5-bit Signed Calculator with Switches

VHDL implemented calculator

Adam Kidwell, Braun Mayette, Christopher Gibson, Melad Haddad
Electrical and Computer Engineering Department
School of Engineering and Computer Science
Oakland University, Rochester, MI
e-mails: bkidwell@oakland.edu, bmayette@oakland.edu, clgibson@oakland.edu, meladjoseph@oakland.edu

***Abstract*—We will create a calculator with the ability to add, subtract, multiply, and divide two five-bit unsigned numbers: the purpose is to create an easy switch operated way to compute numbers. We will use a 7-segment serializer to display the results on four 7-segment displays. The 7-segment display will create an easy way to show the data for the user.**

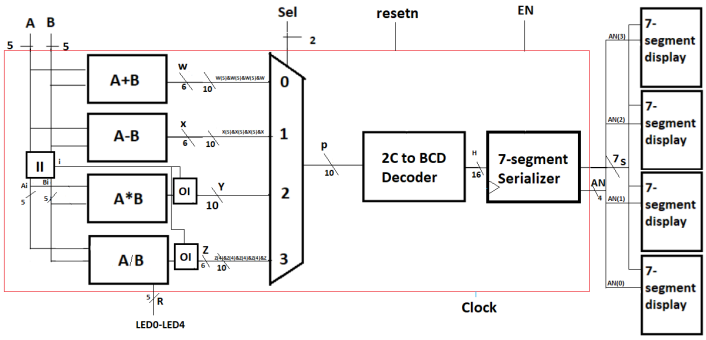


Figure 1

I. Introduction

For our project we will be designing a digital system through the Vivado software that will be able to add, subtract, multiply and divide two five-bit unsigned numbers. The configuration of this system will require coding, troubleshooting, and implementation. The purpose of our project is to provide a user with an easy to use calculator that can perform simple calculations. From this project we will be able to increase our knowledge of how switches and a seven-segment display interact with each other. Specifically our project will use our knowledge of the simple Arithmetic Logic Unit (ALU), and the decoder that we learned in lab four, we will also change the amount of bits from four to five. We have also added our knowledge of the Iterative Divider Implementation that we learned in lab six. Time management and teamwork will play an important role in the creation of our project. Together we will develop a system that individuals can use on a daily basis to help with simple mathematical calculations.

II. Methodology

The main components of our circuit are the 6-bit adder, the 6-bit subtractor, the 5-bit x 5-bit multiplier, the 5-bit / 5-bit divider, the input and output inverters, the operation selecting MUX, the 2C-to-BCD Decoder, the 7-segment serializer, and, of course, the topfile and xdc file.

A. 6-bit Adder - Melad Haddad

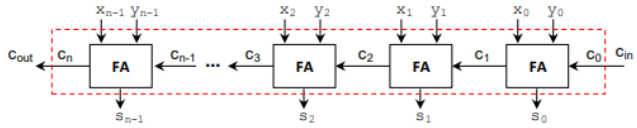


Figure 2

The 6-bit signed adder is a simple combinational circuit where the inputs are A and B (both 5 bits sign extended to 6-bits to avoid overflow). With the use of six full-adders with signals Cn-1 will be able to produce signal S 5 downto 0. This will result in the sum of two numbers either positive or negative.

B. 6-bit Subtractor - Braun Mayette

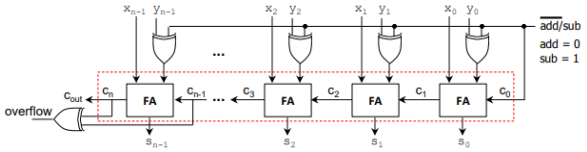


Figure 3

The 6-bit signed subtractor is a simple combinational circuit where the inputs are A and B (both 5 bits sign extended to 6-bit to avoid overflow). With the use of six full-adders with signals C 6 downto 0 will be able to produce signal S 5 downto 0. This will result in the difference of two numbers either positive or negative.

C. 5-bit x 5-bit Multiplier - Christopher Gibson

The multiplier circuit takes in two 5-bit inputs and outputs the product in the form of a 10-bit answer. Originally we planned for the multiplier to work with signed numbers, so 10 bits were necessary to encompass all of our outputs. In the end the multiplier was changed to only work with unsigned numbers, so it would have been acceptable to have a 9-bit output. The code we used followed a lecture for an n-bit unsigned multiplier posted in Unit 7. The elements included in this circuit are a register, a parallel access shift register, an adder and a fsm. The adder element controls the arithmetic operation through decimal transitions and

replication of inputs. The FSM controls the operation of the circuit through constant checks within each state. The block diagram for the circuit can be seen in Figure 4, the FSM state diagram can be seen in Figure 5, and the algorithm that describes how the component works can be seen in Figure 6. Images courtesy of Professor Llamocca's Unit 7 notes.

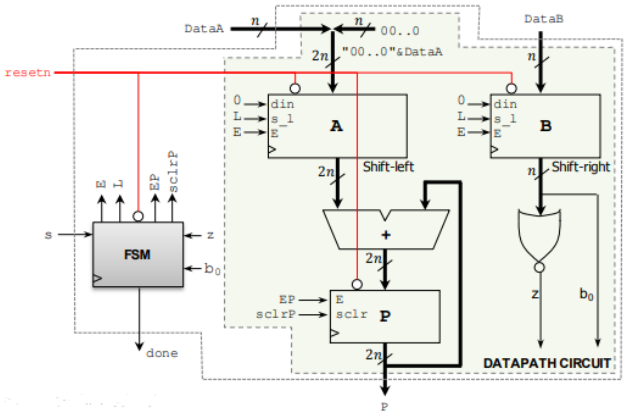


Figure 4

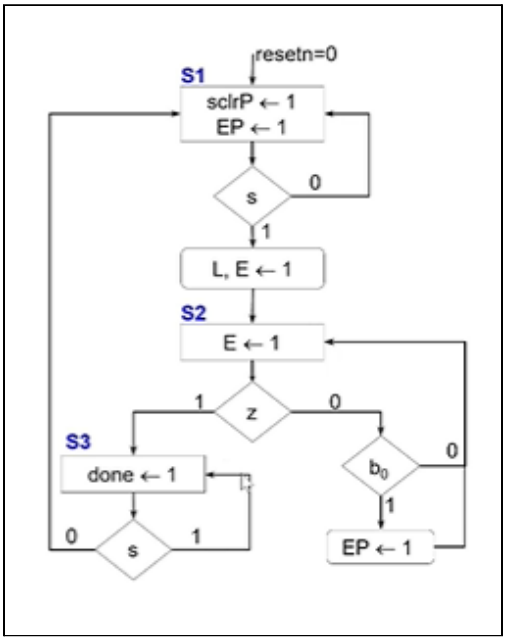


Figure 5

```
P ← 0, Load A,B
while B ≠ 0
   if b0 = 1 then
      P ← P + A
   end if
   left shift A
   right shift B
end while
```

Figure 6

D. 5-bit / 5-bit Divider - Adam Kidwell

Our divider circuit takes in two 5-bit inputs and outputs a 6-bit quotient with a 5-bit remainder. It works as long division works, going digit by digit until there is a remainder smaller than the denominator. It follows the algorithm shown in figure 8, under our block diagram in Figure 7. R is stored in the bottom right parallel access shift register, i is the number of bits of input A, R>B is computed through the carryout of the fulladder, the subtraction R-B is computed through the fulladder, and the bit qi is simply the carryout of the fulladder. A FSM controls the enables and selects of each of the shift registers to allow them to act appropriately. Our code is a modified version of the code we created for lab 6. The code was modified to fit the bit size of the inputs, and also added a MUX at the output that outputs "000000" if the input B is "00000". This modification prevents errors that may occur when dividing by 0. One of the main difficulties of implementing this divider for our project stemmed from the fact that this divider only works with unsigned numbers. To circumvent this problem, we designed two additional blocks: input inverter and output inverter. Section E contains additional details about these blocks.

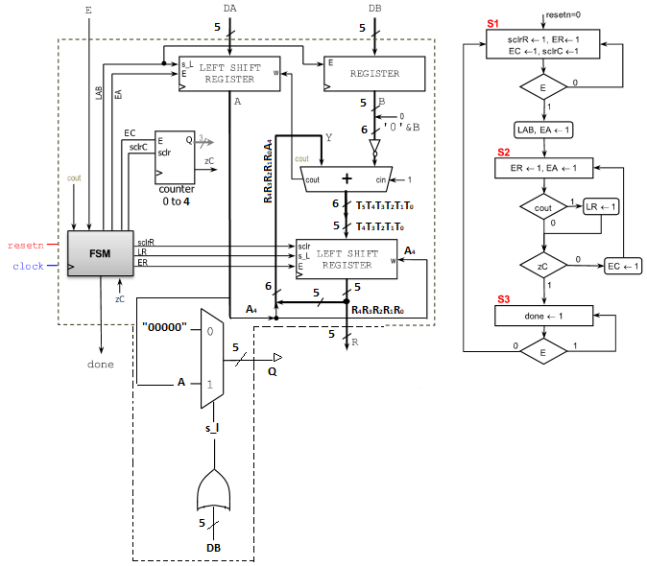
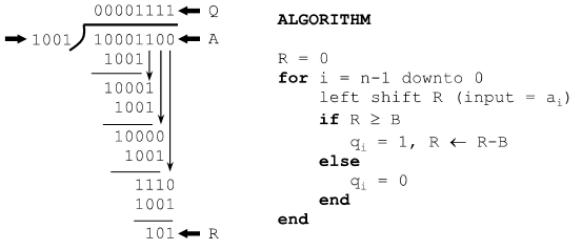


Figure 7




```
ALGORITHM

R = 0
for i = n-1 downto 0
    left shift R (input = ai)
    if R ≥ B
       qi = 1, R ← R-B
    else
       qi = 0
    end
end
```

Figure 8

E. Input and Output Inverter - Adam Kidwell

The input and output inverters were two blocks that were necessary for our unsigned divider and unsigned multiplier to work with in our signed calculator. The input inverter takes the inputs A and B and, based on their MSB, it performs the 2C operation on them to invert them. If the MSB is 1, it indicates that the input is negative and needs to have the 2C operation applied to it to make it positive. Because of this, the inputs going into the multiplier and divider are always positive. The input inverter also has a bit to track whether or not the output of the multiplier or divider needs to be inverted. For example, if A is negative and B is positive, the output will need to be inverted. The bit i is decided by performing the xor operation on the MSB's of the inputs. The outputs of the multiplier and divider both feed into output inverters which use the i signal to decide if the outputs need inverted or not. The block diagram for the input inverter is shown in figure 9 and the block diagram for the output inverter is shown in figure 10. Note that the output inverter in figure 10 is for the divider - the output inverter for the multiplier operates with 10-bits.

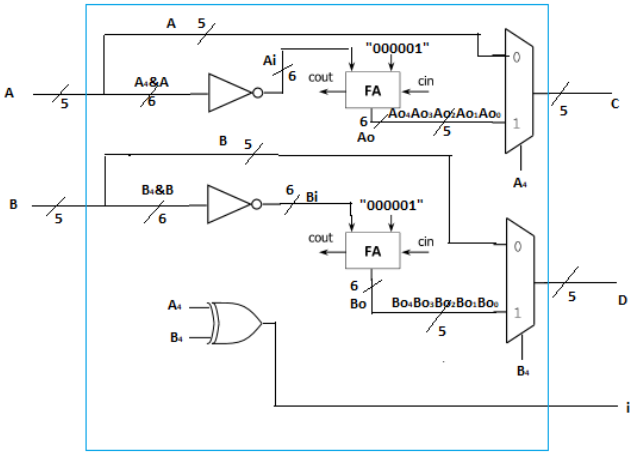


Figure 9

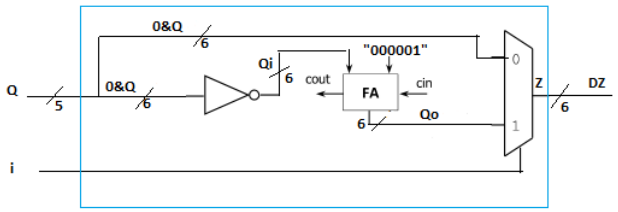


Figure 10

F. Operation Selecting MUX - Adam Kidwell

The operation selecting MUX is a very simple multiplexor that takes the users input as the selector and has four options, one for each mathematical operation that our calculation can perform. "00" selects addition, "01" selects subtraction, "10" selects multiplication, and "11" selects division. The output of this MUX continues through the circuit to be outputted on the 7-segment displays.

G. 2C-to-BCD Decoder - Adam Kidwell

The 2C-to-BCD Decoder was a fairly straightforward, if tedious, component to create. We utilized the with select function to map each of our possible 10-bit inputs [-240,256] to four 4-bit outputs connected as one 16-bit output. For example, we would map the number 144 (0010010000) to 1, 4, and 4 (0000000101000100). Note that 1010 is mapped to show - in the case of a negative number. To cut down the amount of numbers that needed to be mapped, we utilized a 16's times table to avoid mapping numbers that our circuit could not output. This 16-bit result R outputs to the 7-segment serializer, which receives it as four 4-bit inputs A=R(15)&R(14)&R(13)&R(12) B=R(11)&R(10)&R(9)&R(8), C=R(7)&R(6)&R(5)&R(4), and D=R(3)&R(2)&R(1)&R(0).

H. 7-segment Serializer - Adam Kidwell

The 7-segment serializer takes the four inputs A, B, C, and D, and uses a multiplexor to determine which input gets displayed on the 7-segment displays. A counter in combination with a FSM gives the MUX a selector value which it holds for 1ms before moving on to the next selector value. The selector value is also sent to a 2-to-4 decoder whose output is used to enable only one 7-segment display at a time. In conjunction, these two operations allow us to display different numbers seemingly at the same time on the 7-segment displays. This block was available on the VHDL meta-page, so we used that code as a starting place. We made a slight modification to give the 7-segment displays an enable for an improved user experience. This was accomplished by adding a MUX that uses enable as its selector. If enable is 0, it outputs "0000". Figure 11 shows an internal mapping of the 7-segment serializer.

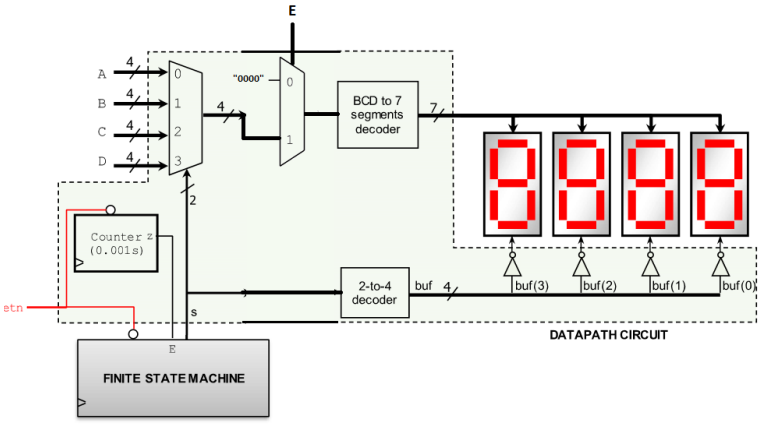


Figure 11

I. Topfile and XDC file - Adam Kidwell

In the topfile, all of the previously mentioned components are mapped together as can be seen in figure 1 (above). The user will use switches 0-4 as their input B, switches 9-5 as their input A, switches 10-11 as the selector for the operation selecting MUX, switch 12 as the enable switch, and the CPU reset button as resetn.

III. Experimental Setup

Our main tool for verifying the functionality of our calculator was the simulation function in Vivado. Some errors we caught through simulation were realizing the multiplier was unsigned, realizing our adder and subtractor needed to be 6-bit to avoid overflow, and realizing a few incorrectly mapped signals. Fortunately, our project was relatively easy to testbench as we control the inputs and the outputs should merely match simple mathematical operations. When the outputs did not match, we loaded in all of the internal signals from the respective component. Doing this allowed us to identify which signal was not acting as expected, and we solved them accordingly. One specific example was realizing that when we divided 0 and 0 in the divider, the output was “11111”. Realizing this error leads to the modification of that circuit to avoid errors when dividing by 0. One error we realized after uploading to the Nexys board that we did not notice in the simulation was that the results of the calculations were updating as the user flipped the input switches, which did not look good and was often confusing. This realization led to the modification of the 7-segment serializer code that added in an enable functionality.

IV. Results

Most of our results came in the form of simulations and timing diagrams. The practice we got in Unit 7 of the class was the perfect preparation for understanding the complex timing diagrams we were faced with. To keep things as simple as possible, we ran different simulations for each arithmetic component. It got most complex when the output of the component did not match the expected result based on the inputs. When this occurred, we had to add in many of the internal signals of the component and go through clock cycle by clock cycle to identify which signals were behaving incorrectly. The timing diagram for each of our components, including all of their internal signals, are shown below to demonstrate the complexity of some of the diagrams we had to understand. There are also images of the project implemented on the Nexys board. Figures 12 and 16 correspond to the adder, figures 13 and 17 to the subtractor, figures 14 and 18 to the multiplier, and figures 15 and 19 to the divider.

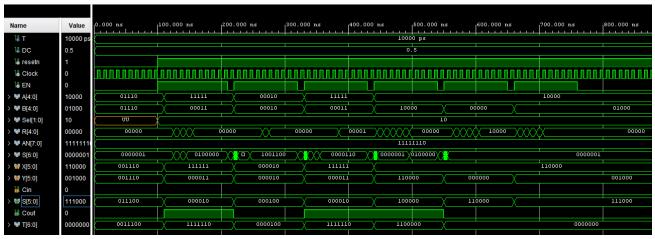


Figure 12 (Adder Signals)

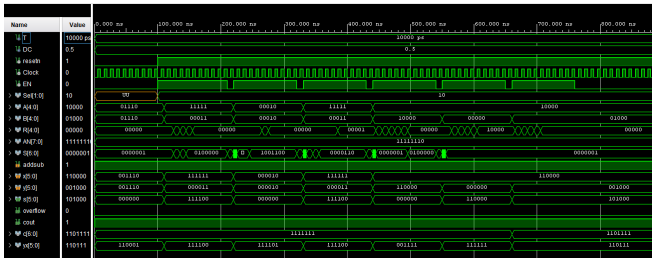


Figure 13 (Subtractor Signals)

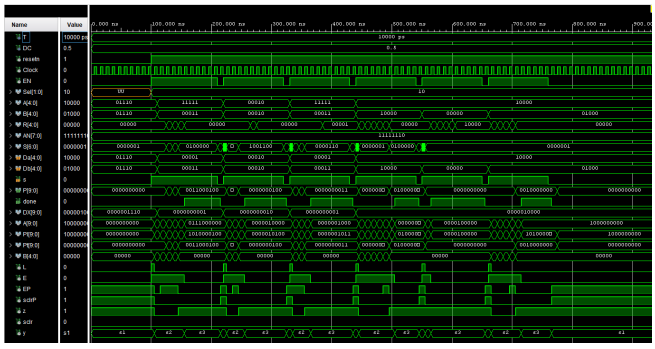


Figure 14 (Multiplier Signals)

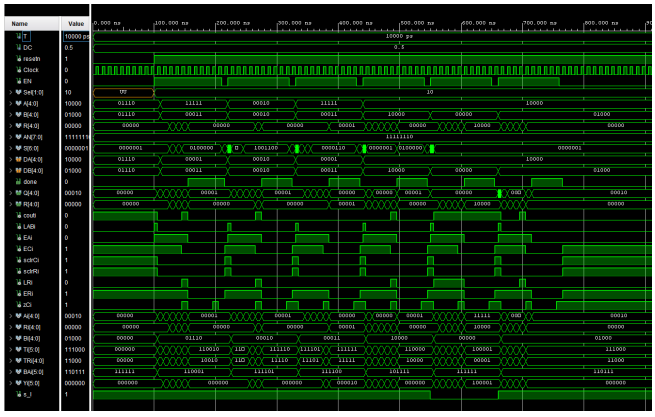


Figure 15 (Divider Signals)

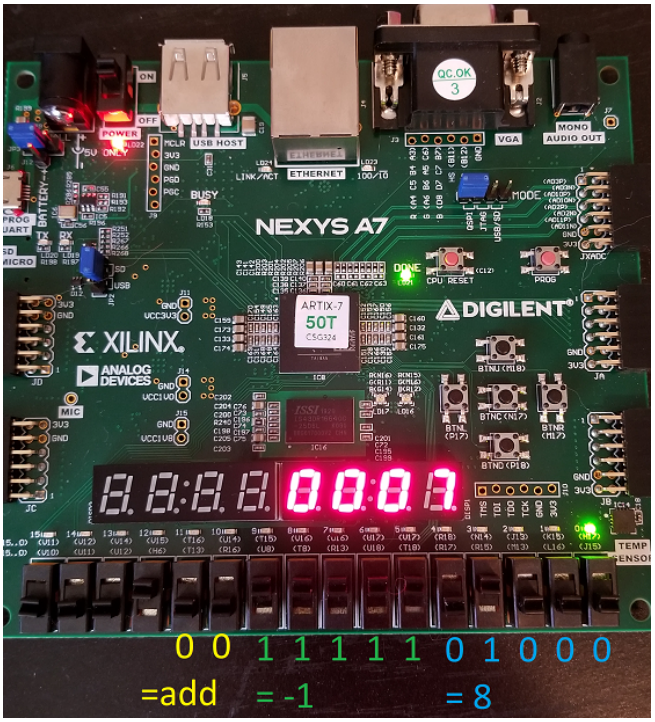


Figure 16 (Adder)

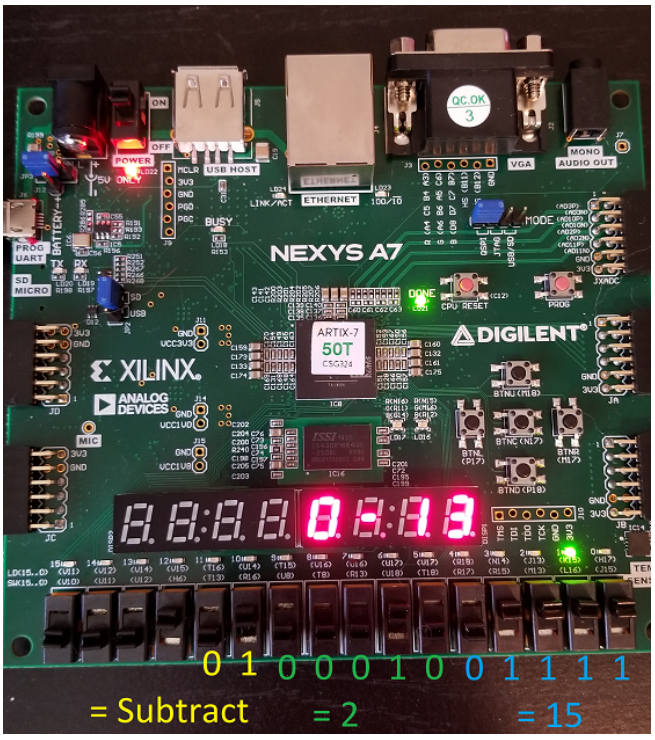


Figure 17 (Subtractor)

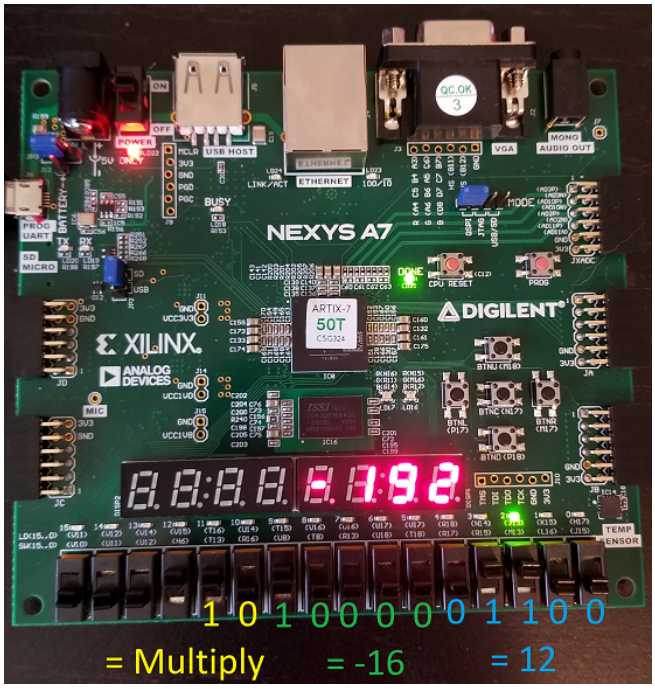


Figure 18 (Multiplier)

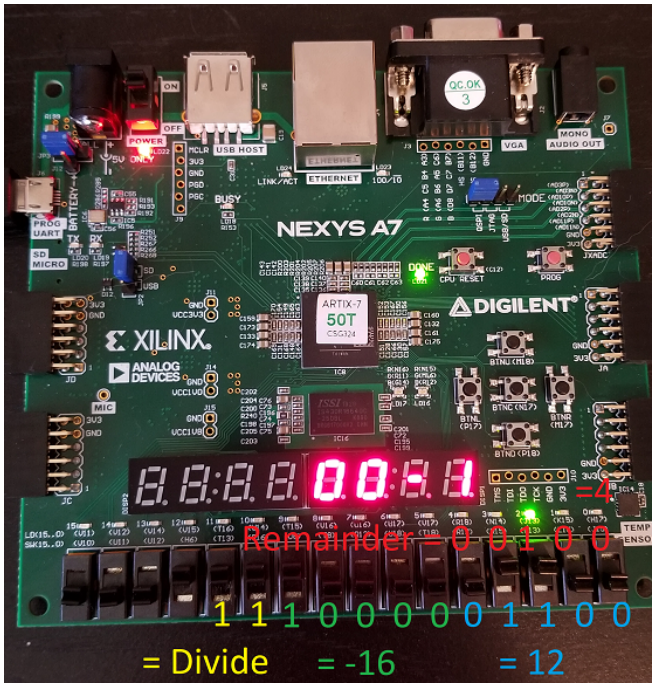


Figure 19 (Divider)

Conclusions

In conclusion, having this project was great practice in designing and implementing our own digital systems. For components like the input and output inverter, designing the component from the ground up to solve a problem we were faced with was an exciting and rewarding task. We learned a lot about debugging problems and

interpreting complex timing diagrams. The necessary understanding of the more complex components like the multiplier and the divider was great practice to make us more comfortable with a range of hardware components, especially FSMs. While we are very proud of our circuit, there are of course improvements that could be made. For one, the remainder output of the divider block displays no matter which function the user has selected. A solution could be using a 4-to-1 MUX whose selector is the selector of the arithmetic function. This MUX inside of the divider could have the remainder connected to the 4th input of the MUX with the other 3 inputs simply having “00000”. This would make it so the remainder only displayed when the user selects the division function. Another improvement could be made in our 2C-to-BCD decoder. Algorithms could be utilized to map the 2C inputs to BCD instead of manually mapping each number as our decoder does. This is not a big problem for our circuit as there are only a couple hundred of outputs that are possible. If our circuit were to be 6 bit or even larger, use of an algorithm would be absolutely necessary. Further improvements could be made to optimize the code, such as using more parameterized components. For example, we have multiple codes for 2-to-1 MUXs with different sized inputs. We could have instead used one 2-to-1 MUX code that works for N-bit inputs. Similarly we have multiple fulladders for different sized inputs when one N-bit addsub could have been used for our adder, our subtractor, and anywhere else in our circuit that needed an adder. These inefficiencies are negligible due to the simplicity of our circuit, but in a more complex circuit these changes may be paramount.

References

[1] "Llamocca, Daniel. Oakland University, ECE 2700" “using block diagrams, previous code, and VHDL examples to aid us in our design”