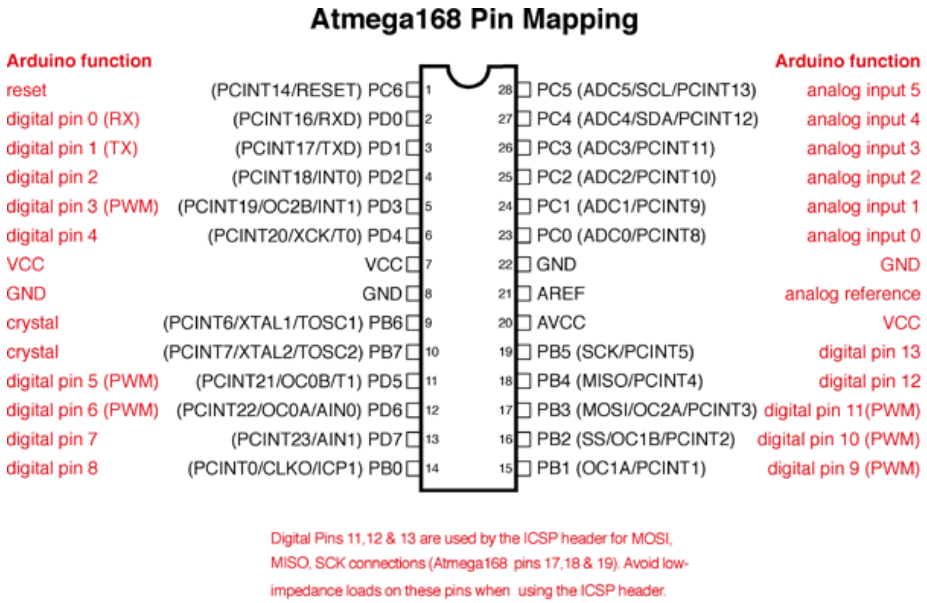
Setup code notes

- 2 `byte` -> the same as unsigned char (0 -> 255) 定义后自动初始化到0(`0b00000000`)
- 27 `volatile int` **volatile**它的作用是告诉编译器volatile变量是随时可能发生变化的，与volatile变量有关的运算，不要自作主张进行编译优化，以免出错，每次读取都从地址中获取。Volatile 意思是“易变的”，应该解释为“直接存取原始内存地址”比较合适。 C 语言书籍这样定义volatile关键字：volatile提醒编译器它后面所定义的变量随时都有可能改变，因此编译后的程序每次需要存储或读取这个变量的时候，告诉编译器对该变量不做优化，都会直接从变量内存地址中读取数据，从而可以提供对特殊地址的稳定访问。
- 41->45
  - `|=` operation：二进制位与运算
  - interruption programming: *Arduino Interrupt Tutorial | Microcontroller Tutorials (teachmemicro.com)*.
    - 外部中断
      - Uno板的终端引脚只有2，3
      - 用法：`attachInterrupt(digitalPinToInterrupt(interruptPin), function, CHANGE);`
      - The *attachInterrupt()* function has three parameters: first is the interrupt pin converted to Interrupt using the *digitalPinToInterrupt()* function, second is the *isr* function name（具体函数自己在后面定义，void类型，无参数）, and last is the mode which describes when the interrupt is triggered. Here, it’s set to CHANGE which means the interrupt is triggered when the pin changes its state. Other possible options are:
        - LOW to trigger the interrupt whenever the pin is low
        - RISING to trigger when the pin goes from low to high
        - FALLING for when the pin goes from high to low
    - Pin Change Interrupt
      - 引脚图



      - 更详细的参考：*Arduino – Pin Change Interrupt – Harry Gilbert (harry-gilbert.de)*

        - PCINT is short for pin change interrupt and it covers all pins from D0 to D13 and A0 to A5.
        - 控制原理：（本节所有引脚对应bit图示可以在上面网页*Arduino – Pin Change Interrupt – Harry Gilbert (harry-gilbert.de)*中找到）。所有PCINT引脚分为三组(PCIE, Pin Change Interrupt Enable Bit)，D8 to D13(PCIE0)，A0 to A5(PICE1)，D0 to D7(PCIE2)。每组只能提供一个中断信号。各组是否启用的信息存储在PCICR(Pin Change Interrupt Control Register)的后三个bit，从左到右对应PICE2，1，0。对于每个组，还可以进一步控制启用哪个引脚，PICE0,1,2组引脚的启用信息分别存储在PCMSK0,1,2中(Pin Change Enable Mask Register)，各PCMSK中各bit对应的引脚编号从右到左依次增大。因此启用某个引脚作为中断来源时，需要启用这个PICEx组(PCICR中)，并启用具体引脚(PCMSKx中)，两个操作谁先都可以。
        - 操作方法：（其中x是组编号，y是引脚编号（在引脚图中可以找到），实际使用时替换为数字）
            - 启用PICEx：`PCICR |= (1 << PCIEx);`
            - 启用特定引脚：`PCMSKx |= (1 << PCINTy)`
            - 写具体的中断函数(Interrup Service Routine)，并在参数中指明中断来源于哪个组（注意这里虽然写的是PCINT但后面的数字代表组数，即0-2）（上面的外部中断函数则不需要参数）

```
ISR(PCINTx_vect) {
// statements
        }
```

            - 网页*Arduino Interrupt Tutorial | Microcontroller Tutorials (teachmemicro.com)*末尾有一个更简单的方法可以参考

- `PINB` 代表端口B串口状态的寄存器。通过这种寄存器可以快速访问端口的信号。Arduino有3组端口，分类方式同上：B（数字引脚8到13），C（模拟输入引脚），D（数字引脚0到7）。三组分别对应`PINB,PINC,PIND`寄存器。每个寄存器内引脚从小到大对应寄存器位数从0增大。类似地，这三组引脚除了`PINx`还有`DDRx`和`PORTx`，它们分别代表端口输入/输出状态寄存器和端口输出信号寄存器。如`DDRD = B11111110; //将Arduino引脚1至7设置为输出，将引脚0设置为输入`，`PORTD = B10101000; // sets digital pins 7,5,3 HIGH`。详见这个网页（*Arduino板上通过操作端口寄存器来进行控制*）。

- 55 `Serial.print(F(''))`：用`F('')`可以把字符串存储在存储器（FLASH）而不是RAM中，以节省空间

- 63 TWBR：设置两线串行接口速度（波特率？）

- 65 `#if`：预处理语句，若条件为真才对其后的语句进行编译，否则其后的语句不参与编译

- 85 `void wait_for_receiver()` ：

    - 在ISR中给`last_channel_x`赋值了上一次的电平高/低
    - 作用：确认各通道的RC信号都在900-2100之间，最长等待10秒。若超过十秒，且没有任何通道有信号，则输出`No valid receiver signals found!!! (ERROR 1)`。不过若超时但至少有一个通道有信号，似乎不会报错。

- 90-104：等待10秒后存储各操纵杆中心位置(center position)

- 793 `ISR(PCINT0_vect)` This routine is called every time input 8, 9, 10 or 11 changed state. 读取每次高电平脉冲的宽度，并作为 `receiver_input_channel_x`.

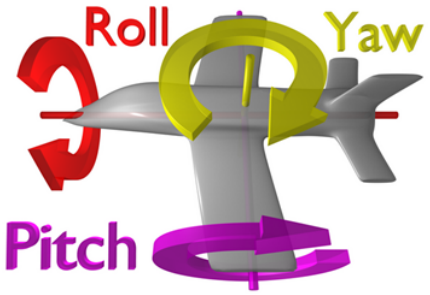
- 119-121 `void check_receiver_inputs(movement=1)`

  - Check if a receiver input value is changing within 30 seconds. 检查是否有输入在30秒内达到1750以上或1250以下。
  - `movment == 1` 时检测油门信号来源。此函数对 `channel_3_assign` 赋值，值为此时变化信号来源对应的通道。在主程序 121 行调用了此函数，并显示此时油门信号来源对应的引脚(Dx，不是channel几)
  - 左手油门遥控器的频道与摇杆对应图

- 124 `wait_sticks_zero()` 等待直到所有channel信号都回到center位置正负20范围内

- 130-140 `check_receiver_inputs(movement=2)` 检查Roll信号来源

  -



- 167 `void register_min_max()` 检测各通道信号的最低和最高值

  - 716-719：先把当前位置作为各通道信号最低值
  - 720：等待操作者移动操纵杆离开中心位置，这里只检测通道1（右操纵杆左右移动），如果操纵杆一直留在中心位置，就一直等待。
  - 721 开始测试时串口会显示 `"Measuring endpoints...."`
  - 722-734：记录整个过程中信号的最大值和最小值，直到所有操纵杆都回到中心。因此需要避免在测试完成前所有操纵杆回中，比较好的办法是让操纵杆画圈
  - 170-194：显示此过程中采集到的最小值-中心值-最大值

- 209-268 寻找陀螺仪(gyro)：用I2C协议查找了3种陀螺仪，分别是MPU-6050，L3G4200D，L3GD20H，对应`type`值1，2，3.

  - 503 `byte search_gyro(int gyro_address, int who_am_i)`：在地址`gyro_address` 上查找陀螺仪传感器，具体方法是发送`who_am_i`寄存器的地址（对MPU6050，`who_am_i`寄存器地址是`0x75`），等待传感器返回值。若在`gyro_address`上连接有MPU-6050，则返回的值总是`0x68`。所以可以以此判断`gyro_address`是否连接有MPU-6050，或其它什么传感器。函数总是将`gyro_address`的值赋给`address`

- 270-276：设置初始化陀螺仪

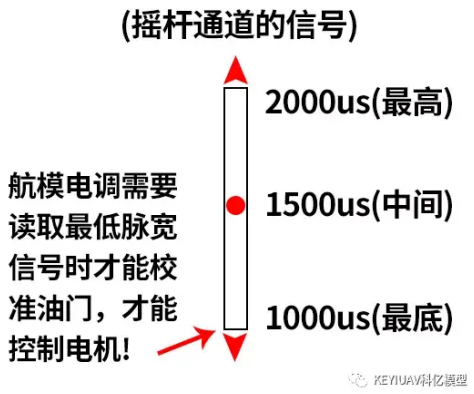
  - 515 `void start_gyro()`：只看546 MPU-6050的设置。
  - 548-551：初始化，复位，启动传感器。`0x6B` 表示 MPU-6050 的PWR_MGMT_1寄存器的编号，它用于设置电源模式和时钟源。其内容在技术手册*RS-MPU-6000A-00 (tdk.com)*40页上可以找到。向其中写入0可以复位传感器并启动。

    - 553-559：确认`0x6B`PWR_MGMT_1寄存器内的值
    - 561-564：在`0x1B`的GYRO_CONFIG寄存器设置陀螺仪量程。通过写入`0x08`设置其值为`00001000`,进而设置量程为$\pm 500^{\circ}/s$。
    - 566-572：确认`0x1B`的GYRO_CONFIG寄存器内的值。
- 281：开始校准陀螺仪
    - 294 -> 577 `gyro_signalen()`：只看597以下MPU-6050部分
    - 599：`0x43`是MPU-6050存储角度数据的第一个寄存器`GYRO_XOUT[15:8]`，每个轴的角度数据都有16位，一个寄存器存储8位（一个byte），总共由两个寄存器存储，因此所有角度数据存储在6个寄存器里。
    - 601：`Wire.requestFrom(address,6);`读取`0x43`及其以后共6个寄存器的值，即3个轴的角度数据。
    - 603-608：将各寄存器的值还原为角度数值。将第一个寄存器(高8位)向左移8位，再和第二个寄存器（低8位）做或位运算，可以还原出16位的整数信号。
    - 对`gyro_roll`,`gyro_pitch`,`gyro_yaw`赋值（若已完成校准，则会减去偏置再赋值）
- 292-303：重复读取各轴角度2000次，再求平均，作为偏置补偿`gyro_roll_cal`(cal->calibrate)
- 315-367：检测检测MPU6050的各轴对应飞机的哪些动作
    - `void check_gyro_axes()` 函数
- 368-378：检测LED是否正常
- 382-402：最后检查receiver和gyro的check byte，看有没有哪个通道/轴被遗漏
- 405-449：向EEPROM存储器中存储数据

Problem

RC相关

- 各摇杆对应的 `receiver_input` 值的范围是多少，
    - 下文已有解释
- `wait_for_receiver()` 里面900-2100的范围对应摇杆什么范围？油门最低位置有多大信号？最高位置呢？
    - 900-2100是只要有正常信号就一定会处于的范围。
- `check_receiver_inputs(movement)` 里面1750和1250对应什么样的油门位置？
    - 对应较低和较高的油门位置，但不是最低和最高
- 验证以下这张图



> - 油门杆范围在988-2000之间，右手频道3范围则在996-2000之间，若向下拨动遥控器左上角D/R（大小舵）开关，则右手频道3的范围到1250-1750之间。

- RC信号刷新率62.5Hz

Calibrate code notes

- 64-65 `DDRB,DDRC,DDRD` 控制B,C,D三组引脚输入/输出状态的寄存器。
- 67-71 设置中断（同setup）
- 73 读取EEPROM中的参数
- 77 `void set_gyro_registers()`设置MPU6050的参数，包括
  - 向PWR_MGMT_1（0x6B）中写入0以启用传感器
  - 向GYRO_CONFIG（0x1B）中写入0x08来设置陀螺仪 +/-500度/s 的量程
  - 向ACCEL_CONFIG（0x1C）中写入0x10来设置加速度计 +/- 8g的量程
  - 467-475 验证参数是否写入成功
    - 477-480 向CONFIG寄存器（0x1A）中写入0x03来设置43Hz的低通滤波
- 80-83 检验EEPROM存储数据是否正常，不正常则LED闪烁报警
- 84 测试RC接收器信号是否正常
- 85 设置计时0点
- 87 清空串口缓存区
- 93-94 等待4毫秒，时间清零
- 128 `convert_receiver_channel(byte function)`：将1500和setup中`center_channel_x`对准，把实际信号映射到1000-2000us之间
- 129 判断油门在不在最低位置，不在最低位置就翻转信号转换函数
- 135-142：把所有电机的转速都设置为油门杆信号
  - 141 `esc_pulse_output()` 向电调输出脉冲，长度代表转速
- vibration_counter是什么?
- 274-275，284-292没太看懂
- esc要求的刷新率250Hz。脉冲长度范围1000-2000us
- 电池12.6v约对应A0端口输入5v，`analogRead(A0)`为1023，还需要加上65作为二极管压降补偿，因此电源电压$V=(A_0+65)\times 1206/1023=(A_0+65)\times 1.2317$
- esc校准：Normally, you do not need to do throttle calibration for new multicopter ESCs. However, in some special cases, you still need to do it. Remove your props, and disconnect the battery. Push your radio throttle to maximum, connect the flight battery, wait for the 3 beeps(di da di), wait for 1s, you will hear a short beep(di). Then immediately lower the throttle to minimum. You will then hear another unique 2 beeps (di di), indicating you are setting the low value. If you hear a long beep, it means the low throttle set. Disconnect the battery, throttle calibration is done. If you hear a short beep, it means low throttle set failed. You need to do the calibration again. If still no success, pls contact us. (from *lynxmotion-simonk-esc-guide.pdf*)

Controller code notes

setup

- 76 读eeprom数据
- 77 start置0
- 78 读取gyro_address
- 80 打开I2C通讯
- 82 设置I2C时钟速度400KHz
- 85-86 设置输出引脚
- 89-96 点亮警报LED，检查eeprom中数据是否完整，是否连接MPU6050
- 98 `set_gyro_registers();` 初始化MPU6050
- 100-105 等待5秒，给esc输入频率250Hz，长度1000us脉冲，避免它发出警告声
- 108-123 校准MPU6050，消除（角速度）偏置
  - `gyro_signalen();` 从MPU6050中读取信号，从0x3B(59)号寄存器`ACCEL_XOUT[15:8]`开始读取，直到0x48(72)号寄存器`GYRO_ZOUT[7:0]`，读取加速度，温度，角速度数据。
- 125-129 设置用于接收RC信号的Pin Change Interrupt
- 132-145 若油门杆信号无效，或油门杆不在最低位置，或偏航杆在左侧，都闪烁LED报警，并等待
- 154 计算电池电压
- 156 设置循环计时器
- 159 关闭警告LED

loop

- 167-169 将gyro信号转换为gyro_input信号。为减小抖动，只加入新测量值的30%，加上原值的70%，构成新的gyro_input.
- 180-185：计算pitch和roll角度，但184-185不太理解
- 188-202 计算静止时加速度计的角度偏差（若飞机水平，则加速度应与z轴同向，通过计算加速度与z轴的偏移，可以确定飞机静止时的角度），并在角度中补偿
- 204-205 计算pitch和roll角度修正，但为什么要乘15？-->将角度转换为油门脉冲长度
- 214 在对最低油门，偏航杆左的情况下才启动
- 216-230 最低油门，偏航回中，启动
  - 第一次启动时，姿态角重置为重力偏角，重置PID参数
- 232 若启动后油门最低且右偏航，关闭电机

- 236-242 设置roll的pid输入信号

  - 238-239 若信号在1492-1508us，都会被视为0，让无人机静止时更稳定
  - 241 信号减去（近似）当前角度，避免动作过猛
  - 242 获得大约166dps的最大控制信号

- 247-253 设置pitch的pid输入信号

- 257-262 设置yaw的pid输入信号

- 264 计算pid输出信号

  - 453 `calculate_pid();` 分4步
  - step 1 计算姿态差别`pid_error_temp`：当前角速度`gyro_xxx_input`减去操纵杆输入`pid_xxx_setpoint`
  - step 2 计算积分信号`pid_i_mem_xxx`：用积分系数`pid_i_gain_xxx`乘当前姿态差别`pid_error_temp`，加到上次的积分信号上；若信号超过最大幅度`pid_max_xxx`，则将其限制为最大幅度
  - step 3 计算pid输出信号`pid_output_xxx`：输出信号等于比例信号`pid_p_gain_xxx * pid_error_temp`，加积分信号 `pid_i_mem_xxx`，加微分信号`pid_d_gain_xxx * (pid_error_temp - pid_last_xxx_d_error)`
  - step 4 重置微分中的上次姿态差别`pid_last_roll_d_error`：将当前姿态差别`pid_error_temp`赋值给上次姿态差别

- 269-272 获取电池电压，若电池电压低于10v，故障灯亮起

- 275 将修正后的油门信号传入`throttle`

- 278 控制最大油门为1800，给pid控制留出200us的空间

- 278 混合油门信号与pid修正。若某个姿态角速度大于pid输入值，则其pid输出为正。因此，要在有利于这种角速度的电机上减去这个pid输出，在不利于这种角速度的电机上加上这个pid输出。如：若俯仰角`pid_output_pitch`为正，则说明飞机过度后仰，因此要在前面的两个电机（1、4）上减去pid输出，在后面的两个电机（2，3）上加上pid输出

- 284 补偿电池压降带来的转速降低

- 291-294 设定esc的最小信号为1100，避免电机停转

- 296-299 设定esc最大信号为2000，避免超速

- 321 循环时间控制：若循环时间大于4050微秒，此时程序失效，点亮故障灯

- 325 等待，直到循环时间到达4000us

- 326 重设循环时间戳

- 328 向4各esc全部输出高信号

- 329-332 计算各esc需要输出的脉冲下降沿时间点

- 336 读取RC，陀螺仪，加速度数据

- 338 等待各esc的下降沿，将其信号设为低，结束循环

Problems

- [x] ~~34-37 gain变量名是yaw，但注释是pitch？~~
  - ~~pitch已在29-32行设置为与roll相同，34-37就是yaw的设置~~