

BBC micro:bit for Electronicists (2)

Data acquisition and oscilloscope functions

By **Burkhard Kainka** (Germany)

Every microcontroller that features an A-to-D converter and a PC interface can be used as a logging device for data acquisition systems. But with the BBC micro:bit you get the bonus of a small LED display and a wirefree interface into the bargain. Just the job for special applications in your electronics lab!

The small footprint of the BBC micro:bit would alone make it a go-to choice for measurement tasks. Regardless of whether powered over a USB cable or by batteries, and whether it used Bluetooth or some simplified wireless protocol to communicate with the outside world, it can always be installed close to the object under examination.

A USB oscilloscope

For programming the BBC micro:bit the mbed platform has proven its worth. The

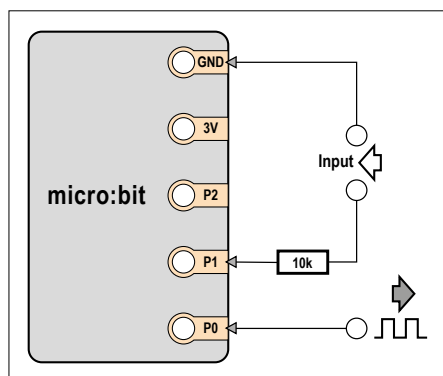
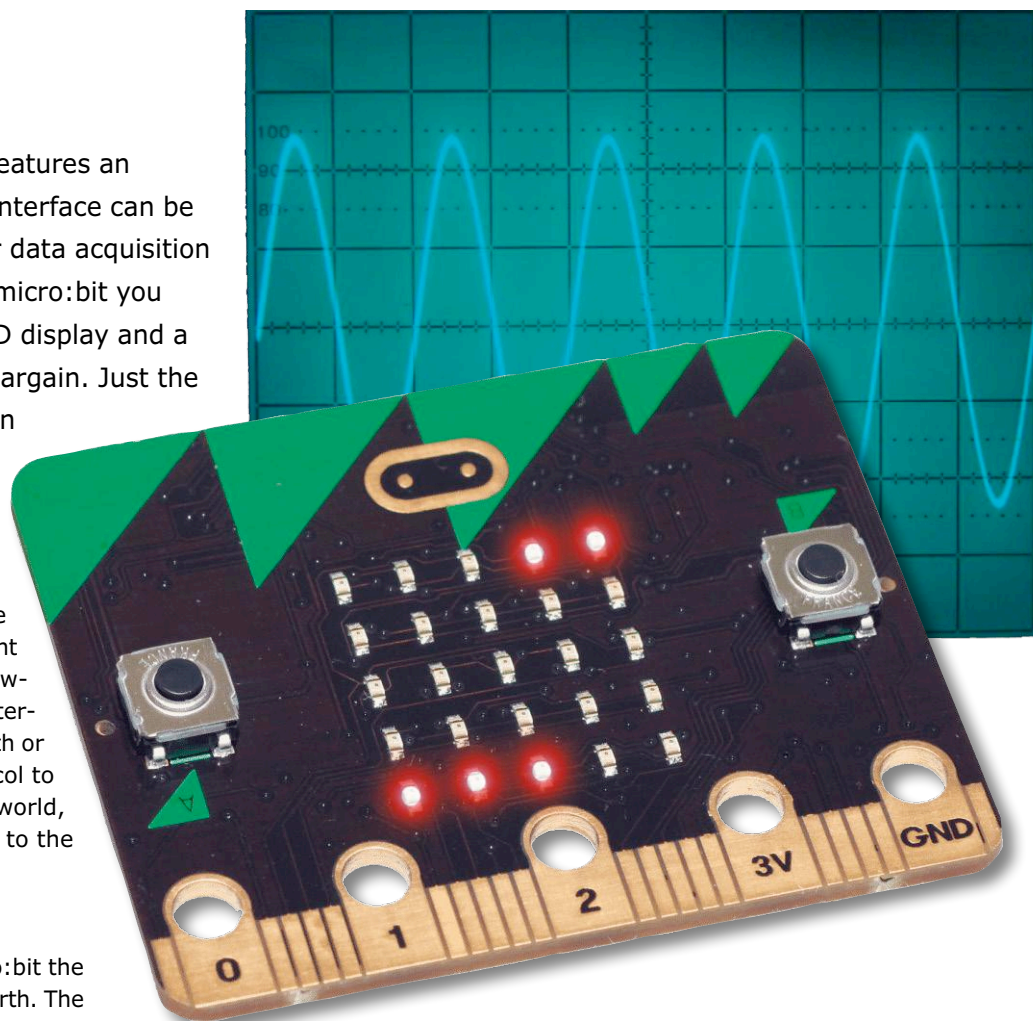


Figure 1. Measurement input and signal output.

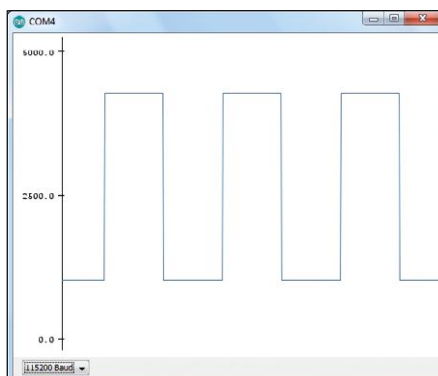


Figure 2. A squarewave signal with 10 Hz repetition rate.

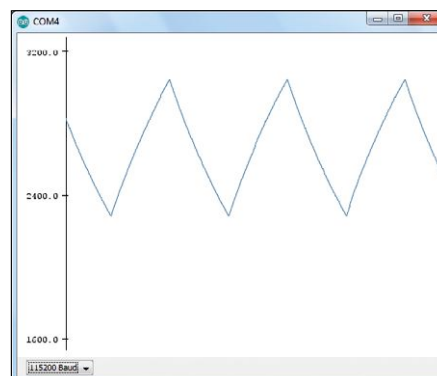


Figure 3. The filtered signal.

basics are explained in reference [1]. All the programs mentioned in this article are available to download as text files on the Elektor website [2] and need to be copied into an existing mbed project. When deploying the BBC micro:bit for general-purpose measurements you must ensure that the voltage range under examination cannot overstep the range between GND and V_{cc} . A protective resistor of 10 k Ω in series with the input will restrict the current flow in all situations, necessary if you accidentally exceed safe limits (**Figure 1**). As well as the analog input there's also a square-wave output, with which you can produce a handy test signal.

For maximum speed the program in **Listing 1** captures data without buffering for immediate transfer of each measured value. A significant element of the cycle time arises from the serial transfer at 115,200 baud. If the measurements captured are a mixture of, say, single-digit (3 mV) and four-digit (3000 mV) amounts, these variable figures will result in uneven data flow. Consequently we raise the voltage by 1000 mV, making the possible values from 1000 mV to 4300 mV and always taking the same time to process. The program also provides its own signal source so that you can measure something without additional overheads. P1 becomes a PWM output with a PWM frequency of 10 Hz and a period of 100 ms. For evaluating the data there are plenty of options. You could take in the data using a terminal program and then present it as a spreadsheet. A convenient alternative is the serial plotter in the Arduino IDE (version 1.6.8 onwards). This software provides a scrolling screen display, adjusted automatically to the display range, making range switching or reformatting unnecessary. With the settings shown we can measure a symmetrical squarewave signal with 10-Hz repetition rate (**Figure 2**). At the same time we now have a known time axis. The entire oscillogram clearly depicts a data acquisition duration of 300 ms. The serial plotter of the Arduino IDE always plots first from left to right until the screen is completely filled. After this, the picture scrolls to the left to make old data disappear. The choice between a continuous or static display is made with the button A (`if(uBit.buttonA.isPressed())` in Listing 1). Just press button A for the duration of the measure-

Listing 1. Rapid measurement with direct data transfer.

```
//Voltage Logger/Scope
#include "MicroBit.h"
MicroBit uBit;

int main()
{
    uBit.init();
    MicroBitSerial serial(USBTX, USBRX);
    uBit.io.P1.setAnalogValue(512);
    uBit.io.P1.setAnalogPeriodUs(100000);
    while (1) {
        if(uBit.buttonA.isPressed()){
            int u = 1000+3300 * uBit.io.P0.getAnalogValue()/ 1023;
            uBit.serial.printf("%d\r\n", u);
            // uBit.sleep(100);
        }
    }
}
```

ment and a scrolling display is shown. As soon as you release the button, the last display is frozen, so you can examine it more closely or save a copy of it.

The additional signal output can be useful for investigating circuits or components. Using a low-pass filter with 4.7 k Ω and 22 μ F a sawtooth signal is gener-

ated from the squarewave, as would be expected (**Figure 3**). The measurement range of the Arduino plotter adapts automatically to smaller voltages.

Faster sampling by buffering

If you are minded to reduce the time taken by serial data, it is only the acqui-

Listing 2. Rapid saving and subsequent transfer.

```
//Fast Scope
#include "MicroBit.h"
MicroBit uBit;

int main(){
    char d[400];
    uBit.init();
    MicroBitSerial serial(USBTX, USBRX);
    uBit.io.P1.setAnalogValue(512);
    uBit.io.P1.setAnalogPeriodUs(2000);
    while (1) {
        if(uBit.buttonA.isPressed()){
            for(int i = 0; i < 400; i++){
                d[i] = uBit.io.P0.getAnalogValue()/ 4;
            }
            for(int i = 0; i < 50; i++) uBit.serial.printf("%d\r\n", 0);
            for(int i = 0; i < 400; i++){
                uBit.serial.printf("%d\r\n", d[i]);
            }
            for(int i = 0; i < 50; i++) uBit.serial.printf("%d\r\n", 255);
        }
        uBit.sleep(500);
    }
}
```

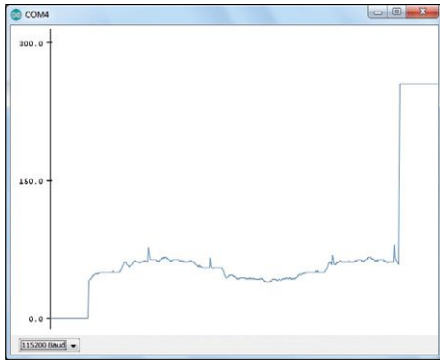


Figure 4. Measuring with a higher sampling rate.

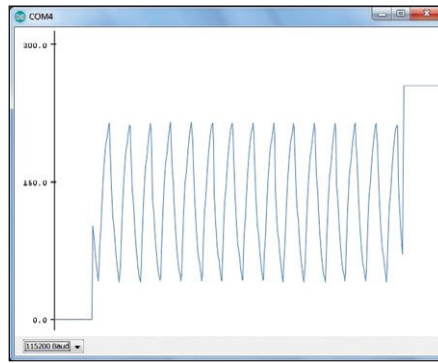


Figure 5. A 500-Hz sawtooth signal

sition time of the A-to-D converter that limits the achievable sampling rate. So you create a Data Array, fill it with data measurements and send these off to the PC. But it's exactly here that problems arise that you had not reckoned with. Although the controller is well endowed with RAM, an Array of the type `int d[100]` is pushing the limits, because the Microbit Runtime does not have much spare capacity. But if you want to use the serial plotter, there should already be 500 measured values.

You definitely need to be aware that the type of `int` in a 32-bit system has a size of four bytes. Accordingly we have 400 bytes at our disposal. Therefore we

use a Byte Array with 400 bytes using `char d[400]`. By dividing by 4, the 10-bit data of the A-to-D converter becomes an economical 8 bits.

We now store and transfer 400 bytes (**Listing 2**). We are still 100 bytes short for filling the plotter. But we can make a virtue out of necessity and prefix a zero-bytes header and suffix a trailer having 255-bytes. This causes the serial plotter to always display the full measurement range and provides the viewer with clear visibility of the range limits. The reading in **Figure 4** depicts a 50-Hz signal with typical interference pulses, which you pick up with an exposed measurement lead. Because around 1.5 oscillations are

shown, the measurement lasts around 30 ms, which means that with 400 data points a sampling rate of approximately 13 kHz can be deduced. Because the PWM frequency in this program was raised by 500 Hz, you can easily verify this with a signal of your own. **Figure 5** shows the PWM signal at the output a low-pass filter with 4.7 kΩ and 100 nF.

Wireless transfer of captured data

The BBC micro:bit is equipped with Bluetooth Low Energy (BLE). You will look in vain on the board for a dedicated chip for this, as the RF circuitry is already built into the microcontroller. The nRF51822 from Nordic Semiconductor was originally developed for applications such as wireless keyboards and mice, which did not call for extensive range but did have to use battery power economically. The BBC micro:bit takes advantage of these capabilities. You can power the board using a 3-V battery and then dispense with even the USB cable. This makes the system viable even for long-term applications powered by batteries. The shortform lineup of its main features speaks for itself:

- 2.4-GHz transceiver
- -93 dBm sensitivity in Bluetooth® low energy mode
- 250 kbps, 1 Mbps, 2 Mbps supported data rates
- Tx Power -20 to +4 dBm in 4-dB steps
- Tx Power -30 dBm whisper mode
- 13 mA peak Rx, 10.5 mA peak Tx (0 dBm)
- 9.7 mA peak Rx, 8 mA peak Tx (0 dBm) with DC/DC
- RSSI (1-dB resolution)
- ARM® Cortex™-M0 32-bit processor
- 275 μA/MHz running from flash memory
- 150 μA/MHz running from RAM
- Serial Wire Debug (SWD)

Programming with mbed enables the use of Bluetooth, allowing you to transmit data direct to a smartphone or tablet. Admittedly this calls for a pretty sizeable software stack and worse, it leaves you to develop the custom apps.

But there's a far simpler way. You see, you can address the 2.4-GHz transceiver at a lower level, dispensing altogether with the complicated Bluetooth protocol. Making this possible is the MicroBitRadio

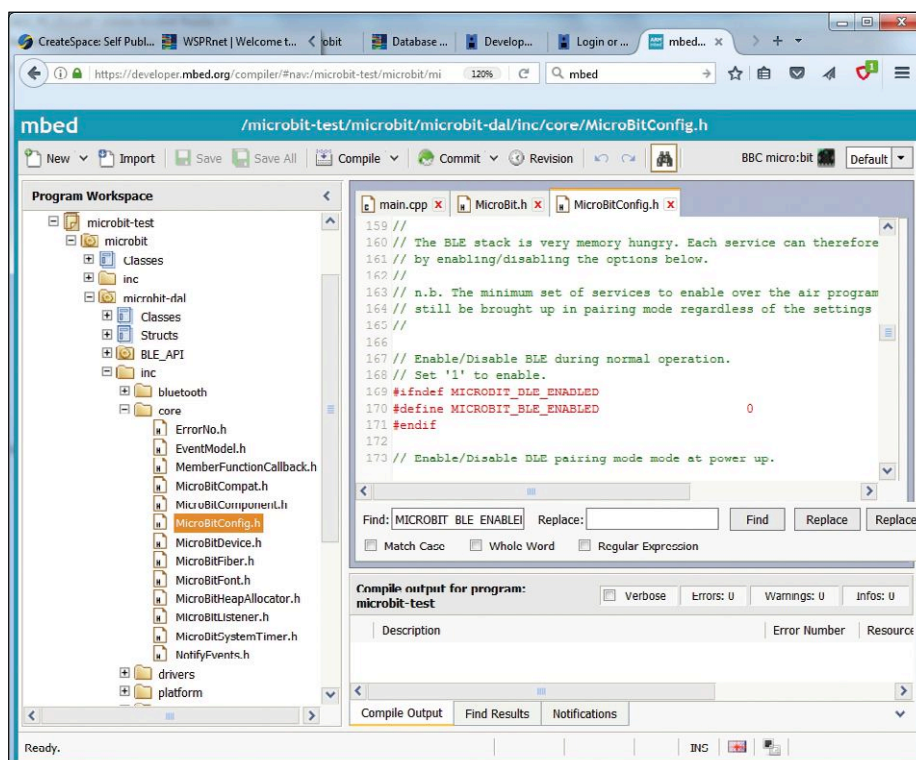


Figure 6. Deactivating BLE.

support platform [6] with simple datagrams (text messages) that the transceiver can handle unaltered. One BBC micro:bit module sends a text message and all other modules in range can receive it. To make this work a default channel and a default transmit power have been defined, so you really don't need to concern yourself with anything. That makes this one of the simplest methods of transferring data without wires. At the same time it gives you the ability to make isolated (potential-free) measurements. A typical application might be an electrocardiogram device, as the electrical (galvanic) separation eliminates any troublesome electrical hum interference.

All you need is two BBC micro:bit modules. One is employed as the measuring instrument for transmitting the data, the second one is programmed as the receiver, for displaying the data or transferring it to a PC over a USB cable. To use MicroBitRadio in mbed you do need to deactivate Bluetooth Low Energy. There's a note about this in the BBC micro:bit documentation: *It is not currently possible to run the*

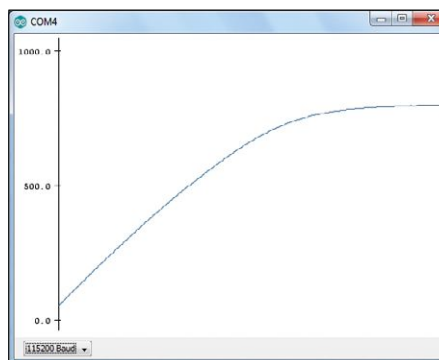


Figure 7. Data measurements sent by wireless link.

MicroBitRadio component and Bluetooth Low Energy (BLE) at the same time. If you want to use the MicroBitRadio functionality, you need to disable the BLE stack on your micro:bit by compiling the runtime with `#define MICROBIT_BLE_ENABLED 0` in your `inc/MicroBitConfig.h` file.

It is not entirely simple to locate the correct point in the numerous files of the runtime system. The exact path is: `microbit\microbit-dal\inc\core\MicroBitConfig.h` (see **Figure 6**). In this

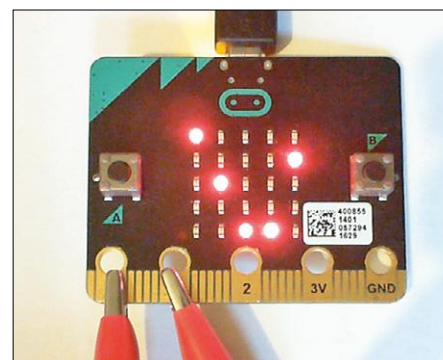


Figure 8. Measuring the internal test signal.

file is an entry `MICROBIT_BLE_ENABLED 1`, in which you need to replace the 1 with a 0. In the next compilation that you make BLE is then deactivated, enabling you to use the simplified MicroBitRadio.

The program in **Listing 3** sends and receives datagrams of measurement data. The sender and receiver can therefore use the same program and also exchange data reciprocally. The sender executes a measurement at P1 and transmits the reading in mV. The receiver passes the received data forward via the

Advertisement

Join the Elektor Community Take out a GOLD Membership now!



GOLD MEMBERSHIP

- ✓ 6x Elektor Magazine (Print)
- ✓ 6x Elektor Magazine (PDF)
- ✓ Access to Elektor Archive (Thousands of Articles)
- ✓ Access to over 1,000 Gerber files
- ✓ Elektor Annual DVD
- ✓ 10% Discount in Elektor Store
- ✓ Exclusive Offers

GREEN MEMBERSHIP


- ✓ 6x Elektor Magazine (PDF)
- ✓ Access to Elektor Archive (Thousands of Articles)
- ✓ Access to over 1,000 Gerber files
- ✓ 10% Discount in Elektor Store
- ✓ Exclusive Offers

Also available:
The all-paperless GREEN Membership!
www.elektor.com/member

USB connection. Nothing changes on the PC side. Here we can again make use of the serial plotter. **Figure 7** illustrates a measurement reading. The transmitter is battery-driven and is located three meters (10 feet) away from the receiver. The link works for up to approx. 10 m (30 ft.). A 10- μ F electrolytic was attached to analog input P1. This was charged up slowly using the 10-M Ω pull-up resistor provided on the PCB. You may notice some deviation from the normal charging curve, because this capacitor had not been used for a long time. In a case like this just a low leakage current flows initially that increases only gradually. On the right-hand side of the diagram the measured voltage lies clearly below 1 V and rises only slowly.

Mini-oscilloscope with LED display

An extremely basic oscilloscope is better than none at all and sometimes it's more important that the device is very small, standalone and easy to handle. Here we see measurement data displayed graphically on the LED display using 5x5 LEDs (**Listing 4**). Even if you are accustomed to a far more sophisticated instrument, you can definitely get results with this little alternative. It is quite remarkable, what is still discernible with such a simple 'scope.

Once again the mini-oscilloscope uses Port 1 as an analog input and additionally employs Port 0 as a PWM output. With a repetition rate of 500 μ s, an output signal with a frequency of 2 kHz is generated. A direct connection to the measurement input shows the limits of the A-to-D converter (**Figure 8**). The sampling time is obviously too long to display sharp edges of the PWM signal. The limiting frequency of this simple oscilloscope is therefore somewhere below 10 kHz. This is not enough for an RF lab but probably adequate for many simple measurements and experiments. 

(160384)

Web Links

- [1] www.elektormagazine.com/160273
- [2] www.elektormagazine.com/160384
- [3] <https://developer.mbed.org/>
- [4] <https://lancaster-university.github.io/microbit-docs/ubit>

Listing 3. Sending and receiving datagrams.

```
//Radio Data
#include "MicroBit.h"
MicroBit uBit;

void onData(MicroBitEvent e)
{
    ManagedString s = uBit.radio.datagram.recv();
    uBit.serial.send (s);
    uBit.serial.send (" \r\n");
}

int main()
{
    uBit.init();
    uBit.messageBus.listen(MICROBIT_ID_RADIO, MICROBIT_RADIO_EVT_DATAGRAM, onData);
    uBit.radio.enable();
    char output[16];
    while (1) {
        int u = 3300 * uBit.io.P1.getAnalogValue()/ 1023;
        itoa (u, output);
        uBit.radio.datagram.send(output);
        uBit.sleep(100);
    }
}
```

- [5] B. Kainka, BBC micro:bit Tests Tricks Secrets Code, CreateSpace 2016
- [6] <https://lancaster-university.github.io/microbit-docs/ubit/radio/>

Listing 4. Using the LED display.

```
//LED-Scope
#include "MicroBit.h"
MicroBit uBit;

int main()
{
    int y;
    uBit.init();
    uBit.io.P0.setAnalogValue(512);
    uBit.io.P0.setAnalogPeriodUs(500);
    uBit.display.enable();
    MicroBitImage image(5,5);
    while (1) {
        for(int x = 0; x < 5; x++){
            y = 4- (uBit.io.P1.getAnalogValue()/205);
            image.setPixelValue(x,y,255);
        }
        uBit.display.print(image);
        uBit.sleep(500);
        image.clear();
    }
}
```