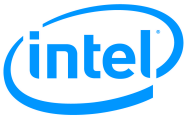
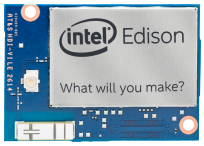
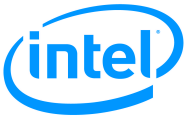




Intel® Edison Tutorial: 9DOF IMU



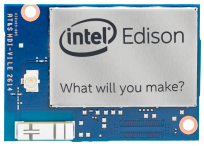
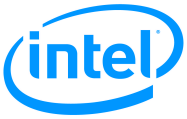
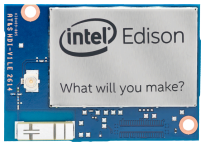


Table of Contents

Revision history		
Version	**Date**	**Comment**
1.0	10/28/2015	Initial release





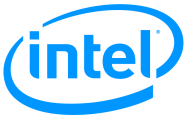
Introduction

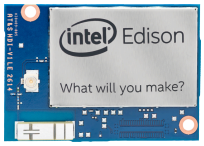
In this tutorial you will

1. Setup the hardware to use the 9DOF IMU
2. Learn to use the 9DOF IMU

List of Required Materials and Equipment

1. 1x Intel Edison Compute Module.
2. SparkFun Blocks:
    1. 1x Base Block https://www.sparkfun.com/products/13045
    2. 1x 9DOF IMU https://www.sparkfun.com/products/13033
    3. 1x Battery Block https://www.sparkfun.com/products/13037
    4. 1x Hardware Pack https://www.sparkfun.com/products/13187
3. 2x USB 2.0 A-Male to Micro B Cable (micro USB cable).
4. 1x Roll of Electrical Tape.
5. 1x Personal Computer.
6. 1x Wi-Fi network with access to the Internet.





9DOF IMU

To quote the product description provided by SparkFun: "The **9 D**egrees **o**f **F**reedom Block for the Intel® Edison uses the LSM9DS0 9DOF IMU for full-range motion sensing. This chip combines a 3-axis accelerometer, a 3-axis gyroscope, and a 3-axis magnetometer".

To clarify, please refer to the below images to understand the data collected by the 9DOF IMU.

Some of the potential 9DOF projects include:

- Head tracking for virtual reality (rotation information).
- Fall detection (acceleration information).
- Measuring quality of exercise (acceleration and rotation information).
- Location tracking (acceleration information integrated over time).

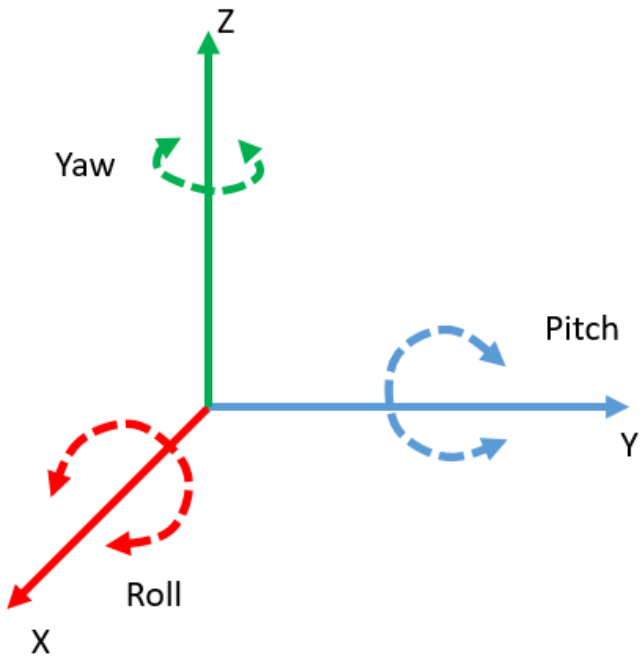


Figure 1: Diagram illustrating the type of data collected by the gyroscope

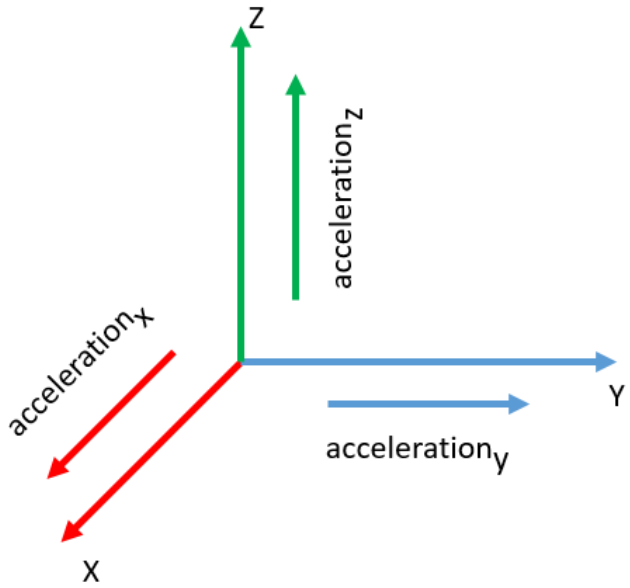


Figure 2: Diagram illustrating the type of data collected by the accelerometer

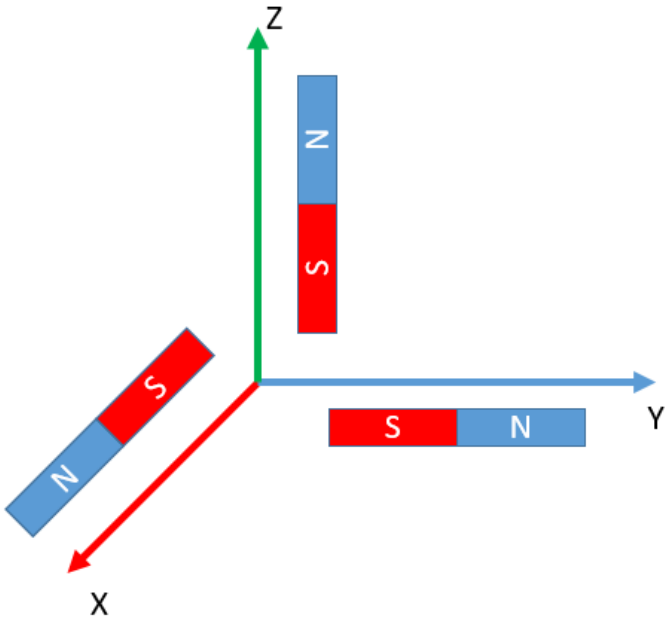


Figure 3: Diagram illustrating the type of data collected by the magnetometer

Assembling the parts

First, make sure you have all of the necessary components

1. **MAKE SURE ALL DEVICES AND COMPONENTS HAVE NO POWER BEING SUPPLIED TO THEM PRIOR TO UNPLUGGING THEM OR ATTEMPTING TO ASSEMBLE THEM.**
2. Base Block.

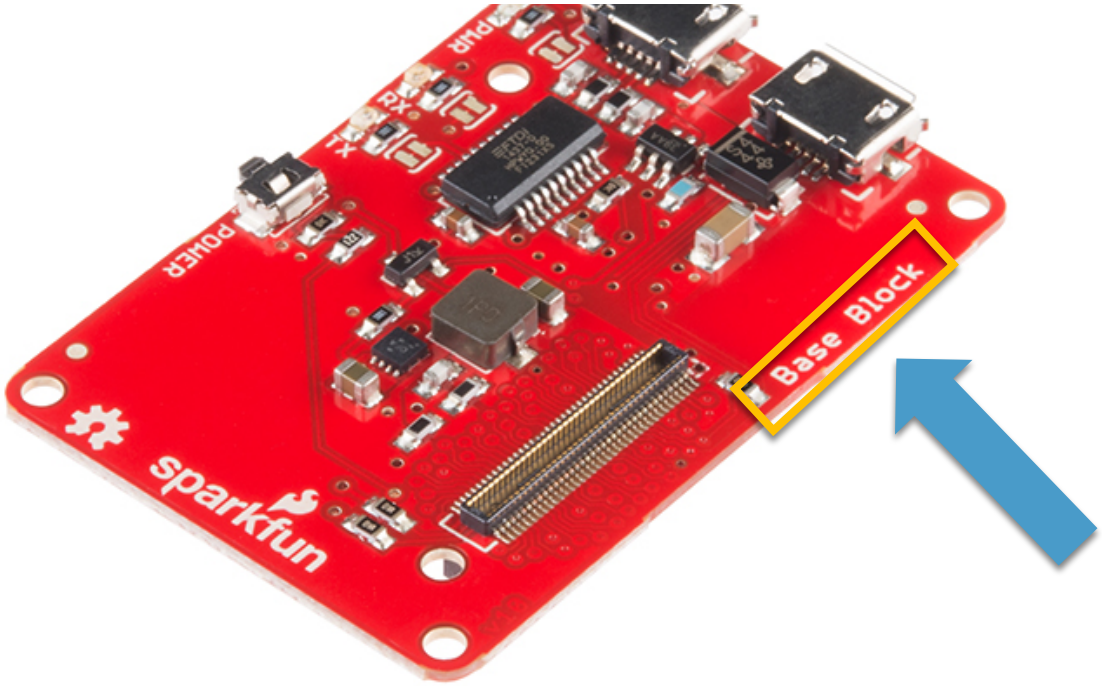


Figure 4: Base block, required to access the shell on the Intel Edison Compute Module via a wireline serial interface

3. 9DOF Block.

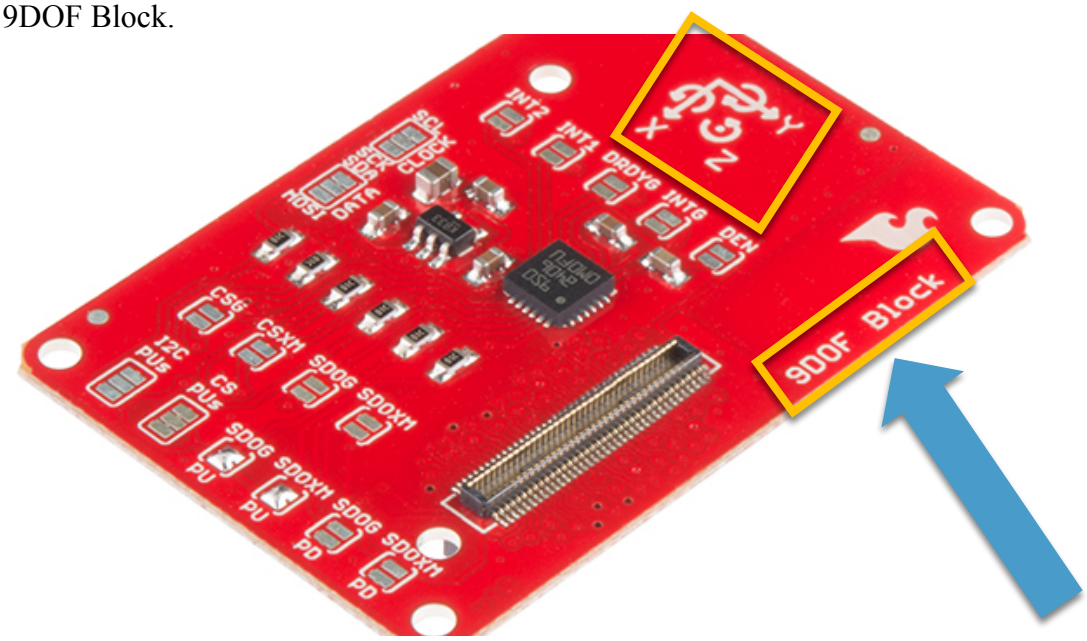


Figure 5: 9DOF IMU, provides rotation, acceleration, magnetic and temperature information via I2C to the Intel Edison Compute Module

4. Battery Block.

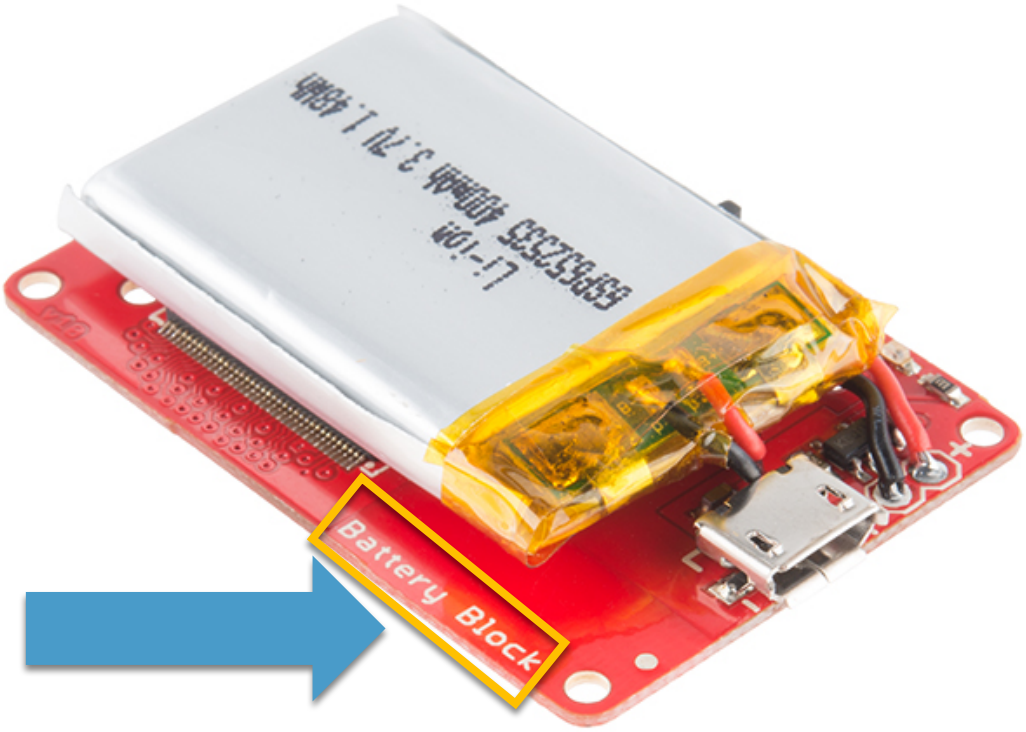


Figure 6: Battery Block, provides energy to Intel Edison Compute Module

5. Hardware Pack.

Figure 7: Hardware Pack, use these to secure the blocks and Intel Edison Compute Module to one another

6. Notice on the battery block how there are two exposed metallic contacts. These metallic contacts are connected to the positive and negative terminals of the battery. Leaving these terminals exposed is potentially dangerous as the components, such as the Intel Edison Compute Module, sensors, or battery, could be damaged due to a short circuit. To address this issue, cut out a piece of electrical tape and place it over the contacts such that they are no longer exposed as illustrated in the figure below.

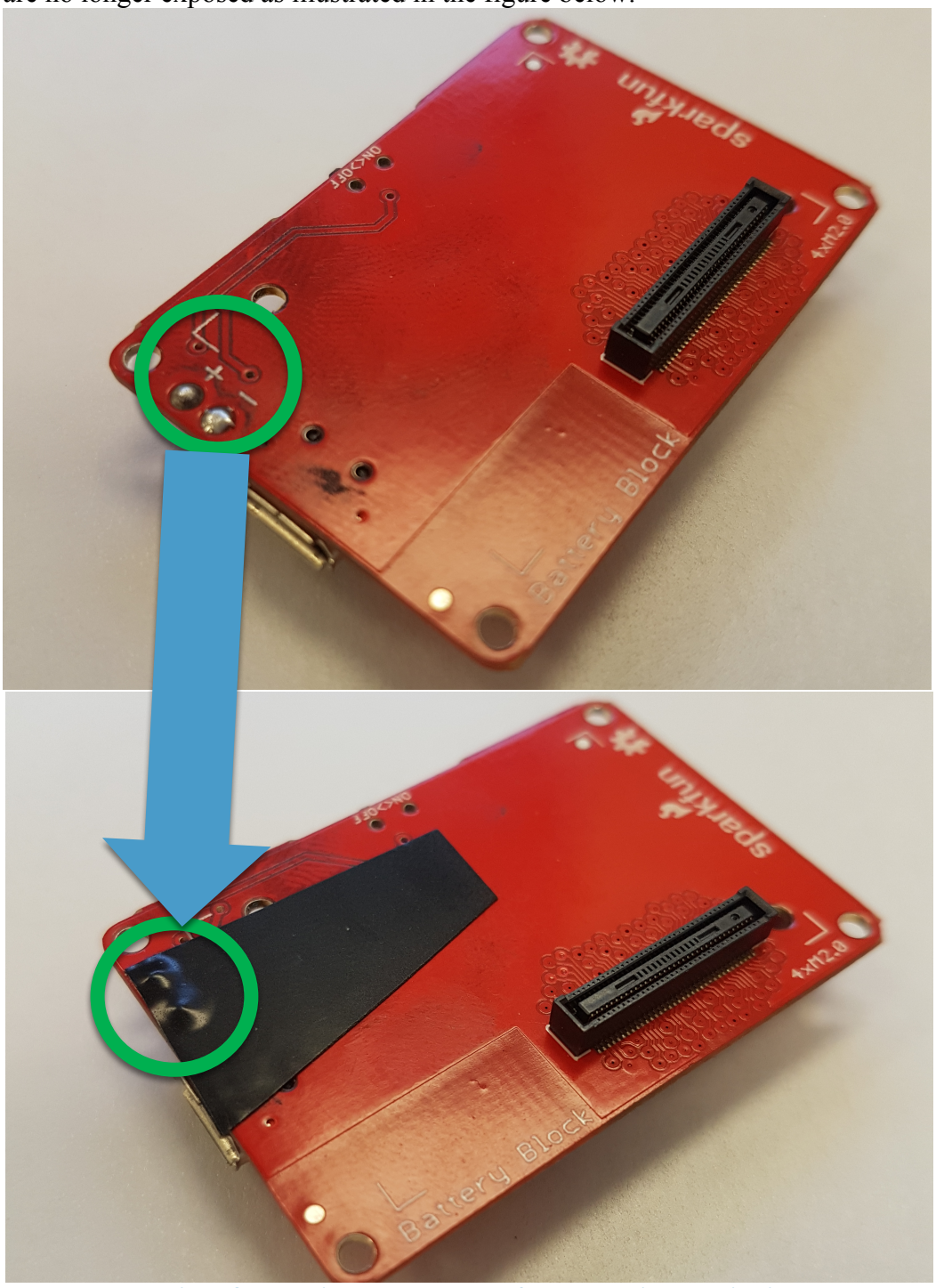


Figure 8: Protecting the battery block from a potential short circuit

7. Re-orient the board to match the below figure, and place a small silver screw in the corner.

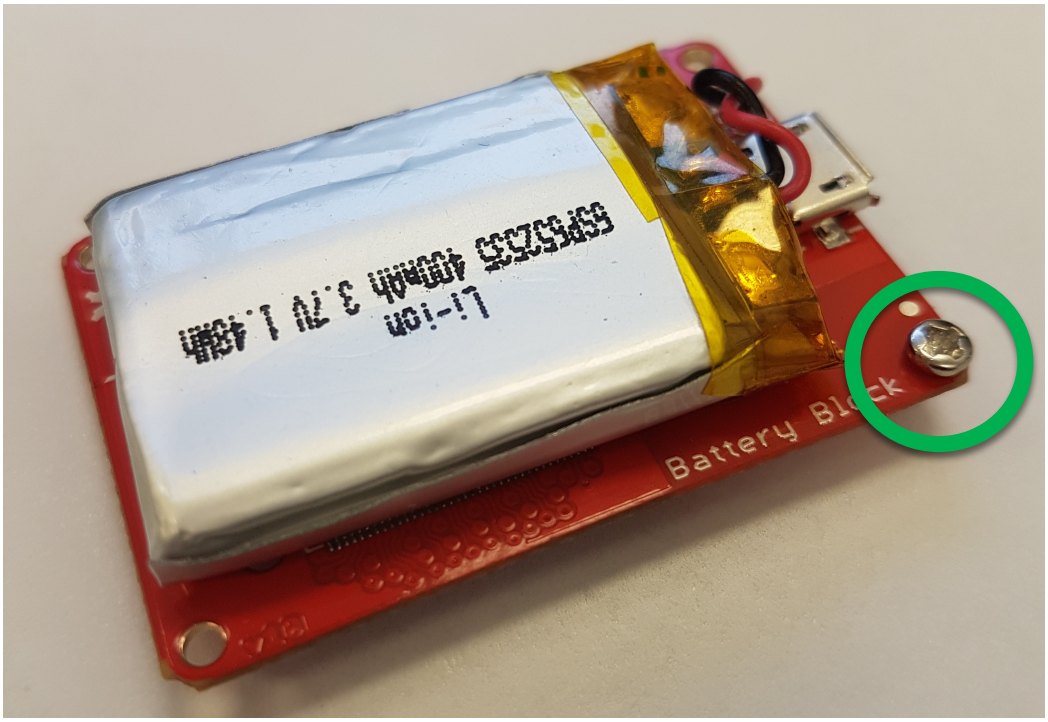


Figure 9: Correct screw placement

8. Flip the board over again and screw the silver screw into the standoff (golden screw).

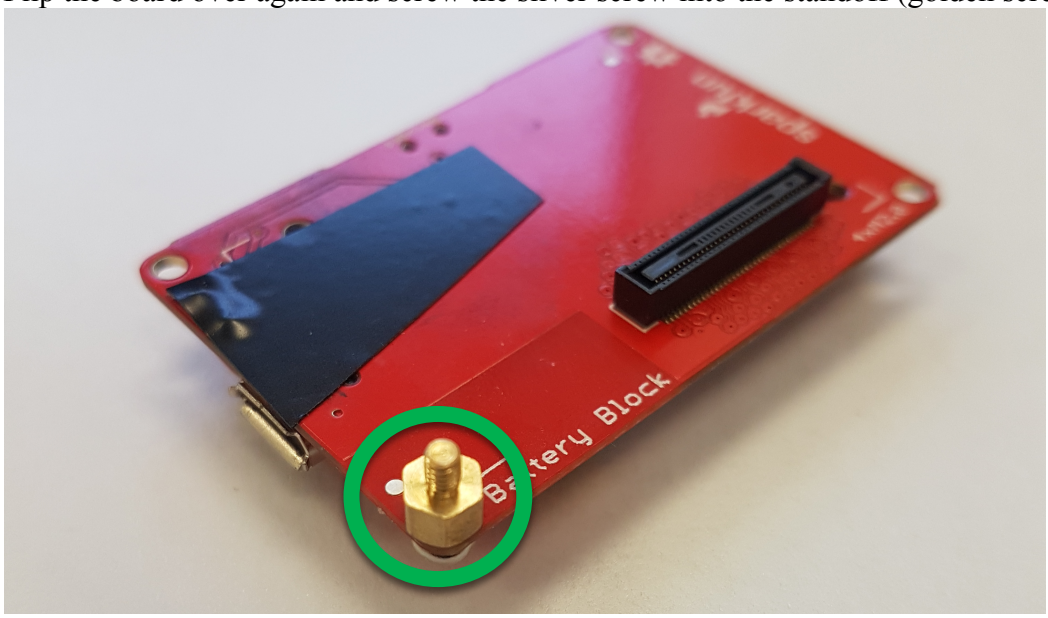


Figure 10: Correct standoff placement

9. Attach screws and standoffs to the remaining four corners.

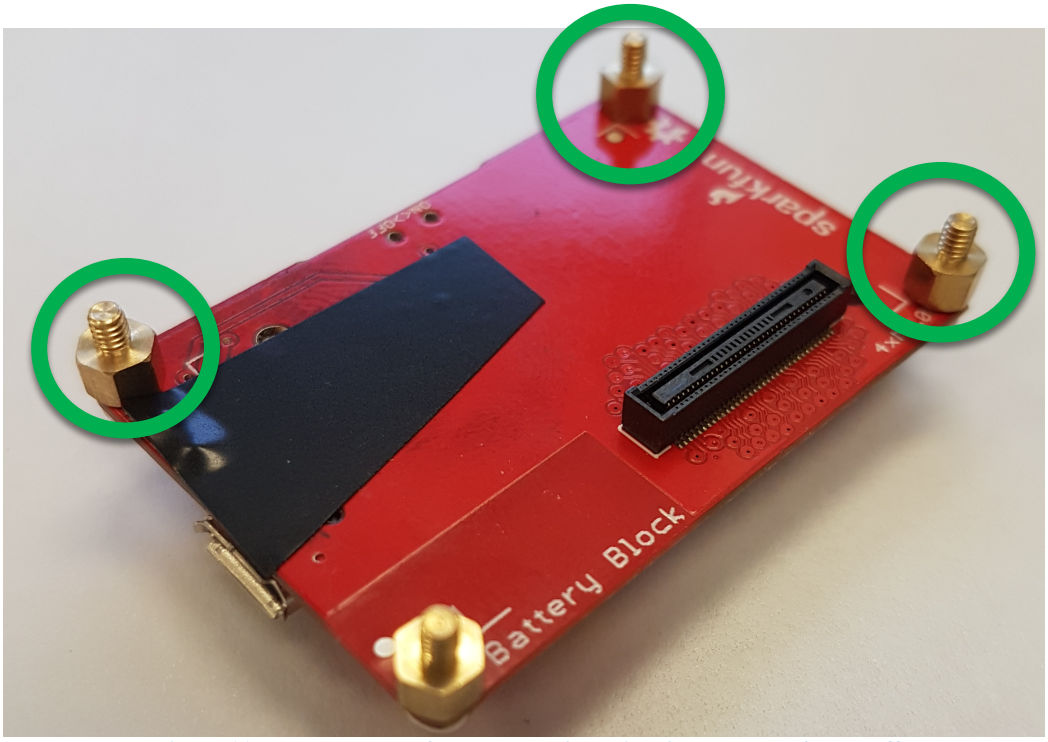


Figure 11: Battery block after correctly inserting 4 screws and 4 standoffs

10. Attach the 9DOF block as shown. Make sure the black headers align, and push firmly until you feel/hear a “click”.

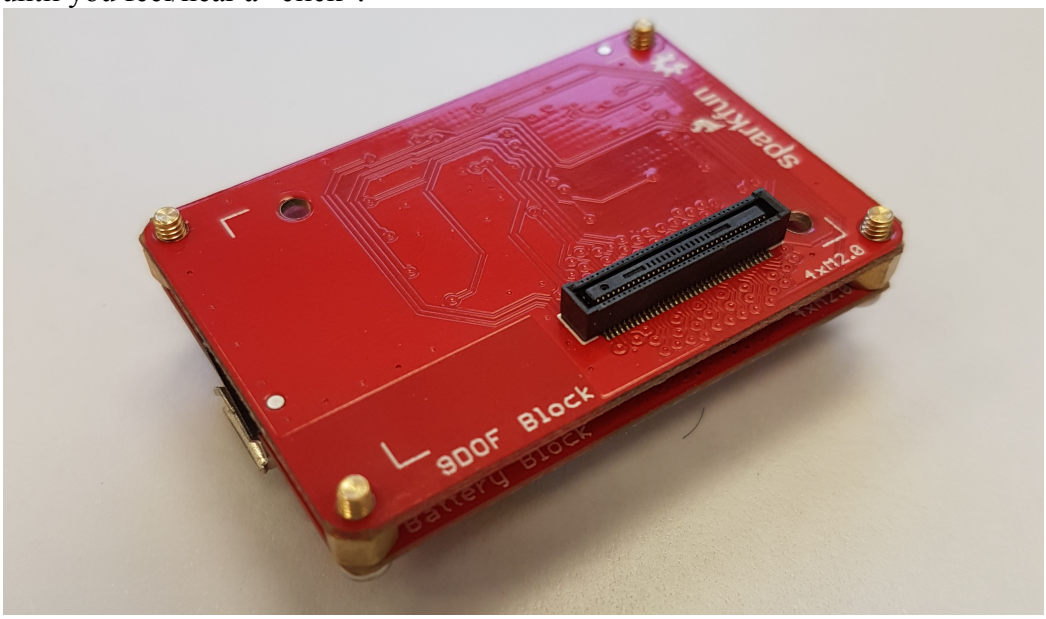


Figure 12: Attaching the 9DOF block to the Battery block in a secure manner

11. Attach three more standoffs (golden screws) as indicated.

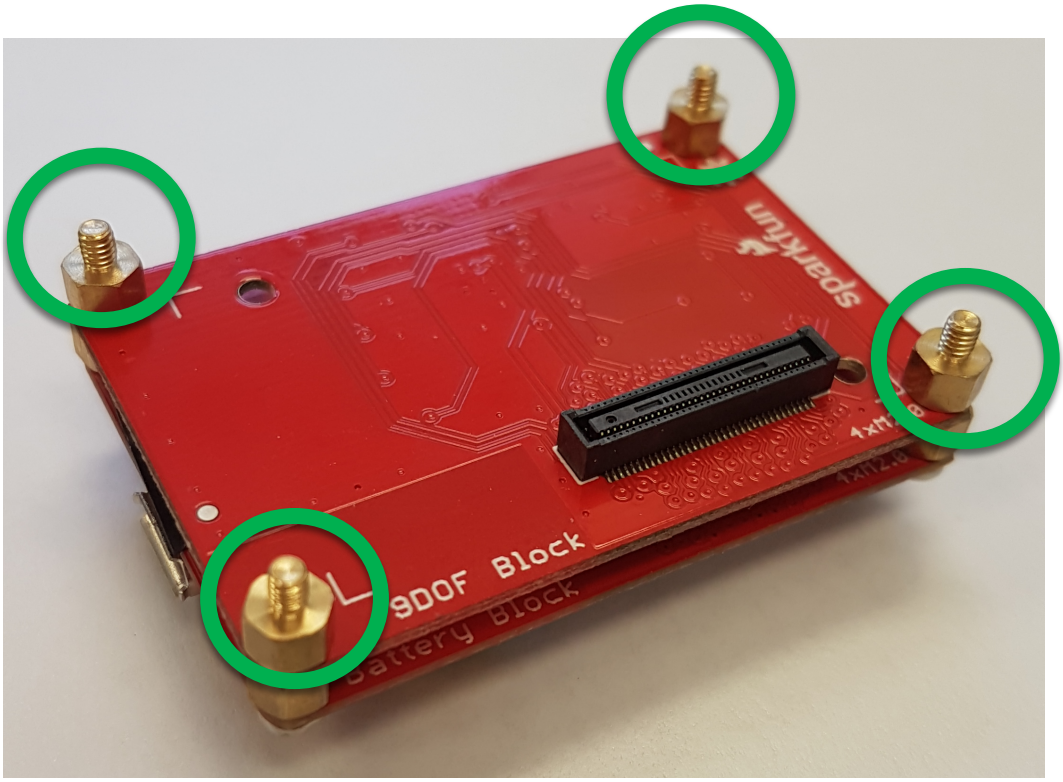


Figure 13: Attaching standoffs to the 9DOF block

12. Assemble the base block according to the following instructions:
    Place a screw in the indicated slot.

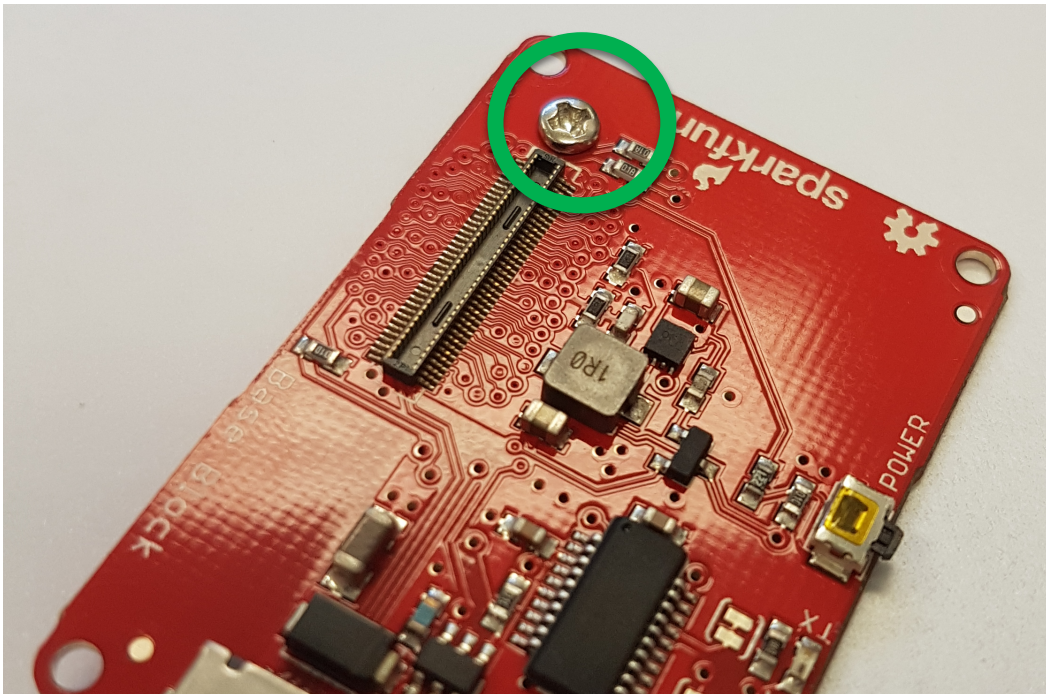


Figure 14: Attaching screws to Base block such that the Intel Edison Compute Module can be attached in a secure manner to the stack of SparkFun sensor blocks

Attach a standoff (golden screw) to this screw.

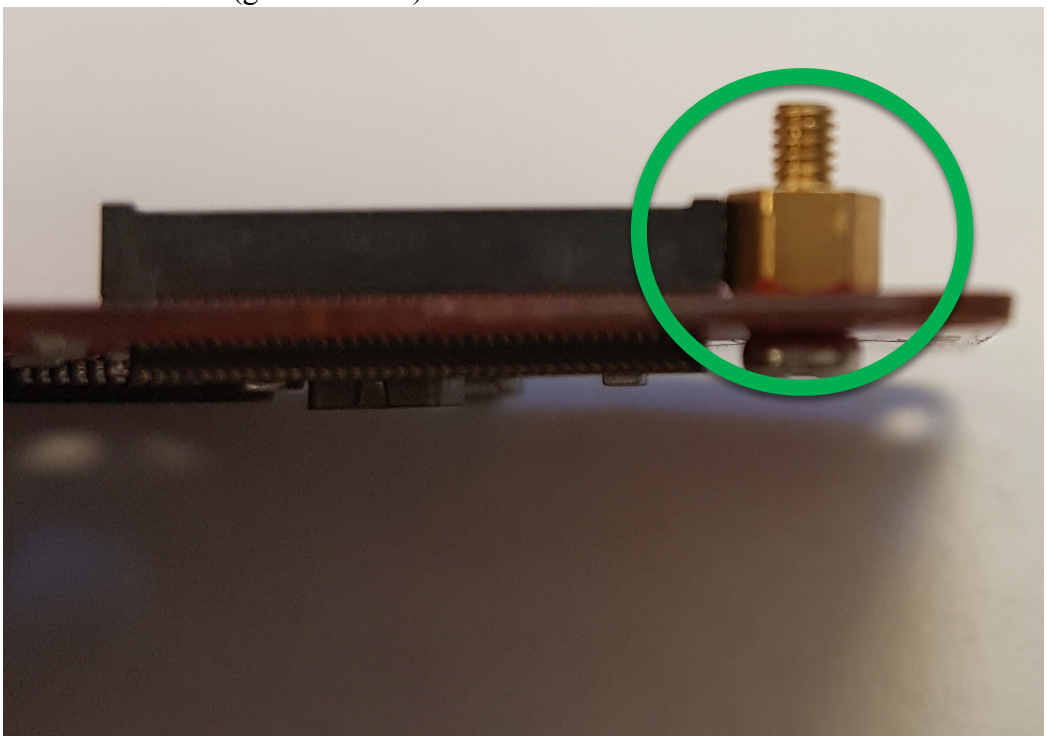


Figure 15: Correctly assembled standoff and screw for base block

Attach another screw and standoff to the other hole that is not at one of the 4 corners.

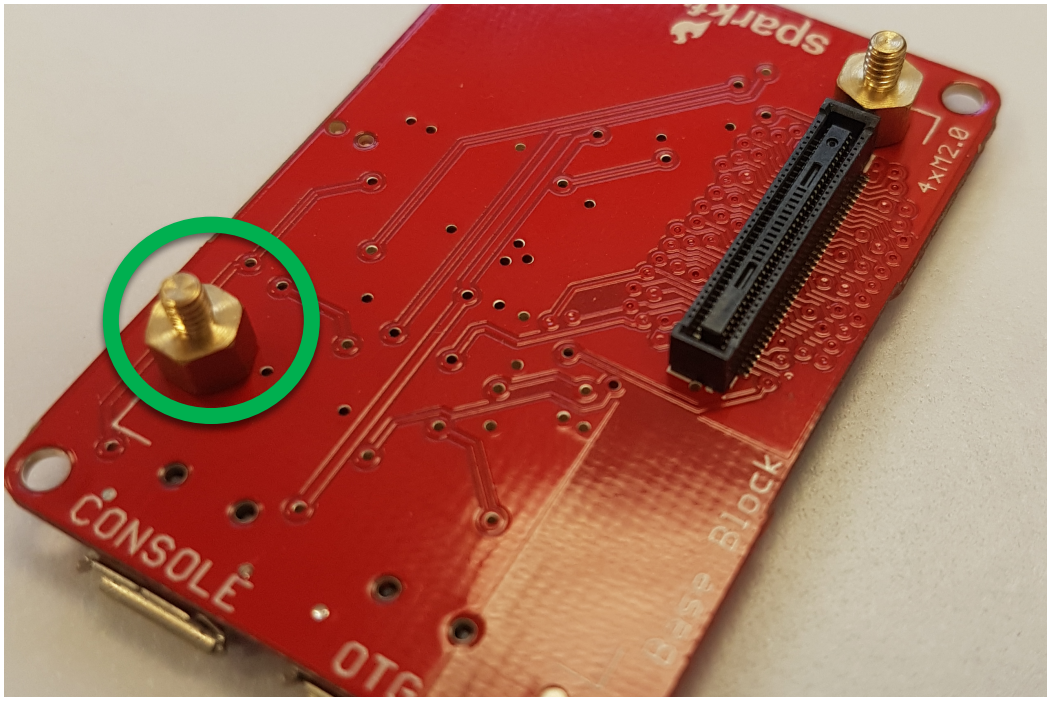


Figure 16: Both screws correctly assembled for base block

13. Attach the Intel Edison Compute Module to the base block. Press down until you hear or feel a “click”.

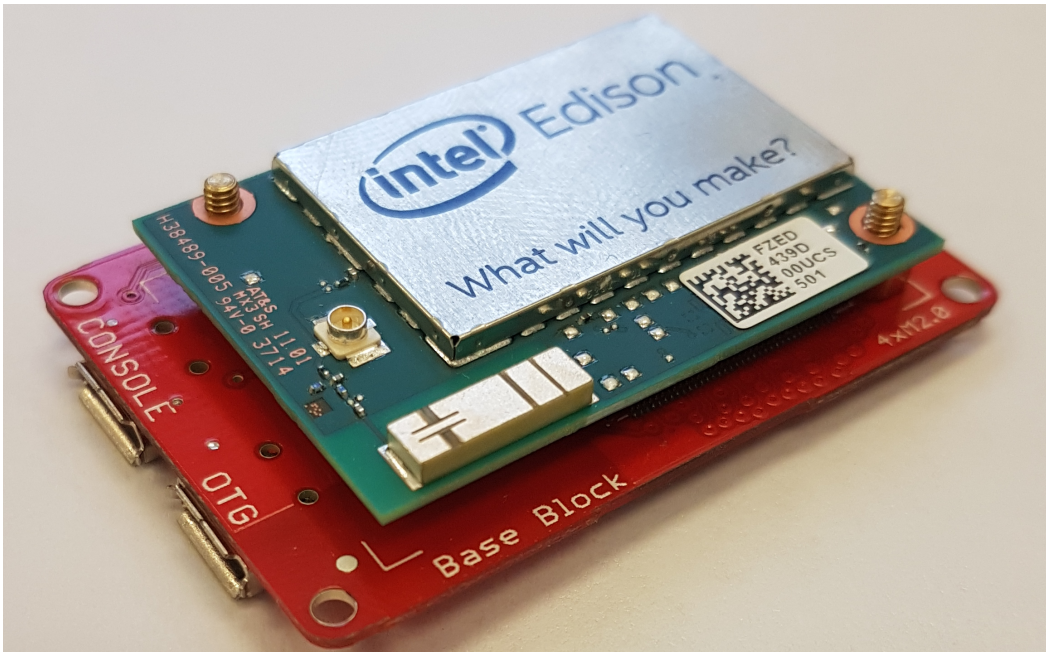


Figure 17: Intel Edison Compute Module attached to the base block

14. Attach the base block to the 9DOF (which is already attached to the battery block).

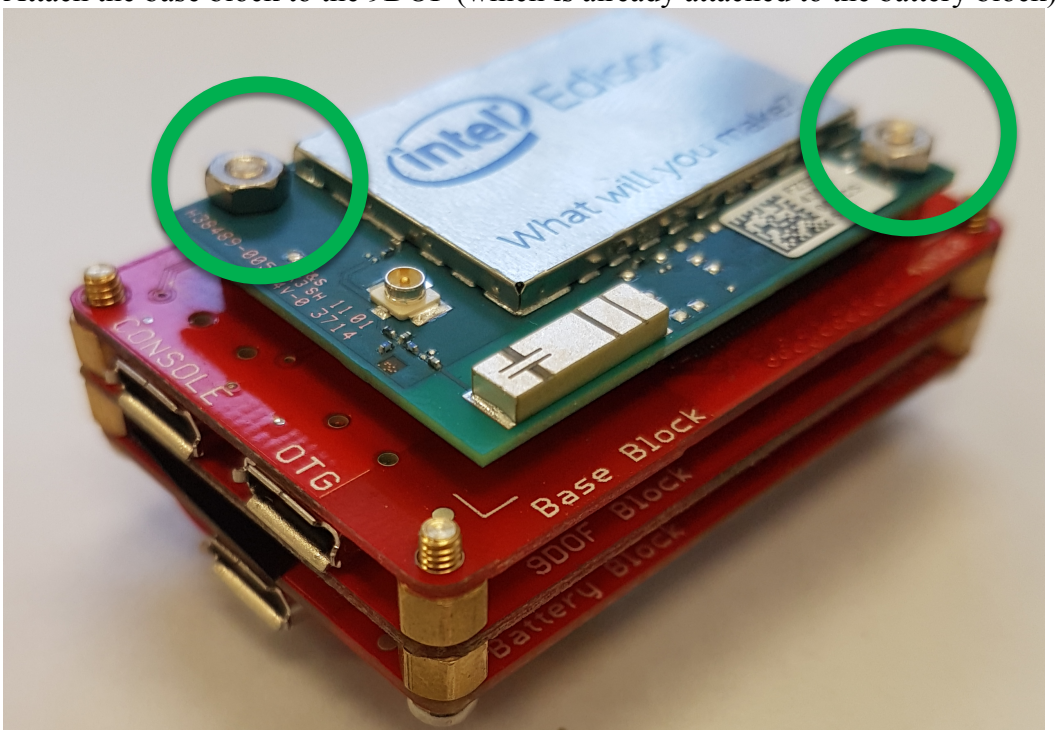


Figure 18: Attaching the base block to the rest of the SparkFun blocks. Notice how the screws secure the Intel Edison Compute Module to the rest of the SparkFun blocks

15. To charge the battery, plug the micro USB wire into the bottom part of the Edison.

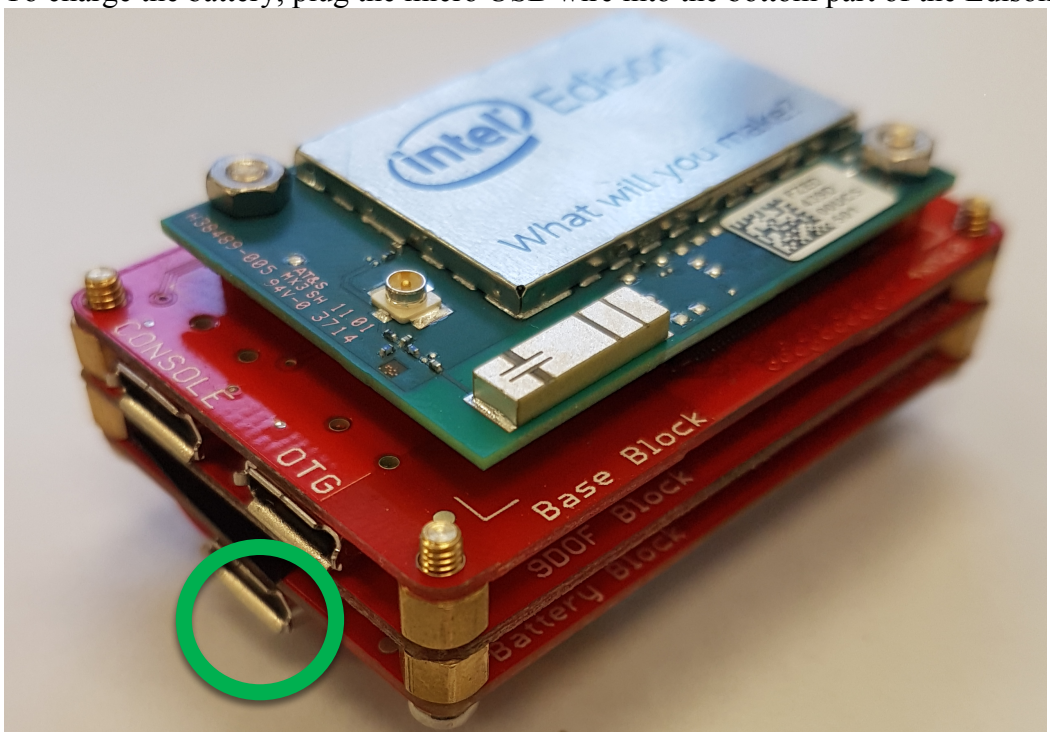


Figure 19: Correct interface to attach micro-B side of a USB cable to charge the battery on the SparkFun battery block

16. To access the Intel Edison via serial connection, plug the micro USB into the terminal marked "console", and follow the steps in the document labelled ***Intel Edison Tutorial – Introduction, Shell Access and SFTP***.

    Establishing a wireline connection between your Intel Edison and your personal computer will provide the Intel Edison with enough energy to be on, even if the battery is off.

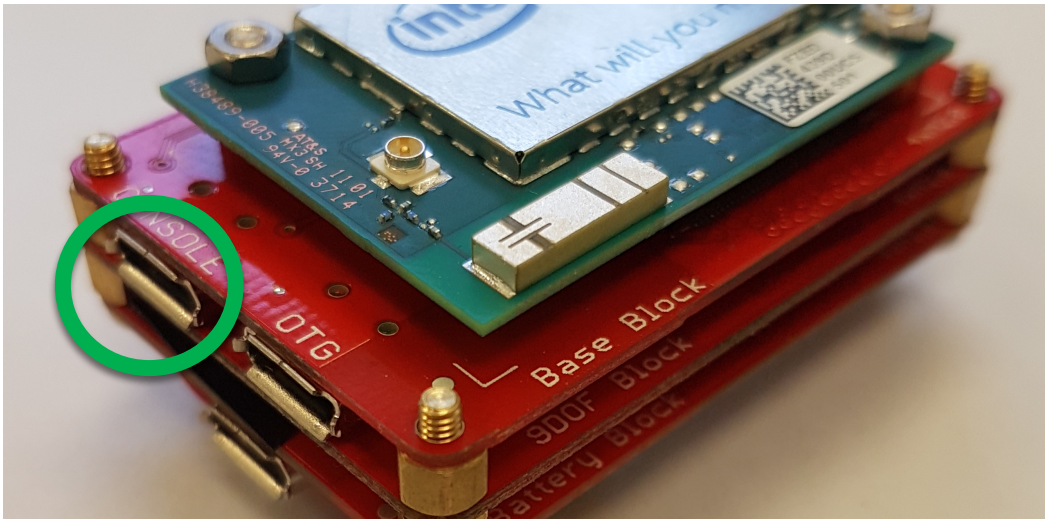


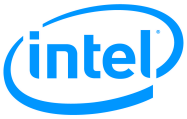
Figure 20: Interface to access the shell on the Intel Edison Compute Module via a wirelines serial interface

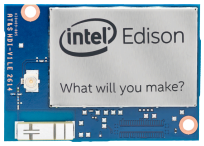
17. To access the shell of your Intel Edison Compute Module without a wireline serial interface (for example, if your project prohibits having a wireline USB connection), follow the below steps:

18. Access the shell of your Intel Edison using the SSH protocol as discussed in the document labelled ***Intel Edison Tutorial – Introduction, Shell Access and SFTP***.
19. Turn the battery on as shown in the figure below:

Figure 21: Battery block in 'off' position

Figure 22: Battery block in 'on' position



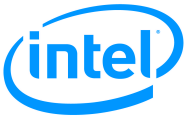


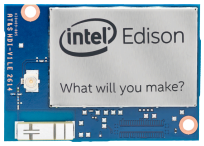
C Library Overview

To ease development, you have been provided with a set of functions to interface with the 9DOF block. These set of functions are defined in the header file **LSM9DS0.h** and implemented in the C code source file **LSM9DS0.c**.

Some of the functions present in the above files include:

1. **Initialization Functions:**
    a. **accel_init**: This function initializes the I2C connection between the Edison and the accelerometer/magnetometer. It then enables the IMU, sets the sample rate to 200Hz, and sets the scale to 2g.
    b. **gyro_init**: This function initializes the I2C connection between the Edison and the gyroscope. It then enables the IMU, sets the sample rate to 760Hz, and sets the scale to 245 dps.
    c. **mag_init**: This function initializes the I2C connection between the Edison and the accelerometer/magnetometer. It then enables the IMU, sets the sample rate to 100Hz, sets the scale to 2GS, and sets the IMU mode to continuous conversion mode.
2. **Resolution Determining Functions:**
    a. **calc_accel_res**: This function calculates g / (ADC tick).
    b. **calc_gyro_res**: This function calculates degrees per second / (ADC tick).
    c. **calc_mag_res**: This function calculates gauss / (ADC tick).
3. **Sensor Sampling Rate:**
    a. **set_accel_ODR:** set sampling rate of accelerometer to the user-specified value.
    b. **set_gyro_ODR:** set sampling rate of gyroscope to the user-specified value.
    c. **set_mag_ODR:** set sampling rate of magnetometer to the user-specified value.
4. **Data Scaling:**
    a. **set_accel_scale:** set accelerometer scale to the user-specified value.
    b. **set_gyro_scale:** set gyroscope scale to the user-specified value.
    c. **set_mag_scale:** set magnetoometer scale to the user-specified value.
5. **Read Data:**
    a. **read_accel:** acquire data gathered from accelerometer.
    b. **read_gyro:** acquire data gathered from gyroscope.
    c. **read_mag:** acquire data gathered from magnetometer.
    d. **read_temp:** acquire data gathered from temperature sensor.
6. **Offset Calculation:**
    a. **calc_gyro_offset:** This function calculates and returns the average output value of each gyroscope axis.





Programming Guide

This section guides you through IMU initialization, scale setup, sample rate setup, resolution calculation, and IMU data.

IMU Initialization

Examine the below code:

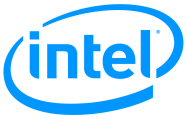
```
#include <stdio.h>
#include <mraa/i2c.h>
#include "LSM9DS0.h"

int main() {
     mraa_i2c_context accel, gyro, mag;

     accel = accel_init();
     gyro = gyro_init();
     mag = mag_init();

     return 0;
}
```

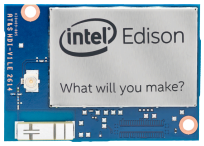
Figure 23 IMU initialization

The initialization functions (i.e. accel_init, gyro_init, and mag_init) return mraa I2C contexts, which are MRAA library's abstraction to handle the I2C connection. Thus, we need to include mraa library as shown in line 2.

In order to use the LSM9DS0 C library functions, we also need to include LSM9DS0.h header file as shown in line 3. This header file and LSM9DS0.c file must be included in the same directory along with your C code file.

In the main function, we can declare the MRAA I2C contexts as shown in line 6. Then, we can assign the returned values of the initialization functions to the I2C contexts as shown in lines 8, 9, and 10.





Scaling

Please examine the below code to set up the scale of each sensor.

```
#include <stdio.h>
#include <mraa/i2c.h>
#include "LSM9DS0.h"

int main() {
     mraa_i2c_context accel, gyro, mag;

     accel = accel_init();

     set_accel_scale(accel, A_SCALE_2G);

     gyro = gyro_init();
     set_gyro_scale(gyro, G_SCALE_245DPS);

     mag = mag_init();
     set_mag_scale(mag, M_SCALE_2GS);

     return 0;
}
```

Figure 24 Scale setup

The highlighted lines are scale setup examples. The functions need two arguments: An I2C context and a scale value. The scale value is a multiplier that adjusts the scale of the raw data values collected from the I2C interface to the user-specified value. These user-specified values are tabulated below:

Accelerometer

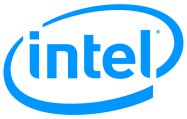
1. A_SCALE_2G *(2 g)*
2. A_SCALE_4G *(4 g)*
3. A_SCALE_6G *(6 g)*
4. A_SCALE_8G *(8 g)*
5. A_SCALE_16G *(16 g)*

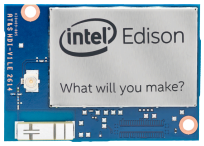
Gyroscope

1. G_SCALE_245DPS *(245 dps)*
2. G_SCALE_500DPS *(500 dps)*
3. G_SCALE_2000DPS *(2000 dps)*

Magnetometer

1. M_SCALE_2GS *(2 Gs)*
2. M_SCALE_4GS *(4 Gs)*
3. M_SCALE_8GS *(8 Gs)*
4. M_SCALE_12GS *(12 Gs)*





Sample Rate Setup

Please take a look at the following code.

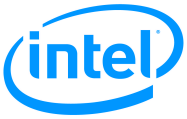
```
#include <stdio.h>
#include <mraa/i2c.h>
#include "LSM9DS0.h"

int main() {
     mraa_i2c_context accel, gyro, mag;

     accel = accel_init();
     set_accel_scale(accel, A_SCALE_2G);
     set_accel_ODR(accel, A_ODR_100);

     gyro = gyro_init();
     set_gyro_scale(gyro, G_SCALE_245DPS);
     set_gyro_ODR(accel, G_ODR_190_BW_70);

     mag = mag_init();
     set_mag_scale(mag, M_SCALE_2GS);
     set_mag_ODR(mag, M_ODR_125);

     return 0;
}
```

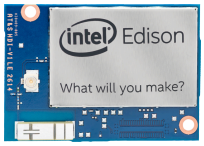
Figure 25 Sample Rate Setup

Similar to the scale initialization functions, a sample rate initialization function also takes two arguments: An I2C context and a sample rate values. Unlike other two functions, the gyroscope sample rate setup function will also initialize the bandwidth as well. The following is the list of the possible sample rate values:

Accelerometer

1. A_POWER_DOWN *(Power-down mode)*
2. A_ODR_3125 *(3.125 Hz)*
3. A_ODR_625 *(6.25 Hz)*
4. A_ODR_125 *(12.5 Hz)*
5. A_ODR_25 *(25 Hz)*
6. A_ODR_50 *(50 Hz)*
7. A_ODR_100 *(100 Hz)*
8. A_ODR_200 *(200 Hz)*
9. A_ODR_400 *(400 Hz)*
10. A_ODR_800 *(800 Hz)*
11. A_ODR_1600 *(1600 Hz)*



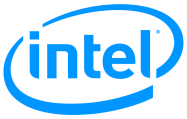


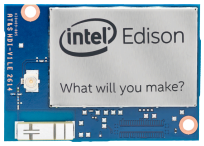
Gyroscope

1. G_ODR_95_BW_125 *(rate: 95 Hz, BW:12.5)*
2. G_ODR_95_BW_25 *(rate: 95 Hz, BW: 25)*
3. G_ODR_190_BW_125 *(rate: 190 Hz, BW: 12.5)*
4. G_ODR_190_BW_25 *(rate: 190 Hz, BW: 25)*
5. G_ODR_190_BW_50 *(rate: 190 Hz, BW: 50)*
6. G_ODR_190_BW_70 *(rate: 190 Hz, BW: 70)*
7. G_ODR_380_BW_20 *(rate: 380 Hz, BW: 20)*
8. G_ODR_380_BW_25 *(rate: 380 Hz, BW: 25)*
9. G_ODR_380_BW_50 *(rate: 380 Hz, BW: 50)*
10. G_ODR_380_BW_100 *(rate: 380 Hz, BW: 100)*
11. G_ODR_760_BW_30 *(rate: 760 Hz, BW: 30)*
12. G_ODR_760_BW_35 *(rate: 760 Hz, BW: 35)*
13. G_ODR_760_BW_50 *(rate: 760 Hz, BW: 50)*
14. G_ODR_760_BW_100 *(rate: 760 Hz, BW: 100)*

Magnetometer

1. M_ODR_3125 (3.125 Hz)
2. M_ODR_625 (6.25 Hz)
3. M_ODR_125 (12.5 Hz)
4. M_ODR_25 (25 Hz)
5. M_ODR_50 (50 Hz)
6. M_ODR_100 (100 Hz)





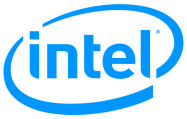
Resolution Calculation

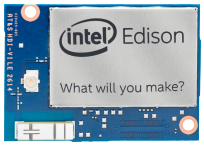
The library includes convenient functions to calculate the IMU resolution. Please take a look at the following code.

```
#include <stdio.h>
#include <mraa/i2c.h>
#include "LSM9DS0.h"

int main() {
     mraa_i2c_context accel, gyro, mag;
     float a_res, g_res, m_res;

     accel = accel_init();
     set_accel_scale(accel, A_SCALE_2G);
     set_accel_ODR(accel, A_ODR_100);
     a_res = calc_accel_res(A_SCALE_2G);

     gyro = gyro_init();
     set_gyro_scale(gyro, G_SCALE_245DPS);
     set_gyro_ODR(accel, G_ODR_190_BW_70);
     g_res = calc_gyro_res(G_SCALE_245DPS);

     mag = mag_init();
     set_mag_scale(mag, M_SCALE_2GS);
     set_mag_ODR(mag, M_ODR_125);
     m_res = calc_mag_res(M_SCALE_2GS);

     return 0;
}
```

Figure 26 Resolution calculation

First, we need to declare float variables to store the resolution of each IMU onboard the 9DOF block. The resolution calculation functions take the scale value as an argument and return the resolution. It is important to note that the scale value passed to the function **MUST BE THE SAME** value used to set the scale. For instance, **set_accel_scale(accel, A_SCALE_2G)** and **calc_accel_res(A_SCALE_2G)** have the same scale value.

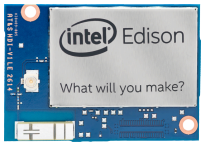




Data Collection:

```
#include <stdio.h>
#include <mraa/i2c.h>
#include "LSM9DS0.h"

int main() {
     mraa_i2c_context accel, gyro, mag;
     float a_res, g_res, m_res;
     data_t accel_data, gyro_data, mag_data;
     int16_t temperature;

     accel = accel_init();
     set_accel_scale(accel, A_SCALE_2G);
     set_accel_ODR(accel, A_ODR_100);
     a_res = calc_accel_res(A_SCALE_2G);

     gyro = gyro_init();
     set_gyro_scale(gyro, G_SCALE_245DPS);
     set_gyro_ODR(accel, G_ODR_190_BW_70);
     g_res = calc_gyro_res(G_SCALE_245DPS);

     mag = mag_init();
     set_mag_scale(mag, M_SCALE_2GS);
     set_mag_ODR(mag, M_ODR_125);
     m_res = calc_mag_res(M_SCALE_2GS);

     while(1) {
          accel_data = read_accel(accel, a_res);
          gyro_data = read_gyro(gyro, g_res);
          mag_data = read_mag(mag, m_res);
          temperature = read_temp(accel);
          printf("X: %f\t Y: %f\t Z: %f\t||", accel_data.x,
accel_data.y, accel_data.z);
          printf("\tX: %f\t Y: %f\t Z: %f\t||", gyro_data.x,
gyro_data.y, gyro_data.z);
          printf("\tX: %f\t Y: %f\t Z: %f\t||", mag_data.x,
mag_data.y, mag_data.z);
          printf("\t%ld\n", temperature);
          usleep(100000);
     }
     return 0;
}
```

Figure 27 IMU data



There are 4 functions that return the read data from the 9DOF block: **read_accel**, **read_gyro**, and **read_mag.** These functions take an I2C context and resolution as arguments and return the data collected as a **struct data_t**.

data_t is defined as follows:

```
typedef struct {
    float x, y, z
} data_t;
```

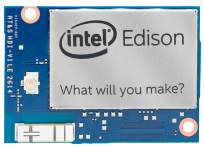
Therefore, if you declare a variable as

```
data_t accel_data;
```

You can access the x, y, and z components of the acceleration vector by using the following syntax:

acceleration_x: accel_data.x
acceleration_y: accel_data.y
acceleration_z: accel_data.z

The function **read_temp**, returns the temperature data as a 16-bit signed integer format.



Programming Example

To examine the data being collected from the 9DOF block, please follow the below steps:

1. Download the code available in the folder labelled **FILES** from the same location as where you downloaded this PDF. The three files you should have are:
   LSM9DS0.c
   LSM9DS0.h
   example.c

2. Access the shell on your Intel Edison using SSH as outlined in the document labelled ***Intel Edison Tutorial – Introduction, Shell Access and SFTP***. Ensure that you are using an SSH connection as you may need to resize your window to see all the data. Connecting via SSH will also ensure you have a stable internet connection for file transfer.

3. Upload all three files to your Intel Edison via SFTP as outlined in the document labelled ***Intel Edison Tutorial – Introduction, Shell Access and SFTP***

4. Compile the code by issuing the below command:

   $ gcc –lmraa -o <desired_program_name> <your_c_code_file_name> LSM9DS0.c

```
root@EE96_TA:~/9DOF# ls
LSM9DS0.c  LSM9DS0.h  example.c
root@EE96_TA:~/9DOF# gcc -lmraa -o example example.c LSM9DS0.c
root@EE96_TA:~/9DOF#
```

Figure 28: Compilation of example.c. All three files (LSM9DS0.c, LSM9DS0.h, and example.c) are required for successful compilation.

5. **<u>MAKE SURE THE 9DOF BLOCK IS STATIONARY PRIOR TO PROCEEDING TO THE NEXT STEP. ESNURE IT STAYS STABLE UNTIL THE OFFSET CALCULATION HAS BEEN COMPLETED.</u>**

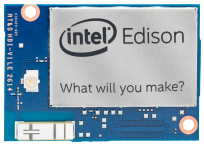
6. Execute the code by issuing the following command:

   $ ./example

```
root@EE96_TA:~/9DOF# ./example
Calculating the offsets...
x: 3.260998 y: 0.430694 z: -0.521304
                Accelerometer                ||                Gyroscope                ||               Magnetometer                ||  Temperature
X: -0.085327    Y: 0.030151    Z: 0.977295    ||    X: -0.502057    Y: -0.258727    Z: 0.035311    ||    X: 0.315308    Y: 0.050171    Z: 0.212463    ||  39
X: -0.085693    Y: 0.028931    Z: 0.976929    ||    X: -0.584381    Y: -0.086761    Z: -0.091794   ||    X: 0.305481    Y: 0.035217    Z: 0.202637    ||  38
X: -0.081055    Y: 0.029053    Z: 0.974976    ||    X: -0.404858    Y: -0.124145    Z: 0.132510    ||    X: 0.339661    Y: 0.045959    Z: 0.222229    ||  38
X: -0.085205    Y: 0.030884    Z: 0.976929    ||    X: -0.345044    Y: -0.139098    Z: 0.237185    ||    X: 0.306763    Y: 0.055298    Z: 0.221191    ||  37
X: -0.085449    Y: 0.030518    Z: 0.974365    ||    X: -0.546917    Y: -0.438170    Z: -0.278715   ||    X: 0.306030    Y: 0.035095    Z: 0.207153    ||  37
X: -0.084351    Y: 0.030151    Z: 0.976196    ||    X: -0.457196    Y: 0.032868     Z: 0.072695    ||    X: 0.305725    Y: 0.051208    Z: 0.184570    ||  37
X: -0.081665    Y: 0.031372    Z: 0.974487    ||    X: -0.928235    Y: -0.273681    Z: 0.169894    ||    X: 0.313171    Y: 0.042542    Z: 0.211426    ||  38
X: -0.087158    Y: 0.031616    Z: 0.977295    ||    X: -0.387660    Y: -0.221343    Z: 0.199801    ||    X: 0.320557    Y: 0.073181    Z: 0.217773    ||  41
X: -0.083496    Y: 0.027588    Z: 0.976685    ||    X: -0.427289    Y: -0.640044    Z: 0.065219    ||    X: 0.319458    Y: 0.029968    Z: 0.229675    ||  41
X: -0.084961    Y: 0.033447    Z: 0.979736    ||    X: -0.427289    Y: -0.183959    Z: 0.132510    ||    X: 0.307190    Y: 0.026489    Z: 0.210205    ||  39
X: -0.084351    Y: 0.031738    Z: 0.974487    ||    X: -0.135693    Y: -0.288634    Z: 0.162417    ||    X: 0.304816    Y: 0.040283    Z: 0.197815    ||  38
X: -0.086304    Y: 0.027222    Z: 0.974121    ||    X: -0.337567    Y: -0.056853    Z: -0.159086   ||    X: 0.317383    Y: 0.051025    Z: 0.260010    ||  40
X: -0.086304    Y: 0.031128    Z: 0.962402    ||    X: -0.322613    Y: -0.191436    Z: 0.192324    ||    X: 0.316711    Y: 0.044250    Z: 0.187866    ||  37
X: -0.089111    Y: 0.027466    Z: 0.967651    ||    X: -0.703930    Y: -0.221343    Z: 0.035311    ||    X: 0.298767    Y: 0.051514    Z: 0.215149    ||  37
X: -0.084961    Y: 0.030029    Z: 0.974243    ||    X: -0.502057    Y: -0.340972    Z: 0.087649    ||    X: 0.298950    Y: 0.039490    Z: 0.204041    ||  39
X: -0.082764    Y: 0.031128    Z: 0.980591    ||    X: -0.322613    Y: 0.159974     Z: 0.057742    ||    X: 0.328491    Y: 0.063416    Z: 0.254150    ||  37
X: -0.083618    Y: 0.030029    Z: 0.973145    ||    X: -1.055340    Y: -0.415748    Z: 0.027835    ||    X: 0.320312    Y: 0.031616    Z: 0.209778    ||  39
X: -0.085327    Y: 0.031982    Z: 0.976440    ||    X: -0.449719    Y: 0.137544     Z: -0.031980   ||    X: 0.310120    Y: 0.036011    Z: 0.230837    ||  37
X: -0.083008    Y: 0.031738    Z: 0.975586    ||    X: -0.098309    Y: -0.094237    Z: 0.244662    ||    X: 0.332581    Y: 0.042847    Z: 0.199158    ||  37
X: -0.082520    Y: 0.029785    Z: 0.975708    ||    X: 0.020797     Y: -0.169005    Z: 0.162417    ||    X: 0.304382    Y: 0.041382    Z: 0.206238    ||  40
```

Figure 29: Running the program

7. Re-orientate the 9DOF block while the code is running and examine the output.



Tasks

1. Edit the code available in **example.c** such that it outputs "Up" followed by the acceleration in the z-axis if it is positive, or "Down" followed by the acceleration in the z-axis if it is negative. Please see the below diagram for an example.

```
Up, 0.025574
Up, 0.013367
Up, 0.020752
Up, 0.023315
Up, 0.066345
Down, -0.217651
Down, -0.703491
Down, -1.018677
Down, -0.990173
Down, -0.998596
```

Figure 30: Sample output from example.c after successful modification, compilation, and execution

2. **EXTENSION:** While the 9DOF block is stationary, calculate the angle of the 9DOF block by analyzing the x, y, and z acceleration information.

   HINT: you will need to use trigonometry to calculate this angle.

References

1. https://learn.sparkfun.com/tutorials/sparkfun-blocks-for-intel-edison---9-degrees-of-freedom-block-?_ga=1.219694809.1549320362.1444206960
2. http://www.adafruit.com/datasheets/LSM9DS0.pdf