Stern, Arion
Class #: 11303
Ian Santamauro
October 13, 2025

REQUIREMENTS NOT MET

N/A

PROBLEMS ENCOUNTERED

I encountered several issues throughout lab 4. Initially, the LEDs appeared dim and unstable because the circuit was powered from the Digilent Analog Discovery instead of the µPAD or DE10. After switching to the proper regulated supply and tying all grounds together, the LEDs operated correctly. Additionally, while configuring the 74HC573/574 latches, I misunderstood the polarity of the Latch Enable (LE) and Output Enable (OE) pins, causing inconsistent address behavior. Rewiring the latch so that LE = high (latch open) and OE = low (output active) fixed the issue. My logic analyzer initially showed incorrect address and data values because the bus lines were grouped in the wrong bit order, and the analyzer's endianness was reversed. After re-ordering the address (Q7 to Q0) and data (D3 to D0) channels, the displayed values matched the program's expected 0x10 - 0x20 addresses and 0xF - 0x5 data. After these corrections, all hardware and timing issues were resolved, and the final waveforms matched the expected EBI read and write behavior. One additional thing, is that I was unsure what to do for the pre-lab exercise when components were not needed in the hardware design.

FUTURE WORK/APPLICATIONS

If provided with additional time and hardware, the work completed in this lab could be extended to design a more advanced memory-mapped peripheral system. The existing EBI framework could be expanded to interface with multiple external devices, such as ADCs, sensors, or display controllers, allowing the microcontroller to communicate with complex components through the same bus structure. Additional software could also be developed to implement dynamic address decoding, DMA-style data transfers, or performance benchmarking between internal and external memory operations. These extensions would demonstrate how the EBI system can serve as the foundation for scalable embedded systems with flexible memory and I/O architectures.

PRE-LAB EXERCISES

i. For each SRAM configuration within the EBI system of the ATxmega128A1U microcontroller, to which address lines do you have external, physical access? Additionally, for the SRAM 3-PORT ALE1 configuration, is it possible to have external, physical access to any address lines above A15? Why or why not?
    i. For each each SRAM configuration within the EBI system of the ATxmega128A1U, you have external physical access to the following address lines:
        1. 3PORT ALE1: A[19:0]
        2. 3PORT ALE12: A[23:0]
        3. 4PORT ALE2: A[23:0]
        4. 4PORT NOALE: A[21:0]
        5. LPC 2PORT ALE1: A[7:0]
        6. LPC 3PORT/4PORT ALE1: A[19:0]
        7. LPC 2/3/4PORTALE12: A[19:0]
    ii. For the SRAM 3-PORT ALE1 configuration, the ATxmega128A1U provides external access to address lines A[15:0] through PORT K, but the higher address lines (A19-A16) share pins with the chip select outputs on PORT H (PH4-PH7). If any of the chip select signals are not used, those corresponding pins can function as additional address outputs. However, the remaining upper address bits, A20-A23, are not externally available in this configuration.

ii. Describe what performing full address decoding and partial address decoding signifies. Provide examples of both types for [1] an SRAM chip and [2] either an input port or output port.
  a. Full Address Decoding
    i. Full address decoding means that every internal address within some external component has a unique mapping within the microcontroller data memory space; that all higher, unused address lines are included in the chip select logic, so each external device is assigned a unique and non-repeating address range in the memory map.
    ii. Ex SRAM: For example, if there are 16 address lines, a 2K SRAM chip could be fully decoded by using a PLD to generate its chip-select line only when A15=0, A14=1, A13=0, and A12=0. This would map the SRAM precisely to the address range 0x4000–0x47FF. Any other address combination would leave the chip unselected, ensuring the memory device is accessed only within that 2K window.
    iii. Ex PORT: For example, if the microcontroller has 16 address lines (A15-A0), an output port using a 74HC574 latch could be fully decoded to respond only at address 0x4A37, meaning the chip-select is asserted only when all address bits match 0b 0100 1010 0011 0111, and any change in even a single address bit would deactivate the port.
  b. Partial Address Decoding:
    i. Partial address decoding occurs when only a subset of the higher address lines are used in the chip select logic. This reduces the hardware needed for decoding but cause the same device to be correspond to multiple values in the memory space since unused address bits can vary without changing the decoded output.
    ii. Ex SRAM: For example, if there are 16 address lines and a 2K SRAM only uses A15 and A14 to determine its chip-select signal (CS = E * /A15 * A14), then the device will be

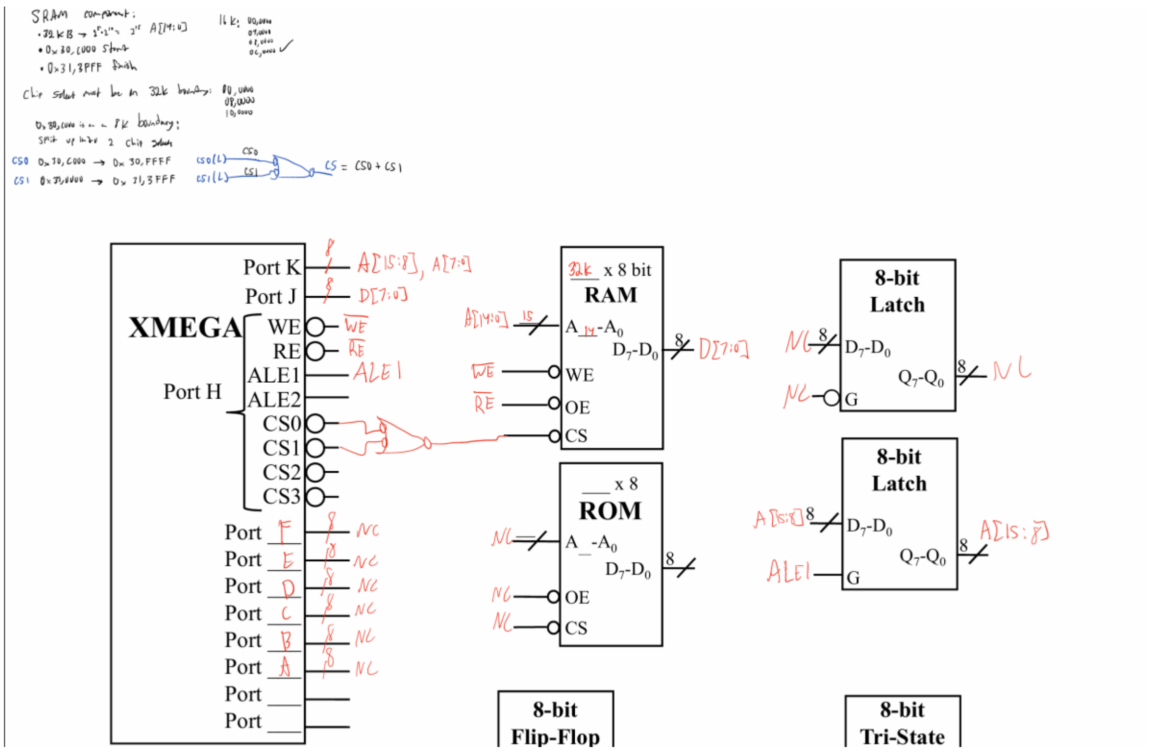
active for every combination of the remaining high address bits. This means the same SRAM would appear not only at 0x4000-0x47FF, but also at 0x4800-0x4FFF, 0x7000-0x77FF, and other mirrored 2 KB ranges throughout memory, since A13–A0 are ignored in the decoding logic.

iii. Ex PORT: For an input port implemented with a 74HC573 latch, partial decoding might use only one high-order bit, such as A8, for its enable signal (CS = A8 * R/~W). This would cause the port to be readable at multiple addresses like 0x0100, 0x1100, 0x3100, and 0x7100, all corresponding to the same physical input port because the unused higher address lines are not checked by the decoding logic.

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

a. In theory you can map 16,760,832 8-bit I/O ports to the external data memory space of an ATxmega128A1U microcontroller, assuming that at most one port utilizes any particular address range. This is because in theory, one 8-bit I/O port can be mapped to each unique external byte address, giving a maximum equal to the size of the XMEGA's external data memory region from 0x4000 to 0xFFFFFF. This corresponds to 2^24 -2^14 = 16,777,216 – 16,384 = 16,760,832 distinct addresses, allowing up to 16,760,832 separate 8-bit ports if each is fully decoded to a single address with no mirroring. As explained in Exercise ii, partial decoding would cause a single physical port to appear at multiple mirrored addresses, reducing the total number of unique ports, whereas full decoding (using all address bits) ensures each port occupies one exact address. Although the EBI chip-selects operate on 4 KB boundaries, external logic such as PLDs can still fully decode each address within that window.

iv. Assume that an SRAM component with the same size as the one on the OOTB Memory Base has to be added to a different computing system containing an ATxmega128A1U but no OOTB Memory Base (i.e., this is not the OOTB µPAD computing system), with the first address of the SRAM starting at address 0x30 C000, instead of the address specified in the "Lab 4 – Hardware Expansion" quiz. Design, on paper, a hardware expansion for this new system, utilizing the same structure of steps laid out in this quiz. Use the same "overall" memory-mapping constraints presented within the quiz. . Last memory address: 0x31 3FFF.


SRAM component:
- 32 KB → $2^5 \cdot 2^{10} = 2^{15}$ A[14:0]
- 0x 30,C000 Start
- 0x 31,3FFF finish

16 k: 00,0000 04,0000 08,0000 0C,0000 ✓

Chip select must be on 32k boundary: 00,0000 08,0000 10,0000

0x 30,C000 is on a 8K boundary; split up into 2 chip selects

CS0 0x 30,C000 → 0x 30,FFFF
CS1 0x 31,0000 → 0x 31,3FFF

v. Assume that some external 128 KB SRAM must be fully mapped to the data memory space of the ATxmega128A1U, with the first memory location of the SRAM corresponding to data memory address 0xA9 0000. Design, on paper, a hardware expansion for this new system, utilizing the same structure of steps laid out in the relevant pre-lab quiz. Use the same "overall" memory-mapping constraints presented within this quiz. Hint: Consider how many address signals are needed to make the SRAM fully addressable and then consider how you will gain access to all these signals.

SRAM component:

- 128 KB → $2^7 \cdot 2^{10} = 2^{17}$ A[16:0]
- 0x A9,0000 start
- 0x AA,FFFF finish

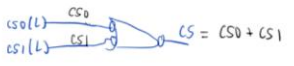
32K = 0x F000
64K = 0x10000
128K = 0x20,000

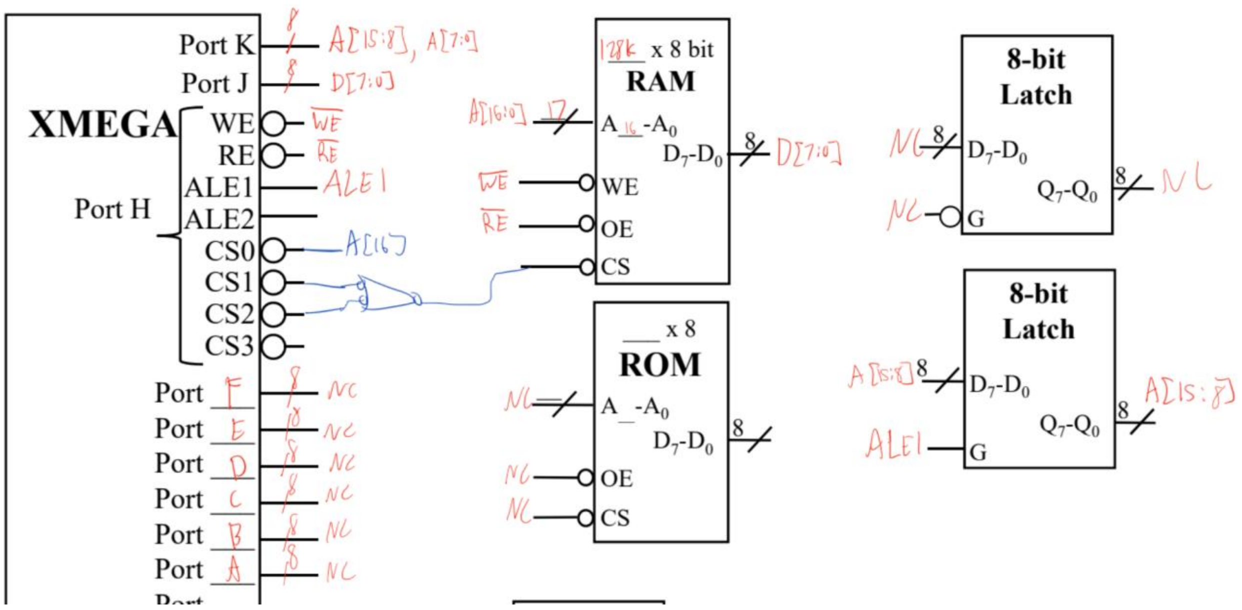
$$\begin{array}{r} \text{A9,0000} \\ + \quad \text{1,FFFF} \\ \hline \text{AA,FFFF} \end{array}$$

Chip select must be in 128K boundary:

0x A9,0000 is on a 64K boundary: 0x 10,000
Split up into 2 chip selects

CS1 0x A9,0000 → 0x A9,FFFF
CS2 0xAA,0000 → 0x AA,FFFF





vi. Assume that some 8-bit input port should be accessible via the 256 consecutive addresses starting at 0xDA D400 within the ATxmega128A1U data memory space. Design, on paper, a hardware expansion for this port, utilizing the same structure of steps laid out in the relevant pre-lab quiz. Use the same "overall" memory-mapping constraints presented within this quiz.

Input Port component:

- 256 → 2^8 2^8 → 1 0000 one = 0x100
- 0x DA,D400 Start
- 0x DA,D4FF finish

$$\begin{array}{r} \text{DA,D400} \\ +\ \text{0FF} \\ \hline \text{DA,D4FF} \end{array}$$

Chip select must be on 4K boundary minimum: 0x1000

0x DA,D400 is not on a 4K boundary: 0x1000

- Use 1 chip select starting at DA, D000 and additional gates

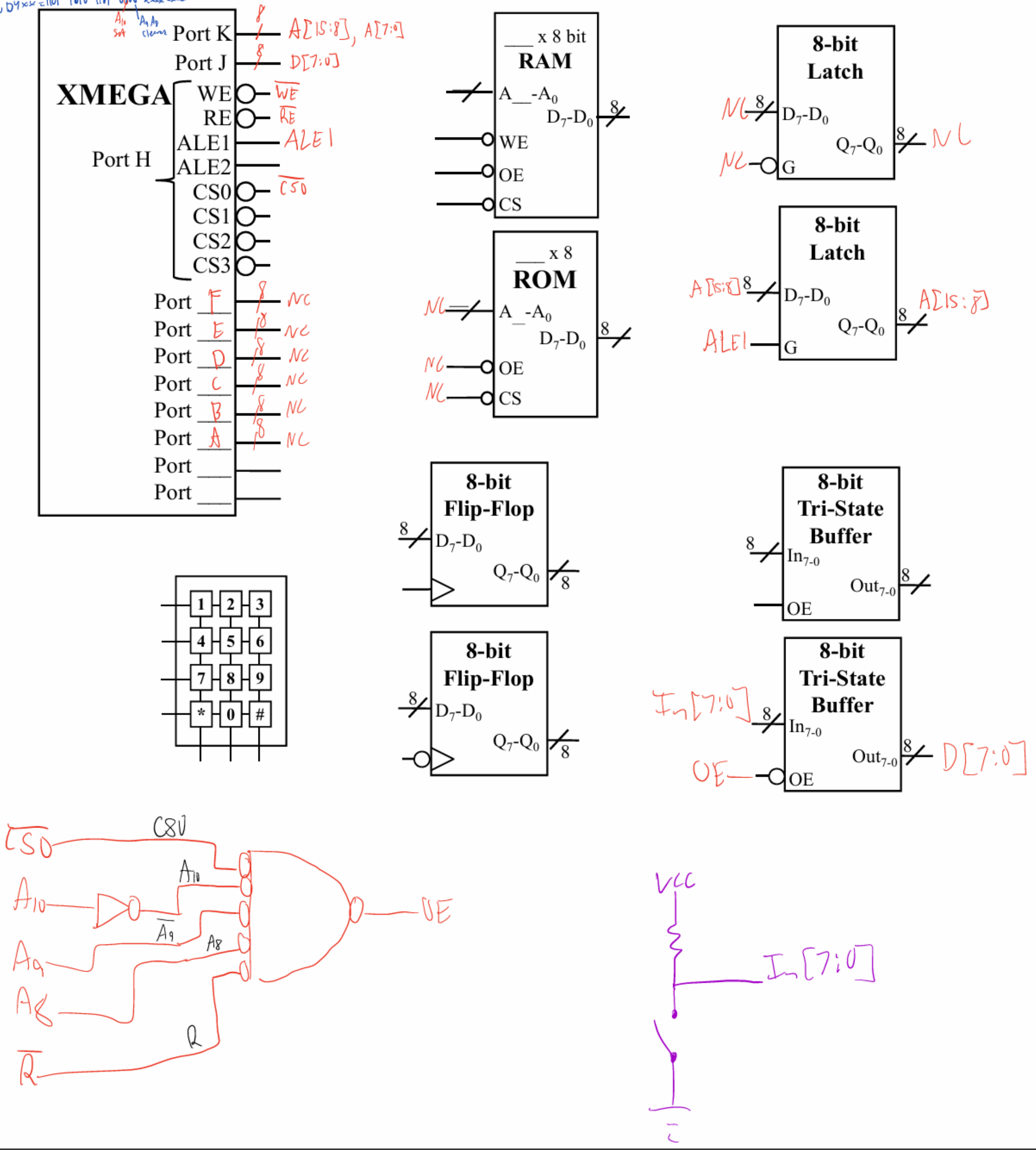
0xDA,D000 → 0x DA,D7FF
2K size

1K = 0x400
2K = 0x800

Additional mapping: $A_{10} \cdot \overline{A_9} \cdot \overline{A_8}$

DA,D000 = 1101 1010 1101 0000 0000 0000
DA,D7FF = 1101 1010 1101 0111 1111 1111
DA,D4xx = 1101 1010 1101 0100 xxxx xxxx

$CS0 = A_{23} \cdot A_{22} \cdot \overline{A_{21}} \cdot A_{20} \cdot A_{19} \cdot \overline{A_{18}} \cdot A_{17} \cdot \overline{A_{16}} \cdot A_{15} \cdot A_{14} \cdot \overline{A_{13}} \cdot A_{12} \cdot \overline{A_{11}}$



vii. Assume that some external 1 MB SRAM must be fully mapped to the data memory space of the ATxmega128A1U, with the first memory location of the SRAM corresponding to data memory address 0x50 0000. Design, on paper, a hardware expansion for this new system, utilizing the same

SRAM component:

- 1 MB → 1000 KB ≈ $2^{10} \cdot 2^{10} = 2^{20}$ = 1,0000,0000,0000,0000,0000
- 0x50,0000 start
- 0x5F,FFFF finish

0x 1 0 0 0 0 0

0x 10,0000

0x50,0000
+0x0F,FFFF
0x5F,FFFF

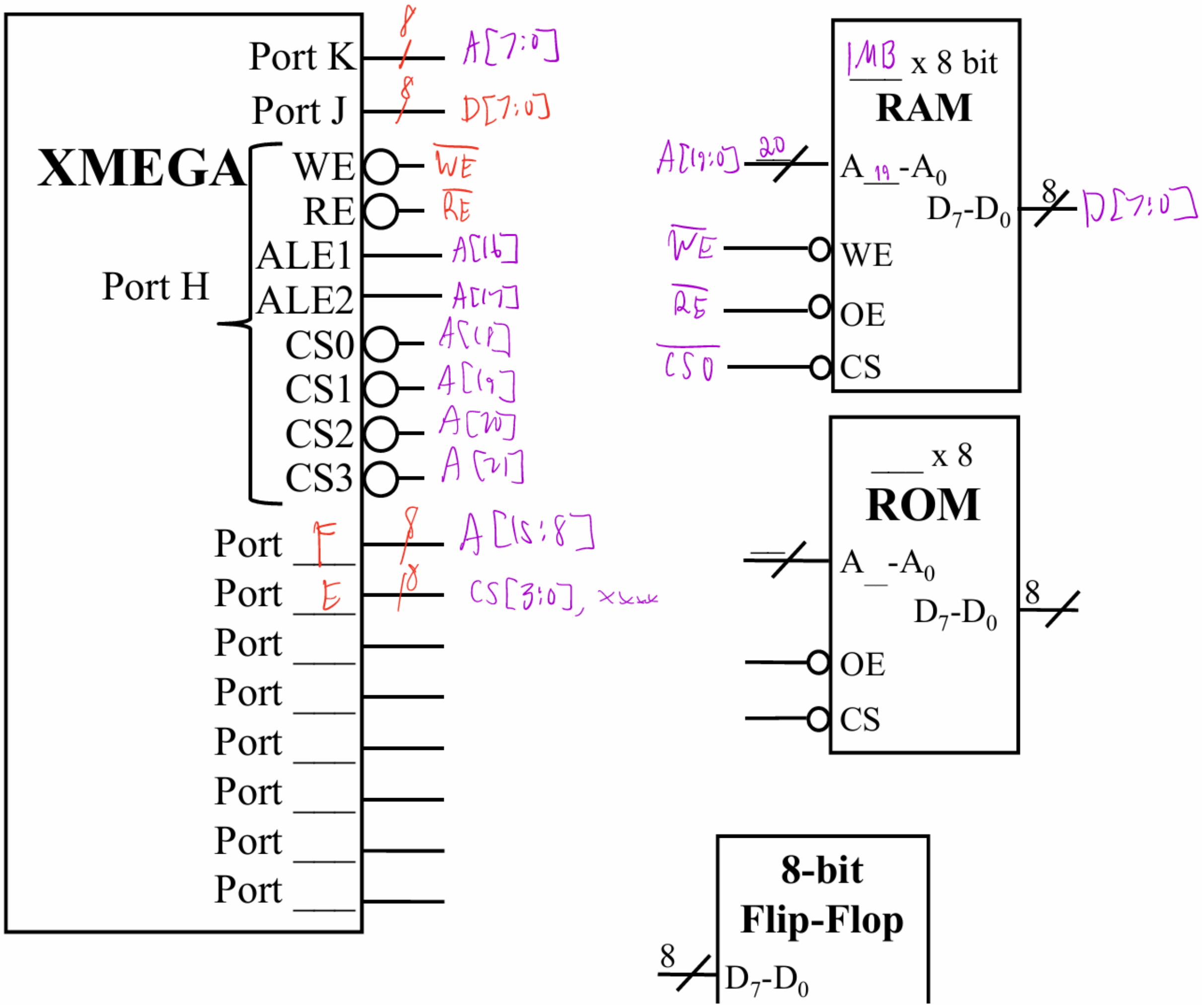
Chip select is on a 1MB boundary: 0x 10,0000

Use different EBI configuration since we need to access higher address bits externally for 1MB SRAM

SRAM 4 PORT NO ALE

0x50,0000 → 0101 0000 0000 0000 0000 0000
0x5F,FFFF → 0101 1111 1111 1111 1111 1111
19 5 11 7 3

$CS0 = \overline{A_{23}} \cdot A_{22} \cdot \overline{A_{21}} \cdot A_{20}$



viii. How would one go about utilizing the SRAM 4PORT NOALE mode? (Note: Looking at the Alternate Pin Functions only ports H, J, and K by default can be used with the EBI system. How can the other port be activated)?

a. To utilize the SRAM 4-PORT NOALE mode, the External Bus Interface (EBI) must first be configured for four-port operation by setting the SRMODE bits in the EBI.CTRL register to 11 and the IFMODE bits in the EBI.CTRL register to 10, which enables four dedicated ports for address, data, and control signals without the need for multiplexing or an ALE. In this mode, Ports J and K are automatically used for the data (D[7:0]) and lower address lines (A[7:0]), while Port H provides the control signals (/CS, /WE, /RE) and higher address bits (A[19:16]). To access all 20 address lines, an additional port (either Port E or Port F) must be activated using the EBIOUT register. The EBIOUT_EBIADROUT[1:0] bits determine which port outputs the mid-level address lines (A[15:8]): setting these bits to 00 or 10 routes them to Port F, while 01 or 11 routes them to Port E (10 and 11 only route the upper nibble). Similarly, the EBIOUT_EBICSOUT[1:0] bits determine which port outputs the chip-select lines (CS0–CS3), with available options being Port H (00), Port L (01), Port F (10), or Port E (11). These configurations are only valid in 4-PORT mode, and the selected ports must also have their data direction registers configured as outputs to drive the external bus.

PSEUDOCODE/FLOWCHARTS

SECTION 1

lab4_2.asm (ebi_io_init and ebi_init taken from pre-lab quiz):

```
;assembler directives
;equates

.org 0
Rjmp main

.org 0x100
MAIN:
;init stack
Ldi r16, high(0x3FFF)
Sts CPU_SPH, r16
Ldi r16, low(0x3fff)
Sts CPU_SPL, r16

Rcall EBI_IO_INIT
Rcall EBI_INIT

LOOP:
;load value from input port, 0xBEE640 → 0xBEE67F

;write to the leds in OUTPUT port, 0xBEE640 → 0xBEE67F

Rjmp LOOP
```

```
/*------------------------------------------------------------
 ebi_io_init --

 Description:
  Initialize and enable the EBI IO pins for the relevant
  hardware expansion.

  Input(s): N/A
  Output(s): N/A
------------------------------------------------------------*/
EBI_IO_INIT:

  ; Symbols for start of relevant memory address ranges.
  .equ SRAM_START_ADDR 0xA68000


  .equ IO_START_ADDR = 0xBEE640



  ; Preserve the relevant register.
push r16



  ; Initialize the relevant EBI control signals to be
  ; in a `false` state.
  ldi r16, 0b01010011


  sts PORTH_OUTSET, r16

  ldi r16 0b00000100


  sts PORTH_OUTCLR, r16

  ; Initialize the EBI control signals to be output from
  ; the microcontroller.
  ldi r16, 0b01010111


  sts <blank>, r16                          PORTH_DIRSET

  ; Initialize the address signals to be output
  ; from the microcontroller.
  ldi r16, 0xFF


  sts <blank>, r16                          PORTK_DIRSET

  ; Recover the relevant register.
  pop r16



  ; Return from subroutine.
  ret
```

```
/*------------------------------------------------------------
 ebi_init --

 Description:
  Initialize and enable the EBI system for the relevant
  hardware expansion.

  Input(s): N/A
  Output(s): N/A
------------------------------------------------------------*/
EBI_INIT:

  ; Preserve the relevant register. (See 3 above)
  <blank>           push r16

  ; Initialize the EBI system for SRAM 3-PORT ALE1 mode.
  ldi r16, <blank>         0b00000001


  sts EBI_CTRL, r16

  ; Initialize the relevant chip select(s).
  ; The following dropdown boxes contain a blank
   ; (i.e., empty) option; select this option if
   ; no other option applies.




  ; Configure chip select CS2, only if necessary.
  ldi r16, 0b00001101


  sts EBI_CS2_CTRLA, r16


  ldi r16, byte2(IO_START_ADDR)"


  sts EBI_CS2_BASEADDR, r16


 ldi r16, byte3(IO_START_ADDR)


  sts EBI_CS2_BASEADDR+1, r16





  ; Recover the relevant register.
  pop r16



  ; Return from subroutine.
  ret
```

lab4_3a.asm (ebi_io_init and ebi_init taken from pre-lab quiz):

```
;assembler directives
;equates


.org 0
Rjmp main


.org TCC0_OVF_vect
Rjmp TCC0_OVF_ISR



.org 0x100
MAIN:
;init stack
Ldi r16, high(0x3FFF)
Sts CPU_SPH, r16
Ldi r16, low(0x3fff)
Sts CPU_SPL, r16


;initialize EBI I/O pins and control system
Rcall EBI_IO_INIT
Rcall EBI_INIT


;initialize timer for 300ms overflow period
Rcall INIT_TCC0


;initialize interrupts for timer overflow
Rcall INTR_INIT
```

;copy data from program memory (sram_data_asm.txt)

;to external SRAM starting at 0xA68000

Rcall SRAM_WRITE_BLOCK

;initialize pointers for read-back and LED output

Rcall POINTER_INIT

LOOP:

Rjmp LOOP

```
/*------------------------------------------------------------
 ebi_io_init --

 Description:
  Initialize and enable the EBI IO pins for the relevant
  hardware expansion.

  Input(s): N/A
  Output(s): N/A
------------------------------------------------------------*/
EBI_IO_INIT:

  ; Symbols for start of relevant memory address ranges.
  .equ SRAM_START_ADDR 0xA68000


  .equ IO_START_ADDR = 0xBEE640



  ; Preserve the relevant register.
push r16



  ; Initialize the relevant EBI control signals to be
  ; in a `false` state.
  ldi r16, 0b01010011


  sts PORTH_OUTSET, r16

  ldi r16 0b00000100
  sts PORTH_OUTCLR, r16

  ; Initialize the EBI control signals to be output from
  ; the microcontroller.
  ldi r16, 0b01010111


  sts <blank>, r16                         PORTH_DIRSET

  ; Initialize the address signals to be output
  ; from the microcontroller.
  ldi r16, 0xFF


  sts <blank>, r16                         PORTK_DIRSET

  ; Recover the relevant register.
  pop r16



  ; Return from subroutine.
  ret
```

```
/*------------------------------------------------------------
 ebi_init --

 Description:
  Initialize and enable the EBI system for the relevant
  hardware expansion.

  Input(s): N/A
  Output(s): N/A
------------------------------------------------------------*/
EBI_INIT:

  ; Preserve the relevant register. (See 3 above)
  push r16

  ; Initialize the EBI system for SRAM 3-PORT ALE1 mode.
  ldi r16, 0b00000001


  sts EBI_CTRL, r16

  ; Initialize the relevant chip select(s).
  ; The following dropdown boxes contain a blank
   ; (i.e., empty) option; select this option if
   ; no other option applies.

  ; Configure chip select CS0, only if necessary.
  ldi r16, 0b00011101


  sts EBI_CS0_CTRLA, r16


  ldi r16, byte2(SRAM_START_ADDR)


sts EBI_CS0_BASEADDR, r16
ldi r16, byte3(SRAM_START_ADDR)


sts EBI_CS0_BASEADDR+1, r16
```

```
  ; Configure chip select CS2, only if necessary.
  ldi r16, 0b00001101


  sts EBI_CS2_CTRLA, r16


  ldi r16, byte2(IO_START_ADDR)"


  sts EBI_CS2_BASEADDR, r16


 ldi r16, byte3(IO_START_ADDR)


  sts EBI_CS2_BASEADDR+1, r16







  ; Recover the relevant register.
  pop r16



  ; Return from subroutine.
  ret
```

;NAME: INIT_TCC0

;PURPOSE: Configure TCC0 to overflow every 300ms

;Input(s):

;Output(s):

;Registers Affected:

INIT_TCC0:

;push registers

;set CNT = 0

;set PER to correspond to 300ms (with chosen prescaler)

;set prescaler and enable timer

;pop registers

Ret

;NAME: INTR_INIT

```
;PURPOSE: Enable interrupts for TCC0 overflow
;Input(s):
;Output(s):
;Registers Affected:
INTR_INIT:
;push registers
;set TCC0 overflow interrupt level (low)
;enable low-level interrupts in PMIC
;enable global interrupt flag (sei)
;pop registers
Ret


;NAME: SRAM_WRITE_BLOCK
;PURPOSE: Copy data from program memory to external SRAM
;Input(s): sram_data_asm.txt (in program memory)
;Output(s): data written sequentially to 0xA68000+
;Registers Affected:
SRAM_WRITE_BLOCK:
;push registers
;initialize Z to start of sram_data
;initialize X to start of external SRAM
;FOR each byte in sram_data
;   read byte from FLASH (LPM)
;   write byte to SRAM (ST X+)
;END FOR
;pop registers
ret


;NAME: POINTER_INIT
```

```
;PURPOSE: Prepare pointers for read-back and output to LEDs
;Input(s):
;Output(s):
;Registers Affected:
POINTER_INIT:
;push registers
;set X pointer to first SRAM byte (read-back)
;set Y pointer to I/O port address (0xBEE640)
;pop registers
Ret




;NAME: TCC0_OVF_ISR
;PURPOSE: Read next byte from external SRAM every 300ms
;         and write that byte to LED output port.
;Input(s): Data from SRAM via pointer X
;Output(s): Data sent to LEDs via shared I/O address
;Registers Affected:
TCC0_OVF_ISR:
;preserve status register
;clear overflow flag
;load byte from SRAM (LD X+)
;write byte to I/O port (ST Y)
;if end of data reached, reset X pointer to first byte
;restore status register
Reti
```

Lab4_3b.asm:

```
;assembler directives
;equates

.org 0
Rjmp main

.org 0x100
MAIN:
;init stack
Ldi r16, high(0x3FFF)
Sts CPU_SPH, r16
Ldi r16, low(0x3fff)
Sts CPU_SPL, r16

;initialize EBI I/O pins and control system
Rcall EBI_IO_INIT
Rcall EBI_INIT

;continuously perform alternating write and read cycles to two SRAM addresses
;used to observe EBI signals on LSA
LOOP:
;load X pointer with first SRAM address (ADDR0)

;write 0x0A to ADDR0 and read it back

;load X pointer with second SRAM address (ADDR1)

;write 0x05 to ADDR1 and read it back
```

Rjmp loop

```
/*------------------------------------------------------------
 ebi_io_init --

 Description:
  Initialize and enable the EBI IO pins for the relevant
  hardware expansion.

  Input(s): N/A
  Output(s): N/A
------------------------------------------------------------*/
EBI_IO_INIT:

  ; Symbols for start of relevant memory address ranges.
  .equ SRAM_START_ADDR 0xA68000


  .equ IO_START_ADDR = 0xBEE640



  ; Preserve the relevant register.
push r16



  ; Initialize the relevant EBI control signals to be
  ; in a `false` state.
  ldi r16, 0b01010011


  sts PORTH_OUTSET, r16

  ldi r16 0b00000100


  sts PORTH_OUTCLR, r16

  ; Initialize the EBI control signals to be output from
  ; the microcontroller.
  ldi r16, 0b01010111


  sts <blank>, r16                        PORTH_DIRSET

  ; Initialize the address signals to be output
  ; from the microcontroller.
  ldi r16, 0xFF


  sts <blank>, r16                        PORTK_DIRSET

  ; Recover the relevant register.
  pop r16



  ; Return from subroutine.
  ret
```

```
/*------------------------------------------------------------
 ebi_init --

 Description:
  Initialize and enable the EBI system for the relevant
  hardware expansion.

  Input(s): N/A
  Output(s): N/A
------------------------------------------------------------*/
EBI_INIT:

  ; Preserve the relevant register. (See 3 above)
  push r16

  ; Initialize the EBI system for SRAM 3-PORT ALE1 mode.
  ldi r16, 0b00000001


  sts EBI_CTRL, r16

  ; Initialize the relevant chip select(s).
  ; The following dropdown boxes contain a blank
   ; (i.e., empty) option; select this option if
   ; no other option applies.

  ; Configure chip select CS0, only if necessary.
  ldi r16, 0b00011101


  sts EBI_CS0_CTRLA, r16


  ldi r16, byte2(SRAM_START_ADDR)


sts EBI_CS0_BASEADDR, r16
ldi r16, byte3(SRAM_START_ADDR)


sts EBI_CS0_BASEADDR+1, r16



  ; Configure chip select CS2, only if necessary.
  ldi r16, 0b00001101


  sts EBI_CS2_CTRLA, r16


  ldi r16, byte2(IO_START_ADDR)"


  sts EBI_CS2_BASEADDR, r16


 ldi r16, byte3(IO_START_ADDR)


  sts EBI_CS2_BASEADDR+1, r16
```

```
    ; Recover the relevant register.
    pop r16



    ; Return from subroutine.
    ret
```

PROGRAM CODE

SECTION 2

Lab4_2.asm

```
;
; Lab4.asm
;
; Created: 10/11/2025 10:52:46 PM
; Author : arist
;
;===============================================================================
; Lab 4, Section 22
; Name: Arion Stern
; Class #: 11303
; PI Name: Ian Santamauro
; Description: Configure and test external 8-bit I/O ports using the EBI system.
;              Initializes the EBI control and address signals, maps an external
;              input port and output port, and continually mirrors DIP switch
;              values from the input port to the LED bank on the output port.
;===============================================================================

;SECTION 1

;lab4_2.asm (ebi_io_init and ebi_init taken from pre-lab quiz):

;assembler directives
.include "ATxmega128A1Udef.inc"
;equates
  .equ SRAM_START_ADDR  = 0xA68000

  .equ IO_START_ADDR = 0xBEE640

.org 0
Rjmp main

.org 0x100
MAIN:
;init stack
Ldi r16, high(0x3FFF)
Sts CPU_SPH, r16
Ldi r16, low(0x3fff)
Sts CPU_SPL, r16

Rcall EBI_IO_INIT
Rcall EBI_INIT

;load the y register with the input port address
;load value from input port, 0xBEE640 □ 0xBEE67F
ldi YL, byte1( IO_START_ADDR)
ldi YH, byte2(IO_START_ADDR)
ldi r16, byte3( IO_START_ADDR)
sts CPU_RAMPY, r16


LOOP:
;load value from input port, 0xBEE640 □ 0xBEE67F (Y reg)
ld r16, Y
```

```
;write to the leds in OUTPUT port, 0xBEE640 □ 0xBEE67F
st Y, r16

Rjmp LOOP




/*********************************************************************************
* Name:         EBI_IO_INIT
* Purpose:      Configure Port H and Port K pins for EBI control and address signals.
* Inputs:       None
* Outputs:      None
* Affected:     PORTH_DIRSET, PORTH_OUTSET, PORTH_OUTCLR, PORTK_DIRSET
*********************************************************************************/
EBI_IO_INIT:

  ; Symbols for start of relevant memory address ranges.


  ; Preserve the relevant register.
push r16


  ; Initialize the relevant EBI control signals to be
  ; in a `false` state.
  ldi r16, 0b01010011

  sts PORTH_OUTSET, r16

  ldi r16, 0b00000100

  sts PORTH_OUTCLR, r16

  ; Initialize the EBI control signals to be output from
  ; the microcontroller.
  ldi r16, 0b01010111

  sts  PORTH_DIRSET, r16

  ; Initialize the address signals to be output
  ; from the microcontroller.
  ldi r16, 0xFF

  sts  PORTK_DIRSET, r16

  ; Recover the relevant register.
  pop r16


  ; Return from subroutine.
  ret

/*********************************************************************************
* Name:         EBI_INIT
* Purpose:      Initialize the External Bus Interface (EBI) for SRAM 3-PORT ALE1 mode
*               and configure chip select CS2 for the external I/O port.
* Inputs:       None
* Outputs:      None
* Affected:    EBI_CTRL, EBI_CS2_CTRLA, EBI_CS2_BASEADDR
*************************************************************************************/
EBI_INIT:

  ; Preserve the relevant register. (See 3 above)
  push r16

  ; Initialize the EBI system for SRAM 3-PORT ALE1 mode.
  ldi r16, 0b00000001

  sts EBI_CTRL, r16

  ; Initialize the relevant chip select(s).
  ; The following dropdown boxes contain a blank
   ; (i.e., empty) option; select this option if
   ; no other option applies.



  ; Configure chip select CS2, only if necessary.
  ldi r16, 0b00001101

  sts EBI_CS2_CTRLA, r16

  ldi r16, byte2(IO_START_ADDR)

  sts EBI_CS2_BASEADDR, r16

 ldi r16, byte3(IO_START_ADDR)

  sts EBI_CS2_BASEADDR+1, r16




  ; Recover the relevant register.
  pop r16


  ; Return from subroutine.
  ret
```

Lab4_3a.asm

```
/*
 * lab4_3a.asm
 *
 *  Created: 10/12/2025 12:17:11 AM
 *   Author: arist
 */

; lab4_3a.asm (ebi_io_init and ebi_init taken from pre-lab quiz):
;==============================================================================
; Lab 4, Section 22
; Name: Arion Stern
; Class #: 11303
; PI Name: Ian Santamauro
; Description: Configure EBI for external SRAM and sequentially write data from
;              program memory (sram_data_asm.txt) into external SRAM at 0xA68000.
;              Then, using Timer/Counter C0 overflow interrupts (~300 ms period),
;              read back one byte at a time and display it on the external LED
;              output port mapped through the I/O region.
;===============================================================================


;assembler directives
.include "ATxmega128A1Udef.inc"

.org 0x2000
sram_data:
    .include "sram_data_asm.txt"
sram_data_end:

;equates
  .equ SRAM_START_ADDR = 0xA68000
  .equ IO_START_ADDR = 0xBEE640

  .equ DATA_COUNT = sram_data_end - sram_data
  .equ SRAM_END_ADDR   = SRAM_START_ADDR + DATA_COUNT


.org 0
Rjmp main

.org TCC0_OVF_vect
Rjmp TCC0_OVF_ISR


.org 0x100
MAIN:
;init stack
Ldi r16, high(0x3FFF)
Sts CPU_SPH, r16
Ldi r16, low(0x3fff)
Sts CPU_SPL, r16

;init portC for debug purposes
ldi  r16, (1<<0)
sts  PORTC_DIRSET, r16
sts  PORTC_OUTSET, r16

;initialize EBI I/O pins and control system
Rcall EBI_IO_INIT
Rcall EBI_INIT

;initialize timer for 300ms overflow period
Rcall INIT_TCC0

;initialize interrupts for timer overflow
Rcall INTR_INIT

;copy data from program memory (sram_data_asm.txt)
;to external SRAM starting at 0xA68000
Rcall SRAM_WRITE_BLOCK
```

```
;initialize pointers for read-back and LED output
Rcall POINTER_INIT

;start timer
ldi r16, TC_CLKSEL_DIV256_gc
sts TCC0_CTRLA, r16

LOOP:
Rjmp LOOP





/*************************************************************************************
* Name:         EBI_IO_INIT
* Purpose:      Configure Port H and Port K pins for EBI control and address signals.
* Inputs:       None
* Outputs:      None
* Affected:     PORTH_DIRSET, PORTH_OUTSET, PORTH_OUTCLR, PORTK_DIRSET
*************************************************************************************/

EBI_IO_INIT:

  ; Symbols for start of relevant memory address ranges.


  ; Preserve the relevant register.
push r16


  ; Initialize the relevant EBI control signals to be
  ; in a `false` state.
  ldi r16, 0b01010011

  sts PORTH_OUTSET, r16

  ldi r16, 0b00000100

  sts PORTH_OUTCLR, r16

  ; Initialize the EBI control signals to be output from
  ; the microcontroller.
  ldi r16, 0b01010111

  sts PORTH_DIRSET, r16

  ; Initialize the address signals to be output
  ; from the microcontroller.
  ldi r16, 0xFF

  sts PORTK_DIRSET, r16

  ; Recover the relevant register.
  pop r16


  ; Return from subroutine.
  ret

/*************************************************************************************
```

```
* Name:         EBI_INIT
* Purpose:      Configure EBI for SRAM 3-PORT ALE1 mode.
* Inputs:       None
* Outputs:      None
* Affected:     EBI_CTRL, EBI_CS0_CTRLA, EBI_CS0_BASEADDR, EBI_CS2_CTRLA, EBI_CS2_BASEADDR
*********************************************************************************/

EBI_INIT:

  ; Preserve the relevant register. (See 3 above)
  push r16

  ; Initialize the EBI system for SRAM 3-PORT ALE1 mode.
  ldi r16, 0b00000001

  sts EBI_CTRL, r16

  ; Initialize the relevant chip select(s).
  ; The following dropdown boxes contain a blank
   ; (i.e., empty) option; select this option if
   ; no other option applies.

  ; Configure chip select CS0, only if necessary.
  ldi r16, 0b00011101

  sts EBI_CS0_CTRLA, r16

  ldi r16, byte2(SRAM_START_ADDR)

sts EBI_CS0_BASEADDR, r16
ldi r16, byte3(SRAM_START_ADDR)

sts EBI_CS0_BASEADDR+1, r16


  ; Configure chip select CS2, only if necessary.
  ldi r16, 0b00001101

  sts EBI_CS2_CTRLA, r16

  ldi r16, byte2(IO_START_ADDR)

  sts EBI_CS2_BASEADDR, r16

 ldi r16, byte3(IO_START_ADDR)

  sts EBI_CS2_BASEADDR+1, r16


  ; Recover the relevant register.
  pop r16


  ; Return from subroutine.
  ret

/*********************************************************************************
* Name:         INIT_TCC0
* Purpose:      Configure Timer/Counter C0 to overflow approximately every 300 ms.
* Inputs:       None
* Outputs:      None
* Affected:    TCC0_CNT, TCC0_PER
*************************************************************************************/

INIT_TCC0:
;push registers
push r16
;set CNT = 0
ldi r16, 0
sts TCC0_CNT, r16
sts TCC0_CNT + 1, r16
;set PER to correspond to 300ms (with 256 prescaler: around 2344)
ldi  r16, low (2344)
sts TCC0_PER, r16
ldi r16, high (2344)
sts TCC0_PER + 1, r16


;pop registers
pop r16

Ret


/*************************************************************************************
* Name:        INTR_INIT
* Purpose:     Enable interrupts for TCC0 overflow.
* Inputs:      None
* Outputs:     None
* Affected:    TCC0_INTCTRLA, PMIC_CTRL
*************************************************************************************/

INTR_INIT:
;push registers
push r16
;set TCC0 overflow interrupt level (low)
ldi r16, TC_OVFINTLVL_LO_gc
sts TCC0_INTCTRLA, r16
;enable low-level interrupts in PMIC
ldi r16, PMIC_LOLVLEN_bm
sts PMIC_CTRL, r16

;enable global interrupt flag (sei)
sei
;pop registers
pop r16
Ret



/*************************************************************************************
* Name:        SRAM_WRITE_BLOCK
* Purpose:     Copy data from program memory (sram_data_asm.txt) to external SRAM.
* Inputs:      Data at label sram_data in program memory.
* Outputs:     Data written sequentially to SRAM starting at 0xA68000.
* Affected:    CPU_RAMPZ, CPU_RAMPX, SRAM memory contents
*************************************************************************************/

SRAM_WRITE_BLOCK:
;push registers
push r16
push ZL
push ZH
push YL
push YH
;initialize Z to start of sram_data
ldi ZL, low(sram_data<<1)
ldi ZH, high(sram_data<<1)
ldi r16, byte3(sram_data<<1)
sts CPU_RAMPZ, r16

;initialize X to start of external SRAM
 ldi  XL, byte1(SRAM_START_ADDR)
 ldi  XH, byte2(SRAM_START_ADDR)
 ldi  r16, byte3(SRAM_START_ADDR)
 sts  CPU_RAMPX, r16

  ; Y = DATA_COUNT
    ldi  YL, low(DATA_COUNT)
    ldi  YH, high(DATA_COUNT)

;FOR each byte in sram_data
WRITE_LOOP:
       ;if cnt = 0 done
       cpi  YL, 0
       ldi  r16, 0
       cpc  YH, r16
       breq WRITE_DONE

;   read byte from FLASH (LPM) inc Z
       lpm r16, Z+
;   write byte to SRAM (ST X+) inc x
       st X+, r16
       ;dec count
       subi YL, 1
    sbci YH, 0
    rjmp WRITE_LOOP




;END FOR
WRITE_DONE:
;pop registers
pop YH
pop YL
pop ZH
pop ZL
pop r16

ret


/*************************************************************************************
* Name:         POINTER_INIT
* Purpose:      Initialize X pointer for SRAM read-back and Y pointer for I/O output.
* Inputs:       None
* Outputs:      None
* Affected:     CPU_RAMPX, CPU_RAMPY
*************************************************************************************/

POINTER_INIT:
;push registers
```

```
push r16
;set X pointer to first SRAM byte (read-back)
 ldi  XL, byte1(SRAM_START_ADDR)
    ldi  XH, byte2(SRAM_START_ADDR)
    ldi  r16, byte3(SRAM_START_ADDR)
    sts  CPU_RAMPX, r16
;set Y pointer to I/O port address (0xBEE640)
    ldi  YL, byte1(IO_START_ADDR)
    ldi  YH, byte2(IO_START_ADDR)
    ldi  r16, byte3(IO_START_ADDR)
    sts  CPU_RAMPY, r16
;pop registers
pop r16
Ret




/*********************************************************************************
* Name:         TCC0_OVF_ISR
* Purpose:      On each TCC0 overflow (~300 ms), read next byte from external SRAM
*               and write it to the LED output port. Resets pointer when reaching
*               the end of data.
* Inputs:       Data from SRAM via pointer X.
* Outputs:      LED data via pointer Y.
* Affected:     TCC0_INTFLAGS, PORTC_OUT, CPU_RAMPX
*********************************************************************************/

TCC0_OVF_ISR:
;preserve status register
push r16
in r16, CPU_SREG
push r16

;clear overflow flag
ldi r16, TC0_OVFIF_bm
sts TCC0_INTFLAGS, r16

  ldi  r16, (1<<0)
  sts  PORTC_OUTTGL, r16

;load byte from SRAM (LD X+)
ld r16, X+
;write byte to I/O port (ST Y)
st Y, r16
;if end of data reached, reset X pointer to first byte
ldi  r16, byte1(SRAM_END_ADDR)
cp      XL, r16
ldi  r16, byte2(SRAM_END_ADDR)
cpc  XH, r16

brne ISR_DONE
; reset X pointer
    ldi  XL, byte1(SRAM_START_ADDR)
    ldi  XH, byte2(SRAM_START_ADDR)
    ldi  r16, byte3(SRAM_START_ADDR)
    sts  CPU_RAMPX, r16

ISR_DONE:
;restore status register
pop r16
```

```
out CPU_SREG, r16
pop r16


Reti
```

Lab4_3b.asm

```
/*
 * lab4_3b.asm
 *
 *  Created: 10/12/2025 5:07:34 PM
 *   Author: arist
 */

 ;===============================================================================
; Lab 4, Section 22
; Name: Arion Stern
; Class #: 11303
; PI Name: Ian Santamauro
; Description: Continuously perform EBI write and read cycles to two SRAM
;              addresses to observe complete bus activity (ALE, CS, WE, RE)
;              on the logic analyzer. No timers or interrupts are used.
;===============================================================================


;Lab4_3b.asm:

;assembler directives
;equates
  .equ SRAM_START_ADDR = 0xA68000
  .equ IO_START_ADDR = 0xBEE640
  .equ ADDR0 = 0xA68010      ; first SRAM address
  .equ ADDR1 = 0xA68120      ; second SRAM address (forces ALE toggle)
  .equ DATA0 = 0x0A
  .equ DATA1 = 0x05

.org 0
Rjmp main



.org 0x100
MAIN:
;init stack
    ldi r16, low(0x3FFF)
    out CPU_SPL, r16
    ldi r16, high(0x3FFF)
    out CPU_SPH, r16


;initialize EBI I/O pins and control system
    rcall EBI_IO_INIT
    rcall EBI_INIT
```

```
;continuously perform alternating write and read cycles to two SRAM addresses
;used to observe EBI signals on LSA
LOOP:
;load X pointer with first SRAM address (ADDR0)
    ldi Xl, low(ADDR0)
    ldi Xh, high(ADDR0)
    ldi r16, byte3(ADDR0)
    sts CPU_RAMPX, r16

       ;write 0x0A to ADDR0 and read it back
    ldi r17, 0x0F
    st X, r17
    ld r18, X

       ;load X pointer with second SRAM address (ADDR1)
    ldi Xl, low(ADDR1)
    ldi Xh, high(ADDR1)
    ldi r16, byte3(ADDR1)
    sts CPU_RAMPX, r16
       ;write 0x05 to ADDR1 and read it back
    ldi r17, 0x05
    st X, r17
    ld r18, X

       ;repeat
    rjmp LOOP








/*************************************************************************************
* Name:        EBI_IO_INIT
* Purpose:     Configure Port H and Port K pins for EBI control and address signals.
* Inputs:      None
* Outputs:     None
* Affected:    PORTH_DIRSET, PORTH_OUTSET, PORTH_OUTCLR, PORTK_DIRSET
*************************************************************************************/

EBI_IO_INIT:

  ; Symbols for start of relevant memory address ranges.

  ; Preserve the relevant register.
push r16


  ; Initialize the relevant EBI control signals to be
  ; in a `false` state.
  ldi r16, 0b01010011

  sts PORTH_OUTSET, r16

  ldi r16, 0b00000100

  sts PORTH_OUTCLR, r16
  ; Initialize the EBI control signals to be output from
  ; the microcontroller.
  ldi r16, 0b01010111

  sts PORTH_DIRSET, r16

  ; Initialize the address signals to be output
  ; from the microcontroller.
  ldi r16, 0xFF

  sts PORTK_DIRSET, r16

  ; Recover the relevant register.
  pop r16


  ; Return from subroutine.
  ret

/*************************************************************************************
* Name:        EBI_INIT
* Purpose:     Initialize the External Bus Interface (EBI) for SRAM 3-PORT ALE1 mode.
* Inputs:      None
* Outputs:     None
* Affected:    EBI_CTRL, EBI_CS0_CTRLA, EBI_CS0_BASEADDR,
*              EBI_CS2_CTRLA, EBI_CS2_BASEADDR
*************************************************************************************/

EBI_INIT:

  ; Preserve the relevant register. (See 3 above)
  push r16

  ; Initialize the EBI system for SRAM 3-PORT ALE1 mode.
  ldi r16, 0b00000001

  sts EBI_CTRL, r16

  ; Initialize the relevant chip select(s).
  ; The following dropdown boxes contain a blank
   ; (i.e., empty) option; select this option if
   ; no other option applies.

  ; Configure chip select CS0, only if necessary.
  ldi r16, 0b00011101

  sts EBI_CS0_CTRLA, r16

  ldi r16, byte2(SRAM_START_ADDR)

sts EBI_CS0_BASEADDR, r16
ldi r16, byte3(SRAM_START_ADDR)

sts EBI_CS0_BASEADDR+1, r16

  ; Configure chip select CS2, only if necessary.
  ldi r16, 0b00001101

  sts EBI_CS2_CTRLA, r16
  ldi r16, byte2(IO_START_ADDR)

  sts EBI_CS2_BASEADDR, r16

 ldi r16, byte3(IO_START_ADDR)

  sts EBI_CS2_BASEADDR+1, r16




  ; Recover the relevant register.
  pop r16


  ; Return from subroutine.
  ret
```

APPENDIX

Supporting ASM/C Code and Additional Information

- Included/Referenced Files and Headers:
    - ATxmega128A1Udef.inc
    - Sram_data_asm.txt
    - Used code from the Lab 4 pre-lab quiz
- Additional Screenshots/Images:

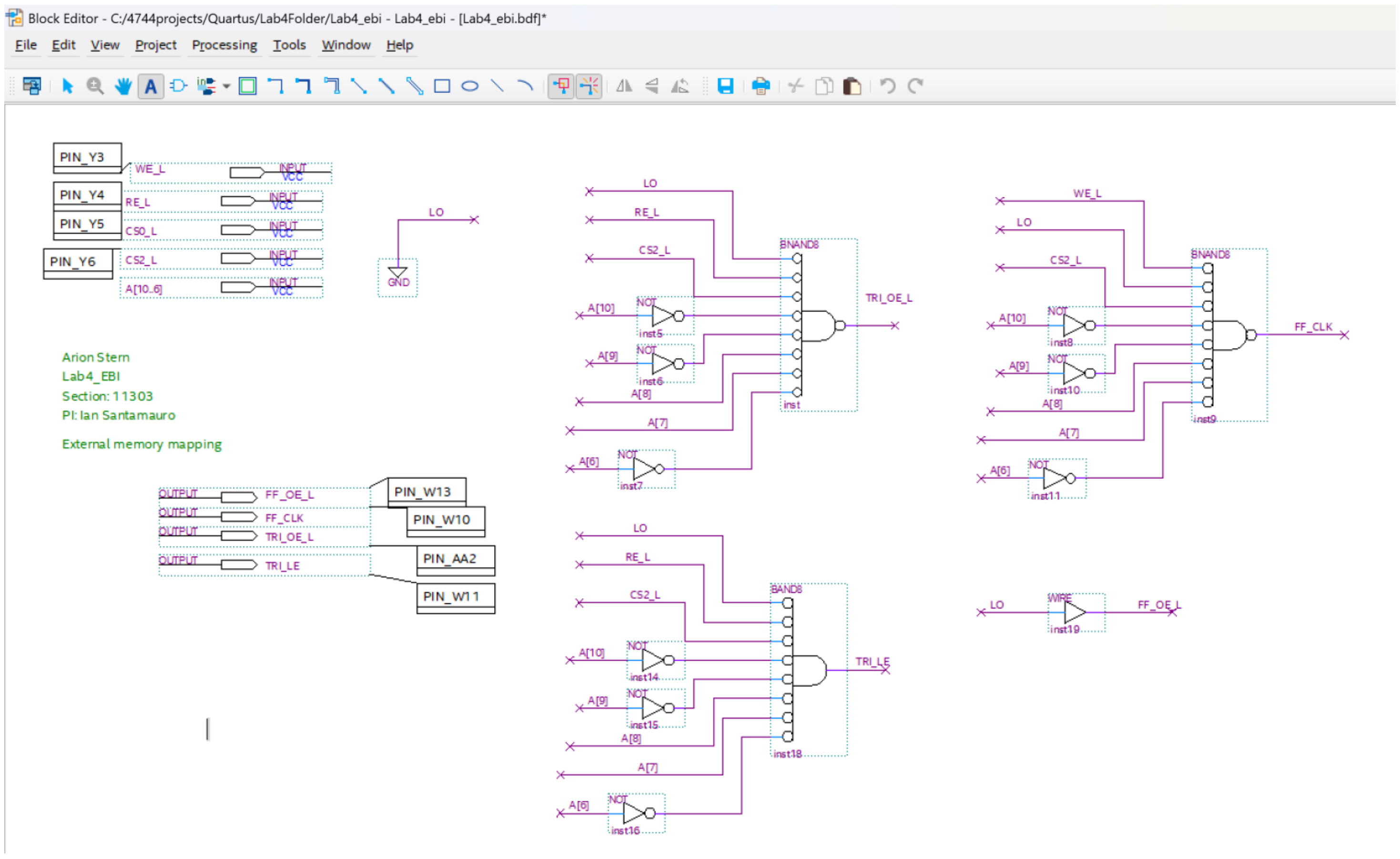


Figure 1: PLD Address Decoder Block Diagram (BDF)

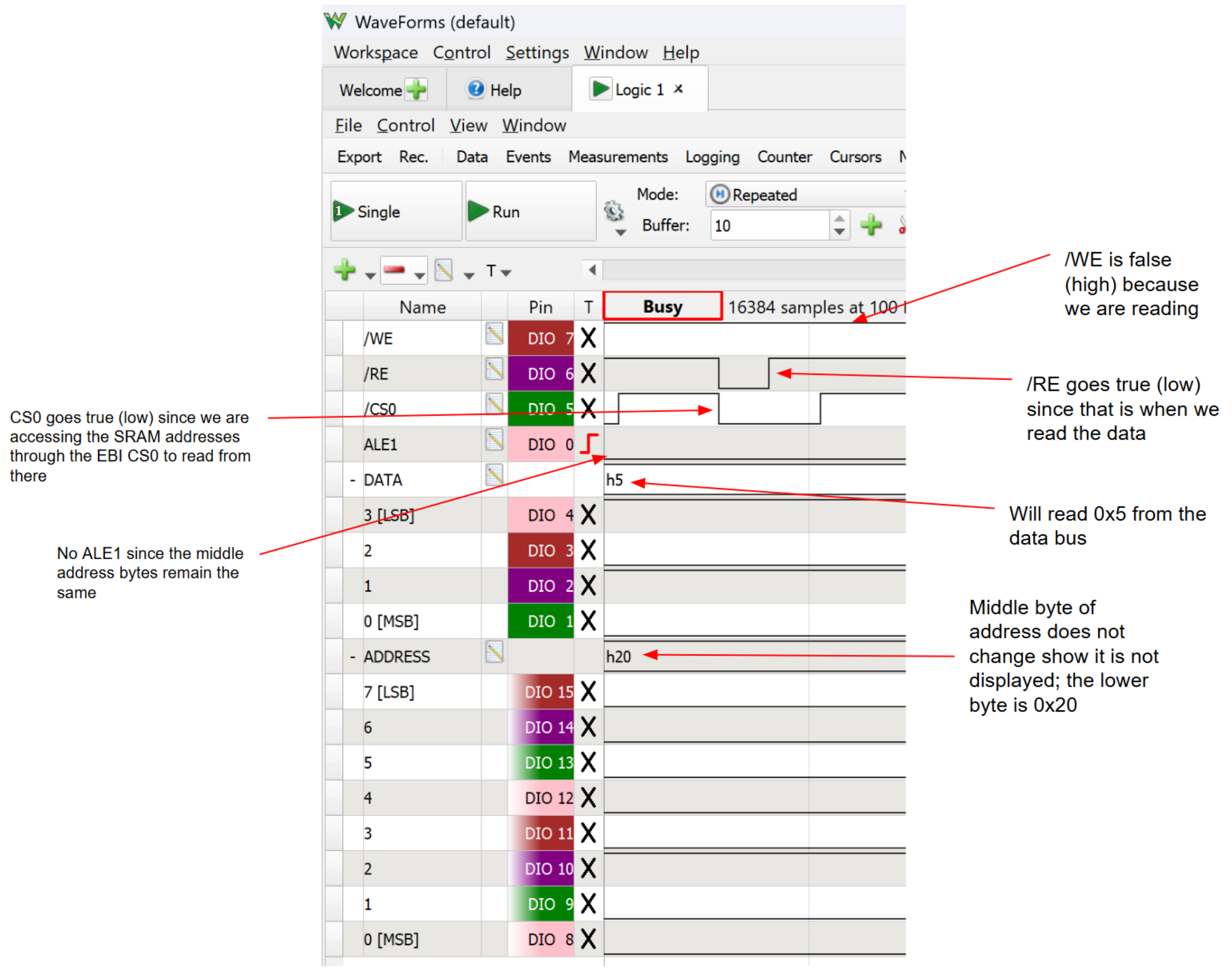
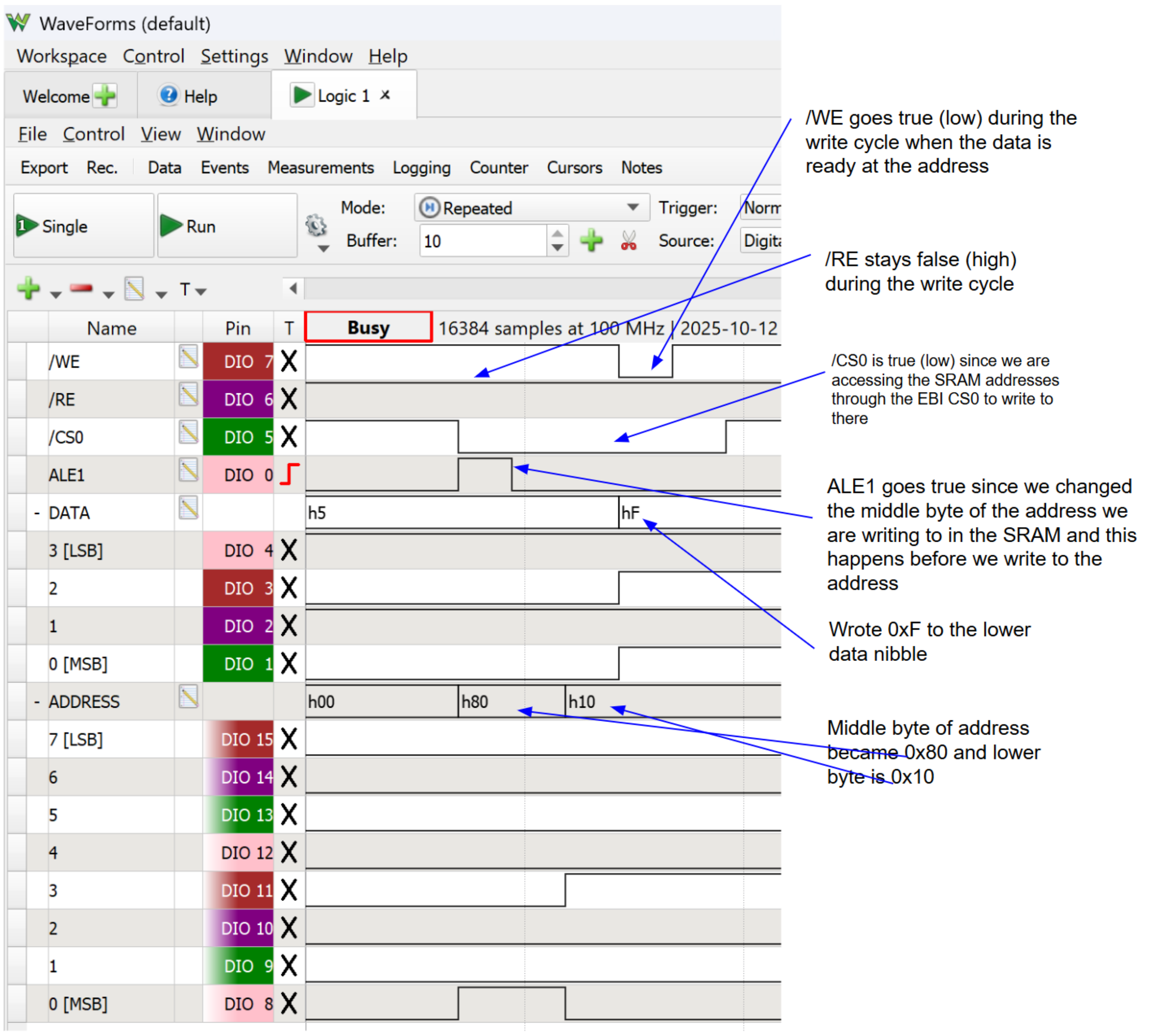

WaveForms (default)
Workspace Control Settings Window Help
Welcome
Help
Logic 1
File Control View Window
Export Rec. Data Events Measurements Logging Counter Cursors
Single
Run
Mode: Repeated
Buffer: 10
Name
Pin
T
Busy
16384 samples at 100
/WE DIO 7
/RE DIO 6
/CS0 DIO 5
ALE1 DIO 0
DATA h5
3 [LSB] DIO 4
2 DIO 3
1 DIO 2
0 [MSB] DIO 1
ADDRESS h20
7 [LSB] DIO 15
6 DIO 14
5 DIO 13
4 DIO 12
3 DIO 11
2 DIO 10
1 DIO 9
0 [MSB] DIO 8
/WE is false (high) because we are reading
/RE goes true (low) since that is when we read the data
CS0 goes true (low) since we are accessing the SRAM addresses through the EBI CS0 to read from there
Will read 0x5 from the data bus
No ALE1 since the middle address bytes remain the same
Middle byte of address does not change show it is not displayed; the lower byte is 0x20

Figure 2: Annotated EBI Read Cycle Waveform



/WE goes true (low) during the write cycle when the data is ready at the address

/RE stays false (high) during the write cycle

/CS0 is true (low) since we are accessing the SRAM addresses through the EBI CS0 to write to there

ALE1 goes true since we changed the middle byte of the address we are writing to in the SRAM and this happens before we write to the address

Wrote 0xF to the lower data nibble

Middle byte of address became 0x80 and lower byte is 0x10

Figure 3: Annotated EBI Write Cycle Waveform

Information Regarding my Connections:

MicroP to DE10:

INPUTS:

We to y3

Re to y4

Cs0 to y5

Cs2 to y6

A6 to aa15

A7 to v5

A8 to w7

A9 to w8

A10 to w9

OUTPUTS:

Latch OE to aa2

LE to w11

Flip flop OE to w13

Clk to w10

DAD CONNECTIONS:

We - 7

Re - 6

Cs0 - 5

D0 - 4

D1 - 3

D2 - 2

D3 - 1

A0 - 15

A1 - 14

A2 - 13

A3 - 12

A4 - 11

A5 - 10

A6 - 9

A7 - 8

Ale1 - 0