Lecture 30

April 9, 2012

The MC9S12 Expanded Mode (2/5)

Using MSI Logic for Port Expansion

- Accessing External Memory and Ports on the MC9S12 in Expanded Mode
- Byte Order in Microprocessors
- How to determine if a bus cycle accesses one or two bytes
- A Simple Parallel Output Port
- Using MSI Logic To Build An Output Port
- Using MSI Logic To Build An Output Port

Accessing External Memory and Ports on the MC9S12 in Expanded Mode

- In expanded mode, the MC9S12 has a multiplexed 16-bit address and data bus.
- With a 16-bit address bus, the MC9S12 can access $2^{16} = 65,536$ bytes of data
- With a 16-bit data bus, the MC9S12 can access 16 bits (two bytes) in a single bus cycle
- In expanded mode, the MC9S12 uses Port A and Port B as the multiplexed address/data bus
- Timing is controlled by the E clock
- When the E clock is low, the MC9S12 places the address on the multiplexed bus
  - Port A is used for address bits 15-8
  - Port B is used for address bits 7-0
- When the E clock is high, the MC9S12 uses the multiplexed bus for data: bus
  - Port A is used for the byte at the even address
  - Port B is used for the byte at the odd address

  For example, if accessing the sixteen-bit word at address 0x4000 (the bytes at addresses 0x4000 and 0x4001), Port A will access the byte at address 0x4000, and Port B will access the byte at address 0x4001.

Byte Order in Microprocessors

- There are two ways to store bytes in a microprocessor memory. For example, if you wanted to store the 16-bit word 0x1234 into memory locations 0x2000 and 0x2001, you could do it in two ways:

	Big Endian		Little Endian	
Address	0x2000	0x2001	0x2000	0x2001
Byte	0x12	0x34	0x34	0x12

- Motorola and Freescale (and some other manufacturers) use Big Endian (big end, or most significant part, appears first in memory, big part is in lower part of memory)
- Intel (and some other manufacturers) use Little Endian (little end appears first, smaller part of the number is in lower part of memory)
- Data types of more than one byte written on a Motorola machine will not be read properly on an Intel machine without first swapping byte order (and vice versa).
- In the discussion which follows, even byte refers to a byte at an even address, odd byte refers to a byte at an odd address. High byte refers to the most significant byte of a 16-bit word, low byte refers to the least significant byte of a 16-bit word. For the MC9S12, the high byte is at the even address, and the low byte is at the odd address for a 16-bit access.

How to determine if a bus cycle accesses one or two bytes

- Sometimes you only want to access one byte at a time. For example,
  - `ldaa $4001`

  will access the single byte at address 0x4001.
- To determine whether it should access one byte or two bytes, the MC9S12 uses the $\overline{\text{LSTRB}}$ and A0 lines.
  - $\overline{\text{LSTRB}}$ low means that the MC9S12 is accessing the lower byte (byte at the odd address) of a sixteen-bit word
  - $\overline{\text{LSTRB}}$ high means that the MC9S12 is accessing the upper byte (byte at the even address) of a sixteen-bit word
  - A0 low means that the MC9S12 is accessing the upper (even) byte of a sixteen-bit word
  - A0 high means that the MC9S12 is accessing the lower (odd) byte of a sixteen-bit word

$\overline{\text{LSTRB}}$	A0	Type of Access
0	0	16-bit access of an even address Accesses bytes at even address and subsequent odd address
0	1	8-bit access of an odd address
1	0	8-bit access of an even address
1	1	Not allowed on external bus

- The instruction
  - `ldaa $4000`

  accesses the byte at address 0x4000, but doesn't access the byte at address 0x4001. For this access, the MC9S12 will put 0x4000 on the bus (A0 = 0, access byte at even address), and will make $\overline{\text{LSTRB}}$ = 1 (don't access byte at the odd address).
- The instruction
  - `ldaa $4001`

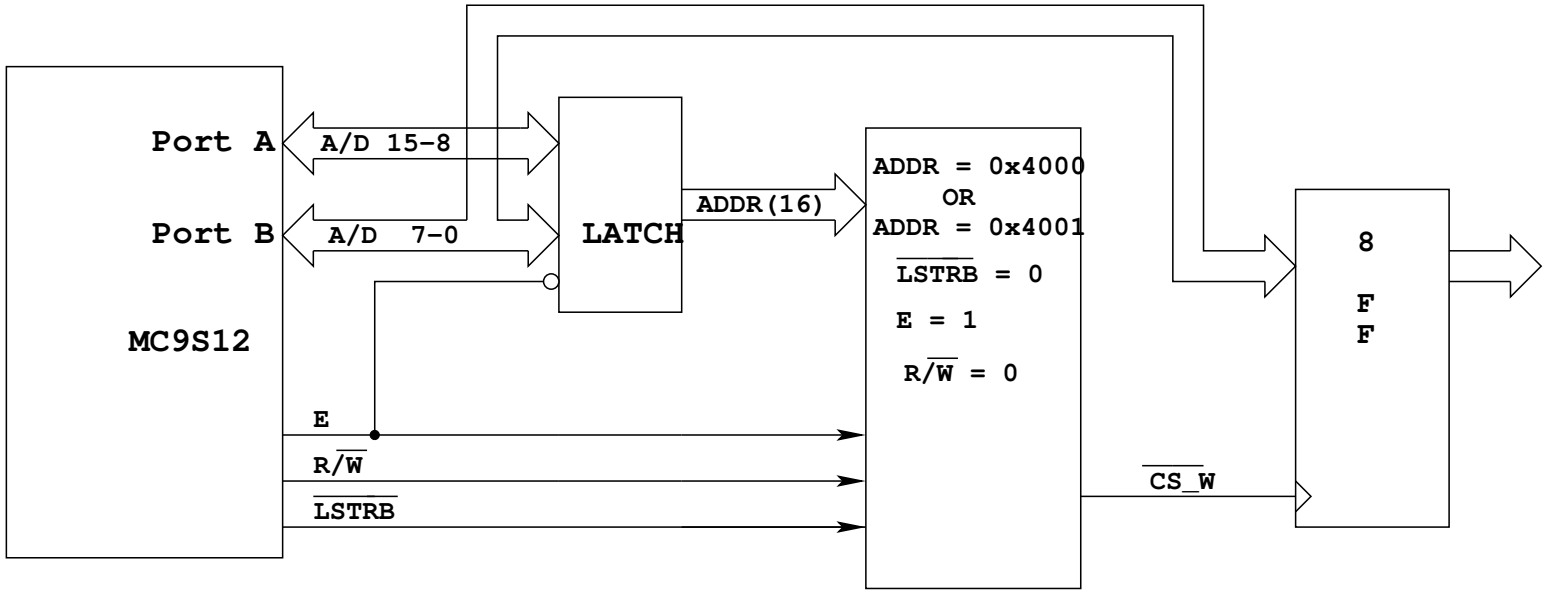
  accesses the byte at address 0x4001, but doesn't access the byte at address 0x4000. For this access, the MC9S12 will put 0x4001 on the bus (A0 = 1, do not access byte at even address), and will make $\overline{\text{LSTRB}}$ = 0 (access byte at odd address).
- The instruction
  - `ldd $4000`

  accesses the bytes at addresses 0x4000 and 0x4001. For this access, the MC9S12 will put 0x4000 on the bus (A0 = 0, access byte at even address), and will make $\overline{\text{LSTRB}}$ = 0 (access byte at odd address).

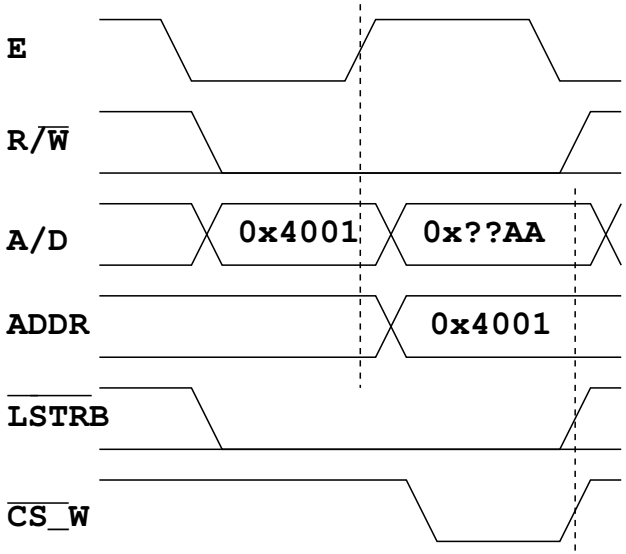
- What to check for on the bus to determine if the MC9S12 is accessing a particular byte
  - To check to see if the byte at address 0x4000 is being accessed, look for 0x4000 on the address bus (do not need to check $\overline{\text{LSTRB}}$).
  - To check to see if the byte at address 0x4001 is being accessed, look for either 0x4000 or 0x4001 on the address bus (i.e., A0 is a don't care), and make sure $\overline{\text{LSTRB}}$ is low.

A Simple Parallel Output Port

- We want a port which will write 8 bits of data to the outside
- Such a port is similar to Port A or Port B when all pins are set up as output
- We need some hardware to latch the output data at the time the MC9S12 puts the data on the data bus
- We can use a set of 8 D flip-flops to latch the data
  - The D inputs will be connected to the data bus
  - The clock to latch the flip-flops should make its low-to-high transition when the MC9S12 has the appropriate data on the bus
  - The MC9S12 will access the flip-flops by writing to an address. We must assign an address for the tri-state buffer
  - We must have hardware to demultiplex the address from the data, and to determine when the MC9S12 is writing to this address
  - The 8-bit inputs of the D flip-flops will be connected to 8 bits of the 16-bit address/data bus of the MC9S12
    - If the address of the input is even, we need to connect the flip flop inputs to the even (high) byte of the bus, which is connected to AD15-8 (what was Port A)
    - If the address of the input is odd, we need to connect the flip flop inputs to the odd (low) byte of the bus, which is connected to AD7-0 (what was Port B)
  - The hardware should latch the data on the high-to-low transition of the E-clock
  - Our hardware should bring the clock of the flip-flops low when
    1. The address of the flip-flops is on the address bus
    2. The MC9S12 is writing to this address
    3. The MC9S12 is writing the high byte if the address is even, or the low byte if the address is odd
    4. E is high
- For example, consider an output port at address 0x4001 (an odd address, or low byte):



Example: Write an 0xAA to address 0x0401

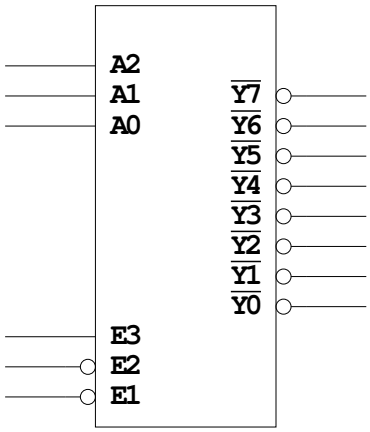


Note: ADDR can be 0x4000 or 0x4001 with LSTRB = 0

Using MSI Logic To Build An Output Port

- Many designs use standard MSI logic for microprocessor expansion
- This provides an inexpensive way to expand microprocessors
- One MSI device often used in such expansions is a decoder, such as the 74HC138 decoder chip

74HC138



When E3 high, E2 and E1 low, one of the outputs of the 74138 will go low

A2, A1, A0 determine which output will be low.

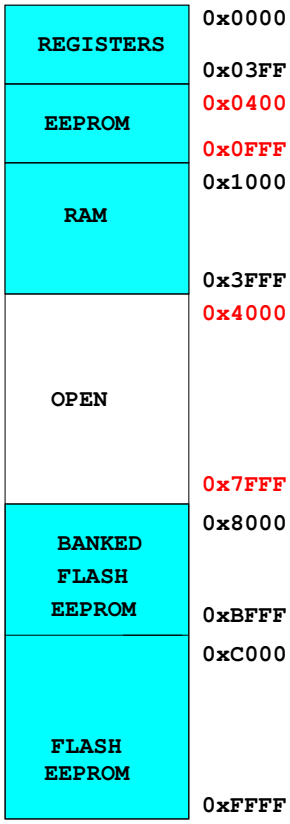
A2	A1	A0	Output
0	0	0	Y0
0	0	1	Y1
0	1	0	Y2
0	1	1	Y3
1	0	0	Y4
1	0	1	Y5
1	1	0	Y6
1	1	1	Y7

Using MSI Logic To Build An Output Port

- A 74HC138 decoder can be used to select a range of addresses
- The outputs of the 74HC138 are used to select up to 8 different devices
- The enable inputs are used to determine when one of the 8 different devices should be selected
- Because the data part of the memory cycle occurs when E is high, the active high enable input is usually connected to the MC9S12's E clock
- The active low enable inputs are usually connected to highest-order address lines (A15 and A14), or to combinational logic which is driven by highest-order address lines
- The address inputs of the 74HC138 are usually connected to the next highest-order address bits. Sometimes the low-order address input is connected to the R/W line of the MC9S12 to allow separate selection of input and output devices
- For example, on the figure on the next page, A15 and $\overline{\text{A14}}$ are connected to the active low enable inputs. This means that one of the outputs will be selected only when A15 and A14 are low, or for the address range 0x4000 to 0x7fff
- A13, A12 and A11 are connected to the three address inputs of the 74HC138. For example, addres Y5 will be low when A13 A2 A11 are 1 0 1. Thus Y5 will be select for addresses from 0010100000000000_2 to 0010111111111111_2, or from 0x6800 to 0x6fff
- The Y5 output may be connected to a device which needs only one address
  - In this case, the device will be accessed with any address from 0x6800 to 0x2fff
  - This is called partial address decoding — to save expense, we do not decode all address lines, but only enough to put in the number of devices we need
  - We also need to demultiplex only those address lines we decode, which also saves some expense

MC9S12 Memory Map

(Expanded Mode)

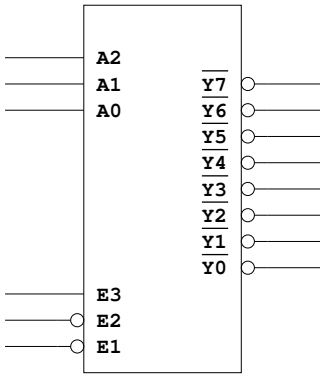


Can map external device into any unused space

When using Altera for address decoding, can select any desired address.

Can map using less expensive chip like 74HC138

74HC138



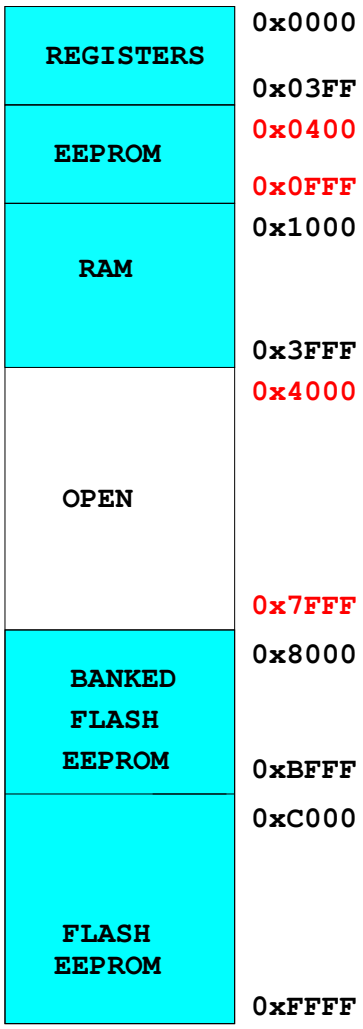
Want to select external device when E is high -- Connect E to E3

Will use A15 and A14 as the other enables. This will divide memory into 1/4 of total

A15	A14	Chip enabled?
0	0	0x0000 – 0x3fff
0	1	0x4000 – 0x7fff
1	0	0x8000 – 0xbfff
1	1	0xc000 – 0xffff

MC9S12 Memory Map

(Expanded Mode)

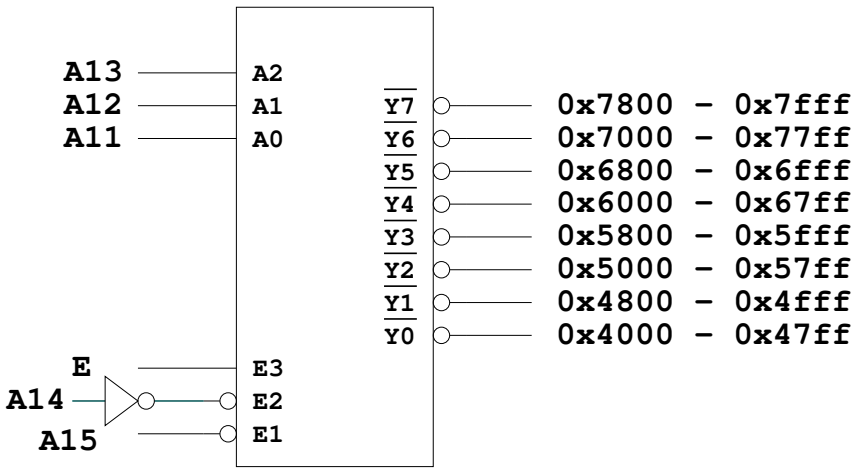


Can map external device into any unused space

When using Altera for address decoding, can select any desired address.

Can map using less expensive chip like 74HC13

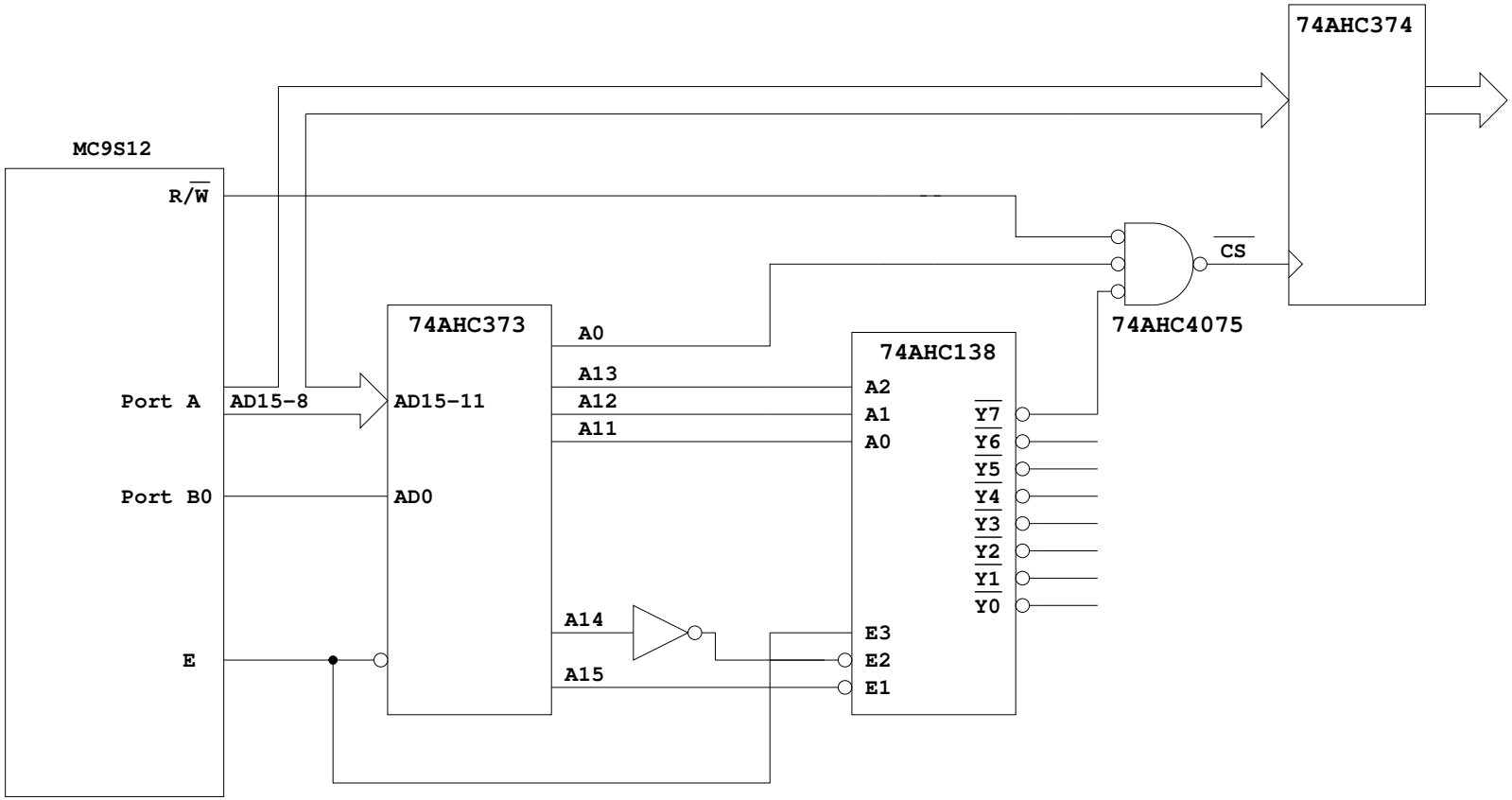
74HC138



Nothing is mapped into memory spaces occupied by outputs Y2 through Y7

A Simple Output Port

- We will use some MSI chips to implement a simple output port
- We will use a 74AHC374 chip as our output port — this is a chip which has 8 flip-flops
- We need to demultiplex several of the address lines. We will use another 74AHC374 chip to do this
- We will use a 74AHC138 decoder chip to select the output port
- For this example we will select the output port based on output Y7 from the 74AHC138. For the connections shown of the next page, A15 and A14 must be low, and A13, A12 and A11 must be high to select output Y7. Thus, the output port will be selected for addresses in the range 0x7800 to 0x7fff
- The address port needs to be connected to 8 bits of the address/data bus. We must connect it to either the high (even) byte (AD15-8) or the low (odd) byte (AD7-0). For this example we will connect it to the high byte
- We need to select the output port only when we write to an even address in the range 0x7800 to 0x7fff. In addition to address Y7 being active (low), the R/W line must be low (to indicate a write) and A0 must be low (to indicate an even address). We will use a 3-input OR gate to do this.
- Now, a write to any even address between 0x7800 and 0x7fff will write data to the output port



Simple output port for the HC12.

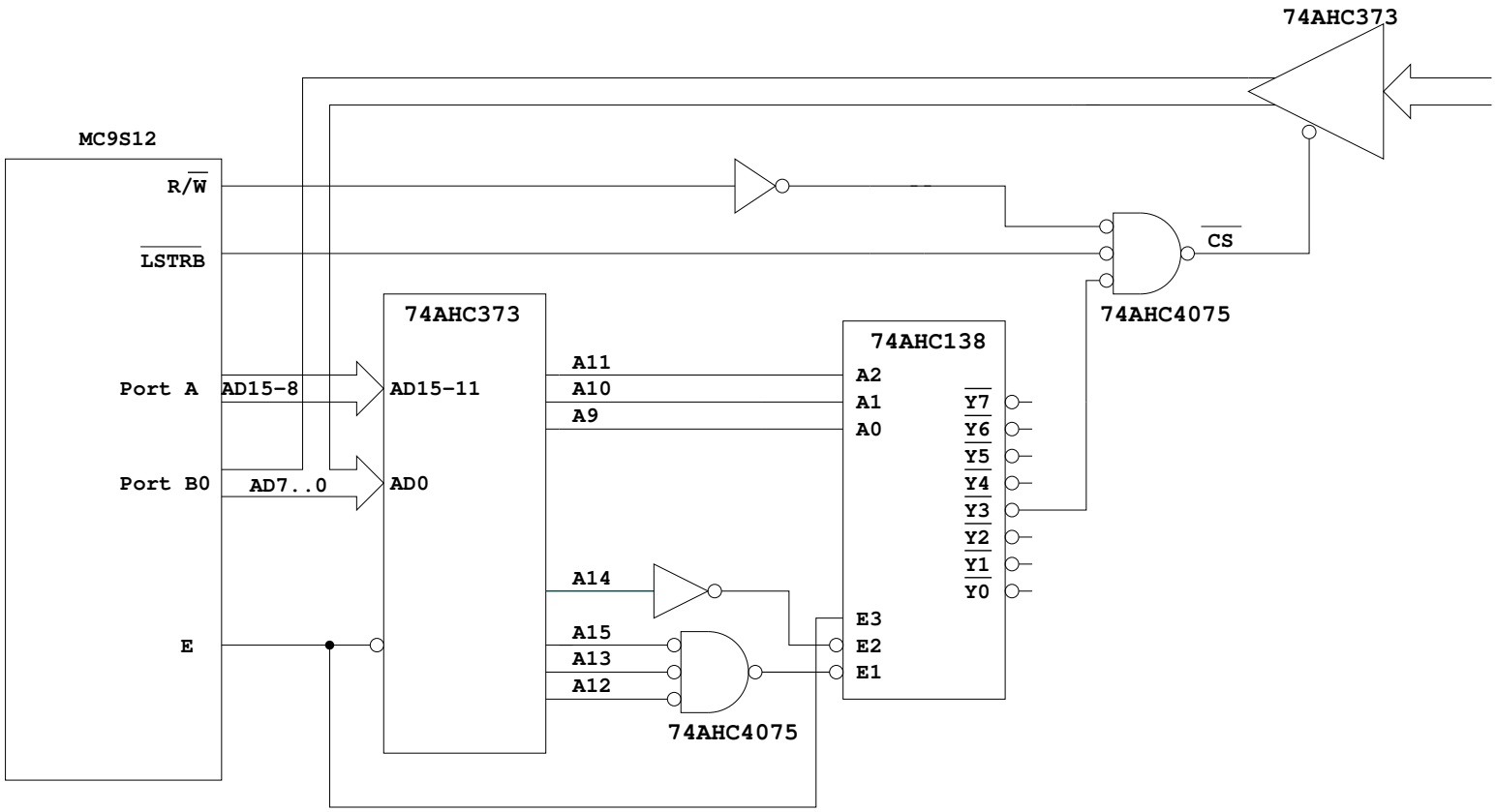
When address is between 0x7800 and 0x7fff and E is high, Y7 will go low

On a write to an even address in this range, CS will go low

Data on AD15-8 will be latched into port when CS goes high (when E goes low)

A Simple Input Port

- We will design a simple input port in a similar manner
- We will narrow the range of addresses used by using more high-order address lines to enable the 74AHC138. In the example shown, the address range of a single output is 512 bytes. For the ouput port example above, the range was 4 kB
- For an input port we need a tri-state buffer to drive the input data transparent latch (with tri-state outputs) for our buffer
- In this case we connect the tri-state buffer to the low (odd) data byte
- To access the chip for odd address, we need to select the chip only when LSTRB is low



Simple input port for the HC12.

When address is between 0x4600 and 0x47ff and E is high, Y3 will go low

On a read from an odd address in this range, CS will go low

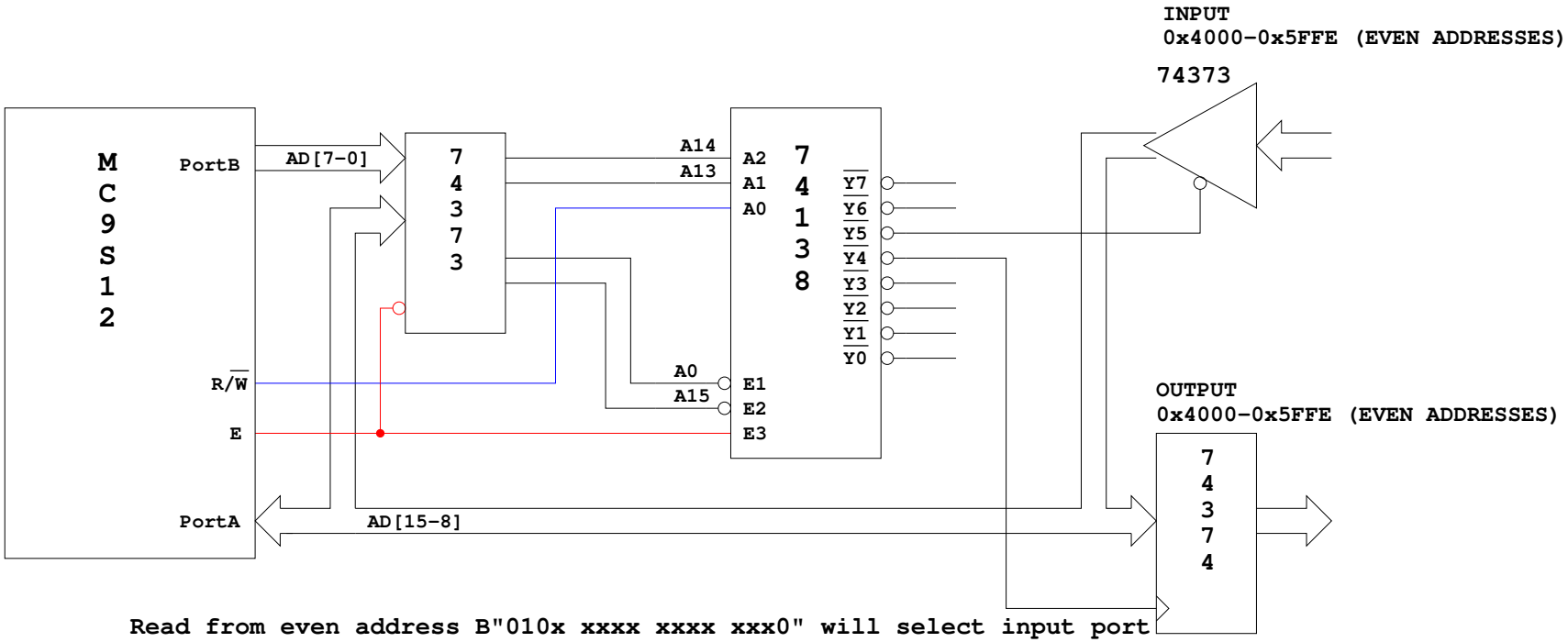
Data from the tri-state buffer will be driven onto AD7-0

The HC12 will latch the data into internal registers when E goes from high to low

A Simple Input-Output Port

- The next example shows another way to connect the 74138 chip to select input and output devices
- We connect A0 to one of the active low enable inputs. The 74138 will select a device only when A0 is low — only for even addresses
- We connect R/W to the low-order address input of the 74138
  - To select Y0, Y2, Y4 or Y6, R/W must be low. Thus, the devices selectec by Y0, Y2, Y4 and Y6 are ouput devices (devices the MC9S12 writes to).
  - To select Y1, Y3, Y5 or Y7, R/W must be high. Thus, the devices selectec by Y1, Y3, Y5 and Y7 are input devices (devices the MC9S12 reads from).

Simple Input and Output Ports



Read from even address B"010x xxxx xxxx xxx0" will select input port

Write to even address B"010x xxxx xxxx xxx0" will latch data in output port