**Ahmed Abuhjar**

**Cpre 281**

**Final Project**

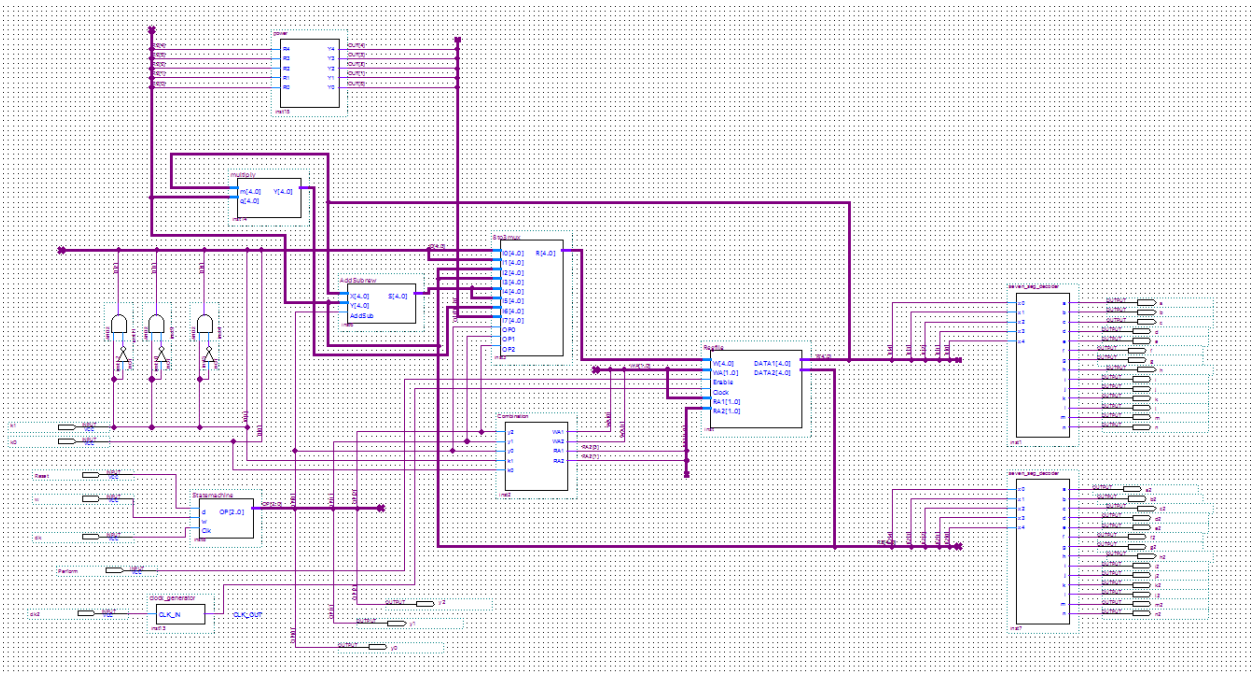
# Final Project: Handheld Calculator

## Introduction:

In this lab, I designed a handheld calculator using Quartus Software. The design is consisted of gates, some of which were implemented using hard code in VHDL. The entire design consists of several parts. The finite state machine, the arithmetic unit, and the register file. Using the FPGA Cyclone board, I tested the design to verify that it outputs the correct result based on the operation chosen.

## Top level diagram of the design

The design of the calculator is shown in the picture below.

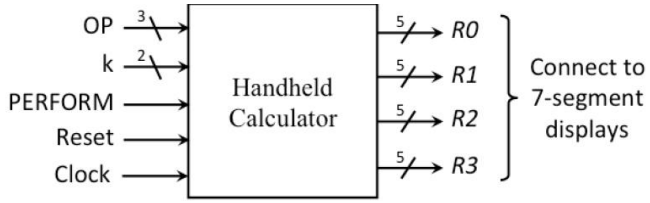


The design has the following inputs.

- W input which specifies the operational three digit inputs. The choice of the input OP can be done by the FSM when w is high. Else, when w is low, the FSM will immediately give the OP 000, which is in this case conceded as the standard state.
- K input of two digits, which will specify the input to be loaded or the address of which register to load into.
- Perform key, which will enable or disable the operation chosen by the FSM.
- Reset, which will reset the values stored in all of the registers to the standard values.
- Clock signal that will be used to switch between the operations.

The design has the following outputs:

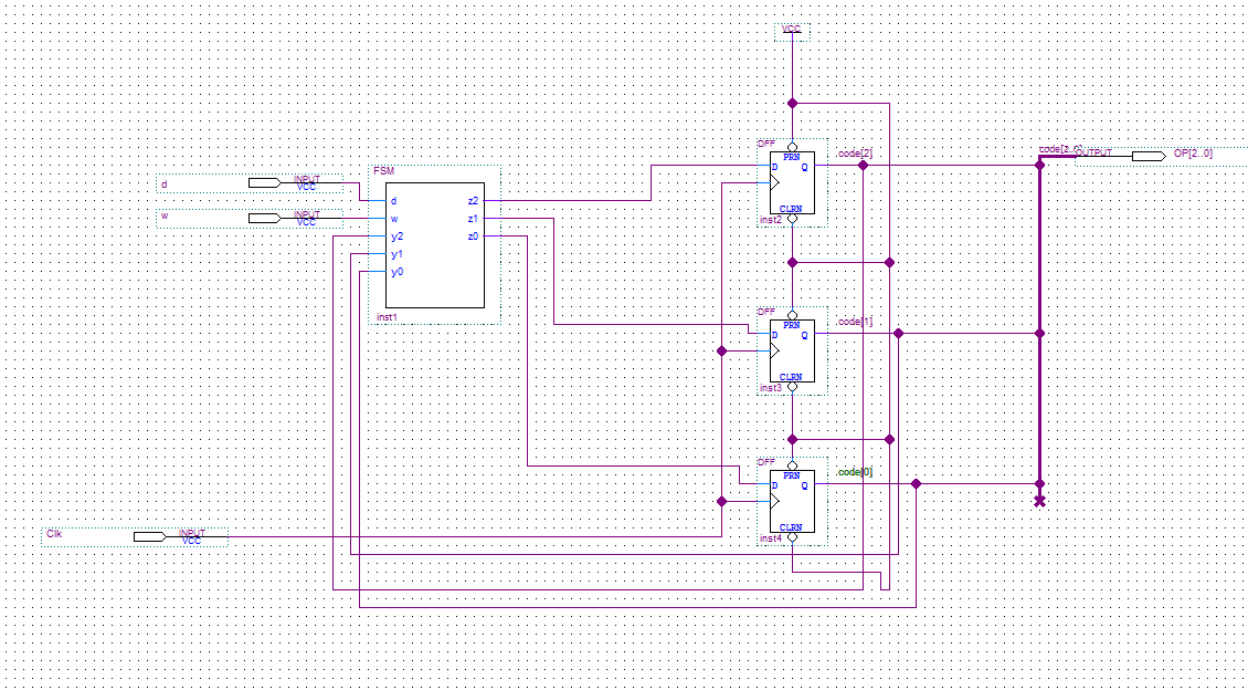
- Four 5 bit binary outputs each of which is stored in each register. The design will display the vales two registers at once since each register will have 2 decimal digits to display.




The design has been broken into different stages described as follows.

**Finite State Machine:**

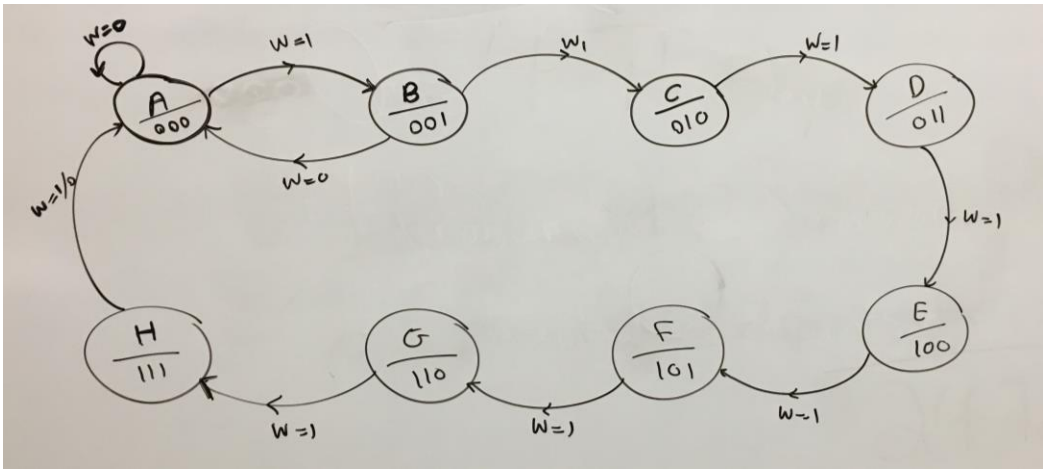
The finite state machine is consisted of D-flip flops to store the value of the present states that are used as inputs to determine the next state. The state machine also includes combinational circuits to express the relationship between the current and the next state in the FSM. The following shows a sketch.




The combinational circuit showed in the sketch and labeled FSM has the code implemented as the following

```
module FSM(d,w,y2,y1,y0,z2,z1,z0);

   input d,w,y2,y1,y0;
   output z2,z1,z0;
   reg  z2,z1,z0;

   //always@ (*)
      //begin
            //assign a = (~x3&~x2&~x1&x0)|(~x3&x2&~x1&~x0)|(x3&~x2&x1&x0)|(x3&x2&~x1&x
            //assign b = (x2&x1&~x0)|(x3&x1&x0)|(x3&x2&~x0)|(~x3&x2&~x1&x0);
            //assign c = (x3&x2&~x0)|(x3&x2&x1)|(~x3&~x2&x1&~x0);
            //assign d = (x2&x1&x0)|(~x3&~x2&~x1&x0)|(~x3&x2&~x1&~x0)|(x3&~x2&x1&~x0);
            //assign e = ~x3&(x0|x2&~x1)| ~x2&~x1&x0;
         // assign f = ~x3&~x2&(x1|x0) | ~x3&x1&x0 | x3&x2&~x1&x0;
         // assign g = ~x3&(~x2&~x1 | x2&x1&x0)| (x3&x2&~x1&~x0);


   //reg Y2,Y1,Y0;
      always@(d or w or y2 or y1 or y0)
      begin
         case({d,w,y2,y1,y0})
         5'b00000:{z2,z1,z0}= 3'b000;
         5'b00001:{z2,z1,z0}= 3'b000;
         5'b00010:{z2,z1,z0}= 3'b000;
         5'b00011:{z2,z1,z0}= 3'b000;
         5'b00100:{z2,z1,z0}= 3'b000;
         5'b00101:{z2,z1,z0}= 3'b000;
         5'b00110:{z2,z1,z0}= 3'b000;
         5'b00111:{z2,z1,z0}= 3'b000;


         5'b01000:{z2,z1,z0}= 3'b001;
         5'b01001:{z2,z1,z0}= 3'b010;
         5'b01010:{z2,z1,z0}= 3'b011;
         5'b01011:{z2,z1,z0}= 3'b100;
         5'b01100:{z2,z1,z0}= 3'b101;
         5'b01101:{z2,z1,z0}= 3'b110;
         5'b01110:{z2,z1,z0}= 3'b111;
         5'b01111:{z2,z1,z0}= 3'b000;


         5'b10000:{z2,z1,z0}= 3'b000;
         5'b10001:{z2,z1,z0}= 3'b000;
         5'b10010:{z2,z1,z0}= 3'b000;
         5'b10011:{z2,z1,z0}= 3'b000;
         5'b10100:{z2,z1,z0}= 3'b000;
         5'b10101:{z2,z1,z0}= 3'b000;
         5'b10110:{z2,z1,z0}= 3'b000;
         5'b10111:{z2,z1,z0}= 3'b000;


         5'b11000:{z2,z1,z0}= 3'b000;
         5'b11001:{z2,z1,z0}= 3'b000;
         5'b11010:{z2,z1,z0}= 3'b000;
         5'b11011:{z2,z1,z0}= 3'b000;
         5'b11100:{z2,z1,z0}= 3'b000;
         5'b11101:{z2,z1,z0}= 3'b000;
         5'b11110:{z2,z1,z0}= 3'b000;
         5'b11111:{z2,z1,z0}= 3'b000;
```

The state diagram of the state machine was first drawn as follows. As mentioned above, the state of which the operation is chosen moves to the next state if the input w is high. Otherwise it will return to the standard state at which the operation is 000. This will occur when the input w is low.



**ALU or the arithmetic unit:**

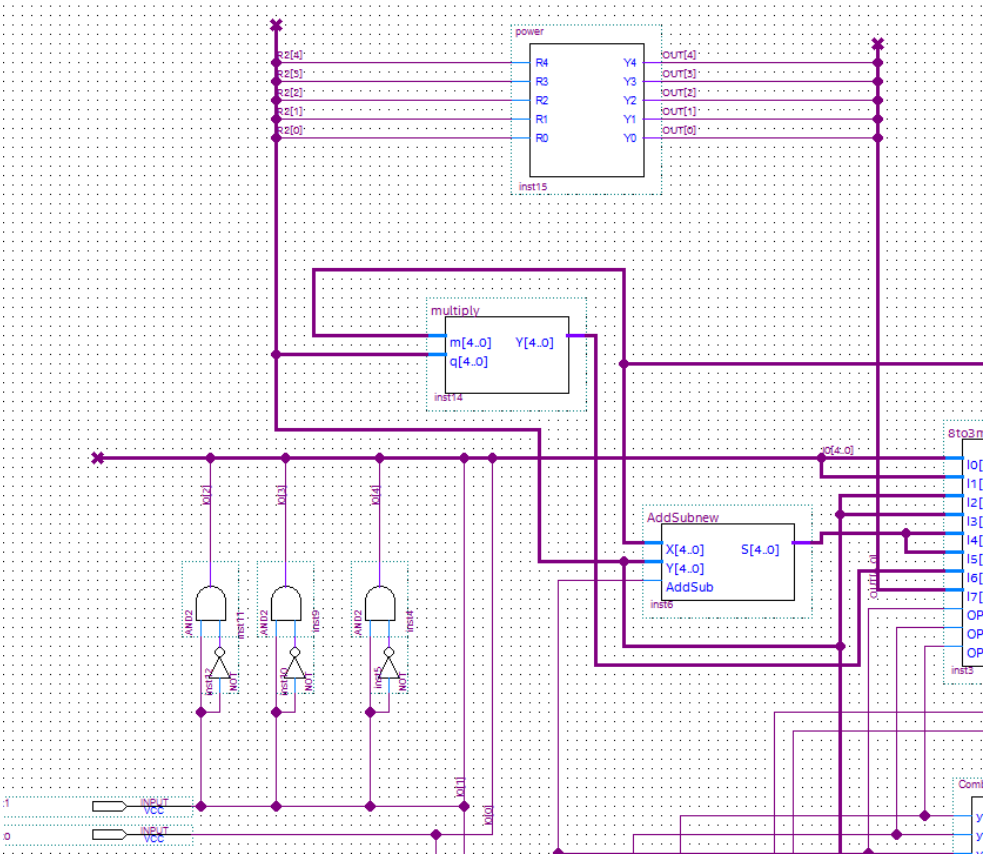
This unit consists of several combinational circuits that perform the operation based on the input specified by the FSM. The operations that can be done using this design specifically are the following.

| OP | Operation |
|---|---|
| 000 | $R0 \leftarrow 0, R1 \leftarrow 1, R2 \leftarrow 2, R3 \leftarrow 3$ |
| 001 | $R0 \leftarrow k$ |
| 010 | $R0 \leftarrow Rk$ |
| 011 | $Rk \leftarrow R0$ |
| 100 | $R0 \leftarrow R0 + Rk$ |
| 101 | $R0 \leftarrow R0 - Rk$ |
| 110 | $R0 \leftarrow R0 \times Rk$ |
| 111 | $R0 \leftarrow 2^{Rk}$ |

The arithmetic operations shown in the table above are the following

- Adding / Subtracting
- Multiplying
- Exponent of 2

The sketch below shows the combinational circuits that perform the operations above.

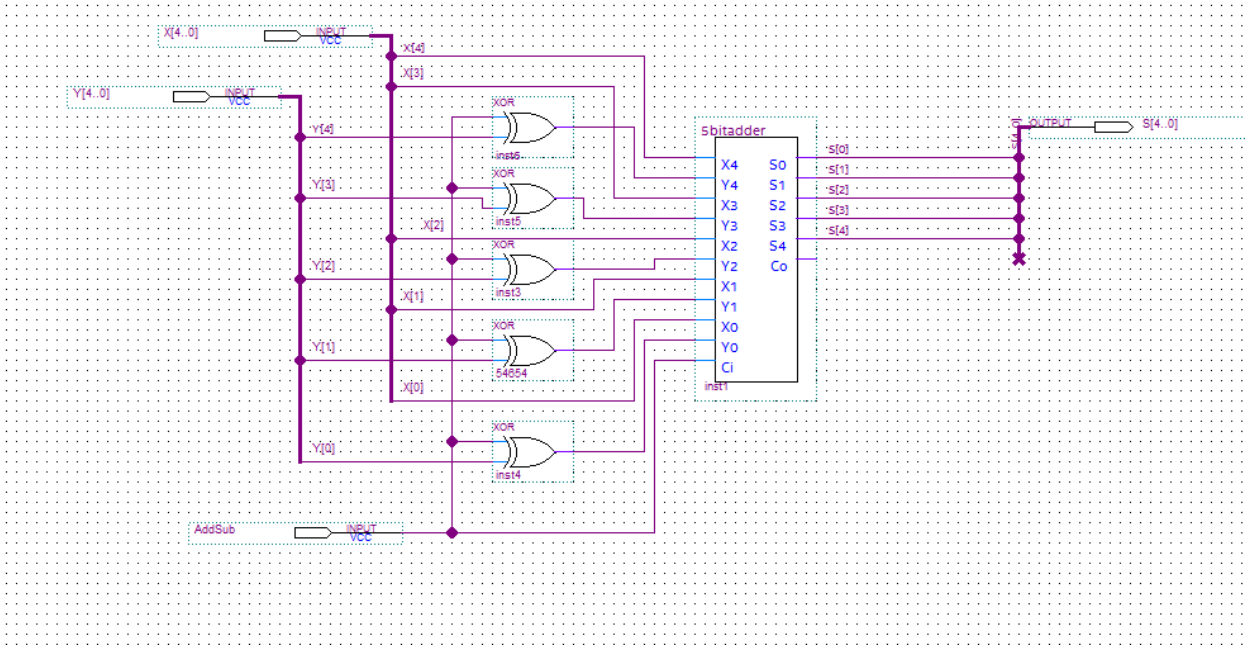


**Adding/ Subtracting:**

This unit uses five bit adder to perform the operation of the two 5-bit inputs. Since the addition should be performed when the OP is 100 and the subtraction should be performed when OP is 101, the least input y0 from FSM will determine the addition or subtraction operation.

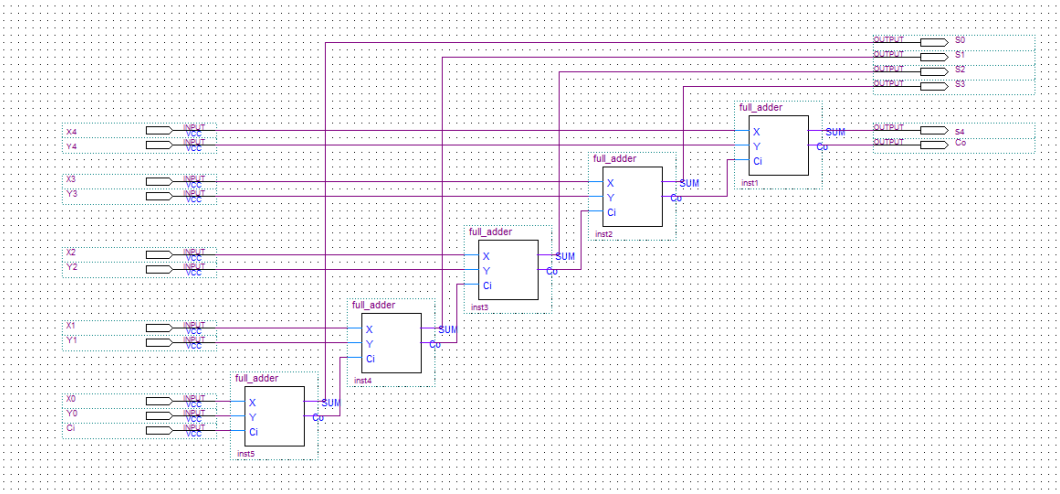
When subtracting, we're basically adding the 2's complement of the second input to the first input. This is done by having XORing all the inputs with the least input y0 from the FSM. This y0 is supplied to the input ci in the five bit adder.

The sketch below shows the unit combinational circuit. This will have an output of 5 bit binary number.

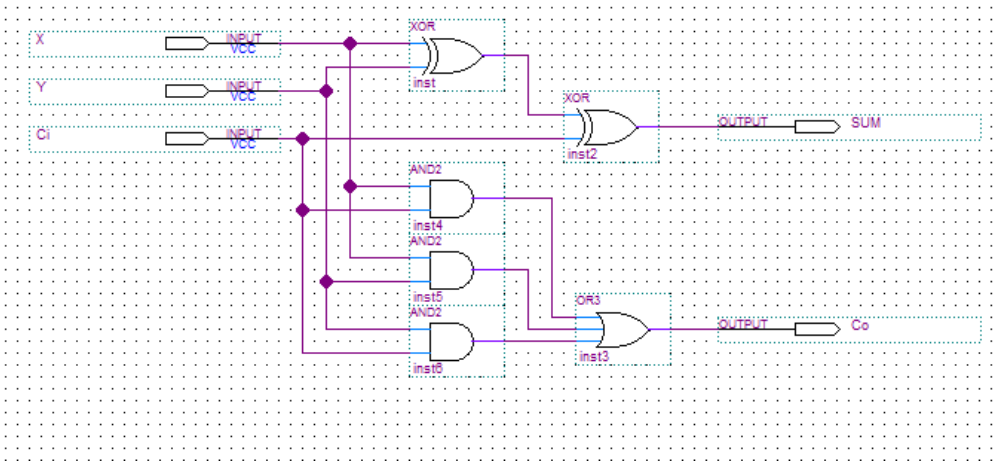



**Five bit adder**

The unit that performs the addition is shown below, which used a full adder as a sun unit

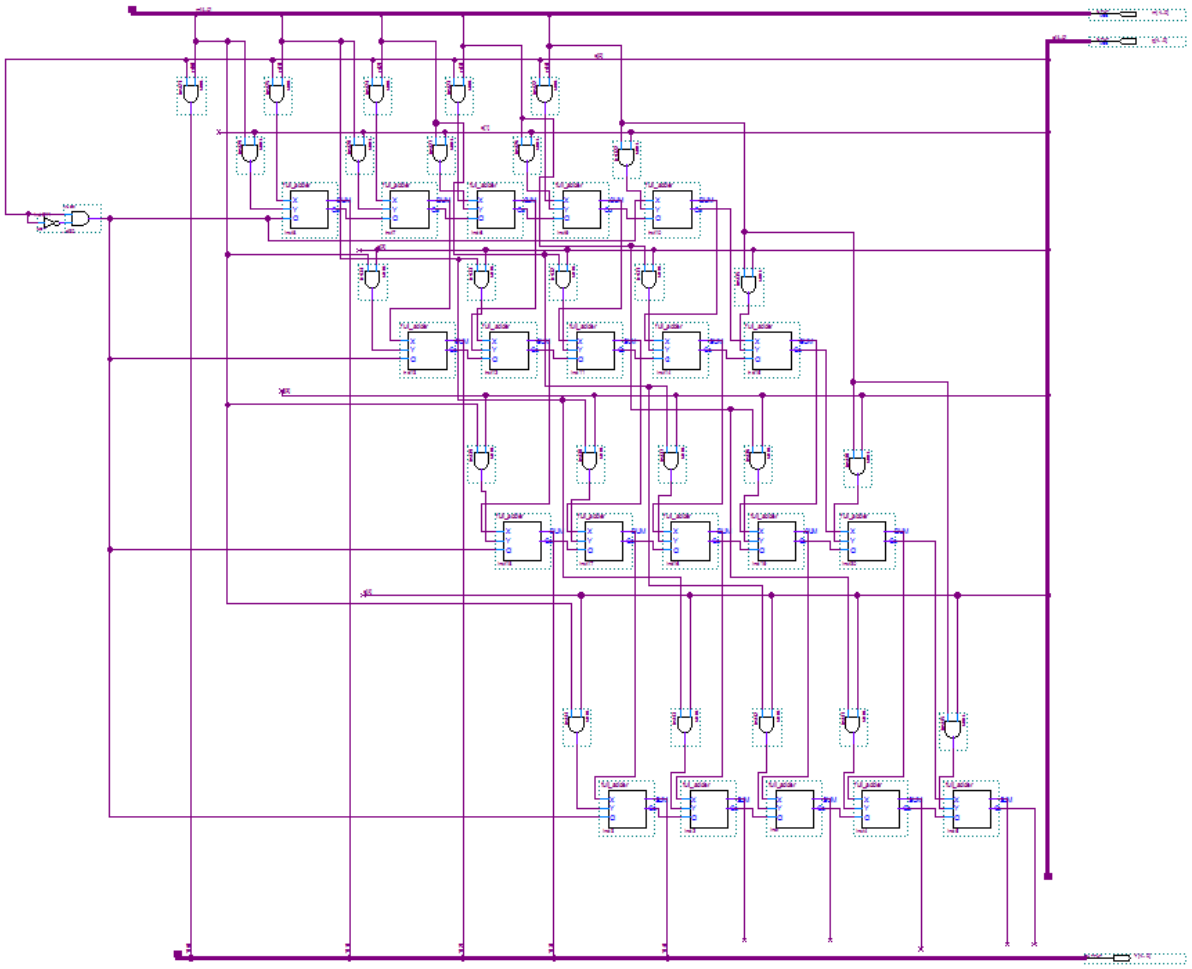


**The full adder is shown below**




**Multiplying:**

This unit is implemented as follows, and will output a 10 digit binary number. Since we're assigning the number to a five bit register, the register will only take the five first bits of the result.



**Exponent of 2:**

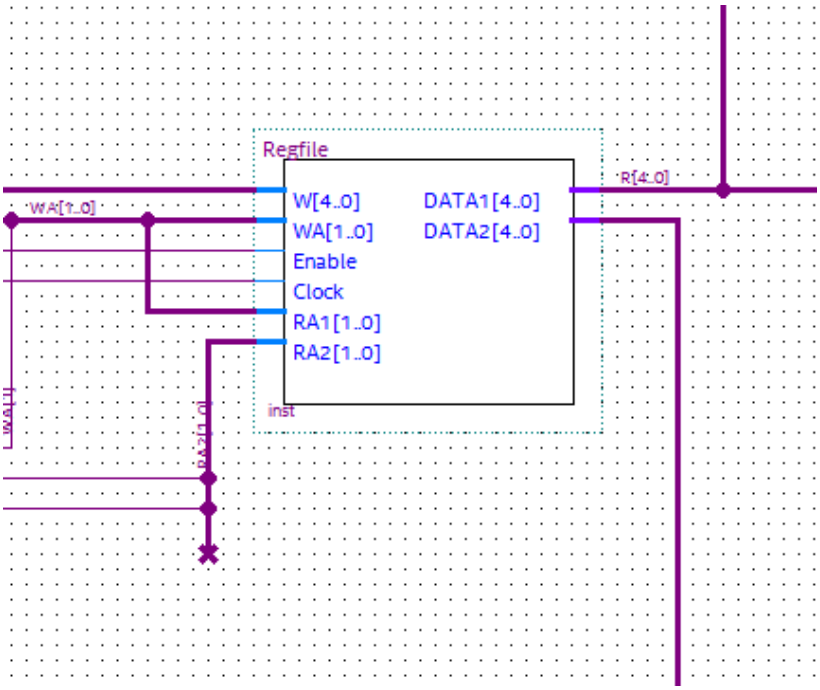
This unit was hard coded using VHDL language, and it will output the result based on the value chosen from the register of address k.

```
module power(R4,R3,R2,R1,R0,Y4,Y3,Y2,Y1,Y0);

   input R4,R3,R2,R1,R0;
   output Y4,Y3,Y2,Y1,Y0;
   reg  Y4,Y3,Y2,Y1,Y0;

   //always@ (*)
      //begin
            //assign a = (~x3&~x2&~x1&x0)|(~x3&x2&~x1&~x0)|(x3&~x2&x1&x0)|(x3&x2&~x1&x0);
            //assign b = (x2&x1&~x0)|(x3&x1&x0)|(x3&x2&~x0)|(~x3&x2&~x1&x0);
            //assign c = (x3&x2&~x0)|(x3&x2&x1)|(~x3&~x2&x1&~x0);
            //assign d = (x2&x1&x0)|(~x3&~x2&~x1&x0)|(~x3&x2&~x1&~x0)|(x3&~x2&x1&~x0);
            //assign e = ~x3&(x0|x2&~x1)| ~x2&~x1&x0;
         // assign f = ~x3&~x2&(x1|x0) | ~x3&x1&x0 | x3&x2&~x1&x0;
         // assign g = ~x3&(~x2&~x1 | x2&x1&x0)| (x3&x2&~x1&~x0);


   //reg Y2,Y1,Y0;
      always@(R4 or R3 or R2 or R1 or R0)
      begin
         case({R4,R3,R2,R1,R0})
         5'b00000:{Y4,Y3,Y2,Y1,Y0}= 5'b00001;
         5'b00001:{Y4,Y3,Y2,Y1,Y0}= 5'b00010;
         5'b00010:{Y4,Y3,Y2,Y1,Y0}= 5'b00100;
         5'b00011:{Y4,Y3,Y2,Y1,Y0}= 5'b01000;
         5'b00100:{Y4,Y3,Y2,Y1,Y0}= 5'b10000;
         5'b00101:{Y4,Y3,Y2,Y1,Y0}= 5'b00000;
         5'b00110:{Y4,Y3,Y2,Y1,Y0}= 5'b00000;
         5'b00111:{Y4,Y3,Y2,Y1,Y0}= 5'b00000;


         5'b01000:{Y4,Y3,Y2,Y1,Y0}= 5'b00000;
         5'b01001:{Y4,Y3,Y2,Y1,Y0}= 5'b00000;
         5'b01010:{Y4,Y3,Y2,Y1,Y0}= 5'b00000;
         5'b01011:{Y4,Y3,Y2,Y1,Y0}= 5'b00000;
         5'b01100:{Y4,Y3,Y2,Y1,Y0}= 5'b00000;
         5'b01101:{Y4,Y3,Y2,Y1,Y0}= 5'b00000;
         5'b01110:{Y4,Y3,Y2,Y1,Y0}= 5'b00000;
         5'b01111:{Y4,Y3,Y2,Y1,Y0}= 5'b00000;


         5'b10000:{Y4,Y3,Y2,Y1,Y0}= 5'b00000;
         5'b10001:{Y4,Y3,Y2,Y1,Y0}= 5'b00000;
         5'b10010:{Y4,Y3,Y2,Y1,Y0}= 5'b00000;
         5'b10011:{Y4,Y3,Y2,Y1,Y0}= 5'b00000;
         5'b10100:{Y4,Y3,Y2,Y1,Y0}= 5'b00000;
         5'b10101:{Y4,Y3,Y2,Y1,Y0}= 5'b00000;
         5'b10110:{Y4,Y3,Y2,Y1,Y0}= 5'b00000;
         5'b10111:{Y4,Y3,Y2,Y1,Y0}= 5'b00000;


         5'b11000:{Y4,Y3,Y2,Y1,Y0}= 5'b00000;
         5'b11001:{Y4,Y3,Y2,Y1,Y0}= 5'b00000;
         5'b11010:{Y4,Y3,Y2,Y1,Y0}= 5'b00000;
         5'b11011:{Y4,Y3,Y2,Y1,Y0}= 5'b00000;
         5'b11100:{Y4,Y3,Y2,Y1,Y0}= 5'b00000;
         5'b11101:{Y4,Y3,Y2,Y1,Y0}= 5'b00000;
         5'b11110:{Y4,Y3,Y2,Y1,Y0}= 5'b00000;
         5'b11111:{Y4,Y3,Y2,Y1,Y0}= 5'b00000;
```

**Register File:**

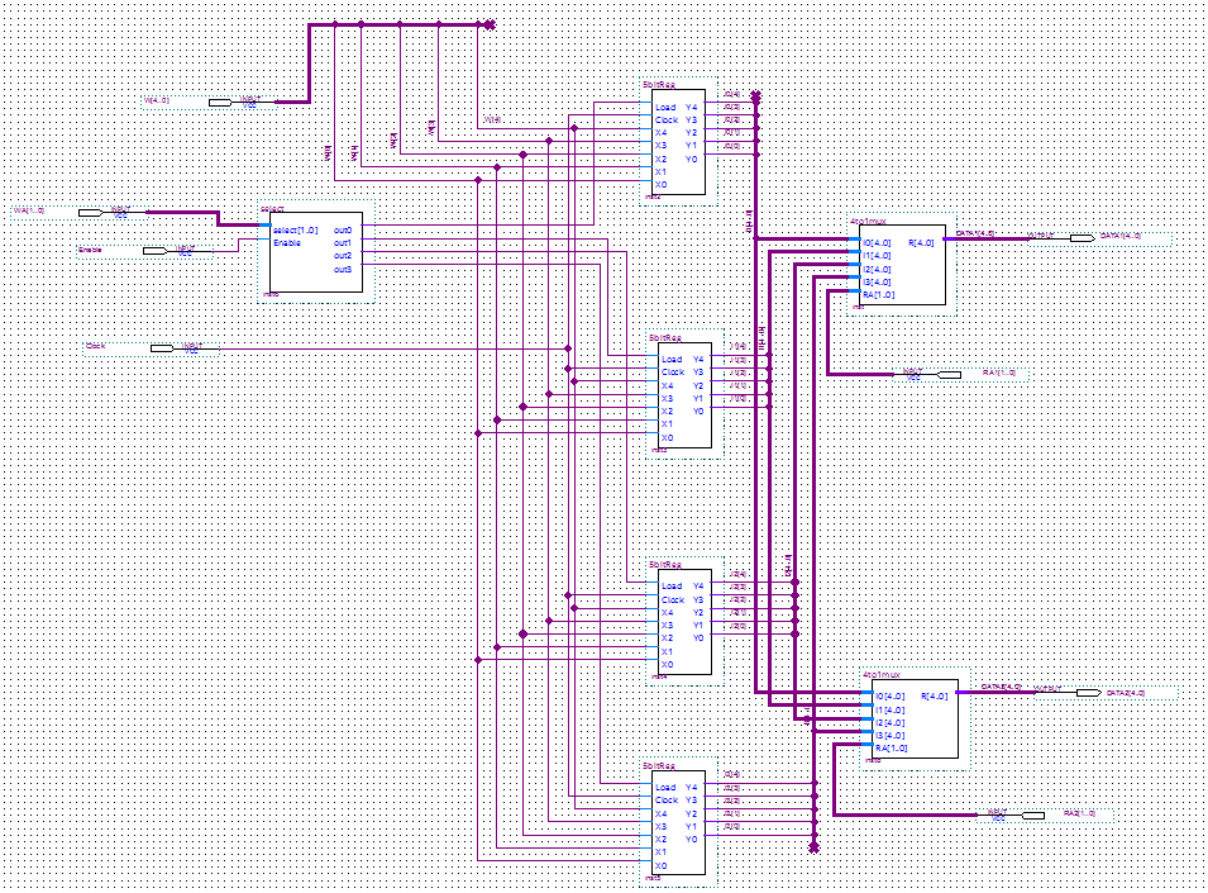
The register file is useful to store the values needed to perform the next operation. The inputs for this unit are the following

- WA will specify which address to write into.
- W of 4 bits which will have the 4 bits loaded to the register specified by the address WA
- Enable will determine whether or not to perform the operation specified by the FSM.
- The clock here is attached to a clock generator to have the operation done automatically once the perform is requested.
- RA will specify the address from we need to read and display the value on the board.




The outputs of the unit is the data from the register of which the address is specified by the corresponding RA.

The following shows the interior design of the register file

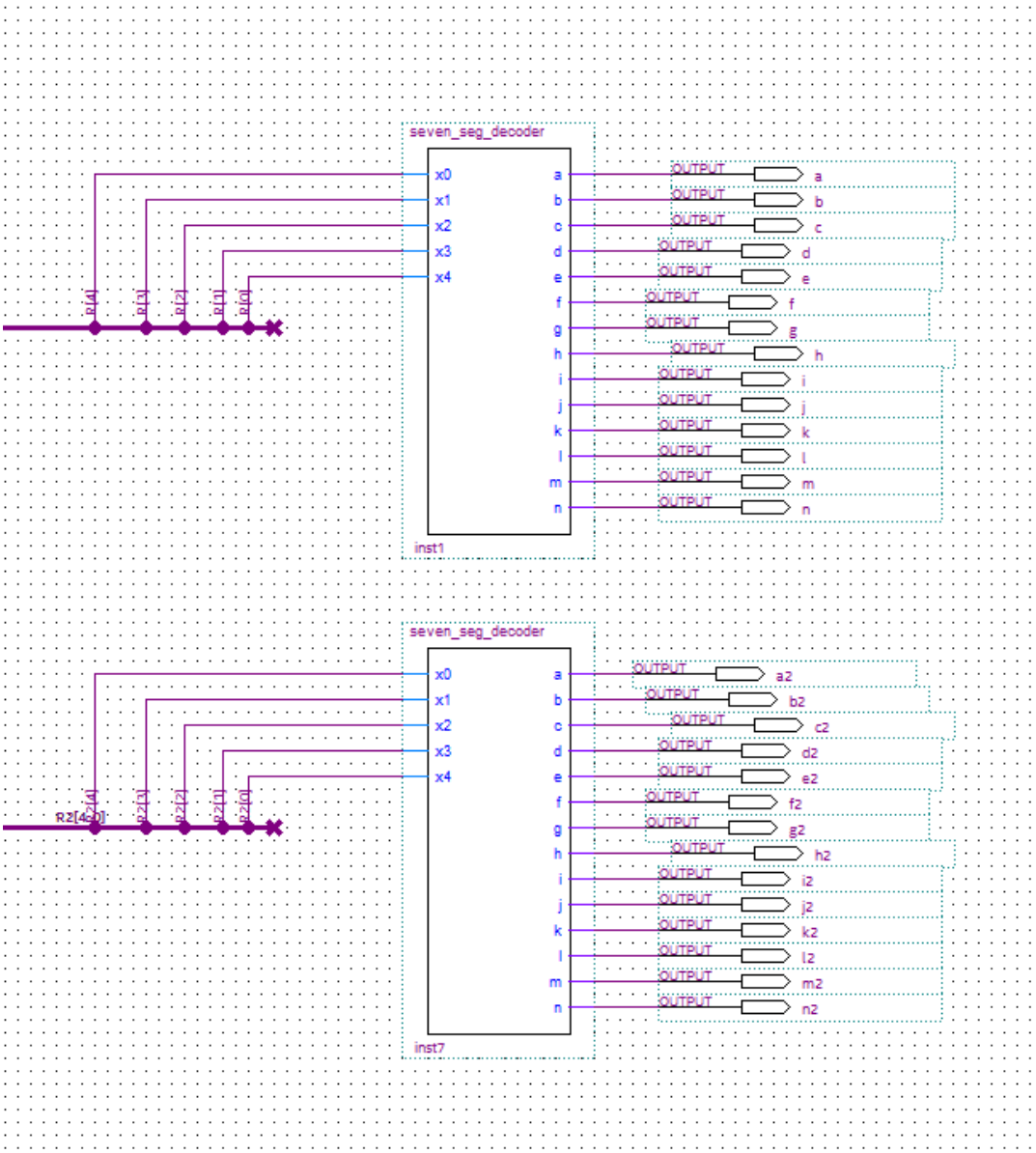



The interior design includes the following

- Select circuit that will determine the address of the register we need to assign a value into.
- 4 five bit register that will load new values if the enable key is high. 2 to 1 multiplexer is used determined the enable or disable of the performance of the circuit.
- Two 4 to 1 mux to read two values from to registers.

**14 segment decoder:**

After having read two values from two registers using the register file, those values are displayed on the board using 14 segment display shown below,



The 14 seg-displays are hard coded as the following

```
module seven_seg_decoder(x0,x1,x2,x3,x4,a,b,c,d,e,f,g,h,i,j,k,l,m,n);

   input x0,x1,x2,x3,x4;
   output a,b,c,d,e,f,g,h,i,j,k,l,m,n;
   reg  a,b,c,d,e,f,g,h,i,j,k,l,m,n;

   //always@ (*)
      //begin
            //assign a = (~x3&~x2&~x1&x0)|(~x3&x2&~x1&~x0)|(x3&~x2&x1&x0)|(x3&x2&~x1&x0);
            //assign b = (x2&x1&~x0)|(x3&x1&x0)|(x3&x2&~x0)|(~x3&x2&~x1&x0);
            //assign c = (x3&x2&~x0)|(x3&x2&x1)|(~x3&~x2&x1&~x0);
            //assign d = (x2&x1&x0)|(~x3&~x2&~x1&x0)|(~x3&x2&~x1&~x0)|(x3&~x2&x1&~x0);
            //assign e = ~x3&(x0|x2&~x1)| ~x2&~x1&x0;
         // assign f = ~x3&~x2&(x1|x0) | ~x3&x1&x0 | x3&x2&~x1&x0;
         // assign g = ~x3&(~x2&~x1 | x2&x1&x0)| (x3&x2&~x1&~x0);


   //reg Y2,Y1,Y0;
      always@(x0 or x1 or x2 or x3 or x4)
      begin
         case({x0,x1,x2,x3,x4})
         5'b0000000:{a,b,c,d,e,f,g,h,i,j,k,l,m,n}= 14'b00000010000001;
         5'b0000001:{a,b,c,d,e,f,g,h,i,j,k,l,m,n}= 14'b10011110000001;
         5'b0000010:{a,b,c,d,e,f,g,h,i,j,k,l,m,n}= 14'b00100100000001;
         5'b0000011:{a,b,c,d,e,f,g,h,i,j,k,l,m,n}= 14'b00001100000001;
         5'b0000100:{a,b,c,d,e,f,g,h,i,j,k,l,m,n}= 14'b10011000000001;
         5'b0000101:{a,b,c,d,e,f,g,h,i,j,k,l,m,n}= 14'b01001000000001;
         5'b0000110:{a,b,c,d,e,f,g,h,i,j,k,l,m,n}= 14'b01000000000001;
         5'b0000111:{a,b,c,d,e,f,g,h,i,j,k,l,m,n}= 14'b00011110000001;
         5'b0001000:{a,b,c,d,e,f,g,h,i,j,k,l,m,n}= 14'b00000000000001;
         5'b0001001:{a,b,c,d,e,f,g,h,i,j,k,l,m,n}= 14'b00001000000001;

         5'b0001010:{a,b,c,d,e,f,g,h,i,j,k,l,m,n}= 14'b00000011001111;
         5'b0001011:{a,b,c,d,e,f,g,h,i,j,k,l,m,n}= 14'b10011111001111;
         5'b0001100:{a,b,c,d,e,f,g,h,i,j,k,l,m,n}= 14'b00100101001111;
         5'b0001101:{a,b,c,d,e,f,g,h,i,j,k,l,m,n}= 14'b00001101001111;
         5'b0001110:{a,b,c,d,e,f,g,h,i,j,k,l,m,n}= 14'b10011001001111;
         5'b0001111:{a,b,c,d,e,f,g,h,i,j,k,l,m,n}= 14'b01001001001111;
         5'b0010000:{a,b,c,d,e,f,g,h,i,j,k,l,m,n}= 14'b01000001001111;
         5'b0010001:{a,b,c,d,e,f,g,h,i,j,k,l,m,n}= 14'b00011111001111;
         5'b0010010:{a,b,c,d,e,f,g,h,i,j,k,l,m,n}= 14'b00000001001111;
         5'b0010011:{a,b,c,d,e,f,g,h,i,j,k,l,m,n}= 14'b00001001001111;

         5'b0010100:{a,b,c,d,e,f,g,h,i,j,k,l,m,n}= 14'b00000010010010;
         5'b0010101:{a,b,c,d,e,f,g,h,i,j,k,l,m,n}= 14'b10011110010010;
         5'b0010110:{a,b,c,d,e,f,g,h,i,j,k,l,m,n}= 14'b00100100010010;
         5'b0010111:{a,b,c,d,e,f,g,h,i,j,k,l,m,n}= 14'b00001100010010;
         5'b0011000:{a,b,c,d,e,f,g,h,i,j,k,l,m,n}= 14'b10011000010010;
         5'b0011001:{a,b,c,d,e,f,g,h,i,j,k,l,m,n}= 14'b01001000010010;
         5'b0011010:{a,b,c,d,e,f,g,h,i,j,k,l,m,n}= 14'b01000000010010;
         5'b0011011:{a,b,c,d,e,f,g,h,i,j,k,l,m,n}= 14'b00011110010010;
         5'b0011100:{a,b,c,d,e,f,g,h,i,j,k,l,m,n}= 14'b00000000010010;
         5'b0011101:{a,b,c,d,e,f,g,h,i,j,k,l,m,n}= 14'b00001000010010;

         5'b0011110:{a,b,c,d,e,f,g,h,i,j,k,l,m,n}= 14'b00000010000110;
         5'b0011111:{a,b,c,d,e,f,g,h,i,j,k,l,m,n}= 14'b10011110000110;

      endcase
      end
   endmodule
```

**Demonstration and the result**

The test plan to determine if the result is correct can be done in either of the following ways

Operation → 000 → 010 → 100 → 110 → 001 → 011 → 101 → 111