EE446 PRELIMINARY WORK #2

Laboratory Work 2 - Arithmetic Logic Processor Design

Introduction:

The goal of the initial laboratory activity is to create a Verilog library comprised of the essential modules that will be utilized throughout the computer design process. Furthermore, simple datapath design will be explored by creating simple architectures using the modules in the built library to do certain simple tasks.

This laboratory activity will familiarize me with developing modules with Verilog HDL and constructing architectures with schematic design. This laboratory activity is intended to familiarize me with the software—Quartus and Modelsim—that will be used throughout the semester. Finally, the designs will be practiced on a development board, DE0-Nano, which is equipped with a field programmable gate array and numerous peripheral units such as switch inputs, general purpose I/O pins, and LED outputs.

1.2.1 Datapath Design:

1.

I designed a datapath so that the 2's complement of the number stored in the R0 or R1 register is taken and stored back to the corresponding register with a proper control signal.

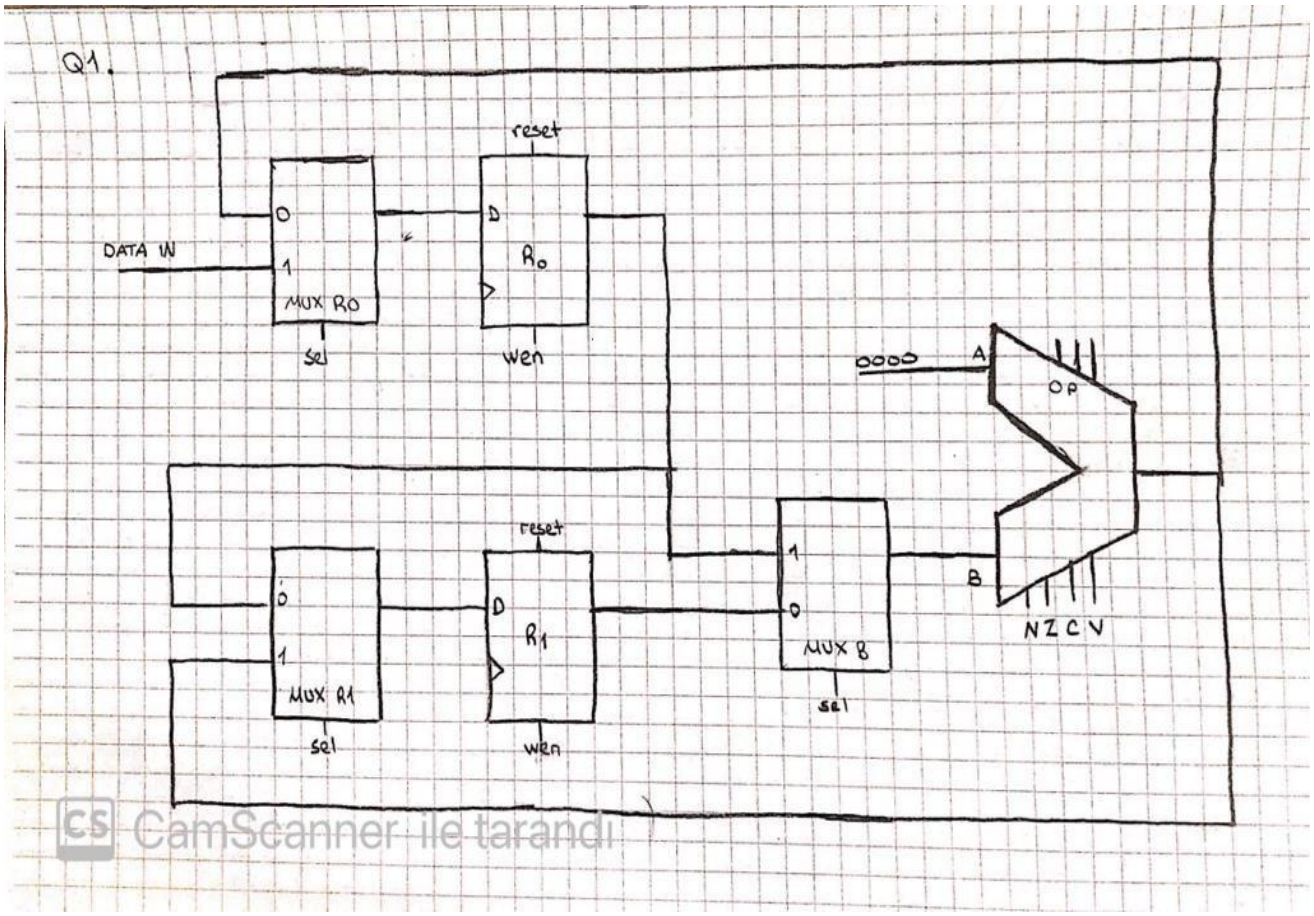


Figure 1. Implementation of the data path for question 1.

the 2’s complement of the number stored in the R0

R0 wen -> 1

MUX R0 sel -> 0

MUX B sel -> 1

the 2’s complement of the number stored in the R01

R1 wen -> 1

MUX R1 sel -> 1

MUX B sel -> 0

In both cases, I need to do subtraction operation.

OP -> 001

2.

I extend my data path to support the ALP operations except for multiplication and division.

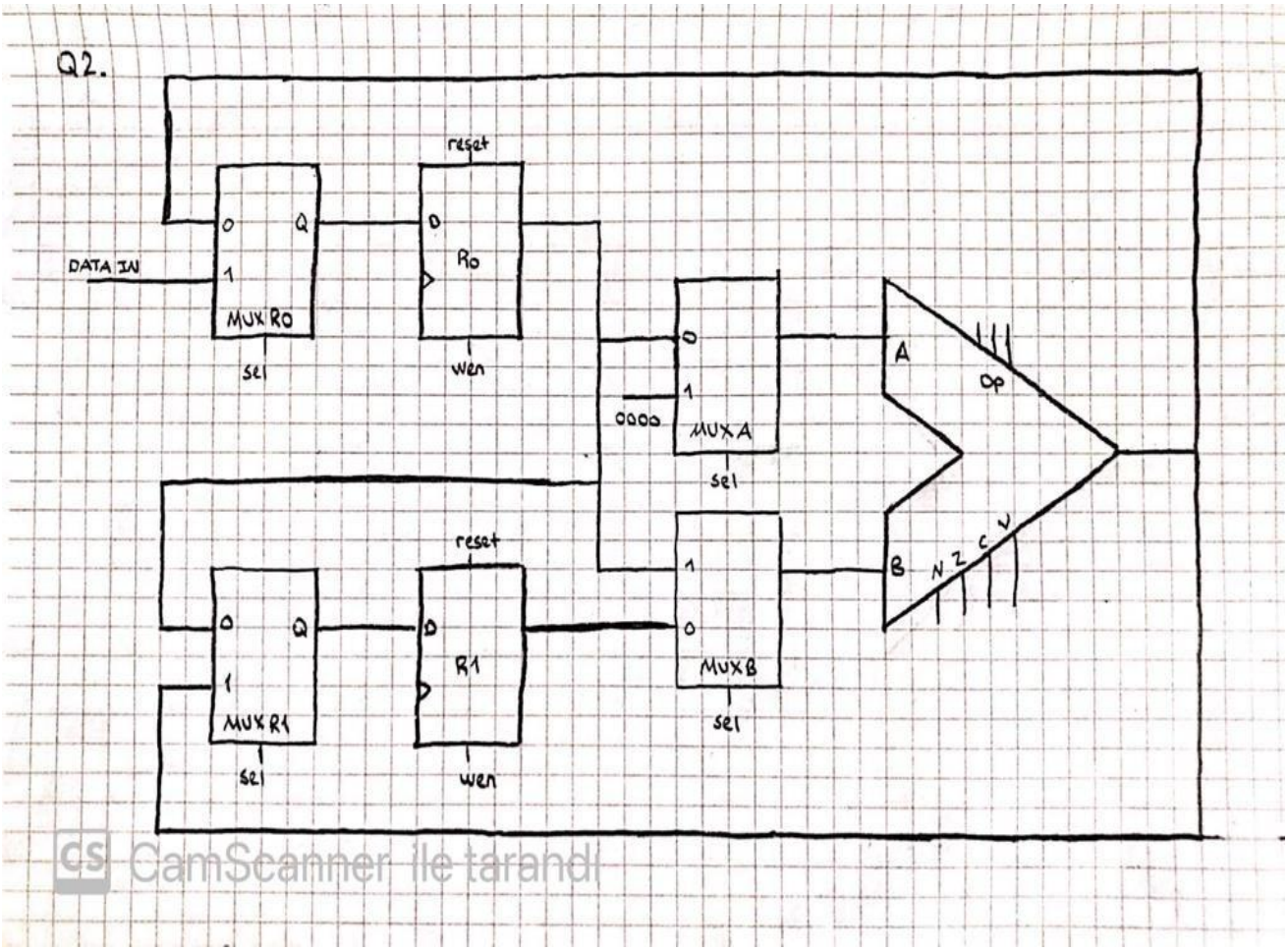


Figure 2. Implementation of the data path for question 2.

The only difference from the previous part is MUX A. It is added to the input A of the ALU.

3.

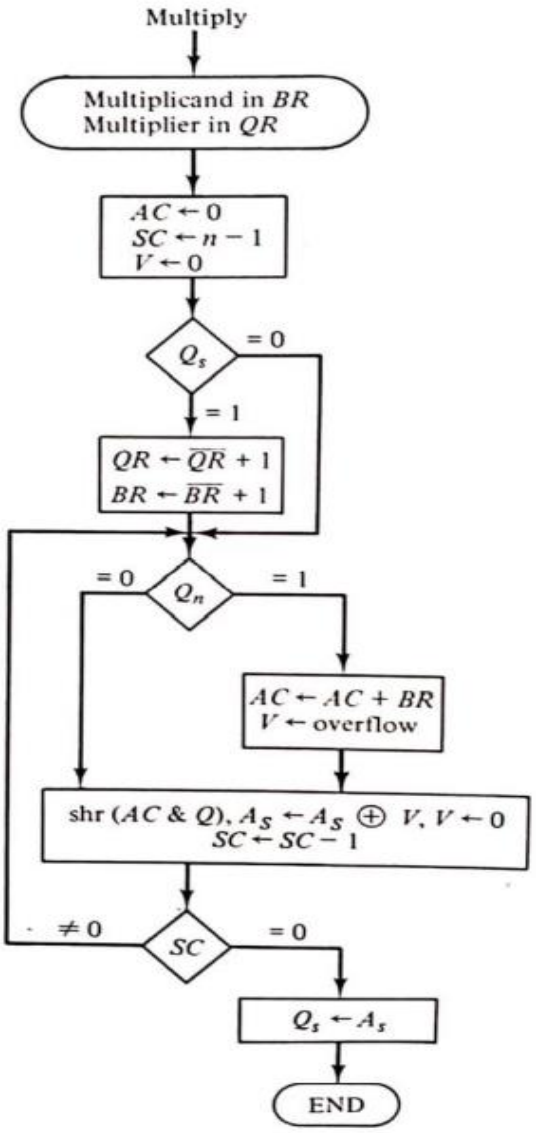


Figure 3. Signed multiplication algorithm

I extend my datapath to support signed multiplication operation.

The algorithm works with a positive multiplier (positive/negative multiplicand).

For a negative multiplier we need to take 2's complement of both multiplicand and multiplier.

Qn and Qs are supplied to the controller in order to obtain the right control signals. In the controller logic, SC will be generated.

Complement operations should be performed if Qs = 1.

For the sake of clarity, I've separated these two complement operations into two states.

Because I'm utilizing all of the bits in the Q shift register, if SC = 0, I'm shifting both Ac and Q to the right to obtain the right result.

After that, I introduced another state which allows me to write the results to R0 and R1.

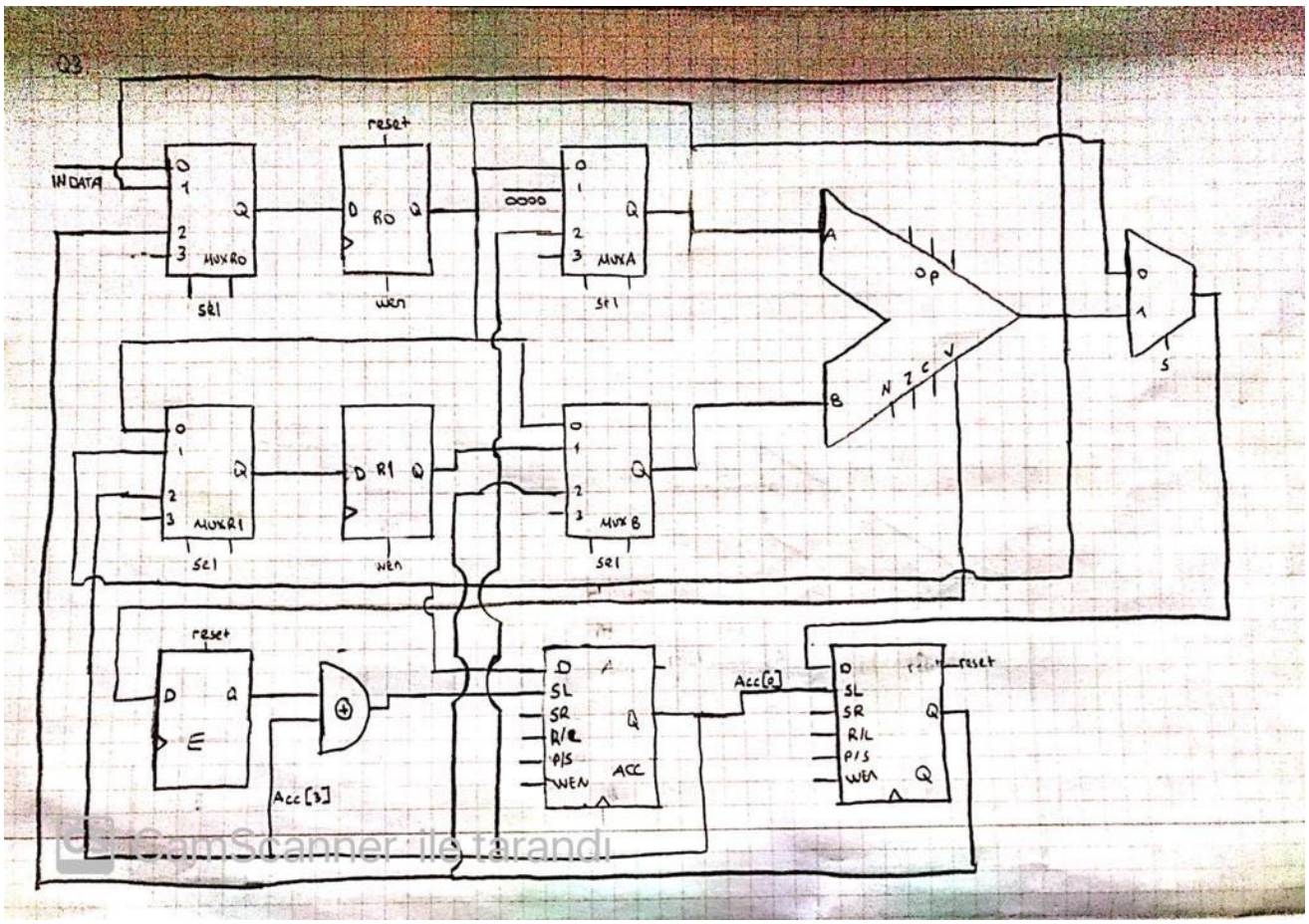


Figure 4. Extended version of my datapath to support signed multiplication operation

4.

5.

I state the control signal inputs of my overall design and the outputs from my design to the controller as the inputs.

Outputs from the datapath to the controller

R0 output	R0_out_in[3:0]
LSB of Q output	Qout_in

Control inputs to the datapath:

R0 reset signal	REGR0_reset_in
R0 write signal	REGR0_write_in
R1 reset signal	REGR1_reset_in
R1 write signal	REGR1_write_in
R0 input 4x1 mux select	MUXR0_sel_in[1:0]
R1 input 4x1 mux select	MUXR1_sel_in[1:0]
ALU A input 4x1 mux select	MUXA_sel_in[1:0]
ALU B input 4x1 mux select	MUXB_sel_in[1:0]
SR Q parallel input 2x1 mux select	MUXSRQ_sel_in
SR Q load signal	SRQ_load_in
SR Q reset signal	SRQ_reset_in
SQ Q shift right signal	SRQ_SR_in
SQ Q write enable signal	SRQ_wen_in
SR Acc load signal	SRAcc_load_in
SR Acc reset signal	SRAcc_reset_in
SQ Acc shift right signal	SRAcc_SR_in
SQ Acc write enable signal	SRAcc_wen_in
ALU OP	ALU_OP_in[2:0]
E reset	E_reset_in

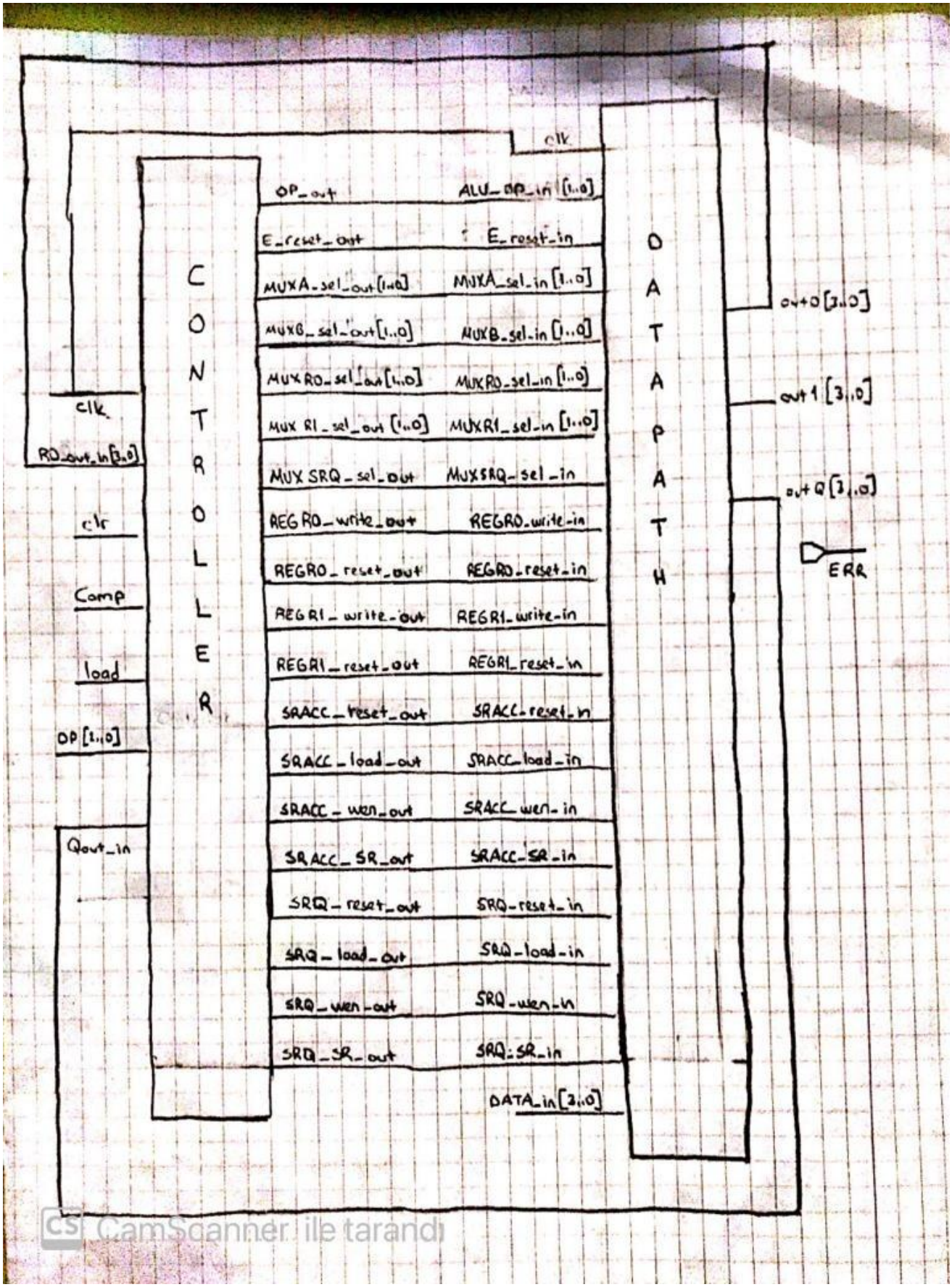


Figure 5. Drawing of black box diagram of my architecture by indicating the inputs and outputs.

6.

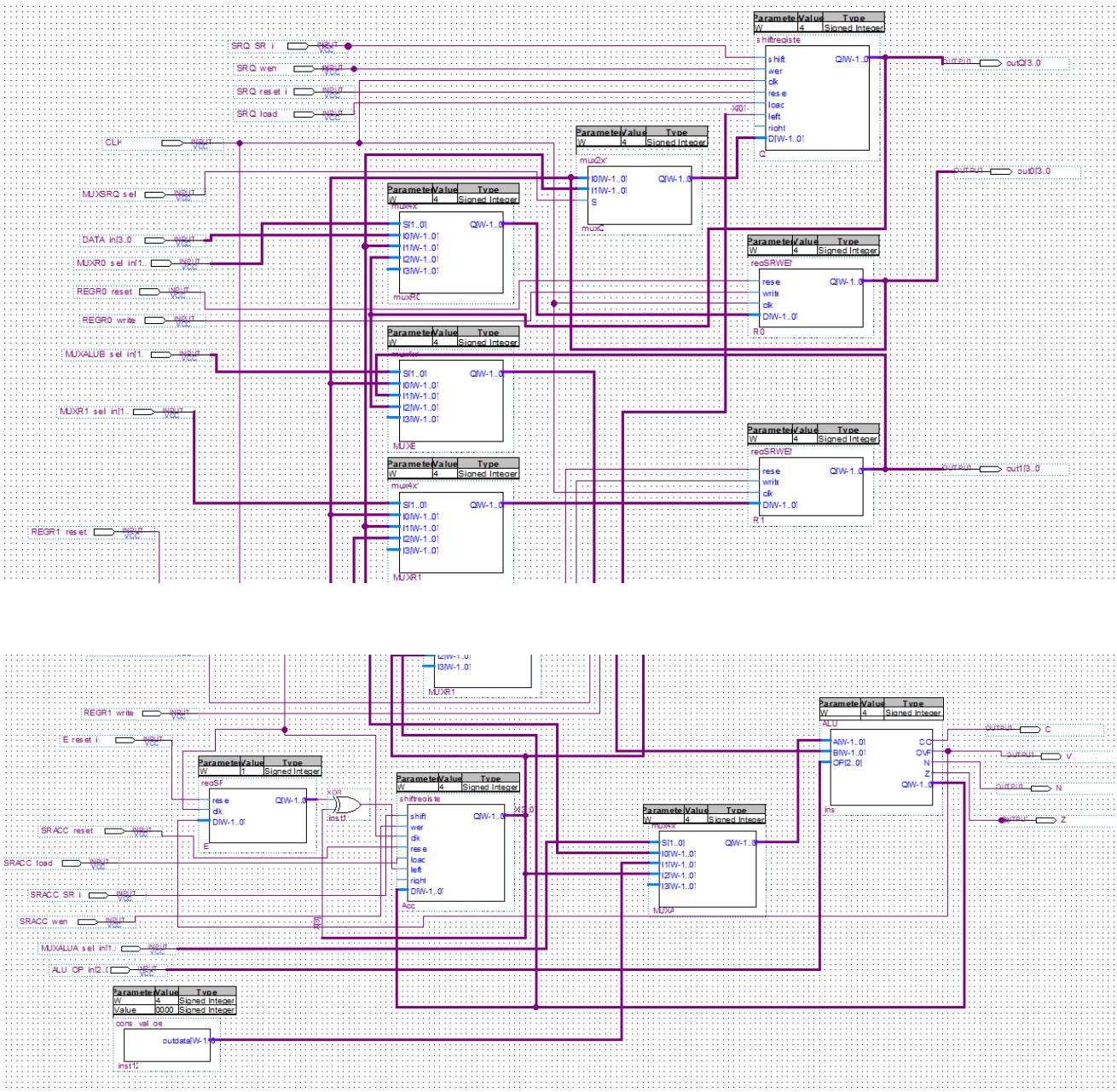


Figure 6. Implementation of my datapath design in Schematic Editor of Quartus

1.2.2 Controller Design:

1.

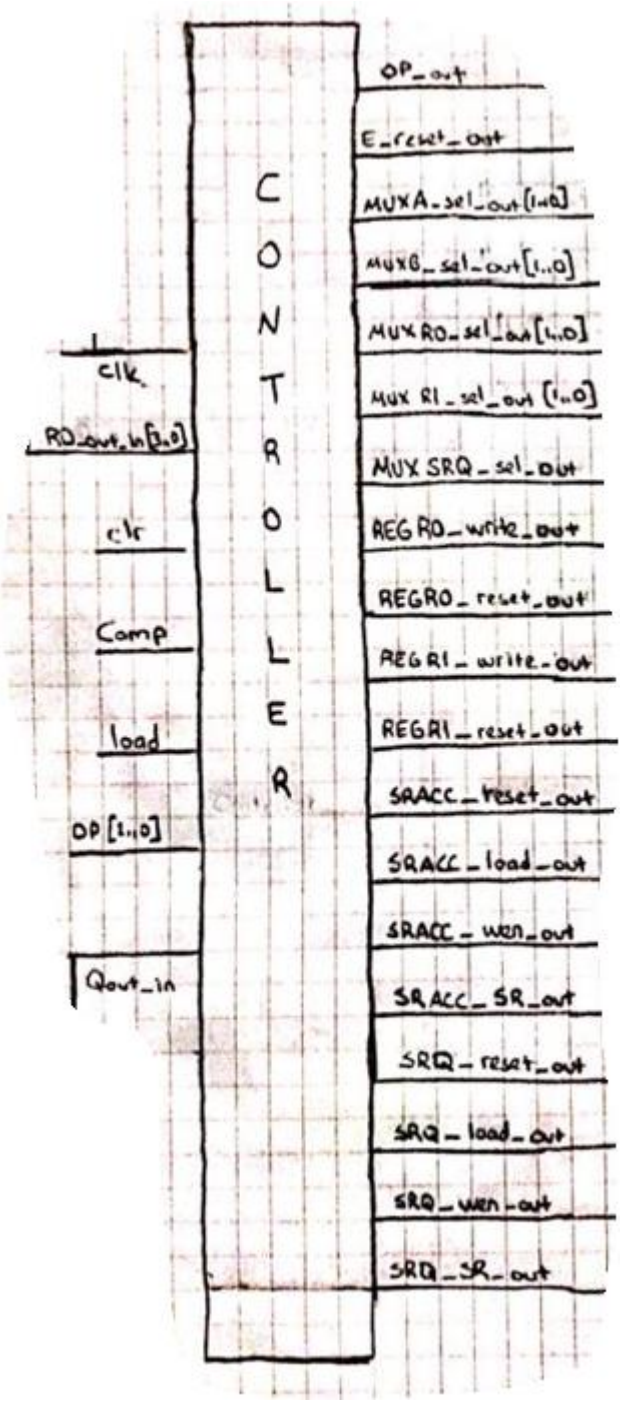
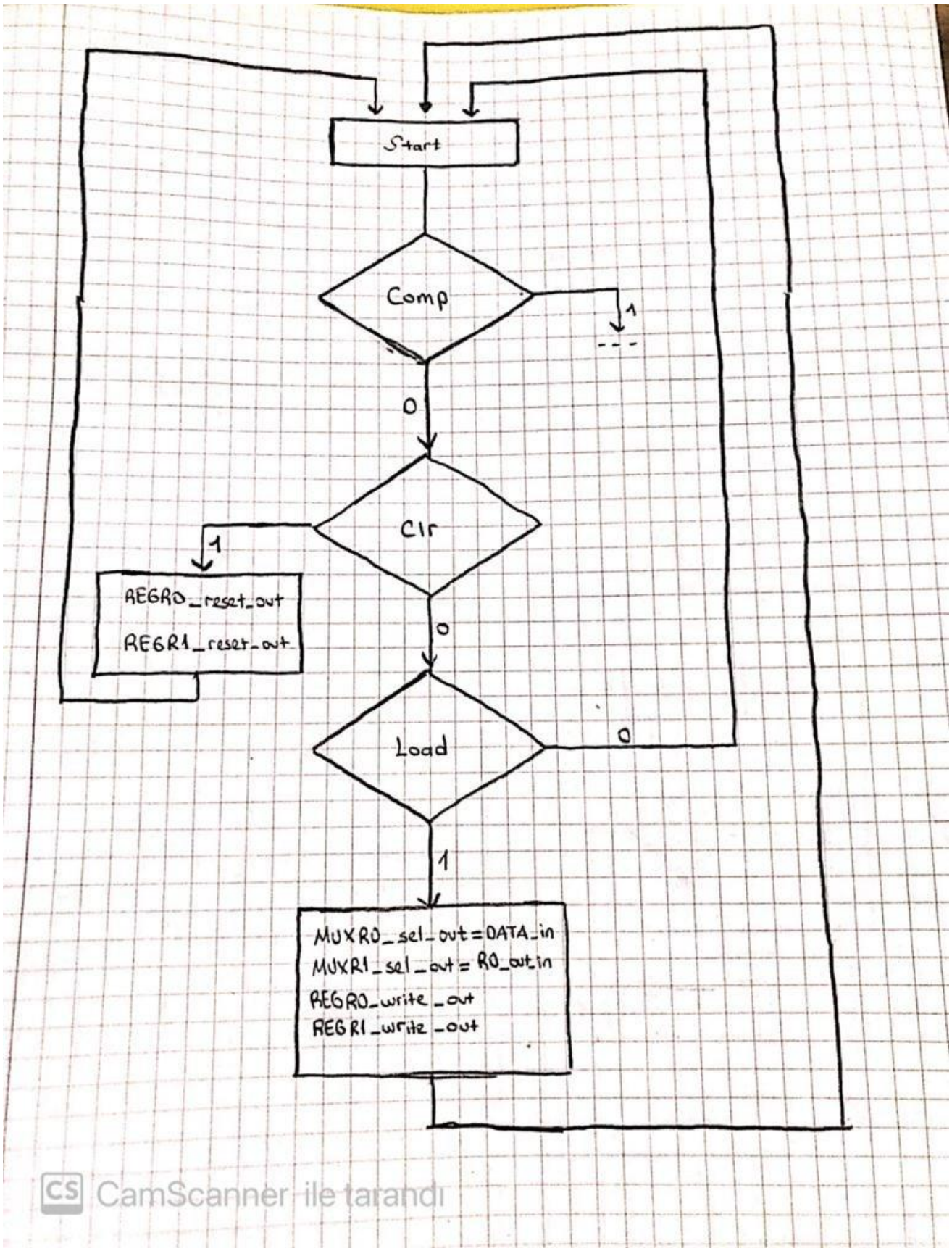
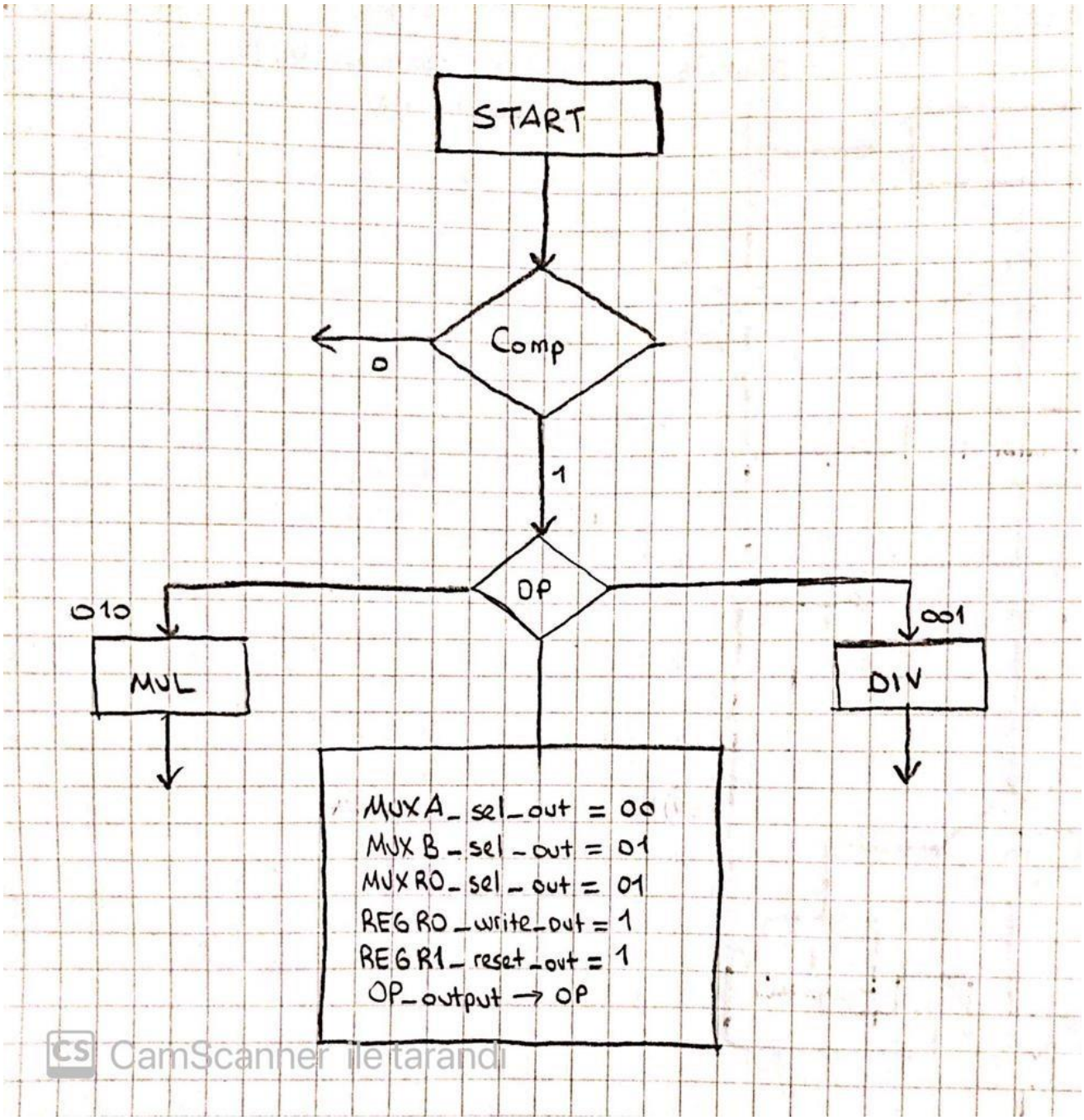


Figure 7. Drawing of the controller unit as a block box diagram and indication of inputs and outputs

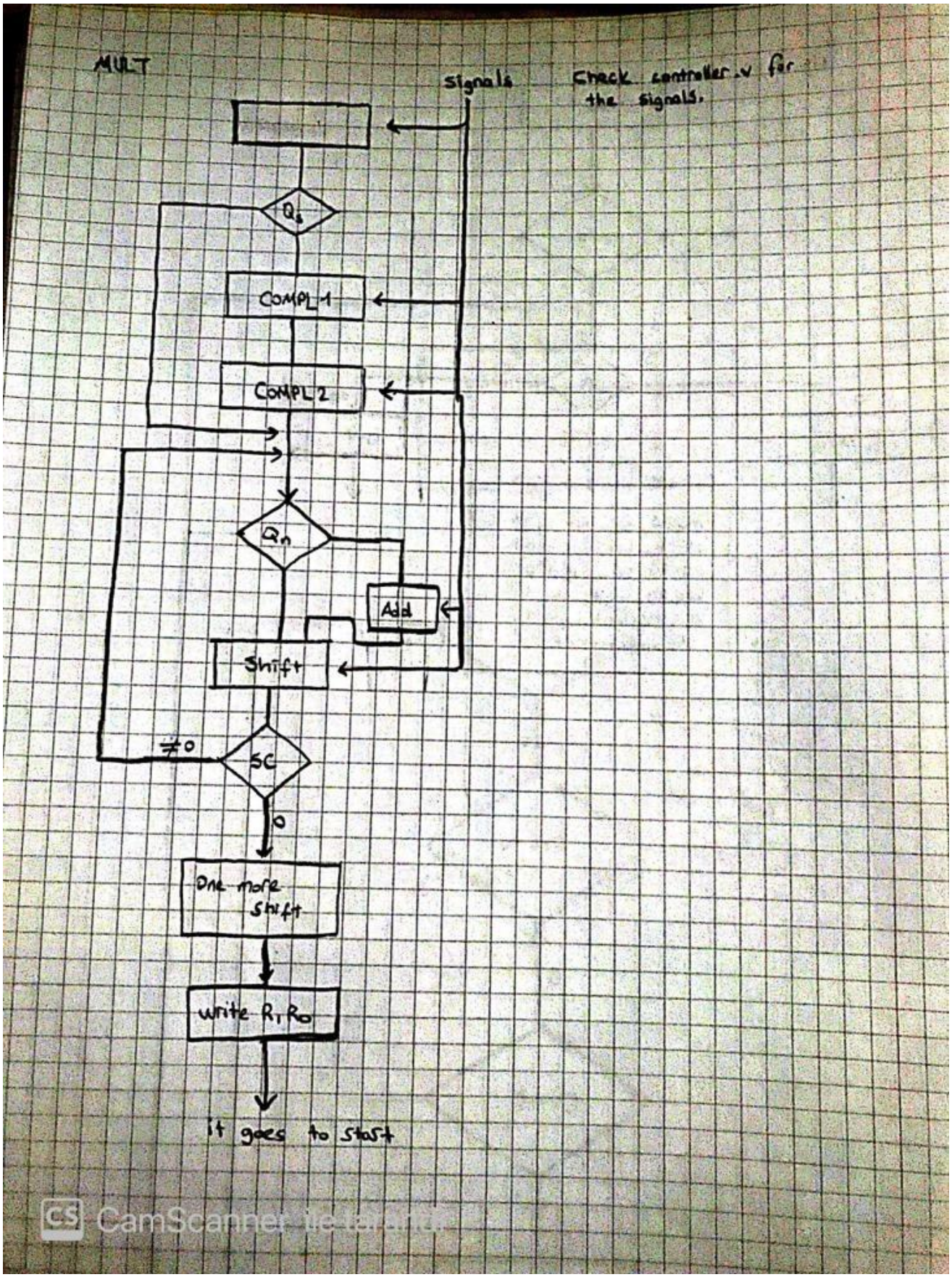
2.



3.



4.


MULT
signals
Check controller.v for the signals.
Q_s
COMPL 1
COMPL 2
Q_n
Add
Shift
≠0
SC
0
One more shift
write R_1 R_0
it goes to start

6)

Considering the overall design,

Except for MULT, COMP takes one cycle.

MULT takes minimum 7 cycles, maximum 9 cycles

LOAD takes 1 cycle

CLR takes 1 cycle

7)

It consists of a total of 19 states. STATE includes CLR, LOAD, and COMP. COMP includes eight states for each ALU operation. The operation of multiplication has its own set of states.

If the OP changes, the output may no longer be visible.

8) I implemented my controller in Verilog HDL.

Simulation Results

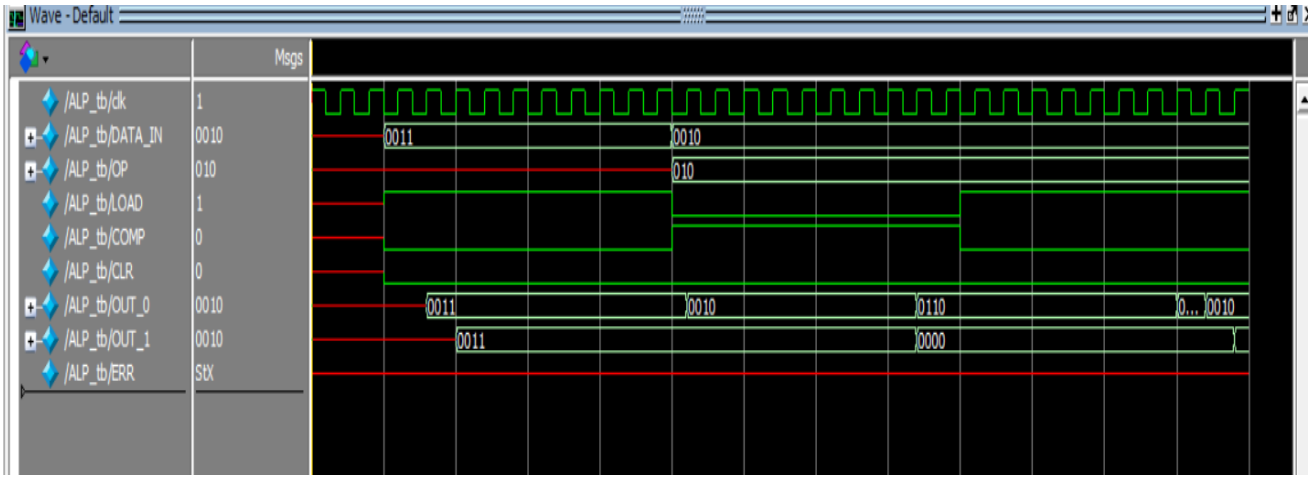


Figure 8. Simulation of multiplication operation with Modelsim

As you can see, first I load R0 with 0011, then I load it to R1. Next, I load R0 with 0010. As expected, I get 0110.

{R1, R0} ← R1 × R0 which is 0000 0110 which is correct.

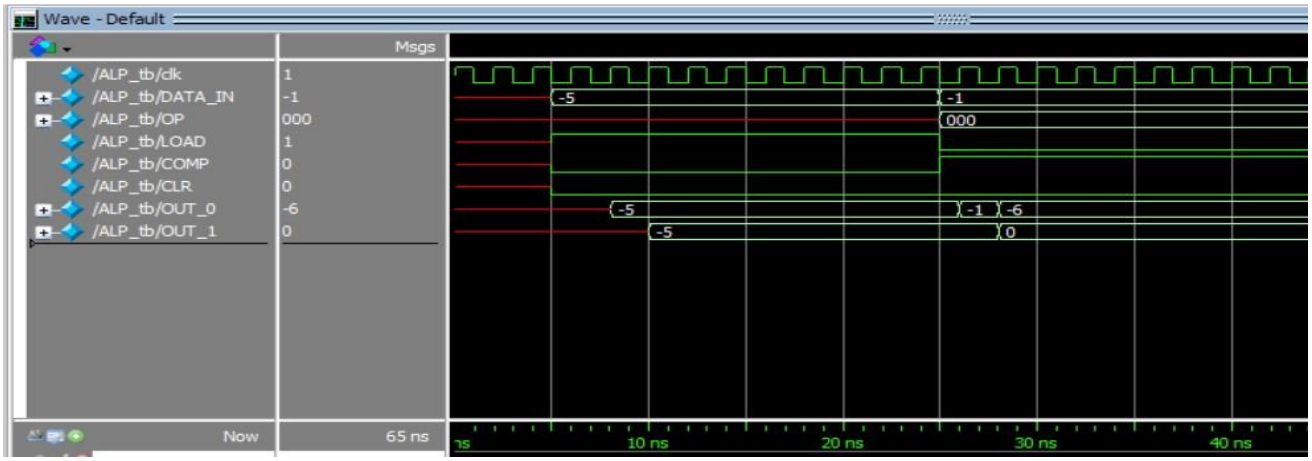


Figure 9. Simulation of addition operation with Modelsim (negative signed)

As you can see, first I load R0 with -5, then I load it to R1. Next, I load R0 with -1. As expected, I get -6.

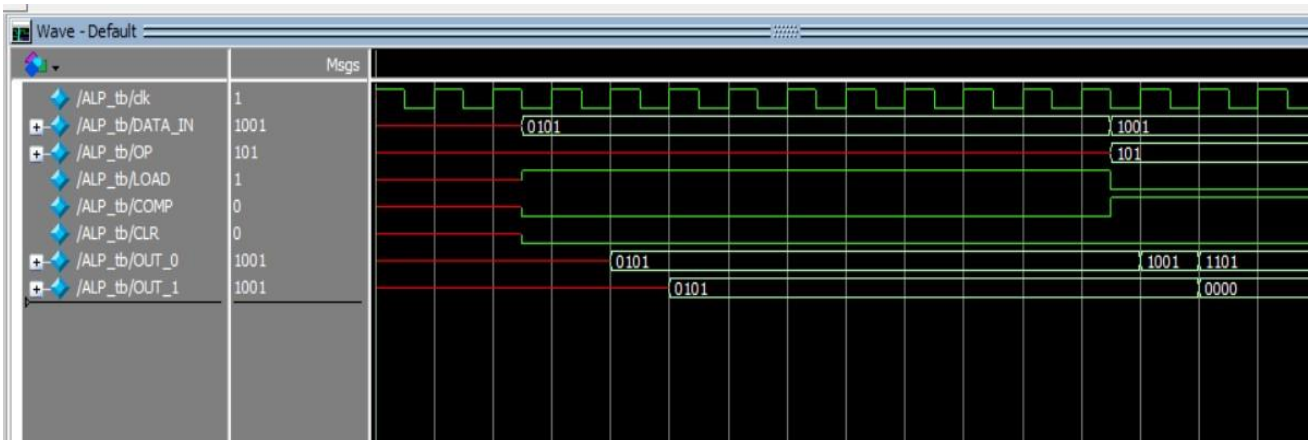


Figure 10. Simulation of OR operation with Modelsim

As you can see, first I load R0 with 0101, then I load it to R1. Next, I load R0 with 1001.

As expected, I get 1101.

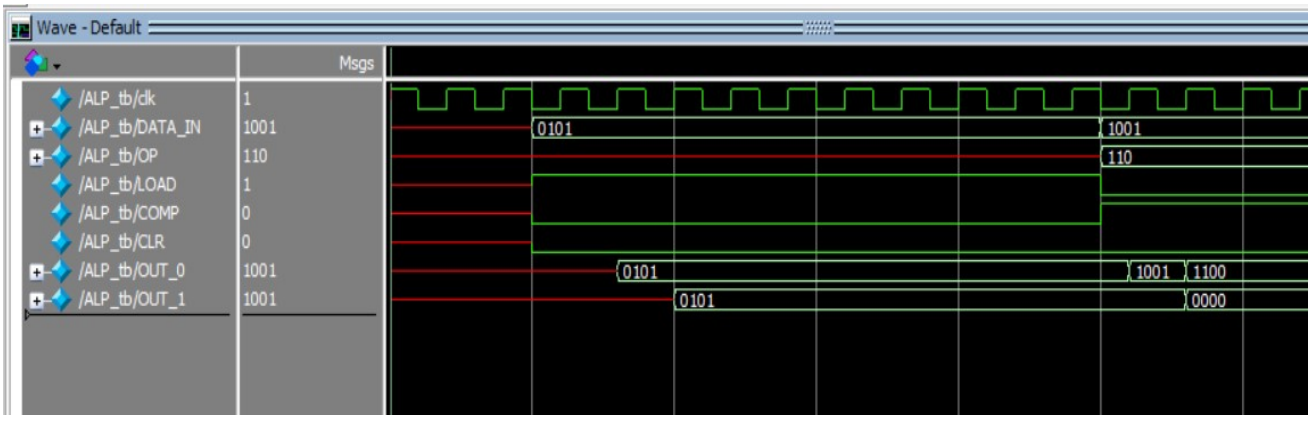


Figure 11. Simulation of EXOR operation with Modelsim

As you can see, first I load R0 with 0101, then I load it to R1. Next, I load R0 with 1001.

As expected, I get 1100.

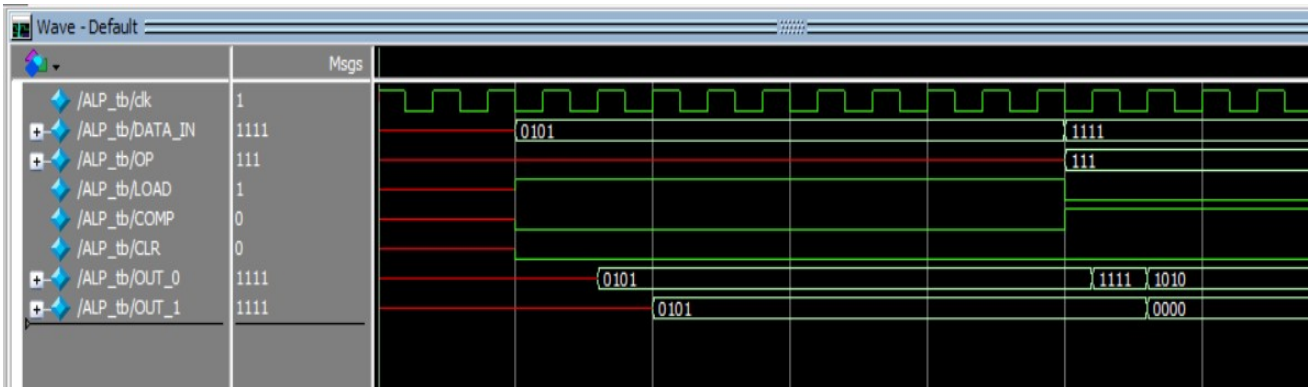


Figure 12. Simulation of BITC operation with Modelsim

As you can see, first I load R0 with 0101, then I load it to R1. Next, I load R0 with 1111.

As expected, I get 1010.

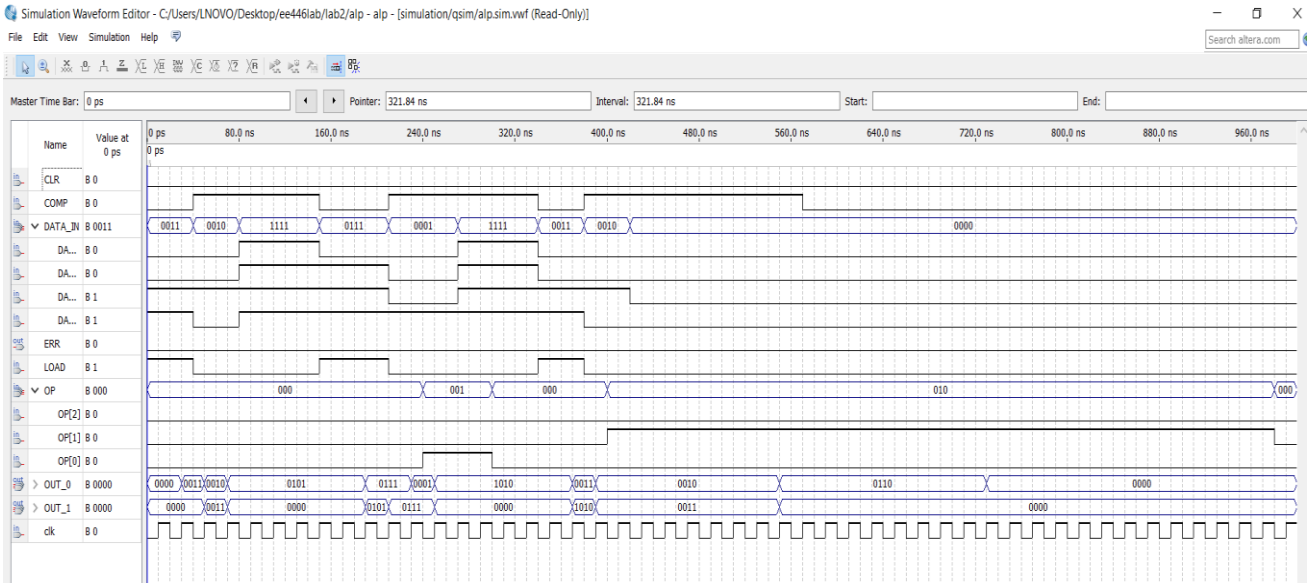


Figure 13. General simulation of addition, subtraction, multiplication and operation with Modelsim

As you can see, first I load R0 with 0011, then I load it to R1. Next, I load R0 with 0010.

When OP is 000, the result is 0101 as expected.

Second, I load R0 with 0111, then I load it to R1. Next, I load R0 with 0001.

When OP is 001, the result is 1010 as expected.

Third, I load R0 with 0011, then I load it to R1. Next, I load R0 with 0010.

When OP is 010, the result is 0000 0110 as expected.