Developing Software & Algorithms For Totem Open-Health

Table of Contents

Board Components:

Part	Name
MCU	NRF51822-**QFAA**
IMU	MPU6050
STORAGE	MCU FLASH & SD-Card
TEMPERATURE SENSOR	TMP102

Interfaces:

From	To	Type	Maximum throughput
MCU	STORAGE (SD-Card)	SPI	<TBD>
MCU	IMU	I2C / Software I2C	<TBD> / <TBD>
MCU	TEMPERATURE SENSOR	I2C / Software I2C	<TBD> / <TBD>
MCU	Outside world	Bluetooth Low Energy	1.5 kb/s
MCU	Outside world	Blue Light Emitting Diode	Not Applicable

Required tools for firmware development:

Tool	Function
MBED	Online development environment and compiler toolchain. While it is not a hard requirement to use MBED for new firmware development, we highly recommend using this tool.
J-LINK Programmer	Recover totems after an invalid or failed firmware upload procedure and speed up development by improving the design-compile-upload process over regular Over The Air(OTA) uploading. Get one here: http://eud.dx.com/product/j-link-v8-arm-usb-jtag-adapter-emulator-black-844149039#.VgGeBLTqeTU
Bluetooth Low Energy capable device	For uploading new firmware and testing bluetooth low energy stack functionality.

Components - MCU

WARNING, the QF<XX> section in the MCU name is important, only upload code compiled for the QFAA version of the NRF51822. There are minor hardware radio section changes, that causes software compiled for a different version to behave erratic or produces hard-faults. Not adhering to this requirement could render your device **inoperable** if Bluetooth Low Energy is your primary means of uploading new Firmware.

MCU Properties

Flash	256 kb -> 128 kb usable
RAM	16 kb -> 8 kb usable
Peripherals	1x SPI, 1x USART, 1x I2C
Power	1 mA, Active idle. 2.6 uA sleep.
Timers	1x 32 Bit, 2 x 16 Bit, 1x RTC

MCU Description

The NRF51822 is a Bluetooth Low Energy SoC created for low cost, low power applications. It differentiates itself from more common microcontrollers (PIC, Atmega, MSP430) by including a bluetooth low energy radio directly on the same silicon. While this has advantages (volume, price, power, complexity) over a design where the radio and processor are separate devices, it also carries a downside, application flow will be interrupted to handle soft device functions.

A word you will come across a lot when developing for this MCU is: softdevice. It can best be explained as a software bluetooth stack that operates on the internal radio hardware of the NRF51822. It is important to remember that this resides in the MCU FLASH, thus taking up space for user applications. The RAM is also shared between the soft device and user applications. The architecture of the system memory can be found in Figure 1. The memory of the MCU is split in two sections, first the soft device that depending on the version takes up to 128 kb FLASH and 8 kb of RAM.

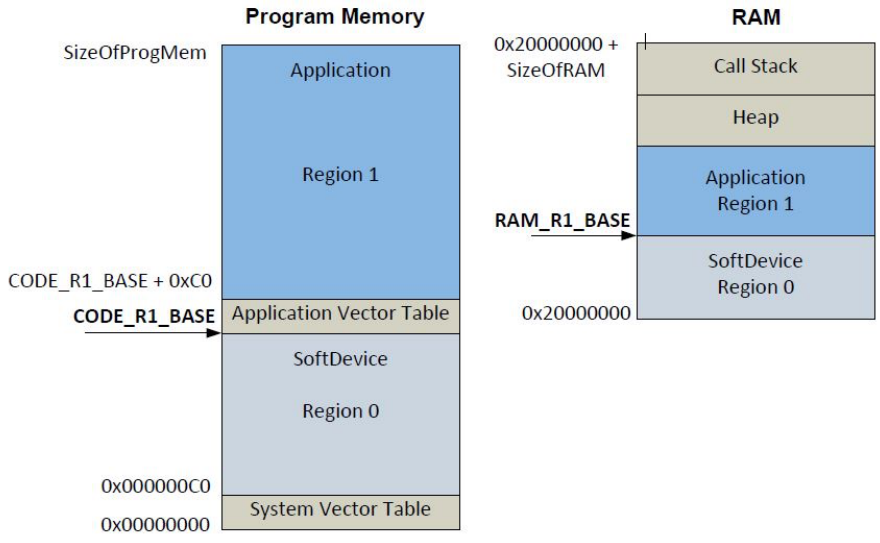


Figure 1.

Components - MCU Radio

Radio Properties	
type	Bluetooth Low Energy 4.0 & ANT+
Output power	-23 dBm to 4 dbm
Antenna Gain	-1.5 dBm
Range	<tbd> Meters Line of Sight

Radio properties

The radio is controlled by the soft device. The totem is pre-loaded with the bluetooth low energy soft device, this however can be changed with a J-Link programmer and the nRFGoStudio.

Bluetooth low energy provides a convenient method for establishing connections and data sharing between a peripheral (totem) and a central (smartphone / computer) device.
The peripheral exposes services to the central device. The services contain characteristics(variables). The central device can request updates of the characteristics on value change. The peripheral can update a characteristic value to send data to the central device. The central device is also able to change the characteristic value in order to send data to the peripheral device.

The radio and soft device are abstracted by a bluetooth low energy API that is accessible by code. Totem comes preconfigured with two services. The DFU(Device firmware upgrade) service and the communication service. The DFU is used to update the totem firmware over the air using bluetooth and the Nordic Master Control Panel App on Android or IOS. The communication service contains a single characteristic that is used for the transmission and reception of data.

More information on the Bluetooth Low Energy API can be found here:
https://developer.mbed.org/teams/Bluetooth-Low-Energy/

Components - IMU

IMU Properties

Sample-rate	1.25 - 1000 Hz
Accelerometer	3 Axis, 16G
Gyro	3 Axis, 16.4 LSB/deg/sec
Interface	I2C
FIFO	1024 bytes
Power consumption	Gyro Off: 1,25 Hz -> 10 uA 5 Hz -> 20 uA 20 Hz -> 60 uA 40 hz -> 110 uA > 40 hz -> 500 uA Gyro On: 3.6 mA + Gyro Off Accelerometer current

IMU Description

The MPU6050 is a versatile motion sensing device. It is very feature rich, but configuring it properly can be a challenge. It features an internal programmable DMP that is capable of performing advanced sensor fusion. In the regular totem firmware with standard settings enabled, all IMU features are turned off, and the accelerometer is sampling at 20 Hz.

IMU Interface

The I2C interface of the MPU6050 is abstracted by a library that handles all communication and data transfer. It should not be necessary to add or remove to this library as all possible device functions and configurations are present.

The IMU is connected to the MCU using the two mandatory I2C lines (Data and Clock) and an additional INT line for handeling Interrupts. This line is used as I2C has no means for a slave device (MPU6050) to signal a processor that new data is present.

The SDA line is connected to the P0_3 pin of the MCU,
The SCL line is connected to the P0_2 pin of the MCU,
The INT line is connected to the P0_29 pin of the MCU.

I2C 7 bit slave address: 0x69

Warning, earlier in this document it was stated that the NRF51822 shares its FLASH and RAM with the softdevice. The soft device WILL interrupt user code in order to handle all BLE related activities in a timely manner. This means that communication between MCU and IMU can be disrupted by BLE activities(For up to 5 ms). It appears that the I2C hardware implemented is not able to withstand these interruptions. The IMU could enter an erroneous state where only garbage data is produced instead of actual motion measurements. To circumvent this, all I2C communication in the totem are handled by a SOFTWARE I2C driver.

Components - Storage, SD-Card

Flash properties

Size	Depends on used soft device. At least 10 kb

SD-Card properties

Type	MicroSD
Class	<= 10
Interface	SPI
Power	100 mA Active writing. 100uA Sleep

Storage description

There are two methods of information storage available on the totem. The first and lowest power mode option is using the internal flash. Please keep in mind that FLASH cells have a limited amount of write cycles before they wear out, 10000 in our case. Use this memory only for data that changes with a low frequency. For example, you can use this memory to store the amount of free fall's detected during a day, or the total amount of footsteps in a day. Storing raw sensor measurements every time they arrive from the IMU would be an improper use of this memory type and will likely result in the device FLASH memory wearing out, resulting in defects, errors and unexplained behavior.

The second storage option is the optional SD-Card of arbitrary size. Use this memory if you need to store a vast amount of data, or data that changes in high frequency. Although an SD-Card is subject to the same write cycle limitations as the internal flash, it can be replaced by a new card if it wears out, and is thus the preferred storage option.

Storage Interface

The internal flash is accessed by a FLASH driver that was written to abstract the PStorage behavior of the soft device(the soft device is responsible for storage and retrieval of flash data). This driver does not make ANY assumptions about the format of your data, you are responsible for converting your data to and from a bitstream.

The totem is equipped with an SD-card driver that can handle all SD-Cards up to, and including class 10. On top of the SD-Card driver resides a FAT filesystem driver. This ensures that the data written to the SD-Card is readable by all common computer systems like win/lin/mac computers&laptops and microsoft,apple and google smartphones. All access is file based and has strong similarities with file handling in C on linux systems (fopen,fclose, fprintf, etc).

The SD-Card IO's are accessible on the MCU on pins:
MOSI - P0_23, MISO - P0_25, SCK - P0_24, SS - P0_22,
Card inserted - P0_1(Warning: USE AS DIGITALIN ONLY!, failing to do so can short-circuit the MCU if an sd-card is inserted. See the Code Examples - Storage SD Card).
WARNING, writing to an SD-Card poses strain on the battery that is used. It is recommended to minimize the amount of write/read operations in order to ensure that the longest possible measurement time from a single battery is achieved. Cache & accumulate data before writing!

Components - Temperature Sensor

Temperature Sensor properties

Accuracy	0.5 deg
Resolution	0.0625 deg
Interface	I2C
Power	10 uA Active, 1 uA Sleep

Temperature Sensor description

The TMP102 sensor is a simple I2C ultra low power temperature sensor.

Temperature Sensor interface

The totem is equipped with a driver for the TMP102. This abstracts all device communication and reduces it to a single function call <temperatureobj>.read().

The SDA line is connected to the P0_3 pin of the MCU,
The SCL line is connected to the P0_2 pin of the MCU,

I2C 8 bit slave address: 0x90

Warning: because the temperature sensor has to coexist with the IMU on the same bus, writes and reads to both devices have to be SEQUENTIAL. Although the regular I2C module could be used to read the temperature, it is recommended to reuse the already existing SOFTWARE I2C driver that is used for the MPU6050.

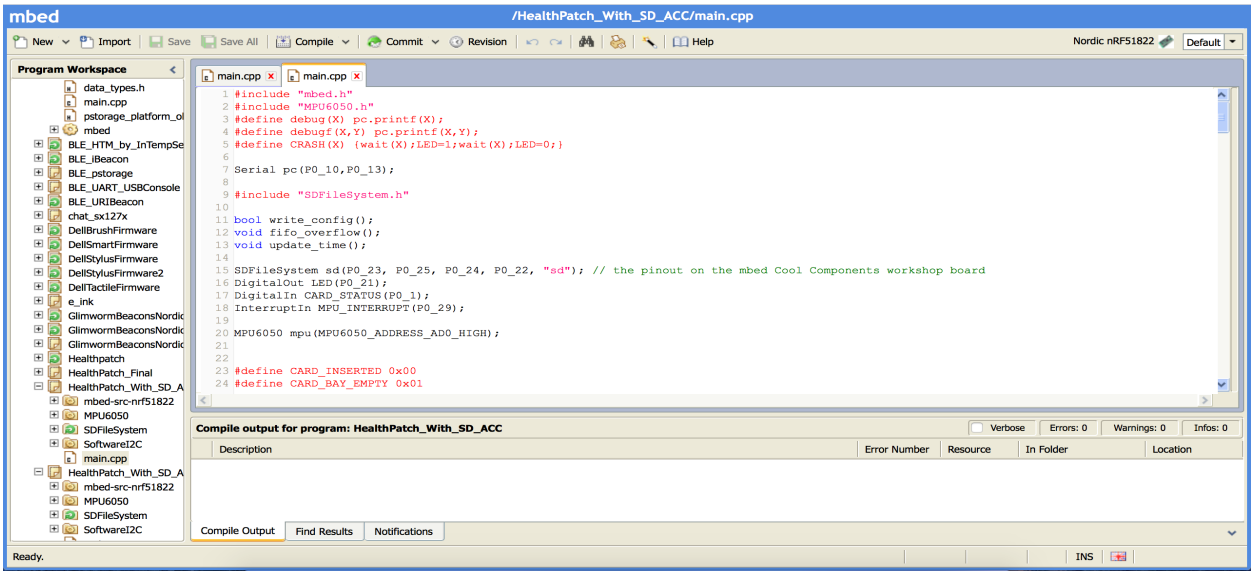
Programming the Totem with MBED

Prerequisites

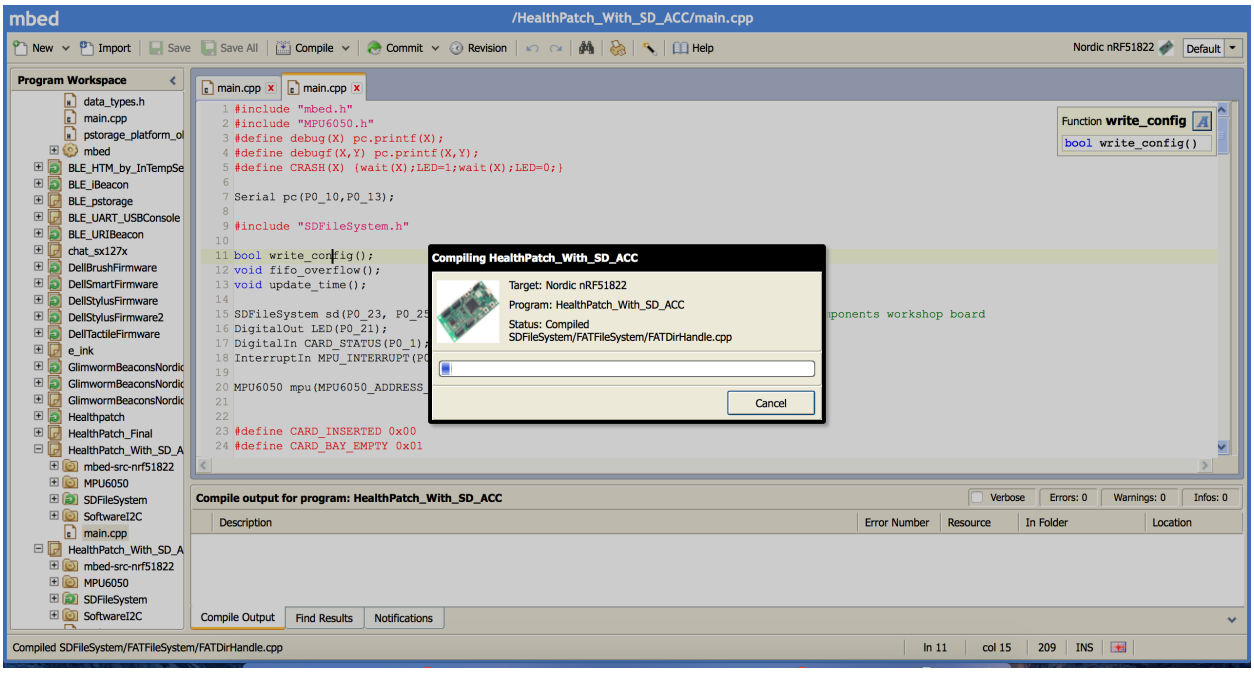
- MBED online compiler account,
- NRFGOStudio Installed,
- J-Link Edu programmer installed & attached to computer,
- Healthpatch programming rig.

Steps

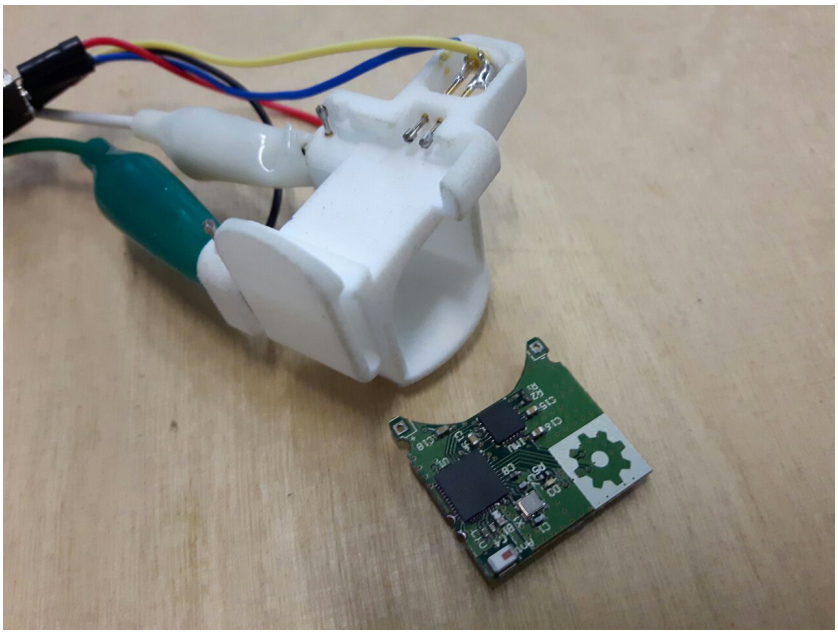
1. Write your code.

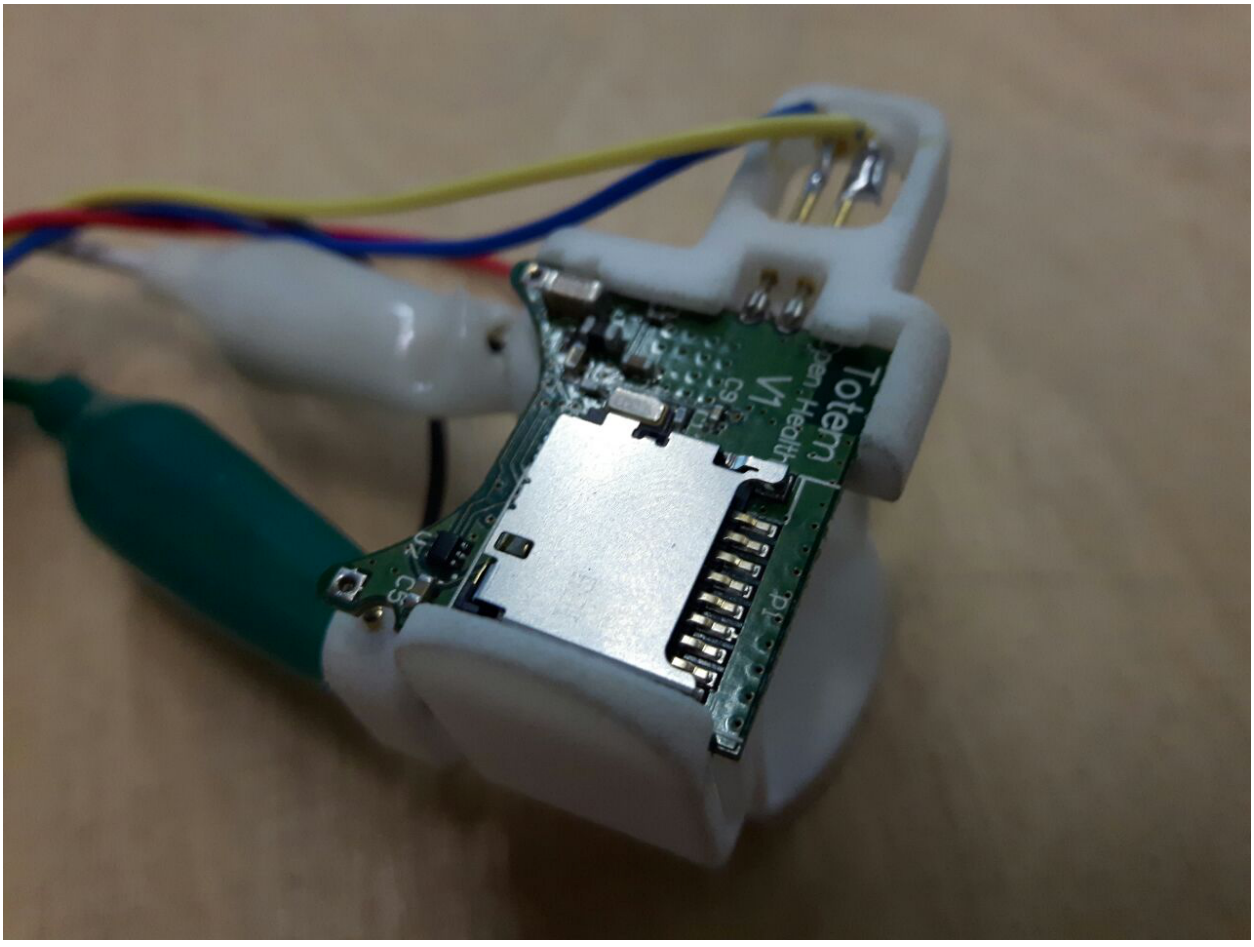


2. Press Compile in the top bar of the MBED online compiler. The compiler will generate an .hex firmware file that is automatically downloaded to your computers downloads folder.

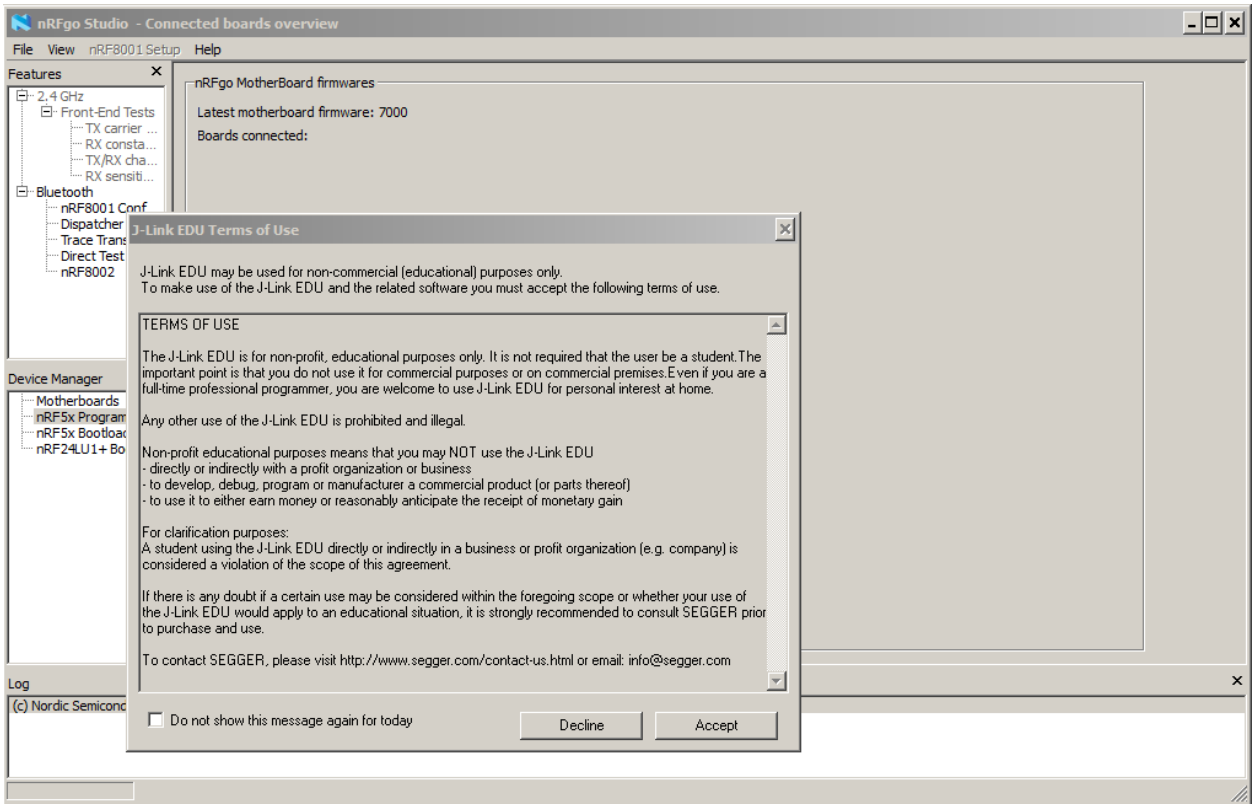


3. Insert the Totem in its programming rig, make sure all pogo pins connect to their respective socket on the totem. WARNING, THERE IS ONLY ONE POSSIBLE MOUNTING POSITION

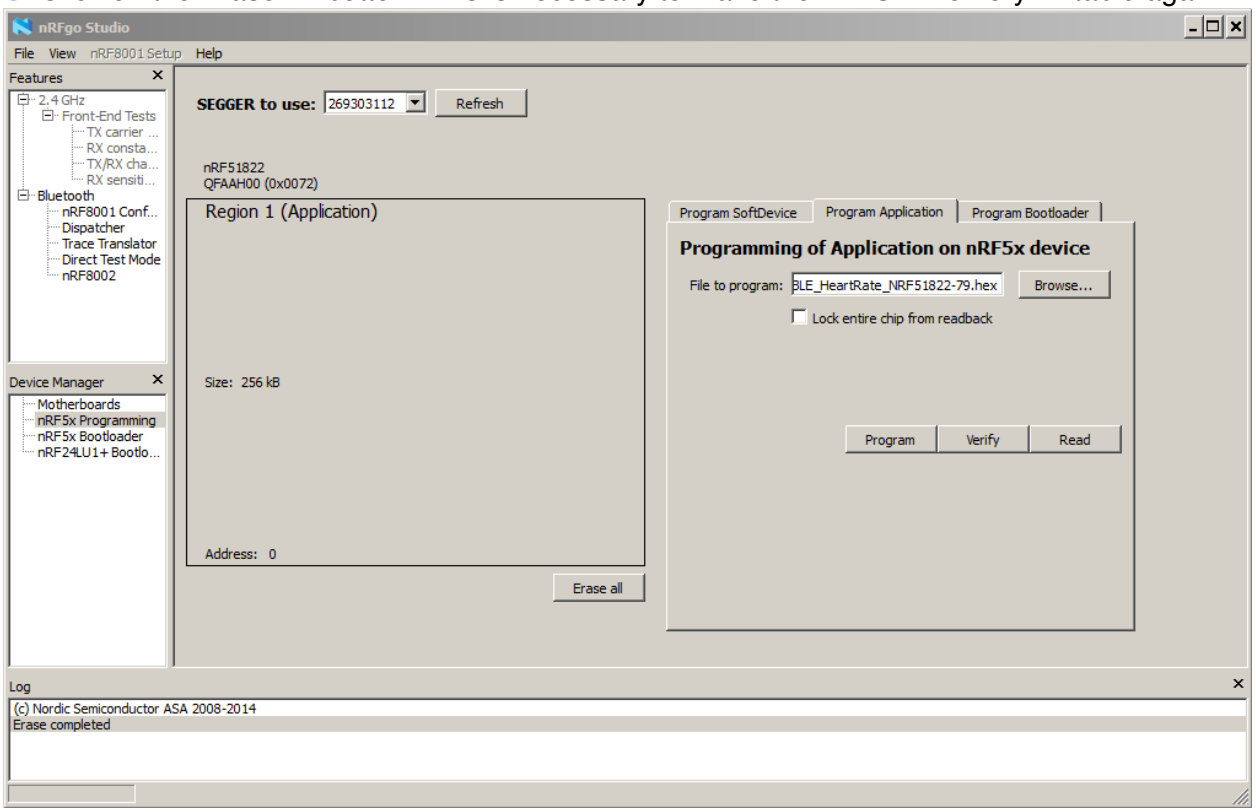




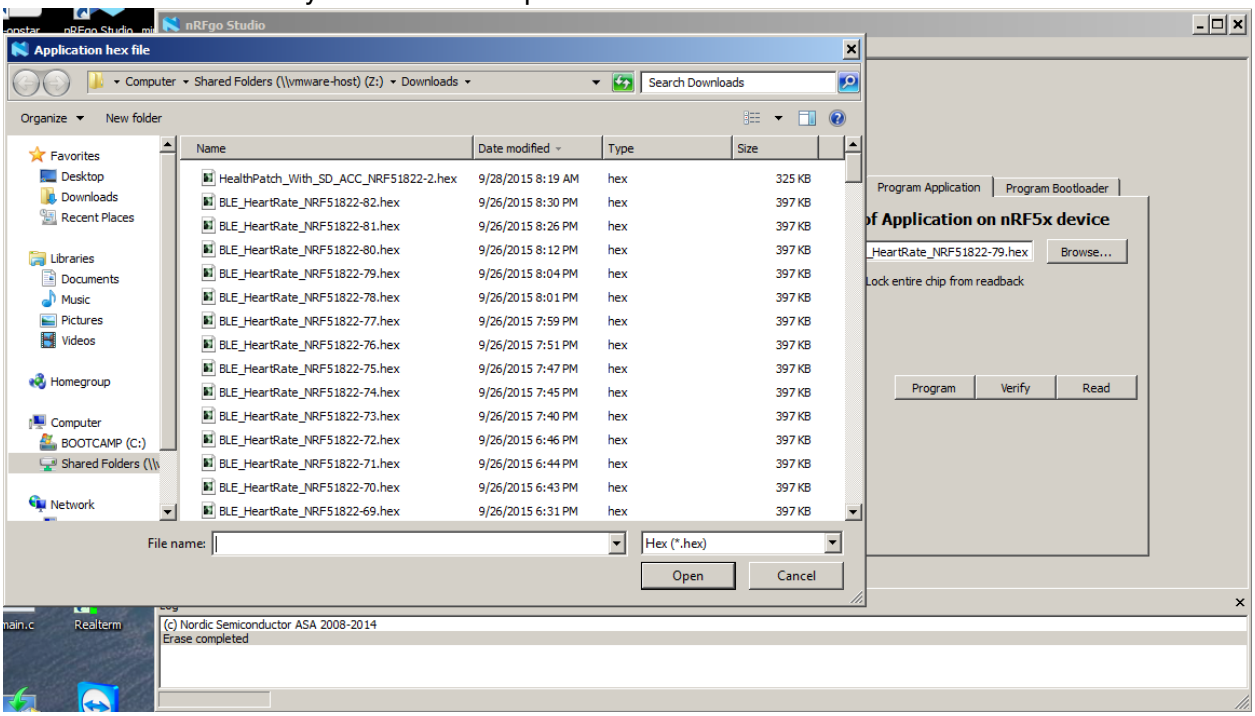
4. Open nRFgo Studio, and click on nRF5x Programming in the lower left selection box. If you use an J-LINK edu than you need to accept the license first. This popup will be shown once a day.



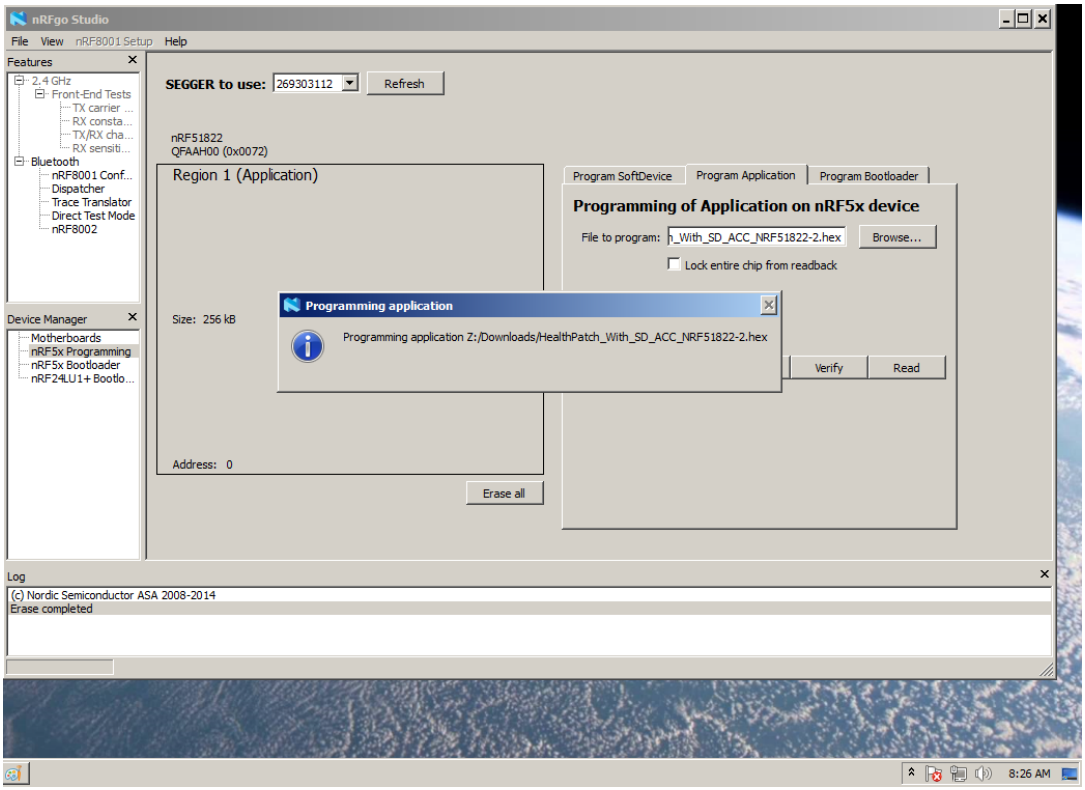
5. Click on the Erase All button. This is necessary to make the FLASH memory writable again.



6. Click on the ‘Program Application tab on the right. Then click Browse and select the .hex file that was downloaded by the MBED compiler.



7. Click on program to upload the new firmware to your totem.



8. Finished ;)

Code Examples - Blinking LED

```
#include "mbed.h"

DigitalOut LED(P0_21);

int main() {
    while(1)
     {
          LED = 1; // Turn LED on
          wait(1); //delay 1 second
          LED = 0  // Turn LED off
          wait(1);
     }
}
```

The following code can be used to blink the blue LED on the totem

Code Examples - Storage SD-Card Check

The following code can be used to check if an SD-Card is present.

```
#include "mbed.h"
#include "SDFileSystem.h"

#define CARD_INSERTED 0x00
#define CARD_BAY_EMPTY 0x01

DigitalIn CARD_STATUS(P0_1);

int main() {
    CARD_STATUS.mode(PullUp); // Pull up the control line
    wait(0.1);
    if(CARD_STATUS == CARD_BAY_EMPTY) // Check if control line
is pulled low, if so, a card is present in the bay.
    {
       // Perform steps card absent
    }
    else
    {
       // Perform steps card present
    }
    CARD_STATUS.mode(PullDown); // Change mode to pulldown
to save current.
}
```

Code Examples - Storage SD-Card

The following code can be used to create a folder ‘totem’ and then write a file to the folder called

```
#include "mbed.h"
#include "SDFileSystem.h"

SDFileSystem sd(P0_23, P0_25, P0_24, P0_22, "sd");

int main() {
    mkdir("/sd/totem", 0777);

    FILE *fp = fopen(“/sd/totem/totem.txt”, "w");
    if(fp == NULL) {
        error("Could not open file for write\n");
    }
    fprintf(fp, “It rocks! ;)”);
    fclose(fp);
}
```

‘totem.txt’ with the content ‘It rocks! ;)’
Warning: Please note the fclose at the end. It is MANDATORY to call this function BEFORE you power down the totem. Omitting to do so WILL result in the loss of the current opened document.

There are two approaches to circumvent the risk of data loss. The first is writing small files, so a powerloss causes only the last file to become corrupt. An alternative to this approach is using the on-board Power-On Failure peripheral. This peripheral can issue an interrupt when the device supply voltage drop’s below a certain level, you can then close the file there. This function is untested, its effectiveness depends on the quality of your sd-card(Under voltage lockout), the battery level before it was pulled and the code running currently on the device.

Code Examples - IMU basic

The following code can be used to read the acceleration and rotation directly without the interrupt line or FIFO buffer.

```
#include "mbed.h"
#include "MPU6050.h"

MPU6050 mpu(MPU6050_ADDRESS_AD0_HIGH);

int main() {
    mpu.reset();  // Reset the IMU
    wait(1);      // Wait for the reset to complete
    mpu.initialize(); // Initialize IMU
    bool mpu6050TestResult = mpu.testConnection(); // Check
connection with IMU
    if(!mpu6050TestResult)
    {
        // IMU IN UNKNOWN STATE, better take out the battery
        while(1); // HANG FOREVER;
    }

    while(1){
       int16_t ax;
       int16_t ay;
       int16_t az;

       int16_t gx;
       int16_t gy;
       int16_t gz;
       mpu.getMotion6(&ax,&ay,&az,&gx,&gy,&gz); // read the
acceleration and rotation in all axis.
    }
}
```

Code Examples - IMU Interrupt

If there is the need to perform other operations while there is no new accelerometer data, interrupts can be used. This example will set up the interrupt line and IMU to deliver interrupts when there is new data from the IMU.

```
#include "mbed.h"
#include "MPU6050.h"

void IMU_INT();

MPU6050 mpu(MPU6050_ADDRESS_AD0_HIGH);
InterruptIn MPU_INTERRUPT(P0_29);

int main() {
    mpu.reset();  // Reset the IMU
    wait(1);      // Wait for the reset to complete
    mpu.initialize(); // Initialize IMU
    bool mpu6050TestResult = mpu.testConnection(); // Check
connection with IMU
    if(!mpu6050TestResult)
    {
        // IMU IN UNKNOWN STATE, better take out the battery
        while(1); // HANG FOREVER;
    }
    MPU_INTERRUPT.mode(PullDown);
    MPU_INTERRUPT.rise(&IMU_INT);

    mpu.setSleepEnabled(false);
    mpu.setTempSensorEnabled(false);
    mpu.setInterruptMode(0);
    mpu.setInterruptDrive(0);
    mpu.setInterruptLatch(true);
    mpu.setInterruptLatchClear(true);
    mpu.setIntDataReadyEnabled(true);

    while(1){
       if(new_data)
          {
          new_data=false;
          int16_t ax;
          int16_t ay;
          int16_t az;
          int16_t gx;
          int16_t gy;
          int16_t gz;
```

```
          mpu.getMotion6(&ax,&ay,&az,&gx,&gy,&gz); // read the
acceleration and rotation in all axis.
          }
          else
          {
          // Time to do some other processing
          }
     }
}


void IMU_INT()
{
 new_data=true;
}
```

Code Examples - IMU FIFO buffer

The IMU contains an internal buffer of 1024 bytes that can hold samples. This sample enables the buffer for 3 axis acceleration and rotation.

```
#include "mbed.h"
#include "MPU6050.h"

void IMU_INT();

MPU6050 mpu(MPU6050_ADDRESS_AD0_HIGH);
InterruptIn MPU_INTERRUPT(P0_29);

int main() {
    mpu.reset();  // Reset the IMU
    wait(1);      // Wait for the reset to complete
    mpu.initialize(); // Initialize IMU
    bool mpu6050TestResult = mpu.testConnection(); // Check
connection with IMU
    if(!mpu6050TestResult)
    {
        // IMU IN UNKNOWN STATE, better take out the battery
        while(1); // HANG FOREVER;
    }
    MPU_INTERRUPT.mode(PullDown);
    MPU_INTERRUPT.rise(&IMU_INT);

    mpu.setSleepEnabled(false);
    mpu.setTempSensorEnabled(false);
    mpu.setInterruptMode(0);
    mpu.setInterruptDrive(0);
    mpu.setInterruptLatch(true);
    mpu.setInterruptLatchClear(true);
    mpu.setFIFOEnabled(true);
    mpu.setIntFIFOBufferOverflowEnabled(true);
    mpu.setAccelFIFOEnabled(true);
    mpu.setXGyroFIFOEnabled(true);
    mpu.setYGyroFIFOEnabled(true);
    mpu.setZGyroFIFOEnabled(true);


    while(1){
       if(new_data)
          {
               new_data=false;
               uint16_t fifo_count = mpu.getFIFOCount();
```

```
                while(fifo_count >= 12)
                     {
                          uint8_t sample[12];
                          mpu.getFIFOBytes(sample, 12);
                          int16_t ax = sample[0]<<8|sample[1];
                          int16_t ay = sample[2]<<8|sample[3];
                          int16_t az = sample[4]<<8|sample[5];
                          int16_t gx = sample[6]<<8|sample[7];
                          int16_t gy = sample[8]<<8|sample[9];
                          int16_t gz = sample[10]<<8|sample[11];
                     }
           }
           else
           {
           // Time to do some other processing
           }
     }
}


void IMU_INT()
{
 new_data=true;
}
```

Abstract Totem Software State machine