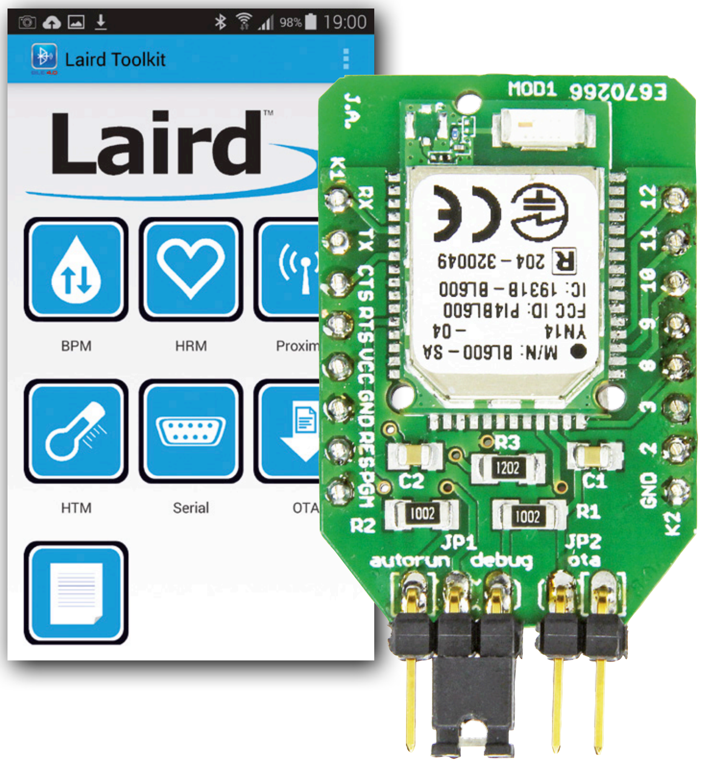
BL600 e-BoB

Part 3

smartBASIC programming for the Bluetooth Low Energy module



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

The aim of this series around the BL600 e-BoB is to make it easier to implement this remarkable module for wireless communication with devices you design yourself. The fact it can be programmed in smartBASIC is by no means the least of the BL600's qualities. In order to take full advantage of it, you do need get used to handling the events that make smartBASIC so powerful.

Following on from the description of the module's hardware and the tool needed to use it, we're now going to take a look at smartBASIC. This enables you to program the BL600 using what Laird Technologies call "events". I recommend you read their documentation [1]. As an application example, I'm going to be using the coding for our light-chaser from last month [2], and more specifically managing the offset time for turning the LEDs on and off and the chaser movement direction – all using these famous events. So in order to be able to follow, it's best if you've read the previous article. To go a bit further into it, you'll find it best to have it to hand.
Then, as an example of using Bluetooth communication, starting out from the UART program that's already been mentioned, we're going to control a 3-color LED. We'll intercept the characters send by the smartphone to turn the RGB LED on or off. This will give you the opportunity to use our eBOB-BL600 to control a commercial 3-color lamp or string of lights, for example. My `BLE RGB Lite` program is available on Google Play [3].

Handlers in the light-chaser

Our chaser program in the previous article (a simple *for next* loop for the timing, along with, for the LED chase, button position detection using an *if* condition) did not exploit the possibilities of the events in *smartBASIC* [see box]. This time, to do the same thing but more cleverly, we're using handlers, i.e. event managers.
We're going to be seeing two types of events: counting/timing and the changing state of a button. As much of the interest of *smartBASIC* lies in managing such events, it's essential to understand this little program properly before moving on to the next step.
The six LEDs connected to the eBoB-BL600's outputs 3–12 (see circuit and components list in last month's article [2]) turn on and off in succession. In the code in **Listing 1**, we're not going to linger over the black section, described in the previous issue of Elektor, but are going to take a look at what's going on in the red section of the code:

`WAITEVENT`

The command (or statement) `WAITEVENT` makes it possible to run the event manager. It's a sort of wait loop in which the system scans for the presence of events. This waiting stage is usually placed at the end of the main program (*main*).
The events are coupled to the managers (handlers) by `ONEVENT … CALL …` instructions, e.g. `ONEVENT EVTMR0 CALL FuncTimer0`, which means the `FuncTimer0` function is the handler for the event `EVTMR0`. The event names are predefined, but you can choose the names of the handlers.

`FuncTimer0` function
Here, we turn the LEDs on and off alternately at intervals of 200 ms (an arbi-

trary value). To do this, we create an event **EVTMR0** (EVTMR corresponds to a timer event, 0 corresponds to the number of the timer we've chosen) which is going to call the function **FuncTimer0** (this name is arbitrary) thanks to coupling via the instruction `ONEVENT EVTMR0 CALL FuncTimer0`.
Timer 0 is started by:
TIMERSTART(0,10,0)
where 0 is the event number, the same as in EVTMR0; 10 is the duration in ms of the timer counter; and lastly 0 for non-iterative; 1 for iterative.
The offset between turning the LEDs on and off is obtained by incrementing a counter (in our example, `led`) which turns one LED off and the next one on depending on its value. When the counter reaches the number of LEDs — for us, that's 6 — it is reset to zero and the LED sequence starts over.
At the end of `FuncTimer0`, timer 0 is restarted, this time for 200 ms.

Changing direction, FuncTimer1
To reverse the direction of our chaser, all we have to do is to decrement a counter starting from the number of LEDs — six, here — instead of incrementing it. At each decrement, depending on the counter value, one LED is turned off and the previous one is lit. Once the counter reaches zero, it is reset to 6 for a new LED sequence. That's what we have done here using Timer 1 with its event `EVTMR1` and its handler `FuncTimer1`. We could have achieved this more simply, but the aim here is to demonstrate events.

```
//TIMERSTART(0,10,0)
led = 6
TIMERSTART(1,10,0)
```

Try these lines… When you save your code, remember to delete the old program in the BL600 (don't forget the AT&F 1 command). Compile, transfer, and run. You'll see that the chaser starts in the other direction.

This example shows the simplicity of using timers: we can run a timer for a single (final parameter set to 0) or repeated (final parameter set to 1) count; it produces an event which runs a function [see box].

The "button" event
Before moving on, let's go back to the original code:

Listing 1.

```
Dim led, rc
'//-----------------------------------------------------------------------
FUNCTION FuncTimer0()
  PRINT "WAY + ";led;" \n"
  IF (led == 0) THEN : GpioWrite(12,0) : GpioWrite(3,1) : ENDIF
  IF (led == 1) THEN : GpioWrite(3,0) : GpioWrite(8,1) : ENDIF
  IF (led == 2) THEN : GpioWrite(8,0) : GpioWrite(9,1) : ENDIF
  IF (led == 3) THEN : GpioWrite(9,0) : GpioWrite(10,1) : ENDIF
  IF (led == 4) THEN : GpioWrite(10,0) : GpioWrite(11,1) : ENDIF
  IF (led == 5) THEN : GpioWrite(11,0) : GpioWrite(12,1) : ENDIF
  led = led + 1
  IF ( led >= 6) THEN : led = 0 : ENDIF
  TIMERSTART(0,200,0)
ENDFUNC 1
'//-----------------------------------------------------------------------
FUNCTION FuncTimer1()
  PRINT "WAY - ";led;" \n"
  IF (led == 6) THEN : GpioWrite(3,0) : GpioWrite(12,1) : ENDIF
  IF (led == 5) THEN : GpioWrite(12,0) : GpioWrite(11,1) : ENDIF
  IF (led == 4) THEN : GpioWrite(11,0) : GpioWrite(10,1) : ENDIF
  IF (led == 3) THEN : GpioWrite(10,0) : GpioWrite(9,1) : ENDIF
  IF (led == 2) THEN : GpioWrite(9,0) : GpioWrite(8,1) : ENDIF
  IF (led == 1) THEN : GpioWrite(8,0) : GpioWrite(3,1) : ENDIF
  led = led - 1
  IF ( led <= 0) THEN : led = 6 : ENDIF
  TIMERSTART(1,200,0)
ENDFUNC 1
'//-----------------------------------------------------------------------
FUNCTION Btn0Press()
  PRINT "PRESS DOWN\n"
  rc = GpioBindEvent(1,2,0)
  TIMERCANCEL(0)
  TIMERSTART(1,10,0)
ENDFUNC 1
FUNCTION Btn1Press()
  PRINT "PRESS UP\n"
  rc = GpioBindEvent(0,2,1)
  TIMERCANCEL(1)
  TIMERSTART(0,10,0)
ENDFUNC 1
'//----------------------------------------------------------------------
ONEVENT EVTMR0 CALL FuncTimer0
ONEVENT EVTMR1 CALL FuncTimer1
ONEVENT EVGPIOCHAN0 CALL Btn0Press
ONEVENT EVGPIOCHAN1 CALL Btn1Press
'//-----------------------------------------------------------------------
rc = GpioSetFunc(2,1,2)
rc = GpioBindEvent(0,2,1)
// init all GPIO at value Low
rc = GpioSetFunc(3,2,0)    // pin 3
rc = GpioSetFunc(8,2,0)    // pin 8
rc = GpioSetFunc(9,2,0)    // pin 9
rc = GpioSetFunc(10,2,0)   // pin 10
rc = GpioSetFunc(11,2,0)   // pin 11
rc = GpioSetFunc(12,2,0)   // pin 12
led = 0
TIMERSTART(0,10,0)
//led = 6
//TIMERSTART(1,10,0)
WAITEVENT
```

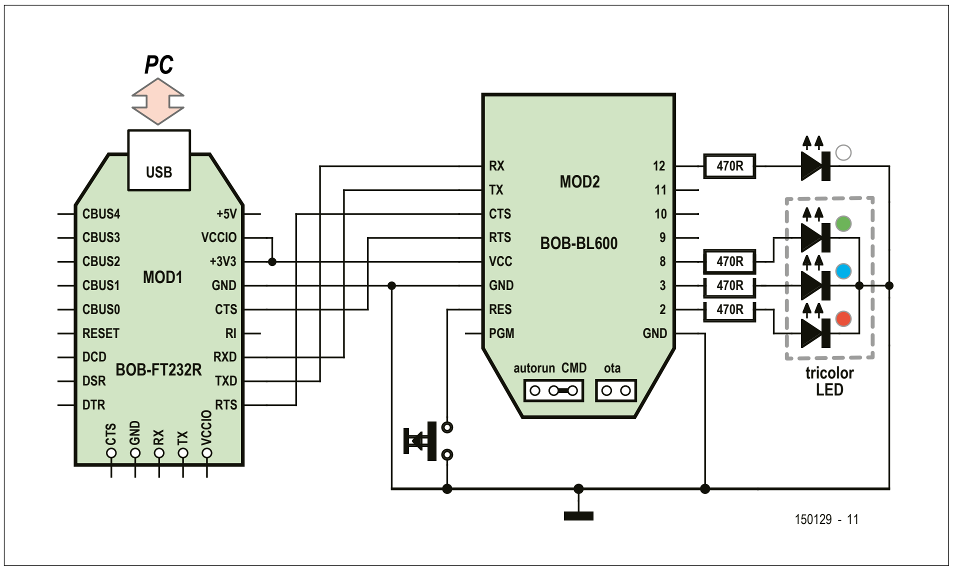


Figure 1. Experimental circuit for wireless control of a 3-color LED. Using the BL600 module, you can turn the LED on and off and choose the color using a smartphone.

Component List

(LED RGB control)

Resistors
R1–R4 = 470Ω

Semiconductors
D1 = LED, 3mm (select color)
D2 = LED, RGB, common cathode

Miscellaneous
K1 = pushbutton
MOD1 = assembled FT232 e-BoB module, # 110553-91 (www.elektor.com)
MOD2 = assembled BL600 e-BoB module, # 140270-91 (www.elektor.com)

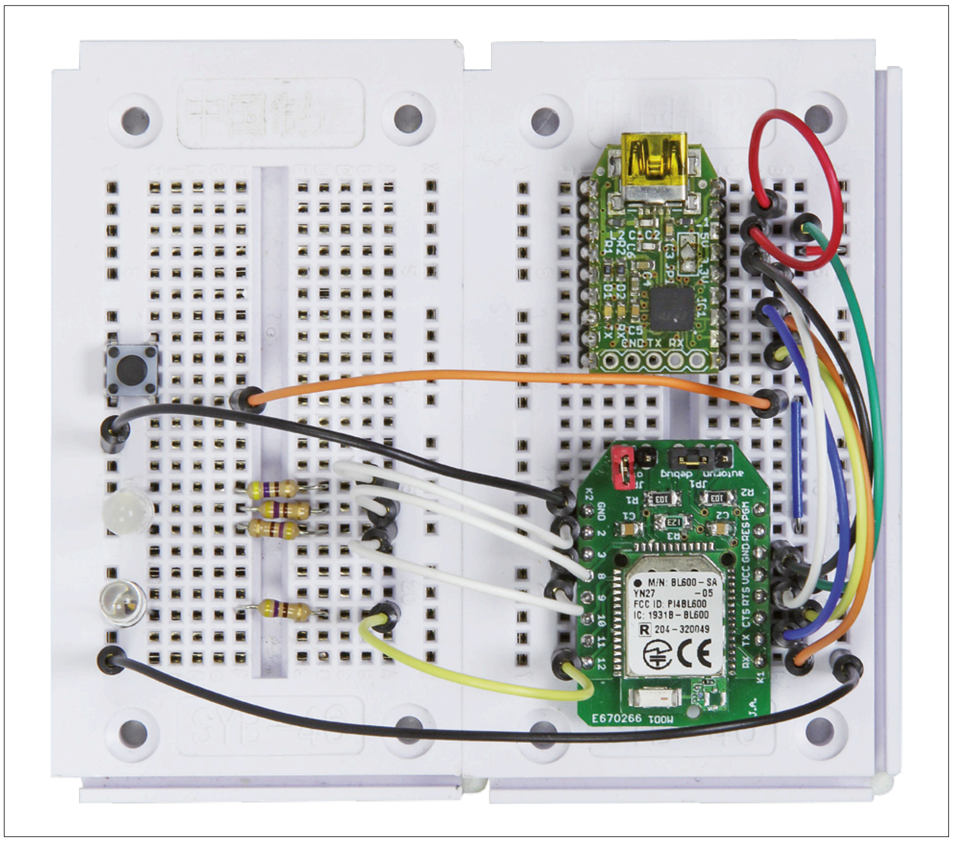


Figure 2. The circuit is easy to build on a solderless prototyping board.

```
TIMERSTART(0,10,0)
//led = 6
//TIMERSTART(1,10,0)
```

and take a look at the code in green in **Listing 1**.

When the button connected to pin 2 is pressed, an event `EVGPIOCHAN0` occurs, where `EVGPIOCHAN` represents "change of state on one of the module inputs", while 0 is the event number (chosen by us). This event is handled in the `Btn0Press` function via the instruction `ONEVENT EVGPIOCHAN1 CALL Btn1Press`. After pin 2 has been declared as an input by `GpioSetFunc(2,1,2)`, `GpioBindEvent(0,2,1)` establishes for this pin a link between the event and a transition (see below).

Declaring pin 2 as an input

`rc = GpioSetFunc(2,1,2)`
`nSigNum = 2:` pin GPIO 2
`nFunction = 1:` port as input
`nSubFunc = 2:` internal pull-up resistor
"rc" is the code returned by the function, which is 0x0000 if everything goes according to plan.

Declaring a link for an event to an input level transition

`rc = GpioBindEvent(0,2,1)`
`nEventNum = 0:` event number: **`EVGPIOCHAN0`** (the zero)
`nSigNum = 2:` pin GPIO 2
`nPolarity = 1:`0 for a Low-to-High transition
1 for a High-to-Low transition
2 for a Low-to-High or High-to-Low transition
"rc" is the code returned by the function, which is 0 if the function has not encountered any problems.

When the button is pressed, the TIMERSTART function (first parameter: **Event 1**) runs the chaser code (in blue) from output 12 to output 3. The reverse happens when the button is released, the TIMERSTART function (first parameter: **Event 0**) runs the chaser code (in mauve) from output 3 to output 12.

Three-color LED

Now you know how the events are handled, it's time to get the BL600 e-BoB to communicate with your phone via Bluetooth. The module is going to receive the phone data, via Bluetooth Low Energy, in order to light up the color(s) of the 3-color LED in the circuit in **Figure 1**. Just like the light-chaser described last month, we're going to build this new circuit on a prototype board (**Figure 2**).

We're not going to dwell on the hardware, but will describe the basis of the program, the interception of the data, changing the colors of the LED, and the connection status.

The basis of the program

Let's start by preparing a tidy environment for your first program. You'll need to:
Copy the smartBASIC_Sample_Apps directory into your working directory, rename it (e.g. MyProjectBL600), open this directory, delete everything except the `lib` directory, `upass.vsp.sb` (an example used as the basis for our program), `UwTerminal.exe` (UART terminal software to let us compile and transfer to the module), *XComp_BL600r2_8CF9_450E.exe* (compiler specific to the module version). Then you need to rename `upass.vsp.sb` as `pgmRGB.sb` (this will be saved with the name `pgmRGB-step0.sb` in the Elektor file. You'll find the files for all the steps in this article on our website). Then you need to compile, transfer, and run the whole thing on your BL600 e-BoB as described on page 64 in last month's article.

You may recognize this screen from our UART (**Figure 3**). You can do a test again using the `Serial` application from Laird Technologies downloaded from Google Play [3] as described in the article mentioned [2]

Intercepting the data

We have a simple program, let's modify it so as to intercept the data arriving via Bluetooth from the phone. This is going to be much easier than you might fear, as we're using a library to do the work for us. We count the number of characters in order to determine the length of the string that has arrived at our module via Bluetooth and display it using the PRINT command in the UWTerminal application that has been kept open on the PC.

`pgmRGB.sb` file

Not much to it except the declaration of the variables! The program makes use of the `cli.upass.vsp.sblib` library. This version is saved with the name `pgmRGB-step1.sb` in the file that can be downloaded from the Elektor Magazine website [4].

cli.upass.vsp.sblib library

We're not going to study this file in detail, but we are going to take a moment to look at handlers and the HandlerLoop function. The data arriving at the module's UART port or arriving at the module via Bluetooth are handled by the same handler. We suggest copying these four handlers and the associated function into our `pgmRGB.sb` program.
To avoid duplicates that would cause a compilation error, let's rename our function **My**HandlerLoop. You don't need to execute this version — all you need do is verify your code by compiling it (Xcompile option).

```
function MyHandlerLoop()
  BleVspUartBridge()
endfunc 1//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
```

`pgmRGB-step2.sb` in download [4]

The `BleVspUartBridge` function sets up a loop: the phone data are sent back to the phone.

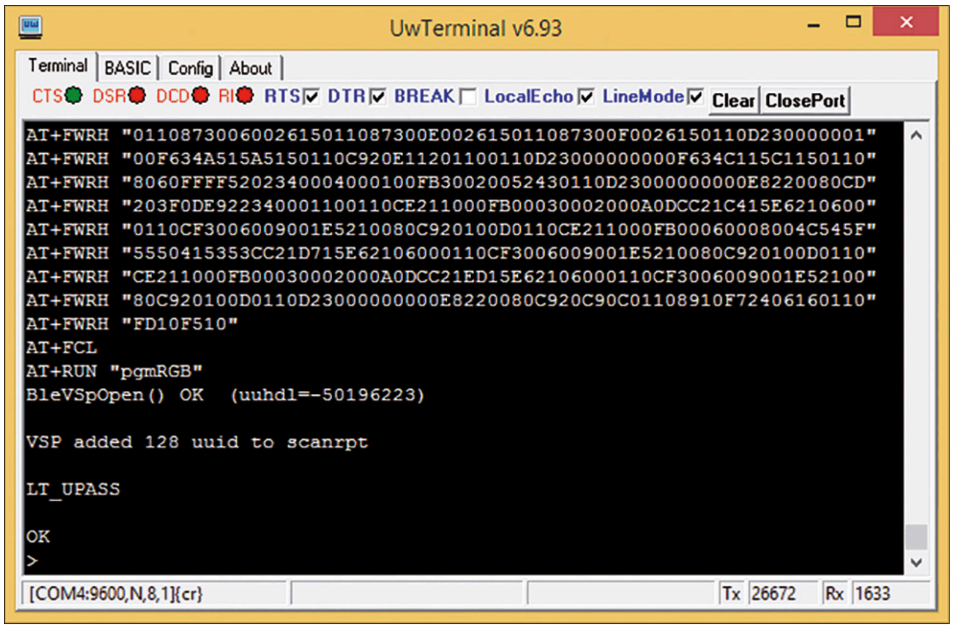
Length of received data

In order to read the phone data, we're going to replace the `BleVspUartBridge` function by `BleVspRead`:
`n = BleVspRead(tempo$,20)`
`strMsg` = tempo$: receive buffer
`nMaxRead` = 20: number of data to be read (max. 20)
`n` = length of receive buffer

```
function MyHandlerLoop()
   DIM n, rc, tempo$
  tempo$ = ""
  n = BleVSpRead(tempo$,20)
  IF (n > 0) THEN
   PRINT n;" data receive\n"
  ENDIF
endfunc 1
```

`pgmRGB-step3.sb` in download [4]

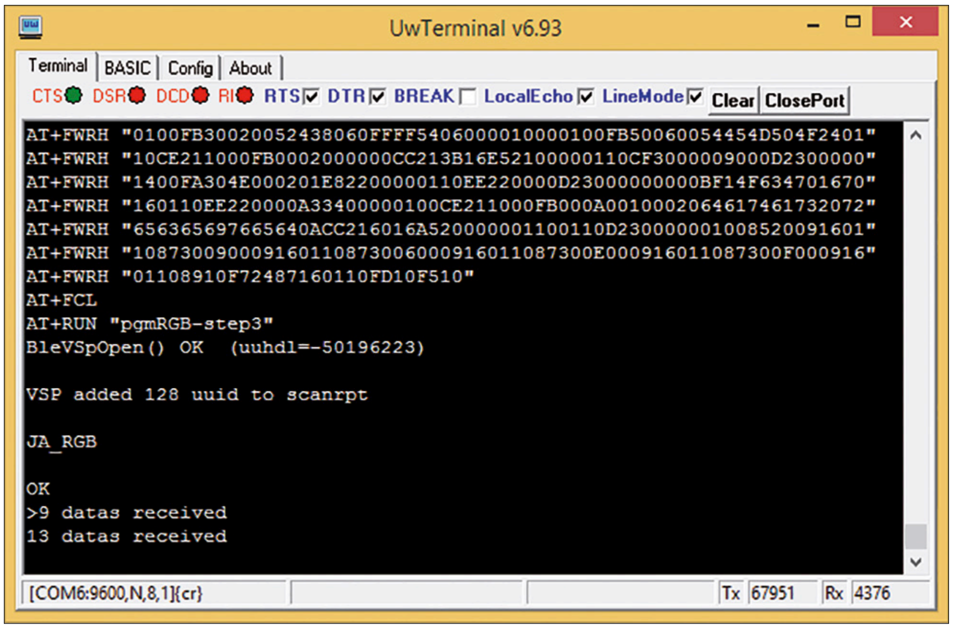
You can use your phone and the `Serial` application to send the data to the module. The `UWTerminal` application displays the number of characters sent (**Figure 4**), plus the end-of-line character (carriage return).



```
AT+FWRH "01108730060026150110873000E002615011087300F0026150110D230000001"
AT+FWRH "00F634A515A5150110C920E11201100110D23000000000F634C115C1150110"
AT+FWRH "8060FFFF5202340004000100FB30020052430110D23000000000E8220080CD"
AT+FWRH "203F0DE922340001100110CE211000FB00030002000A0DCC21C415E6210600"
AT+FWRH "0110CF3006009001E5210080C920100D0110CE211000FB00060008004C545F"
AT+FWRH "5550415353CC21D715E62106000110CF3006009001E5210080C920100D0110"
AT+FWRH "CE211000FB00030002000A0DCC21ED15E62106000110CF3006009001E52100"
AT+FWRH "80C920100D0110D23000000000E8220080C920C90C01108910F72406160110"
AT+FWRH "FD10F510"
AT+FCL
AT+RUN "pgmRGB"
BleVSpOpen() OK   (uuhdl=-50196223)

VSP added 128 uuid to scanrpt

LT_UPASS

OK
>
```

[COM4:9600,N,8,1]{cr}   Tx 26672   Rx 1633

Figure 3. Message from the UART prior to our modifications.



```
AT+FWRH "0100FB300200524380600FFFF5406000010000100FB50060054454D504F2401"
AT+FWRH "10CE211000FB0002000000CC213B16E52100000110CF3000009000D2300000"
AT+FWRH "1400FA304E000201E822000000110EE220000D2300000000BF14F634701670"
AT+FWRH "160110EE220000A33400000100CE211000FB000A00100020646174617 32072"
AT+FWRH "6563656976656 40ACC216016A5200000001100110D230000000100852009 1601"
AT+FWRH "1087300900091601108730060009160110873 00E0009160110873 00F000916"
AT+FWRH "01108910F72487160110FD10F510"
AT+FCL
AT+RUN "pgmRGB-step3"
BleVSpOpen() OK  (uuhdl=-50196223)

VSP added 128 uuid to scanrpt

JA_RGB

OK
>9 datas received
13 datas received
```

```
[COM6:9600,N,8,1]{cr}        Tx 67951   Rx 4376
```

Figure 4. Display of the number of characters sent by the phone.

Processing the received data: if the character R is received, the color will be red; if it's G, the color will be green; and if it's B, the color will be blue. The character string has no order, position, or length. Here's what happens in `MyHandlerLoop` e.g. for processing the color green.

```
tx$ = "G"
pos = STRPOS(text$,tx$,0)
DbgMsgVal("G :",pos)
IF ( pos >=0 ) THEN
GpioWrite(8,1)
ENDIF
```

RGB LED colors

We know how to intercept the data received from the phone; now let's process this information in order to turn our 3-color LED on or off.

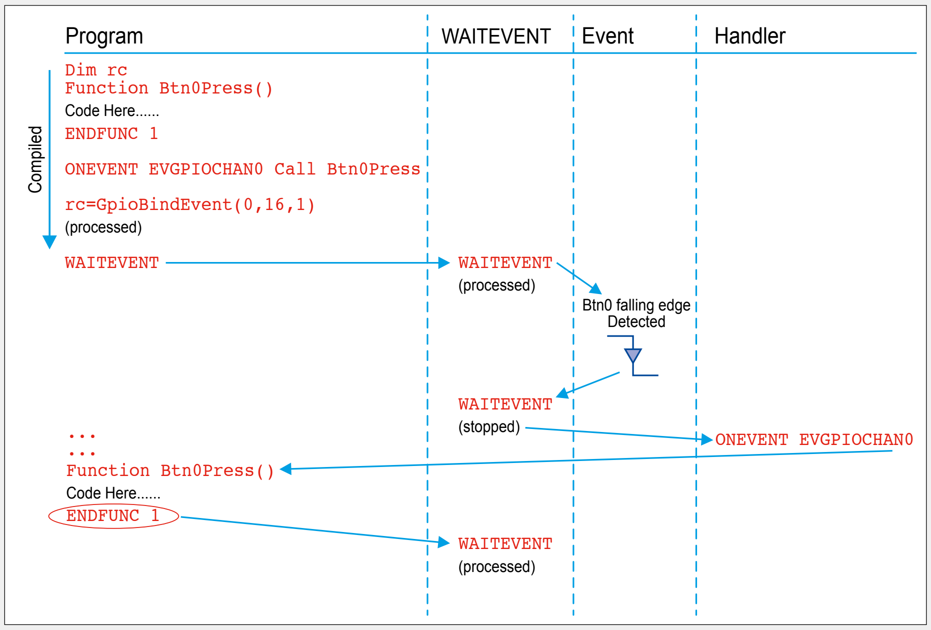
Output ports:
in the `main` section, using the `GpioSetFunc` function described in the previous article, we configure ports 2, 3 and 8 as active-low outputs.

To avoid the Bluetooth (advertising) loop timing out, we add into our program (the `MyBlrAdvTimOut` handler) the lines – see end of the article.
In the main program:
`rc = bleadvertstart(0,Adr$,25,0,0)`

And in the list of handlers:
`OnEvent EVBLE_ADV_TIMEOUT call MyBlrAdvTimOut // TimeOut`
Watch out, you'll need to rename the handler (e.g. My…)

events and handlers in smartBASIC

smartBASIC revolves around sequences of events that are handled in turn. The `WAITEVENT` function makes it possible to wait for events to arrive. If an event is detected during `WAITEVENT`, the runtime engine checks if there is a specific handler for this event. If an event is detected during `WAITEVENT`, the runtime engine checks if there is a specific handler for this event. If yes, the runtime engine calls the function associated with this event handler. At the end of handling the function, a code is returned. If it is 1, `WAITEVENT` starts again waiting for a new event.



For example, in this program, the event `EVGPIOCHAN0` is triggered by the falling edge produced by the button `Btn0`, and associated with the `Btn0Press` function or handler with the help of the instruction `ONEVENT … CALL ….` The `Btn0Press` function is called *IF and only IF* `WAITEVENT` is running. As soon as the falling edge is detected, `WAITEVENT` proper stops, while the handler manager starts working. When the `Btn0Press` function has ended correctly (`ENDFUNC 1`), `WAITEVENT` starts up again.

In the functions:

```
//============================
// This handler is called when
  there is an advert timeout
//============================
function MyBlrAdvTimOut() as
  integer
if AdvMngrOnAdvTimeOut() == 0
  then
  DbgMsg( "\nAdvert stopped via
  timeout" )
  dim Adr$
        Adr$=""
  rc =
  bleadvertstart(0,Adr$,25,0,0)
   endif
endfunc 1
```

`pgmRGB-step4.sb` in download [4]

Using the `Serial` application from Laird Technologies, you can send orders like: R--, RGB, ---, GB-, and so on.

You can download my BLE RGB Lite program on Google Play [3]. The source code for this program (**Figure 5**) will be available on the Elektor site.

Connection status

A little bonus: we're going to light an LED on output 12 of our module when it is connected; this will be turned off when the module is disconnected. Don't forget the function for initializing port 12 as an output in the main program – you know how to do that now.

We're going to copy the Bluetooth message handler from the `cli.manager.sblib` library and create our own handler, like this:

In the list of global variables:

```
'//*******************************
'// Global Variable Declarations
'//*******************************
dim hConnLast
```

In the list of handlers:
(Watch out, you'll need to rename the handler, e.g. My...)

```
OnEvent  EVBLEMSG           call
MyHandlerBleMsg
```

We'll add the MyHandlerBleMsg function. When the message concerns a connection, we turn our LED on and when it involves a disconnection, we turn our LED off (code in red). Nothing very complicated:

```
function MyHandlerBleMsg(BYVAL
nMsgId AS INTEGER, BYVAL nCtx
AS INTEGER) as integer
  …… code here …..
  select nMsgId
    case BLE_EVBLEMSGID_CONNECT
      DbgMsgVal(" --- Connect :
",nCtx)
      GpioWrite(12,1)
      hConnLast = nCtx
      ShowConnParms(nCtx)
    case
BLE_EVBLEMSGID_DISCONNECT
      DbgMsgVal(" --- Disconnect
: ",nCtx)
      GpioWrite(12,0)
…… code here …..
```

`pgmRGB.sb` in download [5]

MyBlrAdvTimOut

The purpose of this handler is to re-launch the possibility for connecting to our module in Bluetooth following a timeout. To do this, we're going to copy the default handler from the `cli.manager.sblib` library and create our own `MyHandlerBlrAdvTimOut` handler. We've designed it to re-launch the advertising, i.e. the Bluetooth, via the following code:

```
rc = bleadvertstart(0,Adr$,25,0,0)
```

Acknowledgements:
Jennifer Gibbs (Laird Technologies); Philippe

(150129)

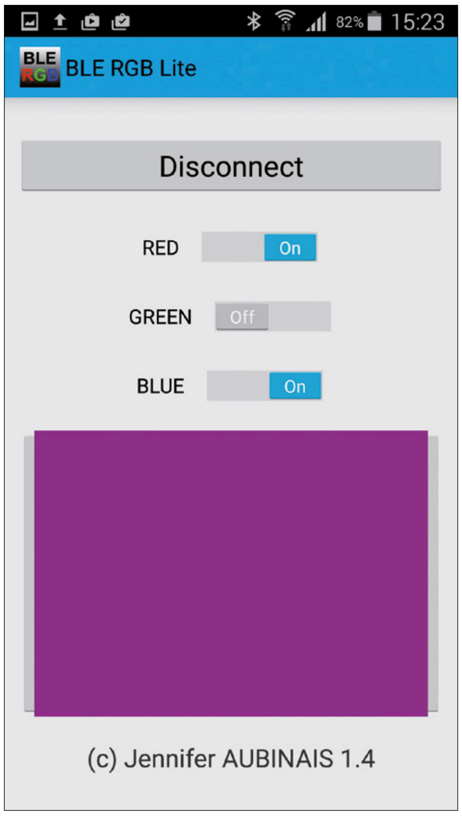


Figure 5. Screen from the BLE RGB application available from Google Play.

Selection of topics

to be covered in future episodes of this series on the BL600 e-Bob:

- Low Energy, 5 µA
- the I²C | SPI ports
- Bluetooth communication
- explanation of the remote wireless thermometer program
- writing a program for Android
- writing a program for iOS

Weblinks:

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

[2] e-BoB BL600 | Elektor no. 442, March 2015, p. 64
www.elektor.com/150014

[3] https://play.google.com/

[4] www.elektor-magazine.com/150129

[5] e-BoB BL600 | Elektor no. 441, March 2015, p. 34
www.elektor-magazine.com/140270

[6] Bluetooth LE Wireless Thermometer
Elektor nos. 439–440, Jan. & Feb. 2015, p. 72
www.elektor-magazine.com/140190