Sullivan Hart

Md Maruf Ahamed

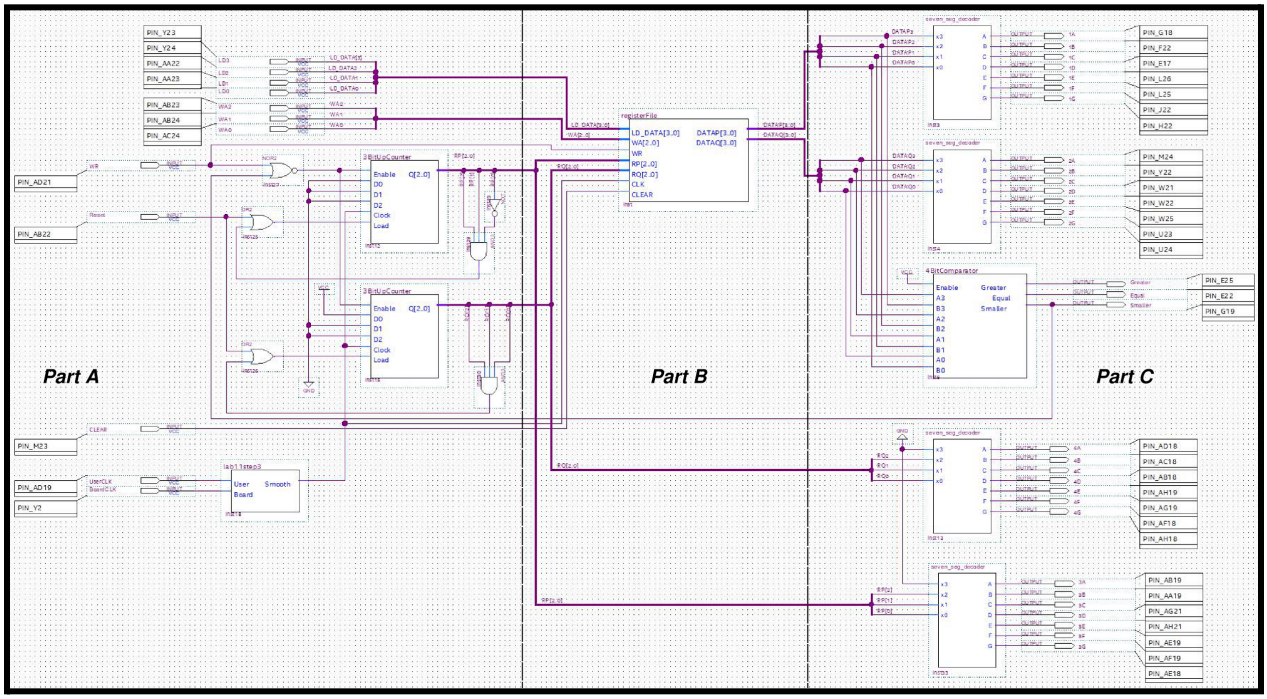
CPR E 281: Section 7

5 May 2023

Final Project Report: “Project #1”

I chose to complete Project #1, circuit for checking if a list of numbers is sorted. My report contains a holistic view and description of my design as well as part-by-part view and explanation of each module.

Top Level Diagram



The Top Level Diagram includes three parts: Part A, Part B, and Part C. Together, the parts make a complete circuit that is able to toggle between two modes: load, intake four bit numbers from the board to store in a register file; and search, iterate in a loop

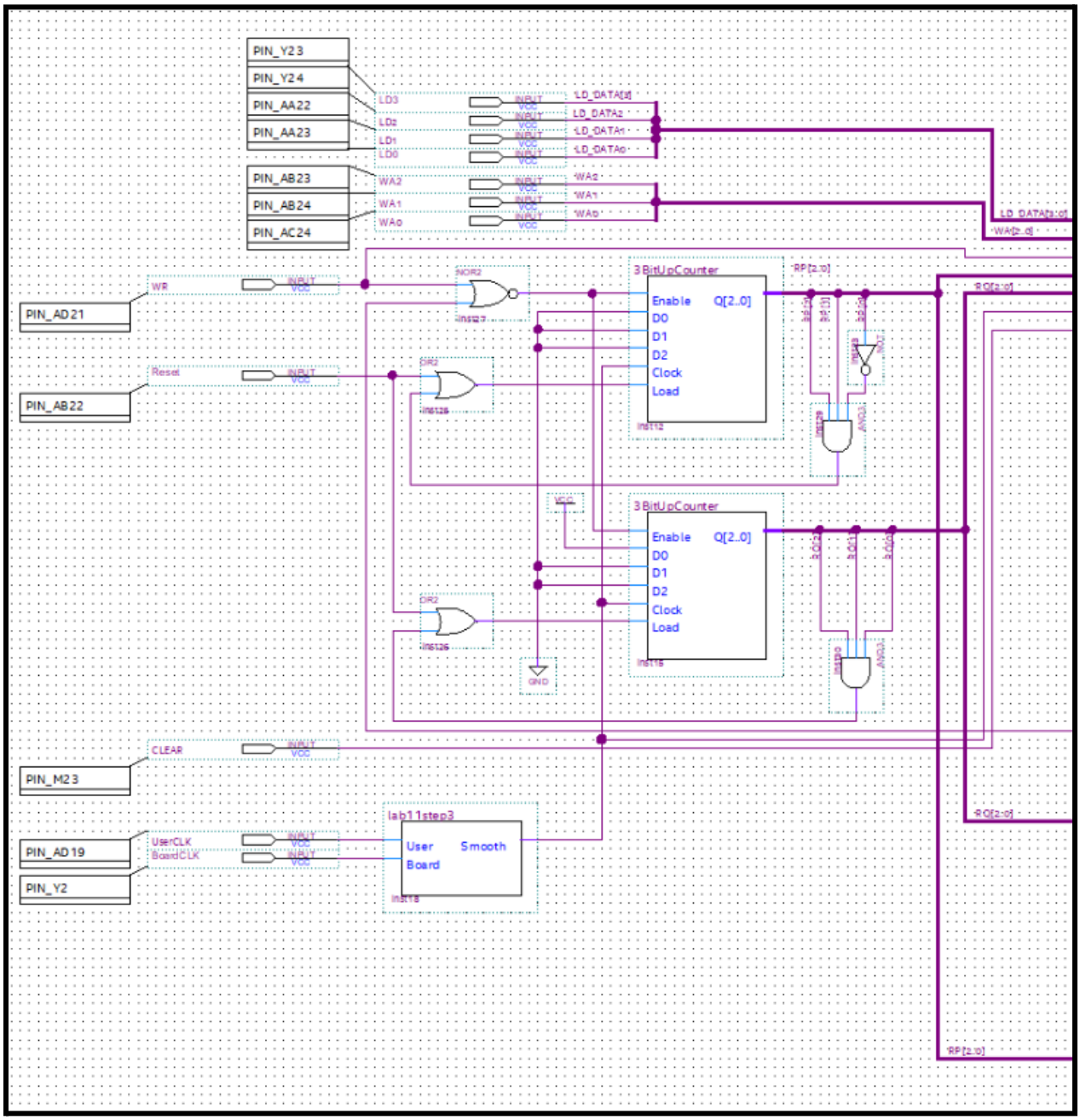
through that same register file to ensure the numbers are sorted in ascending order (higher indexes have higher numbers).

The inputs are a variety of board-generated signals: LD0 to LD3 (the number to insert in the register file), WA0 to WA2 (the index in the register file to write the LD number to), WR (the toggle between the two modes of the circuit, load and search), Reset (the reset toggle for the search loop), CLEAR (sets each register in the file to 0000), userCLK (the manual clock to iterate through the search loop and input the numbers in load), BoardCLK (the board's clock of 50 MHz), and various VCCs and GNDs. The outputs include an assortment of lights to make the output appear in a human-readable way: 1A to 1G (the value stored in the highest index being compared), 2A to 2G (the value stored in the lowest index being compared), Greater (asserts if the higher index's stored value is larger than the lowest index's), Equal (asserts if the higher index's stored value is the same as the lowest index's), Smaller (asserts if the higher index's stored value is less than the lowest index's), and 3A to 3G (the lowest index being accessed), 4A to 4G (the highest index being accessed).

To produce the desired outputs, the WR switch dictates if the circuit is in load or search mode. When WR asserts, the circuit is in load mode. The register file uses LD and WA to store values. When WR doesn't assert, the register file uses a state machine to cycle through the register file. As long as the higher index's value is greater or equal than the lower index's value, the user is able to press the button to search through the register file. When the higher index's value is smaller, so not in ascending order, the user is no longer able to cycle through the file; when the array isn't ascending, the output is frozen until the user resets the search with the Reset button. To make the output clearer, a green light

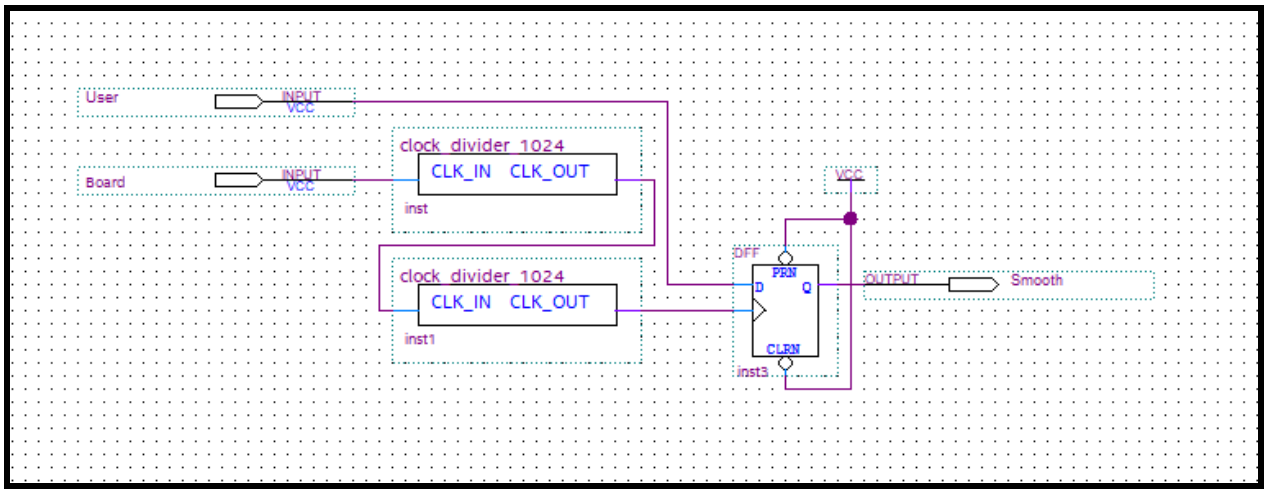
illuminates when the pair of numbers being compared is ascending or equal, and the light switches to red when the pair is descending.

Part A



Part A consists of the inputs, which were previously discussed, and the state machine. The state machine takes input from the comparator output, the board clock, the user's clock, WR, and reset. The counters count in the sequence (1, 0), (2, 1), (3, 2), (4, 3), (5, 4), (6, 5), (7, 6), and back to (1, 0) as long as WR, Smaller, and Reset don't assert. If WR or Smaller asserts, the counters are no longer enabled, so they are effectively paused. If Reset asserts, the counters output (1, 0) for as long as reset asserts. The starting values are given in each counter's D0 to D2 input, and receive their desired starting values as binary made with VCC and GND. The counters reset to their respective starting values (1 and 0) when Load receives a high input. To accomplish this, Load receives a high input whenever Reset is asserted or the output is the last number in the cycle (7 and 6). Finally, Part A includes a damper for the clock signal. The damper uses the board's clock signal and a button controlled by the user to create a more accurate input from the user.

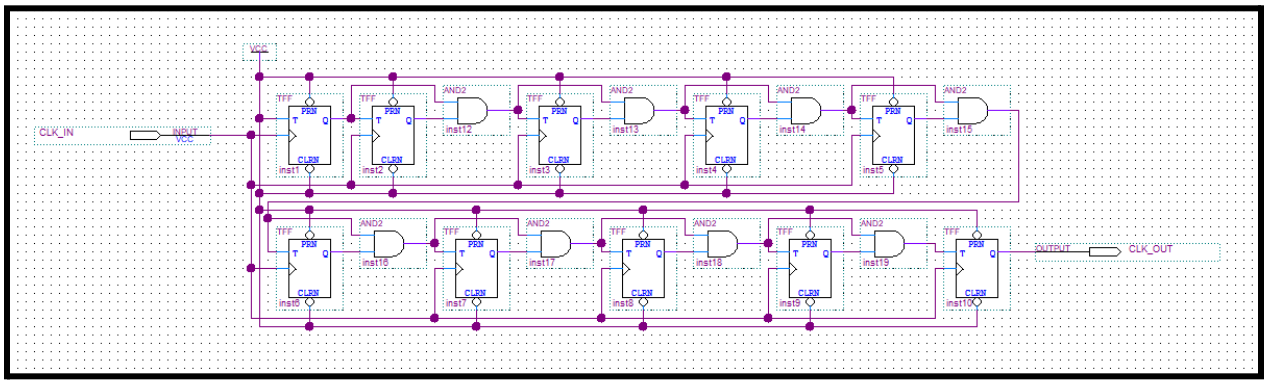
Clock Debouncer



The user input is passed through a D flip-flop each time a slowed version of the board's clock encounters a rising edge. The board's clock passes through two clock divider

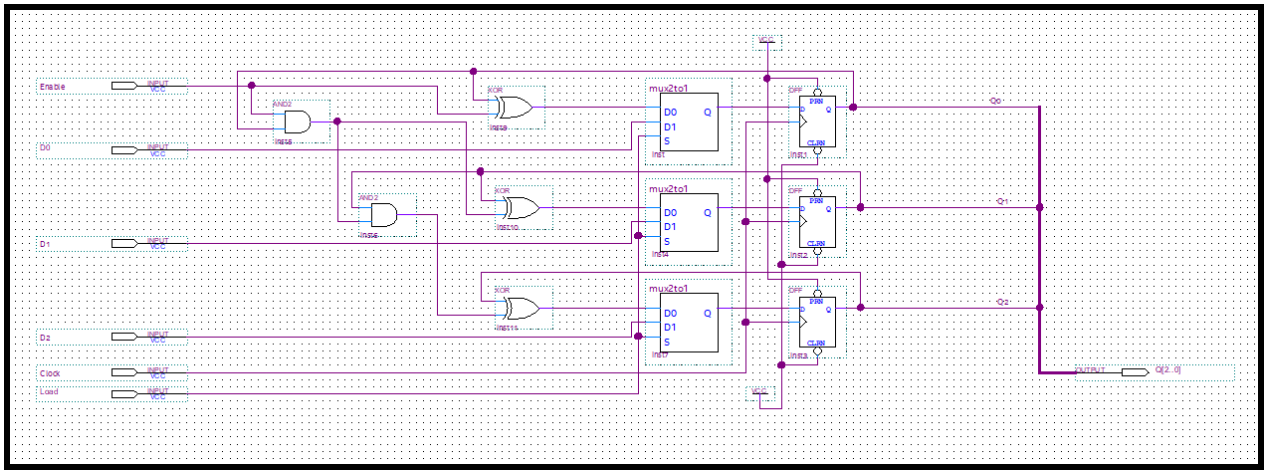
circuits. Each time it passes through, it reduces the number of rising edges per second to 1/1024 of the previous rate. Since it does this twice, the original speed of 50 MHz turns into ~47.68 Hz or .02 seconds. The output, smooth, is a consistent output when compared to the board's button's output. The board's clock is slowed by using the clock divider.

Clock Divider



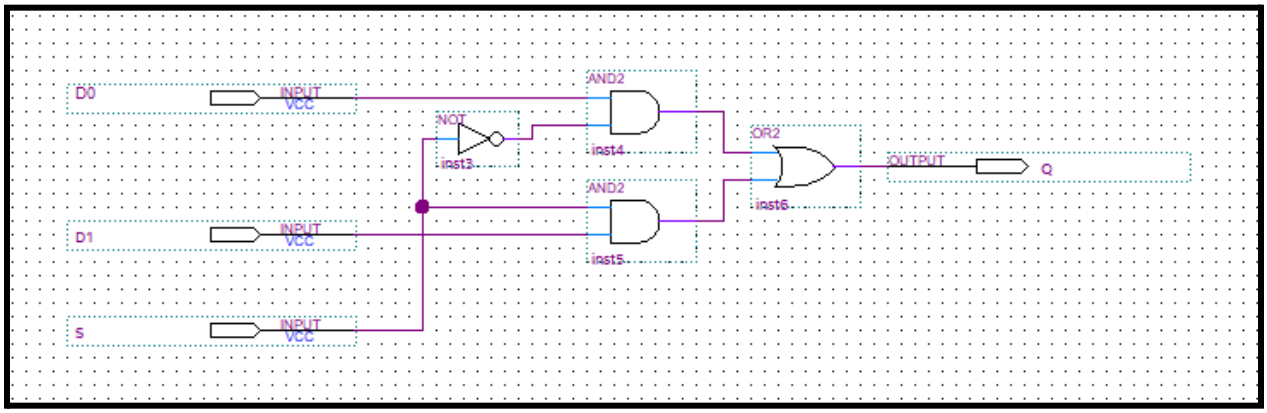
The clock divider reduces the input signal by using a counter. The counter, consisting of ten T flip-flops, asserts an output, after it receives 1024 rising edges. This can be seen by the fact that two to the tenth power is 1024.

3 Bit Counter



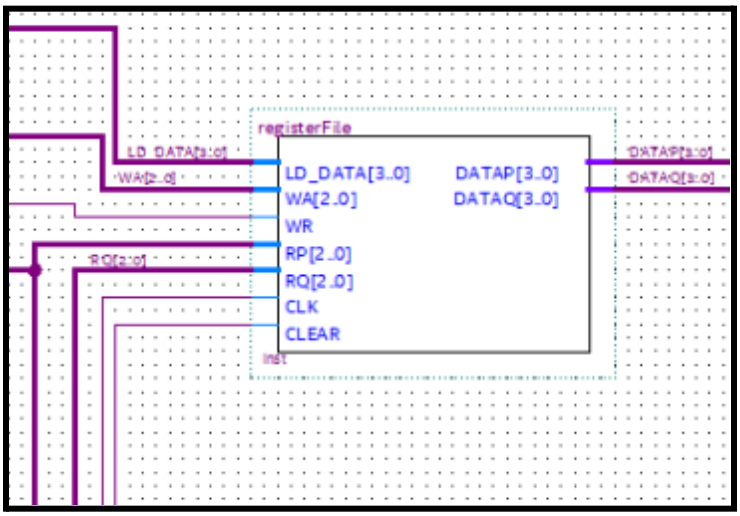
The counter, when enabled, will start at a value (D0, D1, D2) and continue to count up each time it receives a clock signal until it reaches the largest number it can hold (1, 1, 1). When not enabled, it won't do anything. When load is asserted, the initial values will be outputted each time there's a clock signal. To achieve this, it uses AND gates to ensure the circuit only counts when it has a high enable input. It uses multiplexers to pass through the load signal when load is asserted, and the D flip-flop's output from the previous clock cycle. It takes the current output of the three D flip-flops and combines them to an output bus.

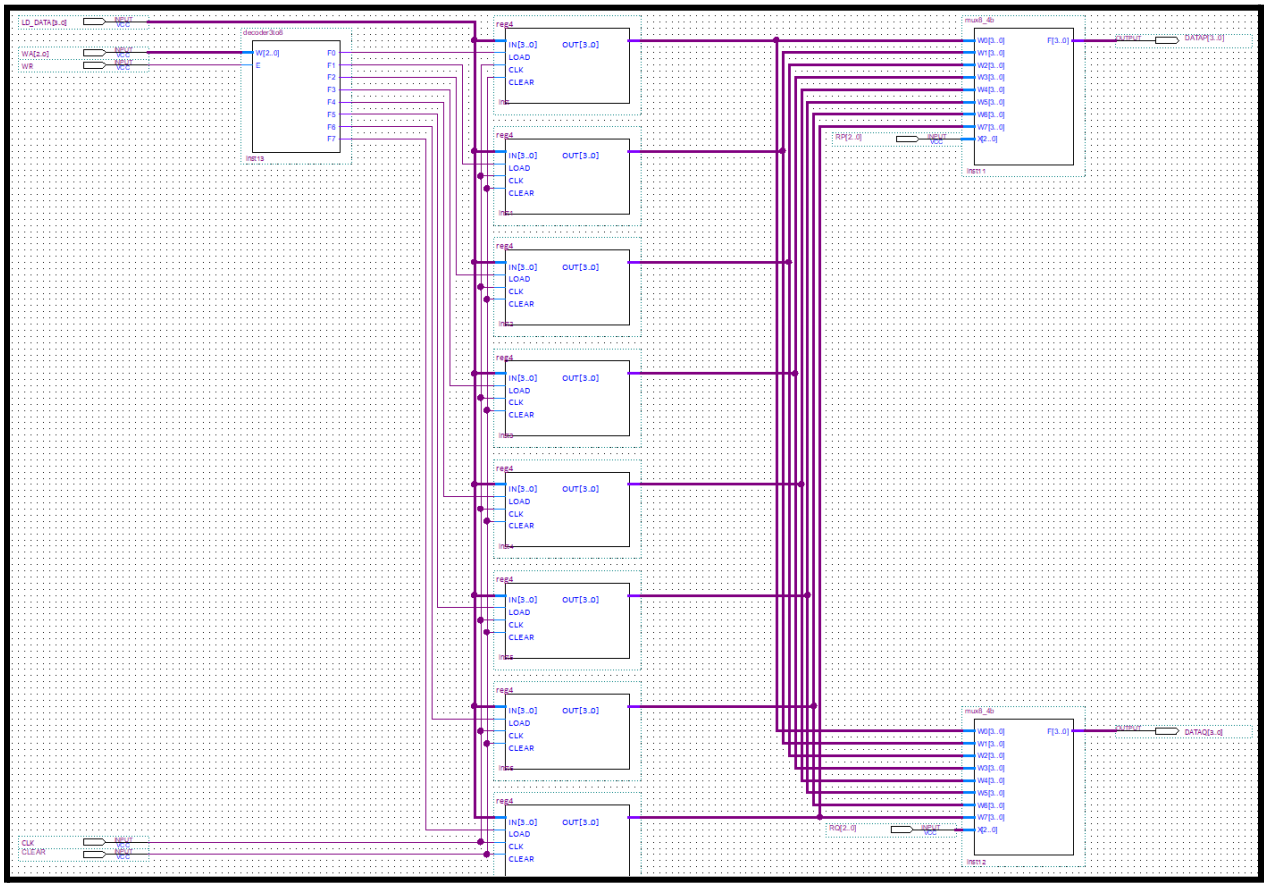
Two to One Multiplexer



The multiplexer uses a select line, S, to choose which of the two input values, D0 and D1, to pass through to the output, Q. When the select line asserts, D1 is passed through because D1's AND gate has the potential to assert and D0's can't assert. Thus, Q asserts if D1 asserts and doesn't assert if D1 doesn't assert. The same is true for D0 and its ANDgate.

Part B: Register File





Part B only has one component, a register file. Both the block diagram and circuit diagram are shown above. The circuit represents a parallel-load register file, capable of

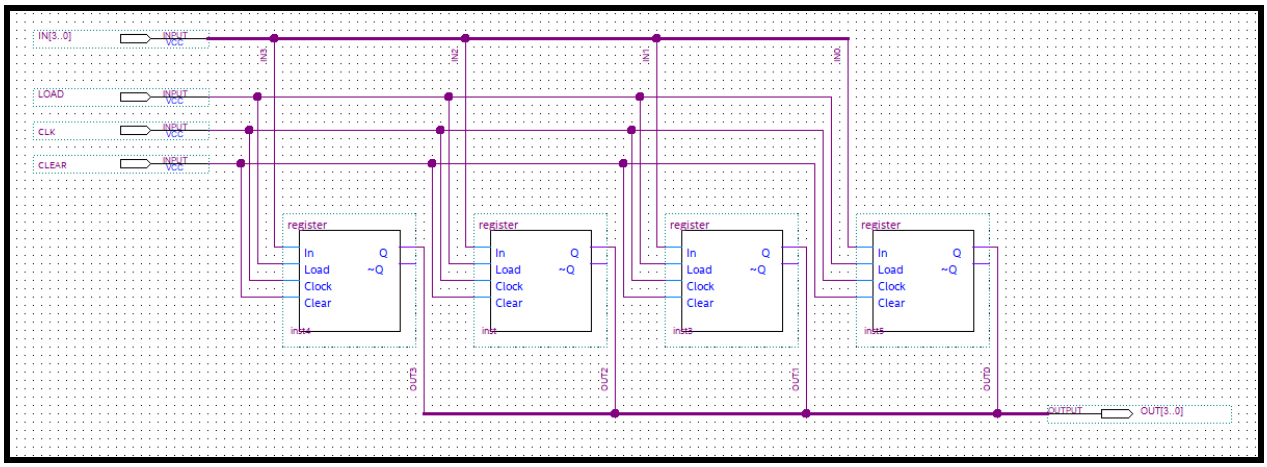
holding eight four bit numbers. The register file has one write port and two read ports. The inputs consist of a clock (update each register), clear (resets each register), WA (a three bit number that specifies which register to write to), WR (enables the circuit to write to a register), LD_Data (a four bit number that is stored in the selected register when WR asserts), RP (a three bit number that specifies which register to read from), and RQ (another three bit number that specifies which register to read from). The outputs consist of two four bit numbers, DATAP and DATAQ; both of these outputs represent the value stored in the respective registers being accessed. The register file works by using a decoder to assert the LOAD input of the desired register based on WA (on the next clock cycle). If the decoder isn't enabled, no value will be stored. The four bit eight to one multiplexers outputs the value of one of the registers. The register is chosen by either RP or RQ.

Decoder

```
 1   module decoder3to8 (W, E, F0, F1, F2,F3, F4, F5, F6, F7);
 2      input [2:0]W;
 3      input E;
 4
 5      output F0, F1, F2,F3, F4, F5, F6, F7;
 6
 7      assign F0 = E ? (W[2] ? 0 : (W[1] ? 0 : (W[0] ? 0 : 1))) : 0; // 000
 8      assign F1 = E ? (W[2] ? 0 : (W[1] ? 0 : (W[0] ? 1 : 0))) : 0; // 001
 9      assign F2 = E ? (W[2] ? 0 : (W[1] ? (W[0] ? 0 : 1) : 0)) : 0; // 010
10      assign F3 = E ? (W[2] ? 0 : (W[1] ? (W[0] ? 1 : 0) : 0)) : 0; // 011
11      assign F4 = E ? (W[2] ? (W[1] ? 0 : (W[0] ? 0 : 1)) : 0) : 0; // 100
12      assign F5 = E ? (W[2] ? (W[1] ? 0 : (W[0] ? 1 : 0)) : 0) : 0; // 101
13      assign F6 = E ? (W[2] ? (W[1] ? (W[0] ? 0 : 1) : 0) : 0) : 0; // 110
14      assign F7 = E ? (W[2] ? (W[1] ? (W[0] ? 1 : 0) : 0) : 0) : 0; // 111
15
16  endmodule
```

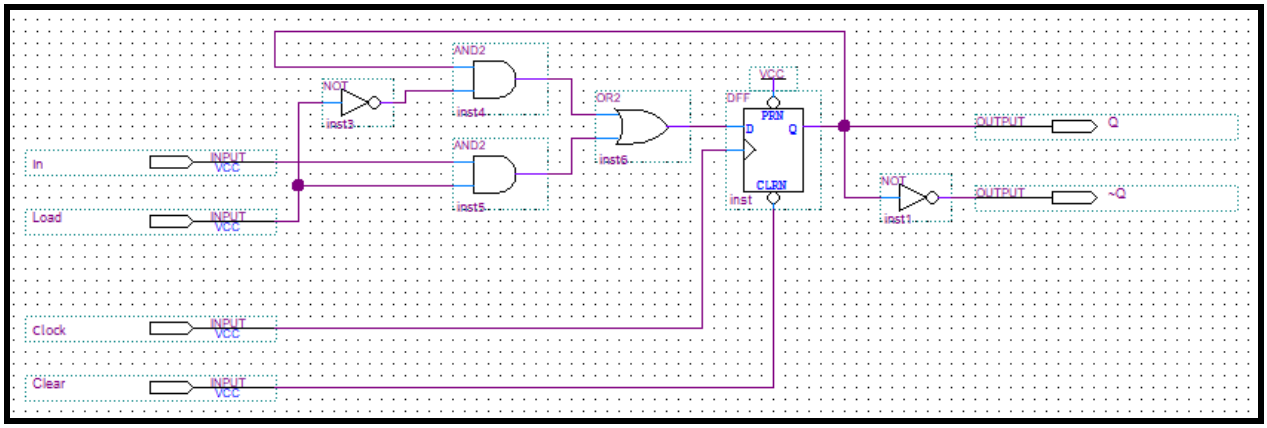
The decoder has eight statements. Each statement depends on W, a three bit number indicating which output to assert, and E, enable. If E asserts, the program will assert the corresponding output, one of F0 to F7. If E doesn't assert, no output asserts.

Four Bit Register



The four bit register is a combination of four register blocks. Each register block receives one of the bits from the four bit input, IN. When the LOAD input asserts, the registers all update on the same clock cycle. The CLEAR input sets all register blocks to 0 on the same clock cycle. Each register's output, Q, combines to make this circuit's output, OUT.

Register



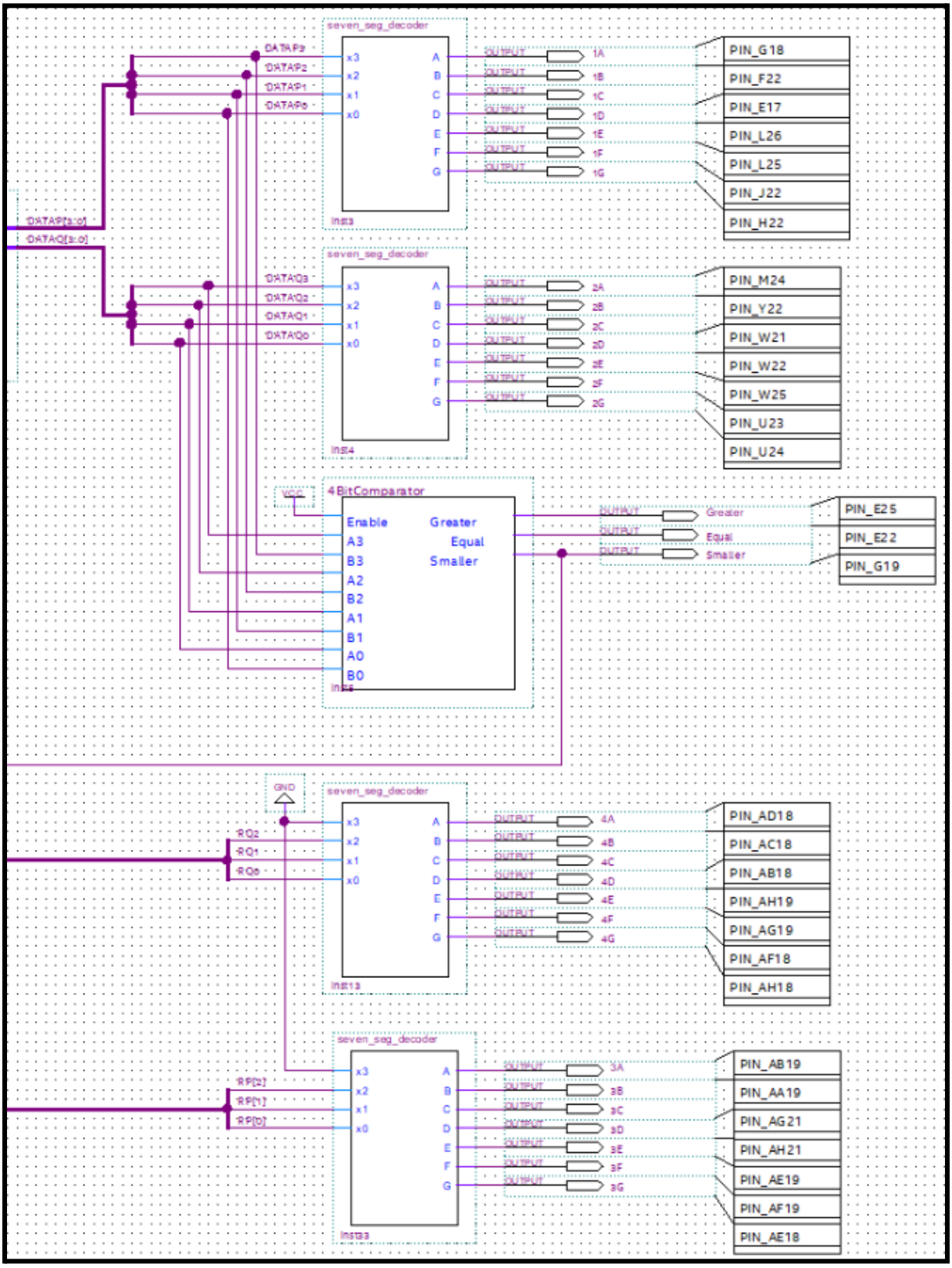
The register is the lowest level of the register file. The input bit, IN, is passed through the D flip-flop when Load asserts. When Load doesn't assert, the D flip-flop's output from the last clock cycle is kept. This is accomplished using a two to one multiplexer as described prior. However, the multiplexer circuit is implemented instead of its block symbol. The Clear symbol sets the D flip-flop to zero on the next clock cycle when enabled. Lastly, the D flip-flop's output, Q, is used as this circuit's output. An inverted version of Q is also present as an output.

Eight to One Four Bit Multiplexer

```
1   module mux8_4b(W0, W1, W2, W3, W4, W5, W6, W7, X, F);
2      input [2:0] X;
3      input [3:0] W0, W1, W2, W3, W4, W5, W6, W7;
4      output [3:0] F;
5
6      assign F = X[2] ? (X[1] ? (X[0] ? W7 : W6) : (X[0] ? W5 : W4)) : (X[1] ? (X[0] ? W3 : W2) : (X[0] ? W1 : W0));
7   endmodule
```

The eight to one multiplexer works in a similar way to the decoder. However, it passes a four bit input, W0 through W7, to the output, F. It chooses which input to output based on X, a three bit input, to choose the respective W.

Part C

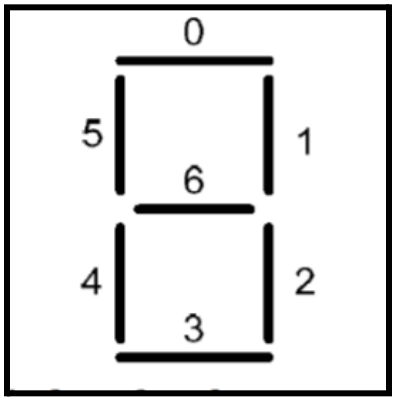

seven_seg_decoder
DATAP[3:0]
DATAQ[3:0]
Inst3
Inst4
Inst13
Inst33
4BitComparator
Enable
A3
B3
A2
B2
A1
B1
A0
B0
Greater
Equal
Smaller
VCC
GND
PIN_G18
PIN_F22
PIN_E17
PIN_L26
PIN_L25
PIN_J22
PIN_H22
PIN_M24
PIN_Y22
PIN_W21
PIN_W22
PIN_W25
PIN_U23
PIN_U24
PIN_E25
PIN_E22
PIN_G19
PIN_AD18
PIN_AC18
PIN_AB18
PIN_AH19
PIN_AG19
PIN_AF18
PIN_AH18
PIN_AB19
PIN_AA19
PIN_AG21
PIN_AH21
PIN_AE19
PIN_AF19
PIN_AE18

Part C has two purposes: interpret the date from the register file, and output the data from the circuit. To interpret, Part C uses a comparator. The comparator compares the two four bit numbers being accessed in the register file. DATAP, the data being stored in the highest index being compared, and DATAQ, the data being stored in the lowest index being compared, are interpreted and displayed by the seven segment decoder. The interpreted version of DATAP and DATAQ assert collections of output pins, 1A through 1G and 2A through 2G. DATAP and DATAQ also are interpreted by the comparator. The comparator checks the relationship between DATAP and DATAQ, and asserts one of the related output pins (Greater, Equal, or Smaller). In a similar way to how DATAP and DATAQ are displayed, the address of DATAP and DATAQ are shown on the board.

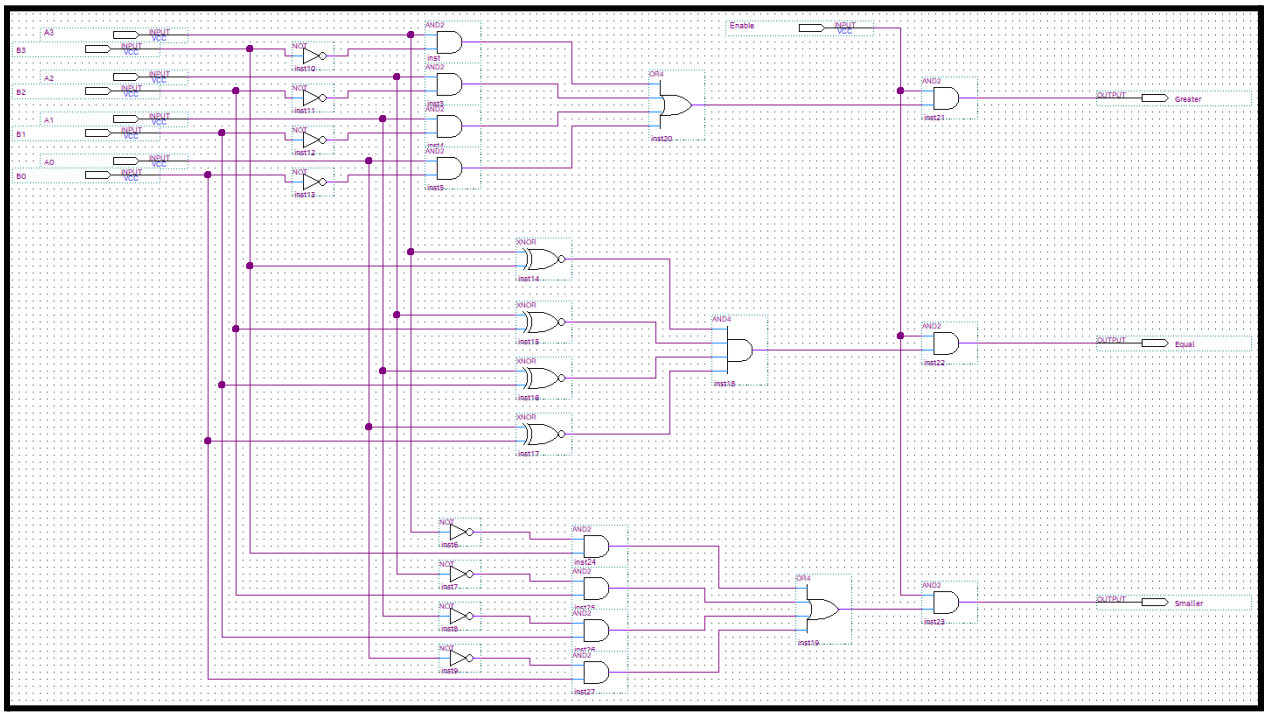
Seven Segment Decoder

```
1   module seven_seg_decoder (x3,x2,x1,x0,A,B,C,D,E,F,G);
2      input x3,x2,x1,x0;
3      output A,B,C,D,E,F,G;
4      assign A = (~x3&~x2&~x1&x0)|(~x3&x2&~x1&~x0)|(x3&x2&~x1&x0)|(x3&~x2&x1&x0);
5      assign B = (~x3&x2&~x1&x0)|(x3&x2&~x1&~x0)|(x3&x1&x0)|(x2&x1&~x0);
6      assign C = (~x3&~x2&x1&~x0)|(x3&x2&~x1&~x0)|(x3&x2&x1);
7      assign D = (~x3&~x2&~x1&x0)|(~x3&x2&~x1&~x0)|(x2&x1&x0)|(x3&~x2&x1&~x0);
8      assign E = (~x3&x0)|(~x3&x2&~x1)|(~x2&~x1&x0);
9      assign F = (~x3&~x2&x0)|(~x3&x1&x0)|(~x3&~x2&x1)|(x3&x2&~x1&x0);
10     assign G = (~x3&~x2&~x1)|(~x3&x2&x1&x0)|(x3&x2&~x1&~x0);
11  endmodule
```

The seven segment decoder asserts one of the seven outputs, A through G, depending on the four bit input number, x0 through x3. Which output asserts depends on the physical location of the light on the display, shown on the diagram below (A = 0, B = 1, … G = 6). Each output is asserted based on the hexadecimal representation (ranging from 0 to F) of the input.



Four Bit Comparator



The four bit comparator compares a high number, B0 through B3, to a low number, A0 through A3, and determines the relationship when Enable is high. If B is larger, Greater asserts; If If A is larger, Smaller asserts; and if they are the same value, Equal asserts. To accomplish this, the circuit has three parts which are clearly separated in the circuit diagram. Using XNOR gates outputting to an AND, Equal asserts when the nth bit of A is the same as the nth bit of B for n zero, one, two, and three. If the AND gate asserts and Enable is high, the equal pin asserts. The Greater pin asserts when B's equivalent value is greater than A's. To accomplish this, the circuit checks for any instance when B has a one in the index that A has a zero. If this occurs and Enable is high, Greater asserts. Smaller is nearly identical to Greater, but checks for instances when B has a zero and A has a one.