MP-4

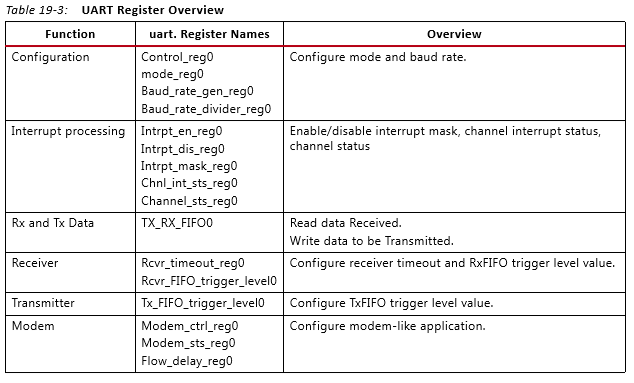
UAV Control

# UART 0

## The simple things

We changed the ucf and the mhs to route the signals to the PMOD.

# Setting up UART 0

This table comes from ug585-Zynq-7000-TRM.pdf. This is all of the registers related to the UART subsystem.

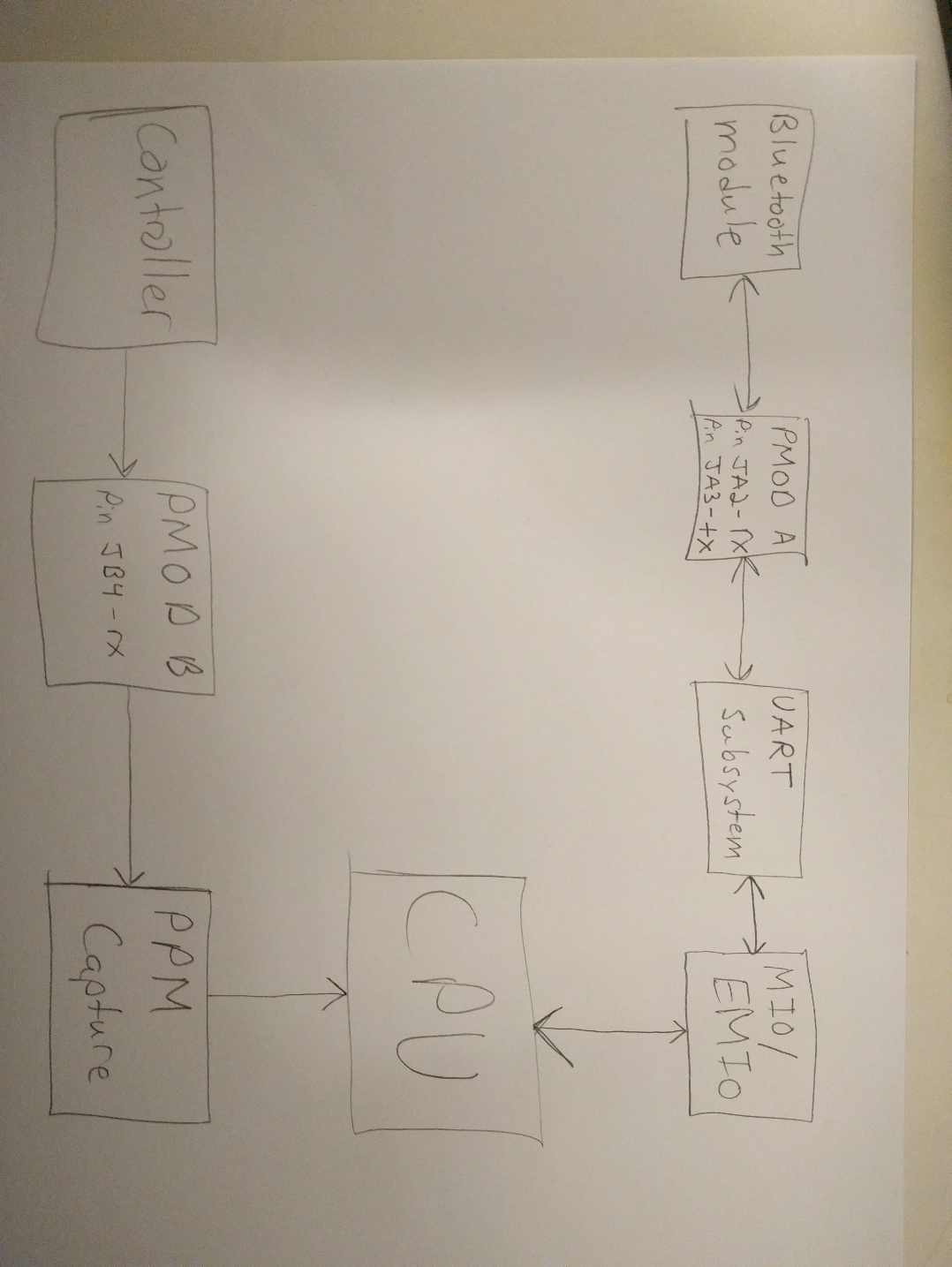
These are the relevant ones:

* Control\_reg0 – controls TX and RX (enables, resets, etc.).
* Mode\_reg0 – set up format of the data (data bits, parity, stop bits, etc.)
* Baud rate regs – Sets the baud rate
* TX\_RX\_FIFI0 – This is where the data is stored upon sending/receiving.
* Modem\_ctrl\_reg0 – sets the modem to automatic mode.

To set all of these registers, we used these functions: XUartPs\_LookupConfig() and XUartPs\_CfgInitialize(). The default config was exactly what we needed so we just used that. We explicitly called XUartPs\_SetBaudRate() to be redundant in setting the baud rate though.

# Command IDs

|  |  |
| --- | --- |
| **Command** | **ID** |
| Request IMU Data | 102 |
| Orientation Information | 108 |
| Send roll, pitch, yaw, and throttle | 200 |



# High Level Diagram

# Data Analysis

|  |  |  |
| --- | --- | --- |
| Pitch | | |
| Orientation (degrees) | Accelerometer (degrees) | Gyroscope (degrees) |
| -45 | -39.8 | -13.4 |
| 0 | -0.4 | -0.3 |
| 45 | 41.3 | 7.0 |

|  |  |  |
| --- | --- | --- |
| Roll | | |
| Orientation (degrees) | Accelerometer (degrees) | Gyroscope (degrees) |
| -45 | -43.7 | -14.7 |
| 0 | -0.2 | -3.8 |
| 45 | 44.0 | 7.0 |

Accelerometer

Roll and pitch are obtained by scaling the IMU data to be between -1 and 1 and then taking the arcsine of that value.

Our accelerometer readings matched pretty well. They always came up short but they were a lot closer than the gyroscope readings. It was not necessary to know the units of acceleration. We assumed they were in Gs but that doesn’t really help us.

Shaking the accelerometer caused a lot of noise in our data. The data would fluctuate between -5 to 5 degrees.

Gyroscope

We used two pieces of data to calculate the angle using the gyroscope. The first piece is the time between data points, which we got from the global timer counter on the board. The second piece is the value from the gyroscope. We divided the gyroscopes reading by the Scale Factor found in the datasheet to get degrees per second. Lastly we multiplied our time by degrees per second to get just degrees and then kept a running total of that.

Our gyroscope data was off by a lot. Our range seemed to be from -15 to 7 degrees when it should have been -45 to 45 degrees. We noticed that we were dividing our time difference by COUNTS\_PER\_SECOND which was defined in XTime\_l.h. TA Joe said that he used XPAR\_CPU\_CORTEXA9\_CORE\_CLOCK\_FREQ\_HZ which was twice as much. That might expand our range.

Shaking the gyroscope didn’t really affect our data at all. The data stayed within a degree of its value.

# C:\cpre488mp4\images\gps_with_debug_message.PNGPseudo-GPS

# PID Controller

Not sure how this write-up step is different from the next one.

When we started testing, we followed Dr. Jones’ example from class. We started by setting I and D to zero and only messing with P. Once we got a P value that caused the quad to respond in a way that seemed quick enough, we added a D term to get rid of the oscillation. Lastly, we added a small I term to help account for friction and other static forces in the system.

# Flight Test

Not sure how this write-up step is different from the previous one.

We tested to see if our constants for P, I, and D make the quad respond in an acceptable fashion.

We did not try flying the quad off of the ground, but when we started the quad at an offset it did orient itself to 0 degrees yaw.