilog

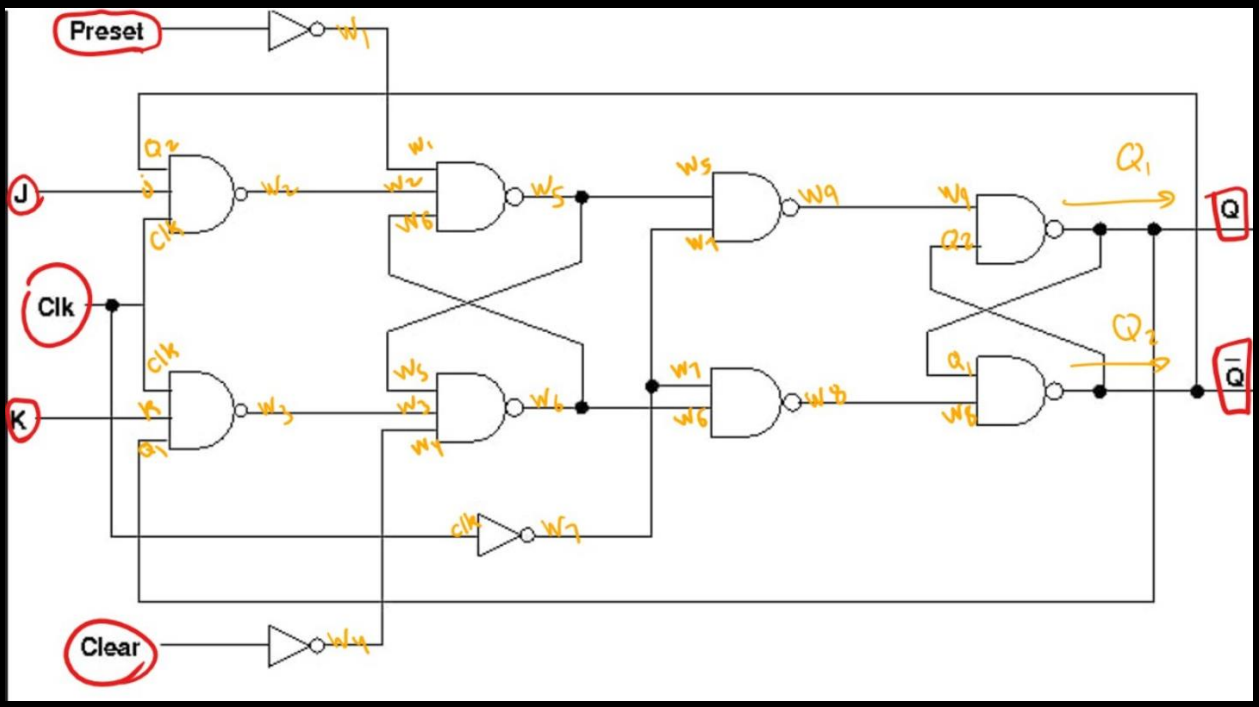
These are The components that we used:

4.1 JK Flip Flop with asynchronous Reset and Preset

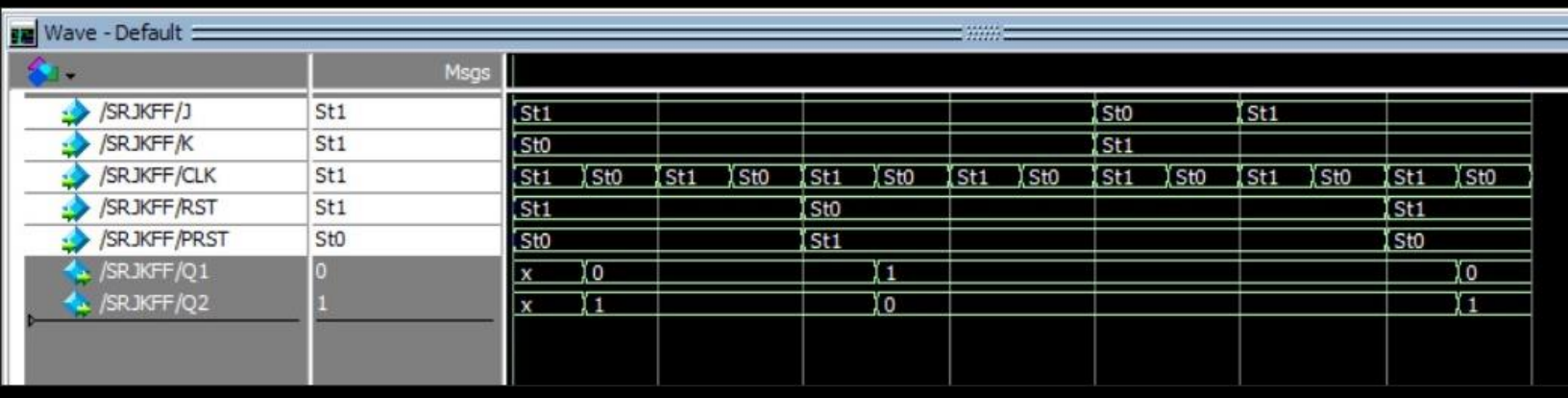
J-K

Preset	Clear	CLK	J	K	Q	Q'
X	1	X	X	X	0	1
1	X	X	X	X	1	0
0	0	posedge	1	0	1	0
0	0	posedge	0	1	0	1
0	0	posedge	1	1	1	0
0	0	posedge	0	0	1	0

Schematic (Note the wires written are the wires used in coding)

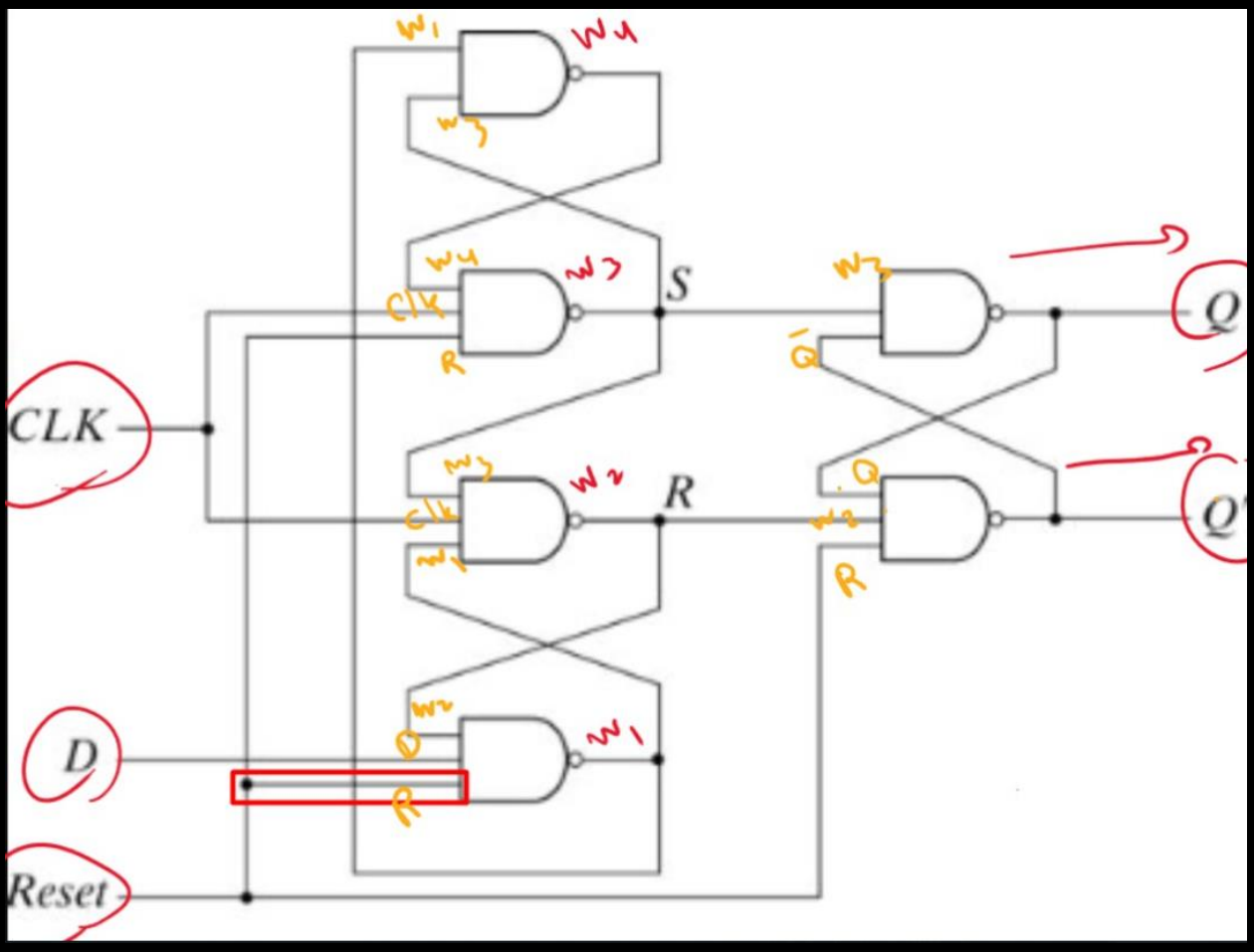


```
module  SRJKFF (
    Q1, Q2, J,K, CLK, RST,PRST
);
    input J,K,CLK, RST, PRST; //Our inputs
    output Q1,Q2;  //Our outputs
    wire W1,W2,W3,W4,W5,W6,W7,W8,W9; //Our wires

     // Assigning the values according to our schemitic
     not(W1,PRST);
     not(W4,RST);
     not(W7,CLK);

     nand(W2,J,Q2,CLK);
     nand(W3,Q1,CLK,K);
     nand(W5,W1,W2,W6);
     nand(W6,W5,W3,W4);

     nand(W8,W7,W6);
     nand(W9,W5,W7);
     nand(Q1,W9,Q2);
     nand(Q2,Q1,W8);

endmodule
```



4.2 D Flip Flop

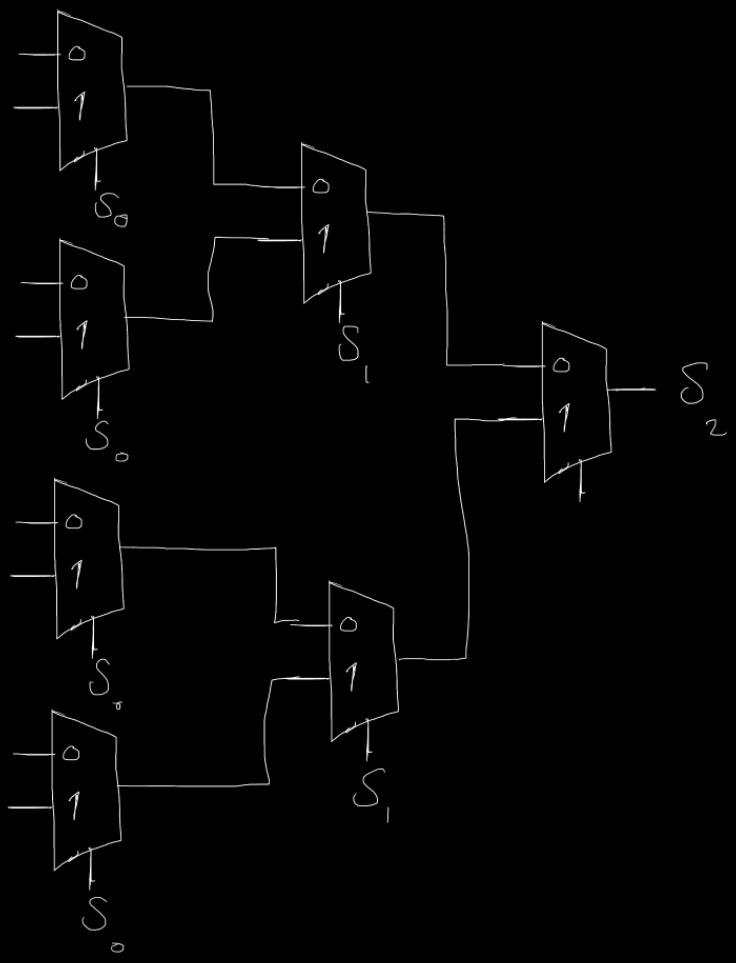
CLK	D	Reset	Q	Q'
posedge	0	0	0	1
posedge	0	1	0	1
posedge	1	0	1	0
posedge	1	1	0	1

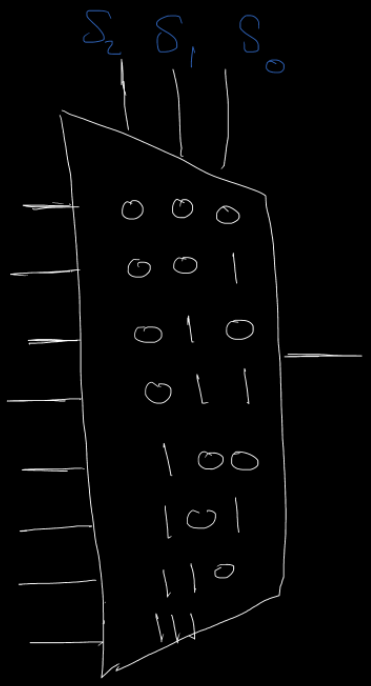


```
module  DFFR (
    Q1, Q2, D, RST , CLK
);
    input D, RST, CLK; //Our inputs
    output Q1,Q2;  //Our outputs
    wire W1,W2,W3,W4; //Our wires

     // Assigning the values according to our schemitic
    //  not(W5,RST);
     nand(W1,W2,D,RST);
     nand(W2,W1,CLK,W3);
     nand(W3,W4,CLK,RST);
     nand(W4,W1,W3);
     nand(Q1,W3,Q2);
     nand(Q2,Q1,W2,RST);

endmodule
```

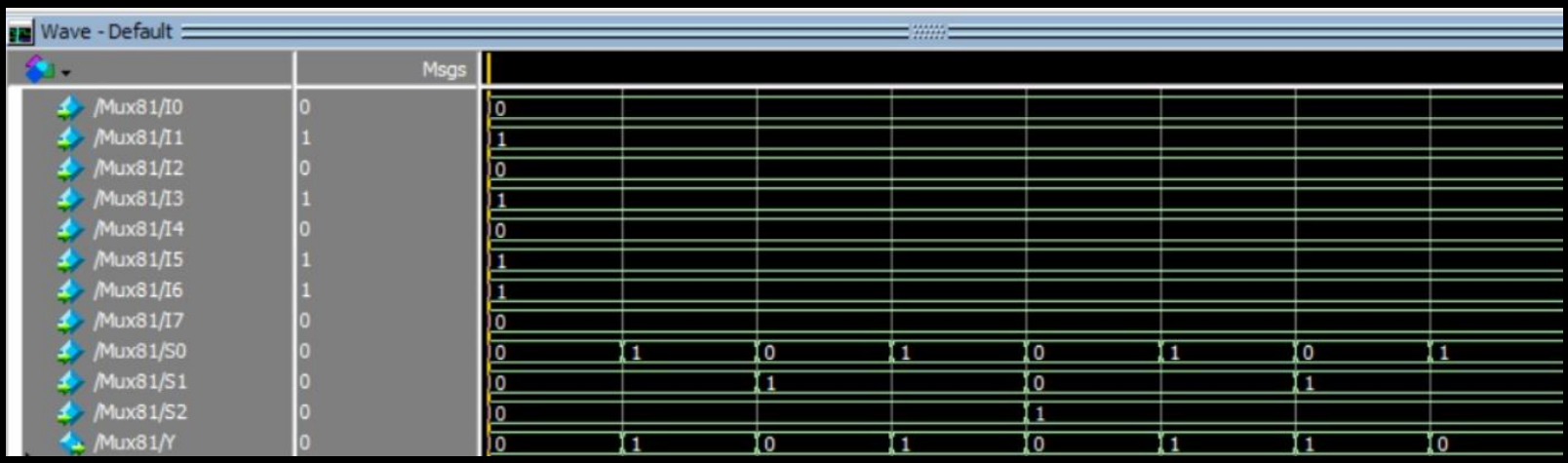
4.3 MUX

S2	S1	S0	Y
0	0	0	A0
0	0	1	A1
0	1	0	A2
0	1	1	A3
1	0	0	A4
1	0	1	A5
1	1	0	A6
1	1	1	A7



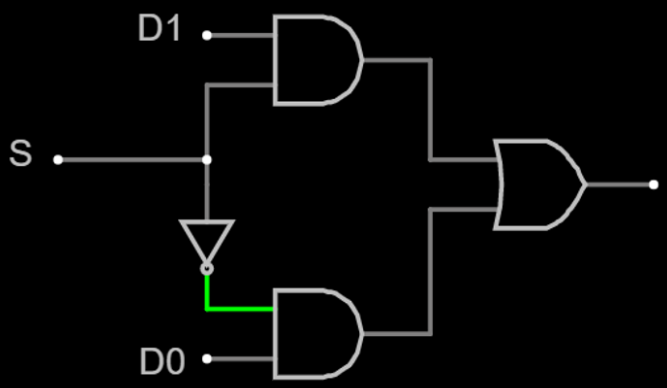


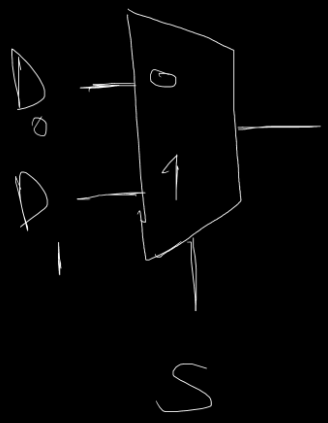
```
module  Mux81 (
    Y,I,S
);
    input I[7:0];
    input S[2:0];
    output Y;
// first stage
    wire w1[3:0];
    Mux21 M1(w1[0],I[1:0],S[0]);
    Mux21 M2(w1[1],I[3:2],S[0]);
    Mux21 M3(w1[2],I[5:4],S[0]);
    Mux21 M4(w1[3],I[7:6],S[0]);
//second stage
    wire w2[1:0];
    Mux21 M5(w2[0],w1[1:0],S[1]);
    Mux21 M6(w2[1],w1[3:2],S[1]);
//third stage
    Mux21 M7(Y,w2[1:0],S[2]);
endmodule
```

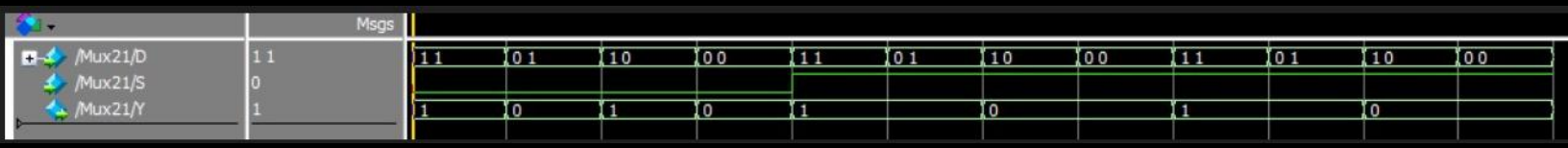


4.4 Mux 2

S1	Y
0	D0
1	D1



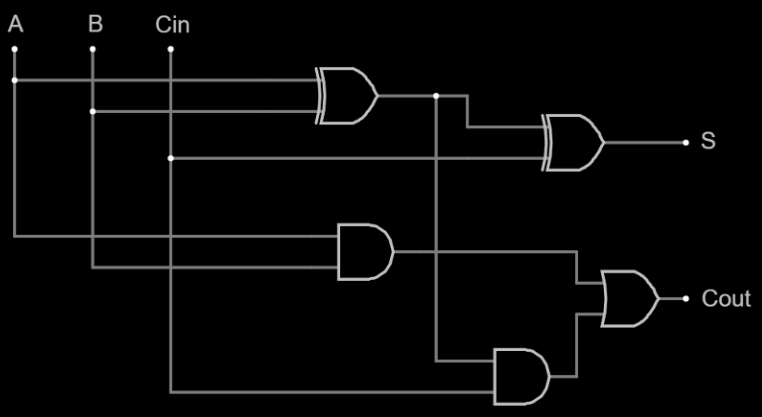




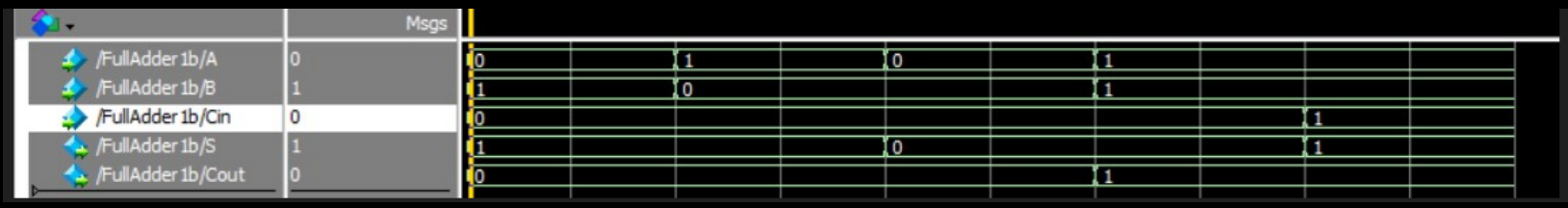

```
module  Mux21 (
    Y,D,S
);
    input D[0:1];
    input S;
    output Y;
    wire Sn;
    not(Sn,S);
     wire X1,X2;
    and(X1,Sn,D[0]);
    and(X2,S,D[1]);
    or(Y,X1,X2);
endmodule
```

4.5 Full Adder 1bit

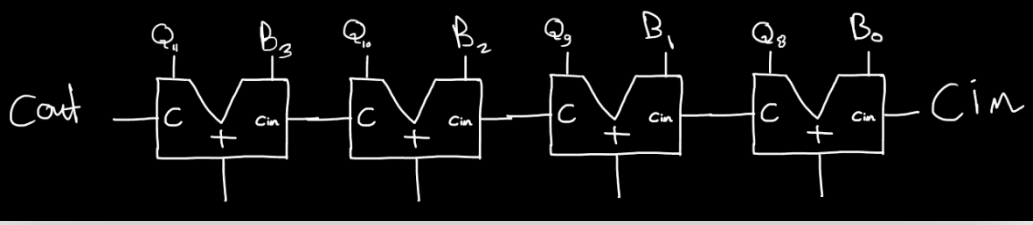
A	B	cin	sum	cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1



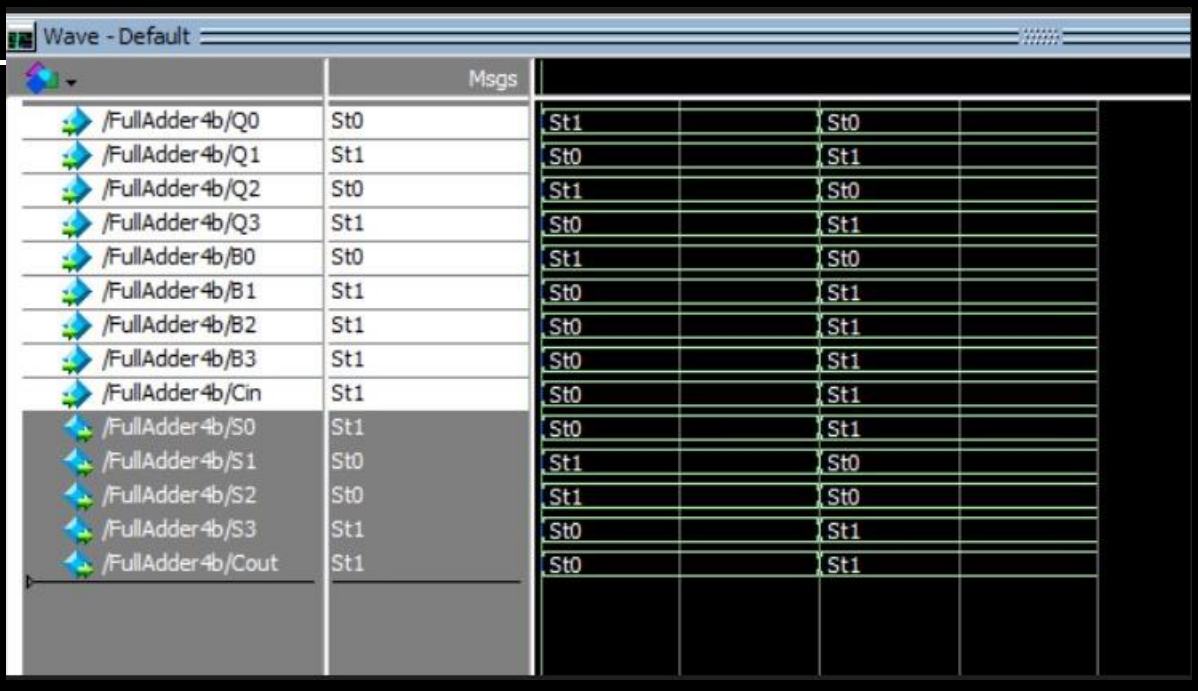

```
module FullAdder1b (
    S,Cout,A,B,Cin
);
    input A,B,Cin; //inputs
    output S,Cout; // outputs
    wire W1,W2,W3; //identifying the wires
    //logic using the drawn schematic
    xor(W1,A,B);
    and(W2,A,B);
    and(W3,W1,Cin);
    or(Cout,W3,W2);
    xor(S,W1,Cin);
endmodule
```



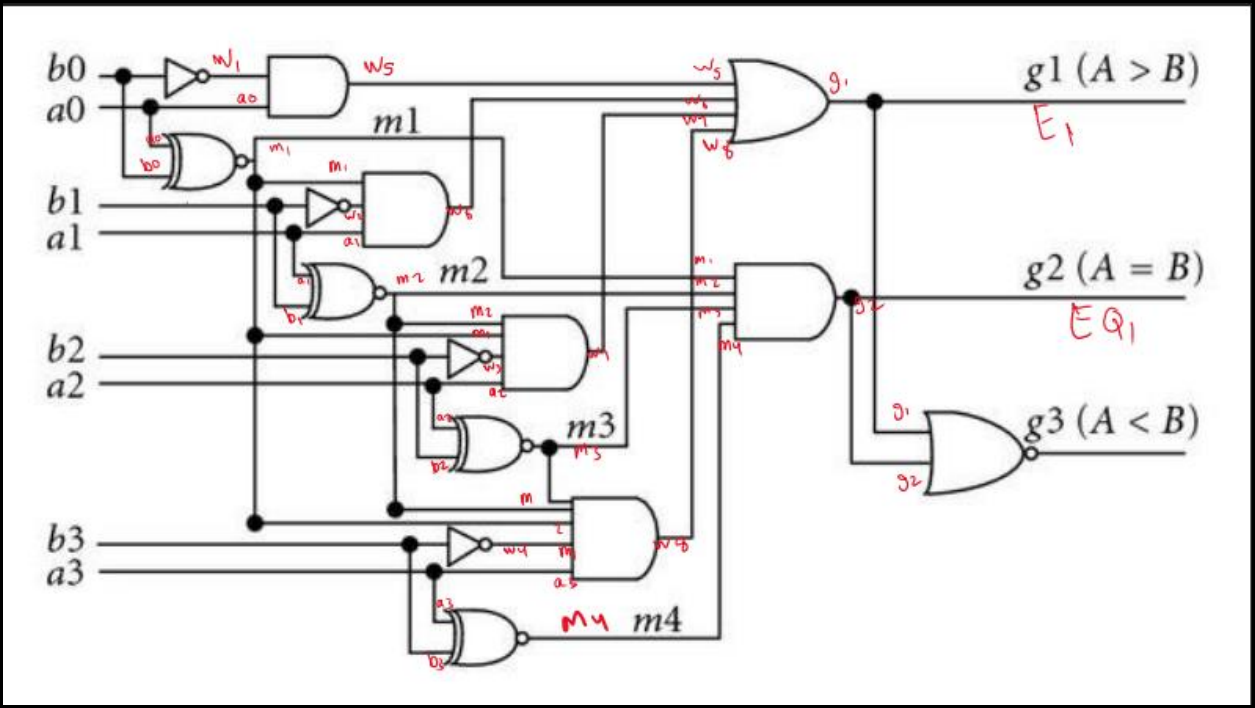
4.6 Full Adder 4 bit



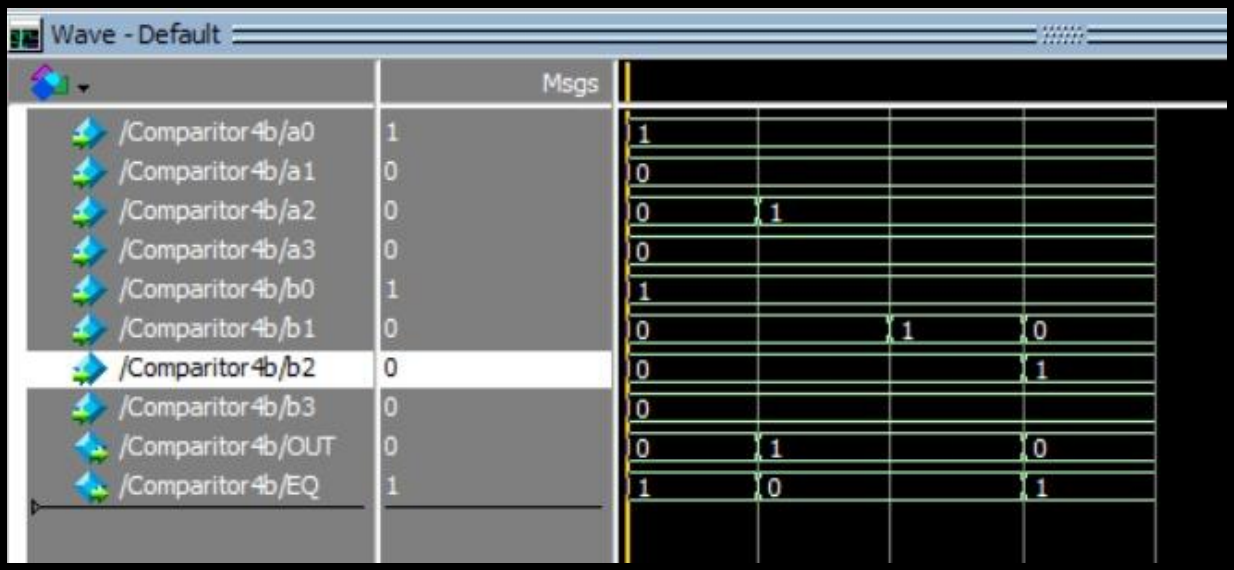

```
module FullAdder4b (
    S0,S1,S2,S3,
    Cout,
    Q0,Q1,Q2,Q3,
    B0,B1,B2,B3,
    Cin
);
    input Q0,Q1,Q2,Q3;
    input B0,B1,B2,B3;
    input Cin;
    output S0,S1,S2,S3;
    output Cout;
    wire wCout[2:0],wCin[14:0];
    FullAdder1b A0(S0,wCout[0],Q0,B0,Cin);
    FullAdder1b A1(S1,wCout[1],Q1,B1,wCout[0]);
    FullAdder1b A2(S2,wCout[2],Q2,B2,wCout[1]);
    FullAdder1b A3(S3,Cout,Q3,B3,wCout[2]);
endmodule
```



4.8 BCD Comparator 4bit




```
module Comparitor4  (
    OUTb E
Q,  a0,a1,a2,a
3,  b0,b1,b2,b
3); // Trure if A >=
    input a0,a1,a2,a
3;  input b0,b1,b2,b
3;  output OUT,E
Q;  wire W1,W2,W3,W4,W5,W6,W7,W8,m1,m2,m3,m4,g1,g2,g
        3;
    not(W1,b
3); not(W2,b
2); not(W3,b
1); not(W4,b
0);
    xnor(m1,a3,b
3); xnor(m2,a2,b
2); xnor(m3,a1,b
1); xnor(m4,a0,b
0);
    and(W5,W1,a
3); and(W6,m1,W2,a
2); and(W7,m2,m1,W3,a
1); and(W8,m3,m2,m1,W4,a
0);
    or(OUT,W5,W6,W7,W
8); and(EQ,m1,m2,m3,m
4); nor(g3,g1,g
2);
    // or(OUT,g1,g
    2);

endmodul
e
```


Wave - Default
Msgs
/Comparitor4b/a0
/Comparitor4b/a1
/Comparitor4b/a2
/Comparitor4b/a3
/Comparitor4b/b0
/Comparitor4b/b1
/Comparitor4b/b2
/Comparitor4b/b3
/Comparitor4b/OUT
/Comparitor4b/EQ
1
0
0
0
1
0
0
1
0
1

5. Faced Problems

- We first had a problem in making the counter count up and down and stop and also load the minimum value and add so we did this buy MUX8 but if we managed to make it count up and down we had a problem at loading so we figured out a way to make the counter act like a register using the mux8.

- Then we had a problem in resetting the value of the counter to zero after it reaches the 9 so we did that by simple and gates connected to the asynchronous reset of each JK Flip Flop.

- The logic of adding two minutes was the most difficult logic because we had to compare with the maximum value after we add and we did that using comparator

- Then we discovered a problem in simulation of adding unit, which is that the adder we used exceeded the value we wanted which is 9 and it was adding to 1111 which is 15 so we had to make a BCD adder.

- Making BCD adder was very tricky to understand and the resources were not easy.

- We faced the same problem in subtraction as well as adding so we had to make BCD subtractor.

- We tried to not manipulate the clock in the clock divider but we couldn’t think of anything else.

- In counting down there were a problem in resetting the value of the counter to 9 or 5 again and that was solved by simple logic and gates and inverters that takes input from the Q’s and the output of it is going to the reset and preset of the counter.

- We used at first Flastad website to simulate and then we used Logisim for better space and better control

- We had a problem after designing all counters in Logisim, the counter didn’t function in Model Sim and we fixed it by and gates as mentioned before.

- The connection of each counter using trigger we thought of making an and gate and its output is connected to the J K and the input of the AND gate is the trigger and the output of the MUX 8 but this violated our logic in loading so we put the trigger in the CLK of the following counter.

6. Tasks Distributions among the team members

Mohamed Elsayed:

- The Clock divider unit (logic, coding, and drawing)
- The Operation control unit (logic, coding, and some drawing)
- BCD Adder (logic and coding)
- BCD Subtractor (logic and coding)
- BCD Comparator (logic and coding)
- Coded most of the project modules
- Helped in solving some issues of the BCD counter & 06 counter
- Wrote some sections in the report

Ahmed Bassam:

- The BCD Counter & 06 counter (logic and drawing)
- BCD Checker (logic, coding, and drawing)
- Draw and redraw most of the schematic and any brainstorming
- Helped in the Clock divider unit
- Helped in the Operation control unit
- Wrote most of the sections in the report

Khaled Ashraf:

- Wrote the truth table of most of the modules
- Simulated all the modules and took screenshots of them
- The 7 segment decoder