Digital System Design Applications

Final Project Report

(Project Number 4)

Hasan Enes Şimşek

040160221

08.02.2021

1. About the Project

> Using one of the methods described at [3], calculate $C = \frac{A}{B}$ where A is a 8-bit positive integer and B is a 4-bit positive integer. There will be two 8-bit positive integer outputs as quotient, Q, and remainder, R.

According to the reference given in the project instruction above, I used the algorithm as seen in the [1, Fig. 1] where N is dividend, D is divider, Q is quotient, R is remainder. I will increase quotient +1 every time I subtract divider from dividend. In the final, last result of the subtraction will be remainder and iteration number will be quotient as I increase every iterations.

```
function divide_unsigned(N, D)
  Q := 0; R := N
  while R ≥ D do
    Q := Q + 1
    R := R - D
  end
  return (Q, R)
end
```

Figure 1: A Division Algorithm for Unsigned Binary Numbers.

2. Pseudo Code of the algorithm

Before explaining Algorithmic State Machine design, it is good to produce pseudo code that is more abstract then ASM.

```
Get N to reg15
reg 14 = 0 //reset Q
Get_Divider:
    Get D to reg1
    Reg1<<1
    if Z=1,
        reg0 = 0 // Control “divide by zero” condition
    Load 1111_1111 to reg2
    Reg2 = reg2 + reg1
    Load 0000_0001 to reg1
    Reg2 = reg2 + reg1
    Move reg15 to reg1
    Calculate reg1+reg2
    If C=1,
        reg15 = reg1
        Goto cont: //continue subtraction
    Else If C=0,
        reg1 = reg15
```

```
        Goto stop: //stop process
Stop:
    Reg0 = reg14
Cont:
    Reg1 = reg14
    Load 0000_0001 to reg2
    Reg14 = reg1 + reg2
    Goto Get_Divider
```

Code 1: Pseudo Code of the CU

I will explain details in ASM.

3. Algorithmic State Machine

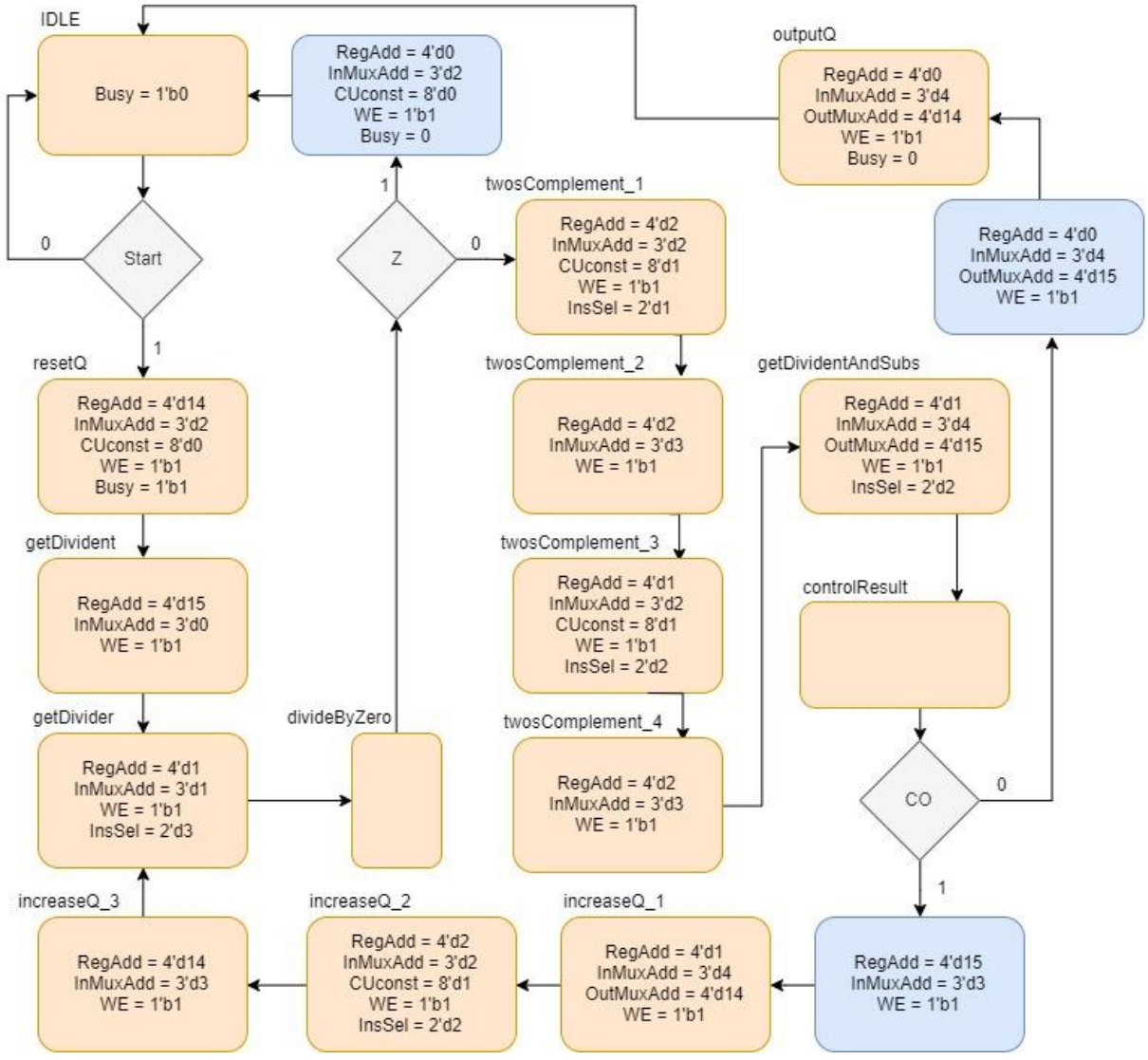

Figure 2: ASM of the Designed CU

In the **IDLE** state, circuit is ready to get inputs and start signal to operate.

After getting **Start** signal, circuit goes **resetQ** state.

In the **resetQ** state, Q(quotient) is set to zero. Register14 is reserved for Q.

The, with **getDividend** state, dividend is taken as external input from **inA.** Register15 is reserved for remainder. In the final, this register becomes remainder instead of dividend.

Like dividend, divider is taken as external input from **inB** in the state **getDivider.** And circular left shift operation is done with divider.

If the **Z flag** is 1, then ASM goes idle state while producing output zero. (zero divider error)

If the **Z flag** is 0, then ASM goes twosComplement_1 state.

In the **twosComplement_1, _2, _3 and _4 states,** I am calculating N = N - D operation. However, designed ALU has only adder circuit, so firstly I need to find two's complement of D and I should sum with N.

More specifically, in the state **twosComplement_1,** NOT D operation is done. However, there is no NOT gate in the ALU, so (D XOR 1111_1111) is used instead.

In the **twosComplement_2,** output of the operation coming from ALU, is written into register 2.

In the **twosComplement_3,** calculated NOT D is summed with number 1 to get two's complement of D. (two's comp of D = NOT D +1)

In the **twosComplement_4,** output of the operation coming from ALU, is written into register 2.

In the **getDividentAndSubs,** N = N – D operation is calculated as summing N and two's comp. of D. Firstly N is getting from reserved register15 to reg1 and summed with –D in the reigister1.

In the **controlResult**, **if CO flag is 1,** calculation is continue with increaseQ registers**(N=>D).** Output of the sum is written into reg 15.

- **increaseQ_1 state:** previous Q in the reserved register 14 is written into reg1
- **increaseQ_2 state:** previous Q is summed with 1
- **increaseQ_3 state:** increased Q is written into reserved register14. Next state is getDivider. So calculation is continuing. Circuit will calculate N = N –D again.

If **CO flag is 0(N<D),** calculation is done. Output is not written. Previous output is written into reg1 in order to generate external output. Thanks to this state, **remainder stays 1 clock cycle in the output port.** Then it goes to **outputQ** state.

- In the **outputQ** state, register 14 is written into register1 to generate **quotient** in output. Next state is idle state so output stays until new start signal has come.

Note: In the project instructions, dividend is set as 8-bit unsigned, divisor is set as 4-bit unsigned integer. I use 4-bit zero padding for divisor.

4. RTL Schematics

4.1 Top Module

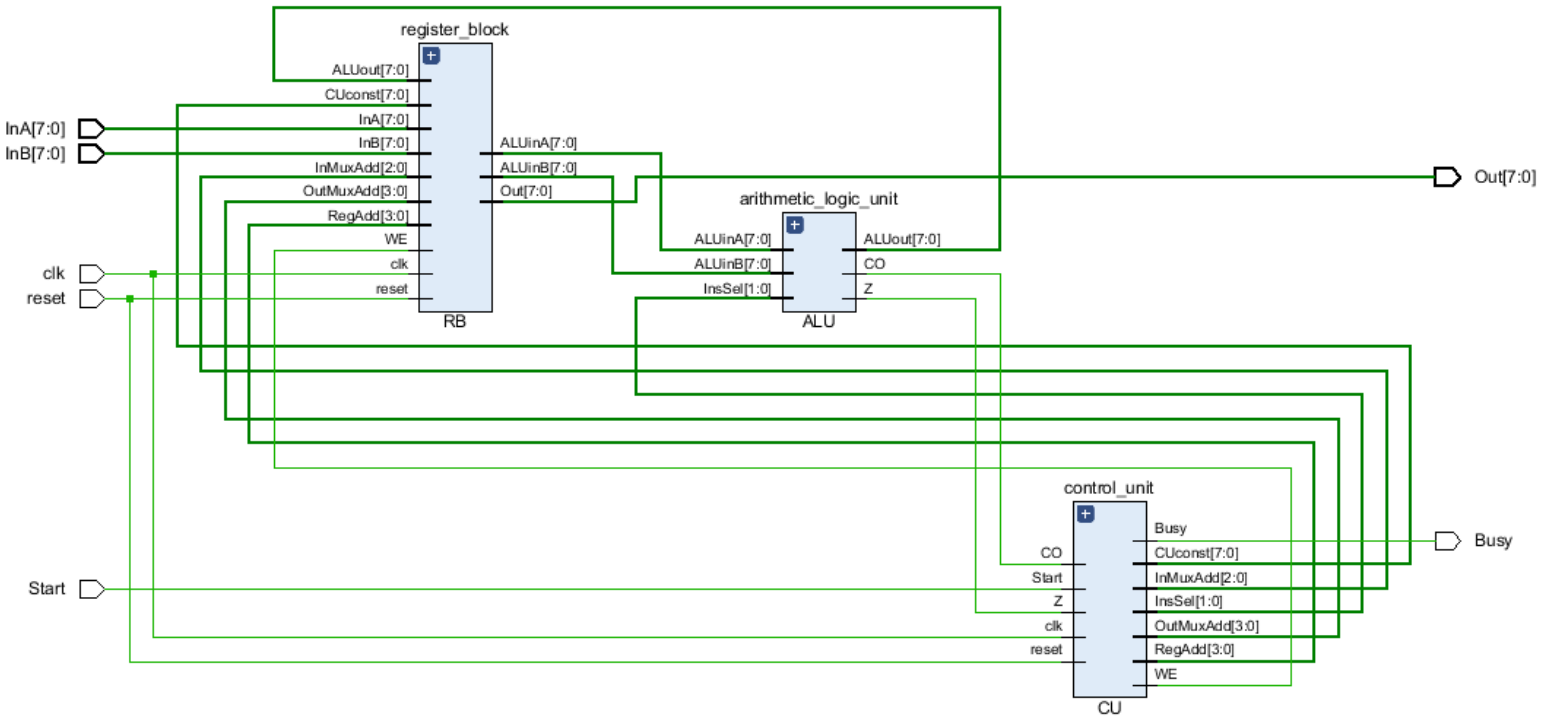

Figure 3: RTL Schematic of the Top Module

Name	Slice LUTs (63400)	Slice Registers (126800)	F7 Muxes (31700)	F8 Muxes (15850)	Bonded IOB (210)	BUFGCTRL (32)
TOP	315	133	24	8	28	1
arithmetic_logic_unit (ALU)	67	0	0	0	0	0
control_unit (CU)	51	5	0	0	0	0
register_block (RB)	184	128	24	8	0	0

Figure 4: Utilization Summary

4.2 ALU

In the 8-bit adder of the ALU, ripple carry adders are used.

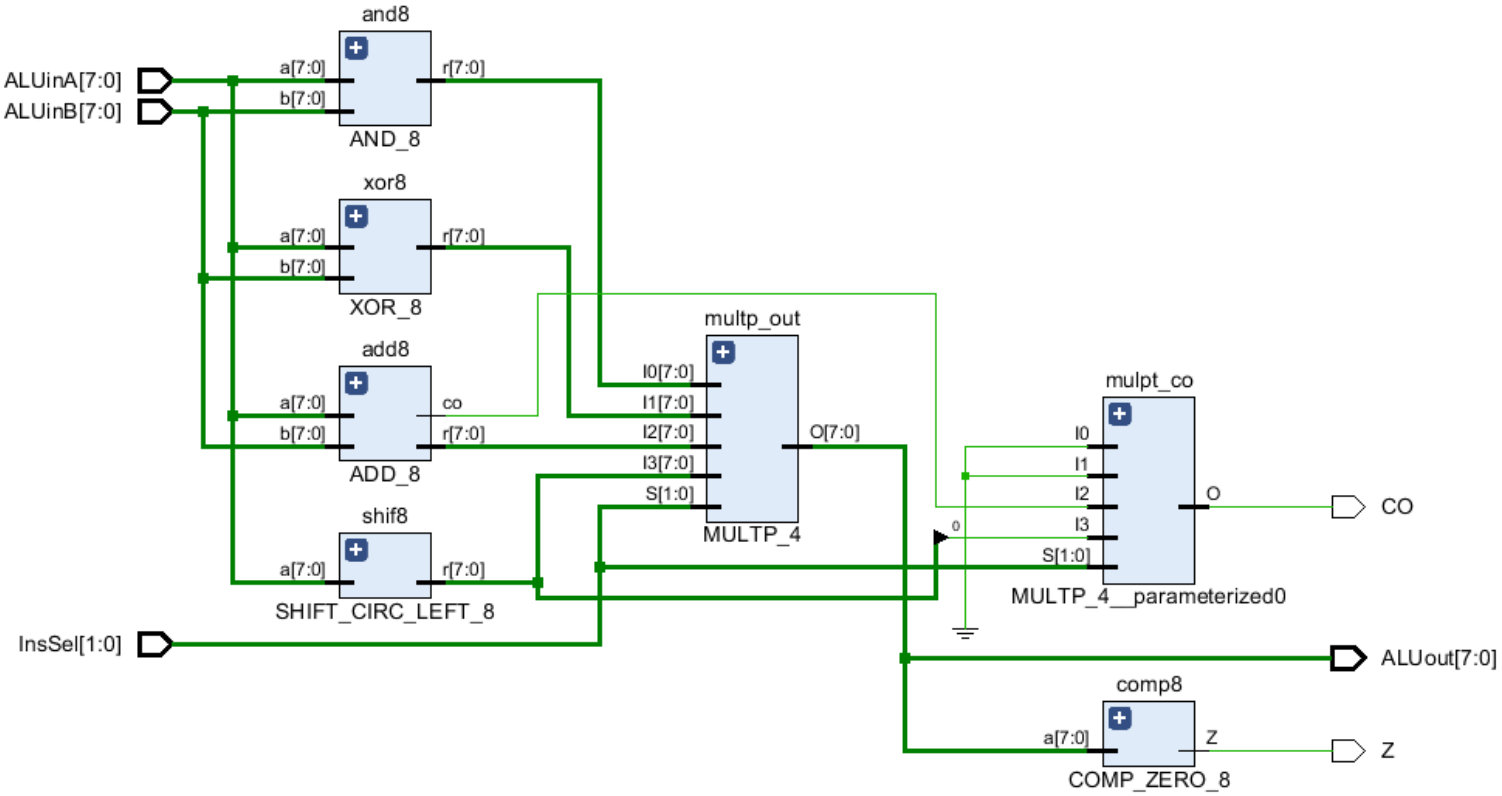

Figure 5: RTL Schematics of the ALU

4.3 Register Block

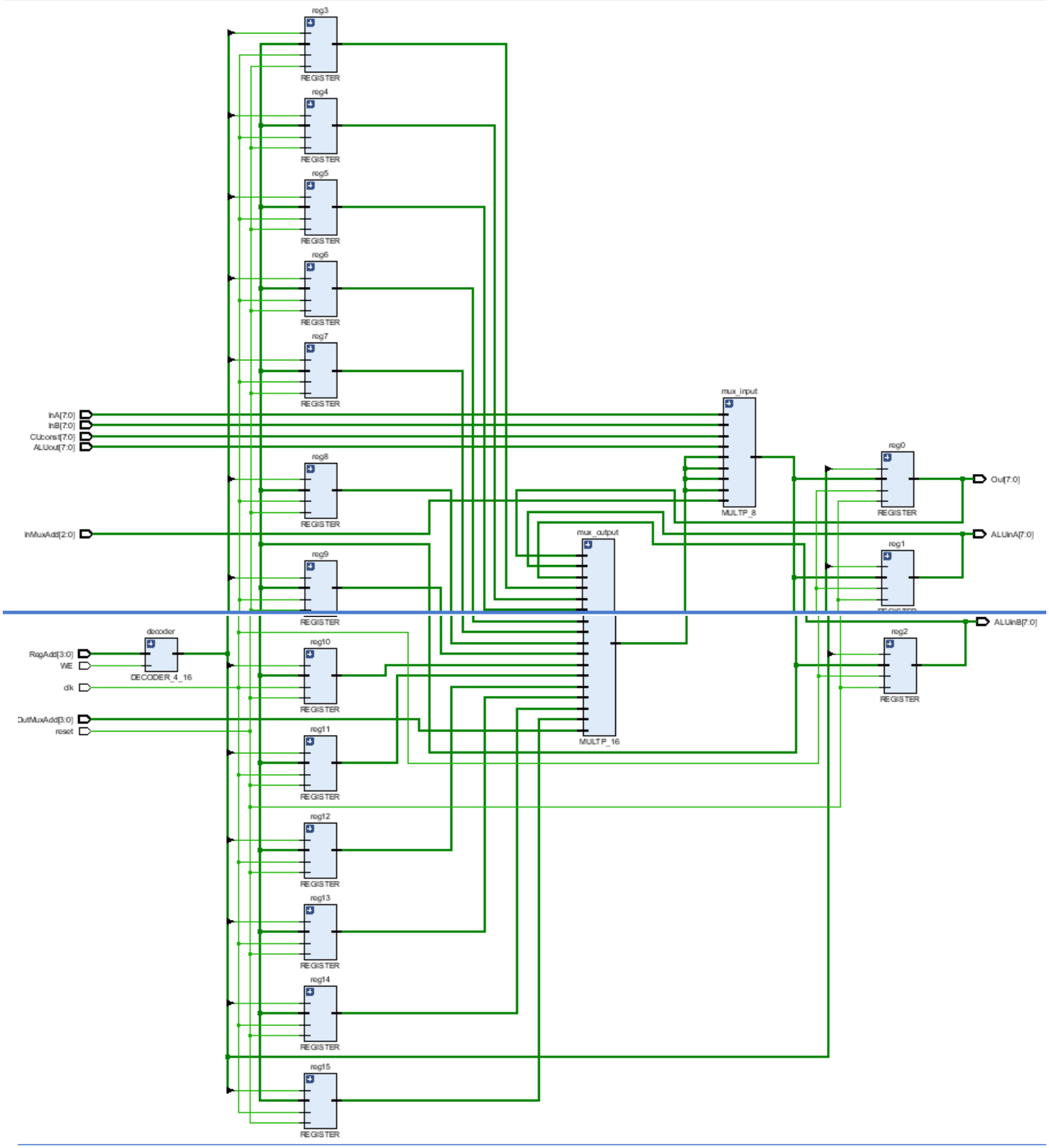

Figure 6: RTL Schematics of the Register Block

4.4 Control Unit (CU)

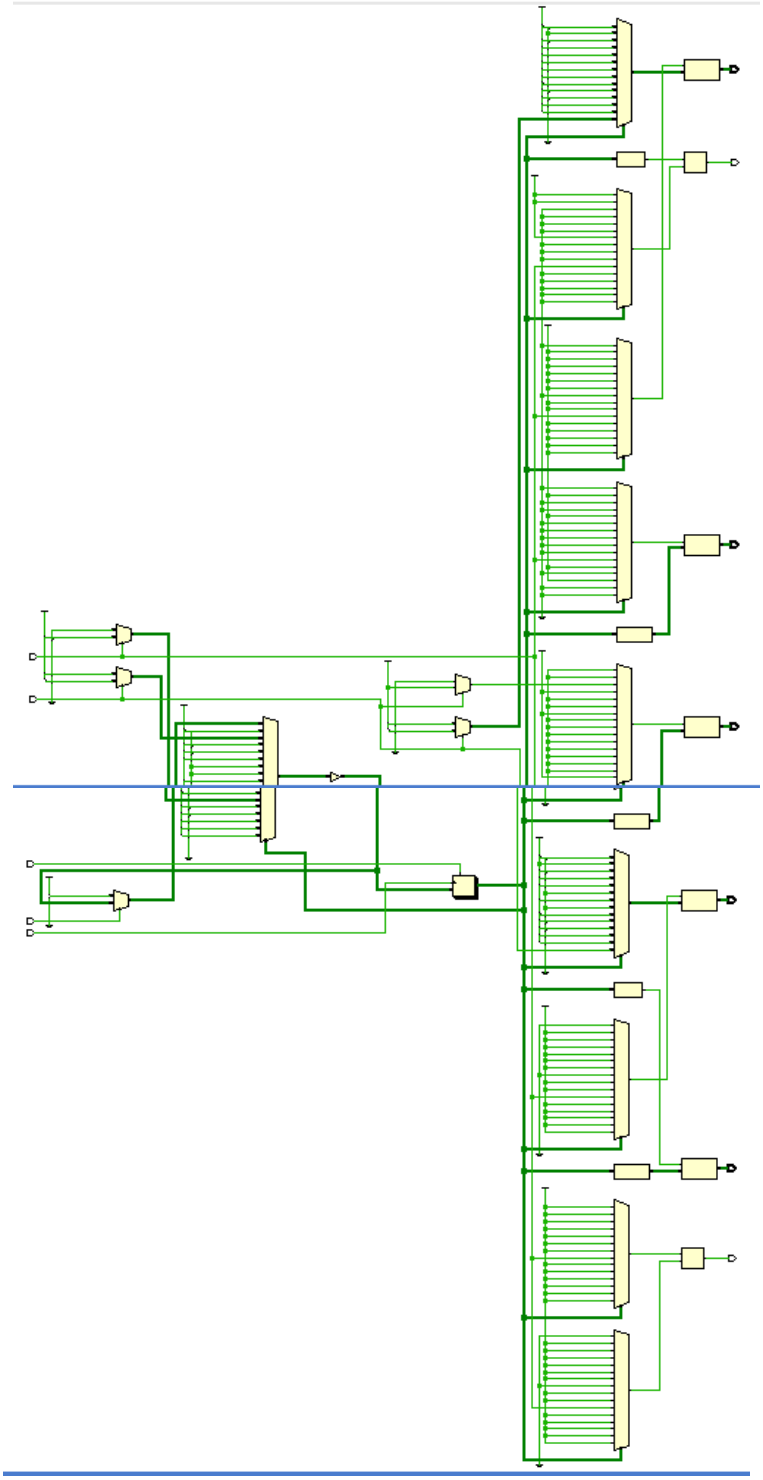

Figure 6: RTL Schematics of the CU

5. Simulation Results

5.1 Ordinary Numbers

In the figure 7, random ordinary numbers, 17/8 is calculated. 17/8= 2*8 +1

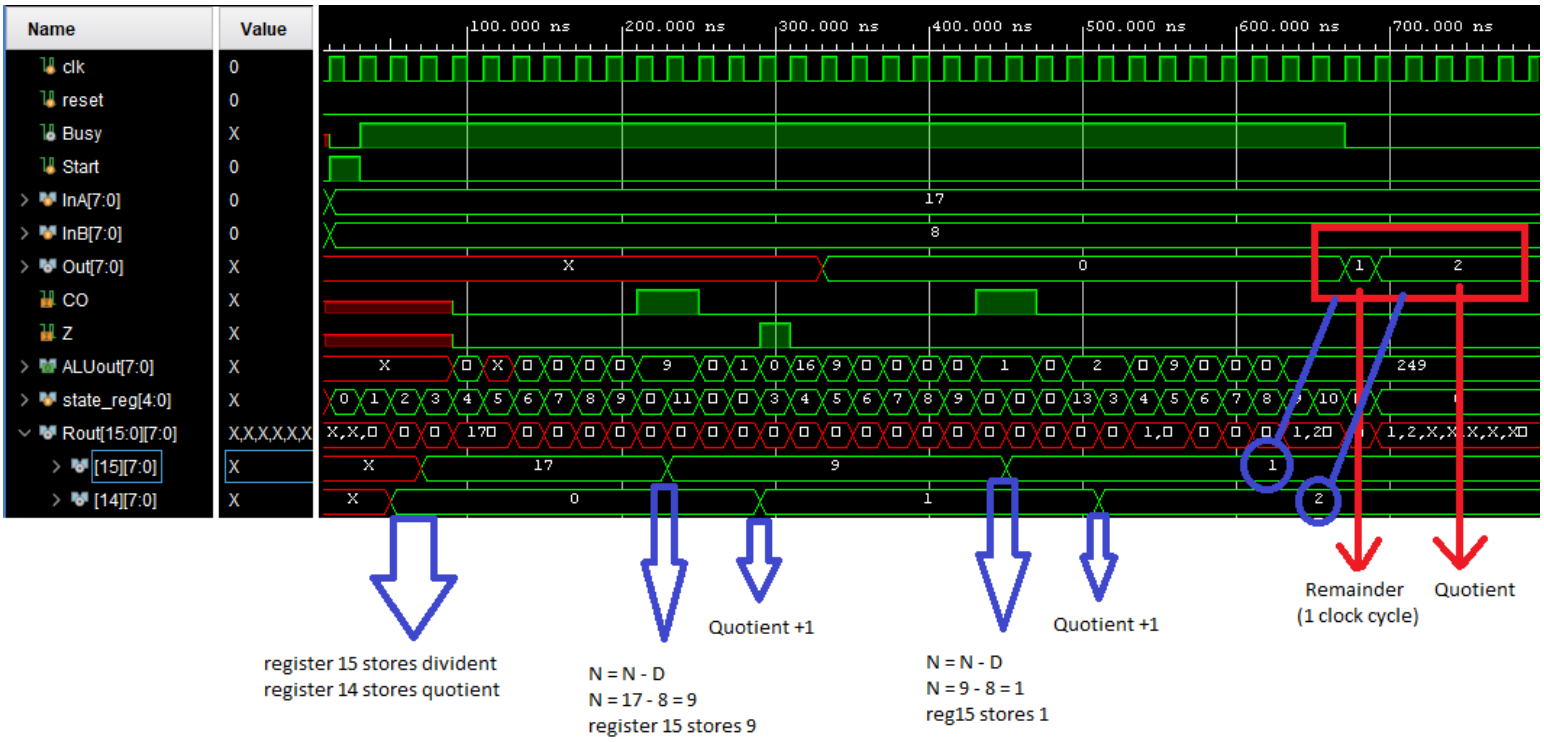

Figure 7: Behavioral Simulation of Two Random Numbers

5.2 The case divident is less than divider

In the figure 8, 12/15 is calculated. 12/15 = 0*15+12

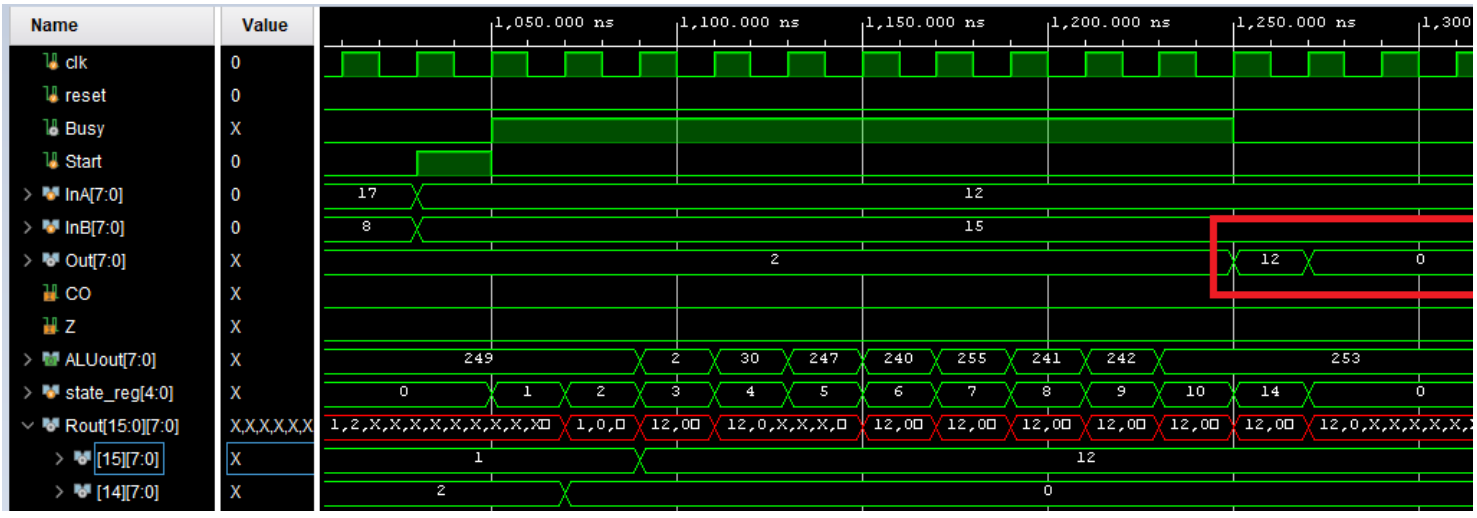

Figure 8: Behavioral Simulation of Special Case

5.3 The case divident is zero

0/13 = 0*13 + 0

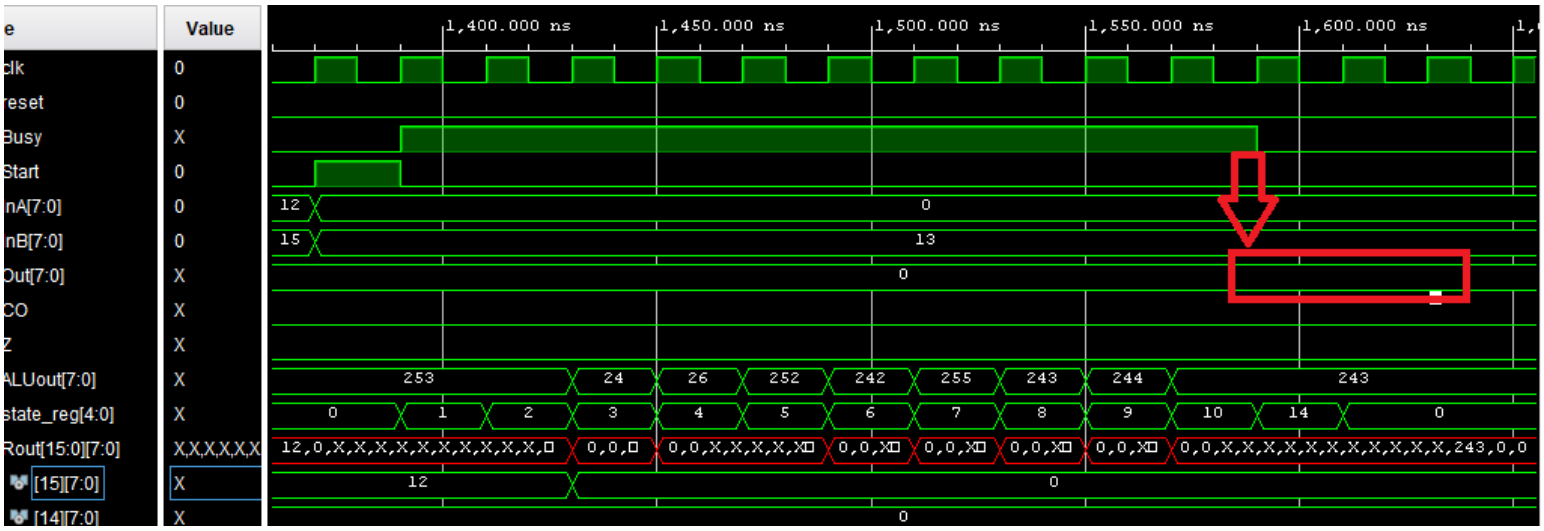

Figure 9: Behavioral Simulation of Special Case Zero Divident

5.4 The case divider is zero

Division with zero is not defined. There is no external flag as output in the design to warn user. Therefore, I chose zero as undefined output.

40/0 = 0 (actually undefined)

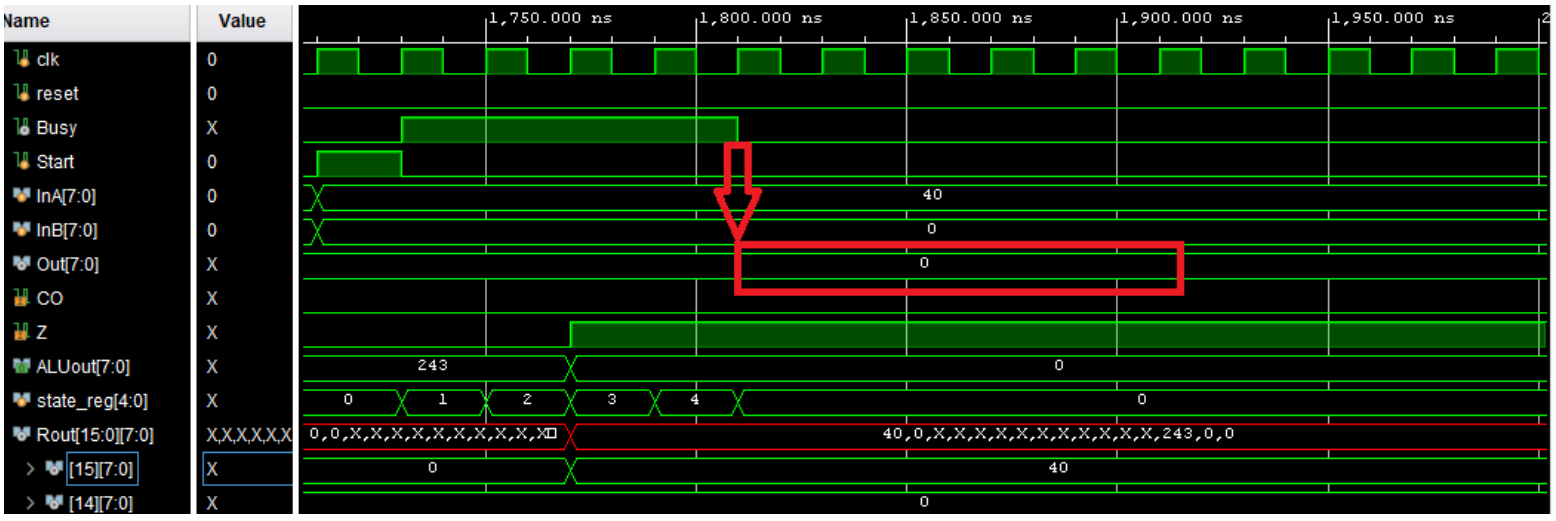

Figure 10: Behavioral Simulation of Special Case Zero Divider

5.5 The case both dividend and divider is zero

0/0 is not defined. Therefore as I did in the section 5.4, I chose zero as output.

0/0 = 0 (actually undefined)

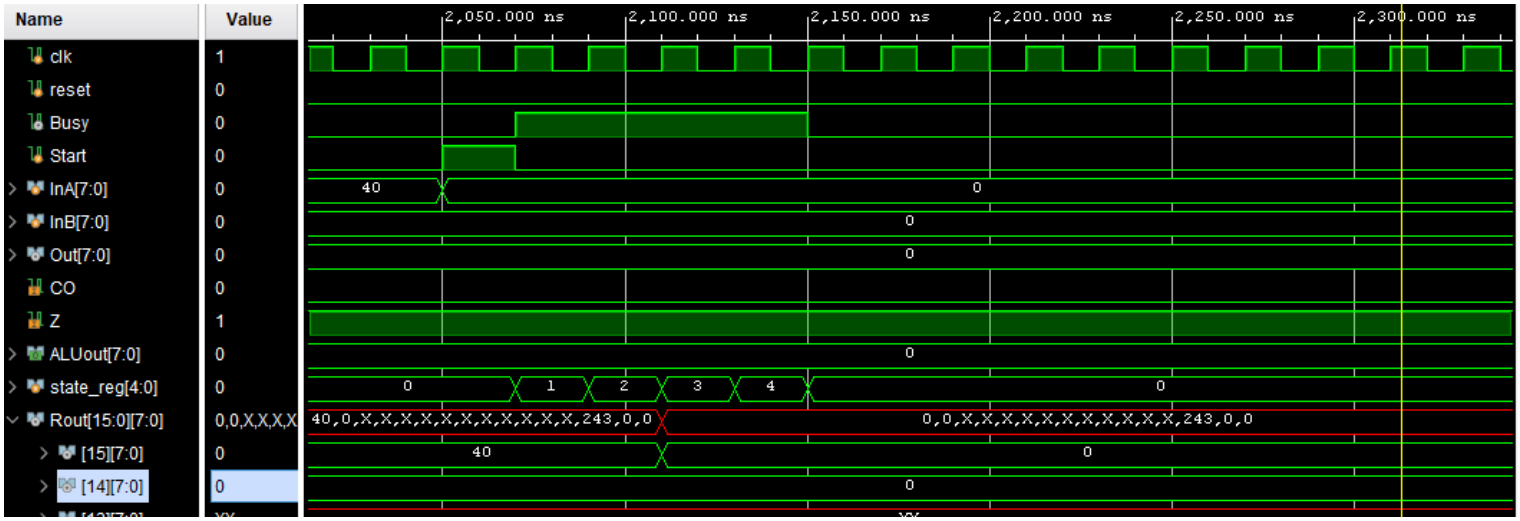

Figure 11: Behavioral Simulation of Special Case 0/0

5.6 Interrupted Calculation

In this section, I investigate reset signal. While a calculation is running, I sent the reset signal to stop running calculation and then I start again with different input numbers.

Firstly 213/8 calculation is given to the circuit and stopped with reset signal.

Then 13/5 calculation is done as 2*5+3

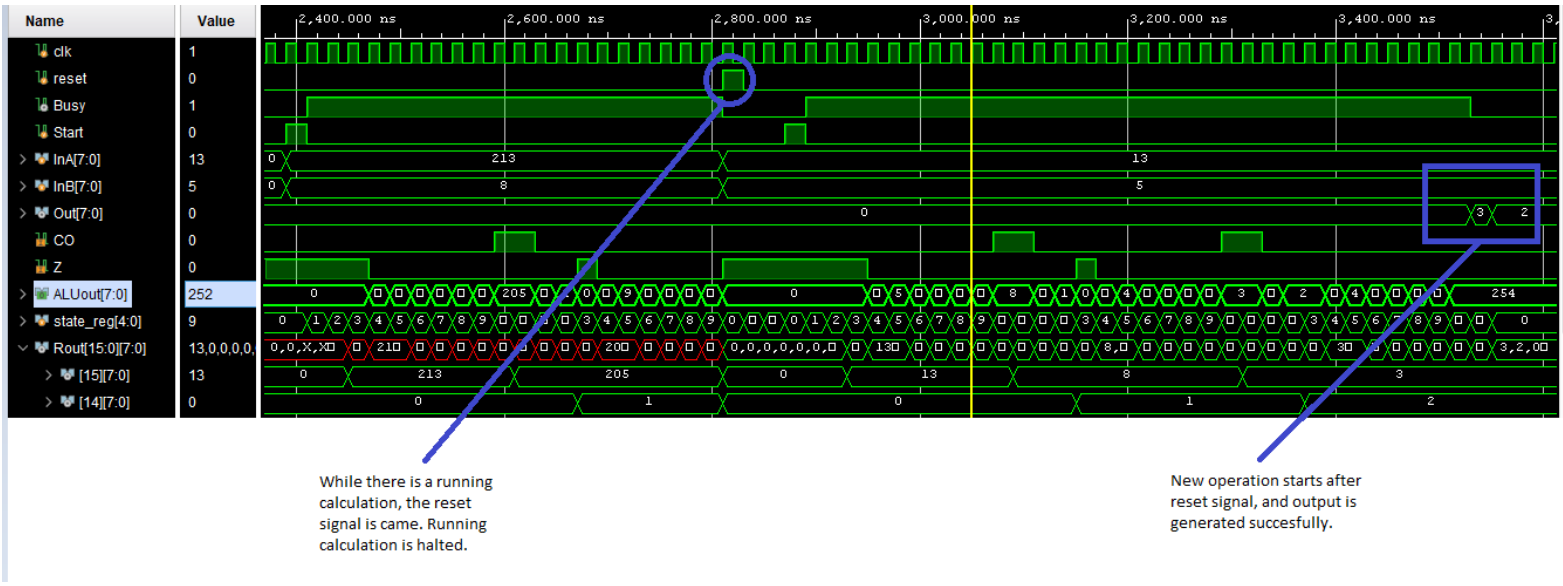

Figure 12: Behavioral Simulation of Special Case Reset Signal

5.7 Max Limits

In the project instructions, max dividend is set 255 as 8-bit unsigned, max divisor is set to 15 as 4-bit unsigned integer. I use 4-bit zero padding for divisor.

255/15 = 17*15+0

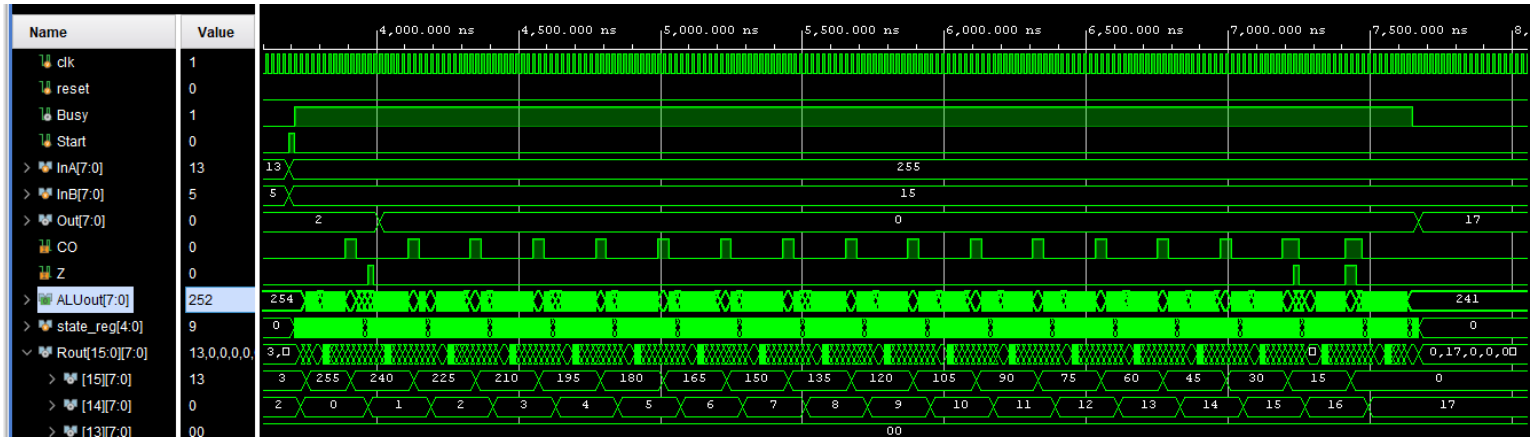

Figure 13: Behavioral Simulation of Special Case Max Inputs

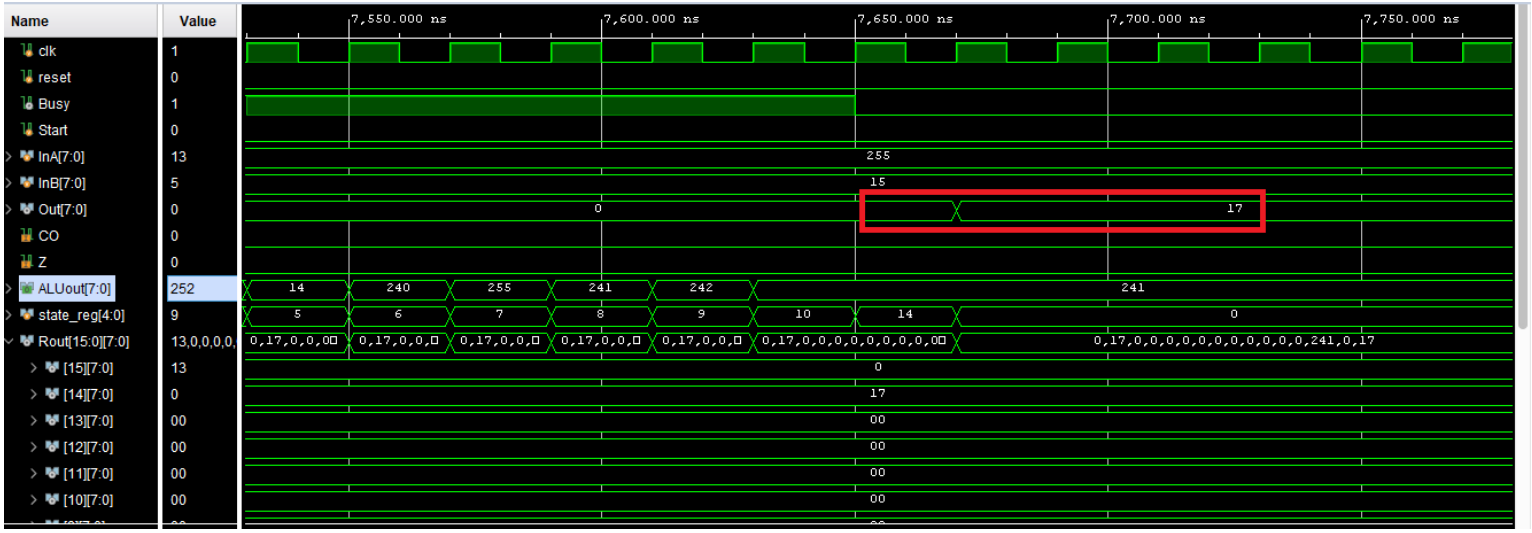

Figure 14: Zoomed version of figure 14 to see outputs clearly.

6. References

[1] “Division Algorithm”, Accessed on: Feb. 9, 2021. [Online]. Available: https://en.wikipedia.org/wiki/Division_algorithm