ECE 120

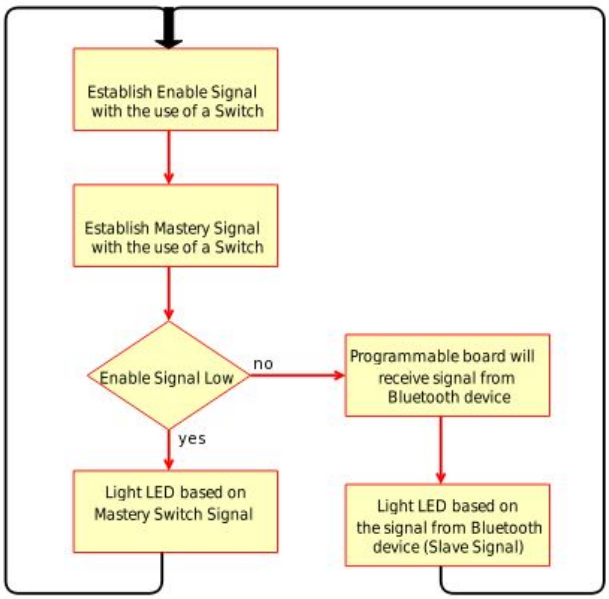
Chris Willenborg cwillen2

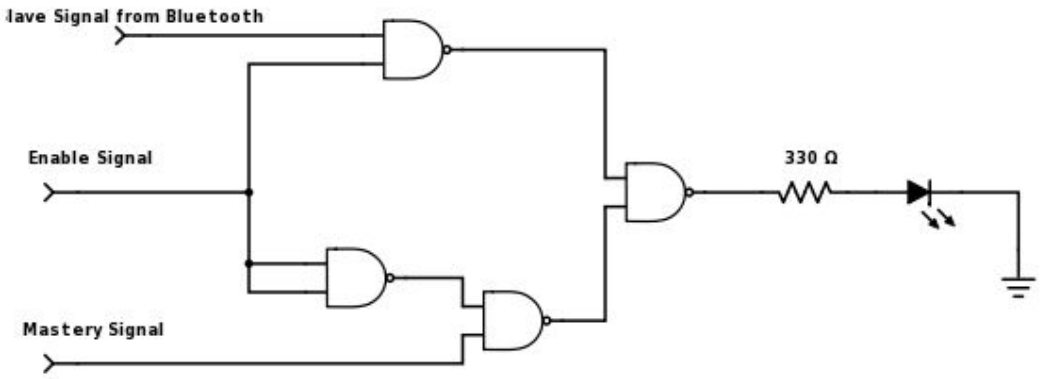
Xiaowei Zhang xzhan145

We were trying to implement a circuit that would make it possible for someone to turn on and off the lights with the use of an electronic device via bluetooth. Our major goal was to have a printed circuit board ready to demo on the demonstration day, but that did not quite come together as we had hoped it would. The bluetooth controlled light switch could have many different applications in the society or industry. The obvious use in society would be to make the lights in people's homes accessible via electronic devices with bluetooth accessibility. Our design could be used in industry directly or with modification to control other more specific need based aspects of industry.

The system that we designed features: logical components that allow a user complete manual control over the bluetooth signal, a Bluetooth sensor used to receive inputs, an app that was supported by the app store that was pre existing, and signal transmitters and receivers used to communicate with them. More explicitly we use a Simblee BLE Breakout board, two SPST switches, one LED, a few resistors, a quad 2 input NAND chip, and a voltage source to power our circuit.

We used one Single Pull Single Throw (SPST) switches to control our Enable Signal and another SPST to control our Mastery Signal. In our Design we wanted a system that would allow someone to exclusively control the state of the LED without interference from the Bluetooth signal (Slave Signal). We implemented this control with the use of Logical NAND gates. To implement the NAND gates to allow us to have total control we said that when the Enable signal is Low we will use only the signal from the Mastery switch, but when the Enable signal is High we will use the state defined by the Slave signal only. The state of the light is determined by the Slave Signal which gets interpreted in the Programmable board that we used by code.





Truth Table

E (Enable)	M (Mastery)	S (Slave)	O (Output)
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0

1	1	1	1

K-Map

E (Enable) / SIGNAL	M (Mastery) S (Slave) 00	01	11	10
0	0	0	1	1
1	0	1	1	0

Bluetooth part

Since we use a sparkfun adapted version of bluetooth low energy (BLE) 4.0 module, we use arduino for building software. As for the smartphone interface, sparkfun has its own application already named simblee and we basically use that.

Bluetooth specification:

1.power class 2+
2.BLE compatible
3.The profile we will use is Serial Port Profile(SPP)
4.**Confirmed by Apple tech Support staff that iphone support SPP

CODES

```
/*************************************************************************
 * LED_Control LED Control from smart phone Mike Hord @ SparkFun Electronics 26
 * Jan 2016 https://github.com/sparkfun/Simblee_Tutorials
 *
 * This example demonstrates the use of the SimbleeForMobile library to control
 * a pin on a Simblee module from a phone app. We'll show both a button and a
 * switch in this example.
 *
```

```
 * Resources: Please install the Simblee support files before attempting to use
 * this sketch; see
 * https://learn.sparkfun.com/tutorials/simblee-concepts#setting-up-arduino for
 * details.
 *
 * Development environment specifics: Arduino.cc IDE v1.6.7
 *
 * This code is beerware; if you see me (or any other SparkFun employee) at the
 * local, and you've found our code helpful, please buy us a round!
 * ********************************************************************************/

// To use the SimbleeForMobile library, you must include this file at the top
// of your sketch. **DO NOT** include the SimbleeBLE.h file, as it will cause
// the library to silently break.
#include <SimbleeForMobile.h>

const int led = 2; // The Simblee BOB (WRL-13632) has an LED on pin 2.
int ledState = LOW;

// Every draw command returns a uint8_t result which is the object id that was
// created. If you wish to change the object later, you'll need this value,
// and if you want to catch an event created by an object, you'll need it
// there, too. Make sure you create these id variables outside of any function,
// as you'll need to refer to them in many other functions.

uint8_t TBID;
uint8_t switchID;

void setup()
{
  pinMode(led, OUTPUT);
  digitalWrite(led, ledState);

  // advertisementData shows up in the app as a line under deviceName. Note
  // that the length of these two fields combined must be less than 16
  // characters!
  SimbleeForMobile.advertisementData = "Blink";
  SimbleeForMobile.deviceName = "WRL-13632";

  // txPowerLevel can be any multiple of 4 between -20 and +4, inclusive. The
  // default value is +4; at -20 range is only a few feet.
  SimbleeForMobile.txPowerLevel = -4;
```

```
  // This must be called *after* you've set up the variables above, as those
  // variables are only written during this function and changing them later
  // won't actually propagate the settings to the device.
  SimbleeForMobile.begin();
}

void loop()
{
  // This function must be called regularly to process UI events.
  SimbleeForMobile.process();
}

// ui() is a SimbleeForMobile specific function which handles the specification
// of the GUI on the mobile device the Simblee connects to.
void ui()
{
  // color_t is a special type which contains red, green, blue, and alpha
  // (transparency) information packed into a 32-bit value. The functions rgb()
  // and rgba() can be used to create a packed value.
  color_t darkgray = rgb(85,85,85);

  // These variable names are long...let's shorten them. They allow us to make
  // an interface that scales and scoots appropriately regardless of the screen
  // orientation or resolution.
  uint16_t wid = SimbleeForMobile.screenWidth;
  uint16_t hgt = SimbleeForMobile.screenHeight;

  // The beginScreen() function both sets the background color and serves as a
  // notification that the host should try to cache the UI functions which come
  // between this call and the subsequent endScreen() call.
  SimbleeForMobile.beginScreen(WHITE);



  // Create a button slightly more than halfway down the screen, 100 pixels
  // wide, in the middle of the screen. The last two parameters are optional;
  // see the tutorial for more information about choices for them. The BOX_TYPE
  // button has a bounding box which is roughly 38 pixels high by whatever the
  // third parameter defines as the width.
  TBID = SimbleeForMobile.drawText(
                        (wid/2) - 62,         // x location
                        (hgt/2) - 22,         // y location
                        "LED OFF",        // text shown on button
```

```
                    BLACK,              // color of button
                    36);            // type of button



  // Buttons, by default, produce only EVENT_PRESS type events. We want to also
  // do something when the user releases the button, so we need to invoke the
  // setEvents() function. Note that, even though EVENT_PRESS is default, we
  // need to include it in setEvents() to avoid accidentally disabling it.
  //SimbleeForMobile.setEvents(btnID, EVENT_PRESS | EVENT_RELEASE);

  // Create a switch above the button. Note the lack of a title option; if you
  // want to label a switch, you'll need to create a textBox object separately.
  // A switch's bounding box is roughly 50 by 30 pixels.
  switchID = SimbleeForMobile.drawSwitch(
                    (wid/2) - 25,         // x location
                    (hgt/2)+22,           // y location
                    BLACK);               // color (optional)


  SimbleeForMobile.endScreen();
}

// This function is called whenever a UI event occurs. Events are fairly easy
// to predict; for instance, touching a button produces a "PRESS_EVENT" event.
// UI elements have default event generation settings that match their expected
// behavior, so you'll only rarely have to change them.
void ui_event(event_t &event)
{
  // We created the btnID and switchID variables as globals, set them in the
  // ui() function, and we'll use them here.



  // If the event was a switch press, we want to toggle the ledState variable
  // and then write it to the pin.
  if (event.id == switchID)
  {
    if (ledState == HIGH) ledState = LOW,

   SimbleeForMobile.updateText(TBID,"LED OFF");
```

```
else ledState = HIGH,SimbleeForMobile.updateText(TBID, "LED ON");

   digitalWrite(led, ledState);
 } }
```

In our original plan we were going to implement this circuit and design printed circuit board for it simultaneously after we reached a certain stage in the completion of the hardware design. We discovered that it would not be a good idea to start designing our PCB until after we had successfully implemented all portions of our circuit on a breadboard. One of the largest problems that we had with this project was time. It took a few weeks for us to figure out which programmable bluetooth board would work for our needs and then it took approximately a month for us to get it. While we were waiting we designed the logical portion, so it was not a total loss of time. Another challenge that we had was that neither of us had any experience with Bluetooth or PCBs so there was a large learning curve for both portions, but I am glad to now have this experience in these areas.

In the future I hope that we can design the PCB and have it created for us to demo on EOH. I think that it will be much easier now than it was previously because the programmable Bluetooth board that we chose already has an Eagle design on the Sparkfun website for us to use. This will make it easier to get the spacing for the PCB because we can route the pre existing design to the locations that we want to use without much spacing issues.

FOR MORE REFERENCE:
https://docs.google.com/document/d/1igjXyXyJiLFourtGIShI42v7Ilou9brmnvKJY4AG5do/edit