Date: 27.05.2021
Author: Sławomir Batruch

I declare that this piece of work that is a basis for the grading of edc1 laboratory 6,7 task was performed on my own without any help (indirect or direct) from other students.

Sławomir Batruch; 303827

EDC1 – Sequencer design and testing (lab6 and lab7)

The task for lab5 and lab6 was the design of a sequencer and the testing of it. Starting from the flowchart, then implementing the flowchart in Xilinx ISE. After that, connecting the designed sequencer unit to a slightly modified operational unit from lab5. All of this is tested via a VHDL testbench simulation.

Part 1 – Flowchart design and algorithm

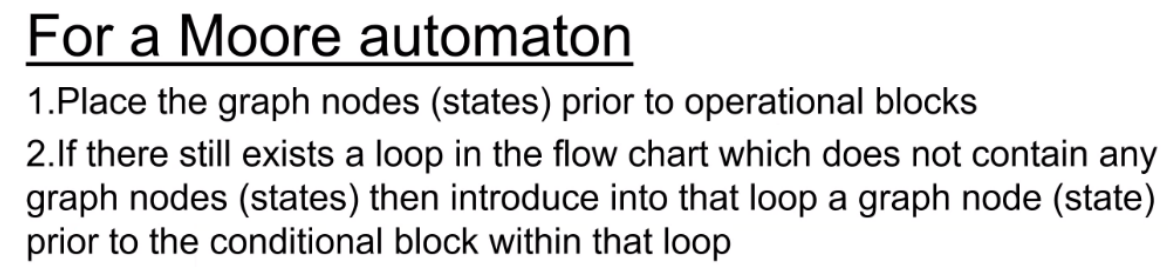
First, I designed a flowchart. I followed all the necessary steps that the circuit needs to take in order for it to work properly. Here is a simplified algorithm:

1. Set start to 1, clear registers
2. Enable load of registers storing N. Await until the number is supplied
3. Check state of all_loaded output.
4. If it's not 0
    a. Await a number
    b. If it's supplied, enable load on result register, and countdown on N-storing register (for tracking the number of numbers processed)
    c. If NRdy is changed to 0, repeat from point 0 if the condition is fulfilled
5. If the all_loaded output is 1
    a. Check the status of shifting. If it's = 1, jump to next point
    b. Set sync_shift to 1
6. Set ResRdy to 1 and loop infinitely. If start is changed to 1, repeat from point 1

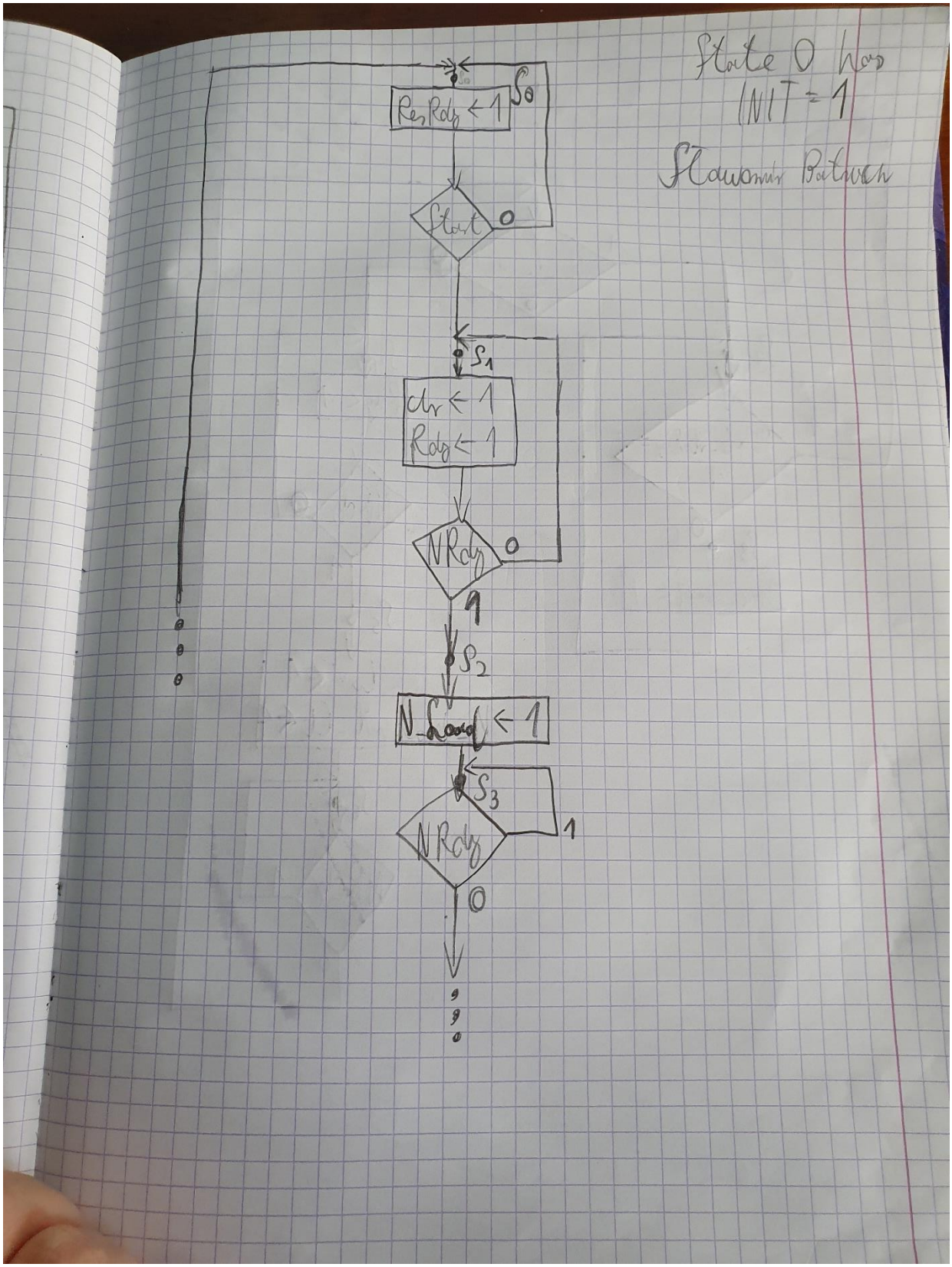
To remind, below is the written algorithm from the last laboratory regarding the operational unit:

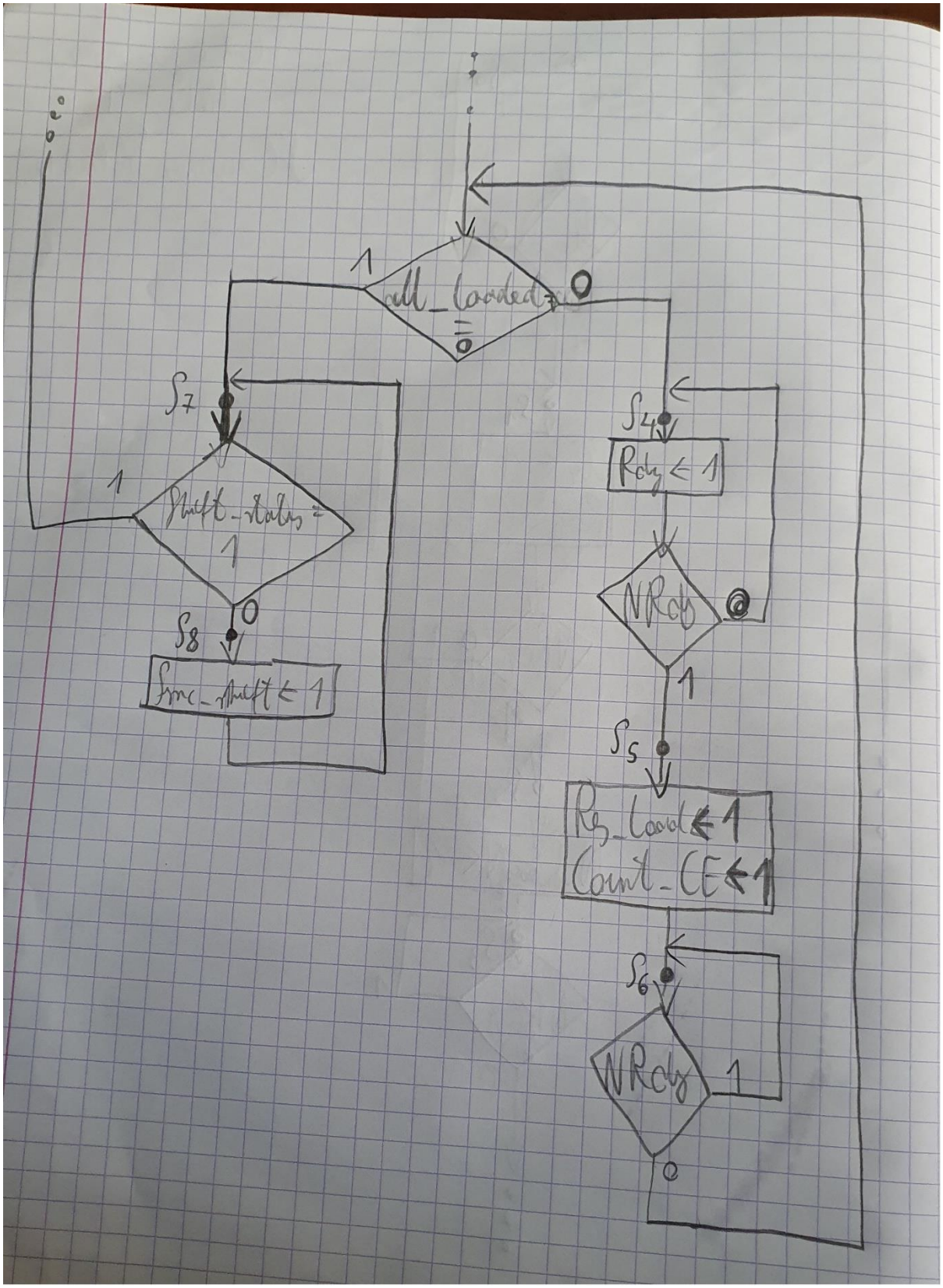
1. *Enable load on the counter and shift register from the bottom of the schematic*
2. *Load n into the input*
3. *Disable load on abovementioned counter and shift register*
4. *Load a number*
5. *Enable load on the register storing our result*
6. *Load numbers until the total of loaded numbers is N*
    a. *This is tracked by the counter decrementing by 1 every number loaded*
7. *When all numbers have been loaded, a control output will change to 1 (this indicates counter has reached 0 – so all numbers were loaded)*
8. *Disable the counter*
9. *Start shifting to right in both registers*
10. *When the register with n has shifted and obtained 00…0001, stop shift operation in both register*
    a. *This is tracked by an output indicating 1 when the shifting should stop, i.e. it indicates when bit at position 0 is equal to 1. This is due to n being a power of two, so it's binary representation is one '1' and rest is '0'*
11. *Declare that the result is ready and disable the LOAD and CLOCK_ENABLE function of all registers*

Applying these two algorithms (especially the first one), I designed the flowcharts and inserted the necessary states. States were inserted according to the Moore state insertion algorithm found on the lecture slides. Here is a snippet of it explaining the principle of state insertion:



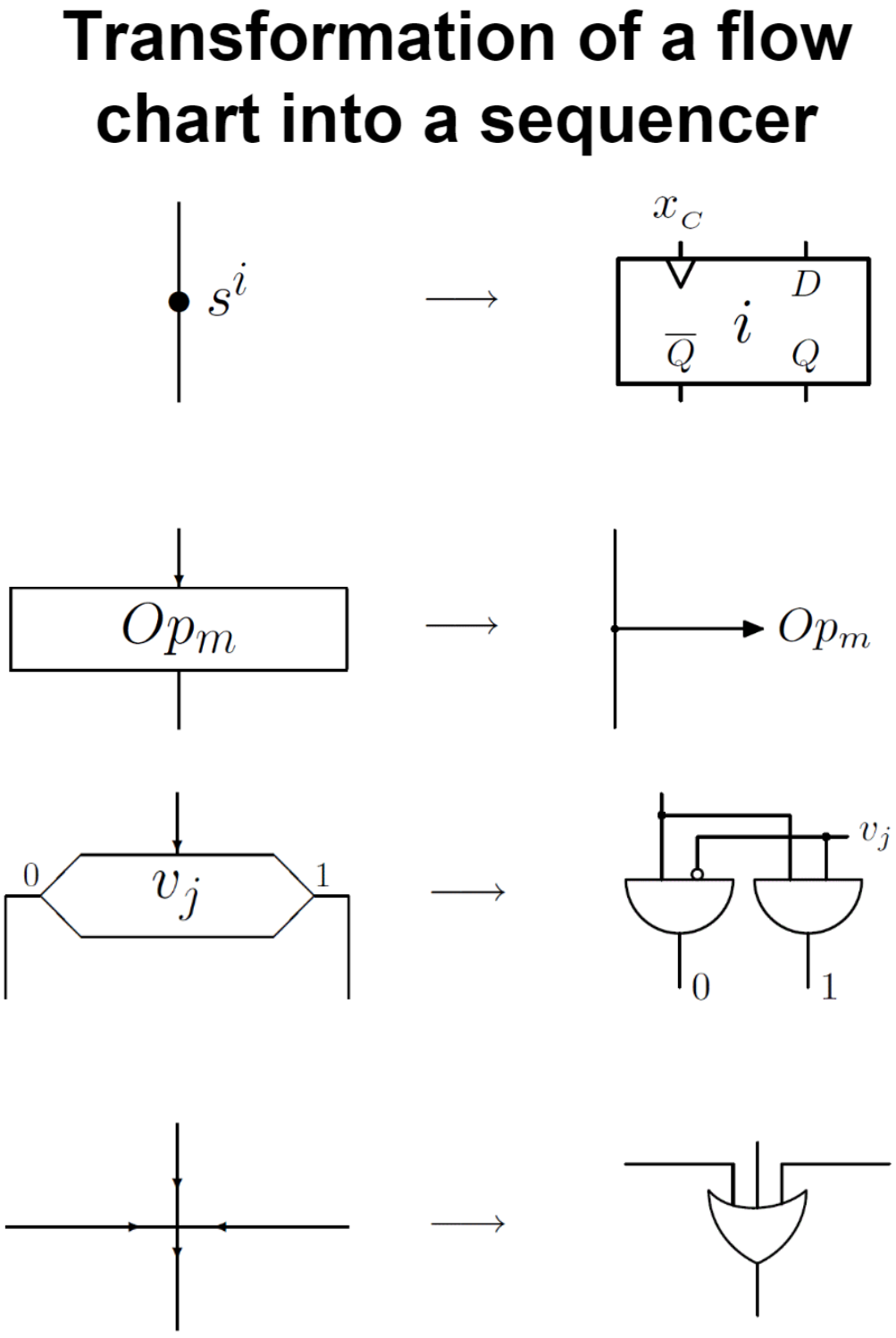
Furthermore due to some Moore automaton issues, if there is a operational block that changes a value, and this change might influence the next conditional block, a state should be inserted as well. For my case, state 8 on the flowchart was inserted according to this rule.


State 0 has
INIT = 1
Slawomir Patwich
Res_Rdy ← 1
S0
Start
0
S1
clr ← 1
Rdy ← 1
NRdy
0
1
S2
N_Load ← 1
S3
NRdy
1
0

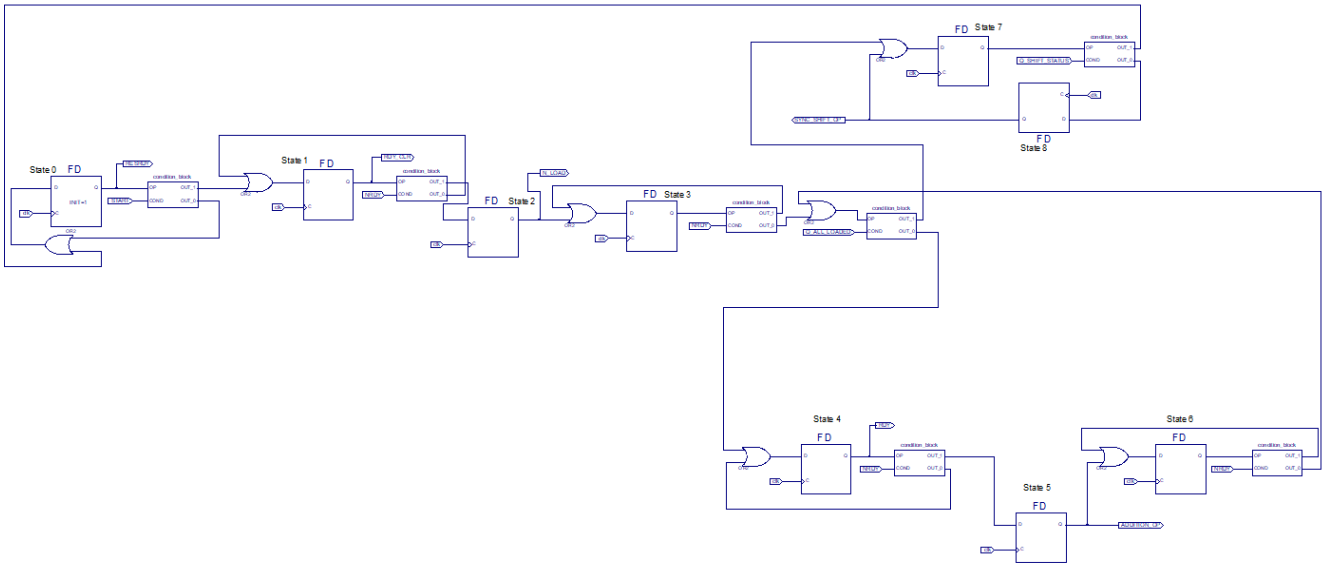

all_loaded
1
0
S_7
Mult_Rdy = 1
1
0
S_8
Src_Mult ← 1
S_4
Rdy ← 1
WRdy
@
1
S_5
Rg_Load ← 1
Count_CE ← 1
S_6
WRdy
1
0

Step 2 – Sequencer schematic design

Basing on the designed flowchart, I converted it into a schematic in Xilinx. I used the following lecture presentation snippet to convert flowchart elements into Xilinx blocks:

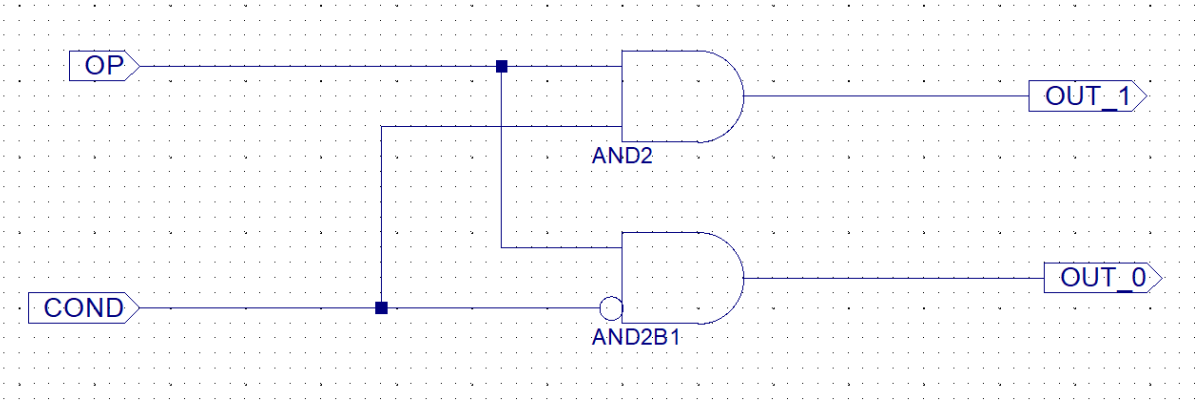


And here is the schematic of the sequencer, done accordingly to the abovementioned rules:



Please refer to the provided .zip file for better quality.

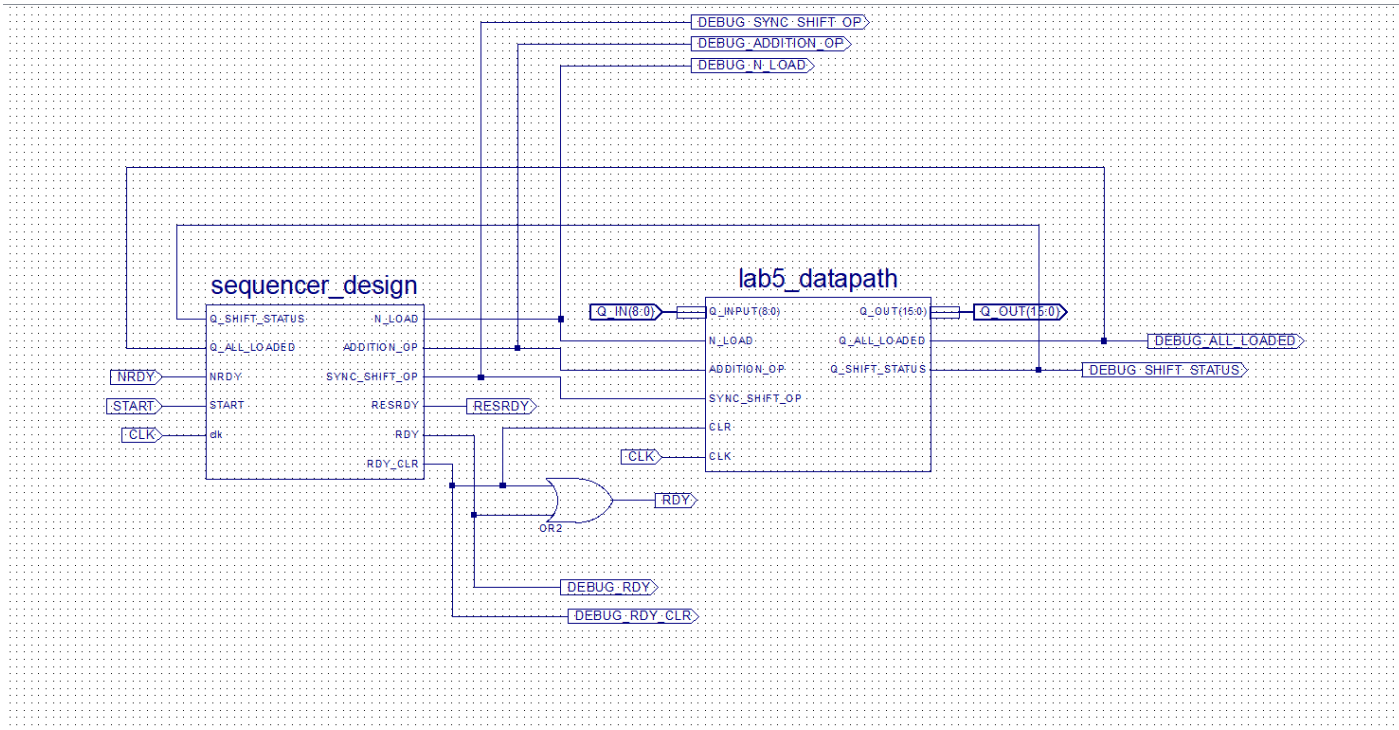
To clarify, the conditional_block is exactly the same as the circuit used for the conditional blocks. Here is how it looks like:



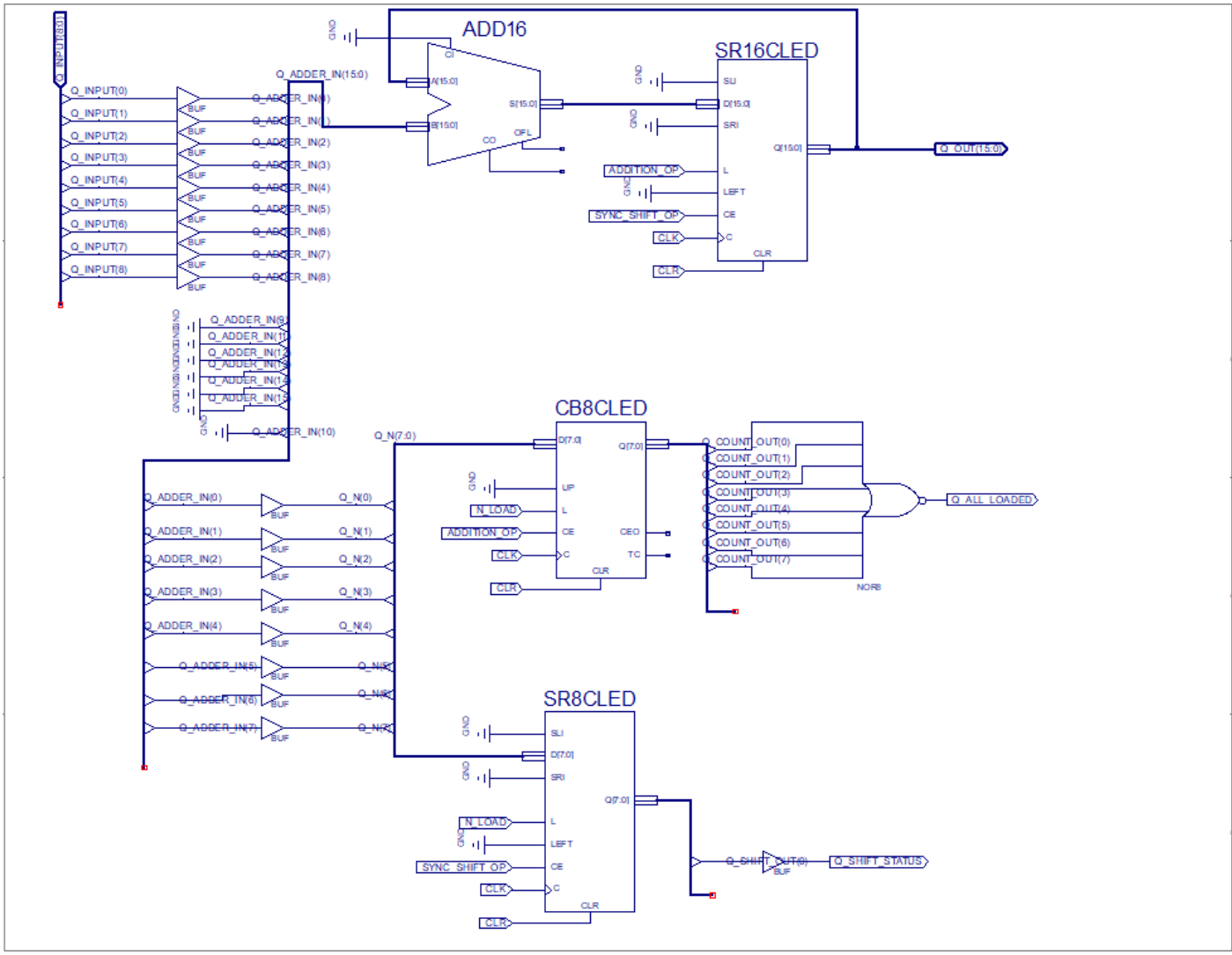
It passes signal from OP into OUT_1 if COND is TRUE, if COND is FALSE, the signal is passed into OUT_0.

Step 3 – connecting the Sequencer to Datapath unit

Connection of sequencer into the Datapath was done the following way:
Note that the outputs with DEBUG prefix are used to indicate proper working of the sequencer and Datapath unit.



Note that the Datapath unit has been modified so that it can be operated by the sequencer. Here is a screenshot showing how it looks like after slight modifications to input names (next page):



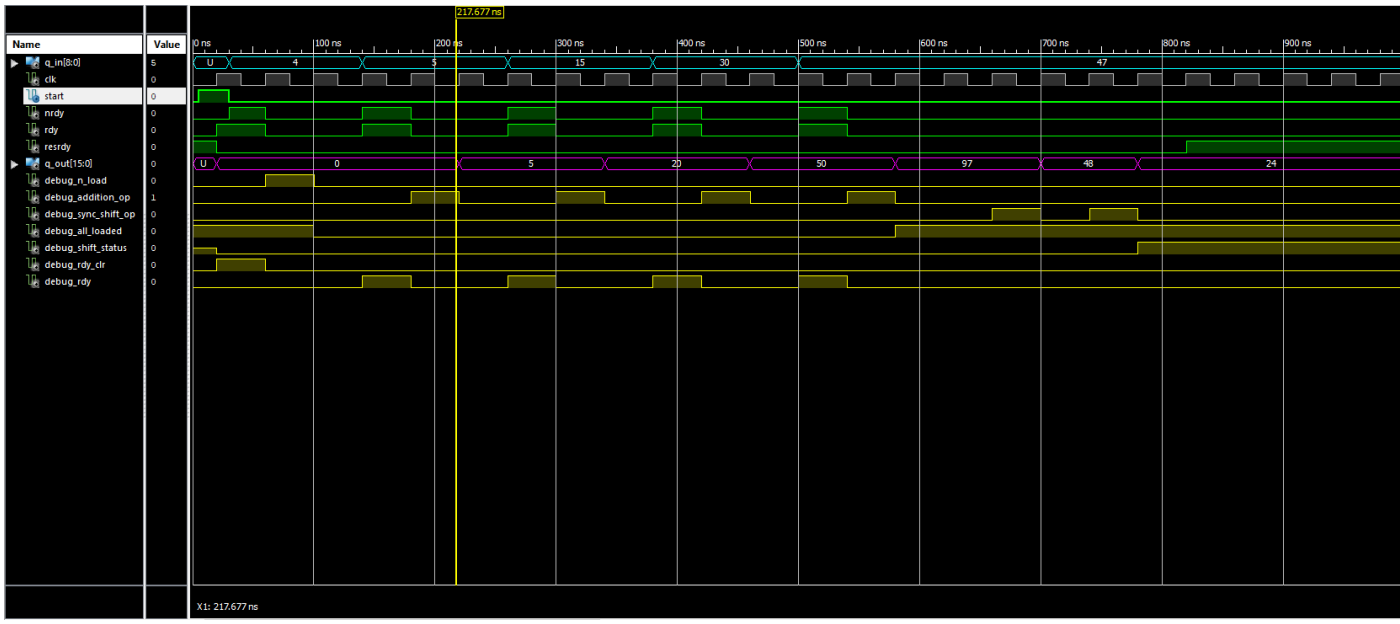
Step 4 – Design of the VHDL testbench

The design of VHDL testbench was done via debugging and “Trial & error” to find the proper timings for the NRDY and START inputs. Most problematic part was the start of the sequencer. The rest apart from the beginning was waiting for RDY to be TRUE – Then inserting the number and declaring NRDY as FALSE until RDY was back to TRUE state again. That all can be seen in an example code snippet taken from the VHDL testbench:

```
WAIT FOR 5 ns;
START <= '1';
WAIT FOR 25 ns;

START <= '0';

Q_IN <= "000000100"; -- N = 4
NRDY <= '1';
WAIT UNTIL RDY = '0';
NRDY <= '0';

WAIT UNTIL RDY = '1';
Q_IN <= "000000101"; -- 5
NRDY <= '1';

WAIT UNTIL RDY = '0';
NRDY <= '0';

WAIT UNTIL RDY = '1';
Q_IN <= "000001111"; -- 15
NRDY <= '1';

WAIT UNTIL RDY = '0';
NRDY <= '0';
```

Step 5 – Simulation, conclusions

Running the VHDL testbench and then changing order, radix, colours etc. in the simulation waveforms shows that the Datapath is properly controller by the Sequencer unit. This is how the simulation output looks like:



Refer to provided .zip file for better quality

The simulation was done for the following data:

$$N = 4; Numbers\ inserted: 5{,}15{,}30{,}47$$
$$Sum: 5 + 15 + 30 + 47 = 97; \frac{97}{4} = 24{,}25$$

The result is correct knowing that it was rounded down due to the nature of binary numbers. When 97 (binary 1100001) was shifted to right, the “1” at LSB was lost, resulting in decimal 48 (binary 110000). After next shift, 24 was obtained (binary 11000), which is our result.