遗传算法入门实例：对 PID 参数寻优[原创]

[这乌龟飙得好快啊
http://sunxflower.blog.163.com]

开始之前：

假设你已经：能运用 C 语言，初步了解 PID、遗传算法的原理。

遗传算法能干什么？

（我有个毛病：每当遇到一个东东，我首先会设法知道：这个东东能干什么呢？）

遗传算法可以解决非线性、难以用数学描述的复杂问题。也许这样的陈述让你觉得很抽象，把它换成白话说就是：有个问题我不知道甚至不可能用数学的方法去推导、解算，那么也许我就可以用遗传算法来解决。遗传算法的优点是：**你不需要知道怎么去解决一个问题；**你需要知道的仅仅是，用怎么的方式对可行解进行编码，使得它能能被遗传算法机制所利用。

如果你运用过 PID 来控制某个系统，那你一定非常清楚：PID 麻烦就在那三个参量的调整上，很多介绍 PID 的书上常搬一些已知数学模型的系统来做实例环节，但事实上我们面对的往往是不可能用数学模型描述的系统，这个时候该怎么取 PID 的参值呢？

1、 可以依靠经验凑试，耗时耗精力。2、离线规划，这就是下文要做的事情 3、在线规划，比方说神经网络 PID（后续文章将推出，做个广告先^_^）。

一、 将 PID 用在本次试验中

来个问题先：AVR 怎样利用片上和少量的外围器件快速准确地实现 D/A 输出？（0~5V）

1、 实验电路的搭建：

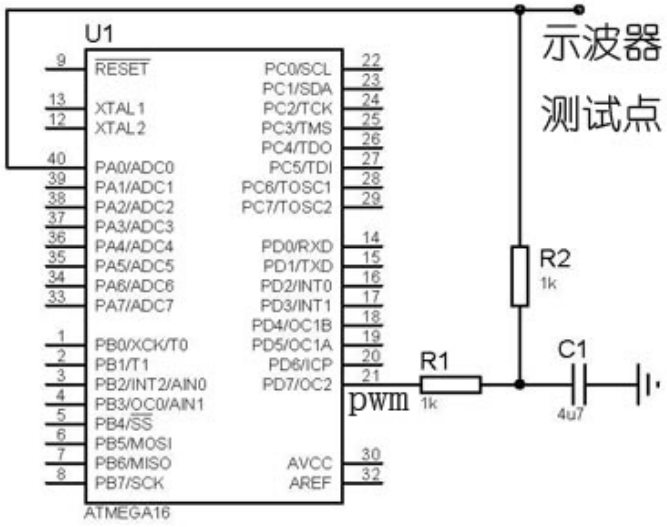
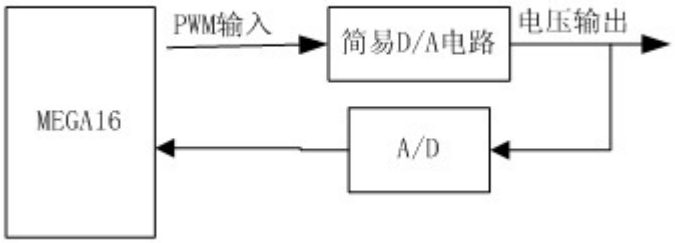


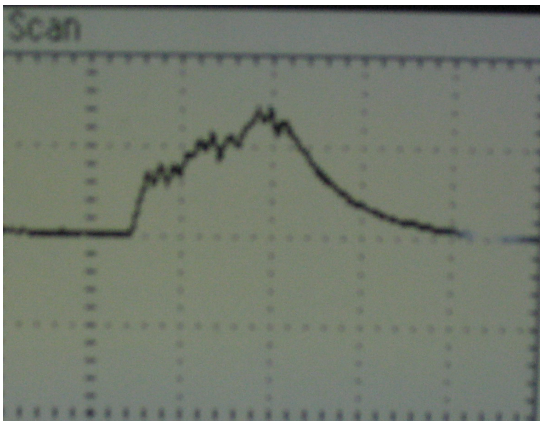
图 1 ：实验原理图

搭建这样的电路纯粹是为了本次实验的直观（超调、调整不足等现象通过示波器一目了然），当然，如果实际工程这么简单那也用不到 PID，更用不到遗传算法了。回归话题，解释下上面的电路：M16 单片机的 OC2 输出 0~100%占空比的 PWM，经过 RC，可以得到 0~5V 的直流电压，这就实现了简易的 D / A（实际实验，发现输出电压是 1.XX 伏~4.XX 伏，未带负载）。用一个图表示：

这个时候如果我要输出 3.5V（可以是其它值）电压，该加怎样的ＰＷＭ呢？（有个简单的方法：标定，但是这种方法系统调整响应速度较为缓慢，理由见图 5 下附言）也许我们可以把这个输出电压加到Ａ／Ｄ反馈到系统，这样就形成了闭环控制：系统输出 PWM——>>PWM 转换成电压——>>A/D 采集，获得实际值与目标值的偏差（例如 3.5V）——>>将偏差进行 PID 加载到 PWM 输出(然后输出又影响下一次的输入……)



把示波器加到测试点上，调整扫描周期，使示波器能看到完整的一个调整过程。这样，PID 调整的过程就可以在示波器上非常直观地显示出来。如下图：



２、PID 调整的过程（图 5）：

a)首先、让 OC2 输出一个初始电压（本次试验取 OCR2＝100，可以是其它值），当其稳定时进入ＰＩＤ调整环节

b)ＰＩＤ调整：采样输出电压，经过增量ＰＩＤ公式计算得到输入增量，将增量加到输入端，再次采样，继续下一次的调整。（这里有必要说明的是：数字 PID 是离散的，就是说，不能让程序 while(1)在那不停地执行，因为每次实验 while(1)的周期可能不同，这就会导致ＰＩＤ调整的周期是变化的，而这个变化会导致 ek、Δek 计算错误。正确的做法是设置一个时间变量（time_pid），用定时器来使其置位，而循环里就是查询这个变量的值，这样做虽然没有在定时中断里直接执行 PID 代码迅速，但推荐的做法还是像上面陈述的一样，虽然时间上可能误差几十个 us,但对于一个几百 ms 的 PID 周期来说这点误差还是可以接受的，另外，这个周期不能太短，比方说上次调整尚未完成，时间变量又已经置 1 了。）

对于本文的实验，如果 PID 参数较好，那调整 30 次早就已经达到了目标值，所以这个时候我们将系统复原：结束 PID 调整，并重新输出一个初始电压，开始下一次 PID 的过程。

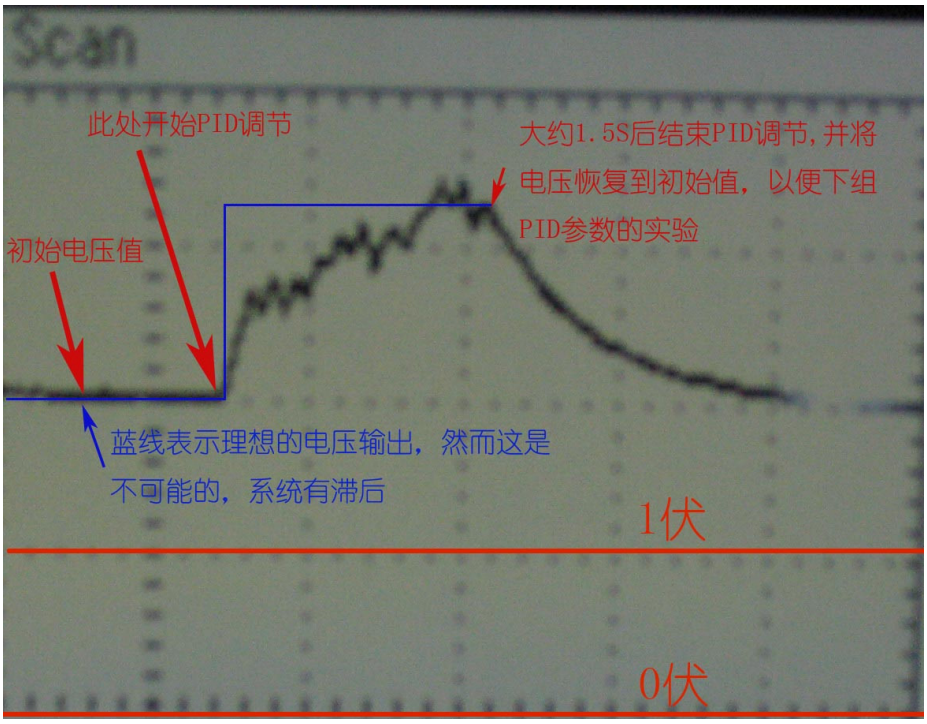
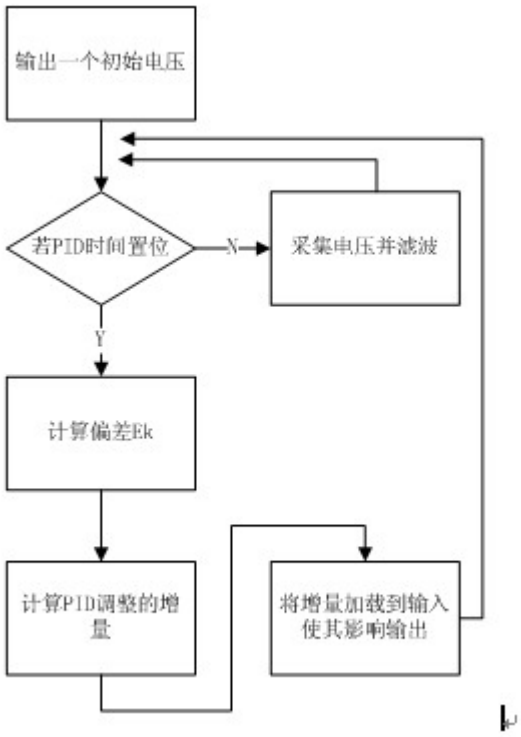


图 5：PID 调整的过程

另：见图 5，结束ＰＩＤ调整后是，电压是缓慢下降的，说明系统是滞后的，如果用标定的方法来实现Ｄ／Ａ，效果会像图中下降的那段曲线。

流程如下：



当改变ＰＩＤ的三个参量，曲线也会随之发生变化，我们可以多设置几组ＰＩＤ参数，看看超调是怎么样的，震荡又是怎么样的……

二、接下来让我们卷起袖子开始捣鼓遗传算法

1、介绍下遗传算法

如果你已经了解了遗传算法的基本原理，请跳过这段。

根据达尔文的自然选择学说，地球上的生物具有很强的繁殖能力。在繁殖过程中，大多数生物通过遗传，使物种保持相似的后代；部分生物由于变异，后代具有明显差别，甚至形成新物种。正是由于生物的不断繁殖后代，生物数目大量增加，而自然界中生物赖以生存的资源却是有限的。因此，为了生存，生物就需要竞争。生物在生存竞争中，根据对环境的适应能力，适者生存，不适者消亡。自然界中的生物，就是根据这种优胜劣汰的原则，不断地进行进化。

那么，到底怎样让这个算法在计算机里运作起来呢？或者说我们怎样模拟这个进化的过程呢？仔细阅读上面那段文字，发现需要解决以下问题：

A)、怎样表示一个个体？

B)、怎样繁殖后代？

C)、怎样实现优胜劣汰？

2、针对ＡＢＣ三个问题，用实际例子解答

Ａ）怎样表示一个个体——染色体编码和解码（采用二进制编码方法）

其实非常简单，一个变量即是一个个体，假如我们定义了一个个体名字叫 Joy (unsigned int Joy=12345;)

Joy 的基因为 12345，转换成二进制为 11000000111001，那我们大可以这样约定：Joy 的基因中前 2 个位表示了 Joy 的发色（比如 00 为黑色；01 为绿色），接着 3 个位表示了 Joy 的肤色，后 4 个位表示了 Joy 的眼睛颜色……相信你已经看出其间的奥秘了吧？

本文涉及 PID 三个参量:Kp,Ki,Kd ，我们也完全可以将 3 个参量理解成上面的特征（发色、肤色、眼睛颜色），将这些特征组装在一起就可以表示为一个个体， N 个个体组成一个种群……。具体怎么编呢：

Unsigned int colony=12332; //创造一个个体，并随机地赋予它某些特征。

（注：12332（十进制）＝０ ０ １ １ ０ ０　０ ０ ０ ０ １　０ １ １ ０ ０（二进制））

colony 的 0 ～ 4 位组成 kd，即 0 1 1 0 0（二进制）

5 ～ 9 位拿出来组成 ki，即 0 0 0 0 1（二进制）

１０～１５位组成 kp，０ ０ １ １ ０ ０（二进制）

然而这样还不行，因为这三个数都是整数，如果需要小数呢？缩放！

拿 Kd 来说，Kd 的取值范围是 0 ～ 1 1 1 1 1，如果 Kd 只需取小于 1 的小数，则将 Kd 除以 3 1（11111 的十进制），这样就得到了 0~1 的 Kd，如果想得到 0~2 的 Kd，则将 Kd 乘以 2 再除以 31。Kp,Ki 也是类似操作，只不过缩放程度不同。想必我们已经把怎么编码搞清楚了，事实上本例中我们没有必要写染色体编码的函数，只需知道染色体是怎么编的，进而知道怎么解码，初始化群体的时候只要随机产生一些数就好了。

解码举例：假如个体基因 62267，对应的 PID 参值多少呢？将其传递给 void redressal(unsigned int colon)函数，可知 Kp=19.047;Ki=0.80;Kd=0.87

看下解码的函数：

```
/*********************************************
*函数名:void redressal(unsigned int k_dip)  PID 参数解码
*说明：
    1、调用函数：
    2、变量说明：
        float ABC[3]  全局浮点变量，分别对应 Kp,Ki,Kd
```

```
        3、常量说明：
    *入口：unsigned int k_dip    传递个体染色体
    *出口：
    ********************************************/
        void redressal(unsigned int colon)
        {
        ABC[0]=(( colon&0xFC00)>>10)*20.0/63;
        ABC[1]=(( colon&0x3E0)>>5)/31.0;
        ABC[2]=( colon&0x1F)/31.0;
        }
```

我们只要稍作修改，就产生了一堆个体，即一个群体

```
        unsigned int colony[COL_MAX]={
    62267,15148,39769,31849,58411,49944,29915,58375 ,53831
    ,29144,40332,51900,60411,48378,11552,26588,61306,60089
    ,26887,58565, 3794,23125,53291,   646, 9102,13288,13023
    ,39570,17838,13029, 1001,48941,29169,61066,30539,27436
    ,55457,34416,13280,44049,54926, 1287,44647,24869,54512
    ,32952,46495,28107,19963,12429};// COL_MAX 是宏定义，定义种群的大小
```

这个数组即为初始化的群体，其中的某个元素即为一个个体，对应一组 PID 的参值。遗传算法就从这个种群开始，根据优胜劣汰的原则，不断地进行进化。

B）怎样繁殖后代——染色体杂交与变异

杂交：

记得高中生物有句话：杂种的优势~呵呵。言归正传，怎么将两个个体杂交得到新的个体呢？假如有个个体我们称之为父本(11110000),另一个个体我们称之为母本(10101010),假如他们发生了那个那个~~生了个小孩，如果交叉点在第 3 位和第 4 位间，那么得到新的个体是 11110010 和 10101000。一般这个杂交点选择基因长度的 0.7 左右，这里为了方便，就直接将一个 int 型剪成两个 char 型的交换了。定义父本（unsigned int dad）,母本（unsigned int mum）,杂交代码如下：

```
        baby1=dad&0xff;                    //取得低半截染色体
        baby1|=mum&0xff00;
        baby2=mum&0xff;
        baby2|=dad&0xff00;
```

变异：

若二进制编码的染色体发生变异，无非就是 0 变 1，1 变 0。其实变异是很少发生的事情，代码为函数：

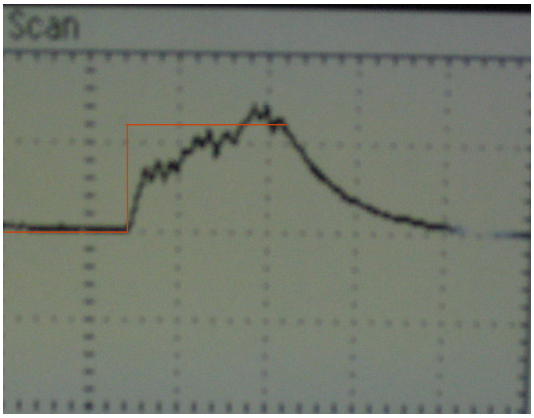
```
    #define var_p 650
        if(rand()<var_p)    //如果随机值小于 var_p 则变异，rand()在 0~65535
            {               //间变化，则小于 650 的概率约为 0.01
            /**此处放 0 变 1，1 变 0 的代码**/
            }
```

变异是按位分别查询是否发生的，并非所有位同事进行。

C）怎样实现优胜劣汰？——评估个体适应值，轮盘赌转

评估个体适应值

如果一套 PID 参数能快速、准确地控制电压输出，那就是一套好参数一个好的个体，所以我们设置一个适应值变量 A，A=偏差绝对值的积分+超调量，A 越小则适应值越高。



图中红线表示最理想的状态，那么，将黑线（实际电压）与红线（理想值）之差的绝对值累加，同时加上系统超调量，即可得到个体的适应值。

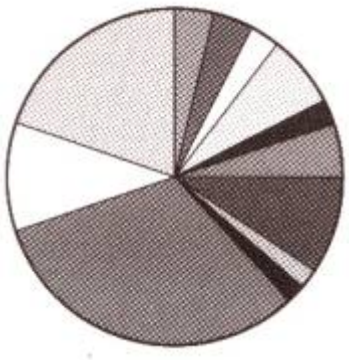
具体代码看函数：

unsigned int evaluating_unit(unsigned int unit);

轮盘赌转的方法选出高适应值的个体

《游戏编程中的人工智能技术》这本书上对此的解释已经非常通俗易懂了，在此引用一下：轮盘赌选择是从染色体群体中选择一些成员的方法，被选中的机率和它们的适应性分数成比例，染色体的适应性分数愈高，被选中的概率也愈多。这不保证适应性分数最高的成员一定能选入下一代，仅仅说明它有最大的概率被选中。其工作过程是这样的：

设想群体全体成员的适当性分数由一张饼图来代表，这一饼图就和用于赌博的转轮形状一样。我们要为群体中每一染色体指定饼图中一个小块。块的大小与染色体的适应性分数成比例，适应性分数愈高，它在饼图中对应的小块所占面积也愈大。为了选取一个染色体，你要做的，就是旋转这个轮子，并把一个小球抛入其中，让它翻来翻去地跳动，直到轮盘停止时，看小球停止在哪一块上，就选中与它对应的那个染色体。

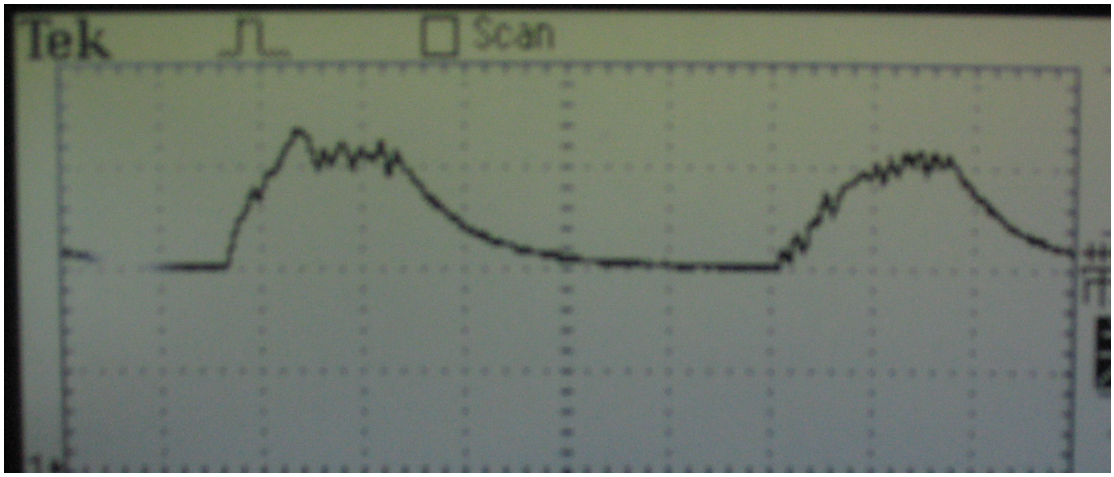


那么我们接下来怎么做呢？将种群适应值数组的头和尾相连，这就像一个圆盘。在种群中随机选择一个个体，作为转动的起点，我们要做的就是将适应值（可以理解为圆盘中扇面的大小）累加，若累加的和小于阀值，则继续累加，反正数组已经形成环形列队了，这样一直加到大于阀值，这个时候最后那个累加的个体即被选中。这种方法使适应值高的个体容易被选出，同时又使适应值低的个体有被选中的机会。

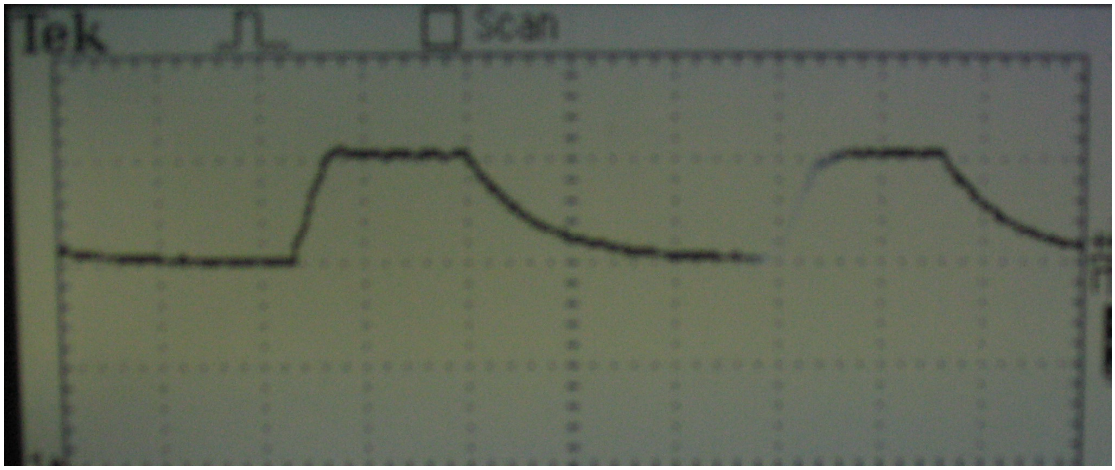
```
i=(unsigned char)(rand()/7282);//0~50 个个体中随机选择一个作为起点
while(1)
    {
    temp+=*(health+i);    //累加适应值，*health 指向适应值数组首地址
    if (temp>1200)              //若累加的结果大于阀值（可以是其它）
        {
        return (i);             //返回 0~50 编号
        }
    if( temp<1200 && col_MAX-1==++i )  //实现首尾相接
        {
        i=0;
        }
    }
```

三、进化吧……

有了以上的算子，接下来要做的就是将它们装到一起，让它完整地运行起来。其实遗传算法有各种各样的写法，这里是这样的：首先评估初始群体里的每个个体，接着选择出最糟糕的两个个体，将它们删除。然后选择两个相对较优的个体，杂交变异后得到两个新的个体，这就完成了一代的进化，接着要做的就是重复的淘汰与创造，直到得出满意的结果。



上图为初始个体的 PID 效果，调整不理想（图中有两个个体）



再看进化了几十代以后的某个个体，上升的曲线已基本“快且准”地输出设定电压（图中有两

个个体）

```
#include <stdlib.h>
#include <util/delay.h>
#include "inherit.h"
#include "pid.h"
#include "output.h"
#include "main.h"
#include "ad.h"
#include "filter.h"

/*********************************************
*函数名: void inherit(void)          遗传进化 PID 参数
*说明：
    1、调用函数：
            found(unsigned int *colony,unsigned int *health)
            evaluating_unit(unsigned int unit)
            roulette(unsigned int *health,unsigned int health_sum)
            variation(unsigned int baby)
            evaluating(unsigned int *health,unsigned *mini_p)
    2、变量说明：
            colony[]        含 N 个个体的种群
            health[]        N 个个体对应有 N 个适应值，就好像每个人都有张身份证
            health_sum          适应值总合
            dad,mum             父本，母本
            baby1,baby2         子代 1，子代 2
            mini,mini_id    最小适应值，最小适应值的 id
            epoch               遗传代数
            temp,i              一般用
            time_pid        全局变量，PID 调节的时间标志
    3、常量说明：
            col_MAX                 // 群体空间大小
            var_p               //变异概率：65 对应的变异概率约等于 0.001，650 为 0.01
            epoch_MAX           //进化代数
*入口：
*出口：
*********************************************/
#define col_MAX 50
#define var_p 650
#define epoch_MAX 200

void inherit(void)
{
    unsigned int colony[col_MAX]={
```

```
62267,15148,39769,31849,58411,49944,29915,58375 ,53831
,29144,40332,51900,60411,48378,11552,26588,61306,60089
,26887,58565, 3794,23125,53291,   646, 9102,13288,13023
,39570,17838,13029, 1001,48941,29169,61066,30539,27436
,55457,34416,13280,44049,54926, 1287,44647,24869,54512
,32952,46495,28107,19963,12429
                                    };
unsigned int health[col_MAX];                   // 对应 colony[] ，每个个体的适应值，
unsigned int health_sum;                        //适应值总合
unsigned int dad,mum,baby1,baby2;
unsigned int mini,mini_id;
unsigned int temp;
unsigned char i,epoch;


mini=found(&colony[0],&health[0]);                  //评估初始个体，并作适应值缩放

for(epoch=0;epoch<epoch_MAX;epoch++)                //开始进化
    {
    i=roulette(&health[0],health_sum);
    dad=colony[i];
    i=roulette(&health[0],health_sum);
    mum=colony[i];

    baby1=dad&0xff;
    baby1|=mum&0xff00;
    baby2=mum&0xff;
    baby2|=dad&0xff00;

    baby1=variation(baby1);                         //变异
    baby2=variation(baby2);

    temp=evaluating_unit(baby1);
    if(temp>mini)
        {
        mini_id=evaluating(&health[0]);         //取得最差个体的适应值，及其 id
        colony[mini_id]=baby1;
        health[mini_id]=temp-mini;
        }

    temp=evaluating_unit(baby2);
    if(temp>mini)
        {
        mini_id=evaluating(&health[0]);         //取得最差个体的适应值，及其 id
```

```
                colony[mini_id]=baby2;
                health[mini_id]=temp-mini;
                }
            }
            while(1);                                   //结束进化
}


/*********************************************
*函数名:unsigned int evaluating_unit(unsigned int unit)          //评估个体适应值
*说明：
    1、调用函数：
                unsigned int ad(void)                               //AD 采集

                unsigned int filter_8pj(unsigned int temp)      //滤波
                unsigned int filter_yj(unsigned int new_data)   //滤波
                void redressal(unsigned int k_dip);             //对基因进行解码
                int pid(int nonce,int aim);                         //增量 PID 调整
                void output(unsigned char pwm);                     //系统输出
    2、变量说明：
                unit            //个体基因
                ad_value        //采集的电压值
                max                 //最大超调量
                test_pwm        //无调制时的输出值
                uk                  //PID 增量
                temp,i              //一般用
    3、常量说明：
                ev_N                //PID 调整次数，采集每次调整的绝对误差，并累
加
                aim_value           //期望电压值，350 对应 3.50 伏

*入口：         evaluating_unit(unsigned int unit) //传递个体基因
*出口：         return (ret)                        //返回适应值
*********************************************/
#define ev_N 25
#define aim_value 300
unsigned int evaluating_unit(unsigned int unit)
{
    unsigned int ret=0,temp=0,ad_value=0,max=0;
    unsigned char i=0;
    unsigned char test_pwm=100;
    int uk=0;
```

```
redressal(unit);                    //根据个体，修改 PID 三个参数
output(test_pwm);
for(i=0;i<120;i++)
      {
      _delay_ms(10);
      }

for(i=0;i<28;i++)
      {
      ad_value=ad();
      ad_value=filter_8pj(ad_value);
      ad_value=filter_yj(ad_value);
      ad_value=ad_value*20/41;
      }
i=0;
while(i<ev_N)                                  //PID 调整
      {
      if((1==time_pid))
            {
            time_pid=0;
            if(ad_value>aim_value)
                  {
                  temp=ad_value-aim_value;
                  if(temp>max)
                        {
                        max=temp;
                        }
                  }
            else temp=aim_value-ad_value;

            ret+=temp;
            uk=pid(ad_value,aim_value);
            if((test_pwm+uk)>255)
                  {
                  output(255);
                  }
            else output(test_pwm+=uk);
            i++;
            }
      else
            {
            ad_value=ad();                              //采集电压
            ad_value=filter_8pj(ad_value);          //递推平均滤波
            ad_value=filter_yj(ad_value);       //一阶自调整滤波
```

```
                ad_value=ad_value*20/41;                //转换成实际电压(比实际值放大了 100
倍)
                }
            }
        output(test_pwm);
        ret=65500/(ret/ev_N+max);
        return (ret);
}

/*********************************************
*函数名:unsigned int found(unsigned int *colony,unsigned int *health)    //计算初始群体适应
值，并找出最小值
*说明：
    1、调用函数：
                unsigned char evaluating(unsigned int *health)              //评估个体适应
值函数
    2、变量说明：
                mini      //最小适应值
    3、常量说明：
                无
*入口：unsigned int found(unsigned int *colony,unsigned int *health)    //传递群体及群体适
应值数组指针
*出口：return (mini)          //返回最小适应值
*********************************************/
unsigned int found(unsigned int *colony,unsigned int *health)
{
    unsigned char i;
    unsigned int mini=0xff;
    for(i=0;i<col_MAX;i++)
        {
        *(health+i)=evaluating_unit(*(colony+i));
        if(*(health+i)<mini)
            {
            mini=*(health+i);
            }
        }

    for(i=0;i<col_MAX;i++)                  //适应值缩放
        {
        *(health+i)-=mini;
        }
    return (mini);
}
```

```
/*****************************************
*函数名:unsigned char roulette(unsigned int *health,unsigned int health_sum)  //轮盘赌转
*说明：
    1、调用函数：
                rand()          //产生 0~0XFFFF 随机值（最大随机值在 stdlib.h 中修改）
    2、变量说明：
                i,temp          //一般用
    3、常量说明：
                col_MAX             //  群体空间大小
*入口：unsigned char roulette(unsigned int *health,unsigned int health_sum)  //传递适应值数组指针及适应值总合
*出口：return (i)               //返回被选中的 id(数组下标)
*****************************************/
unsigned char roulette(unsigned int *health,unsigned int health_sum)
{
    unsigned char i;
    unsigned int temp=0;
    i=(unsigned char)(rand()/7282);          //0~50 随机选择起点
    while(1)
        {
        i++;
        temp+=*(health+i);                   //累加适应值
        if (temp>1200)
            {
            return (i);
            }
        if( temp<1200 && col_MAX-1==i )    //实现首尾相接
            {
            i=0;
            }
        }
    return i;
}

/*****************************************
*函数名:unsigned char evaluating(unsigned int *health)    //取得最小适应值的个体  id(数组下标)
*说明：
    1、调用函数：           无
    2、变量说明：
                id          //最小适应值的个体 id(数组下标)
                mini //当前最小值
    3、常量说明：           无
*入口：unsigned char evaluating(unsigned int *health)     //传递适应值数组指针
```

```
*出口：return (id)        //返回最小适应值的个体 id(数组下标)
*****************************************/
unsigned char evaluating(unsigned int *health)
{
    unsigned char i,id;
    unsigned int mini=0xffff;
    for(i=0;i<col_MAX;i++)
        {
        if(*(health+i)<mini)
            {
            mini=*(health+i);
            id=i;
            }
        }
    return id;
}

/*****************************************
*函数名:unsigned int variation(unsigned int baby)            //对基因进行变异
*说明：
    1、调用函数：
            无
    2、变量说明：
            i     循环用
    3、常量说明：
            var_p                                    //变异概率：65 对应的变异概率
约等于 0.001，650 为 0.01
*入口：unsigned int variation(unsigned int baby)//传递个体基因
*出口：return (baby)                                //返回变异后的基因
*****************************************/
unsigned int variation(unsigned int baby)
{
    unsigned char i;
    for(i=0;i<16;i++)
        {
        if(rand()<var_p)
            {
            if(0==(baby&(1<<i)))
                {
                baby|=(1<<i);
                }
            else baby&=~(1<<i);
            }
        }
```

```
        return baby;
}
```