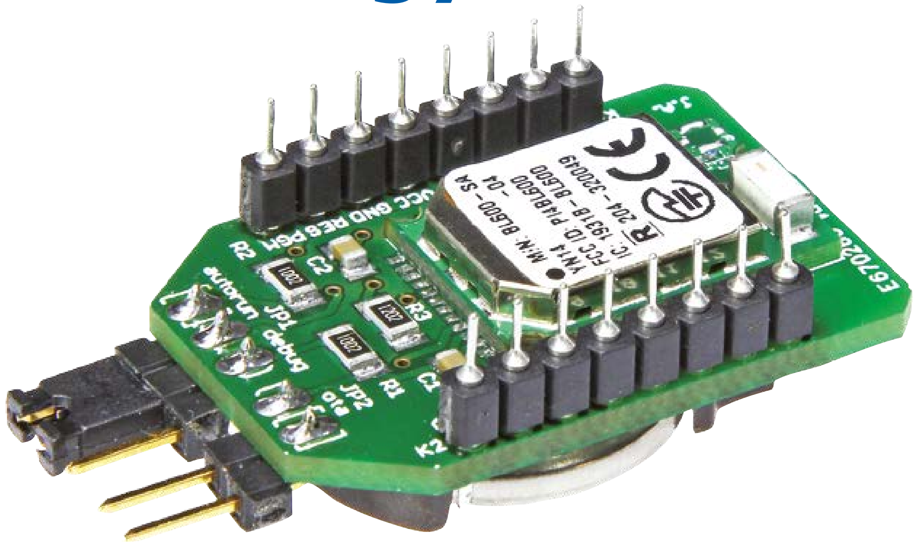
BL600 e-BoB
Bluetooth Low Energy Module

(part 6)

- Low Energy, 5 µA
- The TIMER event
- Data exchange events
- The connection event



By **Jennifer Aubinais** (France) elektor@aubinais.net

In the previous installments, we have examined together the basics of programming our BL600 e-BoB BL600 and shown how to implement it. In this the sixth and final installment in the series we take a look at its low consumption (Low Energy) which is precisely its strong point! Then you'll be flying solo.

Low Energy, 5 µA
In deep-sleep mode, the consumption is only 0.4 µA, but in this mode it can only be woken up by resetting the module or by a change of state on an input. Instead, we're going to be using the stand-by mode, with a consumption of 5 µA, and module will be woken up by our program, e.g. by using TIMERs. To illustrate these explanations, we're going to be using just a single LED. If you connect other devices to your BL600, you'll need to adapt your program so as to maintain the 5 µA in stand-by mode. To highlight the module's low consumption, we're going to make it operate alternately in normal and stand-by mode. Connect your (micro-) ammeter as per the diagram: when the LED is out, the current is 5 µA.

The source code for the *bl600-6.sb* test program shown below is available from the Elektor Magazine website [1]. If you want to alter it, refer to the previous installments which give all the details on this subject.
The stages:

- **Stopping the Bluetooth:** the Bluetooth is stopped using the *BleAdvertStop* function and woken up again by the *BleAdvertStart* function. The alternation will be achieved using TIMERs (*TIMER* paragraph for the description of the code in red and green).

- **LED off:** Everything connected to the module consumes power. So our LED will be taken LOW by the *GPIOWrite* function. We'll turn it on when the Bluetooth is running and turn it off when it is stopped (using functions in the TIMER – see below).

- **UART stopped in NON DEBUG mode**: The module's serial communication facility also consumes power. We'll turn it off using the *UartClose* function, the TX and RTS ports will be set to outputs with *High* and *Low* states respectively. Two modifications are needed here to avoid a ~~~*default*~~~ error:

  - If we close the serial port when the program is started, , the serial port buffer in the MAIN program is still full. Hence for it to be emptied, we shall wait 1000 ms (an arbitrary duration) before closing it, using a TIMER in the mauve code.
  - Once the serial port is closed, the HandlerLoop handler transfers the data received by the Bluetooth to the serial port and vice-versa using the *BleVspUartBridge* function. We'll put this function as a comment and will compose our own code in orange (see *Data exchange events*).

- **Measuring:** The high internal resistance of some current meters can hinder the measurement in awake mode. For example, with a resistance of 100 Ω, for a current of 10 mA (LED lit) the voltage drop across the meter will be 1 V: so the module won't work anymore… Select the 2000 µA range rather than 200 µA.

The TIMER event

The TIMER, covered in the May 2015 article on the light-chaser [4], is used to create alternating stops and restarts of the Bluetooth. To do this, we need to code:

- the declaration of its handle: *OnEvent EVTMR1 Call FuncTimer1*
- the function: *Function FuncTimer1()*
- the trigger for the TIMER: *TIMERSTART(1,4000,0)*

In order for the module's low consumption to show up on an meter, we enable the Bluetooth (LED on) for 25 ms (*bleadvertstop* function, code in red), and stop it (LED off) for 4000 ms (*bleadvertstart* function, code in green).

```
n = Bleadvertstart(0,Adrt$,25,4000,0)
```

The TIMER will be used again to delay the closing of the UART port.

Data exchange events

The data received by the Bluetooth and on the serial port are processed by a handler, just as we did, for example, to intercept the data coming from your phone to control a 3-colour LED [4]. Here, this poses us a problem, as we have closed the UART port; this is why we have deleted the *BleVspUartBridge* code (put into a comment). To illustrate this example, the text received in Bluetooth will be sent back in Bluetooth, but with an offset of one character (I'll come back to this later).

The connection event

In the previous articles, we've already tackled the *EVBLEMSG* event in order to be able to handle connection and disconnection events. In the thermometer program in the January/February 2015 article [1], when connecting, the first step is to recover the value from the NTC, the second is to send this to the smartphone, and the last, to disconnect the module from the phone. This time, we're going to light an LED on output 2 of our module when it is connected; this will be turned off when disconnected.
Don't forget the initialization of port 2 when exiting the main program. We're going to copy the Bluetooth message handler *EVBLEMSG* from the *cli.manager.sblib* library and create our own handler, renaming it *MyHandlerBleMsg* (code in blue).

Creating our program

To end this list of the elements to be recycled in order to create our *Low Energy* program here, we must lastly mention the *upass.vsp.sb* program used in other articles as the basis for the UART services [1][4][6]. This will be reworked and will be used to modify the data received by our module. To do this, we rename it *BL600-6.sb* and change two parameters in this first version:

- ENABLE_DEBUG_PRINTS: 0 (*debug* mode disabled)
- DEVICENAME: "JA_TEST"
- You can test the operation of your program by transferring it to your BL600 e-Bob. It will run automatically, otherwise enter the command *AT+RUN* [3]. You can now test your program's communication using the *SERIAL* program from *TOOLKIT* on your smartphone [2].

The two TIMERs:

- Code in green:
  - declaring its handle: *OnEvent EVTMR1 Call FuncTimer1*
  - the function: *Function FuncTimer1()*
  - turning the LED on: *GpioWrite(2,1)*
  - starting the Bluetooth: *BleAdvertStart*
  - the trigger for the next TIMER: TIMERSTART(2,25,0)

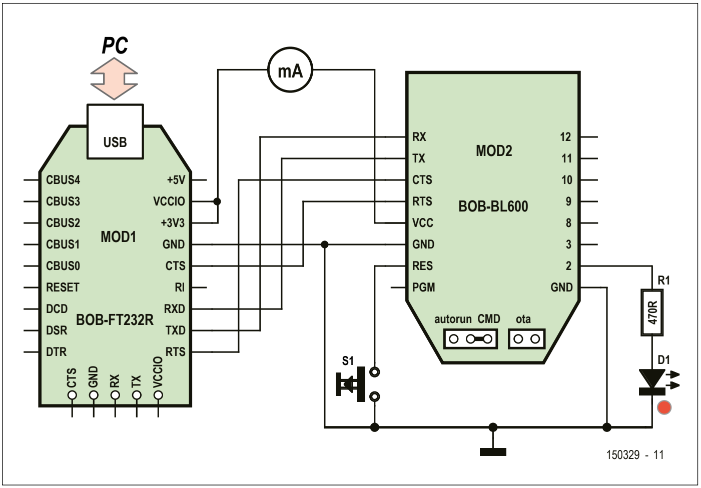


Figure 1. As usual, we'll be using the Elektor USB-serial FT232 bridge [8] for communicating with the BL600.

Component List

Resistors :
R1 = 470Ω

Semiconductors
D1 = LED, 3mm (select color)

Miscellaneous
KS1 = pushbutton
MOD1 = FT232 e-BoB, ready assembled, Elektor Store # 110553-91 [7]
MOD2 = BL600 e-BoB, ready assembled, Elektor Store # 140270-91 [7]
or printed circuit board # 140270-1 [7]

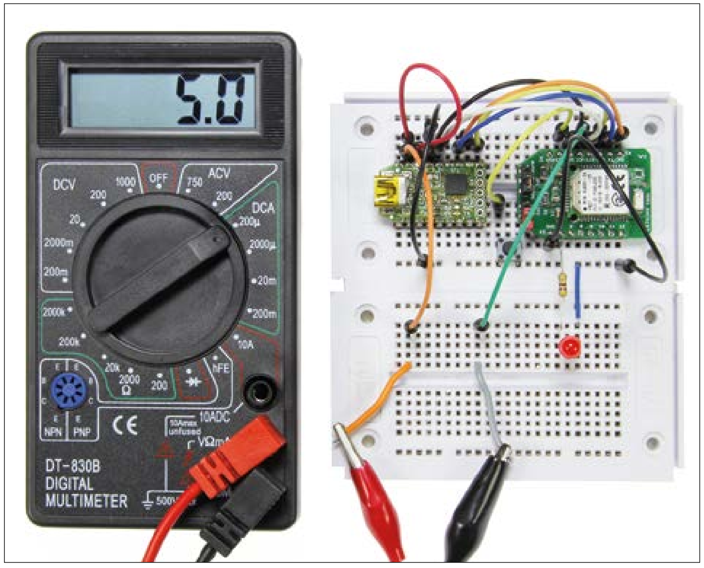


Figure 2. Photo of the experimental assembly on a prototyping board.

- Code in red:
  - declaring its handle: *OnEvent EVTMR2 Call FuncTimer2*
  - the function: *Function FuncTimer2()*
  - turning the LED off: *GpioWrite(2,0)*
  - stopping the Bluetooth: *BleAdvertStop*
  - the trigger for the next TIMER: *TIMERSTART(1,4000,0)*

The *BleVspUartBridge* function must be deleted if we want to close the UART and hence consume less power. To show the module is functioning, we send back the received text with an offset of one character: a becomes b, d becomes e, etc. (code in orange).

- We've taken the code for the *HandlerLoop* function from the *cli.upass.vsp.sblib* library. To avoid duplicates that would cause a compilation error, let's rename our function ***My****HandlerLoop*.

- The four handlers that call the *HanderLoop* function:
  - data arrive at the module's UART port
  - data arrive at the module's Bluetooth port
  - the module's Bluetooth port buffer is empty
  - the module's UART port buffer is empty

- The *BleVspUartBridge* function creates a loop between the phone and the UART port. We put this line of code into a comment.

- The *BleVSpRead* function lets us read the data received by the Bluetooth. Attention, the manufacturer has imposed a limit of 20 characters.

- The *STRPOS* function makes it possible to know if the text received ends with a carriage return.

- To send back the received text with an offset of one character, we're going to use the two functions *StrGetChr* (convert a character at a given position in a string to a byte) and *StrSetChr* (change the value of a character by the value of a byte at a defined position).

- To end, the text will be transmitted in Bluetooth via the *BleVspWrite* function.

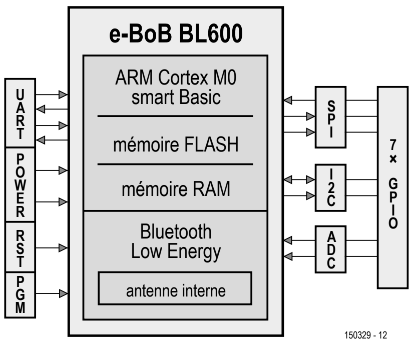
e-BoB BL600

Reminder of the BL600 module specifications:

- UART, I²C, SPI interfaces
- 28 general-purpose inputs/outputs (GPIO)
- 6 analog inputs (10-bit ADC)
- consumption:
- 5 µA in stand-by / 0.4 µA in deep-sleep mode
- 10 mA during transmission
- easy programming in smartBASIC
- nRF51822 chip from Nordic Semiconductor.
- compact: 19 × 12.5 × 3 mm
- free-field range: up to 20 m

Bluetooth in BASIC!

The BL6000-SA's smartBASIC (event-oriented) programming language simplifies the incorporation of Bluetooth into your applications by making it easier not only to manage sensors connected directly to the module, but also to transmit the measured values to any Bluetooth v4.0 receiver (smartphone or tablet, computer, bridge, etc.)



Specifications of the Elektor e-BoB breakout board using the BL600-SA:

port K1	serial port used for loading the program into the BL600 power pins (3.3 V) reset line (RESET) PGM pin (for possible updates to the module's firmware)
port K2 7 logic input/ outputs	2 × 10-bit analog inputs (pins 2 and 3) I²C port (pins 8 and 9) SPI port (pins 10, 11, and 12)
jumper JP1	cmd: AT commands autorun
jumper JP2	OTA (Over The Air)

Android application and program

On the BL600 manufacturer's website [7], you will find the source code for the *Toolkit* application for Android (and also iOS) which includes the following services:

- BPM (blood pressure)
- HRM (heart rate)
- *Proximity*
- MTH (medical thermometer)
- *Serial* (UART)
- OTA (Over The Air)
- *Batch*

To thrive, connected object networks need low power consumption wireless communication. This breakout board for the Bluetooth Low Energy module is the dream accessory for exploring the IoT.

The program will no longer send data to the UART port, so we can close the UART port, after an arbitrary delay of 1000 ms (code in **mauve**):

- declaring its handle: *OnEvent EVTMR0   Call FuncClose*
- the function: *Function FuncClose()*
- closing the port: *UartClose()*
- the TX and RTS ports will be set to outputs with *High* and *Low* states respectively: *GPIOSetFunc*
- triggering the TIMER in the MAIN program: *TIMERSTART(0,1000,0)*

We want to intercept the status of the connection so as to turn the LED on when connecting or off when disconnecting (in reality, the LED will flash as a result of TIMER1 and TIMER2).

The *HandlerBleMsg* function code in blue comes from the *cli.upass.vsp.sblib* library:

- We've taken the code for the *HandlerBleMsg* function from the *cli.upass.vsp.sblib* library. To avoid duplicates that would cause a compilation error, let's rename our function ***My****HandleBleMsg*.
- When connecting via BLE_EVBLEMSGID_CONNECT:
  - the TIMERs are stopped: *TIMERCANCEL*
  - the LED is turned on: *GpioWrite(2,1)*

We have offered you for download [6] the extremely simplified source code for an Android phone application using just the UART service. We have used Android Studio, available under Windows, MAC OS and Linux [7]. A reminder that the manufacturer has created a library `laird_library_ver.0.18.1.1.jar` in order to speed up development of Android applications in normal Bluetooth and Bluetooth Low Energy.

The functions described in this series of articles:

inputs / outputs:
- configuring the pin on the BL600 (nFunction = 1 or 2): `rc = GPIOSETFUNC(nSigNum, nFunction, nSubFunc)`
- reading the signal 0 or 1: `rc = GPIOREAD(nSigNum)`
- writing (nNewValue = 0 or 1): `rc = GPIOWRITE(nSigNum, nNewValue)`

analog/digital converter:
- configuring the pin on the BL600 (nFonction = 3) : `rc = GPIOSETFUNC(nSigNum, nFunction, nSubFunc)`
- reading the signal (0 to 1023): `rc = GPIOREAD(nSigNum)`

I²C port:
- opening the port: `rc = I2COpen(nClockHz, nCfgFlags, nHandle)`
- writing an 8-bit value: `rc = I2CWriteReg8(nSlaveAddr, nRegAddr, nRegValue)`
- reading an 8-bit value: `rc = I2CReadReg8(nSlaveAddr, nRegAddr, nRegValue)`
- closing the port: `I2Close(handle)`

SPI port:
- opening the port: `rc = SpiOpen(nMode, nClockHz, nCfgFlags, nHandle)`
- reading / writing: `rc = SpiReadWrite(stWrite$, stRead$)`
- closing the port: `SpiClose(handle)`

Tools for compiling and transferring

All the tools and examples are available for download [7]. When you apply to open an account on the manufacturer's website, specify: "Elektor reader". All you have to do is download and then unzip the document `Firmware Files version 1.5.70.0 – Revision 5 i` which contains:

- program examples in the `smartBASIC_Sample_Apps` directory
- the `UwTerminal.exe` program in the `smartBASIC_Sample_Apps` directory
- the library in the `smartBASIC_Sample_Apps/lib` directory
- a number of examples (`UserManualExampleCode`)
- the special Notepad++ configuration for smartBASIC (`smartBASIC(notepad++).xml`)

Normal commands

- AT I 0: BL600 revision number
- AT I 3: BL600 firmware version
- AT+DIR: list of the programs in the BL600
- ATZ: reset BL600
- AT&F 1: clear memory and restart BL600
- AT+RUN "xxxx": run program xxxx

- When disconnecting via BLE_EVBLEMSGID_DISCONNECT:
  - the LED is turned off: *GpioWrite(2,0)*
  - the first TIMER is started again: *TIMERSTART(1,10,0)*

Now all the elements are there for you to create your own `BL600-6.sb` program. After checking the low consumption of our module, it's desirable to modify the value of the TIMER defined as 1 from 4000 ms to e.g. 300 ms for a better connection with your smartphone: *TIMERSTART(1,300,0).* The moment has come to launch yourself into programming *your* BL600, on *your* own e-BoB, for *your* ANDROID application; the information you need is all in last month's article [6].

Open conclusion

With this summary of what we have described since the initial Bluetooth thermometer project, we are closing this 6-installment article with double satisfaction: our efforts to facilitate the implementation of this module have been recognized and appreciated by not only the Elektor readers who are buying the e-BoB, but also the manufacturer of the BL600-SA. Other projects based on the BL600 are waiting in the author's and Elektor's laboratories. What's more, the BL600 has a cousin, the BL620, another Bluetooth module, but with the role of Master, which makes communication even easier. You'll be seeing it soon in Elektor Magazine…

(150329)

```
'//**************************************************
'// Laird Technologies (c) 2013
'// Jennifer AUBINAIS (c) 2015
'// ++++++++++++++++++++++++++++++++++++++++++++++++

'//**************************************************
'// Definitions
'//**************************************************
#define AUTO_STARTUP                        1
'//Set this to 0 to disable all debugging messages
#define ENABLE_DEBUG_PRINTS                 0
#define DEVICENAME                          "JA_TEST"
#define DEVICENAME_WRITABLE                 1
***** code here *****

'//*************************************************
'// Library Import
'//*************************************************
#include "lib\cli.upass.vsp.sblib"
'//*************************************************
'// Global Variable Declarations
'//*************************************************
DIM text$
'//*************************************************
'// Function and Subroutine definitions
'//*************************************************

'//================================================
'// CLOSE UART
'//================================================
Function FuncClose()
  DIM rc
  UartClose()
  rc = GPIOSetFunc(21,2,1) '// TX
  rc = GPIOSetFunc(23,2,0) '// RTS
  TIMERSTART(1,10,0)
ENDFUNC 1
```

```
'//================================================
'// Receive data
'//================================================
function MyHandlerLoop()
  //BleVspUartBridge()
  DIM n, rc, tempo$, tx$
  DIM pos, return$
  // Wait return from received data
  tx$ = "0D"
  return$ = StrDehexize$(tx$)
  tempo$ = ""
  n = BleVSpRead(tempo$,20)
  text$ = text$ + tempo$
  pos = STRPOS(text$,return$,0)
  IF ( pos >= 0 ) THEN
    DIM i, x
    pos = pos - 1
    FOR i = 0 TO  pos
      x = StrGetChr(text$,i)
      rc = StrSetChr(text$,x+1,i)
    NEXT
    rc = BleVspWrite(text$)
    text$ = ""
  ENDIF
endfunc 1

'//==================================================
'// TIMER 1
'//==================================================
FUNCTION FuncTimer1()
  dim rc, Adr$
  Adr$ = ""
  // led on
  GpioWrite(2,1)
  rc = bleadvertstart(0,Adr$,25,25,0)
  TIMERSTART(2,25,0)
ENDFUNC 1
```

Web Links

[1] **Elektor January & February 2015**
www.elektormagazine.com/140190
Bluetooth Low Energy wireless thermometer
Remote temperature display on your smartphone

[2] **Elektor March & April 2015**
www.elektormagazine.com/140270
BL600 e-BoB: Part 1:
Wireless communication on a plate

[3] **Elektor May & June 2015**
www.elektormagazine.com/150014
BL600 e-BoB | Part 2:
Editing, compiling, and transferring a program using the Bluetooth Low Energy module

[4] **Elektor July & August 2015**
www.elektormagazine.com/150129
BL600 e-BoB | Part 3:
smartBASIC programming for the Bluetooth Low Energy module

[5] **Elektor September & October 2015**
www.elektormagazine.com/150130
BL600 e-BoB | Part 4:
The I²C port and the temperature sensor

[6] **Elektor November & December 2015**
www.elektormagazine.com/150272
BL600 e-BoB | Part 5:
SPI port & digital/analog converter
Android application

[7] https://laird-ews-support.desk.com/?b_id=1945

[8] www.elektor.com/ft232r-usb-serial-bridge-bob-110553-91

```
'//====================================================
'// TIMER 2
'//====================================================
FUNCTION FuncTimer2()
  // led off
  GpioWrite(2,0)
  rc = bleadvertstop()
  TIMERSTART(1,4000,0)
ENDFUNC 1
'//====================================================
'// This handler is called when there is a BLE
message
'//====================================================
function MyHandlerBleMsg(BYVAL nMsgId AS INTEGER,
BYVAL nCtx AS INTEGER) as integer
'// Inform libraries
ConnMngrOnBleMsg(nMsgId,nCtx)
AdvMngrOnBleMsg(nMsgId,nCtx)
select nMsgId
  case BLE_EVBLEMSGID_CONNECT
    TIMERCANCEL(2)
    TIMERCANCEL(1)
    DbgMsgVal(" --- Connect : ",nCtx)
    ShowConnParms(nCtx)
    // set at High
    GpioWrite(2,1)
  case BLE_EVBLEMSGID_DISCONNECT
    DbgMsgVal(" --- Disconnect : ",nCtx)
    // set at Low
    GpioWrite(2,0)
    TIMERSTART(1,10,0)
  ***** code here *****
endselect
endfunc 1
'-----------------------------------------------------
'// TIME OUT = nothing
'-----------------------------------------------------
Function MyHandlerTimOut() as integer
  'NOTHING
EndFunc 1
'//**************************************************
'// Handler definitions
'//**************************************************
ONEVENT EVTMR1 CALL FuncTimer1
ONEVENT EVTMR2 CALL FuncTimer2
OnEvent EVTMR0  Call FuncClose
OnEvent  EVBLEMSG           call MyHandlerBleMsg //
EVBLEMSG indicate when signifficant BLE event occurs.
OnEvent  EVBLE_ADV_TIMEOUT  Call MyHandlerTimOut
//all events have the same handler
OnEvent  EVVSPRX            call MyHandlerLoop //
EVVSPRX is thrown when VSP is open and data has
arrived
OnEvent  EVUARTRX           call MyHandlerLoop //
EVUARTRX  = data has arrived at the UART interface
OnEvent  EVVSPTXEMPTY       call MyHandlerLoop
OnEvent  EVUARTTXEMPTY      call MyHandlerLoop
'//**************************************************
'// main
'//**************************************************
text$ = ""
// pin 2 output at low
rc = GPIOSetFunc(2,2,0)
rc = bleadvertstop()
IF (ENABLE_DEBUG_PRINTS == 1) THEN
  UartRsp(0)
  TIMERSTART(2,10,0)
ELSE
  PRINT "low Energy"
  TIMERSTART(0,1000,0)
ENDIF
WaitEvent
```