1、 实验目的

PWM 波形调制算法的 DSP 实现与功能验证。

2、 解决方案

PWM 波形调制的原理其实是，将输入的信号与一组高频三角波进行比较，若输入信号大于三角波则输出 1，输入信号小于三角波则输出 0。实际上可以理解为通过输入信号的幅度控制输出 PWM 波的占空比，不过三角波比较的形式更为直观。

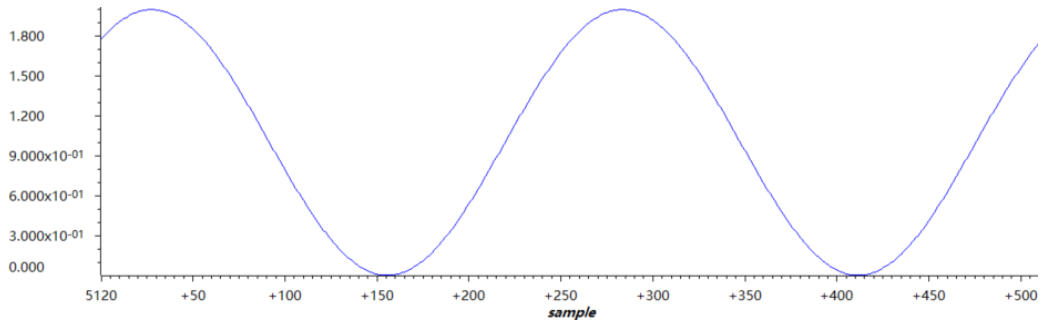
InpuWave 生成了一个频率为 2*pi/256 的正弦波，在 1000000Hz 采样率下约为 3906.25Hz，为了方便观察实验结果，我在 PWM 输出后添加了一个 IIR 滤波器用于观察滤波后的输出波形是否正常。IIR 为截止频率 10kHz，1dB 纹波的四阶切比雪夫低通滤波器。

Trianglewave 生成了一个周期为 8 个点的三角波，对应的实际频率应该为 1000000/8=12.5kHz。

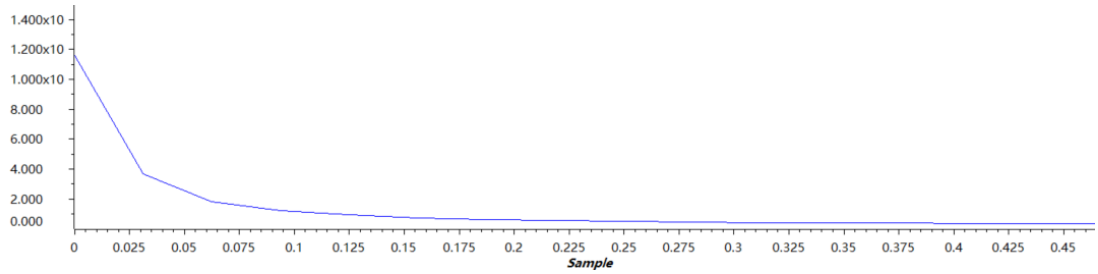
Compare 负责比较两个函数的大小，当三角波大于正弦波时返回 0，正弦波大于三角波时返回 1。

注：最初的实验使用了 512 个数组成数组（float(512)）不过由于数组过大，后期问题分析时 RAM 经常报错，所以问题分析以及附录中的程序均为 256 个数的数组。

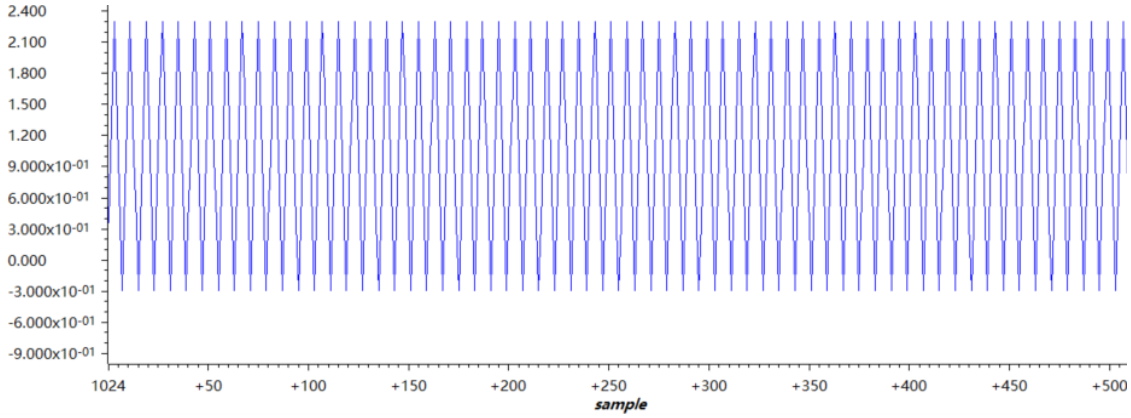
3、 实验结果与分析



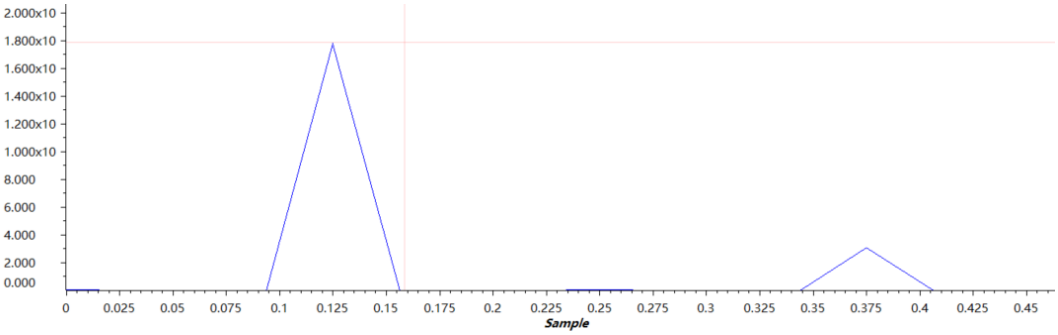
图一：输入信号



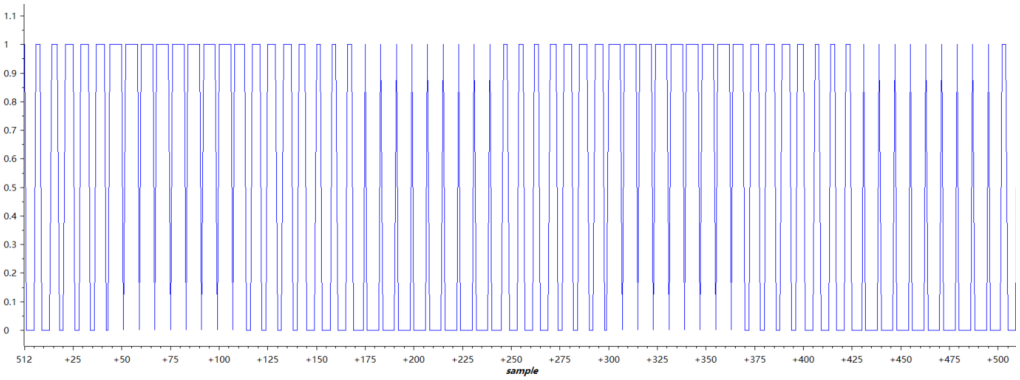
图二：输入信号的频谱



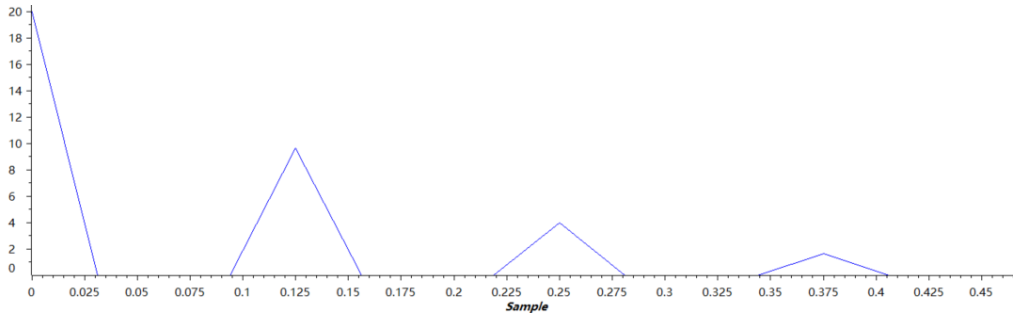
图三：生成的三角波



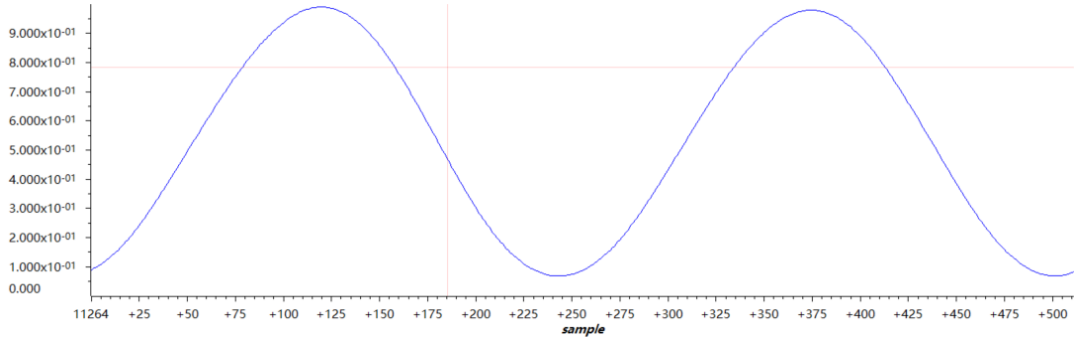
图四：三角波的频谱



图五：调制得到的输出 PWM 波形

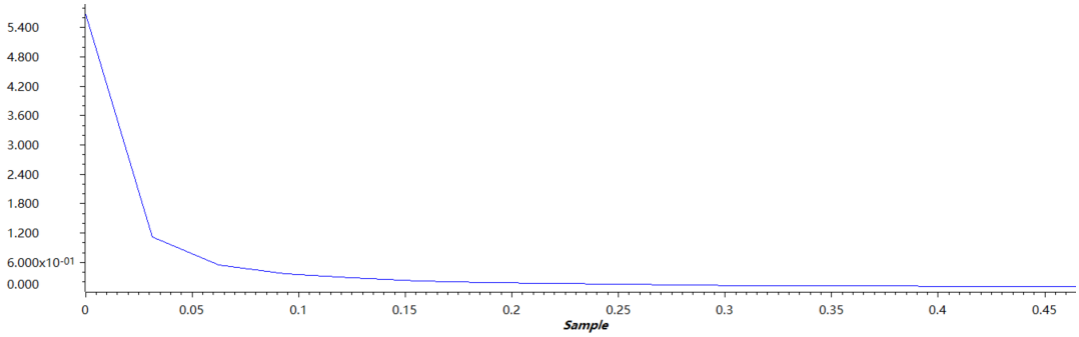


图六：PWM 波的频谱

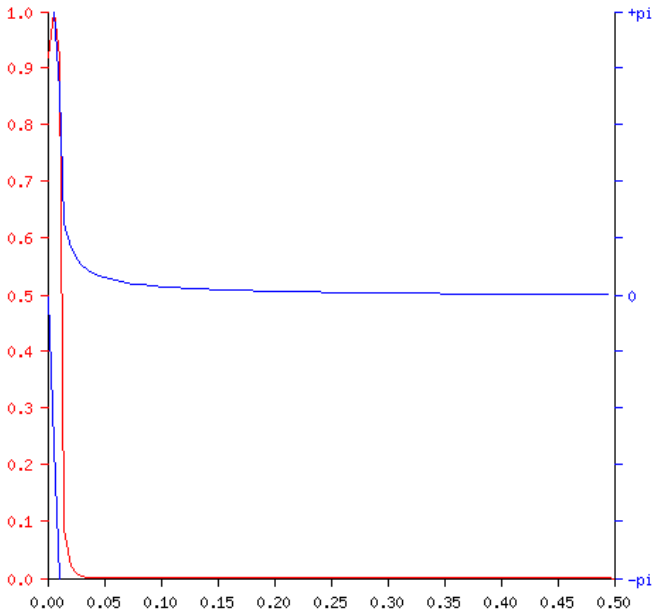


图七：IIR 滤波之后的 PWM 波时域波形

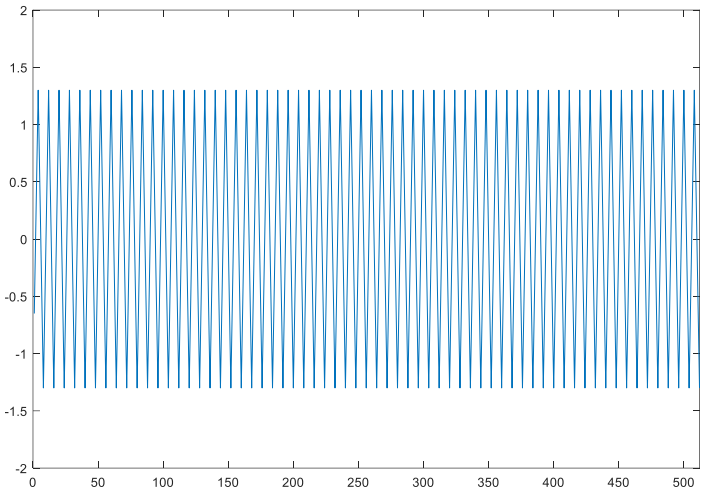
（由于不是在一次运行中截取的所有结果，图与图之间可能有相位差异）



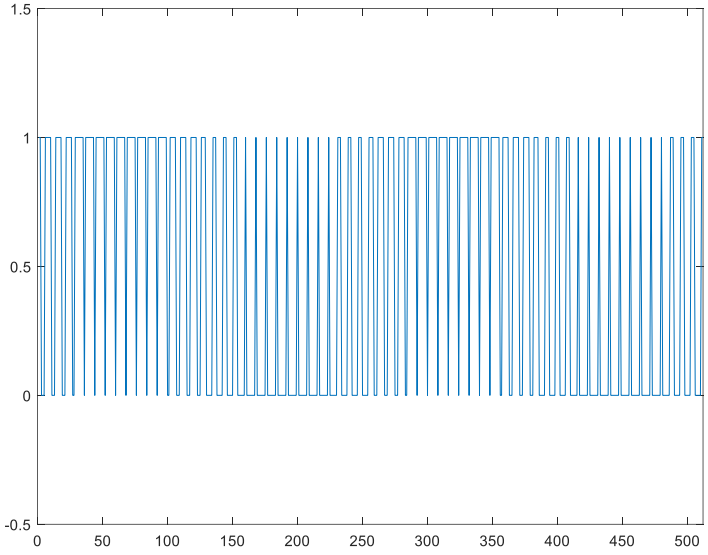
图八：IIR 滤波之后的频域波形



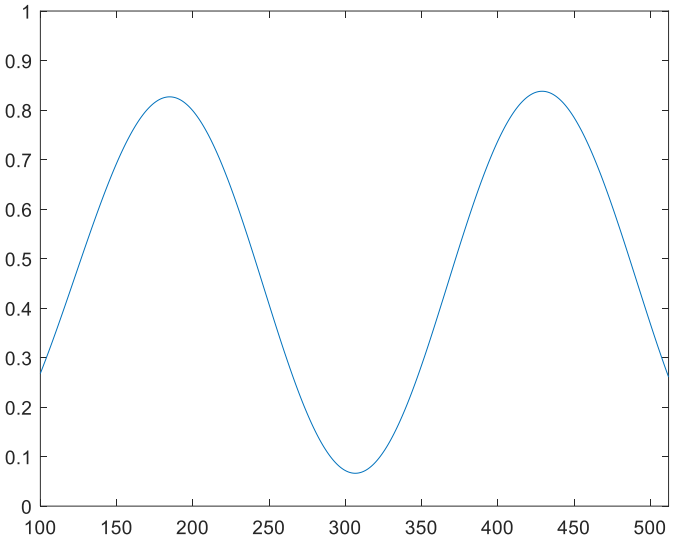
图九：IIR 滤波器的幅度与相位特性



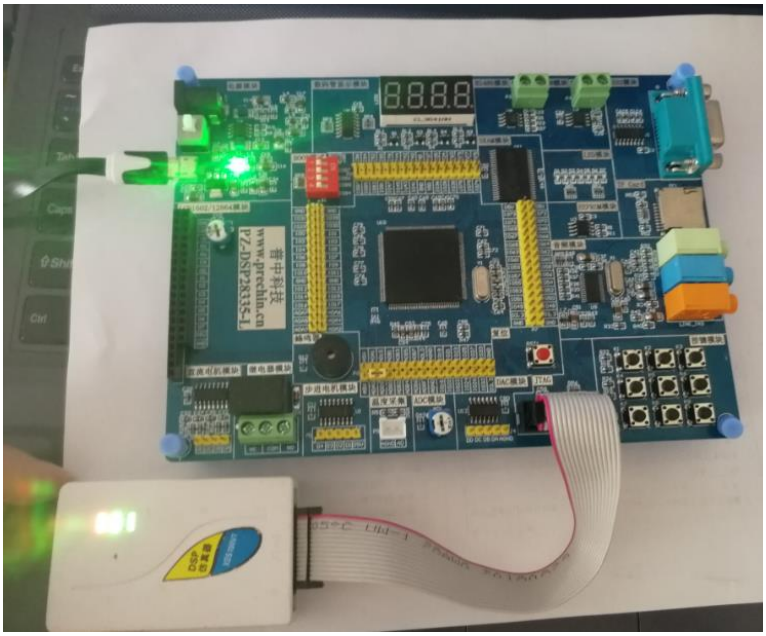
图十：matlab 生成的三角波



图十一：matlab 得到的 PWM 调制输出



图十二：matlab 中 PWM 滤波后

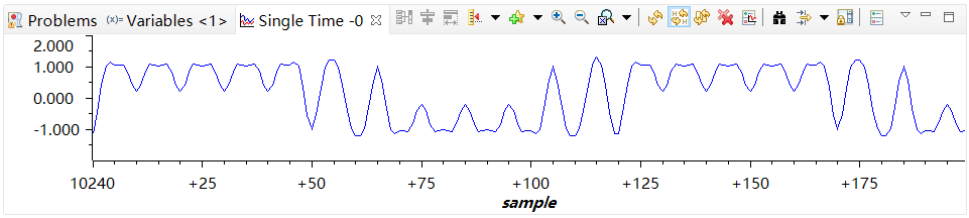


图十三：所有程序均在 28335 上进行硬件仿真验证 DSP 中的运行结果与 matlab 中基本一致，证明程序正确。

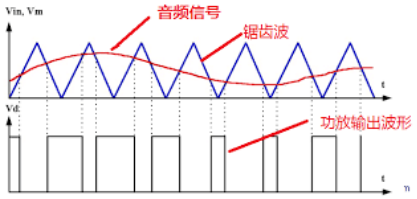
4、 问题与分析

(1).最初我的设计思路是设一条直线为 k*x+b，当上升时则 k>0,b=0。下降时则为 k<0,b=k*N/2。最后输出一条不断翻转的三角波。不过实际运行的时候我发现实现较为困难，最终我优化成了上升周期时 N++，三角波幅度下降时 N--。Y=k*N 即为输出波形，最终成功的生成了三角波，可以用于进行比较。

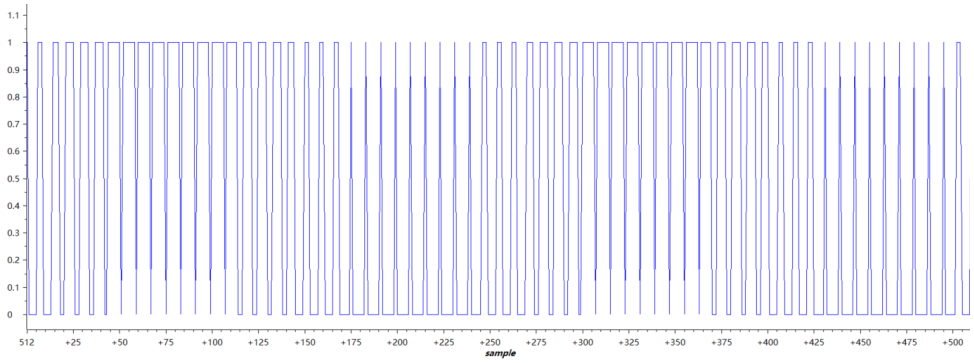
(2).最初我设计的输出波形滤波后情况很差：



经过分析后，我认为有几点原因：三角波的幅度设置过小，三角波频率不够高，滤波器截止频率不够低。在查阅资料时，我发现了这张图：



在第一次设计时，我将三角波的幅度与正弦波的幅度设为了相同大小，我认为这导致了输出 PWM 波在正弦波达到波峰与波谷的时候输出不准确。再将三角波的幅度调整为正弦波的 1.3 倍后，输出了正确的结果。再调整了其他参数后，得到了最终结果：



(3).另一个有问题的地方是，DSP中并没有bit型变量（仅在C51核），PWM的输出信号被存在了一个int型变量组里，导致占用了大量的无用空间。之后在优化时，可以考虑设置一组unsigned int变量，变量中的每一位变量都储存一次PWM输出结果，通过不断移位储存结果，保证每一个位都不浪费。于是我添加了如下代码

SavePWM[shiftteamI]=SavePWM[shiftteamI]|Dat;

SavePWM[shiftteamI]=SavePWM[shiftteamI]<<1;（完整程序在附录中）

成功的将PWM信号存入了int变量组中。

Expression	Type	Value	Address
[0]	unsigned int	36750	0x0000CE00@Data
[1]	unsigned int	36750	0x0000CE01@Data
[2]	unsigned int	57310	0x0000CE02@Data
[3]	unsigned int	57310	0x0000CE03@Data
[4]	unsigned int	57310	0x0000CE04@Data
[5]	unsigned int	57310	0x0000CE05@Data
[6]	unsigned int	53134	0x0000CE06@Data
[7]	unsigned int	36750	0x0000CE07@Data
[8]	unsigned int	1798	0x0000CE08@Data
[9]	unsigned int	1794	0x0000CE09@Data
[10]	unsigned int	514	0x0000CE0A@Data
[11]	unsigned int	514	0x0000CE0B@Data
[12]	unsigned int	514	0x0000CE0C@Data
[13]	unsigned int	514	0x0000CE0D@Data
[14]	unsigned int	1798	0x0000CE0E@Data
[15]	unsigned int	1798	0x0000CE0F@Data
[16]	unsigned int	36750	0x0000CE10@Data
[17]	unsigned int	36750	0x0000CE11@Data
[18]	unsigned int	57310	0x0000CE12@Data
[19]	unsigned int	57310	0x0000CE13@Data
[20]	unsigned int	57310	0x0000CE14@Data
[21]	unsigned int	57310	0x0000CE15@Data
[22]	unsigned int	53134	0x0000CE16@Data
[23]	unsigned int	36750	0x0000CE17@Data
[24]	unsigned int	1798	0x0000CE18@Data
[25]	unsigned int	1794	0x0000CE19@Data
[26]	unsigned int	514	0x0000CE1A@Data
[27]	unsigned int	514	0x0000CE1B@Data
[28]	unsigned int	514	0x0000CE1C@Data
[29]	unsigned int	514	0x0000CE1D@Data
[30]	unsigned int	1798	0x0000CE1E@Data
[31]	unsigned int	1798	0x0000CE1F@Data

可以看到，数值与PWM输出信号明显相关，输出时需要进行二进制转换，逐个移位即可。

(4).对于实际应用来说，假如应用在D类功放中，那么开关管的开关频率是调制三角波的频率。但实际上输出的信号频率其实是DSP内部的输出频率，而非三角波本身，不过因为有大量连续的0和1，实际输出的开关频率应该是三角波的频率。为了与实际设置的采样率1MHz相对应，DSP在输出PWM驱动信号的时候也应该符合1MHz的速率，这里应当配合定时器进行稳定的输出，同时注意时钟的准确性，或者直接调用DSP的PWM功能。并且应当对输出函数进行一定的处理。假设上管为00000011，下管应当为11111100，为了防止输出误差导致半桥的上下管同时开启而烧毁，可以改为00000011与11111000，即增加一个CLK的死区，1MHz输出下约为1μS(对于MOS来说有些低了)。不过，死区会较大的影响信号质量，设置的时间长短需要根据实际情况进行修改，这里附一段调整死区时间以及输出差分信号的程序。

```
difPWMout[ii]=PWMout[ii];difPWMout[ii]=!difPWMout[ii];}//比较完成后，生成差分输出
if(PWMout[ii]==0&&PWMout[ii+1]==1)
    {for(iii=1;iii<DT+1;iii++){ PWMout[ii+iii]=0;}}
if(PWMout[ii]==1&&PWMout[ii+1]==0)
    {for(iii=1;iii<DT+1;iii++){difPWMout[ii+iii]=0;}}
```

（完整程序在附录中）

(4).在实际应用中，PWM应当需要更高的频率。当前主流音频D类功放的开关频率约在400~600kHz左右，不过这对于DSP来说有些过高了，当前使用纯三角波的频率为12.5kHz，如果改用锯齿波则可提升至25 kHz。我认为在实际应用中，可以考虑通过模拟的方法产生D类功放所需的高频开关信号，或者采用运算速度更高的处理器。如果采用数字方法，想要取得更高的开关频率还可以考虑通过插值升频提高采样率和三角波频率，提高功放输出开关频率。

5、 附件

```
#include "DSP2833x_Device.h"     // DSP2833x Headerfile Include File
#include"math.h"

//滤波器参数
#define NZEROS 4
#define NPOLES 4
#define GAIN   3.836492796e+06

static float xv[NZEROS+1], yv[NPOLES+1];

//输入信号参数
#define SIGNAL1F 3906.25
#define PI 3.1415926

//定义函数
float InputWave();
float Trianglewave();
int Compare();
float filterloop();
void shiftPWM();
void dif();
void DeadTime(DT);

unsigned int SavePWM[16];
unsigned int shiftteamI,shiftI;
void INTshiftPWM();


float ftng,fn;
float fInput,fOutput,fCompare;
float fSignal1;
float fStepSignal1;
float f2PI;
int bit;
float fIn[256],fCp[256],fOut[256];
int nIn,Ntng,PWMoneCLK,Cp;
int PWMout[256];
int difPWMout[256];//差分输出推动半桥


void main()
```

```
{
    InitSysCtrl();
    INTshiftPWM();
    nIn=0;
    f2PI=2*PI;
    Ntng=1;
    bit=1;
    fSignal1=0.0;
    fStepSignal1=2*PI/256;
    shiftteamI=0;
    shiftI=0;
    for(;;)
    {
        fCompare=Trianglewave();
        fCp[nIn]=fCompare;//观察三角波

        fInput=InputWave();
        fIn[nIn]=fInput;//观察正弦波

        Cp=Compare();
        PWMout[nIn]=Cp;

        shiftPWM();

        fOut[nIn]=filterloop();

        nIn++;

        if(nIn==256)
        {
            nIn=0;
            dif();
            DeadTime(5);

        }
    }

}


float InputWave()
{
    fn=sin(fSignal1);
    fSignal1+=fStepSignal1;
```

```
    if ( fSignal1>=f2PI )  fSignal1-=f2PI;
    return(fn);
}

float Trianglewave()
{
    ftng=0.65*Ntng-0.3-1;
    if ( Ntng==4)   {bit=0;}//此时N--
    if ( Ntng==0)   {bit=1;}//此时N++
    switch(bit)
    {
        case 0:Ntng--;
        break;
        case 1:Ntng++;
        break;
        default:break;
    }
    return(ftng);
}

int Compare()
{
    if(fCompare>=fInput)PWMoneCLK=0;
    else PWMoneCLK=1;
    return(PWMoneCLK);
}


float filterloop()
  {
       xv[0] = xv[1]; xv[1] = xv[2]; xv[2] = xv[3]; xv[3] = xv[4];
       xv[4] = Cp/GAIN;
       yv[0] = yv[1]; yv[1] = yv[2]; yv[2] = yv[3]; yv[3] = yv[4];
       yv[4] =   (xv[0] + xv[4]) + 4 * (xv[1] + xv[3]) + 6 * xv[2]
                  + ( -0.9418920323 * yv[0]) + (  3.8201946290 * yv[1])
                  + ( -5.8145385566 * yv[2]) + (  3.9362317895 * yv[3]);
       return(yv[4]);

  }
void INTshiftPWM()
{
    shiftteamI=0;
    shiftI=0;
    int iii;
```

```
    unsigned int maxint=65535;
    for(iii=0;iii<=32;iii++){SavePWM[iii]=maxint;}
}
void shiftPWM()
{
    shiftI++;
    unsigned int Dat;
    Dat=Cp;
    SavePWM[shiftteamI]=SavePWM[shiftteamI]|Dat;
    SavePWM[shiftteamI]=SavePWM[shiftteamI]<<1;
    if(shiftI>15){shiftteamI++;shiftI=0;}
}

void dif()
{
    unsigned int ii;
    for(ii=0;ii!=256;ii++)
    {difPWMout[ii]=PWMout[ii];difPWMout[ii]=!difPWMout[ii];}//比较完成后，生成
差分输出
}

void DeadTime(DT)
{
    int ii,iii;
    for(ii=DT;ii<512;ii++)
    {
    if(PWMout[ii]==0&&PWMout[ii+1]==1)
        {for(iii=1;iii<DT+1;iii++){ PWMout[ii+iii]=0;}}
    if(PWMout[ii]==1&&PWMout[ii+1]==0)
        {for(iii=1;iii<DT+1;iii++){difPWMout[ii+iii]=0;}}
    }
}
```

滤波器设计结果：

Command line: /www/usr/fisher/helpers/mkfilter -Ch -1.0000000000e+00 -Lp -o 4 -a 1.0000000000e-02 0.0000000000e+00

raw alpha1 = 0.0100000000
raw alpha2 = 0.0100000000
warped alpha1 = 0.0100032912
warped alpha2 = 0.0100032912
gain at dc : mag = 3.836492796e+06 phase = -0.0000000000 pi
gain at centre: mag = 3.836492796e+06 phase = 0.7239253492 pi
gain at hf : mag = 0.000000000e+00

S-plane zeros:

S-plane poles:
-0.0087701907 + j 0.0618078705
-0.0211731132 + j 0.0256016582
-0.0211731132 + j -0.0256016582
-0.0087701907 + j -0.0618078705

Z-plane zeros:
-1.0000000000 + j 0.0000000000 4 times

Z-plane poles:
0.9893846854 + j 0.0612113977
0.9787312093 + j 0.0250640580
0.9787312093 + j -0.0250640580
0.9893846854 + j -0.0612113977

Recurrence relation:
y[n] = (1 * x[n- 4])
+ (4 * x[n- 3])
+ (6 * x[n- 2])
+ (4 * x[n- 1])
+ (1 * x[n- 0])

+ (-0.9418920323 * y[n- 4])
+ (3.8201946290 * y[n- 3])
+ (-5.8145385566 * y[n- 2])
+ (3.9362317895 * y[n- 1])

原始 C 代码

```
#define NZEROS 4
#define NPOLES 4
#define GAIN   3.836492796e+06

static float xv[NZEROS+1], yv[NPOLES+1];

static void filterloop()
  { for (;;)
    { xv[0] = xv[1]; xv[1] = xv[2]; xv[2] = xv[3]; xv[3] = xv[4];
      xv[4] = next input value / GAIN;
      yv[0] = yv[1]; yv[1] = yv[2]; yv[2] = yv[3]; yv[3] = yv[4];
      yv[4] =   (xv[0] + xv[4]) + 4 * (xv[1] + xv[3]) + 6 * xv[2]
                + ( -0.9418920323 * yv[0]) + (   3.8201946290 * yv[1])
                + ( -5.8145385566 * yv[2]) + (   3.9362317895 * yv[3]);
      next output value = yv[4];
    }
  }
```

matlab 程序

```
clc;close all;clear all;
fs=1000000;
Ts=1/fs;
N=1:512;
y=sin(2*pi*N/256);

tng=1.3*sawtooth(2*pi*0.125*N,1/2);

for i=1:512
    if y(i)>=tng(i)
        PWM(i)=1
    elseif y(i)<tng(i)
        PWM(i)=0
    end
end
% plot(PWM);axis([0 512 -0.5 1.5]);
% plot(tng);axis([0 512 -2 2]);
[b,a] = cheby1(4,1,0.01);
yx=filter(b,a,PWM);
plot(yx);axis([100 512 0 1]);
```