THE AMERICAN UNIVERSITY IN CAIRO

School of Sciences and Engineering

Project 2 Report

By:

Dalia El-Naggar 900191234

Hanaa Shaaban 900191050

Nada Atia 900193555

CSCE 2301-01 Digital Design I

Fall 2020

Introduction:

Through this code, we implement signed sequential multiplication and division processes using Verilog coding and implementing it using the Basys 3 FPGA board. The code depends on a number of modules and source files to build the construction.

The code process:

1. The code simply receives an input to choose between division and multiplication
2. The code receives two 8-bit binary values to multiply or divide (in case of division, A must be divisible by B).
3. The code represents the decimal result of the multiplication or the division process on a seven segment display.

Code analysis:

- Multiplier (top module):

  Main Inputs and Outputs:

  - input clk (clock)
  - input RxD (the input for the keyboard buttons)
  - input [7:0] A (the first input value)
  - input [7:0] B (the second input value)
  - output reg [6:0] sevenSegment (the wire connected to the seven segment display)
  - output reg [3:0] enable (the enables to choose among the four seven segment display on the board)

Registers and wires:

- wire en (a flag to initialize the process and get the complement of the values if needed)
- wire rst (a reset for the program to initialize everything by zero)
- wire div (a flag for the type of process: 0 for multiplication and 1 for division)
- reg EA=0 (a flag for finishing the process of complement and to start the actual multiplication/division)
- reg EP=0 (a flag for finishing the process of multiplication or division and to start displaying the result)
- wire [15:0] P (the wires that holds the final binary value of the result of the multiplication or the division)
- reg [15:0] A_comp=0 (the register that holds the first input value or its complement in case it is negative)
- reg [7:0] B_comp=0 (the register that holds the second input value or its complement in case it is negative)
- reg [15:0] P_temp=0 (the register that holds the binary value of the result throughout the process)
- wire [6:0] onesSeg, tensSeg, hundredsSeg (the wires that could be connected to the seven segment display to show each digit)
- wire [3:0] ones, tens (the wires that holds the binary value of the tens and units digits of the result)

- wire [1:0] hundreds (the wires that holds the binary value of the hundredths digit of the result)
- wire clk_out (the wire that holds the new adjusted clock with a frequency of 100Hz)
- wire [1:0] count (the count the changes between different enables and wires for the display)

Process:

1. We have three UART modules: letter ‘a’ for en, letter ‘b’ for rst, and letter ‘c’ for div0.
2. If div0 was pressed, we do the division otherwise multiplication is the done.
3. The div0 gets into a stabilizer, to guarantee that div holds the same value and keep it in div.
4. We create an always block that is exectuted at every positive edge of the clock with an if statement that checks the value of rst and when it is 1, we initialize the registers (P_temp, A_comp, B_comp, EA, EP) with zero.
5. In the same always block, when the rst is not pressed, we check the en. If en was pressed, we check the last digit of the two inputs; if it is 1, we get the complement of this input. When this is done, the EA flag is marked as 1.
6. In the multiplication process, the value of p_temp gets updated through the addition of the old p_temp and the latest value of A_temp,

as long as B_temp not equal to 0, EA is one, and the first digit in the B_temp is 1. B_comp gets right-shifted with one bit and the A_comp gets left-shifted with one bit. When B_comp reaches zero, the EP flag gets the value 1.

7. If the div button was pressed, the division process is applied through checking whether the subtraction of A_comp and B_comp is bigger than 0 and EA is one, updating the A_comp with value of the subtraction of A_comp and B_comp, and increment p_temp by 1.
8. The P gets the value of 0 as long as the process of multiplication or division didn't finish or rst was pressed, otherwise it gets the value of p_temp.
9. We call the bin2bcd module to divide the P product value into three decimal digits for display.
10. A SevenSegmentDecoder module for each digit is instantiated to turn every digit into the suitable display value on the seven segment display.
11. We call the module ClockDivider to adjust the clock on the required frequency 100 Hz for better vision on the seven segment display.
12. We call 2-bit Counter so that count could hold 4 different values and display different digits on seven segment display with frequency that allows us to see several digits at the same time.
13. The last always block is for display by checking the value of count and according to it we choose among the four decimal values of

display (the most left seven segment one for the sign display) and the enables attached to it.

- UART:

  A module to receive a command of reading a letter from the laptop’s keyboard and connect it to wires in the top module.

- Stabilizer:

  It is a FSM to guarantee that div keeps the same value without change (if div0 is 1, div will keep the value of 1 even if the button is not pressed) unless rst is 1.

- bin2bcd:

  It is a module to transform the binary value of the product into three buses where each hold the value of a decimal digit.

- SevenSegmentDecoder:

  It is a module to generate the wires needed for the seven segment display according to the decimal digit in the input.

- ClockDivider:

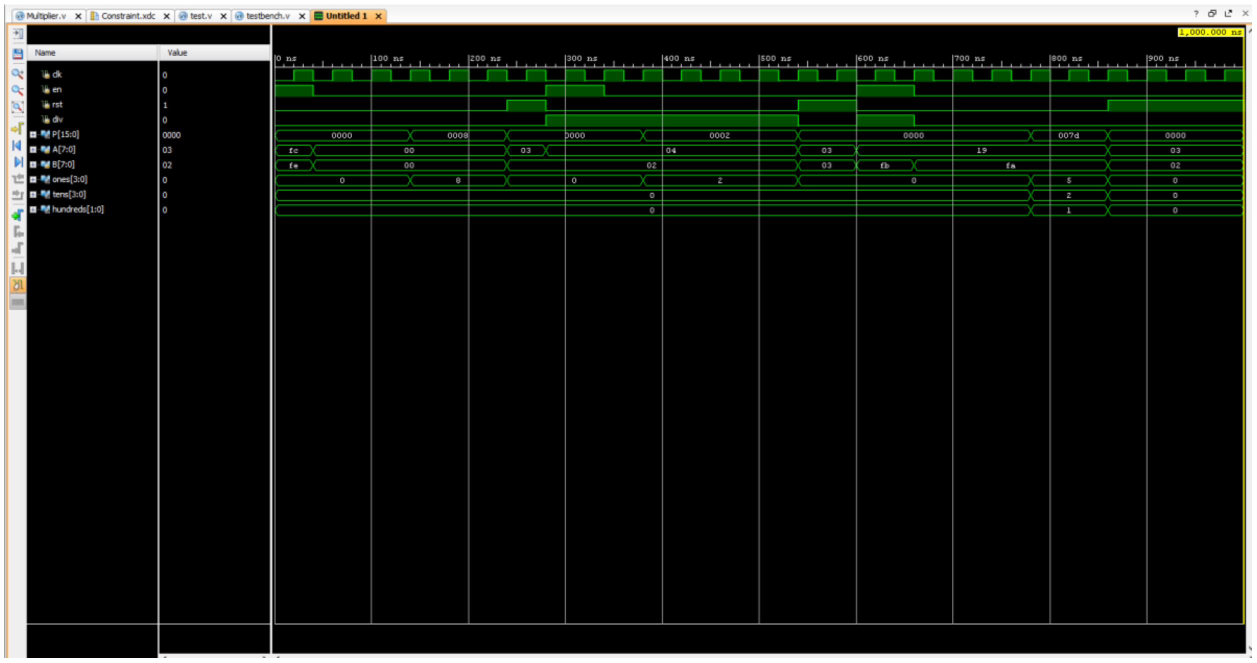
  It is a module to adjust the clock from 100MHz to 100Hz only.

- Counter:

  It is an n-bit counter.

Simulation:

Before applying our Verilog code on the chip, we tested our logic using a testbench simulation and the following is a screenshot of our results:



Test Cases:

Test Case 1:

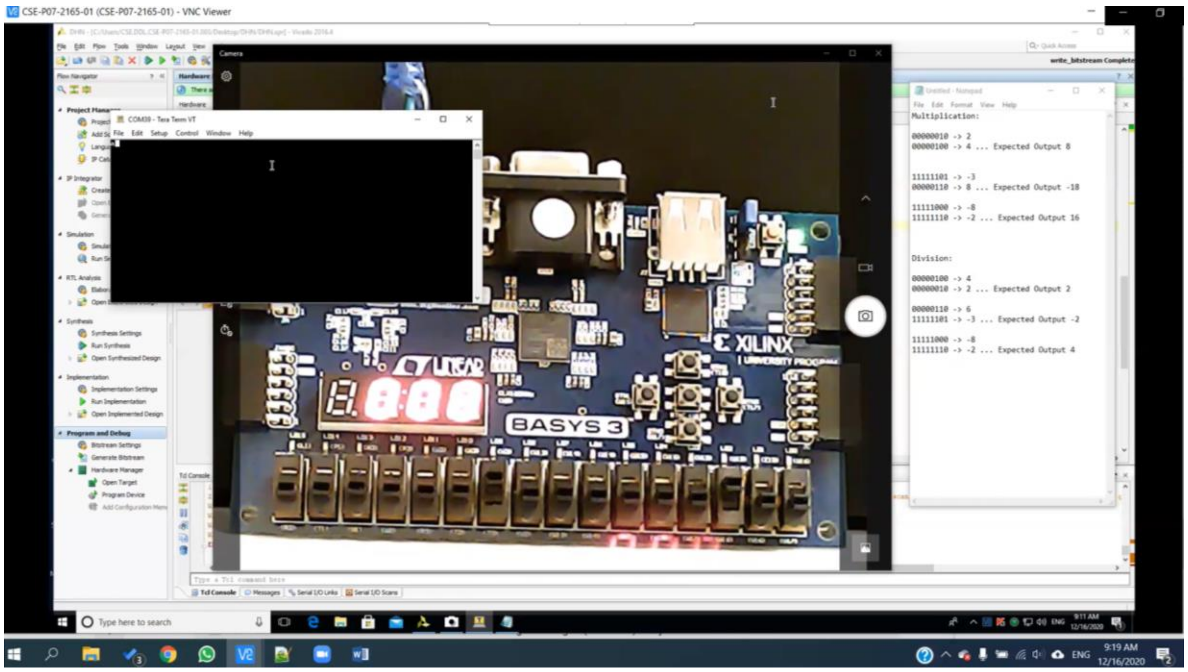
Process: Multiplication

A = 00000100 -> 4

B= 00000010 -> 2

Expected Output: 8

Result:



```
Multiplication:

00000010 -> 2
00000100 -> 4 ... Expected Output 8


11111101 -> -3
00000110 -> 8 ... Expected Output -18

11111000 -> -8
11111110 -> -2 ... Expected Output 16



Division:

00000100 -> 4
00000010 -> 2 ... Expected Output 2

00000110 -> 6
11111101 -> -3 ... Expected Output -2

11111000 -> -8
11111110 -> -2 ... Expected Output 4
```

Test Case 2:

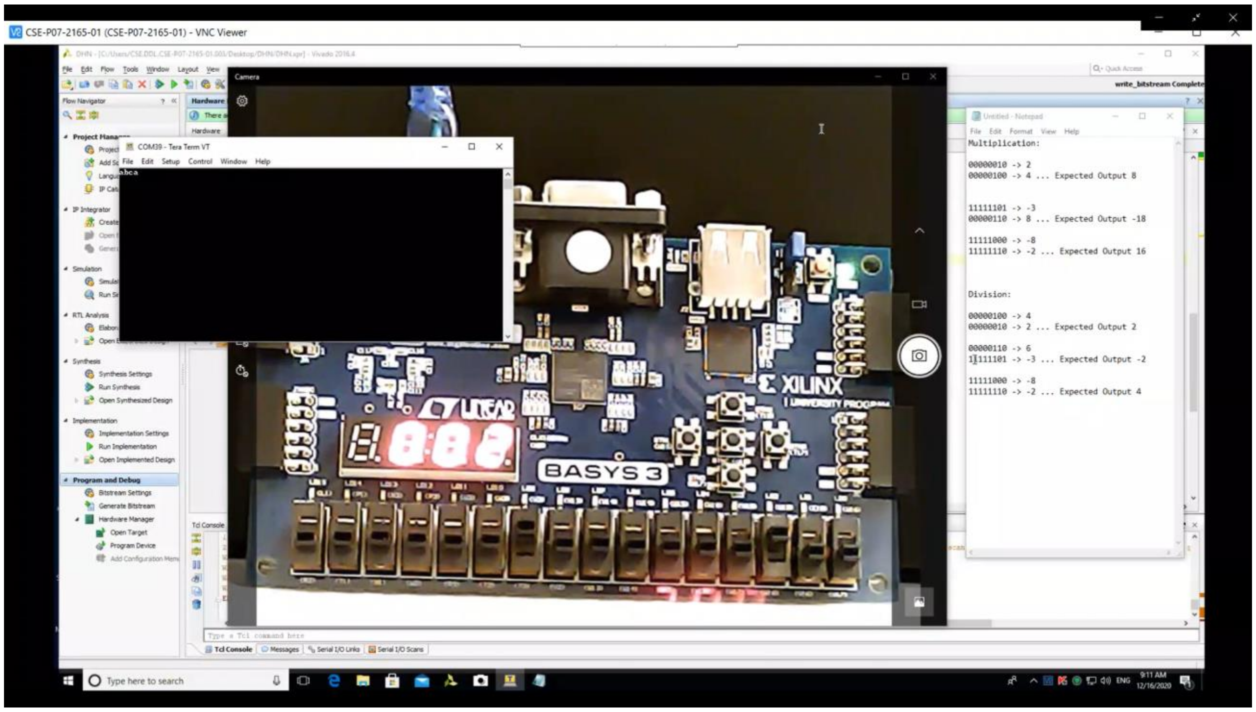
Process: Division

A = 00000100 -> 4

B= 00000010 -> 2

Expected Output: 2

Result:



```
Multiplication:

00000010 -> 2
00000100 -> 4 ... Expected Output 8


11111101 -> -3
00000110 -> 8 ... Expected Output -18

11111000 -> -8
11111110 -> -2 ... Expected Output 16



Division:

00000100 -> 4
00000010 -> 2 ... Expected Output 2

00000110 -> 6
11111101 -> -3 ... Expected Output -2

11111000 -> -8
11111110 -> -2 ... Expected Output 4
```

Test Case 3:

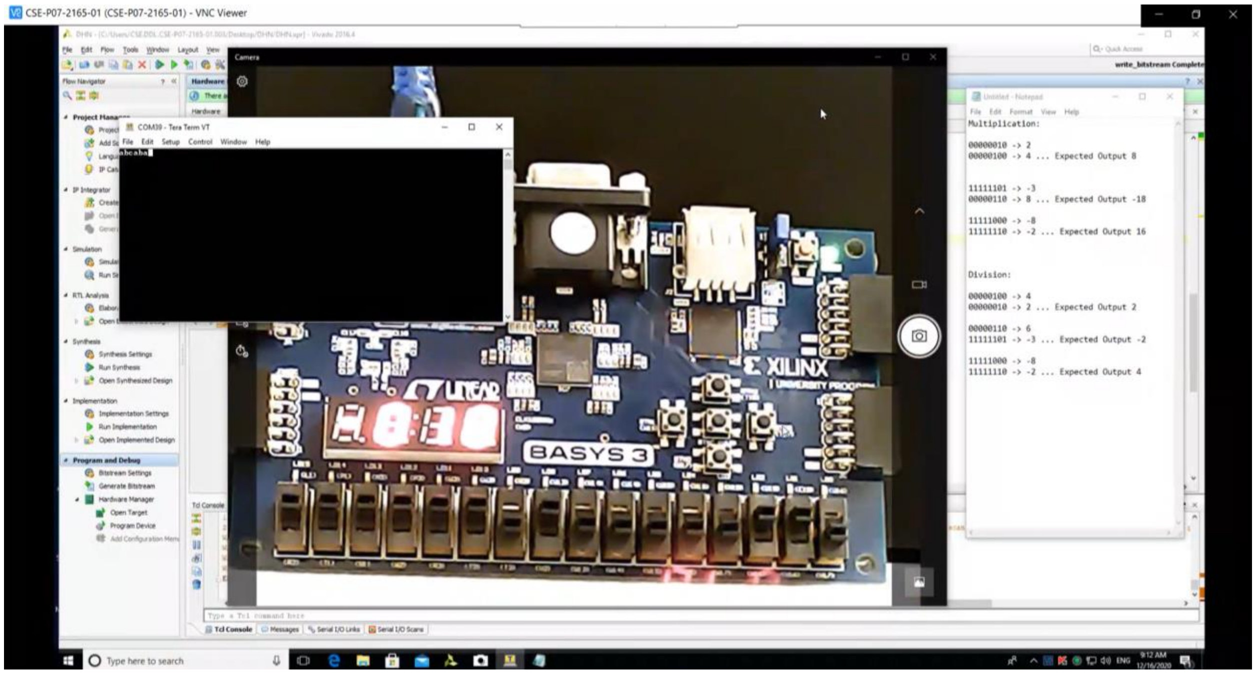
Process: Multiplication

B= 11111101 -> -3

A= 00000110 -> 6

Expected Output: -18

Result:



```
Multiplication:

00000010 -> 2
00000100 -> 4 ... Expected Output 8


11111101 -> -3
00000110 -> 8 ... Expected Output -18

11111000 -> -8
11111110 -> -2 ... Expected Output 16



Division:

00000100 -> 4
00000010 -> 2 ... Expected Output 2

00000110 -> 6
11111101 -> -3 ... Expected Output -2

11111000 -> -8
11111110 -> -2 ... Expected Output 4
```

Test Case 4:

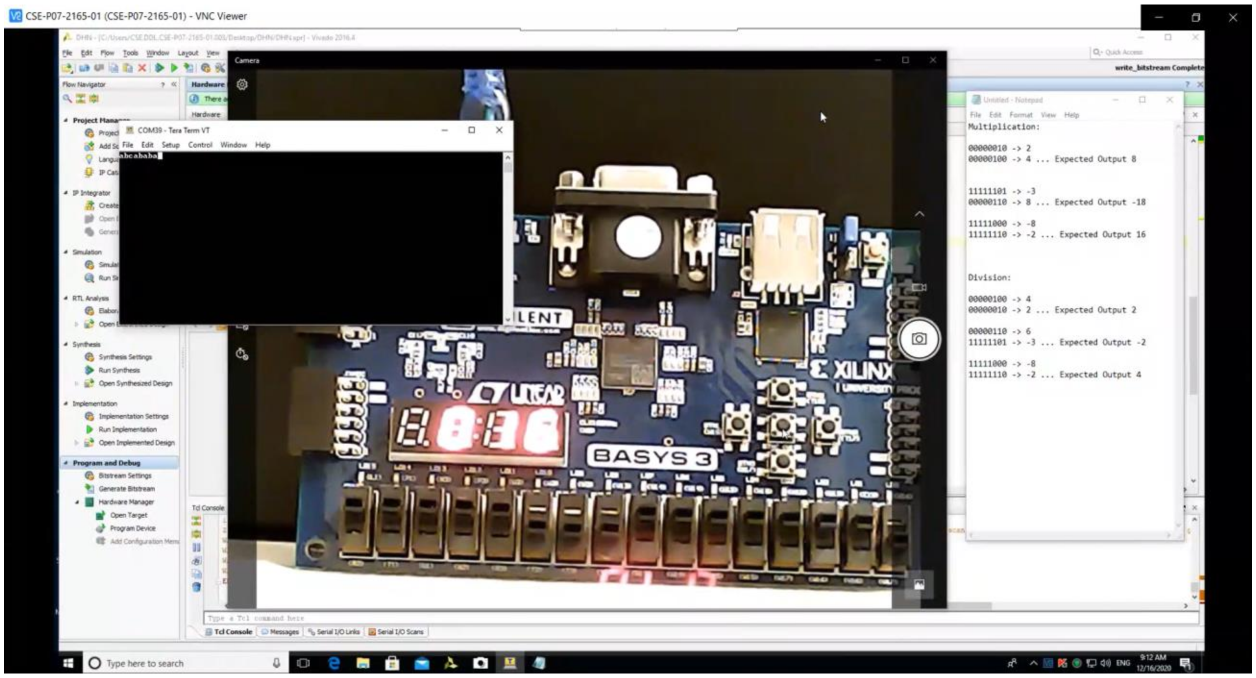
Process: Multiplication

A= 11111000 -> -8

B= 11111110 -> -2

Expected Output: 16

Result:



```
abcababa
```

```
Multiplication:

00000010 -> 2
00000100 -> 4 ... Expected Output 8


11111101 -> -3
00000110 -> 8 ... Expected Output -18

11111000 -> -8
11111110 -> -2 ... Expected Output 16



Division:

00000100 -> 4
00000010 -> 2 ... Expected Output 2

00000110 -> 6
11111101 -> -3 ... Expected Output -2

11111000 -> -8
11111110 -> -2 ... Expected Output 4
```

Test Case 5:

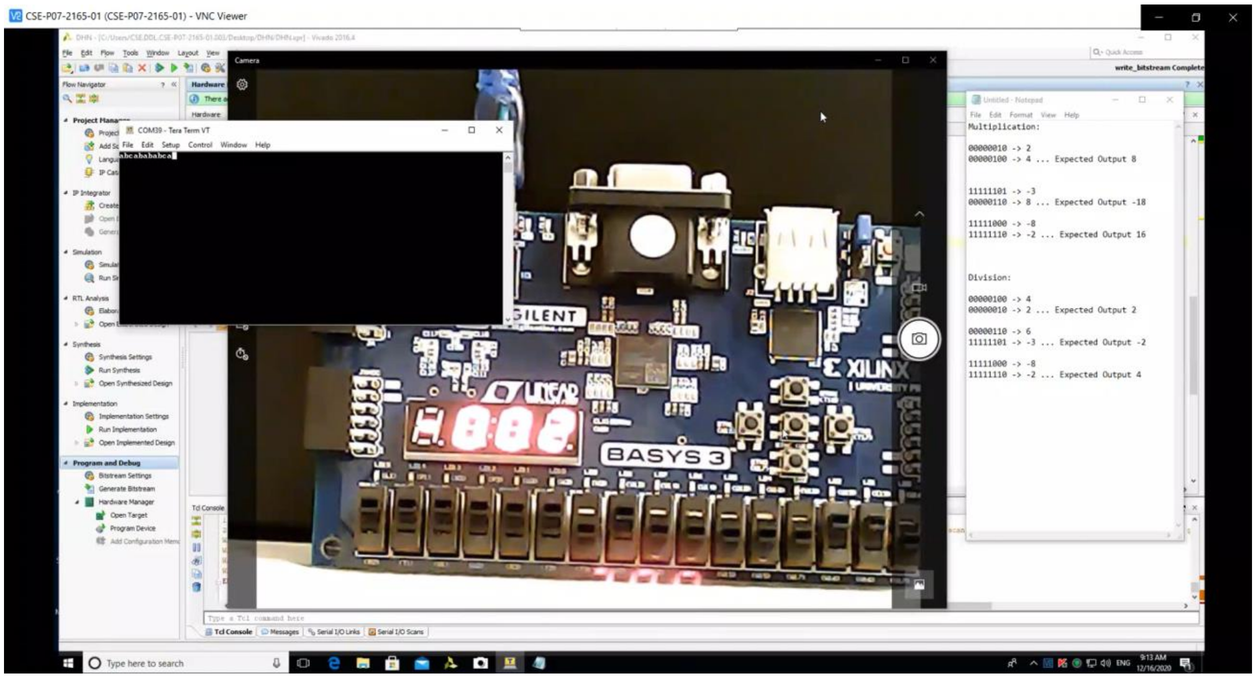
Process: Division

A= 00000110 -> 6

B= 11111101 -> -3

Expected Output: -2

Result:



```
Multiplication:

00000010 -> 2
00000100 -> 4 ... Expected Output 8


11111101 -> -3
00000110 -> 8 ... Expected Output -18

11111000 -> -8
11111110 -> -2 ... Expected Output 16



Division:

00000100 -> 4
00000010 -> 2 ... Expected Output 2

00000110 -> 6
11111101 -> -3 ... Expected Output -2

11111000 -> -8
11111110 -> -2 ... Expected Output 4
```

Test Case 6:

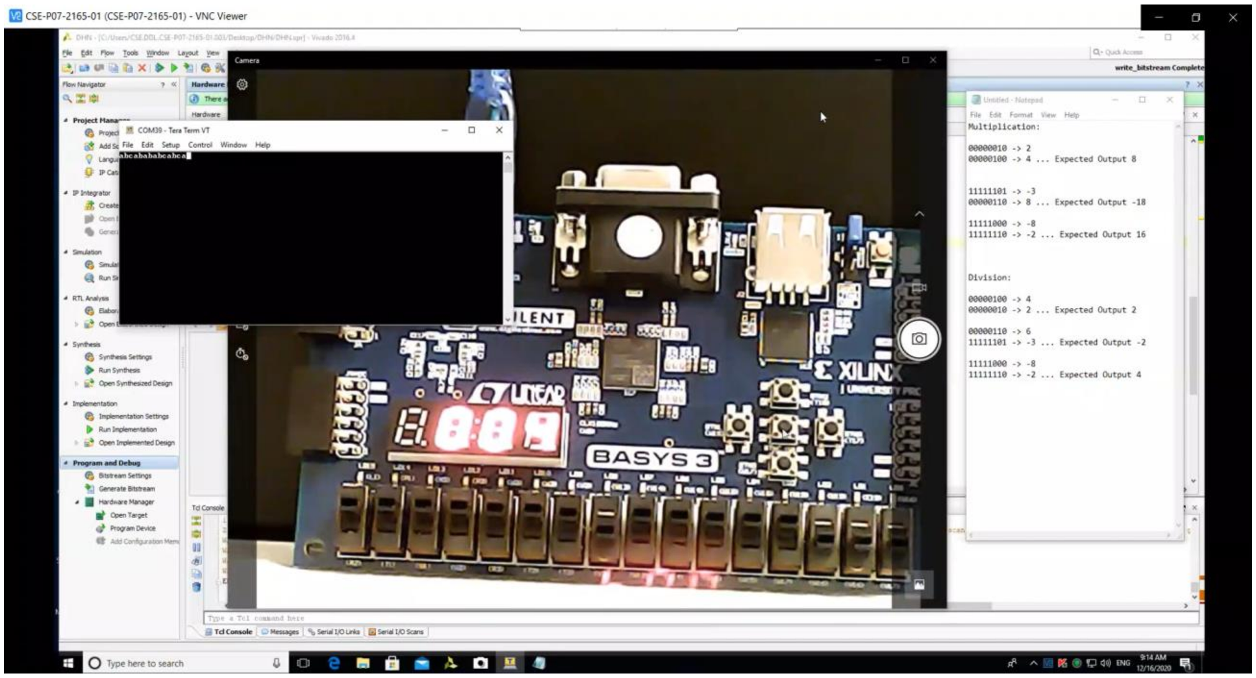
Process: Division

A= 11111000 -> -8

B= 11111110 -> -2

Expected Output: 4

Result:



```
abcababababcabca
```

```
Multiplication:

00000010 -> 2
00000100 -> 4 ... Expected Output 8


11111101 -> -3
00000110 -> 8 ... Expected Output -18

11111000 -> -8
11111110 -> -2 ... Expected Output 16


Division:

00000100 -> 4
00000010 -> 2 ... Expected Output 2

00000110 -> 6
11111101 -> -3 ... Expected Output -2

11111000 -> -8
11111110 -> -2 ... Expected Output 4
```