Section Overview
These experiments are intended to show some of the application possibilities of the Mechatronics board. The application examples are broken into groups based on which functionalities of the mx_ctlr.0 are used. The experiments progress from simple to more complex functionalities. All experiments will contain the required code, explanations, and hardware setup for each. The code for each experiment can be downloaded from mx_ctlr.0 website (Appendix B). Help on commands can also be found in the mech_lib documentation (Appendix B).

Soldering the Motors and Switches
Included in the kit is a length of CAT 5 cable that will be used to solder the motors and switches. Cut off a section of the blue insulation, being careful not to cut the wires inside (after 10" of the insulation has been cut, the rest of the wires can be removed by pulling them out of the insulation). Cut a desired length of wire for the motors and strip about ¼" of the insulation off both ends, leaving a bare section of copper wire. Attach the wires as shown in Figure 11; this will help prevent the leads from breaking off of the motors. Solder the leads to the terminals on the motor as shown in Figure 12.

Figure 11: Stripping wires.
The switches can be hooked up as normally open or normally closed switches. This means that they can be connected so that when the switch is not activated, the two wires are shorted together (normally closed) or not shorted (normally open). To make a normally open switch, connect one wire to the center terminal and the other wire to the terminal that is not connected to the center when the switch is not pressed. To make a normally closed switch, attach the center wire and the terminal that is connected to the center when the switch is not pressed.

Once the motor and switches have been wired, they can be inserted into the board using the terminal blocks as shown in Figure 14. Screw the tops down to make a snug connection.
Figure 14: Attaching devices to the mx_ctlr board.
Power Supply, ISP Connector, Reset Button

**Basic Programming**

For this experiment, hook up two motors to motor terminals 1 and 2. Download the program for this experiment from the mx_ctlr website (Appendix B) and refer to section 1 on how to program the board. This program turns the motors forward and then reverse; once it has completed the cycle, the reset button must be pressed to start the sequence again. A program such as this is similar to what a VCR might do: once the tape has finished, the VCR would need to rewind it to play again.

Simple sequential control like this can create many different systems and unique devices. Most assembly lines and machines break down to this type of control.

```c
#include <mechlib.h> //This library contains the functions specific to the mechatronics board

int main(){
    while(1){ //this while loop is always true, it will continue forever
        forward(1); //this function turns the motor number inside the parentheses forward
        forward(2);
        forward(3);
        forward(4);
        delay(20); //this function delays the program from continuing for 2 seconds
        reverse(1); //this function puts the numbered motor in reverse
        reverse(3);
        delay(20);
        halt(); //this function stops all actions of the program
        //reset must be pushed to start over
    }
}
```
Basic Inputs, Limit Switches, Motors

Motor Control

This section will introduce the basic input terminals and how to use them with the limit switches. Attach a motor to motor terminal 1 and 2, and switches on inputs 1 and 2. Download the program for this section onto the board. This program uses the inputs from the switches to turn the motors forward until the switch is released. Programs similar to this might be used in control system for a car, as long as you turn the steering wheel, the car continues to turn. Another option is to have a system where you press a switch once to start an event, and then press it again to stop it. Many different types of control systems used some type of feedback to double-check what is occurring. Even machines that run through a timed sequence of events will have built-in fail-safes that are always monitored to prevent injury to people.

Using limit switches is very important for many mechanical devices. As humans we have a lot of built in ‘position control’ sensors, we know where all of the parts of our body are at all times; machines don’t inherently have this ability, so it has to be designed in.

Let’s take a common computer printer for example. How does the printer know that the printer head is at the edge of the paper? In many printers there is a limit switch that the head presses when it is at the edge of the paper.
State Control Logic

The program in this section uses state logic to control the motors. Depending on what state the motors are in, activating the input switch will have different output combinations. For this section, attach the motors to terminals 1 and 2 and a switch to input 1. When the switch is depressed the motors will change state. Similar systems are used when a series of outputs are needed from a single unit, such as an assembly line where the task will vary from lifting to attaching to moving.

A state controller like this one allows for a lot of different functionalities. Many different inputs could allow for lots of different outputs depending on what was going on with the machine at the time. For example, in a washing machine, it wouldn’t make any sense to add the fabric softener during the wash cycle.

<table>
<thead>
<tr>
<th>Motor 1</th>
<th>Motor 2</th>
<th>Motor 1 – Next State</th>
<th>Motor 2 – Next State</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stop</td>
<td>Stop</td>
<td>Stop</td>
<td>Forward</td>
</tr>
<tr>
<td>Stop</td>
<td>Forward</td>
<td>Forward</td>
<td>Stop</td>
</tr>
<tr>
<td>Forward</td>
<td>Stop</td>
<td>Forward</td>
<td>Forward</td>
</tr>
<tr>
<td>Forward</td>
<td>Forward</td>
<td>Stop</td>
<td>Stop</td>
</tr>
</tbody>
</table>

```c
#include <mechlib.h>

main()
    unsigned char state=0;
    while(1)
    {
        if(poll_sensor(1)&&state==0){
            brake(1);
            forward(2);
            state=1;
            delay(3);
        }
        if(poll_sensor(1)&&state==1){
            forward(1);
            brake(2);
            state=2;
            delay(3);
        }
        if(poll_sensor(1)&&state==2){
            forward(1);
            forward(2);
            state=3;
            delay(3);
        }
        if(poll_sensor(1)&&state==3){
            brake(1);
            brake(2);
            state=0;
            delay(3);
        }
    }

}extern main
```

This program turns on the motor outputs in a binary sequence when a switch is pressed.
Digital Switches

Elevator Service

For this experiment, an LED will need to be hooked up in series with a 100Ω resistor (brown, black, brown) with two leads attached.

Note: be careful to solder the shorter lead of the LED to the resistor; the LED has polarity and will not work if soldered on backward.

Figure 15: Attaching a LED.

Once the LED has been soldered, attach the lead on the LED without the resistor attached to Vcc on the alternate input port and the lead on the resistor not connected to the LED to digital switch 1. You will also need to attach three switches (inputs 1, 2 and 3). Motors are optional for this lab since the motor LEDs will indicate which motor is on and which direction it is going.

In a building with four elevators, in order to conserve power and be more efficient, only three elevators will be on for normal use. Each switch will turn on a motor terminal that represents an elevator. When two of the
elevators are being used an indicator light will start blinking. Once all three elevators are in use, the forth elevator will start automatically and the light will turn on, indicating that all four elevators are in use.

The digital switch could be used for many different applications. It is not powerful enough to run an electromagnet or a motor, but it can easily trigger a relay that could run nearly any size device you would want—even something like a light bulb in a house or a large siren.
Pulse Width Modulation

Motor Speed
Depending on the task at hand, you will need to have different operating speeds. This program allows you to step through four different motor speeds as well as reverse, by pressing the limit switches. You will need to attach one motor and two switches. This is similar to manual gears in a car: you press one button and the motor will speed up to a certain point, press the other switch and the motor will slow down to a stop. The precision command allows you to vary the speed of the motor in forward and reverse with 255 bits of accuracy.

Pulse width modulation (PWM), is used to control many motors today. It works by rapidly turning the electricity off and on to the motor. The more often the electricity is off, the slower the motor will turn. This method is very efficient because little energy is wasted since all of the electricity goes through the motor.

Alternate Inputs, Analog to Digital Converter, Sensors

Light detecting Motor
Solder a photo resistor and a 470Ω resistor (yellow, violet, brown) in series. Connect a wire to each end and to the middle of the resistor/photo resistor pair, Figure 16. Connect the lead connected only to the resistor to Vcc on the alternate input port. Connect the lead only connected to the photo resistor to the GND on the alternate input. Connect the center lead to alternate input 1.
Autonomous devices, such as some robots, need to have the ability to detect objects around them. One possible solution is having a light detector that will find the brightest point or could detect when a light is behind an object. This program will start turning a motor once a switch has been pressed. When the switch is pressed again, usually after one rotation the motor will stop and go back to the brightest point. When the switch is pressed the third time, the motor will begin searching again.

**Simple Motor Positioning**

A Hall effect sensor can be used as a simple position sensor to measure the rotation of the motor. The Hall effect sensors supplied in the Mechatronics kit are ‘omni polar’ sensors. This means they can detect both the north and south poles of a magnetic field. To use one, simply supply it with Vcc and GND and monitor the output. Figure 17 shows a diagram of the hall effect sensor. Connect Vcc and GND to Vcc and GND on the alternate input port. Connect the Output pin to input 1 on the alternate input port. You can use the ‘poll_ir’ command to check the Hall effect sensor.

```c
#include <mechlib.h>  // Uses switch

main(){
  unsigned char max_light=0, temp_value=0, count=0, position=0;
  while(1){
    while (!poll_sensor(1))
      delay(3);  /*collects data on brightness*/
    while (!poll_sensor(1))
      precise(1,1,60);
    delay(1);
    brake(1);
    temp_value=read_adc(0);
    count++;
    /*saves the location of brightest point*/
    if(temp_value > max_light && count>3){
      max_light=temp_value;
      position=count;
    }
    delay(3);  /*returns to brightest point*/
    if (count>3){
      brake(1);
      temp_value=count-position;
      for(temp_value; temp_value--; temp_value>0){
        precise(1,0,60);
        delay(1);
        brake(1);
      }
      count=0;
    max_light=0;
    temp_value=0;
    position=0;
  }
}
```
Tape a small magnet to the side of the disc on the motor. The sensor will need to be mounted such that the flat side of the sensor faces the magnet. The sensor must be placed fairly close to the magnet on the motor to work. The motor will turn until the magnet and the sensor are lined up together. If the sensor is then moved away, the motor will turn until the sensor is again located.

**Serial Communication**

**Computer – Board: Motor Control**

Nearly every electronic device today is controlled by some sort of a computer. The built in serial communication on the mx_ctlr.0 allows you to connect the computer to the board using any serial connection program. Download the program and connect the board to the computer using an extension male to male DB9 cable to either Com Port 1 or 2.

On your computer click Start → Programs → Accessories → Communication → HyperTerminal. You will then be prompted to enter a name for the communication. Choose any name you like, select an icon, and press OK. Select the port that the board is connected to and press OK. Under bits per second select 9600, and under Stop bits choose 2, and then click OK. You are now ready to control your board.
Table 4: Commands for Serial Program

<table>
<thead>
<tr>
<th>Motor</th>
<th>Forward</th>
<th>Reverse</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>7</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>8</td>
</tr>
</tbody>
</table>

The program will print anything you type on the keyboard back onto the screen for hyper terminal except for the numbers 1-8, which will turn the motors on forward or reverse as described in Table 4.

You can add additional commands, such as stop, using other keys by finding their corresponding values in an ASCII table. This can also be expanded to other functions on the board besides the motor outputs.

Board – Computer: Simple Voltmeter

The Mechatronics board can also be used to send data to the computer. This program uses the built-in ADCs on the mx_ctlr.0, allowing you to take voltage measurements up to 5V. To measure a higher voltage, make a voltage divider and change the program to accommodate for the difference. In the code shown, the positive input voltage will be taken from alternate input 0. For external voltage sources, connect GND on the alternate input port to the ground of the external source.

Hyper terminal will also need to be set up the same way as the previous experiment. The voltmeter works by reading a value from the ADC, which is based on a reference voltage, in this case 5V. Some calculations are performed, giving a corresponding output voltage that can be sent to the computer. Because of the simplicity

```c
#include <mechlib.h>
#include <stdio.h>
#include <mega8535.h>

void USART_Init( void )
{
    int number=0;
    USART_Init();
    while(1)
    {
        number=getchar();
        putchar(number);
        /*similar to multiple if statements*/
        switch(number)
        {
        /*if number=48, then*/
        case 48: coast(1);
        case 49: forward(1);
        case 50: forward(2);
        case 51: forward(3);
        case 52: forward(4);
        break;
        case 53: reverse(1);
        case 54: reverse(2);
        case 55: reverse(3);
        case 56: reverse(4);
        break;
        default: break;
        }
        delay(2);
    }
}

void USART_Init( void )
{
    UBRRH=0x00;
    UBRRL=0x67;
    /* Set frame format: 8data, 2stop bit */
}
```
of this setup, the voltage may off by as much as .05V in the 0-5V range.

Innovative Projects

Bumpbot

The mx_ctlr.0 board can be used for many different things; one of these is as a robotic base. All that needs to be done is to mount motors to a

```c
#include <mechlib.h>
main(){
    while(1){
        forward(1);
        forward(2);
        if(poll_sensor(1)){
            reverse(1);
            delay(3);
            forward(1);
            delay(3);
            forward(2);
            delay(3);
        }
        if(poll_sensor(2)){
            reverse(2);
            delay(3);
            forward(2);
            delay(3);
            forward(1);
            delay(3);
        }
    }
}
```

This program acts as a bumpbot, it will go until a sensor is activated, backup, turn away, and go on. Depending on how the motor are hooked up, the robot will behave differently.

```c
#include <mega8535.h>
#include <mechlib.h> #include <stdio.h>
void USART_Init( void );
void USART_Transmit( int data );
main(){
    int ones=0, tenths=0, hund=0, count=0;
    float number=0;
    USART_Init();
    while(1){
        if(poll_sensor(1)){
            reverse(1);
            reverse(2);
            delay(3);
            forward(2);
            delay(3);
            forward(1);
            delay(3);
            if(poll_sensor(2)){
                reverse(1);
                reverse(2);
                delay(3);
                forward(2);
                delay(3);
                forward(1);
                delay(3);
            }
        }
    }
}
```

This program uses the built in ADC on the chip to make a simple voltmeter. A voltage reading is taken every time a switch is pressed.

```c
#include <mechlib.h>
main(){
    int ones=0, tenths=0, hund=0, count=0;
    float number=0;
    USART_Init();
    while(1){
        if(poll_sensor(1)){
            Reverse(1);
            Reverse(2);
            Delay(3);
            Forward(1);
            Delay(3);
            Forward(2);
            Delay(3);
        }
    }
}
```

This program acts as a bumpbot, it will go until a sensor is activated, backup, turn away, and go on. Depending on how the motor are hooked up, the robot will behave differently.

```
#include <mechlib.h>
main(){
    while(1){
        forward(1);
        forward(2);
        if(poll_sensor(1)){
            reverse(1);
            reverse(2);
            delay(3);
            forward(1);
            delay(3);
            forward(2);
            delay(3);
            if(poll_sensor(2)){
                reverse(1);
                reverse(2);
                delay(3);
                forward(2);
                delay(3);
                forward(1);
                delay(3);
            }
        }
    }
}
```
flat piece of material and attach wheels. The mx_ctlr.0 then controls these motors to maneuver the robot around. This simple robotic base is similar to the TekBots base used in the School of EECS at Oregon State University, and can compete alongside them in the annual TekBots Triathlon.

For the bumpbot program to operate, two limit switches or other sensors will need to be mounted in front of the robot and connected to inputs 1 and 2. It will then roam around aimlessly until a sensor signals a hit, then back up, turn away, and continue forward. If motor 1 and sensor 1 are hooked up on the same side, the robot will backup and turn away from the object that hit the sensor. If the motors and switches are crossed, the robot will turn into the direction of the object.