Name: Nick Olson Date: 5-1-2017

Lab 3 – Serial Communication and Hardware Sensors

Objectives

Part 1 – Implementing Serial

Part 2 – Working with Sensor Data

Background/Scenario

Serial communication is a commonly used communication method for a lot of devices. It may be old, but the ease of implementation and the low cost makes it still a great choice for current hardware projects.

This lab will cover the basics of implementing serial functionality in the Arduino environment using the Serial class and in .NET using the SerialPort class. You will set up a device that reads and writes values over a serial port to the computer and the computer will read the data and display it back.

You will also be working with some of the analog and digital sensors on the Circuit Playground and creating a small demonstration project that requests sensor data from the device, parses the received data, then displays the data on a GUI that is meaningful to the user.

Required Resources

- Visual Studio
- Arduino IDE
- Adafruit Circuit Playground
- MicroUSB Cable

Part 1: Implementing Serial

For this part of the lab you will write both the Arduino code for the device that reads and transmits sensor data and C# code for the user interface that receives and displays the sensor data back to the user. The objectives of this part is to:

a) Design a user interface that allows a user to select the Circuit Playground and connect to it.

b) Display some of the sensor data in real-time.

c) Provide a method for the user to update the RGB LEDs on the board.

Fill in the table below with some definitions.

Term	Definition
Baud Rate	The rate at which data is sent over serial. Measured in bits per second.
Data Bits	The bits of the packet that holds the data.
Parity	The 8th bit of data bits that tells if the preceding bits are even or odd. Simple way of error checking.
Stop Bits	The bits that signify the end of a packet
8N1	8 data bits, no parity and 1 stop bit

Step 1: The Arduino code will have these requirements:

a) Read the following sensor data: (See Appendix A for function prototypes)
   1. Accelerometer (X, Y, and Z)
   2. Temperature (Both Celsius and Fahrenheit)
   3. Light
   4. Sound
   5. Buttons (Left and right)
   6. Slide switch (+ and -)

b) Packetize and send the sensor data over serial.

c) When serial data is received:
   1. Parse the serial data.
   2. Identify which command to do.
   3. Execute the command.

d) There are two commands to implement:
   1. Turn on a specific RGB LED with a specified color value.
   2. Turn off all LEDs.

What are the default Serial parameters for Arduino if not specified?

Parameter	Default
Data bits	8
Parity	none
Stop bits	1

Step 2: The C# code will have these requirements:

a) Create a class in a class library that implements the interface ICircuitPlayground (See attached cs on Blackboard or Appendix D). The class can have any number of additional methods that either helps with interface functionality or provides additional functionality, but every interface method has to be implemented and working.

b) Create a user interface that uses the class interface to:

1. Display a list of devices.
2. Allow the user to select and connect to the device.
3. Display in real-time, the data received from the device.

   You have the option of displaying the data in any method you choose. It could either be text, graphical, or even both.
4. Specify an RGB value and pixel identifier.
5. Clear all pixels.

The class should hold the handle to the serial device and do all of the parsing and device handling. The user interface code should be completely separated from the device handling code. The goal is to not need to change the user interface code if any of the device code changes. For example, in next lab we will be working with the HID communication protocol and if the user interface and device code are properly separated, the user interface won't care how we connect to the device whether by serial or HID.

Part 2: Working with Sensor Data

In this part, you will add extra functionality to Part 1's code which makes use of some of the sensor data. You will also add an extra function which will crossfade a specified LED.

Cross fading differs from the simple fading in the previous week's lab where instead of going from one color to off, you go from one color to another color. For example, if you were to cross fade an LED from red to blue, you would decrement the red value while simultaneously incrementing the blue value.

Step 1: Implement the ILab3 interface.

Implement ILab3 on the same class that implemented ICircuitPlayground. Remember, C# allows you to implement any number of interfaces on a class. ILab3 includes four additional functions: IsBeingHeld, SetPixelCrossfade, SetPattern, and ExportSensorData.

ILab3 can be downloaded from Blackboard or alternatively see Appendix D.

Step 2: Implement "being held" functionality.

The IsBeingHeld function will determine whether the board is currently being held up or lying still on a surface. If the user is holding the board up, the function should return true, otherwise it will return false.

Create a section of your UI which will display to the user whether the board is currently being held or is lying on the table. This data should be real-time and updates automatically when the device is picked up or put down.

If you need a hint or just an idea of where to start, see Appendix B for some hints on different methods to implement it.

Step 3: Implement crossfading.

To implement the crossfading functionality, you are going to need to add an additional command to the Arduino code to parse.

Create a section in the UI to allow the user to specify a pixel number, a starting color, an ending color, a time in milliseconds, and whether to repeat or not.

If the user decided to repeat, the LEDs should crossfade from the start color to the end color and back to the start color and repeat indefinitely.

Also, update the SetPixel and ClearPixels commands to override the crossfade animation. For example, if an LED is currently crossfading, and the user decides to call SetPixel, the LED should stop crossfading and then display the color as indicated by the SetPixel function. Similarly, if the ClearPixels function is called, all crossfading should be stopped and all LEDs should be immediately turned off.

Step 4: Implement SetPattern.

This step is going to take the code you wrote in Lab 2 Part 4 and apply it here. SetPattern will activate the same three part pattern sequence from that lab. The function takes in an integer value and depending on the value, it will run differently.

Pattern ID numbers:

0. The full sequence from Lab 2 Part 4. All three patterns together matching exactly like how it was done for the lab.
1. Just the first pattern. Incrementally turning on the LEDs red then turning off. Incrementally green, then off. Incrementally blue, then off. Repeat.
2. Just the second pattern. Fading in cyan, then fading off.
3. Just the third pattern. Flashing the sequence of colors red, green, blue, cyan, yellow, magenta, and white.

Because these patterns are not compatible with any currently set LEDs, before activating them, you are to turn off all the LEDs and disable all crossfades before starting one.

If any pixel is set from the SetPixelColor or SetPixelCrossFade functions, you are to stop the pattern, clear all the pixels that are currently on from the pattern, then set the LED according to the function that was called.

If the ClearPixels function is called, the pattern is stopped and the pixels are cleared.

Step 5: Implement exporting data.

The exported data will be saved as a comma separated value file (or CSV) with a log of the data that was collected by the application.

Modify the UI to allow the user to export the data giving the user the option of specifying where to save the file.

Modify the application to store read sensor data while the application is running.

Save to the specified path.

See Appendix C for an example of what the CSV file contents look like.

Reflection

1. Search the internet for a few devices with serial interfaces that you can purchase. Give an example of one. What baud rate and other serial settings does it operate at?

2. Did you run into any unexpected problems with the lab?

I found this lab a little frustrating, mostly because there isn't a real good way to debug the code running the circuit playground. Sometimes it would be really difficult to tell why a command wasn't executing. Also not used to WPF and haven't used MVVM before.

Appendix A – CircuitPlayground additional functions

uint16_t CircuitPlayground.lightSensor();

uint16_t CircuitPlayground.soundSensor();

float CircuitPlayground.temperature();

float CircuitPlayground.temperatureF();

float CircuitPlayground.motionX();

float CircuitPlayground.motionY();

float CircuitPlayground.motionZ();

Appendix B – Implementing IsBeingHeld() Hints

There are many ways of identifying if the device is being held, each with different edge cases on when they would work and not. Here are a couple of methods that you can try:

1. The "snapshot" method. (Easy)

   Lay the device flat on the table and take a note of the sensor readouts. Store that value as lying flat. Start picking up the device and watch how the sensors respond. Identify a threshold value and if the threshold is surpassed with respect to the lying down value, count that the device is picked up.

   The downside to this method is that it will only work for that one particular surface. If for example, you set the device on a slightly tilted surface surpassing the threshold, the device will always report it is being held.

2. The heuristic sampling method. (Hard)

   Sample and store a set of reads from the sensor. As new data comes in, compare the new data to the stored data and identify a difference. Identify if the data differs from a difference threshold and if it has, treat that as the device is picked up.

   The downside to this method is that it takes a considerable amount of memory to store the previous samples depending on how many you decide to store. It will also take more calculations to iterate over the collected sample the more you chose to store.

Appendix C – Example CSV File Contents

This is an example csv file with three columns. Each column is separated by a comma and each row is separated by a new line. The first row denotes the column heading.

```
Number,Baud,UBRRn
1,2400,207
2,4800,103
3,9600,51
4,14400,34
5,19200,25
6,28800,16
7,38400,12
8,57600,8
9,76800,6
10,115200,3
```

Appendix D – Interface definitions

```
    public struct LedColor
    {
        public byte R;
        public byte G;
        public byte B;
    }
    public struct AccelData
    {
        public float X;
        public float Y;
        public float Z;
    }

    public interface ICircuitPlayground
    {
        /// <summary>
        /// Returns an array of device identifiers which should be passed into Open()
        /// to identify which device to open.
        /// </summary>
        /// <returns>A string array of device identifiers.</returns>
        string[] GetDevices();

        /// <summary>
        /// Returns true if the device is open.
        /// </summary>
        bool IsOpen { get; }

        /// <summary>
        /// Opens the device and allows for reading and writing.
        /// </summary>
        /// <param name="device">The device identifier.</param>
        /// <returns>Returns true if successfully opened. False if failed to
open.</returns>
        bool Open(string device);

        /// <summary>
        /// Closes the device and cleans up any open device handles.
        /// </summary>
        void Close();

        /// <summary>
        /// This event should fire whenever any data is received from the device.
        /// </summary>
        event EventHandler<EventArgs> DataReceived;

        /// <summary>
        /// Gets the current accel sensor reading.
        /// Returns a struct containing the X, Y, and Z values.
        /// </summary>
        AccelData ReadAccel { get; }

        /// <summary>
        /// Gets the current temperature in C.
        /// </summary>
        float ReadTempC { get; }
```

```
        /// <summary>
        /// Gets the current temperature in F.
        /// </summary>
        float ReadTempF { get; }

        /// <summary>
        /// Gets the current light sensor value.
        /// 10-bit value. From 0 to 1023.
        /// </summary>
        UInt16 ReadLight { get; }
        /// <summary>
        /// Gets the current microphone sensor value.
        /// 10-bit value. From 0 to 1023.
        /// </summary>
        UInt16 ReadSound { get; }

        /// <summary>
        /// Reads the left button state.
        /// True if pressed, false if not.
        /// </summary>
        bool ReadLeftButton { get; }
        /// <summary>
        /// Reads the right button state.
        /// True if pressed, false if not.
        /// </summary>
        bool ReadRightButton { get; }
        /// <summary>
        /// Reads the slide swtich state.
        /// True if in the left or (+) position, false if in the right (-) position.
        /// </summary>
        bool ReadSlideSwitch { get; }

        /// <summary>
        /// Sets the LED to a specified color.
        /// </summary>
        /// <param name="pixel">The pixel to set. 0-9</param>
        /// <param name="color">The color to set the pixel.</param>
        void SetPixelColor(int pixel, LedColor color);

        /// <summary>
        /// Turns off all the pixels.
        /// </summary>
        void ClearAllPixels();
       }
    public interface ILab3
    {
        /// <summary>
        /// Immediately sets the RGB LED to the starting color. Then fades the
        /// LED to the ending color over the specified time.
        /// </summary>
        /// <param name="pixel">The LED to set. 0-9</param>
        /// <param name="start">The starting color.</param>
        /// <param name="end">The ending color.</param>
        /// <param name="fadeTime">The time in milliseconds to apply the fade
over.</param>
        /// <param name="repeat">Repeats the fade if true. Defaults to false.</param>
```

```
        void SetPixelCrossFade(int pixel, LedColor start, LedColor end,
            int fadeTime, bool repeat = false);

        /// <summary>
        /// Determines if the Circuit Playground is being held up intead of flat on a
surface.
        /// </summary>
        /// <returns>Returns true if being held.</returns>
        bool IsBeingHeld();

        /// <summary>
        /// Sets the LEDs to the specified pattern. The pattern comes from Lab 2 Part 4.
        /// ID of 0 will be the full 3 part pattern.
        /// ID of 1 will be the first pattern.
        /// ID of 2 will be the second pattern.
        /// ID of 3 will be the third pattern.
        /// </summary>
        /// <param name="id">The ID of the pattern to change to.</param>
        void SetPattern(int id);

        /// <summary>
        /// Exports a log of the light sensor data to a csv file.
        /// </summary>
        /// <param name="filename">The filename to export as.</param>
        void ExportSensorData(string filename);
    }
```