Xilun Jin, Zekun Wei
05/06/2016
ECE 110 Honor

Final Report for
Electric Skateboard

I. Introduction:

A. The Formation of our idea

There are always some classrooms far away from our dorm. We wanted to spend less time on the road. And we also wanted a easy and cool way to get around campus. So we came up with the idea — building an electric skateboard. But considering of the fund, we decided to build a model skateboard instead.

Our skateboard should

- Be powered by electric motors
- Be controlled remotely
- Display the speed to the user.

B. Our Solutions

- Use the motors in the ECE 110 kit to power our skateboard.
- Use Bluetooth board to make the skateboard can be controlled by the cellphone.
- Use the hall effect sensor to count the revolution in a time duration

II. Design:

A. Block Diagrams

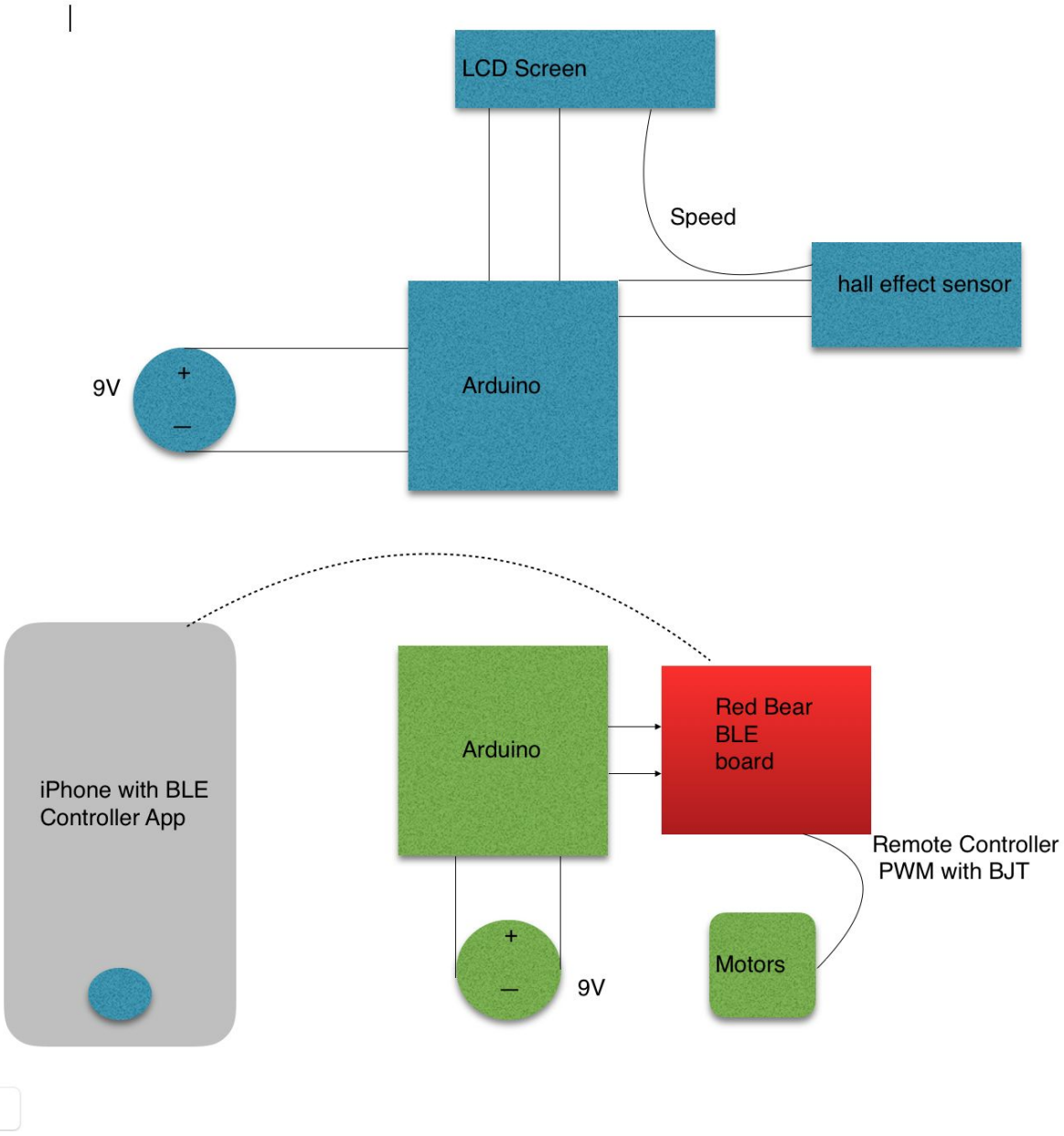

Figure 1

B. Written Descriptions of Blocks

1.The top diagram in *Figure 1*, the LCD and Hall Effect sensor diagram, shows how we built the speedometer.

We sticked a magnet inside the wheel (see *Figure 2* below), so we can use a hall effect sensor to count the number of revolutions of the wheel. The hall effect sensor will send 0s and 1s to arduino. In our case, it sends 1 to the arduino when it is not in a magnetic field, and sends 0 otherwises. We programmed the arduino so that it can count the times of change of the data from 1 to 0 (see *Appendix : Code Used*).

We mounted the LCD display on the top of our skateboard (see *Figure 3*) so we can see the current speed of the skateboard. The connections of the LCD are a little complicate, so we just followed the tutorial from adafruit.com. We have a potentiometer, which can be used to control the contrast of the screen.

2. The bottom diagram is the Bluetooth controller part of our project.

We purchased the Bluetooth board from RedBear Lab. And at the same time, we also got the free application, which is called BLE Controller, provided by them (see *Figure 5* and *Figure 6*). We installed the Bluetooth board on the arduino board, and powered it with 9V power source. As showed in the *Figure 6* below, we can either read in or output voltage through the application interface. In our design, we used the PWM output with BJTs to control the speed of the motor.

C. Drawings/Pictures of device

Figure 2

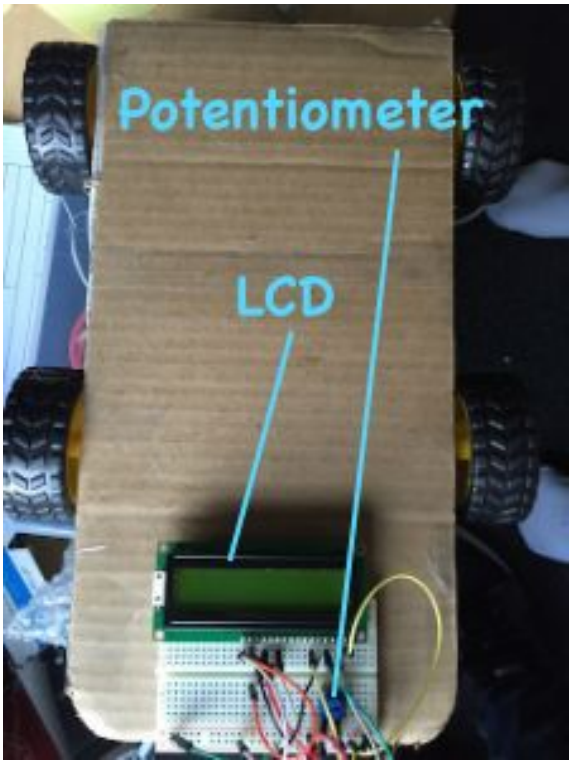

Figure 3

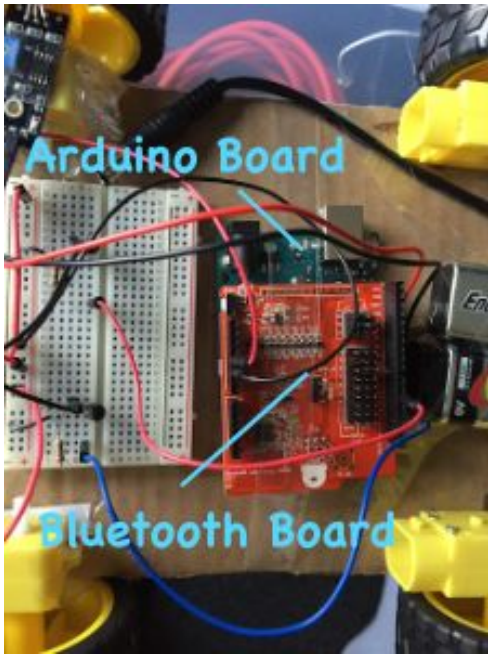

Figure 4

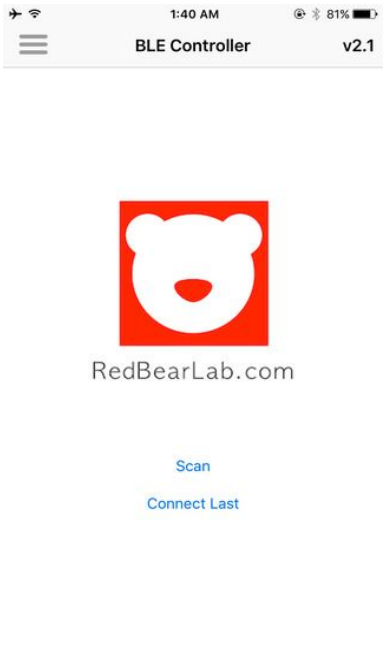

Figure 5

Figure 6

III. Results

A. Present Results

We accomplished three of our goals:

- Power it with electric motors.
- Build a speedometer.
- Add a remote controller.

B. Qualitative Analysis of results

Electric motor:

Our motor is sufficient to drive our model skateboard. But if we want to power a real skateboard, we will need much more powerful motors.

Speedometer:

Our speedometer can successfully display the current speed. However, we think the speedometer won't appear in a real skateboard because people won't able to read it when they are riding the skateboard.

Remote controller:

We used our cell phone as the controller. Because we provide the voltage separately to the two motor, we can not only control the speed of the skateboard but also the direction it turns. However, in a real skateboard, cell phone can be really hard to used to control it when the skateboard is moving, and it can easily cause accident.

C. Quantitative Analysis of results

Electric motor:

The highest speed our skateboard can run is approximately 0.76 m/s, which is measured by our speedometer (This speed may not be accurate and the reason will be stated below). The speed of a real skateboard will be around 25--40 kilometres per hour, or 7--11 m/s, which is much faster than our board. So when it comes to build a real skateboard, we will need a huge adjustment.

Speedometer:

Our speedometer can display the current speed of the wheel. But there is a flaw here. Because of the method we used to measure and calculate the speed (measure the number of revolutions in 1s and multiply it with the length circumference of the wheel), the speed can only be the multiples of 0.19, which is the length of the circumference of the wheel. One of our solutions is adding more magnets to the wheel to decrease the "base number" (in our case, it is 0.19).

IV. Future Work

Next semester we will build a real electric skateboard and try to implement all the functions same as in our model on it.

Speedometer

We will not put a screen on our skateboard because it will be too dangerous if people try to read the speed when they are riding the skateboard. Instead, we will build an app and store the speed information in the app. Then the clients will be able to track the the speed or the distance they travel during a day.

Remote controller

We will change the remote control for the skateboard because of safety issue. It's may be a little danger to control the skateboard using your phone while it's moving. Instead, we will add new control system that is gesture control system. People can control the skateboard to speed up and break using their body movement. For detailing way of controlling, we are doing the research online and we will ask some professors' opinions who specialize in gesture controlling area.

V. Conclusion

A. What Worked?

All the parts in the electric skateboard are working as we desired. The hall effect sensor helps us to calculate the speed of how fast the skateboard is moving by putting a magnet on the wheel and the sensor can respond to the magnetic field when the magnet gets close to the hall effect sensor. In this way, we can count the revolution of wheel in a certain time period. And we can use that information to calculate the current speed of the skateboard and display it on the LCD screen which is on the top of the skateboard. Another important part of this skateboard is remote control function. We used the redbear BLE board as remote controller. We can use BLE board and IPhone to achieve the bluetooth communication between skateboard and IPhone in order to control the PWM value of each motor on our phone.

B. What did we learn?

The most difficult part of the project is how to use hall effect sensor to measure the speed of skateboard. We can only get 0(when magnet get close to the hall effect sensor) and 1 (when magnet is far away) on serial monitor but we don't know how to use these 0 and 1 to get the speed of skateboard on serial monitor. After doing some research online, we learned that we can count the times of change of the data from 1 to 0 in a certain period of time and using that revolution per time multiply by the circumference of wheel to get the final speed. And we also did some research on how to display the data on LCD instead of serial monitor. Also, we chose the bluetooth communication because we believe the most important part for remote controlling are communication distance between the skateboard and the energy usage for the remote controller. BLE communication are best choice in this project because the distance of BLE communication between remote controller and client are about 100 meter without obstacle in front of them. And it's also use little energy for both phone and BLE board. A 9V battery can be used for a pretty long time for just powering up the BLE board .

Above all, the most important takeaway from this project is that we must make use of the Internet. Google knows many things we don't know.

VI. Appendix

A.Code used:

Speedometer(LCD Display and the Hall Effect Sensor):

```
#include <LiquidCrystal.h>
const int hePin = 4;
bool heState = 0;
unsigned long time;
const unsigned long fixedTime = 1000;

LiquidCrystal lcd(7, 8, 9, 10, 11, 12);

void setup() {
  // set up the LCD's number of columns and rows:
  lcd.begin(16, 2);
  // Print a message to the LCD.
  lcd.print("Current Speed");

}

void loop() {
  //calculate the speed
  int rps = count();
  double cir = 0.06000000*3.1415926;
  double sp = double(rps) * cir;
  //print the speed on the LCD display
  lcd.setCursor(0, 1);
  lcd.print("speed ");
  lcd.print(sp);
  lcd.print(" m/s");
}
int count()
{
  int count=0;
  boolean flag=LOW;
  unsigned long curTime=0;
  unsigned long startTime=millis();
  while (curTime<=fixedTime)
  {
    if (!digitalRead(hePin)==HIGH)
    {
      flag=HIGH;
    }
    if (!digitalRead(hePin)==LOW && flag==HIGH)
    {
      count++;
      flag=LOW;
    }
    curTime=millis()-startTime;
  }
  int count2rps = count;
  return count2rps;
}
```

Bluetooth Controller (This is provided by RedBear Lab):

Part 1

```
/*

Copyright (c) 2012, 2013 RedBearLab

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and
associated documentation files (the "Software"), to deal in the Software without restriction,
including without limitation the rights to use, copy, modify, merge, publish, distribute,
sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or
substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT
NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND
NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM,
DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

*/

#include <Servo.h>
#include <SPI.h>
#include <EEPROM.h>
#include <boards.h>
#include <RBL_nRF8001.h>
#include "Boards.h"

#define PROTOCOL_MAJOR_VERSION   0 //
#define PROTOCOL_MINOR_VERSION   0 //
#define PROTOCOL_BUGFIX_VERSION  2 // bugfix

#define PIN_CAPABILITY_NONE      0x00
#define PIN_CAPABILITY_DIGITAL   0x01
#define PIN_CAPABILITY_ANALOG    0x02
#define PIN_CAPABILITY_PWM       0x04
#define PIN_CAPABILITY_SERVO     0x08
#define PIN_CAPABILITY_I2C       0x10

// pin modes
//#define INPUT                 0x00 // defined in wiring.h
//#define OUTPUT                0x01 // defined in wiring.h
#define ANALOG                  0x02 // analog pin in analogInput mode
#define PWM                     0x03 // digital pin in PWM output mode
#define SERVO                   0x04 // digital pin in Servo output mode

byte pin_mode[TOTAL_PINS];
byte pin_state[TOTAL_PINS];
byte pin_pwm[TOTAL_PINS];
byte pin_servo[TOTAL_PINS];

Servo servos[MAX_SERVOS];

void setup()
{
  Serial.begin(57600);
```

```
  Serial.println("BLE Arduino Slave");

  /* Default all to digital input */
  for (int pin = 0; pin < TOTAL_PINS; pin++)
  {
    // Set pin to input with internal pull up
    pinMode(pin, INPUT);
    digitalWrite(pin, HIGH);

    // Save pin mode and state
    pin_mode[pin] = INPUT;
    pin_state[pin] = LOW;
  }

  // Default pins set to 9 and 8 for REQN and RDYN
  // Set your REQN and RDYN here before ble_begin() if you need
  //ble_set_pins(3, 2);

  // Set your BLE Shield name here, max. length 10
  //ble_set_name("My Name");

  // Init. and start BLE library.
  ble_begin();
}
static byte buf_len = 0;
void ble_write_string(byte *bytes, uint8_t len)
{
  if (buf_len + len > 20)
  {
    for (int j = 0; j < 15000; j++)
      ble_do_events();
    buf_len = 0;
  }
  for (int j = 0; j < len; j++)
  {
    ble_write(bytes[j]);
    buf_len++;
  }
  if (buf_len == 20)
  {
    for (int j = 0; j < 15000; j++)
      ble_do_events();
    buf_len = 0;
  }
}
byte reportDigitalInput()
{
  if (!ble_connected())
    return 0;
  static byte pin = 0;
  byte report = 0;
  if (!IS_PIN_DIGITAL(pin))
  {
    pin++;
    if (pin >= TOTAL_PINS)
      pin = 0;
    return 0;
```

```
  }
  if (pin_mode[pin] == INPUT)
  {
      byte current_state = digitalRead(pin);
      if (pin_state[pin] != current_state)
      {
        pin_state[pin] = current_state;
        byte buf[] = {'G', pin, INPUT, current_state};
        ble_write_string(buf, 4);
        report = 1;
      }
  }
  pin++;
  if (pin >= TOTAL_PINS)
    pin = 0;
  return report;
}
void reportPinCapability(byte pin)
{
  byte buf[] = {'P', pin, 0x00};
  byte pin_cap = 0;
  if (IS_PIN_DIGITAL(pin))
    pin_cap |= PIN_CAPABILITY_DIGITAL;
  if (IS_PIN_ANALOG(pin))
    pin_cap |= PIN_CAPABILITY_ANALOG;
  if (IS_PIN_PWM(pin))
    pin_cap |= PIN_CAPABILITY_PWM;
  if (IS_PIN_SERVO(pin))
    pin_cap |= PIN_CAPABILITY_SERVO;
  buf[2] = pin_cap;
  ble_write_string(buf, 3);
}
void reportPinServoData(byte pin)
{
//  if (IS_PIN_SERVO(pin))
//    servos[PIN_TO_SERVO(pin)].write(value);
//  pin_servo[pin] = value;
  byte value = pin_servo[pin];
  byte mode = pin_mode[pin];
  byte buf[] = {'G', pin, mode, value};
  ble_write_string(buf, 4);
}
byte reportPinAnalogData()
{
  if (!ble_connected())
    return 0;
  static byte pin = 0;
  byte report = 0;
  if (!IS_PIN_DIGITAL(pin))
  {
    pin++;
    if (pin >= TOTAL_PINS)
      pin = 0;
    return 0;
  }
  if (pin_mode[pin] == ANALOG)
  {
```

```
    uint16_t value = analogRead(pin);
    byte value_lo = value;
    byte value_hi = value>>8;
    byte mode = pin_mode[pin];
    mode = (value_hi << 4) | mode;
    byte buf[] = {'G', pin, mode, value_lo};
    ble_write_string(buf, 4);
  }
  pin++;
  if (pin >= TOTAL_PINS)
    pin = 0;
  return report;
}
void reportPinDigitalData(byte pin)
{
  byte state = digitalRead(pin);
  byte mode = pin_mode[pin];
  byte buf[] = {'G', pin, mode, state};
  ble_write_string(buf, 4);
}

void reportPinPWMData(byte pin)
{
  byte value = pin_pwm[pin];
  byte mode = pin_mode[pin];
  byte buf[] = {'G', pin, mode, value};
  ble_write_string(buf, 4);
}

void sendCustomData(uint8_t *buf, uint8_t len)
{
  uint8_t data[20] = "Z";
  memcpy(&data[1], buf, len);
  ble_write_string(data, len+1);
}

byte queryDone = false;

void loop()
{
  while(ble_available())
  {
    byte cmd;
    cmd = ble_read();
    Serial.write(cmd);
    // Parse data here
    switch (cmd)
    {
      case 'V': // query protocol version
        {
          byte buf[] = {'V', 0x00, 0x00, 0x01};
          ble_write_string(buf, 4);
        }
        break;
      case 'C': // query board total pin count
        {
          byte buf[2];
```

```
          buf[0] = 'C';
          buf[1] = TOTAL_PINS;
          ble_write_string(buf, 2);
        }
        break;
      case 'M': // query pin mode
        {
          byte pin = ble_read();
          byte buf[] = {'M', pin, pin_mode[pin]}; // report pin mode
          ble_write_string(buf, 3);
        }
        break;
      case 'S': // set pin mode
        {
          byte pin = ble_read();
          byte mode = ble_read();
          if (IS_PIN_SERVO(pin) && mode != SERVO && servos[PIN_TO_SERVO(pin)].attached())
            servos[PIN_TO_SERVO(pin)].detach();
          /* ToDo: check the mode is in its capability or not */
          /* assume always ok */
          if (mode != pin_mode[pin])
          {
            pinMode(pin, mode);
            pin_mode[pin] = mode;
            if (mode == OUTPUT)
            {
              digitalWrite(pin, LOW);
              pin_state[pin] = LOW;
            }
            else if (mode == INPUT)
            {
              digitalWrite(pin, HIGH);
              pin_state[pin] = HIGH;
            }
            else if (mode == ANALOG)
            {
              if (IS_PIN_ANALOG(pin)) {
                if (IS_PIN_DIGITAL(pin)) {
                  pinMode(PIN_TO_DIGITAL(pin), LOW);
                }
              }
            }
            else if (mode == PWM)
            {
              if (IS_PIN_PWM(pin))
              {
                pinMode(PIN_TO_PWM(pin), OUTPUT);
                analogWrite(PIN_TO_PWM(pin), 0);
                pin_pwm[pin] = 0;
                pin_mode[pin] = PWM;
              }
            }
            else if (mode == SERVO)
            {
              if (IS_PIN_SERVO(pin))
              {
                pin_servo[pin] = 0;
```

```
                pin_mode[pin] = SERVO;
                if (!servos[PIN_TO_SERVO(pin)].attached())
                  servos[PIN_TO_SERVO(pin)].attach(PIN_TO_DIGITAL(pin));
              }
            }
          }
  //         if (mode == ANALOG)
  //           reportPinAnalogData(pin);
          if ( (mode == INPUT) || (mode == OUTPUT) )
            reportPinDigitalData(pin);
          else if (mode == PWM)
            reportPinPWMData(pin);
          else if (mode == SERVO)
            reportPinServoData(pin);
        }
        break;
      case 'G': // query pin data
        {
          byte pin = ble_read();
          reportPinDigitalData(pin);
        }
        break;
      case 'T': // set pin digital state
        {
          byte pin = ble_read();
          byte state = ble_read();
          digitalWrite(pin, state);
          reportPinDigitalData(pin);
        }
        break;
      case 'N': // set PWM
        {
          byte pin = ble_read();
          byte value = ble_read();
          analogWrite(PIN_TO_PWM(pin), value);
          pin_pwm[pin] = value;
          reportPinPWMData(pin);
        }
        break;
      case 'O': // set Servo
        {
          byte pin = ble_read();
          byte value = ble_read();
          if (IS_PIN_SERVO(pin))
            servos[PIN_TO_SERVO(pin)].write(value);
          pin_servo[pin] = value;
          reportPinServoData(pin);
        }
        break;
      case 'A': // query all pin status
        for (int pin = 0; pin < TOTAL_PINS; pin++)
        {
          reportPinCapability(pin);
          if ( (pin_mode[pin] == INPUT) || (pin_mode[pin] == OUTPUT) )
            reportPinDigitalData(pin);
          else if (pin_mode[pin] == PWM)
            reportPinPWMData(pin);
```

```
          else if (pin_mode[pin] == SERVO)
            reportPinServoData(pin);
        }
        queryDone = true;
        {
          uint8_t str[] = "ABC";
          sendCustomData(str, 3);
        }
        break;
      case 'P': // query pin capability
        {
          byte pin = ble_read();
          reportPinCapability(pin);
        }
        break;
      case 'Z':
        {
          byte len = ble_read();
          byte buf[len];
          for (int i=0;i<len;i++)
            buf[i] = ble_read();
          Serial.println("->");
          Serial.print("Received: ");
          Serial.print(len);
          Serial.println(" byte(s)");
          Serial.print(" Hex: ");
          for (int i=0;i<len;i++)
            Serial.print(buf[i], HEX);
          Serial.println();
        }
    }
    // send out any outstanding data
    ble_do_events();
    buf_len = 0;
    return; // only do this task in this loop
  }
  // process text data
  if (Serial.available())
  {
    byte d = 'Z';
    ble_write(d);
    delay(5);
    while(Serial.available())
    {
      d = Serial.read();
      ble_write(d);
    }
    ble_do_events();
    buf_len = 0;
    return;
  }
  // No input data, no commands, process analog data
  if (!ble_connected())
    queryDone = false; // reset query state
  if (queryDone) // only report data after the query state
  {
    byte input_data_pending = reportDigitalInput();
```

```
    if (input_data_pending)
    {
      ble_do_events();
      buf_len = 0;
      return; // only do this task in this loop
    }
    reportPinAnalogData();
    ble_do_events();
    buf_len = 0;
    return;
  }
  ble_do_events();
  buf_len = 0;
}
```

Part 2

```
/* Boards.h - Hardware Abstraction Layer for Firmata library */

#ifndef Boards_h
#define Boards_h

#include <inttypes.h>

#if defined(ARDUINO) && ARDUINO >= 100
#include "Arduino.h"  // for digitalRead, digitalWrite, etc
#else
#include "WProgram.h"
#endif

// Normally Servo.h must be included before Firmata.h (which then includes
// this file).  If Servo.h wasn't included, this allows the code to still
// compile, but without support for any Servos.  Hopefully that's what the
// user intended by not including Servo.h
#ifndef MAX_SERVOS
#define MAX_SERVOS 0
#endif

/*
 Firmata Hardware Abstraction Layer

 Firmata is built on top of the hardware abstraction functions of Arduino,
 specifically digitalWrite, digitalRead, analogWrite, analogRead, and
 pinMode.  While these functions offer simple integer pin numbers, Firmata
 needs more information than is provided by Arduino.  This file provides
 all other hardware specific details.  To make Firmata support a new board,
 only this file should require editing.

 The key concept is every "pin" implemented by Firmata may be mapped to
 any pin as implemented by Arduino.  Usually a simple 1-to-1 mapping is
 best, but such mapping should not be assumed.  This hardware abstraction
 layer allows Firmata to implement any number of pins which map onto the
```

```
Arduino implemented pins in almost any arbitrary way.


General Constants:

These constants provide basic information Firmata requires.

TOTAL_PINS: The total number of pins Firmata implemented by Firmata.
Usually this will match the number of pins the Arduino functions
implement, including any pins pins capable of analog or digital.
However, Firmata may implement any number of pins.  For example,
on Arduino Mini with 8 analog inputs, 6 of these may be used
for digital functions, and 2 are analog only.  On such boards,
Firmata can implement more pins than Arduino's pinMode()
function, in order to accommodate those special pins.  The
Firmata protocol supports a maximum of 128 pins, so this
constant must not exceed 128.

TOTAL_ANALOG_PINS: The total number of analog input pins implemented.
The Firmata protocol allows up to 16 analog inputs, accessed
using offsets 0 to 15.  Because Firmata presents the analog
inputs using different offsets than the actual pin numbers
(a legacy of Arduino's analogRead function, and the way the
analog input capable pins are physically labeled on all
Arduino boards), the total number of analog input signals
must be specified.  16 is the maximum.

VERSION_BLINK_PIN: When Firmata starts up, it will blink the version
number.  This constant is the Arduino pin number where a
LED is connected.


Pin Mapping Macros:

These macros provide the mapping between pins as implemented by
Firmata protocol and the actual pin numbers used by the Arduino
functions.  Even though such mappings are often simple, pin
numbers received by Firmata protocol should always be used as
input to these macros, and the result of the macro should be
used with with any Arduino function.

When Firmata is extended to support a new pin mode or feature,
a pair of macros should be added and used for all hardware
access.  For simple 1:1 mapping, these macros add no actual
overhead, yet their consistent use allows source code which
uses them consistently to be easily adapted to all other boards
with different requirements.

IS_PIN_XXXX(pin): The IS_PIN macros resolve to true or non-zero
if a pin as implemented by Firmata corresponds to a pin
that actually implements the named feature.

PIN_TO_XXXX(pin): The PIN_TO macros translate pin numbers as
implemented by Firmata to the pin numbers needed as inputs
to the Arduino functions.  The corresponding IS_PIN macro
should always be tested before using a PIN_TO macro, so
these macros only need to handle valid Firmata pin
```

```
 numbers for the named feature.


 Port Access Inline Funtions:

 For efficiency, Firmata protocol provides access to digital
 input and output pins grouped by 8 bit ports.  When these
 groups of 8 correspond to actual 8 bit ports as implemented
 by the hardware, these inline functions can provide high
 speed direct port access.  Otherwise, a default implementation
 using 8 calls to digitalWrite or digitalRead is used.

 When porting Firmata to a new board, it is recommended to
 use the default functions first and focus only on the constants
 and macros above.  When those are working, if optimized port
 access is desired, these inline functions may be extended.
 The recommended approach defines a symbol indicating which
 optimization to use, and then conditional complication is
 used within these functions.

 readPort(port, bitmask):  Read an 8 bit port, returning the value.
 port:    The port number, Firmata pins port*8 to port*8+7
 bitmask: The actual pins to read, indicated by 1 bits.

 writePort(port, value, bitmask):  Write an 8 bit port.
 port:    The port number, Firmata pins port*8 to port*8+7
 value:   The 8 bit value to write
 bitmask: The actual pins to write, indicated by 1 bits.
 */

/*==============================================================================
 * Board Specific Configuration
 *============================================================================*/

#ifndef digitalPinHasPWM
#define digitalPinHasPWM(p)      IS_PIN_DIGITAL(p)
#endif

// Arduino Duemilanove, Diecimila, and NG
#if defined(__AVR_ATmega168__) || defined(__AVR_ATmega328P__)
#if defined(NUM_ANALOG_INPUTS) && NUM_ANALOG_INPUTS == 6
#define TOTAL_ANALOG_PINS        6
#define TOTAL_PINS               20 // 14 digital + 6 analog
#else
#define TOTAL_ANALOG_PINS        8
#define TOTAL_PINS               22 // 14 digital + 8 analog
#endif
#define VERSION_BLINK_PIN        13
#define IS_PIN_DIGITAL(p)        (((p) >= 2 && (p) <= 19) && !((p) >= 8 && (p) <= 13))
//#define IS_PIN_DIGITAL(p)        ( ((p) >= 2 && (p) <= 7) || ((p) >= 13 && (p) <= 19) )
#define IS_PIN_ANALOG(p)         ((p) >= 14 && (p) < 14 + TOTAL_ANALOG_PINS)
#define IS_PIN_PWM(p)            (digitalPinHasPWM(p) && !((p) >= 8 && (p) <= 12))
#define IS_PIN_SERVO(p)          ( (p) >= 2 && (p) <= 7 )
#define IS_PIN_I2C(p)            ((p) == 18 || (p) == 19)
#define PIN_TO_DIGITAL(p)        (p)
#define PIN_TO_ANALOG(p)         ((p) - 14)
#define PIN_TO_PWM(p)            PIN_TO_DIGITAL(p)
```

```
#define PIN_TO_SERVO(p)          ((p) - 2)
#define ARDUINO_PINOUT_OPTIMIZE 1


// Wiring (and board)
#elif defined(WIRING)
#define VERSION_BLINK_PIN        WLED
#define IS_PIN_DIGITAL(p)        ((p) >= 0 && (p) < TOTAL_PINS)
#define IS_PIN_ANALOG(p)         ((p) >= FIRST_ANALOG_PIN && (p) <
(FIRST_ANALOG_PIN+TOTAL_ANALOG_PINS))
#define IS_PIN_PWM(p)            digitalPinHasPWM(p)
#define IS_PIN_SERVO(p)          ((p) >= 0 && (p) < MAX_SERVOS)
#define IS_PIN_I2C(p)            ((p) == SDA || (p) == SCL)
#define PIN_TO_DIGITAL(p)        (p)
#define PIN_TO_ANALOG(p)         ((p) - FIRST_ANALOG_PIN)
#define PIN_TO_PWM(p)            PIN_TO_DIGITAL(p)
#define PIN_TO_SERVO(p)          (p)


// old Arduinos
#elif defined(__AVR_ATmega8__)
#define TOTAL_ANALOG_PINS        6
#define TOTAL_PINS               20 // 14 digital + 6 analog
#define VERSION_BLINK_PIN        13
#define IS_PIN_DIGITAL(p)        ((p) >= 2 && (p) <= 19)
#define IS_PIN_ANALOG(p)         ((p) >= 14 && (p) <= 19)
#define IS_PIN_PWM(p)            digitalPinHasPWM(p)
#define IS_PIN_SERVO(p)          (IS_PIN_DIGITAL(p) && (p) - 2 < MAX_SERVOS)
#define IS_PIN_I2C(p)            ((p) == 18 || (p) == 19)
#define PIN_TO_DIGITAL(p)        (p)
#define PIN_TO_ANALOG(p)         ((p) - 14)
#define PIN_TO_PWM(p)            PIN_TO_DIGITAL(p)
#define PIN_TO_SERVO(p)          ((p) - 2)
#define ARDUINO_PINOUT_OPTIMIZE 1


// Arduino Mega
#elif defined(__AVR_ATmega1280__) || defined(__AVR_ATmega2560__)
#define TOTAL_ANALOG_PINS        16
#define TOTAL_PINS               70 // 54 digital + 16 analog
#define VERSION_BLINK_PIN        13
#define IS_PIN_DIGITAL(p)        ((p) >= 2 && (p) < TOTAL_PINS)
#define IS_PIN_ANALOG(p)         ((p) >= 54 && (p) < TOTAL_PINS)
#define IS_PIN_PWM(p)            digitalPinHasPWM(p)
#define IS_PIN_SERVO(p)          ((p) >= 2 && (p) - 2 < MAX_SERVOS)
#define IS_PIN_I2C(p)            ((p) == 20 || (p) == 21)
#define PIN_TO_DIGITAL(p)        (p)
#define PIN_TO_ANALOG(p)         ((p) - 54)
#define PIN_TO_PWM(p)            PIN_TO_DIGITAL(p)
#define PIN_TO_SERVO(p)          ((p) - 2)


// Teensy 1.0
#elif defined(__AVR_AT90USB162__)
#define TOTAL_ANALOG_PINS        0
#define TOTAL_PINS               21 // 21 digital + no analog
#define VERSION_BLINK_PIN        6
```

```
#define IS_PIN_DIGITAL(p)        ((p) >= 0 && (p) < TOTAL_PINS)
#define IS_PIN_ANALOG(p)         (0)
#define IS_PIN_PWM(p)            digitalPinHasPWM(p)
#define IS_PIN_SERVO(p)          ((p) >= 0 && (p) < MAX_SERVOS)
#define IS_PIN_I2C(p)            (0)
#define PIN_TO_DIGITAL(p)        (p)
#define PIN_TO_ANALOG(p)         (0)
#define PIN_TO_PWM(p)            PIN_TO_DIGITAL(p)
#define PIN_TO_SERVO(p)          (p)


// Blend Micro
#elif defined(BLEND_MICRO)
#define TOTAL_ANALOG_PINS        6
#define TOTAL_PINS               24 // 11 digital + 12 analog
#define VERSION_BLINK_PIN        13
#define IS_PIN_DIGITAL(p)        ( (p) >= 0 && (p) < 24 && !((p) == 4) && !((p) >= 6 && (p) <=
7) && !((p) >=14 && (p) <= 17) )
#define IS_PIN_ANALOG(p)         ((p) >= 18 && (p) < 24)
#define IS_PIN_PWM(p)            ( (p) == 3 || (p) == 5 || (p) == 9 || (p) == 10 || (p) == 11
|| (p) == 13 )
#define IS_PIN_SERVO(p)          ( (p) >= 0 && (p) < MAX_SERVOS && !((p) == 4) && !((p) >= 6 &&
(p) <= 7) )
#define IS_PIN_I2C(p)            ((p) == 5 || (p) == 6)
#define PIN_TO_DIGITAL(p)        (p)
#define PIN_TO_ANALOG(p)         ((p)-18)
#define PIN_TO_PWM(p)            PIN_TO_DIGITAL(p)
#define PIN_TO_SERVO(p)          (p)


// Teensy 2.0
#elif defined(__AVR_ATmega32U4__)
#define TOTAL_ANALOG_PINS        6
#define TOTAL_PINS               24 // 11 digital + 12 analog
#define VERSION_BLINK_PIN        13
#define IS_PIN_DIGITAL(p)        ( (p) >= 0 && (p) < 24 && !((p) >= 8 && (p) <= 9) && !((p)
>=14 && (p) <= 17) )
#define IS_PIN_ANALOG(p)         ((p) >= 18 && (p) < 24)
#define IS_PIN_PWM(p)            ( (p) == 3 || (p) == 5 || (p) == 6 || (p) == 10 || (p) == 11
|| (p) == 13 )
#define IS_PIN_SERVO(p)          ( (p) >= 0 && (p) < MAX_SERVOS && !((p) >= 8 && (p) <= 9) )
#define IS_PIN_I2C(p)            ((p) == 5 || (p) == 6)
#define PIN_TO_DIGITAL(p)        (p)
#define PIN_TO_ANALOG(p)         ((p)-18)
#define PIN_TO_PWM(p)            PIN_TO_DIGITAL(p)
#define PIN_TO_SERVO(p)          (p)


// Teensy++ 1.0 and 2.0
#elif defined(__AVR_AT90USB646__) || defined(__AVR_AT90USB1286__)
#define TOTAL_ANALOG_PINS        8
#define TOTAL_PINS               46 // 38 digital + 8 analog
#define VERSION_BLINK_PIN        6
#define IS_PIN_DIGITAL(p)        ((p) >= 0 && (p) < TOTAL_PINS)
#define IS_PIN_ANALOG(p)         ((p) >= 38 && (p) < TOTAL_PINS)
#define IS_PIN_PWM(p)            digitalPinHasPWM(p)
#define IS_PIN_SERVO(p)          ((p) >= 0 && (p) < MAX_SERVOS)
```

```
#define IS_PIN_I2C(p)            ((p) == 0 || (p) == 1)
#define PIN_TO_DIGITAL(p)        (p)
#define PIN_TO_ANALOG(p)         ((p) - 38)
#define PIN_TO_PWM(p)            PIN_TO_DIGITAL(p)
#define PIN_TO_SERVO(p)          (p)


// Sanguino
#elif defined(__AVR_ATmega644P__) || defined(__AVR_ATmega644__)
#define TOTAL_ANALOG_PINS        8
#define TOTAL_PINS               32 // 24 digital + 8 analog
#define VERSION_BLINK_PIN        0
#define IS_PIN_DIGITAL(p)        ((p) >= 2 && (p) < TOTAL_PINS)
#define IS_PIN_ANALOG(p)         ((p) >= 24 && (p) < TOTAL_PINS)
#define IS_PIN_PWM(p)            digitalPinHasPWM(p)
#define IS_PIN_SERVO(p)          ((p) >= 0 && (p) < MAX_SERVOS)
#define IS_PIN_I2C(p)            ((p) == 16 || (p) == 17)
#define PIN_TO_DIGITAL(p)        (p)
#define PIN_TO_ANALOG(p)         ((p) - 24)
#define PIN_TO_PWM(p)            PIN_TO_DIGITAL(p)
#define PIN_TO_SERVO(p)          ((p) - 2)


// Illuminato
#elif defined(__AVR_ATmega645__)
#define TOTAL_ANALOG_PINS        6
#define TOTAL_PINS               42 // 36 digital + 6 analog
#define VERSION_BLINK_PIN        13
#define IS_PIN_DIGITAL(p)        ((p) >= 2 && (p) < TOTAL_PINS)
#define IS_PIN_ANALOG(p)         ((p) >= 36 && (p) < TOTAL_PINS)
#define IS_PIN_PWM(p)            digitalPinHasPWM(p)
#define IS_PIN_SERVO(p)          ((p) >= 0 && (p) < MAX_SERVOS)
#define IS_PIN_I2C(p)            ((p) == 4 || (p) == 5)
#define PIN_TO_DIGITAL(p)        (p)
#define PIN_TO_ANALOG(p)         ((p) - 36)
#define PIN_TO_PWM(p)            PIN_TO_DIGITAL(p)
#define PIN_TO_SERVO(p)          ((p) - 2)


// Arduino DUE
#elif defined(__SAM3X8E__)
#define TOTAL_ANALOG_PINS        12
#define TOTAL_PINS               66 // 54 digital + 12 analog
#define VERSION_BLINK_PIN        13
#define IS_PIN_DIGITAL(p)        ((p) >= 2 && (p) < TOTAL_PINS)
#define IS_PIN_ANALOG(p)         ((p) >= 54 && (p) < TOTAL_PINS)
#define IS_PIN_PWM(p)            digitalPinHasPWM(p)
#define IS_PIN_SERVO(p)          ((p) >= 2 && (p) - 2 < MAX_SERVOS)
#define IS_PIN_I2C(p)            ((p) == 20 || (p) == 21) // 70 71
#define PIN_TO_DIGITAL(p)        (p)
#define PIN_TO_ANALOG(p)         ((p) - 54)
#define PIN_TO_PWM(p)            PIN_TO_DIGITAL(p)
#define PIN_TO_SERVO(p)          ((p) - 2)


// chipKIT
#elif defined(__PIC32MX__)
```

```
#define TOTAL_ANALOG_PINS       6
#define TOTAL_PINS              20 // 14 digital + 6 analog
#define VERSION_BLINK_PIN       13
#define IS_PIN_DIGITAL(p)       ((p) >= 2 && (p) <= 19)
#define IS_PIN_ANALOG(p)        ((p) >= 14 && (p) <= 19)
#define IS_PIN_PWM(p)           digitalPinHasPWM(p)
#define IS_PIN_SERVO(p)         (IS_PIN_DIGITAL(p) && (p) - 2 < MAX_SERVOS)
#define IS_PIN_I2C(p)           ((p) == 18 || (p) == 19)
#define PIN_TO_DIGITAL(p)       (p)
#define PIN_TO_ANALOG(p)        ((p) - 14)
#define PIN_TO_PWM(p)           PIN_TO_DIGITAL(p)
#define PIN_TO_SERVO(p)         ((p) - 2)

// anything else
#else
#error "Please edit Boards.h with a hardware abstraction for this board"
#endif


/*==============================================================================
 * readPort() - Read an 8 bit port
 *============================================================================*/

static inline unsigned char readPort(byte, byte) __attribute__((always_inline, unused));
static inline unsigned char readPort(byte port, byte bitmask)
{
#if defined(ARDUINO_PINOUT_OPTIMIZE)
        if (port == 0) return (PIND & 0xFC) & bitmask; // ignore Rx/Tx 0/1
        if (port == 1) return ((PINB & 0x3F) | ((PINC & 0x03) << 6)) & bitmask;
        if (port == 2) return ((PINC & 0x3C) >> 2) & bitmask;
        return 0;
#else
        unsigned char out=0, pin=port*8;
        if (IS_PIN_DIGITAL(pin+0) && (bitmask & 0x01) && digitalRead(PIN_TO_DIGITAL(pin+0)))
out |= 0x01;
        if (IS_PIN_DIGITAL(pin+1) && (bitmask & 0x02) && digitalRead(PIN_TO_DIGITAL(pin+1)))
out |= 0x02;
        if (IS_PIN_DIGITAL(pin+2) && (bitmask & 0x04) && digitalRead(PIN_TO_DIGITAL(pin+2)))
out |= 0x04;
        if (IS_PIN_DIGITAL(pin+3) && (bitmask & 0x08) && digitalRead(PIN_TO_DIGITAL(pin+3)))
out |= 0x08;
        if (IS_PIN_DIGITAL(pin+4) && (bitmask & 0x10) && digitalRead(PIN_TO_DIGITAL(pin+4)))
out |= 0x10;
        if (IS_PIN_DIGITAL(pin+5) && (bitmask & 0x20) && digitalRead(PIN_TO_DIGITAL(pin+5)))
out |= 0x20;
        if (IS_PIN_DIGITAL(pin+6) && (bitmask & 0x40) && digitalRead(PIN_TO_DIGITAL(pin+6)))
out |= 0x40;
        if (IS_PIN_DIGITAL(pin+7) && (bitmask & 0x80) && digitalRead(PIN_TO_DIGITAL(pin+7)))
out |= 0x80;
        return out;
#endif
}

/*==============================================================================
 * writePort() - Write an 8 bit port, only touch pins specified by a bitmask
 *============================================================================*/
```

```
static inline unsigned char writePort(byte, byte, byte) __attribute__((always_inline,
unused));
static inline unsigned char writePort(byte port, byte value, byte bitmask)
{
#if defined(ARDUINO_PINOUT_OPTIMIZE)
        if (port == 0) {
                bitmask = bitmask & 0xFC;  // do not touch Tx & Rx pins
                byte valD = value & bitmask;
                byte maskD = ~bitmask;
                cli();
                PORTD = (PORTD & maskD) | valD;
                sei();
        } else if (port == 1) {
                byte valB = (value & bitmask) & 0x3F;
                byte valC = (value & bitmask) >> 6;
                byte maskB = ~(bitmask & 0x3F);
                byte maskC = ~((bitmask & 0xC0) >> 6);
                cli();
                PORTB = (PORTB & maskB) | valB;
                PORTC = (PORTC & maskC) | valC;
                sei();
        } else if (port == 2) {
                bitmask = bitmask & 0x0F;
                byte valC = (value & bitmask) << 2;
                byte maskC = ~(bitmask << 2);
                cli();
                PORTC = (PORTC & maskC) | valC;
                sei();
        }
#else
        byte pin=port*8;
        if ((bitmask & 0x01)) digitalWrite(PIN_TO_DIGITAL(pin+0), (value & 0x01));
        if ((bitmask & 0x02)) digitalWrite(PIN_TO_DIGITAL(pin+1), (value & 0x02));
        if ((bitmask & 0x04)) digitalWrite(PIN_TO_DIGITAL(pin+2), (value & 0x04));
        if ((bitmask & 0x08)) digitalWrite(PIN_TO_DIGITAL(pin+3), (value & 0x08));
        if ((bitmask & 0x10)) digitalWrite(PIN_TO_DIGITAL(pin+4), (value & 0x10));
        if ((bitmask & 0x20)) digitalWrite(PIN_TO_DIGITAL(pin+5), (value & 0x20));
        if ((bitmask & 0x40)) digitalWrite(PIN_TO_DIGITAL(pin+6), (value & 0x40));
        if ((bitmask & 0x80)) digitalWrite(PIN_TO_DIGITAL(pin+7), (value & 0x80));
#endif
}
```

```
#ifndef TOTAL_PORTS
#define TOTAL_PORTS                ((TOTAL_PINS + 7) / 8)
#endif


#endif /* Firmata_Boards_h */
```

B.Reference

1.LCD tutorial
https://learn.adafruit.com/adafruit-arduino-lesson-11-lcd-displays-1/breadboard-layout
2. Count the revolution of the wheel using Hall Effect Sensor
https://sites.google.com/site/measuringstuff/more-sensor-examples
3. BLE Controller
http://redbearlab.com/getting-started-bleshield/