Microcontroller documentation

Set-up instructions:

Although the Sparkfun breakout board itself supports Arduino compatibility, the example code focused on using the Bluetooth peripheral and didn't help with other peripherals that are necessary for our project. Rather than continue working with the already frustrating bootloader on the breakout board, we were able to connect the nRF52832 hardware development kit to the breakout board and directly program the breakout board via the SWD protocol.

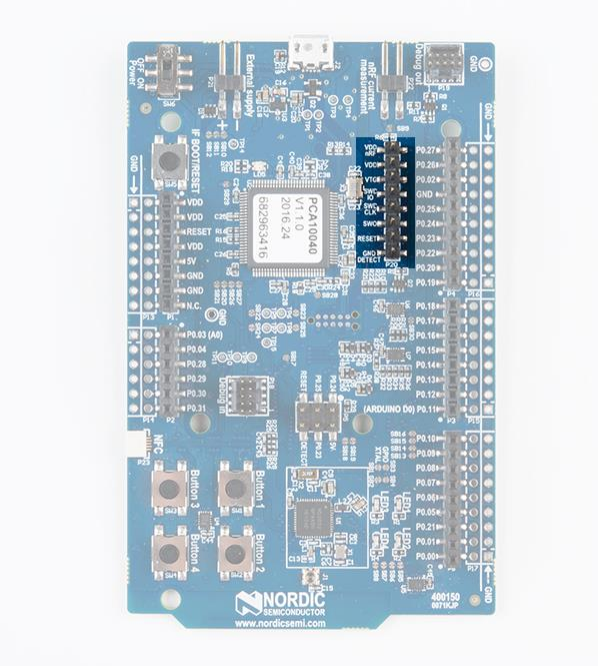

Figure 2 The nRF52832 hardware development kit. The highlighted area of the board shows what pins are used to connect to the breakout board for direct programming. This picture was taken from Sparkfun's hookup guide.

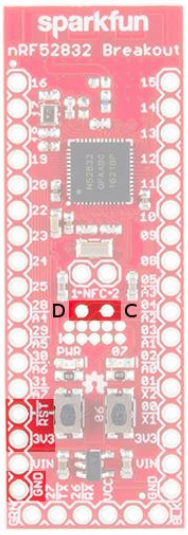

Figure 1 The nRF52832 breakout board. The highlighted areas show what pins are used to connect to the hardware development kit. This picture was taken from Sparkfun's hookup guide.

To begin programming onto the development kit hardware, the nRF software development kit (SDK) had to be installed. The SDK offers a variety of drivers, libraries, and examples for programming and can be downloaded as a .zip archive. Since the SDK is contained in a .zip archive, various IDE and compilers can be used to run code. We chose to use SEGGER embedded studio as our IDE. In addition to downloading the SDK and the IDE, a SoftDevice was needed for bluetooth capability.

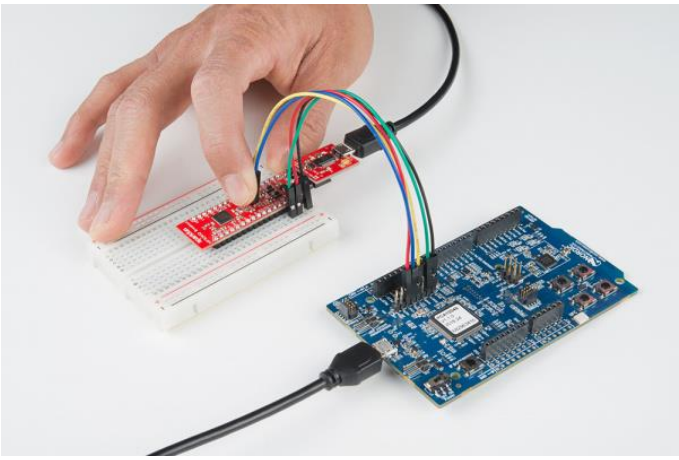

Figure 3 The boards are connected. Using 5 Male/Female wires, the voltage, reset, and ground are connected on both boards as well as the SWDIO and SWDCLK. This picture was taken from Sparkfun's hookup guide.

To run example programs, the SDK folder contains various example programs for each peripheral, and the program for our specific board can be opened with the SEGGER embedded studio. The program can then be built and downloaded onto the hardware development kit.

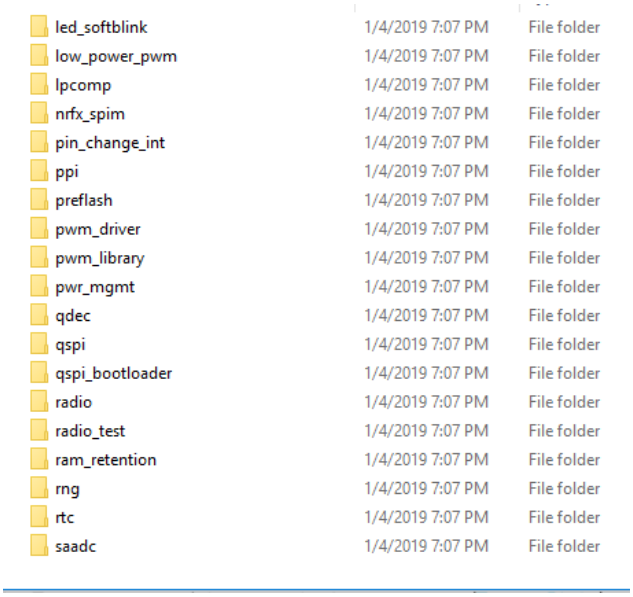

Figure 4 Showing just a few example programs for peripheral devices. There are also many examples available for Bluetooth.

To create a program for our project, a new project was created, existing code to help with our project was used, and new code was written in C. The various peripheral drivers and libraries from Nordic were used to help accomplish our project goals. It is important to note that since we are programming onto the breakout board, some of the pin connections had to be remapped to match the pins of the breakout board.

Test plan:

1. Power each board

The first step to programming the breakout board to fit our project needs was to successfully connect and power each board. The Sparkfun breakout board had to have pins soldered onto it and connected to a breadboard to allow for any connections. In addition, a separate FTDI module had to be used with either a micro USB or mini USB cable connection. Similarly, the hardware development kit was powered via a micro USB connection.

Initially, there was a problem connecting with the hardware development kit board. Instead of the green power LED turning on, it had continuously flashed. After a bit of troubleshooting, it was discovered that the USB cable didn't allow for the necessary connections, and the board connected properly with a different cable.

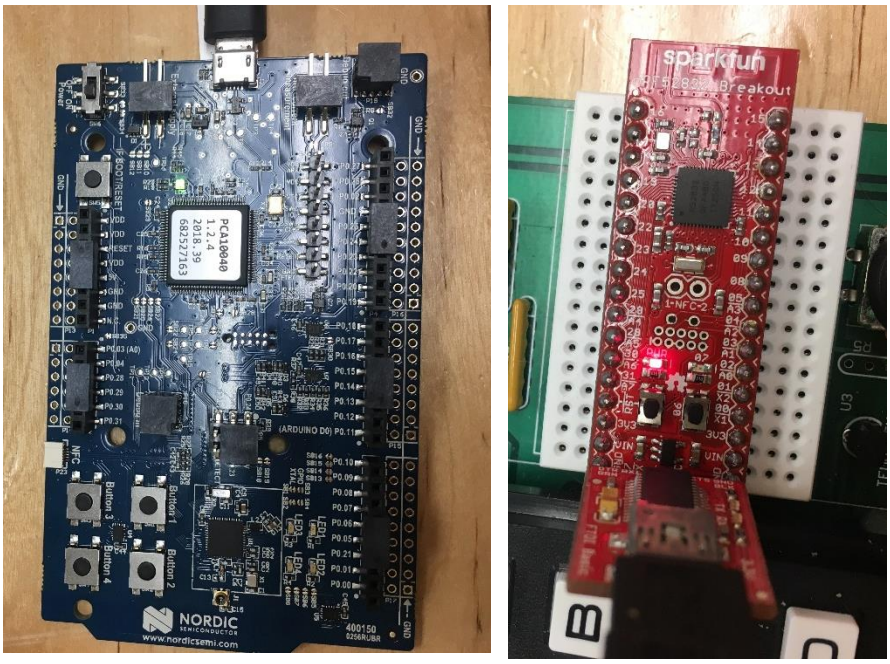

Figure 5 Both boards are properly powered on and connected as shown by the power LED light being active (Green on left, red on right).

2. Connect boards together

The next step to programming the breakout board is to make the necessary connections between the hardware development kit and the breakout board. This allows for the Nordic SDK programs to be flashed onto the breakout board directly. This is done with the SWD protocol, and the SWDIO and SWDCLK from the hardware development kit is connected to the breakout board.

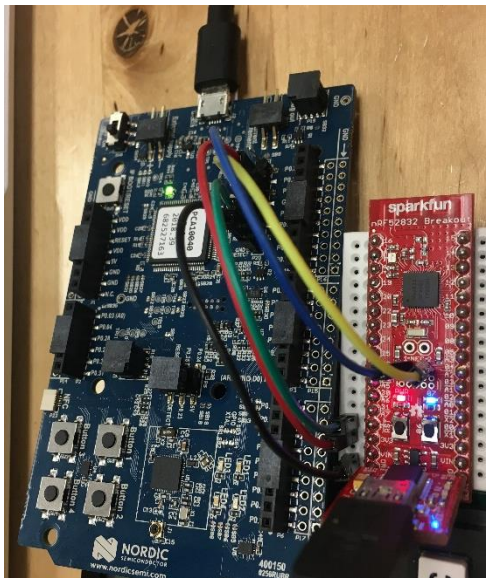

Figure 6 The connections mentioned in the set up section are made to allow for direct programming onto the breakout board.

3. Run example programs on each board

Next, simple example programs were run on each board to ensure that they could properly run code. The hardware development kit was programmed directly separate from the breakout board and only needed the micro USB connection. The breakout board had to be connected to the hardware development kit as well as powered on with the mini USB cable. After the program is flashed to the breakout board, it no longer has to be connected to run the program.

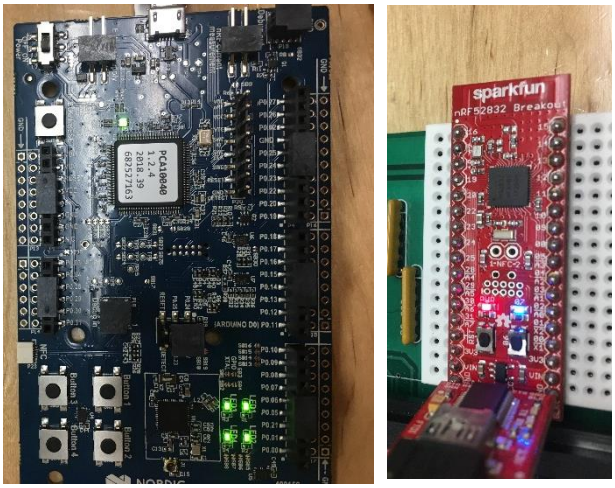

Figure 7 Example programs are writen to each board. Both of them toggle LED's on and off using a timer (green on left, blue on right).

4. Example program with ADC peripheral

An important aspect of our overall project is to be able to read voltages created from the sensors in the shoe. To do this, the sensors can be connected to an ADC unit on the microcontroller. To begin working with the ADC units, an example program was created to make sure that the ADC unit was reading voltages correctly.

A custom program was written using the ADC and the LED light. The LED light would turn on when voltage was applied to the ADC, and would turn off when no voltage was being applied. We initially had problems with toggling the LED light but quickly realized the LED was active low and we had to adjust accordingly.

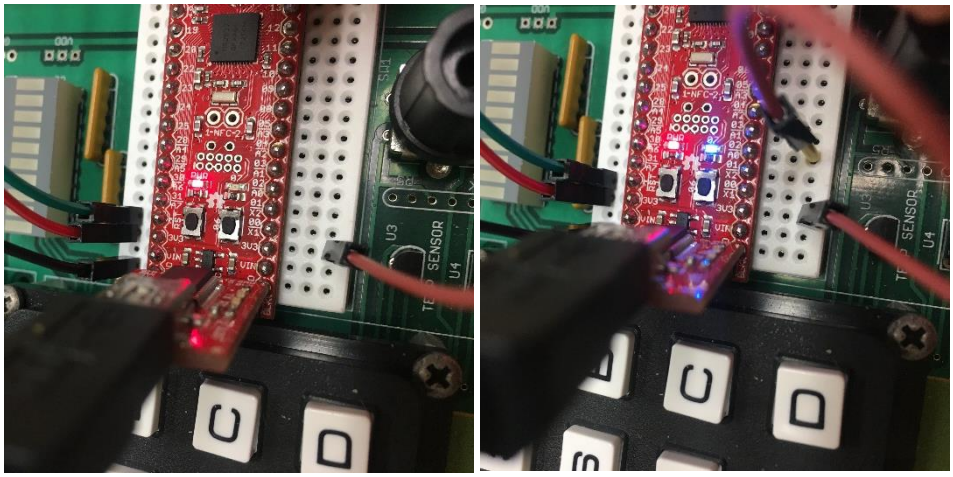

Figure 8 The ADC sample program. The brown wire is connected to a 3.3 volt output. When the ADC is not connected to any voltage, the LED is off (left). When a voltage is applied to the ADC, the LED turns on (right).

5. Example program with BLE peripheral

Another important aspect of our overall project is to be able to send data via Bluetooth to a mobile device. This will allow the microcontroller to continuously transmit step data as long as the microcontroller remains powered. Luckily, Nordic provides many example programs and libraries for Bluetooth capabilities.

To connect with the breakout board through Bluetooth, a Bluetooth connection app needed to be downloaded. Nordic provides an nRF Connect application to connect with any nRF microcontroller and is available on desktop and mobile devices. For displaying purposes, we used the desktop version of nRF Connect. The desktop version connected through the hardware development kit and was able to search for and connect with our breakout board.

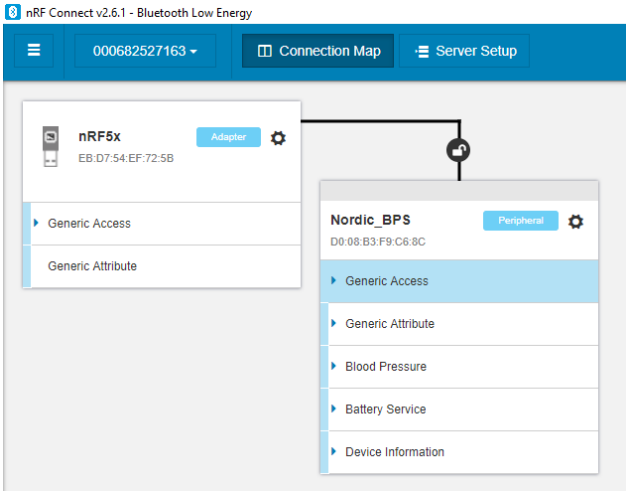

Figure 9 An example Bluetooth program that successfully connects with our breakout board. Various descriptions of our breakout board can be seen through the Bluetooth connection.

6. Our program (sends ADC data over BLE)

Finally, the last step for the microcontroller to work for our project was for the microcontroller to be able to send ADC data over a Bluetooth connection to the mobile device. A circuit was built to allow for the force sensor to create a voltage that would then be read by the ADC.

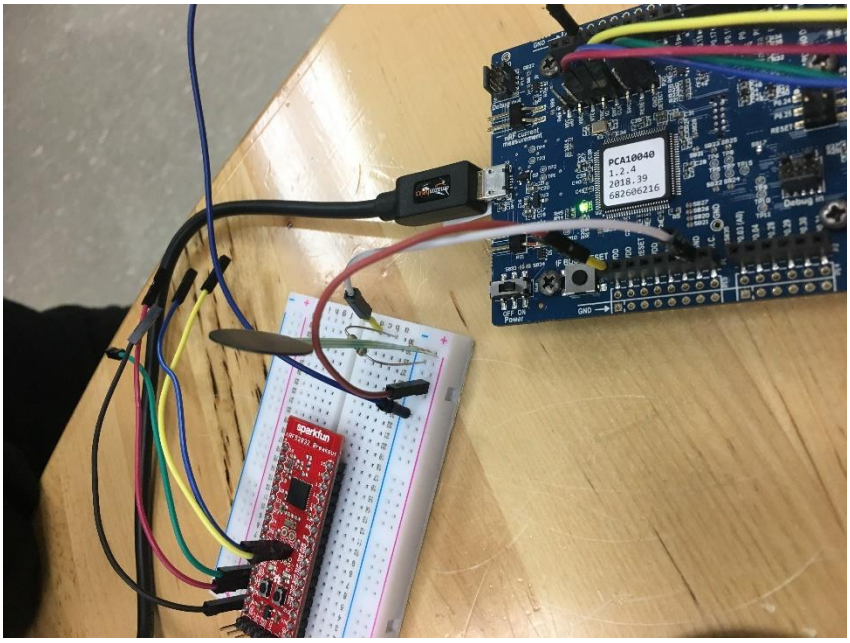

Figure 10 The circuit created to allow the force sensor to create voltages. The force sensor is connected to a voltage source and a 10k ohm resistor, and the resistor is connected to ground. The ADC is then connected between the force sensor and the resistor to read voltages created by the sensor.

After the circuit was built to allow for the ADC to read voltages, the ADC values had to be sent over Bluetooth. A Bluetooth connection was established between the breakout board and the nRF Connect app, and values from the ADC peripheral were continuously sent over Bluetooth.

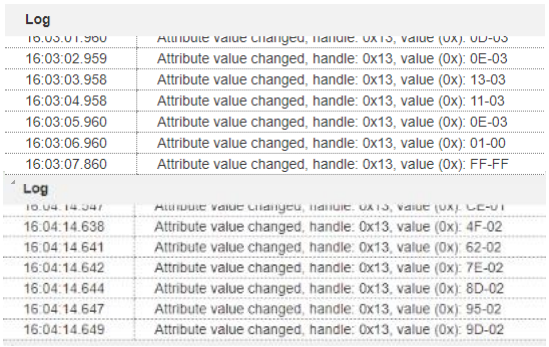

Figure 11 A few examples of values being read from the ADC through the nRF Connect app. The ADC values change based on the amount of force applied to the force sensor.

There were many problems encountered when we attempted to combine the ADC examples with the Bluetooth examples to meet our project needs. In general, there were many problems that had to do with the sdkconfig.h file. That file declares various things about the nRF52 microchip such as, for example, what timers are enabled, or the functions enabled of that timer. There were many things that came up that were not immediately obvious that had to do with settings in the sdkconfig.h.

One major problem that occurred had to do with a timer being used by multiple microchip systems. What happened was the SoftDevice, which controls the nRF52 Bluetooth capabilities, used a timer, and when the program attempted to use the same timer for the ADC peripheral, the system would say “Error” and reset itself. This was not noted anywhere easily visible.

Another error came in with making our own Bluetooth service and characteristics. Every Bluetooth service needs its own UUID, or universal unique identifier. Many are already defined by the Bluetooth SIG, and are 16-bit, but those that aren’t, like ours, need to have a special 128-bit one created for them. Nordic’s sdk allows for the creation of them, but they don’t easily tell everything that has to be changed to make them work. In order to make the Bluetooth vendor specific ID, there are two things outside of the source code that need to be changed. The first is in the sdkconfig.h file, where you need set the actual number of vendor specific IDs allowed. Then you need to change go into the project settings, use the common configuration,

and change the linker memory placement macros to allow for space in memory being taken up by your vendor specific IDs. This was also not easily visible in the documentation used.