BYOC\_HW5:

Tsahi Barshevsky 311334544

Kostya Lokshin 310765821

Implementation report is at the end of this file

**Simulation report**

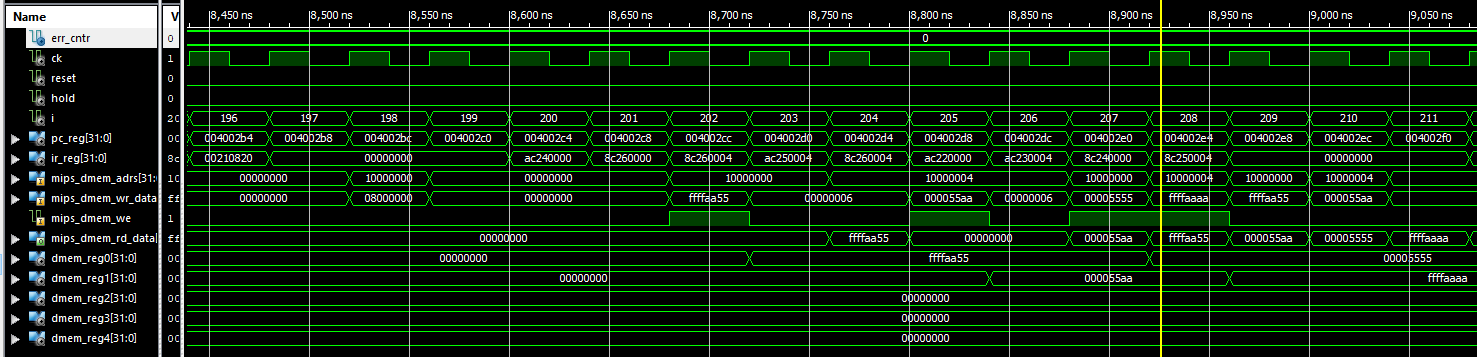
3.1) The listed below signals should be presented in the screen capture you need to attach to

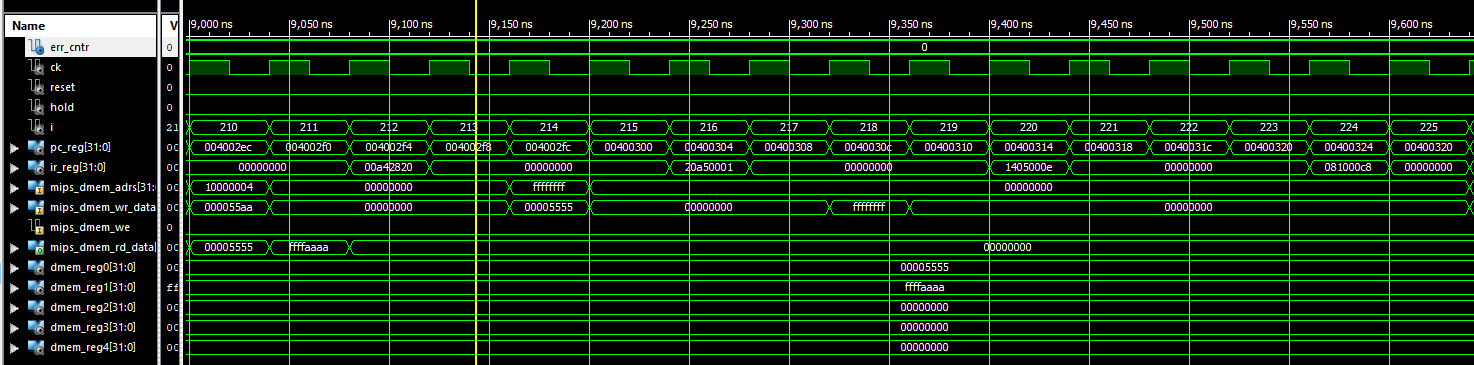
your report. Show clock cycles 196-224 (following the end of the reset pulse, find i=196-224)

and make the values of all signals readable. For this you will probably need to show clocks

196-210 and 210-224 separately. These are the signals that can help you in “testing” the

DMem.





3.2) Explain in detail what happens, i.e., what do we see here. Note that it is essential to the

success of your future design that you will verify that the design does what we wanted it to

do in these CK cycles.

**We store data using SW inside of a loop using branching to fill the memory**

3.3) What is the latency of an R-type instruction? That is: How many nop-s should be inserted

between two consecutive R-type instructions if the 2nd one uses the result of the 1st one?

**There should be 3 nop-s between two consecutive R-type instructions. i.e., the latency of a R-type instruction is 3 CK cycles.**

3.4) Explain the limitation of beq that tests a register that is calculated by Rtype instruction.

As an example, translate the following C if statement:

for (i=0;i<10;i++) { … }

where i resides in register $3.

**It takes 3 clocks to the data in rtype to be written in the memory so we need to make sure there is at least 3 clocks between the rtype instruction and the beq instruction**

**sw $3 0**

**sw $4 10**

**loop:**

**addi $3 $3 1**

**no op**

**no op**

**beq $3 $4 loop**

3.5) Are there any other limitations due to the pipeline structure in the instructions we

implemented (Rtype, addi, beq, bne, lw, sw, j)? How can we overcome these limitations

(e.g., by adding nop-s)? Try to list all of the **SW** & **HW** based solutions you can think of.

**We need to make sure that there is enough clock cycles between memory write/read commands so that there will be no conflict while reading/writing to a specific register**

**Implementation report**

1. What is the value of register $2 after 122 cks?

**The address of the first line: 20000018.**

1. What happens after 126 CKs?

**The beginning of printing a rectangle on the screen, i.e. the first line is printing on the  
screen.**

1. What happens when you press the RUN button?

**A rectangle has been drawn on the screen which connected to the board.**

1. Explain the **HW5\_rect4** program (what is the job of every register used. What is done in each loop, etc.)  
     
   **The main aim of the program is to draw a rectangle to the screen. It do that by:  
     
   Register $1 saves the number of lines of the rectangle that will be drawn.  
   Register $2 represents the address in the screen of the pixel that we want to change.  
   Register $3 is the value that allows you to paint specific pixels (each bit marked with 1 turns on the screen).  
   Register $4 is the loop counter for register $2.  
     
   label shft\_lp: Increasing Register $2 to 20 million.  
   label drawlp: write to screen the rectangle.   
   label end: end program. An endless loop.**
2. How long does it take [in seconds] to draw a 32x32 white square when we use the draw loop of the **HW5\_rect4** program?

**It takes 7 clock cycles to draw a single line, so drawing 32 lines will take us 7x31 = 217 clock cycles which are 8,680 nanoseconds.**

1. Can you shorten the loop? If you can, write the code and explain.

**The loop can be shortened by canceling the designated counter and using it in  
comparison to the final value expected from the register $2, i.e. 20003294 (52X 63).**

**the code:**

**drawlp: sw $3, 0($2) # write to screen**

**addi $2,$2,52 # increment ptr ($2)**

**nop**

**nop**

**bne $2,20003294, drawlp # if $2 not end addrs goto drawlp**

1. Can you think of a faster way to draw the square in the same short loop? If you can, write the code and explain.

A faster way to perform the painting in the draw loop is to define another register that will point to the next row (after the initial value of register $2) and to increase by 104 instead of 52 of the two registers.

The code:

**drawlp: sw $3, 0($2) # write to screen**

**sw $3, 0($5)**

**addi $1,$1,-2 # decrement counter**

**addi $2,$2,104 # increment ptr ($2)**

**addi $5,$5,104 # increment ptr ($7)**

**nop**

**nop**

**bne $1,$0, drawlp # if cntr not 0 goto drawlp**