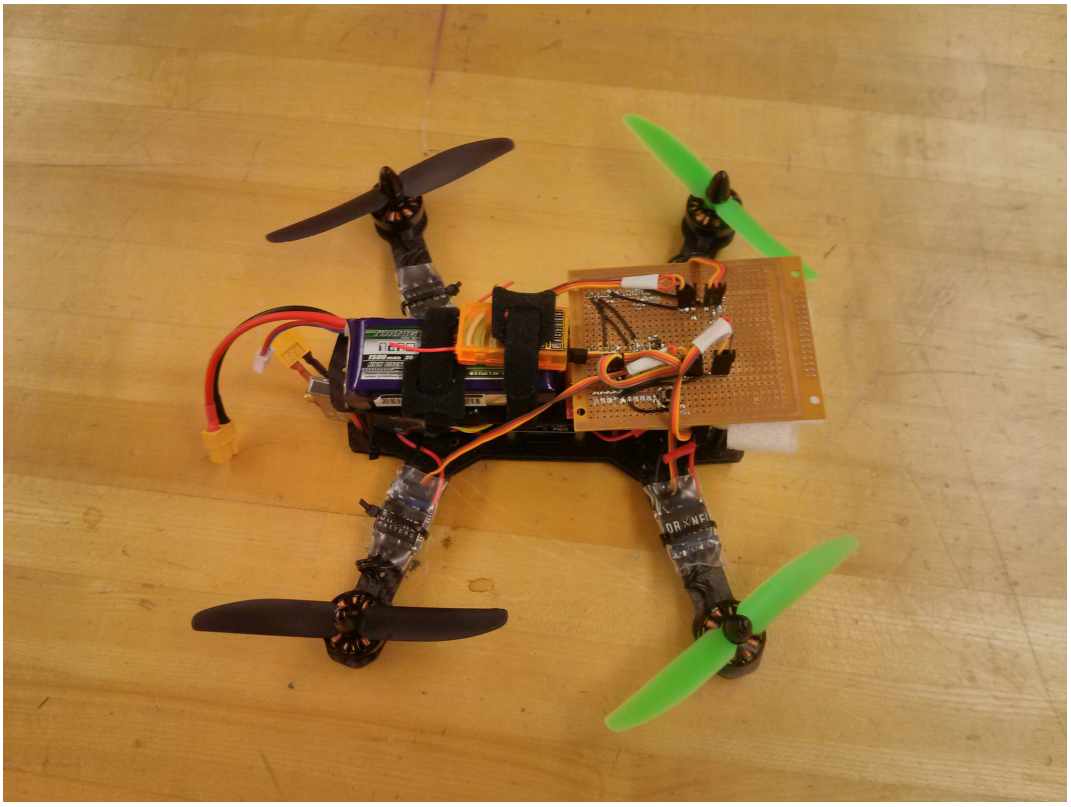
Embedded Systems - ENGR 844

TI LaunchPad Flight Controller

Kristian Sloth Lauszus (915613834)
lauszus@gmail.com
San Francisco State University



1 Abstract

This report describes the development of a flight controller for a quadcopter in x-configuration using the TI Tiva EK-TM4C123GXL Launchpad development board. The quadcopter is using a MPU-6500 which is a 3-axis accelerometer and 3-axis gyroscope. The gyroscope data is used as input to three PID controllers that stabilize the quadcopter in each axis.

The main focus of this report is on the firmware aspect of the design. The entire firmware is written in C and is available under the GPLv2 license at: `https://github.com/Lauszus/LaunchPadFlightController`.

2 Introduction

It seems like not a day goes by without so called drones are brought up in the news, but most people have no idea what it really is and what actually makes them fly. When people hear the word drones they often think about the huge planes the US military use in there different conflicts, but it can also be small autonomous multirotors that have anything from typically 3-8 rotors. These types of aircrafts require a small on-board computer to stabilize them or they would simply not be able to fly. This is often implemented on a microcontroller taking input from a variety of sensors like accelerometers, gyroscopes, gps, barometers, radar, ultrasonic etc. This data is then used to stabilize the multirotor and can also be used to make it fully autonomous.

In this report I will describe how I implemented a flight controller on a Tiva EK-TM4C123GXL Launchpad featuring the TM4C123GH6PM 32-bit ARM Cortex-M4 microcontroller running at 80 MHz. It will only take input from a 3-axis gyroscope, thus it will not be able to fly on its own, but this could be implemented in the future.

The reason why I want work on this came after I built several self balancing robots[1,2]. These are actually not that different from how a quadcopter is stabilized, as both take input from an IMU (Inertial Measurement Unit) and then uses this as an input to a PID controller which output is then used to drive motors that makes the system able to function.

3 IMU

One of the key aspects of the flight controller is the IMU. I decided to use the MPU-6500[3] which is a 3-axis accelerometer and 3-axis gyroscope. In this case I only used the gyroscope. This allows the quadcopter to stabilize itself, but it will not self-level i.e. go back to horizontal when the user let go of the sticks. This gyro only mode is often referred to as acro mode, as it allows the user to do acrobatic maneuvers.

The communication with the MPU-6500 is done via I2C running at 400 kHz. New measurements are taking every 1 ms. The gyroscopes range can be configured by writing to special registers in the MPU-6500, thus I set it to ± 2000 deg/s for the gyroscope. Furthermore a DLPF (Digital Low-pass Filter) in the MPU-6500 is used to smooth out the values from the gyroscope. After the data is read a offset is then subtracted which is found by reading the gyroscope at start-up, thus it is important to keep the quadcopter still while powering it on, so the calibration value does not get affected. The code for communicating with the MPU-6050 can be found in `MPU6500.c` in the appendix.

The gyroscope data for all three axes i.e. roll, pitch and yaw are then used as a input to three PID controllers. The PID outputs are then sent to the motors as shown in the code

[1] `http://balanduino.net/`
[2] `http://blog.tkjelectronics.dk/2014/07/full-size-diy-balancing-robot/`
[3] `http://www.invensense.com/mems/gyro/mpu6500.html`

snippet below. Where motor 0 is the bottom right one, motor 1 is the top right, motor 2 is the bottom left and motor 3 is the top left one.

Thus if it senses that the quadcopter is starting to roll clockwise it will speed up the two right motors and slow down the other two. In the same way it would increase the speed of the two front motors and slow the two back motor down if the quadcopters pitches down.

Furthermore it also compensate for yaw. This is done by the fact that the bottom right and top left both spin CW while the other two spins CCW, thus by applying more speed to any pair will make the robot rotate along its yaw axis.

```
166 motors[0] -= rollOut;
167 motors[1] -= rollOut;
168 motors[2] += rollOut;
169 motors[3] += rollOut;
170
171 motors[0] += pitchOut;
172 motors[1] -= pitchOut;
173 motors[2] += pitchOut;
174 motors[3] -= pitchOut;
175
176 motors[0] -= yawOut;
177 motors[1] += yawOut;
178 motors[2] += yawOut;
179 motors[3] -= yawOut;
```

4 Receiver

In order to take input from the operator a receiver operating at 2.4 GHz is used[4]. The receiver sends out a value for each channel using CPPM (Coherent Pulse Position Modulation), as shown in Fig. 1. The width of each pulse represents the input from each channel sent from the transmitter, so by measuring the width of each pulse one can determine the channel value.

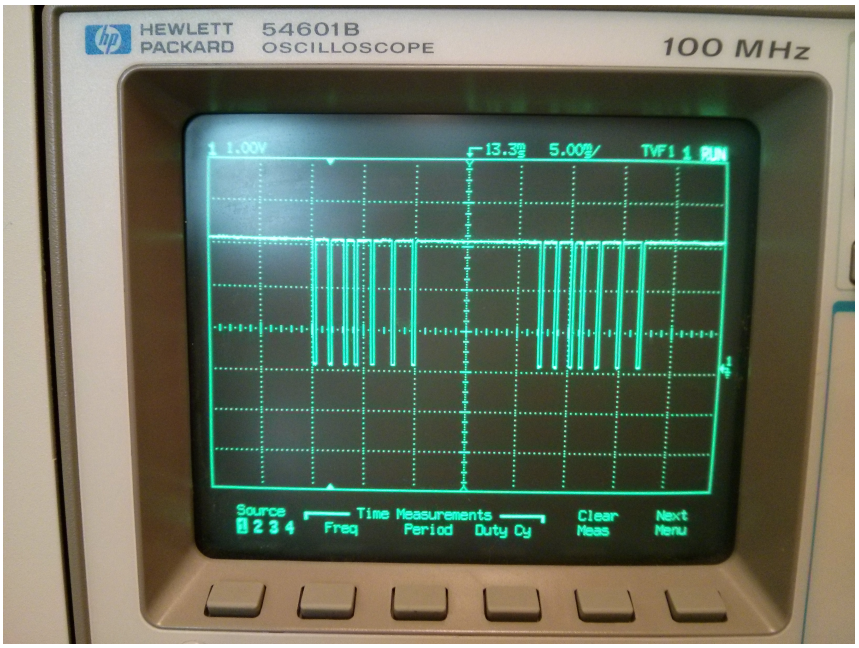


Fig. 1: CPPM output sent from the receiver

[4] http://www.hobbyking.com/hobbyking/store/__46632__OrangeRx_R615X_DSM2_DSMX_Compatible_6Ch_2_4GHz_Receiver_w_CPPM.html

The width is measured using WTimer1A in Timer Input Capture interrupt mode. It is configured to call an interrupt routine every time the edge changes. The interrupt routine then simply reads the value and check if it just went from a high to low signal. If this is the case it takes the current value and subtracts it from the previous value. It then converts this value to microseconds and check if it is above 2700 μs, which indicates that a entire frame was received. If that is not the case it simply stores the value in an array. The relevant code can be found in the file `RX.c` in the appendix.

The throttle is applied directly to the motor values, as shown below.

```
162 float throttle = map(rxChannel[RX_THROTTLE_CHAN], RX_MIN_INPUT, RX_MAX_INPUT, -100.0f,
        100.0f);
163 for (uint8_t i = 0; i < 4; i++)
164     motors[i] = throttle;
```

The aileron, elevator and rudder channels are applied to the motors outputs as shown below. I divide the aileron and elevator values by two in order to make it less sensitive to these inputs.

```
181 // Roll Control
182 float aileron = map(rxChannel[RX_AILERON_CHAN], RX_MIN_INPUT, RX_MAX_INPUT, -100.0f,
        100.0f);
183 motors[0] -= aileron / 2.0f;
184 motors[1] -= aileron / 2.0f;
185 motors[2] += aileron / 2.0f;
186 motors[3] += aileron / 2.0f;
187
188 // Pitch Control
189 float elevator = map(rxChannel[RX_ELEVATOR_CHAN], RX_MIN_INPUT, RX_MAX_INPUT, -100.0f,
        100.0f);
190 motors[0] += elevator / 2.0f;
191 motors[1] -= elevator / 2.0f;
192 motors[2] += elevator / 2.0f;
193 motors[3] -= elevator / 2.0f;
194
195 // Rudder Control
196 float rudder = map(rxChannel[RX_RUDDER_CHAN], RX_MIN_INPUT, RX_MAX_INPUT, -100.0f, 100.0f
        );
197 motors[0] -= rudder;
198 motors[1] += rudder;
199 motors[2] += rudder;
200 motors[3] -= rudder;
```

5 Motors

The motors used for the quadcopter are 3-phase sensorless BLDC (Brushless DC) motors[5]. These are controlled by a dedicated ESC[6] (Electronic Speed Controller) for each motor that takes care of driving it. The speed of the motors is controlled by varying the width of a PPM (Pulse Position Modulation) signal connected to the ESC. The ESCs used in this application is running optimized firmware meant for multirotors[7].
The PPM signal is generated using dedicated PWM functionality on the microcontroller at a frequency of 400 Hz. A pulse width of 1064 μs corresponds to minimum speed while 1864 μs corresponds to full speed. The relevant code can be found in `PPM.c` in the appendix.

[5] `http://www.dronematters.com/index.php/sunnysky-x2204-2300kv-brushless-motor.html`
[6] `http://www.dronematters.com/index.php/power-system/escs/drone-matters-naked-blue-series-12a-rapidesc-simonk.html`
[7] `https://github.com/sim-/tgy/tree/2013-05-15`

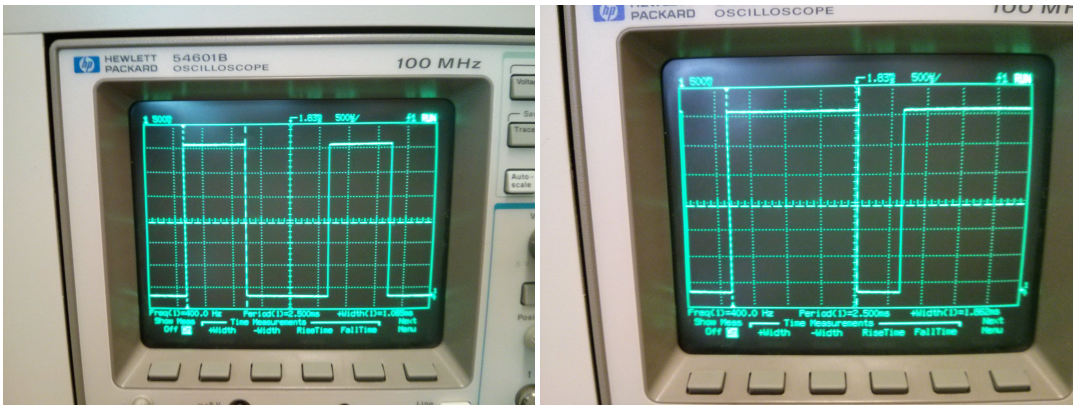


Fig. 2: Oscilloscope output when sending a pulse with 1065 μs & 1862 μs width respectively

6 Safety

It is important to take safety into account when working on a system like this, as the propellers are spinning at several thousands RPM. So they are actually able to do severe damage both to property and people. Therefore the propeller should not be mounted until one have made certain that everything is working properly. The most dangerous part is thus to tune the PID controllers, as it is necessary for the propellers to be mounted in this step.

I would recommend doing an initial test without the propellers, but with a little bit of throttle applied. The quadcopter's functionality can then be confirmed by tilting the quadcopter in all three axis an observing that it behaves as intended.

Furthermore one must also consider what happens if the connection to the transmitter is lost. In this case my receiver simply outputs the minimum values for all channels, but one must also take into account if the connection from the receiver and flight controller for some reason breaks. In this case I used WTimer1B as a periodic timer. I simply configured it to call an interrupt every 100 ms and setting a flag indicating that the connection was lost. The timer value is reset every time a full frame is received, thus the interrupt is only called if there has not been any data for 100 ms. The main loop simply checks this flag contentiously and if it is set it shuts down all motors.

I also used the onboard LEDs to indicate whenever the flight controller was armed or not. It lights up green if everything is running correctly and switches to red once the AUX1 channel is in the armed position. This gives some useful visual indication both to the operator, but also to potential spectators.

7 Code

Fig. 3 shows a flowchart for `main.c`. First the microcontroller is setup to use the external crystal and the PLL is set up, so the operating clock ends up being 80 MHz. There is no reason not to just run at full speed, as the motors use so much power compared to the microcontroller and it is also critical to have a stable clock for a design like this.

It then sets up the PPM, UART, timers used for basic timekeeping, the receiver, the MPU-6500 and the LEDs.

Next step is to set the PID values and initialize the different terms. It then checks if there is both valid RX data and that the AUX1 channel is in the unarmed position. If this is not the case it simply waits until it is true.

Now begins the main loop. First it checks if there is any incoming UART data. This is used to set the PID values via UART. The relevant code can be found in `UART.c` in the

appendix. It does not currently store the new values in the EEPROM, so one has to write them in the code manually if they want to keep them after the power is turned off.

After that it checks if the is valid data from the receiver and if the AUX1 channel is in the armed position. If this is true it turns on the red LED if not the green LED will be turned on.

It then checks if there is any new IMU data. This is done by reading a special INT pin on the MPU-6500 that is configured to go high every time there is new measurements available. The update rate is set at 1 kHz.

The final step is to check if it is both armed and it has been more than 2.5 ms since the last update to the motors. This is done because the ESCs are limited to a 400 Hz update rate, so there is no point of updating the motor output more than that. If it is the case it updates all three PID controllers and apply the aileron, elevator and rudder controls based on the values from the receiver. The output values are then outputted on the PWM pins connected to the four ESCs.

As it can be seen the microcontroller is actually sitting and waiting most of the time due to the limited update rate of the IMU and ESCs. The ESCs update rate could possible be increased by using for instance I2C to interface with them instead of a PPM signal.

I initially wanted to use SPI for the communication with the MPU-6500, but I had some problems to get it working, so I just ended up using I2C. It does not really matter anyway right now, as the microcontroller is just waiting most of the time, but it might be more critical in a more advanced flight controller, so it is something I intend to implement in the future.

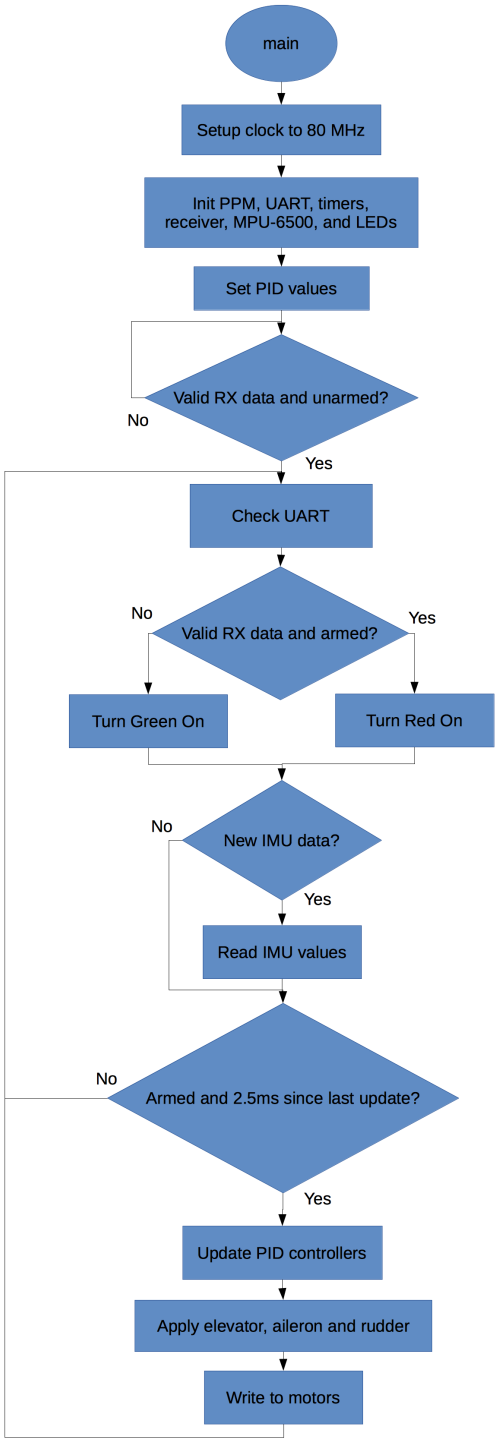


Fig. 3: Flowchart of main loop

I also use several interrupts. This include an UART interrupt which are called every time new serial data is received on the UART bus. I will not go into details with this, as this is part of the TivaWare library[8].

Fig. 4 shows the flowchart for the three interrupts routines I wrote.

I use the System Timer (SysTick) to increment a counter every 1 μs. The counter can then be read to estimate how long time it has been since the interrupt was started in μs. By subtracting the current value with a previous one, it is possible to estimate the Δtime. This is used throughout the code. For instance to limit the motor update rate to 2.5 ms. The relevant code can be found in `time.c` in the appendix.

The next interrupt handler shown in Fig. 4 is the Timer Input Capture interrupt used to measure the width of the pulses sent from the receiver. At first it clears the interrupt, then it reads the timer value. If it detect that it went from a positive to falling edge it calculates the pulse width by subtracting the current timer value with the previous one and then converts it into μs. It then checks if the value is more than 2700 μs, which indicates that a so called sync pulse is received which indicates that a full frame has been received. If this is the case it resets the index back to 0 and checks if all channels contain valid data and set the RX valid flag accordingly. It finally clears the WTimer1B timer if all the data was valid.

If a sync pulse is not received it stores the width in an array and increments the index. Finally it stores the previous edge and timer value before it returns.

The last interrupt handler is the one called by WTimer1B which is set up as a periodic timer. If no data is received from the receiver for 100 ms the timeout interrupt handler will be called. It simply clears the interrupt and clears the RX valid flag. The main loop checks for this flag contentiously and shuts off all motors if it is cleared.

The code for the CaptureHandler and TimeoutHandler can be found in `RX.c` in the appendix.

[8] `http://www.ti.com/tool/sw-tm4c`

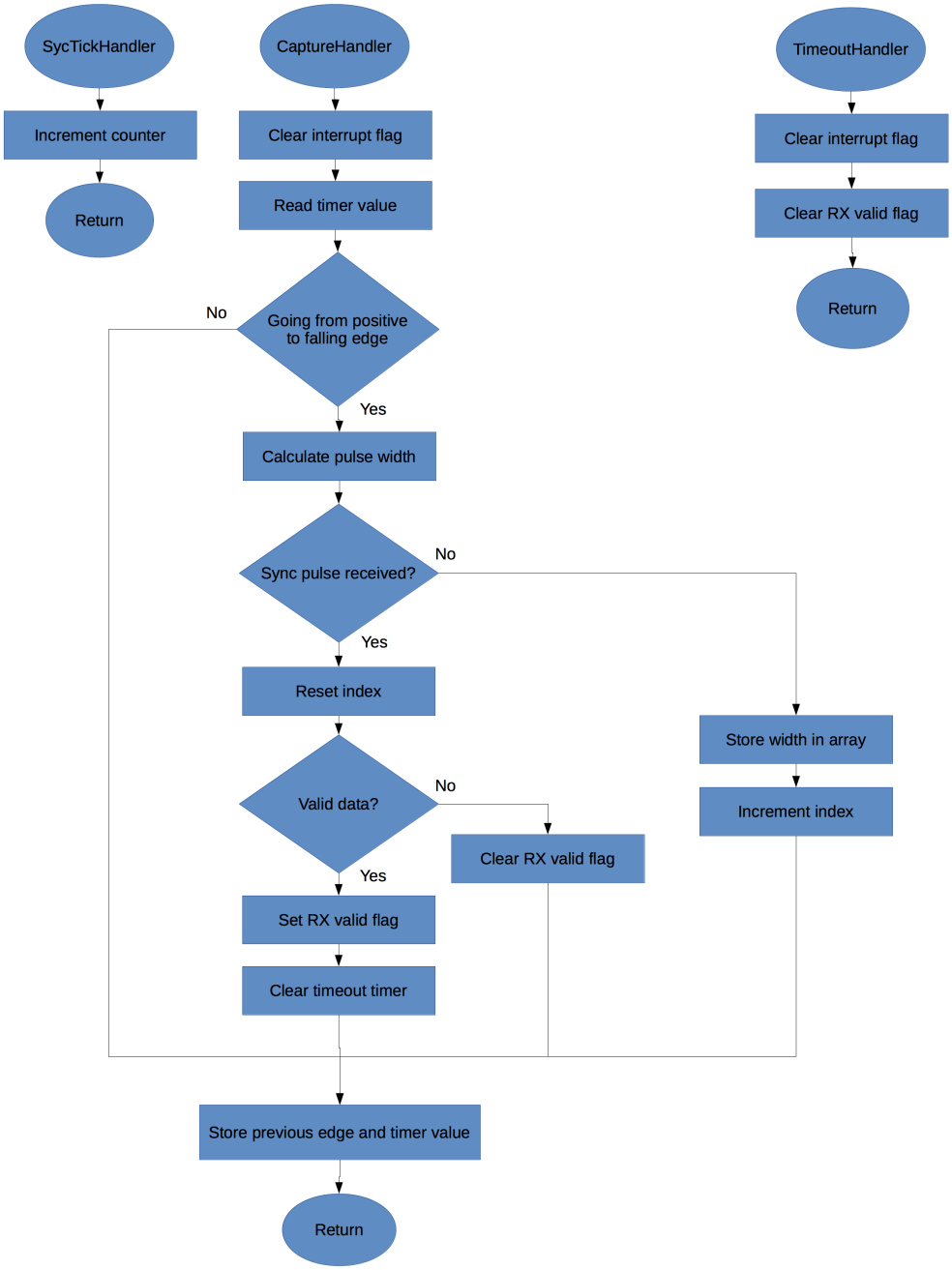


Fig. 4: Flowchart of interrupts

8 Results & Discussion

The final flight controller turned out really well. It flies really well and behaves just like a commercial available one. Though it is a bit limited in the aspect that it only supports one kind of receiver and one configuration of multiroter design i.e. quadcopter in x-configuration. The later could easily be implemented using a so called mix.

Tuning the PID controller was actually not that hard once it was implemented in the code, as I have a lot of experience with PID controllers both in regard with self balancing robots and other multiroters I have built in the past with commercial available flight controller. Right now I actually only use a PI controller and it seems to work quite well. I might try to tune Kd as well later on.

Furthermore it does not currently support self-level functionality, so it actually does not use the accelerometer right now, but this is something I will be working on in the future. Another improvement would be to make it fully autonomous.

I would also like to make some kind of GUI either as a computer application, Android application or a touchscreen display. This would allow me to set the different configuration values without having to change them in the code and then reupload it. I might also work on adjusting the PID values using the knobs on the transmitter. The new PID values will then be saved in the EEPROM on the microcontroller.

I have also thought about making a dedicated PCB with the TM4C123GH6PM, MPU-6500 and all required components on it, so it could be made smaller and more user friendly.The microcontroller cost \$11.55[9] in small quantity and the MPU-6500 breakout board cost about \$4.27 online[10]. These are by far the most expensive components, so I would expect to be able to make a custom board in small volume for less than \$30.

Some demonstration videos of the quadcopter flying can be found on my YouTube channel[11].

9 Conclusion

I achieved my initial goal which was to implement a flight controller on the TI Tiva EK-TM4C123GXL Launchpad. I limited my work to only focus on one specific hardware setup and to only get acro mode stabilization working.

I learned a great deal from this project regarding the TI Tiva EK-TM4C123GXL Launchpad and how a flight controller for a multirotor actually works. I also got more familiar with the Keil μVision IDE.

One important improvement would be to implement self-level mode which is really useful for beginners.

[9] http://www.ti.com/product/tm4c123gh6pm/samplebuy

[10] http://www.ebay.com/sch/sis.html?_kw=3+Axis+MPU-6500+Gyroscope+and+Accelerator+Sensor+Replace+MPU-6050+For+Arduino&_id=251605268116

[11] https://www.youtube.com/user/kslauszus

10 Appendix

10.1 Hardware

List of hardware components used:

- Microcontroller: Tiva C Series TM4C123G LaunchPad
- IMU: MPU-6500
- Motors: Sunnysky X2204 2300kv
- ESCs: Blue Series 12A RapidESC (SimonK v2013-05-15)
- Propellers: Gemfan 5x3
- Frame: 250 FPV Quadcopter (Bought on eBay)
- LiPo: Turnigy nano-tech 1500 mAh
- RX: OrangeRX R615X
- TX: Turnigy 9XR
- TX module: OrangeRX 2.4GHz transmitter module

10.2 Code

10.2.1 main.c

```
/* Copyright (C) 2014 Kristian Lauszus, TKJ Electronics. All rights reserved.

 This software may be distributed and modified under the terms of the GNU
 General Public License version 2 (GPL2) as published by the Free Software
 Foundation and appearing in the file GPL2.TXT included in the packaging of
 this file. Please note that GPL2 Section 2[b] requires that all works based
 on this software must also be made publicly available under the terms of
 the GPL2 ("Copyleft").

 Contact information
 -------------------

 Kristian Lauszus, TKJ Electronics
 Web      :  http://www.tkjelectronics.com
 e-mail   :  kristianl@tkjelectronics.com
*/

#include <stdint.h>
#include <stdbool.h>
#include <stdlib.h>

#include "RX.h"
#include "UART.h"
#include "time.h"
#include "PPM.h"
#include "PID.h"
#include "MPU6500.h"

#include "inc/hw_memmap.h"
#include "inc/tm4c123gh6pm.h"
#include "driverlib/gpio.h"
#include "driverlib/interrupt.h"
#include "driverlib/sysctl.h"
#include "utils/uartstdio.h" // Add "UART_BUFFERED" to preprocessor

// Only acro mode is actually working for now
#define ACRO_MODE 1

#define SYSCTL_PERIPH_LED SYSCTL_PERIPH_GPIOF
```

```
40 #define GPIO_LED_BASE      GPIO_PORTF_BASE
41 #define GPIO_RED_LED       GPIO_PIN_1
42 #define GPIO_GREEN_LED     GPIO_PIN_3
43
44 int main(void) {
45     // Set the clocking to run directly from the external crystal/oscillator.
46     SysCtlClockSet(SYSCTL_SYSDIV_2_5 | SYSCTL_USE_PLL | SYSCTL_OSC_MAIN |
           SYSCTL_XTAL_16MHZ); // Set clock to 80MHz (400MHz(PLL) / 2 / 2.5 = 80 MHz)
47
48     initPPM();
49     initUART();
50     delay(100);
51     UARTprintf("Started\n");
52     initTime();
53     initRX();
54
55     //initMPU6500();
56     initMPU6500_i2c();
57
58     SysCtlPeripheralEnable(SYSCTL_PERIPH_LED); // Enable GPIOF peripheral
59     SysCtlDelay(2); // Insert a few cycles after enabling the peripheral to allow the
           clock to be fully activated
60     GPIOPinTypeGPIOOutput(GPIO_LED_BASE, GPIO_RED_LED | GPIO_GREEN_LED); // Set red and
           blue LEDs as outputs
61
62     IntMasterEnable();
63
64     delay(100); // Wait a little for everything to initialize
65
66     UARTprintf("CLK %d\n", SysCtlClockGet());
67     UARTprintf("min: %d, max: %d, period: %d\n", PPM_MIN, PPM_MAX, getPeriod());
68
69 #if !ACRO_MODE
70     const float restAngleRoll = 1.67f, restAnglePitch = -2.55f; // TODO: Make a
           calibration routine for these values
71 #endif
72
73 #if ACRO_MODE
74     // 0.0200.0.1000.0.0000
75     pidRoll.Kp = 0.016f;
76     pidRoll.Ki = 0.050f;
77     pidRoll.Kd = 0.0f;
78 #else
79     pidRoll.Kp = 1.75f;
80     pidRoll.Ki = 1.0f;//2.3f;
81     pidRoll.Kd = 0.0f;
82 #endif
83     pidRoll.integratedError = 0.0f;
84     pidRoll.lastError = 0.0f;
85
86     pidPitch = pidRoll; // Use same PID values for both pitch and roll
87
88     // x2 the values work pretty well - TODO: Fine-tune these
89     pidYaw = pidRoll;
90     pidYaw.Kp *= 2.0f;
91     pidYaw.Ki *= 2.8f; // I increased this in order for it to stop yawing slowly
92     pidYaw.Kd *= 2.0f;
93
94     printPIDValues();
95
96 #if ACRO_MODE
97     static int16_t gyroData[3];
98 #else
99     float roll, pitch;
100 #endif
101     static uint32_t imuTimer = 0, pidTimer = 0;
102
103     // Motor 0 is bottom right, motor 1 is top right, motor 2 is bottom left and motor 3
           is top left
104     static float motors[4] = { -100.0f, -100.0f, -100.0f, -100.0f };
105
106     static bool armed = false;
107
108     while (!validRXData || rxChannel[RX_AUX1_CHAN] > 1000) {
109         // Wait until we have valid data and we are unarmed
110     }
111
112     while (1) {
```

```
113         checkUARTData();
114
115         // Make sure there is valid data, AUX channel is armed and that throttle is
                applied
116         // The throttle check can be removed if one prefer the motors to spin once it is
                armed
117         // TODO: Arm using throttle low and yaw right
118         if (!validRXData || rxChannel[RX_AUX1_CHAN] < 1000 || rxChannel[RX_THROTTLE_CHAN]
                < RX_MIN_INPUT + 25) {
119             writePPMAllOff();
120             pidRoll.integratedError = pidRoll.lastError = 0.0f;
121             pidPitch.integratedError = pidPitch.lastError = 0.0f;
122             pidYaw.integratedError = pidYaw.lastError = 0.0f;
123             armed = false;
124         } else
125             armed = true;
126
127         GPIOPinWrite(GPIO_PORTF_BASE, GPIO_RED_LED | GPIO_GREEN_LED, !validRXData ||
                rxChannel[RX_AUX1_CHAN] < 1000 ? GPIO_GREEN_LED : GPIO_RED_LED); // Turn on
                red led if there is valid data and AUX channel is in armed position otherwise
                 turn on green LED
128
129         if (dataReadyMPU6500()) {
130             float dt = (float)(micros() - imuTimer) / 1000000.0f;
131             //UARTprintf("%d\n", micros() - imuTimer);
132             imuTimer = micros();
133
134 #if ACRO_MODE
135             getMPU6500Gyro(gyroData);
136
137             /*UARTprintf("%d\t%d\t%d\n", gyroData[0], gyroData[1], gyroData[2]);
138             UARTFlushTx(false);*/
139 #else
140             getMPU6500Angles(&roll, &pitch, dt);
141
142             /*UARTprintf("%d.%02d\t%d.%02d\n", (int16_t)roll, (int16_t)abs(roll * 100.0f)
                    % 100, (int16_t)pitch, (int16_t)abs(pitch * 100.0f) % 100);
143             UARTFlushTx(false);*/
144 #endif
145         }
146
147         float dt = (float)(micros() - pidTimer);
148         if (armed && dt > 2500) { // Limit to 2.5ms (400 Hz)
149             //UARTprintf("%d\n", micros() - pidTimer);
150             pidTimer = micros();
151             dt /= 1000000.0f; // Convert to seconds
152
153 #if ACRO_MODE
154             float rollOut = updatePID(&pidRoll, 0, gyroData[1], dt);
155             float pitchOut = updatePID(&pidPitch, 0, gyroData[0], dt);
156 #else
157             float rollOut = updatePID(&pidRoll, restAngleRoll, roll, dt);
158             float pitchOut = updatePID(&pidPitch, restAnglePitch, pitch, dt);
159 #endif
160             float yawOut = updatePID(&pidYaw, 0, gyroData[2], dt);
161
162             float throttle = map(rxChannel[RX_THROTTLE_CHAN], RX_MIN_INPUT, RX_MAX_INPUT,
                    -100.0f, 100.0f);
163             for (uint8_t i = 0; i < 4; i++)
164                 motors[i] = throttle;
165
166             motors[0] -= rollOut;
167             motors[1] -= rollOut;
168             motors[2] += rollOut;
169             motors[3] += rollOut;
170
171             motors[0] += pitchOut;
172             motors[1] -= pitchOut;
173             motors[2] += pitchOut;
174             motors[3] -= pitchOut;
175
176             motors[0] -= yawOut;
177             motors[1] += yawOut;
178             motors[2] += yawOut;
179             motors[3] -= yawOut;
180
181             // Roll Control
```

```
182             float aileron = map(rxChannel[RX_AILERON_CHAN], RX_MIN_INPUT, RX_MAX_INPUT,
                    -100.0f, 100.0f);
183             motors[0] -= aileron / 2.0f;
184             motors[1] -= aileron / 2.0f;
185             motors[2] += aileron / 2.0f;
186             motors[3] += aileron / 2.0f;
187
188             // Pitch Control
189             float elevator = map(rxChannel[RX_ELEVATOR_CHAN], RX_MIN_INPUT, RX_MAX_INPUT,
                     -100.0f, 100.0f);
190             motors[0] += elevator / 2.0f;
191             motors[1] -= elevator / 2.0f;
192             motors[2] += elevator / 2.0f;
193             motors[3] -= elevator / 2.0f;
194
195             // Rudder Control
196             float rudder = map(rxChannel[RX_RUDDER_CHAN], RX_MIN_INPUT, RX_MAX_INPUT,
                    -100.0f, 100.0f);
197             motors[0] -= rudder;
198             motors[1] += rudder;
199             motors[2] += rudder;
200             motors[3] -= rudder;
201
202             updateMotorsAll(motors);
203
204             //UARTprintf("%d\t%d\n", (int16_t)elevator, (int16_t)aileron);
205 #if 0
206             UARTprintf("%d\t%d\t\t", (int16_t)roll, (int16_t)pitch);
207             UARTprintf("%d\t%d\t\t", (int16_t)rollOut, (int16_t)pitchOut);
208             UARTprintf("%d\t%d\t%d\t%d\n", (int16_t)motors[0], (int16_t)motors[1], (
                    int16_t)motors[2], (int16_t)motors[3]);
209             UARTFlushTx(false);
210 #endif
211         }
212     }
213 }
214
215 // TODO:
216     // Save PID values in EEPROM
217     // Adjust PID values using pots on transmitter
218     // Only enable peripheral clock once
219     // Tune yaw PID values separately
220     // Make limit of integrated error adjustable
221     // Get self-level working - enable DLPF for accelerometer
222     // Use SPI instead of I2C
223     // Set Kd as well
```

10.2.2 KalmanX.c

```
1  /* Copyright (C) 2012 Kristian Lauszus, TKJ Electronics. All rights reserved.
2
3   This software may be distributed and modified under the terms of the GNU
4   General Public License version 2 (GPL2) as published by the Free Software
5   Foundation and appearing in the file GPL2.TXT included in the packaging of
6   this file. Please note that GPL2 Section 2[b] requires that all works based
7   on this software must also be made publicly available under the terms of
8   the GPL2 ("Copyleft").
9
10  Contact information
11  -------------------
12
13  Kristian Lauszus, TKJ Electronics
14  Web      :  http://www.tkjelectronics.com
15  e-mail   :  kristianl@tkjelectronics.com
16  */
17
18 // I modifed it, so it can be used with C99 instead of C++
19
20 #include "KalmanX.h"
21
22 /* Kalman filter variables */
23 float Q_angleX; // Process noise variance for the accelerometer
24 float Q_biasX; // Process noise variance for the gyro bias
25 float R_measureX; // Measurement noise variance - this is actually the variance of the
       measurement noise
```

```
26
27 float angleX; // The angle calculated by the Kalman filter - part of the 2x1 state vector
28 float biasX; // The gyro bias calculated by the Kalman filter - part of the 2x1 state
       vector
29 float rateX; // Unbiased rate calculated from the rate and the calculated bias - you have
       to call getAngle to update the rate
30
31 float PX[2][2]; // Error covariance matrix - This is a 2x2 matrix
32
33 void KalmanXInit(void) {
34     /* We will set the variables like so, these can also be tuned by the user */
35     Q_angleX = 0.001f;
36     Q_biasX = 0.003f;
37     R_measureX = 0.03f;
38
39     angleX = 0.0f; // Reset the angle
40     biasX = 0.0f; // Reset bias
41
42     PX[0][0] = 0.0f; // Since we assume that the bias is 0 and we know the starting angle
           (use setAngle), the error covariance matrix is set like so - see: http://en.
         wikipedia.org/wiki/Kalman_filter#Example_application.2C_technical
43     PX[0][1] = 0.0f;
44     PX[1][0] = 0.0f;
45     PX[1][1] = 0.0f;
46 }
47
48 // The angle should be in degrees and the rate should be in degrees per second and the
       delta time in seconds
49 float getAngleX(float newAngle, float newRate, float dt) {
50     // KasBot V2  -  Kalman filter module - http://www.x-firm.com/?page_id=145
51     // Modified by Kristian Lauszus
52     // See my blog post for more information: http://blog.tkjelectronics.dk/2012/09/a-
           practical-approach-to-kalman-filter-and-how-to-implement-it
53
54     // Discrete Kalman filter time update equations - Time Update ("Predict")
55     // Update xhat - Project the state ahead
56     /* Step 1 */
57     rateX = newRate - biasX;
58     angleX += dt * rateX;
59
60     // Update estimation error covariance - Project the error covariance ahead
61     /* Step 2 */
62     PX[0][0] += dt * (dt*PX[1][1] - PX[0][1] - PX[1][0] + Q_angleX);
63     PX[0][1] -= dt * PX[1][1];
64     PX[1][0] -= dt * PX[1][1];
65     PX[1][1] += Q_biasX * dt;
66
67     // Discrete Kalman filter measurement update equations - Measurement Update ("Correct
           ")
68     // Calculate Kalman gain - Compute the Kalman gain
69     /* Step 4 */
70     float S = PX[0][0] + R_measureX; // Estimate error
71     /* Step 5 */
72     float K[2]; // Kalman gain - This is a 2x1 vector
73     K[0] = PX[0][0] / S;
74     K[1] = PX[1][0] / S;
75
76     // Calculate angle and bias - Update estimate with measurement zk (newAngle)
77     /* Step 3 */
78     float y = newAngle - angleX; // Angle difference
79     /* Step 6 */
80     angleX += K[0] * y;
81     biasX += K[1] * y;
82
83     // Calculate estimation error covariance - Update the error covariance
84     /* Step 7 */
85     float P00_temp = PX[0][0];
86     float P01_temp = PX[0][1];
87
88     PX[0][0] -= K[0] * P00_temp;
89     PX[0][1] -= K[0] * P01_temp;
90     PX[1][0] -= K[1] * P00_temp;
91     PX[1][1] -= K[1] * P01_temp;
92
93     return angleX;
94 };
95
```

```
96 void setAngleX(float newAngle) { angleX = newAngle; }; // Used to set angle, this should
       be set as the starting angle
97 float getRateX(void) { return rateX; }; // Return the unbiased rate
98
99 /* These are used to tune the Kalman filter */
100 void setQangleX(float newQ_angle) { Q_angleX = newQ_angle; };
101 void setQbiasX(float newQ_bias) { Q_biasX = newQ_bias; };
102 void setRmeasureX(float newR_measure) { R_measureX = newR_measure; };
103
104 float getQangleX(void) { return Q_angleX; };
105 float getQbiasX(void) { return Q_biasX; };
106 float getRmeasureX(void) { return R_measureX; };
```

10.2.3 KalmanX.h

```
1  /* Copyright (C) 2012 Kristian Lauszus, TKJ Electronics. All rights reserved.
2
3   This software may be distributed and modified under the terms of the GNU
4   General Public License version 2 (GPL2) as published by the Free Software
5   Foundation and appearing in the file GPL2.TXT included in the packaging of
6   this file. Please note that GPL2 Section 2[b] requires that all works based
7   on this software must also be made publicly available under the terms of
8   the GPL2 ("Copyleft").
9
10  Contact information
11  -------------------
12
13  Kristian Lauszus, TKJ Electronics
14  Web       :  http://www.tkjelectronics.com
15  e-mail    :  kristianl@tkjelectronics.com
16  */
17
18 #ifndef __kalmanx_h__
19 #define __kalmanx_h__
20
21 #ifdef __cplusplus
22 extern "C" {
23 #endif
24
25 void KalmanXInit(void);
26 float getAngleX(float newAngle, float newRate, float dt);
27 void setAngleX(float newAngle);
28 float getRateX(void);
29 void setQangleX(float newQ_angle);
30 void setQbiasX(float newQ_bias);
31 void setRmeasureX(float newR_measure);
32
33 float getQangleX(void);
34 float getQbiasX(void);
35 float getRmeasureX(void);
36
37 #ifdef __cplusplus
38 }
39 #endif
40
41 #endif
```

10.2.4 KalmanY.c

```
1  /* Copyright (C) 2012 Kristian Lauszus, TKJ Electronics. All rights reserved.
2
3   This software may be distributed and modified under the terms of the GNU
4   General Public License version 2 (GPL2) as published by the Free Software
5   Foundation and appearing in the file GPL2.TXT included in the packaging of
6   this file. Please note that GPL2 Section 2[b] requires that all works based
7   on this software must also be made publicly available under the terms of
8   the GPL2 ("Copyleft").
9
10  Contact information
11  -------------------
12
13  Kristian Lauszus, TKJ Electronics
```

```
14  Web      :  http://www.tkjelectronics.com
15  e-mail   :  kristianl@tkjelectronics.com
16  */
17
18 // I modifed it, so it can be used with C99 instead of C++
19
20 #include "KalmanY.h"
21
22 /* Kalman filter variables */
23 float Q_angleY; // Process noise variance for the accelerometer
24 float Q_biasY; // Process noise variance for the gyro bias
25 float R_measureY; // Measurement noise variance - this is actually the variance of the
       measurement noise
26
27 float angleY; // The angle calculated by the Kalman filter - part of the 2x1 state vector
28 float biasY; // The gyro bias calculated by the Kalman filter - part of the 2x1 state
       vector
29 float rateY; // Unbiased rate calculated from the rate and the calculated bias - you have
        to call getAngle to update the rate
30
31 float PY[2][2]; // Error covariance matrix - This is a 2x2 matrix
32
33 void KalmanYInit(void) {
34     /* We will set the variables like so, these can also be tuned by the user */
35     Q_angleY = 0.001f;
36     Q_biasY = 0.003f;
37     R_measureY = 0.03f;
38
39     angleY = 0.0f; // Reset the angle
40     biasY = 0.0f; // Reset bias
41
42     PY[0][0] = 0.0f; // Since we assume that the bias is 0 and we know the starting angle
           (use setAngle), the error covariance matrix is set like so - see: http://en.
          wikipedia.org/wiki/Kalman_filter#Example_application.2C_technical
43     PY[0][1] = 0.0f;
44     PY[1][0] = 0.0f;
45     PY[1][1] = 0.0f;
46 }
47
48 // The angle should be in degrees and the rate should be in degrees per second and the
       delta time in seconds
49 float getAngleY(float newAngle, float newRate, float dt) {
50     // KasBot V2  -  Kalman filter module - http://www.x-firm.com/?page_id=145
51     // Modified by Kristian Lauszus
52     // See my blog post for more information: http://blog.tkjelectronics.dk/2012/09/a-
           practical-approach-to-kalman-filter-and-how-to-implement-it
53
54     // Discrete Kalman filter time update equations - Time Update ("Predict")
55     // Update xhat - Project the state ahead
56     /* Step 1 */
57     rateY = newRate - biasY;
58     angleY += dt * rateY;
59
60     // Update estimation error covariance - Project the error covariance ahead
61     /* Step 2 */
62     PY[0][0] += dt * (dt*PY[1][1] - PY[0][1] - PY[1][0] + Q_angleY);
63     PY[0][1] -= dt * PY[1][1];
64     PY[1][0] -= dt * PY[1][1];
65     PY[1][1] += Q_biasY * dt;
66
67     // Discrete Kalman filter measurement update equations - Measurement Update ("Correct
           ")
68     // Calculate Kalman gain - Compute the Kalman gain
69     /* Step 4 */
70     float S = PY[0][0] + R_measureY; // Estimate error
71     /* Step 5 */
72     float K[2]; // Kalman gain - This is a 2x1 vector
73     K[0] = PY[0][0] / S;
74     K[1] = PY[1][0] / S;
75
76     // Calculate angle and bias - Update estimate with measurement zk (newAngle)
77     /* Step 3 */
78     float y = newAngle - angleY; // Angle difference
79     /* Step 6 */
80     angleY += K[0] * y;
81     biasY += K[1] * y;
82
83     // Calculate estimation error covariance - Update the error covariance
```

```
84      /* Step 7 */
85      float P00_temp = PY[0][0];
86      float P01_temp = PY[0][1];
87
88      PY[0][0] -= K[0] * P00_temp;
89      PY[0][1] -= K[0] * P01_temp;
90      PY[1][0] -= K[1] * P00_temp;
91      PY[1][1] -= K[1] * P01_temp;
92
93      return angleY;
94  };
95
96  void setAngleY(float newAngle) { angleY = newAngle; }; // Used to set angle, this should
        be set as the starting angle
97  float getRateY(void) { return rateY; }; // Return the unbiased rate
98
99  /* These are used to tune the Kalman filter */
100 void setQangleY(float newQ_angle) { Q_angleY = newQ_angle; };
101 void setQbiasY(float newQ_bias) { Q_biasY = newQ_bias; };
102 void setRmeasureY(float newR_measure) { R_measureY = newR_measure; };
103
104 float getQangleY(void) { return Q_angleY; };
105 float getQbiasY(void) { return Q_biasY; };
106 float getRmeasureY(void) { return R_measureY; };
```

10.2.5 KalmanY.h

```
1   /* Copyright (C) 2012 Kristian Lauszus, TKJ Electronics. All rights reserved.
2
3    This software may be distributed and modified under the terms of the GNU
4    General Public License version 2 (GPL2) as published by the Free Software
5    Foundation and appearing in the file GPL2.TXT included in the packaging of
6    this file. Please note that GPL2 Section 2[b] requires that all works based
7    on this software must also be made publicly available under the terms of
8    the GPL2 ("Copyleft").
9
10   Contact information
11   -------------------
12
13   Kristian Lauszus, TKJ Electronics
14   Web      :  http://www.tkjelectronics.com
15   e-mail   :  kristianl@tkjelectronics.com
16   */
17
18  #ifndef __kalmany_h__
19  #define __kalmany_h__
20
21  #ifdef __cplusplus
22  extern "C" {
23  #endif
24
25  void KalmanYInit(void);
26  float getAngleY(float newAngle, float newRate, float dt);
27  void setAngleY(float newAngle);
28  float getRateY(void);
29  void setQangleY(float newQ_angle);
30  void setQbiasY(float newQ_bias);
31  void setRmeasureY(float newR_measure);
32
33  float getQangleY(void);
34  float getQbiasY(void);
35  float getRmeasureY(void);
36
37  #ifdef __cplusplus
38  }
39  #endif
40
41  #endif
```

10.2.6 MPU6500.c

```
1   /* Copyright (C) 2014 Kristian Lauszus, TKJ Electronics. All rights reserved.
```

```
2
3   This software may be distributed and modified under the terms of the GNU
4   General Public License version 2 (GPL2) as published by the Free Software
5   Foundation and appearing in the file GPL2.TXT included in the packaging of
6   this file. Please note that GPL2 Section 2[b] requires that all works based
7   on this software must also be made publicly available under the terms of
8   the GPL2 ("Copyleft").
9
10  Contact information
11  -------------------
12
13  Kristian Lauszus, TKJ Electronics
14  Web      :  http://www.tkjelectronics.com
15  e-mail   :  kristianl@tkjelectronics.com
16 */
17
18 #include <stdint.h>
19 #include <stdbool.h>
20 #include <math.h>
21
22 #include "MPU6500.h"
23 #include "time.h"
24 #include "KalmanX.h"
25 #include "KalmanY.h"
26
27 #include "inc/hw_memmap.h"
28 #include "driverlib/gpio.h"
29 #include "driverlib/pin_map.h"
30 #include "driverlib/ssi.h"
31 #include "driverlib/i2c.h"
32 #include "driverlib/sysctl.h"
33 #include "utils/uartstdio.h" // Add "UART_BUFFERED" to preprocessor
34
35 #define SLAVE_ADDRESS   0x68
36 #define PI              3.1415926535897932384626433832795f
37 #define RAD_TO_DEG      57.295779513082320876798154814105f
38
39 #define GPIO_MPU_INT_PERIPH     SYSCTL_PERIPH_GPIOE
40 #define GPIO_MPU_INT_BASE       GPIO_PORTE_BASE
41 #define GPIO_MPU_INT_PIN        GPIO_PIN_3
42
43 static int16_t gyroZero[3];
44
45 bool dataReadyMPU6500(void) {
46     return GPIOPinRead(GPIO_MPU_INT_BASE, GPIO_MPU_INT_PIN);
47 }
48
49 void getMPU6500Gyro(int16_t *gyroData) {
50     uint8_t buf[6];
51
52     i2cReadData(0x43, buf, 6);
53     gyroData[0] = (buf[0] << 8) | buf[1]; // X
54     gyroData[1] = (buf[2] << 8) | buf[3]; // Y
55     gyroData[2] = (buf[4] << 8) | buf[5]; // Z
56
57     for (uint8_t axis = 0; axis < 3; axis++)
58         gyroData[axis] -= gyroZero[axis];
59 }
60
61 void getMPU6500Angles(float *roll, float *pitch, float dt) {
62     int16_t accData[3], gyroData[3];
63     updateMPU6500(accData, gyroData);
64
65     // Source: https://github.com/cleanflight/cleanflight
66     const float accz_lpf_cutoff = 5.0f;
67     const float fc_acc = 0.5f / (PI * accz_lpf_cutoff); // Calculate RC time constant
           used in the accZ lpf
68     static float accz_smooth = 0;
69     accz_smooth = accz_smooth + (dt / (fc_acc + dt)) * (accData[2] - accz_smooth); // Low
           pass filter
70
71     float gyroRate[3];
72     for (uint8_t axis = 0; axis < 3; axis++)
73         gyroRate[axis] = (float)(gyroData[axis] - gyroZero[axis]) / 16.4f;
74
75     // Pitch should increase when pitching quadcopter downward
76     // and roll should increase when tilting quadcopter clockwise
77
```

```
78      /*static float gyroAngle[3] = { 0, 0, 0 };
79      for (uint8_t axis = 0; axis < 3; axis++)
80          gyroAngle[axis] += gyroRate[axis] * dt; // Gyro angle is only used for debugging */
81
82      float rollAcc = atanf(accData[0] / sqrtf(accData[1] * accData[1] + accz_smooth * accz_smooth)) * RAD_TO_DEG;
83      float pitchAcc  = atan2f(-accData[1], -accz_smooth) * RAD_TO_DEG;
84
85      *roll = getAngleX(rollAcc, gyroRate[1], dt);
86      *pitch = getAngleY(pitchAcc, gyroRate[0], dt);
87  /*
88      UARTprintf("%d\t%d\t%d\t\t", (int16_t)rollAcc, (int16_t)gyroAngle[1], (int16_t)*roll);
89      delay(1);
90      UARTprintf("%d\t%d\t%d\n", (int16_t)pitchAcc, (int16_t)gyroAngle[0], (int16_t)*pitch);
91      UARTFlushTx(false);
92  */
93  /*
94      static float compAngleX, compAngleY;
95      compAngleX = 0.93f * (compAngleX + gyroRate[0] * dt) + 0.07f * rollAcc; // Calculate the angle using a Complimentary filter
96      compAngleY = 0.93f * (compAngleY + gyroRate[1] * dt) + 0.07f * pitchAcc;
97  */
98  }
99
100 /*void printMPU6050Debug(void) {
101     while (1) {
102 #if 0
103         UARTprintf("%d\t%d\t\t", (int16_t)KalmanX, (int16_t)KalmanY);
104         UARTprintf("%d\t%d\t\t", (int16_t)compAngleX, (int16_t)compAngleY);
105         UARTprintf("%d\t%d\t\t", (int16_t)roll, (int16_t)pitch);
106         UARTprintf("%d\t%d\t%d\n", (int16_t)gyroAngle[0], (int16_t)gyroAngle[1], (int16_t)gyroAngle[2]);
107 #else
108         UARTprintf("%d\t%d\t", (int16_t)roll, (int16_t)gyroAngle[0]);
109         delay(1);
110         UARTprintf("%d\t%d\t\t", (int16_t)compAngleX, (int16_t)KalmanX);
111         delay(1);
112         UARTprintf("%d\t%d\t", (int16_t)pitch, (int16_t)gyroAngle[1]);
113         delay(1);
114         UARTprintf("%d\t%d\t\n", (int16_t)compAngleY, (int16_t)KalmanY);
115 #endif
116         delay(10);
117     }
118 }*/
119
120 void initMPU6500_i2c(void) {
121     SysCtlPeripheralEnable(SYSCTL_PERIPH_I2C1); // Enable I2C1 peripheral
122     SysCtlDelay(2); // Insert a few cycles after enabling the peripheral to allow the clock to be fully activated
123     SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA); // Enable GPIOA peripheral
124     SysCtlDelay(2); // Insert a few cycles after enabling the peripheral to allow the clock to be fully activated
125
126     // Use altenate function
127     GPIOPinConfigure(GPIO_PA6_I2C1SCL);
128     GPIOPinConfigure(GPIO_PA7_I2C1SDA);
129
130     GPIOPinTypeI2CSCL(GPIO_PORTA_BASE, GPIO_PIN_6); // Use pin with I2C SCL peripheral
131     GPIOPinTypeI2C(GPIO_PORTA_BASE, GPIO_PIN_7); // Use pin with I2C peripheral
132
133     I2CMasterInitExpClk(I2C1_BASE, SysCtlClockGet(), true); // Enable and set frequency to 400 kHz
134
135     delay(100);
136
137     uint8_t i2cBuffer[4]; // Buffer for I2C data
138
139     i2cBuffer[0] = i2cRead(0x75);
140     if (i2cBuffer[0] == 0x70) // Read "WHO_AM_I" register
141         UARTprintf("MPU-6500 found\n");
142     else {
143         UARTprintf("Could not find MPU-6500: %2X\n", i2cBuffer[0]);
144         while (1);
145     }
146
```

```
147     i2cWrite(0x6B, (1 << 7)); // Reset device, this resets all internal registers to
            their default values
148     delay(100);
149     while (i2cRead(0x6B) & (1 << 7)) {
150         // Wait for the bit to clear
151     };
152     delay(100);
153     i2cWrite(0x6B, (1 << 3) | (1 << 0)); // Disable sleep mode, disable temperature
            sensor and use PLL as clock reference
154
155     i2cBuffer[0] = 0; // Set the sample rate to 1kHz - 1kHz/(1+0) = 1kHz
156     i2cBuffer[1] = 0x03; // Disable FSYNC and set 41 Hz Gyro filtering, 1 KHz sampling
157     i2cBuffer[2] = 3 << 3; // Set Gyro Full Scale Range to +-2000deg/s
158     i2cBuffer[3] = 2 << 3; // Set Accelerometer Full Scale Range to +-8g
159     // TODO: Enable DLPF for accelerometer as well:
160     //i2cBuffer[4] = 0x03; // 41 Hz Acc filtering
161     i2cWriteData(0x19, i2cBuffer, 4); // Write to all four registers at once
162
163     /* Enable Data Ready Interrupt on INT pin */
164     i2cBuffer[0] = (1 << 5) | (1 << 4); // Enable LATCH_INT_EN and INT_RD_CLEAR
165                                         // When this bit is equal to 1, the INT pin is
                                                held high until the interrupt is cleared
166                                         // When this bit is equal to 1, interrupt status
                                                bits are cleared on any read operation
167     i2cBuffer[1] = (1 << 0); // Enable DATA_RDY_EN - When set to 1, this bit enables the
            Data Ready interrupt, which occurs each time a write operation to all of the
            sensor registers has been completed
168     i2cWriteData(0x37, i2cBuffer, 2); // Write to both registers at once
169
170     // Set INT input pin
171     SysCtlPeripheralEnable(GPIO_MPU_INT_PERIPH); // Enable GPIO peripheral
172     SysCtlDelay(2); // Insert a few cycles after enabling the peripheral to allow the
            clock to be fully activated
173     GPIOPinTypeGPIOInput(GPIO_MPU_INT_BASE, GPIO_MPU_INT_PIN); // Set as input
174
175     delay(100); // Wait for sensor to stabilize
176
177     //printMPU6050Debug();
178
179     while (!dataReadyMPU6500()) {
180         // Wait until date is ready
181     }
182
183     // TOOD: Read gyro values multiple times and check if it's moved while doing so
184
185     int16_t accData[3]; // This is just tossed away
186     updateMPU6500(accData, gyroZero); // Get gyro zero values
187
188     KalmanXInit();
189     KalmanYInit();
190
191     setAngleX(0.0f); // Set starting angle
192     setAngleY(0.0f);
193 }
194
195 void updateMPU6500(int16_t *accData, int16_t *gyroData) {
196     uint8_t buf[14];
197
198     i2cReadData(0x3B, buf, 14);
199
200     accData[0] = (buf[0] << 8) | buf[1]; // X
201     accData[1] = (buf[2] << 8) | buf[3]; // Y
202     accData[2] = (buf[4] << 8) | buf[5]; // Z
203
204     gyroData[0] = (buf[8] << 8) | buf[9]; // X
205     gyroData[1] = (buf[10] << 8) | buf[11]; // Y
206     gyroData[2] = (buf[12] << 8) | buf[13]; // Z
207 }
208
209 void i2cWrite(uint8_t addr, uint8_t data) {
210     i2cWriteData(addr, &data, 1);
211 }
212
213 void i2cWriteData(uint8_t addr, uint8_t *data, uint8_t length) {
214     I2CMasterSlaveAddrSet(I2C1_BASE, SLAVE_ADDRESS, false); // Set to write mode
215
216     I2CMasterDataPut(I2C1_BASE, addr); // Place address into data register
217     I2CMasterControl(I2C1_BASE, I2C_MASTER_CMD_BURST_SEND_START); // Send start condition
```

```
218     while (I2CMasterBusy(I2C1_BASE)); // Wait until transfer is done
219
220     for (uint8_t i = 0; i < length - 1; i++) {
221         I2CMasterDataPut(I2C1_BASE, data[i]); // Place data into data register
222         I2CMasterControl(I2C1_BASE, I2C_MASTER_CMD_BURST_SEND_CONT); // Send continues
                condition
223         while (I2CMasterBusy(I2C1_BASE)); // Wait until transfer is done
224     }
225
226     I2CMasterDataPut(I2C1_BASE, data[length - 1]); // Place data into data register
227     I2CMasterControl(I2C1_BASE, I2C_MASTER_CMD_BURST_SEND_FINISH); // Send finish
            condition
228     while (I2CMasterBusy(I2C1_BASE)); // Wait until transfer is done
229 }
230
231 uint8_t i2cRead(uint8_t addr) {
232     I2CMasterSlaveAddrSet(I2C1_BASE, SLAVE_ADDRESS, false); // Set to write mode
233
234     I2CMasterDataPut(I2C1_BASE, addr); // Place address into data register
235     I2CMasterControl(I2C1_BASE, I2C_MASTER_CMD_SINGLE_SEND); // Send data
236     while (I2CMasterBusy(I2C1_BASE)); // Wait until transfer is done
237
238     I2CMasterSlaveAddrSet(I2C1_BASE, SLAVE_ADDRESS, true); // Set to read mode
239
240     I2CMasterControl(I2C1_BASE, I2C_MASTER_CMD_SINGLE_RECEIVE); // Tell master to read
            data
241     while (I2CMasterBusy(I2C1_BASE)); // Wait until transfer is done
242     return I2CMasterDataGet(I2C1_BASE); // Read data
243 }
244
245 void i2cReadData(uint8_t addr, uint8_t *data, uint8_t length) {
246     I2CMasterSlaveAddrSet(I2C1_BASE, SLAVE_ADDRESS, false); // Set to write mode
247
248     I2CMasterDataPut(I2C1_BASE, addr); // Place address into data register
249     I2CMasterControl(I2C1_BASE, I2C_MASTER_CMD_SINGLE_SEND); // Send data
250     while (I2CMasterBusy(I2C1_BASE)); // Wait until transfer is done
251
252     I2CMasterSlaveAddrSet(I2C1_BASE, SLAVE_ADDRESS, true); // Set to read mode
253
254     I2CMasterControl(I2C1_BASE, I2C_MASTER_CMD_BURST_RECEIVE_START); // Send start
            condition
255     while (I2CMasterBusy(I2C1_BASE)); // Wait until transfer is done
256     data[0] = I2CMasterDataGet(I2C1_BASE); // Place data into data register
257
258     for (uint8_t i = 1; i < length - 1; i++) {
259         I2CMasterControl(I2C1_BASE, I2C_MASTER_CMD_BURST_RECEIVE_CONT); // Send continues
                condition
260         while (I2CMasterBusy(I2C1_BASE)); // Wait until transfer is done
261         data[i] = I2CMasterDataGet(I2C1_BASE); // Place data into data register
262     }
263
264     I2CMasterControl(I2C1_BASE, I2C_MASTER_CMD_BURST_RECEIVE_FINISH); // Send finish
            condition
265     while (I2CMasterBusy(I2C1_BASE)); // Wait until transfer is done
266     data[length - 1] = I2CMasterDataGet(I2C1_BASE); // Place data into data register
267 }
268
269 void spiSelect(bool enable) {
270     GPIOPinWrite(GPIO_PORTA_BASE, GPIO_PIN_3, enable ? 0 : GPIO_PIN_3); // The SS pin is
            active low
271 }
272
273 void initMPU6500(void) {
274     SysCtlPeripheralEnable(SYSCTL_PERIPH_SSI0); // Enable SSI0 peripheral
275     SysCtlDelay(2); // Insert a few cycles after enabling the peripheral to allow the
            clock to be fully activated
276     SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA); // Enable GPIOA peripheral
277     SysCtlDelay(2); // Insert a few cycles after enabling the peripheral to allow the
            clock to be fully activated
278
279     // Use alternate function
280     GPIOPinConfigure(GPIO_PA2_SSI0CLK);
281     GPIOPinConfigure(GPIO_PA4_SSI0RX);
282     GPIOPinConfigure(GPIO_PA5_SSI0TX);
283
284 #if 0
285     GPIOPinConfigure(GPIO_PA3_SSI0FSS);
```

```
286     GPIOPinTypeSSI(GPIO_PORTA_BASE, GPIO_PIN_2 | GPIO_PIN_3 | GPIO_PIN_4 | GPIO_PIN_5);
            // Use pins with SSI peripheral
287 #else
288     GPIOPinTypeGPIOOutput(GPIO_PORTA_BASE, GPIO_PIN_3); // Set SS as output
289     spiSelect(false);
290     GPIOPinTypeSSI(GPIO_PORTA_BASE, GPIO_PIN_2 | GPIO_PIN_4 | GPIO_PIN_5); // Use pins
            with SSI peripheral
291 #endif
292
293     //SSIClockSourceSet(SSI0_BASE, SSI_CLOCK_SYSTEM); // Set clock source
294
295     // Configure the SSI to MODE0, 1 MHz, and 8-bit data
296     SSIConfigSetExpClk(SSI0_BASE, SysCtlClockGet(), SSI_FRF_MOTO_MODE_0, SSI_MODE_MASTER,
            1000000, 8);
297     SSIEnable(SSI0_BASE); // Enable the SSI module
298
299     // PWR_MGMT_1
300     // Reset the device
301     spiWriteData(0x6B, 0x80);
302
303     delay(100);
304
305     // PWR_MGMT_1
306     // Auto selects the best available clock source - PLL if ready, else use the Internal
            oscillator
307     // Disable sleep mode
308     spiWriteData(0x6B, 0x01);
309
310     delay(100);
311
312     // USER_CTRL
313     // Reset I2C Slave module and put the serial interface in SPI mode only
314     spiWriteData(0x6A, 0x10/* | 0x8 | 0x4 | 0x1*/);
315
316     delay(100);
317
318 /*
319   uint32_t pui32DataRx[3];
320     while (SSIDataGetNonBlocking(SSI0_BASE, &pui32DataRx[0]))
321   {
322         // Empty FIFO
323   }
324 */
325     uint32_t buffer[100];
326     spiReadData(0x75, buffer);
327
328     if (buffer[0] == 0x70)
329         UARTprintf("MPU-6500 initialized\n");
330     else
331         UARTprintf("Could not initialize MPU-6500: %2X\n", buffer[0]);
332 }
333
334 void spiReadData(uint32_t addr, uint32_t *buffer) {
335     spiSelect(true);
336     SSIDataPut(SSI0_BASE, addr | 0x80); // Indicate read operation
337     while (SSIBusy(SSI0_BASE));
338     SSIDataGet(SSI0_BASE, buffer);
339     while (SSIBusy(SSI0_BASE));
340     spiSelect(false);
341 }
342
343 void spiWriteData(uint32_t addr, uint32_t buffer) {
344     spiSelect(true);
345     SSIDataPut(SSI0_BASE, addr);
346     SSIDataPut(SSI0_BASE, buffer);
347     while (SSIBusy(SSI0_BASE));
348     spiSelect(false);
349 }
```

10.2.7 MPU6500.h

```
1 /* Copyright (C) 2014 Kristian Lauszus, TKJ Electronics. All rights reserved.
2
3  This software may be distributed and modified under the terms of the GNU
4  General Public License version 2 (GPL2) as published by the Free Software
```

```
 Foundation and appearing in the file GPL2.TXT included in the packaging of
 this file. Please note that GPL2 Section 2[b] requires that all works based
 on this software must also be made publicly available under the terms of
 the GPL2 ("Copyleft").

 Contact information
 -------------------

 Kristian Lauszus, TKJ Electronics
 Web      :  http://www.tkjelectronics.com
 e-mail   :  kristianl@tkjelectronics.com
*/

#ifndef __mpu6500_h__
#define __mpu6500_h__

#ifdef __cplusplus
extern "C" {
#endif

bool dataReadyMPU6500(void);
void getMPU6500Angles(float *roll, float *pitch, float dt);
void getMPU6500Gyro(int16_t *gyroData);

void initMPU6500_i2c(void);
void updateMPU6500(int16_t *accData, int16_t *gyroData);

void i2cWrite(uint8_t addr, uint8_t data);
void i2cWriteData(uint8_t addr, uint8_t *date, uint8_t length);
uint8_t i2cRead(uint8_t addr);
void i2cReadData(uint8_t addr, uint8_t *data, uint8_t length);

void initMPU6500(void);
void spiReadData(uint32_t addr, uint32_t *buffer);
void spiWriteData(uint32_t addr, uint32_t buffer);

#ifdef __cplusplus
}
#endif

#endif
```

10.2.8 PID.c

```
/* Copyright (C) 2014 Kristian Lauszus, TKJ Electronics. All rights reserved.

 This software may be distributed and modified under the terms of the GNU
 General Public License version 2 (GPL2) as published by the Free Software
 Foundation and appearing in the file GPL2.TXT included in the packaging of
 this file. Please note that GPL2 Section 2[b] requires that all works based
 on this software must also be made publicly available under the terms of
 the GPL2 ("Copyleft").

 Contact information
 -------------------

 Kristian Lauszus, TKJ Electronics
 Web      :  http://www.tkjelectronics.com
 e-mail   :  kristianl@tkjelectronics.com
*/

#include <stdint.h>
#include <stdbool.h>

#include "PID.h"

pid_t pidRoll, pidPitch, pidYaw;

float updatePID(pid_t *pid, float setPoint, float input, float dt) {
    float error = setPoint - input;
    float pTerm = pid->Kp * error;
    pid->integratedError += error * dt;
    pid->integratedError = constrain(pid->integratedError, -10.0f, 10.0f); // Limit the
        integrated error
    float iTerm = pid->Ki * pid->integratedError;
```

```
31      float dTerm = pid->Kd * (error - pid->lastError) / dt;
32      pid->lastError = error;
33      return pTerm + iTerm + dTerm;
34  }
```

10.2.9 PID.h

```
1   /* Copyright (C) 2014 Kristian Lauszus, TKJ Electronics. All rights reserved.
2
3    This software may be distributed and modified under the terms of the GNU
4    General Public License version 2 (GPL2) as published by the Free Software
5    Foundation and appearing in the file GPL2.TXT included in the packaging of
6    this file. Please note that GPL2 Section 2[b] requires that all works based
7    on this software must also be made publicly available under the terms of
8    the GPL2 ("Copyleft").
9
10   Contact information
11   -------------------
12
13   Kristian Lauszus, TKJ Electronics
14   Web      :  http://www.tkjelectronics.com
15   e-mail   :  kristianl@tkjelectronics.com
16  */
17
18  #ifndef __pid_h__
19  #define __pid_h__
20
21  #ifdef __cplusplus
22  extern "C" {
23  #endif
24
25  // From Arduino source code
26  #define constrain(amt,low,high) ((amt)<(low)?(low):((amt)>(high)?(high):(amt)))
27
28  typedef struct {
29      float Kp, Ki, Kd; // PID variables
30      float lastError, integratedError;
31  } __attribute__((packed)) pid_t;
32
33  extern pid_t pidRoll, pidPitch, pidYaw;
34
35  float updatePID(pid_t *pid, float setPoint, float input, float dt);
36
37  #ifdef __cplusplus
38  }
39  #endif
40
41  #endif
```

10.2.10 PPM.c

```
1   /* Copyright (C) 2014 Kristian Lauszus, TKJ Electronics. All rights reserved.
2
3    This software may be distributed and modified under the terms of the GNU
4    General Public License version 2 (GPL2) as published by the Free Software
5    Foundation and appearing in the file GPL2.TXT included in the packaging of
6    this file. Please note that GPL2 Section 2[b] requires that all works based
7    on this software must also be made publicly available under the terms of
8    the GPL2 ("Copyleft").
9
10   Contact information
11   -------------------
12
13   Kristian Lauszus, TKJ Electronics
14   Web      :  http://www.tkjelectronics.com
15   e-mail   :  kristianl@tkjelectronics.com
16  */
17
18  #include <stdint.h>
19  #include <stdbool.h>
20
21  #include "PPM.h"
```

```
22
23 #include "inc/hw_memmap.h"
24 #include "driverlib/gpio.h"
25 #include "driverlib/pin_map.h"
26 #include "driverlib/pwm.h"
27 #include "driverlib/sysctl.h"
28
29 static uint16_t period;
30
31 void initPPM(void) {
32     SysCtlPWMClockSet(SYSCTL_PWMDIV_4); // Set divider to 4
33
34     SysCtlPeripheralEnable(SYSCTL_PERIPH_PWM0); // Enable PWM peripheral
35     SysCtlDelay(2); // Insert a few cycles after enabling the peripheral to allow the
           clock to be fully activated
36     SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOB); // Enable GPIOB peripheral
37     SysCtlDelay(2); // Insert a few cycles after enabling the peripheral to allow the
           clock to be fully activated
38
39     // Use alternate function
40     GPIOPinConfigure(GPIO_PB6_M0PWM0);
41     GPIOPinConfigure(GPIO_PB7_M0PWM1);
42     GPIOPinConfigure(GPIO_PB4_M0PWM2);
43     GPIOPinConfigure(GPIO_PB5_M0PWM3);
44     GPIOPinTypePWM(GPIO_PORTB_BASE, GPIO_PIN_6 | GPIO_PIN_7 | GPIO_PIN_4 | GPIO_PIN_5);
           // Use pin with PWM peripheral
45
46     // Configure the PWM generator for count down mode with immediate updates to the
           parameters.
47     PWMGenConfigure(PWM0_BASE, PWM_GEN_0, PWM_GEN_MODE_DOWN | PWM_GEN_MODE_NO_SYNC);
48     PWMGenConfigure(PWM0_BASE, PWM_GEN_1, PWM_GEN_MODE_DOWN | PWM_GEN_MODE_NO_SYNC);
49
50     // The value is given by (SysClk * period) / divider
51     // The period is set to 2.5ms (400 Hz)
52     period = (SysCtlClockGet() / 10000 * 25) / 4; // 50000
53     PWMGenPeriodSet(PWM0_BASE, PWM_GEN_0, period); // Set the period.
54     PWMGenPeriodSet(PWM0_BASE, PWM_GEN_1, period); // Set the period.
55
56     // Start the timers in generator 0.
57     PWMGenEnable(PWM0_BASE, PWM_GEN_0);
58     PWMGenEnable(PWM0_BASE, PWM_GEN_1);
59
60     // Enable the outputs.
61     PWMOutputState(PWM0_BASE, PWM_OUT_0_BIT | PWM_OUT_1_BIT | PWM_OUT_2_BIT |
           PWM_OUT_3_BIT, true);
62
63     writePPMAllOff();
64 }
65
66 uint16_t getPeriod(void) {
67     return period;
68 }
69
70 void writePPMAllOff(void) {
71     // Turn of all motors
72     for (uint8_t i = 0; i < 4; i++)
73         writePPMUs(i, PPM_MIN);
74 }
75
76 float map(float x, float in_min, float in_max, float out_min, float out_max) {
77     float value = (x - in_min) * (out_max - out_min) / (in_max - in_min) + out_min; //
           From Arduino source code: https://github.com/arduino/Arduino/blob/ide-1.5.x/
           hardware/arduino/avr/cores/arduino/WMath.cpp
78     if (value > out_max)
79         value = out_max;
80     else if (value < out_min)
81         value = out_min;
82     return value;
83 }
84
85 void updateMotor(uint8_t motor, float value) {
86     // Motors are in the range from -100 to 100
87     if (value > 100.0f)
88         value = 100.0f;
89     else if (value < -100.0f)
90         value = -100.0f;
91
92     uint16_t motorOutput = map(value, -100.0f, 100.0f, PPM_MIN, PPM_MAX);
```

```
93      writePPMUs(motor, motorOutput);
94  }
95
96  void updateMotorsAll(float *values) {
97      for (uint8_t i = 0; i < 4; i++)
98          updateMotor(i, values[i]);
99  }
100
101 void writePPMUs(uint8_t motor, uint16_t us) {
102     writePPMWidth(motor, period * us / 2500); // 400 Hz
103 }
104
105 void writePPMWidth(uint8_t motor, uint16_t width) {
106     PWMPulseWidthSet(PWM0_BASE, motor == 0 ? PWM_OUT_0 : motor == 1 ? PWM_OUT_1 : motor
            == 2 ? PWM_OUT_2 : PWM_OUT_3, width);
107 }
```

10.2.11 PPM.h

```
1   /* Copyright (C) 2014 Kristian Lauszus, TKJ Electronics. All rights reserved.
2
3    This software may be distributed and modified under the terms of the GNU
4    General Public License version 2 (GPL2) as published by the Free Software
5    Foundation and appearing in the file GPL2.TXT included in the packaging of
6    this file. Please note that GPL2 Section 2[b] requires that all works based
7    on this software must also be made publicly available under the terms of
8    the GPL2 ("Copyleft").
9
10   Contact information
11   -------------------
12
13   Kristian Lauszus, TKJ Electronics
14   Web      :  http://www.tkjelectronics.com
15   e-mail   :  kristianl@tkjelectronics.com
16  */
17
18  #ifndef __ppm_h__
19  #define __ppm_h__
20
21  #ifdef __cplusplus
22  extern "C" {
23  #endif
24
25  #define PPM_MIN 1064 // From SimonK firmware
26  #define PPM_MAX 1864 // From SimonK firmware
27
28  void initPPM(void);
29  void writePPMAllOff(void);
30  void updateMotor(uint8_t motor, float value);
31  void updateMotorsAll(float *values);
32  void writePPMUs(uint8_t motor, uint16_t us);
33  void writePPMWidth(uint8_t motor, uint16_t width);
34  uint16_t getPeriod(void);
35
36  float map(float x, float in_min, float in_max, float out_min, float out_max);
37
38  #ifdef __cplusplus
39  }
40  #endif
41
42  #endif
```

10.2.12 RX.c

```
1   /* Copyright (C) 2014 Kristian Lauszus, TKJ Electronics. All rights reserved.
2
3    This software may be distributed and modified under the terms of the GNU
4    General Public License version 2 (GPL2) as published by the Free Software
5    Foundation and appearing in the file GPL2.TXT included in the packaging of
6    this file. Please note that GPL2 Section 2[b] requires that all works based
7    on this software must also be made publicly available under the terms of
8    the GPL2 ("Copyleft").
```

```
9
10  Contact information
11  -------------------
12
13  Kristian Lauszus, TKJ Electronics
14  Web       :  http://www.tkjelectronics.com
15  e-mail    :  kristianl@tkjelectronics.com
16 */
17
18 #include <stdint.h>
19 #include <stdbool.h>
20
21 #include "RX.h"
22 #include "time.h"
23
24 #include "inc/hw_memmap.h"
25 #include "inc/hw_ints.h"
26 #include "driverlib/gpio.h"
27 #include "driverlib/interrupt.h"
28 #include "driverlib/pin_map.h"
29 #include "driverlib/sysctl.h"
30 #include "driverlib/systick.h"
31 #include "driverlib/timer.h"
32 //#include "utils/uartstdio.h" // Add "UART_BUFFERED" to preprocessor
33
34 volatile uint16_t rxChannel[RX_NUM_CHANNELS];
35 volatile bool validRXData;
36
37 void CaptureHandler(void) {
38     static uint8_t channelIndex = 0;
39     static uint32_t prev = 0;
40     static bool last_edge = false;
41     //static uint32_t prev_micros = 0;
42
43     TimerIntClear(WTIMER1_BASE, TIMER_CAPA_EVENT); // Clear interrupt
44     uint32_t curr = TimerValueGet(WTIMER1_BASE, TIMER_A); // Read capture value
45     bool edge = GPIOPinRead(GPIO_PORTC_BASE, GPIO_PIN_6); // Read the GPIO pin
46
47     if (last_edge && !edge) { // Check that we are going from a positive to falling egde
48         uint32_t diff = curr - prev; // Calculate diff
49         uint32_t diff_us = 1000000UL / (SysCtlClockGet() / diff); // Convert to us
50 #if 0
51         UARTprintf("%u %u %d\n", diff, diff_us,  micros() - prev_micros);
52 #else
53         if (diff_us > 2700) { // Check if sync pulse is received - see: https://github.
               com/multiwii/baseflight/blob/master/src/drv_pwm.c
54             channelIndex = 0; // Reset channel index
55             validRXData = true;
56             for (uint8_t i = 0; i < RX_NUM_CHANNELS; i++) {
57                 if (rxChannel[i] == 0) // Make sure that all are above 0
58                     validRXData = false;
59             }
60             if (validRXData)
61                 TimerLoadSet(WTIMER1_BASE, TIMER_B, SysCtlClockGet() / 10 - 1); // Reset
                       timeout value to 100ms
62 #if 0
63             for (uint8_t i = 0; i < RX_NUM_CHANNELS; i++) {
64                 if (rxChannel[i] > 0)
65                     UARTprintf("%u\t", rxChannel[i]);
66                 else
67                     break;
68             }
69             UARTprintf("\n");
70 #endif
71         } else if (channelIndex < RX_NUM_CHANNELS)
72             rxChannel[channelIndex++] = diff_us;
73 #endif
74     }
75
76     prev = curr; // Store previous value
77     last_edge = edge; // Store last edge
78     //prev_micros = micros();
79 }
80
81 void TimeoutHandler(void) {
82     TimerIntClear(WTIMER1_BASE, TIMER_TIMB_TIMEOUT); // Clear interrupt
83     validRXData = false; // Indicate that connection was lost
84 }
```

```
85
86  // WTimer1A is used to measure the width of the pulses
87  // WTimer1B is used to turn off motors if the connection to the RX is lost
88  void initRX(void) {
89      SysCtlPeripheralEnable(SYSCTL_PERIPH_WTIMER1); // Enable Wide Timer1 peripheral
90      SysCtlDelay(2); // Insert a few cycles after enabling the peripheral to allow the
            clock to be fully activated
91
92      SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOC); // Enable GPIOC peripheral
93      SysCtlDelay(2); // Insert a few cycles after enabling the peripheral to allow the
            clock to be fully activated
94      GPIOPinConfigure(GPIO_PC6_WT1CCP0); // Use alternate function
95      GPIOPinTypeTimer(GPIO_PORTC_BASE, GPIO_PIN_6); // Use pin with timer peripheral
96
97      // Split timers and enable timer A event up-count timer and timer B as a periodic
            timer
98      TimerConfigure(WTIMER1_BASE, TIMER_CFG_SPLIT_PAIR | TIMER_CFG_A_CAP_TIME_UP |
            TIMER_CFG_B_PERIODIC);
99
100     // Configure WTimer1A
101     TimerControlEvent(WTIMER1_BASE, TIMER_A, TIMER_EVENT_BOTH_EDGES); // Interrupt on
            both edges
102     TimerIntRegister(WTIMER1_BASE, TIMER_A, CaptureHandler); // Register interrupt
            handler
103     TimerIntEnable(WTIMER1_BASE, TIMER_CAPA_EVENT); // Enable timer capture A event
            interrupt
104     IntPrioritySet(INT_WTIMER1A, 0); // Configure Timer 1A interrupt priority as 0
105     IntEnable(INT_WTIMER1A); // Enable wide Timer 1A interrupt
106
107     // Configure WTimer1B
108     TimerLoadSet(WTIMER1_BASE, TIMER_B, SysCtlClockGet() / 10 - 1); // Set to interrupt
            every 100ms
109     TimerIntRegister(WTIMER1_BASE, TIMER_B, TimeoutHandler); // Register interrupt
            handler
110     TimerIntEnable(WTIMER1_BASE, TIMER_TIMB_TIMEOUT); // Enable timer timeout interrupt
111     IntPrioritySet(INT_WTIMER1B, 0); // Configure Timer0A interrupt priority as 0
112     IntEnable(INT_WTIMER1B); // Enable wide Timer 1B interrupt
113
114     TimerEnable(WTIMER1_BASE, TIMER_BOTH); // Enable both timers
115
116     validRXData = false;
117 }
```

10.2.13 RX.h

```
1  /* Copyright (C) 2014 Kristian Lauszus, TKJ Electronics. All rights reserved.
2
3   This software may be distributed and modified under the terms of the GNU
4   General Public License version 2 (GPL2) as published by the Free Software
5   Foundation and appearing in the file GPL2.TXT included in the packaging of
6   this file. Please note that GPL2 Section 2[b] requires that all works based
7   on this software must also be made publicly available under the terms of
8   the GPL2 ("Copyleft").
9
10  Contact information
11  -------------------
12
13  Kristian Lauszus, TKJ Electronics
14  Web      :  http://www.tkjelectronics.com
15  e-mail   :  kristianl@tkjelectronics.com
16  */
17
18  #ifndef __rx_h__
19  #define __rx_h__
20
21  #ifdef __cplusplus
22  extern "C" {
23  #endif
24
25  enum {
26      RX_AILERON_CHAN = 0,
27      RX_ELEVATOR_CHAN,
28      RX_THROTTLE_CHAN,
29      RX_RUDDER_CHAN,
30      RX_AUX1_CHAN,
```

```
31      RX_AUX2_CHAN,
32      RX_NUM_CHANNELS,
33  };
34
35  #define RX_MIN_INPUT 665
36  #define RX_MAX_INPUT 1730
37
38  extern volatile uint16_t rxChannel[RX_NUM_CHANNELS];
39  extern volatile bool validRXData;
40
41  void initRX(void);
42
43  #ifdef __cplusplus
44  }
45  #endif
46
47  #endif
```

10.2.14 time.c

```
1   /* Copyright (C) 2014 Kristian Lauszus, TKJ Electronics. All rights reserved.
2
3    This software may be distributed and modified under the terms of the GNU
4    General Public License version 2 (GPL2) as published by the Free Software
5    Foundation and appearing in the file GPL2.TXT included in the packaging of
6    this file. Please note that GPL2 Section 2[b] requires that all works based
7    on this software must also be made publicly available under the terms of
8    the GPL2 ("Copyleft").
9
10   Contact information
11   -------------------
12
13   Kristian Lauszus, TKJ Electronics
14   Web      :  http://www.tkjelectronics.com
15   e-mail   :  kristianl@tkjelectronics.com
16  */
17
18  #include <stdint.h>
19  #include <stdbool.h>
20
21  #include "time.h"
22
23  #include "driverlib/sysctl.h"
24  #include "driverlib/systick.h"
25
26  static volatile uint32_t counter;
27
28  void SycTickHandler(void) {
29      counter++;
30  }
31
32  void initTime(void) {
33      SysTickPeriodSet(SysCtlClockGet() / 1000000UL); // 1000 for miliseconds & 1000000 for
            microseconds
34      SysTickIntRegister(SycTickHandler);
35      SysTickIntEnable();
36      SysTickEnable();
37  }
38
39  void delay(uint32_t ms) {
40      SysCtlDelay(SysCtlClockGet() / 3000UL * ms); // TOOD: Check if it needs to be tuned
41  }
42
43  void delayMicroseconds(uint32_t us) {
44      SysCtlDelay(SysCtlClockGet() / 3000000UL * us); // TOOD: Check if it needs to be
            tuned
45  }
46
47  uint32_t millis(void) {
48      return counter / 1000UL;
49  }
50
51  uint32_t micros(void) {
52      return counter;
53  }
```

10.2.15 time.h

```
/* Copyright (C) 2014 Kristian Lauszus, TKJ Electronics. All rights reserved.

 This software may be distributed and modified under the terms of the GNU
 General Public License version 2 (GPL2) as published by the Free Software
 Foundation and appearing in the file GPL2.TXT included in the packaging of
 this file. Please note that GPL2 Section 2[b] requires that all works based
 on this software must also be made publicly available under the terms of
 the GPL2 ("Copyleft").

 Contact information
 -------------------

 Kristian Lauszus, TKJ Electronics
 Web      :  http://www.tkjelectronics.com
 e-mail   :  kristianl@tkjelectronics.com
*/

#ifndef __time_h__
#define __time_h__

#ifdef __cplusplus
extern "C" {
#endif

void initTime(void);
void delay(uint32_t ms);
void delayMicroseconds(uint32_t us);
uint32_t millis(void);
uint32_t micros(void);

#ifdef __cplusplus
}
#endif

#endif
```

10.2.16 UART.c

```
/* Copyright (C) 2014 Kristian Lauszus, TKJ Electronics. All rights reserved.

 This software may be distributed and modified under the terms of the GNU
 General Public License version 2 (GPL2) as published by the Free Software
 Foundation and appearing in the file GPL2.TXT included in the packaging of
 this file. Please note that GPL2 Section 2[b] requires that all works based
 on this software must also be made publicly available under the terms of
 the GPL2 ("Copyleft").

 Contact information
 -------------------

 Kristian Lauszus, TKJ Electronics
 Web      :  http://www.tkjelectronics.com
 e-mail   :  kristianl@tkjelectronics.com
*/

#include <stdint.h>
#include <stdbool.h>
#include <stdlib.h>
#include <string.h>

#include "PID.h"
#include "time.h"
#include "UART.h"

#include "inc/hw_memmap.h"
#include "driverlib/gpio.h"
#include "driverlib/pin_map.h"
#include "driverlib/sysctl.h"
#include "utils/uartstdio.h" // Add "UART_BUFFERED" to preprocessor
#include "utils/ustdlib.h"

void initUART(void) {
```

```
35      // Enable the GPIO port containing the pins that will be used.
36      SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);
37      SysCtlDelay(2); // Insert a few cycles after enabling the peripheral to allow the
            clock to be fully activated
38
39      // Configure the GPIO pin muxing for the UART function.
40      // This is only necessary if your part supports GPIO pin function muxing.
41      // Study the data sheet to see which functions are allocated per pin.
42      GPIOPinConfigure(GPIO_PA0_U0RX);
43      GPIOPinConfigure(GPIO_PA1_U0TX);
44
45      // Since GPIO A0 and A1 are used for the UART function, they must be
46      // configured for use as a peripheral function (instead of GPIO).
47      GPIOPinTypeUART(GPIO_PORTA_BASE, GPIO_PIN_0 | GPIO_PIN_1);
48
49      UARTStdioConfig(0, 115200, SysCtlClockGet()); // Mode is set to 8N1 on UART0
50      UARTEchoSet(false);
51  }
52
53  void printPIDValues(void) {
54      UARTprintf("%d.%04d\t%d.%04d\t%d.%04d\n",
55                                              (int16_t)pidRoll.Kp, (int16_t)abs(pidRoll.Kp
                                                    * 10000.0f) % 10000,
56                                              (int16_t)pidRoll.Ki, (int16_t)abs(pidRoll.Ki
                                                    * 10000.0f) % 10000,
57                                              (int16_t)pidRoll.Kd, (int16_t)abs(pidRoll.Kd
                                                    * 10000.0f) % 10000);
58      UARTFlushTx(false);
59  }
60
61  void setValues(char *input) {
62      if (input[0] == 'G' && input[1] == 'P') // Send "GP;" to get the current PID Values
63          printPIDValues(); // Print PID Values
64      else if (input[0] == 'S' && input[2] == ',') { // Set different values
65          float value = ustrtof(input + 3, NULL); // Skip first three letters
66          if (input[1] == 'P')
67              pidRoll.Kp = value;
68          else if (input[1] == 'I')
69              pidRoll.Ki = value;
70          else if (input[1] == 'D')
71              pidRoll.Kd = value;
72
73          // Use same PID values for both pitch and roll
74          pidPitch.Kp = pidRoll.Kp;
75          pidPitch.Ki = pidRoll.Ki;
76          pidPitch.Kd = pidRoll.Kd;
77
78          pidYaw.Kp = pidRoll.Kp * 2.0f; // TODO: Tune these separately
79          pidYaw.Ki = pidRoll.Ki * 2.8f; // I increased this in order for it to stop yawing
                slowly
80          pidYaw.Kd = pidRoll.Kd * 2.0f;
81
82          printPIDValues(); // Print new PID Values
83      }
84  }
85
86  static char dataInput[100]; // Use this buffer to store the incoming values
87
88  void checkUARTData(void) {
89      if (UARTRxBytesAvail()) {
90          uint8_t i = 0;
91          while (1) {
92              dataInput[i] = UARTgetc(); // This is a blocking call
93              if (dataInput[i] == ';') // Keep reading until it reads a semicolon
94                  break;
95              if (++i >= sizeof(dataInput) / sizeof(dataInput[0]) - 1) // String is too
                    long
96                  return;
97          }
98          dataInput[i + 1] = '\0'; // Add null-character
99          UARTprintf("%s\n", dataInput); // Echo message back
100         setValues(dataInput);
101     }
102 }
```

10.2.17 UART.h

```
/* Copyright (C) 2014 Kristian Lauszus, TKJ Electronics. All rights reserved.

 This software may be distributed and modified under the terms of the GNU
 General Public License version 2 (GPL2) as published by the Free Software
 Foundation and appearing in the file GPL2.TXT included in the packaging of
 this file. Please note that GPL2 Section 2[b] requires that all works based
 on this software must also be made publicly available under the terms of
 the GPL2 ("Copyleft").

 Contact information
 -------------------

 Kristian Lauszus, TKJ Electronics
 Web      :  http://www.tkjelectronics.com
 e-mail   :  kristianl@tkjelectronics.com
*/

#ifndef __uart_h__
#define __uart_h__

#ifdef __cplusplus
extern "C" {
#endif

void initUART(void);
void checkUARTData(void);
void printPIDValues(void);

#ifdef __cplusplus
}
#endif

#endif
```

10.2.18 startup_rvmdk.S

```
; <<< Use Configuration Wizard in Context Menu >>>
;*******************************************************************************
;
; startup_rvmdk.S - Startup code for use with Keil's uVision.
;
; Copyright (c) 2012-2014 Texas Instruments Incorporated.  All rights reserved.
; Software License Agreement
;
; Texas Instruments (TI) is supplying this software for use solely and
; exclusively on TI's microcontroller products. The software is owned by
; TI and/or its suppliers, and is protected under applicable copyright
; laws. You may not combine this software with "viral" open-source
; software in order to form a larger program.
;
; THIS SOFTWARE IS PROVIDED "AS IS" AND WITH ALL FAULTS.
; NO WARRANTIES, WHETHER EXPRESS, IMPLIED OR STATUTORY, INCLUDING, BUT
; NOT LIMITED TO, IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
; A PARTICULAR PURPOSE APPLY TO THIS SOFTWARE. TI SHALL NOT, UNDER ANY
; CIRCUMSTANCES, BE LIABLE FOR SPECIAL, INCIDENTAL, OR CONSEQUENTIAL
; DAMAGES, FOR ANY REASON WHATSOEVER.
;
; This is part of revision 2.1.0.12573 of the EK-TM4C123GXL Firmware Package.
;
;*******************************************************************************

;*******************************************************************************
;
; <o> Stack Size (in Bytes) <0x0-0xFFFFFFFF:8>
;
;*******************************************************************************
Stack   EQU     0x00000100

;*******************************************************************************
;
; <o> Heap Size (in Bytes) <0x0-0xFFFFFFFF:8>
;
```

```
37 ;*******************************************************************************
38 Heap    EQU     0x00000000
39
40 ;*******************************************************************************
41 ;
42 ; Allocate space for the stack.
43 ;
44 ;*******************************************************************************
45         AREA    STACK, NOINIT, READWRITE, ALIGN=3
46 StackMem
47         SPACE   Stack
48 __initial_sp
49
50 ;*******************************************************************************
51 ;
52 ; Allocate space for the heap.
53 ;
54 ;*******************************************************************************
55         AREA    HEAP, NOINIT, READWRITE, ALIGN=3
56 __heap_base
57 HeapMem
58         SPACE   Heap
59 __heap_limit
60
61 ;*******************************************************************************
62 ;
63 ; Indicate that the code in this file preserves 8-byte alignment of the stack.
64 ;
65 ;*******************************************************************************
66         PRESERVE8
67
68 ;*******************************************************************************
69 ;
70 ; Place code into the reset code section.
71 ;
72 ;*******************************************************************************
73         AREA    RESET, CODE, READONLY
74         THUMB
75
76 ;*******************************************************************************
77 ;
78 ; External declarations for the interrupt handlers used by the application.
79 ;
80 ;*******************************************************************************
81                 EXTERN UARTStdioIntHandler
82         ;EXTERN SycTickHandler
83                 ;EXTERN Timer1Handler
84
85 ;*******************************************************************************
86 ;
87 ; The vector table.
88 ;
89 ;*******************************************************************************
90         EXPORT  __Vectors
91 __Vectors
92         DCD     StackMem + Stack            ; Top of Stack
93         DCD     Reset_Handler               ; Reset Handler
94         DCD     NmiSR                       ; NMI Handler
95         DCD     FaultISR                    ; Hard Fault Handler
96         DCD     IntDefaultHandler           ; The MPU fault handler
97         DCD     IntDefaultHandler           ; The bus fault handler
98         DCD     IntDefaultHandler           ; The usage fault handler
99         DCD     0                           ; Reserved
100        DCD     0                           ; Reserved
101        DCD     0                           ; Reserved
102        DCD     0                           ; Reserved
103        DCD     IntDefaultHandler           ; SVCall handler
104        DCD     IntDefaultHandler           ; Debug monitor handler
105        DCD     0                           ; Reserved
106        DCD     IntDefaultHandler           ; The PendSV handler
107        DCD     IntDefaultHandler           ; The SysTick handler
108        DCD     IntDefaultHandler           ; GPIO Port A
109        DCD     IntDefaultHandler           ; GPIO Port B
110        DCD     IntDefaultHandler           ; GPIO Port C
111        DCD     IntDefaultHandler           ; GPIO Port D
112        DCD     IntDefaultHandler           ; GPIO Port E
113        DCD     UARTStdioIntHandler         ; UART0 Rx and Tx
114        DCD     IntDefaultHandler           ; UART1 Rx and Tx
```

```
115         DCD     IntDefaultHandler           ; SSI0 Rx and Tx
116         DCD     IntDefaultHandler           ; I2C0 Master and Slave
117         DCD     IntDefaultHandler           ; PWM Fault
118         DCD     IntDefaultHandler           ; PWM Generator 0
119         DCD     IntDefaultHandler           ; PWM Generator 1
120         DCD     IntDefaultHandler           ; PWM Generator 2
121         DCD     IntDefaultHandler           ; Quadrature Encoder 0
122         DCD     IntDefaultHandler           ; ADC Sequence 0
123         DCD     IntDefaultHandler           ; ADC Sequence 1
124         DCD     IntDefaultHandler           ; ADC Sequence 2
125         DCD     IntDefaultHandler           ; ADC Sequence 3
126         DCD     IntDefaultHandler           ; Watchdog timer
127         DCD     IntDefaultHandler           ; Timer 0 subtimer A
128         DCD     IntDefaultHandler           ; Timer 0 subtimer B
129         DCD     IntDefaultHandler           ; Timer 1 subtimer A
130         DCD     IntDefaultHandler           ; Timer 1 subtimer B
131         DCD     IntDefaultHandler           ; Timer 2 subtimer A
132         DCD     IntDefaultHandler           ; Timer 2 subtimer B
133         DCD     IntDefaultHandler           ; Analog Comparator 0
134         DCD     IntDefaultHandler           ; Analog Comparator 1
135         DCD     IntDefaultHandler           ; Analog Comparator 2
136         DCD     IntDefaultHandler           ; System Control (PLL, OSC, BO)
137         DCD     IntDefaultHandler           ; FLASH Control
138         DCD     IntDefaultHandler           ; GPIO Port F
139         DCD     IntDefaultHandler           ; GPIO Port G
140         DCD     IntDefaultHandler           ; GPIO Port H
141         DCD     IntDefaultHandler           ; UART2 Rx and Tx
142         DCD     IntDefaultHandler           ; SSI1 Rx and Tx
143         DCD     IntDefaultHandler           ; Timer 3 subtimer A
144         DCD     IntDefaultHandler           ; Timer 3 subtimer B
145         DCD     IntDefaultHandler           ; I2C1 Master and Slave
146         DCD     IntDefaultHandler           ; Quadrature Encoder 1
147         DCD     IntDefaultHandler           ; CAN0
148         DCD     IntDefaultHandler           ; CAN1
149         DCD     0                           ; Reserved
150         DCD     0                           ; Reserved
151         DCD     IntDefaultHandler           ; Hibernate
152         DCD     IntDefaultHandler           ; USB0
153         DCD     IntDefaultHandler           ; PWM Generator 3
154         DCD     IntDefaultHandler           ; uDMA Software Transfer
155         DCD     IntDefaultHandler           ; uDMA Error
156         DCD     IntDefaultHandler           ; ADC1 Sequence 0
157         DCD     IntDefaultHandler           ; ADC1 Sequence 1
158         DCD     IntDefaultHandler           ; ADC1 Sequence 2
159         DCD     IntDefaultHandler           ; ADC1 Sequence 3
160         DCD     0                           ; Reserved
161         DCD     0                           ; Reserved
162         DCD     IntDefaultHandler           ; GPIO Port J
163         DCD     IntDefaultHandler           ; GPIO Port K
164         DCD     IntDefaultHandler           ; GPIO Port L
165         DCD     IntDefaultHandler           ; SSI2 Rx and Tx
166         DCD     IntDefaultHandler           ; SSI3 Rx and Tx
167         DCD     IntDefaultHandler           ; UART3 Rx and Tx
168         DCD     IntDefaultHandler           ; UART4 Rx and Tx
169         DCD     IntDefaultHandler           ; UART5 Rx and Tx
170         DCD     IntDefaultHandler           ; UART6 Rx and Tx
171         DCD     IntDefaultHandler           ; UART7 Rx and Tx
172         DCD     0                           ; Reserved
173         DCD     0                           ; Reserved
174         DCD     0                           ; Reserved
175         DCD     0                           ; Reserved
176         DCD     IntDefaultHandler           ; I2C2 Master and Slave
177         DCD     IntDefaultHandler           ; I2C3 Master and Slave
178         DCD     IntDefaultHandler           ; Timer 4 subtimer A
179         DCD     IntDefaultHandler           ; Timer 4 subtimer B
180         DCD     0                           ; Reserved
181         DCD     0                           ; Reserved
182         DCD     0                           ; Reserved
183         DCD     0                           ; Reserved
184         DCD     0                           ; Reserved
185         DCD     0                           ; Reserved
186         DCD     0                           ; Reserved
187         DCD     0                           ; Reserved
188         DCD     0                           ; Reserved
189         DCD     0                           ; Reserved
190         DCD     0                           ; Reserved
191         DCD     0                           ; Reserved
192         DCD     0                           ; Reserved
```

```
193         DCD     0                           ; Reserved
194         DCD     0                           ; Reserved
195         DCD     0                           ; Reserved
196         DCD     0                           ; Reserved
197         DCD     0                           ; Reserved
198         DCD     0                           ; Reserved
199         DCD     0                           ; Reserved
200         DCD     IntDefaultHandler           ; Timer 5 subtimer A
201         DCD     IntDefaultHandler           ; Timer 5 subtimer B
202         DCD     IntDefaultHandler           ; Wide Timer 0 subtimer A
203         DCD     IntDefaultHandler           ; Wide Timer 0 subtimer B
204         DCD     IntDefaultHandler           ; Wide Timer 1 subtimer A
205         DCD     IntDefaultHandler           ; Wide Timer 1 subtimer B
206         DCD     IntDefaultHandler           ; Wide Timer 2 subtimer A
207         DCD     IntDefaultHandler           ; Wide Timer 2 subtimer B
208         DCD     IntDefaultHandler           ; Wide Timer 3 subtimer A
209         DCD     IntDefaultHandler           ; Wide Timer 3 subtimer B
210         DCD     IntDefaultHandler           ; Wide Timer 4 subtimer A
211         DCD     IntDefaultHandler           ; Wide Timer 4 subtimer B
212         DCD     IntDefaultHandler           ; Wide Timer 5 subtimer A
213         DCD     IntDefaultHandler           ; Wide Timer 5 subtimer B
214         DCD     IntDefaultHandler           ; FPU
215         DCD     0                           ; Reserved
216         DCD     0                           ; Reserved
217         DCD     IntDefaultHandler           ; I2C4 Master and Slave
218         DCD     IntDefaultHandler           ; I2C5 Master and Slave
219         DCD     IntDefaultHandler           ; GPIO Port M
220         DCD     IntDefaultHandler           ; GPIO Port N
221         DCD     IntDefaultHandler           ; Quadrature Encoder 2
222         DCD     0                           ; Reserved
223         DCD     0                           ; Reserved
224         DCD     IntDefaultHandler           ; GPIO Port P (Summary or P0)
225         DCD     IntDefaultHandler           ; GPIO Port P1
226         DCD     IntDefaultHandler           ; GPIO Port P2
227         DCD     IntDefaultHandler           ; GPIO Port P3
228         DCD     IntDefaultHandler           ; GPIO Port P4
229         DCD     IntDefaultHandler           ; GPIO Port P5
230         DCD     IntDefaultHandler           ; GPIO Port P6
231         DCD     IntDefaultHandler           ; GPIO Port P7
232         DCD     IntDefaultHandler           ; GPIO Port Q (Summary or Q0)
233         DCD     IntDefaultHandler           ; GPIO Port Q1
234         DCD     IntDefaultHandler           ; GPIO Port Q2
235         DCD     IntDefaultHandler           ; GPIO Port Q3
236         DCD     IntDefaultHandler           ; GPIO Port Q4
237         DCD     IntDefaultHandler           ; GPIO Port Q5
238         DCD     IntDefaultHandler           ; GPIO Port Q6
239         DCD     IntDefaultHandler           ; GPIO Port Q7
240         DCD     IntDefaultHandler           ; GPIO Port R
241         DCD     IntDefaultHandler           ; GPIO Port S
242         DCD     IntDefaultHandler           ; PWM 1 Generator 0
243         DCD     IntDefaultHandler           ; PWM 1 Generator 1
244         DCD     IntDefaultHandler           ; PWM 1 Generator 2
245         DCD     IntDefaultHandler           ; PWM 1 Generator 3
246         DCD     IntDefaultHandler           ; PWM 1 Fault
247
248 ;*******************************************************************************
249 ;
250 ; This is the code that gets called when the processor first starts execution
251 ; following a reset event.
252 ;
253 ;*******************************************************************************
254         EXPORT  Reset_Handler
255 Reset_Handler
256         ;
257         ; Enable the floating-point unit.  This must be done here to handle the
258         ; case where main() uses floating-point and the function prologue saves
259         ; floating-point registers (which will fault if floating-point is not
260         ; enabled).  Any configuration of the floating-point unit using
261         ; DriverLib APIs must be done here prior to the floating-point unit
262         ; being enabled.
263         ;
264         ; Note that this does not use DriverLib since it might not be included
265         ; in this project.
266         ;
267         MOVW    R0, #0xED88
268         MOVT    R0, #0xE000
269         LDR     R1, [R0]
270         ORR     R1, #0x00F00000
```

```
271         STR     R1, [R0]
272
273         ;
274         ; Call the C library enty point that handles startup.  This will copy
275         ; the .data section initializers from flash to SRAM and zero fill the
276         ; .bss section.
277         ;
278         IMPORT  __main
279         B       __main
280
281 ;*******************************************************************************
282 ;
283 ; This is the code that gets called when the processor receives a NMI.  This
284 ; simply enters an infinite loop, preserving the system state for examination
285 ; by a debugger.
286 ;
287 ;*******************************************************************************
288 NmiSR
289         B       NmiSR
290
291 ;*******************************************************************************
292 ;
293 ; This is the code that gets called when the processor receives a fault
294 ; interrupt.  This simply enters an infinite loop, preserving the system state
295 ; for examination by a debugger.
296 ;
297 ;*******************************************************************************
298 FaultISR
299         B       FaultISR
300
301 ;*******************************************************************************
302 ;
303 ; This is the code that gets called when the processor receives an unexpected
304 ; interrupt.  This simply enters an infinite loop, preserving the system state
305 ; for examination by a debugger.
306 ;
307 ;*******************************************************************************
308 IntDefaultHandler
309         B       IntDefaultHandler
310
311 ;*******************************************************************************
312 ;
313 ; Make sure the end of this section is aligned.
314 ;
315 ;*******************************************************************************
316         ALIGN
317
318 ;*******************************************************************************
319 ;
320 ; Some code in the normal code section for initializing the heap and stack.
321 ;
322 ;*******************************************************************************
323         AREA    |.text|, CODE, READONLY
324
325 ;*******************************************************************************
326 ;
327 ; The function expected of the C library startup code for defining the stack
328 ; and heap memory locations.  For the C library version of the startup code,
329 ; provide this function so that the C library initialization code can find out
330 ; the location of the stack and heap.
331 ;
332 ;*******************************************************************************
333     IF :DEF: __MICROLIB
334         EXPORT  __initial_sp
335         EXPORT  __heap_base
336         EXPORT  __heap_limit
337     ELSE
338         IMPORT  __use_two_region_memory
339         EXPORT  __user_initial_stackheap
340 __user_initial_stackheap
341         LDR     R0, =HeapMem
342         LDR     R1, =(StackMem + Stack)
343         LDR     R2, =(HeapMem + Heap)
344         LDR     R3, =StackMem
345         BX      LR
346     ENDIF
347
348 ;*******************************************************************************
```

```
349 ;
350 ; Make sure the end of this section is aligned.
351 ;
352 ;*******************************************************************************
353         ALIGN
354
355 ;*******************************************************************************
356 ;
357 ; Tell the assembler that we're done.
358 ;
359 ;*******************************************************************************
360         END
```