FPGA Quadcopter Capstone Project Report

2017

Team:

Svyatoslav Zhuchenya
Charley Hill
Casey Montgomery
Lucas Myers

Sponsor:

Company Representative: Rahulkumar Koche

Faculty Advisor: Malgorzata Chrzanowska-Jeske

Maseeh College of Engineering and Computer Science
PORTLAND STATE UNIVERSITY

Contents

Abstract

This project explores the versatility and capabilities of Lattice Semiconductor FPGA technology in motor control applications by attempting to use a Lattice FPGA as a quadcopter drone flight controller. The belief is that a single FPGA can replace multiple microprocessors present on existing flight controllers, and that doing so could potentially reduce the power consumption of such devices while simultaneously reducing the cost. Additional hoped for benefits are ease of redesign and streamlined development. This project will be attempted using a Lattice MachXO2 FPGA development kit in conjunction with a Cheerson CX-20 drone. Primary development will be done in Verilog using Lattice Diamond IDE. The end goal will be a drone that has basic flight functions provided by an FPGA instead of a microcontroller.

Motivation

Our primary motivation was to provide an expanded view of the potential intrinsic to FPGA technology. By establishing that an FPGA can effectively replace multiple microcontrollers in a drone, it shows that the technology can be an asset not only in systems with multiple motor control, but in any system that has multiple disparate inputs and outputs. Moreover, utilizing the ability of hardware implementation to perform multiple tasks truly in parallel, can often supply a clear advantage over software and microprocessor based solutions. FPGAs allow for these solutions to be rapidly developed and implemented within a complex system.

Problem Identification

Problem Statement

Drones that are currently on the market typically utilize multiple micro-controller infrastructure that often includes expensive high-end processors. These are utilized to control many elements, such as motor control, signal generation, feedback, and others. There are a couple of problems with this solution. The use of multiple micro-controllers raises the cost of the end product and the power utilized by the systems needed to keep the drone in the air, thus reducing maximum flight time. Development time is also extended as designers must spend development time making systems which were developed independently communicate together.

A more ideal solution would be to design multiple systems to work in parallel on a single chip supplied by Lattice Semiconductors. The result would be lower power consumption and a lower BOM (bill of materials) cost. It potentially also greatly eases development, as the modules could be designed to communicate more smoothly within the chip. An additional benefit would be for system redesigns. Sensors and other external components that communicate with the FPGA would not require the change of board components if they communicate differently. The module that communicates with that component would simply be rewritten and uploaded to the board, potentially saving even larger sums in development in the long term.

Solution / Deliverables

Therefore, in order to explore the potential solution to the problems discussed, the primary deliverable for this project will be a quadcopter with basic flight controls (throttle, roll, pitch and yaw) from a radio remote controller. This quadcopter is to have a flight controller replaced with a Lattice FPGA development board. The device will be limited to very basic manual flight, having no automation features or complex sensors such as; GPS, accelerometer, gyroscope, barometer, etc.

All Verilog source code used in this project will also be delivered. The source code will be clear and will include appropriate comments and documentation to ensure future utilization of the code is possible by future parties.

Project documentation will include thorough explanation of all relevant aspects of the project, hardware, software, and logistical. This will be done in such a way as to ensure understanding of the project so that any future party can pick up and continue the project or take it in another direction with as little complication as possible.

Approach

We begin the development phase of the project with a design plan and approach that will lead to adequate exploration and solutions of the problems defined in our Problem Statement, and one that will fulfill the requirements outlined in the project deliverables.

Quadcopter Selection

A key component to the project and the first step in development was selecting the quadcopter platform that was to be used. There were a couple approaches that were considered, one of which was using a DIY drone platform and building the quadcopter from scratch. This approach was not taken due to the potential unwanted complexities of building the drone and potential shipping wait times for parts. The approach that was taken was to buy an off the shelf quadcopter and "hack" it to be controlled by an FPGA.

The drone that we were to purchase was to be relatively inexpensive, due to the potential need to purchase a replacement. The device was to have a large enough size to house the MachXO2 board. It also needed to be a brushless motor drone (per request of the project sponsor), and have good online community support. Therefore, the team purchased the Cheerson CX-20 which is a larger, open source, brushless motor drone that met all the described requirements. From this quadcopter, we were able to have most of the necessary parts to be able to complete the project.

Figure 1: Cheerson CX-20 Quadcopter

Flight Controller Characterization

From the beginning, it was clear that the existing flight controller unit software was far too complex to completely emulate on our FPGA design. Presumably the microcontroller may be mapping the signals using some differential equations or other complex math expressions. Due to the limitations in Verilog with complex math, we needed a simpler approach.

In order to match the signal mapping characteristics of our FPGA to the existing flight controller, we decided to take on a “black box” approach. With this approach, we do not need to fully understand and replicate all the functions of the flight controller. All we would look at is how the outputs react to the inputs of the controller. We achieve this by characterizing the types of signals coming in and out of the existing flight controller. This way we can have the FPGA simply map the input signals to the corresponding output behavior.

Software Development

After we characterized and defined the inputs and outputs that our FPGA flight controller would have. The next step would be to start the software development phase. Since the project is a team project, it was extremely important to be able to work on most of the code as a team. This was done by splitting the Verilog code into separate modules with their own functions and explicitly defined inputs and outputs that each member could work on, simulate, test and verify independently.

Integration

Once all the Verilog modules were completed the next step would be to integrate each module and program the FPGA. After the basic functionality of the FPGA was verified, we could begin to construct to drone and connect all the hardware inputs and outputs. Completing this would allow for a full testing platform. We scheduled this milestone early on during Spring term in order to account for re-design time and potential setbacks.

Testing and Debugging

The final stage of the design process would be the testing and debugging phase. In this phase, we would begin to verify the flight characteristics of the quadcopter itself and make any necessary modifications to the code until basic flight is achieved.

Characterization

Radio Receiver Signal Characterization

Objective:

The purpose of this testing was to characterize the signals coming out of the radio receiver on the Cheerson CX-20 quadcopter. We are interested in what type of signals our FPGA Verilog code is required to receive. The assumption is that the receiver, like the signals that control the electronic motor speed controllers, are Pulse Width Modulated (PWM) signals. We are interested in characterizing the duty cycles at the maximum and minimum points of all the basic controls (throttle, pitch, yaw and roll) that we are interested in from the radio controller.

Pinout:

The figure below demonstrates which pins we are interested in on the receiver outputs. These are the pins that will act as the inputs into the FPGA. The goal is to take these signals and map them to produce the proper motor control signals to fly the quadcopter with the remote control.

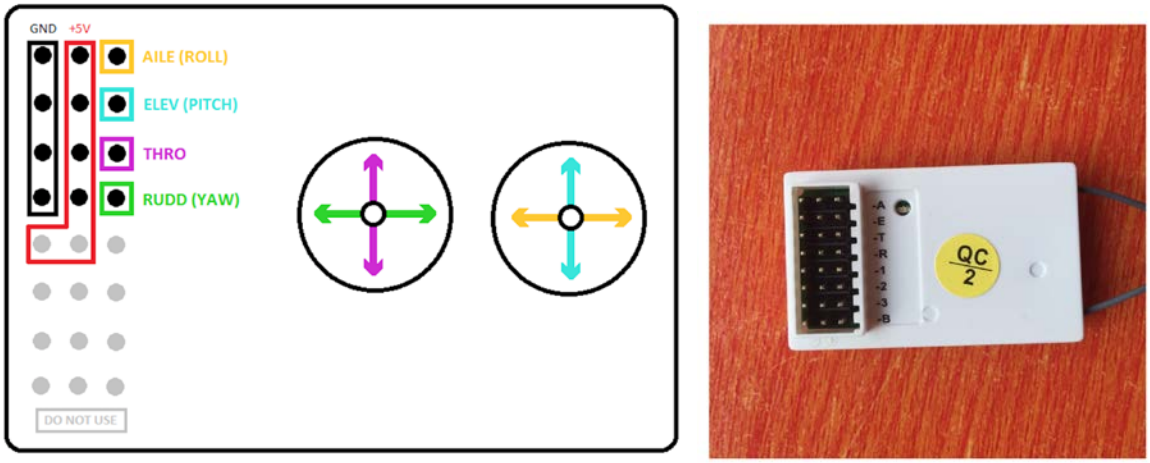

Figure 2: Cheerson CX-20 Radio Receiver

Data:

The following figure is a screenshot of what the receiver signals look like on an oscilloscope. The signals appear to be 5 Volt pulse width modulated signals. The duty cycle of the signals varies from 5% to about 10% for the extremes of each control. The signals are connected in the following order.

Channel 1: Roll
Channel 2: Pitch
Channel 3: Throttle
Channel 4: Yaw

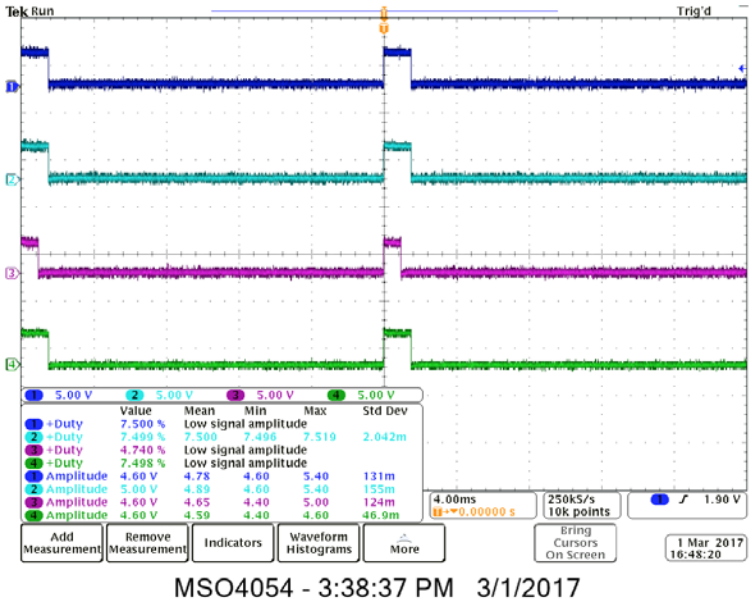

Figure 3: Radio Receiver Signals - Idle State

After taking measurements of all the control limits (i.e. roll all the way right, all the way left), we were able to construct the following table of duty cycle values for each of the controls.

Table 1: CX-20 Radio Receiver Control Signal Duty Cycles

Control	Left/Up/Min	Idle	Right/Down/Max
Roll	5.0%	7.5%	9.98%
Pitch	5.04%	7.49%	9.98%
Throttle	4.74%	---	9.88%
Yaw	4.99%	7.49%	9.89%

Electronic Speed Control Signals Characterization

Objective

In the Cheerson CX-20, the flight controller sends a signal to an Electronic Speed Controller (ESC). This is a device that controls the speed of the brushless motors of the drone. Since we will also be using the existing ESCs, we need to characterize the signals going into them from the controller. This way we will know what signals our FPGA flight controller will need to be outputting in order to give us control of the motor speeds.

Data

We are assuming that the actual signal mapping for all the controls in the flight controller is far too complicated, and most likely using complex math to give us the observable outputs. Therefore, with this characterization report we only look at the throttle, which should at least give us the signals required to give us the motor maximum and minimum speeds.

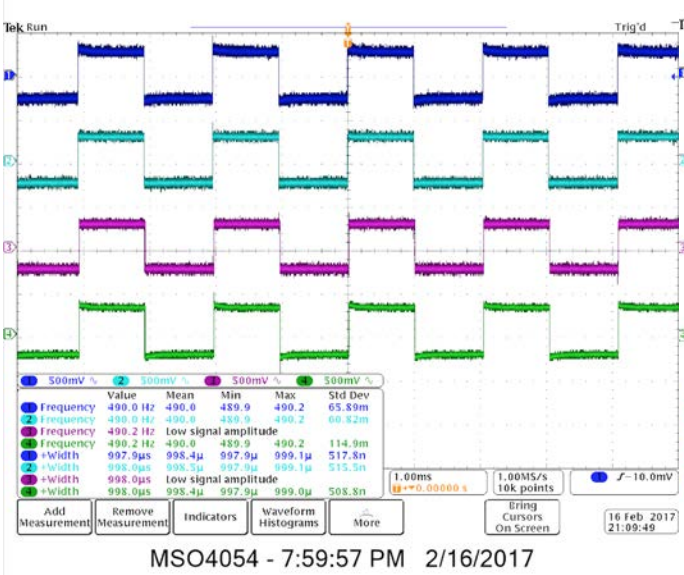

Figure 4: Signals to ESC - Throttle Min

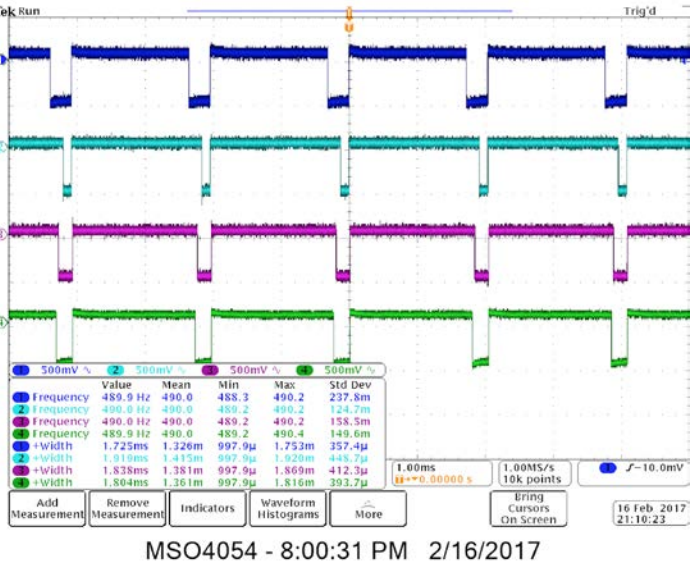

Figure 5: Signals to ESC - Throttle Max

We can see that the signals going into the ECSs are also PWM signals that control the motor speeds using the duty cycle. We determined that ideally the motors are off at < 50% duty cycle, on at > 50% duty cycle, and are at full speed at 100% duty cycle. Realistically theses duty cycle values were observed to be slightly off and slightly different for each of the motors. This is presumably due to some of the initial calibration that was done, or simple physical discrepancies.

Next, we attempted to characterize the changes in duty cycles of each of the motors based on the ranges of each of the controls. It was observed that the changes were not consistent for different throttle positions. The table below is a small sample of some of the duty cycle values for different levels of throttle, from it we can observe the complexity of the signal mapping. Ideally, we would want to map more of the increments in the control ranges, however the following data will give us a good starting point and understanding of how the duty cycles change for the different control combinations, and allow us to determine a design approach for our Verilog code.

Table 2: Variable Throttle Duty Cycle Values

Measured motor output behavior, expressed as a percentage change in duty cycle.

	duty cycle at motors (throttle ONLY)			measured change in motor PWM duty cycle (corresponding change in receiver PWM output)					
	throttle (idle)	throttle (X)	% change	roll left (-2.5%)	roll right (+2.5%)	pitch down (+2.5%)	pitch up (-2.5%)	yaw left (-2.5%)	yaw right (+2.5%)
Motor 1 (min)	48.9	55	6.1	0	15	0	15	0	17
Motor 2 (min)	48.9	64	15.1	11	-4	10	-5	-2	14
Motor 3 (min)	48.9	59	10.1	10	-5	0	11	15	-4
Motor 4 (min)	48.9	60	11.1	0	12	12	-6	12	3
Motor 1 (mid)	48.9	78	29.1	-5	5	-5	5	-20	10
Motor 2 (mid)	48.9	85	36.1	5	-8	4	-7	-20	10
Motor 3 (mid)	48.9	73	24.1	5	-8	-6	8	20	-15
Motor 4 (mid)	48.9	75	26.1	-5	7	4	-10	10	-20
Motor 1 (max)	48.9	87	38.1	-10	4	-8	7	-20	0
Motor 2 (max)	48.9	94	45.1	0	-10	0	-8	-20	0
Motor 3 (max)	48.9	90	41.1	0	-10	-8	5	0	-20
Motor 4 (max)	48.9	92	43.1	-10	4	0	-8	0	-20

Design Overview

Software

Verilog Modules

This section will supply a description of the Verilog modules and diagrams to aid understanding. The actual Verilog code will be included separately.

The code will be separated into four distinct types of modules. We begin by describing the types and then describe the way that they interact with each other.

The module that we call the “offset generator” does most of the work. There will be four variations of the module: throttle, pitch, roll, and yaw. The purpose of the module is to take in the magnitude of the signal that corresponds to the modules variation (throttle, pitch, roll, yaw) and determine how much to adjust each of the four motors, generating a corresponding offset. This is the only module with multiple variations and contains only combinational logic.

The PWM reader module takes in the PWM signal directly from the radio receiver and interprets the signal into a corresponding magnitude. It only exists in one variation and is instantiated once in each of the four offset generator modules.

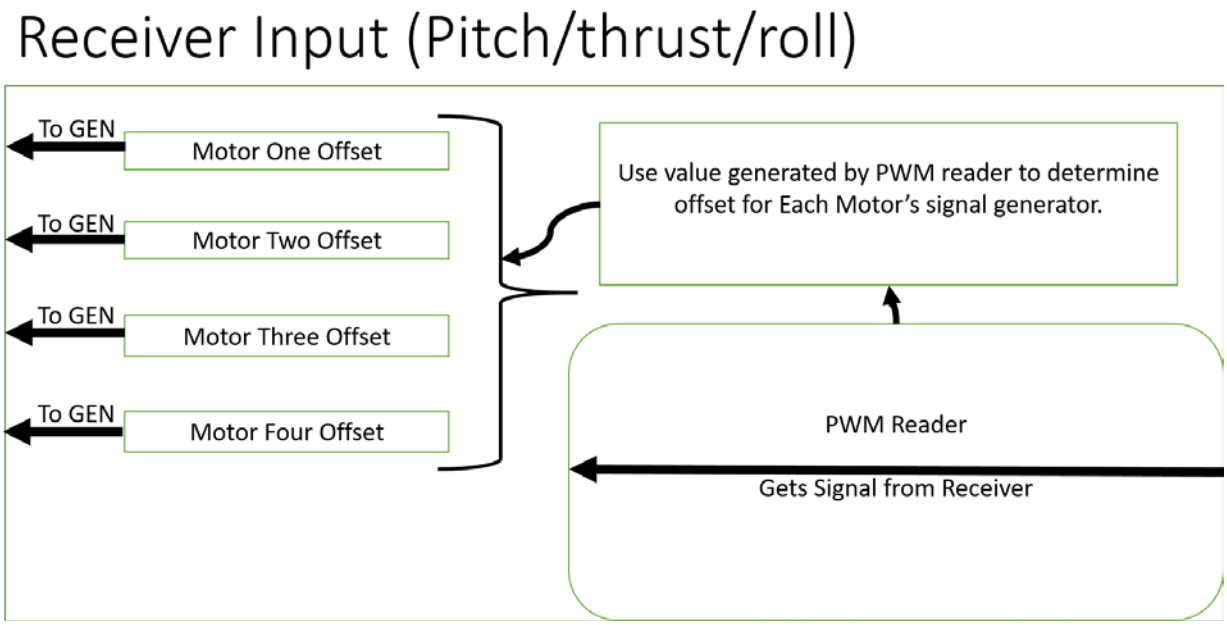

Figure 6: Receiver Input Verilog Block Diagram

Each signal generator module takes in an offset from each of the offset generator modules and combines them, with a predefined base value, to create a value that represents the speed at which that specific motor should be moving. Within each signal generator is instantiated a PWM generator.

The PWM generator takes in the value supplied by the signal generator and generates a PWM signal that is then sent to the external ESC.

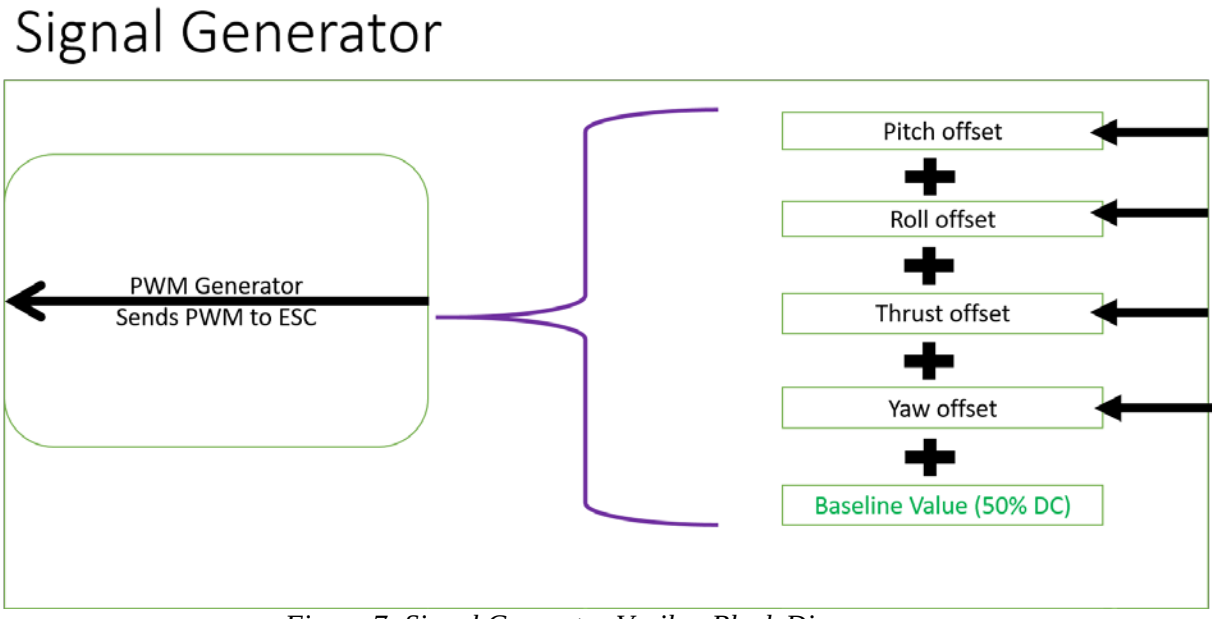

Figure 7: Signal Generator Verilog Block Diagram

By having four offset generators working in parallel, each one supplying an offset to all four signal generators, which are also working in parallel, we are able to easily and elegantly take into account all of the control signals

to each motor without conflict between the signals or significant delay. The PWM reader provides the entry port into the entire design and PWM generator provides the final output from the design.

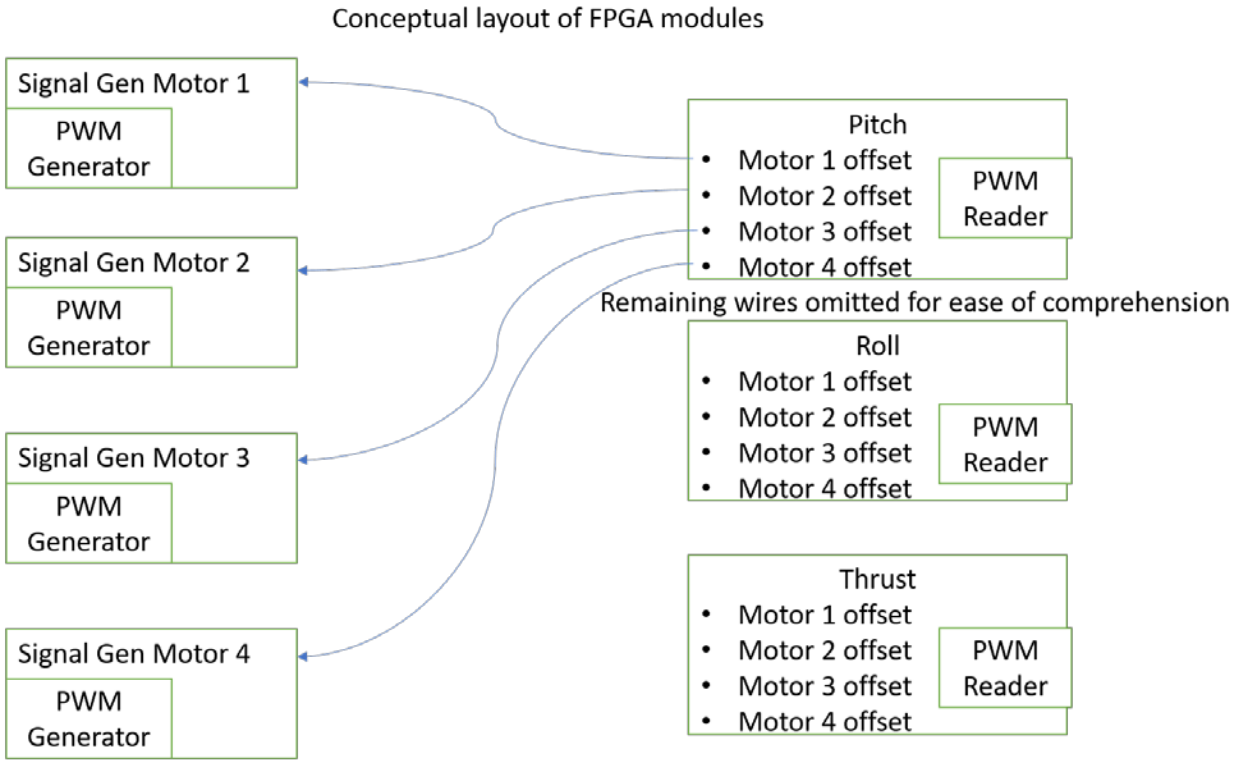

Figure 8: Main Verilog Block Diagram

Hardware

Since we used an off the shelf quadcopter, most of the hardware that is required for a functioning drone was already in place. This included essential components; the motors, ESCs, frame, propellers, power distribution board, remote and receiver.

Our task now was to modify the hardware to use our MachXO2 board in place of the flight controller. To do this we took out the old flight controller as well as all non-essential hardware modules such as the external sensors. This freed up enough space to snuggly fit the FPGA board.

Block Diagram

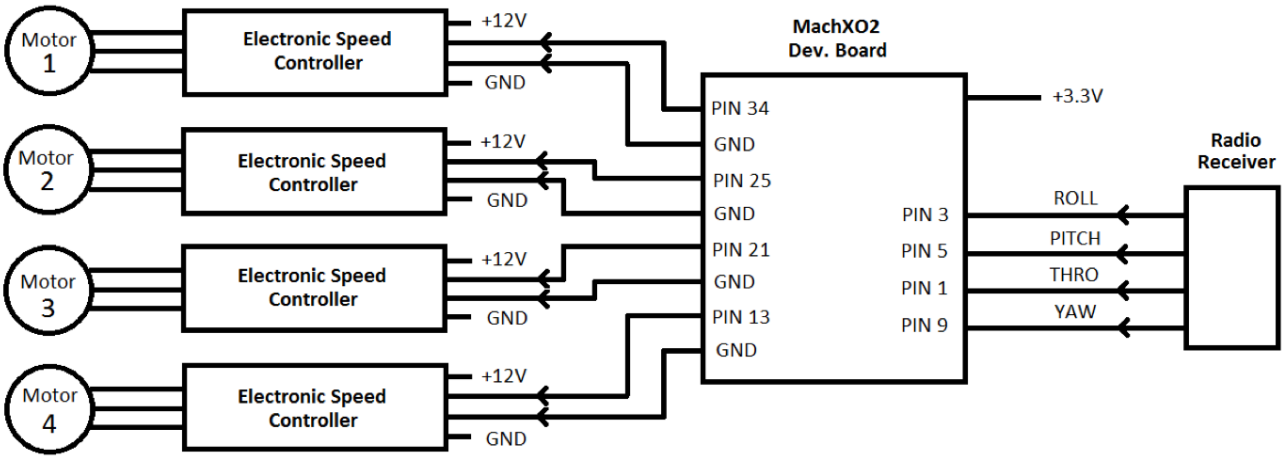

Figure 9: Hardware Block Diagram

This is a basic block diagram of the MachXO2 hardware setup and connections. The pin labels are the corresponding pin connections and assignments of the FPGA developer board. We can see that the basic inputs are from the Cheerson CX-20 drone radio receiver and the outputs of the FPGA flight controller drive the electronic speed controller for each of the corresponding motors. The motor number assignments are based on a standard x-configuration quadcopter numbering (shown below).

Figure 10: X-configuration Quadcopter Motor Numbering

Hardware Build

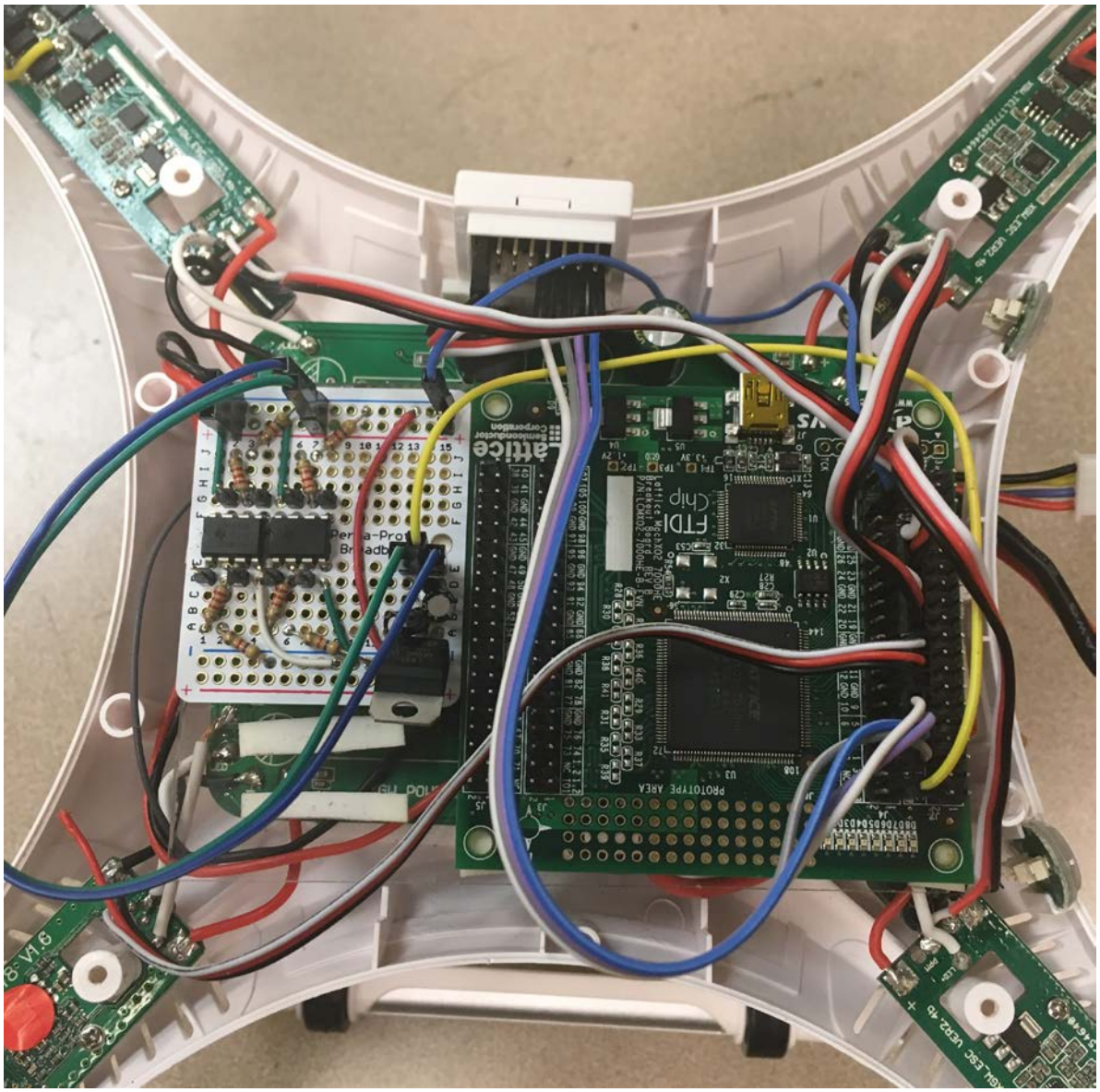

Figure 11: MachXO2 Board Inside Cheerson CX-20 Frame

FPGA Resource Utilization

The following information is obtained from Lattice Diamond, illustrating the FPGA resource utilization. In other words; the information presented shows us how much of the FPGA is actually being used.

Cost Table Summary

Level/ Cost [ncd]	Number Unrouted	Worst Slack	Timing Score	Worst Slack(hold)	Timing Score(hold)	Run Time	NCD Status
5_1 *	0	4.164	0	0.379	0	11	Complete

Device utilization summary:

PIO (prelim)	17+4(JTAG)/336	6% used
	17+4(JTAG)/115	18% bonded
IOLOGIC	12/336	3% used
SLICE	647/3432	18% used
OSC	1/1	100% used

```
Number of Signals: 1809
Number of Connections: 4248

Pin Constraint Summary:
   17 out of 17 pins locked (100% locked).


I/O Usage Summary (final):
   17 + 4(JTAG) out of 336 (6.3%) PIO sites used.
   17 + 4(JTAG) out of 115 (18.3%) bonded PIO sites used.
   Number of PIO comps: 17; differential: 0.
   Number of Vref pins used: 0.

I/O Bank Usage Summary:
+----------+---------------+------------+-----------+
| I/O Bank | Usage         | Bank Vccio | Bank Vref |
+----------+---------------+------------+-----------+
| 0        | 0 / 28 (  0%) | -          | -         |
| 1        | 9 / 29 ( 31%) | 2.5V       | -         |
| 2        | 0 / 29 (  0%) | -          | -         |
| 3        | 2 / 9 ( 22%)  | 2.5V       | -         |
| 4        | 2 / 10 ( 20%) | 2.5V       | -         |
| 5        | 4 / 10 ( 40%) | 2.5V       | -         |
+----------+---------------+------------+-----------+

Total placer CPU time: 6 secs
```

```
Design Summary
   Number of registers:    401 out of  7209 (6%)
      PFU registers:          389 out of  6864 (6%)
      PIO registers:           12 out of   345 (3%)
   Number of SLICEs:       647 out of  3432 (19%)
      SLICEs as Logic/ROM:    647 out of  3432 (19%)
      SLICEs as RAM:            0 out of  2574 (0%)
      SLICEs as Carry:        279 out of  3432 (8%)
   Number of LUT4s:        1289 out of  6864 (19%)
      Number used as logic LUTs:        731
      Number used as distributed RAM:     0
      Number used as ripple logic:      558
      Number used as shift registers:     0
   Number of PIO sites used: 17 + 4(JTAG) out of 115 (18%)
   Number of block RAMs:  0 out of 26 (0%)
   Number of GSRs:  0 out of 1 (0%)
   EFB used :       No
   JTAG used :      No
   Readback used :  No
   Oscillator used :  Yes
   Startup used :   No
   POR :            On
   Bandgap :        On
   Number of Power Controller:  0 out of 1 (0%)
   Number of Dynamic Bank Controller (BCINRD):  0 out of 6 (0%)
   Number of Dynamic Bank Controller (BCLVDSO):  0 out of 1 (0%)
   Number of DCCA:  0 out of 8 (0%)
   Number of DCMA:  0 out of 2 (0%)
   Number of PLLs:  0 out of 2 (0%)
   Number of DQSDLLs:  0 out of 2 (0%)
   Number of CLKDIVC:  0 out of 4 (0%)
   Number of ECLKSYNCA:  0 out of 4 (0%)
   Number of ECLKBRIDGECS:  0 out of 2 (0%)
   Notes:-
      1. Total number of LUT4s = (Number of logic LUT4s) + 2*(Number of
     distributed RAMs) + 2*(Number of ripple logic)
      2. Number of logic LUT4s does not include count of distributed RAM and
     ripple logic.
   Number of clocks:  1
     Net clk: 236 loads, 236 rising, 0 falling (Driver: clock_inst/OSCH_inst )
   Number of Clock Enables:  12
```

We can see from the resource data presented that the FPGA is not being used to its full potential. For most of the components listed on the report, such as the registers and LUTs, less that 20% are being utilized. This means that the device definitely has room for future expandability. This could prove to be extremely useful if the project was to be continued and the other external sensors were to be implemented. This data could be interpreted in a sense that a smaller (cheaper?) FPGA could easily do the work that this project requires.

Testing

Flight Testing

From initial flight tests, it was observed that when throttle only was engaged, the quadcopter had a tendency to pitch, roll or yaw (or a combination of all three) slightly upon achieving lift, and any attempt to correct this slight shifting would result in the drone flying out of control. One hypothesis is that the motors, although are receiving the same values, are realistically spinning at different speeds. This would explain why when no other controls are engaged, the device is unstable. This discrepancy between motor speed and FPGA output signals is in all likelihood a result of some sort of calibration built into the hardware of the ESCs themselves, but the removal of the microprocessor-based flight controller module makes the ESCs unable to be calibrated.

A couple of weeks into the flight testing it became evident that in the absence of feedback from an accelerometer or gyroscope, we cannot possibly expect autonomous flight stability. The focus at that time was then shifted to attempting to improve the stability to the point that basic flight could still be achieved, even though we were quite certain that full long duration flight would not be possible. The hope was that if we could, by observation and tethered flight testing, get the motors to spin at nearly the same speed when throttle was engaged, we could achieve relatively stable lift and have the ability to at least marginally control the quadcopter manually. One such method we used in an attempt to balance the motor speeds was to use hall-effect sensors to measure the motor speed directly.

Motor Speeds

Objective:

This section is a test report of the motor speeds of the FPGA controlled quadcopter. The purpose of this test is to locate the source of the general flight instability of the quadcopter when the electronic speed controls receive equal throttle values from the FPGA, and correct said issues to provide stable lift.

Hall-effect sensor test setup:

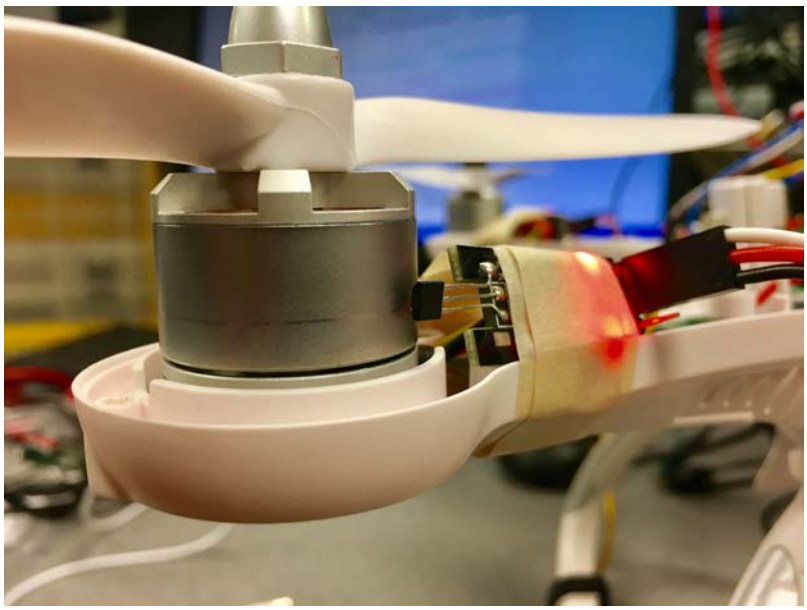

Figure 12: Hall Effect Sensor Motor Speed Test Setup

To measure the actual motor speed and see how all the motors compare to each other, we started by mounting hall effect sensors to next to the motors. The sensors used (PN: LYSB009M86TFG-CMPTRACCS) have a digital output. The output goes high when the required magnetic field is measured. In the brushless motors used on the drone, there are a series of magnets that spin around as the motor spins. Therefore, from this setup, we can see a pulse signal on an oscilloscope by measuring the digital outputs of the sensors (as shown in the figure below). We will be comparing the frequency of each pulse signal for each of the motors for several different throttle positions on the remote.

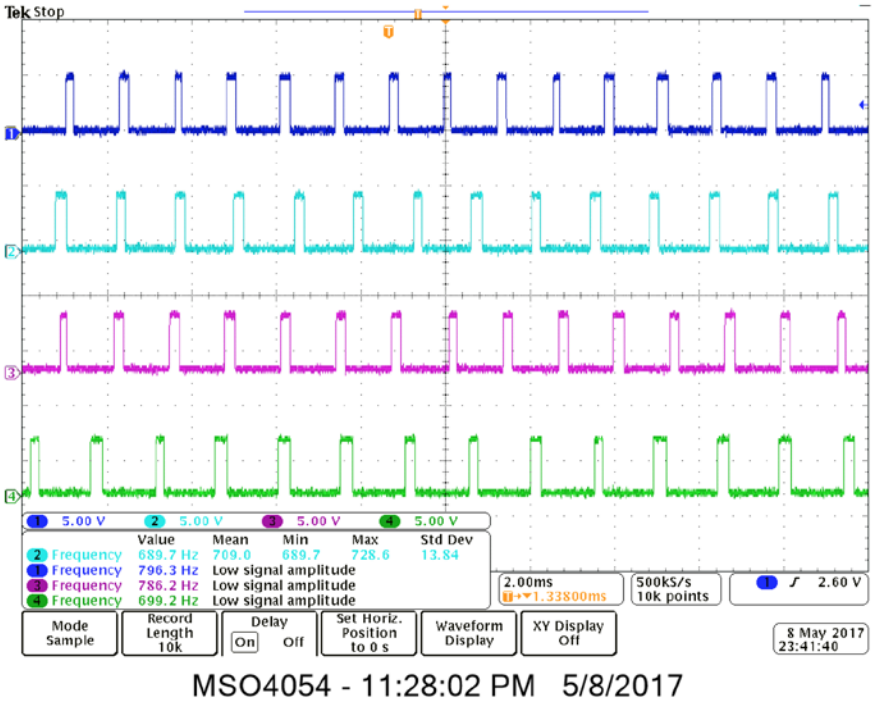

Figure 13: Hall Effect Sensor Digital Out for 25% Throttle

Data:

Data was collected over several different throttle values and for both configurations; with propellers and without propellers.

Table 3: Hall Effect Data (No Propellers)

Motor	Throttle Position	Hall D_{out} Freq. (Hz)
1	25%	767.3
	50%	1225
	75%	1287
	100%	1371
2	25%	705.2
	50%	1222
	75%	1300
	100%	1358
3	25%	736.4
	50%	1186
	75%	1182
	100%	1230
4	25%	659.7
	50%	1150
	75%	1211
	100%	1279

Table 4: Hall Effect Data (With Propellers)

Motor	Throttle Position	Hall Dout Freq. (Hz)
1	50%	535.2
	75%	702.4
2	50%	509.2
	75%	700.3
3	50%	492.7
	75%	677.5
4	50%	492.7
	75%	652.6

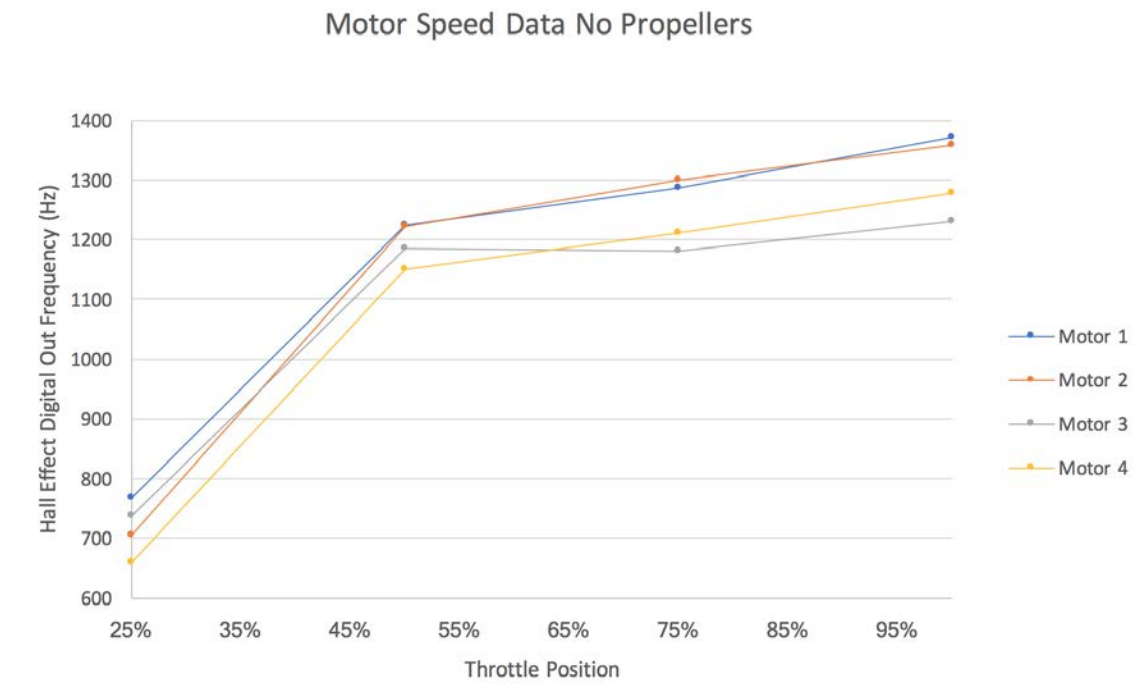

Figure 14: Motor Speed Data Plot - No Propellers

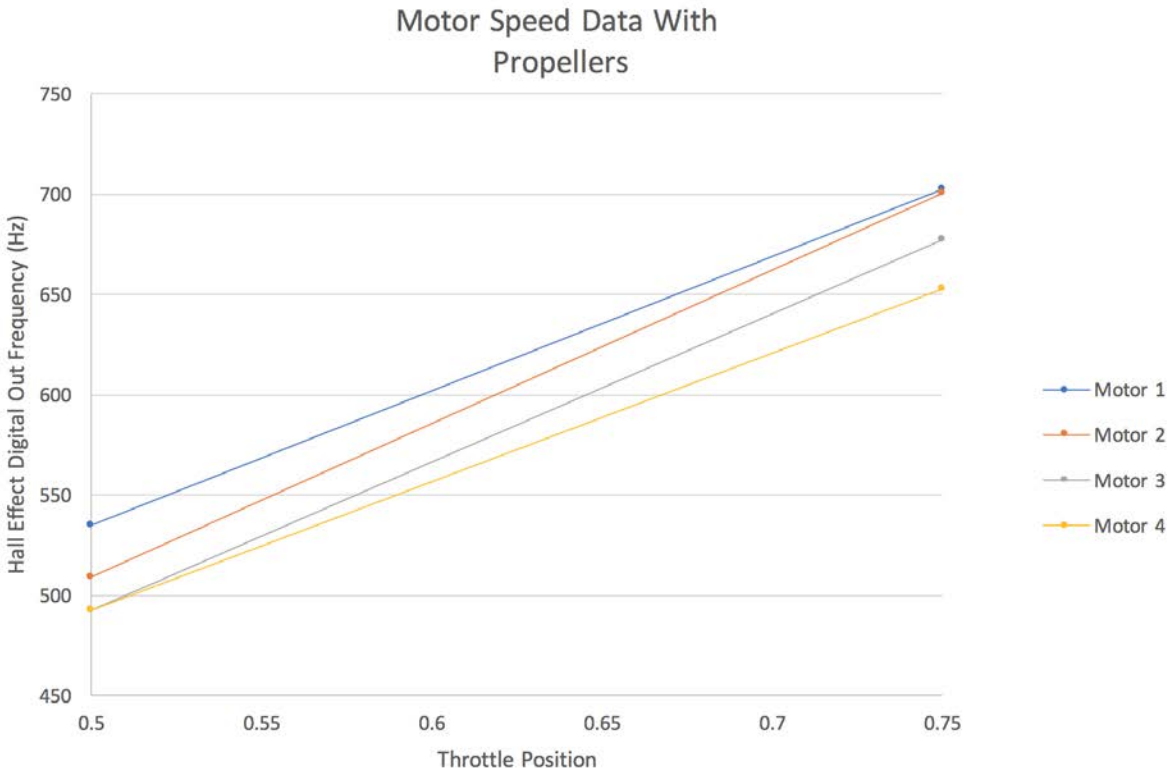

Figure 15: Motor Speed Data Plot - With Propellers

Testing Conclusions

From the hall effect sensor data and flight testing, we concluded that the difference in actual motor speeds was negligible. The differences were deemed to be small enough to where they were not the main cause of the severe instability. After many attempts to tweak the Verilog offset values and long flight tests, it was determined that the instability in flight was caused by many factors. There were potential hardware sources for instability such as the rigidity of the frame and other physical aspects of the drone. Locating each source and attempting to manually compensate would be extremely difficult if not impossible. Therefore, the focus was turned to getting a safe test flight setup to be able to demo the basic flight controls. This was done by tethering the drone to the ceiling, allowing the controls to be demonstrated with low throttle values.

Project Management

Concurrent Engineering

One of the main challenges with working in a team on a project is being able to split the workload among team members effectively. This becomes a challenge especially with software development, since most members have a specific style of coding and design approach. To combat this, we focused heavily on being able to split the project into modules and parts both in software and hardware. The idea was always to have each part in a way that a team member could work on the part or module alone, while understanding how their part relates to and connects with the part of another member. Therefore, when ready, the modules and parts could be combined and work together seamlessly without each member being heavily invested in each part. This a model that we followed throughout the various stages of the project in order to effectively utilize the abilities of each member in the given time frame.

Communication

Another important aspect of the project management was effective communication. Remote communication was especially important in a school project like this. Each member in our team had a different class and work schedule, in addition to being located in different parts of town. This made team meetings difficult to schedule.

Due to some of the scheduling and geographic challenges that we had, it was important to establish a reliable and quick informal communication method among the team members. We used an online group chat platform called GROUPME. For more formal communication with our sponsor and faculty advisor we primarily used email.

It was also essential that we established a file sharing system and had the ability to maintain revision control as well as have some live documentation capabilities. For our code sharing and revision control we used GitHub. To work on documents together and share basic informal documents, we utilized Google Drive.

Project Schedule

In order to manage our time and ensure that we are working on the project at a good pace it was essential that we establish a project schedule. The project schedule was created with two main focuses. The first focus was splitting the project into tasks or milestones that could be placed on a calendar as a due date or completion goal. This was important not only to see that the team is on time in completing the project, but also worked as a great motivating tool, to allocate some time each week to complete each of the smaller project tasks.

The second focus of the project schedule was assigning separate tasks to each member of the team. This put accountability on each of the project members to complete the task that was required of them before each due date.

Leadership

Our project management structure allowed for each member of the team to gain experience in leadership. Each half of a quarter the team would switch off who would be in the role of the project leader. The leader would be in charge of scheduling meetings, ensuring that deadlines are being met, communication with the advisor and sponsors, and making sure that the project is going as expected overall.

This was important experience to have for each of the members. It helped both the team leader in gaining experience, as well as the team members who were able to focus more on the project itself and not the project management aspect.

Conclusion

Project Deliverables

The deliverables for this project are the entire drone system. This includes the drone itself and the flight controller system developed on the FPGA. The drone includes the entire setup with the lattice FPGA development board inside of it, connected to the ESCs which control all four motors, and the transceiver which gets signals from the remote. The remote controller is also part of the drone deliverables as this is how the drone receives commands to output PWM signals to the motors. The drone flight controller is in the form of Verilog (.v) source files that are compiled and synthesized onto the FPGA.

Figure 16: Radio Controller.

Figure 17: Quadcopter.

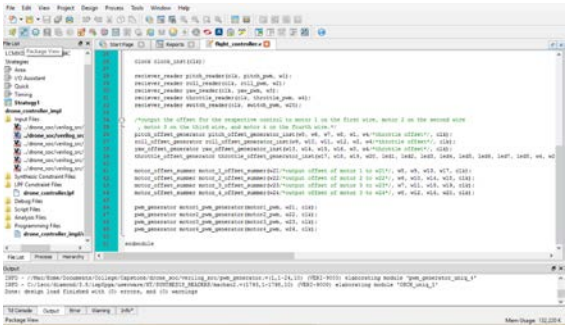

Figure 18: Verilog source files

Summary

Overall this is a successful project. We successfully developed a drone that is controlled by a lattice FPGA. The drone doesn't 'hover' reliably, but this is due to the constraints of the project and resources, not necessarily our design. The project meets the specifications outlined in the project proposal and verified by our project sponsor. The drone is controlled by the remote controller which communicates with the FPGA and outputs to the ESCs that control the motors. Operation of the drone clearly demonstrates the capabilities of the flight controller and how the remote controller accurately turns the drone in the desired direction, even though it lacks the feedback to make it hover. This is a successful basic design for an FPGA drone, but as elaborated in the following "improvements" section of the report, adding feedback from physical and geospatial sensors such as an accelerometer or a gyroscope would make it possible to create a more automated flight control that can allow the drone to hover reliably.

Improvements

Obviously, the single most important improvement that needs to be made in order to achieve stable flight with this platform is to purchase a two or three axis accelerometer and/or gyroscope and use the feedback signals from that sensor to perform real-time stabilization for the throttle of the four motors. This would be achievable as a two or three-month project using the existing Verilog modules, provided that the gyroscope or accelerometer is accurate enough and has the proper output type for our design.

If we were to continue this project ourselves, the Analog Devices ADXL212 two-axis accelerometer with PWM output (1) would be our first choice, because the output of the sensor is already PWM, which could be sensed on the inputs of the FPGA by our existing PWM_READER module with only minor changes to code. The ADXL212 is also available as an evaluation board (UG-315) which would make the connections between the FPGA and sensor exceedingly simple (2). Another possibility would be to acquire a device with SPI or I2C digital communication protocols and use an IP package in Lattice Diamond to parse the data from inputs. Personally, we would attempt the former as it would work far better with our current flight controller design.

The next big improvement to the project would be to either find a method for manually calibrating the existing ESCs, or purchase and install a set of ESCs which have good supporting documentation so that the motors can be calibrated by the FPGA as part of the initialization of the flight control module. There is simply too much uncertainty at present with regards to the motor control signal path, and the replacement of the ESCs with a calibratable variety with robust and thorough supporting documentation would be a huge step in the right direction with regards to stabilizing the drone.

There is also some concern with regards to exactly how the existing ESCs buffer the varying PWM signals on their inputs. We are certain from our flight and motor output signal characterization testing that the inputs of the ESC have some sort of buffer which stores and averages recent values, probably in an attempt to protect against large and possibly accidental changes in control signal. This is an excellent feature for an auto-stabilized drone, but a very poor feature for a manually-controlled drone. Furthermore, this does not seem to be a feature on all (or even most) ESCs, so if replacement ESCs are desired for future use in the drone project, we would strongly suggest obtaining the variety which do not have a buffer on the inputs and simply pass the motor control value directly to the motors.

References

(1) ADXL212 Datasheet*, Analog Devices*, WEB.
http://www.analog.com/media/en/technical-documentation/data-sheets/ADXL212.pdf

(2) UG-315 Evaluation Board Datasheet*, Analog Devices*, WEB.
http://www.analog.com/media/en/technical-documentation/user-guides/UG-315.pdf