

## 实验三 快速傅立叶变换(FFT)算法应用实验

### 一、实验目的

1. 了解 DSP 的定时器。
2. 了解 DSP 片内 AD 的控制方法。
3. 了解 DSP 片内 DA（选做）。
4. 掌握 FFT 的应用。

### 二、实验设备

计算机

ICETEK-F28335-AF 实验箱（带 ICETEK 仿真器）

示波器

### 三、实验原理

#### 1. 片内自带模数转换模块特性

12 位模数转换模块 ADC；

快速转换功能，时钟周期为 25MHz，最小采样带宽为 12.5MSPS；

16 个模拟输入通道（AIN0—AIN15）。

内置双采样-保持器

模拟输入电压范围：0-3v，**切记输入 ad 的信号不要超过这个范围，否则会烧坏芯片。**

#### 2. 模数模块介绍

ADC 模块有 16 个通道，可配置为两个独立的 8 通道模块以方便为事件管理器 A 和 B 服务。两个独立的 8 通道模块可以级连组成 16 通道模块。虽然有多个输入通道和两个序列器，但在 ADC 内部只有一个转换器，同一时刻只有 1 路 AD 进行转换数据。

#### 3. 模数转换的程序控制

模数转换相对于计算机来说是一个较为缓慢的过程。一般采用中断方式启动转换或保存结果，这样在 CPU 忙于其他工作时可以少占用处理时间。设计转换程序应首先考虑处理过程如何与模数转换的时间相匹配，根据实际需要选择适当的触发转换的手段，也要能及时地保存结果。

#### 4. FFT 的原理和参数生成公式

$$x(k) = \sum_{r=0}^{\frac{N}{2}-1} x_1(r) W_N^{rk} + W_N^k \sum_{r=0}^{\frac{N}{2}-1} x_2(r) W_N^{rk} = X_1(k) + W_N^k X_2(k)$$

公式（1）FFT 运算公式

FFT 并不是一种新的变换，它是离散傅立叶变换（DFT）的一种快速算法。由于我们在计算 DFT 时一次复数乘法需用四次实数乘法和二次实数加法；一次复数加法则需二次实数加法。每运算一个  $X(k)$  需要  $4N$  次复数乘法及  $2N+2(N-1)=2(2N-1)$  次实数加法。所以整个 DFT 运算总共需要  $4N^2$  次实数乘法和  $N*2(2N-1)=2N(2N-1)$  次实数加法。如此一来，计算时乘法次数和加法次数都是和  $N^2$  成正比的，当  $N$  很大时，运算量是可观的，因而需要改进对 DFT 的算法减少运算速度。

根据傅立叶变换的对称性和周期性，我们可以将 DFT 运算中有些项合并。

我们先设序列长度为  $N=2^L$ ,  $L$  为整数。将  $N=2^L$  的序列  $x(n)(n=0,1,\dots, N-1)$ , 按  $N$  的奇偶分成两组, 也就是说我们将一个  $N$  点的 DFT 分解成两个  $N/2$  点的 DFT, 他们又从而新组合成一个如下式所表达的  $N$  点 DFT:

一般来说, 输入被假定为连续的。当输入为纯粹的实数的时候, 我们就可以利用左右

$$x(k) = \sum_{r=0}^{\frac{N}{2}-1} x_1(r) W_N^{rk} + W_N^k \sum_{r=0}^{\frac{N}{2}-1} x_2(r) W_N^{rk} = X_1(k) + W_N^k X_2(k)$$

对

称的特性更好的计算 DFT。

我们称这样的 RFFT 优化算法是包装算法: 首先  $2N$  点实数的连续输入称为“进包”。其次  $N$  点的 FFT 被连续被运行。最后作为结果产生的  $N$  点的合成输出是“打开”成为最初的与 DFT 相符合的  $2N$  点输入。

使用这战略, 我们可以划分 FFT 的大小, 它有一半花费在包装输入  $O(N)$  的操作和打开输出上。这样的 RFFT 算法和一般的 FFT 算法同样迅速, 计算速度几乎都达到了两次 DFT 的连续输入。下列一部分将描述更多的在 TMS320C54x 上算法和运行的细节。

## 四、实验内容

1. 基本内容: (完成 8 分, 不能完成视情况依内容给定时器 2 分、AD 转换 2 分、FFT 变换 2 分, 综合 2 分)

采集正弦、方波和三角波信号, 并对其进行 FFT 变换, 需要良好的人机交互界面。

完成此项基本内容, 需要首先分别完成如下内容: (1) 定时器调试正常, 定时时间正确; (2) AD 采集正常, 数据正确; (3) FFT 变换正确。最后将几项内容综合在一起, 完成对信号的 FFT 变换。

2. 拓展内容

拓展内容为研究内容, 如: 采样周期变化对 FFT 结果的影响; 加窗类型对 FFT 结果的影响; 你在实验过程中发现的需要研究的问题等。

## 五、实验程序流程图

实验流程需要根据自己的实验画出。

## 六. 实验步骤

(1) 准备硬件: 按照实验一准备实验箱的硬件连接。

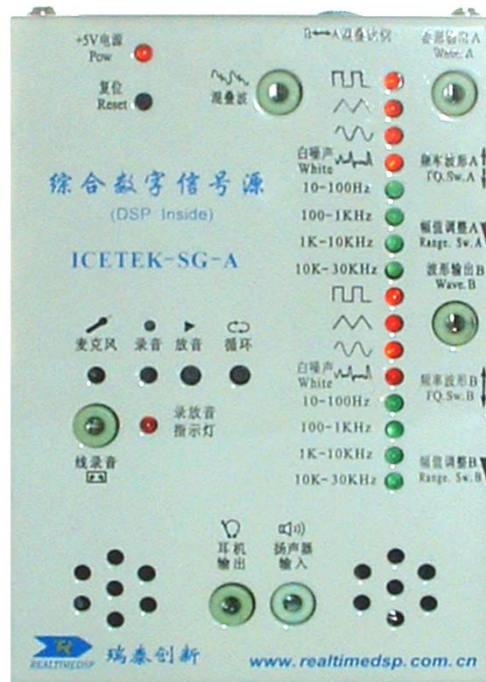
(2) 调试定时器程序。

(3) 熟悉实验箱里信号源的设置

向内侧按波形频率选择旋钮, 相应波形的指示灯点亮。

上下调节波形频率选择旋钮, 相应频率范围的指示灯点亮。

调节幅值调整旋钮, 可调整输出波形的幅值, 通过示波器可观察波形的各项参数。



(4) 将信号源输出连接至 AD 输入，编写 AD 程序，调试 AD 程序 ( )

①取出 1 根实验箱附带的信号线 (如下图，两端均为双声道语音插头)。



②用 1 根信号线连接实验箱左侧信号源的波形输出 A 端口和“ADC-Ch.A”插座注意插头要插牢、到底。这样，信号源波形输出 A 的输出波形即可送到 AD 输入通道 0。

(5) 编写 FFT 程序，调试 FFT 程序

可以先用仿真调试，再加硬件。

(6) 检查结果

## 七、例程

### 1. 定时器例程

```
#include "DSP2833x_Device.h"    // DSP2833x Headerfile Include File
#include "DSP2833x_Examples.h"  // DSP2833x Examples Include File

// Prototype statements for functions found within this file.
interrupt void cpu_timer0_isr(void);
//interrupt void cpu_timer1_isr(void);
//interrupt void cpu_timer2_isr(void);

// #define mem (*(unsigned short int *)0x200000)
#define LED (*(unsigned short int *)0x180000)
#define startCpuTimer0() CpuTimer0Regs.TCR.bit.TSS=0
int i=0, ncount;
unsigned int uLBD;
```

```

void main(void)
{

// Step 1. Initialize System Control:
// PLL, WatchDog, enable Peripheral Clocks
// This example function is found in the DSP2833x_SysCtrl.c file.
    InitSysCtrl();

// Step 2. Initialize GPIO:
// This example function is found in the DSP2833x_Gpio.c file and
// illustrates how to set the GPIO to it's default state.
// InitGpio(); // Skipped for this example
    InitXintf16Gpio(); //zq

// Step 3. Clear all interrupts and initialize PIE vector table:
// Disable CPU interrupts
    DINT;

// Initialize the PIE control registers to their default state.
// The default state is all PIE interrupts disabled and flags
// are cleared.
// This function is found in the DSP2833x_PieCtrl.c file.
    InitPieCtrl();

// Disable CPU interrupts and clear all CPU interrupt flags:
    IER = 0x0000;
    IFR = 0x0000;

// Initialize the PIE vector table with pointers to the shell Interrupt
// Service Routines (ISR).
// This will populate the entire table, even if the interrupt
// is not used in this example. This is useful for debug purposes.
// The shell ISR routines are found in DSP2833x_DefaultIsr.c.
// This function is found in DSP2833x_PieVect.c.
    InitPieVectTable();

// Interrupts that are used in this example are re-mapped to
// ISR functions found within this file.
    EALLOW; // This is needed to write to EALLOW protected registers
    PieVectTable.TINT0 = &cpu_timer0_isr;
    //PieVectTable.XINT13 = &cpu_timer1_isr;
    //PieVectTable.TINT2 = &cpu_timer2_isr;
    EDIS; // This is needed to disable write to EALLOW protected registers

```

```

// Step 4. Initialize the Device Peripheral. This function can be
//          found in DSP2833x_CpuTimers.c
    InitCpuTimers(); // For this example, only initialize the Cpu Timers

#if (CPU_FRQ_150MHZ)
// Configure CPU-Timer 0, 1, and 2 to interrupt every second:
// 150MHz CPU Freq, 1 second Period (in uSeconds)

    ConfigCpuTimer(&CpuTimer0, 150, 1000000);
    //ConfigCpuTimer(&CpuTimer1, 150, 1000000);
    //ConfigCpuTimer(&CpuTimer2, 150, 1000000);
#endif

#if (CPU_FRQ_100MHZ)
// Configure CPU-Timer 0, 1, and 2 to interrupt every second:
// 100MHz CPU Freq, 1 second Period (in uSeconds)

    ConfigCpuTimer(&CpuTimer0, 100, 1000000);
    //ConfigCpuTimer(&CpuTimer1, 100, 1000000);
    //ConfigCpuTimer(&CpuTimer2, 100, 1000000);
#endif

// To ensure precise timing, use write-only instructions to write to the entire
// register. Therefore, if any
// of the configuration bits are changed in ConfigCpuTimer and InitCpuTimers
// (in DSP2833x_CpuTimers.h), the
// below settings must also be updated.

    //CpuTimer0Regs.TCR.all = 0x4001; // Use write-only instruction to set TSS
    bit = 0
    //CpuTimer1Regs.TCR.all = 0x4001; // Use write-only instruction to set TSS
    bit = 0
    //CpuTimer2Regs.TCR.all = 0x4001; // Use write-only instruction to set TSS
    bit = 0

// Step 5. User specific code, enable interrupts:
//CpuTimer0Regs.PRD.all=0xffff;
CpuTimer0Regs.TPR.all=0;
CpuTimer0Regs.TIM.all=0;
CpuTimer0Regs.TPRH.all=0;
CpuTimer0Regs.TCR.bit.TSS=1;
CpuTimer0Regs.TCR.bit.SOFT=1;
CpuTimer0Regs.TCR.bit.FREE=1;
CpuTimer0Regs.TCR.bit.TRB=1;
CpuTimer0Regs.TCR.bit.TIE=1;

```

```

CpuTimer0.InterruptCount=0;
startCpuTimer0();

// Enable CPU int1 which is connected to CPU-Timer 0, CPU int13
// which is connected to CPU-Timer 1, and CPU int 14, which is connected
// to CPU-Timer 2:
IER |= M_INT1;
//IER |= M_INT13;
//IER |= M_INT14;

// Enable TINT0 in the PIE: Group 1 interrupt 7
PieCtrlRegs.PIEIER1.bit.INTx7 = 1;

// Enable global Interrupts and higher priority real-time debug events:
EINT;    // Enable Global interrupt INTM
ERTM;    // Enable Global realtime interrupt DBGM

// Step 6. IDLE loop. Just sit and loop forever (optional):
for(;;)
{
    }

}

interrupt void cpu_timer0_isr(void)
{
    CpuTimer0.InterruptCount++;

    // Acknowledge this interrupt to receive more interrupts from group 1
    PieCtrlRegs.PIEACK.all = PIEACK_GROUP1;
    CpuTimer0Regs.TCR.bit.TIF=1;
    CpuTimer0Regs.TCR.bit.TRB=1;
    if(ncount==0)
    {
        LED=uLBD;
        uLBD++;uLBD%=16;
    }
    ncount++;ncount%=194;
}

```

## 2. AD 例程

```

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

```

```

#include "DSP2833x_Examples.h" // DSP2833x Examples Include File

// Determine when the shift to right justify the data takes place
// Only one of these should be defined as 1.
// The other two should be defined as 0.
#define POST_SHIFT 0 // Shift results after the entire sample table is full
#define INLINE_SHIFT 1 // Shift results as the data is taken from the results
register
#define NO_SHIFT 0 // Do not shift the results

// ADC start parameters
#if (CPU_FRQ_150MHZ) // Default - 150 MHz SYSCLKOUT
    #define ADC_MODCLK 0x3 // HSPCLK = SYSCLKOUT/2*ADC_MODCLK2 = 150/(2*3) =
    25.0 MHz
#endif
#if (CPU_FRQ_100MHZ)
    #define ADC_MODCLK 0x2 // HSPCLK = SYSCLKOUT/2*ADC_MODCLK2 = 100/(2*2) =
    25.0 MHz
#endif
#define ADC_CKPS 0x0 // ADC module clock = HSPCLK/1 = 25.5MHz/(1) =
    25.0 MHz
#define ADC_SHCLK 0x1 // S/H width in ADC module periods =
    2 ADC cycle
#define AVG 1000 // Average sample limit
#define ZOFFSET 0x00 // Average Zero offset
#define BUF_SIZE 1024 // Sample buffer size

// Global variable for this example
Uint16 SampleTable[BUF_SIZE];
Uint16 SampleTable1[BUF_SIZE];
Uint16 AD0[64];

main()
{
    Uint16 i;
    Uint16 j,k;
    Uint16 array_index;
    Uint16 array_index1;

    // Step 1. Initialize System Control:
    // PLL, WatchDog, enable Peripheral Clocks
    // This example function is found in the DSP2833x_SysCtrl.c file.
    InitSysCtrl();

```

```

// Specific clock setting for this example:
EALLOW;
SysCtrlRegs.HISPCP.all = ADC_MODCLK;    // HSPCLK = SYSCLKOUT/ADC_MODCLK
EDIS;

// Step 2. Initialize GPIO:
// This example function is found in the DSP2833x_Gpio.c file and
// illustrates how to set the GPIO to it's default state.
// InitGpio(); // Skipped for this example
// Enable the pin GPIO34 as output
EALLOW;
GpioCtrlRegs.GPBMUX1.bit.GPIO34 = 0;    // GPIO pin
GpioCtrlRegs.GPBDIR.bit.GPIO34 = 1;    // Output pin
EDIS;

// Step 3. Clear all interrupts and initialize PIE vector table:
// Disable CPU interrupts
DINT;

// Initialize the PIE control registers to their default state.
// The default state is all PIE interrupts disabled and flags
// are cleared.
// This function is found in the DSP2833x_PieCtrl.c file.
InitPieCtrl();

// Disable CPU interrupts and clear all CPU interrupt flags:
IER = 0x0000;
IFR = 0x0000;

// Initialize the PIE vector table with pointers to the shell Interrupt
// Service Routines (ISR).
// This will populate the entire table, even if the interrupt
// is not used in this example. This is useful for debug purposes.
// The shell ISR routines are found in DSP2833x_DefaultIsr.c.
// This function is found in DSP2833x_PieVect.c.
InitPieVectTable();

// Step 4. Initialize all the Device Peripherals:
// This function is found in DSP2833x_InitPeripherals.c
// InitPeripherals(); // Not required for this example
InitAdc();           // For this example, init the ADC

// Specific ADC setup for this example:
AdcRegs.ADCCTRL1.bit.ACQ_PS = ADC_SHCLK; // Sequential mode: Sample rate =

```



```

1/[(2+ACQ_PS)*ADC clock in ns]
//                                     = 1/(3*40ns) =8.3MHz (for 150 MHz
SYSCLKOUT)
//                                     = 1/(3*80ns) =4.17MHz (for 100 MHz
SYSCLKOUT)
// If Simultaneous mode enabled: Sample rate =
1/[(3+ACQ_PS)*ADC clock in ns]
  AdcRegs.ADCCTRL3.bit.ADCCLKPS = ADC_CKPS;
  AdcRegs.ADCCTRL1.bit.SEQ_CASC = 1;      // 1 Cascaded mode
  //AdcRegs.ADCCHSELSEQ1.bit.CONV00 = 0xf;
  //AdcRegs.ADCCHSELSEQ1.bit.CONV01 = 0x0;
  AdcRegs.ADCCHSELSEQ1.bit.CONV02 = 0x2;
  AdcRegs.ADCCHSELSEQ1.bit.CONV03 = 0x3;
  AdcRegs.ADCCTRL1.bit.CONT_RUN = 1;      // Setup continuous run

  AdcRegs.ADCCTRL1.bit.SEQ_OVRD = 1;      // Enable Sequencer override feature
  // AdcRegs.ADCCHSELSEQ1.all = 0x320f;    // Initialize all ADC channel
selects to A2,A3
  // AdcRegs.ADCCHSELSEQ2.all = 0x0;
  // AdcRegs.ADCCHSELSEQ3.all = 0x0;
  //AdcRegs.ADCCHSELSEQ4.all = 0x0;
  AdcRegs.ADCMAXCONV.bit.MAX_CONV1 = 0x1; // convert and store in 8 results
registers

// Step 5. User specific code, enable interrupts:

// Clear SampleTable
for (i=0; i<BUF_SIZE; i++)
{
  SampleTable[i] = 0;
  SampleTable1[i] = 0;
}

for(i=0;i<64;i++)
  AD0[i]=0;
// Start SEQ1
AdcRegs.ADCCTRL2.all = 0x2000;

for(;;)
{
  array_index = 0;
  array_index1 = 0;

```

```

    for (i=0; i<(BUF_SIZE); i++)
    {
        // Wait for int1
        while (AdcRegs.ADCST.bit.INT_SEQ1== 0){}
        GpioDataRegs.GPBSET.bit.GPIO34 = 1; // Set GPIO34 for monitoring
        -optional

        AdcRegs.ADCST.bit.INT_SEQ1_CLR = 1;

        SampleTable[array_index++]= ( (AdcRegs.ADCRESULT2)>>4);
        SampleTable1[array_index1++]= ( (AdcRegs.ADCRESULT3)>>4);
        for(j=0;j<100;j++)
            k++;
    }

    for (i=0; i<BUF_SIZE; i++)
    {
        SampleTable[i] = ((SampleTable[i]) >>4);
        SampleTable1[i] = ((SampleTable1[i]) >>4);
    }

    GpioDataRegs.GPBCLEAR.bit.GPIO34 = 1; // Break point
}
}

```

### 3. FFT例程

```

#include"math.h"
#define PI 3.1415926
#define SAMPLENUMBER 128

void InitForFFT();
void MakeWave();
//void FFT(float dataR[SAMPLENUMBER],float dataI[SAMPLENUMBER]);

int INPUT[SAMPLENUMBER],DATA[SAMPLENUMBER];
float fWaveR[SAMPLENUMBER],fWaveI[SAMPLENUMBER],w[SAMPLENUMBER];
float sin_tab[SAMPLENUMBER],cos_tab[SAMPLENUMBER];

void FFT(float dataR[SAMPLENUMBER],float dataI[SAMPLENUMBER])
{
    int x0,x1,x2,x3,x4,x5,x6,xx;
    int i,j,k,b,p,L;

```

```

float TR, TI, temp;

/***** following code invert sequence *****/
for ( i=0; i<SAMPLENUMBER; i++ )
{
    x0=x1=x2=x3=x4=x5=x6=0;
    x0=i&0x01; x1=(i/2)&0x01; x2=(i/4)&0x01; x3=(i/8)&0x01; x4=(i/16)&0x01;
x5=(i/32)&0x01; x6=(i/64)&0x01;
    xx=x0*64+x1*32+x2*16+x3*8+x4*4+x5*2+x6;
    dataI[xx]=dataR[i];
}
for ( i=0; i<SAMPLENUMBER; i++ )
{
    dataR[i]=dataI[i]; dataI[i]=0;
}

/***** following code FFT *****/
for ( L=1; L<=7; L++ )
{ /* for(1) */
    b=1; i=L-1;
    while ( i>0 )
    {
        b=b*2; i--;
    } /* b= 2^(L-1) */
    for ( j=0; j<=b-1; j++ ) /* for (2) */
    {
        p=1; i=7-L;
        while ( i>0 ) /* p=pow(2,7-L)*j; */
        {
            p=p*2; i--;
        }
        p=p*j;
        for ( k=j; k<128; k=k+2*b ) /* for (3) */
        {
            TR=dataR[k]; TI=dataI[k]; temp=dataR[k+b];

dataR[k]=dataR[k]+dataR[k+b]*cos_tab[p]+dataI[k+b]*sin_tab[p];

dataI[k]=dataI[k]-dataR[k+b]*sin_tab[p]+dataI[k+b]*cos_tab[p];
            dataR[k+b]=TR-dataR[k+b]*cos_tab[p]-dataI[k+b]*sin_tab[p];
            dataI[k+b]=TI+temp*sin_tab[p]-dataI[k+b]*cos_tab[p];
        } /* END for (3) */
    } /* END for (2) */
} /* END for (1) */

```

```

    for ( i=0;i<SAMPLENUMBER/2;i++ )
    {
        w[i]=sqrt(dataR[i]*dataR[i]+dataI[i]*dataI[i]);
    }
} /* END FFT */

```

```

main()
{
    int i;

    InitForFFT();
    MakeWave();
    for ( i=0;i<SAMPLENUMBER;i++ )
    {
        fWaveR[i]=INPUT[i];
        fWaveI[i]=0.0f;
        w[i]=0.0f;
    }
    FFT(fWaveR,fWaveI);
    for ( i=0;i<SAMPLENUMBER;i++ )
    {
        DATA[i]=w[i];
    }
    while ( 1 );    // break point
}

```

```

void InitForFFT()
{
    int i;

    for ( i=0;i<SAMPLENUMBER;i++ )
    {
        sin_tab[i]=sin(PI*2*i/SAMPLENUMBER);
        cos_tab[i]=cos(PI*2*i/SAMPLENUMBER);
    }
}

```

```

void MakeWave()
{
    int i;

    for ( i=0;i<SAMPLENUMBER;i++ )

```

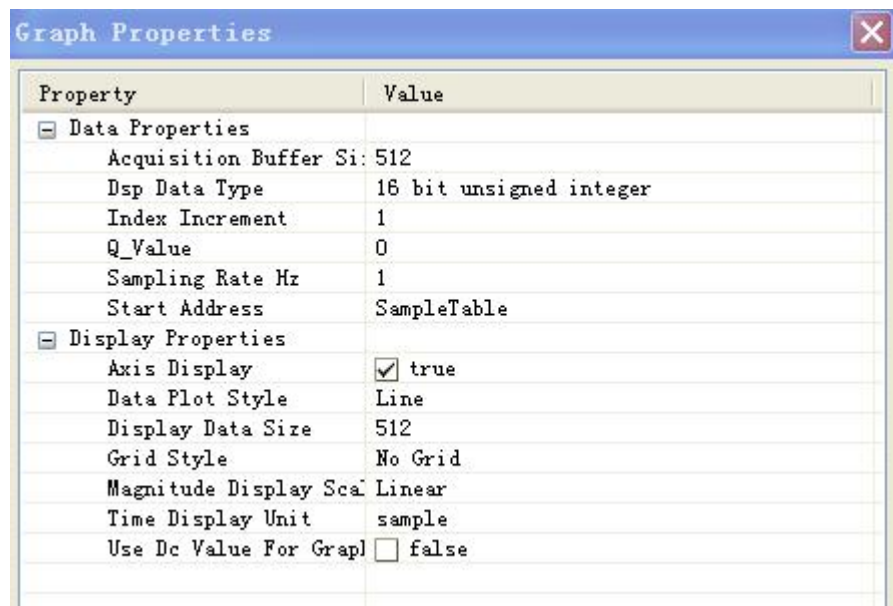
```

{
    INPUT[i]=sin(PI*2*i/SAMPLENUMBER*3)*1024;
}
}

```

## 八、观察窗口

1. 选择菜单 Tools->Graph->Single Time 做如下设置，然后单击“OK”按钮：




2. 通过设置，可打开图形窗口观察模数转换和 FFT 转换的结果。

7. 运行程序观察结果

点击菜单 Run->Resume，运行程序，或者直接点击  按钮；

-按  停止运行，观察窗口中的图形显示；

注：有时候 CCS 显示会出现问题，例如停止程序后，图形观察窗口显示空白或者混乱的

波形：可以点击一下按钮  (Reset the Graph),复位图形观察窗口。然后再次运行程序，停止程序。

适当改变信号源，再次运行，停止后观察图形窗口中的显示。