Assignment 5: Serial Line and Photocell Bargraph

out Mon Oct 6
due Wed Oct 15

INTRO

In this assignment, we will continue work with the HC11, learning about serial communications and the ASCII character set, and using the analog-to-digital (A/D) converter.

First up is demonstration of BootLoad, an improved program for loading assembled HC11 code into the chip.

BOOTLOAD

In Assignment 4, you hand-copied the HC11 program you were interested in running into an instance of the "HC11Boot" source file. This is painstaking and annoying. In fact, of the students in the class took it upon himself to improve the tools to solve this problem.

Here is my version of the same idea. The new process for getting your code into the HC11 is the following:

1. Write your HC11 code in to a source file with a "`.s`" suffix.
2. Run the `as6811` assembler on your `.s` file, giving it the `-ol` flag. This will produce a "relocatable" file with a `.rel` suffix containing the object code bytes to be downloaded.
3. Use the new `BootLoad.java` program to download the `.rel` file to your HC11.

Following is a walk-through.

Start off by downloading `BootLoad.java` from the course website. Also, redownload `Serial.java`—I have made a slight change to it.

Now compile both these files:

```
javac BootLoad.java Serial.java
```

Next, assemble an HC11 source file with the flag to produce the `.rel` file. For instance, assemble the `beep.s` program from the last assignment:

```
as6811 -ol beep.s
```

This should produce a `beep.rel` program, with contents like this:

```
XH2
H 1 areas 1 global symbols
S .__.ABS. Def0000
A _CODE size 11 flags 0
T 00 00 CE 10 00 1C 00 10 4A 26 FD 1D 00 10 4A
R 00 00 00 00
T 00 0D 26 FD 20 F2
R 00 00 00 00
```

The lines starting with "T" contain the object code. The first two bytes in these lines are the address

where the code goes; the remainder of the bytes on these lines is the object code itself.

Finally, we are ready to download the beep code into the HC11. Do with with the new BootLoad:

```
java BootLoad beep COM1
```

The program will then download the code from `beep.rel` into the HC11, producing the following output:

```
Using serial port COM1
Read 17 bytes of code from file beep.rel
Serial port is open.  Turn on HC11 and press <ENTER> to continue...
Writing to serial line...
done.
```

OK. Now we're ready to more productively code on the HC11.

PROBLEM 5–1: INTRO TO SERIAL COMMUNICATIONS

In this exercise, we will learn how to transmit ASCII data over the serial line from the HC11, and receive and display it on the PC.

The following program `serialxmit.s` (available on the course website) will cause the HC11 to repeatedly transmit the ASCII character set beginning with code 0x21 (the exclamation point, '!') and ending with code 0x5a (the capital letter 'Z'):

```
;;; serialxmit.s
;;; counts from 0x21 to 0x5a on 68HC11 serial port and then repeats

base=   0x1000
scsr=   0x102e                  ; serial comms status reg
scdr=   0x102f                  ; serial comms data reg

        ldx #base               ; pointer to register base
loop:   ldaa #0x21              ; start with ASCII 0x21 to transmit
xmit:   staa <scdr,x            ; transmit reg a
        brclr <scsr,x,#0x40,.   ; loop here until byte transmitted
        inca                    ; inc char to be transmitted
        cmpa #0x5B              ; see if it's beyond ASCII 0x5a
        bne xmit                ; no, send the next one
        bra loop                ; reset char to 0x21
```

Let's look at the program in some detail.

The program begins by installing the value 0x1000 into the X register (the label `base` was previously assigned as 0x1000). This allows indexed-by-X instructions to be used to talk to the serial control registers.

Next, the value 0x21 is loaded into the A register. This will be the first byte that is transmitted.

At the next line, beginning with the label `xmit`, the value in the A register is written to the `scdr` (serial communications data register). This will cause the value in A to be written out the serial port.

Note that the indexed-by-X addressing mode is used. This requires a one-byte value which is added to the value in X to form the extended address. The less-than-symbol < is an operator in the assembler which takes the low byte of a label's value. Hence, the low byte of the `scdr`, or 0x2f, is taken and added to the 0x1000 in X to point to the `scdr` register.

To avoid overrunning the register, it is necessary to check a "busy" bit in the `scsr` (the serial communications status register). The next instruction does this. This is the "branch-if-bit-clear" instruction, which uses an indexed-by-X addressing mode. The `<scsr` when added to X points to the status register. The `#0x40` operand indicates we are interested in testing the 2nd highest bit in this register (which, if you refer to the HC11 documention, is the Transmit Complete flag, with a 1 indicating it is complete). Finally, the dot at the end of the instruction is the address to which to jump. In the assembler syntax, the dot has a special meaning, which is: the address of the beginning of this instruction. Thus, the `brclr` will literally jump to itself if it bit is clear, which means that the serial transmit is still underway. Hence, we will be stuck here in this instruction until the transmit is done.

The next instructions increment the A register and loop in the appropriate way. After being incremented, the A register is compared with the value 0x5b (one higher than the last value we want transmitted). The compare value performs a subtraction (A – 0x5b, in this case) and then throws the result away. But the compare does set the condition code bits, and next we test if the result was "not equal to zero," using the `bne` instruction. If it wasn't zero, we want to transmit this next value of A, so we jump to the `xmit` label. If it *was* zero, then A must have been 0x5b, and we don't want to transmit that, so we jump to the `loop` label, which resets A to 0x21 before transmitting again.

The `serialxmit.s` program is available on the course website. Assemble it to produce the `.rel` object file, and then download it to the HC11 using `BootLoad`.

Now, you need to run a terminal emulator on your PC in order to see what the code is doing. You may either (1) use a standard terminal emulator on your PC (e.g., HyperTerminal), with serial settings 1200-N-8-1 and no flow control, or (2) compile and run `BootTerm.java` (available on the course website). This is a version of the `BootLoad` program that, after booting the HC11, runs a tiny terminal program with the correct settings.

Regardless of the terminal emulator method, when you have it all working, you should see the following on your screen:

```
!"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ!"#$%&'()*+,-./0123456789:
;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ!"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRST
UVWXYZ!"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ!"#$%&'()*+,-./01234
56789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ!"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMN
OPQRSTUVWXYZ!"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ!"#$%&'()*+,-.
/0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ
```

etc.

Does it work? Make sure you have the HC11's serial transmit line (pin 43 of the chip) plugged into the "xmt" port of your dev board, not LED8!

There is nothing to turn in for this exercise, but you can't really go on to the next two problems until you have this working.

PROBLEM 5–2: PORTC SERIAL DISPLAY PROGRAM

Using the techniques presented in `serialxmit.s`, write an HC11 program that reads the value of the Port C register, converts it into an ASCII representation of its hexadecimal value, writes those two ASCII characters out the serial port, and then repeats. Also, between each "printing" of the Port C value, output the binary values 0xA followed by 0xD out the serial port. (These are the linefeed and carriage return characters, respectively, and will cause the terminal emulator to display your hex values each on their own line, rather than sprawled across the screen.)

Some notes on this exercise:

- Connect the eight Port C pins to your switch bank (pushbuttons and slide switches), so you can readily generate different values on Port C.
- The Port C register defaults as inputs in the HC11's bootstrap mode, so you can simply read the PORTC register (no other set-up is required) to read the pins.
- There is a sample ASCII table on page 58 of the Motorola M68HC11ERG document.
- Converting the Port C value to hexadecimal is relatively straightforward. You convert the high "nybble" (upper four bits) into the correct ASCII digit from 0 to 9 or A to F, send it out, and then convert the lower nybble and set it out.
- Masking operations (e.g., "ANDA #0xF0") and bit shift operations (e.g., "LSRA") should be helpful.
- Don't read the Port C register twice, first transforming the high nybble and then reading it again and transforming the low nybble! This could introduce errors if the Port C value changed between processing of the high nybble and the low nybble. Read it once and cache it between these two operations.
- You may wish to solve this problem incrementally rather than writing the whole entire code and then trying to get it to work! E.g., get the high-nybble conversion working and printing, then do the low nybble.

What to Turn in

Provide a copy of:

1. The assembled listing file (.lst) of your final HC11 code.

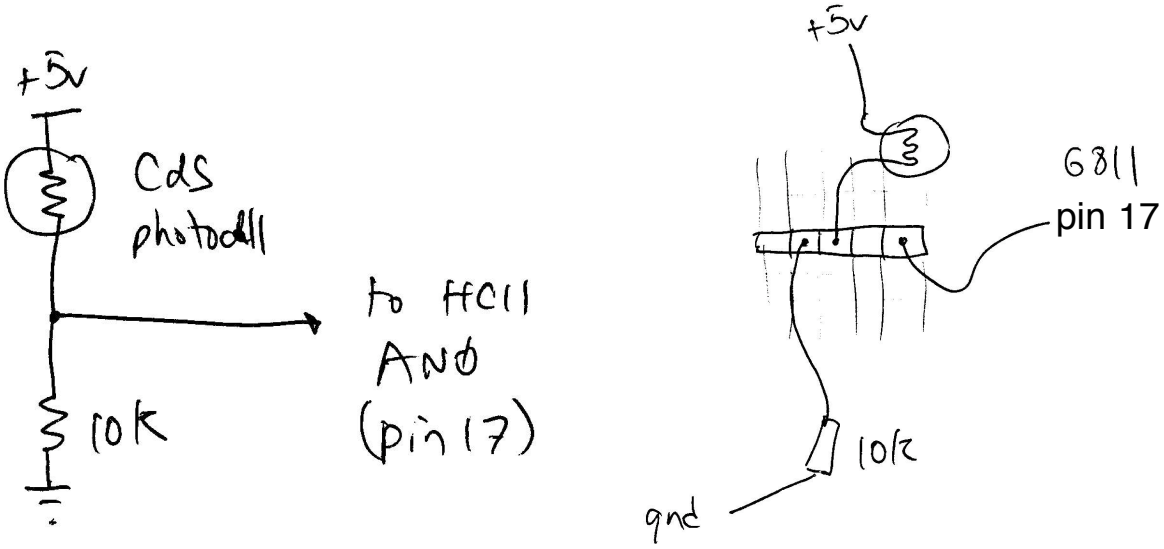
2. A print-out of a screen-snapshot of the results, showing you got it working. The screensnap should show the pixels on your PC's display, not a cut-n-paste of the characters being printed to the terminal emulator.

PROBLEM 5–3: SERIAL LINE BARGRAPH LIGHT SENSOR

In this exercise, you will construct a voltage divider using a light-sensitive photocell, and wire its output into one of the HC11 analog-to-digital (A/D) inputs. Using provided sample code, you will write a program that repeatedly displays the converted analog reading as a bargraph of 0 to 15 asterisks.

How To

The photocell is wired in a "voltage divider" circuit as shown. The diagram on the left is the electrical schematic; on the right is a visual wiring guide.



The voltage divider generates an output voltage that is a function of the ratio of the two resistances in the legs of the converter. In this instance, the photocell is in the upper leg. Its resistance decreases with increasing light, causing a higher voltage result in this case.

Sample Code

The file `analogdemo.s` demonstrates how to initialize the HC11's analog-to-digital converter and make a conversion. The core functionality in the following code snippet:

```
        bset <OPTION,X,#0x80    ; set high bit of option -> turn on A/Ds

loop:   ldaa #0                 ; select channel 0
        staa ADCTL              ; start conversion
donelp: ldaa ADCTL
        bpl donelp              ; if high bit set, it's done
        ldaa ADR1               ; grab result!
```

The full sample program repeatedly makes analog-to-digital conversions and prints to the serial line an ASCII representation of their hex value.

How to Go About It

To verify that your hardware is functioning properly, it is suggested that you first build the circuit and run the supplied `analogdemo.s` demo program. The demo program will repeatedly make analog

readings and transmit their value, represented as a single ASCII character, out the serial line.

Features

Your solution must:

- demonstrate the use of subroutines. Make sure to initialize the stack pointer at the beginning of your code.
- repeatedly print the bar graph of 0 – 15 asterisks based on the high nybble of the converted value, with the linefeed and carriage return characters between each bar. You should see a sideways scrolling bar graph on your terminal emulator when it is working.

For extra credit, you can generate a graph of 0 – 31 asterisks using the upper five bits of the converted value.

What to Turn In

Hand in your assembled .lst file and a screen capture of your program in action.

If You Don't Have a Photocell

If your kit does not contain a photocell, you can use a potentiometer (also known as a volume knob, available in lab). This is a three terminal resistor with a center tap. Wire the one of the outer terminals to +5v, the other outer terminal to ground, and the variable center tap terminal to the HC11's analog input.

PROBLEM 5–4: LIGHT-CONTROLLED OSCILLATION

FOR HONORS/GRAD STUDENTS OR FOR EXTRA CREDIT

Using the techniques shown in this assignment and the previous one, create a program that will oscillate, with the frequency of oscillation continuously varying based on the value of the converted photocell reading.

Scale the frequency of oscillation so that it varies over the audible range of about 100 Hz to about 5 kHz as the photocell value varies between 0 and 255. (You may choose as to whether small readings correspond to low frequencies or high frequencies.)

What to Turn In

Turn in your program, and an analysis that shows the frequency that corresponds to analog readings of 0, 128, and 255.