```
#include <util/delay.h>

#ifndef F_CPU
#define F_CPU 12000000UL
#endif
#define BDRATE_BAUD 9600

#define MIN_SYNC_LEN 4000L
#define MIN_PULSE_LEN 800L
#define MAX_PULSE_LEN 2200L
#define NEUTRAL_PULSE_LEN 1500L
#define NEG_PULSE_LEN 200L
#define NORM_FRAME_LEN 22500L
#define SYNC_SPACER NORM_FRAME_LEN/5
#define MIN_FRAME_LEN NORM_FRAME_LEN-SYNC_SPACER
#define MAX_FRAME_LEN NORM_FRAME_LEN+SYNC_SPACER

#define NUM_PULSE_PER_FRAME 1
#define MIN_NUM_VALID_FRAMES 2 // number of valid frames required to switch back from NORC
#define MAX_NUM_INVALID_FRAMES 4 // number of invalid frames required to switch to NORC


uint16_t inPW[NUM_PULSE_PER_FRAME] ; // servo values from R/C receiver
uint16_t digitalData[NUM_PULSE_PER_FRAME]; // 5 bits control data
uint16_t neutralPW[NUM_PULSE_PER_FRAME]; // neutral servo values AKA central positions
uint16_t stat[NUM_PULSE_PER_FRAME];

uint16_t DecimalToBinary(const uint8_t Decnumber)
{
        uint16_t Result = 0;
        uint8_t i = 1;
        uint8_t tmp = Decnumber;
        while (tmp > 0) {
                Result = (tmp % 2)*i;
                i = i*10;
                tmp = (tmp/2) - (tmp%2);
        }
        return Result;
}
void init_stdio2uart0(void)
{
        // configure UART0 baud rate, one start bit, 8-bit, no parity and one stop bit
        UBRR0H = (F_CPU/(BDRATE_BAUD*16L)-1) >> 8;
        UBRR0L = (F_CPU/(BDRATE_BAUD*16L)-1);
        UCSR0B = _BV(RXEN0)|_BV(TXEN0);
        UCSR0C = _BV(UCSZ00)|_BV(UCSZ01);

}
void binaryoutput(const uint16_t Outputnumber)
{
        DDRA = 0xFF;
        uint8_t remainer = 0;
        uint16_t tmp = Outputnumber;
        while (tmp > 1) {
                remainer = tmp % 10;
                tmp = tmp/10 - remainer;
                if (remainer == 1) {
                        PORTA |= _BV(PINA1);
                        _delay_ms(1);
                } else {
                        PORTA |= ~_BV(PINA1);
                        _delay_ms(1);
```



```
                }
        }
}
void Transfer_numb(const uint16_t Controldata)
{
        // output the 5 bits data through UDR0
        while (!(UCSR0A & _BV(UDRE0)));
        UDR0 = Controldata;
}
static inline uint8_t ValidPulseLen (const uint16_t pulseLen)
{
        return (pulseLen > MIN_PULSE_LEN && pulseLen < MAX_PULSE_LEN);
}

int main(void)
{

        uint8_t i;
        uint16_t lastIcrTime = 0;
        uint8_t inPulseCount = 0;
        uint16_t inSyncTime = 0;
        uint16_t inFrameLen = 0;

        uint8_t validSync = 0;

        // Initiate the servo values with neutral values
        for (i=0;i< NUM_PULSE_PER_FRAME;i++) {
                inPW[i] = NEUTRAL_PULSE_LEN;
        }
        // Power and noise reduction
        PRR = _BV(PRTIM2)|_BV(PRTIM0)|_BV(PRSPI)|_BV(PRUSART0)|_BV(PRADC);

        DDRD = 0x00; //PD6 as input (ICP1)

        ICR1 = 0;

        TCCR1B = (0 << ICES1)|_BV(CS11); //prescaler 8, input capture according to PPM nega


        while (1) {
                if (TIFR1 & _BV(ICF1)) {
                        uint16_t currIcrTime = ICR1;
                        TIFR1 = _BV(ICF1);
                        uint16_t PulseLen = currIcrTime - lastIcrTime;
                        lastIcrTime = currIcrTime;
                        if ((PulseLen > MIN_SYNC_LEN) && (PulseLen < NORM_FRAME_LEN)) {
                                //sync detected
                                inFrameLen = currIcrTime - inSyncTime;
                                inSyncTime = currIcrTime;
                                validSync = 1;
                                inPulseCount = 0;
                        } else if (validSync && ValidPulseLen(PulseLen) && (inPulseCount <
                                inPW[inPulseCount++] = PulseLen;
                        } else {
                                validSync = 0;
                        }
                }

                for (i=0; i<NUM_PULSE_PER_FRAME;i++) {
                        digitalData[i] = DecimalToBinary((inPW[i]-800)/44);
                } // convert into 5 bits data
                for (i=0; i<NUM_PULSE_PER_FRAME;i++) {
```

```
                        Transfer_numb(digitalData[i]);
                } // sending 5-bit binary through UDR0
                for (i=0; i<NUM_PULSE_PER_FRAME;i++) {
                        binaryoutput(digitalData[i]);
                }

        }
}
```

5 Serial Telemetry Proccessor

```
/*

Harry Beadle
D4 Thames
Telemetry (telemetry.c)

UART0 Tx Bluetooth Host
UART0 Rx Bluetooth Host

UART1 Rx PPM Decoder (RC Data)

*/

#include "inc/telemetry.h"

ISR(USART0_RX_vect)
{
        // UART0 Rx from Bluetooth Host
        // Add control tuning data to control buffer.
        buffer8_add(&control_buffer, UDR0);
}

ISR(USART1_RX_vect)
{
        // UART1 RX from PPM Decoder
        // Add remote control data to output buffer.
        buffer8_add(&output_buffer, UDR1);
}

ISR(SPI_STC_vect)
{
        // Pull PB4 Low
        PORTB &= ~_BV(PB4);
        // Put Rx'd byte in rx buffer
        uint8_t spi_data = SPDR;
        if (spi_data) {
                buffer8_add(&spi_rx_buffer, SPDR);
        }
}

int main(void)
{
        // Initialise SPI
        DDRB = _BV(PB4) | _BV(PB5) | _BV(PB7);
        SPCR = _BV(SPE) | _BV(SPIE) | _BV(MSTR) | _BV(SPR0);

        // Initalise UART0 and UART1
        // Set Baud Rate.
        UBRR0H = (F_CPU/(BAUD*16L)-1) >> 8;
        UBRR1H = (F_CPU/(BAUD*16L)-1) >> 8;
        UBRR0L = (F_CPU/( BAUD*16L)-1);
        UBRR1L = (F_CPU/( BAUD*16L)-1);
        // Enable Tx, Rx, Tx Interupt, Rx Interupt.
        UCSR0B = _BV(RXEN0) | _BV(RXCIE0) | _BV(TXEN0) | _BV(TXCIE0);
        UCSR1B = _BV(RXEN1) | _BV(RXCIE1) | _BV(TXEN1) | _BV(TXCIE1);
        // 1 Start, 8 Data and 1 Stop bit.
        UCSR0C = _BV(UCSZ00) | _BV(UCSZ01);
        UCSR1C = _BV(UCSZ10) | _BV(UCSZ11);
        // Gloally Enable interrupts.
        sei();

        // Initalise the SPI device
        spi_write(WRITE & MPUREG_USER_CTRL, BIT_I2C_IF_DIS);
        spi_write(WRITE & MPUREG_INT_PIN_CFG, 0x22);
        spi_write(WRITE & MPUREG_PWR_MGMT_1, BIT_H_RESET);
        spi_write(WRITE & MPUREG_PWR_MGMT_1, 0x01);
        spi_write(READ & MPUREG_WHOAMI, 0x00);

        // Loop Forever
        while (1) {
                // Wait for the output buffer to be ready.
                if (buffer8_rdy(&output_buffer)) {
                        // Wait wile the UART is busy.
                        while (!(UCSR0A & _BV(UDRE0)));
                        // Output the contents of the buffer.
                        UDR0 = buffer8_pop(&output_buffer);
                }
                if (buffer8_rdy(&spi_rx_buffer)) {
                        while (!(UCSR0A & _BV(UDRE0)));
                        UDR0 = buffer8_pop(&spi_rx_buffer);
                }
                if (!(SPSR & _BV(SPIF)) && buffer8_rdy(&spi_tx_buffer)) {
                        PORTB |= _BV(PB4);
                        SPDR = buffer8_pop(&spi_tx_buffer);
                }
        }
}
```

6 MPU Decoder

```
/* MPU9250 Basic Example Code
 by: Kris Winer
 date: April 1, 2014
 license: Beerware - Use this code however you'd like. If you
 find it useful you can buy me a beer some time.
 Modified by Brent Wilkins July 19, 2016

 Demonstrate basic MPU-9250 functionality including parameterizing the register
 addresses, initializing the sensor, getting properly scaled accelerometer,
 gyroscope, and magnetometer data out. Addition of 9 DoF sensor fusion using open source
 Madgwick and Mahony filter algorithms. Sketch runs on the 3.3 V 8 MHz Pro Mini
 and the Teensy 3.1.

 SDA and SCL should have external pull-up resistors (to 3.3V).
 10k resistors are on the EMSENSR-9250 breakout board.

 Hardware setup:
 MPU9250 Breakout ------------ Arduino
 VDD ------------------------- 3.3V
 VDDI ------------------------ 3.3V
 SDA ------------------------- A4
 SCL ------------------------- A5
 GND ------------------------- GND
```

```
 */

#include "quaternionFilters.h"
#include "MPU9250.h"

#define AHRS true         // Set to false for basic data read
#define SerialDebug true  // Set to true to get Serial output for debugging

// Pin definitions
int intPin = 12;  // These can be changed, 2 and 3 are the Arduinos ext int pins
int myLed  = 13;  // Set up pin 13 led for toggling

MPU9250 myIMU;

void setup()
{
  Wire.begin();
  Serial.begin(57600);

  // Set up the interrupt pin, its set as active high, push-pull
  pinMode(intPin, INPUT);
  digitalWrite(intPin, LOW);
  pinMode(myLed, OUTPUT);
  digitalWrite(myLed, HIGH);


  // Read the WHO_AM_I register, this is a good test of communication
  byte c = myIMU.readByte(MPU9250_ADDRESS, WHO_AM_I_MPU9250);
  if (c == 0x73)
  {
    // Start by performing self test
    myIMU.MPU9250SelfTest(myIMU.SelfTest);

    // Calibrate gyro and accelerometers, load biases in bias registers
    myIMU.calibrateMPU9250(myIMU.gyroBias, myIMU.accelBias);

    myIMU.initMPU9250();
    // Initialize device for active mode read of acclerometer, gyroscope, and
    // temperature

    // Read the WHO_AM_I register of the magnetometer, this is a good test of
    // communication
    byte d = myIMU.readByte(AK8963_ADDRESS, WHO_AM_I_AK8963);

    // Get magnetometer calibration from AK8963 ROM
    myIMU.initAK8963(myIMU.magCalibration);

    // Initialize device for active mode read of magnetometer
    Serial.println("AK8963 initialized for active data mode....");


  }
  if (c == 0x73) //Check if communication is established
  else
  {
    while(1) ; // Loop forever if communication doesn't happen
  }
}

uint16_t pitch_send, roll_send, yaw_send;

void loop()
```



```
{
  // If intPin goes high, all data registers have new data
  // On interrupt, check if data ready interrupt
  if (myIMU.readByte(MPU9250_ADDRESS, INT_STATUS) & 0x01)
  {
    myIMU.readAccelData(myIMU.accelCount);  // Read the x/y/z adc values
    myIMU.getAres();

    // Now we'll calculate the accleration value into actual g's
    // This depends on scale being set
    myIMU.ax = (float)myIMU.accelCount[0]*myIMU.aRes; // - accelBias[0];
    myIMU.ay = (float)myIMU.accelCount[1]*myIMU.aRes; // - accelBias[1];
    myIMU.az = (float)myIMU.accelCount[2]*myIMU.aRes; // - accelBias[2];

    myIMU.readGyroData(myIMU.gyroCount);  // Read the x/y/z adc values
    myIMU.getGres();

    // Calculate the gyro value into actual degrees per second
    // This depends on scale being set
    myIMU.gx = (float)myIMU.gyroCount[0]*myIMU.gRes;
    myIMU.gy = (float)myIMU.gyroCount[1]*myIMU.gRes;
    myIMU.gz = (float)myIMU.gyroCount[2]*myIMU.gRes;

    myIMU.readMagData(myIMU.magCount);  // Read the x/y/z adc values
    myIMU.getMres();
    // User environmental x-axis correction in milliGauss, should be
    // automatically calculated
    myIMU.magbias[0] = +470.;
    // User environmental x-axis correction in milliGauss TODO axis??
    myIMU.magbias[1] = +120.;
    // User environmental x-axis correction in milliGauss
    myIMU.magbias[2] = +125.;

    // Calculate the magnetometer values in milliGauss
    // Include factory calibration per data sheet and user environmental
    // corrections
    // Get actual magnetometer value, this depends on scale being set
    myIMU.mx = (float)myIMU.magCount[0]*myIMU.mRes*myIMU.magCalibration[0] -
               myIMU.magbias[0];
    myIMU.my = (float)myIMU.magCount[1]*myIMU.mRes*myIMU.magCalibration[1] -
               myIMU.magbias[1];
    myIMU.mz = (float)myIMU.magCount[2]*myIMU.mRes*myIMU.magCalibration[2] -
               myIMU.magbias[2];
  } // if (readByte(MPU9250_ADDRESS, INT_STATUS) & 0x01)

  // Must be called before updating quaternions!
  myIMU.updateTime();

  // Sensors x (y)-axis of the accelerometer is aligned with the y (x)-axis of
  // the magnetometer; the magnetometer z-axis (+ down) is opposite to z-axis
  // (+ up) of accelerometer and gyro! We have to make some allowance for this
  // orientationmismatch in feeding the output to the quaternion filter. For the
  // MPU-9250, we have chosen a magnetic rotation that keeps the sensor forward
  // along the x-axis just like in the LSM9DS0 sensor. This rotation can be
  // modified to allow any convenient orientation convention. This is ok by
  // aircraft orientation standards! Pass gyro rate as rad/s
//  MadgwickQuaternionUpdate(ax, ay, az, gx*PI/180.0f, gy*PI/180.0f, gz*PI/180.0f,
my,  mx, mz);
  MahonyQuaternionUpdate(myIMU.ax, myIMU.ay, myIMU.az, myIMU.gx*DEG_TO_RAD,
                         myIMU.gy*DEG_TO_RAD, myIMU.gz*DEG_TO_RAD, myIMU.my,
                         myIMU.mx, myIMU.mz, myIMU.deltat);

```

```
147         myIMU.yaw   = atan2(2.0f * (*(getQ()+1) * *(getQ()+2) + *getQ() *
148                       *(getQ()+3)), *getQ() * *getQ() + *(getQ()+1) * *(getQ()+1)
149                       - *(getQ()+2) * *(getQ()+2) - *(getQ()+3) * *(getQ()+3));
150         myIMU.pitch = -asin(2.0f * (*(getQ()+1) * *(getQ()+3) - *getQ() *
151                       *(getQ()+2)));
152         myIMU.roll  = atan2(2.0f * (*getQ() * *(getQ()+1) + *(getQ()+2) *
153                       *(getQ()+3)), *getQ() * *getQ() - *(getQ()+1) * *(getQ()+1)
154                       - *(getQ()+2) * *(getQ()+2) + *(getQ()+3) * *(getQ()+3));
155         myIMU.pitch *= RAD_TO_DEG;
156         myIMU.yaw   *= RAD_TO_DEG;
157
158         myIMU.yaw   -= 8.5;
159         myIMU.roll  *= RAD_TO_DEG;
160
161         pitch_send = (uint16_t) map(myIMU.pitch, -180, 180, 0, 65536);
162         roll_send = (uint16_t) map(myIMU.roll, -180, 180, 0, 65536);
163         yaw_send = (uint16_t) map(myIMU.yaw, -180, 180, 0, 65536);
164
165         #define PITCH_S 0xF1
166         #define ROLL_S  0xF2
167         #define YAW_S   0xF3
168
169         Serial.write(PITCH_S);
170         Serial.write((char) pitch_send >> 8);
171         Serial.write((char) pitch_send);
172         Serial.write(ROLL_S);
173         Serial.write((char) roll_send >> 8);
174         Serial.write((char) roll_send);
175         Serial.write(YAW_S);
176         Serial.write((char) yaw_send >> 8);
177         Serial.write((char) yaw_send);
178
179         myIMU.count = millis();
180         myIMU.sumCount = 0;
181         myIMU.sum = 0;
182 }
```

APPENDIX E: Project Meeting Agendas & Minutes

D4 First meeting 23/2:

https://github.com/orgs/d4thames/projects/1

D4 Second meeting 24/2:

- How the RC talks to to mcu
- UART DUPLEXING
- Materials balsarwood (plexi is brittle)
- aliminum brackets
- Motor Mounting
- Landing Gear
- Max Size 80cm diameter
- more than 6-axis sensor
- power

"Quadcopter, also known as quadrotor, is a helicopter with four rotors.

The rotors are directed upwards and they are placed in a square formation with equal distance from the center of mass of the quadcopter."

hover thrust

where to put the ranger??? needs to work with and without load

---

27/2 Lab Session

-Sensors: Grove - IMU 9DOF v2.0

price: 10.99

accelerometer/gyro/mag/processing

I2C

http://ow.ly/hYvC309ogP0

Sharp GP2Y0A21YK0F Reflective Sensor

price: 7.10

analogue

http://ow.ly/sea8309ogMU

Total: 18.09

Design Clinic

- 1 person define interfaces between different modules
- memory
- instrumentation of the subsystems so that they can be exercised on their own
- number of processing cycles needed to create stability
- PWM to SPR
- digital receiver
- maybe preprocess sensor data

To do:

- figure out the size of data coming out of IMU SPI (done)
- syslog ppm to digital decoder
- check if there is internal clock on IMU (done)
- bill of Materials (done)

To do:

- Project proposal form
- Cost estimates (handbook p.25)
- Prototyping (mockups)

$$$ Limit Pitch Roll Yaw range to a number of degrees

Is there a way to calculate the load?

Check if atmga 32 and 644 have the same pinouts

Meeting Thur 2/3

- Friday
    - •Test PPM-Digital
    - •Test Control
- Tuesday

- Investigate R/C System !!!ME!!!

Geoff Meeting

- MPU- Filering

Power Circuit