Jessica Kuo and Caitlyn Svitek

Digital Logic Design

Professor Meghana Jain

December 14, 2018

Final Project Documentation

Component Descriptions

Our calculator takes two 8-bit numbers and either adds, subtracts, or multiplies them. The result is then displayed on a seven segment display. Inputs are created by the user flipping a series of switches. They can flip up to 8 switches, each one representing a bit. Since the switches can only be flipped up or down, calculations are performed with binary numbers; outputs are displayed in hexadecimal. After operations are performed, there are 16 output pins due to calculations being done in binary, and after the result is generated shall the binary number be converted to hexadecimal and displayed on the seven segment display.

In order to save the necessary input, an 8-bit register is used. The register saves the first number input, but then also saves the operation, hence all three operations are put through an OR gate and lead into the register. The inputs are given synchronizers to reduce glitches. The second value is also put through synchronizers and an 8-bit register, although the operations are not present because they're already saved from entering the first number. The numbers are saved to the second register once the 'equals' button, represented by the left button, is pushed.

Each operation is given an operation code. Hold, or wait, is opcode 00, as nothing is being pressed, addition is opcode 01, subtraction is opcode 10, and multiplication is opcode 11. All opcodes are generated using an ALU that is composed of sixteen 4-to-1 multiplexers connected to buses, and each operation has specific gates to spawn the correct opcode bits to the select bits.

Addition is done with an 8-bit adder, created by linking together 8 full adders and using busses for inputs and outputs. Each adder is made of an XOR gate with inputs A, B, and C to determine the solution out bit and three AND gates going into an OR gate to find the carry out bit. Subtraction was designed similarly, with 8 full subtractors and three busses. The inputs for both these components were A(7:0) and B(7:0), and the output was D(8:0) since two 8-bit inputs produce a 9-bit result.

Multiplication, a more complex operation, is done with a series of cascading full and half adders. Multiplication involves finding partial products for every combination of bits, and then adding those partial products together to reach a final result. This is done by putting pairs of one A and one B input through an AND gate, and then each pair is put through a number of full and half adders. Since the bits are being entered in binary, the AND gate is able to calculate the

partial product because the only inputs that would result in a partial product of 1 would be 1 AND 1, as it would be with multiplication done by hand. The pairs of bits entering each adder can be calculated similarly to when numbers are shifted and 0's are added to the end of partial products in hand multiplication. There too, are cascading outputs where each partial product is added to the partial product below it, but that next product's line of outputs is shifted to the right by one position. Additionally, since the number of bits that need to be added together increases for each position the user is calculating, in our schematic, the number of adders must increase as well. This process continues until S7, when the number of adders being used starts to decrease, as seen in regular multiplication when the user runs out of numbers to add until they reach position S15, where the position is simply the carry out bit from S14.

After the desired operation has been performed, the result is sent through one of three 16-bit busses. Although for addition and subtraction, bits 9 to 15 are sent to ground pins because they don't need more than 9 bits to perform all ranges of operations. The busses are connected to an ALU, which we designed using 16 regular 4-to-1 muxes. This was so we would be able to send entire busses through the mux rather than one input at a time. After being sent through the mux, the result would be sent through the seven segment display driver, waiting for the user to press the 'equals' button to display.

The primary component for displaying the result is the seven segment display driver. The driver divides the greatest possible result, a 16 bit number, into groups of 4 bits, separated by their positions on the display. Bits 0 through 3 are on the rightmost position, bits 4 through 7 in the right-center, bits 8-11 in the left-center, and bits 12-15 in the leftmost position using a bus. Additionally, the final result only displays when the user presses the 'equals' button, signified by the left directional push button. This was done by attaching an SR Latch to the converter also known as *segsev*. Each time an operation that wasn't 'equals' was pressed, it would turn the reset on, which would turn on every segment on the display, and when 'equals' was pressed, the reset would turn off and enable would be on, allowing the result to display. Although before being put through the latch, we also created a component that would convert the binary result that had entered the driver to a hexadecimal value due to the limited positions on the display. Lastly, the outputs were mapped to the seven segment display pins, and would display the result.

Diagrams and Schematics

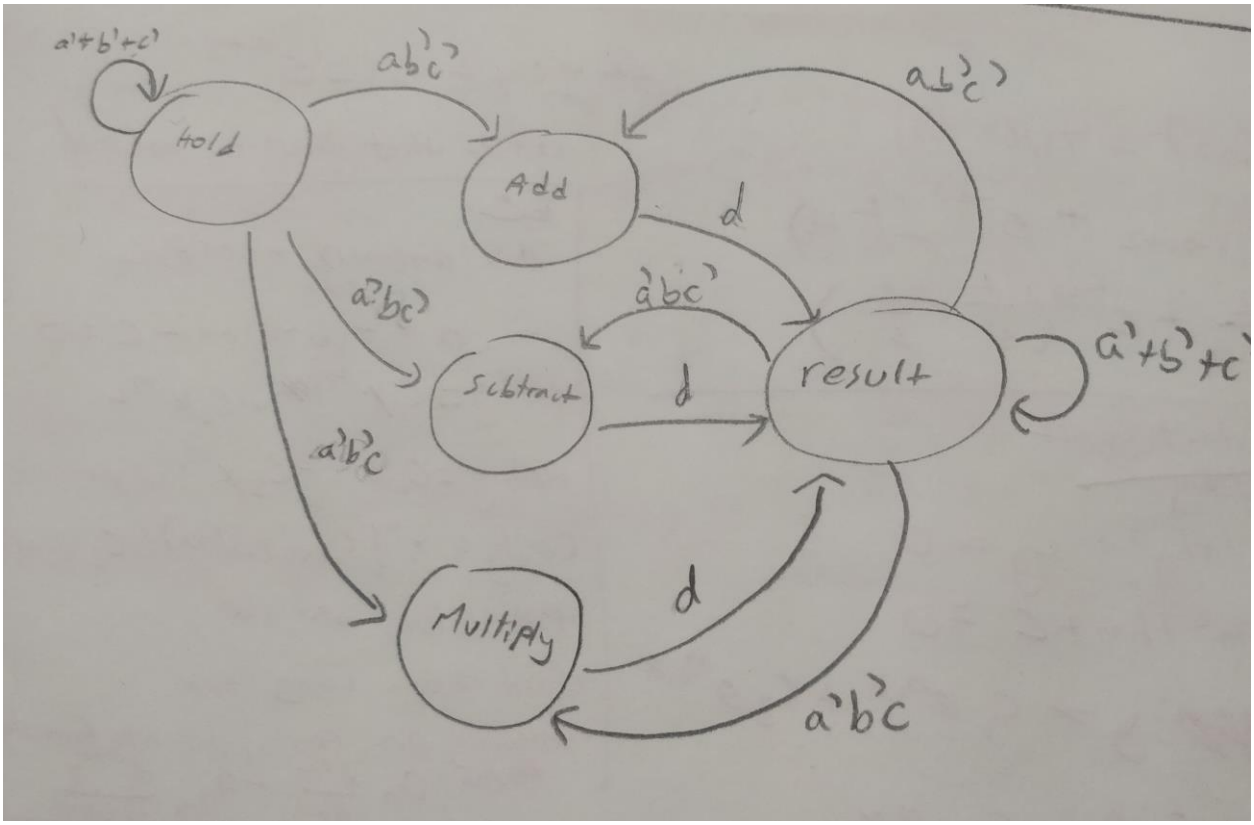

Figure 1: Initial Calculator FSM

Testing and Conclusion

The calculator was tested by performing a series of tests for each operation. Each operation was performed using several sets of same bit numbers and different numbers. First, to make sure the registers were properly saving numbers, we connected an 8-bit register to 8 switches and the output to each of the 8 respective LEDs. The test was that when any switch went high, the LED in the same position would turn on once you pressed a button to save the value.

For choosing an operation, we tested the multiplexer by mapping the select bits to the direction buttons and logic gates. The test was run four times, with each time it was run a vcc would be connected to a different path for the multiplexer to operate. Then the output was mapped to an LED. When testing, all of the buttons were pressed one at a time to make sure that the LED would only light up for one button. This verifies that only one button would be able to access each operation in the calculator.

To prove that each operation was working, we would load the schematic for the 8-bit adder, 8-bit subtractor, and 8-bit multiplier to the Papilio board and test it with several equations

of varying values and bit-length. Although we had not tested the display yet, and at this point the result was being displayed as switches were being moved for the second number.

Now that each operation was complete, for testing the seven segment driver we used an 8-bit register, 8 switches, and a button to display them. The driver and decoder were added along with the gates needed to turn each segment on the display on when it was necessary for each hexadecimal value. The values would be determined with the switches and displayed on the left half of the display. By pressing the button, the value would be saved and displayed on the right half of the display. Doing this allowed us to make sure that values could be stored and displayed on a seven segment display at the push of a button.

The next test was to have the seven segment display only display the result when the ‘equals’ button was pressed, rather than having it display results constantly as one would be entering the second number; which would be confusing to the user. To do this, we used an SR Latch. We mapped the output to an LED, which would turn on when ‘equals’ was pressed, and turn off when one of the other three directional buttons were pressed. Once the test was valid, we put the memory element back in the main calculator file along with the operation and display components.

Finally, with the operations and display in order, we combined them to make sure they’d work together. We tested several equations for each operation before being assured that the calculator was finished.