349 Term Project: Fall 2018

Shanley Mullen and Jesse Rosart-Brodnitz
Texas A&M University, College Station, TX 77843
Email: shanleymullen@tamu.edu, jessaggie@tamu.edu

Abstract
This document outlines how to program in seven-segment values onto a DE10-lite board through a launchpad without using preexisting means. The methods used to figure out how to successfully do this were learned at the undergraduate level. For this project, the students had to combine different elements from other classes that they have taken which helped to deepen their understanding of the material that they had covered. In this paper, the authors will go into detail of the multiple techniques used to determine the equations and code needed for this project. Additionally, they will discuss the importance of discerning how the circuits should work to properly program the boards. For this project, it was important to build upon previous knowledge covered in order to properly deduce what the output of the board should be. Also, this paper will talk about the different approaches used to code in both Verilog and ARM.

Keywords
Verilog, ARM, Texas Instruments DE10-Lite board, TM4C123GH6PM Launchpad, Flip-Flops

Introduction

This document is broken up into two pieces, Method A and Method B. Both parts outlining methods of the creation of an 8-bit microcontroller that displays results on a seven-segment display. Method A is a little less complex than Method B. Method B has more capabilities than Method A and can perform simple operations. Both of these methods utilize a TM4C123GH6PM Launchpad, and a Texas Instruments DE10-Lite board and are implemented using an ARM Architecture for Assembly pairing with a Verilog HDL. The methods share a physical set up. The Launchpad outputs bits through the pins on the sides of the device. These pins are connected via Male to Female jumper wire connections and are accepted through the Arduino_IO ports on the DE10-Lite board. Both methods use a series of Flip-Flops configured as J Not K's or D flip-flops to act as a counter and a register. These methods also rely on reduced Sum-Of Product Equations to program seven-segment displays.

ARM is a common architecture based on reduced instruction sets. Designed by Arm Holdings, it was first introduced in 1985. Originally deriving from Acorn Risc Machine, the architecture eventually made its way into what it is today and is commonly used in many everyday devices. Thumb-2, which was introduced in 2003, allowed for the use of up to 32 bits instead of just 16. This was done by adding 32-bit instructions sets. Thumb-2 still allows for use of the original 16 bit Thumb commands as well. In this document, Thumb-2's instruction sets will be used.

Verilog HDL is a Hardware Description Language which focuses on the design of electronic systems. Verilog was heavily influenced by FORTRAN and C programming. It is also the predecessor to SystemVerilog. First produced in 1984, This program allows for modular creation and can be built up in hierarchies to achieve multiple levels of abstraction. Inside each module, the statements are executed sequentially, however, all the modules are executed in parallel. This is also known as Pipelining, making this a Dataflow system or language. The software is also capable of synthesizing the results in a Register-Transfer Level known as RTL. This method of RTL translates the written code into wiring block diagrams, which can assist the designer with his or her work.

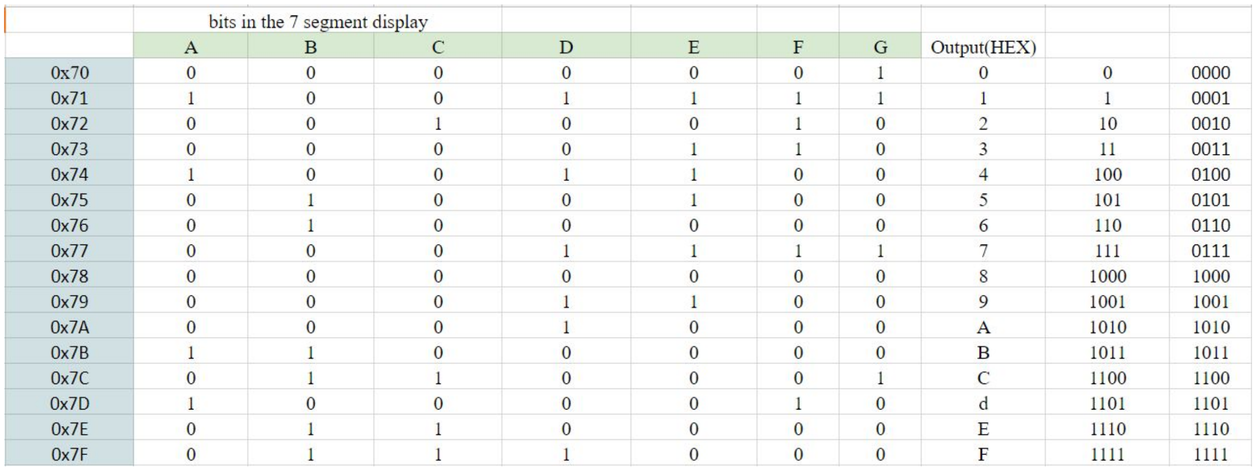
A specific digital logic device that is used to control operations in the second method is the Arithmetic Logic Unit, or ALU for short. This device is used inside of Verilog HDl. The device allows for mathematical and logical operations to be assigned to it via opcode. The designer is capable of picking what they want to be assigned for what opcode they choose. The device also takes two operands in the form of inputs. The operands then undergo the operation specified by the opcode. A zerocheck is also capable of being implemented. This checks to see if the output of the operation is zero.

In the first method that will be discussed, ARM Thumb-2 is used to output 16 hexadecimal coded values across the pins on the Launchpad into the DE10-Lite board. Verilog HDL is used to accept these inputs and wire them to seven LEDs as well as into the flop flops. A ground is also wired across the boards and one of the signals is wired as a clock. The Flip flops take four of the inputs as well as a clock signal and send them onto the HDL module which converts them into HEX codes for the seven-segment display. The board should display 0-F in hexadecimal as it cycles through the input values.

In the second method that will be discussed, ten hexadecimal coded values will be output across from the Launchpad. These bits will be used to input values, control an ALU, and control the clock signal. Four of the input bits will be paired with zeroes in the form of 0000XXXX and will be wired as an operand into the ALU. The ALU will also take in three bits as an opcode to determine what operation the ALU performs. This device will send out a signal in the form of a zero check which will be attached to an LED. This should signal the user if the end result of the operation is zero. The other output of the ALU will be wired into the accumulator which is just an 8 bit register. A clock signal is wired into the register as well. The accumulator will output back into the ALU as the second operand as well as into the HDL model. The HDl model converts the output into Hexadecimal codes for the seven-segment display. The display on the board should be the ending result of the ALU operation.

Designing a Seven-Segment Display

The purpose of method A was to design a fully functioning seven-segment display using different concepts covered over multiple classes. To begin the design process truth tables based on the lower four bits of the hexadecimal inputs 0x70 through 0x7F would need to be created. Once the truth tables had been made then it was necessary to generate k-maps in order to find the equations needed to make the values display, the figure below shows the truth table created for this project. These equations will be used in the Verilog portion of the program.



	bits in the 7 segment display									
	A	B	C	D	E	F	G	Output(HEX)		
0x70	0	0	0	0	0	0	1	0	0	0000
0x71	1	0	0	1	1	1	1	1	1	0001
0x72	0	0	1	0	0	1	0	2	10	0010
0x73	0	0	0	0	1	1	0	3	11	0011
0x74	1	0	0	1	1	0	0	4	100	0100
0x75	0	1	0	0	1	0	0	5	101	0101
0x76	0	1	0	0	0	0	0	6	110	0110
0x77	0	0	0	1	1	1	1	7	111	0111
0x78	0	0	0	0	0	0	0	8	1000	1000
0x79	0	0	0	1	1	0	0	9	1001	1001
0x7A	0	0	0	1	0	0	0	A	1010	1010
0x7B	1	1	0	0	0	0	0	B	1011	1011
0x7C	0	1	1	0	0	0	1	C	1100	1100
0x7D	1	0	0	0	0	1	0	d	1101	1101
0x7E	0	1	1	0	0	0	0	E	1110	1110
0x7F	0	1	1	1	0	0	0	F	1111	1111

Figure 1

Next, the ARM code was to be created. This was done by loading in the GPIO_Init. Once loaded in it could then be divided into both port E and port F which was important since the lower four input bits were PF2, PF3, PE1, and PE2 as located on the launchpad. This was all located in the main part of the program. After that, the first loop is encountered, startpoint, this sets the R7 and R8 values to be #0x70, these values are then moved to R1 and R3. These values will keep increasing until R7 is greater than 0x7F until then it enters into the next loop, cycle. In cycle, the R1 value is then shifted left two and R3 is shifted right one. These values are stored in R0 and R2 respectively. These values are then set back to zero within the cycle, this then goes into the delay function. Once out of the delay function it enters into the clock function. After this the code goes into another loop, parachute, this loop will keep increasing the R7 and R8 values by increments of ones entering into cycle every time until a value greater than 0x7F is reached. Once this value has been reached the program will enter into its final loop thus terminating it. The arm portion is important as it determines what the output will be on the 7 segment display. In this case, the output is 0 through F.

Finally, it is necessary to move onto the Verilog portion of method A. In this section the first thing that needed to be declared was the clock which was LED 9 connected with ARDUINO_IO 2. Once that had been created a large function, bigmoodflop flobert, was created to pass through the values LED 6:0, HEX0 [6:0], ARDUINO_IO [9:3], and ARDUINO_IO [2]. This then went into the module of the same name. Here the outputs Q and HEX are specified along with thing inputs D and the CLK. This information is then passed into the myDFlop function. It is in this module that the smaller flip-flops are specified. Only the lower four bits are passed into these flip-flops. After these actions have been completed it then goes back to the bigmoodfloop module and then into the countthis module. It is in this module that the HDL is made and the equations found in the k-maps are then inputted. This is what controls the seven segment display a photo of the board in action can be found in the figure below.

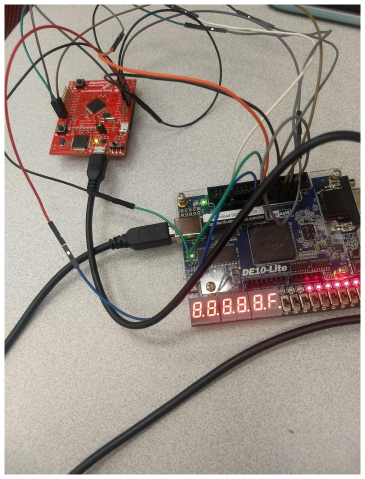
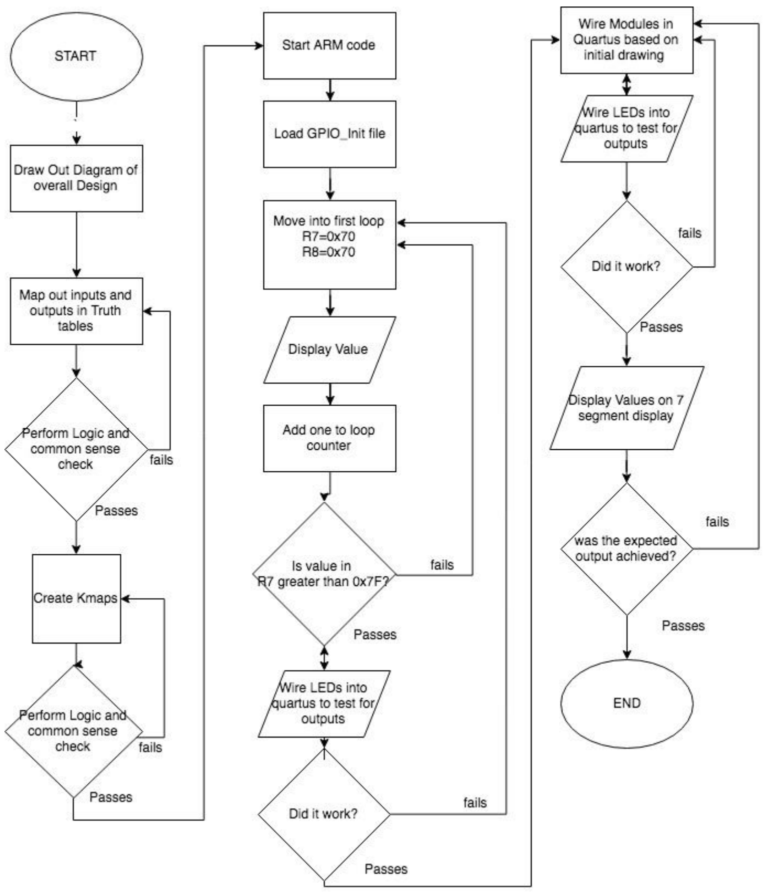


Figure 2

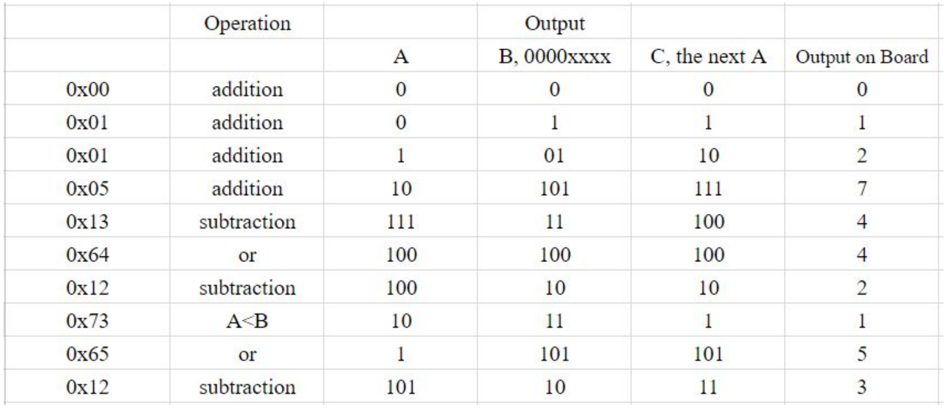
The design process to be followed can be seen below in Figure Overall_Design_MethodA.


START
Draw Out Diagram of overall Design
Map out inputs and outputs in Truth tables
Perform Logic and common sense check
fails
Passes
Create Kmaps
Perform Logic and common sense check
fails
Passes
Start ARM code
Load GPIO_Init file
Move into first loop R7=0x70 R8=0x70
Display Value
Add one to loop counter
Is value in R7 greater than 0x7F?
fails
Passes
Wire LEDs into quartus to test for outputs
Did it work?
fails
Passes
Wire Modules in Quartus based on initial drawing
Wire LEDs into quartus to test for outputs
Did it work?
fails
Passes
Display Values on 7 segment display
was the expected output achieved?
fails
Passes
END

Figure_Overall_Design_Method_A

Designing a Seven-Segment Display with ALU

Method B was used in designing a seven segment display that took in specific inputs and outputs them through an ALU. In order to successfully complete this section of the project the first thing that needs to be done is the creation of the truth table. This is done by writing out the hexadecimal values 0x00, 0x01, 0x01, 0x05, 0x13, 0x64, 0x12, 0x73, 0x65, and 0x12 into binary. Only the lower four bits are needed to generate the output. These values are used to determine the operation needed to come up with the expected output of the board. The operations are based on what is in the ALU. For the ALU used in this project 0 is addition, 1 is subtraction, 2 is anding, 6 is oring, and 7 is set to the input A is greater than the B input. The truth table for this can be seen in the figure below, this will be used in the Verilog section.



	Operation		Output		
		A	B, 0000xxxx	C, the next A	Output on Board
0x00	addition	0	0	0	0
0x01	addition	0	1	1	1
0x01	addition	1	01	10	2
0x05	addition	10	101	111	7
0x13	subtraction	111	11	100	4
0x64	or	100	100	100	4
0x12	subtraction	100	10	10	2
0x73	A<B	10	11	1	1
0x65	or	1	101	101	5
0x12	subtraction	101	10	11	3

Figure 3

Once the initial design phase has been completed the next portion is to code in ARM. The first step is similar to in method A where GPIO_Init is uploaded into the program and then divided it into two parts one for port F and one for port E. Next, the program moves into the first loop startpoint which sets the R1 and R3 to the hexadecimal value 0x00 and then shifts R1 left 2 and shifts R3 right 1. These values are then stored in R0 and R2 respectively. This then gets set back to 0x00 then it gets passed into the delay and the clock. For each remaining hexadecimal value, this is repeated. After all, this has been completed the code is then taken to loop death where it stays on the last value.

The next portion of method B is done in Verilog. Starting this part is very similar to how it was done in method A where LED 9 is set to ARDUINO_IO 2. An operand needed to be made as well where the four lower bits are assigned to ARDUINO_IO [6:3] and the other four are set to zeros. Everything that has been assigned can then be passed into the ALU, the accumulator milliondreams, and countthis. In the ALU module ALUout, zerocheck (LED 8), ACC, B, and ALUopcode are needed. It is there that the zerocheck is set so that when ALUout is equal to zero the led will turn on. It will go into the operation code provided previously in this document. Then, it goes to the accumulator that houses the flip-flops. The same countthis module is used as the one found in part A. A picture of the working code can be found in the figure below.

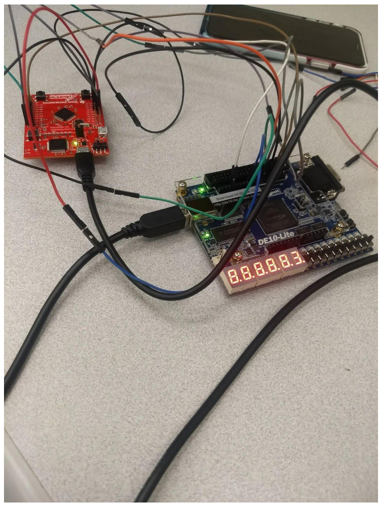
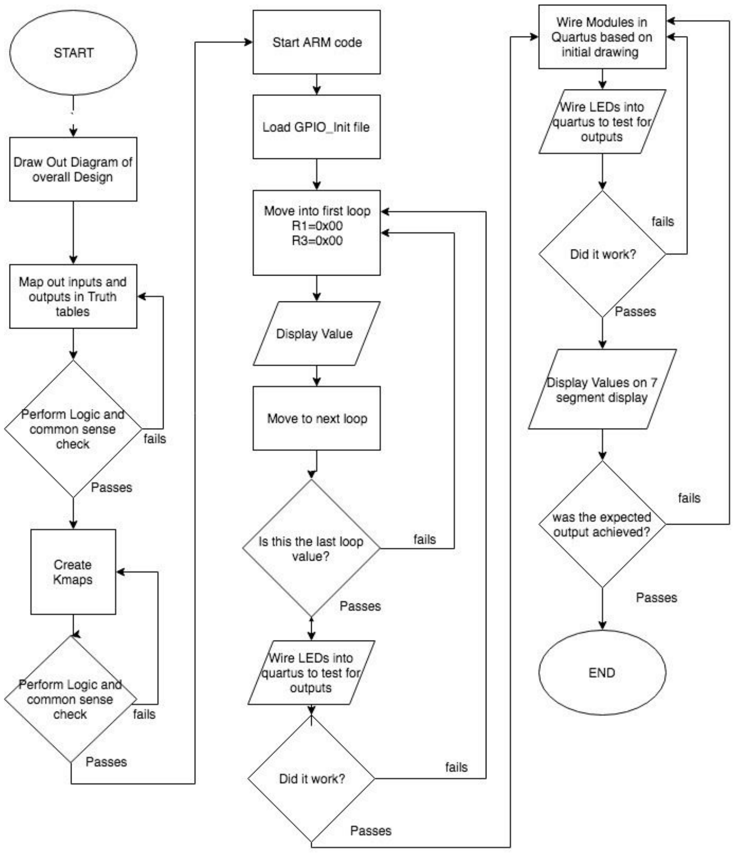


Figure 4

The design process to be followed can be seen below in Figure Overall_Design_MethodB


START
Draw Out Diagram of overall Design
Map out inputs and outputs in Truth tables
Perform Logic and common sense check
fails
Passes
Create Kmaps
Perform Logic and common sense check
fails
Passes
Start ARM code
Load GPIO_Init file
Move into first loop R1=0x00 R3=0x00
Display Value
Move to next loop
Is this the last loop value?
fails
Passes
Wire LEDs into quartus to test for outputs
Did it work?
fails
Passes
Wire Modules in Quartus based on initial drawing
Wire LEDs into quartus to test for outputs
Did it work?
fails
Passes
Display Values on 7 segment display
was the expected output achieved?
fails
Passes
END

Figure_Overall_Design_Method_B

Conclusion

This document outlines two methods for creating 8-bit microcontrollers. The first method being wiring the inputs straight into flip-flops to be converted and display the output. The second method involving the wiring of an ALU to be converted to the output. ARM and Verilog are also covered briefly in this document but to the extent for a basic understanding of the work presented. While there are many other methods to go about creating simple 8-bit microcontrollers, these two methods are fairly straightforward and a good stepping point for anyone starting to dabble in microcontrollers.

References

"Verilog Resources." Verilog.com, 2012, www.verilog.com/.