6.111 Final Project Report

Active Stabilization 3-Axis Sensor Platform

Scott Torborg
Kyle Vogt
Mike Scharfstein

Spring 2005
TA: Chris Forker

Abstract

Autonomous robotic platforms require input from a wide variety of sensors. In order to be effective in rough and demanding environments, these sensors must be stabilized to produce efficient output. To do this, high-level sensors are mounted on a 3-axis gimbal platform controlled by a correction system. The system uses accelerometers, rate gyroscopes, and feedback from motors to sense the actual position of each axis, and then correct for any changes by controlling brushless motor amplifiers that are used to move each axis. A finite-impulse response filter allows the system to correct for motion smoothly and effectively, and take advantage of multiple sensors.

Introduction and Lab Kit Overview

DARPA's Grand Challenge competition is a 200-mile race across the California desert. The race is entirely autonomous, and employs cutting edge algorithms and sensor technology. The MIT Grand Challenge vehicle will primarily use two sensors: a LIDAR (scanning laser rangefinder) unit and a stereo camera pair. In order to be usable, both of these sensors require much more stability than a truck suspension can offer. To provide this, the team seeks to build an actively stabilized three axis gimbal sensor platform. This platform will be controlled by a low-latency control loop implemented in programmable logic.

The system uses accelerometers, rate gyroscopes, and quadrature encoder feedback from motors to sense the actual position of each axis, both in reference to the base of the platform and to the ground. The inertial data is passed through a pair of FIR filters and then combined with the encoder data in an absolute control block. This data is output to a DAC which controls brushless motor amplifiers for each axis.

For maximum flexibility, the filter is completely reconfigurable on the fly, and most configuration parameters of the system can be easily modified.

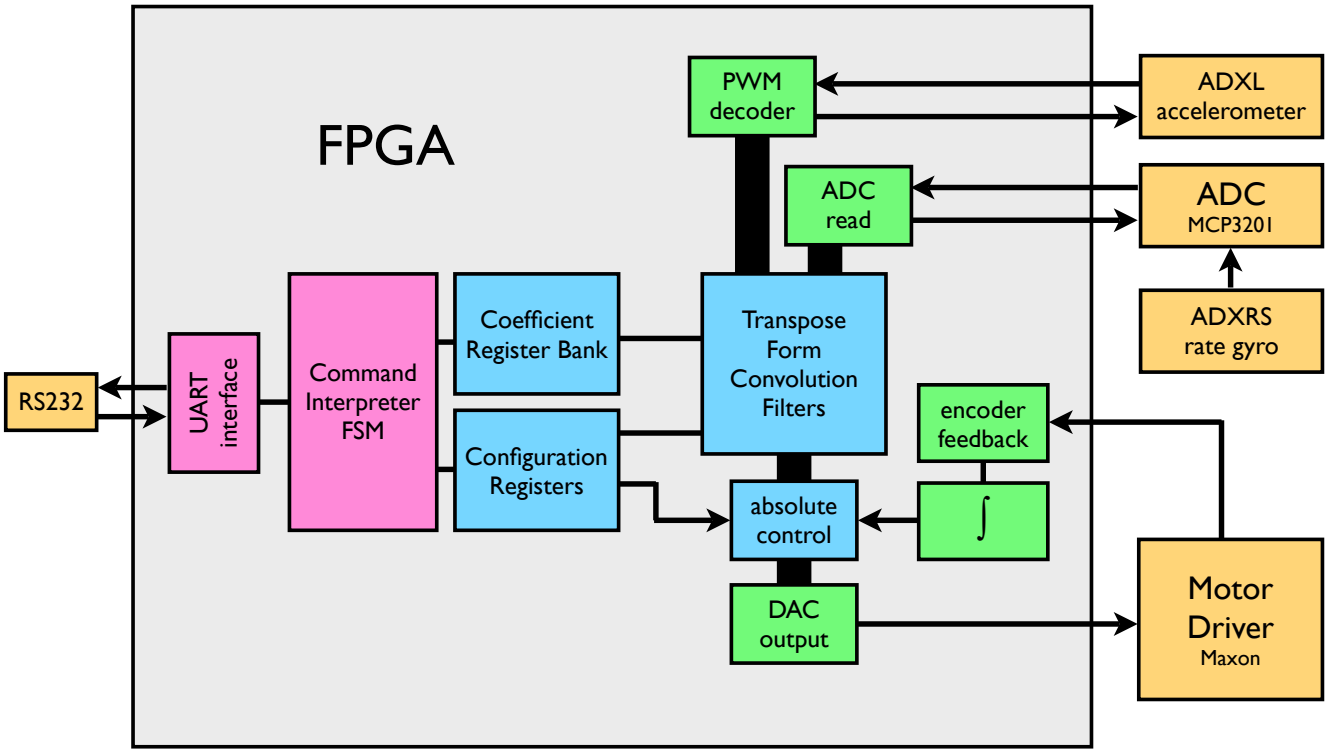
Hardware

The axes of the sensor platform are actuated by 120 watt Maxon EC brushless motors. Each of these motors is powered by a Maxon DES digital servo amplifier. On each axis is mounted an Analog Devices ADXL213EB evaluation board and ADXRS300EB evaluation board.

On the lab kit breadboards there are several chips: Analog Devices AD558 DACs for output voltage control, Microchip MCP3201 ADCs for sensor input. There are also various passives for power supply decoupling, filtering, and attenuation.

Module Description and Implementation

The stabilization system is divided into three parts: the high-level control and interface blocks, the configuration and feedback blocks, and the I/O blocks for sensors and actuators. Because there is no shared data between the different axes, and the control system can be identical across the axes, the overall stabilization system can be divided into three instances of the same 'axis' module. Each instance of the feedback system handles only its own axis.



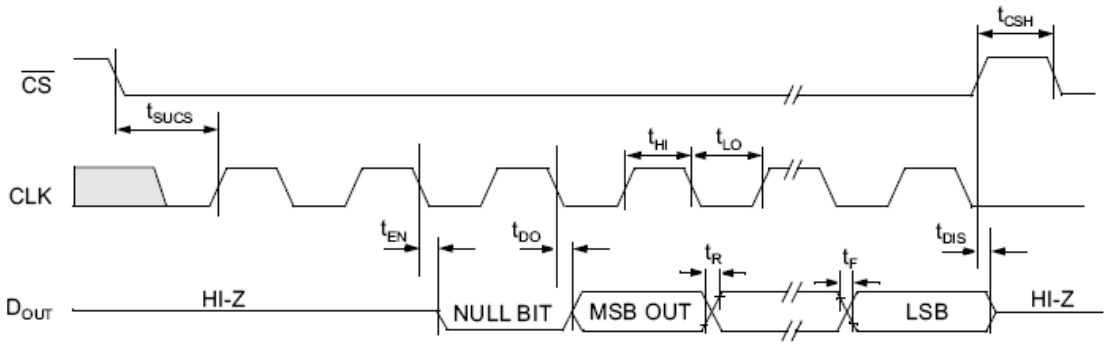
Block Diagram Overview of one axis of the Stabilization System

Sensor and Actuator Input/Output Blocks - Mike Scharfstein

Gyro ADC Input FSM

We used the ADXRS300 rate gyroscope from Analog Devices to help us determines the rate at which each axis is rotating. The gyroscopes are rated for sensing angular speeds of up to 300 degrees per second, and output an analog signal between 0V and 5V accordingly. The signal is offset so that 2.5V means an angular speed of 0 degrees per second.

Since the gyroscopes have such a wide range of operation, we decided to use 12-bit ADCs to convert their output into a digital signal. The ADCs that we settled on are the Microchip MCP3201, which use the SPI protocol, to transfer digital data. SPI is a serial protocol, which involves the master device (the labkit in our case) sending the slave (the MCP3201) a clock signal, and a chip select signal, while the slave pumps data out serially through a third, data line. A simplified timing diagram and specifications can be found below. Using a serial ADC also made wiring up our design much simpler. Only running 3 wires to the ADC, instead of over 12 was very convenient, and prevented a lot of tangling hardware debugging.



Name	Description	Min	Max
T_{SUCS}	CS fall to first riding clock edge	100ns	N/A
T_{CSH}	CS disable time (time needed to get ready for next conversion)	625ns	N/A
T_{HI}	CLK high time	312ns	N/A
T_{LO}	CLK low time	312ns	N/A

Simplified Timing Diagram for MCP3201, courtesy of Microchip

Accelerometer PWM Decoder

The ADXL213 two channel accelerometers we used are capable of sensing acceleration in two perpendicular axes parallel to the plane of the chip with a range of +/- 1.2G. This information is then conveyed using a pulse width modulation (PWM) signal. This signal is essentially a clock signal with a variable duty cycle. The duration of the high phase of the cycle compared to the clock period represents the output value. Again, this sensor outputs its data with an offset so that a 0% duty cycle means -1.2G, 50% duty cycles means 0G, and 100% duty cycle means 1.2G. Again, only having to run a few wires from the sensor, instead of one for each bit of resolution greatly simplified our hardware.

We could have decoded this signal by having it charge a capacitor, and sampling the capacitor's charge with and ADC, but we decided to make our own PWM decoder on the FPGA since it would give us a more accurate signal. We configured the sensor to have a 1 millisecond clock period, so that we would have no trouble getting 12-bit resolution out of the sensor.

The accelerometer's data was used to orient each axis of our gimbal to earth. Using a ratio between the acceleration sensed on each axis of the accelerometer, we are able to find out at what angle to ground the axis rotated.

Motor Encoder Block

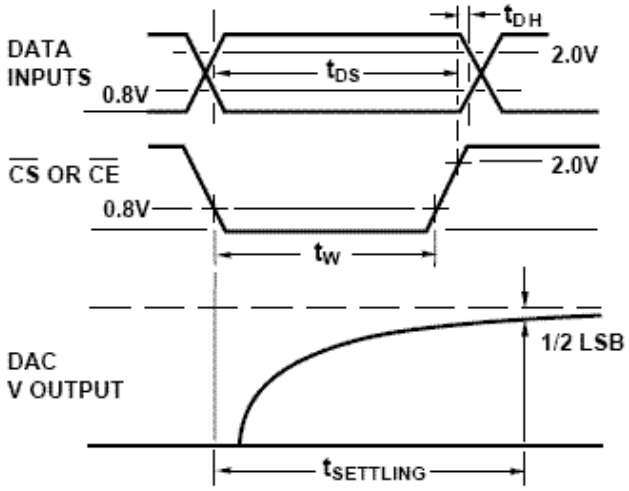
Another signal that was used as an input was the encoder signal from each of our motors. In contrast with the rate gyroscope and accelerometer, which were used to tell how the axis they are on has moved, the encoder was used to tell us how much the motor actually moved. The encoders are rated at 500 counts per revolution, and when you multiply that by the 74:1 gear head we used, that comes out to 37,000 counts per revolution of the motor, which is almost 16-bit resolution.

The signal coming from the encoder consists of two channels, each of which is a square wave, with each period representing one count. The reason that two channels are needed is to find the direction of rotation. Since each channel is situated 90 degrees out of phase from the other, we can find the direction of rotation simply by looking at which channel leads the other.

After decoding these signals, we output a pulse representing a “count,” and a “direction” bit to an integrator unit. This unit is responsible for keeping track of the position at which the motor currently is. It also has the ability to set the count to zero when the system is being calibrated.

DAC Output FSM

We used AD558 DACs to convert our output signal into a digital one that the amplifiers used to control the motors’ speed. The design of this module was similar to the one used in Lab 3, but instead of controlling this FSM with some Major FSM, it ran continuously. In order to run continuously, the output FSM had to have an internal pipeline stage to ensure that the data that needed to be outputted stayed constant as long as the DAC was enabled.



AD558 Timing Diagram, courtesy of Analog Devices

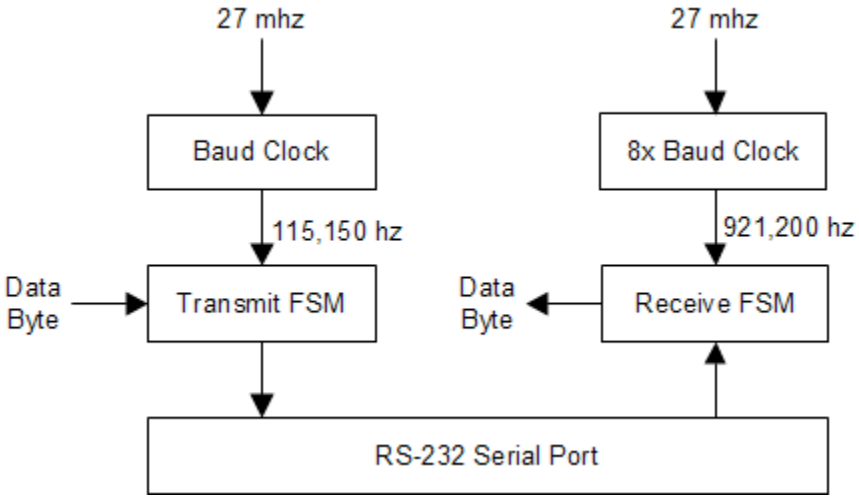
All of the I/O blocks are separated from the rest of the system by pipeline stages. This allows the sensors to run at their own maximum speeds, while the rest of the system is free to run at a slower speed while still getting the most up-to-date sensor information.

Host Interface and Command Interpreter - Kyle Vogt

UART Interface

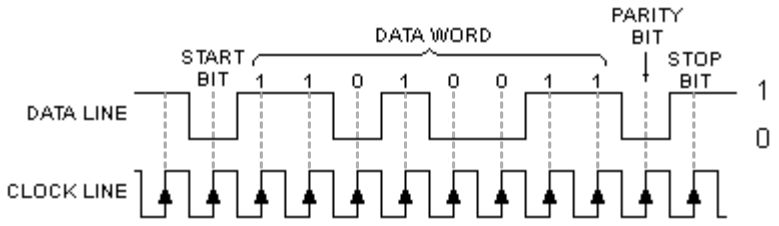
Although it’s entirely possible for a stabilization system to operate independently, it is very useful to be able to tune filter parameters and override the system remotely. To facilitate this, an RS-232 interface is integrated into the stabilizer design. The abilities enabled with this approach include absolute position control, tuning filter parameters and coefficients, and sending enable or disable commands to each of the three axis.

There are two parts to a RS-232 based UART. A transmitter module was written to take a byte wide input, serialize it, and pass it out on the transmit pin at the correct baud rate. The receiver module is clocked at the correct baud rate and oversamples the input eight times in order to eliminate bit errors due to glitches or noise from cabling. There are also two separate modules generating baud clocks for the transmitter and receiver modules.



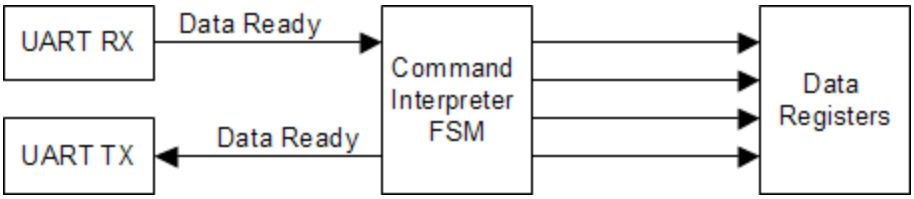
To sync with the host computer's serial clock, the stabilizer's clock must be divided down to 115,200 hertz. Since the 27mhz clock used for the stabilizer's FPGA does not divide to 115,200 using integers, a simple counter was used to generate a one-cycle baud clock pulse about 115,150 times per second. This leads to a baud rate error of about 0.04%, which is well within the specified allowable baud rater error for the RS-232 protocol.

The transmitter module has a simple FSM to serialize the supplied data byte. The state is advanced on every baud clock high pulse, so that a new bit is sent at the proper time. There is also a start bit and stop bit added to each data byte before it is sent out as a single packet. The receiver module works in a similar manner, but with a different clock. The clock is run eight times faster so the input can be oversampled. The data is filtered and registered to make sure that noise and glitches are not treated as data. The bits are then shifted into an output byte and sent on to the command interpreter.



Command Interpreter

Once data is deserialized by the RS-232 UART module, it is ready to be processed. The command interpreter FSM simply waits for an incoming command byte, then for a data byte. Once it has these two bytes it updates the contents of the selected registers as specified.



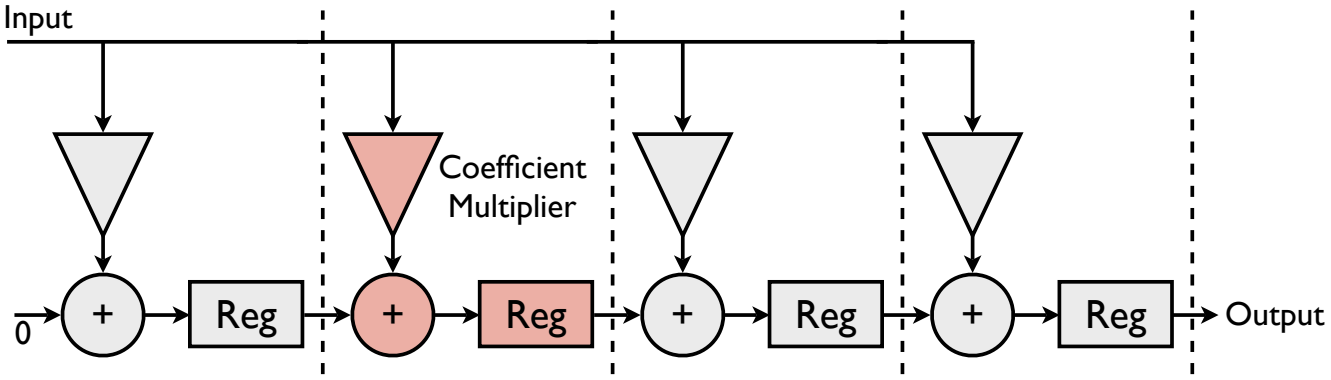
The FSM is run every time the UART receiver module flags that a new byte is ready. When this flag goes high, the command interpreter reads the contents of the reciever's data register and proceeds to the next state. The final state of the FSM loads the transmit module with a reply byte, which reports either success or failure, and triggers the UART for transmitting.

Feedback Filter and Absolute Controller - Scott Torborg

Transpose FIR Filter

There were two significant goals in the design of the FIR filter: low latency and high resolution. To make generating coefficients easier, a straight convolution was preferred. For this reason, both lattice structures and cascade structures were eliminated. To maximize performance for large filters, a transpose form filter structure was chosen. This eliminates the adder delay associated with a direct form filter, by spreading the cascaded adders (required for each fiter tap) from one pipeline stage to many pipeline stages.

The filter then uses a set of transpose filter tap blocks, each of which has a very short critical path. To further reduce the critical path, hardware multipliers (built into the Virtex II FPGA) are used. This critical path, consisting of only a hardware multiplier and adder, should enable the filter to be run at a very high clock speed (several hundred megahertz).



Transpose Form FIR Filter Structure, with one pipeline stage highlighted.

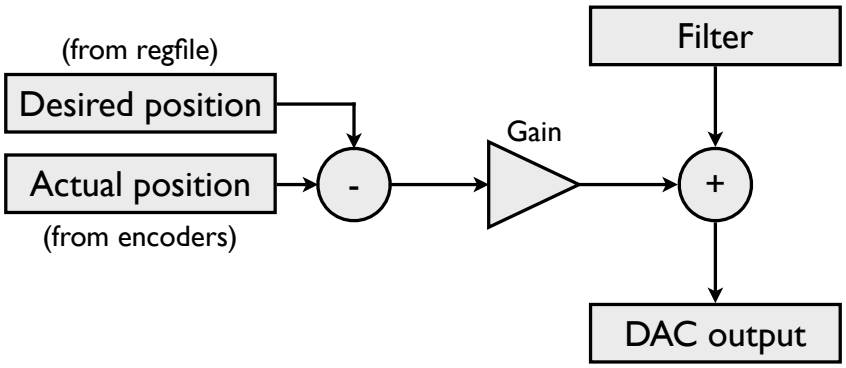
In order for the filter to be useful, the coefficients must be carefully tuned for the specific application. It is likely that this tuning will happen on the fly and in the field, so the filter coefficients are adjustable. To simplify the design and increase performance, a register bank is used instead of an SRAM block.

The repetitive nature of the code required for this Verilog block may lead to errors. To counteract this, a script is used to generate the Verilog source code for an arbitrary large filter

of this form, using a transpose_tap module as a building block. The script and example output code are in the appendix.

Absolute Control Block

The absolute control block allows an external system (for example, a navigation computer or a sensor controller) to give high-level control commands to the sensor platform and change the direction of the platform, without completely overriding the stabilization function of the system. It is set up as a simple negative feedback loop with adjustable gain and adjustable desired position.



Overview of the absolute control block.

The control inputs to the absolute control are configuration registers, and the encoder value comes from the encoder integrator input block.

Testing and Debugging

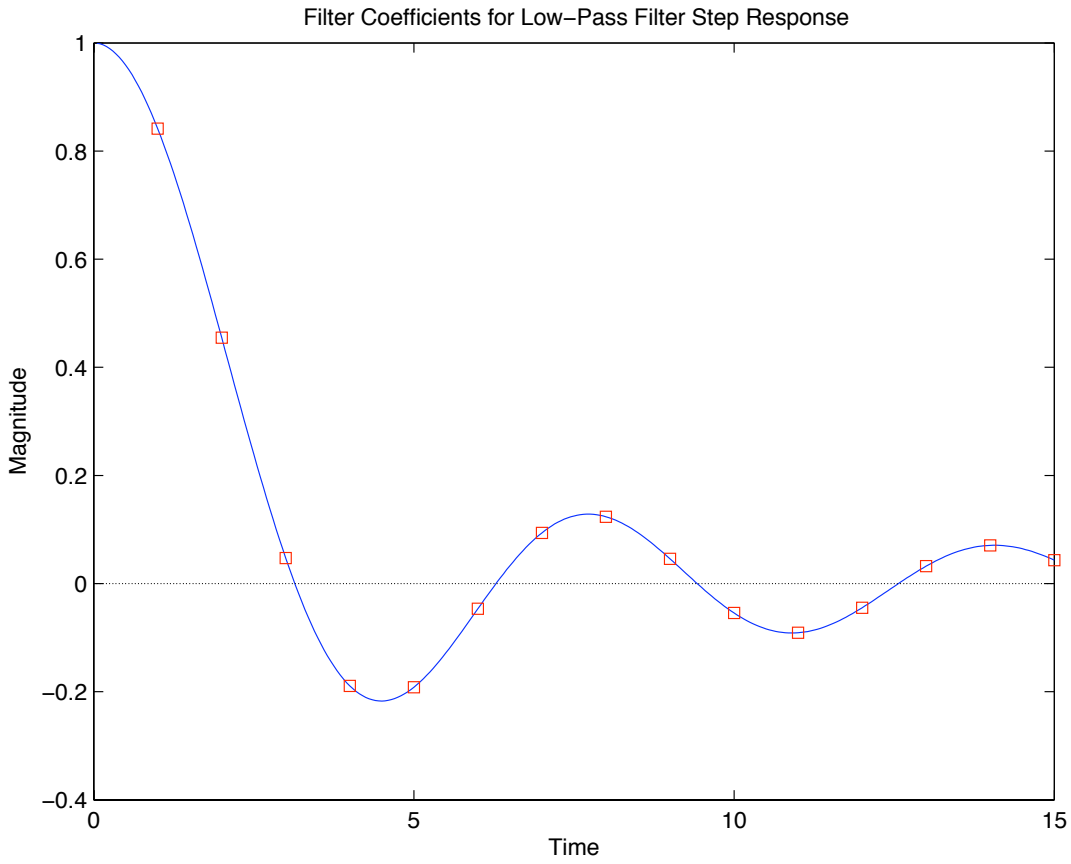
To test the receiver module, it was simulated by feeding random bits into the serial line input. Once data started to appear in the data register, it could be tested with an actual serial port. The lab kit's LED's were continuously assigned to the receiver module's data register. A HyperTerminal connection was then opened on the host computer. By pressing different keys on the keyboard and observing which LED's lit up, the module's functionality could be confirmed.

The transmitter module is tested in nearly the same way, but in reverse. Data is sent from the lab kit's switches to the host computer where the proper data bytes could be seen in the HyperTerminal window.

The command interpreter was tested by simply assigning the contents of a particular data register to LED's and using a HyperTerminal connection to send the proper test data.

To test the I/O blocks, we used a two-step process. First, each module was tested by itself with ModelSim. This was relatively easy to do, since each input block was independent from the rest of the system. Second, and integration test was performed in hardware. The input blocks were integrated with the sensors and DACs to make sure that a simple feed-through loop functioned correctly. Each module passed if the output signal shadowed the input signal.

To facilitate testing of the FIR filter, a programmatic approach was used. First, the desired filter shape was created in Matlab. This filter shape was then inverse fourier transformed and quantized at the right granularity to produce a set of coefficients (shown as red squares on the plot below).



These coefficients were then convolved with some test input to produce a sample output. The simulated output of the filter for the same test input could then be verified against the sample output from Matlab. This testing was done using the Verilog $display and $monitor commands rather than the graphical waveform output; with text, the results for a complex numeric system can be checked much more easily. This allowed the filter to be verified fairly effectively, although a more thorough examination could model the quantization error resulting from lower-order bits being discarded in the FPGA.

Conclusion

The design and construction of a three axis sensor stabilization system proved to be an exciting yet difficult challenge. The electronic components, sensors, and motor electronics were not available near the beginning of the project, and some major modules were not completed early enough, so there was very little time available to integrate the system and test it on actual hardware. Therefore, it was not possible to implement all the features that were initially planned, including the command interface and host interface (UART and CAN).

The end result of the project is a functional two axis stabilizer. It operates with much of the desired functionality, although it has a few bugs that could be corrected by further integrating the completed code modules that have already been written and tested.

We discovered that the simple filter coefficients that we initially tried were not effective in counteracting large angular displacements. The system showed the symptoms of an under damped oscillator, and eventually the system oscillated out of control and the motor amplifiers went into current limiting mode. Tuning the filter more carefully on the complete system and controlling timing more effectively would probably help to solve these problems. The late construction of the gimbal hardware prevented us from spending as much time tuning the system as we would have liked.

This project was a great way to become familiar with the new lab kit and test out some of its features. We used most of the available breadboard space and took advantage of the expansion ports and logic analyzer. The gimbal hardware was the limiting factor in the full implementation of the system, but it demonstrated most of the desired stabilization functionality with two axes.

Acknowledgments

Thanks to **Analog Devices** and **Maxon Motor AG** for giving us free parts on short notice.