ELE6902: PROJET DE MAÎTRISE II

DEVELOPMENT OF AN ARM/LINUX BASED TESTBED FOR RAPID CONTROL SYSTEM PROTOTYPING VIA MATLAB/SIMULINK

R. Dustin Kahawita
romesh.kahawita@polymtl.ca

POLYTECHNIQUE MONTRÉAL

LE GÉNIE EN PREMIÈRE CLASSE

Presented to

Prof. David Saussié

Prof. Jérome Le Ny

December 15th 2014

ABSTRACT

The following report details the work carried out by R. Dustin Kahawita in compliance with criteria for the course "Projet de maîtrise en ingénierie II (ELE6902)", required as part of the degree program "Masters in Engineering – Aerospace" at the University of Montreal – Polytechnique. Project supervision and guidance was carried out by Dr. David Saussié and Dr. Jerome Le Ny of the Department of Electrical Engineering.

This project aimed to develop a test-bed allowing for rapid hardware-in-loop prototyping of various control systems via Matlab/Simulink. The ARM-Cortex A8 equipped BeagleBone Black "system on board" (SoB), running embedded GNU/Linux, was selected as the target hardware for this endeavor. A Simulink block library, dubbed the "BLACKlink" library was created and includes standard blocks to control various functionality of the BeagleBone Black's GPIO pins and user LEDs. Also included in the BLACKlink library are drivers developed specifically for the sensor suite used as part of this project. The sensor suite, consisting of an accelerometer, gyroscope and magnetometer, was used in conjunction with a relatively new decent gradient sensor fusion algorithm to create a real-time attitude and heading reference system (AHRS). A user interface was developed in order to visualize live attitude estimation and sensory data. Finally, an overall evaluation of the developed test-bed was carried out.

All hardware and software specifications as well as setup procedures have been detailed in this report so as to help guide future projects aiming to use or expand on the functionality of this test-bed.

TABLE OF CONTENTS

1.0 LIST OF FIGURES

2.0 LIST OF TABLES

3.0 INTRODUCTION

Matlab is a powerful tool for the control engineer allowing for the design, testing, analysis and refinement of new and existing control algorithms. These algorithms may be applied to virtually any system consisting of electrical, mechanical, hydraulic and pneumatic components. Simulink serves as an excellent simulation tool to apply, visualize and analyze the performance of control algorithms as they are applied to mathematical models of physical systems. Although real world factors such as sensor noise or physical perturbations may also be modeled and introduced into the simulated system, they remain mathematical approximations. This project aims to facilitate the next step in the cyclic development process of testing, analyzing and refining by providing a platform for rapid deployment of control algorithms from Simulink onto a live physical system. This will allow for faster comparative analysis between simulated mathematical models and the actual behavior of their real-world counterparts by exposing the much dreaded limitations of physical systems versus ideal systems.

Over the past ten years, the rise of the "Maker Movement" along with improvements in processor technology has created the right market conditions for the introduction of several low-cost, open source single-board computers. Most prevalent among these are the Arduino and RaspberryPi which have spawned large user communities as well as countless, third-party-developed, open source add-on boards (called "Shields" or "Capes"). These add on boards are made to interface with specific electronic and electro-mechanical components for a variety of specialized projects. In this respect, the BeagleBone Black (BBB) is a relative newcomer to the affordable microprocessor market and hence its "ecosystem" is not as well developed. This is evident in the relatively smaller size of the user community and lack of tutorials, technical information and user created and supported functional blocks as compared to boards which have been on the market longer.

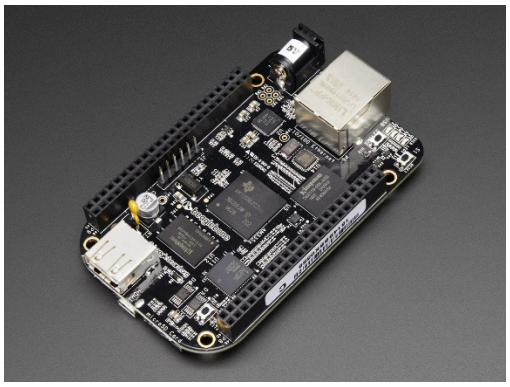


Figure 1: BeagleBone Black (Adafruit)

It should be noted that Simulink support for the RaspberryPi and Arduino boards does exist, whereas at the outset of this project, no such support existed for the BeagleBone Black. Simulink support does however exist for the BeagleBoard and BeagleBoard-Xm, the larger and more peripherally intensive predecessors to the BeagleBone and BeagleBone Black. This inherent functionality was tweaked for use with the BeagleBone Black platform.

Despite the lack of an expansive ecosystem, the BeagleBone Black's superior technical specifications, most notably a 1 GHz processor, positions itself as an ideal platform for deployment of Simulink models onto physical systems. This fact provided the initial motivation to pursue the objectives of this project.

4.0 HARDWARE SETUP

4.1 DEVELOPMENT STATION

The development of the test-bed was carried out on a MacBook Pro Retina (Mid 2012) running a Windows 8.1 (64-Bit) virtual machine via Parallels virtualization software. The virtual machine runs Matlab/Simulink version 2013a. Drivers for the BeagleBone Black's USB interface we downloaded and installed from the BeagelBoard.org website. Additional software installed for testing and development purposes include:

- Telnet / SSH clients:
  - PuTTY: Available at www.putty.org
  - SmarTTY: Available at http://smartty.sysprogs.com/
- Integrated Development Environment (IDE): Note that the installation of an IDE is optional and hence not required for the setup of the Simulink test bed. However, setting up a cross-platform tool chain was investigated and implemented during the course of this project as it eliminates the dependence on direct Matlab/Simulink support for the BeagleBone Black. Instead, a Simulink model may be converted to C-code via Matlab/Simulink Coder and then compiled and download directly to the BeagleBone Black via a properly configured IDE.
  - Eclipse: Available at https://www.eclipse.org/downloads/
  - Qt: Available at http://qt-project.org/

4.2 BEAGLEBONE BLACK

The BeagleBone Black is an open source development board equipped with an ARM AM335x 1GHz Cortex – A8 microprocessor. The microprocessor itself is capable of running both Linux and Android high level operating systems (HLOS). The BeagleBone Black (rev. B) includes 512MB of DDR3 RAM, 2GB of flash storage, a 3D graphics accelerator, a NEON floating point accelerator and two programmable real time unit (PRU) microcontrollers. Connectivity to the BBB is managed through USB or Ethernet with the possibility of Wi-Fi connectivity added via the USB host.

The following list highlights the hardware specifications of the BeagleBone Black:

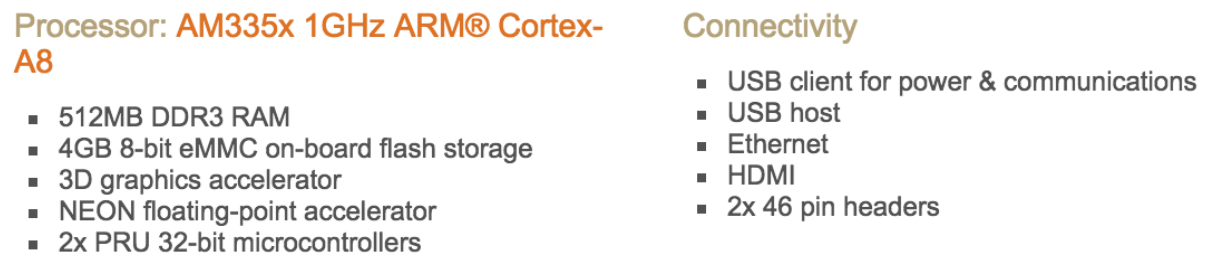


Figure 2: BeagleBone Black Hardware Specifications

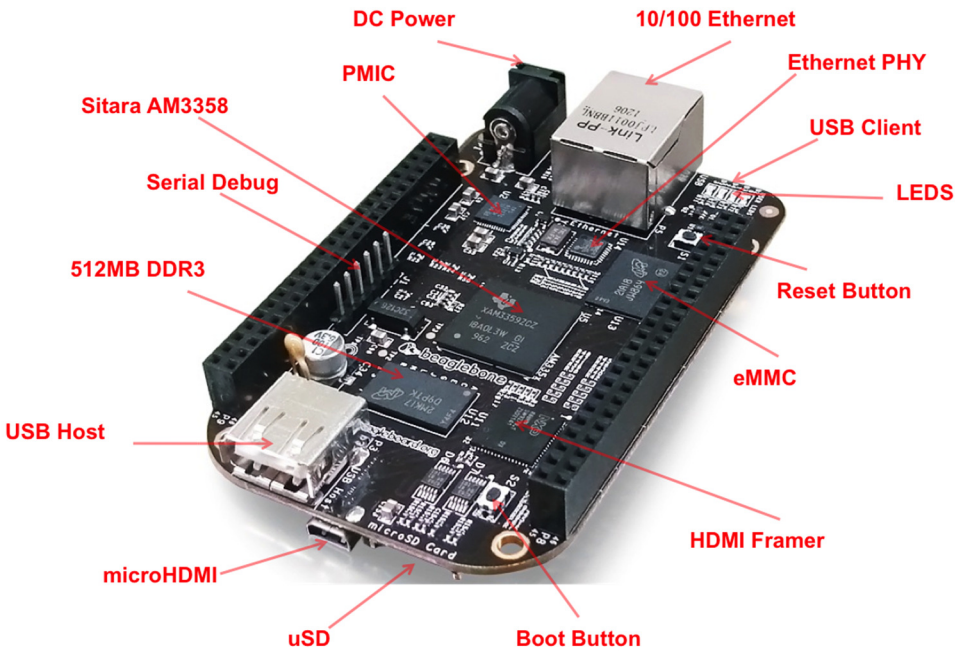


Figure 3: BeagleBone Black Hardware Details (Beagleboard.org)

One of the more impressive features of the BeagleBone Black are the many expansion headers that allow for various I/O capabilities including GPIO, Pulse Width Modulation (PWM), Analog input as well as communication via universal asynchronous receiver/transmitter (UART), serial peripheral interface (SPI) and Inter-Integrated Circuit (I2C) protocols.

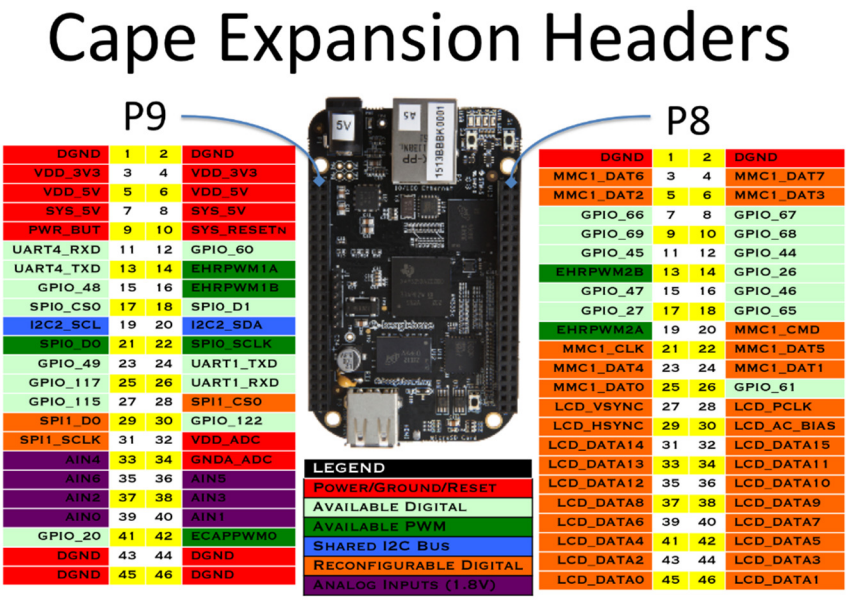


Figure 4: BeagleBone Black Expansion Headers (Beagleboard.org)

The version of the BeagleBone Black available at the outset of this project was revision "B". Although currently revision "C" is available, all procedures and development detailed in this report may be completely replicated on the latest version without any issues. The difference between the two versions is primarily relegated to a faster and larger flash memory (eMMC) of 4GB on revision "C" as opposed to 2GB on the earlier revisions. Revision "C" also now ships with a Debian distribution of embedded Linux whereas the previous version shipped with the less user-friendly Angstrom Linux operating system.

The BeagleBone Black allows the option of booting either from the onboard eMMC or via a MicroSD card. By default, the board boots from the onboard eMMC unless bootable media is detected in the MicroSD slot. During this project, given the 2GB limitation on eMMC storage and the necessity of running a different version of embedded Linux (Ubuntu) from that installed by default on the BBB's eMMC (Angstrom), the option to boot from a MicroSD card was used. This allowed for keeping the eMMC with the original operating system in tact as a safety along with the ability to backup and clone the MicroSD card at milestone intervals or before conducting tests with risky outcomes. Note the final image of the MicroSD card used in this project is available on the GitHub website eliminating a fair amount of setup time (link found in conclusion)

The BBB may be powered via the USB connection which provides approximately 0.5 – 0.9 Amps at 5VDC. Alternatively an external 5VDC output/ 110VAC input switching supply power connection may be used via the barrel jack connector for larger current-draw applications. Although both setups were tested during this project, the former was used more consistently as the power draw for the complete sensor suite was well within the capacity for the USB port.

Communication between the BeagleBone Black and the development station was tested via USB, Ethernet cable and a Wi-Fi adaptor (running the rtl8192cu chipset). All communication media worked at a basic level, however communication over Wi-Fi proved slow and unreliable at times. This may have been due to insufficient current provided by the switching power supply which was rated at 2 Amps (the minimum recommendation for a current supply when using a Wi-Fi adaptor with the BBB). Hence, the most commonly employed set up for this project used a USB cable between the board and development station.

4.3 SENSORS

The overarching objective of this project is to develop the BeagleBone Black as a viable platform for implementing Simulink designed control systems. In order to test and prove this platform's viability, IMU and AHRS systems were implemented. This involved the integration of an accelerometer, angular rate sensor (gyroscope), and compass (magnetometer). Initially, specific drivers were developed for each sensor, from which generic drivers were derived for basic I/O functionality and later added to the BLACKlink library (e.g. reading analog inputs, read/write over I2C to a specific device and register address).

4.3.1 ADXL330 Accelerometer

The Analog Devices ADXL330 accelerometer was selected for the AHRS. The ADXL330 is sold by Dimension Engineering as a breakout board in DIP-16 form factor under the product number DE-ACCM3D. The ADXL330 is a tri-axis accelerometer with a sensing range of +/- 3g and an analog sensitivity of 330mV/g (under an input voltage of 3.3V). Voltage outputs are set on three separate pins for the X/Y/Z axis components as shown in the figure below.

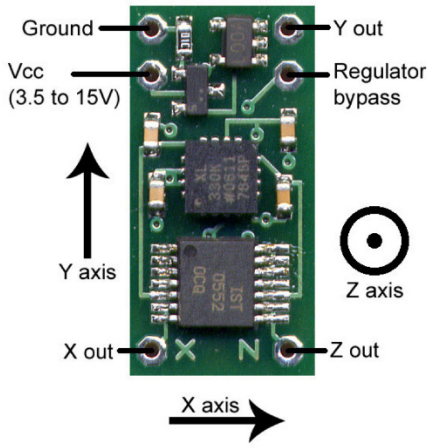


Figure 5: DE-ACCM3D Accelerometer Breakout Board (Dimension Engineering)

Note that when the accelerometer is static, the acceleration measured corresponds to the earth's gravitational field and hence tilt angle and orientation can be derived. However, should the sensor experience additional external acceleration, determining its orientation by considering only the gravitational vector will not yield accurate results. Therefore additional sensors must be included in the sensor suite along with a filtering algorithm. The following figure demonstrates how the accelerometer's voltage responds given its static orientation.

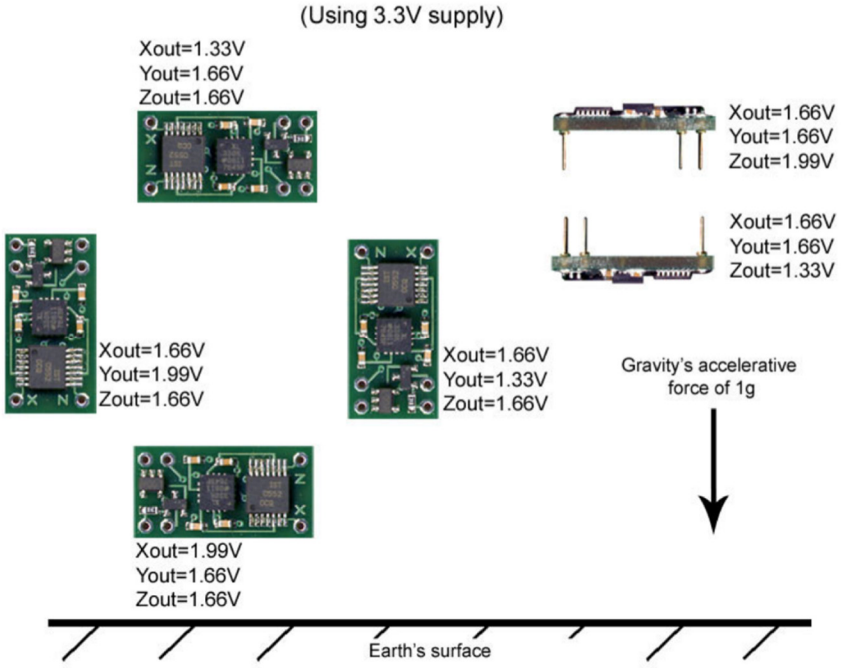


Figure 6: Accelerometer Static Orientation-Based Voltage Levels (Dimension Engineering)

Acceleration data must therefore be read by the BeagleBone Black via three separate analog input pins. Note that the maximum input voltage allowed for each analog input pin is 1.8V. Given the sensitivity rating and sensing range of the accelerometer the output voltage on each pin may vary from 0.666V to 2.66V with the zero gravity point at 1.66V for a 3.3V input source. Clearly this is above the acceptable range for the BeagleBone Black's analog pins and so a voltage divider circuit must be employed. A voltage divider, as its name suggests, creates an output voltage

equal to a fraction of the input voltage by using resistors in series to create the required voltage drop. A voltage divider circuit may be illustrated as:

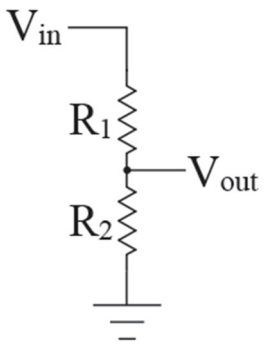


Figure 7: Voltage Divider Circuit

The output voltage may be calculated as:

$$V_{out} = \frac{R_2}{R_1 + R_2} V_{in} \qquad (4.1)$$

For the accelerometer application, two 1KOhm resistors were selected in order to divide the input voltage in half. This yields a maximum output voltage of 1.33V which is under the analog pin voltage rating of 1.8V. The maximum input current for the analog pins is then relegated to 2.66mA which is well under their maximum input of 8mA.

Finally, in order to read the analog voltage provided by the accelerometer, the corresponding header pins must be correctly muxed to analog input mode. This will be discussed in greater detail in the section outlining device tree overlays.

4.3.2 L3G4200D Gyroscope

The ST Microelectronics L3G4200D gyroscope module was selected as the angular rate sensor for the AHRS. The L3G4200D is sold in a breakout board format for easy breadboard mounting by Parallax Inc.

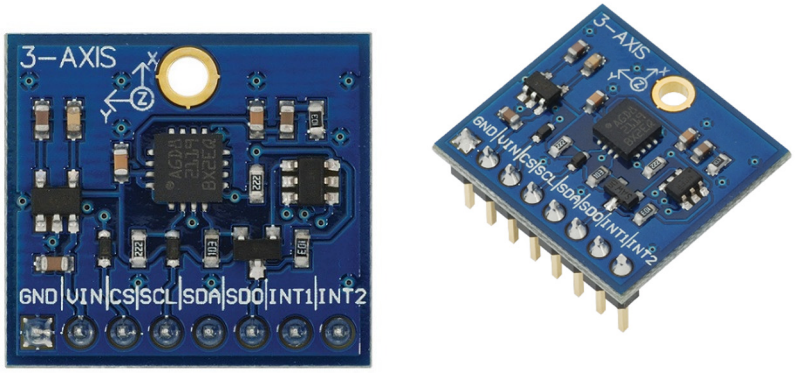


Figure 8: L3G4200D Gyroscope Breakout Board (Parallax)

The module itself is capable of measuring angular rates about its X, Y and Z axes at three selectable angular rate ranges of 200, 500 or 2000 degrees per second with 16 bit analog to digital conversion. The gyroscope module includes an embedded temperature sensor allowing for on-the-fly compensation of sensitivity due to the difference between ambient and calibration temperature. Communication between the gyroscope and BeagleBone Black may be handled via the SPI or I2C protocols. The module is to be powered by a supply voltage of between 2.7 to 6.5VDC.

4.3.3 HMC5883L Compass

The Honeywell HMC5883L magnetometer was selected as the magnetic heading sensor for the AHRS system. As will be detailed in a subsequent section, originally an inertial measurement unit (IMU) was developed for attitude derivation however, as expected, it lacked accurate heading indication due to accumulated yaw rate integration error. The magnetometer was thus added in order to allow the implemented decent gradient algorithm to provide both accurate attitude and heading data. The HMC5883L magnetometer is sold in breakout board format by Adafruit.

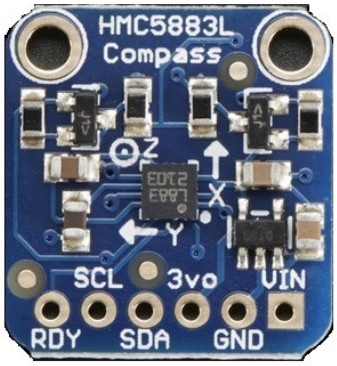


Figure 9: HMC5883L Compass Breakout Board (Parallax)

The magnetometer is capable of measuring the magnetic field components along its X, Y and Z axes within a maximum range of +/- 8 gauss and with 12 bit analog to digital conversion. This range, along with the magnetometer's sensitivity, sampling rate and averaged sampling size may be configured via the sensor's registers. The sensor also includes offset straps which may be used to create an artificial magnetic field bias on the sensor via an applied DC current. This allows the compass to execute a self-test by comparing values read from the artificial magnetic field against those of the ambient magnetic field. The difference in measurement is used to estimate and compensate for temperature effects on the magnetometer's sensitivity. Communication between the compass module and BeagleBone Black is handled via the I2C protocol. The module may be powered by a supply voltage of between 3.3 to 5VDC.

4.3.4 Sensor Suite Summary

The following table presents a summary of the key parameters for the sensor suite used in the IMU and AHRS systems.

Sensor	**ADXL330 Accelerometer**	**L3G4200D Gyroscope**	**HMC5883L Magnetometer**
Axis	X/Y/Z	X/Y/Z	X/Y/Z
Field Range	± 3 g	200/500/2000 dps	± 8 gauss
Output	333 mV/g @ 3.33V	16-Bit ADC	12-Bit ADC
Bandwidth / Data Output Rate	500 Hz	100/200/400/800 Hz	75 Hz (Continuous Measurement Mode)
Communication	Analog Output	I^2C @ 400 kbps	I^2C @ 400 kbps
Supply Voltage	2 – 3.6 VDC	2.7 – 6.5 VDC	3.3 – 5 VDC
Average Current Draw	0.9 mA	6.1 mA	0.1 mA

Table 1: IMU/AHRS Sensor Suite Specifications

4.4 EMBEDDED COMMUNICATION PROTOCOL

As mentioned in the previous section, the L3G4200D Gyroscope allows for the option of using both the SPI and I2C protocols whereas the HMC5883L Compass has been designed with only I2C functionality. Therefore, the I2C communication protocol was used for both the gyroscope and compass devices during this project.

The I2C protocol consists of a common serial clock bus line (SCL) and a bidirectional serial data bus line (SDA) between master and slave devices. Note that a master device is simply defined as the device which initiates the data transfer, therefore a single device may act as both a slave and master. In the case of the developed AHRS, the BeagleBone Black is consistently cast as the master device while the gyroscope and compass act as slaves. The data line is bidirectional and hence is capable of sending and receiving data to and from multiple slave devices sent over the SDA bus with each slave device identified via its own unique 7-bit address. The data transmitted over the SDA line is divided into 8-bit (1 byte) packets. The speed of data transmission is typically set to 100 kbps, 400 kbps or 3.4 Mbps represented as standard, fast or high speed modes respectively. Both I2C enabled sensors used in this project are run at 400 kbps.

Physically, both the SCL and SDA lines are open drain with both lines being “pulled-up” to the VCC voltage via two pull up resistors. Therefore, the master or slave devices may only drive the line signals low by shorting them to ground. This is illustrated in the following figure:

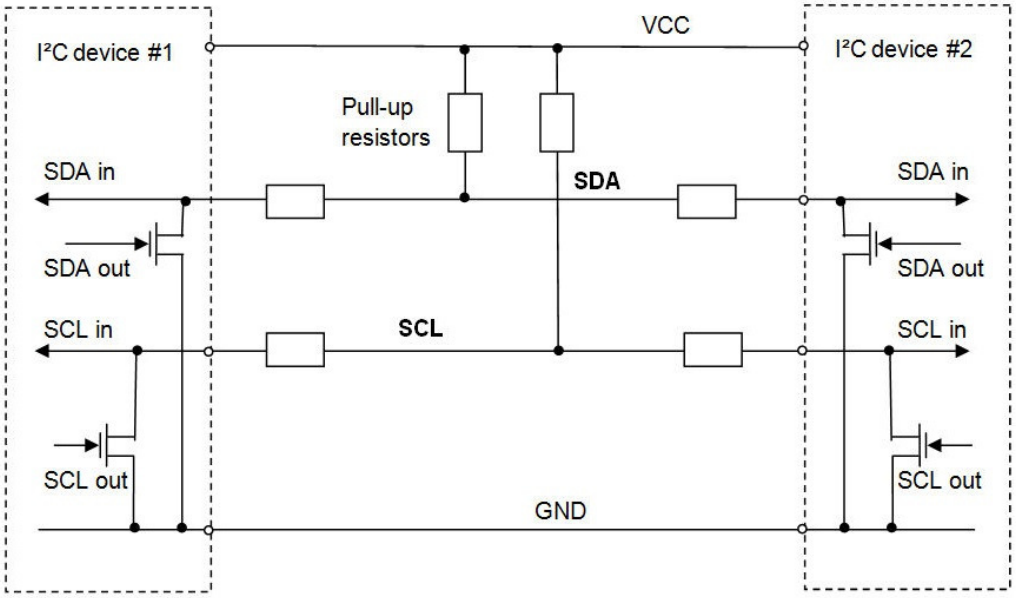


Figure 10: I2C Hardware Configuration [15]

Data exchange between the master and slave devices is initiated and terminated with a signature of SCL and SDA line configuration defined by the I2C protocol and illustrated in the figure below.

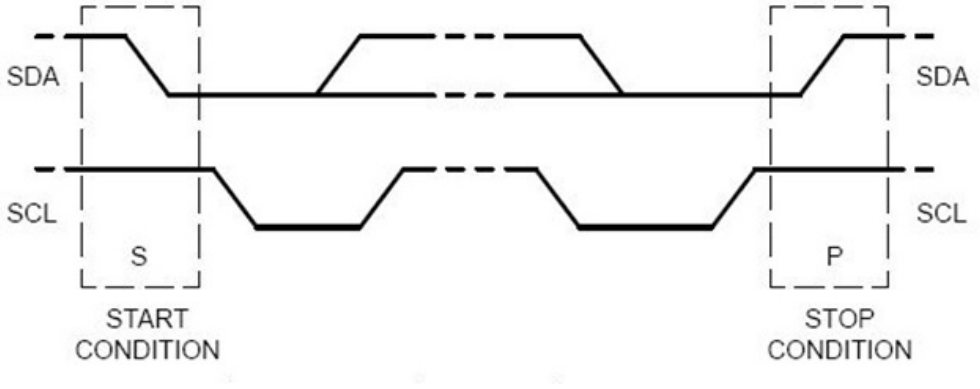


Figure 11: I2C Start / Stop Signatures [15]

To begin transmission, the master will pull down the SDA line while keeping the SCL line high. This grabs the attention of all slave devices on the bus which places them into listening mode. Next the master broadcasts the 7-bit address of the slave it wishes to communicate with along with a RD/nWR bit indicating whether it wishes to read-from or write-to the slave. All slaves then compare the broadcasted address with their own. Should the address values match, the device will transmit an acknowledge signal. If the addresses do not match, the device will simply wait for the bus to be released via the stop SCL/SDA signature shown in the above figure.

The clock signal is controlled by the master and used to discretize the data being transmitted to or from the slave. This is to say, that outside the start and stop signatures, the SDA line is held consistently high or low while the SCL line is high. The SDA line may change value while the SCL line is low in order to represent the logical value (true/false) of the subsequent bit. When the SCL line is high, the logical value is read from either the master or slave depending

on the read/write settings. The speed at which the SCL line pulses to indicate a new bit of SDA data is defined by the speed of the bus. This concept is illustrated in the figure below:

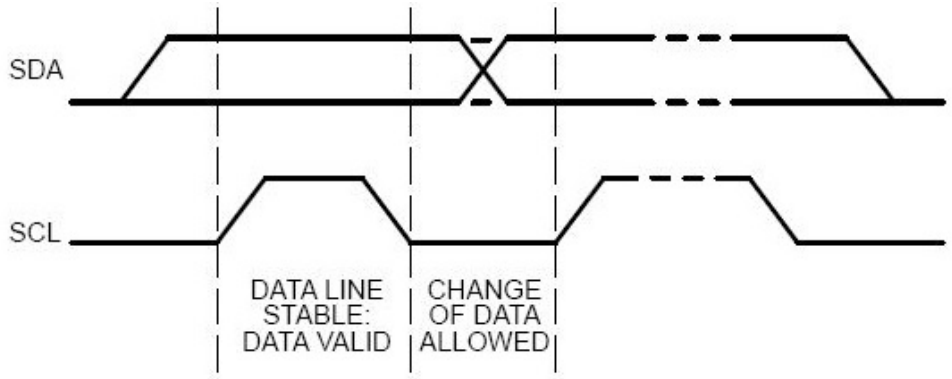


Figure 12: I2C SDA & SCL Sequencing Rules for Data Transmission [15]

Therefore transmission of a complete data byte over the I2C bus will appear as follows, where "D0" – "D7" represent discreet true/false logic bits and "ACK" represents the acknowledge bit. The acknowledge bit is set low from the device receiving the data stream if all data has been received and it is ready to receive the next data byte. A high value retuned in the acknowledge bit indicates that the receiving device is not ready to receive data and the transmission should be aborted.

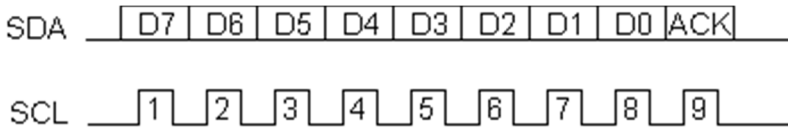


Figure 13: I2C Signature for Data Transmission [15]

4.5 THE SYSFS FILE SYSTEM

Unlike the BeagleBone Black's competitors, no API library exits to interface directly with the C-code written in the S-Function block. For example, the Arduino module functionality may be controlled directly by including a pre-existing header file "Arduino.h" which consists of function declarations allowing for operations such as configuring pin modes and executing digital read/write commands.

The BeagleBone Black, interfaces with its hardware peripherals via the "sysfs" file system included in the Linux kernel. The "sysfs" file system acts as a channel mapping internal (kernel) subsystems to external userspace for manipulation by system users or applications. Internal kernel objects, object attributes and object relationships are respectively mapped to directories, regular files and symbolic links. It is important to note that information structured in userspace is formatted in simple ASCII strings for easy access and manipulation by various applications. Therefore controlling devices on the BeagleBoard distills down to reading and writing data to and from sysfs files.

As is standard convention on almost all Linux distributions, the "sysfs" filesystem is mounted in the "/sys/" Linux directory. Within this directory lies all major subsystems registered within "sysfs". In the case of the BeagleBone Black, the directory structure appears as:

/sys/

| -- block
| -- bus
| -- class
| -- dev
| -- devices
| -- firmware
| -- fs
| -- kernel
| -- module
| -- power

A brief description of each directory is provided below:

- **Block**: Contains subdirectories for each block device discovered on the system. Block devices critical to the overall performance of the system and are typically used for the storage of application code and data as well as user data.
- **Bus**: Contains the subdirectories for each physical bus type supported in the current kernel of the Linux system. For example, the BeagleBone Black's bus directory includes folders for i2C and SPI bus types.
- **Class**: Contains directories which describe a functional type of device. For example, the BeagleBone Black has directories for its GPIO pins as well as for its USRLEDs. Note that this directory is therefore used when writing the C-code for several of the S-Function driver blocks. The subdirectories of each device typically contain symbolic links to the device and driver directories in the global device hierarchy which may be found in the "/sys/devices" directory.
- **Dev:** Contains two folders: "char" and "block" which provides a method to look up the sysfs path for a device using the "stat()" command line function. The files located in each directory are symbolic links with point to the corresponding sysfs path for the given device.
- **Devices:** Contains the global hierarchy of all physical devices that have been discovered by all bus types available on the Linux system. This is to say all bus types that have been registered within the kernel.
- **Firmware:** Contains subdirectories allowing for viewing and manipulation of objects and attributes related to the system's firmware. The system's firmware is the software that is executed upon power-on such as the system bios.
- **Fs:** Contains information and settings regarding each filesystem mounted on the system. Each directory represents a filesystem type and holds information for each device or partition formatted with that specific file system.
- **Kernel:** Contains files and subdirectories pertaining to settings, information and security policies for the system.
- **Module:** Contains subdirectories for all modules loaded on the system kernel. Modules are code snippets that may be dynamically loaded/unloaded to/from the system kernel. This basically extends the functionality of the kernel without having to reboot the system.
- **Power:** Contains information on the systems power state as well as information regarding the number of times the system has hibernated/slept etc.

In the case of this project, C-code deployed to the BeagleBone Black is used to interact with the sysfs filesystem via the following core functions:

- Open/Fopen: Used to connect to the sysfs file stream for reading or writing values
- Write/Fwrite: Used to write a specific value to the opened sysfs file
- Read/Fread: Used to read a specific value from the opened sysfs file
- Close/Fclose: Used to close the connection to the sysfs file stream

All Simulink function blocks and device drivers developed during the course of this project employ the use of sysfs file manipulation to toggle user LEDS, GPIO I/O modes and communicate over the I2C bus. All code has been included in the appendix so the exact application process may be duplicated. Where possible, standard functions were created and significant error handling and debugging code has been included.

4.6 PROTOTYPING TEST BED

The order to test the functionality of drivers developed for the BeagleBone Black, a solderless breadboard was used to mount and connect the accelerometer, gyroscope and magnetometer sensors. Additional circuits were all also added in order to test the functionality of digital I/O read and write. Note that given the development of an IMU and AHRS, the orientation of the sensors on the breadboard is of concern and hence all sensors were oriented with their X, Y and Z axis intersecting or in-line with each other. The following photo demonstrates how all components were mounted and connected to the BeagleBone Black. Note the red and black wires connect to 3.3 Vdd and ground respectively while the paired blue and yellow wires between the gyroscopes, compass and header is the I2C bus.

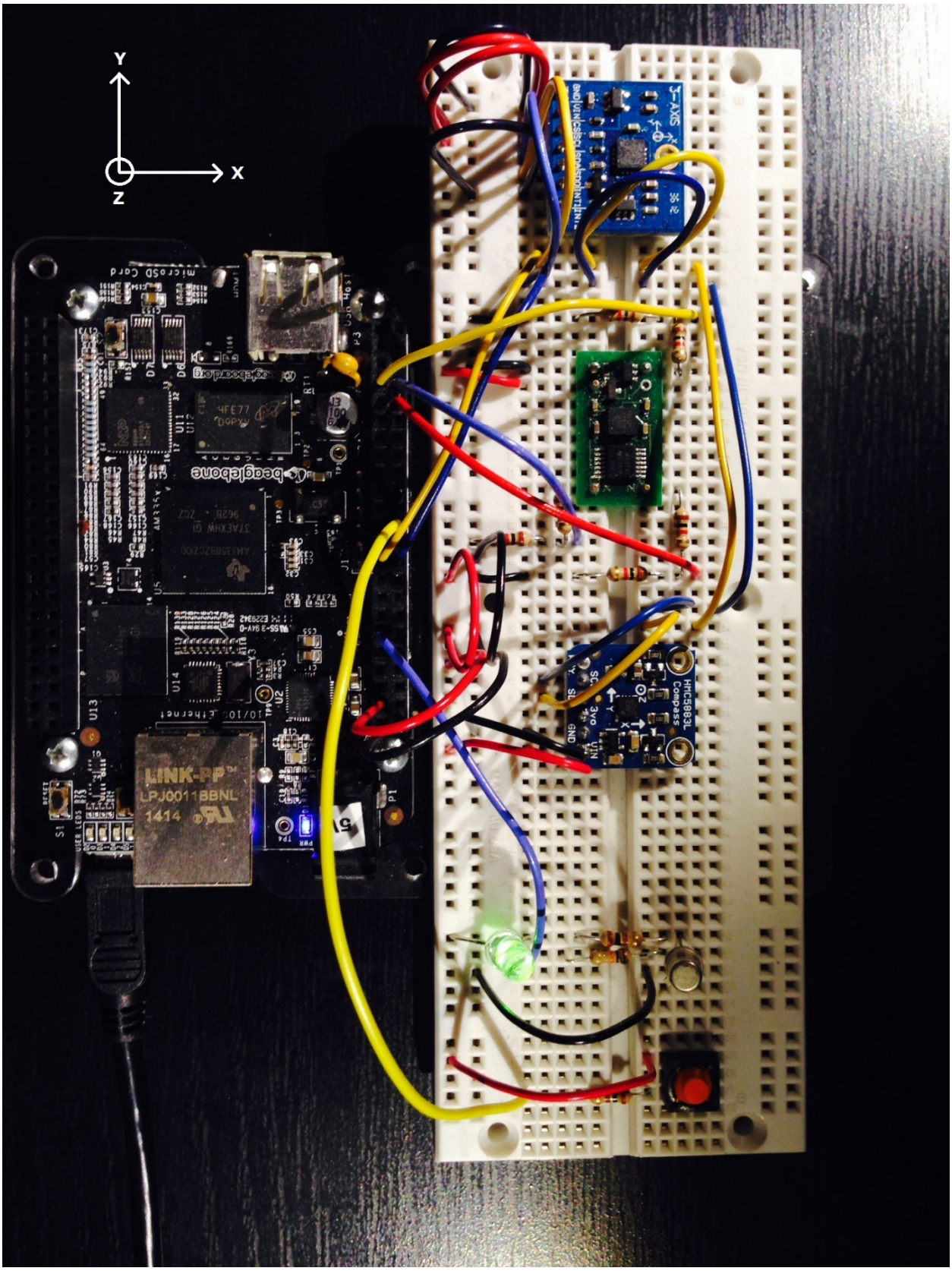


Figure 14: Prototype Board Setup

5.0 SOFTWARE SETUP

5.1 INSTALLATION AND SETUP OF BEAGLEBOARD SUPPORT PACKAGE FOR MATLAB/SIMULINK

The capability to program the BeagleBone Black from a desktop computer is typically managed by what may be referred to as a toolchain. A toolchain consists of a group of development tools including program editors and compilers, libraries and debuggers some of which may be built into an integrated development environment (IDE).These tools are used both in parallel and sequentially, allowing for a program to be written, debugged, compiled and deployed to a target device from a development station such a PC. In some cases, the development station will run on a different operating system than that of the target hardware, therefore requiring the need for a slightly more complicated "cross—compiler" toolchain. Although not detailed in this report, a cross-compiler toolchain was developed allowing for C/C++ programs written within the Eclipse IDE running on a Windows 8.1 platform to be deployed onto a target system running embedded Linux.

Matlab provides a support package which acts as a proprietary toolchain capable of converting Simulink and Matlab code into embedded Linux executable files. This is carried out in a relatively seamless manner, significantly facilitating the deployment and tuning of Simulink programs while avoiding the hassle of configuring various elements of a cross-compilation toolchain. Unfortunately, at the outset of this project, the existing support package was available only for the BeagleBoard xM and BeagleBoard – Cx hardware platforms but not for the BeagleBone Black platform. As such an ad-hoc procedure was developed in order to allow the BeagleBone Black to capitalize on the inherent toolchain capability of the existing support package.

Procedure:

1) The support package for the BeagleBoard may be downloaded free of charge from the MathWorks website [4] or installed via the Matlab command line via the command "targetinstaller". Select "BeagelBoad" from the Support Package Installer window. Note that a MathWorks account ID is required for this method of installation. The support package installer is shown below.

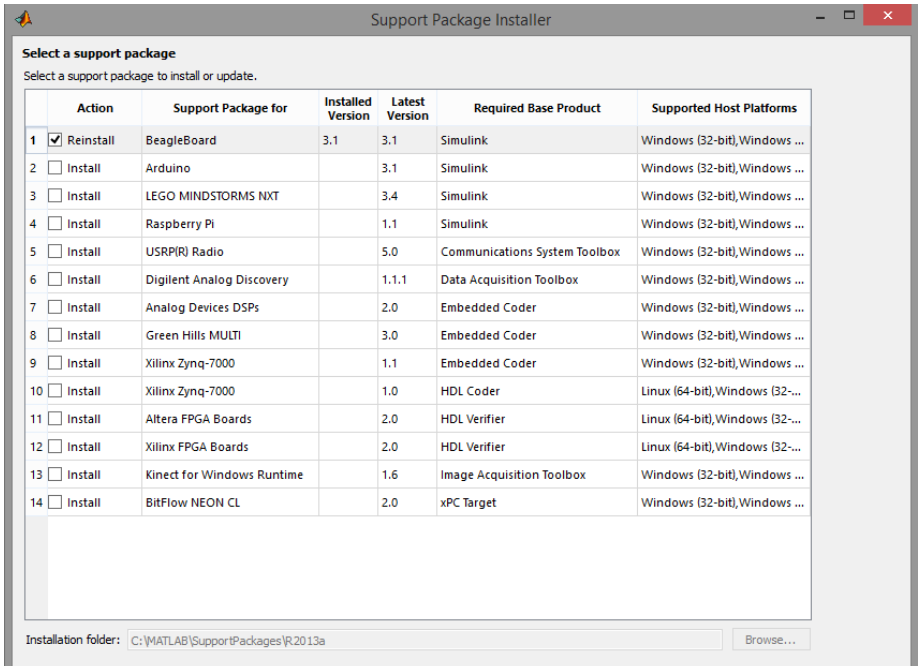


Figure 15: BeagleBoard Support Package Install

Once installed, a new Simulink library will appear in the Simulink Library Browser entitled "Simulink Support Package for BeagleBoard Hardware". The library consists of eleven standard blocks with functionality ranging from control of the on-board LEDs to the GPIO ports of the BeagleBoard. As these blocks are pre-developed and masked they cannot be modified to work with the BeagleBone Black and hence new blocks must be created.

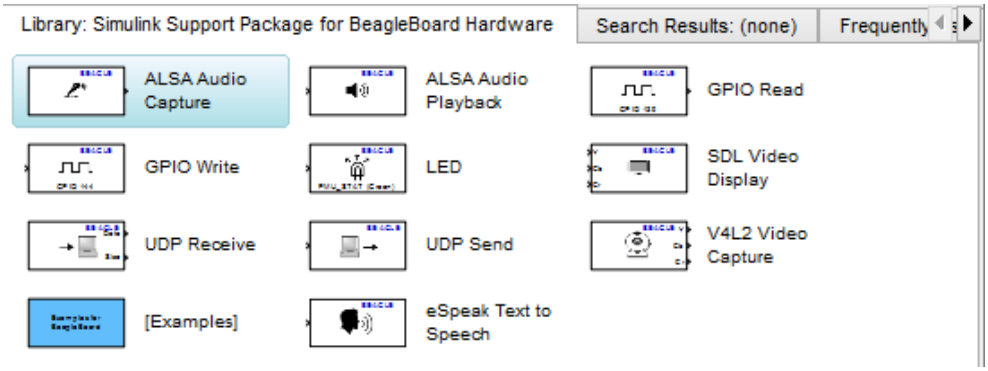


Figure 16: BeagleBoard Support Package Standard Blocks

2) Next, the toolchain used by Simulink must be set-up. This is carried out via the configuration of a "makefile" which essentially specifies the used software within the toolchain allowing for an automated build process. Typing "xmakefilesetup" in the Matlab command window brings up the XMakefile User Configuration window. For Matlab versions 2012b and 2013a/b configure the "Template" and "Configuration" options as shown below:

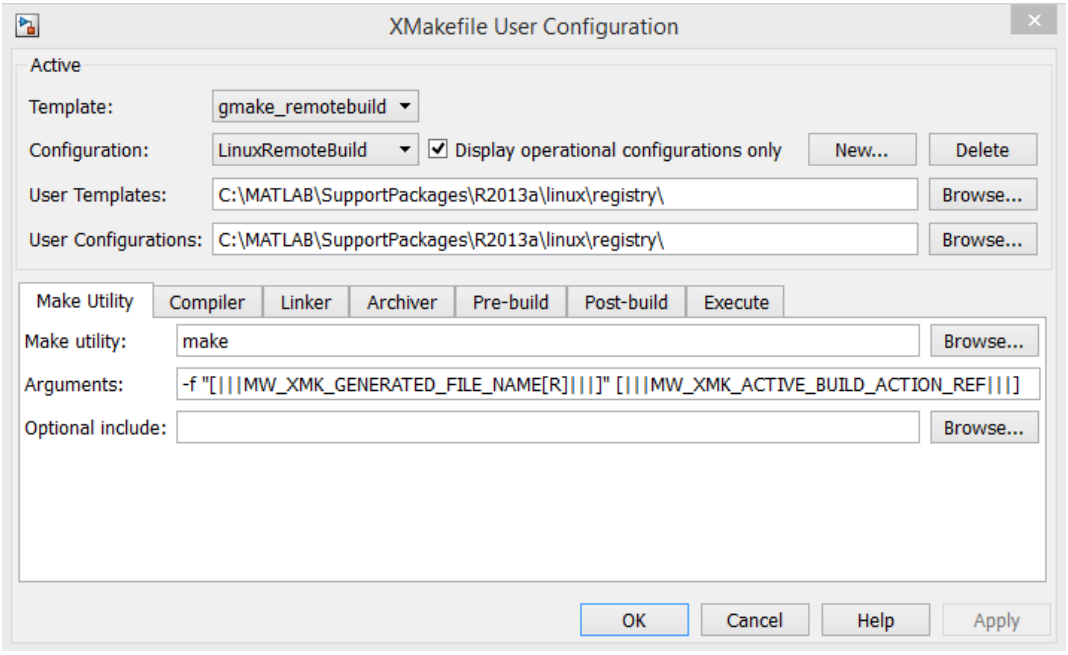


Figure 17: Matlab Make File Configuration

Finally, ensure that the compiler is configured as follows:

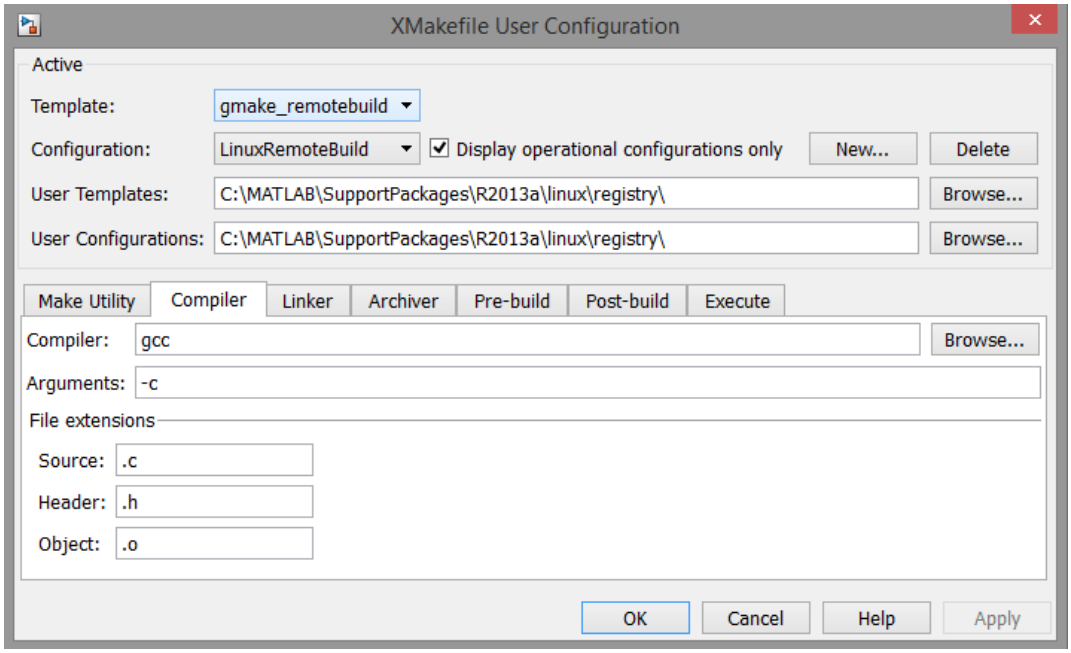


Figure 18: Matlab Compiler Configuration

5.2 MOUNTING UBUNTU ON THE BEAGLEBONE BLACK

The BeagleBoard support software was built so as to allow for deployment of Simulink models to the target hardware running an *Ubuntu* distribution of embedded GNU/Linux. As such, in order to capitalize on the toolchain-like capabilities of the support software, Ubuntu must be installed on the BeagleBone Black.

As mentioned earlier, the BBB rev. B is shipped with the Angstrom Linux distribution pre-installed on its eMMC. Two options exist for installing Ubuntu on the BeagleBone Black. The first is to install the operating system directly onto the onboard eMMC which, through the eMMC flashing process, permanently erases the pre-installed Angstrom Linux distribution. The second option is to create a bootable image on a micro SD card and instruct the BeagleBone's firmware to boot directly from the micro SD card instead of its eMMC. The latter option was select for this project as it offered greater flexibility in reverting to the original system in the case of undesirable behaviour or missteps in the testing and development process.

The following instruction list details the process for the installation and configuration of Ubuntu 13.10 – *Saucy Salamander* for booting from the micro SD image. Note that later versions of Ubuntu may also be used.

Required Downloads:

- *Ubuntu 13.10 for BeagleBone Black* micro SD card image
  - https://rcn-ee.net/deb/microsd/saucy/BBB-ubuntu-13.10-2013-12-17-4gb.img.xz
- *Ubuntu 14.10 for BeagleBone Black* image
  - *http://www.armhf.com/boards/beaglebone-black/#trusty*
- *Win32 Disk Imager* for writing image files to USB and SD flash storage drives
  - http://sourceforge.net/projects/win32diskimager/files/latest/download

- *7-Zip* open source file compression / decompression software
    - http://www.7-zip.org/
- *SmarTTY* SSH Client for communication with BBB Bash shell
    - http://smartty.sysprogs.com/

Procedure:

1) Download the Ubuntu SD card image onto development machine
2) Download and install Win32 Disk Imager and 7-Zip onto development machine
3) Decompress the Ubuntu image file via 7zip
4) Using Win32 Disk Imager, write the Ubuntu image file to the micro SD card
5) With the BBB powered down, insert the micro SD card into the onboard micro SD port. On power up, hold the "Boot" button down until USRLEDs 1 and 2 appear in solid blue. Note that recent tests indicate holding down the "Boot" button during power-on is not required. The BBB seems to detect a bootable drive in the micro SD port and automatically boots from it.
6) Once the BBB appears to enter normal operating mode (indicated by the regular heartbeat flash on USRLED 0), connect to it via the SmarTTY SSH client. Note that the BBB's IP address will vary depending on its network configuration. Log on to the BBB with default username "**ubuntu**" and password "**temppwd**". If executed correctly, a screen similar to the following should appear:

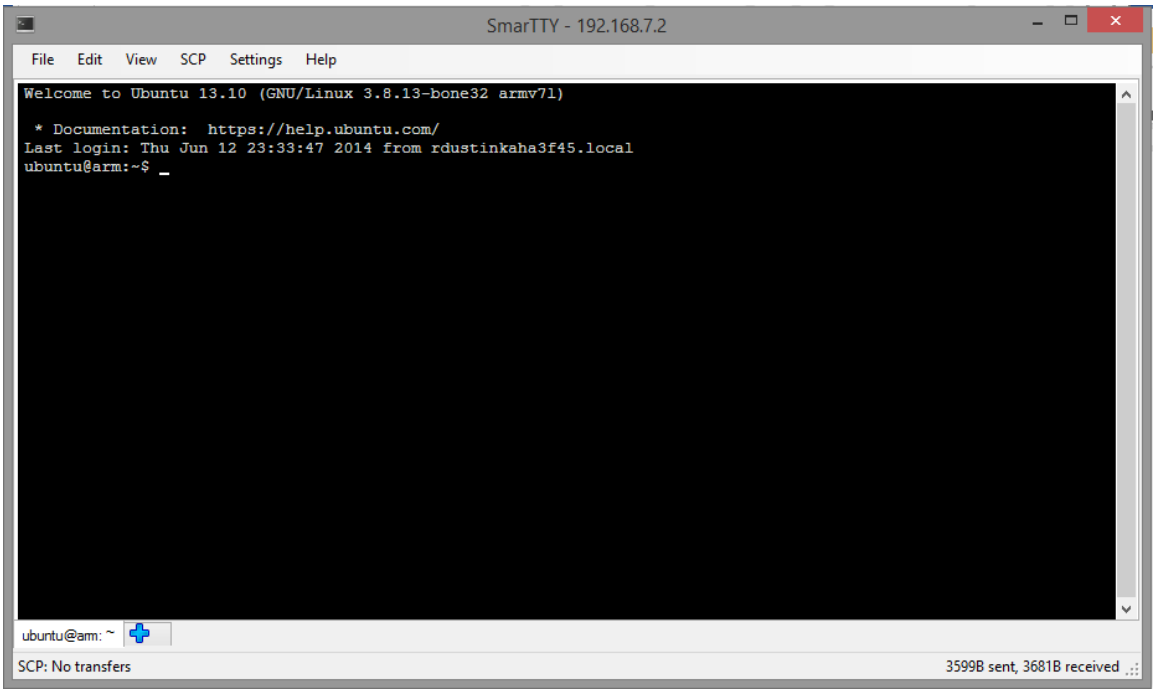


Figure 19: Logging on to the BeagleBone Black

7) Next, a swap file is created in order to boost the system memory. This proves advantageous when the system is under the stress of executing programs locally and communicating with Simulink on the host machine. A swap file of 1024 MB was created via the following BASH commands:
    a. "sudo su" – Places focus in root environment and ask for user password instead of root password
    b. "mkdir -p /var/cache/swap/" – creates swap directory

c. “dd if=/dev/zero of=/var/cache/swap/swapfile bs=1M count=1024” – The “dd” command copies a file while converting the format of the data in the process. The “bs=1M” sets the individual read and write bytes size to one megabyte. The “count” operand sets the total copy size of the input blocks to 1024 Megabytes.

8) The swap file is set to load automatically on boot by adding it to the file systems table via the following BASH commands:
   a. “nano /etc/fstab” – This opens the “fstab” file in the built-in Nano text editor
   b. “/var/cache/swap/swapfile none swap sw 0 0” – This line must be appended to the “fstab” file as shown below. Save and exit the Nano editor once the text has been entered.

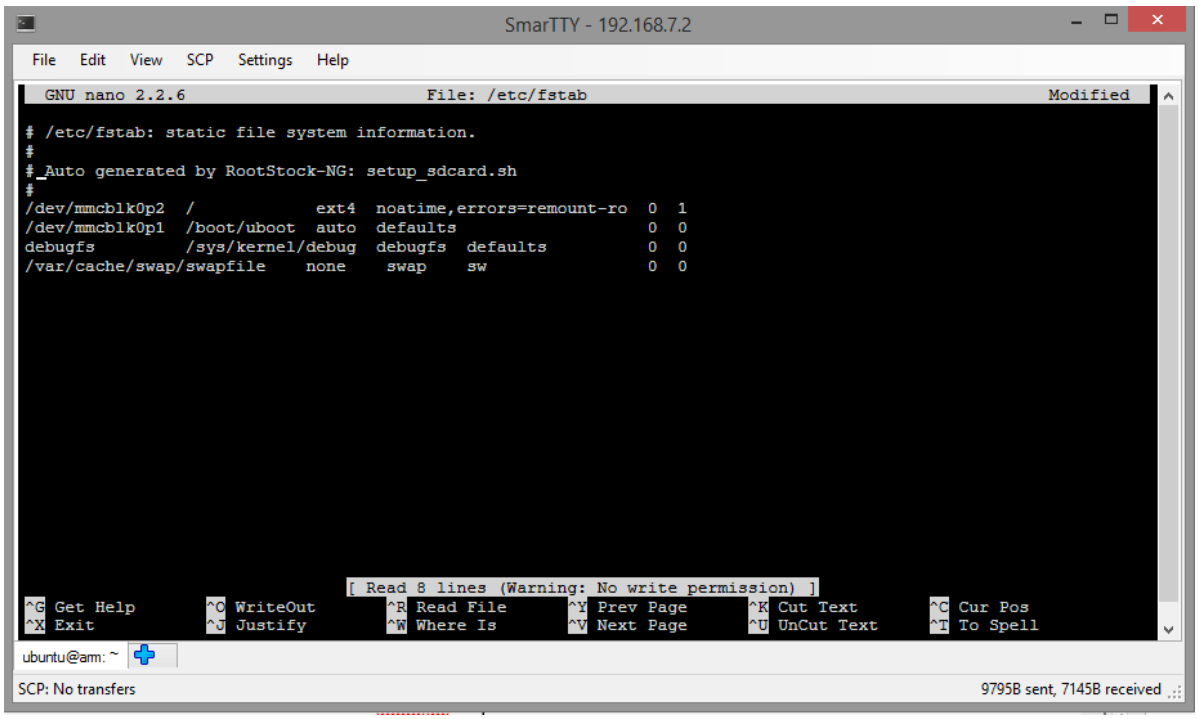


Figure 20: Swap File Creation via Nano Editor

9) The final step in configuring the BBB for operation with Simulink is to set up the build environment. This involves updating and installing all required embedded Linux software via the Ubuntu repositories as well as creating symbolic links between files. Software updates will automatically download and install the requested software. Note that the Beagle Bone Black must be connected to the internet either directly via its Ethernet port or through a bridged network via its UBS port. The following commands are executed in BASH.
   a. “apt-get update” – Downloads all package lists from the repositories and updates to the newest versions of packages and their respective dependencies.
   b. “ln -s /usr/include/arm-linux-gnueabihf/sys /usr/include/sys” – Create symbolic link
   c. “ln -s /usr/include/arm-linux-gnueabihf/asm /usr/include/asm” – Create symbolic link
   d. “ln -s /usr/include/arm-linux-gnueabihf/bits /usr/include/bits” – Create symbolic link
   e. “ln -s /usr/include/arm-linux-gnueabihf/gnu /usr/include/gnu” – Create symbolic link

5.3 SETTING UP THE SIMULINK ENVIRONMENT FOR THE BEAGLEBONE BLACK

With the Matlab/SIMULINK tool chain and BeagleBone Black build environment configured, the focus now shifts to the Simulink simulation environment. The following steps allow Simulink to generate the equivalent C code of the Simulink mode, download it to the BeagleBone Black, compile, build and run it on the target hardware. The process also allows Simulink to continually read and write data variables while the model is executing on the target board.

Procedure:

1) On the Simulink toolbar, select *Tools → Run on Target Hardware → Prepare to Run*

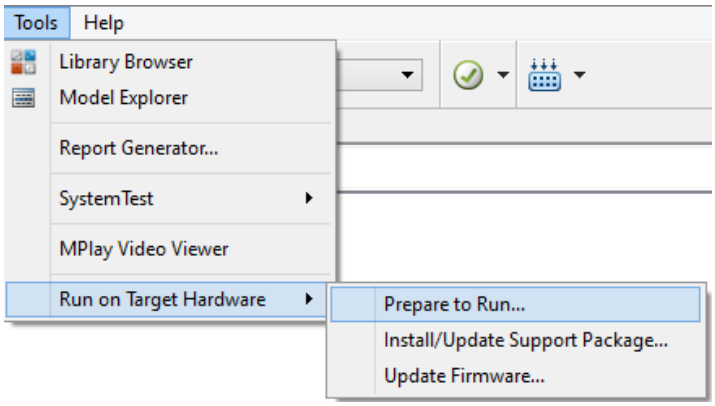


Figure 21: Simulink Environment Setup - Run on Target Hardware

2) The Configuration Parameters menu box for "Run on Hardware" will appear. Select "*Run on Target Hardware"* and configure the parameters as show below:

3)

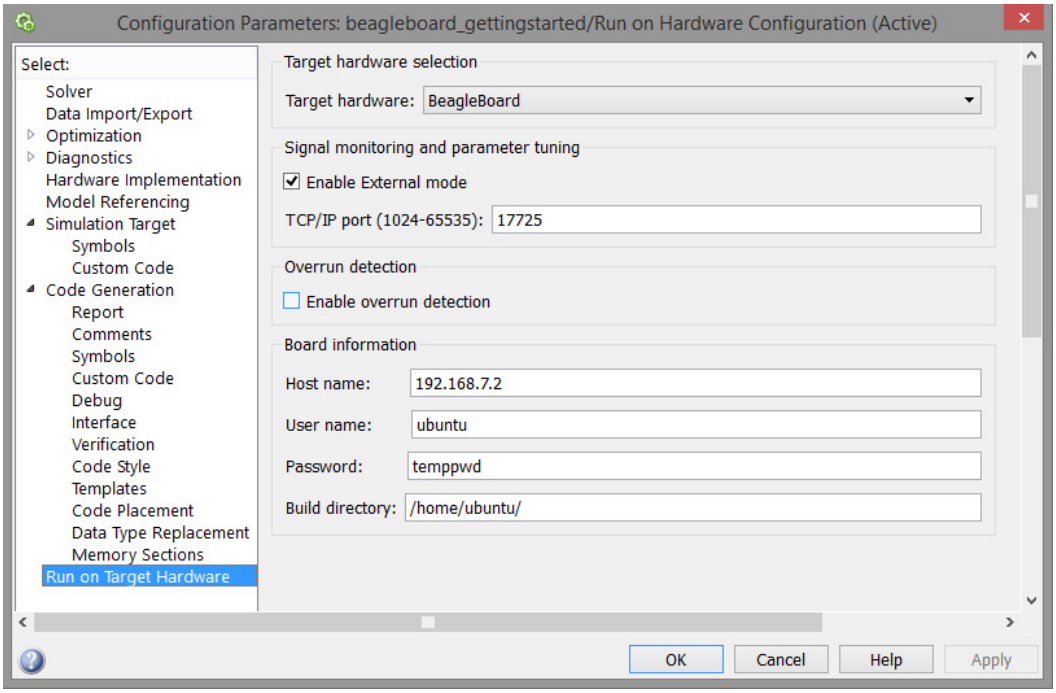


Figure 22: Simulink Environment Setup: Target Hardware Configuration

Note that these are the default settings for a BeagleBone Black with a fresh install of Ubuntu.

4) In the "Solver" configuration, set the "Stop Time" to infinity as shown below. This will allow Simulink to run the model indefinitely on the BeagleBone Black until the "Stop" button is activated.

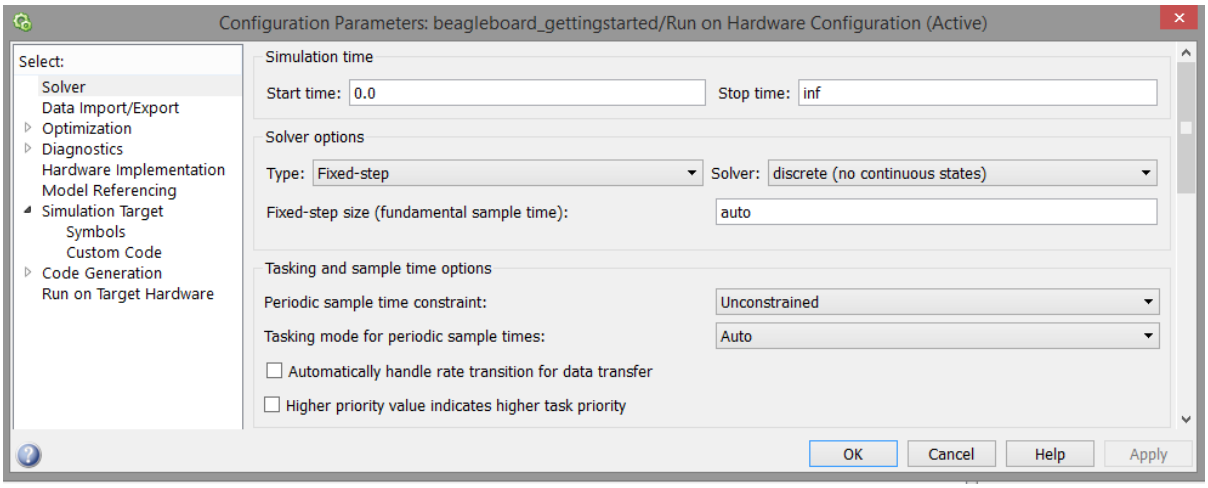


Figure 23: Simulink Environment Setup: Simulation Settings

5) Set the Simulation Mode to "*External*" from the drop down menu.

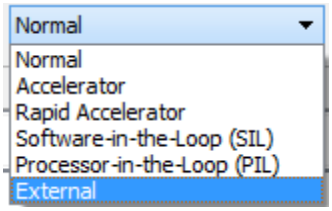


Figure 24: Simulation Mode Setting

5.4 TESTING MATLAB/SIMULINK FUNCTIONALITY WITH THE BEAGLEBONE BLACK

Simulink is now configured to run any user-built model on the BeagleBone Black. The following section will demonstrate this functionality with a simple pre-built demonstration model.

Procedure:

1) In the MATLAB command window, type "*beagleboard_gettingstarted*" and hit enter. A Simulink window will automatically open with the following model as show below.

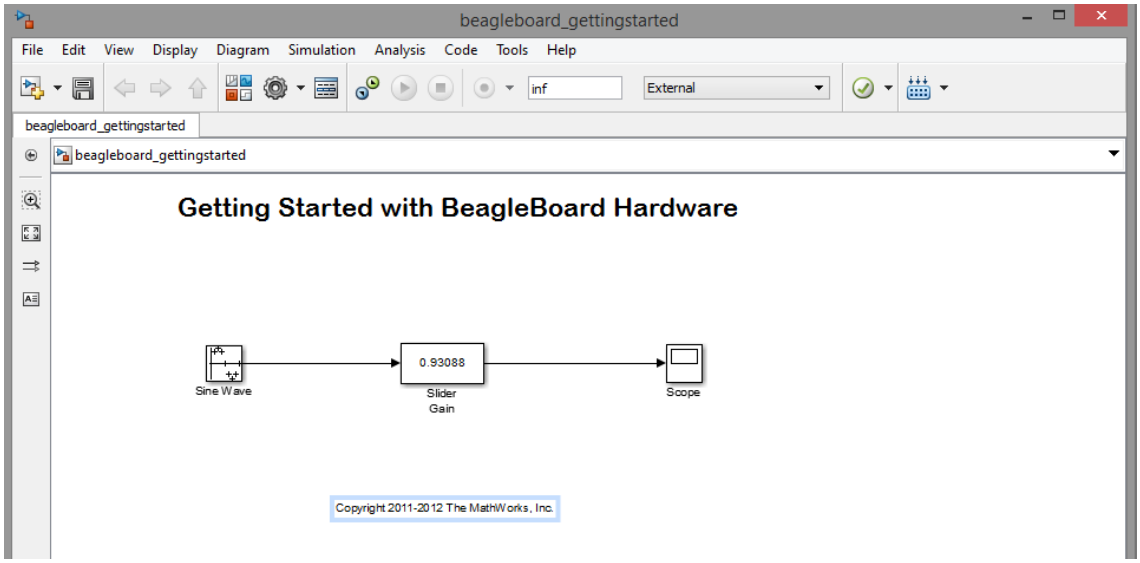


Figure 25: BeagleBoard – Getting Started Model

2) The first step in running a Simulink model on the BBB is to execute a "Build" command. This essentially launches the process by which the Simulink model is converted to C code, downloaded and built on the BeagleBone Black system. Note that this must be carried out each time a program is run in external mode. Click the "Build" button and wait a few moments while the process executes. At the end of the process, a Windows command prompt will appear as shown:

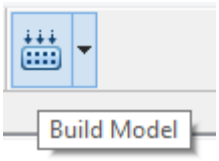


Figure 27: Build Model

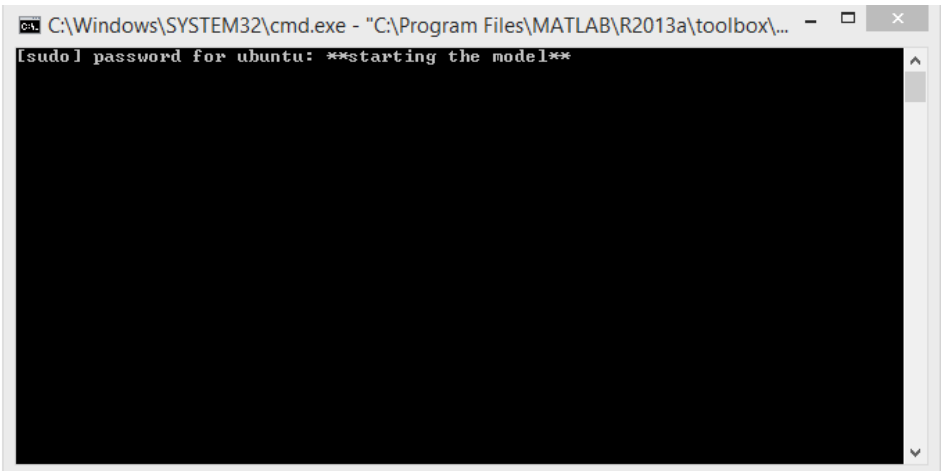


Figure 26: Windows Command Prompt: Model Start Message

3) Next, connect Simulink to the BBB via the "Connect to Target" button. This essentially opens the previously configured external port, for signal monitoring and parameter tuning. Simulink is now ready to send and receive data from the deployed model.

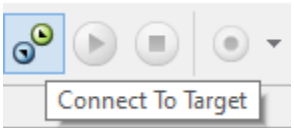


Figure 28: Connect Target

4) Starting execution of the Simulink model is carried out by the "Run" button located on the Simulink taskbar. Once Simulink has successfully connected to the target hardware in the previous step, the "Run" button will appear enabled. Click it to trigger execution of the compiled program on the BeagleBone Black.

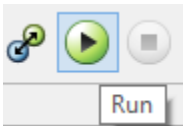


Figure 29: Run on Target

5) Once the model begins execution, the simulation time counter found on the lower right hand side of the Simulink window will begin to count up indefinitely or until the "Stop" button is activated.

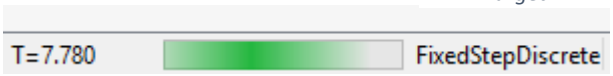


Figure 30: Runtime Counter

6) Execution of the program may be verified by viewing the generated sine wave on the scope found in the model. This highlights one of two key functionalities of "External Mode" execution: Data generated by the running program may be scoped by the user for real time analysis. External mode instructs the model running on the target hardware to send back to Simulink, any data entering a sink block such as a scope. If no sink block is used within the Simulink model, then real time monitoring via Simulink is not possible. The second key functionality is the ability to tune specific parameters during real-time execution. Should a parameter be updated via a Simulink control, such as a slider, this parameter is updated within the program running on the target hardware and effectively takes effect on the following time step. This essentially allows for Simulink to be employed as a user interface for rapid, real-time testing, analysis and tuning. The following figures demonstrate this functionality:

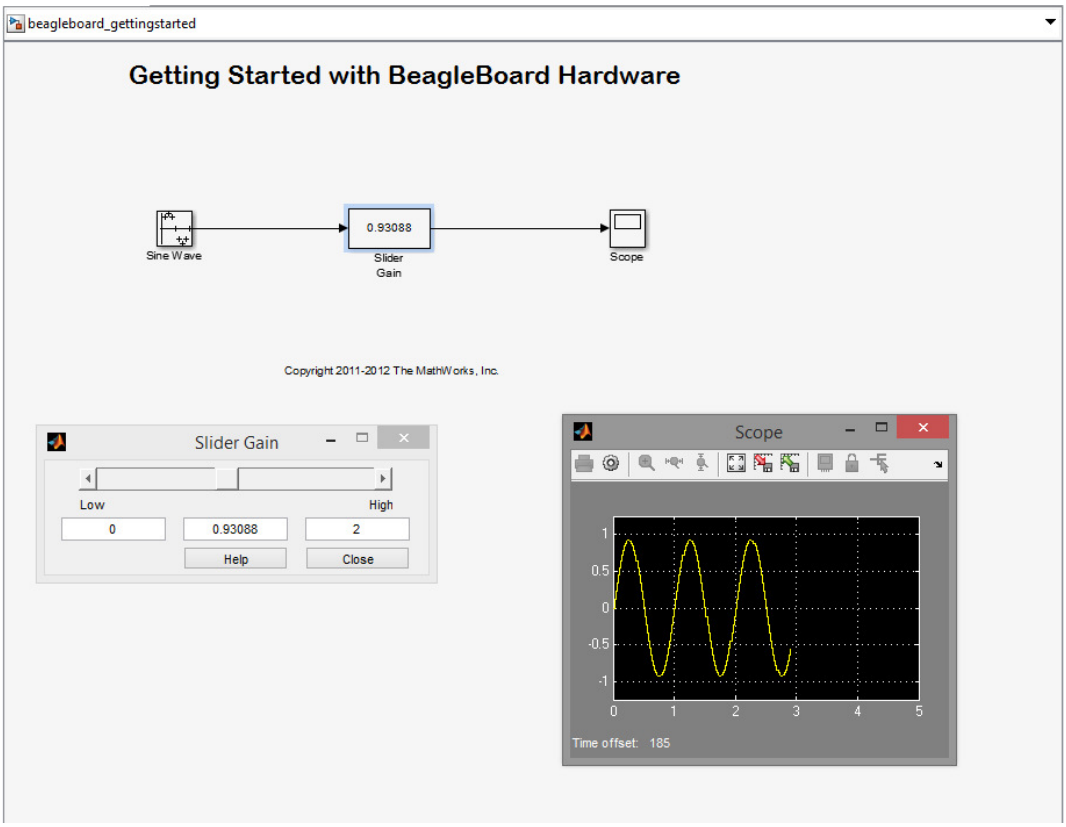


Figure 31: Simulink Parameter Tuning - Gain Setting 1

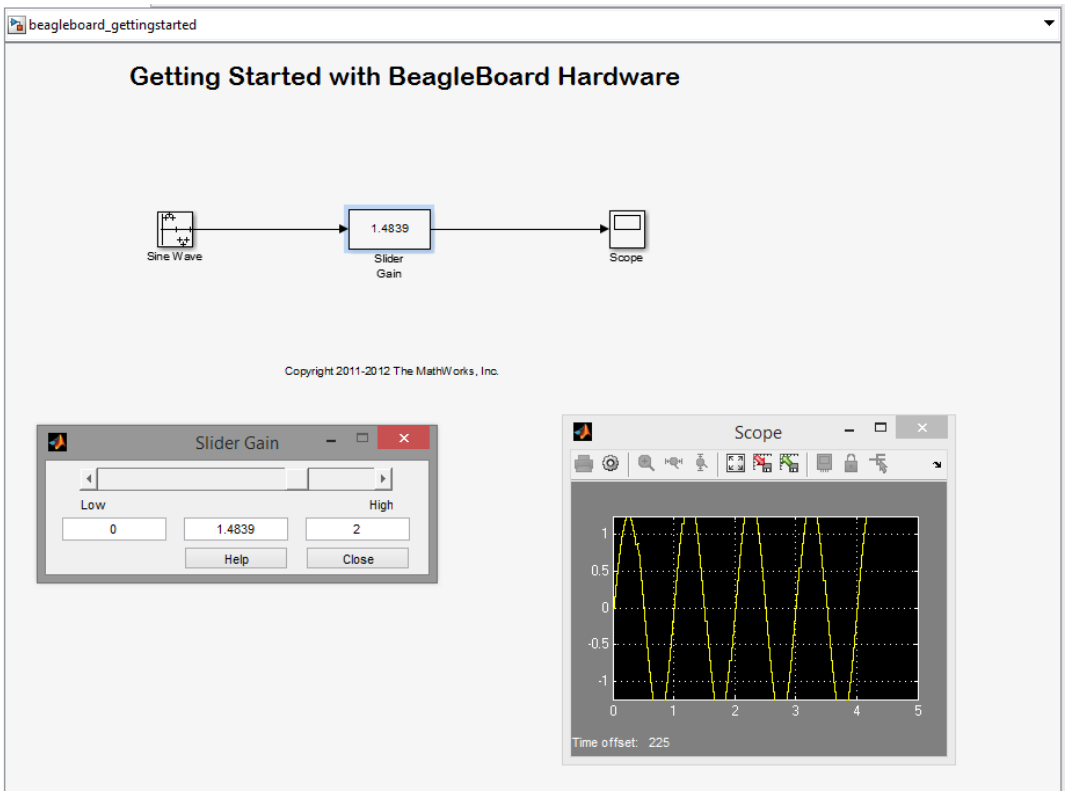


Figure 32: Simulink Parameter Tuning - Gain Setting 2

7) Finally, model execution on the target hardware may be stopped via the "Stop" button on the Simulink toolbar. The Windows command prompt will also confirm that the model has successfully exited run mode via a message similar to the one shown below:

Figure 34: Stop Simulation

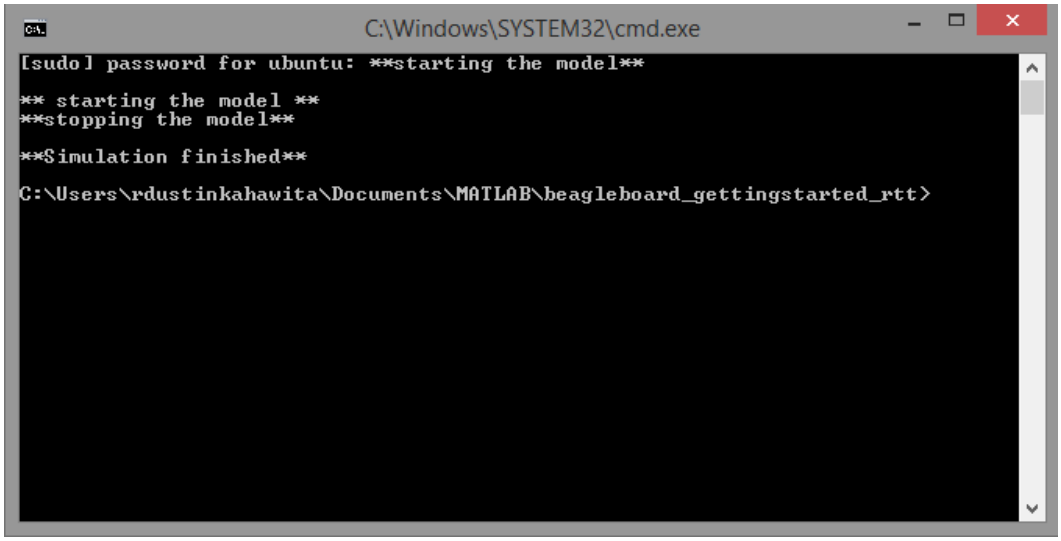


Figure 33: Model Execution Stop Message from Target Hardware

Note that running in external mode with an embedded scope in the model increases the processing burden on the target hardware. .

6.0 DEVELOPMENT OF S-FUNCTION DRIVER BLOCKS FOR DEPLOYMENT TO THE BEAGLEBONE BLACK

6.1 INTRODUCTION TO THE S-FUNCTION BUILDER BLOCK

Development of the BeagleBone Black specific driver blocks for Simulink allows for live interaction between the target hardware and a Simulink model. This is to say that a Simulink model gains the ability to interact with the physical world. An extension of this is to control the behaviour of a system based on sensed inputs from the surrounding environment. These driver blocks harness the BeagleBone's ability to read digital or analog data from sensors as well as control physical objects through analog and digital outputs or pulse width modulation (PWM). Therefore sensing (input) drivers and actuating (output) drivers may be combined with a well-designed control algorithm to rapidly prototype autonomous control of a user defined system.

The basis for developing driver blocks used in this project is the S-Function Builder found in the Simulink Library Browser under *Libraries → User-Defined Functions → S-Function Builder.* Essentially, the S-Function builder allows the user to create a Matlab executable file (MEX) from source code written in C. Typically, this means that a user can create blocks to be run in simulation mode using pre-existing C functions and libraries which are readily available and somewhat more prevalent than existing Matlab solutions.

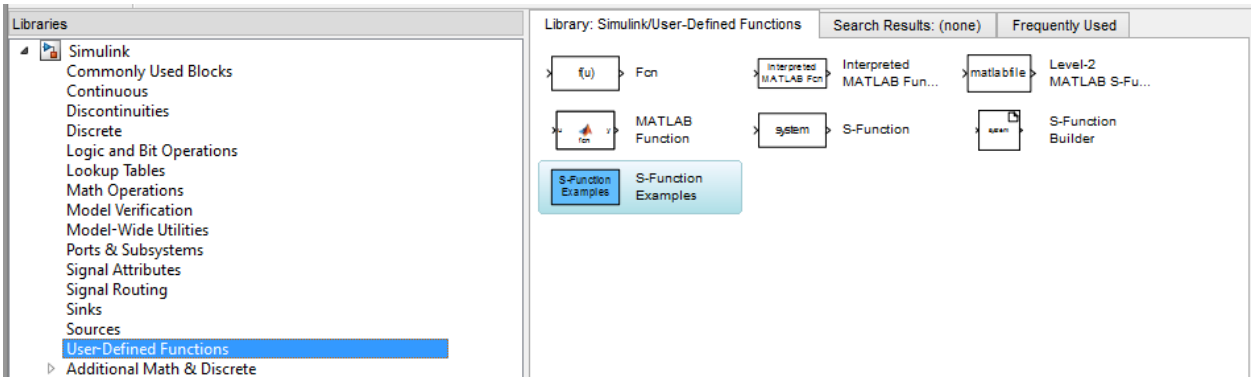


Figure 35: S-Function Builder

In this application, the C-code wrapped by the S-Function Builder is used to interact with the BeagleBone Black's Linux based OS via its virtual file system, commonly referred to as "sysfs". This file system exposes the various drivers present on the BeagleBone Black allowing for read/write access to its GPIO ports or controlling the on-board LEDs. Actual device control is carried out by altering a given set of configuration variables. It should be noted, that code developed for the driver blocks *do not* execute on the host computer in simulation mode. They are, in fact, executed only on the specified target hardware when Simulink is placed into "External Mode".

6.2 S-FUNCTION BUILDER CONFIGURATION

With the S-Function Builder block inserted into a new Simulink model, one must simply double click the block to open the configuration box. It is from this configuration dialog box that all attributes relating to the block must be specified and configured.

Note that the following explanation will use the "USRLED" driver as an example. The USRLED block was developed during the course of this project to allow full control to the BeagleBone Black's four blue user LEDs. These surface mount LEDs are used by the OS to indicate system heart beat as well as memory and CPU usage. The USRLED driver

bock allows the user to toggle the LEDs through a Boolean input and hence may be used for diagnostics or debugging purposes while the Simulink model is executing on the BeagleBone Black.

S-Function Parameters Panel

The parameters panel features the general configuration of the S-Function block. Unsurprisingly, this includes the name to be given to the S-Function and will serve as the filename root for all files generated during the build process.

The S-Function parameter's sub-panel displays the name, data type and current value of the parameter to be passed into the block from the user. These parameters are configured in the "Data Properties" tab of the "Port and Parameter Properties" panel which will be discussed in the following section.

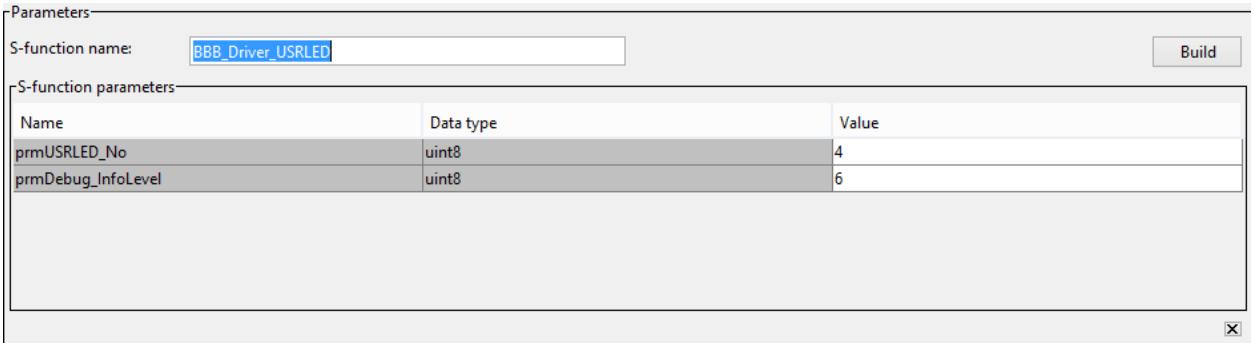


Figure 36: S-Function Builder: Parameters Pane

S-Function Data Properties Tab

The Data Properties tab serves as the centralized location for configuring all input, output and parameter variables processed within the S-Function.

The **Input Port and Output Port** tabs allows for the definition of the expected data entering and exiting the block. Here, ports may be added or deleted and renamed to reflect their functionality. These names will then appear next to the ports when the block is added to a Simulink model. Note that up to 2-dimenstional signal inputs and outputs are supported. Should a two dimensional input be used the size of the data stream may be configured via the "Rows" and "Columns" parameters. The input and output ports may also be configured to accept real or complex values under the "Complexity" column. Finally if the signal inputs or outputs to a bus, switch the "Bus" parameter to "On" and set the bus name. Parameter blocks developed for this project exclusively use one dimensional real signals and do not employ the use of buses. The input and output ports are for the "USRLED" driver block are configured as show in the following screenshots:

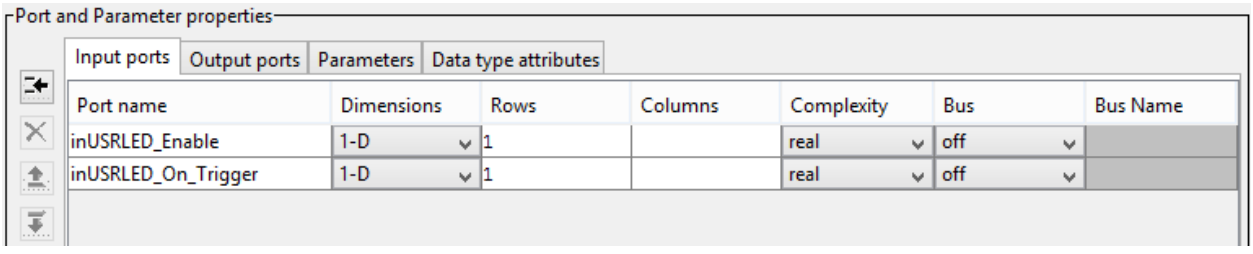


Figure 37: USRLED Input Port Tab

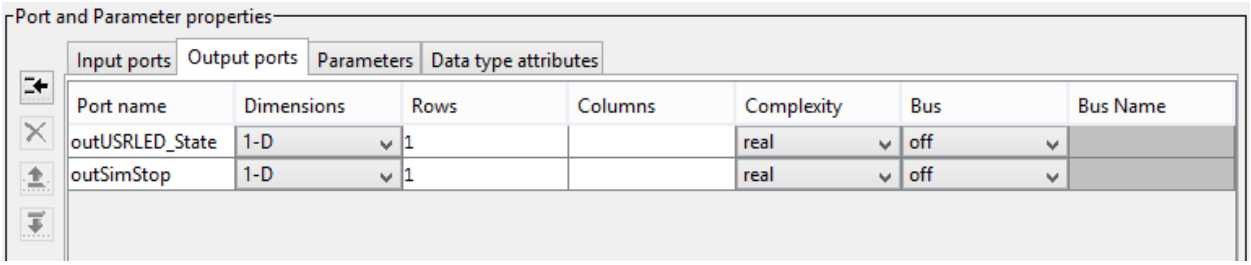


Figure 38: USRLED Output Port Tab

The **Parameters** tab allows for addition or removal of block specific parameter variables. These variables grants the block designer the ability to add a level of user configurability to the S-Function. Typically, the values within these variables may be set by the block user while the model is offline. However, with proper masking, these values may be set to allow modifications while the simulation is running. This functionality will be further elaborated on in the section regarding masking. The data type of the parameter must also be selected in order to reflect the size of the expected input value. For the case of the USRLED driver block, parameters were added to allow the user to select which of the four blue LEDS to control as well as the amount of debug information printed to the Windows command prompt. Note that the actual value of the parameter is set in the "Parameters Pane" shown in figure (36) above.

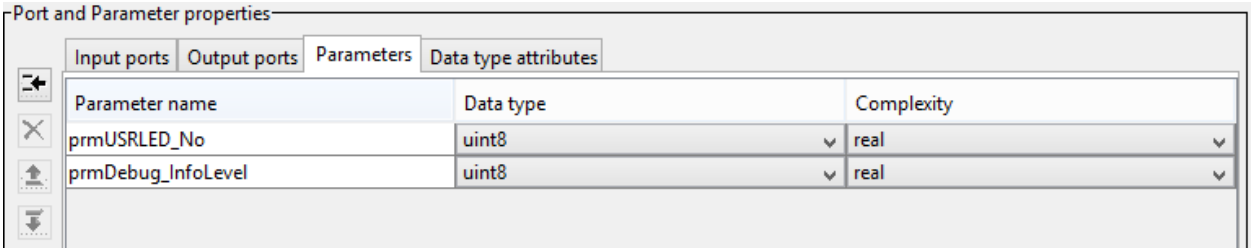


Figure 39: USRLED Parameters Tab

The **Data Type Attributes** tab is used to configure the expected data type, via the "Data Type" dropdown menu, for each input and output port already been added to the S-function. The columns indicating "Word Length", "Signed", "Fraction Length", Slope" and "Bias" are only enabled for fixed-point data types and are therefore disabled in the USRLED example. Note that input and output ports cannot be added or re-ordered from this pane

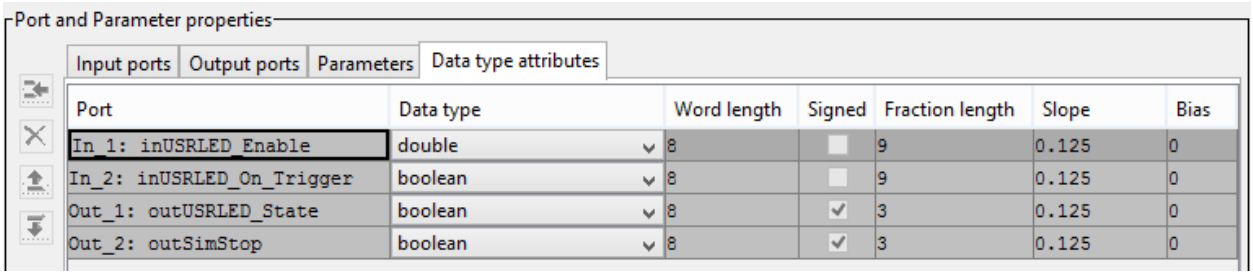


Figure 40: USRLED Input/Output Data Types

With the input and output ports configured and parameter variables set, the design of the block's overall interface shell is now complete.

S-Function Libraries Tab

The libraries tab handles the referencing and linking of external source files located outside the Matlab or project directory as well as the declaration of external functions and include files. External variables may also be declared here.

In the **Library/Object/Source** field, the block designer must declare the location path of any external functions or source code that is referenced from code written in the "Outputs", "Discrete Update" or "Continuous Derivatives" tabs. Note that the format of this declaration will depend on where the file is actually located and only one statement may be issued per line.

The **Include** field is used to list the various header files referenced from the C-code within the S-Functions. This is synonymous with programs written in C, and as such the declaration of header files is carried out via the "#include" statement. Again, only one statement should be issued per line and the syntax used will depend on if a standard (#include <stdio.h>) or custom (#include "BBBFuncs.h") header is used. Also note that if the custom header is not located in the current Matlab directory then a path reference must be added to the Library/Object/Source field.

The **External Function** declaration field lets custom external functions be written within the S-Function block and provides the designer with an alternative to having to declare functions within header files. Any functions defined within this plane may be called from the "Outputs", "Discrete Update" or "Continuous Derivatives" fields.

As mentioned earlier, caution must be taken when declaring external functions and variables in S-Function blocks which may have several instances present in a single Simulink model. In such a situation, it is likely that multiple instances of the same variable is not recognized and therefore is updated from the different S-Function blocks present in the same Simulink model. Hence, if multiple S-Function block instances are expected, declaration of external variables should be avoided and instead declared locally within the "Outputs", "Discrete Update" or "Continuous Derivatives" fields.

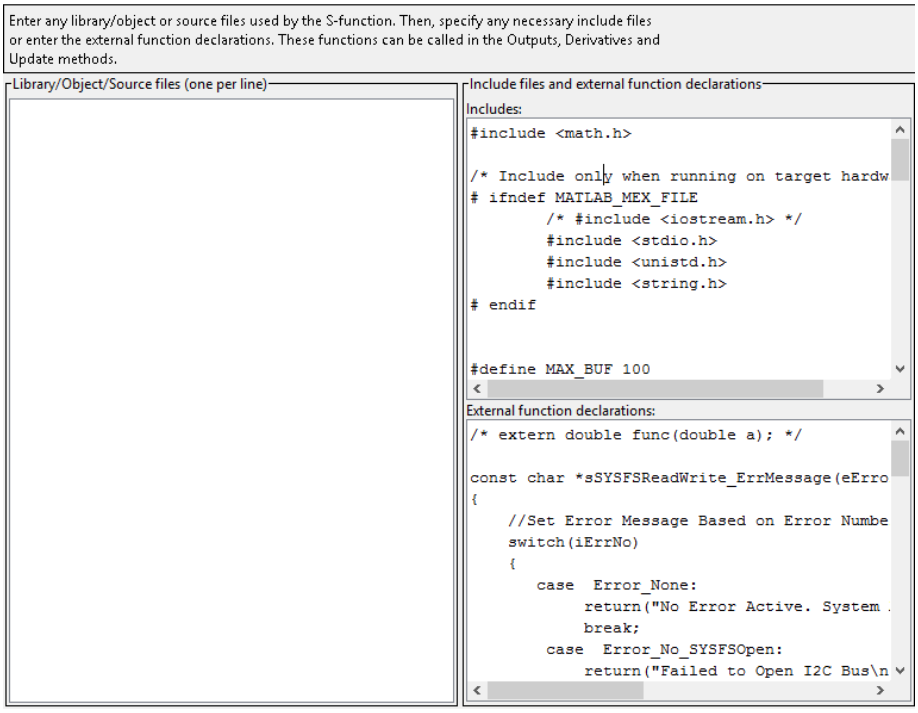


Figure 41: USRLED Libraries Tab

S-Function Outputs Tab

The **Code Description** pane of the *Outputs* tab contains the main source C-code to be executed by S-Function block at every time step during the simulation (assuming a continuous S-Function). C-Code is typed directly into this pane by the block's designer.

The source code entered here has read/write access to the output variables configured via the *Output Ports* pane in the *Data Properties* tab. It also has read access to the data variables configured via the *Input Ports* pane in the *Data Properties* tab. In order to access input data the option box at the bottom of the Code Description stating "Inputs are needed in the output function (direct feed through)" pane must be checked. The syntax used for accessing input, output, discrete and parameter variables are as follows:

- Input Data: Input_Variable_Name[0]
  - Examples:
    - inUSRLED_On_Trigger[0]
    - inUSRLED_Enable[0]
- Output Data: Output_Variable_Name[0]
  - Examples:
    - outUSRLED_State[0]
    - outSimStop[0]
- Discrete Variables : (Note: no variable names can be assigned)
  - xD[0], xD[1], xD[2]….xD[n]
- Parameter Variables: Parameter_Variable_Name[0]
  - Examples:
    - prmUSRLED_No[0]
    - prmDebug_InfoLevel[0]

Note that local variables declared within the *Code Description* pane as well as external variables which may have been declared in the *Include* pane of the *Libraries* tab are not stored from one time step to another. Should this functionality be required, a discrete variable may be declared in the *Discrete Update* tab. The output source code has the capability to read discrete variables however their values may only be modified by code entered into the *Code Description* panel of the *Discrete Update* tab.

As all data entered into the *Outputs* code plane are placed inside an *Outputs* function in the wrapper file, variables declared within the function are considered local and hence are not accessible externally. As noted before, variables declared in the *Includes* pane in the *Library* tab will be globally accessible from any code within the S-Function as they are declared outside of all functions in the generated wrapper file.

All driver blocks developed for the purpose of external execution on the Beaglebone Black contain the following lines of code:

```
/* Run only on target - BeagleBoneBlack */
# ifndef MATLAB_MEX_FILE
    /* Output code */
#else
#endif
```

As previously mentioned, during the build process, a Matlab executable (MEX) file is generated in order to execute the Simulink model. The above code ensures that code found within in the "ifndef" statement is only executed on

the target hardware and not on the host system running Simulink. This conditional structure may also be used in the "Libraries" and "Discrete Update" tabs.

Code description

Enter your C-code or call your algorithm. If available, discrete and continuous states should be referenced as, xD[0]...xD[n], xC[0]...xC[n] respectively. Input ports, output ports and parameters should be referenced using the symbols specified in the Data Properties. These references appear directly in the generated S-function.

```
/* Run only on target - BeagleBoneBlack */
# ifndef MATLAB_MEX_FILE

static FILE *iFILE_BBB_USRLED_Handle = NULL;
const char *sSYSFS_USRLED="/sys/class/leds/beaglebone:green:usr";
char sSYSFS_USRLED_Brightness[MAX_BUF]="";
char sSYSFS_USRLED_Trigger[MAX_BUF]="";
char sSYSFS_USRLED_Path[MAX_BUF]="";
int iSYSFS_Write_Result=0;
int iSYSFS_Trigger_Result=0;

// Initialize USR LED State
eUSRLED_STATE iUSRLED_State;

// Initialize Error
eError_No iError_No = Error_None;
// Initialize Error Level
eError_Level iError_Level = Error_Level_OK;
// Initialize Debug Output Level
eDebugLevel iDebug_Level = Debug_None;


/*****************************************/
// Function: Remove Standard Trigger
/*****************************************/
int iBBB_USRLED_TriggerOff()
```

☑ Inputs are needed in the output function(direct feedthrough)

Figure 42: USRLED Outputs Code

S-Function Discrete Update Tab

The purpose of the *Discrete Update* tab is to refresh the values of the discrete state vector via a user defined function. Therefore data variables stored in the state vector propagate from one time-step to the next. This provides a fairly useful functionality for applications such as counters or initialization / first scan flags.

As is the case for the *Outputs* code plane, all code inside the *Discrete Update* code plane will be inside an *Update* function within the S-Function wrapper file. As such, no variables declared within the Discrete Update code plane will be accessible from outside the function. As noted before, variables declared in the *Includes* pane in the *Library* tab will be globally accessible from any code within the S-Function as they are declared outside of all functions in the wrapper file.

Write access to the discrete state vector is only granted to code within the *Discrete Update* code plane. Read access is to these state variables however is provided to all other code segments and hence may be used as an initialization flag in the *Outputs* code.

The actual discrete variables are declared and configured in the *Initialization* tab as will be discussed in the following section.

Code description

This section is optional and use to update the discrete states. It is called only if the S-function has one or more discrete states. The states of the S-function are of type double and must be referenced as xD[0], xD[1], etc. respectively. Input ports, output ports and parameters should be referenced using the symbols specified in the Data Properties. These references appear directly in the generated S-function.

```
# ifndef MATLAB_MEX_FILE

// Notes On Discrete States
// xD[0]: First Scan
// xD[1]: Output State Memorization
// Initialize Debug Output Level

eDebugLevel iDebug_Level = Debug_None;

// Check Debug Level - Read Parameter Data -Debug Level
iDebug_Level = iSet_Debug_Level(prmDebug_InfoLevel[0]);

// Record Output State
xD[1]=outUSRLED_State[0];

if ((xD[0]!=1) && (inUSRLED_Enable[0]==true))
{
    // Set First Scan Bit
    xD[0]=1;
    if ((iDebug_Level >=Debug_Level_0)){printf("USRLED Msg: First Scan Bit Set\n");}
    // Print Gap
    if ((iDebug_Level >=Debug_Level_0)){printf("\n");}

}
else if ((xD[0]==1) && (inUSRLED_Enable[0]==false))
{
    // Reset First Scan Bit
```

Figure 43: USRLED Discrete Update Code

S – Function Initialization Tab

The Initialization tab serves to configure the sample mode as well as the discrete and continuous states. Given that the code is running on external hardware, the driver block executes in discrete time. Therefore no continuous states are to be configured.

In the **S-Function Settings** plane, the number of discrete states is set. Note that if there are zero discrete states then the code entered in the *Discrete Update* plane is not called and therefor does not execute. The initial condition values for each discrete state must be configured via the **Discrete States IC** field. Note that the values entered must be separated by a comma. The S-function sampling mode may be set by the **Sample Mode** dropdown menu. Continuous, discrete or inherited modes may be selected. If the sampling mode is set to discrete, then a sampling time must be stipulated. For this reason, "Inherited" sampling mode was selected for all driver blocks. This allows the sampling time to be set centrally in the Simulink model's simulation parameters as opposed to individually on all driver blocks.

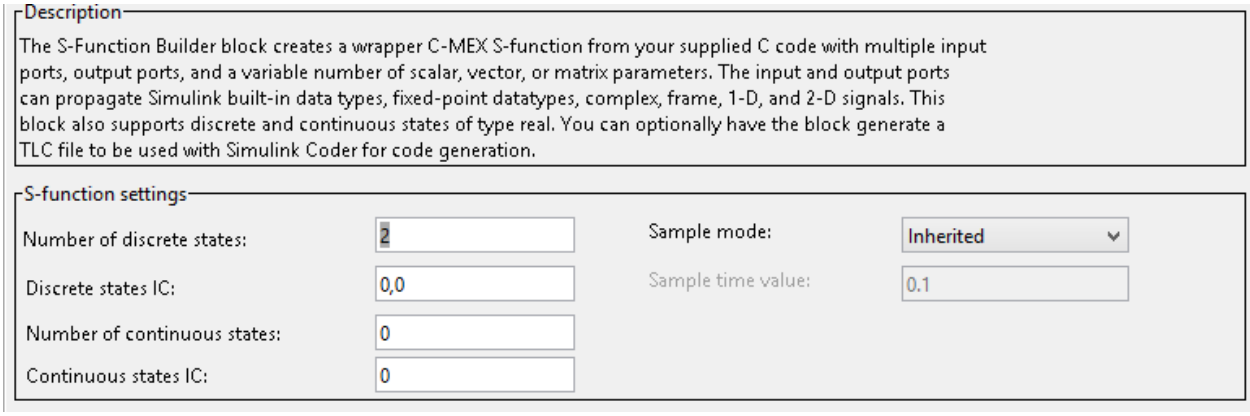


Figure 44: USRLED Initialization Tab

S – Function Build Info Tab

Once the S-Function is fully configured it must be complied and placed into a wrapper file. This task, among others, is carried out in the build process and may be monitored from the Build Info tab's *Compilation Diagnostic*s pane. The overall state of the compilation as well as any syntax or reference errors detected during compilation will be indicated here. Note then that this pane may be used as a relatively simple debugging tool. Should the S-function compile correctly, the messages received will resemble those shown in following screenshot. Note that the "Generate wrapper TLC" option must be selected. This ensures a TLC file, used to build the target executable file, is generated. A more detailed look into the build process will be presented in the following section.

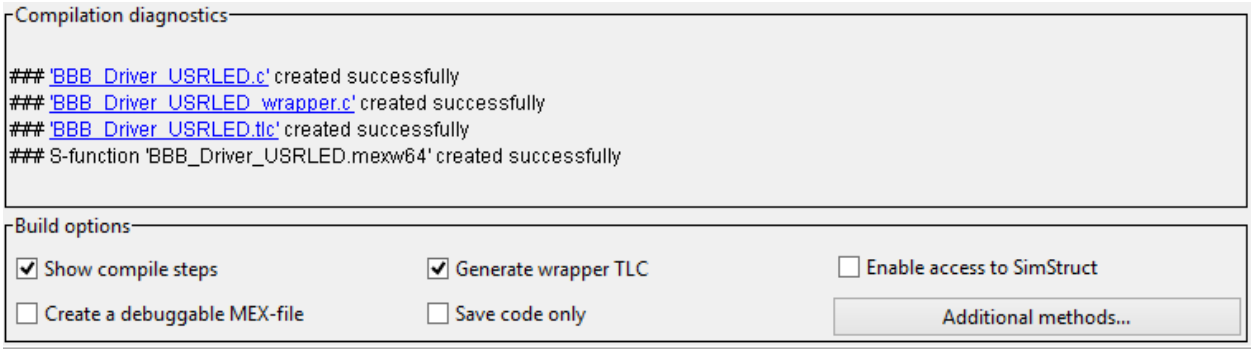


Figure 45: USRLED Compilation Diagnostics

6.3 S–FUNCTION BUILD PROCESS & GENERATED WRAPPER FILE

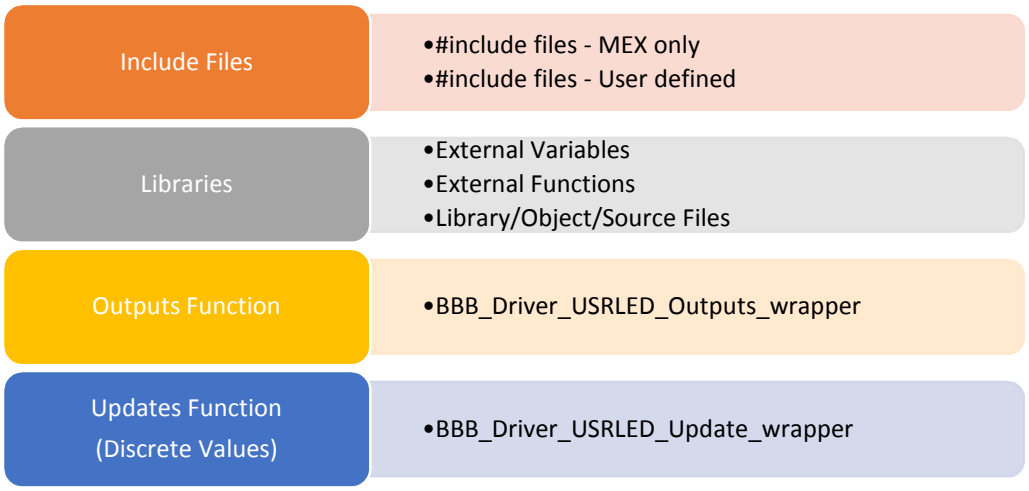
The build process launched from the S-Function Builder dialog box generates a set of source files for interfacing with Simulink and the target hardware. These files include:

- **BBB_Driver_USRLED.c**: Contains C source code for external declarations of the wrapped Outputs and Update functions found in the BBB_Driver_USRLED_wrapper.c file. Standard S-Function methods are included to validate parameter inputs and initialize the size and value of the input, output and discrete data variables. As such the file acts primarily as an S-Function C file to interface with Simulink in an on-host system simulation-only capacity.
- **BBB_Driver_USRLED.tlc**: This Target Language Complier (TLC) file allows Matlab's Real Time Workshop (RTW) to include the S-Function block code in the generated C-Code for the Simulink model. The RTW, now referred to as Simulink Coder, is the primary engine for generating and executing C and C++ code from

Simulink models. Essentially, the TLC file acts as an interface allowing the RTW to grab the S-Function block code and include it in the overall C code generated for the entire Simulink model. This overall C code will then be compiled to run on the target hardware.

- **BBB_Driver_USRLED.mex64**: A simulation-only Matlab Executable file generated from the C code entered into the S-Function builder dialog box.
- **BBB_Driver_USRLED_wrapper.c**: This file contains all the custom code created within the S-Function builder dialog box. The code written into the *Discrete Update*, *Outputs* and *Continuous Derivatives* panes are wrapped into individual functions and placed into this wrapper file. These functions are then referenced from the TLC file during on-hardware execution and the MEX filed during on-host simulation. Any data entered into the *Libraries* pane is found outside the wrapped functions and therefore may be called from any one of them.

The generated wrapper file is of most interest given that it contains the source code for the newly configured S-Function and hence is referenced by both the simulation only BBB_Driver_USRLED.c and external target BBB_Driver_USRLED.tlc file. As its structure also dictates accessibility and general functionality, an overview is presented below:



The following screen shots show a truncated version of the USRLED code. Placeholders indicate the locations for the declaration of external variables and functions as well as source code for the Outputs and Updates functions:

```
/*
 * Include Files
 */
#if defined(MATLAB_MEX_FILE)
#include "tmwtypes.h"
#include "simstruc_types.h"
#else
#include "rtwtypes.h"
#endif

/* %%%-SFUNWIZ_wrapper_includes_Changes_BEGIN --- EDIT HERE TO _END */
#include <math.h>

/* Include only when running on target hardware */
# ifndef MATLAB_MEX_FILE
    #include <stdio.h>
    #include <unistd.h>
    #include <string.h>
# endif
```

```
// Define USRLED External Variales
// Define USRLED External Functions

/* %%%-SFUNWIZ_wrapper_includes_Changes_END --- EDIT HERE TO _BEGIN */
#define u_width 1
#define y_width 1
/*
 * Create external references here.
 *
 */
/* %%%-SFUNWIZ_wrapper_externs_Changes_BEGIN --- EDIT HERE TO _END */
/* extern double func(double a); */
```

Figure 46: USRLED Include File and External Declarations

```
/* %%%-SFUNWIZ_wrapper_externs_Changes_END --- EDIT HERE TO _BEGIN */
/*
 * Output functions
 */
void BBB_Driver_USRLED_Outputs_wrapper(const real_T *inUSRLED_Enable,
            const boolean_T *inUSRLED_On_Trigger,
            boolean_T *outUSRLED_State,
            boolean_T *outSimStop ,
            const real_T  *xD,
            const uint8_T  *prmUSRLED_No, const int_T  p_width0,
            const uint8_T  *prmDebug_InfoLevel, const int_T p_width1)
{
/* %%%-SFUNWIZ_wrapper_Outputs_Changes_BEGIN --- EDIT HERE TO _END */
    /* Run only on target - BeagleBoneBlack */
    # ifndef MATLAB_MEX_FILE
        // USRLED Ouputs Function Source Code
    # endif
/* %%%-SFUNWIZ_wrapper_Outputs_Changes_END --- EDIT HERE TO _BEGIN */
}
```

```
/*
 * Updates function
 */
void BBB_Driver_USRLED_Update_wrapper(const real_T *inUSRLED_Enable,
            const boolean_T *inUSRLED_On_Trigger,
            const boolean_T *outUSRLED_State,
            const boolean_T *outSimStop ,
            real_T *xD,
            const uint8_T  *prmUSRLED_No,  const int_T  p_width0,
            const uint8_T *prmDebug_InfoLevel, const int_T  p_width1)
{
  /* %%%-SFUNWIZ_wrapper_Update_Changes_BEGIN --- EDIT HERE TO _END */
# ifndef MATLAB_MEX_FILE

    // USRLED Update Function Source Code

#endif
}
```

Figure 47: USRLED Outputs and Update Function Wrapper

6.4 S-FUNCTION MASKING

Masking grants the S-Block designer the ability to create a custom user interface for the S-Function. Masking hides the source code and provides a user experience similar to a standard Simulink block. The block itself may also be configured to have its own icon as well as parameter dialog box. This not only protects the source code from unwanted modifications but also eases proliferation and deployment to multiple users.

Create New Instance of S-Function

With the original S-Function Builder successfully built and tested, drag and drop an *S-Function block* into the existing Simulink model or a new Simulink Model.

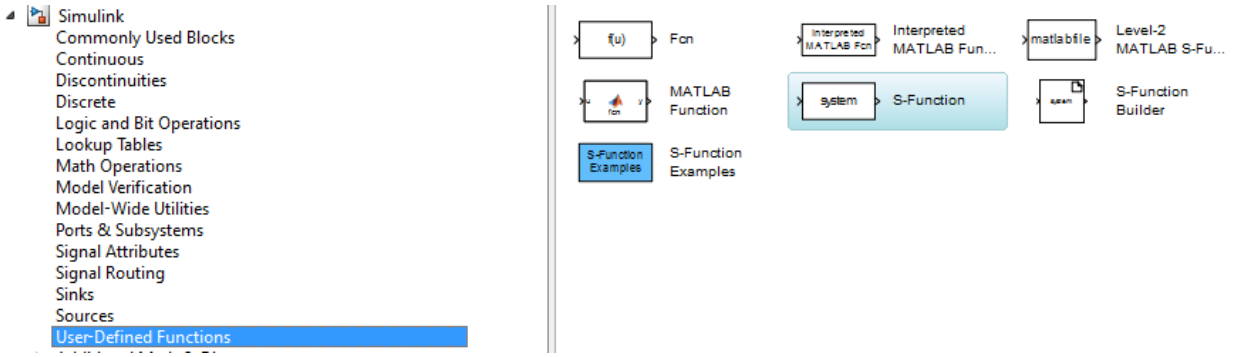


Figure 48: S-Function Block – Instance of S-Function

Next, create a new instance of the newly developed S-function by double clicking on the S-Function block. The following dialog box will appear. Complete the fields in the manner shown. Pay special attention to how the input parameters are formatted.

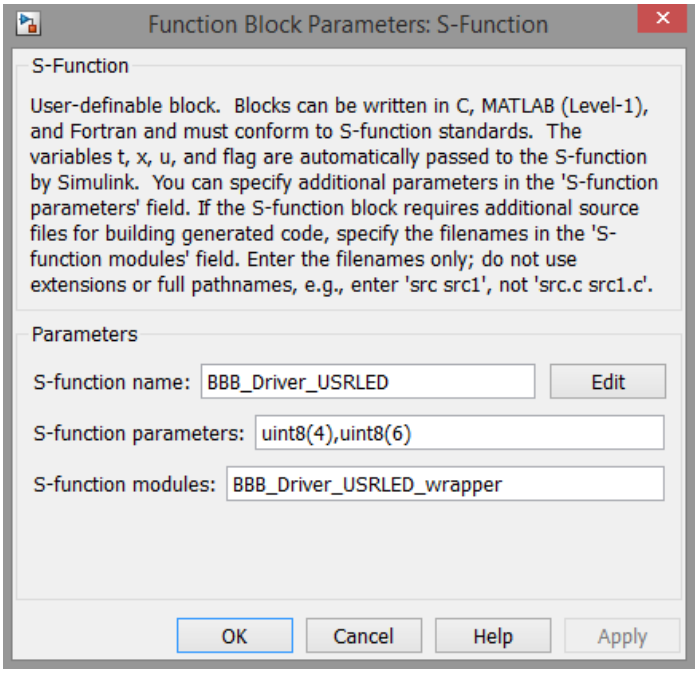


Figure 49: S-Function – Linking to Source Wrapper Code

Click on “Apply” and a fresh instance of the original S-Function should appear with the correct number of input and output ports while sporting the S-Function name:

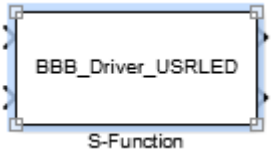


Figure 50: Linked S-Function Block

Launch the Mask editor by right clicking on the S-Function and selecting “Mask” and then “Create Mask”

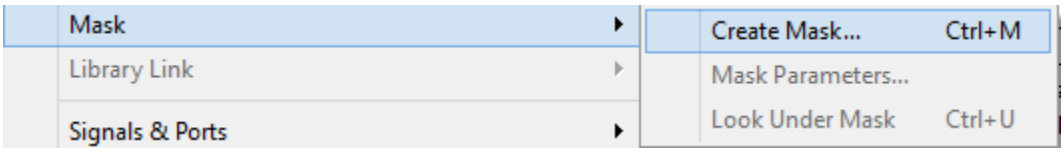


Figure 51: Create Mask

S-Function Masking – Icon & Ports Tab

The Icon & Ports tab is used to select and configure the block’s icon image as well as to label the input and output ports. The code displayed in the following figure links an image to the block’s face and defines green input/output port text labels. Finally the “BLACKlink” text in orange is added to the face at coordinate position (0, 6) in pixel units.

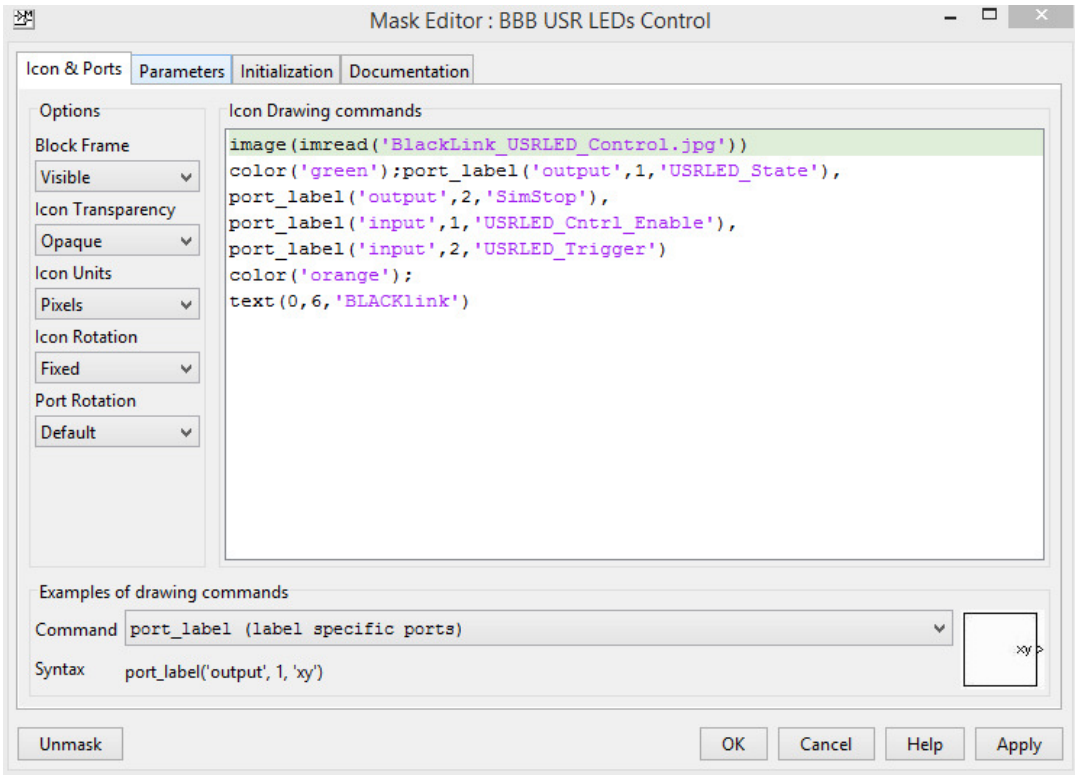


Figure 52: Mask Editor – Icons and Ports Tab

S-Function Masking – Parameters Tab

The parameters tab is used to develop the block configuration dialog box for the masked S-Function. The parameters created here should match the number and type of parameters existing in the original S-Function.

In the *Prompt* column, enter the text to be used to provide instructions regarding the nature of the parameter and how it should be configured.

Next, under *Variable*, set the variable name used to store the parameter data. Note that the original parameters will point directly to this variable name and will therefore evaluate its exact contents.

Under *Type* set the control that will be used to enter the parameter data (e.g. Drop-down menu, checkbox, direct edit field etc.)

The *Evaluate* checkbox forces Simulink to directly evaluate the value entered into the filed as is. By default this box should always be selected.

The *Tunable* checkbox, if selected, allows for the parameter to be adjusted while the simulation is running. If not selected, the user will only be able to adjust the parameter while the simulation is off line.

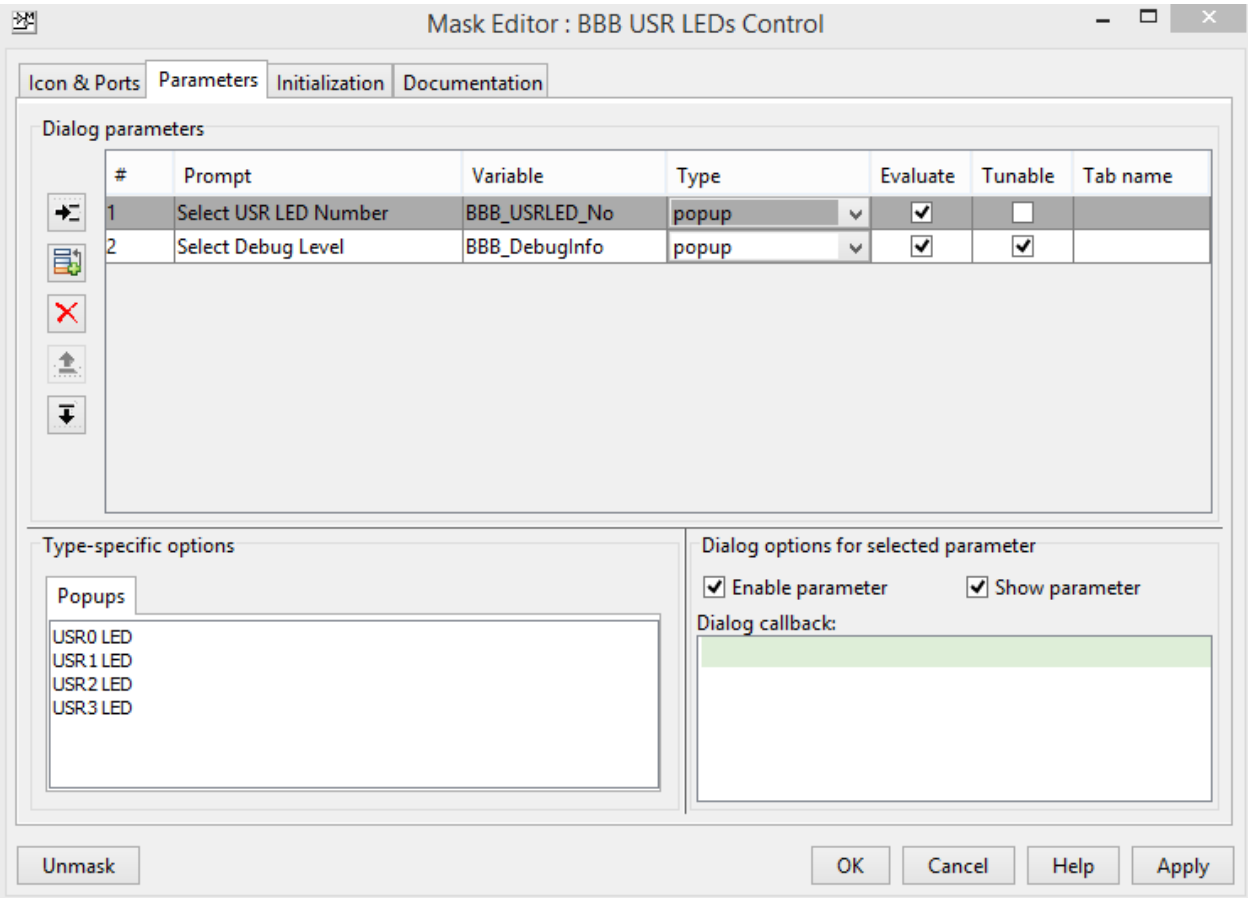


Figure 53: Mask Editor – Parameters Tab

S-Function Masking – Initialization Tab

The initialization tab allows for Matlab commands to be entered in order to initialize the masked S-Function block upon opening of its parent Simulink block. This option was not used during the course of this project.

S-Function Masking – Documentation Tab

The documentation tab, as the name suggests, allows the user to add a description to the masked model's parameter configuration dialog box. The **Mask Description** pane may be used to provide information regarding the general functionality of the block as it will appear at the top of the configuration dialog box. The **Mask Help** may be used to provide troubleshooting information, further description of the parameters or programmer contact/program version information.

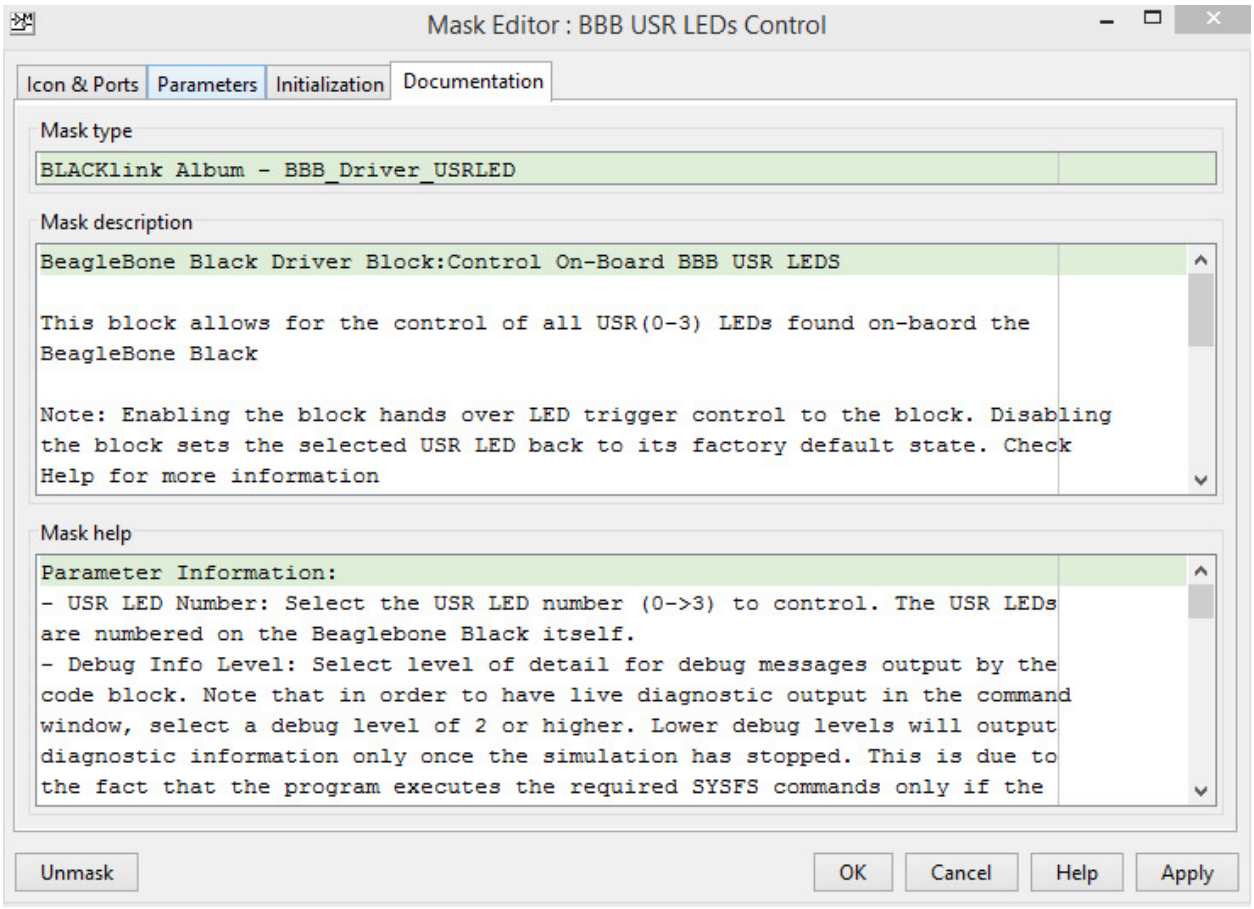


Figure 54: Mask Editor – Documentation Tab

S-Function Masking – Swap Parameters

With the mask completed and applied to the S-Function, right click on the block, select "Mask" and then "Look Under Mask":

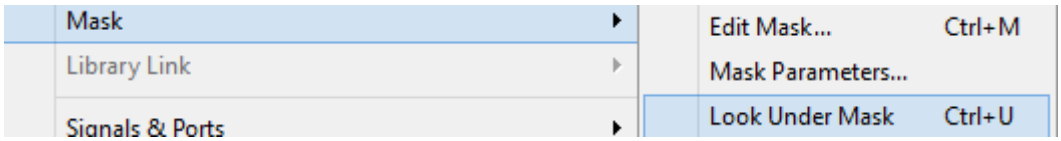


Figure 55: Masks – Look Under Mask Menu

Ensure that the parameters are now configured such that they point to the parameter variables assigned in the *Parameters* tab of the Mask Editor as shown in figure (53).

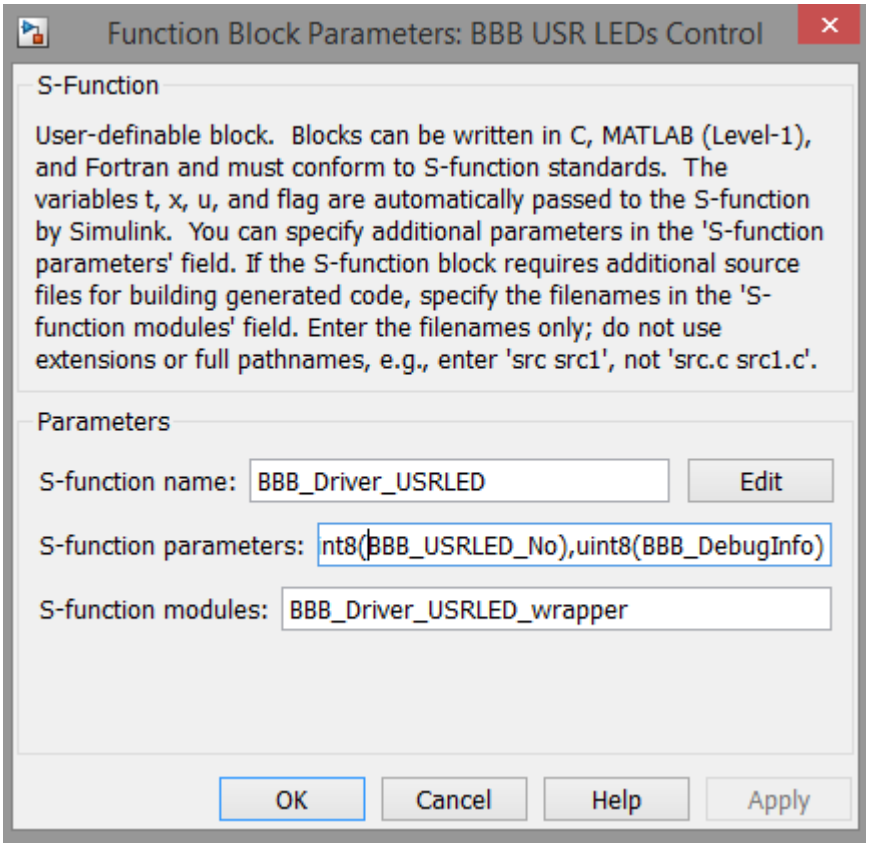


Figure 56: S-Function Parameter Linking

The mask is complete and the S-Function block is now ready to be replicated and deployed. Note that all instances still point to the BBB_Driver_USRLED_wrapper.c file, therefore any changes made to the source via the original S-Function Builder block will propagate down to the masked instances.

The final product of the masking process for the USRLED block is shown in section (7.8) below.

7.0 THE BLACKLINK LIBRARY

7.1 OVERVIEW OF THE BLACKLINK LIBRARY

The lack of Matlab/Simulink support for the BeagleBone Black at the outset of this project, it seemed an honorable quest to develop standardized function blocks for use by the open source BeagleBone community. Hence the creation of the "BLACKLink" Simulink library as pictured below. At the time of writing, a total of nine blocks have been developed consisting of six standard blocks and three component specific driver blocks. The standard blocks include functionality for GPIO read/write, I2C read/write, analog input read and USRLED control. Specific driver blocks were written for the ADXL330 Accelerometer, the L3G4200D Gyroscope and HMC5883L Magnetometer. Each block will be elaborated on in some detail in the following sections. Future additions to the BLACKLink library will include blocks for motor control via pulse width modulation (PWM) as well as for SPI communication.

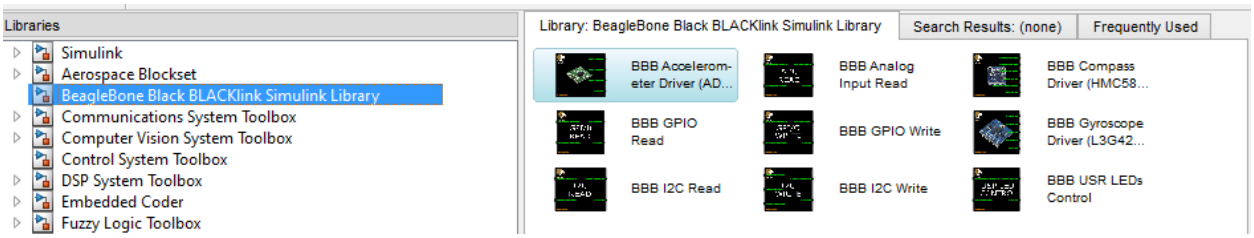


Figure 57: The BLACKLink Library

7.2 DEVICE TREE OVERLAY

The Device Tree overlay serves as a descriptor of hardware present on a given system. Implemented on the 3.8 Linux Kernel for the BeagleBone Black, the device tree serves an interface between hardware and system. An example of this, as will be seen in a few of the developed driver blocks, is how analog inputs are configured on the BeagleBone's Linux system. The Linux kernel must be given instructions regarding which hardware (header pins) to use and how this hardware must be configured (pin muxing) as well as any specific drivers or tools to use. This interfacing is carried out through Device Tree Overlays (DTOs).

Several DTOs for various device interfaces are already written, compiled and stored on the BeagleBoard Black. These DTOs must be applied as they are likely not automatically loaded into the kernel on boot. The "echo" BASH command is used to apply the "cape-bone-iio" (analog input) DTO stored in the "/lib/firmware/" directory as seen below:

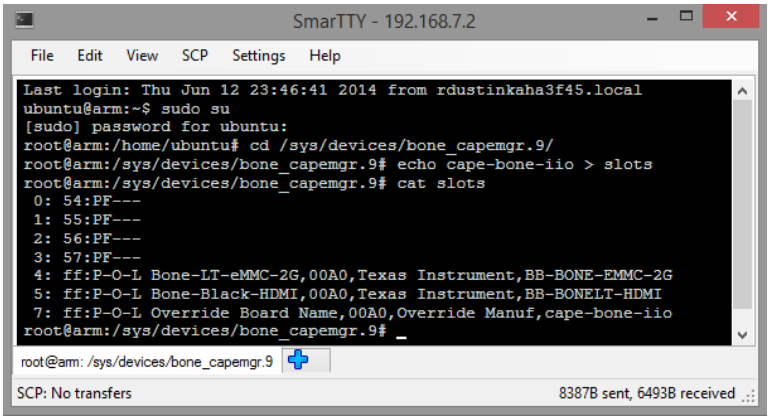


Figure 58: Applying a Device Tree Overlay

The “cat slots” command lists all DTO’s currently applied on the BeagleBone Black. DTOs are simply text files (*.dts) that are compiled into binary format (*.dtb) via a Device Tree Compiler. An example of a Device Tree Overlay for enabling analog input pins on the BeagleBone Black is shown below:

```
/*
 * Copyright (C) 2012 Texas Instruments Incorporated - http://www.ti.com/
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License version 2 as
 * published by the Free Software Foundation.
 */
/dts-v1/;
/plugin/;

/ {
        compatible = "ti,beaglebone", "ti,beaglebone-black";

        /* identification */
        part-number = "iio-test";

        /* state the resources this cape uses */
        exclusive-use =
                /* the pin header uses */
                "P9.39",        /* AIN0 */
                "P9.40",        /* AIN1 */
                "P9.37",        /* AIN2 */
                "P9.38",        /* AIN3 */
                "P9.33",        /* AIN4 */
                "P9.36",        /* AIN5 */
                "P9.35",        /* AIN6 */
                /* the hardware IP uses */
                "tscadc";

        fragment@0 {
                target = <&ocp>;
                __overlay__ {
                        /* avoid stupid warning */
                        #address-cells = <1>;
                        #size-cells = <1>;

                        tscadc {
                                compatible = "ti,ti-tscadc";
                                reg = <0x44e0d000 0x1000>;

                                interrupt-parent = <&intc>;
                                interrupts = <16>;
                                ti,hwmods = "adc_tsc";
                                status = "okay";

                                adc {
                                        ti,adc-channels = <0 1 2 3 4 5 6 7>;
                                };
                        };

                        test_helper: helper {
                                compatible = "bone-iio-helper";
                                vsense-name  = "AIN0", "AIN1", "AIN2", "AIN3", "AIN4", "AIN5", "AIN6", "AIN7";
                                vsense-scale = <100     100     100     100     100     100     100     100>;
                                status = "okay";
                        };
                };
        };
};
```

Figure 59: Device Tree Overlay Source Code

7.3 BLACKLINK LIBRARY BLOCK – GPIO READ

Block Functionality Overview

The GPIO read block, as its name suggests, allows for the reading of digital IO data from any of the GPIO configurable ports on the BeagleBone Black. The following image illustrates its practical implementation in Simulink where the status of a push button on the prototyping test bed is read in real time.

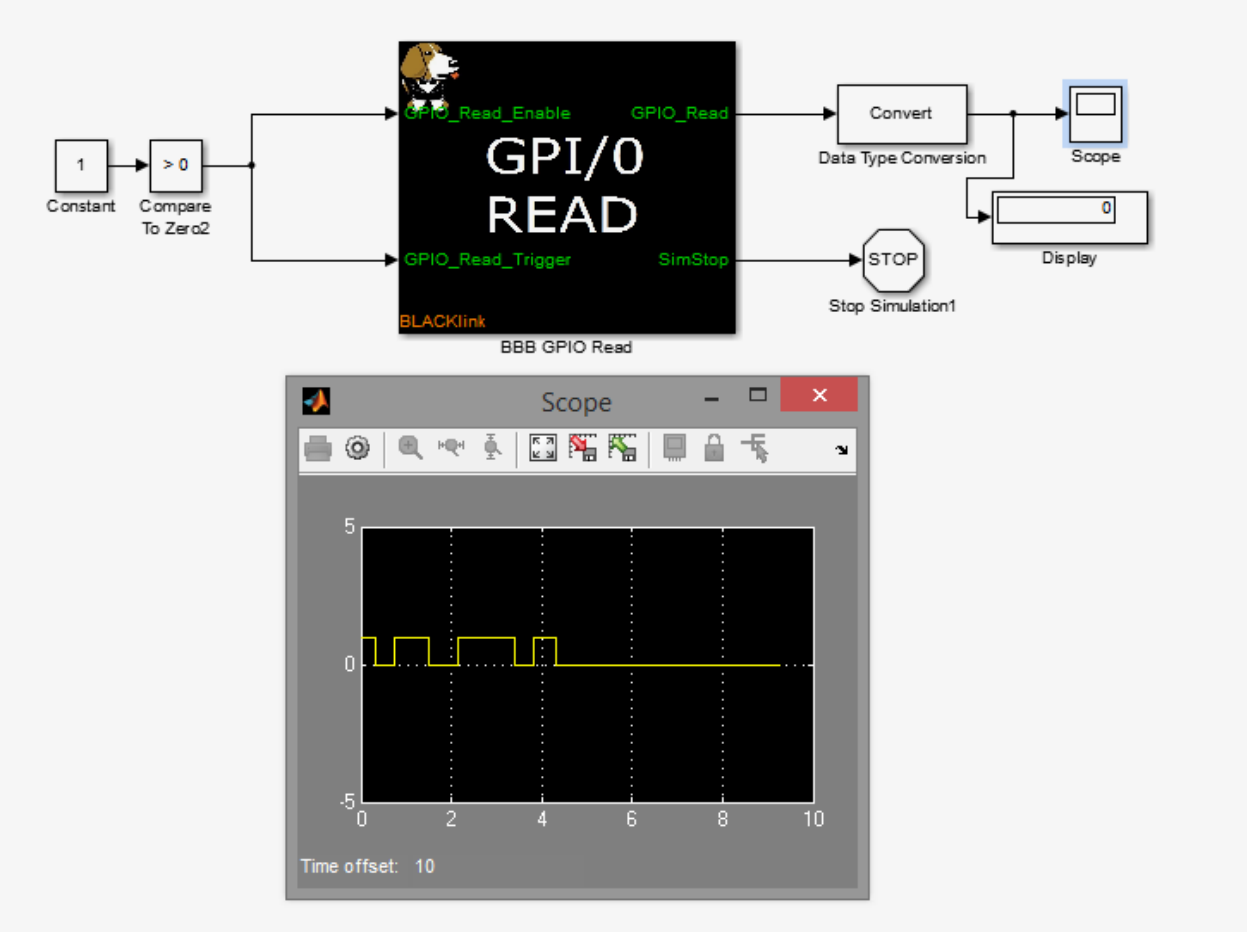


Figure 60 GPIO Read Simulink Implementation

Block Functionality Requirements

In order to for the GPIO read block to function properly, the selected GPIO pin's pinmux mode must be set to "**Mode 7"**. A Device Tree Overlay should be applied so as to the set the pin mode to "**Input"**.

Block Inputs

GPIO_Read_Enable (Boolean): The enable input allows for dynamics enabling/disabling of the block during run time. Should the enable input be set to false, all code within the block will be bypassed.

GPIO_Read_Trigger (Boolean): The GPIO trigger allows for dynamic real time triggering of the read functionality. When set to false, the block will connect to the BeagleBone Black's 'sysfs' interface but will not pole the GPIO status until the trigger is set to true.

Block Outputs

GPIO_Read (Boolean): The GPIO read output simply polls the current GPIO state and places into a Boolean signal which may be used for logic operations in the Simulink model.

SimStop (Boolean): The stop simulation produces a true Boolean signal if a critical error is detected in the block. This is typically caused by communication or mismatched configuration errors. The Simulink simulation will automatically stop if this port is connected to a "Stop Simulation" block. Should this port not be connected, the simulation will continue and the block will simply continue executing its task on every time step. Depending on the debug reporting level, the error may be displayed in the Window's command prompt.

Block Configurable Parameters

GPIO Pin: This parameter allows the user to select one of the BeagleBone's GPIO pins. Note that the selected GPIO pin must be configured via a DTO according to the functionality requirements stated above.

Debug Level: This parameter allows the user to set the level of diagnostic information written to the Window's command prompt and may prove useful for debugging purposes. The user may select between having no debug information to having warnings, critical errors, block specific diagnostics and program flow information written to the command prompt. Note that each subsequent debug level adds its respective information to debug levels found below it.

Block Configuration Interface

The following figure is a screenshot of the block's configuration interface:

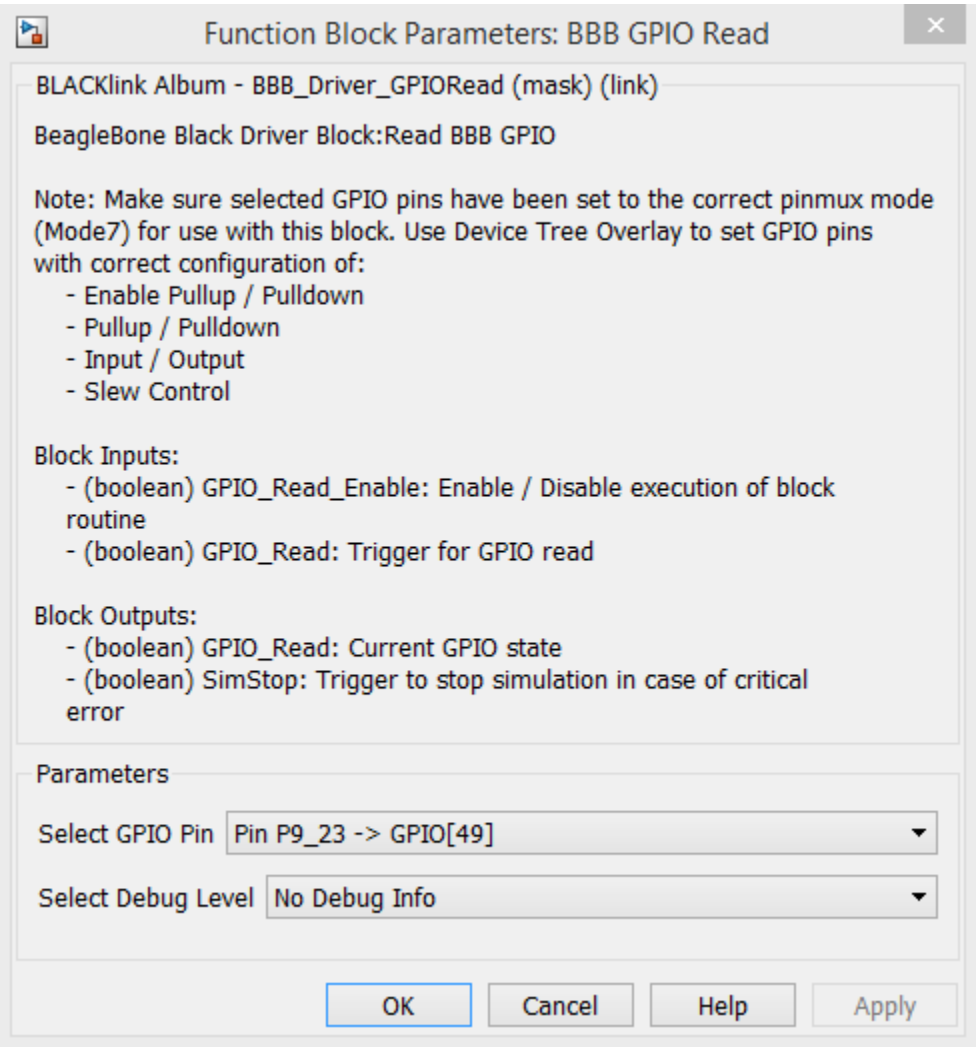


Figure 61: GPIO Read Configuration Interface

7.4 BLACKLINK LIBRARY BLOCK – GPIO WRITE

Block Functionality Overview

The GPIO write block, again, as its name suggests, allows for the writing of digital IO data to any of the GPIO configurable ports on the BeagleBone Black. The following image illustrates its practical implementation in Simulink where a slider or sine wave triggers the real-time activation of a digital output connected to a LED on the prototyping test bed.

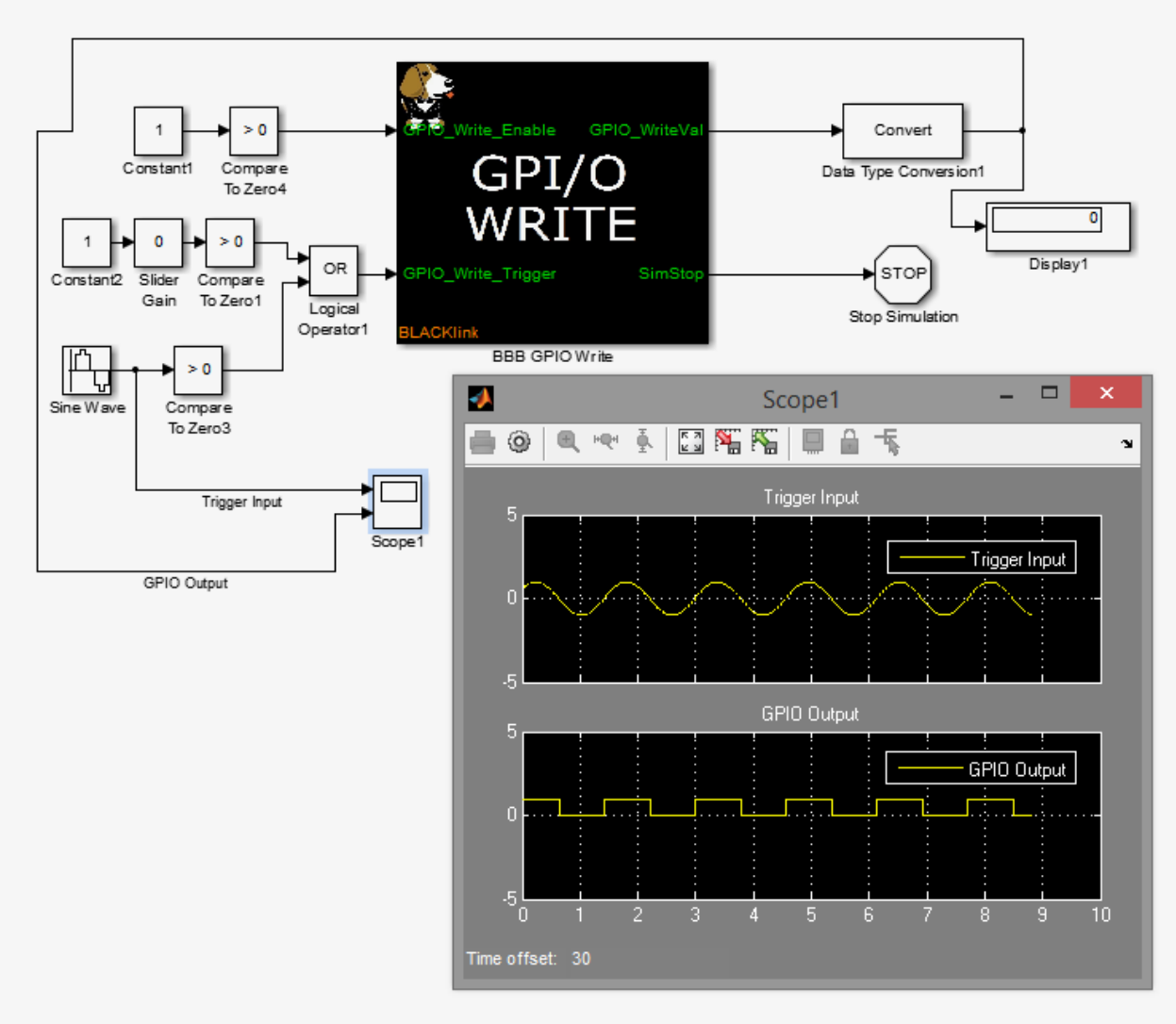


Figure 62: GPIO Write Simulink Implementation

Block Functionality Requirements

In order to for the GPIO write block to function properly, the selected GPIO pin's pinmux mode must be set to "**Mode 7"**. A Device Tree Overlay should be used to the set the pin mode to "**Output"**.

Block Inputs

GPIO_Write_Enable (Boolean): The enable input allows for dynamic enabling/disabling of the block during run time. Should the enable input be set to false, all code within the block will be bypassed.

GPIO_Write_Trigger (Boolean): The GPIO trigger allows for dynamic real time triggering of the write functionality. When set to false, the block will connect to the BeagleBone Black's 'sysfs' interface and set the selected pin value to low. When the trigger is set to true, the pin value will be set to high.

Block Outputs

GPIO_WriteVal (Boolean): The GPIO write value output polls the current GPIO state and places it into a Boolean signal which may be used for logic operations in the Simulink model. The output therefore acts as a feedback confirming the state of the pin.

SimStop (Boolean): The stop simulation produces a true Boolean signal if a critical error is detected in the block. This is typically caused by communication or mismatched configuration errors. The Simulink simulation will automatically stop if this port is connected to a "Stop Simulation" block. Should this port not be connected, the simulation will continue and the block will simply continue executing its task on every time step. Depending on the debug reporting level, the error may be displayed in the Window's command prompt.

Block Configurable Parameters

GPIO Pin: This parameter allows the user to select the one of the BeagleBone's GPIO pins. Note that the selected GPIO pin must be configured according to the functionality requirements stated above.

Debug Level: This parameter allows the user to set the level of diagnostic information written to the Window's command prompt and may prove useful for debugging purposes. The user may select between having no debug information to having warnings, critical errors, block specific diagnostics and program flow information written to the command prompt. Note that each subsequent debug level adds its respective information to debug levels found below it.

Block Configuration Interface

The following figure is a screenshot of the block's configuration interface:

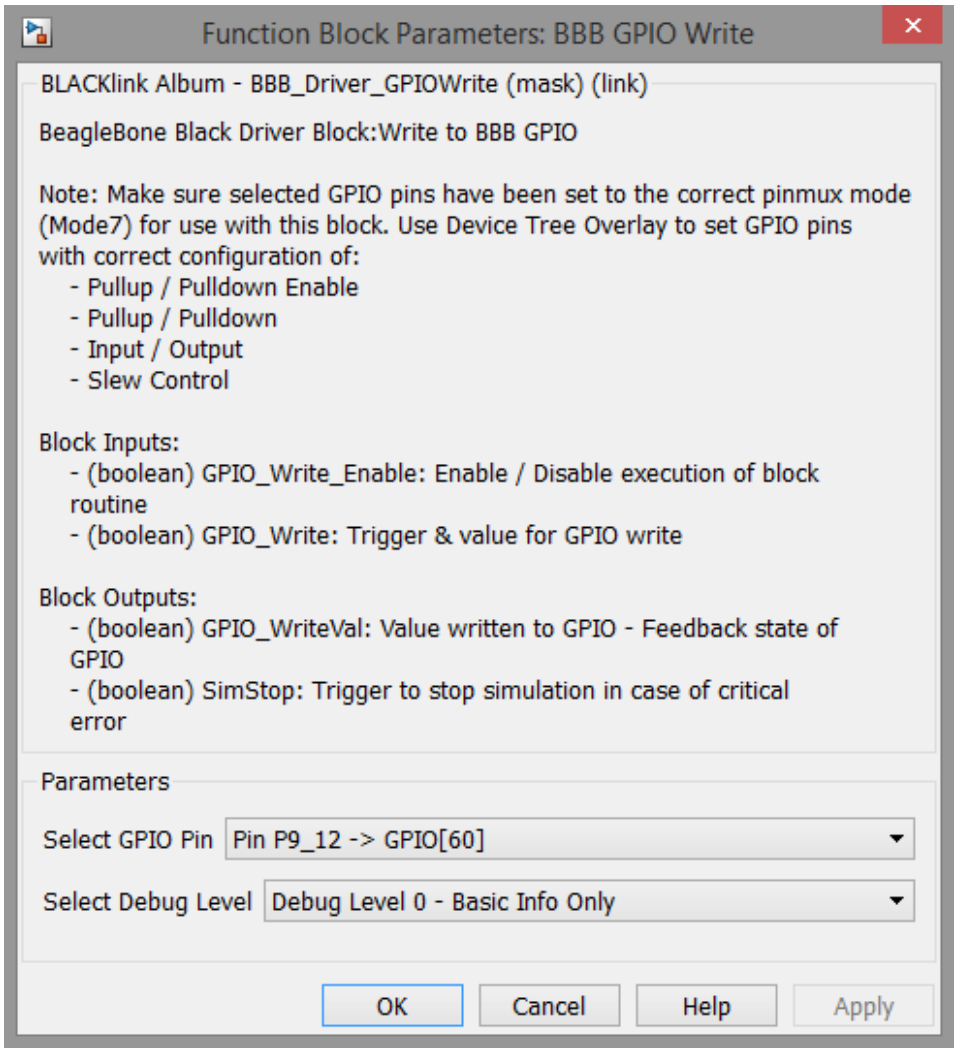


Figure 63: GPIO Write Parameter Interface

7.5 BLACKLINK LIBRARY BLOCK – I2C READ

Block Functionality Overview

The I2C Read block grants data communication to any device connected to one of the two existing I2C bus channels on the BeagleBone Black. With the correct data I2C bus and device address configured, the block will read a specific byte from the device's resister. Note that for the moment, only a single byte may be read per time-step. A future update to the block will allow for a specified range of bytes to be read from a device's register. The following image illustrates a practical implementation in Simulink with the I2C block reading the first register byte (0x00) from the I2C enabled L3G4200D gyroscope. The gyroscope is connected to I2C Bus 1 with a device bus address of (0X68). The captured value from the device's register is sent, via the block's "I2C_Read_Value" output port, to the Display block.

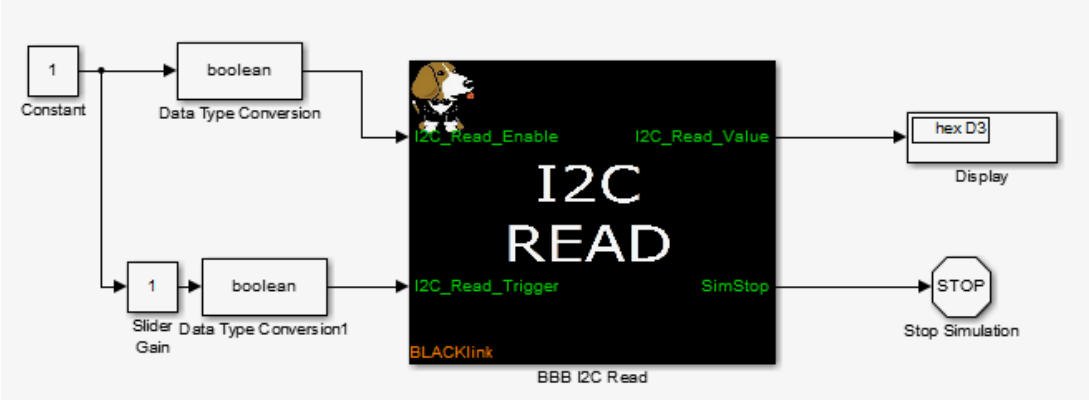


Figure 64: I2C Read Simulink Implementation

Block Functionality Requirements

In order for the I2C read block to function correctly, the selected I2C channel must be enabled on the BeagleBone Black via the Device Tree Overlay.

Block Inputs

I2C_Read_Enable (Boolean): The enable input allows for dynamic enabling/disabling of the block during run time. Should the enable input be set to false, all code within the block will be bypassed.

I2C_Read_Trigger (Boolean): The I2C read trigger allows for dynamic real time triggering of the I2C read functionality. When set to false, the block will connect to the device found on the selected BeagleBone Black's I2C channel but will not pole the register. When the trigger is set to true, current data in the selected register will be read and saved within the block for eventual output.

Block Outputs

I2C_Read_Value (uint8): The data read from the configured device register will be placed into the read value output port signal as an unsigned eight bit integer (byte).

SimStop (Boolean): The stop simulation produces a true Boolean signal if a critical error is detected in the block. This is typically caused by communication or mismatched configuration errors. The Simulink simulation will automatically stop if this port is connected to a "Stop Simulation" block. Should this port not be connected, the simulation will continue and the block will simply continue executing its task on every time step. Depending on the debug reporting level, the error may be displayed in the Window's command prompt.

Block Configurable Parameters

I2C Channel: This parameter allows the user to select between the two I2C channels present on the BeagleBone Black.

I2C Device Address: As multiple devices may be connected to a single I2C bus, each device has an identifying address, typically represented by a value between 0->255. Hence the address of the device targeted for reading must be entered in this field by the user.

I2C Device Register Number: I2C enabled devices store configuration and sensing data in their device registers. The information stored in each register is stipulated by the manufacturer and can typically be found in the device's technical documentation. The specific register address to be read must be entered in this field by the user.

Debug Level: This parameter allows the user to set the level of diagnostic information written to the Window's command prompt and may prove useful for debugging purposes. The user may select between having no debug information to having warnings, critical errors, block specific diagnostics and program flow information written to the command prompt. Note that each subsequent debug level adds its respective information to debug levels found below it.

Block Configuration Interface

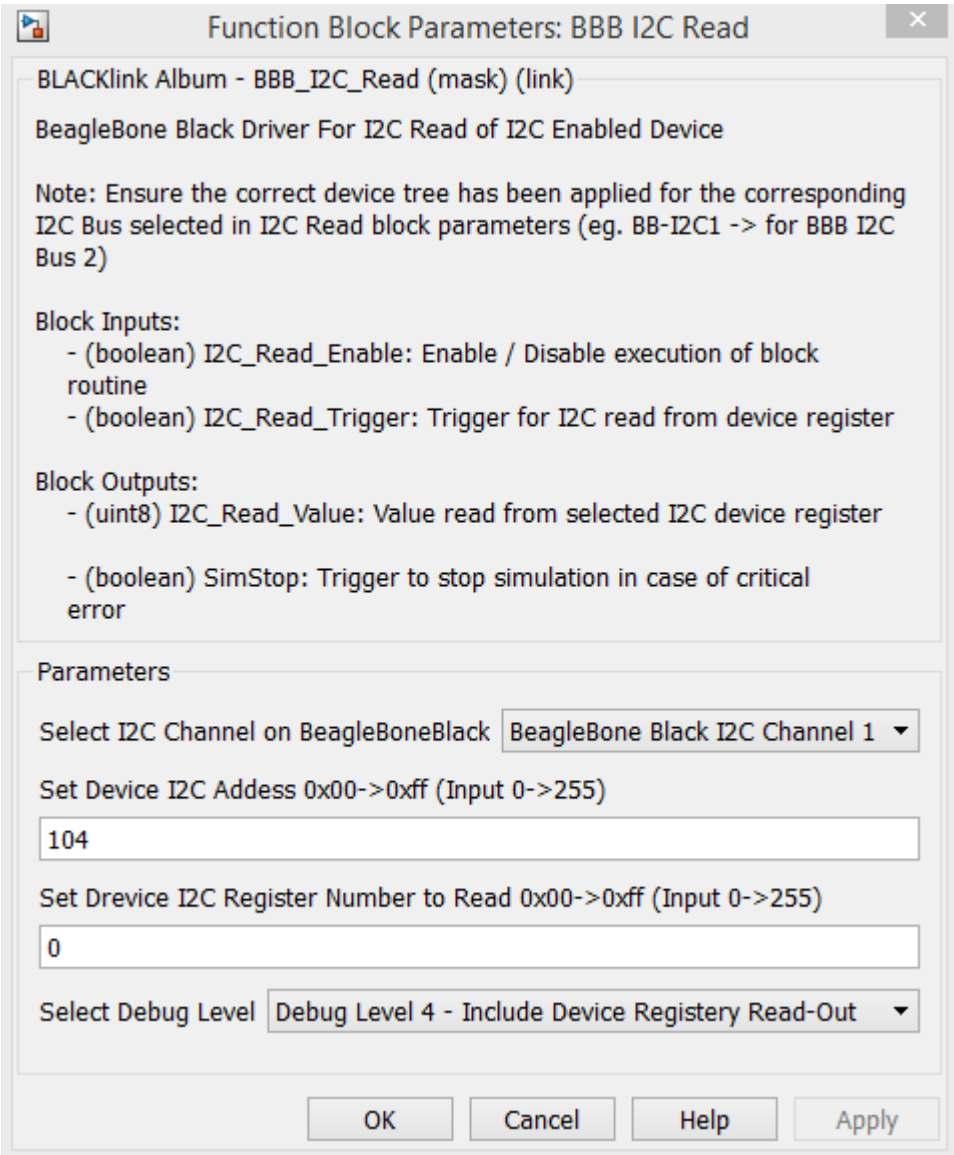


Figure 65: I2C Read Parameter Interface

7.6 BLACKLINK LIBRARY BLOCK – I2C WRITE

Block Functionality Overview

The I2C Write block grants data communication to any device connected to one of the two existing I2C channels on the BeagleBone Black. With the correct data I2C bus and device address configured, the block will write a user specified byte value to the device's resister. The following image illustrates a practical implementation in Simulink with the I2C block writing to the (0x20) control register byte on the I2C enabled L3G4200D gyroscope. This sets the gyroscope to normal mode and enables X, Y and Z axis readings. The gyroscope is connected to I2C Bus 1 with a device bus address of (0X68). The value read back from the device's register is sent to the "I2C_Written_Value" output port and revealed on the Display block.

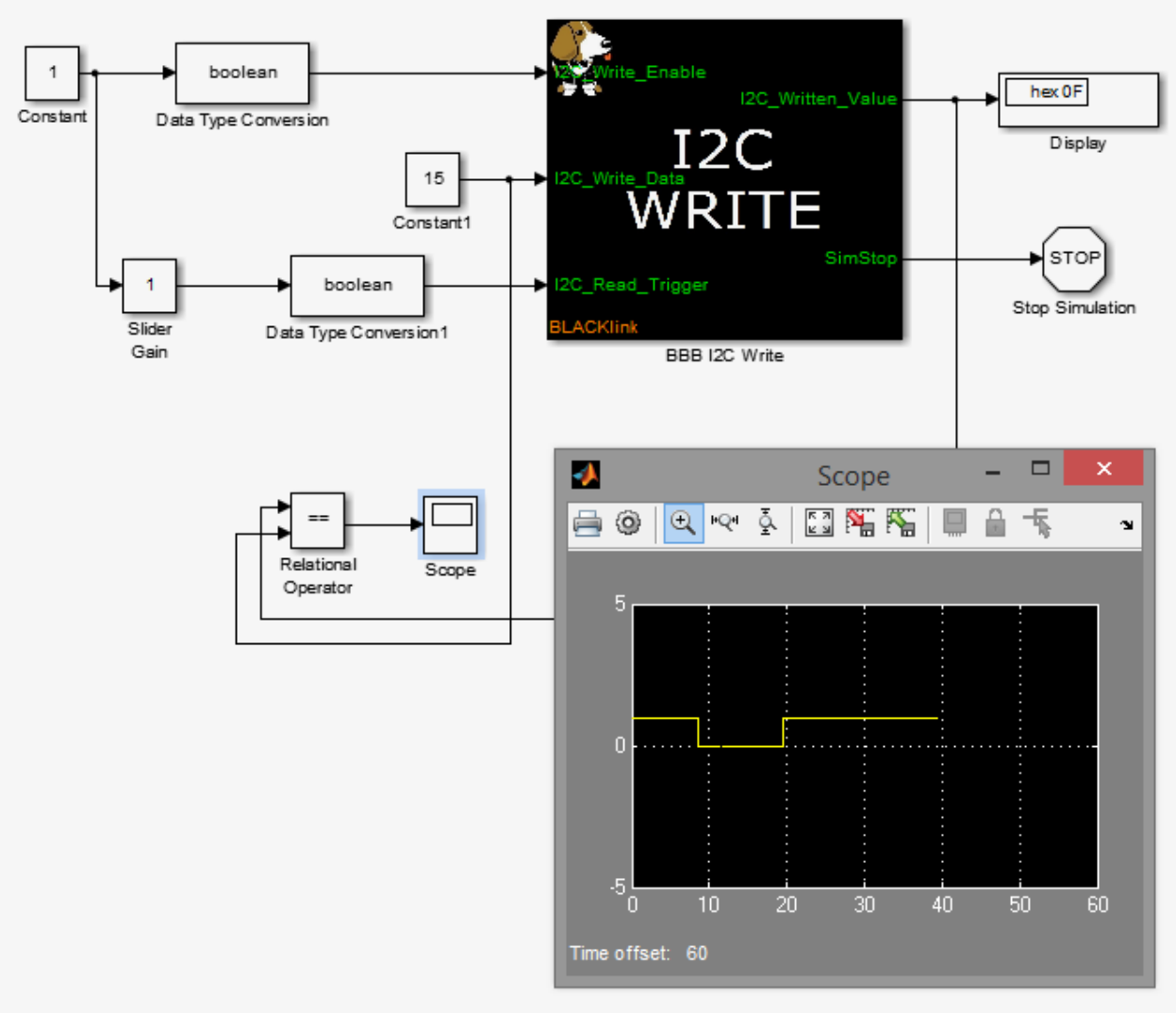


Figure 66: I2C Write Simulink Implementation

Block Functionality Requirements

In order for the I2C write block to function correctly, the selected I2C channel must be enabled on the BeagleBone Black via the Device Tree Overlay.

Block Inputs

I2C_Write_Enable (Boolean): The enable input allows for dynamic enabling/disabling of the block during run time. Should the enable input be set to false, all code within the block will be bypassed.

I2C_Write_Data (Byte): The value of the data byte to be written to the configured I2C device register. Note that this may be changed dynamically during simulation.

I2C_Write_Trigger (Boolean): The I2C write trigger allows for dynamic real time triggering of the I2C write functionality. When set to false, the block will connect to the device found on the selected BeagleBone Black's I2C channel but will not write to the register. When the trigger is set to true, the value from the block's input data port will be written to the selected register. Note that the output port may be used as a feedback loop to confirm data written to the register and therefore disable the trigger avoiding re-writing of the same value on each time-step.

Block Outputs

I2C_Write_Value (uint8): The targeted device register will be read while the write trigger is active and the value sent to the write value output port. The output of this port may therefore be used to confirm success of the write instruction.

SimStop (Boolean): The stop simulation produces a true Boolean signal if a critical error is detected in the block. This is typically caused by communication or mismatched configuration errors. The Simulink simulation will automatically stop if this port is connected to a "Stop Simulation" block. Should this port not be connected, the simulation will continue and the block will simply continue executing its task on every time step. Depending on the debug reporting level, the error may be displayed in the Window's command prompt.

Block Configurable Parameters

I2C Channel: This parameter allows the user to select between the two I2C channels present on the Beagle Board Black.

I2C Device Address: As multiple devices may be connected to a single I2C bus, each device has an identifying address, typically represented by a value between 0->255. Hence the address of the device targeted for writing must be entered in this field by the user.

I2C Device Register Number: I2C enable devices store configuration and sensing data in their device registers. The information stored in each register is stipulated by the manufacturer and can typically be found in the device's technical documentation. The specific register data to be written to must be entered in this field by the user.

Debug Level: This parameter allows the user to set the level of diagnostic information written to the Window's command prompt and may prove useful for debugging purposes. The user may select between having no debug information to having warnings, critical errors, block specific diagnostics and program flow information written to the command prompt. Note that each subsequent debug level adds its respective information to debug levels found below it.

Block Configuration Interface

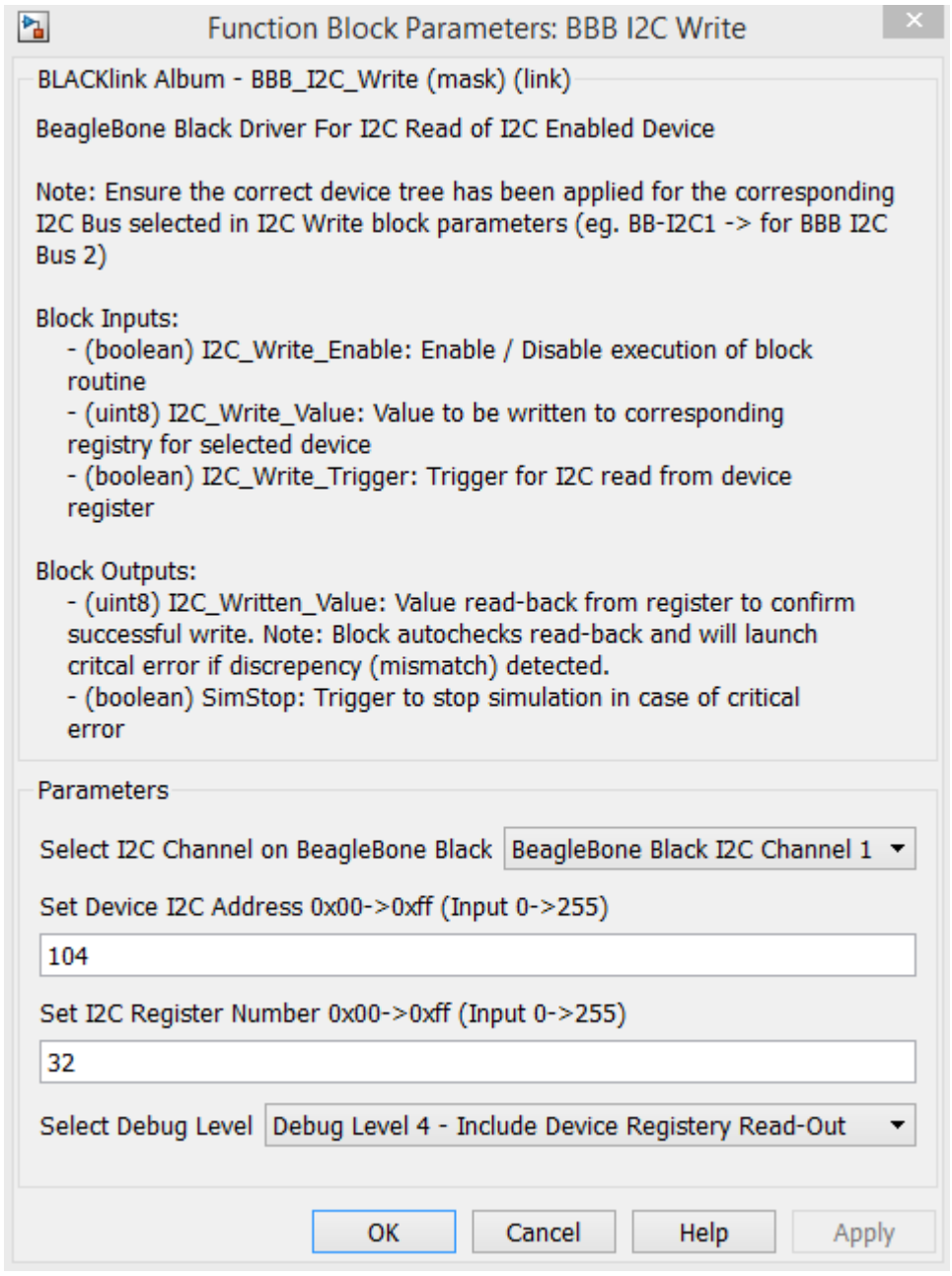


Figure 67: I2C Write Parameter Interface

7.7 BLACKLINK LIBRARY BLOCK – ANALOG INPUT READ

Block Functionality Overview

The analog input block provides the user with the capability of reading the voltage from any one of the seven analog input pins present on the BeagleBone Black's header. The block also provides the option of running a calibration routine in which the first ten time-step scans are averaged and the result applied as an offset to subsequent readings. This may be used as a zeroing function if the analog input is connected to a sensor. Note that in such a case, the sensor must be in its nominal configuration when the simulation is started.

The following image demonstrates a practical implementation in Simulink with the ADC block reading the output voltage from the X axis of the ADXL3300 accelerometer.

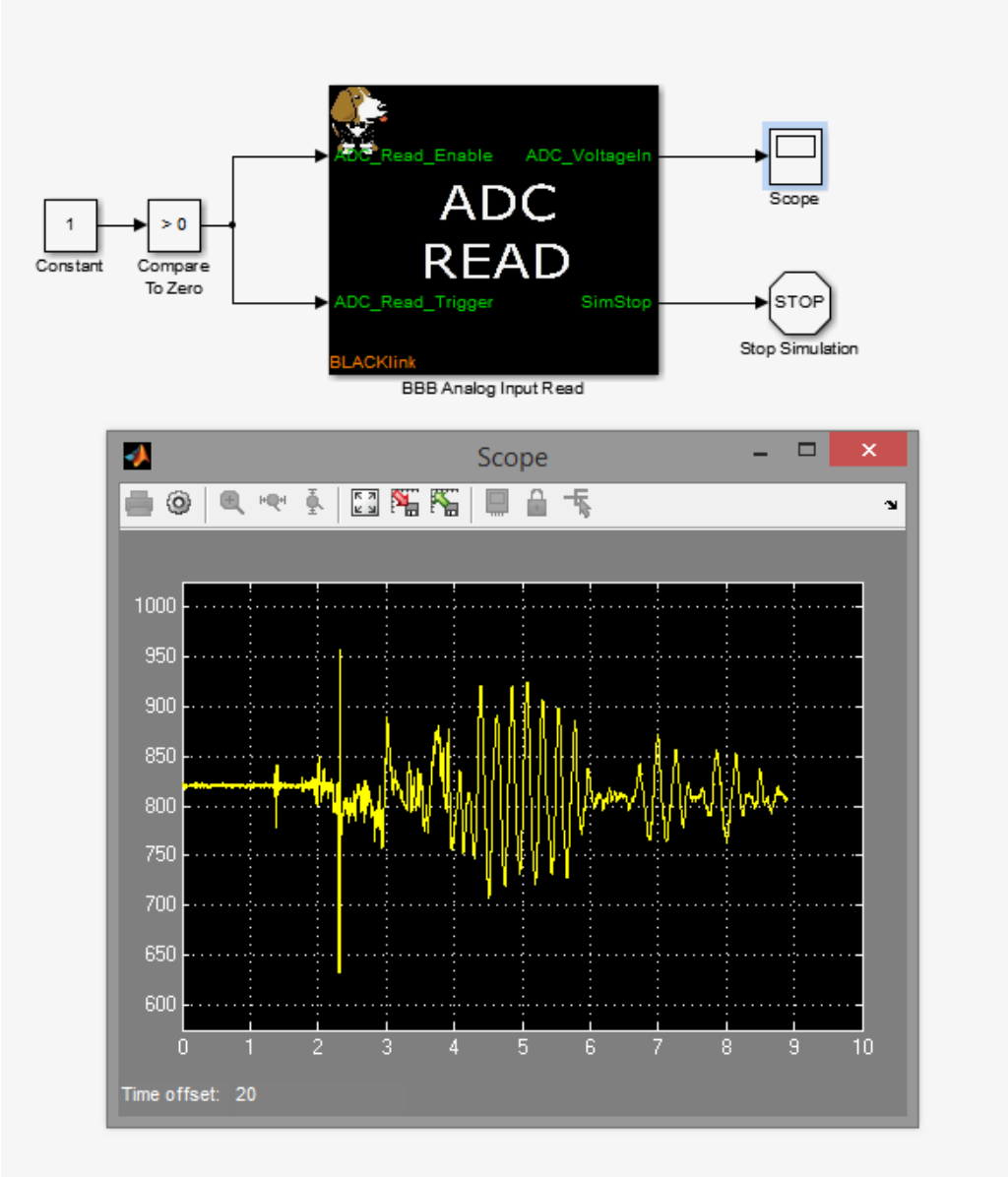


Figure 68: Analog Input Read Simulink Implementation

Block Functionality Requirements

In order for the analog read block to function correctly, the selected analog pins must correctly muxed and the corresponding Device Tree Overlay applied. In the case of this project, analog inputs were configured via the "cape-bone-iio" DTO.

Block Inputs

ADC_Read_Enable (Boolean): The enable input allows for dynamic enabling/disabling of the block during run time. Should the enable input be set to false, all code within the block will be bypassed.

ADC_Read_Trigger (Boolean): The analog read trigger allows for dynamic real time triggering of the analog read functionality. When set to false, the block will connect to the BealgeBone Black's "sysfs" interface but will not poll the data from the selected analog input pin. When the trigger is set to true the data from the analog pin will be read and sent to the "ADC_VoltageIn" output port.

Block Outputs

ADC_VoltageIn (Double) (mV): Voltage read (in mV) from the analog input port is sent to the ADC_VoltageIn output port while the read trigger is active.

SimStop (Boolean): The stop simulation produces a true Boolean signal if a critical error is detected in the block. This is typically caused by communication or mismatched configuration errors. The Simulink simulation will automatically stop if this port is connected to a "Stop Simulation" block. Should this port not be connected, the simulation will continue and the block will simply continue executing its task on every time step. Depending on the debug reporting level, the error may be displayed in the Window's command prompt

Block Configurable Parameters

AIN Pin: The user must select the specific analog pin from which the voltage is to be read. The BeagleBone Black is equipped with a total of seven ADC pins.

Enable Calibration Routine: The user may choose to enable or disable and automatic calibration routine. If enabled, the block averages the first ten voltage scans and applies their average as an offset to subsequent readings. This may prove handy in the zeroing of sensor readings, however, the user must ensure the sensor is in its nominal sate when the block is initially triggered.

Debug Level: This parameter allows the user to set the level of diagnostic information written to the Window's command prompt and may prove useful for debugging purposes. The user may select between having no debug information to having warnings, critical errors, block specific diagnostics and program flow information written to the command prompt. Note that each subsequent debug level adds its respective information to debug levels found below it.

Block Configuration Interface

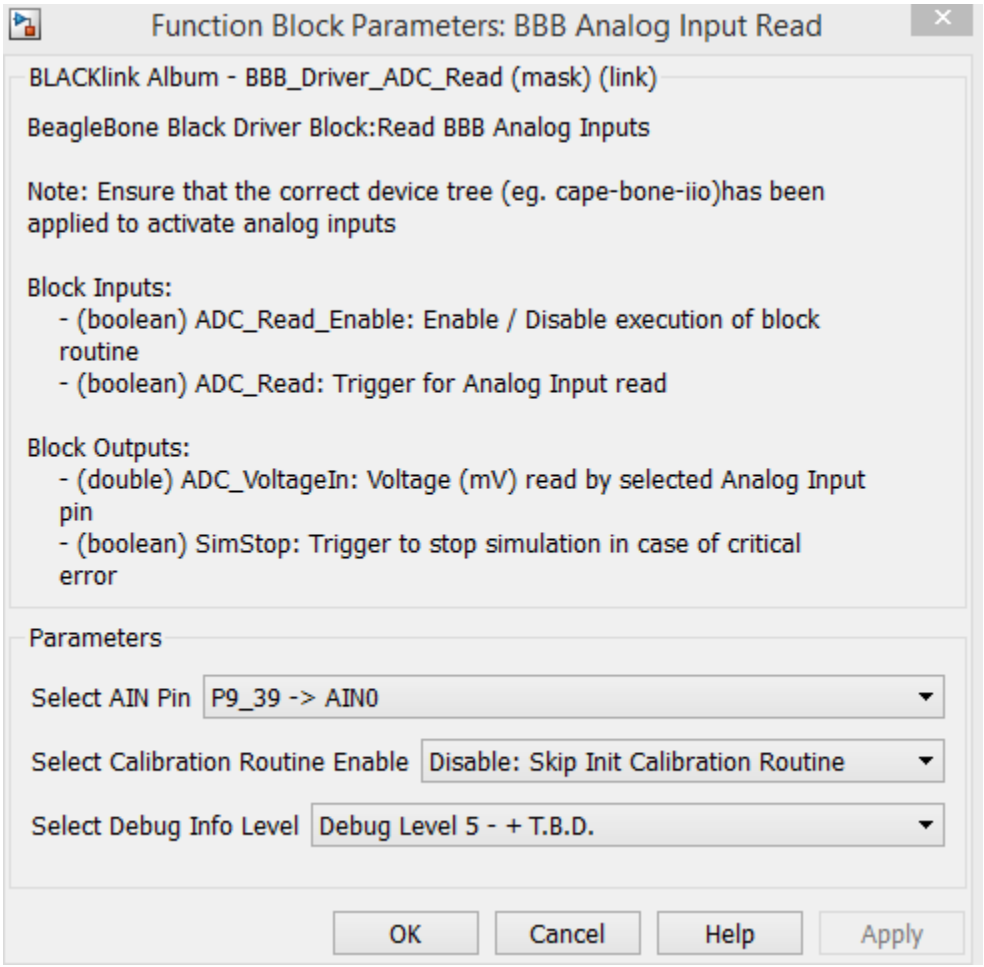


Figure 69: BBB ADC Read Configuration Interface

7.8 BLACKLINK LIBRARY BLOCK – USER LED CONTROL

Block Functionality Overview

The user LED control block allows for the toggling of the four onboard LEDs based on a supplied trigger input. The four LEDS are typically set to flash on factory default triggers from BeagleBone Black's OS:

- USR0: Heartbeat
- USR1: MMC0
- USR2: CPU0
- USR3: MMC1

Enabling the USR LED block replaces these triggers with ones defined by the user in the Simulink model while disabling the block returns the LEDs to their default triggers. This USRLED control functionality may be helpful for custom debugging or diagnostic purposes.

The following image demonstrates a practical implementation in Simulink of all four LEDS begin triggered at varying frequencies via USR LED blocks.

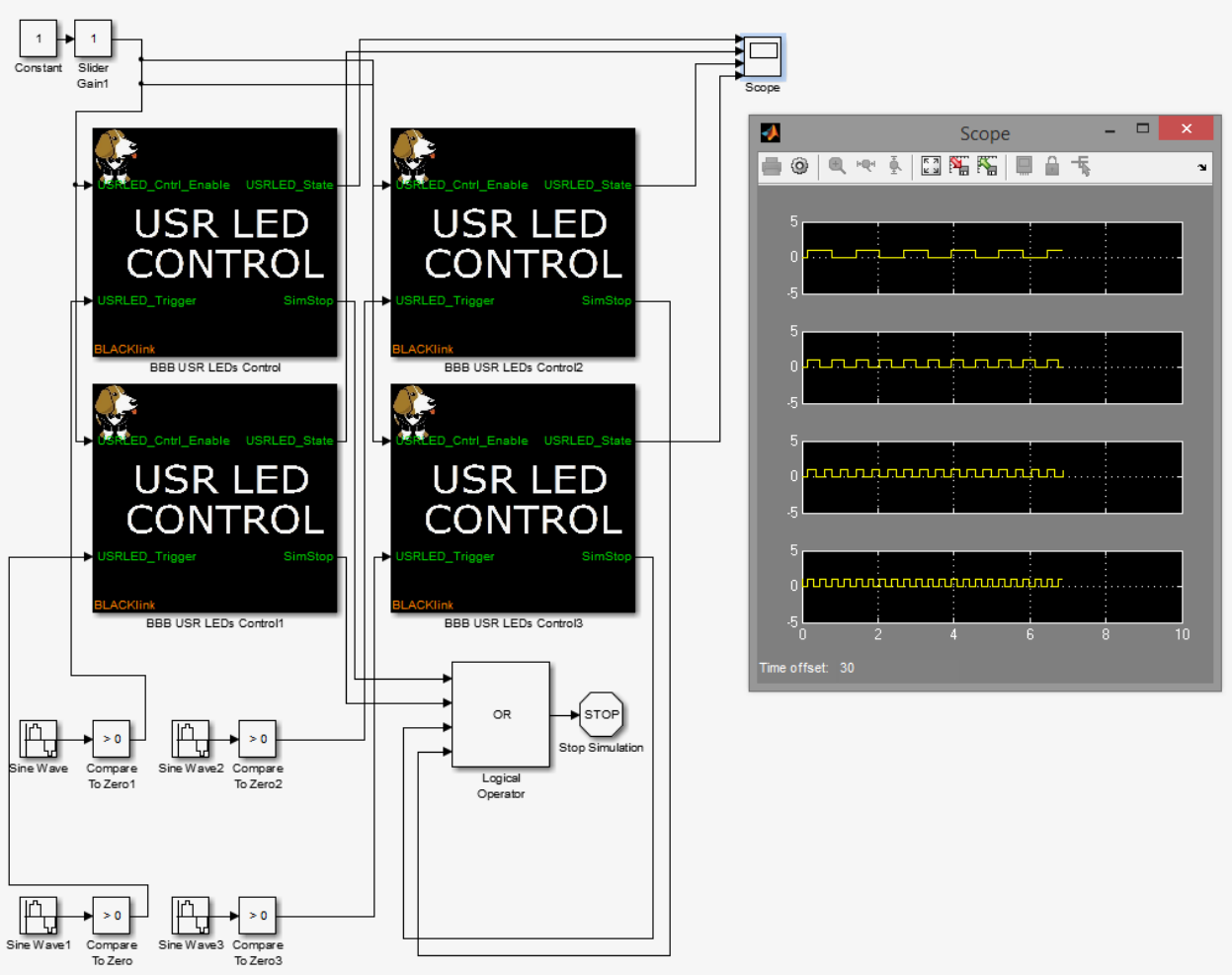


Figure 70: USRLED Simulink Implementation

Block Functionality Requirements

Enabling the block hands over LED trigger control to the block. Disabling the block sets the selected USR LED back to its factory default state.

Block Inputs

USRLED_Cntrl_Enable (Double): The control enable input allows for dynamic enabling/disabling of the block during run time. Should the enable input be set to false, all code within the block will be bypassed

USRLED_Trigger (Boolean): The user LED read trigger allows for dynamic real time triggering of the four onboard LEDS. When set to false, the block will connect to the BeagleBone Black's "sysfs" interface and will set the selected LED's brightness status to low. When the trigger is set to true the LED's brightness status will be set to high.

Block Outputs

USRLED_State (Boolean): The current state of the selected USR LED's brightness value is polled and sent to the state output port. The state outputs true if the brightness is currently set to high and false in the reverse case.

SimStop (Boolean): The stop simulation produces a true Boolean signal if a critical error is detected in the block. This is typically caused by communication or mismatched configuration errors. The Simulink simulation will automatically stop if this port is connected to a "Stop Simulation" block. Should this port not be connected, the simulation will continue and the block will simply continue executing its task on every time step. Depending on the debug reporting level, the error may be displayed in the Window's command prompt.

Block Configurable Parameters

Select USR LED: The user must select which of the four LEDs (USR0 -> USR3) is to be controlled. The LEDs numbering may be found printed on the BeagleBoard Black itself.

Debug Level: This parameter allows the user to set the level of diagnostic information written to the Window's command prompt and may prove useful for debugging purposes. The user may select between having no debug information to having warnings, critical errors, block specific diagnostics and program flow information written to the command prompt. Note that each subsequent debug level adds its respective information to debug levels found below it.

Block Configuration Interface

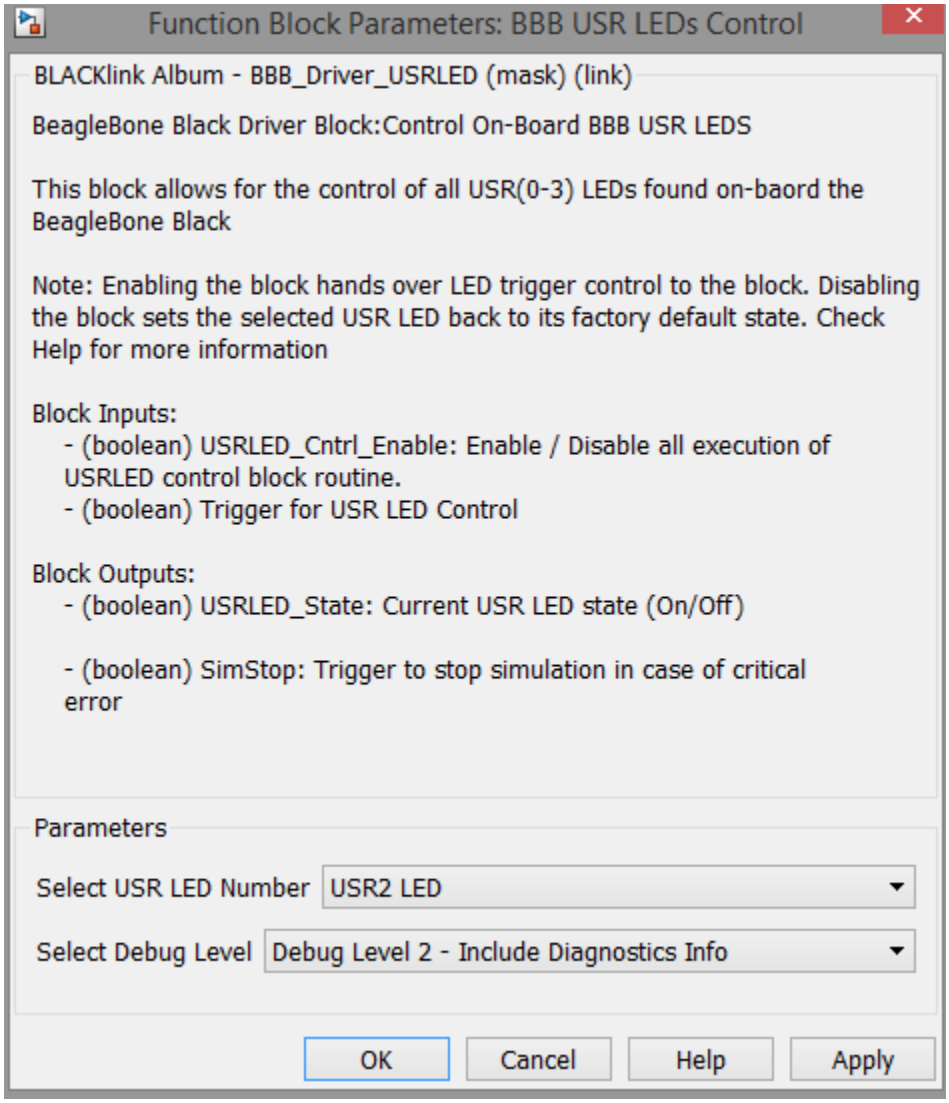


Figure 71: USRLED Configuration Interface

7.9 BLACKLINK LIBRARY BLOCK – DEVICE DRIVER – ADXL330 ACCELEROMETER

Block Functionality Overview

The ADXL300 Accelerometer driver block provides the user with the capability to capture acceleration data along the sensor's X, Y and Z axis. This data is then made available in Simulink for further processing. As discussed in section (4.3.1), the ADXL300 sensor outputs acceleration data as voltage from three separate pins. Therefore the sensor must be connected to three of the BeagleBone Black's analog inputs. The block converts the voltage data into acceleration data depending on the sensitivity parameter selected by the user. A calibration routine may be enabled in order to zero the accelerometer readings and is a function of the selected sensitivity setting.

The following image demonstrates a practical implementation in Simulink of the ADXL300 driver block reading from all three of the sensor axis.

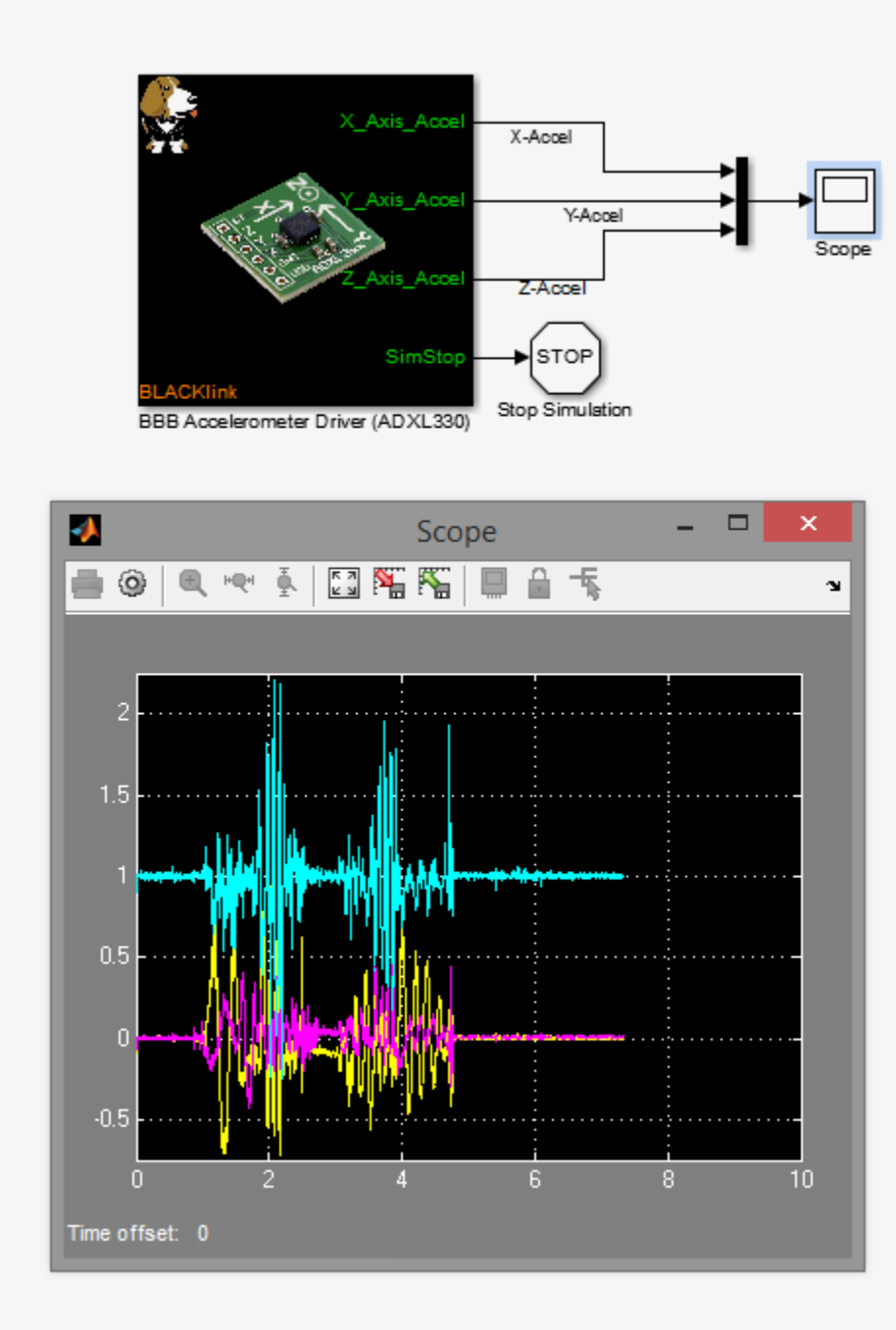


Figure 72: ADXL330 Driver Simulink Implementation

Block Functionality Requirements

The driver block has a preconfigured “pin-out” to connect to the accelerometer, therefore the user must make sure the AIN0/AIN1/AIN2 analog pins are connected, respectively, to the X, Y and Z axis outputs of the accelerometer.

The BeagleBone Black must also have its header pins correctly muxed and the corresponding Device Tree Overlay applied. In the case of this project, analog inputs were configured via the “cape-bone-iio” DTO.

Block Outputs

X_Axis_Accel (Double) (g): The instantaneous acceleration sensed by the ADXL330 along its X axis represented in g-force and based on the selected sensitivity.

Y_Axis_Accel (Double) (g): The instantaneous acceleration sensed by the ADXL330 along its Y axis represented in g-force and based on the selected sensitivity.

Z_Axis_Accel (Double) (g): The instantaneous acceleration sensed by the ADXL330 along its Z axis represented in g-force and based on the selected sensitivity.

SimStop (Boolean): The stop simulation produces a true Boolean signal if a critical error is detected in the block. This is typically caused by communication or mismatched configuration errors. The Simulink simulation will automatically stop if this port is connected to a “Stop Simulation” block. Should this port not be connected, the simulation will continue and the block will simply continue executing its task on every time step. Depending on the debug reporting level, the error may be displayed in the Window’s command prompt.

Block Configurable Parameters

Accelerometer Sensitivity: The sensitivity relates the physical force sensed by the accelerometer to the voltage level it produces. The user must select the sensitivity rating which is determined by the source voltage powering the accelerometer.

Debug Level: This parameter allows the user to set the level of diagnostic information written to the Window’s command prompt and may prove useful for debugging purposes. The user may select between having no debug information to having warnings, critical errors, block specific diagnostics and program flow information written to the command prompt. Note that each subsequent debug level adds its respective information to debug levels found below it.

Block Configuration Interface

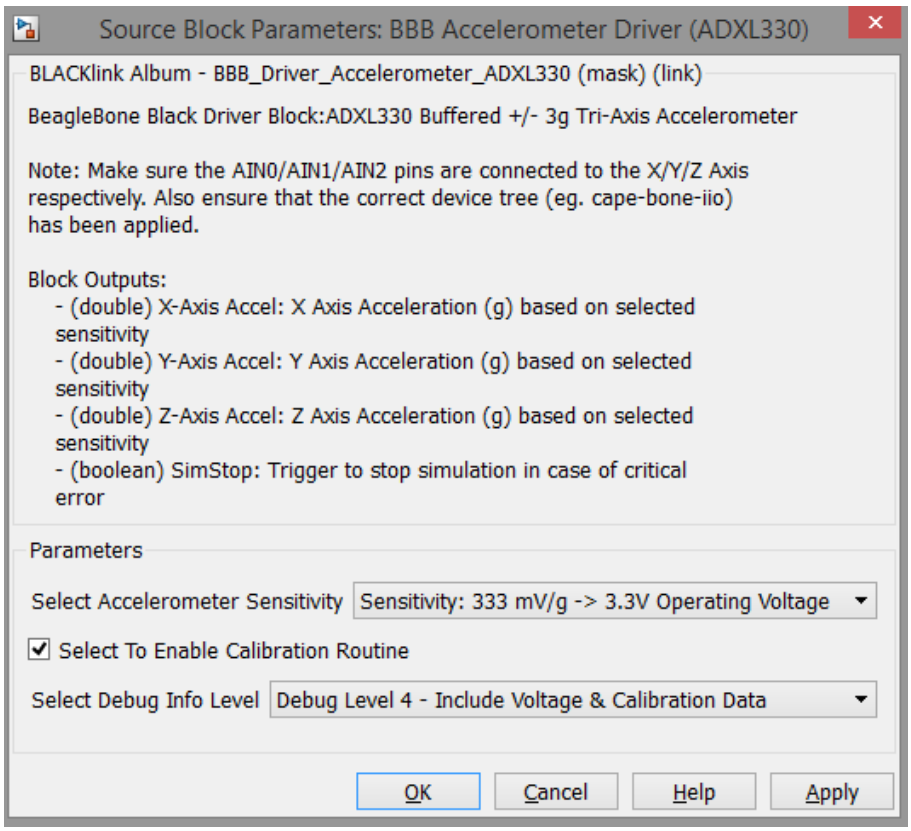


Figure 73: ADXL330 Configuration Interface

7.10 BLACKLINK LIBRARY BLOCK – DEVICE DRIVER – L3G4200D GYROSCOPE

Block Functionality Overview

The LG34200D Gyroscope driver block provides the user with the captured angular rate data around the sensor's X, Y and Z axis. This data is then made available in Simulink for further processing. As discussed in section (4.3.2), the LG34200D sensor is I2C enabled and therefore its angular rate data is read directly from its registries over the I2C bus. The sensor's I2C address and bus location must be configured by the user via the block's parameters.

The gyroscope itself has three different range settings which vary from 250 degrees per second to 2000 degrees per second. This setting can be configured from the driver block which in turn will modify the appropriate register settings on the device.

The following image demonstrates a practical implementation in Simulink of the L3G4200D driver block retrieving data from all three of the sensor's axes.

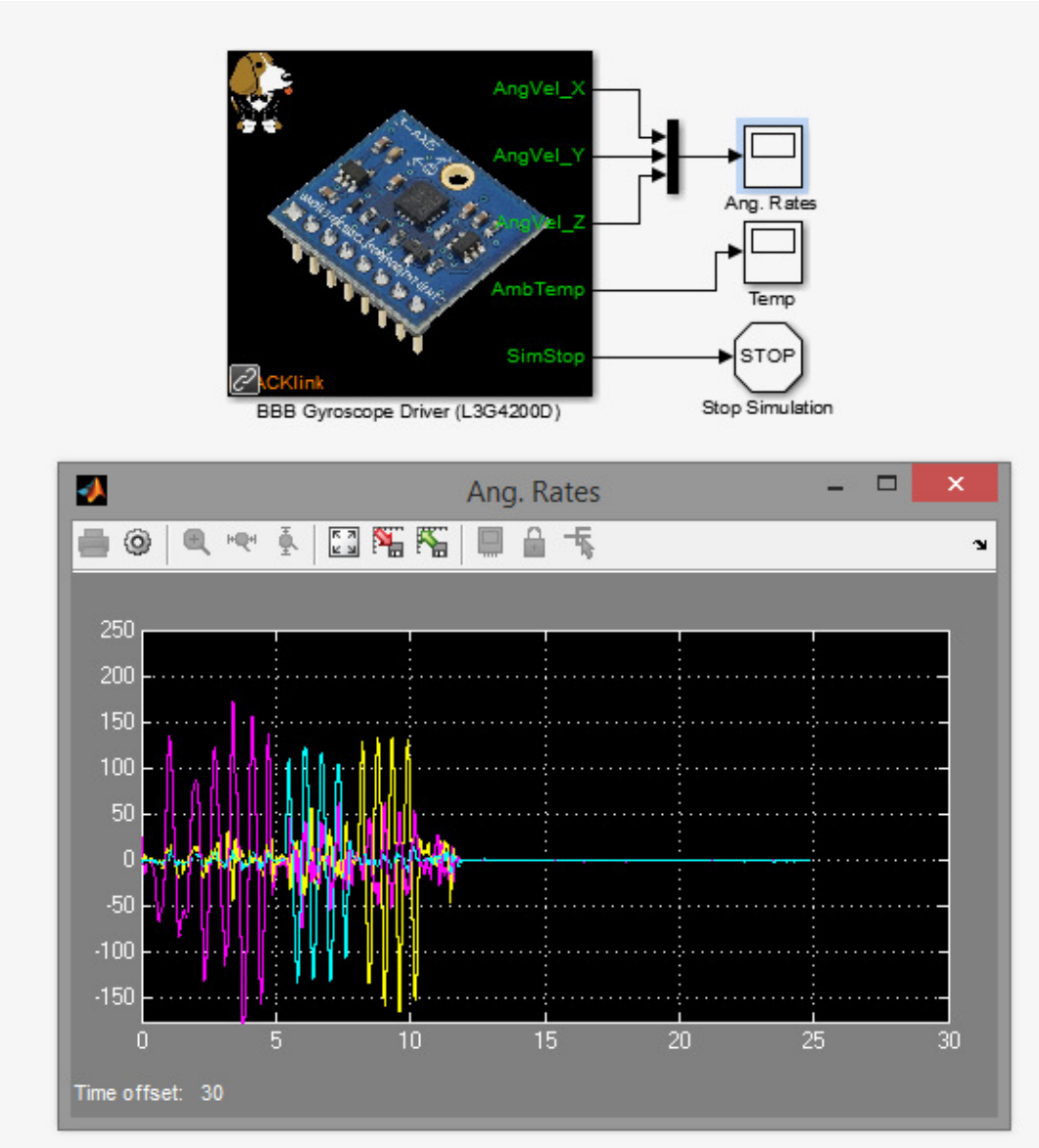


Figure 74: L3G4200D Simulink Implementation

Block Functionality Requirements

In order for the L3G4200D driver block to function correctly, the selected I2C channel must be enabled on the BeagleBone Black via the Device Tree Overlay.

Two possible address settings are available for the L3G4200D (0x068 or 0x069). This may be verified via the gyroscope's technical manual or by executing an address sweep of the I2C bus from the Linux/GNU shell. The address sweep may be executed via the command "i2cdetect -r 0" or "i2cdetect -r 1" for I2C channel 1 or 2 respectively.

Block Outputs

AngVel_X (Double) (deg/s): The instantaneous angular velocity sensed by the L3G4200D around its X axis represented in degrees per second.

AngVel_Y (Double) (deg/s): The instantaneous angular velocity sensed by the L3G4200D around its Y axis represented in degrees per second.

.**AngVel_Z** (Double) (deg/s): The instantaneous angular velocity sensed by the L3G4200D around its Z axis represented in degrees per second.

AmbTemp (int8) (deg C): The ambient temperature measured by the L3G4200D (used internally for temperature compensation of the sensor's readings)

SimStop (Boolean): The stop simulation produces a true Boolean signal if a critical error is detected in the block. This is typically caused by communication or mismatched configuration errors. The Simulink simulation will automatically stop if this port is connected to a "Stop Simulation" block. Should this port not be connected, the simulation will continue and the block will simply continue executing its task on every time step. Depending on the debug reporting level, the error may be displayed in the Window's command prompt.

Block Configurable Parameters

I2C Channel: This parameter allows the user to select which of the two I2C channels on the BeagleBone Black the gyroscope is connected to.

I2C Device Address: User must select of two possible I2C addresses which come pre-configured on the L3G3200D gyroscope. The user may verify this configuration via the sensor's technical documentation or through the "i2cdetect" GNU/Linux shell command.

I2C Gyro Range: The gyroscope's sensing range may be configured between 250, 500 and 2000 degrees per second. The driver block will automatically configure the device's register to match the user's selection on the first time-step of the simulation.

Debug Level: This parameter allows the user to set the level of diagnostic information written to the Window's command prompt and may prove useful for debugging purposes. The user may select between having no debug information to having warnings, critical errors, block specific diagnostics and program flow information written to the command prompt. Note that each subsequent debug level adds its respective information to debug levels found below it.

.

Block Configuration Interface

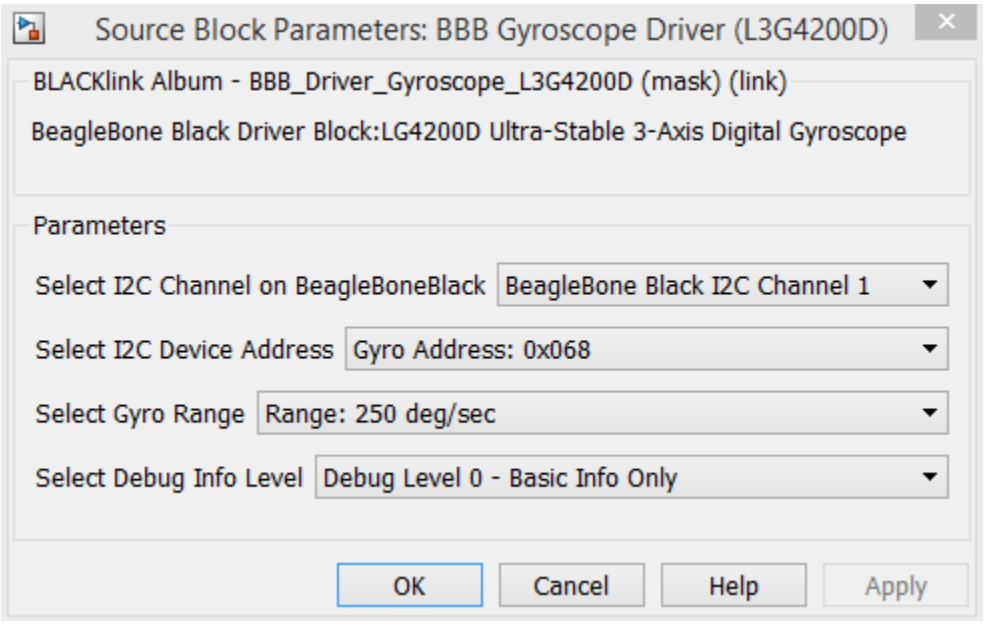


Figure 75: Gyroscope Parameter Interface

7.11 BLACKLINK LIBRARY BLOCK – DEVICE DRIVER – HMC5883L MAGNETOMETER

Block Functionality Overview

The HMC5883L Magnetometer driver block provides the user with capability to capture magnetic field data along the sensor's X, Y and Z axis. The block uses this data to calculate the overall heading as well as the resultant magnetic field strength experienced at the sensor's location. As discussed in section (4.3.1), the HMC5883L sensor is I2C enabled and therefore its magnetic field data is read directly from its registries over the I2C bus. The sensor's I2C address and bus location must be configured by the user via the block's parameters

Heading data may be referenced from either true or magnetic north depending on the user's preference. Note however, if a true north output is required, the user will have to provide the magnetic declination at the sensor's current location (e.g. -14°40' West for Montreal - the "Paris of North America") in the block's parameters.

The following image demonstrates a practical implementation in Simulink of the HMC5883L driver block calculating the its current heading based on its sensing of the magnetic field vectors.

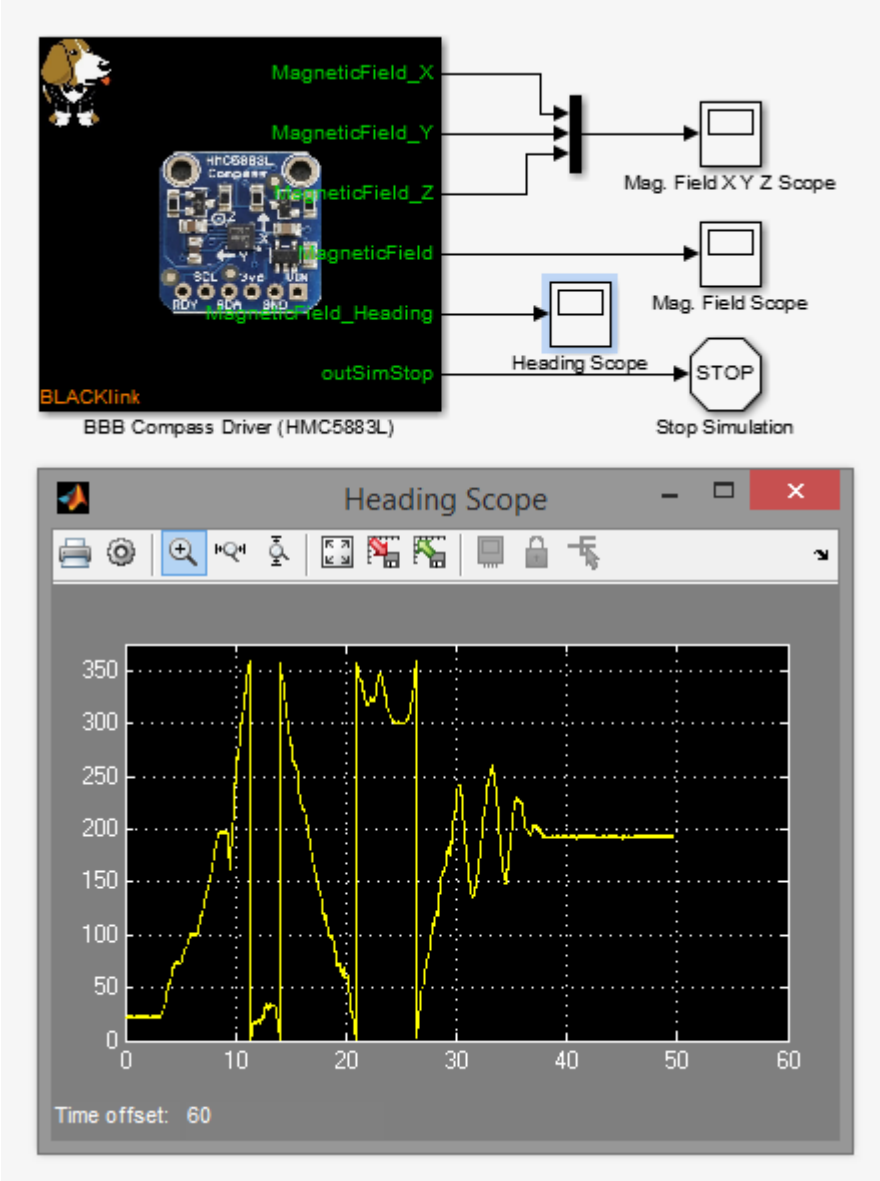


Figure 76: HMC5883L Simulink Implementation

Block Functionality Requirements

In order for the HMC5883L driver block to function correctly, the selected I2C channel must be enabled on the BeagleBone Black via the Device Tree Overlay.

The HMC5883L currently comes configured with only one possible address setting: 0x1E.This may be verified by executing an address sweep of the I2C bus from the Linux/GNU shell. The address sweep may be executed via the command “i2cdetect -r 0” or “i2cdetect -r 1” for I2C channel 1 or 2 respectively.

Block Outputs

MagneticField _X (Double) (uT): The instantaneous magnetic field strength along the sensor’s X axis measured in micro Teslas

MagneticField _Y (Double) (uT): The instantaneous magnetic field strength along the sensor's Y axis measured in micro Teslas

MagneticField _Z (Double) (uT): The instantaneous magnetic field strength along the sensor's Z axis measured in micro Teslas

MagneticField (Double) (uT): The instantaneous resultant (total magnitude) magnetic field strength experienced by the sensor and measured in micro Teslas.

MagneticField_Heading(Double)(degrees): The instantaneous heading measured in degrees from true or magnetic north (depending on block parameters).

SimStop (Boolean): The stop simulation produces a true Boolean signal if a critical error is detected in the block. This is typically caused by communication or mismatched configuration errors. The Simulink simulation will automatically stop if this port is connected to a "Stop Simulation" block. Should this port not be connected, the simulation will continue and the block will simply continue executing its task on every time step. Depending on the debug reporting level, the error may be displayed in the Window's command prompt.

Block Configurable Parameters

I2C Channel: This parameter allows the user to select which of the two I2C channels on the BeagleBone Black the magnetometer is connected to.

I2C Device Address: The user must select the I2C address of the HMC5883L. Note that currently, the magnetometer comes pre-configured with only a single address therefore only one option exists in the dropdown menu.

Number of Samples Averaged: The user may select the number samples taken and averaged for each new measurement output. The driver block will to match the user's selection and automatically configure the devices register on the first time-step of the simulation.

Output Rate: The user may select the rate at which the magnetic measurements are taken and written to output registers in continuous measurement mode. Note that a maximum rate of 160 Hz can be obtained via the use of the DRDY interrupt pin, although this functionality is not supported in the current driver version. The driver block will automatically configure the devices register to match the user's selection on the first time-step of the simulation.

Measurement Mode: The user may select the measurement configuration for the device. Default configuration applies no bias as the positive and negative pins of the resistive loads for all axes are left floating. Positive bias forces a positive current across the resistive loads of all axes. Negative bias forces a negative current across the resistive loads of all axes. The driver block will automatically configure the devices register to match the user's selection on the first time-step of the simulation.

Compass Gain: The user may select the gain settings for the compass. The gain settings should be selected based on the strength of the magnetic field at the current location. A higher gain number (GN#) should be chosen if the magnetic field strength causes an overflow (saturation) in one or more of the axis registers. The driver block will automatically configure the devices register to match the user's selection on the first time-step of the simulation.

Operation Mode: The user may select the desired operating mode for the compass. The operating mode is set during the first time-step of the Simulink block and cannot be modified until the simulation is stopped and restarted. In "Continuous" mode the device continuously performs magnetic field measurements and places the results in the

corresponding data registers. The driver block waits for the "RDY" register bit to go high and then reads, treats and outputs the data to the block's output pins. In "Single" measurement mode, the device performs a single measurement, sets the "RDY" register bit high and returns to idle mode. The measurement remains in the data output register and “RDY” remains high until the data output register is read or another measurement is performed. Note that "Single" measurement mode has not been tested in the current driver version. In "Idle" measurement mode, the device performs no measurements and hence draws less current.

Heading Mode: The user may select between "Magnetic" and "True North" heading mode. Note that the "True North" heading mode requires the magnetic declination angle for the sensor’s current location.

Heading Offset: Enter any heading offset in decimal form to be applied to the "Heading from Magnetic/True North" output pin of the driver block (for fine tuning purposes)

Declination Angle: The user must enter the magnetic declination angle for the sensor’s current location as a decimal. Note that the declination angle is only used if heading mode is set to "True North".

Enable Compass Self-Test: A self-test mode may be enabled. If enabled, the compass will perform a "Self-Test" on the first time-step of the simulation. During the self-test the sensor is internally excited with a nominal magnetic field (in either positive or negative bias configuration). This field is then measured and reported. An internal current source generates DC current (about 10 mA) from the VDD supply. This DC current is applied to the offset straps of the magneto-resistive sensor, which creates an artificial magnetic field bias on the sensor. The difference between this measurement and that of the ambient field will be put in the data output register for each of the three axes. The driver block automatically calculates the acceptable limits for each axis based on the gain used for the self-test. Should the self-test fail, the simulation will be automatically stopped and a diagnostic message displayed in the Window’s command prompt. If not, the compass will be placed into the set operation mode and the simulation will continue.

Self-Test Gain: Select the gain for the self-test. As a guideline, the "Self-Test" gain should be chosen as one level less than the normal operation gain value set in Compass Gain parameter.

Debug Level: This parameter allows the user to set the level of diagnostic information written to the Window’s command prompt and may prove useful for debugging purposes. The user may select between having no debug information to having warnings, critical errors, block specific diagnostics and program flow information written to the command prompt. Note that each subsequent debug level adds its respective information to debug levels found below it.

Block Configuration Interface

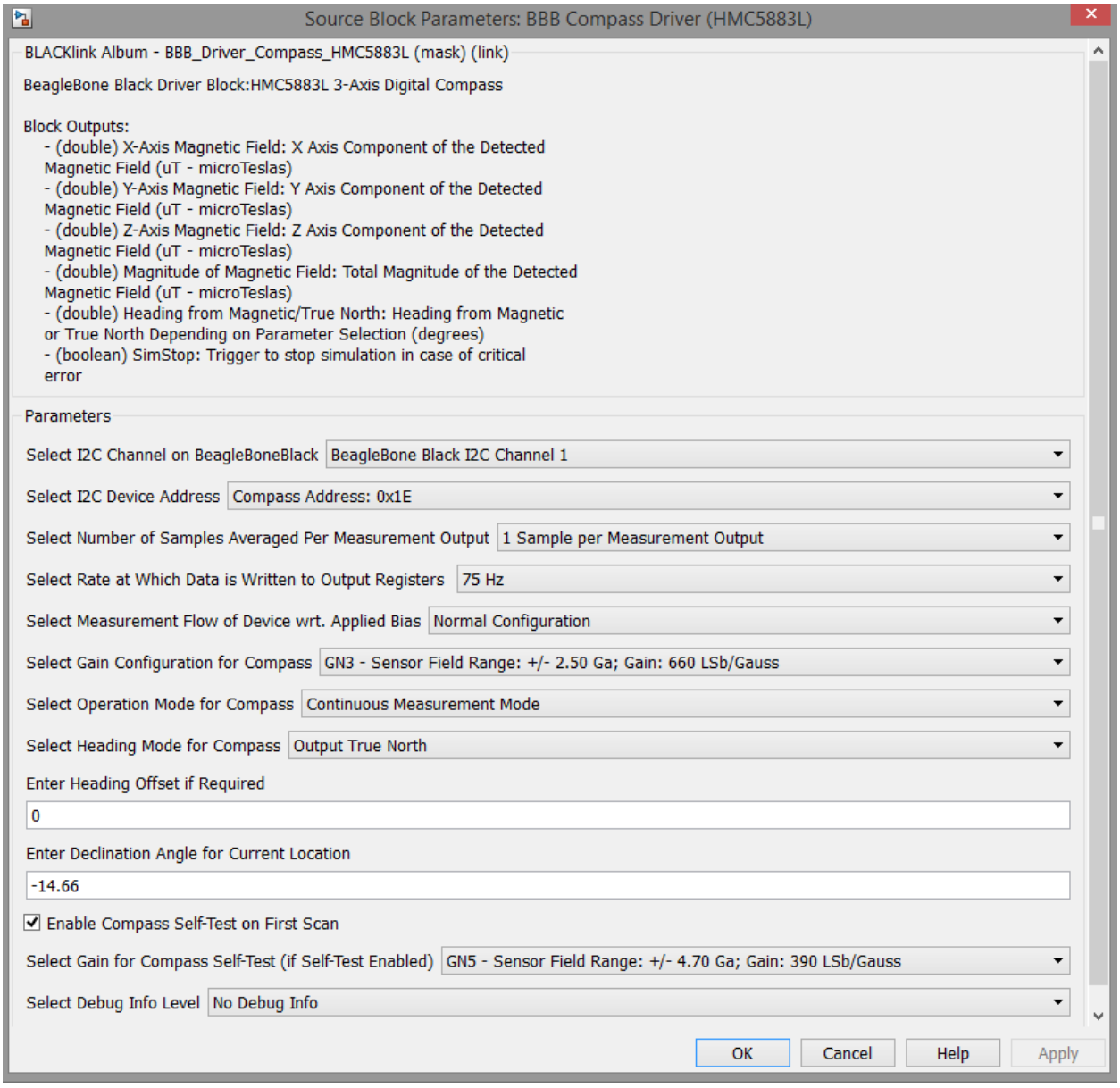


Figure 77: HMC5883L Configuration Interface

8.0 ATTITUDE ESTIMATION VIA GRADIENT DESCENT ALGORITHM

8.1 ALGORITHM MOTIVATION

Attitude estimation for the IMU and AHRS systems developed during the course of this project employed the use of a relatively new descent gradient algorithm [2] for sensor fusion. Initial research revealed a small but growing interest in this algorithm within the open source UAV autopilot community over the well know heavy-weights such as the Kalman and Extended Kalman filter approaches. Reasons for this increased interest may be distilled into three main factors. The first, and most prevalent, appears to be the low computational load required in the implementation of this orientation filter. This fact proves especially attractive for embedded systems where microcontroller and microprocessor power, although increasing powerful, are still behind what may be found in a conventional computer. Lower required computational power translates into lighter packages and smaller footprints opening the integration possibilities to small UAVs and even wearable technology. In line with the first attractive factor, the second stems from the ability to obtain higher estimation accuracy at lower sampling rates. This is due to the fact an actual analytical solution has been derived for the descent gradient algorithm as opposed to the linear regression iterations required by the Kalman filter approach. The final pillar supporting the popularity of the approach is the employment of quaternions in order to avoid "Gimbal Lock" type singularities that are prevalent with Euler angle attitude representation.

8.2 QUATERNION REPRESENTATION

The quaternion representation of attitude orientation in three dimensional space may be given by a four dimensional complex number. The conceptual motivation to this approach is to represent any arbitrary rotation of a frame B relative to frame A as a rotation of angle θ around a specific axis ${}^{A}\hat{r}$ defined in the A frame. A graphical representation is presented in the following figure:

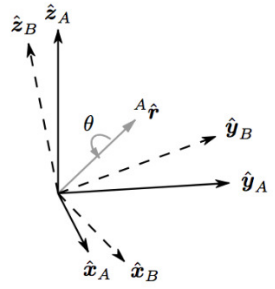


Figure 78: Quaternion Frame Rotation

The quaternion representation of the above frame rotation may be given as:

$$ {}^{A}_{B}\hat{q} = [q_1 \quad q_2 \quad q_3 \quad q_4] = \left[\cos\frac{\theta}{2} \quad -\mathrm{r_x}\cos\frac{\theta}{2} \quad -\mathrm{r_y}\cos\frac{\theta}{2} \quad -\mathrm{r_z}\cos\frac{\theta}{2}\right] \tag{8.1} $$

The quaternion conjugate representing the inverse relationship, that is to say, the orientation of frame A with respect to frame B, may be given as:

$$ {}^{A}_{B}\hat{q}^{*} = {}^{B}_{A}\hat{q} = [q_1 \quad -q_2 \quad -q_3 \quad -q_4] \tag{8.2} $$

For compound orientations representing two independent rotations such as ${}^{A}_{B}\hat{q}$ and ${}^{B}_{C}\hat{q}$ the rotation ${}^{A}_{C}\hat{q}$ may be calculated via the quaternion product denoted by $\otimes$ and represented as:

$$ {}^{A}_{C}\hat{q} = {}^{B}_{C}\hat{q} \otimes {}^{A}_{B}\hat{q} \tag{8.3} $$

The mechanics of the quaternion product may be demonstrated as:

$$ a \otimes b \neq b \otimes \text{a} \tag{8.4} $$

$$ a \otimes b = [a_1 \quad a_2 \quad a_3 \quad a_4] \otimes [b_1 \quad b_2 \quad b_3 \quad b_4] \tag{8.5} $$

$$ a \otimes b = \begin{bmatrix} a_1b_1 - a_2b_2 - a_3b_3 - a_4b_4 \\ a_1b_2 + a_2b_1 + a_3b_4 - a_4b_3 \\ a_1b_3 - a_2b_4 + a_3b_1 - a_4b_1 \\ a_1b_4 + a_2b_3 - a_3b_2 + a_4b_1 \end{bmatrix} \tag{8.6} $$

Rotation of a three dimensional vector v, via a quaternion may be calculated as:

$$ {}^{B}v = {}^{A}_{B}\hat{q} \otimes {}^{A}v \otimes {}^{A}_{B}\hat{q}^{*} \tag{8.7} $$

Here, ${}^{A}v$ and ${}^{B}v$ are representations of the same vector as described from frames A and B respectively. The vector representation may be cast into a four element row vector as shown below:

$$ v = [0 \quad v_x \quad v_y \quad v_z] \tag{8.8} $$

A quaternion rotation such as ${}^{A}_{B}\hat{q}$ may be cast as a rotational matrix ${}^{A}_{B}R$ where:

$$ {}^{A}_{B}R = \begin{bmatrix} 2q_1^2 - 1 + 2q_2^2 & 2(q_2q_3 + q_1q_4) & 2(q_2q_4 - q_1q_3) \\ 2(q_2q_3 - q_1q_4) & 2q_1^2 - 1 + 2q_3^2 & 2(q_3q_4 - q_1q_2) \\ 2(q_2q_4 + q_1q_3) & 2(q_3q_4 - q_1q_2) & 2q_1^2 - 1 + 2q_4^2 \end{bmatrix} \tag{8.9} $$

The Euler angle casting of the quaternion rotation ${}^{A}_{B}\hat{q}$ may be given by the following equations:

$$ \psi = atan2(2q_2q_3 - 2q_1q_4, 2q_1^2 + 2q_2^2 - 1) \tag{8.10} $$

$$ \theta = -\sin^{-1}(2q_2q_4 + 2q_1q_3) \tag{8.11} $$

$$ \phi = atan2(2q_3q_4 - 2q_1q_2, 2q_1^2 + 2q_4^2 - 1) \tag{8.12} $$

8.3 EXTRACTION OF ORIENTATION FROM GYROSCOPE ANGULAR RATES

The angular velocity provided by the gyroscope sensor may be represented in the following vector format:

$$ {}^{S}\omega = [0 \quad \omega_x \quad \omega_y \quad \omega_z] \tag{8.13} $$

From the above representation of the angular rates, the quaternion representation of the rate of change of the earth frame relative to the sensor frame may be calculated as:

$$ {}^{S}_{E}\dot{q} = \frac{1}{2} {}^{S}_{E}\hat{q} \otimes {}^{S}\omega \tag{8.14} $$

Hence the orientation of the earth frame relative to the sensor frame at a given point in time may be computed via the integration of equation (8.14) in discrete time. The discrete time representation of equation (8.14) may be expressed as:

$$ {}^{S}_{E}\dot{q}_{\omega,t} = \frac{1}{2} {}^{S}_{E}\hat{q}_{est,t-1} \otimes {}^{S}\omega_{t} \tag{8.15} $$

The numerical integration of the above equation yields:

$$ {}^{S}_{E}q_{\omega,t} = {}^{S}_{E}\hat{q}_{est,t-1} + {}^{S}_{E}\dot{q}_{\omega,t}\Delta t \tag{8.16} $$

Equation (8.16) now represents, in quaternions, the orientation of the sensor frame with respect to the earth frame as deduced by integration of the gyroscopic angular rates.

8.4 EXTRACTION OF ORIENTATION FROM FIELD VECTOR OBSERVATIONS

Before developing explicit solutions for orientation based on accelerometer or magnetometer readings, the general case of determining the rotation of the sensor frame with respect the earth frame given the sensor's field measurements is explored. Conceptually, it may be stated that if the direction of the earth field in the earth frame is known (e.g. gravity, magnetic north), measurement of the field's direction with respect to the sensor frame can be manipulated to determine the orientation of sensor frame with respect to the earth frame. Unfortunately, this approach will not generate a unique solution as infinite orientations are possible when the sensor's axis is perfectly parallel with the measured earth field. In such a case, the sensor's exact orientation round the axis in-line with the earth field cannot be determined. To rectify this, the orientation problem is cast as an optimization problem. The goal is then to determine the orientation of the sensor frame relative to the earth frame, ${}^{S}_{E}\hat{q}$ such that it aligns a predefined reference direction in the earth field, ${}^{E}\hat{d}$ with the measured field components in the sensor frame ${}^{S}\hat{s}$.

Therefore, the optimization problem to derive ${}^{S}_{E}\hat{q}$ may be represented as:

$$ \min_{{}^{S}_{E}\hat{q}\in\Re^4} f({}^{S}_{E}\hat{q}, {}^{E}\hat{d}, {}^{S}\hat{s}) \tag{8.17} $$

Where the objective function is defined as:

$$ f({}^{S}_{E}\hat{q}, {}^{E}\hat{d}, {}^{S}\hat{s}) = {}^{S}_{E}\hat{q}^{*} \otimes {}^{E}\hat{d} \otimes {}^{S}_{E}\hat{q} - {}^{S}\hat{s} \tag{8.18} $$

The vector representation of the sensor frame orientation, reference direction and sensor measurements are defined respectively as:

$$ {}^{S}_{E}\hat{q} = [q_1 \quad q_2 \quad q_3 \quad q_4] \tag{8.19} $$

$$ {}^{E}\hat{d} = [0 \quad d_x \quad d_y \quad d_z] \tag{8.20} $$

$$ {}^{S}\hat{s} = [0 \quad s_x \quad s_y \quad s_z] \tag{8.21} $$

As the name of the proposed filter would suggest, the gradient descent algorithm is chosen as the optimization algorithm of choice given its simplicity, ease of implementation and low computational cost. Therefore, the estimation of the sensor orientation with respect to the earth frame after k iterations with step-size μ and an initial orientation guess of ${}^{S}_{E}\hat{q}_0$ is calculated as follows:

$$ {}^{S}_{E}\hat{q}_{k+1} = {}^{S}_{E}\hat{q}_k - \mu \frac{\nabla f\left({}^{S}_{E}\hat{q},\ {}^{E}\hat{d},\ {}^{S}\hat{s}\right)}{\left\|\nabla f\left({}^{S}_{E}\hat{q},\ {}^{E}\hat{d},\ {}^{S}\hat{s}\right)\right\|}, k = 0,1,2 \dots n \tag{8.22} $$

The gradient of the solution surface may be computed as:

$$ \nabla f\left({}^{S}_{E}\hat{q},\ {}^{E}\hat{d},\ {}^{S}\hat{s}\right) = J^T\left({}^{S}_{E}\hat{q}_k,\ {}^{E}\hat{d}\right) f\left({}^{S}_{E}\hat{q}_k,\ {}^{E}\hat{d},\ {}^{S}\hat{s}\right) \tag{8.23} $$

The vector representation of the objective function and its corresponding Jacobian may be expressed respectively as:

$$ \begin{aligned} & f\left({}^{S}_{E}\hat{q},\ {}^{E}\hat{d},\ {}^{S}\hat{s}\right) \\ &= \begin{bmatrix} 2d_x\left(\frac{1}{2} - q_3^2 - q_4^2\right) + 2d_y(q_1q_4 + q_2q_3) + 2d_z(q_2q_4 - q_1q_3) - s_x \\ 2d_x(q_2q_3 - q_1q_4) + 2d_y\left(\frac{1}{2} - q_2^2 - q_4^2\right) + 2d_z(q_1q_2 - q_3q_4) - s_y \\ 2d_x(q_1q_3 - q_2q_4) + 2d_y(q_3q_4 - q_1q_2) + 2d_z\left(\frac{1}{2} - q_2^2 - q_3^2\right) - s_z \end{bmatrix} \end{aligned} \tag{8.24} $$

$$ \begin{aligned} & J\left({}^{S}_{E}\hat{q}_k,\ {}^{E}\hat{d}\right) \\ &= \begin{bmatrix} 2d_yq_4 - 2d_zq_3 & 2d_yq_3 + 2d_zq_4 & -4d_xq_3 + 2d_yq_2 - 2d_zq_1 & -4d_xq_4 + 2d_yq_1 + 2d_zq_2 \\ -2d_xq_4 + 2d_zq_2 & 2d_xq_3 - 4d_yq_2 + 2d_zq_1 & 2d_xq_2 + 2d_zq_4 & -2d_xq_1 - 4d_yq_4 + 2d_zq_3 \\ 2d_xq_3 - 2d_yq_2 & 2d_xq_4 - 2d_yq_1 - 4d_zq_2 & 2d_xq_1 + 2d_yq_4 - 4d_zq_3 & 2d_xq_2 + 2d_yq_3 \end{bmatrix} \end{aligned} \tag{8.25} $$

Hence, equations (8.22 – 8.25) define the general solution for determining the earth orientation with respect to the sensor frame given a predefined known reference direction in the earth field.

8.5 EXTRACTION OF ORIENTATION FROM ACCELEROMETER GRAVITY VECTOR OBSERVATIONS

The general equations (8.22 – 8.25) for determining the sensor frame orientation relative to the earth frame may now be applied to the case of the gravity field sensed by the accelerometer. Assuming (quite confidently) that the orientation of the gravity vector relative to the earth frame is known and defines the earth frame z-axis, then the reference direction for the normalized earth filed is given as:

$$ {}^{E}\hat{d} = [0 \quad 0 \quad 0 \quad 1] \tag{8.26} $$

The *normalized* accelerometer measurements are represented as:

$$^{S}\hat{a} = [0 \quad a_x \quad a_y \quad a_z] \tag{8.27}$$

Substituting the above two equations in equations (8.20 & 8.21) yields the following versions of the objective function and its Jacobian yield:

$$f_g\left(\,{}^{S}_{E}\hat{q},\ {}^{S}\hat{a}\right) = \begin{bmatrix} 2d_z(q_2q_4 - q_1q_3) - a_x \\ 2d_z(q_1q_2 - q_3q_4) - a_y \\ 2d_z\left(\frac{1}{2} - q_2^2 - q_3^2\right) - a_z \end{bmatrix} \tag{8.28}$$

$$J_g\left(\,{}^{S}_{E}\hat{q}\right), = \begin{bmatrix} -2q_3 & 2q_4 & -2q_1 & 2q_2 \\ 2q_2 & 2q_1 & 2q_4 & 2q_3 \\ 0 & -4q_2 & -4q_3 & 0 \end{bmatrix} \tag{8.29}$$

8.6 EXTRACTION OF ORIENTATION FROM COMPASS MAGNETIC FIELD VECTOR OBSERVATIONS

Applying the general equations (8.22 – 8.25) to the case of the magnetometer, where the earth's magnetic field is assumed to have only a horizontal and vertical component yields the following representation of normalized earth filed and sensor measurements:

$$^{E}\hat{b} = [0 \quad b_x \quad 0 \quad b_z] \tag{8.30}$$

$$^{S}\hat{m} = [0 \quad m_x \quad m_y \quad m_z] \tag{8.31}$$

Again, substituting the above two equations in equations (8.20 & 8.21) yields the following versions of the objective function and its Jacobian:

$$f_b\left(\,{}^{S}_{E}\hat{q},\ {}^{E}\hat{b},\ {}^{S}m\right) = \begin{bmatrix} 2b_x\left(\frac{1}{2} - q_3^2 - q_4^2\right) + 2b_z(q_2q_4 - q_1q_3) - m_x \\ 2b_x(q_2q_3 - q_1q_4) + 2b_z(q_1q_2 + q_3q_4) - m_y \\ 2b_x(q_1q_3 + q_2q_4) + 2b_z\left(\frac{1}{2} - q_2^2 - q_3^2\right) - m_z \end{bmatrix} \tag{8.32}$$

$$J_b\left(\,{}^{S}_{E}\hat{q},\ {}^{E}\hat{b}\right) = \begin{bmatrix} -2b_zq_3 & 2b_zq_4 & -4b_xq_3 - 2b_zq_1 & -4b_xq_4 + 2b_zq_2 \\ -2b_xq_4 + 2b_zq_2 & 2b_xq_3 + 2b_zq_1 & 2b_xq_2 + 2b_zq_4 & -2b_xq_1 + 2b_zq_3 \\ 2b_xq_3 & -2b_xq_4 - 4b_zq_2 & 2b_xq_1 - 4b_zq_3 & 2b_xq_2 \end{bmatrix} \tag{8.33}$$

8.7 EXTRACTION OF UNIQUE ORIENTATION SOLUTION

In order to determine the solution *point* of the objective function, the solutions of both the accelerometer and magnetometer must be combined such that: (assuming$b_x \neq 0$)

$$f_{g,b}\left({}^{S}_{E}\hat{q},\ {}^{S}\hat{a},\ {}^{E}\hat{b},\ {}^{S}\hat{m}\right)=\begin{bmatrix} f_g\left({}^{S}_{E}\hat{q},\ {}^{S}\hat{a}\right) \\ f_b\left({}^{S}_{E}\hat{q},\ {}^{E}b,\ {}^{S}m\right)\end{bmatrix} \tag{8.34}$$

$$J\left({}^{S}_{E}\hat{q},\ {}^{E}\hat{b}\right)=\begin{bmatrix} J_g^T\left({}^{S}_{E}\hat{q}\right), \\ J_b^T\left({}^{S}_{E}\hat{q},\ {}^{E}\hat{b}\right)\end{bmatrix} \tag{8.35}$$

Therefore, the expression for the iterative calculation of the estimated orientation based on the previous time-step estimate, is given as:

$${}^{S}_{E}\hat{q}_{\nabla,\mathrm{t}}={}^{S}_{E}\hat{q}_{est,t-1}-\mu_t\frac{\nabla f}{\|\nabla f\|} \tag{8.36}$$

Where, for the case of an IMU (accelerometer only):

$$\nabla f=J_g^T\left({}^{S}_{E}\hat{q}_{est,t-1}\right)f_g\left({}^{S}_{E}\hat{q}_{est,t-1},\ {}^{S}\hat{a}\right) \tag{8.37}$$

And for the case of the AHRS (accelerometer and magnetometer):

$$\nabla f=J_b^T\left({}^{S}_{E}\hat{q}_{est,t-1},\ {}^{E}\hat{b}\right)f_b\left({}^{S}_{E}\hat{q}_{est,t-1},\ {}^{E}b,\ {}^{S}m\right) \tag{8.38}$$

Note that the step-size for the gradient decent iteration may be given by:

$$\mu_t=\alpha\left\|{}^{S}_{E}\dot{q}_{\omega,t}\right\|\Delta t,\alpha>1 \tag{8.39}$$

Where Δt is the sampling period, ${}^{S}_{E}\dot{q}_{\omega,t}$ is the orientation rate sensed by the gyroscope and α is a constant aimed at counteracting the effects of noise on the accelerometer and magnetometer.

8.8 FILTER FUSION ALGORITHM

The sensory data extracted from the gyroscope, accelerometer and/or magnetometer must be combined to obtain a single orientation estimate of the sensor frame relative to the earth frame. This is carried out by the fusion of the orientation estimated by the numerical integration of the gyroscope's angular rate readings and the orientation estimated by the gradient descent solution of the objective function stated in section (8.4) and (8.7) respectively. This calculation is carried out by the weighted linear combination of the two estimates mentioned above and may be expressed as:

$${}^{S}_{E}\hat{q}_{est,t}=\gamma_t{}^{S}_{E}\hat{q}_{\nabla,\mathrm{t}}+(1-\gamma_t){}^{S}_{E}q_{\omega,t},0\leq\gamma_t\leq1 \tag{8.40}$$

For optimal fusion, the weighting factor may be chosen as:

$$\gamma_t=\frac{\beta}{\frac{\mu}{\Delta t}+\beta} \tag{8.41}$$

Where $\frac{\mu}{\Delta t}$ is a representation of the rate of convergence of ${}^{S}_{E}\hat{q}_{\nabla}$ and β represents the divergence rate of ${}^{S}_{E}q_{\omega}$.

The objective of the fusion algorithm is to improve the orientation estimate by using the angular rate integral ($_E^S q_\omega$) provided by the gyroscope to filter out high frequency errors present in the gradient decent ($_E^S \hat{q}_{\nabla,t}$) estimate. The gradient descent estimate is used to correct the accumulated error from the integral and to provide convergence from initial conditions.

In order to implement attitude estimation, both the IMU and AHRS algorithms were adapted into C-code and embedded within an S-Function block. The full S-Function code is available in the appendix.

9.0 REAL TIME HARDWARE-IN-LOOP IMU & AHRS SIMULATION

9.1 IMU IMPLEMENTATION

A hardware implementation of an inertial measurement unit was carried out by employing the diver blocks developed for the L3G4200D gyroscope and ADXL300 accelerometer. An S-Function block was created with the IMU gradient descendent algorithm translated into C-Code. The Gradient Descent IMU block accepts six inputs which include the three angular rate inputs from the gyroscope (rad/s) and three acceleration components from the accelerometer (g). The block outputs the estimated orientation via quaternion values $q_1, q_2, q_3 \; and \; q_4$. A slightly cluttered version of the Simulink model is shown below.

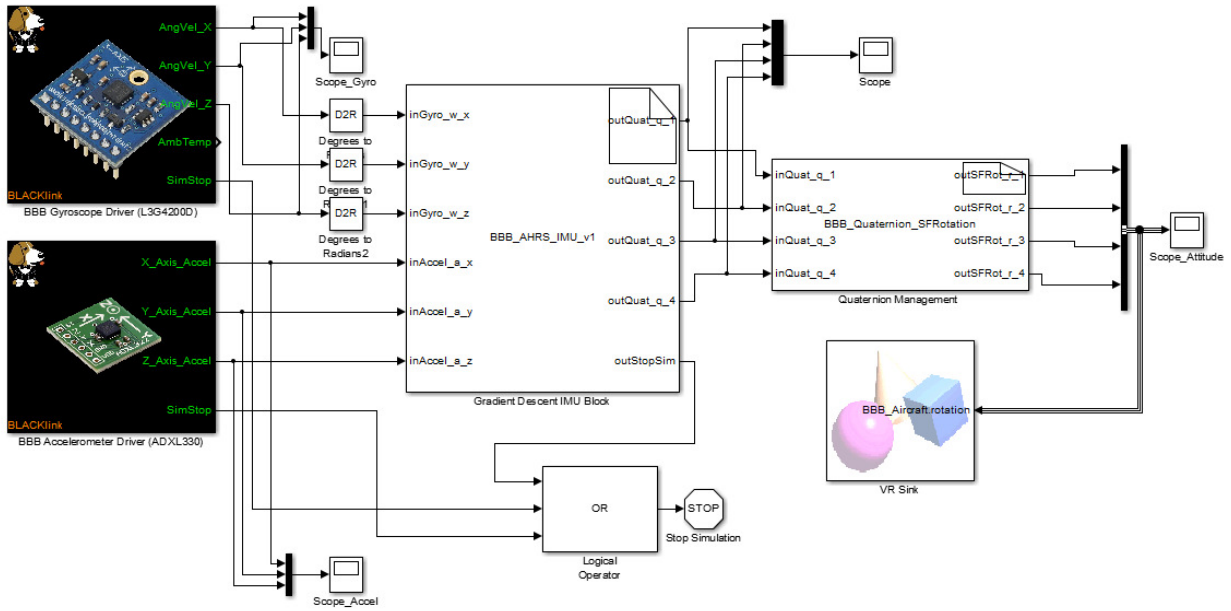


Figure 79: IMU Simulink Model

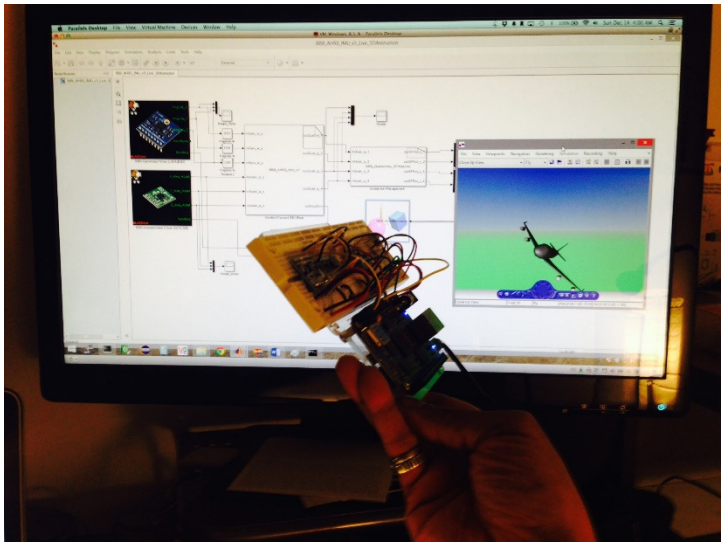


Figure 80: IMU Runtime

9.2 AHRS IMPLEMENTATION

A hardware implementation of an attitude and heading reference system was carried by out employing the diver blocks developed for the L3G4200D gyroscope, ADXL300 accelerometer and HMC5883L magnetometer. An S-Function block was created and the AHRS gradient descendent algorithm translated into C-Code. The Gradient Descent AHRS block accepts nine inputs which include the three angular rate inputs from the gyroscope (rad/s), three acceleration components from the accelerometer (g) and thee magnetic field components (micro Tesla). The block outputs the estimated orientation via quaternion values $q_1, q_2, q_3 \; and \; q_4$. A heavily cluttered version of the Simulink model is shown below.

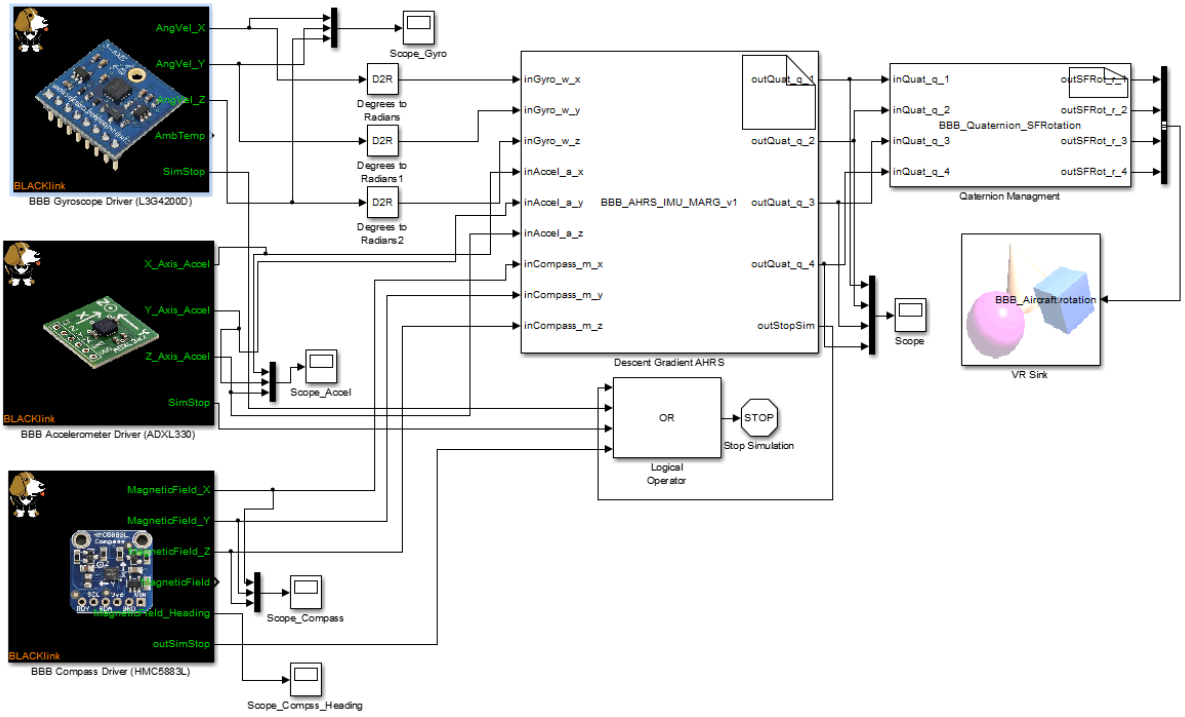


Figure 82: AHRS Simulink Model

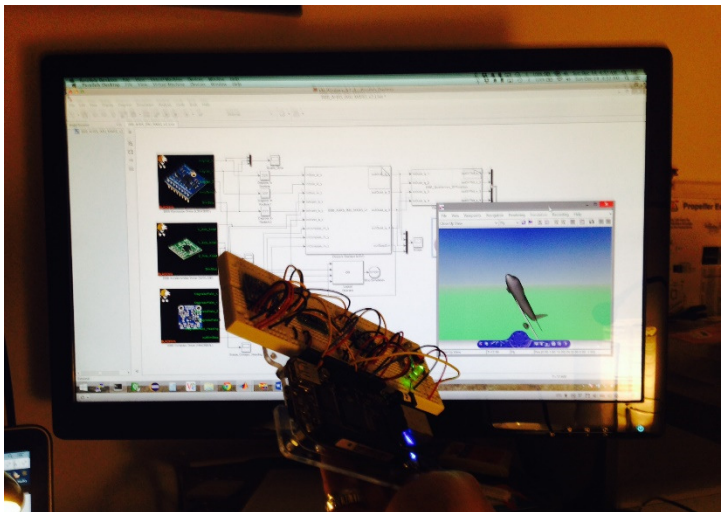


Figure 81: AHRS - Runtime

9.3 3D ESTIMATED ATTITUDE VISUALIZATION

A 3D animation is added to both IMU and AHRS systems in order to better visualize the performance of the attitude estimation of the gradient descent algorithm. Note that an additional quaternion management block was required given that the quaternion structuring used within the estimation algorithm does not match with that expected from the rotational node of the aircraft imported into the virtual world developed via VR Builder. The following image is of the 3D Boeing 747 model used in the virtual reality visualization. Note that the current quaternion configuration is such that the sensors have their positive y-axis pointing through the nose of the aircraft, their positive x-axis extending over the aircraft's starboard wing while their positive z-axes point upwards.

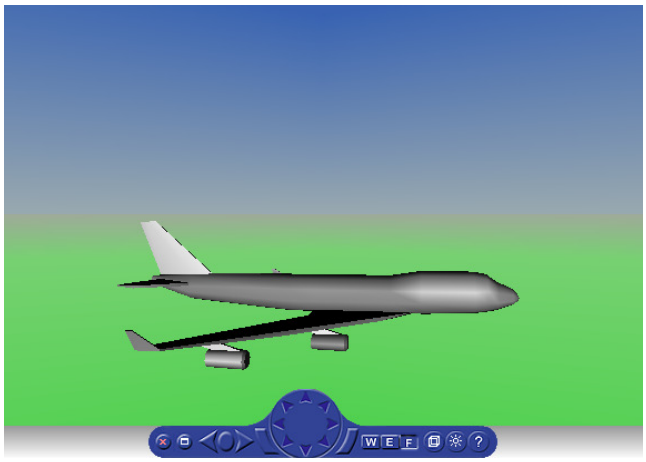


Figure 83: Boeing 747 3D Model in the developed VR World

9.4 IMU/AHRS USER INTERFACE

For improved visualization purposes, a full-fledged user interface was developed during the course of this project. Its primary mandate is to serve as a diagnostic tool in monitoring the performance of the IMU or AHRS gradient decent attitude estimation algorithm. Note the following features:

- 3D – MulitView – The HMI displays two separate views of the current attitude of the aircraft. This includes a front view for accurate visualization of pitch and roll as well as a top view to better capture the aircraft's yaw dynamics.
- Accelerometer Scope – All three acceleration components are visualized in real time on the accelerometer plot. This allows for an implicit understanding of the influence of the sensors current measurements on the displayed estimated attitude, especially when acceleration forces other than the gravitational force are exerted on the sensor. Note that the plot can be enable or disabled during runtime.
- Gyroscope Scope – All three angular rates are displayed in real time on the gyro plot. Again, this allows for tracking of gyroscope drift as well as a better appreciation of the noise experienced by the sensor. Note that the plot can be enable or disabled during runtime.
- Simulation Control – The interface allows the user to build, start and stop the external mode simulation via the control buttons in the Simulation Control Panel.
- Simulation State Monitoring – In order to capture data from the sensors, "event listeners" are embedded into the Simulink model. Unfortunately this causes the simulation to lag meaning physical inputs made to the sensors on the prototyping board were not immediately displayed on the user interface. The "Simulation Time" and "System Time" indicators as well as the "Lag Time" progress bar helps to visualize

this by indicating the time difference between the system and simulation. Disabling the two scope plots reduces the simulation load considerably allowing the simulation to eventually catch up to real time.

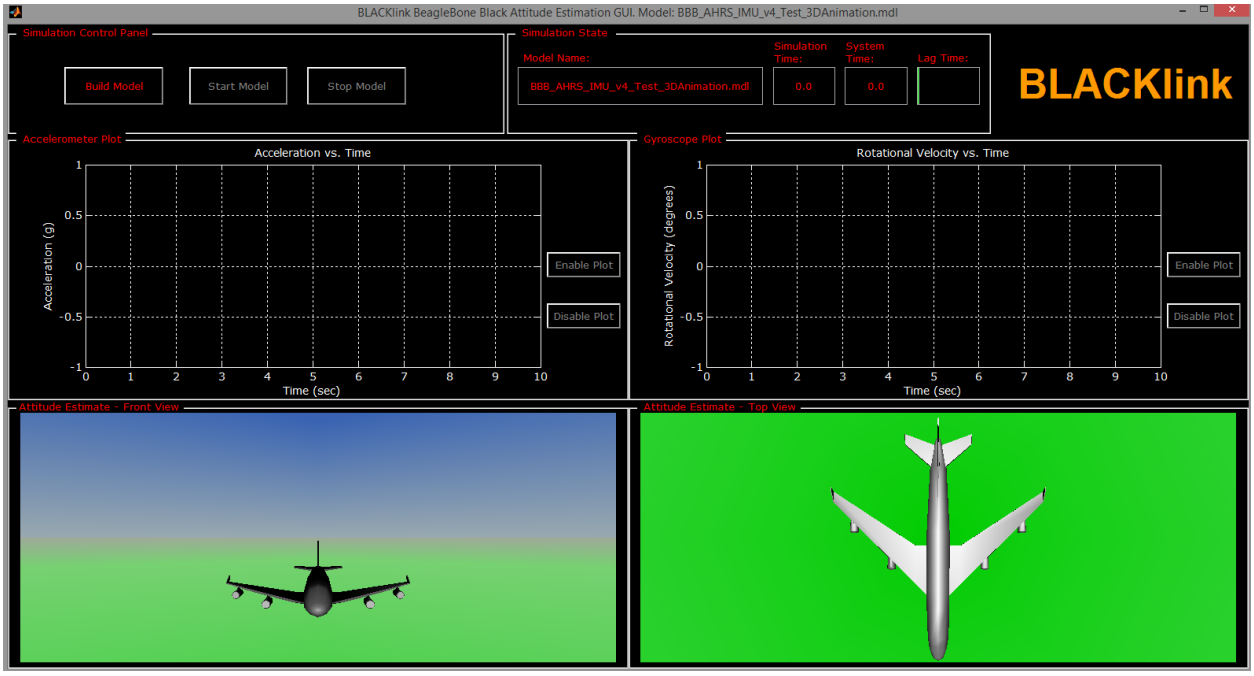


Figure 84: IMU/AHRS User Interface

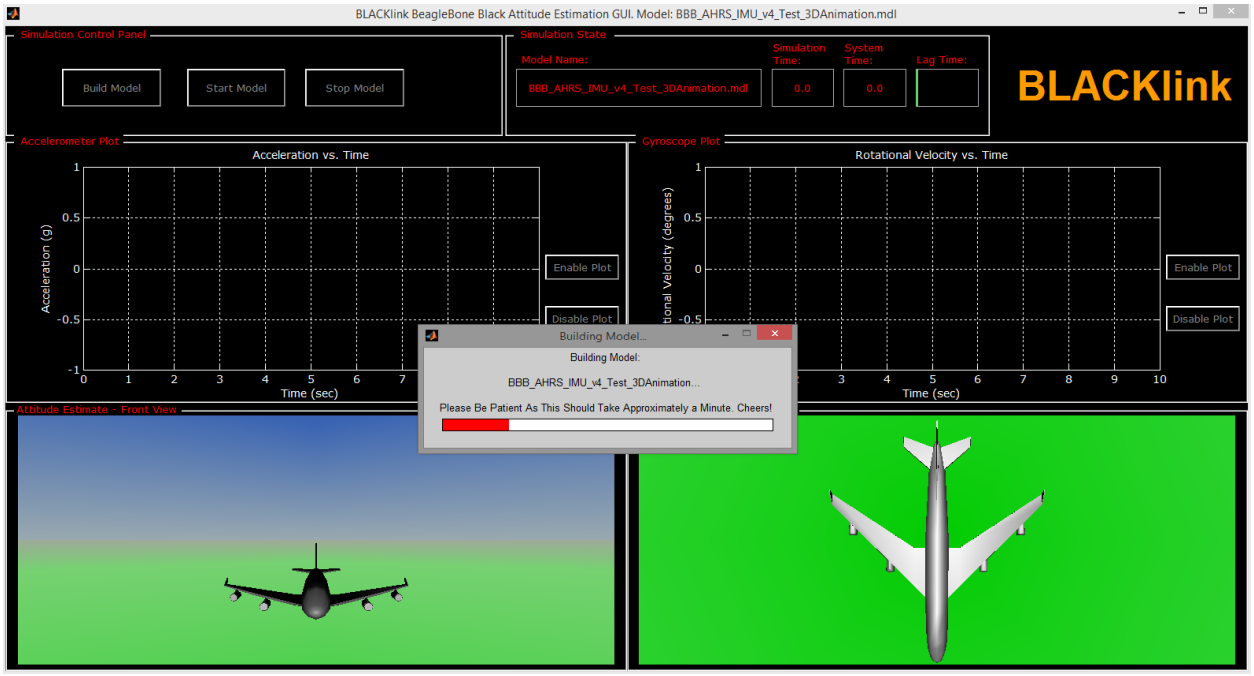


Figure 85: IMU/AHRS User Interface - Building Model

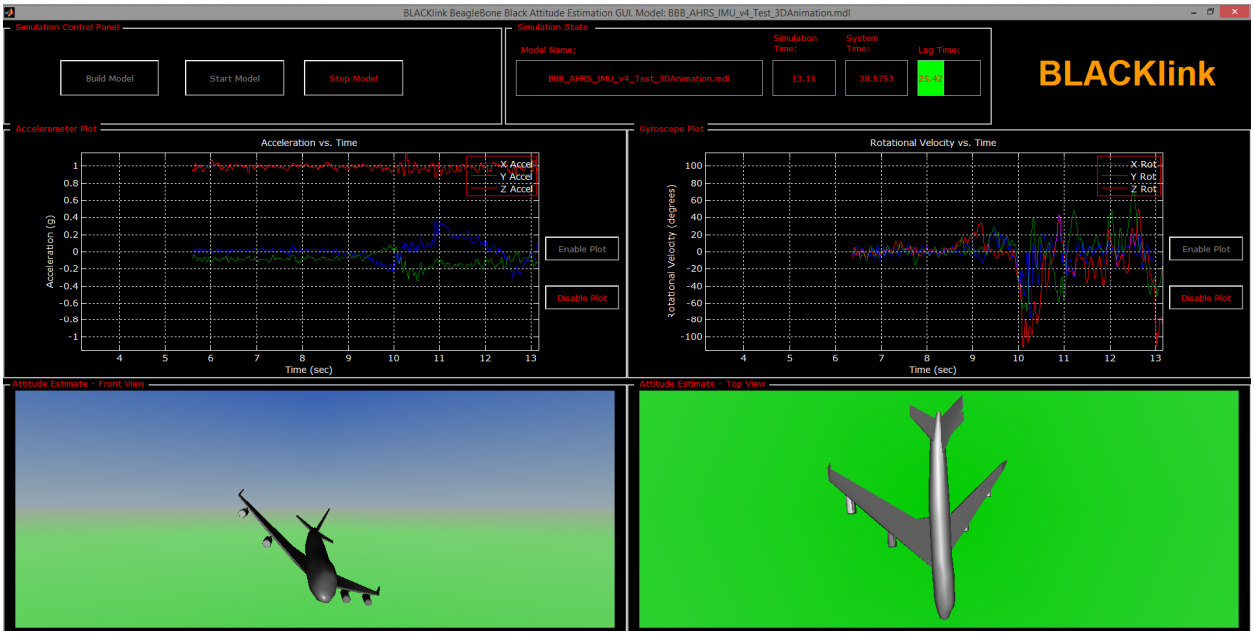


Figure 86: IMU/AHRS User Interface - Runtime

10.0 EVALUATION OF THE RAPID PROTOTYPING SYSTEM

The initial testing of the BeagleBone Black as a prototyping platform for Matlab/Simulink proved encouraging. Proper mounting of the BeagleBone Black in order to function with the current version of Simulink provided a seamless experience in building, downloading, compiling and launching programs. Feedback information such as sensor data through viewed through scopes in the Simulink Model appeared extremely responsive, despite a relatively large step time.

Development of driver blocks for interfacing with the Linux kernel's virtual file system ("sysfs") displayed rapid, consistent and reliable performance. System lag originating from the BeagleBone Black was virtually a non-issue. It is worth noting however that throughout this project, the simulation step time has been consistently set to 0.01 seconds and was rarely pushed further. A smaller step time may be necessary if, for example, the prototyped control design targets a high bandwidth, inherently unstable system such as a small quad rotor, at which point the limitations of this platform may be better tested.

Communication between the host computer and the BeagleBone Black was carried out primarily over USB cable which was capable of handling data transfer requirements with ease for the applications tested. Communication speed and reliability may suffer if a non-tethered solutions is required as was discovered while testing communication over Wi-Fi.

Unfortunately, actuation control has yet to be tested at the time of writing although a pulse width modulation driver block is currently under development. Investigation in the BeagleBone Black's programmable real time units (PRU's) may prove useful especially for control systems requiring extremely fast motor actuation. PRUs are basically 32 bit reduced instruction set (RISC) microprocessors running at 200MHz free from the operating system and therefor runs in "hard real-time", dedicated to a specific task which is configurable by the OS. These PRUs are also optimized for software implementations of peripherals. Currently Linux drivers exist which map the PRUs control registries to user space therefor it should be possible to develop a Simulink driver block to take advantage of this functionality. This is likely an interesting topic to explore to increase the applicability of the prototyping platform.

Finally, it should be noted that as of Oct 2014, The MathWorks has released support for the BeagleBone Black. Note however that this support exists only for Matlab versions 2014b and later. At the time of writing, only the USRLED and GPIO blocks provided by the newly released MathWorks software overlap with the blocks developed over the course of this project and therefore the majority of these developed blocks remain relevant. More importantly, the procedure for Simulink block development elaborated in this document remains intact.

11.0 CONCLUSION

Given its processing power, numerous and flexible I/O capabilities as well as its affordable price, the BeagleBone Black easily distinguishes itself from competitors such as the Arduino and the Raspberry Pi microcontrollers, especially for computationally heavy applications. As such it is only a matter of time before the platform becomes unanimous in university control engineering curriculums and graduate research and development labs.

This project has aimed to open up the BeagleBone Black for use with the ubiquitous Matlab/Simulink software as it already serves at the predominant tool for control system design, simulation and analysis. Over the course of this project, a BeagleBone Black (revision B) was mounted with the Ubuntu 13.10 Linux distribution and configured for use with the pre-existing support software for the *BeagleBoard*. With the initial functionality confirmed, research into the development of custom made C-coded S-Function blocks was carried out yielding the procedure discussed in this report. Unlike the Arduino, there are no pre-existing APIs or libraries to leverage for interacting with hardware on the BeagleBone. Instead the core functionality of the developed driver blocks involve direct C-code interaction with the Linux kernel via the virtual file system "sysfs" which exports requested hardware information from kernel-space to user-space for the appropriate manipulation. Through this approach several Simulink blocks were developed to execute standardized functions. Device specific driver blocks were also developed. Masking was applied to all blocks in order to standardize appearance and improve the user experience. Each block consists of user-configurable parameters aimed at increasing flexibility and applicability to a multitude of applications.

As an initial test of the BeagleBone Black and Simulink as a viable rapid-prototyping platform, IMU and AHRS models were developed. A relatively new, computationally efficient algorithm was implemented for sensor fusion. The algorithm casts attitude estimation as an optimization problem and employs gradient descent to solve the objective function. This yielded encouraging results, even at the low sampling rate of 10Hz.

Finally, a user interface was developed to allow for more thorough performance analysis of the attitude estimation platforms.

Please note that all code developed for this project will be posted on the author's GitHub page within the Blacklink repository. This may be found via the following link: https://github.com/rdustinkahawita/BLACKlink.git

Also note that videos demonstrating the functionality of the BeagleBone Black on Simulink are available on the author's YouTube channel. This may be found via the following link:
https://www.youtube.com/channel/UCwD8J5X55tO1eyf1tXEYWTQ/feed

It is hoped that this initial work will serve as a launching pad for the continued development of Simulink blocks in order to extend the capabilities of such a rapid-prototyping platform.

12.0 REFERENCES

[1] Barret, F.S., Kridner, J. (2013): Bad to the Bone: Crafting Electronic Systems with BeagleBone and BeagleBone Black, Morgan & Claypool Publishers

[2] Madgwick, S.O.H, Harrison, A.J.L, Vaidyanathan R. (2011): Estimation of IMU and MARG Orientation Using a Gradient Descent Algorithm, 2011 IEEE International Conference on Rehabilitation Robotics.

[3] Farrell, A. J., (2008): Aided Navigation: GPS with High Rate Sensors, McGraw Hill Professional

[4] MathWorks (2014): "Hardware Support: BeagleBoard Support from Simulink" http://www.mathworks.com/hardware-support/beagleboard.html (accessed June 2014)

[5] Mathworks (2014): "Writing A Simulink Driver Block: A Step by Step Guide" http://www.mathworks.com/matlabcentral/fileexchange/39354-device-drivers (accessed June 2014)

[6] Molloy, D. (2014): "Electronic Engineering Education and Innovation" http://derekmolloy.ie/ (accessed June 2014)

[7] Google Groups (2014): "BeagelBoard: Simulink on BeagleBone Black" https://groups.google.com/forum/#!topic/beagleboard/fC-Bg_-Ia34 (accessed June 2014)

[8] BeagleBoard.org (2014): "BeagleBone: Open-Hardware Expandable Computer" http://beagleboard.org/Support/bone101 (accessed June 2014)

[9] Armhf.com (2014): "ARMhf: Linux for ARMhf Devices" http://www.armhf.com/ (accessed July 2014)

[10] Will. S 2013. "Installing Ubuntu 13.10 Saucy on the BeagleBone Black" http://sheldondwill.wordpress.com/2013/12/14/installing-ubuntu-13-10-saucy-on-the-beaglebone-black/ (accessed June 2014)

[11] DeviceTree.org (2014). "Device Tree Usage" http://devicetree.org/Device_Tree_Usage (accessed July 2014)

[12] Adafruit (2014): "Tutorial: Introduction to the BeagleBone Black Device Tree" http://www.adafruit.com/blog/2013/07/29/tutorial-introduction-to-the-beaglebone-black-device-tree/ (accessed July 2014)

[13] eLinux.org (2014): "Embedded Linux Wiki: BeagleBoard Ubuntu", http://elinux.org/BeagleBoardUbuntu (accessed June 2014)

[14] The Linux Kernel Organization (2014): "The Linux Kernel Archives", https://www.kernel.org/ (accessed July 2014)

[15] Byte Paradigm (2014): "Introduction to I2C and SPI protocols", http://www.byteparadigm.com/applications/introduction-to-i2c-and-spi-protocols/ (accessed July 2014)

[16] Mochel, P (2005) : "The sysfs Filesystem", https://www.kernel.org/pub/linux/kernel/people/mochel/doc/papers/ols-2005/mochel.pdf (accessed September 2014)

13.0 APPENDIX

13.1 S-FUNCTION BLOCK WRAPPER CODE: GPIO READ BLOCK

```
/*
  *
  *   --- THIS FILE GENERATED BY S-FUNCTION BUILDER: 3.0 ---
  *
  *   This file is a wrapper S-function produced by the S-Function
  *   Builder which only recognizes certain fields.  Changes made
  *   outside these fields will be lost the next time the block is
  *   used to load, edit, and resave this file. This file will be overwritten
  *   by the S-function Builder block. If you want to edit this file by hand,
  *   you must change it only in the area defined as:
  *
  *        %%%-SFUNWIZ_wrapper_XXXXX_Changes_BEGIN
  *            Your Changes go here
  *        %%%-SFUNWIZ_wrapper_XXXXXX_Changes_END
  *
  *   For better compatibility with the Simulink Coder, the
  *   "wrapper" S-function technique is used.  This is discussed
  *   in the Simulink Coder User's Manual in the Chapter titled,
  *   "Wrapper S-functions".
  *
  *   Created: Sun Dec 14 04:52:20 2014
  */


/*
 * Include Files
 *
 */
#if defined(MATLAB_MEX_FILE)
#include "tmwtypes.h"
#include "simstruc_types.h"
#else
#include "rtwtypes.h"
#endif

/* %%%-SFUNWIZ_wrapper_includes_Changes_BEGIN --- EDIT HERE TO _END */
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
//#include <unistd.h>
#include <fcntl.h>
//#include <poll.h>

#define sSYSFS_GPIO_Dir "/sys/class/gpio"

#define MAX_BUF 100

// Declare Pin Value Enum
```

```
typedef enum
{
        iPin_Low=0,
    iPin_High=1
}eGPIO_PinValue;

// Declare Pin Direction Enum
typedef enum
{
        iPin_Input,
    iPin_Output
}eGPIO_PinDirection;


// Declare Pin Number Enum
typedef enum
{
        iPin_GPIO_49=49,
    iPin_GPIO_51=51,
    iPin_GPIO_60=60
}eGPIO_PinNo;


// Declare Error Handling
// extern int errno;

// Define Debug Levels
typedef enum
{
        Debug_None,
    Debug_Level_0, // Basic Debug Info Output
        Debug_Level_1, // + Critical Info Only
    Debug_Level_2, // + Diagnostics Info
    Debug_Level_3, // + Program Flow Info
    Debug_Level_4, // + Registry Data
    Debug_Level_5  // + TBD
}eDebugLevel;

// Define Error Number
typedef enum
{
    Error_None=0,
    Error_No_SYSFSOpen,
    Error_No_SYSFSRead,
    Error_No_SYSFSWrite,
    Error_No_ParamOutofRange
}eError_No;

// Initialize Error
// eError_No iError_No = Error_None;

// Define Error Severity Levels
typedef enum
{
    Error_Level_OK=0,
```

```
    Error_Level_Critical, // (Stop Simulation)
    Error_Level_Warning
}eError_Level;

// Initialize Error Level
// eError_Level iError_Level = Error_Level_OK;
/* %%%-SFUNWIZ_wrapper_includes_Changes_END --- EDIT HERE TO _BEGIN */
#define u_width 1
#define y_width 1
/*
 * Create external references here.
 *
 */
/* %%%-SFUNWIZ_wrapper_externs_Changes_BEGIN --- EDIT HERE TO _END */
/* extern double func(double a); */
/* %%%-SFUNWIZ_wrapper_externs_Changes_END --- EDIT HERE TO _BEGIN */

/*
 * Output functions
 *
 */
void BBB_Driver_GPIO_Read_Outputs_wrapper(const boolean_T
*inGPIO_Read_Enable,
                          const boolean_T *inGPIO_Read,
                          boolean_T *outGPIO_ReadValue,
                          boolean_T *outSimStop  ,
                          const uint16_T  *prmGPIO_No, const int_T  p_width0,
                          const uint16_T  *prmDebug_InfoLevel,  const int_T
p_width1)
{
/* %%%-SFUNWIZ_wrapper_Outputs_Changes_BEGIN --- EDIT HERE TO _END */
// Run Only on Target - BeagleBoneBlack */
# ifndef MATLAB_MEX_FILE


// Declare and Initialize Error
eError_No iError_No = Error_None;

// Declare and Initialize Error Level
eError_Level iError_Level = Error_Level_OK;

// Declare and Initialize SYSFS Strings
char sBBB_GPIO_FILE[MAX_BUF]="";
char sBBB_GPIO_No[3]="";
char sSYSFS_GPIO_UnExp[MAX_BUF]="";
char sSYSFS_GPIO_Exp[MAX_BUF]="";
char sSYSFS_GPIO_Direction[MAX_BUF]="";
char sSYSFS_GPIO_Value[MAX_BUF]="";
char cSYSFS_GPIO_ReadVal="";

// Declare and Initialize File Handles
int iFILE_BBB_GPIO_Open_Handle=0;
int iFILE_BBB_GPIO_Read_Handle=0;
int iFILE_BBB_GPIO_Write_Handle=0;
```

```
// Declare and Initialize Ints
unsigned int iGPIO_No_Req=0;
unsigned int iGPIO_ReadValue=iPin_Low;

// Initialize Debug Output Level
eDebugLevel iDebug_Level = Debug_None;


/*************************************/
// Export (Enable) GPIO File For Read
/*************************************/
int iBBB_GPIOExport(eGPIO_PinNo iGPIONo)
{

    // Create Handle to SYSFS File Stream
    sprintf(sSYSFS_GPIO_Exp,"%s/export",sSYSFS_GPIO_Dir);

    // Check GPIO Bus Concatenation
    if (iDebug_Level >=Debug_Level_2)  { printf("GPIORead Msg: GPIO Export
SYSFS Path: %s\n",sSYSFS_GPIO_Exp); }

    // Open Data IO Stream
    iFILE_BBB_GPIO_Open_Handle = open(sSYSFS_GPIO_Exp, O_WRONLY);

    // Confirm SYSFS IOStream Open
    if (iFILE_BBB_GPIO_Open_Handle<0)
    {
        // Error Handling
        if (iDebug_Level >=Debug_Level_1)  { printf("GPIORead Error: Failed
Find/Open GPIO SYSFS Export File - Check GPIO Number/Device Tree\n"); }
        // Set Error Number
        iError_No = Error_No_SYSFSOpen;
        // Set Error Level
        iError_Level = Error_Level_Critical;
        // Return Otuput Error Level
        // return -1;
    }

    // Ouput Program Flow
    if (iDebug_Level >=Debug_Level_3) {printf("GPIORead Msg: Program Flow
State 2: Open GPIO SYSFS File\nError No: %i Error Level: %i\n",iError_No,
iError_Level);}

    // Write GPIO For Export Command
    if (iError_No == Error_None)
    {

        // Create Handle to SYSFS File Stream
        sprintf(sBBB_GPIO_No,"%d",iGPIONo);

        // Check GPIO Bus Concatenation
        if (iDebug_Level >=Debug_Level_2)  { printf("GPIORead Msg: GPIO No:
%s\n",sBBB_GPIO_No); }
```

```
        // Write GPIO SYSFS Data

iFILE_BBB_GPIO_Write_Handle=write(iFILE_BBB_GPIO_Open_Handle,sBBB_GPIO_No,siz
eof(sBBB_GPIO_No));

        // Error Handling
        if (iFILE_BBB_GPIO_Write_Handle<0)
        {
            if (iDebug_Level >=Debug_Level_1) {printf("GPIORead Error: Failed
to Export GPIO Number To User Space\n");}
            // Set Error Number
            iError_No = Error_No_SYSFSWrite;
            // Set Error Level
            iError_Level = Error_Level_Critical;
            // Return Otuput Error Level
            // return -1;
        }
    }

    // Close File Handle
    close(iFILE_BBB_GPIO_Open_Handle);

    // Set Return State
    if (iError_No==0){return 0;} else {return -1;}

}

/*************************************/
// Unexport (Disable) GPIO For Read
/*************************************/
int iBBB_GPIOUnExport(eGPIO_PinNo iGPIONo)
{

    // Create Handle to SYSFS File Stream
    sprintf(sSYSFS_GPIO_UnExp,"%s/unexport",sSYSFS_GPIO_Dir);

    // Check GPIO Bus Concatenation
    if (iDebug_Level >=Debug_Level_2)  { printf("GPIORead Msg: GPIO UnExport
SYSFS Path: %s\n",sSYSFS_GPIO_UnExp); }

    // Open Data IO Stream
    iFILE_BBB_GPIO_Open_Handle = open(sSYSFS_GPIO_UnExp, O_WRONLY);

    // Confirm SYSFS IOStream Open
    if (iFILE_BBB_GPIO_Open_Handle<0)
    {
        // Error Handling
        if (iDebug_Level >=Debug_Level_1)  { printf("GPIORead Error: Failed
Find/Open GPIO UnExport SYSFS File - Check GPIO Number/Device Tree\n"); }
        // Set Error Number
        iError_No = Error_No_SYSFSOpen;
        // Set Error Level
        iError_Level = Error_Level_Critical;
        // Return Otuput Error Level
```

```
        // return -1;
    }

    // Output Program Flow
    if (iDebug_Level >=Debug_Level_3) {printf("GPIORead Msg: Program Flow
State 2: Open GPIO SYSFS File\nError No: %i Error Level: %i\n",iError_No,
iError_Level);}

    // Write GPIO For Export Command
    if (iError_No == Error_None)
    {

        // Create Handle to SYSFS File Stream
        sprintf(sBBB_GPIO_No,"%d",iGPIONo);

        // Write GPIO SYSFS Data

iFILE_BBB_GPIO_Write_Handle=write(iFILE_BBB_GPIO_Open_Handle,sBBB_GPIO_No,siz
eof(sBBB_GPIO_No));

        // Error Handling
        if (iFILE_BBB_GPIO_Write_Handle<0)
        {
            if (iDebug_Level >=Debug_Level_1) {printf("GPIORead Error: Failed
to UnExport GPIO Number To User Space\n");}
            // Set Error Number
            iError_No = Error_No_SYSFSWrite;
            // Set Error Level
            iError_Level = Error_Level_Critical;
            // Return Otuput Error Level
            // return -1;
        }
    }

    // Close File Handle
    close(iFILE_BBB_GPIO_Open_Handle);

    // Set Return State
    if (iError_No==0){return 0;} else {return -1;}
}

/*************************************/
// Set GPIO Direction (Read/Write)
/*************************************/
int iBBB_GPIOSetDirection(eGPIO_PinNo iGPIONo, eGPIO_PinDirection
iGPIODirection)
{

    // Create Handle to SYSFS File Stream

sprintf(sSYSFS_GPIO_Direction,"%s/gpio%d/direction",sSYSFS_GPIO_Dir,iGPIONo);

    // Check GPIO Bus Concatenation
    if (iDebug_Level >=Debug_Level_2)  { printf("GPIORead Msg: GPIO Direction
SYSFS Path: %s\n",sSYSFS_GPIO_Direction); }
```

```
    // Open Data IO Stream
    iFILE_BBB_GPIO_Open_Handle = open(sSYSFS_GPIO_Direction, O_WRONLY);

    // Confirm SYSFS IOStream Open
    if (iFILE_BBB_GPIO_Open_Handle<0)
    {
        // Error Handling
        if (iDebug_Level >=Debug_Level_1)  { printf("GPIORead Error: Failed
Find/Open GPIO Direction SYSFS File - Check GPIO Number/Device Tree\n"); }
        // Set Error Number
        iError_No = Error_No_SYSFSOpen;
        // Set Error Level
        iError_Level = Error_Level_Critical;
        // Return Otuput Error Level
        // return -1;
    }

    // Ouput Program Flow
    if (iDebug_Level >=Debug_Level_3) {printf("GPIORead Msg: Program Flow
State 2: Open GPIO SYSFS File\nError No: %i Error Level: %i\n",iError_No,
iError_Level);}

    // Write GPIO For Export Command
    if (iError_No == Error_None)
    {

        // Create Handle to SYSFS File Stream
        // sprintf(sBBB_GPIO_No,"%d",iGPIONo);

        if (iGPIODirection==iPin_Input)
        {

            // Write GPIO SYSFS Data

iFILE_BBB_GPIO_Write_Handle=write(iFILE_BBB_GPIO_Open_Handle,"in",sizeof("in"
));
        }
        else
        {
            // Write GPIO SYSFS Data

iFILE_BBB_GPIO_Write_Handle=write(iFILE_BBB_GPIO_Open_Handle,"out",sizeof("ou
t"));
        }

        // Error Handling
        if (iFILE_BBB_GPIO_Write_Handle<0)
        {
            if (iDebug_Level >=Debug_Level_1) {printf("GPIORead Error: Failed
to Set GPIO In/Out Direction In User Space\n");}
            // Set Error Number
            iError_No = Error_No_SYSFSWrite;
            // Set Error Level
            iError_Level = Error_Level_Critical;
```

```
            // Return Otuput Error Level
            // return -1;
        }
    }

    // Close File Handle
    close(iFILE_BBB_GPIO_Open_Handle);

    // Set Return State
    if (iError_No==0){return 0;} else {return -1;}

}

/*************************************/
// Read GPIO Input Value
/*************************************/
int iBBB_GPIORead(eGPIO_PinNo iGPIONo, unsigned int *iGPIOVal)
{

    // Create Handle to SYSFS File Stream
    sprintf(sSYSFS_GPIO_Value,"%s/gpio%d/value",sSYSFS_GPIO_Dir,iGPIONo);

    // Check GPIO Bus Concatenation
    if (iDebug_Level >=Debug_Level_2)  { printf("GPIORead Msg: GPIO Value
SYSFS Path: %s\n",sSYSFS_GPIO_Value); }

    // Open Data IO Stream
    iFILE_BBB_GPIO_Open_Handle = open(sSYSFS_GPIO_Value, O_RDONLY);

    // Confirm SYSFS IOStream Open
    if (iFILE_BBB_GPIO_Open_Handle<0)
    {
        // Error Handling
        if (iDebug_Level >=Debug_Level_1)  { printf("GPIORead Error: Failed
Find/Open GPIO Value SYSFS File - Check GPIO Number/Device Tree\n"); }
        // Set Error Number
        iError_No = Error_No_SYSFSOpen;
        // Set Error Level
        iError_Level = Error_Level_Critical;
        // Return Otuput Error Level
        // return -1;
    }

    // Ouput Program Flow
    if (iDebug_Level >=Debug_Level_3) {printf("GPIORead Msg: Program Flow
State 2: Open GPIO SYSFS File\nError No: %i Error Level: %i\n",iError_No,
iError_Level);}

    // Write GPIO For Export Command
    if (iError_No == Error_None)
    {

        // Create Handle to SYSFS File Stream
        // sprintf(sBBB_GPIO_No,"%d",iGPIONo);
```

```
        // Write GPIO SYSFS Data
iFILE_BBB_GPIO_Read_Handle=read(iFILE_BBB_GPIO_Open_Handle,&cSYSFS_GPIO_ReadV
al,sizeof(&cSYSFS_GPIO_ReadVal));

         // Check GPIO Bus Concatenation
        if (iDebug_Level >=Debug_Level_2)  { printf("GPIORead Msg: GPIO Value
Read: %s\n",&cSYSFS_GPIO_ReadVal); }


        // Check Value
        if (cSYSFS_GPIO_ReadVal=='1')
        {
            // Transfer Out Read Value
            *iGPIOVal=iPin_High;
        }
        else
        {
            // Transfer Out Read Value
            *iGPIOVal=iPin_Low;
        }

        // Error Handling
        if (iFILE_BBB_GPIO_Read_Handle<0)
        {
            if (iDebug_Level >=Debug_Level_1) {printf("GPIORead Error: Failed
to Read GPIO Value In User Space\n");}
            // Set Error Number
            iError_No = Error_No_SYSFSRead;
            // Set Error Level
            iError_Level = Error_Level_Critical;
            // Return Otuput Error Level
            // return -1;
        }
    }

    // Close File Handle
    close(iFILE_BBB_GPIO_Open_Handle);

    // Set Return State
    if (iError_No==0){return 0;} else {return -1;}

}

/*************************************/
// Write GPIO Output Value
/*************************************/
int iBBB_GPIOWrite(eGPIO_PinNo iGPIONo, eGPIO_PinValue iGPIOVal)
{

    // Create Handle to SYSFS File Stream
    sprintf(sSYSFS_GPIO_Value,"%s/gpio%d/value",sSYSFS_GPIO_Dir,iGPIONo);

    // Check GPIO Bus Concatenation
```

```
    if (iDebug_Level >=Debug_Level_2)  { printf("GPIORead Msg: GPIO Write
SYSFS Path: %s\n",sSYSFS_GPIO_Value); }

    // Open Data IO Stream
    iFILE_BBB_GPIO_Open_Handle = open(sSYSFS_GPIO_Value, O_WRONLY);

    // Confirm SYSFS IOStream Open
    if (iFILE_BBB_GPIO_Open_Handle<0)
    {
        // Error Handling
        if (iDebug_Level >=Debug_Level_1)  { printf("GPIORead Error: Failed
Find/Open GPIO Value SYSFS File - Check GPIO Number/Device Tree\n"); }
        // Set Error Number
        iError_No = Error_No_SYSFSOpen;
        // Set Error Level
        iError_Level = Error_Level_Critical;
        // Return Otuput Error Level
        // return -1;
    }

    // Ouput Program Flow
    if (iDebug_Level >=Debug_Level_3) {printf("GPIORead Msg: Program Flow
State 2: Open GPIO SYSFS File\nError No: %i Error Level: %i\n",iError_No,
iError_Level);}

    // Write GPIO For Export Command
    if (iError_No == Error_None)
    {

        // Check Requested Write Value
        if (iGPIOVal==iPin_High)
        {
            // Write GPIO SYSFS Data

iFILE_BBB_GPIO_Write_Handle=write(iFILE_BBB_GPIO_Open_Handle,"1",sizeof("1"))
;
        }
        else
        {
            // Write GPIO SYSFS Data

iFILE_BBB_GPIO_Write_Handle=write(iFILE_BBB_GPIO_Open_Handle,"0",sizeof("0"))
;
        }



        // Error Handling
        if (iFILE_BBB_GPIO_Write_Handle<0)
        {
            if (iDebug_Level >=Debug_Level_1) {printf("GPIORead Error: Failed
to Write GPIO Value In User Space\n");}
            // Set Error Number
            iError_No = Error_No_SYSFSWrite;
            // Set Error Level
```

```
            iError_Level = Error_Level_Critical;
            // Return Otuput Error Level
            // return -1;
        }
    }

    // Close File Handle
    close(iFILE_BBB_GPIO_Open_Handle);

    // Set Return State
    if (iError_No==0){return 0;} else {return -1;}

}


const char *sGPIORead_ErrMessage(eError_No iGPIORead_ErrNo)
{
    //Set Error Message Based on Error Number
    switch(iGPIORead_ErrNo)
    {
       case  Error_None:
            return("No Error Active. System A-OK!\n");
            break;
        case  Error_No_SYSFSOpen:
            return("Failed to Open I2C Bus\n - Check Bus Number\n - Check
SCL/SCA Lines\n"
                   " - Check Device\n - Check Device Tree on BBB\n");
            break;
        case Error_No_SYSFSRead:
            return("Failed to Communicate With Slave Device\n - Check Bus
Number\n - Check SCL/SCA Lines\n"
                   " - Check Device Address\n - Check Device Tree on BBB\n");
            break;
        case Error_No_SYSFSWrite:
            return("Failed to Set Buffer Read Range\n - Check Bus Number\n -
Check SCL/SCA Lines\n"
                   " - Check Device Address\n - Check Device Tree on BBB\n");
            break;
        case Error_No_ParamOutofRange:
            return("Parameter Out Of Range\n - Check Block Parameter Input
Options\n");
            break;
        default:
            // Error -Return Default
            return("Error Code Not Recognized\n");
            break;
    }
}


/***************************************/
/***************************************/
// Main Program Start
/***************************************/
/***************************************/
```

```
// Run Through Program Only If Block Enabled
if ((inGPIO_Read[0]==true)&&(inGPIO_Read_Enable[0]==true))
{

// Read Parameter Data -Device Number
    switch(prmGPIO_No[0])
    {
        case  1: //iAngVel_250dps:
            iGPIO_No_Req=iPin_GPIO_49;
            break;
        case  2:
            iGPIO_No_Req=iPin_GPIO_51;
            break;
        case  3:
            iGPIO_No_Req=iPin_GPIO_60;
            break;
        default:
            iGPIO_No_Req=iPin_GPIO_49;
            break;
    }

    // Read Parameter Data -Debug Level
    switch(prmDebug_InfoLevel[0])
    {
        case  1:
            iDebug_Level=Debug_None;
            break;
        case  2:
            iDebug_Level=Debug_Level_0;
            break;
        case  3:
            iDebug_Level=Debug_Level_1;
            break;
        case  4:
            iDebug_Level=Debug_Level_2;
            break;
        case  5:
            iDebug_Level=Debug_Level_3;
            break;
        case  6:
            iDebug_Level=Debug_Level_4;
            break;
        case  7:
            iDebug_Level=Debug_Level_5;
            break;
        default:
            iDebug_Level=Debug_None;
            break;
    }

// Start Line
if (iDebug_Level >=Debug_Level_0){printf("**** Start of GPIO Read S-Function
Block Execution **** \n");}
```

```
// Export GPIO Pin
    iBBB_GPIOExport(iGPIO_No_Req);

// Set Direction
    if (iError_No == Error_None)
    {
        // Set GPIO Direction
        iBBB_GPIOSetDirection(iGPIO_No_Req, iPin_Input);
    }

// Read Data
    if (iError_No == Error_None)
    {
        // Read GPIO Data
        iBBB_GPIORead(iGPIO_No_Req, &iGPIO_ReadValue);
    }

// UnExport GPIO sysfs
    if (iError_No == Error_None)
    {
        // UnExport GPIO Pin
        iBBB_GPIOUnExport(iGPIO_No_Req);
    }

// Check For Critical Errors and Stop Simulation
    if ((iError_No!=0)&&(iError_Level==Error_Level_Critical))
    {

        // Stop Simulation If Critical Error
        if (iDebug_Level >=Debug_Level_0) {printf("GPIORead Msg: Critical
Error Detected; Simulation Stopping! \nError No: %i Error Level:
%i\n",iError_No, iError_Level);}

        // Output Error Message:
        if (iDebug_Level >=Debug_Level_0) {printf("GPIO Error Details:
%s",sGPIORead_ErrMessage(iError_No));}


        // Stop Simulation
        outSimStop[0]=true;
    }
    else
    {
        // Carry on
        outSimStop[0]=false;

        // Set Output Data
        if (iGPIO_ReadValue==1)
        {
            outGPIO_ReadValue[0]=true;
        }
        else
        {
            outGPIO_ReadValue[0]=false;
        }
```

```
    }

    // Print Output
    if (iDebug_Level >=Debug_Level_0){printf("Data From GPIO: %d  Output:
%d\n",iGPIO_ReadValue,outGPIO_ReadValue[0]);}


// Print Gap
if (iDebug_Level >=Debug_Level_0){printf("\n");}

}

# else
# endif
/* %%%-SFUNWIZ_wrapper_Outputs_Changes_END --- EDIT HERE TO _BEGIN */
}
```

13.2 S-FUNCTION BLOCK WRAPPER CODE: GPIO WRITE BLOCK

```
/*
  *
  *   --- THIS FILE GENERATED BY S-FUNCTION BUILDER: 3.0 ---
  *
  *   This file is a wrapper S-function produced by the S-Function
  *   Builder which only recognizes certain fields.  Changes made
  *   outside these fields will be lost the next time the block is
  *   used to load, edit, and resave this file. This file will be overwritten
  *   by the S-function Builder block. If you want to edit this file by hand,
  *   you must change it only in the area defined as:
  *
  *        %%%-SFUNWIZ_wrapper_XXXXX_Changes_BEGIN
  *            Your Changes go here
  *        %%%-SFUNWIZ_wrapper_XXXXXX_Changes_END
  *
  *   For better compatibility with the Simulink Coder, the
  *   "wrapper" S-function technique is used.  This is discussed
  *   in the Simulink Coder User's Manual in the Chapter titled,
  *   "Wrapper S-functions".
  *
  *   Created: Sun Dec 14 04:55:04 2014
  */


/*
 * Include Files
 *
 */
#if defined(MATLAB_MEX_FILE)
#include "tmwtypes.h"
#include "simstruc_types.h"
#else
#include "rtwtypes.h"
#endif

/* %%%-SFUNWIZ_wrapper_includes_Changes_BEGIN --- EDIT HERE TO _END */
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
//#include <unistd.h>
#include <fcntl.h>
//#include <poll.h>

#define sSYSFS_GPIO_Dir "/sys/class/gpio"

#define MAX_BUF 100

// Declare Pin Value Enum
typedef enum
{
```

```
        iPin_Low=0,
    iPin_High=1
}eGPIO_PinValue;

// Declare Pin Direction Enum
typedef enum
{
        iPin_Input,
    iPin_Output
}eGPIO_PinDirection;


// Declare Pin Number Enum
typedef enum
{
        iPin_GPIO_49=49,
    iPin_GPIO_51=51,
    iPin_GPIO_60=60
}eGPIO_PinNo;


// Declare Error Handling
// extern int errno;

// Define Debug Levels
typedef enum
{
        Debug_None,
    Debug_Level_0, // Basic Debug Info Output
        Debug_Level_1, // + Critical Info Only
    Debug_Level_2, // + Diagnostics Info
    Debug_Level_3, // + Program Flow Info
    Debug_Level_4, // + Registry Data
    Debug_Level_5  // + TBD
}eDebugLevel;


// Define Error Number
typedef enum
{
    Error_None=0,
    Error_No_SYSFSOpen,
    Error_No_SYSFSRead,
    Error_No_SYSFSWrite,
    Error_No_ParamOutofRange
}eError_No;

// Initialize Error
// eError_No iError_No = Error_None;

// Define Error Severity Levels
typedef enum
{
    Error_Level_OK=0,
    Error_Level_Critical, // (Stop Simulation)
```

```
    Error_Level_Warning
}eError_Level;

// Initialize Error Level
// eError_Level iError_Level = Error_Level_OK;
/* %%%-SFUNWIZ_wrapper_includes_Changes_END --- EDIT HERE TO _BEGIN */
#define u_width 1
#define y_width 1
/*
 * Create external references here.
 *
 */
/* %%%-SFUNWIZ_wrapper_externs_Changes_BEGIN --- EDIT HERE TO _END */
/* extern double func(double a); */
/* %%%-SFUNWIZ_wrapper_externs_Changes_END --- EDIT HERE TO _BEGIN */

/*
 * Output functions
 *
 */
void BBB_Driver_GPIO_Write_Outputs_wrapper(const boolean_T
*inGPIO_Write_Enable,
                          const boolean_T *inGPIO_Write,
                          boolean_T *outGPIO_WriteValue,
                          boolean_T *outSimStop ,
                  const real_T  *xD,
                          const uint16_T  *prmGPIO_No, const int_T  p_width0,
                          const uint16_T  *prmDebug_InfoLevel, const int_T
p_width1)
{
/* %%%-SFUNWIZ_wrapper_Outputs_Changes_BEGIN --- EDIT HERE TO _END */
// Run Only on Target - BeagleBoneBlack */
# ifndef MATLAB_MEX_FILE


// Declare and Initialize Error
eError_No iError_No = Error_None;

// Declare and Initialize Error Level
eError_Level iError_Level = Error_Level_OK;

// Declare and Initialize SYSFS Strings
char sBBB_GPIO_FILE[MAX_BUF]="";
char sBBB_GPIO_No[3]="";
char sSYSFS_GPIO_UnExp[MAX_BUF]="";
char sSYSFS_GPIO_Exp[MAX_BUF]="";
char sSYSFS_GPIO_Direction[MAX_BUF]="";
char sSYSFS_GPIO_Value[MAX_BUF]="";
char cSYSFS_GPIO_ReadVal="";

// Declare and Initialize File Handles
int iFILE_BBB_GPIO_Open_Handle=0;
int iFILE_BBB_GPIO_Read_Handle=0;
int iFILE_BBB_GPIO_Write_Handle=0;
```

```
// Declare and Initialize Ints
unsigned int iGPIO_No_Req=0;
unsigned int iGPIO_ReadValue=iPin_Low;

// Initialize Debug Output Level
eDebugLevel iDebug_Level = Debug_None;

/*************************************/
// Export (Enable) GPIO File For Read
/*************************************/
int iBBB_GPIOExport(eGPIO_PinNo iGPIONo)
{

    // Create Handle to SYSFS File Stream
    sprintf(sSYSFS_GPIO_Exp,"%s/export",sSYSFS_GPIO_Dir);

    // Check GPIO Bus Concatenation
    if (iDebug_Level >=Debug_Level_2)  { printf("GPIOWrite Msg: GPIO Export
SYSFS Path: %s\n",sSYSFS_GPIO_Exp); }

    // Open Data IO Stream
    iFILE_BBB_GPIO_Open_Handle = open(sSYSFS_GPIO_Exp, O_WRONLY);

    // Confirm SYSFS IOStream Open
    if (iFILE_BBB_GPIO_Open_Handle<0)
    {
        // Error Handling
        if (iDebug_Level >=Debug_Level_1)  { printf("GPIOWrite Error: Failed
Find/Open GPIO SYSFS Export File - Check GPIO Number/Device Tree\n"); }
        // Set Error Number
        iError_No = Error_No_SYSFSOpen;
        // Set Error Level
        iError_Level = Error_Level_Critical;
        // Return Otuput Error Level
        // return -1;
    }

    // Ouput Program Flow
    if (iDebug_Level >=Debug_Level_3) {printf("GPIOWrite Msg: Program Flow
State 2: Open GPIO SYSFS File\nError No: %i Error Level: %i\n",iError_No,
iError_Level);}

    // Write GPIO For Export Command
    if (iError_No == Error_None)
    {

        // Create Handle to SYSFS File Stream
        sprintf(sBBB_GPIO_No,"%d",iGPIONo);

        // Check GPIO Bus Concatenation
        if (iDebug_Level >=Debug_Level_2)  { printf("GPIOWrite Msg: GPIO No:
%s\n",sBBB_GPIO_No); }
```

```
        // Write GPIO SYSFS Data

iFILE_BBB_GPIO_Write_Handle=write(iFILE_BBB_GPIO_Open_Handle,sBBB_GPIO_No,siz
eof(sBBB_GPIO_No));

        // Error Handling
        if (iFILE_BBB_GPIO_Write_Handle<0)
        {
            if (iDebug_Level >=Debug_Level_1) {printf("GPIOWrite Error:
Failed to Export GPIO Number To User Space\n");}
            // Set Error Number
            iError_No = Error_No_SYSFSWrite;
            // Set Error Level
            iError_Level = Error_Level_Critical;
            // Return Otuput Error Level
            // return -1;
        }
    }

    // Close File Handle
    close(iFILE_BBB_GPIO_Open_Handle);

    // Set Return State
    if (iError_No==0){return 0;} else {return -1;}

}

/*************************************/
// Unexport (Disable) GPIO For Read
/*************************************/
int iBBB_GPIOUnExport(eGPIO_PinNo iGPIONo)
{

    // Create Handle to SYSFS File Stream
    sprintf(sSYSFS_GPIO_UnExp,"%s/unexport",sSYSFS_GPIO_Dir);

    // Check GPIO Bus Concatenation
    if (iDebug_Level >=Debug_Level_2)  { printf("GPIOWrite Msg: GPIO UnExport
SYSFS Path: %s\n",sSYSFS_GPIO_UnExp); }

    // Open Data IO Stream
    iFILE_BBB_GPIO_Open_Handle = open(sSYSFS_GPIO_UnExp, O_WRONLY);

    // Confirm SYSFS IOStream Open
    if (iFILE_BBB_GPIO_Open_Handle<0)
    {
        // Error Handling
        if (iDebug_Level >=Debug_Level_1)  { printf("GPIOWrite Error: Failed
Find/Open GPIO UnExport SYSFS File - Check GPIO Number/Device Tree\n"); }
        // Set Error Number
        iError_No = Error_No_SYSFSOpen;
        // Set Error Level
        iError_Level = Error_Level_Critical;
        // Return Otuput Error Level
        // return -1;
```

```
    }

    // Output Program Flow
    if (iDebug_Level >=Debug_Level_3) {printf("GPIOWrite Msg: Program Flow
State 2: Open GPIO SYSFS File\nError No: %i Error Level: %i\n",iError_No,
iError_Level);}

    // Write GPIO For Export Command
    if (iError_No == Error_None)
    {

        // Create Handle to SYSFS File Stream
        sprintf(sBBB_GPIO_No,"%d",iGPIONo);

        // Write GPIO SYSFS Data

iFILE_BBB_GPIO_Write_Handle=write(iFILE_BBB_GPIO_Open_Handle,sBBB_GPIO_No,siz
eof(sBBB_GPIO_No));

        // Error Handling
        if (iFILE_BBB_GPIO_Write_Handle<0)
        {
            if (iDebug_Level >=Debug_Level_1) {printf("GPIOWrite Error:
Failed to UnExport GPIO Number To User Space\n");}
            // Set Error Number
            iError_No = Error_No_SYSFSWrite;
            // Set Error Level
            iError_Level = Error_Level_Critical;
            // Return Otuput Error Level
            // return -1;
        }
    }

    // Close File Handle
    close(iFILE_BBB_GPIO_Open_Handle);

    // Set Return State
    if (iError_No==0){return 0;} else {return -1;}
}

/*************************************/
// Set GPIO Direction (Read/Write)
/*************************************/
int iBBB_GPIOSetDirection(eGPIO_PinNo iGPIONo, eGPIO_PinDirection
iGPIODirection)
{

    // Create Handle to SYSFS File Stream

sprintf(sSYSFS_GPIO_Direction,"%s/gpio%d/direction",sSYSFS_GPIO_Dir,iGPIONo);

    // Check GPIO Bus Concatenation
    if (iDebug_Level >=Debug_Level_2)  { printf("GPIOWrite Msg: GPIO
Direction SYSFS Path: %s\n",sSYSFS_GPIO_Direction); }
```

```
    // Open Data IO Stream
    iFILE_BBB_GPIO_Open_Handle = open(sSYSFS_GPIO_Direction, O_WRONLY);

    // Confirm SYSFS IOStream Open
    if (iFILE_BBB_GPIO_Open_Handle<0)
    {
        // Error Handling
        if (iDebug_Level >=Debug_Level_1)  { printf("GPIOWrite Error: Failed
Find/Open GPIO Direction SYSFS File - Check GPIO Number/Device Tree\n"); }
        // Set Error Number
        iError_No = Error_No_SYSFSOpen;
        // Set Error Level
        iError_Level = Error_Level_Critical;
        // Return Otuput Error Level
        // return -1;
    }

    // Ouput Program Flow
    if (iDebug_Level >=Debug_Level_3) {printf("GPIOWrite Msg: Program Flow
State 2: Open GPIO SYSFS File\nError No: %i Error Level: %i\n",iError_No,
iError_Level);}

    // Write GPIO For Export Command
    if (iError_No == Error_None)
    {

        // Create Handle to SYSFS File Stream
        // sprintf(sBBB_GPIO_No,"%d",iGPIONo);

        if (iGPIODirection==iPin_Input)
        {

            // Write GPIO SYSFS Data

iFILE_BBB_GPIO_Write_Handle=write(iFILE_BBB_GPIO_Open_Handle,"in",sizeof("in"
));
        }
        else
        {
            // Write GPIO SYSFS Data

iFILE_BBB_GPIO_Write_Handle=write(iFILE_BBB_GPIO_Open_Handle,"out",sizeof("ou
t"));
        }

        // Error Handling
        if (iFILE_BBB_GPIO_Write_Handle<0)
        {
            if (iDebug_Level >=Debug_Level_1) {printf("GPIOWrite Error:
Failed to Set GPIO In/Out Direction In User Space\n");}
            // Set Error Number
            iError_No = Error_No_SYSFSWrite;
            // Set Error Level
            iError_Level = Error_Level_Critical;
            // Return Otuput Error Level
```

```
            // return -1;
        }
    }

    // Close File Handle
    close(iFILE_BBB_GPIO_Open_Handle);

    // Set Return State
    if (iError_No==0){return 0;} else {return -1;}

}

/*************************************/
// Read GPIO Input Value
/*************************************/
int iBBB_GPIORead(eGPIO_PinNo iGPIONo, unsigned int *iGPIOVal)
{

    // Create Handle to SYSFS File Stream
    sprintf(sSYSFS_GPIO_Value,"%s/gpio%d/value",sSYSFS_GPIO_Dir,iGPIONo);

    // Check GPIO Bus Concatenation
    if (iDebug_Level >=Debug_Level_2)  { printf("GPIOWrite Msg: GPIO Value
SYSFS Path: %s\n",sSYSFS_GPIO_Value); }

    // Open Data IO Stream
    iFILE_BBB_GPIO_Open_Handle = open(sSYSFS_GPIO_Value, O_RDONLY);

    // Confirm SYSFS IOStream Open
    if (iFILE_BBB_GPIO_Open_Handle<0)
    {
        // Error Handling
        if (iDebug_Level >=Debug_Level_1)  { printf("GPIOWrite Error: Failed
Find/Open GPIO Value SYSFS File - Check GPIO Number/Device Tree\n"); }
        // Set Error Number
        iError_No = Error_No_SYSFSOpen;
        // Set Error Level
        iError_Level = Error_Level_Critical;
        // Return Otuput Error Level
        // return -1;
    }

    // Ouput Program Flow
    if (iDebug_Level >=Debug_Level_3) {printf("GPIOWrite Msg: Program Flow
State 2: Open GPIO SYSFS File\nError No: %i Error Level: %i\n",iError_No,
iError_Level);}

    // Write GPIO For Export Command
    if (iError_No == Error_None)
    {

        // Create Handle to SYSFS File Stream
        // sprintf(sBBB_GPIO_No,"%d",iGPIONo);
```

```
        // Write GPIO SYSFS Data

iFILE_BBB_GPIO_Read_Handle=read(iFILE_BBB_GPIO_Open_Handle,&cSYSFS_GPIO_ReadV
al,sizeof(&cSYSFS_GPIO_ReadVal));

         // Check GPIO Bus Concatenation
        if (iDebug_Level >=Debug_Level_2)  { printf("GPIOWrite Msg: GPIO
Value Read: %s\n",&cSYSFS_GPIO_ReadVal); }


        // Check Value
        if (cSYSFS_GPIO_ReadVal=='1')
        {
            // Transfer Out Read Value
            *iGPIOVal=iPin_High;
        }
        else
        {
            // Transfer Out Read Value
            *iGPIOVal=iPin_Low;
        }

        // Error Handling
        if (iFILE_BBB_GPIO_Read_Handle<0)
        {
            if (iDebug_Level >=Debug_Level_1) {printf("GPIOWrite Error:
Failed to Read GPIO Value In User Space\n");}
            // Set Error Number
            iError_No = Error_No_SYSFSRead;
            // Set Error Level
            iError_Level = Error_Level_Critical;
            // Return Otuput Error Level
            // return -1;
        }
    }

    // Close File Handle
    close(iFILE_BBB_GPIO_Open_Handle);

    // Set Return State
    if (iError_No==0){return 0;} else {return -1;}

}

/*************************************/
// Write GPIO Output Value
/*************************************/
int iBBB_GPIOWrite(eGPIO_PinNo iGPIONo, eGPIO_PinValue iGPIOVal)
{

    // Create Handle to SYSFS File Stream
    sprintf(sSYSFS_GPIO_Value,"%s/gpio%d/value",sSYSFS_GPIO_Dir,iGPIONo);

    // Check GPIO Bus Concatenation
```

```
    if (iDebug_Level >=Debug_Level_2)  { printf("GPIOWrite Msg: GPIO Write
SYSFS Path: %s\n",sSYSFS_GPIO_Value); }

    // Open Data IO Stream
    iFILE_BBB_GPIO_Open_Handle = open(sSYSFS_GPIO_Value, O_WRONLY);

    // Confirm SYSFS IOStream Open
    if (iFILE_BBB_GPIO_Open_Handle<0)
    {
        // Error Handling
        if (iDebug_Level >=Debug_Level_1)  { printf("GPIOWrite Error: Failed
Find/Open GPIO Value SYSFS File - Check GPIO Number/Device Tree\n"); }
        // Set Error Number
        iError_No = Error_No_SYSFSOpen;
        // Set Error Level
        iError_Level = Error_Level_Critical;
        // Return Otuput Error Level
        // return -1;
    }

    // Ouput Program Flow
    if (iDebug_Level >=Debug_Level_3) {printf("GPIOWrite Msg: Program Flow
State 2: Open GPIO SYSFS File\nError No: %i Error Level: %i\n",iError_No,
iError_Level);}

    // Write GPIO For Export Command
    if (iError_No == Error_None)
    {

        // Check Requested Write Value
        if (iGPIOVal==iPin_High)
        {
            // Write GPIO SYSFS Data

iFILE_BBB_GPIO_Write_Handle=write(iFILE_BBB_GPIO_Open_Handle,"1",2);//sizeof(
"1"));
        }
        else
        {
            // Write GPIO SYSFS Data

iFILE_BBB_GPIO_Write_Handle=write(iFILE_BBB_GPIO_Open_Handle,"0",2);//sizeof(
"0"));
        }



        // Error Handling
        if (iFILE_BBB_GPIO_Write_Handle<0)
        {
            if (iDebug_Level >=Debug_Level_1) {printf("GPIOWrite Error:
Failed to Write GPIO Value In User Space\n");}
            // Set Error Number
            iError_No = Error_No_SYSFSWrite;
            // Set Error Level
```

```
            iError_Level = Error_Level_Critical;
            // Return Otuput Error Level
            // return -1;
        }
    }

    // Close File Handle
    close(iFILE_BBB_GPIO_Open_Handle);

    // Set Return State
    if (iError_No==0){return 0;} else {return -1;}

}


/*************************************/
// GPIO Error Message Output
/*************************************/
const char *sGPIORead_ErrMessage(eError_No iGPIORead_ErrNo)
{
    //Set Error Message Based on Error Number
    switch(iGPIORead_ErrNo)
    {
       case  Error_None:
            return("No Error Active. System A-OK!\n");
            break;
        case  Error_No_SYSFSOpen:
            return("Failed to Open I2C Bus\n - Check Bus Number\n - Check
SCL/SCA Lines\n"
                   " - Check Device\n - Check Device Tree on BBB\n");
            break;
        case Error_No_SYSFSRead:
            return("Failed to Communicate With Slave Device\n - Check Bus
Number\n - Check SCL/SCA Lines\n"
                   " - Check Device Address\n - Check Device Tree on BBB\n");
            break;
        case Error_No_SYSFSWrite:
            return("Failed to Set Buffer Read Range\n - Check Bus Number\n -
Check SCL/SCA Lines\n"
                   " - Check Device Address\n - Check Device Tree on BBB\n");
            break;
        case Error_No_ParamOutofRange:
            return("Parameter Out Of Range\n - Check Block Parameter Input
Options\n");
            break;
        default:
            // Error -Return Default
            return("Error Code Not Recognized\n");
            break;
    }
}


/*************************************/
/*************************************/
```

```
// Main Program Start
/**************************************/
/**************************************/

// Run Through Program Only If Block Enabled
if (inGPIO_Write_Enable[0]==true)
{

// Read Parameter Data -Device Number
    switch(prmGPIO_No[0])
    {
        case  1: //GPIO to Control
            iGPIO_No_Req=iPin_GPIO_49;
            break;
        case  2:
            iGPIO_No_Req=iPin_GPIO_51;
            break;
        case  3:
            iGPIO_No_Req=iPin_GPIO_60;
            break;
        default:
            iGPIO_No_Req=iPin_GPIO_49;
            break;
    }

    // Read Parameter Data -Debug Level
    switch(prmDebug_InfoLevel[0])
    {
        case  1:
            iDebug_Level=Debug_None;
            break;
        case  2:
            iDebug_Level=Debug_Level_0;
            break;
        case  3:
            iDebug_Level=Debug_Level_1;
            break;
        case  4:
            iDebug_Level=Debug_Level_2;
            break;
        case  5:
            iDebug_Level=Debug_Level_3;
            break;
        case  6:
            iDebug_Level=Debug_Level_4;
            break;
        case  7:
            iDebug_Level=Debug_Level_5;
            break;
        default:
            iDebug_Level=Debug_None;
            break;
    }

    // Start Line
```

```
    if (iDebug_Level >=Debug_Level_0){printf("**** Start of GPIO Write S-
Function Block Execution **** \n");}

    // Export GPIO Pin Only On First Scan
    if (xD[0]==0)
    {

        // Export GPIO Pin Only On First Scan
        if (iBBB_GPIOExport(iGPIO_No_Req)<0)
        {
            // GPIO Already Exported. Reset Error
            iError_No = Error_None;
            // Reset Error Level
            iError_Level = Error_Level_OK;
        }
        else
        {
            // Do Nothing - GPIO Properly Exported
        }
    }

// Set Direction - Move Inside First Scan?
    if (iError_No == Error_None)
    {
        // Set GPIO Direction
        iBBB_GPIOSetDirection(iGPIO_No_Req, iPin_Output);
    }

// Write Data Depending on Input
    if (iError_No == Error_None)
    {

        // Check Input Command
        if (inGPIO_Write[0]==true)
        {
            // Set GPIO On
            iBBB_GPIOWrite(iGPIO_No_Req,iPin_High);
        }
        else
        {
            // Set GPIO On
            iBBB_GPIOWrite(iGPIO_No_Req,iPin_Low);
        }
    }

// UnExport GPIO sysfs
    if (iError_No == Error_None)
    {
        // UnExport GPIO Pin
        // iBBB_GPIOUnExport(iGPIO_No_Req);
    }

// Check For Critical Errors and Stop Simulation
    if ((iError_No!=0)&&(iError_Level==Error_Level_Critical))
    {
```

```
        // Stop Simulation If Critical Error
        if (iDebug_Level >=Debug_Level_0) {printf("GPIORead Msg: Critical
Error Detected; Simulation Stopping! \nError No: %i Error Level:
%i\n",iError_No, iError_Level);}

        // Output Error Message:
        if (iDebug_Level >=Debug_Level_0) {printf("GPIO Error Details:
%s",sGPIORead_ErrMessage(iError_No));}


        // Stop Simulation
        outSimStop[0]=true;
    }
    else
    {
        // Carry on
        outSimStop[0]=false;

        // Set Output Data

        // Get Feedback From GPIO Pin
        iBBB_GPIORead(iGPIO_No_Req, &iGPIO_ReadValue);

        if (iGPIO_ReadValue==true)
        {
            outGPIO_WriteValue[0]=true;
        }
        else
        {
            outGPIO_WriteValue[0]=false;
        }
    }

    // Print Output
    if (iDebug_Level >=Debug_Level_0){printf("Data To GPIO Output:
%d\n",outGPIO_WriteValue[0]);}


// Print Gap
if (iDebug_Level >=Debug_Level_0){printf("\n");}

}

# else
# endif
/* %%%-SFUNWIZ_wrapper_Outputs_Changes_END --- EDIT HERE TO _BEGIN */
}

/*
  * Updates function
  *
  */
```

```
void BBB_Driver_GPIO_Write_Update_wrapper(const boolean_T
*inGPIO_Write_Enable,
                          const boolean_T *inGPIO_Write,
                          const boolean_T *outGPIO_WriteValue,
                          const boolean_T *outSimStop ,
                          real_T *xD,
                          const uint16_T  *prmGPIO_No,  const int_T
p_width0,
                           const uint16_T *prmDebug_InfoLevel, const int_T
p_width1)
{
  /* %%%-SFUNWIZ_wrapper_Update_Changes_BEGIN --- EDIT HERE TO _END */
// Run Only on Target - BeagleBoneBlack */
# ifndef MATLAB_MEX_FILE

// Execute on First Scan
if (xD[0]==0)
{
    // Set First Scan Bit
    xD[0]=1;
}

# endif
/* %%%-SFUNWIZ_wrapper_Update_Changes_END --- EDIT HERE TO _BEGIN */
}
```

13.3 S-FUNCTION BLOCK WRAPPER CODE: I2C READ BLOCK

```
/*
  *
  *   --- THIS FILE GENERATED BY S-FUNCTION BUILDER: 3.0 ---
  *
  *   This file is a wrapper S-function produced by the S-Function
  *   Builder which only recognizes certain fields.  Changes made
  *   outside these fields will be lost the next time the block is
  *   used to load, edit, and resave this file. This file will be overwritten
  *   by the S-function Builder block. If you want to edit this file by hand,
  *   you must change it only in the area defined as:
  *
  *        %%%-SFUNWIZ_wrapper_XXXXX_Changes_BEGIN
  *            Your Changes go here
  *        %%%-SFUNWIZ_wrapper_XXXXXX_Changes_END
  *
  *   For better compatibility with the Simulink Coder, the
  *   "wrapper" S-function technique is used.  This is discussed
  *   in the Simulink Coder User's Manual in the Chapter titled,
  *   "Wrapper S-functions".
  *
  *   Created: Sun Dec 14 05:01:03 2014
  */


/*
 * Include Files
 *
 */
#if defined(MATLAB_MEX_FILE)
#include "tmwtypes.h"
#include "simstruc_types.h"
#else
#include "rtwtypes.h"
#endif

/* %%%-SFUNWIZ_wrapper_includes_Changes_BEGIN --- EDIT HERE TO _END */
#include <math.h>
#include <stdio.h>
#include <stddef.h>
#include <time.h>
#include <stdlib.h>
#include <string.h>
//#include <inttypes.h>
//#include <errno.h>


# ifndef MATLAB_MEX_FILE

    #include <linux/i2c.h>
    #include <linux/i2c-dev.h>
    #include <sys/ioctl.h>
```

```
    #include <unistd.h>
    #include <sys/types.h>
    #include <sys/stat.h>
    #include <fcntl.h>
    //#include <simstruc.h>
# else

    #include <mex.h>
    #include <simstruc.h>

# endif


//#define S_FUNCTION_NAME BBB_Driver_Gyroscope_L3G4200D
//#define S_FUNCTION_LEVEL 2

#define I2CMode_Read 1
#define I2CMode_Write 2


// Declare Error Handling
// extern int errno;


// Define Debug Levels
typedef enum
{
        Debug_None,
    Debug_Level_0, // Basic Debug Info Output
        Debug_Level_1, // + Critical Info Only
    Debug_Level_2, // + Diagnostics Info
    Debug_Level_3, // + Program Flow Info
    Debug_Level_4, // + Registry Data
    Debug_Level_5  // + TBD
}eDebugLevel;

// Define Error Number
typedef enum
{
    Error_None=0,
    Error_No_I2CComm,
    Error_No_I2CInit,
    Error_No_I2CBuffSet,
    Error_No_I2CBuffRead,
    Error_No_I2CBuffSync,
    Error_No_I2CBuffWrite,
    Error_No_I2CBuffValidate,
    Error_No_ParamOutofRange
}eError_No;

// Initialize Error
// eError_No iError_No = Error_None;

// Define Error Severity Levels
```

```
typedef enum
{
    Error_Level_OK=0,
    Error_Level_Critical, // (Stop Simulation)
    Error_Level_Warning
}eError_Level;

// Initialize Error Level
// eError_Level iError_Level = Error_Level_OK;

// Define Parameter Selection: Gyro Sensitivity
typedef enum
{
    iAngVel_250dps=1,
    iAngVel_500dps=2,
    iAngVel_2000dps=3
}eGyro_Range;
/* %%%-SFUNWIZ_wrapper_includes_Changes_END --- EDIT HERE TO _BEGIN */
#define u_width 1
#define y_width 1
/*
 * Create external references here.
 *
 */
/* %%%-SFUNWIZ_wrapper_externs_Changes_BEGIN --- EDIT HERE TO _END */
/* extern double func(double a); */
const char *sI2CReadWrite_ErrMessage(eError_No iErrNo)
{
    //Set Error Message Based on Error Number
    switch(iErrNo)
    {
       case  Error_None:
            return("No Error Active. System A-OK!\n");
            break;
        case  Error_No_I2CComm:
            return("Failed to Open I2C Bus\n - Check Bus Number\n - Check
SCL/SCA Lines\n"
                   " - Check Device\n - Check Device Tree on BBB\n");
            break;
        case Error_No_I2CInit:
            return("Failed to Communicate With Slave Device\n - Check Bus
Number\n - Check SCL/SCA Lines\n"
                   " - Check Device Address\n - Check Device Tree on BBB\n");
            break;
        case Error_No_I2CBuffSet:
            return("Failed to Set Buffer Read Range\n - Check Bus Number\n -
Check SCL/SCA Lines\n"
                   " - Check Device Address\n - Check Device Tree on BBB\n");
            break;
        case Error_No_I2CBuffRead:
            return("Failed to Read Data in Registry Buffer\n - Check Bus
Number\n - Check SCL/SCA Lines\n"
                   " - Check Device Address\n - Check Device Tree on BBB\n");
            break;
        case Error_No_I2CBuffSync:
```

```
            return("Failed to Validate Registry Data - Inconsistent Data
Stream\n"
                   " - Check For Multiple Master Access to Slave Device on
I2C Bus Number\n"
                   " - Check SCL/SCA Lines\n - Check Device Address\n -
Check Device Tree on BBB\n");
            break;
        case Error_No_I2CBuffWrite:
            return("Failed to Write Data to Device Registry\n - Check Bus
Traffic/Interference\n"
                  " - Check Bus Number\n - Check SCL/SCA Lines\n - Check
Device Address\n - Check Device Tree on BBB\n");
            break;
    case Error_No_I2CBuffValidate:
            return("Failed to Validate Data Written To Registry\n - Check Bus
Traffic/Interference\n"
                   " - Check Bus Number\n - Check SCL/SCA Lines\n - Check
Device Address\n - Check Device Tree on BBB\n");
            break;
        case Error_No_ParamOutofRange:
            return("Parameter Out Of Range\n - Check Block Parameter Input
Options\n");
            break;
        default:
            // Error -Return Default
            return("Error Code Not Recognized\n");
            break;


    }
}
// Set Debug Level
int iSet_Debug_Level(eDebugLevel iDebug_Level_X)
{

    switch(iDebug_Level_X)
    {
        case  1:
            return (Debug_None);
            break;
        case  2:
            return (Debug_Level_0);
            break;
        case  3:
            return (Debug_Level_1);
            break;
        case  4:
            return (Debug_Level_2);
            break;
        case  5:
            return (Debug_Level_3);
            break;
        case  6:
            return (Debug_Level_4);
            break;
        case  7:
```

```
            return (Debug_Level_5);
            break;
        default:
            return (Debug_None);
            break;
    }
}
/* %%%-SFUNWIZ_wrapper_externs_Changes_END --- EDIT HERE TO _BEGIN */

/*
 * Output functions
 *
 */
void BBB_Driver_I2C_Read_Outputs_wrapper(const boolean_T *inI2C_R_Enable,
                          const boolean_T *inI2C_R_Trigger,
                          uint8_T *outI2C_R_Data,
                          boolean_T *outSimStop ,
                  const real_T  *xD,
                          const uint8_T  *prmI2C_BusNo, const int_T
p_width0,
                          const uint8_T  *prmI2C_DeviceAddress, const int_T
p_width1,
                          const uint8_T  *prmI2C_RegisterNo, const int_T
p_width2,
                          const uint8_T  *prmDebug_InfoLevel, const int_T
p_width3)
{
/* %%%-SFUNWIZ_wrapper_Outputs_Changes_BEGIN --- EDIT HERE TO _END */
// Define Output Data - Simulink

// Run Only on Target - BeagleBoneBlack */
# ifndef MATLAB_MEX_FILE

// Define File Sys Pointer - For I2C Bus#1 -> Make Parameter
int iFILE_BBB_I2C_Handle=0;
int iIOCTL_BBB_I2C_Handle=0;
int iSYNC_BBB_I2C_Handle=0;
int iWrite_BBB_I2C_Handle=0;
// Function Outputs
int iI2C_Read=0;
int iI2C_Write=0;

//Define Device Driver sysfs Location
// const char *sBBB_I2C_BUS1_FILE ="/dev/i2c-1";
// char *sBBB_I2C_BUS1_FILE ="/dev/i2c-";
char sBBB_I2C_BUS1_FILE[100]="";

// Define I2C Parameters
int iI2C_BytesRead=0;
// Define I2C Buffer Size
const int iI2C_BufferSize=1;
// Define I2C Buffer Dump Variable
char cI2C_Buffer[1]={0x00}; //unsigned
// Define I2C Buffer Dump Start Position. Note: Value = Start Address w/ MSB
Set To "1"
char cI2C_Buffer_Sync[1]={0x00}; // Start at reg 0x00
```

```
// Define Write Buffer
char cI2C_Buffer_Write[2]={0x00};
// Define I2C Buffer Status
char cI2C_Buffer_Status=0;//unsigned

// Define Loop Counters
int iLoop=0;
int iLoop_Counter;

// Define Device Address
char cDevice_I2C_Addr_Para=0x00;
char cDevice_I2C_Addr=0X00;
char cDevice_I2C_RegAddr[1]={0x00};
char cDevice_I2C_WriteValue=0x00;

// Initialize Error
eError_No iError_No = Error_None;
// Initialize Error Level
eError_Level iError_Level = Error_Level_OK;
// Initialize Debug Output Level
eDebugLevel iDebug_Level = Debug_None;

// Define Function iI2C_Read_Register
int iI2C_Read_Register(char *cRegister_Address, char *cRegister_Value)
{

    // Create Handle to SYSFS File Stream
    sprintf(sBBB_I2C_BUS1_FILE,"/dev/i2c-%u",prmI2C_BusNo[0]);

    // Check I2C Bus Concatenation
    if (iDebug_Level >=Debug_Level_2)  { printf("I2C_Read Msg: I2C Bus File
Location: %s\n",sBBB_I2C_BUS1_FILE); }

    // Open Data IO Stream
    iFILE_BBB_I2C_Handle = open(sBBB_I2C_BUS1_FILE, O_RDWR);

    // Confirm I2C Bus Open
    if (iFILE_BBB_I2C_Handle<0)
    {
        // Error Handling
        if (iDebug_Level >=Debug_Level_1)  { printf("I2C_Read Error: Failed
to Open I2C Bus - Check Device/Device Tree\n"); }
        // Set Error Number
        iError_No = Error_No_I2CComm;
        // Set Error Level
        iError_Level = Error_Level_Critical;
        // Return Otuput Error Level
        // return -1;
    }

    // Ouput Program Flow
    if (iDebug_Level >=Debug_Level_3) {printf("I2C_Read Msg: Program Flow
State 2: Connect To I2C Bus\nError No: %i Error Level: %i\n",iError_No,
iError_Level);}
```

```
    // Initiate I2C Communicaton
    if (iError_No == Error_None)
    {
        // Initiate Communicaiton

iIOCTL_BBB_I2C_Handle=ioctl(iFILE_BBB_I2C_Handle,I2C_SLAVE,cDevice_I2C_Addr);

        // Error Handling
        if (iIOCTL_BBB_I2C_Handle<0)
        {
            if (iDebug_Level >=Debug_Level_1) {printf("I2C_Read Error: Failed
to Communicate With Slave Device: %#04x\n",cDevice_I2C_Addr);}
            // Set Error Number
            iError_No = Error_No_I2CInit;
            // Set Error Level
            iError_Level = Error_Level_Critical;
            // Return Otuput Error Level
            // return -1;
        }
    }

    // Ouput Program Flow
    if (iDebug_Level >=Debug_Level_3) {printf("I2C_Read Msg: Program Flow
State 3: Connect to I2C Device\nError No: %i Error Level: %i\n",iError_No,
iError_Level);}

    // Set Buffer Read Range
    if (iError_No == Error_None)
    {
        iWrite_BBB_I2C_Handle =
write(iFILE_BBB_I2C_Handle,cRegister_Address,1); //cI2C_Buffer_Sync
        // Error Handling
        if (iWrite_BBB_I2C_Handle !=1)
        {
            // Error Handling
            if (iDebug_Level >=Debug_Level_1) { printf("I2C_Read Error:
Failed to Write/Communicate With Device and/or Set Buffer Read Range\n");}
            // Set Error Number
            iError_No = Error_No_I2CBuffSet;
            // Set Error Level
            iError_Level = Error_Level_Critical;
            // Return Otuput Error Level
            // return -1;
        }
        else
        {
            if (iDebug_Level >=Debug_Level_1) { printf("I2C_Read Msg: Buffer
Range Set\n");}
        }
    }

    // Ouput Program Flow
    if (iDebug_Level >=Debug_Level_3) {printf("I2C_Read Msg: Program Flow
State 4: Write to Device Registry\nError No: %i Error Level: %i\n",iError_No,
iError_Level);}
```

```
    // Read Buffer Data
    if (iError_No == Error_None)
    {
        // Read and Store Registry Data in Buffer
        iI2C_BytesRead =
read(iFILE_BBB_I2C_Handle,cRegister_Value,1);//cI2C_Buffer
sizeof(cRegister_Value));
        // Close File
        close(iFILE_BBB_I2C_Handle);

        // Error Handling - Confirm Buffer Size
        if (iI2C_BytesRead<0)
        {
            // Error on Buffer Read!
            if (iDebug_Level >=Debug_Level_1) { printf("I2C_Read Error:
Failed to Read Device Register: %#04x Bytes Read:
%i\n",cRegister_Address,iI2C_BytesRead );}
            // Set Error Number
            iError_No = Error_No_I2CBuffRead;
            // Set Error Level
            iError_Level = Error_Level_Critical;
            // Return Otuput Error Level
            // return -1;
        }
        else
        {
            // Registry Data Read!
            if (iDebug_Level >=Debug_Level_1) { printf("I2C_Read Msg: Device
Register Read. Bytes Read: %i\n",iI2C_BytesRead );}

        }
    }

    // Set Return State
    if (iError_No==0){return 0;} else {return -1;}


}

// Define Function iGryro_I2CWriteBuff
int iI2C_Write_Register(char cRegister_Address, char cRegister_Value)
{

    // Create Handle to SYSFS File Stream
    sprintf(sBBB_I2C_BUS1_FILE,"/dev/i2c-%u",prmI2C_BusNo[0]);

    // Check I2C Bus Concatenation
    if (iDebug_Level >=Debug_Level_2)  { printf("I2C_Read Msg: I2C Bus File
Location: %s\n",sBBB_I2C_BUS1_FILE); }

    // Open Data IO Stream
    iFILE_BBB_I2C_Handle = open(sBBB_I2C_BUS1_FILE, O_RDWR);

    // Confirm I2C Bus Open
    if (iFILE_BBB_I2C_Handle<0)
```

```
    {
        // Error Handling
        if (iDebug_Level >=Debug_Level_1)  { printf("I2C_Read Error: Failed
to Open I2C Bus - Check Device/Device Tree\n"); }
        // Set Error Number
        iError_No = Error_No_I2CComm;
        // Set Error Level
        iError_Level = Error_Level_Critical;
        // Return Otuput Error Level
        // return -1;
    }

    // Ouput Program Flow
    if (iDebug_Level >=Debug_Level_3) {printf("I2C_Read Msg: Program Flow
State 2: Connect To I2C Bus\nError No: %i Error Level: %i\n",iError_No,
iError_Level);}


    // Initiate I2C Communicaton
    if (iError_No == Error_None)
    {
        // Initiate Communicaiton

iIOCTL_BBB_I2C_Handle=ioctl(iFILE_BBB_I2C_Handle,I2C_SLAVE,cDevice_I2C_Addr);

        // Error Handling
        if (iIOCTL_BBB_I2C_Handle<0)
        {
            if (iDebug_Level >=Debug_Level_1) {printf("I2C_Read Error: Failed
to Communicate With Slave Device\n");}
            // Set Error Number
            iError_No = Error_No_I2CInit;
            // Set Error Level
            iError_Level = Error_Level_Critical;
            // Return Otuput Error Level
            // return -1;
        }
    }

    // Ouput Program Flow
    if (iDebug_Level >=Debug_Level_3) {printf("I2CRead Msg: Program Flow
State 3: Connect to I2C Device\nError No: %i Error Level: %i\n",iError_No,
iError_Level);}

    // Write Data to Register
    if (iError_No == Error_None)
    {

        // Set Gyro To Register Address!
        // Set Gyro Register Value
        cI2C_Buffer_Write[0]= cRegister_Address;//Address
        cI2C_Buffer_Write[1]= cRegister_Value;//Value
        iWrite_BBB_I2C_Handle = write(iFILE_BBB_I2C_Handle,
cI2C_Buffer_Write, 2);
```

```
        // Ouput Program Flow
        if (iDebug_Level >=Debug_Level_3) {printf("I2CRead Msg: Data Value :
%#04x Written to Registry %#04x\n",cI2C_Buffer_Write[1],
cI2C_Buffer_Write[0]);}


        if (iWrite_BBB_I2C_Handle!=2)
        {
            // Error Handling
            if (iDebug_Level >=Debug_Level_1) { printf("I2C_Read Error:
Failed to Communicate With Device and/or Write To Device Register\n");}
            // Set Error Number
            iError_No = Error_No_I2CBuffWrite;
            // Set Error Level
            iError_Level = Error_Level_Critical;
            // Return Otuput Error Level
            // return -1;
        }
        else
        {
            // Power Mode Set
            if (iDebug_Level >=Debug_Level_1){printf("I2C_Read Msg: Data
Written to Gyro Register!\n");}
        }
    }
    // Close File
    close(iFILE_BBB_I2C_Handle);

    // Set Return State
    if (iError_No==0){return 0;} else {return -1;}
}

////// MAIN PROGRAM START ///////

// Check For Block Enable
if (inI2C_R_Enable[0]==true)
{

    // Check Debug Level - Read Parameter Data -Debug Level
    iDebug_Level = iSet_Debug_Level(prmDebug_InfoLevel[0]);

    // Start Line
    if (iDebug_Level >=Debug_Level_0){printf("**** Start of I2C Read S-
Function Block Execution **** \n");}

    // Check Parameters
    if (iDebug_Level >=Debug_Level_1){printf("I2C_Read Msg: Input Parameter
Data: I2CBus: %#04x I2CAddress:
%#04x\n",prmI2C_BusNo[0],prmI2C_DeviceAddress[0]);}

    // Ouput Program Flow
    if (iDebug_Level >=Debug_Level_3) {printf("I2C_Read Msg: Program Flow
State 1: Parameters Initialized \nError No: %i Error Level: %i\n",iError_No,
iError_Level);}
```

```
    // Record Device Address
    cDevice_I2C_Addr = prmI2C_DeviceAddress[0]; //0x68
    cDevice_I2C_RegAddr[0] = prmI2C_RegisterNo[0];

    if (inI2C_R_Trigger[0]==true)
    {

        // Read Device Register
        iI2C_Read=iI2C_Read_Register(cDevice_I2C_RegAddr,cI2C_Buffer);

        // Check For Error
        if (iI2C_Read<0)
        {
            // Error Handling
            if (iDebug_Level >=Debug_Level_1){printf("I2CRead Error: Failed
to Read Registery %#04x From Device\n",prmI2C_RegisterNo[0]);}
        }
        else
        {
            // Error Handling
            if (iDebug_Level >=Debug_Level_1){printf("I2CRead Msg:
Successfully Read Device Register %#04x. Value Read:
%#04x\n",prmI2C_RegisterNo[0],cI2C_Buffer[0]);}
        }

    }
    else
    {

        // Ouput Program Flow
        if (iDebug_Level >=Debug_Level_3) {printf("I2C_Read Msg: Program Flow
State 2-4: Read Trigger Disabled \nError No: %i Error Level: %i\n",iError_No,
iError_Level);}

    }

    // Ouput Program Flow
    if (iDebug_Level >=Debug_Level_3) {printf("I2C_Read Msg: Program Flow
State 6: Loop Step Complete\nError No: %i Error Level: %i\n",iError_No,
iError_Level);}

    // Check For Critical Errors and Stop Simulation
    if ((iError_No!=0)&&(iError_Level==Error_Level_Critical))
    {

        // Stop Simulation If Critical Error
        if (iDebug_Level >=Debug_Level_0) {printf("I2C_Read Msg: Critical
Error Detected; Simulation Stopping! \nError No: %i Error Level:
%i\n",iError_No, iError_Level);}

        // Output Error Message:
        if (iDebug_Level >=Debug_Level_0) {printf("I2C_Read Error Details:
%s",sI2CReadWrite_ErrMessage(iError_No));}
```

```
        // Stop Simulation
        outSimStop[0]=true;
    }
    else
    {
        // Output Registry Data
        outI2C_R_Data[0]=cI2C_Buffer[0];


        // Carry on
        outSimStop[0]=false;
    }

}

# else
# endif
/* %%%-SFUNWIZ_wrapper_Outputs_Changes_END --- EDIT HERE TO _BEGIN */
}

/*
  * Updates function
  *
  */
void BBB_Driver_I2C_Read_Update_wrapper(const boolean_T *inI2C_R_Enable,
                           const boolean_T *inI2C_R_Trigger,
                           const uint8_T *outI2C_R_Data,
                           const boolean_T *outSimStop ,
                           real_T *xD,
                           const uint8_T  *prmI2C_BusNo,  const int_T
p_width0,
                           const uint8_T  *prmI2C_DeviceAddress,  const int_T
p_width1,
                           const uint8_T  *prmI2C_RegisterNo,  const int_T
p_width2,
                            const uint8_T *prmDebug_InfoLevel, const int_T
p_width3)
{
  /* %%%-SFUNWIZ_wrapper_Update_Changes_BEGIN --- EDIT HERE TO _END */
// Run Only on Target - BeagleBoneBlack */
# ifndef MATLAB_MEX_FILE

// Initialize Debug Output Level
eDebugLevel iDebug_Level = Debug_None;

// Check Debug Level - Read Parameter Data -Debug Level
iDebug_Level = iSet_Debug_Level(prmDebug_InfoLevel[0]);

// Update Init Bit
if (xD[0]==0)
{
    // Set Init Complete
    xD[0]=1;
```

```
    // Output Update Compete
    if (iDebug_Level >=Debug_Level_1){printf("I2C_Read Msg: Program Flow
State: Init First Scan Complete\n");}
}

// Print Gap
if (iDebug_Level >=Debug_Level_0){printf("\n");}


# else

// Do Nothing


#endif
/* %%%-SFUNWIZ_wrapper_Update_Changes_END --- EDIT HERE TO _BEGIN */
}
```

13.4 S-FUNCTION BLOCK WRAPPER CODE: I2C WRITE BLOCK

```
/*
  *
  *   --- THIS FILE GENERATED BY S-FUNCTION BUILDER: 3.0 ---
  *
  *   This file is a wrapper S-function produced by the S-Function
  *   Builder which only recognizes certain fields.  Changes made
  *   outside these fields will be lost the next time the block is
  *   used to load, edit, and resave this file. This file will be overwritten
  *   by the S-function Builder block. If you want to edit this file by hand,
  *   you must change it only in the area defined as:
  *
  *        %%%-SFUNWIZ_wrapper_XXXXX_Changes_BEGIN
  *            Your Changes go here
  *        %%%-SFUNWIZ_wrapper_XXXXXX_Changes_END
  *
  *   For better compatibility with the Simulink Coder, the
  *   "wrapper" S-function technique is used.  This is discussed
  *   in the Simulink Coder User's Manual in the Chapter titled,
  *   "Wrapper S-functions".
  *
  *   Created: Sun Dec 14 05:00:00 2014
  */


/*
 * Include Files
 *
 */
#if defined(MATLAB_MEX_FILE)
#include "tmwtypes.h"
#include "simstruc_types.h"
#else
#include "rtwtypes.h"
#endif

/* %%%-SFUNWIZ_wrapper_includes_Changes_BEGIN --- EDIT HERE TO _END */
#include <math.h>
#include <stdio.h>
#include <stddef.h>
#include <time.h>
#include <stdlib.h>
#include <string.h>
//#include <inttypes.h>
//#include <errno.h>


# ifndef MATLAB_MEX_FILE

    #include <linux/i2c.h>
    #include <linux/i2c-dev.h>
    #include <sys/ioctl.h>
```

```
    #include <unistd.h>
    #include <sys/types.h>
    #include <sys/stat.h>
    #include <fcntl.h>
    //#include <simstruc.h>
# else

    #include <mex.h>
    #include <simstruc.h>

# endif


//#define S_FUNCTION_NAME BBB_Driver_Gyroscope_L3G4200D
//#define S_FUNCTION_LEVEL 2

#define I2CMode_Read 1
#define I2CMode_Write 2


// Declare Error Handling
// extern int errno;


// Define Debug Levels
typedef enum
{
        Debug_None,
    Debug_Level_0, // Basic Debug Info Output
        Debug_Level_1, // + Critical Info Only
    Debug_Level_2, // + Diagnostics Info
    Debug_Level_3, // + Program Flow Info
    Debug_Level_4, // + Registry Data
    Debug_Level_5  // + TBD
}eDebugLevel;

// Define Error Number
typedef enum
{
    Error_None=0,
    Error_No_I2CComm,
    Error_No_I2CInit,
    Error_No_I2CBuffSet,
    Error_No_I2CBuffRead,
    Error_No_I2CBuffSync,
    Error_No_I2CBuffWrite,
    Error_No_I2CBuffValidate,
    Error_No_ParamOutofRange
}eError_No;

// Initialize Error
// eError_No iError_No = Error_None;

// Define Error Severity Levels
```

```
typedef enum
{
    Error_Level_OK=0,
    Error_Level_Critical, // (Stop Simulation)
    Error_Level_Warning
}eError_Level;

// Initialize Error Level
// eError_Level iError_Level = Error_Level_OK;
/* %%%-SFUNWIZ_wrapper_includes_Changes_END --- EDIT HERE TO _BEGIN */
#define u_width 1
#define y_width 1
/*
 * Create external references here.
 *
 */
/* %%%-SFUNWIZ_wrapper_externs_Changes_BEGIN --- EDIT HERE TO _END */
/* extern double func(double a); */
const char *sI2CReadWrite_ErrMessage(eError_No iErrNo)
{
    //Set Error Message Based on Error Number
    switch(iErrNo)
    {
       case  Error_None:
            return("No Error Active. System A-OK!\n");
            break;
        case  Error_No_I2CComm:
            return("Failed to Open I2C Bus\n - Check Bus Number\n - Check
SCL/SCA Lines\n"
                   " - Check Device\n - Check Device Tree on BBB\n");
            break;
        case Error_No_I2CInit:
            return("Failed to Communicate With Slave Device\n - Check Bus
Number\n - Check SCL/SCA Lines\n"
                   " - Check Device Address\n - Check Device Tree on BBB\n");
            break;
        case Error_No_I2CBuffSet:
            return("Failed to Set Buffer Read Range\n - Check Bus Number\n -
Check SCL/SCA Lines\n"
                   " - Check Device Address\n - Check Device Tree on BBB\n");
            break;
        case Error_No_I2CBuffRead:
            return("Failed to Read Data in Registry Buffer\n - Check Bus
Number\n - Check SCL/SCA Lines\n"
                   " - Check Device Address\n - Check Device Tree on BBB\n");
            break;
        case Error_No_I2CBuffSync:
            return("Failed to Validate Registry Data - Inconsistent Data
Stream\n"
                    " - Check For Multiple Master Access to Slave Device on
I2C Bus Number\n"
                    " - Check SCL/SCA Lines\n - Check Device Address\n -
Check Device Tree on BBB\n");
            break;
        case Error_No_I2CBuffWrite:
            return("Failed to Write Data to Device Registry\n - Check Bus
Traffic/Interference\n"
```

```
                    " - Check Bus Number\n - Check SCL/SCA Lines\n - Check
Device Address\n - Check Device Tree on BBB\n");
            break;
    case Error_No_I2CBuffValidate:
            return("Failed to Validate Data Written To Registry\n - Check Bus
Traffic/Interference\n"
                    " - Check Bus Number\n - Check SCL/SCA Lines\n - Check
Device Address\n - Check Device Tree on BBB\n");
            break;
        case Error_No_ParamOutofRange:
            return("Parameter Out Of Range\n - Check Block Parameter Input
Options\n");
            break;
        default:
            // Error -Return Default
            return("Error Code Not Recognized\n");
            break;


    }
}
// Set Debug Level
int iSet_Debug_Level(eDebugLevel iDebug_Level_X)
{

    switch(iDebug_Level_X)
    {
        case  1:
            return (Debug_None);
            break;
        case  2:
            return (Debug_Level_0);
            break;
        case  3:
            return (Debug_Level_1);
            break;
        case  4:
            return (Debug_Level_2);
            break;
        case  5:
            return (Debug_Level_3);
            break;
        case  6:
            return (Debug_Level_4);
            break;
        case  7:
            return (Debug_Level_5);
            break;
        default:
            return (Debug_None);
            break;
    }
}
/* %%%-SFUNWIZ_wrapper_externs_Changes_END --- EDIT HERE TO _BEGIN */

/*
```

```
 * Output functions
 *
 */
void BBB_Driver_I2C_Write_Outputs_wrapper(const boolean_T *inI2C_W_Enable,
                          const uint8_T *inI2C_W_Value,
                          const boolean_T *inI2C_W_Trigger,
                          uint8_T *outI2C_W_Data,
                          boolean_T *outSimStop ,
                  const real_T  *xD,
                          const uint8_T  *prmI2C_BusNo, const int_T
p_width0,
                          const uint8_T  *prmI2C_DeviceAddress, const int_T
p_width1,
                          const uint8_T  *prmI2C_RegisterNo, const int_T
p_width2,
                          const uint8_T  *prmDebug_InfoLevel, const int_T
p_width3)
{
/* %%%-SFUNWIZ_wrapper_Outputs_Changes_BEGIN --- EDIT HERE TO _END */
// Define Output Data - Simulink

// Run Only on Target - BeagleBoneBlack */
# ifndef MATLAB_MEX_FILE

// Define File Sys Pointer - For I2C Bus#1 -> Make Parameter
int iFILE_BBB_I2C_Handle=0;
int iIOCTL_BBB_I2C_Handle=0;
int iSYNC_BBB_I2C_Handle=0;
int iWrite_BBB_I2C_Handle=0;
// Function Outputs
int iI2C_Read=0;
int iI2C_Write=0;

//Define Device Driver sysfs Location
// const char *sBBB_I2C_BUS1_FILE ="/dev/i2c-1";
// char *sBBB_I2C_BUS1_FILE ="/dev/i2c-";
char sBBB_I2C_BUS1_FILE[100]="";

// Define I2C Parameters
int iI2C_BytesRead=0;
// Define I2C Buffer Size
const int iI2C_BufferSize=1;
// Define I2C Buffer Dump Variable
char cI2C_Buffer[1]={0x00}; //unsigned
// Define I2C Buffer Dump Start Position. Note: Value = Start Address w/ MSB
Set To "1"
char cI2C_Buffer_Sync[1]={0x00}; // Start at reg 0x00
// Define Write Buffer
char cI2C_Buffer_Write[2]={0x00};
// Define I2C Buffer Status
char cI2C_Buffer_Status=0;//unsigned


// Define Loop Counters
int iLoop=0;
int iLoop_Counter;
```

```
// Define Device Address
char cDevice_I2C_Addr_Para=0x00;
char cDevice_I2C_Addr=0X00;
char cDevice_I2C_RegAddr[1]={0x00};
char cDevice_I2C_WriteValue=0x00;

// Initialize Error
eError_No iError_No = Error_None;
// Initialize Error Level
eError_Level iError_Level = Error_Level_OK;
// Initialize Debug Output Level
eDebugLevel iDebug_Level = Debug_None;

// Define Function iI2C_Read_Register
int iI2C_Read_Register(char *cRegister_Address, char *cRegister_Value)
{

    // Create Handle to SYSFS File Stream
    sprintf(sBBB_I2C_BUS1_FILE,"/dev/i2c-%u",prmI2C_BusNo[0]);

    // Check I2C Bus Concatenation
    if (iDebug_Level >=Debug_Level_2)  { printf("I2C_Write Msg: I2C Bus File
Location: %s\n",sBBB_I2C_BUS1_FILE); }

    // Open Data IO Stream
    iFILE_BBB_I2C_Handle = open(sBBB_I2C_BUS1_FILE, O_RDWR);

    // Confirm I2C Bus Open
    if (iFILE_BBB_I2C_Handle<0)
    {
        // Error Handling
        if (iDebug_Level >=Debug_Level_1)  { printf("I2C_Write Error: Failed
to Open I2C Bus - Check Device/Device Tree\n"); }
        // Set Error Number
        iError_No = Error_No_I2CComm;
        // Set Error Level
        iError_Level = Error_Level_Critical;
        // Return Otuput Error Level
        // return -1;
    }

    // Ouput Program Flow
    if (iDebug_Level >=Debug_Level_3) {printf("I2C_Write Msg: Program Flow
State 2: Connect To I2C Bus\nError No: %i Error Level: %i\n",iError_No,
iError_Level);}

    // Initiate I2C Communicaton
    if (iError_No == Error_None)
    {
        // Initiate Communicaiton

iIOCTL_BBB_I2C_Handle=ioctl(iFILE_BBB_I2C_Handle,I2C_SLAVE,cDevice_I2C_Addr);

        // Error Handling
        if (iIOCTL_BBB_I2C_Handle<0)
```

```
        {
            if (iDebug_Level >=Debug_Level_1) {printf("I2C_Write Error:
Failed to Communicate With Slave Device: %#04x\n",cDevice_I2C_Addr);}
            // Set Error Number
            iError_No = Error_No_I2CInit;
            // Set Error Level
            iError_Level = Error_Level_Critical;
            // Return Otuput Error Level
            // return -1;
        }
    }

    // Ouput Program Flow
    if (iDebug_Level >=Debug_Level_3) {printf("I2C_Write Msg: Program Flow
State 3: Connect to I2C Device\nError No: %i Error Level: %i\n",iError_No,
iError_Level);}

    // Set Buffer Read Range
    if (iError_No == Error_None)
    {
        iWrite_BBB_I2C_Handle =
write(iFILE_BBB_I2C_Handle,cRegister_Address,1); //cI2C_Buffer_Sync
        // Error Handling
        if (iWrite_BBB_I2C_Handle !=1)
        {
            // Error Handling
            if (iDebug_Level >=Debug_Level_1) { printf("I2C_Write Error:
Failed to Write/Communicate With Device and/or Set Buffer Read Range\n");}
            // Set Error Number
            iError_No = Error_No_I2CBuffSet;
            // Set Error Level
            iError_Level = Error_Level_Critical;
            // Return Otuput Error Level
            // return -1;
        }
        else
        {
            if (iDebug_Level >=Debug_Level_1) { printf("I2C_Write Msg: Buffer
Range Set\n");}
        }
    }

    // Ouput Program Flow
    if (iDebug_Level >=Debug_Level_3) {printf("I2C_Write Msg: Program Flow
State 4: Write to Device Registry\nError No: %i Error Level: %i\n",iError_No,
iError_Level);}

    // Read Buffer Data
    if (iError_No == Error_None)
    {
        // Read and Store Registry Data in Buffer
        iI2C_BytesRead =
read(iFILE_BBB_I2C_Handle,cRegister_Value,1);//cI2C_Buffer
sizeof(cRegister_Value));
        // Close File
        close(iFILE_BBB_I2C_Handle);
```

```
        // Error Handling - Confirm Buffer Size
        if (iI2C_BytesRead<0)
        {
            // Error on Buffer Read!
            if (iDebug_Level >=Debug_Level_1) { printf("I2C_Write Error:
Failed to Read Device Register: %#04x Bytes Read:
%i\n",cRegister_Address[0],iI2C_BytesRead );}
            // Set Error Number
            iError_No = Error_No_I2CBuffRead;
            // Set Error Level
            iError_Level = Error_Level_Critical;
            // Return Otuput Error Level
            // return -1;
        }
        else
        {
            // Registry Data Read!
            if (iDebug_Level >=Debug_Level_1) { printf("I2C_Write Msg: Device
Register Read. Bytes Read: %i\n",iI2C_BytesRead );}

        }
    }

    // Set Return State
    if (iError_No==0){return 0;} else {return -1;}


}

// Define Function iGryro_I2CWriteBuff
int iI2C_Write_Register(char cRegister_Address, char cRegister_Value)
{

    // Create Handle to SYSFS File Stream
    sprintf(sBBB_I2C_BUS1_FILE,"/dev/i2c-%u",prmI2C_BusNo[0]);

    // Check I2C Bus Concatenation
    if (iDebug_Level >=Debug_Level_2)  { printf("I2C_Write Msg: I2C Bus File
Location: %s\n",sBBB_I2C_BUS1_FILE); }

    // Open Data IO Stream
    iFILE_BBB_I2C_Handle = open(sBBB_I2C_BUS1_FILE, O_RDWR);

    // Confirm I2C Bus Open
    if (iFILE_BBB_I2C_Handle<0)
    {
        // Error Handling
        if (iDebug_Level >=Debug_Level_1)  { printf("I2C_Write Error: Failed
to Open I2C Bus - Check Device/Device Tree\n"); }
        // Set Error Number
        iError_No = Error_No_I2CComm;
        // Set Error Level
        iError_Level = Error_Level_Critical;
        // Return Otuput Error Level
```

```
        // return -1;
    }

    // Ouput Program Flow
    if (iDebug_Level >=Debug_Level_3) {printf("I2C_Write Msg: Program Flow
State 2: Connect To I2C Bus\nError No: %i Error Level: %i\n",iError_No,
iError_Level);}


    // Initiate I2C Communicaton
    if (iError_No == Error_None)
    {
        // Initiate Communicaiton

iIOCTL_BBB_I2C_Handle=ioctl(iFILE_BBB_I2C_Handle,I2C_SLAVE,cDevice_I2C_Addr);

        // Error Handling
        if (iIOCTL_BBB_I2C_Handle<0)
        {
            if (iDebug_Level >=Debug_Level_1) {printf("I2C_Write Error:
Failed to Communicate With Slave Device\n");}
            // Set Error Number
            iError_No = Error_No_I2CInit;
            // Set Error Level
            iError_Level = Error_Level_Critical;
            // Return Otuput Error Level
            // return -1;
        }
    }

    // Ouput Program Flow
    if (iDebug_Level >=Debug_Level_3) {printf("I2CReadWrite Msg: Program Flow
State 3: Connect to I2C Device\nError No: %i Error Level: %i\n",iError_No,
iError_Level);}

    // Write Data to Register
    if (iError_No == Error_None)
    {

        // Set Gyro To Register Address!
        // Set Gyro Register Value
        cI2C_Buffer_Write[0]= cRegister_Address;//Address
        cI2C_Buffer_Write[1]= cRegister_Value;//Value
        iWrite_BBB_I2C_Handle = write(iFILE_BBB_I2C_Handle,
cI2C_Buffer_Write, 2);

        // Ouput Program Flow
        if (iDebug_Level >=Debug_Level_3) {printf("I2CReadWrite Msg: Data
Value : %#04x Written to Registry %#04x\n",cI2C_Buffer_Write[1],
cI2C_Buffer_Write[0]);}


        if (iWrite_BBB_I2C_Handle!=2)
        {
            // Error Handling
```

```
            if (iDebug_Level >=Debug_Level_1) { printf("I2C_Write Error:
Failed to Communicate With Device and/or Write To Device Register\n");}
            // Set Error Number
            iError_No = Error_No_I2CBuffWrite;
            // Set Error Level
            iError_Level = Error_Level_Critical;
            // Return Otuput Error Level
            // return -1;
        }
        else
        {
            // Power Mode Set
            if (iDebug_Level >=Debug_Level_1){printf("I2C_Write Msg: Data
Written to Device Register!\n");}
        }
    }
    // Close File
    close(iFILE_BBB_I2C_Handle);

    // Set Return State
    if (iError_No==0){return 0;} else {return -1;}
}

//*****************************//
//*****************************//
////// MAIN PROGRAM START ///////
//*****************************//
//*****************************//

// Check For Block Enable
if (inI2C_W_Enable[0]==true)
{

    // Check Debug Level - Read Parameter Data -Debug Level
    iDebug_Level = iSet_Debug_Level(prmDebug_InfoLevel[0]);

    // Start Line
    if (iDebug_Level >=Debug_Level_0){printf("**** Start of I2C Write S-
Function Block Execution **** \n");}

    // Check Parameters
    if (iDebug_Level >=Debug_Level_1){printf("I2C_Write Msg: Input Parameter
Data: I2CBus: %#04x I2CAddress:
%#04x\n",prmI2C_BusNo[0],prmI2C_DeviceAddress[0]);}

    // Ouput Program Flow
    if (iDebug_Level >=Debug_Level_3) {printf("I2C_Write Msg: Program Flow
State 1: Parameters Initialized \nError No: %i Error Level: %i\n",iError_No,
iError_Level);}

    // Record Device Address
    cDevice_I2C_Addr = prmI2C_DeviceAddress[0]; //0x68
    cDevice_I2C_RegAddr[0] = prmI2C_RegisterNo[0];
    cDevice_I2C_WriteValue=inI2C_W_Value[0];
```

```
    if (inI2C_W_Trigger[0]==true)
    {

        // Write To Device Register
        iI2C_Write_Register(cDevice_I2C_RegAddr[0],cDevice_I2C_WriteValue);//
cDevice_I2C_WriteValue);

        // Check For Error
        if (iI2C_Write<0)
        {
            // Error Handling
            if (iDebug_Level >=Debug_Level_1){printf("I2CWrite Error: Failed
to Write Value: %#04x to Device Register:
%#04x\n",inI2C_W_Value[0],prmI2C_RegisterNo[0]);}
        }
        else
        {
            // Error Handling
            if (iDebug_Level >=Debug_Level_1){printf("I2CWrite Error:
Sucessfully Wrote Value: %#04x to Device Register:
%#04x\n",inI2C_W_Value[0],prmI2C_RegisterNo[0]);}
        }

        // Read Back Data To Validate Write Functionality
        iI2C_Read=iI2C_Read_Register(cDevice_I2C_RegAddr,cI2C_Buffer);

        // Check For Error
        if (iI2C_Read<0)
        {
            // Error Handling
            if (iDebug_Level >=Debug_Level_1){printf("I2CRead Error:
Validation Failure! Failed to Read Registery %#04x From
Device\n",prmI2C_RegisterNo[0]);}
        }
        else
        {
            // Error Handling
            if (iDebug_Level >=Debug_Level_1){printf("I2CRead Msg:
Successfully Read Device Register %#04x. Value Read: %#04x For
Validation\n",prmI2C_RegisterNo[0],cI2C_Buffer[0]);}
        }

        // Validate Data
        if (cI2C_Buffer[0]!=inI2C_W_Value[0])
        {
            // Error Handling
            if (iDebug_Level >=Debug_Level_1) { printf("I2CWrite Error:
Failed to Validate Data Written to Registery: %#04x !=
%#04x\n",inI2C_W_Value[0],cI2C_Buffer[0]);}
            // Set Error Number
            iError_No = Error_No_I2CBuffValidate;
            // Set Error Level
            iError_Level = Error_Level_Critical;
            // Return Otuput Error Level
            // return -1;
        }
```

```
    }
    else
    {

        // Ouput Program Flow
        if (iDebug_Level >=Debug_Level_3) {printf("I2C_Write Msg: Program
Flow State 2-4: Read Trigger Disabled \nError No: %i Error Level:
%i\n",iError_No, iError_Level);}

    }

    // Ouput Program Flow
    if (iDebug_Level >=Debug_Level_3) {printf("I2C_Write Msg: Program Flow
State 6: Loop Step Complete\nError No: %i Error Level: %i\n",iError_No,
iError_Level);}

    // Check For Critical Errors and Stop Simulation
    if ((iError_No!=0)&&(iError_Level==Error_Level_Critical))
    {

        // Stop Simulation If Critical Error
        if (iDebug_Level >=Debug_Level_0) {printf("I2C_Write Msg: Critical
Error Detected; Simulation Stopping! \nError No: %i Error Level:
%i\n",iError_No, iError_Level);}

        // Output Error Message:
        if (iDebug_Level >=Debug_Level_0) {printf("I2C_Write Error Details:
%s",sI2CReadWrite_ErrMessage(iError_No));}


        // Stop Simulation
        outSimStop[0]=true;
    }
    else
    {
        // Output Registry Data
        outI2C_W_Data[0]=cI2C_Buffer[0];


        // Carry on
        outSimStop[0]=false;
    }

}

# else
# endif
/* %%%-SFUNWIZ_wrapper_Outputs_Changes_END --- EDIT HERE TO _BEGIN */
}

/*
  * Updates function
  *
```

```
  */
void BBB_Driver_I2C_Write_Update_wrapper(const boolean_T *inI2C_W_Enable,
                          const uint8_T *inI2C_W_Value,
                          const boolean_T *inI2C_W_Trigger,
                          const uint8_T *outI2C_W_Data,
                          const boolean_T *outSimStop ,
                          real_T *xD,
                          const uint8_T  *prmI2C_BusNo,  const int_T
p_width0,
                          const uint8_T  *prmI2C_DeviceAddress,  const int_T
p_width1,
                          const uint8_T  *prmI2C_RegisterNo,  const int_T
p_width2,
                           const uint8_T *prmDebug_InfoLevel, const int_T
p_width3)
{
  /* %%%-SFUNWIZ_wrapper_Update_Changes_BEGIN --- EDIT HERE TO _END */
// Run Only on Target - BeagleBoneBlack */
# ifndef MATLAB_MEX_FILE

// Initialize Debug Output Level
eDebugLevel iDebug_Level = Debug_None;

// Check Debug Level - Read Parameter Data -Debug Level
iDebug_Level = iSet_Debug_Level(prmDebug_InfoLevel[0]);


// Update Init Bit
if (xD[0]==0)
{
    // Set Init Complete
    xD[0]=1;

    // Output Update Compete
    if (iDebug_Level >=Debug_Level_1){printf("I2C_Write Msg: Program Flow
State: Init First Scan Complete\n");}
}

// Print Gap
if (iDebug_Level >=Debug_Level_0){printf("\n");}


# else

// Do Nothing


#endif
/* %%%-SFUNWIZ_wrapper_Update_Changes_END --- EDIT HERE TO _BEGIN */
}
```

13.5 S-FUNCTION BLOCK WRAPPER CODE: ANALOG INPUT READ BLOCK

```
/*
  *
  *   --- THIS FILE GENERATED BY S-FUNCTION BUILDER: 3.0 ---
  *
  *   This file is a wrapper S-function produced by the S-Function
  *   Builder which only recognizes certain fields.  Changes made
  *   outside these fields will be lost the next time the block is
  *   used to load, edit, and resave this file. This file will be overwritten
  *   by the S-function Builder block. If you want to edit this file by hand,
  *   you must change it only in the area defined as:
  *
  *        %%%-SFUNWIZ_wrapper_XXXXX_Changes_BEGIN
  *            Your Changes go here
  *        %%%-SFUNWIZ_wrapper_XXXXXX_Changes_END
  *
  *   For better compatibility with the Simulink Coder, the
  *   "wrapper" S-function technique is used.  This is discussed
  *   in the Simulink Coder User's Manual in the Chapter titled,
  *   "Wrapper S-functions".
  *
  *   Created: Sun Dec 14 04:58:47 2014
  */


/*
 * Include Files
 *
 */
#if defined(MATLAB_MEX_FILE)
#include "tmwtypes.h"
#include "simstruc_types.h"
#else
#include "rtwtypes.h"
#endif

/* %%%-SFUNWIZ_wrapper_includes_Changes_BEGIN --- EDIT HERE TO _END */
#include <math.h>
#include <stdio.h>
#include <stddef.h>
#include <time.h>

# ifdef MATLAB_MEX_FILE
    #include <mex.h>
    #include <simstruc.h>
# endif

// Define Debug Levels
typedef enum
{
        Debug_None,
    Debug_Level_0, // Basic Debug Info Output
```

```
        Debug_Level_1, // + Critical Info Only
    Debug_Level_2, // + Diagnostics Info
    Debug_Level_3, // + Program Flow Info
    Debug_Level_4, // + Registry Data
    Debug_Level_5  // + TBD
}eDebugLevel;


// Define Error Number
typedef enum
{
    Error_None=0,
    Error_No_SYSFSOpen,
    Error_No_SYSFSSeek,
    Error_No_SYSFSScan,
    Error_No_SYSFSClose,
    Error_No_ParamOutofRange
}eError_No;

// Initialize Error
// eError_No iError_No = Error_None;

// Define Error Severity Levels
typedef enum
{
    Error_Level_OK=0,
    Error_Level_Critical, // (Stop Simulation)
    Error_Level_Warning
}eError_Level;
/* %%%-SFUNWIZ_wrapper_includes_Changes_END --- EDIT HERE TO _BEGIN */
#define u_width 1
#define y_width 1
/*
 * Create external references here.
 *
 */
/* %%%-SFUNWIZ_wrapper_externs_Changes_BEGIN --- EDIT HERE TO _END */
/* extern double func(double a); */
/* %%%-SFUNWIZ_wrapper_externs_Changes_END --- EDIT HERE TO _BEGIN */

/*
 * Output functions
 *
 */
void BBB_Driver_ADC_Read_Outputs_wrapper(const boolean_T *inADC_Read_Enable,
                          const boolean_T *inADC_Read_Trigger,
                          uint32_T *outADC_Voltage,
                          boolean_T *outSimStop ,
                  const real_T  *xD,
                          const uint8_T  *prmAINPin, const int_T  p_width0,
                          const boolean_T  *prmRunCalibration, const int_T
p_width1,
                          const uint16_T  *prmDebug_InfoLevel, const int_T
p_width2)
{
/* %%%-SFUNWIZ_wrapper_Outputs_Changes_BEGIN --- EDIT HERE TO _END */
```

```
// Define Output Data - Simulink

// Run Only on Target - BeagleBoneBlack */
# ifndef MATLAB_MEX_FILE

// Declare and Initialize Error
eError_No iError_No = Error_None;

// Declare and Initialize Error Level
eError_Level iError_Level = Error_Level_OK;

// Define File Sys Pointer - Note Code Will Use fopen vs. open
static FILE *fBBB_Handle = NULL;
char sBBB_ADC_AINX[64]="";
char cBBB_ADC_Pin="";

// Define Internal Variable Data
float dBBB_ADC_Volt_Val=0;
float dBBB_ACC_Val=0;
float const dBBB_ACC_Val_Nominal=1666+(-0.06006*333);
float dBBB_ADC_Volt_CalibOffset=0;

// Initialize Debug Output Level
eDebugLevel iDebug_Level = Debug_None;

/*************************************/
// Accelerometer Error Message Output
/*************************************/
const char *sAccel_ErrMessage(eError_No iAccel_ErrNo)
{
    //Set Error Message Based on Error Number
    switch(iAccel_ErrNo)
    {
       case  Error_None:
            return("No Error Active. System A-OK!\n");
            break;
        case  Error_No_SYSFSOpen:
            return("Failed to Open I2C Bus\n - Check Bus Number\n - Check
SCL/SCA Lines\n"
                   " - Check Device\n - Check Device Tree on BBB\n");
            break;
        case  Error_No_SYSFSSeek:
            return("Failed to Open I2C Bus\n - Check Bus Number\n - Check
SCL/SCA Lines\n"
                   " - Check Device\n - Check Device Tree on BBB\n");
            break;
        case Error_No_SYSFSScan:
            return("Failed to Communicate With Slave Device\n - Check Bus
Number\n - Check SCL/SCA Lines\n"
                   " - Check Device Address\n - Check Device Tree on BBB\n");
            break;
        case Error_No_SYSFSClose:
            return("Failed to Set Buffer Read Range\n - Check Bus Number\n -
Check SCL/SCA Lines\n"
                   " - Check Device Address\n - Check Device Tree on BBB\n");
```

```
            break;
        case Error_No_ParamOutofRange:
            return("Parameter Out Of Range\n - Check Block Parameter Input
Options\n");
            break;
        default:
            // Error -Return Default
            return("Error Code Not Recognized\n");
            break;
    }
}

/*****************************************/
/*****************************************/
// Main Program Start
/*****************************************/
/*****************************************/

// Read Parameter Data -Debug Level
switch(prmDebug_InfoLevel[0])
{
    case  1:
        iDebug_Level=Debug_None;
        break;
    case  2:
        iDebug_Level=Debug_Level_0;
        break;
    case  3:
        iDebug_Level=Debug_Level_1;
        break;
    case  4:
        iDebug_Level=Debug_Level_2;
        break;
    case  5:
        iDebug_Level=Debug_Level_3;
        break;
    case  6:
        iDebug_Level=Debug_Level_4;
        break;
    case  7:
        iDebug_Level=Debug_Level_5;
        break;
    default:
        iDebug_Level=Debug_None;
        break;
}



// Start Line
if (iDebug_Level >=Debug_Level_0){printf("**** Start of ADC Read S-Function
Block Execution **** \n");}

// Write Read Raw Voltage Data
if (iError_No == Error_None)
{
```

```
    // Concatenate SYSFS File I/O Name
    sprintf(sBBB_ADC_AINX,"/sys/bus/platform/drivers/bone-iio-
helper/helper.15/AIN%d",prmAINPin[0]-1);

    if (iDebug_Level >=Debug_Level_1)  { printf("ADCRead Msg: ADC SYSFS
Path:\n %s\n",sBBB_ADC_AINX); }

    // Open Streams
    fBBB_Handle = fopen(sBBB_ADC_AINX, "r");

// Confirm SYSFS IOStream Open
    if (sBBB_ADC_AINX==NULL)
    {
        // Error Handling
        if (iDebug_Level >=Debug_Level_1)  { printf("ADCRead Error: Failed To
Open I/O File Stream - Check ""cape-bone-iio"" Device Tree Active\n"); }
        // Set Error Number
        iError_No = Error_No_SYSFSOpen;
        // Set Error Level
        iError_Level = Error_Level_Critical;
        // Return Otuput Error Level
        // return -1;
    }
}

// Ouput Program Flow
if (iDebug_Level >=Debug_Level_3) {printf("ADCRead Msg: Program Flow State 1:
Open SYSFS File I/O Stream\nError No: %i Error Level: %i\n",iError_No,
iError_Level);}


// Write Read Raw Voltage Data
if (iError_No == Error_None)
{
    // Read ADC Voltage
    if (fseek(fBBB_Handle, 0, SEEK_SET)!=0)
    {
        // Error Handling
        if (iDebug_Level >=Debug_Level_1)  { printf("ADCRead Error:
""fseek()"" Failed To Set Position Indicator in I/O File Stream\n"); }
        // Set Error Number
        iError_No = Error_No_SYSFSSeek;
        // Set Error Level
        iError_Level = Error_Level_Critical;
    }
    if (fscanf(fBBB_Handle, "%f", &dBBB_ADC_Volt_Val)!=1)
    {
        // Error Handling
        if (iDebug_Level >=Debug_Level_1)  { printf("ADCRead Error:
""fscan()"" Failed To Read Data From I/O File Stream\n"); }
        // Set Error Number
        iError_No = Error_No_SYSFSScan;
        // Set Error Level
        iError_Level = Error_Level_Critical;
```

```
    }


}

// Ouput Program Flow
if (iDebug_Level >=Debug_Level_3) {printf("ADCRead Msg: Program Flow State 2:
Seek & Scan Data From File I/O Stream\nError No: %i Error Level:
%i\n",iError_No, iError_Level);}


// Calibrate Voltage
if (iError_No == Error_None)
{
    // Calibrate Voltage

}

// Ouput Program Flow
if (iDebug_Level >=Debug_Level_3) {printf("ADCRead Msg: Program Flow State 3:
Calibrate Voltage Data \nError No: %i Error Level: %i\n",iError_No,
iError_Level);}

// Close File on Exit
if (fclose(fBBB_Handle)!=0)
{
    // Error Handling
    if (iDebug_Level >=Debug_Level_1)  { printf("ADCRead Error: ""fclose()""
Failed To Properly Close Data From I/O File Stream\n"); }
    // Set Error Number
    iError_No = Error_No_SYSFSClose;
    // Set Error Level
    iError_Level = Error_Level_Critical;
}

// Ouput Program Flow
if (iDebug_Level >=Debug_Level_3) {printf("ADCRead Msg: Program Flow State 4:
Close SYSFS File I/O Stream\nError No: %i Error Level: %i\n",iError_No,
iError_Level);}


// Print Voltage Data
if (iDebug_Level >=Debug_Level_0)  { printf("AIN%d Voltage
(mV):%f\n",prmAINPin[0]-1,dBBB_ADC_Volt_Val);}

// Check For Critical Errors and Stop Simulation
if ((iError_No!=0)&&(iError_Level==Error_Level_Critical))
{

    // Stop Simulation If Critical Error
    if (iDebug_Level >=Debug_Level_0) {printf("ADCRead Msg: Critical Error
Detected; Simulation Stopping! \nError No: %i Error Level: %i\n",iError_No,
iError_Level);}
```

```
    // Output Error Message:
    if (iDebug_Level >=Debug_Level_0) {printf("ADCRead Error Details:
%s",sAccel_ErrMessage(iError_No));}


    // Stop Simulation
    outSimStop[0]=true;
}
else
{
    // Carry on
    outSimStop[0]=false;

    // Transfer Data to Outputs!
    outADC_Voltage[0]=(unsigned long)dBBB_ADC_Volt_Val;
}

# else


# endif
/* %%%-SFUNWIZ_wrapper_Outputs_Changes_END --- EDIT HERE TO _BEGIN */
}

/*
  * Updates function
  *
  */
void BBB_Driver_ADC_Read_Update_wrapper(const boolean_T *inADC_Read_Enable,
                          const boolean_T *inADC_Read_Trigger,
                          const uint32_T *outADC_Voltage,
                          const boolean_T *outSimStop ,
                          real_T *xD,
                          const uint8_T  *prmAINPin,  const int_T  p_width0,
                          const boolean_T  *prmRunCalibration,  const int_T
p_width1,
                           const uint16_T *prmDebug_InfoLevel, const int_T
p_width2)
{
  /* %%%-SFUNWIZ_wrapper_Update_Changes_BEGIN --- EDIT HERE TO _END */
// Run Only on Target - BeagleBoneBlack */
# ifndef MATLAB_MEX_FILE

// Execute on First Scan
if (xD[0]==0)
{
    // Set First Scan Bit
    xD[0]=1;
}

// Print Gap
if (prmDebug_InfoLevel[0]>=Debug_Level_0){printf("\n");}


// Parameter Definition
```

```
// xD[0]= First Scan / Calibration Complete
// xD[1]= Calibration Sample Quantity
// xD[2]= X Axis Voltage Total Sum
// xD[3]= Y Axis Voltage Total Sum
// xD[4]= Z Axis Voltage Total Sum

// // Execute on First Scan
// if (prmRunCalibration[0]==true)
// {
//     if (xD[0]<=xD[1])
//     {
//         // Sum Raw Voltage Data
//         xD[2]=xD[2]+dBBB_ADC_Volt_Val_X;
//         xD[3]=xD[3]+dBBB_ADC_Volt_Val_Y;
//         xD[4]=xD[4]+dBBB_ADC_Volt_Val_Z;
//     }
//     else
//     {
//        //Calculate Average OffSet Voltage
//        xD[2]=xD[2]/xD[1]
//        // NOTE - 0g Varies based on sensitiviy - 2V = 1V @ 0g
//     }
// }
//
//     // Set Calibration Complete / First Scan Bit
//     xD[0]=1;


// Read Selected AIN Pin Data -Debug Level
// switch(prmAINPin[0])
// {
//     case  1:
//         cBBB_ADC_Pin="0";
//         break;
//     case  2:
//         cBBB_ADC_Pin="1";
//         break;
//     case  3:
//         cBBB_ADC_Pin="2";
//         break;
//     case  4:
//         cBBB_ADC_Pin="3";
//         break;
//     case  5:
//         cBBB_ADC_Pin="4";
//         break;
//     case  6:
//         cBBB_ADC_Pin="5";
//         break;
//     case  7:
//         cBBB_ADC_Pin="6";
//         break;
//     case  8:
//         cBBB_ADC_Pin="7";
//         break;
//     default:
```

```
//          cBBB_ADC_Pin="";
//          // Error Handling
//          if (iDebug_Level >=Debug_Level_1)  { printf("ADCRead Error:
Requested AIN Pin Not Recognized\n"); }
//          // Set Error Number
//          iError_No = Error_No_ParamOutofRange;
//          // Set Error Level
//          iError_Level = Error_Level_Critical;
//          break;
// }




# endif
/* %%%-SFUNWIZ_wrapper_Update_Changes_END --- EDIT HERE TO _BEGIN */
}
```

13.6 S-FUNCTION BLOCK WRAPPER CODE: USR LED CONTROL BLOCK

```
/*
  *
  *   --- THIS FILE GENERATED BY S-FUNCTION BUILDER: 3.0 ---
  *
  *   This file is a wrapper S-function produced by the S-Function
  *   Builder which only recognizes certain fields.  Changes made
  *   outside these fields will be lost the next time the block is
  *   used to load, edit, and resave this file. This file will be overwritten
  *   by the S-function Builder block. If you want to edit this file by hand,
  *   you must change it only in the area defined as:
  *
  *        %%%-SFUNWIZ_wrapper_XXXXX_Changes_BEGIN
  *            Your Changes go here
  *        %%%-SFUNWIZ_wrapper_XXXXXX_Changes_END
  *
  *   For better compatibility with the Simulink Coder, the
  *   "wrapper" S-function technique is used.  This is discussed
  *   in the Simulink Coder User's Manual in the Chapter titled,
  *   "Wrapper S-functions".
  *
  *   Created: Sun Dec 14 05:03:39 2014
  */


/*
 * Include Files
 *
 */
#if defined(MATLAB_MEX_FILE)
#include "tmwtypes.h"
#include "simstruc_types.h"
#else
#include "rtwtypes.h"
#endif

/* %%%-SFUNWIZ_wrapper_includes_Changes_BEGIN --- EDIT HERE TO _END */
#include <math.h>

/* Include only when running on target hardware */
# ifndef MATLAB_MEX_FILE
    /* #include <iostream.h> */
    #include <stdio.h>
    #include <unistd.h>
    #include <string.h>
# endif


#define MAX_BUF 100


typedef enum
```

```
{
    BBB_USRLED0=1,
    BBB_USRLED1=2,
    BBB_USRLED2=3,
    BBB_USRLED3=4,
}eUSRLED_LEDID;

typedef enum
{
        BBB_USRLED_OFF=0,
    BBB_USRLED_ON=1
}eUSRLED_STATE;

// Define Debug Levels
typedef enum
{
        Debug_None,
    Debug_Level_0, // Basic Debug Info Output
    Debug_Level_1, // + Critical Info Only
    Debug_Level_2, // + Diagnostics Info
    Debug_Level_3, // + Program Flow Info
    Debug_Level_4, // + Registry Data
    Debug_Level_5  // + TBD
}eDebugLevel;


// Define Error Number
typedef enum
{
    Error_None=0,
    Error_No_SYSFSOpen,
    Error_No_SYSFSRead,
    Error_No_SYSFSWrite,
    Error_No_ParamOutofRange,
    Error_No_USRLED_TriggerOn,
    Error_No_USRLED_TriggerOff
}eError_No;

// Initialize Error
// eError_No iError_No = Error_None;

// Define Error Severity Levels
typedef enum
{
    Error_Level_OK=0,
    Error_Level_Critical, // (Stop Simulation)
    Error_Level_Warning
}eError_Level;
/* %%%-SFUNWIZ_wrapper_includes_Changes_END --- EDIT HERE TO _BEGIN */
#define u_width 1
#define y_width 1
/*
 * Create external references here.
 *
 */
/* %%%-SFUNWIZ_wrapper_externs_Changes_BEGIN --- EDIT HERE TO _END */
```

```
/* extern double func(double a); */

const char *sSYSFSReadWrite_ErrMessage(eError_No iErrNo)
{
    //Set Error Message Based on Error Number
    switch(iErrNo)
    {
       case  Error_None:
            return("No Error Active. System A-OK!\n");
            break;
        case  Error_No_SYSFSOpen:
            return("Failed to Open I2C Bus\n - Check Bus Number\n - Check
SCL/SCA Lines\n"
                   " - Check Device\n - Check Device Tree on BBB\n");
            break;
        case Error_No_SYSFSRead:
            return("Failed to Communicate With Slave Device\n - Check Bus
Number\n - Check SCL/SCA Lines\n"
                   " - Check Device Address\n - Check Device Tree on BBB\n");
            break;
        case Error_No_SYSFSWrite:
            return("Failed to Set Buffer Read Range\n - Check Bus Number\n -
Check SCL/SCA Lines\n"
                   " - Check Device Address\n - Check Device Tree on BBB\n");
            break;
        case Error_No_USRLED_TriggerOn:
            return("Failed to Read Data in Registry Buffer\n - Check Bus
Number\n - Check SCL/SCA Lines\n"
                   " - Check Device Address\n - Check Device Tree on BBB\n");
            break;
        case Error_No_USRLED_TriggerOff:
            return("Failed to Validate Registry Data - Inconsistent Data
Stream\n"
                    " - Check For Multiple Master Access to Slave Device on
I2C Bus Number\n"
                    " - Check SCL/SCA Lines\n - Check Device Address\n -
Check Device Tree on BBB\n");
            break;
        case Error_No_ParamOutofRange:
            return("Parameter Out Of Range\n - Check Block Parameter Input
Options\n");
            break;
        default:
            // Error -Return Default
            return("Error Code Not Recognized\n");
            break;


    }
}
// Set Debug Level
int iSet_Debug_Level(eDebugLevel iDebug_Level_X)
{

    switch(iDebug_Level_X)
    {
```

```
        case  1:
            return (Debug_None);
            break;
        case  2:
            return (Debug_Level_0);
            break;
        case  3:
            return (Debug_Level_1);
            break;
        case  4:
            return (Debug_Level_2);
            break;
        case  5:
            return (Debug_Level_3);
            break;
        case  6:
            return (Debug_Level_4);
            break;
        case  7:
            return (Debug_Level_5);
            break;
        default:
            return (Debug_None);
            break;
    }
}
/* %%%-SFUNWIZ_wrapper_externs_Changes_END --- EDIT HERE TO _BEGIN */

/*
 * Output functions
 *
 */
void BBB_Driver_USRLED_Outputs_wrapper(const real_T *inUSRLED_Enable,
                          const boolean_T *inUSRLED_On_Trigger,
                          boolean_T *outUSRLED_State,
                          boolean_T *outSimStop ,
                  const real_T  *xD,
                          const uint8_T  *prmUSRLED_No, const int_T
p_width0,
                          const uint8_T  *prmDebug_InfoLevel, const int_T
p_width1)
{
/* %%%-SFUNWIZ_wrapper_Outputs_Changes_BEGIN --- EDIT HERE TO _END */
/* Run only on target - BeagleBoneBlack */
# ifndef MATLAB_MEX_FILE

static FILE *iFILE_BBB_USRLED_Handle = NULL;
const char *sSYSFS_USRLED="/sys/class/leds/beaglebone:green:usr";
char sSYSFS_USRLED_Brightness[MAX_BUF]="";
char sSYSFS_USRLED_Trigger[MAX_BUF]="";
char sSYSFS_USRLED_Path[MAX_BUF]="";
int iSYSFS_Write_Result=0;
int iSYSFS_Trigger_Result=0;

// Initialize USR LED State
eUSRLED_STATE iUSRLED_State;
```

```
// Initialize Error
eError_No iError_No = Error_None;
// Initialize Error Level
eError_Level iError_Level = Error_Level_OK;
// Initialize Debug Output Level
eDebugLevel iDebug_Level = Debug_None;


/*************************************/
// Function: Remove Standard Trigger
/*************************************/
int iBBB_USRLED_TriggerOff()
{
    sprintf(sSYSFS_USRLED_Trigger,"%s/trigger",sSYSFS_USRLED_Path);
    // Validate Path - Check USR LED Trigger Concatenation
    if (iDebug_Level >=Debug_Level_2)  { printf("USRLED Msg: USRLED SYSFS
Path: %s\n",sSYSFS_USRLED_Trigger); }

    if((iFILE_BBB_USRLED_Handle = fopen(sSYSFS_USRLED_Trigger, "r+")) !=
NULL)
    {
        if (fwrite("none", sizeof("none"), 1, iFILE_BBB_USRLED_Handle)<=0)
        {
            // Error Handling
            if (iDebug_Level >=Debug_Level_1)  { printf("USRLED Error: Failed
to Write to USRLED SYSFS Trigger (None) File - Check Connection to BBB\n"); }
            // Set Error Number
            iError_No = Error_No_SYSFSWrite;
            // Set Error Level
            iError_Level = Error_Level_Critical;
        }
        fclose(iFILE_BBB_USRLED_Handle);
    }
    else
    {
        // Error Handling
        if (iDebug_Level >=Debug_Level_1)  { printf("USRLED Error: Failed
Find/Open USRLED SYSFS Trigger (None) File  - Check USRLED Number/Connection
to BBB\n"); }
        // Set Error Number
        iError_No = Error_No_SYSFSOpen;
        // Set Error Level
        iError_Level = Error_Level_Critical;
    }
    // printf("iError_No= %i Error_None= %i\n",iError_No,Error_None);
    // Return Error Code
    if (iError_No!=Error_None)
    {   // Return Otuput Error Level
        // Error Handling
        if (iDebug_Level >=Debug_Level_1)  { printf("USRLED Error: - Error
Return From ""iBBB_USRLED_TriggerOff()"" Function\n");}
        // Return Error Code
        return -1;

    }
```

```
    else
    {
        return 0;
    }

}

/*************************************/
// Function: Re-Apply Standard Trigger   /
/*************************************/
int iBBB_USRLED_TriggerOn()
{
    sprintf(sSYSFS_USRLED_Trigger,"%s/trigger",sSYSFS_USRLED_Path);
    // Validate Path - Check USR LED Trigger Concatenation
    if (iDebug_Level >=Debug_Level_2)  { printf("USRLED Msg: USRLED SYSFS
Path - Trigger: %s\n",sSYSFS_USRLED_Trigger); }

    if((iFILE_BBB_USRLED_Handle = fopen(sSYSFS_USRLED_Trigger, "r+")) !=
NULL)
    {
        // Re-Apply Trigger Corresponding to USR LED
        switch(prmUSRLED_No[0])
        {
            case BBB_USRLED0:
                iSYSFS_Write_Result=fwrite("heartbeat", sizeof("heartbeat"),
1, iFILE_BBB_USRLED_Handle);
                if (iDebug_Level >=Debug_Level_1)  { printf("USRLED Msg: Re-
Applying ""heartbeat"" Trigger \n"); }
                break;
            case BBB_USRLED1:
                iSYSFS_Write_Result=fwrite("mmc0", sizeof("mmc0"), 1,
iFILE_BBB_USRLED_Handle);
                if (iDebug_Level >=Debug_Level_1)  { printf("USRLED Msg: Re-
Applying ""mmc0"" Trigger \n"); }
                break;
            case BBB_USRLED2:
                iSYSFS_Write_Result=fwrite("cpu0", sizeof("cpu0"), 1,
iFILE_BBB_USRLED_Handle);
                if (iDebug_Level >=Debug_Level_1)  { printf("USRLED Msg: Re-
Applying ""cpu0"" Trigger \n"); }
                break;
            case BBB_USRLED3:
                iSYSFS_Write_Result=fwrite("mmc1", sizeof("mmc1"), 1,
iFILE_BBB_USRLED_Handle);
                if (iDebug_Level >=Debug_Level_1)  { printf("USRLED Msg: Re-
Applying ""mmc1"" Trigger \n"); }
                break;
            default:
                // Do Nothing
                break;
        }
        if (iSYSFS_Write_Result<=0)
        {
            // Error Handling
            if (iDebug_Level >=Debug_Level_1)  { printf("USRLED Error: Failed
to Write to USRLED SYSFS Trigger (On) File - Check Connection to BBB\n"); }
```

```
            // Set Error Number
            iError_No = Error_No_SYSFSWrite;
            // Set Error Level
            iError_Level = Error_Level_Critical;
        }
        // Close File
        fclose(iFILE_BBB_USRLED_Handle);

    }
    else
    {
        // Error Handling
        if (iDebug_Level >=Debug_Level_1)  { printf("USRLED Error: Failed
Find/Write to USRLED SYSFS Trigger(Reset) File - Check USRLED
Number/Connection to BBB\n"); }
        // Set Error Number
        iError_No = Error_No_SYSFSOpen;
        // Set Error Level
        iError_Level = Error_Level_Critical;
    }
    // Return Error Code
    if (iError_No!=Error_None)
    {   // Return Otuput Error Level
        // Error Handling
        if (iDebug_Level >=Debug_Level_1)  { printf("USRLED Error: - Error
Return From ""iBBB_USRLED_TriggerOn()"" Function\n");}
        // Return Error Code
        return -1;

    }
    else
    {
        return 0;
    }
}

/*****************************************/
/*****************************************/
// Main Program Start
/*****************************************/
/*****************************************/

if (inUSRLED_Enable[0]==true)
{
    if (xD[1]!=(double)inUSRLED_On_Trigger[0])
    {

        // Set Debug Level - Read Parameter Data -Debug Level
        iDebug_Level = iSet_Debug_Level(prmDebug_InfoLevel[0]);

        // Start Line
        if (iDebug_Level >=Debug_Level_0){printf("**** Start of USRLED
Control S-Function Block Execution **** \n");}
```

```
        // Create SYSFS Path for Specified USR LED
        sprintf(sSYSFS_USRLED_Path,"%s%d",sSYSFS_USRLED,prmUSRLED_No[0]-1);

        // Validate Path - Check USR LED Concatenation
        if (iDebug_Level >=Debug_Level_2)  { printf("USRLED Msg: USRLED SYSFS
Path: %s\n",sSYSFS_USRLED_Path);}

        /* Check Initialization */
        if (xD[0]==0)
        {

            // printf("iBBB_USRLED_TriggerOff: %i
%f\n",iBBB_USRLED_TriggerOff(),iBBB_USRLED_TriggerOff());
            //iError_Level = Error_Level_Critical;
            // Remove All Triggers
            iSYSFS_Trigger_Result=iBBB_USRLED_TriggerOff();
            printf("iSYSFS_Trigger_Result: %i
%f\n",iSYSFS_Trigger_Result,iSYSFS_Trigger_Result);
            if ((iSYSFS_Trigger_Result)<=-1)
            {
                // Error Handling
                if (iDebug_Level >=Debug_Level_1)  { printf("USRLED Error:
Failed to Remove Standard Trigger - Check Connection to BBB\n"); }
                // Set Error Number
                iError_No = Error_No_USRLED_TriggerOff;
                // Set Error Level
                iError_Level = Error_Level_Critical;
            }
        }

        // Confirm No Errors
        if (iError_Level==Error_None)
        {
            // Create SYSFS Path for Specified USR LED

sprintf(sSYSFS_USRLED_Brightness,"%s/brightness",sSYSFS_USRLED_Path);

            // Validate Path - Check USR LED Concatenation
            if (iDebug_Level >=Debug_Level_2)  { printf("USRLED Msg: USRLED
SYSFS Path - Brightness: %s\n",sSYSFS_USRLED_Brightness);}

            /*Turn Off/On USR LED */
            if (inUSRLED_On_Trigger[0]==true) /* && In_Leds_On[0]==0) */
            {
                /* Turn On USR LED */
                if((iFILE_BBB_USRLED_Handle = fopen(sSYSFS_USRLED_Brightness,
"r+")) != NULL)
                {
                    if (fwrite("1", sizeof(char), 1,
iFILE_BBB_USRLED_Handle)<=0)
                    {
                        // Error Handling
                        if (iDebug_Level >=Debug_Level_1)  { printf("USRLED
Error: Failed to Write to USRLED SYSFS Brightness (On) File - Check
Connection to BBB\n"); }
                        // Set Error Number
```

```
                        iError_No = Error_No_SYSFSWrite;
                        // Set Error Level
                        iError_Level = Error_Level_Critical;
                    }
                    fclose(iFILE_BBB_USRLED_Handle);
                    // Set Led State
                    iUSRLED_State=BBB_USRLED_ON;
                }
                else
                {
                    // Error Handling
                    if (iDebug_Level >=Debug_Level_1)  { printf("USRLED
Error: Failed Find/Open/Write to USRLED SYSFS Brightness File (On) - Check
Connection to BBB\n"); }
                    // Set Error Number
                    iError_No = Error_No_SYSFSOpen;
                    // Set Error Level
                    iError_Level = Error_Level_Critical;
                }
            }
            else
            {
                /* Turn Off USR LED */
                if((iFILE_BBB_USRLED_Handle = fopen(sSYSFS_USRLED_Brightness,
"r+")) != NULL)
                {
                    if (fwrite("0", sizeof(char), 1,
iFILE_BBB_USRLED_Handle)<=0)
                    {
                        // Error Handling
                        if (iDebug_Level >=Debug_Level_1)  { printf("USRLED
Error: Failed to Write to USRLED SYSFS Brightness (Off) File - Check
Connection to BBB\n"); }
                        // Set Error Number
                        iError_No = Error_No_SYSFSWrite;
                        // Set Error Level
                        iError_Level = Error_Level_Critical;
                    }
                    fclose(iFILE_BBB_USRLED_Handle);
                    // Set Led State
                    iUSRLED_State=BBB_USRLED_OFF;
                }
                else
                {
                    // Error Handling
                    if (iDebug_Level >=Debug_Level_1)  { printf("USRLED
Error: Failed Find/Open/Write to USRLED SYSFS Brightness (Off) File - Check
Connection to BBB\n"); }
                    // Set Error Number
                    iError_No = Error_No_SYSFSOpen;
                    // Set Error Level
                    iError_Level = Error_Level_Critical;
                }

            }
        }
        // Check For Critical Errors and Stop Simulation
```

```
        if ((iError_No!=0)&&(iError_Level==Error_Level_Critical))
        {

            // Stop Simulation If Critical Error
            if (iDebug_Level >=Debug_Level_0) {printf("USRLED Msg: Critical
Error Detected; Simulation Stopping! \nError No: %i Error Level:
%i\n",iError_No, iError_Level);}

            // Output Error Message:
            if (iDebug_Level >=Debug_Level_0) {printf("USRLED Error Details:
%s",sSYSFSReadWrite_ErrMessage(iError_No));}


            // Stop Simulation
            outSimStop[0]=true;
        }
        else
        {
            // Output USRLED State
            outUSRLED_State[0]=iUSRLED_State;


            // Carry on
            outSimStop[0]=false;
        }

    }
    // Print Gap
    if ((iDebug_Level >=Debug_Level_0)){printf("\n");}
}
else
{
    // Reset USR LED
    if  (xD[0]==1)
    {

        // Set Debug Level - Read Parameter Data -Debug Level
        iDebug_Level = iSet_Debug_Level(prmDebug_InfoLevel[0]);

        // Start Line
        if (iDebug_Level >=Debug_Level_0){printf("**** Start of USRLED
Control S-Function Block Execution **** \n");}

        // Set SYSFS Path for Specified USR LED
        sprintf(sSYSFS_USRLED_Path,"%s%d",sSYSFS_USRLED,prmUSRLED_No[0]-1);

        // Validate Path - Check USR LED Concatenation
        if (iDebug_Level >=Debug_Level_2)  { printf("USRLED Msg: USRLED SYSFS
Path: %s\n",sSYSFS_USRLED_Path);}

        // Create SYSFS Path for Specified USR LED
        sprintf(sSYSFS_USRLED_Brightness,"%s/brightness",sSYSFS_USRLED_Path);

        // Validate Path - Check USR LED Concatenation
```

```
        if (iDebug_Level >=Debug_Level_2)  { printf("USRLED Msg: USRLED SYSFS
Path - Brightness: %s\n",sSYSFS_USRLED_Brightness);}

        /* Turn Off USR LED */
        if((iFILE_BBB_USRLED_Handle = fopen(sSYSFS_USRLED_Brightness, "r+"))
!= NULL)
        {
            if (fwrite("0", sizeof(char), 1, iFILE_BBB_USRLED_Handle)<=0)
            {
                // Error Handling
                if (iDebug_Level >=Debug_Level_1)  { printf("USRLED Error:
Failed to Write to USRLED SYSFS Brightness (Off) File - Check Connection to
BBB\n"); }
                // Set Error Number
                iError_No = Error_No_SYSFSWrite;
                // Set Error Level
                iError_Level = Error_Level_Critical;
            }
            fclose(iFILE_BBB_USRLED_Handle);
            // Set Led State
            iUSRLED_State=BBB_USRLED_OFF;
        }
        else
        {
            // Error Handling
            if (iDebug_Level >=Debug_Level_1)  { printf("USRLED Error: Failed
Find/Open/Write to USRLED SYSFS Brightness (Off) File - Check Connection to
BBB\n"); }
            // Set Error Number
            iError_No = Error_No_SYSFSOpen;
            // Set Error Level
            iError_Level = Error_Level_Critical;
        }

        if (iError_Level != Error_Level_Critical)
        {
            // Reset USR LED Standard Triggers
            if (iBBB_USRLED_TriggerOn()<0)
            {
                // Error Handling
                if (iDebug_Level >=Debug_Level_1)  { printf("USRLED Error:
Failed to Re-Apply Standard Trigger - Check Connection to BBB\n"); }
                // Set Error Number
                iError_No = Error_No_USRLED_TriggerOn;
                // Set Error Level
                iError_Level = Error_Level_Critical;
            }
        }
    }
     // Check For Critical Errors and Stop Simulation
    if ((iError_No!=0)&&(iError_Level==Error_Level_Critical))
    {

        // Stop Simulation If Critical Error
```

```
        if (iDebug_Level >=Debug_Level_0) {printf("USRLED Msg: Critical Error
Detected; Simulation Stopping! \nError No: %i Error Level: %i\n",iError_No,
iError_Level);}

        // Output Error Message:
        if (iDebug_Level >=Debug_Level_0) {printf("USRLED Error Details:
%s",sSYSFSReadWrite_ErrMessage(iError_No));}


        // Stop Simulation
        outSimStop[0]=true;
    }
    else
    {
        // Output USRLED State
        outUSRLED_State[0]=iUSRLED_State;


        // Carry on
        outSimStop[0]=false;
    }
    // Print Gap
    if ((iDebug_Level >=Debug_Level_0)){printf("\n");}
}


#else
# endif
/* %%%-SFUNWIZ_wrapper_Outputs_Changes_END --- EDIT HERE TO _BEGIN */
}

/*
  * Updates function
  *
  */
void BBB_Driver_USRLED_Update_wrapper(const real_T *inUSRLED_Enable,
                          const boolean_T *inUSRLED_On_Trigger,
                          const boolean_T *outUSRLED_State,
                          const boolean_T *outSimStop ,
                          real_T *xD,
                          const uint8_T  *prmUSRLED_No,  const int_T
p_width0,
                           const uint8_T *prmDebug_InfoLevel, const int_T
p_width1)
{
  /* %%%-SFUNWIZ_wrapper_Update_Changes_BEGIN --- EDIT HERE TO _END */
# ifndef MATLAB_MEX_FILE

// Notes On Discrete States
// xD[0]: First Scan
// xD[1]: Output State Memorization
// Initialize Debug Output Level

eDebugLevel iDebug_Level = Debug_None;
```

```
// Check Debug Level - Read Parameter Data -Debug Level
iDebug_Level = iSet_Debug_Level(prmDebug_InfoLevel[0]);

// Record Output State
xD[1]=outUSRLED_State[0];

if ((xD[0]!=1) && (inUSRLED_Enable[0]==true))
{
    // Set First Scan Bit
    xD[0]=1;
    if ((iDebug_Level >=Debug_Level_0)){printf("USRLED Msg: First Scan Bit
Set\n");}
    // Print Gap
    if ((iDebug_Level >=Debug_Level_0)){printf("\n");}

}
else if ((xD[0]==1) && (inUSRLED_Enable[0]==false))
{
    // Reset First Scan Bit
    xD[0]=0;
    if ((iDebug_Level >=Debug_Level_0)){printf("USRLED Msg: First Scan Bit
ReSet\n");}
    // Print Gap
    if ((iDebug_Level >=Debug_Level_0)){printf("\n");}
}
// Print Standard Output
if ((iDebug_Level
>=Debug_Level_2)&&(inUSRLED_Enable[0]==true)){printf("USRLED Msg: Waiting For
Input Change\n");}

// Print Gap
// if ((iDebug_Level >=Debug_Level_0)&&(inUSRLED_Enable[0]==true) &&
(xD[1]!=(double)inUSRLED_On_Trigger[0])){printf("\n");}

#endif

//     /* Only Execute On Target*/
//     # ifndef MATLAB_MEX_FILE
//
//     static FILE *LEDHandle1 = NULL;
//     const char
*LED_00_Trigger="/sys/class/leds/beaglebone:green:usr3/trigger";
//
//
//     if((LEDHandle1 = fopen(LED_00_Trigger, "r+")) != NULL)
//         {
//          fwrite("none", sizeof(char), 4, LEDHandle1);
//          fclose(LEDHandle1);
//         }
//
//     # endif
//
//     /* Update Discrete Bit */
//     xD[0]=1;
/* %%%-SFUNWIZ_wrapper_Update_Changes_END --- EDIT HERE TO _BEGIN */
}
```

13.7 S-FUNCTION BLOCK WRAPPER CODE: ADXL330 ACCELEROMETER DRIVER BLOCK

```
/*
  *
  *   --- THIS FILE GENERATED BY S-FUNCTION BUILDER: 3.0 ---
  *
  *   This file is a wrapper S-function produced by the S-Function
  *   Builder which only recognizes certain fields.  Changes made
  *   outside these fields will be lost the next time the block is
  *   used to load, edit, and resave this file. This file will be overwritten
  *   by the S-function Builder block. If you want to edit this file by hand,
  *   you must change it only in the area defined as:
  *
  *        %%%-SFUNWIZ_wrapper_XXXXX_Changes_BEGIN
  *            Your Changes go here
  *        %%%-SFUNWIZ_wrapper_XXXXXX_Changes_END
  *
  *   For better compatibility with the Simulink Coder, the
  *   "wrapper" S-function technique is used.  This is discussed
  *   in the Simulink Coder User's Manual in the Chapter titled,
  *   "Wrapper S-functions".
  *
  *   Created: Sun Dec 14 05:04:47 2014
  */


/*
 * Include Files
 *
 */
#if defined(MATLAB_MEX_FILE)
#include "tmwtypes.h"
#include "simstruc_types.h"
#else
#include "rtwtypes.h"
#endif

/* %%%-SFUNWIZ_wrapper_includes_Changes_BEGIN --- EDIT HERE TO _END */
#include <math.h>
#include <stdio.h>
#include <stddef.h>
#include <time.h>

# ifdef MATLAB_MEX_FILE
    #include <mex.h>
    #include <simstruc.h>
# endif


#define dBBB_ADC_Op_Voltage_3V6 3.6
#define dBBB_ADC_Op_Voltage_3V33 3.33
#define dBBB_ADC_Op_Voltage_3V0 3.0
```

```
#define dBBB_ADC_Op_Voltage_2V0 2.0
#define iBBB_ADC_Sensitivity_3V6 360
#define iBBB_ADC_Sensitivity_3V33 333
#define iBBB_ADC_Sensitivity_3V0 300
#define iBBB_ADC_Sensitivity_2V0 195

float dBBB_ADC_Volt_Val_X_Global=0;
float dBBB_ADC_Volt_Val_Y_Global=0;
float dBBB_ADC_Volt_Val_Z_Global=0;

// Define Debug Levels
typedef enum
{
        Debug_None,
    Debug_Level_0, // Basic Debug Info Output
        Debug_Level_1, // + Critical Info Only
    Debug_Level_2, // + Diagnostics Info
    Debug_Level_3, // + Program Flow Info
    Debug_Level_4, // + Registry Data
    Debug_Level_5  // + TBD
}eDebugLevel;


// Define Error Number
typedef enum
{
    Error_None=0,
    Error_No_SYSFSOpen,
    Error_No_SYSFSSeek,
    Error_No_SYSFSScan,
    Error_No_SYSFSClose,
    Error_No_ParamOutofRange
}eError_No;

// Initialize Error
// eError_No iError_No = Error_None;

// Define Error Severity Levels
typedef enum
{
    Error_Level_OK=0,
    Error_Level_Critical, // (Stop Simulation)
    Error_Level_Warning
}eError_Level;
/* %%%-SFUNWIZ_wrapper_includes_Changes_END --- EDIT HERE TO _BEGIN */
#define u_width
#define y_width 1
/*
 * Create external references here.
 *
 */
/* %%%-SFUNWIZ_wrapper_externs_Changes_BEGIN --- EDIT HERE TO _END */
/* extern double func(double a); */
/* %%%-SFUNWIZ_wrapper_externs_Changes_END --- EDIT HERE TO _BEGIN */
```

```
/*
 * Output functions
 *
 */
void BBB_Driver_Accelerometer_ADXL330_Outputs_wrapper(real_T
*outX_Axis_Accel,
                          real_T *outY_Axis_Accel,
                          real_T *outZ_Axis_Accel,
                          boolean_T *outSimStop ,
                  const real_T  *xD,
                          const uint16_T  *prmAccel_Sensitivity, const int_T
p_width0,
                          const boolean_T  *prmRunCalibration, const int_T
p_width1,
                          const uint16_T  *prmDebug_InfoLevel, const int_T
p_width2)
{
/* %%%-SFUNWIZ_wrapper_Outputs_Changes_BEGIN --- EDIT HERE TO _END */
// Define Output Data - Simulink

// Run Only on Target - BeagleBoneBlack */
# ifndef MATLAB_MEX_FILE

// Declare and Initialize Error
eError_No iError_No = Error_None;

// Declare and Initialize Error Level
eError_Level iError_Level = Error_Level_OK;

// Define File Sys Pointer - Note Code Will Use fopen vs. open
static FILE *fBBB_Handle_X = NULL;
static FILE *fBBB_Handle_Y = NULL;
static FILE *fBBB_Handle_Z = NULL;
const char *sBBB_ACC_AIN_X ="/sys/bus/platform/drivers/bone-iio-
helper/helper.15/AIN0";
const char *sBBB_ACC_AIN_Y ="/sys/bus/platform/drivers/bone-iio-
helper/helper.15/AIN1";
const char *sBBB_ACC_AIN_Z ="/sys/bus/platform/drivers/bone-iio-
helper/helper.15/AIN2";

// Define Internal Variable Data
float dBBB_ADC_Volt_Val_X=0;
float dBBB_ADC_Volt_Val_Y=0;
float dBBB_ADC_Volt_Val_Z=0;
float dBBB_ACC_Val_X=0;
float dBBB_ACC_Val_Y=0;
float dBBB_ACC_Val_Z=0;
float const dBBB_ACC_Val_X_Nominal=1666+(-0.06006*333);
float const dBBB_ACC_Val_Y_Nominal=1666+(-1.234234*333);//+(-0.234234*333);
float const dBBB_ACC_Val_Z_Nominal=1666+((0.912013-1)*333);
float dBBB_ADC_Volt_CalibOffset_X=0;
float dBBB_ADC_Volt_CalibOffset_Y=0;
float dBBB_ADC_Volt_CalibOffset_Z=0;
float dBBB_ADC_Volt_Nominal_X=0;
float dBBB_ADC_Volt_Nominal_Y=0;
float dBBB_ADC_Volt_Nominal_Z=0;
```

```
float dBBB_ADC_Volt_Correction_X=0;
float dBBB_ADC_Volt_Correction_Y=0;
float dBBB_ADC_Volt_Correction_Z=0;
int iBBB_ADC_Sensitivity[5]={
    0,
    iBBB_ADC_Sensitivity_3V6,
    iBBB_ADC_Sensitivity_3V33,
    iBBB_ADC_Sensitivity_3V0,
    iBBB_ADC_Sensitivity_2V0};


float dBBB_ADC_OpVoltage[5]={
    0,
    dBBB_ADC_Op_Voltage_3V6,
    dBBB_ADC_Op_Voltage_3V33,
    dBBB_ADC_Op_Voltage_3V0,
    dBBB_ADC_Op_Voltage_2V0};

// Initialize Debug Output Level
eDebugLevel iDebug_Level = Debug_None;

/*************************************/
// Accelerometer Error Message Output
/*************************************/
const char *sAccel_ErrMessage(eError_No iAccel_ErrNo)
{
    //Set Error Message Based on Error Number
    switch(iAccel_ErrNo)
    {
       case  Error_None:
            return("No Error Active. System A-OK!\n");
            break;
        case  Error_No_SYSFSOpen:
            return("Failed to Open I2C Bus\n - Check Bus Number\n - Check
SCL/SCA Lines\n"
                   " - Check Device\n - Check Device Tree on BBB\n");
            break;
        case  Error_No_SYSFSSeek:
            return("Failed to Open I2C Bus\n - Check Bus Number\n - Check
SCL/SCA Lines\n"
                   " - Check Device\n - Check Device Tree on BBB\n");
            break;
        case Error_No_SYSFSScan:
            return("Failed to Communicate With Slave Device\n - Check Bus
Number\n - Check SCL/SCA Lines\n"
                   " - Check Device Address\n - Check Device Tree on BBB\n");
            break;
        case Error_No_SYSFSClose:
            return("Failed to Set Buffer Read Range\n - Check Bus Number\n -
Check SCL/SCA Lines\n"
                   " - Check Device Address\n - Check Device Tree on BBB\n");
            break;
        case Error_No_ParamOutofRange:
            return("Parameter Out Of Range\n - Check Block Parameter Input
Options\n");
            break;
```

```
        default:
            // Error -Return Default
            return("Error Code Not Recognized\n");
            break;
    }
}

/*****************************************/
/*****************************************/
// Main Program Start
/*****************************************/
/*****************************************/

// Read Parameter Data -Debug Level
switch(prmDebug_InfoLevel[0])
{
    case  1:
        iDebug_Level=Debug_None;
        break;
    case  2:
        iDebug_Level=Debug_Level_0;
        break;
    case  3:
        iDebug_Level=Debug_Level_1;
        break;
    case  4:
        iDebug_Level=Debug_Level_2;
        break;
    case  5:
        iDebug_Level=Debug_Level_3;
        break;
    case  6:
        iDebug_Level=Debug_Level_4;
        break;
    case  7:
        iDebug_Level=Debug_Level_5;
        break;
    default:
        iDebug_Level=Debug_None;
        break;
}

// Start Line
if (iDebug_Level >=Debug_Level_0){printf("**** Start of GPIO Write S-Function
Block Execution **** \n");}

// Open Streams
fBBB_Handle_X = fopen(sBBB_ACC_AIN_X, "r");
fBBB_Handle_Y = fopen(sBBB_ACC_AIN_Y, "r");
fBBB_Handle_Z = fopen(sBBB_ACC_AIN_Z, "r");

// Confirm SYSFS IOStream Open
if ((fBBB_Handle_X==NULL) | (fBBB_Handle_Y==NULL) | (fBBB_Handle_Z==NULL))
{
    // Error Handling
```

```
    if (iDebug_Level >=Debug_Level_1)  { printf("Accelerometer Error: Failed
To Open I/O File Stream - Check ""cape-bone-iio"" Device Tree Active\n"); }
    // Set Error Number
    iError_No = Error_No_SYSFSOpen;
    // Set Error Level
    iError_Level = Error_Level_Critical;
    // Return Otuput Error Level
    // return -1;
}

// Ouput Program Flow
if (iDebug_Level >=Debug_Level_3) {printf("Accelerometer Msg: Program Flow
State 1: Open SYSFS File I/O Stream\nError No: %i Error Level:
%i\n",iError_No, iError_Level);}


// Write Read Raw Voltage Data
if (iError_No == Error_None)
{
    // Read X Accel
    if (fseek(fBBB_Handle_X, 0, SEEK_SET)!=0)
    {
        // Error Handling
        if (iDebug_Level >=Debug_Level_1)  { printf("Accelerometer Error:
""fseek()"" Failed To Set Position Indicator in X-Axis I/O File Stream\n"); }
        // Set Error Number
        iError_No = Error_No_SYSFSSeek;
        // Set Error Level
        iError_Level = Error_Level_Critical;
    }
    if (fscanf(fBBB_Handle_X, "%f", &dBBB_ADC_Volt_Val_X)!=1)
    {
        // Error Handling
        if (iDebug_Level >=Debug_Level_1)  { printf("Accelerometer Error:
""fscan()"" Failed To Read Data From X-Axis I/O File Stream\n"); }
        // Set Error Number
        iError_No = Error_No_SYSFSScan;
        // Set Error Level
        iError_Level = Error_Level_Critical;
    }
    else
    {
        // Record Voltage Data in Global Variable
        dBBB_ADC_Volt_Val_X_Global=dBBB_ADC_Volt_Val_X;
    }

    // Read Y Accel
    if (fseek(fBBB_Handle_Y, 0, SEEK_SET)!=0)
    {
        // Error Handling
        if (iDebug_Level >=Debug_Level_1)  { printf("Accelerometer Error:
""fseek()"" Failed To Set Position Indicator in Y-Axis I/O File Stream\n"); }
        // Set Error Number
        iError_No = Error_No_SYSFSSeek;
        // Set Error Level
        iError_Level = Error_Level_Critical;
```

```
    }
    if (fscanf(fBBB_Handle_Y, "%f", &dBBB_ADC_Volt_Val_Y)!=1)
    {
        // Error Handling
        if (iDebug_Level >=Debug_Level_1)  { printf("Accelerometer Error:
""fscan()"" Failed To Read Data From Y-Axis I/O File Stream\n"); }
        // Set Error Number
        iError_No = Error_No_SYSFSScan;
        // Set Error Level
        iError_Level = Error_Level_Critical;
    }
    else
    {
        // Record Voltage Data in Global Variable
        dBBB_ADC_Volt_Val_Y_Global=dBBB_ADC_Volt_Val_Y;
    }

    // Read Z Accel
    if (fseek(fBBB_Handle_Z, 0, SEEK_SET)!=0)
    {
        // Error Handling
        if (iDebug_Level >=Debug_Level_1)  { printf("Accelerometer Error:
""fseek()"" Failed To Set Position Indicator in Z-Axis I/O File Stream\n"); }
        // Set Error Number
        iError_No = Error_No_SYSFSSeek;
        // Set Error Level
        iError_Level = Error_Level_Critical;
    }
    if (fscanf(fBBB_Handle_Z, "%f", &dBBB_ADC_Volt_Val_Z)!=1)
    {
        // Error Handling
        if (iDebug_Level >=Debug_Level_1)  { printf("Accelerometer Error:
""fscan()"" Failed To Read Data From Z-Axis I/O File Stream\n"); }
        // Set Error Number
        iError_No = Error_No_SYSFSScan;
        // Set Error Level
        iError_Level = Error_Level_Critical;
    }
    else
    {
        // Record Voltage Data in Global Variable
        dBBB_ADC_Volt_Val_Z_Global=dBBB_ADC_Volt_Val_Z;
    }
}

// Ouput Program Flow
if (iDebug_Level >=Debug_Level_3) {printf("Accelerometer Msg: Program Flow
State 2: Seek & Scan Data From File I/O Stream\nError No: %i Error Level:
%i\n",iError_No, iError_Level);}

// Voltage Calibration
if ((prmRunCalibration[0]==true) && (xD[0]==1))
{


    // Check Sensitivity setting
```

```
    switch(prmAccel_Sensitivity[0])
    {

        case  1: // 3.6V
            dBBB_ADC_Volt_Correction_X=(1000*dBBB_ADC_Op_Voltage_3V6/2)-
2*xD[3];
            dBBB_ADC_Volt_Correction_Y=(1000*dBBB_ADC_Op_Voltage_3V6/2)-
2*xD[4];

dBBB_ADC_Volt_Correction_Z=(1000*(dBBB_ADC_Op_Voltage_3V6/2)+iBBB_ADC_Sensiti
vity_3V6)-2*xD[5];
            break;
        case  2: // 3.3V
            dBBB_ADC_Volt_Correction_X=(1000*dBBB_ADC_Op_Voltage_3V33/2)-
2*xD[3];
            dBBB_ADC_Volt_Correction_Y=(1000*dBBB_ADC_Op_Voltage_3V33/2)-
2*xD[4];

dBBB_ADC_Volt_Correction_Z=(1000*(dBBB_ADC_Op_Voltage_3V33/2)+iBBB_ADC_Sensit
ivity_3V33)-2*xD[5];
            break;
        case  3: // 3.0V
            dBBB_ADC_Volt_Correction_X=(1000*dBBB_ADC_Op_Voltage_3V0/2)-
2*xD[3];
            dBBB_ADC_Volt_Correction_Y=(1000*dBBB_ADC_Op_Voltage_3V0/2)-
2*xD[4];

dBBB_ADC_Volt_Correction_Z=(1000*(dBBB_ADC_Op_Voltage_3V0/2)+iBBB_ADC_Sensiti
vity_3V0)-2*xD[5];
            break;
        case  4: // 2.0V
            dBBB_ADC_Volt_Correction_X=(1000*dBBB_ADC_Op_Voltage_2V0/2)-
2*xD[3];
            dBBB_ADC_Volt_Correction_Y=(1000*dBBB_ADC_Op_Voltage_2V0/2)-
2*xD[4];

dBBB_ADC_Volt_Correction_Z=(1000*(dBBB_ADC_Op_Voltage_2V0/2)+iBBB_ADC_Sensiti
vity_2V0)-2*xD[5];
            break;
        default:
            break;

    }
}



// Convert Voltage To Acceleration (in g) Depending on Sensitivity Setting
if (iError_No == Error_None)
{
    // Convert Voltage To Acceleration (in g)
    dBBB_ACC_Val_X=(((2*dBBB_ADC_Volt_Val_X)+dBBB_ADC_Volt_Correction_X)-
(1000*dBBB_ADC_OpVoltage[prmAccel_Sensitivity[0]]/2))/iBBB_ADC_Sensitivity[pr
mAccel_Sensitivity[0]];
```

```
    dBBB_ACC_Val_Y=(((2*dBBB_ADC_Volt_Val_Y)+dBBB_ADC_Volt_Correction_Y)-
(1000*dBBB_ADC_OpVoltage[prmAccel_Sensitivity[0]]/2))/iBBB_ADC_Sensitivity[pr
mAccel_Sensitivity[0]];
    dBBB_ACC_Val_Z=(((2*dBBB_ADC_Volt_Val_Z)+dBBB_ADC_Volt_Correction_Z)-
(1000*dBBB_ADC_OpVoltage[prmAccel_Sensitivity[0]]/2))/iBBB_ADC_Sensitivity[pr
mAccel_Sensitivity[0]];

}

// Ouput Program Calibration Data
if (iDebug_Level >=Debug_Level_3) {printf("Accelerometer Msg: Avrg. Recorded
Voltage Reading X: %.2f, Y: %.2f Z: %.2f \n",xD[3],xD[4],xD[5]);}

// Ouput Program Calibration Data
if (iDebug_Level >=Debug_Level_3) {printf("Accelerometer Msg: Voltage
Correcton Data: X: %.2f, Y: %.2f Z: %.2f
\n",dBBB_ADC_Volt_Correction_X,dBBB_ADC_Volt_Correction_Y,dBBB_ADC_Volt_Corre
ction_Z);}

// Ouput Program Calibration Data
if (iDebug_Level >=Debug_Level_3) {printf("Accelerometer Msg: Voltage
Sensitivity Level (mV/g): %i
\n",iBBB_ADC_Sensitivity[prmAccel_Sensitivity[0]]);}


// Ouput Program Flow
if (iDebug_Level >=Debug_Level_3) {printf("Accelerometer Msg: Program Flow
State 3: Convert Voltage Data to Acceleration \nError No: %i Error Level:
%i\n",iError_No, iError_Level);}

// Close File on Exit
if (fclose(fBBB_Handle_X)!=0)
{
    // Error Handling
    if (iDebug_Level >=Debug_Level_1)  { printf("Accelerometer Error:
""fclose()"" Failed To Properly Close Data From X-Axis I/O File Stream\n"); }
    // Set Error Number
    iError_No = Error_No_SYSFSClose;
    // Set Error Level
    iError_Level = Error_Level_Critical;
}

// Close File on Exit
if (fclose(fBBB_Handle_Y)!=0)
{
    // Error Handling
    if (iDebug_Level >=Debug_Level_1)  { printf("Accelerometer Error:
""fclose()"" Failed To Properly Close Data From Y-Axis I/O File Stream\n"); }
    // Set Error Number
    iError_No = Error_No_SYSFSClose;
    // Set Error Level
    iError_Level = Error_Level_Critical;
}

// Close File on Exit
```

```
if (fclose(fBBB_Handle_Z)!=0)
{
    // Error Handling
    if (iDebug_Level >=Debug_Level_1)  { printf("Accelerometer Error:
""fclose()"" Failed To Properly Close Data From Z-Axis I/O File Stream\n"); }
    // Set Error Number
    iError_No = Error_No_SYSFSClose;
    // Set Error Level
    iError_Level = Error_Level_Critical;
}

// Ouput Program Flow
if (iDebug_Level >=Debug_Level_3) {printf("Accelerometer Msg: Program Flow
State 4: Close SYSFS File I/O Stream\nError No: %i Error Level:
%i\n",iError_No, iError_Level);}


// Print Acceleration Data
if (iDebug_Level >=Debug_Level_0)  { printf("Accel: X: %f  Y: %f  Z:
%f\n",dBBB_ACC_Val_X,dBBB_ACC_Val_Y,dBBB_ACC_Val_Z);}
// Print Voltage Data
if (iDebug_Level >=Debug_Level_1)  { printf("Voltage: X: %f  Y: %f  Z:
%f\n",dBBB_ADC_Volt_Val_X,dBBB_ADC_Volt_Val_Y,dBBB_ADC_Volt_Val_Z);}

// Check For Critical Errors and Stop Simulation
if (((iError_No!=0)&&(iError_Level==Error_Level_Critical))) //||
(xD[0]==true))
{

    // Stop Simulation If Critical Error
    if (iDebug_Level >=Debug_Level_0) {printf("Accelerometer Msg: Critical
Error Detected; Simulation Stopping! \nError No: %i Error Level:
%i\n",iError_No, iError_Level);}

    // Output Error Message:
    if (iDebug_Level >=Debug_Level_0) {printf("Accelerometer Error Details:
%s",sAccel_ErrMessage(iError_No));}


    // Stop Simulation
    outSimStop[0]=true;
}
else
{
    // Carry on
    outSimStop[0]=false;

    // Transfer Data to Outputs!
    outX_Axis_Accel[0]=dBBB_ACC_Val_X;
    outY_Axis_Accel[0]=dBBB_ACC_Val_Y;
    outZ_Axis_Accel[0]=dBBB_ACC_Val_Z;

    // Print Lint Space:
    if (iDebug_Level >=Debug_Level_0) {printf("\n");}
```

```
}

# else


# endif
/* %%%-SFUNWIZ_wrapper_Outputs_Changes_END --- EDIT HERE TO _BEGIN */
}

/*
  * Updates function
  *
  */
void BBB_Driver_Accelerometer_ADXL330_Update_wrapper(const real_T
*outX_Axis_Accel,
                          const real_T *outY_Axis_Accel,
                          const real_T *outZ_Axis_Accel,
                          const boolean_T *outSimStop ,
                          real_T *xD,
                          const uint16_T  *prmAccel_Sensitivity,  const int_T
p_width0,
                          const boolean_T  *prmRunCalibration,  const int_T
p_width1,
                           const uint16_T *prmDebug_InfoLevel, const int_T
p_width2)
{
  /* %%%-SFUNWIZ_wrapper_Update_Changes_BEGIN --- EDIT HERE TO _END */
// Run Only on Target - BeagleBoneBlack */
# ifndef MATLAB_MEX_FILE


// Parameter Definition
// xD[0]= First Scan / Calibration Complete / Init: 0
// xD[1]= Calibration Sample Counter / Init: 0
// xD[2]= Calibration Sample Quantity / Init: 10
// xD[3]= X Axis Voltage Total Sum / Init: 0
// xD[4]= Y Axis Voltage Total Sum / Init: 0
// xD[5]= Z Axis Voltage Total Sum / Init: 0


// Check & Execute Calibraiton Routine
if ((prmRunCalibration[0]==true) && (xD[0]==0))
{
   if (xD[1]<xD[2])
   {
      // Update X Voltage Sum
      xD[3]=xD[3]+dBBB_ADC_Volt_Val_X_Global;
      // Update Y Voltage Sum
      xD[4]=xD[4]+dBBB_ADC_Volt_Val_Y_Global;
      // Update Z Voltage Sum
      xD[5]=xD[5]+dBBB_ADC_Volt_Val_Z_Global;

      // Increment Sample Counter
      xD[1]=xD[1]+1;
   }
```

```
   else
   {
     // Voltage Sample Sum Completed Calculate Average
     xD[3]=xD[3]/xD[2]; // X Voltage
     xD[4]=xD[4]/xD[2]; // Y Voltage
     xD[5]=xD[5]/xD[2]; // Z Voltage

     // Set Calibration Flag Complete/ First Scan Bit
      xD[0]=1;
   }
}
else
{
    if (xD[0]==0)
    {
    // Set First Scan Bit
    xD[0]=1;
    }
}

// Print Space Line
//printf("/n");

# endif
/* %%%-SFUNWIZ_wrapper_Update_Changes_END --- EDIT HERE TO _BEGIN */
}
```

13.8 S-FUNCTION BLOCK WRAPPER CODE: L3G4200D GYROSCOPE DRIVER BLOCK

```
/*
  *
  *   --- THIS FILE GENERATED BY S-FUNCTION BUILDER: 3.0 ---
  *
  *   This file is a wrapper S-function produced by the S-Function
  *   Builder which only recognizes certain fields.  Changes made
  *   outside these fields will be lost the next time the block is
  *   used to load, edit, and resave this file. This file will be overwritten
  *   by the S-function Builder block. If you want to edit this file by hand,
  *   you must change it only in the area defined as:
  *
  *        %%%-SFUNWIZ_wrapper_XXXXX_Changes_BEGIN
  *            Your Changes go here
  *        %%%-SFUNWIZ_wrapper_XXXXXX_Changes_END
  *
  *   For better compatibility with the Simulink Coder, the
  *   "wrapper" S-function technique is used.  This is discussed
  *   in the Simulink Coder User's Manual in the Chapter titled,
  *   "Wrapper S-functions".
  *
  *   Created: Sun Dec 14 05:06:45 2014
  */


/*
 * Include Files
 *
 */
#if defined(MATLAB_MEX_FILE)
#include "tmwtypes.h"
#include "simstruc_types.h"
#else
#include "rtwtypes.h"
#endif

/* %%%-SFUNWIZ_wrapper_includes_Changes_BEGIN --- EDIT HERE TO _END */
#include <math.h>
#include <stdio.h>
#include <stddef.h>
#include <time.h>
#include <stdlib.h>
#include <string.h>
//#include <inttypes.h>
//#include <errno.h>


# ifndef MATLAB_MEX_FILE

    #include <linux/i2c.h>
    #include <linux/i2c-dev.h>
    #include <sys/ioctl.h>
```

```
    #include <unistd.h>
    #include <sys/types.h>
    #include <sys/stat.h>
    #include <fcntl.h>
    //#include <simstruc.h>
# else

    #include <mex.h>
    #include <simstruc.h>

# endif


//#define S_FUNCTION_NAME BBB_Driver_Gyroscope_L3G4200D
//#define S_FUNCTION_LEVEL 2

// Define Gyproscope Registry Locations
# define cAngVelReg_X_LSB 0x28
# define cAngVelReg_X_MSB 0x29
# define cAngVelReg_Y_LSB 0x2A
# define cAngVelReg_Y_MSB 0x2B
# define cAngVelReg_Z_LSB 0x2C
# define cAngVelReg_Z_MSB 0x2D
# define cAmbTempReg      0x26
# define cGyro_I2C_Addr1  0x68
# define cGyro_I2C_Addr2  0x69
# define cGyro_I2C_BufferSize 0x30 // Read Only Interested Part of Register
# define cGyro_WhoAmIReg  0x0F
# define cGyro_WhoAmI     0xD3
# define cGyro_CtrlReg1   0x20
# define cGyro_CtrlReg4   0x23
# define cGyro_NormalMode 0x0F
# define cGyrp_StatusReg  0x27


// Set Gyroscope Sensitivity Ratings
#define dGyro_Sens_250dps  0.00875
#define dGyro_Sens_500dps  0.0175
#define dGyro_Sens_2000dps 0.075

// Set Gyroscope Measurement Range Register Values
#define cGyro_Sens_250dps  0x00
#define cGyro_Sens_500dps  0x10
#define cGyro_Sens_2000dps 0x30

/*
// Declare Sensitivity Ratings
double dGyro_Sens_250dps = 0.00875;
double dGyro_Sens_500dps = 0.0175;
double dGyro_Sens_2000dps = 0.075;
*/

//SimStruct *S;

// Declare Error Handling
```

```
// extern int errno;

// Function Options (1=On/0=Off)
int iOpt_TwoComplement = 0; // Data To Be Treated as Two's Complement

// Delcare Global Variables
int iLoop_Counter_DS=0;

// Define Debug Levels
typedef enum
{
        Debug_None,
    Debug_Level_0, // Basic Debug Info Output
        Debug_Level_1, // + Critical Info Only
    Debug_Level_2, // + Diagnostics Info
    Debug_Level_3, // + Program Flow Info
    Debug_Level_4, // + Registry Data
    Debug_Level_5  // + TBD
}eDebugLevel;

// Initialize Debug Output Level
eDebugLevel iDebug_Level = Debug_None;

// Define Error Number
typedef enum
{
    Error_None=0,
    Error_No_I2CComm,
    Error_No_I2CInit,
    Error_No_I2CBuffSet,
    Error_No_I2CBuffRead,
    Error_No_I2CBuffSync,
    Error_No_I2CBuffWrite,
    Error_No_ParamOutofRange
}eError_No;

// Initialize Error
// eError_No iError_No = Error_None;

// Define Error Severity Levels
typedef enum
{
    Error_Level_OK=0,
    Error_Level_Critical, // (Stop Simulation)
    Error_Level_Warning
}eError_Level;

// Initialize Error Level
// eError_Level iError_Level = Error_Level_OK;

// Define Parameter Selection: Gyro Sensitivity
typedef enum
{
    iAngVel_250dps=1,
    iAngVel_500dps=2,
```

```
    iAngVel_2000dps=3
}eGyro_Range;



/*
double dGyro_SetSens(eGyro_Range iGyro_Range)
{
    //Set Sensitivity Value Based on Gyro Range Setting
    switch(iGyro_Range)
    {
        case  iAngVel_250dps: //iAngVel_250dps:
            return((double)dGyro_Sens_250dps);
            break;
        case iAngVel_500dps:
            return((double)dGyro_Sens_500dps);
            break;
        case iAngVel_2000dps:
            return((double)dGyro_Sens_2000dps);
            break;
        default:
            // Error
            return(-1);
    }
}
*/
/* %%%-SFUNWIZ_wrapper_includes_Changes_END --- EDIT HERE TO _BEGIN */
#define u_width
#define y_width 1
/*
 * Create external references here.
 *
 */
/* %%%-SFUNWIZ_wrapper_externs_Changes_BEGIN --- EDIT HERE TO _END */
/* extern double func(double a); */

double dGyro_SetSens(eGyro_Range iGyro_Range)
{
    //Set Sensitivity Value Based on Gyro Range Setting
    switch(iGyro_Range)
    {
        case  iAngVel_250dps: //iAngVel_250dps:
            return((double)dGyro_Sens_250dps);
            break;
        case iAngVel_500dps:
            return((double)dGyro_Sens_500dps);
            break;
        case iAngVel_2000dps:
            return((double)dGyro_Sens_2000dps);
            break;
        default:
            // Error
            return(-1);
    }
}
```

```
char cGyro_SetRange(eGyro_Range iGyro_Range)
{
    //Set Sensitivity Value Based on Gyro Range Setting
    switch(iGyro_Range)
    {
        case  iAngVel_250dps: //iAngVel_250dps:
            return((char)cGyro_Sens_250dps);
            break;
        case iAngVel_500dps:
            return((char)cGyro_Sens_500dps);
            break;
        case iAngVel_2000dps:
            return((char)cGyro_Sens_2000dps);
            break;
        default:
            // Error -Return Default
            return((char)cGyro_Sens_250dps);
            break;
    }
}

const char *sGyro_SetRange(eGyro_Range iGyro_Range)
{
    //Set Sensitivity Value Based on Gyro Range Setting
    switch(iGyro_Range)
    {
        case  iAngVel_250dps: //iAngVel_250dps:
            return("250 dps\n");
            break;
        case iAngVel_500dps:
            return("500 dps\n");
            break;
        case iAngVel_2000dps:
            return("2000 dps\n");
            break;
        default:
            // Error -Return Default
            return("Range Request Error\n");
            break;
    }
}


const char *sGyro_ErrMessage(eError_No iGyro_ErrNo)
{
    //Set Error Message Based on Error Number
    switch(iGyro_ErrNo)
    {
       case  Error_None:
            return("No Error Active. System A-OK!\n");
            break;
        case  Error_No_I2CComm:
            return("Failed to Open I2C Bus\n - Check Bus Number\n - Check
SCL/SCA Lines\n"
                   " - Check Device\n - Check Device Tree on BBB\n");
            break;
```

```
        case Error_No_I2CInit:
            return("Failed to Communicate With Slave Device\n - Check Bus
Number\n - Check SCL/SCA Lines\n"
                   " - Check Device Address\n - Check Device Tree on BBB\n");
            break;
        case Error_No_I2CBuffSet:
            return("Failed to Set Buffer Read Range\n - Check Bus Number\n -
Check SCL/SCA Lines\n"
                   " - Check Device Address\n - Check Device Tree on BBB\n");
            break;
        case Error_No_I2CBuffRead:
            return("Failed to Read Data in Registry Buffer\n - Check Bus
Number\n - Check SCL/SCA Lines\n"
                   " - Check Device Address\n - Check Device Tree on BBB\n");
            break;
        case Error_No_I2CBuffSync:
            return("Failed to Validate Registry Data - Inconsistent Data
Stream\n"
                    " - Check For Multiple Master Access to Slave Device on
I2C Bus Number\n"
                    " - Check SCL/SCA Lines\n - Check Device Address\n -
Check Device Tree on BBB\n");
            break;
        case Error_No_I2CBuffWrite:
            return("Failed to Write Data to Device Registry\n - Check Bus
Traffic/Interference\n"
                   " - Check Bus Number\n - Check SCL/SCA Lines\n - Check
Device Address\n - Check Device Tree on BBB\n");
            break;
        case Error_No_ParamOutofRange:
            return("Parameter Out Of Range\n - Check Block Parameter Input
Options\n");
            break;
        default:
            // Error -Return Default
            return("Error Code Not Recognized\n");
            break;


    }
}
/* %%%-SFUNWIZ_wrapper_externs_Changes_END --- EDIT HERE TO _BEGIN */

/*
 * Output functions
 *
 */
void BBB_Driver_Gyroscope_L3G4200D_Outputs_wrapper(real_T *outAngVel_X,
                          real_T *outAngVel_Y,
                          real_T *outAngVel_Z,
                          int8_T *outAmbTemp,
                          boolean_T *outSimStop ,
                  const real_T  *xD,
                          const uint8_T  *prmI2C_BusNo, const int_T
p_width0,
```

```
                        const uint8_T  *prmI2C_DeviceNo, const int_T
p_width1,
                        const uint8_T  *prmGyro_AngVelRange, const int_T
p_width2,
                        const uint8_T  *prmDebug_InfoLevel, const int_T
p_width3)
{
/* %%%-SFUNWIZ_wrapper_Outputs_Changes_BEGIN --- EDIT HERE TO _END */
// Define Output Data - Simulink
int iTst=0;
// Run Only on Target - BeagleBoneBlack */
# ifndef MATLAB_MEX_FILE

// Define File Sys Pointer - For I2C Bus#1 -> Make Parameter
int iFILE_BBB_I2C_Handle=0;
int iIOCTL_BBB_I2C_Handle=0;
int iSYNC_BBB_I2C_Handle=0;
int iWrite_BBB_I2C_Handle=0;
// Function Outputs
int iI2C_Read=0;
int iI2C_Write=0;

//Define Device Driver sysfs Location
// const char *sBBB_I2C_BUS1_FILE ="/dev/i2c-1";
// char *sBBB_I2C_BUS1_FILE ="/dev/i2c-";
char sBBB_I2C_BUS1_FILE[100]="";

// Define Internal Variables Data
int iGyro_AngVel_X=0;
int iGyro_AngVel_Y=0;
int iGyro_AngVel_Z=0;
int iGyro_AngVel_Temp=0;

// Declare Temporary Variables For Bit Manipulation
double dGyro_AngVel_Temp=0;

// Declare Variables For Device Registry Read/Write
unsigned char cI2C_StatReg_Temp=0;

// Define I2C Parameters
int iI2C_BytesRead=0;
// Define I2C Buffer Size
const int iI2C_BufferSize=48;
// Define I2C Buffer Dump Variable
char cI2C_Buffer[cGyro_I2C_BufferSize]={0x00}; //unsigned
// Define I2C Buffer Dump Start Position. Note: Value = Start Address w/ MSB
Set To "1"
char cI2C_Buffer_Sync[1]={0x80}; // Start at reg 0x00
// Define Write Buffer
char cI2C_Buffer_Write[2]={0x00};
// Define I2C Buffer Status
char cI2C_Buffer_Status=0;//unsigned
// Define I2C Status Bits:
int iI2C_STATUS_REG_ZYXOR=0;  // Bit 7 - X,Y or Z Data Overwritten
int iI2C_STATUS_REG_ZYXDA=0;  // Bit 3 - X,Y or Z New Data Available
// Define Loop Counters
```

```
int iLoop=0;
int iLoop_Counter;
// Define Device Address
char cGyro_I2C_Addr_Para=0x00;
char cGyro_I2C_Addr=0X00;

// Initialize Error
eError_No iError_No = Error_None;

// Initialize Error Level
eError_Level iError_Level = Error_Level_OK;

// Initialize Debug Output Level
// eDebugLevel iDebug_Level = Debug_Level_1;

/*
// Set Gyroscope Sensitivity Ratings
const double dGyro_Sens_250dps = 0.00875;
const double dGyro_Sens_500dps = 0.0175;
const double dGyro_Sens_2000dps = 0.075;
*/

// Define Angular Velocity Holder Variables For Data From Gyro Registry
int iAngVel_X_Flt=0;
int iAngVel_Y_Flt=0;
int iAngVel_Z_Flt=0;
double dAngVel_X_Flt=0;
double dAngVel_Y_Flt=0;
double dAngVel_Z_Flt=0;




// Define Function iGryro_I2CReadBuff
int iGryro_I2CReadBuff()
{

    // Create Handle to SYSFS File Stream
    sprintf(sBBB_I2C_BUS1_FILE,"/dev/i2c-%u",prmI2C_BusNo[0]);

    // Check I2C Bus Concatenation
    if (iDebug_Level >=Debug_Level_2)  { printf("Gyro Msg: I2C Bus File
Location: %s\n",sBBB_I2C_BUS1_FILE); }

    // Open Data IO Stream
    iFILE_BBB_I2C_Handle = open(sBBB_I2C_BUS1_FILE, O_RDWR);

    // Confirm I2C Bus Open
    if (iFILE_BBB_I2C_Handle<0)
    {
        // Error Handling
        if (iDebug_Level >=Debug_Level_1)  { printf("Gyro Error: Failed to
Open I2C Bus - Check Device/Device Tree\n"); }
        // Set Error Number
```

```
        iError_No = Error_No_I2CComm;
        // Set Error Level
        iError_Level = Error_Level_Critical;
        // Return Otuput Error Level
        // return -1;
    }

    // Ouput Program Flow
    if (iDebug_Level >=Debug_Level_3) {printf("Gyro Msg: Program Flow State
2: Connect To I2C Bus\nError No: %i Error Level: %i\n",iError_No,
iError_Level);}

    // Initiate I2C Communicaton
    if (iError_No == Error_None)
    {
        // Initiate Communicaiton

iIOCTL_BBB_I2C_Handle=ioctl(iFILE_BBB_I2C_Handle,I2C_SLAVE,cGyro_I2C_Addr);

        // Error Handling
        if (iIOCTL_BBB_I2C_Handle<0)
        {
            if (iDebug_Level >=Debug_Level_1) {printf("Gyro Error: Failed to
Communicate With Slave Device\n");}
            // Set Error Number
            iError_No = Error_No_I2CInit;
            // Set Error Level
            iError_Level = Error_Level_Critical;
            // Return Otuput Error Level
            // return -1;
        }
    }

   // Ouput Program Flow
    if (iDebug_Level >=Debug_Level_3) {printf("Gyro Msg: Program Flow State
3: Connect to I2C Device\nError No: %i Error Level: %i\n",iError_No,
iError_Level);}

    // Set Buffer Read Range
    if (iError_No == Error_None)
    {
        iWrite_BBB_I2C_Handle = write(iFILE_BBB_I2C_Handle, cI2C_Buffer_Sync,
1);
        // Error Handling
        if (iWrite_BBB_I2C_Handle !=1)
        {
            // Error Handling
            if (iDebug_Level >=Debug_Level_1) { printf("Gyro Error: Failed to
Communicate With Device and/or Set Buffer Read Range\n");}
            // Set Error Number
            iError_No = Error_No_I2CBuffSet;
            // Set Error Level
            iError_Level = Error_Level_Critical;
            // Return Otuput Error Level
            // return -1;
        }
```

```
        else
        {
            if (iDebug_Level >=Debug_Level_1) { printf("Gyro Msg: Buffer
Range Set\n");}
        }
    }

    // Ouput Program Flow
    if (iDebug_Level >=Debug_Level_3) {printf("Gyro Msg: Program Flow State
4: Write to Device Registry\nError No: %i Error Level: %i\n",iError_No,
iError_Level);}

    // Read Buffer Data
    if (iError_No == Error_None)
    {
        // Read and Store Registry Data in Buffer
        iI2C_BytesRead =
read(iFILE_BBB_I2C_Handle,cI2C_Buffer,iI2C_BufferSize);
        // Close File
        close(iFILE_BBB_I2C_Handle);

        // Error Handling - Confirm Buffer Size
        if (iI2C_BytesRead<0)
        {
            // Error on Buffer Read!
            if (iDebug_Level >=Debug_Level_1) { printf("Gyro Error: Failed to
Read Device Register. Bytes Read: %i\n",iI2C_BytesRead );}
            // Set Error Number
            iError_No = Error_No_I2CBuffRead;
            // Set Error Level
            iError_Level = Error_Level_Critical;
            // Return Otuput Error Level
            // return -1;
        }
        else
        {
            // Registry Data Read!
            if (iDebug_Level >=Debug_Level_1) { printf("Gyro Msg: Device
Register Read. Bytes Read: %i\n",iI2C_BytesRead );}

            // Confirm Data Synced
            if (cI2C_Buffer[cGyro_WhoAmIReg]!=cGyro_WhoAmI)
            {
                // Error on Buffer Sync!
                if (iDebug_Level >=Debug_Level_1) { printf("Gyro Error:
Buffer Data Invalid! Who_Am_I != 0xD3:
%#04x\n",cI2C_Buffer[cGyro_WhoAmIReg]);}
                // Set Error Number
                iError_No = Error_No_I2CBuffSync;
                // Set Error Level
                iError_Level = Error_Level_Critical;
                // Return Otuput Error Level
                // return -1;
            }
        }
    }
```

```
    // Set Return State
    if (iError_No==0){return 0;} else {return -1;}


}

// Define Function iGryro_I2CWriteBuff
int iGryro_I2CWriteBuff(char cRegister_Address, char cRegister_Value)
{

    // Create Handle to SYSFS File Stream
    sprintf(sBBB_I2C_BUS1_FILE,"/dev/i2c-%u",prmI2C_BusNo[0]);

    // Check I2C Bus Concatenation
    if (iDebug_Level >=Debug_Level_2)  { printf("Gyro Msg: I2C Bus File
Location: %s\n",sBBB_I2C_BUS1_FILE); }

    // Open Data IO Stream
    iFILE_BBB_I2C_Handle = open(sBBB_I2C_BUS1_FILE, O_RDWR);

    // Confirm I2C Bus Open
    if (iFILE_BBB_I2C_Handle<0)
    {
        // Error Handling
        if (iDebug_Level >=Debug_Level_1)  { printf("Gyro Error: Failed to
Open I2C Bus - Check Device/Device Tree\n"); }
        // Set Error Number
        iError_No = Error_No_I2CComm;
        // Set Error Level
        iError_Level = Error_Level_Critical;
        // Return Otuput Error Level
        // return -1;
    }

    // Ouput Program Flow
    if (iDebug_Level >=Debug_Level_3) {printf("Gyro Msg: Program Flow State
2: Connect To I2C Bus\nError No: %i Error Level: %i\n",iError_No,
iError_Level);}


    // Initiate I2C Communicaton
    if (iError_No == Error_None)
    {
        // Initiate Communicaiton

iIOCTL_BBB_I2C_Handle=ioctl(iFILE_BBB_I2C_Handle,I2C_SLAVE,cGyro_I2C_Addr);

        // Error Handling
        if (iIOCTL_BBB_I2C_Handle<0)
        {
            if (iDebug_Level >=Debug_Level_1) {printf("Gyro Error: Failed to
Communicate With Slave Device\n");}
            // Set Error Number
```

```
            iError_No = Error_No_I2CInit;
            // Set Error Level
            iError_Level = Error_Level_Critical;
            // Return Otuput Error Level
            // return -1;
        }
    }

    // Ouput Program Flow
    if (iDebug_Level >=Debug_Level_3) {printf("Gyro Msg: Program Flow State
3: Connect to I2C Device\nError No: %i Error Level: %i\n",iError_No,
iError_Level);}

    // Write Data to Register
    if (iError_No == Error_None)
    {

        // Set Gyro To Register Address!
        // Set Gyro Register Value
        cI2C_Buffer_Write[0]= cRegister_Address;//Address
        cI2C_Buffer_Write[1]= cRegister_Value;//Value
        iWrite_BBB_I2C_Handle = write(iFILE_BBB_I2C_Handle,
cI2C_Buffer_Write, 2);
        if (iWrite_BBB_I2C_Handle!=2)
        {
            // Error Handling
            if (iDebug_Level >=Debug_Level_1) { printf("Gyro Error: Failed to
Communicate With Device and/or Write To Device Register\n");}
            // Set Error Number
            iError_No = Error_No_I2CBuffWrite;
            // Set Error Level
            iError_Level = Error_Level_Critical;
            // Return Otuput Error Level
            // return -1;
        }
        else
        {
            // Power Mode Set
            if (iDebug_Level >=Debug_Level_1){printf("Gyro Msg: Data Written
to Gyro Register!\n");}
        }
    }
    // Close File
    close(iFILE_BBB_I2C_Handle);

    // Set Return State
    if (iError_No==0){return 0;} else {return -1;}
}

////// MAIN PROGRAM START ///////


// Read Parameter Data -Device Number
switch(prmI2C_DeviceNo[0])
{
```

```
    case  1: //iAngVel_250dps:
        cGyro_I2C_Addr=cGyro_I2C_Addr1;
        break;
    case  2:
        cGyro_I2C_Addr=cGyro_I2C_Addr2;
        break;
    default:
        cGyro_I2C_Addr=0x00;
        break;
}

// Read Parameter Data -Device Number
switch(prmDebug_InfoLevel[0])
{
    case  1:
        iDebug_Level=Debug_None;
        break;
    case  2:
        iDebug_Level=Debug_Level_0;
        break;
    case  3:
        iDebug_Level=Debug_Level_1;
        break;
    case  4:
        iDebug_Level=Debug_Level_2;
        break;
    case  5:
        iDebug_Level=Debug_Level_3;
        break;
    case  6:
        iDebug_Level=Debug_Level_4;
        break;
    case  7:
        iDebug_Level=Debug_Level_5;
        break;
    default:
        iDebug_Level=Debug_None;
        break;
}

// Start Line
if (iDebug_Level >=Debug_Level_0){printf("**** Start of Gyro S-Function Block
Execution **** \n");}

// Check Parameters
if (iDebug_Level >=Debug_Level_1){printf("Gyro Msg: Input Parameter Data:
I2CBus: %#04x I2CAddress: %#04x Range:
%s",prmI2C_BusNo[0],cGyro_I2C_Addr,sGyro_SetRange(prmGyro_AngVelRange[0]));}

// Ouput Program Flow
if (iDebug_Level >=Debug_Level_3) {printf("Gyro Msg: Program Flow State 1:
Parameters Initialized \nError No: %i Error Level: %i\n",iError_No,
iError_Level);}

// Run Init Routines
if (xD[0]==0)
```

```
{
    // Set Gyro to Normal Mode
    iI2C_Write=iGryro_I2CWriteBuff(cGyro_CtrlReg1,cGyro_NormalMode);
    // Check For Error
    if (iI2C_Write<0)
    {
        // Error Handling
        if (iDebug_Level >=Debug_Level_1){printf("Gyro Error: Failed to Set
Device Power Mode\n");}
    }
    else
    {
        // Error Handling
        if (iDebug_Level >=Debug_Level_1){printf("Gyro Msg: Device Set to
Power Mode\n");}
    }


    if (iError_No == Error_None)
    {
        // Set Measurement Range Based on Parameter Input
        if (iDebug_Level >=Debug_Level_1){printf("Gyro Msg: User Requested
Measurement Range: %s",sGyro_SetRange(prmGyro_AngVelRange[0]));}

        if (prmGyro_AngVelRange[0]>3)
        {
            // Error Handling
            if (iDebug_Level >=Debug_Level_1){printf("Gyro Error: Requested
Device Mesaurement Range Not Vaild!\n");}
            // Set Error Number
            iError_No = Error_No_ParamOutofRange;
            // Set Error Level
            iError_Level = Error_Level_Critical;
        }
        else
        {
            // Write User Selected Range To Gyro

iI2C_Write=iGryro_I2CWriteBuff(cGyro_CtrlReg4,cGyro_SetRange(prmGyro_AngVelRa
nge[0]));
            // Check For Error
            if (iI2C_Write<0)
            {
                // Error Handling
                if (iDebug_Level >=Debug_Level_1){printf("Gyro Error: Failed
to Set Device Measurement Range\n");}
            }
            else
            {
                // Error Handling
                if (iDebug_Level >=Debug_Level_1){printf("Gyro Msg: Device
Set to Requested Measurement Range\n");}
            }
        }

    }
```

```
    // Ouput Program Flow
    if (iDebug_Level >=Debug_Level_3) {printf("Gyro Msg: Program Flow State
1A: Init Routine Complete\nError No: %i Error Level: %i\n",iError_No,
iError_Level);}

}
if (iError_No == Error_None)
{
    // Read Gyro Buffer
    iI2C_Read=iGryro_I2CReadBuff();
}
// Good To Go! Run Initialization Routine on First Timestep

// Read Gyro Data
if (iError_No == Error_None)
{
    // Read Data Status Registry
    cI2C_StatReg_Temp=cI2C_Buffer[cGyrp_StatusReg];
    if (iDebug_Level >=Debug_Level_2){printf("Gyro Diagnostics: STATUS_REG
0x00: %#04x \n",cI2C_Buffer[cI2C_StatReg_Temp]);}

    // Check For Overwritten Data
    if (cI2C_StatReg_Temp &  0b10000000)
    {
        iI2C_STATUS_REG_ZYXOR=1;
    }

    // Check For New Data
    if (cI2C_StatReg_Temp &  0b00001000)
    {
        iI2C_STATUS_REG_ZYXDA=1;
    }

    if (iDebug_Level >=Debug_Level_2) {printf("Gyro Msg: Data Available: %i
Data Overwritten %i\n",iI2C_STATUS_REG_ZYXDA,iI2C_STATUS_REG_ZYXOR);}

    // Read Angular Velocity Data As
    if (iI2C_STATUS_REG_ZYXDA!=0)
    {
        // Write Function to Read and Combine Buffer Data

        // Get X-Angular Velocity Data
        iGyro_AngVel_Temp = cI2C_Buffer[cAngVelReg_X_MSB];
        iGyro_AngVel_Temp = (iGyro_AngVel_Temp<<8) |
cI2C_Buffer[cAngVelReg_X_LSB];
        if (iOpt_TwoComplement == true)
        {
            iGyro_AngVel_Temp = ~iGyro_AngVel_Temp + 1; // Calculate 2's
complement
        }
        // Output MBS and LSB
        if (iDebug_Level >=Debug_Level_0){printf("Gyro Msg: X AngVel Register
Values Read. MSB: %#04x ; LSB: %#04x
\n",cI2C_Buffer[cAngVelReg_X_MSB],cI2C_Buffer[cAngVelReg_X_LSB]);}
        // Convert Int (Int32) to Int16
```

```
        iGyro_AngVel_Temp=(short)iGyro_AngVel_Temp;
        // Convert Int16 to Double
        dGyro_AngVel_Temp =
(double)(iGyro_AngVel_Temp*dGyro_SetSens(prmGyro_AngVelRange[0]));
        // Set Output!
        outAngVel_X[0]=dGyro_AngVel_Temp;

        // Get Y-Accel Data
        iGyro_AngVel_Temp = 0;
        iGyro_AngVel_Temp = cI2C_Buffer[cAngVelReg_Y_MSB];
        iGyro_AngVel_Temp = (iGyro_AngVel_Temp<<8) |
cI2C_Buffer[cAngVelReg_Y_LSB];
        if (iOpt_TwoComplement == true)
        {
            iGyro_AngVel_Temp = ~iGyro_AngVel_Temp + 1; // Calculate 2's
complement
        }
        // Output MBS and LSB
        if (iDebug_Level >=Debug_Level_0){printf("Gyro Msg: Y AngVel Register
Values Read. MSB: %#04x ; LSB: %#04x
\n",cI2C_Buffer[cAngVelReg_Y_MSB],cI2C_Buffer[cAngVelReg_Y_LSB]);}
        // Convert Int (Int32) to Int16
        iGyro_AngVel_Temp=(short)iGyro_AngVel_Temp;
        // Convert Int16 to Double
        dGyro_AngVel_Temp =
(double)(iGyro_AngVel_Temp*dGyro_SetSens(prmGyro_AngVelRange[0]));
        // Set Output!
        outAngVel_Y[0]=dGyro_AngVel_Temp;

        // Get Z-Accel Data
        iGyro_AngVel_Temp = 0;
        iGyro_AngVel_Temp = cI2C_Buffer[cAngVelReg_Z_MSB];
        iGyro_AngVel_Temp = (iGyro_AngVel_Temp<<8) |
cI2C_Buffer[cAngVelReg_Z_LSB];
        if (iOpt_TwoComplement == true)
        {
            iGyro_AngVel_Temp = ~iGyro_AngVel_Temp + 1; // Calculate 2's
complement
        }
        // Output MBS and LSB
        if (iDebug_Level >=Debug_Level_0){printf("Gyro Msg: Z AngVel Register
Values Read. MSB: %#04x ; LSB: %#04x
\n",cI2C_Buffer[cAngVelReg_Z_MSB],cI2C_Buffer[cAngVelReg_Z_LSB]);}
        // Convert Int (Int32) to Int16
        iGyro_AngVel_Temp=(short)iGyro_AngVel_Temp;
        // Convert Int16 to Double
        dGyro_AngVel_Temp =
(double)(iGyro_AngVel_Temp*dGyro_SetSens(prmGyro_AngVelRange[0]));
        // Set Output!
        outAngVel_Z[0]=dGyro_AngVel_Temp;

        // Get Temperature
        outAmbTemp[0]=cI2C_Buffer[cAmbTempReg];

        // Ouput Program Flow
```

```
        if (iDebug_Level >=Debug_Level_3) {printf("Gyro Msg: Program Flow
State 5: Device Data Read\nError No: %i Error Level: %i\n",iError_No,
iError_Level);}

        // Print Output
        if (iDebug_Level >=Debug_Level_0){printf("Gyro Msg: Angular Vel  X:
%f Y: %f Z: %f \n",outAngVel_X[0],outAngVel_Y[0],outAngVel_Z[0]);}

        // Print Registry Data
        if (iDebug_Level >=Debug_Level_4)
        {

            // Print Select Registry Data
            printf("Gyro Diagnostics: WHO_AM_I 0x00: %#04x
\n",cI2C_Buffer[0x0f]);
            printf("Gyro Diagnostics: CTRL_REG1 0x00: %#04x
\n",cI2C_Buffer[0x20]);
            printf("Gyro Diagnostics: CTRL_REG2 0x00: %#04x
\n",cI2C_Buffer[0x21]);
            printf("Gyro Diagnostics: CTRL_REG3 0x00: %#04x
\n",cI2C_Buffer[0x22]);
            printf("Gyro Diagnostics: CTRL_REG4 0x00: %#04x
\n",cI2C_Buffer[0x23]);
            printf("Gyro Diagnostics: CTRL_REG5 0x00: %#04x
\n",cI2C_Buffer[0x24]);
            // Test
            for ( iLoop=0; iLoop<iI2C_BufferSize; iLoop++)
            {
                printf("Gyro Diagnostics: Data: Address %#04x = %#04x \n",
iLoop, cI2C_Buffer[iLoop]);
            }
        }
        if (iI2C_STATUS_REG_ZYXOR==1)
        {
            if (iDebug_Level >=Debug_Level_0){printf("Gyro Msg: Read Rate Too
Slow. Data Overwritten\n");}
        }

    }
    else
    {
        if (iDebug_Level >=Debug_Level_0){printf("Gyro Msg: No New Data
Available\n");}

    }
}
// Ouput Program Flow
if (iDebug_Level >=Debug_Level_3) {printf("Gyro Msg: Program Flow State 6:
Loop Step Complete\nError No: %i Error Level: %i\n",iError_No,
iError_Level);}

// Check For Critical Errors and Stop Simulation
if ((iError_No!=0)&&(iError_Level==Error_Level_Critical))
{
```

```
    // Stop Simulation If Critical Error
    if (iDebug_Level >=Debug_Level_0) {printf("Gyro Msg: Critical Error
Detected; Simulation Stopping! \nError No: %i Error Level: %i\n",iError_No,
iError_Level);}

    // Output Error Message:
    if (iDebug_Level >=Debug_Level_0) {printf("Gyro Error Details:
%s",sGyro_ErrMessage(iError_No));}


    // Stop Simulation
    outSimStop[0]=true;
}
else
{
    // Carry on
    outSimStop[0]=false;
}

# else
# endif
/* %%%-SFUNWIZ_wrapper_Outputs_Changes_END --- EDIT HERE TO _BEGIN */
}

/*
  * Updates function
  *
  */
void BBB_Driver_Gyroscope_L3G4200D_Update_wrapper(const real_T *outAngVel_X,
                          const real_T *outAngVel_Y,
                          const real_T *outAngVel_Z,
                          const int8_T *outAmbTemp,
                          const boolean_T *outSimStop ,
                          real_T *xD,
                          const uint8_T  *prmI2C_BusNo,  const int_T
p_width0,
                          const uint8_T  *prmI2C_DeviceNo,  const int_T
p_width1,
                          const uint8_T  *prmGyro_AngVelRange,  const int_T
p_width2,
                           const uint8_T *prmDebug_InfoLevel, const int_T
p_width3)
{
  /* %%%-SFUNWIZ_wrapper_Update_Changes_BEGIN --- EDIT HERE TO _END */
// Run Only on Target - BeagleBoneBlack */
# ifndef MATLAB_MEX_FILE

// Update Init Bit
if (xD[0]==0)
{
    // Set Init Complete
    xD[0]=1;

    // Output Update Compete
```

```
    if (iDebug_Level >=Debug_Level_1){printf("Gyro Msg: Program Flow State
6A: Init First Scan Complete\n");}

    //exit(EXIT_SUCCESS);
    //ssSetStopRequested(S, 1);
    //set_param(bdroot, 'SimulationCommand', 'stop');
}

// Print Gap
if (iDebug_Level >=Debug_Level_0){printf("\n");}


# else

// Do Nothing


#endif
/* %%%-SFUNWIZ_wrapper_Update_Changes_END --- EDIT HERE TO _BEGIN */
}
```

13.9 S-FUNCTION BLOCK WRAPPER CODE: HMC5883L MAGNETOMETER DRIVER BLOCK

```
/*
  *
  *   --- THIS FILE GENERATED BY S-FUNCTION BUILDER: 3.0 ---
  *
  *   This file is a wrapper S-function produced by the S-Function
  *   Builder which only recognizes certain fields.  Changes made
  *   outside these fields will be lost the next time the block is
  *   used to load, edit, and resave this file. This file will be overwritten
  *   by the S-function Builder block. If you want to edit this file by hand,
  *   you must change it only in the area defined as:
  *
  *        %%%-SFUNWIZ_wrapper_XXXXX_Changes_BEGIN
  *            Your Changes go here
  *        %%%-SFUNWIZ_wrapper_XXXXXX_Changes_END
  *
  *   For better compatibility with the Simulink Coder, the
  *   "wrapper" S-function technique is used.  This is discussed
  *   in the Simulink Coder User's Manual in the Chapter titled,
  *   "Wrapper S-functions".
  *
  *   Created: Sun Dec 14 05:08:04 2014
  */


/*
 * Include Files
 *
 */
#if defined(MATLAB_MEX_FILE)
#include "tmwtypes.h"
#include "simstruc_types.h"
#else
#include "rtwtypes.h"
#endif

/* %%%-SFUNWIZ_wrapper_includes_Changes_BEGIN --- EDIT HERE TO _END */
#include <math.h>
#include <math.h>
#include <stdio.h>
#include <stddef.h>
#include <time.h>
#include <stdlib.h>
#include <string.h>



# ifndef MATLAB_MEX_FILE

    #include <linux/i2c.h>
    #include <linux/i2c-dev.h>
    #include <sys/ioctl.h>
```

```
    #include <unistd.h>
    #include <sys/types.h>
    #include <sys/stat.h>
    #include <fcntl.h>

# else

    #include <mex.h>
    #include <simstruc.h>

# endif

// Define Compass I2C Address
# define cCompass_I2C_Addr1 0x1E

// Define Read/Write Command Reg
# define cCompass_I2C_Read 0x3D
# define cCompass_I2C_Write 0x3C

// Define Register Read Size
# define cCompass_I2C_BufferSize 0x0D

// Define Register Read Start Position
# define cCompass_I2C_StartPosReg 0x09
// Note that start position must be set to 0x09 instead of 0x00 as compas
// module automatically resets read pointer to start of data block registers
// "The HMC5883L will automatically re-point back to register 3 for the next
0x3D query"

// Define Compass ID (For Data Validation / CheckSum)
# define cCompass_IDA_Value 0x48
# define cCompass_IDB_Value 0x34
# define cCompass_IDC_Value 0x33

// Define Conversion Constant Gauss to MicroTesla
# define iCompass_Gauss_uTesla_Conv 100

// Define Compass Self Test Settings
# define cCompass_SelfTest_ConfigAReg 0x71 // 8-average 15 Hz, Positive
# define cCompass_SelfTest_Exit 0x70 // Exit Self Test
# define cCompass_SelfTest_LimMax 575 // Max Limit for Gain = 5
# define cCompass_SelfTest_LimMin 243 // Min Limit for Gain = 5

// Define Compass Registry Locations
typedef enum
{
    cCompass_ConfigAReg     =0x00,
    cCompass_ConfigBReg     =0x01,
    cCompass_ModeReg        =0x02,
    cMagField_X_MSBReg      =0x03,
    cMagField_X_LSBReg      =0x04,
    cMagField_Z_MSBReg      =0x05,
    cMagField_Z_LSBReg      =0x06,
    cMagField_Y_MSBReg      =0x07,
    cMagField_Y_LSBReg      =0x08,
```

```
    cCompass_StatusReg      =0x09,
    cCompass_IDAReg         =0x0A,
    cCompass_IDBReg         =0x0B,
    cCompass_IDCReg         =0x0C
} eCompass_RegVal;

// Define Compass Mode Settings
typedef enum
{
    cCompass_Mode_ContMeasure   =0x00,
    cCompass_Mode_SingleMeasure =0x01,
    cCompass_Mode_Idle          =0x02
} eCompass_ModeSet;

// Define Compass Gain Settings
typedef enum
{
    cCompass_Gain_1370  =0x00,
    cCompass_Gain_1090  =0x20,
    cCompass_Gain_820   =0x40,
    cCompass_Gain_660   =0x60,
    cCompass_Gain_440   =0x80,
    cCompass_Gain_390   =0xA0,
    cCompass_Gain_330   =0xC0,
    cCompass_Gain_230   =0xE0
} eCompass_GainSet;

// Define Compass Number of Samples for Averaged Output
typedef enum
{
    cCompass_AvrgSamples_1  =0x00,
    cCompass_AvrgSamples_2  =0x20,
    cCompass_AvrgSamples_4  =0x40,
    cCompass_AvrgSamples_8  =0x60
} eCompass_AvrgSampleSet;

// Define Compass Data Output Rate
typedef enum
{
    cCompass_DataOutRate_075    =0x00,
    cCompass_DataOutRate_150    =0x04,
    cCompass_DataOutRate_300    =0x08,
    cCompass_DataOutRate_750    =0x0C,
    cCompass_DataOutRate_1500   =0x10,
    cCompass_DataOutRate_3000   =0x14,
    cCompass_DataOutRate_7500   =0x18,
    cCompass_DataOutRate_Rsv    =0x1C
} eCompass_DataOutRateSet;

// Define Compass Measurement Mode
typedef enum
{
    cCompass_ModeMeas_Normal    =0x00,
    cCompass_ModeMeas_PosBias   =0x01,
    cCompass_ModeMeas_NegBias   =0x02,
    cCompass_ModeMeas_Rsv       =0x03
```

```
} eCompass_MeasModeSet;

// Function Options (1=On/0=Off)
// int iOpt_TwoComplement = 1; // Data To Be Treated as Two's Complement

// Delcare Global Variables
// int iLoop_Counter_DS=0;
int iLoop_DataNotReady_Counter=0;

// Define Debug Levels
typedef enum
{
    Debug_None,
    Debug_Level_0, // Basic Debug Info Output
    Debug_Level_1, // + Critical Info Only
    Debug_Level_2, // + Diagnostics Info
    Debug_Level_3, // + Program Flow Info
    Debug_Level_4, // + Registry Data
    Debug_Level_5  // + TBD
}eDebugLevel;

// Initialize Debug Output Level
// eDebugLevel iDebug_Level = Debug_None;

// Define Error Number
typedef enum
{
    Error_None=0,
    Error_No_I2CComm,
    Error_No_I2CInit,
    Error_No_I2CBuffSet,
    Error_No_I2CBuffRead,
    Error_No_I2CBuffSync,
    Error_No_I2CBuffWrite,
    Error_No_ParamOutofRange,
    // Define Compass Specific Error Numbers
    Error_Compass_No_SelfTest = 11
}eError_No;

// Define Error Severity Levels
typedef enum
{
    Error_Level_OK=0,
    Error_Level_Critical, // (Stop Simulation)
    Error_Level_Warning
}eError_Level;

// Initialize Error Level
// eError_Level iError_Level = Error_Level_OK;

// Define Parameter Selection: Compass Sensitivity
typedef enum
{
    iAngVel_250dps=1,
    iAngVel_500dps=2,
```

```
    iAngVel_2000dps=3
}eGyro_Range;
/* %%%-SFUNWIZ_wrapper_includes_Changes_END --- EDIT HERE TO _BEGIN */
#define u_width
#define y_width 1
/*
 * Create external references here.
 *
 */
/* %%%-SFUNWIZ_wrapper_externs_Changes_BEGIN --- EDIT HERE TO _END */
/* extern double func(double a); */
/* %%%-SFUNWIZ_wrapper_externs_Changes_END --- EDIT HERE TO _BEGIN */

/*
 * Output functions
 *
 */
void BBB_Driver_Compass_HMC5883L_Outputs_wrapper(real_T *outMagField_HX,
                          real_T *outMagField_HY,
                          real_T *outMagField_HZ,
                          real_T *outMagField_HE,
                          real_T *outMagField_Heading,
                          real_T *outSimStop ,
                  const real_T  *xD,
                          const uint8_T  *prmI2C_DeviceNo, const int_T
p_width0,
                          const uint8_T  *prmI2C_BusNo, const int_T
p_width1,
                          const uint8_T  *prmCompass_DataOutAvrg, const int_T
p_width2,
                          const uint8_T  *prmCompass_DataOutRate, const int_T
p_width3,
                          const uint8_T  *prmCompass_MeasureMode, const int_T
p_width4,
                          const uint8_T  *prmCompass_GainValue, const int_T
p_width5,
                          const uint8_T  *prmCompass_OpMode, const int_T
p_width6,
                          const uint8_T  *prmCompass_HeadingMode, const int_T
p_width7,
                          const real_T  *prmCompass_HeadingOffset, const
int_T  p_width8,
                          const real_T  *prmCompass_DeclinationAng, const
int_T  p_width9,
                          const boolean_T  *prmCompass_SelfTest, const int_T
p_width10,
                          const uint8_T  *prmCompass_SelfTest_GainValue,
const int_T  p_width11,
                          const uint8_T  *prmDebug_InfoLevel, const int_T
p_width12)
{
/* %%%-SFUNWIZ_wrapper_Outputs_Changes_BEGIN --- EDIT HERE TO _END */
// Run Only on Target - BeagleBoneBlack */
# ifndef MATLAB_MEX_FILE

// Define File Sys Pointer - For I2C Bus#1 -> Make Parameter
```

```
int iFILE_BBB_I2C_Handle=0;
int iIOCTL_BBB_I2C_Handle=0;
int iSYNC_BBB_I2C_Handle=0;
int iWRITE_BBB_I2C_Handle=0;
// Function Outputs
int iI2C_Read=0;
int iI2C_Write=0;

//Define Device Driver sysfs Location
char sBBB_I2C_BUS1_FILE[100]="";

// Define Internal Variables Data
int iCompass_MagField_X=0;
int iCompass_MagField_Y=0;
int iCompass_MagField_Z=0;
int iCompass_MagField_Temp=0;

// Declare Temporary Variables For Bit Manipulation
double dCompass_MagField_Temp=0.0f;

// Declare Variables For Device Registry Read/Write
unsigned char cI2C_StatReg_Temp=0;

// Define I2C Parameters
int iI2C_BytesRead=0;
// Define I2C Buffer Size
const int iI2C_BufferSize=(int)cCompass_I2C_BufferSize;
// Define I2C Buffer Dump Variable
char cI2C_Buffer[cCompass_I2C_BufferSize]={0x00}; //unsigned
cCompass_I2C_BufferSize
// Define I2C Temp Buffer Dump Variable
char cI2C_Buffer_Temp[cCompass_I2C_BufferSize]={0x00};
// Define I2C Buffer Dump Start Position.
char cI2C_Buffer_Sync[1]={cCompass_I2C_StartPosReg}; // Start at reg 0x00
cCompass_I2C_StartPosReg
// Define I2C Read Buffer Size
char cI2C_Buffer_Read[1]={cCompass_I2C_BufferSize};
// Define I2C Buffer Offser For Remapping
char cI2C_Buffer_Offset=(cCompass_I2C_BufferSize - cCompass_I2C_StartPosReg);
// Define Write Buffer
char cI2C_Buffer_Write[2]={0x00};
// Define I2C Buffer Status
char cI2C_Buffer_Status=0;//unsigned
// Define I2C Status Bits:
int iI2C_STATUS_REG_DATARDY=0;  // Bit 0 - X,Y or Z Data Overwritten
int iI2C_STATUS_REG_DATALOCK=0;  // Bit 1 - X,Y or Z New Data Available
// Define Loop Counters
int iLoop=0;
int iLoop_Counter;
// Define Device Address
char cCompass_I2C_Addr_Para=0x00;
char cCompass_I2C_Addr=0X00;
// Define Operation Mode Parameter Placeholder
char cCompass_OpMode=0x00;
// Define Measurement Mode Parameter Placeholder
char cCompass_MeasureMode=0x00;
```

```
// Define Out Average Data Parameter Placeholder
char cCompass_DataOutAvrg=0x00;
// Define Data Out Rate Parameter Placeholder
char cCompass_DataOutRate=0x00;
// Define Gain Value Parameter Placeholder
char cCompass_GainValue=0x00;
// Define Temp Register Storage Variables
char cCompass_ConfigRegA_Val=0x00;
char cCompass_ConfigRegB_Val=0x00;

// Initialize Error
eError_No iError_No = Error_None;

// Initialize Error Level
eError_Level iError_Level = Error_Level_OK;

// Initialize Debug Output Level - From Parameter Selection
eDebugLevel iDebug_Level = Debug_Level_1;

// Function Options (1=On/0=Off)
int iOpt_TwoComplement = 1; // Data To Be Treated as Two's Complement

// ************************************//
// Define Function iCompass_I2CReadBuff
// ************************************//
int iCompass_I2CReadBuff()
{

    // Create Handle to SYSFS File Stream
    sprintf(sBBB_I2C_BUS1_FILE,"/dev/i2c-%u",prmI2C_BusNo[0]);

    // Check I2C Bus Concatenation
    if (iDebug_Level >=Debug_Level_2)  { printf("Compass Msg: I2C Bus File
Location: %s\n",sBBB_I2C_BUS1_FILE); }

    // Open Data IO Stream
    iFILE_BBB_I2C_Handle = open(sBBB_I2C_BUS1_FILE, O_RDWR);

    // Confirm I2C Bus Open
    if (iFILE_BBB_I2C_Handle<0)
    {
        // Error Handling
        if (iDebug_Level >=Debug_Level_1)  { printf("Compass Error: Failed to
Open I2C Bus - Check Device/Device Tree\n"); }
        // Set Error Number
        iError_No = Error_No_I2CComm;
        // Set Error Level
        iError_Level = Error_Level_Critical;
        // Return Otuput Error Level
        // return -1;
    }

    // Ouput Program Flow
```

```
    if (iDebug_Level >=Debug_Level_3) {printf("Compass Msg: Program Flow
State 2: Connect To I2C Bus\nError No: %i Error Level: %i\n",iError_No,
iError_Level);}

    // Initiate I2C Communicaton
    if (iError_No == Error_None)
    {
        // Initiate Communicaiton

iIOCTL_BBB_I2C_Handle=ioctl(iFILE_BBB_I2C_Handle,I2C_SLAVE,cCompass_I2C_Addr)
;

        // Error Handling
        if (iIOCTL_BBB_I2C_Handle<0)
        {
            if (iDebug_Level >=Debug_Level_1) {printf("Compass Error: Failed
to Communicate With Slave Device\n");}
            // Set Error Number
            iError_No = Error_No_I2CInit;
            // Set Error Level
            iError_Level = Error_Level_Critical;
            // Return Otuput Error Level
            // return -1;
        }
    }

   // Ouput Program Flow
    if (iDebug_Level >=Debug_Level_3) {printf("Compass Msg: Program Flow
State 3: Connect to I2C Device\nError No: %i Error Level: %i\n",iError_No,
iError_Level);}

    // Set Buffer Start Position
    if (iError_No == Error_None)
    {
        iWRITE_BBB_I2C_Handle = write(iFILE_BBB_I2C_Handle,
cI2C_Buffer_Sync,sizeof(cI2C_Buffer_Sync));
        // Error Handling
        if (iWRITE_BBB_I2C_Handle!=sizeof(cI2C_Buffer_Sync))
        {
            // Error Handling
            if (iDebug_Level >=Debug_Level_1) { printf("Compass Error: Failed
to Communicate With Device and/or Set Buffer Read Range\n");}
            // Set Error Number
            iError_No = Error_No_I2CBuffSet;
            // Set Error Level
            iError_Level = Error_Level_Critical;
            // Return Otuput Error Level
            // return -1;
        }
        else
        {
            if (iDebug_Level >=Debug_Level_1) { printf("Compass Msg: Register
Read Start Position Set\n");}
        }
    }
```

```
    // Ouput Program Flow
    if (iDebug_Level >=Debug_Level_3) {printf("Compass Msg: Program Flow
State 4: Write Registry Read Start Position Set\nError No: %i Error Level:
%i\n",iError_No, iError_Level);}

    // Read Buffer Data
    if (iError_No == Error_None)
    {
        // Read and Store Registry Data in Buffer
        iI2C_BytesRead =
read(iFILE_BBB_I2C_Handle,cI2C_Buffer_Temp,sizeof(cI2C_Buffer_Temp)); //
iI2C_BufferSize
        // Close File
        close(iFILE_BBB_I2C_Handle);

        // Error Handling - Confirm Buffer Size
        if (iI2C_BytesRead<0)
        {
            // Error on Buffer Read!
            if (iDebug_Level >=Debug_Level_1) { printf("Compass Error: Failed
to Read Device Register. Bytes Read: %i\n",iI2C_BytesRead );}
            // Set Error Number
            iError_No = Error_No_I2CBuffRead;
            // Set Error Level
            iError_Level = Error_Level_Critical;
            // Return Otuput Error Level
            // return -1;
        }
        else
        {
            // Registry Data Read!
            if (iDebug_Level >=Debug_Level_1) { printf("Compass Msg: Device
Register Read. Bytes Read: %i\n",iI2C_BytesRead );}

            // Print Registry Data
            if (iDebug_Level >=Debug_Level_4)
            {
                // Output Buffer Data
                for ( iLoop=0; iLoop<sizeof(cI2C_Buffer_Temp); iLoop++)
                {
                    printf("Compass Diagnostics: Temp Buffer Address %#04x =
%#04x \n", iLoop, cI2C_Buffer_Temp[iLoop]);
                }
            }

            // Re-Map Buffer
            for ( iLoop=0; iLoop<(sizeof(cI2C_Buffer)-cI2C_Buffer_Offset);
iLoop++)
            {

cI2C_Buffer[iLoop]=cI2C_Buffer_Temp[iLoop+cI2C_Buffer_Offset];
            }
            for ( iLoop=(sizeof(cI2C_Buffer)-cI2C_Buffer_Offset);
iLoop<sizeof(cI2C_Buffer); iLoop++)
            {
```

```
                cI2C_Buffer[iLoop]=cI2C_Buffer_Temp[iLoop-
(sizeof(cI2C_Buffer)-cI2C_Buffer_Offset)];
            }

            // Print Registry Data
            if (iDebug_Level >=Debug_Level_4)
            {
                // Output Buffer Data
                for ( iLoop=0; iLoop<sizeof(cI2C_Buffer); iLoop++)
                {
                    printf("Compass Diagnostics: Buffer Address %#04x = %#04x
\n", iLoop, cI2C_Buffer[iLoop]);
                }
            }

            // Confirm Data Synced
            if
((cI2C_Buffer[cCompass_IDAReg]!=cCompass_IDA_Value)||(cI2C_Buffer[cCompass_ID
BReg]!=cCompass_IDB_Value)||(cI2C_Buffer[cCompass_IDCReg]!=cCompass_IDC_Value
))
            {
                // Error on Buffer Sync!
                if (iDebug_Level >=Debug_Level_1) { printf("Compass Error:
Buffer Data Invalid! ID != 0x48,0x34,0x33 :
%#04x,%#04x,%#04x\n",cI2C_Buffer[cCompass_IDAReg],cI2C_Buffer[cCompass_IDBReg
],cI2C_Buffer[cCompass_IDCReg]);}
                // Set Error Number
                iError_No = Error_No_I2CBuffSync;
                // Set Error Level
                iError_Level = Error_Level_Critical;
                // Return Otuput Error Level
                // return -1;
            }
        }
    }

    // Set Return State
    if (iError_No==0){return 0;} else {return -1;}

}

// ************************************//
// Define Function iCompass_I2CWriteBuff
// ************************************//
int iCompass_I2CWriteBuff(char cRegister_Address, char cRegister_Value)
{

    // Create Handle to SYSFS File Stream
    sprintf(sBBB_I2C_BUS1_FILE,"/dev/i2c-%u",prmI2C_BusNo[0]);

    // Check I2C Bus Concatenation
    if (iDebug_Level >=Debug_Level_2)  { printf("Compass Msg: I2C Bus File
Location: %s\n",sBBB_I2C_BUS1_FILE); }

    // Open Data IO Stream
```

```
    iFILE_BBB_I2C_Handle = open(sBBB_I2C_BUS1_FILE, O_RDWR);

    // Confirm I2C Bus Open
    if (iFILE_BBB_I2C_Handle<0)
    {
        // Error Handling
        if (iDebug_Level >=Debug_Level_1)  { printf("Compass Error: Failed to
Open I2C Bus - Check Device/Device Tree\n"); }
        // Set Error Number
        iError_No = Error_No_I2CComm;
        // Set Error Level
        iError_Level = Error_Level_Critical;
        // Return Otuput Error Level
        // return -1;
    }

    // Ouput Program Flow
    if (iDebug_Level >=Debug_Level_3) {printf("Compass Msg:
iCompass_I2CWriteBuff State 1: Connect To I2C Bus\nError No: %i Error Level:
%i\n",iError_No, iError_Level);}


    // Initiate I2C Communicaton
    if (iError_No == Error_None)
    {
        // Initiate Communicaiton

iIOCTL_BBB_I2C_Handle=ioctl(iFILE_BBB_I2C_Handle,I2C_SLAVE,cCompass_I2C_Addr)
;

        // Error Handling
        if (iIOCTL_BBB_I2C_Handle<0)
        {
            if (iDebug_Level >=Debug_Level_1) {printf("Compass Error: Failed
to Communicate With Slave Device\n");}
            // Set Error Number
            iError_No = Error_No_I2CInit;
            // Set Error Level
            iError_Level = Error_Level_Critical;
            // Return Otuput Error Level
            // return -1;
        }
    }

    // Ouput Program Flow
    if (iDebug_Level >=Debug_Level_3) {printf("Compass Msg:
iCompass_I2CWriteBuff State 2: Connect to I2C Device\nError No: %i Error
Level: %i\n",iError_No, iError_Level);}

    // Write Data to Register
    if (iError_No == Error_None)
    {
        // Set Compass To Register Address!
        // Set Compass Register Value
        cI2C_Buffer_Write[0]= cRegister_Address;    //cRegister_Address;
```

```
        cI2C_Buffer_Write[1]= cRegister_Value;  //cRegister_Value
        iWRITE_BBB_I2C_Handle = write(iFILE_BBB_I2C_Handle,
cI2C_Buffer_Write, sizeof(cI2C_Buffer_Write));
        if (iDebug_Level >=Debug_Level_1) { printf("Compass Diag: Size of
Write Data: %i Handle Output:
%i\n",sizeof(cI2C_Buffer_Write),iWRITE_BBB_I2C_Handle);}

        if (iWRITE_BBB_I2C_Handle!=sizeof(cI2C_Buffer_Write))
        {
            // Error Handling
            if (iDebug_Level >=Debug_Level_1) { printf("Compass Error: Failed
to Communicate With Device and/or Write To Device Register\n");}
            // Set Error Number
            iError_No = Error_No_I2CBuffWrite;
            // Set Error Level
            iError_Level = Error_Level_Critical;
            // Return Otuput Error Level
            // return -1;
        }
        else
        {
            // Power Mode Set
            if (iDebug_Level >=Debug_Level_1){printf("Compass Msg:
iCompass_I2CWriteBuff State 3: Data Written to Compass Register!\n");}
        }
    }
    // Close File
    close(iFILE_BBB_I2C_Handle);

    // Set Return State
    if (iError_No==0){return 0;} else {return -1;}
}

// ************************************//
// Define Function sCompass_ErrMessage
// ************************************//
const char *sCompass_ErrMessage(eError_No iCompass_ErrNo)
{
    //Set Error Message Based on Error Number
    switch(iCompass_ErrNo)
    {
       case  Error_None:
            return("No Error Active. System A-OK!\n");
            break;
        case  Error_No_I2CComm:
            return("Failed to Open I2C Bus\n - Check Bus Number\n - Check
SCL/SCA Lines\n"
                   " - Check Device\n - Check Device Tree on BBB\n");
            break;
        case Error_No_I2CInit:
            return("Failed to Communicate With Slave Device\n - Check Bus
Number\n - Check SCL/SCA Lines\n"
                   " - Check Device Address\n - Check Device Tree on BBB\n");
            break;
        case Error_No_I2CBuffSet:
```

```
            return("Failed to Set Buffer Read Range\n - Check Bus Number\n -
Check SCL/SCA Lines\n"
                   " - Check Device Address\n - Check Device Tree on BBB\n");
            break;
        case Error_No_I2CBuffRead:
            return("Failed to Read Data in Registry Buffer\n - Check Bus
Number\n - Check SCL/SCA Lines\n"
                   " - Check Device Address\n - Check Device Tree on BBB\n");
            break;
        case Error_No_I2CBuffSync:
            return("Failed to Validate Registry Data - Inconsistent Data
Stream\n"
                    " - Check For Multiple Master Access to Slave Device on
I2C Bus Number\n"
                    " - Check SCL/SCA Lines\n - Check Device Address\n -
Check Device Tree on BBB\n");
            break;
        case Error_No_I2CBuffWrite:
            return("Failed to Write Data to Device Registry\n - Check Bus
Traffic/Interference\n"
                   " - Check Bus Number\n - Check SCL/SCA Lines\n - Check
Device Address\n - Check Device Tree on BBB\n");
            break;
        case Error_No_ParamOutofRange:
            return("Parameter Out Of Range\n - Check Block Parameter Input
Options\n");
            break;
        case Error_Compass_No_SelfTest:
            return("Compass Self Test Failed\n - Adjust Self Test Gain and
Try Again\n");
            break;
        default:
            // Error -Return Default
            return("Error Code Not Recognized\n");
            break;
    }
}

// ************************************//
// Define Function iSet_Debug_Level
// ************************************//
int iSet_Debug_Level(eDebugLevel iDebug_Level_X)
{

    switch(iDebug_Level_X)
    {
        case  1:
            return (Debug_None);
            break;
        case  2:
            return (Debug_Level_0);
            break;
        case  3:
            return (Debug_Level_1);
            break;
        case  4:
```

```
            return (Debug_Level_2);
            break;
        case  5:
            return (Debug_Level_3);
            break;
        case  6:
            return (Debug_Level_4);
            break;
        case  7:
            return (Debug_Level_5);
            break;
        default:
            return (Debug_None);
            break;
    }
}
// ************************************//
// Define Function cCompass_SetGain
// ************************************//
char cCompass_SetGain(iprmCompass_GainValue)
{
    // Read Selected Gain Data
    switch(iprmCompass_GainValue)
    {
        case  1:
            return(cCompass_Gain_1370);
            break;
        case  2:
            return(cCompass_Gain_1090);
            break;
        case  3:
            return(cCompass_Gain_820);
            break;
        case  4:
            return(cCompass_Gain_660);
            break;
        case  5:
            return(cCompass_Gain_440);
            break;
        case  6:
            return(cCompass_Gain_390);
            break;
        case  7:
            return(cCompass_Gain_330);
            break;
        case  8:
            return(cCompass_Gain_230);
            break;
        default:
            return(0x00);
            break;
    }
}
// ************************************//
// Define Function cCompass_SetDataOutRate
// ************************************//
char cCompass_SetDataOutRate(iprmCompass_DataOutRate)
```

```
{
    // Read Selected Data Output Rate
    switch(iprmCompass_DataOutRate)
    {
        case  1:
            return(cCompass_DataOutRate_075);
            break;
        case  2:
            return(cCompass_DataOutRate_150);
            break;
        case  3:
            return(cCompass_DataOutRate_300);
            break;
        case  4:
            return(cCompass_DataOutRate_750);
            break;
        case  5:
            return(cCompass_DataOutRate_1500);
            break;
        case  6:
            return(cCompass_DataOutRate_3000);
            break;
        case  7:
            return(cCompass_DataOutRate_7500);
            break;
        case  8:
            return(cCompass_DataOutRate_Rsv);
            break;
        default:
            return(0x00);
            break;
    }
}

// ************************************//
// Define Function cCompass_SetDataOutAvrg
// ************************************//
char cCompass_SetDataOutAvrg(iprmCompass_DataOutAvrg)
{
    // Read Selected Data Samples for Average Calculation of Output Data
    switch(iprmCompass_DataOutAvrg)
    {
        case  1:
            return(cCompass_AvrgSamples_1);
            break;
        case  2:
            return(cCompass_AvrgSamples_2);
            break;
        case  3:
            return(cCompass_AvrgSamples_4);
            break;
        case  4:
            return(cCompass_AvrgSamples_8);
            break;
        default:
            return(0x00);
            break;
```

```
    }
}
// ************************************//
// Define Function cCompass_SetMeasureMode
// ************************************//
char cCompass_SetMeasureMode(iprmCompass_MeasureMode)
{
    // Read Selected Compass Measurement Mode
    switch(iprmCompass_MeasureMode)
    {
        case  1:
            return(cCompass_ModeMeas_Normal);
            break;
        case  2:
            return(cCompass_ModeMeas_PosBias);
            break;
        case  3:
            return(cCompass_ModeMeas_NegBias);
            break;
        case  4:
            return(cCompass_ModeMeas_Rsv);
            break;
        default:
            return(0x00);
            break;
    }
}

// ************************************//
// Define Function cCompass_SetOpMode
// ************************************//
char cCompass_SetOpMode(iprmCompass_OpMode)
{
    // Read Selected Compass Mode Operation Mode
    switch(iprmCompass_OpMode)
    {
        case  1:
            return(cCompass_Mode_ContMeasure);
            break;
        case  2:
             return(cCompass_Mode_SingleMeasure);
            break;
        case  3:
             return(cCompass_Mode_Idle);
            break;
        default:
             return(0x00);
            break;
    }
}
// ************************************//
// Define Function cCompass_GetGainVal
// ************************************//
int cCompass_GetGainVal(eCompass_GainSet iCompass_GainSelect)
{
    // Read Gain Setting And Return Corresponding Gain Conversion (LSb/Gauss)
    switch(iCompass_GainSelect)
```

```
    {
        case  cCompass_Gain_1370:
            return(1370);
            break;
        case  cCompass_Gain_1090:
            return(1090);
            break;
        case  cCompass_Gain_820:
            return(820);
            break;
        case  cCompass_Gain_660:
            return(660);
            break;
        case  cCompass_Gain_440:
            return(440);
            break;
        case  cCompass_Gain_390:
            return(390);
            break;
        case  cCompass_Gain_330:
            return(330);
            break;
        case  cCompass_Gain_230:
            return(230);
            break;
        default:
            return(0);
            break;
    }
}
// ************************************//
// Define Function vCompass_Initialize
// Description:
// Set Compass Parameters
// ************************************//
void vCompass_Initialize()
{
    // Combine Bit Data For Registry A
    cCompass_ConfigRegA_Val =
(cCompass_SetDataOutAvrg(prmCompass_DataOutAvrg[0]) |
cCompass_SetDataOutRate(prmCompass_DataOutRate[0]) |
cCompass_SetMeasureMode(prmCompass_MeasureMode[0]));
    // Verify Registry A Data
    if (iDebug_Level >=Debug_Level_3) {printf("Compass Msg: Composite
Registry A Data: %#04x\n",cCompass_ConfigRegA_Val);}
    // Set Compass Data Output Average, Data Output Rate and Compass
Measurement Mode

iI2C_Write=iCompass_I2CWriteBuff(cCompass_ConfigAReg,cCompass_ConfigRegA_Val)
;
    // Check For Error
    if (iI2C_Write<0)
    {
        // Error Handling
        if (iDebug_Level >=Debug_Level_1){printf("Compass Error: Failed to
Set Device Output and Measurement Modes\n");}
    }
```

```
    else
    {
        // Error Handling
        if (iDebug_Level >=Debug_Level_1){printf("Compass Msg: Device Output
and Measurement Modes Set\n");}
    }

    // Set Compass Gain Parameter For Registry B
    if (iError_No == Error_None)
    {
        // Write User Selected Gain To Compass

iI2C_Write=iCompass_I2CWriteBuff(cCompass_ConfigBReg,cCompass_SetGain(prmComp
ass_GainValue[0]));
        // Check For Error
        if (iI2C_Write<0)
        {
            // Error Handling
            if (iDebug_Level >=Debug_Level_1){printf("Compass Error: Failed
to Set Device Gain Value Range\n");}
        }
        else
        {
            // Error Handling
            if (iDebug_Level >=Debug_Level_1){printf("Compass Msg: Device Set
to Requested Gain Value\n");}
        }
    }

    // Set Compass Mode
    if (iError_No == Error_None)
    {
        // Write User Selected Mode To Compass

iI2C_Write=iCompass_I2CWriteBuff(cCompass_ModeReg,cCompass_SetOpMode(prmCompa
ss_OpMode[0]));
        // Check For Error
        if (iI2C_Write<0)
        {
            // Error Handling
            if (iDebug_Level >=Debug_Level_1){printf("Compass Error: Failed
to Set Device Operation Mode \n");}
        }
        else
        {
            // Error Handling
            if (iDebug_Level >=Debug_Level_1){printf("Compass Msg: Device Set
to Requested Operation Mode \n");}
        }
    }

    // Output Info - Init Complete
    if (iError_No == Error_None)
    {
        // Ouput Program Flow
```

```
        if (iDebug_Level >=Debug_Level_3) {printf("Compass Msg: Program Flow
State 1A: Init Routine Complete\nError No: %i Error Level: %i\n",iError_No,
iError_Level);}
    }
    else
    {
        // Ouput Program Flow
        if (iDebug_Level >=Debug_Level_3) {printf("Compass Msg: Program Flow
State 1A: Init Routine Complete w/ Errors: \nError No: %i Error Level:
%i\n",iError_No, iError_Level);}
    }
}

// ************************************//
// Define Function iCompass_TwoComplement
// ************************************//
short iCompass_TwoComplement(bOpt_TwoComplement,iTwoComplement_Int)
{
    // Declare Local Variable
    short _iTwoComplement_Temp=0;
    // Check For Two's Complement Option Activated
    if (bOpt_TwoComplement == true)
    {
        // Transfer Input Data To Temp Variable
        _iTwoComplement_Temp = iTwoComplement_Int;
        // Start Conversion From 2s Complement to Base To Integer - Check
Sign MSb
        if ((short)iTwoComplement_Int &  0b1000000000000000)
        {
            // Flip All Bits and Add One
            _iTwoComplement_Temp = ~_iTwoComplement_Temp + 1;
            _iTwoComplement_Temp= -_iTwoComplement_Temp;
        }
        else
        {
            // No Change if MSb = 0
            // _iTwoComplement_Temp = _iTwoComplement_Temp;
        }

        return _iTwoComplement_Temp;
    }
}


// ************************************//
// Define Function vCompass_SelfTest
// ************************************//
void vCompass_SelfTest()
{
    int _iSelfTest_AllAxisPassed=0;
    int _iSelfTest_XAxis_Passed=0;
    int _iSelfTest_YAxis_Passed=0;
    int _iSelfTest_ZAxis_Passed=0;
    int _iSelfTest_XAxis_Average=0;
    int _iSelfTest_YAxis_Average=0;
    int _iSelfTest_ZAxis_Average=0;
```

```
    int _iSelfTest_LimMax=0;
    int _iSelfTest_LimMin=0;
    int _iLoop_Counter=0;
    int _iSelfTest_Average_Count=8;

    // Set Registry A For Self Test Mode
    cCompass_ConfigRegA_Val = cCompass_SelfTest_ConfigAReg;
    // Verify Registry A Data
    if (iDebug_Level >=Debug_Level_3) {printf("Compass Msg (Self Test):
Composite Registry A Data : %#04x\n",cCompass_ConfigRegA_Val);}
    // Set Compass Data Output Average, Data Output Rate and Compass
Measurement Mode

iI2C_Write=iCompass_I2CWriteBuff(cCompass_ConfigAReg,cCompass_ConfigRegA_Val)
;
    // Check For Error
    if (iI2C_Write<0)
    {
        // Error Handling
        if (iDebug_Level >=Debug_Level_1){printf("Compass Error (Self Test):
Failed to Set Device Output and Measurement Modes\n");}
    }
    else
    {
        // Error Handling
        if (iDebug_Level >=Debug_Level_1){printf("Compass Msg (Self Test):
Device Output and Measurement Modes Set\n");}
    }

    // Set Registry B Seft Test Gain Parameter
    if (iError_No == Error_None)
    {
        // Write User Selected Gain To Compass

iI2C_Write=iCompass_I2CWriteBuff(cCompass_ConfigBReg,cCompass_SetGain(prmComp
ass_SelfTest_GainValue[0]));
        // Check For Error
        if (iI2C_Write<0)
        {
            // Error Handling
            if (iDebug_Level >=Debug_Level_1){printf("Compass Error (Self
Test): Failed to Set Device Gain Value Range\n");}
        }
        else
        {
            // Error Handling
            if (iDebug_Level >=Debug_Level_1){printf("Compass Msg (Self
Test): Device Set to Requested Gain Value\n");}
        }
    }

    // Set Compass Mode to Continuous for Self Test
    if (iError_No == Error_None)
    {
        // Write User Selected Mode To Compass
```

```
iI2C_Write=iCompass_I2CWriteBuff(cCompass_ModeReg,cCompass_Mode_ContMeasure);
        // Check For Error
        if (iI2C_Write<0)
        {
            // Error Handling
            if (iDebug_Level >=Debug_Level_1){printf("Compass Error (Self
Test): Failed to Set Device Operation Mode \n");}
        }
        else
        {
            // Error Handling
            if (iDebug_Level >=Debug_Level_1){printf("Compass Msg (Self
Test): Device Set to Requested Operation Mode \n");}
        }
    }

    // Clear Out Registry Data With Previous Gain Reading via Read Command
    iI2C_Read=iCompass_I2CReadBuff();

    // Wait For New Data - Sleep 1 second
    sleep(1);

    // Take Multiple Readings
    for ( _iLoop_Counter = 1; _iLoop_Counter <=_iSelfTest_Average_Count ;
_iLoop_Counter++ )
    {
        // Re-Read Registry
        iI2C_Read=iCompass_I2CReadBuff();

        // Get X-Axis Component of Magnetic Field
        iCompass_MagField_Temp=0;
        dCompass_MagField_Temp=0.0f;
        iCompass_MagField_Temp = cI2C_Buffer[cMagField_X_MSBReg];
        iCompass_MagField_Temp = (iCompass_MagField_Temp<<8) |
cI2C_Buffer[cMagField_X_LSBReg];
        // Check For Two's Complement Conversion

iCompass_MagField_Temp=iCompass_TwoComplement(iOpt_TwoComplement,iCompass_Mag
Field_Temp);
        // Output MBS and LSB
        if (iDebug_Level >=Debug_Level_0){printf("Compass Msg: X Field> MSB:
%#04x ; LSB: %#04x Count: %#06x Value:
%i\n",cI2C_Buffer[cMagField_X_MSBReg],cI2C_Buffer[cMagField_X_LSBReg],iCompas
s_MagField_Temp,iCompass_MagField_Temp);}
        // Convert Int (Int32) to Int16
        _iSelfTest_XAxis_Average += abs((short)iCompass_MagField_Temp);

        // Get Y-Axis Component of Magnetic Field
        iCompass_MagField_Temp=0;
        dCompass_MagField_Temp=0.0f;
        iCompass_MagField_Temp = cI2C_Buffer[cMagField_Y_MSBReg];
        iCompass_MagField_Temp = (iCompass_MagField_Temp<<8) |
cI2C_Buffer[cMagField_Y_LSBReg];
        // Check For Two's Complement Conversion
```

```
iCompass_MagField_Temp=iCompass_TwoComplement(iOpt_TwoComplement,iCompass_Mag
Field_Temp);
        // Output MBS and LSB
        if (iDebug_Level >=Debug_Level_0){printf("Compass Msg: Y Field> MSB:
%#04x ; LSB: %#04x Count: %#06x Value:
%i\n",cI2C_Buffer[cMagField_Y_MSBReg],cI2C_Buffer[cMagField_Y_LSBReg],iCompas
s_MagField_Temp,iCompass_MagField_Temp);}
        // Convert Int (Int32) to Int16
        _iSelfTest_YAxis_Average +=abs((short)iCompass_MagField_Temp);

        // Get Z-Axis Component of Magnetic Field
        iCompass_MagField_Temp=0;
        dCompass_MagField_Temp=0.0f;
        iCompass_MagField_Temp = cI2C_Buffer[cMagField_Z_MSBReg];
        iCompass_MagField_Temp = (iCompass_MagField_Temp<<8) |
cI2C_Buffer[cMagField_Z_LSBReg];
        // Check For Two's Complement Conversion

iCompass_MagField_Temp=iCompass_TwoComplement(iOpt_TwoComplement,iCompass_Mag
Field_Temp);
        // Output MBS and LSB
        if (iDebug_Level >=Debug_Level_0){printf("Compass Msg: Z Field> MSB:
%#04x ; LSB: %#04x Count: %#06x Value:
%i\n",cI2C_Buffer[cMagField_Z_MSBReg],cI2C_Buffer[cMagField_Z_LSBReg],iCompas
s_MagField_Temp,iCompass_MagField_Temp);}
        // Convert Int (Int32) to Int16
        _iSelfTest_ZAxis_Average +=abs((short)iCompass_MagField_Temp);

        // Time For Data Recapture
        sleep(0.67);
        // End Loop
    }

    // Set Compass Out of Self-Test Mode
    cCompass_ConfigRegA_Val = cCompass_SelfTest_Exit;
    // Verify Registry A Data
    if (iDebug_Level >=Debug_Level_3) {printf("Compass Msg (Self Test):
Composite Registry A Data : %#04x\n",cCompass_ConfigRegA_Val);}
    // Set Compass Data Output Average, Data Output Rate and Compass
Measurement Mode

iI2C_Write=iCompass_I2CWriteBuff(cCompass_ConfigAReg,cCompass_ConfigRegA_Val)
;

    // Calculate Averages

_iSelfTest_XAxis_Average=_iSelfTest_XAxis_Average/_iSelfTest_Average_Count;

_iSelfTest_YAxis_Average=_iSelfTest_YAxis_Average/_iSelfTest_Average_Count;

_iSelfTest_ZAxis_Average=_iSelfTest_ZAxis_Average/_iSelfTest_Average_Count;

    // Calculate Expected Min / Max for Selected Self Test Gain
```

```
    _iSelfTest_LimMax =
cCompass_SelfTest_LimMax*cCompass_GetGainVal(cCompass_SetGain(prmCompass_Self
Test_GainValue[0]))/cCompass_GetGainVal(cCompass_Gain_390);
    if (iDebug_Level >=Debug_Level_3){printf("Compass Msg (Self Text): Max
Lim=%i * %i /
%i\n",cCompass_SelfTest_LimMax,cCompass_GetGainVal(cCompass_SetGain(prmCompas
s_SelfTest_GainValue[0])),cCompass_GetGainVal(cCompass_Gain_390));}

    // Calculate Expected Min / Max for Selected Self Test Gain
    _iSelfTest_LimMin =
cCompass_SelfTest_LimMin*cCompass_GetGainVal(cCompass_SetGain(prmCompass_Self
Test_GainValue[0]))/cCompass_GetGainVal(cCompass_Gain_390);
    if (iDebug_Level >=Debug_Level_3){printf("Compass Msg (Self Text): Max
Lim=%i * %i /
%i\n",cCompass_SelfTest_LimMin,cCompass_GetGainVal(cCompass_SetGain(prmCompas
s_SelfTest_GainValue[0])),cCompass_GetGainVal(cCompass_Gain_390));}

    // Verify if Self Test Within Limit for X Axis
    if ((_iSelfTest_XAxis_Average>_iSelfTest_LimMin) &&
(_iSelfTest_XAxis_Average<_iSelfTest_LimMax))
    {
        _iSelfTest_XAxis_Passed=1;
        if (iDebug_Level >=Debug_Level_0){printf("Compass Msg (Self Text) X-
Axis Passed> Min: %i X-Value: %i Max:
%i\n",_iSelfTest_LimMin,_iSelfTest_XAxis_Average,_iSelfTest_LimMax);}
    }
    else
    {
        _iSelfTest_XAxis_Passed=0;
        if (iDebug_Level >=Debug_Level_0){printf("Compass Msg (Self Text) X-
Axis Failed> Min: %i X-Value: %i Max:
%i\n",_iSelfTest_LimMin,_iSelfTest_XAxis_Average,_iSelfTest_LimMax);}
        // Set Error
        iError_No=Error_Compass_No_SelfTest;
        // Set Error Level
        iError_Level = Error_Level_Critical;
    }

    // Verify if Self Test Within Limit for Y Axis
    if ((_iSelfTest_YAxis_Average>_iSelfTest_LimMin) &&
(_iSelfTest_YAxis_Average<_iSelfTest_LimMax))
    {
        _iSelfTest_YAxis_Passed=1;
        if (iDebug_Level >=Debug_Level_0){printf("Compass Msg (Self Text) Y-
Axis Passed> Min: %i Y-Value: %i Max:
%i\n",_iSelfTest_LimMin,_iSelfTest_YAxis_Average,_iSelfTest_LimMax);}

    }
    else
    {
        _iSelfTest_YAxis_Passed=0;
        if (iDebug_Level >=Debug_Level_0){printf("Compass Msg (Self Text) Y-
Axis Failed> Min: %i Y-Value: %i Max:
%i\n",_iSelfTest_LimMin,_iSelfTest_YAxis_Average,_iSelfTest_LimMax);}
        // Set Error
        iError_No=Error_Compass_No_SelfTest;
```

```
        // Set Error Level
        iError_Level = Error_Level_Critical;
    }

    // Verify if Self Test Within Limit for Z Axis
    if ((_iSelfTest_ZAxis_Average>_iSelfTest_LimMin) &&
(_iSelfTest_ZAxis_Average<_iSelfTest_LimMax))
    {
        _iSelfTest_ZAxis_Passed=1;
        if (iDebug_Level >=Debug_Level_0){printf("Compass Msg (Self Text) Z-
Axis Passed> Min: %i Z-Value: %i Max:
%i\n",_iSelfTest_LimMin,_iSelfTest_ZAxis_Average,_iSelfTest_LimMax);}
    }
    else
    {
        _iSelfTest_ZAxis_Passed=0;
        if (iDebug_Level >=Debug_Level_0){printf("Compass Msg (Self Text) Z-
Axis Failed> Min: %i Z-Value: %i Max:
%i\n",_iSelfTest_LimMin,_iSelfTest_ZAxis_Average,_iSelfTest_LimMax);}
        // Set Error
        iError_No=Error_Compass_No_SelfTest;
        // Set Error Level
        iError_Level = Error_Level_Critical;
    }


    // Output Info - Self Test Complete
    if (iError_No == Error_None)
    {
        // Ouput Program Flow
        if (iDebug_Level >=Debug_Level_3) {printf("Compass Msg (Self Test):
Program Flow State: Self Test Completed \nError No: %i Error Level:
%i\n",iError_No, iError_Level);}
    }
    else
    {
        // Ouput Program Flow
        if (iDebug_Level >=Debug_Level_3) {printf("Compass Msg (Self Test):
Program Flow State: Self Test Failed \nError No: %i Error Level:
%i\n",iError_No, iError_Level);}
    }
}


// ************************************//
////// MAIN PROGRAM START ///////
// ************************************//


//sleep(1); // Sleep Time in Seconds

// Manually Set Parameter Data - Development
/*
prmI2C_DeviceNo[0]=1;
prmI2C_BusNo[0]=1;
```

```
prmCompass_DataOutRate[0]=7;
prmCompass_DataOutAvrg[0]=1;
prmCompass_MeasureMode[0]=1;
prmCompass_GainValue[0]=2;
prmCompass_OpMode[0]=1;
prmCompass_DeclinationAng[0]=1;
prmDebug_InfoLevel[0]=7;
*/

// Read Parameter Data -Device Number
switch(prmI2C_DeviceNo[0])
{
    case  1:
        cCompass_I2C_Addr=cCompass_I2C_Addr1;
        break;
    default:
        cCompass_I2C_Addr=0x00;
        break;
}

// Read Parameter Data -Device Number - Check for Functions in Later Version
iDebug_Level = iSet_Debug_Level(prmDebug_InfoLevel[0]);

// Start Line
if (iDebug_Level >=Debug_Level_0){printf("**** Start of Compass S-Function
Block Execution **** \n");}

// Check Parameters
if (iDebug_Level >=Debug_Level_1){printf("Compass Msg: Input Parameter Data:
I2CBus: %#04x I2CAddress: %#04x\n",prmI2C_BusNo[0],cCompass_I2C_Addr);}

// Ouput Program Flow
if (iDebug_Level >=Debug_Level_3) {printf("Compass Msg: Program Flow State 1:
Parameters Initialized \nError No: %i Error Level: %i\n",iError_No,
iError_Level);}

// Run Init Routines
if (xD[0]==0)
{
    //  Run Self Test
    if (prmCompass_SelfTest=true)
    {
       vCompass_SelfTest();
    }

    // Initialize Compass
    if (iError_No == Error_None)
    {
        vCompass_Initialize();
    }
}

// Good To Go! Read Device Buffer
if (iError_No == Error_None)
{
```

```
    // Read Compass Buffer
    iI2C_Read=iCompass_I2CReadBuff();
}

// Process Compass Data
if (iError_No == Error_None)
{
    // Read Data Status Registry
    cI2C_StatReg_Temp=cI2C_Buffer[cCompass_StatusReg];
    if (iDebug_Level >=Debug_Level_2){printf("Compass Diagnostics: STATUS_REG
0x00: %#04x \n",cI2C_Buffer[cCompass_StatusReg]);}

    // Check For Overwritten Data
    if (cI2C_StatReg_Temp &  0b00000001)
    {
        iI2C_STATUS_REG_DATARDY=1;
    }
    // Output
    if (iDebug_Level >=Debug_Level_2) {printf("Compass Msg: Data Available:
%i \n",iI2C_STATUS_REG_DATARDY);}

    // Read Compass Data If Data Ready
    if (iI2C_STATUS_REG_DATARDY==1)
    {
        // Extract Raw Data and Calculate Heading

        // Get X-Axis Component of Magnetic Field
        iCompass_MagField_Temp=0;
        dCompass_MagField_Temp=0.0f;
        iCompass_MagField_Temp = cI2C_Buffer[cMagField_X_MSBReg];
        iCompass_MagField_Temp = (iCompass_MagField_Temp<<8) |
cI2C_Buffer[cMagField_X_LSBReg];
        // Check For Two's Complement Conversion

iCompass_MagField_Temp=iCompass_TwoComplement(iOpt_TwoComplement,iCompass_Mag
Field_Temp);
        // Output MBS and LSB
        if (iDebug_Level >=Debug_Level_0){printf("Compass Msg: X Field> MSB:
%#04x ; LSB: %#04x Count: %#06x Value:
%i\n",cI2C_Buffer[cMagField_X_MSBReg],cI2C_Buffer[cMagField_X_LSBReg],iCompas
s_MagField_Temp,iCompass_MagField_Temp);}
        // Convert Int (Int32) to Int16
        iCompass_MagField_Temp=(short)iCompass_MagField_Temp;
        //
        if (iDebug_Level >=Debug_Level_0){printf("Compass Msg: X Field> MSB:
%#04x ; LSB: %#04x Count: %#06x Short Value:
%i\n",cI2C_Buffer[cMagField_X_MSBReg],cI2C_Buffer[cMagField_X_LSBReg],iCompas
s_MagField_Temp,iCompass_MagField_Temp);}
         // Convert Int16 to Double - Output Data in Micro Teslas - Calc Op 1
        dCompass_MagField_Temp =
(double)iCompass_MagField_Temp/cCompass_GetGainVal(cCompass_SetGain(prmCompas
s_GainValue[0]));
         // Convert Int16 to Double - Output Data in Micro Teslas - Calc Op 2
        dCompass_MagField_Temp =
(double)iCompass_Gauss_uTesla_Conv*dCompass_MagField_Temp;
       // Set Output!
```

```
        outMagField_HX[0]=dCompass_MagField_Temp;
        // Check Calculation
        if (iDebug_Level >=Debug_Level_0){printf("Compass Msg: Calc:
ConvFact*Field/Gain =
%i*%i/%i=%.2f\n",iCompass_Gauss_uTesla_Conv,iCompass_MagField_Temp,cCompass_G
etGainVal(cCompass_SetGain(prmCompass_GainValue[0])),outMagField_HX[0]);}

        // Get Y-Axis Component of Magnetic Field
        iCompass_MagField_Temp=0;
        dCompass_MagField_Temp=0.0f;
        iCompass_MagField_Temp = cI2C_Buffer[cMagField_Y_MSBReg];
        iCompass_MagField_Temp = (iCompass_MagField_Temp<<8) |
cI2C_Buffer[cMagField_Y_LSBReg];
         // Check For Two's Complement Conversion

iCompass_MagField_Temp=iCompass_TwoComplement(iOpt_TwoComplement,iCompass_Mag
Field_Temp);
        // Output MBS and LSBcom
        if (iDebug_Level >=Debug_Level_0){printf("Compass Msg: Y Field> MSB:
%#04x ; LSB: %#04x Count: %#06x Value:
%i\n",cI2C_Buffer[cMagField_Y_MSBReg],cI2C_Buffer[cMagField_Y_LSBReg],iCompas
s_MagField_Temp,iCompass_MagField_Temp);}
        // Convert Int (Int32) to Int16
        iCompass_MagField_Temp = (short)iCompass_MagField_Temp;
        //
        if (iDebug_Level >=Debug_Level_0){printf("Compass Msg: Y Field> MSB:
%#04x ; LSB: %#04x Count: %#06x Short Value:
%i\n",cI2C_Buffer[cMagField_Y_MSBReg],cI2C_Buffer[cMagField_Y_LSBReg],iCompas
s_MagField_Temp,iCompass_MagField_Temp);}
         // Convert Int16 to Double - Output Data in Micro Teslas - Calc Op 1
        dCompass_MagField_Temp =
(double)iCompass_MagField_Temp/cCompass_GetGainVal(cCompass_SetGain(prmCompas
s_GainValue[0]));
         // Convert Int16 to Double - Output Data in Micro Teslas - Calc Op 2
        dCompass_MagField_Temp =
(double)iCompass_Gauss_uTesla_Conv*dCompass_MagField_Temp;
        // Set Output!
        outMagField_HY[0]=dCompass_MagField_Temp;
        // Check Calculation
        if (iDebug_Level >=Debug_Level_0){printf("Compass Msg: Calc:
ConvFact*Field/Gain =
%i*%i/%i=%.2f\n",iCompass_Gauss_uTesla_Conv,iCompass_MagField_Temp,cCompass_G
etGainVal(cCompass_SetGain(prmCompass_GainValue[0])),outMagField_HY[0]);}

        // Get Z-Axis Component of Magnetic Field
        iCompass_MagField_Temp=0;
        dCompass_MagField_Temp=0.0f;
        iCompass_MagField_Temp = cI2C_Buffer[cMagField_Z_MSBReg];
        iCompass_MagField_Temp = (iCompass_MagField_Temp<<8) |
cI2C_Buffer[cMagField_Z_LSBReg];
         // Check For Two's Complement Conversion

iCompass_MagField_Temp=iCompass_TwoComplement(iOpt_TwoComplement,iCompass_Mag
Field_Temp);
        // Output MBS and LSB
```

```
        if (iDebug_Level >=Debug_Level_0){printf("Compass Msg: Z Field> MSB:
%#04x ; LSB: %#04x Count: %#06x Value:
%i\n",cI2C_Buffer[cMagField_Z_MSBReg],cI2C_Buffer[cMagField_Z_LSBReg],iCompas
s_MagField_Temp,iCompass_MagField_Temp);}
        // Convert Int (Int32) to Int16
        iCompass_MagField_Temp = (short)iCompass_MagField_Temp;
        if (iDebug_Level >=Debug_Level_0){printf("Compass Msg: Z Field> MSB:
%#04x ; LSB: %#04x Count: %#06x Short Value:
%i\n",cI2C_Buffer[cMagField_Z_MSBReg],cI2C_Buffer[cMagField_Z_LSBReg],iCompas
s_MagField_Temp,iCompass_MagField_Temp);}
         // Convert Int16 to Double - Output Data in Micro Teslas - Calc Op 1
        dCompass_MagField_Temp =
(double)iCompass_MagField_Temp/cCompass_GetGainVal(cCompass_SetGain(prmCompas
s_GainValue[0]));
         // Convert Int16 to Double - Output Data in Micro Teslas - Calc Op 2
        dCompass_MagField_Temp =
(double)iCompass_Gauss_uTesla_Conv*dCompass_MagField_Temp;
        // Set Output!
        outMagField_HZ[0]=dCompass_MagField_Temp;
        // Check Calculation
        if (iDebug_Level >=Debug_Level_0){printf("Compass Msg: Calc:
ConvFact*Field/Gain =
%i*%i/%i=%.2f\n",iCompass_Gauss_uTesla_Conv,iCompass_MagField_Temp,cCompass_G
etGainVal(cCompass_SetGain(prmCompass_GainValue[0])),outMagField_HZ[0]);}

        // Calculate Heading (Non Compensated for Roll/Pitch) wrt Magnetic
North

outMagField_Heading[0]=atan2(outMagField_HY[0],outMagField_HX[0])*180.0f/M_PI
;

        // Output Heading Depending on Mode Selected
        switch(prmCompass_HeadingMode[0])
        {
            case  1:
                 // Calculate Heading (Non Compensated for Roll/Pitch) wrt
Magnetic North
                 // Do Nothing;
                break;
            case  2:
                // Calculate Heading (Non Compensated for Roll/Pitch) wrt
True North
                outMagField_Heading[0] = outMagField_Heading[0] +
(double)prmCompass_DeclinationAng[0];
                break;
            default:
                // Do Nothing;
                break;
        }

        // Adjust for Any Interference/Unexplained Offset Error
        outMagField_Heading[0] += prmCompass_HeadingOffset[0];

        // Check For Negative Heading
        if (outMagField_Heading[0]<0.0f)
        {
```

```
            outMagField_Heading[0] = outMagField_Heading[0]+360.0f;
        }

        // Calcualte Magnetic Field Strength (Micro Teslas) - Note: Montreal
= 54.1267
        outMagField_HE[0] =
sqrt(powf(outMagField_HX[0],2.0f)+powf(outMagField_HY[0],2.0f)+powf(outMagFie
ld_HZ[0],2.0f));

        // Ouput Program Flow
        if (iDebug_Level >=Debug_Level_3) {printf("Compass Msg: Program Flow
State 5: Device Data Read\nError No: %i Error Level: %i\n",iError_No,
iError_Level);}

        // Print Output - Magnetic Fields
        if (iDebug_Level >=Debug_Level_0){printf("Compass Msg: Magnetic Field
(uTelsas) X: %.1f Y: %.1f Z: %.1f Magnitude:
%.1f\n",outMagField_HX[0],outMagField_HY[0],outMagField_HZ[0],outMagField_HE[
0]);}
        // Print Output - Heading
        if (iDebug_Level >=Debug_Level_0){printf("Compass Msg: Heading
(degrees) : %f \n",outMagField_Heading[0]);}
    }
    else
    {
        if (iDebug_Level >=Debug_Level_0){printf("Compass Msg: No New Data
Ready\n");}
        // Update Data Skip Count
        iLoop_DataNotReady_Counter=xD[1];
        iLoop_DataNotReady_Counter +=1;
    }

    // Output Data Skip Count
    if (iDebug_Level >=Debug_Level_3) {printf("Compass Msg: Data Skip Count:
%i\n",iLoop_DataNotReady_Counter);}
}
// Ouput Program Flow
if (iDebug_Level >=Debug_Level_3) {printf("Compass Msg: Program Flow State 6:
Loop Step Complete\nError No: %i Error Level: %i\n",iError_No,
iError_Level);}

// Check For Critical Errors and Stop Simulation
if ((iError_No!=0)&&(iError_Level==Error_Level_Critical))
{
    // Stop Simulation If Critical Error
    if (iDebug_Level >=Debug_Level_0) {printf("Compass Msg: Critical Error
Detected; Simulation Stopping! \nError No: %i Error Level: %i\n",iError_No,
iError_Level);}

    // Output Error Message:
    if (iDebug_Level >=Debug_Level_0) {printf("Compass Error Details:
%s",sCompass_ErrMessage(iError_No));}

    // Stop Simulation
    outSimStop[0]=true;
```

```
}
else
{
    // Carry on
    outSimStop[0]=false;
}

// Print Gap
if (iDebug_Level >=Debug_Level_0){printf("\n");}
# else
# endif
/* %%%-SFUNWIZ_wrapper_Outputs_Changes_END --- EDIT HERE TO _BEGIN */
}

/*
  * Updates function
  *
  */
void BBB_Driver_Compass_HMC5883L_Update_wrapper(const real_T *outMagField_HX,
                          const real_T *outMagField_HY,
                          const real_T *outMagField_HZ,
                          const real_T *outMagField_HE,
                          const real_T *outMagField_Heading,
                          const real_T *outSimStop ,
                          real_T *xD,
                          const uint8_T  *prmI2C_DeviceNo,  const int_T
p_width0,
                          const uint8_T  *prmI2C_BusNo,  const int_T
p_width1,
                          const uint8_T  *prmCompass_DataOutAvrg,  const
int_T  p_width2,
                          const uint8_T  *prmCompass_DataOutRate,  const
int_T  p_width3,
                          const uint8_T  *prmCompass_MeasureMode,  const
int_T  p_width4,
                          const uint8_T  *prmCompass_GainValue,  const int_T
p_width5,
                          const uint8_T  *prmCompass_OpMode,  const int_T
p_width6,
                          const uint8_T  *prmCompass_HeadingMode,  const
int_T  p_width7,
                          const real_T  *prmCompass_HeadingOffset,  const
int_T  p_width8,
                          const real_T  *prmCompass_DeclinationAng,  const
int_T  p_width9,
                          const boolean_T  *prmCompass_SelfTest,  const int_T
p_width10,
                          const uint8_T  *prmCompass_SelfTest_GainValue,
const int_T  p_width11,
                           const uint8_T *prmDebug_InfoLevel, const int_T
p_width12)
{
  /* %%%-SFUNWIZ_wrapper_Update_Changes_BEGIN --- EDIT HERE TO _END */
// Run Only on Target - BeagleBoneBlack */
# ifndef MATLAB_MEX_FILE
```

```
// Update Init Bit
if (xD[0]==0)
{
    // Set Init Complete
    xD[0]=1;

}

// Update Register Locked / No New Data Counter
xD[1]=iLoop_DataNotReady_Counter;

# else

// Do Nothing


# endif
/* %%%-SFUNWIZ_wrapper_Update_Changes_END --- EDIT HERE TO _BEGIN */
}
```

13.10 S-FUNCTION BLOCK WRAPPER CODE: GRADIENT DESCENT IMU ATTITUDE ESTIMATION BLOCK

```
 /*
  *
  *   --- THIS FILE GENERATED BY S-FUNCTION BUILDER: 3.0 ---
  *
  *   This file is a wrapper S-function produced by the S-Function
  *   Builder which only recognizes certain fields.  Changes made
  *   outside these fields will be lost the next time the block is
  *   used to load, edit, and resave this file. This file will be overwritten
  *   by the S-function Builder block. If you want to edit this file by hand,
  *   you must change it only in the area defined as:
  *
  *        %%%-SFUNWIZ_wrapper_XXXXX_Changes_BEGIN
  *            Your Changes go here
  *        %%%-SFUNWIZ_wrapper_XXXXXX_Changes_END
  *
  *   For better compatibility with the Simulink Coder, the
  *   "wrapper" S-function technique is used.  This is discussed
  *   in the Simulink Coder User's Manual in the Chapter titled,
  *   "Wrapper S-functions".
  *
  *   Created: Sun Dec 14 04:17:25 2014
  */

/*
 * Include Files
 *
 */
#if defined(MATLAB_MEX_FILE)
#include "tmwtypes.h"
#include "simstruc_types.h"
#else
#include "rtwtypes.h"
#endif

/* %%%-SFUNWIZ_wrapper_includes_Changes_BEGIN --- EDIT HERE TO _END */
#include <math.h>

typedef enum
{
        Debug_None,
    Debug_Level_0, // Basic Debug Info Output
        Debug_Level_1, // + Critical Info Only
    Debug_Level_2, // + Diagnostics Info
    Debug_Level_3, // + Program Flow Info
    Debug_Level_4, // + Registry Data
    Debug_Level_5  // + TBD
}eDebugLevel;


// Initialize Error
// eError_No iError_No = Error_None;

// Define Error Severity Levels
typedef enum
```

```
{
    Error_Level_OK=0,
    Error_Level_Critical, // (Stop Simulation)
    Error_Level_Warning
}eError_Level;
/* %%%-SFUNWIZ_wrapper_includes_Changes_END --- EDIT HERE TO _BEGIN */
#define u_width 1
#define y_width 1
/*
 * Create external references here.
 *
 */
/* %%%-SFUNWIZ_wrapper_externs_Changes_BEGIN --- EDIT HERE TO _END */
/* extern double func(double a); */
/* %%%-SFUNWIZ_wrapper_externs_Changes_END --- EDIT HERE TO _BEGIN */

/*
 * Output functions
 *
 */
void BBB_AHRS_IMU_v1_Outputs_wrapper(const real_T *inGyro_w_x,
                           const real_T *inGyro_w_y,
                           const real_T *inGyro_w_z,
                           const real_T *inAccel_a_x,
                           const real_T *inAccel_a_y,
                           const real_T *inAccel_a_z,
                           real_T *outQuat_q_1,
                           real_T *outQuat_q_2,
                           real_T *outQuat_q_3,
                           real_T *outQuat_q_4,
                           boolean_T *outStopSim,
                           const real_T  *xD)
{
/* %%%-SFUNWIZ_wrapper_Outputs_Changes_BEGIN --- EDIT HERE TO _END */
# ifndef MATLAB_MEX_FILE

// Initialize Estimated Quaternions
double dEst_q_1=0.0f;
double dEst_q_2=0.0f;
double dEst_q_3=0.0f;
double dEst_q_4=0.0f;
double dEst_q_Norm=0.0f;

double dEst_S_E_q_1_dot=0;
double dEst_S_E_q_2_dot=0;
double dEst_S_E_q_3_dot=0;
double dEst_S_E_q_4_dot=0;

// Declare and Initialize Gyro and Accel Data Temp Variables
double dGyro_w_x=0.0f,dGyro_w_y=0.0f,dGyro_w_z=0.0f;
double dAccel_a_x=0.0f,dAccel_a_y=0.0f,dAccel_a_z=0.0f;
double
dAccel_Norm=0.0f,dAccel_a_Norm_x=0.0f,dAccel_a_Norm_y=0.0f,dAccel_a_Norm_z=0.
0f;

// Declare and Initialize Quaternion Angular Rate Variables
```

```
double dS_E_q_w_1_dot=0;
double dS_E_q_w_2_dot=0;
double dS_E_q_w_3_dot=0;
double dS_E_q_w_4_dot=0;

// Declare Objective Function Data
double dObjective_f_11=0,dObjective_f_21=0,dObjective_f_31=0;
double
dObjective_J_11=0,dObjective_J_12=0,dObjective_J_13=0,dObjective_J_14=0;
double
dObjective_J_21=0,dObjective_J_22=0,dObjective_J_23=0,dObjective_J_24=0;
double
dObjective_J_31=0,dObjective_J_32=0,dObjective_J_33=0,dObjective_J_34=0;
double
dObjective_J_41=0,dObjective_J_42=0,dObjective_J_43=0,dObjective_J_44=0;
double
dObjective_Grad_f_11=0.0f,dObjective_Grad_f_21=0.0f,dObjective_Grad_f_31=0.0f
,dObjective_Grad_f_41=0.0f;
double dObjective_Grad_f_Norm=0.0f;

// Define Constants
#define dSample_Period_Delta_t 0.02f
#define dGain_Beta 0.1f // Divergence Rate of S_E_q_w
#define dGain_Alpha 0.1f // Convergence Reate of S_E_q_w_Grad

// Define Error Number
typedef enum
{
    Error_None=0,
    Error_No_SYSFSOpen,
    Error_No_SYSFSRead,
    Error_No_SYSFSWrite,
    Error_No_ParamOutofRange
}eError_No;

// Initialize Error
eError_No iError_No = Error_None;

// Initialize Debug Level
eDebugLevel iDebug_Level = Debug_None;


/*************************************************************************/
/*************************************************************************/
/*************************************************************************/
// Main Program Start
/*************************************************************************/
/*************************************************************************/
/*************************************************************************/

// Read Input Data To Local Variables
dGyro_w_x=inGyro_w_x[0];
dGyro_w_y=inGyro_w_y[0];
dGyro_w_z=inGyro_w_z[0];
dAccel_a_x=inAccel_a_x[0];
```

```
dAccel_a_y=inAccel_a_y[0];
dAccel_a_z=inAccel_a_z[0];


// Initialize Quaternion Data During First Scan
if (xD[0]==0)
{
   // Set Initial Quaternion Values - Senor Frame Aligned with Earth Frame
    dEst_q_1=1.0f;
    dEst_q_2=0.0f;
    dEst_q_3=0.0f;
    dEst_q_4=0.0f;
    // Output Estimated Quaternion Data
   if (iDebug_Level >=Debug_Level_3) { printf("Quaternion Data Init> Q1: %.2f
Q2: %.2f Q3: %.2f Q4: %.2f\n",dEst_q_1,dEst_q_2,dEst_q_3,dEst_q_4);}
}
else if (xD[0]==1)
{
    // Populate Quaternion Local Variables With Saved Persistent Data From
    // Previous Timestep (t-1)
    dEst_q_1=xD[1];
    dEst_q_2=xD[2];
    dEst_q_3=xD[3];
    dEst_q_4=xD[4];
    // Output Estimated Quaternion Data
    if (iDebug_Level >=Debug_Level_3) { printf("Quaternion Data Update> Q1:
%.2f Q2: %.2f Q3: %.2f Q4: %.2f\n",dEst_q_1,dEst_q_2,dEst_q_3,dEst_q_4);}

}

// Output Estimated Quaternion Data
//if (iDebug_Level >=Debug_Level_3) { printf("Quaternion Data Update> Q1:
%.2f Q2: %.2f Q3: %.2f Q4: %.2f %s\n",dEst_q_1,dEst_q_2,dEst_q_3,dEst_q_4);}


// Read Input Data into Temporary Varialbes

/*************************************************************************/
// Step 1: Filter > Get Attitude from Angular Rate (Gyros)
/*************************************************************************/
// Step 1.1: > Calculate Quaternion Derivative for Rate of Change of
// Orientation of Earth Frame Relative To Sensor Frame
dS_E_q_w_1_dot=0.5f*(-dEst_q_2*dGyro_w_x - dEst_q_3*dGyro_w_y -
dEst_q_4*dGyro_w_z);
dS_E_q_w_2_dot=0.5f*( dEst_q_1*dGyro_w_x + dEst_q_3*dGyro_w_z -
dEst_q_4*dGyro_w_y);
dS_E_q_w_3_dot=0.5f*( dEst_q_1*dGyro_w_y - dEst_q_2*dGyro_w_z +
dEst_q_4*dGyro_w_x);
dS_E_q_w_4_dot=0.5f*(-dEst_q_1*dGyro_w_z + dEst_q_2*dGyro_w_y -
dEst_q_3*dGyro_w_x);

// Step 1.2: > Carry out Numerical Integration of Quaternion Derivative
// Note: Step 1.2 Not required as "alpha" constant - Constant used to
// augment "mu" (stepsize) - is assumed >>> 0.
```

```
/*********************************************************************/
// Step 2: Calculate Gradient of the Objective Function "f"
// Which Descibes The Optimization Problem.
// Note: The optimization problem aims to minimize the difference between
// a given reference direction of the field (gravity) in the earth field
// and the measured direction of the field in sensor frame via a rotational
// transform operation.
/*********************************************************************/

// Step 2.1 > Normalize Accelerometer Measurements
dAccel_Norm =
sqrt(dAccel_a_x*dAccel_a_x+dAccel_a_y*dAccel_a_y+dAccel_a_z*dAccel_a_z);
dAccel_a_Norm_x=dAccel_a_x/dAccel_Norm;
dAccel_a_Norm_y=dAccel_a_y/dAccel_Norm;
dAccel_a_Norm_z=dAccel_a_z/dAccel_Norm;

// Step 2.2 > Calculate Objective Function Array "f"
dObjective_f_11=(2.0f*(dEst_q_2*dEst_q_4-dEst_q_1*dEst_q_3)-dAccel_a_Norm_x);
dObjective_f_21=(2.0f*(dEst_q_1*dEst_q_2+dEst_q_3*dEst_q_4)-dAccel_a_Norm_y);
dObjective_f_31=(2.0f*(0.5f-dEst_q_2*dEst_q_2-dEst_q_3*dEst_q_3)-
dAccel_a_Norm_z);

// Step 2.3 > Calculate Jacobian Matrix of Objective Function
dObjective_J_11=-2.0f*dEst_q_3;
dObjective_J_12= 2.0f*dEst_q_4;
dObjective_J_13=-2.0f*dEst_q_1;
dObjective_J_14= 2.0f*dEst_q_2;
dObjective_J_21= 2.0f*dEst_q_2;
dObjective_J_22= 2.0f*dEst_q_1;
dObjective_J_23= 2.0f*dEst_q_4;
dObjective_J_24= 2.0f*dEst_q_3;
dObjective_J_31= 0.0f;
dObjective_J_32=-4.0f*dEst_q_2;
dObjective_J_33=-4.0f*dEst_q_3;
dObjective_J_34= 0.0f;

// Step 2.3 > Calculate Gradient of Objective Function
dObjective_Grad_f_11=dObjective_J_11*dObjective_f_11 +
dObjective_J_21*dObjective_f_21 + dObjective_J_31*dObjective_f_31;
dObjective_Grad_f_21=dObjective_J_12*dObjective_f_11 +
dObjective_J_22*dObjective_f_21 + dObjective_J_32*dObjective_f_31;
dObjective_Grad_f_31=dObjective_J_13*dObjective_f_11 +
dObjective_J_23*dObjective_f_21 + dObjective_J_33*dObjective_f_31;
dObjective_Grad_f_41=dObjective_J_14*dObjective_f_11 +
dObjective_J_24*dObjective_f_21 + dObjective_J_34*dObjective_f_31;

// Step 2.4 > Calculate Gradient Norm
dObjective_Grad_f_Norm=sqrt(dObjective_Grad_f_11*dObjective_Grad_f_11+dObject
ive_Grad_f_21*dObjective_Grad_f_21+dObjective_Grad_f_31*dObjective_Grad_f_31+
dObjective_Grad_f_41*dObjective_Grad_f_41);

/*********************************************************************/
// Step 3: Fusion of Accel and Gyro Sensor Data To Generate Increment
Estimate
```

```
/*****************************************************************************/
dEst_S_E_q_1_dot=(dS_E_q_w_1_dot -
dGain_Beta*dObjective_Grad_f_11/dObjective_Grad_f_Norm);
dEst_S_E_q_2_dot=(dS_E_q_w_2_dot -
dGain_Beta*dObjective_Grad_f_21/dObjective_Grad_f_Norm);
dEst_S_E_q_3_dot=(dS_E_q_w_3_dot -
dGain_Beta*dObjective_Grad_f_31/dObjective_Grad_f_Norm);
dEst_S_E_q_4_dot=(dS_E_q_w_4_dot -
dGain_Beta*dObjective_Grad_f_41/dObjective_Grad_f_Norm);

/*****************************************************************************/
// Step 4: Numerical Integration of Quaternion Rate Data
/*****************************************************************************/

// Step 4.1 > Carry Out Numerical Integration

dEst_q_1=dEst_q_1+dEst_S_E_q_1_dot*dSample_Period_Delta_t;
dEst_q_2=dEst_q_2+dEst_S_E_q_2_dot*dSample_Period_Delta_t;
dEst_q_3=dEst_q_3+dEst_S_E_q_3_dot*dSample_Period_Delta_t;
dEst_q_4=dEst_q_4+dEst_S_E_q_4_dot*dSample_Period_Delta_t;

// Step 4.2 > Calculate Quaternion Norm
dEst_q_Norm = sqrt(dEst_q_1*dEst_q_1 + dEst_q_2*dEst_q_2 +dEst_q_3*dEst_q_3
+dEst_q_4*dEst_q_4);

/*****************************************************************************/
// Step 5: Normalize and Output Quaternion Data
/*****************************************************************************/
dEst_q_1 = dEst_q_1/dEst_q_Norm;
dEst_q_2 = dEst_q_2/dEst_q_Norm;
dEst_q_3 = dEst_q_3/dEst_q_Norm;
dEst_q_4 = dEst_q_4/dEst_q_Norm;

/*****************************************************************************/
// Step 6: Switch Base Reference Frame If Needed
// Note: Represent quarternions as sensor attitude relative to fixed earth
// i.e d_Est_E_S_q_X where X = 1,2,3,4
/*****************************************************************************/
/*
outQuat_q_1[0]=dEst_q_1;
outQuat_q_2[0]=-dEst_q_2;
outQuat_q_3[0]=-dEst_q_3;
outQuat_q_4[0]=-dEst_q_4;
*/
outQuat_q_1[0]=dEst_q_1;
outQuat_q_2[0]=dEst_q_2;
outQuat_q_3[0]=dEst_q_3;
outQuat_q_4[0]=dEst_q_4;

// Output Quaternion Data
if (iDebug_Level >=Debug_Level_1) { printf("Quaternion Data> Q1: %.2f Q2:
%.2f Q3: %.2f Q4:
%.2f\n",outQuat_q_1[0],outQuat_q_2[0],outQuat_q_3[0],outQuat_q_4[0]);}
```

```
# endif
/* %%%-SFUNWIZ_wrapper_Outputs_Changes_END --- EDIT HERE TO _BEGIN */
}

/*
  * Updates function
  *
  */
void BBB_AHRS_IMU_v1_Update_wrapper(const real_T *inGyro_w_x,
                           const real_T *inGyro_w_y,
                           const real_T *inGyro_w_z,
                           const real_T *inAccel_a_x,
                           const real_T *inAccel_a_y,
                           const real_T *inAccel_a_z,
                           const real_T *outQuat_q_1,
                           const real_T *outQuat_q_2,
                           const real_T *outQuat_q_3,
                           const real_T *outQuat_q_4,
                           const boolean_T *outStopSim,
                           real_T *xD)
{
  /* %%%-SFUNWIZ_wrapper_Update_Changes_BEGIN --- EDIT HERE TO _END */
/* Only Execute On Target*/
# ifndef MATLAB_MEX_FILE

// Recall Discrete State Data
// xD[0] = First Scan
// xD[1] = Estimated Quaternion dEst_q_1
// xD[2] = Estimated Quaternion dEst_q_2
// xD[3] = Estimated Quaternion dEst_q_3
// xD[4] = Estimated Quaternion dEst_q_4

if (xD[0]==0)
{
    // Initialize Quaternions

    // Set First Scan Bit
    xD[0]=1;
}

// Update Persistent Quaternion Data
xD[1]=outQuat_q_1[0];
xD[2]=outQuat_q_2[0];
xD[3]=outQuat_q_3[0];
xD[4]=outQuat_q_4[0];


# endif
/* %%%-SFUNWIZ_wrapper_Update_Changes_END --- EDIT HERE TO _BEGIN */
}
```

13.11 S-FUNCTION BLOCK WRAPPER CODE: GRADIENT DESCENT AHRS ATTITUDE ESTIMATION BLOCK

```
/*
  *
  *   --- THIS FILE GENERATED BY S-FUNCTION BUILDER: 3.0 ---
  *
  *   This file is a wrapper S-function produced by the S-Function
  *   Builder which only recognizes certain fields.  Changes made
  *   outside these fields will be lost the next time the block is
  *   used to load, edit, and resave this file. This file will be overwritten
  *   by the S-function Builder block. If you want to edit this file by hand,
  *   you must change it only in the area defined as:
  *
  *        %%%-SFUNWIZ_wrapper_XXXXX_Changes_BEGIN
  *            Your Changes go here
  *        %%%-SFUNWIZ_wrapper_XXXXXX_Changes_END
  *
  *   For better compatibility with the Simulink Coder, the
  *   "wrapper" S-function technique is used.  This is discussed
  *   in the Simulink Coder User's Manual in the Chapter titled,
  *   "Wrapper S-functions".
  *
  *   Created: Sun Dec 14 04:09:03 2014
  */


/*
 * Include Files
 *
 */
#if defined(MATLAB_MEX_FILE)
#include "tmwtypes.h"
#include "simstruc_types.h"
#else
#include "rtwtypes.h"
#endif

/* %%%-SFUNWIZ_wrapper_includes_Changes_BEGIN --- EDIT HERE TO _END */
#include <math.h>

typedef enum
{
        Debug_None,
    Debug_Level_0, // Basic Debug Info Output
    Debug_Level_1, // + Critical Info Only
    Debug_Level_2, // + Diagnostics Info
    Debug_Level_3, // + Program Flow Info
    Debug_Level_4, // + Registry Data
    Debug_Level_5  // + TBD
}eDebugLevel;


// Initialize Error
// eError_No iError_No = Error_None;
```

```
// Define Error Severity Levels
typedef enum
{
    Error_Level_OK=0,
    Error_Level_Critical, // (Stop Simulation)
    Error_Level_Warning
}eError_Level;

// Declare Flux in Earth Frame - Global Variables
double dCompass_b_x;
double dCompass_b_z;

// Declare Gyro Bias Error - Global Variables
double dGyro_w_x_Bias;
double dGyro_w_y_Bias;
double dGyro_w_z_Bias;
/* %%%-SFUNWIZ_wrapper_includes_Changes_END --- EDIT HERE TO _BEGIN */
#define u_width 1
#define y_width 1
/*
 * Create external references here.
 *
 */
/* %%%-SFUNWIZ_wrapper_externs_Changes_BEGIN --- EDIT HERE TO _END */
/* extern double func(double a); */
/* %%%-SFUNWIZ_wrapper_externs_Changes_END --- EDIT HERE TO _BEGIN */

/*
 * Output functions
 *
 */
void BBB_AHRS_IMU_MARG_v1_Outputs_wrapper(const real_T *inGyro_w_x,
                          const real_T *inGyro_w_y,
                          const real_T *inGyro_w_z,
                          const real_T *inAccel_a_x,
                          const real_T *inAccel_a_y,
                          const real_T *inAccel_a_z,
                          const real_T *inCompass_m_x,
                          const real_T *inCompass_m_y,
                          const real_T *inCompass_m_z,
                          real_T *outQuat_q_1,
                          real_T *outQuat_q_2,
                          real_T *outQuat_q_3,
                          real_T *outQuat_q_4,
                          boolean_T *outStopSim,
                          const real_T  *xD)
{
/* %%%-SFUNWIZ_wrapper_Outputs_Changes_BEGIN --- EDIT HERE TO _END */
# ifndef MATLAB_MEX_FILE

// Initialize Estimated Quaternions
double dEst_q_1=0.0f;
double dEst_q_2=0.0f;
double dEst_q_3=0.0f;
double dEst_q_4=0.0f;
```

```
double dEst_q_Norm=0.0f;

double dEst_S_E_q_1_dot=0;
double dEst_S_E_q_2_dot=0;
double dEst_S_E_q_3_dot=0;
double dEst_S_E_q_4_dot=0;

double dEst_S_E_q_1_hat_dot=0;
double dEst_S_E_q_2_hat_dot=0;
double dEst_S_E_q_3_hat_dot=0;
double dEst_S_E_q_4_hat_dot=0;

// Declare and Initialize Gyro and Accel Data Temp Variables
double dGyro_w_x=0.0f,dGyro_w_y=0.0f,dGyro_w_z=0.0f;
double dGyro_w_x_Error=0.0f,dGyro_w_y_Error=0.0f,dGyro_w_z_Error=0.0f;
// double dGyro_w_x_Bias=0.0f,dGyro_w_y_Bias=0.0f,dGyro_w_z_Bias=0.0f;
double dAccel_a_x=0.0f,dAccel_a_y=0.0f,dAccel_a_z=0.0f;
double
dAccel_Norm=0.0f,dAccel_a_Norm_x=0.0f,dAccel_a_Norm_y=0.0f,dAccel_a_Norm_z=0.
0f;
double dCompass_m_x=0.0f,dCompass_m_y=0.0f,dCompass_m_z=0.0f;
double dCompass_h_x=0.0f, dCompass_h_y=0.0f, dCompass_h_z=0.0f; // Earths
Magnetic Field in Current Orientation
double dCompass_Norm=0.0f, dCompass_m_Norm_x=0.0f, dCompass_m_Norm_y=0.0f,
dCompass_m_Norm_z=0.0f;
//double dCompass_b_x=1.0f, dCompass_b_z=0.0f; // Flux in Earth Frame -
Global

// Declare and Initialize Quaternion Angular Rate Variables
double dS_E_q_w_1_dot=0;
double dS_E_q_w_2_dot=0;
double dS_E_q_w_3_dot=0;
double dS_E_q_w_4_dot=0;


// Declare Objective Function Data
double dObjective_f_11=0,dObjective_f_21=0,dObjective_f_31=0;
double dObjective_f_41=0,dObjective_f_51=0,dObjective_f_61=0;
double
dObjective_J_11=0,dObjective_J_12=0,dObjective_J_13=0,dObjective_J_14=0;
double
dObjective_J_21=0,dObjective_J_22=0,dObjective_J_23=0,dObjective_J_24=0;
double
dObjective_J_31=0,dObjective_J_32=0,dObjective_J_33=0,dObjective_J_34=0;
double
dObjective_J_41=0,dObjective_J_42=0,dObjective_J_43=0,dObjective_J_44=0;
double
dObjective_J_51=0,dObjective_J_52=0,dObjective_J_53=0,dObjective_J_54=0;
double
dObjective_J_61=0,dObjective_J_62=0,dObjective_J_63=0,dObjective_J_64=0;
double
dObjective_Grad_f_11=0.0f,dObjective_Grad_f_21=0.0f,dObjective_Grad_f_31=0.0f
,dObjective_Grad_f_41=0.0f;
double dObjective_Grad_f_Norm=0.0f;
```

```
// Define Constants
#define M_pi 3.1415926535897931
#define dGyro_Measurement_Error M_pi*(5.0f/180.0f)
#define dGyro_Measurement_Drift M_pi*(0.2f/180.0f)
#define dSample_Period_Delta_t 0.02f
#define dGain_Beta sqrt(3.0f/4.0f)*dGyro_Measurement_Error // Filter Gain
Divergence Rate of S_E_q_w
#define dGain_Zeta sqrt(3.0f/4.0f)*dGyro_Measurement_Drift // Filter Gain
Convergence Rate of Gyro Bias Error S_E_q_wdot
#define dGain_Alpha 0.1f // Convergence Reate of S_E_q_w_Grad

//#define dGain_Beta 0.1f //sqrt(3.0f/4.0f)*dGyro_Measurement_Error // Filter
Gain Divergence Rate of S_E_q_w
//#define dGain_Zeta 0.001f //sqrt(3.0f/4.0f)*dGyro_Measurement_Drift //
Filter Gain Convergence Rate of Gyro Bias Error S_E_q_wdot
//#define dGain_Alpha 0.1f // Convergence Reate of S_E_q_w_Grad

// Define Error Number
typedef enum
{
    Error_None=0,
    Error_No_SYSFSOpen,
    Error_No_SYSFSRead,
    Error_No_SYSFSWrite,
    Error_No_ParamOutofRange
}eError_No;

// Initialize Error
eError_No iError_No = Error_None;

// Initialize Debug Level
eDebugLevel iDebug_Level = Debug_Level_5;


/*****************************************************************************/
/*****************************************************************************/
/*****************************************************************************/
// Main Program Start
/*****************************************************************************/
/*****************************************************************************/
/*****************************************************************************/

// Read Input Data To Local Variables
dGyro_w_x=inGyro_w_x[0];
dGyro_w_y=inGyro_w_y[0];
dGyro_w_z=inGyro_w_z[0];
dAccel_a_x=inAccel_a_x[0];
dAccel_a_y=inAccel_a_y[0];
dAccel_a_z=inAccel_a_z[0];
dCompass_m_x=inCompass_m_x[0];
dCompass_m_y=inCompass_m_y[0];
dCompass_m_z=inCompass_m_z[0];

// Initialize Quaternion Data During First Scan
if (xD[0]==0)
```

```
{
   // Set Initial Quaternion Values - Senor Frame Aligned with Earth Frame
    dEst_q_1=1.0f;
    dEst_q_2=0.0f;
    dEst_q_3=0.0f;
    dEst_q_4=0.0f;

    // Set Initial Values For Normalized Flux Vector
    dCompass_b_x=1.0f;
    dCompass_b_z=0.0f;

    // Populate Gyro Bias Error
    dGyro_w_x_Bias=0.0f;
    dGyro_w_y_Bias=0.0f;
    dGyro_w_z_Bias=0.0f;

    // Output Estimated Quaternion Data
   if (iDebug_Level >=Debug_Level_3) { printf("Quaternion Data Init: Q1: %.2f
Q2: %.2f Q3: %.2f Q4: %.2f\n",dEst_q_1,dEst_q_2,dEst_q_3,dEst_q_4);}
}
else if (xD[0]==1)
{
    // Populate Quaternion Local Variables With Saved Persistent Data From
    // Previous Timestep (t-1)
    dEst_q_1=xD[1];
    dEst_q_2=xD[2];
    dEst_q_3=xD[3];
    dEst_q_4=xD[4];
    //Populate Normalized Flux Vector in Earth Frame
    dCompass_b_x=xD[5];
    dCompass_b_z=xD[6];
    // Populate Gyro Bias Error
    dGyro_w_x_Bias=xD[7];
    dGyro_w_y_Bias=xD[8];
    dGyro_w_z_Bias=xD[9];
    // Output Estimated Quaternion Data
    if (iDebug_Level >=Debug_Level_3) { printf("Quaternion Data Update> Q1:
%.2f Q2: %.2f Q3: %.2f Q4: %.2f\n",dEst_q_1,dEst_q_2,dEst_q_3,dEst_q_4);}

}

// Output Estimated Quaternion Data
//if (iDebug_Level >=Debug_Level_3) { printf("Quaternion Data Update> Q1:
%.2f Q2: %.2f Q3: %.2f Q4: %.2f %s\n",dEst_q_1,dEst_q_2,dEst_q_3,dEst_q_4);}


/*********************************************************************/
// Step 1: Calculate Gradient of the Objective Function "f"
// Which Descibes The Optimization Problem.
// Note: The optimization problem aims to minimize the difference between
// a given reference direction of the field (gravity) in the earth field
// and the measured direction of the field in sensor frame via a rotational
// transform operation.
/*********************************************************************/
```

```
// Step 1.1 > Normalize Accelerometer Measurements
dAccel_Norm =
sqrt(dAccel_a_x*dAccel_a_x+dAccel_a_y*dAccel_a_y+dAccel_a_z*dAccel_a_z);
dAccel_a_Norm_x=dAccel_a_x/dAccel_Norm;
dAccel_a_Norm_y=dAccel_a_y/dAccel_Norm;
dAccel_a_Norm_z=dAccel_a_z/dAccel_Norm;

// Step 1.2 > Normalize Compass Measurements
dCompass_Norm =
sqrt(dCompass_m_x*dCompass_m_x+dCompass_m_y*dCompass_m_y+dCompass_m_z*dCompas
s_m_z);
dCompass_m_Norm_x=dCompass_m_x/dCompass_Norm;
dCompass_m_Norm_y=dCompass_m_y/dCompass_Norm;
dCompass_m_Norm_z=dCompass_m_z/dCompass_Norm;

// Step 1.3 > Calculate Objective Function Array "f"
dObjective_f_11=(2.0f*(dEst_q_2*dEst_q_4-dEst_q_1*dEst_q_3)-dAccel_a_Norm_x);
dObjective_f_21=(2.0f*(dEst_q_1*dEst_q_2+dEst_q_3*dEst_q_4)-dAccel_a_Norm_y);
dObjective_f_31=(2.0f*(0.5f-dEst_q_2*dEst_q_2-dEst_q_3*dEst_q_3)-
dAccel_a_Norm_z);
dObjective_f_41=(2.0f*dCompass_b_x*(0.5f - dEst_q_3*dEst_q_3 -
dEst_q_4*dEst_q_4)+2.0f*dCompass_b_x*(dEst_q_2*dEst_q_4-dEst_q_1*dEst_q_3)-
dCompass_m_Norm_x);
dObjective_f_51=(2.0f*dCompass_b_x*(dEst_q_2*dEst_q_3 -
dEst_q_1*dEst_q_4)+2.0f*dCompass_b_z*(dEst_q_1*dEst_q_2+dEst_q_3*dEst_q_4)-
dCompass_m_Norm_y);
dObjective_f_61=(2.0f*dCompass_b_x*(dEst_q_1*dEst_q_3 -
dEst_q_2*dEst_q_4)+2.0f*dCompass_b_z*(0.5-dEst_q_2*dEst_q_2-
dEst_q_3*dEst_q_3)-dCompass_m_Norm_z);

// Step 1.4 > Calculate Jacobian Matrix of Objective Function
dObjective_J_11=-2.0f*dEst_q_3;
dObjective_J_12= 2.0f*dEst_q_4;
dObjective_J_13=-2.0f*dEst_q_1;
dObjective_J_14= 2.0f*dEst_q_2;
dObjective_J_21= 2.0f*dEst_q_2;
dObjective_J_22= 2.0f*dEst_q_1;
dObjective_J_23= 2.0f*dEst_q_4;
dObjective_J_24= 2.0f*dEst_q_3;
dObjective_J_31= 0.0f;
dObjective_J_32=-4.0f*dEst_q_2;
dObjective_J_33=-4.0f*dEst_q_3;
dObjective_J_34= 0.0f;
dObjective_J_41=-2*dCompass_b_z*dEst_q_3;
dObjective_J_42=2*dCompass_b_z*dEst_q_4;
dObjective_J_43=-4*dCompass_b_x*dEst_q_3-2*dCompass_b_z*dEst_q_1;
dObjective_J_44=-4*dCompass_b_x*dEst_q_4+2*dCompass_b_z*dEst_q_2;
dObjective_J_51=-2*dCompass_b_x*dEst_q_4+2*dCompass_b_z*dEst_q_2;
dObjective_J_52=2*dCompass_b_x*dEst_q_3+2*dCompass_b_z*dEst_q_1;
dObjective_J_53=2*dCompass_b_x*dEst_q_2+2*dCompass_b_z*dEst_q_4;
dObjective_J_54=-2*dCompass_b_x*dEst_q_1+2*dCompass_b_z*dEst_q_3;
dObjective_J_61=2*dCompass_b_x*dEst_q_3;
dObjective_J_62=2*dCompass_b_x*dEst_q_4-4*dCompass_b_z*dEst_q_2;
dObjective_J_63=2*dCompass_b_x*dEst_q_1-4*dCompass_b_z*dEst_q_3;
dObjective_J_64=2*dCompass_b_x*dEst_q_2;
```

```
// Step 1.5 > Calculate Gradient of Objective Function
dObjective_Grad_f_11=dObjective_J_14*dObjective_f_21 +
dObjective_J_11*dObjective_f_11 + dObjective_J_41*dObjective_f_41 +
dObjective_J_51*dObjective_f_51 + dObjective_J_61*dObjective_f_61;
dObjective_Grad_f_21=dObjective_J_12*dObjective_f_11 +
dObjective_J_22*dObjective_f_21 + dObjective_J_32*dObjective_f_31 +
dObjective_J_42*dObjective_f_41 + dObjective_J_52*dObjective_f_51 +
dObjective_J_62*dObjective_f_61;
dObjective_Grad_f_31=dObjective_J_12*dObjective_f_21 +
dObjective_J_33*dObjective_f_31 + dObjective_J_13*dObjective_f_11 +
dObjective_J_43*dObjective_f_41 + dObjective_J_53*dObjective_f_51 +
dObjective_J_63*dObjective_f_61;
dObjective_Grad_f_41=dObjective_J_14*dObjective_f_11 +
dObjective_J_24*dObjective_f_21 + dObjective_J_44*dObjective_f_41 +
dObjective_J_54*dObjective_f_51 + dObjective_J_64*dObjective_f_61;

// Step 1.6 > Calculate Gradient Norm
dObjective_Grad_f_Norm=sqrt(dObjective_Grad_f_11*dObjective_Grad_f_11+dObject
ive_Grad_f_21*dObjective_Grad_f_21+dObjective_Grad_f_31*dObjective_Grad_f_31+
dObjective_Grad_f_41*dObjective_Grad_f_41);

// Step 1.7 > Calculate Direction of Error for Estimated Rate of Change of
Orientation
dEst_S_E_q_1_hat_dot=(dObjective_Grad_f_11/dObjective_Grad_f_Norm);
dEst_S_E_q_2_hat_dot=(dObjective_Grad_f_21/dObjective_Grad_f_Norm);
dEst_S_E_q_3_hat_dot=(dObjective_Grad_f_31/dObjective_Grad_f_Norm);
dEst_S_E_q_4_hat_dot=(dObjective_Grad_f_41/dObjective_Grad_f_Norm);

/*************************************************************************/
// Step 2: Gyroscope Bias Drift Compensation
/*************************************************************************/

// Step 2.1 > Compute Angular Error of Each Gyroscope Axis
// Note Only Orientation Considered, Hence 2*dEst_q_1*dEst_S_E_q_1_hat_dot-
2*dEst_q_2*dEst_S_E_q_2_hat_dot-2*dEst_q_3*dEst_S_E_q_3_hat_dot-
2*dEst_q_4*dEst_S_E_q_4_hat_dot = 0
dGyro_w_x_Error=2*dEst_q_1*dEst_S_E_q_2_hat_dot-
2*dEst_q_2*dEst_S_E_q_1_hat_dot-
2*dEst_q_3*dEst_S_E_q_4_hat_dot+2*dEst_q_4*dEst_S_E_q_3_hat_dot;
dGyro_w_y_Error=2*dEst_q_1*dEst_S_E_q_3_hat_dot+2*dEst_q_2*dEst_S_E_q_4_hat_d
ot-2*dEst_q_3*dEst_S_E_q_1_hat_dot-2*dEst_q_4*dEst_S_E_q_2_hat_dot;
dGyro_w_x_Error=2*dEst_q_1*dEst_S_E_q_4_hat_dot-
2*dEst_q_2*dEst_S_E_q_3_hat_dot+2*dEst_q_3*dEst_S_E_q_2_hat_dot+2*dEst_q_4*dE
st_S_E_q_1_hat_dot;

// Step 2.2 > Compute Gyroscope Bias
dGyro_w_x_Bias=dGyro_w_x_Bias+dGain_Zeta*dGyro_w_x_Error*dSample_Period_Delta
_t;
dGyro_w_y_Bias=dGyro_w_y_Bias+dGain_Zeta*dGyro_w_y_Error*dSample_Period_Delta
_t;
dGyro_w_z_Bias=dGyro_w_z_Bias+dGain_Zeta*dGyro_w_z_Error*dSample_Period_Delta
_t;

// Step 2.3 > Calculate Compensated Gyroscope Measurements
dGyro_w_x=dGyro_w_x - dGyro_w_x_Bias;
dGyro_w_y=dGyro_w_y - dGyro_w_y_Bias;
```

```
dGyro_w_z=dGyro_w_z - dGyro_w_z_Bias;

/*********************************************************************/
// Step 3: Compute Real Time Rate Quaternion from Gyro
/*********************************************************************/

// Step 3.1: > Calculate Quaternion Derivative for Rate of Change of
// Orientation of Earth Frame Relative To Sensor Frame
dS_E_q_w_1_dot=0.5f*(-dEst_q_2*dGyro_w_x - dEst_q_3*dGyro_w_y -
dEst_q_4*dGyro_w_z);
dS_E_q_w_2_dot=0.5f*( dEst_q_1*dGyro_w_x + dEst_q_3*dGyro_w_z -
dEst_q_4*dGyro_w_y);
dS_E_q_w_3_dot=0.5f*( dEst_q_1*dGyro_w_y - dEst_q_2*dGyro_w_z +
dEst_q_4*dGyro_w_x);
dS_E_q_w_4_dot=0.5f*(-dEst_q_1*dGyro_w_z + dEst_q_2*dGyro_w_y -
dEst_q_3*dGyro_w_x);

// Step 3.2: > Carry out Numerical Integration of Quaternion Derivative
// Note: Step 4.2 Not required as "alpha" constant - Constant used to
// augment "mu" (stepsize) - is assumed >>> 0.

/*********************************************************************/
// Step 4: Fusion of Accel / Gyro / Compass Sensor Data To Generate Increment
Estimate
/*********************************************************************/
dEst_S_E_q_1_dot=(dS_E_q_w_1_dot -
dGain_Beta*dObjective_Grad_f_11/dObjective_Grad_f_Norm);
dEst_S_E_q_2_dot=(dS_E_q_w_2_dot -
dGain_Beta*dObjective_Grad_f_21/dObjective_Grad_f_Norm);
dEst_S_E_q_3_dot=(dS_E_q_w_3_dot -
dGain_Beta*dObjective_Grad_f_31/dObjective_Grad_f_Norm);
dEst_S_E_q_4_dot=(dS_E_q_w_4_dot -
dGain_Beta*dObjective_Grad_f_41/dObjective_Grad_f_Norm);

/*********************************************************************/
// Step 5: Numerical Integration of Quaternion Rate Data
/*********************************************************************/

// Step 5.1 > Carry Out Numerical Integration
dEst_q_1=dEst_q_1+dEst_S_E_q_1_dot*dSample_Period_Delta_t;
dEst_q_2=dEst_q_2+dEst_S_E_q_2_dot*dSample_Period_Delta_t;
dEst_q_3=dEst_q_3+dEst_S_E_q_3_dot*dSample_Period_Delta_t;
dEst_q_4=dEst_q_4+dEst_S_E_q_4_dot*dSample_Period_Delta_t;

// Step 5.2 > Calculate Quaternion Norm
dEst_q_Norm = sqrt(dEst_q_1*dEst_q_1 + dEst_q_2*dEst_q_2 +dEst_q_3*dEst_q_3
+dEst_q_4*dEst_q_4);

/*********************************************************************/
// Step 6: Normalize and Output Quaternion Data
/*********************************************************************/
dEst_q_1 = dEst_q_1/dEst_q_Norm;
dEst_q_2 = dEst_q_2/dEst_q_Norm;
dEst_q_3 = dEst_q_3/dEst_q_Norm;
dEst_q_4 = dEst_q_4/dEst_q_Norm;
```

```
/*************************************************************************/
// Step 7: Switch Base Reference Frame If Needed
// Note: Represent quarternions as sensor attitude relative to fixed earth
// i.e d_Est_E_S_q_X where X = 1,2,3,4
/*************************************************************************/
/*
outQuat_q_1[0]=dEst_q_1;
outQuat_q_2[0]=-dEst_q_2;
outQuat_q_3[0]=-dEst_q_3;
outQuat_q_4[0]=-dEst_q_4;
*/
outQuat_q_1[0]=dEst_q_1;
outQuat_q_2[0]=dEst_q_2;
outQuat_q_3[0]=dEst_q_3;
outQuat_q_4[0]=dEst_q_4;

/*************************************************************************/
// Step 8: Compute Normalized Flux in Earth Frame
/*************************************************************************/
// Step 8.1 Compute Flux in Earth Frame
dCompass_h_x = 2*dCompass_m_Norm_x*(0.5f - dEst_q_3*dEst_q_3-
dEst_q_4*dEst_q_4)+2*dCompass_m_Norm_y*(dEst_q_2*dEst_q_3-
dEst_q_1*dEst_q_4)+2*dCompass_m_Norm_z*(dEst_q_2*dEst_q_4+dEst_q_1*dEst_q_3);
dCompass_h_y =
2*dCompass_m_Norm_x*(dEst_q_2*dEst_q_3+dEst_q_1*dEst_q_4)+2*dCompass_m_Norm_y
*(0.5f-dEst_q_2*dEst_q_2-
dEst_q_4*dEst_q_4)+2*dCompass_m_Norm_z*(dEst_q_3*dEst_q_4-dEst_q_1*dEst_q_2);
dCompass_h_z = 2*dCompass_m_Norm_x*(dEst_q_2*dEst_q_4-
dEst_q_1*dEst_q_3)+2*dCompass_m_Norm_y*(dEst_q_3*dEst_q_4+dEst_q_1*dEst_q_2)+
2*dCompass_m_Norm_z*(0.5f-dEst_q_2*dEst_q_2-dEst_q_3*dEst_q_3);

// Step 8.2 Normalize Flux in Earth Frame along X and Z axes.
dCompass_b_x = sqrt((dCompass_h_x*dCompass_h_x)+(dCompass_h_y*dCompass_h_y));
dCompass_b_z = dCompass_h_z;

// Output Quaternion Data
if (iDebug_Level >=Debug_Level_1) { printf("Quaternion Data> Q1: %.2f Q2:
%.2f Q3: %.2f Q4:
%.2f\n",outQuat_q_1[0],outQuat_q_2[0],outQuat_q_3[0],outQuat_q_4[0]);}


# endif
/* %%%-SFUNWIZ_wrapper_Outputs_Changes_END --- EDIT HERE TO _BEGIN */
}

/*
  * Updates function
  *
  */
void BBB_AHRS_IMU_MARG_v1_Update_wrapper(const real_T *inGyro_w_x,
                          const real_T *inGyro_w_y,
                          const real_T *inGyro_w_z,
                          const real_T *inAccel_a_x,
                          const real_T *inAccel_a_y,
```

```
                        const real_T *inAccel_a_z,
                        const real_T *inCompass_m_x,
                        const real_T *inCompass_m_y,
                        const real_T *inCompass_m_z,
                        const real_T *outQuat_q_1,
                        const real_T *outQuat_q_2,
                        const real_T *outQuat_q_3,
                        const real_T *outQuat_q_4,
                        const boolean_T *outStopSim,
                        real_T *xD)
{
  /* %%%-SFUNWIZ_wrapper_Update_Changes_BEGIN --- EDIT HERE TO _END */
/* Only Execute On Target*/
# ifndef MATLAB_MEX_FILE

// Recall Discrete State Data
// xD[0] = First Scan
// xD[1] = Estimated Quaternion dEst_q_1
// xD[2] = Estimated Quaternion dEst_q_2
// xD[3] = Estimated Quaternion dEst_q_3
// xD[4] = Estimated Quaternion dEst_q_4

if (xD[0]==0)
{
    // Initialize Quaternions

    // Set First Scan Bit
    xD[0]=1;
}

// Update Persistent Quaternion Data
xD[1]=outQuat_q_1[0];
xD[2]=outQuat_q_2[0];
xD[3]=outQuat_q_3[0];
xD[4]=outQuat_q_4[0];

xD[5]=dCompass_b_x;
xD[6]=dCompass_b_z;

// Update Gyro Bias Error
xD[7]=dGyro_w_x_Bias;
xD[8]=dGyro_w_y_Bias;
xD[9]=dGyro_w_z_Bias;


# endif
/* %%%-SFUNWIZ_wrapper_Update_Changes_END --- EDIT HERE TO _BEGIN */
}
```

13.12 MATLAB IMU/AHRS USER INFERFACE CODE

```
%% BBB GUI Function
function varargout = BBB_QuadRotor_GUI_Program_v4
% Close All Figures
close all;
clear all;
BBB_GUI=[];
BBB_GUI_Init_Data;
BBB_GUI_Construct_Fig;
BBB_GUI_Construct_VR;
BBB_GUI_Init_VR;
BBB_GUI_CheckState;

%EventListener_Scope_Attitude_Add;
%EventListener_Scope_Accel_Add;
%assignin('base','BBB_GUI_WS_Attitude',BBB_GUI.EventLstnrHandle.Attitude); %
Try GUIData instead!
%assignin('base','BBB_GUI_WS_Accel',BBB_GUI.EventLstnrHandle.Acceleration); %
Try GUIData instead!
%assignin('base','BBB_GUI_WS',BBB_GUI.q1);

varargout{1} = BBB_GUI.Figure;

%% Construct Figure
    function BBB_GUI_Construct_Fig
        %% Build Figure, W/ Configured Properties
        BBB_GUI.Figure = figure(...%'Tag',mfilename,...
        'Toolbar','none',...
        'MenuBar','none',...
        'IntegerHandle','off',...
        'Color',[0,0,0],...
        'Units','normalized',...
        'Position',[0.1 0.1, 0.8, 0.8],...
        'Resize','on',...
        'NumberTitle','off',...
        'HandleVisibility','callback',...
        'Name',sprintf('BLACKlink BeagleBone Black Attitude Estimation GUI.
Model: %s.mdl',BBB_GUI.Model.ModelName),...
        'CloseRequestFcn',@Func_UI_Close,...
        'Visible','off');

        %Set Graphic Variables
        Figure_FontName='verdana';
        Figure_FontSize=10;

        %%   Create Accelerometer Panel
        BBB_GUI.Panel_Accel = uipanel('Parent',BBB_GUI.Figure,...
            'Units','normalized',...
            'Position',[0 0.416 0.5 0.416],...
            'Title',' Accelerometer Plot ',...
            'BackgroundColor',get(BBB_GUI.Figure,'Color'),...
            'ForegroundColor','r',...
            'FontName',Figure_FontName,...
            'FontSize',Figure_FontSize,...
            'HandleVisibility','callback',...
```

```
    'Tag','AccelAxesPanel');

% Create Accelerometer Axis
BBB_GUI.Control.Axis_A = axes('Parent',BBB_GUI.Panel_Accel,...
    'HandleVisibility','callback',...
    'Unit','normalized',...
    'OuterPosition',[0 0 0.95 1],...
    'Xlim',[0 10],...
    'YLim',[-1 1],...
    'XColor','w',...
    'YColor','w',...
    'FontName',Figure_FontName,...
    'FontSize',Figure_FontSize,...
    'Tag','AccelAxes',...
    'Color','none');
title( BBB_GUI.Control.Axis_A,'Acceleration vs. Time');
set(get(BBB_GUI.Control.Axis_A,'Title'),'Color','w');
xlabel( BBB_GUI.Control.Axis_A,'Time (sec)');
ylabel( BBB_GUI.Control.Axis_A,'Acceleration (g)');
grid( BBB_GUI.Control.Axis_A,'on');
box( BBB_GUI.Control.Axis_A,'on');
%legend( BBB_GUI.Control.Axis_A,'X Accel', 'Y Accel', 'Z Accel');

% Create Accelerometer Axis Control Buttons -  Start Plot
uicontrol('Parent',BBB_GUI.Panel_Accel,...
    'Style','pushbutton',...
    'Units','normalized',...
    'Position',[0.87 0.48 0.12 0.1],...
    'BackgroundColor',get(BBB_GUI.Figure,'Color'),...
    'ForegroundColor','r',...
    'String','Enable Plot',...
    'Enable','On',...
    'FontName',Figure_FontName,...
    'FontSize',Figure_FontSize,...
    'Callback',@Func_CB_AccelPlotCtrl_On,...
    'HandleVisibility','callback',...
    'Tag','pbAccelPlotCtrl_On')

% Create Accelerometer Axis Control Buttons -  Stop Plot
uicontrol('Parent',BBB_GUI.Panel_Accel,...
    'Style','pushbutton',...
    'Units','normalized',...
    'Position',[0.87 0.28 0.12 0.1],...
    'BackgroundColor',get(BBB_GUI.Figure,'Color'),...
    'ForegroundColor','r',...
    'String','Disable Plot',...
    'Enable','Off',...
    'FontName',Figure_FontName,...
    'FontSize',Figure_FontSize,...
    'Callback',@Func_CB_AccelPlotCtrl_Off,...
    'HandleVisibility','callback',...
    'Tag','pbAccelPlotCtrl_Off')

% Create Handles For Accelerometer Axis Lines
Handle_Axis_A = BBB_GUI.Control.Axis_A;
Handle_Axis_A_Lines=nan(1,3);
```

```
        % Create Handles To Axis Lines For Accelerometer Readings
        % BBB_GUI.Control.Axis_A.LineHandles = nan(1,3);
        Axis_A_ColorOrder = get(Handle_Axis_A,'ColorOrder');
        for Axis_A_Line_Index = 1:length(Handle_Axis_A_Lines)
            Handle_Axis_A_Lines(Axis_A_Line_Index) =
line('Parent',BBB_GUI.Control.Axis_A,...
                'XData',[],...
                'YData',[],...
                'Color',Axis_A_ColorOrder(mod(Axis_A_Line_Index-
1,size(Axis_A_ColorOrder,1))+1,:),...
                'EraseMode','xor',...
                'Tag',sprintf('Axis_A_SignalLine_%d',Axis_A_Line_Index));
        end


        BBB_GUI.Control.Axis_A.LineHandles_1 = Handle_Axis_A_Lines(1);
        BBB_GUI.Control.Axis_A.LineHandles_2 = Handle_Axis_A_Lines(2);
        BBB_GUI.Control.Axis_A.LineHandles_3 = Handle_Axis_A_Lines(3);

        %%   Create Gyro Panel
        BBB_GUI.Panel_Gyro = uipanel('Parent',BBB_GUI.Figure,...
            'Units','normalized',...
            'Position',[0.5 0.416 0.5 0.416],...
            'Title',' Gyroscope Plot ',...
            'BackgroundColor',get(BBB_GUI.Figure,'Color'),...
            'ForegroundColor','r',...
            'FontName',Figure_FontName,...
            'FontSize',Figure_FontSize,...
            'HandleVisibility','callback',...
            'Tag','GyroAxesPanel');

        % Create Gyro Axis
        BBB_GUI.Control.Axis_G = axes('Parent',BBB_GUI.Panel_Gyro,...
            'HandleVisibility','callback',...
            'Unit','normalized',...
            'OuterPosition',[0.0 0.0 0.95 1],...
            'Xlim',[0 10],...
            'YLim',[-1 1],...
            'XColor','w',...
            'YColor','w',...
            'FontName',Figure_FontName,...
            'FontSize',Figure_FontSize,...
            'Tag','GyroAxes',...
            'Color','none');
        title( BBB_GUI.Control.Axis_G,'Rotational Velocity vs. Time');
        set(get(BBB_GUI.Control.Axis_G,'Title'),'Color','w');
        xlabel( BBB_GUI.Control.Axis_G,'Time (sec)');
        ylabel( BBB_GUI.Control.Axis_G,'Rotational Velocity (degrees)');
        grid( BBB_GUI.Control.Axis_G,'on');
        box( BBB_GUI.Control.Axis_G,'on');

         % Create Gyroscope Axis Control Buttons - Enable Plot
        uicontrol('Parent',BBB_GUI.Panel_Gyro,...
            'Style','pushbutton',...
            'Units','normalized',...
            'Position',[0.87 0.48 0.12 0.1],...
```

```
            'BackgroundColor',get(BBB_GUI.Figure,'Color'),...
            'ForegroundColor','r',...
            'String','Enable Plot',...
            'Enable','On',...
            'FontName',Figure_FontName,...
            'FontSize',Figure_FontSize,...
            'Callback',@Func_CB_GyroPlotCtrl_On,...
            'HandleVisibility','callback',...
            'Tag','pbGyroPlotCtrl_On')

        % Create Accelerometer Axis Control Buttons -  Stop Plot
        uicontrol('Parent',BBB_GUI.Panel_Gyro,...
            'Style','pushbutton',...
            'Units','normalized',...
            'Position',[0.87 0.28 0.12 0.1],...
            'BackgroundColor',get(BBB_GUI.Figure,'Color'),...
            'ForegroundColor','r',...
            'String','Disable Plot',...
            'FontName',Figure_FontName,...
            'FontSize',Figure_FontSize,...
            'Enable','Off',...
            'Callback',@Func_CB_GyroPlotCtrl_Off,...
            'HandleVisibility','callback',...
            'Tag','pbGyroPlotCtrl_Off')

        % Create Handles For Gyro Lines
        Handle_Axis_G = BBB_GUI.Control.Axis_G;
        Handle_Axis_G_Lines=nan(1,3);
        % Create Handles To Axis Lines For Gyroscope Readings
        % BBB_GUI.Control.Axis_A.LineHandles = nan(1,3);
        Axis_G_ColorOrder = get(Handle_Axis_G,'ColorOrder');
        for Axis_G_Line_Index = 1:length(Handle_Axis_G_Lines)
            Handle_Axis_G_Lines(Axis_G_Line_Index) =
line('Parent',BBB_GUI.Control.Axis_G,...
                'XData',[],...
                'YData',[],...
                'Color',Axis_G_ColorOrder(mod(Axis_G_Line_Index-
1,size(Axis_G_ColorOrder,1))+1,:),...
                'EraseMode','xor',...
                'Tag',sprintf('Axis_G_SignalLine_%d',Axis_G_Line_Index));
        end

        % Transfer Handles To Global Data
        BBB_GUI.Control.Axis_G.LineHandles_1 = Handle_Axis_G_Lines(1);
        BBB_GUI.Control.Axis_G.LineHandles_2 = Handle_Axis_G_Lines(2);
        BBB_GUI.Control.Axis_G.LineHandles_3 = Handle_Axis_G_Lines(3);

        %%   Create VR Canvas Panel - Front View!
        BBB_GUI.Panel_AttitudeFV= uipanel('Parent',BBB_GUI.Figure,...
            'Units','normalized',...
            'Position',[0 0 0.5 0.416],...
            'Title','Attitude Estimate - Front View ',...
            'BackgroundColor',get(BBB_GUI.Figure,'Color'),...
            'ForegroundColor','r',...
            'FontName',Figure_FontName,...
            'FontSize',Figure_FontSize,...
```

```
    'HandleVisibility','callback',...
    'Tag','pnlAttitudeFVVR');

%   Create VR Canvas Panel - Top View!
BBB_GUI.Panel_AttitudeTV= uipanel('Parent',BBB_GUI.Figure,...
    'Units','normalized',...
    'Position',[0.5 0 0.5 0.416],...
    'Title',' Attitude Estimate - Top View ',...
    'BackgroundColor',get(BBB_GUI.Figure,'Color'),...
    'ForegroundColor','r',...
    'FontName',Figure_FontName,...
    'FontSize',Figure_FontSize,...
    'HandleVisibility','callback',...
    'Tag','pnlAttitudeTVVR');

%%   Create Simulation Control Panel
BBB_GUI.Panel_SimCntrl= uipanel('Parent',BBB_GUI.Figure,...
    'Units','normalized',...
    'Position',[0 0.83 0.4 0.168],...
    'Title',' Simulation Control Panel ',...
    'BackgroundColor',get(BBB_GUI.Figure,'Color'),...
    'ForegroundColor','r',...
    'FontName',Figure_FontName,...
    'FontSize',Figure_FontSize,...
    'HandleVisibility','callback',...
    'Tag','pnlSimCntrl');


 % CreateSimulation Control Panel Buttons - Build Model
uicontrol('Parent',BBB_GUI.Panel_SimCntrl,...
    'Style','pushbutton',...
    'Units','normalized',...
    'Position',[0.112 0.3 0.2 0.4],...
    'BackgroundColor',get(BBB_GUI.Figure,'Color'),...
    'ForegroundColor','r',...
    'String','Build Model',...
    'Enable','On',...
    'FontName',Figure_FontName,...
    'FontSize',Figure_FontSize,...
    'Callback',@Func_CB_ModelBuild,...
    'HandleVisibility','callback',...
    'Tag','pbModelBuild')

 % CreateSimulation Control Panel Buttons - Start Model
uicontrol('Parent',BBB_GUI.Panel_SimCntrl,...
    'Style','pushbutton',...
    'Units','normalized',...
    'Position',[0.364 0.3 0.2 0.4],...
    'BackgroundColor',get(BBB_GUI.Figure,'Color'),...
    'ForegroundColor','r',...
    'String','Start Model',...
    'Enable','Off',...
    'FontName',Figure_FontName,...
    'FontSize',Figure_FontSize,...
    'Callback',@Func_CB_ModelStart,...
    'HandleVisibility','callback',...
```

```
            'Tag','pbModelStart')

         % CreateSimulation Control Panel Buttons - Stop Model
        uicontrol('Parent',BBB_GUI.Panel_SimCntrl,...
            'Style','pushbutton',...
            'Units','normalized',...
            'Position',[0.604 0.3 0.2 0.4],...
            'BackgroundColor',get(BBB_GUI.Figure,'Color'),...
            'ForegroundColor','r',...
            'String','Stop Model',...
            'Enable','Off',...
            'FontName',Figure_FontName,...
            'FontSize',Figure_FontSize,...
            'Callback',@Func_CB_ModelStop,...
            'HandleVisibility','callback',...
            'Tag','pbModelStop')

        %%   Create Simulation State Panel
        BBB_GUI.Panel_SimCntrl= uipanel('Parent',BBB_GUI.Figure,...
            'Units','normalized',...
            'Position',[0.402 0.83 0.39 0.168],...
            'Title',' Simulation State  ',...
            'BackgroundColor',get(BBB_GUI.Figure,'Color'),...
            'ForegroundColor','r',...
            'FontName',Figure_FontName,...
            'FontSize',Figure_FontSize,...
            'HandleVisibility','callback',...
            'Tag','pnlSimState');

        % Create Model Name Text Box Label
         BBB_GUI.Text_ModelNameLabel =
uicontrol('Parent',BBB_GUI.Panel_SimCntrl,...
            'Style','text',...
            'Units','normalized',...
            'Position',[0.03 0.67 0.2 0.2],...
            'BackgroundColor',get(BBB_GUI.Figure,'Color'),...
            'ForegroundColor','r',...
            'String','Model Name: ',...
            'FontName',Figure_FontName,...
            'FontSize',Figure_FontSize,...
            'HorizontalAlignment','left',...
            'HandleVisibility','callback',...
            'Tag','txtModelNameLabel'); %#ok

        % Create Model Name Text Box
        BBB_GUI.Text_ModelName =
uicontrol('Parent',BBB_GUI.Panel_SimCntrl,...
            'Style','edit',...
            'Units','normalized',...
            'Position',[0.02 0.3 0.51 0.4],...
            'String',sprintf('%s.mdl',BBB_GUI.Model.ModelName),...
            'Enable','inactive',...
            'Backgroundcolor','k',...
            'ForegroundColor','r',...
            'FontName',Figure_FontName,...
            'FontSize',Figure_FontSize,...
```

```
            'HandleVisibility','callback',...
            'Tag','textModelName'); %#ok

        % Create Simulation Time Text Box Label
         BBB_GUI.Text_SimTimeLabel =
uicontrol('Parent',BBB_GUI.Panel_SimCntrl,...
            'Style','text',...
            'Units','normalized',...
            'Position',[0.55 0.75 0.12 0.25],...
            'BackgroundColor',get(BBB_GUI.Figure,'Color'),...
            'ForegroundColor','r',...
            'String','Simulation Time: ',...
            'HorizontalAlignment','left',...
            'FontName',Figure_FontName,...
            'FontSize',Figure_FontSize,...
            'HandleVisibility','callback',...
            'Tag','txtSimulationTimeLabel'); %#ok

        % Create Simulation Time Text Box
        BBB_GUI.Text_SimTime = uicontrol('Parent',BBB_GUI.Panel_SimCntrl,...
            'Style','edit',...
            'Units','normalized',...
            'Position',[0.55 0.3 0.13 0.4],...
            'String',sprintf('0.0'),...
            'Enable','inactive',...
            'Backgroundcolor','k',...
            'ForegroundColor','r',...
            'HorizontalAlignment','center',...
            'FontName',Figure_FontName,...
            'FontSize',Figure_FontSize,...
            'HandleVisibility','callback',...
            'Tag','txtSimulationTime'); %#ok

        % Create System Time Text Box Label
         BBB_GUI.Text_SysTimeLabel =
uicontrol('Parent',BBB_GUI.Panel_SimCntrl,...
            'Style','text',...
            'Units','normalized',...
            'Position',[0.7 0.75 0.12 0.25],...
            'BackgroundColor',get(BBB_GUI.Figure,'Color'),...
            'ForegroundColor','r',...
            'String','System Time: ',...
            'HorizontalAlignment','left',...
            'FontName',Figure_FontName,...
            'FontSize',Figure_FontSize,...
            'HandleVisibility','callback',...
            'Tag','txtSystemTimeLabel'); %#ok

        % Create System Time Text Box
        BBB_GUI.Text_SysTime = uicontrol('Parent',BBB_GUI.Panel_SimCntrl,...
            'Style','edit',...
            'Units','normalized',...
            'Position',[0.7 0.3 0.13 0.4],...
            'String',sprintf('0.0'),...
            'Enable','inactive',...
            'Backgroundcolor','k',...
```

```
            'ForegroundColor','r',...
            'HorizontalAlignment','center',...
            'FontName',Figure_FontName,...
            'FontSize',Figure_FontSize,...
            'HandleVisibility','callback',...
            'Tag','txtSystemTime'); %#ok

        % Create Lag Time Text Box Label
         BBB_GUI.Text_LagTimeLabel =
uicontrol('Parent',BBB_GUI.Panel_SimCntrl,...
            'Style','text',...
            'Units','normalized',...
            'Position',[0.85 0.67 0.13 0.2],...
            'BackgroundColor',get(BBB_GUI.Figure,'Color'),...
            'ForegroundColor','r',...
            'String','Lag Time: ',...
            'HorizontalAlignment','left',...
            'FontName',Figure_FontName,...
            'FontSize',Figure_FontSize,...
            'HandleVisibility','callback',...
            'Tag','txtLagTimeLabel'); %#ok

        % Create Lag Time Text Box
        BBB_GUI.Text_LagTime = uicontrol('Parent',BBB_GUI.Panel_SimCntrl,...
            'Style','edit',...
            'Units','normalized',...
            'Position',[0.85 0.3 0.13 0.4],...
            'String','',...
            'Enable','inactive',...
            'BackgroundColor','k',...
            'ForegroundColor','r',...
            'HorizontalAlignment','center',...
            'FontName',Figure_FontName,...
            'FontSize',Figure_FontSize,...
            'HandleVisibility','callback',...
            'Tag','txtLagTime'); %#ok
        %set(BBB_GUI.Text_LagTime,'color','none');

        % Create Lag Time Text Box - Progress Bar
        BBB_GUI.Text_LagTimeProgBar =
uicontrol('Parent',BBB_GUI.Panel_SimCntrl,...
            'Style','edit',...
            'Units','normalized',...
            'Position',[0.85 0.3 0.005 0.4],...
            'String','',...
            'Enable','inactive',...
            'Backgroundcolor','g',...
            'ForegroundColor','r',...
            'HorizontalAlignment','center',...
            'FontName',Figure_FontName,...
            'FontSize',Figure_FontSize,...
            'HandleVisibility','callback',...
            'Tag','txtLagTimeProgBar'); %#ok
```

```
        %   Create BlackLink Logo Control Panel
        BBB_GUI.Text_Logo = uicontrol('Parent',BBB_GUI.Figure,...
            'Style','text',...
            'Units','normalized',...
            'Position',[0.8 0.83 0.2 0.12],...
            'BackgroundColor',get(BBB_GUI.Figure,'Color'),...
            'ForegroundColor',[1, 0.6, 0.0],...
            'FontSize', 40,...
            'FontWeight','bold',...
            'FontName','MS Sans Serif',...
            'HorizontalAlignment','center',...
            'String',' BLACKlink ',...
            'HandleVisibility','callback',...
            'Tag','txtLogo');

        %% GUI Final Setup
        % Create Handles Structure
        BBB_GUI.Handles = guihandles(BBB_GUI.Figure);

        % Save Application Data
         guidata(BBB_GUI.Figure,BBB_GUI);

        %Figure_Handles=findall(BBB_GUI.Figure)
        %Object_Handles=findobj(BBB_GUI.Figure)

        % Position the UI in the centre of the screen
        movegui(BBB_GUI.Figure,'center')

        % Make UI Visible
        set(BBB_GUI.Figure,'Visible','on');

    end

%% Function: Check Simulation State
    function BBB_GUI_CheckState

        % Load Model if Closed
        if
isempty(find_system('Type','block_diagram','Name',BBB_GUI.Model.ModelName));
            load_system(BBB_GUI.Model.ModelName); %load_system
        end

        %%% RDK Check HERE

        % Get Simulation Execution State
        SimState = get_param(BBB_GUI.Model.ModelName,'SimulationStatus');

        % Enable / Disable Functionality Depending on Simulation State
        switch SimState
            case 'stopped'
                % Set Button Enable States
                % Disable "Start Model" Button
                set(BBB_GUI.Handles.pbModelStart,'Enable','off');
                % Disable "Stop Model" Button
                set(BBB_GUI.Handles.pbModelStop,'Enable','off');
```

```
                % Enable "Build Model" Button
                set(BBB_GUI.Handles.pbModelBuild,'Enable','on');
                % Disable "Enable Plot - Accel" Button
                set(BBB_GUI.Handles.pbAccelPlotCtrl_On,'Enable','off')
                % Disable "Enable Plot - Gyro" Button
                set(BBB_GUI.Handles.pbGyroPlotCtrl_On,'Enable','off')
            case 'external'
                % Disable "Start Model" Button
                set(BBB_GUI.Handles.pbModelStart,'Enable','off');
                % Enable "Stop Model" Button
                set(BBB_GUI.Handles.pbModelStop,'Enable','on');
                % Disable "Build Model" Button
                set(BBB_GUI.Handles.pbModelBuild,'Enable','off');
                % Add Attitude Event Listener
                EventListener_Scope_Attitude_Add;
                % Scope Data To Workspace

assignin('base','BBB_GUI_WS_Attitude',BBB_GUI.EventLstnrHandle.Attitude); %
Try GUIData insteadr;
            otherwise
                errordlg('Model Is Not In Recognized State. Stop Model and
Relaunch UI',...
                    'UI Error - Model in Unrecognized State','modal');
        end
    end

%% Function: Build Model
    function Func_CB_ModelBuild(hObject,eventdata)
        % Build Model

        % Disable "Build Model" Button
        set(BBB_GUI.Handles.pbModelBuild,'Enable','off');

        % Configure Wait Bar Text
        BuildModel_WaitBar_String = sprintf('%s\n\n%s%s\n\n%s',...
            'Building Model:',strrep(BBB_GUI.Model.ModelName,'_','\_'),'...
',...
            'Please Be Patient As This Should Take Approximately a Minute.
Cheers!');
        BuildModel_WaitBar = waitbar(0,BuildModel_WaitBar_String,...
            'Name','Building Model...   ');
        %Pause
        pause(0.1);
        % Animate WaitBar - For Fun
        for BuildModel_Time_Increment = 0:20
            % Pause
            %pause(BuildModel_Estimated_Waiting/1000);
            % Update WaitBar

waitbar(BuildModel_Time_Increment/100,BuildModel_WaitBar,BuildModel_WaitBar_S
tring,...
            'Name','Building Model...   ');
        end

        % Set the Simulation Mode to External
        set_param(BBB_GUI.Model.ModelName,'SimulationMode','external');
```

```
        % Build the Model
        rtwbuild(BBB_GUI.Model.ModelName);
        % Estimated Build Waiting
        BuildModel_Estimated_Waiting=35;

        % Animate WaitBar - For Fun
        for BuildModel_Time_Increment = 20:100
            % Pause
            %pause(BuildModel_Estimated_Waiting/1000);
            % Update WaitBar

waitbar(BuildModel_Time_Increment/100,BuildModel_WaitBar,BuildModel_WaitBar_S
tring,...
            'Name','Building Model...    ');
        end

        % Reset the Simulation Mode
        % set_param(BBB_GUI.Model.ModelName,'SimulationMode','external');
        % Build Complete Delete the Waitbar
        delete(BuildModel_WaitBar);

        % Enable "Start Model" Button
        set(BBB_GUI.Handles.pbModelStart,'Enable','on');

        % Set Model Build Flag
        BBB_GUI.Model.BuildFlag=true;

        % Flush the Graphics Buffer
        %drawnow
    end

%% Function: Start Model
    function Func_CB_ModelStart(hObject,eventdata)
        % Start Model

        % Load Model if Closed
        if
isempty(find_system('Type','block_diagram','Name',BBB_GUI.Model.ModelName));
            load_system(BBB_GUI.Model.ModelName);
        end

        % Reset Plot Data Lines
        set(BBB_GUI.Handles.Axis_A_SignalLine_1,'XData',[],'YData',[]);
        set(BBB_GUI.Handles.Axis_A_SignalLine_2,'XData',[],'YData',[]);
        set(BBB_GUI.Handles.Axis_A_SignalLine_3,'XData',[],'YData',[]);
        set(BBB_GUI.Handles.Axis_G_SignalLine_1,'XData',[],'YData',[]);
        set(BBB_GUI.Handles.Axis_G_SignalLine_2,'XData',[],'YData',[]);
        set(BBB_GUI.Handles.Axis_G_SignalLine_3,'XData',[],'YData',[]);

        % Set Simulation Stop Time To Inifinity
        set_param(BBB_GUI.Model.ModelName,'StopTime','inf');
        % Set the Simulation Mode to External
        %set_param(BBB_GUI.Model.ModelName,'SimulationMode','external');
        % Start the GRT code
        % system(sprintf('%s -tf inf -w &',BBB_GUI.Model.ModelName));
```

```
        % Connect to the code
        set_param(BBB_GUI.Model.ModelName,'SimulationCommand','connect');
        % Start Model
        set_param(BBB_GUI.Model.ModelName,'SimulationCommand','start');

        % Start Measurement of System Time
        BBB_GUI.Time.System=tic;
        % Disable Start Button
        set(BBB_GUI.Handles.pbModelStart,'Enable','off');
        % Enable Stop button
        set(BBB_GUI.Handles.pbModelStop,'Enable','on');
        % Enable "Enable Plot - Accel" Button
        set(BBB_GUI.Handles.pbAccelPlotCtrl_On,'Enable','on')
        % Enable "Enable Plot - Gyro" Button
        set(BBB_GUI.Handles.pbGyroPlotCtrl_On,'Enable','on')


        % Add Attitude Event Listener
        EventListener_Scope_Attitude_Add;
        % Scope Data To Workspace

assignin('base','BBB_GUI_WS_Attitude',BBB_GUI.EventLstnrHandle.Attitude); %
Try GUIData insteadr;

    end

%% Function: Stop Model
    function Func_CB_ModelStop(hObject,eventdata)


        % Disable Stop Button
        set(BBB_GUI.Handles.pbModelStop,'Enable','off');
        % Enable Build Button
        set(BBB_GUI.Handles.pbModelBuild,'Enable','on');
        % Enable Start Button
        set(BBB_GUI.Handles.pbModelStart,'Enable','off');
        % Disable "Enable Plot - Accel" Button
        set(BBB_GUI.Handles.pbAccelPlotCtrl_On,'Enable','off')
        % Disable "Enable Plot - Gyro" Button
        set(BBB_GUI.Handles.pbGyroPlotCtrl_On,'Enable','off')
        % Disable "Disable Plot - Accel" Button
        set(BBB_GUI.Handles.pbAccelPlotCtrl_Off,'Enable','off')
        % Disable "Disable Plot - Gyro" Button
        set(BBB_GUI.Handles.pbGyroPlotCtrl_Off,'Enable','off')


        % Stop Model Simulation
        set_param(BBB_GUI.Model.ModelName,'SimulationCommand','stop');
        % Disconnect Model
        set_param(BBB_GUI.Model.ModelName,'SimulationCommand','disconnect');

        % Remove Attitude Event Listner (Not Required)
         delete(BBB_GUI.EventLstnrHandle.Attitude)

    end
```

```
%% Function: Acceleration Plot Control
    function Func_CB_AccelPlotCtrl_On(hObject,eventdata)
        % Call Function To Add Event Listener
        EventListener_Scope_Accel_Add;
    end

%% Function: Acceleration Plot Control
    function Func_CB_AccelPlotCtrl_Off(hObject,eventdata)
        % Call Function To Add Event Listener
        EventListener_Scope_Accel_Remove;
    end

%% Function: Gyroscope Plot Control
    function Func_CB_GyroPlotCtrl_On(hObject,eventdata)
        % Add Event Listener
        EventListener_Scope_Gyro_Add
    end

%% Function: Gyroscope Plot Control
    function Func_CB_GyroPlotCtrl_Off(hObject,eventdata)
        % Remove Event Listener
        EventListener_Scope_Gyro_Remove;
    end

%% Build VR Viewers
    function BBB_GUI_Construct_VR
        % Set VR Preferences
        vrsetpref('DefaultViewer', 'internalv5')
        % Obtain vrworld and construct canvas
        BBB_GUI.QR_VR_World =
vrworld('.\Software_Quadrotor_VRML\BBB_Plane.WRL');
        open(BBB_GUI.QR_VR_World);
        %Create the VR Viewer Embedded in MATLAB GUI
        % set(BBB_GUI.Canvas,'Units','normalized')
        % BBB_GUI.Canvas = vr.canvas(BBB_GUI.QR_VR_World,BBB_GUI.Figure,[0 0
0.5 0.416]);

        % Configure Front View Viewer
        BBB_GUI.Canvas.Front =
vr.canvas(BBB_GUI.QR_VR_World,'Parent',BBB_GUI.Figure,...
        'Units','normalized',...
        'Position',[0.01 0.01 0.48 0.386],...
        'Viewpoint','Close Up View');

         % Configure Top View Viewer
         BBB_GUI.Canvas.Top =
vr.canvas(BBB_GUI.QR_VR_World,'Parent',BBB_GUI.Figure,...
        'Units','normalized',...
        'Position',[0.51 0.01 0.48 0.386],...
        'Viewpoint','Top View');
    end

%% Init VR Viewer Data
    function BBB_GUI_Init_VR
```

```
        %Accessing the vrnode to animate
        BBB_GUI.BBB_Aircraft_node=vrnode(BBB_GUI.QR_VR_World,'BBB_Aircraft');
        BBB_GUI.q1=0;
        BBB_GUI.q2=0;
        BBB_GUI.q3=0;
        BBB_GUI.q4=0;
    end

%% Init GUI Data
    function BBB_GUI_Init_Data
        %Accessing the vrnode to animate
        BBB_GUI.Counter_1=0;

BBB_GUI.Model.ModelName='BBB_AHRS_IMU_v4_Test_3DAnimation_HMI_Backup';
        BBB_GUI.Axis_A_YLim_New=1;
        BBB_GUI.Axis_G_YLim_New=1;
        BBB_GUI.Axis_C_YLim_New=1;

        % Get Simulation State

        % Set Model Build Flag
        BBB_GUI.Model.BuildFlag=false;

        %BBB_GUI.EventLstnrHandle.Attitude=[];
    end

%% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Callback Function for adding an event listener to the gain block
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    function EventListener_Scope_Attitude_Add


        ScopeName=sprintf('%s/Scope_Attitude',BBB_GUI.Model.ModelName);

        % Add Listener For Aircraft Attitude Data Change

BBB_GUI.EventLstnrHandle.Attitude=add_exec_event_listener(ScopeName,...
            'PostOutputs',@EventListener_Scope_Attitude_Func);

    end

%% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Callback Function for executing the event listener on the gain block
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    function EventListener_Scope_Attitude_Func(block, eventdata) %#ok

        % Note: This callback is called by all the block listeners.  No
effort has
        % been made to time synchronise the data from each signal.  Rather it
is
        % assumed that since each block calls this function at every time
step and
        % hence the time synchronisation will come "for free".  This may not
be the
```

```
        % case for other models and additional code may be required for them
to
        % work/display data correctly.

        % disp('Trigger?')
        % Read Data From Attitude Scope
        ScopeData=block.InputPort(1).Data;
        BBB_GUI.q1=ScopeData(1);
        BBB_GUI.q2=ScopeData(2);
        BBB_GUI.q3=ScopeData(3);
        BBB_GUI.q4=ScopeData(4);
        %BBB_GUI.q2=block.InputPort(2).Data
        %BBB_GUI.q3=block.InputPort(3).Data
        %BBB_GUI.q4=block.InputPort(4).Data

        % Update Aircraft Aittitude in VR World

        BBB_GUI.BBB_Aircraft_node.rotation=[BBB_GUI.q1 BBB_GUI.q2 BBB_GUI.q3
BBB_GUI.q4];

        % Update VR Images
        vrdrawnow;
        %drawnow;

        % Update Simulation Time

set(BBB_GUI.Handles.txtSimulationTime,'String',get_param(BBB_GUI.Model.ModelN
ame,'SimulationTime'));

        % Update System Time

set(BBB_GUI.Handles.txtSystemTime,'String',toc(BBB_GUI.Time.System));

        % Illustrate Lag Time On Progress Bar
        % Calculate LagTime - Max Bar Out If Lag Time > 60
        LagTime=abs(get_param(BBB_GUI.Model.ModelName,'SimulationTime')-
toc(BBB_GUI.Time.System))/60;
        % Set Lag Time Max
        if LagTime>1
            LagTime=1;
        end
        % Display Lag Time Bar
        set(BBB_GUI.Handles.txtLagTimeProgBar,'Position',[0.85 0.3
LagTime*0.13 0.4]);
        % Display Lag Time When Space Adequate % 'Position',[0.85 0.3 0.005
0.4],...
        if LagTime>0.2

set(BBB_GUI.Handles.txtLagTimeProgBar,'String',num2str(LagTime*60));
        end

    end

%% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Callback Function for adding an event listener to the gain block
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    function EventListener_Scope_Accel_Add


        ScopeName=sprintf('%s/Scope_Accel',BBB_GUI.Model.ModelName);

        % Add Listener For Aircraft Attitude Data Change

BBB_GUI.EventLstnrHandle.Acceleration=add_exec_event_listener(ScopeName,...
            'PostOutputs',@EventListener_Scope_Accel_Func);

        % Disable "Enable Plot" Button
        set(BBB_GUI.Handles.pbAccelPlotCtrl_On,'Enable','off');

        % Enable "Disable Plot" Button
        set(BBB_GUI.Handles.pbAccelPlotCtrl_Off,'Enable','on');

        % Trigger Bit
        BBB_GUI.EventLstnrHandle.Trigger_A=false;

    end

%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Callback Function For Removing Event Listener on Accelerometer Scope Block
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
    function EventListener_Scope_Accel_Remove


        delete(BBB_GUI.EventLstnrHandle.Acceleration);

        % Disable "Disable Plot" Button
        set(BBB_GUI.Handles.pbAccelPlotCtrl_Off,'Enable','off');

        % Enable "Enable Plot" Button
        set(BBB_GUI.Handles.pbAccelPlotCtrl_On,'Enable','on');

    end

%% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Callback Function for adding an event listener to the gain block
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    function EventListener_Scope_Gyro_Add


        ScopeName=sprintf('%s/Scope_Gyro',BBB_GUI.Model.ModelName);

        % Add Listener For Aircraft Attitude Data Change

BBB_GUI.EventLstnrHandle.Gyroscope=add_exec_event_listener(ScopeName,...
            'PostOutputs',@EventListener_Scope_Gyro_Func);
```

```
        % Disable "Enable Plot" Button
        set(BBB_GUI.Handles.pbGyroPlotCtrl_On,'Enable','off');

        % Enable "Disable Plot" Button
        set(BBB_GUI.Handles.pbGyroPlotCtrl_Off,'Enable','on');

        % Trigger Bit
        BBB_GUI.EventLstnrHandle.Trigger_G=false;

    end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% Callback Function For Removing Event Listener on Gyroscope Scope Block
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
    function EventListener_Scope_Gyro_Remove


        delete(BBB_GUI.EventLstnrHandle.Gyroscope);

        % Disable "Disable Plot" Button
        set(BBB_GUI.Handles.pbGyroPlotCtrl_Off,'Enable','off');

        % Enable "Enable Plot" Button
        set(BBB_GUI.Handles.pbGyroPlotCtrl_On,'Enable','on');

    end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Callback Function For Gyro Event Listener Gyroscope Scope Block
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    function EventListener_Scope_Gyro_Func(block, eventdata) %#ok

        Handle_Figure=guidata(BBB_GUI.Figure);
        Handle_Axis_G = BBB_GUI.Control.Axis_G;
        Handle_Axis_G_Line_1=BBB_GUI.Control.Axis_G.LineHandles_1;
        Handle_Axis_G_Line_2=BBB_GUI.Control.Axis_G.LineHandles_2;
        Handle_Axis_G_Line_3=BBB_GUI.Control.Axis_G.LineHandles_3;

        % Get the data currently being displayed on the axis
        Scope_Xdata = get(Handle_Axis_G_Line_1,'XData');
        Scope_Ydata_Gyro_X = get(Handle_Axis_G_Line_1,'YData');
        Scope_Ydata_Gyro_Y = get(Handle_Axis_G_Line_2,'YData');
        Scope_Ydata_Gyro_Z = get(Handle_Axis_G_Line_3,'YData');

        % Get the simulation time and the block data
        Scope_Time = block.CurrentTime;

        % Read Data From Gyro Scope
        ScopeData=block.InputPort(1).Data;
        Scope_Gyro_X=ScopeData(1);
        Scope_Gyro_Y=ScopeData(2);
```

```
        Scope_Gyro_Z=ScopeData(3);


        % Only the last 1001 points worth of data
        % The model sample time is 0.001 so this represents 1000 seconds of
data
        % RDK - Here current time is appended to X Data
        % RDK - Here current data from input port(1) is appended to Y Data
        if length(Scope_Xdata) < 1001
            Scope_Xdata_New = [Scope_Xdata Scope_Time];
            Scope_Ydata_Gyro_X_New = [Scope_Ydata_Gyro_X Scope_Gyro_X];
            Scope_Ydata_Gyro_Y_New = [Scope_Ydata_Gyro_Y Scope_Gyro_Y];
            Scope_Ydata_Gyro_Z_New = [Scope_Ydata_Gyro_Z Scope_Gyro_Z];
        else
            Scope_Xdata_New = [Scope_Xdata(2:end) Scope_Time];
            Scope_Ydata_Gyro_X_New = [Scope_Ydata_Gyro_X(2:end)
Scope_Gyro_X];
            Scope_Ydata_Gyro_Y_New = [Scope_Ydata_Gyro_Y(2:end)
Scope_Gyro_Y];
            Scope_Ydata_Gyro_Z_New = [Scope_Ydata_Gyro_Z(2:end)
Scope_Gyro_Z];
        end

        % Display the new data set
        set(Handle_Axis_G_Line_1,...
            'XData',Scope_Xdata_New,...
            'YData',Scope_Ydata_Gyro_X_New);


        set(Handle_Axis_G_Line_2,...
            'XData',Scope_Xdata_New,...
            'YData',Scope_Ydata_Gyro_Y_New);


        set(Handle_Axis_G_Line_3,...
            'XData',Scope_Xdata_New,...
            'YData',Scope_Ydata_Gyro_Z_New)

        % The axes limits may also need changing
        Scope_XLim_New = [max(0,Scope_Time-10) max(10,Scope_Time)];
        Scope_YLim_Data_Array=[Scope_Gyro_X,Scope_Gyro_Y,Scope_Gyro_Z];
        Scope_YMax = max(abs(Scope_YLim_Data_Array));

        if Scope_YMax>BBB_GUI.Axis_G_YLim_New
            % Update Max
            Scope_YLim_New = Scope_YMax*[-1,1];
            BBB_GUI.Axis_G_YLim_New=Scope_YMax;
        else
            % Keep Old Max
            Scope_YLim_New = BBB_GUI.Axis_G_YLim_New*[-1,1];
        end

        % RDK - Note that will need to apply this to Y - Axis as well
        set(BBB_GUI.Handles.GyroAxes,'Xlim',Scope_XLim_New);
        set(BBB_GUI.Handles.GyroAxes,'Ylim',Scope_YLim_New);
```

```
        % Transfer Data Back To Handles?
        %BBB_GUI.Control.Axis_G.LineHandles_1=Handle_Axis_G_Line_1;
        %BBB_GUI.Control.Axis_G.LineHandles_2=Handle_Axis_G_Line_2;
        %BBB_GUI.Control.Axis_G.LineHandles_3=Handle_Axis_G_Line_3;

        % Add Legend To Graph
        if BBB_GUI.EventLstnrHandle.Trigger_G == false

           BBB_GUI.Control.Axis_G.Legend=legend(BBB_GUI.Handles.GyroAxes,'X
Rot', 'Y Rot', 'Z Rot');
           set(BBB_GUI.Control.Axis_G.Legend,'TextColor','w');
           set(BBB_GUI.Control.Axis_G.Legend,'EdgeColor','r');
           set(BBB_GUI.Control.Axis_G.Legend,'Color','none');
           BBB_GUI.EventLstnrHandle.Trigger_G=true;
        end
    end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Callback Function For Accelerometer Event Listener on Accelerometer Scope
Block
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    function EventListener_Scope_Accel_Func(block, eventdata) %#ok

        Handle_Figure=guidata(BBB_GUI.Figure);
        Handle_Axis_A = BBB_GUI.Control.Axis_A;
        Handle_Axis_A_Line_1=BBB_GUI.Control.Axis_A.LineHandles_1;
        Handle_Axis_A_Line_2=BBB_GUI.Control.Axis_A.LineHandles_2;
        Handle_Axis_A_Line_3=BBB_GUI.Control.Axis_A.LineHandles_3;

        % Get the data currently being displayed on the axis
        Scope_Xdata = get(Handle_Axis_A_Line_1,'XData');
        Scope_Ydata_Accel_X = get(Handle_Axis_A_Line_1,'YData');
        Scope_Ydata_Accel_Y = get(Handle_Axis_A_Line_2,'YData');
        Scope_Ydata_Accel_Z = get(Handle_Axis_A_Line_3,'YData');

        % Get the simulation time and the block data
        Scope_Time = block.CurrentTime;

        % Read Data From Attitude Scope
        ScopeData=block.InputPort(1).Data;
        Scope_Accel_X=ScopeData(1);
        Scope_Accel_Y=ScopeData(2);
        Scope_Accel_Z=ScopeData(3);


        % Only the last 1001 points worth of data
        % The model sample time is 0.001 so this represents 1000 seconds of
data
        % RDK - Here current time is appended to X Data
        % RDK - Here current data from input port(1) is appended to Y Data
        if length(Scope_Xdata) < 1001
            Scope_Xdata_New = [Scope_Xdata Scope_Time];
            Scope_Ydata_Accel_X_New = [Scope_Ydata_Accel_X Scope_Accel_X];
            Scope_Ydata_Accel_Y_New = [Scope_Ydata_Accel_Y Scope_Accel_Y];
```

```
            Scope_Ydata_Accel_Z_New = [Scope_Ydata_Accel_Z Scope_Accel_Z];
        else
            Scope_Xdata_New = [Scope_Xdata(2:end) Scope_Time];
            Scope_Ydata_Accel_X_New = [Scope_Ydata_Accel_X(2:end)
Scope_Accel_X];
            Scope_Ydata_Accel_Y_New = [Scope_Ydata_Accel_Y(2:end)
Scope_Accel_Y];
            Scope_Ydata_Accel_Z_New = [Scope_Ydata_Accel_Z(2:end)
Scope_Accel_Z];
        end

        % Display the new data set
        set(Handle_Axis_A_Line_1,...
            'XData',Scope_Xdata_New,...
            'YData',Scope_Ydata_Accel_X_New);


        set(Handle_Axis_A_Line_2,...
            'XData',Scope_Xdata_New,...
            'YData',Scope_Ydata_Accel_Y_New);


        set(Handle_Axis_A_Line_3,...
            'XData',Scope_Xdata_New,...
            'YData',Scope_Ydata_Accel_Z_New)

        % The axes limits may also need changing
        Scope_XLim_New = [max(0,Scope_Time-10) max(10,Scope_Time)];
        Scope_YLim_Data_Array=[Scope_Accel_X,Scope_Accel_Y,Scope_Accel_Z];
        Scope_YMax = max(abs(Scope_YLim_Data_Array));

        if Scope_YMax>BBB_GUI.Axis_A_YLim_New
            % Update Max
            Scope_YLim_New = Scope_YMax*[-1,1];
            BBB_GUI.Axis_A_YLim_New=Scope_YMax;
        else
            % Keep Old Max
            Scope_YLim_New = BBB_GUI.Axis_A_YLim_New*[-1,1];
        end

        % RDK - Note that will need to apply this to Y - Axis as well
        set(BBB_GUI.Handles.AccelAxes,'Xlim',Scope_XLim_New);
        set(BBB_GUI.Handles.AccelAxes,'Ylim',Scope_YLim_New);

        % Transfer Data Back To Handles?
        %BBB_GUI.Control.Axis_A.LineHandles_1=Handle_Axis_A_Line_1;
        %BBB_GUI.Control.Axis_A.LineHandles_2=Handle_Axis_A_Line_2;
        %BBB_GUI.Control.Axis_A.LineHandles_3=Handle_Axis_A_Line_3;

        % Add Legend To Graph
        if BBB_GUI.EventLstnrHandle.Trigger_A == false

           BBB_GUI.Control.Axis_A.Legend=legend(BBB_GUI.Handles.AccelAxes,'X
Accel', 'Y Accel', 'Z Accel');
            set(BBB_GUI.Control.Axis_A.Legend,'TextColor','w');
```

```
            set(BBB_GUI.Control.Axis_A.Legend,'EdgeColor','r');
            set(BBB_GUI.Control.Axis_A.Legend,'Color','none');
            BBB_GUI.EventLstnrHandle.Trigger_A=true;
        end




    end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Callback Function For UI Close
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    function Func_UI_Close(hObject,eventdata) %#ok

        % Close UI If Not Model Loaded
        if
isempty(find_system('Type','block_diagram','Name',BBB_GUI.Model.ModelName));
            % Delete Window
            delete(gcbo);
        else
            % Only Allow UI Close if Simuation Model Has Been Stopped

            % Get Simulation Execution State
            SimState = get_param(BBB_GUI.Model.ModelName,'SimulationStatus');

            % Enable / Disable Functionality Depending on Simulation State
            switch SimState
                case 'stopped'
                    % Allow UI To Close
                    %Close the Simulink model
                    close_system(BBB_GUI.Model.ModelName,0);
                    % destroy the window
                    delete(gcbo);
                otherwise
                    errordlg('The Model Must Be Stopped Before Closing The
UI',...
                        'UI Close Error','modal');
            end


        end
    end
end
```