ECE 544 Theory of Operation : Nexy4 DDR Quadcopter

Francisco Lopez Garibay, Jaisil Muttakulath Joy
Spoorthi Chandra Kanchi , Shadman Samin

March 22, 2017

1 Introduction

Quadcopters have become increasingly popular nowadays for its beneficial uses in various areas such as photography for commercial and personal applications, agriculture e.t.c. It has even found a great demand for personal purposes too. However this system as a whole is very complex in itself to be controlled as it is inherently unstable. To keep the 4 rotors of a quadcopter spinning at a right speed in order to keep the quadcopter levelled is difficult. Even the commercial quadcopters that are in market today are hard to be configured. This project aims to use simple, intelligible system and on-board sensors as a platform to design and configure estimators and controllers of the quadcopter, as well as allowing a scope to more advanced control in the future. The quadcopter is controlled by using a simple android application which enables us to increase the height of the quadcopter and change the direction of its flight. While in flight the quadcopter experiences many perturbations caused by its own instability issues and various external factors, the presented project is designed to deal with fundamental PID based control system. The project uses on-board accelerometer to extract roll, pitch and yaw angles which depict the current position of the quadcopter. These values are verified against the given angular values to calculate the error which is to be rectified during the flight, thus making it stable. This approach to fine tune the quadcopter is very difficult but yet we have achieved to make the system eligible for a decent flight with considerable design characteristics within the given time frame to complete the project. The objective of the project is to build, model, design and control a quadcopter platform, which needs the following:

1. We must obtain fast and accurate angular estimates along pitch, roll and yaw angular directions
2. We must implement a stable PID control system for the quadcopter by taking accurate pitch, roll, yaw angular measurements using accelerometer.
3. We must integrate an user interface for sending simple commands, that communicates wirelessly through bluetooth with the quadcopter.

2 Demo Links

Here is a link with the control system Demo:

- Control System Demo: https://www.youtube.com/watch?v=FSTehIebiGc

3 Functionality

: Here are a list of functionality that we were able to achieve and not able to achieve.

- **Calibration** - We were able to achieve remote calibration using our bluetooth controller. So that we can perfectly calibrate the motors if the quadcopter loses calibration.
- **Bluetooth Communication** - We ended up using android communcation over bluetooth to control the throttle, roll and pitch of our quadcopter.
- **PWM Control** - Used existing Vivado IP library and designed a PCB to control each individual motor with independent PWM signal.
- **Pitch & Roll Controll** - By designing and position marking the motors, we were able to come with an algorithm the provides the pitch and roll balances for the motors. And causes the motors to go forward, backward, left and right.
- **Control System** - The control system was designed using feedback from the onboard accelerometers. The accelerometers provide X,Y & Z accelerations. We calculated the pitch & roll using the raw accelerometer data to provide for Calculated Roll & Pitch. The control system was designed using a PD (Proportional & Differential) Control System.
- **Voyage Flight** - We tried implementing the design for the Voyage flight. But because we ran out of time and were not able to tune the control system. Therefore, the quadcopter does not have a stable flight.

4 Block Diagram

We started the design using the existing hardware from Project 2. We wanted to build on the project and make changes and test one thing at a time. For the initial stage of our design process, we broke the design into modular parts. We added one design at a time so that it was easier to verify functionality before we moved onto the next part of the design.

The core components of the design include the hardware platform set up in NEXYS4 DDR, the quadcopter controller set up as an android application, and the PID control algorithm implemented as an application software which runs on the NEXYS4 hardware platform.

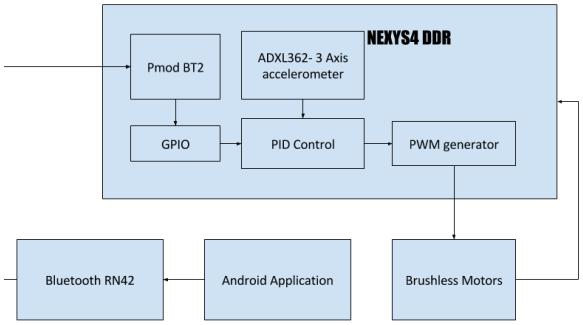


Figure 1: System Level Block diagram

The embedded block diagram represents our final design. It represents how we communication signals via the nexy4 board into our microblaze microcontroller. The following

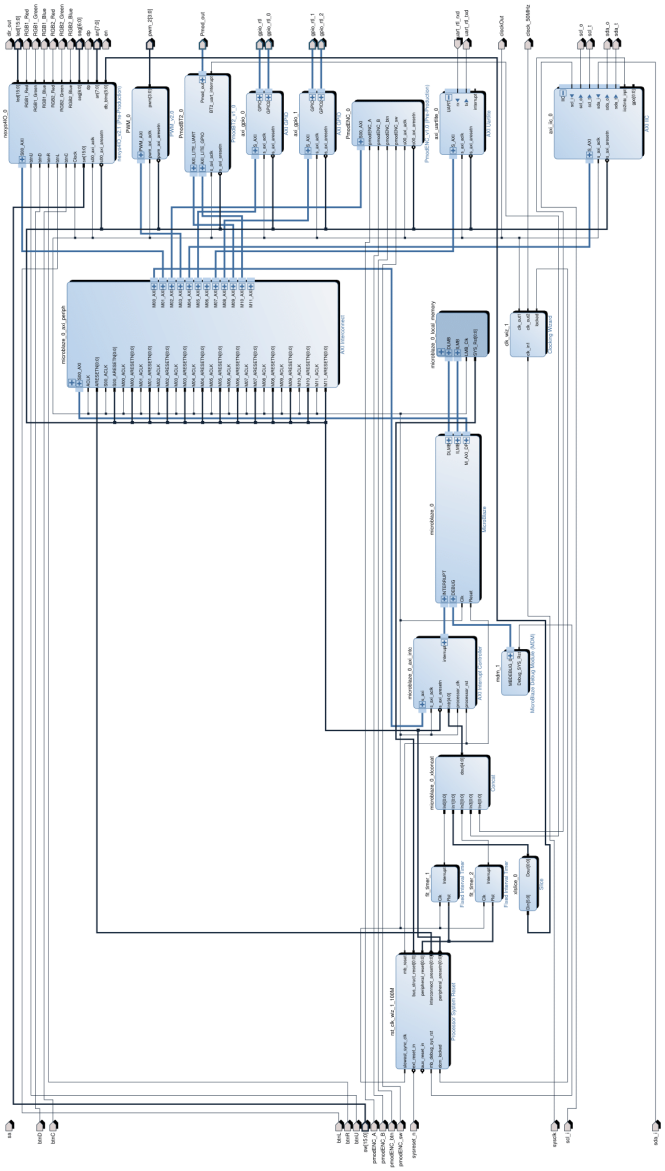


Figure 2: Embsys Block Diagram

5 Design Implementation

5.1 Hardware Platform

The essential components of the embedded system are explained in the following subsections below, there are the following modules that handle various functions in the quadcopter controller. They are as follows.

- **PMODBT2 IP:** This module is responsible for decoding the received signal from the RN42 hardware unit. The Bluetooth communication follows UART protocol. The IP is capable of generating interrupt when there is an incoming bluetooth signal. Therefore, in the software side, the data reception can be easily done in either polled mode or interrupt mode. According to the design, the signal received through bluetooth is read with the help of fit interrupt. The processed bluetooth signal from the PMODBT2 IP is read at every 5 milliseconds.

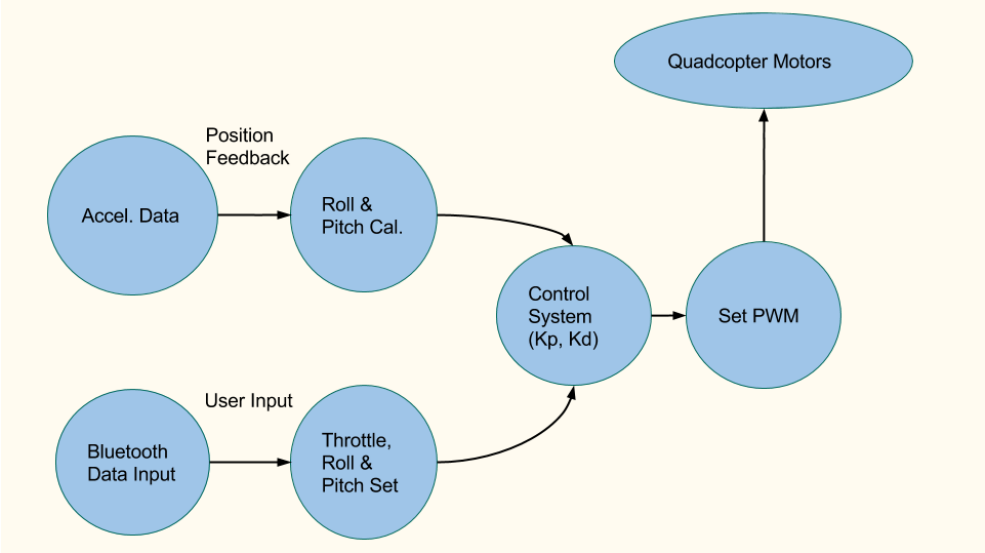
- **ADXL362ctrl:** The data from the accelerometer is read at regular intervals with the help of this module, which is primarily provided by digilent. The accelerometer can provide the acceleration in three axes. The module is written in VHDL and has parameters that can be configured for our application. The data is read from the accelerometer at every 1 millisecond, and each acceleration component has a resolution of 12 bits. Since the acceleration can take negative values, the data given by this module is in 2s complement format, and should be processed accordingly to get the correct acceleration values.

- **GPIO modules:** Two GPIO modules are used to send the three acceleration components read from the ADXL362ctrl module. Each port of the GPIO modules is configured to be a 12 bit input port in order to carry the acceleration data in X,Y, and Z axes.

- **PWM generators:** The PWM generator module controls the speed of four brushless DC motors by controlling the duty cycle of the PWM signal. The driver of this PWM generator IP gives provision to set independent duty cycles to the four motors. The duty cycle to be given to these motors are controlled by the control system. The given duty cycle is proportional to the speed of the respective motor, therefore, it depends on the control signal received by the android remote controller. In order to throttle the quadcopter up, the speed of four motors has to be increased. The roll and pitch control is achieved by increasing/decreasing the speed of two adjacent motors. There is one more control possible for a quadcopter, which is the yaw movement. The yaw movement of a quadcopter is the rotation on a plane parallel to the ground (or orthogonal to the direction of gravity). This is achieved by increasing the speed of diagonal motors.Since a pair of diagonal motors is running in clockwise direction and the other pair is running in anticlockwise direction, increasing the speed of clockwise pair will make the quadcopter rotate in clockwise direction. The yaw control is not implemented since it requires interfacing of an additional magnetometer.

5.2 Design Implementation: SDK

The following diagram illustrates the flow diagram in our SDK. Each part is explained in details and code snippets are provided in the following parts of this section. The functionality of the flow diagram of the SDK are, initially the user set data is received and parsed from the bluetooth UART. Once parsed, we calculated the pitch and roll values using the X,Y and Z accelerometer data. Once that is received the set and calculated values are fed into the control system. The control system then sets the required PWM needed to have a controlled

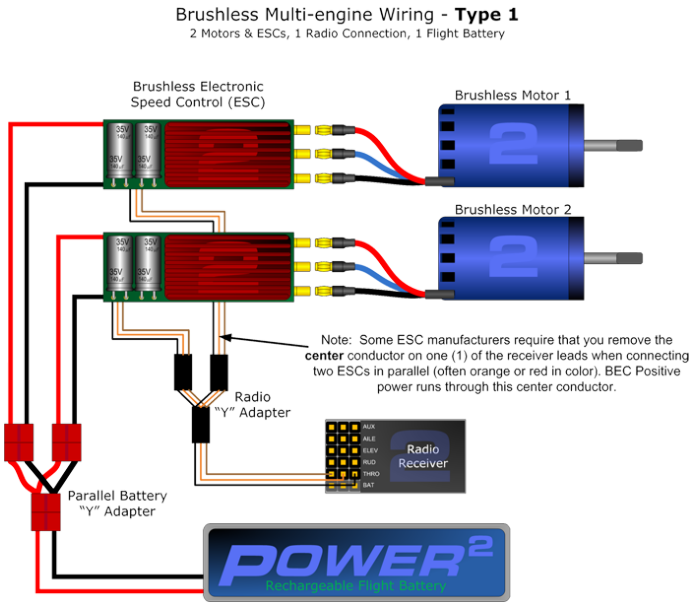


5.3 Brushless Motors & ESC

For the design of our project it was crucial to have the correct parts. For the initial testing we used the ESC(Electronic Controllers) which generates a three phase PWM controller using the PWM Signal from the Nexys4 board. Initially we tested the motors using a function generator. The initial testing was done to get the proper PWM frequency and the duty cycle needed to run the motors. We used the PWM ranges needed for the ESC to map it onto our SDK and bluetooth.

Parts used for the design:

- RAYCorp® 1045 10x4.5 Propellers
- ARRIS Swift Series BLHeli 20A 2-4S BEC 5V/1A Brushless ESC
- 1000 KV Brushless Motors



5.4 Data Communication

5.4.1 Android Application

The android application is the one responsible for controlling the quadcopter, by acting as a transmitter. The android controller is made using an app called Bluetooth electronics, which enables the user to make custom controllers for different bluetooth devices. The app has a collection of UI control elements (such as switches, joysticks, buttons, sliders etc.) which can can be used to create a sophisticated control remote for any bluetooth device. The app also allows to configure the behavior of the control elements, the format of data to be sent, the rate at which the data has to be sent etc.

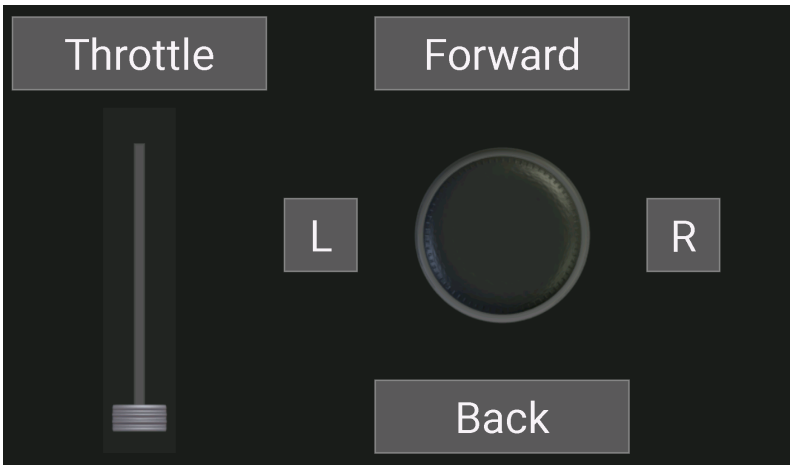


Figure 3: Android Controller

The controller for this quadcopter has a slider, which is used to control the throttle given to the quadcopter. The packets sent from this slider are enclosed in between two 'A's so that it will be easier to parse the value at the reception side. Eg: A50A. The range of the throttle value is from 0 to 100. To control the roll and pitch a joystick control is used. The joystick, which can be moved in 360 degrees on a plane, sends the X and Y coordinates of its movement. This packet is send in the format PX33Y44P, where 'P' acts like a boundary, 'X' indicates the start of roll value, and 'Y' indicates the start of pitch value.

5.4.2 Bluetooth Communcation - RN42

For bluetooth communication we used the RN42 module, it is the same module used by the digilant pmod that uses UART protocol for communication. To interface the RN42, we initially configured the rn42 using an arduino for fast prototyping. Once the arduino was configured as a slave module, 115k baud rate and the arduino was given a specific name. We started implementing the **PMOD Bt2** IP within our nexys4 block diagram. Within our SDK we wrote an algorithm that parses the individual data packets that coming in, each data packet has delimited which is used to parse the data.

```
1    // Process only if there is any data received present in the bluetooth
     buffer
     if(strlen(myDevice.recv) > 0)
3    {
       int    counter_throttle = 0;   //count variable for array that stores
     the parsed throtte values
5      int    counter_pitch = 0;      //count variable for array that stores
     the parsed pitch values
       int    counter_roll = 0;     //count variable for array that stores the
     roll pitch
7      char   Throttle[100];        //holds the parsed throttle values from
     bluetooth
```

```
  char  Pitch[100];          //holds the parsed pitch values from
bluetooth
  char  Roll[100];           //holds the parsed roll values from bluetooth
  int   flag = 0;          //flag to parse throttle value
  int   control_flag =0;       //flag to parse roll and pitch value
  int   pitch_flag=0;        //flag to parse pitch value
  int   roll_flag=0;         //flag to parse roll value
  int   set_roll_temp, set_pitch_temp,set_throttle_temp;

  for(int i=0; i<strlen(myDevice.recv); i++)
  {
    //The throttle value is sent from the bluetooth enclosed between two
'A' eg A50A
    //setting the flags to parse the throttle value
    if((flag == 0) && (myDevice.recv[i] == 'A'))
    {
      flag = 1;
    }
    else if((flag == 1) && (myDevice.recv[i] == 'A'))
    {
      //clearing the flag after the end of throttle value
      flag = 0;
    }

    //The roll and pitch values are sent from the bluetooth enclosed
between two 'P'
    //Eg: PX50Y50P
    //setting the flags to parse the roll and pitch values
    if((control_flag == 0) && (myDevice.recv[i] == 'P'))
    {
      control_flag = 1;
    }
    else if((control_flag == 1) && (myDevice.recv[i] == 'P'))
    {
      //clearing the flag at the end of after detecting final 'P'
      control_flag = 0;
    }

    //setting the flag to parse pitch
    //pitch value starts with an 'X' , eg: X50
    if(control_flag ==1 && pitch_flag ==0 && (myDevice.recv[i] == 'X') )
    {
      pitch_flag = 1;
    }
    else if(control_flag ==1 && pitch_flag ==1  &&  (myDevice.recv[i] ==
'Y'))
    {
      pitch_flag = 0;
    }

    //setting the flag to parse roll
    //pitch value starts with an 'Y' , eg: Y50
    if(control_flag ==1 && roll_flag==0 && (myDevice.recv[i] == 'Y') )
    {
      roll_flag = 1;
    }
    else if(control_flag ==0 && roll_flag==1 && (myDevice.recv[i] == 'P')
)
    {
      roll_flag = 0;
    }

    //parsing throttle as long as the throttle flag is set
    if((flag == 1) && (myDevice.recv[i] != 'A')){
      Throttle[counter_throttle] = myDevice.recv[i];
      Throttle[counter_throttle+1] = '\0';
      Throttle[counter_throttle+2] = '\0';
      counter_throttle++;
    }

    //parsing pitch as long as the pitch flag is set
    if((pitch_flag == 1) && (myDevice.recv[i] != 'X')){
      Pitch[counter_pitch] = myDevice.recv[i];
      Pitch[counter_pitch+1] = '\0';
      Pitch[counter_pitch+2] = '\0';
      counter_pitch++;
    }

    //parsing roll as long as the roll flag is set
    if((roll_flag == 1) && (myDevice.recv[i] != 'Y')){
      Roll[counter_roll] = myDevice.recv[i];
      Roll[counter_roll+1] = '\0';
      Roll[counter_roll+2] = '\0';
      counter_roll++;
```

```
87          }
        }
89
        //converting parsed values to integers
91      set_throttle_temp = char2int(Throttle, 3);
        set_pitch_temp = char2int(Pitch, 3)-30;
93      set_roll_temp = char2int(Roll, 3)-30;

95      //Normalizing the values
        if(roll_flag == 0 && pitch_flag ==0 && flag== 0 && control_flag==0)
97      {
          set_throttle = set_throttle_temp > 100? set_throttle:
      set_throttle_temp;
99        set_pitch = set_pitch_temp;
          set_roll = set_roll_temp;
101     }

103
}
```

Listing 1: Bluetooth Data Communication

5.5 Control System

The core part of the design is the control system. Quadcopters are inherently aerodynamically unstable. Unless it is balanced continuously by taking feedback of its present state, a quadcopter can not have a stable flight. It should also be taken into account that the motors, even though they are identical in rating, may not run at the exact same speed when provided with with identical pwm signals. This change itself is enough to disturb the balance of a quadcopter.

The set values for the control system are from the android Bluetooth remote controller. There are three control signals coming from the android - throttle, roll and pitch. These values are read in the fit interrupt handler. The feedback signals are from the onboard accelerometer. The module that reads the accelerometer values gives the data in 2s complement format which has to be processed accordingly to obtain the correct values of acceleration. Since the accelerometer gives only the acceleration values, it has to be converted into roll and pitch angles. Once both the set values and current values for roll and pitch are available, the error signal can be generated. This error is multiplied with the corresponding Ki,Kd, and Kp values, added with the current roll and pitch values to obtain four independent control signal for the motors.

5.5.1 Feedback: Pitch & Roll Calculation

```
1       //reading the X,Y,Z values of acceleration from GPIO
        x = XGpio_DiscreteRead(&GPIOInst0, GPIO_0_INPUT_0_CHANNEL);
3       z = XGpio_DiscreteRead(&GPIOInst0, GPIO_0_INPUT2_0_CHANNEL);
        y = XGpio_DiscreteRead(&GPIOInst1, GPIO_1_INPUT_0_CHANNEL);
5
        //converting acceleration which is in 2s complement format to normal form
7       x = convert_from_two_complement(x);
        y = convert_from_two_complement(y);
9       z = convert_from_two_complement(z);

11      // applying Low Pass filter on the signals
        fXg = (x) * alpha + (prev_fXg * (1.0 - alpha));
13      fYg = (y) * alpha + (prev_fYg * (1.0 - alpha));
        fZg = (z) * alpha + (prev_fZg * (1.0 - alpha));
15
        //saving the previous state for filtering operation
17      prev_fXg = fXg;
        prev_fYg = fYg;
19      prev_fZg = fZg;

21      //converting the values to proper range
        fXg = x*((2)/(pow(2,11)));
23      fYg = y*((2)/(pow(2,11)));
        fZg = z*((2)/(pow(2,11)));
25

27      //Pitch and roll Equation
```

```
    calculated_pitch = ((atan2(fYg, sqrt(fXg * fXg + fZg * fZg)) * 180.0) /
    M_PI )+1;
    calculated_roll  = normalize_angle(((atan2(-fXg, fZg)*180.0)/M_PI)-93);
```

Listing 2: Pitch & Roll Calculation

5.6 PD Controller

```
    // Proportional control for pitch
    err_pitch = set_pitch - calculated_pitch;
    p_delta_pitch = err_pitch * kp;

    // Derivative Control for pitch
    err_chg_pitch = err_pitch - err_old_pitch;
    d_delta_pitch = err_chg_pitch * kd;
    err_old_pitch=err_pitch;

    // Delta with PID Control
    delta_pitch = p_delta_pitch + d_delta_pitch;
    corrected_pitch = set_pitch + delta_pitch;

    // Proportional control for roll
    err_roll = set_roll - calculated_roll;
    p_delta_roll = err_roll * kp;

    // Derivative Control for roll
    err_chg_roll = err_roll - err_old_roll;
    d_delta_roll = err_chg_roll * kd;
    err_old_roll=err_roll;

    // Delta with PID Control
    delta_roll = p_delta_roll +  d_delta_roll;
    corrected_roll = set_roll + delta_roll;

    //compensation for throttle value, when quadcopter performs roll or pitch
     movements
    //the additional throttle required is proportional to the cosine of roll
    and pitch angle
    corrected_throttle /= cos(calculated_pitch)*cos(calculated_roll);
```

Listing 3: PD Controller

5.7 GPU 521 - Accelormeter & Gyro

We used MPU-6050 gyro and accelerometer chip from Invensense, in the breakout board designated as GY-521. The chip offers integrated gyro and accelerometer measurements sparing the need to worry about cross-axis alignment. Although it is possible to gain basic level stability of the quadcopter by using only an accelerometer (which detects the direction of gravitational pull with respect to the chip frame, in addition to movement acceleration), the accelerometer measurements tend to be quite sensitive to noise and vibration, and a gyro sensor is needed to detect rotation about the gravitational axis (yaw). A powerful mathematical quantity utilized in flight control is the quaternion, that contains information about orientation relative to a reference frame. A 3-axis quaternion (only using gyro) contains orientation info relative to an arbitrary reference, whereas 6-axis quaternion yields orientation relative to earth's gravitational field (discriminates up from down direction). Sensor Fusion is typically a series of mathematical algorithms that combine gyro and accelerometer data to yield a single 6-axis quaternion object. The algorithms can be computationally intensive, and for this reason the MPU-6050 provides on-chip Digital Motion Processor (DMP) that can be programmed with firmware to offload the mathematical computations into the sensor chip itself, freeing the host controller for other tasks. This is especially valuable in time-sensitive control applications, like flight control. The MPU6050 communicates via I2C bus with host controller. For this purpose we included Vivado's IIC controller IP module. There is a free version (IIC) and a paid version (I2C). The free version was found to contain several bugs, which delayed implementation of the project. We were able to successfully establish communication

with the sensor and obtain raw accelerometer and gyro data, which we could feed into an external algorithm to obtain sensor fusion data. We decided however, to take advantage of the DMP resources, which proved to be quite difficult to tap into. It was necessary to port libraries from a couple different sources (some C++, some C) and create functions for those libraries to interface with Microblaze (sensor.c and sensor.h, attached). These libraries are generally useful for applications using I2C bus.

6 Work Load Distribution

Responsibility	Francisco	Jaisil	Spoorthi	Shadman
Hardware Build				✓
Android		✓	✓	
Bluetooth Communication		✓	✓	
Drivers (IP Design)	✓			✓
SDK	✓			✓
Control System	✓	✓	✓	✓
Hardware Interface	✓	✓	✓	✓

7 Conclusion: results, challenges & discussion

The link provided at the beginning of this document is a good illustration of the results we are getting, also the in class demo provided for a good demonstration of our preliminary success of the design. We were able to implement whatever we promised, except we ran out of time at the end to implement a stable flight.

7.1 Motor Sensitivity & Calibration:

Due to the inherent nature of quadcopters, and each brushless motor having a slightly different sensitivity. We would could not quite get to a point where each motor was rotating perfectly in sync, which essentially made flight control very difficult. We continuously had to calibrate it, in order to make sure all the motors were relatively in sync so that the quadcopter did not take off in a certain direction.

7.2 Bluetooth & Android: Data Stream

Bluetooth communication was not that difficult to setup, however the packet was not stable, and it was also causing the buffer to overflow. We had to use a fit timer to slow down the system so that we read the packet much slower than what we initially planned for. In that way our data was almost stable, we would get errors, but we were able to ignore the errors in our design. Another issue was with reading the bluetooth data in interrupt mode. The interrupt provides three handlers - receive, send, and time_out. The issue was, when trying to read the data in interrupt mode, the control always goes to time_out handler, which made reading the data super slow.

7.3 Control Algorithm.

Not having a direction feedback from the motors and also the accelerometer not being the perfect sensor to provide for all the degrees of freedom. We could not tune the

control system perfectly.

7.4 Gyroscope - GPU 521

One of the bugs found in Vivado IIC controller was the inability to read a single byte of data. The problem was hard to find but easy to fix, simply read two bytes and discard one of them. Apparently this is a known bug but it's been around for years. At this point the implementation of the DMP would seem straightforward, but it turns out the documentation by invensense is even more buggy and I could not succeed at reading the FIFO buffer, even when following code available online for Arduino board. Attached you will find several files that were adapted from other platforms, none of which ultimately resulted in successful communication. Invensense was not very helpful when approached about the issue either.