# IMU GEN 3 DOCUMENTATION

## Daniel Greenheck

## 11/12/15

# WARNING

When handling the board, please handle it by the edges. Touch any of the sensors or the microcontroller may result in an electrostatic discharge (ESD), destroying the component. For extra precaution- although not necessary for non-flight boards- it is recommend that you wear an anti-static wrist strap when handling the board.

# HARDWARE CONNECTIONS

## Programmer Connections

The programmer is used to load new software onto the microcontroller. The IMU is programmed from Texas Instrument’s CCSv6. When installing CCSv6, make sure you choose to install the drivers for the XDS100v2

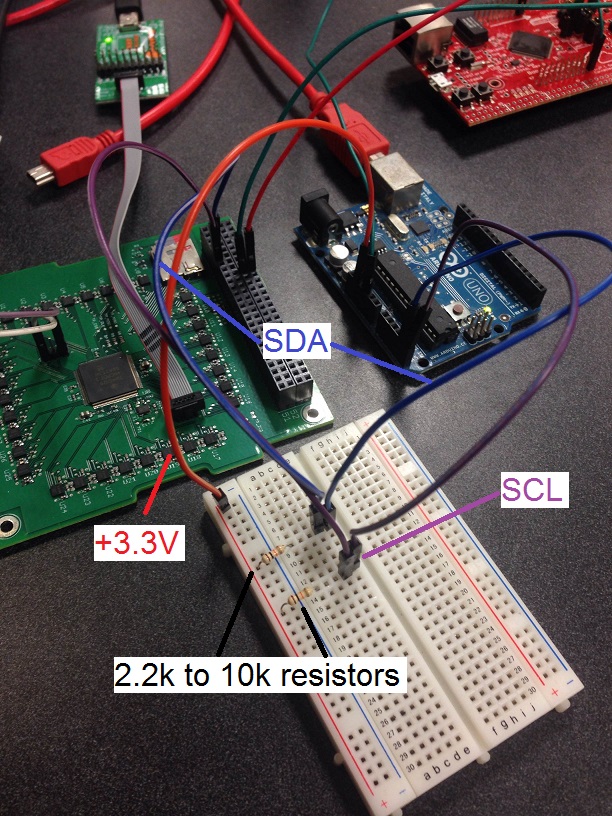
## C:\Users\Daniel Greenheck\Google Drive\Research\GenIII\Documentation\User Instructions\ProgrammingCable.JPG

1. Install the drivers for the XDS100v2 by following the instructions that came with it
2. Once installed, connect the XDS100v2 ARM programming board to the IMU as shown
3. Note the orientation of the red wire in the ribbon cable
4. Verify that the XDS100v2 is receiving power by checking that the green LED is on

## Arduino Uno/I2C Connections

The Arduino is used to collect data from the IMU when the IMU is in I2C output mode.

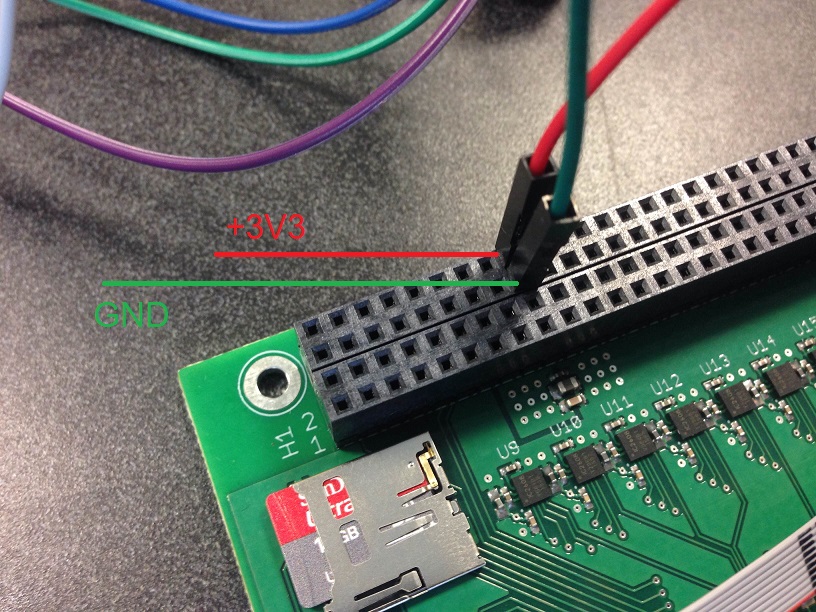
1. Connect one of the GND pins on the Arduino to the power supply ground.
2. The I2C bus requires a 4.7k pullup resistor on both the SDA and SCL lines. The easiest way to include these is to use a breadboard as an in-between.
3. Connect pin A4 (SDA) on the Arduino to one row of the breadboard. Run another wire from that row to H1-P9 (SCL) on the IMU
4. Connect pin A5 (SDA) on the Arduino to one row of the breadboard. Run another wire from that row to H1-P11 (SDA) on the IMU
5. Connect a 4.7k (or 2.2k up to 10k acceptable) resistor from SCL to the breadboard power rail.
6. Connect a 4.7k (or 2.2k up to 10k acceptable) resistor from SDA to the breadboard power rail.
7. Connect the Arduino’s 3.3V power to the breadboard power rail.



1. Connect the FTDI UART cable to the Arduino. First, connect the UART ground to the Arduino ground. Then connect pin 11 on the Arduino to UART RX. The UART is used for data logging only so it is not necessary to connect UART TX to the Arduino.

## Power Connections

1. The IMU should be powered by a low-noise regulated 3.3V power supply. Begin by turning OFF the power supply.
2. Connect the power supply ground to H2-P17 (shown below)
3. Connect +3.3V to H2-P18 (shown below)



1. DOUBLE CHECK THAT YOU DID NOT SWITCH THE POWER AND GROUND. The IMU has no reverse polarity detection or protection. Switching the power and ground will destroy the board instantly.
2. Turn ON the power supply
3. Use a multimeter to verify the voltage between the test points TP\_3V3 and TP\_GND (located next to U16).

# SOFTWARE

## Required Software

1. MATLAB
2. Arduino
3. TeraTerm (for logging data)
4. CCSv6 (for programming IMU)

## Preparing the Arduino Environment

The Arduino I2C library can only handle a maximum of 32 byte reads by default. Since the data acquisition program reads 128 bytes at a time, this limit needs to be increased.

1. Navigate to C:\Program Files (x86)\Arduino\hardware\arduino\avr\ libraries\Wire
2. Open **Wire.h**
3. Delete the function definitions **requestFrom(int address, int quantity)** and **requestFrom(int address, int quantity, int sendStop).** This will prevent warning messages from popping up as a result of changing the buffer size.
4. Open **Wire.c**
5. Delete the functions **requestFrom(int address, int quantity)** and **requestFrom(int address, int quantity, int sendStop).**
6. Change #define BUFFER\_LENGTH 32 to #define BUFFER\_LENGTH 150
7. Save and close the file.
8. Open the **utility** folder and open **twi.h**
9. Change #define TWI\_BUFFER\_LENGTH 32 to #define TWI\_BUFFER\_LENGTH 150

## Preparing TeraTerm

1. Navigate to C:\Program Files (x86)\teraterm
2. Open **TERATERM.INI**
3. Search for “Debug” and verify that “Debug=On” is in the file.

# DATA ACQUISITION

## Streaming Real-time Data

In this mode, the Arduino commands the IMU to enter *streaming mode.* The Arduino reads off the calibrated data from the IMU data registers over the I2C interface. This is the operating mode the IMU will be in by default and during normal flight operations.

1. Remove power from the IMU
2. Connect the Arduino to the IMU board and FTDI UART cable to the Arduino
3. Open **IMU\_Streaming.ino** in the Arduino software
4. Set *calibratedOutput* to the desired value. When set to true, the outputs are the following
   * Col 1: Tick count
   * Col 2-4: Latest delta theta
   * Col 5-7: Latest delta velocity
   * Col 8-11: Integrated Attitude Quaternion
   * Col 12: Averaged Temperature (Celsius)
   * Col 13-15: Accumulated velocity

If *calibratedOutput* is false, the outputs are

* + Col 1: Tick count
  + Col 2-4: Current angular velocity (rad/s)
  + Col 5-7: Current specific force (m/s^2)

1. Set *outputDivider* to the desired value. The sample rate by default is 250 Hz (subject to change), so outputDivider = 10 will result in an output rate of 25 Hz.
2. Click **Upload**
3. Open **TeraTerm** and connect to the Serial port for the FTDI UART cable.
4. Go to **Setup->Serial port…** and set the baud rate to 230,400. Hit OK.
5. Go to **File->Log…** and enter in the desired file name. Check the **Plain Text** checkbox and uncheck the **Append** checkbox. Hit **Save**.
6. Reconnect power to the IMU
7. Once you are finished collecting data, either disconnect the Teraterm session or remove power from the IMU
8. Go to **File->Show Log Dialog…** and hit **Close** to close the log file.
9. Open MATLAB
10. Move the data file to your working directory in MATLAB
11. If you set *calibratedOutput =* true, run *ParseCSVDataCal.m*. Type ‘help ParseCSVDataCal’ for instructions on how to use function.
12. If you set *calibratedOutput =* false, run *ParseCSVDataRaw.m*. Type ‘help ParseCSVDataRaw’ for instructions on how to use function.

## Storing Data on SD card

In this mode, the Arduino commands the IMU enter ‘SD write’ mode. The IMU begins collecting raw and calibrated data and stores it on the SD card. This mode should be used for calibration and analysis of the IMU.

1. Remove power from the IMU
2. Insert a microSD card into the IMU board
3. Connect the Arduino to the IMU board
4. Reconnect power to the IMU
5. Open **CollectDataSD.ino** in the Arduino software
6. Set *acqTime* to the desired acquisition time
7. Click **Upload**
8. Open the Serial monitor and wait until the data acquisition has finished.
9. Disconnect power from the IMU
10. Remove the SD card and insert into your PC
11. Open MATLAB
12. Move the data file to your working directory in MATLAB
13. Run *ParseBinaryData*. Type ‘help ParseBinaryData’ for instructions on how to use function.

## Storing Data on SD card and Reading Data over I2C

In this mode, the Arduino commands the IMU to enter ‘SD write’ mode. The IMU begins collecting raw and calibrated data and stores it on the SD card. Once data has been collected for the allotted time, the Arduino commands the IMU to enter ‘SD read’ mode and the data is (slowly) read from the SD card over the I2C interface. This mode will be used on STF-1 to get on-orbit raw data. NOT RECOMMEND FOR COLLECTING DATA IN THE LAB.

1. Remove power from the IMU
2. Insert a microSD card into the IMU board
3. Connect the Arduino to the IMU board board and FTDI UART cable to the Arduino
4. Reconnect power to the IMU
5. Open **TeraTerm** and connect to the Serial port for the FTDI UART cable.
6. Go to **Setup->Serial port…** and set the baud rate to 230,400. Hit OK.
7. Go to **File->Log…** and enter in the desired file name. Check the **Binary** checkbox and uncheck the **Append** checkbox. Hit **Save**.
8. Open **ReadDataSDtoI2C.ino** in the Arduino software
9. Set *acqTime* to the desired acquisition time
10. Click **Upload**
11. The TeraTerm terminal should start flowing with gibberish characters. Once it stops, the data transfer is completed.
12. Go to **File->Show Log Dialog…** and hit **Close** to close the log file.
13. Open MATLAB
14. Move the data file to your working directory in MATLAB
15. Run *[imu\_raw, imu\_cal, err] = ParseBinaryData(‘<name\_of\_data\_file>’)*, replacing <name\_of\_data\_file> with the name of the text file.