Lab 3 Report

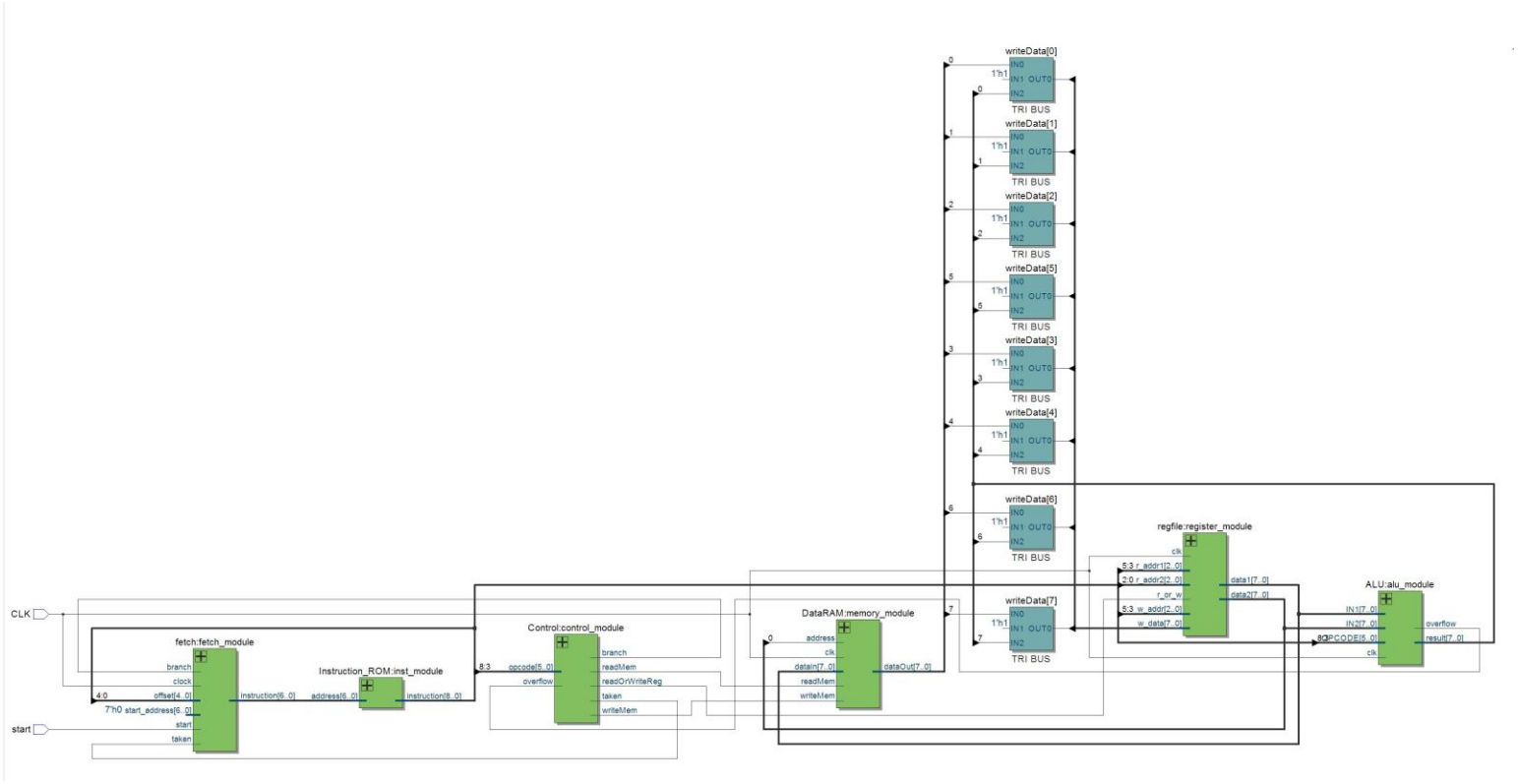
By: Brandon Chin (A11845998), Leandro Lubrico (A12077909), and Kevin Chen (A10623152)

Q1. Briefly describe what you did to connect the modules

To connect the modules, we defined each modules inputs and outputs as modules that we could pass into each module. There are wires that are outputs to one module and inputs to another. In this case, we followed the module signature and made sure that there weren't any concurrency issues.

Q2. Provide proof that it compiles in both Quartus and ModelSim

Quartus



ModelSim

```
# Model Technology ModelSim ALTERA vlog 10.4b Compiler 2015.05 May 27 2015
# Start time: 20:44:06 on Nov 18,2016
# vlog -reportprogress 300 -work work C:/Users/chaos_000/Dropbox/CSE141L/cse141l/alu.sv
# -- Compiling module ALU
#
# Top level modules:
#         ALU
# End time: 20:44:06 on Nov 18,2016, Elapsed time: 0:00:00
# Errors: 0, Warnings: 0
vlog -reportprogress 300 -work work C:/Users/chaos_000/Dropbox/CSE141L/cse141l/Control.sv
# Model Technology ModelSim ALTERA vlog 10.4b Compiler 2015.05 May 27 2015
# Start time: 20:44:06 on Nov 18,2016
# vlog -reportprogress 300 -work work C:/Users/chaos_000/Dropbox/CSE141L/cse141l/Control.sv
# -- Compiling module Control
#
# Top level modules:
#         Control
# End time: 20:44:06 on Nov 18,2016, Elapsed time: 0:00:00
# Errors: 0, Warnings: 0
vlog -reportprogress 300 -work work C:/Users/chaos_000/Dropbox/CSE141L/cse141l/DataRAM.sv
# Model Technology ModelSim ALTERA vlog 10.4b Compiler 2015.05 May 27 2015
# Start time: 20:44:06 on Nov 18,2016
# vlog -reportprogress 300 -work work C:/Users/chaos_000/Dropbox/CSE141L/cse141l/DataRAM.sv
# -- Compiling module DataRAM
#
# Top level modules:
#         DataRAM
# End time: 20:44:06 on Nov 18,2016, Elapsed time: 0:00:00
# Errors: 0, Warnings: 0
vlog -reportprogress 300 -work work C:/Users/chaos_000/Dropbox/CSE141L/cse141l/fetch.sv
# Model Technology ModelSim ALTERA vlog 10.4b Compiler 2015.05 May 27 2015
# Start time: 20:44:06 on Nov 18,2016
# vlog -reportprogress 300 -work work C:/Users/chaos_000/Dropbox/CSE141L/cse141l/fetch.sv
# -- Compiling module fetch
#
# Top level modules:
#         fetch
# End time: 20:44:06 on Nov 18,2016, Elapsed time: 0:00:00
# Errors: 0, Warnings: 0
vlog -reportprogress 300 -work work C:/Users/chaos_000/Dropbox/CSE141L/cse141l/Instruction_ROM.sv
# Model Technology ModelSim ALTERA vlog 10.4b Compiler 2015.05 May 27 2015
# Start time: 20:44:06 on Nov 18,2016
# vlog -reportprogress 300 -work work C:/Users/chaos_000/Dropbox/CSE141L/cse141l/Instruction_ROM.sv
# -- Compiling module Instruction_ROM
#
# Top level modules:
#         Instruction_ROM
# End time: 20:44:06 on Nov 18,2016, Elapsed time: 0:00:00
# Errors: 0, Warnings: 0
vlog -reportprogress 300 -work work C:/Users/chaos_000/Dropbox/CSE141L/cse141l/regfile.sv
# Model Technology ModelSim ALTERA vlog 10.4b Compiler 2015.05 May 27 2015
# Start time: 20:44:06 on Nov 18,2016
# vlog -reportprogress 300 -work work C:/Users/chaos_000/Dropbox/CSE141L/cse141l/regfile.sv
# -- Compiling module regfile
```

```
#
# Top level modules:
#           regfile
# End time: 20:44:06 on Nov 18,2016, Elapsed time: 0:00:00
# Errors: 0, Warnings: 0
vlog -reportprogress 300 -work work C:/Users/chaos_000/Dropbox/CSE141L/cse141l/top.sv
# Model Technology ModelSim ALTERA vlog 10.4b Compiler 2015.05 May 27 2015
# Start time: 20:44:06 on Nov 18,2016
# vlog -reportprogress 300 -work work C:/Users/chaos_000/Dropbox/CSE141L/cse141l/top.sv
# -- Compiling module top
#
# Top level modules:
#           top
# End time: 20:44:07 on Nov 18,2016, Elapsed time: 0:00:01
# Errors: 0, Warnings: 0
```

Q3: Lab 1 Assembly Code

```
; PRODUCT
        load $r3, $r3           ; R[3] = M[R[3]] ($r3 = A)
        load $r4, $r4           ; R[4] = M[R[4]] ($r4 = B)
        load $r5, $r5           ; R[5] = M[R[5]] ($r5 = C)


; MULTIPLY A AND B
while1:
        and $r4                 ; $overflow = B & 1
        bno if1                 ; branch if LSB of B is 0
        add $r1, $r3            ; add lower bits
        bno if1                 ; branch if no overflow
        incr $r0                ; increment upper bits
        add $r0, $r2            ; add upper bits


if1:
        lsl $r2                 ; shift upper bits of A left
        lsl $r3                 ; shift lower bits of A left
        bno if2                 ; skip if MSB lower bit is 0
        incr $r2                ; add overflow to upper bits of A


if2:
        lsr $r4                 ; shift B right
        eqz $r4                 ; overflow = (B == 0)
        bno while1                      ; loop while B != 0


# MULTIPLY AB AND C
        zero $r2                        ; initialize $r2 to 0
        zero $r3                        ; initialize $r3 to 0


while2:
        and $r5                 ; $overflow = C & 1
        bno if3                 ; branch if LSB of C is 0
        add $r3, $r1            ; add lower bits
        bno if3                 ; branch if no overflow
        incr $r2                ; increment upper bits
        add $r2, $r0            ; add upper bits


if3:
        lsl $r0                 ; shift upper bits of AB left
        lsl $r1                 ; shift lower bits of AB left
        bno if4                 ; skip if MSB lower bit was 0
        incr $r0                ; add overflow to upper bits of AB


if4:
        lsr $r5                 ; shift C right
        eqz $r5                 ; overflow = (C == 0)
        bno while2                      ; loop while C != 0
```

```
        store $r2, $r6          ; store upper bits in memory
        store $r3, $r7          ; store lower bits in memory
        halt

;STRING MATCH
        ;LOAD VALUES OF OPERANDS
        load $r1, $r1   ; load pattern from memory


        ;OUTER LOOP THROUGH ALL INPUT BYTE STRINGS
        s_while1:
        load $r0, $r6   ; load first byte string from memory


        ;INNER LOOP THROUGH EVERY 4 BIT COMBO IN EACH STRING
        s_while2:
        match $r0, $r1  ; check if lower 4 bits of string match pattern
        bof s_if1               ; if pattern match, branch
        lsr $r0         ; if no match, s >> 1
        incr $r5        ; increment inner loop index, j, by 1
        lt $r4, $r5     ; check inner while condition (j < length)
        bno s_if2               ; exit loop
        bof while2              ; if (j < length) is true, loop inner


        s_if1:
        incr $r2        ; increment count by 1


        s_if2:
        incr $r6        ; increment outer loop index, i, by 1
        lt $r6, $r7     ; check outer while condition (i < size)
        bof s_while1            ; if (i < size) is true, loop outer

        store $r2, $r3  ; store the count to memory
        halt

; CLOSEST PAIR
        ; OUTER LOOP THROUGH EVERY VALUE IN ARRAY TO COMPARE
        c_while1:
        load $r0, $r2   ; x = array[i]
        incr $r2        ; i++
        zero $r4                ; j = 0
        add $r4, $r2    ; j += i (so that j = i)


        ;INNER LOOP THROUGH EVERY OTHER VALUE IN ARRAY TO COMPARE TO
        c_while2:
        load $r1, $r4   ; y = array[j]
        incr $r4        ; j++
        dist $r1, $r0   ; calculate dist, overwrite y
        lt $r1, $r6     ; check if dist < min
        bno c_if1               ; if dist >= min, skip min update
        zero $r6                ; min = 0
        add $r6, $r1    ; min += dist
```

```
c_if1:
lt $r4, $r5        ; check if j < 20
bof c_while2               ; if j < 20 is true, then loop inner


lt $r2, $r3        ; check if i < 19
bof c_while1               ; if i < 19 is true, then loop outer


store $r6, $r7    ; store min in result address
halt
```

Q4: Machine code output

```
000011011
000100100
000101101
110011100
111000101
010001011
111000011
110010000
010000010
110000010
110000011
111000010
110010010
110001100
110100100
111010100
110101010
110101011
110011101
111000101
010011001
111000011
110010010
010010000
110000000
110000001
111000010
110010000
110001101
110100101
111010100
001010110
001011111
110111000
000001001
000000110
011000001
111100110
110001000
110010101
100100101
```

111000011
111101000
110010010
110010110
100110111
111110101
001010011
110111000
000000010
110010010
110101100
010100010
000001100
110010100
101001000
100001110
111000011
110101110
010110001
100100101
111111000
100010011
111110010
001110111
110111000