南京理工大学
NANJING UNIVERSITY OF SCIENCE & TECHNOLOGY

DSP 应用技术实验

DSP 数据采集实验报告

作　　者：许晓明　　学　　号：9161040G0734

同 组 人：李玥　　学　　号：9161040G0703

同 组 人：陈锦涛　　学　　号：9161040G0614

学　　院：电子工程与光电技术学院

专　　业：电子信息工程

班　　级：电信 3 班

组　　号：第二组 B4

题　　目：DSP 应用技术实验

DSP 数据采集实验报告

指 导 者：李彧晟

2019 年 11 月

目录

1 实验目的

1. 熟悉 DSP 硬件开发平台；
2. 掌握 F28335 的 ADC 外设的控制；
3. 掌握 F28335 中断的设置；
4. 熟悉 DSP 代码调试基本方法。

2 实验仪器

2.1 实验仪器清单

1. DSP 仿真平台（仿真器、DSP 实验箱、计算机） 一套
2. 示波器 一台
3. 信号发生器 一台

2.2 硬件连接示意图

实验硬件连接大致如图 2. 1 所示，F28335 的 ADC 原理如图 2. 2 所示。

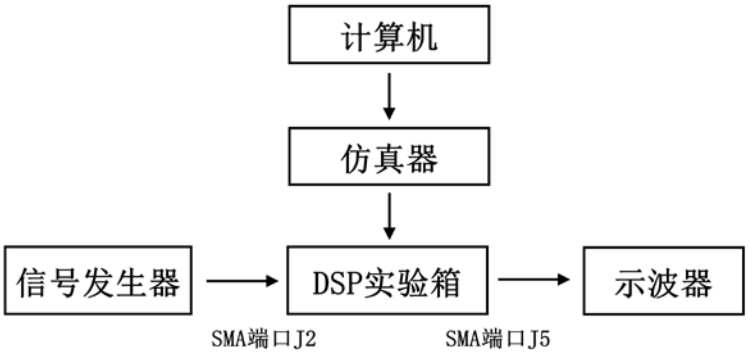

图 2. 1 硬件连接示意图

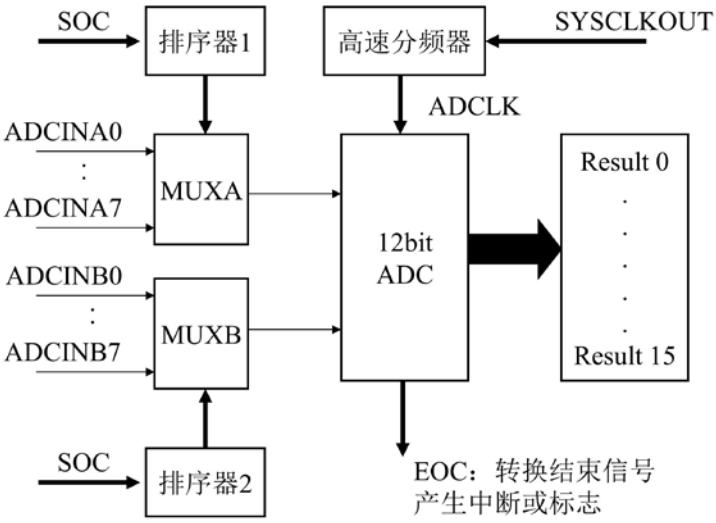

图 2. 2 F28335 的 ADC 原理

3 实验步骤及现象

3.1 程序流程图

结合实验要求，程序流程大致为：依赖于 ADC、DSP 以及 DAC 三大基本部件，由 ADC 将采集的数据送到 DSP，通过中断的方式，在中断服务程序中，将采集到的数据存储在内存中，并输出到 DAC。于是，程序流程图如图 3.1 所示。

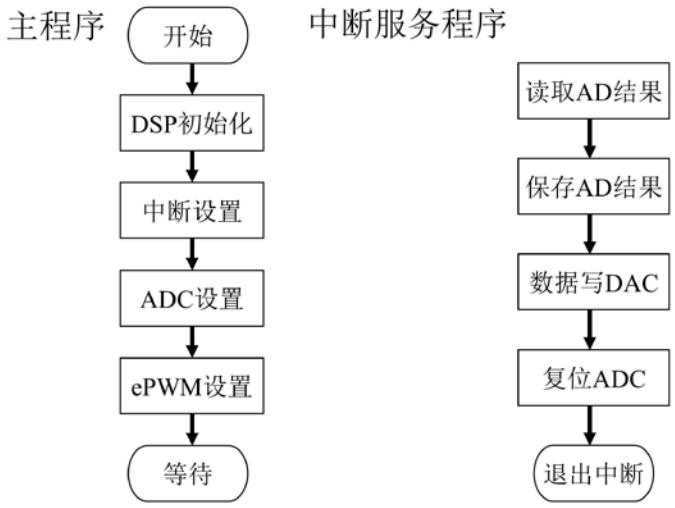

图 3.1 DSP 数据采集程序流程图

3.2 检查设备并启动开发环境

检查仿真器、 F28335 DSP 教学实验箱、计算机之间的连接是否正确。确认无误后开启电源，并在计算机上启动开发环境。

3.3 编写数据存储代码

3.3.1 数据存储的原理

中断服务程序触发的条件是 ADC 采样信号到来，原本的范例程序中直接将这个信号输送给 DA，实现数据实时输入输出。

将这个采集到的信号同时存入 SampleTable1 数据空间，可以实现数据保存。但每次中断到来，只能保存一个数据。因此需要设定一个数据存放位置指示变量 ConvCount，每次存放数据后，位置加一。当指示变量超过 SampleTable1 数据空间的长度（1024 个值）后，将 ConvCount 归零，以实现 SampleTable1 数据空间中数据的不断更新。

3.3.2 数据存储代码

综合以上内容，线性调频信号查找表的产生代码如下：

```
interrupt void epwm1_timer_adc_isr(void)     //中断函数
{
//DA
    xn= (AdcRegs.ADCRESULT1 & 0xFFF0);
     if(ConvCount<1024)
    {
        SampleTable1[ConvCount]=xn;
        ConvCount++;
    }
    else
    {
        ConvCount=0;
    }
    *Da_out= xn ;
```

3.4 建立工程并运行、调试程序

连接信号发生器至教学实验箱 SMA 输入端口 J2、教学实验箱 SMA 输出端口 J5 至示波器，编译链接工程并进入调试调试界面，运行程序后，查看存储空间中时域波形。

3.5 修改采样频率并验证

阅读程序，发现主程序中的如下代码：

```
// Set Period for EPWM1
   EPwm1Regs.TBPRD = 208;     //设定时间基准器计数器的周期        208-fs 20kHz,139-f
   EPwm1Regs.TBCTL.bit.CTRMODE = TB_COUNT_UPDOWN;  //增减计数模式
// Setup Compare A = 2 TBCLK counts
   EPwm1Regs.CMPA.half.CMPA = 2;    //计数比较寄存器A CMPA    当前工作的CMPA的值不断
// Phase is 0 for Synchronization Event
   EPwm1Regs.TBPHS.half.TBPHS = 0x0000;      //TBCTR不装载相位寄存器TBPHS的值
// Clear  TB counter
   EPwm1Regs.TBCTR = 0x0000;   //事件基准计数寄存器TBCTR    读取写到其中的TBCTR的值
```

可知修改 TBPRD 的值可以实现修改采样频率，修改值为 139，则理论采样频率变为 30kHz，通过软件和硬件的方法分别验证采样频率是否正确。

4 实验结果及思考题回答

4.1 外设初始化信息

根据范例，写出各外设初始化的寄存器、数值及其含义。

查看主程序中的注释信息，如图 4.1 所示，系统初始化了 PIE 中断设置相关的寄存器、XINTF 中 zone7 相关的寄存器、ePWM 相关的寄存器和 ADC 相关的寄存器。

```c
void main(void)
{
    InitSysCtrl();
    InitMcbspaGpio();
    DINT;                   //禁止 CPU中断，禁止全局中断
    InitPieCtrl();          //初始化PIE控制寄存器
    IER=0x0000;             //禁用所有CPU中断并清除CPU中断标志位
    IFR=0x0000;
    InitPieVectTable();   //初始化PIE向量表     里面包含了      PieCtrlRegs.PIECTRL.bit.ENPIE=1

    EALLOW;
    PieVectTable.SEQ1INT =&epwm1_timer_adc_isr;  //第一组第三中断
    EDIS;

    InitAdcParameters();
    InitEPwm1Parameters();

    PieCtrlRegs.PIEIER1.bit.INTx1 = 1;  // Enable SEQ1INT interrupt in PIE
    PieCtrlRegs.PIECTRL.bit.ENPIE=1;  //打开PIE中断,使能PIE
    IER |= M_INT1;      //打开CPU第1组中断

    EINT;      //使能全局中断，允许中断响应
    ERTM;

    init_zone7();     //初始化地址空间zone7

    init_mcbsp_spi();
    init_AD9747();
```

```c
void init_zone7(void)
{
    EALLOW;
    // Make sure the XINTF clock is enabled
    SysCtrlRegs.PCLKCR3.bit.XINTFENCLK = 1;
    EDIS;
    // Configure the GPIO for XINTF with a 16-bit data bus
    // This function is in DSP2833x_Xintf.c
    InitXintf16Gpio();

    // All Zones---------------------------------
    // Timing for all zones based on XTIMCLK = SYSCLKOUT
    EALLOW;
    XintfRegs.XINTCNF2.bit.XTIMCLK = 0;  //   XTIMCLK=SYSCLKOUT/1
    // Buffer up to 3 writes
    XintfRegs.XINTCNF2.bit.WRBUFF = 3;   //   写缓冲模式配置
    // XCLKOUT is enabled
    XintfRegs.XINTCNF2.bit.CLKOFF = 0;   //   XCLKOUT使能
    // XCLKOUT = XTIMCLK
    XintfRegs.XINTCNF2.bit.CLKMODE = 0;  //   XCLKOUT=XTIMCLK

    // zone7 配置-----------------------------------
    // When using ready, ACTIVE must be 1 or greater
    // Lead must always be 1 or greater
    // Zone write timing
    XintfRegs.XTIMING7.bit.XWRLEAD = 2;   //写周期各阶段时序配置  8个时钟周期
    XintfRegs.XTIMING7.bit.XWRACTIVE = 4;
    XintfRegs.XTIMING7.bit.XWRTRAIL = 2;
    // Zone read timing
    XintfRegs.XTIMING7.bit.XRDLEAD = 1;   //读周期各阶段时序配置 7个时钟周期
    XintfRegs.XTIMING7.bit.XRDACTIVE = 5;
    XintfRegs.XTIMING7.bit.XRDTRAIL = 1;

    // don't double all Zone read/write lead/active/trail timing
    XintfRegs.XTIMING7.bit.X2TIMING = 0;
```

```
120
121         // Zone will not sample XREADY signal
122         XintfRegs.XTIMING7.bit.USEREADY = 0;     //不采样XREADY信号
123         XintfRegs.XTIMING7.bit.READYMODE = 0;
124
125         // 1,1 = x16 data bus
126         // 0,1 = x32 data bus
127         // other values are reserved
128         XintfRegs.XTIMING7.bit.XSIZE = 3;    //使用16位数据线
129         EDIS;
130         //Force a pipeline flush to ensure that the write to
131         //the last register configured occurs before returning.
132         asm(" RPT #7 || NOP");
133     }
```

```
182 void init_AD9747(void)
183 {
184
185           *SPI_Reset = 1 ;
186           DELAY_US(100);
187           *SPI_Reset = 0 ;
188
189           mcbsp_write(0x0020);
190           DELAY_US(10);
191           mcbsp_write(0x0000);
192           DELAY_US(10);
193           //data control register//////////////////////////////////////////
194           mcbsp_write(0x02C0);
195           DELAY_US(10);
196           ////////////////////////////////////////
197           mcbsp_write(0x0300);
198           DELAY_US(10);
199           mcbsp_write(0x0A00);
200           DELAY_US(10);
201           mcbsp_write(0x0B3D);
202           DELAY_US(10);
203           mcbsp_write(0x0C00);
204           DELAY_US(10);
205           mcbsp_write(0x0D00);
206           DELAY_US(10);
207           mcbsp_write(0x0E00);
208           DELAY_US(10);
209           mcbsp_write(0x0F3D);
210           DELAY_US(10);
211           mcbsp_write(0x1000);
```

```
281 void InitEPwm1Parameters(void)
282 {
283 //   InitEPwm1Gpio();
284
285 // Disable TBCLK within the ePWM
286    EALLOW;
287    SysCtrlRegs.PCLKCR0.bit.TBCLKSYNC = 0; //停止epwm模块内部的时间基准时钟
288    EDIS;
289
290 // High Speed Time-base Clock Prescale Bits,These bits determine part of the time-base clock prescale
291 // TBCLK = SYSCLKOUT / (HSPCLKDIV*CLKDIV)=150/(6*1)=25
292    EPwm1Regs.TBCTL.bit.HSPCLKDIV =0x03;        //高速时间基准时钟预分频位                    两倍
293    EPwm1Regs.TBCTL.bit.CLKDIV = 0x00;          //时间基准时钟预分频位 等于0即1分频
294
295 // Set Period for EPWM1
296    EPwm1Regs.TBPRD = 208;    //设定时间基准器计数器的周期       208-fs 20kHz,139-fs 30kHz  149--27.9kHz T(
297    EPwm1Regs.TBCTL.bit.CTRMODE = TB_COUNT_UPDOWN;  //增减计数模式
298 // Setup Compare A = 2 TBCLK counts
299    EPwm1Regs.CMPA.half.CMPA = 2;   //计数比较寄存器A CMPA   当前工作的CMPA的值不断和时间基准计数器TBCTR比较
300 // Phase is 0 for Synchronization Event
301    EPwm1Regs.TBPHS.half.TBPHS = 0x0000;     //TBCTR不装载相位寄存器TBPHS的值
302 // Clear  TB counter
303    EPwm1Regs.TBCTR = 0x0000;   //事件基准计数寄存器TBCTR   读取写到其中的TBCTR的值     清除
304
305 // Phase loading disabled
306    EPwm1Regs.TBCTL.bit.PHSEN = TB_DISABLE;//禁止TBCTR对TBPHS的装载
307 // Enable the TBCTL Shadow
308    EPwm1Regs.TBCTL.bit.PRDLD = TB_SHADOW;//TBCTR装载其映射寄存器的值
309 // Disable EPWMxSYNCO signal
310    EPwm1Regs.TBCTL.bit.SYNCOSEL = TB_SYNC_DISABLE; //禁用EPWMxSYNCO signal
311 // CMPA Register operating mode, 0 means operates as a double buffer, all writes via the CUP access the sha
312    EPwm1Regs.CMPCTL.bit.SHDWAMODE = CC_SHADOW;//映射模式. 双缓冲模式. 所有CPU写操作将访问映射寄存器
313 // Active CMPA Load From Shadow Select Mode when CTR=0
314    EPwm1Regs.CMPCTL.bit.LOADAMODE = CC_CTR_ZERO;  // load on CTR = Zero
```

```
// Set actions
// Force EPWMA output high when the counter equals the active CMPA register and the counter is incrementing
   EPwm1Regs.AQCTLA.bit.CAU = AQ_SET;//计数递增    强制ePWMxA输出高
// Force EPWMA output low Action when the counter equals the active CMPA register and the counter is decrementing
   EPwm1Regs.AQCTLA.bit.CAD = AQ_CLEAR;//计数递减    强制ePWMxA输出低

// Dead-Band Generator Rising Edge Delay Count Register=0
//   EPwm1Regs.DBRED=0;
// Dead-Band Generator Falling Edge Delay Count Register=0
//   EPwm1Regs.DBFED=0;

// Enable ADC Start of SOCA Pulse
   EPwm1Regs.ETSEL.bit.SOCAEN = 1;     //使能ePWMxSOCA脉冲
// Select SOC from CPMA on upcount
   EPwm1Regs.ETSEL.bit.SOCASEL = 2;     //TBCTR=TBPRD时产生ePWMxSOCA
// Select how many selected ETSEL events need to occur before an EPWMxSOCA pulse is generated;//在第三个事件产生ePWMxSOCA脉冲
   EPwm1Regs.ETPS.bit.SOCAPRD = 3;

// Enable event time-base counter equal to period (TBCTR = TBPRD)
   EPwm1Regs.ETSEL.bit.INTSEL = ET_CTR_PRD; // TBCTR=TBPRD时产生ePWMxSOCA
// Enable EPWMx_INT generation
   EPwm1Regs.ETSEL.bit.INTEN = 0;           //禁止ePWMx_INT产生
// These bits determine how many selected ETSEL[INTSEL] events need to occur before an interrupt is generated.
   EPwm1Regs.ETPS.bit.INTPRD = ET_3RD;      //在第三个事件产生中断

// Enable TBCLK within the ePWM
   EALLOW;
   SysCtrlRegs.PCLKCR0.bit.TBCLKSYNC = 1;
   EDIS;
}
```

图 4.1 主程序中的相关代码

4.2 ADC 采样频率计算公式

根据范例程序，给出 ADC 的采样频率计算公式。

通过阅读例程，并查阅资料，由于范例程序中，TB 计数模式为增减计数、且每三次事件产生一次采样中断，可得此程序中 ADC 采样频率公式如下：

$$T_{PWM1} = \frac{TBCLK}{TBPRD * 2 * 3} = \frac{25}{208 * 2 * 3} = 0.02MHz$$

而其中，高速时间基准时钟预分频位为 010b，即 6 分频；时间基准时钟预分频位为 000b，即 1 分频，于是有：

$$TBCLK = \frac{SYSCLKOUT}{HSPCLKDIV * CLKDIV} = \frac{150}{6 * 1} = 25MHz$$

4.3 信号波形存储地址及作图

指出信号波形的存储地址，并作图显示。

如图 4. 2 所示，可得到波形存储地址为 0x0000C040，利用 graph 工具绘图得到图 4. 3。

Expression	Type	Value	Address
Da_out	unsigned int *	0x00200400	0x0000C004@Data
SampleTable1	unsigned int[1024]	0x0000C040@Data	0x0000C040@Data
[0 ... 99]			
[0]	unsigned int	18592	0x0000C040@Data
[1]	unsigned int	4352	0x0000C041@Data
[2]	unsigned int	4464	0x0000C042@Data
[3]	unsigned int	20448	0x0000C043@Data
[4]	unsigned int	10592	0x0000C044@Data

图 4.2 波形存储地址

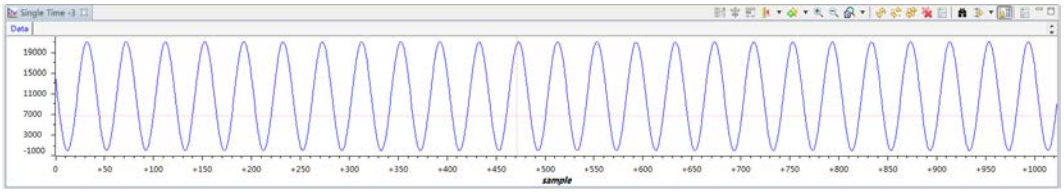

图 4.3 采样频率为 20kHz 时的波形

4.4 观察不同频率的输出情况

改变信号源的频率，观察示波器上输出。

修改信号源频率，示波器输出结果如图4. 4到图4. 7所示。

图 4. 4 信号源频率 1kHz

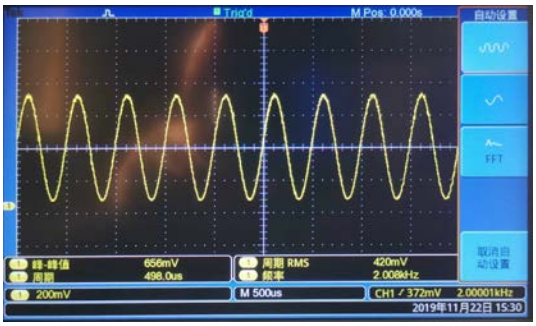

图 4. 5 信号源频率 2kHz

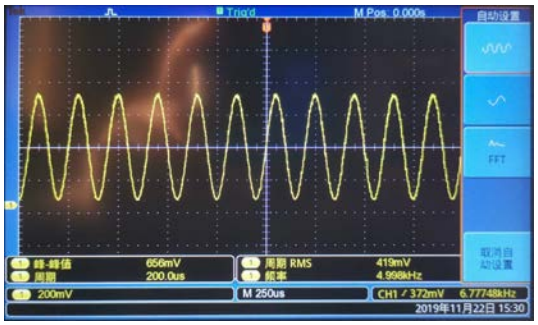

图 4. 6 信号源频率 5kHz

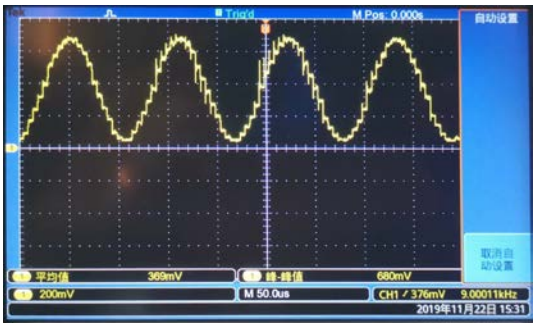

图 4. 7 信号源频率 10kHz

4.5 ADC 采样频率的软件验证

4.5.1 修改前的采样频率的验证

在 graph 绘制的波形图中，统计一个周期内的点数，与信号源输入频率相乘，即可得到大致的 ADC 采样频率。

修改 ADC 采样频率前的波形图如图 4. 3 所示，此时记录两个最高点所在位置为 537、578，输入频率为 500Hz（如图 4. 8 所示），则计算所得的采样频率为 (578-537)*500=20500Hz，与 20kHz 的理论值接近。

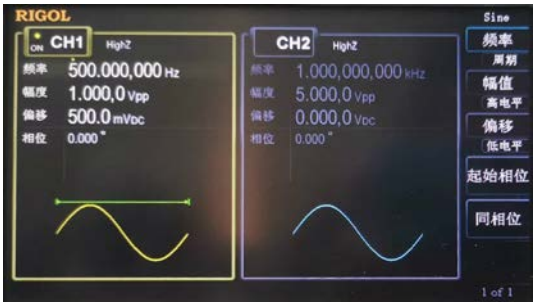

图 4. 8 信号源输入频率

4.5.2 修改后的采样频率的验证

验证方法与修改前一致，此时存储空间内的波形如图 4. 9 所示，记录两个最高点的所在位置为 560、619，输入频率同样为 500Hz，则计算所得的采样频率为 (619-560)*500=29500Hz，与理论值 30kHz 接近。

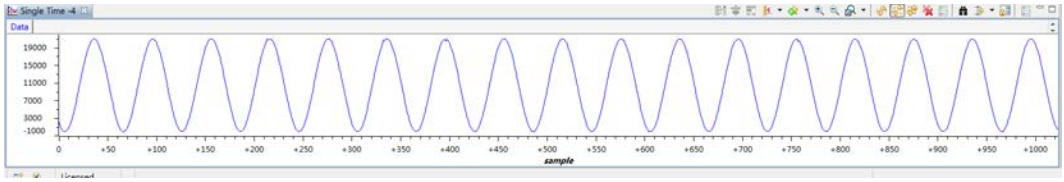

图 4. 9 采样频率为 30kHz 时的波形

4.6 ADC 采样频率的硬件验证

4.6.1 修改前的采样频率的验证

在每次进入中断时，使 DA 高电平、下次输出低电平，如此往复，对应的代码如下：

```
346  unsigned int property=10000;
347  interrupt void epwm1_timer_adc_isr(void)     //中断函数
348  {
349  //DA
350      /* xn= (AdcRegs.ADCRESULT1 & 0xFFF0);
351       if(ConvCount<1024)
352       {
353           SampleTable1[ConvCount]=xn;
354           ConvCount++;
355       }
356       else
357       {
358           ConvCount=0;
359       }
360       *Da_out= xn ;
361       */
362        *Da_out=property;
363        property=10000-property;
```

此时，可在示波器上观察到方波如图 4. 10 所示，方波的频率是采样频率的一半，即硬件验证的采样频率为 10.00*2=20kHz，与理论值一致。

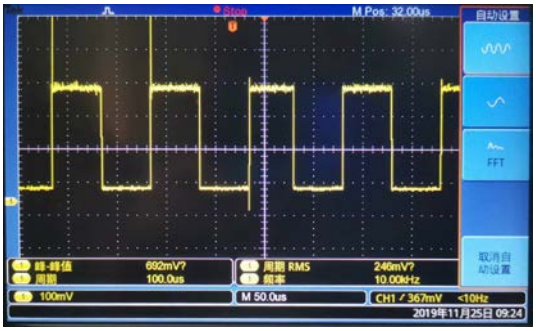

图 4. 10 硬件验证 20kHz 采样频率时的示波器波形

4.6.2 修改后的采样频率的验证

验证方法与修改前一致，此时示波器波形如图 4. 11 所示，则硬件验证的采样频率为 15.37*2=30.74kHz，与理论值接近。

图 4. 11 硬件验证 30kHz 采样频率时的示波器波形

5 实验总结

5.1 实验中遇到的问题及解决方法

1.存储空间中的数值没有变化

在第一次编写代码时，运行后发现 SampleTable1 对应的存储空间中的数值没有发生实时的变化。查看赋值语句的修改代码后发现，是没有处理好位置指示变量 ConvCount。重新编写代码，当 ConvCount 大于 1023 后对其赋 0，重新编译链接，运行后数据可以实现实时变化。

2. 软件、硬件验证得到的采样频率与理论值误差极大

在第三次实验开始验证采样频率时，发现无论用硬件、还是软件验证，得到的采样频率均为 50kHz 左右，与理论值误差极大。后来，在第四次实验课开始，老师提示程序代码有误，修改 AdcRegs.ADCTRL1.bit.CONT_RUN=1;语句的值为 0 后，验证的采样频率与理论值较符合。

3.graph 图形工具绘制的波形不正确

在使用 graph 工具绘图时，绘制的波形如图 5. 1 所示，反复修改参数后仍无法解决。后来在老师的提示下，我意识到是由于 graph 观察的位置不对，即在实时处理的过程中观察了波形，而此时可能存在后采样的数据覆盖前采样得到的数据而产生波形重叠的现象。在位置指示变量 ConvCount 赋 0 语句处设置断点，运行后可以绘制比较美观的波形。

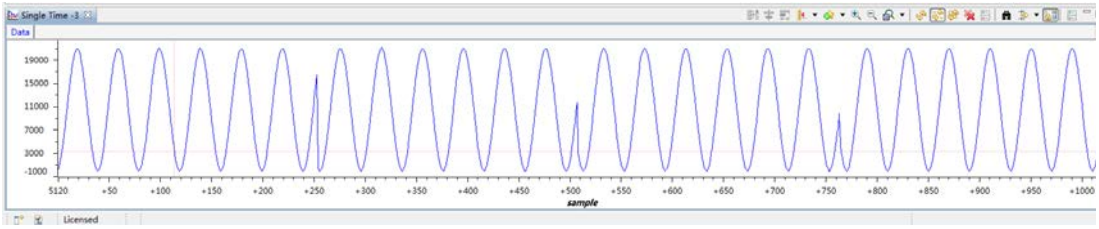

图 5. 1 错误的绘图结果

5.2 实验心得体会

这次实验中，由于范例程序中的代码差异使我们花费的时间略久，但操作上却也越发熟练了。

这次的实验，需要对 DSP 实验箱中 AD 的部分进行配置，修改其采样频率。程序范例中有相关的注释，通过查阅课件，了解修改采样频率的原理后，在实践上加以验证。

在这次实验中，我们采用的硬件验证方案是在每次中断进行过程中，交替的对 DA 赋高低电平，则示波器上测得的方波频率为采样频率的一半。事实上，在硬件验证采样频率时，还有另一种方案，在进入中断程序的一开始给 DA 高电平，中断中的程序照常执行、但不赋值给 DA，中断程序的最后一条语句给 DA 低电平。在这种方案下，产生的方波周期就是采样频率的周期。同时，还可以用这种方法验证程序的实时性（高电平的持续时间是否足够短）。

这些方法，相信都能为最后一个实验，FIR 滤波器的设计打好基础。