使用 CCS 进行 DSP 编程（二）

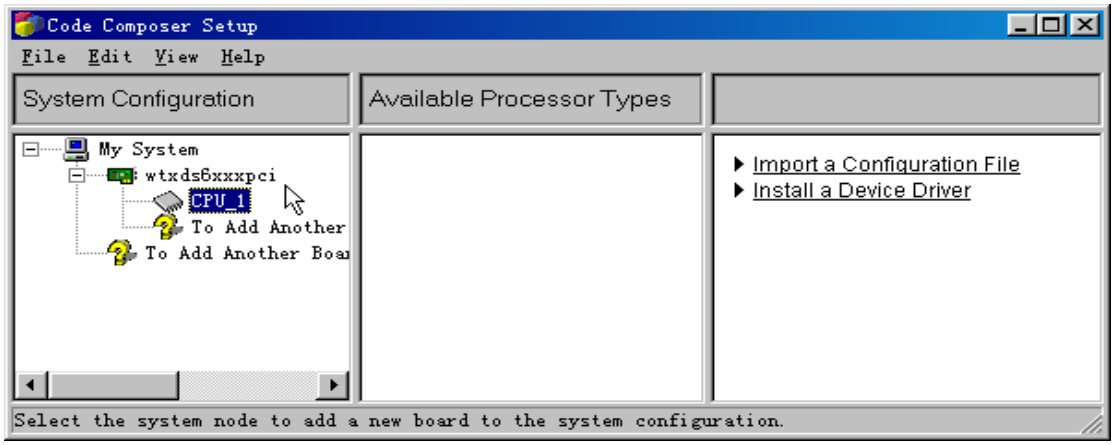
——实现 FFT

pacificxu

现在讨论使用 TI 公司的 CCS 进行 DSP 编程，首先假定读者对 CCS 的使用已经比较了解，如果读者还不太了解，请参阅《使用 CCS 进行 DSP 编程（一）——CCS 编程入门》及其他 CCS 的学习文档。

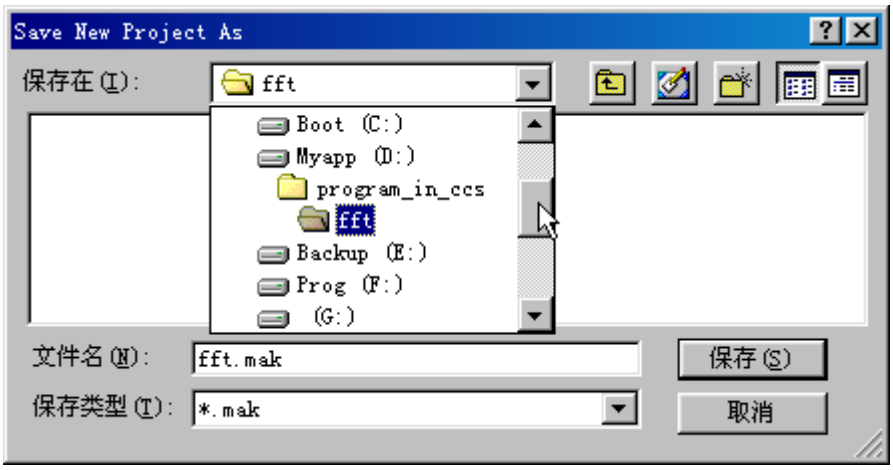
作数字信号处理的同志们总是喜欢用 FFT 来对信号处理系统做检验，下面用闻亭公司的 C6xP 板、C6xPa 板硬件实现 FFT 算法，本算法对其他的 C6x 板同样实用（只是硬件资源稍微有差异），并通过闻亭公司的 PCI 仿真器对目标板加载运行，运行结果在 CCS 中可视化显示。

首先启动 CCS Setup，对仿真器硬件进行设置，本人使用的是闻亭公司 PCI 的仿真器自带的驱动 wtxds6xxxpci.dvr，设置画面如下：

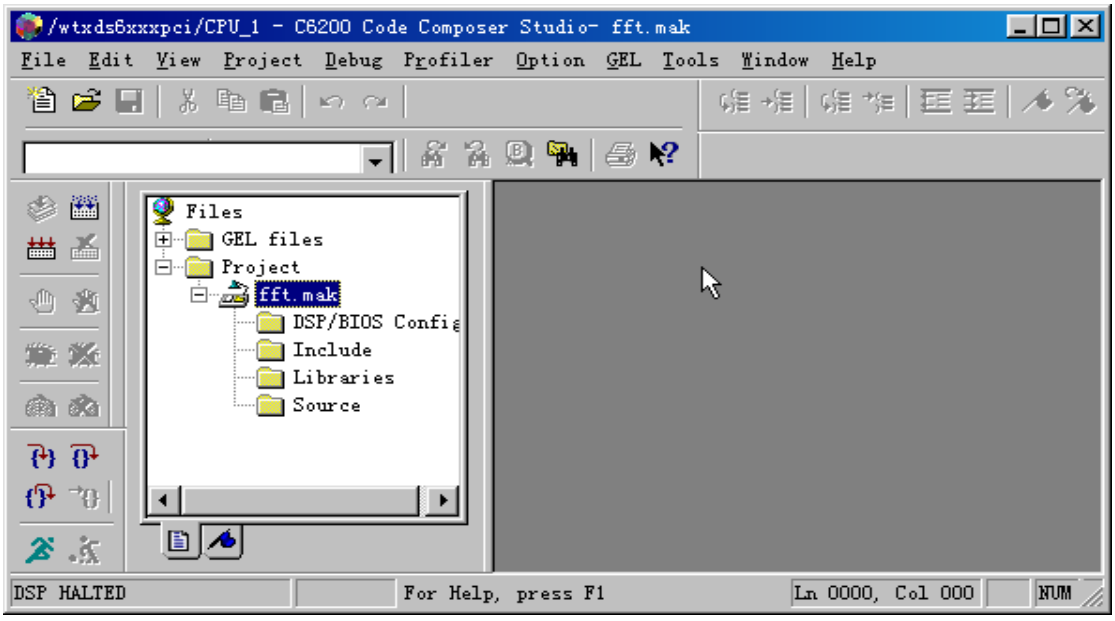


下面就可以运行 CCS 了，在 CCS 中，创建一个新的 Project，

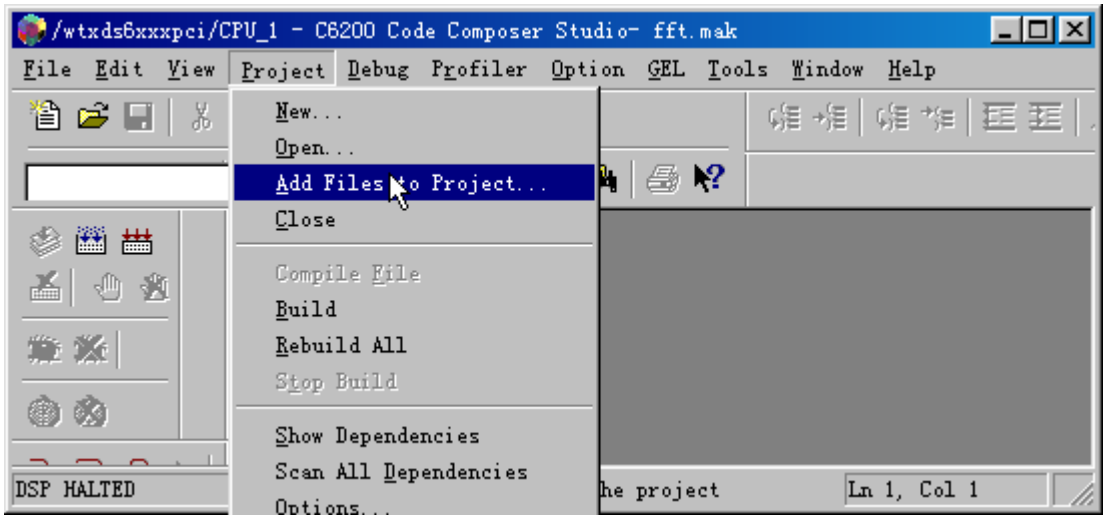
我的工程文件放置在如下的目录中，读者可以放在自己喜欢的目录下：



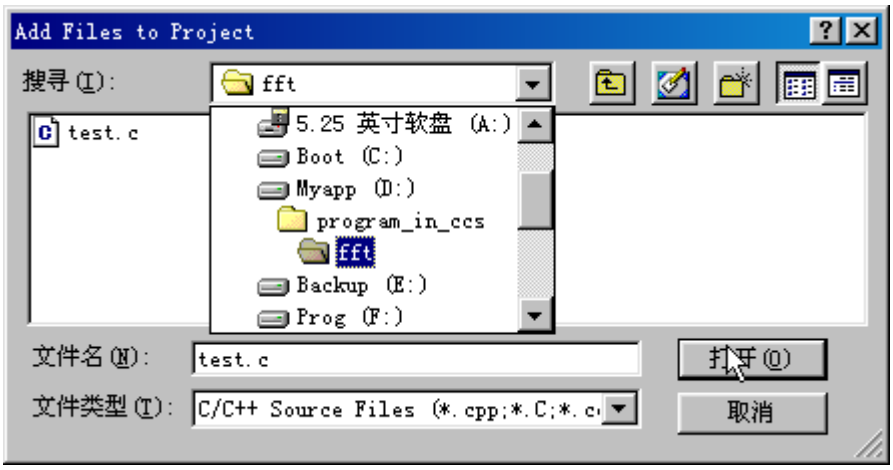
双击“ + ”展开 fft.mak，可以看到整个工程文件是空的，



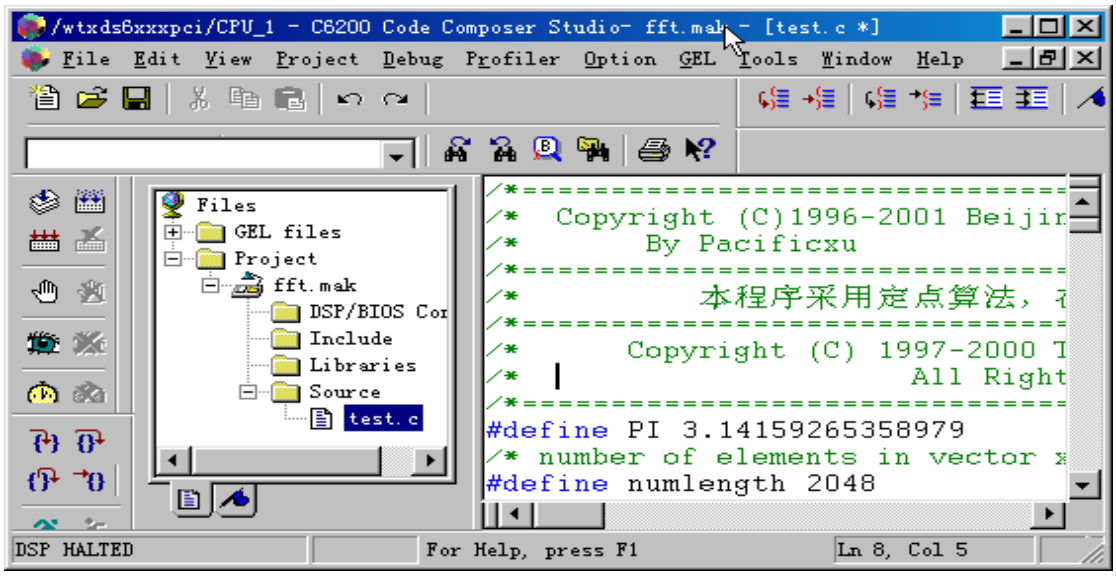
我们需要把*.c、*.cmd、*.lib 文件添加到工程文件中，



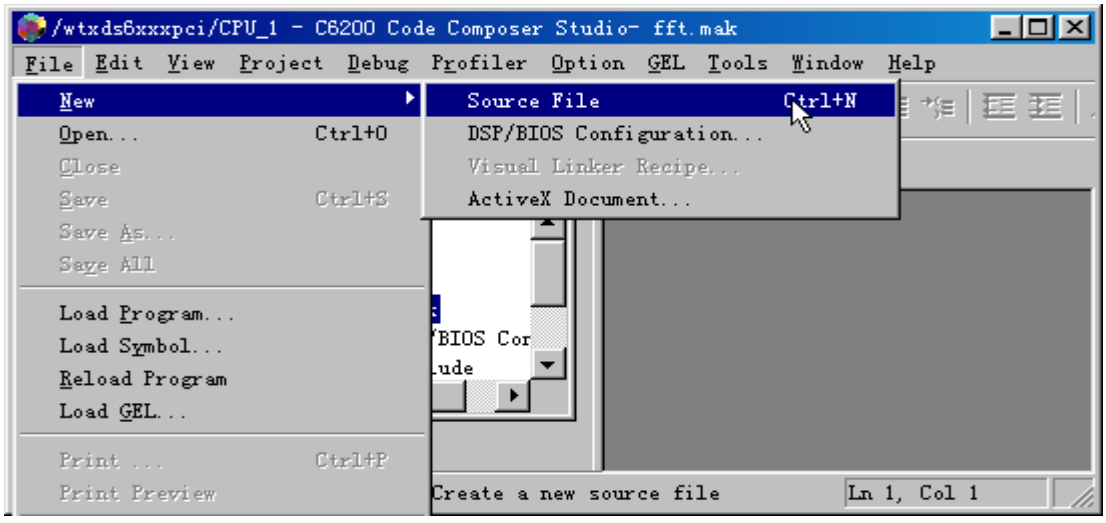
首先是*.c 文件，本例中是 test.c 文件，



同样的方法可以用来添加其他的文件。双击工程中的源文件，会在右边的窗口中看见原码：



如果*.c 文件不存在，可以在 CCS 集成开发环境中生成，



本例中 test.c 的主程序源代码如下：

```
/*============================================================================*/
/*  Copyright (C)1996-2001 Beijing Wintech Technology Co.,Ltd.                */
/*            By Pacificxu,          All Rights Reserved                      */
/*                        本程序采用定点算法，在浮点DSP芯片如6701上同样实用           */
/*============================================================================*/
#define PI 3.14159265358979
/* number of elements in vector x*/
#define numlength 2048
#include <C:\ti\c6000\cgtools\include\math.h>

/*声明函数*/
 void radix2(int n, short *xy, short *w);
 void bitrev_cplx(int *x, short *index, int nx);
 void digitrev_index(short *index, int n, int radix);

/*定义全局变量*/
static short index1[64],w1[numlength];
static short x1[numlength*2];
static int x2[numlength];
static int radix;
int nx1;
int i;

/*主函数开始*/
main()
{

  double delta;

  radix=2;                    /*基数——基2FFT*/
  nx1=numlength;              /*FFT计算点数*/

      /* init the index for fft bitrev */
  digitrev_index(index1,nx1, radix);

      /*init the FFT coefficients*/
  delta=2*PI/nx1;
      for (i=0;i<nx1/2;i++){
              w1[2*i]=32767*(-cos((double)i*delta));
              w1[2*i+1]=32767*(-sin((double)i*delta));
      }

/*构造要做FFT的序列，实部0、虚部0，实部1、虚部1...顺序构造，本例中实部为三个频率的信号，虚部为0*/
  for (i=0;i<nx1;i++){
              x1[2*i]=(short)((cos(PI*i/10.0)+cos(PI*i/20.0)+cos(PI*i/5.0))*0x80);
              x1[2*i+1]=0;
   }

/*做FFT算法*/
  radix2(nx1, x1, w1);
  for (i=0;i<nx1;i++){
              x2[i]=sqrt(x1[2*i]*x1[2*i]+x1[2*i+1]*x1[2*i+1]);
  }


    /*倒序*/
    bitrev_cplx(x2, index1, nx1);

}
```

调用的子程序有三个：

```
/*===========================================================================
/*      Copyright (C) 1997-2000 Texas Instruments Incorporated.
/*                      All Rights Reserved
/*===========================================================================
/*      生成倒序表子程序                                                    */
void digitrev_index(short *index, int n, int radix){
        int i,j,k;
        short nbits, nbot, ntop, ndiff, n2, raddiv2;
        nbits = 0;
        i = n;
        while (i > 1){
                i = i >> 1;
                nbits++;
                }

        raddiv2 = radix >> 1;
        nbot = nbits >> raddiv2;
        nbot = nbot << raddiv2 - 1;
        ndiff = nbits & raddiv2;
        ntop = nbot + ndiff;
        n2 = 1 << ntop;

        index[0] = 0;
        for ( i = 1, j = n2/radix + 1; i < n2 - 1; i++){
                index[i] = j - 1;
                for(k = n2/radix; k*(radix-1) < j;k /= radix)

                        j -= k*(radix-1);

               j += k;
        }
        index[n2 - 1] = n2 - 1;
}

/*           倒序子程序
/*           x    = Input Array to be Bit-Reversed
*           nx   = Number of points in array (must be a power of 2)
*           index = Array of ~sqrt(nx) created by the routine
*                   digitrev_index to allow the fast implementation of the
*                   bit-reversal
*     DESCRIPTION
*           This routine performs the bit-reversal of the input array x[].
*           where x[] is an array of length nx 16-bit complex pairs of data.
*           This requires the index array provided by the program below.
*           This index should be generated at compile time not by the DSP.

void bitrev_cplx(int *x, short *index, int nx)
{
           int     i;
           short       i0, i1, i3;
           short       j0, j1, j3;
           int     xi0, xi1,  xi3;
           int     xj0, xj1, xj3;
```

```
    short       t;
    int     a, b, ia, ib, ibs;
    int     mask;
    int     nbits, nbot, ntop, ndiff, n2, halfn;

    nbits = 0;
    i = nx;
    while (i > 1){
        i = i >> 1;
        nbits++;}

    nbot    = nbits >> 1;
    ndiff   = nbits & 1;
    ntop    = nbot + ndiff;
    n2      = 1 << ntop;
    mask    = n2 - 1;
    halfn   = nx >> 1;

    for (i0 = 0; i0 < halfn; i0 += 2) {
        b   = i0 & mask;
        a   = i0 >> nbot;
        if (!b) ia  = index[a];
        ib  = index[b];
        ibs = ib << nbot;

        j0  = ibs + ia;

        t   = i0 < j0;
        xi0 = x[i0];
        xj0 = x[j0];

        if (t){x[i0] = xj0;
            x[j0] = xi0;}

        i1  = i0 + 1;
        j1  = j0 + halfn;
        xi1 = x[i1];
        xj1 = x[j1];
        x[i1] = xj1;
        x[j1] = xi1;

        i3  = i1 + halfn;
        j3  = j1 + 1;
        xi3 = x[i3];
        xj3 = x[j3];
        if (t){x[i3] = xj3;
            x[j3] = xi3;}
    }
}

/*      频率抽取FFT算法函数体，用户如果对算法感兴趣，请参阅
                胡广书老师的《数字信号处理-理论、算法与实现》 第5章 快速傅立叶变换
/*      xy[] --- input and output sequences (dim-n)      (input/output)
```

```
*        n     --- FFT size                (input)
*        w[]   --- FFT coefficients (dim-n/2)     (input)
*
*
*      DESCRIPTION
*        This routine is used to compute FFT of a complex sequece
*        of size n, a power of 2, with "decimation-in-frequency
*        decomposition" method, ie, the output is in bit-reversed
*        order. Each complex value is with interleaved 16-bit real
*        and imaginary parts.

void radix2(int n, short xy[], short w[])
{
          short n1,n2,ie,ia,i,j,k,l;
          short xt,yt,c,s;

          n2 = n;
          ie = 1;
          for (k=n; k > 1; k = (k >> 1) ) {
            n1 = n2;
            n2 = n2>>1;
            ia = 0;
            for (j=0; j < n2; j++) {
              c = w[2*ia];
              s = w[2*ia+1];
              ia = ia + ie;

              for (i=j; i < n; i += n1) {
                l = i + n2;
                xt       = xy[2*l] - xy[2*i];
                xy[2*i] = xy[2*i] + xy[2*l];
                yt       = xy[2*l+1] - xy[2*i+1];
                xy[2*i+1] = xy[2*i+1] + xy[2*l+1];
                xy[2*l]    = (c*xt + s*yt)>>15;
                xy[2*l+1] = (c*yt - s*xt)>>15;
              }
            }
            ie = ie<<1;
          }
}
```

从上面的源程序可以看到，使用 CCS 的 C 语言编程跟普通的 C 语言编程没有太大的区别，这正是 TI 所追求的，兼容的 ANSI C 标准和如此的编译高效率也正是 TI 的领先之处。读者可以不必学习烦琐的汇编和线性汇编，直接对数字信号处理的算法进行研究，同时享受高速的处理速度，只有在对速度要求极严的条件下，不得不使用汇编和线性汇编，那时读者已经有了一定的基础，再学习汇编语言已是水到渠成。而使用 C 语言编程是大势所趋。

如果有人对算法本身感兴趣，请参阅胡广书老师的《数字信号处理—理论、算法与实现》 第 5 章 快速傅立叶变换，这里不在对算法进行展开讨论。

程序的结构本身很简单，使用过 C 语言的朋友一看就明白，不需要再做进一步的说明，需要指出几点，

1. 本程序中的 math.h 与 Visual C++中的 math.h 是不同的，TI 的 CCS 专门为数学计算作了运行时库，是利用硬件对计算作加速的，与 Visual C++中的速度是不可同日而语的。因此如果我们需要用到相应的“头文件”，就应该在 TI 的目录中查找，同时要包含相应的运行时库（*.lib）文件，我一直在强调这一点，初学者往往忽略这一点而出现许多编译链接错误。下面的演示中还会看到这一点。
2. 本例是以定点 DSP 芯片‘C6201 为例的。如果对定点运算还不太熟悉，只好找些文档来学习一下了，这里也不再展开。在浮点 DSP 芯片‘C6701 中本程序可以不加修改地运行，但浮点 DSP 芯片中可以直接进行有硬件支持的浮点运算，速度会更快。
3. CCS 的 C 语言中的数据类型是与硬件相关的，使用时需要注意。

   char 8 bits

   short 16 bits

   int 32 bits

   long 40 bits

   float 32 bits

   double 64 bits

   TMS320C6000 Online Programmer's Guide (SPRH048) Copyright?2000 Texas Instruments

接下来继续进行，向工程中添加*.cmd 文件。读者现在应该会执行这个操作了，如果没有就自己造一个吧，也可以拿别人的来改造一下。其中有些不太明白也没有关系，但是在与具体的硬件相关的地方还是要搞明白。

首先要搞明白目标板“target”上到底有多少存储空间，包括 DSP 芯片内部有多大空间，目标板上有多少外部存储器（SDRAM、SBSRAM、双口 RAM），以及它们的其始地址和长度，以我使用的闻亭公司‘C6Xpa 板为例，

存储器类型	起始地址	存储器大小		结束地址
SDRAM	0x2000000	4M*32bit	(0x400000*4)	0x3000000
SBSRAM	0x400000	128k*32bit	(0x20000*4)	0x480000
DPRAM	0x1400000	4k*32bit	(0x1000*4)	0x1404000

在这里还要强调一点，是“ *32bit ”，因此 SDRAM 的结束地址是

0x2000000 + 0x400000*4 = 0x3000000

而不是

0x2000000 + 0x400000 = 0x2400000

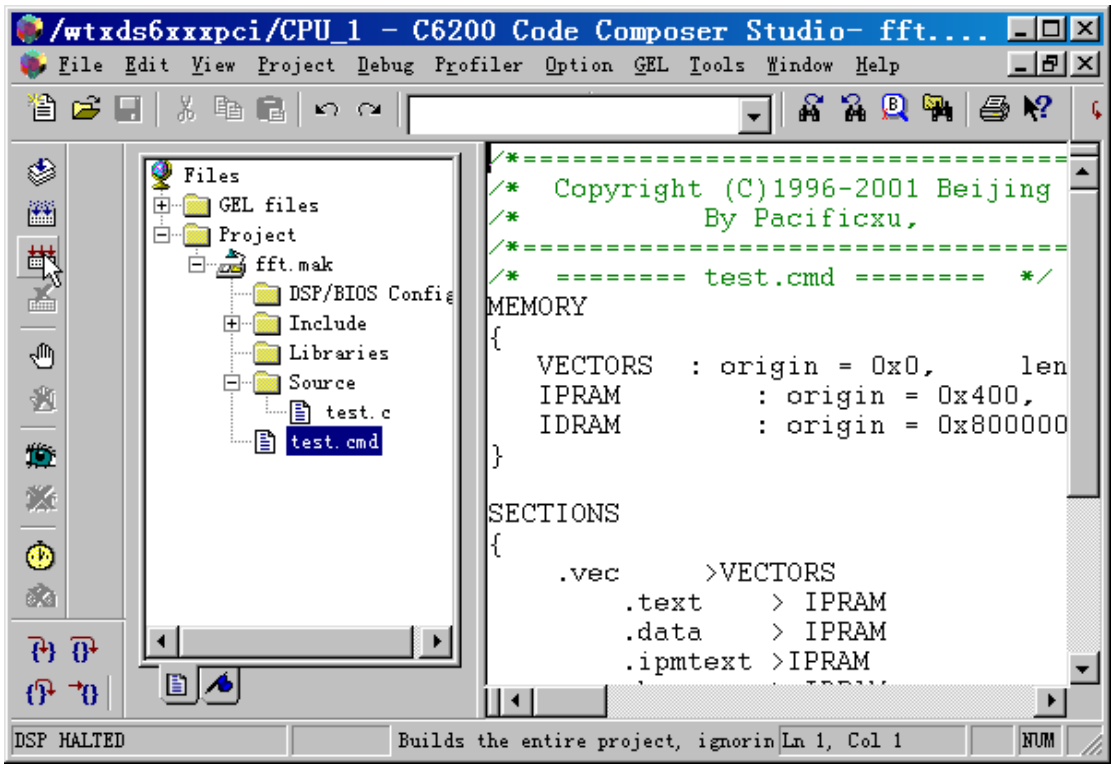
其他存储空间也是如此。许多同志们忽略了这一点，在使用数组、指针及对空间地址操作时造成“ 不可理解 ”的错误，“ 我往么个地址写数怎么找不到，#%^#&^%%^———坏 了！”，好好研究一下，就不好意思这么快下结论了。

下面是我用的*.cmd 文件的源代码，需要的话可以拿去用喔。

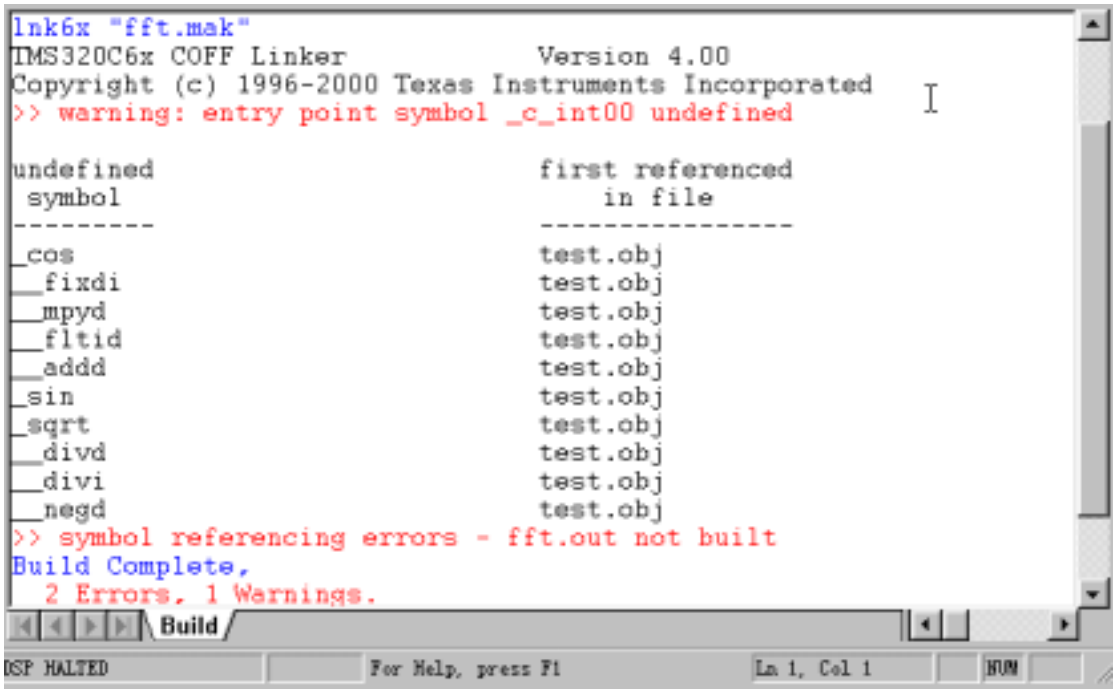
```
/*============================================================
/*  Copyright (C)1996-2001 Beijing Wintech Technology Co.,Ltd.
/*            By Pacificxu,           All Rights Reserved
/*============================================================
/*  ======== test.cmd ========  */
MEMORY
{
   VECTORS      : origin = 0x0,          len = 0x400
   IPRAM        : origin = 0x400,        len = 0xf000
   IDRAM        : origin = 0x80000000,   len = 0x10000
}

SECTIONS
{
        .vec      >VECTORS
        .text     > IPRAM
        .data     > IPRAM
        .ipmtext  >IPRAM
        .bss      > IDRAM
        .cinit    > IDRAM
        .const    > IDRAM
        .far      > IDRAM
        .stack    > IDRAM
        .cio      > IDRAM
        .sysmem   > IDRAM
}
```

里面“ .vec、.text …. ”的东西可以先不考虑，留给 CCS 去处理好了。只要保证定义的空间在物理上存在就可以了。

心急的朋友会开始编译了，



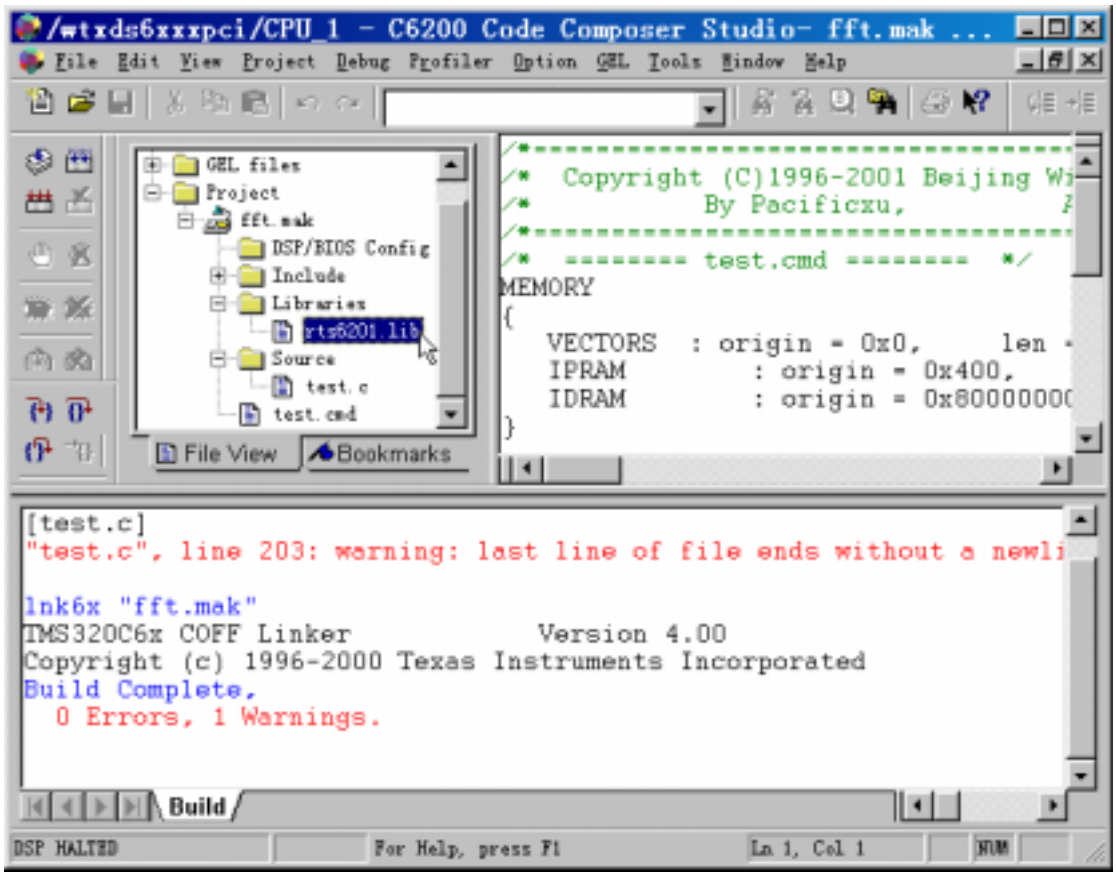
又会出现下面的结果，



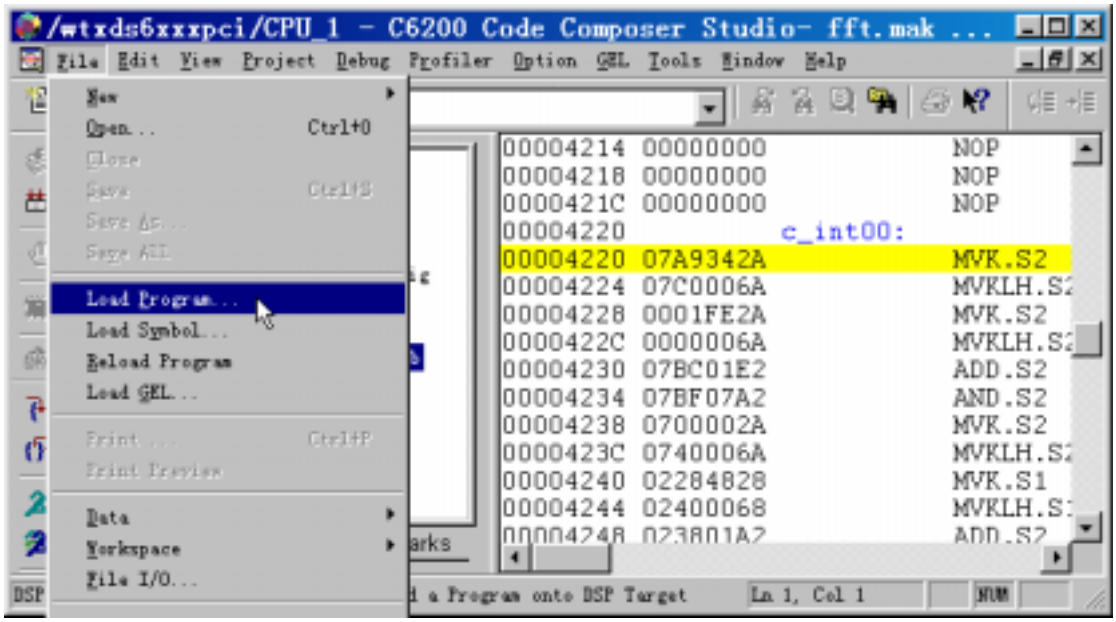
“ $%*^&^，又忘了加运行时库！”。（下次再忘就不应该了 !）。

这里展开一点，有些朋友说：“ 我包含了头文件<intr.h>，调用中断函数intr_hook()时怎么也编译链接通不过，去掉这一句就通过了。”大家现在就应该知

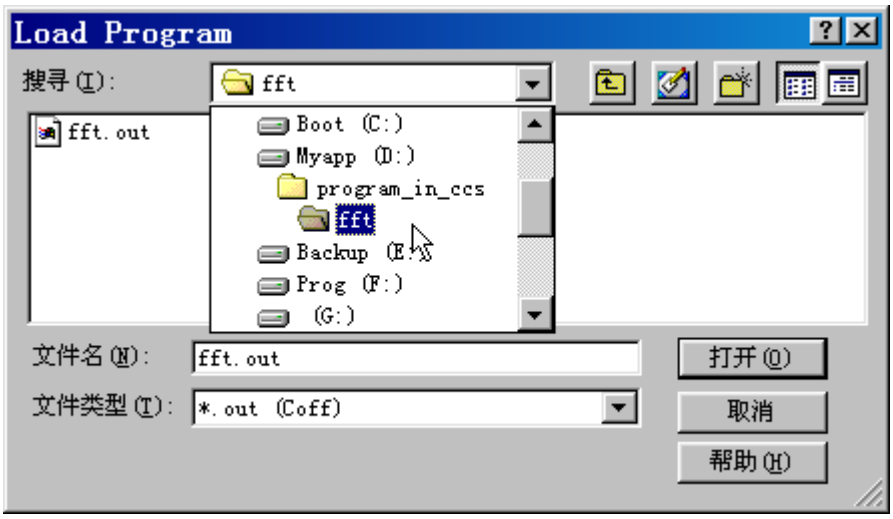
道怎么处理这个问题了吧？熟悉 C 语言编程的同志都知道“ *.lib ”的实质，中断函数 intr_hook()在“ dev6x.lib ”里。加上这个库问题估计该解决了吧！ TI 的运行时库都有具体的说明，如果大家实在懒得去看，又不想知其所以然，有一个绝招在此：“ Find all *.lib in TI 目录 ”一个一个试一下好了。



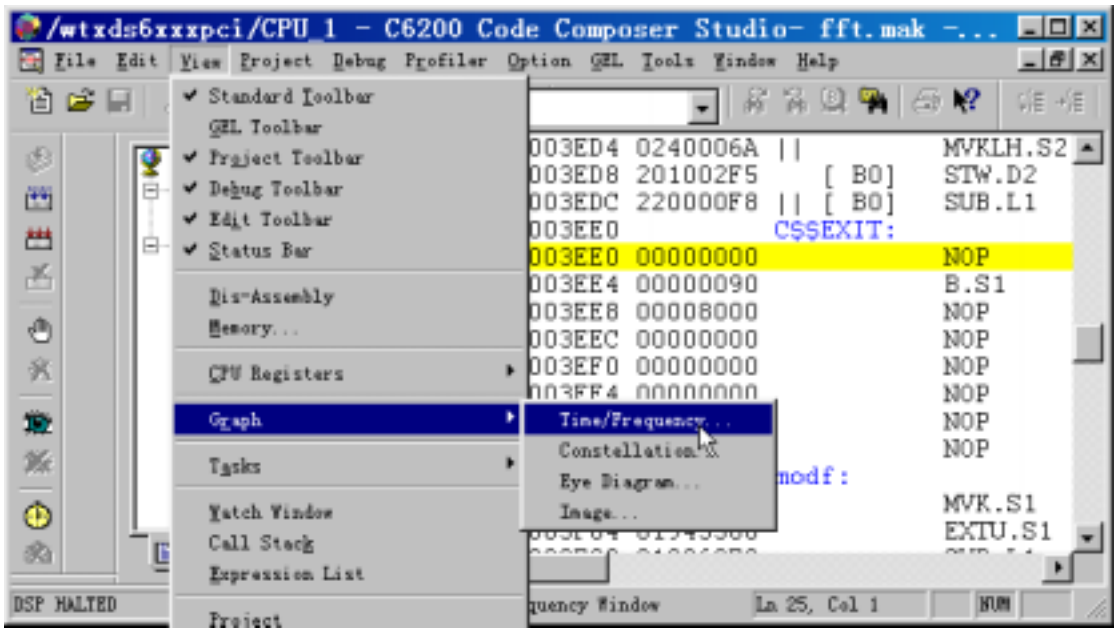
我们的问题解决了，接下来看一看能不能运行，执行对不对。



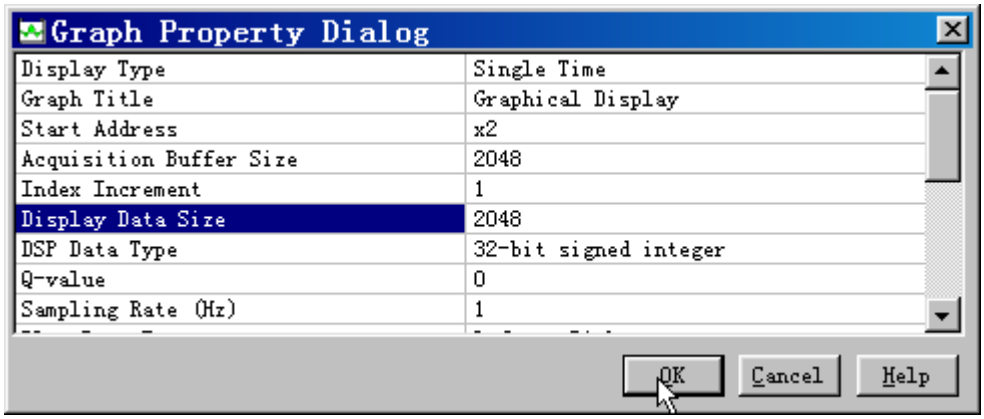
装入“ fft.out ”文件，



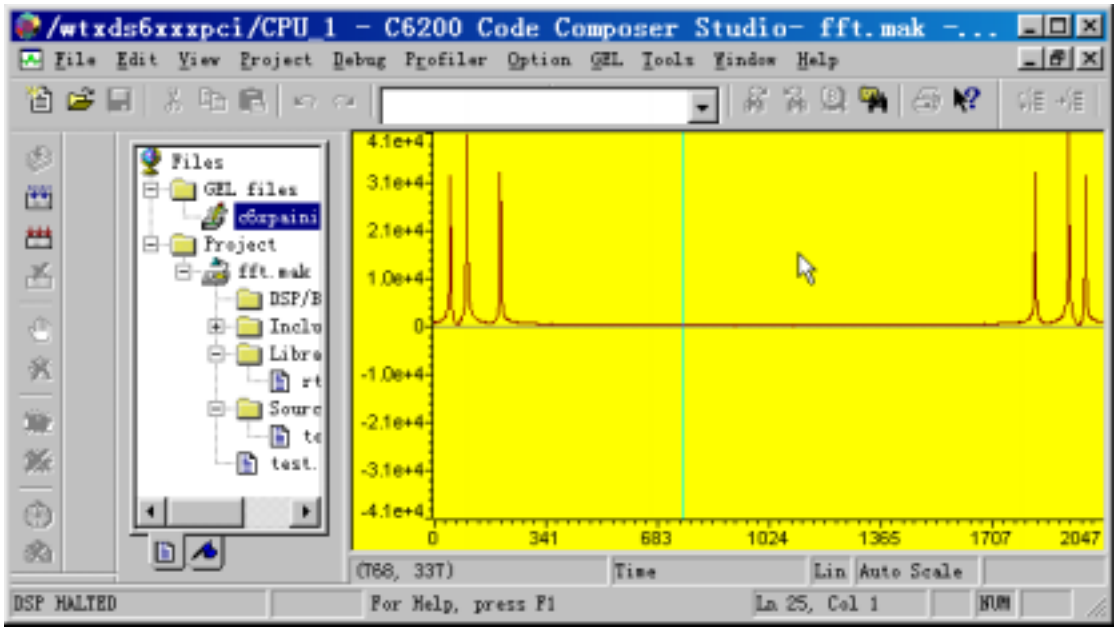
然后运行程序（F5）（热键、工具条、菜单都可以用啦），看一下结果对不对，顺便体验一下 CCS 提供的“ Very Good ”功能：



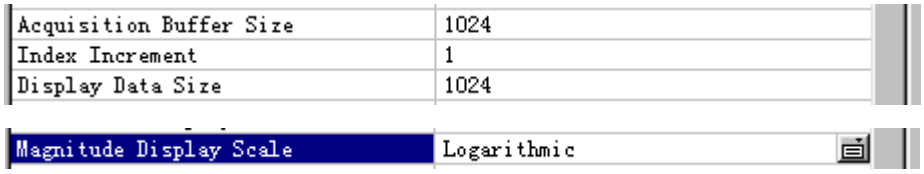
弹出对话框，设置一下吧，



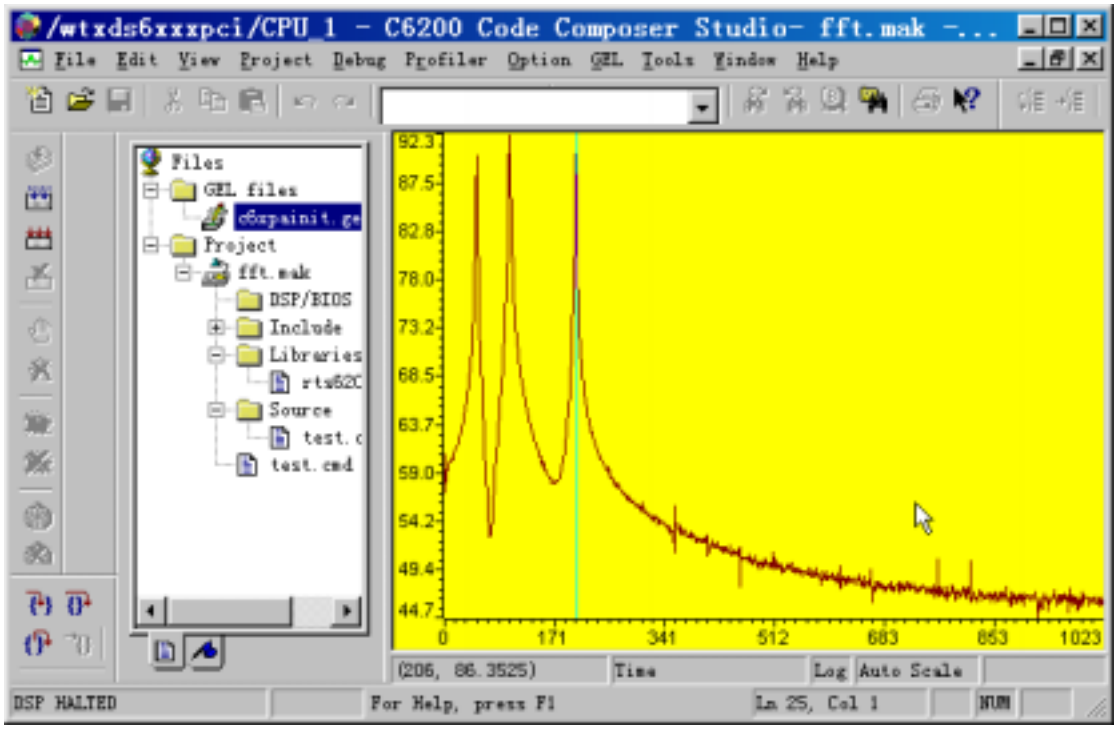
会出现下面的画面，



修改一下属性，



下面好看多了，

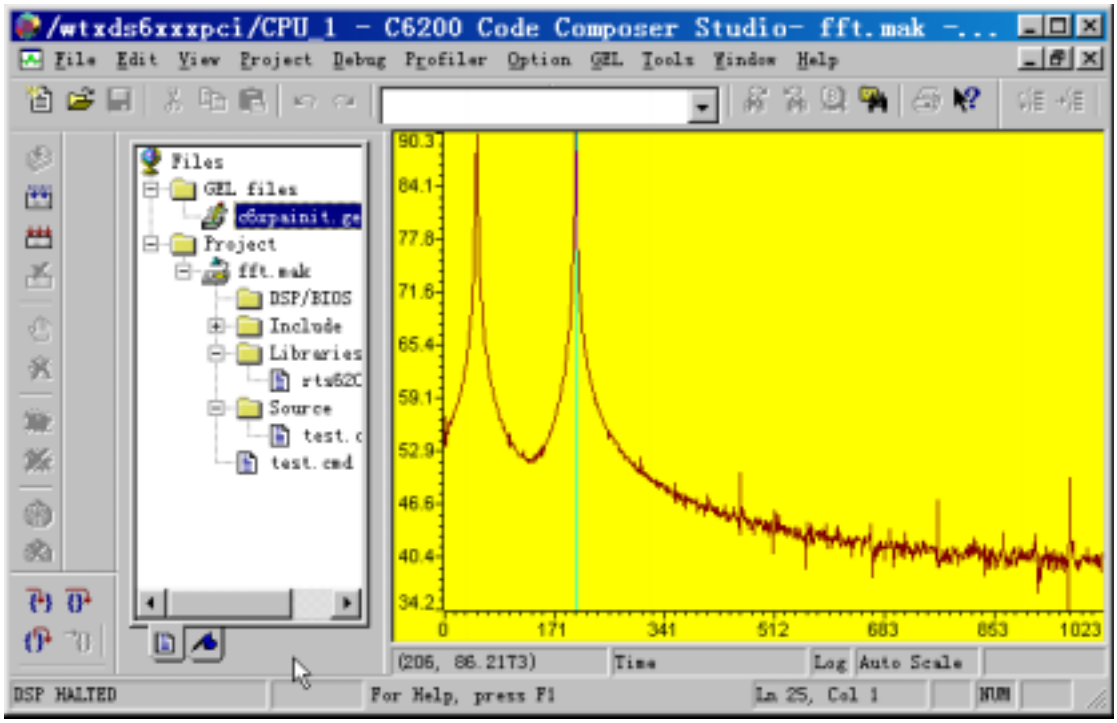


里面的三个谱峰看起来跟信号里造的三个频率的信号好象是对应的吧。在源文件

里进行些修改，例如将信号频率改为两个，

```
/*构造要做FFT的序列，按实部0、虚部0，实部1、虚部1...顺序构造，本例中实部为三
  for (i=0;i<nx1;i++){
                x1[2*i]=(short)((cos(PI*i/20.0)+cos(PI*i/5.0))*0x80);
                x1[2*i+1]=0;
   }
```

重新编译、链接、执行、显示，中间的谱线不见了。



好象我们成功了！(哗哗哗哗哗--------)

总结：在这里演示了使用 CCS 的 C 语言实现 FFT 程序，仅仅是实现而已。一个实际的系统，需要做许多优化处理，对核心算法进行详细的分析、规划，初学时可以先不考虑这些，等熟悉后在进行深入研究。