S-TRACK SENSE

Bluetooth Sensor

Documentation

VERSION	Date	History	Writer
PA1	30/11/18	Initial push	Mr
PA2	2/12/18	Add comments	Ltn
PA3	3/12/2018	Comments applied, 5.6.1 removed, 5.6.2 now 5.6.1, 5.7 added	MR
PA4	4/1/2019	Add Pi Script	Ltn
PA5	31/1/2019	Review and document approval	AvA
PA6	05/3/2016	Updated branding	AvA

Contents

1 INTRODUCTION

The S-track Sense tag is a Bluetooth Low Energy device intended for use as a sensor node for remote monitoring and reporting of environmental conditions.

1.1 Integrated sensors and peripherals:

- Temperature
- Humidity
- Ambient Light Level
- Acceleration
- Integrated NFC tag
- 1MB non-volatile memory
- Red led
- 32768Hz crystal for maximum timing accuracy
- TTL interface for programming and debug
- IP67 environmental protection

1.2 Default application features:

- Configurable high and low level alarm settings for all sensors
- Configurable reporting interval
- Automatic time synchronisation with Client (data receiver)
- Timestamped data transmissions
- Ultra-low power consumption for long battery life.

2 SYSTEM DESCRIPTION

All peripheral devices have been chosen for their high performance and low power operation over a wide range of operating voltages. Sensors are connected to the Bluetooth module using the inbuilt i2c interface.

2.1 Bluetooth

The module at the core of the S-track Sense is a Laird BL652, based on the Nordic nRF52832 chipset, with Laird SmartBasic or Nordic SDK for application programming. Full information may be obtained from https://www.lairdtech.com/products/bl652-ble-module

2.2 Accelerometer

The accelerometer is a ST Microelectronics LIS2DE12 featuring configurable full scale ranges of ±2g, ±4g, ±8g and ±16g. Detection of orientation, tap, free-fall and motion is possible. Indication is by configurable interrupt.

2.3 Temperature and Humidity

The detector is a HTU21D featuring typical accuracy of ±2% Rh with 12bit resolution and ±0.3˚C with 14bit resolution. A special membrane filter built into the case allows maintenance of an IP65 rating.

2.4 Ambient Light level

The light sensor is a TAOS TLS45315 featuring a very high dynamic range of 2 lux through to 220k lux, allowing its use in low light levels through to full sunlight.

2.5 Flash memory

The flash memory is an 8Mbit Macronix MX25R8035F device connected to a special SPI interface on the BL652 to allow easy memory access.

2.6 Battery

The S-track Sense is designed to be used with a CR2450 Lithium battery which has a capacity of 600mAh to ensure long life. The battery is accessed through a removable cover with 'O' ring sealing to preserve an IP67 rating.

3 PRINCIPLES OF OPERATION

Bluetooth operates on the principle that two devices, one configured as a Server and the other as a Client, may communicate and exchange data.

Bluetooth Org has defined that a device, which collects data from sensors is normally configured as a Server and the receiving device takes the Client role. This tends to cause confusion to those more used to the normal client/server environment as in this case we can have many Servers communicating with a single Client.

In the case of the S-track Sense, it can be programmed to take either role. As a Client it will receive data from tags and output that data on the serial TTL interface which can be directly connected to an MCU or to a PC application via a TTL to USB connection.

The normal sequence of operation is that the Tag (Server) is configured to advertise its presence at regular intervals as being discoverable. The Client is configured to scan for advertising messages. When the Client detects an advertising message from a Tag it can respond to it and exchange data on the capabilities of the Tag. Filtering may be implemented at this point to only process advertising data from a specific range of tags (a whitelist) or Tags with a specific device name. The whitelist is of restricted use as the number of permissible entries is normally quite low. If the Client sees a service, it is interested in it will request connection to that service which will normally cause the Tag to send specific data about that service to the Client or allow the Client to send specific data to the Tag. On completion of the operation the Client breaks the connection to the tag and is free to make new connections. If configured correctly, a Client can make up to 8 simultaneous connections, but this requires more complicated message handling.

Documentation is available at https://www.lairdtech.com/products/bl652-ble-module on the use of SmartBasic to configure the modules for various types of operation.

4 SAMPLE APPLICATIONS

COPYRIGHT NOTICE

All sample applications are the property of TT Electronics and may not be used for any commercial purpose. They are supplied on an 'as-is' basis with no warranties as to suitability or function.

Sample applications may be obtained on www.ttelectronics.com.

4.1 Loading and Running Applications

Smartbasic applications are compiled, and loaded, using a Laird supplied terminal program called UwTerminalX. This may be obtained from

https://github.com/LairdCP/UwTerminalX/releases

A User Guide can be found on the module webpage.

4.2 Sample_tag.sb

This is the application, written in SmartBasic, which configures the tag to take the Server role and handles data reading from sensors and transmission to a Client.

Setting the variable debug to 1 allows debug messages to be monitored, and displayed with the UwTerminalX software, using the TTL interface

It is highly event driven, as with all Bluetooth applications. Event driven applications allow the Bluetooth stack in the module to detect that no processing is taking place and automatically switch to a low power state while waiting for the next event. The module MCU current may drop below 2uA during this phase instead of the more normal 3mA while processing.

SmartBasic does not allow direct access to the MCU RTC so it is necessary to use timers for all timing functions. The core timer is a 1 second timer which is used as a timestamp. For the timestamp to be meaningful its value must be synchronised to the actual time, which is obtained from the Client.

When the application starts the module will advertise its presence every 5 seconds until discovered by a Client. When discovered the Tag sends its current timestamp to the Client. As this is just the count of seconds elapsed since start up the Client detects this as being an invalid timestamp and sends a series of messages to the Tag. These messages are, the current timestamp, reporting cycle time, data reporting window within that cycle and configuration data for the tag sensors. These values are compared with those already stored on the tag in non-volatile memory and the stored values are updated if necessary. The tag then enters a low power state where only the accelerometer is active.

Every 10 seconds the module wakes up and the sensors are read. If any sensor value is outside the programmed limits, then a timestamped alarm message is created and stored in flash memory.

The accelerometer is always active and if the programmed acceleration threshold is exceeded then its mode changes and 32 samples of the data on all 3 axes are collected and analysed. The peak vector sum of the samples is used to create a timestamped alarm message which is stored in flash memory.

The Tag may be configured to supply data from only a subset of the sensors.

When the Tag detects that the reporting window is open then all sensors are read, and it starts to advertise its presence. The Client will connect and request that the Tag reports all data that it has been programmed to provide, followed by any alarm data collected since the last connection. The Client will check the current timestamp of the received data and if this deviates from the Client timestamp be 2, or more, seconds then the Client will send a new timestamp to the tag. If Tag configuration settings have been updated on the Client then these new settings will also be sent to the Tag.

When communication is complete the Tag will be disconnected from the Client and enter a low power mode.

If the tag fails to communicate with a Client on 10 consecutive attempts, it will switch back to advertising its presence every 5 seconds. This provides a mechanism to re-establish connection if time synchronisation between Tag and Client has been lost for any reason

4.3 Sample_client.sb

This is the application, written in SmartBasic, which configures the Client to take and handles data transfers to, and from, Tags.

Setting the variable debug to 1 allows debug messages to be monitored, and displayed with a PC application, using the TTL interface

When the application starts it sends a message to a connected MCU, or PC application, requesting the current timestamp. The Client will then wait until the timestamp is a multiple of the report cycle time before scanning for Tag adverts. This maximises the chance that an already running tag will advertise during the reporting window.

The Client will connect to Tags discovered during the reporting window, receive sensor data, and send updated configuration data to the tag.

Received sensor data is stored in flash memory until completion of the report window. The Clint then issues an indication to a connected MCU, or PC Application that data is available, and it may be requested during a 5 second window. If not requested, the data will be retained.

4.4 PC Application

A PC application may be used, instead of a MCU, to communicate with the Client.

The sample application, written in Visual Studio 2013 C#, allows configuration of Tag sensor limits and selection of sensor data to be reported.

The application can also provide the timestamp needed at start up. To make use of this facility the Client application should be loaded but not started. This is the normal state at power up on a Client where autorun is not operational. When the Client is started using the ‘Start Client’ button the application will provide the necessary timestamp.

The PC application automatically responds to the data available indication by requesting the data from the Client.

4.5 Messages sent to the Client by MCU or PC

4.5.1 THRESH

Message Format	<ID>, THRESH, <threshold>, <duration>
Function	Causes the tags to check accelerometer values and, if a value exceeds the <thresh> value for the time <duration>, to trigger collection of accelerometer alarm event data
Example	000000000000,THRESH,700,15

4.5.2 TMAXMIN

Message Format	<ID>, TMAXMIN, <max>, <min>
Function	Causes the tags to check temperature values and, if a value is outside the range <max> to <min>, to start issuing an alarm message. <max> and <min> may be negative and are in increments of 1/100 ˚C
Example	000000000000,TMAXMIN,3500,2000

4.5.3 HMAXMIN

Message Format	<ID>, HMAXMIN, <max>, <min>
Function	Causes the tags to check humidity values and, if a value is outside the range <max> to <min>, to start issuing an alarm message. <max> and <min> may be negative and are in increments of 1/100 %Rh
Example	000000000000,HMAXMIN,8000,1000

4.5.4 LMAXMIN

Message Format	<ID>, LMAXMIN, <max>, <min>
Function	Causes the tags to check light values and, if a value is outside the range <max> to <min>, to start issuing an alarm message. <max> and <min> are in increments of 10 lux
Example	000000000000,LMAXMIN,2000,0

4.5.5 DONOFF

Message Format	<ID>, DONOFF, battery>, <temperature>, <humidity>, <light>, <acceleration>
Function	If <parameter> = 1, the tag will send data at report intervals. If 0 no data will be sent. Alarm messages will always be sent
Example	000000000000,DONOFF,1,1,1,0,0

4.5.6 TIME

Message Format	<ID>, TIME, <sec count>, <report cycle time>, <wake window>
Function	Values are in seconds. This command causes the Client to set its timestamp counter to the value received and sets default values for report cycle time and wake window. When a tag communicates, the timestamp of the message will be checked against the current Client value and if it differs by more than 2 seconds will cause the Client to issue a command to the tag to update its count. Any tag communicating with the client will initially have a timestamp below 10000000. This causes the Client to send a message to the tag to update its timestamp, report cycle time and report window. This synchronises Client and tag timestamps and reporting.
Example	000000000000,TIME,1543582000,30,10

4.5.7 SLEEP

Message Format	SLEEP
Function	Causes the Client BL652 to enter a deep sleep state. Once activated, the client can only be woken by cycling the power

4.5.8 SETTINGS

Message Format	SETTINGS
Function	Retrieves the stored configuration settings from the Client
Typical Return	All return messages are in the order below and are preceded by, and terminated with, <CR><LF> THRESH,700,15 TMAXMIN,3000,500 LMAXMIN,10000,0 HMAXMIN,10000,100 DONOFF,1,1,0,0,1

4.5.9 REQ,DATA

Message Format	REQ, DATA
Function	Retrieves stored messages from the Client
Typical Return	All return messages are preceded by, and terminated with, <CR><LF> and start with a header followed by the ID of the tag. The TIME message is followed by the timestamp. All other messages are followed by a 3 character descriptor for the parameter (4 character starting with 'A' if it is an alarm message), followed by the timestamp followed by the data. Every communication by a Client with a tag will produce a TIME message. Other messages depend on whether the parameter has

	been set to report. A final data message ENDDATA, <Timestamp> is received by the MCU to indicate that no further data will be sent in the session. If no data is available then a message NODATA, <Timestamp> will be received Example data message sequence TIME,DB2FC598F6C7,1543398511 DATA,DB2FC598F6C7,BAT,1543398511,3185 DATA,DB2FC598F6C7,TMP,1543398511,2109 DATA,DB2FC598F6C7,HUM,1543398511,4745 ENDDATA,1543398520

4.6 Messages sent by Client to MCU or PC

4.6.1 REQ,TIME

Function	Requests the MCU or PC to send a current timestamp, report cycle time and report interval
Typical Return	TIME, 1543582000,30,10

4.7 GATT Database

The Tag creates a GATT database with entries for the various descriptors and services offered.

Two Primary services are defined (see GattIninit):

1. A service which configures a characteristic allow the tag to notify data values. It allows a single 'notify' command to initiate the transfer of all activated parameters. An alternative would be to create separate characteristics for each parameter.
2. A service which configures a characteristic to accept data strings and is used to allow the tag to receive configuration settings from the Client.

5 Pi Client

The following paragraph illustrates a simple client which can be used with the Bluetooth tag. It has been written in Python language. One need to install the module bluepy in order to tun it.

5.1 Functions

The source code depicted in the paragraph 5.2 can be copied in a file which will need to be made executable.

1. Install Bluepy (https://github.com/IanHarvey/bluepy)
2. Launch the **Blescan** command or use a BLE scanner in order to find out the Mac address of the Tag
3. Launch the copied file with one of the different possibilities

3.1 -**m** AA:BB:CC:DD:EE:FF
3.2 -**p** xx : Define the **periodicity** of the report
3.3 -**w** xx : Define the **wake** up time for the report session
3.4 **-l x** : x is equal to 0 or 1 if one want to report or not the **light** value
3.5 **-t x** : x is equal to 0 or 1 if one want to report or not the **temperature** value
3.6 **-a x** :x is equal to 0 or 1 if one want to report or not the **accelerometer** value
3.7 **-H x** : x is equal to 0 or 1 if one want to report or not the **humidity** value

5.2 Output

The main function will print out the wanted parameter every period with a green font and the alarm event such as an accelerometer event with be displayed in yellow.

```
--------------------------------------------------------------------------------

Connection number 808: Error = 0 :timeResync = 0: HCI = 0

--------------------------------------------------------------------------------

Waiting for next period = 100

2019-01-03 02:18:27

currentTime=1546478307

Going to sleep for 93 wait for next sync

Connecting...to F8:C1:7B:CA:5A:F4

found Time

delta in time is 3

Something went wrong

disconecting

[['TIME', '1546478401'], ['BAT', '1546478401', '2984'], ['ACCX1546478401', '-16'],
['ACCY1546478401', '-16'], ['ACCZ1546478401', '0'], ['TMP', '1546478401', '2284'],
['AAX154647833796'], ['AAY15464783372032'], ['AAZ15464783371648'], ['AAX1546478338720'],
['AAY1546478338-336'], ['AAZ1546478338-2048'], ['AAX1546478338-1680'], ['AAY1546478338-
1472'], ['AAZ15464783380'], ['AAX15464783401056'], ['AAY1546478340-2048'],
['AAZ15464783400']]

Battery was 2.984 volt on 2019-01-03 02:20:01

Accel X axis was -0.16 g on 2019-01-03 02:20:01

Accel Y axis was -0.16 g on 2019-01-03 02:20:01

Accel Z axis was 0.0 g on 2019-01-03 02:20:01

Temperature was 22.84 Cel on 2019-01-03 02:20:01

Accel Alarm X: was 0.96 g on 2019-01-03 02:18:57

Accel Alarm Y: was 20.32 g on 2019-01-03 02:18:57

Accel Alarm Z: was 16.48 g on 2019-01-03 02:18:57

Accel Alarm X: was 7.2 g on 2019-01-03 02:18:58

Accel Alarm Y: was -3.36 g on 2019-01-03 02:18:58

Accel Alarm Z: was -20.48 g on 2019-01-03 02:18:58

Accel Alarm X: was -16.8 g on 2019-01-03 02:18:58

Accel Alarm Y: was -14.72 g on 2019-01-03 02:18:58

Accel Alarm Z: was 0.0 g on 2019-01-03 02:18:58

Accel Alarm X: was 10.56 g on 2019-01-03 02:19:00

Accel Alarm Y: was -20.48 g on 2019-01-03 02:19:00

Accel Alarm Z: was 0.0 g on 2019-01-03 02:19:00

--------------------------------------------------------------------------------
```

```
Connection number 809: Error = 0 :timeResync = 0: HCI = 0

--------------------------------------------------------------------------------

Waiting for next period = 100

2019-01-03 02:20:08

currentTime=1546478408

Going to sleep for 92 wait for next sync

Connecting...to F8:C1:7B:CA:5A:F4

found Time

delta in time is 3

Something went wrong
```

This simple example did not implement threshold changes. The reader needs to refer to THRESH, TMAXMIN, LMAXMIN, HMAXMIN to affect those parameters.

6 Source Code

```
#!/usr/bin/python
from bluepy import btle
import  time,datetime
import binascii
import sys
import os
import argparse

class bcolors:
    HEADER = '\033[95m'
    OKBLUE = '\033[94m'
    OKGREEN = '\033[92m'
    WARNING = '\033[93m'
    FAIL = '\033[91m'
    ENDC = '\033[0m'
    BOLD = '\033[1m'
    UNDERLINE = '\033[4m'

def waitForNext(period):
                #epoch time
                print "Waiting for next period = " +str(period)
                currentTime = int(time.time())
                print datetime.datetime.fromtimestamp(currentTime)
                print "currentTime=" + str(currentTime)
                timeToSleep = period - currentTime % period
                print "Going to sleep for " + str(timeToSleep) + " wait for next
sync"
                time.sleep(timeToSleep)

def showData(d):
                s = d.getServices()
                c = d.getCharacteristics()

                #Print the name of the tag
                c = d.readCharacteristic(3)
                print c
```

```
                print "Services..."
                for svc in dev.services:
                        print str(svc)

                chList = d.getCharacteristics()
                print "Handle   UUID                                 Properties"
                print "-------------------------------------------------------"
                for ch in chList:
                        print ("  0x"+ format(ch.getHandle(),'02X')  +"
"+str(ch.uuid) +" " + ch.propertiesToString())

def printData(d):
        print d
        for l in d:
                color = bcolors.OKGREEN
                div =100
                if len(l)==3:
                        try:
                                date = datetime.datetime.fromtimestamp(int(l[1]))
                        except:
                                print "*"*80
                                print "timestamp invalid"
                                date = datetime.datetime.fromtimestamp(0)
                        if l[0]=="BAT":
                                typeSens="Battery"
                                unit="volt"
                                div = 1000
                        elif l[0]=="TMP":
                                typeSens="Temperature"
                                unit="Cel"
                        elif l[0]=="LHT":
                                typeSens="Light"
                                unit=""
                        elif l[0]=="HUM":
                                typeSens="Humidity"
                                unit="%"
                        elif l[0]=="ALMT":
                                color = bcolors.HEADER
                                typeSens="Alarm Temperature"
                                unit="%"
                        elif l[0]=="ALMH":
                                color = bcolors.HEADER
                                typeSens="Alarm Humidity"
                                unit="%"
                        elif l[0]=="ALML":
                                color = bcolors.HEADER
                                typeSens="Alarm Light"
                                unit="%"
                        else:
                                typeSens="unknown"
                                unit=""
                        data =l[2]
                elif len(l)==1:
                        try:
                                date =
datetime.datetime.fromtimestamp(int(l[0][3:13]))
                        except:
```

```
                                print "*"*80
                                print "timestamp invalid"
                                date = datetime.datetime.fromtimestamp(0)
                        a=l[0][:3]
                        data = l[0][13:]
                        unit ="g"
                        color = bcolors.WARNING
                        if a=="AAX":
                                typeSens ="Accel Alarm X:"
                        elif a=="AAY":
                                typeSens ="Accel Alarm Y:"
                        elif a=="AAZ":
                                typeSens ="Accel Alarm Z:"
                        else:
                                typeSens ="Accel Alarm : unknown"
                else:
                        #Accelerometer
                        if l[0]=="TIME":
                                continue
                        a=l[0][:4]
                        try:
                                date =
datetime.datetime.fromtimestamp(int(l[0][4:]))
                        except:
                                print "*"*80
                                print "timestamp invalid"
                                date = datetime.datetime.fromtimestamp(0)
                        unit = "g"
                        if a=="ACCX":
                                typeSens="Accel X axis"
                        elif a=="ACCY":
                                typeSens="Accel Y axis"
                        elif a=="ACCZ":
                                typeSens="Accel Z axis"
                        else:
                                typeSens="Unknown"
                        data =l[1]
                try:
                        print color+typeSens + " was "+ str(float(data)/div) + "
" + unit + " on " + str(date) + bcolors.ENDC
                except:
                        color = bcolors.FAIL
                        print color+typeSens + " error : data=" + str(data) +
";div =" +str(div) + bcolors.ENDC


def donoff(p):
        if p.battery:
                b="1"
        else:
                b="0"
        if p.temperature:
                t="1"
        else:
                t="0"
        if p.humidity:
                h="1"
        else:
```

```
                h="0"
        if p.light:
                l="1"
        else:
                l="0"
        if p.accel:
                a="1"
        else:
                a="0"
        return "DONOFF,"+b+","+t+","+h+","+l+","+a

def repint(p):
        return "REPINT,"+str(p.period)+","+str(p.wake)

def startCom(args):

        config = True
        Mac = sys.argv[1]
        data =[]
        os.system("clear")
        period = args.period
        counter = 1
        counterError = 0
        counterTime = 0
        counterHci = 0
        while (1):
                print "-"*80
                print "Connection number " + str(counter) + ": Error = " + str
(counterError) + " :timeResync = " + str(counterTime) + \
                ": HCI = " + str(counterHci)
                print "-"*80
                waitForNext(period)

                print "Connecting...to " + args.mac
                try:
                        dev = btle.Peripheral(args.mac,"random")
                except:
                        print "Could not connect"
                        counterError = counterError + 1
                        if counterError>=10:
                                os.system("sudo hciconfig hci0 down")
                                os.system("sudo hciconfig hci0 up")
                                counterHci = counterHci + 1
                        continue

                if config == True:

                        currentTime = int(time.time())
                        print datetime.datetime.fromtimestamp(currentTime)

                        #Set time
                        tim = "TIME,"+str(currentTime)
                        dev.writeCharacteristic(0x14,tim)
                        time.sleep(0.2)
                        dev.writeCharacteristic(0x14,donoff(args))
                        time.sleep(0.2)
                        dev.writeCharacteristic(0x14,repint(args))
                        time.sleep(0.2)
```

```
                config = False

                resp = dev.writeCharacteristic(0x11,"\x01")
                #r=dev.readCharacteristic(0x10)

                while 1:
                        currentTime = int(time.time())
                        if (currentTime % period<9):
                                if not dev.waitForNotifications(2):
                                        print "Something went wrong"
                                        break
                                r=dev.readCharacteristic(0x10).split(",")
                                data.append(r)
                                if r[0]=="TIME":
                                        print "found Time"
                                        currentTime = int(time.time())
                                        delta = currentTime-int(r[1])
                                        print "delta in time is " + str(delta)
                                        if delta>3:
                                                config=True
                                                counterTime = counterTime + 1
                                                print "T"*80
                                                print "Reconfig wanted"
                                                print "T"*80
                        else:
                                print "10sec exceedied"
                                break

                print "disconecting"
                dev.disconnect()
                printData(data)
                data=[]
                print("\n\n\n\n")
                counter=counter+1


if __name__ == "__main__":
        parser = argparse.ArgumentParser()
        parser.add_argument("-p","--period",help="define the periodicity of the
tag",default=30,type=int,nargs='?')
        parser.add_argument("-w","--wake",help="define the wakeup
time",default=10,type =int,nargs='?')
        parser.add_argument("-b","--battery",help="define if battery is
displayed",default=1,type =int,nargs='?')
        parser.add_argument("-t","--temperature",help="define if temperature is
displayed",default=1,type =int,nargs='?')
        parser.add_argument("-l","--light",help="define if light is
displayed",default=0,type =bool,nargs='?')
        parser.add_argument("-a","--accel",help="define if accelerometer is
displayed",default=1,type =int,nargs='?')
        parser.add_argument("-H","--humidity",help="define if humidity is
displayed",default=0,type =int,nargs='?')
        parser.add_argument("-m","--mac",help="give the Mac address of the
BtSense",nargs='?')
        args=parser.parse_args()

        startCom(args)
```