How To Guide: ArduPilot 2.0 Simulink Blockset

François Hugon
Robert F. Hartley
Brian DeRosa
Gulfstream Aerospace Corporation
Embry-Riddle Aeronautical University
Savannah, Georgia

Christopher Carvalho
Embry-Riddle Aeronautical University
Daytona Beach, Florida

November 2, 2012

Acknowledgements

Thank you to our wives and significant others–Sara, Laura, Bailey, and Lauren–who patiently dealt with long nights, neglected chores, and were our biggest supporters to be successful in this adventure.

Thanks to Brian DeRosa for allowing us to takeover his house and convert his dining room table into a hangar.

Thanks to the world renowned RC pilot, George Hicks, for being our safety pilot, being a great resource for tips and tricks for RC aircraft, always having the parts we needed up to and including entire airframes, and for always being willing to fly with almost zero notice.

Thanks to Richard Ruff from Mathworks for providing technical expertise and guidance while developing the APM2 Blockset.

Abstract

As part of developing a new graduate level Guidance, Navigation, and Control course at Embry-Riddle Aeronautical University, a Simulink blockset has been developed that will allow students to work solely in Matlab/Simulink for the development of GNC code. This code can be downloaded directly to the ArduPilot 2.0 integrated sensor and processing package allowing the students to easily develop unmanned aerial vehicles. The blockset is useful in that it eliminates the requirement to have a knowledgeable programmer, and a knowledgeable electrical engineer to integrate hardware and software with the airframe. This guide is intended to describe the functionality of the blockset, provide recommendations for how to implement control designs, and steps on how to embed the code onto the ArduPilot directly from Simulink. The blockset is in early development and there is room for improvement; it is expected that future students or the academic community as a whole will advance the capabilities going forward.

Contents

List of Figures

List of Tables

Nomenclature

b_x	Magnetic Field Strength Along Body x-Axis
b_y	Magnetic Field Strength Along Body y-Axis
b_z	Magnetic Field Strength Along Body z-Axis
n_x	Acceleration Along Body x-Axis
n_y	Acceleration Along Body y-Axis
n_z	Acceleration Along Body z-Axis
p	Roll Rate, deg/sec
q	Pitch Rate, deg/sec
r	Yaw Rate, deg/sec
A/D	Analog to Digital
APM1	ArduPilot Mega 1.0
APM2	ArduPilot Mega 2.0
CG	Center of Gravity
ERAU	Embry-Riddle Aeronautical University
GAC	Gulfstream Aerospace Corporation
GPS	Global Positioning System
$\mathrm{I^2C}$	Inter-Integrated Circuit
I/O	Input/Output
IDE	Integrated Development Environment
IMU	Inertial Measurement Unit
LED	Light Emitting Diode

Mb	Megabyte
MEMS	Micro-Electro-Mechanical System
MHz	Megahertz
MIMO	Multiple Input Multiple Output
ms	milliseconds
NVM	Non-Volatile Memory
PWM	Pulse Width Modulation
QFN	Quad Flat No-Leads
RC	Remote Control
ROTH	Run On Target Hardware
SISO	Single Input Single Output
SPI	Serial Peripheral Interface
TM	Telemetry
UAV	Unmanned Aerial Vehicle
UINT	Unsigned Integer
USB	Universal Serial Bus
UTC	Coordinated Universal Time, also known as Zulu Time

Chapter 1

Introduction

The ArduPilot 2.0 is an off-the-shelf product that integrates a sensor suite with an Atmega 2560 processor for use as an autopilot. This document describes how to use the Simulink blockset that was developed for this product, and will describe how a single Simulink model can be developed and embedded directly to the ArduPilot 2.0. This model will include the necessary code to read the sensors, read the RC PWM signals from the RC transmitter, output PWM signals to control servos and an engine, output data to a telemetry system, and record data to a flash memory chip.

Chapter 2

Setup and Usage

2.1 Matlab

In order for Simulink models to be embedded to target hardware, the user must have access to a Matlab version of 2012a or newer. As of the 2012a release, the Mathworks included a Run On Target Hardware (ROTH) feature that essentially performs the same function as the Embedded Coder, however the actual compiled code is not viewable by the user. ROTH is included as part of the standard Simulink license, but it should be noted, however, that ROTH currently only works in Windows. Mathworks has indicated that it will eventually work in Linux or on a Mac, but as of the release of this document, that capability has not been implemented.

A supported compiler must also be installed. The Matlab help documentation has guidance on which compilers are supported and how to make Matlab recognize them [6].

2.2 Simulink Configuration Parameters

The actual model that is being embedded onto the ArduPilot must have certain Configuration Parameters set. The model must have the following settings:

- Type: Fixed Step
- Time Step: This is your board frame time, therefore needs to be determined such that the board runs in real time. Use the timing blocks, as described in Section 3.3, to assist in determining the computation time of your code.
- Solver: Discrete (no continuous states)
- Hardware Implementation `->` Device Vendor: Atmel

2.3 Run On Target Hardware

The first time that ROTH is used, the actual target software must be installed. Since this is an ArduPilot, with an Arduino Mega 2560 chip, the user must install the Arduino Target Hardware. To accomplish this, from Simulink, select the Tools menu, then Run On Target Hardware, and then Install Target. Select Arduino. Simulink will install the target in the following location: `C:/MATLAB-/Targets/R2012b`.

2.4 Arduino Target Cores Directory

The Arduino IDE is included as part of the target install. The IDE includes a folder of standard files for compiling code. These files had to be modified for the Simulink Blockset. The path of the folder is `C:/MATLAB/Targets/R2012b-/arduino-1.0/hardware/arduino/cores /arduino`.

Included with this How To Guide is a new `cores/arduino`, folder that will replace the default one. It is recommended that the default folder is saved with a modified filename (i.e. `arduino_default`), in the event that the raw Arduino IDE needs to be used.

2.5 IOWrappers.cpp file

There are two files included as part of the target download that need to be modified. Eventually, these files may be deleted as improvements are made to this blockset. However, at this time, modifications need to be made to these files to prevent compiler errors. The two files are `src/io _wrapper.cpp` and `include/io_wrapper.h`, in the directory `C:/MATLAB/Targets/R2012b/arduino`.

For both files, the user should simply add the line `#if 0` before any of the code, and add the line `#endif` to the last line of the files. It is recommended to save backup copies of the original files, or at least to comment the new lines to indicate that they are the modifications from the original file.

2.6 Simulink Path

The folders shown in Table 2.1 must be added to the Matlab path.

2.7 S-Function Build

The final step in the setup is to compile the code for the Simulink blocks. A Matlab script has been included to do just this. Simply navigate to the APM2 folder for the Matlab directory and run the file `ArduinoBuildMexFiles.m`. The script calls files in the directory structure from this folder location. It should be noted that a Matlab supported compiler is required for this to function. A list of Matlab supported compilers is provided in reference [6].

ArduPilot/APM2		
	Arduino	
	Blocks	
		Analog_IO
		Auxiliary
		Baro
		DataTypeConversions
		Discrete_IO
		FlashMemory
		GPS
		IMU
		Magnetometer
		Pitot
		RCChannels
		Serial
		Timing

Table 2.1: Folders to add to Matlab path

2.8 Downloading a Model

Once the environment has been setup as described above and a Simulink model has been built and tested in simulation (see Chapter 4 for example models to use), downloading the model to the APM2 hardware is a simple task. First, connect the APM2 to the computer using a miniUSB cable. This will cause the board to power up; however, the USB bus does not power the servos so do not be alarmed if the servos appear inoperative. From the Simulink model, select the `Tools` menu, then the `Run On Target Hardware` menu. If this is the first time that the model is being downloaded to the APM2, then ROTH will need to prepare the model for download. Select the `Prepare` menu, which will open the Configuration Parameters window. Simply use the default settings (including the 9600 baud rate for the serial ports).

Once the ROTH Configuration Parameters are set, navigate the same ROTH menus and then select `Run`. Simulink will begin the automatic process of generating the run-time code, compiling the executable, and downloading to the APM2. The process may take upwards of a minute for more complicated models and success will be indicated in the status bar area of the Simulink model window.

Chapter 3

ArduPilot Mega 2.0 Library Blocks

3.1 Sensor Blocks

3.1.1 Inertial Measurement Unit

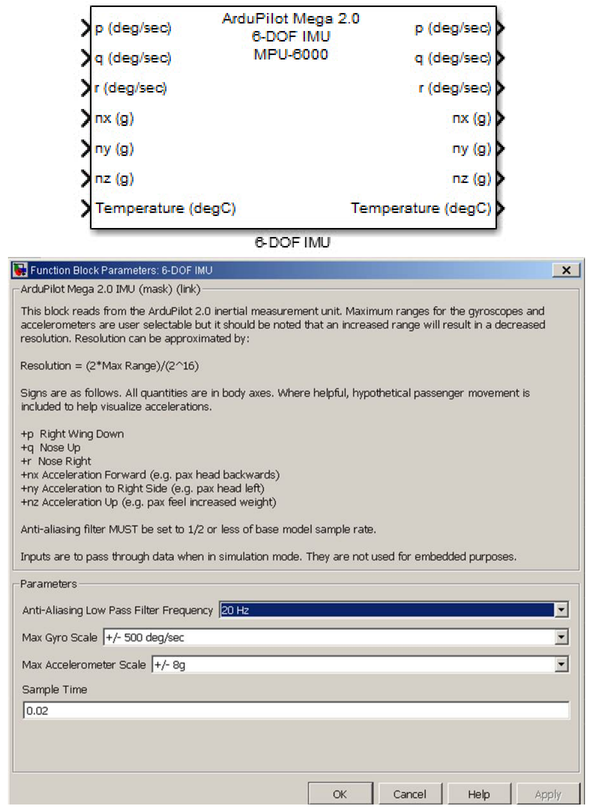
The InvenSense MPU-6000 is a MEMS device that incorporates a 3-axis accelerometer with a 3-axis gyroscope together in a tiny 4x4x0.9mm QFN footprint. This digital device contains a 16-bit A/D converter which allows for very fine resolution on the inertial parameters. Also, since the chip handles the analog to digital conversion, integration into the user model is simply a matter of scaling an integer. The device communicates with the microprocessor over the common board level peripheral interface bus, SPI.

To prevent anti-aliasing, the chip has a built in digital low pass filters with user selectable cut-off frequencies. This is a major advantage because it allows use with a slower process without any additional hardware. The accelerometer has a maximum range of $\pm$16g, while the gyros have a maximum range of $\pm$2000 deg/sec. Temperature, in degrees Celsius from standard day is also available.

The device also has a built in self test that is used by the board firmware to assess the health of the unit upon start up. More information regarding this device can be found at reference [5].

The IMU block has three options that the user must select in the mask of the IMU block. First, the low pass filter frequency must be set to half or less of the base sample rate. The range of the gyros and accelerometers must also be set. Finally, the sample time should match the overall sample time of the ArduPilot. Figure 3.1 shows the IMU block and mask.

The inputs to the block are simply the simulated values of each of the accelerations and angular rates as well as temperature, which are detailed in Table 3.1. These are passed through the block in simulation, and not used when the model is embedded to the APM2. The outputs of the block are the measured values from the inertial measurement unit, each of which are detailed in Table 3.2.



ArduPilot Mega 2.0 IMU (mask) (link)

This block reads from the ArduPilot 2.0 inertial measurement unit. Maximum ranges for the gyroscopes and accelerometers are user selectable but it should be noted that an increased range will result in a decreased resolution. Resolution can be approximated by:

Resolution = (2*Max Range)/(2^16)

Signs are as follows. All quantities are in body axes. Where helpful, hypothetical passenger movement is included to help visualize accelerations.

+p Right Wing Down
+q Nose Up
+r Nose Right
+nx Acceleration Forward (e.g. pax head backwards)
+ny Acceleration to Right Side (e.g. pax head left)
+nz Acceleration Up (e.g. pax feel increased weight)

Anti-aliasing filter MUST be set to 1/2 or less of base model sample rate.

Inputs are to pass through data when in simulation mode. They are not used for embedded purposes.

Parameters

Anti-Aliasing Low Pass Filter Frequency: 20 Hz
Max Gyro Scale: +/- 500 deg/sec
Max Accelerometer Scale: +/- 8g
Sample Time: 0.02

Figure 3.1: IMU block and mask

Calibration Behind the scenes, the ArduPilot will calibrate the IMU upon startup. The aircraft should be left level and stationary during power up until the board relinquishes control to the pilot transmitter. Any disturbances during the calibration routine could result in inaccurate acceleration and angular rate measurements.

3.1.2 Barometric Pressure Sensor

The MEAS Switzerland MS5611 is a small form factor digital device that contains an impressive 24 bit ADC that comes calibrated from the factory. The chip required very little setup other than defining the data bus rates and was capable of providing static pressure so precise that the altitude of the aircraft could be determined within inches. The update rate of the sensor is 75 Hz.

Port No.	Label	Units	Description
1	p (sim)	deg/s	Simulated value of body roll rate
2	q (sim)	deg/s	Simulated value of body pitch rate
3	r (sim)	deg/s	Simulated value of body yaw rate
4	nx (sim)	g	Simulated value of body x axis component of acceleration
5	ny (sim)	g	Simulated value of body y axis component of acceleration
6	nz (sim)	g	Simulated value of body z axis component of acceleration
7	Temperature (sim)	degC	Simulated value of the temperature of the IMU

Table 3.1: 6-DOF IMU Block, Inputs

Port No.	Label	Units	Description
1	p	deg/s	Body roll rate
2	q	deg/s	Body pitch rate
3	r	deg/s	Body yaw rate
4	nx	g	Body x axis component of acceleration
5	ny	g	Body y axis component of acceleration
6	nz	g	Body z axis component of acceleration
7	Temperature	degC	Temperature of the IMU

Table 3.2: 6-DOF IMU Block, Outputs

For more information regarding this device, see reference [11].

The inputs to the block are simply the simulated values of the static pressure and the temperature, which are detailed in Table 3.3. These are passed through the block in simulation, and not used when the model is embedded to the APM2. The outputs of the block are the measured values of static pressure and temperature as detailed in Table 3.4.

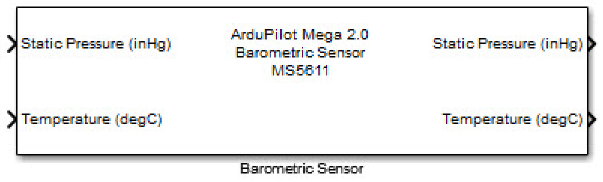


Figure 3.2: Barometric Pressure block

The barometric sensor block outputs the local barometric pressure. It also outputs the ambient temperature.

Port No.	Label	Units	Description
1	Static Pressure (sim)	inHg	Simulated value of atmospheric static pressure
2	Temperature (sim)	degC	Simulated value of atmospheric temperature

Table 3.3: Barometric Sensor Block, Inputs

Port No.	Label	Units	Description
1	Static Pressure	inHg	Atmospheric static pressure
2	Temperature	degC	Atmospheric temperature

Table 3.4: Barometric Sensor Block, Outputs

3.1.3 Magnetometer

The Honeywell HMC5843 3-axis digital magnetometer measures the magnetic field strength in each of the three axes of the unit. Like the IMU, it is a digital device that communicates with the microprocessor over a high speed serial bus. Also like the IMU, it has self test features that are excited during board start up so as to calibrate the unit to a known induced offset.

For more information regarding this device, see reference [4].

As the field strength of Earth is very weak, the magnetometer is extremely sensitive. While this is good for resolution, it also means that the device is very susceptible to electromagnetic noise and constant biases due to local conductive materials in the vicinity of the sensor. The engine, with its spinning magnets and large induced magnetic field, further degrades the performance of the unit as a function of engine speed. These problems are made worse by the requirement that the APM2 be installed near the center of gravity —and thus near the engine—in order to reduce inertial location effects. A remotely located I^2C magnetometer that connects to the APM2's external I^2C port is one option for improving this problem, but was neither researched for this project nor a Simulink block created as part of the APM2 blockset.

To calibrate for the local magnetic field, reference [10] was used to obtain the known total magnetic field strength for a given location. Next, the aircraft was configured to record all three field components while the airframe was rotated through all possible attitudes in an area free of interference and large metal objects. Once the data were downloaded, a non-linear minimization routine such as Matlab's `fmincon` or `fminsearch` was used to find the vector, C, that minimized the following cost function,

$$J = \left(B_T - \sqrt{K_{bx}(b_x + O_x)^2 + K_{by}(b_y + O_y)^2 + K_{bz}(b_z + O_z)^2} \right)^2 \tag{3.1}$$

where,

$$C = \begin{bmatrix} K_{bx} \\ K_{by} \\ K_{bz} \\ O_x \\ O_y \\ O_z \end{bmatrix}$$

and B_T was the known total field strength.

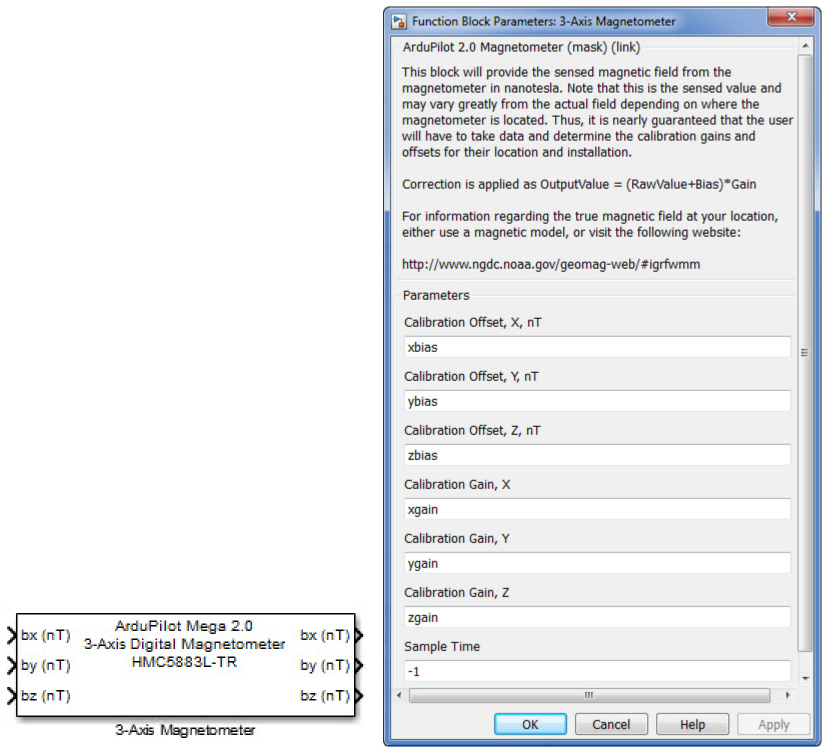
Once the vector, C, had been determined, it was entered directly into the magnetometer block mask, as shown in Figure 3.3 and was therefore applied to all measurements output from the block.

It should be noted that additional logic must be inserted to account for dynamic offsets introduced by the motor. This blockset is intended to replicate the raw sensor measurements, therefore any engine affects will need to be addressed separately by the user.

The inputs to the block are simply the simulated values of the magnetic field components, which are detailed in Table 3.5. These are passed through the block in simulation, and not used when the model is embedded to the APM2. The outputs of the block are the measured magnetic fields as detailed in Table 3.6.

3.1.4 GPS

The MediaTek MT3329 10 Hz GPS Receiver includes the GPS receiver, processor, and a patch antenna complete in a very small package. It communicates



Function Block Parameters: 3-Axis Magnetometer

ArduPilot 2.0 Magnetometer (mask) (link)

This block will provide the sensed magnetic field from the magnetometer in nanotesla. Note that this is the sensed value and may vary greatly from the actual field depending on where the magnetometer is located. Thus, it is nearly guaranteed that the user will have to take data and determine the calibration gains and offsets for their location and installation.

Correction is applied as OutputValue = (RawValue+Bias)*Gain

For information regarding the true magnetic field at your location, either use a magnetic model, or visit the following website:

http://www.ngdc.noaa.gov/geomag-web/#igrfwmm

Parameters

Calibration Offset, X, nT: xbias

Calibration Offset, Y, nT: ybias

Calibration Offset, Z, nT: zbias

Calibration Gain, X: xgain

Calibration Gain, Y: ygain

Calibration Gain, Z: zgain

Sample Time: -1

Figure 3.3: Magnetometer block and mask

Port No.	Label	Units	Description
1	bx, nT (sim)	nT	Simulated value of magnetic field, x body axis component
2	by, nT (sim)	nT	Simulated value of magnetic field, y body axis component
3	bz, nT (sim)	nT	Simulated value of magnetic field, z body axis component

Table 3.5: Magnetometer Block, Inputs

Port No.	Label	Units	Description
1	bx, nT	nT	Magnetic field, x body axis component
2	by, nT	nT	Magnetic field, y body axis component
3	bz, nT	nT	Magnetic field, z body axis component

Table 3.6: Magnetometer Block, Outputs

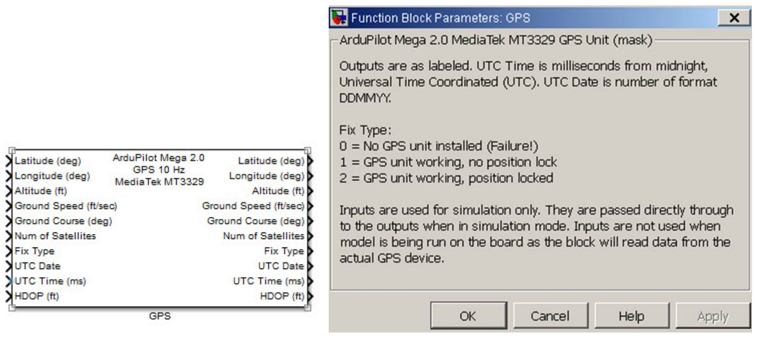
with the AT Mega microprocessor over a standard RS-232 serial link.

For more information regarding this device, see reference [9].

The inputs to the block are simply the simulated values of the GPS measurements, which are detailed in Table 3.7. These are passed through the block in simulation, and not used when the model is embedded to the APM2. The outputs of the block are the measured GPS parameters as detailed in Table 3.8.

The Fix Type output from the GPS block is a simple numerical representation of the status of the GPS unit. A value of zero indicates that no GPS device is available to the processor. This should never be seen, and if it is, almost certainly indicates a hardware failure. A value of one indicates that the processor and the GPS unit are communicating normally but that the GPS does not yet have a lock. Conversely, a value of two means that the GPS has acquired a 3-D lock on the unit's location and is ready for navigation. The block will also permanently illuminate the blue LED when locked.

It should be noted that the time and date are UTC based, with the time being in milliseconds after midnight. Figure 3.4 presents the GPS block and mask.



ArduPilot Mega 2.0 MediaTek MT3329 GPS Unit (mask)

Outputs are as labeled. UTC Time is milliseconds from midnight, Universal Time Coordinated (UTC). UTC Date is number of format DDMMYY.

Fix Type:
0 = No GPS unit installed (Failure!)
1 = GPS unit working, no position lock
2 = GPS unit working, position locked

Inputs are used for simulation only. They are passed directly through to the outputs when in simulation mode. Inputs are not used when model is being run on the board as the block will read data from the actual GPS device.

Figure 3.4: GPS block and mask

Port No.	Label	Units	Description
1	Latitude (sim)	deg	Simulated value of latitude
2	Longitude (sim)	deg	Simulated value of longitude
3	Altitude (sim)	ft	Simulated value of altitude
4	Ground Speed (sim)	ft/sec	Simulated value of ground speed
5	Ground Course (sim)	deg	Simulated value of ground course
6	Num of Satellites (sim)	–	Simulated value of number of satellites used for GPS
7	Fix Type (sim)	–	Simulated value of fix type as described above
8	UTC Date (sim)	string	Simulated value of UTC Date
9	UTC Time (sim)	ms	Simulated value of UTC time
10	HDOP (sim)	ft	Simulated value of horizontal dilution of precision

Table 3.7: GPS Block, Inputs

Port No.	Label	Units	Description
1	Latitude	deg	Value of latitude
2	Longitude	deg	Value of longitude
3	Altitude	ft	Value of altitude
4	Ground Speed	ft/sec	Value of ground speed
5	Ground Course	deg	Value of ground course
6	Num of Satellites	–	Value of number of satellites used for GPS
7	Fix Type	–	Value of fix type as described above
8	UTC Date	string	Value of UTC Date
9	UTC Time	ms	Value of UTC time
10	HDOP	ft	Value of horizontal dilution of precision

Table 3.8: GPS Block, Outputs

3.1.5 Pitot Probe

Airspeed measurement is a very important parameter for both state estimation and control. The APM2, however, does not include a way to measure airspeed. Therefore, a miniature pitot probe was installed in the nose of the airplane measuring the impact pressure at the probe location. The ports of the probe were connected to a Freescale MPXV7002DP Differential Pressure Sensor. The sensor was then connected to the APM2 via one of the analog input pin sets. This sensor was the only analog sensor onboard the aircraft and needed to be calibrated using a known flow field. Once calibrated the output of this sensor was a voltage proportional to dynamic pressure and this signal was used to compute true airspeed.

One curious behavior regarding this sensor was its atrocious performance at low speed. For example, when the airplane was stationary, it was common to see a very noisy 20 ft/sec being reported from the pitot probe. Fortunately, the accuracy of the device improved and the noise decreased as dynamic pressure increased.

For more information regarding this device, see reference [3].

The input to the block is simply the simulated values of dynamic pressure, which is detailed in Table 3.9. This value is passed through the block in simulation, and not used when the model is embedded to the APM2. The output of the block is the measured dynamic pressure from the pitot probe and differential pressure sensor, as detailed in Table 3.8.

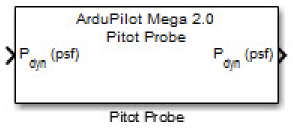


Figure 3.5: Differential Pressure Sensor block

Port No.	Label	Units	Description
1	P_{dyn} (sim)	psf	Simulated value of dynamic pressure

Table 3.9: Pitot Probe Block, Input

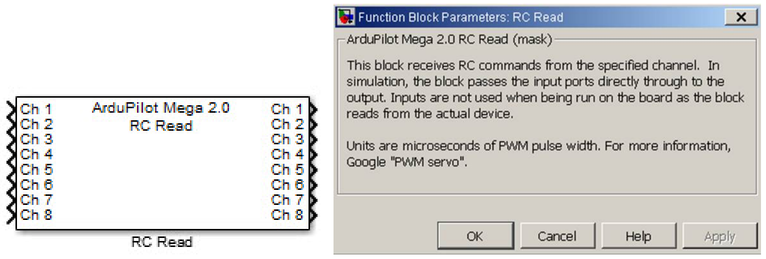
Port No.	Label	Units	Description
1	P_{dyn} (sim)	psf	Dynamic pressure

Table 3.10: Pitot Probe Block, Output

3.2 Receiver Input / Servo Output

3.2.1 RC Read

The RC Read block reads the PWM signals connected to the RC Input channels on the APM. These inputs are typically the outputs of the RC receiver. The PWM signals are in microseconds of pulse width. Figure 3.6 shows the RC Read block and mask. Table 3.11 and Table 3.12 detail the inputs and outputs of the RC Read block.



ArduPilot Mega 2.0 RC Read (mask)

This block receives RC commands from the specified channel. In simulation, the block passes the input ports directly through to the output. Inputs are not used when being run on the board as the block reads from the actual device.

Units are microseconds of PWM pulse width. For more information, Google "PWM servo".

Figure 3.6: RC Read block and mask

Port No.	Label	Units	Description
1	Ch 1 (sim)	μs	Simulated value of channel 1 PWM signal
2	Ch 2 (sim)	μs	Simulated value of channel 2 PWM signal
3	Ch 3 (sim)	μs	Simulated value of channel 3 PWM signal
4	Ch 4 (sim)	μs	Simulated value of channel 4 PWM signal
5	Ch 5 (sim)	μs	Simulated value of channel 5 PWM signal
6	Ch 6 (sim)	μs	Simulated value of channel 6 PWM signal
7	Ch 7 (sim)	μs	Simulated value of channel 7 PWM signal
8	Ch 8 (sim)	μs	Simulated value of channel 8 PWM signal

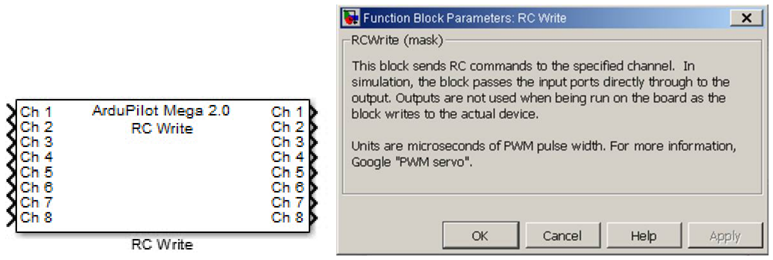
Table 3.11: RC Read Block, Inputs

Port No.	Label	Units	Description
1	Ch 1	μs	Value of channel 1 PWM signal
2	Ch 2	μs	Value of channel 2 PWM signal
3	Ch 3	μs	Value of channel 3 PWM signal
4	Ch 4	μs	Value of channel 4 PWM signal
5	Ch 5	μs	Value of channel 5 PWM signal
6	Ch 6	μs	Value of channel 6 PWM signal
7	Ch 7	μs	Value of channel 7 PWM signal
8	Ch 8	μs	Value of channel 8 PWM signal

Table 3.12: RC Read Block, Outputs

3.2.2 RC Write

The RC Write block outputs PWM signals to the output channels on the APM which are typically connected to the servos. The PWM signals are in microseconds of pulse width. Figure 3.7 shows the RC Write block and mask. Table 3.13 and Table 3.14 detail the inputs and outputs of the RC Write block.



This block sends RC commands to the specified channel. In simulation, the block passes the input ports directly through to the output. Outputs are not used when being run on the board as the block writes to the actual device.

Units are microseconds of PWM pulse width. For more information, Google "PWM servo".

Figure 3.7: RC Write block and mask

Port No.	Label	Units	Description
1	Ch 1	μs	Value of channel 1 PWM signal
2	Ch 2	μs	Value of channel 2 PWM signal
3	Ch 3	μs	Value of channel 3 PWM signal
4	Ch 4	μs	Value of channel 4 PWM signal
5	Ch 5	μs	Value of channel 5 PWM signal
6	Ch 6	μs	Value of channel 6 PWM signal
7	Ch 7	μs	Value of channel 7 PWM signal
8	Ch 8	μs	Value of channel 8 PWM signal

Table 3.13: RC Write Block, Inputs

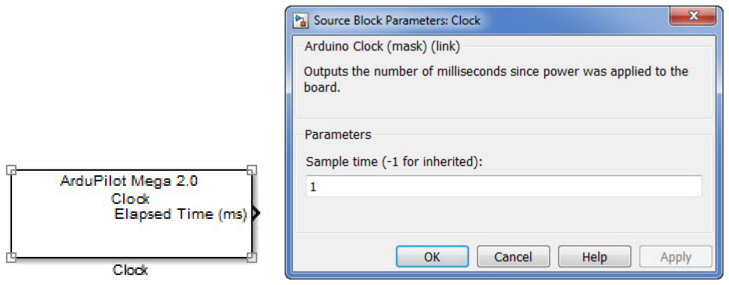
Port No.	Label	Units	Description
1	Ch 1 (sim)	μs	Simulated value of channel 1 PWM signal
2	Ch 2 (sim)	μs	Simulated value of channel 2 PWM signal
3	Ch 3 (sim)	μs	Simulated value of channel 3 PWM signal
4	Ch 4 (sim)	μs	Simulated value of channel 4 PWM signal
5	Ch 5 (sim)	μs	Simulated value of channel 5 PWM signal
6	Ch 6 (sim)	μs	Simulated value of channel 6 PWM signal
7	Ch 7 (sim)	μs	Simulated value of channel 7 PWM signal
8	Ch 8 (sim)	μs	Simulated value of channel 8 PWM signal

Table 3.14: RC Write Block, Outputs

3.3 Timing

3.3.1 Clock

The clock block outputs the time, in milliseconds, since the board was powered on. This is executed at each time step. Figure 3.8 shows the block and mask for the Clock. Table 3.15 details the and outputs of the Clock block.



Arduino Clock (mask) (link)

Outputs the number of milliseconds since power was applied to the board.

Parameters

Sample time (-1 for inherited):

1

Figure 3.8: Clock block and mask

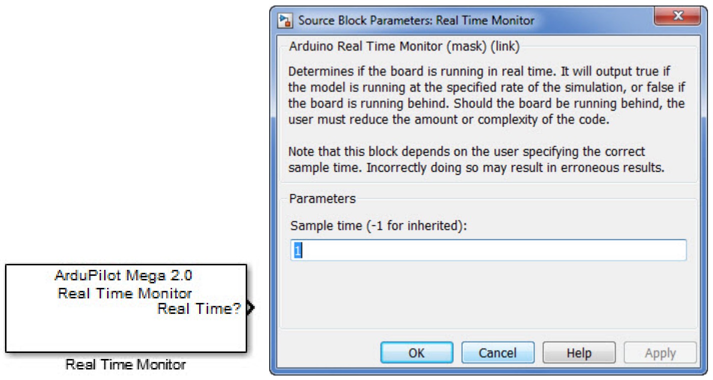
3.3.2 Real Time Monitor

The real time monitor block outputs a Boolean value that is true if the board is completing all of its calculations within the specified frame time. This block

Port No.	Label	Units	Description
1	Elapsed Time	ms	Time elapsed since the board has been powered

Table 3.15: Clock Block, Output

is useful in determining if the code is too complex or if the board frequency is too high. Figure 3.9 shows the Real Time Monitor block and mask. Table 3.16 details the and outputs of the Real Time Monitor block.



Arduino Real Time Monitor (mask) (link)

Determines if the board is running in real time. It will output true if the model is running at the specified rate of the simulation, or false if the board is running behind. Should the board be running behind, the user must reduce the amount or complexity of the code.

Note that this block depends on the user specifying the correct sample time. Incorrectly doing so may result in erroneous results.

Parameters

Sample time (-1 for inherited): 1

Figure 3.9: Real Time Monitor block and mask

The real time monitor block outputs a Boolean value that is true if the board is completing all of its calculations within the specified frame time. This block is very useful as an indicator that code is too complex for the specified sample time.

Port No.	Label	Units	Description
1	Real Time?	–	True if the board is operating in real time

Table 3.16: Real Time Monitor Block, Output

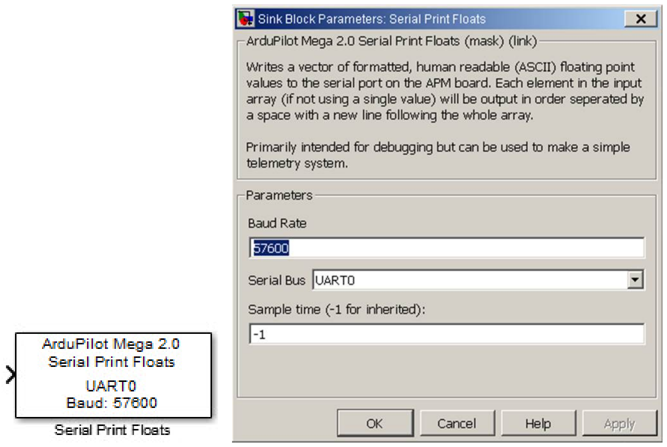
3.4 Data Output

3.4.1 Telemetry

There are 2 blocks that can be used for transmitting data back to the ground using the XBee transceiver. The first will provide a readable stream of data that can be displayed using a terminal interface. The second method will provide the raw, uncoded, binary data, that can be post-processed on the ground. This is useful for a Ground Station protocol that can read the raw data and parse it appropriately for the Ground Station protocol.

Serial Print Floats

In order to output data in a simple, readable format, a Serial Print Floats block has been created. Figure 3.10 shows the Serial Print Floats block and mask.


Sink Block Parameters: Serial Print Floats

ArduPilot Mega 2.0 Serial Print Floats (mask) (link)

Writes a vector of formatted, human readable (ASCII) floating point values to the serial port on the APM board. Each element in the input array (if not using a single value) will be output in order seperated by a space with a new line following the whole array.

Primarily intended for debugging but can be used to make a simple telemetry system.

Parameters

Baud Rate: 57600

Serial Bus: UART0

Sample time (-1 for inherited): -1

ArduPilot Mega 2.0 Serial Print Floats UART0 Baud: 57600

Serial Print Floats

Figure 3.10: Serial Print Floats block and mask

The Serial Print Floats block takes in an array (mux) of floating point values, and outputs it over a serial bus so it can be read and displayed as text in a terminal interface such as Putty. This is very useful for debugging or to serve as a simple ground station. The serial port desired can be selected in the mask, as can the baud rate. Generally, a baud rate of 57,600 or 115,200 produces reliable data at reasonably fast rates. Note that the slower the baud rate, the longer the block takes to write data which can negatively affect the system frame time.

By default, the ArduPilot has the USB serial port connected to the same serial port that is used by the XBee. This only matters if the user wants to be able to use the USB serial port separately from the XBee serial. If the user does want to have independent data on the USB port from the XBee port, then a minor board operation will need to be performed. The specifics are described on the ArduPilot XBee website, reference [2]. The procedure simply cuts 2 existing solder bridges, and makes 2 different ones. This will isolate the 2 serial ports from each other.

Port No.	Label	Units	Description
1	N/A	Array of Floats	Displays the array of floats over selected serial port

Table 3.17: Serial Print Floats Block, Input

Serial Write

For a more complicated telemetry system that requires raw binary data, The Serial Write block writes a vector of bytes to the telemetry serial port on the APM 2. Note that the default bus is UART0, however, since this is shared with the USB bus, one can follow the solder bridge instructions on the ArduPilot page (reference [2]) to enable UART2 if so desired. The bus to be used is selectable in the mask.

Note that this block writes the raw bytes fed to it. This typically means that some amount of code will be required on the receiving end to reconstruct the data into a useful display.

Port No.	Label	Units	Description
1	N/A	Array of Bytes	Outputs the byte array over serial port

Table 3.18: Serial Write Block, Input

3.4.2 USB Output

The Serial Print Floats block can also be used to output via the USB Serial Port. As described in the Telemetry section, by default, the USB serial port is the same as the XBee serial port. This data can be viewed with a serial port interface such as Putty.

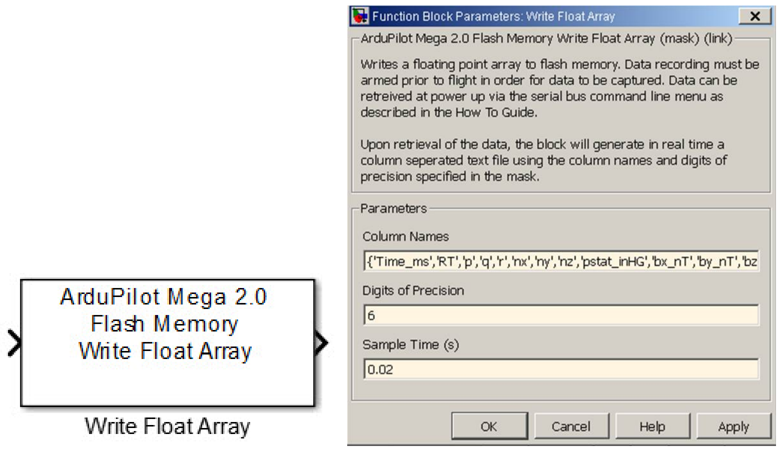
3.4.3 Flash Memory Chip

The Flash Memory block will write an array of float values to the flash memory chip that is built onto the APM2. The mask for the block allows the user to enter the names of each parameter in the array. It should be noted that the block does not compare the list of headings to the list of parameters in the array. The user should use caution to ensure that the heading names do in fact match the data being written to the flash memory chip. Additionally, within the mask, the user can set the digits of precision of the float parameters. The sample time must also be set to the overall sample time to ensure data is collected for each time step.

The flash memory block does have an output that can be useful for debugging. A display block can easily be connected for monitoring data in simulation.

Enabling the Flash Memory Chip to record data, as well as how to download recorded data, is described in Section 3.4.4

Figure 3.11 shows the Flash Memory block and mask.



ArduPilot Mega 2.0 Flash Memory Write Float Array (mask) (link)

Writes a floating point array to flash memory. Data recording must be armed prior to flight in order for data to be captured. Data can be retreived at power up via the serial bus command line menu as described in the How To Guide.

Upon retrieval of the data, the block will generate in real time a column seperated text file using the column names and digits of precision specified in the mask.

Parameters

Column Names

{'Time_ms','RT','p','q','r','nx','ny','nz','pstat_inHG','bx_nT','by_nT','bz

Digits of Precision

6

Sample Time (s)

0.02

Figure 3.11: Flash Memory block and mask

Port No.	Label	Units	Description
1	N/A	Float Array	Array of floats to be logged

Table 3.19: Write Float Array Block, Input

Port No.	Label	Units	Description
1	N/A (sim)	Float Array	Simulation debugging port

Table 3.20: Write Float Array Block, Output

3.4.4 Downloading Data

To download data that has been recorded to the Flash Memory chip, the ArduPilot must be reset. Upon a power up or reset, the first thing that the ArduPilot does is check to see if data is stored on the memory chip. A menu is displayed via the USB serial port that the user can interact with using Putty. If the memory chip does not have any data, the user is prompted to do one of the following:

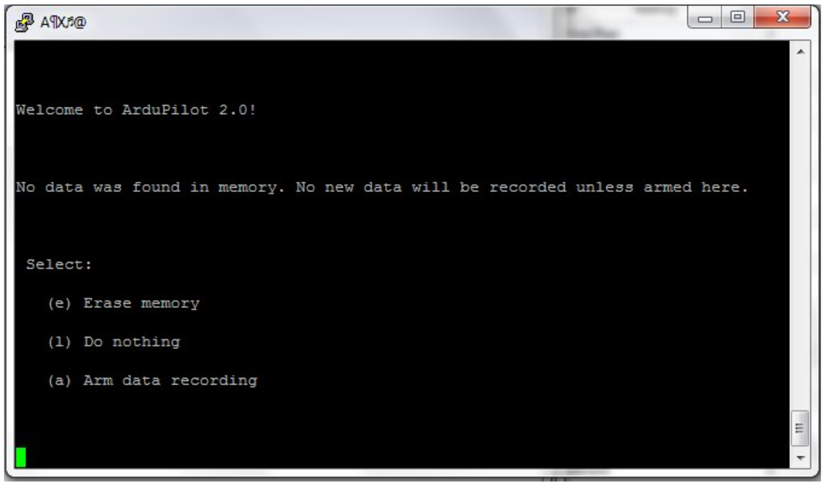
- Arm the board for recording - user presses the `a` key

- Erase the board - user presses the `e` key
- Do nothing, Leave the board alone - user presses the `l` key

If the memory chip does have data, the user prompt is slightly different. The menu is:

- Overwrite the data on the board - user presses the `o` key
- Erase the board - user presses the `e` key
- Print the Data to the Putty Terminal - user presses the `p` key
- Do nothing, Leave the board alone - user presses the `l` key

Figure 3.12 and Figure 3.13 show the putty terminal display of each of the menus described above.



```
Welcome to ArduPilot 2.0!


No data was found in memory. No new data will be recorded unless armed here.


 Select:

   (e) Erase memory

   (l) Do nothing

   (a) Arm data recording
```

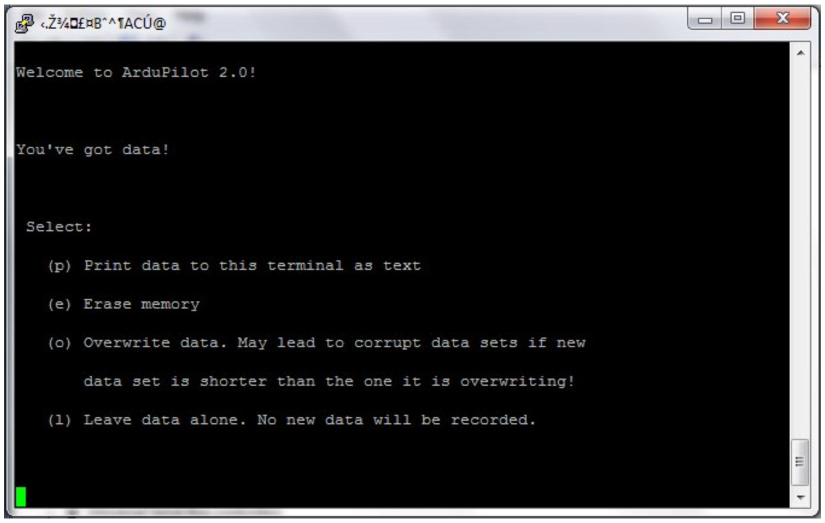
Figure 3.12: Flash Memory interaction menu - without data

3.5 Other Miscellaneous Blocks

There are several other blocks that are useful during the development of the APM2 software. While these may or may not be used in the final design, they are available, and described in the following sections.

3.5.1 Discrete Input

This block reads a discrete value from a specified pin on the Arduino board.



```
Welcome to ArduPilot 2.0!


You've got data!


 Select:

   (p) Print data to this terminal as text

   (e) Erase memory

   (o) Overwrite data. May lead to corrupt data sets if new

       data set is shorter than the one it is overwriting!

   (l) Leave data alone. No new data will be recorded.
```

Figure 3.13: Flash Memory interaction menu - with data

Port No.	Label	Units	Description
1	Pin Number	–	Outputs a 0 or 1 for low or high signal (respectively) read on the specified pin

Table 3.21: Discrete Input Block, Output

3.5.2 Discrete Output

This block sends a discrete value to a specified pin on the Arduino board.

Port No.	Label	Units	Description
1	Pin Number	–	Specified pin will be high or low if input is 1 or 0 respectively

Table 3.22: Discrete Output Block, Input

3.5.3 LED Blue

This block illuminates the blue LED.

Port No.	Label	Units	Description
1	On	–	An input of 1 turns the blue LED on and 0 turns the blue LED off

Table 3.23: LED Blue Block, Input

3.5.4 LED Yellow

This block illuminates the yellow LED.

Port No.	Label	Units	Description
1	On	–	An input of 1 turns the yellow LED on and 0 turns the yellow LED off

Table 3.24: LED Yellow Block, Input

3.5.5 LED Red

This block illuminates the red LED.

Port No.	Label	Units	Description
1	On	–	An input of 1 turns the red LED on and 0 turns the red LED off

Table 3.25: LED Red Block, Input

3.5.6 Analog Input

This block reads the analog voltage input from the specified analog input pin on the Arduino board. The range of the input on the pin is between 0 and 5 volts corresponding to a block output of 0 to 1023.

Port No.	Label	Units	Description
1	Pin Number	Integer	Outputs a value between 0 to 1023 corresponding to 0 to 5 volts

Table 3.26: Analog Input Block, Output

3.5.7 Analog Output

This block approximates an analog output on the specified pin on the Arduino board using a PWM signal with a frequency of approximately 490 Hz. The input to this block needs to be an integer between 0 and 255 corresponding to 0 to 5 volts.

Port No.	Label	Units	Description
1	Pin Number	Integer	Outputs an analog signal on specified pin between 0 and 5V corresponding to an input of 0 to 255

Table 3.27: Analog Output Block, Input

3.5.8 Serial Read

This block reads one byte from the serial interface specified in the mask. It will output -1 if no data is available to be read.

Port No.	Label	Units	Description
1	N/A	Byte	Reads one byte from the serial interface

Table 3.28: Serial Read Block, Output

3.5.9 Convert Float to uint8

This block takes a single floating point number and converts it to four bytes that can be transmitted via the Serial Write block. This is very useful for building a telemetry packet.

Port No.	Label	Units	Description
1	N/A	float	floating point value

Table 3.29: Convert Float to UINT8 Block, Input

Port No.	Label	Units	Description
1	N/A	uint8	uint8 value

Table 3.30: Convert Float to UINT8 Block, Output

Chapter 4

Examples

As of the initial release of the APM2 Simulink Blockset, the actual embedded code does not work with Reference Models. It will however work with library models. Future improvements will add the capability to use Reference Models, but for now, the user should avoid them in the model that is actually embedded to the ArduPilot. The file described in the examples below is included in the .zip file provided with this guide in the location `/Arduino_Code/ArduPilot/-APM2/Sensors_Example.mdl`.

4.1 Basic Example

The minimum functionality required to embed code onto the ArduPilot would simply be the RC Read block with its outputs connected to the RC Write block. This model will simply operate the aircraft in pure manual mode. The ArduPilot will process the pilot commands and output the equivalent PWM servo or throttle commands. Figure 4.1 shows this basic model.

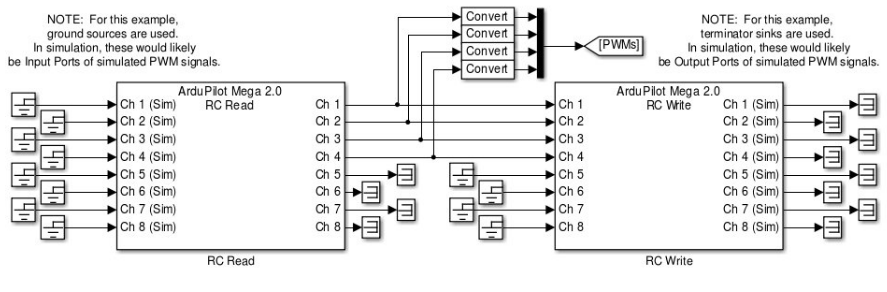

NOTE: For this example, ground sources are used. In simulation, these would likely be Input Ports of simulated PWM signals.
ArduPilot Mega 2.0 RC Read
Ch 1 (Sim)
Ch 2 (Sim)
Ch 3 (Sim)
Ch 4 (Sim)
Ch 5 (Sim)
Ch 6 (Sim)
Ch 7 (Sim)
Ch 8 (Sim)
Ch 1
Ch 2
Ch 3
Ch 4
Ch 5
Ch 6
Ch 7
Ch 8
RC Read
Convert
Convert
Convert
Convert
[PWMs]
NOTE: For this example, terminator sinks are used. In simulation, these would likely be Output Ports of simulated PWM signals.
ArduPilot Mega 2.0 RC Write
Ch 1
Ch 2
Ch 3
Ch 4
Ch 5
Ch 6
Ch 7
Ch 8
Ch 1 (Sim)
Ch 2 (Sim)
Ch 3 (Sim)
Ch 4 (Sim)
Ch 5 (Sim)
Ch 6 (Sim)
Ch 7 (Sim)
Ch 8 (Sim)
RC Write

Figure 4.1: Basic Simulink model using RC blocks

4.2 IMU and Data Output Example

To test the APM2 on the ground, the IMU block can be connected to a Serial output block for simple monitoring of linear accelerations and rotational rates. The IMU block will provide the measured values and the Serial Print Floats block will output a serial stream of data that is comma separated. A simple program to read serial data from the USB port (Putty is a popular choice) can be used to view the serial data stream in real time, as well as logging the data for future use. Figure 4.2 shows this simple model, including the timing blocks to allow for time stamped data.

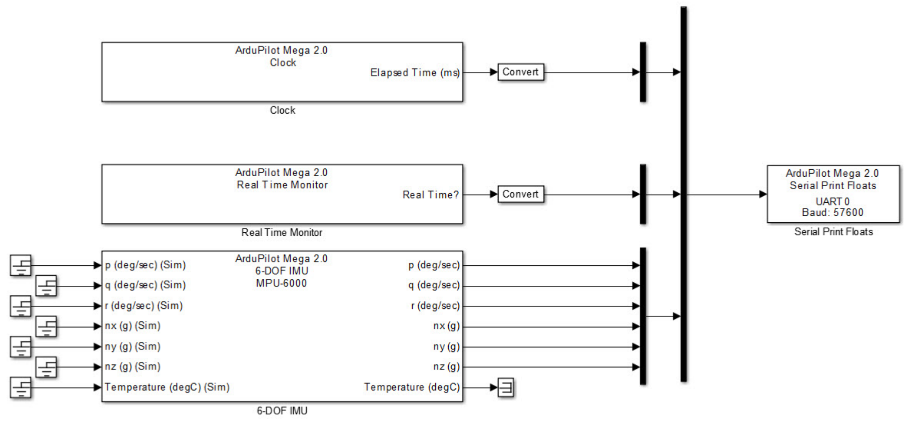


Figure 4.2: IMU measurements output to serial port

4.3 Sensors Example

A more complex example model includes the sensor and data logging blocks in addition to the RC blocks. This model will allow each component on the ArduPilot to function, with the raw sensor values, in engineering units, recorded for post processing. This data can be used to help identify the system dynamics for simulation. Figure 4.3 shows a Simulink model using all of the sensor, timing, serial outputs, and flash memory blocks. This demonstrates how arrays can be used for the flash memory block as well as the serial outpus (USB and XBee). It should be noted that the flash memory block and serial print floats blocks require a floating point value for the input, therefore a convert block is placed on all signals that are not a floating point value by default. In all total, there will be 23 parameters written to the flash memory chip with this model. Figure 4.3 does not show the basic RC blocks from Figure 4.1, but those are included in this example model.

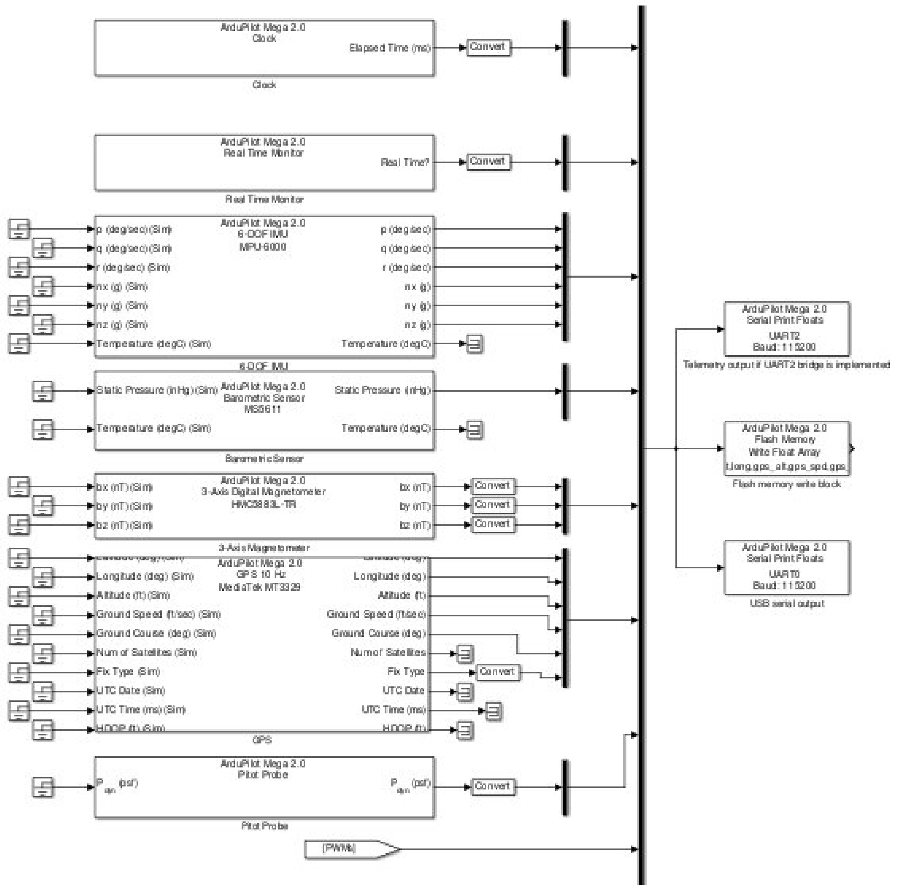

ArduPilot Mega 2.0 Clock
Elapsed Time (ms)
Convert
Clock
ArduPilot Mega 2.0 Real Time Monitor
Real Time?
Real Time Monitor
ArduPilot Mega 2.0 6-DOF IMU MPU-6000
6-DOF IMU
ArduPilot Mega 2.0 Barometric Sensor MS5611
Barometric Sensor
ArduPilot Mega 2.0 3-Axis Digital Magnetometer HMC5883L-TR
3-Axis Magnetometer
ArduPilot Mega 2.0 GPS 10 Hz MediaTek MT3329
GPS
ArduPilot Mega 2.0 Pitot Probe
Pitot Probe
[PWMs]
ArduPilot Mega 2.0 Serial Print Floats UART2 Baud: 115200
Telemetry output if UART2 bridge is implemented
ArduPilot Mega 2.0 Flash Memory Write Float Array
Flash memory write block
ArduPilot Mega 2.0 Serial Print Floats UART0 Baud: 115200
USB serial output

Figure 4.3: Simulink model using all sensor, timing, and flash memory blocks

4.4 Downloading a Model

Section 2.8 describes the full steps required to download the model to the APM2 using ROTH. Once setup, it is as simple as connecting the APM2 to the computer with a miniUSB cable, and then from the model, selecting the `Tools` menu, followed by the `Run on Target Hardware` menu, followed by the `Run` option.

4.5 Next Steps

With the above basic examples, additional Simulink subsystems or models can be included to use the sensor data for state estimation and closed loop control. This is the fun part. Good luck!

Chapter 5

Simulink Blockset Architecture

A Simulink library has been created with a set of blocks as shown in Figure 5.1 that will allow for interaction with the hardware components on the ArduPilot Mega 2.0 board via Simulink. There are blocks for accessing the inertial measurement unit, writing data to serial ports, turning on LEDs, and many other operations. Most blocks include inputs to allow them to be used in place and pass through simulated sensor data when the model is being run as part of a desktop simulation. Once the model has been validated and is deemed flight-worthy, a new feature of Simulink called Run On Target Hardware can be used to auto-generate C/C++ code, compile the code, and flash the executable to the board all via USB and all without any user interaction. When these blocks are being auto-coded, they will automatically replace their simulation inputs with calls to the real hardware. Details regarding individual blocks can be found in Section 3.

The following sections will discuss the inner workings of the blocks most of which share a similar format. In an effort to provide a guide to those who carry the development further, the next two sections are intended to outline the basic architecture of the blocks. Note that some blocks may deviate from this slightly depending on their individual implementation.

5.1 APM C/C++ Code Base

The underlying code that became the APM2 blockset was derived from three sources. First, a blockset supporting the APM1 board was obtained from Mathworks, and this was used as a guide to building its successor. Due to a major shift in hardware between the APM1 and APM2, very little code is shared with this project.

Secondly, a large majority of the firmware code was obtained from reference [1]. The task for the team then became primarily creating wrapper code

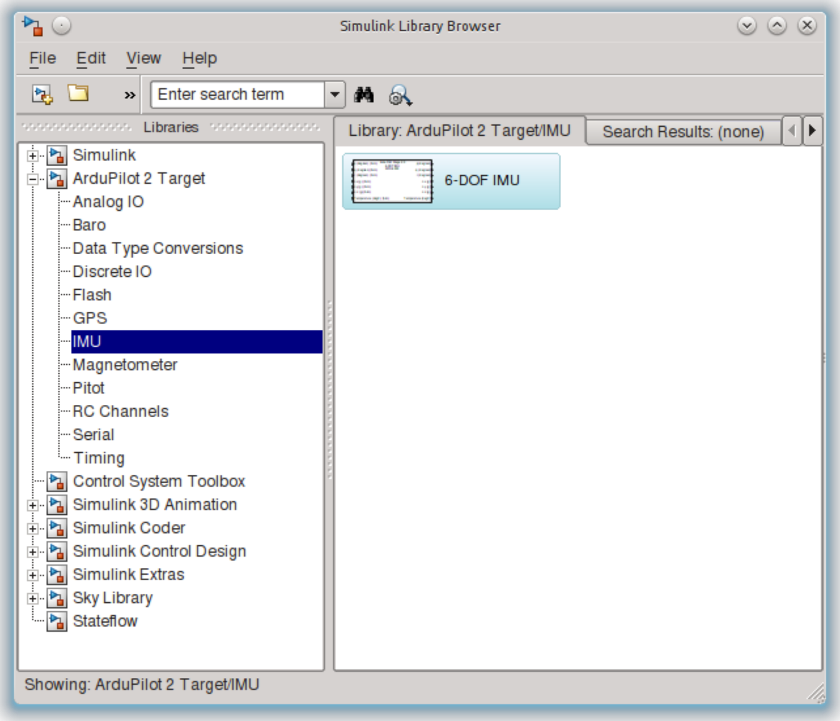


Figure 5.1: ArduPilot Mega 2.0 Simulink Library

that would link to these classes and interface with Simulink. Many cosmetic modifications were made to this code such as the removal of relative paths in the include statements, modifying signs or units, but the majority of it was left intact.

The last source for firmware code was custom generation. C++ classes such as `Pitot` and `TimeStats` were written by the team since a ready solution was not available. In addition, a significant amount of wrapper functions, primarily contained in the files `Simulink_APM2_Wrapper.h` and `Simulink_APM2_Wrapper.cpp`, were written in order to declare necessary global variables, perform system wide board start up routines, and provide an interface for Simulink to access the APM classes and hardware. Most of this C/C++ code can be found in the `cores/arduino` folder that is provided in the .zip file included with this guide.

To give an example of this set up, we shall examine the C++ code required to gain access to the IMU. Most blocks follow a similar process.

First, the function prototypes were declared in `Simulink_APM2_Wrapper.h` as,

```
/* Inertial Measurement Unit (calibrated) */
void        imu_init(uint32_t, uint32_t, uint32_t);
void        imu_flash_leds(bool);
void        imu_update(void);
void        imu_get_accels(float*);
void        imu_get_gyros(float*);
float       imu_temperature(void);
```

Next, in `Simulink_APM2_Wrapper.cpp`, an instance of the `AP_IMU_INS` class was declared,

```
AP_InertialSensor_MPU6000 ins( 53 ); /* chip select is pin 53 */
AP_IMU_INS imu(&ins);
```

And finally, wrappers were defined for each function that accessed the class methods of the `imu` variable,

```
extern "C" void        imu_init(uint32_t lpf_filt_freq_hz,
                                uint32_t gyro_scale,
                                uint32_t accel_scale)
{
  imu.init(IMU::COLD_START,
           delay,
           imu_flash_leds,
           &scheduler,
           lpf_filt_freq_hz,
           gyro_scale, accel_scale);
};
extern "C" void        imu_update(void)        {imu.update();}
extern "C" void        imu_get_accels(float* a)
{
  Vector3f accel;
  accel = imu.get_accel();
  a[0]=accel.x;
  a[1]=accel.y;
  a[2]=accel.z;
}
extern "C" void        imu_get_gyros(float* g)
{
  Vector3f gyro;
  gyro = imu.get_gyro();
  g[0]=gyro.x;
  g[1]=gyro.y;
  g[2]=gyro.z;
}
```

```
extern "C" float       imu_temperature(void)
{
  return ins.temperature();
}
```

At this point, the wrapper functions can be accessed by the Simulink code as will be discussed further in Section 5.3.

One important function to note is the initialization function, `Simulink_APM2_Startup` found in the `Simulink_APM2_Wrapper.cpp` and `Simulink_APM2_Wrapper.h` files. This must be run prior to any block code and is responsible for starting up and configuring the hardware, buses, etc. Every block in the APM2 library will check, upon code generation, to see if a call to this function has been inserted in the start up routine and will insert it if not. Again, more details will be discussed in Section 5.3.

5.2 Level 2 S-Function

Each block in the library has a special C file that corresponds to the block. Without going into too much detail as it is not entirely relevant to this project, it should be noted that this file essentially defines the behavior of the block when being used in a simulation as well as provides some information to the code generation process regarding parameters and ports. It is also important that the user first compiles these S-Functions before attempting to use any blocks or an error will be thrown. The Matlab command to compile, for example, the S-Function associated with the IMU block is `mex Arduino_IMU_sfcn.c`.

For more information, see reference [8].

5.3 Target Language Compiler

The target language compiler file, like the S-Functions described above, is another file that is associated with each block. This file contains instructions to the auto-coder as to how exactly to generate code. There is an entire language dedicated to this process and the reader is encouraged to browse reference [7] for an idea of how powerful (and slightly confusing) this set up can be.

Of paramount interest to the reader are the `BlockTypeSetup`, `Start`, and `Outputs` functions within the TLC file. This is where the C code that links the block ports to the wrappers from Section 5.1 is defined. Each function shall be briefly explained here.

The first function, `BlockTypeSetup`, is called once per block type and was used to insert any additional headers required for that particular block type. In general, all blocks simply ensure that the global header, `Simulink_APM2_Wrapper.h`, has been declared.

The second function, `Start`, inserts code in the function that the compiled program will run prior to starting and running the actual Simulink model code.

This is where additional resources are started such as a serial bus necessary for a specific block operation, or calls to calibrate sensors are placed. This enables, for example, the board to execute its IMU calibration only when the IMU block is actually in the model since the TLC for the IMU block uses the `Start` function to insert a call to `imu_init`. The TLC also has access to mask parameters and can insert their values into the generated C code for operations such as setting the baud rate or selecting the IMU sensor ranges.

The first operation of each and every `Start` function is to check and see if any other block have called `Simulink_APM2_Startup` and insert the call if necessary. Thus, the following TLC code is common to all APM2 blocks,

```
%% Start up APM2 board. Common to all APM2 blocks. Must be
%% included FIRST!
  %if EXISTS("_DONE_SYSTEM_INIT_") == 0
     %assign :: _DONE_SYSTEM_INIT_ = 1
     /* APM2 System startup function. Required for any blocks.
        Inserted by:
        %<Type> (%<ParamSettings.FunctionName>): %<Name> */
     Simulink_APM2_Startup();
  %endif
```

where all % <> tokens will be replaced with the appropriate values during code generation. For instance, the C code generated from the above TLC code is the following assuming that no previous blocks have inserted this call:

```
/* APM2 System startup function. Required for any blocks.
   Inserted by:
   Block (6-DOF IMU): 6-DOF IMU */
   Simulink_APM2_Startup();
```

Finally, the `Output` function generates the code that is called by the embedded program each time the block is executed. This is where the the majority of the code resides for most blocks. Again, looking at the IMU, the TLC file that generates the C code is shown below,

```
%% Function: Outputs ===========================================
%%
%function Outputs(block, system) Output

  %if (CompiledModel.TargetStyle!="SimulationTarget")

    float imu_accel[3];
    float imu_gyro[3];
    float imu_temp;

    /* Stop the baro chip from holding the SPI bus */
    pinMode(40, OUTPUT);
    digitalWrite(40, HIGH);

    imu_update();
    imu_get_gyros(imu_gyro);
    imu_get_accels(imu_accel);
    imu_temp = imu_temperature();

    /* Degrees per second */
    %<LibBlockOutputSignal(0, "", "", 0)> = imu_gyro[0];
    %<LibBlockOutputSignal(1, "", "", 0)> = imu_gyro[1];
    %<LibBlockOutputSignal(2, "", "", 0)> = imu_gyro[2];

    /* G's */
    %<LibBlockOutputSignal(3, "", "", 0)> = imu_accel[0];
    %<LibBlockOutputSignal(4, "", "", 0)> = imu_accel[1];
    %<LibBlockOutputSignal(5, "", "", 0)> = -imu_accel[2];

    /* Deg C */
    %<LibBlockOutputSignal(6, "", "", 0)> = imu_temp;
  %endif

%endfunction
```

Keep in mind that there is a TLC file for each block, and C code generated using the TLC above will be inserted into one large update function in the final stages of the auto-code process. The order in which Simulink Embedded Coder inserts the code is variable and code must be written so that it can be called at any time. Functions that must take precedent, for example, initializing the shared peripheral buses before devices try to access them are handled by the `Simulink_APM2_Startup` function as described earlier.

Chapter 6

Future Work

6.1 Future Work

This blockset for Simulink was created for a specific purpose. The robustness for variable uses has not been implemented yet. There are goals to improve this blockset by the original creators, but as more people use it, it is expected that some of them will be able to implement improvements. The following list identifies some known issues or areas for improvement.

- Currently, when using ROTH in Simulink, it does not appear to always allow the use of Reference Models. This feature is highly desirable, however time constraints did not allow it to be implemented.
- The Flash Memory block currently only allows float data types to be used. While this does work, it uses more memory than is really required for some terms. The goal will be to develop the code so that multiple types on a single array or bus can be written to the Flash Chip.
- The heading names for the Flash Memory block currently need to be manually entered in the mask. It would be nice if the names were automatically determined from the signal names of data coming into the block. This of course would require signals to be named.
- If the XBee's firmware is modified using XCTU from Digi, then the menu that is displayed to arm recording or download data will not display properly. This is because the file `Simulink_APM2_Wrapper.cpp` in the modified cores directory has the baud rate for displaying the memory menu hard coded to 115200 baud. Ideally, this term would be able to be set within Simulink or the Matlab workspace.
- Modify the library blocks so that in simulation, they can be used to represent the actual sensors. Currently they simply pass any inputs through directly to the outputs in simulation, but it would be useful if any noise,

variance, or drift effects could be modeled using the block mask. These mask settings would only be used in simulation, not when actually downloaded to the board.

- Develop automation to perform the setup steps described in Chapter 2, including defining the path.
- Develop properly formatted Simulink Help files. Currently, this How To Guide is the best reference for help with the blockset.

Chapter 7

Support

This guide is intended to provide all of the details necessary for utilizing the ArduPilot 2.0 Simulink blockset. However, there will inevitably be issues that arise as a result of many unknowns as more people begin to use the blockset. Please email `apm2.simulink@gmail.com` if support is required to properly use the APM2 Simulink blockset. All reasonable attempts will be made to support the users of this blockset.

References

[1] DIY Drones. *Source Checkout - ardupilot-mega - Official ArduPlane repository - Google Project Hosting:* Oct. 2012. URL: http://code.google.com/p/ardupilot-mega/source/checkout.

[2] DIY Drones. *Using Xbee radios for telemetry with APM 2*. Oct. 2012. URL: http://code.google.com/p/ardupilot-mega/wiki/APM2Wireless.

[3] Freescale. *MPXV7002: Integrated Pressure Sensor*. Oct. 2012. URL: http://www.freescale.com/webapp/sps/site/prod_summary.jsp?code=MPXV7002&fsrch=1&sr=1.

[4] Honeywell. *3-Axis Digital Compass IC - HMC5843*. Oct. 2012. URL: http://www51.honeywell.com/aero/common/documents/myaerospacecatalog-documents/Defense_Brochures-documents/HMC5843.pdf.

[5] InvenSense. *MEMS Gyro-Accel — Gyroscope — Accelerometer — Processing - MPU-6000*. Oct. 2012. URL: http://invensense.com/mems/gyro/mpu6000.html.

[6] The Mathworks. *Supported and Compatible Compilers Release 2012a*. Oct. 2012. URL: http://www.mathworks.com/support/compilers/R2012a/win64.html.

[7] The Mathworks. *Target Language Compiler*. Oct. 2012. URL: http://www.mathworks.com/help/rtw/block-authoring-with-tlc.html.

[8] The Mathworks. *What Is an S-Function?* Oct. 2012. URL: http://www.mathworks.com/help/simulink/sfg/what-is-an-s-function.html.

[9] MediaTek. *MediaTek - MT3329*. Oct. 2012. URL: http://www.mediatek.com/en/Products/product_content.php?sn=50.

[10] NOAA. *NOAA's Geophysical Data Center - Geomagnetic Online Calculator*. Oct. 2012. URL: http://www.ngdc.noaa.gov/geomagmodels/IGRFWMM.jsp.

[11] MEAS Switzerland. *MS5611-01BA01Pressure SensorsAltimeter Pressure Sensor ModulesMeasurement Specialties:* Oct. 2012. URL: http://www.meas-spec.com/product/t_product.aspx?id=8501.