Aviles, Jean-Ralph
EEL3744
Section 1539
September 22nd, 2015
Lab 3

Prelab Questions

1. What is the advantage of mirroring the input and output ports to $8000 - $9FFF (also known as partial address decoding)? What would be necessary if you wanted to only place the ports at 0x8000 and no place else?

   The advantage of using partial address decoding is that it makes our design less complicated. Instead of having to specify an exact address to decode we specify a range which makes our logic easier. If we wanted to place the ports at address 0x8000 and nowhere else it our decoding logic would need to use more then just CS0.

2. Write the address decode equation to put the input and output ports at addresses 0x1000 - 0x7FFF.

   I believe this is impossible because 0x1000 is at an address reserved for EEPROM and base addresses for Chip Selects must lie on a 4K boundary but ignoring that we would simply need to use the address lines that the CPLD has access to. Since 0x1000 starts with 0b0001 and 0x7FFF starts with 0b0111 we would need to add $\overline{A_{15}}$ to the logic selecting equation for the input and output ports in our schematics.

3. What is the complete range of external memory on the XMEGA?

   According to the manual, the range of external memory addresses we have access to is from 0x3000 - 0xFFFFFF.

4. The uPAD Proto Base is configured to use the EBI in SRAM ALE1 mode, which does not give us address bits A23:16. Which mode(s) can be used to access these higher address bits? Describe the change(s) necessary to use one of these modes. What additional hardware is needed, if any? If we set the EBI to SRAM ALE2 or ALE12 modes we would have access to those higher order bits. If we swapped over to the new mode we would need a new board to access all the new address bits we now have since we only have address to $A_{12} - A_{15}$.

Problems Encountered

It took me a bit to understand what was begin asked for the address decoding section. I think the lab was worded a bit weird. The decoding logic also was difficult to get right, also OSX El Capitan broke my Virtual Box.

Future Work/Applications

We can now memory map any device we want to the board and multiplex them to conserve the number of pins we have on the board.

Appendix

Input/Output Ports

Start address

0x8000 so EBI_CS0_BASEADDR = 0x8000.

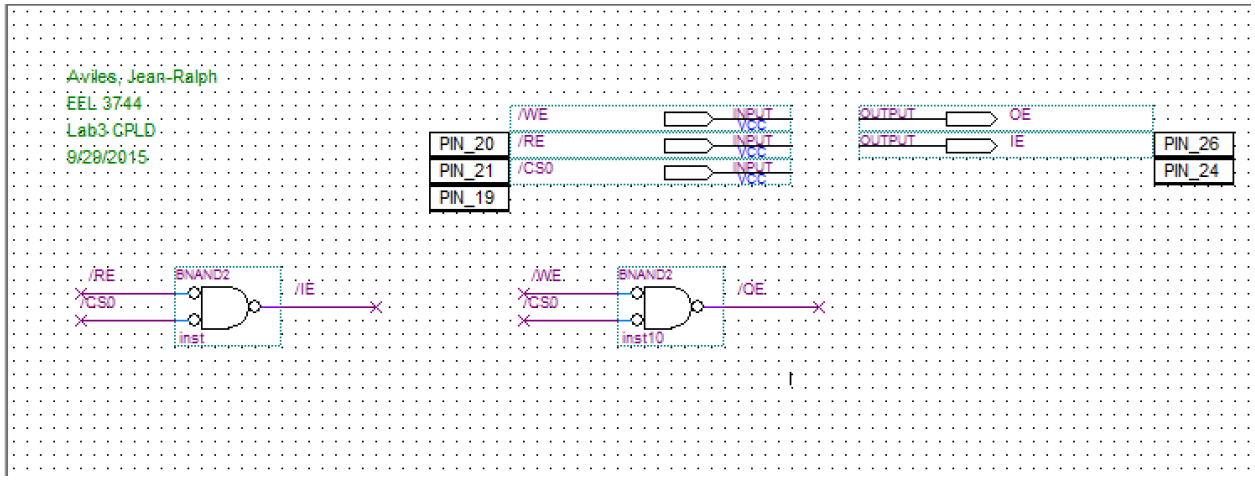
End address

0x9FFF

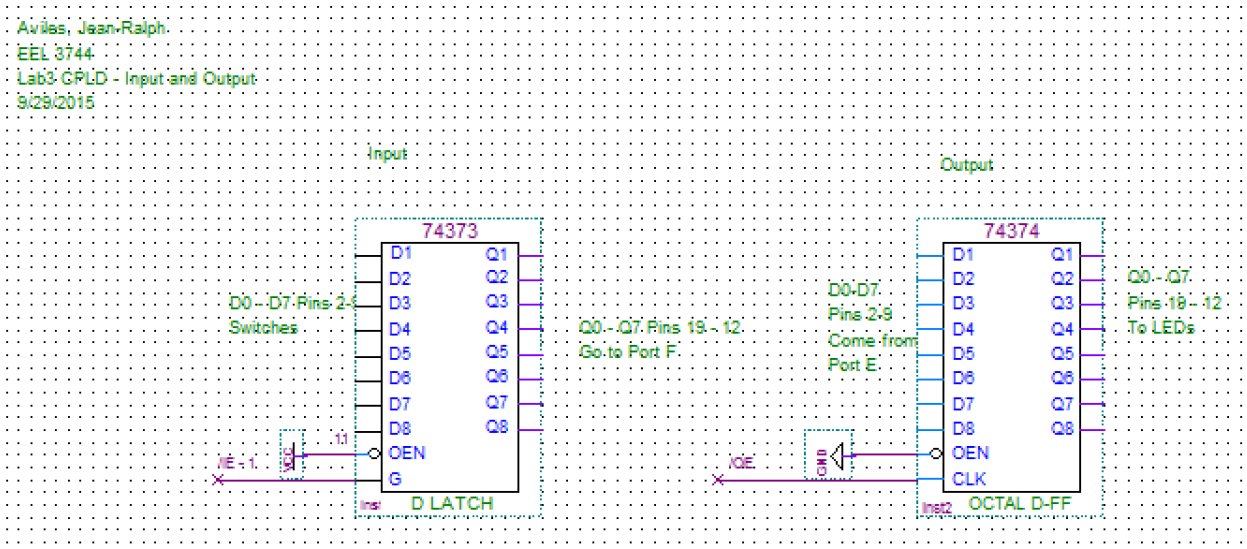
Address size

0x9FFF - 0x8000 = 0x1FFF. 0x1FFF + 1 = 0x2000 = **8192** addresses. Since we are indexing 0x2000 addresses we'll need 13 address bits.
EBI_CS0_CTRLA = 13.

CPLD Schematic



Wiring diagram for input and output



Pseudocode/Flowcharts

I/O Software

```
program x:
begin
loop forever
  x := input(0x8000) * Read input
  for i := 0 to 7
    output(x << i, 0x8000) * Shift bits
    sleep(2000) * Sleep 2000ms
  done
pool
end x.
```

Programs

I/O Software

```
/*
 * Part3_JA.asm
 *
 *  Created: 9/17/2015
 *  Author: Jean-Ralph Aviles
 *  This program reads from the switches mapped to 0x8000
 *  and writes those values to the LEDs at 0x8000 and shifts
```

```
 *  the bits left by one once every two seconds for 8
 *  seconds before reading from input again and looping.
 */

.include "ATxmega128A1Udef.inc"

.equ START = 0x8000 ; Start Address

.org 0x100
  rjmp MAIN

.org 0x200
MAIN:
  ldi R16, 0b10111     ;set /WE, /RE, /CS0 to Output
  sts PORTH_DIR, R16
  ldi R16, 0b10100     ;set /RE, /WE, /CS0 to default of 1
  sts PORTH_OUTSET, R16
  ldi R16, 0xFF        ;set all PORTJ pins (D0-D7) to be outputs
                       ;As requried
  sts PORTJ_DIR, R16   ;in the data sheet.
  ldi R16, 0xFF        ;set all PORTK pins (A0-A15) to be outputs
                       ;As requried
  sts PORTK_DIR, R16   ;in the data sheet.
  ldi R16, 0x01        ;Store 0x01 in EBI_CTRL register to select
                       ;3 port EBI(H, J, K) mode and SRAM ALE1
                       ;mode
  sts EBI_CTRL, R16

  ldi r16, 0b0010101   ;Set CTRL A to 8K address size
                       ;(0b00101) and SRAM Mode (0b01)
                       ;pg 335 8331
  sts EBI_CS0_CTRLA, r16

  ldi ZL, LOW(EBI_CS0_BASEADDR)  ; Load Z with BASEADDR
  ldi ZH, HIGH(EBI_CS0_BASEADDR) ; We will load the upper 12
  ldi r16, byte2(START)          ; bits of the START address
  st Z+, r16                     ; into BASEADDR register.
  ldi r16, byte3(START)
  st Z, r16

  ldi r16, byte3(START)          ; Set third byte of X
  sts CPU_RAMPX, r16
  ldi XL, LOW(START)             ; Load X with START address
  ldi XH, HIGH(START)
```

```
  ldi r16, 0x0                    ; Zero out the LEDs
  st X, r16

LOOP:
  ld r16, X                       ; Load value from switches.
  ldi r17, 0x8                    ; Load loop2 counter
LOOP2:
  cpi r17, 0x0                    ; Compare r17 to 0
  breq LOOP                       ; Restart when r17 is 0
  dec r17                         ; Decrement r17
  st X, r16                       ; Store value into LEDs
  push r16                        ; Save r16
  ldi r16, 0xC8                   ; Delay for 200*10 ms = 2s
  CALL DELAY                      ; Delay
  pop r16                         ; Restore r16
  ldi r18, 0x0                    ; Set r18 to 0
  ror r16                         ; Rotate right
  ror r18                         ; Get carry bit to r18
  or r16, r18                     ; Put Carry in MSB of r16
  rjmp LOOP2                      ; Return to LOOP2

DELAY:                            ; Delays by r16 x 10ms
                                  ; 66*100 instructions=10ms
  push r16                        ; Push r16 onto the stack
  cpi r16, 0                      ; Compare counter
  breq DELAY_RET                  ; If counter is 0 return
DELAY_LOOP:
  push r16                        ; Counter onto the stack
  ldi r16, 66                     ; Load outer time loop
DELAY_OUTERLOOP:
  push r16                        ; Push outer loop to stack
  ldi r16, 100                    ; Load inner time loop
DELAY_INNERLOOP:
  dec r16                         ; Decrement inner counter
  brne DELAY_INNERLOOP
  pop r16                         ; Pop outer time loop
  dec r16                         ; Decrement outer counter
  brne DELAY_OUTERLOOP
  pop r16                         ; Pop counter off of stack
  dec r16                         ; Decrement Counter
  brne DELAY_LOOP                 ; If counter != 0 loop again
DELAY_RET:
  pop r16                         ; Pop r16 off
  ret                             ; Return
```