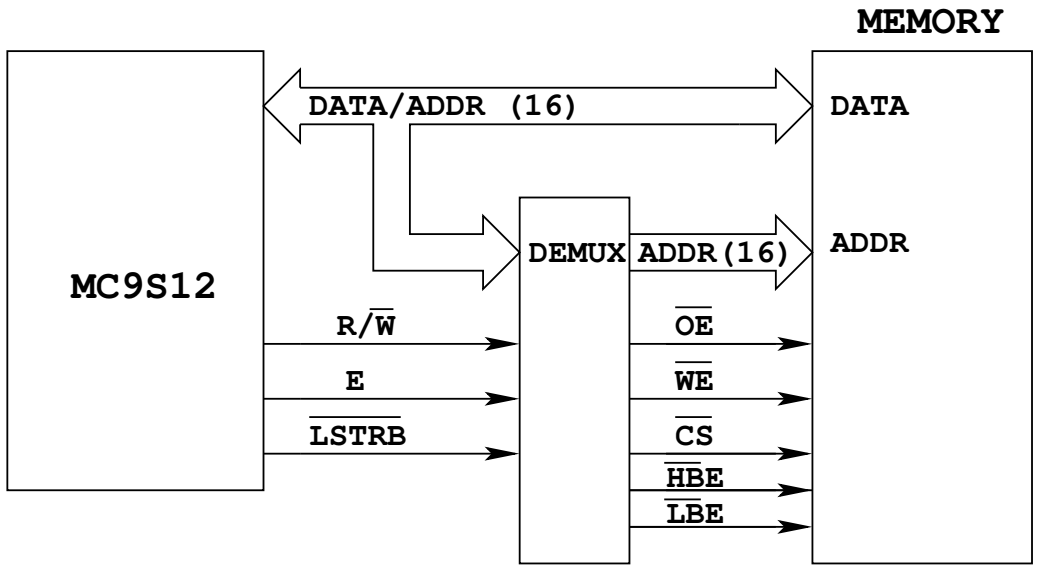
Multiplexed Address-Data Bus



$\overline{OE}$ – Output Enable (Read)

$\overline{WE}$ – Write Enable

$\overline{CE}$ – Chip Enable

$\overline{HBE}$ – High Byte Enable

$\overline{LBE}$ – Low Byte Enable

MC9S12 has 16-bit address and 16-bit data buses

Requires 35 bits

Not enough pins on MC9S12 to allocate 35 pins for buses and pins for all other functions

Solution: multiplex address and data buses

16-bit Bus: While E low, bus supplies address
While E high, bus supplies data

- The MC9S12 has a multiplexed address/data bus, AD15-0

- When in expanded mode the MC9S12 uses the pins for PORTA and PORTB as the multiplexed address/data bus
- When in expanded mode you cannot use PORTA or PORTB
- The MC9S12 can write one byte or two bytes in one memory cycle

Getting Into Expanded Mode

- The MC9S12 can operate in several modes:
  - Normal Single-Chip Mode (the way we have been using the MC9S12)
  - Normal Expanded Wide (16-bit address bus, 16-bit data bus)
  - Normal Expanded Narrow (16-bit address bus, 8-bit data bus)
  - Special Single Chip Mode (will not discuss)
  - Special Test Mode (will not discuss)
  - Emulation Expanded Wide Mode (will not discuss)
  - Emulation Expanded Narrow Mode (will not discuss)
  - Special Peripheral Mode (will not discuss)
- How does the MC9S12 know what mode to run in?
- There are two ways:
  - When you reset the MC9S12 it looks at the state of three external pins: MODA, MODB and MODC (BKGD)

    Based on the logic levels on these three pins the MC9S12 goes into one of the eight possible modes:

MODC	MODB	MODA	Mode
0	0	0	Special Single Chip
0	0	1	Emulation Expanded Wide
0	1	0	Special Test
0	1	1	Emulation Expanded Wide
1	0	0	Normal Single Chip
1	0	1	Normal Expanded Narrow
1	1	0	Peripheral
1	1	1	Normal Expanded Wide

  - You can write to the MODE register to change the mode

    In Normal modes you can switch operating modes once by writing to the MODE register

    In Normal modes the MODE register is a **write-once** register

Coming out of reset the MC9S12 checks the state of the following three pins, and starts in one of several modes:

BKGD	MODB	MODA	
0	0	0	Special Single Chip
0	0	1	Emulation Expanded Wide
0	1	0	Special Peripheral
0	1	1	Emulation Expanded Wide
1	0	0	Normal Single Chip (Flash EEPROM on)
1	0	1	Normal Expanded Narrow
1	1	0	Peripheral
1	1	1	Normal Expanded Wide (Flash EEPROM off)

On the EVBU, the MC9S12 starts in Normal Single Chip

After reset, can change modes by writing to MODE register

MODC	MODB	MODA	0	IVIS	0	EMK	EME	MODE 0x000B
1	0	0	0	0	0	0	0	Reset (Normal Single Chip)
1	1	1	0	1	0	0	0	Need for Normal Expanded Wide

In normal modes can change MODE once.

To switch from Normal Single Chip to Normal Expanded Wide, write 111 to MODC:MODB:MODA.

Also turn on internal visibility by setting IVIS bit to 1

IVIS = 1 tells the HC12 to display internal bus cycles on the external bus

```
BSET $000B,#$E8
```

Setting Up the MC9S12 For Expanded Mode

The PEAR Register

- We will start the MC9S12 in Normal Single Chip Mode, and write to the MODE register to switch it into Normal Expanded Wide Mode
- In Normal Expanded Mode the MC9S12 uses the E-clock, the R/$\overline{W}$ and $\overline{LSTRB}$ lines for control of the external bus
- In Normal Single Chip Mode, the MC9S12 does not use these control lines, so it sets up their pins for general purpose I/O
- You need to tell the MC9S12 to drive these signals onto external pins in
- When enabled these signals are driven onto Port E in order to use the MC9S12 in expanded mode
- You do this by writing to PEAR (Port E Assignment Register)
- PEAR is a write-once register

In Normal Single Chip mode, the E-Clock, LSTRB, and R_W lines are set up as general purpose I/O lines. When switched to Normal Expanded Wide need to make these control lines for expanded mode. Write to PEAR register:

NOACCE	0	PIPOE	**NECLK**	**LSTRE**	**RDWE**	0	0	**PEAR 0x000A**
0	0	0	**1**	**0**	**0**	0	0	**Reset Value (Single Chip)**
0	0	1	**0**	**1**	**1**	0	0	**Need for Expanded Wide**

Write 0 to NECLK to enable E-Clock on external pin
Write 1 to LSTRE to enable LSTRB on external pin
Write 1 to RDWE to enable R_W on external pin
Write 1 to PIPOE to indicate state of instruction queue on external pins

```
MOVB #$2C,$000A
```

The MISC Register

- The MISC (Miscellaneous) register needs to be set up to allow the MC9S12 to operate properly in Expanded Mode
- The MISC register allows you to change the number of clock streches used by the MC9S12
- The MISC register allows you to disable the Flash EEPROM
- The MISC register allows you to move the Flash EEPROM to address block 0x0000 to 0x7fff
- When reset the MC9S12 forces three E-clock stretches to external data accesses
- This is useful when using slow memory and peripheral devices
- We will use a fast peripheral device, so we will speed up external data accesses by forcing the MC9S12 to use no E-clock stretches
- We will leave the Flash EEPROM enabled at its normal address block of 0x8000 to 0xffff

0	0	0	0	**EXSTR1**	**EXSTR0**	**ROMHM**	**ROMON**	**MISC 0x0013**
0	0	0	**0**	**1**	**1**	**0**	**1**	**Reset Value (Single Chip)**
0	0	0	**0**	**0**	**0**	**1**	**1**	**Need for Expanded Wide**

Change EXSTR1:EXSTR0 from 11 to 00 to change from three E-clock stretches to no E-clock strech.

Leave ROMON at 1 to have DBug12 at 0x8000-0xFFFF
Change ROMHM to 1 to turn off Flas EEPROM from 0x4000 to 0x7FFF

```
MOVB #$03,$0013
```

Putting the MC9S12 into Normal Expanded Wide Mode

- To put the MC9S12 into Normal Expanded Wide you must write to the MODE, PEAR and MISC registers
- When D-Bug12 runs, it writes to these registers as part of its start-up code
- Because these three registers are write-once, you must write to them before DBug-12 does
- In order to switch to Normal Expanded Wide mode, we will put code to write to these registers in EEPROM and set the jumpers on the EVBU to run out of EEPROM
- In addition, it is necessary to do some setup which DBug-12 normally does – set the bus clock to 24 MHz.

```
main()
{
/* Set bus clock to 24 MHz */
    asm(" sei;");
    CLKSEL &= ~0x80;
    PLLCTL |= 0x40;
    SYNR = 0x05;
    REFDV = 0x01;
    while ((CRGFLG & 0x08) == 0) ;
    CLKSEL |= 0x80;

/* Put MC9S12 into wide expanded mode */
    MODE = 0xe8;              /* Expanded wide mode, IV on */
    PEAR = 0x0c;              /* Turn on R/W, LSTRB, E */
    EBICTL = 0x01;            /* Use E-clock to control external bus */
    MISC = 0x03;              /* No E-clock stretch, disable ROM from
                                 4000-7FFF */
```

One-Byte and Two-Byte Data Accesses in Normal Expanded Wide Mode

- In Normal Expanded Wide Mode the MC9S12 can access one byte or two bytes in a single memory cycle
- A single byte can be located at an even address or an odd address
- The data lines for bytes at even addresses are connected to AD15-8 lines
  - Because the data for even bytes are accessed through the upper 8 bits of the data bus, these are called Upper Bytes (or High Bytes)
- The data lines for bytes at odd addresses are connected to AD7-0 lines
  - Because the data for odd bytes are accessed through the lower 8 bits of the data bus, these are called Low Bytes
- The MC9S12 can access a High Byte and the following Low Byte in one memory access
  - For such a 16-bit access the 15 upper address lines are the same; only the lower address line is different
- The MC9S12 cannot access a Low Byte and the following High Byte
  - The 15 upper address bits are different for these bytes, and it would be difficult to build a decoder which could deal with this
  - For example, the addresses of the bytes at 0x7fff and 0x8000 are:

```
011111111111111   0x7fff
100000000000000   0x8000
```

One-Byte and Two-Byte Data Accesses in Normal Expanded Wide Mode

- To determine whether the MC9S12 is trying to access a single High Byte, a single Low Byte, or two bytes (High Byte and following Low Byte) the MC9S12 uses the A0 address line and the $\overline{\text{LSTRB}}$ control line
- $\overline{\text{LSTRB}} = 0$ means access low (odd) byte
- A0 = 0 means access high (even) byte

$\overline{\text{LSTRB}}$	A0	Access Type
1	0	8-bit access of even address (high byte)
0	1	8-bit access of odd address (low byte)
0	0	16-bit access of even address (high and low byte)
1	1	Cannot occur

- Note: $\overline{\text{LSTRB}} = 1$ and A0 = 1 would imply access of low (odd) byte and following high (even) byte. This will not occur for external data accesses on the MC9S12
- If the MC9S12 needs to access a 16-bit number from an odd address, it will do this in two memory cycles — it will access the 8-bit number at the odd address, followed by the 8-bit number at the even address
- For example the instruction

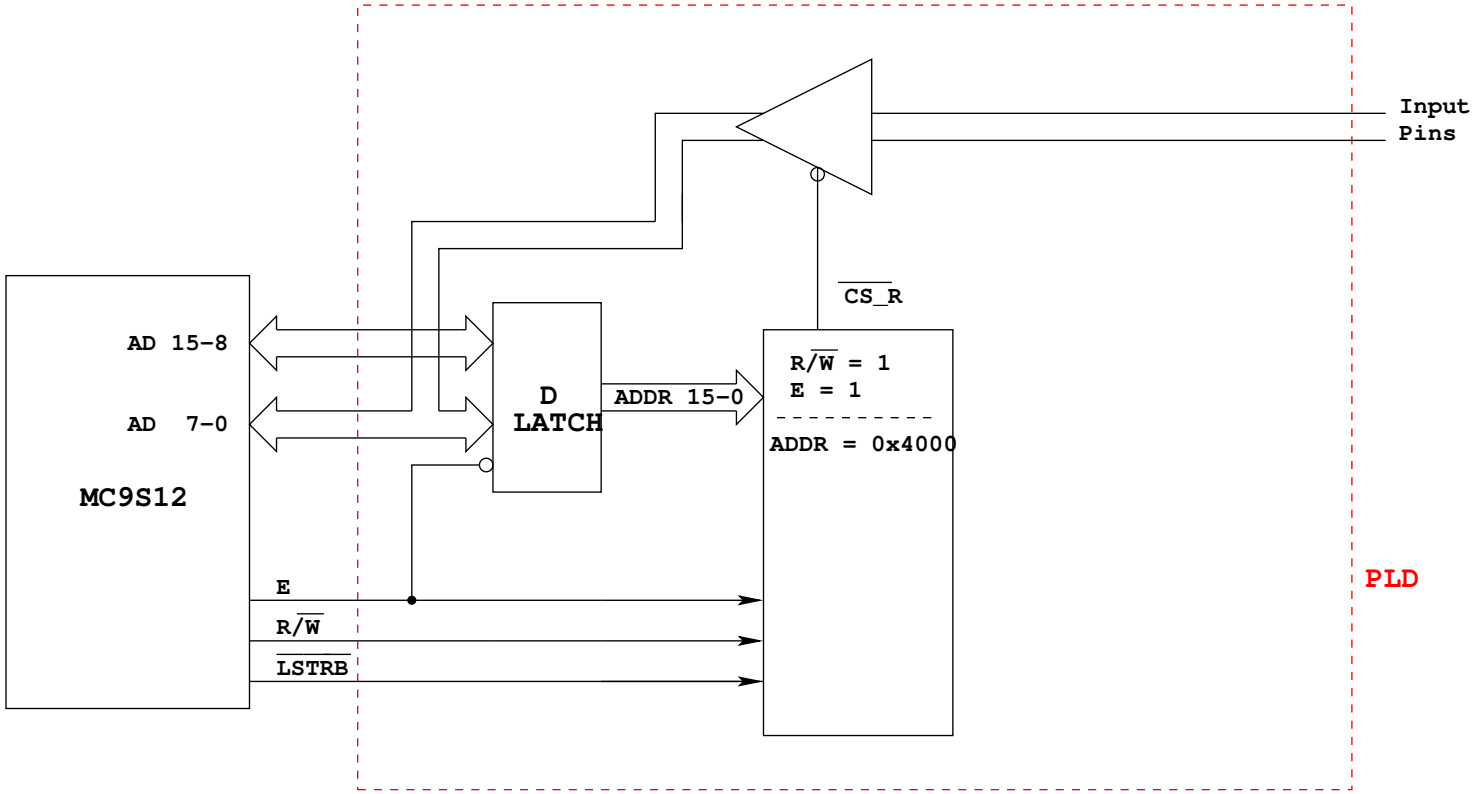
```
movw #$55aa,$08ff
```

  will write the 0x55 to memory location 0x08ff on one memory cycle, and the 0xaa to memory location 0x0900 on the next memory cycle.

The MC9S12 in Expanded Wide Mode

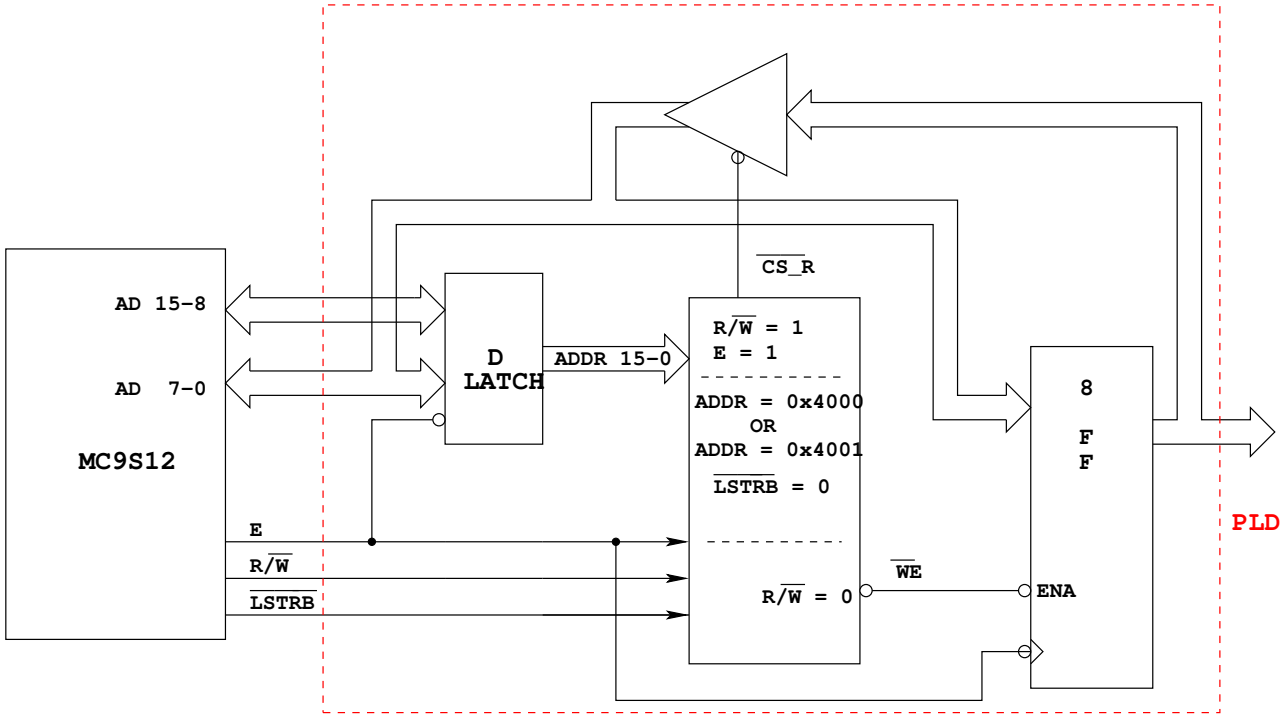
- In expanded mode the MC9S12 can communicate with external memory and devices over the multiplexed address/data bus
- To connect an external device to the MC9S12 bus you need to identify an unused region of memory
- By turning off some of the flash EEPROM, the region from address 0x4000 to address 0x7FFF is available.
- We will map an output port to address 0x4001
- Need decoder to demultiplex address from data, and to generate control lines needed by output port

Input Port for Lab 5



Reading from address 0x4000 (ADDR = 0x4000, R/W high, E high) will bring CS_R low
This will drive the data from the input pins onto the data bus
The MC9S12 will read the data on the input pins on the high-to-low transition of the E-clock

Output Port for Lab 5



Writing to address 0x4001 (ADDR = 0x4000 or 0x4001, LSTRB low, R/W low) will bring WE low.
On the high-to-low transition of E with WE low, the data is latched into the flip-flops

Reading from address 0x4001 (ADDR = 0x4000 or 0x4001, LSTRB low, R/W high, E high) will bring CS_R low
This will drive the data from the flip-flops onto the data bus
The MC9S12 will read the data on the flip-flops on the high-to-low transition of the E-clock

Expanded Ports for Lab 5

- For Lab 5, you need to write a Verilog program which will allow you to use your EE 231 FPGA board to implement input and output ports. The following lines from the MC9S12 will be connected to the FPGA: A/D 15-0, E, R/$\overline{\text{W}}$, and $\overline{\text{LSTRB}}$. An 8-bit output port at address 0x4001 will be connected to the LEDs of the EE 231 board, and a four-bit input port at address 0x4000 will be connected to the four switches. The program should be fairly easy to write, using the code fragments in the notes. Here is a start. The signal `cs_r_4000` will be active when reading from address 0x4000, the signal `cs_r_4001` will be active when reading from address 0x4001, and the signal `cs_w_4001` will be active when writing to address 0x4001.

```
module port_expansion (
    input e, rw, lstrb,
    inout [7:0] port_a, port_b,
    input [3:0] sw,
    output reg [7:0] exp_port
    output cs_w_4001, cs_r_4000, cs_r_4001;
);

reg [15:0] address

...

endmodule
```