Simple BLE Distance and Object Presence Detector
BL652

Application Note v1.0

INTRODUCTION

This application note describes how to use the Laird BL652 Bluetooth Low Energy module to interface a smartphone to a commercially-available parking sensor for the automobile aftermarket.

REQUIRED EQUIPMENT

The following hardware is required for this interface:

- Laird BL652 Development Kit
- A parking sensor with a beeping output
- NPN transistor (any small signal type)
- Two resistors (1k and 10k)

HARDWARE

Many commercial parking sensors signal a vehicle’s proximity to a road obstacle by issuing an increasingly frequent series of beeps as the vehicle approaches a stationary object. In our example, we’ve connected this beeper circuitry to the BL652 development board, and use the signal to interpret the proximity to the obstacle by counting the “beeps” in a 1.2 second interval.

From the parking sensor’s electronics, the 12-volt signal (which drives the beeper) is fed to the base of an NPN transistor through the 1k resistor (IN). This generates a “low” at the collector (OUT), which is used to trigger a GPIO on the Laird BL652 development kit. Connect OUT to D4 on the Arduino connector. 5V and GND are also taken from there.

Note: SIO15 (button2 on the development kit) is the same as D4 on the Arduino connector.

All three components (parking sensor, BL652 development kit, and the interface with 1xT and 2xR) must share the same ground (GND).

The schematic for this hardware setup, as well as hardware setup photos, are shown below.
Schematic

Figure 1: Arduino interfacing breadboard schematic

Hardware Setup Photos

Figure 2: Full setup with parking sensor board, parking sensor, BL652 Development Board and Arduino-interfacing breadboard with NPN transistor and resistors

Figure 3: Wiring from the parking sensor board, where the 5V out, the 12V beeper signal and GND are connected to the Arduino-interfacing breadboard

Figure 4: The Arduino-interfacing breadboard, with the NPN transistor and two resistors wired in. The pin at far left will be mounted to D4 on the BL652 Development Board.

Figure 5: The full setup, with the Arduino-interfacing breadboard mounted onto the BL652 Development Board
SOFTWARE

The software is an application running on the smartBASIC engine within the BL652 (it is included in Appendix: BLE Distance Meter Application). Every 1.2 seconds, it counts the signals coming from the parking sensor. From this count, the following five range categories were formulated.

<table>
<thead>
<tr>
<th>Signal Count</th>
<th>Interpretation by smartBASIC application</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No signal/beep – Object is far away</td>
</tr>
<tr>
<td>1</td>
<td>One or two signals/beeps – Object is detected</td>
</tr>
<tr>
<td>2</td>
<td>Three or four signals/beeps – Object is closer</td>
</tr>
<tr>
<td>3</td>
<td>Five or six signals/beeps – Object is closer than it was at three/four beeps</td>
</tr>
<tr>
<td>4</td>
<td>Seven or more signals/beeps – Object is really close</td>
</tr>
<tr>
<td>5</td>
<td>Constant signal/beep – Object is too close (about to make contact with object)</td>
</tr>
</tbody>
</table>

Once the count is completed, it is then advertised as a BLE advert which can be detected by a smartphone application (such as BLExp or nRFconnect, as shown in Figure 6). The numbers are also stored in the BL652 module’s GATT table and the advert is made connectable. When connected, each can be read as a characteristic, as shown in Figure 7.

Note: Due to the fact that the signal from the parking meter shares the GPIO with button2, that button can be used to simulate the beeps.
POTENTIAL APPLICATIONS

Some potential applications include the following:

- A sensor on the back wall of a garage to prevent family members from colliding with it
- Simple, affordable retrofit to commercial parking sensors
- Burglar alarm – If the object is removed, the smartphone alarm goes off
- General ‘presence of object’ detection

For ordering information about the BL652 Development Board, visit the BL652 page at lairdtech.com: www.lairdtech.com/products/bl652-ble-module

APPENDIX: BLE DISTANCE METER APPLICATION

// Simple BLE Distance and Object Presence Detector

// Declare variables
DIM rc            //declare an integer variable
    //called rc, rc stands for result code
DIM count        //declare counter variable for
    //counting beeps
DIM advRpt$ : advRpt$ = "" //the content of the Advert Report
DIM scnRpt$ : scnRpt$ = "" //the content of the Scan Report
DIM dismsg$ : dismsg$ = ""  
DIM disnum : disnum = 0
DIM Addr$        //declare an empty peer address for advertising
DIM baseUuid$    
DIM hBaseUuid
DIM hGPIOSvcUuid //service UUID handle
DIM hGPIOSvc
dIM hInputs      //Button/LED Service::Total Inputs
    //Characteristic handle

#define BASE_UUID   "\33\47\00\00\FB\94\11\E2\A8\E4\F2\3C\91\AE\C0\5E"
#define GPIO_SVC_UUID 0xAAA0
#define INPUTS_CHAR_UUID 0xAAA1

// User Functions
FUNCTION handlerTimer0()  //this handler function is called when Timer 0 expires
    if count == 0 & gpioread(15) == 1 then
        disnum = 0
    elseif count == 1 || count == 2 then
        disnum = 1
    elseif count == 3 || count == 4 then
        disnum = 2
    elseif count == 5 || count == 6 then
        disnum = 3
    elseif count >= 7 then
        disnum = 4
    endif
    if gpioread(15) == 0 then
        disnum = 5
    endif
    rc = BleEncode8(dismsg$,disnum,0)
    print dismsg$; ",n"
rc = BleAdvRptinit(advRpt$, 0, 0, 5)
rc = BleAdvRptAppendAD(advRpt$, 0xff, dismsg$)
rc = BleAdvRptsCommit(advRpt$, scnRpt$)
rc = BleAdvertStart(0, Addr$, 25, 1200, 0) // starts advertising
rc = BleCharValueWrite(hInputs, dismsg$) // put distance into characteristic

count = 0
TIMERSTART(0, 1200, 0)

ENDFUNC 1

FUNCTION button1pressed()
    // a pressed button pulls GPIO15 low
    count = count + 1
ENDFUNC 1

SUB CreateServices()
    // register base UUID handle
    seUuid$=BASE_UUID
    hBseUuid=BleHandleUuid128(bseUuid$)
    // create service handles
    hGpioSvcUuid=BleHandleUuidSibling(hBseUuid, GPIO_SVC_UUID) // Button/LED Service
    dim dta$ : dta$="\00"
    dim rc2
    // create services
    rc=BleServiceNew(1, hGpioSvcUuid, hGpioSvc)
        // create characteristics
        rc=BleCharNew(0x02, BleHandleUuidSibling(hBseUuid, (INPUTS_CHAR_UUID)),
                        BleAttrMetaDataEx(1,0,1,1,rc2),0,0)
        rc=BleCharCommit(hGpioSvc, dta$, hInputs)
    rc=BleServiceCommit(hGpioSvc)
ENDSUB

SUB InitialiseServices()
    // Initialise Button/LED Service::Total
    // Inputs Characteristic
    rc=BleCharValueWrite(hInputs, dismsg$)
ENDSUB

SUB GpioInit()
    rc = gpiosetfunc(11,1,2) // sets sio11 (Button 1) as a digital in
        // with a weak pull up resistor
    rc = gpiosetfunc(15,1,2) // sets sio15 (Button 2) as a digital in
        // with a weak pull up resistor
    rc = gpiobindevent(0,11,1) // binds a gpio transition to low to an
        // event. sio11 (button 1)
    rc = gpiobindevent(1,15,1) // binds a gpio transition to low to an
        // event. sio15 (button 2)
    rc = gpiosetfunc(17,2,0) // sets sio17 (LED1) as a digital out
    rc = gpiosetfunc(19,2,0) // sets sio19 (LED2) as a digital out
ENDSUB

// Main body executed on program start
CreateServices()
InitialiseServices()

count = 0
GpioInit()
```c
TIMERSTART (0,1200,0)
ONEVENT EVTMR0 call handlerTimer0 //when timer 0 expires run the timer0
                     //handler function
ONEVENT EVGPIOCHAN1 call button1pressed //when button 2 is pressed call
                     //button1pressed() to count beeps
                     //within one period
WAITEVENT //wait in low power for an event
            //to happen
PRINT "error: program has ended" //prints to the UART, this line
            //should never happen
```

**Revision History**

<table>
<thead>
<tr>
<th>Version</th>
<th>Date</th>
<th>Notes</th>
<th>Approver</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0</td>
<td>16 Nov 2016</td>
<td>Initial Release</td>
<td>Jonathan Kaye</td>
</tr>
</tbody>
</table>