EE 308 – LAB 4

Using the HCS12 Parallel Ports

Introduction

Bring your digital breadboard to this week's lab! You will need to use the LEDs on it.

In this week's lab you will write an assembly-language program to display various patterns on the LEDs of your breadboard. You will use the HCS12's Port A as an output port to display the LED patterns, and Port B as an input port to decide which pattern to display.

Ports A and B are the easiest HCS12 parallel ports to understand and use. For this week's lab, you will create programs to write to Port A, and read from Port B. You will use information from switches connected to Port B to control the pattern you output on Port A. You will test your programs by connecting Port A to 8 LEDs, and vary the Port A output by changing the switch settings connected to Port B.

Pre-Lab

Write a program to set up Port A as an 8-bit output port, and to implement (i) a binary down up, (ii) a flasher, (iii) a psuedo-random code generator, and (iv) a turn signal on Port A. Samples from the sequences that you should generate are shown in Fig. 1. You will use eight LEDs to see the Port A output. Include an appropriate delay (about 200 ms) between changing the LED pattern so that you can easily and comfortably see them flash. Use a subroutine to implement the delay. Also, set up Port B as an 8-bit input port, and use Port B bits 4 and 1 to control which of the Port A functions are performed as shown in Fig. 2. When you switch between functions, the new function should start up where it ended when it was last activated, so set aside variables to save the states of the various patterns.

Write the program before coming to lab. Be sure to write the program using structured, easy-to-read code. Be mindful of the delay requirement before changing the display or reading the Port B pins to determine if you should switch to a new function.

The Lab

1. Run your program on the `ZAP` simulator. (Note that you will want to reduce the delay considerably before running on the simulator). You can simulate different input values on Port B by changing the value in Address `$0001`. If you have difficulty getting your program to work, start by trying to implement one function only — say, the down counter. Once this works, start working on your next functions.
2. Set a breakpoint at the first line of your delay subroutine. When the breakpoint is reached, check the value of the stack pointer, and the data on the stack. Make sure you understand what these mean.

1. A binary down counter:

```
· · · · · · · ·
• • • • • • • •
• • • • • • • ·
• • • • • • · •
```

Continue counting down.

2. A flasher:

```
· • · • · • · •
• · • · • · • ·
```

Repeat the above sequence.

3. Generate a pseudo-random code in the following manner: treat ACCA as an eight-bit shift register which shifts to the right. The number to input into the left-hand side of the shift register is the XOR of Bit 0 and Bit 4 of ACCA. (You can use the instruction RORA to do this.)

```
· · · · · · · •
• · · · · · · ·
· • · · · · · ·
· · • · · · · ·
· · · • · · · ·
• · · · • · · ·
· • · · · • · ·
· · • · · · • ·
· · · • · · · •
```

Repeat the above sequence. This produces a sequence which repeats after twelve cycles. By taking the XOR of bits 0, 1, 6 and 7 you can generate a sequence which repeats every 255 cycles, and appears to be random. (For your program looks at bits 0 and 4 rather than using a table of twelve values.)

4. A Ford Thunderbird style turn signal that alternates between right and left:

```
· · · · · · · ·
· · · • · · · ·
· · • • · · · ·
· • • • · · · ·
• • • • · · · ·
· · · · · · · ·
· · · · • · · ·
· · · · • • · ·
· · · · • • • ·
· · · · • • • •
```

Repeat the above sequence.

Figure 1: Samples of the functions to be performed using Port A as an output. "•" is LED on and "·" is LED off.

PB4	PB1	Port A Function
0	0	Down counter
0	1	Flasher
1	0	Pseudorandom Code
1	1	Turn Signal

Figure 2: Port B inputs to control the Port A functions.

3. After your program works on ZAP, load it into your HCS12. Wire up Port A to 8 LEDs, and Port B bits 4 and 1 to a connection that can be switched between 5 volts and ground. Use the switches on your breadboard to connect to the Port B pins.

   Begin by wiring up the Port B pins to switches, and make sure you can read the state of the switches. You can do this using D-Bug 12 to display the contents of Address `$0001` for several values of the switch settings. After you are sure you can read the state of the switches on Port B, try running your program on the HCS12.

   When you get your program to work, have your lab instructor or TA verify the program operation.

4. Set a breakpoint at the first line of your delay subroutine. When the breakpoint is reached, check the value of the stack pointer, and the data on the stack. These should match what you found in Part 2 of the lab.

5. The HCS12DP256 has EEPROM (Electrically Erasable Programmable Read Only Memory) between `0x0400` and `0x0FFF`. If you put your program into EEPROM the program will remain there when you turn off power. To put your program into EEPROM, all you need to do is change the starting address of your program to `0x0400`. Note that you will want to store the array which has the turn signal patterns in EEPROM (so the array will not disappear when you turn off power). To do this include any tables of constant patterns in the CODE section of your program. You will want variables which will change as the program is executed to be placed in RAM, say at location `0x2000`, as usual.

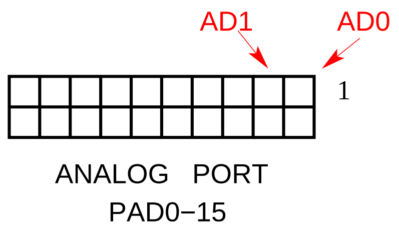
   When D-Bug12 starts running, it checks the state of pins AD0 and AD1. If these are both low, D-Bug12 runs normally. If, however, AD0 is high and AD1 is low, D-Bug12 will immediately jump to your program in EEPROM. Thus, you can run a program without having to have the HCS12 connected to a serial port to receive a D-Bug12 command. In order for the HCS12 to run properly, however, your program must do some hardware setup which is normally done by D-Bug12. To set up the hardware properly, your program needs to execute the following instructions. The reasons for doing this will be discussed later in the semester.

```
ldaa     #$55       ; Reset COP
staa     $003f
coma
staa     $003f
clr      $003c      ; Turn off COP
ldab     #$11       ; Map RAM into proper location
nop
stab     $0010
```

```
        ldab     #$00       ; Set clock reference divider to 0
        stab     $0035
        ldab     #$05       ; Set PLL to multiply oscillator clock by 6
        stab     $0034
        nop                 ; wait
        nop
        nop
        nop
l1:     brclr    $0037,#$08,l1    ; Wait for PLL to lock
        bset     $0039,#$80       ; Switch to PLL clock
```

Add the above instructions as the very first instructions in your program. Change the address of your CODE section to `0x0400`. Be sure to do this in both the program file and the `lkf` file. Load the program as you have done in the past. Use the `ASM` command of D-Bug12 to verify that the code has been loaded into address `0x0400`.

Connect a 10 kΩ resistor from `GND` to `AD1` and a 10 kΩ resistor from `+5 V` to `AD0`. Verify that your program runs correctly without reloading after cycling the power on your HCS12.


AD1
AD0
1
ANALOG PORT
PAD0−15