**Final Project**

**Implementation of Calculator design using FPGA**

Section 005

Sana Hasan

hd9402

Hussein El Hajj

hf7139

Table of contents

[Abstract 3](#_Toc89770014)

[Introduction 4](#_Toc89770015)

[Design 5](#_Toc89770016)

[State diagram explanation: 7](#_Toc89770017)

[FSM source file: 8](#_Toc89770018)

[Counter code file: 8](#_Toc89770019)

[Display decoder for the 7-segment unit: 9](#_Toc89770020)

[Top level module: 9](#_Toc89770021)

[Results 11](#_Toc89770022)

[Simulation: 11](#_Toc89770023)

[Constraints: 12](#_Toc89770024)

[Hardware Implementation: 13](#_Toc89770025)

[Conclusion 16](#_Toc89770026)

[Appendix 1 17](#_Toc89770027)

[Appendix 2 18](#_Toc89770028)

Abstract   
In this lab our purpose was to make a calculator implemented on the FPGA board. Using the push buttons as number inputs, the switches as a reset and for operators, and the 7-segment display to view the operands and the answer, we implemented this calculator. The procedures were to make an FSM, and use a counter and display decoder, and a top-level module that instantiates each of these codes to connect them and implement to the FPGA. We got satisfactory results and a working calculator, including fixing the result to be decimal like a normal calculator.

Introduction

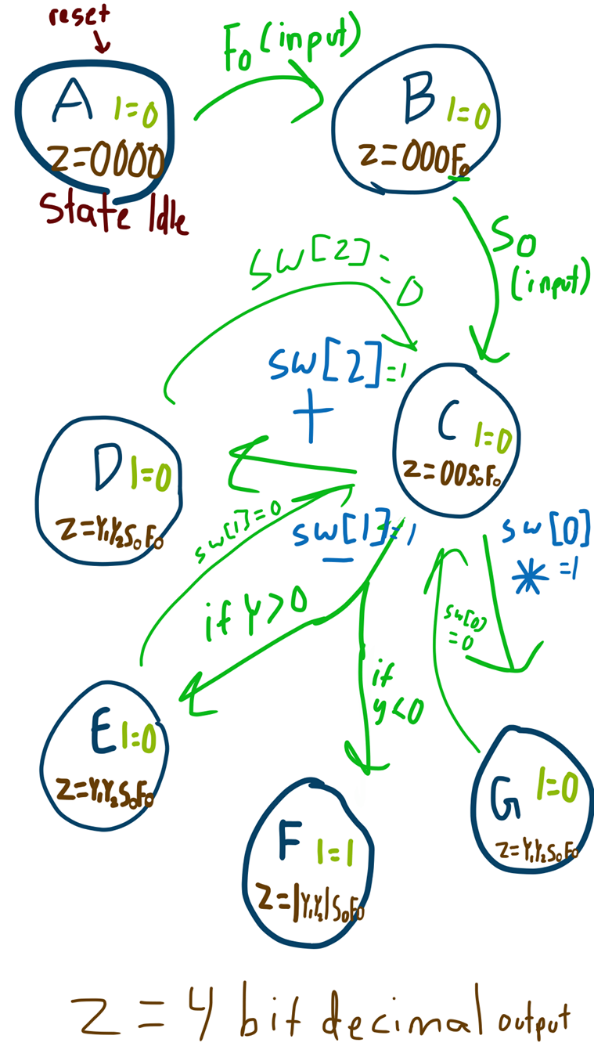
In this lab, we undertook the creation of a working calculator of the three operations: subtraction, addition, multiplication. To do this we created and designed a Finite State Machine (fsm) in the FPGA board. In this final project, we have created a source file that includes all the states that were mentioned in our state diagram (see results for this diagram) and needed to program our FPGA board.

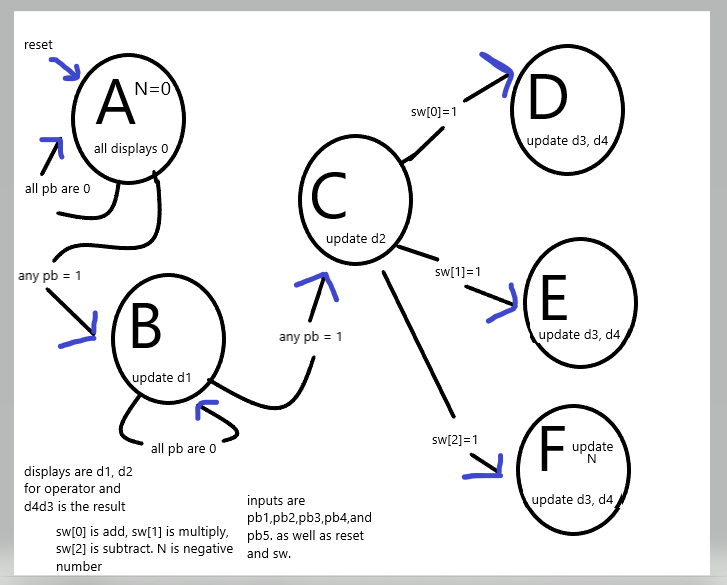
The first state is A known as the idle state, the second state and third state are respectively B and C known as selection and display of our desired inputs. Lastly, we created 3 other states, D, E, and F so we can implement three operations such as addition, multiplication, and subtraction.

We then implemented into the FPGA board and got results that were good for the basic calculator. The initial result was a hexadecimal number, and we only used an LED to indicate negative number. We then fixed display 4 so that it would show a minus if negative and fixed the FSM code so the result would be a decimal number like a normal calculator.

# Design

For the design of our calculator, the original state diagram is below along with the edited one we went with. The only real difference between these state diagrams is that we went with 1 state for the subtraction state.





## State diagram explanation:

Output N is an led, 0 in every state except F. If the subtraction yields a result less than 0, it will be 1 to indicate a negative number. the displays are the outputs d1,d2,d3,d4, indicating a 7-segment display.

A = idle state, waiting for inputs. achieved by setting input reset to 1 output: 7-segment display says 0000 (each display indicates what is shown on the 7-segment display) input: reset, waiting on input from a pb to progress to B.

(in the old diagram fo meant first operator which was any push button, so was second operator, I was N, and the displays were indicated by Z.)

B= displaying first operator on d1 as inputted by a pb.[1], the rest say 0, waiting on input so to proceed to C output is updating display 2 or d2. In the final code, another switch input =1 is needed to be able to take another pb input and go to C.

C= displaying d1 and d2 for the operators as inputted. D3 and d4 still have 0, waiting on an operator from sw to proceed to corresponding state. Input is one switch out of 3 inputs sw[2:0].

We then do the operation and go to the corresponding state. In the original, we needed to know the result to pick the subtraction state but in the final diagram the result is not needed until the corresponding state.

D= addition. d1 and d2 are added, and the sum is displayed on d4 and d3. d4 displays the msb, and d3 displays the lsb.

E= multiplication. d1 and d2 are multiplied, and the product is displayed on d4 and d3. d4 displays the msb, and d3 displays the lsb.

F= subtraction. The result is d2-d1, and the product is displayed on d4 and d3. Any subtraction will be within 4 bits, so d3 shows the result. If the answer is less than 1 d4 is updated with the signal for a minus sign, and N is updated to 1, and d3 shows the absolute value.

the input reset brings it back to the idle state A no matter what state the FSM is in.

## FSM source file:

*See Appendix 2, as the length of the code is too large and will break up the flow of the description.*

We used the FSM to code these states and how we get from one state to another. We use the <= to indicate a non-blocking statement when switching states.

Although the method used for converting to decimal is not efficient, as it is hard coding, there were only a handful of numbers needed to be changed so hard coding was easiest. A simple method to change this would be a statement saying (pseudocode)

*if the result is less than 19 and greater than 9, d3 is result – 10, and d4 is 1. else if the result is less than 29 and greater than 19, d4 is 2 and d3 is d-20. Else d3 is result[3:0] and d4 is result[7:4].*

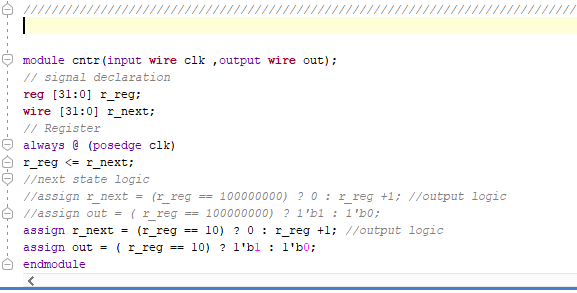
Using a case switch design for which operands and which operation was picked, we then went to the appropriate state for the operation and performed the operation.

Each operation state has the code to properly display the types of results it will yield. We simply used Verilog to multiply and add normally.

For subtraction, we did the second operation minus the first operand using the twos complement method. Finding the 2’s complement of the first operand and adding this to operand two. Then displaying the 2’s complement of the result if it was less than zero and outputting the LED signal and the signal for the minus sign.

## Counter code file:

We used this counter to lower the clock from 50 MHz to 10 Hz so the FPGA and display would work as intended



## Display decoder for the 7-segment unit:

We used this file to convert the result (which is a 4-bit hexadecimal number) into a viewable number using the 7-segment display. There is also code here that allows the negative sign to be visible if the 4 hexadecimal digits for e are in the code for display 4 (which would never be needed to be displayed according to our inputs and operations).

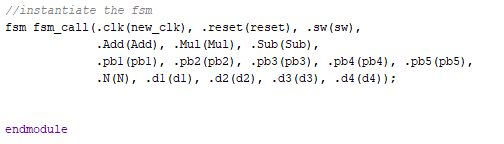
*See Appendix 2, as the length of the code is too large and will break up the flow of the description.*

The ‘an’ variable is for the anodes of the 7-segment display that will only turn on/be affected when there is an active-low signal.

## Top level module:

This module simply instantiates the rest of the files to allow the counter to be applied to the clock, the display to be usable to show the results, and the FSM code for the operations of the calculator finite state machine.



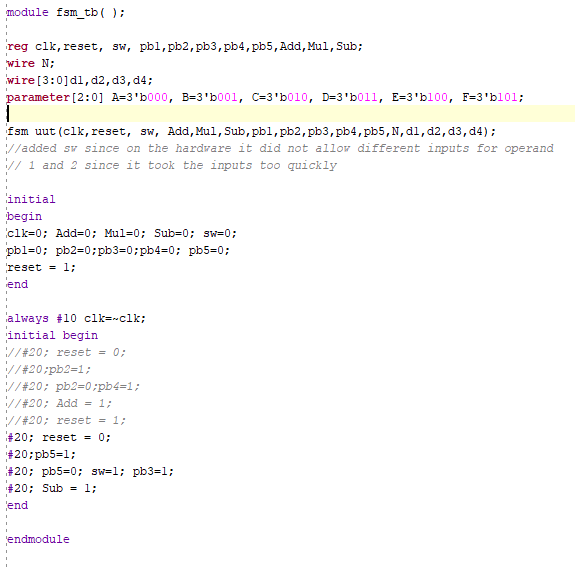


# Results

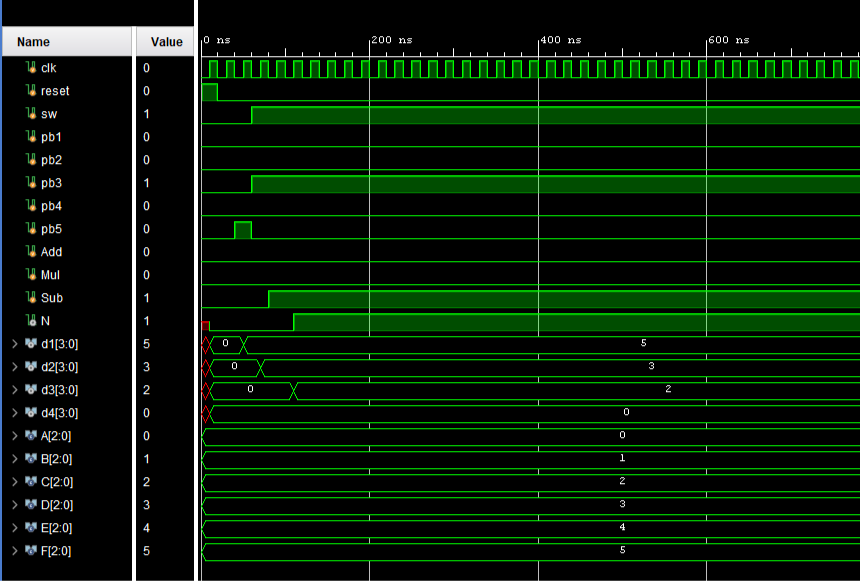
First, we have created the source file for our FSM. Second, we tested our code through our test bench file to verify that our FSM worked as we planned. Third, we checked and verified that every input and operation results were accurate in our timing diagram.

## Simulation:

Test bench



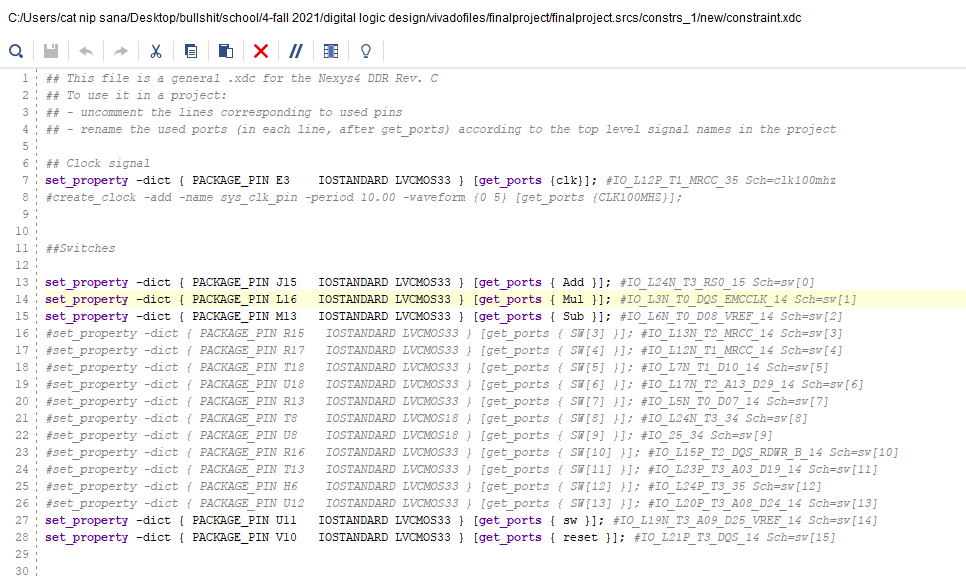
Timing diagram:

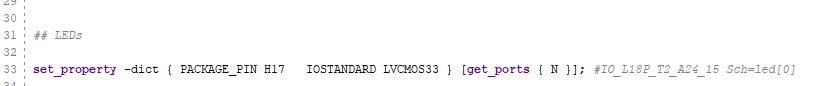


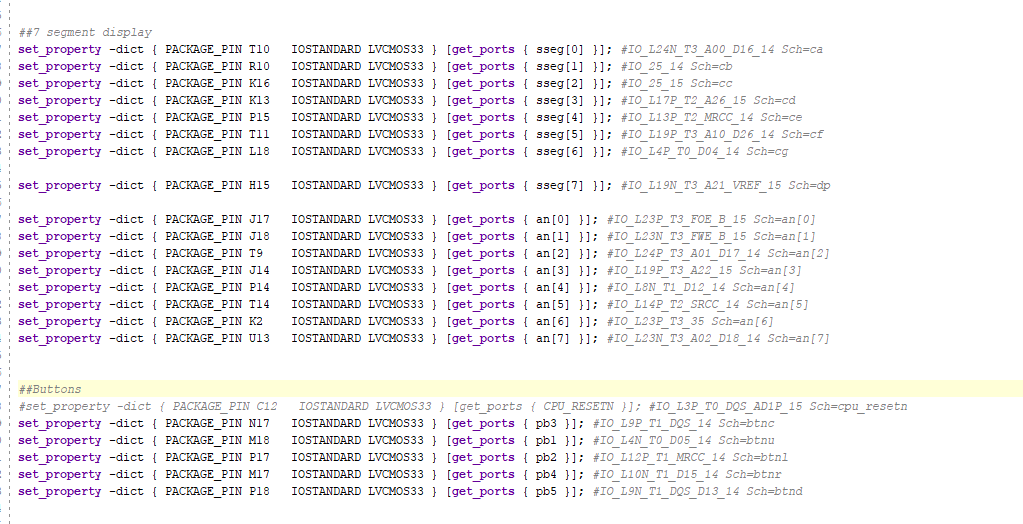
We did this process for different numbers and operations being selected to make sure that they all worked as planned

Lastly, once these were verified, we edited the constraints file to implement it on our FPGA.

## Constraints:



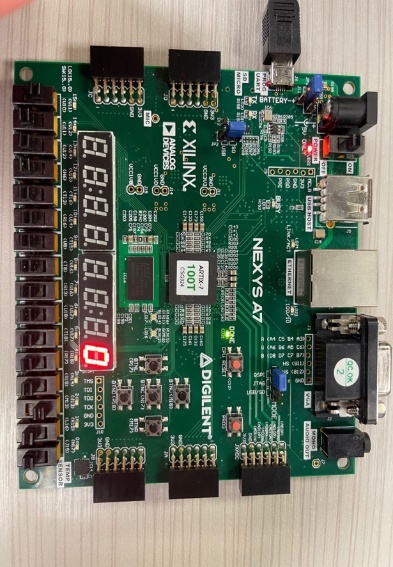




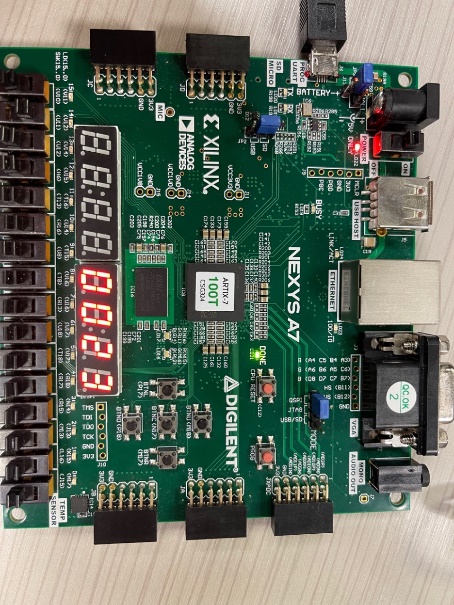
To implement, we have to run through synthesis, implementation, and generate a bitstream.

## Hardware Implementation:

To display our results on the FPGA board screen, we must pass through certain steps. See Appendix 1 for a link to a video. First, we turn on switch number 16 to make sure we are in the idle state(A).

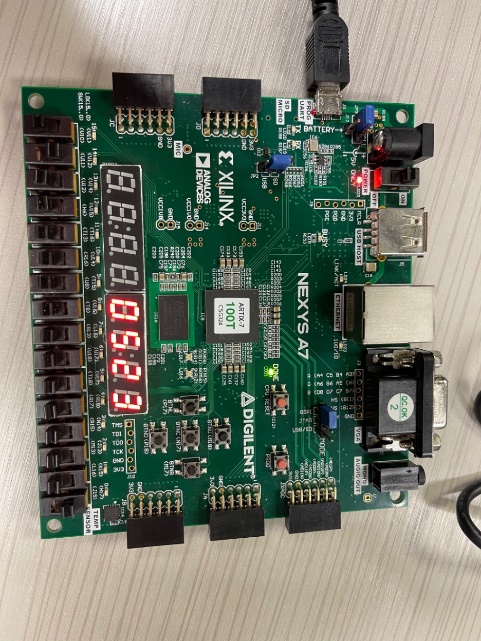


Second, turn off that reset switch and pick the number we wanted to use for the operand in disp1. Then we turn switch number 15 (farthest on left) on, so the board knows that we are choosing a second number (disp2).

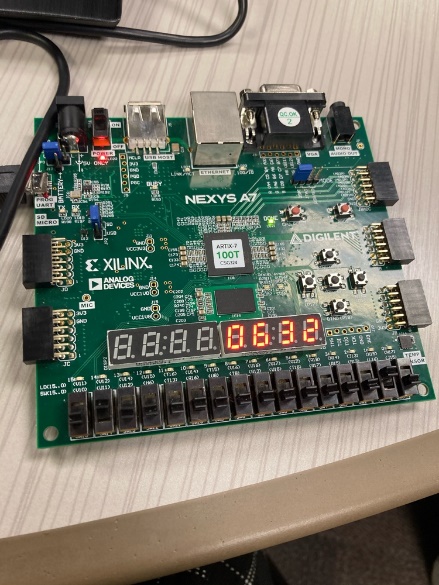


Third, we pick our desired operation either addition, multiplication, or subtraction by respectively turning on one of the switches; 1,2, or 3. 1 is farthest on the right, 3 is third farthest. The answer shows on disp 4 and 3.

Addition:



Multiplication:



There is an exception for the subtraction part, if the first number is smaller than the subtracted one, we would have the answer appear in disp3 as an absolute value, and a negative sign next to it in disp4 so the user can clearly notice the answer being negative. Plus, an LED light will turn on above switch number 1.



# Conclusion

To conclude, we created an FSM source file, counter code, decoder for the display unit, top level module, and an edited constraint file. We finally got our FPGA working and have tested all the states and operations that have been coded in those files above. Plus, we made sure that our timing diagram results were accurate and correct for the FSM before implementing.

Additionally, we have also edited the decoder file using an if statement in the 7-segment display method to let the negative sign appear when it needed to. We have also allowed the result to be in decimal format

# Appendix 1

Supporting materials

We believe that access to the code files to be able to test the code is necessary in this report for this final project.

[link to code files](https://waynestateprod-my.sharepoint.com/:u:/g/personal/hd9402_wayne_edu/EUZCOAVsf0JDui83bOupPUEBY6wFE_wAxWiBjzC09Ortdw?e=Aofsgs)

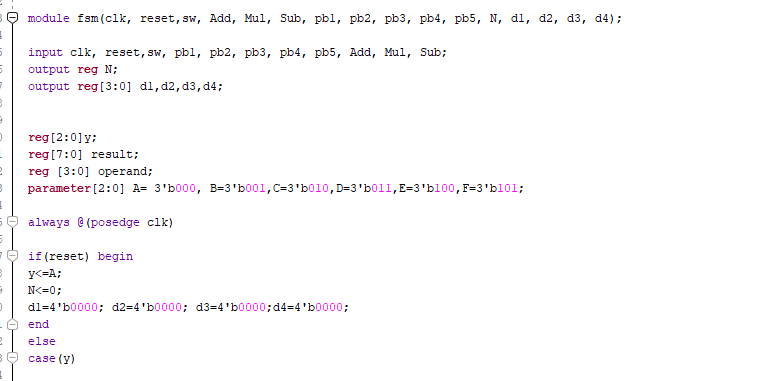
Additionally, here is the results video mentioned in the results section.

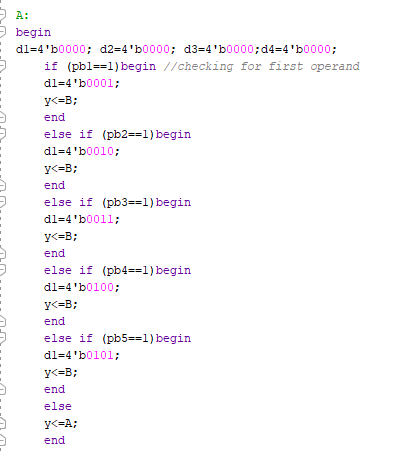
[link to a video displaying the results](https://waynestateprod-my.sharepoint.com/:v:/g/personal/hd9402_wayne_edu/EWfYVpshl7BOkaSBWIQNj6ABeGho9jTVYzxzPO3X1r2pRw?e=8dIU7L)

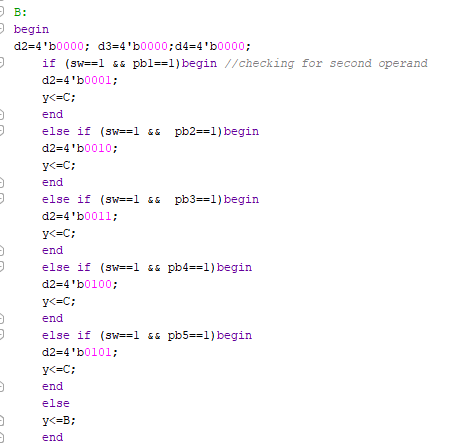
# Appendix 2

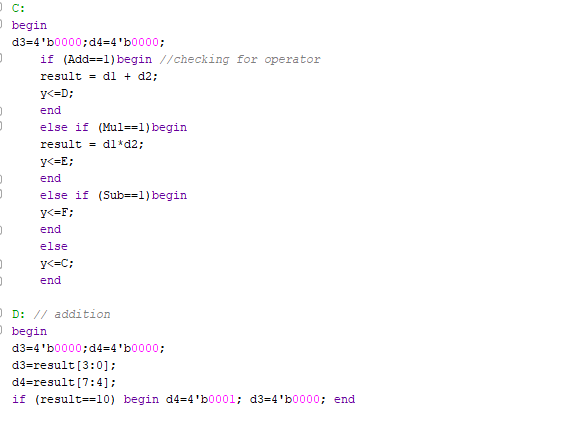
FSM code

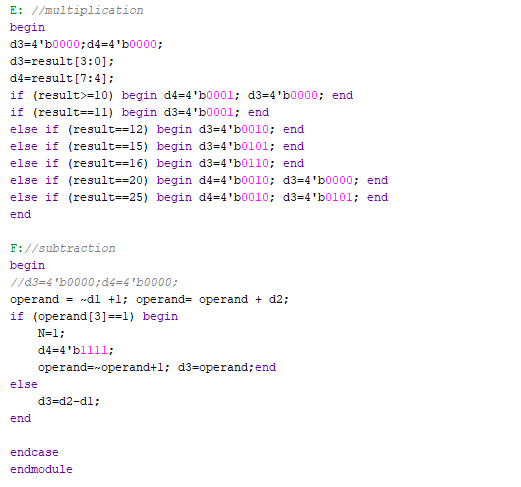
screenshots











Disp Hex Mux code

screenshots

