EEL3744

Today’s Menu

- Hardware Interfacing Concepts
- Instruction Cycle, Machine Transfers, and E-Cycles
  > Fetch, Decode, & Execute
  > Memory Read & Write
- Address and Data Bus Timing

See examples on web-site: `Input_Port.asm`

EEL3744

Instruction Cycle, Machine Transfers, and E-Cycles

- An instruction cycle consists of a sequence of machine transfers. A machine transfer corresponds to a single machine operation (a single E-clock pulse) on the external buses.
- Each machine transfer is one of the following types:
  1. Opcode Fetch 2. Memory Read
  3. Memory Write 4. Execution - No Transfer.
- **6811 (non-pipelined) Example:** Each machine transfer requires 1 E-clock cycle
  > The corresponding time depends on the frequency of the internal clock
  > If operating with an 8-MHz crystal, the fundamental E-clock frequency is 2 MHz
    – E-clock pulse requires 500 ns (1 / 2 MHz)
    – To execute a 4 cycle instruction (e.g., STAA with extended addressing) takes 4 × 500 ns = 2 μs.

EEL3744 **6811** Hardware Interfacing Concepts

- At the beginning of each machine transfer, the E-clock pulse goes from low to high, to indicate the beginning of a cycle
  - > After the operating mode has been selected using the MODA pin input immediately after RESET, the MODA/LIR (Load Instruction Register) open drain output indicates the execution of an opcode fetch machine transfer
  - > **LIR is low for first E-cycle of each new instruction.**
- Each type of machine transfer can be identified using a combination of these signals, as shown in the simplified table below

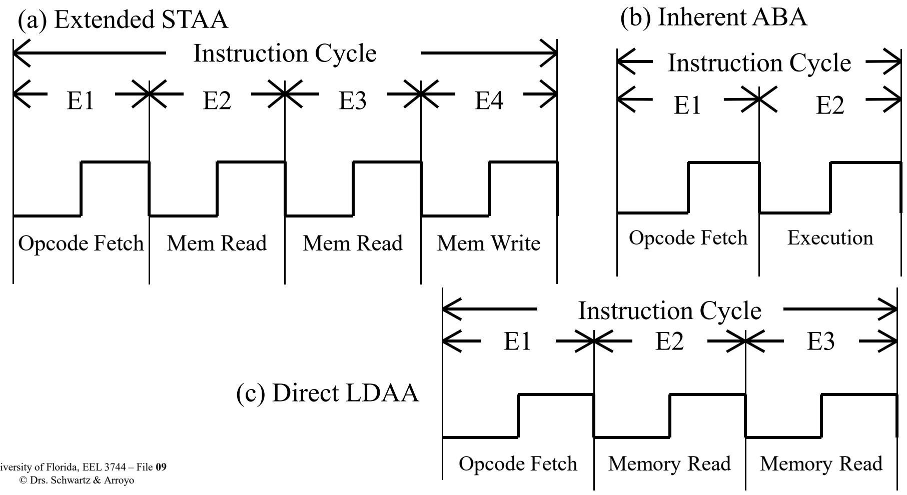
68HC11 Machine Transfer Chart

Machine Transfer	LIR(L)	R/~W
Opcode Fetch	Low	High
Memory Read	High	High
Memory Write	High	Low

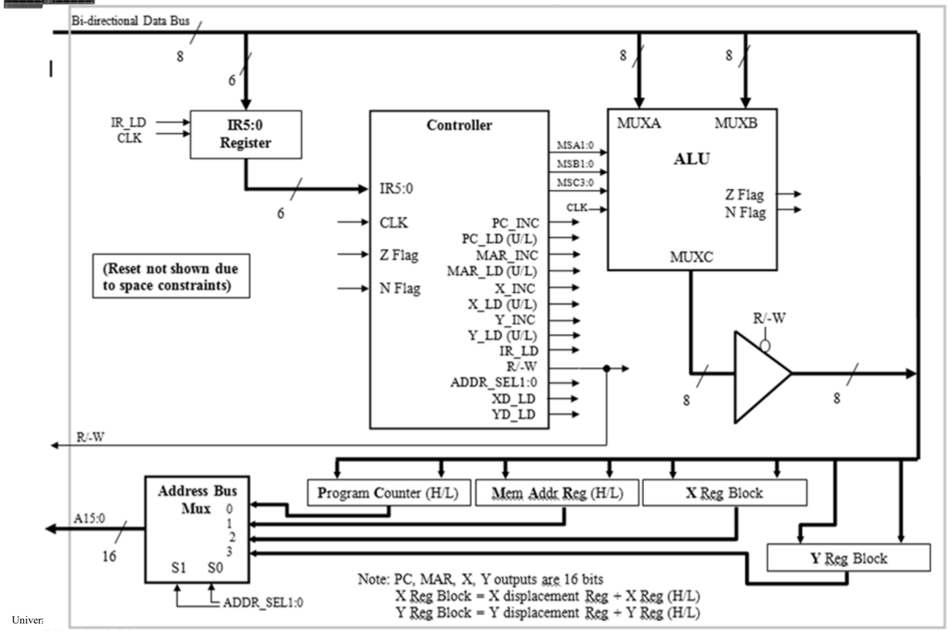
$\overline{\text{LIR}}$=LIR(L)

EEL3744 **6811** Hardware Interfacing Concepts

- Instruction cycles, machine transfers, and E-cycles for instructions (see **6811 Reference Manual**)

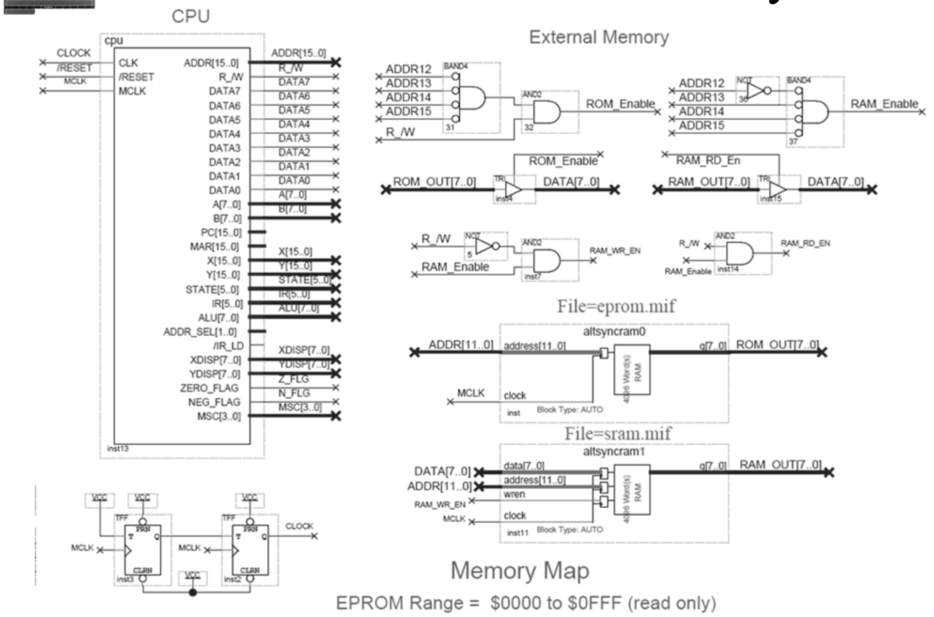


EEL3744 **GCPU** Block Diagram



Note: PC, MAR, X, Y outputs are 16 bits
X Reg Block = X displacement Reg + X Reg (H/L)
Y Reg Block = Y displacement Reg + Y Reg (H/L)

EEL3744 **GCPU** with Memory & I/O



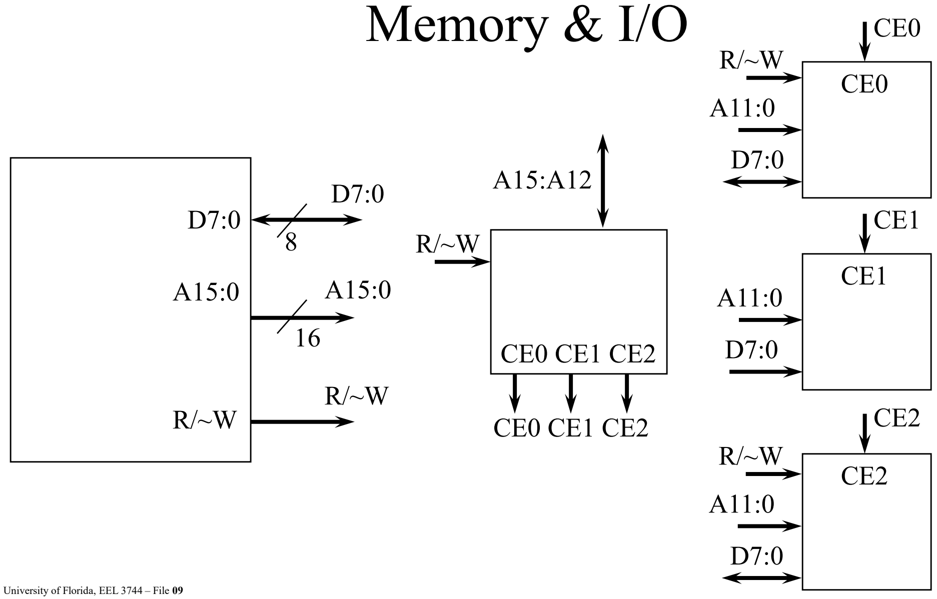
Memory Map

EPROM Range = $0000 to $0FFF (read only)

SRAM Range = $1000 to $1FFF (read/write)

EEL3744

GCPU with Memory & I/O



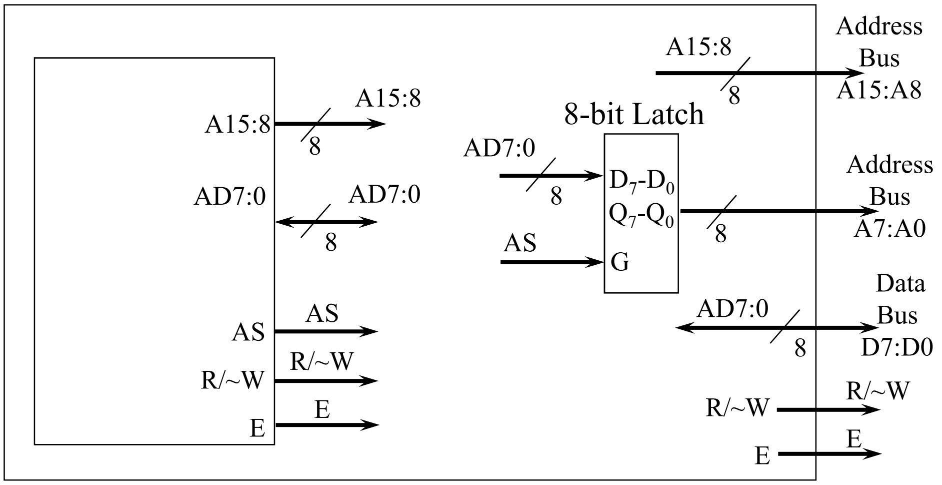
EEL3744

6811 Address/Data Buses

- 16-bit address bus
  > A15-A8
  > AD7-AD0
- 8-bit data bus
  > AD7-AD0
- The address/data bus AD7-AD0 is **time-multiplexed**
  > When AS (address strobe) is true (high), AD7-AD0 are address lines
  > When the E clock timing signal is high, AD7-AD0 are data lines

EEL3744

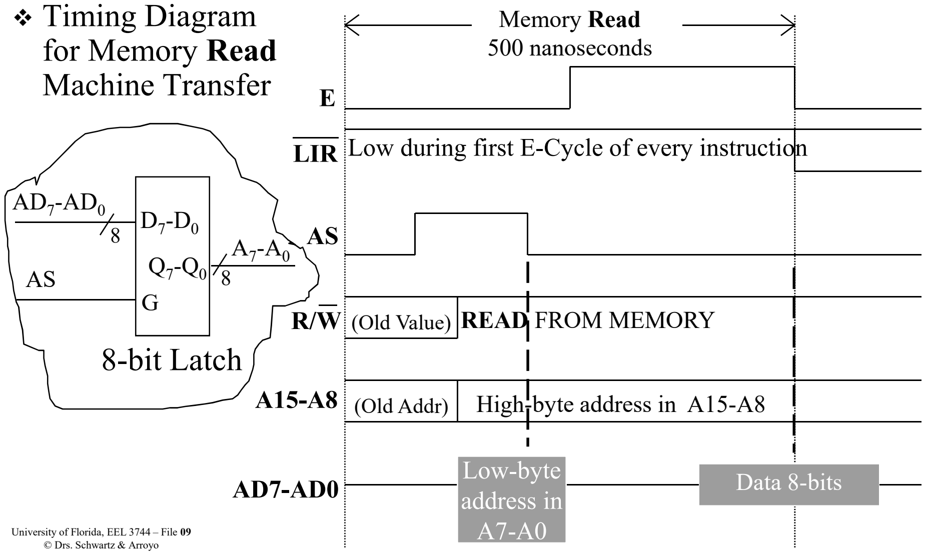
6811 Expanded Mode Implementation



EEL3744

6811 Hardware Interfacing Concepts

- Timing Diagram for Memory **Read** Machine Transfer



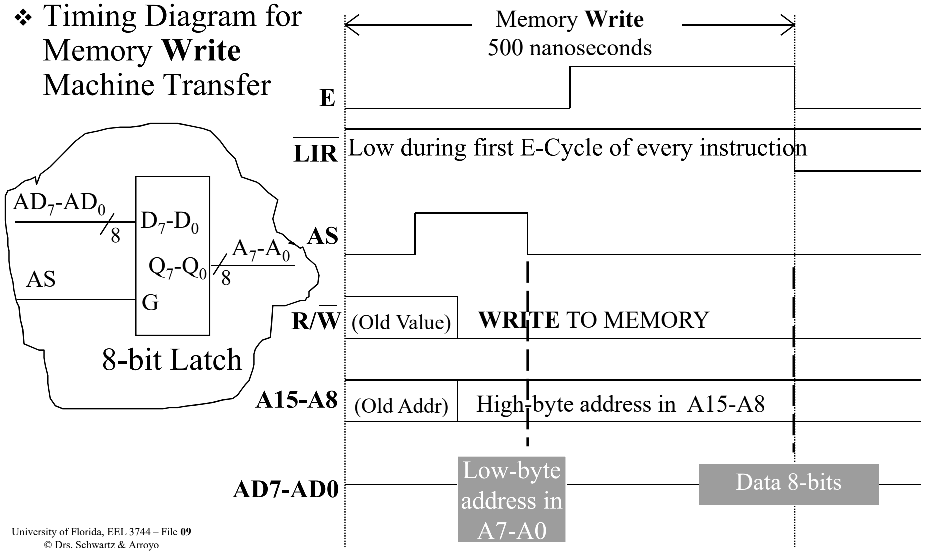
EEL3744 **6811** Steps for Memory Read Operation (describes picture)

1. The high-order 8 bits of the 16-bit address are placed onto A_{15}-A_8 and the low order 8 bits onto AD_7-AD_0.
2. The signal AS (Address Strobe or Stable) is pulsed during the first half of the corresponding E-cycle. This signal causes the low-order byte of the address to be latched at the **trailing** edge of the AS pulse.
3. With the low-order 8-bits of the address saved in the 8-bit latch, the AD_7-AD_0 lines are now free to be used as a data bus. Therefore, the μP relinquishes control of the data bus so that the memory module can place the subsequently retrieved 8-bit data onto the data bus.
4. At this point the signal R/W’ should be high and remain high for the duration of the E-clock pulse.
5. After an appropriate delay (small enough to guarantee that the data is on the data bus prior to E-clock pulse going from H to L again), the data (or opcode) is available on the data bus (and on AD_7-AD_0).
6. The control unit places the data into the appropriate internal register.

EEL3744

6811 Hardware Interfacing Concepts

- Timing Diagram for Memory **Write** Machine Transfer

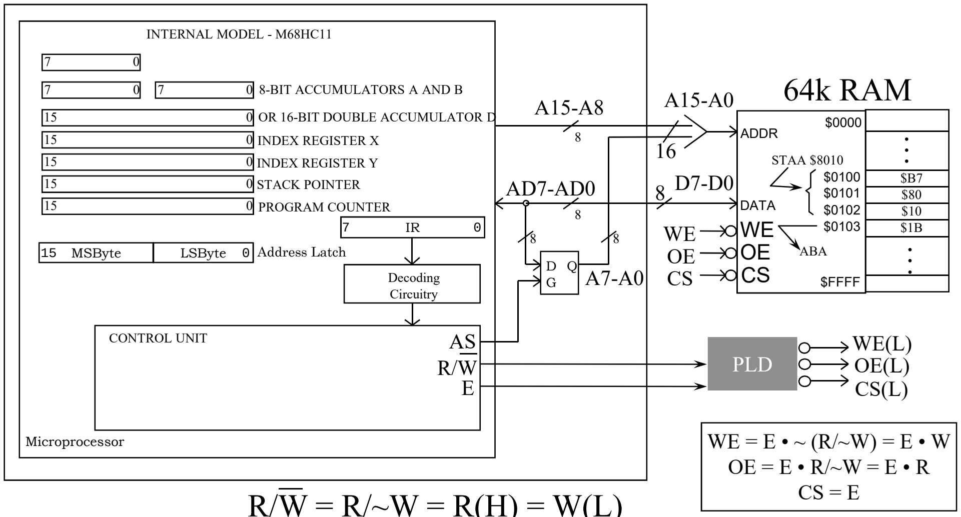


EEL3744 **6811** Steps for Memory Write Operation (describes picture)

1. The high-order 8 bits of the 16-bit address are placed onto A_{15}-A_8 and the low order 8 bits onto AD_7-AD_0.
2. The signal AS (Address Strobe or Stable) is pulsed during the first half of the corresponding E-cycle, indicating the address/data bus contains the low-order byte of the address. This signal causes the low-order byte of the address to be latched **at the trailing edge** of the AS pulse. Consequently, **the 16-bit address is now available** on the 16-bit address bus.
3. With the low-order 8-bits of the address saved in the 8-bit latch, **the AD_7-AD_0 lines are now free to be used as a data bus**. Therefore, the µP will place the data to be transferred onto the data bus during the second half of the corresponding E-clock pulse.
4. At this point the signal R/W’ should be low and remain low for the duration of the E-clock pulse.

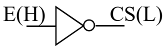
EEL3744 **6811** Hardware Interfacing Concepts

❖ **Example**: µP system with **64K** memory module



WE = E • ~ (R/~W) = E • W
OE = E • R/~W = E • R
CS = E

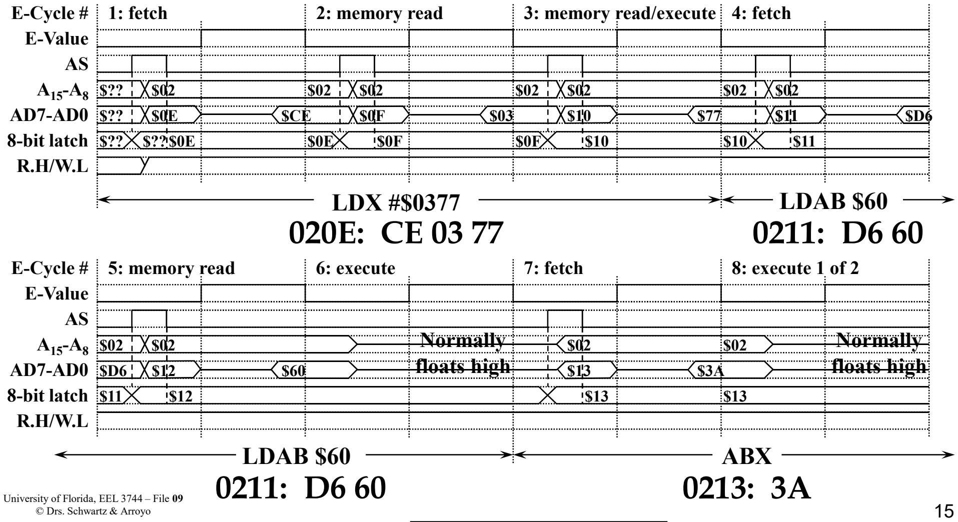
$R/\overline{W}$ = R/~W = R(H) = W(L)



Active-High: E, R;   Active-Low: W, WE, OE, CS

EEL3744 **6811** Fetch-Decode-Execute Example

- Suppose we execute the code starting at location $020E



EEL3744 **6811** Fetch-Decode-Execute Example (describes picture)

– Assume that the execution of the instruction in memory location $036F has just been completed and that the µP is ready to execute the next instruction (**STAA $8010**), the opcode of which is in memory location $0370.

$0370 STAA $8010

– In other words, the µP is ready to begin the fetch-decode-execute cycle for this instruction. At this time, the contents of the PC are $0370. The following sequence of machine transfers are required to fetch, decode, and execute this instruction.

Machine Cycle 1 (Opcode Fetch)

1. The high-order 8 bits, $03 (%0000 0011), of the PC are placed onto A_{15}-A_8, and the low-order 8 bits, $70 (%0111 0000), onto AD_7-AD_0.
2. AS is pulsed, causing $70 to be latched and become available on A_7-A_0.
3. The µP relinquishes control of the AD lines.

EEL3744 **6811** Fetch-Decode-Execute Example(describes picture)

4. The control unit causes R/W’ to become high for a read operation.
5. The E-clock pulse goes high. After a delay required by the memory module to output the contents of a memory location, the opcode $B7 (from location $0370) is available on the AD_7-AD_0.

STAA $8010

6. The control unit places the opcode $B7 into the IR.
7. The instruction is decoded and determined to be an STAA instruction. The control unit “knows” that the address of the destination location is stored in the next two locations in memory following the location containing the STAA opcode. Therefore, two additional memory read machine transfers are required. The PC now contains $0371.

Machine Cycle 2 (Memory Read)

1. The high-order 8 bits, $03 (%0000 0011), of the PC are placed onto A_{15}-A_8, and the low-order 8 bits, $71 (%0111 0001), onto AD_7-AD_0.
2. AS is pulsed, causing $71 to be latched and become available on A_7-A_0.

EEL3744 **6811** Fetch-Decode-Execute Example (describes picture)

$0370 STAA $8010

3. The μP relinquishes control of the AD lines.
4. The control unit causes R/W’ to become high for a read operation.
5. The E-clock pulse goes high. After a delay required by the memory module to output the contents of a memory location, the high-byte $80 (from location $0371) is available on the AD_7-AD_0.
6. The control unit places the $80 into the **high-order** byte of a 16-bit temporary effective address register. The PC is incremented.

Machine Cycle 3 (Memory Read)

The operations for this memory read machine transfer are similar to those of Machine Transfer 2. The only difference is that the contents of memory location $0372 are retrieved and stored in the **low-order** byte of the temporary effective address register, thereby completing the 16-bit destination address ($8010) of the STAA instruction. The PC is incremented.

EEL3744 **6811** Fetch-Decode-Execute Example (describes picture)

Machine Cycle 4 (Memory Write) **STAA $8010**

1. The high-order 8 bits, $80 (%1000 0000), of the temporary effective address register are placed onto A_{15}-A_8, and the low-order 8 bits, $10 (%0001 0000), onto AD_7-AD_0.
2. AS is pulsed, causing $10 to be latched and become available on A_7-A_0.
3. The contents ($57) of Register A are placed onto AD_7-AD_0, thereby making them available on D_7-D_0.
4. The control unit causes R/W’ to become low for a write operation.
5. After a delay required by the memory module to place data into a memory location, the contents of memory location $8010 are changed to $57, thereby completing the memory write machine transfer and the execution of the STAA instruction.

Done!

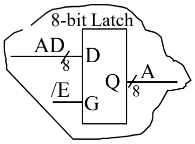
EEL3744 **6811** Fetch-Decode-Execute Example (describes picture)

- Below is the program (list file) from which the previous example was extracted

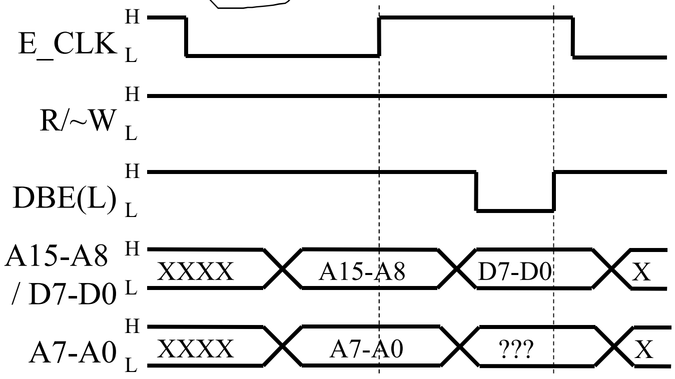
Mach_cyc.lst

```
0001  0060                    ORG   $0060
0002  0060 01                 DC.B  $01
0003
0004  0377                    ORG   $0377
0005  0377 8e 04 88           LDS   #$0488
0006
0007  020e                    ORG   $20E
0008  020e ce 03 77   MAIN:   LDX   #$0377
0009  0211 d6 60              LDAB  $60
0010  0213 3a                 ABX
0011  0214 a6 01              LDAA  1,X
0012  0216 20 fe      HERE:   BRA   HERE
```

EEL3744 **68HC12** Expanded Narrow-Mode **Read** Timing



See M68HC12B/D, section 19.16 for more info

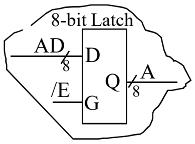


1. Addr setup time from E_CLK rise: 94ns (min)
2. Addr hold time E_CLK rise: 107ns (min)
3. Data setup from ECLK fall: 25ns (min)
4. Data hold time from E_CLK fall: 0ns (no need)
5. DBE delay time from E_CLK rise: 133ns (max)
6. DBE valid: 115ns (min)
7. DBE hold time to E_CLK fall: -10ns to 3ns

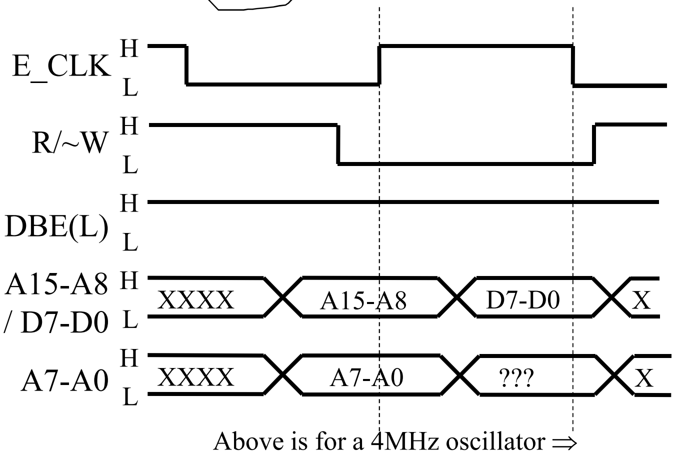
Above is for a 4MHz oscillator ⇒ f_{E_CLK} = 2MHz and T_{E_CLK} = 500ns

Q: When do you read?
A: DBE*R*f(Addr)

EEL3744 **68HC12** Expanded Narrow-Mode **Write** Timing



See M68HC12B/D, section 19.16 for more info



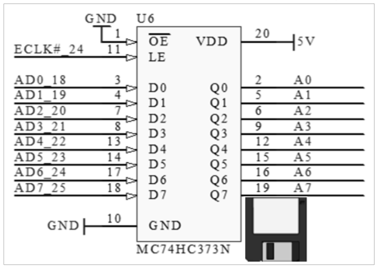
1. Addr setup time from E_CLK rise: 94ns (min)
2. Addr hold time E_CLK rise: 107ns (min)
3. Data setup from ECLK fall: 83ns (min)
4. Data hold time from E_CLK fall: 20ns (min)

Q: When do you write?
A: W*E*f(Addr)

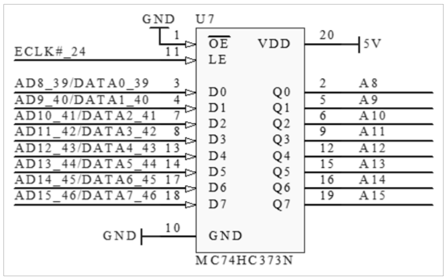
Above is for a 4MHz oscillator ⇒ f_{E_CLK} = 2MHz and T_{E_CLK} = 500ns

EEL3744 Latching the **68HC12** Address (Expanded Narrow-Mode)

- The AD15:AD8 pins of **Port A** are used for both A15:A8 (high 8-bits of the address) and D7:D0 (8-bit data)
  > AD15:AD8 are also know as D7:D0, although they function only as D7:D0 at specific times (see timing diagrams)
- The AD7:AD0 pins of **Port B** are used for A7:A0 (low 8-bits of the address) and **not** for data (in expanded narrow-mode)
- The 74‘373 on PortB (AD7-0) is **not** needed, in theory
  > But it **might** be needed in practice. This can be tested with an LSA.

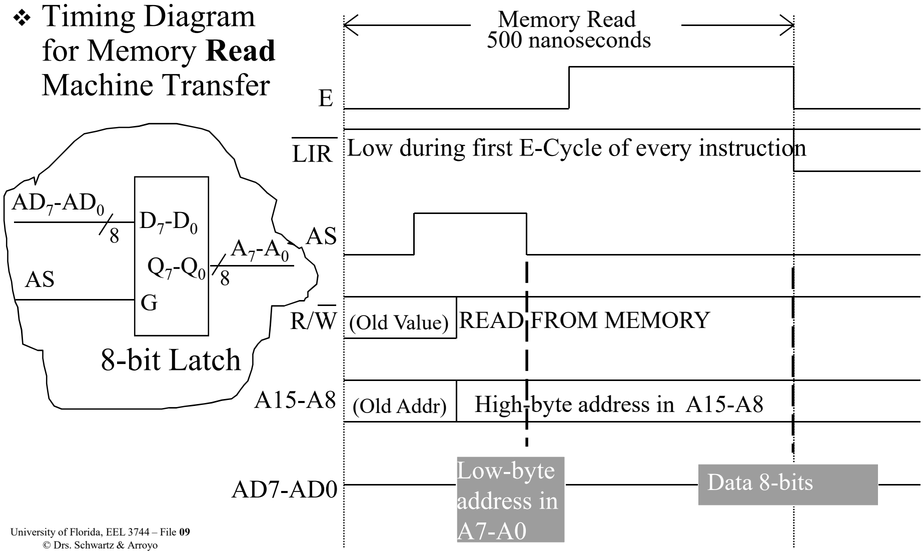


6812_Board_Manual


GND U7
1
OE
VDD
20
5V
ECLK#_24
11
LE
AD8_39/DATA0_39 3 D0 Q0 2 A8
AD9_40/DATA1_40 4 D1 Q1 5 A9
AD10_41/DATA2_41 7 D2 Q2 6 A10
AD11_42/DATA3_42 8 D3 Q3 9 A11
AD12_43/DATA4_43 13 D4 Q4 12 A12
AD13_44/DATA5_44 14 D5 Q5 15 A13
AD14_45/DATA6_45 17 D6 Q6 16 A14
AD15_46/DATA7_46 18 D7 Q7 19 A15
GND 10 GND
MC74HC373N

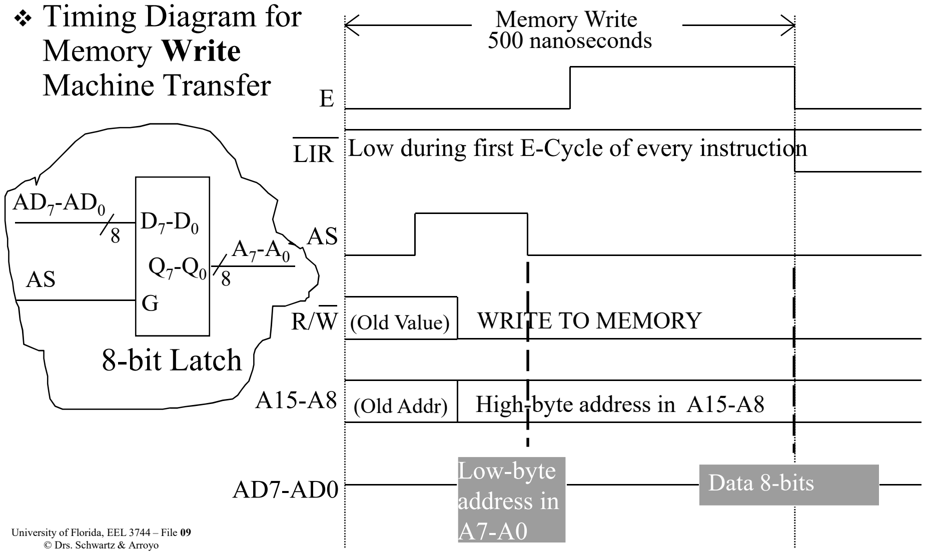
EEL3744 **68HC11** Expanded Mode **Read** Timing

❖ Timing Diagram for Memory **Read** Machine Transfer



EEL3744
68HC11 Expanded Mode **Write** Timing

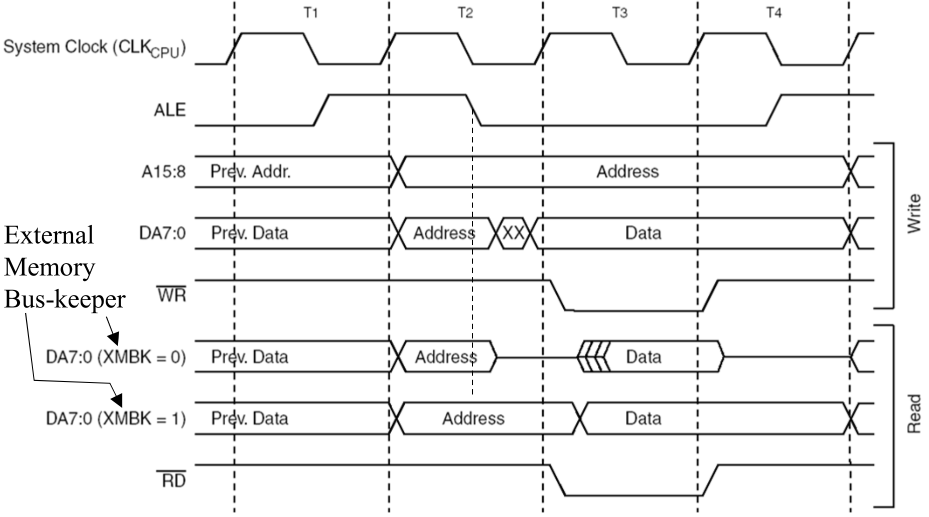
- Timing Diagram for Memory **Write** Machine Transfer



EEL3744 **Atmel AVR Mega8515L**
External Memory **R/W** Timing

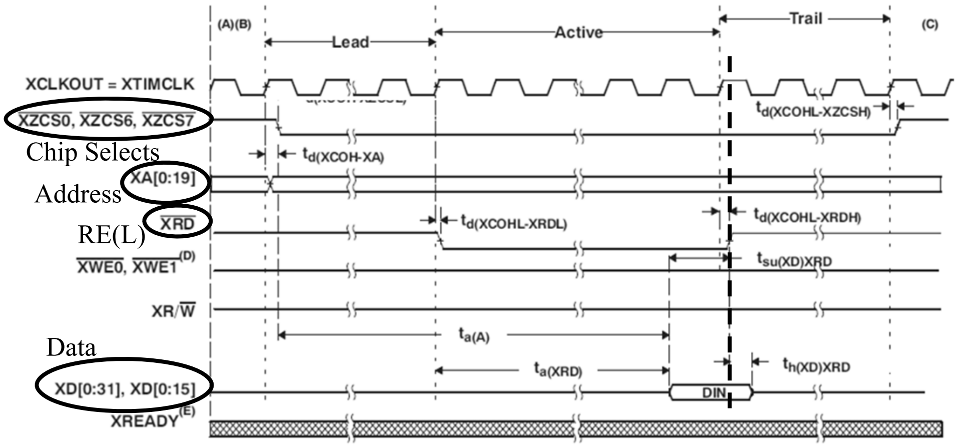
- Need same latch as 68HC11 (Atmel's ALE = 68HC12's AS)
- Atmel's CLK is inverse of Motorola's E-clock
- Atmel uses pipelining, i.e., T1-T4 are cycles in parallel with others

`Atmel_Mega8515L.pdf`



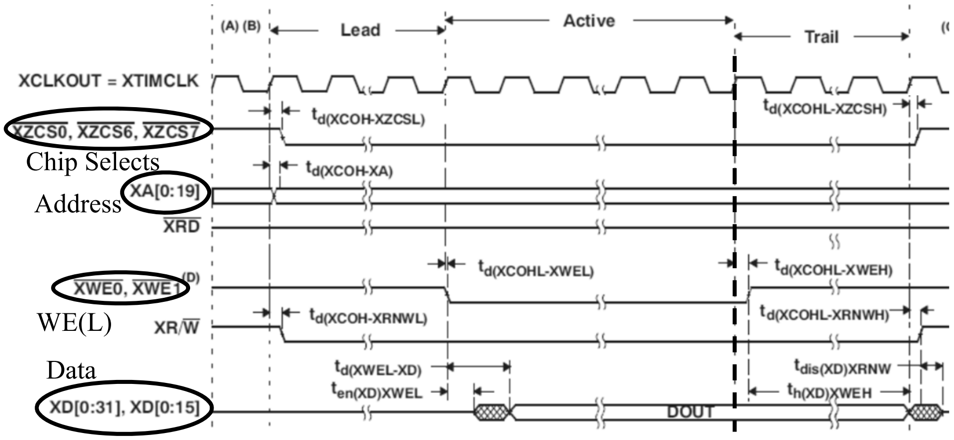
EEL3744 **DSC F2833** Generic **Read** Timing (Figure 6-23, TMS320F2833x Data Manual)

TMS320F2833x
Data Manual.pdf



EEL3744 **DSC F2833** Generic **Write** Timing (Figure 6-24, TMS320F2833x Data Manual)

TMS320F2833x
Data Manual.pdf



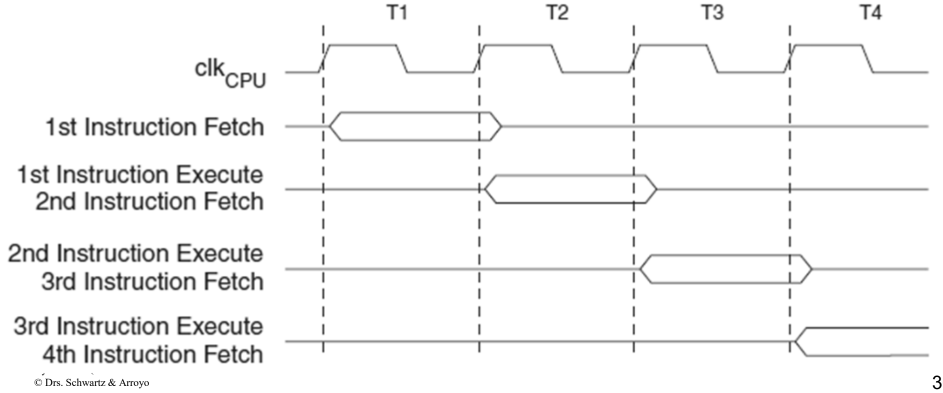
EEL3744 **XMEGA** Write followed by Read and Pipeline

- Be careful when having a **write followed by a read access**
  - > This has been documented to occur with the keypad when a read immediately follows a write
    - – If you can find documentation of this in the Atmel manuals, please tell me where you found it!
  - > **If this does occur,** add a **NOP** assembly instructions between a write and read instructions
    - – If one NOP does not work, add 2; if 2 does not work, add 3
      - ▪ For the XMEGA, one NOP should work
      - ▪ If you need more than one NOP, let me know!

EEL3744 **XMEGA** Pipeline: Execution Timing

- The parallel instruction fetches and instruction executions timing

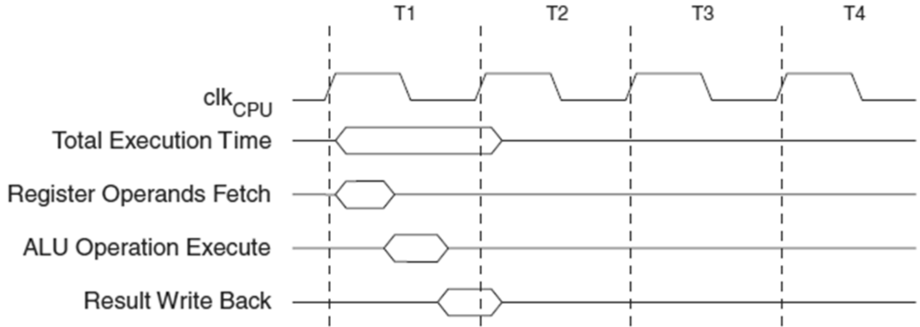
See doc8331, Figure 3-2



EEL3744 **XMEGA** Pipeline: Execution Timing

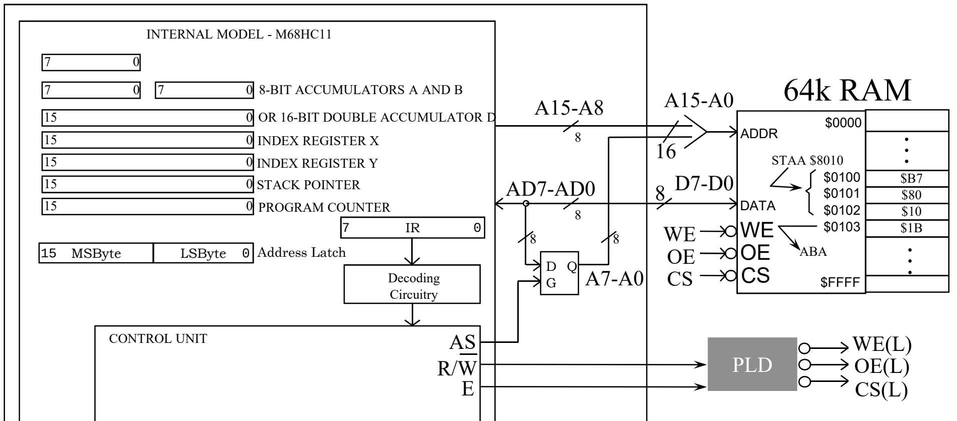
See doc8331, Figure 3-3

- Single cycle ALU instruction
  > In a single clock cycle, an ALU operation using two register operands is executed and the result is stored back to the destination register



EEL3744 **6811** Hardware Interfacing Concepts

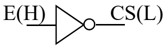
❖ **Example**: µP system with **64K** memory module



WE = E • ~ (R/~W) = E • W
OE = E • R/~W = E • R
CS = E

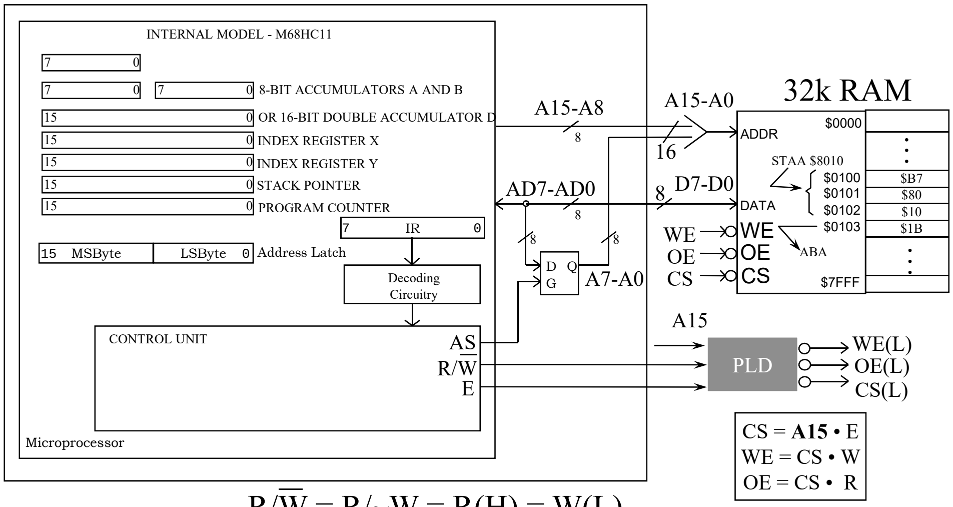
$R/\overline{W}$ = R/~W = R(H) = W(L)

Active-High: E, R;   Active-Low: W, WE, OE, CS



EEL3744 **6811** Hardware Interfacing Concepts (w/ Addr Decoding)

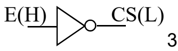
❖ **Example**: µP system with **32K** memory module at **$8000-$FFFF**



CS = **A15** • E
WE = CS • W
OE = CS • R

R/W̄ = R/~W = R(H) = W(L)

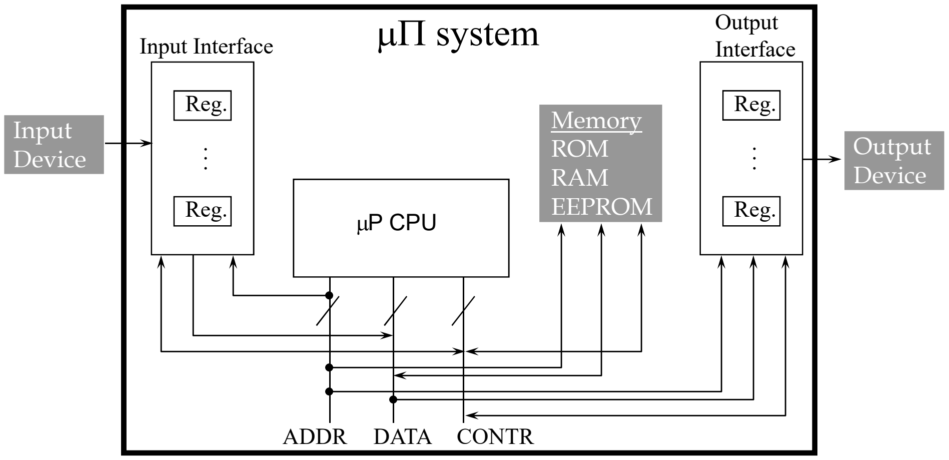
Active-High: E, R; Active-Low: W, WE, OE, CS



EEL3744

Hardware/Interfacing Review

- Block Diagram of a Basic Microprocessor System

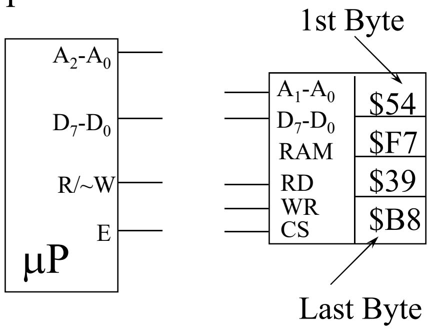


EEL3744 **68HC12** Interfacing Signals/Pins

- A 16-bit Address Bus **A_{15}-A_0**
  - > A_{15}-A_8 are available through PA_7-PA_0 in Expanded Mode
  - > A_7-A_0 are available through PB_7-PB_0 in Expanded Mode
  - > Address pins valid when E-clock is rising
- An 8-bit Data Bus **D_7-D_0**
  - > D_7-D_0 through PA_7-PA_0 in Expanded Narrow Mode when E=1
- A 4-bit Control Bus **E, DBE, R/~W, Reset**
  - > PE_2 becomes **R/~W** in Expanded Mode
  - > PE_7 becomes **DBE** (active-low data bus enable) in Expanded Mode
  - > PE_4 becomes **E** (event clock) in Expanded mode
    - – For 4MHz oscillator, E=2 MHz; i.e., 500ηs= 0.5µs Event Clk
  - > **Reset** (active-low) is always available
    - – Needed to protect external devices from corruption during the reset operation

EEL3744 **68HC11** Hardware/Interfacing

- Example R1: Add a 4×8 bit (4-bytes) RAM module to a hypothetical µP with 3 address pins, 8 data pins and control pins R/~W & E



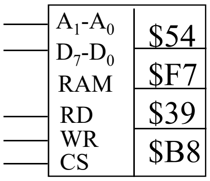
- D_7-D_0 on the µP connect to D_7-D_0 on the RAM
- RD = E • R/~W
- WR = E • (R/~W)’
- Two of the three address lines go to A1-A0 on the RAM; CS = A_i

1. CS=A_2; A1=A_1; A0=A_0
2. CS=/A_2; A1=A_1; A0=A_0
3. CS=A_1; A1=A_2; A0=A_0
4. CS=A_0; A1=A_2; A0=A_1

EEL3744

68HC11 Hardware/Interfacing

- What are the consequences of these choices?
  > Choice 1: **CS=A_2; A1=A_1; A0=A_0**
    - When the µP issues address 000; the RAM does not respond since **CS=A_2**=0; similarly for addresses 001, 010, 011
    - For address 100 the µP reads $54, for 101 the µP reads $F7, for 110 the µP reads $39, for 111 the µP reads $B8
    - The 4-byte RAM starts at address 100



1. CS=A_2; A1=A_1; A0=A_0
2. CS=/A_2; A1=A_1; A0=A_0
3. CS=A_1; A1=A_2; A0=A_0
4. CS=A_0; A1=A_2; A0=A_1

Addr	Choice 1	Choice 2	Choice 3	Choice 4
000	**None**	$54	None	None
001	**None**	$F7	None	$54
010	**None**	$39	$54	None
011	**None**	$B8	$F7	$F7
100	**$54**	None	None	None
101	**$F7**	None	None	$39
110	**$39**	None	$39	None
111	**$B8**	None	$B8	$B8

EEL3744

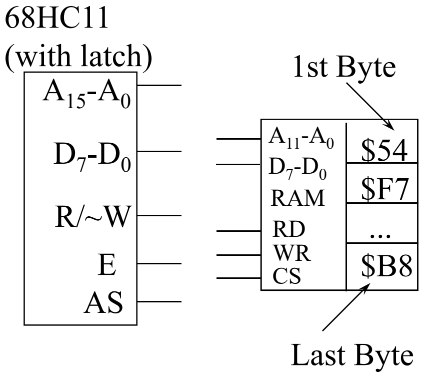
Conclusions from Example R1

- The data in the RAM will not be accessed contiguously unless we connect the matching contiguous low order lines to the RAM, i.e., A1=A_1 and A0=A_0
- We have a choice of CS=/A_2 or CS=A_2
  > If want the RAM in the "**low memory range,**" choose **CS**= /A_2
  > If want the RAM in the "**high memory range,**" choose **CS** = A_2
- For **contiguous access** we always connect the **low order** address pins to **all** the RAM address pins
- CS = *f* (**unused high order** address lines). If we have **m** unused address lines we will have **2^m** possible starting addresses for the contiguous memory block

EEL3744

68HC11
Hardware/Interfacing

- Example R2: Add a 4k×8 bit (4k-bytes, 4K) RAM module to the M68HC11



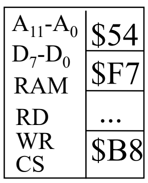
- D_7-D_0 on the µP connect to D_7-D_0 on the RAM
- RD = E • R/~W
- WR = E • (R/~W)’
- µP A_{11}-A_0 to A_{11}-A_0 on the RAM; CS=*f*(A_{15}-A_{12})

1. CS=/A_{15}•/A_{14}•/A_{13}•/A_{12}
2. CS=/A_{15}•/A_{14}•/A_{13}•A_{12}
3. CS=/A_{15}•/A_{14}•A_{13}•/A_{12}
4. CS=/A_{15}•/A_{14}•A_{13}•A_{12}

EEL3744
68HC11 Hardware/Interfacing Review

1. CS=/A_{15}•/A_{14}•/A_{13}•/A_{12}
2. CS=/A_{15}•/A_{14}•/A_{13}•A_{12}
3. CS=/A_{15}•/A_{14}•A_{13}•/A_{12}
4. CS=/A_{15}•/A_{14}•A_{13}•A_{12}

- What are the consequences of these choices?
  > Choice 1: CS=/A_{15}•/A_{14}•/A_{13}•/A_{12}
    – When the µP issues address 0000 xxxx xxxx xxxx; the RAM responds since CS=1•1•1•1 (address lines are active-high)
    – For address 0000 0000 0000 0000 the µP reads $54, for $0001 the µP reads $F7, for $0FFF the µP reads $B8
    – The 4K-byte RAM starts at address $0000



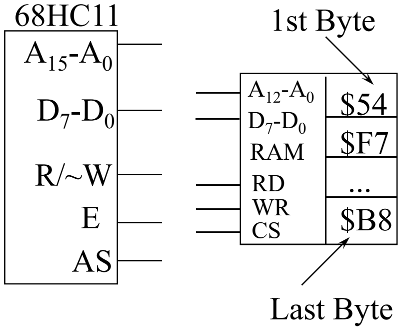
Address	C1	C2	C3	C4
0000 0000 0000 0000	**$54**	None	None	None
0000 0000 0000 0001	**$F7**	None	None	None
0001 0000 0000 0000	**None**	$54	None	None
0001 0000 0000 0001	**None**	$F7	None	None
0010 0000 0000 0000	**None**	None	$54	None
0010 0000 0000 0001	**None**	None	$F7	None
0011 0000 0000 0000	**None**	None	None	$54
0011 0000 0000 0001	**None**	None	None	$F7

EEL3744 **68HC11** Conclusions from Example R2

- For choice 1, $CS=/A_{15}\bullet/A_{14}\bullet/A_{13}\bullet/A_{12}$ the 4K memory block begins at address \$0000; the range is \$0000-\$0FFF
- For choice 2, $CS=/A_{15}\bullet/A_{14}\bullet/A_{13}\bullet A_{12}$ the 4K memory block begins at address \$1000; the range is \$1000-\$1FFF
- For choice 3, $CS=/A_{15}\bullet/A_{14}\bullet A_{13}\bullet/A_{12}$ the 4K memory block begins at address \$2000; the range is \$2000-\$2FFF
- For choice 4, $CS=/A_{15}\bullet/A_{14}\bullet A_{13}\bullet A_{12}$ the 4K memory block begins at address \$3000; the range is \$3000-\$3FFF
- There are **2^4=16** choices for a starting address for the 4K block, mainly, \$0000, \$1000, \$2000,..., \$E000, \$F000

EEL3744 **68HC11** Hardware/Interfacing

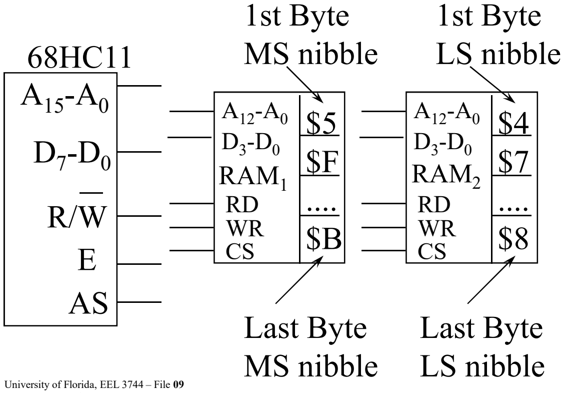
- Example R3: Add a 8k×8 bit (8k-bytes, 8K) RAM module to the M68HC11 starting at address \$4000



- D_7-D_0 on the μP connect to D_7-D_0 on the RAM
- RD = E • R/~W
- WR = E • (R/~W)’
- μP A_{12}-A_0 to A_{12}-A_0 on the RAM; CS=*f*(A_{15}-A_{13})
- Since starting address is **010**0 0000 0000 0000 the only choice is $CS=/A_{15}\bullet A_{14}\bullet/A_{13}$
- The address range is \$4000-\$5FFF

EEL3744 **68HC11** Hardware/Interfacing

- Example R4: Add a 8K of RAM using two 8k x 4 RAM chips to the M68HC11 starting at address \$A000



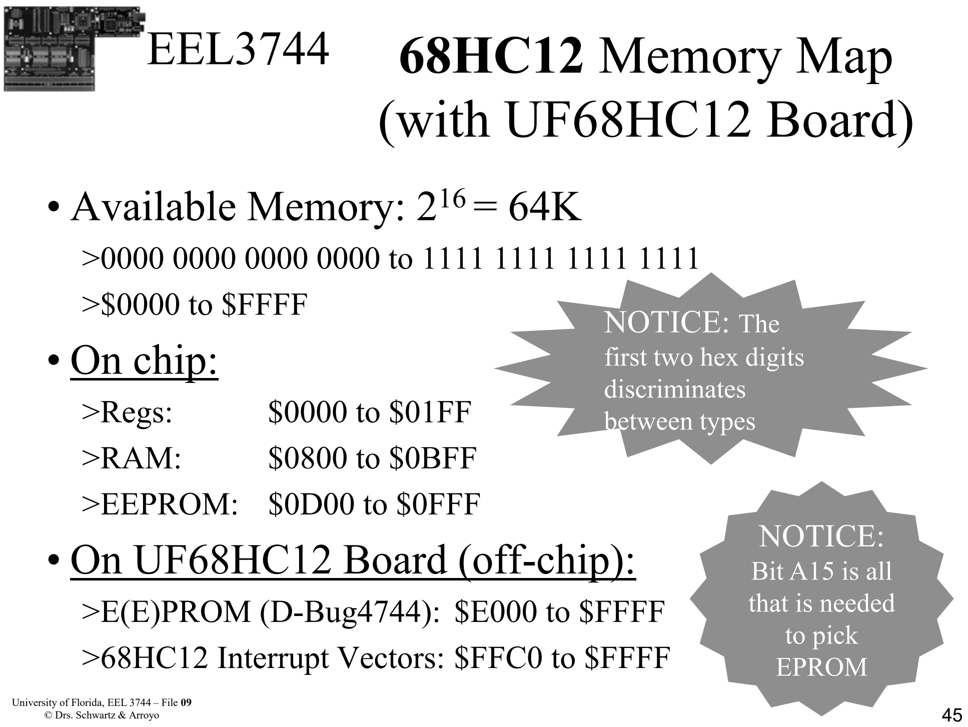
- D_3-D_0 on the µP to D_3-D_0 on RAM_2 (LS nibble)
- D_7-D_4 on the µP to D_3-D_0 on RAM_1 (MS nibble)
- WR = E • (R/~W)’
- RD = E • R/~W
- µP A_{12}-A_0 to A_{12}-A_0 on the RAM; CS=*f*(A_{15}-A_{13})
- For starting address **101**0 0000 0000 0000 the only choice is CS=A_{15}•/A_{14}•A_{13}
- The address range is \$A000-\$BFFF

EEL3744 **68HC11** Memory Map (with BUFFALO)

❖ Let us draw a memory map for the M68HC11E9 EVBU Board

START	END	TYPE
\$0000	\$01FF	RAM
\$0200	\$0FFF	{EMPTY}
\$1000	\$103F	Internal Registers
\$1040	\$B5FF	{EMPTY}
\$B600	\$B7FF	EEPROM
\$B800	\$CFFF	{EMPTY}
\$D000	\$FFFF	ROM {BUFFALO}

❖ We can place additional 4K memory modules at address \$2000, \$3000, \$4000, \$5000, \$6000, \$7000, \$8000, \$9000, \$A000, \$C000



EEL3744

68HC12 Memory Map (with UF68HC12 Board)

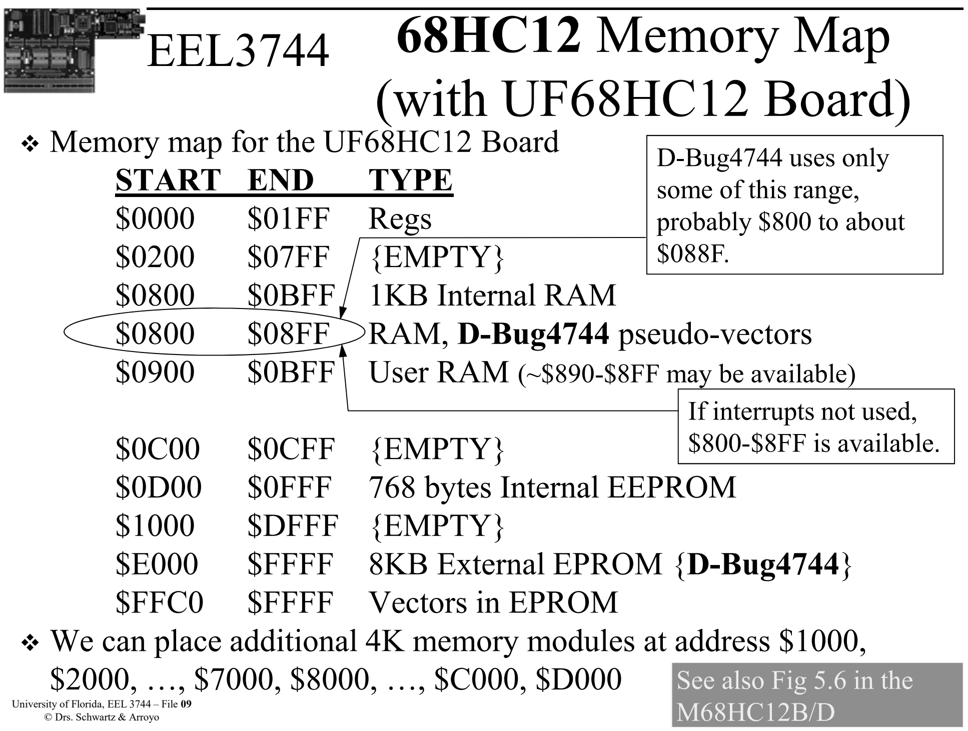
- Available Memory: 2^{16} = 64K
  - >0000 0000 0000 0000 to 1111 1111 1111 1111
  - >$0000 to $FFFF
- <u>On chip:</u>
  - >Regs: $0000 to $01FF
  - >RAM: $0800 to $0BFF
  - >EEPROM: $0D00 to $0FFF
- <u>On UF68HC12 Board (off-chip):</u>
  - >E(E)PROM (D-Bug4744): $E000 to $FFFF
  - >68HC12 Interrupt Vectors: $FFC0 to $FFFF

NOTICE: The first two hex digits discriminates between types

NOTICE: Bit A15 is all that is needed to pick EPROM

University of Florida, EEL 3744 – File **09**
© Drs. Schwartz & Arroyo

45



EEL3744

68HC12 Memory Map (with UF68HC12 Board)

❖ Memory map for the UF68HC12 Board

START	**END**	**TYPE**
$0000	$01FF	Regs
$0200	$07FF	{EMPTY}
$0800	$0BFF	1KB Internal RAM
$0800	$08FF	RAM, **D-Bug4744** pseudo-vectors
$0900	$0BFF	User RAM (~$890-$8FF may be available)
$0C00	$0CFF	{EMPTY}
$0D00	$0FFF	768 bytes Internal EEPROM
$1000	$DFFF	{EMPTY}
$E000	$FFFF	8KB External EPROM {**D-Bug4744**}
$FFC0	$FFFF	Vectors in EPROM

D-Bug4744 uses only some of this range, probably $800 to about $088F.

If interrupts not used, $800-$8FF is available.

❖ We can place additional 4K memory modules at address $1000, $2000, …, $7000, $8000, …, $C000, $D000

See also Fig 5.6 in the M68HC12B/D

University of Florida, EEL 3744 – File **09**
© Drs. Schwartz & Arroyo

EEL3744

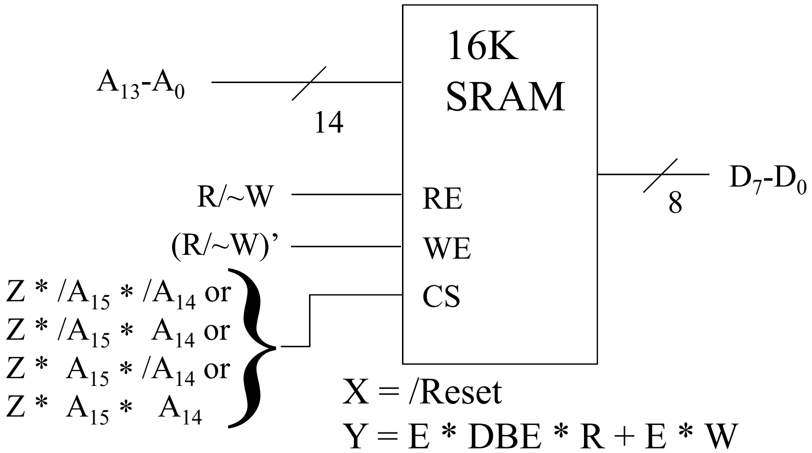
XMEGA **Data Memory** Map

See doc8385, Figure 7-2 CORRECTED

Addr (hex)	Description
0 – 0FFF	I/O Registers (4kB) [0 – BC3 on ours]
1000 – 17FF	EEPROM (2kB)
1800 – 1FFF	Reserved
2000 – 3FFF	Internal SRAM (8kB)
4000 – FF FFFF	External “Memory” (0 to ~16MB)

EEL3744

68HC12 16KB Memory Expansion



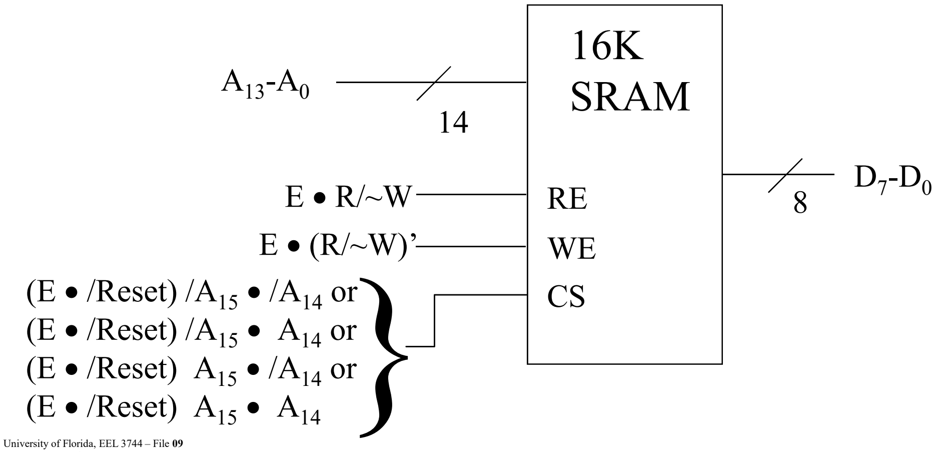
X = /Reset

Y = E * DBE * R + E * W

Z = X * Y

EEL3744

68HC11 16KB Memory Expansion



16K SRAM

A_{13}-A_0 14

E • R/~W RE

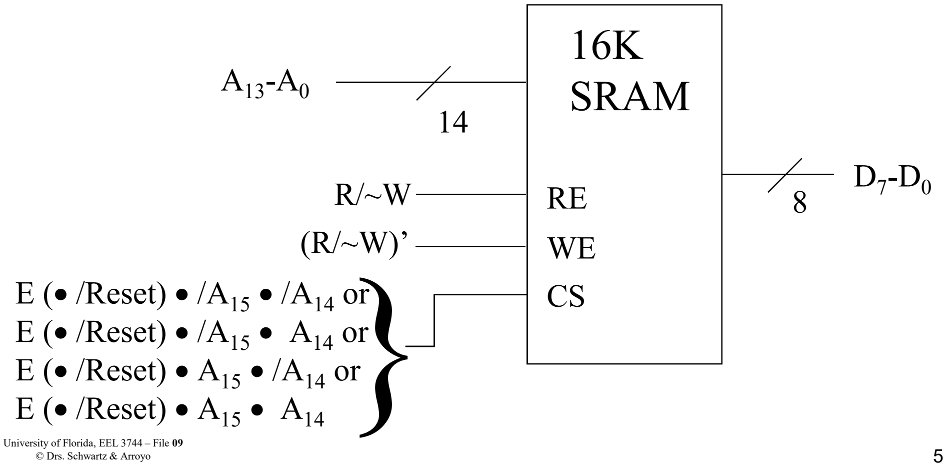
E • (R/~W)' WE

(E • /Reset) /A_{15} • /A_{14} or
(E • /Reset) /A_{15} • A_{14} or
(E • /Reset) A_{15} • /A_{14} or
(E • /Reset) A_{15} • A_{14}
CS

D_7-D_0 8

EEL3744

68HC11 16KB Memory Expansion



16K SRAM

A_{13}-A_0 14

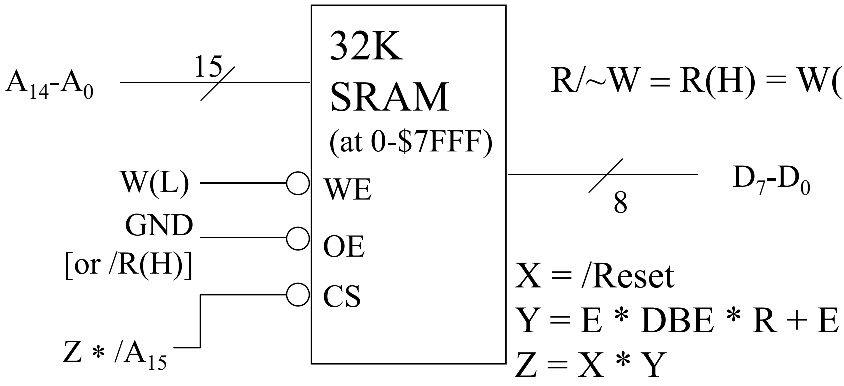
R/~W RE

(R/~W)' WE

E (• /Reset) • /A_{15} • /A_{14} or
E (• /Reset) • /A_{15} • A_{14} or
E (• /Reset) • A_{15} • /A_{14} or
E (• /Reset) • A_{15} • A_{14}
CS

D_7-D_0 8

EEL3744 **68HC12** 32KB Memory Expansion

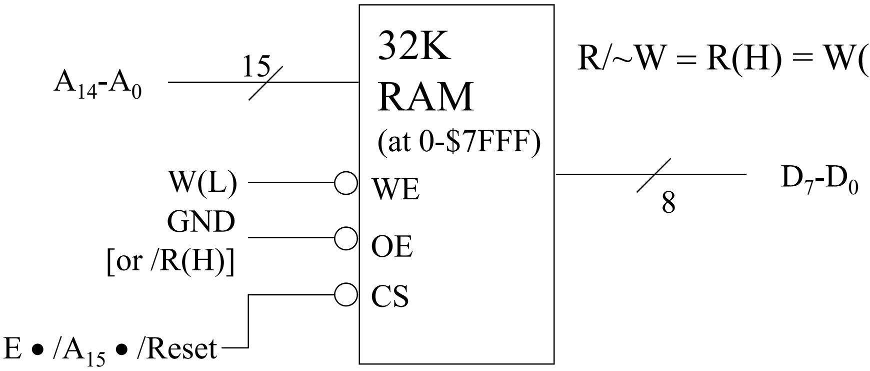


R/~W = R(H) = W(L)

X = /Reset
Y = E * DBE * R + E * W
Z = X * Y

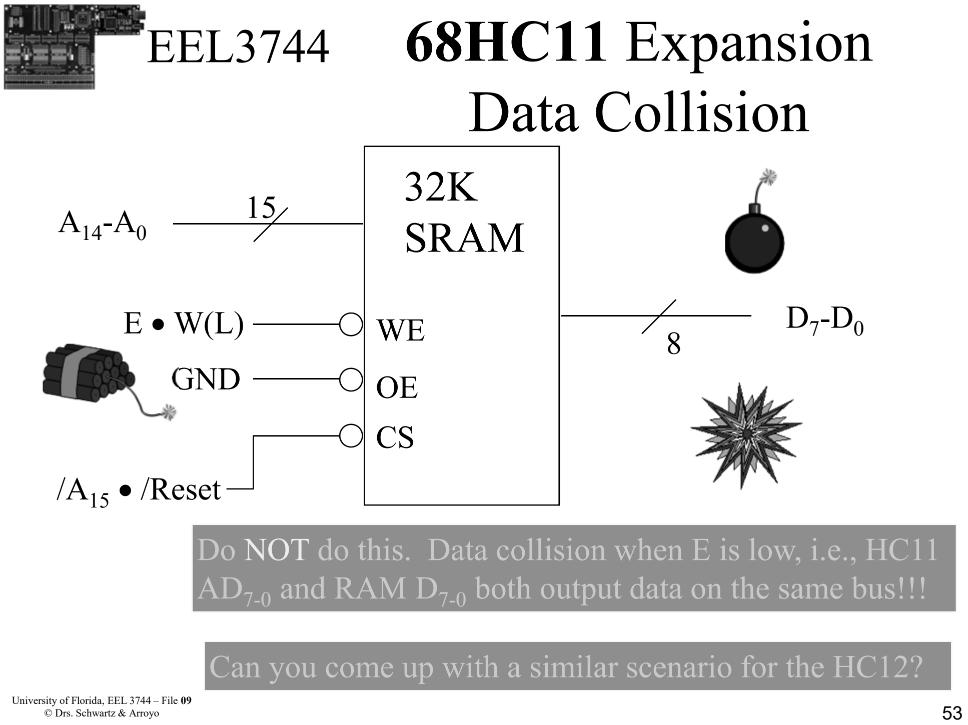
- The output buffer is automatically disabled whenever WE is asserted, even if OE asserted
- I.e., WE has priority over OE.

EEL3744 **68HC11** 32KB Memory Expansion



R/~W = R(H) = W(L)

- The output buffer is automatically disabled whenever WE is asserted, even if OE asserted
- I.e., WE has priority over OE.



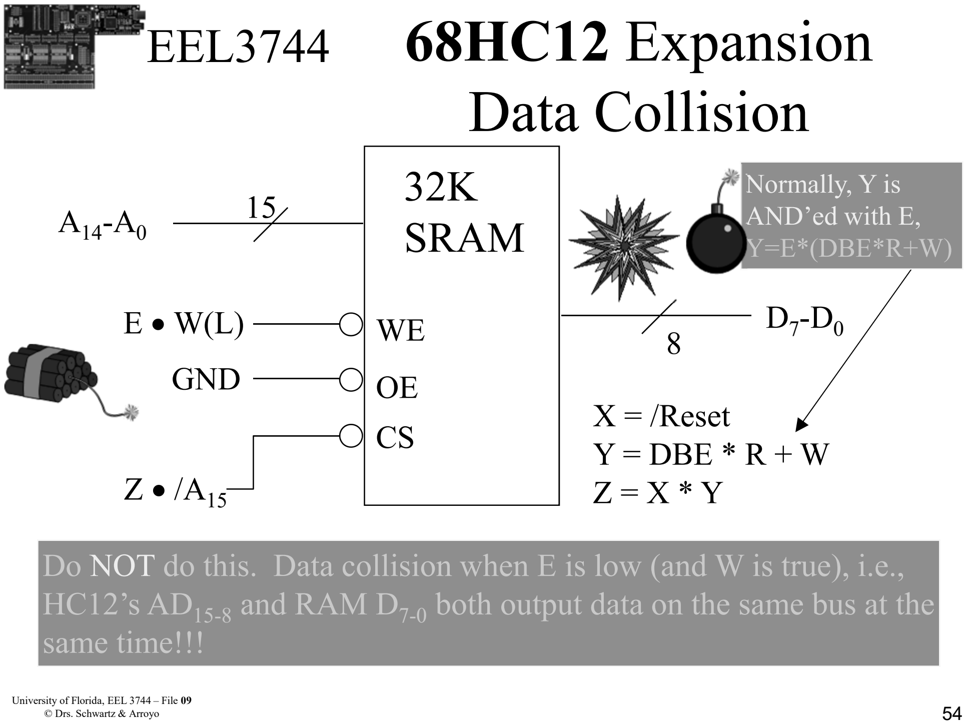
EEL3744 **68HC11** Expansion Data Collision

Do NOT do this. Data collision when E is low, i.e., HC11 AD_{7-0} and RAM D_{7-0} both output data on the same bus!!!

Can you come up with a similar scenario for the HC12?

University of Florida, EEL 3744 – File **09**
© Drs. Schwartz & Arroyo

53



EEL3744 **68HC12** Expansion Data Collision

Normally, Y is AND'ed with E, Y=E*(DBE*R+W)

X = /Reset
Y = DBE * R + W
Z = X * Y

Do NOT do this. Data collision when E is low (and W is true), i.e., HC12's AD_{15-8} and RAM D_{7-0} both output data on the same bus at the same time!!!

University of Florida, EEL 3744 – File **09**
© Drs. Schwartz & Arroyo

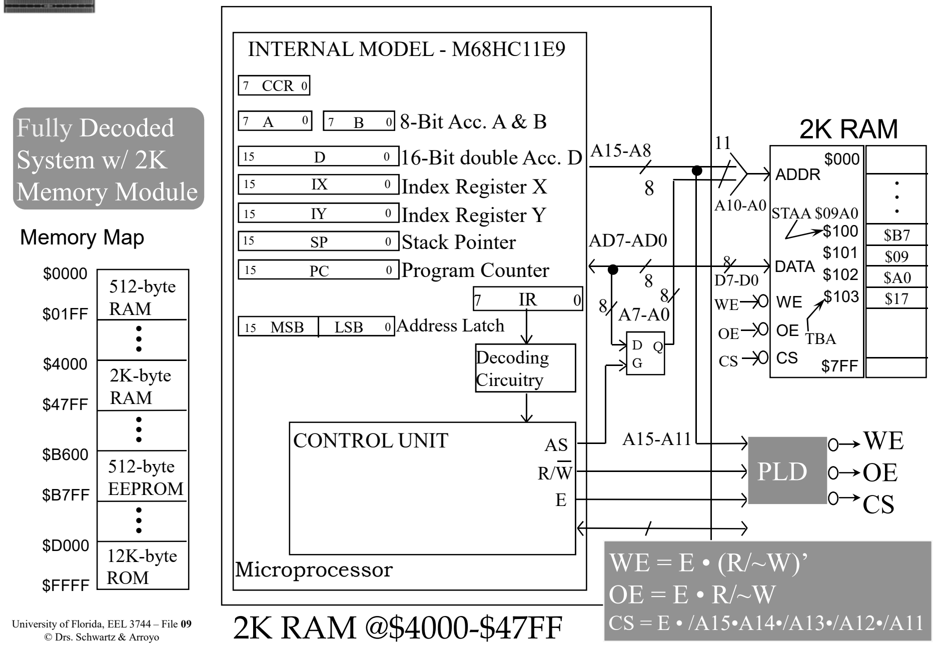
54

EEL3744 **68HC11** Interfacing the Memory Module

- Since there is already **some** on-board RAM, ROM and EEPROM in a M68HC11 system, external memory modules are often much smaller than 64K bytes, and so require fewer than 16 address bits.
- **[Example]** A µP system configuration with a **2K** external memory module starting at Memory Location **$4000** (through $47FF since 2k=$2^{11}$).
- **<Method 1> A Fully Decoded System**
- The first location of the 2K module: $4000 (%**0100 0**000 0000 0000)
- The last location of the 2K module: $47FF (%**0100 0**111 1111 1111)
- Hence, the first 5-bit of address (%**01000** above) can be used for CS (chip select) as follows:

$$CS = E \bullet /A_{15} \bullet A_{14} \bullet /A_{13} \bullet /A_{12} \bullet /A_{11}$$

EEL3744 **68HC11** Fully Decoded Expansion



WE = E • (R/~W)’
OE = E • R/~W
CS = E • /A15•A14•/A13•/A12•/A11

2K RAM @$4000-$47FF

EEL3744

Partial Address Decoding

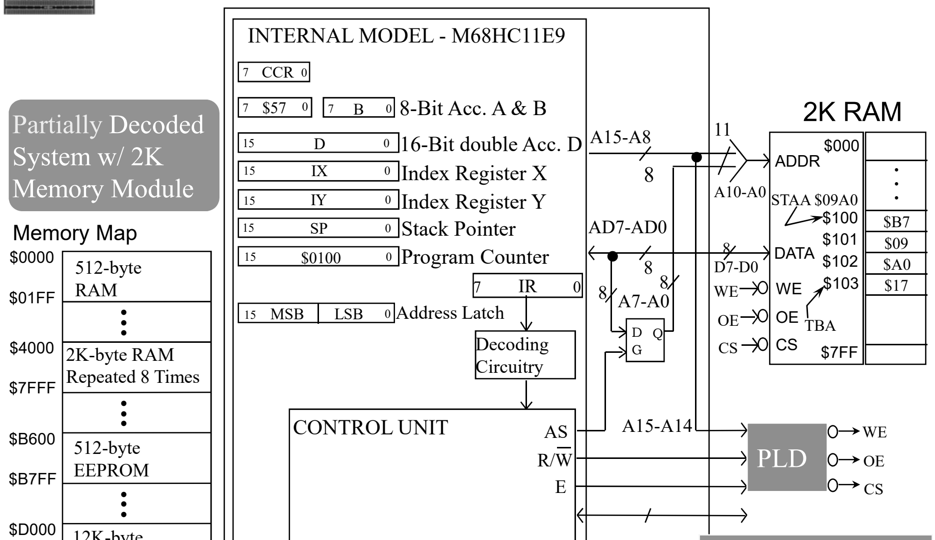
<Method 2> A Partially Decoded System using the first 2-bits

- Unlike in the fully decoded system, three of the five high address bits, mainly A_{13}, A_{12}, and A_{11} are left **unconnected** and so are **not** used to select a memory location in the memory module. Consequently, these **3 bits** of the address are effectively **don't cares**. As a result, each location in the 2K-byte external memory module has **2^3 = 8 different** addresses or ***aliases*** as follows:

%***0100*** **0**000 0000 0000 ~ %***0100*** **0**111 1111 1111 ($4000 - $47FF)
%***0100*** **1**000 0000 0000 ~ %***0100*** **1**111 1111 1111 ($4800 - $4FFF)
%***0101*** **0**000 0000 0000 ~ %***0101*** **0**111 1111 1111 ($5000 - $57FF)
⋮
%***0111*** **1**000 0000 0000 ~ %***0111*** **1**111 1111 1111 ($7800 - $7FFF)

- Hence, the first 2-bit of address (%01) can be used for CS (chip select) as follows: CS = E • /A_{15} • A_{14}
%***01***-- -000 0000 0000 ~ %***01***-- -1111 1111 1111 (start - end)

EEL3744



Partially Decoded System w/ 2K Memory Module

WE = E • (R/~W)’
OE = E • R/~W
CS = E • /A15 • A14

2K RAM @ $4000-$47FF or $4800-$4FFF … or $7000-$77FF or $7800-$7FFF

EEL3744

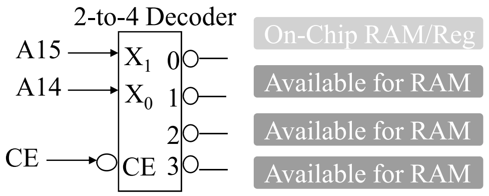
Partial -vs- Full Address Decoding

- A partially decoded system has the advantage of being simple and inexpensive since fewer circuitry is required. However, it has the disadvantage that since each memory location has more than one address, care must be taken to ensure that the addresses are referenced in some systematic manner in the software programs. Otherwise, difficulties may arise in the maintenance of the software and hardware of the µP system. For example, it may be difficult to expand the memory module at some later date.
- This is not a problem for a fully decoded system, which has the advantage that each memory location has just one address. However, a fully decoded system has the disadvantage of requiring additional complexity in the decoding circuitry, which may be substantial, especially if the memory module is complex.

EEL3744

Alternatives to Decoding by Basic Logic Chips (**use a Decoder!**)

Example: Add an additional 16K of memory. Where can we place it?



6812 Board Resources	
Reg:	0-01FF
RAM:	0800-0BFF
EEPROM:	0D00-0FFF
FLASH:	8000-FFFF (single chip only)

- If $\{A_{15}\text{-}A_{14}\}$ = 00, the address range will be 0000 0000 0000 0000 to 0011 1111 1111 1111 -or- \$0000 to \$3FFF
- If $\{A_{15}\text{-}A_{14}\}$ = 01, the address range will be 0100 0000 0000 0000 to 0111 1111 1111 1111 -or- \$4000 to \$7FFF
- If $\{A_{15}\text{-}A_{14}\}$ = 10, the address range will be 1000 0000 0000 0000 to 1011 1111 1111 1111 -or- \$8000 to \$BFFF
- If $\{A_{15}\text{-}A_{14}\}$ = 11, the address range will be 1100 0000 0000 0000 to 1111 1111 1111 1111 -or- \$C000 to \$FFFF

6811 Board Resources	
Reg:	1000-103F
RAM:	0-01FF
EEPROM:	B600-B7FF
ROM:	D000-FFFF

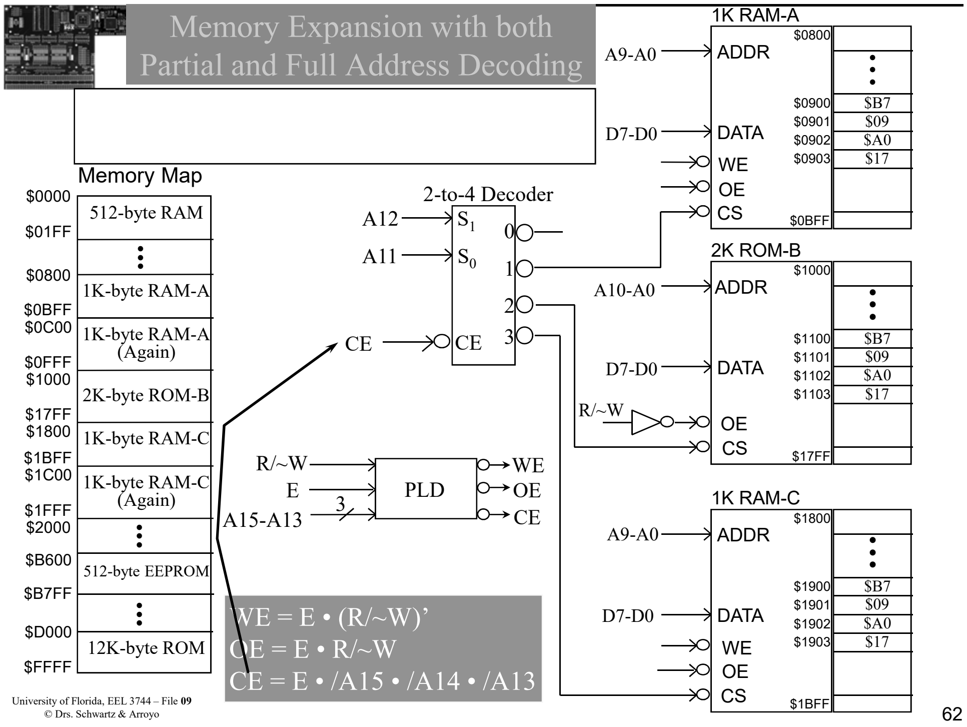
Only decoder output "1" can be used for RAM

EEL3744

Memory Expansion with both Partial and Full Address Decoding

[Example] A system memory module comprises three memory units: two 1K-byte RAMs and a single 2K-byte ROM. **A mixture of fully decoded and partially decoded schemes is used**. Note that for a memory referenced by the μP, the E-clock signal must be high (when accessing memory) to enable the 2-to-4 decoder, which in turn controls the enabling of the individual memory units.

Memory Expansion with both Partial and Full Address Decoding



Memory Map

$0000	512-byte RAM
$01FF	
	⋮
$0800	1K-byte RAM-A
$0BFF	
$0C00	1K-byte RAM-A (Again)
$0FFF	
$1000	2K-byte ROM-B
$17FF	
$1800	1K-byte RAM-C
$1BFF	
$1C00	1K-byte RAM-C (Again)
$1FFF	
$2000	⋮
$B600	512-byte EEPROM
$B7FF	⋮
$D000	12K-byte ROM
$FFFF	

WE = E • (R/~W)’
OE = E • R/~W
CE = E • /A15 • /A14 • /A13

EEL3744 Memory Expansion with both Partial and Full Address Decoding

(1) For memory unit ROM-B (2K-byte), every bit of the 16-bit address is used to specify a location:

$\{A_{15} = 0, A_{14} = 0, A_{13} = 0\}$ → to enable the decoder

$\{A_{12} = 1, A_{11} = 0\}$ → the inputs to activate decoder output 2

$\{A_{10} \sim A_0\}$ → the actual address within ROM-B

Therefore, the address range for ROM-B is from %0001 0000 0000 0000 to %0001 0111 1111 1111 ($1000 to $17FF). Also, since a ROM is read-only, just the inverted R/~W signal from the μP control bus is required to be connected to the output enable (OE) of the ROM.

(2) For memory unit RAM-A, every bit of the 16-bit address, **except A_{10}** is used to specify a location:

$\{A_{15} = 0, A_{14} = 0, A_{13} = 0\}$ → to enable the decoder

$\{A_{12} = 0, A_{11} = 1\}$ → the inputs to activate decoder output 1

$\{A_{10}\}$ → Don't Care

$\{A_9 \sim A_0\}$ → the actual address within RAM-A

Since the addresses for RAM-A are **partially decoded**, each location has **two** different addresses. the addresses range from $0800 to $0BFF, and repeat from $0C00 to $0FFF.

EEL3744 Memory Expansion with both Partial and Full Address Decoding (continued)

(3) The operation of RAM-C is similar to that of RAM-A. Each location in RAM-C also has **two** different addresses. These addresses range from $1800 to $1BFF, and repeat from $1C00 to $1FFF. This repeating is a result of A_{10} being a don't care. Also, address bits A_{11} and A_{12} must both be 1 to activate the appropriate decoder output 3 to enable the RAM-C chip enable (CE) input.

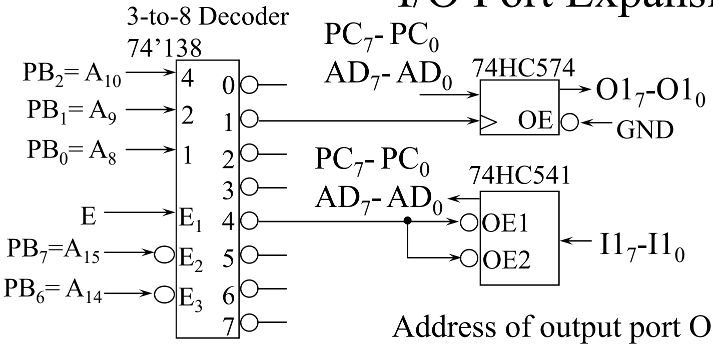
EEL3744 Memory Expansion with both Partial and Full Address Decoding (continued)

- The addresses in the range $0200 to $07FF are not valid since the corresponding decoder output 0 does not enable any memory unit
  > Recall the M68HC11E9 has 512-byte of RAM from $0000 to $01FF
- Any address greater than $1FFF (except for those for the on-board 512-byte EEPROM at $B600-$B7FF and the 12K ROM from $D000-$FFFF) is invalid since no memory unit is enabled.
  > For each of these addresses, at least one of A15, A14, and A13 is non-zero, which prevents the enabling of the 2-to-4 decoder.
- **Question:** What if the CS input of RAM-A was connected to the decoder output 0 instead of to decode output 1?
- **Answer:**
  > Obviously, the memory map for the mP would change.
  > Then the address range for RAM-A would become $0000 to $07FF, and the addresses $0800 to $0FFF would become invalid. (cont.)

EEL3744 Memory Expansion with both Partial and Full Address Decoding (continued)

- Since there is on-board RAM located from $0000 to $01FF, any address within this internal RAM range addresses the **on-board** RAM and not the external RAM-A memory module resulting in a 512-byte loss of the 1K RAM-A space.
- Conceptually, the memory module can be configured and the addresses can be decoded in any manner desired by the µP system designer.
- However, the addresses referred in the software programs must be consistent with the hardware configuration. In other words, **the data must be where the program "thinks" it is**.
- The key point here is that the µP system designer must be aware of the hardware/software interaction that is inherent in the design of a µP system.

EEL3744 Creative Interfacing: **68HC11** I/O Port Expansion



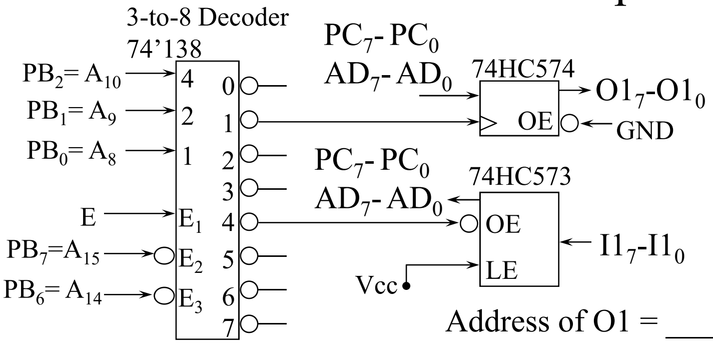
Address of output port O1 =________

Address of input port I1 =_________

- 74HC574 is an 8-bit Flip-Flop
- 74HC541 is an 8-bit Tri-State Buffer (<u>not</u> in your parts kit)

Address of O1 = 00XX X001 XXXX XXXX
For example: \$01XX, \$11XX, \$39XX
Address of I1 = 00XX X100 XXXX XXXX
For example: \$04XX , \$14XX, \$3CXX

EEL3744 Creative Interfacing: **68HC11** I/O Port Expansion



Address of O1 = ___________

Address of I1 = ___________

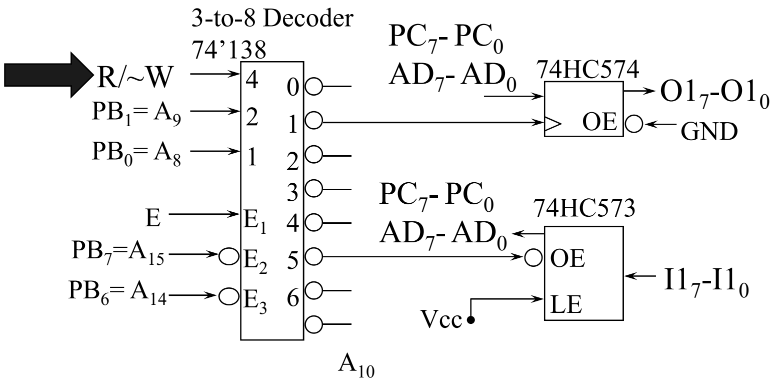
- 74HC574 is an 8-bit Flip-Flop
- 74HC573 is an 8-bit Tri-State Buffer

Addr of O1 =	00XX X001 XXXX XXXX
For example:	\$01XX, \$11XX, \$39XX
Addr of I1 =	00XX X100 XXXX XXXX
For example:	\$04XX , \$14XX, \$3CXX

74'573

OE(L)	LE(H)	D_i(H)	Q_i(H)
0=H	X	X	Z
1=L	0=L	X	Q_i
1=L	1=H	D_i	D_i

EEL3744 Creative Interfacing: **68HC11** I/O Port Expansion



Address of O1 = `00XX XX01 XXXX XXXX`

Examples: `0000 0001 0000 0000 = $0100, $3DFF`

Address of I1 = `00XX XX01 XXXX XXXX`

Examples: `0000 0001 0000 0000 = $0100, $3DFF`

EEL3744 Looking Ahead

Lab 2: Use Built-in Ports for Inputs and Outputs (plus LSA)
- Use Atmel XMEGA I/O Ports with LEDs

Lab 3: EBI (RAM Expansion) and Timers
- Find non-conflicting address for SRAM; timers instead of timing loop

Lab 4: External Interrupts & PWM
- External interrupts
- PWM for dynamic RGB LED control

Lab 5: Serial Communication and Accelerometer (UART & SPI)
- Asynchronous serial communication with interrupts (SCI or other)
- SPI with IMU's accelerometer
- Using C for the first time

Lab 6: DAC & DMA (making music!)
- Digital to analog; direct memory access; timers; create musical keyboard

Lab 7: ADC & DMA with SPI Flash ROM
- Analog to digital conversion
- Use output timing system to generate waveforms to speaker (music!)

Lab 8: Multitasking
- TBD

See doc8385, Section 28

See doc8331, Section 27

XMEGA’s EBI: External Bus Interface

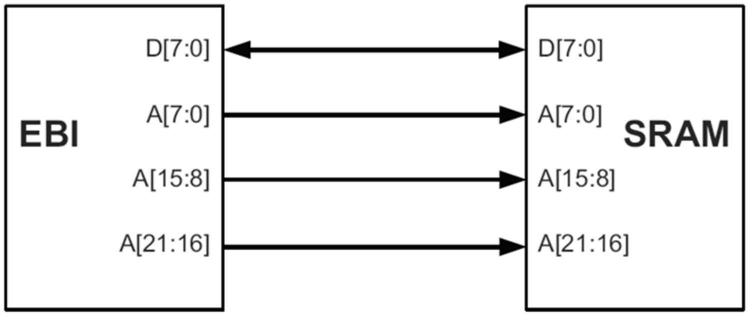
- Read **<u>the entire</u>** section 27 in doc8331!
- Many external device options are available
  > Use the external memory address range
    – 0x4000 through 0xFF FFFF
    – 0x4000 through 0x4FFF is 2^{12} = 4k = 4096
      ▪ 2^{12} because 12 bits go through all possible values
    – 0x1 0000 through 0x1 FFFF is 2^{16} = 64k = 65,536
    – Therefore, 0x4000 through 0xFF FFFF gives
      64k*256 - 4*4k = $2^{16} \times 2^{8}$ – 16k = 2^{24} = 16M – 16k
      (where M = 2^{20} , k = 2^{10})
- A write to SRAM takes 1 cycle and a read from SRAM takes 2 cycles

EEL3744

See doc8331, Figure 27-3

XMEGA’s EBI: External Bus Interface

- Non-multiplexed SRAM connection

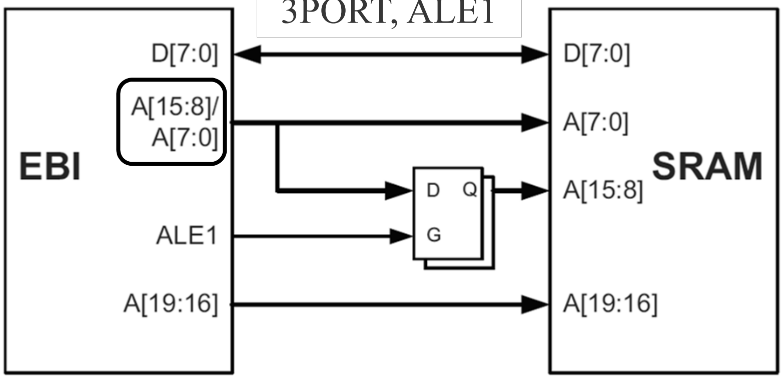


EEL3744 XMEGA's EBI: External Bus Interface

See doc8331, Figure 27-4

- Multiplexed SRAM connection using ALE1
  > **Our <u>uPad</u>** uses **<u>this</u>** multiplexed expansion mode!

<u>Mode</u>: SRAM, 3PORT, ALE1

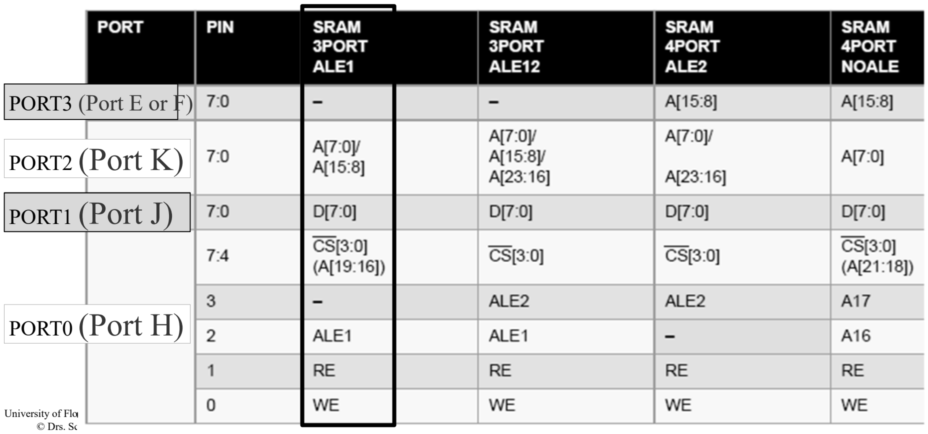


EEL3744 XMEGA's EBI: Pin-out for SRAM

See doc8331, Table 27-4

See doc8385, Sec 33.2

- Our uPad uses PortH (Port0), PortJ (Port1), and PortK (Port 2) as described below
  > ALE's are **<u>active-high</u>**; some manuals are **<u>wrong</u>**



PORT	PIN	SRAM 3PORT ALE1	SRAM 3PORT ALE12	SRAM 4PORT ALE2	SRAM 4PORT NOALE
PORT3 (Port E or F)	7:0	–	–	A[15:8]	A[15:8]
PORT2 (Port K)	7:0	A[7:0]/ A[15:8]	A[7:0]/ A[15:8]/ A[23:16]	A[7:0]/ A[23:16]	A[7:0]
PORT1 (Port J)	7:0	D[7:0]	D[7:0]	D[7:0]	D[7:0]
	7:4	$\overline{CS}$[3:0] (A[19:16])	$\overline{CS}$[3:0]	$\overline{CS}$[3:0]	$\overline{CS}$[3:0] (A[21:18])
PORT0 (Port H)	3	–	ALE2	ALE2	A17
	2	ALE1	ALE1	–	A16
	1	RE	RE	RE	RE
	0	WE	WE	WE	WE

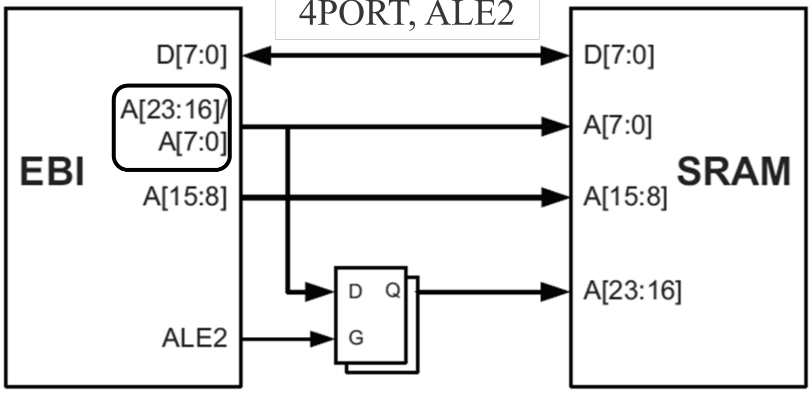
EEL3744

XMEGA’s EBI: External Bus Interface

See doc8331, Figure 27-5

- Multiplexed SRAM connection using ALE2

Mode: SRAM, 4PORT, ALE2



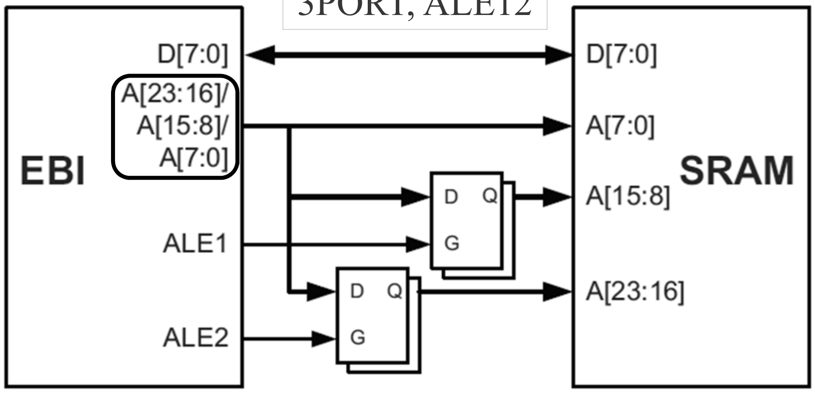
EEL3744

XMEGA’s EBI: External Bus Interface

See doc8331, Figure 27-6

- Multiplexed SRAM connection using ALE1 and ALE2

Mode: SRAM, 3PORT, ALE12



EEL3744

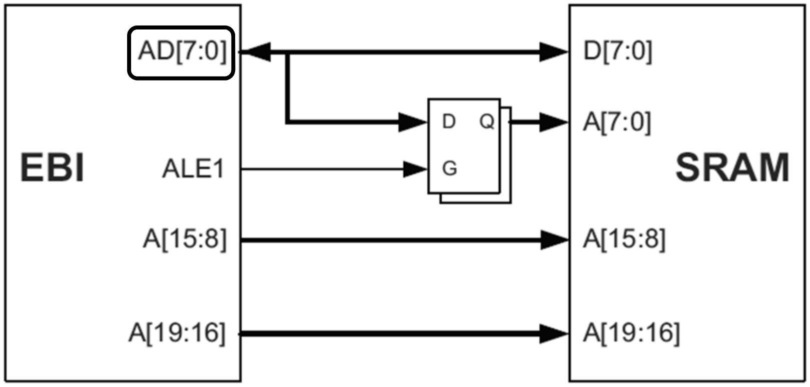
See doc8331, Figure 27-7

XMEGA's EBI: External Bus Interface

- Multiplexed SRAM LPC connection using ALE1

Mode: SRAM LPC, 4PORT, ALE1

LPC = Low Pin Count



EEL3744

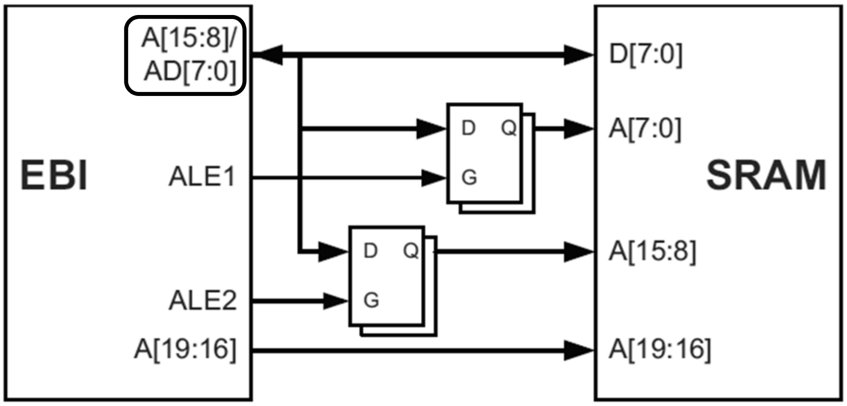
See doc8331, Figure 27-7

XMEGA's EBI: External Bus Interface

- Multiplexed SRAM LPC connection using ALE1 and ALE2

Mode: SRAM LPC, 4PORT, ALE12

LPC = Low Pin Count

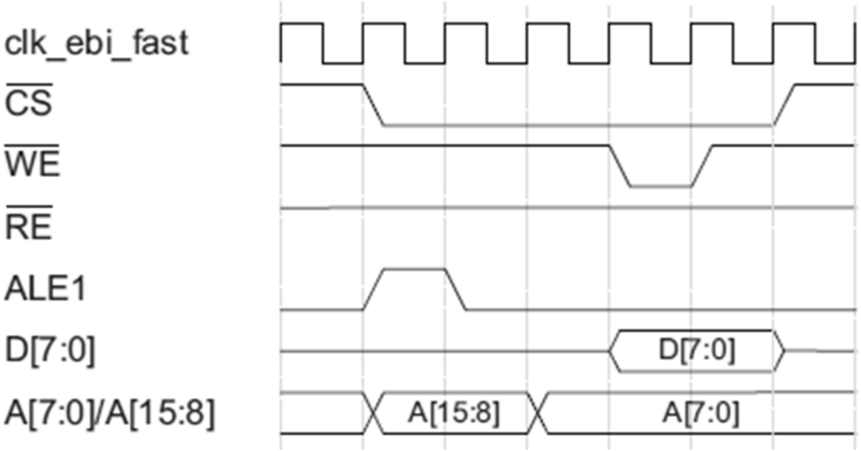


EEL3744

See doc8331, Section 36.1

XMEGA **Write** (SRAM 3-Port ALE1) Timing Diagram

- Notice the ALE1**(H)** and **WE(L)** signals

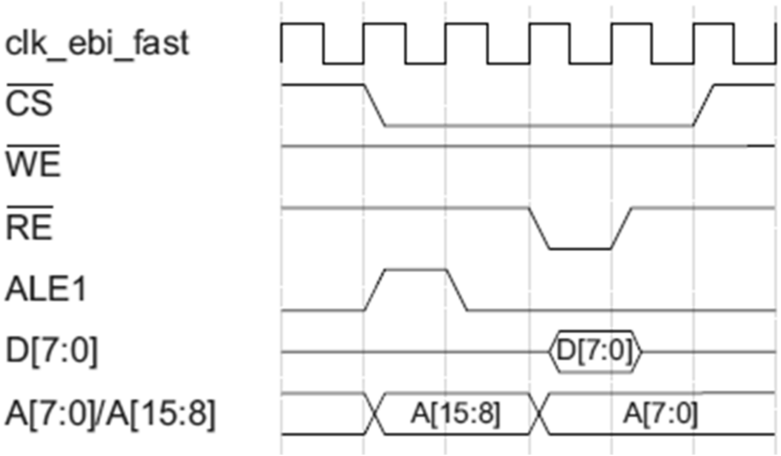


EEL3744

See doc8331, Section 36.1

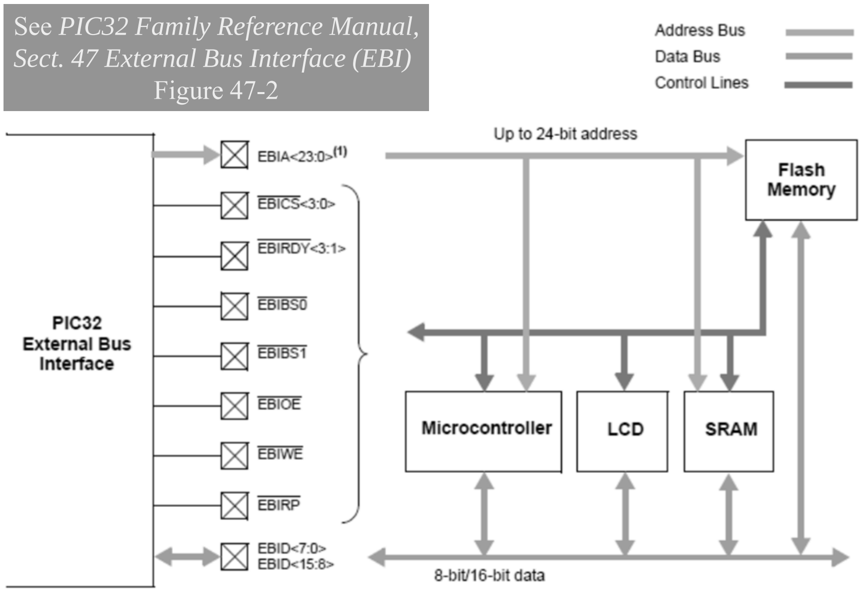
XMEGA **Read** (SRAM 3-Port ALE1) Timing Diagram

- Notice the ALE1**(H)** and **RE(L)** signals



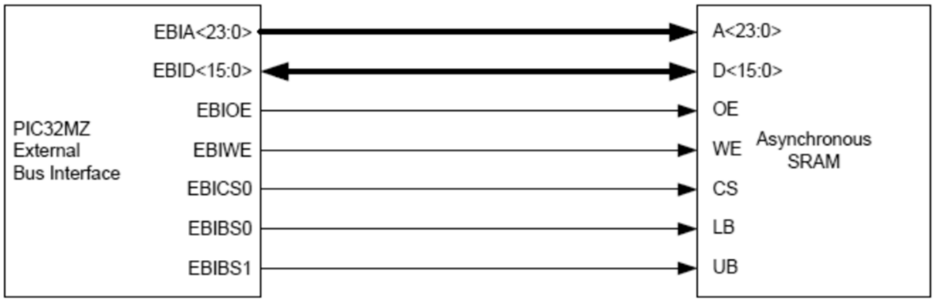
EEL3744 (Microchip's) PIC32 EBI Connections

See *PIC32 Family Reference Manual, Sect. 47 External Bus Interface (EBI)* Figure 47-2



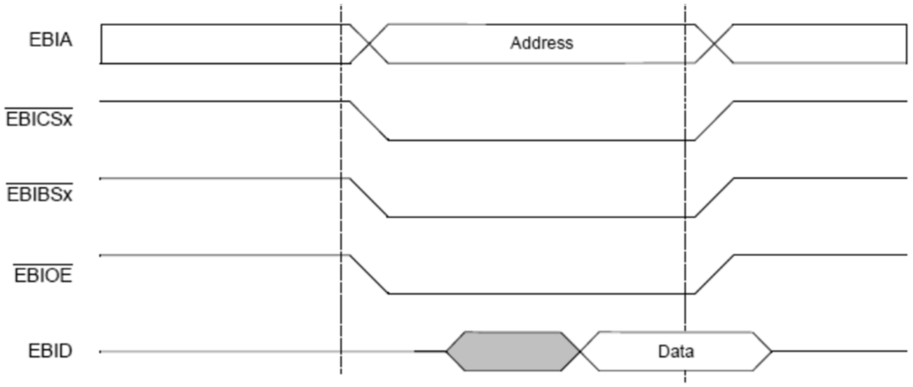
EEL3744 PIC32 EBI Interface to Asynchronous SRAM

See *PIC32 Family Reference Manual, Sect. 47 External Bus Interface (EBI)* Figure 47-4



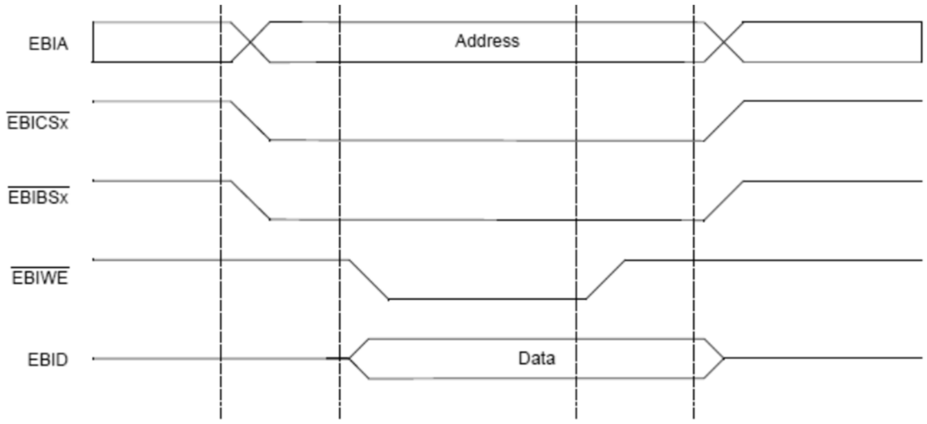
EEL3744 PIC32 EBI SRAM Read Timing

See *PIC32 Family Reference Manual, Sect. 47 External Bus Interface (EBI)* Figure 47-9


EBIA
Address
$\overline{\text{EBICSx}}$
$\overline{\text{EBIBSx}}$
$\overline{\text{EBIOE}}$
EBID
Data

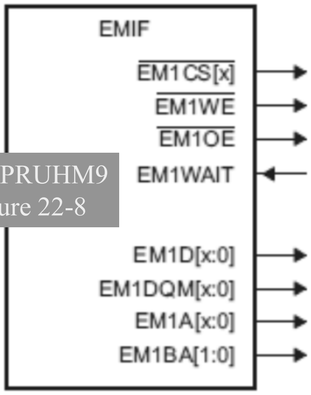
EEL3744 PIC32 EBI SRAM Write Timing

See *PIC32 Family Reference Manual, Sect. 47 External Bus Interface (EBI)* Figure 47-11


EBIA
Address
$\overline{\text{EBICSx}}$
$\overline{\text{EBIBSx}}$
$\overline{\text{EBIWE}}$
EBID
Data

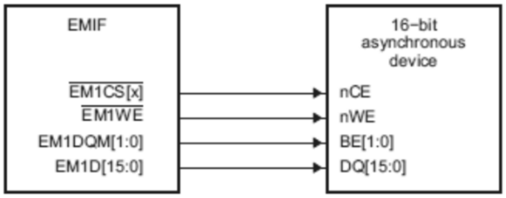
EEL3744 (TI's) Piccolo EMIF Connections

- External Memory Interface (EMIF)
  - > EM1A: Address bus; EM1BA: Bank address b
  - > EM1D: Data bus



See SPRUHM9 Figure 22-8

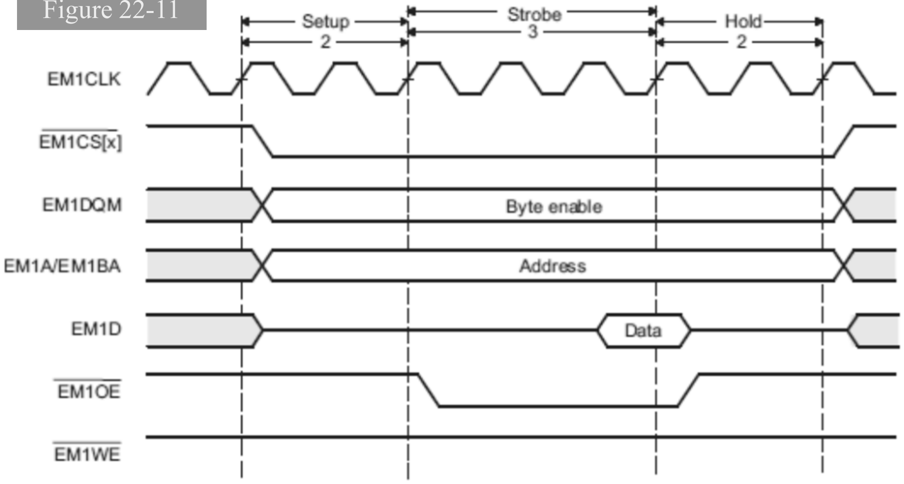
See SPRUHM9 Figure 22-10



- > EM1WE: Write enable
- > EM1DQM: Enables
- > EM1CS(x): Chip select

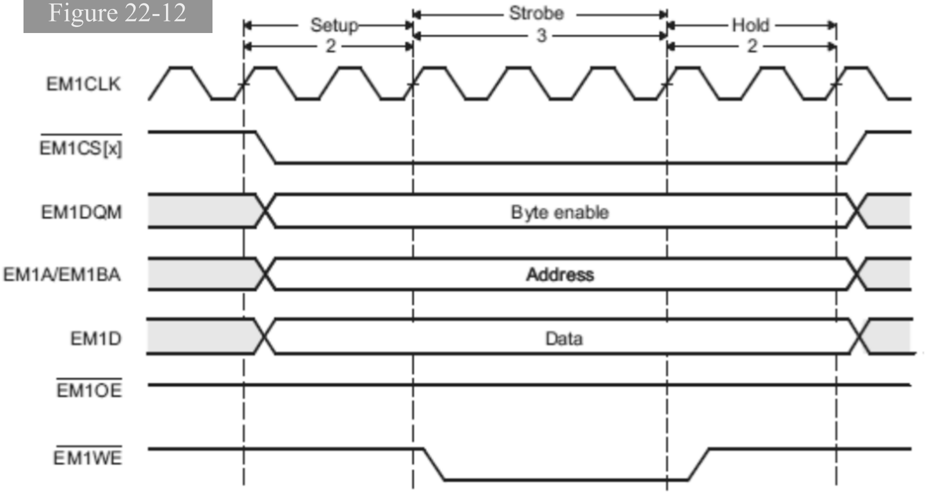
EEL3744 (TI's) Piccolo EMIF Async Read Timing

See SPRUHM9 Figure 22-11



EEL3744 (TI's) Piccolo EMIF Async Write Timing

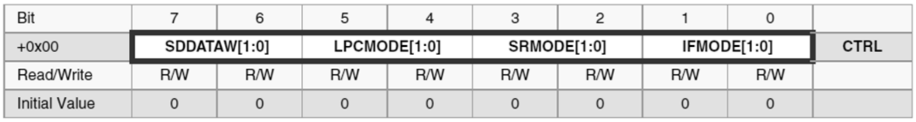
See SPRUHM9 Figure 22-12



XMEGA EBI: CTRL – Control Register

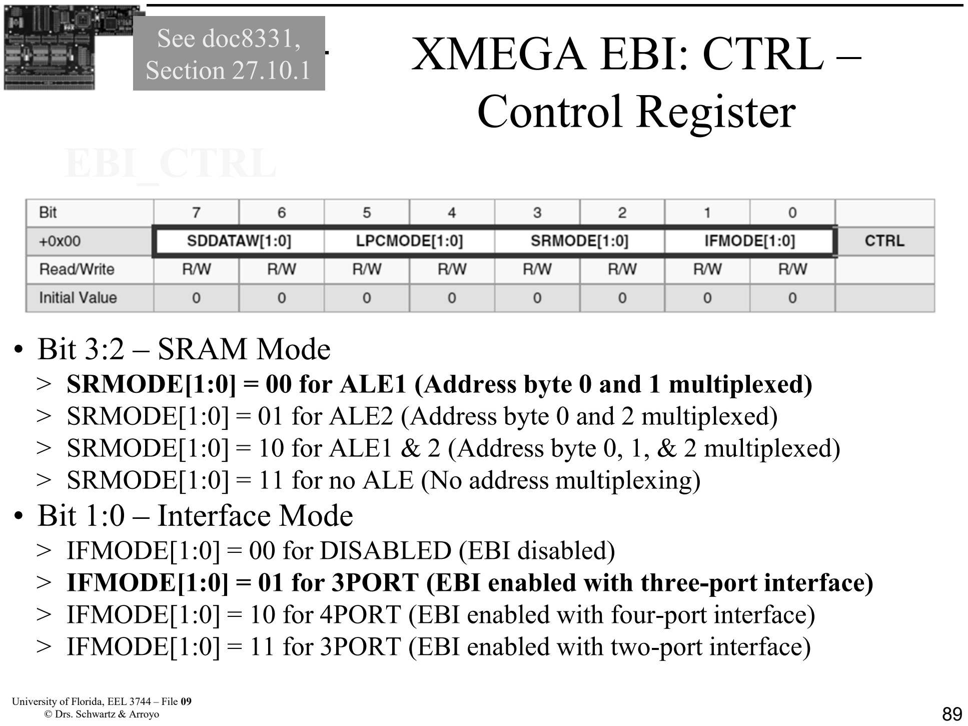
See doc8331, Section 27.10.1

EBI_CTRL



Bit	7	6	5	4	3	2	1	0	
+0x00	SDDATAW[1:0]		LPCMODE[1:0]		SRMODE[1:0]		IFMODE[1:0]		CTRL
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- Bit 7:6 – SDDATAW[1:0]: SDRAM Data Width Setting
  - > **SDDATAW[1:0] = 00 for 4-bit data bus (we need for our uPad)**
  - > SDDATAW[1:0] = 01 for 8-bit data bus
  - > SDDATAW[1:0] = 1X not available
- Bit 5:4 – LPCMODE[1:0]: SRAM Low Pin Count Mode
  - > **LPCMODE[1:0] = 00 for ALE1 (Data multiplexed with Address byte 0)**
  - > LPCMODE[1:0] = X1 not available
  - > LPCMODE[1:0] = 10 for ALE1 and ALE2 (Data multiplexed with Address byte 0 and 1)



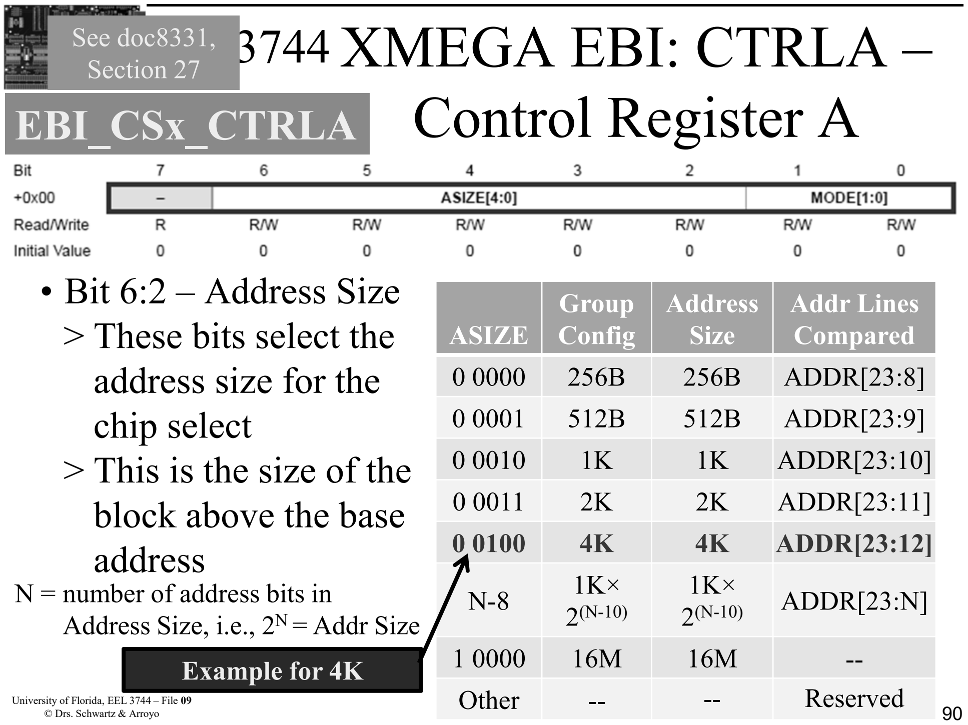
See doc8331, Section 27.10.1

XMEGA EBI: CTRL – Control Register

EBI_CTRL

Bit	7	6	5	4	3	2	1	0	
+0x00	SDDATAW[1:0]		LPCMODE[1:0]		SRMODE[1:0]		IFMODE[1:0]		CTRL
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- Bit 3:2 – SRAM Mode
  - > **SRMODE[1:0] = 00 for ALE1 (Address byte 0 and 1 multiplexed)**
  - > SRMODE[1:0] = 01 for ALE2 (Address byte 0 and 2 multiplexed)
  - > SRMODE[1:0] = 10 for ALE1 & 2 (Address byte 0, 1, & 2 multiplexed)
  - > SRMODE[1:0] = 11 for no ALE (No address multiplexing)
- Bit 1:0 – Interface Mode
  - > IFMODE[1:0] = 00 for DISABLED (EBI disabled)
  - > **IFMODE[1:0] = 01 for 3PORT (EBI enabled with three-port interface)**
  - > IFMODE[1:0] = 10 for 4PORT (EBI enabled with four-port interface)
  - > IFMODE[1:0] = 11 for 3PORT (EBI enabled with two-port interface)



See doc8331, Section 27

3744 XMEGA EBI: CTRLA – Control Register A

EBI_CSx_CTRLA

Bit	7	6	5	4	3	2	1	0
+0x00	–	ASIZE[4:0]					MODE[1:0]	
Read/Write	R	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Initial Value	0	0	0	0	0	0	0	0

- Bit 6:2 – Address Size
  - > These bits select the address size for the chip select
  - > This is the size of the block above the base address

ASIZE	Group Config	Address Size	Addr Lines Compared
0 0000	256B	256B	ADDR[23:8]
0 0001	512B	512B	ADDR[23:9]
0 0010	1K	1K	ADDR[23:10]
0 0011	2K	2K	ADDR[23:11]
0 0100	**4K**	**4K**	**ADDR[23:12]**
N-8	1K× $2^{(N-10)}$	1K× $2^{(N-10)}$	ADDR[23:N]
1 0000	16M	16M	--
Other	--	--	Reserved

N = number of address bits in Address Size, i.e., 2^N = Addr Size

Example for 4K

See doc8331, Section 27

3744 XMEGA EBI: CTRLA – Control Register A

- Size of device ➔ which address bits can be used for base address
- The base address for each chip select **must be on a 4KB boundary**

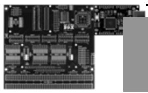
<u>Examples:</u>

- > For a 256 size, $EF 37xx
  - – But address $EF 3yxx will work for all y!
- > For a 4K size, $EF 3xxx
- > For a 64K, $EF xxxx
- > For a 1M, $Ex xxxx
- > **Anything less than 4k, it will still use 4k! (since only the top 4 nibbles are used)**
- Note that **<u>ALL</u>** unused address bits must have **<u>ALL</u>** values available

N = number of address bits in Address Size, i.e., 2^N = Addr Size

ASIZE	Group Config	Address Size	Addr Lines Compared
0 0000	256B	256B	ADDR[23:8]
0 0001	512B	512B	ADDR[23:9]
0 0010	1K	1K	ADDR[23:10]
0 0011	2K	2K	ADDR[23:11]
0 0100	**4K**	**4K**	**ADDR[23:12]**
N-8	1K× $2^{(N-10)}$	1K× $2^{(N-10)}$	ADDR[23:N]
1 0000	16M	16M	--
Other	--	--	Reserved

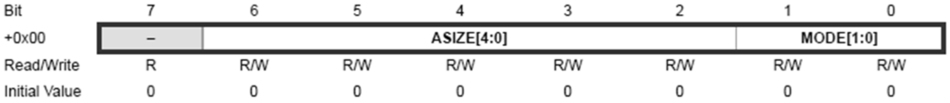
Example for 4K



See doc8331, Section 27

4 XMEGA EBI: CTRLA – Control Register A

EBI_CSx_CTRLA



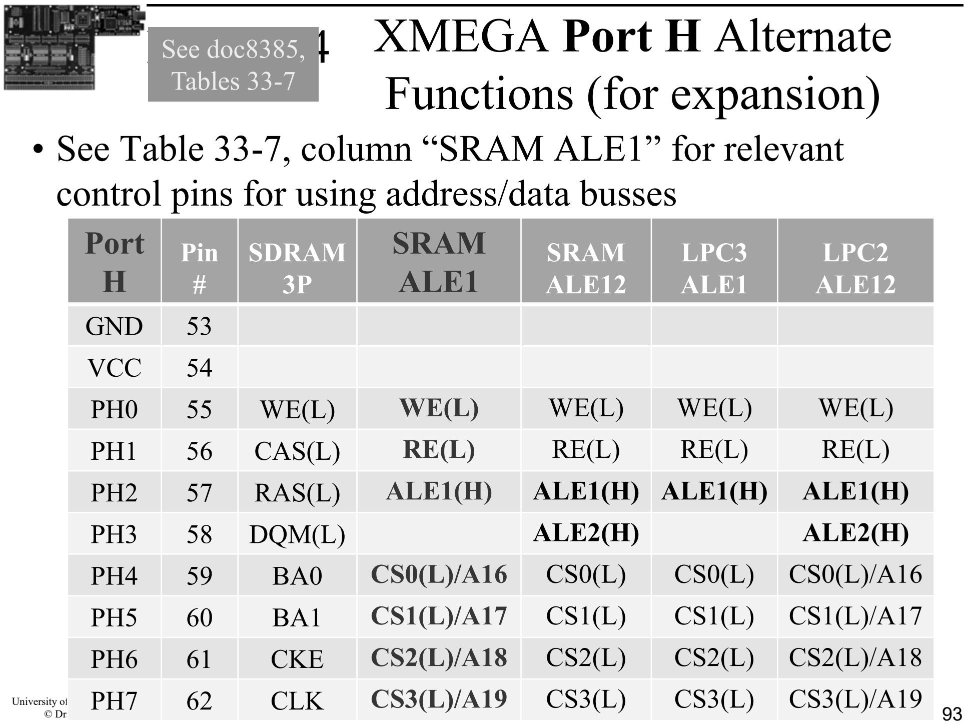
Bit	7	6	5	4	3	2	1	0
+0x00	–	ASIZE[4:0]					MODE[1:0]	
Read/Write	R	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Initial Value	0	0	0	0	0	0	0	0

- Bit 1:0 – Chip Select Mode
  - > These bits select the chip select mode and decide what type of interface is used for the external memory or peripheral

MODE	Group Config	Description
00	DISABLE	Chip select disabled
01	**SRAM**	**Enable chip select for SRAM**
10	LPC	Enable chip select for SRAM LPC
11	SDRAM	Enable chip select for SDRAM

Need for our labs

Ex: **EBI_CS0_CTRLA**



See doc8385, Tables 33-7

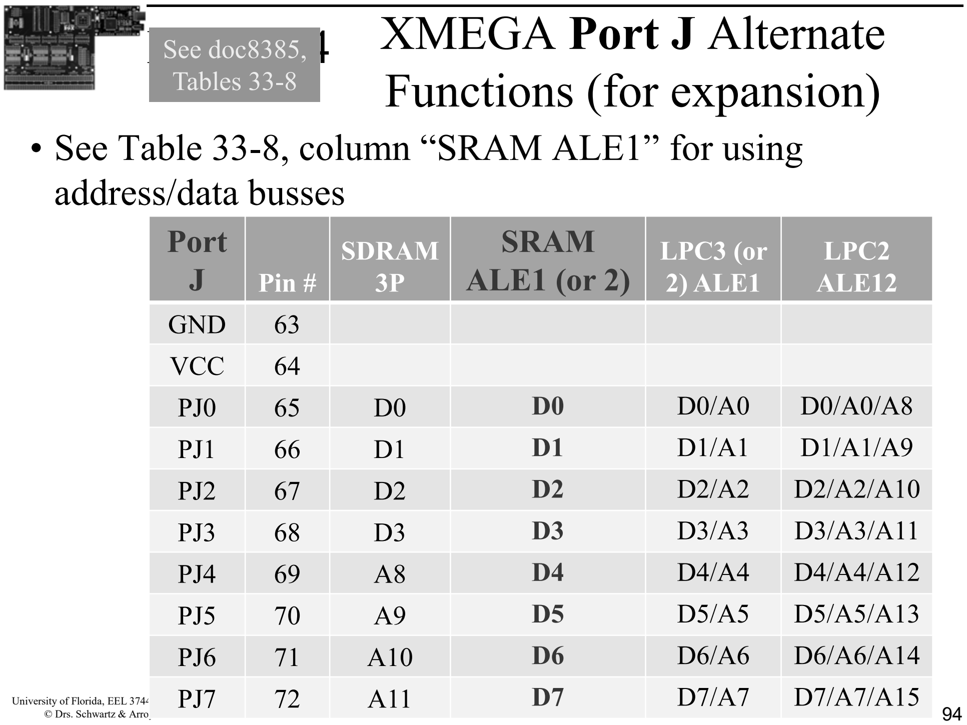
4 XMEGA **Port H** Alternate Functions (for expansion)

- See Table 33-7, column “SRAM ALE1” for relevant control pins for using address/data busses

Port H	Pin #	SDRAM 3P	SRAM ALE1	SRAM ALE12	LPC3 ALE1	LPC2 ALE12
GND	53					
VCC	54					
PH0	55	WE(L)	**WE(L)**	WE(L)	WE(L)	WE(L)
PH1	56	CAS(L)	**RE(L)**	RE(L)	RE(L)	RE(L)
PH2	57	RAS(L)	**ALE1(H)**	**ALE1(H)**	**ALE1(H)**	**ALE1(H)**
PH3	58	DQM(L)		**ALE2(H)**		**ALE2(H)**
PH4	59	BA0	**CS0(L)/A16**	CS0(L)	CS0(L)	CS0(L)/A16
PH5	60	BA1	**CS1(L)/A17**	CS1(L)	CS1(L)	CS1(L)/A17
PH6	61	CKE	**CS2(L)/A18**	CS2(L)	CS2(L)	CS2(L)/A18
PH7	62	CLK	**CS3(L)/A19**	CS3(L)	CS3(L)	CS3(L)/A19

University of
© Dr

93



See doc8385, Tables 33-8

4 XMEGA **Port J** Alternate Functions (for expansion)

- See Table 33-8, column “SRAM ALE1” for using address/data busses

Port J	Pin #	SDRAM 3P	SRAM ALE1 (or 2)	LPC3 (or 2) ALE1	LPC2 ALE12
GND	63				
VCC	64				
PJ0	65	D0	**D0**	D0/A0	D0/A0/A8
PJ1	66	D1	**D1**	D1/A1	D1/A1/A9
PJ2	67	D2	**D2**	D2/A2	D2/A2/A10
PJ3	68	D3	**D3**	D3/A3	D3/A3/A11
PJ4	69	A8	**D4**	D4/A4	D4/A4/A12
PJ5	70	A9	**D5**	D5/A5	D5/A5/A13
PJ6	71	A10	**D6**	D6/A6	D6/A6/A14
PJ7	72	A11	**D7**	D7/A7	D7/A7/A15

University of Florida, EEL 3744
© Drs. Schwartz & Arro

94

See doc8385, Tables 33-9

4 XMEGA **Port K** Alternate Functions (for expansion)

- See Table 33-9, column "SRAM ALE1" for using address bus

Port K	Pin #	SDRAM 3P	SRAM ALE1	SRAM ALE2	LPC3 ALE1
GND	73				
VCC	74				
PK0	75	A0	**A0/A8**	A0/A8/A16	A8
PK1	76	A1	**A1/A9**	A1/A9/A17	A9
PK2	77	A2	**A2/A10**	A2/A10/A18	A10
PK3	78	A3	**A3/A11**	A3/A11/A19	A11
PK4	79	A4	**A4/A12**	A4/A12/A20	A12
PK5	80	A5	**A5/A13**	A5/A13/A21	A13
PK6	81	A6	**A6/A14**	A6/A14/A22	A14
PK7	82	A7	**A7/A15**	A7/A15/A23	A15

EEL 3744

Input_Port.asm

XMEGA EBI Chip Select Base Address

- When setting up EBI for a specific chip select, the base address for that chip select must be chosen to initialize and reserve an address range that will trigger the correct chip select lines
- The base address will be used with the settings in EBI_CS**x**_CTRLA to determine a block of addresses for a specific chip select (**x** = 0, 1, 2, or 3)
- The base address has the following properties
  - > Consists of up to 12 (=24-n) bits for the address, A_{23}:A_n (n=12, 13, …)
    - – The lower n bits A_{n-1}:A_0 are assumed to be zero

EEL 3744

Input_Port.asm

XMEGA EBI Chip Select Base Address

- First, we define the first address that external memory can be placed, which will become the base address, e.g.,

  .set IN_PORT = 0x37E000
- Next, we have to point to the address which holds the desired chip selects base address
  > We use ZH and ZL registers to hold the high and low address components of EBI_CS**x**_BASEADDR (**x** = 0, 1, 2, or 3) [could use X or Y instead of Z], i.e.,

  ldi ZH, HIGH(EBI_CS0_BASEADDR)
  ldi ZL, LOW(EBI_CS0_BASEADDR)

EEL 3744

Input_Port.asm

XMEGA EBI Chip Select Base Address

ldi ZH, HIGH(EBI_CS0_BASEADDR)
ldi ZL, LOW(EBI_CS0_BASEADDR)

- After above, Z points to the lower byte of the desired Chip Select Base Address
- We now want to load the chosen base address using Z (or X or Y)
  > Use only the **top 12 bits**

ADDR Name	Value
EBI_CS0_BASEADDR (Lower)	??
EBI_CS0_BASEADDR (Upper)	??
EBI_CS1_BASEADDR (Lower)	??
EBI_CS1_BASEADDR (Upper)	??
EBI_CS2_BASEADDR (Lower)	??
EBI_CS2_BASEADDR (Upper)	??
EBI_CS3_BASEADDR (Lower)	??
EBI_CS3_BASEADDR (Upper)	??

EEL 3744

Input_Port.asm

XMEGA EBI Chip Select Base Address

- First take the middle byte of the 24 bit address (A15:8), and then store it, i.e.,

  ldi R16, byte2(IN_PORT)
  st Z+, R16

  > For example, with IOPORT = 0x37E000
    – R16 = 0xE0
- This will be stored to the location pointed to by Z
- Z is then incremented to point at the **upper byte** of the base address

ADDR Name	Value
EBI_CS0_BASEADDR (Lower)	**0xE0**
EBI_CS0_BASEADDR(Upper)	??
EBI_CS1_BASEADDR (Lower)	??
EBI_CS1_BASEADDR (Upper)	??
…	…

EEL 3744

Input_Port.asm

XMEGA EBI Chip Select Base Address

- Next we shift the desired base address by 16 bits, load into a register, i.e.,

  ldi R16, byte3(IN_PORT)
  st Z, R16

  > For example, with IOPORT = 0x37E000
    – R16 = 0x37
- This will be stored to the location pointed to by Z

ADDR Name	Value
EBI_CS0_BASEADDR (Lower)	0xE0
EBI_CS0_BASEADDR(Upper)	**0x37**
EBI_CS1_BASEADDR (Lower)	??
EBI_CS1_BASEADDR (Upper)	??
…	…

EEL3744

XMEGA EBI Chip Select Base Address

`Input_Port.asm`

- If your desired base address is only two bytes, only the top nibble will matter
  > For example, only the 0x7 of 0x7000 would be used
- If a 3-byte base address is desired, for example 0x1E_3000
  > The top three nibbles are passed to the base address location, i.e., 0x1E3
  > The chip select is triggered with any address starting at 0x1E_3000 and going up to the size chosen in the EBI_CTRLA register
  – If size = 16k, the CS (you might assume) is true for address 0x1E_3000-0x1E_6FFF; but this is **incorrect**! **> 1 AND gate** would be needed!
- You may hard code the values, but using the method shown in the **Input_Port** example allows for more flexibility with base address values

EEL3744

XMEGA EBI Chip Select (CS0-CS3)

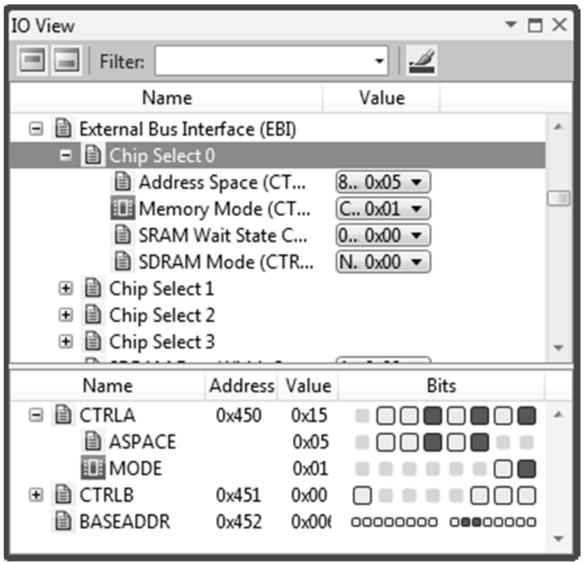
- Each chip select **must start** at an address boundary
  > The block size is determined by the address bits that can change
  – With the N changing address bits, the block size is 2^N
  > The other bits A23:AN **must be fixed**
  – A 4k block can start anywhere in expansion memory, e.g., 0x4000, 0x5000, 0x6000, … 0x1 0000, 0x2 0000, 0xF 0000, 0x10 0000, …, 0x37 0000, …, 0xFF 0000
    ▪ The first hex digit does not change for any one of these
  – An 8k block can start at 0x4000, 0x6000, 0x8000, …
    ▪ It can **NOT** start at 0x5000, since the A15:A12 = 0101 or 0110
  – A 16k block can start at 0x4000, 0x8000, 0xC000, 0x1 0000, …
    ▪ It can **NOT** start at 0x5000, 0x6000, 0x7000
    ▪ For a block starting at 0x4000, A15:A14=01

EEL3744

XMEGA EBI Chip Select Base Address

`Input_Port.asm`

- When simulating in Atmel Studio, you can watch the EBI port using the IO View window and finding “External Bus Interface (EBI)”



IO View

Filter:

Name	Value
⊟ External Bus Interface (EBI)	
⊟ Chip Select 0	
Address Space (CT...	8.. 0x05 ▾
Memory Mode (CT...	C.. 0x01 ▾
SRAM Wait State C...	0.. 0x00 ▾
SDRAM Mode (CTR...	N. 0x00 ▾
⊞ Chip Select 1	
⊞ Chip Select 2	
⊞ Chip Select 3	

Name	Address	Value	Bits
⊟ CTRLA	0x450	0x15	○○●○●○●
ASPACE		0x05	○○●○●
MODE		0x01	○●
⊞ CTRLB	0x451	0x00	○ ○○○
BASEADDR	0x452	0x00(	○○○○○○○○ ○●●○○○○○

EEL3744

The End!