DARTMOUTH
ENGINEERING
THAYER SCHOOL

ENGS 31: Digital Electronics
Final Project Report
Digital Calculator
Majd Hamdan and Ruben Vargas

Contents

1 Abstract and Project Description

The goal of this project was to construct a digital calculator that could perform basic mathematical operations such as addition, subtraction, integer division, and multiplication and display the results and inputs on the Basys3 board. The inputs are fed to the Basys3 board using the SCI protocol. In order to achieve this goal, we needed to find a way to receive inputs from the SCI generator on the AD2 and display them, along with the result of any operation, on the 7-segment display of the Basys3.

The idea of "top-down" design was used as we started with a high-level state machine to try and understand the entire process, and then created more specific finite state machines and datapaths for our digital calculator. The two main components of our calculator are the sci receiver and the digital calculator shell. The sci receiver is a component of the digital calculator shell that is responsible for processing the SCI that come from the AD2. The digital calculator shell is where the operations are performed (i.e. subtraction, addition, and multiplication) to the inputs that came from the sci receiver. The 7-segment display is also a component of the shell and it allows the inputs, as well as the operation results, to be displayed on the board.

2 Design Solution

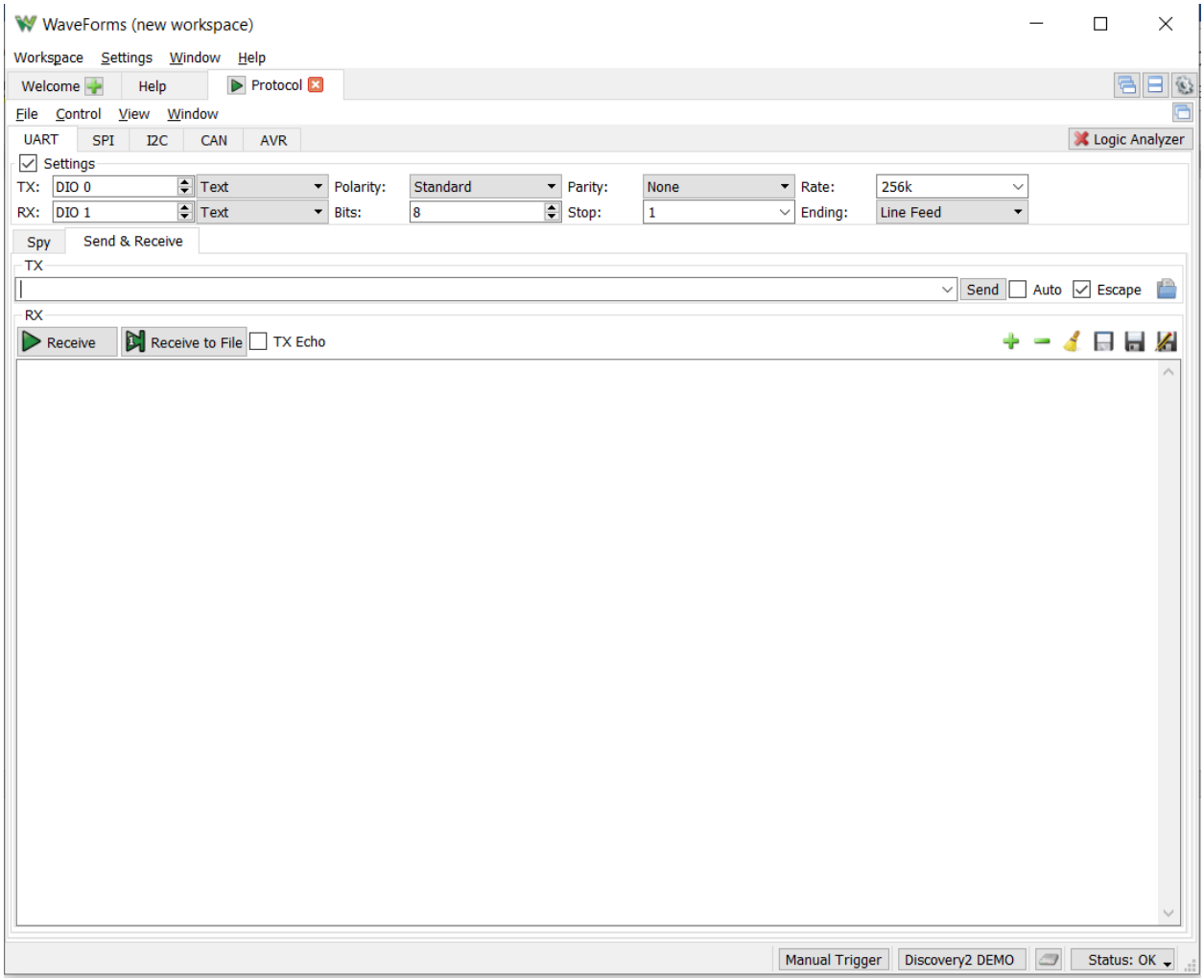
2.1 Specification

We designed this digital calculator to receive inputs from an SCI protocol. The frequency of the calculator clk 1 MHz and the baud rate for the SCI protocol is 256000. While the calculator is currently programmed on these numbers, we wrote the code with the goal of adaptability in mind. The calculator SCI receiver can be adjusted to run on any clk speed and baud rate by simply changing the clk_freq and baud_rate variables with no additional changes to the calculator shell. The calculator will receive inputs from AD2 UART (SCI) protocol. Every time the AD2 sends a number, it will end its transmission with a line feed character (line feed is selected as an ending in the UART options). Since we only have 4 7-segment displays, the maximum number that can be displayed is 9999. Currently, we did not design the calculator to handle overflow. In idle and on reset, the calculator will display "0000". The user must input a number, an operation, then a second number to see a result. Every time the user inputs a number, the number will be displayed on the 7-segment displays. After the user inputs the operation and the second number, the result will be displayed on the displays. The user can then enter an operation, then another number to continue performing operations, or send the "!" character to reset the calculator. The operations supported are addition, subtraction, multiplication and integer division. The calculator will ignore any inputted characters except number characters, the four operations (-, +, *, and /), the reset character (!), and the line feed. As of now, we have not tested the way the calculator handles non-standard input arrangements (for example: number, number then operation or operation, number, then number).

2.2 Operating instructions

1- Star the Baysis3 and program it with the bit code of the calculator in the standard way.
2- Connect the AD2 in the standard way, then select "Protocol" from the welcome screen.

3- Select the UART protocol, then “send and receive”, and set its options to the following:

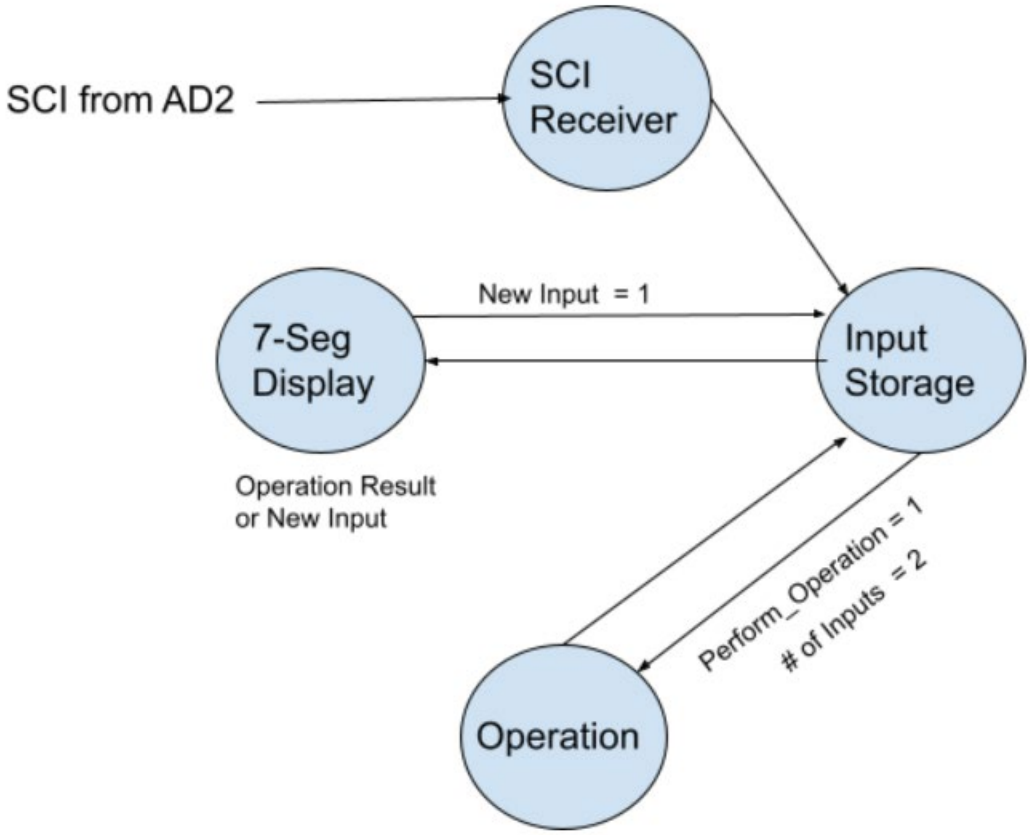


4- Connect DIO 0 of the AD2 to JA1 of the Baysis3 JA PMOD header and the AD2 ground to the JA PMOD ground (In the same raw as JA1).

5- Finally, enter number and operations in the TX field. Each number and operation must be sent independently (do not enter “1+3”, but “1”, then click send, following by “+”, then click send, and finally “3” then click send. To reset the calculator, enter “!” then send.

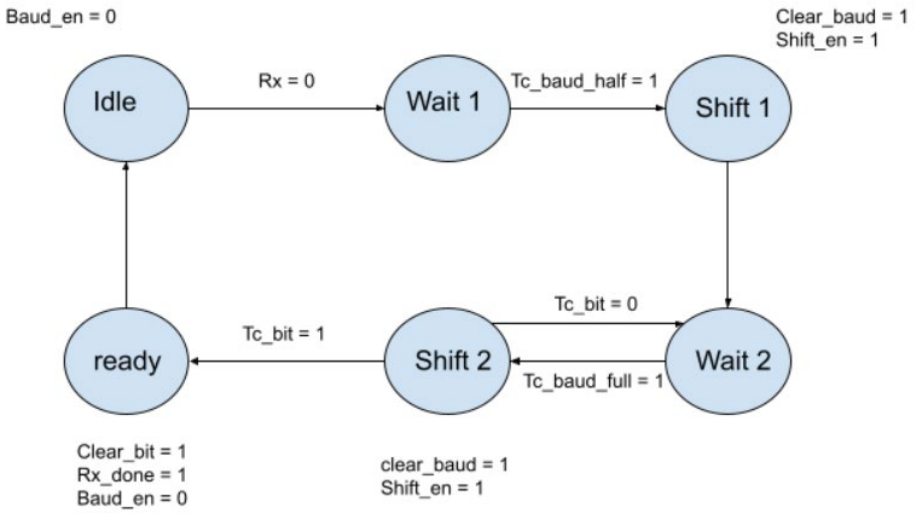
2.3 Theory of operation

2.3.1 High Level State Machine

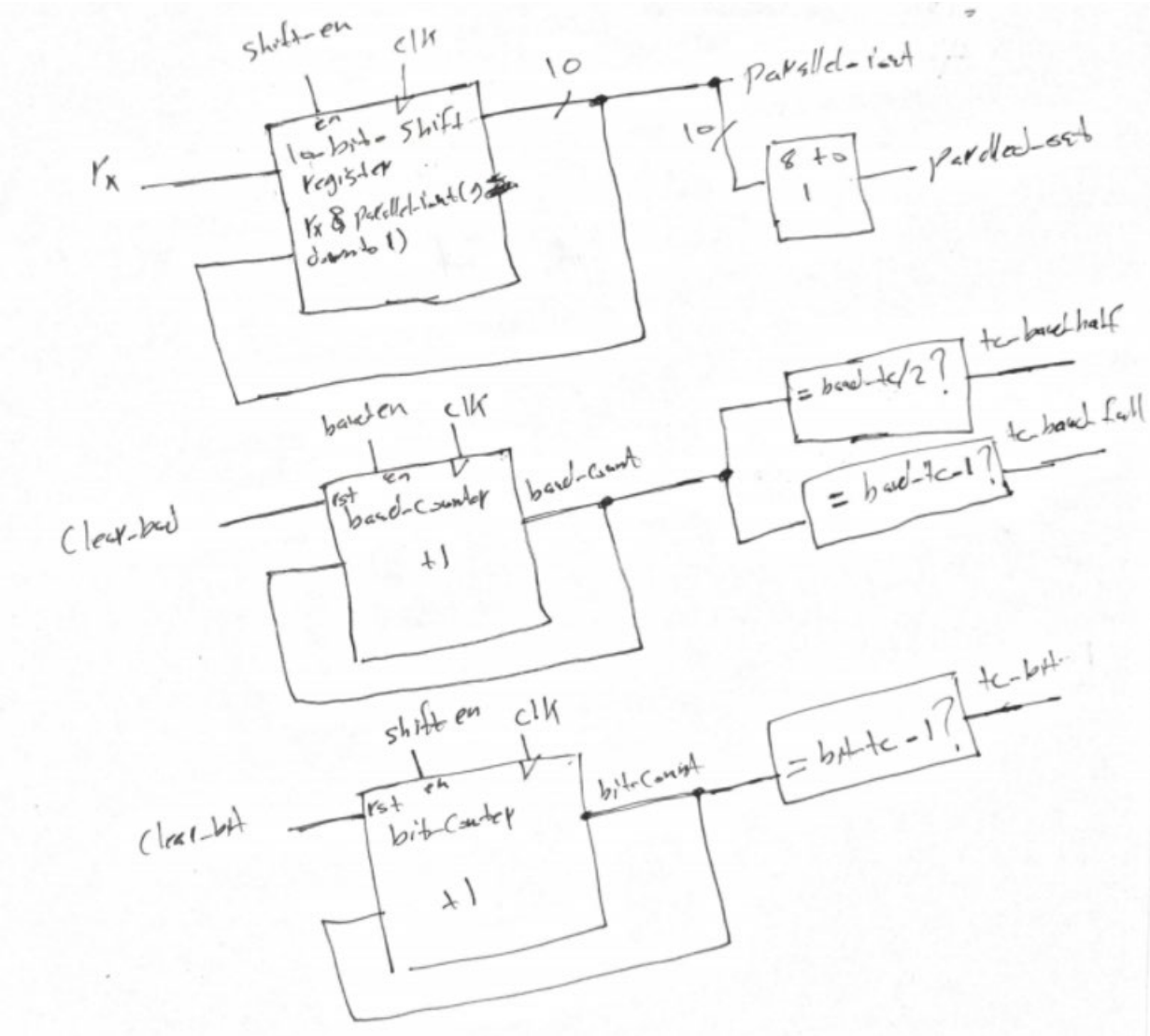


2.3.2 SCI Receiver

FSM

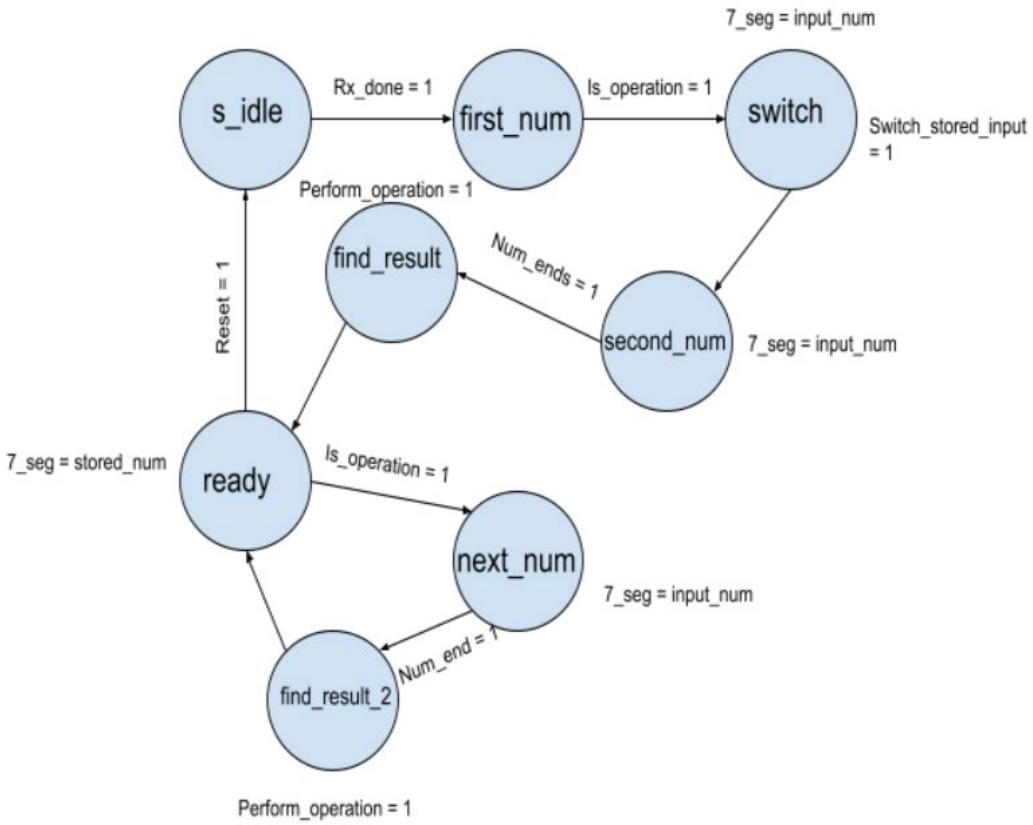


Datapath Block diagram

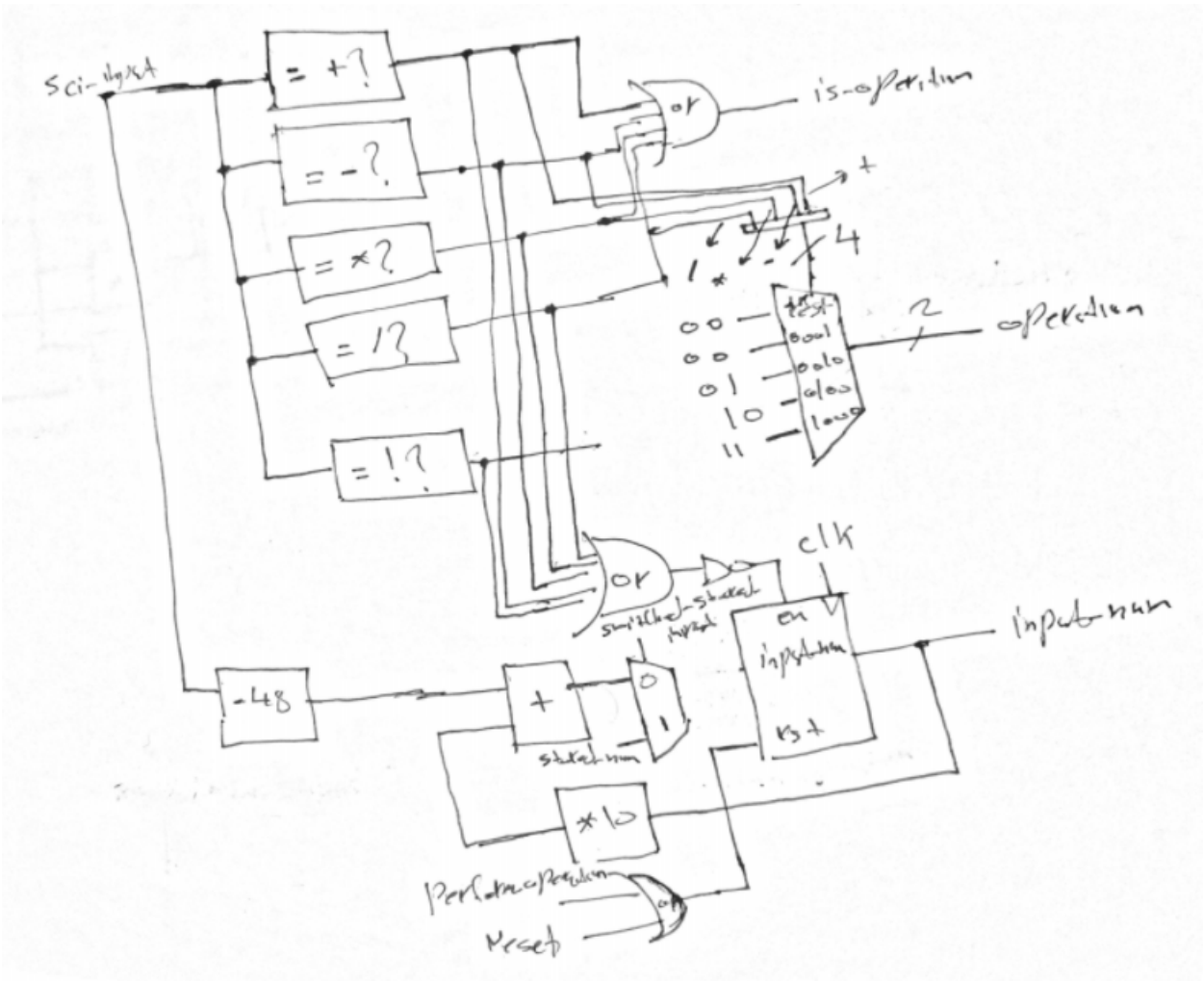


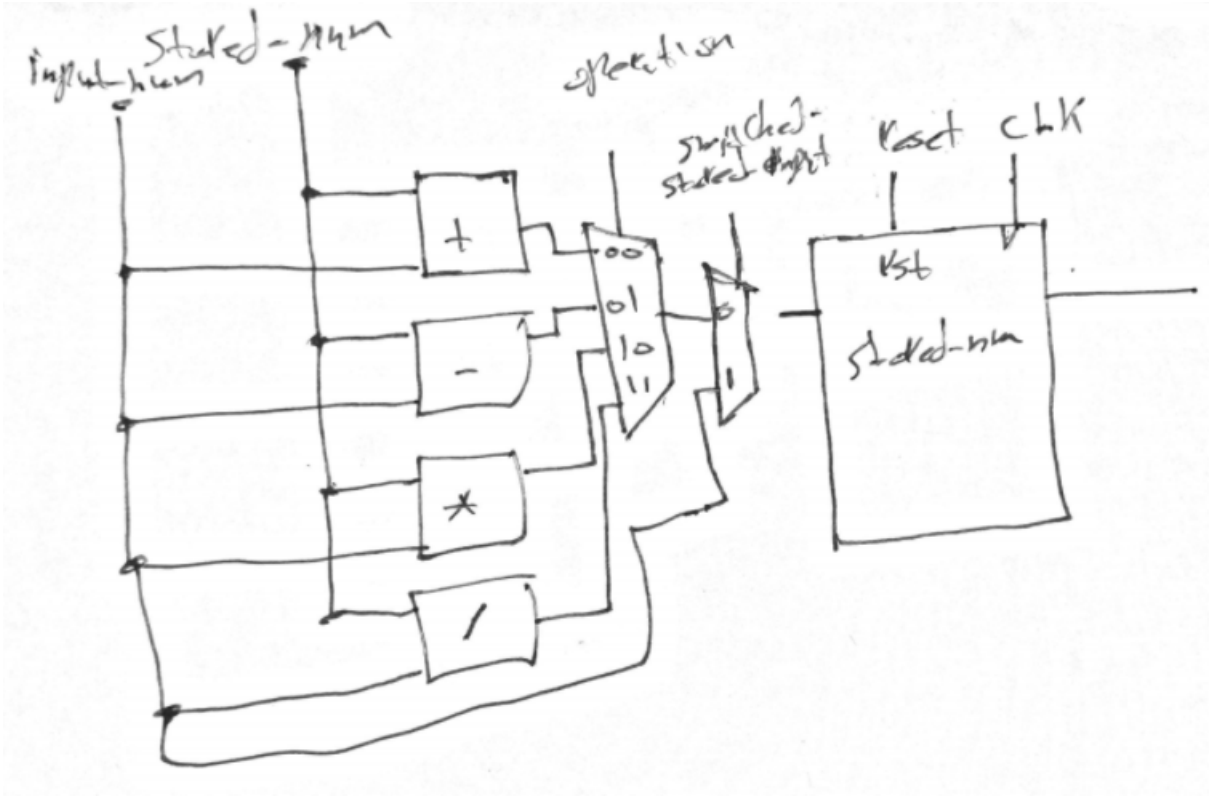
2.3.3 Digital Calculator Shell

FSM



Datapath Block diagram


Sci-digit
= +?
= -?
= *?
= /?
= !?
or
is-operation
+
-
*
/
4
00
00
01
10
11
2
operation
or
clk
switched-stack input
en
input-num
-48
+
0
1
stack-num
rst
*10
Perform operation
or
Reset



2.4 Project VHDL Code

The Digital Calculator shell (entire project) code and test bench can be found here:
https://www.edaplayground.com/x/qhNa

SCI Receiver code and test bench can be found here:

https://www.edaplayground.com/x/SZ7T

The 7-segment display code can be found here:

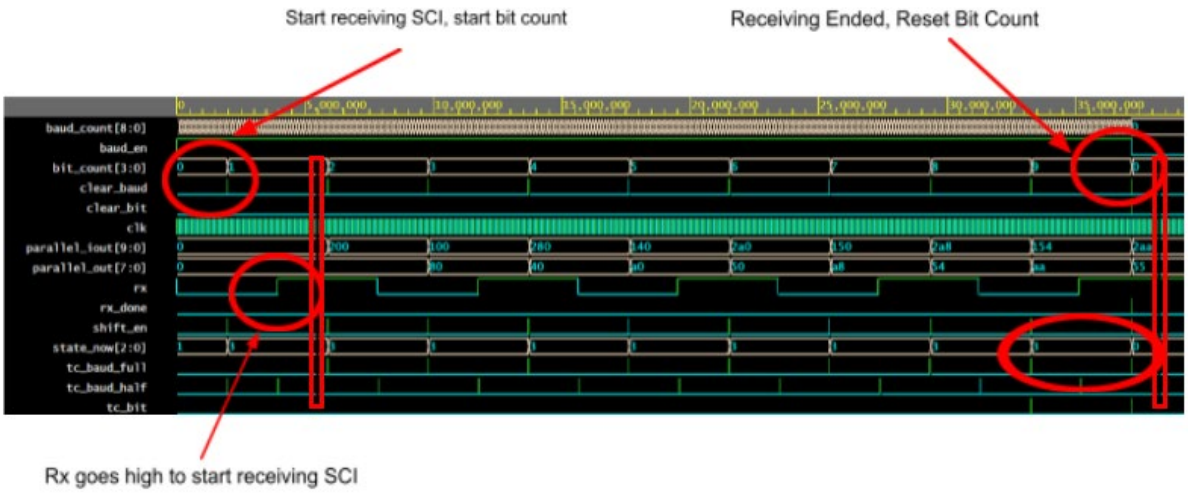
https://www.edaplayground.com/x/HE_d (we used the lab 3 mux7seg)

The constraints for the Vivado project can be found here:
https://www.edaplayground.com/x/ZRND

2.5 Construction and debugging

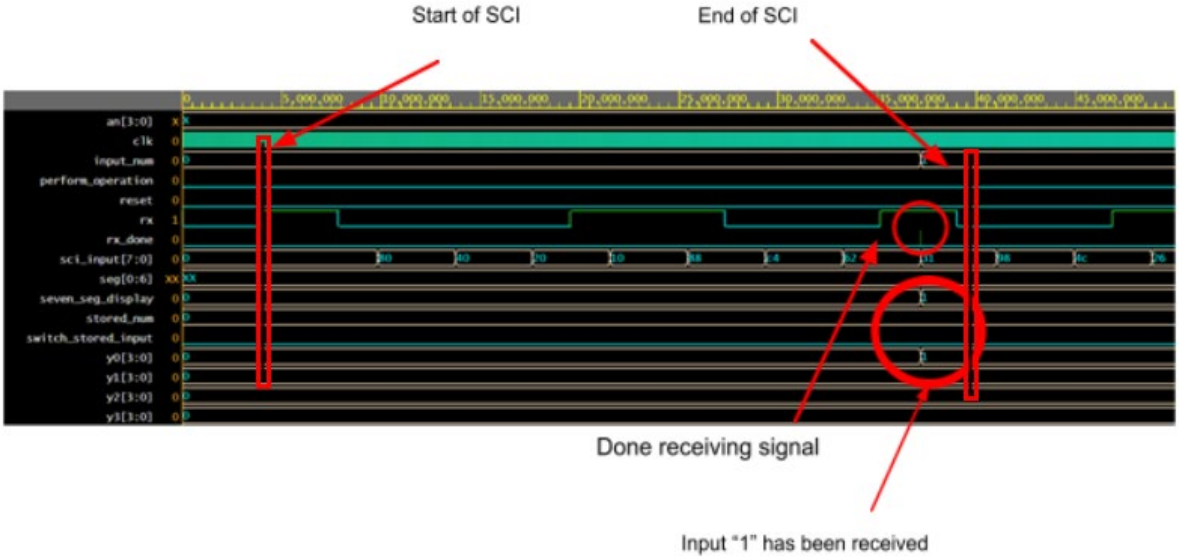
The SCI receiver will decode the SCI info received into ASCII characters. When it finished decoding one character, rx_done will go high indicating that the number on the parallel_out is ready to be read out. When the shell detects high rx_done, it will read parallel_out and turns it into a binary integer (if it is a number) or prepare for operation if it is an operation. When the shell detects an input, reads it and transform it to non-ASCII binary representation, it will get displayed on the 7-segment displays. When a result is ready, it will get displayed on the 7-segment displays.

2.5.1 SCI Receiver testing

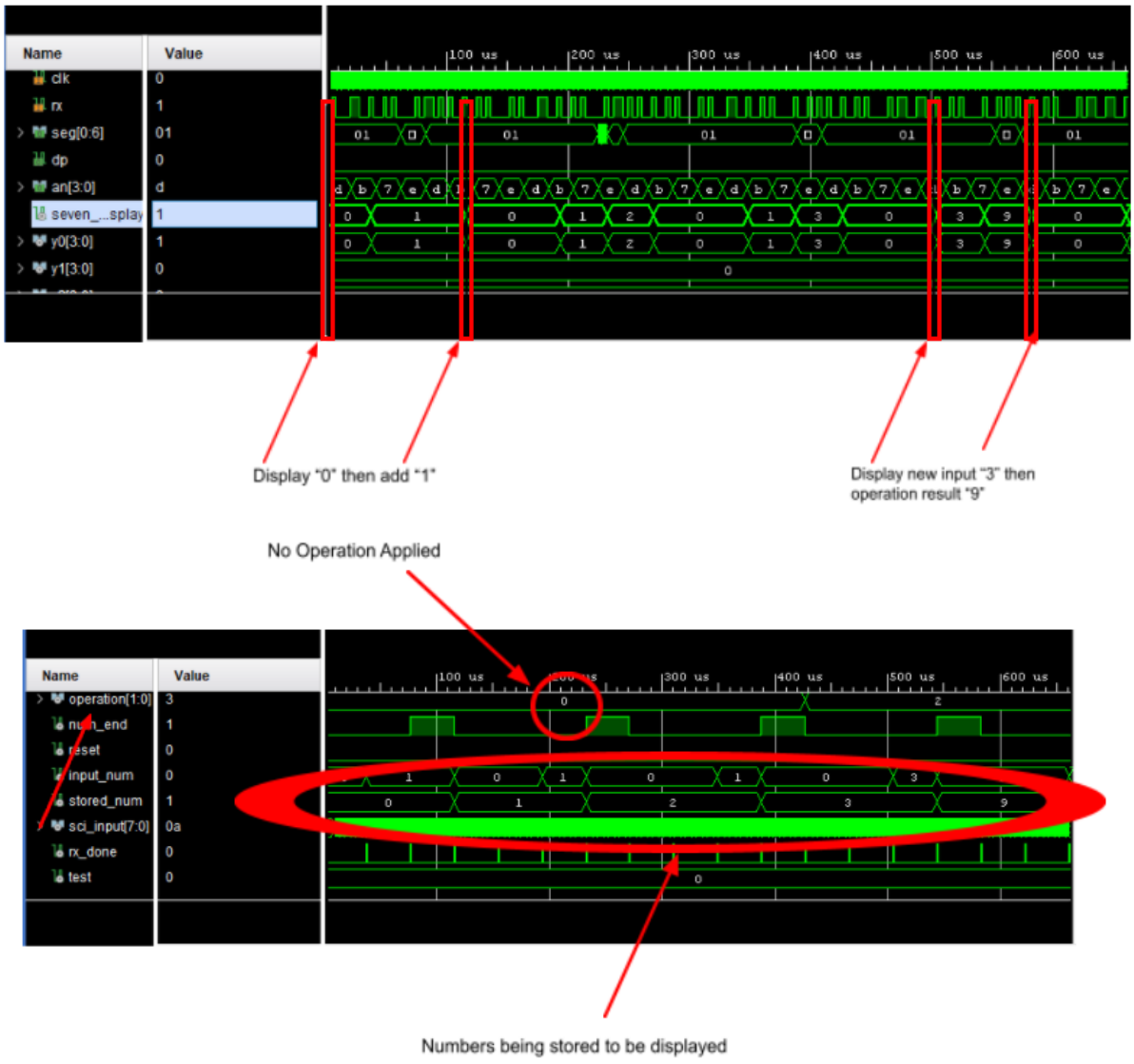


The waveform above shows our functioning SCI receiver as it receives “01010101” one bit at a time. We can also see that our output signal, parallel_out, is working as intended. Since the 8-bit input starts with a “0”, we can see that parallel_out remains at 0 for an extended period of time. Then, the first “1” is received so parallel out is 80 which represents “10000000” in hex. Here, the first “0” was shifted to the right and “1” is now the most significant bit. This process occurs until the entire 8-bit number “01010101” is received. This number is represented by 55 in hex, and we can see that it is the last value the parallel_out signal takes. The signal, parallel_iout, works the same as parallel_out, but it includes a leading and ending bit. When the SCI generator sends a signal, there is always a leading “1” bit and an ending “0” bit. This is why we see the parallel_iout signal take on a value before parallel_out. Rx represents the receiving signal and rx_done goes high when the entirety of the generated SCI has been received. Clear_bit also goes high to reset the bit count that keeps track of the processed SCI. Bit_count will reset when it is 9 because all SCI signals will only be 7 bits plus 2 bits for the end and start bit. While the SCI receiver is receiving inputs, it remains in state 3 as shown by the state_now signal. The signal, baud_en, remains high until the SCI is received. The other baud signals control the rate at which the SCI is received and processed.

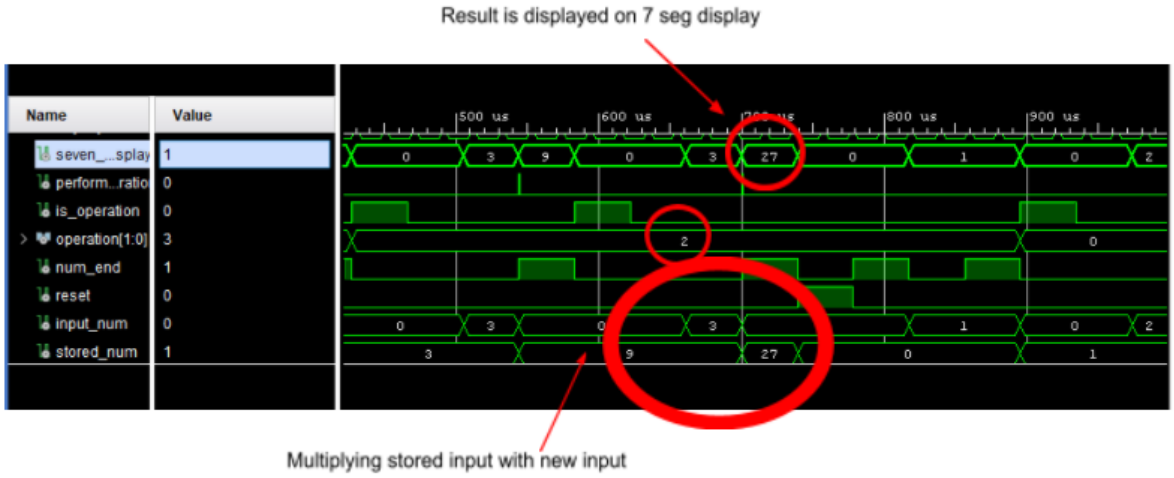
2.5.2 Digital Calculator Shell



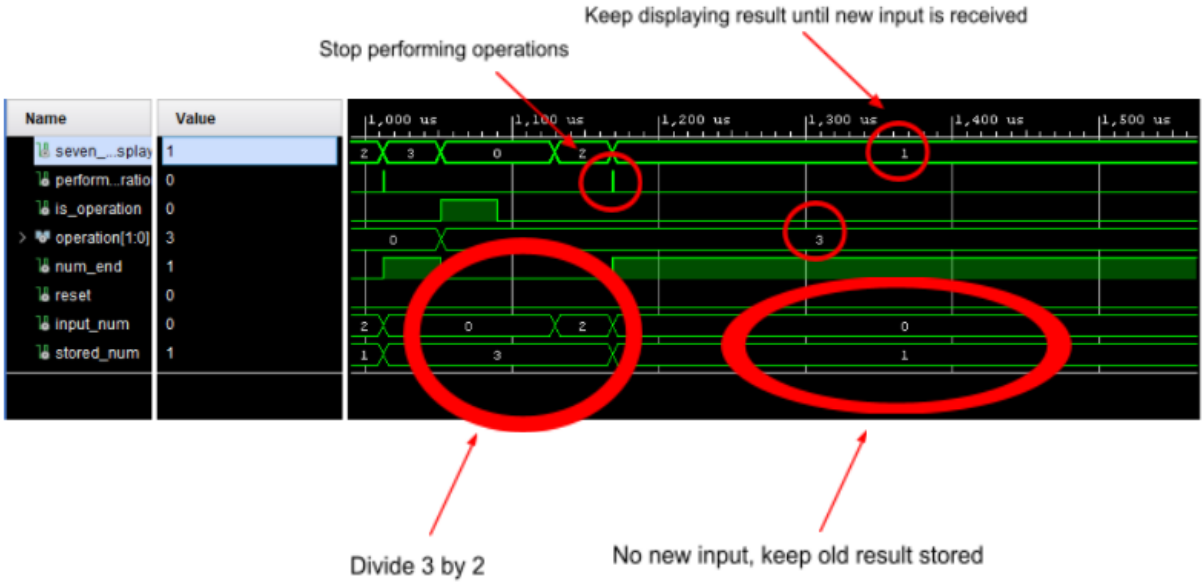
The waveform above shows the sci receiver working with the digital calculator shell. There are no operations being performed at the beginning of the test, so perform_operation is 0. Here, we are just sending the number "1" from the SCI generator and processing it so that it is displayed on the seven-segment display. We can see that the process here is very similar to the previous waveform of the SCI receiver where the SCI is being processed one bit at a time, constantly shifting the most significant bit to the right until we get "00110001" which is 31 in hex and represents the number "1". This then causes rx_done to go high, suggesting that the entire SCI has been received. The signals input_num and seven_seg_display is "1", showing us that we successfully received the input number "1" and it is being displayed on the seven-segment display. The signal y0[3:0] is also "1" because it represents the least significant bit of a four-bit number. Since "1" is actually "0001", y0 will display "1" and y1, y2, and y3 will display "0". This approach is similar to the one done in class labs.



Here is a better display of our code working in Vivado. We see that our input_num and stored_num are working as intended because they are storing the integer number from the processed SCI. We can see that our program is successfully storing the input numbers 0,1,2,3 and 9. The signal, num_end, goes high every time an input has been fully received. Since the operation[1:0] signal is 0, it means that we are adding the inputs together. In the big red oval, we see that we are constantly adding the number "1" until reaching the number "3". The input number is being added to the stored number. Then, operation[1:0] becomes "2", so we are now multiplying the inputs. We can see that we multiply the stored number "3" with the new input number "3" to obtain "9". It is also important to note that our signal, seven_deg_display is correctly keeping track of all the inputs as well as the results of the operation being performed. Seven_seg_display is then "9".



This waveform also shows multiplication operations. We see that operation[1:0] is "2" (corresponds to multiplication in our code) and then multiplying the stored input "9" (from the previous product of 3x3) with another "3" to obtain "27". Our seven_seg_display signal is also letting us know that we are successfully displaying the number "27".



Here, we attempted division because it was the only operation that would lead to non-integer results. We attempt to divide the stored number "3" with the input number "2". The result of the operation is "1", which now becomes a stored value and is displayed on the 7-segment display. We thought it would be interesting to see what vivado would display if a result was not an integer. We can now see that the result is rounded down to the nearest whole number because 3/2 is 1.5 and our waveform is displaying "1". We can also see what happens when there is no new input. Our code keeps the last stored value and keeps on displaying that on the 7 segment display. Even

though our operation[1:0] is set for division, our perform_operation[1:0] is stopping "1" from being divided by "0".

3 Evaluation, Conclusion and Recommendations

While the calculator performed as expected in the simulation stage, it did not work once connected to Baysis3. There seem to be an issue with implementing and synthesizing the code. We tried to debug this issue by first checking if the SCI Receiver would display the input received on Baysis3. This also did not work. We looked at the RTL design for the receiver, we noticed that many of the control signal were not connected to their respective registers. We tried to write the code for the receiver in multiple ways, we referenced previous lab codes for counter and shifter, but it did not synthesize right. The code that gave the most success in connecting the signals to their registers can be found here: https://www.edaplayground.com/x/KMFA. However, even with this code, the signals baud_en, int_rx_done, and shift_en were not connected to their registers in the RTL design. We believe this was the cause of the issue for the SCI Receiver. When we finally narrowed down this issue, we did not have enough time to solve it before the due date of the report. Because we were focused on the SCI Receiver, we could not debug the calculator for possible implementation issues.

Another issue we encountered is converting the seven_seg_display integer (holds the number value that should be displayed on Baysis 3) to 4 8-bit numbers for the 7-segment displays. We wrote a function that attempt to perform this. This function show the correct results when the number displayed in only 1 digits. When it is more than one, the only correct display is the least significant number display. The function did not work as intended because of the time the signals needed to update. In some case, for a signal to hold a correct value, it needs the value of another signal that currently being updated. Instead of getting the new value of the signal, the current signals get the old value of the other signal, which cause the incorrect representation. We believe it would have been easier to use a COE file to convert our inputs into displayable outputs. We had done this in previous labs to display voltage, and it may have worked better than the method that we attempted.

4 References

We used the mux7seg code provided in lab 3. While we did not use any other code provided in this class, or anywhere else, the labs were very helpful to debug and put our components togethers. We did not use any external resources.

5 Appendix

Project VHDL Code

The Digital Calculator shell (entire project) code and test bench can be found here: https://www.edaplayground.com/x/qhNa

SCI Receiver code and test bench can be found here:

https://www.edaplayground.com/x/SZ7T

The 7-segment display code can be found here:

https://www.edaplayground.com/x/HE_d (we used the lab 3 mux7seg)

The constraints for the Vivado project can be found here: