This writeup describes a rover and controller that was used in a real-time embedded class. The controller has two parts. The first is a controller consisting of a joy stick, an LED matrix display, a Zigbee modem, and an Arduino. The second part, the rover, has a zigbee modem, two motors, a battery, a power board, and a couple microcontrollers.   First we should talk about the controller and the software problems with it.

The LED matrix display will be used to give feedback to the driver, the datasheet for which is posted. The main thing to understand is that this is a matrix, where only a pair of columns are on at a time. Thus it needs to be updated on a regular basis.

DisplayData.h

```
#ifndef DisplayData_h
#define DisplayData_h

#define ColumnSets 5
#define DisplayColumns 10
#define DisplayRows    12

class DisplayData { private: unsigned long DisplayBits[ColumnSets], CurrColumn;
                    public:  int index;
                             int bit;
                             void Initialize();
                             void SetBit(int r, int c);
                            int  GetBit(int r, int c);
                            void ClearBit(int r, int c);
                            unsigned long NextWord(); };

#endif
```

DisplayData.cpp

```
#include <DisplayData.h>
// Object for display
// Code to initialize the structure DisplayData.
void DisplayData::Initialize( )
{
       DisplayBits[0] = 0x01000000;  // Each line represents a set of
       DisplayBits[1] = 0x02000000;  // 24 led's or 2 columns of 12 led's.
       DisplayBits[2] = 0x04000000;
       DisplayBits[3] = 0x08000000;  // Upper bits set the pair of columns.
       DisplayBits[4] = 0x10000000;
    CurrColumn     = 0;  // Current set of led's that are one.
}

// Function to turn on an LED at the location of row and column (r,c).
void DisplayData::SetBit( int r, int c )
{
    unsigned long One = 1;
       index = c % 5;
       bit   = r%12 + 12*((c%10) / 5 );
       DisplayBits[index] |= (One<<bit);
}

// Read back the current value of a bit.
int DisplayData::GetBit( int r, int c )
{
       index = c % 5;
       bit   = r%12 + 12*((c%10) / ColumnSets );
       return( DisplayBits[index] & (1<<bit) );
}

// Function to turn off an LED at the location of row and column (r,c).
void DisplayData::ClearBit( int r, int c )
{
       unsigned long One = 1;
       index = c % 5;
       bit   = r%12 + 12*((c%10) / ColumnSets );
       DisplayBits[index] &= ~(One<<bit);
}
```

```
// This function is called to move to the next pair of columns of LEDs
// and then return the top 16 bits for the column pair, which should
// be sent over to the display via spi.
unsigned long DisplayData::NextWord()
{
	CurrColumn++;
	if( CurrColumn == ColumnSets )
		CurrColumn = 0;
	return ( DisplayBits[CurrColumn] );
}
```

Commands sent to rover, over zigbee link, have the following form. The right motor is controlled by unit 1, while the left motor is controlled by unit 2.

```
;MotorInterface -
;
; This program was created to drive a motor via an H bridge.
;
; The program is part of a system using a Daisy Chained Serial
; (DCS) bus to communicate between a collection of devices.
; One device is the master and will select and send commands to each device.
;
; Commands have a zero in the top bit (bit 7) and bits 6-4 are the device number.
; The exact command is given in the lower 4 bits, some of the command can however
; be used as data.
;
; Command bit pattern 0uuu cccc
; Data bit pattern 1ddd dddd
;
; For the Motor controller, the main/only command is the set duty cycle.
;
; Set Duty Cycle = 0uuu 0DDd / 1ddd dddd
;
; where uuu is a three bit unit number
; DD are the Direction bits 01 "forward" and 10 "backward".
; dddd dddd is the duty cycle with 0x00 is off and 0xff full on.
;
; The response from the unit is the same unit number and direction
; however bit 3 is set and the data bit is set low.
;
; Acknowledge = 0uuu 1DD0
;
```

LED is controlled by Unit 3.

```
; Set Led Flash Command = 0uuu 00LL 1ddd dddd
; where uuu is a three bit unit number
; LL is the led in question (01-red, 10-green, 11-blue)
; ddd dddd is data or "Pulse Width" for each color
;
; The response to an LED set command is 0010 10LL
; (or unit 0x28 or'ed with the led number).
```

Rover Controller Code.

```
#include <MsTimer2.h>

#include <DisplayData.h>

DisplayData DispData;

// Update the display indicating speed of that motor.
void UpdateVertical(int vert, int column)
{
        int V = DisplayRows*vert / 511; // Compute row on display
        int k;
        for (k = 0; k < DisplayRows; k++)  // Set the bits on/off
        if (k < V)
                DispData.SetBit(k, column);
        else
                DispData.ClearBit(k, column);
} // End of UpdateVertical

// Local Spi Code.
#define SpiDataBit 4
#define SpiClkBit 5

// Spi Setup
void Spi_Initialize()
{
        bitSet(DDRB, SpiDataBit);
        bitSet(DDRB, SpiClkBit);
}

// Function to transmit 32 bit number
void Spi_Transmit32(unsigned long out)
{
        int k;

        for (k = 0; k < 32; k++)
        {
                // Set Clock low
                bitClear(PORTB, SpiClkBit);

                if (out & 0x80000000) // If top bit is high,
                        bitSet(PORTB, SpiDataBit);
                else  // or low
                        bitClear(PORTB, SpiDataBit);

                // Set Clock High
                bitSet(PORTB, SpiClkBit);

                // Move next bit into top bit.
                out = out << 1;

        } // End of loop through bits.

        return;

} // End of Spi_Transmit32

// Pull up next word for Display (moves to next columns).
void ServiceDisplay()
{
        unsigned long DisplayWord = DispData.NextWord();
        Spi_Transmit32(DisplayWord);
}
```

```
// Analog Timer
unsigned long AnalogTimer;
#define AnalogUpdateTime 75
int Vert, Horz;
int ControlLeft, ControlRight;

// Rover LED Timer
unsigned long RoverLEDTimer;
#define ROVER_LED_TIME 500

#define ButtonPin 4

// Code that sets motor in motion.
void MoveMotor(int Unit,     // Unit 0 or 1.
        int Motion)  // Motion 0 to 511.
{
        unsigned char FirstChar = 0, SecondChar = 0;
        // Set unit bit
        if (Unit)
                FirstChar |= 0x10;

        // Set direction
        if (Motion >= 255)   // Set Direction
                FirstChar |= 0x04;
        else
                FirstChar |= 0x02;

        // Adjust to forward reverse.
        Motion -= 255;
        if (Motion < 0)
                Motion = -Motion;
        if (Motion < 50) // if speed is less that 50
                Motion = 0;   // Just turn off motor.

        Motion = Motion >> 1;  // shift to 8 bits.
        if (Motion & 0x080)
                FirstChar |= 1;    // Upper bit into FirstChar

        SecondChar = (Motion & 0x7f) | 0x80; // lower 7 bits into next byte
        Serial.write(FirstChar);
        Serial.write(SecondChar);
} // End of Motion function

// Setup of program.
void setup()
{
        // put your setup code here, to run once:
        DispData.Initialize();
        Spi_Initialize();

        // Set up timer to service LED Display.
        MsTimer2::set(4, ServiceDisplay);
        MsTimer2::start();

        // Set timer for Reading analog.
        AnalogTimer = millis();
        analogReference(EXTERNAL);

        // Set timer for running the Rover's LED.
        RoverLEDTimer = millis();

        pinMode(ButtonPin, INPUT);

        Serial.begin(9600);
        Serial.write('\x31'); // Red Off
        Serial.write('\x80');
        Serial.write('\x32'); // Green Off
        Serial.write('\x80');
        Serial.write('\x33'); // Blue Off
        Serial.write('\x80');

}
```

```
// State for flashing LED.
int LED_State = 0;
// loop which is continuous called.
void loop()
{       // Check if it is time to measure input voltages.
        if (millis() - RoverLEDTimer >= ROVER_LED_TIME)
        {
                switch (LED_State)
                {case 0:
                        LED_State = 1; // move to RED on state.
                        Serial.write('\x31'); // Red On
                        Serial.write('\xff');
                        Serial.write('\x33'); // Blue Off
                        Serial.write('\x80');
                        Serial.write('\x32'); // Green Off
                        Serial.write('\x80');
                        break;
                 case 1:
                        LED_State = 2; // move to Blue on state.
                        Serial.write('\x31'); // Red Off
                        Serial.write('\x80');
                        Serial.write('\x33'); // Blue On
                        Serial.write('\xff');
                        Serial.write('\x32'); // Green Off
                        Serial.write('\x80');
                        break;
                 case 2:
                        LED_State = 0; // move to GREEN on state.
                        Serial.write('\x31'); // Red Off
                        Serial.write('\x80');
                        Serial.write('\x33'); // Blue Off
                        Serial.write('\x80');
                        Serial.write('\x32'); // Green On
                        Serial.write('\xff');
                        break;
                }
                // Update timer value.
                RoverLEDTimer += ROVER_LED_TIME;
        } // End of RoverLEDTimer

        // Check if it is time to measure input voltages.
        if (millis() - AnalogTimer >= AnalogUpdateTime)
        {
                Vert = analogRead(1) / 2;
                Horz = analogRead(0) / 2;
                ControlRight = Horz + (Vert - 255); // Convert analog data into
                ControlLeft = Horz - (Vert - 255);  // Left and Right motor motion.
                // Limit range of controls.
                if (ControlLeft < 0)
                        ControlLeft = 0;
                else if (ControlLeft >= 511)
                        ControlLeft = 511;
                if (ControlRight < 0)
                        ControlRight = 0;
                else if (ControlRight >= 511)
                        ControlRight = 511;
                // Send motion and set up set display.
                MoveMotor(0, ControlLeft);
                MoveMotor(1, 511 - ControlRight); // Flip since motor in other direction.
                UpdateVertical(ControlLeft, 0);
                UpdateVertical(ControlRight, 9);

                // Update timer value.
                AnalogTimer += AnalogUpdateTime;

                // Watch switch.
                if (digitalRead(ButtonPin) == LOW)
                        DispData.SetBit(11, 1);
                else
                        DispData.ClearBit(11, 1);

        } // End of AnalogUpdateTimer.
        // Service Serial port, no return really being used.
        if (Serial.available())
        {
                Serial.read();
        }// End of Serial service

} // End of loop
```