

UNIVERSITY OF UTAH

SENIOR PROJECT

Project Levitate

Author:

Leif Andersen
Daniel Blakemore
Jon Parker

Supervisor:

Al Davis

December 21, 2012

CONTENTS

I	Introduction	3
II	Functional Specification	3
II-A	Stabilization	3
II-B	Collision Avoidance	3
II-C	Object Tracking	4
III	Components	4
III-A	Quadrotor Chassis	4
III-B	SmartFusion Evaluation Kit	5
III-C	Rangefinders	6
III-D	Camera	6
III-E	Radio	7
III-F	Inertial Measurement Unit	7
IV	Hardware Implementation	8
IV-A	Stabilization Hardware	8
IV-B	Locomotion	9
IV-C	Rangefinders	9
IV-D	Camera	9
V	Software Implementation	9
V-A	Minimum Safe Distance Controller	10
V-B	Image Processing	10
V-C	Flight Loop	11
VI	Difficulties Encountered	13
VI-A	FPGA Size	13
VI-B	Engineering Day Incident	14
VI-C	Vibration, Sensor Noise, and Weight	14
VI-D	Battery Life	14
VI-E	Image Processing Android Application	14
VII	Bill of Materials	15
VIII	Conclusion	15
	Appendix A: autopilot.h/c	17
	Appendix B: CameraActivity.java	22
	Appendix C: cameradatabus.v	24
	Appendix D: CameraOverlay.java	25
	Appendix E: CameraPreview.java	27
	Appendix F: camerawrapper.v	34
	Appendix G: complementaryfilter.v	38
	Appendix H: complementaryfilterwrapper.v	41

Appendix I: customtypes.h	44
Appendix J: filter.h/c	45
Appendix K: i2cinterface.v	48
Appendix L: imudriver.h/c	66
Appendix M: main.c	67
Appendix N: motors2.h/c	68
Appendix O: motorswrapper.v	72
Appendix P: motors2.v	73
Appendix Q: pid.h/c	75
Appendix R: radiodriver.h/c	78
Appendix S: radiowrapper.v	80
Appendix T: rangefinder.h/c	84
Appendix U: rangefinder2.v	86
Appendix V: rangefinderwrapper.v	87
Appendix W: sensorstick.v	88
Appendix X: sensorstickwrapper.v	100
Appendix Y: Levitate Daughterboard Schematic	103
Appendix Z: Levitate Daughterboard PCB	104

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 FPGA 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

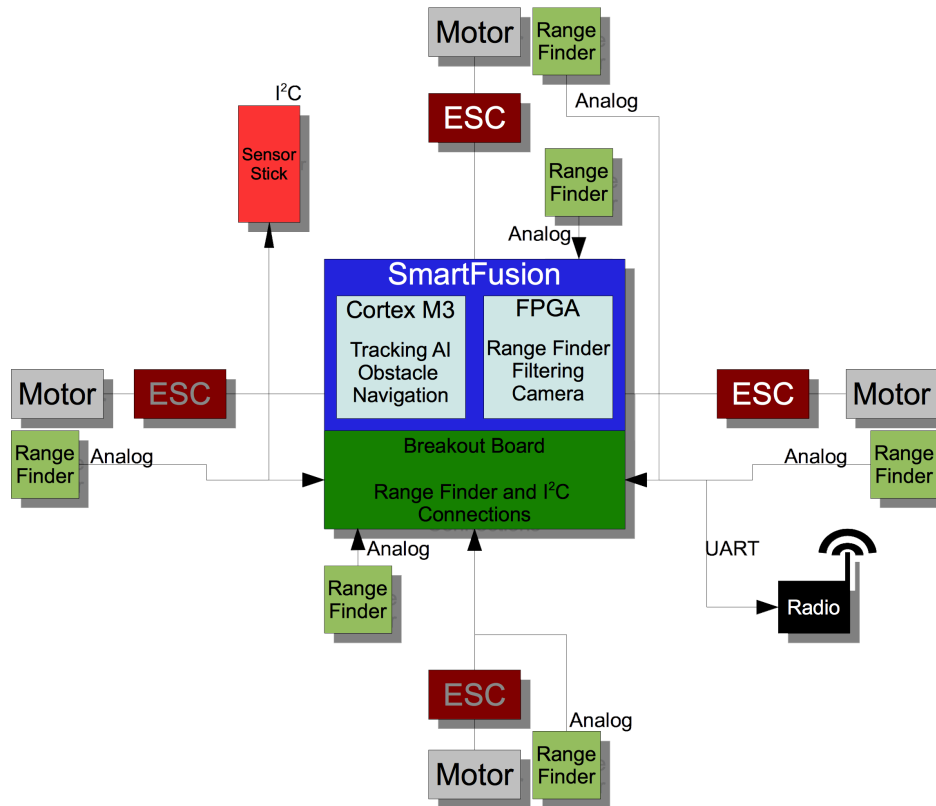


Fig. 1: General hardware block diagram.

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:

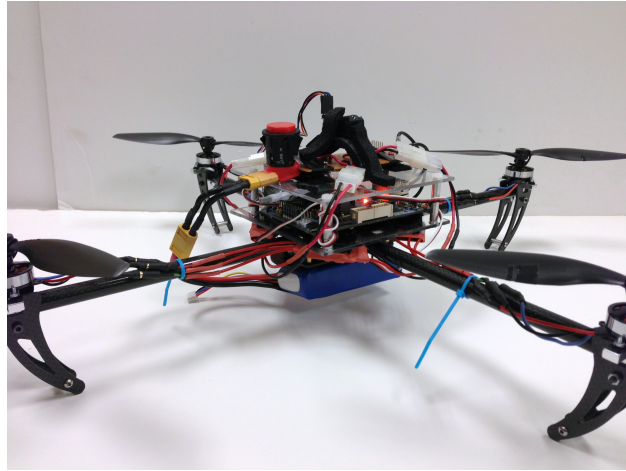


Fig. 2: Preliminary quadrotor platform.

B. SmartFusion Evaluation Kit

The SmartFusion Evaluation Kit is a microcontroller unit consisting of an ARM Cortex-M3, and a 200K-gate (4608 LUT) FPGA¹, 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.

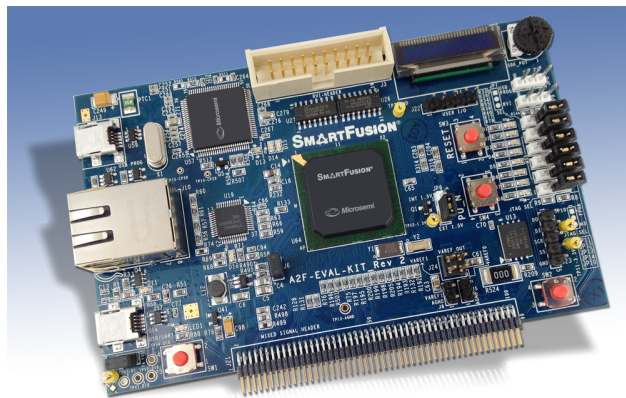


Fig. 3: SmartFusion Evaluation Kit.

¹This 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 formats². 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.

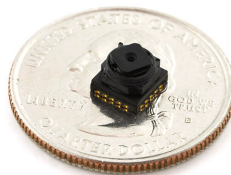


Fig. 6: Camera.

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

²We 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 magnetometer (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).

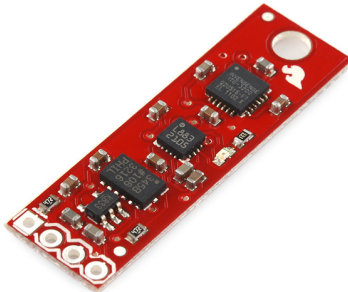


Fig. 8: Nine Degree of Freedom Sensor Stick.

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 doesn't 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:

$$\text{orientation} = .98 \times (\text{previous_orientation} + \text{scaled_gyro_data}) + .02 \times \text{scaled_accelerometer_data}$$

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.

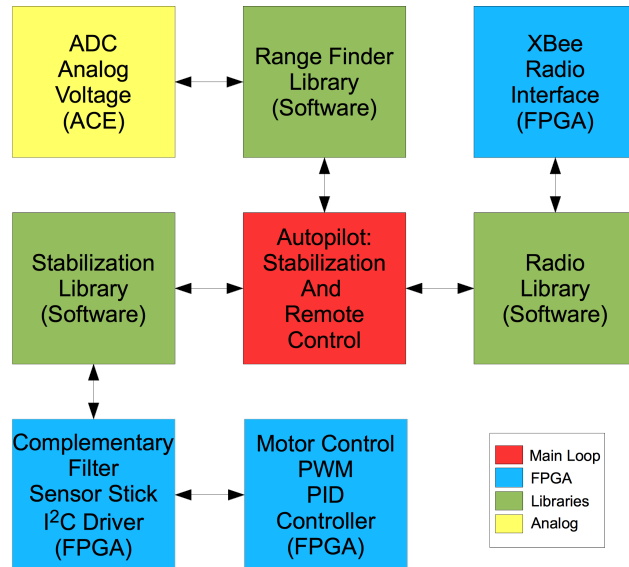


Fig. 9: General software block diagram.

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 CPU’s 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 target’s 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.

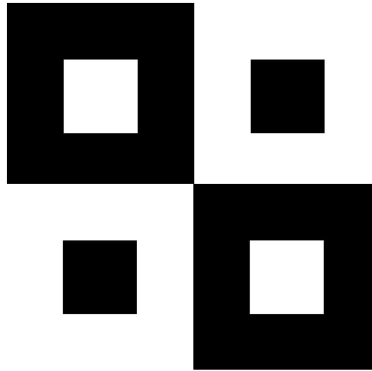


Fig. 10: Camera Target.

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 simply 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.

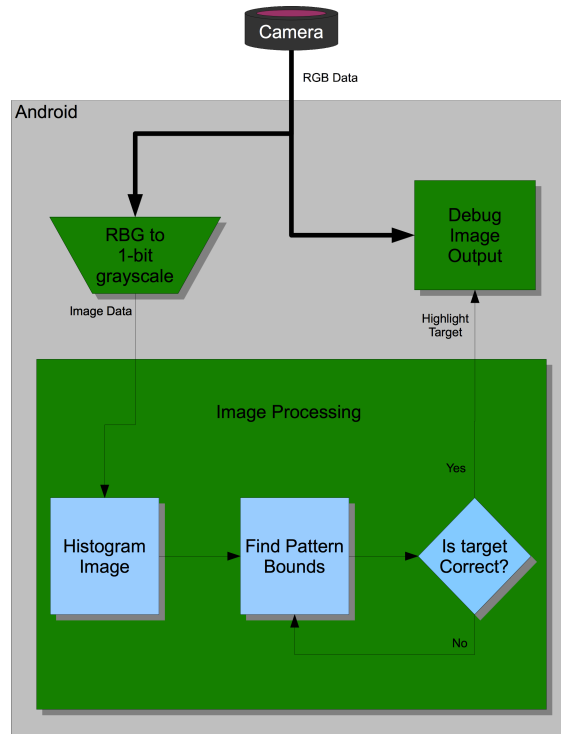


Fig. 11: Camera interfacing.

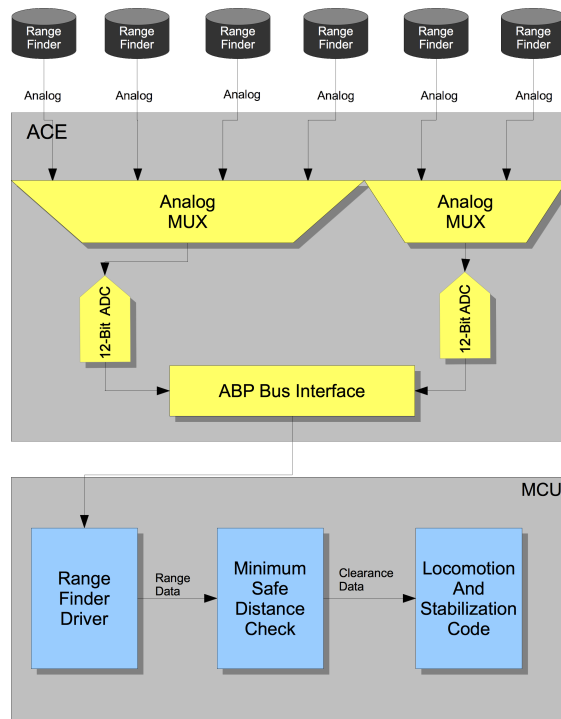


Fig. 12: Rangefinder interfacing.

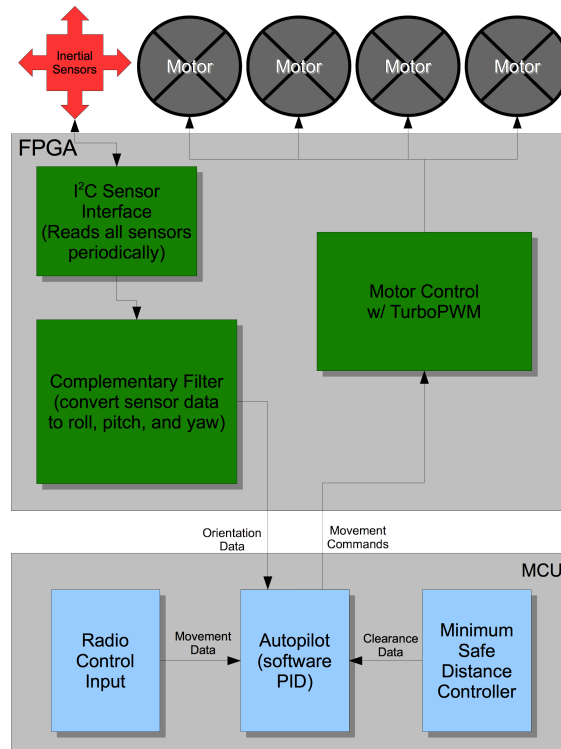


Fig. 13: Interfacing for the motors, IMU, stabilization, and locomotion to flight loop.

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

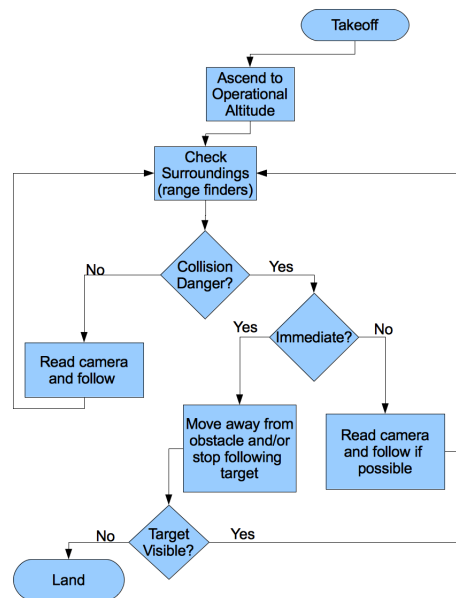


Fig. 14: Software run loop.

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 than 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 internet³.

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.

³Shout 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.digkey.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 anticipated⁴.

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

⁴Also 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

```
1 #ifndef AUTOPILOT_H_ // Only define once
2 #define AUTOPILOT_H_ // Only define once
3
4 #include "customtypes.h"
5 #include "motors2.h"
6 #include "radiodriver.h"
7 #include "pid.h"
8 #include "filter.h"
9
10 int MOVE_DURATION, MOVE_COUNTER, MOVE_RATE;
11
12 bool ready;
13
14 void autopilot_initialize ();
15 void autopilot_update ();
16 void remoteControl ();
17
18 #endif /* AUTOPILOT_H_ */

1 #include "autopilot.h"
2
3 enum {MOVE_i, MOVE_k, MOVE_j, MOVE_l} MOVE_STATE;
4
5 void autopilot_initialize ()
6 {
7     MOVE_STATE = MOVE_i;
8     MOVE_COUNTER = 0;
9     MOVE_RATE = 4;
10    MOVE_DURATION = 2;
11 }
12
13 void autopilot_update ()
14 {
15     if (MOVE_COUNTER == 1)
16     {
17         switch(MOVE_STATE)
18         {
19             case MOVE_i:
20                 filter_modify_pitch_offset(MOVE_RATE);
21                 break;
22
23             case MOVE_k:
24                 filter_modify_pitch_offset(-MOVE_RATE);
25                 break;
26
27             case MOVE_j:
28                 filter_modify_roll_offset(MOVE_RATE);
29                 break;
30
31             case MOVE_l:
32                 filter_modify_roll_offset(-MOVE_RATE);
33                 break;
34         }
35     }
36     if (MOVE_COUNTER > 0)
37         MOVE_COUNTER--;
38
39 }
40
41 void remoteControl ()
42 {
43     if (radio_didReadChar ()) {
44         char gotchar = radio_getChar ();
45
46         switch(gotchar) {
```

```

47 case 'w':
48     // throttle up
49     speed1 += 500;
50     speed2 += 500;
51     speed3 += 500;
52     speed4 += 500;
53     //char foo[512];
54     //int messSize = sprintf(foo, "\n%i\n", speed1);
55     //radio_sendStrOverRadio(foo, messSize);
56     break;
57 case 's':
58     // throttle down
59     speed1 -= 500;
60     speed2 -= 500;
61     speed3 -= 500;
62     speed4 -= 500;
63     if (speed1 < 50000)
64         speed1 = 50000;
65     if (speed2 < 50000)
66         speed2 = 50000;
67     if (speed3 < 50000)
68         speed3 = 50000;
69     if (speed4 < 50000)
70         speed4 = 50000;
71     break;
72 case 'a':
73     // yaw one way
74     yawUpdate(-1 * multiplier);
75     break;
76 case 'd':
77     // yaw the other way
78     yawUpdate(1 * multiplier);
79     break;
80
81 case 'g':
82     // pitch forward
83     // MOTOR_1 Up = -roll
84     // MOTOR_2 Up = -pitch
85     // MOTOR_3 Up = +roll
86     // MOTOR_4 Up = +pitch
87     // Motor 2 represents the front of the quadrotor.
88     // pitch forward means tipping motor 2 down. So add a negative number to offset the
89     // positive that
90     // motor 2 would read.
91     filter_modify_pitch_offset(-1*multiplier); //Pitch forward by 240.
92     break;
93 case 'b':
94     // pitch back
95     filter_modify_pitch_offset(1*multiplier); //Pitch back by 240.
96     break;
97 case 'v':
98     // roll left
99     //roll left means lower motor 2, so motor two will be reading positives, add a negative to
100     //offset.
101     filter_modify_roll_offset(-1*multiplier); // roll left by 240.
102     break;
103 case 'n':
104     // roll right
105     filter_modify_roll_offset(1*multiplier); // roll right by 240.
106     break;
107 case 'i':
108     // pitch forward
109     // MOTOR_1 Up = -roll
110     // MOTOR_2 Up = -pitch
111     // MOTOR_3 Up = +roll
112     // MOTOR_4 Up = +pitch
113     // Motor 2 represents the front of the quadrotor.

```

```

113         // pitch forward means tipping motor 2 down. So add a negative number to offset the
114         // positive that
115         // motor 2 would read.
116         if(MOVE_COUNTER == 0)
117         {
118             MOVE_STATE = MOVE_j;
119             MOVE_COUNTER = MOVE_DURATION;
120             filter_modify_pitch_offset(-MOVE_RATE); //Pitch forward by 240.
121         }
122         else if (MOVE_STATE == MOVE_i)
123             MOVE_COUNTER = MOVE_DURATION;
124         break;
125     case 'k':
126         // pitch back
127         if(MOVE_COUNTER == 0)
128         {
129             MOVE_STATE = MOVE_k;
130             MOVE_COUNTER = MOVE_DURATION;
131             filter_modify_pitch_offset(MOVE_RATE); //Pitch back by 240.
132         }
133         else if (MOVE_STATE == MOVE_k)
134             MOVE_COUNTER = MOVE_DURATION;
135         break;
136     case 'j':
137         // roll left
138         //roll left means lower motor 2, so motor two will be reading positives, add a negative to
139         //offset.
140         if(MOVE_COUNTER == 0)
141         {
142             MOVE_STATE = MOVE_j;
143             MOVE_COUNTER = MOVE_DURATION;
144             filter_modify_roll_offset(-MOVE_RATE); // roll left by 240.
145         }
146         else if (MOVE_STATE == MOVE_j)
147             MOVE_COUNTER = MOVE_DURATION;
148         break;
149     case 'l':
150         // roll right
151         if(MOVE_COUNTER == 0)
152         {
153             MOVE_STATE = MOVE_l;
154             MOVE_COUNTER = MOVE_DURATION;
155             filter_modify_roll_offset(MOVE_RATE); // roll right by 240.
156         }
157         else if (MOVE_STATE == MOVE_l)
158             MOVE_COUNTER = MOVE_DURATION;
159         break;
160     case 't':
161         // take off
162         ready = true;
163         break;
164     case '-':
165         // governor up
166         governor -= 5000;
167         if (governor < 50000)
168             governor = 50000;
169         break;
170     case '=':
171         // governor down
172         governor += 5000;
173         break;
174     // test each motor
175     case '1':
176         speed1 = 55000;
177         speed2 = speed3 = speed4 = 0;
178         break;
179     case '2':

```

```

179     speed2 = 55000;
180     speed1 = speed3 = speed4 = 0;
181     break;
182 case '3':
183     speed3 = 5000;
184     speed1 = speed2 = speed4 = 0;
185     break;
186 case '4':
187     speed4 = 5000;
188     speed1 = speed2 = speed3 = 0;
189     break;
190 // ALL STOP
191 case ' ':
192     ready = false;
193     speed1 = speed2 = speed3 = speed4 = THROTTLE_MIN;
194     break;
195
196 case '5':
197     P_CONST -= 1 * multiplier;
198     if(P_CONST < 0)
199         P_CONST = 0;
200     break;
201
202 case '6':
203     P_CONST += 1 * multiplier;
204     break;
205
206 case '7':
207     I_CONST -= 1 * multiplier;
208     if(I_CONST < 0)
209         I_CONST = 0;
210     break;
211
212 case '8':
213     I_CONST += 1 * multiplier;
214     break;
215
216 case '9':
217     D_CONST -= 1 * multiplier;
218     if(D_CONST < 0)
219         D_CONST = 0;
220     break;
221
222 case '0':
223     D_CONST += 1 * multiplier;
224     break;
225
226 case 'z':
227     PID_GOVERNOR -= 500;
228     if(PID_GOVERNOR < 0)
229         PID_GOVERNOR = 0;
230     break;
231
232 case 'x':
233     PID_GOVERNOR += 500;
234     break;
235
236 case ' ':
237     multiplier = 1;
238     break;
239
240 case '/':
241     multiplier *= 10;
242     break;
243
244 case 'm':
245     ;
246     char foo[512];

```

```
247     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);
248     radio_sendStrOverRadio(foo, messSize);
249     break;
250
251     case 'h':
252         ORIENTATION_GOVERNOR -= multiplier;
253         break;
254
255     case 'y':
256         ORIENTATION_GOVERNOR += multiplier;
257         break;
258     }
259
260     if (!ready) {
261         setMotorSpeed(MOTOR_1, speed1);
262         setMotorSpeed(MOTOR_2, speed2);
263         setMotorSpeed(MOTOR_3, speed3);
264         setMotorSpeed(MOTOR_4, speed4);
265     }
266 }
267 }
```

APPENDIX B
CAMERA_ACTIVITY.JAVA

```
1 package com.projectlevitate.quadcamdroid;
2
3 import java.util.List;
4
5 import android.app.Activity;
6 import android.content.pm.ActivityInfo;
7 import android.hardware.Camera;
8 import android.hardware.Camera.Size;
9 import android.os.Bundle;
10 import android.view.Window;
11 import android.view.WindowManager;
12 import android.view.WindowManager.LayoutParams;
13 import android.widget.FrameLayout;
14 import android.widget.Toast;
15
16 public class CameraActivity extends Activity {
17     // Our variables
18     CameraPreview cv;
19     CameraOverlay dv;
20     FrameLayout alParent;
21
22     @Override
23     public void onCreate(Bundle savedInstanceState) {
24         super.onCreate(savedInstanceState);
25         /* Set the screen orientation to landscape, because
26          * the camera preview will be in landscape, and if we
27          * don't do this, then we will get a stretched image.*/
28         setRequestedOrientation(ActivityInfo.SCREEN_ORIENTATION_LANDSCAPE);
29
30         // requesting to turn the title OFF
31         requestWindowFeature(Window.FEATURE_NO_TITLE);
32
33         // making it full screen
34         getWindow().setFlags(WindowManager.LayoutParams.FLAG_FULLSCREEN,
35             WindowManager.LayoutParams.FLAG_FULLSCREEN);
36     }
37
38     public void Load(){
39         // Try to get the camera
40         Camera c = getCameraInstance();
41
42         List<Size> l = c.getParameters().getSupportedPreviewSizes();
43
44         // If the camera was received, create the app
45         if (c != null){
46             /* Create our layout in order to layer the
47              * draw view on top of the camera preview.
48              */
49             alParent = new FrameLayout(this);
50             alParent.setLayoutParams(new LayoutParams(
51                 LayoutParams.FILL_PARENT,
52                 LayoutParams.FILL_PARENT));
53
54             // Create a new camera view and add it to the layout
55             dv = new CameraOverlay(this);
56             cv = new CameraPreview(this,c, dv);
57             alParent.addView(cv);
58             alParent.addView(dv);
59
60             // Set the layout as the apps content view
61             setContentView(alParent);
62         }
63         // If the camera was not received, close the app
64         else {
65             Toast toast = Toast.makeText(getApplicationContext(),
```

```

66         "Unable to find camera. Closing.", Toast.LENGTH_SHORT);
67         toast.show();
68         finish();
69     }
70 }
71
72 /* This method is strait for the Android API */
73 /** A safe way to get an instance of the Camera object. */
74 public static Camera getCameraInstance(){
75     Camera c = null;
76
77     try {
78         c = Camera.open();// attempt to get a Camera instance
79     }
80     catch (Exception e){
81         // Camera is not available (in use or does not exist)
82         e.printStackTrace();
83     }
84     return c; // returns null if camera is unavailable
85 }
86
87 /* Override the onPause method so that we
88 * can release the camera when the app is closing.
89 */
90 @Override
91 protected void onPause() {
92     super.onPause();
93
94     if (cv != null){
95         cv.onPause();
96         cv = null;
97     }
98 }
99
100 /* We call Load in our Resume method, because
101 * the app will close if we call it in onCreate
102 */
103 @Override
104 protected void onResume(){
105     super.onResume();
106
107     Load();
108 }
109 }

```

APPENDIX C
CAMERADATABUS.V

```
1 // cameradatabus.v
2 module cameradatabus(
3     input IO_1_Y,
4     input IO_3_Y,
5     input IO_6_Y,
6     input IO_7_Y,
7     input IO_9_Y,
8     input IO_12_Y,
9     input IO_13_Y,
10    input IO_15_Y,
11    output [7:0] DATA_CAMERA
12    );
13
14    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};
15
16 endmodule
```


APPENDIX D
CAMERAOVERLAY.JAVA

```
1 package com.projectlevitate.quadcamdroid;
2
3 import android.content.Context;
4 import android.graphics.Bitmap;
5 import android.graphics.Canvas;
6 import android.graphics.Paint;
7 import android.util.DisplayMetrics;
8 import android.view.SurfaceHolder;
9 import android.view.SurfaceView;
10
11 public class CameraOverlay extends SurfaceView {
12
13     private static final int MOVE_UNKOWN = 0;
14     private static final int MOVE_FORWARD = 1;
15     private static final int MOVE_BACKWARD = 2;
16     private static final int MOVE_LEFT = 3;
17     private static final int MOVE_RIGHT = 4;
18     private static final int MOVE_NONE = 5;
19
20     private int mMovement = 0;
21
22     private Paint textPaint = new Paint();
23     private int[] mData;
24
25     private Bitmap mPicture = null;
26
27     private CameraPreview.Tag mTag = null;
28
29     public CameraOverlay(Context context) {
30         super(context);
31         textPaint.setARGB(255, 200, 0, 0);
32         textPaint.setTextSize(60);
33         setWillNotDraw(false);
34     }
35
36     @Override
37     protected void onDraw(Canvas canvas) {
38         switch(mMovement) {
39             case MOVE_UNKOWN:
40                 canvas.drawText("No Target", 50, 50, textPaint);
41                 break;
42             case MOVE_FORWARD:
43                 canvas.drawText("Forward", 50, 50, textPaint);
44                 break;
45             case MOVE_BACKWARD:
46                 canvas.drawText("Backword", 50, 50, textPaint);
47                 break;
48             case MOVE_LEFT:
49                 canvas.drawText("Left", 50, 50, textPaint);
50                 break;
51             case MOVE_RIGHT:
52                 canvas.drawText("Right", 50, 50, textPaint);
53                 break;
54             case MOVE_NONE:
55                 canvas.drawText("Center", 50, 50, textPaint);
56                 break;
57             default:
58                 canvas.drawText("Error", 50, 50, textPaint);
59                 break;
60         }
61     }
62     if(mPicture != null) {
63         canvas.drawBitmap(mPicture, 100, 100, textPaint);
64         canvas.drawBitmap(mPicture, 500, 100, textPaint);
65     }
```

```

66     if(mTag != null) {
67         canvas.drawRect(mTag.start.x + 500, mTag.start.y + 100, mTag.end.x + 500, mTag.end.y + 100,
68             textPaint);
69     }
70 }
71 public int getMovement() {
72     return mMovement;
73 }
74 }
75 public void setMovement(int movement) {
76     mMovement = movement;
77     invalidate();
78 }
79 }
80 public int[] getData() {
81     return mData;
82 }
83 }
84 public void setData(int[] data) {
85     mData = data;
86     mPicture = Bitmap.createBitmap(data, 320, 240, Bitmap.Config.ARGB_8888);
87     invalidate();
88 }
89 }
90 public CameraPreview.Tag getTag() {
91     return mTag;
92 }
93 }
94 public void setTag(CameraPreview.Tag tag) {
95     mTag = tag;
96 }
97 }

```

APPENDIX E
CAMERAPREVIEW.JAVA

```
1 package com.projectlevitate.quadcamdroid;
2
3 import java.io.IOException;
4
5 import android.content.Context;
6 import android.hardware.Camera;
7 import android.hardware.Camera.PreviewCallback;
8 import android.util.Log;
9 import android.view.SurfaceHolder;
10 import android.view.SurfaceView;
11
12 /** A basic Camera preview class */
13 public class CameraPreview extends SurfaceView implements SurfaceHolder.Callback, PreviewCallback
14 {
15     private SurfaceHolder mHolder;
16     private Camera mCamera;
17     private CameraOverlay mCameraOverlay;
18     private String TAG = "QuadCamDroid";
19
20     public CameraPreview(Context context) {
21         super(context);
22     }
23
24     @SuppressWarnings("deprecation")
25     public CameraPreview(Context context, Camera camera, CameraOverlay overlay) {
26         super(context);
27         mCamera = camera;
28         mCameraOverlay = overlay;
29
30         // Install a SurfaceHolder.Callback so we get notified when the
31         // underlying surface is created and destroyed.
32         mHolder = getHolder();
33         mHolder.addCallback(this);
34         // deprecated setting, but required on Android versions prior to 3.0
35         mHolder.setType(SurfaceHolder.SURFACE_TYPE_PUSH_BUFFERS);
36     }
37
38     public void surfaceCreated(SurfaceHolder holder) {
39         // The Surface has been created, now tell the camera where to draw the preview.
40         try {
41             Camera.Parameters camPams = mCamera.getParameters();
42             //camPams.setPreviewFormat(ImageFormat.RGB_565);
43             camPams.setPreviewSize(320, 240);
44             mCamera.setParameters(camPams);
45             mCamera.setPreviewDisplay(holder);
46             mCamera.setPreviewCallback(this);
47             mCamera.startPreview();
48         } catch (IOException e) {
49             Log.d(TAG, "Error setting camera preview: " + e.getMessage());
50         }
51     }
52
53     public void surfaceDestroyed(SurfaceHolder holder) {
54         // empty. Take care of releasing the Camera preview in your activity.
55     }
56
57     public void surfaceChanged(SurfaceHolder holder, int format, int w, int h) {
58         // If your preview can change or rotate, take care of those events here.
59         // Make sure to stop the preview before resizing or reformatting it.
60
61         if (mHolder.getSurface() == null){
62             // preview surface does not exist
63             return;
64         }
65     }
66 }
```

```

65     // stop preview before making changes
66     try {
67         mCamera.stopPreview();
68     } catch (Exception e){
69         // ignore: tried to stop a non-existent preview
70     }
71
72     // set preview size and make any resize, rotate or
73     // reformatting changes here
74
75     // start preview with new settings
76     try {
77         Camera.Parameters camPams = mCamera.getParameters();
78         //camPams.setPreviewFormat(ImageFormat.RGB_565);
79         camPams.setPreviewSize(320, 240);
80         mCamera.setParameters(camPams);
81         mCamera.setPreviewDisplay(holder);
82         mCamera.setPreviewCallback(this);
83         mCamera.startPreview();
84     } catch (Exception e){
85         Log.d(TAG, "Error starting camera preview: " + e.getMessage());
86     }
87 }
88
89 public void onPause() {
90     mCamera.release();
91     mCamera = null;
92 }
93
94 @Override
95 public void onPreviewFrame(byte[] data, Camera camera) {
96     int[] picture = convertYUV420_NV21toRGB8888(data, 320, 240);
97     byte[] picture_bw = new byte[320*240];
98     for(int i = 0; i < picture.length; i++) {
99         int sum = (picture[i]&0xff) + ((picture[i]&0xff00) >> 8) + ((picture[i]&0xff0000) >>
100             16);
101         if(sum < 255*3/2) {
102             picture_bw[i] = 0;
103         } else {
104             picture_bw[i] = 1;
105         }
106     }
107     byte[][] picture_compressed = new byte[240][40];
108     for(int i = 0, m = 0; i < 240; i++) {
109         for(int j = 0; j < 40; j++) {
110             for(int k = 0; k < 8; k++, m++) {
111                 byte pixel = (byte) (picture_bw[m] << k);
112                 picture_compressed[i][j] = (byte) (picture_compressed[i][j] | pixel);
113             }
114         }
115     }
116     int[] picture_draw = new int[320*240];
117     for(int i = 0; i < picture.length; i++) {
118         if(picture_bw[i] == 0) {
119             picture_draw[i] = 0xff000000;
120         } else {
121             picture_draw[i] = 0xffffffff;
122         }
123     }
124     int result = process_frame(picture_compressed);
125     if(result == MOVE_UNKOWN) {
126         if(count > 0) {
127             count--;
128         } else {
129             direction = MOVE_UNKOWN;
130         }
131     } else {
132         direction = result;

```

```

132     count = COUNT_LIM;
133 }
134 mCameraOverlay.setMovement(direction);
135 mCameraOverlay.setData(picture_draw);
136 mCameraOverlay.setTag(tag);
137 Log.d("QuadCamDroid", "Result:" + result);
138 Log.d("QuadCamDroid", "StoreResult:" + direction);
139 Log.d("QuadCamDroid", "Count:" + count);
140 }
141
142 // From: http://stackoverflow.com/questions/5272388/need-help-with-androids-nv21-format
143 /**
144  * Converts YUV420 NV21 to RGB8888
145  *
146  * @param data byte array on YUV420 NV21 format.
147  * @param width pixels width
148  * @param height pixels height
149  * @return a RGB8888 pixels int array. Where each int is a pixels ARGB.
150  */
151 public static int[] convertYUV420_NV21toRGB8888(byte [] data, int width, int height) {
152     int size = width*height;
153     int offset = size;
154     int[] pixels = new int[size];
155     int u, v, y1, y2, y3, y4;
156
157     // i percorre os Y and the final pixels
158     // k percorre os pixels U e V
159     for(int i=0, k=0; i < size; i+=2, k+=2) {
160         y1 = data[i ]&0xff;
161         y2 = data[i+1]&0xff;
162         y3 = data[width+i ]&0xff;
163         y4 = data[width+i+1]&0xff;
164
165         u = data[offset+k ]&0xff;
166         v = data[offset+k+1]&0xff;
167         u = u-128;
168         v = v-128;
169
170         pixels[i ] = convertYUVtoRGB(y1, u, v);
171         pixels[i+1] = convertYUVtoRGB(y2, u, v);
172         pixels[width+i ] = convertYUVtoRGB(y3, u, v);
173         pixels[width+i+1] = convertYUVtoRGB(y4, u, v);
174
175         if (i!=0 && (i+2)%width==0)
176             i+=width;
177     }
178
179     return pixels;
180 }
181
182 private static int convertYUVtoRGB(int y, int u, int v) {
183     int r,g,b;
184
185     r = y + (int)1.402f*v;
186     g = y - (int)(0.344f*u + 0.714f*v);
187     b = y + (int)1.772f*u;
188     r = r>255? 255 : r<0 ? 0 : r;
189     g = g>255? 255 : g<0 ? 0 : g;
190     b = b>255? 255 : b<0 ? 0 : b;
191     return 0xff000000 | (b<<16) | (g<<8) | r;
192 }
193
194 private static final int MOVE_UNKOWN = 0;
195 private static final int MOVE_FORWARD = 1;
196 private static final int MOVE_BACKWARD = 2;
197 private static final int MOVE_LEFT = 3;
198 private static final int MOVE_RIGHT = 4;
199 private static final int MOVE_NONE = 5;

```

```

200
201 private static final int BW_CUTOFF_X = 100;
202 private static final int BW_CUTOFF_Y = 100;
203 private static final int BW_LOW_CUTOFF_X = 100;
204 private static final int BW_LOW_CUTOFF_Y = 100;
205 private static final int BW_EPSILON = 8000;
206
207 private static final int COUNT_LIM = 50;
208
209 private int direction = 0;
210 private int count = 0;
211 private Tag tag;
212
213 public class Coord {
214     int x;
215     int y;
216     public Coord() {
217         x = 0;
218         y = 0;
219     }
220 }
221
222 public class Tag {
223     Coord start;
224     Coord end;
225     public Tag() {
226         start = new Coord();
227         end = new Coord();
228     }
229 }
230
231 int process_frame(byte[][] picture)
232 {
233     // Types
234     long[] histogram = new long[2];
235     int i, j, k, m;
236     Tag largest_tag = new Tag();
237     Tag curr_tag = new Tag();
238     long[] histogram_x = new long[320];
239     long[] histogram_y = new long[240];
240     long black = 0;
241     long white = 0;
242
243     // Initialize Data
244     for(i = 0; i < 2; i++) {
245         histogram[i] = 0;
246     }
247     for(i = 0; i < 320; i++) {
248         histogram_x[i] = 0;
249     }
250     for(i = 0; i < 240; i++) {
251         histogram_y[i] = 0;
252     }
253     largest_tag.start.x = 0;
254     largest_tag.start.y = 0;
255     largest_tag.end.x = 0;
256     largest_tag.end.y = 0;
257     curr_tag.start.x = 0;
258     curr_tag.start.y = 0;
259     curr_tag.end.x = 0;
260     curr_tag.end.y = 0;
261
262     // Create histogram
263     for(i = 0; i < 240; i++) {
264         for(j = 0, m = 0; j < 40; j++) {
265             for(k = 0; k < 8; k++, m++) {
266                 // Get the color
267                 byte color = (byte) ((picture[i][j] >> k) & 0x1);

```

```

268
269         // Add the value to the histogram.
270         histogram[color]++;
271         histogram_x[m] += color;
272         histogram_y[i] += color;
273     }
274 }
275 }
276
277 // Create start/stop of tag
278 if(histogram[0] < histogram[1] - 100) {
279     // Create start/stop of tag (largest white rectangle)
280     for(i = 0; i < 320; i++) {
281         for(j = 0; j < 240; j++) {
282
283             // Reset the current tag
284             curr_tag.start.x = i;
285             curr_tag.start.y = j;
286             curr_tag.end.x = i;
287             curr_tag.end.y = j;
288
289             // Find next possible curr_tag candidate.
290             // start
291             for(k = i; k < 320; k++) {
292                 if(histogram_x[k] > BW_CUTOFF_X) {
293                     curr_tag.start.x = k;
294                     break;
295                 }
296             }
297             for(k = j; k < 240; k++) {
298                 if(histogram_y[k] > BW_CUTOFF_Y) {
299                     curr_tag.start.y = k;
300                     break;
301                 }
302             }
303
304             // end
305             curr_tag.end.x = curr_tag.start.x;
306             curr_tag.end.y = curr_tag.start.y;
307             for(k = curr_tag.start.x; k < 320; k++) {
308                 curr_tag.end.x = k;
309                 if(histogram_x[k] <= BW_CUTOFF_X) {
310                     break;
311                 }
312             }
313             for(k = curr_tag.start.y; k < 240; k++) {
314                 curr_tag.end.y = k;
315                 if(histogram_y[k] <= BW_CUTOFF_Y) {
316                     break;
317                 }
318             }
319
320             // If curr_tag is larger than largest_tag, set largest_tag
321             // to currtag.
322             if((curr_tag.end.x - curr_tag.start.x)
323                 * (curr_tag.end.y - curr_tag.start.y)
324                 > (largest_tag.end.x - largest_tag.start.x)
325                 * (largest_tag.end.y - largest_tag.start.y)) {
326                 largest_tag.start.x = curr_tag.start.x;
327                 largest_tag.start.y = curr_tag.start.y;
328                 largest_tag.end.x = curr_tag.end.x;
329                 largest_tag.end.y = curr_tag.end.y;
330             }
331         }
332     }
333 } else {
334     // Create start/stop of tag (largest white rectangle)
335     for(i = 0; i < 320; i++) {

```

```

336     for(j = 0; j < 240; j++) {
337
338         // Reset the current tag
339         curr_tag.start.x = i;
340         curr_tag.start.y = j;
341         curr_tag.end.x = i;
342         curr_tag.end.y = j;
343
344         // Find next possible curr_tag candidate.
345         // start
346         for(k = i; k < 320; k++) {
347             if(histogram_x[k] < BW_LOW_CUTOFF_X) {
348                 curr_tag.start.x = k;
349                 break;
350             }
351         }
352         for(k = j; k < 240; k++) {
353             if(histogram_y[k] < BW_LOW_CUTOFF_Y) {
354                 curr_tag.start.y = k;
355                 break;
356             }
357         }
358
359         // end
360         curr_tag.end.x = curr_tag.start.x;
361         curr_tag.end.y = curr_tag.start.y;
362         for(k = curr_tag.start.x; k < 320; k++) {
363             curr_tag.end.x = k;
364             if(histogram_x[k] >= BW_LOW_CUTOFF_X) {
365                 break;
366             }
367         }
368         for(k = curr_tag.start.y; k < 240; k++) {
369             curr_tag.end.y = k;
370             if(histogram_y[k] >= BW_LOW_CUTOFF_Y) {
371                 break;
372             }
373         }
374
375         // If curr_tag is larger than largest_tag, set largest_tag
376         // to currtag.
377         if((curr_tag.end.x - curr_tag.start.x)
378             * (curr_tag.end.y - curr_tag.start.y)
379             > (largest_tag.end.x - largest_tag.start.x)
380             * (largest_tag.end.y - largest_tag.start.y)) {
381             largest_tag.start.x = curr_tag.start.x;
382             largest_tag.start.y = curr_tag.start.y;
383             largest_tag.end.x = curr_tag.end.x;
384             largest_tag.end.y = curr_tag.end.y;
385         }
386     }
387 }
388 }
389
390 // Run a histogram to see if it is about the same object
391 for(i = largest_tag.start.y; i < largest_tag.end.y; i++) {
392     for(j = largest_tag.start.x; j < largest_tag.end.x; j++) {
393         byte color = (byte) ((picture[i][j]>>3] >> (j&0x7)) & 0x1);
394         if(color != 0) {
395             white++;
396         } else {
397             black++;
398         }
399     }
400 }
401 tag = largest_tag;
402
403

```



```

404 // Print out results
405 Log.d("QuadCamDroid", "Total Colors:");
406 for(i = 0; i < 2; i++) {
407     Log.d("QuadCamDroid", "" + i + "\t" + histogram[i]);
408 }
409 Log.d("QuadCamDroid", "X-Values:");
410 for(i = 0; i < 320; i++) {
411     Log.d("QuadCamDroid", "" + i + "\t" + histogram_x[i]);
412 }
413 Log.d("QuadCamDroid", "Y-Values:");
414 for(i = 0; i < 240; i++) {
415     Log.d("QuadCamDroid", "" + i + "\t" + histogram_y[i]);
416 }
417 Log.d("QuadCamDroid", "Tag Values:");
418 Log.d("QuadCamDroid", "X-Start:" + largest_tag.start.x);
419 Log.d("QuadCamDroid", "X-End:" + largest_tag.end.x);
420 Log.d("QuadCamDroid", "Y-Start:" + largest_tag.start.y);
421 Log.d("QuadCamDroid", "Y-End:" + largest_tag.end.y);
422 Log.d("QuadCamDroid", "Color Values:");
423 Log.d("QuadCamDroid", "White:" + white + "\tBlack:" + black);
424 Log.d("QuadCamDroid", "Black-White:\t" + Math.abs(black-white));
425
426 // Close program if possible target isn't target
427 if(Math.abs(black-white) > BW_EPSILON) {
428     return MOVE_UNKOWN;
429 };
430
431 // Determin what where next target is on screen.
432 if(largest_tag.end.x < 120) {
433     return MOVE_LEFT;
434 } else if(largest_tag.start.x > 200) {
435     return MOVE_RIGHT;
436 } else if(largest_tag.end.y < 120) {
437     return MOVE_FORWARD;
438 } else if(largest_tag.start.y > 120) {
439     return MOVE_BACKWARD;
440 } else {
441     return MOVE_NONE;
442 }
443
444 // Shouldn't get here
445 // return MOVE_UNKOWN;
446 }
447 }

```

APPENDIX F
CAMERAWRAPPER.V

```

1 // camerawrapper.v
2 module camerawrapper(
3     DATA_CAMERA,
4     DK,
5     //HD,
6     //VD,
7     REQ_I2C,
8     ADDR_I2C,
9     RW_I2C,
10    REG_I2C,
11    DATA_WRITE_I2C,
12    DATA_SIZE_I2C,
13    DATA_READ_I2C,
14    DATA_READY_I2C,
15    DONE_I2C,
16    RDY_I2C,
17    PCLK,
18    PENABLE,
19    PSEL,
20    PRESETN,
21    PWRITE,
22    PREADY,
23    PSLVERR,
24    PADDR,
25    PWDATA,
26    PRDATA,
27    CAMCLK,
28    FABINT
29 );
30
31 // Camera Communications
32 input                                DK; //, HD, VD;
33 input [7:0]                          DATA_CAMERA;
34 output reg                            CAMCLK;
35
36 // I2C Interface
37 input                                DONE_I2C, RDY_I2C, DATA_READY_I2C;
38 input [7:0]                          DATA_READ_I2C;
39 output reg [7:0]                      DATA_SIZE_I2C;
40 output reg                            RW_I2C, REQ_I2C;
41 output reg [6:0]                      ADDR_I2C;
42 output reg [7:0]                      REG_I2C;
43 output reg [7:0]                      DATA_WRITE_I2C;
44
45 // APB Wrapper
46 input                                PCLK, PENABLE, PSEL, PRESETN, PWRITE;
47 input [31:0]                          PWDATA, PADDR;
48 output                                PREADY, PSLVERR;
49 output reg [31:0]                     PRDATA;
50 output reg                            FABINT;
51
52 wire                                  rd_enable;
53 wire                                  wr_enable;
54
55 assign wr_enable                      = (PENABLE && PWRITE && PSEL);
56 assign rd_enable                      = (!PWRITE && PSEL);
57 assign PREADY                        = 1'b1;
58 assign PSLVERR                       = 1'b0;
59 // assign ADDR_I2C                    = 8'b01111000;
60
61 // Picture memory
62 reg [31:0]                            picture_0;
63 reg [31:0]                            picture_1;
64
65 // State

```

```

66     reg [3:0]                                state;
67     reg                                       picture_state;
68
69     reg [1:0]                                startup_state;
70
71     reg [1:0]                                clk_divider;
72
73     always@(posedge PCLK) begin
74         if(~PRESETN) begin
75             clk_divider                       <= 0;
76             CAMCLK                            <= 0;
77         end else begin
78             if(clk_divider == 2) begin
79                 clk_divider                   <= 0;
80                 CAMCLK                        <= ~CAMCLK;
81             end else begin
82                 clk_divider                   <= clk_divider + 1;
83             end
84         end
85     end
86
87     always@(posedge PCLK) begin
88         if(~PRESETN) begin
89             ADDR_I2C                           <= 7'd0;
90             REG_I2C                            <= 8'd0;
91             DATA_WRITE_I2C                   <= 8'd0;
92             DATA_SIZE_I2C                    <= 8'd0;
93             RW_I2C                             <= 1'b0;
94             REQ_I2C                            <= 1'b0;
95             startup_state                      <= 0;
96         end else begin
97             case(startup_state)
98                 0: begin
99                     if(RDY_I2C) begin
100                        ADDR_I2C                <= 7'b0111100;
101                        REG_I2C                 <= 8'h3;
102                        DATA_WRITE_I2C       <= 8'b0000110;
103                        DATA_SIZE_I2C        <= 8'd1;
104                        RW_I2C                 <= 1'b0;
105                        REQ_I2C                <= 1'b0;
106                        startup_state         <= 1;
107                    end else begin
108                        ADDR_I2C                <= 7'd0;
109                        REG_I2C                 <= 8'd0;
110                        DATA_WRITE_I2C       <= 8'd0;
111                        DATA_SIZE_I2C        <= 8'd0;
112                        RW_I2C                 <= 1'b0;
113                        REQ_I2C                <= 1'b0;
114                        startup_state         <= 0;
115                    end
116                end
117                 1: begin
118                     if(DONE_I2C) begin
119                        ADDR_I2C                <= 7'd0;
120                        REG_I2C                 <= 8'd0;
121                        DATA_WRITE_I2C       <= 8'd0;
122                        DATA_SIZE_I2C        <= 8'd0;
123                        RW_I2C                 <= 1'b0;
124                        REQ_I2C                <= 1'b0;
125                        startup_state         <= 2;
126                    end
127                 end
128                 2: begin
129                        ADDR_I2C                <= 7'd0;
130                        REG_I2C                 <= 8'd0;
131                        DATA_WRITE_I2C       <= 8'd0;
132                        DATA_SIZE_I2C        <= 8'd0;
133                        RW_I2C                 <= 1'b0;

```

```

134         REQ_I2C                <= 1'b0;
135         startup_state          <= 2;
136     end
137 endcase // case (startup_state)
138 if(rd_enable) begin
139     if(state == 0) begin
140         PRDATA                <= picture_0;
141     end else begin
142         PRDATA                <= picture_1;
143     end
144 end
145 end
146 end
147
148 always@(negedge DK) begin
149     if(~PRESETN) begin
150         picture_0              <= 0;
151         picture_1              <= 0;
152         state                  <= 0;
153         picture_state          <= 0;
154     end else begin
155         if(picture_state == 0) begin
156             case(state)
157                 0: begin
158                     picture_0[7:0]    <= DATA_CAMERA;
159                     state              <= 1;
160                     FABINT            <= 0;
161                 end
162                 1: begin
163                     picture_0[15:8]   <= DATA_CAMERA;
164                     state              <= 2;
165                     FABINT            <= 0;
166                 end
167                 2: begin
168                     picture_0[23:16]  <= DATA_CAMERA;
169                     state              <= 3;
170                     FABINT            <= 0;
171                 end
172                 3: begin
173                     picture_0[31:24]  <= DATA_CAMERA;
174                     state              <= 0;
175                     picture_state     <= 1;
176                     FABINT            <= 1;
177                 end
178             endcase // case (state)
179         end else begin
180             case(state)
181                 0: begin
182                     picture_1[7:0]    <= DATA_CAMERA;
183                     state              <= 1;
184                     FABINT            <= 0;
185                 end
186                 1: begin
187                     picture_1[15:8]   <= DATA_CAMERA;
188                     state              <= 2;
189                     FABINT            <= 0;
190                 end
191                 2: begin
192                     picture_1[23:16]  <= DATA_CAMERA;
193                     state              <= 3;
194                     FABINT            <= 0;
195                 end
196                 3: begin
197                     picture_1[31:24]  <= DATA_CAMERA;
198                     state              <= 0;
199                     picture_state     <= 0;
200                     FABINT            <= 1;
201                 end

```

```
202         endcase // case (state)
203     end
204 end
205 end
206 endmodule
```

APPENDIX G
COMPLEMENTARYFILTER.V

```

1 // complementary.v
2
3 module complementary(
4     input RESETN,
5     input CLK,
6     input XYZ_RDY,
7     input [15:0] ACCEL_DATA_X,
8     input [15:0] ACCEL_DATA_Y,
9     input [15:0] ACCEL_DATA_Z,
10    input [15:0] GYRO_DATA_X,
11    input [15:0] GYRO_DATA_Y,
12    input [15:0] GYRO_DATA_Z,
13    input [15:0] MAGNETO_DATA_X,
14    input [15:0] MAGNETO_DATA_Y,
15    input [15:0] MAGNETO_DATA_Z,
16    input signed [7:0] GYRO_X_OFFSET,
17    input signed [7:0] GYRO_Y_OFFSET,
18    input signed [7:0] GYRO_Z_OFFSET,
19    input signed [7:0] ACCEL_X_OFFSET,
20    input signed [7:0] ACCEL_Y_OFFSET,
21    input signed [7:0] ACCEL_Z_OFFSET,
22
23    output reg RD_RQ,
24    output reg XYZ_READ,
25
26    output reg signed [15:0] PITCH, // Y
27    output reg signed [15:0] ROLL, // X
28    output reg signed [15:0] YAW,
29    output reg signed [31:0] sumPitch,
30    output reg signed [31:0] sumRoll,
31    output reg signed [15:0] deltaPitch,
32    output reg signed [15:0] deltaRoll
33 );
34
35
36 parameter PASS_RATIO_SHIFT = 4; // input >> PASS_RATIO_SHIFT = input * .03125
37 parameter PASS_RATIO_MASK = {{PASS_RATIO_SHIFT{1'b0}}, {(16-PASS_RATIO_SHIFT){1'b1}}};
38 parameter ACCEL_RATIO_SHIFT_1 = 4; // ACCEL >> ACCEL_RATIO_SHIFT_1 = ACCEL * 16
39 parameter ACCEL_RATIO_SHIFT_2 = 3; // ACCEL >> ACCEL_RATIO_SHIFT_2 = ACCEL * 8
40 parameter ACCEL_RATIO_MASK_1 = {{ACCEL_RATIO_SHIFT_1{1'b0}}, {(16-ACCEL_RATIO_SHIFT_1){1'b1}}};
41 parameter ACCEL_RATIO_MASK_2 = {{ACCEL_RATIO_SHIFT_2{1'b0}}, {(16-ACCEL_RATIO_SHIFT_2){1'b1}}};
42 parameter GYRO_RATIO_SHIFT = 4;
43
44 parameter MAGNETOMETER_X_OFFSET = 0;
45 parameter MAGNETOMETER_Y_OFFSET = 0;
46 parameter MAGNETOMETER_Z_OFFSET = 0;
47
48 wire signed [15:0] GYRO_X;
49 wire signed [15:0] GYRO_Y;
50 wire [15:0] GYRO_Z;
51 wire signed [15:0] ACCEL_X;
52 wire signed [15:0] ACCEL_Y;
53 wire [15:0] ACCEL_Z;
54 wire [15:0] MAGNETO_X;
55 wire [15:0] MAGNETO_Y;
56 wire [15:0] MAGNETO_Z;
57
58 wire signed [15:0] GYRO_TEMP_X;
59 wire signed [15:0] GYRO_TEMP_Y;
60 wire [15:0] GYRO_TEMP_Z;
61 wire signed [15:0] ACCEL_TEMP_X;
62 wire signed [15:0] ACCEL_TEMP_Y;
63 wire [15:0] ACCEL_TEMP_Z;

```

```

64 wire [15:0] MAGNETO_TEMP_X;
65 wire [15:0] MAGNETO_TEMP_Y;
66 wire [15:0] MAGNETO_TEMP_Z;
67
68 wire signed [15:0] PITCH_GYRO_X;
69 wire signed [15:0] ROLL_GYRO_Y;
70
71 assign GYRO_TEMP_X = (GYRO_DATA_X);// + $signed({ {8{GYRO_X_OFFSET[7]}}, GYRO_X_OFFSET[7:0] })
);//{{{8{GYRO_X_OFFSET[7]}},{GYRO_X_OFFSET}}};
72 assign GYRO_X = ($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.
73
74 assign GYRO_TEMP_Y = (GYRO_DATA_Y);// + $signed({ {8{GYRO_Y_OFFSET[7]}}, GYRO_Y_OFFSET[7:0] })
);
75 assign GYRO_Y = ($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.
76
77 assign GYRO_Z = GYRO_DATA_Z + GYRO_Z_OFFSET;
78
79 assign ACCEL_TEMP_X = (ACCEL_DATA_X);// + $signed({ {8{ACCEL_X_OFFSET[7]}}, ACCEL_X_OFFSET
[7:0] }));
80 assign ACCEL_X = ($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.
81
82 assign ACCEL_TEMP_Y = (ACCEL_DATA_Y);// + $signed({ {8{ACCEL_Y_OFFSET[7]}}, ACCEL_Y_OFFSET
[7:0] }));
83 assign ACCEL_Y = ($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.
84
85 assign ACCEL_Z = ACCEL_DATA_Z + ACCEL_Z_OFFSET;
86 assign MAGNETO_X = MAGNETO_DATA_X + MAGNETOMETER_X_OFFSET;
87 assign MAGNETO_Y = MAGNETO_DATA_Y + MAGNETOMETER_Y_OFFSET;
88 assign MAGNETO_Z = MAGNETO_DATA_Z + MAGNETOMETER_Z_OFFSET;
89
90 assign PITCH_GYRO_X = PITCH + GYRO_X;
91 assign ROLL_GYRO_Y = ROLL + GYRO_Y;
92
93 parameter SUSPENDED = 0;
94 parameter UPDATE = 1;
95 parameter PITCH_0 = 2;
96 parameter ROLL_0 = 3;
97 /*parameter ROLL_1 =
98 parameter ROLL_2 =
99 parameter ROLL_3 =
100 parameter YAW_0 =
101 parameter YAW_1 =
102 parameter YAW_2 =
103 parameter YAW_3 = */
104
105 reg [3:0] state;
106
107 always@(posedge CLK)
108 begin
109 if(~RESETN)
110 begin
111 state <= SUSPENDED;
112 RD_RQ <= 0;
113 XYZ_READ <= 0;
114 PITCH <= 0;
115 YAW <= 0;
116 ROLL <= 0;
117 sumPitch <= 0;
118 sumRoll <= 0;
119 deltaPitch <= 0;

```

```

120         deltaRoll <= 0;
121     end
122     else
123         begin
124             case(state)
125             SUSPENDED:
126                 begin
127                     if (XYZ_RDY)
128                         begin
129                             state <= PITCH_0;
130                             RD_RQ <= 0;
131                             XYZ_READ <= 1;
132                         end
133                     else
134                         begin
135                             RD_RQ <= 1;
136                             XYZ_READ <= 0;
137                         end
138                     end
139                 end
140             PITCH_0:
141                 begin
142                     PITCH <= ((PITCH_GYRO_X) - ($signed(PITCH_GYRO_X) >>> PASS_RATIO_SHIFT)) + (
143                         $signed(ACCEL_X) >>> PASS_RATIO_SHIFT);
144                     sumPitch <= sumPitch + ((PITCH_GYRO_X) - ($signed(PITCH_GYRO_X) >>>
145                         PASS_RATIO_SHIFT)) + ($signed(ACCEL_X) >>> PASS_RATIO_SHIFT);
146                     deltaPitch <= ((PITCH_GYRO_X) - ($signed(PITCH_GYRO_X) >>> PASS_RATIO_SHIFT))
147                         + ($signed(ACCEL_X) >>> PASS_RATIO_SHIFT) - PITCH;
148                     state <= ROLL_0;
149                 end
150             ROLL_0:
151                 begin
152                     ROLL <= ((ROLL_GYRO_Y) - ($signed(ROLL_GYRO_Y) >>> PASS_RATIO_SHIFT)) + (
153                         $signed(ACCEL_Y) >>> PASS_RATIO_SHIFT);
154                     sumRoll <= sumRoll + ((ROLL_GYRO_Y) - ($signed(ROLL_GYRO_Y) >>>
155                         PASS_RATIO_SHIFT)) + ($signed(ACCEL_Y) >>> PASS_RATIO_SHIFT);
156                     deltaRoll <= ((ROLL_GYRO_Y) - ($signed(ROLL_GYRO_Y) >>> PASS_RATIO_SHIFT)) +
157                         ($signed(ACCEL_Y) >>> PASS_RATIO_SHIFT) - ROLL;
158                     state <= SUSPENDED;
159                 end
160             endcase
161         end // else: !if(~RESETN)
162     end // always@ (posedge CLK)
163 endmodule // complimentary

```


APPENDIX H
COMPLEMENTARYFILTERWRAPPER.V

```

1 //
  //////////////////////////////////////////////////////////////////////////////////////////////////////////////////
2 // Company: <Name>
3 //
4 // File: complementaryfilterwrapper.v
5 // File history:
6 //     <Revision number>: <Date>: <Comments>
7 //     <Revision number>: <Date>: <Comments>
8 //     <Revision number>: <Date>: <Comments>
9 //
10 // Description:
11 //
12 // <Description here>
13 //
14 // Targeted device: <Family::SmartFusion> <Die::A2F200M3F> <Package::484 FBGA>
15 // Author: <Name>
16 //
17 //
  //////////////////////////////////////////////////////////////////////////////////////////////////////////////////

18
19 module complementaryfilterwrapper(
20     input XYZ_RDY,
21     input [15:0] ACCEL_DATA_X,
22     input [15:0] ACCEL_DATA_Y,
23     input [15:0] ACCEL_DATA_Z,
24     input [15:0] GYRO_DATA_X,
25     input [15:0] GYRO_DATA_Y,
26     input [15:0] GYRO_DATA_Z,
27     input [15:0] MAGNETO_DATA_X,
28     input [15:0] MAGNETO_DATA_Y,
29     input [15:0] MAGNETO_DATA_Z,
30
31     output RD_RQ,
32     output XYZ_READ,
33
34     output [15:0] PITCH, // Y
35     output [15:0] ROLL, // X
36     output [15:0] YAW,
37     output signed [31:0] sumPitch ,
38     output signed [31:0] sumRoll ,
39     output signed [15:0] deltaPitch ,
40     output signed [15:0] deltaRoll ,
41
42     input PCLK,
43     input PENABLE,
44     input PSEL ,
45     input PRESETN,
46     input PWRITE,
47     output PREADY,
48     output PSLVERR,
49     input [7:0] PADDR,
50     input [31:0] PWDATA,
51     output reg [31:0] PRDATA
52 );
53
54 assign BUS_WRITE_EN = (PENABLE && PWRITE && PSEL);
55 assign BUS_READ_EN = (!PWRITE && PSEL); //Data is ready during first cycle to make it available
    on the bus when PENABLE is asserted
56
57 assign PREADY = 1'b1;
58 assign PSLVERR = 1'b0;
59
60 reg [7:0] GYRO_X_OFFSET;

```

```

61 reg [7:0] GYRO_Y_OFFSET;
62 reg [7:0] GYRO_Z_OFFSET;
63 reg [7:0] ACCEL_X_OFFSET;
64 reg [7:0] ACCEL_Y_OFFSET;
65 reg [7:0] ACCEL_Z_OFFSET;
66 reg data_rdy;
67 reg prev_rq; // If prev_rq == 0 and RD_RQ == 1, new data is ready.
68
69 always@(posedge PCLK) begin
70 if(~PRESETN) begin
71 // nothing to do on reset
72 end
73 else begin
74 if(BUS_READ_EN)
75 begin
76 case(PADDR[5:2])
77 4'b0000:
78 begin
79 PRDATA[31:0] <= {16'h0, PITCH};
80 end
81
82 4'b0001:
83 begin
84 PRDATA <= {16'h0, ROLL};
85 end
86
87 4'b0010:
88 begin
89 PRDATA <= {16'h0, YAW};
90 end
91
92 4'b0011:
93 begin
94 PRDATA <= {31'h0, data_rdy};
95 data_rdy <= 1'b0;
96 end
97 endcase
98 end
99 else if(BUS_WRITE_EN)
100 begin
101 case(PADDR[5:2])
102 4'b0100:
103 begin
104 GYRO_X_OFFSET <= PWDATA[7:0];
105 end
106
107 4'b0101:
108 begin
109 GYRO_Y_OFFSET <= PWDATA[7:0];
110 end
111
112 4'b0110:
113 begin
114 GYRO_Z_OFFSET <= PWDATA[7:0];
115 end
116
117 4'b0111:
118 begin
119 ACCEL_X_OFFSET <= PWDATA[7:0];
120 end
121
122 4'b1000:
123 begin
124 ACCEL_Y_OFFSET <= PWDATA[7:0];
125 end
126
127 4'b1001:
128 begin

```

```

129         ACCEL_Z_OFFSET <= PWDATA[7:0];
130     end
131 endcase
132 end
133
134     // Update data_rdy
135     if (!BUS_READ_EN || PADDR[5:2] != 4'b0011) // If we are not reading the status
136         register.
137     begin
138         if (prev_rq == 0 && RD_RQ == 1) // If prev_rq == 0 and RD_RQ == 1, new data is
139             ready.
140             data_rdy <= 1'b1;
141     end
142     prev_rq <= RD_RQ;
143 end
144
145     complementary_complementary_0(
146         .RESETN(PRESETN),
147         .CLK(PCLK),
148         .XYZ_RDY(XYZ_RDY),
149         .ACCEL_DATA_X(ACCEL_DATA_X),
150         .ACCEL_DATA_Y(ACCEL_DATA_Y),
151         .ACCEL_DATA_Z(ACCEL_DATA_Z),
152         .GYRO_DATA_X(GYRO_DATA_X),
153         .GYRO_DATA_Y(GYRO_DATA_Y),
154         .GYRO_DATA_Z(GYRO_DATA_Z),
155         .MAGNETO_DATA_X(MAGNETO_DATA_X),
156         .MAGNETO_DATA_Y(MAGNETO_DATA_Y),
157         .MAGNETO_DATA_Z(MAGNETO_DATA_Z),
158         .GYRO_X_OFFSET(GYRO_X_OFFSET),
159         .GYRO_Y_OFFSET(GYRO_Y_OFFSET),
160         .GYRO_Z_OFFSET(GYRO_Z_OFFSET),
161         .ACCEL_X_OFFSET(ACCEL_X_OFFSET),
162         .ACCEL_Y_OFFSET(ACCEL_Y_OFFSET),
163         .ACCEL_Z_OFFSET(ACCEL_Z_OFFSET),
164
165         .RD_RQ(RD_RQ),
166         .XYZ_READ(XYZ_READ),
167
168         .PITCH(PITCH), // Y
169         .ROLL(ROLL), // X
170         .YAW(YAW),
171         .sumPitch(sumPitch),
172         .sumRoll(sumRoll),
173         .deltaPitch(deltaPitch),
174         .deltaRoll(deltaRoll)
175     );
176
177
178 endmodule

```

APPENDIX I
CUSTOMTYPES.H

```
1 #ifndef CUSTOMTYPES_H_ // Only define once
2 #define CUSTOMTYPES_H_ // Only define once
3
4 #ifndef NULL
5 #define NULL ((void*)0)
6 #endif // #ifndef NULL
7 #ifndef true
8 #define true 1
9 #endif // #ifndef true
10 #ifndef false
11 #define false 0
12 #endif // #ifndef false
13
14 typedef signed char bool;
15
16 #define MIN(A,B) (A < B ? A : B)
17
18 #endif /* CUSTOMTYPES_H_ */
```

APPENDIX J
FILTER.H/C

```

1 #ifndef FILTER_H_ // Only define once
2 #define FILTER_H_ // Only define once
3 #include "customtypes.h"
4 #include <stdint.h>
5 #include "levitatefpga_hw_platform.h"
6 #include <stdio.h>
7 #include "drivers/mss_uart/mss_uart.h"
8 #include <stdio.h>
9 #include "imudriver.h"
10
11 typedef struct
12 {
13     int32_t pitch;
14     int32_t roll;
15     int32_t yaw;
16     uint32_t data_ready;
17     int32_t gyroOffsetX;
18     int32_t gyroOffsetY;
19     int32_t gyroOffsetZ;
20     int32_t accelOffsetX;
21     int32_t accelOffsetY;
22     int32_t accelOffsetZ;
23 }filter_t;
24
25 int32_t pitch_offset;
26 int32_t roll_offset;
27
28 void initializeFilter();
29
30 int32_t getPitch();
31 int32_t getRoll();
32
33 int8_t data_ready();
34
35 void filter_set_gyro_x_offset(int32_t offset);
36 void filter_set_gyro_y_offset(int32_t offset);
37 void filter_set_accel_x_offset(int32_t offset);
38 void filter_set_accel_y_offset(int32_t offset);
39
40 void filter_set_pitch_offset(int32_t offset);
41 void filter_set_roll_offset(int32_t offset);
42
43 void filter_modify_pitch_offset(int32_t offset);
44 void filter_modify_roll_offset(int32_t offset);
45
46 #endif /* TESTSS_H_ */

1 #include "filter.h"
2
3 static volatile filter_t* filter = (filter_t*) COMPLEMENTARYFILTERWRAPPER_0;
4
5 void initializeFilter()
6 {
7     //printf("gyrox: %i gyroy: %i accelx: %i accely: %i\r\n", -get_gyro_x(), -get_gyro_y(), -
8         get_accel_x(), -get_accel_y());
9     filter_set_gyro_x_offset(0); //-get_gyro_x();
10    filter_set_gyro_y_offset(0); //-get_gyro_y();
11    filter_set_accel_x_offset(0); //-get_accel_x();
12    filter_set_accel_y_offset(0); //-get_accel_y();
13    /*((int32_t*) (COMPLEMENTARYFILTERWRAPPER_0 + 6)) = gyroOffsetZ;
14 }
15 int8_t data_ready()
16 {
17     return filter->data_ready & 1;

```

```

18 }
19
20 int32_t getPitch()
21 {
22     return (int32_t)((int16_t)(filter->pitch));
23 }
24
25 int32_t getRoll()
26 {
27     return (int32_t)((int16_t)(filter->roll));
28 }
29
30 //Gyro and accel offsets: 24x accel = gyro/16. That is, gyro = 384 * accel.
31 void filter_set_gyro_x_offset(int32_t offset)
32 {
33     if (offset > 127)
34         offset = 127;
35     if (offset < -128)
36         offset = -128;
37     filter->gyroOffsetX = offset;
38 }
39
40 void filter_set_gyro_y_offset(int32_t offset)
41 {
42     if (offset > 127)
43         offset = 127;
44     if (offset < -128)
45         offset = -128;
46     filter->gyroOffsetY = offset;
47 }
48
49 void filter_set_accel_x_offset(int32_t offset)
50 {
51     if (offset > 255)
52         offset = 256;
53     if (offset < -256)
54         offset = -256;
55     filter->accelOffsetX = offset;
56 }
57
58 void filter_set_accel_y_offset(int32_t offset)
59 {
60     if (offset > 255)
61         offset = 256;
62     if (offset < -256)
63         offset = -256;
64     filter->accelOffsetY = offset;
65 }
66
67 /*
68 * IMPORTANT: Offset will be multiplied by 24, so for example if you change offset by 10, roll
69 *           and pitch
70 *           change by 240.
71 */
72 void filter_set_pitch_offset(int32_t offset)
73 {
74     pitch_offset = offset;
75 }
76 void filter_set_roll_offset(int32_t offset)
77 {
78     roll_offset = offset;
79 }
80
81 void filter_modify_pitch_offset(int32_t offset)
82 {
83     pitch_offset += offset;
84     filter_set_pitch_offset(pitch_offset);

```

```
85 }
86
87 void filter_modify_roll_offset(int32_t offset)
88 {
89     roll_offset += offset;
90     filter_set_roll_offset(roll_offset);
91 }
```

APPENDIX K
I2CINTERFACE.V

```
1 // States are progressed through in order (except for the burst states which are on a counter)
2 // Write States
3 `define RDY_S          7'd0
4 `define START_W_1     7'd1
5 `define START_W_2     7'd2
6 `define ADDRESS_W_1   7'd3
7 `define ADDRESS_W_2   7'd4
8 `define ADDRESS_W_3   7'd5
9 `define ADDR_ACK_W_1  7'd6
10 `define ADDR_ACK_W_2  7'd7
11 `define ADDR_ACK_W_3  7'd8
12 `define REG_W_1       7'd9
13 `define REG_W_2       7'd10
14 `define REG_W_3       7'd11
15 `define REG_ACK_W_1   7'd12
16 `define REG_ACK_W_2   7'd13
17 `define REG_ACK_W_3   7'd14
18 `define DATA_W_1     7'd15
19 `define DATA_W_2     7'd16
20 `define DATA_W_3     7'd17
21 `define DATA_ACK_W_1 7'd18
22 `define DATA_ACK_W_2 7'd19
23 `define DATA_ACK_W_3 7'd20
24 `define STOP_W_0      7'd21
25 `define STOP_W_1      7'd22
26 `define STOP_W_2      7'd23
27 // Read States
28 `define START_R_1     7'd24
29 `define START_R_2     7'd25
30 `define ADDRESS_R_1   7'd26
31 `define ADDRESS_R_2   7'd27
32 `define ADDRESS_R_3   7'd28
33 `define ADDR_ACK_R_1  7'd29
34 `define ADDR_ACK_R_2  7'd30
35 `define ADDR_ACK_R_3  7'd31
36 `define REG_R_1       7'd32
37 `define REG_R_2       7'd33
38 `define REG_R_3       7'd34
39 `define REG_ACK_R_1   7'd35
40 `define REG_ACK_R_2   7'd36
41 `define REG_ACK_R_3   7'd37
42 `define SR_R_1        7'd38
43 `define SR_R_2        7'd39
44 `define SR_R_3        7'd40
45 `define SR_R_4        7'd41
46 `define ADDR_2_R_1    7'd42
47 `define ADDR_2_R_2    7'd43
48 `define ADDR_2_R_3    7'd44
49 `define ADDR_2_ACK_R_1 7'd45
50 `define ADDR_2_ACK_R_2 7'd46
51 `define ADDR_2_ACK_R_3 7'd47
52 `define DATA_BURST_R_1 7'd48
53 `define DATA_BURST_R_2 7'd49
54 `define DATA_BURST_R_3 7'd50
55 `define BURST_ACK_R_1  7'd51
56 `define BURST_ACK_R_2  7'd52
57 `define BURST_ACK_R_3  7'd53
58 `define NACK_R_1      7'd54
59 `define NACK_R_2      7'd55
60 `define NACK_R_3      7'd56
61 `define STOP_R_0      7'd57
62 `define STOP_R_1      7'd58
63 `define STOP_R_2      7'd59
64 `define RESET_BUS_1   7'd60
65 `define RESET_BUS_2   7'd61
```



```

66 `define RESET_BUS_3 7'd62
67 `define RESET_BUS_4 7'd63
68 `define RESET_BUS_5 7'd64
69 `define RESET_BUS_6 7'd65
70
71 // Libero hates me and always makes my parameters all 1's
72 `define HALFSCL 32'd250
73 `define QUARTERSCL 32'd125
74 `define SHORTSCL 32'd25
75 `define STATUS_PERIOD 32'd500
76 `define RESET_PERIOD 32'd50
77
78 // i2cinterface.v
79 module i2cinterface(
80     REQ,
81     ADDR,
82     RW,
83     REGI,
84     DS,
85     DW,
86     DR,
87     DRDY,
88     DONE,
89     RDY,
90     FCLK,
91     RESETN,
92     SCL_IN,
93     SCL_OUT,
94     SCL_EN,
95     SDA_IN,
96     SDA_OUT,
97     SDA_EN,
98     I2C_STATUS_LED,
99     I2C_RESET_LED
100 );
101
102     input [6:0] ADDR; // Hardware address
103     input [7:0] DS; // Data size in bytes (if reading)
104     input [7:0] REGI; // Device register
105     input [7:0] DW; // Data to write to register
106     input REQ, FCLK, RESETN, RW;
107
108     input SCL_IN;
109     output reg SCL_OUT;
110     output reg SCL_EN;
111     input SDA_IN;
112     output reg SDA_OUT;
113     output reg SDA_EN;
114     output reg I2C_STATUS_LED;
115     output reg I2C_RESET_LED;
116     reg [9:0] statuscounter;
117     reg [9:0] resetcounter;
118
119     output reg [7:0] DR; // Data read from register
120     output reg RDY, DONE, DRDY;
121
122     reg [6:0] state;
123
124     reg [2:0] burstcounter;
125     reg [7:0] data_size;
126     reg [9:0] clkcounter;
127
128     reg BCLK;
129 //
130 // always@(posedge FCLK) begin
131 //     if (~RESETN) begin
132 //         BCLK <= 1'b0;
133 //     end

```

```

134 //         else begin
135 //             BCLK <= ~BCLK;
136 //         end
137 //     end
138
139 // I2C Control State Machine
140 /*
141 Write:
142     pull SDA low
143     Wait 1/2 of SCL cycle
144     start SCL (by pulling low)
145     Wait a short amount of time (~500 ns)
146     Send Address
147         Put address bit on SDA
148         Wait 1/2 SCL
149         Push SCL high
150         Wait 1/2 SCL
151         Pull SCL low
152     Repeat 8 times
153     Let SDA high
154     Wait 1/2 SCL
155     Push SCL high
156     Look for ACK
157     Wait 1/2 SCL
158     Pull SCL low
159     Send Register
160         Put register bit on SDA
161         Wait 1/2 SCL
162         Push SCL high
163         Wait 1/2 SCL
164         Pull SCL low
165     Repeat 8 times
166     Let SDA high
167     Wait 1/2 SCL
168     Push SCL high
169     Look for ACK
170     Wait 1/2 SCL
171     Pull SCL low
172     Send Data
173         Put data bit on SDA
174         Wait 1/2 SCL
175         Push SCL high
176         Wait 1/2 SCL
177         Pull SCL low
178     Repeat 8 times
179     Let SDA high
180     Wait 1/2 SCL
181     Push SCL high
182     Look for ACK
183     Wait 1/2 SCL
184     Pull SCL low
185     Wait 1/2 SCL
186     Let SCL high (On posedge after ACK)
187     Wait 1/2 of SCL
188     Let SDA high
189
190 Write-Read:
191     pull SDA low
192     Wait 1/2 of SCL cycle
193     start SCL (by pulling low)
194     Wait a short amount of time (~500 ns)
195     Send Address
196         Put address bit on SDA
197         Wait 1/2 SCL
198         Push SCL high
199         Wait 1/2 SCL
200         Pull SCL low
201     Repeat 8 times

```

```

202     Let SDA high
203     Wait 1/2 SCL
204     Push SCL high
205     Look for ACK
206     Wait 1/2 SCL
207     Pull SCL low
208     Send Register
209         Put register bit on SDA
210         Wait 1/2 SCL
211         Push SCL high
212         Wait 1/2 SCL
213         Pull SCL low
214     Repeat 8 times
215     Let SDA high
216     Wait 1/2 SCL
217     Push SCL high
218     Look for ACK
219     Wait 1/2 SCL
220     Pull SCL low
221     Wait 1/4 of SCL cycle
222     Let SDA go high
223     Wait 1/4 of SCL cycle
224     Let SCL go high
225     Wait 1/4 of SCL cycle
226     Pull SDA low
227     Wait 1/2 of SCL cycle
228     Pull SCL low
229     Wait a short amount of time
230     Send Address
231         Put address bit on SDA
232         Wait 1/2 SCL
233         Push SCL high
234         Wait 1/2 SCL
235         Pull SCL low
236     Repeat 8 times
237     Let SDA high
238     Wait 1/2 SCL
239     Push SCL high
240     Look for ACK
241     Wait 1/2 SCL
242     Pull SCL low
243     Read Data
244         Grab data from slave
245         Wait 1/2 SCL
246         Grab data bit from SDA
247         Push SCL high
248         Wait 1/2 SCL
249         Pull SCL low
250         Repeat 8 times
251         Pull SDA low (ACK)
252         Wait 1/2 SCL
253         Push SCL high
254         Wait 1/2 SCL
255         Let SDA High
256         Pull SCL low
257         (repeat ACK only until datasize is 1)
258     Repeat until datasize is 0
259     Wait 1/2 SCL
260     Push SCL high
261     Wait 1/2 SCL
262     Pull SCL low
263     Wait 1/2 SCL
264     let SCL high
265     wait 1/2 SCL
266     let SDA high
267 */
268 always@(posedge FCLK/* or negedge RESETN*/) begin
269     if (~RESETN) begin

```

```

270     state          <= 'RESET_BUS_1;
271     burstcounter   <= 3'b000;
272     clkcounter     <= 9'd0;
273     RDY            <= 1'b0;
274     DONE          <= 1'b0;
275     DRDY          <= 1'b0;
276     SDA_EN        <= 1'b0;
277     SDA_OUT       <= 1'b0;
278     SCL_EN        <= 1'b0;
279     SCL_OUT       <= 1'b0;
280     data_size     <= 8'b0000_0000;
281     DR            <= 8'b0000_0000;
282     I2C_STATUS_LED <= 1'b1;
283     I2C_RESET_LED <= 1'b1;
284     statuscounter <= 10'd0;
285     resetcounter  <= 10'd0;
286 end
287 else begin
288     case(state)
289     'RESET_BUS_1: begin // Reset the bus
290         if (clkcounter == 'HALFSCL) begin
291             state          <= 'RESET_BUS_2;
292             clkcounter     <= 9'd0;
293         end
294         else begin // Wait 1/2 SCL
295             state          <= 'RESET_BUS_1;
296             clkcounter     <= clkcounter + 9'd1; // Count
297         end
298         SDA_EN            <= 1'b0; // Make sure we let SDA float (if it's under our
299             control at any point)
300         SCL_EN            <= 1'b1; // Make sure we have control of the clock
301         SCL_OUT           <= 1'b1; // Push SCL high
302         I2C_STATUS_LED    <= 1'b1;
303         if (resetcounter == 'RESET_PERIOD) begin
304             I2C_RESET_LED <= ~I2C_RESET_LED;
305             resetcounter  <= 10'd0;
306         end
307     end
308     'RESET_BUS_2: begin
309         if (clkcounter == 'HALFSCL) begin
310             state          <= 'RESET_BUS_3;
311             clkcounter     <= 9'd0;
312         end
313         else begin // Wait 1/2 SCL
314             state          <= 'RESET_BUS_2;
315             clkcounter     <= clkcounter + 9'd1; // Count
316         end
317         SCL_OUT           <= 1'b0; // Pull clock down
318     end
319     'RESET_BUS_3: begin
320         if (clkcounter == 'HALFSCL) begin
321             if (SDA_IN) begin
322                 state          <= 'RESET_BUS_4;
323             end
324             else begin
325                 state          <= 'RESET_BUS_2;
326             end
327             clkcounter     <= 9'd0;
328         end
329         else begin // Wait 1/2 SCL
330             state          <= 'RESET_BUS_3;
331             clkcounter     <= clkcounter + 9'd1; // Count
332         end
333         SCL_OUT           <= 1'b1; // push clock up
334     end
335     'RESET_BUS_4: begin // preface stop with a low clock (since stop needs SCL to
336         be low to make a low-high transition)
337         if (clkcounter == 'HALFSCL) begin

```

```
336         state          <= 'RESET_BUS_5;
337         clkcounter     <= 9'd0;
338     end
339     else begin // Wait 1/2 SCL
340         state          <= 'RESET_BUS_4;
341         clkcounter     <= clkcounter + 9'd1; // Count
342     end
343     SCL_OUT          <= 1'b0; // Pull clock down
344 end
345 'RESET_BUS_5: begin // send a stop to finish the reset
346     if (clkcounter == 'HALFSCL) begin
347         state          <= 'RESET_BUS_6;
348         clkcounter     <= 9'd0;
349     end
350     else begin // Wait 1/2 SCL
351         state          <= 'RESET_BUS_5;
352         clkcounter     <= clkcounter + 9'd1; // Count
353     end
354     SCL_EN          <= 1'b0; // Let SCL high (first part of a stop)
355 end
356 'RESET_BUS_6: begin
357     if (clkcounter == 'HALFSCL) begin
358         if (SDA_IN) begin
359             state          <= 'RDY_S;
360         end
361         else begin
362             state          <= 'RESET_BUS_1; // Reset repeatedly until the bus
363             is floating
364             clkcounter     <= 9'd0;
365         end
366         else begin // Wait 1/2 SCL
367             state          <= 'RESET_BUS_6;
368             clkcounter     <= clkcounter + 9'd1; // Count
369         end
370         RDY          <= 1'b1;
371         SDA_EN       <= 1'b0; // Let SDA high (finish transaction)
372         resetcounter <= resetcounter + 10'd1;
373     end
374     'RDY_S: begin
375         if (REQ == 1'b1) begin
376             if (RW == 1'b0) begin
377                 state          <= 'START_W_1;
378             end
379             else begin
380                 state          <= 'START_R_1;
381             end
382             burstcounter <= 3'b000;
383             clkcounter   <= 9'd0;
384             RDY          <= 1'b1;
385             DONE         <= 1'b0;
386             DRDY         <= 1'b0;
387             SDA_EN       <= 1'b0;
388             SDA_OUT      <= 1'b0;
389             SCL_EN       <= 1'b0;
390             SCL_OUT      <= 1'b0;
391             data_size    <= 8'b0000_0000;
392             DR           <= 8'b0000_0000;
393         end
394         if (statuscounter == 'STATUS_PERIOD) begin
395             I2C_STATUS_LED <= ~I2C_STATUS_LED;
396             statuscounter <= 10'd0;
397         end
398         I2C_RESET_LED <= 1'b1;
399     end
400 //
    //////////////////////////////////////
```

```

401 // #####//
402 // #####//
403 //##### Write Transaction
404 //#####
405 // #####//
406 // #####//

407
408 'START_W_1: begin
409     if (clkcounter == 'HALFSCL) begin
410         state         <= 'START_W_2;
411         clkcounter     <= 9'd0;
412     end
413     else begin // Wait 1/2 SCL
414         state         <= 'START_W_1;
415         clkcounter     <= clkcounter + 9'd1; // Count
416     end
417     RDY             <= 1'b0; // No longer ready for new requests
418     DONE           <= 1'b0; // Not done with request yet
419     SDA_EN         <= 1'b1; // Pull SDA low
420     SDA_OUT        <= 1'b0;
421 end
422 'START_W_2: begin
423     if (clkcounter == 'SHORTSCL) begin // this stretches the first cycle of
424         the address
425         state         <= 'ADDRESS_W_1;
426         clkcounter     <= 9'd0;
427     end
428     else begin // Wait a short amount of time
429         state         <= 'START_W_2;
430         clkcounter     <= clkcounter + 9'd1; // Count
431     end
432     SCL_EN         <= 1'b1; // Start sending the clock out
433     SCL_OUT        <= 1'b0; // Pull clock down
434 end
435 'ADDRESS_W_1: begin
436     if (clkcounter == 'HALFSCL) begin
437         state         <= 'ADDRESS_W_2;
438         clkcounter     <= 9'd0;
439     end
440     else begin // Wait 1/2 SCL
441         state         <= 'ADDRESS_W_1;
442         clkcounter     <= clkcounter + 9'd1; // Count
443     end
444     SDA_OUT        <= 1'b1 & ({ADDR,RW} >> (3'd7 - burstcounter)); // Put
445         address bit on SDA
446 end
447 'ADDRESS_W_2: begin
448     if (clkcounter == 'HALFSCL) begin
449         state         <= 'ADDRESS_W_3;
450         clkcounter     <= 9'd0;
451     end
452     else begin // Wait 1/2 SCL
453         state         <= 'ADDRESS_W_2;
454         clkcounter     <= clkcounter + 9'd1; // Count
455     end
456     SCL_OUT        <= 1'b1; // Push clock up
457 end

```

```

456 'ADDRESS_W_3: begin
457     if (burstcounter == 3'b111) begin
458         state           <= 'ADDR_ACK_W_1;
459         burstcounter    <= 3'b000;    // reset burst counter
460         SDA_EN          <= 1'b0;    // Let SDA go high so we can check for ACK in
                                     the next cycle
461     end
462     else begin
463         state           <= 'ADDRESS_W_1;
464         // Count out all the bits
465         burstcounter    <= burstcounter + 1;
466     end
467     SCL_OUT            <= 1'b0;    // pull clock down
468 end
469 'ADDR_ACK_W_1: begin
470     if (clkcounter == 'HALFSCL) begin
471         state           <= 'ADDR_ACK_W_2;
472         clkcounter      <= 9'd0;
473     end
474     else begin    // Wait 1/2 SCL
475         state           <= 'ADDR_ACK_W_1;
476         clkcounter      <= clkcounter + 9'd1;    // Count
477     end
478     SDA_EN            <= 1'b0;    // Let SDA high
479 end
480 'ADDR_ACK_W_2: begin
481     if (clkcounter == 'HALFSCL) begin
482         if (SDA_IN) begin
483             state       <= 'STOP_W_1;    // Slave failed to ACK
484             clkcounter   <= 9'd0;
485         end
486         else begin
487             state       <= 'ADDR_ACK_W_3;
488             clkcounter   <= 9'd0;
489         end
490     end
491     else begin    // Wait 1/2 SCL
492         state           <= 'ADDR_ACK_W_2;
493         clkcounter      <= clkcounter + 9'd1;    // Count
494     end
495     SCL_OUT            <= 1'b1;    // Push SCL high
496 end
497 'ADDR_ACK_W_3: begin
498     state              <= 'REG_W_1;
499     SCL_OUT            <= 1'b0;    // Pull SCL low
500     SDA_EN            <= 1'b1;    // Take back SDA
501 end
502 'REG_W_1: begin
503     if (clkcounter == 'HALFSCL) begin
504         state           <= 'REG_W_2;
505         clkcounter      <= 9'd0;
506     end
507     else begin    // Wait 1/2 SCL
508         state           <= 'REG_W_1;
509         clkcounter      <= clkcounter + 9'd1;    // Count
510     end
511     SDA_OUT            <= 1'b1 & (REGI >> (3'd7 - burstcounter));    // Put register
                                     bit on SDA
512 end
513 'REG_W_2: begin
514     if (clkcounter == 'HALFSCL) begin
515         state           <= 'REG_W_3;
516         clkcounter      <= 9'd0;
517     end
518     else begin    // Wait 1/2 SCL
519         state           <= 'REG_W_2;
520         clkcounter      <= clkcounter + 9'd1;    // Count
521     end

```

```

522     SCL_OUT          <= 1'b1;           // Push clock up
523 end
524 'REG_W_3: begin
525     if (burstcounter == 3'b111) begin
526         state          <= 'REG_ACK_W_1;
527         burstcounter   <= 3'b000;       // reset burst counter
528         SDA_EN         <= 1'b0;       // Let SDA go high so we can check for ACK in
                                     the next cycle
529     end
530     else begin
531         state          <= 'REG_W_1;
532         // Count out all the bits
533         burstcounter   <= burstcounter + 1;
534     end
535     SCL_OUT          <= 1'b0;       // pull clock down
536 end
537 'REG_ACK_W_1: begin
538     if (clkcounter == 'HALFSCL) begin
539         state          <= 'REG_ACK_W_2;
540         clkcounter     <= 9'd0;
541     end
542     else begin // Wait 1/2 SCL
543         state          <= 'REG_ACK_W_1;
544         clkcounter     <= clkcounter + 9'd1; // Count
545     end
546     SDA_EN           <= 1'b0;       // Let SDA high
547 end
548 'REG_ACK_W_2: begin
549     if (clkcounter == 'HALFSCL) begin
550         if (SDA_IN) begin
551             state      <= 'STOP_W_1; // Slave failed to ACK
552             clkcounter <= 9'd0;
553         end
554         else begin
555             state      <= 'REG_ACK_W_3;
556             clkcounter <= 9'd0;
557         end
558     end
559     else begin // Wait 1/2 SCL
560         state          <= 'REG_ACK_W_2;
561         clkcounter     <= clkcounter + 9'd1; // Count
562     end
563     SCL_OUT          <= 1'b1;       // Push SCL high
564 end
565 'REG_ACK_W_3: begin
566     state          <= 'DATA_W_1;
567     SCL_OUT        <= 1'b0;       // Pull SCL low
568     SDA_EN         <= 1'b1;       // Take control of SDA
569 end
570 'DATA_W_1: begin
571     if (clkcounter == 'HALFSCL) begin
572         state          <= 'DATA_W_2;
573         clkcounter     <= 9'd0;
574     end
575     else begin // Wait 1/2 SCL
576         state          <= 'DATA_W_1;
577         clkcounter     <= clkcounter + 9'd1; // Count
578     end
579     SDA_OUT        <= 1'b1 & (DW >> (3'd7 - burstcounter)); // Put register
                                     bit on SDA
580 end
581 'DATA_W_2: begin
582     if (clkcounter == 'HALFSCL) begin
583         state          <= 'DATA_W_3;
584         clkcounter     <= 9'd0;
585     end
586     else begin // Wait 1/2 SCL
587         state          <= 'DATA_W_2;

```



```

588         clkcounter      <= clkcounter + 9'd1;    // Count
589     end
590     SCL_OUT             <= 1'b1;                // Push clock up
591 end
592 'DATA_W_3: begin
593     if (burstcounter == 3'b111) begin
594         state             <= 'DATA_ACK_W_1;
595         burstcounter      <= 3'b000;          // reset burst counter
596         SDA_EN           <= 1'b0;          // Let SDA go high so we can check for ACK in
                                     the next cycle
597     end
598     else begin
599         state             <= 'DATA_W_1;
600         // Count out all the bits
601         burstcounter      <= burstcounter + 1;
602     end
603     SCL_OUT             <= 1'b0;          // pull clock down
604 end
605 'DATA_ACK_W_1: begin
606     if (clkcounter == 'HALFSCL) begin
607         state             <= 'DATA_ACK_W_2;
608         clkcounter        <= 9'd0;
609     end
610     else begin // Wait 1/2 SCL
611         state             <= 'DATA_ACK_W_1;
612         clkcounter        <= clkcounter + 9'd1;    // Count
613     end
614     SDA_EN             <= 1'b0;          // Let SDA high
615 end
616 'DATA_ACK_W_2: begin
617     if (clkcounter == 'HALFSCL) begin
618         if (SDA_IN) begin
619             state         <= 'STOP_W_0;          // Slave failed to ACK
620             clkcounter    <= 9'd0;
621         end
622         else begin
623             state         <= 'DATA_ACK_W_3;
624             clkcounter    <= 9'd0;
625         end
626     end
627     else begin // Wait 1/2 SCL
628         state             <= 'DATA_ACK_W_2;
629         clkcounter        <= clkcounter + 9'd1;    // Count
630     end
631     SCL_OUT            <= 1'b1;          // Push SCL high
632 end
633 'DATA_ACK_W_3: begin
634     if (clkcounter == 'HALFSCL) begin
635         state             <= 'STOP_W_1;
636         clkcounter        <= 9'd0;
637         DONE             <= 1'b1;          // Done. Let the requester know
638     end
639     else begin // Wait 1/2 SCL
640         state             <= 'DATA_ACK_W_3;
641         clkcounter        <= clkcounter + 9'd1;    // Count
642     end
643     SCL_OUT            <= 1'b0;          // Pull SCL low
644     SDA_EN             <= 1'b1;          // Take control of SDA
645     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)
646 end
647 'STOP_W_0: begin // This state is reached when the slave fails to ACK
648     if (clkcounter == 'HALFSCL) begin
649         state             <= 'STOP_W_1;
650         clkcounter        <= 9'd0;
651     end
652     else begin // Wait 1/2 SCL
653         state             <= 'STOP_W_0;

```

```

654         clkcounter      <= clkcounter + 9'd1;    // Count
655     end
656     SCL_OUT             <= 1'b0;    // Pull SCL low
657     SDA_EN              <= 1'b1;    // Take control of SDA
658 end
659 'STOP_W_1: begin
660     if (clkcounter == 'HALFSCL) begin
661         state           <= 'STOP_W_2;
662         clkcounter      <= 9'd0;
663     end
664     else begin // Wait 1/2 SCL
665         state           <= 'STOP_W_1;
666         clkcounter      <= clkcounter + 9'd1;    // Count
667     end
668     RDY                 <= 1'b1;    // Signal ready now so the fcfsmux has a chance
669     to stop presenting the previous request before we respond to it again
670     DONE                <= 1'b0;    // Done is a pulse.
671     SCL_EN              <= 1'b0;    // Let SCL high (first part of a stop)
672 end
673 'STOP_W_2: begin
674     if (clkcounter == 'HALFSCL) begin
675         statuscounter   <= statuscounter + 10'd1;
676         state           <= 'RDY_S;
677         clkcounter      <= 9'd0;
678     end
679     else begin // Wait 1/2 SCL
680         state           <= 'STOP_W_2;
681         clkcounter      <= clkcounter + 9'd1;    // Count
682     end
683     SDA_EN              <= 1'b0;    // Let SDA high (finish transaction)
684 end
685 //
686 //
687 //
688 //##### Read Transaction
689 //#####
690 //
691 //#####
692 //#####
693
694 'START_R_1: begin
695     if (clkcounter == 'HALFSCL) begin
696         state           <= 'START_R_2;
697         clkcounter      <= 9'd0;
698     end
699     else begin // Wait 1/2 SCL
700         state           <= 'START_R_1;
701         clkcounter      <= clkcounter + 9'd1;    // Count
702     end
703     RDY                 <= 1'b0;    // Not ready anymore
704     DONE                <= 1'b0;    // Not done yet
705     SDA_EN              <= 1'b1;    // Pull SDA low
706     SDA_OUT             <= 1'b0;
707     data_size           <= DS;    // Save the number of bytes we are sending
708 end

```

```

708 'START_R_2: begin
709     if (clkcounter == 'SHORTSCL) begin // this stretches the first cycle of
710         the address
711             state         <= 'ADDRESS_R_1;
712             clkcounter     <= 9'd0;
713         end
714     else begin // Wait a short amount of time
715         state         <= 'START_R_2;
716         clkcounter     <= clkcounter + 9'd1; // Count
717     end
718     SCL_EN           <= 1'b1; // Start sending the clock out
719     SCL_OUT          <= 1'b0; // Pull clock down
720 end
721 'ADDRESS_R_1: begin
722     if (clkcounter == 'HALFSCL) begin
723         state         <= 'ADDRESS_R_2;
724         clkcounter     <= 9'd0;
725     end
726     else begin // Wait 1/2 SCL
727         state         <= 'ADDRESS_R_1;
728         clkcounter     <= clkcounter + 9'd1; // Count
729     end
730     SDA_OUT          <= 1'b1 & ({ADDR, ~RW} >> (3'd7 - burstcounter)); // Put
731         address bit on SDA
732 end
733 'ADDRESS_R_2: begin
734     if (clkcounter == 'HALFSCL) begin
735         state         <= 'ADDRESS_R_3;
736         clkcounter     <= 9'd0;
737     end
738     else begin // Wait 1/2 SCL
739         state         <= 'ADDRESS_R_2;
740         clkcounter     <= clkcounter + 9'd1; // Count
741     end
742     SCL_OUT          <= 1'b1; // Push clock up
743 end
744 'ADDRESS_R_3: begin
745     if (burstcounter == 3'b111) begin
746         state         <= 'ADDR_ACK_R_1;
747         burstcounter   <= 3'b000; // reset burst counter
748         SDA_EN         <= 1'b0; // Let SDA go high so we can check for ACK in
749             the next cycle
750     end
751     else begin
752         state         <= 'ADDRESS_R_1;
753         // Count out all the bits
754         burstcounter   <= burstcounter + 1;
755     end
756     SCL_OUT          <= 1'b0; // pull clock down
757 end
758 'ADDR_ACK_R_1: begin
759     if (clkcounter == 'HALFSCL) begin
760         state         <= 'ADDR_ACK_R_2;
761         clkcounter     <= 9'd0;
762     end
763     else begin // Wait 1/2 SCL
764         state         <= 'ADDR_ACK_R_1;
765         clkcounter     <= clkcounter + 9'd1; // Count
766     end
767     SDA_EN           <= 1'b0; // Let SDA high
768 end
769 'ADDR_ACK_R_2: begin
770     if (clkcounter == 'HALFSCL) begin
771         if (SDA_IN) begin
772             state         <= 'STOP_R_0; // Slave failed to ACK
773             clkcounter     <= 9'd0;
774         end
775     else begin

```

```

773             state          <= 'ADDR_ACK_R_3;
774             clkcounter     <= 9'd0;
775         end
776     end
777     else begin // Wait 1/2 SCL
778         state          <= 'ADDR_ACK_R_2;
779         clkcounter     <= clkcounter + 9'd1; // Count
780     end
781     SCL_OUT          <= 1'b1; // Push SCL high
782 end
783 'ADDR_ACK_R_3: begin
784     state          <= 'REG_R_1;
785     SCL_OUT        <= 1'b0; // Pull SCL low
786     SDA_EN         <= 1'b1; // Take back SDA
787 end
788 'REG_R_1: begin
789     if (clkcounter == 'HALFSCL) begin
790         state          <= 'REG_R_2;
791         clkcounter     <= 9'd0;
792     end
793     else begin // Wait 1/2 SCL
794         state          <= 'REG_R_1;
795         clkcounter     <= clkcounter + 9'd1; // Count
796     end
797     SDA_OUT        <= 1'b1 & (REGI >> (3'd7 - burstcounter)); // Put
// register bit on SDA
798 end
799 'REG_R_2: begin
800     if (clkcounter == 'HALFSCL) begin
801         state          <= 'REG_R_3;
802         clkcounter     <= 9'd0;
803     end
804     else begin // Wait 1/2 SCL
805         state          <= 'REG_R_2;
806         clkcounter     <= clkcounter + 9'd1; // Count
807     end
808     SCL_OUT        <= 1'b1; // Push clock up
809 end
810 'REG_R_3: begin
811     if (burstcounter == 3'b111) begin
812         state          <= 'REG_ACK_R_1;
813         burstcounter  <= 3'b000; // reset burst counter
814         SDA_EN         <= 1'b0; // Let SDA go high so we can check for ACK
// in the next cycle
815     end
816     else begin
817         state          <= 'REG_R_1;
818         // Count out all the bits
819         burstcounter  <= burstcounter + 1;
820     end
821     SCL_OUT        <= 1'b0; // pull clock down
822 end
823 'REG_ACK_R_1: begin
824     if (clkcounter == 'HALFSCL) begin
825         state          <= 'REG_ACK_R_2;
826         clkcounter     <= 9'd0;
827     end
828     else begin // Wait 1/2 SCL
829         state          <= 'REG_ACK_R_1;
830         clkcounter     <= clkcounter + 9'd1; // Count
831     end
832     SDA_EN         <= 1'b0; // Let SDA high
833 end
834 'REG_ACK_R_2: begin
835     if (clkcounter == 'HALFSCL) begin
836         if (SDA_IN) begin
837             state          <= 'STOP_R_0; // Slave failed to ACK
838             clkcounter     <= 9'd0;

```

```

839         end
840         else begin
841             state          <= 'REG_ACK_R_3;
842             clkcounter     <= 9'd0;
843         end
844     end
845     else begin // Wait 1/2 SCL
846         state          <= 'REG_ACK_R_2;
847         clkcounter     <= clkcounter + 9'd1; // Count
848     end
849     SCL_OUT          <= 1'b1; // Push SCL high
850 end
851 'REG_ACK_R_3: begin
852     if (clkcounter == 'QUARTERSCL) begin
853         state          <= 'SR_R_1;
854         clkcounter     <= 9'd0;
855     end
856     else begin // Wait 1/4 SCL
857         state          <= 'REG_ACK_R_3;
858         clkcounter     <= clkcounter + 9'd1; // Count
859     end
860     SCL_OUT          <= 1'b0; // Pull SCL low
861     SDA_EN           <= 1'b1; // Take control of SDA
862     SDA_OUT          <= 1'b0; // Also make sure to actually set it to 0 (
// looks like sending register data otherwise)
863 end
864 'SR_R_1: begin
865     if (clkcounter == 'QUARTERSCL) begin
866         state          <= 'SR_R_2;
867         clkcounter     <= 9'd0;
868     end
869     else begin // Wait 1/4 SCL
870         state          <= 'SR_R_1;
871         clkcounter     <= clkcounter + 9'd1; // Count
872     end
873     SDA_EN           <= 1'b0; // Let SDA pull high
874 end
875 'SR_R_2: begin
876     if (clkcounter == 'QUARTERSCL) begin
877         state          <= 'SR_R_3;
878         clkcounter     <= 9'd0;
879     end
880     else begin // Wait 1/4 SCL
881         state          <= 'SR_R_2;
882         clkcounter     <= clkcounter + 9'd1; // Count
883     end
884     SCL_EN           <= 1'b0; // Let SCL pull high
885 end
886 'SR_R_3: begin
887     if (clkcounter == 'QUARTERSCL) begin
888         state          <= 'SR_R_4;
889         clkcounter     <= 9'd0;
890     end
891     else begin // Wait 1/4 SCL
892         state          <= 'SR_R_3;
893         clkcounter     <= clkcounter + 9'd1; // Count
894     end
895     SDA_EN           <= 1'b1; // Pull SDA Down
896 end
897 'SR_R_4: begin
898     if (clkcounter == 'SHORTSCL) begin
899         state          <= 'ADDR_2_R_1;
900         clkcounter     <= 9'd0;
901     end
902     else begin // Wait a short ammount of time
903         state          <= 'SR_R_4;
904         clkcounter     <= clkcounter + 9'd1; // Count
905     end

```

```

906     SCL_EN          <= 1'b1;    // Pull SCL down
907 end
908 'ADDR_2_R_1: begin
909     if (clkcounter == 'HALFSCL) begin
910         state        <= 'ADDR_2_R_2;
911         clkcounter    <= 9'd0;
912     end
913     else begin      // Wait 1/2 SCL
914         state        <= 'ADDR_2_R_1;
915         clkcounter    <= clkcounter + 9'd1;    // Count
916     end
917     SDA_OUT          <= 1'b1 & ({ADDR,RW} >> (3'd7 - burstcounter));    // Put
           address bit on SDA
918 end
919 'ADDR_2_R_2: begin
920     if (clkcounter == 'HALFSCL) begin
921         state        <= 'ADDR_2_R_3;
922         clkcounter    <= 9'd0;
923     end
924     else begin      // Wait 1/2 SCL
925         state        <= 'ADDR_2_R_2;
926         clkcounter    <= clkcounter + 9'd1;    // Count
927     end
928     SCL_OUT          <= 1'b1;    // Push clock up
929 end
930 'ADDR_2_R_3: begin
931     if (burstcounter == 3'b111) begin
932         state        <= 'ADDR_2_ACK_R_1;
933         burstcounter <= 3'b000;    // reset burst counter
934         SDA_EN        <= 1'b0;    // Let SDA go high so we can check for ACK
           in the next cycle
935     end
936     else begin
937         state        <= 'ADDR_2_R_1;
938         // Count out all the bits
939         burstcounter <= burstcounter + 1;
940     end
941     SCL_OUT          <= 1'b0;    // pull clock down
942 end
943 'ADDR_2_ACK_R_1: begin
944     if (clkcounter == 'HALFSCL) begin
945         state        <= 'ADDR_2_ACK_R_2;
946         clkcounter    <= 9'd0;
947     end
948     else begin      // Wait 1/2 SCL
949         state        <= 'ADDR_2_ACK_R_1;
950         clkcounter    <= clkcounter + 9'd1;    // Count
951     end
952     SDA_EN          <= 1'b0;    // Let SDA high
953 end
954 'ADDR_2_ACK_R_2: begin
955     if (clkcounter == 'HALFSCL) begin
956         if (SDA_IN) begin
957             state        <= 'STOP_R_0;    // Slave failed to ACK
958             clkcounter    <= 9'd0;
959         end
960         else begin
961             state        <= 'ADDR_2_ACK_R_3;
962             clkcounter    <= 9'd0;
963         end
964     end
965     else begin      // Wait 1/2 SCL
966         state        <= 'ADDR_2_ACK_R_2;
967         clkcounter    <= clkcounter + 9'd1;    // Count
968     end
969     SCL_OUT          <= 1'b1;    // Push SCL high
970 end
971 'ADDR_2_ACK_R_3: begin

```

```

972         state          <= 'DATA_BURST_R_1;
973         SCL_OUT        <= 1'b0;    // Pull SCL low
974     end
975     'DATA_BURST_R_1: begin
976         if (clkcounter == 'HALFSCL) begin
977             state      <= 'DATA_BURST_R_2;
978             clkcounter <= 9'd0;
979         end
980         else begin    // Wait 1/2 SCL
981             state      <= 'DATA_BURST_R_1;
982             clkcounter <= clkcounter + 9'd1;    // Count
983         end
984     end
985     'DATA_BURST_R_2: begin
986         if (clkcounter == 'HALFSCL) begin
987             state      <= 'DATA_BURST_R_3;
988             clkcounter <= 9'd0;
989         end
990         else begin    // Wait 1/2 SCL
991             state      <= 'DATA_BURST_R_2;
992             clkcounter <= clkcounter + 9'd1;    // Count
993         end
994         if (clkcounter == 9'd0) begin    // Grab data bit from SDA
995             DR         <= {DR[6:0], SDA_IN};
996         end
997         SCL_OUT      <= 1'b1;    // Push SCL high
998     end
999     'DATA_BURST_R_3: begin
1000         if (burstcounter == 3'b111) begin
1001             if (data_size == 8'd1) begin
1002                 state <= 'NACK_R_1;
1003             end
1004             else begin
1005                 state <= 'BURST_ACK_R_1;
1006             end
1007             data_size <= data_size - 8'd1;    // Decrement bytes left to see
1008             burstcounter <= 9'd0;
1009         end
1010         else begin    // Wait 1/2 SCL
1011             state <= 'DATA_BURST_R_1;
1012             burstcounter <= burstcounter + 3'd1;    // Count bits
1013         end
1014         SCL_OUT      <= 1'b0;    // Pull SCL low
1015     end
1016     'BURST_ACK_R_1: begin
1017         if (clkcounter == 'HALFSCL) begin
1018             state      <= 'BURST_ACK_R_2;
1019             clkcounter <= 9'd0;
1020             DRDY       <= 1'b1;    // Data is valid
1021         end
1022         else begin    // Wait 1/2 SCL
1023             state      <= 'BURST_ACK_R_1;
1024             clkcounter <= clkcounter + 9'd1;    // Count
1025         end
1026         SDA_EN        <= 1'b1;    // pull SDA low to ACK
1027         SDA_OUT       <= 1'b0;
1028     end
1029     'BURST_ACK_R_2: begin
1030         if (clkcounter == 'HALFSCL) begin
1031             state      <= 'BURST_ACK_R_3;
1032             clkcounter <= 9'd0;
1033         end
1034         else begin    // Wait 1/2 SCL
1035             state      <= 'BURST_ACK_R_2;
1036             clkcounter <= clkcounter + 9'd1;    // Count
1037         end
1038         DRDY          <= 1'b0;    // Data is no longer valid
1039         SCL_OUT       <= 1'b1;    // Push SCL high

```

```

1040 end
1041 'BURST_ACK_R_3: begin
1042     state          <= 'DATA_BURST_R_1;
1043     SDA_EN         <= 1'b0;    // let go of SDA so slave can write
1044     SCL_OUT        <= 1'b0;    // pull SCL low
1045 end
1046 'NACK_R_1: begin
1047     if (clkcounter == 'HALFSCL) begin
1048         state          <= 'NACK_R_2;
1049         clkcounter      <= 9'd0;
1050         DRDY           <= 1'b1;    // Data is valid
1051     end
1052     else begin    // Wait 1/2 SCL
1053         state          <= 'NACK_R_1;
1054         clkcounter      <= clkcounter + 9'd1;    // Count
1055     end
1056 end
1057 'NACK_R_2: begin
1058     if (clkcounter == 'HALFSCL) begin
1059         state          <= 'NACK_R_3;
1060         clkcounter      <= 9'd0;
1061     end
1062     else begin    // Wait 1/2 SCL
1063         state          <= 'NACK_R_2;
1064         clkcounter      <= clkcounter + 9'd1;    // Count
1065     end
1066     DRDY            <= 1'b0;    // Data no longer valid
1067     SCL_OUT         <= 1'b1;    // Push SCL high
1068 end
1069 'NACK_R_3: begin
1070     if (clkcounter == 'HALFSCL) begin
1071         state          <= 'STOP_R_1;
1072         clkcounter      <= 9'd0;
1073         DONE           <= 1'b1;    // Done, let the requester know.
1074     end
1075     else begin    // Wait 1/2 SCL
1076         state          <= 'NACK_R_3;
1077         clkcounter      <= clkcounter + 9'd1;    // Count
1078     end
1079     DRDY            <= 1'b0;    // Data no longer valid
1080     SCL_OUT         <= 1'b0;    // Pull SCL low
1081     SDA_EN          <= 1'b1;    // Take SDA back
1082     SDA_OUT         <= 1'b0;    // Pull SDA low
1083 end
1084 'STOP_R_0: begin    // This state is reached when the slave fails to ACK
1085     if (clkcounter == 'HALFSCL) begin
1086         state          <= 'STOP_R_1;
1087         clkcounter      <= 9'd0;
1088     end
1089     else begin    // Wait a short amount of time
1090         state          <= 'STOP_R_0;
1091         clkcounter      <= clkcounter + 9'd1;    // Count
1092     end
1093     SCL_OUT         <= 1'b0;    // Pull SCL low
1094     SDA_EN          <= 1'b1;    // Take control of SDA
1095     SDA_OUT         <= 1'b0;    // Pull SDA low
1096 end
1097 'STOP_R_1: begin
1098     if (clkcounter == 'HALFSCL) begin
1099         state          <= 'STOP_R_2;
1100         clkcounter      <= 9'd0;
1101     end
1102     else begin    // Wait 1/2 SCL
1103         state          <= 'STOP_R_1;
1104         clkcounter      <= clkcounter + 9'd1;    // Count
1105     end
1106     DONE           <= 1'b0;    // Done is a pulse
1107     SCL_EN         <= 1'b0;    // Let SCL high (first part of a stop)

```



```

1108     end
1109     'STOP_R_2: begin
1110         if (clkcounter == 'HALFSCL) begin
1111             statuscounter    <= statuscounter + 10'd1;
1112             state            <= 'RDY_S;
1113             clkcounter       <= 9'd0;
1114             RDY              <= 1'b1;
1115         end
1116         else begin          // Wait 1/2 SCL
1117             state            <= 'STOP_R_2;
1118             clkcounter       <= clkcounter + 9'd1;    // Count
1119         end
1120         SDA_EN              <= 1'b0;    // Let SDA high (finish transaction)
1121     end
1122     default: begin
1123         state                <= 'RESET_BUS_1;
1124         burstcounter         <= 3'b000;
1125         clkcounter          <= 9'd0;
1126         RDY                 <= 1'b0;
1127         DONE                 <= 1'b0;
1128         DRDY                 <= 1'b0;
1129         SDA_EN              <= 1'b0;
1130         SDA_OUT             <= 1'b0;
1131         SCL_EN              <= 1'b0;
1132         SCL_OUT             <= 1'b0;
1133         data_size           <= 8'b0000_0000;
1134         DR                  <= 8'b0000_0000;
1135     end
1136 endcase
1137 end
1138 end
1139
1140 endmodule

```

APPENDIX L
IMUDRIVER.H/C

```
1 #ifndef IMUDRIVER_H_ // Only define once
2 #define IMUDRIVER_H_ // Only define once
3 #include <stdint.h>
4 #include "levitatefpga_hw_platform.h"
5 #include <stdio.h>
6 #include "drivers/mss_uart/mss_uart.h"
7 #include <stdio.h>
8 #include "imudriver.h"
9 #include "customtypes.h"
10
11 typedef struct
12 {
13     int32_t accel_x;
14     int32_t accel_y;
15     int32_t accel_z;
16     int32_t gyro_x;
17     int32_t gyro_y;
18     int32_t gyro_z;
19     int32_t magneto_x;
20     int32_t magneto_y;
21     int32_t magneto_z;
22 }imu_t;
23
24 int32_t get_gyro_x();
25 int32_t get_gyro_y();
26 int32_t get_accel_x();
27 int32_t get_accel_y();
28
29 #endif /* IMUDRIVER_H_ */

1 #include "imudriver.h"
2
3 static volatile imu_t* imu = (imu_t*) SENSORSTICKWRAPPER_0;
4
5 int32_t get_gyro_x()
6 {
7     return (int32_t)((int16_t) imu->gyro_x);
8 }
9
10 int32_t get_gyro_y()
11 {
12     return (int32_t)((int16_t) imu->gyro_y);
13 }
14
15 int32_t get_accel_x()
16 {
17     return (int32_t)((int16_t) imu->accel_x);
18 }
19
20 int32_t get_accel_y()
21 {
22     return (int32_t)((int16_t) imu->accel_y);
23 }
```

APPENDIX M
MAIN.C

```
1 #include <stdio.h>
2 #include <stdint.h>
3 #include "levitatefpga_hw_platform.h"
4 #include "drivers/mss_uart/mss_uart.h"
5 #include "testss.h"
6 #include "testrf.h"
7 #include "software_filter.h"
8 #include "filter.h"
9 #include "rangefinder.h"
10 #include "autopilot.h"
11 #include "pid.h"
12
13 int main()
14 {
15     NVIC_EnableIRQ(Fabric_IRQn);
16     /*initializeFilter(0,0,0);
17     while(true)
18     {
19         volatile int i;
20         for ( i = 0; i < 5000000; i++)
21         {
22             pid_update();
23             //pid_print();
24             pid_set_speeds();
25         }
26     }*/
27     ready = false;
28
29     speed1 = speed2 = speed3 = speed4 = THROTTLE_MIN;
30     initializeMotors();
31     setMotorSpeed(MOTOR_1 | MOTOR_2 | MOTOR_3 | MOTOR_4, 51000);
32
33     initializeFilter();
34     pid_initialize();
35     autopilot_initialize();
36
37     while (true) {
38         while(!ready) {
39             // wait for startup signal
40             autopilot_update();
41             remoteControl();
42         }
43         while(ready) {
44             // start
45             remoteControl();
46             if (pid_update())
47             {
48                 //char foo[512];
49                 //int messSize = sprintf(foo, "\r\nroll:%i\r\n",roll);
50                 //radio_sendStrOverRadio(foo, messSize);
51                 autopilot_update();
52                 pid_set_speeds();
53             }
54         }
55     }
56 }
57 }
58 }
59 }
```

APPENDIX N
MOTORS2.H/C

```
1 #ifndef MOTORS2_H
2 #define MOTORS2_H
3 #include "customtypes.h"
4 #include <stdint.h>
5 #include "levitatefpga_hw_platform.h"
6
7 #define THROTTLE_MIN 50000 // Should be 1ms, given a counter that counts to 1,000,000 on a 50MHz
   clock
8 #define THROTTLE_MAX 100000 // Should be 2ms, given a counter that counts to 1,000,000 on a 50MHz
   clock
9
10 int governor;
11
12 #define MOTOR_1 1 // Up = -roll
13 #define MOTOR_2 2 // Up = -pitch
14 #define MOTOR_3 4 // Up = +roll
15 #define MOTOR_4 8 // Up = +pitch
16
17 #include "CMSIS/a2fxxxm3.h"
18
19 int speed1;
20 int speed2;
21 int speed3;
22 int speed4;
23
24 typedef struct {
25     uint32_t motor1;
26     uint32_t motor2;
27     uint32_t motor3;
28     uint32_t motor4;
29     uint32_t PWMperiod;
30 } motor_t;
31
32 /*
33  * Enables all motors. Motors are enabled with throttle set to minimum.
34  *
35  * INPUTS:
36  *
37  * EXAMPLE1:
38  * initializeMotors();
39  */
40 void initializeMotors();
41
42 /*
43  * INPUTS:
44  * int motors: Motor(s) to modify speed.
45  * int speed: Speed to set motor(s) to.
46  *
47  * EXAMPLE1:
48  * setMotorSpeed(MOTOR_1, 50000); //Sets speed of motor 1 to 50000.
49  *
50  * EXAMPLE2:
51  * setMotorSpeed(MOTOR_1 | MOTOR_2 | MOTOR_3 | MOTOR_4, 25000) //Sets speed of all four motors to
   25000.
52  */
53 void setMotorSpeed(int motors, int speed);
54
55 /*
56  * Sets the period of the PWM ESC control signal.
57  * Should be between 1,000,000 (MAX) and 125,000 (MIN)
58  */
59 int setPWMPeriod(int period);
60
61 /*
62  * INPUTS:
```

```

63 * int motors: Motor(s) to modify speed.
64 * int speed: Speed increase to apply.
65 *
66 * EXAMPLE1:
67 * increaseMotorSpeed(MOTOR_1, 50000); //Increases speed of motor 1 by 50000.
68 *
69 * EXAMPLE2:
70 * increaseMotorSpeed(MOTOR_1 | MOTOR_2 | MOTOR_3 | MOTOR_4, 25000) //Increases speed of all four
    motors by 25000.
71 */
72 void increaseMotorSpeed(int motors , int speed);
73
74 /*
75 * INPUTS:
76 * int change: change to yaw (positive or negative speed change, essentially).
77 */
78 void yawUpdate(int change);
79
80 #endif /* MOTORS2_H */

1 #include "motors2.h"
2
3
4 // Motors in memory-map
5 static volatile motor_t * motorcontrol = (motor_t *)MOTORSWRAPPER_0;
6
7 /*
8 * Enables all motors. Motors are enabled with throttle set to minimum.
9 *
10 * INPUTS:
11 *
12 * EXAMPLE1:
13 * initializeMotors();
14 */
15 void initializeMotors ()
16 {
17     governor = 100000;
18     // wait for some time
19     volatile int foo = 0;
20     while (foo < 10000000) {
21         foo++;
22     }
23
24     // then set the motors to minimum throttle
25     setMotorSpeed(MOTOR_1 | MOTOR_2 | MOTOR_3 | MOTOR_4, 5500);
26
27     // then wait some more
28     foo = 0;
29     while (foo < 1000000) {
30         foo++;
31     }
32
33     // then shut them off (for safety)
34     setMotorSpeed(MOTOR_1 | MOTOR_2 | MOTOR_3 | MOTOR_4, 0);
35
36     foo = 0;
37     while (foo < 10000000) {
38         foo++;
39     }
40
41     setMotorSpeed(MOTOR_1 | MOTOR_2 | MOTOR_3 | MOTOR_4, 5500);
42
43     setPWMPeriod(125000);
44 }
45
46 /*
47 * INPUTS:
48 * int motors: Motor(s) to be enabled.

```

```

49 * int speed: Speed to set motor(s) to.
50 *
51 * EXAMPLE1:
52 * setMotorSpeed(MOTOR_1, 50000); //Sets speed of motor 1 to 50000.
53 *
54 * EXAMPLE2:
55 * setMotorSpeed(MOTOR_1 | MOTOR_2 | MOTOR_3 | MOTOR_4, 25000) //Sets speed of all four motors to
    25000.
56 */
57 void setMotorSpeed(int motors, int speed)
58 {
59     if(motors & MOTOR_1)
60     {
61         if (speed > MIN(governor, THROTTLE_MAX))
62             speed = MIN(governor, THROTTLE_MAX);
63         else if (speed < THROTTLE_MIN)
64             speed = THROTTLE_MIN;
65         motorcontrol->motor1 = speed;
66     }
67     if(motors & MOTOR_2)
68     {
69         if (speed > MIN(governor, THROTTLE_MAX))
70             speed = MIN(governor, THROTTLE_MAX);
71         else if (speed < THROTTLE_MIN)
72             speed = THROTTLE_MIN;
73         motorcontrol->motor2 = speed;
74     }
75     if(motors & MOTOR_3)
76     {
77         if (speed > MIN(governor, THROTTLE_MAX))
78             speed = MIN(governor, THROTTLE_MAX);
79         else if (speed < THROTTLE_MIN)
80             speed = THROTTLE_MIN;
81         motorcontrol->motor3 = speed;
82     }
83     if(motors & MOTOR_4)
84     {
85         if (speed > MIN(governor, THROTTLE_MAX))
86             speed = MIN(governor, THROTTLE_MAX);
87         else if (speed < THROTTLE_MIN)
88             speed = THROTTLE_MIN;
89         motorcontrol->motor4 = speed;
90     }
91 }
92
93 int setPWMPeriod(int period)
94 {
95     motorcontrol->PWMperiod = period;
96 }
97
98 /*
99 * INPUTS:
100 * int motors: Motor(s) to modify speed.
101 * int speed: Speed increase to apply.
102 *
103 * EXAMPLE1:
104 * increaseMotorSpeed(MOTOR_1, 50000); //Increases speed of motor 1 by 50000.
105 *
106 * EXAMPLE2:
107 * increaseMotorSpeed(MOTOR_1, -50000); //Decreases speed of motor 1 by 50000.
108 *
109 * EXAMPLE3:
110 * increaseMotorSpeed(MOTOR_1 | MOTOR_2 | MOTOR_3 | MOTOR_4, 25000) //Increases speed of all four
    motors by 25000.
111 */
112 void increaseMotorSpeed(int motors, int deltaSpeed)
113 {
114     if(motors & MOTOR_1)

```

```

115 {
116     setMotorSpeed(MOTOR_1, speed1 + deltaSpeed);
117 }
118 if(motors & MOTOR_2)
119 {
120     setMotorSpeed(MOTOR_2, speed2 + deltaSpeed);
121 }
122 if(motors & MOTOR_3)
123 {
124     setMotorSpeed(MOTOR_3, speed3 + deltaSpeed);
125 }
126 if(motors & MOTOR_4)
127 {
128     setMotorSpeed(MOTOR_4, speed4 + deltaSpeed);
129 }
130 }
131
132 /*
133 * INPUTS:
134 * int change: change to yaw (positive or negative speed change, essentially).
135 */
136 void yawUpdate(int change)
137 {
138     speed1 += change;
139     speed3 += change;
140     speed2 -= change;
141     speed4 -= change;
142 }

```

APPENDIX O
MOTORWRAPPER.V

```
1 // timerIntWrapper.v
2 // timerWrap.v
3 module motorswrapper(
4     PCLK,
5     PENABLE,
6     PSEL,
7     PRESETN,
8     PWRITE,
9     PREADY,
10    PSLVERR,
11    PADDR,
12    PWDATA,
13    PRDATA,
14    PWM1,
15    PWM2,
16    PWM3,
17    PWM4
18    );
19
20 // APB Bus Interface
21 input PCLK,PENABLE, PSEL, PRESETN, PWRITE;
22 input [31:0] PWDATA;
23 input [31:0] PADDR;
24 output [31:0] PRDATA;
25 output PREADY, PSLVERR;
26 output PWM1;
27 output PWM2;
28 output PWM3;
29 output PWM4;
30
31
32 assign BUS_WRITE_EN = (PENABLE && PWRITE && PSEL);
33 assign BUS_READ_EN = (!PWRITE && PSEL); //Data is ready during first cycle to make it
    available on the bus when PENABLE is asserted
34
35 assign PREADY = 1'b1;
36 assign PSLVERR = 1'b0;
37
38 motors2 motors2_0( .PCLK(PCLK),
39     .PRESETN(PRESETN),
40     .bus_write_en(BUS_WRITE_EN),
41     .bus_read_en(BUS_READ_EN),
42     .bus_addr(PADDR),
43     .bus_write_data(PWDATA),
44     .bus_read_data(PRDATA),
45     .PWM1(PWM1),
46     .PWM2(PWM2),
47     .PWM3(PWM3),
48     .PWM4(PWM4)
49     );
50
51
52 endmodule
```


APPENDIX P
MOTORS2.V

```

1 // Motors.v
2 // PWM compares effectively set motor speed. Addresses 6, 7, 8, 9 are the four PWM Compares.
3 module motors2(
4     PCLK,
5     PRESETN,
6     bus_write_en ,
7     bus_read_en ,
8     bus_addr ,
9     bus_write_data ,
10    bus_read_data ,
11    PWM1,
12    PWM2,
13    PWM3,
14    PWM4
15 );
16
17 input PCLK, PRESETN, bus_write_en , bus_read_en;
18 input [7:0] bus_addr;
19 input [31:0] bus_write_data;
20 output reg [31:0] bus_read_data;
21 output reg PWM1;
22 output reg PWM2;
23 output reg PWM3;
24 output reg PWM4;
25
26 // Bus interaction (set up compare registers)
27 reg [19:0] PWM_compare1;
28 reg [19:0] PWM_compare2;
29 reg [19:0] PWM_compare3;
30 reg [19:0] PWM_compare4;
31 reg [19:0] PWM_Period;
32 always@(posedge PCLK) begin
33     if(~PRESETN) begin
34         PWM_compare1 <= 20'd0;
35         PWM_compare2 <= 20'd0;
36         PWM_compare3 <= 20'd0;
37         PWM_compare4 <= 20'd0;
38         bus_read_data <= 20'd0;
39     end
40     else begin
41         if(bus_write_en) begin : WRITE
42             case(bus_addr[4:2])
43                 3'b000: begin
44                     PWM_compare1 <= bus_write_data;
45                 end
46                 3'b001: begin
47                     PWM_compare2 <= bus_write_data;
48                 end
49                 3'b010: begin
50                     PWM_compare3 <= bus_write_data;
51                 end
52                 3'b011: begin
53                     PWM_compare4 <= bus_write_data;
54                 end
55                 3'b101: begin
56                     PWM_Period <= bus_write_data;
57                 end
58             endcase
59         end
60     end
61 end
62
63 // [Custom]-period counter
64 reg [19:0] counter;
65 always@(posedge PCLK) begin

```

```

66     if(~PRESETN) begin
67         counter <= 20'd0;
68     end
69     else begin
70         if (counter == PWM_Period) begin
71             counter <= 20'd0;
72         end else begin
73             counter <= counter + 20'd1;
74         end
75     end
76 end
77
78 // Generate PWM signals
79 always@(posedge PCLK) begin
80     if (~PRESETN) begin
81         PWM1 <= 0;
82         PWM2 <= 0;
83         PWM3 <= 0;
84         PWM4 <= 0;
85     end
86     else begin
87         if (counter == PWM_compare1) begin
88             PWM1 <= 0;
89         end
90         else if (counter == 20'd0) begin
91             PWM1 <= 1;
92         end
93
94         if (counter == PWM_compare2) begin
95             PWM2 <= 0;
96         end
97         else if (counter == 20'd0) begin
98             PWM2 <= 1;
99         end
100
101         if (counter == PWM_compare3) begin
102             PWM3 <= 0;
103         end
104         else if (counter == 20'd0) begin
105             PWM3 <= 1;
106         end
107
108         if (counter == PWM_compare4) begin
109             PWM4 <= 0;
110         end
111         else if (counter == 20'd0) begin
112             PWM4 <= 1;
113         end
114     end
115 end
116
117 endmodule

```

APPENDIX Q
PID.H/C

```
1 /*
2  * pid.h
3  *
4  * Created on: Nov 24, 2012
5  * Author: Jon
6  */
7
8 #include "filter.h"
9 #include "motors2.h"
10 #include <stdio.h>
11 #include "customtypes.h"
12 #include "radiodriver.h"
13
14 int PID_GOVERNOR;
15 int ORIENTATION_GOVERNOR;
16 int P_CONST; // 75 //GOOD .5 // BAD 1 //BAD 1 //DECENT .75 //BAD .75 // BEST .75 //DECENT .5 //
GOOD .5
17 int I_CONST; // 0 // .01 //.9
18 int D_CONST; // 600 //GOOD 4 // BAD 8 //BAD 6 //DECENT 4 //BAD 9 //BEST 6 //DECENT 12 //GOOD 6
19 int multiplier;
20 int32_t m1, m2, m3, m4;
21
22 int32_t EQUILIBRIUM_1;
23 int32_t EQUILIBRIUM_2;
24 int32_t EQUILIBRIUM_3;
25 int32_t EQUILIBRIUM_4;
26
27 int32_t pitch;
28 int32_t roll;
29 int32_t yaw;
30
31 int32_t sum_pitch;
32 int32_t sum_roll;
33 int32_t sum_yaw;
34
35 int32_t delta_pitch;
36 int32_t delta_roll;
37 int32_t delta_yaw;
38
39 int32_t prev_pitch;
40 int32_t prev_roll;
41 int32_t prev_yaw;
42
43 void pid_initialize();
44 bool pid_update();
45 void pid_print();
46 void pid_set_speeds();
```

```
1 /*
2  * pid.c
3  *
4  * Created on: Nov 24, 2012
5  * Author: Jon
6  */
7
8 #include "pid.h"
9
10 void pid_initialize()
11 {
12     multiplier = 1;
13     PID_GOVERNOR = 50000;
14     ORIENTATION_GOVERNOR = 500;
15
16     P_CONST = 0;
17     I_CONST = 0;
```

```

18  D_CONST = 0;//600;
19
20  EQUILIBRIUM_1 = 0;
21  EQUILIBRIUM_2 = 0;
22  EQUILIBRIUM_3 = 0;
23  EQUILIBRIUM_4 = 0;
24
25  pitch = 0;
26  roll = 0;
27  yaw = 0;
28
29  sum_pitch = 0;
30  sum_roll = 0;
31  sum_yaw = 0;
32
33  delta_pitch = 0;
34  delta_roll = 0;
35  delta_yaw = 0;
36
37  prev_pitch = 0;
38  prev_roll = 0;
39  prev_yaw = 0;
40 }
41
42 bool pid_update()
43 {
44     if (data_ready())
45     {
46         prev_pitch = pitch;
47         prev_roll = roll;
48
49         pitch = getPitch();
50         roll = getRoll();
51
52         //delta_pitch = .75 * delta_pitch + .25 * (pitch - prev_pitch);
53         //delta_roll = .75 * delta_roll + .25 * (roll - prev_roll);
54         delta_pitch = (pitch - prev_pitch);
55         delta_roll = (roll - prev_roll);
56
57         if (delta_roll > ORIENTATION_GOVERNOR)
58             roll = prev_roll + ORIENTATION_GOVERNOR;
59         else if(delta_roll < -ORIENTATION_GOVERNOR)
60             roll = prev_roll - ORIENTATION_GOVERNOR;
61
62         if (delta_pitch > ORIENTATION_GOVERNOR)
63             pitch = prev_pitch + ORIENTATION_GOVERNOR;
64         else if(delta_pitch < -ORIENTATION_GOVERNOR)
65             pitch = prev_pitch - ORIENTATION_GOVERNOR;
66
67
68         sum_pitch += pitch;
69         if (sum_pitch > 50000)
70             sum_pitch = 50000;
71         if (sum_pitch < -50000)
72             sum_pitch = -50000;
73         sum_roll += roll;
74         if (sum_roll > 50000)
75             sum_roll = 50000;
76         if (sum_roll < -50000)
77             sum_roll = -50000;
78         //char foo[512];
79         //int messSize = sprintf(foo, "\r\npitch:%i roll:%i\r\n", pitch, roll);
80         //radio_sendStrOverRadio(foo, messSize);
81         return true;
82     }
83     return false;
84 }
85

```

```

86 void pid_print()
87 {
88     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_roll);
89 }
90
91 void pid_set_speeds()
92 {
93     // MOTOR_1 Up = -roll
94     // MOTOR_2 Up = -pitch
95     // MOTOR_3 Up = +roll
96     // MOTOR_4 Up = +pitch
97     int temp_roll = roll - roll_offset;
98     int temp_pitch = pitch - pitch_offset;
99
100    m1 = (P_CONST*temp_roll + D_CONST*delta_roll + I_CONST*sum_roll)/100;
101    m2 = (P_CONST*temp_pitch + D_CONST*delta_pitch + I_CONST*sum_pitch)/100;
102    m3 = (-P_CONST*temp_roll - D_CONST*delta_roll - I_CONST*sum_roll)/100;
103    m4 = (-P_CONST*temp_pitch - D_CONST*delta_pitch - I_CONST*sum_pitch)/100;
104
105    if (m1 > PID_GOVERNOR)
106        m1 = PID_GOVERNOR;
107    else if (m1 < -PID_GOVERNOR)
108        m1 = -PID_GOVERNOR;
109    if (m2 > PID_GOVERNOR)
110        m2 = PID_GOVERNOR;
111    else if (m2 < -PID_GOVERNOR)
112        m2 = -PID_GOVERNOR;
113    if (m3 > PID_GOVERNOR)
114        m3 = PID_GOVERNOR;
115    else if (m3 < -PID_GOVERNOR)
116        m3 = -PID_GOVERNOR;
117    if (m4 > PID_GOVERNOR)
118        m4 = PID_GOVERNOR;
119    else if (m4 < -PID_GOVERNOR)
120        m4 = -PID_GOVERNOR;
121
122    m1 = speed1 + m1;
123    m2 = speed2 + m2;
124    m3 = speed3 + m3;
125    m4 = speed4 + m4;
126
127    if (m1 < 55000 && speed1 > 55000)
128        m1 = 55000;
129    if (m3 < 55000 && speed3 > 55000)
130        m3 = 55000;
131    if (m2 < 55000 && speed2 > 55000)
132        m2 = 55000;
133    if (m4 < 55000 && speed4 > 55000)
134        m4 = 55000;
135
136    //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);
137
138    setMotorSpeed(MOTOR_1, m1);
139    setMotorSpeed(MOTOR_2, m2);
140    setMotorSpeed(MOTOR_3, m3);
141    setMotorSpeed(MOTOR_4, m4);
142 }

```

APPENDIX R
RADIODRIVER.H/C

```
1 #ifndef RADIODRIVER_H_ // Only define once
2 #define RADIODRIVER_H_ // Only define once
3
4 #include <stdlib.h>
5 #include <stdio.h>
6 #include <string.h>
7 #include "drivers/mss_uart/mss_uart.h"
8 #include "CMSIS/a2fxxxm3.h"
9 #include "levitatefpga_hw_platform.h"
10 #include "customtypes.h"
11
12 typedef struct {
13     uint32_t charin;
14     uint32_t charout;
15     uint32_t busy;
16 } radio_t;
17
18 typedef struct {
19     char readchar;
20     bool didreadchar;
21 } radiostatus_t;
22
23 /**
24  * Send a character over the radio
25  */
26 void radio_sendCharOverRadio(char c);
27
28 /**
29  * Send a string over the radio
30  */
31 void radio_sendStrOverRadio(char * s, int size);
32
33 /**
34  * The radio has a char for the program
35  */
36 bool radio_didReadChar();
37
38 /**
39  * Get character sent to radio
40  */
41 char radio_getChar();
42
43 #endif /* RADIODRIVER_H_ */

```



```
1 #include "radiodriver.h"
2
3 // The radio hardware at its memory map
4 // Doing it this way is better than cast-dereferencing a #define.
5 static volatile radio_t * radio = RADIOWRAPPER_0;
6
7 // Store the status
8 static radiostatus_t radiostatus = {false, '\0'};
9
10 static volatile char test;
11
12 void Fabric_IRQHandler( void )
13 {
14     // TODO -- make a memory mapped FPGA register that indicates which device triggered the
15     // interrupt.
16
17     // TODO -- grab the next character from the radio
18     if (((uint32_t*)FABINTMUX_0) == 0x1) {
19         radiostatus.readchar = radio->charin;
20         radiostatus.didreadchar = true;
21     }
22 }
```

```

21
22 // make sure to clear the interrupt
23 (*(uint32_t*)FABINTMUX_0) = 1;
24 NVIC_ClearPendingIRQ( Fabric_IRQn );
25 }
26
27 void radio_sendCharOverRadio(char c)
28 {
29 // doing only one character does not require waiting
30 radio->charout = c;
31 return;
32 }
33
34 void radio_sendStrOverRadio(char * s, int size)
35 {
36 // print one char at a time
37 int pos = 0;
38 while (pos < size) {
39 radio->charout = s[pos++];
40 while (radio->busy) {
41 // wait for radio to send the data
42 test = s[pos];
43 }
44 }
45 return;
46 }
47
48 bool radio_didReadChar()
49 {
50 return radiostatus.didreadchar;
51 }
52
53 char radio_getChar()
54 {
55 if (radiostatus.didreadchar) {
56 radiostatus.didreadchar = false;
57 return radiostatus.readchar;
58 }
59 return '\0';
60 }

```

APPENDIX S
RADIOWRAPPER.V

```

1 `define TXPERIOD 100
2
3 // radiowrapper.v
4 module radiowrapper(
5     input PCLK,
6     input PENABLE,
7     input PSEL,
8     input PRESETN,
9     input PWRITE,
10    output PREADY,
11    output PSLVERR,
12    input [31:0] PADDR,
13    input [31:0] PWDATA,
14    output reg [31:0] PRDATA,
15    input DIN,
16    output DOUT,
17    output reg FAB_INT,
18    output reg RX_LED,
19    output reg TX_LED
20 );
21
22 parameter BAUD_CYCLE_LENGTH = 5208;
23 parameter BAUD_HALF_LENGTH = 2604;
24 parameter SUSPENDED = 0;
25 parameter TRANSACTION = 1;
26
27 reg [7:0]          radio_data_in;
28
29 assign BUS_WRITE_EN = (PENABLE && PWRITE && PSEL);
30 assign BUS_READ_EN = (!PWRITE && PSEL); //Data is ready during first cycle to make it available on
    the bus when PENABLE is asserted
31
32 assign PREADY = 1'b1;
33 assign PSLVERR = 1'b0;
34
35 reg [12:0]          BAUD_COUNTER;
36 reg [12:0]          BAUD_COUNTER2;
37 reg                BAUD_RATE;
38
39 reg [7:0]           shift_reg;
40 reg [7:0]           radio_data_out;
41 reg [8:0]           shift_radio_data_out;
42 reg [3:0]           count;
43 reg [3:0]           send_count;
44
45 reg                state;
46 reg                send_state;
47 reg                send_complete;
48 reg                send_data;
49 reg                busy;
50
51 assign DOUT = shift_radio_data_out[0];
52
53 //CLOCK DIVIDE AND PUT INTO BAUD_RATE.
54 always@(posedge PCLK) begin
55     if (~PRESETN) begin
56         BAUD_COUNTER2 <= 0;
57         BAUD_RATE <= 0;
58     end
59     else begin
60         if (BAUD_COUNTER2 == BAUD_HALF_LENGTH) begin
61             BAUD_COUNTER2 <= 0;
62             BAUD_RATE <= ~BAUD_RATE;
63         end
64         else begin

```



```

65         BAUD_COUNTER2 <= BAUD_COUNTER2 + 1;
66     end
67 end
68 end
69
70
71 always@(posedge BAUD_RATE or negedge PRESETN)
72 begin
73     if (~PRESETN) begin
74         send_state <= SUSPENDED;
75         send_complete <= 0;
76         shift_radio_data_out <= 1;
77         send_count <= 0;
78         busy <= 0;
79         TX_LED <= 1;
80     end
81     else begin
82         case (send_state)
83             SUSPENDED: begin
84                 if (send_data) begin
85                     send_state <= TRANSACTION;
86                     shift_radio_data_out <= {radio_data_out, 1'b0};
87                     send_complete <= 1;
88                     busy <= 1;
89                 end
90                 else begin
91                     send_complete <= 0;
92                     busy <= 0;
93                 end
94                 TX_LED <= 1;
95             end
96
97             TRANSACTION: begin
98                 send_complete <= 0;
99                 if (send_count == 8) begin
100                     send_count <= 0;
101                     send_state <= SUSPENDED;
102                     shift_radio_data_out <= {1'b1, shift_radio_data_out[8:1]};
103                     busy <= 0;
104                 end
105                 else begin
106                     send_count <= send_count + 1;
107                     shift_radio_data_out <= {1'b1, shift_radio_data_out[8:1]};
108                 end
109                 TX_LED <= 0;
110             end
111         endcase // case (send_state)
112     end//
113 end
114
115 // This is the data coming/going on the bus.
116 always@(posedge PCLK) begin
117     if (~PRESETN) begin
118         PRDATA <= 0;
119         send_data <= 0;
120     end
121     else begin
122         if(BUS_WRITE_EN) begin
123             case(PADDR[3:2])
124                 2'b00: begin
125                     if (send_complete)
126                         send_data <= 0; //Do nothing
127                 end
128                 2'b01: begin
129                     radio_data_out <= PWDATA; //Only do this when busy is 0.
130                     send_data <= 1;
131                 end
132                 default: begin

```

```

133             if (send_complete)
134                 send_data <= 0; //Do nothing
135             end
136         endcase // case (PADDR[3:2])
137     end
138     else if(BUS_READ_EN) begin
139         case(PADDR[3:2])
140             2'b00: begin
141                 PRDATA <= radio_data_in;
142                 if (send_complete)
143                     send_data <= 0;
144             end
145             2'b10: begin
146                 PRDATA <= (busy || send_data); // If busy is 1, or send_data is 1 don't try
147                 to write to PWDATA;
148                 if (send_complete)
149                     send_data <= 0;
150             end
151             default: begin
152                 if (send_complete)
153                     send_data <= 0; //Do nothing
154             end
155         endcase // case (PADDR[2])
156     end
157     else begin
158         if (send_complete)
159             send_data <= 0;
160     end
161 end
162
163 // INPUT TRANSACTION
164 always@(posedge PCLK) begin
165     if (~PRESETN) begin
166         state <= SUSPENDED;
167         FAB_INT <= 0;
168         BAUD_COUNTER <= 0;
169         count <= 0;
170         shift_reg <= 0;
171         radio_data_in <= 0;
172         RX_LED <= 1;
173     end
174     else begin
175         case (state)
176             SUSPENDED: begin
177                 BAUD_COUNTER <= 0;
178                 FAB_INT <= 0;
179                 radio_data_in <= shift_reg;
180                 RX_LED <= 1;
181                 if (!DIN) begin
182                     state <= TRANSACTION;
183                 end
184             end
185
186             TRANSACTION: begin
187                 if (BAUD_COUNTER == BAUD_CYCLE_LENGTH) begin
188                     if (count == 8) begin
189                         count <= 0;
190                         state <= SUSPENDED;
191                         FAB_INT <= 1;
192                     end
193                     else begin
194                         count <= count + 1;
195                     end
196                 end
197                 BAUD_COUNTER <= 0;
198             end
199             else begin
200                 if (BAUD_COUNTER == BAUD_HALF_LENGTH) begin

```

```
200             shift_reg <= {DIN, shift_reg[7:1]};
201         end
202         BAUD_COUNTER <= BAUD_COUNTER + 1;
203     end
204     RX_LED <= 0;
205 end
206 endcase // case (state)
207 end
208 end
209
210 endmodule
```

APPENDIX T
RANGEFINDER.H/C

```

1 #ifndef RANGEFINDER_H_ // Only define once
2 #define RANGEFINDER_H_ // Only define once
3
4 #include <stdlib.h>
5 #include <stdio.h>
6 #include <string.h>
7 #include "CMSIS/a2fxxxm3.h"
8 #include "levitatefpga_hw_platform.h"
9 #include "customtypes.h"
10 #include "drivers/mss_ace/mss_ace.h"
11 #include "drivers_config/mss_ace/ace_handles.h"
12
13 typedef struct {
14     uint32_t init;
15 } rangefinder_t;
16
17 /**
18  * Initialize the range finders and the ACE
19  */
20 void rangefinder_init();
21
22 /**
23  * Get the value for rangefinder 0 (ADC2)
24  */
25 uint16_t rangefinder_get_rf0();
26
27 /**
28  * Get the value for rangefinder 1 (ADC3)
29  */
30 uint16_t rangefinder_get_rf1();
31
32 /**
33  * Get the value for rangefinder 2 (ADC4)
34  */
35 uint16_t rangefinder_get_rf2();
36
37 /**
38  * Get the value for rangefinder 3 (ADC5)
39  */
40 uint16_t rangefinder_get_rf3();
41
42 /**
43  * Get the value for rangefinder 4 (ADC6)
44  */
45 uint16_t rangefinder_get_rf4();
46
47 /**
48  * Get the value for rangefinder 5 (ADC7)
49  */
50 uint16_t rangefinder_get_rf5();
51
52
53 #endif /* RANGEFINDER_H_ */

1 /*
2  * rangefinder.c
3  *
4  * Created on: Oct 28, 2012
5  * Author: Daniel
6  */
7 #include "rangefinder.h"
8
9 static volatile rangefinder_t * rangefinder = RANGEFINDERWRAPPER_0;
10
11 ace_channel_handle_t rf_0_handle;

```

```

12 ace_channel_handle_t rf_1_handle;
13 ace_channel_handle_t rf_2_handle;
14 ace_channel_handle_t rf_3_handle;
15 ace_channel_handle_t rf_4_handle;
16 ace_channel_handle_t rf_5_handle;
17
18 /**
19  * Initialize the range finders
20  */
21 void rangefinder_init() {
22     // initialize the ACE
23     ACE_init();
24     rangefinder->init = 1;
25     rf_0_handle = ACE_get_channel_handle((const uint8_t *)"RF_0");
26     rf_1_handle = ACE_get_channel_handle((const uint8_t *)"RF_1");
27     rf_2_handle = ACE_get_channel_handle((const uint8_t *)"RF_2");
28     rf_3_handle = ACE_get_channel_handle((const uint8_t *)"RF_3");
29     rf_4_handle = ACE_get_channel_handle((const uint8_t *)"RF_4");
30     rf_5_handle = ACE_get_channel_handle((const uint8_t *)"RF_5");
31 }
32
33 /**
34  * Get the value for rangefinder 0 (ADC2)
35  */
36 uint16_t rangefinder_get_rf0() {
37     return 0xffc & ACE_get_ppe_sample(rf_0_handle);
38 }
39
40 /**
41  * Get the value for rangefinder 1 (ADC3)
42  */
43 uint16_t rangefinder_get_rf1() {
44     return 0xffc & ACE_get_ppe_sample(rf_1_handle);
45 }
46
47 /**
48  * Get the value for rangefinder 2 (ADC4)
49  */
50 uint16_t rangefinder_get_rf2() {
51     return 0xffc & ACE_get_ppe_sample(rf_2_handle);
52 }
53
54 /**
55  * Get the value for rangefinder 3 (ADC5)
56  */
57 uint16_t rangefinder_get_rf3() {
58     return 0xffc & ACE_get_ppe_sample(rf_3_handle);
59 }
60
61 /**
62  * Get the value for rangefinder 4 (ADC6)
63  */
64 uint16_t rangefinder_get_rf4() {
65     return 0xffc & ACE_get_ppe_sample(rf_4_handle);
66 }
67
68 /**
69  * Get the value for rangefinder 5 (ADC7)
70  */
71 uint16_t rangefinder_get_rf5() {
72     return 0xffc & ACE_get_ppe_sample(rf_5_handle);
73 }

```

APPENDIX U
RANGEFINDER2.V

```

1 `define RESET 0
2 `define WAIT 1
3 `define PULSE 2
4 `define DONE 3
5
6 // rangefinder.v
7 module rangefinder2(
8     input clk ,
9     input nreset ,
10    input rf_trigger_in ,
11    output reg rf_trigger_out ,
12    output reg rf_trigger_en ,
13    output RF_STATUS_LED
14    );
15
16    reg [1:0] state;
17    reg [23:0] counter;
18
19    assign RF_STATUS_LED = rf_trigger_in;
20
21    always@(posedge clk) begin
22        if(~nreset) begin
23            state <= 'RESET;
24            rf_trigger_en <= 1'd0;
25            rf_trigger_out <= 1'd0;
26            counter <= 24'd0;
27        end
28        else begin
29            case (state)
30                'RESET: begin
31                    state <= 'WAIT;
32                end
33                'WAIT: begin // Wait for 300ms
34                    if (counter == 24'd1500000) begin
35                        state <= 'PULSE;
36                    end
37                    else begin
38                        counter <= counter + 1'b1;
39                    end
40                end
41                'PULSE: begin // Push the output high for at least 20us
42                    rf_trigger_en <= 1'd1;
43                    rf_trigger_out <= 1'd1;
44                    if (counter == 24'd11) begin
45                        state <= 'DONE;
46                    end
47                    else begin
48                        counter <= counter + 1'b1;
49                    end
50                end
51                'DONE: begin // Do nothing else FOREVER
52                    state <= 'DONE;
53                    rf_trigger_en <= 1'd0;
54                    rf_trigger_out <= 1'd0;
55                    counter <= 24'd0;
56                end
57            endcase
58        end
59    end
60 end
61
62 endmodule

```

APPENDIX V
RANGEFINDERWRAPPER.V

```

1 // rangefinderwrapper.v
2 module rangefinderwrapper(
3     PCLK,
4     PENABLE,
5     PSEL,
6     PRESETN,
7     PWRITE,
8     PREADY,
9     PSLVERR,
10    PADDR,
11    PWDATA,
12    PRDATA,
13    rf_trigger_in ,
14    rf_trigger_out ,
15    rf_trigger_en ,
16    RF_STATUS_LED
17 );
18
19 // APB Bus Interface
20 input PCLK, PENABLE, PSEL, PRESETN, PWRITE;
21 input [31:0] PWDATA;
22 input [31:0] PADDR;
23 output [31:0] PRDATA;
24 output PREADY, PSLVERR;
25 output RF_STATUS_LED;
26
27 input rf_trigger_in;
28 output rf_trigger_en;
29 output rf_trigger_out;
30
31 assign BUS_WRITE_EN = (PENABLE && PWRITE && PSEL);
32 assign BUS_READ_EN = (!PWRITE && PSEL); //Data is ready during first cycle to make it
    availble on the bus when PENABLE is asserted
33
34 assign PREADY = 1'b1;
35 assign PSLVERR = 1'b0;
36
37 rangefinder3 rangefinder3_0(
38     .clk(PCLK),
39     .nreset(PRESETN),
40     .bus_write_en(BUS_WRITE_EN),
41     .bus_read_en(BUS_READ_EN),
42     .bus_addr(PADDR),
43     .bus_write_data(PWDATA),
44     .bus_read_data(PRDATA),
45     .rf_trigger_in(rf_trigger_in),
46     .rf_trigger_out(rf_trigger_out),
47     .rf_trigger_en(rf_trigger_en),
48     .RF_STATUS_LED(RF_STATUS_LED)
49 );
50
51 endmodule

```

APPENDIX W
SENSORSTICK.V

```

1
2 // sensorstick.v
3 module sensorstick(
4     output reg REQ_I2C,
5     output reg [6:0] ADDR_I2C,
6     output reg RW_I2C,
7     output reg [7:0] REG_I2C,
8     output reg [7:0] DATA_SIZE_I2C,
9     output reg [7:0] DATA_WRITE_I2C,
10    input [7:0] DATA_READ_I2C,
11    input DATA_READY_I2C,
12    input DONE_I2C,
13    input RDY_I2C,
14    input CLK,
15    input RESETN,
16    input RD_RQ, //Complementary filter wants new data to use.
17    input XYZ_READ, //Complementary filter has already used current data.
18    output reg XYZ_RDY, //Data for the Complementary filter is ready.
19    //Data from sensors to be used by Kalman Filter.
20    output reg [15:0] ACCEL_DATA_X,
21    output reg [15:0] ACCEL_DATA_Y,
22    output reg [15:0] ACCEL_DATA_Z,
23    output reg [15:0] GYRO_DATA_X,
24    output reg [15:0] GYRO_DATA_Y,
25    output reg [15:0] GYRO_DATA_Z,
26    output reg [15:0] MAGNETO_DATA_X,
27    output reg [15:0] MAGNETO_DATA_Y,
28    output reg [15:0] MAGNETO_DATA_Z
29 );
30
31 // I2C Interface
32 //input  DONE_I2C, RDY_I2C, DATA_READY_I2C;
33 //input  [7:0] DATA_READ_I2C;
34 //output RW_I2C, REQ_I2C;
35 //output [7:0] ADDR_I2C;
36 //output [7:0] REG_I2C;
37 //output [7:0] DATA_WRITE_I2C;
38
39 // Also, you should have an APB wrapper as part of this wrapper so the software can talk to you
40
41 reg [5:0] state;
42
43 //Addresses
44 parameter ACCEL_ADDR = 8'hA6;
45 parameter MAGNETO_ADDR = 8'h3C;
46 parameter GYRO_ADDR = 8'hD0;
47 parameter ACCEL_REG_START = 8'h32;
48 parameter MAGNETO_REG_START = 8'h03;
49 parameter GYRO_REG_START = 8'h1D;
50
51 //States
52 parameter SUSPENDED = 0;
53 parameter ACCEL_INIT = 1;
54 parameter ACCEL_WAIT = 2;
55 parameter MAGNETO_INIT_1 = 3;
56 parameter MAGNETO_WAIT_1 = 4;
57 parameter MAGNETO_INIT_2 = 5;
58 parameter MAGNETO_WAIT_2 = 6;
59 parameter MAGNETO_INIT_3 = 7;
60 parameter MAGNETO_WAIT_3 = 8;
61 parameter GYRO_INIT_1 = 9;
62 parameter GYRO_WAIT_1 = 10;
63 parameter ACCEL_READ_ACCEPT_X_1 = 11;
64 parameter ACCEL_READ_ACCEPT_X_2 = 12;

```



```

65 parameter ACCEL_READ_ACCEPT_Y_1 = 13;
66 parameter ACCEL_READ_ACCEPT_Y_2 = 14;
67 parameter ACCEL_READ_ACCEPT_Z_1 = 15;
68 parameter ACCEL_READ_ACCEPT_Z_2 = 16;
69 parameter MAGNETO_READ_ACCEPT_X_1 = 17;
70 parameter MAGNETO_READ_ACCEPT_X_2 = 18;
71 parameter MAGNETO_READ_ACCEPT_Y_1 = 19;
72 parameter MAGNETO_READ_ACCEPT_Y_2 = 20;
73 parameter MAGNETO_READ_ACCEPT_Z_1 = 21;
74 parameter MAGNETO_READ_ACCEPT_Z_2 = 22;
75 parameter GYRO_READ_ACCEPT_X_1 = 23;
76 parameter GYRO_READ_ACCEPT_X_2 = 24;
77 parameter GYRO_READ_ACCEPT_Y_1 = 25;
78 parameter GYRO_READ_ACCEPT_Y_2 = 26;
79 parameter GYRO_READ_ACCEPT_Z_1 = 27;
80 parameter GYRO_READ_ACCEPT_Z_2 = 28;
81 parameter ACCEL_READ = 29;
82 parameter MAGNETO_READ = 30;
83 parameter GYRO_READ_1 = 31;
84 parameter ACCEL_READ_WAIT = 32;
85 parameter MAGNETO_READ_WAIT = 33;
86 parameter GYRO_READ_WAIT = 34;
87 parameter GYRO_INIT_2 = 35;
88 parameter GYRO_WAIT_2 = 36;
89 parameter GYRO_INIT_3 = 37;
90 parameter GYRO_WAIT_3 = 38;
91 parameter GYRO_INIT_4 = 39;
92 parameter GYRO_WAIT_4 = 40;
93 parameter GYRO_READ_2 = 41;
94 parameter GYRO_READ_3 = 42;
95
96 always@(posedge CLK)
97 begin
98     if(~RESETN)
99         begin
100             state <= ACCEL_INIT;
101         end
102     else
103         begin
104             case( state )
105
106 /*
107 void IMU_ACCEL_init() {
108     // initialize accelerometer
109     uint8_t write_buf[2];
110     write_buf[0] = 0x2D; // register address
111     write_buf[1] = 0x08; // data
112     // ask for data
113     MSS_I2C_write
114     (
115         &g_mss_i2c1,
116         ACCELEROMETER,
117         write_buf,
118         2,
119         MSS_I2C_RELEASE_BUS
120     );
121
122     while(MSS_I2C_wait_complete(&g_mss_i2c1) == MSS_I2C_IN_PROGRESS) {
123         __asm__("nop");
124     }
125 }
126 */
127
128         ACCEL_INIT:
129         begin
130             if(RDY_I2C)
131                 begin
132                     state <= ACCEL_WAIT;

```

```

133         ADDR_I2C <= ACCEL_ADDR[7:1];
134         REG_I2C <= 8'h2D;
135         DATA_WRITE_I2C <= 8'h08;
136         DATA_SIZE_I2C <= 2;
137         REQ_I2C <= 1;
138         RW_I2C <= 0;
139     end
140 end
141
142 ACCEL_WAIT:
143 begin
144     if (DONE_I2C)
145         begin
146             state <= MAGNETO_INIT_1;
147             ADDR_I2C <= 0;
148             REG_I2C <= 0;
149             DATA_WRITE_I2C <= 0;
150             DATA_SIZE_I2C <= 0;
151             REQ_I2C <= 0;
152             RW_I2C <= 0;
153         end
154     end
155
156     /*
157 void IMU_MAGNETO_init() {
158 uint8_t write_buf[2];
159 // initialize magnetometer
160 write_buf[0] = 0x00; // register address
161 write_buf[1] = 0x70; // data
162 MSS_I2C_write
163 (
164     &g_mss_i2c1,
165     MAGNETOMETER,
166     write_buf,
167     2,
168     MSS_I2C_RELEASE_BUS
169 );
170
171 write_buf[0] = 0x01; // register address
172 write_buf[1] = 0xA0; // data
173 MSS_I2C_write
174 (
175     &g_mss_i2c1,
176     MAGNETOMETER,
177     write_buf,
178     2,
179     MSS_I2C_RELEASE_BUS
180 );
181
182 while(MSS_I2C_wait_complete(&g_mss_i2c1) == MSS_I2C_IN_PROGRESS) {
183     __asm__("nop");
184 }
185 write_buf[0] = 0x02; // register address
186 write_buf[1] = 0x00; // data
187 MSS_I2C_write
188 (
189     &g_mss_i2c1,
190     MAGNETOMETER,
191     write_buf,
192     2,
193     MSS_I2C_RELEASE_BUS
194 );
195
196 while(MSS_I2C_wait_complete(&g_mss_i2c1) == MSS_I2C_IN_PROGRESS) {
197     __asm__("nop");
198 }
199     */
200

```

```

201     MAGNETO_INIT_1:
202     begin
203         if(RDY_I2C)
204             begin
205                 state <= MAGNETO_WAIT_1;
206                 ADDR_I2C <= MAGNETO_ADDR[7:1];
207                 REG_I2C <= 8'h00;
208                 DATA_WRITE_I2C <= 8'h70;
209                 DATA_SIZE_I2C <= 2;
210                 REQ_I2C <= 1;
211                 RW_I2C <= 0;
212             end
213         end
214
215     MAGNETO_WAIT_1:
216     begin
217         if(DONE_I2C)
218             begin
219                 state <= MAGNETO_INIT_2;
220                 ADDR_I2C <= 0;
221                 REG_I2C <= 0;
222                 DATA_WRITE_I2C <= 0;
223                 DATA_SIZE_I2C <= 0;
224                 REQ_I2C <= 0;
225                 RW_I2C <= 0;
226             end
227         end
228
229     MAGNETO_INIT_2:
230     begin
231         if(RDY_I2C)
232             begin
233                 state <= MAGNETO_WAIT_2;
234                 ADDR_I2C <= MAGNETO_ADDR[7:1];
235                 REG_I2C <= 8'h01;
236                 DATA_WRITE_I2C <= 8'hA0;
237                 DATA_SIZE_I2C <= 2;
238                 REQ_I2C <= 1;
239                 RW_I2C <= 0;
240             end
241         end
242
243     MAGNETO_WAIT_2:
244     begin
245         if(DONE_I2C)
246             begin
247                 state <= MAGNETO_INIT_3;
248                 ADDR_I2C <= 0;
249                 REG_I2C <= 0;
250                 DATA_WRITE_I2C <= 0;
251                 DATA_SIZE_I2C <= 0;
252                 REQ_I2C <= 0;
253                 RW_I2C <= 0;
254             end
255         end
256
257     MAGNETO_INIT_3:
258     begin
259         if(RDY_I2C)
260             begin
261                 state <= MAGNETO_WAIT_3;
262                 ADDR_I2C <= MAGNETO_ADDR[7:1];
263                 REG_I2C <= 8'h02;
264                 DATA_WRITE_I2C <= 8'h00;
265                 DATA_SIZE_I2C <= 2;
266                 REQ_I2C <= 1;
267                 RW_I2C <= 0;
268             end

```

```

269         end
270
271     MAGNETO_WAIT_3:
272     begin
273         if(DONE_I2C)
274             begin
275                 state <= GYRO_INIT_1;
276                 ADDR_I2C <= 0;
277                 REG_I2C <= 0;
278                 DATA_WRITE_I2C <= 0;
279                 DATA_SIZE_I2C <= 0;
280                 REQ_I2C <= 0;
281                 RW_I2C <= 0;
282             end
283         end
284
285     /*
286     void IMU_GYRO_init() {
287         // initialize gyro
288         uint8_t write_buf[2];
289         write_buf[0] = 0x3E; // register address
290         write_buf[1] = 0x01; // data
291         // ask for data
292         MSS_I2C_write
293         (
294             &g_mss_i2c1,
295             GYRO,
296             write_buf,
297             2,
298             MSS_I2C_RELEASE_BUS
299         );
300         while(MSS_I2C_wait_complete(&g_mss_i2c1) == MSS_I2C_IN_PROGRESS) {
301             __asm__("nop");
302         }
303     */
304
305     GYRO_INIT_1:
306     begin
307         if(RDY_I2C)
308             begin
309                 state <= GYRO_WAIT_1;
310                 ADDR_I2C <= GYRO_ADDR[7:1];
311                 REG_I2C <= 8'h3E;
312                 DATA_WRITE_I2C <= 8'h01;
313                 DATA_SIZE_I2C <= 2;
314                 REQ_I2C <= 1;
315                 RW_I2C <= 0;
316             end
317         end
318
319     GYRO_WAIT_1:
320     begin
321         if (DONE_I2C)
322             begin
323                 state <= GYRO_INIT_2;
324                 ADDR_I2C <= 0;
325                 REG_I2C <= 0;
326                 DATA_WRITE_I2C <= 0;
327                 DATA_SIZE_I2C <= 0;
328                 REQ_I2C <= 0;
329                 RW_I2C <= 0;
330             end
331         end
332
333         // Write a 19 (0x13) to register 21 (0x15) to set the divider to 19, meaning the Gyro
334         // samples at 50Hz.
335     GYRO_INIT_2:
336     begin

```

```

336         if(RDY_I2C)
337             begin
338                 state <= GYRO_WAIT_2;
339                 ADDR_I2C <= GYRO_ADDR[7:1];
340                 REG_I2C <= 8'h15;
341                 DATA_WRITE_I2C <= 8'h13;
342                 DATA_SIZE_I2C <= 2;
343                 REQ_I2C <= 1;
344                 RW_I2C <= 0;
345             end
346         end
347
348     GYRO_WAIT_2:
349     begin
350         if (DONE_I2C)
351             begin
352                 state <= GYRO_INIT_3;
353                 ADDR_I2C <= 0;
354                 REG_I2C <= 0;
355                 DATA_WRITE_I2C <= 0;
356                 DATA_SIZE_I2C <= 0;
357                 REQ_I2C <= 0;
358                 RW_I2C <= 0;
359             end
360         end
361
362         // Write a 25 (0x19) to register 22 (0x16) to set the internal rate to 1kHz and not
363         // get bogus data.
364     GYRO_INIT_3:
365     begin
366         if(RDY_I2C)
367             begin
368                 state <= GYRO_WAIT_3;
369                 ADDR_I2C <= GYRO_ADDR[7:1];
370                 REG_I2C <= 8'h16;
371                 DATA_WRITE_I2C <= 8'b11011;
372                 DATA_SIZE_I2C <= 2;
373                 REQ_I2C <= 1;
374                 RW_I2C <= 0;
375             end
376         end
377
378     GYRO_WAIT_3:
379     begin
380         if (DONE_I2C)
381             begin
382                 state <= GYRO_INIT_4;
383                 ADDR_I2C <= 0;
384                 REG_I2C <= 0;
385                 DATA_WRITE_I2C <= 0;
386                 DATA_SIZE_I2C <= 0;
387                 REQ_I2C <= 0;
388                 RW_I2C <= 0;
389             end
390         end
391
392         // Write a 1 (0x1) to register 23 (0x17) to set the interrupt whenever data is ready.
393         // Interrupt is cleared upon any reg read.
394     GYRO_INIT_4:
395     begin
396         if(RDY_I2C)
397             begin
398                 state <= GYRO_WAIT_4;
399                 ADDR_I2C <= GYRO_ADDR[7:1];
400                 REG_I2C <= 8'h17;
401                 DATA_WRITE_I2C <= 8'h1;

```

```

402             DATA_SIZE_I2C <= 2;
403             REQ_I2C <= 1;
404             RW_I2C <= 0;
405         end
406     end
407
408     GYRO_WAIT_4:
409     begin
410         if (DONE_I2C)
411             begin
412                 state <= SUSPENDED;
413                 ADDR_I2C <= 0;
414                 REG_I2C <= 0;
415                 DATA_WRITE_I2C <= 0;
416                 DATA_SIZE_I2C <= 0;
417                 REQ_I2C <= 0;
418                 RW_I2C <= 0;
419             end
420         end
421     end
422
423     /*
424     vector3u16_t IMU_ACCEL_read() {
425         vector3u16_t volatile data;
426         MSS_I2C_write_read
427         (
428             &g_mss_i2c1,
429             ACCELEROMETER,
430             &(ACCELEROMETER_REG_START), // the register that starts the data registers
431             1,
432             (uint8_t*)&data,
433             6,
434             MSS_I2C_RELEASE_BUS
435         );
436         while(MSS_I2C_wait_complete(&g_mss_i2c1) == MSS_I2C_IN_PROGRESS) {
437             __asm__("nop");
438         }
439         return data;
440     }
441     */
442
443     ACCEL_READ:
444     begin
445         if(RDY_I2C)
446             begin
447                 state <= ACCEL_READ_ACCEPT_X_1;
448                 ADDR_I2C <= ACCEL_ADDR[7:1];
449                 RW_I2C <= 1;
450                 REQ_I2C <= 1;
451                 REG_I2C <= ACCEL_REG_START;
452                 DATA_SIZE_I2C <= 6;
453             end
454         end
455
456     ACCEL_READ_ACCEPT_X_1:
457     begin
458         if(DATA_READY_I2C)
459             begin
460                 ACCEL_DATA_Y[7:0] <= DATA_READ_I2C;
461                 state <= ACCEL_READ_ACCEPT_X_2;
462             end
463         end
464
465     ACCEL_READ_ACCEPT_X_2:
466     begin
467         if(DATA_READY_I2C)
468             begin
469                 ACCEL_DATA_Y[15:8] <= DATA_READ_I2C;

```

```

470         state <= ACCEL_READ_ACCEPT_Y_1;
471     end
472 end
473
474 ACCEL_READ_ACCEPT_Y_1:
475 begin
476     if(DATA_READY_I2C)
477     begin
478         ACCEL_DATA_X[7:0] <= DATA_READ_I2C;
479         state <= ACCEL_READ_ACCEPT_Y_2;
480     end
481 end
482
483 ACCEL_READ_ACCEPT_Y_2:
484 begin
485     if(DATA_READY_I2C)
486     begin
487         ACCEL_DATA_X[15:8] <= DATA_READ_I2C;
488         state <= ACCEL_READ_ACCEPT_Z_1;
489     end
490 end
491
492 ACCEL_READ_ACCEPT_Z_1:
493 begin
494     if(DATA_READY_I2C)
495     begin
496         ACCEL_DATA_Z[7:0] <= DATA_READ_I2C;
497         state <= ACCEL_READ_ACCEPT_Z_2;
498     end
499 end
500
501 ACCEL_READ_ACCEPT_Z_2:
502 begin
503     if(DATA_READY_I2C)
504     begin
505         ACCEL_DATA_Z[15:8] <= DATA_READ_I2C;
506         state <= ACCEL_READ_WAIT;
507     end
508 end
509
510 ACCEL_READ_WAIT:
511 begin
512     if (DONE_I2C)
513     begin
514         state <= MAGNETO_READ;
515         ADDR_I2C <= 0;
516         REG_I2C <= 0;
517         DATA_WRITE_I2C <= 0;
518         DATA_SIZE_I2C <= 0;
519         REQ_I2C <= 0;
520         RW_I2C <= 0;
521     end
522 end
523
524 /*
525 vector3u16_t IMU_MAGNETO_read() {
526     vector3u16_t volatile data;
527     // read the data
528     MSS_I2C_write_read
529     (
530         &g_mss_i2c1,
531         MAGNETOMETER,
532         &(MAGNETOMETER_REG_START), // the register that starts the data registers
533         1,
534         (uint8_t*)&data,
535         6,
536         MSS_I2C_RELEASE_BUS
537     );

```

```

538
539 while(MSS_I2C_wait_complete(&g_mss_i2c1) == MSS_I2C_IN_PROGRESS) {
540     __asm__("nop");
541 }
542
543 // change endianness
544 data.x = (data.x << 8) | ((data.x >> 8) & 0x00ff);
545 data.y = (data.y << 8) | ((data.y >> 8) & 0x00ff);
546 data.z = (data.z << 8) | ((data.z >> 8) & 0x00ff);
547
548 return data;
549 }
550 */
551
552     MAGNETO_READ:
553     begin
554         if(RDY_I2C)
555             begin
556                 state <= MAGNETO_READ_ACCEPT_X_1;
557                 ADDR_I2C <= MAGNETO_ADDR[7:1];
558                 RW_I2C <= 1;
559                 REQ_I2C <= 1;
560                 REG_I2C <= MAGNETO_REG_START;
561                 DATA_SIZE_I2C <= 6;
562             end
563     end
564
565     MAGNETO_READ_ACCEPT_X_1:
566     begin
567         if(DATA_READY_I2C)
568             begin
569                 MAGNETO_DATA_X[7:0] <= DATA_READ_I2C;
570                 state <= MAGNETO_READ_ACCEPT_X_2;
571             end
572     end
573
574     MAGNETO_READ_ACCEPT_X_2:
575     begin
576         if(DATA_READY_I2C)
577             begin
578                 MAGNETO_DATA_X[15:8] <= DATA_READ_I2C;
579                 state <= MAGNETO_READ_ACCEPT_Y_1;
580             end
581     end
582
583     MAGNETO_READ_ACCEPT_Y_1:
584     begin
585         if(DATA_READY_I2C)
586             begin
587                 MAGNETO_DATA_Y[7:0] <= DATA_READ_I2C;
588                 state <= MAGNETO_READ_ACCEPT_Y_2;
589             end
590     end
591
592     MAGNETO_READ_ACCEPT_Y_2:
593     begin
594         if(DATA_READY_I2C)
595             begin
596                 MAGNETO_DATA_Y[15:8] <= DATA_READ_I2C;
597                 state <= MAGNETO_READ_ACCEPT_Z_1;
598             end
599     end
600
601     MAGNETO_READ_ACCEPT_Z_1:
602     begin
603         if(DATA_READY_I2C)
604             begin
605                 MAGNETO_DATA_Z[7:0] <= DATA_READ_I2C;

```



```

606         state <= MAGNETO_READ_ACCEPT_Z_2;
607     end
608 end
609
610 MAGNETO_READ_ACCEPT_Z_2:
611 begin
612     if (DATA_READY_I2C)
613     begin
614         MAGNETO_DATA_Z[15:8] <= DATA_READ_I2C;
615         state <= MAGNETO_READ_WAIT;
616     end
617 end
618
619 MAGNETO_READ_WAIT:
620 begin
621     if (DONE_I2C)
622     begin
623         state <= GYRO_READ_1;
624         ADDR_I2C <= 0;
625         REG_I2C <= 0;
626         DATA_WRITE_I2C <= 0;
627         DATA_SIZE_I2C <= 0;
628         REQ_I2C <= 0;
629         RW_I2C <= 0;
630     end
631 end
632
633
634 /*
635 vector3u16_t IMU_GYRO_read() {
636     vector3u16_t volatile data;
637     MSS_I2C_write_read
638     (
639         &g_mss_i2c1,
640         GYRO,
641         &(GYRO_REG_START),    // the register that starts the data registers
642         1,
643         (uint8_t*)&data,
644         6,
645         MSS_I2C_RELEASE_BUS
646     );
647
648     while(MSS_I2C_wait_complete(&g_mss_i2c1) == MSS_I2C_IN_PROGRESS) {
649         __asm__("nop");
650     }
651
652     // change endianness
653     data.x = (data.x << 8) | ((data.x >> 8) & 0x00ff);
654     data.y = (data.y << 8) | ((data.y >> 8) & 0x00ff);
655     data.z = (data.z << 8) | ((data.z >> 8) & 0x00ff);
656
657
658     return data;
659 }
660 */
661
662     // Check status register to see if new data is available.
663 GYRO_READ_1:
664 begin
665     if(RDY_I2C)
666     begin
667         state <= GYRO_READ_2;
668         ADDR_I2C <= GYRO_ADDR[7:1];
669         RW_I2C <= 1;
670         REQ_I2C <= 1;
671         REG_I2C <= 8'h1a;
672         DATA_SIZE_I2C <= 1;
673     end

```

```

674     end
675
676     GYRO_READ_2:
677     begin
678         if(DATA_READY_I2C)
679             begin
680                 if(DATA_READ_I2C[0])
681                     begin
682                         state <= GYRO_READ_3;
683                     end
684                     else
685                         begin
686                             state <= GYRO_READ_1;
687                         end
688                         ADDR_I2C <= 0;
689                         RW_I2C <= 0;
690                         REQ_I2C <= 0;
691                         REG_I2C <= 0;
692                         DATA_SIZE_I2C <= 0;
693                     end
694             end
695
696     GYRO_READ_3:
697     begin
698         if(RDY_I2C)
699             begin
700                 state <= GYRO_READ_ACCEPT_X_1;
701                 ADDR_I2C <= GYRO_ADDR[7:1];
702                 RW_I2C <= 1;
703                 REQ_I2C <= 1;
704                 REG_I2C <= GYRO_REG_START;
705                 DATA_SIZE_I2C <= 6;
706             end
707         end
708
709     GYRO_READ_ACCEPT_X_1:
710     begin
711         if(DATA_READY_I2C)
712             begin
713                 GYRO_DATA_X[15:8] <= DATA_READ_I2C;
714                 state <= GYRO_READ_ACCEPT_X_2;
715             end
716         end
717
718     GYRO_READ_ACCEPT_X_2:
719     begin
720         if(DATA_READY_I2C)
721             begin
722                 GYRO_DATA_X[7:0] <= DATA_READ_I2C;
723                 state <= GYRO_READ_ACCEPT_Y_1;
724             end
725         end
726
727     GYRO_READ_ACCEPT_Y_1:
728     begin
729         if(DATA_READY_I2C)
730             begin
731                 GYRO_DATA_Y[15:8] <= DATA_READ_I2C;
732                 state <= GYRO_READ_ACCEPT_Y_2;
733             end
734         end
735
736     GYRO_READ_ACCEPT_Y_2:
737     begin
738         if(DATA_READY_I2C)
739             begin
740                 GYRO_DATA_Y[7:0] <= DATA_READ_I2C;
741                 state <= GYRO_READ_ACCEPT_Z_1;

```

```

742     end
743 end
744
745 GYRO_READ_ACCEPT_Z_1:
746 begin
747     if(DATA_READY_I2C)
748         begin
749             GYRO_DATA_Z[15:8] <= DATA_READ_I2C;
750             state <= GYRO_READ_ACCEPT_Z_2;
751         end
752     end
753
754 GYRO_READ_ACCEPT_Z_2:
755 begin
756     if(DATA_READY_I2C)
757         begin
758             GYRO_DATA_Z[7:0] <= DATA_READ_I2C;
759             state <= GYRO_READ_WAIT;
760         end
761     end
762
763 GYRO_READ_WAIT:
764 begin
765     if (DONE_I2C)
766         begin
767             state <= SUSPENDED;
768             ADDR_I2C <= 0;
769             REG_I2C <= 0;
770             DATA_WRITE_I2C <= 0;
771             DATA_SIZE_I2C <= 0;
772             REQ_I2C <= 0;
773             RW_I2C <= 0;
774             XYZ_RDY <= 1;
775         end
776     end
777
778     // Might want to add else's to if(DATA_READY_I2C) that check if DONE/ERROR from I2C, go to
779     // suspended and output an error bit to Kalman so it knows it didn't get new data.
780
781 SUSPENDED:
782 begin
783     if(RD_RQ && RDY_I2C)
784         begin
785             state <= ACCEL_READ;
786             ADDR_I2C <= 0;
787             REG_I2C <= 0;
788             DATA_WRITE_I2C <= 0;
789             DATA_SIZE_I2C <= 0;
790             REQ_I2C <= 0;
791             XYZ_RDY <= 0;
792         end
793     else if (XYZ_READ)
794         begin
795             XYZ_RDY <= 0;
796         end
797     end
798 endcase
799 end
800 end
801
802 endmodule

```

APPENDIX X
SENSORSTICKWRAPPER.V

```

1
2 // sensorstickwrapper.v
3 module sensorstickwrapper(
4     output REQ_I2C,
5     output [6:0] ADDR_I2C,
6     output RW_I2C,
7     output [7:0] REG_I2C,
8     output [7:0] DATA_SIZE_I2C,
9     output [7:0] DATA_WRITE_I2C,
10    input [7:0] DATA_READ_I2C,
11    input DATA_READY_I2C,
12    input DONE_I2C,
13    input RDY_I2C,
14
15
16    input RD_RQ, //Complementary filter wants new data to use.
17    input XYZ_READ, //Complementary filter has already used current data.
18    output XYZ_RDY, //Data for the Complementary filter is ready.
19
20    input PCLK,
21    input PENABLE,
22    input PSEL,
23    input PRESETN,
24    input PWRITE,
25    output PREADY,
26    output PSLVERR,
27    input [31:0] PADDR,
28    input [31:0] PWDATA,
29    output reg [31:0] PRDATA,
30
31    output [15:0] ACCEL_DATA_X,
32    output [15:0] ACCEL_DATA_Y,
33    output [15:0] ACCEL_DATA_Z,
34    output [15:0] GYRO_DATA_X,
35    output [15:0] GYRO_DATA_Y,
36    output [15:0] GYRO_DATA_Z,
37    output [15:0] MAGNETO_DATA_X,
38    output [15:0] MAGNETO_DATA_Y,
39    output [15:0] MAGNETO_DATA_Z
40 );
41
42 assign BUS_WRITE_EN = (PENABLE && PWRITE && PSEL);
43 assign BUS_READ_EN = (!PWRITE && PSEL); //Data is ready during first cycle to make it available
44    on the bus when PENABLE is asserted
45
46 assign PREADY = 1'b1;
47 assign PSLVERR = 1'b0;
48
49 always@(posedge PCLK) begin
50     if(~PRESETN) begin
51         // nothing to do on reset
52     end
53     else begin
54         if(BUS_READ_EN)
55             begin
56                 case(PADDR[5:2])
57                     4'b0000:
58                         PRDATA[31:0] <= {16'h0, ACCEL_DATA_X};
59                 end
60
61                 4'b0001:
62                     begin
63                         PRDATA <= {16'h0, ACCEL_DATA_Y};
64                     end

```

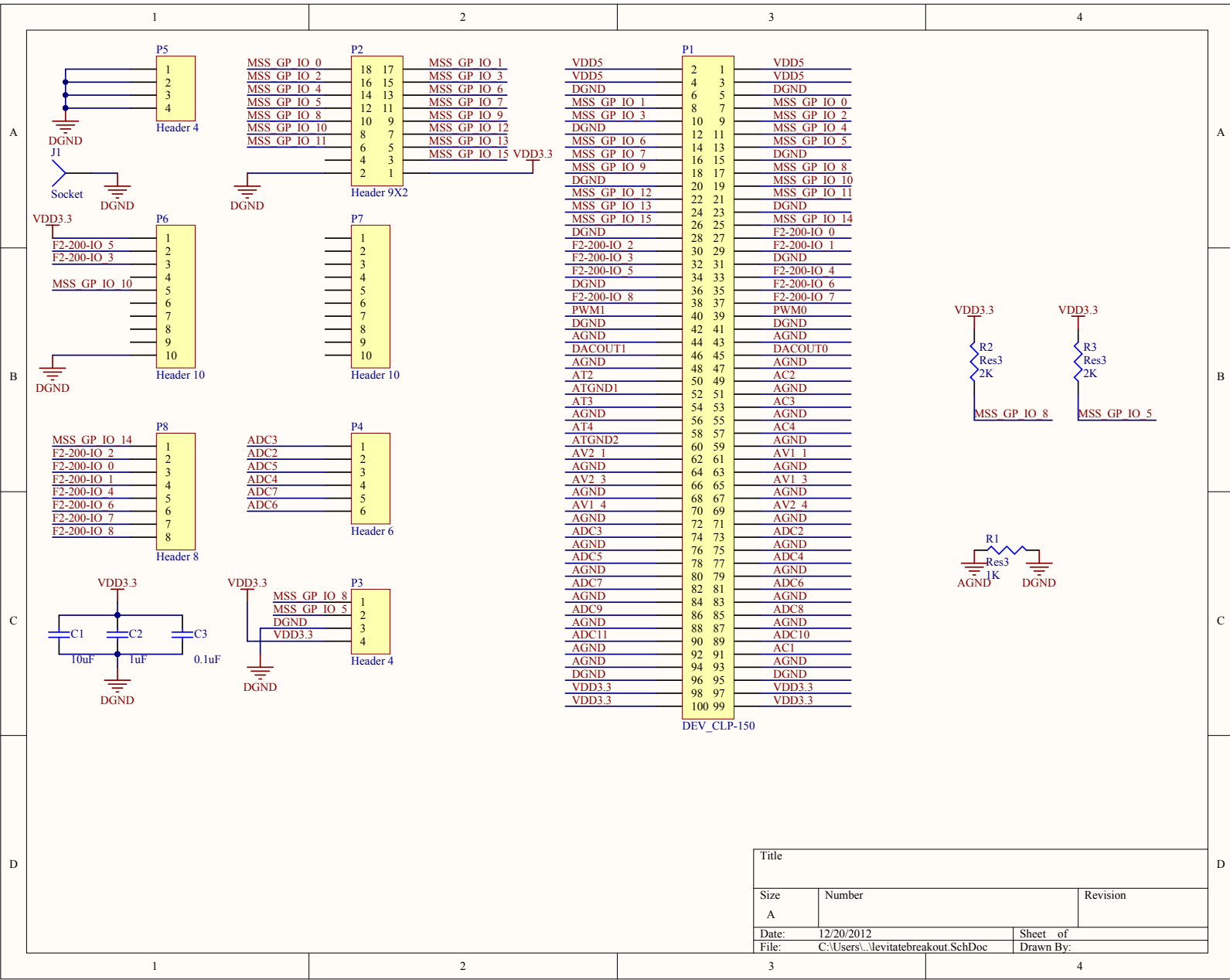
```

65
66     4'b0010:
67     begin
68         PRDATA <= {16'h0, ACCEL_DATA_Z};
69     end
70
71     4'b0011:
72     begin
73         PRDATA <= {16'h0, GYRO_DATA_X};
74     end
75
76     4'b0100:
77     begin
78         PRDATA <= {16'h0, GYRO_DATA_Y};
79     end
80
81     4'b0101:
82     begin
83         PRDATA <= {16'h0, GYRO_DATA_Z};
84     end
85
86     4'b0110:
87     begin
88         PRDATA <= {16'h0, MAGNETO_DATA_X};
89     end
90
91     4'b0111:
92     begin
93         PRDATA <= {16'h0, MAGNETO_DATA_Y};
94     end
95
96     4'b1000:
97     begin
98         PRDATA <= {16'h0, MAGNETO_DATA_Z};
99     end
100    endcase
101    end
102 end
103 end
104
105 sensorstick sensorstick_0(
106     .REQ_I2C(REQ_I2C),
107     .ADDR_I2C(ADDR_I2C),
108     .RW_I2C(RW_I2C),
109     .REG_I2C(REG_I2C),
110     .DATA_SIZE_I2C(DATA_SIZE_I2C),
111     .DATA_WRITE_I2C(DATA_WRITE_I2C),
112     .DATA_READ_I2C(DATA_READ_I2C),
113     .DATA_READY_I2C(DATA_READY_I2C),
114     .DONE_I2C(DONE_I2C),
115     .RDY_I2C(RDY_I2C),
116     .CLK(PCLK),
117     .RESETN(PRESETN),
118     .RD_RQ(RD_RQ), //Kalman filter wants new data to use.
119                     //Data from sensor
120     .XYZ_READ(XYZ_READ), //Complementary filter has already used current data.
121     .XYZ_RDY(XYZ_RDY), //Data for the Complementary filter is ready.
122
123     .ACCEL_DATA_X(ACCEL_DATA_X),
124     .ACCEL_DATA_Y(ACCEL_DATA_Y),
125     .ACCEL_DATA_Z(ACCEL_DATA_Z),
126     .GYRO_DATA_X(GYRO_DATA_X),
127     .GYRO_DATA_Y(GYRO_DATA_Y),
128     .GYRO_DATA_Z(GYRO_DATA_Z),
129     .MAGNETO_DATA_X(MAGNETO_DATA_X),
130     .MAGNETO_DATA_Y(MAGNETO_DATA_Y),
131     .MAGNETO_DATA_Z(MAGNETO_DATA_Z)
132 );

```

```
133  
134 endmodule
```

APPENDIX Y LEVITATE DAUGHTERBOARD SCHEMATIC



Title		
Size	Number	Revision
A		
Date:	12/20/2012	Sheet of
File:	C:\Users\levitatebreakout.SchDoc	Drawn By:

APPENDIX Z
LEVITATE DAUGHTERBOARD PCB

