Manual: Bluetooth Low Energy per a radino

Enginyeria de Sistemes — iTIC

Albert Babí Oller

8 de maig de 2015

Índex

1 Eines necessàries

Per a treballar amb el protocol de Bluetooth Low Energy (BLE) utilitzant la plataforma `radino` i un ordinador portàtil es necessita la configuració descrita a continuació.

1.1 BlueZ

`BlueZ` és l'stack de Bluetooth per a Linux; les dues eines bàsiques que ofereix són:

- `hcitool`: Per a gestionar connexions (cercar dispositius, informació del Bluetooth de l'ordinador, etc.)
- `gatttool`: Per a la comunicació amb un dispositiu (establiment de la connexió, intercanvi de dades, etc.)

Encara que `BlueZ` vingui amb Ubuntu per defecte es recomana instal·lar-ne una versió posterior [eLi15]. També s'ha de tenir en compte que l'ordinador sigui compatible amb BLE/Bluetooth 4.0.

1.2 Drivers radino

Per a treballar amb la plataforma `radino` i l'IDE d'Arduino és necessari importar algunes llibreries:

1. Descarregar el fitxer comprimit de l'enllaç `http://www.ic42.de/BSP/radino/ICT_Boards.zip`
2. Extreure'n les carpetes `hardware` i `libraries` i copiar-les a l'sketchbook d'Arduino (en cas que `libraries` ja hi sigui només n'ha de quedar una)
3. Obrir l'IDE i comprovar que s'han importat els exemples i drivers: `Examples` → `radino`, `Examples` → `BLE` i `Board` → `In-Circuit radino nRF8001`

1.3 BluePy

Les llibreries de `BluePy` ofereixen les funcionalitats de `gatttool` per a Python. Els passos per a instal·lar-lo juntament amb les seves dependències són:

```
sudo apt-get install build-essential libglib2.0-dev libdbus-1-dev
git clone https://github.com/IanHarvey/bluepy.git
cd bluepy/bluepy
make
```

Amb aquesta comanda s'hauria d'haver creat l'executable `bluepy-helper`, en el qual hi ha funcions auxiliars per al mòdul `btle`. Finalment cal copiar els fitxers al directori reservat per a les llibreries Python de l'usuari.

```
cp bluepy-helper /usr/local/lib/python2.7/dist-packages/bluepy-helper
cp __init__.py /usr/local/lib/python2.7/dist-packages/__init__.py
cp btle.py /usr/local/lib/python2.7/dist-packages/btle.py
```

Si la instal·lació és correcta s'hauria de poder importar el mòdul `btle` des de qualsevol path. Per exemple des de l'interpret de Python:

```
>>> import btle
```

Alternatives similars a `bluepy` són `pygatt`, `pybluez` o bé `gatttlib`.

2 Bluetooth Low Energy

En el protocol de comunicació BLE els dispositius poden prendre dos rols: perifèric o central. Els primers solen ser dispositius de recursos reduïts que executen la funció de servidor (`radino`). Els aparells centrals, en canvi, tenen una major capacitat (smartphones o ordinadors) i actuen com a clients demanant dades als perifèrics. Per a cada dispositu central hi pot haver diversos perifèrics connectats, però no al revés.

El protocol BLE es pot dividir en dues fases: establiment de la connexió (GAP) i comunicació entre dispositius (GATT).

2.1 Generic Access Profile (GAP)

GAP és el procés utilitzat per a establir una connexió Bluetooth entre un dispositiu perifèric i un de central (figura 1).

1. El perifèric envia *advertisements* per a què els dispositus centrals el puguin detectar
2. El disp. central que ho desitgi respon amb un *scan response request*
3. En cas que el disp. central demani dades extres (opcional), el perifèric respondrà amb un *scan response data*

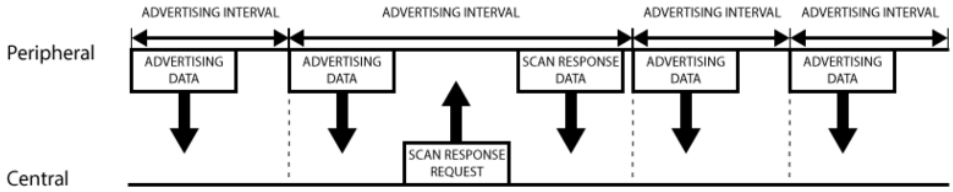

Figura 1: Establiment de connexió

2.2 Generic Attribute Profile (GATT)

Una vegada s'ha estalbert la connexió, aquesta és gestionada des de l'aparell central enviant peticions d'informació al perifèric tal i com es mostra en la figura 2.

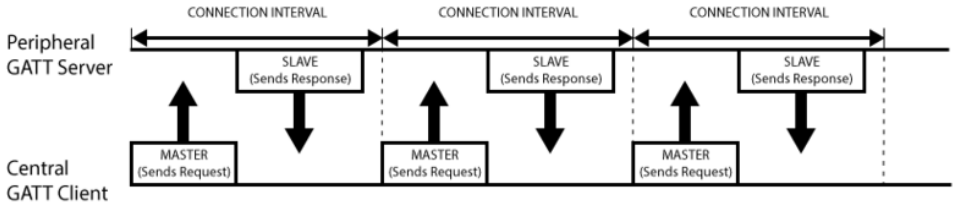

Figura 2: Model client-servidor (GATT)

El servidor emmagatzema les dades en taules d'atributs. Els atributs també es poden anomenar característiques (figura 3) i són l'estructura mínima de dades, formada per tres elements:

- `handle`: Identificador de 16 bits (únic per a cada característica)
- `UUID`: Tipus d'atribut (16 - 64 bits)
- `Valor`: Dades de mida varible

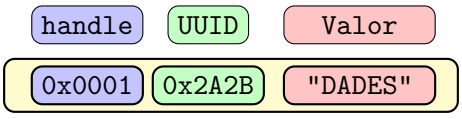

Figura 3: Exemple d'atribut / característica

2.2.1 Serveis i característiques

Les característiques estan agrupades en serveis i perfils. El perfil és el conjunt més gran i ofereix serveis dedicats a una funció concreta (per exemple existeixen perfils per a mesurar la temperatura o la pressió sanguínea [Blu15a]). Els serveis són taules de característiques d'informació i configuració a les quals es pot accedir a partir del seu `handle`. També es pot cercar a les taules a través de l'`UUID` per a conéixer les característiques d'un determinat tipus.

Un exemple d'estàndard de perfil és el de rellotge [Blu15b], el qual conté tres serveis per a conéixer l'hora actual, actualitzar el canvi d'hora d'estiu i actualitzar l'hora a partir d'un altre dispositiu. Cada servei compta amb diferents característiques tal i com es mostra en la figura 4.

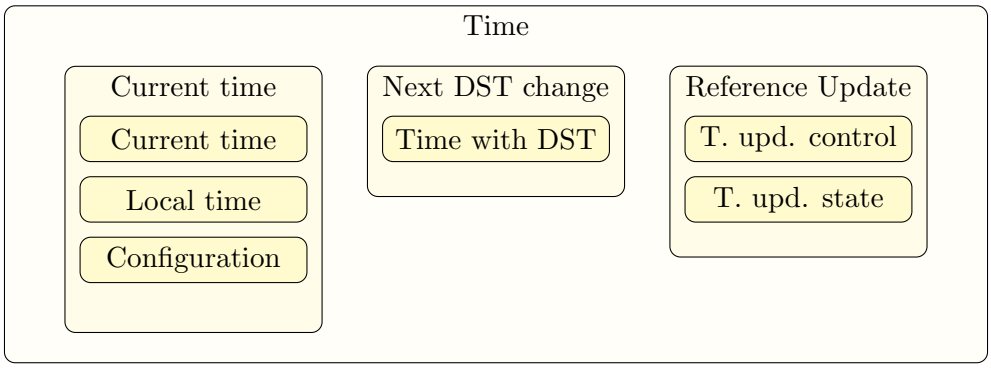

Figura 4: Serveis i característiques del perfil *Time*

Una característica especial és la que té `UUID = 0x2800`, que correspon a l'inici de cada servei. En l'exemple de la figura 5, el rang de `handles` del `0x00` fins al `0xFF` pertanyen al primer servei i del `handle 0x100` fins al `0x105` al segon.

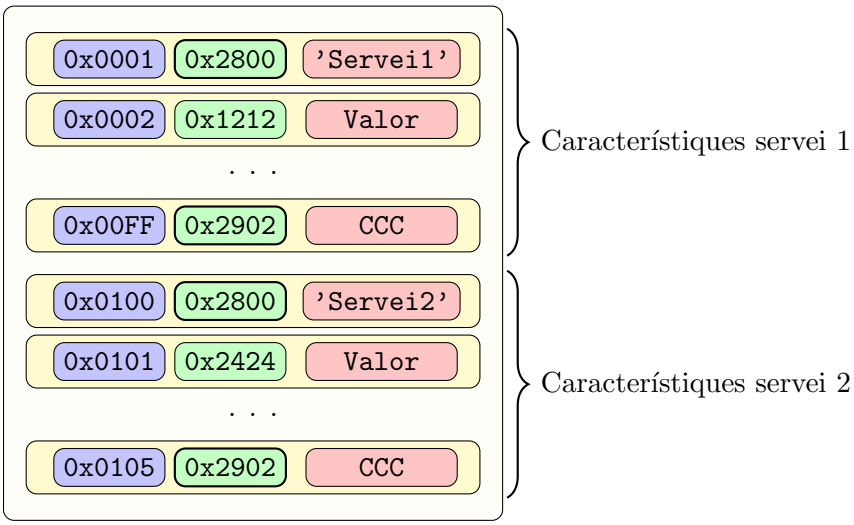

Figura 5: Exemple de taula d'atributs per a dos serveis

Una altre atribut important és el que té `UUID = 0x2902`, dedicat a Configurar la Comunicació amb el Client (CCC). Aquesta característica permet que el servidor enviï dades sense que hi hagi una petició per part del client. Es poden activar notificacions (les dades s'enviaran del servidor

al client) i indicacions (el servidor esperarà també una resposta de reconeixement). Els atributs amb `UUID = 0x2902` poden prendre els següents valors:

- `0x0000` : Notificacions i indicacions deshabilitades
- `0x0100` : Notificacions habilitades
- `0x0200` : Indicacions habilitades
- `0x0300` : Notificacions i indicacions habilitades

En l'exemple de la figura 5, si volguéssim enviar dades de forma autònoma des del primer servei hauríem d'escriure el valor `0x0100` en la característica `0xFF`.

Es pot trobar una informació més detallada sobre el protocol GATT a les referències [Elv14], [Car14] i [Kev14].

3 Implementació

A continuació es descriuen els passos a seguir per a transmetre dades des del `radino` cap a un ordinador de forma periòdica. Les dades es poden mostrar per terminal o bé processar-les des d'una aplicació Python.

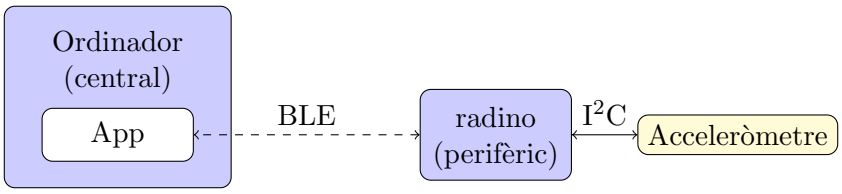

Figura 6: Transmissió a través de BLE

L'ordinador actuarà com a dispositiu central, encarregant-se de gestionar la connexió i activar les notificacions del `radino` per a què pugui transmetre cada vegada que hi hagi una nova mostra de l'acceleròmetre.

3.1 UART over BLE

El primer pas és implementar la comunicació entre ordinador i `radino`. La connexió BLE simularà una transmissió `UART` a partir de dos canals TX i RX (per als quals s'hauran d'activar les notificacions).

Per a comunicar-se amb el `radino` des de l'altre costat s'utilitzarà el port sèrie convencional.

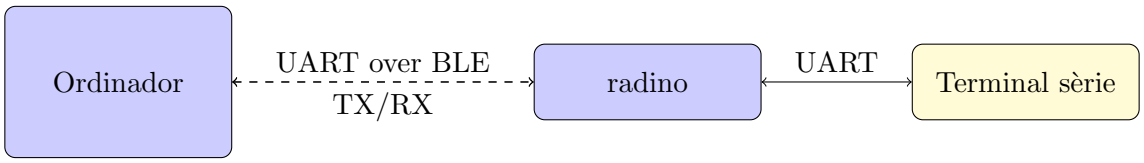

Figura 7: Comunicació sèrie a través de radino

Els passos per a carregar el programa al `radino` són:

1. Obrir l'exemple des de l'entorn IDE (`Examples` → `radino` → `radino_nRF8001` → `radino_nRF8001_IO_HF_USB_Test`)

2. Compilar i carregar el programa amb el `Port` i `Board` corresponents (en cas de tenir problemes visitar apartat 4.1)

3. Obrir el port sèrie. Al cap de 5 segons s'hauria d'iniciar l'enviament d'*advertisements*

Un cop s'ha iniciat el procés de sol·licitud podem connectar-hi qualsevol dispositiu amb Bluetooth 4.0.[1] Per a accedir-hi des de l'ordinador s'utilitzen les eines `hcitool` i `gatttool` (1.1):

1. El primer que cal fer és detectar el perifèric que sol·licita connexió. La seva adreça MAC i nom es poden trobar amb:

   ```
   hcitool lescan
   ```

2. Una vegada coneixem l'adreça podem utilitzar `gatttool`. El mode de comunicació serà interactiu (`-I`) i el tipus d'adreça privada (`-t random`)

   ```
   gatttool -b XX:XX:XX:XX:XX:XX -I -t random
   ```

3. Per a enviar comandes o demanar dades al servidor ens hi hem de connectar:

   ```
   > connect
   ```

4. Si la connexió ha estat possible ja podem accedir al servidor i llistar les seves característiques:

   ```
   > characteristics
   ```

5. Per a què el perifèric pugui enviar dades a l'ordinador s'han d'activar les notificacions corresponents.

   ```
   char-read-uuid 0x2902
   ```

6. La comanda ens retornarà els handles corresponents a les característiques que poden activar notificacions. Per a activar-les s'ha de modificar el valor a 0100:

   ```
   char-write-req 0x001c 0100
   char-write-req 0x0022 0100
   char-read-uuid 0x2902
   ```

   Només hem d'activar les notifiacions dels handles `0x001C` (transmissió) i `0x0022` (recepció), corresponents als canals TX i RX (`https://devzone.nordicsemi.com/documentation/nrf51/6.0.0/s110/html/a00066.html#project_uart_test_steps_nus_eval`).

7. Per a enviar dades des de l'ordinador cap al `radino` podem escriure al handle 0x1f

   ```
   char-write-cmd 0x1f 48454C4C4F
   ```

8. Per a enviar dades des del `radino` cap a l'ordinador escribim directament al port sèrie

[1] Per a fer-ho des d'un smartphone es pot utilitzar l'aplicació nRFUART (`https://play.google.com/store/apps/details?id=com.nordicsemi.nrfUARTv2&hl=en`).

3.2 Aplicació python

Un cop comprovat el correcte funcionament des de la línia de comandes reproduirem els mateixos passos des de l'interpret de Python utilitzant el mòdul `btle` (1.3).

El primer pas és crear un perifèric i connectar-lo a l'adreça MAC del `radino` en mode privat (random)

```
>>> import btle
>>> ble = btle.Peripheral()
>>> ble.connect("XX:XX:XX:XX:XX:XX", "random") # MAC radino
```

Un cop s'ha creat la connexió s'han d'activar les notificacions.

```
>>> ble._readCharacteristicByUUID(0x2902, 1, 2000)
```

La funció hauria de retornar els mateixos handlers que des de `gatttool` ('hnd' : [11, 28, 34, 37]).

```
>>> ble.writeCharacteristic(28, '\x01\x00')
>>> ble.writeCharacteristic(34, '\x01\x00')
>>> ble._readCharacteristicByUUID(0x2902, 1, 2000)
```

Si efectivament s'han modificat els atributs de configuració ja es poden rebre notificacions.

```
ble.waitForNotifications(10)
```

La funció retorna `True` en cas de rebre notificacions i `False` en cas que s'acabi el *timeout* (10 segons). Per a poder atendre a les notificacions s'ha de crear un delegat.

```
>>> class UserDelegate(btle.DefaultDelegate):
        def handleNotification(self, cHandle, data):
            print "Dades: ", data, "Handle: ", cHandle

>>> ble.setDelegate(UserDelegate())
>>> ble.waitForNotifications(10)
```

Per a enviar dades podem accedir també al registre 0x1F:

```
>>> ble.writeCharacteristic(0x1F, '\x48\x45\x4C\x4C\x4F')
```

Aquestes i les altres funcions del mòdul `btle` poden ser utilitzades des de qualsevol programa Python. Per a llegir o modificar-les es pot accedir al codi font (`dist-packages/btle.py`).

3.3 Programa final radino

En l'exemple explicat les dades són enviades cada vegada que s'escriu pel port sèrie. El programa final haurà de llegir periòdicament de l'acceleròmetre i enviar pel canal TX de Bluetooth. Els punts que s'han de tenir en compte en el programa final són:

- Cal importar les llibreries de l'acceleròmetre i inicialitzar-lo en la funció de setup()
- Cal modificar el bucle principal per a què tingui dos estats:
  - Inicialització: Espera a què s'hi hagi connectat un dispositiu central, el canal de transmissió estigui actiu i l'interval de la connexió actualitzat
  - Enviament de mostres: Lectura periòdica (T=30ms) de l'acceleròmetre i enviament de dades amb lib_aci_send_data()

En les dues fases s'encarrega també d'actualitzar l'estat de la radio entrant en el bucle aci_loop():

```
void loop() {
  char buff[20];
  int data[3];
  aci_loop();
  delay(29);
  if (is_connected && pipe_available && interval_changed){
    Accel_1.readAccel(data);
    if (Accel_1.status)
      sprintf(buff, "%d,%d,%d", data[0], data[1], data[2]);
    else
      sprintf(buff, "0,0,0");
    lib_aci_send_data(PIPE_UART_OVER_BTLE_UART_TX_TX,
                      (uint8_t *)buff, strlen(buff));
  }
  else
    delay(1500);
}
```

Les variables globals is_connected, pipe_available i interval_changed són del tipus **static** bool i estan inicialitzades a false. Cal habilitar-les segons els esdevebunebts d' aci_loop():

- El dispositiu estarà connectat quan hi hagi un esdeveniment ACI_EVT_CONNECTED
- El canal TX estarà actiu quan hi hagi un esdeveniment ACI_EVT_PIPE_STATUS i lib_aci_is_pipe_available(...) sigui true
- L'interval de temps s'actualitzarà amb l'estat ACT_EVT_TIMING

- No cal esperar a què hi hagi la connexió d'un terminal sèrie per a què comenci l'execució del programa, es poden comentar les següents línies

```
//while(!Serial);
//delay(5000);
```

- Es pot prescindir de la funcionalitat de recepció

Una vegada realitzades aquestes modificacions el programa estarà en espera mentre no s'hi hagi connectat un altre dispositiu i s'hagi inicialitzat el canal de transmissió. Aleshores iniciarà l'enviament periòdic de dades de l'acceleròmetre. Encara que no és imprescindible per al funcionament del programa, es pot utilitzar el port sèrie per a seguir-ne l'execució o debuggar-lo.

4 Possibles problemes

En aquest apartat es descriuen alguns problemes que s'han trobat a l'hora d'executar els passos anteriors i les respectives solucions o alternatives.

4.1 Càrrega de programes al radino

En algunes versions de Linux no és possible carregar els exemples de mostra `radino` degut al següent error:

```
avrdude: ser_open(): can't open device "/dev/ttyACM0": device or resource busy
```

Pot ser que l'usuari que executa l'Arduino IDE no tingui permís d'accés al port sèrie. Per a corregir-ho cal introduir-lo al grup *dialout*:

```
sudo adduser nomusuari dialout
```

L'error també pot ser causa d'un conflicte amb el ModemManager del sistema [In-15]. Per a resoldre-ho es pot crear una regla d'`udev` [Gre14]:

```
sudo emacs /etc/udev/rules.d/70-modemmanager-ignore-radino.rules
```

El contingut del fitxer ha de ser:

```
ACTION!="add|change", GOTO="ignore_radino_end"
SUBSYSTEM!="usb", GOTO="ignore_radino_end"
ENV{DEVTYPE}!="usb_device", GOTO="ignore_radino_end"

# Ignore ICT products.
ATTRS{idVendor}=="1da9", ENV{ID_MM_DEVICE_IGNORE}="1"

LABEL="ignore_radino_end"
```

Referències

[Blu15a] Bluetooth Special Interest Group. *Bluetooth Developer Portal. Profiles*. Anglès. 2015. URL: `https://developer.bluetooth.org/gatt/profiles/Pages/ProfilesHome.aspx` (consultat 5 de mai. de 2015).

[Blu15b] Bluetooth Special Interest Group. *Bluetooth Development Portal. Time Profile*. Anglès. 2015. URL: `https://developer.bluetooth.org/gatt/profiles/Pages/ProfileViewer.aspx?u=org.bluetooth.profile.time.xml` (consultat 5 de mai. de 2015).

[Car14] Carles Cufí; Akiba; Robert Davidson; Kevin Townsend. *Getting Started with Bluetooth Low Energy*. Addison-Wesley Series in Computer Science and Information Processing. O'Reilly Media, 2014. URL: `https://www.safaribooksonline.com/library/view/getting-started-with/9781491900550/ch04.html` (consultat 5 de mai. de 2015).

[Elv14] Elvis Pfuxzenreuter. *Bluetooth: ATT and GATT*. Anglès. 2014. URL: `https://epxx.co/artigos/bluetooth_gatt.php` (consultat 5 de mai. de 2015).

[Gre14] Greg Kroah-Hartman; Kay Sievers. *udev - Linux Dynamic Device Management*. Anglès. 2014. URL: `https://www.kernel.org/pub/linux/utils/kernel/hotplug/udev/udev.html` (consultat 8 de mai. de 2015).

[In-15] In-Circuit. *Radino Common Problems*. Anglès. 2015. URL: `http://wiki.in-circuit.de/index.php5?title=radino_common_problems#radino_.26_Linux` (consultat 5 de mai. de 2015).

[Kev14] Kevin Towsend. *Introduction to Bluetooth Low Energy. A basic overview of key concepts for BLE*. Anglès. Adafruit Industries. 2014. URL: `https://learn.adafruit.com/introduction-to-bluetooth-low-energy/introduction` (consultat 5 de mai. de 2015).

[eLi15] eLinux Community. *RPi Bluetooth LE*. Anglès. 2015. URL: `http://www.elinux.org/RPi_Bluetooth_LE` (consultat 5 de mai. de 2015).