# Report from Lab 6 & 7

# Functioning sequencer and operational unit

I declare that this piece of work, which is the basis for recognition of achieving learning outcomes in the Digital Circuits course, was completed on my own.

First and last name: Michał Łezka
Student record book number (Student ID number): 303873
Date: 28.05.2021

## Theoretical design

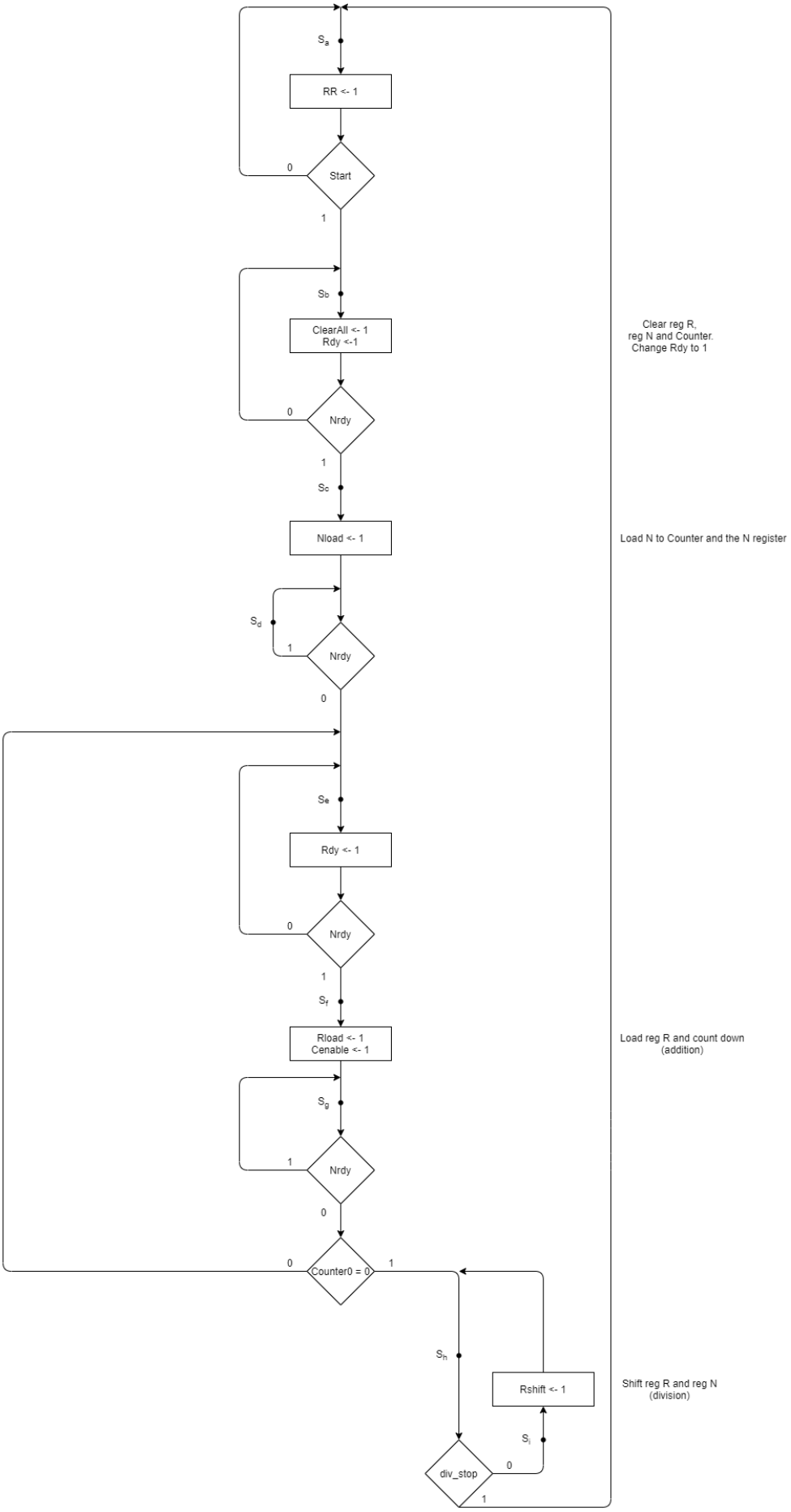
Our main goal is to create a digital system computing following formula:

$$Y = Int\left[\sum_{i=1}^{n} x_i / n\right]$$

As for the operational unit, we can use schematic of operational unit created during previous laboratory (lab 5), but with small adjustments. As for the sequencer, first thing that had to be done was an algorithm:

1) Clear all (reg R, reg N, Counter)
2) Load number N to Counter and N reg
3) Load next number to reg R and count down once
4) Compare Counter output if it's equal to 0 or not
   a) if Counter0 == 0, go back to point 3); otherwise continue
5) check whether output div_stop from operational unit is equal to 0
   a) if it is, shift reg R and reg N and go back to 5); otherwise continue
6) Set ResRdy to 1 and wait

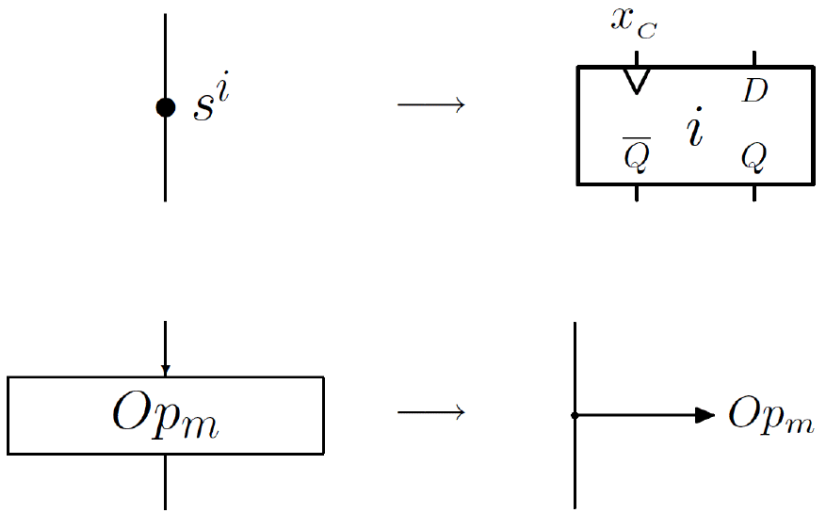
After creating the outline for the program, I have created a flowchart based on it. States were assigned according to the Moore automaton style. I had to add states $S_g$ and $S_h$ because of the problem with Moore automaton that occurs when tested condition depends on the operation just prior to it. It is true only for microoperations that are executed synchronously. To fix that, I had to introduce a new state in between the operational and conditional blocks. The result is the graph on the next page, with a few comments made on the side.


$S_a$
RR <- 1
0
Start
1
$S_b$
ClearAll <- 1
Rdy <-1
Clear reg R,
reg N and Counter.
Change Rdy to 1
0
Nrdy
1
$S_c$
Nload <- 1
Load N to Counter and the N register
$S_d$
1
Nrdy
0
$S_e$
Rdy <- 1
0
Nrdy
1
$S_f$
Rload <- 1
Cenable <- 1
Load reg R and count down
(addition)
$S_g$
1
Nrdy
0
0
Counter0 = 0
1
$S_h$
Rshift <- 1
Shift reg R and reg N
(division)
$S_i$
0
div_stop
1

Now, using that flowchart, I had to transform it into a sequencer. This was done using following translations from lecture:

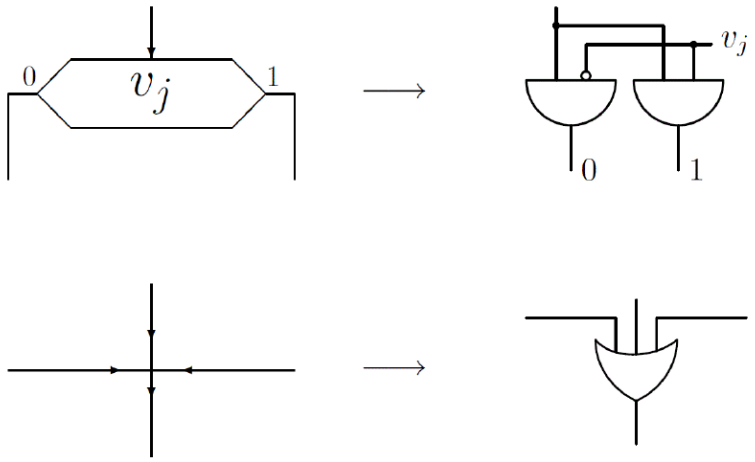


# Transformation of a flow chart into a sequencer




107





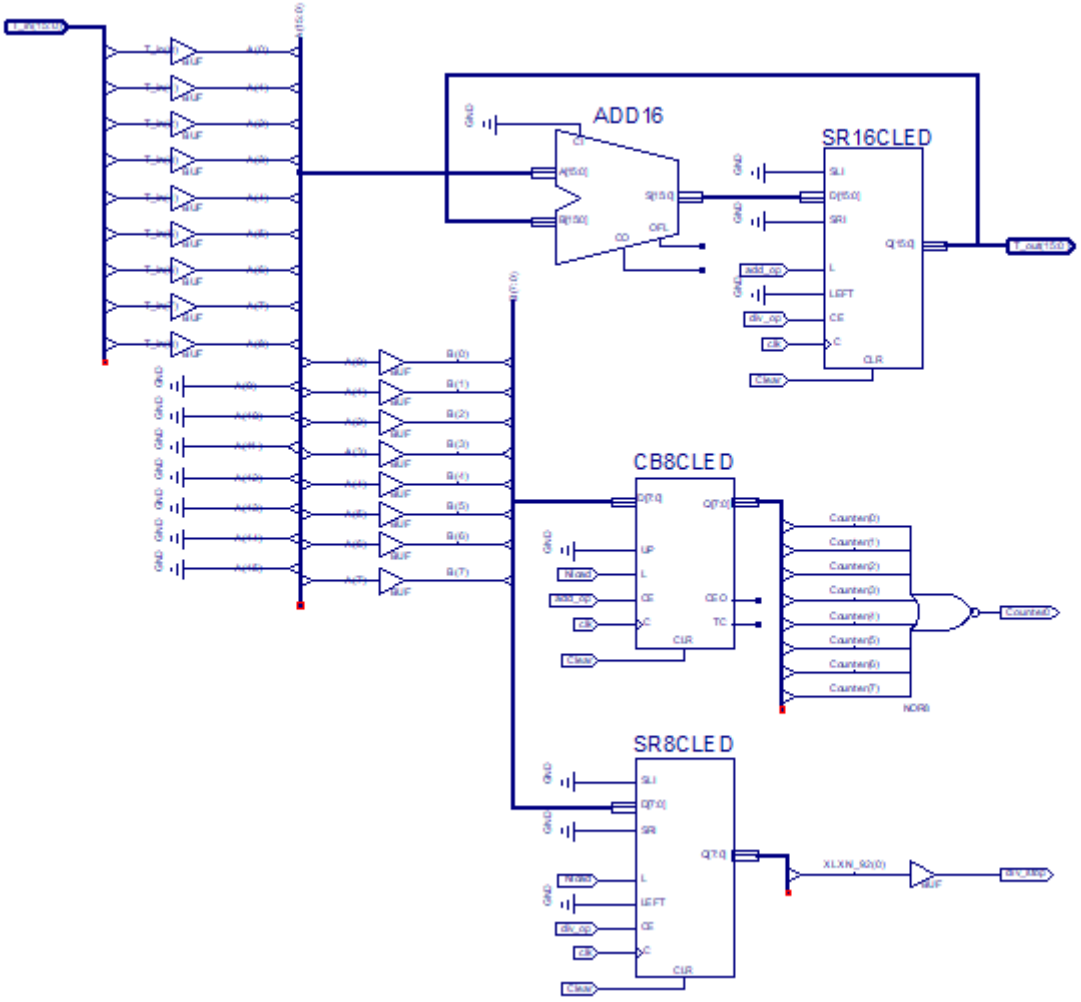

$$\begin{array}{cccccccccc} & & q_{n-1} & \dots & q_{i+1} & q_i & q_{i-1} & \dots & q_0 \\ s^i & \rightarrow & 0 & \dots & 0 & 1 & 0 & \dots & 0 \end{array}$$

108

The task here was rather straightforward, but to speed the process up, I have created a conditional block, so I don’t have to draw two AND gates every time.

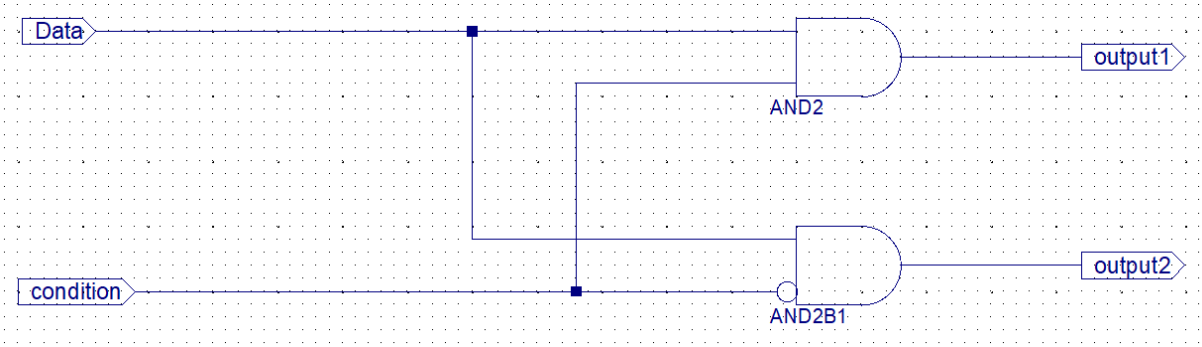
# Xilinx schematics

Operational unit was taken from the previous laboratory, however it had to be slightly adjusted by deleting, now unnecessary, debug outputs and changing some input names and merging them:

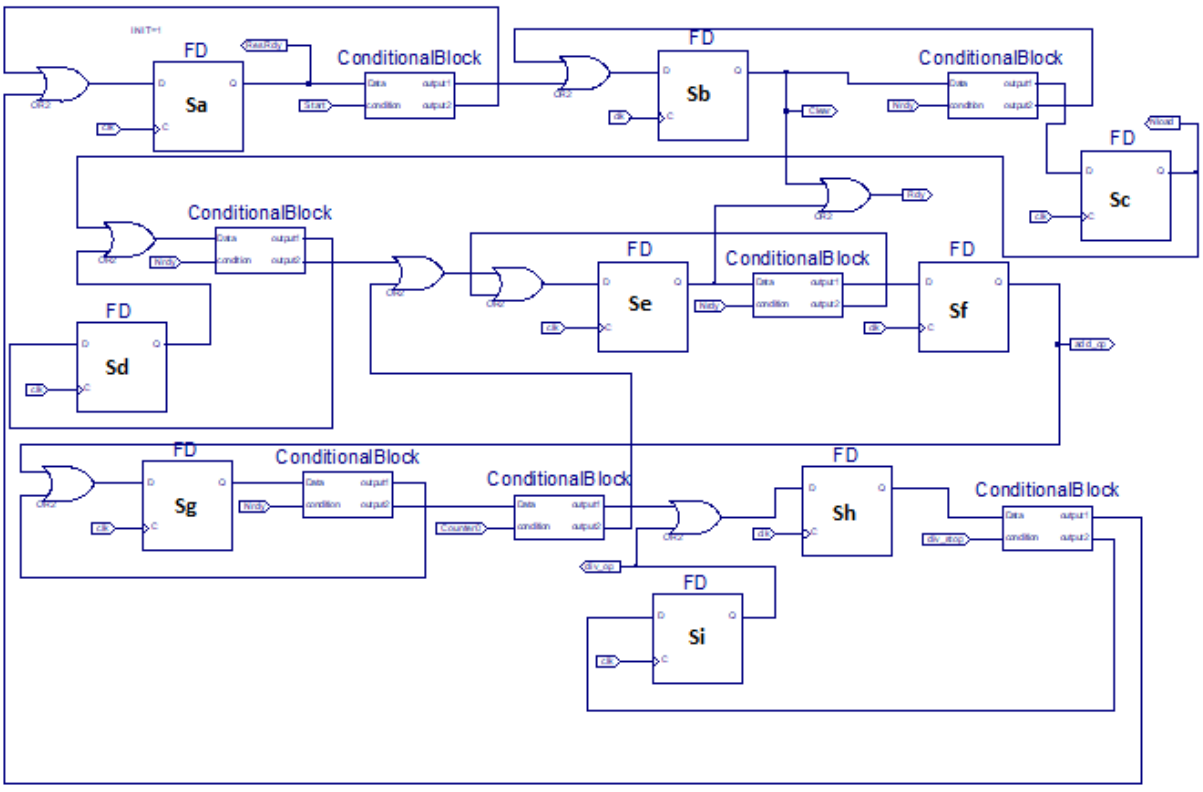



To recap, here is the description of how it works:
From T_in(15:0) we take first 9 bits as k assigned to my group was 9. From those 9 bits, I have transferred 8 to the counter and bottom register for the first clock cycle, because during that cycle through input comes n. After that, we no longer load counter and bottom register and start loading adder and, in result, top register. They are loaded with 9 lowest bits from input and rest is filled with 0. Addition takes place until counter reaches 0. Then bottom register starts shifting together with the top register until lowest bit of the bottom register is equal to 1. After that, we have our result ready.

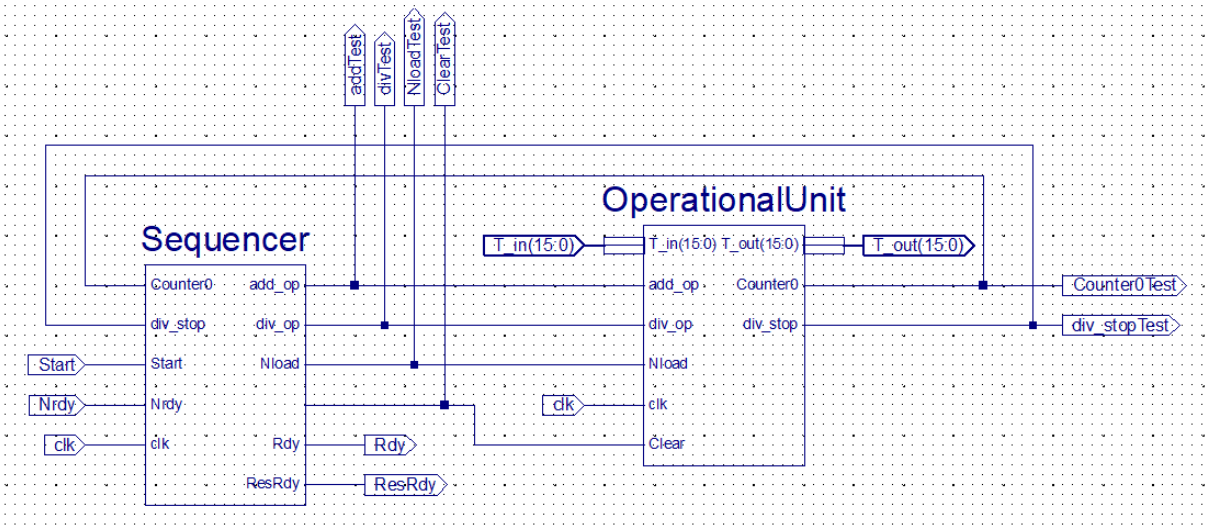
Conditional blocks that are used in the sequencer:




As for the sequencer, it is very similar to the previously shown flowchart, but has some additions/changes. First, I have added initial value = 1 to the first flip flop so that ResRdy is equal to 1 at the beginning and program can start functioning. Second change was to split ClearAll and Rdy after State b, so now in it's place there is clear output and a cable going to OR2, merging two Rdy's that I have on schematic.
To make the schematic more readable, I have added state names on proper flip-flops.



That leads us to the last part, where we change Sequencer and Operation Unit schematic into blocks that will be used in the final schematic. Now all that is left is to connect Sequencer with operational unit and add needed inputs and main output.




For debugging's sake, I have also created other outputs to see how the system works and to fix any problems that appeared.

## VHDL code

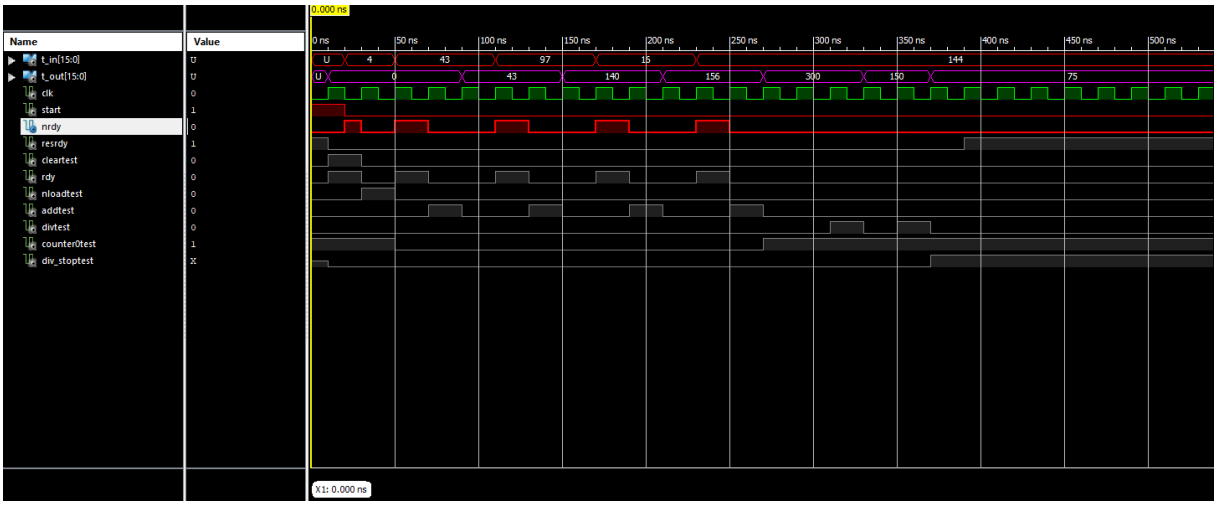
Now there needs to be created a testbench for the last schematic. Initial values that were assigned are clk = 0, Start = 0 and Nrdy = 0

How the program works and what is responsible for what is mostly described in comments for the code below. The whole process was centred around waiting for Rdy to change and with that, changing Nrdy.

```
72  -- *** Test Bench - User Defined Section ***
73     clk <= not clk after 10 ns;
74     tb : PROCESS
75     BEGIN
76
77     Start <= '1';
78     wait for 20 ns;
79     Start <= '0';
80
81     -- as it was proven during previous laboratory, the first 7 bits don't matter
82     -- so I set them all to 0 for simulation clarity
83     T_in <= "0000000000000100";      -- set n to 4
84
85     Nrdy <= '1';
86
87     wait until Rdy = '0';
88     Nrdy <= '0';
89
90     -- loop 1
91     wait until Rdy = '1';
92     T_in <= "0000000000101011";         -- T1 = 43
93     Nrdy <= '1';
94
95     wait until Rdy = '0';
96     Nrdy <= '0';
97     --
98
99     -- loop 2
100    wait until Rdy <= '1';
101    T_in <= "0000000001100001";      -- T2 = 97
102    Nrdy <= '1';
103
104    wait until Rdy <= '0';
105    Nrdy <= '0';
106    --
107
108    -- loop 3
109    wait until Rdy <= '1';
110    T_in <= "0000000000010000";      -- T3 = 16
111    Nrdy <= '1';
112
113    wait until Rdy <= '0';
114    Nrdy <= '0';
115    --
116
117    -- loop 4
118    wait until Rdy <= '1';
119    T_in <= "0000000010010000";      -- T4 = 144
120    Nrdy <= '1';
121
122    wait until Rdy <= '0';
123    Nrdy <= '0';
124    --
125
```

# Simulation results

Clock was marked with green colour, remaining inputs with red colour. Output was marked with pink colour. Outputs created for debugging were marked with grey colour.




Numbers inputted into the program where in order: 4, 43, 97, 16, 144. That means n = 4 and the remaining four numbers are to be added. Adding them one by one:

43 + 97 = 140
140 + 16 = 156
156 + 144 = 300

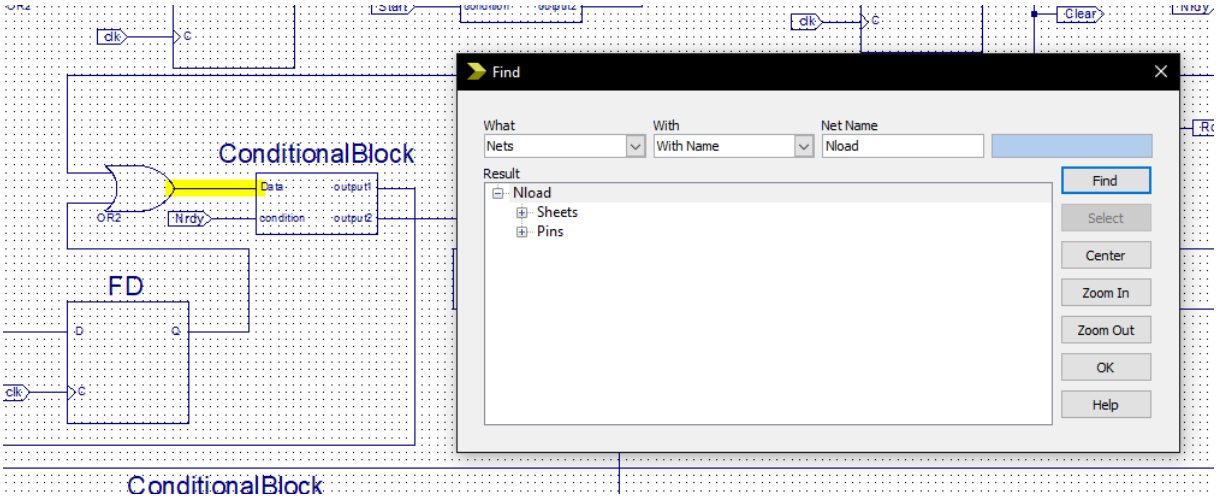
As we can see, program added everything correctly. Now for the division:

300/2 = 150
150/2 = 75

Once again, program executed everything correctly.

## Troubleshooting

There were many issues that needed fixing. Firstly, theory-related. At the begging I didn't take into consideration the problem with Moore automaton that I have mentioned in Theoretical Design part. Because of that problem, I had to move $S_g$ from the "outside" branch to the main branch so now it has to be always executed before checking Nrdy. It also forced me to add an additional state $S_h$ as it wouldn't work without it.
Some problems were caused by the Xilinx software however, because as I was changing and fixing the Sequencer, some net names were incorrect like for example this:




Because in the beginning I forgot to put OR gate, after I have added it, I didn't delete all the connected nets. Because of that, the name stayed, which caused some very unclear problems when looking at the simulation.

## Conclusions

In the end, after a lot of debugging, the program worked correctly. In the end, I have succeeded in creating a device calculating the average number.