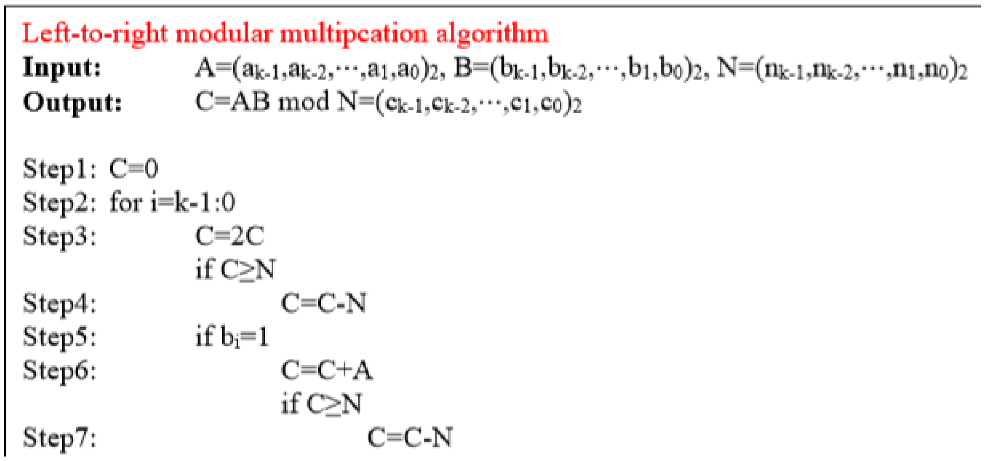
Gerald Topalli
040160911

Homework 3

In this homework we had to build a microcontroller whose functionality is modular multiplication. The algorithm that implements this machine is shown in Figure 1. This algorithm can very easily be used to produce an ASM chart which can very well show the design flow. The ASM was prepared for us and we just had to implement it. The ASM chart is shown in Figure 2.



Left-to-right modular multipcation algorithm
Input: $A=(a_{k-1},a_{k-2},\cdots,a_1,a_0)_2$, $B=(b_{k-1},b_{k-2},\cdots,b_1,b_0)_2$, $N=(n_{k-1},n_{k-2},\cdots,n_1,n_0)_2$
Output: $C=AB \bmod N=(c_{k-1},c_{k-2},\cdots,c_1,c_0)_2$

```
Step1: C=0
Step2: for i=k-1:0
Step3:         C=2C
               if C≥N
Step4:                C=C-N
Step5:         if b_i=1
Step6:                C=C+A
                      if C≥N
Step7:                       C=C-N
```

Figure 1

In my opinion, the most difficult think we have to encounter in ASM charts are the decision making blocks. Those are particularly difficult because we have to make sure the proper input signal is coming from the data path circuit, and also we need to make sure that we give enough time to the data path to produce the correct output values which in this case are the Status_Signals. For this purpose, after each step in ASM chart a'\nd before each decision block in ASM chart, I have put some delay (1 clock cycle) just to ensure that there is enough time for the data path to create the appropriate Status_Signals from the supplied input Control_Signals.

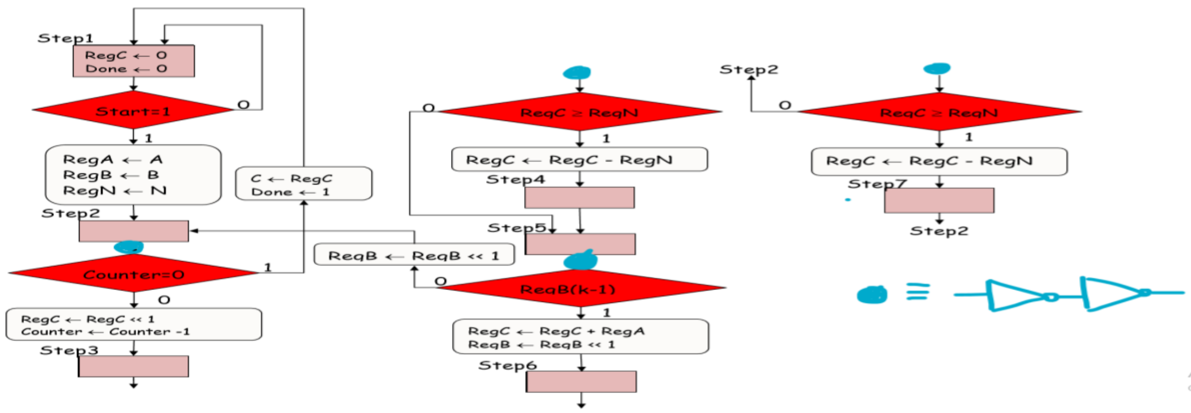


Figure 2

The blue circles are just some synthesizable delay which I am showing as two inverter buffers. I am going to implement those delays by using some extra states. Those are very necessary blocks as I mentioned above since the data path requires some time to settle and if we rush it, we will have wrong outputs.

Controller Circuit 1.a

Please find the Controller code in the zipped file under the name Controller.v. I have briefly explained the lines I wrote for the Controller block in the Controller.v file. The Controller block is a mix type of machine. Some of the outputs are Moore outputs and some others are Mealy Outputs. In the code, the outputs that are inside the decision states with the extension _d, are Mealy outputs. The rest of the outputs are Moore outputs as they do not depend on the Status_Signal. We can say the Moore Control_Signal bits are preparing the Data_Path to produce the correct output signals. The Moore bits that do that control are S_Coun, S_Comp1, S_Comp2, S_AS1, S_C, AS. The rest of the Control_Signal bits are Mealy outputs. You can reach the Controller.v file to check the code I wrote for the controller.

1.b

Please find the implemented sized circuit in Figure 3.

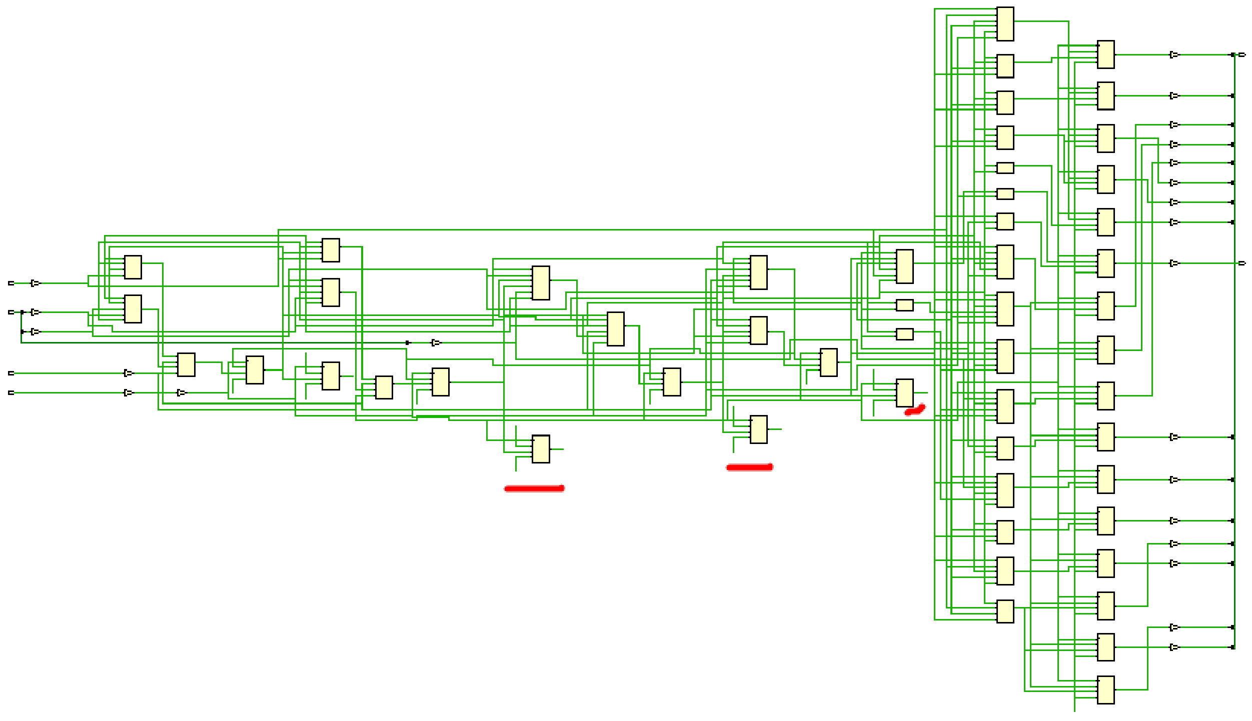
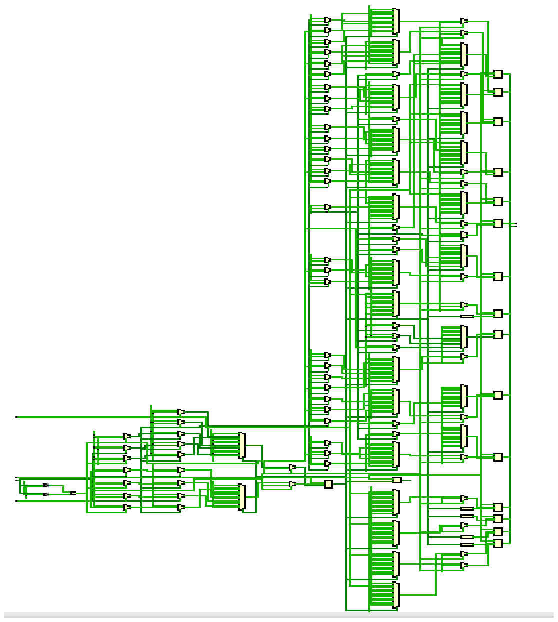
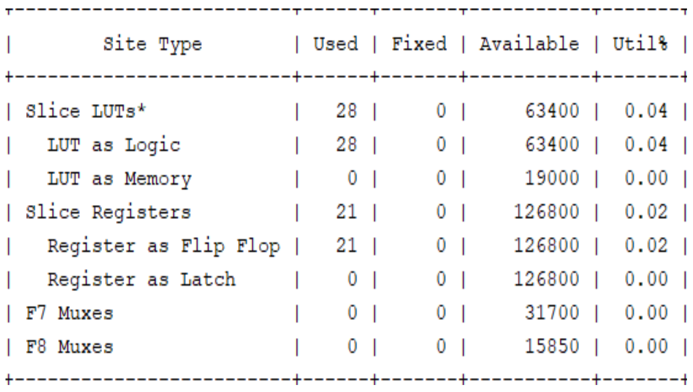


Figure 3

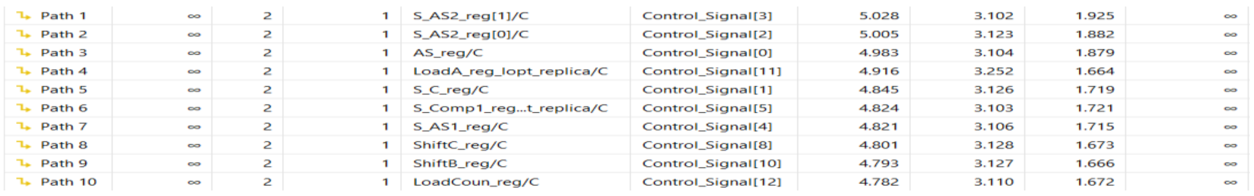
Please see that the underlined flip-flops are the present_state registers. Normally they would be removed but I have used the don't touch command to keep it for testing purposes. They have no practical purpose in the implemented circuit but as I mentioned above I did that for testing and debugging purposes. The RTL schematic, the area utilization and timing summary are shown below:



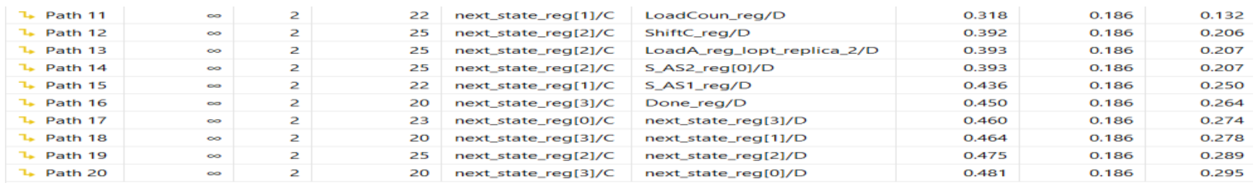


```
+-------------------------+------+-------+-----------+-------+
|        Site Type        | Used | Fixed | Available | Util% |
+-------------------------+------+-------+-----------+-------+
| Slice LUTs*             |   28 |     0 |     63400 |  0.04 |
|   LUT as Logic          |   28 |     0 |     63400 |  0.04 |
|   LUT as Memory         |    0 |     0 |     19000 |  0.00 |
| Slice Registers         |   21 |     0 |    126800 |  0.02 |
|   Register as Flip Flop |   21 |     0 |    126800 |  0.02 |
|   Register as Latch     |    0 |     0 |    126800 |  0.00 |
| F7 Muxes                |    0 |     0 |     31700 |  0.00 |
| F8 Muxes                |    0 |     0 |     15850 |  0.00 |
+-------------------------+------+-------+-----------+-------+
```

Setup delay and Hold delay are shown below:



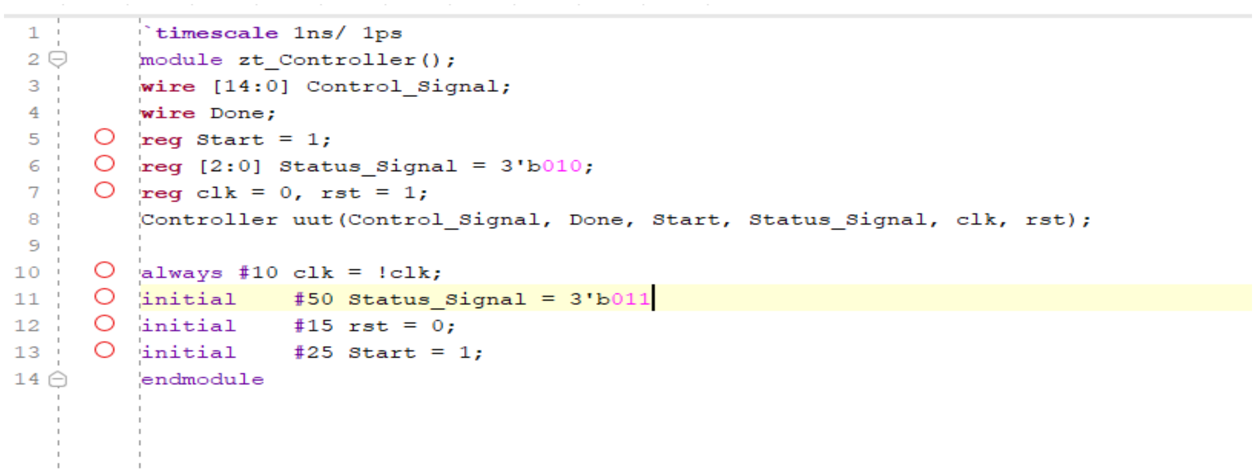
Path 1	∞	2	1	S_AS2_reg[1]/C	Control_Signal[3]	5.028	3.102	1.925	∞
Path 2	∞	2	1	S_AS2_reg[0]/C	Control_Signal[2]	5.005	3.123	1.882	∞
Path 3	∞	2	1	AS_reg/C	Control_Signal[0]	4.983	3.104	1.879	∞
Path 4	∞	2	1	LoadA_reg_lopt_replica/C	Control_Signal[11]	4.916	3.252	1.664	∞
Path 5	∞	2	1	S_C_reg/C	Control_Signal[1]	4.845	3.126	1.719	∞
Path 6	∞	2	1	S_Comp1_reg...t_replica/C	Control_Signal[5]	4.824	3.103	1.721	∞
Path 7	∞	2	1	S_AS1_reg/C	Control_Signal[4]	4.821	3.106	1.715	∞
Path 8	∞	2	1	ShiftC_reg/C	Control_Signal[8]	4.801	3.128	1.673	∞
Path 9	∞	2	1	ShiftB_reg/C	Control_Signal[10]	4.793	3.127	1.666	∞
Path 10	∞	2	1	LoadCoun_reg/C	Control_Signal[12]	4.782	3.110	1.672	∞



Path 11	∞	2	22	next_state_reg[1]/C	LoadCoun_reg/D	0.318	0.186	0.132
Path 12	∞	2	25	next_state_reg[2]/C	ShiftC_reg/D	0.392	0.186	0.206
Path 13	∞	2	25	next_state_reg[2]/C	LoadA_reg_lopt_replica_2/D	0.393	0.186	0.207
Path 14	∞	2	25	next_state_reg[2]/C	S_AS2_reg[0]/D	0.393	0.186	0.207
Path 15	∞	2	22	next_state_reg[1]/C	S_AS1_reg/D	0.436	0.186	0.250
Path 16	∞	2	20	next_state_reg[3]/C	Done_reg/D	0.450	0.186	0.264
Path 17	∞	2	23	next_state_reg[0]/C	next_state_reg[3]/D	0.460	0.186	0.274
Path 18	∞	2	20	next_state_reg[3]/C	next_state_reg[1]/D	0.464	0.186	0.278
Path 19	∞	2	25	next_state_reg[2]/C	next_state_reg[2]/D	0.475	0.186	0.289
Path 20	∞	2	20	next_state_reg[3]/C	next_state_reg[0]/D	0.481	0.186	0.295

1.c

The first thing that I need to do is to reset the device so it goes to state 1. Depending on the hardware, the registers could directly be assigned to 0 or X. In order to avoid this, I will reset the device first to make the next_state = S1, so it continues working as desired (This is how they usually start the devices, by resetting). Then, I would just input some state signals. The testbench I used for the testing of the Controller is shown in
Figure 4.



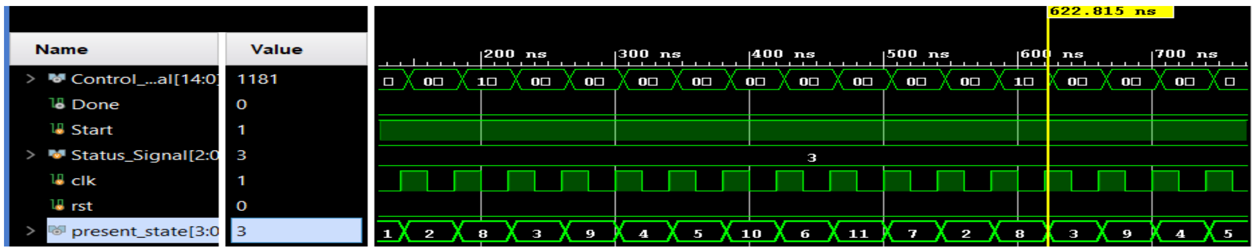
```
`timescale 1ns/ 1ps
module zt_Controller();
wire [14:0] Control_Signal;
wire Done;
reg Start = 1;
reg [2:0] Status_Signal = 3'b010;
reg clk = 0, rst = 1;
Controller uut(Control_Signal, Done, Start, Status_Signal, clk, rst);

always #10 clk = !clk;
initial    #50 Status_Signal = 3'b011
initial    #15 rst = 0;
initial    #25 Start = 1;
endmodule
```

Figure 4

For the particular Status_Signal shown in Figure 4, according to the flow chart the state flow should be the following: (Please see the parameter's values given to states in Controller Block)

S1, S2, S2_d(8), S3, S3_d(9), S4, S5, S5_d(10), S6, S6_d(11), S7, S2….. Repeating pattern. Figure 5 shows the same pattern. Also the control signals are behaving in the manner I want them to work.



Name	Value
Control_...al[14:0]	1181
Done	0
Start	1
Status_Signal[2:0]	3
clk	1
rst	0
present_state[3:0]	3

Figure 5

If I make the status signal equal to 001 when the 3-rd is being executed, I expect the circuit to skip the 4-th and 7-th states. I predict the state flow will be like shown below:

S1, S2, S2_d(8), S3, S3_d(9), S5, S5_d, S6, S6_d(11), S2 Figure 6 shows this state flow.

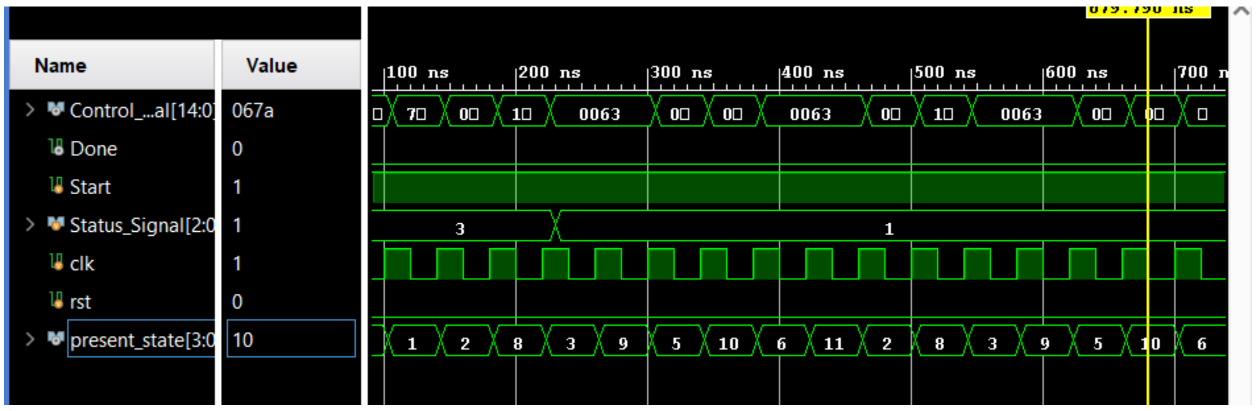


Figure 6

So, until now I have tested the decision block that come after the 3-rd step and the 6-th step. The only decision block that needs to be considered now is the one coming after the 5-th state. Let's make the Status_Signal = 110 at the 5-th step. I predict that the 6-th state and the 7-th state like the following sequence: S1, S2, S2_d(8), S3, S3_d(9), S5, S5_d(10), S2, S2_d, S1 as in Figure 7. So the state flow is same with the prediction. Please note how the done signal turns high this time when enters the state S2_d because of the Status_Signal value. S2_d is a decision state and thus the outputs on this state are Mealy outputs. The previous times Done remained low because the state tested its inputs and the inputs did not satisfy the requirement to make Done = 1;

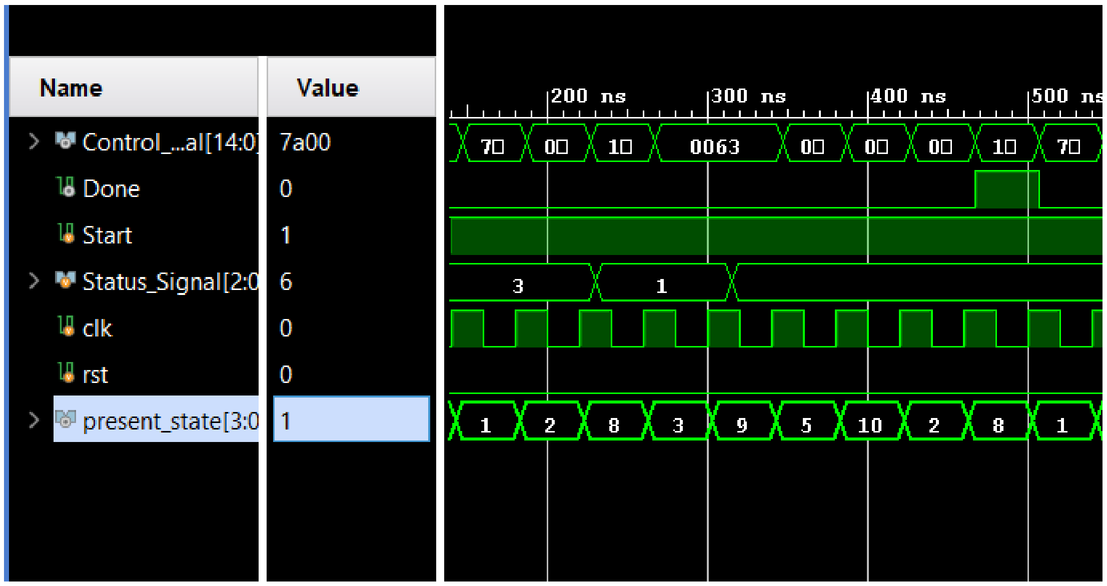


Figure 7

I think there is enough evidence that the controller is working the way we want. Now I will proceed with the Data_Path block.

Data Path circuit

2.a

The Data_Path circuit is shown in the homework zip file under the name Data_Path.v. In order to build it under the requirement that you have set for the bit-length k, I wrote the whole hardware using only 1 module inside which I have built registers, multiplexers, ADD_SUB circuits and one comparator. Please refer to the Data_Path.v file and please check how I build the hardware. Before configuring the hardware, I had to find out what each of the devices in Figure 2 in odev file is. The first column devices are just some registers. The second column's devices are multiplexers and the third column devices are adder-subtract circuit depending on the AS value and a comparator which shows if the Operand1C is greater, equal, greater and equal or smaller than Operand2C

2.b

I made k = 8, and I implemented the circuit. The implement circuit is shown in Figure 8.

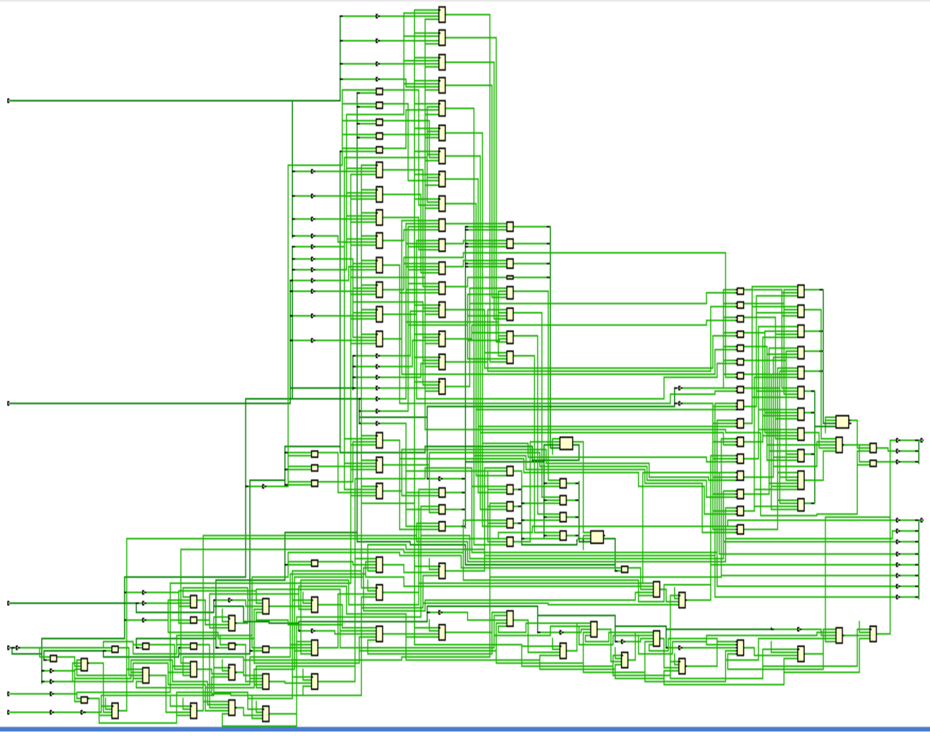


Figure 8

The RTL schematic is shown in Figure 9.

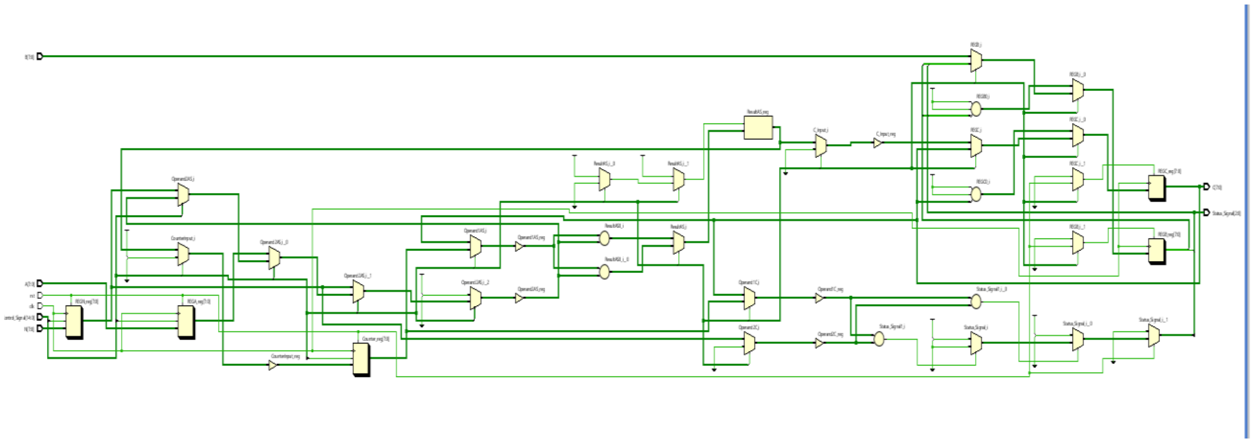
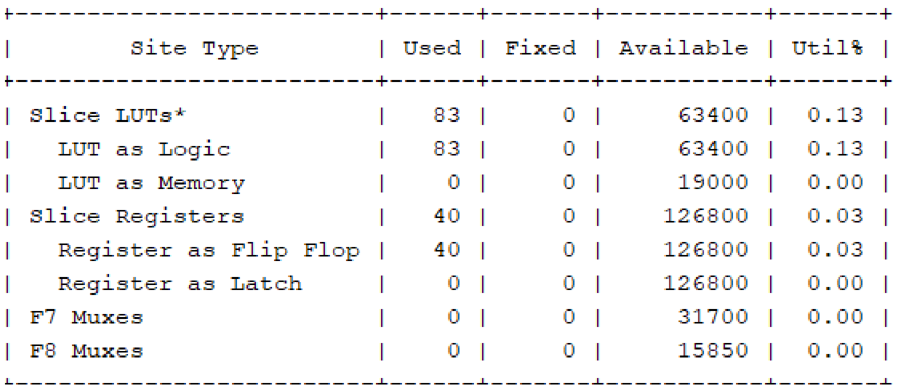


Figure 9

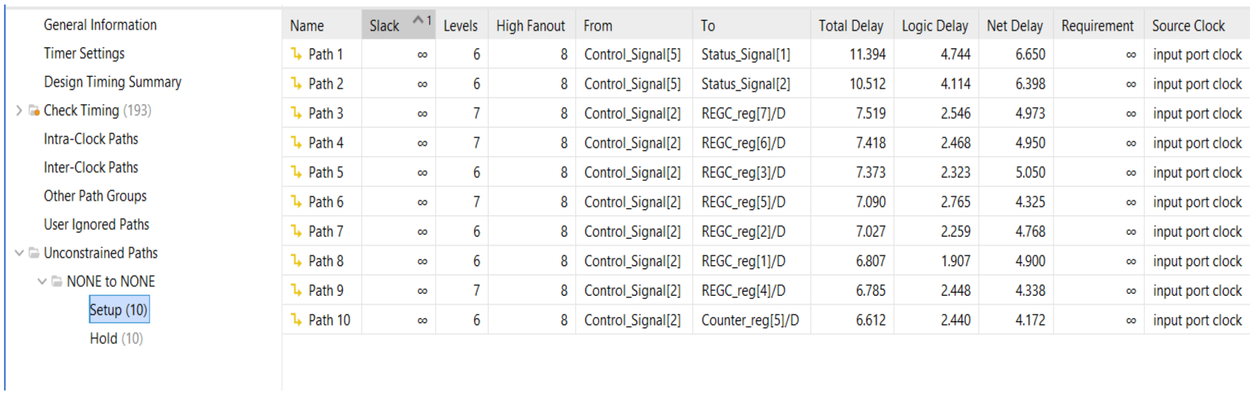
In Figure 10 I am showing the area report for the Data_Path Circuit.



```
+-------------------------+------+-------+-----------+-------+
|        Site Type        | Used | Fixed | Available | Util% |
+-------------------------+------+-------+-----------+-------+
| Slice LUTs*             |   83 |     0 |     63400 |  0.13 |
|   LUT as Logic          |   83 |     0 |     63400 |  0.13 |
|   LUT as Memory         |    0 |     0 |     19000 |  0.00 |
| Slice Registers         |   40 |     0 |    126800 |  0.03 |
|   Register as Flip Flop |   40 |     0 |    126800 |  0.03 |
|   Register as Latch     |    0 |     0 |    126800 |  0.00 |
| F7 Muxes                |    0 |     0 |     31700 |  0.00 |
| F8 Muxes                |    0 |     0 |     15850 |  0.00 |
+-------------------------+------+-------+-----------+-------+
```

Figure 10

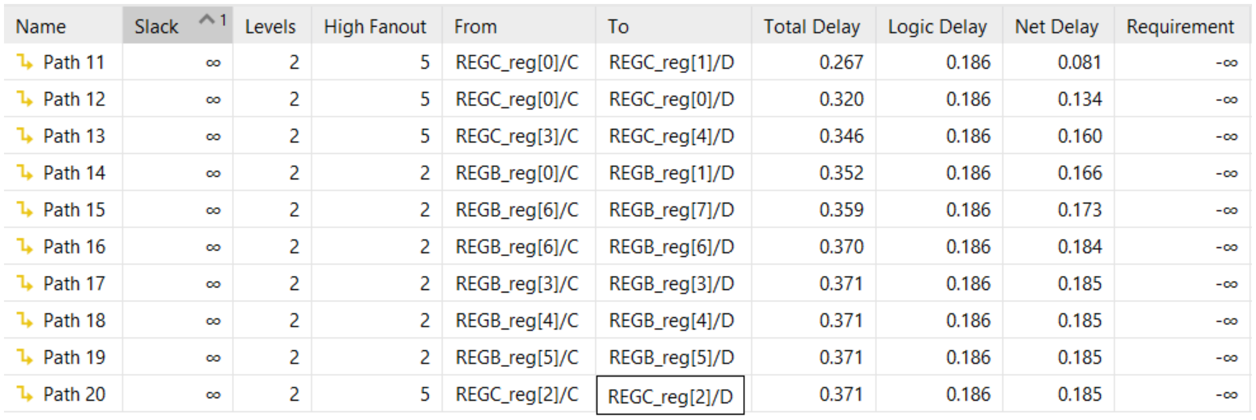
Timing report for setup and hold can be seen in Figure 11 and Figure 12.



- General Information
- Timer Settings
- Design Timing Summary
- Check Timing (193)
- Intra-Clock Paths
- Inter-Clock Paths
- Other Path Groups
- User Ignored Paths
- Unconstrained Paths
  - NONE to NONE
    - Setup (10)
    - Hold (10)

Name	Slack	Levels	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement	Source Clock
Path 1	∞	6	8	Control_Signal[5]	Status_Signal[1]	11.394	4.744	6.650	∞	input port clock
Path 2	∞	6	8	Control_Signal[5]	Status_Signal[2]	10.512	4.114	6.398	∞	input port clock
Path 3	∞	7	8	Control_Signal[2]	REGC_reg[7]/D	7.519	2.546	4.973	∞	input port clock
Path 4	∞	7	8	Control_Signal[2]	REGC_reg[6]/D	7.418	2.468	4.950	∞	input port clock
Path 5	∞	6	8	Control_Signal[2]	REGC_reg[3]/D	7.373	2.323	5.050	∞	input port clock
Path 6	∞	7	8	Control_Signal[2]	REGC_reg[5]/D	7.090	2.765	4.325	∞	input port clock
Path 7	∞	6	8	Control_Signal[2]	REGC_reg[2]/D	7.027	2.259	4.768	∞	input port clock
Path 8	∞	6	8	Control_Signal[2]	REGC_reg[1]/D	6.807	1.907	4.900	∞	input port clock
Path 9	∞	7	8	Control_Signal[2]	REGC_reg[4]/D	6.785	2.448	4.338	∞	input port clock
Path 10	∞	6	8	Control_Signal[2]	Counter_reg[5]/D	6.612	2.440	4.172	∞	input port clock

Figure 11

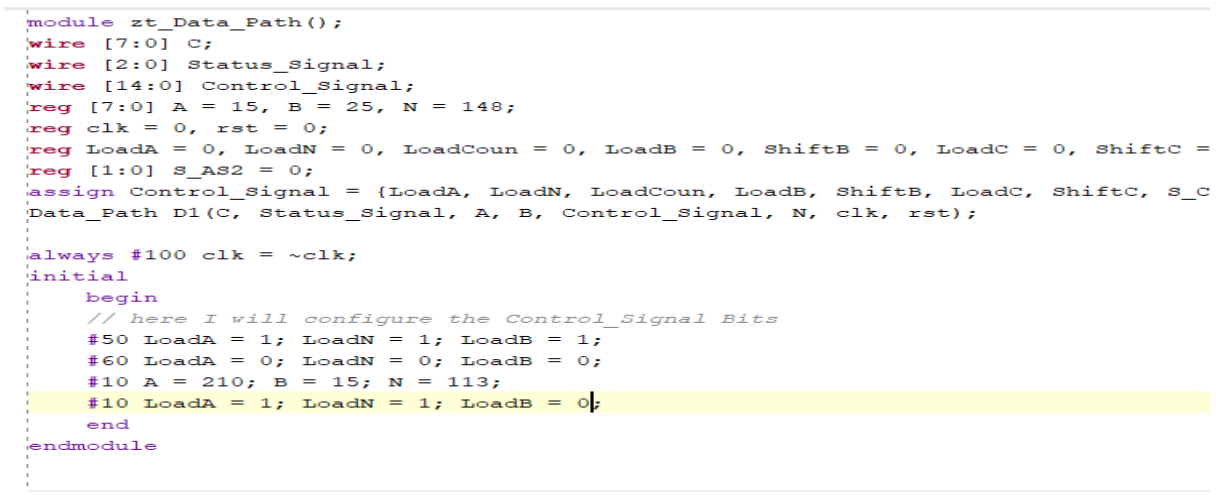


Name	Slack ^1	Levels	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement
Path 11	∞	2	5	REGC_reg[0]/C	REGC_reg[1]/D	0.267	0.186	0.081	-∞
Path 12	∞	2	5	REGC_reg[0]/C	REGC_reg[0]/D	0.320	0.186	0.134	-∞
Path 13	∞	2	5	REGC_reg[3]/C	REGC_reg[4]/D	0.346	0.186	0.160	-∞
Path 14	∞	2	2	REGB_reg[0]/C	REGB_reg[1]/D	0.352	0.186	0.166	-∞
Path 15	∞	2	2	REGB_reg[6]/C	REGB_reg[7]/D	0.359	0.186	0.173	-∞
Path 16	∞	2	2	REGB_reg[6]/C	REGB_reg[6]/D	0.370	0.186	0.184	-∞
Path 17	∞	2	2	REGB_reg[3]/C	REGB_reg[3]/D	0.371	0.186	0.185	-∞
Path 18	∞	2	2	REGB_reg[4]/C	REGB_reg[4]/D	0.371	0.186	0.185	-∞
Path 19	∞	2	2	REGB_reg[5]/C	REGB_reg[5]/D	0.371	0.186	0.185	-∞
Path 20	∞	2	5	REGC_reg[2]/C	REGC_reg[2]/D	0.371	0.186	0.185	-∞

Figure 12

2.c

Now let's test the circuit for different values of A, B and N and I will also test the circuit for different Control_Signal values just to see if the device is working well. I will use post-implementation timing simulation for this circuit. The first thing that I want to do is to see how the circuit loads REGA, REGB and REGN registers. I will load those registers just by configuring the appropriate Control_Signal bits. The initial block in Figure 13 will be used to set the Control_Signal bits so I can test it.



```
module zt_Data_Path();
wire [7:0] C;
wire [2:0] Status_Signal;
wire [14:0] Control_Signal;
reg [7:0] A = 15, B = 25, N = 148;
reg clk = 0, rst = 0;
reg LoadA = 0, LoadN = 0, LoadCoun = 0, LoadB = 0, ShiftB = 0, LoadC = 0, ShiftC =
reg [1:0] S_AS2 = 0;
assign Control_Signal = {LoadA, LoadN, LoadCoun, LoadB, ShiftB, LoadC, ShiftC, S_C
Data_Path D1(C, Status_Signal, A, B, Control_Signal, N, clk, rst);

always #100 clk = ~clk;
initial
    begin
    // here I will configure the Control_Signal Bits
    #50 LoadA = 1; LoadN = 1; LoadB = 1;
    #60 LoadA = 0; LoadN = 0; LoadB = 0;
    #10 A = 210; B = 15; N = 113;
    #10 LoadA = 1; LoadN = 1; LoadB = 0;
    end
endmodule
```

Figure 13

For the test bench shown in Figure 13, I expect the circuit to load different values on the REGA, REGB and REGN registers in the posedge of the clock only if the respective Load bits in the Control_Signal are high. Please note that for the testbench in the Figure 13, REGB should only update once because the second posedge of the clock, the LoadB signal is low and it shouldn't update the value. Please refer to Figure 14.

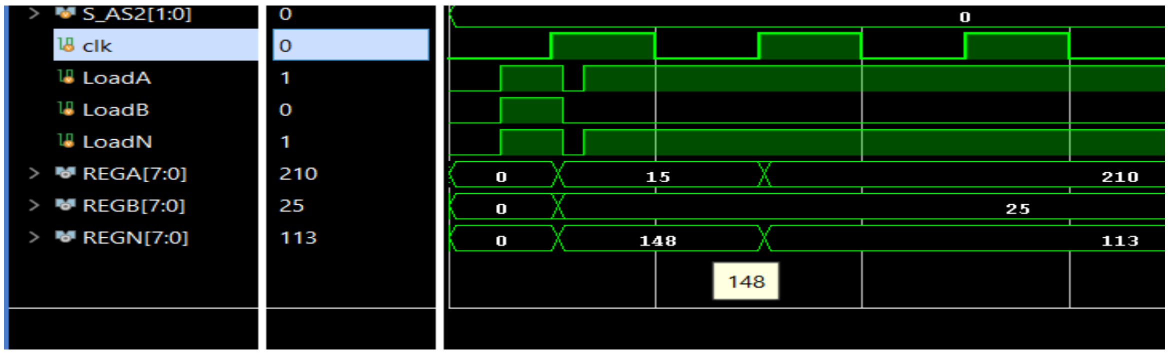
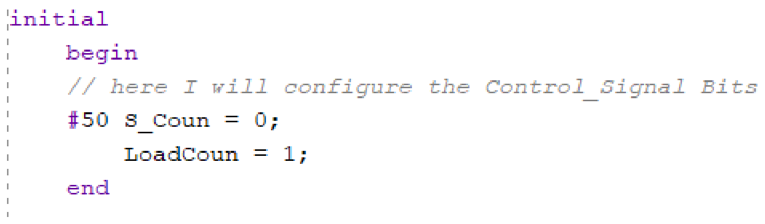


Figure 14

As it seems, the Data_Path is behaving in the appropriate manner for this input signal.

So the loading of REGA REGB and REGB registers is working fine. Now I will load the counter. The counter value is given by a multiplexer whose bit select is S_Coun. I will give the S_Coun the value 1 because I want to load the k parameter at the counter and also I will activate the counter input. I changed the initial block in the circuit because at this moment I will just be testing the Counter register. Figure 15 shows the initial block of the Figure 13.



```
initial
    begin
    // here I will configure the Control_Signal Bits
    #50 S_Coun = 0;
        LoadCoun = 1;
    end
```

Figure 15

Figure 16 shows that Counter register was loaded successfully. So I can say that this part of the circuit works well also.

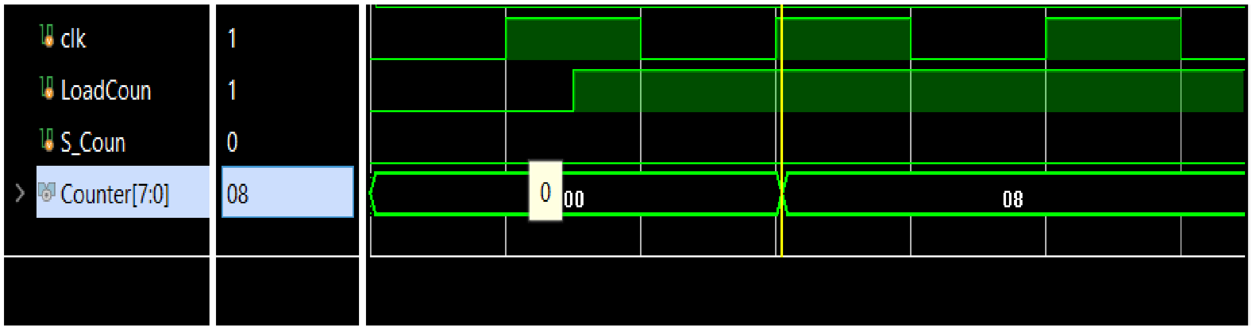
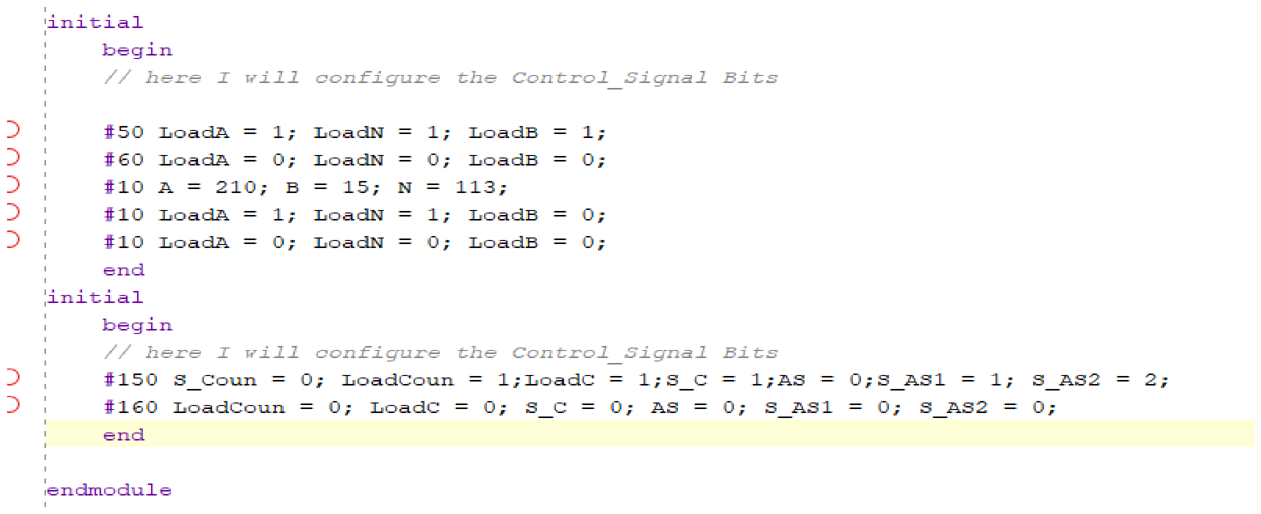


Figure 16

As it seems, the S_Coun = 0, which means the multiplexer will show the value of k at the input of the counter register. The LoadCoun becomes high at 150ns, which means that during the first positive edge of clk, the Counter register will not be loaded as shown in Figure 16. So, up to this point the Data_Path internal signals have been correct. Now I will test the Multiplexers. The inputs to the Multiplexers are ResultAS, Counter, REGC, REGN and REGA. I will load all these registers by using the initial blocks shown in Figure 17.



```
initial
    begin
    // here I will configure the Control_Signal Bits

    #50 LoadA = 1; LoadN = 1; LoadB = 1;
    #60 LoadA = 0; LoadN = 0; LoadB = 0;
    #10 A = 210; B = 15; N = 113;
    #10 LoadA = 1; LoadN = 1; LoadB = 0;
    #10 LoadA = 0; LoadN = 0; LoadB = 0;
    end
initial
    begin
    // here I will configure the Control_Signal Bits
    #150 S_Coun = 0; LoadCoun = 1;LoadC = 1;S_C = 1;AS = 0;S_AS1 = 1; S_AS2 = 2;
    #160 LoadCoun = 0; LoadC = 0; S_C = 0; AS = 0; S_AS1 = 0; S_AS2 = 0;
    end

endmodule
```

Figure 17

Note that the loaded values in Figure 17 will be used through the next couple of tests. I will not be loading the registers with other values but I will be changing some of the Control_Signal Bits in order to see the functionality of the Data_Path circuit. The bits that will be changed will be changed in separate initial blocks and will occur when time = 510ns.

The first multiplexer has 1-bit select and two inputs. One of the inputs is k(parameter) and the other is ResultAS. I will change the value of S_Count so the output value should change accordingly. Please refer to Figure 18.

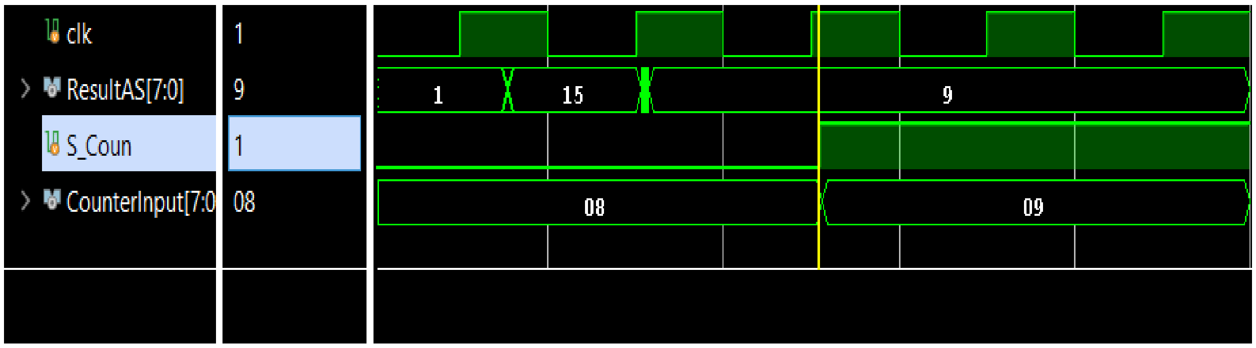


Figure 18

I changed the value of S_Coun at time 510 ns and you see that there was an immediate change on the output CounterInput. Until that moment, the multiplexer was outputting the value of k which is 8. After the select bit became 1, the output showed 9 because that is the value of the second input of the multiplexer (ResultAS).

Now I will test the second multiplexer whose output is Operand1C. The inputs to the MUX are Counter and Register C. the bit select is S_Comp1. This bit decides which of the inputs will go to the compare block. Without further explanation, the result is shown in Figure 19. It seems the MUX is working in the appropriate way.

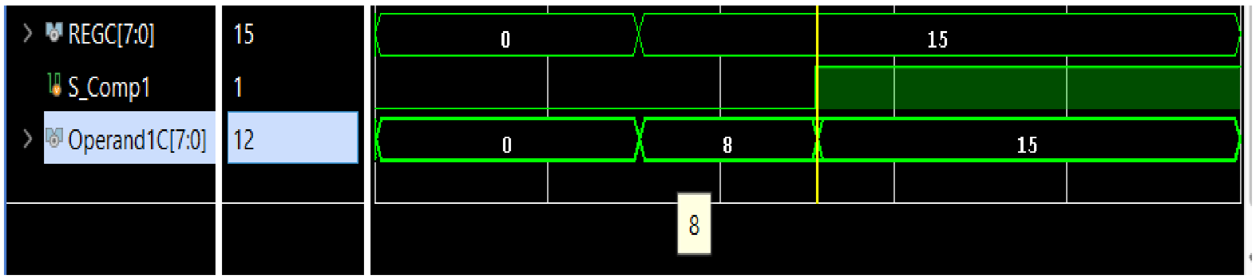


Figure 19

I will skip the third and fourth multiplexers in the second column in Figure 2 in odev3 file since they are the same blocks with the two MUX tested above.

I will now test the 4x1 MUX in Figure 2 in odev3 file. I changed the value of S_AS2. Please see at Figure 20 how the output Operand2C is changing in accordance with the select bits. In general it is showing the correct output.

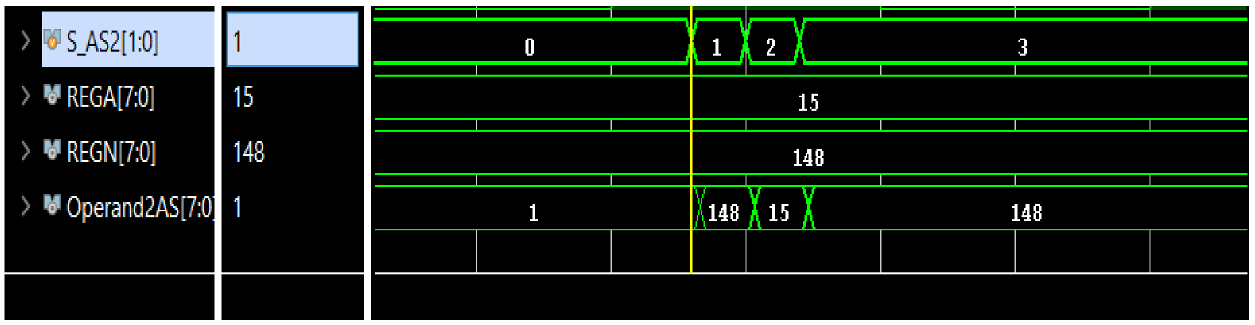


Figure 20

I will now test the ADD-SUB circuit. I derived the functionality of this circuit by looking at the ASM table and the Control_Signal bits in the truth table. When AS signal is low, the circuit should add the two inputs and when AS is high, the circuit should substract the two inputs. Please see the waveforms at Figure 21. Operand1AS and Operand2AS are the two inputs. The AS is the bit which decides which of the operations will be done and ResulatAS is the output.

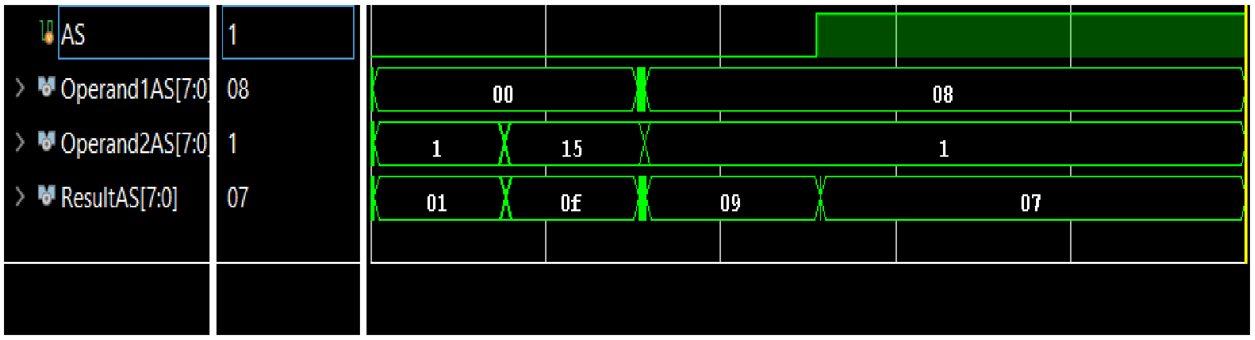


Figure 21

So the circuit is computing the correct value respective to the AS signal value. The Operand1AS is 8 because the S_AS1 and S_AS2 signals are both 0 and Counter and 1 have gone respectively to Operand1AS and Operand2AS.

Now I will test the Compare circuit. Operand1C and Operand2C will be the input to the circuit. I will try to use S_Comp1 and S_Comp2 to set different values in the compare circuit. The outputs are the 2 MSB-s of Status_Signal. If {S_Comp1, S_Comp2} = 0, the Counter register is being compared with 0. Counter in Figure 22 is 8, thus greater than 0 and the 2-MSB of status Signal are 01 as defined in the Comparator in Data_Path code. {S_Comp1, S_Comp2} = 1, the comparator is comparing Counter with REGN. REGN is 148 thus greater than counter and 00 signal is outputted from the Comparator. If {S_Comp1, S_Comp2} = 2 REGC is being compared with 0 and since REGC = 15 thus > than 0 the circuit will output 01 in the 2 MSB of Status_Signal. If {S_Comp1, S_Comp2} = 3, comparator is comparing REGC and REGN. REGN > REGC and thus the signal that should be outputted should be 00. All the predictions made above hold true even for this circuit.

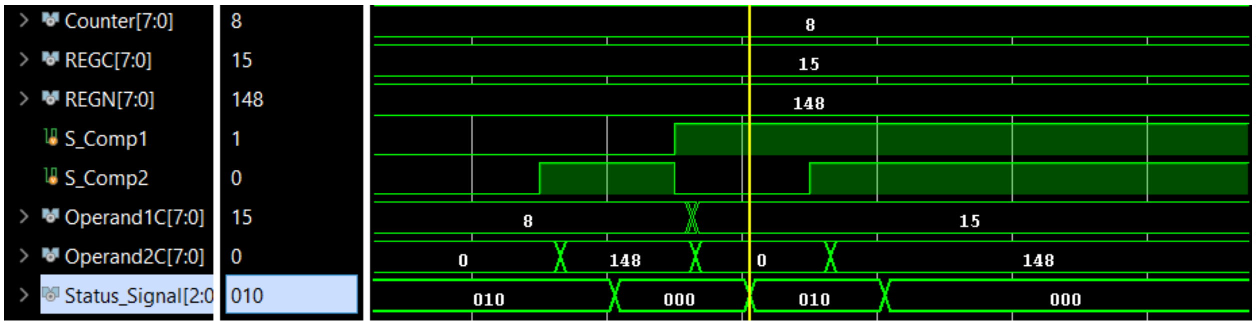


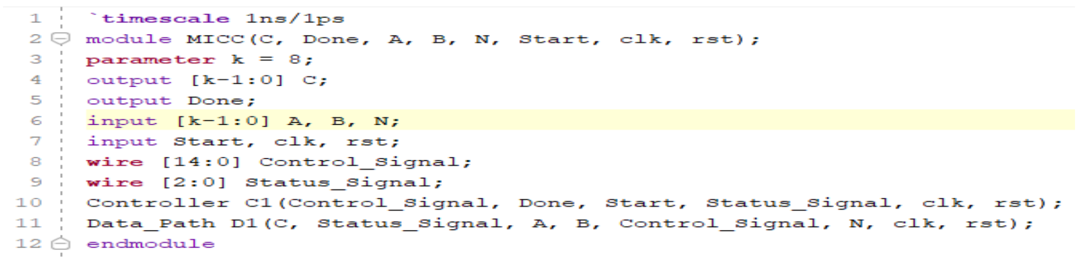
Figure 22

With this I am finishing the testing of the Data_Path. Please note that I didn't try 16 different values for A, B and N since I thought this way of testing shows the functionality of the device better. All the simulations up to know have been post implementation timing simulations.

MICC (Microcontroller)

3.a

The code that I wrote for joining the 2 hardware together is shown in Figure 23.



```
`timescale 1ns/1ps
module MICC(C, Done, A, B, N, Start, clk, rst);
parameter k = 8;
output [k-1:0] C;
output Done;
input [k-1:0] A, B, N;
input Start, clk, rst;
wire [14:0] Control_Signal;
wire [2:0] Status_Signal;
Controller C1(Control_Signal, Done, Start, Status_Signal, clk, rst);
Data_Path D1(C, Status_Signal, A, B, Control_Signal, N, clk, rst);
endmodule
```

Figure 23

3.b The RTL circuit is shown in Figure 24.

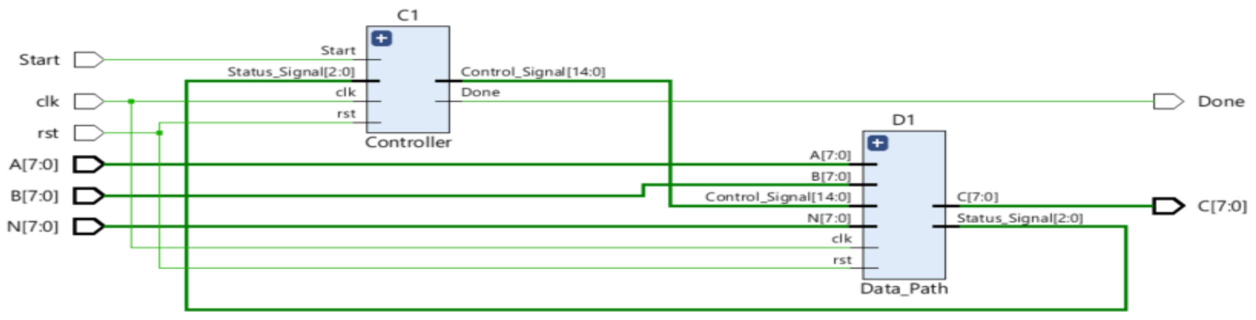


Figure 24

The implemented circuit is shown in Figure 25.

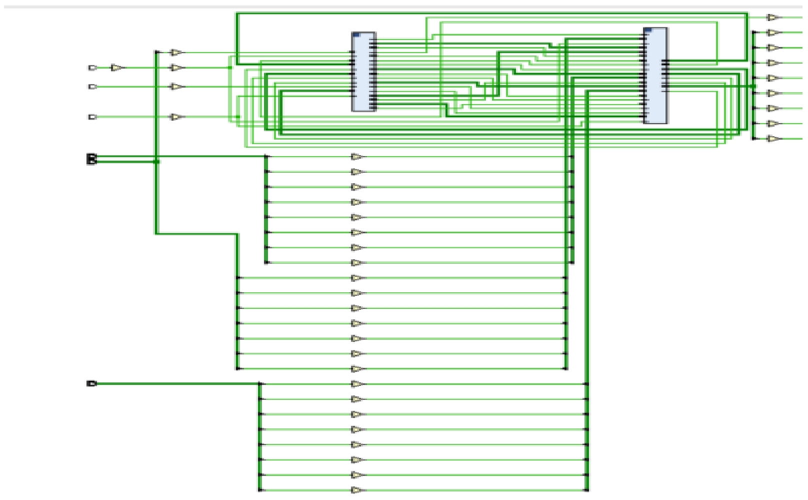
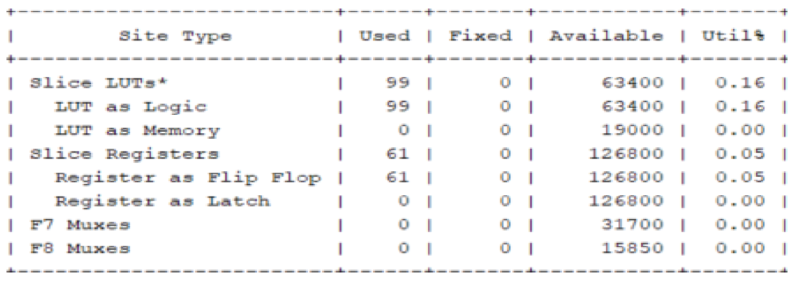


Figure 25

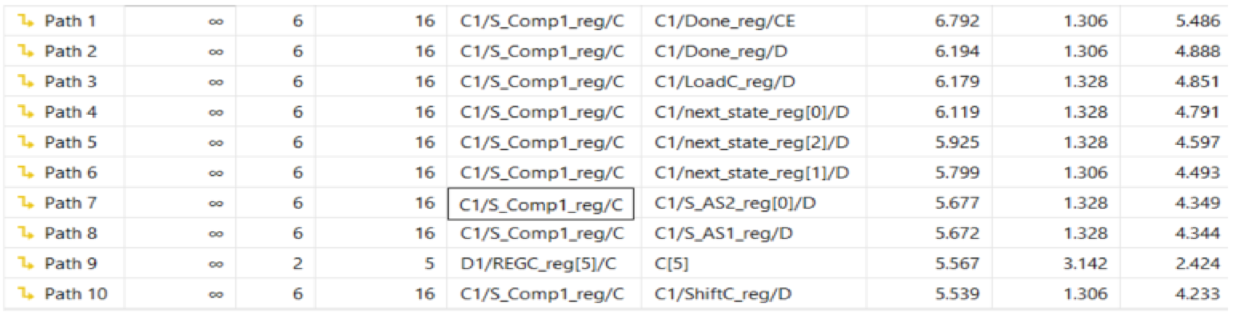
The area usage is shown in Figure 26.



```
+-------------------------+------+-------+-----------+-------+
|        Site Type        | Used | Fixed | Available | Util% |
+-------------------------+------+-------+-----------+-------+
| Slice LUTs*             |   99 |     0 |     63400 |  0.16 |
|   LUT as Logic          |   99 |     0 |     63400 |  0.16 |
|   LUT as Memory         |    0 |     0 |     19000 |  0.00 |
| Slice Registers         |   61 |     0 |    126800 |  0.05 |
|   Register as Flip Flop |   61 |     0 |    126800 |  0.05 |
|   Register as Latch     |    0 |     0 |    126800 |  0.00 |
| F7 Muxes                |    0 |     0 |     31700 |  0.00 |
| F8 Muxes                |    0 |     0 |     15850 |  0.00 |
+-------------------------+------+-------+-----------+-------+
```

Figure 26

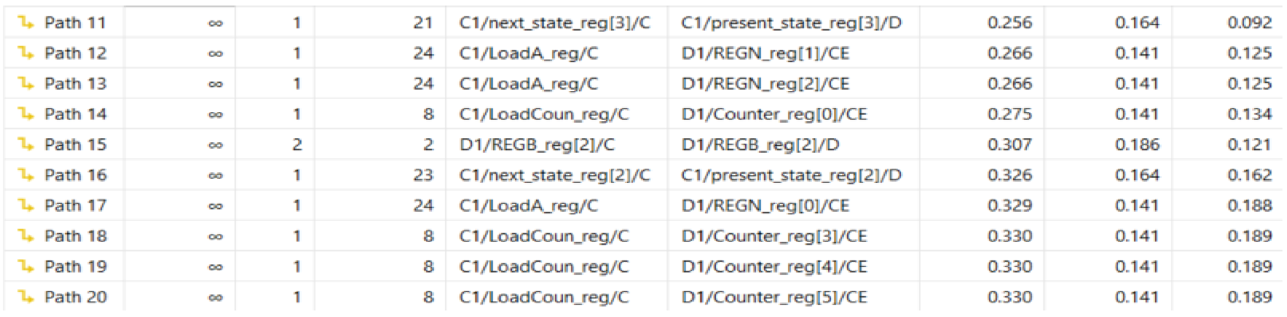
The Setup delay is shown in Figure 27.



Path 1	∞	6	16	C1/S_Comp1_reg/C	C1/Done_reg/CE	6.792	1.306	5.486
Path 2	∞	6	16	C1/S_Comp1_reg/C	C1/Done_reg/D	6.194	1.306	4.888
Path 3	∞	6	16	C1/S_Comp1_reg/C	C1/LoadC_reg/D	6.179	1.328	4.851
Path 4	∞	6	16	C1/S_Comp1_reg/C	C1/next_state_reg[0]/D	6.119	1.328	4.791
Path 5	∞	6	16	C1/S_Comp1_reg/C	C1/next_state_reg[2]/D	5.925	1.328	4.597
Path 6	∞	6	16	C1/S_Comp1_reg/C	C1/next_state_reg[1]/D	5.799	1.306	4.493
Path 7	∞	6	16	C1/S_Comp1_reg/C	C1/S_AS2_reg[0]/D	5.677	1.328	4.349
Path 8	∞	6	16	C1/S_Comp1_reg/C	C1/S_AS1_reg/D	5.672	1.328	4.344
Path 9	∞	2	5	D1/REGC_reg[5]/C	C[5]	5.567	3.142	2.424
Path 10	∞	6	16	C1/S_Comp1_reg/C	C1/ShiftC_reg/D	5.539	1.306	4.233

Figure 27

The Hold delay is shown in Figure 28.



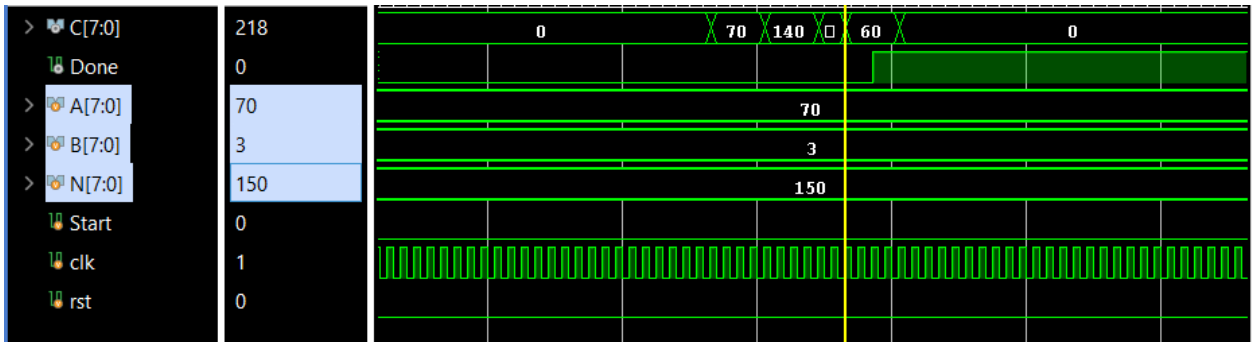
Path 11	∞	1	21	C1/next_state_reg[3]/C	C1/present_state_reg[3]/D	0.256	0.164	0.092
Path 12	∞	1	24	C1/LoadA_reg/C	D1/REGN_reg[1]/CE	0.266	0.141	0.125
Path 13	∞	1	24	C1/LoadA_reg/C	D1/REGN_reg[2]/CE	0.266	0.141	0.125
Path 14	∞	1	8	C1/LoadCoun_reg/C	D1/Counter_reg[0]/CE	0.275	0.141	0.134
Path 15	∞	2	2	D1/REGB_reg[2]/C	D1/REGB_reg[2]/D	0.307	0.186	0.121
Path 16	∞	1	23	C1/next_state_reg[2]/C	C1/present_state_reg[2]/D	0.326	0.164	0.162
Path 17	∞	1	24	C1/LoadA_reg/C	D1/REGN_reg[0]/CE	0.329	0.141	0.188
Path 18	∞	1	8	C1/LoadCoun_reg/C	D1/Counter_reg[3]/CE	0.330	0.141	0.189
Path 19	∞	1	8	C1/LoadCoun_reg/C	D1/Counter_reg[4]/CE	0.330	0.141	0.189
Path 20	∞	1	8	C1/LoadCoun_reg/C	D1/Counter_reg[5]/CE	0.330	0.141	0.189

Figure 28

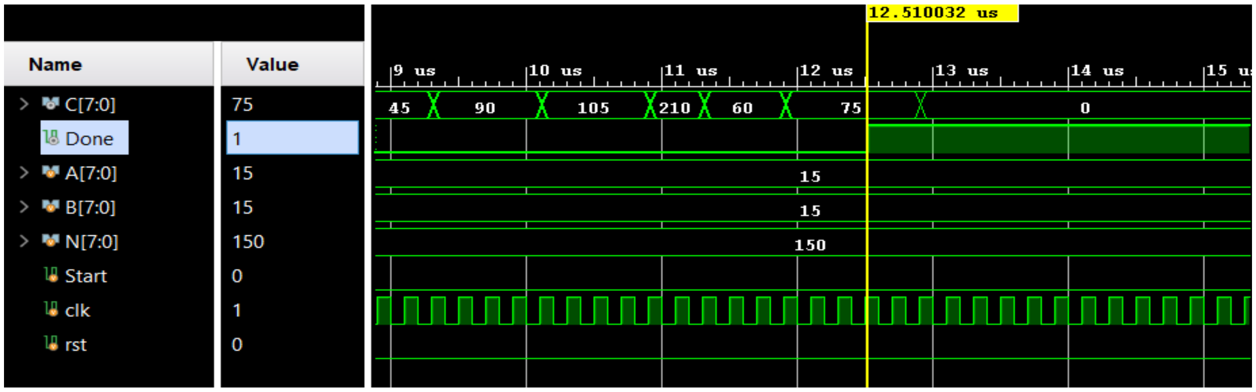
3.c

Now I will test the circuit with 16 different values. The correct value will be observed when Done signal becomes 1.

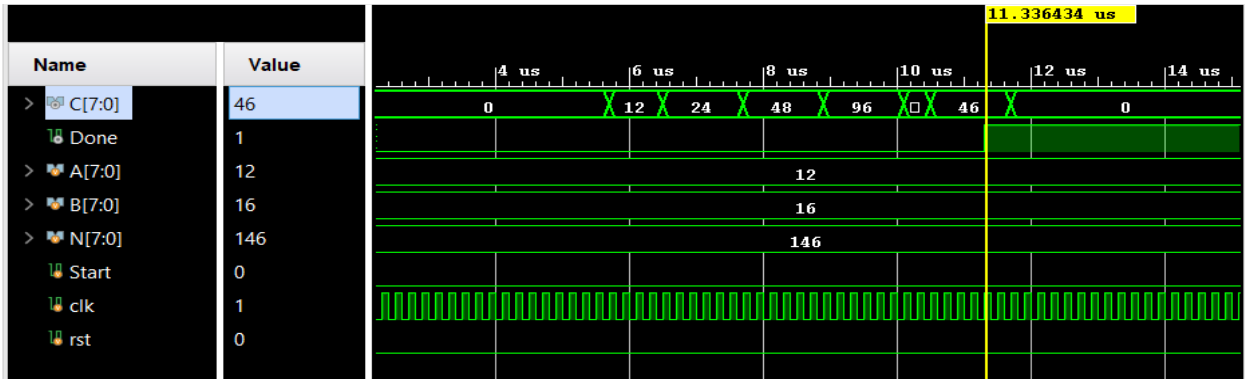
A = 70, B = 3, N = 150 => Expected result: 60, Calculated result: 60



A = 15 , B = 15 , N = 150 => Expected result: 75 . Calculated result: 75

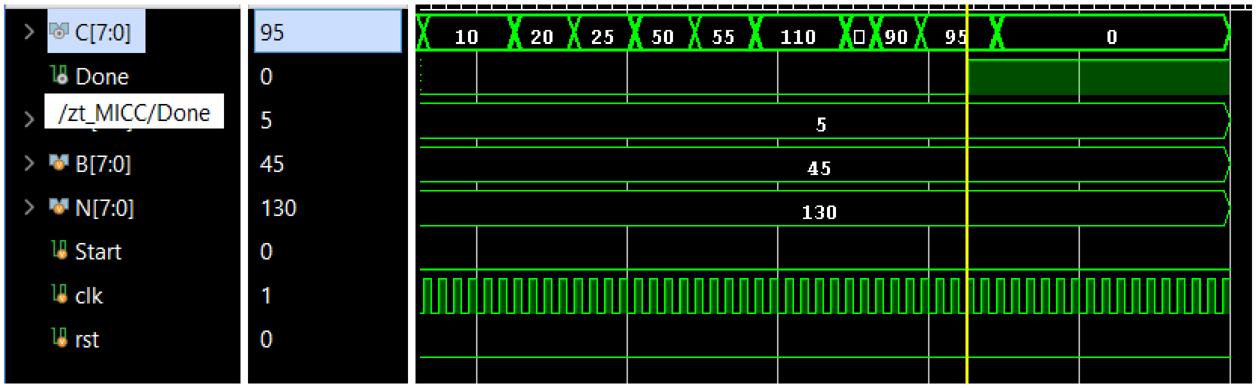


A = 12, B = 16, N = 146 => Expected result: 46 . Calculated result: 46



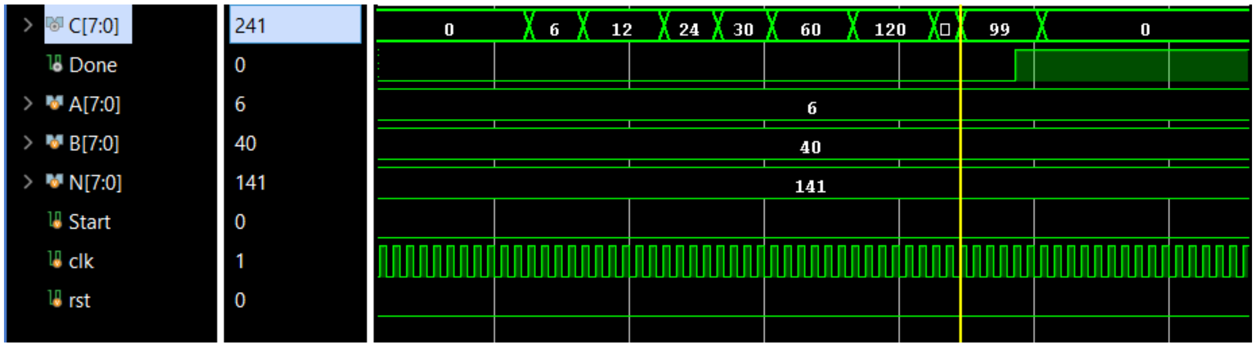
Name	Value
C[7:0]	46
Done	1
A[7:0]	12
B[7:0]	16
N[7:0]	146
Start	0
clk	1
rst	0

A = 5, B = 45 , N = 130 => Expected result: 95 . Calculated result: 95



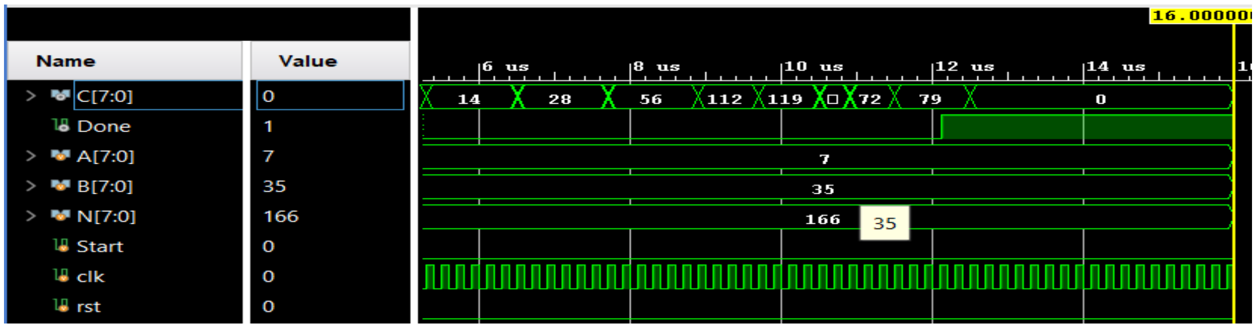
Name	Value
C[7:0]	95
Done	0
/zt_MICC/Done	5
B[7:0]	45
N[7:0]	130
Start	0
clk	1
rst	0

A = 6, B = 40, N = 141 => Expected result: 99 . Calculated result: 99



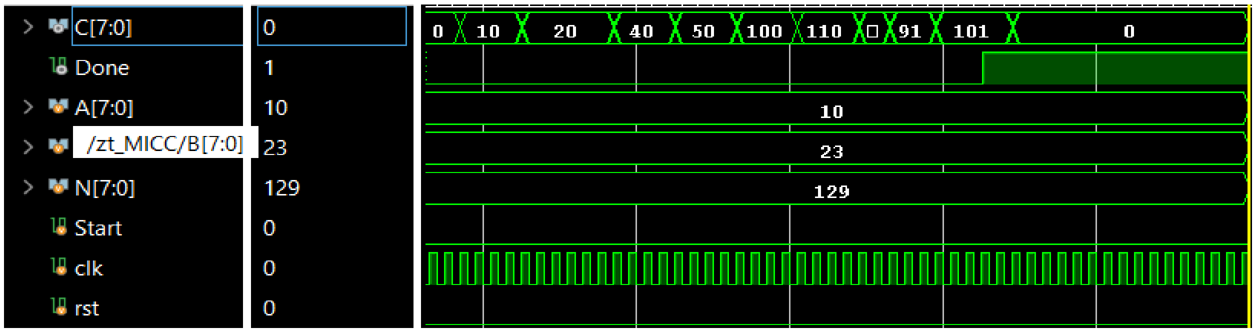
Name	Value
C[7:0]	241
Done	0
A[7:0]	6
B[7:0]	40
N[7:0]	141
Start	0
clk	1
rst	0

A = 7, B = 35, N = 166 => Expected result: 79. Calculated result: 79



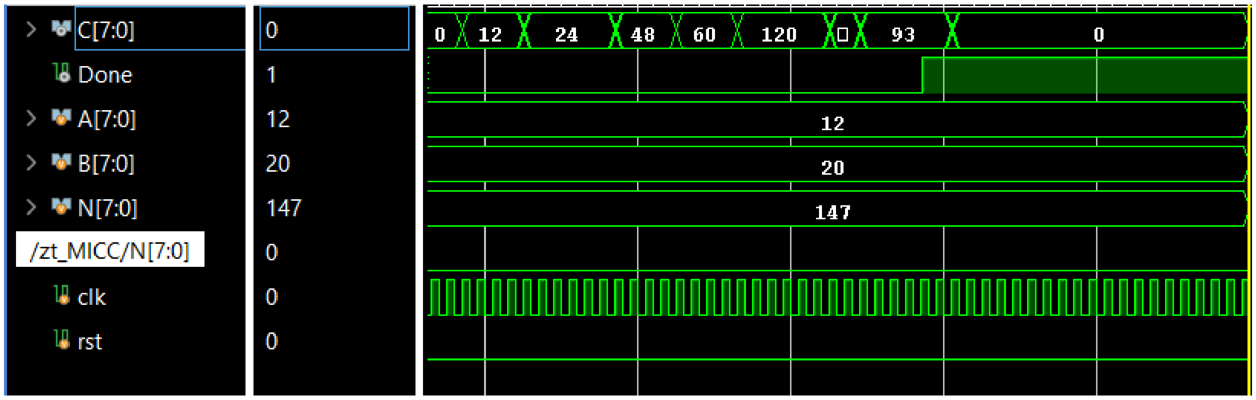
```
Name      Value
C[7:0]    0
Done      1
A[7:0]    7
B[7:0]    35
N[7:0]    166
Start     0
clk       0
rst       0
```

A =10, B = 23, N = 129 => Expected result: 101. Calculated result: 101



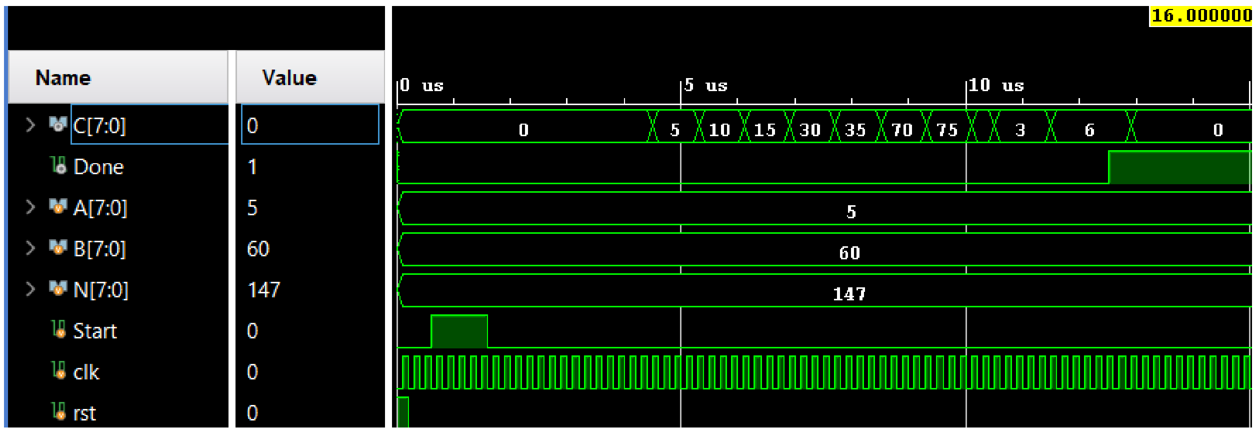
```
C[7:0]            0
Done              1
A[7:0]            10
/zt_MICC/B[7:0]   23
N[7:0]            129
Start             0
clk               0
rst               0
```

A = 12, B = 20, N = 147 => Expected result: 93 . Calculated result: 93



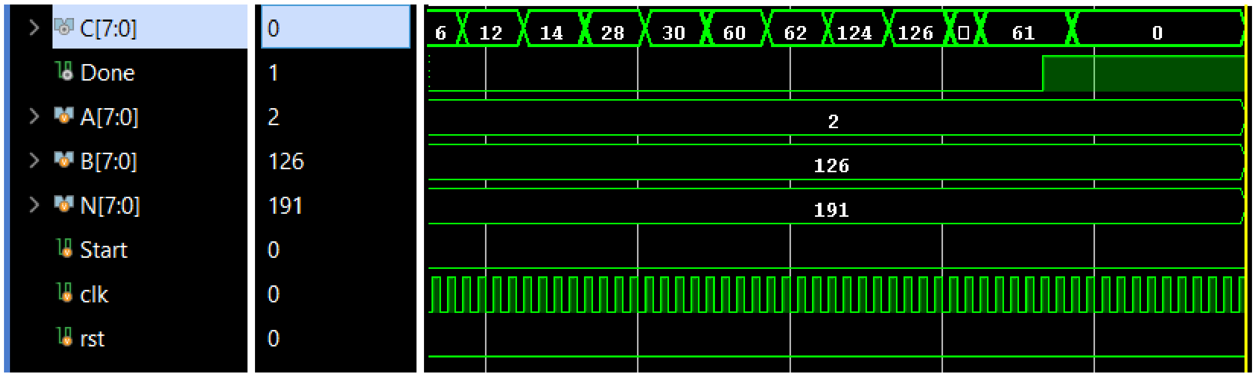
```
C[7:0]            0
Done              1
A[7:0]            12
B[7:0]            20
N[7:0]            147
/zt_MICC/N[7:0]   0
clk               0
rst               0
```

A = 5, B = 60, N = 147 => Expected result: 6. Calculated result: 6



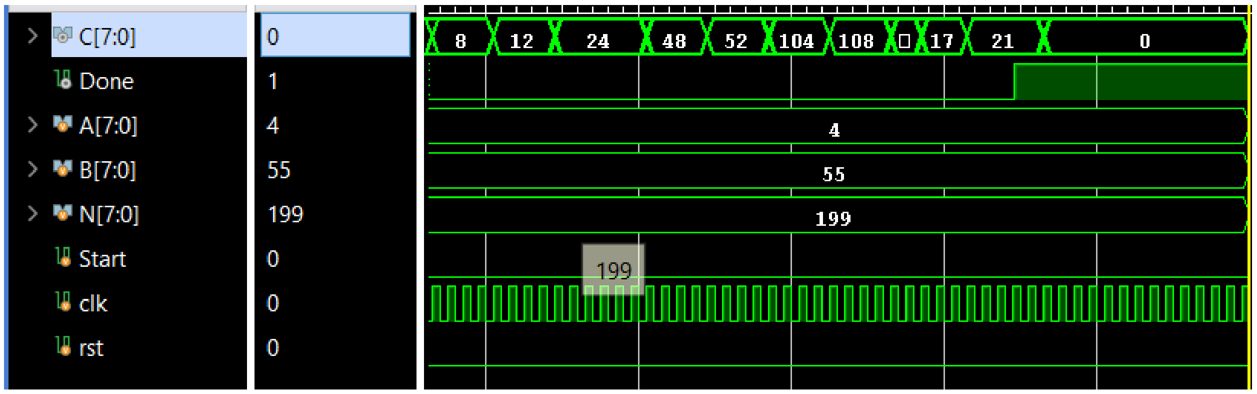
Name	Value
C[7:0]	0
Done	1
A[7:0]	5
B[7:0]	60
N[7:0]	147
Start	0
clk	0
rst	0

A = 2, B = 126, N = 191 => Expected result: 61. Calculated result: 61



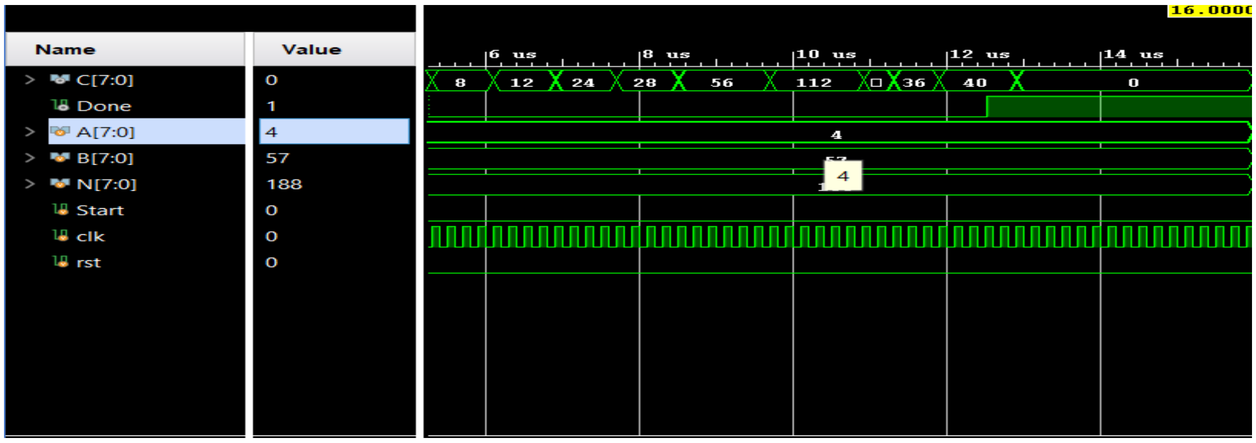
Name	Value
C[7:0]	0
Done	1
A[7:0]	2
B[7:0]	126
N[7:0]	191
Start	0
clk	0
rst	0

A = 4, B = 55, N = 199 => Expected result: 21. Calculated result: 21



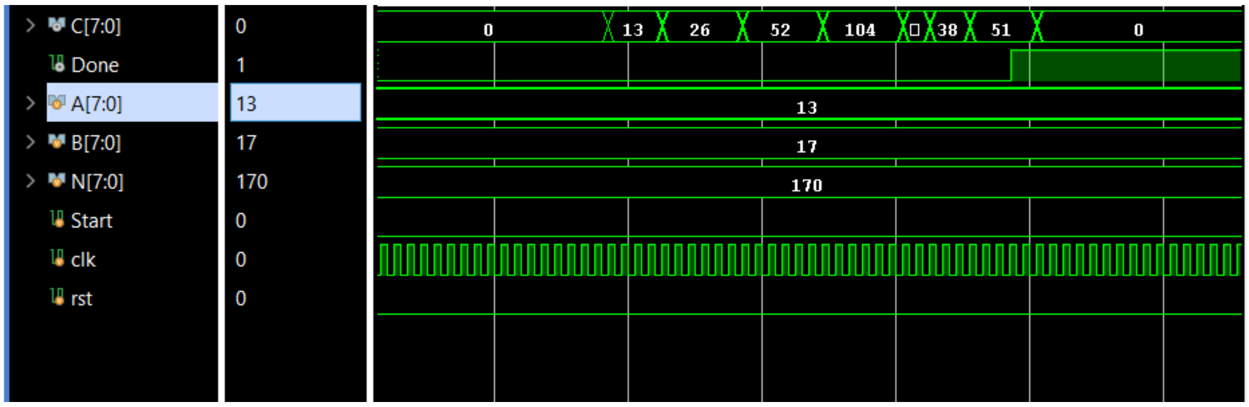
Name	Value
C[7:0]	0
Done	1
A[7:0]	4
B[7:0]	55
N[7:0]	199
Start	0
clk	0
rst	0

A = 4, B = 57, N = 188 => Expected result: 40. Calculated result: 40



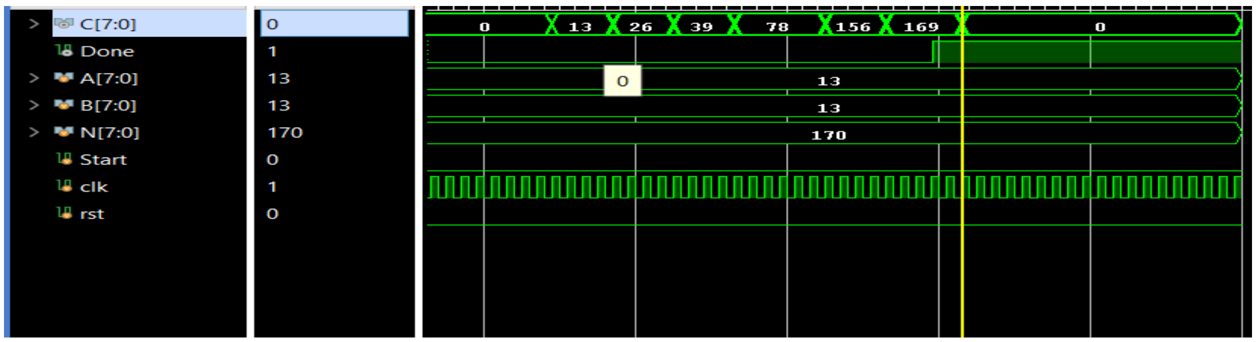
Name	Value
C[7:0]	0
Done	1
A[7:0]	4
B[7:0]	57
N[7:0]	188
Start	0
clk	0
rst	0

A = 13, B = 17, N = 170 => Expected result: 51. Calculated result: 51



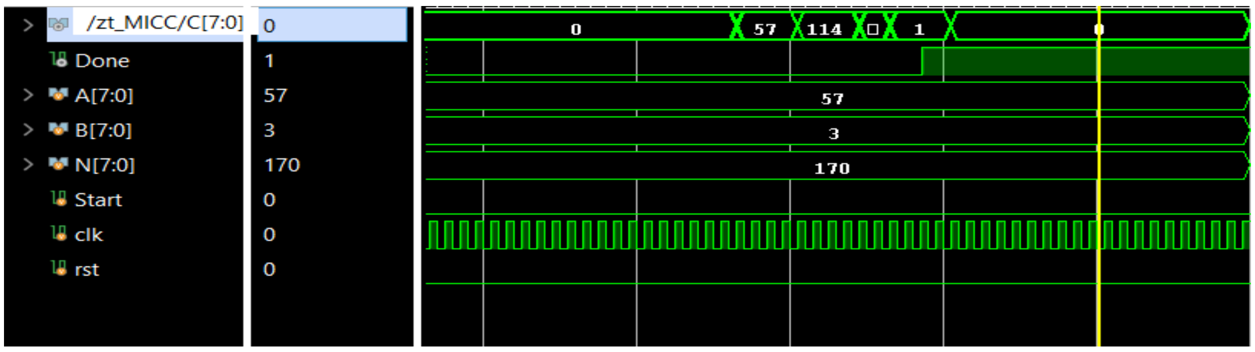
Name	Value
C[7:0]	0
Done	1
A[7:0]	13
B[7:0]	17
N[7:0]	170
Start	0
clk	0
rst	0

A = 13, B = 13, N = 170 => Expected result: 169. Calculated result: 169

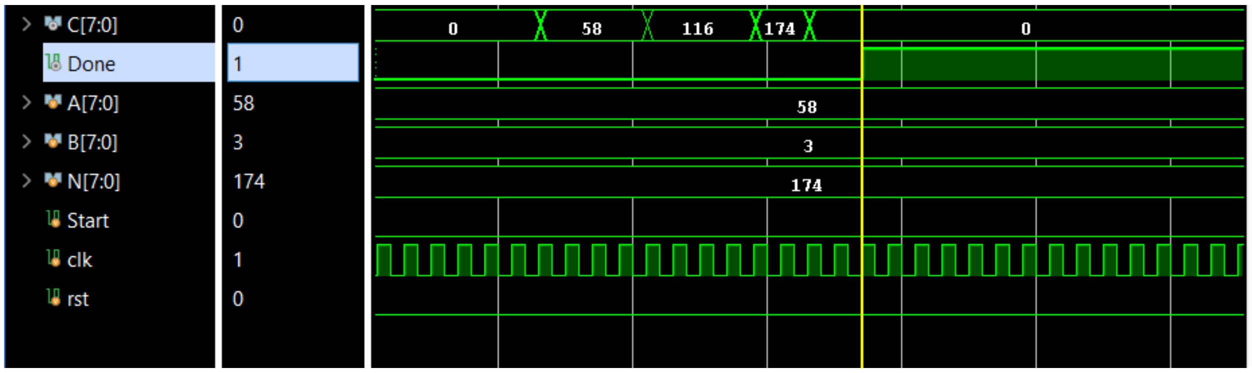


Name	Value
C[7:0]	0
Done	1
A[7:0]	13
B[7:0]	13
N[7:0]	170
Start	0
clk	1
rst	0

A = 57, B = 3, N = 170 => Expected result: 1. Calculated result: 1



A = 58, B = 3, N = 174 => Expected result: 0. Calculated result: 0


C[7:0]
Done
A[7:0]
B[7:0]
N[7:0]
Start
clk
rst
0
1
58
3
174
0
1
0
58
116
174