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

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. Microcontrollers are now used to control automobile systems (engines, transmissions, braking, collision avoidance, traction control, etc), microwave ovens, robotic systems, and many other systems. The microcontroller used in this laboratory and in your robotics project is the Handy Board. The following introduction and laboratory procedure are presented to help students become acquainted with the basic functions and operation of the Handy Board.

The Handy Board

The Handy Board is a small single board computer 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, a 16x2 character LCD screen, and an internal 9.6 Volt NiCad battery pack. The Handy Board is programmed using Interactive C, a C programming language, developed by Randy Sargent of MIT. A diagram of the Handy Board depicting its main features is shown below in Figure 1. The following sections describe some of the basic steps needed to prepare the Handy Board for programming and the more useful pre-programmed functions of the Handy Board. More detailed information about the capabilities of the Handy Board and its programming interface can be found at the Handy Board web site (http://handyboard.com) and on the mechatonics class webpage under project handouts. The *Handy Board Technical Reference* and the *Interactive C for the Handy Board Manual* in the Docs section are of particular value for understanding the Handy Board and using it robotic control applications.

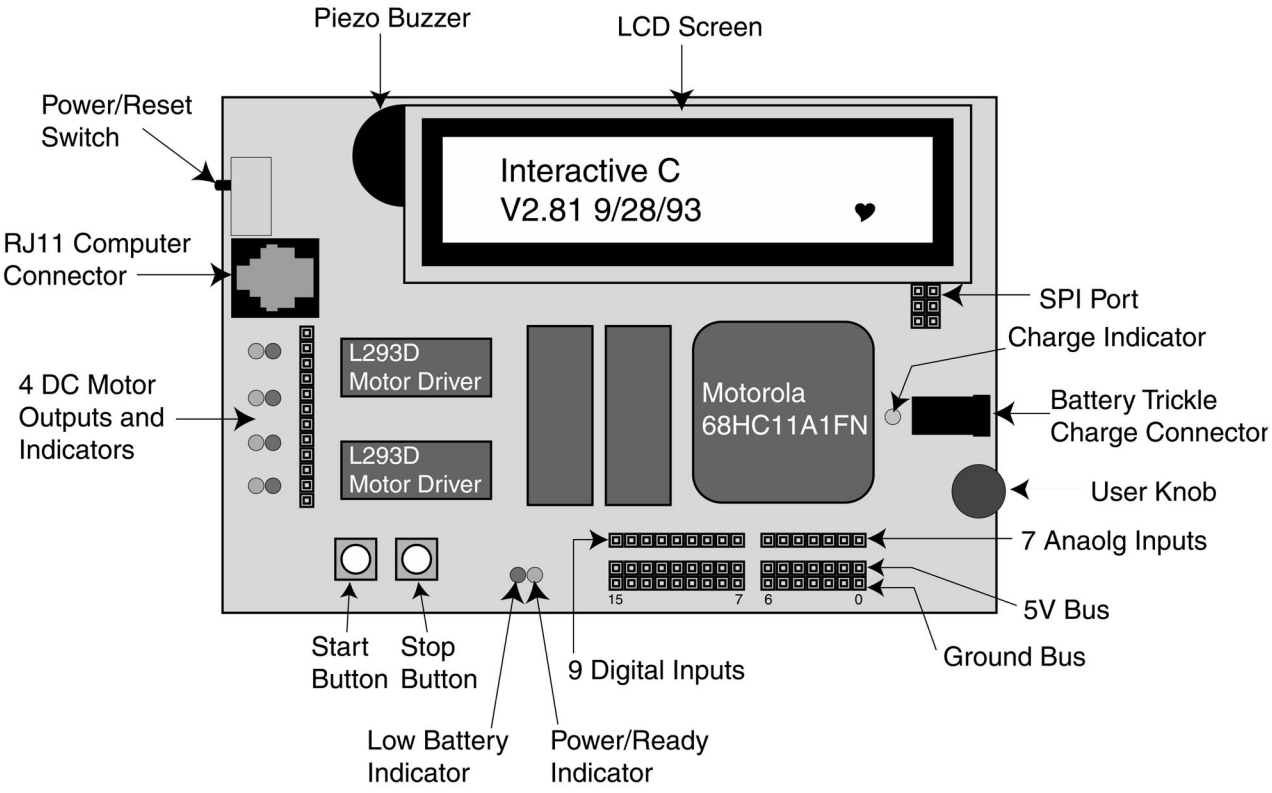

Figure 1: A labeled diagram of the Handy Board's major features.

Preparing the Handy Board for Use

Before you begin using the Handy Board, make sure that you have the following:

- A Handy Board
- A RJ11 modular cable (looks like a telephone cable)
- A Serial Interface & Battery Charger Board
- A 12 Volt, 500mA DC transformer.
- Interactive C installed on a host computer
- A Serial Cable
- A Host Computer

The procedure to prepare the Handy Board for computer interfacing and programming is as follows:

1. Begin charging the Handy Board by inserting one end of the DC transformer into an electrical outlet and the other into the trickle charger connector on the Serial Interface & Battery Charger Board. This step will maintain the charge on the Handy Board's battery while you are working on it, but is not necessary if the batteries are fully charged.

2. Connect the Handy Board to the host computer by

   a. Connecting one end of the RJ11 modular cable into the computer connector port on the Handy Board (see Figure 1) and the other end into the Serial Interface & Battery Charger Board.

   b. Using the Serial Cable, connect the Serial Interface Board to the serial port of the computer.

   Note: If a connection cannot be made between the Handy Board and the computer, a loose serial connection or faulty RJ11 cable are likely causes.

3. Turn on the Handy Board using the Power/Reset Switch (see Figure 1).

4. Start Interactive C on the host computer by double clicking the icon on the desktop or through the Start menu.

If the Handy Board has not been used in some time, the internal memory has been corrupted, or if the internal battery doesn't have enough charge to retain the system memory it may be necessary to reload the Handy Board's operating system (called the pcode). An error in the pcode is indicated when the LCD screen displays a line of boxes immediately after the Handy Board is turned on and Interactive C can't communicate with the Handy Board. If the Handy Board's LCD display resembles Figure 2 below you can proceed to the section entitled "Running the Handy Board from the PC" below; otherwise, load the pcode according to the following instructions.

Loading the Operating System (PCODE)

To reload the operating system code, do the following:

1. Connect the Handy Board to the host computer as directed in the previous section.

2. Upon starting Interactive C an error window appears and asks if you would like to configure the board settings. Select the *YES* button in the error window.

3. In the next window, verify that *Port: COM1* is selected and press *Download Pcode*.

4. If you are working on a computer in the Mechatronics Lab, the next window points you to *C:\IC\libs* and waits for you to select the *Handy_board_1.2.icd* file and press *Open*.

   Note: If you are using a computer that is not in the Mechatronics Lab you may need to browse the local drives for this file. If you do not have it on your computer, ask for a lab TA to make a copy of the file for you.

5. Follow the instructions in the next pop-up window to place the Handy Board into Download Mode. It is critical that the Handy Board is in this mode to reload the operating system.

6. Once the download process is complete you will be prompted to restart the Handy Board by turning it off and then on again. Once you have reset the Handy Board select *OK* in the pop-up window to reload the remainder of the Handy Board code.

   Note: After the Handy Board is restarted the LCD screen should appear as in Figure 2 below with a beating heart icon in the bottom right corner. This screen tells you that the pcode loaded successfully and that the Handy Board is ready for programming.

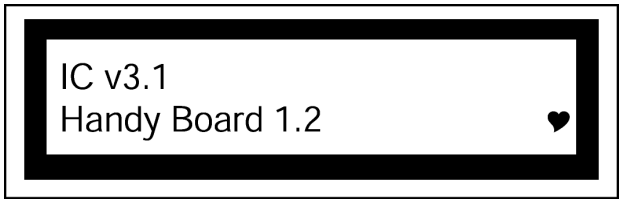

Figure 2: Heathy Handy Board LCD start-up screen.

Upon completion the preceeding steps, the Handy Board is ready for programming in interactive mode described below.

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

If the pcode is correctly loaded (i.e. the Handy Board LCD looks like Figure 2 when the Handy Board is first turned on), place the Handy Board in interactive mode by turning off the Handy Board and pressing the START button while switching the Handy Board back on. Interactive mode enables the user to send commands directly to the Handy Board from the command line of Interactive C. For example, typing "`printf("Hello World!\n");`" at the command prompt and pressing enter will display the text "Hello World" on the LCD screen. 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. Executing Handy Board commands in interactive mode is an effective way to explore unfamiliar functions or commands. This mode is also useful tool used to troubleshoot programs by executing commands one at a time.

Downloading Programs to the Handy Board

With the Handy Board in interactive mode, you are now ready to write a program and download it to the Handy Board. Handy Board program files are created using a text editor such as DOS edit or Windows Notepad and must be saved with a ".c" file extension (i.e. *sample.c*). For example, suppose that you created a program called *download.c* that contained the following text:

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

At the Interactive C command prompt type *load download.c* and press enter. The **load** command, compiles the C code of *download.c* and loads the code into the memory of the Handy Board. If the download is successful no error

messages are displayed in the Interactive C window. To start the program, reset the Handy Board by turning it off and then back on: the LCD screen should then 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 reset in interactive mode or download mode.

Programs 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 **main()** function, other functions, or while in interactive mode. This feature allows the Handy Board to perform multitasking, run several subroutines simultaneously, and allows code to be developed in modules and then loaded separately via a script file.

Handy Board Functions

The behavior of the Handy Board can be modified using a wide variety of commands. These commands can be given directly in the interactive mode or used as function calls in a C program(s) that are downloaded to the Handy Board. The most useful method, however, is creating programs that are loaded directly onto the Handy Board for execution without interaction with the PC host. 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 for the Handy Board Manual* available on the Handy Board web site.

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

Printing information to the screen can be a useful debugging tool, but it also slows program execution significantly. 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 format commands.

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

`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 forward to –100 for full 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).

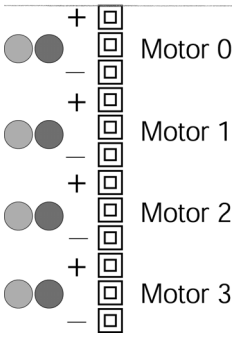

Figure 3: Motor port designation. Note that the middle slot for each motor is used just for spacing and does not carry any signal.

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
94-100	7

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

The motor driver chips (L293D on the Handy Boards) are capable of supplying up to 9.6 volts and 600 mA. A higher current demand is required for loaded DC motors, and an external power source and motor driver circuit must be used. 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.

`alloff() or ao()`

The **alloff()** command turns off all motors simultaneously.

`off(int m)`

This command turns off motor *m*.

`digital(int p)`

An active low digital input bus that returns the value of a digital sensor connected to one of the nine digital input ports (ports 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).

`analog(int p)`

The `analog(int p)` function reads the voltage level on the specified analog input port (ports 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 volt 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). All the analog input ports are connected to the five-volt bus through an internal 47-kΩ pull-up resistor. This internal resistor can change the behavior of some sensors. Specifically, a voltage divider is created when one lead of an analog sensor is inserted into an analog input and the other into ground (refer to Lab 3 for a description of a voltage divider). The voltage across the internal pull-up resistor is the voltage read by the eight-bit ADC of the Handy Board.

knob()

This function 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. This potentiometer divides the voltage between the five-volt bus and ground.

start_button()

This function returns the value of the button labeled "Start" (see Figure 1): 1 if pressed and 0 if released.

stop_button()

This function returns the value of the button labeled "Stop" (see Figure 1): 1 if pressed and 0 if released.

Other Functions and Additional Information

There are many other functions that might be useful in programming the Handy Board. A complete listing of the Handy Board functions is given below; for a complete description of these functions, see the *Interactive C Manual for the Handy Board* on the Handy Board web site or the class website under project handouts. This manual also contains a fundamental C programming tutorial for those who are not familiar with C programming and additional information about The Handy Board, ranging from schematics, modifications, new functions, and much more.

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)
```

Laboratory Exercises

Prepare Your Handy Board

1. Referring to the section *Starting the Handy Board* above, locate all the necessary equipment for your lab station and prepare the Handy Board for use.

2. Start the Handy Board in Download Mode by turning it on while the stop button is pressed until the two LED's to the right of the stop button turn off and practice loading the pcode onto the Handy Board if you haven't done so already.

Programming Assignments

3. Create a program that generates a tone that is dependent on knob position by modifying the code listed in step (a) below. Save this program onto your lab disk; remember to give it the proper .c file extension.

   a. The following code measures the knob position and displays the position as an integer between 0 and 255 on the LCD screen.

```
int x;                                  /*Declare all variables*/
void main()                             /*Define the main program*/
{
while(1){
     printf("Press START\n");
     while(!start_button()){            /*Wait for START to be pressed*/
     }
        while(stop_button()==0){
        x=knob();                       /*define x as the knob position*/
        printf("knob is at %d\n", x);
        }                               /*Close while loop. */
        }
}                                       /*Closes main*/
```

4. Modify the code of step three to include a conditional statement that makes the Handy Board create a tone with a higher frequency tone when the knob is above the halfway point, and a lower frequency tone when below the halfway point (use the tone command)—insert the following code just below the line where x is defined.

```
if(x<128){
     tone(200.,50.);     /*generates a 100Hz tone for 50msec*/
}
else{
     tone(300.,50.);     /*generates a 200Hz tone for 50msec*/
}
```

   Save the new program, load it onto the Handy Board, and demonstrate its behavior to your TA.

5. Modify the code you created in step four to create a tone whose frequency and duration depend linearly on the knob position according to the following equations:

$$f = x + 150$$
$$d = \frac{f}{1000} \tag{1}$$

   where *f* is the frequency of the tone, *x* is the knob position, and *d* is the tone duration. Remember that *f* and *d* must be floating point variables. It may be necessary to convert one of the variables to a floating point variable by including the **(float)** command just before the variable you wish to convert.

6. Modify the code in step five so that it only runs 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)
```

The **mseconds()** function returns a long format number, which is a 32-bit integer. Since a Boolean logical operator can only compare numbers of the same format, the **mseconds()** result requires the **(int)** conversion. Run the code on your Handy Board and demonstrate its behavior to your TA before you continue.

7. Create a new program that will help you measure how much time it takes to read the knob value, perform a calculation, output the value to the Handy Board speaker using the tone command, and then display the time delay on the LCD screen. Use the following code for this activity.

```
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());
}
```

Don't forget to reset the system time at the beginning of your program. Save this program for your records.

   a. Make note of how much time it took to run these commands.

   Modify the code above to measure the time it takes to print text to the screen by repeating the **printf** statement at the end of your program. How much time is lost when you use the **printf** command?

   What percentage of the total time did it take to run the **printf** command?

8. Write a program of your own design that generates a tone on the system speaker that meets the specifications listed below. Save the program for your own records and demonstrate that it functions properly to your TA.

   a. The program will start when the start button is depressed.

   b. The tone frequency will vary linearly with knob position from 600 Hz at the lowest position to 200 Hz at the highest position.

   c. The tone will last fifty milliseconds if the knob value is less than 156 and 100 milliseconds if greater than 156.

   d. The LCD screen will display the knob position and the number of seconds since the program began.

   e. The program will run for only thirty seconds and then wait for the start button to be pressed again.

9. Clean up your station and answer the questions on the next page.

Questions

Refer to the lab handout, the *Interactive C Manual for the Handy Board*, and the *Handy Board Technical Reference* on the Handy Board website to answer the following questions.

1. In the table below, list the three different ways to charge the Handy Board's internal battery pack, the time it takes for each method to fully charge the battery, and the pros and cons for each.

Charging Method	Time to Full Charge	Pros	Cons
1.			
2.			
3.			

2. How fast is the Motorola 68HC11 microprocessor chip?

   How will the speed of the processor effect the control scheme for your robot project?

   Describe at least two methods of increasing program speed and efficiency.

3. Using the list of functions in this handout and the descriptions in the *Interactive C Manual for the Handy Board*, list five or more Handy Board functions that you think will be helpful in programming your robot and describe what they would do for your robot (you may want to discuss this as a project group).