BYOC\_HW4:

Tsahi Barshevsky 311334544

Kostya Lokshin 310765821

Implementation report is at the end of this file

**Simulation Report**

3.1) Attach screen captures showing ck cycles 70 to 118 (following the end of the reset pulse –

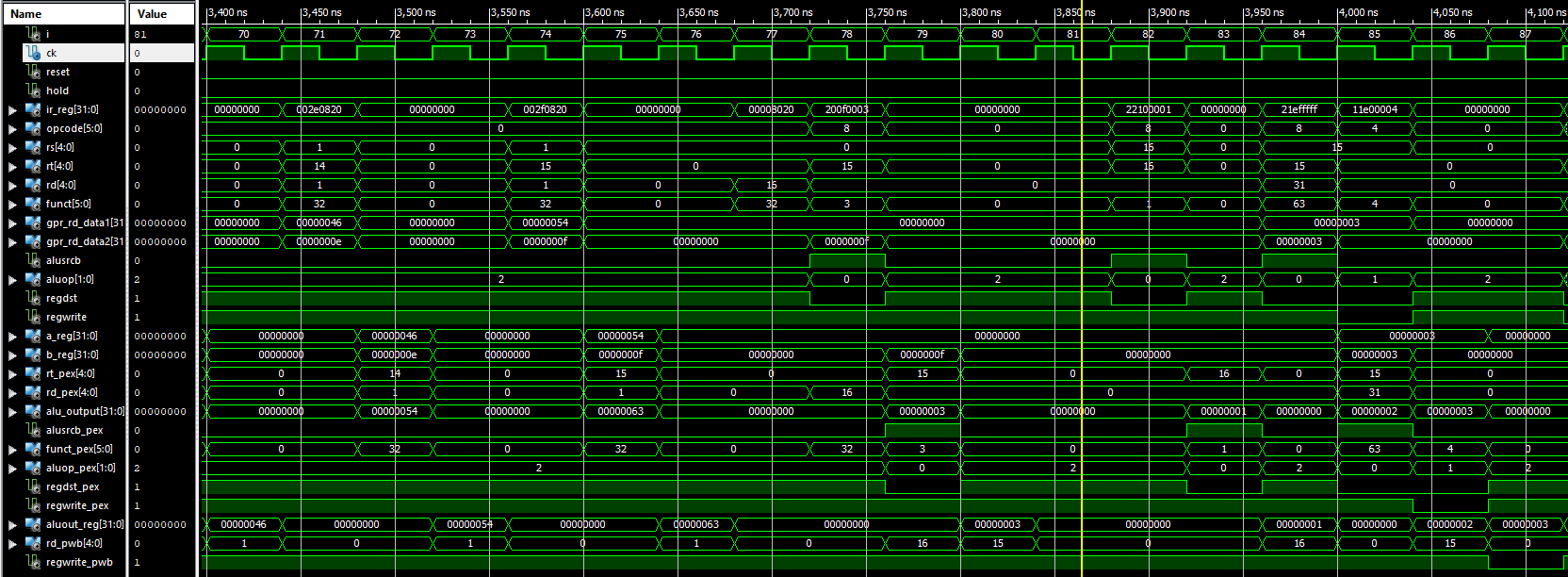
use the i signal value of the TB to count the CK cycles) and make the values of all signals

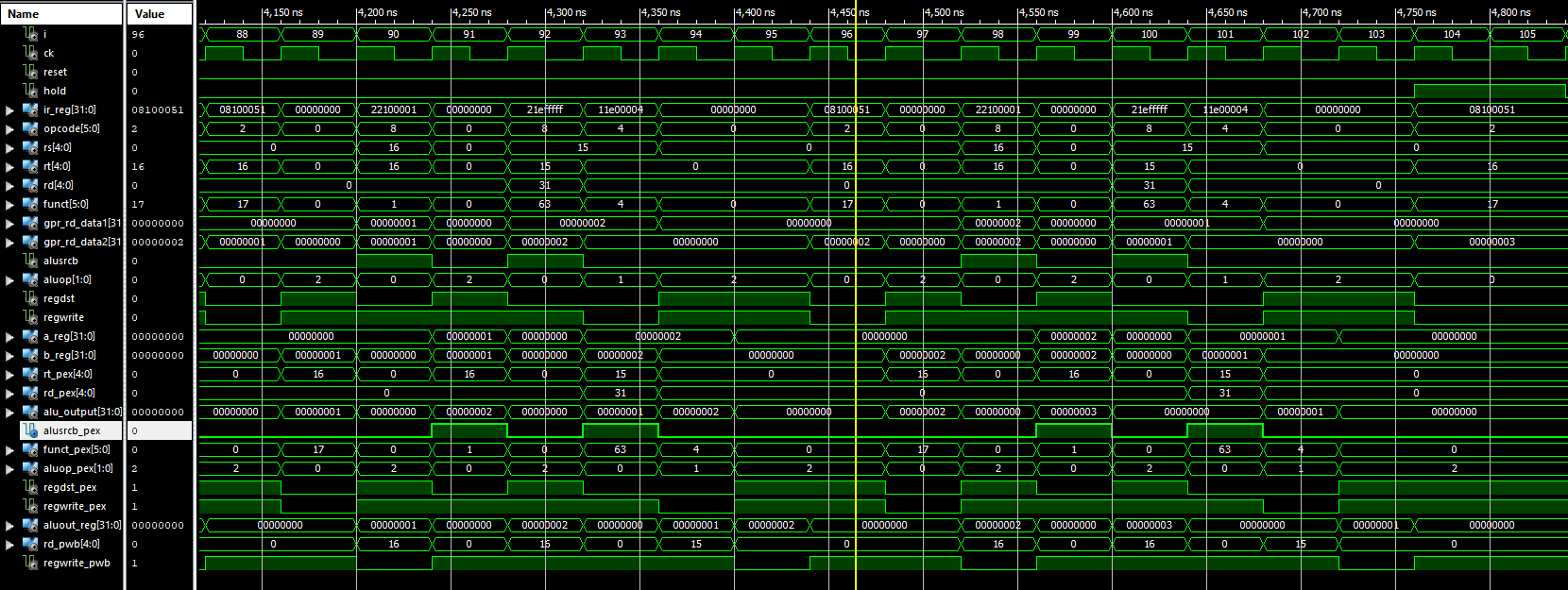
connected to rdbk0-15 readable. All of the signals mentioned in 2.c should be presented in

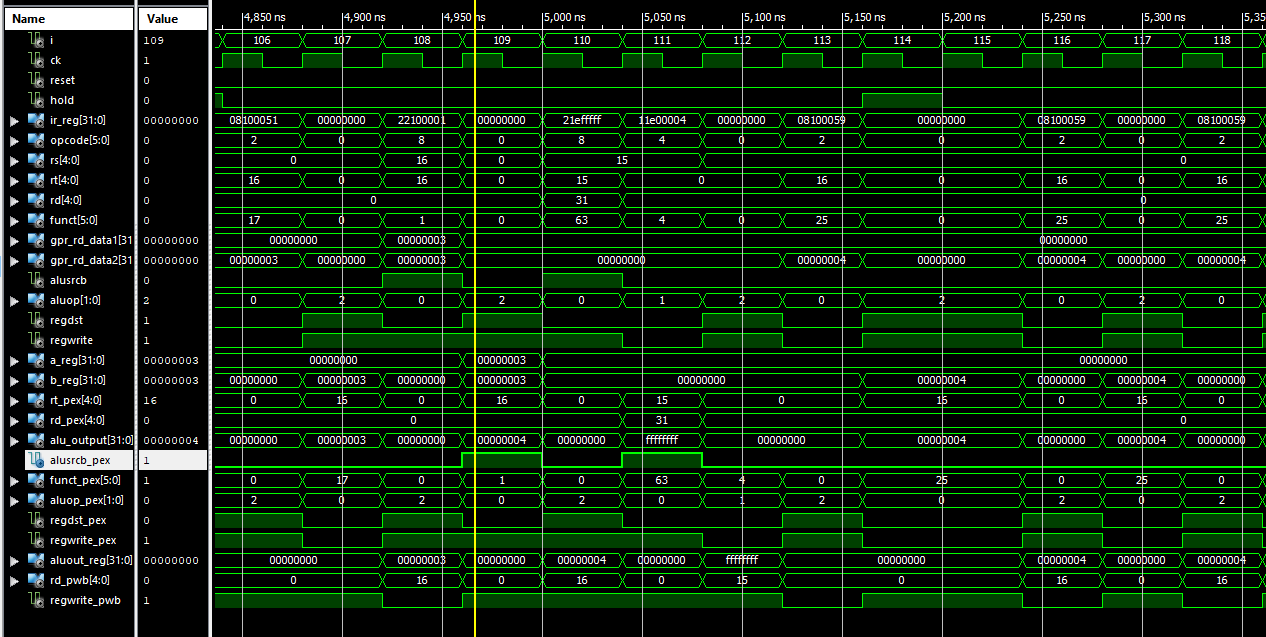
the screen capture. The signal names should be readable as well. These are not TB\_rdbk0 -

TB\_rdbk15 of the TB or rdbk0\_out\_to\_TB-rdbk15\_out\_to\_TB of HW4\_top\_4sim, but PC\_reg,

IR\_reg etc. You may use the signals rdbk3\_vec, rdbk4\_vec, rdbk7\_vec & rdbk13\_vec.







3.2) Explain in details what you see in this screen capture: You should show the execution of the

program using a scheme as described below. Specifically you should show the values of

registers $16 and $15 and to specify the point in time where they changes (in a similar

manner to the drawing below)

Val in $16 ?? 0

Val in $15 ?? 3

PC= 400130 inst= add $16,$0,$0

PC= 400134 inst= addi $15,$0,3

PC= 400138 ….

Put a line like that when $16 or $15 are updated

Also mark the ALUOUT\_reg value at the WB stage when $16 is written to in **RED** (in a similar

manner to the drawing above) on the screen capture. And mark the ALUOUT\_reg value at the

WB stage when $15 is written to in **BLUE**.

Also mark the IR value of branch instructions in **ORANGE** and state whether we branch or not at

that instruction. You can use Excel for drawing this and attach the XL file in the Doc directory.

The purpose of this is to make you familiar in looking at the waveforms and understanding what

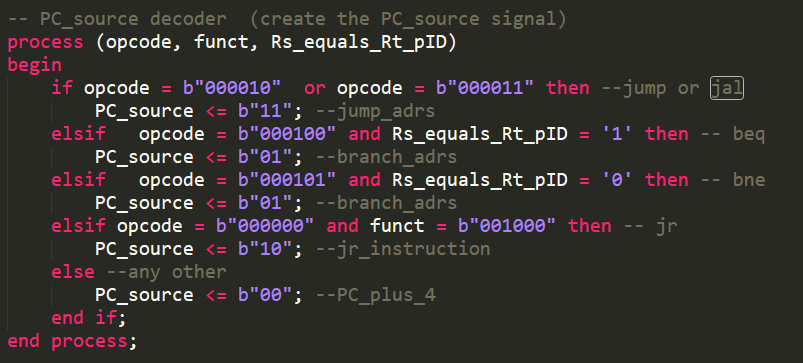
happens in the design via simulations. This knowledge is required in HW5.

[Excel sheet is in this folder](HW4%203.2.xlsx)

3.3) Explain in detail the changes you did in the Fetch\_unit to support beq and bne instructions.

Attach the relevant vhdl code.

In purpose of supporting beq/bne we added an input to the fetch unit Rs\_equals\_Rt\_pID which tells us if the branch condition has been met and if so then set PC\_source to pick the branch address



3.4) Why do we have two nops in the following code (which is part of the code use in the

simulation phase):

x"004000A0" => x"00230820"

x"004000A4" => x"00000000"

x"004000A8" => x"00000000"

x"004000AC" => x"00810820"

x"004000B0" => x"00000000"

x"004000B4" => x"00000000"

x"004000B8" => x"00A10820"

x"004000BC" => x"00000000"

x"004000C0" => x"00000000"

x"004000C4" => x"00260820"

The nops is the time needed for the ALU result to be written in the GPR file, so that the right result will be written in the right register

3.5) The program we load into the IMem checks many functions & instructions. We do check the

sign extension for example. Which of the instructions or functions of the Rtype MIPS CPU

you built are not checked by the program we have in the IMem (the program we disassembled

in the report of HW3 and which is part of the HW4\_BYOC\_Host\_Intf\_4sim.vhd

file) ?

According to the simulation:

* The following instructions were not checked:
  + Sw, lw, lui, ori, bne, jal
* The following functions were not checked:
  + Sub, and, or, xor,slt, jr

**Implementation Report**

5.1) In which instructions do we see errors? (The instruction starts when we have it in the IR)

We see the error in the first instruction IR\_reg = x”20010001”

5.2) Why does that happen? List all of the reasons.

The reason for the error is that the compare file was written according to the memory state of zeroed GPR file, therefore when we run the simulation again we have now junk data in the GPR file which does not affect the rightfulness of the design, it only means that the compare file expecting to see clean memory for every simulation.

5.3) How can we eliminate this? Give a detailed explanation (VHDL code is preferred).

In order to get good simulation each run for this compare file we can make sure that on every run we clear the memory, we can do it by creating a 32 bit vector while each bit represent a register in the GPR file, then each time we access a register for the first time we set it to zero and mark it in the 32 bit vector as ‘1’ (cleared)