

ASSIGNMENT 4: INTRODUCTION TO THE 68HC11

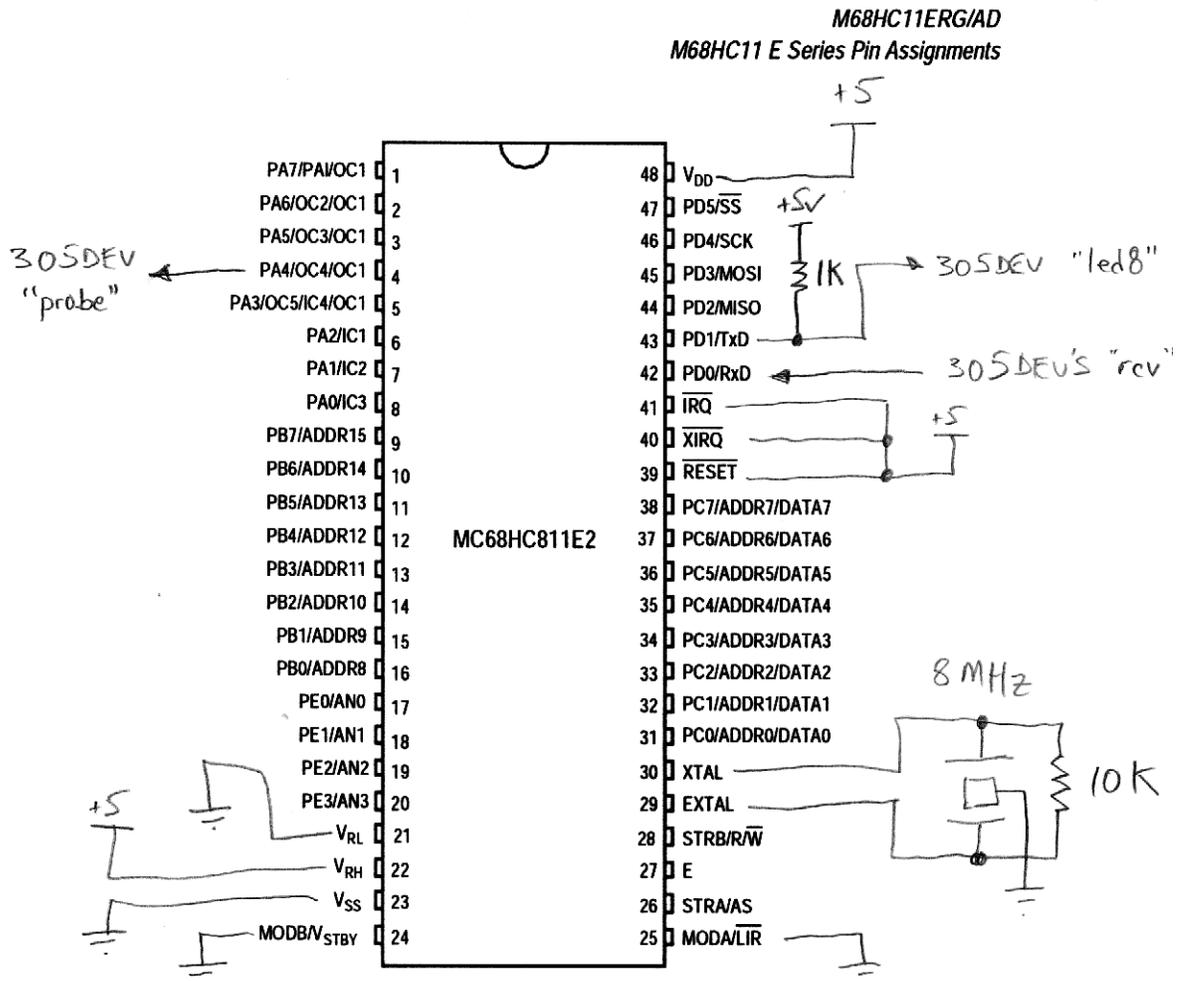
Out: Monday, Sep 29, 2003
Due Monday, Oct 6, 2003

Introduction

In this assignment, you will (1) construct a minimal 68HC11 system on your breadboard, and (2) use the serial port of a computer to load programs into the HC11's internal 512 bytes of RAM (making use of the serial bootstrap feature of the CPU).

Initially, you will load a known-working program into the HC11 to verify that your wiring setup and serial line configuration is working. Afterward, you will modify and write new programs for the HC11 and use your setup to load them into the HC11 for testing and debugging.

Hardware Setup: HC11 Wiring



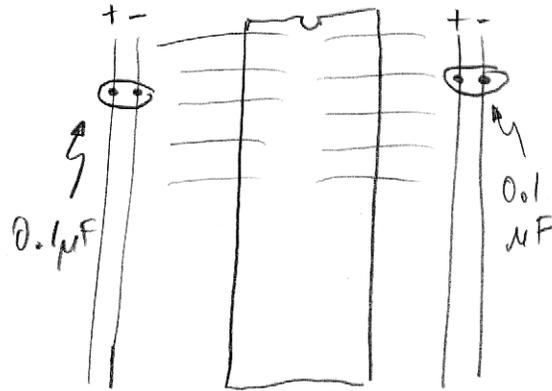
91.305 Assignment 4: Introduction to the 68HC11

The previous drawing shows how the 68HC11 chip should be wired. Here are a few notes of interest:

- Power is applied via Vdd (pin 48); ground is applied at Vss (pin 23).
- The pins \sim RESET, \sim XIRQ, and \sim IRQ must all be tied high.
- The analog input references should be connected: VRL to ground, and VRH to +5v.
- The MODA (pin 25) and MODB (pin 24) lines must be tied to ground. This put the HC11 into the serial bootstrap mode when it is powered on or reset.
- There is a 1K pullup resistor on the HC11's TxD output. This is necessary because when the HC11 starts up in bootstrap, the pin can only assert ground, and must be pulled high to produce a logic one ("wired OR mode").
- Please observe the following notes on the ceramic resonator:
 - The ceramic resonator has 3 pins, and is connected to the XTAL, EXTAL, and ground.
 - **Mount the resonator near to the XTAL and EXTAL lines, and keep all wires to the resonator as short as possible.**
 - The outer two pins of the resonator go to XTAL and EXTAL (either way—there's no polarity).
 - **Wire the center pin of the resonator directly to Vss (ground) of the microprocessor.** Then run a wire from the processor Vss to the ground bus.
 - You must add a 10K resistor in parallel with the outer two pins of the resonator to get the oscillation frequency to stabilize.

Hardware Setup: Power Wiring

use $0.1\mu\text{F}$ capacitors ("104")
between power + ground on busses:



When wiring up the power busses, make sure to insert one $0.1\mu\text{F}$ capacitor (they're marked "104") into each power-ground bus. The diagram above illustrates this. The purpose of these capacitors is to smooth out fluctuations in the power supply, making sure the HC11 operates properly.

Software Setup

The procedure for loading a boot program into the 68HC11 is the following.

1. Turn on the HC11 with MODA and MODB held low. Your circuit (if wired properly) does this. The chip automatically powers on in serial boot mode.
2. Open a serial connection to the HC11 with parameters 1200 baud, no parity, 8 data bits, 1 stop bit (1200-N-8-1).
3. Write a 255 to the serial port. When received by the HC11, this is used to synchronize the baud rate.
4. Write 512 bytes of data to the serial port. These data are loaded into the HC11's internal RAM, and then execution automatically jumps to address 0.

Java routines are provided for performing the steps above. You should either (a) install a Java Development Kit (JDK) on your computer, or (b) use the JDK already installed on the computers in the OS 306 Engaging Computing Lab.

Instructions for installing the JDK or using the existing one are provided in a separate on-line document. Please see <http://www.cs.uml.edu/~fredm/courses/91.305/javasetup.shtml> for instructions.

HC11Boot.java

At end of this handout is the initial test program HC11Test.java. This program is also available from the course web site.

All this program does is load a tiny, three instruction program into the HC11:

```
    ldaa 0x10
    staa 0x1000
Loop: jmp loop
```

Here is what this HC11 program does. First, it loads the value 0x10 into register A. Then, it stores this value to address 0x1000. Address 0x1000 is a special address—it is the location of the PORTA data register. Data that are written to this address appear as ones and zero on the PORTA pins. The effect of writing 0x10 to this address is that bit 4 of PORTA gets set to a one. Note from the HC11 datasheet that its default (power-on) value is zero. Thus, when the program runs, Port A, bit 4 (which happens to be pin 4 of the HC11) will change from zero to a one. Hooray!

The third line of the program creates an infinite loop, jumping to itself endlessly. The CPU must be doing something; it never just stops. So if you want your program to terminate, you should keep it busy by having it loop in a known fashion. Otherwise it will just plow along, executing whatever code it happens to find in memory.

Serial.java

The HC11Boot.java program uses a class that is defined in the file Serial.java. This provides the Serial object that implements rudimentary communication through a serial port.

PROBLEM 4–1: BOOT YOUR 68HC11.

You don't have to write any code here, you just have to put all the pieces just described together and get it working for yourself.

Build up the HC11 circuit as described. Connect the HC11's Port A4 output (pin 4) to the dev board probe display.

Get Java running on a PC.

Compile the HC11Test.java and Serial.java files.

Plug the UML305DEV board into the serial port of the PC. Note whether you are using COM1 or COM2.

Power on your UML305DEV board, thereby powering up the HC11 in the serial bootstrap mode.

Run HC11Test.java (telling it whether to use COM1 or COM2). It will then communicate with the HC11 and install the little test program into the microprocessor.

Witness the Port A4 output (pin 4 of the HC11) go from zero to one, as evidenced by the probe indicator going from green to red.

Congratulations, your 68HC11 is alive!

If you got the PA4 line to turn red, you're ready to proceed. If not, you'll need to debug.

First make sure you don't have any wiring errors.

If this doesn't result in happiness, try to find either or both of (a) a known working HC11 setup, so you can test your development computer configuration, and (b) a known working computer setup, so you can test your board better.

If that fails, come to office hours/lab!

PROBLEM 4–2: BEEPING.

The following small HC11 program, `beep.s`, creates an oscillation on the Port A4 pin:

```
;;; beep.s
;;; toggles Port A4 with delay loops between each toggle,
;;; creating an oscillation in the audible range.

        ldx #0x1000          ; point at register base
loop:   bset 0,x,#0x10       ; set bit 4 in PORTA register
declp1: deca                ; decrement reg A
        bne declp1          ; and loop till it hits zero.
        bclr 0,x,#0x10      ; clear bit 4 of PORTA
declp2: deca                ; decrement reg A
        bne declp2          ; till it hits zero.
        bra loop            ; now recycle at the beginning.
```

The program works by setting and clearing bit 4 in the Port A register, making use of the bit set and bit clear instructions (`bset` and `bclr`). In between each set and clear, there is a delay that results from decrementing the A register till it hits zero.

Using the `as6811` assembler (available from the Resources: Software area of the course web site), assemble the program. Type at the prompt:

```
as6811 -l beep.s
```

so that the assembler produces a listing file. From the listing file (named `beep.lst`), look at the resultant object code. For instance, the first code line of the file assembles to:

```
0000 CE 10 00          5          ldx #0x1000          ; point at register base
```

The first column of numbers, “0000,” is the location that this code assembled to. The second set of numbers “CE 10 00,” is the object code for this instruction. The next number, “5,” is the line number of this statement in the source file.

Examine the rest of the file so that you can find the object code.

Now, create your own copy of the `HC11Test.java` file. Rename it to `BeepTest.java` or similar.

Open the file in your favorite editor, and replace the Java class name `HC11Test` with your new filename (e.g., “`BeepTest`”). You should also replace the class name in the print statement in the setup method, though of course this doesn’t affect functionality.

Now, replace the object code loaded into the `buf` array with the object code from your assembled `beep.lst` file. The first few lines should now look like:

```
buf[i++] = (byte)0xce;
buf[i++] = (byte)0x10;
buf[i++] = (byte)0x00;
```

Continue in this fashion, copying the whole of the HC11 beep program into the Java source file.

91.305 Assignment 4: Introduction to the 68HC11

Now, compile the Java file, turn on your HC11 board, and run the Java program to download the HC11 code.

When the Java program has finished, the code should be running on your HC11. If you plug the output from the Port A 4 line (pin 4 of the HC11) into the 305DEV probe line, you should see both the red and green LEDs on at the same time (actually, they're flashing back and forth, but too fast for you to see).

Now, plug the PA4 line into the piezo input. You should hear a tone!

Problem 4–2a. Your HC11 has an 8 MHz oscillator. This is divided by 4 to result in a 2 MHz instruction clock (the E clock). In the 68HC11 Reference Manual there is a mapping from instructions to cycles; for example, the LDX #0x1000 instruction takes 3 cycles, or 1.5 microseconds, to execute.

What is the frequency of the tone that the program generates? **Your answer should be a frequency in cycles per second (Hz), along with a justification for how you got it.**

Problem 4–2b. Re-write the program to generate precisely 1000 Hz. Assemble it, copy the assembled object code into a new instance of the loader program, and run it to make sure it works. Hint: the NOP instruction does nothing but consume one byte of code space and 2 instruction cycles. **Turn in a printout of your working .lst file, plus a copy of the .java file you used to download it to the HC11.**

Problem 4–2c. Which tone is louder—the original tone, or the 1000 Hz tone? Note: this is somewhat subjective, but take a stab at it. **Turn in the answer by frequency, e.g., “xxxx Hz seems louder.”**

HC11Boot.java

```

// HC11Boot.java: bootstrap program for loading code into HC11
// Fred Martin / UML CS / last modified Mon Sep 29 08:35:40 2003
//
// put the bytes you want installed into HC11 internal RAM
// into "buf" object beginning at buf[0].

class HC11Boot {
    private static Serial s;

    public static void main(String[] argv) {
        setup(argv); // select serial port and init Serial object
        System.out.println("Using serial port " + s.getPortName());

        // make 512 byte buffer and fill with zeroes
        byte[] buf= new byte[512];
        int i;
        for (i=0; i<512; i++) buf[i] = 0;

        // this 3-instruction program will turn on bit 4 of PORTA (PA4)
        // and then loop endlessly
        i=0;
        buf[i++] = (byte)0x86; // RAM loc 0 -- LDAA with 0x10
        buf[i++] = (byte)0x10; //      1      0x10 (bit 4 on)
        buf[i++] = (byte)0xb7; //      2 -- STAA extended to 0x1000
        buf[i++] = (byte)0x10; //      3      0x10
        buf[i++] = (byte)0x00; //      4      0x00 (the PORTA reg)
        buf[i++] = (byte)0x7e; //      5 -- JMP extended to 0x0005
        buf[i++] = (byte)0x00; //      6      0x00
        buf[i++] = (byte)0x05; //      7      0x05 (loop the JMP!)

        System.out.println("Writing to serial line...");
        try {
            s.put(0xff); // send baud rate detect byte
            s.put(buf); // dump 512 bytes to serial port
        } catch (Exception e) {
            e.printStackTrace();
        }
        s.flush(); // make sure all chars get out before exiting
        System.out.println("done.");
    }

    private static void setup(String[] args) {
        if (args.length == 0) {
            System.out.println("Usage: HC11Boot serial-port");
            s = new Serial();
            String[] ports=s.portsAvailable();
            int nPorts= ports.length;
            System.out.println("Available ports are:");
            for (int i=0; i<nPorts; i++) {
                System.out.println(" " + ports[i] + " ");
            }
            System.out.println();
            System.exit(0);
        }
        s = new Serial(args[0]);
    }
}

```