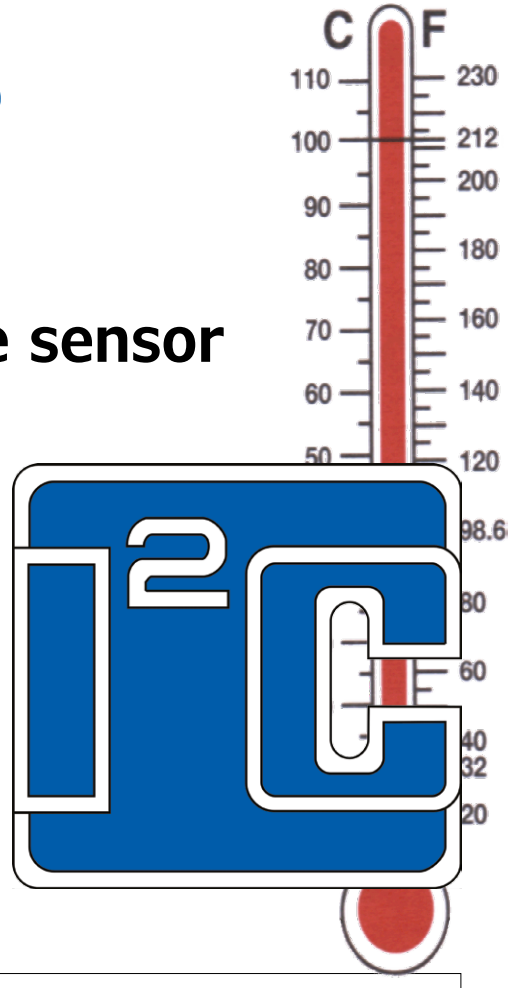
BL600 e-BoB

Part 4

The I²C port and the temperature sensor



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

Our little wireless communications module has only seven input/output ports, but its I²C port gives it considerable possibilities for extension. We can use it for communication with various components such as temperature sensors, D-A and A-D converters, etc. Here we will use it for exchanging data, the new service Health Thermometer via Bluetooth Low Energy instead of the UART service used in previous installments.

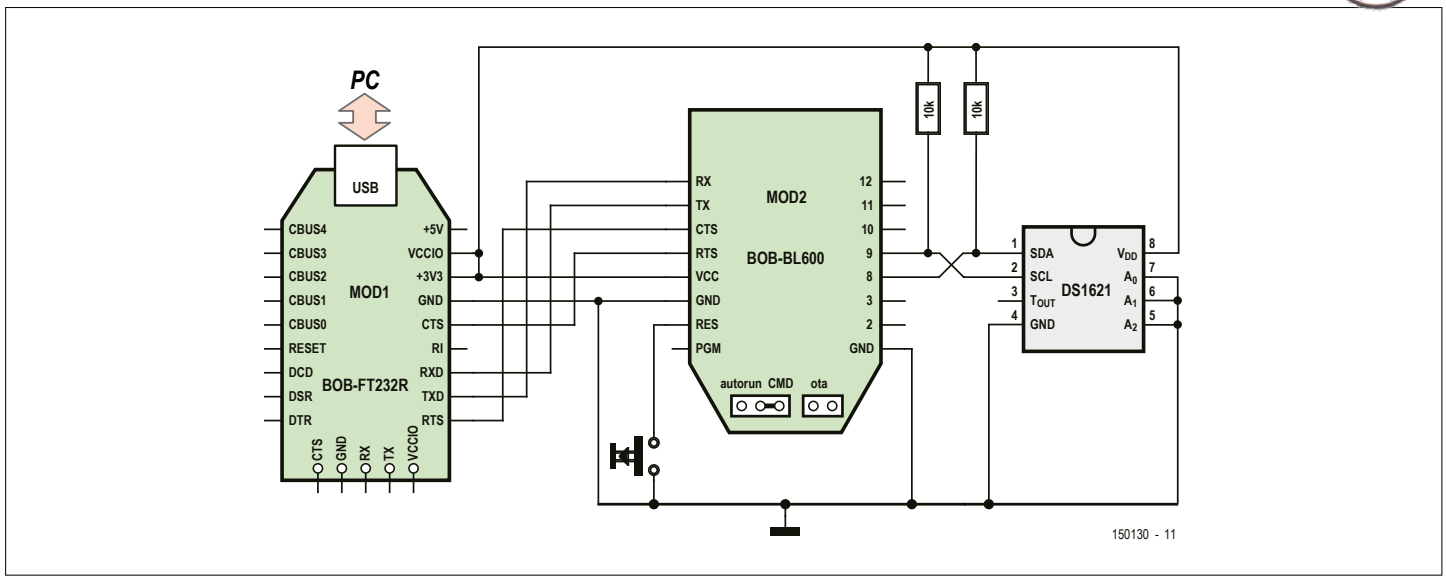


Figure 1. Schematic of a wireless thermometer with a DS1621 temperature sensor which communicates with the BL600 by I²C. The implementation of the e-BoB FT232 has been described in previous installments.

We have learned how to use events in smartBASIC in the previous issue. Now we can start with the I²C port of the BL600 by connecting it with the DS1621 temperature sensor [1] from Maxim Integrated.

It may seem surprising to communicate via I²C with a Bluetooth module… well, "you ain't seen nothin' yet", because this module has several more amazing resources, which will be the subject of future articles. For the moment, we will measure the temperature and display it on a smartphone, either under Android in an Android application which may be downloaded from Google Play [2], or under iOS with an application which may be downloaded from the Apple Store [7] or a small program which can be downloaded from the Elektor site [6].

Listing 1

```
DIM rc, handle, txt$
rc=I2cOpen(100000,0,handle)
IF rc!= 0 THEN
  SPRINT #txt$,INTEGER.H'rc
  DbgMsg("Failed to open I2C interface with error
code 0x" + Right$(txt$,4))
  PRINT "\nDO RESET"
  STOP
ELSE
  DbgMsgVal("\nI2C open success \nHandle is
",handle)
ENDIF
```

Listing 2

```
DIM x, nSlaveAddr, nRegAddr, nRegVal, txt2$
nSlaveAddr = 0x48 :  nRegAddr = 0xAC : nRegVal =
  0xAA
rc = I2cWriteReg8(nSlaveAddr, nRegAddr, nRegVal)
IF rc!= 0 THEN
  SPRINT #txt$,INTEGER.H'rc
  DbgMsg("Failed to Write to slave/register " +
  Right$(txt$,4))
ELSE
  SPRINT #txt$,INTEGER.H'nRegVal
  SPRINT #txt2$,INTEGER.H'nRegAddr
  DbgMsg("0x" + Right$(txt$,2) + " written
  successfully to register 0x" + Right$(txt2$,2))
ENDIF
```

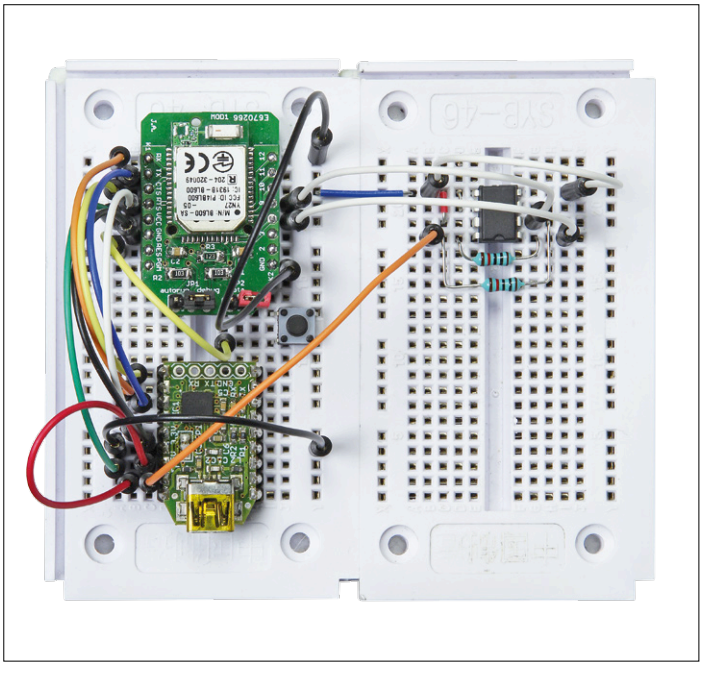


Figure 2. Photo of the whole assembly on a breadboard.

Note: the use of the tools and commands to which I refer here is covered in previous instalments of this series [3] to which the reader may refer.

I²C port and temperature sensor

The DS1621 temperature sensor is an ideal illustration of the read and write functions of the I²C port: the port SDA (data) on pin 8 of the e-BoB BL600 is connected to pin 1 of the DS1621; the port SCK (clock) on pin 9 of the e-BoB is connected to pin 2 of the DS1621 (**Figure 1**). Laird Technologies confirm that pullup resistors (between 4.7 and 10 kΩ) must be used on the I²C bus lines. Following the documentation of the DS1621, we will send it some commands and then read its different registers in order to calculate the temperature. The application Temperature for Android issued by Laird Technologies allows us to trace a graph of the temperature measured by the sensor. The application which we give you for use with iOS, which uses the service Health Thermometer, allows us to do the same thing on an Apple product.

a. Open I²C

The first step is the opening of the I²C port, followed by the processing of the returned code by a simple *if* statement (**Listing 1**).

```
Rc = I2COpen(100000,0,handle)
```

- nClockHz = 100000; clock frequency
- nCfgFlags = 0; must be set to 0
- nHandle = if the returned code is 0, this variable is further used to read, write and close the I²C interface

rc = the returned code must be 0 to continue. If not, we convert its value to a chain of characters using the SPRINT function with the INTEGER.H option to convert to hexadecimal, then display the error code (see doc. Laird Technologies), before stopping the program using STOP.

b. Write a byte to I²C

The next step is to write a byte to the I²C port, to be sent to the DS1621. We have chosen the Access Config register (0xAC), which is read/write, and as an example we will write to it the value 0xAA (**Listing 2**).

```
Rc = I2CWriteReg8(nSlaveAddr,nRedAddr,nRegVal)
```

- nSlaveAddr = slave address of the component, between 0 and 127
- nRedAddr = register address of the component
- nRedVal = value (8 bits) to write to the address of the desired register

Component List

Resistors
R1,R2 = 10kΩ

Semiconductors
IC1 = DS1621

Miscellaneous
K1 = pushbutton connection
MOD1 = FT232e-BoB, assembled, # 110553-91 (www.elektor.com)
MOD2 = BL600e-BoB, assembled, # 140270-91 (www.elektor.com)

Listing 3

```
nSlaveAddr = 0x48 :  nRegAddr = 0xAA
rc = I2cReadReg8(nSlaveAddr, nRegAddr, nRegVal)
IF rc!= 0 THEN
  SPRINT #txt$,INTEGER.H'rc
  DbgMsg("Failed to Read from slave/register " +
  Right$(txt$,4))
ELSE
  SPRINT #txt$,INTEGER.H'nRegVal
  SPRINT #txt2$,INTEGER.H'nRegAddr
  DbgMsg("Value read from register 0x" +
  Right$(txt2$,2) + " is 0x" + Right$(txt$,2))
ENDIF
```

rc = the returned code must be 0 to continue. If not, we convert its value to a chain of characters, then display the error code (see Laird Technologies' documentation.)

c. Read a byte from I²C

We read the register 0xAC which is the register of the high byte of the temperature of the DS1621. As we have not done the initialization procedure of the DS1621, it will not contain a usable value (**Listing 3**).

```
Rc = I2CReadReg8(nSlaveAddr,nRedAddr,nRegVal)
```

- `nSlaveAddr` = slave address of the component, between 0 and 127
- `nRedAddr` = register address of the component
- `nRedVal` = value (8 bits) read from the address of the desired register

rc = the returned code must be 0 to continue. If not, we convert its value to a chain of characters, then display the error code (see doc. Laird Technologies)

d. Close I²C

The last step is to close the I²C port. Laird Technologies recommend doing it twice (**Listing 4**).

```
I2Close(handle)
```

- **nHandle** = value created by `I2cOpen`

The file `demoI2C.sb` may be downloaded from the Elektor site [6].

e. Read the DS1621 using the serial port of the e-BoB FT232

We will not discuss the DS1621 [1] in detail, but we have made available for you a complete program for using it, notably using the functions `I2COpen`, `I2CWriteReg8`, `I2CReadReg8` and `I2CClose` described above. The datasheet of the temperature sensor suggests the following equation to transform the measured data of the DS1621 to a temperature value:

$$TEMPERATURE = TEMP\_READ - 0.25 + \frac{(COUNT\_PER\_C - COUNT\_REMAIN)}{COUNT\_PER\_C}$$

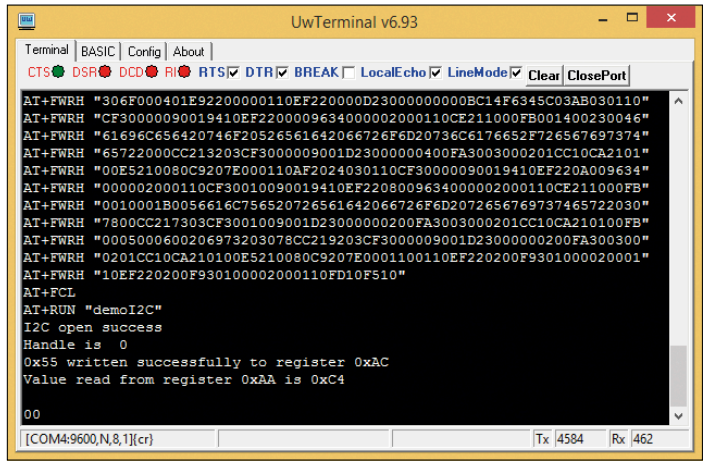


Figure 3. Writing to, and then reading from, the I²C port passing correctly.

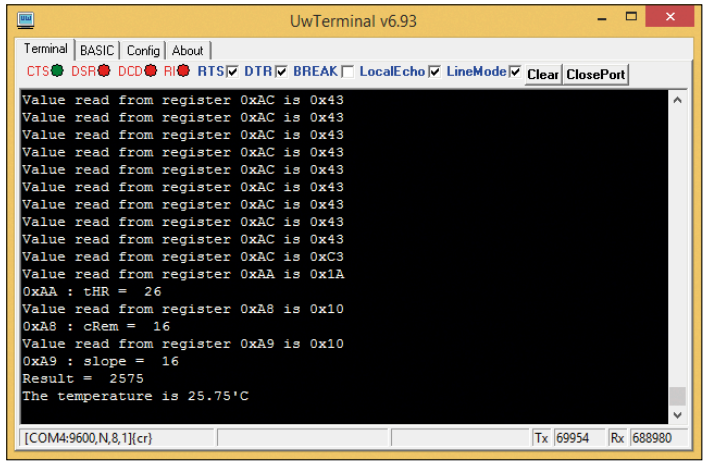


Figure 4. Display of the temperature in degrees: after bit 7 of the configuration register 0xAC goes to 1 (0x43 -> 0xC3), we can read the temperature register and do the calculations.

Listing 4

```
I2cClose(handle)   //close the port
I2cClose(handle)   //no harm done doing it again
```

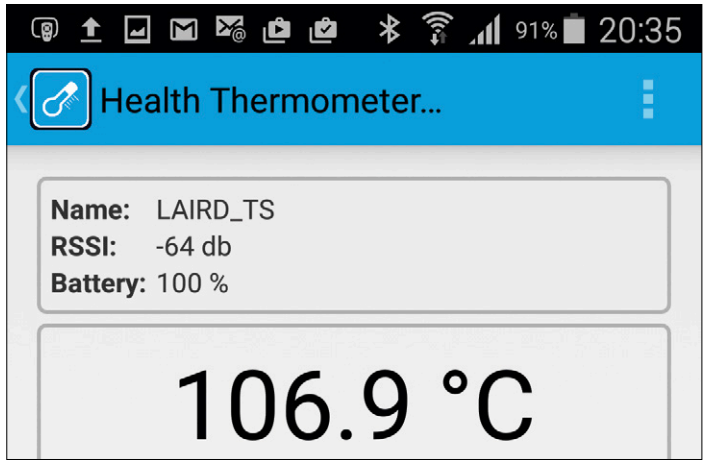


Figure 5. Don't panic, it wasn't 106.9 °C at the author's house or at Elektor Labs!

Listing 5

```
FUNCTION InitI2C()
DIM rc, txt$ // DIM Handle is defined at top of library
DIM rc, x, a, tC, flag, handle, txt$
  rc=I2cOpen(100000,0,handle)
  IF rc!= 0 THEN
    SPRINT #txt$,INTEGER.H'rc
    DbgMsg("Failed to open I2C interface with error
  code 0x" + Right$(txt$,4))
    PRINT "\nDO RESET"
    STOP
  ELSE
    DbgMsgVal("\nI2C open success \nHandle is
  ",handle)
  ENDIF
ENDFUNC rc
'-----------------------------------------------------
FUNCTION InitDS1621()
  DIM rc
  // Tout = active high; 1-shot mode
  DbgMsg("SetConfig")
  rc = SetConfig(POL | ONE_SHOT)
ENDFUNC rc
'-----------------------------------------------------
FUNCTION ConvertDS1621()
  DIM rc, tC
  DbgMsg("DS1621")
  DO
  DbgMsg("\nStartConversion")
  rc = StartConversion() // initiate conversion
  tC = GetHrTemp() // read high-resolution temperature
ENDFUNC tC
'-----------------------------------------------------
FUNCTION ConvertTempTxt$(tC)
  DIM rc, x, a, flag, txt$
  flag = 0
  IF (tC < 0) THEN
    tC = -tC   // fix for integer division
    flag = 1   // indicate negative
  ENDIF
  SPRINT #txt$,tC
  SELECT  StrLen(txt$)
    CASE 1
      txt$ = "0.0" + txt$
    CASE 2
      txt$ = "0." + txt$
    CASE 3
      txt$ = Left$(txt$,1) + "." + Right$(txt$,2)
    CASE 4
      txt$ = Left$(txt$,2) + "." + Right$(txt$,2)
    CASE ELSE
  ENDSELECT
  IF (flag == 1) THEN
    txt$ = "-" + txt$
  ENDIF
  PRINT "The temperature is "+ txt$ + "'C\n"
  for x = 0 to 40000
  next
  DOWHILE (1)
ENDFUNC txt$
'-----------------------------------------------------
FUNCTION CloseI2C()
  I2cClose(handle)  //close the port
  I2cClose(handle)  //no harm done doing it again
ENDFUNC 1
```

As the BL600 can only use integers (from 0 to 65535), all the values are multiplied by 100, after which the temperature will be in 100ths of a degree Celsius. This suits us well for displaying text in the form of a temperature in degrees: it is only necessary to correctly place the decimal point (**Figure 4**).

tHR = (tHR * 100 – 25) + ((slope – cRem) * 100 / slope)

The file DS1621.sb may be downloaded from the Elektor magazine website [6].

f. The DS1621 in HTS (Health Thermometer Service) mode

In my article *Bluetooth Low Energy Wireless Thermometer* published in the January & February 2015 edition of Elektor [4], it was the smartphone application which did the calculations. Here, it's the BL600 that does them. It's easy, thanks to Laird Technologies who offer software for our e-BoB BL600 and an application Temperature for Android. We thus have a wireless thermometer with the BL600 and the DS1621, which displays the temperature on your Android smartphone.

Use of the file from Laird Technologies

We simply compile the file htss.health.thermometer.sensor.sb. To verify that it works, we launch the application Temperature. You will notice that the temperature displayed is over 100 °C ! This is normal, as we have not connected an LM20 sensor on pin 4 of the BL600 as is expected with the development kit DVK-BL600-SA.

The file htss.health.thermometer.sensor.sb is in the software package you can download for the BL600 [5] (**Figure 5**)

Transformation of the program DS1621.sb to a library

First step: we copy our program DS1621.sb to DS1621.sblib. We are separating the main program (main) into several functions (**Listing 5**) which we will call in our new program.

Second step: we modify the program htss.health.thermometer.sensor.sb and rename it $autorun$.htss.ds1621.sb. Here are the modifications:

- Modify DEVICENAME: JA_HTS
- Add the library that we have just created: ds1621.sblib
- Delete the following functions: Adc2mv, Mv2Temperature
- In the function HandlerTimer0, replace the code in red by the code in green in **Listing 6**

Communicate via I²C with a Bluetooth module? You ain't seen nothin' yet!

Listing 6

```
mv = Adc2Mv(GpioRead(4))
DbgMsg("\nAdc mV=")
PRINT mv
tmp = Mv2Temperature(mv)
tmp = ConvertDS1621()
tmp = tmp / 10
```

Listing 7

```
InitTempSensor()
rc = InitI2C()
rc = ConvertDS1621()
TimerStart(0,TEMPERATURE_POLL_MS,1)
```

- In the main program (`main`) replace the code in red by the code in green in **Listing 7**

Nothing complicated; in place of the original analog LM20 sensor we have simply substituted our DS1621. For starters, it's I²C and further, it has a higher resolution.
The file `$autorun$.htss.ds1621.sb` may be downloaded from the Elektor website [6].

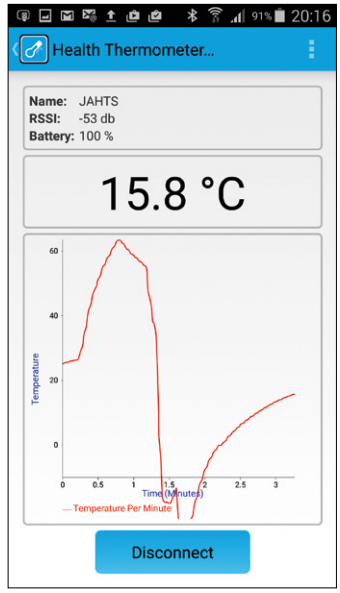


Figure 6. Graph obtained with the DS1621 connected via I²C to our e-BoB BL600. You will notice the peak of over 60 °C (using a hair-dryer) and a dip to –20 °C (obtained with a Freezit aerosol). The DS1621 offers a measurement range of –55 °C to +125 °C.

Figure 7. Screen capture of the BLE HTS application on iPhone.

The program under Android

We now use the application HTM (Health Thermometer Per Minute), from the Laird Toolkit that we have already downloaded. After the scan, we choose the DEVICENAME JA_HTS given to our e-BoB. A red graph line appears; the author amused herself well here — guess how? (answers in the caption of **Figure 6.**) Amazing, huh?

The iOS program

After this example of the use of the HTS service with Android, instead of the Bluetooth Low Energy module's UART service, used since the beginning of this series, we also offer licensed iOS developers the source code of an equivalent small program for Apple (**Figure 7**). This file `BLE HTS.zip` can be downloaded from the Elektor website [6]. On the website of Laird Technologies [5], you can find the complete source code for their application `Toolkit` for iOS comprising the services UART, HTS….

Call for contributions

In this series of articles, we have called upon the program Serial from the Toolkit from Laird Technologies. This has made our lives much easier, but isn't it now time to make your own application for your Android phone? From the next article we will offer you a source code, the simplest possible, which will allow you to increase your knowledge of the e-BoB BL600. Thanks to this module, available ready-to-use from the Elektor Store [8], implementation of Bluetooth communication is easy. Elektor is ready to publish other applications. If you have any ideas for using this module, or designs or projects in development, we invite you to share them with us, the author, and the Elektor community on the website www.elektor-labs.com. Your project might be chosen to appear in a forthcoming edition of Elektor magazine or the Elektor.POST newsletter.

(150130)

Web Links

[1] www.maximintegrated.com/en/products/analog/sensors-and-sensor-interface/DS1621.html
[2] https://play.google.com
[3] Elektor edition 2/2015 (March & April)
[4] Elektor edition 1/2015 (January & February)
[5] https://laird-ews-support.desk.com/?b_id=1945
[6] www.elektormagazine.com/150130
[7] https://itunes.apple.com/en/app/bl600/id594855763?mt=8
[8] www.elektor.com/bl600-e-bob-140270-91