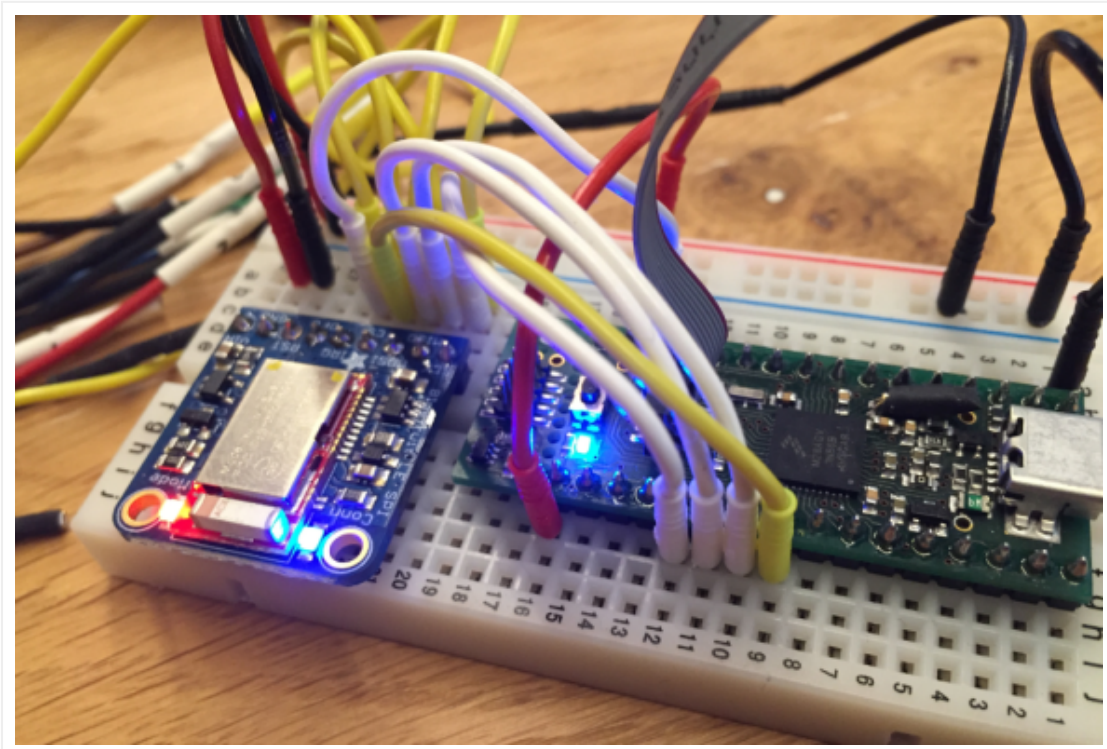
How to Add Bluetooth Low Energy (BLE) Connection to ARM Cortex-M

Posted on **January 9, 2016** by **Erich Styger**

8 Votes

In many of my embedded projects I'm using successfully the Nordic Semiconductor nRF24L01+ (see "Tutorial: Nordic Semiconductor nRF24L01+ with the Freescale FRDM-K64F Board") and the HC-06 Bluetooth transceivers (see "Getting Bluetooth Working with JY-MCU BT_BOARD V1.06") for wireless communication. However, the nRF24L01+ is using a proprietary protocol, and the HC-06 does not work with Apple products (it does very well with Android devices). To close that gap I decided to add Bluetooth Low Energy (BLE, or Bluetooth 4.x). So this post is about how to add Bluetooth Low Energy (BLE) to NXP (formerly Freescale) Kinetis devices:



— BLE Enabled Kinetis

Outline

In this article I describe how to use the Adafruit Bluefruit LE Friend (SPI) module with the Freescale/NXP Kinetis microcontroller. The Adafruit tutorials describe how to use it with the

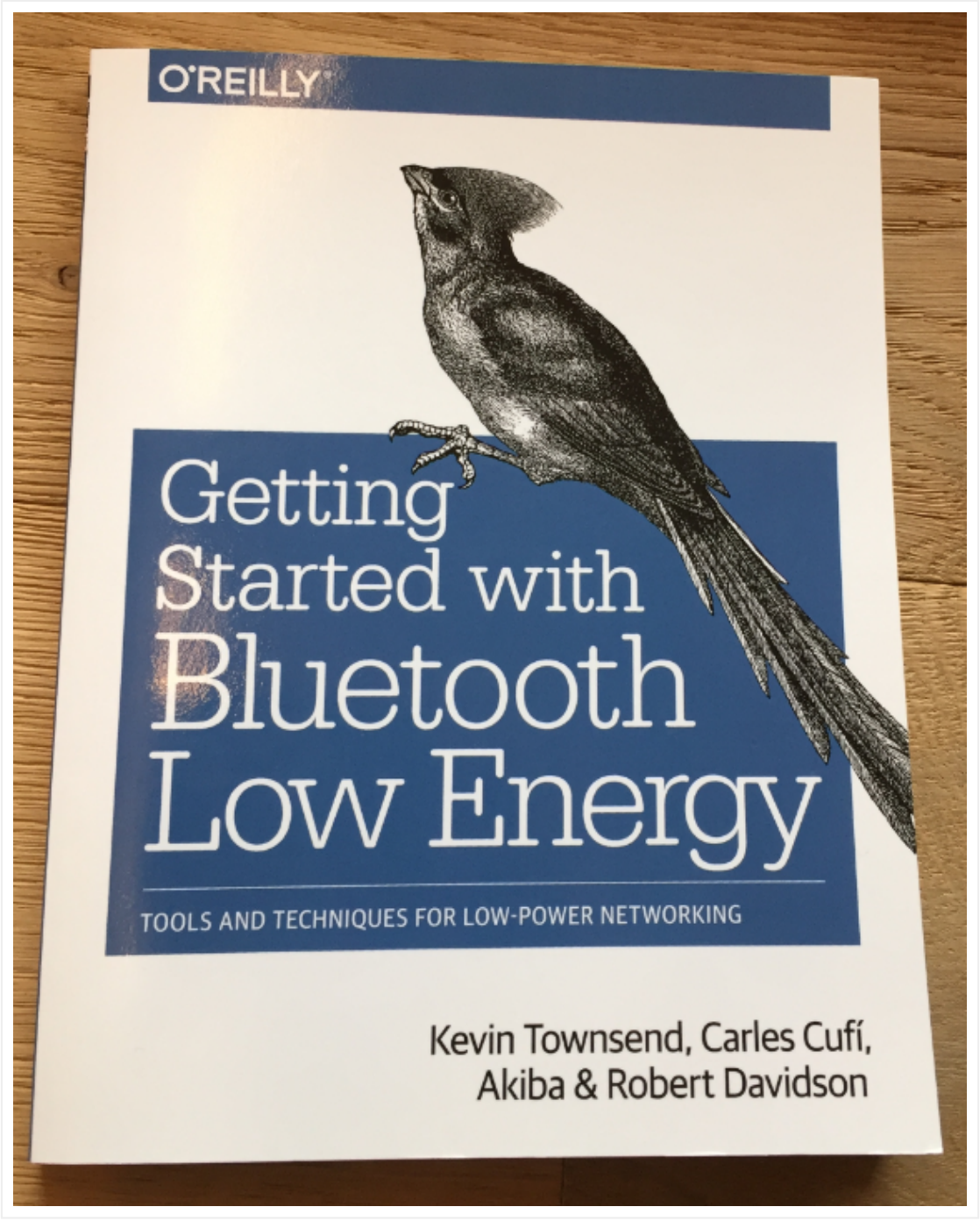
Arduino IDE, but this post is about how to use it with a C/C++ environment (Kinetis Design Studio with GNU ARM Embedded tools). It describes the SPI connection and protocol, and how to use it in a command line mode. As application it implements a UART-over-BLE (virtual UART over BLE) to send and receive text from a mobile phone or tablet. It even adds an RTOS (FreeRTOS) and added Processor Expert components to make life easier, but they are not necessary if you want to do it without them.

The project and source code is available on GitHub: https://github.com/ErichStyger/mcuoneclipse/tree/master/Examples/KDS/tinyK20/tinyK20_Adafruit_BLE

Bluetooth Low Energy

Bluetooth Low Energy is not Bluetooth, although it does use the same 2.4 GHz band. I think they named it after Bluetooth purely for marketing reasons, because BLE is more like the nRF24L01+ protocol with added security. Regardless of the naming behind BLE, it provides a wireless connectivity for low power and small bandwidth applications.

I recommend reading the"Getting Started with Bluetooth Low Energy" book (ISBN 978-1-491-94951-1, O'Reilly, Kevin Townsend, Carles Cufi, Akiba & Robert Davidson).



— Getting Started With Bluetooth Low Energy

What makes BLE great is that it allows to communicate with embedded devices easily from smartphones and tablets. So my goal is to drive robots with it, change the color of LEDs and lamps or do any other interactions with embedded devices.

BLE Module

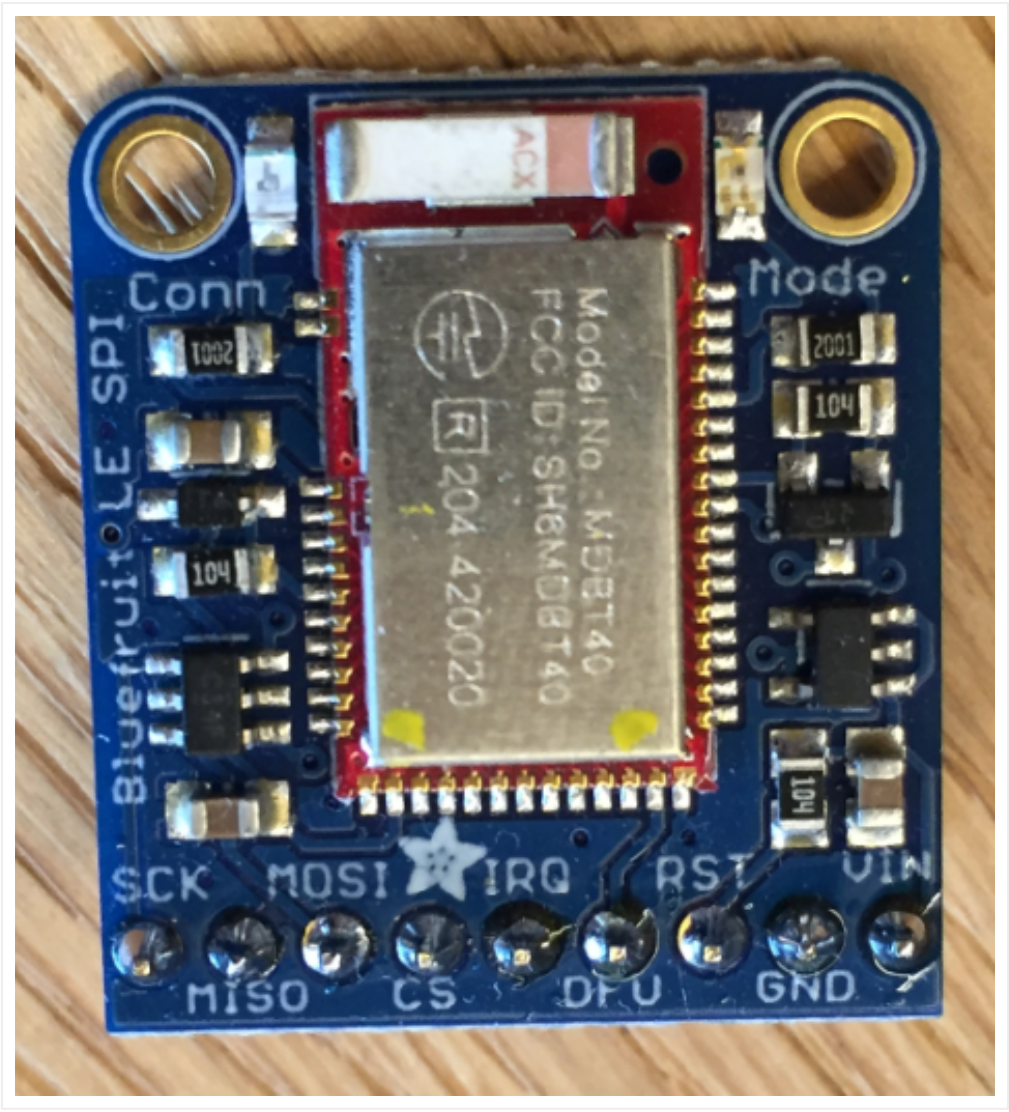
There are plenty of different BLE modules and stacks available on the market. I have evaluated several modules and I have decided to use the Adafruit 'Bluefruit' LE modules:

1. **'Ready-to-use' Modules:** Unlike other solutions, I don't have to mess up with a BLE stack. The modules already have a firmware (UART over BLE) loaded.
2. **Open Source:** Unlike other solutions which come with libraries and 'secret hidden code', Adafruit shares the source code and has tutorials how I can flash the firmware e.g. with a Segger J-Link
3. **Multiple Board options**: I have the choice of UART or SPI breakout boards, Arduino Shield, a USB dongle and BLE sniffer version.
4. **Excellent software and tutorials**: Adafruit provides first class and fun tutorials, with lots of background information. Additionally they have very good community support.
5. **Nordic Semiconductor** transceiver: I see many other module vendors using the Nordic chips, and Nordic has good software and tools support too.

In this post I'm using the Adafruit Bluefruit LE SPI Friend. I selected this one over the UART version because I don't want to give up a serial port with hand shaking signals, and because the board is a bit smaller than the UART version.

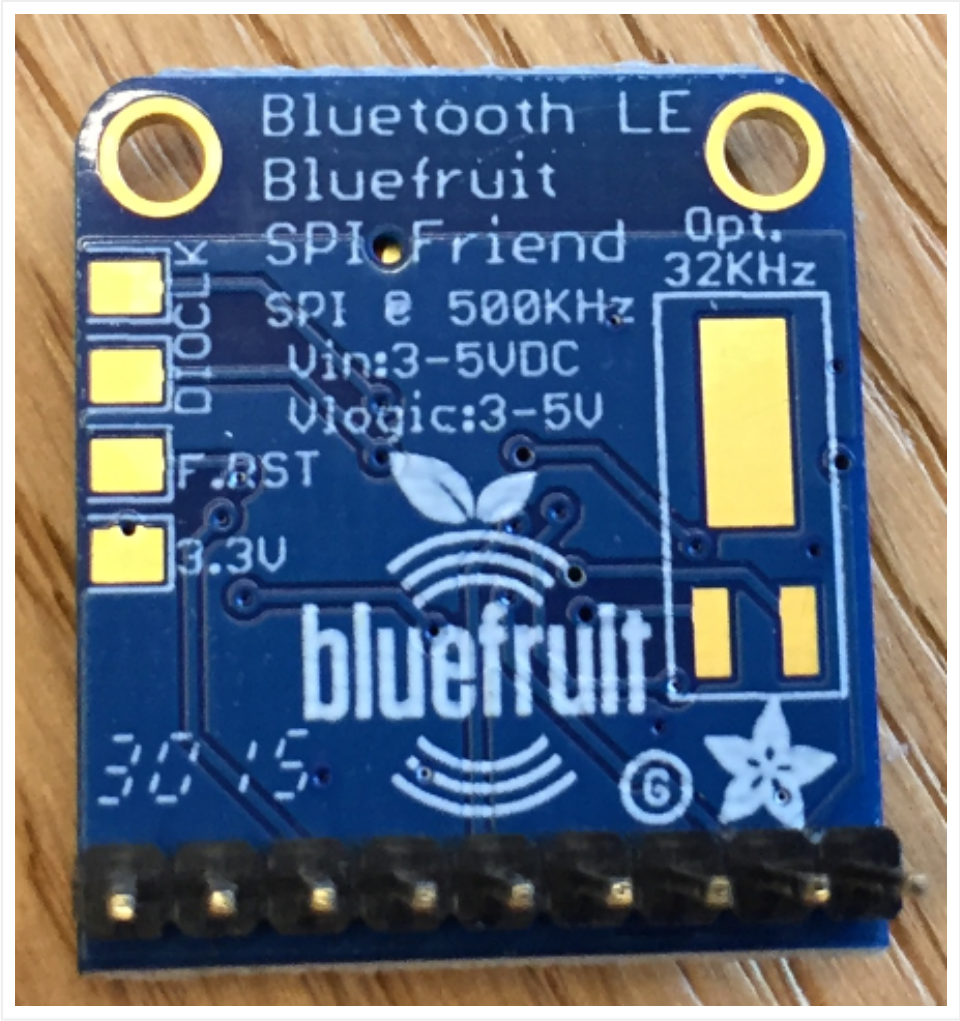
The top side of the module has a blue Connection and a red Mode LED. The signals are bread-board friendly:

- SPI clock **SCK** (4 MHz max)
- SPI **MISO** and **MOSI** (most significant bit first)
- SPI chip select **CS** (low active)
- **IRQ** to signal data available (high active)
- **DFU** pin to force firmware update or factory reset (optional)
- **RST** pin to do reset (optional)
- **GND**
- Supply Voltage **VIN** (3.3-5V) for onboard regulator to 3.3V



— Bluefruit SPI Board Top Side

The bottom side of the module exposes pads for SWD debugging (DIO and CLK), a pad for factory reset and 3.3V output of the onboard regulator (up to 250 mA). Optionally the module can be extended with an optional 32 kHz clock source:



— Bluefruit SPI Board Bottom Side

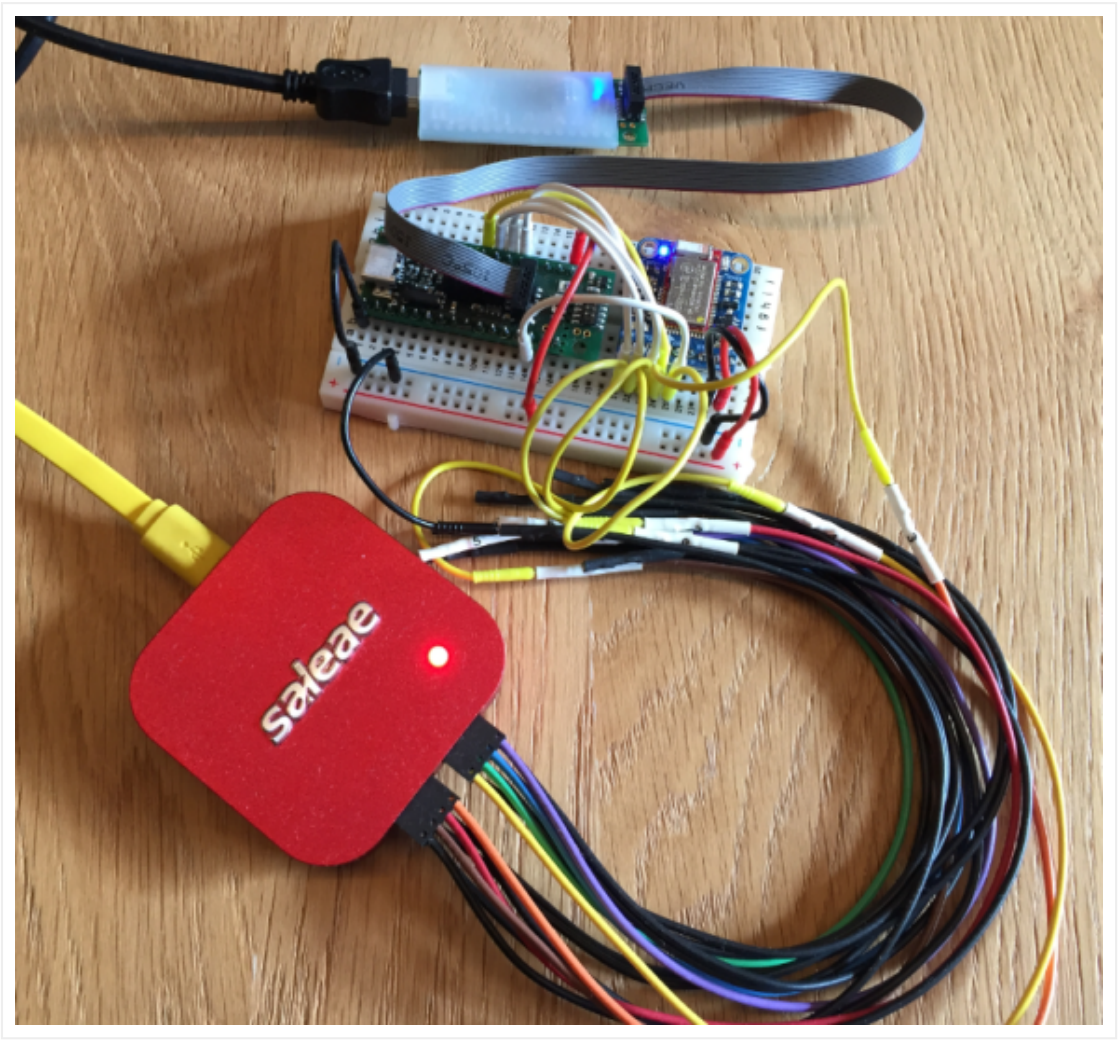
Main Features of the board (mostly based on information from Adafruit):

- Nordic Semiconductor nRF51822
- ARM Cortex M0 core running at 16MHz
- 256KB flash memory
- 32KB SRAM
- 2.4 GHz Chip Antenna
- Peak current draw <20mA (radio actively transmitting/receiving)
- 3.3V and 5V-safe inputs (Arduino Uno friendly, etc.)
- On-board 3.3V voltage regulation
- Bootloader with support for safe OTA firmware updates
- AT command set for easy configuration

The Adafruit UART/SPI breakout modules cost $18.50 which is reasonable to me. Yes, there are cheaper modules from other vendors, and modules with better hardware specs. But to me the available software, tools and tutorials for the Adafruit modules was key for my decision. After the fact, it only took me around one hour to get my first BLE connection with my iPhone .

Hardware Setup

I'm using a tinyK20 (ARM Cortex-M4) with a Freescale/NXP K20DX128 as the application processor. Another tinyK20 is used as debugger. I'm using the tinyK20 because it makes it easy to use it with a bread-board setup:



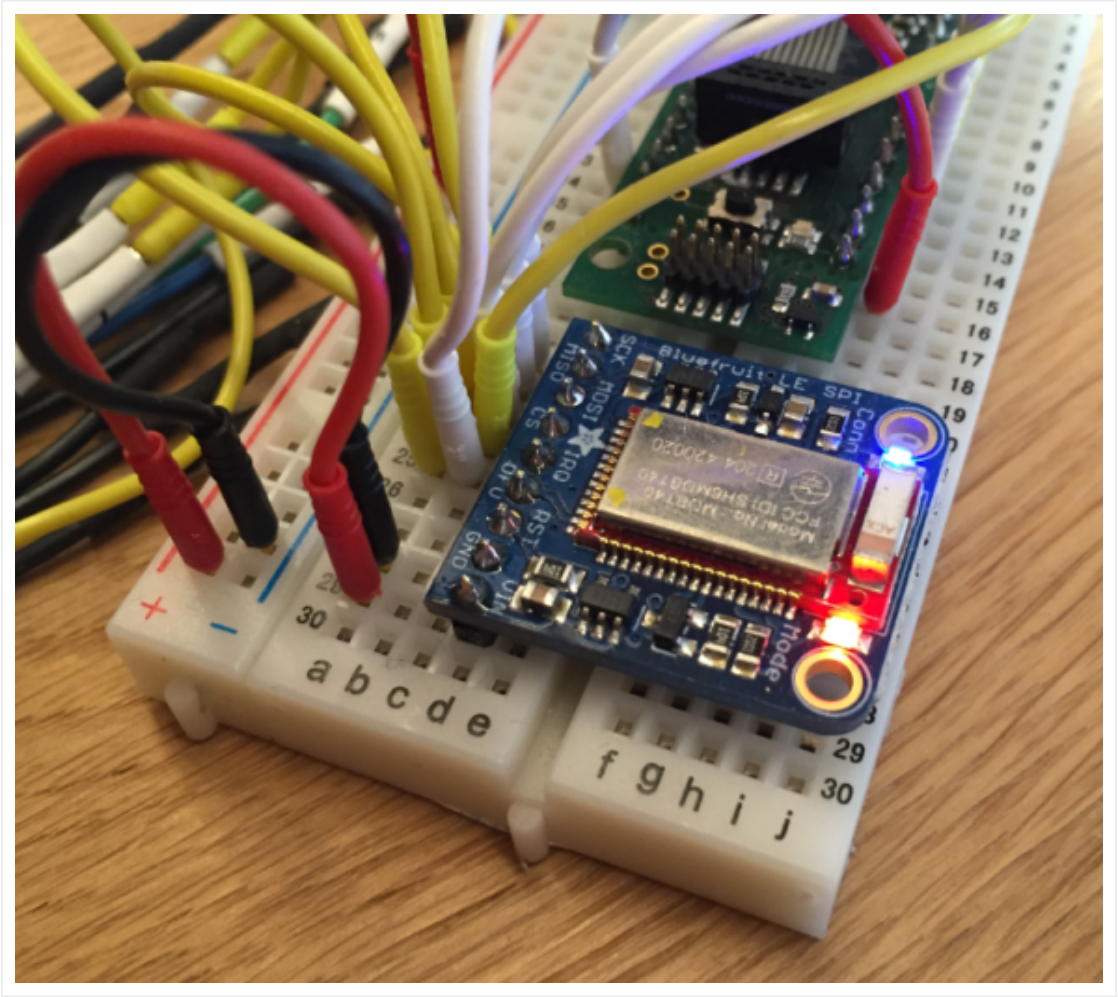
— BLE Board Setup with Logic Analyzer

The setup can be changed to use any other board, e.g. one of the Freescale/NXP Freedom boards. If using the Freedom board, you don't need an extra debugger board as it is already included as OpenSDA.

I highly recommend to use a logic analyzer for this kind of development. If you want to make your own open source logic analyzer, then have a look at "Updated Freedom Board Logic Analyzer with DMA"

Wiring

I'm using a bread-board to connect the module to the microcontroller.

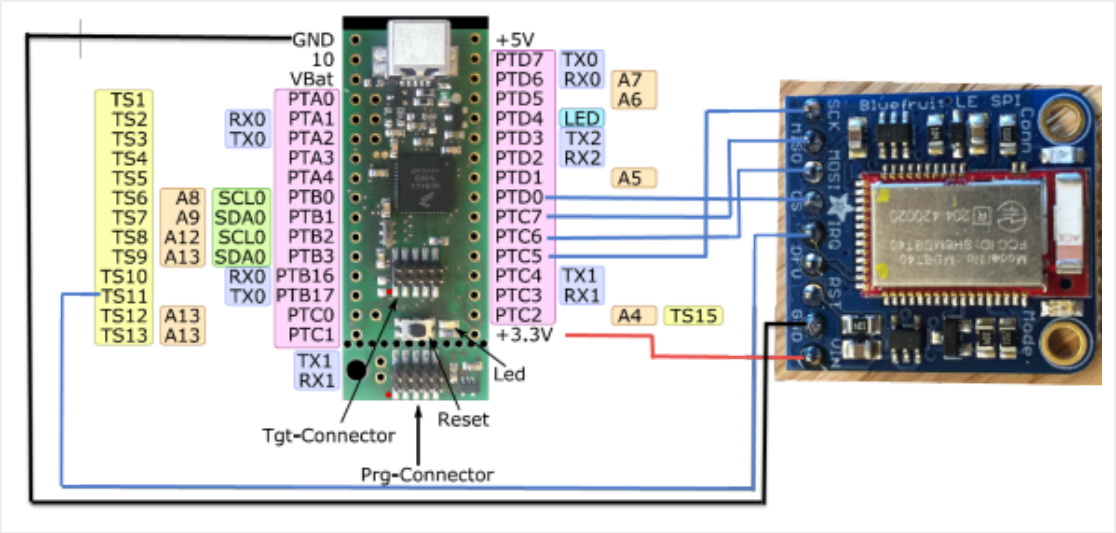


— BLE Breadboard Wiring

I have used the following wiring:

- Power: GND and 3.3V
- SPI MISO: PTC7
- SPI MOSI: PTC6
- SPI CLK: PTC5
- SPI CS: PTD0
- IRQ: PTB17

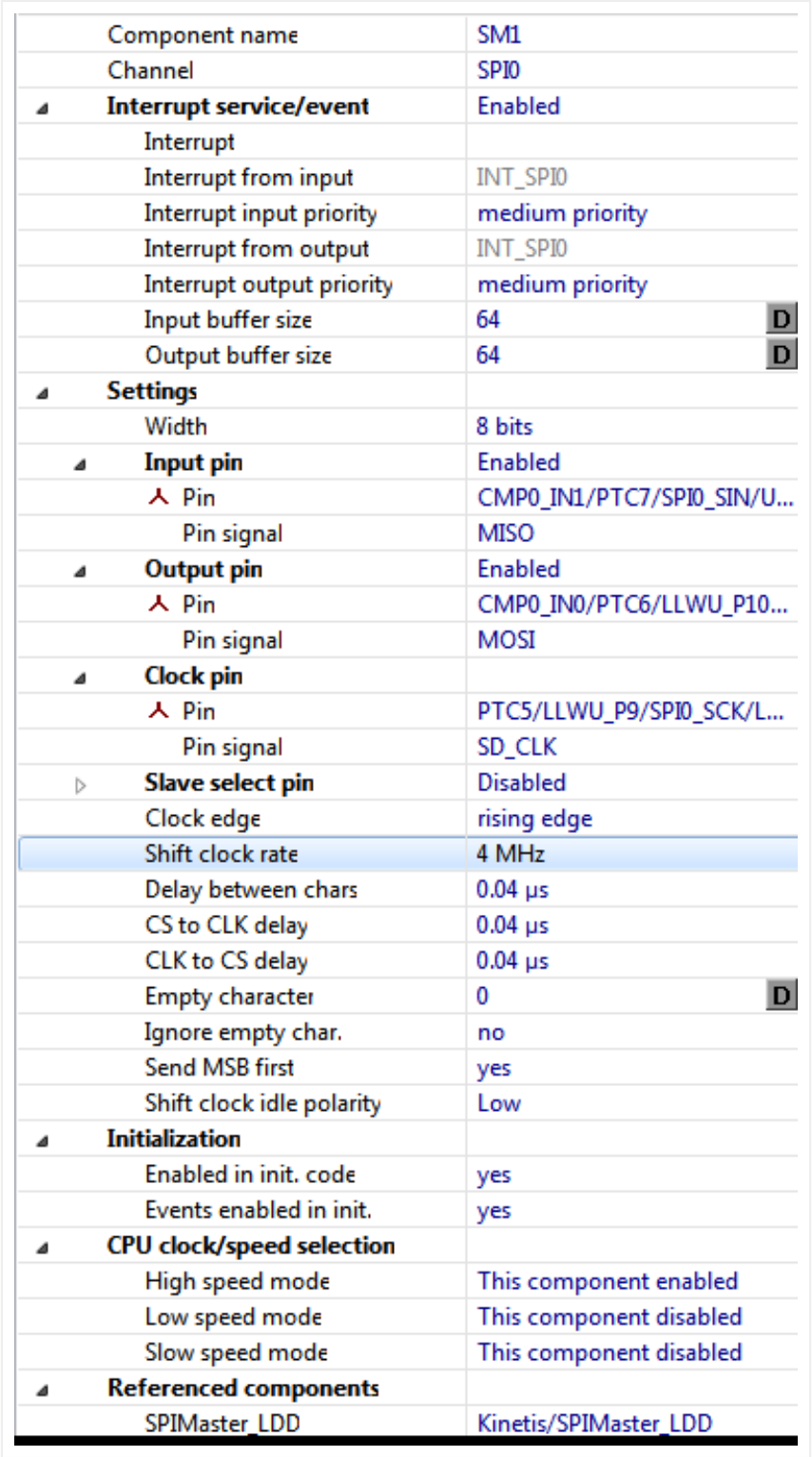
The diagram below shows the complete wiring:



— tinyK20 Wiring with BLE

Software Components

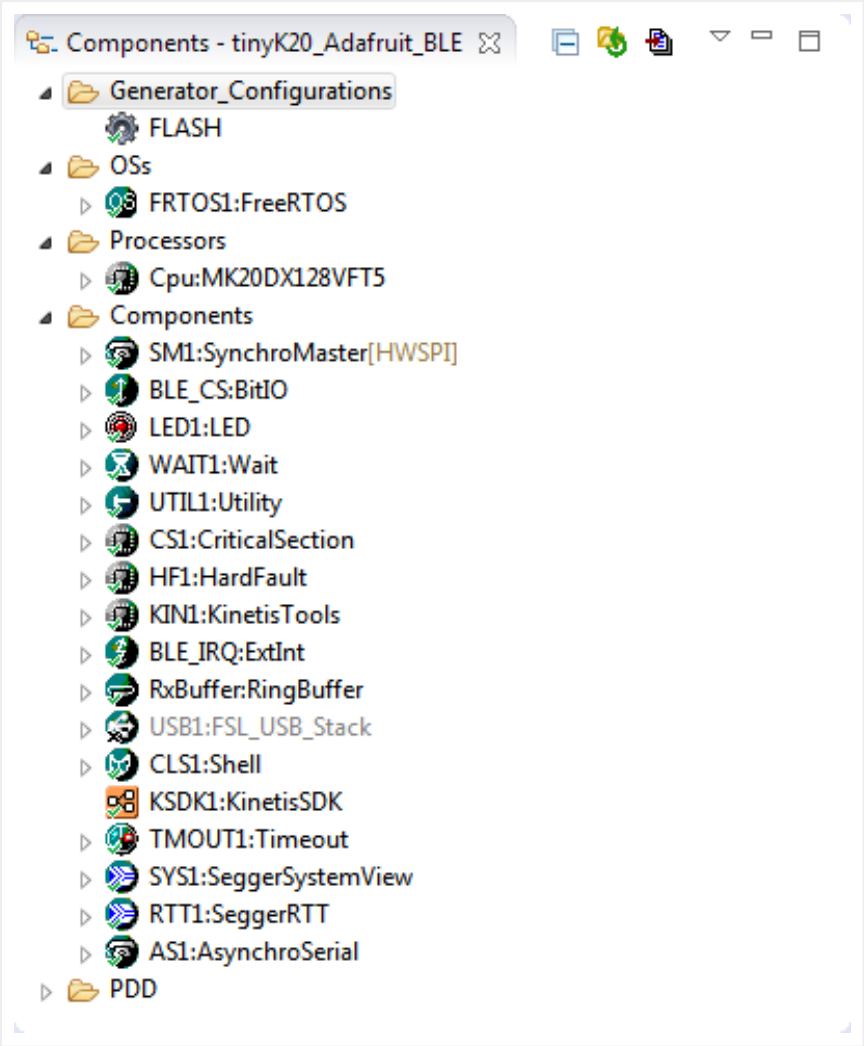
With my tinyK20 I'm using hardware SPI with the help of the Processor Expert SynchroMaster component. But of course it is possible to use any other SPI driver, it is just that with the Processor Expert approach it is super easy. The screenshot below shows the pin assignments and SPI configuration:



Component name	SM1
Channel	SPI0
Interrupt service/event	Enabled
Interrupt	
Interrupt from input	INT_SPI0
Interrupt input priority	medium priority
Interrupt from output	INT_SPI0
Interrupt output priority	medium priority
Input buffer size	64 D
Output buffer size	64 D
Settings	
Width	8 bits
Input pin	Enabled
Pin	CMP0_IN1/PTC7/SPI0_SIN/U...
Pin signal	MISO
Output pin	Enabled
Pin	CMP0_IN0/PTC6/LLWU_P10...
Pin signal	MOSI
Clock pin	
Pin	PTC5/LLWU_P9/SPI0_SCK/L...
Pin signal	SD_CLK
Slave select pin	Disabled
Clock edge	rising edge
Shift clock rate	4 MHz
Delay between chars	0.04 µs
CS to CLK delay	0.04 µs
CLK to CS delay	0.04 µs
Empty character	0 D
Ignore empty char.	no
Send MSB first	yes
Shift clock idle polarity	Low
Initialization	
Enabled in init. code	yes
Events enabled in init.	yes
CPU clock/speed selection	
High speed mode	This component enabled
Low speed mode	This component disabled
Slow speed mode	This component disabled
Referenced components	
SPIMaster_LDD	Kinetis/SPIMaster_LDD

— SPI configuration

The SPI component is the most important one. Besides of that I need an interrupt pin (BLE_IRQ) and the SPI chip select (BLE_CS). The screenshot below shows the full set of components:



— BLE Application Components

- **FreeRTOS**: I'm using the RTOS to simplify running multiple things, but it is easy to use the application bare metal too
- **SynchroMaster** is handling the SPI communication to the BLE module
- **BLE_CS** implements the SPI chip select
- A **LED** is used for status on the tinyK20
- **Wait** implements different delay routines
- **Utility** implements string manipulation routines
- **CriticalSection** implements handling of critical sections outside the RTOS
- **HardFault** is a component to help debug hard faults (see "A Processor Expert Component to Help with Hard Faults").
- **KinetisTools** implements low level Kinetis functionality
- **BLE_IRQ** is for the Bluefruit IRQ pin
- The **Ringbuffer** is used to buffer incoming BLE messages
- **FSL_USB_Stack** implements an optional USB CDC stack
- The **Shell** component offers the command line interface
- **Timeout** is used in the Bluefruit interface to avoid blocking for too long
- **SeggerSystemView** and **SeggerRTT** are used for debugging purposes (see "Segger

SystemView: Realtime Analysis and Visualization for FreeRTOS“)

- **AsynchroSerial** is a UART connection to the debugging tinyK20 (Serial-over-USB)

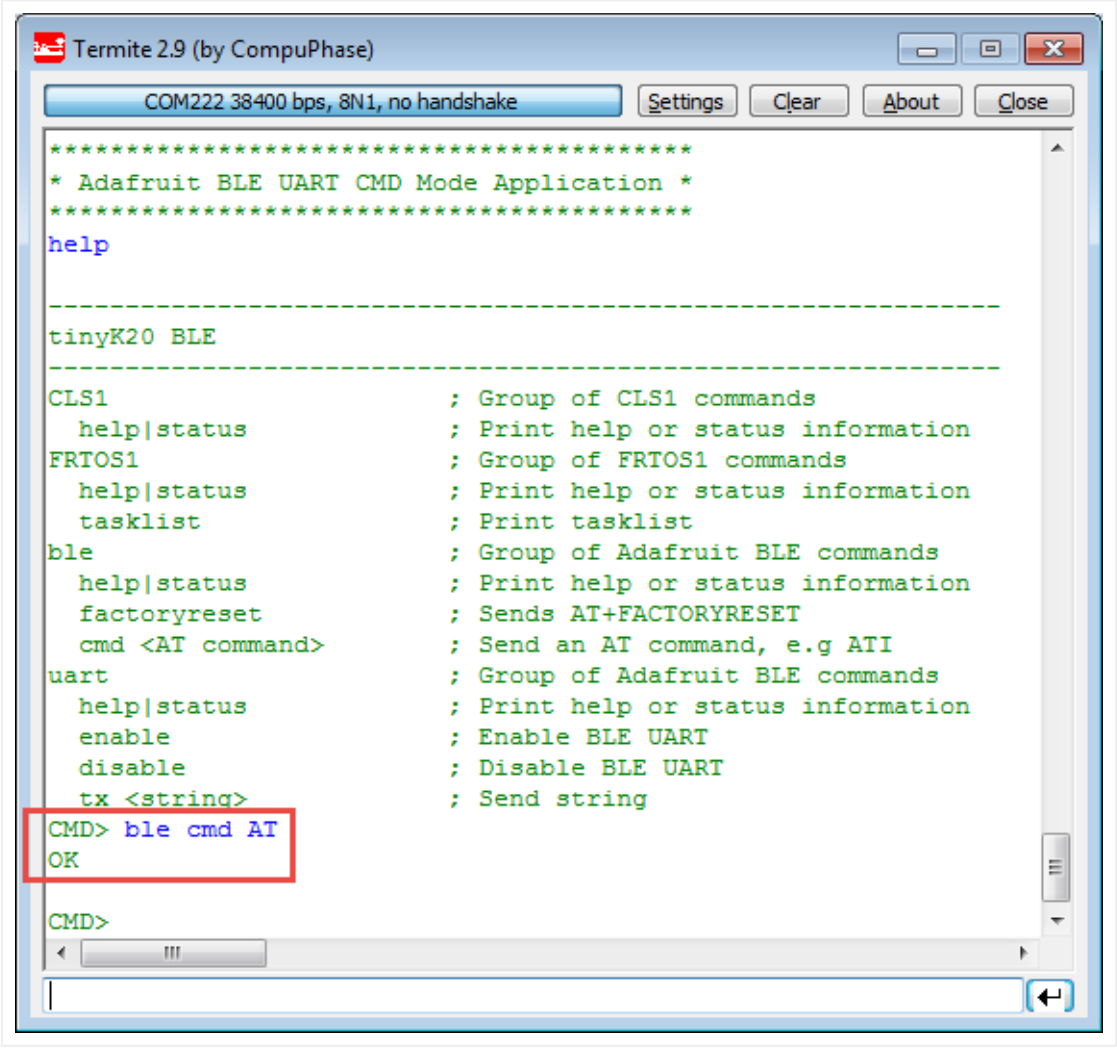
SPI Protocol

The

```
AT\n
```

command is used to verify that the communication is working (command mode). The module shall respond with

```
OK\r\n
```



— BLE AT Command in Shell

The logic analyzer image below shows the

```
AT\n
```

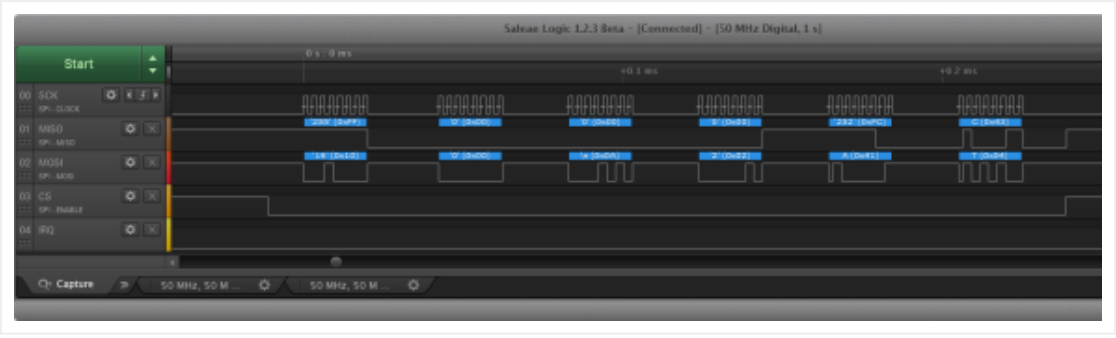
command and the response from the module with

```
OK\r\n
```

— AT Command and Response

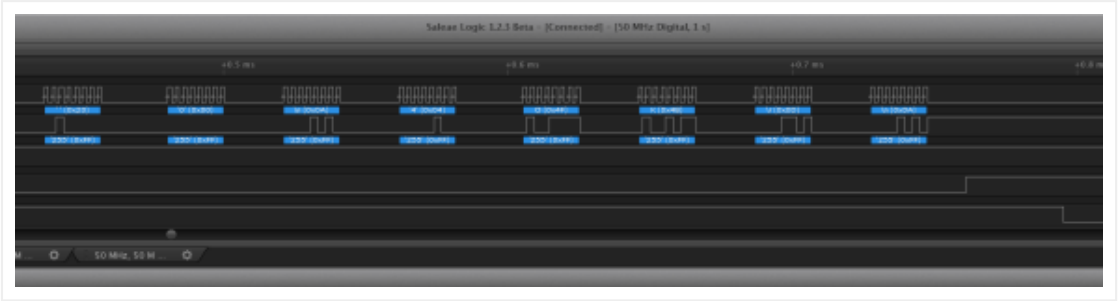
- **CS** (Chip Select) is LOW ACTIVE and kept low during the transaction
- **IRQ** is HIGH ACTIVE and indicates that a message is preset at the module to be retrieved

Below the details of the AT command:



— AT Command Details

And here the response from the module:



— AT Response

The SPI protocol is using the Adafruit **SDEP** (**S**imple **D**ata **E**xchange **P**rotocol, https://github.com/adafruit/Adafruit_BluefruitLE_nRF51/blob/master/SDEP.md) protocol: basically it is a packet oriented protocol over SPI. It is a bus neutral protocol which allows to send request and response over multiple communication channels including SPI. Each message is in the following format:

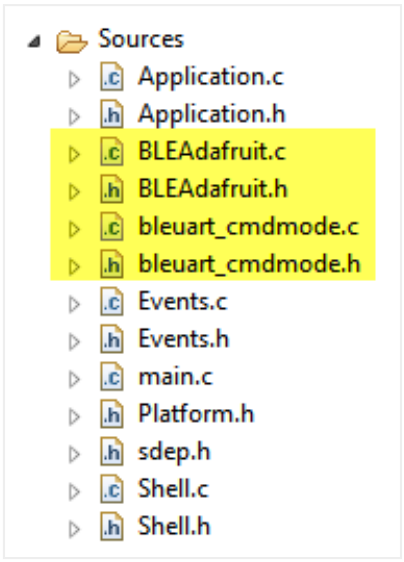
- Message type (uint8_t), e.g. 0x10 for 'command'
- Command ID (uint16_t), e.g. 0x000A for 'AT Wrapper'
- Payload Length (uint8_t) with a 'more data' bit
- Variable Payload

The "AT" command is encoded like this:

```
0x10 : 0x00 0x0A : 0x02   : 0x41 0x54
Type : ID        : Length :  'A'  'T'
```

UART over BLE Example

To test the BLE functionality, I have implemented a simple UART over BLE application. The functionality is implemented in **bleuart_cmdmode.c** with the AT command driver in **BLEAdafruit.c**:



— UART over BLE application

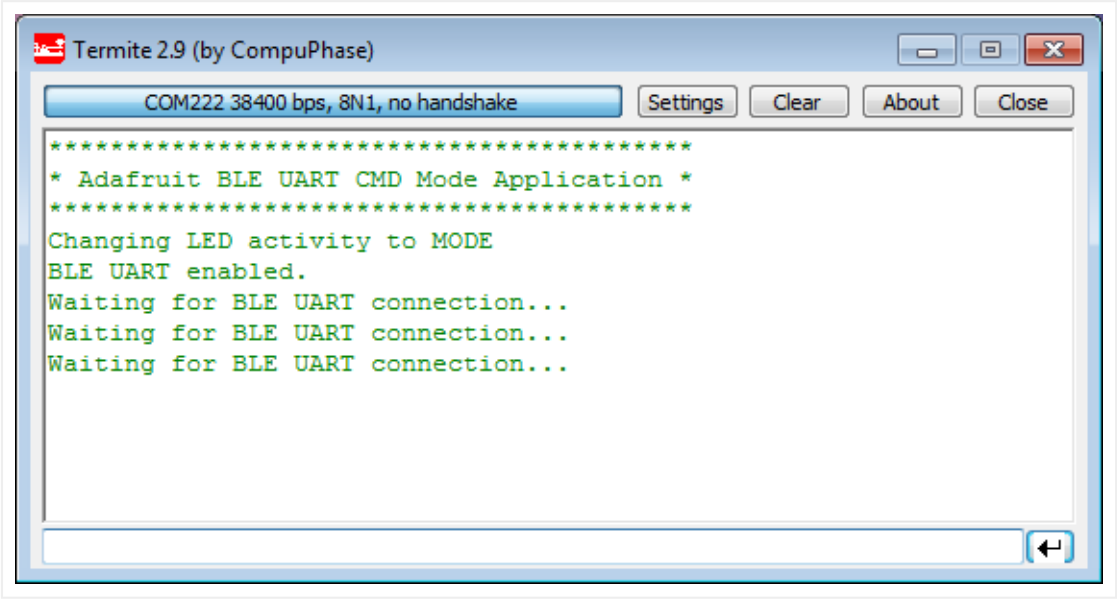
The application runs in an endless loop:

```
#include "bleuart_cmdmode.h"
#include "UTIL1.h"
#include "LED1.h"
#include "FRTOS1.h"
#include "BLEAdafruit.h"
#include "CLS1.h"
#include "UTIL1.h"

#define MAX_TX_MSG_SIZE     48 /* maximum UART message length t
static uint8_t txBuffer[MAX_TX_MSG_SIZE] = "";
static bool isConnected = FALSE;
static bool isEnabled = FALSE;

static void BleUartTask(void *pvParameters) {
  uint8_t buf[MAX_TX_MSG_SIZE];
  uint8_t txBuf[MAX_TX_MSG_SIZE+sizeof("[Tx] ")+sizeof("AT+BLEU
  uint8_t res;
  CLS1_ConstStdIOType *io = CLS1_GetStdio();
  int i;
  bool prevIsEnabled = FALSE;

  BLE_Init(); /* initialize BLE module, has to be done when int
  CLS1_SendStr("*****************************************\r\n"
  CLS1_SendStr("* Adafruit BLE UART CMD Mode Application *\r\n"
  CLS1_SendStr("*****************************************\r\n"
  for(;;) {
    if (!prevIsEnabled && isEnabled) { /* enabled now */
      prevIsEnabled = TRUE;
      BLE_Echo(FALSE); /* Disable command echo from Bluefruit *
      CLS1_SendStr("Changing LED activity to MODE.\r\n", io->st
      res = BLE_SendATCommandExpectedResponse("AT+HWMODELED=1\n
      if (res!=ERR_OK) {
        CLS1_SendStr("Failed setting LED mode.\r\n", io->stdErr
      }
      CLS1_SendStr("BLE UART enabled.\r\n", io->stdOut);
    } else if (prevIsEnabled && !isEnabled) { /* disabled now *
      prevIsEnabled = FALSE;
      CLS1_SendStr("BLE UART disabled.\r\n", io->stdOut);
    }
```

```
40      if (isEnabled) {
41        while(isEnabled && !(isConnected=BLE_IsConnected())) { /*
42          CLS1_SendStr("Waiting for BLE UART connection...\r\n", :
43          for(i=0;i<5 && isEnabled;i++) {
44            FRTOS1_vTaskDelay(pdMS_TO_TICKS(1000));
45            LED1_Neg();
46          }
47        }
48        if (isConnected) {
49          CLS1_SendStr("Connected!\r\n", io->stdOut);
50        }
51        while(isEnabled) { /* will break */
52          isConnected=BLE_IsConnected();
53          if (!isConnected) {
54            CLS1_SendStr("Disconnected!\r\n", io->stdOut);
55            break; /* get out of loop */
56          }
57          if (txBuffer[0]!='\0') { /* have something to tx */
58            /* copy buffer */
59            taskENTER_CRITICAL();
60            UTIL1_strcpy(txBuf, sizeof(txBuf), "AT+BLEUARTTX=");
61            UTIL1_strcat(txBuf, sizeof(txBuf), "[Tx] ");
62            UTIL1_strcat(txBuf, sizeof(txBuf), txBuffer);
63            txBuffer[0] = '\0';
64            taskEXIT_CRITICAL();
65            /* send tx string */
66            res = BLE_SendATCommandExpectedResponse(txBuf, buf, s:
67            if (res!=ERR_OK) {
68              CLS1_SendStr("Failed to Tx string\r\n", io->stdErr)
69            }
70          }
71          /* check Rx */
72          res = BLE_SendATCommandExpectedResponse("AT+BLEUARTRX\n'
73          if (res==ERR_OK) {
74            if (UTIL1_strncmp(buf, "OK\r\n", sizeof("OK\r\n")-1)==
75              /* only OK as response: no data */
76            } else {
77              /* print response */
78              UTIL1_strCutTail(buf, "OK\r\n"); /* cut off the OK |
79              CLS1_SendStr("[Rx] ", io->stdOut);
80              CLS1_SendStr(buf, io->stdOut);
81            }
82          }
83          FRTOS1_vTaskDelay(pdMS_TO_TICKS(50));
84          LED1_Neg();
85        } /* while */
86      } else {
87        FRTOS1_vTaskDelay(pdMS_TO_TICKS(500));
88        LED1_Neg();
89      }
90    }
91  }
```

After initialization, it waits until a connection is established. Then it checks with "**AT+BLEUARTRX\n**" if we have incoming characters. If yes, it prints them with [Rx] in front of it. If we have characters to transmit in the txBuffer, then we send them with "**AT+BLEUARTTX\n**". Strings to be sent in the txBuffer can be sent with the Shell.

I connect to the board with a terminal program (puTTY or Termite). After writing a startup message it will wait for a BLE UART connection:



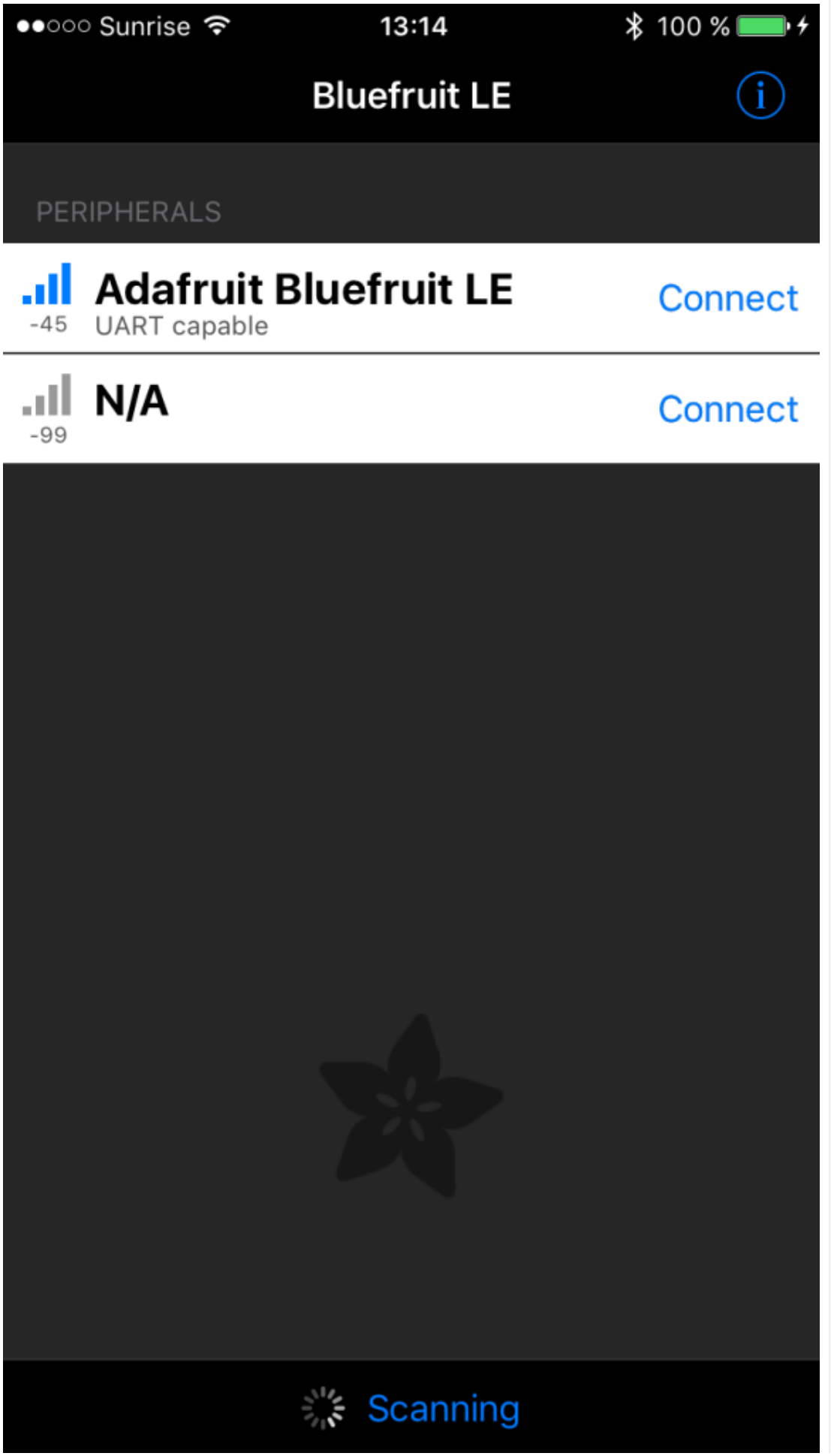
— Waiting for BLE UART Connection

To connect to the BLE module I can use the Adafruit mobile app:

- Android: https://play.google.com/store/apps/details?id=com.adafruit.bluefruit.le.connect
- Apple: https://learn.adafruit.com/bluefruit-le-connect-for-ios/settings

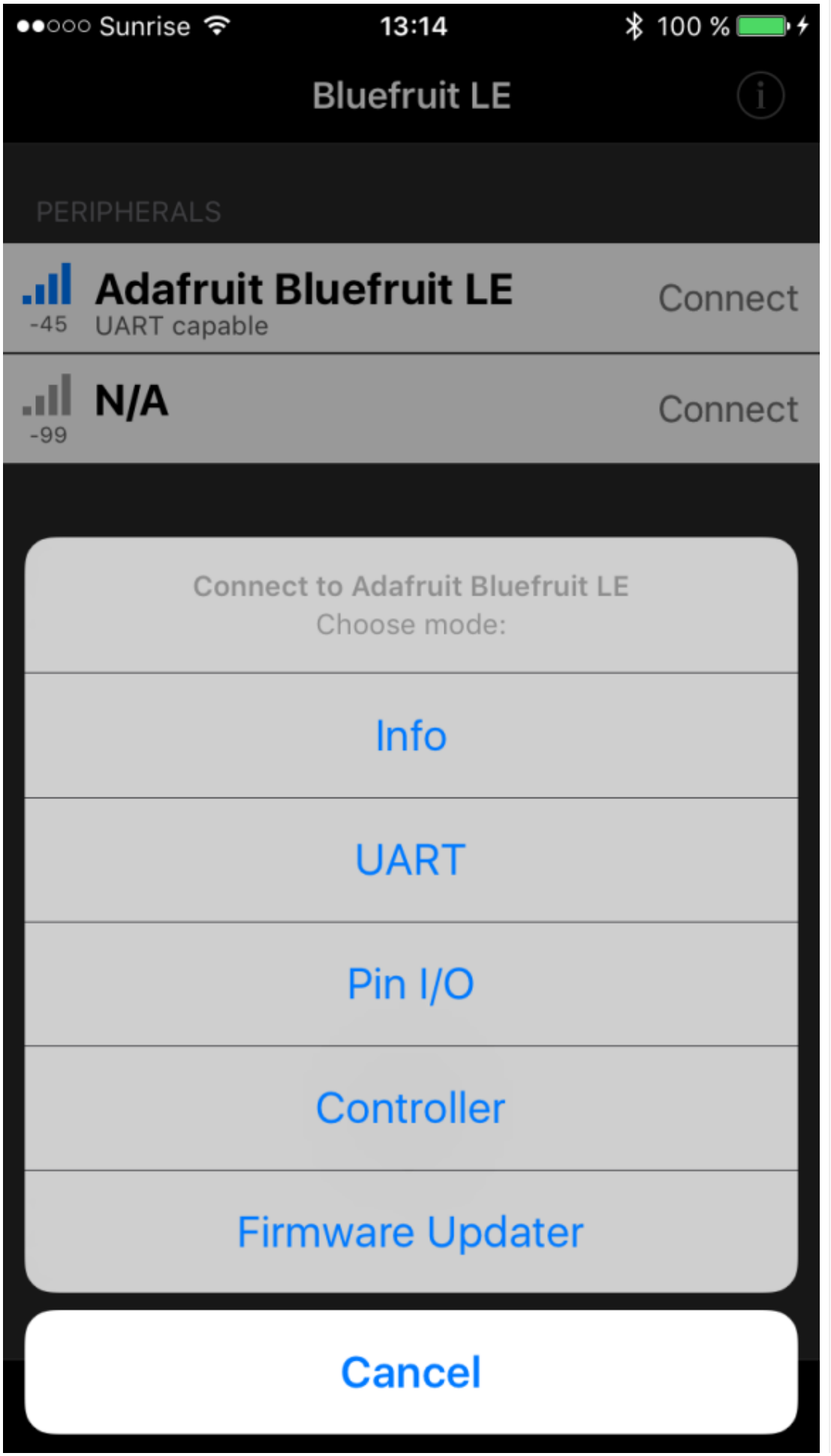
Alternatively, the Nordic Semiconductor UART application can be used:
https://learn.adafruit.com/getting-started-with-the-nrf8001-bluefruit-le-breakout/testing-uart

In the mobile app it scans for available BLE devices:


●●○○○ Sunrise
13:14
100 %
Bluefruit LE
PERIPHERALS
Adafruit Bluefruit LE
-45
UART capable
Connect
N/A
-99
Connect
Scanning

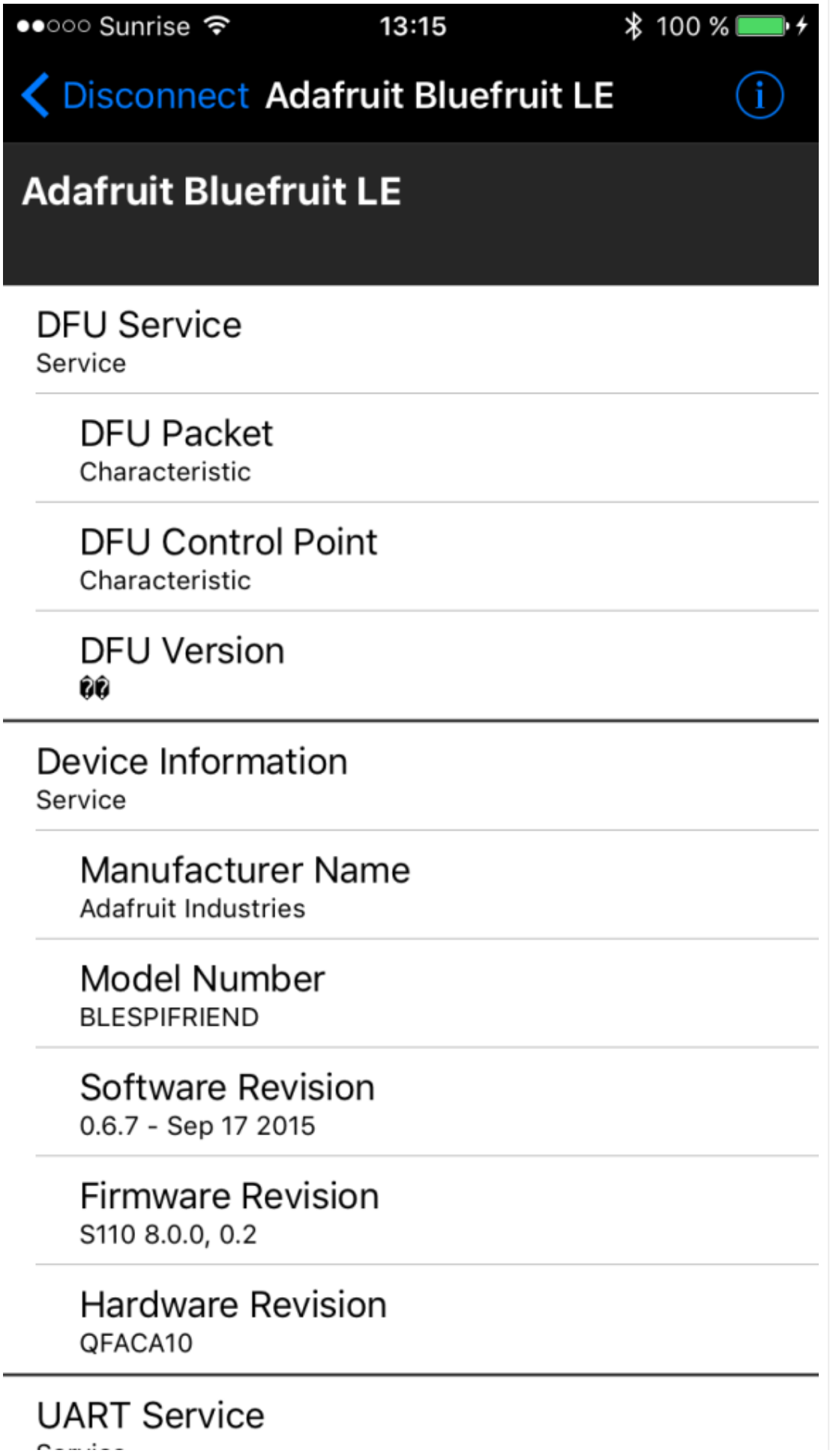
— Scanning for BLE Devices

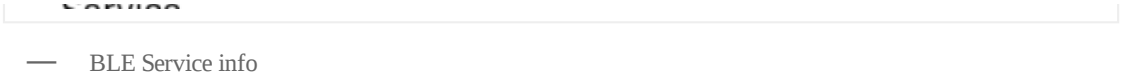
Connecting to the device gives me a list of available services:


●●○○○ Sunrise
13:14
100 %
Bluefruit LE
PERIPHERALS
-45
Adafruit Bluefruit LE
UART capable
Connect
-99
N/A
Connect
Connect to Adafruit Bluefruit LE
Choose mode:
Info
UART
Pin I/O
Controller
Firmware Updater
Cancel

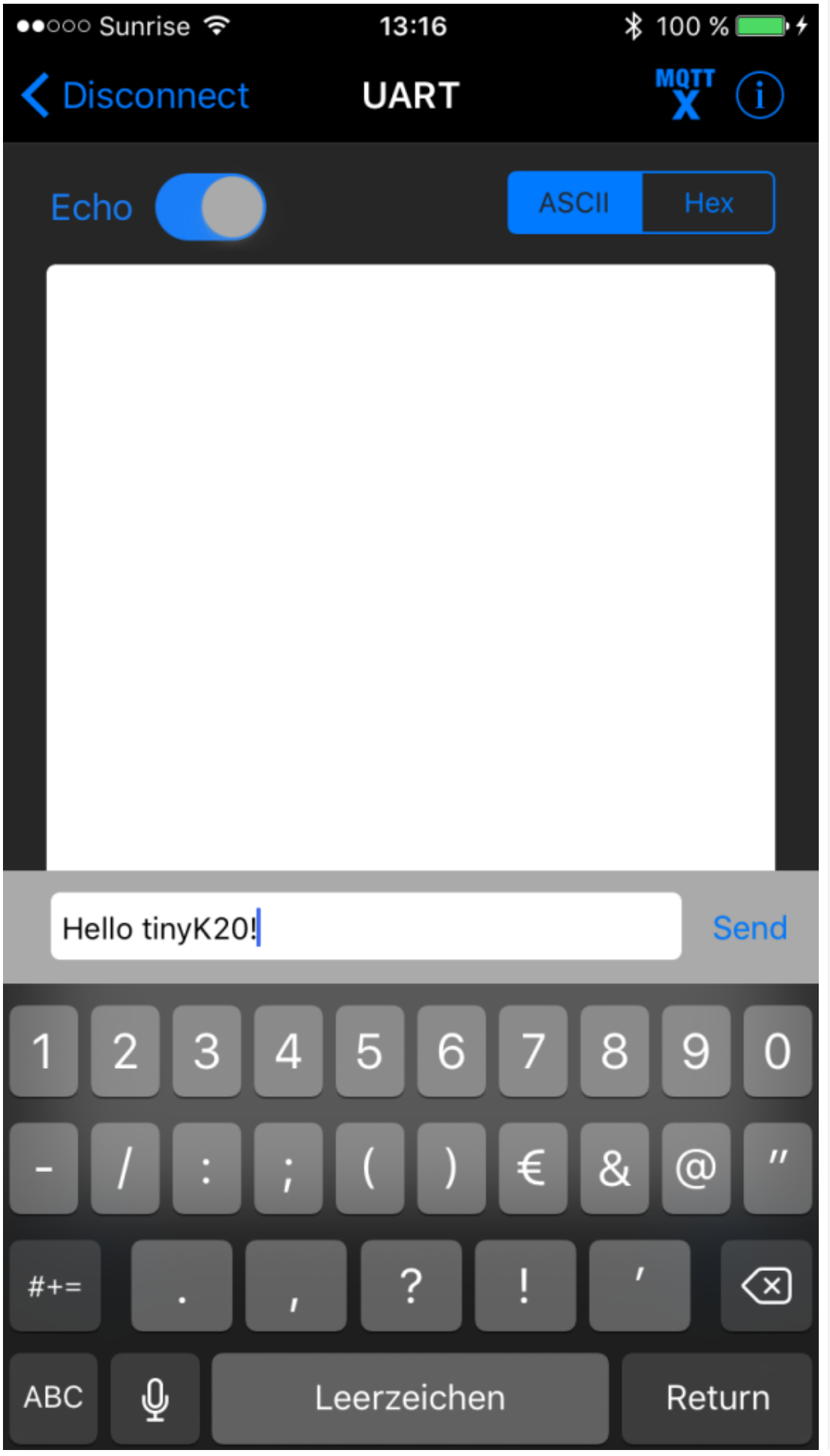
— BLE Services

The Info service provides details about the services available:


●●○○○ Sunrise
13:15
100 %
Disconnect
Adafruit Bluefruit LE
Adafruit Bluefruit LE
DFU Service
Service
DFU Packet
Characteristic
DFU Control Point
Characteristic
DFU Version
Device Information
Service
Manufacturer Name
Adafruit Industries
Model Number
BLESPIFRIEND
Software Revision
0.6.7 - Sep 17 2015
Firmware Revision
S110 8.0.0, 0.2
Hardware Revision
QFACA10
UART Service

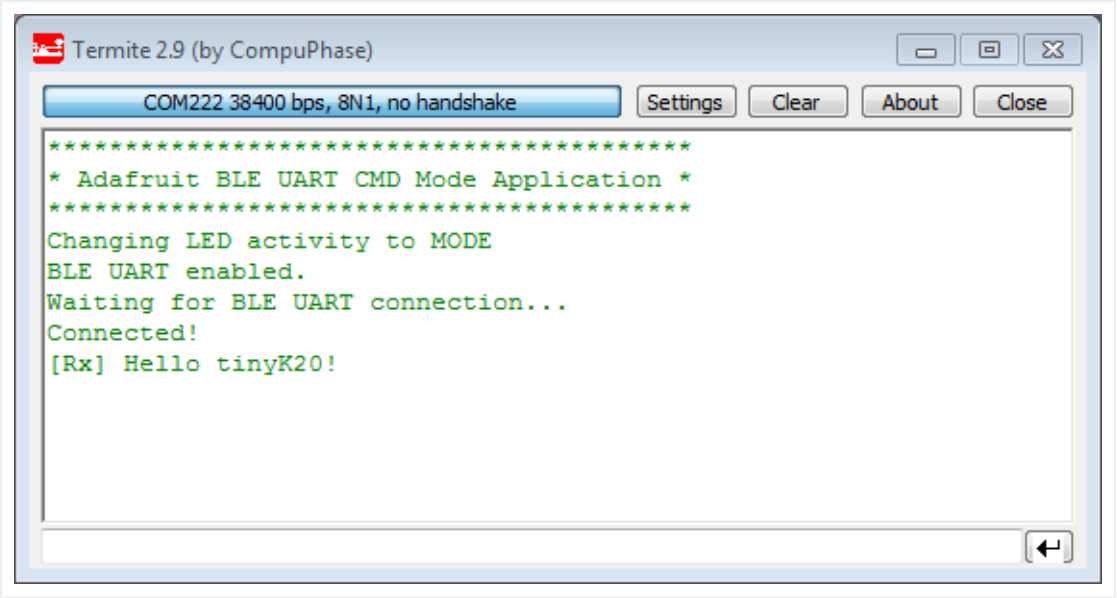


Connecting to the UART service gives me a terminal view where I can send and receive text:


●●○○○ Sunrise
13:16
100 %
Disconnect
UART
MQTT X
Echo
ASCII
Hex
Hello tinyK20!
Send
1 2 3 4 5 6 7 8 9 0
- / : ; ( ) € & @ "
#+=
. , ? ! '
ABC
Leerzeichen
Return

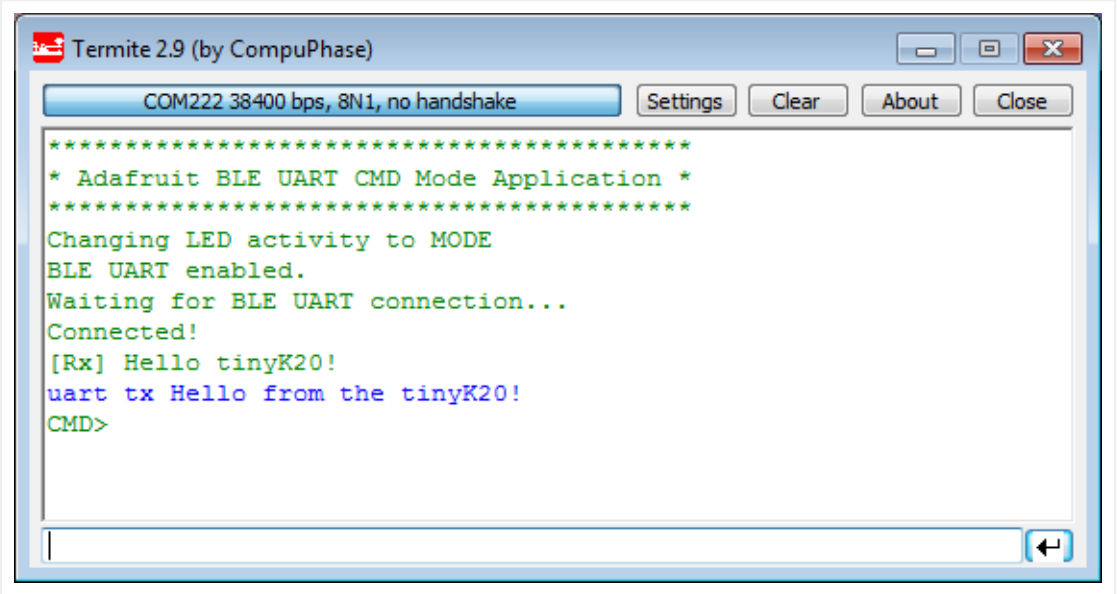
— BLE UART Application

The message is then received by the microcontroller and sent to the terminal connected to it:



— BLE UART Rx Message

The same way I can send a message from the microcontroller to the smart phone:



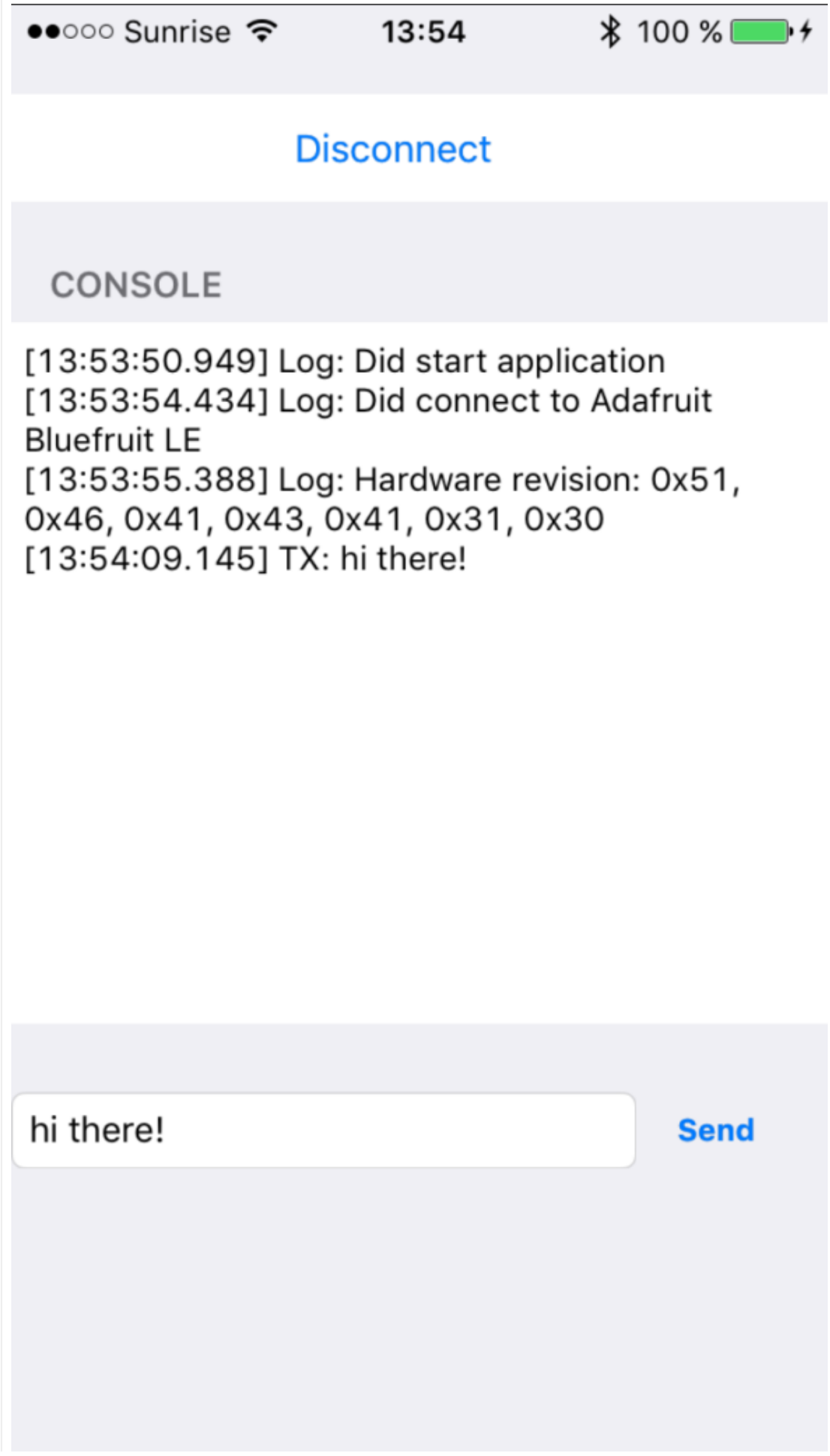
— Sending BLE UART message from Microcontroller

And the message is displayed in the smart phone application:


●●○○○ Sunrise
13:19
100 %
Disconnect
UART
MQTT X
Echo
ASCII
Hex
Hello tinyK20!
[Tx] Hello from the tinyK20!
Send
q w e r t z u i o p ü
a s d f g h j k l ö ä
y x c v b n m
123
Leerzeichen
Return

— Received BLE UART message

It works in a similar way with the nRF UART app from Nordic Semiconductor:


●●○○○ Sunrise
13:54
100 %
Disconnect
CONSOLE
[13:53:50.949] Log: Did start application
[13:53:54.434] Log: Did connect to Adafruit Bluefruit LE
[13:53:55.388] Log: Hardware revision: 0x51, 0x46, 0x41, 0x43, 0x41, 0x31, 0x30
[13:54:09.145] TX: hi there!
hi there!
Send

— nRF UART App

That way I can exchange status, commands and messages between the smartphone and the microcontroller over BLE connection.

Summary

With this I have a working UART-over-BLE connection between a smartphone and the ARM Cortex-M on the tinyK20 board . It is only a small step to extend the current application to a full UART-to-BLE bridge without the interactive part. And the BLE UART connection is only a start and much more to explore:

- Driving a Robot with BLE
- Add BLE connection to a quadrocopter
- Using the microcontroller as BLE mouse or keyboard
- Sending smartphone accelerometer and gyro data to the microntroller
- Using the BLE beacon mode (Apple iBeacon and Google EddyStone)
- Changing the color of LEDs with a color picker
- Read/Write microcontroller pins
- 3D printing a case for the tinyK20 + BLE module
- … and many more things which can be done with BLE ….

But these are all subject of further posts . Until then I will extend this BLE project which is available on GitHub.

Happy BLEing

Links

- Introduction to Bluetooth Low Energy: https://learn.adafruit.com/introduction-to-bluetooth-low-energy/introduction
- Getting started with Bluetooth Low Energy: https://www.adafruit.com/product/1978
- Bluetooth 4.x specification: https://www.bluetooth.com/specifications/adopted-specifications
- Bluetooth Developer Portal: https://www.bluetooth.com/develop-with-bluetooth
- Adafruit BLE Firmware on GitHub: https://github.com/adafruit/Adafruit_BluefruitLE_Firmware
- Adafruit BLE Friend USB dongle: https://www.adafruit.com/product/2267
- Adafruit BLE Friend BLE Sniffer: https://www.adafruit.com/products/2269
- Adafruit BLE Friend UART: https://www.adafruit.com/product/2479
- Adafruit BLE Friend SPI: https://www.adafruit.com/products/2633
- Adafruit BLE Friend Shield: https://www.adafruit.com/products/2746
- Adafruit Tutorial for BLE Friend SPI: https://learn.adafruit.com/introducing-the-adafruit-

bluefruit-spi-breakout

- Adafruit SDEP Description: https://github.com/adafruit/Adafruit_BluefruitLE_nRF51/blob/master/SDEP.md
- Project of this article on GitHub: https://github.com/ErichStyger/mcuoneclipse/tree/master/Examples/KDS/tinyK20/tinyK20_Adafruit_BLE

SHARE THIS:

Print Email Reddit Twitter Facebook LinkedIn 34

Tumblr Pinterest Google Pocket

★ Like

Be the first to like this.

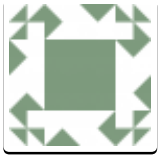
RELATED

Tutorial: Nordic Semiconductor nRF24L01+ with the Freescale FRDM-K64F Board
In "Boards"

FTF: FRDM-K64F, Kinetis Design Studio and Kinetis SDK
In "Boards"

Kinetis Lava LED Light Cube
In "3D Printing"

This entry was posted in **ARM**, **Boards**, **Building**, **CPU's**, **Eclipse**, **Embedded**, **FreeRTOS**, **Freescale**, **KDS**, **NXP**, **Processor Expert**, **Shell**, **SPI**, **TinyK20** and tagged **Adafruit**, **BLE**, **Bluetooth**, **Embedded Component**, **FreeRTOS**, **Freescale**, **NXP**, **open source projects**, **Processor Expert**, **software project**, **technology**, **Tips&Tricks** by **Erich Styger**. Bookmark the **permalink [http://mcuoneclipse.com/2016/01/09/how-to-add-bluetooth-low-energy-ble-connection-to-arm-cortex-m/]** .



About Erich Styger

Embedded is my passion....

View all posts by Erich Styger →

13 THOUGHTS ON "HOW TO ADD BLUETOOTH LOW ENERGY (BLE) CONNECTION TO ARM CORTEX-M"

Martin

on **January 9, 2016 at 20:54** said:

Hi Erich
Have you also considered using a wireless Kinetis (KW30Z)? Depending on the application, a two chip solution seems to be the right way but something which fits in a K20 would probably also fit in a KW30Z or the nrf51 directly…

★ Like

Erich Styger
on **January 10, 2016 at 11:09** said:

Hi Martin,
yes, I have considered the KW30 and KW40. But there are not modules available (unless I have missed them?). Using an integrated solution would lower the BOM costs, but for a small number of boards like I'm going to use the Bluefruit from Adafruit is more economical. And I had BLE up and running in less than one hour. Plus I could use the M0 on the BLE module alone too. But for now I prefer to have the M0 on the BLE module just to run the BLE stack. Using an extra microcontroller for the application provides me much flexiblity and more horse power.

★ Like

Roberto Paolinelli
on **January 10, 2016 at 11:13** said:

Same as Martin with me.
Why not using a single chip solution?
As for me, I'd prefer using a module when an existing project needs to be expanded, not to redesign it from scratch, otherwise if CPU power, memory, pins are enough, I'd like a single chip solution.

★ Like

Erich Styger

on **January 10, 2016 at 11:18** said:

Hi Roberto,
yes, in general I prefer a single chip solution too. I have used single chip solutions for ZigBee and IEEE802.15.4 in other projects. It greatly reduces the PCB size and BOM. On the downside it puts me into a dependency with the vendor: having an external transceiver provides more flexbility as I can use a smaller or larger microcontroller depending on my needs, but I can use the same transceiver for multiple projects. As always, it depends on the requirements.

★ Like

Mat.alm
on **January 10, 2016 at 11:52** said:

It's possible to use nrf24l01 as a BLE beacon with a dirty hack 😬.
Nice to see tiny k20 in action!

★ Like

Peter
on **January 12, 2016 at 12:09** said:

have a look:
http://dmitry.gr/index.php?r=05.Projects&proj=11.%20Bluetooth%20LE%20fakery

★ Like

Erich Styger
on **January 12, 2016 at 12:25** said:

Hi Peter,
thanks for that link, so this means I really have to try this out with my nRF modules. Have you tried it out

already?

★ Like

Peter
on **January 12, 2016 at 14:07** said:

this is also worth a read:
http://simonebaracchi.eu/posts/temperature-beacon/

★ Like

Peter
on **January 12, 2016 at 14:10** said:

Hi Erich,

yes i tryed it some years ago. And it really worked
I was basically using this library:
https://github.com/floe/BTLE

I have an android smartphone. I was able to see the transmitted beacons with this app:
https://play.google.com/store/apps/details?id=no.nordicsemi.android.mcp&hl=de

when I tryed, I was not able to send advertisements from a linux pc/android phone OR receive them with the nrf24l01. Did not figure it out.

this is really interesting. I was thinking of a weighing system for my plants so i can check the state of watering on my phone….

★ Like

Peter
on **January 12, 2016 at 14:27** said:

somehow my comment did not show up after writing it .
yes, I tried it and it worked really well.

★ Like

Pingback: Kinetis Lava LED Light Cube | MCU on Eclipse

ru4mj12 (@ru4mj12)
on **February 17, 2016 at 20:49** said:

Thanks for posting! I've been looking for a low-cost way of adding wireless communication to a cheapy microcontroller. The data rate using the SPP profile (as opposed to the ftp profile) doesn't lend itself for very fast transfers, but better than nothing!

I'm wondering though if designing a board using a micro with built-in Bluetooth capabilities is cheaper than designing with a 50 cent micro + Bluetooth module. I would imagine the bill-of-material for the adafruit module (radio, etc) can be incorporated onto a board for about $5. Any thoughts?

★ Like

Erich Styger
on **February 17, 2016 at 20:54** said:

It all will depend on number of boards, and if you count in certification, shielding, failure rate, margin, etc.
To save costs, you might be better off to transfer the application logic of your micro into that BLE module (as it already has a microcontroller in it). Such combined solutions (like the WS2811, http://mcuoneclipse.com/2014/10/15/cheap-and-simple-wifi-with-esp8266-for-the-frdm-board/) are available today,

and the future (not WiFi, because too power-hungry).
The costly part with wireless solutions is the certification with the regulation laws: this can easily cost you $10K of money.

★ Like