REQUIREMENTS NOT MET

N/A. All Requirements are met in this lab.

PROBLEMS ENCOUNTERED

Some problems encountered with part 1 of the lab includes the hardware expansion portion where it was not remotely reviewed in class and had to be researched heavily from the manuals and digital logic resources to be completed. Many of the statements were generalization of the tasks without specifics on how it ought to be done. This portion took most of the time spent on lab. The problems encountered with part 2 was with wiring the hardware as the drawn designed linked the concepts of the design but not the actual necessary building parts. For instance, there was no tri-state buffer and the enables on the chips were generalized, there are several enables on the chips, making the process much more confusing. For part 3, on major issued encountered was reading from the SRAM as I could easily write data to it but unable to retrieve it to display in the LED. After trying varies ways and researching a lot, the task was completed.

FUTURE WORK/APPLICATIONS

This lab was a good introduction into the implementation and function of external bus interface, EBI. This lab is to be the expansion of more complex assembly programs, able to give the users' another way to code, enabling the running of separate functions from the main program. With the EBI commands users are no longer restricted to the hardware only on their uPad but are now able to add external I/O hardware like LEDs and switches if needed to expand their capability through hardware. A fundamental change is the addition of external memory which allows for larger and more intensive programs to be build including though of circuit designs in Quartus, making the possibilities limitless. Like the subroutine, the way I program is now changed to be inclusive of interrupts for more capability of my programs. If given more time, the code of the lab could have been more organized and have a much neater layout to further reduce the likelihood of mistakes and further enhance the understanding of the program. With more time a more compacted or efficient memory map would have been implemented saving space in the SRAM. Additionally, I could have used better instructions to make the code run more efficiently or learn to write more complex programs.

PRE-LAB EXERCISES

Part 1: introduction to EBI and Hardware Expansion

i. For each SRAM configuration within the EBI system of the ATxmega128A1U microcontroller, which address lines do you have external, physical access to? Additionally, for the SRAM ALE1 configuration, is it possible to have external, physical access to any address lines above A15? Why or why not?

No multiplexing SRAM configuration within the EBI system of the ATxmega128A1U microcontroller has a one to one connection, none of the address lines that are external, there is no physical access available. For the multiplexing address byte 0 and 1, ALE1 is used thus granted access control to A[7:0] and A[15:8] depending on the ALE1 setting. For the multiplexing address byte 0 and 2, ALE2 is used thus granted access control to A[7:0] and A[23:16] depending on the ALE2 setting. For the multiplexing address byte 0, 1, and 2, ALE1 and ALE2 is used thus granted access control to A[7:0] and A[15:8] depending on the ALE1 setting and A[7:0] and A[23:16] depending on the ALE2 setting. For the SRAM ALE1 configuration, is it not possible to have external, physical access to any address lines above A15 because the ALE1 only controls signal access to address byte 0 (A[7:0]) and 1 (A[15:8]), not having any control over byte 2 (A[23:16]) which would require ALE2.

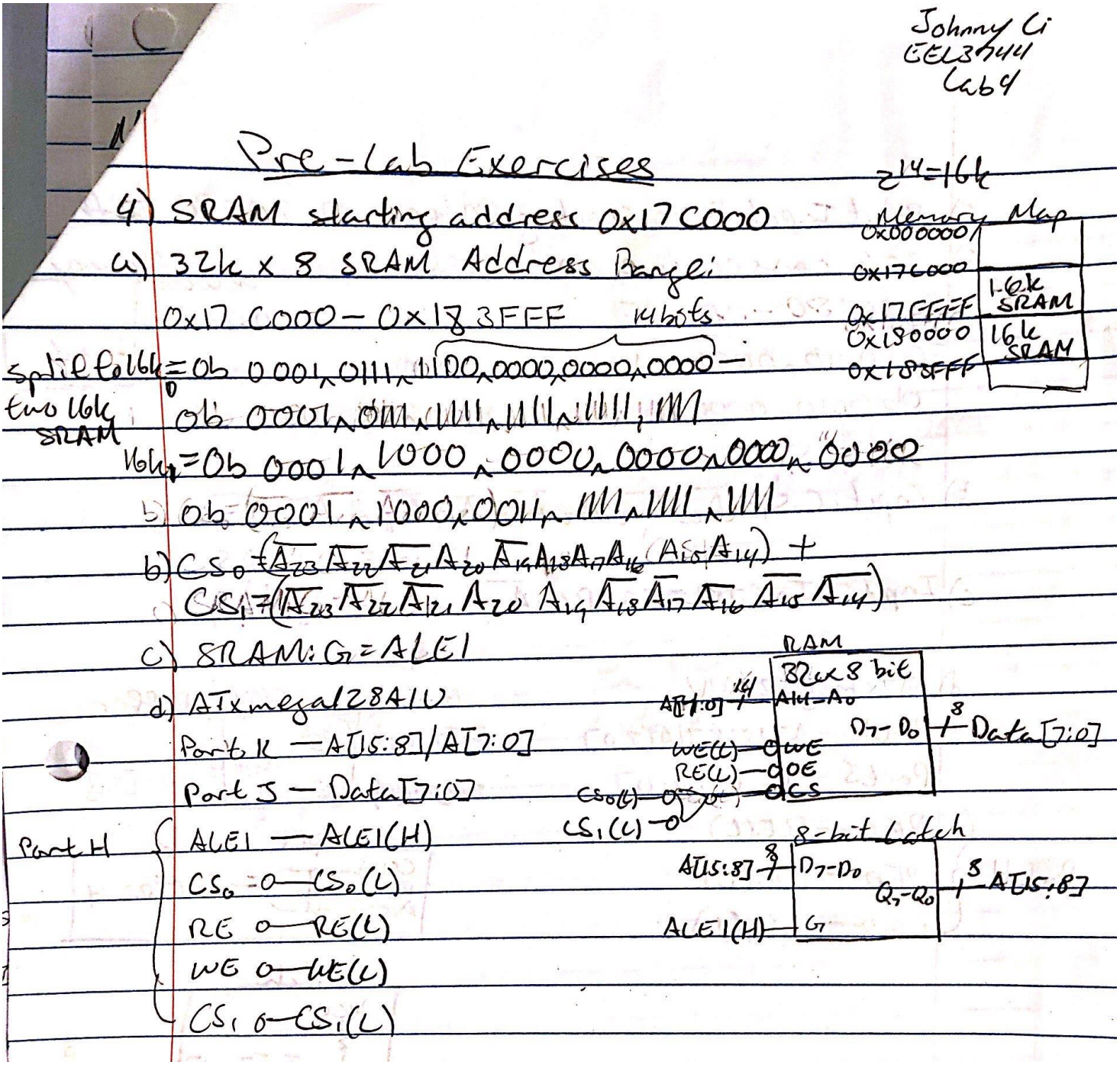
ii. Describe what performing full address decoding and partial address decoding signifies. Provide examples of both types for an SRAM chip, as well as for I/O ports.

Performing full address decoding means that all the address lines are used to specify a memory location and each physical memory location is identified by a unique address while performing partial address decoding means that not all the address space is implemented, only a subset of the address lines are needed to point to the physical memory locations and each physical memory location is identified by several possible addresses where using all combinations of the address lines that were not used, set as don't cares.
Example of full address decoding in SRAM would be a 1K SRAM with all its memory space being full and the last two bytes to be used for decoding. An example of the partial used would be to have 1K SRAM with the last second to last byte being a don't care and the last byte used for decoding. The I/O port, an example of full address decoding would be having a port set over the external memory of 1K address range to constructed about 1K 8-bit I/O ports, one port for an address line. For a partial address decoding of I/O port, having a port set over the external memory address range but able constructed about some 8-bit I/O ports with some addresses and with others addresses returning into a 8-bit I/O port with an address already or to none.

iii. In theory, how many 8-bit I/O ports could be constructed within the external memory space of an ATxmega128A1U microcontroller, assuming that at most one port utilized any chosen address range? Explain your answer, utilizing your response from Exercise ii.

In full address decoding, an address line can be setup to utilize an 8-bit I/O port. With at most one port utilized any chosen address range, the amount of 8-bit I/O ports could be constructed within the external memory space of an ATxmega128A1U microcontroller is equivalent number of address line in the address range. Therefore, the address space for the external memory is selectable from 256 bytes up to 16MB there can be an equivalent number of 8-bit I/O port from about 256 up to 16M under full address decoding, all assuming there are enough external devices to support the setup.

iv. Assume that a SRAM component with the same size as the one on the OOTB Memory Base has to be added to a different computer system containing an ATxmega128A1U but no OOTB Memory Base, with the first address of the SRAM starting at address 0x17 C000, instead of the address specified in Appendix A. Design a hardware expansion for this system, utilizing the same structure of steps laid out in Appendix A.



v. Assume that some 8-bit input port should be accessible via 256 consecutive addresses starting at 0x20580 within the ATxmega128A1U data memory space. Design a hardware expansion for this port, utilizing the same structure of steps laid out in Appendix A.

a) 8-bit Input Port starting address 0x20580 with
a) 256 consecutive addresses.

0x20580 – 0x2067F

= 0b 0010_0000_0101_1000_0000 — 0b

0b 0010_0000_0110_0111_1111

= 0b 0010_0000_0111_1111

Memory Map

0x000000	
0x20580	Input
0x2067F	Input
0x207FF	

b) Input: $CS_1 = (\overline{A_{23}}\,\overline{A_{22}}\,\overline{A_{21}}\,\overline{A_{20}}\,\overline{A_{19}}\,\overline{A_{18}}\,A_{17}\,\overline{A_{16}}\,\overline{A_{15}}\,\overline{A_{14}}\,\overline{A_{13}}\,\overline{A_{12}}\,\overline{A_{11}})$

c) Input: $InCTRL = CS_1 \cdot RE \cdot \overline{A_0}$

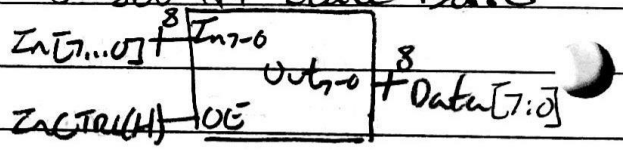
d) ATxmega128A1U

PortK – A[15:8]/A[7:0]

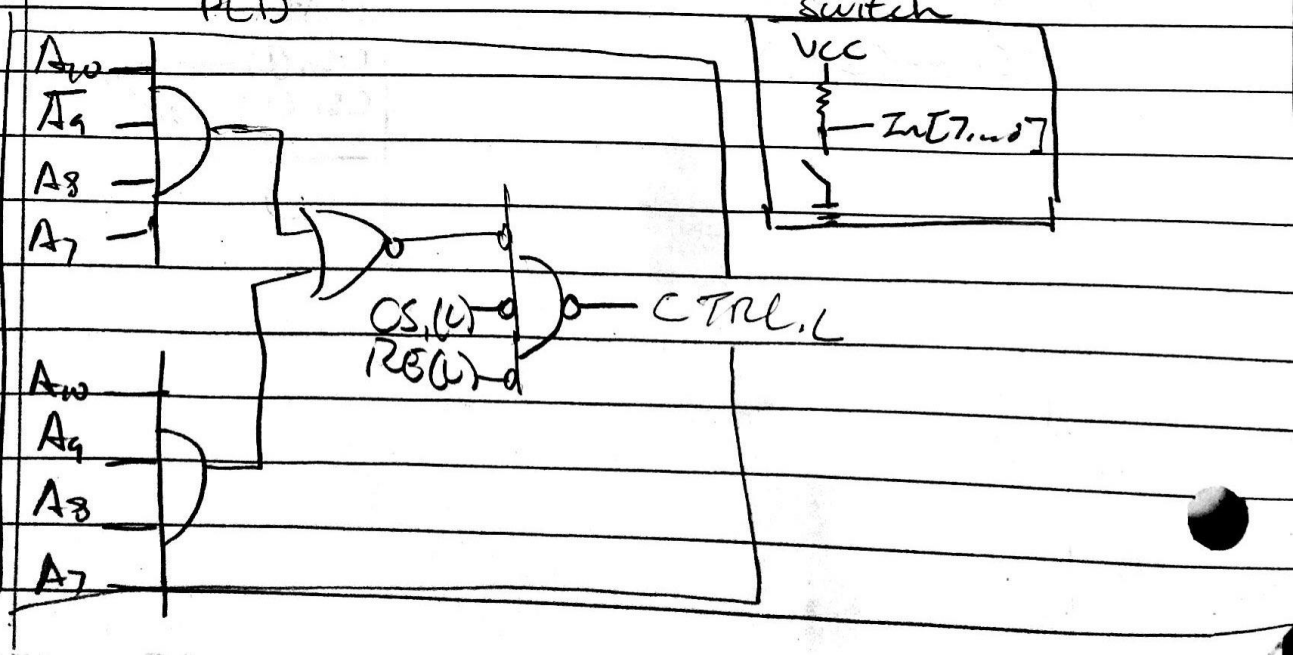
PortJ – Data[7:0]

PortH
- RE0 – RE(L)
- WE0 – WE(L)
- CS_10 – CS_1(L)

8-bit Tri-State Buffer



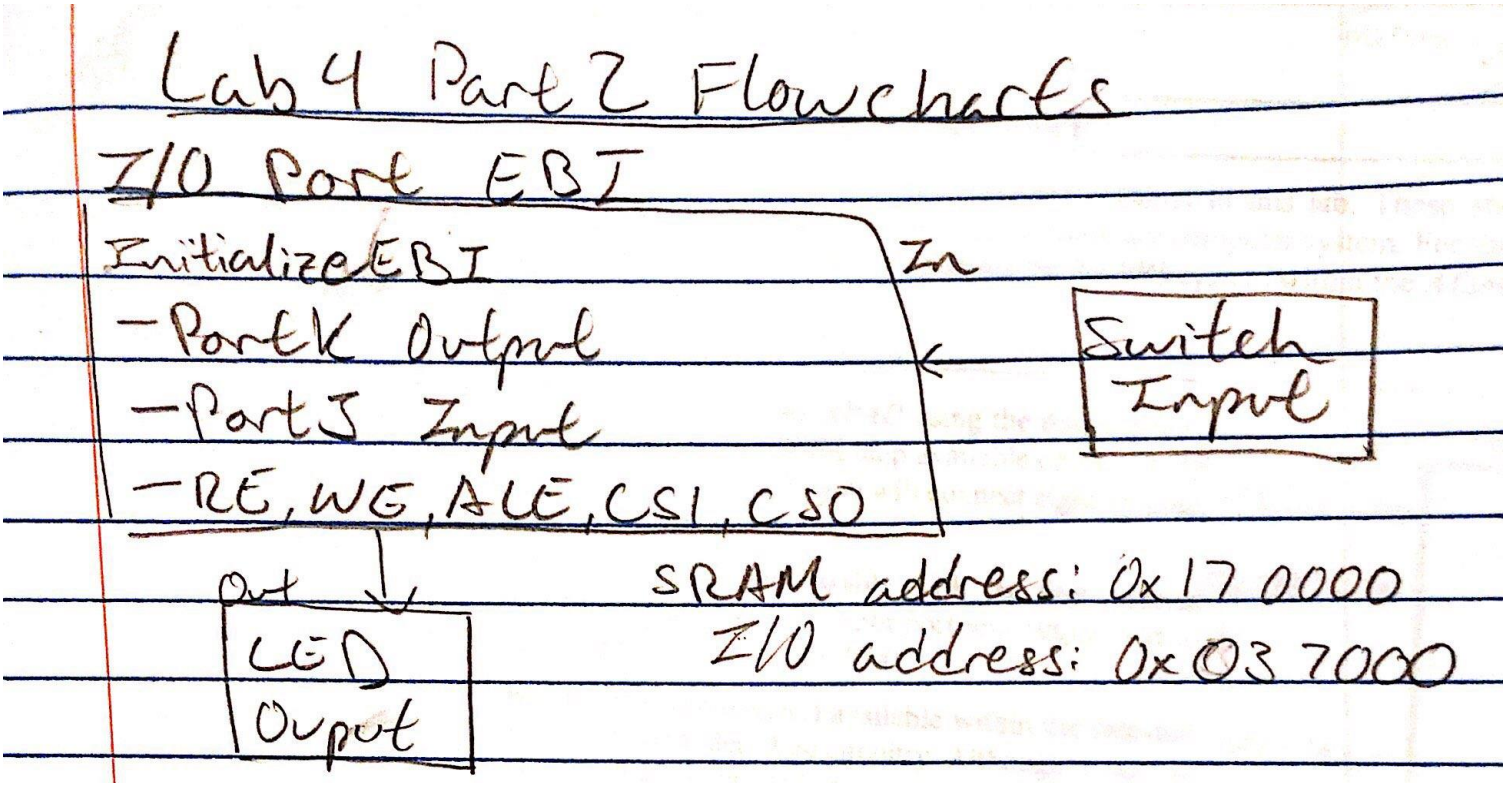
PLD



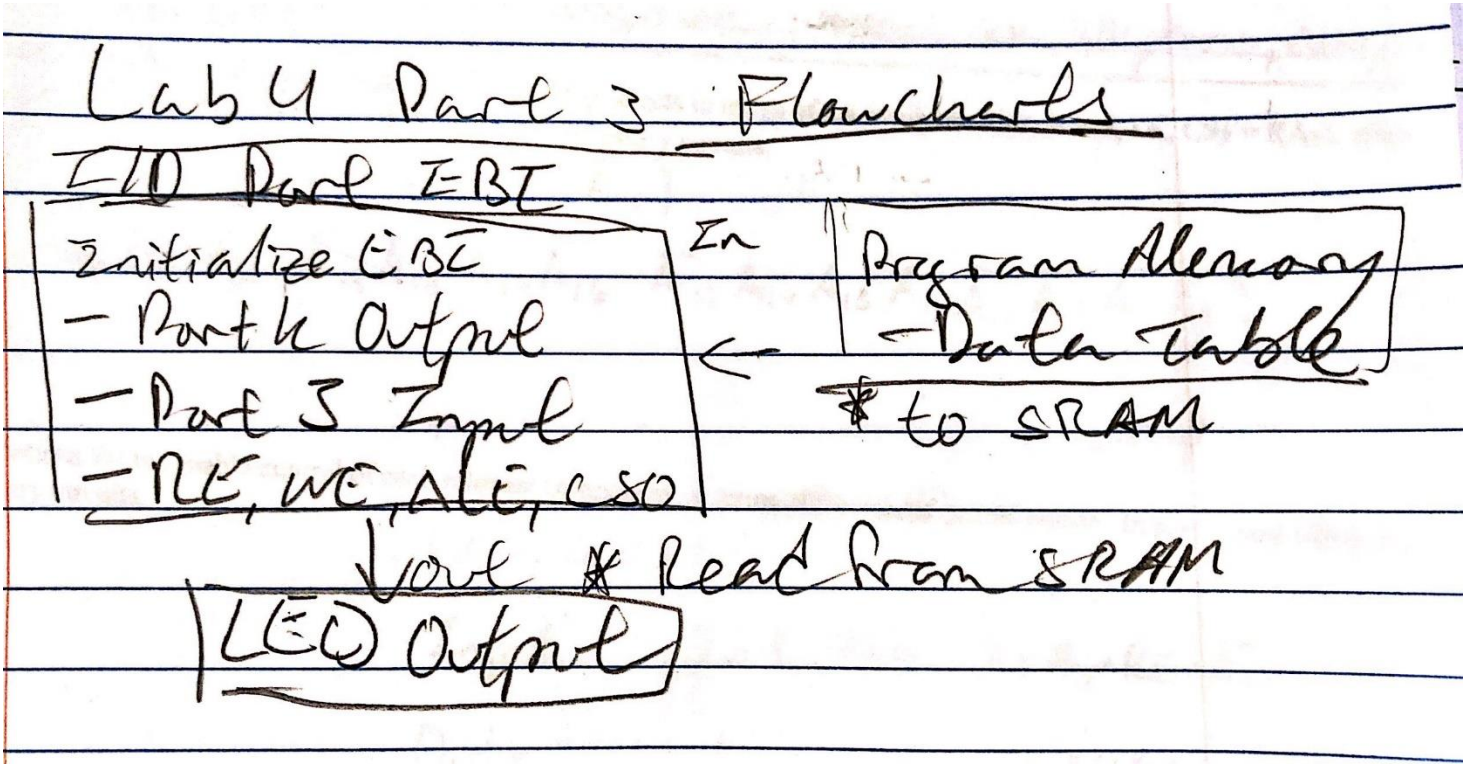
PSEUDOCODE/FLOWCHARTS

SECTION X (1, 2, etc.)

Part 2: Interfacing with External I/O Ports



Part 3: Interfacing with External SRAM



PROGRAM CODE

SECTION X (1, 2, etc.)

Part 2: Interfacing with External I/O Ports

```
;*********************************MAIN PROGRAM***********************************
       .cseg
.org 0x0000
       rjmp MAIN

.org 0x100
MAIN:
       ldi r16, 0b00110011         ; Since RE(L), WE(L), CS0(L) and CS1(L) are active low signals, we
must set
       sts PORTH_OUTSET, r16       ; the default output to 1 = H = false. See 8331, sec 27.9.
                                   ; (ALE defaults to 0 = L = false)

       ldi r16, 0b00110111         ; Configure the PORTH bits 4, 2 and 1 as outputs.
       sts PORTH_DIRSET, r16       ; These are the CS1(L), CS0(L), ALE1(H), WE(L) and RE(L) outputs.
                                   ; (CS0 is bit 4; ALE1 is bit 2; RE is bit 1)
                                   ; see 8385, Table 33-7

       ldi r16, 0xFF               ; Set all PORTK pins (A15-A0) to be outputs. As requried
       sts PORTK_DIRSET, r16       ; in the data sheet. See 8331, sec 27.9.

       ldi r16, 0xFF               ; Set all PORTJ pins (D7-D0) to be outputs. As requried
       sts PORTJ_DIRSET, r16       ; in the data sheet. See 8331, sec 27.9.

       ldi r16, 0x01               ; Store 0x01 in EBI_CTRL register to select 3 port EBI(H,J,K)
       sts EBI_CTRL, r16           ; mode and SRAM ALE1 mode.

;Reserve a chip-select zone for our input port. The base address register is made up of
;  12 bits for the address (A23:A12). The lower 12 bits of the address (A11-A0) are
;  assumed to be zero. This limits our choice of the base addresses.

;Load the middle byte (A15:8) of the three byte address into a register and store it as the
;  LOW Byte of the Base Address, BASEADDRL.  This will store only bits A15:A12 and ignore
;  anything in A11:8 as again, they are assumed to be zero.
       ldi r16, byte2(IO)
       sts EBI_CS1_BASEADDR, r16

;Load the highest byte (A23:16) of the three byte address into a register and store it as the
;  HIGH byte of the Base Address, BASEADDRH.
       ldi r16, byte3(IO)
       sts EBI_CS1_BASEADDR+1, r16

; Set to 32K chip select space and turn on SRAM mode.
       ldi r16, 0x01 ;256
       sts EBI_CS1_CTRLA, r16

       ldi r16, byte3(IO)          ; initalize a pointer to point to the base address of the SRAM
       sts CPU_RAMPX, r16                ; use the CPU_RAMPX register to set the third byte of the
pointer

       ldi XL, low(IO)
       ldi XH, high(IO)            ; set the middle (XH) and low (XL) bytes of the pointer as usual

TEST:
       ld r16, X                                          ; read the input port into r16
```

```
        nop
        nop
        st X, r16
        rjmp TEST                                               ; put a breakpoint on me and check r16!

        ;End of program (never reached)
DONE:
        rjmp DONE
```

Part 3: Interfacing with External SRAM

```
;*********************************MAIN PROGRAM***********************************
        .cseg
.org 0x0000
        rjmp MAIN

.org 0x100
MAIN:
        ldi r16, 0b00110011         ; Since RE(L), WE(L), CS0(L) and CS1(L) are active low signals, we
must set
        sts PORTH_OUTSET, r16       ; the default output to 1 = H = false. See 8331, sec 27.9.
                                    ; (ALE defaults to 0 = L = false)

        ldi r16, 0b00110111         ; Configure the PORTH bits 4, 2 and 1 as outputs.
        sts PORTH_DIRSET, r16       ; These are the CS1(L), CS0(L), ALE1(H), WE(L) and RE(L) outputs.
                                    ; (CS0 is bit 4; ALE1 is bit 2; RE is bit 1)
                                    ; see 8385, Table 33-7

        ldi r16, 0xFF               ; Set all PORTK pins (A15-A0) to be outputs. As requried
        sts PORTK_DIRSET, r16       ; in the data sheet. See 8331, sec 27.9.

                                    ; Set all PORTJ pins (D7-D0) to be outputs. As requried
        sts PORTJ_DIRSET, r16       ; in the data sheet. See 8331, sec 27.9.

        ldi r16, 0x01               ; Store 0x01 in EBI_CTRL register to select 3 port EBI(H,J,K)
        sts EBI_CTRL, r16           ; mode and SRAM ALE1 mode.

;Reserve a chip-select zone for our input port. The base address register is made up of
;  12 bits for the address (A23:A12). The lower 12 bits of the address (A11-A0) are
;  assumed to be zero. This limits our choice of the base addresses.

;ENABLE SRAM-------------------------------------------------------------------------
; Set to size chip select space and turn on SRAM mode.
        ldi r16, 0b00011101  ;32k
        sts EBI_CS0_CTRLA, r16

;ENABLE IO---------------------------------------------------------------------------------
        ; Set to size chip select space and turn on SRAM mode.
        ldi r16, 0x01 ;256
        sts EBI_CS1_CTRLA, r16

        ; point appropriate indices to input/output tables
        ;Z points to input table.
        ldi ZL, low(DATA << 1) ; Shift due to being 16-bits in a 8-bit processor
        ldi ZH, high(DATA << 1) ; Z points to the table
        ldi r16, BYTE3(DATA << 1)
    sts CPU_RAMPZ,r16

        ;Y points to input table.
        ldi YL, low(IO) ; Shift due to being 16-bits in a 8-bit processor
```

```
       ldi YH, high(IO) ; Z points to the table
       ldi r16, BYTE3(IO)
    sts CPU_RAMPY,r16

       ;X points to input table.
       ldi XL, low(SRAM) ; Shift due to being 16-bits in a 8-bit processor
       ldi XH, high(SRAM) ; Z points to the table
       ldi r16, BYTE3(SRAM)
    sts CPU_RAMPX,r16
;Initialize----------------------------------------------------------------
;Load the middle byte (A15:8) of the three byte address into a register and store it as the
;  LOW Byte of the Base Address, BASEADDRL.  This will store only bits A15:A12 and ignore
;  anything in A11:8 as again, they are assumed to be zero.
       ldi r16, byte2(SRAM)
       sts EBI_CS0_BASEADDR, r16

;Load the highest byte (A23:16) of the three byte address into a register and store it as the
;  HIGH byte of the Base Address, BASEADDRH.
       ldi r16, byte3(SRAM)
       sts EBI_CS0_BASEADDR+1, r16

;Load the middle byte (A15:8) of the three byte address into a register and store it as the
;  LOW Byte of the Base Address, BASEADDRL.  This will store only bits A15:A12 and ignore
;  anything in A11:8 as again, they are assumed to be zero.
       ldi r16, byte2(IO)
       sts EBI_CS1_BASEADDR, r16

;Load the highest byte (A23:16) of the three byte address into a register and store it as the
;  HIGH byte of the Base Address, BASEADDRH.
       ldi r16, byte3(IO)
       sts EBI_CS1_BASEADDR+1, r16


       ; loop through input table
LOOP:
       ; load value from input table into an appropriate register
       elpm r16,Z+
       ;Store in SRAM
       st X+, r16
       ; determine if the end of table has been reached (perform general check)
       cpi r16, 0xFF ;compare NULL and r16 to see if NULL is reached

       ; if end of table (EOT) has not been reached,
       BRNE Loop

       ;Reset X points to SRAM
       ldi XL, low(SRAM)
       ldi XH, high(SRAM)
       ldi r16, BYTE3(SRAM)
    sts CPU_RAMPX,r16
READ:
       //READ to LED
       ld r16,x+
       st Y,r16

       ;Initialize Counter
       ldi r22, 0b0111 ;Prescaler 512
       sts TCC0_CTRLA, r22 ;Sets the Clock Select from Register A

       ;Load high and low byte of Period
       ldi r22, 0xA2 ;Load r16 with 0x62
       sts TCC0_PER, r22 ;Store low byte first
```

```
	ldi r22, 07;Load r16 with 01
	sts TCC0_PER + 1, r22 ;Store high byte

	;Check Overflow
	FLAG1:
	lds r20, TCC0_INTFLAGS ;Loads the Flag into r20
	sbrs r20, 0 ;Compares r20 with 0
	rjmp FLAG1 ;rjmp to FLAG1 until SET

	sts TCC0_INTFLAGS, r20 ;Store this value into the INTFLAGS Register
	cpi r16, 0 ;Compare r16 with 0
	brne READ

	;Branches to repeat if not equal to zero
	rjmp LOOP ;Branches to loop if it is 0

	;End of program (never reached)
DONE:
	rjmp DONE
;*****************************END OF MAIN PROGRAM ****************************
;*****************************END OF "lab4_2.asm"*****************************
```

APPENDIX

Part 1: introduction to EBI and Hardware Expansion

Johnny Li

APPENDIXES

A. EBI HARDWARE EXPANSION

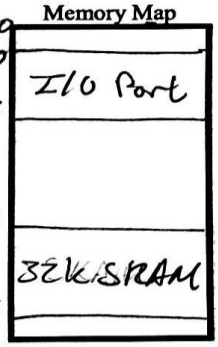
The following is an accompanying template of steps to create the hardware expansion required in this lab. These steps are not a generalization of all hardware expansions, although similar steps could be performed for most any computer system. For the purposes of this lab, pay close attention to the design of the *µPAD*, and to the limitations imposed by the EBI system within the *ATxmega128A1U*. Refer to any appropriate schematics and datasheets, if necessary.

Below, you will design a memory-mapped hardware expansion for the *µPAD* using the data memory space within the *ATxmega128A1U*. The expansion will contain the SRAM chip available on the *OOTB Memory Base*, an 8-bit input port, and an 8-bit output port. Additionally, you will connect eight external switches to the input port, and eight external LEDs to the output port.

Using full address decoding, design the SRAM to be fully addressable with its first address corresponding to 0x17 0000. Using partial address decoding, design the input port and output port to each be accessible only via the 128 consecutive addresses starting at 0x03 7000.

Whenever appropriate, perform address decoding with any chip select(s) available within the relevant EBI system, or in other words, minimize the amount of external decoding circuitry. Although it may be extraneous information, recall that only one chip select can be enabled at any given time. Whenever external decoding circuitry is necessary, construct it with a PLD.

Memory Map



a) Complete the table below, identifying appropriate memory address ranges for each memory-mapped component within the hardware expansion. If helpful, you may also use the memory map figure to the right.

32k x 8 SRAM Address Range:
0x17 0000 - 0x177FFF = 0b0001 0111 0000 0000 0000 0000 - 0b0001 0111 1111 1111 1111 1111

8-bit Input Port Address Range:
0x03 7000 - 0x03 707F = 0b0000 0011 0111 0000 0000 0000 - 0b0000 0011 0111 0000 0111 1111

8-bit Output Port Address Range:
0x03 7000 - 0x03 707F = 0b0000 0011 0111 0000 0000 0000 - 0b0000 0011 0111 0000 0111 1111

b) Derive address decoding functions for any appropriate chip selects in terms of the available address bits, i.e., $CS_X = f(A_Y)$, where $0 \le X \le 3$ and $0 \le Y \le 23$. In part d, you will design the necessary circuits.

SRAM: $CS_0 = (\overline{A_{23}}\ \overline{A_{22}}\ \overline{A_{21}}\ A_{20} + \overline{A_{19}}\ A_{18}\ A_{17}\ A_{16}\ \overline{A_{15}})$

I/O: $CS_1 = (\overline{A_{23}}\ \overline{A_{22}}\ \overline{A_{21}}\ \overline{A_{20}}\ \overline{A_{19}}\ \overline{A_{18}}\ A_{17}\ A_{16}\ \overline{A_{15}}\ A_{14}\ A_{13}\ A_{12}\ \overline{A_{11}}\ \overline{A_{10}}\ \overline{A_9}\ \overline{A_8})$

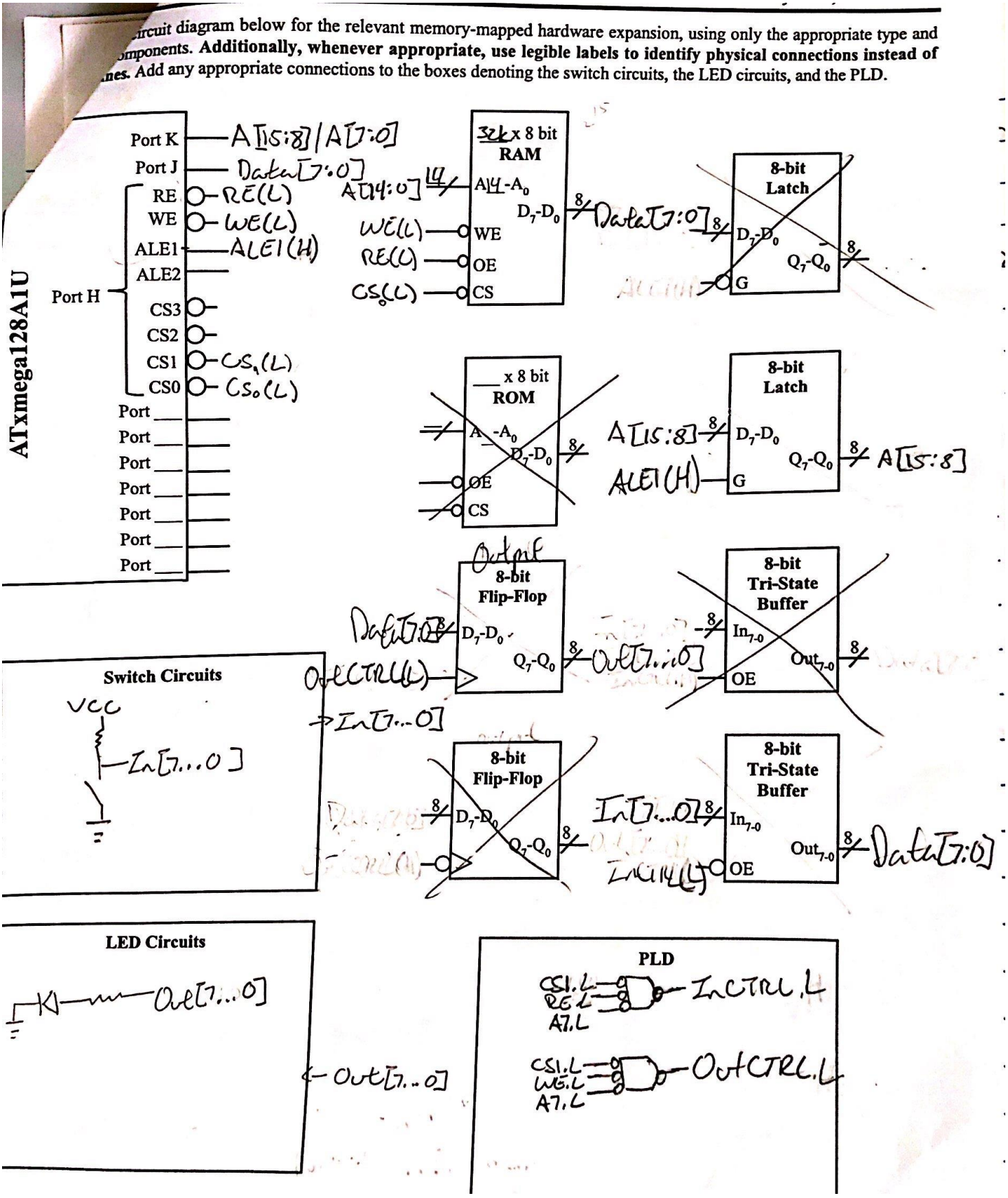
c) Derive functions for the enable/control of each relevant component, in terms of the available EBI signals. In part d, you will design the necessary circuits.

SRAM: $G = ALE1$

Input: $InCTRL = CS_1 \cdot RE \cdot \overline{A_7}$

Output: $OutCTRL = CS_1 \cdot WE \cdot \overline{A_7}$

$OutCTRL = \overline{InCTRL}$

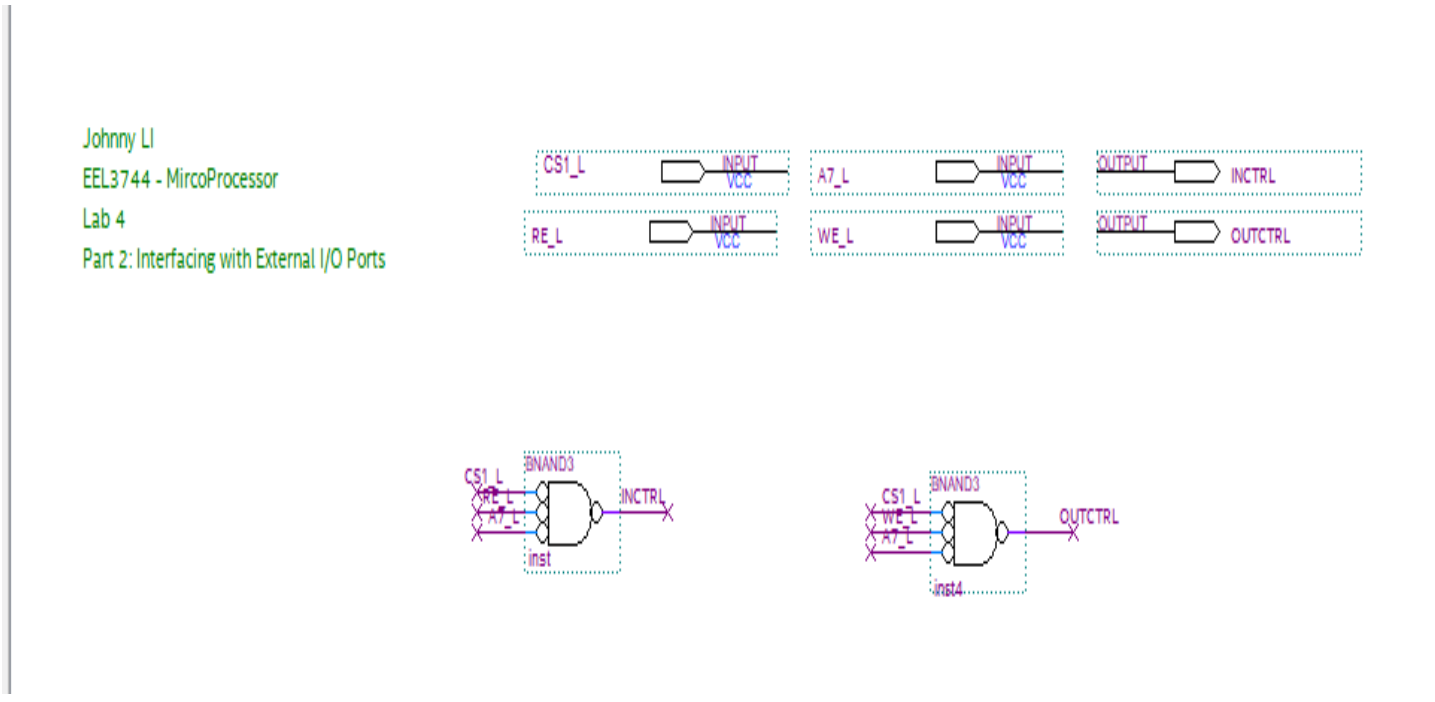


Images 1: EBI Hardware Expansion

Part 2: Interfacing with External I/O Ports

```
1   ;*****************************************************************************
2   ;Lab 4 Part 2
3   ;Section #: 1823
4   ;Name: Johnny Li
5   ;Class #: 12378
6   ;PI Name: Jared Holley
7   ;Description: Interfacing with External I/O Ports
8   ;***********************************INCLUDES***********************************
9   .include "ATxmega128a1udef.inc"
10  ;*******************************END OF INCLUDES*********************************
11  ;*******************************DEFINED SYMBOLS*********************************
12  .equ SRAM = 0x170000
13  .equ IO = 0x037000
14  ;***************************END OF DEFINED SYMBOLS*****************************
15  ;*******************************MEMORY CONSTANTS********************************
16      ; data memory allocation
17      /*.dseg
18      .org DATA_ADDR      ;For table
19  DATA:
20      .byte DATA_END      ; TABLE SIZE*/
21  ;***************************END OF MEMORY CONSTANTS*****************************
```

Screenshot 1: Part 2-I/O Memory Configuration

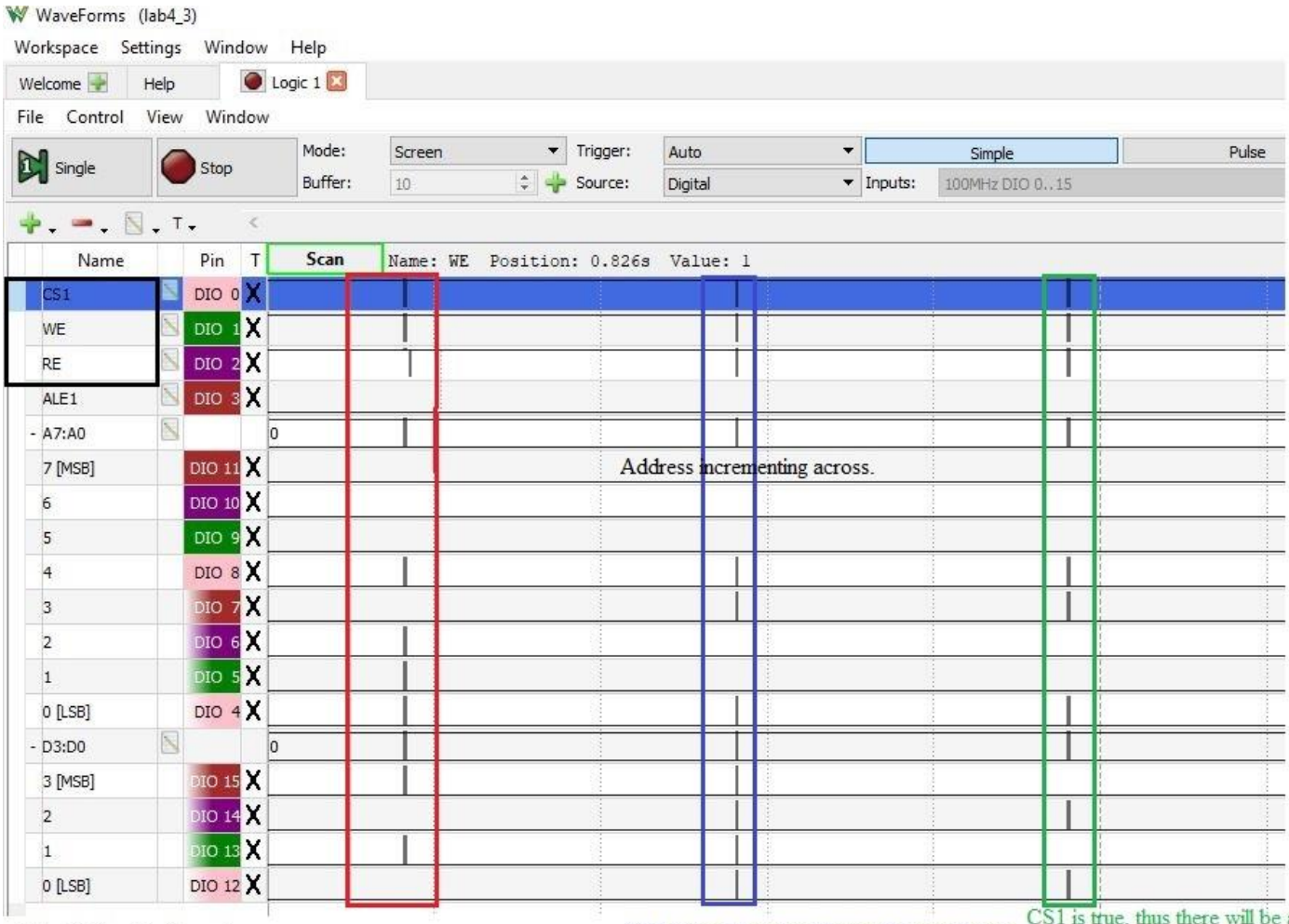


Screenshot 2: Quartus Diagram

Part 3: Interfacing with External SRAM

```
;*****************************************************************************
;Lab 4 Part 3
;Section #: 1823
;Name: Johnny Li
;Class #: 12378
;PI Name: Jared Holley
;Description: INTERFACING WITH EXTERNAL SRAM
;***********************************INCLUDES***********************************
.include "ATxmega128a1udef.inc"
;*******************************END OF INCLUDES********************************
;*******************************DEFINED SYMBOLS********************************
.equ SRAM = 0x170000    //SRAM table starting address
.equ IO = 0x037000  //IO table starting address
.equ IN_TABLE_START_ADDR = 0x3000   //Input table starting address
;***************************END OF DEFINED SYMBOLS*****************************
;*******************************MEMORY CONSTANTS*******************************
    ;Given table content
    .org IN_TABLE_START_ADDR
DATA:
.db 0x4d, 0x61, 0x6b, 0x69, 0x6e, 0x67, 0x20, 0x79, 0x6f, 0x75
.db 0x72, 0x20, 0x77, 0x61, 0x79, 0x20, 0x69, 0x6e, 0x20, 0x74
.db 0x68, 0x65, 0x20, 0x77, 0x6f, 0x72, 0x6c, 0x64, 0x20, 0x74
.db 0x6f, 0x64, 0x61, 0x79, 0x20, 0x74, 0x61, 0x6b, 0x65, 0x73
.db 0x20, 0x65, 0x76, 0x65, 0x72, 0x79, 0x74, 0x68, 0x69, 0x6e
.db 0x67, 0x20, 0x79, 0x6f, 0x75, 0x27, 0x76, 0x65, 0x20, 0x67
.db 0x6f, 0x74, 0x2e, 0x0a, 0x54, 0x61, 0x6b, 0x69, 0x6e, 0x67
.db 0x20, 0x61, 0x20, 0x62, 0x72, 0x65, 0x61, 0x6b, 0x20, 0x66
.db 0x72, 0x6f, 0x6d, 0x20, 0x61, 0x6c, 0x6c, 0x20, 0x79, 0x6f
```

Screenshot 2: Part 3-SRAM Memory Configuration



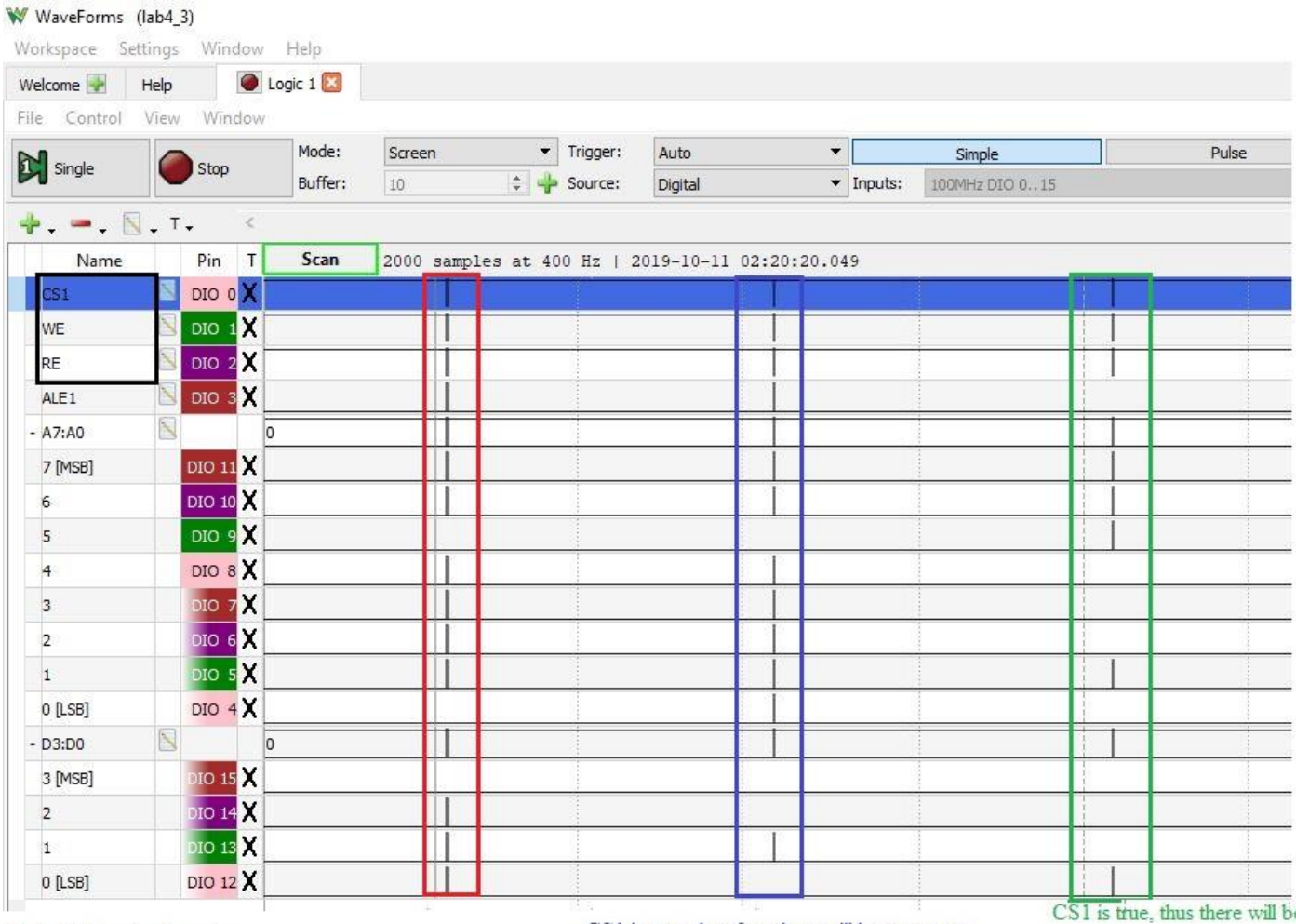
Note: Only active low pins.

Chip select 1 is true, therefore there will be an output to the LED.
ALE1 not used in this case.
Write to SRAM then read from SRAM enabled.
The selected addresses are chosen to be used where A7 is to decode the control lines.
Data of the first 4 bits =A to be stored to SRAM and then displayed in LED.

CS1 is true, therefore there will be an output to the LED.
ALE1 not used in this case.
Write to SRAM then read from SRAM enabled.
The selected addresses are chosen to be used where A7 is to decode the control lines.
Data of the first 4 bits = F to be stored to SRAM and then displayed in LED.

CS1 is true, thus there will be an LED output.
ALE1 not used in this case.
Write to SRAM then read from SRAM enabled.
The selected addresses are chosen to be used where A7 is to decode the control lines.
Data of the first 4 bits = 5 to be stored to SRAM and then displayed in LED.

Image 2: Waveform 1



Note: Only active low pins.

Chip select 1 is true, therefore there will be an output to the LED.
ALE1 is used in this case.
Write to SRAM then read from SRAM enabled.
The selected addresses are chosen to be used where it is now using A[15:8] rather than A[7:0].
Data of the first 4 bits = 7 to be stored to SRAM and then displayed in LED.

CS1 is true, therefore there will be an output to the LED.
ALE1 is used in this case.
Write to SRAM then read from SRAM enabled.
The selected addresses are chosen to be used where A[15:8] is used.
Data of the first 4 bits = 2 to be stored to SRAM and then displayed in LED.

CS1 is true, thus there will be an LED output.
ALE1 not used in this case.
Write to SRAM then read from SRAM enabled.
The selected addresses are chosen to be used where A7 is to decode the control lines.
Data of the first 4 bits = 1 to be stored to SRAM and then displayed in LED.

Image 3: Waveform 2