Lab 5 – Part 1
Final Project: Port Expansion for the MC9s12

In this sequence of labs you will learn how to interface with additional hardware and implement a motor speed control system. At the end of this sequence you will have to write a report that will be handed separately.

Introduction and Objectives

It is sometimes necessary to add additional memory and/or hardware to a microprocessor or microcontroller. While interfaces such as the SPI allow you to add some hardware, it is often necessary to interface directly to the address/data bus. For a microprocessor, which does not have built-in peripherals, the address/data bus is the only way to add additional memory or hardware. In this lab you will add an extra output port to your MC9S12.

Figure 1 shows a block diagram for adding an external input port and output port at address 0x4000 to the MC9S12 microcontroller. For the lab, just implement the output port. Connect the eight bits of the output port to eight LEDs from your EE 231 lab kit.

1. Prelab

Look up the pinouts for the three 74AHC MSI chips on the web. Draw a schematic for connecting the three chips to eight LEDs to implement the output port.

2. The Lab

1. Wire up the output port, using the schematic from the pre-lab as a guide. Note that there will be a lot of wires to run, so it is essential that you are neat in your wiring.

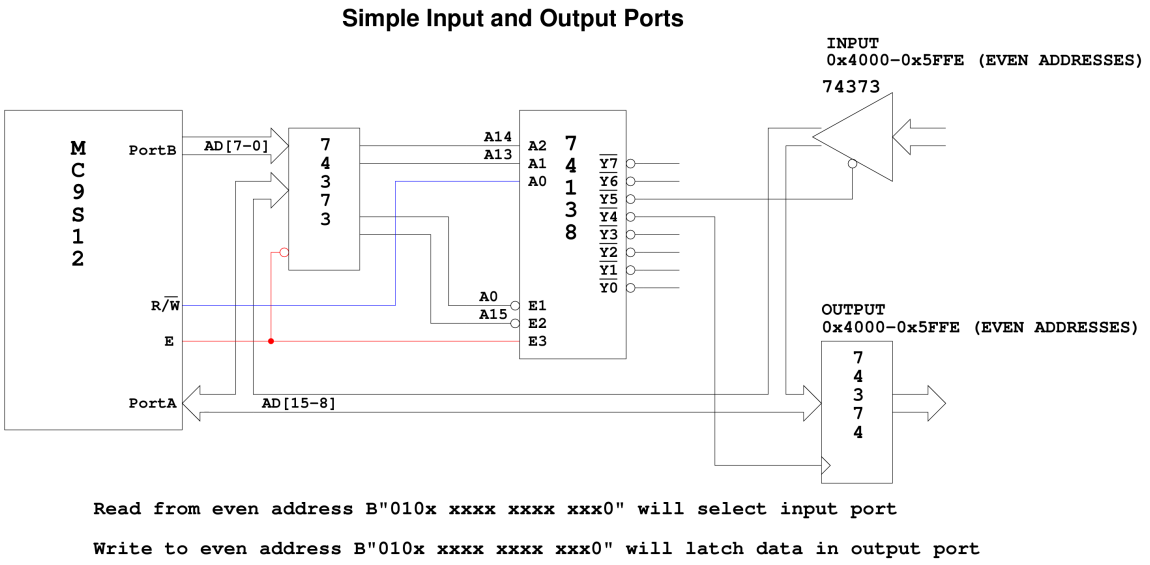

Figure 1: Block diagram of HCS12 output port at address 0x4001

2. Check the functioning of your port using D-Bug12. When you start your MC9S12 using D-Bug12, the microcontroller is in single chip mode. In this mode you can manipulate AD-15-0, E, R/W and LSTRB as general purpose I/O lines. You can use the MM command of D-Bug12 to write data to the output port by changing AD15-0 (PORTA and PORTB), E, R/W and LSTRB in the same sequence that the MC9S12 would if it were in expanded wide mode.

(a) Use the DDRE register to make E, R/W and LSTRB output pins. (Note: E is bit 4 of PORTE, R/W is bit 2 of PORTE, and LSTRB is bit 3 of PORTE.
(b) Bring E low by writing to PORTE.
(c) Put 0x4000 on PORTA and PORTB.
(d) Bring R/W and LSTRB low.
(e) Bring E high.
(f) Put the data you want to write to the port on PORTB.
(g) Bring E low.

3. The program below can be put into EEPROM so you can run your board in wide expanded mode. To get into wide expanded mode, you will have to put this program into EEPROM starting at address 0x0400, and then set DIP Switch SW7 so you run your EEPROM program rather than DBug12. (You cannot run in expanded wide mode using DBug12, since DBug12 uses the Flash EEPROM in the region 0x4000-0x7fff.)

```
#include <hidef.h>                          /* common defines and macros */
#include ”derivative.h”                 /* derivative−specific definitions */
#include <stdio.h>
#include <termio.h>
#define OUT_PORT (*( volatile char *) 0x4000 )
#define BIT7 0x80
#define BIT6 0x40
#define BIT5 0x20
#define BIT4 0x10
#define BIT3 0x08
#define BIT2 0x04
#define BIT1 0x02
#define BIT0 0x01
void interrupt RTI_isr ( void ) ;

main ( )
{
        volatile unsigned int i, j;
        /* Set bus clock to 24 MHz */
        asm( sei );
        CLKSEL &= ˜0 x80;
        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 */
        DDRP = DDRP | 0x0F ;    /* Make 4 LSB of Port P outputs */
        PTP = PTP | 0x0F;               /* Turn off seven−seg LEDs */

        for ( ; ; ) {
                for (i = 0; i < 50000; i = i + 1)
                        for (j = 0; j < 10; j = j + 1)
                                OUT_PORT = OUT_PORT + 1;
        }
}
```

4. An MC9S12 with a functioning expansion port will be available at one of the logic analyzers during lab this week. The HCS12 is running the following loop:

```
                org $0480
loop:           ldx $4000
```

```
inc $4000
ldaa $4000
bra loop
```

The label loop is at address 0x0480.

(a) Hand-assemble this program to determine the op codes and op code addresses.

(b) Use the logic analyzer to grab data from the HCS12 address/data bus. Identify the memory cycle which reads data from address 0x4001, and the memory cycle which writes data to address 0x4001. The HCS12 address/data bus uses 19 lines – AD15-0 and the three control line E, R/W, and LSTRB. The HCS12 will either be fetching instructions from EEPROM (address 0x0400-0x0fff), or accessing the external port at 0x4001. Thus, adress bits D15, D13 and D12 will always be zero. These three lines will not be connected to the logic analyzer. Figure A-9 of the MC9S12DP256B Device Users Guide shows the external bus timing. As best you can, measure the following times. The numbers in parentheses are the labeled numbers on Figure A-9 and Table A-20. Compare the numbers to the values listed in Table A-20.

i. Cycle time (2)
ii. Pulse width, E low (3)
iii. Pulse width, E high (4)
iv. Address delay time (5)
v. Muxed address hold time (7)
vi. Write data hold time (13)
vii. Read/write delay time (24)
viii. Read/write hold time (26)
ix. Low strobe delay time (27)
x. Low strobe hold time (29)