最具创新性的设计理念奖

遥控器 4 通道改 9 通道

注册编号：MCHP16bitCDC0302

参赛队员：马　山

4 通道遥控器

9 通道遥控器

1 引言

模型飞机遥控器是有较高科技含量的产品，其价格随通道数增加而剧增，如以日本 FUTABA 品牌为例，4 通道（4 通道的意思见下面解释）T4VF 一般价格 650 元，见下面淘宝连接。

http://search1.taobao.com/browse/50008737/t-g,kq2fmrq----------40--commend-0-all-50008737.htm?at_topsearch=1

而 9 通道（9 通道的意思见下面解释）的 FF9 的一般价格为 3300 元。

http://search1.taobao.com/browse/50008737/t-g,izdds----------40--commend-0-all-50008737.htm?at_topsearch=1

可是 9 通道的比 4 通道的只是多了一些通道数和混控（意思见下面解释）功能而已。

（4 通道，9 通道：指遥控器可以同时控制的动作数量，4 通道一般可以同时控制油门，副翼，升降和左右 4 个动作，9 通道可以在以上基础在同时控制起落架，航拍相机快门，模拟投弹等等动作。）

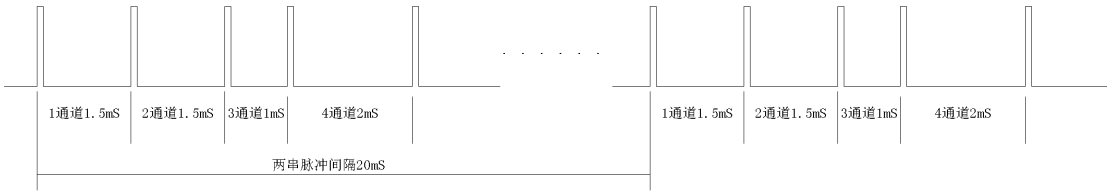
（混控：有一些特殊的飞机飞行时如果副翼动则升降也要跟着动，同样，升降动时副翼也需配合着动作，这样就叫副翼升降混控；同理还有副翼襟翼混控，副翼方向混控，油门升降混控等等，是由控制芯片完成混控的信号分配。）

本项目利用 dsPIC33FJ12MC202 完成 4 通道到 9 通道的功能，这样，模型爱好者可以只花 4 通道遥控器的价钱再加上一个 dsPIC 单片机的费用就得到了一个 9 通道遥控器的性能。

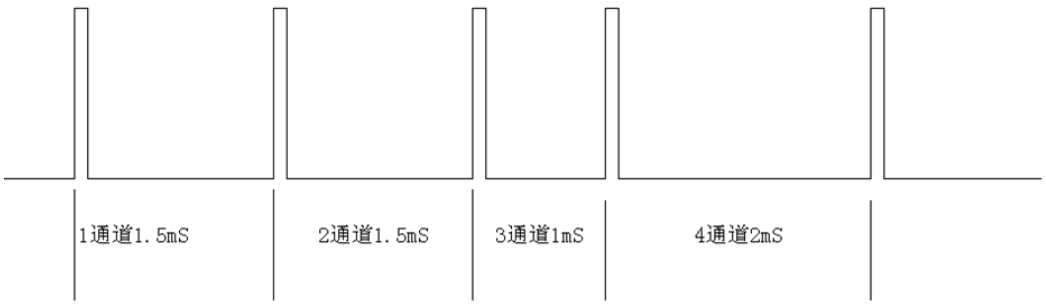
2 设计概述

2.1 通道数的电子学意义及在示波器上看到的波形：

遥控器的本质就是一个产生脉冲的电路加一个高频发射电路，高频发射电路在这里是不需要改动的，我们只要改动脉冲电路的部分，把他从 4 通道的脉冲加多为 9 通道的脉冲再送入高频发射电路即可，4 通道的脉冲波形如下所示

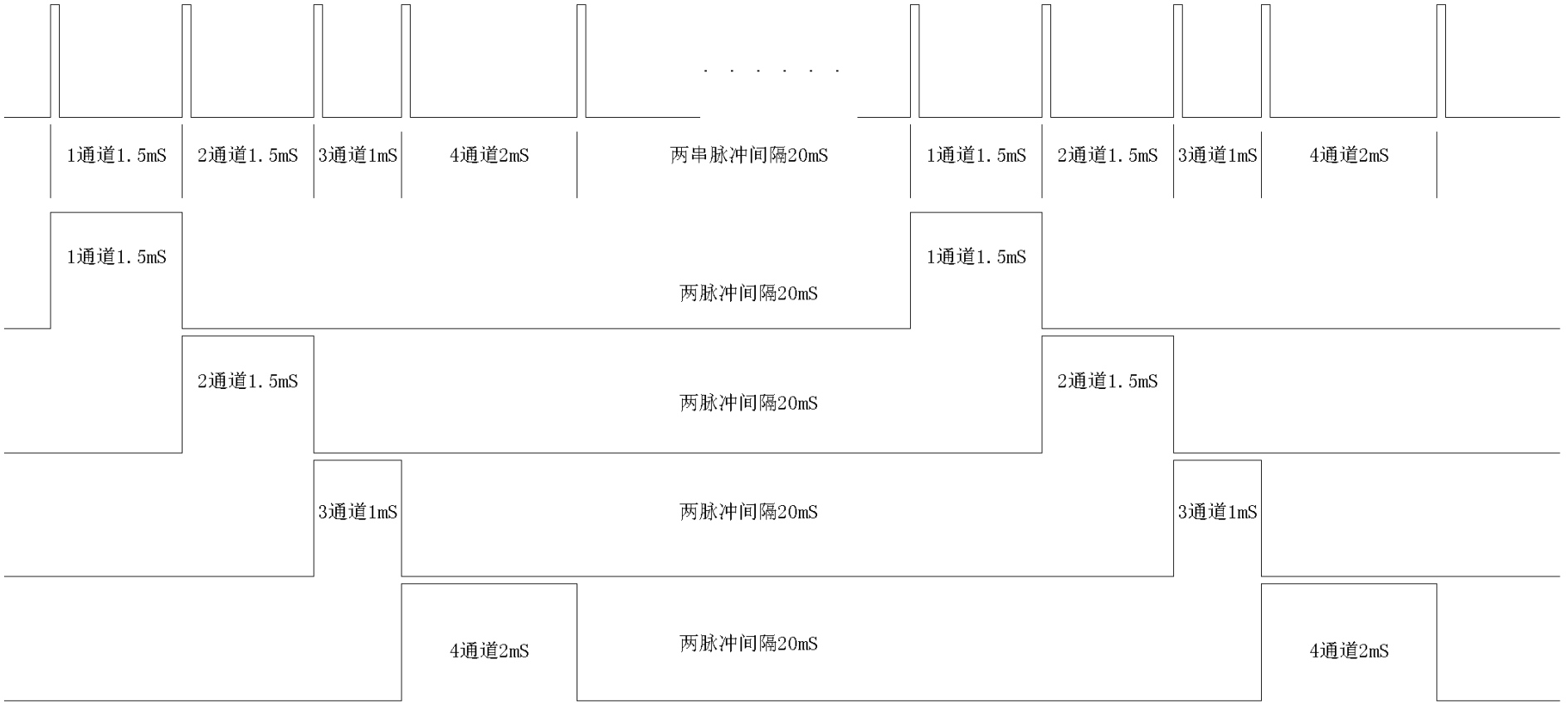


上图为两组 4 通道的脉冲串，每隔 20mS 有一串由 5 个脉冲组成的脉冲串，这个脉冲串可以直接送到发射电路发射，一个脉冲串放大出来如下图所示：

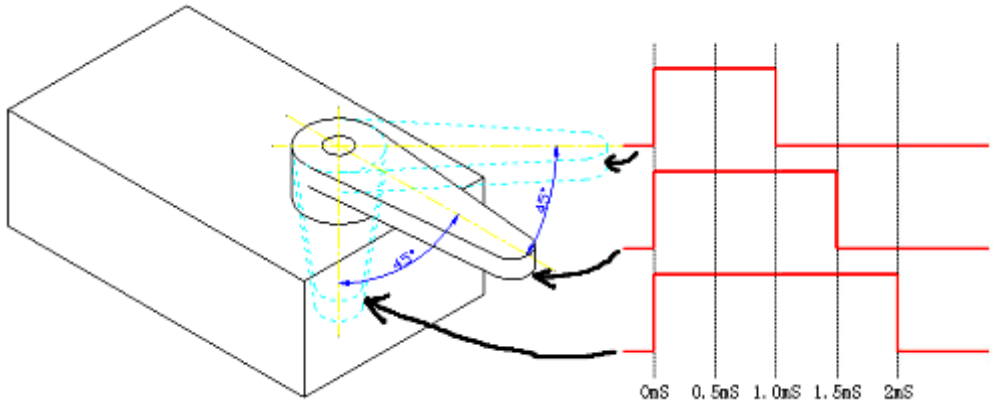


它携带的信号是：1 通道 1.5mS；2 通道 1.5mS；3 通道 1mS；4 通道 2mS。

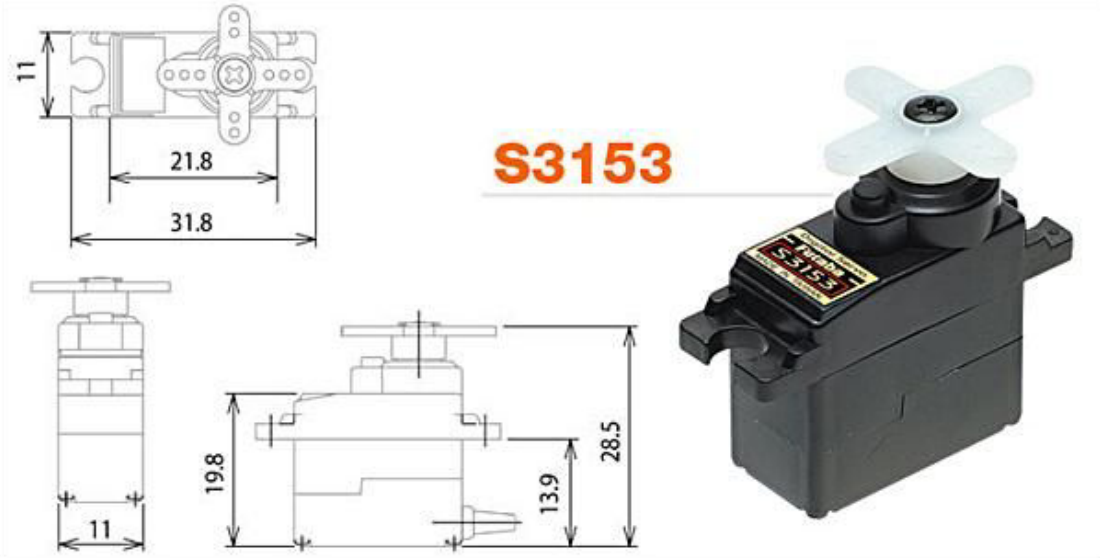
此信号在接收部分接收后送入解码电路把各个通道的脉冲提取出来送给舵机，舵机完成电信号到机械信号的转换，控制机构就可以按照电信号的要求动作了。

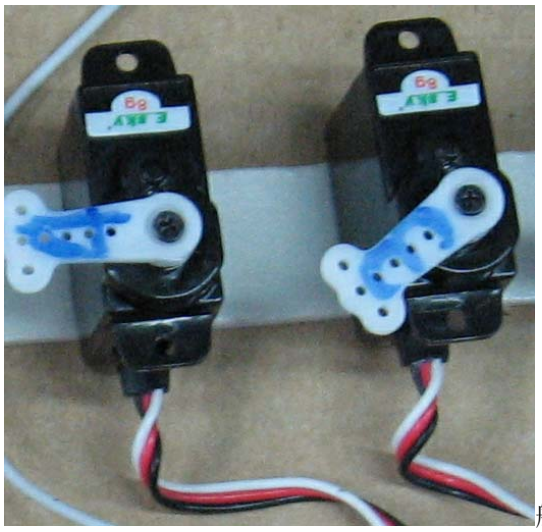

1通道1.5mS
2通道1.5mS
3通道1mS
4通道2mS
两串脉冲间隔20mS
1通道1.5mS
2通道1.5mS
3通道1mS
4通道2mS
1通道1.5mS
两脉冲间隔20mS
1通道1.5mS
2通道1.5mS
两脉冲间隔20mS
2通道1.5mS
3通道1mS
两脉冲间隔20mS
3通道1mS
4通道2mS
两脉冲间隔20mS
4通道2mS

1mS、1.5mS、2mS 的意义见下图，1ms 的脉冲使舵机摇臂在最左边的位置；1.5ms 的脉冲使舵机摇臂位于中位；2ms 的脉冲使舵机摇臂在最右边的位置；舵机的摇臂连接飞机的各执行机构，摇臂摇动时飞机的执行机构如尾翼等也跟着摆动。



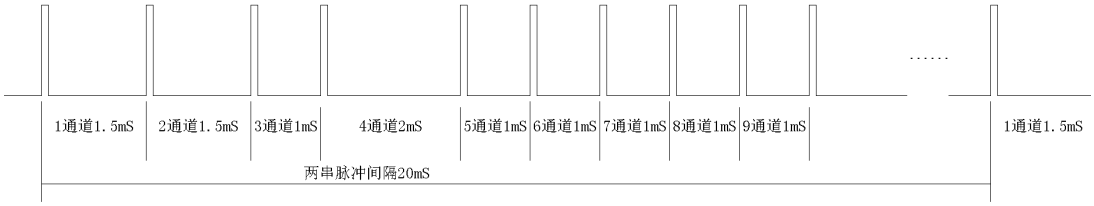
舵机三视图及照片见下（十字形摇臂）



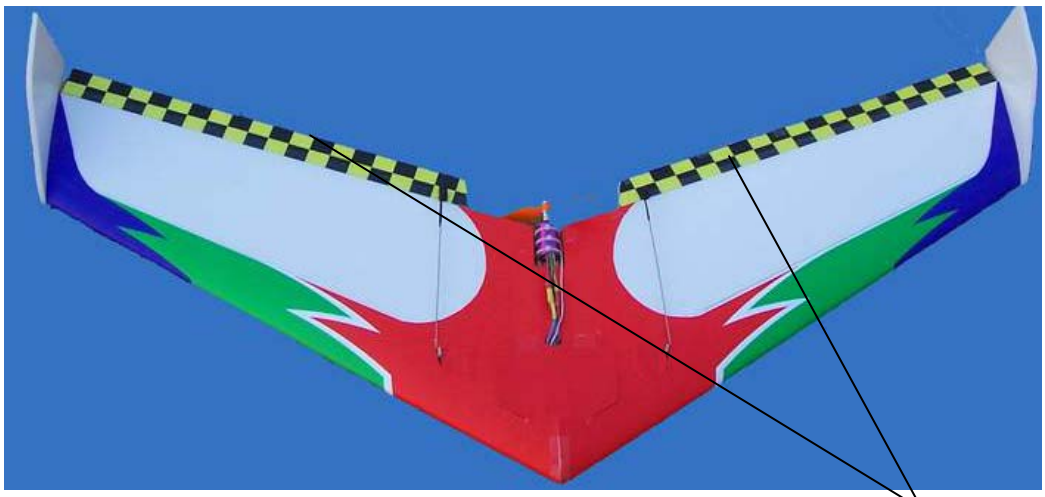


舵机照片（一字形摇臂，送的信号不同，舵机停留的位置不同）

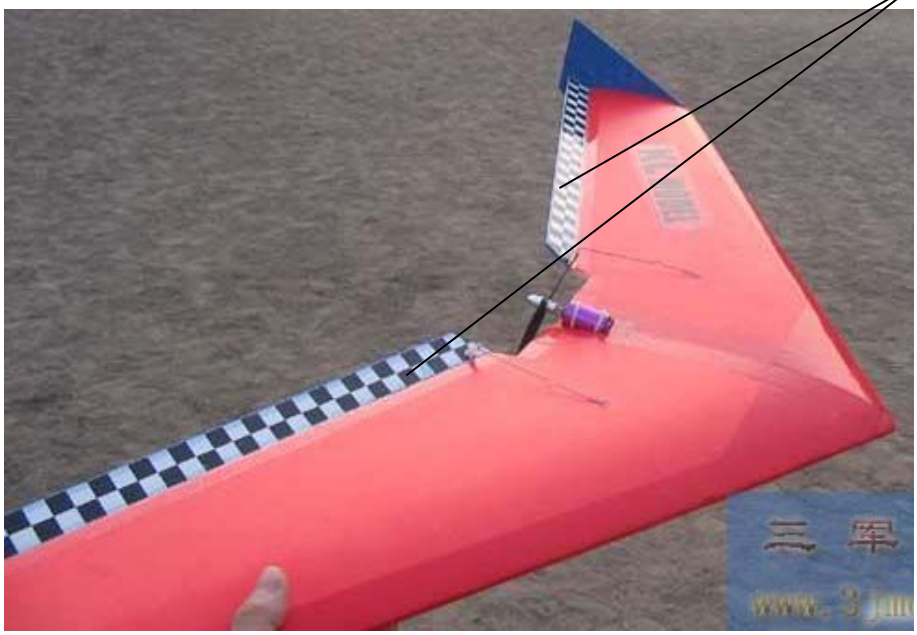
要改为 9 通道，只要把上面所讲的 包含有 4 通道信号的每隔 20mS 一串由 5 个脉冲组成的脉冲串改为 每隔 20mS 一串由 10 个脉冲组成的脉冲串 ，他就可以包含 9 个通道的信号了。



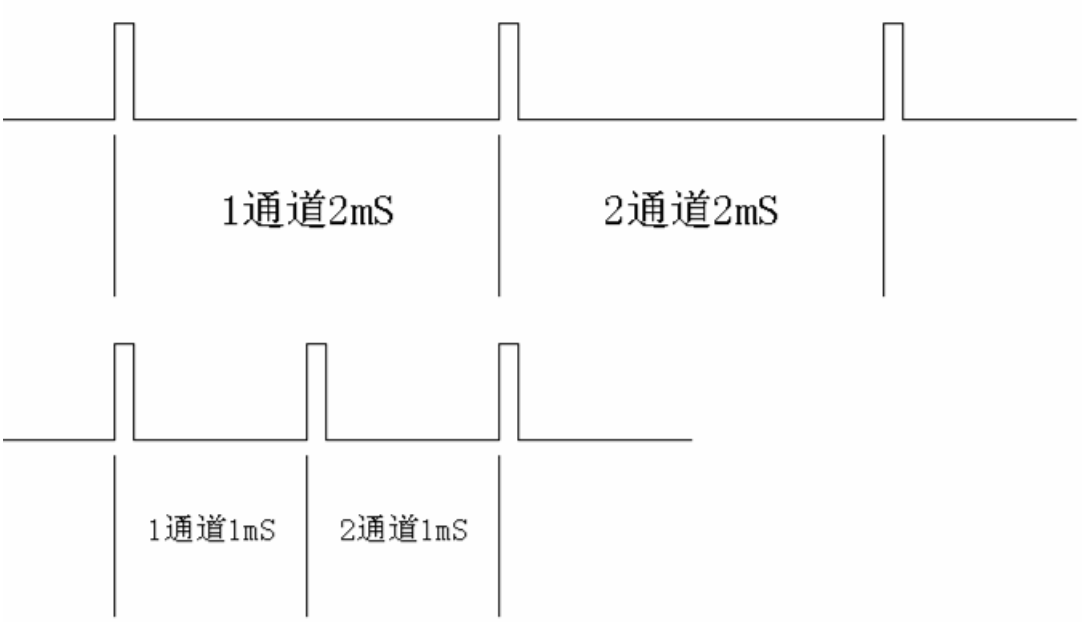
2.2 混控的意义及原理：对于一些非典型飞行器，如 F117 类飞机为飞翼类飞机，它的副翼与水平升降尾翼公用，这样就需要对副翼和升降混控，



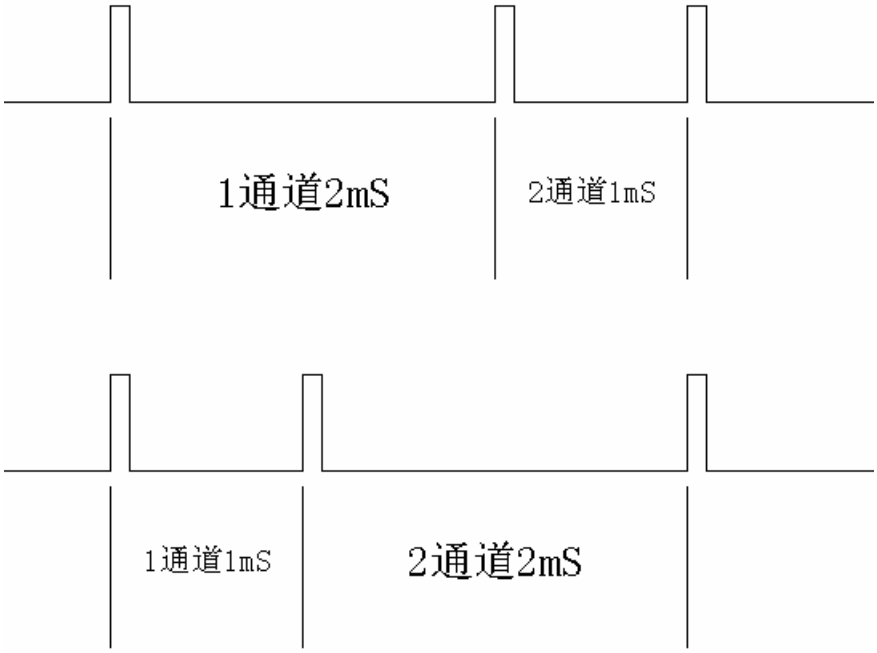
公用的副翼与水平升降尾翼



当需要升降时，遥控器 1 通道的副翼不动，2 通道的升降动作，则两翼向相同方向动作，反映在波形上是 1 通道的副翼与 2 通道的升降的脉宽一起变大变小，



当需要横滚时，遥控器 1 通道的副翼动作，2 通道的升降不动，则两翼向相反方向动作，一个向上一个向下，在波形上是 1 通道的副翼与 2 通道的升降的脉宽一个变大一个变小，



上述一个控制相同方向，一个控制相反方向，两个通道混合控制两个通道的方式叫做混控，如果两个通道分别是副翼和升降则叫做副翼升降混控，这种混控可以使飞机的副翼与升降翼合二为一；同理还有副翼襟翼混控用于飞机起降时增大升力和飞机巡航式减小阻力；副翼方向混控用于飞机转弯时实现协调转弯，内部人员只会感觉重力增加，不会有被离心力甩出去的感觉；油门升降混控用于飞机在空中加速时，此时速度增加则升力增加，飞机会上升，但我们只加了油门没有让飞机上升，这个混控可以在加油门时让飞机稍微低一点头，把刚才多产生的升力抵消一些，使飞机高度保持不变；以上为混控的意义。

2.3 大小舵的意义

大小舵是为切换专业玩家和初学者而设计的开关，飞机飞行时如果操作动作柔和则飞行的稳定，但观赏性不强；如果操作动作剧烈则飞行的不稳定，容易失速跌落，但观赏性强。这样就可以用大小舵开关切换，初学者用小舵动作柔和，专业玩家用大舵观赏性强。

2.4 实现原理

外部中断每个脉冲进入中断一次，并计算与上一个脉冲的间隔，这个间隔即为此通道的脉宽信号，此脉宽信号按标准应该在 1mS 到 2mS 之间，如果读到的脉宽大于 2mS 那就是开始下一串脉冲了，要更新每一个通道的脉宽值了。每一串信号应该有固定的个数，如果读到的信号个数有问题，那就是有干扰了，要把当前读到的一串脉宽丢掉，用中间值代替，但是油门通道送最低值，防止失控时飞机全油门大动力造成危险。

按上述方法把原有的 4 通道值读进来以后就可以按我们的要求增加其他通道了，这时要做两件事，1，增加 5 个通道；2，增加混控。增加 5 个通道，简单，找 4 个电位器接上，4 个模拟连续可调通道就有了，再找个开关，一个开关通道就来了，混控，就是软件做一些加减，只要装几个开关控制要不要混控就行。

大小舵的开关起作用时，就是把本应该送出去的脉宽的宽度按比例缩小后再送入脉冲串对应的位置。

3 硬件描述

3.1 读取遥控器信号的电路

电路由 J1-4，R3，Q2，R1，MCU-RB15 构成，J1 接来遥控器的 9V 信号经 Q2 转换为 3.3V 的电平送入单片机的 RB15，这里没有用电阻分压法是考虑到各厂牌遥控器的电压不同，用三级管做电平转换兼容性比较好。

3.2 送信号进遥控器

电路由 MCU-RB14，R4，Q1，R2，J1-3 构成，单片机送出的信号是 3.3V 电平，用 Q1 接遥控器的电源 VCC 把它做电平转换后送入遥控器。

3.3 4 个模拟连续可调通道

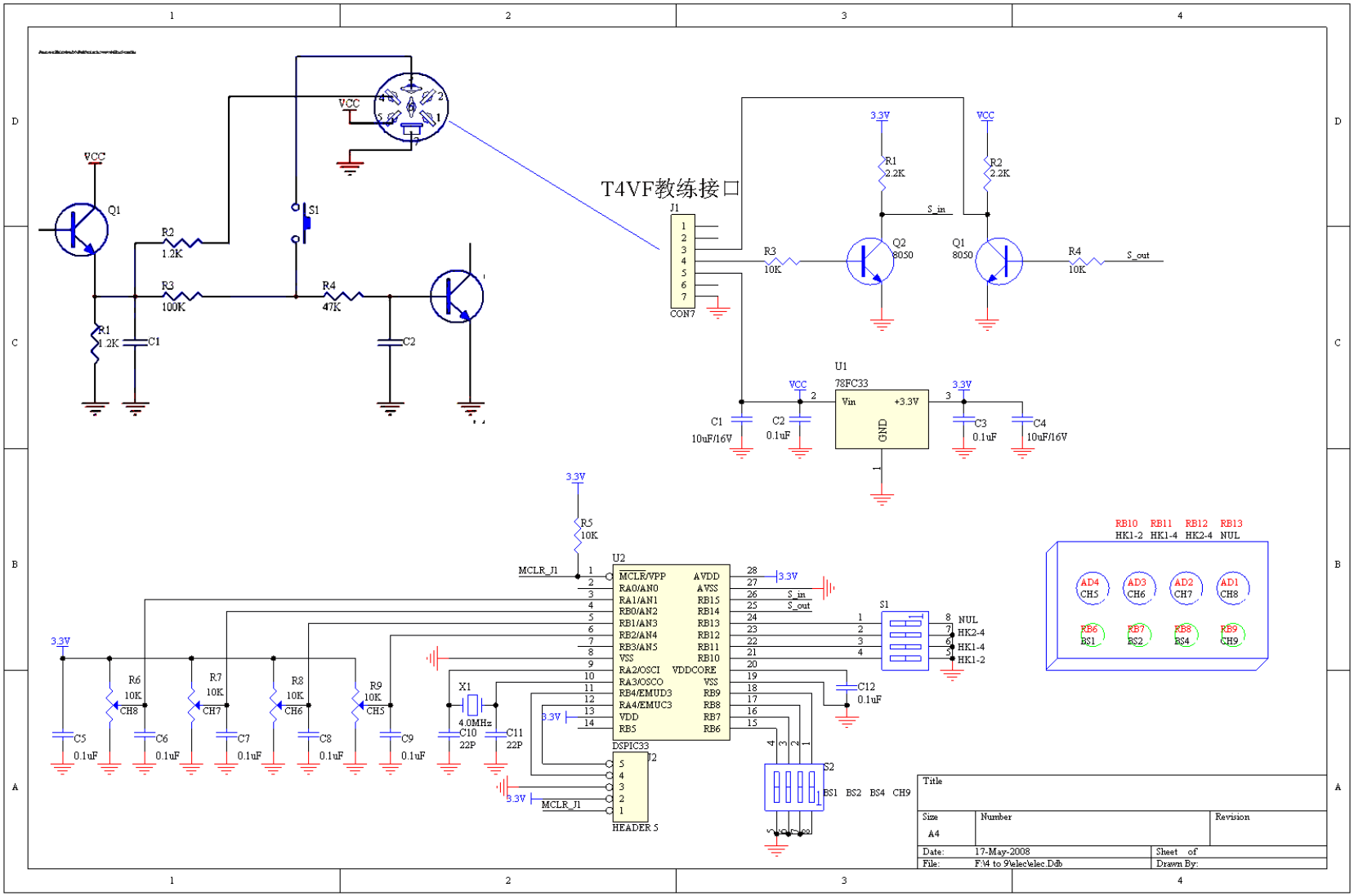
R6，R7，R8，R9，MCU-AN1，MCU-AN2，MCU-AN3，MCU-AN4，4 个电位器接在单片机的 4 个模拟输入，作 A/D 后放入脉冲串。

3.4 一个开关通道，混控开关和大小舵开关

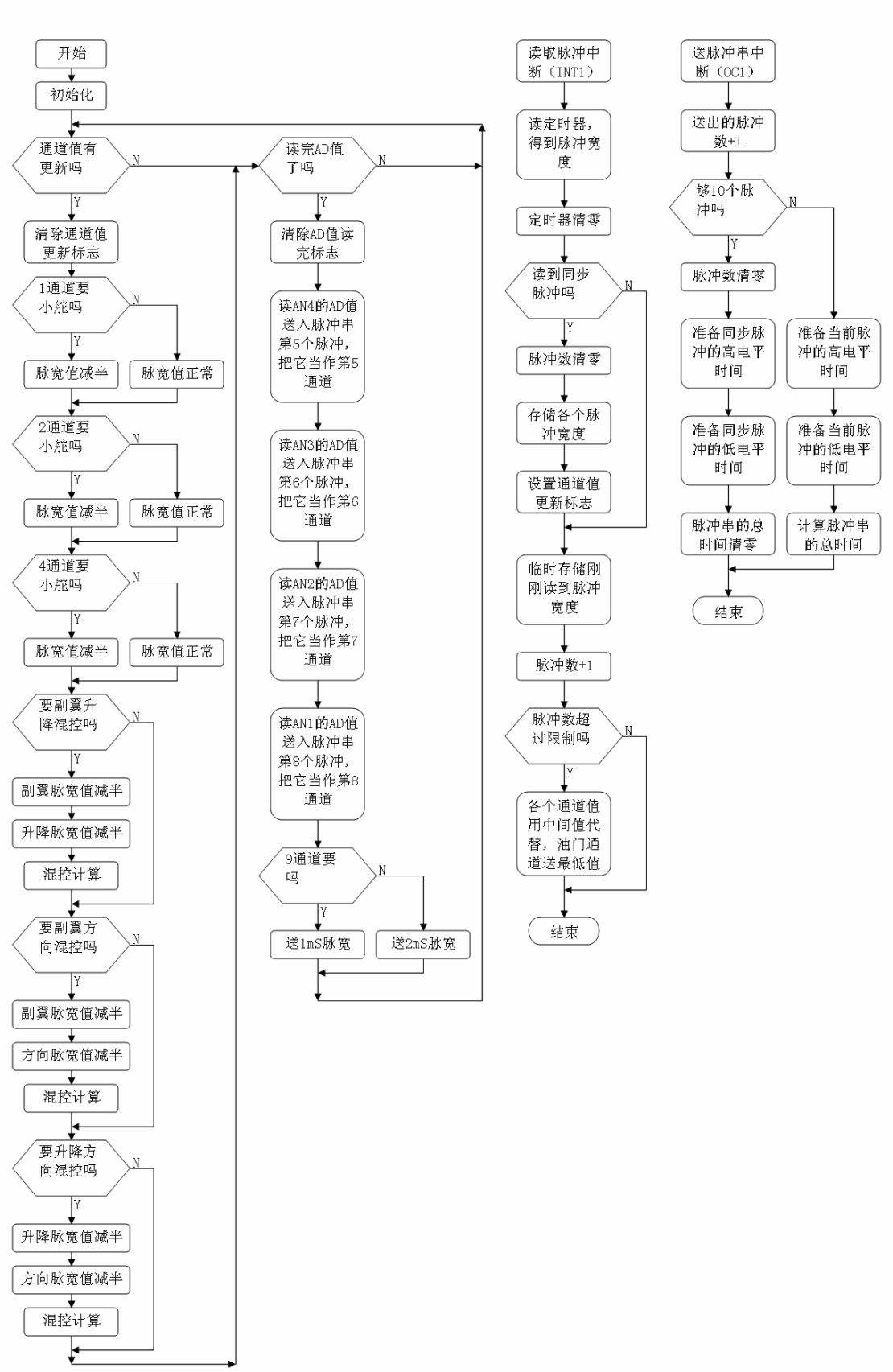
MCU-RB6，7，8，9，10，11，12 接的开关作为上述功能的切换信号。

3.5 遥控器内部的接口部分

电路图左上角的图是遥控器内部的接口部分，R12 为信号输出，R4 为信号输入，遥控器自身信号经 R3 送入 R4，我们的电路经过接口的 4 脚从 R12 取信号，经过处理后的信号经 S1 送入 R4（此信号内阻很低，原信号经 R3 过来后起不到作用）。


T4VF教练接口
VCC
Q1
R2
1.2K
R3
100K
R4
47K
R1
1.2K
C1
C2
S1
J1
CON7
3.3V
R1
2.2K
R2
2.2K
S_in
Q2
8050
Q1
8050
R3
10K
R4
10K
S_out
U1
78FC33
Vin
+3.3V
GND
C1
10uF/16V
C2
0.1uF
C3
0.1uF
C4
10uF/16V
R5
10K
MCLR_J1
U2
DSPIC33
MCLR/VPP
RA0/AN0
RA1/AN1
RB0/AN2
RB1/AN3
RB2/AN4
RB3/AN5
VSS
RA2/OSCI
RA3/OSCO
RB4/EMUD3
RA4/EMUC3
VDD
RB5
AVDD
AVSS
RB15
RB14
RB13
RB12
RB11
RB10
VDDCORE
VSS
RB9
RB8
RB7
RB6
R6
10K
CH8
R7
10K
CH7
R8
10K
CH6
R9
10K
CH5
C5
0.1uF
C6
0.1uF
C7
0.1uF
C8
0.1uF
C9
0.1uF
X1
4.0MHz
C10
22P
C11
22P
C12
0.1uF
S1
NUL
HK2-4
HK1-4
HK1-2
RB10 RB11 RB12 RB13
HK1-2 HK1-4 HK2-4 NUL
AD4 CH5
AD3 CH6
AD2 CH7
AD1 CH8
RB6 BS1
RB7 BS2
RB8 BS4
RB9 CH9
S2
BS1 BS2 BS4 CH9
HEADER 5
Title
Size
A4
Number
Revision
Date: 17-May-2008
Sheet of
File: F:\4 to 9\elec\elec.Ddb
Drawn By:

4 软件描述



4.1 读取遥控信号（INT1 中断）：每个脉冲进中断一次，进入后读取脉冲宽度并判断是否

同步脉冲（正常信号的宽度是 1mS-2mS，同步信号是一个大于 2mS 的脉宽），如果是同步脉冲，表示下一个脉冲串开始，各个脉宽要重新赋值，如果不是同步脉冲就表示这是脉冲串里的某一个脉冲，把它的脉宽送入对应的数组即可。

脉冲数的判断是为了预防干扰，有干扰时有几种处理方法，1：保持上次正确的数值不变；2：各个通道送入中间值；3：送入一个让飞机保持某一固定姿态的值，如让副翼保持水平，升降为水平略下，让飞机稍微低头，方向舵略左偏，让飞机左盘旋，油门为怠速稍高，使飞机低头慢慢左盘旋下降，回到遥控器信号有效的范围。本文使用的是第二种方法。

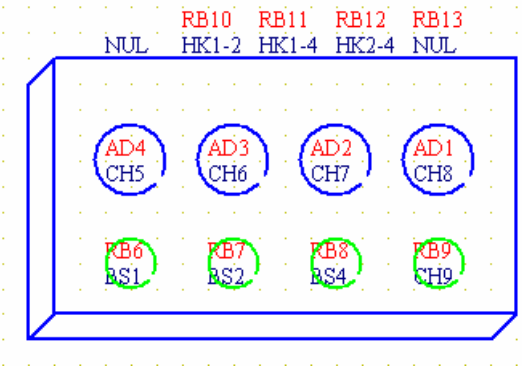
4.2 读取需要加的通道脉宽（A/D）：A/D 每 4 个一组，A/D 中断一次读取一个，读完 4 个后告诉主程序完毕，主程序取来数值放入脉冲串。

4.3 各个开关信号的处理（主程序）：主程序判断现在各个脉宽信号有没有读取完，如有，按照大小舵混控的开关要求运算，运算后数值放入脉冲串，同时读取第 9 通道的开关信号，转换为 1-2mS 的脉宽也送入脉冲串。

4.4 脉冲串的送出（输出比较中断）：脉冲串已经在数组里排好队，本中断里只要按照顺序把脉宽转成时间送出即可，需要注意的有两个地方：1，各个通道的脉宽在这个脉冲串里的表现是两个脉冲的间隔，和脉冲串里的脉宽是不同的概念，脉冲串里的脉宽是不需太计较的，本程序里用的宽度是 0.4mS；2，同步脉冲的宽度：计算各个脉宽的时间，加总后计算 离遥控器信号串间隔标准 20mS 还有多少，就把这个脉宽定位多少，因为它要>=2mS，每个通道都最大时，9 个通道 18mS 离 20mS 只剩 2mS，这个时间太短，是不能做同步脉冲的，所以使用本控制盒不能工作在所有通道都是最大值的情况，这是因为遥控器的标准导致的，解决的办法有减少通道数，增加 20mS 标准等，在此不作详细讨论。

5 操作说明

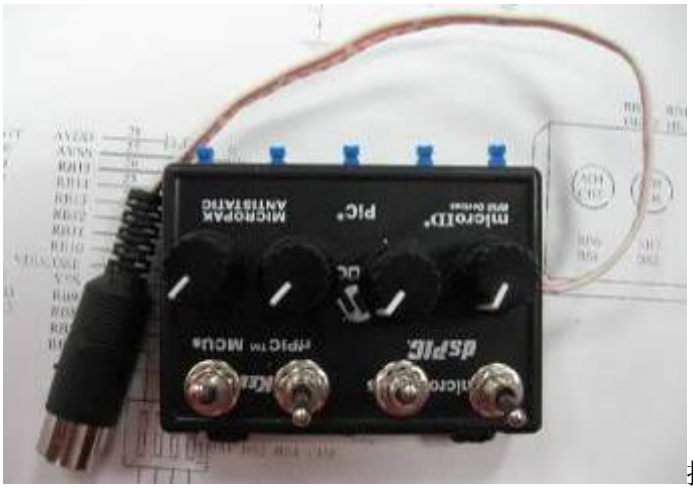
1 通道：副翼；2 通道：升降；3 通道：油门；4 通道：方向；其余通道按照飞机不同配置为不同的功能。



控制盒示意图

对照控制盒示意图看照片，上面一排 5 个蓝色按键的中间三个分别是 HK1-2（1-2

通道混控)；HK1-4（1-4 通道混控)；HK1-2（2-4 通道混控)，混控开关为按下有效；中间的四个旋钮从左至右分别为 CH5->5 通道；CH6->6 通道；CH7->7 通道和 CH8->8 通道，4 个通道旋钮逆时针减小，顺时针增大；下面的一排开关从左至右分别为 BS1->1 通道（副翼）大小舵；BS2->2 通道（方向）大小舵；BS4->通道（方向）大小舵和 CH9->第 9 通道开关，大小舵开关拨上为大舵，拨下为小舵；第 9 通道开关拨上打开，拨下关闭。伸出的一根插头是与遥控器连接用的。



控制盒照片


遥控器的背面

遥控器背后有一个 7 针接口，放大如下

接口特写


连接方法

把控制盒的插头插入遥控器的接口即可

安装完毕总观

安装后打开发射机电源，接通接收机电源，拨动遥控器 4 个通道的遥杆和 5，6，7，8 通道的旋钮及 9 通道的开关，可以看到接收机的九个舵机分别动作（因暂时没有买到支持 9 通道的接收机，所以只能看到八个通道），拨动大小舵开关后再拨动遥控器的遥杆可以看到舵量明显变小，按下混控开关，如 HK1-2，拨动一通道副翼（遥控器右边的遥杆左右拨动）可以看到一二通道的舵机动作相反，拨动二通道升降（遥控器左边的遥杆上下拨动）可以看到一二通道的舵机动作相同，分别测试各个混控后都没问题后，OK，带上你的飞机去爽飞吧！

6 结语

本项目实现的是一个低成本的 9 通道性能的 4 通道价格的遥控器，如能推广其优秀的性价比必可以得到广大航模爱好者的热情欢迎……

谢谢各位评委的细心阅读。

（备注：可以提供样品，包括：4 通道遥控器*1；控制盒*1；9 通道接收机*1；舵机*9）

7 附件（源程序）

```
#include "p33FJ12MC202.h"

#define ChanlNum 10 //9 个通道需 10 个脉冲
//-----------------------
//#define FCY 40,000,000              //Instruction Cycle Frequency
//#define BAUDRATE 115200
//#define BRGVAL ((FCY/BAUDRATE)/16)-1
//-----------------------
#define BS1 PORTBbits.RB6//1 通道大小舵
#define BS2 PORTBbits.RB7//2 通道大小舵
#define BS4 PORTBbits.RB8//4 通道大小舵
#define Ch9 PORTBbits.RB9//9 通道开关
#define H12 PORTBbits.RB10//副翼升降混控
#define H14 PORTBbits.RB11//副翼方向混控
#define H24 PORTBbits.RB12//升降方向混控
#define NUSE PORTBbits.RB13//备用的
//-----------------------
unsigned int ChVal[ChanlNum];//各个通道值
unsigned int ChValTmp[5];//各个通道值计算时的临时变量
unsigned int AdVal[5];//AD 值
//-----------------------
union{
  int FlagByte;
  struct{
```

```
            unsigned ChValCanRead:1;//通道值更新标志
            unsigned AdValCanRead:1;//AD 值读完标志
            unsigned :14;
  }bits;
}MyFlag;
//-----------------------
void InitClk(void);
void InitIO(void);
void InitTMR(void);
void InitADC12(void);
//-----------------------
int main(void)
{
  InitClk();
  InitIO();
  InitADC12();
  InitTMR();
//-----------------------
  IEC1bits.INT1IE = 1;
  IEC0bits.OC1IE = 1;
  IEC0bits.AD1IE = 1;
  AD1CON1bits.ADON = 1;
  INTCON1bits.NSTDIS = 0;
  _INT1IP = 7;//读取脉冲串中断 1 为 7
  _OC1IP = 6;//送出脉冲串中断 1 为 6
//-----------------------
  while(1)
  {
     if(MyFlag.bits.ChValCanRead)//读完一个脉冲串吗？（通道值有更新吗？）
     {
          MyFlag.bits.ChValCanRead = 0;//清除通道值更新标志
          if (BS1)//1 通道需要大小舵吗？
          {
               ChVal[1] = (ChValTmp[1] + 7500) / 2;//脉宽值减半
          }else
          {
               ChVal[1] = ChValTmp[1];//脉宽值正常
          }
```

```
        if (BS2)//2 通道需要大小舵吗？
        {
            ChVal[2] = (ChValTmp[2] + 7500) / 2;//脉宽值减半
        }else
        {
            ChVal[2] = ChValTmp[2];//脉宽值正常
        }
        if (BS4)//4 通道需要大小舵吗？
        {
            ChVal[4] = (ChValTmp[4] + 7500) / 2;//脉宽值减半
        }else
        {
            ChVal[4] = ChValTmp[4];//脉宽值正常
        }
        if (!H12)//需要副翼升降混控吗？
        {
            ChValTmp[1] = (ChValTmp[1] + 7500) / 2;//脉宽值先减半
            ChValTmp[2] = (ChValTmp[2] + 7500) / 2;//脉宽值先减半
            ChVal[1] = ChValTmp[1] + ChValTmp[2] - 7500;//混控计算-叠加
            ChVal[2] = ChValTmp[1] - ChValTmp[2] + 7500;//混控计算-减小
        }
        if (!H14)//需要副翼方向混控吗？
        {
            ChValTmp[1] = (ChValTmp[1] + 7500) / 2;//脉宽值先减半
            ChValTmp[4] = (ChValTmp[4] + 7500) / 2;//脉宽值先减半
            ChVal[1] = ChValTmp[1] + ChValTmp[4] - 7500;//混控计算-叠加
            ChVal[4] = ChValTmp[1] - ChValTmp[4] + 7500;//混控计算-减小
        }
        if (!H24)//需要升降方向混控吗？
        {
            ChValTmp[2] = (ChValTmp[2] + 7500) / 2;//脉宽值先减半
            ChValTmp[4] = (ChValTmp[4] + 7500) / 2;//脉宽值先减半
            ChVal[2] = ChValTmp[2] + ChValTmp[4] - 7500;//混控计算-叠加
            ChVal[4] = ChValTmp[2] - ChValTmp[4] + 7500;//混控计算-减小
        }
    }
```

```
        if(MyFlag.bits.AdValCanRead)//读完 AD 值了吗？
        {
            MyFlag.bits.AdValCanRead = 0;//清除 AD 值读完标志
            ChVal[5] = AdVal[4]*5/4 + 5000;//读 AN4 的 AD 值送入脉冲串第 5 个脉冲，把它当作第 5 通道
            ChVal[6] = AdVal[3]*5/4 + 5000;//读 AN3 的 AD 值送入脉冲串第 6 个脉冲，把它当作第 6 通道
            ChVal[7] = AdVal[2]*5/4 + 5000;//读 AN2 的 AD 值送入脉冲串第 7 个脉冲，把它当作第 7 通道
            ChVal[8] = AdVal[1]*5/4 + 5000;//读 AN1 的 AD 值送入脉冲串第 8 个脉冲，把它当作第 8 通道

            if (Ch9)//9 通道要吗？
            {
                ChVal[9] = 5000;//要，送 1mS 脉宽
            }else
            {
                ChVal[9] = 10000;//不要，送 2mS 脉宽
            }

        }
    }
    return 0;
}
//---------------------------------------------------------------------
//---------------------------------------------------------------------
void __attribute__((__interrupt__, no_auto_psv)) _OC1Interrupt(void)
{
    //这是送脉冲串的程序
    static unsigned char IintNum = 0;
    static unsigned long RunOnTime = 0;
    IFS0bits.OC1IF = 0;
    if (++IintNum >= 10)//够 10 个脉冲吗
    {
        IintNum = 0;//脉冲数清零
        TMR2 = 0;
        OC1R = (unsigned int)(100000 - RunOnTime);//送同步脉冲，脉宽为 20mS
```

```
（100000）- 脉冲串的时间加总
        OC1RS = OC1R + 2000;
        //OC1R = 63000;
        //OC1RS = 65000;
        RunOnTime = 0;  //重新计算时间
    }else
    {
        OC1R = ChVal[IintNum] - 2000;//按照脉冲串的时间 1 长短分别送脉冲出去
        OC1RS = ChVal[IintNum];
        TMR2 = 0;
        RunOnTime += OC1RS;//脉冲串的时间加总
    }
}
void __attribute__((__interrupt__, no_auto_psv)) _ADC1Interrupt(void)
{
    //这是读取几个模拟通道的 AD 值
    IFS0bits.AD1IF = 0;
    AdVal[AD1CHS0bits.CH0SA++] = ADC1BUF0;
    if (AD1CHS0bits.CH0SA >= 5)
    {
        AD1CHS0bits.CH0SA = 1;
        MyFlag.bits.AdValCanRead = 1;
        //__asm__ ("BTG PORTB,#5");
    }
}
void __attribute__((__interrupt__, no_auto_psv)) _INT1Interrupt(void)
{
    //这是读取各个脉冲宽度的程序
    unsigned int temp;
    static unsigned int RxdBuf[6];
    static unsigned char bytes;

    IFS1bits.INT1IF = 0;
    temp = TMR1;
    TMR1 = 0;
    if(temp > 15000)//读取的脉冲 >2.5ms 吗？（读到同步脉冲吗？）
    {
        bytes = 0;//重新计算读到的脉冲数
```

```
        ChValTmp[1] = RxdBuf[1];//刚刚读到脉冲宽度有效，存起来
        ChValTmp[2] = RxdBuf[2];
        ChVal[3] = RxdBuf[3];
        ChValTmp[4] = RxdBuf[4];
        MyFlag.bits.ChValCanRead = 1;//告诉主程序，读完了一个脉冲串吗，也就是通道值有更新了
    }
    RxdBuf[bytes++] = temp;//读脉冲宽度
    if(bytes > ChanlNum)//脉冲数超过限制吗？
    {
        bytes = ChanlNum;//超过限制，刚刚读到脉冲宽度无效
        ChVal[1] = 7500;//各个通道值用中间值代替
        ChVal[2] = 7500;//各个通道值用中间值代替
        ChVal[3] = 5000;//油门通道送最低值
        ChVal[4] = 7500;//各个通道值用中间值代替
    }
}
//以下为初始化程序，不多解释了
void InitClk(void)
{
    PLLFBD = 78;                    // M=80
    CLKDIVbits.PLLPOST=0;       // N1=2
    CLKDIVbits.PLLPRE=0;        // N2=2
    RCONbits.SWDTEN=0;
    while(OSCCONbits.LOCK!=1) {};
}
void InitIO(void)
{
    CNPU1bits.CN13PUE = 1;
    CNPU1bits.CN14PUE = 1;
    CNPU1bits.CN15PUE = 1;
    CNPU2bits.CN16PUE = 1;
    CNPU2bits.CN21PUE = 1;
    CNPU2bits.CN22PUE = 1;
    CNPU2bits.CN23PUE = 1;
    CNPU2bits.CN24PUE = 1;

    TRISB &= 0xBFD7;        //PIN25-RB14-OUT;PIN14-RB5-OUT;PIN7-RB3-OUT
```

```
  TRISB |= 0xBFC7;
//-----------------------
  RPINR0 = 0x0F00;          //INT1->RP15
  INTCON2bits.INT1EP = 0; //触发
  IFS1bits.INT1IF = 0;
  IEC1bits.INT1IE = 0;
//-----------------------
  RPOR7 = 0x0012;           //输出比较 1 ->RP14
  OC1CONbits.OCM2 = 1;  //脉冲
  OC1CONbits.OCM1 = 0;
  OC1CONbits.OCM0 = 1;
  OC1CONbits.OCTSEL = 0;
  OC1R = 63000;
  OC1RS = 65000;
  IFS0bits.OC1IF = 0;
  IEC0bits.OC1IE = 0;
}
void InitTMR(void)
{
  TMR1 = 0;                 //读外部中断 1 的定时
  PR1 = 0xFFFF;             // interrupt every 13.1072ms
  IFS0bits.T1IF = 0;        // clr interrupt flag
  IEC0bits.T1IE = 0;        // set interrupt enable bit
  T1CON = 0x8010;               // 1:8 prescale, start TMR1
//-----------------------
  TMR2 = 0;                 //输出比较，脉冲，的定时控制
  PR2 = 0xFFFF;
  IFS0bits.T2IF = 0;        // clr interrupt flag
  IEC0bits.T2IE = 0;        // set interrupt enable bit
  T2CON = 0x8010;               // 1:8 prescale, start TMR3
//-----------------------
  TMR3 = 0;                 //AD 转换的定时控制
  PR3 = 27500;              // interrupt every 5.5ms
  IFS0bits.T3IF = 0;        // clr interrupt flag
  IEC0bits.T3IE = 0;        // set interrupt enable bit
  T3CON = 0x8010;               // 1:8 prescale, start TMR3
}
void InitADC12(void)
```

```
{
  AD1CON1bits.FORM     = 0;        // Data Output Format: unsigned int
  AD1CON1bits.SSRC     = 2;        // Sample Clock Source: GP Timer starts
conversion
  AD1CON1bits.ASAM     = 1;        // ADC Sample Control: Sampling begins
immediately after conversion
  AD1CON1bits.AD12B    = 1;        // 12-bit ADC operation
  AD1CON3bits.ADRC = 0;       // ADC Clock is derived from Systems Clock
  AD1CON3bits.ADCS = 3;       // ADC Conversion Clock Tad=Tcy*(ADCS+1)=
(1/40M)*4 = 100ns
                              // ADC Conversion Time for 10-bit Tc=12*Tad = 1.2us

  AD1CON2bits.SMPI    = 0;    // SMPI must be 0
  AD1CHS0bits.CH0SA=1;        // MUXA +ve input selection (AN1) for CH0
  AD1CHS0bits.CH0NA=0;        // MUXA -ve input selection (Vref-) for CH0
  AD1PCFGL =0xFFFF;
  AD1PCFGLbits.PCFG1 = 0;         // AN1 as Analog Input
  AD1PCFGLbits.PCFG2 = 0;         // AN2 as Analog Input
  AD1PCFGLbits.PCFG3 = 0;         // AN3 as Analog Input
  AD1PCFGLbits.PCFG4 = 0;         // AN4 as Analog Input
  IFS0bits.AD1IF = 0;             // Clear the A/D interrupt flag bit
  IEC0bits.AD1IE = 0;             // Do Not Enable A/D interrupt
  AD1CON1bits.ADON = 0;       // Turn on the A/D converter
}
```