UNIVERSITY OF UTAH

SENIOR PROJECT

Project Levitate

Author:
Leif Andersen
Daniel Blakemore
Jon Parker

Supervisor:
Al Davis

December 21, 2012

CONTENTS

I. Introduction

Project Levitate is a quadrotor designed to be capable of autonomous flight and visual object tracking. Quadrotors are multi-rotor aircraft with four vertical rotors (two pairs of counter rotating blades) equidistant from a center point in a ”plus sign” configuration. These aircraft tend to be more stable and easier to maneuver than traditional single rotor helicopters from a mechanical standpoint because they do not require an articulated rotor system to change pitch or roll. On single-rotor helicopters, the angle of the rotor relative to the ground must be adjusted to control speed and direction. Additionally, traditional helicopters need a tail rotor on an orthogonal axis to stabilize against the torque of the main rotor. Torque from the main rotor spins the aircraft opposite from the direction the rotor spins. With four rotors, two rotors rotate clockwise and the other two rotate counter-clockwise. If the quadrotor begins to spin in one direction, increasing the speed of the two rotors that spin in the same direction will cancel out the torque on the aircraft.

All of the computing and sensing hardware and software is included onboard the quadrotor, without any need to offload processing tasks to an external computer. The complete design includes a camera with very basic image recognition software to track objects tagged with a recognizable pattern similar to a barcode. However, due to complications with inadequate FGPA hardware (the other sensors already utilized over 90% of the fabric), this element could not be included and was demonstrated separately. The onboard hardware and software also includes a radio for remote control, flight adjustments, and debugging printouts. The quadrotor is a redesign of a quadrotor built for a prior project, with the power distribution board and individual chips being the only carryover in the final product.

II. Functional Specification

The Project Levitate quadrotor was designed to autonomously track and follow objects based on the input from a forward-facing camera. In addition, it was designed to avoid collisions with objects in its environment while tracking its target. Once airborne, Project Levitate was intended to find a specific pattern tagged to a moving person and follow that person from a safe distance. In order to accomplish this task, the platform needed to first be capable of stable flight, then of avoiding collisions, and finally of perceiving and tracking a target.

A. *Stabilization*

The most basic functionality required of an autonomous aerial vehicle is stable flight. This can range from maintaining orientation in one or more axes to staying relatively still in a specified point in space. The Project Levitate platform required 3-axis orientation stabilization meaning that roll, pitch, and yaw would be maintained without any external input. This is accomplished with data from a 3-axis accelerometer, a 3-axis gyroscope, and a 3-axis compass. From this starting point, the object tracking software can provide movement commands which augment the stabilized motor control signals to move the quadrotor with the target. This kind of stabilization does not prevent the quadrotor from drifting at a near constant velocity in any direction. As a result, the quadrotor would not be entirely ready for unsupervised flights with only this component implemented. It would first need the ability to avoid crashing into things.

B. *Collision Avoidance*

In order to prevent crashing into and damaging people or property, an autonomous quadrotor needs to be able to perceive its surroundings and gather information about the proximity of objects. Project Levitate incorporates several ultrasonic rangefinders for this purpose. Four are facing outward laterally, one on each of the corners of the chassis below the motors, one is upward facing, and one is downward facing. This configuration is engineered for indoor flight where both altitude and nearness to the ceiling must be considered. These rangefinders give the quadrotor information about objects in every direction and allow the flight software to move the platform away from anything deemed too close for comfort. This system, combined with stabilization would result in a platform which could be set loose in a room and would drift from one wall to another, ”bouncing” like a pinball as it changed direction to avoid a collision. As interesting as a quadrotor-based pong game would be, the full specification of project levitate requires more than autonomous drifting.

C. Object Tracking

The final piece of the system is the ability to track and follow objects tagged with a pattern. A simple image-based pattern recognition algorithm is used, combined with a camera to find the desired target and follow it. Once the pattern (like a QR Code) is found in the image, the size and position of the pattern relative to the frame of the image encodes the required movement command: pattern on the right or left means go in the corresponding direction; pattern at top means go up; small pattern means go forward; and big pattern means hold steady. Once data is brought in from the camera, the onboard flight software can process that image data and decide the next movement adjustment. This system, in concert with the collision avoidance and stabilization components, make up a complete system capable of following a specified target.

III. Components

The hardware components of the project consist of:

- A custom quadrotor chassis,
- One SmartFusion Evaluation Kit,
- Six rangefinders,
- One inertial sensor stick,
- One camera,
- One radio

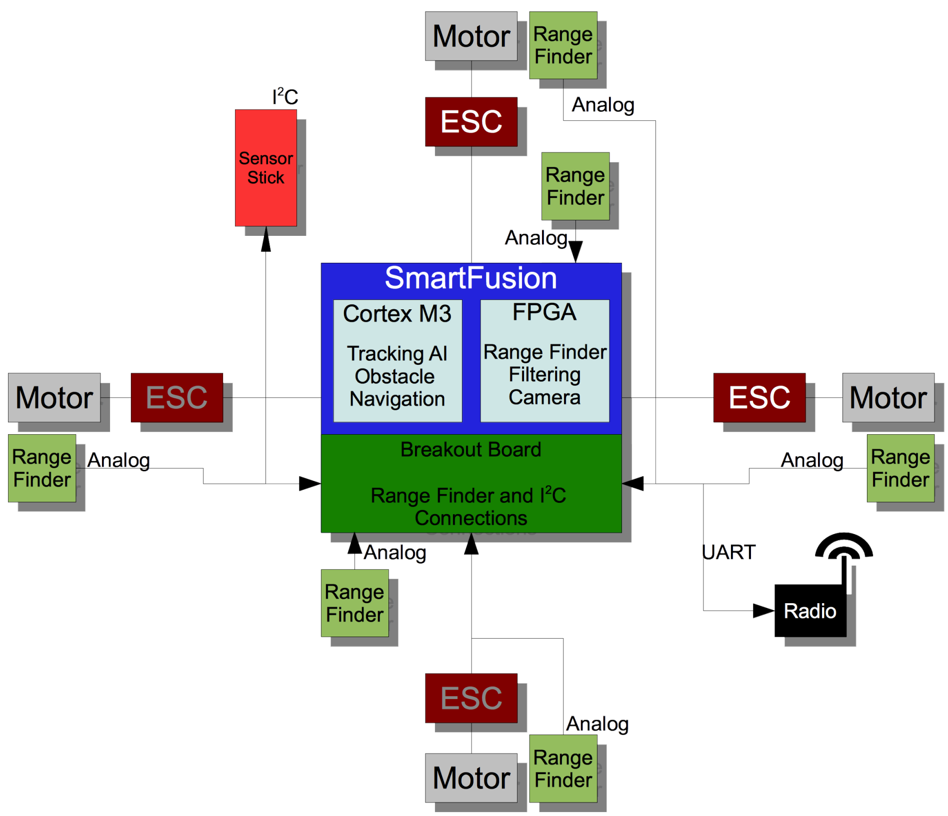


Fig. 1: General hardware block diagram.

A. Quadrotor Chassis

This quadrotor is custom built and based on a Turnigy Talon frame. Mounted to the frame are four brushless DC motors with eight-inch propellers. Each rotor has an approximate thrust of 400g at maximum speed, for a combined maximum platform thrust of 1.6kg. In order to ensure proper power/thrust headroom for maneuverability, the motors must run at around 50% of their top speed when the quadrotor is hovering; this means the ideal weight of the platform and electronics is 800g. At maximum speed, the power draw of these motors for 1600g of thrust is 360W (draw from the other electronics is negligible at roughly 165mW), and they are powered by a 2200mAh 11.1V battery that provides up to 66A of continuous current. This gives an estimated flight time of five to ten minutes, which is enough for a short demonstration of functionality. Figure 2 displays a preliminary version of the quadrotor:

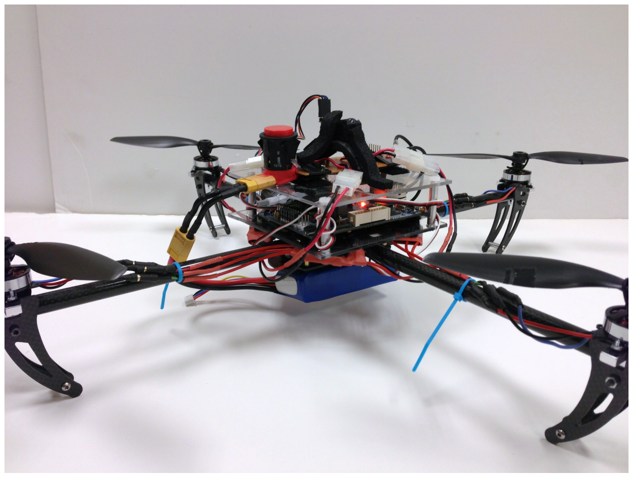


Fig. 2: Preliminary quadrotor platform.

B. *SmartFusion Evaluation Kit*

The SmartFusion Evaluation Kit is a microcontroller unit consisting of an ARM Cortex-M3, and a 200K-gate (4608 LUT) FPGA[1], and an Analog Compute Engine (ACE) with several DACs and ADCs placed on an evaluation board with various other peripherals. The SmartFusion serves as the central controller to read sensors and run software for the quadrotor. The SmartFusion was chosen for its ability to run C code and interact with custom hardware on the FPGA specified in verilog. Our system is designed to benefit from the ability to complete some computations in hardware and simultaneously do others in software. The SmartFusion has 512KB of nonvolatile code memory, 64KB of SRAM, and 4.6KB of FPGA block RAM.

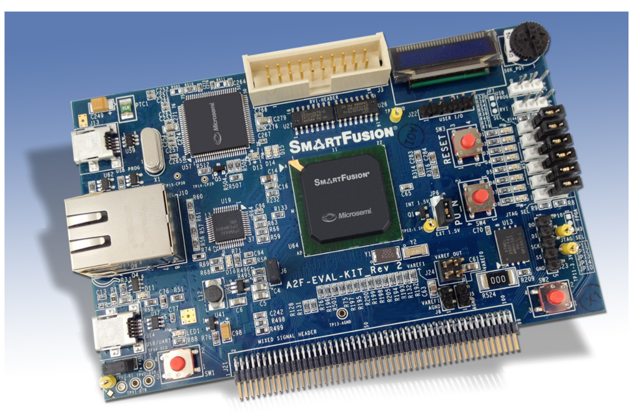


Fig. 3: SmartFusion Evaluation Kit.

[1]This rather small FPGA is the primary reason for our delays. We did not fully comprehend how small 200,000 gates actually is.

C. *Rangefinders*

Maxbotix LV-EZ1 ultrasonic range finders are used to detect proximity to objects in the quadrotor's surroundings. These rangefinders were chosen for their price, and their multiple output formats[2]. The SmartFusion FPGA makes interfacing with analog simple, and requires the allocation of fewer I/O pins than the alternative interfaces. The LV-EZ1s can detect objects from eight inches up to twenty feet away, which provides plenty of warning before the quadrotor is too close to an object and avoids reporting extraneously long distances.

Fig. 4: Rangefinder.

D. *Camera*

The camera used for image input is a Toshiba TCM8230D. It is capable of outputting 640x480px 16-bit color images at 30 frames-per-second and was selected for its small size and low price. Since the SmartFusion has limited memory, some preprocessing of the image must be performed in order to store it in its entirety and to reduce total image processing time; this pre-processing scales the image down to a black and white image at 128x96px. This means the stored frame only needs to be 3800 bytes of data. Figure 5 illustrates the results of the scaling on a typical image:

Fig. 5: Illustration of image downscaling from 640x480 at 16-bit color to 128x96 at 1-bit color.

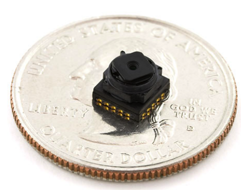


Fig. 6: Camera.

Code for the camera hardware can be found in appendices C and F.

[2]We were originally going to use PWM output, but found that analog fit our design constraints much better.

E. Radio

A XBee 802.15.4 radio set to transparent UART mode is used to implement point-to-point communication for controlling and calibrating the quadrotor (appendices R and S). Pressing a key on the keyboard of a computer with another XBee on the same personal area network sends the corresponding ascii byte to the quadrotor. All commands are single bytes to minimize encoding/decoding overhead. This interface is used to control speed, adjust trim, initiate flight, and terminate flight during testing.

Fig. 7: Xbee radio.

F. Inertial Measurement Unit

An accelerometer, gyroscope, and magentometer (compass) comprise the quadrotors inertial sensors, all of which were included on the Sparkfun 9DOF (degree of freedom) Sensor Stick. The 9 degrees of freedom refers to each of the three sensors taking measurements on an x, y, and z axis for a total of nine pieces of data. The gyroscope data provides information about how much the sensor has rotated about each of three axes since the last measurement cycle, the accelerometer provides the current acceleration in the three axes, and the magnetometer provides the magnetic field strength in each of the three axes (the magnetic field of the Earth provides an absolute reference for orientation). The drivers use a custom I2C interface written in verilog (appendix K), controlled by a state machine which reads each sensor value in sequence (appendices W and X, and software in appendix L).

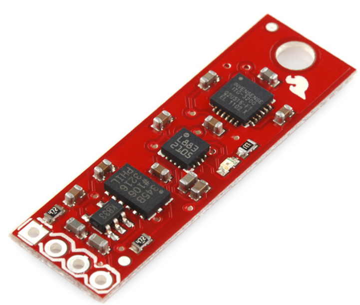


Fig. 8: Nine Degree of Freedom Sensor Stick.

IV. Hardware Implementation

Computational components of Project Levitate were specifically divided between the hardware and software portions of the SmartFusion platform. The line between hardware and software was blurred more and more as the project progressed, however the originally intended division will be discussed here. The updated modifications to the initial design will be discussed in the Difficulties section.

A. Stabilization Hardware

The stabilization of the quadrotor was designed to run in hardware due to its fixed-function nature. Stabilization takes the outputs of the IMU sensors and applies a Complementary Filter, a functional unit that converts sensor data into orientation about the x, y, and z axes of the quadrotor (appendices G and H). There are other filter options such as Kalman Filters, but the complementary filter is the most accurate one that doesnt require floating point calculations. The filter must also avoid drift that is inherent in using MEMS gyroscopes. A Complementary Filter works by scaling the accelerometer and gyro data to both be in the same units, then combines them and calculates a new orientation using the following formula:

$$orientation = .98 \times (previous\_orientation + scaled\_gyro\_data) + .02 \times scaled\_accelerometer\_data$$

Next, a PID (proportional-integral-derivative) controller takes the pitch, roll, and yaw data as inputs and computes the proper motor speeds to keep the platform level based on these inputs. A PID controller operates by calculating a response (in this case a motor speed) based on the difference between the measured orientation of the quadrotor and its desired orientation (appendix Q). By taking the derivative and integral of this error in real time, and multiplying these by fixed constants, the motor speeds can be manipulated to return the quadrotor to a flat orientation and prevent oscillation due to overcorrection (motorcontrol hardware and software in appendices N and P). Each axis of orientation and the related error is processed independently with a separate PID controller. This alleviates the need to combine the error about each axis into a 3D vector, consequently also simplifying the computations necessary for stabilization.

B. Locomotion

Locomotion is an extension of the stabilization hardware. It works by altering the goal which the PID controller is trying to reach. Normally the PID controller works to level the quadrotor, but if the reference position is set to something other than being level, it instead works to orient the quadrotor to move in the specified direction (by tilting toward that direction).

C. Rangefinders

The hardware which interacts with the rangefinders consists of six parallel PWM decoders which convert the PWM output of the rangefinders to inches. These measurements are then filtered using a simple running mean of the most recent four measurements. This is implemented by adding and right-shifting-by-2 the most recent measurements. These mean measurements are then presented to the software loop via memory-mapped registers.

D. Camera

The hardware that interacts with the camera is responsible for starting up the camera into the proper mode. The resolution is set to 128x96, and the color is set to black and white. Communications to the camera are sent over an I2C bus. An FPGA camera handler is responsible for sending a clock to the camera, which the camera mirrors back when the picture is being sent. Once the camera is configured, the camera handler waits for the camera to send a start code to signal the beginning of the picture. The picture is stored in an implicit block ram.

Once the entire picture is collected, the camera handler waits and will not collect any more data. The CPU can then request any pixel be sent over the APB (advanced peripheral bus). Once the CPU has collected all of the data it needs for the image processing, it sends a signal to the camera handler (again over the APB), telling it to collect another picture.

V. Software Implementation

The software components of Project Levitate interface with the hardware and with additional external peripherals to complete the control of the quadrotor. There are several primary components of the software:

- Image Processing
- Minimum Safe Distance Controller

- Flight Loop

These components function by finding the target in the quadrotor's field of vision, deciding whether it is safe to move (including whether evasive action is necessary), and either avoiding a collision or continuing to follow the target.

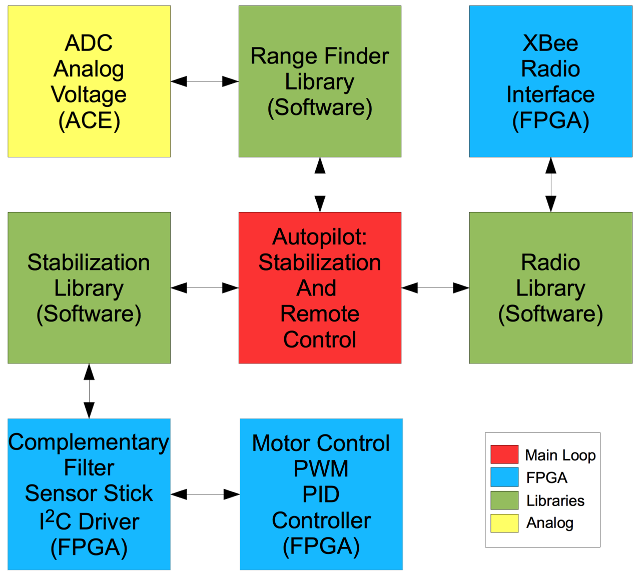


Fig. 9: General software block diagram.

A. *Minimum Safe Distance Controller*

The minimum safe distance controller (MSDC) is a small software component which decides whether there is something too close to the quadrotor and, if so, supersedes the tracking code to move away from the obstruction until a minimum safe distance is reached. After the obstacle is avoided, tracking can resume.

B. *Image Processing*

The actual image processing is done on the CPU itself. It collects all of the image data from the FPGA, and stores it in the CPUs memory. From there, it generates a histogram of the image (how much black there is compared to white, and how much black and white exist on each row and column of the image). It is also possible to stream the data, thus only storing the histogram, which minimizes memory usage.

From there, the quadrotor uses changes in the histogram to determine the targets location. For example, if the picture is mostly black, but a part of the histogram is white, it tries to find the largest chunk and use it as the target. Likewise, if the picture was mostly white, it would find the largest black chunk. Other versions of the algorithm test every possible target rather than only the largest possible one. This is not used here partially due to the space required to store the image data, but also due to the time required to collect the picture data and analyze each potential target.

Once a target has been determined, the software decides if it is the actual target or just some unusual blob on the picture. If the target has approximately the same amount of black as white, then it believes that it is the target; based on where that was in the picture, a flight bearing is determined. As the software will not find a target in every frame it captures, it tells the quadrotor to continue in the same direction for a certain amount of time; if the target is not soon reacquired, the flight loop is informed that there is no target.

Figure 10 is a copy of the tag used as a target. This tag was chosen because the pattern on it is unlikely to occur in the wild. Also, each row and column of the target has approximately the same amount of black and white. Having the same amount of black and white allows the software to distinguish the target from randomly occurring patterns.

Additionally, the figure can be skewed and rotated quite a bit and the image processing will find a rectangle inside the target which should still have the same amount of black and white.

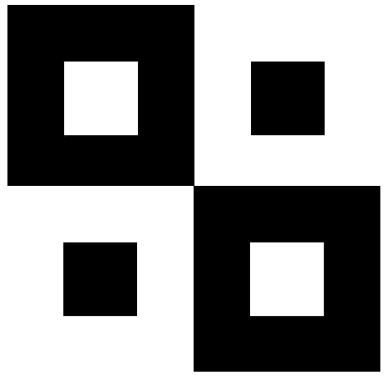


Fig. 10: Camera Target.

Due to the lack of space on the FPGA, an additional version of the software was planned that used the CPU to do what much of the FPGA did. In this version, the FPGA simple forwarded the data from I2C to the APB for the image processing to handle, and did not do any work to determine where the picture began/ended etc. Once the CPU had the data, it would simply turn the raw data into an image, and run the same image processing algorithms on it.

C. Flight Loop

The software flight loop runs the MSDC and the image processing together. This means that, devoid of a target, the quadrotor will attempt to stay level and avoid collisions. When the target is detected, the quadrotor attempts to follow it. This code also contains functions for takeoff and landing sequences.

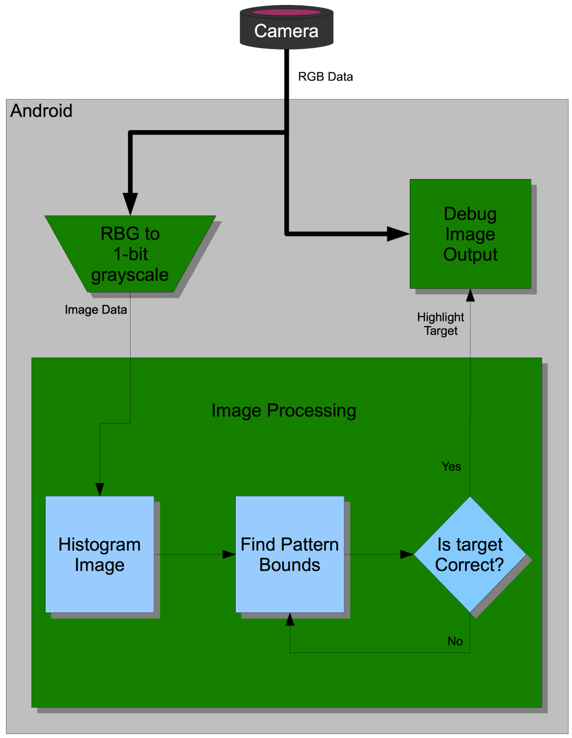


Fig. 11: Camera interfacing.

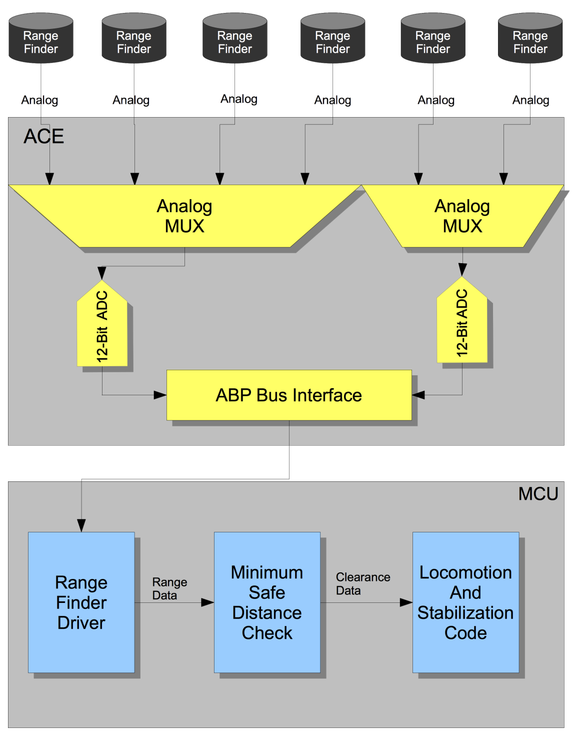


Fig. 12: Rangefinder interfacing.

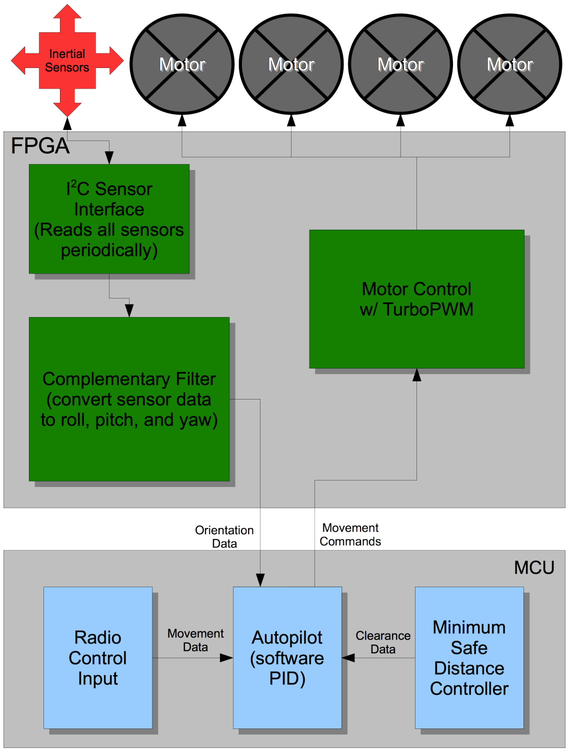


Fig. 13: Interfacing for the motors, IMU, stabilization, and locomotion to flight loop.

Figure 14 displays the software run loop based on the flight loop model:

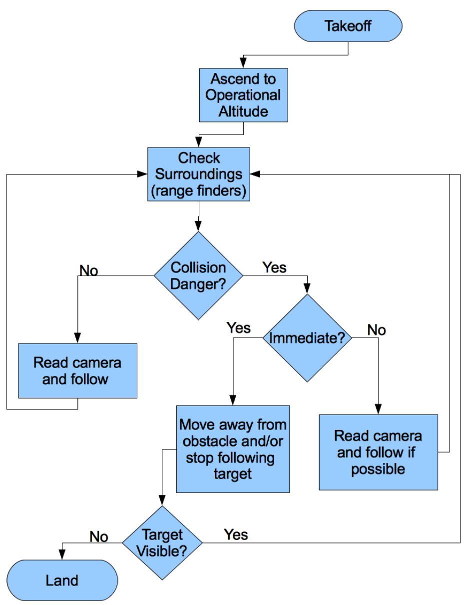


Fig. 14: Software run loop.

The code for the software flight loop can be found in Appendix A and M.

VI. Difficulties Encountered

Several setbacks and design oversights in the original specification resulted in major redesigns of the system midway through the project. These setbacks included the limited size of the FPGA hardware, a crash which disabled the quadrotor for a period of time, trouble compensating for the effect of mechanical vibration on the inertial sensor data, limited battery life, and slow voltage drop-off of the battery during operation.

A. FPGA Size

One of the largest unforeseen issues with the design specifications was the size of the FPGA on the SmartFusion. Initially there was no indication that there would be an issue fitting all the hardware slated for the FPGA on simultaneously. When the first revision of the rangefinder verilog code was synthesized, it alone took up more that 100% of the available FPGA fabric. At this point, it became clear heavy hardware optimization would be necessary in order to fit anything, let alone everything. Several weeks were spent on this optimization with varying success. Eventually, the rangefinder module was removed and postponed for redesign while other components were synthesized in its place. Even with this reduction in utilization, when the final collection of hardware was combined simultaneously (still without rangefinder or camera hardware), there was still not enough room.

This caused two changes in project architecture: most of the sensor filtering, and all of stabilization would be done in software (appendix J), and rangefinders would be done with analog using the built-in ACE on the SmartFusion (appendices T, U, and V). This second change required that we redesign the daughterboard PCB and have it manufactured quickly (appendices Y and Z). This was completed and the rangefinders were successfully connected to the software and read; however, the delay caused by working out these FPGA issues did not leave enough time for them to be integrated into the final project, despite the fact that they were working. The other major consequence of working for so long on optimization was that the software version of the stabilization code did not materialize until late November, by which time most of the time allocated for working on the project was gone.

B. Engineering Day Incident

During the University of Utah Engineering Day demo, a combination of over enthusiasm and poor preflight checks resulted in the destructive detachment of one of the motors from the frame. Not only was the propeller destroyed, the motor wiring was severed too, rendering the quadrotor untestable for over a week before a new motor could be attached and the remaining motor mounts could be properly secured. Although it did make a good story, this setback completely stalled testing of verilog during this time.

C. Vibration, Sensor Noise, and Weight

Once the project had progressed to the point that all the working elements had been combined into one piece, the time came to flight test it. With access to the ARLab flight room, flight tests were conducted using a laptop to control throttle and adjust stabilization constants. Tethered tests of stabilization at low thrust across a single axis were successful. However, once all four motors were spun up to flight speed, it became apparent that there were severe issues with vibration making gyroscope data too noisy to glean any meaningful orientation data. The sensor board was rigidly mounted to the bottom of the frame and we found that many hobbyists who build similar platforms attach their sensors to foam tape to dampen vibrations in the frame caused by the motors. In addition, many individuals weight their sensors to increase the inertia of the sensors and reduce the effect of vibrations transmitted along the wires connecting the sensors to the controller. Both of these concepts were applied to the quadrotor with moderate success. Weight was applied to our IMU in the form of a lead block, and the sensor was mounted to the frame on a loop of foam rubber. This reduced the sensor noise substantially and testing proceeded with much more success thanks to the reduction. Unfortunately, vibration and noise were much worse at higher throttle. In addition, the throttle required to start attaining buoyancy was much too high. This meant that the system weighed too much for the motors and the 2:1 thrust to weight ratio had been violated. To reduce the noise from these higher speeds and reduce the required thrust for takeoff, the quadrotor chassis was rebuilt with lighter components where possible. The result was that takeoff could be achieved at lower speed and therefore the noise at flight speed was reduced. Both of these issues were very mechanical in nature and were not anticipated during the design of this project (evidenced by the fact that the sensors were originally rigidly mounted to the frame), however they were overcome with clever engineering and a bit of help from the internet[3].

D. Battery Life

The final issue encountered while finishing work on this project was battery life during testing. Between 5 and 10 minutes of battery life in an average flight was expected, but the effect of doing short tests which slowly wear down the charge in the battery was not anticipated. As the charge decreased, the motors started changing behavior. The motor the farthest from the battery in the power circuitry would start to lose throttle relative to the others, but slowly enough that ongoing calibration efforts for the stabilization code would start to incorporate compensation for this motor change. When the battery finally did give out, the fully charged replacement would behave much differently, making much of the previous fine-tuning invalid. In addition, the amount of testing required to fully tune the system strained the ability of the battery chargers to keep up with the quadrotor's consumption. This limited the amount of testing that could be done in one sitting when flight tests became possible during the last two weeks of the project.

E. Image Processing Android Application

The FPGA size constraints made it impossible to demonstrate the image processing software on the same build as the actual flight software. However, the only output that the image processing can give on the required hardware platform is standard out over a COM3 port, and actually telling the quadrotor to fly in a certain direction. As the latter was impossible, the only alternative was to write data to standard out. This produces an unsatisfying demonstration of this portion of the project, as the only output would be text saying what the camera saw. Alternatively, the entire picture and interpretation could be sent over COM3, and visualization software could be made to interpret it. Neither of these solutions were viable for a demonstration.

For these reasons, a version of the image processing software was written for Android. This required rewriting most of the code into Java, as well as connecting the android camera to the screen and image processing software. Finally, an additional module was added to the software to display what the camera saw, as well as potential targets and bearings. Figure 15 demonstrates how the software works.

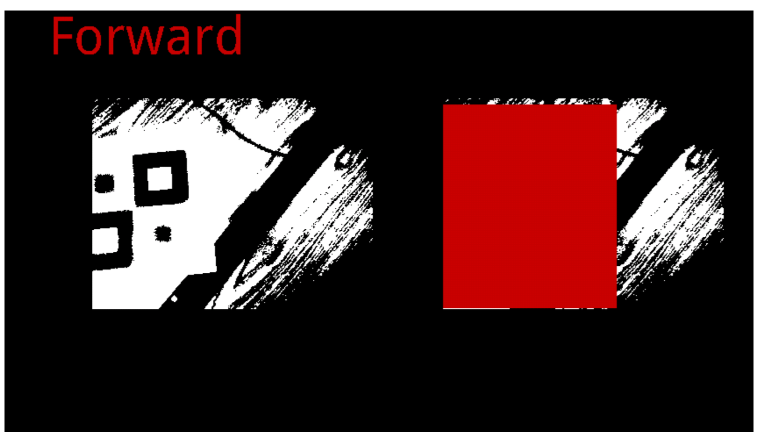


Fig. 15: QuadCamDroid Android Application

Code for this android application (including the algorithms used on the board) is in appendices B, D, and E.

[3]Shout out to Ace Hardware for having *every* possible screw and nut that we would need during the multiple reconstruction efforts. Also to West Valley Hobby for having such knowledgeable and friendly staff.

VII. Bill of Materials

Part	Vendor	Price	Link
AR.Drone frame	Parrot	$25	http://ardrone.parrotshopping.com/us/p_ardrone_product.aspx?i=199953
4 x DC Brushless Motors	Hobbyking	4 x $9	http://www.hobbyking.com/hobbyking/store/__2069__hexTronik_24gram_Brushless_Outrunner_1300kv.html
4 x Electronic Speed Controllers	Hobbyking	4 x $7	http://www.hobbyking.com/hobbyking/store/__6456__Hobbyking_SS_Series_15_18A_ESC.html
9DOF Sensor Stick	Sparkfun	$99	http://www.sparkfun.com/products/10252
6 x Maxbotix LV-EZ1	Sparkfun	6 x $26	http://www.sparkfun.com/products/639
Toshiba TCM8230MD	Sparkfun	$10	https://www.sparkfun.com/products/8667
2200 mAh, 11.1V, 30C Battery	Hobbyking	$14	http://www.hobbyking.com/hobbyking/store/__9394__Turnigy_2200mAh_3S_30C_Lipo_Pack.html
Actel SmartFusion Eval Kit	DigitKey	$120	http://search.digikey.com/scripts/DkSearch/dksus.dll?WT.z_header=search_go&lang=en&site=us&keywords=A2F-EVAL-KIT&x=4&y=13
6 x TURNIGY Plush 18amp Speed Controller	Hobbyking	6 x $12	http://www.hobbyking.com/hobbyking/store/__4312__TURNIGY_Plush_18amp_Speed_Controller.html
6 x 2205C 1400Kv Brushless motor	Hobbyking	6 x $10	http://www.hobbyking.com/hobbyking/store/__7520__2205C_1400Kv_Brushless_motor_.html
Turnigy Talon Carbon Fiber Quadcopter Frame	Hobbyking	$34	http://www.hobbyking.com/hobbyking/store/__22397__Turnigy_Talon_Carbon_Fiber_Quadcopter_Frame.html
Total	$654		

Fig. 16: Bill of Materials

VIII. Conclusion

As it turns out, making something that flies without physically destroying itself is non-trivial. Even less so is making it fly on its own, without adult supervision. The level of difficulty related to the physical realities of flight was not apparent to us during the planning phase of the project. Nor was the scale of the hardware which we would be required to write. During the design process, everything was worked out as a software problem and our self-reported largest unknowns and liabilities were somewhat misplaced. Despite being mechanically unprepared to complete this project (in both the theory and aptitude sense), we managed to mitigate most of the problems encountered. When the FPGA was found to be too small, most of the hardware was either optimized or moved to C code to maintain functionality. What functionality still didn't fit was achieved with analog interfaces on the SmartFusion. When vibration and weight issues became apparent, the entire quadrotor was rebuilt with new, lighter, less vibratory components. The end result was a quadrotor, designed from the ground up, which had the beginnings of stable flight and was very near being tuned to sustain stable hovering and movement. In the process of struggling with design oversights and hardware limitation, we learned quite a bit about FPGA-targeted optimization and more about mechanical engineering and control systems than any of us could have anticipated[4].

The repository for this project can be found at: http://wiesel.ece.utah.edu/redmine/projects/projectlevitate

[4]Also the final product played Jingle Bells on startup, which was fun and lighthearted after the long nights in the flight room.

APPENDIX A

AUTOPILOT.H/C

```c
#ifndef AUTOPILOT_H_  // Only define once
#define AUTOPILOT_H_  // Only define once

#include "customtypes.h"
#include "motors2.h"
#include "radiodriver.h"
#include "pid.h"
#include "filter.h"

int MOVE_DURATION, MOVE_COUNTER, MOVE_RATE;

bool ready;

void autopilot_initialize();
void autopilot_update();
void remoteControl();

#endif /* AUTOPILOT_H_ */
```

```c
#include "autopilot.h"

enum {MOVE_i, MOVE_k, MOVE_j, MOVE_l} MOVE_STATE;

void autopilot_initialize()
{
  MOVE_STATE = MOVE_i;
  MOVE_COUNTER = 0;
  MOVE_RATE = 4;
  MOVE_DURATION = 2;
}

void autopilot_update()
{
  if (MOVE_COUNTER == 1)
  {
    switch(MOVE_STATE)
    {
    case MOVE_i:
      filter_modify_pitch_offset(MOVE_RATE);
    break;

    case MOVE_k:
      filter_modify_pitch_offset(-MOVE_RATE);
    break;

    case MOVE_j:
      filter_modify_roll_offset(MOVE_RATE);
    break;

    case MOVE_l:
      filter_modify_roll_offset(-MOVE_RATE);
    break;
    }
  }
  if (MOVE_COUNTER > 0)
    MOVE_COUNTER--;

}

void remoteControl()
{
  if (radio_didReadChar()) {
    char gotchar = radio_getChar();

    switch(gotchar) {
```

```
    case 'w':
      // throttle up
      speed1 += 500;
      speed2 += 500;
      speed3 += 500;
      speed4 += 500;
      //char foo[512];
      //int messSize = sprintf(foo, "\n%i\n", speed1);
      //radio_sendStrOverRadio(foo, messSize);
          break;
    case 's':
      // throttle down
      speed1 -= 500;
      speed2 -= 500;
      speed3 -= 500;
      speed4 -= 500;
      if (speed1 < 50000)
        speed1 = 50000;
      if (speed2 < 50000)
        speed2 = 50000;
      if (speed3 < 50000)
        speed3 = 50000;
      if (speed4 < 50000)
        speed4 = 50000;
          break;
    case 'a':
      // yaw one way
      yawUpdate(-1 * multiplier);
          break;
    case 'd':
      // yaw the other way
      yawUpdate(1 * multiplier);
          break;

    case 'g':
          // pitch forward
          // MOTOR_1 Up = -roll
          // MOTOR_2 Up = -pitch
          // MOTOR_3 Up = +roll
          // MOTOR_4 Up = +pitch
          // Motor 2 represents the front of the quadrotor.
          // pitch forward means tipping motor 2 down.  So add a negative number to offset the positive that
          // motor 2 would read.
      filter_modify_pitch_offset(-1*multiplier); //Pitch forward by 240.
      break;
    case 'b':
      // pitch back
      filter_modify_pitch_offset(1*multiplier); //Pitch back by 240.
      break;
    case 'v':
      // roll left
      //roll left means lower motor 2, so motor two will be reading positives, add a negative to offset.
      filter_modify_roll_offset(-1*multiplier); // roll left by 240.
      break;
    case 'n':
      // roll right
      filter_modify_roll_offset(1*multiplier); // roll right by 240.
      break;

    case 'i':
          // pitch forward
          // MOTOR_1 Up = -roll
          // MOTOR_2 Up = -pitch
          // MOTOR_3 Up = +roll
          // MOTOR_4 Up = +pitch
          // Motor 2 represents the front of the quadrotor.
```

```
        // pitch forward means tipping motor 2 down.  So add a negative number to offset the positive that
        // motor 2 would read.
      if(MOVE_COUNTER == 0)
      {
        MOVE_STATE = MOVE_i;
        MOVE_COUNTER = MOVE_DURATION;
        filter_modify_pitch_offset(-MOVE_RATE); //Pitch forward by 240.
      }
      else if (MOVE_STATE == MOVE_i)
        MOVE_COUNTER = MOVE_DURATION;
      break;
    case 'k':
      // pitch back
      if(MOVE_COUNTER == 0)
      {
        MOVE_STATE = MOVE_k;
        MOVE_COUNTER = MOVE_DURATION;
        filter_modify_pitch_offset(MOVE_RATE); //Pitch back by 240.
      }
      else if (MOVE_STATE == MOVE_k)
        MOVE_COUNTER = MOVE_DURATION;
      break;
    case 'j':
      // roll left
      //roll left means lower motor 2, so motor two will be reading positives, add a negative to offset.
      if(MOVE_COUNTER == 0)
      {
        MOVE_STATE = MOVE_j;
        MOVE_COUNTER = MOVE_DURATION;
        filter_modify_roll_offset(-MOVE_RATE); // roll left by 240.
      }
      else if (MOVE_STATE == MOVE_j)
        MOVE_COUNTER = MOVE_DURATION;
      break;
    case 'l':
      // roll right
      if(MOVE_COUNTER == 0)
      {
        MOVE_STATE = MOVE_l;
        MOVE_COUNTER = MOVE_DURATION;
        filter_modify_roll_offset(MOVE_RATE); // roll right by 240.
      }
      else if (MOVE_STATE == MOVE_l)
        MOVE_COUNTER = MOVE_DURATION;
      break;

    case 't':
      // take off
      ready = true;
          break;
    case '-':
      // governor up
      governor -= 5000;
      if (governor < 50000)
        governor = 50000;
          break;
    case '=':
      // governor down
      governor += 5000;
          break;
    // test each motor
    case '1':
      speed1 = 55000;
      speed2 = speed3 = speed4 = 0;
          break;
    case '2':
```

```
      speed2 = 55000;
      speed1 = speed3 = speed4 = 0;
          break;
    case '3':
      speed3 = 5000;
      speed1 = speed2 = speed4 = 0;
          break;
    case '4':
      speed4 = 5000;
      speed1 = speed2 = speed3 = 0;
          break;
    // ALL STOP
    case ' ':
      ready = false;
      speed1 = speed2 = speed3 = speed4 = THROTTLE_MIN;
      break;

    case '5':
      P_CONST -= 1 * multiplier;
      if(P_CONST < 0)
        P_CONST = 0;
      break;

    case '6':
      P_CONST += 1 * multiplier;
      break;

    case '7':
      I_CONST -= 1 * multiplier;
      if(I_CONST < 0)
        I_CONST = 0;
      break;

    case '8':
      I_CONST += 1 * multiplier;
      break;

    case '9':
      D_CONST -= 1 * multiplier;
      if(D_CONST < 0)
        D_CONST = 0;
      break;

    case '0':
      D_CONST += 1 * multiplier;
      break;

    case 'z':
      PID_GOVERNOR -= 500;
      if(PID_GOVERNOR < 0)
        PID_GOVERNOR = 0;
      break;

    case 'x':
      PID_GOVERNOR += 500;
      break;

    case '.':
      multiplier = 1;
      break;

    case '/':
      multiplier *= 10;
      break;

    case 'm':
      ;
      char foo[512];
```

```
      int messSize = sprintf(foo, "\r\nm1:%i m3:%i m2:%i m4:%i p:%i i:%i d:%i multiplier:%i pitch:%i roll:%i o_gov:%i\r\n", m1, m3, m2, m4, P_CONST, I_CONST, D_CONST, multiplier, pitch, roll, ORIENTATION_GOVERNOR);
      radio_sendStrOverRadio(foo, messSize);
      break;

    case 'h':
      ORIENTATION_GOVERNOR -= multiplier;
      break;

    case 'y':
      ORIENTATION_GOVERNOR += multiplier;
      break;
    }

    if (!ready) {
      setMotorSpeed(MOTOR_1, speed1);
      setMotorSpeed(MOTOR_2, speed2);
      setMotorSpeed(MOTOR_3, speed3);
      setMotorSpeed(MOTOR_4, speed4);
    }
  }
}
```

APPENDIX B
CAMERAACTIVITY.JAVA

```java
package com.projectlevitate.quadcamdroid;

import java.util.List;

import android.app.Activity;
import android.content.pm.ActivityInfo;
import android.hardware.Camera;
import android.hardware.Camera.Size;
import android.os.Bundle;
import android.view.Window;
import android.view.WindowManager;
import android.view.WindowManager.LayoutParams;
import android.widget.FrameLayout;
import android.widget.Toast;

public class CameraActivity extends Activity {
  // Our variables
  CameraPreview cv;
  CameraOverlay dv;
  FrameLayout alParent;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        /* Set the screen orientation to landscape, because
         * the camera preview will be in landscape, and if we
         * don't do this, then we will get a streached image.*/
        setRequestedOrientation(ActivityInfo.SCREEN_ORIENTATION_LANDSCAPE);

        // requesting to turn the title OFF
        requestWindowFeature(Window.FEATURE_NO_TITLE);

        // making it full screen
        getWindow().setFlags(WindowManager.LayoutParams.FLAG_FULLSCREEN,
            WindowManager.LayoutParams.FLAG_FULLSCREEN);
    }

    public void Load(){
      // Try to get the camera
        Camera c = getCameraInstance();

        List<Size> l = c.getParameters().getSupportedPreviewSizes();

        // If the camera was received, create the app
        if (c != null){
          /* Create our layout in order to layer the
           * draw view on top of the camera preview.
           */
            alParent = new FrameLayout(this);
            alParent.setLayoutParams(new LayoutParams(
                LayoutParams.FILL_PARENT,
                LayoutParams.FILL_PARENT));

            // Create a new camera view and add it to the layout
            dv = new CameraOverlay(this);
            cv = new CameraPreview(this,c, dv);
            alParent.addView(cv);
            alParent.addView(dv);

            // Set the layout as the apps content view
            setContentView(alParent);
        }
        // If the camera was not received, close the app
        else {
          Toast toast = Toast.makeText(getApplicationContext(),
```

```java
                "Unable to find camera. Closing.", Toast.LENGTH_SHORT);
            toast.show();
            finish();
        }
    }

    /* This method is strait for the Android API */
    /** A safe way to get an instance of the Camera object. */
    public static Camera getCameraInstance(){
        Camera c = null;

        try {
            c = Camera.open();// attempt to get a Camera instance
        }
        catch (Exception e){
            // Camera is not available (in use or does not exist)
          e.printStackTrace();
        }
        return c; // returns null if camera is unavailable
    }

    /* Override the onPause method so that we
     * can release the camera when the app is closing.
     */
    @Override
    protected void onPause() {
        super.onPause();

        if (cv != null){
          cv.onPause();
          cv = null;
        }
    }

    /* We call Load in our Resume method, because
     * the app will close if we call it in onCreate
     */
    @Override
    protected void onResume(){
      super.onResume();

      Load();
    }
}
```

APPENDIX C
CAMERADATABUS.V

```
// cameradatabus.v
module cameradatabus(
                    input IO_1_Y,
                    input IO_3_Y,
                    input IO_6_Y,
                    input IO_7_Y,
                    input IO_9_Y,
                    input IO_12_Y,
                    input IO_13_Y,
                    input IO_15_Y,
                    output [7:0] DATA_CAMERA
                    );

    assign DATA_CAMERA = {IO_15_Y, IO_13_Y, IO_12_Y, IO_9_Y, IO_7_Y, IO_6_Y, IO_3_Y, IO_1_Y};

endmodule
```

APPENDIX D
CAMERAOVERLAY.JAVA

```java
package com.projectlevitate.quadcamdroid;

import android.content.Context;
import android.graphics.Bitmap;
import android.graphics.Canvas;
import android.graphics.Paint;
import android.util.DisplayMetrics;
import android.view.SurfaceHolder;
import android.view.SurfaceView;

public class CameraOverlay extends SurfaceView {

    private static final int MOVE_UNKOWN = 0;
    private static final int MOVE_FORWARD = 1;
    private static final int MOVE_BACKWARD = 2;
    private static final int MOVE_LEFT = 3;
    private static final int MOVE_RIGHT = 4;
    private static final int MOVE_NONE = 5;

    private int mMovement = 0;

  private Paint textPaint = new Paint();
  private int[] mData;

  private Bitmap mPicture = null;

  private CameraPreview.Tag mTag = null;

  public CameraOverlay(Context context) {
    super(context);
    textPaint.setARGB(255, 200, 0, 0);
    textPaint.setTextSize(60);
    setWillNotDraw(false);

  }

  @Override
  protected void onDraw(Canvas canvas) {
    switch(mMovement) {
    case MOVE_UNKOWN:
        canvas.drawText("No Target", 50, 50, textPaint);
        break;
    case MOVE_FORWARD:
        canvas.drawText("Forward", 50, 50, textPaint);
        break;
    case MOVE_BACKWARD:
        canvas.drawText("Backword", 50, 50, textPaint);
        break;
    case MOVE_LEFT:
        canvas.drawText("Left", 50, 50, textPaint);
        break;
    case MOVE_RIGHT:
        canvas.drawText("Right", 50, 50, textPaint);
        break;
    case MOVE_NONE:
        canvas.drawText("Center", 50, 50, textPaint);
        break;
      default:
        canvas.drawText("Error", 50, 50, textPaint);
        break;
    }
    if(mPicture != null) {
      canvas.drawBitmap(mPicture, 100, 100, textPaint);
      canvas.drawBitmap(mPicture, 500, 100, textPaint);
    }
```

```java
    if(mTag != null) {
      canvas.drawRect(mTag.start.x + 500, mTag.start.y + 100, mTag.end.x + 500, mTag.end.y + 100,
           textPaint);
    }
  }

  public int getMovement() {
    return mMovement;
  }

  public void setMovement(int movement) {
    mMovement = movement;
    invalidate();
  }

  public int[] getData() {
    return mData;
  }

  public void setData(int[] data) {
    mData = data;
    mPicture = Bitmap.createBitmap(data, 320, 240, Bitmap.Config.ARGB_8888);
    invalidate();
  }

  public CameraPreview.Tag getTag() {
    return mTag;
  }

  public void setTag(CameraPreview.Tag tag) {
    mTag = tag;
  }
}
```

APPENDIX E
CAMERAPREVIEW.JAVA

```java
package com.projectlevitate.quadcamdroid;

import java.io.IOException;

import android.content.Context;
import android.hardware.Camera;
import android.hardware.Camera.PreviewCallback;
import android.util.Log;
import android.view.SurfaceHolder;
import android.view.SurfaceView;

/** A basic Camera preview class */
public class CameraPreview extends SurfaceView implements SurfaceHolder.Callback, PreviewCallback {
    private SurfaceHolder mHolder;
    private Camera mCamera;
    private CameraOverlay mCameraOverlay;
    private String TAG = "QuadCamDroid";

    public CameraPreview(Context context) {
      super(context);
    }

    @SuppressWarnings("deprecation")
  public CameraPreview(Context context, Camera camera, CameraOverlay overlay) {
        super(context);
        mCamera = camera;
        mCameraOverlay = overlay;

        // Install a SurfaceHolder.Callback so we get notified when the
        // underlying surface is created and destroyed.
        mHolder = getHolder();
        mHolder.addCallback(this);
        // deprecated setting, but required on Android versions prior to 3.0
        mHolder.setType(SurfaceHolder.SURFACE_TYPE_PUSH_BUFFERS);
    }

    public void surfaceCreated(SurfaceHolder holder) {
        // The Surface has been created, now tell the camera where to draw the preview.
        try {
            Camera.Parameters camPams = mCamera.getParameters();
            //camPams.setPreviewFormat(ImageFormat.RGB_565);
            camPams.setPreviewSize(320, 240);
            mCamera.setParameters(camPams);
            mCamera.setPreviewDisplay(holder);
            mCamera.setPreviewCallback(this);
            mCamera.startPreview();
        } catch (IOException e) {
            Log.d(TAG, "Error setting camera preview: " + e.getMessage());
        }
    }

    public void surfaceDestroyed(SurfaceHolder holder) {
        // empty. Take care of releasing the Camera preview in your activity.
    }

    public void surfaceChanged(SurfaceHolder holder, int format, int w, int h) {
        // If your preview can change or rotate, take care of those events here.
        // Make sure to stop the preview before resizing or reformatting it.

        if (mHolder.getSurface() == null){
          // preview surface does not exist
          return;
        }

```

```java
        // stop preview before making changes
        try {
            mCamera.stopPreview();
        } catch (Exception e){
          // ignore: tried to stop a non-existent preview
        }

        // set preview size and make any resize, rotate or
        // reformatting changes here

        // start preview with new settings
        try {
            Camera.Parameters camPams = mCamera.getParameters();
            //camPams.setPreviewFormat(ImageFormat.RGB_565);
            camPams.setPreviewSize(320, 240);
            mCamera.setParameters(camPams);
            mCamera.setPreviewDisplay(holder);
            mCamera.setPreviewCallback(this);
            mCamera.startPreview();
        } catch (Exception e){
            Log.d(TAG, "Error starting camera preview: " + e.getMessage());
        }
    }

    public void onPause() {
      mCamera.release();
      mCamera = null;
    }

    @Override
    public void onPreviewFrame(byte[] data, Camera camera) {
        int[] picture = convertYUV420_NV21toRGB8888(data, 320, 240);
        byte[] picture_bw = new byte[320*240];
        for(int i = 0; i < picture.length; i++) {
          int sum = (picture[i]&0xff) + ((picture[i]&0xff00) >> 8) + ((picture[i]&0xff0000) >>
              16);
          if(sum < 255*3/2) {
            picture_bw[i] = 0;
          } else {
            picture_bw[i] = 1;
          }
        }
        byte[][] picture_compressed = new byte[240][40];
        for(int i = 0, m = 0; i < 240; i++) {
          for(int j = 0; j < 40; j++) {
            for(int k = 0; k < 8; k++, m++) {
              byte pixel = (byte) (picture_bw[m] << k);
              picture_compressed[i][j] = (byte) (picture_compressed[i][j] | pixel);
            }
          }
        }
        int[] picture_draw = new int[320*240];
        for(int i = 0; i < picture.length; i++) {
          if(picture_bw[i] == 0) {
            picture_draw[i] = 0xff000000;
          } else {
            picture_draw[i] = 0xffffffff;
          }
        }
        int result = process_frame(picture_compressed);
        if(result == MOVE_UNKOWN) {
          if(count > 0) {
            count--;
          } else {
            direction = MOVE_UNKOWN;
          }
        } else {
          direction = result;
```

```java
            count = COUNT_LIM;
        }
        mCameraOverlay.setMovement(direction);
        mCameraOverlay.setData(picture_draw);
        mCameraOverlay.setTag(tag);
        Log.d("QuadCamDroid", "Result:" + result);
        Log.d("QuadCamDroid", "StoreResult:" + direction);
        Log.d("QuadCamDroid", "Count:" + count);
    }

    // From: http://stackoverflow.com/questions/5272388/need-help-with-androids-nv21-format
    /**
     * Converts YUV420 NV21 to RGB8888
     *
     * @param data byte array on YUV420 NV21 format.
     * @param width pixels width
     * @param height pixels height
     * @return a RGB8888 pixels int array. Where each int is a pixels ARGB.
     */
    public static int[] convertYUV420_NV21toRGB8888(byte [] data, int width, int height) {
        int size = width*height;
        int offset = size;
        int[] pixels = new int[size];
        int u, v, y1, y2, y3, y4;

        // i percorre os Y and the final pixels
        // k percorre os pixles U e V
        for(int i=0, k=0; i < size; i+=2, k+=2) {
            y1 = data[i  ]&0xff;
            y2 = data[i+1]&0xff;
            y3 = data[width+i  ]&0xff;
            y4 = data[width+i+1]&0xff;

            u = data[offset+k  ]&0xff;
            v = data[offset+k+1]&0xff;
            u = u-128;
            v = v-128;

            pixels[i  ] = convertYUVtoRGB(y1, u, v);
            pixels[i+1] = convertYUVtoRGB(y2, u, v);
            pixels[width+i  ] = convertYUVtoRGB(y3, u, v);
            pixels[width+i+1] = convertYUVtoRGB(y4, u, v);

            if (i!=0 && (i+2)%width==0)
                i+=width;
        }

        return pixels;
    }

    private static int convertYUVtoRGB(int y, int u, int v) {
        int r,g,b;

        r = y + (int)1.402f*v;
        g = y - (int)(0.344f*u +0.714f*v);
        b = y + (int)1.772f*u;
        r = r>255? 255 : r<0 ? 0 : r;
        g = g>255? 255 : g<0 ? 0 : g;
        b = b>255? 255 : b<0 ? 0 : b;
        return 0xff000000 | (b<<16) | (g<<8) | r;
    }

    private static final int MOVE_UNKOWN = 0;
    private static final int MOVE_FORWARD = 1;
    private static final int MOVE_BACKWARD = 2;
    private static final int MOVE_LEFT = 3;
    private static final int MOVE_RIGHT = 4;
    private static final int MOVE_NONE = 5;
```

```java

    private static final int BW_CUTOFF_X = 100;
    private static final int BW_CUTOFF_Y = 100;
    private static final int BW_LOW_CUTOFF_X = 100;
    private static final int BW_LOW_CUTOFF_Y = 100;
    private static final int BW_EPSILON = 8000;

    private static final int COUNT_LIM = 50;

    private int direction = 0;
    private int count = 0;
    private Tag tag;

    public class Coord {
      int x;
      int y;
      public Coord() {
        x = 0;
        y = 0;
      }
    }

    public class Tag {
      Coord start;
      Coord end;
      public Tag() {
        start = new Coord();
        end = new Coord();
      }
    }

    int process_frame(byte[][] picture)
    {
        // Types
        long[] histogram = new long[2];
        int i, j, k, m;
        Tag largest_tag = new Tag();
        Tag curr_tag = new Tag();
        long[] histogram_x = new long[320];
        long[] histogram_y = new long[240];
        long black = 0;
        long white = 0;

        // Initialize Data
        for(i = 0; i < 2; i++) {
            histogram[i] = 0;
        }
        for(i = 0; i < 320; i++) {
            histogram_x[i] = 0;
        }
        for(i = 0; i < 240; i++) {
            histogram_y[i] = 0;
        }
        largest_tag.start.x = 0;
        largest_tag.start.y = 0;
        largest_tag.end.x = 0;
        largest_tag.end.y = 0;
        curr_tag.start.x = 0;
        curr_tag.start.y = 0;
        curr_tag.end.x = 0;
        curr_tag.end.y = 0;

        // Create histogram
        for(i = 0; i < 240; i++) {
            for(j = 0, m = 0; j < 40; j++) {
                for(k = 0; k < 8; k++, m++) {
                    // Get the color
                    byte color = (byte) ((picture[i][j] >> k) & 0x1);
```

```

                // Add the value to the histogram.
                histogram[color]++;
                histogram_x[m] += color;
                histogram_y[i] += color;
            }
        }
    }

    // Create start/stop of tag
    if(histogram[0] < histogram[1] - 100) {
      // Create start/stop of tag (largest white rectangle)
      for(i = 0; i < 320; i++) {
        for(j = 0; j < 240; j++) {

          // Reset the current tag
          curr_tag.start.x = i;
          curr_tag.start.y = j;
          curr_tag.end.x = i;
          curr_tag.end.y = j;

          // Find next possible curr_tag candidate.
          // start
          for(k = i; k < 320; k++) {
            if(histogram_x[k] > BW_CUTOFF_X) {
              curr_tag.start.x = k;
              break;
            }
          }
          for(k = j; k < 240; k++) {
            if(histogram_y[k] > BW_CUTOFF_Y) {
              curr_tag.start.y = k;
              break;
            }
          }

          // end
          curr_tag.end.x = curr_tag.start.x;
          curr_tag.end.y = curr_tag.start.y;
          for(k = curr_tag.start.x; k < 320; k++) {
            curr_tag.end.x = k;
                  if(histogram_x[k] <= BW_CUTOFF_X) {
                    break;
                  }
          }
          for(k = curr_tag.start.y; k < 240; k++) {
            curr_tag.end.y = k;
            if(histogram_y[k] <= BW_CUTOFF_Y) {
              break;
            }
          }

          // If curr_tag is larger than largest_tag, set largest_tag
          // to currtag.
          if((curr_tag.end.x - curr_tag.start.x)
              * (curr_tag.end.y - curr_tag.start.y)
              > (largest_tag.end.x - largest_tag.start.x)
              * (largest_tag.end.y - largest_tag.start.y)) {
            largest_tag.start.x = curr_tag.start.x;
            largest_tag.start.y = curr_tag.start.y;
            largest_tag.end.x = curr_tag.end.x;
            largest_tag.end.y = curr_tag.end.y;
                }
        }
      }
    } else {
      // Create start/stop of tag (largest white rectangle)
      for(i = 0; i < 320; i++) {
```

```
    for(j = 0; j < 240; j++) {

      // Reset the current tag
      curr_tag.start.x = i;
      curr_tag.start.y = j;
      curr_tag.end.x = i;
      curr_tag.end.y = j;

      // Find next possible curr_tag candidate.
      // start
      for(k = i; k < 320; k++) {
        if(histogram_x[k] < BW_LOW_CUTOFF_X) {
          curr_tag.start.x = k;
          break;
        }
      }
      for(k = j; k < 240; k++) {
        if(histogram_y[k] < BW_LOW_CUTOFF_Y) {
          curr_tag.start.y = k;
          break;
        }
      }

      // end
      curr_tag.end.x = curr_tag.start.x;
      curr_tag.end.y = curr_tag.start.y;
      for(k = curr_tag.start.x; k < 320; k++) {
        curr_tag.end.x = k;
              if(histogram_x[k] >= BW_LOW_CUTOFF_X) {
                break;
              }
      }
      for(k = curr_tag.start.y; k < 240; k++) {
        curr_tag.end.y = k;
        if(histogram_y[k] >= BW_LOW_CUTOFF_Y) {
          break;
        }
      }

      // If curr_tag is larger than largest_tag, set largest_tag
      // to currtag.
      if((curr_tag.end.x - curr_tag.start.x)
          * (curr_tag.end.y - curr_tag.start.y)
          > (largest_tag.end.x - largest_tag.start.x)
          * (largest_tag.end.y - largest_tag.start.y)) {
        largest_tag.start.x = curr_tag.start.x;
        largest_tag.start.y = curr_tag.start.y;
        largest_tag.end.x = curr_tag.end.x;
        largest_tag.end.y = curr_tag.end.y;
            }
    }
  }
}

// Run a histogram to see if it is about the same object
for(i = largest_tag.start.y; i < largest_tag.end.y; i++) {
    for(j = largest_tag.start.x; j < largest_tag.end.x; j++) {
        byte color = (byte) ((picture[i][j>>3] >> (j&0x7)) & 0x1);
        if(color != 0) {
            white++;
        } else {
            black++;
        }
    }
}

tag = largest_tag;
```

```
        // Print out results
        Log.d("QuadCamDroid", "Total Colors:");
        for(i = 0; i < 2; i++) {
            Log.d("QuadCamDroid", "" + i + "\t" + histogram[i]);
        }
        Log.d("QuadCamDroid", "X-Values:");
        for(i = 0; i < 320; i++) {
            Log.d("QuadCamDroid", "" + i + "\t" + histogram_x[i]);
        }
        Log.d("QuadCamDroid", "Y-Values:");
        for(i = 0; i < 240; i++) {
            Log.d("QuadCamDroid", "" + i + "\t" + histogram_y[i]);
        }
        Log.d("QuadCamDroid", "Tag Values:");
        Log.d("QuadCamDroid", "X-Start:" + largest_tag.start.x);
        Log.d("QuadCamDroid", "X-End:" + largest_tag.end.x);
        Log.d("QuadCamDroid", "Y-Start:" + largest_tag.start.y);
        Log.d("QuadCamDroid", "Y-End:" + largest_tag.end.y);
        Log.d("QuadCamDroid", "Color Values:");
        Log.d("QuadCamDroid", "White:" + white + "\tBlack:" + black);
        Log.d("QuadCamDroid", "Black-White:\t" + Math.abs(black-white));

        // Close program if possible target isn't target
        if(Math.abs(black-white) > BW_EPSILON) {
            return MOVE_UNKOWN;
        };

        // Determin what where next target is on screen.
        if(largest_tag.end.x < 120) {
            return MOVE_LEFT;
        } else if(largest_tag.start.x > 200) {
            return MOVE_RIGHT;
        } else if(largest_tag.end.y < 120) {
            return MOVE_FORWARD;
        } else if(largest_tag.start.y > 120) {
            return MOVE_BACKWARD;
        } else {
            return MOVE_NONE;
        }

        // Shouldn't get here
        // return MOVE_UNKOWN;
    }
}
```

APPENDIX F
CAMERAWRAPPER.V

```
// camerawrapper.v
module camerawrapper(
                     DATA_CAMERA,
                     DK,
                     //HD,
                     //VD,
                     REQ_I2C,
                     ADDR_I2C,
                     RW_I2C,
                     REG_I2C,
                     DATA_WRITE_I2C,
                     DATA_SIZE_I2C,
                     DATA_READ_I2C,
                     DATA_READY_I2C,
                     DONE_I2C,
                     RDY_I2C,
                     PCLK,
                     PENABLE,
                     PSEL,
                     PRESETN,
                     PWRITE,
                     PREADY,
                     PSLVERR,
                     PADDR,
                     PWDATA,
                     PRDATA,
                     CAMCLK,
                     FABINT
                     );

    // Camera Comunications
    input                          DK;//, HD, VD;
    input [7:0]                    DATA_CAMERA;
    output reg                     CAMCLK;

    // I2C Interface
    input                          DONE_I2C, RDY_I2C, DATA_READY_I2C;
    input [7:0]                    DATA_READ_I2C;
    output reg [7:0]               DATA_SIZE_I2C;
    output reg                     RW_I2C, REQ_I2C;
    output reg [6:0]               ADDR_I2C;
    output reg [7:0]               REG_I2C;
    output reg [7:0]               DATA_WRITE_I2C;

    // APB Wrapper
    input                          PCLK, PENABLE, PSEL, PRESETN, PWRITE;
    input [31:0]                   PWDATA, PADDR;
    output                         PREADY, PSLVERR;
    output reg [31:0]              PRDATA;
    output reg                     FABINT;

    wire                           rd_enable;
    wire                           wr_enable;

    assign wr_enable               = (PENABLE && PWRITE && PSEL);
    assign rd_enable               = (!PWRITE && PSEL);
    assign PREADY                  = 1'b1;
    assign PSLVERR                 = 1'b0;
    // assign ADDR_I2C             = 8'b01111000;

    // Picture memory
    reg [31:0]                     picture_0;
    reg [31:0]                     picture_1;

    // State
```

```
    reg [3:0]                             state;
    reg                                   picture_state;

    reg [1:0]                             startup_state;

  reg [1:0]                            clk_divider;

    always@(posedge PCLK) begin
        if(~PRESETN) begin
            clk_divider                                     <= 0;
            CAMCLK                                          <= 0;
        end else begin
            if(clk_divider == 2) begin
                clk_divider                                 <= 0;
                CAMCLK                                      <= ~CAMCLK;
            end else begin
                clk_divider                                 <= clk_divider + 1;
            end
        end
    end

    always@(posedge PCLK) begin
        if(~PRESETN) begin
            ADDR_I2C                                        <= 7'd0;
            REG_I2C                                         <= 8'd0;
            DATA_WRITE_I2C                                  <= 8'd0;
            DATA_SIZE_I2C                                   <= 8'd0;
            RW_I2C                                          <= 1'b0;
            REQ_I2C                                         <= 1'b0;
            startup_state                                   <= 0;
        end else begin
            case(startup_state)
                0: begin
                    if(RDY_I2C) begin
                        ADDR_I2C                            <= 7'b0111100;
                        REG_I2C                             <= 8'h3;
                        DATA_WRITE_I2C                      <= 8'b0000110;
                        DATA_SIZE_I2C                       <= 8'd1;
                        RW_I2C                              <= 1'b0;
                        REQ_I2C                             <= 1'b0;
                        startup_state                       <= 1;
                    end else begin
                        ADDR_I2C                            <= 7'd0;
                        REG_I2C                             <= 8'd0;
                        DATA_WRITE_I2C                      <= 8'd0;
                        DATA_SIZE_I2C                       <= 8'd0;
                        RW_I2C                              <= 1'b0;
                        REQ_I2C                             <= 1'b0;
                        startup_state                       <= 0;
                    end
                end
                1: begin
                    if(DONE_I2C) begin
                        ADDR_I2C                            <= 7'd0;
                        REG_I2C                             <= 8'd0;
                        DATA_WRITE_I2C                      <= 8'd0;
                        DATA_SIZE_I2C                       <= 8'd0;
                        RW_I2C                              <= 1'b0;
                        REQ_I2C                             <= 1'b0;
                        startup_state                       <= 2;
                    end
                end
                2: begin
                    ADDR_I2C                                <= 7'd0;
                    REG_I2C                                 <= 8'd0;
                    DATA_WRITE_I2C                          <= 8'd0;
                    DATA_SIZE_I2C                           <= 8'd0;
                    RW_I2C                                  <= 1'b0;
```

```
                REQ_I2C                         <= 1'b0;
                startup_state                   <= 2;
            end
        endcase // case (startup_state)
        if(rd_enable) begin
            if(state == 0) begin
                PRDATA                      <= picture_0;
            end else begin
                PRDATA                      <= picture_1;
            end
        end
    end
end

always@(negedge DK) begin
    if(~PRESETN) begin
        picture_0                           <= 0;
        picture_1                           <= 0;
        state                               <= 0;
        picture_state                       <= 0;
    end else begin
        if(picture_state == 0) begin
            case(state)
                0: begin
                    picture_0[7:0]          <= DATA_CAMERA;
                    state                   <= 1;
                    FABINT                  <= 0;
                end
                1: begin
                    picture_0[15:8]         <= DATA_CAMERA;
                    state                   <= 2;
                    FABINT                  <= 0;
                end
                2: begin
                    picture_0[23:16]        <= DATA_CAMERA;
                    state                   <= 3;
                    FABINT                  <= 0;
                end
                3: begin
                    picture_0[31:24]        <= DATA_CAMERA;
                    state                   <= 0;
                    picture_state           <= 1;
                    FABINT                  <= 1;
                end
            endcase // case (state)
        end else begin
            case(state)
                0: begin
                    picture_1[7:0]          <= DATA_CAMERA;
                    state                   <= 1;
                    FABINT                  <= 0;
                end
                1: begin
                    picture_1[15:8]         <= DATA_CAMERA;
                    state                   <= 2;
                    FABINT                  <= 0;
                end
                2: begin
                    picture_1[23:16]        <= DATA_CAMERA;
                    state                   <= 3;
                    FABINT                  <= 0;
                end
                3: begin
                    picture_1[31:24]        <= DATA_CAMERA;
                    state                   <= 0;
                    picture_state           <= 0;
                    FABINT                  <= 1;
                end
```

```
                endcase // case (state)
            end
        end
    end
endmodule
```

Appendix G
COMPLEMENTARYFILTER.V

```
// complementary.v

module complementary(
                    input RESETN,
                    input CLK,
                    input XYZ_RDY,
          input [15:0] ACCEL_DATA_X,
          input [15:0] ACCEL_DATA_Y,
          input [15:0] ACCEL_DATA_Z,
          input [15:0] GYRO_DATA_X,
          input [15:0] GYRO_DATA_Y,
          input [15:0] GYRO_DATA_Z,
          input [15:0] MAGNETO_DATA_X,
          input [15:0] MAGNETO_DATA_Y,
          input [15:0] MAGNETO_DATA_Z,
                    input signed [7:0] GYRO_X_OFFSET,
                    input signed [7:0] GYRO_Y_OFFSET,
                    input signed [7:0] GYRO_Z_OFFSET,
                    input signed [7:0] ACCEL_X_OFFSET,
                    input signed [7:0] ACCEL_Y_OFFSET,
                    input signed [7:0] ACCEL_Z_OFFSET,

                    output reg RD_RQ,
                    output reg XYZ_READ,

                    output reg signed [15:0] PITCH, // Y
                    output reg signed [15:0] ROLL, // X
                    output reg signed [15:0] YAW,
                    output reg signed [31:0] sumPitch,
                    output reg signed [31:0] sumRoll,
                    output reg signed [15:0] deltaPitch,
                    output reg signed [15:0] deltaRoll
          );


   parameter PASS_RATIO_SHIFT = 4; // input >> PASS_RATIO_SHIFT = input * .03125
   parameter PASS_RATIO_MASK = {{PASS_RATIO_SHIFT{1'b0}}, {(16-PASS_RATIO_SHIFT){1'b1}}};
   parameter ACCEL_RATIO_SHIFT_1 = 4; // ACCEL >> ACCEL_RATIO_SHIFT_1 = ACCEL * 16
   parameter ACCEL_RATIO_SHIFT_2 = 3; // ACCEL >> ACCEL_RATIO_SHIFT_2 = ACCEL * 8
   parameter ACCEL_RATIO_MASK_1 = {{ACCEL_RATIO_SHIFT_1{1'b0}}, {(16-ACCEL_RATIO_SHIFT_1){1'b1}}};
   parameter ACCEL_RATIO_MASK_2 = {{ACCEL_RATIO_SHIFT_2{1'b0}}, {(16-ACCEL_RATIO_SHIFT_2){1'b1}}};
   parameter GYRO_RATIO_SHIFT = 4;

   parameter MAGNETOMETER_X_OFFSET = 0;
   parameter MAGNETOMETER_Y_OFFSET = 0;
   parameter MAGNETOMETER_Z_OFFSET = 0;

   wire signed [15:0] GYRO_X;
   wire signed [15:0] GYRO_Y;
   wire [15:0] GYRO_Z;
   wire signed [15:0] ACCEL_X;
   wire signed [15:0] ACCEL_Y;
   wire [15:0] ACCEL_Z;
   wire [15:0] MAGNETO_X;
   wire [15:0] MAGNETO_Y;
   wire [15:0] MAGNETO_Z;

   wire signed [15:0] GYRO_TEMP_X;
   wire signed [15:0] GYRO_TEMP_Y;
   wire [15:0] GYRO_TEMP_Z;
   wire signed [15:0] ACCEL_TEMP_X;
   wire signed [15:0] ACCEL_TEMP_Y;
   wire [15:0] ACCEL_TEMP_Z;
```

```
wire [15:0] MAGNETO_TEMP_X;
wire [15:0] MAGNETO_TEMP_Y;
wire [15:0] MAGNETO_TEMP_Z;

wire signed [15:0] PITCH_GYRO_X;
wire signed [15:0] ROLL_GYRO_Y;

assign GYRO_TEMP_X = (GYRO_DATA_X);// + $signed({ {8{GYRO_X_OFFSET[7]}}, GYRO_X_OFFSET[7:0] }));//{{(8{GYRO_X_OFFSET[7]}),{GYRO_X_OFFSET}});
assign GYRO_X = ($signed(GYRO_TEMP_X) >>> 4)+ $signed({ {4{GYRO_X_OFFSET[7]}}, GYRO_X_OFFSET[7:0], 4'b0 }); // gyro units ~= accel units * 384.  Divide gyro units by 16 so max value of pitch (~90000) doesnt overflow.  Offset multiplied by 16.

assign GYRO_TEMP_Y = (GYRO_DATA_Y);// + $signed({ {8{GYRO_Y_OFFSET[7]}}, GYRO_Y_OFFSET[7:0] }));
assign GYRO_Y = ($signed(GYRO_TEMP_Y) >>> 4) + $signed({ {4{GYRO_Y_OFFSET[7]}}, GYRO_Y_OFFSET[7:0], 4'b0 }); // gyro units ~= accel units * 384.  Divide gyro units by 16 so max value of pitch (~90000) doesnt overflow.  Offset multiplied by 16.

assign GYRO_Z = GYRO_DATA_Z + GYRO_Z_OFFSET;

assign ACCEL_TEMP_X = (ACCEL_DATA_X);// + $signed({ {8{ACCEL_X_OFFSET[7]}}, ACCEL_X_OFFSET[7:0] }));
assign ACCEL_X = ($signed(ACCEL_TEMP_X) <<< ACCEL_RATIO_SHIFT_1) + ($signed(ACCEL_TEMP_X) <<< ACCEL_RATIO_SHIFT_2) + $signed({ {4{ACCEL_X_OFFSET[7]}}, ACCEL_X_OFFSET[7:0], 4'b0 });  // multiply accel data by 24.  Offset multiplied by 16.

assign ACCEL_TEMP_Y = (ACCEL_DATA_Y);// + $signed({ {8{ACCEL_Y_OFFSET[7]}}, ACCEL_Y_OFFSET[7:0] }));
assign ACCEL_Y = ($signed(ACCEL_TEMP_Y) <<< ACCEL_RATIO_SHIFT_1) + ($signed(ACCEL_TEMP_Y) <<< ACCEL_RATIO_SHIFT_2) + $signed({ {4{ACCEL_Y_OFFSET[7]}}, ACCEL_Y_OFFSET[7:0], 4'b0 });  // multiply accel data by 24.  Offset multiplied by 16.

assign ACCEL_Z = ACCEL_DATA_Z + ACCEL_Z_OFFSET;
assign MAGNETO_X = MAGNETO_DATA_X + MAGNETOMETER_X_OFFSET;
assign MAGNETO_Y = MAGNETO_DATA_Y + MAGNETOMETER_Y_OFFSET;
assign MAGNETO_Z = MAGNETO_DATA_Z + MAGNETOMETER_Z_OFFSET;

assign PITCH_GYRO_X = PITCH + GYRO_X;
assign ROLL_GYRO_Y = ROLL + GYRO_Y;

parameter SUSPENDED = 0;
parameter UPDATE = 1;
parameter PITCH_0 = 2;
parameter ROLL_0 = 3;
/*parameter ROLL_1 =
parameter ROLL_2 =
parameter ROLL_3 =
parameter YAW_0 =
parameter YAW_1 =
parameter YAW_2 =
parameter YAW_3 = */

reg [3:0]                  state;

always@(posedge CLK)
  begin
   if(~RESETN)
   begin
     state <= SUSPENDED;
          RD_RQ <= 0;
          XYZ_READ <= 0;
          PITCH <= 0;
          YAW <= 0;
          ROLL <= 0;
          sumPitch <= 0;
          sumRoll <= 0;
          deltaPitch <= 0;
```

```
                deltaRoll <= 0;
        end
        else
            begin
                case(state)
                  SUSPENDED:
                    begin
                        if (XYZ_RDY)
                          begin
                              state <= PITCH_0;
                              RD_RQ <= 0;
                              XYZ_READ <= 1;
                          end
                        else
                          begin
                              RD_RQ <= 1;
                              XYZ_READ <= 0;
                          end
                    end

                  PITCH_0:
                    begin
                        PITCH <= ((PITCH_GYRO_X) - ($signed(PITCH_GYRO_X) >>> PASS_RATIO_SHIFT)) + ($signed(ACCEL_X) >>> PASS_RATIO_SHIFT);
                        sumPitch <= sumPitch + ((PITCH_GYRO_X) - ($signed(PITCH_GYRO_X) >>> PASS_RATIO_SHIFT)) + ($signed(ACCEL_X) >>> PASS_RATIO_SHIFT);
                        deltaPitch <= ((PITCH_GYRO_X) - ($signed(PITCH_GYRO_X) >>> PASS_RATIO_SHIFT)) + ($signed(ACCEL_X) >>> PASS_RATIO_SHIFT) - PITCH;
                        state <= ROLL_0;
                    end

                  ROLL_0:
                    begin
                        ROLL <= ((ROLL_GYRO_Y) - ($signed(ROLL_GYRO_Y) >>> PASS_RATIO_SHIFT)) + ($signed(ACCEL_Y) >>> PASS_RATIO_SHIFT);
                        sumRoll <= sumRoll + ((ROLL_GYRO_Y) - ($signed(ROLL_GYRO_Y) >>> PASS_RATIO_SHIFT)) + ($signed(ACCEL_Y) >>> PASS_RATIO_SHIFT);
                        deltaRoll <= ((ROLL_GYRO_Y) - ($signed(ROLL_GYRO_Y) >>> PASS_RATIO_SHIFT)) + ($signed(ACCEL_Y) >>> PASS_RATIO_SHIFT) - ROLL;
                        state <= SUSPENDED;
                    end

                endcase
      end // else: !if(~RESETN)
end // always@ (posedge CLK)
endmodule // complimentary
```

Appendix H
COMPLEMENTARYFILTERWRAPPER.V

```
// ////////////////////////////////////////////////////////////////////////////////////////
// Company: <Name>
//
// File: complementaryfilterwrapper.v
// File history:
//      <Revision number>: <Date>: <Comments>
//      <Revision number>: <Date>: <Comments>
//      <Revision number>: <Date>: <Comments>
//
// Description:
//
// <Description here>
//
// Targeted device: <Family::SmartFusion> <Die::A2F200M3F> <Package::484 FBGA>
// Author: <Name>
//
// ////////////////////////////////////////////////////////////////////////////////////////

module complementaryfilterwrapper(
                    input XYZ_RDY,
          input [15:0] ACCEL_DATA_X,
          input [15:0] ACCEL_DATA_Y,
          input [15:0] ACCEL_DATA_Z,
          input [15:0] GYRO_DATA_X,
          input [15:0] GYRO_DATA_Y,
          input [15:0] GYRO_DATA_Z,
          input [15:0] MAGNETO_DATA_X,
          input [15:0] MAGNETO_DATA_Y,
          input [15:0] MAGNETO_DATA_Z,

                    output RD_RQ,
                    output XYZ_READ,

                    output [15:0] PITCH, // Y
                    output [15:0] ROLL, // X
                    output [15:0] YAW,
                    output signed [31:0] sumPitch,
                    output signed [31:0] sumRoll,
                    output signed [15:0] deltaPitch,
                    output signed [15:0] deltaRoll,

          input PCLK,
                    input PENABLE,
                    input PSEL,
                    input PRESETN,
                    input PWRITE,
                    output PREADY,
                    output PSLVERR,
                    input [7:0] PADDR,
                    input [31:0] PWDATA,
                    output reg [31:0] PRDATA
          );

  assign BUS_WRITE_EN = (PENABLE && PWRITE && PSEL);
  assign BUS_READ_EN = (!PWRITE && PSEL); //Data is ready during first cycle to make it availble on the bus when PENABLE is asserted

  assign PREADY = 1'b1;
  assign PSLVERR = 1'b0;

    reg [7:0] GYRO_X_OFFSET;
```

```
    reg [7:0] GYRO_Y_OFFSET;
    reg [7:0] GYRO_Z_OFFSET;
    reg [7:0] ACCEL_X_OFFSET;
    reg [7:0] ACCEL_Y_OFFSET;
    reg [7:0] ACCEL_Z_OFFSET;
    reg data_rdy;
    reg prev_rq; // If prev_rq == 0 and RD_RQ == 1, new data is ready.

    always@(posedge PCLK) begin
    if(~PRESETN) begin
      // nothing to do on reset
    end
    else begin
      if(BUS_READ_EN)
      begin
        case(PADDR[5:2])
        4'b0000:
        begin
          PRDATA[31:0] <= {16'h0, PITCH};
        end

        4'b0001:
        begin
          PRDATA <= {16'h0, ROLL};
        end

        4'b0010:
        begin
          PRDATA <= {16'h0, YAW};
        end

        4'b0011:
        begin
          PRDATA <= {31'h0, data_rdy};
                     data_rdy <= 1'b0;
        end
        endcase
      end
            else if(BUS_WRITE_EN)
      begin
        case(PADDR[5:2])
        4'b0100:
        begin
          GYRO_X_OFFSET <= PWDATA[7:0];
        end

        4'b0101:
        begin
          GYRO_Y_OFFSET <= PWDATA[7:0];
        end

        4'b0110:
        begin
          GYRO_Z_OFFSET <= PWDATA[7:0];
        end

        4'b0111:
        begin
          ACCEL_X_OFFSET <= PWDATA[7:0];
        end

        4'b1000:
        begin
          ACCEL_Y_OFFSET <= PWDATA[7:0];
        end

        4'b1001:
        begin
```

```
          ACCEL_Z_OFFSET <= PWDATA[7:0];
        end
        endcase
      end

          // Update data_rdy
          if (!BUS_READ_EN || PADDR[5:2] != 4'b0011) // If we are not reading the status register.
          begin
              if (prev_rq == 0 && RD_RQ == 1) // If prev_rq == 0 and RD_RQ == 1, new data is ready.
                  data_rdy <= 1'b1;
          end

          prev_rq <= RD_RQ;
    end
  end

    complementary complementary_0(
                    .RESETN(PRESETN),
                    .CLK(PCLK),
                    .XYZ_RDY(XYZ_RDY),
            .ACCEL_DATA_X(ACCEL_DATA_X),
            .ACCEL_DATA_Y(ACCEL_DATA_Y),
            .ACCEL_DATA_Z(ACCEL_DATA_Z),
            .GYRO_DATA_X(GYRO_DATA_X),
            .GYRO_DATA_Y(GYRO_DATA_Y),
            .GYRO_DATA_Z(GYRO_DATA_Z),
            .MAGNETO_DATA_X(MAGNETO_DATA_X),
            .MAGNETO_DATA_Y(MAGNETO_DATA_Y),
            .MAGNETO_DATA_Z(MAGNETO_DATA_Z),
                    .GYRO_X_OFFSET(GYRO_X_OFFSET),
                    .GYRO_Y_OFFSET(GYRO_Y_OFFSET),
                    .GYRO_Z_OFFSET(GYRO_Z_OFFSET),
                    .ACCEL_X_OFFSET(ACCEL_X_OFFSET),
                    .ACCEL_Y_OFFSET(ACCEL_Y_OFFSET),
                    .ACCEL_Z_OFFSET(ACCEL_Z_OFFSET),

                    .RD_RQ(RD_RQ),
                    .XYZ_READ(XYZ_READ),

                    .PITCH(PITCH), // Y
                    .ROLL(ROLL), // X
                    .YAW(YAW),
                    .sumPitch(sumPitch),
                    .sumRoll(sumRoll),
                    .deltaPitch(deltaPitch),
                    .deltaRoll(deltaRoll)
            );


endmodule
```

Appendix I
customTypes.h

```
#ifndef CUSTOMTYPES_H_  // Only define once
#define CUSTOMTYPES_H_  // Only define once

#ifndef NULL
#define NULL ((void*)0)
#endif // #ifndef NULL
#ifndef true
#define true 1
#endif // #ifndef true
#ifndef false
#define false 0
#endif // #ifndef false

typedef signed char bool;

#define MIN(A,B) (A < B ? A : B)

#endif /* CUSTOMTYPES_H_ */
```

Appendix J
FILTER.H/C

```c
#ifndef FILTER_H_  // Only define once
#define FILTER_H_  // Only define once
#include "customtypes.h"
#include <stdint.h>
#include "levitatefpga_hw_platform.h"
#include <stdio.h>
#include "drivers/mss_uart/mss_uart.h"
#include <stdio.h>
#include "imudriver.h"

typedef struct
{
  int32_t pitch;
  int32_t roll;
  int32_t yaw;
  uint32_t data_ready;
  int32_t gyroOffsetX;
  int32_t gyroOffsetY;
  int32_t gyroOffsetZ;
  int32_t accelOffsetX;
  int32_t accelOffsetY;
  int32_t accelOffsetZ;
}filter_t;

int32_t pitch_offset;
int32_t roll_offset;

void initializeFilter();

int32_t getPitch();
int32_t getRoll();

int8_t data_ready();

void filter_set_gyro_x_offset(int32_t offset);
void filter_set_gyro_y_offset(int32_t offset);
void filter_set_accel_x_offset(int32_t offset);
void filter_set_accel_y_offset(int32_t offset);

void filter_set_pitch_offset(int32_t offset);
void filter_set_roll_offset(int32_t offset);

void filter_modify_pitch_offset(int32_t offset);
void filter_modify_roll_offset(int32_t offset);

#endif /* TESTSS_H_ */
```

```c
#include "filter.h"

static volatile filter_t* filter = (filter_t*) COMPLEMENTARYFILTERWRAPPER_0;

void initializeFilter()
{
  //printf("gyrox: %i  gyroy: %i  accelx: %i  accely: %i\r\n", -get_gyro_x(), -get_gyro_y(), -
      get_accel_x(), -get_accel_y());
  filter_set_gyro_x_offset(0);//-get_gyro_x());
  filter_set_gyro_y_offset(0);//-get_gyro_y());
  filter_set_accel_x_offset(0);//-get_accel_x());
  filter_set_accel_y_offset(0);//-get_accel_y());
  //*(((int32_t*) (COMPLEMENTARYFILTERWRAPPER_0) + 6)) = gyroOffsetZ;
}

int8_t data_ready()
{
  return filter->data_ready & 1;
```

```
}

int32_t getPitch()
{
  return (int32_t)((int16_t)(filter->pitch));
}

int32_t getRoll()
{
  return (int32_t)((int16_t)(filter->roll));
}

//Gyro and accel offsets: 24x accel = gyro/16.  That is, gyro = 384 * accel.
void filter_set_gyro_x_offset(int32_t offset)
{
  if (offset > 127)
    offset = 127;
  if (offset < -128)
    offset = -128;
  filter->gyroOffsetX = offset;
}

void filter_set_gyro_y_offset(int32_t offset)
{
  if (offset > 127)
    offset = 127;
  if (offset < -128)
    offset = -128;
  filter->gyroOffsetY = offset;
}

void filter_set_accel_x_offset(int32_t offset)
{
  if (offset > 255)
    offset = 256;
  if (offset < -256)
    offset = -256;
  filter->accelOffsetX = offset;
}

void filter_set_accel_y_offset(int32_t offset)
{
  if (offset > 255)
    offset = 256;
  if (offset < -256)
    offset = -256;
  filter->accelOffsetY = offset;
}

/*
 * IMPORTANT:   Offset will be multiplied by 24, so for example if you change offset by 10, roll and pitch
 *         change by 240.
 */
void filter_set_pitch_offset(int32_t offset)
{
  pitch_offset = offset;
}

void filter_set_roll_offset(int32_t offset)
{
  roll_offset = offset;
}

void filter_modify_pitch_offset(int32_t offset)
{
  pitch_offset += offset;
  filter_set_pitch_offset(pitch_offset);
```

```
}

void filter_modify_roll_offset(int32_t offset)
{
  roll_offset += offset;
  filter_set_roll_offset(roll_offset);
}
```

APPENDIX K
I2CINTERFACE.V

```
// States are progressed through in order (except for the burst states which are on a counter)
// Write States
`define RDY_S          7'd0
`define START_W_1      7'd1
`define START_W_2      7'd2
`define ADDRESS_W_1    7'd3
`define ADDRESS_W_2    7'd4
`define ADDRESS_W_3    7'd5
`define ADDR_ACK_W_1   7'd6
`define ADDR_ACK_W_2   7'd7
`define ADDR_ACK_W_3   7'd8
`define REG_W_1        7'd9
`define REG_W_2        7'd10
`define REG_W_3        7'd11
`define REG_ACK_W_1    7'd12
`define REG_ACK_W_2    7'd13
`define REG_ACK_W_3    7'd14
`define DATA_W_1       7'd15
`define DATA_W_2       7'd16
`define DATA_W_3       7'd17
`define DATA_ACK_W_1   7'd18
`define DATA_ACK_W_2   7'd19
`define DATA_ACK_W_3   7'd20
`define STOP_W_0       7'd21
`define STOP_W_1       7'd22
`define STOP_W_2       7'd23
// Read States
`define START_R_1      7'd24
`define START_R_2      7'd25
`define ADDRESS_R_1    7'd26
`define ADDRESS_R_2    7'd27
`define ADDRESS_R_3    7'd28
`define ADDR_ACK_R_1   7'd29
`define ADDR_ACK_R_2   7'd30
`define ADDR_ACK_R_3   7'd31
`define REG_R_1        7'd32
`define REG_R_2        7'd33
`define REG_R_3        7'd34
`define REG_ACK_R_1    7'd35
`define REG_ACK_R_2    7'd36
`define REG_ACK_R_3    7'd37
`define SR_R_1         7'd38
`define SR_R_2         7'd39
`define SR_R_3         7'd40
`define SR_R_4         7'd41
`define ADDR_2_R_1     7'd42
`define ADDR_2_R_2     7'd43
`define ADDR_2_R_3     7'd44
`define ADDR_2_ACK_R_1 7'd45
`define ADDR_2_ACK_R_2 7'd46
`define ADDR_2_ACK_R_3 7'd47
`define DATA_BURST_R_1 7'd48
`define DATA_BURST_R_2 7'd49
`define DATA_BURST_R_3 7'd50
`define BURST_ACK_R_1  7'd51
`define BURST_ACK_R_2  7'd52
`define BURST_ACK_R_3  7'd53
`define NACK_R_1       7'd54
`define NACK_R_2       7'd55
`define NACK_R_3       7'd56
`define STOP_R_0       7'd57
`define STOP_R_1       7'd58
`define STOP_R_2       7'd59
`define RESET_BUS_1    7'd60
`define RESET_BUS_2    7'd61
```

```
`define RESET_BUS_3  7'd62
`define RESET_BUS_4  7'd63
`define RESET_BUS_5  7'd64
`define RESET_BUS_6  7'd65

// Libero hates me and always makes my parameters all 1's
`define HALFSCL 32'd250
`define  QUARTERSCL 32'd125
`define  SHORTSCL 32'd25
`define  STATUS_PERIOD 32'd500
`define  RESET_PERIOD 32'd50

// i2cinterface.v
module i2cinterface(
                    REQ,
                    ADDR,
                    RW,
                    REGI,
                    DS,
                    DW,
                    DR,
                    DRDY,
                    DONE,
                    RDY,
                    FCLK,
                    RESETN,
                    SCL_IN,
                    SCL_OUT,
                    SCL_EN,
                    SDA_IN,
                    SDA_OUT,
                    SDA_EN,
                    I2C_STATUS_LED,
                    I2C_RESET_LED
                    );

    input [6:0] ADDR;      // Hardware address
    input [7:0] DS;         // Data size in bytes (if reading)
    input [7:0] REGI;     // Device register
    input [7:0] DW;         // Data to write to register
    input REQ, FCLK, RESETN, RW;

    input SCL_IN;
    output reg SCL_OUT;
    output reg SCL_EN;
    input SDA_IN;
    output reg SDA_OUT;
    output reg SDA_EN;
    output reg I2C_STATUS_LED;
    output reg I2C_RESET_LED;
    reg [9:0] statuscounter;
    reg [9:0] resetcounter;

    output reg [7:0] DR;     // Data read from register
    output reg RDY, DONE, DRDY;

    reg [6:0] state;

    reg [2:0] burstcounter;
    reg [7:0] data_size;
    reg [9:0] clkcounter;

//    reg BCLK;
//
//    always@(posedge FCLK) begin
//        if (~RESETN) begin
//            BCLK <= 1'b0;
//        end
```

```
//        else begin
//             BCLK <= ~BCLK;
//        end
//    end

    // I2C Control State Machine
    /*
    Write:
        pull SDA low
        Wait 1/2 of SCL cycle
        start SCL (by pulling low)
        Wait a short amount of time (~500 ns)
        Send Address
            Put address bit on SDA
            Wait 1/2 SCL
            Push SCL high
            Wait 1/2 SCL
            Pull SCL low
        Repeat 8 times
        Let SDA high
        Wait 1/2 SCL
        Push SCL high
        Look for ACK
        Wait 1/2 SCL
        Pull SCL low
        Send Register
            Put register bit on SDA
            Wait 1/2 SCL
            Push SCL high
            Wait 1/2 SCL
            Pull SCL low
        Repeat 8 times
        Let SDA high
        Wait 1/2 SCL
        Push SCL high
        Look for ACK
        Wait 1/2 SCL
        Pull SCL low
        Send Data
            Put data bit on SDA
            Wait 1/2 SCL
            Push SCL high
            Wait 1/2 SCL
            Pull SCL low
        Repeat 8 times
        Let SDA high
        Wait 1/2 SCL
        Push SCL high
        Look for ACK
        Wait 1/2 SCL
        Pull SCL low
        Wait 1/2 SCL
        Let SCL high (On posedge after ACK)
        Wait 1/2 of SCL
        Let SDA high

    Write-Read:
        pull SDA low
        Wait 1/2 of SCL cycle
        start SCL (by pulling low)
        Wait a short amount of time (~500 ns)
        Send Address
            Put address bit on SDA
            Wait 1/2 SCL
            Push SCL high
            Wait 1/2 SCL
            Pull SCL low
        Repeat 8 times
```

```
    Let SDA high
    Wait 1/2 SCL
    Push SCL high
    Look for ACK
    Wait 1/2 SCL
    Pull SCL low
    Send Register
        Put register bit on SDA
        Wait 1/2 SCL
        Push SCL high
        Wait 1/2 SCL
        Pull SCL low
    Repeat 8 times
    Let SDA high
    Wait 1/2 SCL
    Push SCL high
    Look for ACK
    Wait 1/2 SCL
    Pull SCL low
    Wait 1/4 of SCL cycle
    Let SDA go high
    Wait 1/4 of SCL cycle
    Let SCL go high
    Wait 1/4 of SCL cycle
    Pull SDA low
    Wait 1/2 of SCL cycle
    Pull SCL low
    Wait a short amount of time
    Send Address
        Put address bit on SDA
        Wait 1/2 SCL
        Push SCL high
        Wait 1/2 SCL
        Pull SCL low
    Repeat 8 times
    Let SDA high
    Wait 1/2 SCL
    Push SCL high
    Look for ACK
    Wait 1/2 SCL
    Pull SCL low
    Read Data
        Grab data from slave
            Wait 1/2 SCL
            Grab data bit from SDA
            Push SCL high
            Wait 1/2 SCL
            Pull SCL low
        Repeat 8 times
        Pull SDA low (ACK)
        Wait 1/2 SCL
        Push SCL high
        Wait 1/2 SCL
        Let SDA High
        Pull SCL low
        (repeat ACK only until datasize is 1)
    Repeat until datasize is 0
    Wait 1/2 SCL
    Push SCL high
    Wait 1/2 SCL
    Pull SCL low
    Wait 1/2 SCL
    let SCL high
    wait 1/2 SCL
    let SDA high
*/
always@(posedge FCLK/* or negedge RESETN*/) begin
    if (~RESETN) begin
```

```
            state           <= `RESET_BUS_1;
            burstcounter    <= 3'b000;
            clkcounter      <= 9'd0;
            RDY             <= 1'b0;
            DONE            <= 1'b0;
            DRDY            <= 1'b0;
            SDA_EN          <= 1'b0;
            SDA_OUT         <= 1'b0;
            SCL_EN          <= 1'b0;
            SCL_OUT         <= 1'b0;
            data_size       <= 8'b0000_0000;
            DR              <= 8'b0000_0000;
            I2C_STATUS_LED  <= 1'b1;
            I2C_RESET_LED   <= 1'b1;
            statuscounter   <= 10'd0;
            resetcounter    <= 10'd0;
        end
        else begin
            case(state)
                `RESET_BUS_1: begin     // Reset the bus
                    if (clkcounter == `HALFSCL) begin
                        state           <= `RESET_BUS_2;
                        clkcounter      <= 9'd0;
                    end
                    else begin    // Wait 1/2 SCL
                        state           <= `RESET_BUS_1;
                        clkcounter      <= clkcounter + 9'd1;     // Count
                    end
                    SDA_EN          <= 1'b0;     // Make sure we let SDA float (if it's under our control at any point)
                    SCL_EN          <= 1'b1;     // Make sure we have control of the clock
                    SCL_OUT         <= 1'b1;     // Push SCL high
                    I2C_STATUS_LED  <= 1'b1;
                    if (resetcounter == `RESET_PERIOD) begin
                        I2C_RESET_LED   <= ~I2C_RESET_LED;
                        resetcounter    <= 10'd0;
                    end
                end
                `RESET_BUS_2: begin
                    if (clkcounter == `HALFSCL) begin
                        state           <= `RESET_BUS_3;
                        clkcounter      <= 9'd0;
                    end
                    else begin    // Wait 1/2 SCL
                        state           <= `RESET_BUS_2;
                        clkcounter      <= clkcounter + 9'd1;     // Count
                    end
                    SCL_OUT         <= 1'b0;         // Pull clock down
                end
                `RESET_BUS_3: begin
                    if (clkcounter == `HALFSCL) begin
                        if (SDA_IN) begin
                            state           <= `RESET_BUS_4;
                        end
                        else begin
                            state           <= `RESET_BUS_2;
                        end
                        clkcounter      <= 9'd0;
                    end
                    else begin    // Wait 1/2 SCL
                        state           <= `RESET_BUS_3;
                        clkcounter      <= clkcounter + 9'd1;     // Count
                    end
                    SCL_OUT         <= 1'b1;     // push clock up
                end
                `RESET_BUS_4: begin     // preface stop with a low clock (since stop needs SCL to be low to make a low-high transition)
                    if (clkcounter == `HALFSCL) begin
```

```
        state           <= `RESET_BUS_5;
        clkcounter      <= 9'd0;
    end
    else begin    // Wait 1/2 SCL
        state           <= `RESET_BUS_4;
        clkcounter      <= clkcounter + 9'd1;    // Count
    end
    SCL_OUT         <= 1'b0;        // Pull clock down
end
`RESET_BUS_5: begin    // send a stop to finish the reset
    if (clkcounter == `HALFSCL) begin
        state           <= `RESET_BUS_6;
        clkcounter      <= 9'd0;
    end
    else begin    // Wait 1/2 SCL
        state           <= `RESET_BUS_5;
        clkcounter      <= clkcounter + 9'd1;    // Count
    end
    SCL_EN          <= 1'b0;    // Let SCL high (first part of a stop)
end
`RESET_BUS_6: begin
    if (clkcounter == `HALFSCL) begin
        if (SDA_IN) begin
            state           <= `RDY_S;
        end
        else begin
            state           <= `RESET_BUS_1;    // Reset repeatedly until the bus is floating
        end
        clkcounter      <= 9'd0;
    end
    else begin    // Wait 1/2 SCL
        state           <= `RESET_BUS_6;
        clkcounter      <= clkcounter + 9'd1;    // Count
    end
    RDY             <= 1'b1;
    SDA_EN          <= 1'b0;    // Let SDA high (finish transaction)
    resetcounter    <= resetcounter + 10'd1;
end
`RDY_S: begin
    if (REQ == 1'b1) begin
        if (RW == 1'b0) begin
            state           <= `START_W_1;
        end
        else begin
            state           <= `START_R_1;
        end
        burstcounter    <= 3'b000;
        clkcounter      <= 9'd0;
        RDY             <= 1'b1;
        DONE            <= 1'b0;
        DRDY            <= 1'b0;
        SDA_EN          <= 1'b0;
        SDA_OUT         <= 1'b0;
        SCL_EN          <= 1'b0;
        SCL_OUT         <= 1'b0;
        data_size       <= 8'b0000_0000;
        DR              <= 8'b0000_0000;
    end
    if (statuscounter == `STATUS_PERIOD) begin
        I2C_STATUS_LED        <= ~I2C_STATUS_LED;
        statuscounter   <= 10'd0;
    end
    I2C_RESET_LED   <= 1'b1;
end
// //////////////////////////////////////////////////////////////////////////////////////////
```

```
//  ######################################################################################//
//  #////////////////////////////////////////////////////////////////////////////////////#//
//#////////////////////////////////// Write Transaction ////////////////////////////////////#//
//  #////////////////////////////////////////////////////////////////////////////////////#//
//  ######################################################################################//
//  //////////////////////////////////////////////////////////////////////////////////////

`START_W_1: begin
    if (clkcounter == `HALFSCL) begin
        state          <= `START_W_2;
        clkcounter     <= 9'd0;
    end
    else begin    // Wait 1/2 SCL
        state          <= `START_W_1;
        clkcounter     <= clkcounter + 9'd1;    // Count
    end
    RDY            <= 1'b0;    // No longer ready for new requests
    DONE           <= 1'b0;    // Not done with request yet
    SDA_EN         <= 1'b1;    // Pull SDA low
    SDA_OUT        <= 1'b0;
end
`START_W_2: begin
    if (clkcounter == `SHORTSCL) begin    // this stretches the first cycle of the address
        state          <= `ADDRESS_W_1;
        clkcounter     <= 9'd0;
    end
    else begin    // Wait a short amount of time
        state          <= `START_W_2;
        clkcounter     <= clkcounter + 9'd1;    // Count
    end
    SCL_EN         <= 1'b1;    // Start sending the clock out
    SCL_OUT        <= 1'b0;    // Pull clock down
end
`ADDRESS_W_1: begin
    if (clkcounter == `HALFSCL) begin
        state          <= `ADDRESS_W_2;
        clkcounter     <= 9'd0;
    end
    else begin    // Wait 1/2 SCL
        state          <= `ADDRESS_W_1;
        clkcounter     <= clkcounter + 9'd1;    // Count
    end
    SDA_OUT        <= 1'b1 & ({ADDR,RW} >> (3'd7 - burstcounter));    // Put address bit on SDA
end
`ADDRESS_W_2: begin
    if (clkcounter == `HALFSCL) begin
        state          <= `ADDRESS_W_3;
        clkcounter     <= 9'd0;
    end
    else begin    // Wait 1/2 SCL
        state          <= `ADDRESS_W_2;
        clkcounter     <= clkcounter + 9'd1;    // Count
    end
    SCL_OUT        <= 1'b1;        // Push clock up
end
```

```
`ADDRESS_W_3: begin
    if (burstcounter == 3'b111) begin
        state           <= `ADDR_ACK_W_1;
        burstcounter    <= 3'b000;    // reset burst counter
        SDA_EN          <= 1'b0;    // Let SDA go high so we can check for ACK in the next cycle
    end
    else begin
        state           <= `ADDRESS_W_1;
        // Count out all the bits
        burstcounter    <= burstcounter + 1;
    end
    SCL_OUT         <= 1'b0;    // pull clock down
end
`ADDR_ACK_W_1: begin
    if (clkcounter == `HALFSCL) begin
        state           <= `ADDR_ACK_W_2;
        clkcounter      <= 9'd0;
    end
    else begin    // Wait 1/2 SCL
        state           <= `ADDR_ACK_W_1;
        clkcounter      <= clkcounter + 9'd1;    // Count
    end
    SDA_EN          <= 1'b0;    // Let SDA high
end
`ADDR_ACK_W_2: begin
    if (clkcounter == `HALFSCL) begin
        if (SDA_IN) begin
            state           <= `STOP_W_1;    // Slave failed to ACK
            clkcounter      <= 9'd0;
        end
        else begin
            state           <= `ADDR_ACK_W_3;
            clkcounter      <= 9'd0;
        end
    end
    else begin    // Wait 1/2 SCL
        state           <= `ADDR_ACK_W_2;
        clkcounter      <= clkcounter + 9'd1;    // Count
    end
    SCL_OUT         <= 1'b1;    // Push SCL high
end
`ADDR_ACK_W_3: begin
    state           <= `REG_W_1;
    SCL_OUT         <= 1'b0;    // Pull SCL low
    SDA_EN          <= 1'b1;    // Take back SDA
end
`REG_W_1: begin
    if (clkcounter == `HALFSCL) begin
        state           <= `REG_W_2;
        clkcounter      <= 9'd0;
    end
    else begin    // Wait 1/2 SCL
        state           <= `REG_W_1;
        clkcounter      <= clkcounter + 9'd1;    // Count
    end
    SDA_OUT         <= 1'b1 & (REGI >> (3'd7 - burstcounter));    // Put register bit on SDA
end
`REG_W_2: begin
    if (clkcounter == `HALFSCL) begin
        state           <= `REG_W_3;
        clkcounter      <= 9'd0;
    end
    else begin    // Wait 1/2 SCL
        state           <= `REG_W_2;
        clkcounter      <= clkcounter + 9'd1;    // Count
    end
```

```
                    SCL_OUT         <= 1'b1;        // Push clock up
                end
                `REG_W_3: begin
                    if (burstcounter == 3'b111) begin
                        state           <= `REG_ACK_W_1;
                        burstcounter    <= 3'b000;    // reset burst counter
                        SDA_EN          <= 1'b0;    // Let SDA go high so we can check for ACK in the next cycle
                    end
                    else begin
                        state           <= `REG_W_1;
                        // Count out all the bits
                        burstcounter    <= burstcounter + 1;
                    end
                    SCL_OUT         <= 1'b0;    // pull clock down
                end
                `REG_ACK_W_1: begin
                    if (clkcounter == `HALFSCL) begin
                        state           <= `REG_ACK_W_2;
                        clkcounter      <= 9'd0;
                    end
                    else begin    // Wait 1/2 SCL
                        state           <= `REG_ACK_W_1;
                        clkcounter      <= clkcounter + 9'd1;    // Count
                    end
                    SDA_EN          <= 1'b0;    // Let SDA high
                end
                `REG_ACK_W_2: begin
                    if (clkcounter == `HALFSCL) begin
                        if (SDA_IN) begin
                            state           <= `STOP_W_1;    // Slave failed to ACK
                            clkcounter      <= 9'd0;
                        end
                        else begin
                            state           <= `REG_ACK_W_3;
                            clkcounter      <= 9'd0;
                        end
                    end
                    else begin    // Wait 1/2 SCL
                        state           <= `REG_ACK_W_2;
                        clkcounter      <= clkcounter + 9'd1;    // Count
                    end
                    SCL_OUT         <= 1'b1;    // Push SCL high
                end
                `REG_ACK_W_3: begin
                    state           <= `DATA_W_1;
                    SCL_OUT         <= 1'b0;    // Pull SCL low
                    SDA_EN          <= 1'b1;    // Take control of SDA
                end
                `DATA_W_1: begin
                    if (clkcounter == `HALFSCL) begin
                        state           <= `DATA_W_2;
                        clkcounter      <= 9'd0;
                    end
                    else begin    // Wait 1/2 SCL
                        state           <= `DATA_W_1;
                        clkcounter      <= clkcounter + 9'd1;    // Count
                    end
                    SDA_OUT         <= 1'b1 & (DW >> (3'd7 - burstcounter));    // Put register bit on SDA
                end
                `DATA_W_2: begin
                    if (clkcounter == `HALFSCL) begin
                        state           <= `DATA_W_3;
                        clkcounter      <= 9'd0;
                    end
                    else begin    // Wait 1/2 SCL
                        state           <= `DATA_W_2;
```

```
                clkcounter      <= clkcounter + 9'd1;    // Count
            end
            SCL_OUT         <= 1'b1;        // Push clock up
        end
        `DATA_W_3: begin
            if (burstcounter == 3'b111) begin
                state           <= `DATA_ACK_W_1;
                burstcounter    <= 3'b000;    // reset burst counter
                SDA_EN          <= 1'b0;    // Let SDA go high so we can check for ACK in the next cycle
            end
            else begin
                state           <= `DATA_W_1;
                // Count out all the bits
                burstcounter    <= burstcounter + 1;
            end
            SCL_OUT         <= 1'b0;    // pull clock down
        end
        `DATA_ACK_W_1: begin
            if (clkcounter == `HALFSCL) begin
                state           <= `DATA_ACK_W_2;
                clkcounter      <= 9'd0;
            end
            else begin    // Wait 1/2 SCL
                state           <= `DATA_ACK_W_1;
                clkcounter      <= clkcounter + 9'd1;    // Count
            end
            SDA_EN          <= 1'b0;    // Let SDA high
        end
        `DATA_ACK_W_2: begin
            if (clkcounter == `HALFSCL) begin
                if (SDA_IN) begin
                    state           <= `STOP_W_0;    // Slave failed to ACK
                    clkcounter      <= 9'd0;
                end
                else begin
                    state           <= `DATA_ACK_W_3;
                    clkcounter      <= 9'd0;
                end
            end
            else begin    // Wait 1/2 SCL
                state           <= `DATA_ACK_W_2;
                clkcounter      <= clkcounter + 9'd1;    // Count
            end
            SCL_OUT         <= 1'b1;    // Push SCL high
        end
        `DATA_ACK_W_3: begin
            if (clkcounter == `HALFSCL) begin
                state           <= `STOP_W_1;
                clkcounter      <= 9'd0;
                DONE            <= 1'b1;    // Done. Let the requester know
            end
            else begin    // Wait 1/2 SCL
                state           <= `DATA_ACK_W_3;
                clkcounter      <= clkcounter + 9'd1;    // Count
            end
            SCL_OUT         <= 1'b0;    // Pull SCL low
            SDA_EN          <= 1'b1;    // Take control of SDA
            SDA_OUT         <= 1'b0;    // Also make sure to actually set it to 0 (looks like a repeated start if the last bit sent was a 1)
        end
        `STOP_W_0: begin    // This state is reached when the slave fails to ACK
            if (clkcounter == `HALFSCL) begin
                state           <= `STOP_W_1;
                clkcounter      <= 9'd0;
            end
            else begin    // Wait 1/2 SCL
                state           <= `STOP_W_0;
```

```
                clkcounter      <= clkcounter + 9'd1;    // Count
            end
            SCL_OUT         <= 1'b0;    // Pull SCL low
            SDA_EN          <= 1'b1;    // Take control of SDA
        end
        `STOP_W_1: begin
            if (clkcounter == `HALFSCL) begin
                state           <= `STOP_W_2;
                clkcounter      <= 9'd0;
            end
            else begin    // Wait 1/2 SCL
                state           <= `STOP_W_1;
                clkcounter      <= clkcounter + 9'd1;    // Count
            end
            RDY             <= 1'b1;    // Signal ready now so the fcfsmux has a chance to stop presenting the previous request before we respond to it again
            DONE            <= 1'b0;    // Done is a pulse.
            SCL_EN          <= 1'b0;    // Let SCL high (first part of a stop)
        end
        `STOP_W_2: begin
            if (clkcounter == `HALFSCL) begin
                statuscounter   <= statuscounter + 10'd1;
                state           <= `RDY_S;
                clkcounter      <= 9'd0;
            end
            else begin    // Wait 1/2 SCL
                state           <= `STOP_W_2;
                clkcounter      <= clkcounter + 9'd1;    // Count
            end
            SDA_EN          <= 1'b0;    // Let SDA high (finish transaction)
        end

        // //////////////////////////////////////////////////////////////////////////////////////////
        // ########################################################################################//
        // #////////////////////////////////////////////////////////////////////////////////////////#//
        //#///////////////////////////////// Read Transaction ///////////////////////////////////#//
        // #////////////////////////////////////////////////////////////////////////////////////////#//
        // ########################################################################################//
        // //////////////////////////////////////////////////////////////////////////////////////////

        `START_R_1: begin
            if (clkcounter == `HALFSCL) begin
                state           <= `START_R_2;
                clkcounter      <= 9'd0;
            end
            else begin    // Wait 1/2 SCL
                state           <= `START_R_1;
                clkcounter      <= clkcounter + 9'd1;    // Count
            end
            RDY             <= 1'b0;    // Not ready anymore
            DONE            <= 1'b0;    // Not done yet
            SDA_EN          <= 1'b1;    // Pull SDA low
            SDA_OUT         <= 1'b0;
            data_size       <= DS;        // Save the number of bytes we are sending
        end
```

```
            `START_R_2: begin
                if (clkcounter == `SHORTSCL) begin    // this stretches the first cycle of the address
                    state           <= `ADDRESS_R_1;
                    clkcounter      <= 9'd0;
                end
                else begin    // Wait a short amount of time
                    state           <= `START_R_2;
                    clkcounter      <= clkcounter + 9'd1;    // Count
                end
                SCL_EN          <= 1'b1;    // Start sending the clock out
                SCL_OUT         <= 1'b0;    // Pull clock down
            end
            `ADDRESS_R_1: begin
                if (clkcounter == `HALFSCL) begin
                    state           <= `ADDRESS_R_2;
                    clkcounter      <= 9'd0;
                end
                else begin    // Wait 1/2 SCL
                    state           <= `ADDRESS_R_1;
                    clkcounter      <= clkcounter + 9'd1;    // Count
                end
                SDA_OUT         <= 1'b1 & ({ADDR,~RW} >> (3'd7 - burstcounter));    // Put address bit on SDA
            end
            `ADDRESS_R_2: begin
                if (clkcounter == `HALFSCL) begin
                    state           <= `ADDRESS_R_3;
                    clkcounter      <= 9'd0;
                end
                else begin    // Wait 1/2 SCL
                    state           <= `ADDRESS_R_2;
                    clkcounter      <= clkcounter + 9'd1;    // Count
                end
                SCL_OUT         <= 1'b1;     // Push clock up
            end
            `ADDRESS_R_3: begin
                if (burstcounter == 3'b111) begin
                    state           <= `ADDR_ACK_R_1;
                    burstcounter    <= 3'b000;    // reset burst counter
                    SDA_EN          <= 1'b0;    // Let SDA go high so we can check for ACK in the next cycle
                end
                else begin
                    state           <= `ADDRESS_R_1;
                    // Count out all the bits
                    burstcounter    <= burstcounter + 1;
                end
                SCL_OUT         <= 1'b0;    // pull clock down
            end
            `ADDR_ACK_R_1: begin
                if (clkcounter == `HALFSCL) begin
                    state           <= `ADDR_ACK_R_2;
                    clkcounter      <= 9'd0;
                end
                else begin    // Wait 1/2 SCL
                    state           <= `ADDR_ACK_R_1;
                    clkcounter      <= clkcounter + 9'd1;    // Count
                end
                SDA_EN          <= 1'b0;    // Let SDA high
            end
            `ADDR_ACK_R_2: begin
                if (clkcounter == `HALFSCL) begin
                    if (SDA_IN) begin
                        state           <= `STOP_R_0;    // Slave failed to ACK
                        clkcounter      <= 9'd0;
                    end
                    else begin
```

```
                state           <= `ADDR_ACK_R_3;
                clkcounter      <= 9'd0;
            end
        end
        else begin    // Wait 1/2 SCL
            state           <= `ADDR_ACK_R_2;
            clkcounter      <= clkcounter + 9'd1;    // Count
        end
        SCL_OUT         <= 1'b1;    // Push SCL high
    end
    `ADDR_ACK_R_3: begin
        state           <= `REG_R_1;
        SCL_OUT         <= 1'b0;    // Pull SCL low
        SDA_EN          <= 1'b1;    // Take back SDA
    end
    `REG_R_1: begin
        if (clkcounter == `HALFSCL) begin
            state           <= `REG_R_2;
            clkcounter      <= 9'd0;
        end
        else begin    // Wait 1/2 SCL
            state            <= `REG_R_1;
            clkcounter        <= clkcounter + 9'd1;    // Count
        end
        SDA_OUT            <= 1'b1 & (REGI >> (3'd7 - burstcounter));    // Put register bit on SDA
    end
    `REG_R_2: begin
        if (clkcounter == `HALFSCL) begin
            state            <= `REG_R_3;
            clkcounter        <= 9'd0;
        end
        else begin    // Wait 1/2 SCL
            state            <= `REG_R_2;
            clkcounter        <= clkcounter + 9'd1;    // Count
        end
        SCL_OUT         <= 1'b1;        // Push clock up
    end
    `REG_R_3: begin
        if (burstcounter == 3'b111) begin
            state            <= `REG_ACK_R_1;
            burstcounter    <= 3'b000;    // reset burst counter
            SDA_EN            <= 1'b0;    // Let SDA go high so we can check for ACK in the next cycle
        end
        else begin
            state            <= `REG_R_1;
            // Count out all the bits
            burstcounter <= burstcounter + 1;
        end
        SCL_OUT            <= 1'b0;    // pull clock down
    end
    `REG_ACK_R_1: begin
        if (clkcounter == `HALFSCL) begin
            state            <= `REG_ACK_R_2;
            clkcounter        <= 9'd0;
        end
        else begin    // Wait 1/2 SCL
            state            <= `REG_ACK_R_1;
            clkcounter        <= clkcounter + 9'd1;    // Count
        end
        SDA_EN            <= 1'b0;    // Let SDA high
    end
    `REG_ACK_R_2: begin
        if (clkcounter == `HALFSCL) begin
            if (SDA_IN) begin
                state            <= `STOP_R_0;    // Slave failed to ACK
                clkcounter        <= 9'd0;
```

```
                end
                else begin
                    state           <= `REG_ACK_R_3;
                    clkcounter      <= 9'd0;
                end
            end
            else begin    // Wait 1/2 SCL
                state           <= `REG_ACK_R_2;
                clkcounter      <= clkcounter + 9'd1;    // Count
            end
            SCL_OUT         <= 1'b1;    // Push SCL high
        end
        `REG_ACK_R_3: begin
            if (clkcounter == `QUARTERSCL) begin
                state           <= `SR_R_1;
                clkcounter      <= 9'd0;
            end
            else begin    // Wait 1/4 SCL
                state           <= `REG_ACK_R_3;
                clkcounter      <= clkcounter + 9'd1;    // Count
            end
            SCL_OUT         <= 1'b0;    // Pull SCL low
            SDA_EN          <= 1'b1;    // Take control of SDA
            SDA_OUT         <= 1'b0;    // Also make sure to actually set it to 0 (
                looks like sending register data otherwise)
        end
        `SR_R_1: begin
            if (clkcounter == `QUARTERSCL) begin
                state           <= `SR_R_2;
                clkcounter      <= 9'd0;
            end
            else begin    // Wait 1/4 SCL
                state           <= `SR_R_1;
                clkcounter      <= clkcounter + 9'd1;    // Count
            end
            SDA_EN          <= 1'b0;    // Let SDA pull high
        end
        `SR_R_2: begin
            if (clkcounter == `QUARTERSCL) begin
                state           <= `SR_R_3;
                clkcounter      <= 9'd0;
            end
            else begin    // Wait 1/4 SCL
                state           <= `SR_R_2;
                clkcounter      <= clkcounter + 9'd1;    // Count
            end
            SCL_EN          <= 1'b0;    // Let SCL pull high
        end
        `SR_R_3: begin
            if (clkcounter == `QUARTERSCL) begin
                state           <= `SR_R_4;
                clkcounter      <= 9'd0;
            end
            else begin    // Wait 1/4 SCL
                state           <= `SR_R_3;
                clkcounter      <= clkcounter + 9'd1;    // Count
            end
            SDA_EN          <= 1'b1;    // Pull SDA Down
        end
        `SR_R_4: begin
            if (clkcounter == `SHORTSCL) begin
                state           <= `ADDR_2_R_1;
                clkcounter      <= 9'd0;
            end
            else begin    // Wait a short ammount of time
                state           <= `SR_R_4;
                clkcounter      <= clkcounter + 9'd1;    // Count
            end
```

```
            SCL_EN          <= 1'b1;    // Pull SCL down
        end
        `ADDR_2_R_1: begin
            if (clkcounter == `HALFSCL) begin
                state           <= `ADDR_2_R_2;
                clkcounter      <= 9'd0;
            end
            else begin    // Wait 1/2 SCL
                state           <= `ADDR_2_R_1;
                clkcounter      <= clkcounter + 9'd1;    // Count
            end
            SDA_OUT         <= 1'b1 & ({ADDR,RW} >> (3'd7 - burstcounter));    // Put address bit on SDA
        end
        `ADDR_2_R_2: begin
            if (clkcounter == `HALFSCL) begin
                state           <= `ADDR_2_R_3;
                clkcounter      <= 9'd0;
            end
            else begin    // Wait 1/2 SCL
                state           <= `ADDR_2_R_2;
                clkcounter      <= clkcounter + 9'd1;    // Count
            end
            SCL_OUT      <= 1'b1;        // Push clock up
        end
        `ADDR_2_R_3: begin
            if (burstcounter == 3'b111) begin
                state           <= `ADDR_2_ACK_R_1;
                burstcounter    <= 3'b000;    // reset burst counter
                SDA_EN          <= 1'b0;    // Let SDA go high so we can check for ACK in the next cycle
            end
            else begin
                state           <= `ADDR_2_R_1;
                // Count out all the bits
                burstcounter    <= burstcounter + 1;
            end
            SCL_OUT         <= 1'b0;    // pull clock down
        end
        `ADDR_2_ACK_R_1: begin
            if (clkcounter == `HALFSCL) begin
                state           <= `ADDR_2_ACK_R_2;
                clkcounter      <= 9'd0;
            end
            else begin    // Wait 1/2 SCL
                state           <= `ADDR_2_ACK_R_1;
                clkcounter      <= clkcounter + 9'd1;    // Count
            end
            SDA_EN          <= 1'b0;    // Let SDA high
        end
        `ADDR_2_ACK_R_2: begin
            if (clkcounter == `HALFSCL) begin
                if (SDA_IN) begin
                    state           <= `STOP_R_0;    // Slave failed to ACK
                    clkcounter      <= 9'd0;
                end
                else begin
                    state           <= `ADDR_2_ACK_R_3;
                    clkcounter      <= 9'd0;
                end
            end
            else begin    // Wait 1/2 SCL
                state           <= `ADDR_2_ACK_R_2;
                clkcounter      <= clkcounter + 9'd1;    // Count
            end
            SCL_OUT         <= 1'b1;    // Push SCL high
        end
        `ADDR_2_ACK_R_3: begin
```

```
                        state              <= `DATA_BURST_R_1;
                        SCL_OUT            <= 1'b0;     // Pull SCL low
                    end
                    `DATA_BURST_R_1: begin
                        if (clkcounter == `HALFSCL) begin
                            state              <= `DATA_BURST_R_2;
                            clkcounter         <= 9'd0;
                        end
                        else begin    // Wait 1/2 SCL
                            state              <= `DATA_BURST_R_1;
                            clkcounter         <= clkcounter + 9'd1;    // Count
                        end
                    end
                    `DATA_BURST_R_2: begin
                        if (clkcounter == `HALFSCL) begin
                            state              <= `DATA_BURST_R_3;
                            clkcounter         <= 9'd0;
                        end
                        else begin    // Wait 1/2 SCL
                            state              <= `DATA_BURST_R_2;
                            clkcounter         <= clkcounter + 9'd1;    // Count
                        end
                        if (clkcounter == 9'd0) begin   // Grab data bit from SDA
                            DR                 <= {DR[6:0], SDA_IN};
                        end
                        SCL_OUT            <= 1'b1;     // Push SCL high
                    end
                    `DATA_BURST_R_3: begin
                        if (burstcounter == 3'b111) begin
                            if (data_size == 8'd1) begin
                                state              <= `NACK_R_1;
                            end
                            else begin
                                state              <= `BURST_ACK_R_1;
                            end
                            data_size          <= data_size - 8'd1;    // Decrement bytes left to see
                            burstcounter       <= 9'd0;
                        end
                        else begin    // Wait 1/2 SCL
                            state              <= `DATA_BURST_R_1;
                            burstcounter       <= burstcounter + 3'd1;    // Count bits
                        end
                        SCL_OUT            <= 1'b0;     // Pull SCL low
                    end
                    `BURST_ACK_R_1: begin
                        if (clkcounter == `HALFSCL) begin
                            state              <= `BURST_ACK_R_2;
                            clkcounter         <= 9'd0;
                            DRDY               <= 1'b1;     // Data is valid
                        end
                        else begin    // Wait 1/2 SCL
                            state              <= `BURST_ACK_R_1;
                            clkcounter         <= clkcounter + 9'd1;    // Count
                        end
                        SDA_EN             <= 1'b1;     // pull SDA low to ACK
                        SDA_OUT            <= 1'b0;
                    end
                    `BURST_ACK_R_2: begin
                        if (clkcounter == `HALFSCL) begin
                            state              <= `BURST_ACK_R_3;
                            clkcounter         <= 9'd0;
                        end
                        else begin    // Wait 1/2 SCL
                            state              <= `BURST_ACK_R_2;
                            clkcounter         <= clkcounter + 9'd1;    // Count
                        end
                        DRDY               <= 1'b0;     // Data is no longer valid
                        SCL_OUT            <= 1'b1;     // Push SCL high
```

```
            end
            `BURST_ACK_R_3: begin
                state           <= `DATA_BURST_R_1;
                SDA_EN          <= 1'b0;    // let go of SDA so slave can write
                SCL_OUT         <= 1'b0;    // pull SCL low
            end
            `NACK_R_1: begin
                if (clkcounter == `HALFSCL) begin
                    state           <= `NACK_R_2;
                    clkcounter      <= 9'd0;
                    DRDY            <= 1'b1;    // Data is valid
                end
                else begin    // Wait 1/2 SCL
                    state           <= `NACK_R_1;
                    clkcounter      <= clkcounter + 9'd1;    // Count
                end
            end
            `NACK_R_2: begin
                if (clkcounter == `HALFSCL) begin
                    state           <= `NACK_R_3;
                    clkcounter      <= 9'd0;
                end
                else begin    // Wait 1/2 SCL
                    state           <= `NACK_R_2;
                    clkcounter      <= clkcounter + 9'd1;    // Count
                end
                DRDY            <= 1'b0;    // Data no longer valid
                SCL_OUT         <= 1'b1;    // Push SCL high
            end
            `NACK_R_3: begin
                if (clkcounter == `HALFSCL) begin
                    state           <= `STOP_R_1;
                    clkcounter      <= 9'd0;
                    DONE            <= 1'b1;    // Done, let the requester know.
                end
                else begin    // Wait 1/2 SCL
                    state           <= `NACK_R_3;
                    clkcounter      <= clkcounter + 9'd1;    // Count
                end
                DRDY            <= 1'b0;    // Data no longer valid
                SCL_OUT         <= 1'b0;    // Pull SCL low
                SDA_EN          <= 1'b1;    // Take SDA back
                SDA_OUT         <= 1'b0;    // Pull SDA low
            end
            `STOP_R_0: begin    // This state is reached when the slave fails to ACK
                if (clkcounter == `HALFSCL) begin
                    state           <= `STOP_R_1;
                    clkcounter      <= 9'd0;
                end
                else begin    // Wait a short amount of time
                    state           <= `STOP_R_0;
                    clkcounter      <= clkcounter + 9'd1;    // Count
                end
                SCL_OUT         <= 1'b0;    // Pull SCL low
                SDA_EN          <= 1'b1;    // Take control of SDA
                SDA_OUT         <= 1'b0;    // Pull SDA low
            end
            `STOP_R_1: begin
                if (clkcounter == `HALFSCL) begin
                    state           <= `STOP_R_2;
                    clkcounter      <= 9'd0;
                end
                else begin    // Wait 1/2 SCL
                    state           <= `STOP_R_1;
                    clkcounter      <= clkcounter + 9'd1;    // Count
                end
                DONE            <= 1'b0;    // Done is a pulse
                SCL_EN          <= 1'b0;    // Let SCL high (first part of a stop)
```

```verilog
                end
                `STOP_R_2: begin
                    if (clkcounter == `HALFSCL) begin
                        statuscounter   <= statuscounter + 10'd1;
                        state           <= `RDY_S;
                        clkcounter      <= 9'd0;
                        RDY             <= 1'b1;
                    end
                    else begin    // Wait 1/2 SCL
                        state           <= `STOP_R_2;
                        clkcounter      <= clkcounter + 9'd1;    // Count
                    end
                    SDA_EN              <= 1'b0;    // Let SDA high (finish transaction)
                end
                default: begin
                    state               <= `RESET_BUS_1;
                    burstcounter        <= 3'b000;
                    clkcounter          <= 9'd0;
                    RDY                 <= 1'b0;
                    DONE                <= 1'b0;
                    DRDY                <= 1'b0;
                    SDA_EN              <= 1'b0;
                    SDA_OUT             <= 1'b0;
                    SCL_EN              <= 1'b0;
                    SCL_OUT             <= 1'b0;
                    data_size           <= 8'b0000_0000;
                    DR                  <= 8'b0000_0000;
                end
            endcase
        end
    end

endmodule
```

APPENDIX L
IMUDRIVER.H/C

```c
#ifndef IMUDRIVER_H_  // Only define once
#define IMUDRIVER_H_  // Only define once
#include <stdint.h>
#include "levitatefpga_hw_platform.h"
#include <stdio.h>
#include "drivers/mss_uart/mss_uart.h"
#include <stdio.h>
#include "imudriver.h"
#include "customtypes.h"

typedef struct
{
  int32_t accel_x;
  int32_t accel_y;
  int32_t accel_z;
  int32_t gyro_x;
  int32_t gyro_y;
  int32_t gyro_z;
  int32_t magneto_x;
  int32_t magneto_y;
  int32_t magneto_z;
}imu_t;

int32_t get_gyro_x();
int32_t get_gyro_y();
int32_t get_accel_x();
int32_t get_accel_y();

#endif /* IMUDRIVER_H_ */
```

```c
#include "imudriver.h"

static volatile imu_t* imu = (imu_t*) SENSORSTICKWRAPPER_0;

int32_t get_gyro_x()
{
  return (int32_t)((int16_t) imu->gyro_x);
}

int32_t get_gyro_y()
{
  return (int32_t)((int16_t) imu->gyro_y);
}

int32_t get_accel_x()
{
  return (int32_t)((int16_t) imu->accel_x);
}

int32_t get_accel_y()
{
  return (int32_t)((int16_t) imu->accel_y);
}
```

APPENDIX M
MAIN.C

```c
#include <stdio.h>
#include <stdint.h>
#include "levitatefpga_hw_platform.h"
#include "drivers/mss_uart/mss_uart.h"
#include "testss.h"
#include "testrf.h"
#include "software_filter.h"
#include "filter.h"
#include "rangefinder.h"
#include "autopilot.h"
#include "pid.h"

int main()
{
    NVIC_EnableIRQ(Fabric_IRQn);
    /*initializeFilter(0,0,0);
    while(true)
    {
      volatile int i;
      for ( i = 0; i < 5000000; i++)
      {
        pid_update();
        //pid_print();
        pid_set_speeds();
      }
    }*/

    ready = false;

  speed1 = speed2 = speed3 = speed4 = THROTTLE_MIN;
  initializeMotors();
  setMotorSpeed(MOTOR_1 | MOTOR_2 | MOTOR_3 | MOTOR_4, 51000);

  initializeFilter();
  pid_initialize();
  autopilot_initialize();

  while (true) {

    while(!ready) {
      // wait for startup signal
      autopilot_update();
      remoteControl();
    }

    while(ready) {
      // start
      remoteControl();
      if (pid_update())
      {
        //char foo[512];
        //int messSize = sprintf(foo, "\r\nroll:%i\r\n",roll);
        //radio_sendStrOverRadio(foo, messSize);
        autopilot_update();
        pid_set_speeds();
      }
    }
  }
}
```

Appendix N

motors2.h/c

```c
#ifndef MOTORS2_H
#define MOTORS2_H
#include "customtypes.h"
#include <stdint.h>
#include "levitatefpga_hw_platform.h"

#define THROTTLE_MIN 50000 // Should be 1ms, given a counter that counts to 1,000,000 on a 50MHz
    clock
#define THROTTLE_MAX 100000 // Should be 2ms, given a counter that counts to 1,000,000 on a 50MHz
    clock

int governor;

#define MOTOR_1 1  // Up = -roll
#define MOTOR_2 2  // Up = -pitch
#define MOTOR_3 4  // Up = +roll
#define MOTOR_4 8  // Up = +pitch

#include "CMSIS/a2fxxxm3.h"

int speed1;
int speed2;
int speed3;
int speed4;

typedef struct {
    uint32_t motor1;
    uint32_t motor2;
    uint32_t motor3;
    uint32_t motor4;
    uint32_t PWMperiod;
} motor_t;

/*
 * Enables all motors.  Motors are enabled with throttle set to minimum.
 *
 * INPUTS:
 *
 * EXAMPLE1:
 * initializeMotors();
 */
void initializeMotors();

/*
 * INPUTS:
 * int motors: Motor(s) to modify speed.
 * int speed: Speed to set motor(s) to.
 *
 * EXAMPLE1:
 * setMotorSpeed(MOTOR_1, 50000); //Sets speed of motor 1 to 50000.
 *
 * EXAMPLE2:
 * setMotorSpeed(MOTOR_1 | MOTOR_2 | MOTOR_3 | MOTOR_4, 25000) //Sets speed of all four motors to
     25000.
 */
void setMotorSpeed(int motors, int speed);

/*
 * Sets the period of the PWM ESC control signal.
 * Should be between 1,000,000 (MAX) and 125,000 (MIN)
 */
int setPWMPeriod(int period);

/*
 * INPUTS:
```

```
 * int motors: Motor(s) to modify speed.
 * int speed: Speed increase to apply.
 *
 * EXAMPLE1:
 * increaseMotorSpeed(MOTOR_1, 50000); //Increases speed of motor 1 by 50000.
 *
 * EXAMPLE2:
 * increaseMotorSpeed(MOTOR_1 | MOTOR_2 | MOTOR_3 | MOTOR_4, 25000) //Increases speed of all four
     motors by 25000.
 */
void increaseMotorSpeed(int motors, int speed);

/*
 * INPUTS:
 * int change: change to yaw (positive or negative speed change, essentially).
 */
void yawUpdate(int change);

#endif /* MOTORS2_H */
```

```
#include "motors2.h"


// Motors in memory-map
static volatile motor_t * motorcontrol = (motor_t *)MOTORSWRAPPER_0;

/*
 * Enables all motors.  Motors are enabled with throttle set to minimum.
 *
 * INPUTS:
 *
 * EXAMPLE1:
 * initializeMotors();
 */
void initializeMotors()
{
  governor = 100000;
    // wait for some time
  volatile int foo = 0;
  while (foo < 10000000) {
    foo++;
  }

    // then set the motors to minimum throttle
    setMotorSpeed(MOTOR_1 | MOTOR_2 | MOTOR_3 | MOTOR_4, 5500);

    // then wait some more
  foo = 0;
  while (foo < 1000000) {
    foo++;
  }

  // then shut them off (for safety)
    setMotorSpeed(MOTOR_1 | MOTOR_2 | MOTOR_3 | MOTOR_4, 0);

  foo = 0;
  while (foo < 10000000) {
    foo++;
  }

  setMotorSpeed(MOTOR_1 | MOTOR_2 | MOTOR_3 | MOTOR_4, 5500);

  setPWMPeriod(125000);
}

/*
 * INPUTS:
 * int motors: Motor(s) to be enabled.
```

```
 * int speed: Speed to set motor(s) to.
 *
 * EXAMPLE1:
 * setMotorSpeed(MOTOR_1, 50000); //Sets speed of motor 1 to 50000.
 *
 * EXAMPLE2:
 * setMotorSpeed(MOTOR_1 | MOTOR_2 | MOTOR_3 | MOTOR_4, 25000) //Sets speed of all four motors to 25000.
 */
void setMotorSpeed(int motors, int speed)
{
  if(motors & MOTOR_1)
  {
    if (speed > MIN(governor, THROTTLE_MAX))
      speed = MIN(governor, THROTTLE_MAX);
    else if (speed < THROTTLE_MIN)
      speed = THROTTLE_MIN;
    motorcontrol->motor1 = speed;
  }
  if(motors & MOTOR_2)
  {
    if (speed > MIN(governor, THROTTLE_MAX))
      speed = MIN(governor, THROTTLE_MAX);
    else if (speed < THROTTLE_MIN)
      speed = THROTTLE_MIN;
    motorcontrol->motor2 = speed;
  }
  if(motors & MOTOR_3)
  {
    if (speed > MIN(governor, THROTTLE_MAX))
      speed = MIN(governor, THROTTLE_MAX);
    else if (speed < THROTTLE_MIN)
      speed = THROTTLE_MIN;
    motorcontrol->motor3 = speed;
  }
  if(motors & MOTOR_4)
  {
    if (speed > MIN(governor, THROTTLE_MAX))
      speed = MIN(governor, THROTTLE_MAX);
    else if (speed < THROTTLE_MIN)
      speed = THROTTLE_MIN;
    motorcontrol->motor4 = speed;
  }
}

int setPWMPeriod(int period)
{
  motorcontrol->PWMperiod = period;
}

/*
 * INPUTS:
 * int motors: Motor(s) to modify speed.
 * int speed: Speed increase to apply.
 *
 * EXAMPLE1:
 * increaseMotorSpeed(MOTOR_1, 50000); //Increases speed of motor 1 by 50000.
 *
 * EXAMPLE2:
 * increaseMotorSpeed(MOTOR_1, -50000); //Decreases speed of motor 1 by 50000.
 *
 * EXAMPLE3:
 * increaseMotorSpeed(MOTOR_1 | MOTOR_2 | MOTOR_3 | MOTOR_4, 25000) //Increases speed of all four motors by 25000.
 */
void increaseMotorSpeed(int motors, int deltaSpeed)
{
  if(motors & MOTOR_1)
```

```
  {
    setMotorSpeed(MOTOR_1, speed1 + deltaSpeed);
  }
  if(motors & MOTOR_2)
  {
    setMotorSpeed(MOTOR_2, speed2 + deltaSpeed);
  }
  if(motors & MOTOR_3)
  {
    setMotorSpeed(MOTOR_3, speed3 + deltaSpeed);
  }
  if(motors & MOTOR_4)
  {
    setMotorSpeed(MOTOR_4, speed4 + deltaSpeed);
  }
}

/*
 * INPUTS:
 * int change: change to yaw (positive or negative speed change, essentially).
 */
void yawUpdate(int change)
{
  speed1 += change;
  speed3 += change;
  speed2 -= change;
  speed4 -= change;
}
```

Appendix O
MOTORSWRAPPER.V

```
// timerIntWrapper.v
// timerWrap.v
module motorswrapper(
            PCLK,
            PENABLE,
            PSEL,
            PRESETN,
            PWRITE,
            PREADY,
            PSLVERR,
            PADDR,
            PWDATA,
            PRDATA,
            PWM1,
            PWM2,
            PWM3,
            PWM4
                        );

    // APB Bus Interface
    input PCLK,PENABLE, PSEL, PRESETN, PWRITE;
    input [31:0] PWDATA;
    input [31:0]  PADDR;
    output [31:0] PRDATA;
    output        PREADY, PSLVERR;
    output        PWM1;
    output        PWM2;
    output        PWM3;
    output        PWM4;


    assign BUS_WRITE_EN = (PENABLE && PWRITE && PSEL);
    assign BUS_READ_EN = (!PWRITE && PSEL); //Data is ready during first cycle to make it
        availble on the bus when PENABLE is asserted

    assign PREADY = 1'b1;
    assign PSLVERR = 1'b0;

    motors2 motors2_0(   .PCLK(PCLK),
                    .PRESETN(PRESETN),
                    .bus_write_en(BUS_WRITE_EN),
                    .bus_read_en(BUS_READ_EN),
                    .bus_addr(PADDR),
                    .bus_write_data(PWDATA),
                    .bus_read_data(PRDATA),
                .PWM1(PWM1),
                .PWM2(PWM2),
                .PWM3(PWM3),
                .PWM4(PWM4)
                    );


endmodule
```

Appendix P
motors2.v

```
// Motors.v
// PWM compares effectively set motor speed.  Addresses 6, 7, 8, 9 are the four PWM Compares.
module motors2(
        PCLK,
        PRESETN,
        bus_write_en,
        bus_read_en,
        bus_addr,
        bus_write_data,
        bus_read_data,
        PWM1,
        PWM2,
        PWM3,
        PWM4
              );

    input PCLK, PRESETN, bus_write_en, bus_read_en;
    input [7:0] bus_addr;
    input [31:0] bus_write_data;
    output reg [31:0] bus_read_data;
    output reg        PWM1;
    output reg        PWM2;
    output reg        PWM3;
    output reg        PWM4;

    // Bus interaction (set up compare registers)
    reg [19:0]        PWM_compare1;
    reg [19:0]        PWM_compare2;
    reg [19:0]        PWM_compare3;
    reg [19:0]        PWM_compare4;
    reg [19:0]        PWM_Period;
    always@(posedge PCLK) begin
        if(~PRESETN) begin
            PWM_compare1 <= 20'd0;
            PWM_compare2 <= 20'd0;
            PWM_compare3 <= 20'd0;
            PWM_compare4 <= 20'd0;
            bus_read_data <= 20'd0;
        end
        else begin
            if(bus_write_en) begin : WRITE
                case(bus_addr[4:2])
                    3'b000: begin
                        PWM_compare1 <= bus_write_data;
                    end
                    3'b001: begin
            PWM_compare2 <= bus_write_data;
          end
                    3'b010: begin
            PWM_compare3 <= bus_write_data;
          end
                    3'b011: begin
            PWM_compare4 <= bus_write_data;
          end
                    3'b101: begin
            PWM_Period <= bus_write_data;
          end
                endcase
            end
          end
      end

    // [Custom]-period counter
    reg [19:0] counter;
    always@(posedge PCLK) begin
```

```
        if(~PRESETN) begin
            counter <= 20'd0;
        end
        else begin
            if (counter == PWM_Period) begin
                counter <= 20'd0;
            end else begin
                counter <= counter + 20'd1;
            end
        end
    end

    // Generate PWM signals
    always@(posedge PCLK) begin
        if (~PRESETN) begin
          PWM1 <= 0;
          PWM2 <= 0;
          PWM3 <= 0;
          PWM4 <= 0;
        end
        else begin
        if (counter == PWM_compare1) begin
            PWM1 <= 0;
            end
        else if (counter == 20'd0) begin
                PWM1 <= 1;
            end

        if (counter == PWM_compare2) begin
            PWM2 <= 0;
            end
        else if (counter == 20'd0) begin
                PWM2 <= 1;
            end

        if (counter == PWM_compare3) begin
            PWM3 <= 0;
            end
        else if (counter == 20'd0) begin
                PWM3 <= 1;
            end

        if (counter == PWM_compare4) begin
            PWM4 <= 0;
            end
        else if (counter == 20'd0) begin
                PWM4 <= 1;
            end
        end
    end

endmodule
```

APPENDIX Q
PID.H/C

```c
/*
 * pid.h
 *
 *  Created on: Nov 24, 2012
 *      Author: Jon
 */

#include "filter.h"
#include "motors2.h"
#include <stdio.h>
#include "customtypes.h"
#include "radiodriver.h"

int PID_GOVERNOR;
int ORIENTATION_GOVERNOR;
int P_CONST;// 75 //GOOD .5 // BAD 1 //BAD 1 //DECENT .75 //BAD .75 // BEST .75 //DECENT .5 // GOOD .5
int I_CONST;// 0 // .01 //.9
int D_CONST;// 600 //GOOD 4 // BAD 8 //BAD 6 //DECENT 4 //BAD 9 //BEST 6 //DECENT 12 //GOOOD 6
int multiplier;
int32_t m1, m2, m3, m4;

int32_t EQUILIBRIUM_1;
int32_t EQUILIBRIUM_2;
int32_t EQUILIBRIUM_3;
int32_t EQUILIBRIUM_4;

int32_t pitch;
int32_t roll;
int32_t yaw;

int32_t sum_pitch;
int32_t sum_roll;
int32_t sum_yaw;

int32_t delta_pitch;
int32_t delta_roll;
int32_t delta_yaw;

int32_t prev_pitch;
int32_t prev_roll;
int32_t prev_yaw;

void pid_initialize();
bool pid_update();
void pid_print();
void pid_set_speeds();
```

```c
/*
 * pid.c
 *
 *  Created on: Nov 24, 2012
 *      Author: Jon
 */

#include "pid.h"

void pid_initialize()
{
  multiplier = 1;
  PID_GOVERNOR = 50000;
  ORIENTATION_GOVERNOR = 500;

  P_CONST = 0;
  I_CONST = 0;
```

```
  D_CONST = 0;//600;

  EQUILIBRIUM_1 = 0;
  EQUILIBRIUM_2 = 0;
  EQUILIBRIUM_3 = 0;
  EQUILIBRIUM_4 = 0;

  pitch = 0;
  roll = 0;
  yaw = 0;

  sum_pitch = 0;
  sum_roll = 0;
  sum_yaw = 0;

  delta_pitch = 0;
  delta_roll = 0;
  delta_yaw = 0;

  prev_pitch = 0;
  prev_roll = 0;
  prev_yaw = 0;
}

bool pid_update()
{
  if (data_ready())
  {
    prev_pitch = pitch;
    prev_roll = roll;

    pitch = getPitch();
    roll = getRoll();

    //delta_pitch = .75 * delta_pitch + .25 * (pitch - prev_pitch);
    //delta_roll = .75 * delta_roll + .25 * (roll - prev_roll);
    delta_pitch = (pitch - prev_pitch);
    delta_roll = (roll - prev_roll);

    if (delta_roll > ORIENTATION_GOVERNOR)
      roll = prev_roll + ORIENTATION_GOVERNOR;
    else if(delta_roll < -ORIENTATION_GOVERNOR)
      roll = prev_roll - ORIENTATION_GOVERNOR;

    if (delta_pitch > ORIENTATION_GOVERNOR)
      pitch = prev_pitch + ORIENTATION_GOVERNOR;
    else if(delta_pitch < -ORIENTATION_GOVERNOR)
      pitch = prev_pitch - ORIENTATION_GOVERNOR;


    sum_pitch += pitch;
    if (sum_pitch > 50000)
      sum_pitch = 50000;
    if (sum_pitch < -50000)
      sum_pitch = -50000;
    sum_roll += roll;
    if (sum_roll > 50000)
      sum_roll = 50000;
    if (sum_roll < -50000)
      sum_roll = -50000;
    //char foo[512];
    //int messSize = sprintf(foo, "\r\npitch:%i roll:%i\r\n", pitch, roll);
    //radio_sendStrOverRadio(foo, messSize);
    return true;
  }
  return false;
}
```

```c
void pid_print()
{
  printf("pitch: %i  roll: %i  sum_pitch: %i  sum_roll: %i  delta_pitch: %i  delta_roll: %i\r\n", pitch/75, roll/75, sum_pitch, sum_roll, delta_pitch, delta_pitch);
}

void pid_set_speeds()
{
  // MOTOR_1 Up = -roll
  // MOTOR_2 Up = -pitch
  // MOTOR_3 Up = +roll
  // MOTOR_4 Up = +pitch
  int temp_roll = roll - roll_offset;
  int temp_pitch = pitch - pitch_offset;

  m1 = (P_CONST*temp_roll + D_CONST*delta_roll + I_CONST*sum_roll)/100;
  m2 = (P_CONST*temp_pitch + D_CONST*delta_pitch + I_CONST*sum_pitch)/100;
  m3 = (-P_CONST*temp_roll - D_CONST*delta_roll - I_CONST*sum_roll)/100;
  m4 = (-P_CONST*temp_pitch - D_CONST*delta_pitch - I_CONST*sum_pitch)/100;

  if (m1 > PID_GOVERNOR)
    m1 = PID_GOVERNOR;
  else if (m1 < -PID_GOVERNOR)
    m1 = -PID_GOVERNOR;
  if (m2 > PID_GOVERNOR)
    m2 = PID_GOVERNOR;
  else if (m2 < -PID_GOVERNOR)
    m2 = -PID_GOVERNOR;
  if (m3 > PID_GOVERNOR)
    m3 = PID_GOVERNOR;
  else if (m3 < -PID_GOVERNOR)
    m3 = -PID_GOVERNOR;
  if (m4 > PID_GOVERNOR)
    m4 = PID_GOVERNOR;
  else if (m4 < -PID_GOVERNOR)
    m4 = -PID_GOVERNOR;

  m1 = speed1 + m1;
  m2 = speed2 + m2;
  m3 = speed3 + m3;
  m4 = speed4 + m4;

  if (m1 < 55000 && speed1 > 55000)
    m1 = 55000;
  if (m3 < 55000 && speed3 > 55000)
    m3 = 55000;
  if (m2 < 55000 && speed2 > 55000)
    m2 = 55000;
  if (m4 < 55000 && speed4 > 55000)
    m4 = 55000;

  //printf("s1: %i s2: %i s3: %i s4: %i m1: %i  m2: %i  m3: %i  m4: %i  roll: %i  pitch: %i  deltaroll: %i  deltapitch: %i\r\n", speed1, speed2, speed3, speed4, m1, m2, m3, m4, roll, pitch, delta_roll, delta_pitch);

  setMotorSpeed(MOTOR_1, m1);
  setMotorSpeed(MOTOR_2, m2);
  setMotorSpeed(MOTOR_3, m3);
  setMotorSpeed(MOTOR_4, m4);
}
```

APPENDIX R
RADIODRIVER.H/C

```c
#ifndef RADIODRIVER_H_  // Only define once
#define RADIODRIVER_H_  // Only define once

#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include "drivers/mss_uart/mss_uart.h"
#include "CMSIS/a2fxxxm3.h"
#include "levitatefpga_hw_platform.h"
#include "customtypes.h"

typedef struct {
    uint32_t charin;
    uint32_t charout;
    uint32_t busy;
} radio_t;

typedef struct {
    char readchar;
    bool didreadchar;
} radiostatus_t;

/**
 * Send a character over the radio
 */
void radio_sendCharOverRadio(char c);

/**
 * Send a string over the radio
 */
void radio_sendStrOverRadio(char * s, int size);

/**
 * The radio has a char for the program
 */
bool radio_didReadChar();

/**
 * Get character sent to radio
 */
char radio_getChar();

#endif /* RADIODRIVER_H_ */
```

```c
#include "radiodriver.h"

// The radio hardware at its memory map
// Doing it this way is better than cast-dereferenceing a #define.
static volatile radio_t * radio = RADIOWRAPPER_0;

// Store the status
static radiostatus_t radiostatus = {false, '\0'};

static volatile char test;

void Fabric_IRQHandler( void )
{
    // TODO -- make a memory mapped FPGA register that indicates which device triggered the interrupt.

  // TODO -- grab the next character from the radio
  if ((*(uint32_t*)FABINTMUX_0) == 0x1) {
    radiostatus.readchar = radio->charin;
    radiostatus.didreadchar = true;
  }
```

```

    // make sure to clear the interrupt
  (*(uint32_t*)FABINTMUX_0) = 1;
    NVIC_ClearPendingIRQ( Fabric_IRQn );
}

void radio_sendCharOverRadio(char c)
{
    // doing only one character does not require waiting
    radio->charout = c;
    return;
}

void radio_sendStrOverRadio(char * s, int size)
{
    // print one char at a time
  int pos = 0;
  while (pos < size) {
    radio->charout = s[pos++];
    while (radio->busy) {
      // wait for radio to send the data
      test = s[pos];
    }
  }
    return;
}

bool radio_didReadChar()
{
  return radiostatus.didreadchar;
}

char radio_getChar()
{
  if (radiostatus.didreadchar) {
    radiostatus.didreadchar = false;
    return radiostatus.readchar;
  }
  return '\0';
}
```

APPENDIX S
RADIOWRAPPER.V

```verilog
`define TXPERIOD 100

// radiowrapper.v
module radiowrapper(
                input PCLK,
                input PENABLE,
                input PSEL,
                input PRESETN,
                input PWRITE,
                output PREADY,
                output PSLVERR,
                input [31:0] PADDR,
                input [31:0] PWDATA,
                output reg [31:0] PRDATA,
                input DIN,
                output DOUT,
                output reg FAB_INT,
                output reg RX_LED,
                output reg TX_LED
                );

parameter BAUD_CYCLE_LENGTH = 5208;
parameter BAUD_HALF_LENGTH = 2604;
parameter SUSPENDED = 0;
parameter TRANSACTION = 1;

reg [7:0]                  radio_data_in;

assign BUS_WRITE_EN = (PENABLE && PWRITE && PSEL);
assign BUS_READ_EN = (!PWRITE && PSEL); //Data is ready during first cycle to make it availble on the bus when PENABLE is asserted

assign PREADY = 1'b1;
assign PSLVERR = 1'b0;

reg [12:0]                 BAUD_COUNTER;
reg [12:0]                 BAUD_COUNTER2;
reg                        BAUD_RATE;

reg [7:0]                  shift_reg;
reg [7:0]                  radio_data_out;
reg [8:0]                  shift_radio_data_out;
reg [3:0]                  count;
reg [3:0]                  send_count;

reg                        state;
reg                        send_state;
reg                        send_complete;
reg                        send_data;
reg                           busy;

assign DOUT = shift_radio_data_out[0];

//CLOCK DIVIDE AND PUT INTO BAUD_RATE.
always@(posedge PCLK) begin
    if (~PRESETN) begin
         BAUD_COUNTER2 <= 0;
         BAUD_RATE <= 0;
    end
    else begin
        if (BAUD_COUNTER2 == BAUD_HALF_LENGTH) begin
            BAUD_COUNTER2 <= 0;
            BAUD_RATE <= ~BAUD_RATE;
        end
        else begin
```

```
            BAUD_COUNTER2 <= BAUD_COUNTER2 + 1;
        end
    end
 end


always@(posedge BAUD_RATE or negedge PRESETN)
 begin
    if (~PRESETN) begin
        send_state <= SUSPENDED;
        send_complete <= 0;
        shift_radio_data_out <= 1;
        send_count <= 0;
        busy <= 0;
        TX_LED <= 1;
      end
    else begin
        case (send_state)
            SUSPENDED: begin
                if (send_data) begin
                    send_state <= TRANSACTION;
                    shift_radio_data_out <= {radio_data_out, 1'b0};
                    send_complete <= 1;
                    busy <= 1;
                end
                else begin
                    send_complete <= 0;
                    busy <= 0;
                end
                TX_LED <= 1;
            end

            TRANSACTION: begin
                send_complete <= 0;
                if (send_count == 8) begin
                    send_count <= 0;
                    send_state <= SUSPENDED;
                    shift_radio_data_out <= {1'b1, shift_radio_data_out[8:1]};
                    busy <= 0;
                end
                else begin
                    send_count <= send_count + 1;
                    shift_radio_data_out <= {1'b1, shift_radio_data_out[8:1]};
                end
                TX_LED <= 0;
            end
        endcase // case (send_state)
    end//
 end

// This is the data coming/going on the bus.
always@(posedge PCLK) begin
    if (~PRESETN) begin
        PRDATA <= 0;
        send_data <= 0;
    end
    else begin
        if(BUS_WRITE_EN) begin
            case(PADDR[3:2])
                2'b00: begin
                    if (send_complete)
                        send_data <= 0; //Do nothing
                end
                2'b01: begin
                    radio_data_out <= PWDATA; //Only do this when busy is 0.
                    send_data <= 1;
                end
                default: begin
```

```
                    if (send_complete)
                        send_data <= 0; //Do nothing
                end
            endcase // case (PADDR[3:2])
          end
        else if(BUS_READ_EN) begin
            case(PADDR[3:2])
                2'b00: begin
                    PRDATA <= radio_data_in;
                    if (send_complete)
                        send_data <= 0;
                end
                2'b10: begin
                    PRDATA <= (busy || send_data);  // If busy is 1, or send_data is 1 don't try to write to PWDATA;
                    if (send_complete)
                        send_data <= 0;
                end
                default: begin
                    if (send_complete)
                        send_data <= 0; //Do nothing
                end
            endcase // case (PADDR[2])
        end
        else begin
            if (send_complete)
                send_data <= 0;
        end
    end
end

// INPUT TRANSACTION
always@(posedge PCLK) begin
    if (~PRESETN) begin
        state <= SUSPENDED;
        FAB_INT <= 0;
        BAUD_COUNTER <= 0;
        count <= 0;
        shift_reg <= 0;
        radio_data_in <= 0;
        RX_LED <= 1;
    end
    else begin
        case (state)
            SUSPENDED: begin
                BAUD_COUNTER <= 0;
                FAB_INT <= 0;
                radio_data_in <= shift_reg;
                RX_LED <= 1;
                if (!DIN)  begin
                    state <= TRANSACTION;
                end
            end

            TRANSACTION: begin
                if (BAUD_COUNTER == BAUD_CYCLE_LENGTH) begin
                    if (count == 8) begin
                        count <= 0;
                        state <= SUSPENDED;
                        FAB_INT <= 1;
                    end
                    else begin
                        count <= count + 1;
                    end
                    BAUD_COUNTER <= 0;
                end
                else begin
                    if (BAUD_COUNTER == BAUD_HALF_LENGTH) begin
```

```
                        shift_reg <= {DIN, shift_reg[7:1]};
                    end
                    BAUD_COUNTER <= BAUD_COUNTER + 1;
                end
                RX_LED <= 0;
            end
        endcase // case (state)
    end
end

endmodule
```

Appendix T
rangefinder.h/c

```c
#ifndef RANGEFINDER_H_  // Only define once
#define RANGEFINDER_H_  // Only define once

#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include "CMSIS/a2fxxxm3.h"
#include "levitatefpga_hw_platform.h"
#include "customtypes.h"
#include "drivers/mss_ace/mss_ace.h"
#include "drivers_config/mss_ace/ace_handles.h"

typedef struct {
    uint32_t init;
} rangefinder_t;

/**
 * Initialize the range finders and the ACE
 */
void rangefinder_init();

/**
 * Get the value for rangefinder 0 (ADC2)
 */
uint16_t rangefinder_get_rf0();

/**
 * Get the value for rangefinder 1 (ADC3)
 */
uint16_t rangefinder_get_rf1();

/**
 * Get the value for rangefinder 2 (ADC4)
 */
uint16_t rangefinder_get_rf2();

/**
 * Get the value for rangefinder 3 (ADC5)
 */
uint16_t rangefinder_get_rf3();

/**
 * Get the value for rangefinder 4 (ADC6)
 */
uint16_t rangefinder_get_rf4();

/**
 * Get the value for rangefinder 5 (ADC7)
 */
uint16_t rangefinder_get_rf5();


#endif /* RANGEFINDER_H_ */
```

```c
/*
 * rangefinder.c
 *
 *  Created on: Oct 28, 2012
 *      Author: Daniel
 */
#include "rangefinder.h"

static volatile rangefinder_t * rangefinder = RANGEFINDERWRAPPER_0;

ace_channel_handle_t rf_0_handle;
```

```c
ace_channel_handle_t rf_1_handle;
ace_channel_handle_t rf_2_handle;
ace_channel_handle_t rf_3_handle;
ace_channel_handle_t rf_4_handle;
ace_channel_handle_t rf_5_handle;

/**
 * Initialize the range finders
 */
void rangefinder_init() {
  // initalize the ACE
  ACE_init();
  rangefinder->init = 1;
  rf_0_handle = ACE_get_channel_handle((const uint8_t *)"RF_0");
  rf_1_handle = ACE_get_channel_handle((const uint8_t *)"RF_1");
  rf_2_handle = ACE_get_channel_handle((const uint8_t *)"RF_2");
  rf_3_handle = ACE_get_channel_handle((const uint8_t *)"RF_3");
  rf_4_handle = ACE_get_channel_handle((const uint8_t *)"RF_4");
  rf_5_handle = ACE_get_channel_handle((const uint8_t *)"RF_5");
}

/**
 * Get the value for rangefinder 0 (ADC2)
 */
uint16_t rangefinder_get_rf0() {
  return 0xfffC & ACE_get_ppe_sample(rf_0_handle);
}

/**
 * Get the value for rangefinder 1 (ADC3)
 */
uint16_t rangefinder_get_rf1() {
  return 0xfffC & ACE_get_ppe_sample(rf_1_handle);
}

/**
 * Get the value for rangefinder 2 (ADC4)
 */
uint16_t rangefinder_get_rf2() {
  return 0xfffC & ACE_get_ppe_sample(rf_2_handle);
}

/**
 * Get the value for rangefinder 3 (ADC5)
 */
uint16_t rangefinder_get_rf3() {
  return 0xfffC & ACE_get_ppe_sample(rf_3_handle);
}

/**
 * Get the value for rangefinder 4 (ADC6)
 */
uint16_t rangefinder_get_rf4() {
  return 0xfffC & ACE_get_ppe_sample(rf_4_handle);
}

/**
 * Get the value for rangefinder 5 (ADC7)
 */
uint16_t rangefinder_get_rf5() {
  return 0xfffC & ACE_get_ppe_sample(rf_5_handle);
}
```

APPENDIX U
RANGEFINDER2.V

```
`define RESET 0
`define WAIT 1
`define PULSE 2
`define DONE 3

// rangefinder.v
module rangefinder2(
                    input clk,
                    input nreset,
                    input rf_trigger_in,
                    output reg rf_trigger_out,
                    output reg rf_trigger_en,
                    output RF_STATUS_LED
                    );

    reg [1:0] state;
    reg [23:0] counter;

    assign RF_STATUS_LED = rf_trigger_in;

    always@(posedge clk) begin
        if(~nreset) begin
            state <= `RESET;
            rf_trigger_en <= 1'd0;
            rf_trigger_out <= 1'd0;
            counter <= 24'd0;
        end
        else begin
            case (state)
                `RESET: begin
                    state <= `WAIT;
                end
                `WAIT: begin // Wait for 300ms
                    if (counter == 24'd15000000) begin
                        state <= `PULSE;
                    end
                    else begin
                        counter <= counter + 1'b1;
                    end
                end
                `PULSE: begin // Push the output high for at least 20us
                    rf_trigger_en <= 1'd1;
                    rf_trigger_out <= 1'd1;
                    if (counter == 24'd11) begin
                        state <= `DONE;
                    end
                    else begin
                        counter <= counter + 1'b1;
                    end

                end
                `DONE: begin // Do nothing else FOREVER
                    state <= `DONE;
                    rf_trigger_en <= 1'd0;
                    rf_trigger_out <= 1'd0;
                    counter <= 24'd0;
                end
            endcase
        end
    end

endmodule
```

APPENDIX V
RANGEFINDERWRAPPER.V

```verilog
// rangefinderwrapper.v
module rangefinderwrapper(
                            PCLK,
                            PENABLE,
                            PSEL,
                            PRESETN,
                            PWRITE,
                            PREADY,
                            PSLVERR,
                            PADDR,
                            PWDATA,
                            PRDATA,
                            rf_trigger_in ,
                            rf_trigger_out ,
                            rf_trigger_en ,
                            RF_STATUS_LED
                            );

    // APB Bus Interface
    input PCLK, PENABLE, PSEL, PRESETN, PWRITE;
    input  [31:0] PWDATA;
    input  [31:0] PADDR;
    output [31:0] PRDATA;
    output PREADY, PSLVERR;
    output RF_STATUS_LED;

    input rf_trigger_in;
    output rf_trigger_en;
    output rf_trigger_out;

    assign BUS_WRITE_EN = (PENABLE && PWRITE && PSEL);
    assign BUS_READ_EN = (!PWRITE && PSEL); //Data is ready during first cycle to make it
        availble on the bus when PENABLE is asserted

    assign PREADY = 1'b1;
    assign PSLVERR = 1'b0;

    rangefinder3 rangefinder3_0(
                    .clk(PCLK),
                    .nreset(PRESETN),
                    .bus_write_en(BUS_WRITE_EN),
                    .bus_read_en(BUS_READ_EN),
                    .bus_addr(PADDR),
                    .bus_write_data(PWDATA),
                    .bus_read_data(PRDATA),
                    .rf_trigger_in(rf_trigger_in),
                    .rf_trigger_out(rf_trigger_out),
                    .rf_trigger_en(rf_trigger_en),
                    .RF_STATUS_LED(RF_STATUS_LED)
                );

endmodule
```

APPENDIX W
SENSORSTICK.V

```verilog

// sensorstick.v
module sensorstick(
                    output reg REQ_I2C,
                    output reg [6:0] ADDR_I2C,
                    output reg RW_I2C,
                    output reg [7:0] REG_I2C,
                    output reg [7:0] DATA_SIZE_I2C,
                    output reg [7:0] DATA_WRITE_I2C,
                    input [7:0] DATA_READ_I2C,
                    input DATA_READY_I2C,
                    input DONE_I2C,
                    input RDY_I2C,
                    input CLK,
                    input RESETN,
          input RD_RQ, //Complementary filter wants new data to use.
          input XYZ_READ, //Complementary filter has already used current data.
                    output reg XYZ_RDY, //Data for the Complementary filter is ready.
                    //Data from sensors to be used by Kalman Filter.
                    output reg [15:0] ACCEL_DATA_X,
                    output reg [15:0] ACCEL_DATA_Y,
                    output reg [15:0] ACCEL_DATA_Z,
                    output reg [15:0] GYRO_DATA_X,
                    output reg [15:0] GYRO_DATA_Y,
                    output reg [15:0] GYRO_DATA_Z,
                    output reg [15:0] MAGNETO_DATA_X,
                    output reg [15:0] MAGNETO_DATA_Y,
                    output reg [15:0] MAGNETO_DATA_Z
                  );

  // I2C Interface
  //input  DONE_I2C, RDY_I2C, DATA_READY_I2C;
  //input  [7:0] DATA_READ_I2C;
  //output RW_I2C, REQ_I2C;
  //output [7:0] ADDR_I2C;
  //output [7:0] REG_I2C;
  //output [7:0] DATA_WRITE_I2C;

  // Also, you should have an APB wrapper as part of this wrapper so the software can talk to you.

  reg [5:0] state;

  //Addresses
  parameter ACCEL_ADDR = 8'hA6;
  parameter MAGNETO_ADDR = 8'h3C;
  parameter GYRO_ADDR = 8'hD0;
  parameter ACCEL_REG_START = 8'h32;
  parameter MAGNETO_REG_START = 8'h03;
  parameter GYRO_REG_START = 8'h1D;

  //States
  parameter SUSPENDED = 0;
  parameter ACCEL_INIT = 1;
  parameter ACCEL_WAIT = 2;
  parameter MAGNETO_INIT_1 = 3;
  parameter MAGNETO_WAIT_1 = 4;
  parameter MAGNETO_INIT_2 = 5;
  parameter MAGNETO_WAIT_2 = 6;
  parameter MAGNETO_INIT_3 = 7;
  parameter MAGNETO_WAIT_3 = 8;
  parameter GYRO_INIT_1 = 9;
  parameter GYRO_WAIT_1 = 10;
  parameter ACCEL_READ_ACCEPT_X_1 = 11;
  parameter ACCEL_READ_ACCEPT_X_2 = 12;
```

```
  parameter ACCEL_READ_ACCEPT_Y_1 = 13;
  parameter ACCEL_READ_ACCEPT_Y_2 = 14;
  parameter ACCEL_READ_ACCEPT_Z_1 = 15;
  parameter ACCEL_READ_ACCEPT_Z_2 = 16;
  parameter MAGNETO_READ_ACCEPT_X_1 = 17;
  parameter MAGNETO_READ_ACCEPT_X_2 = 18;
  parameter MAGNETO_READ_ACCEPT_Y_1 = 19;
  parameter MAGNETO_READ_ACCEPT_Y_2 = 20;
  parameter MAGNETO_READ_ACCEPT_Z_1 = 21;
  parameter MAGNETO_READ_ACCEPT_Z_2 = 22;
  parameter GYRO_READ_ACCEPT_X_1 = 23;
  parameter GYRO_READ_ACCEPT_X_2 = 24;
  parameter GYRO_READ_ACCEPT_Y_1 = 25;
  parameter GYRO_READ_ACCEPT_Y_2 = 26;
  parameter GYRO_READ_ACCEPT_Z_1 = 27;
  parameter GYRO_READ_ACCEPT_Z_2 = 28;
  parameter ACCEL_READ = 29;
  parameter MAGNETO_READ = 30;
  parameter GYRO_READ_1 = 31;
  parameter ACCEL_READ_WAIT = 32;
  parameter MAGNETO_READ_WAIT = 33;
  parameter GYRO_READ_WAIT = 34;
  parameter GYRO_INIT_2 = 35;
  parameter GYRO_WAIT_2 = 36;
  parameter GYRO_INIT_3 = 37;
  parameter GYRO_WAIT_3 = 38;
  parameter GYRO_INIT_4 = 39;
  parameter GYRO_WAIT_4 = 40;
  parameter GYRO_READ_2 = 41;
  parameter GYRO_READ_3 = 42;

  always@(posedge CLK)
  begin
    if(~RESETN)
      begin
        state <= ACCEL_INIT;
      end
    else
    begin
      case(state)

/*
void IMU_ACCEL_init() {
    // initialize accelerometer
    uint8_t write_buf[2];
    write_buf[0] = 0x2D; // register address
    write_buf[1] = 0x08; // data
    // ask for data
    MSS_I2C_write
    (
        &g_mss_i2c1,
        ACCELEROMETER,
        write_buf,
        2,
        MSS_I2C_RELEASE_BUS
    );

    while(MSS_I2C_wait_complete(&g_mss_i2c1) == MSS_I2C_IN_PROGRESS) {
        __asm__("nop");
    }
}
*/

            ACCEL_INIT:
            begin
                if(RDY_I2C)
                    begin
                        state <= ACCEL_WAIT;
```

```
                        ADDR_I2C <= ACCEL_ADDR[7:1];
                        REG_I2C <= 8'h2D;
                        DATA_WRITE_I2C <= 8'h08;
                        DATA_SIZE_I2C <= 2;
                        REQ_I2C <= 1;
                        RW_I2C <= 0;
                    end
            end

            ACCEL_WAIT:
            begin
                if (DONE_I2C)
                    begin
                        state <= MAGNETO_INIT_1;
                        ADDR_I2C <= 0;
                        REG_I2C <= 0;
                        DATA_WRITE_I2C <= 0;
                        DATA_SIZE_I2C <= 0;
                        REQ_I2C <= 0;
                        RW_I2C <= 0;
                    end
            end

            /*
    void IMU_MAGNETO_init() {
    uint8_t write_buf[2];
    // initialize magnetometer
    write_buf[0] = 0x00; // register address
    write_buf[1] = 0x70; // data
    MSS_I2C_write
    (
        &g_mss_i2c1,
        MAGNETOMETER,
        write_buf,
        2,
        MSS_I2C_RELEASE_BUS
    );

write_buf[0] = 0x01; // register address
    write_buf[1] = 0xA0; // data
    MSS_I2C_write
    (
        &g_mss_i2c1,
        MAGNETOMETER,
        write_buf,
        2,
        MSS_I2C_RELEASE_BUS
    );

    while(MSS_I2C_wait_complete(&g_mss_i2c1) == MSS_I2C_IN_PROGRESS) {
        __asm__("nop");
    }
    write_buf[0] = 0x02; // register address
    write_buf[1] = 0x00; // data
    MSS_I2C_write
    (
        &g_mss_i2c1,
        MAGNETOMETER,
        write_buf,
        2,
        MSS_I2C_RELEASE_BUS
    );

    while(MSS_I2C_wait_complete(&g_mss_i2c1) == MSS_I2C_IN_PROGRESS) {
        __asm__("nop");
    }
            */

```

```
            MAGNETO_INIT_1:
            begin
              if(RDY_I2C)
                    begin
                         state <= MAGNETO_WAIT_1;
                         ADDR_I2C <= MAGNETO_ADDR[7:1];
                         REG_I2C <= 8'h00;
                         DATA_WRITE_I2C <= 8'h70;
                         DATA_SIZE_I2C <= 2;
                         REQ_I2C <= 1;
                         RW_I2C <= 0;
                    end
            end

            MAGNETO_WAIT_1:
            begin
                if(DONE_I2C)
                    begin
                         state <= MAGNETO_INIT_2;
                         ADDR_I2C <= 0;
                         REG_I2C <= 0;
                         DATA_WRITE_I2C <= 0;
                         DATA_SIZE_I2C <= 0;
                         REQ_I2C <= 0;
                         RW_I2C <= 0;
                    end
            end

            MAGNETO_INIT_2:
            begin
              if(RDY_I2C)
                    begin
                         state <= MAGNETO_WAIT_2;
                         ADDR_I2C <= MAGNETO_ADDR[7:1];
                         REG_I2C <= 8'h01;
                         DATA_WRITE_I2C <= 8'hA0;
                         DATA_SIZE_I2C <= 2;
                         REQ_I2C <= 1;
                         RW_I2C <= 0;
                    end
            end

            MAGNETO_WAIT_2:
            begin
                if(DONE_I2C)
                    begin
                         state <= MAGNETO_INIT_3;
                         ADDR_I2C <= 0;
                         REG_I2C <= 0;
                         DATA_WRITE_I2C <= 0;
                         DATA_SIZE_I2C <= 0;
                         REQ_I2C <= 0;
                         RW_I2C <= 0;
                    end
            end

            MAGNETO_INIT_3:
            begin
              if(RDY_I2C)
                    begin
                         state <= MAGNETO_WAIT_3;
                         ADDR_I2C <= MAGNETO_ADDR[7:1];
                         REG_I2C <= 8'h02;
                         DATA_WRITE_I2C <= 8'h00;
                         DATA_SIZE_I2C <= 2;
                         REQ_I2C <= 1;
                         RW_I2C <= 0;
                    end
```

```
            end

            MAGNETO_WAIT_3:
            begin
                if(DONE_I2C)
                    begin
                        state <= GYRO_INIT_1;
                        ADDR_I2C <= 0;
                        REG_I2C <= 0;
                        DATA_WRITE_I2C <= 0;
                        DATA_SIZE_I2C <= 0;
                        REQ_I2C <= 0;
                        RW_I2C <= 0;
                    end
            end
    /*
    void IMU_GYRO_init() {
        // initialize gyro
    uint8_t write_buf[2];
    write_buf[0] = 0x3E; // register address
    write_buf[1] = 0x01; // data
    // ask for data
    MSS_I2C_write
    (
        &g_mss_i2c1,
        GYRO,
        write_buf,
        2,
        MSS_I2C_RELEASE_BUS
    );
    while(MSS_I2C_wait_complete(&g_mss_i2c1) == MSS_I2C_IN_PROGRESS) {
        __asm__("nop");
    }
    */

            GYRO_INIT_1:
            begin
              if(RDY_I2C)
                    begin
                        state <= GYRO_WAIT_1;
                        ADDR_I2C <= GYRO_ADDR[7:1];
                        REG_I2C <= 8'h3E;
                        DATA_WRITE_I2C <= 8'h01;
                        DATA_SIZE_I2C <= 2;
                        REQ_I2C <= 1;
                        RW_I2C <= 0;
                    end
            end

            GYRO_WAIT_1:
            begin
                if (DONE_I2C)
                    begin
                        state <= GYRO_INIT_2;
                        ADDR_I2C <= 0;
                        REG_I2C <= 0;
                        DATA_WRITE_I2C <= 0;
                        DATA_SIZE_I2C <= 0;
                        REQ_I2C <= 0;
                        RW_I2C <= 0;
                    end
            end


            // Write a 19 (0x13) to register 21 (0x15) to set the divider to 19, meaning the Gyro
                 samples at 50Hz.
            GYRO_INIT_2:
            begin
```

```
            if(RDY_I2C)
                begin
                    state <= GYRO_WAIT_2;
                    ADDR_I2C <= GYRO_ADDR[7:1];
                    REG_I2C <= 8'h15;
                    DATA_WRITE_I2C <= 8'h13;
                    DATA_SIZE_I2C <= 2;
                    REQ_I2C <= 1;
                    RW_I2C <= 0;
                end
        end

        GYRO_WAIT_2:
        begin
            if (DONE_I2C)
                begin
                    state <= GYRO_INIT_3;
                    ADDR_I2C <= 0;
                    REG_I2C <= 0;
                    DATA_WRITE_I2C <= 0;
                    DATA_SIZE_I2C <= 0;
                    REQ_I2C <= 0;
                    RW_I2C <= 0;
                end
        end


        // Write a 25 (0x19) to register 22 (0x16) to set the internal rate to 1kHz and not
            get bogus data.
        GYRO_INIT_3:
        begin
            if(RDY_I2C)
                begin
                    state <= GYRO_WAIT_3;
                    ADDR_I2C <= GYRO_ADDR[7:1];
                    REG_I2C <= 8'h16;
                    DATA_WRITE_I2C <= 8'b11011;
                    DATA_SIZE_I2C <= 2;
                    REQ_I2C <= 1;
                    RW_I2C <= 0;
                end
        end

        GYRO_WAIT_3:
        begin
            if (DONE_I2C)
                begin
                    state <= GYRO_INIT_4;
                    ADDR_I2C <= 0;
                    REG_I2C <= 0;
                    DATA_WRITE_I2C <= 0;
                    DATA_SIZE_I2C <= 0;
                    REQ_I2C <= 0;
                    RW_I2C <= 0;
                end
        end


        // Write a 1 (0x1) to register 23 (0x17) to set the interrupt whenever data is ready.
              Interrupt is cleared upon any reg read.
        GYRO_INIT_4:
        begin
            if(RDY_I2C)
                begin
                    state <= GYRO_WAIT_4;
                    ADDR_I2C <= GYRO_ADDR[7:1];
                    REG_I2C <= 8'h17;
                    DATA_WRITE_I2C <= 8'h1;
```

```
                        DATA_SIZE_I2C <= 2;
                        REQ_I2C <= 1;
                        RW_I2C <= 0;
                    end
            end

            GYRO_WAIT_4:
            begin
                if (DONE_I2C)
                    begin
                        state <= SUSPENDED;
                        ADDR_I2C <= 0;
                        REG_I2C <= 0;
                        DATA_WRITE_I2C <= 0;
                        DATA_SIZE_I2C <= 0;
                        REQ_I2C <= 0;
                        RW_I2C <= 0;
                    end
            end

    /*
    vector3u16_t IMU_ACCEL_read() {
        vector3u16_t volatile data;
        MSS_I2C_write_read
        (
            &g_mss_i2c1,
            ACCELEROMETER,
            &(ACCELEROMETER_REG_START),    // the register that starts the data registers
            1,
            (uint8_t*)&data,
            6,
            MSS_I2C_RELEASE_BUS
        );

        while(MSS_I2C_wait_complete(&g_mss_i2c1) == MSS_I2C_IN_PROGRESS) {
            __asm__("nop");
        }

        return data;
    }
    */
            ACCEL_READ:
            begin
                if(RDY_I2C)
                    begin
                        state <= ACCEL_READ_ACCEPT_X_1;
                        ADDR_I2C <= ACCEL_ADDR[7:1];
                        RW_I2C <= 1;
                        REQ_I2C <= 1;
                        REG_I2C <= ACCEL_REG_START;
                        DATA_SIZE_I2C <= 6;
                    end
            end

            ACCEL_READ_ACCEPT_X_1:
            begin
                if(DATA_READY_I2C)
                begin
                    ACCEL_DATA_Y[7:0] <= DATA_READ_I2C;
                    state <= ACCEL_READ_ACCEPT_X_2;
                end
            end

            ACCEL_READ_ACCEPT_X_2:
            begin
                if(DATA_READY_I2C)
                begin
                    ACCEL_DATA_Y[15:8] <= DATA_READ_I2C;
```

```
                    state <= ACCEL_READ_ACCEPT_Y_1;
                end
            end

            ACCEL_READ_ACCEPT_Y_1:
            begin
                if(DATA_READY_I2C)
                begin
                    ACCEL_DATA_X[7:0] <= DATA_READ_I2C;
                    state <= ACCEL_READ_ACCEPT_Y_2;
                end
            end

            ACCEL_READ_ACCEPT_Y_2:
            begin
                if(DATA_READY_I2C)
                begin
                    ACCEL_DATA_X[15:8] <= DATA_READ_I2C;
                    state <= ACCEL_READ_ACCEPT_Z_1;
                end
            end

            ACCEL_READ_ACCEPT_Z_1:
            begin
                if(DATA_READY_I2C)
                begin
                    ACCEL_DATA_Z[7:0] <= DATA_READ_I2C;
                    state <= ACCEL_READ_ACCEPT_Z_2;
                end
            end

            ACCEL_READ_ACCEPT_Z_2:
            begin
                if(DATA_READY_I2C)
                begin
                    ACCEL_DATA_Z[15:8] <= DATA_READ_I2C;
                    state <= ACCEL_READ_WAIT;
                end
            end

            ACCEL_READ_WAIT:
            begin
                if (DONE_I2C)
                    begin
                        state <= MAGNETO_READ;
                        ADDR_I2C <= 0;
                        REG_I2C <= 0;
                        DATA_WRITE_I2C <= 0;
                        DATA_SIZE_I2C <= 0;
                        REQ_I2C <= 0;
                        RW_I2C <= 0;
                    end
            end

/*
vector3u16_t IMU_MAGNETO_read() {
    vector3u16_t volatile data;
    // read the data
    MSS_I2C_write_read
    (
        &g_mss_i2c1,
        MAGNETOMETER,
        &(MAGNETOMETER_REG_START),   // the register that starts the data registers
        1,
        (uint8_t*)&data,
        6,
        MSS_I2C_RELEASE_BUS
    );
```

```
    while(MSS_I2C_wait_complete(&g_mss_i2c1) == MSS_I2C_IN_PROGRESS) {
        __asm__("nop");
    }

    // change endianess
    data.x = (data.x << 8) | ((data.x >> 8) & 0x00ff);
    data.y = (data.y << 8) | ((data.y >> 8) & 0x00ff);
    data.z = (data.z << 8) | ((data.z >> 8) & 0x00ff);

    return data;
}
*/

            MAGNETO_READ:
            begin
                if(RDY_I2C)
                    begin
                        state <= MAGNETO_READ_ACCEPT_X_1;
                        ADDR_I2C <= MAGNETO_ADDR[7:1];
                        RW_I2C <= 1;
                        REQ_I2C <= 1;
                        REG_I2C <= MAGNETO_REG_START;
                        DATA_SIZE_I2C <= 6;
                    end
            end

            MAGNETO_READ_ACCEPT_X_1:
            begin
                if(DATA_READY_I2C)
                begin
                    MAGNETO_DATA_X[7:0] <= DATA_READ_I2C;
                    state <= MAGNETO_READ_ACCEPT_X_2;
                end
            end

            MAGNETO_READ_ACCEPT_X_2:
            begin
                if(DATA_READY_I2C)
                begin
                    MAGNETO_DATA_X[15:8] <= DATA_READ_I2C;
                    state <= MAGNETO_READ_ACCEPT_Y_1;
                end
            end

            MAGNETO_READ_ACCEPT_Y_1:
            begin
                if(DATA_READY_I2C)
                begin
                    MAGNETO_DATA_Y[7:0] <= DATA_READ_I2C;
                    state <= MAGNETO_READ_ACCEPT_Y_2;
                end
            end

            MAGNETO_READ_ACCEPT_Y_2:
            begin
                if(DATA_READY_I2C)
                begin
                    MAGNETO_DATA_Y[15:8] <= DATA_READ_I2C;
                    state <= MAGNETO_READ_ACCEPT_Z_1;
                end
            end

            MAGNETO_READ_ACCEPT_Z_1:
            begin
                if(DATA_READY_I2C)
                begin
                    MAGNETO_DATA_Z[7:0] <= DATA_READ_I2C;
```

```
                    state <= MAGNETO_READ_ACCEPT_Z_2;
                end
            end

            MAGNETO_READ_ACCEPT_Z_2:
            begin
                if(DATA_READY_I2C)
                begin
                    MAGNETO_DATA_Z[15:8] <= DATA_READ_I2C;
                    state <= MAGNETO_READ_WAIT;
                end
            end

            MAGNETO_READ_WAIT:
            begin
                if (DONE_I2C)
                    begin
                        state <= GYRO_READ_1;
                        ADDR_I2C <= 0;
                        REG_I2C <= 0;
                        DATA_WRITE_I2C <= 0;
                        DATA_SIZE_I2C <= 0;
                        REQ_I2C <= 0;
                        RW_I2C <= 0;
                    end
            end


/*
vector3u16_t IMU_GYRO_read() {
    vector3u16_t volatile data;
    MSS_I2C_write_read
    (
        &g_mss_i2c1,
        GYRO,
        &(GYRO_REG_START),    // the register that starts the data registers
        1,
        (uint8_t*)&data,
        6,
        MSS_I2C_RELEASE_BUS
    );

    while(MSS_I2C_wait_complete(&g_mss_i2c1) == MSS_I2C_IN_PROGRESS) {
        __asm__("nop");
    }

    // change endianess
    data.x = (data.x << 8) | ((data.x >> 8) & 0x00ff);
    data.y = (data.y << 8) | ((data.y >> 8) & 0x00ff);
    data.z = (data.z << 8) | ((data.z >> 8) & 0x00ff);


    return data;
}
*/

            // Check status register to see if new data is available.
      GYRO_READ_1:
      begin
        if(RDY_I2C)
          begin
            state <= GYRO_READ_2;
            ADDR_I2C <= GYRO_ADDR[7:1];
            RW_I2C <= 1;
            REQ_I2C <= 1;
            REG_I2C <= 8'h1a;
            DATA_SIZE_I2C <= 1;
          end
```

```
        end

        GYRO_READ_2:
        begin
          if(DATA_READY_I2C)
          begin
                        if(DATA_READ_I2C[0])
              begin
                state <= GYRO_READ_3;
              end
                        else
                        begin
                state <= GYRO_READ_1;
                        end
                        ADDR_I2C <= 0;
            RW_I2C <= 0;
            REQ_I2C <= 0;
            REG_I2C <= 0;
            DATA_SIZE_I2C <= 0;
          end
        end

        GYRO_READ_3:
        begin
                    if(RDY_I2C)
          begin
            state <= GYRO_READ_ACCEPT_X_1;
            ADDR_I2C <= GYRO_ADDR[7:1];
            RW_I2C <= 1;
            REQ_I2C <= 1;
            REG_I2C <= GYRO_REG_START;
            DATA_SIZE_I2C <= 6;
          end
        end

        GYRO_READ_ACCEPT_X_1:
        begin
          if(DATA_READY_I2C)
          begin
            GYRO_DATA_X[15:8] <= DATA_READ_I2C;
            state <= GYRO_READ_ACCEPT_X_2;
          end
        end

        GYRO_READ_ACCEPT_X_2:
        begin
          if(DATA_READY_I2C)
          begin
            GYRO_DATA_X[7:0] <= DATA_READ_I2C;
            state <= GYRO_READ_ACCEPT_Y_1;
          end
        end

        GYRO_READ_ACCEPT_Y_1:
        begin
          if(DATA_READY_I2C)
          begin
            GYRO_DATA_Y[15:8] <= DATA_READ_I2C;
            state <= GYRO_READ_ACCEPT_Y_2;
          end
        end

        GYRO_READ_ACCEPT_Y_2:
        begin
          if(DATA_READY_I2C)
          begin
            GYRO_DATA_Y[7:0] <= DATA_READ_I2C;
            state <= GYRO_READ_ACCEPT_Z_1;
```

```
          end
        end

        GYRO_READ_ACCEPT_Z_1:
        begin
          if(DATA_READY_I2C)
          begin
            GYRO_DATA_Z[15:8] <= DATA_READ_I2C;
            state <= GYRO_READ_ACCEPT_Z_2;
          end
        end

        GYRO_READ_ACCEPT_Z_2:
        begin
          if(DATA_READY_I2C)
          begin
            GYRO_DATA_Z[7:0] <= DATA_READ_I2C;
            state <= GYRO_READ_WAIT;
          end
        end

        GYRO_READ_WAIT:
        begin
          if (DONE_I2C)
            begin
              state <= SUSPENDED;
              ADDR_I2C <= 0;
              REG_I2C <= 0;
              DATA_WRITE_I2C <= 0;
              DATA_SIZE_I2C <= 0;
              REQ_I2C <= 0;
              RW_I2C <= 0;
                          XYZ_RDY <= 1;
            end
        end

        // Might want to add else's to if(DATA_READY_I2C) that check if DONE/ERROR from I2C, go to
            suspended and output an error bit to Kalman so it knows it didn't get new data.

        SUSPENDED:
        begin
          if(RD_RQ && RDY_I2C)
          begin
            state <= ACCEL_READ;
            ADDR_I2C <= 0;
            REG_I2C <= 0;
            DATA_WRITE_I2C <= 0;
            DATA_SIZE_I2C <= 0;
            REQ_I2C <= 0;
                      XYZ_RDY <= 0;
          end
                  else if (XYZ_READ)
                  begin
                      XYZ_RDY <= 0;
                  end
        end

      endcase
    end
  end

endmodule
```

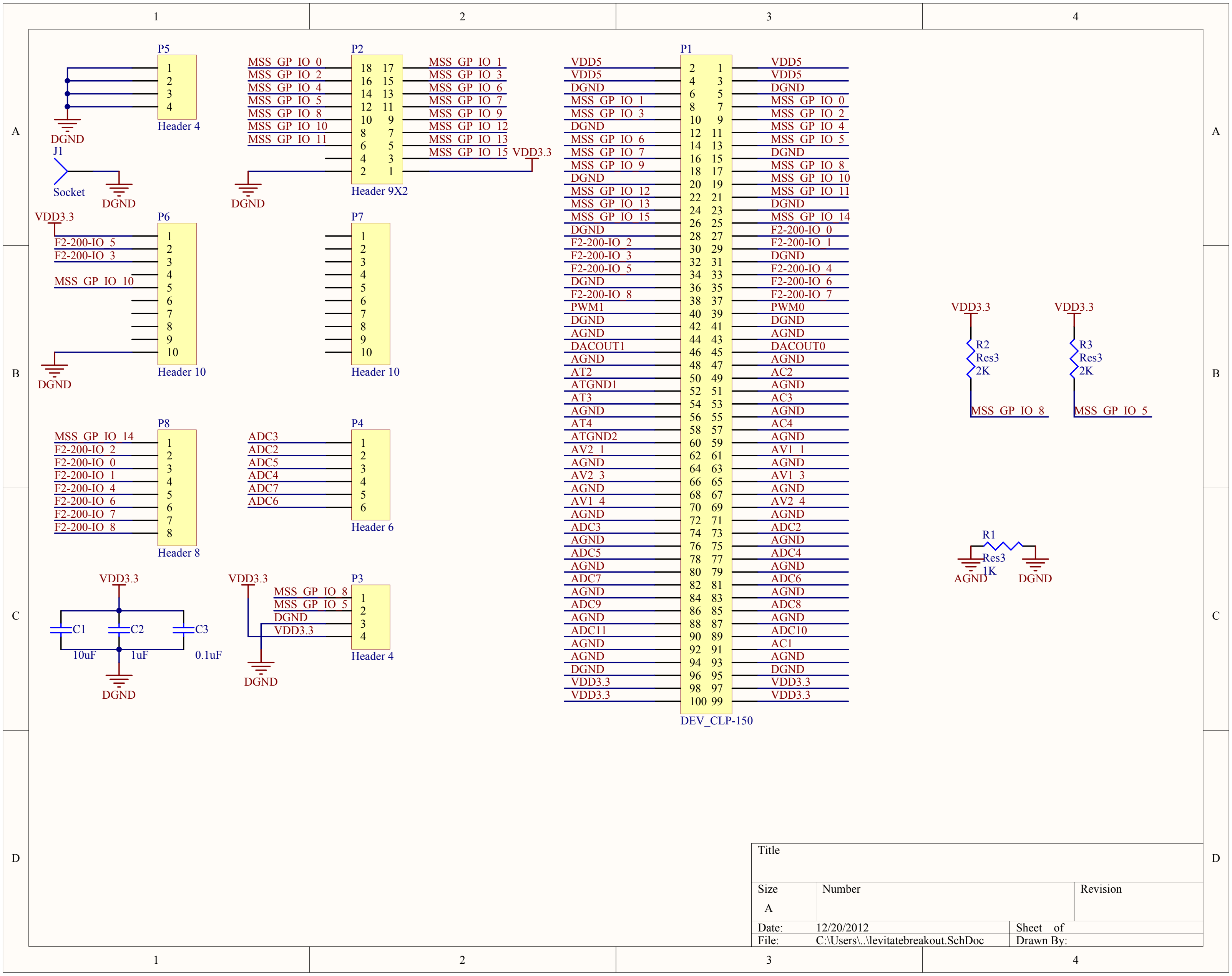
APPENDIX X
SENSORSTICKWRAPPER.V

```verilog

// sensorstickwrapper.v
module sensorstickwrapper(
                              output REQ_I2C,
                             output [6:0] ADDR_I2C,
                             output RW_I2C,
                             output [7:0] REG_I2C,
                             output [7:0] DATA_SIZE_I2C,
                             output [7:0] DATA_WRITE_I2C,
                             input [7:0] DATA_READ_I2C,
                             input DATA_READY_I2C,
                             input DONE_I2C,
                             input RDY_I2C,


              input RD_RQ, //Complementary filter wants new data to use.
              input XYZ_READ, //Complementary filter has already used current data.
              output XYZ_RDY, //Data for the Complementary filter is ready.

                             input PCLK,
                             input PENABLE,
                             input PSEL,
                             input PRESETN,
                             input PWRITE,
                             output PREADY,
                             output PSLVERR,
                             input [31:0] PADDR,
                             input [31:0] PWDATA,
                             output reg [31:0] PRDATA,

                             output [15:0] ACCEL_DATA_X,
                             output [15:0] ACCEL_DATA_Y,
                             output [15:0] ACCEL_DATA_Z,
                             output [15:0] GYRO_DATA_X,
                             output [15:0] GYRO_DATA_Y,
                             output [15:0] GYRO_DATA_Z,
                             output [15:0] MAGNETO_DATA_X,
                             output [15:0] MAGNETO_DATA_Y,
                             output [15:0] MAGNETO_DATA_Z
              );

  assign BUS_WRITE_EN = (PENABLE && PWRITE && PSEL);
  assign BUS_READ_EN = (!PWRITE && PSEL); //Data is ready during first cycle to make it availble on the bus when PENABLE is asserted

  assign PREADY = 1'b1;
  assign PSLVERR = 1'b0;

  always@(posedge PCLK) begin
    if(~PRESETN) begin
      // nothing to do on reset
    end
    else begin
      if(BUS_READ_EN)
      begin
        case(PADDR[5:2])
        4'b0000:
        begin
          PRDATA[31:0] <= {16'h0, ACCEL_DATA_X};
        end

        4'b0001:
        begin
          PRDATA <= {16'h0, ACCEL_DATA_Y};
        end
```

```

        4'b0010:
        begin
          PRDATA <= {16'h0, ACCEL_DATA_Z};
        end

        4'b0011:
        begin
          PRDATA <= {16'h0, GYRO_DATA_X};
        end

        4'b0100:
        begin
          PRDATA <= {16'h0, GYRO_DATA_Y};
        end

        4'b0101:
        begin
          PRDATA <= {16'h0, GYRO_DATA_Z};
        end

        4'b0110:
        begin
          PRDATA <= {16'h0, MAGNETO_DATA_X};
        end

        4'b0111:
        begin
          PRDATA <= {16'h0, MAGNETO_DATA_Y};
        end

        4'b1000:
        begin
          PRDATA <= {16'h0, MAGNETO_DATA_Z};
        end
        endcase
      end
    end
  end

  sensorstick sensorstick_0(
    .REQ_I2C(REQ_I2C),
    .ADDR_I2C(ADDR_I2C),
    .RW_I2C(RW_I2C),
    .REG_I2C(REG_I2C),
    .DATA_SIZE_I2C(DATA_SIZE_I2C),
    .DATA_WRITE_I2C(DATA_WRITE_I2C),
    .DATA_READ_I2C(DATA_READ_I2C),
    .DATA_READY_I2C(DATA_READY_I2C),
    .DONE_I2C(DONE_I2C),
    .RDY_I2C(RDY_I2C),
    .CLK(PCLK),
    .RESETN(PRESETN),
    .RD_RQ(RD_RQ), //Kalman filter wants new data to use.
                          //Data from sensor
    .XYZ_READ(XYZ_READ), //Complementary filter has already used current data.
        .XYZ_RDY(XYZ_RDY), //Data for the Complementary filter is ready.

    .ACCEL_DATA_X(ACCEL_DATA_X),
    .ACCEL_DATA_Y(ACCEL_DATA_Y),
    .ACCEL_DATA_Z(ACCEL_DATA_Z),
    .GYRO_DATA_X(GYRO_DATA_X),
    .GYRO_DATA_Y(GYRO_DATA_Y),
    .GYRO_DATA_Z(GYRO_DATA_Z),
    .MAGNETO_DATA_X(MAGNETO_DATA_X),
    .MAGNETO_DATA_Y(MAGNETO_DATA_Y),
    .MAGNETO_DATA_Z(MAGNETO_DATA_Z)
  );
```

```
endmodule
```

Appendix Y
Levitate Daughterboard Schematic



LEVITATE DAUGHTERBOARD PCB