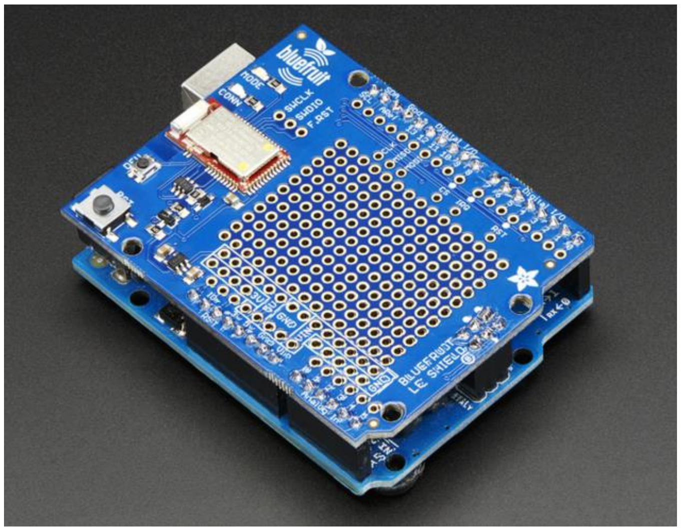
Adafruit Bluefruit LE Shield



1) Technical Specifications

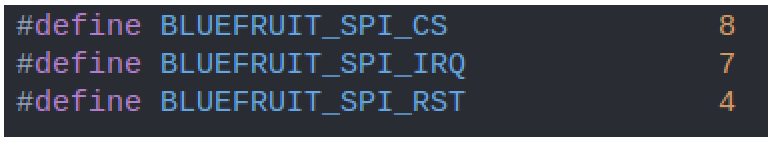
- ARM Cortex M0 core running at 16 MHz
- 256 KB flash memory
- 32 KB SRAM
- Transport: SPI at 4MHz
- On-board 3.3V voltage regulation
- Easy AT command set to get up and running quickly

2) Configuration

Bluefruit LE Shield has an SPI-connected NRF51822 module. You can use this with Atmega328 (Arduino UNO or compatible), ATmega32u4 (Arduino Leonardo or compatible) and possibly others.

The pinouts are Hardware SPI, CS =8, IRQ = 7, RST =4

You should change the location of the CS, IRQ or RST pins by opening **BluefruitConfig.h** file. You need to change the pin to an appropriate value



```
#define BLUEFRUIT_SPI_CS              8
#define BLUEFRUIT_SPI_IRQ             7
#define BLUEFRUIT_SPI_RST             4
```

3) SPI Pins

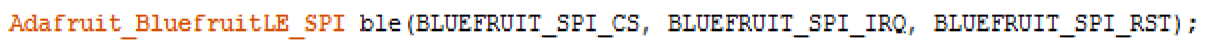
- **SCK**: This is the serial clock pin, by default connected to the Hardware SPI clock pin on the 2x3 ICSP header
- **MISO**: This is the Master In Slave Out SPI pin (nRF51 → Arduino communication) by default connected to the Hardware SPI MISO pin on the 2x3 ICSP header
- **MOSI**: This is the Master Out Slave In SPI pin (Arduino → nRF51 communication) by default connected to the Hardware SPI MOSI pin on the 2x3 ICSP header
- **CS**: This is the Chip Select SPI pin, which is used to indicate that the SPI device is currently in use. By default connected to digital #10
- **IRQ**: This is the nRF51 → Arduino 'interrupt' pin that lets the Arduino or MCU know when data is available on the nRF51, indicating that a new SPI transaction should be initiated by the Arduino/MCU. By default connected to digital #7
- **RST**: Holding this pin low will put the Bluefruit module into reset. By default connected to digital #4

4) Initial Bluetooth module, create service and add characteristic

- <u>Create the bluefruit object.</u>

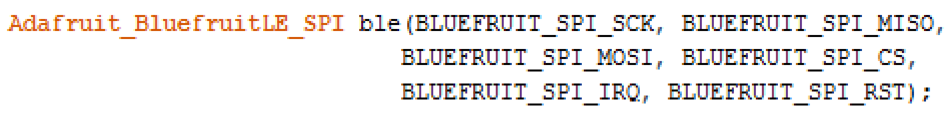
  You can communicate with the Bluefruit LE: hardware SPI or software SPI.

  a) If you want to use hardware SPI:



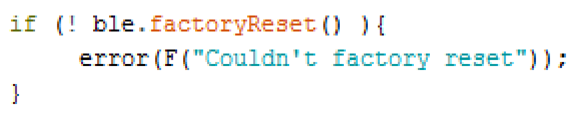
```
Adafruit_BluefruitLE_SPI ble(BLUEFRUIT_SPI_CS, BLUEFRUIT_SPI_IRQ, BLUEFRUIT_SPI_RST);
```

  b) If you want to use software SPI:



```
Adafruit_BluefruitLE_SPI ble(BLUEFRUIT_SPI_SCK, BLUEFRUIT_SPI_MISO,
                             BLUEFRUIT_SPI_MOSI, BLUEFRUIT_SPI_CS,
                             BLUEFRUIT_SPI_IRQ, BLUEFRUIT_SPI_RST);
```

- <u>Perform a factory reset to make sure everything is in a known state</u>



```
if (! ble.factoryReset() ){
     error(F("Couldn't factory reset"));
}
```

- Print bluefruit information

```
/* Print Bluefruit information */
ble.info();
```

- Set the device name

```
if (! ble.sendCommandCheckOK(F("AT+GAPDEVNAME=SHAF Bluetooth")) ) {
  error(F("Could not set device name?"));
}
```

Explain:

a) Send a command from a flash string

```
bool Adafruit_BLE::sendCommandCheckOK(const char cmd[])
```

b) AT+GAPDEVNAME

Gets or sets the device name, which is included in the advertising payload for the Bluefruit LE module

- Create the service

```
success = ble.sendCommandWithIntReply( F("AT+GATTADDSERVICE=UUID=0x180D"), &ServiceId);
if (! success) {
  error(F("Could not add SHAF bluetooth service"));
}
```

Explain:

a) Send a command from a SRAM string, and parse an int reply

```
bool Adafruit_BLE::sendCommandWithIntReply(const char cmd[], int32_t *reply)
```

b) AT+GATTADDSERVICE

Adds a new custom service definition to the device

Parameters: This command accepts a set of comma-separated key-value pairs that are used to define the service properties. The following key-value pairs can be used:

UUID: The 16-bit UUID to use for this service. 16-bit values should be in hexadecimal format (0x1234)

UUID128: The 128-bit UUID to use for this service. 128-bit values should be in the following format: 00-11-22-33-44-55-66-77-88-99-AA-BB-CC-DD-EE-FF

Response: The index value of the service in the custom GATT service lookup table. (It’s important to keep track of these index values to work with the service later)

- <u>Add the reps count characteristic</u>

```
success = ble.sendCommandWithIntReply( F("AT+GATTADDCHAR=UUID=0x2A37, PROPERTIES=0x10, MIN_LEN=1, MAX_LEN=2, VALUE=0"),
  if (! success) {
  error(F("Could not add reps count characteristic"));
}
```

Explain:

a) AT+GATTADDCHAR

Adds a custom characteristic to the last service that was added to the peripheral (via AT+GATTADDSERVICE)

Parameters: This command accepts a set of comma-separated key-value pairs that are used to define the characteristic properties. The following key-value pairs can be used:

UUID: The 16-bit UUID to use for the characteristic (which will be insert in the 3rd and 4th bytes of the parent services 128-bit UUID). This value should be entered in hexadecimal format (ex. 'UUID=0x1234'). This value must be unique, and should not conflict with bytes 3+4 of the parent service's 128-bit UUID.

Properties: The 8-bit characteristic properties field, as defined by the Bluetooth SIG. The following values can be used:

0x02 – Read

0x04 – Write Without Response

0x08 – Write

0x10 – Notify

0x20 – Indicatte

MIN_LEN: The minimum size of the values for this characteristic (in bytes, min = 1, max = 20, default = 1)

MAX_LEN: The maximum size of the values for the characteristic (in bytes, min = 1, max = 20, default = 1)

Value: The initial value to assign to this characteristic (within the limits of 'MIN_LEN' and 'MAX_LEN'). Value can be an integer ("-100", "27"), a hexadecimal value ("0xABCD"), a byte array ("aa-bb-cc-dd") or a string ("GATT!").

Response: The index value of the characteristic in the custom GATT characteristic lookup table. (It's important to keep track of these characteristic index values to work with the characteristic later.)