ME 3200 Mechatronics Laboratory
Lab Exercise 5: Introduction to Handy Board

1. Introduction

The purpose of this lab is to give you experience with microcontrollers. Their small size, relatively inexpensive price, and ever increasing speed makes them ideal for control applications where a large computer is not necessary or the space is not available. They are now used in such areas as control of automobile engines, microwave ovens, robotics, and many other applications. The microcontroller we will use is the Handy Board. The information presented in this lab only introduces the Handy Board. More detailed information can be found at the Handy Board web site www.handyboard.com.

The Handy Board is a small experimental board developed by Fred Martin of the MIT Media Laboratories. The main features of the Handy Board are a Motorola 68HC11 microprocessor chip, four DC motor outputs, seven 8-bit analog inputs, nine digital inputs, and a 16x2 LCD screen, as well as other features. It is easily programmable using Interactive C (a C programming language) developed by Randy Sargent of MIT. A diagram of the Handy Board with labeled features is shown below in Figure 1.

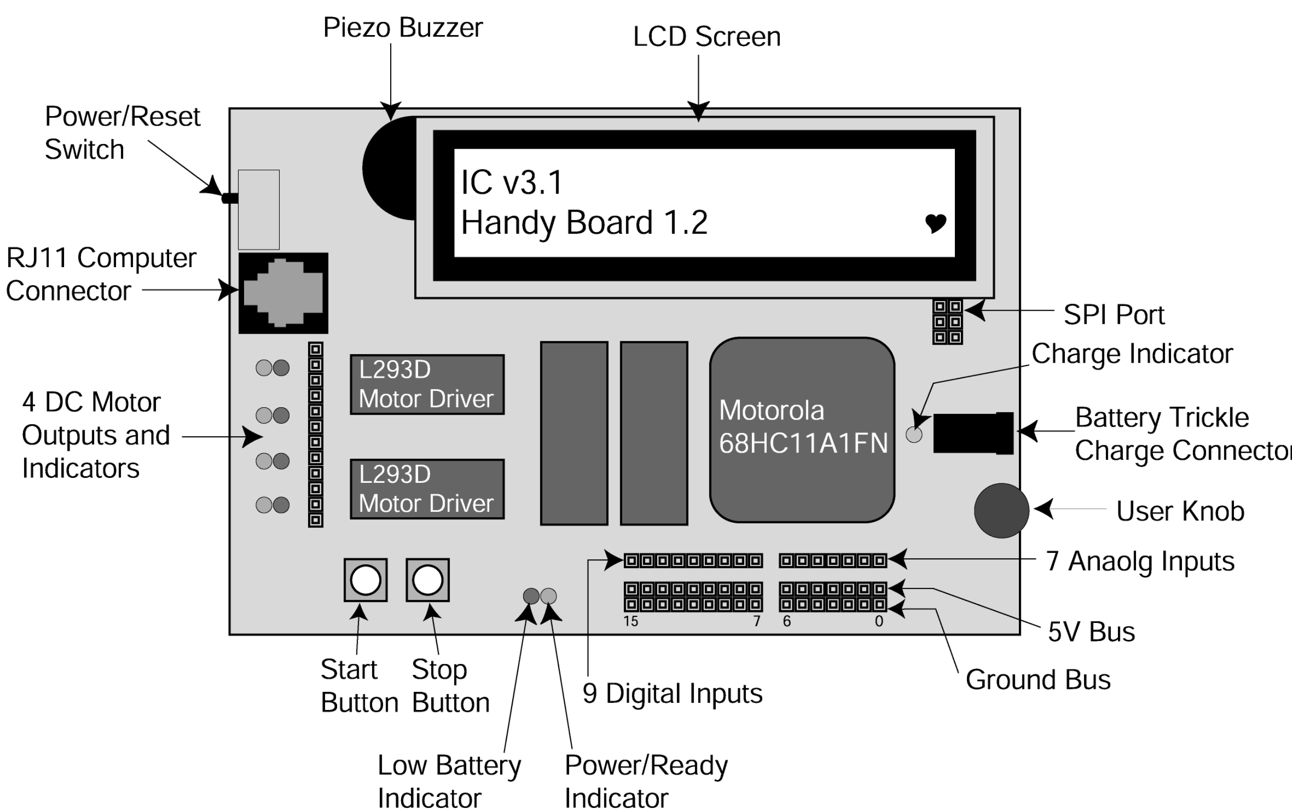

Figure 1. Handy Board labeled features.

2. Starting the Handy Board

Before starting the Handy Board, make sure the following are done:

1. The Handy Board is connected to the computer.
    a) One end of the RJ11 modular cable (looks like a telephone cable) should be plugged into the computer connector port on the Handy Board (see Figure 1).
    b) The other end of the RJ11 modular cable should be plugged into the Serial Interface & Battery Charger Board, a separate small interface printed circuit board located at each computer workstation.
    c) From the Serial Interface Board, a serial cable should be connected to the serial port of the computer. If a connection cannot be made between the Handy Board and the computer, a loose connection here is a likely cause.
2. Insert the plug-in transformer into a wall outlet. Insert the other end into the receptacle on the Handy Board battery trickle-charge connector (see Figure 1). This step will keep the batteries on the Handy Board charged while you are working on it, but is not necessary if the batteries are fully charged.
3. Start **Interactive C** by double clicking the icon on the desktop or through the *Start* button

Sometimes it is necessary to reload the operating system (pcode). To reload the pcode, follow the loading instructions

3. Loading the Operating System (PCODE)

1. If the board is not configured error window pops up then Press *YES* to configure the board settings in the window and make sure that *Port :COM1* is selected and press *Download Pcode* it should point you to *C: > IC > libs* directory, then select *Handy_board_1.2.icd* file to download (otherwise browse through and find it)
2. Follow the instructions on the computer screen to place thehandy board in **download mode**. It is critical that the Handy Board is in this mode to reload the operating system code.
3. Once the download process is complete, restart the Handy Board. If everything is working well, the LCD screen should appear as in Figure 2 with the heart beating. If the download process is unsuccessful, then ask your TA for help.

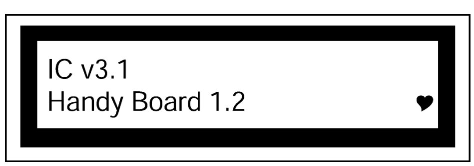

Figure 2. Healthy Handy Board LCD screen at startup.

4. Running the Handy Board directly from the PC (Interactive Mode)

The Handy board is now ready for interactive mode. Otherwise**, to start the Handy Board in interactive mode, hold down the START button as you switch on the Handy Board**

You are now in the interactive mode for the Handy Board. The Handy Board is now ready to use. If the board does not synchronize with the computer, try again. If this does not work, get help from your TA.

Commands to the Handy Board can be sent directly to it from the host computer in Interactive Mode. For example, at the prompt, try typing:

```
printf("Hello World!\n"); <enter>
```

The screen on the Handy Board should read **Hello World!** The character **\n** at the end of the string signifies *end-of-line*. When an *end-of-line* character is printed, the LCD screen will be cleared when a subsequent character is printed.

Now type `beep();<enter>`. You should hear a short tone coming from the piezo buzzer (see Figure 1). Try typing `tone(800.,5.);<enter>`. You should hear a tone at 800 Hz for 5 seconds. Other commands are included at the end of this lab handout. Explore them later.

5. Downloading Programs to the Handy Board

We are now ready to write a program to download to the Handy Board. Open a new file and save the file as sample.c in the **c:\handy** directory. Type the following in your program:

```
void main(){
  printf("You now know how to download\n");
  }
```

At the prompt type **load sample.c** <enter>. When the **load** command is called, the C code written is compiled and loaded into the memory of the Handy Board. If the download was successful, you should not see any error messages. Now to start the program, reset the Handy Board by turning it off and then back on. The screen should read, **"You now know how to download"**. Each time the board is turned off and then back on, the program will be executed until it is replaced or the Handy Board is started in interactive mode or download mode.

Remark: The program can be named any name allowable by Dos (up to 8 characters with certain characters being restricted) but must have a **.c** DOS file extension. Multiple programs can be loaded (up to 32K) onto the Handy Board, but only one can have the function **main()**. It will be this program that is run on start up of the Handy Board. The other programs are callable either from the program containing the function **main()**, other functions, or from the interactive mode. This feature allows the Handy Board to perform multitasking, or run several subroutines simultaneously.

6. Handy Board Functions

There are many commands that can be used to control the Handy Board. You have seen three of them, **printf**, **beep**, and **tone**. These commands can be given directly in the interactive mode (as in step 3) or used as function calls in a C program (as in step 4) and downloaded to the Handy Board. We will be mainly writing programs and downloading them to the Handy Board.

A brief description of the more useful functions and how to use them are given below. For a more complete listing of the functions, see the **Interactive C Manual** available on the Handy Board web site.

6.1. `printf(format-string, [arg-1], . . . , [arg-N])`

Printing information to the screen can be a useful debugging tool. Suppose the value of parameter **x** is needed for testing the code. The **printf** command would be used as follows:

```
printf("Value is %d\n",x);.
```

The special form **%d** is used to format the printing of a floating-point number in decimal form. A listing of other formats is given below in Table 1.

Table 1. Number printing formats.

Format Command	Data Type	Description
%d	int	decimal number
%x	int	Hexadecimal number
%f	float	floating point number
%s	char	character array (string)

6.2. `motor(int m, int p)`

Turns on motor *port m* at power *level p*. There are four motor ports on the Handy Board, labeled Motor-0, Motor-1, Motor-2, Motor-3 (see Figures 1 and 3 for location). The power level ranges from 100 for full on forward to –100 for full power in reverse. Although the range of power levels goes from –100 to 100 in increments of 1, there are actually only 15 motor speeds (See Table 2 below).

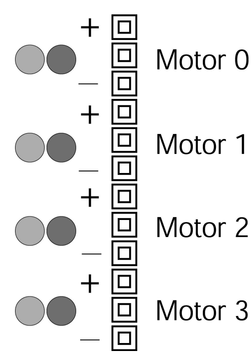

Figure 3. Motor ports designation. Note that the middle slot for each motor is used just for spacing and does not carry any signal.

The motor driver chips (L293D on the Handy Boards) are capable of putting out up to 9.6 volts and 600 mA. A higher current demand is required for loaded DC motors, and an additional motor driver circuit must be used. Refer to the class web site for additional information on external motor driver circuits.

The motor drivers use pulse width modulation (PWM) to control the voltage output. This means that full voltage is switched on and off at high frequency for varying lengths of time. The effective voltage seen by a motor (which has a slow response time compared to the frequency of the PWM) is proportional to the average time spent in the high state (full voltage). Note that if you measure the output with an oscilloscope, you will see a high frequency signal.

Table 2. Motor output levels.

Power Level *	Forward Motor Speed
0-10	off
11-24	1
25-38	2
39-52	3
53-66	4
67-80	5
81-94	6
95-100	7

**A negative power level corresponds to the appropriate Motor Speed, but in reverse.*

6.3. `alloff() or ao()`

Turns off all motors.

6.4. `off(int m)`

Turns off motor *m*.

6.5. `digital(int p)`

An active low digital input bus that returns the value of the digital sensor in digital sensor *port (7-15)*. If zero volts are applied to the port, it returns true (1). If over 2.5 volts are applied, the value returned is false (0).

6.6. `analog(int p)`

A function that reads the voltage level on the specified *port (0-6)* on the analog input bus, which is connected to an 8-bit analog-to-digital converter. The value returned is between 0 and 255. Zero applied voltage returns a value of 0, while a level of 5 volts returns a value 255.

Refer to Figure 1 for the location of the analog and digital ports. Note that these ports are the top of the three strips in that area, and are numbered from right to left. The bottom strip is ground, and the middle strip is a +5 supply voltage that can be used to power sensors. The top strip is for the sensor signal (this is the value that will be read by the **analog** and **digital** functions).

Another thing to note about the analog input ports is the internal pull-up resistor. There is a 10-kΩ resistor internal to the analog input bus. A potential of 5 volts is applied to one end of the resistor, while the other end of the resistor is connected to the analog input ports. When one lead of an analog sensor is inserted into analog input and the other into ground, a voltage divider is created. The voltage across the internal pull-up resistor is what gives the sensory reading.

6.7. `knob()`

A function that returns the position of the user knob (see Figure 1) as a value between 0 and 255. The knob is a potentiometer whose wiper is connected to an 8-bit analog-to-digital converter.

6.8. `start_button()`

Returns the value of the button labeled "Start" (see Figure 1). Returns 1 if pressed and 0 if released.

6.9. `stop_button()`

Returns the value of the button labeled "Stop" (see Figure 1). Returns 1 if pressed and 0 if released.

Example: (Note that the “!” means logical NOT.)

```
/* Wait until start button is pressed */
while(!start_button()){}
```

7. Other Functions and Additional Information

There are many other functions that might be useful in programming the Handy Board. A complete listing is given on the following page. For a complete description of these functions, see **The Interactive C Manual for the Handy Board** on the Handy Board web site. This reference contains a basic C programming tutorial for those who are not familiar with C programming. Further information about The Handy Board, ranging from schematics, modifications, new functions, and much more is available on the World Wide Web at www.handyboard.com.

==

List of Handy Board Functions

LCD Screen Printing

```
printf(format-string, [arg-1], . . . , [arg-N])
```

DC Motors

```
void fd(int m)
void bk(int m)
void off(int m)
void alloff()
void ao()
void motor(int m, int p)
```

Servo Motors

```
void servo_a7_init(n)
int servo_a7_pulse
void servo_a5_inti(n)
int servo_a5_pulse
```

Sensor Input

```
int digital(int p)
int analog(int p)
```

User Buttons and Knob

```
int stop_button()
int start_button()
void stop_press()
void start_press()
int knob()
```

Infrared Subsystem

```
int sony_init(1)
int sony_init(0)
```

```
int ir_data(int dummy)
```

Time Commands

```
void_reset_system_time()
long mseconds()
float seconds()
float sleep(float sec)
void msleep()
```

Tone Functions

```
void beep()
void tone(float frequency, float length)
void set_beeper_pitch(float frequency)
void beeper_on()
void beeper_off()
```

Multi-Tasking

```
start_process(function-call(. . .), [TICKS],[STACK-SIZE])
int kill_process(int pid)
```

Process Management

```
void hog_processor()
void defer()
```

Arithmetic

```
float sin(float angle)
float cos(float angle)
float tan(float angle)
float atan(float angle)
float sqrt(float num)
float log10(float num)
float log(float num)
float exp10(float num)
float exp(float num)
(float) a^(float) b
```

Memory Access

```
int peek(int loc)
int peekword(int loc)
void poke(int loc, int byte)
void pokeword(int loc, int word)
void bit_set(int loc, int mask)
void bit_clear(int loc, int mask)
```

8. Laboratory Exercises

Read through the microcontroller introduction handout in lab and learn how to use the Handy Board. You must demonstrate your code to the TA and answer any questions.

1. Generate a tone that is dependent on knob position. Save this program in a file whose name is of your choosing. Remember to give it the proper DOS file extension **.c**.
   a. Measure knob position and display the output so you can determine the range of knob output.

```
int x;            /*ALL variables used must be declared */
void main()       /* Defining the main program */
{  /* Spacing delimiters like this can help you navigate the loops. */
    while(1)
    {
        printf("Press START\n");
        while(!start_button())  /* Wait for START to be pressed */
        { }
        while(stop_button()==0) /* Notice ! and ==0 are equivalent. */
        {
            x=knob(); /* define x as the value read on the knob. */
            printf("knob is at %d\n", x);
        } /* Closes the while loop. */
    }
} /* Closes main.  This program will never reach this point. */
```

   b. Modify this code to include a conditional statement for two tones using the if-else conditional statement as in the following code. Make the greater knob value sound off with a higher frequency tone, and the lesser knob value sound off with a lower frequency tone. Remember that the arguments for **tone** are the frequency and the duration, both floats. All floats must have a decimal point placed within.

```
int x;
void main()
{
   while(1)
   {
      printf("Press START\n");
      while(!start_button())
      { }
      while(stop_button()==0)
      {
         x=knob();
         printf("knob is at %d\n", x);
         if(x<128)
         {
            tone(250.,.1);
         }
         else
         {
            tone(550.,.1);
         }
      }
   }
}
```

   c. It is possible to convert an integer variable to a floating-point variable. This is accomplished by simply placing the word **(float)** in front of the variable as follows:

```
int x;
float f;
void main()
{
   while(1)
   {
      printf("Press START\n");
      while(!start_button())
      { }
      while(stop_button()==0)
      {
         x=knob();
         printf("knob is at %d\n", x);
         f=(float)x+150.;
         tone(f,f/1000.);
      }
   }
}
```

d. Modify to only run for 20 seconds by modifying the while **(stop_button()==0)** statement to include a Boolean AND statement:

```
reset_system_time();
while (stop_button()==0 && (int)mseconds()<20000)
```

Recall that **mseconds()** returns a long format number, which is a 32-bit int format number. A Boolean logical operator can only compare numbers of the same format. It is for this reason that the **mseconds()** result requires conversion.

2. Measure the time to read a data value (**knob**), perform a calculation, output the value to the tone generator, and then display the delay time on the screen. Resetting the millisecond counter at the end of the loop does this.

```
float x,y;
void main()
{
   reset_system_time();
   y=(float)knob();
   x=y*250./3.5;
   tone(x,0.01);
   printf("Delay time is %d msec\n",mseconds());
}
```

Take note of how much time it took to run these commands. Modify the above code to measure the time it takes to print to the screen by repeating the **printf** statement at the end so it says it twice. How much time is lost? What is the percentage of time it took to print?

3. Write your own program to control the tone from the knob. This time, the 0 position on the knob will give a 600 Hz tone, while the 255 position on the knob will give a 200 Hz tone. If the knob value is less than 156 have the tone last 50 msec. If it is greater than or equal to 156, have the tone last 100 msec. Have your program shut off after 30 seconds.