DSP Platform

for

BF53X

(2011)

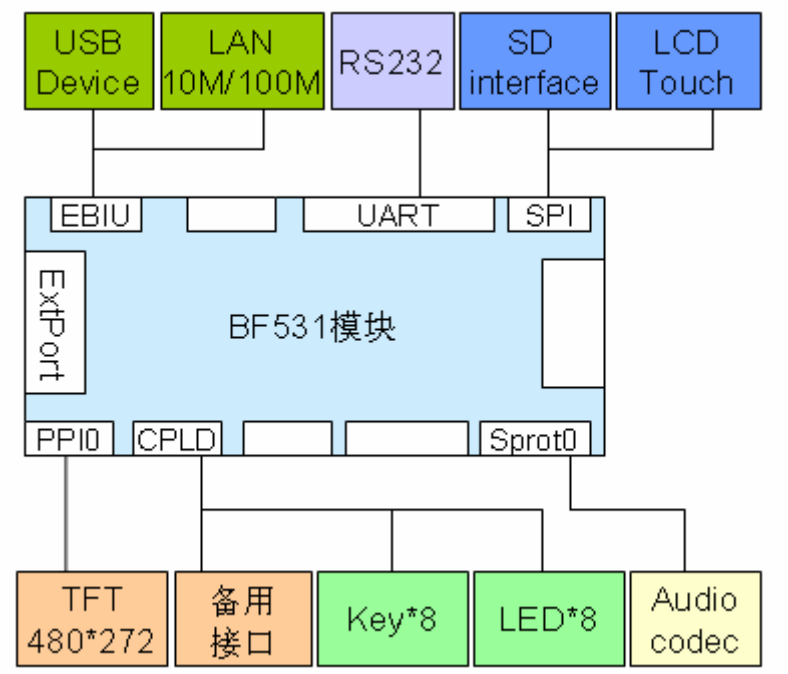
Part Number： ADSP-EDU-BF53X

Update: December, 2010, Beijing, China

ADSP-EDU-BF53X 说明文档

硬件说明

硬件配置



最小系统配置

CPU：ADSP-BF531 处理器；
DRAM：256M bit SDRAM；
FLASH：16M bit NOR FLASH。

外设接口及功能模块

视频输出模块

1 个 4.3 寸 480*272 像素的真彩液晶屏。

实时时钟

支持 RTC 实时时钟。

USB 模块

1 个 MINIUSB　设备接口；
支持 USB 设备功能。

串口模块

1 路 RS232 标准的串口。

触摸屏

1 个 4.3 寸与液晶屏匹配的触摸屏。

轻触键盘

8 个板载按键键盘。

音频模块

1 路 LIN IN 接口；
1 路 HPOUT 接口。

SD/MMC 卡接口

1 个 SPI 模式的 SD/MMC 卡接口。

指示灯

8 个 LED 指示灯。

网络接口

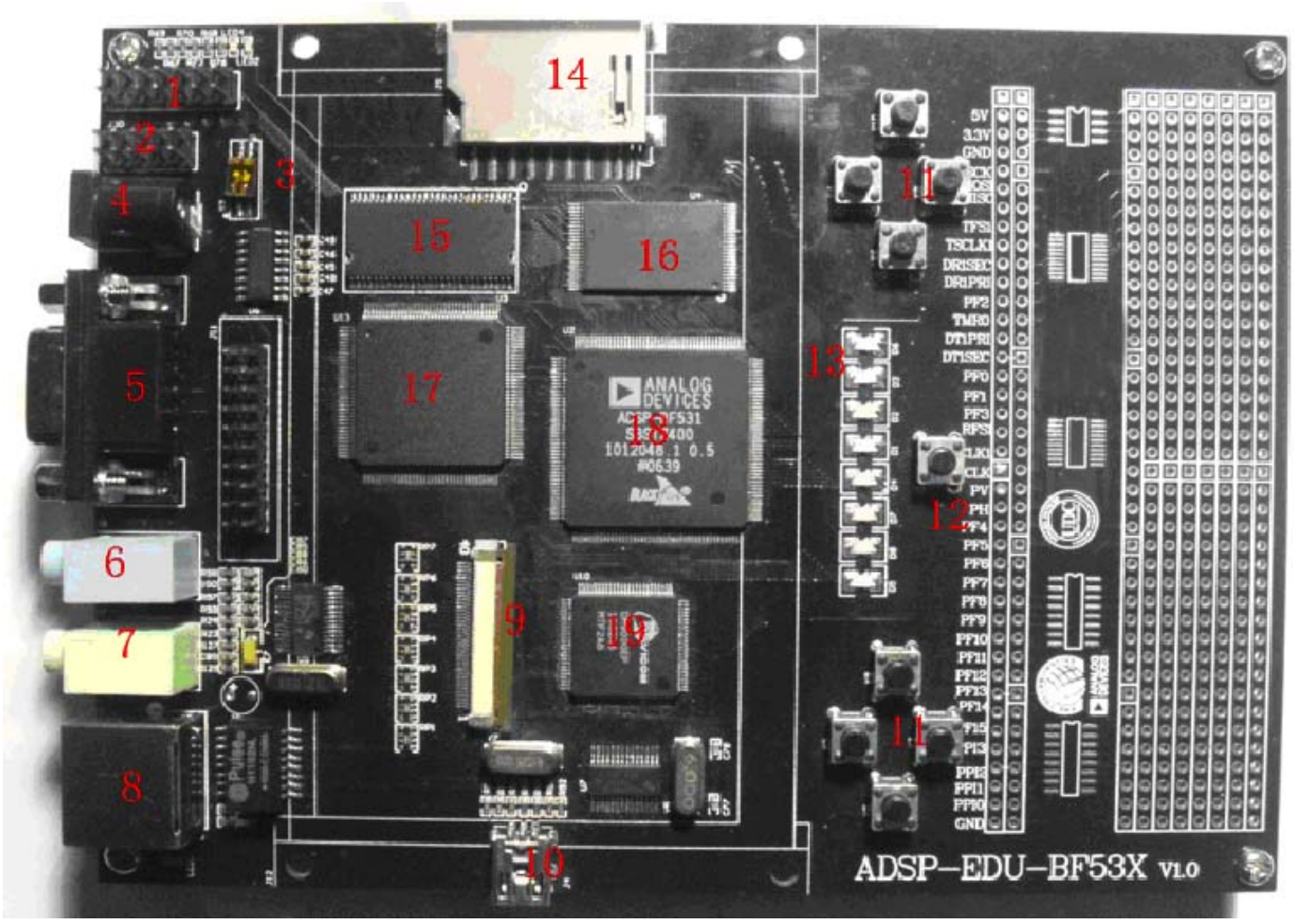
1 个 10M/100M 自适应网络接口。

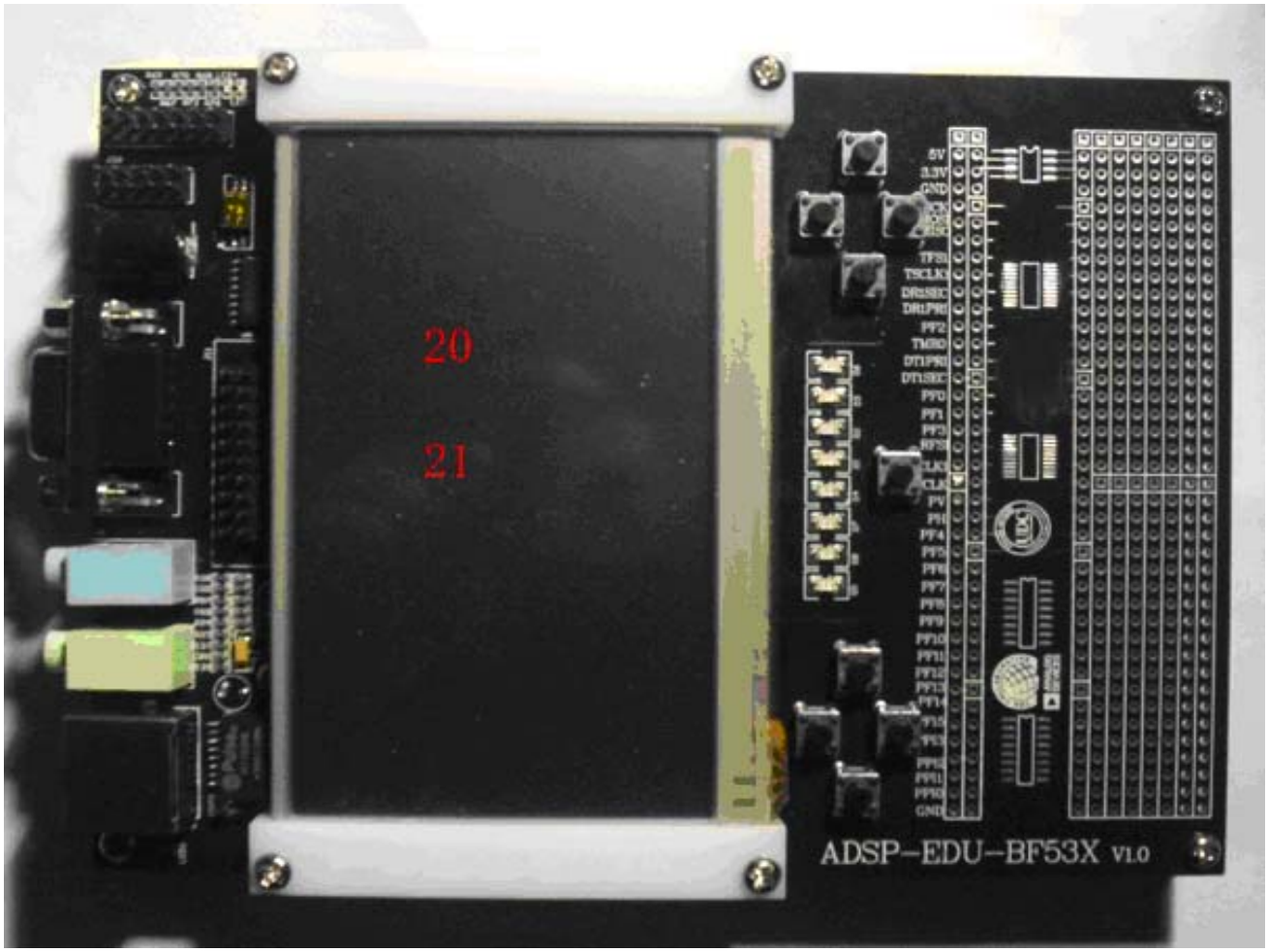
扩展接口

1 组 PPI 扩展接口；
1 组 SPI 扩展接口；
1 组 SPORT 扩展接口；
1 个 CPLD 模块资源扩展；
万用板扩展接口。

硬件接口说明

ADSP-EDU-BF53x 开发板硬件接口如下图：





1. JTAG 仿真接口
2. CPLD 烧写接口
3. BMODE 模式设置开关
4. 7.5V 电源接口
5. RS232 串口
6. 音频 LIN IN 接口
7. 音频 HP OUT 接口
8. 网口
9. TFT 接口
10. USB 设备接口
11. 按键
12. 复位按键
13. LED 灯
14. SD/MMC 接口
15. SDRAM
16. NOR FLASH
17. CPLD

18. ADSP-BF53x
19. 网卡芯片
20. 4.3 寸 480*272 点阵真彩液晶屏
21. 触摸屏

启动模式选择

ADSP-BF53x 处理器有 4 种启动模式，通过芯片上的 BMOOD0 和 BMOOD1 管脚设置，开发板上将这两个管脚引到拨码开关 S2，通过拨码开关来设置启动模式，当拨码开关拨至 ON 时，对应的管脚被拉到 0。
ADSP-BF53x 启动模式：

BMODE1–0	Description
00	Execute from 16-bit external memory (bypass boot ROM)
01	Boot from 8-bit or 16-bit flash
10	Boot from SPI host slave mode
11	Boot from SPI serial EEPROM (8-, 16-, or 24-bit address range)

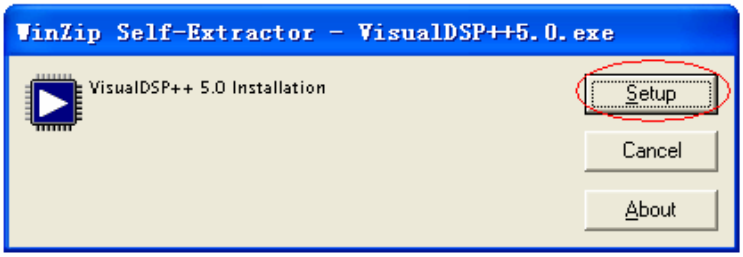
对应拨码开关设置：

BMODE1-0	拨码开关设置
00	S2-1：ON S2-2：ON
01	S2-1：ON S2-2：OFF
10	S2-1：OFF S2-2：ON
11	S2-1：OFF S2-2：OFF

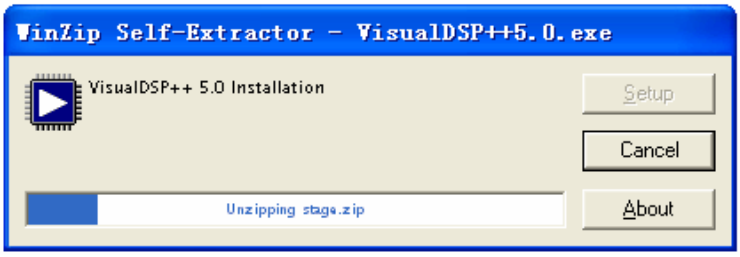
使用 ADSP-EDU-BF53X

Visual DSP++ 5.0 软件安装

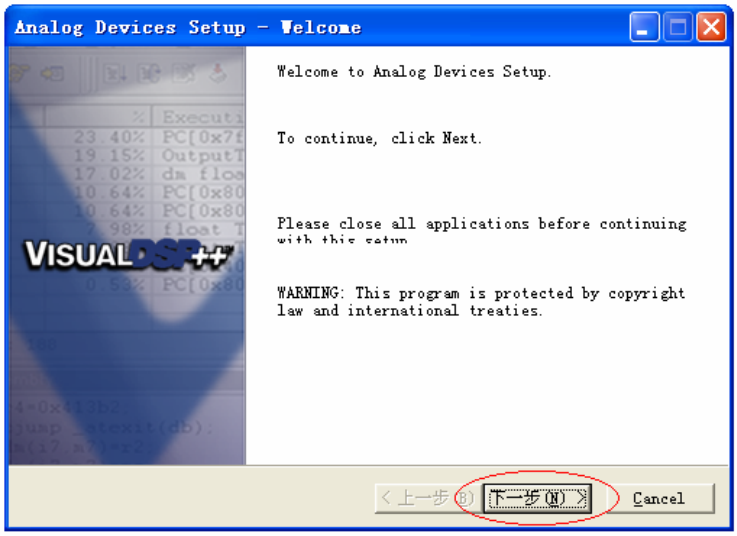
运行安装文件，弹出如下会话框，点“Setup”



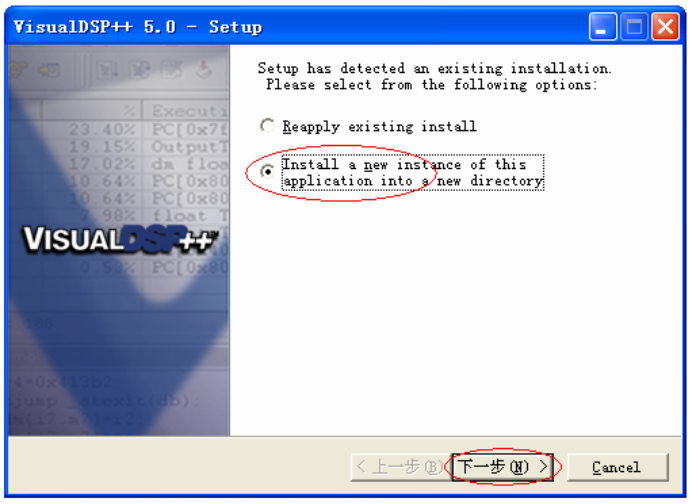
等待进度条完成



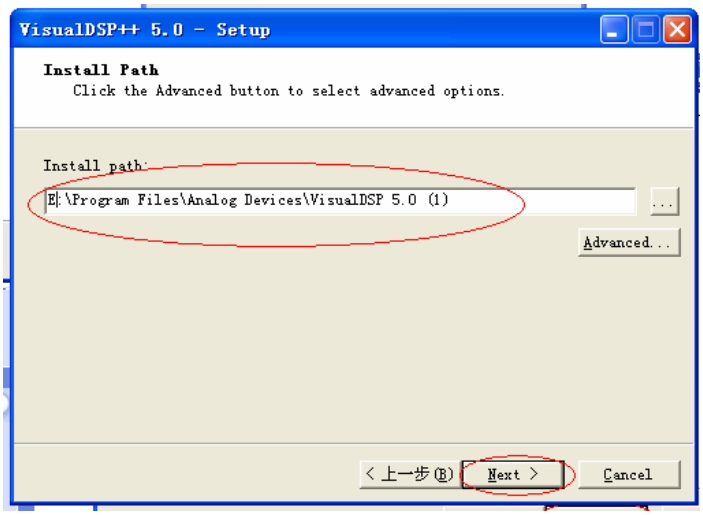
进度条完成后，弹出如下图会话框，点“下一步”



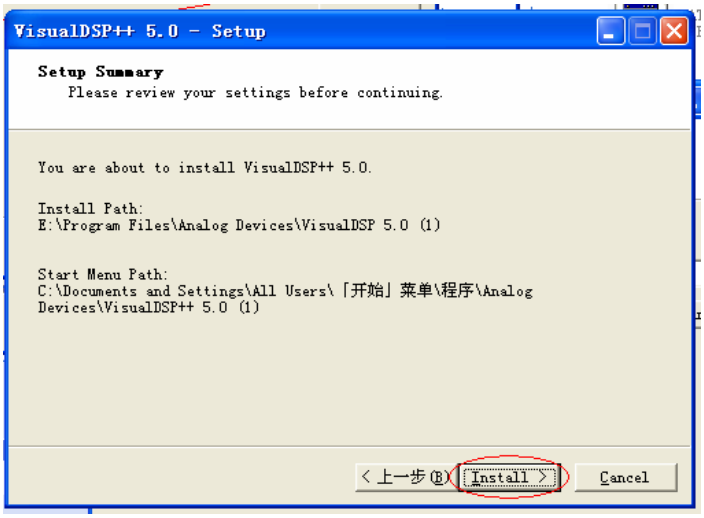
首次安装不会出现下面的选项，非首次安装，出现如下选项，选择红框圈的选项



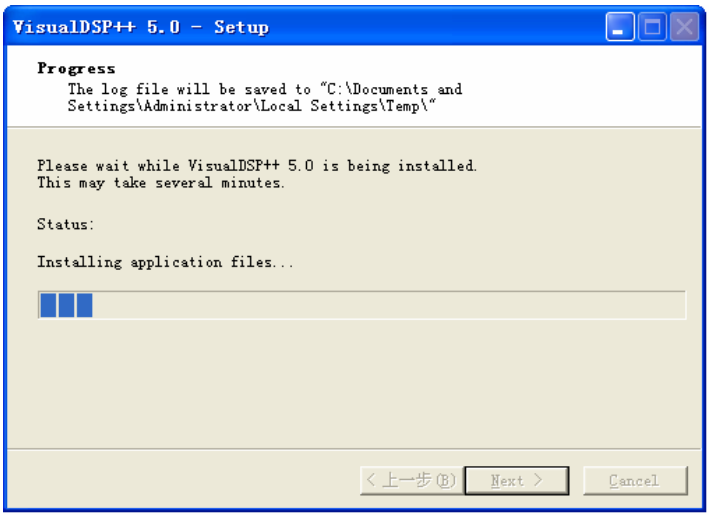
选择软件的安装路径，可任意选一个磁盘路径



点“Install”

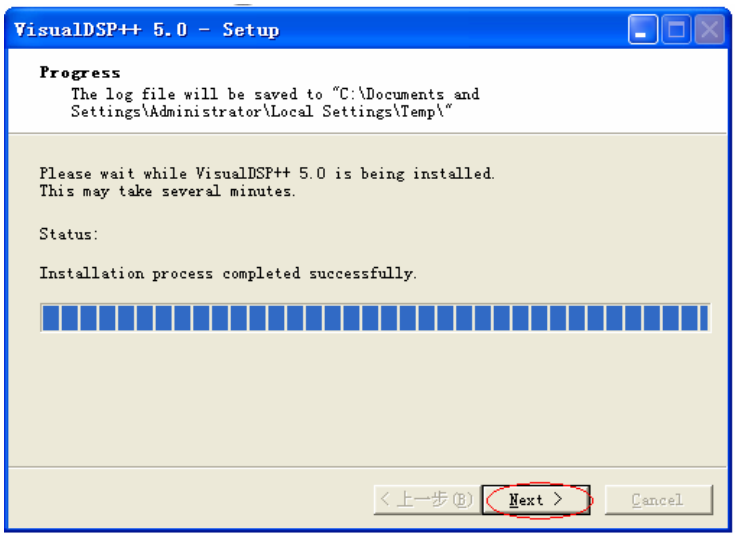


等待安装完成

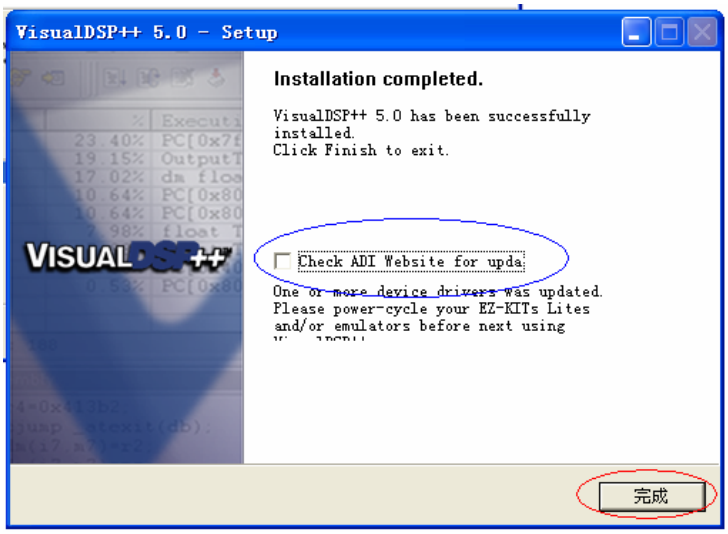


安装完成后，会弹出会话框提示，点“确定”

在进图条页面点击“Next>”

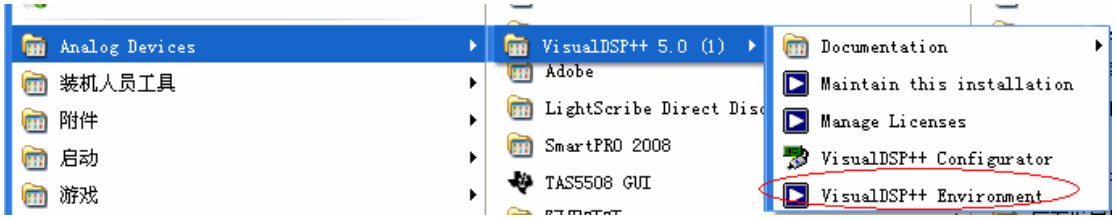


将更新提示选项上的勾去掉，点“完成”

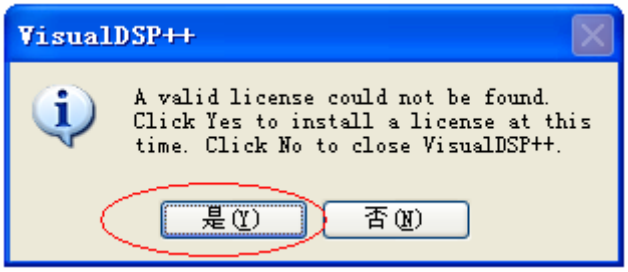


注意：Visual DSP++ 5.0 的序列号在不注册时只能使用 30 天，30 天后软件不能使用，为了延长使用的时间，可以采用技巧，修改系统时间来延长使用时间。

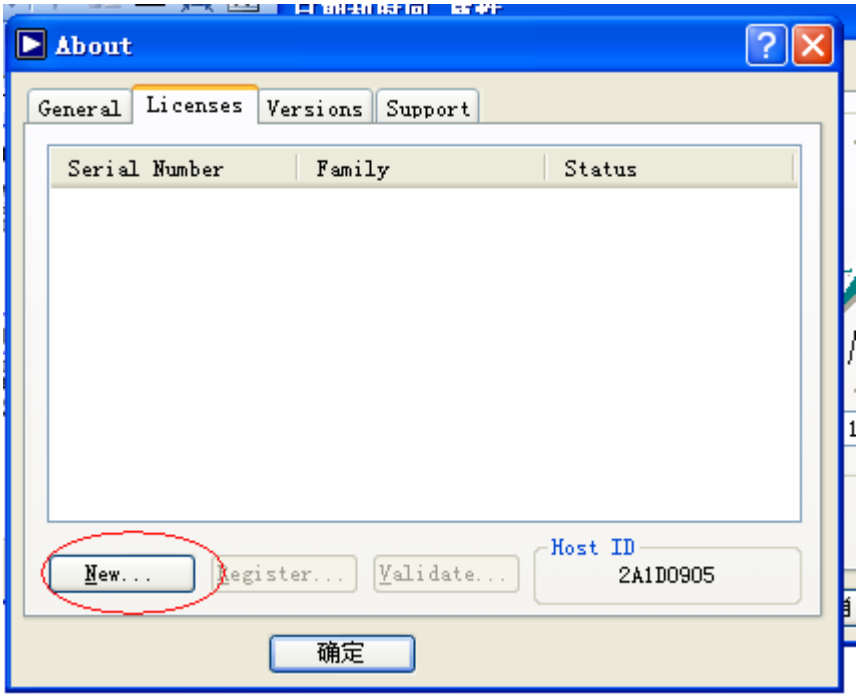
在开始菜单 -->程序中，找到”Analog Devices” 的软件，运行红框框的图标选项



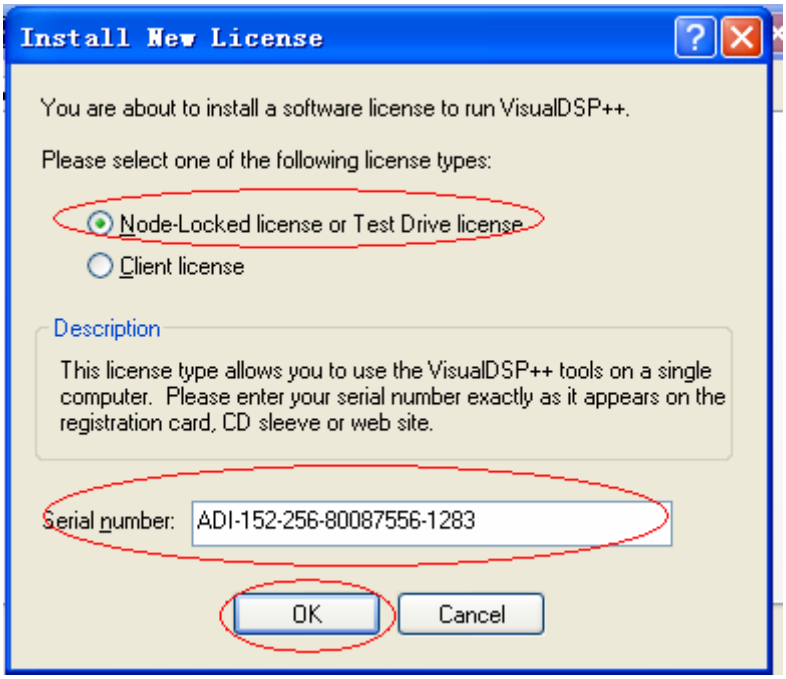
弹出输入序列号的会话框，点“是”



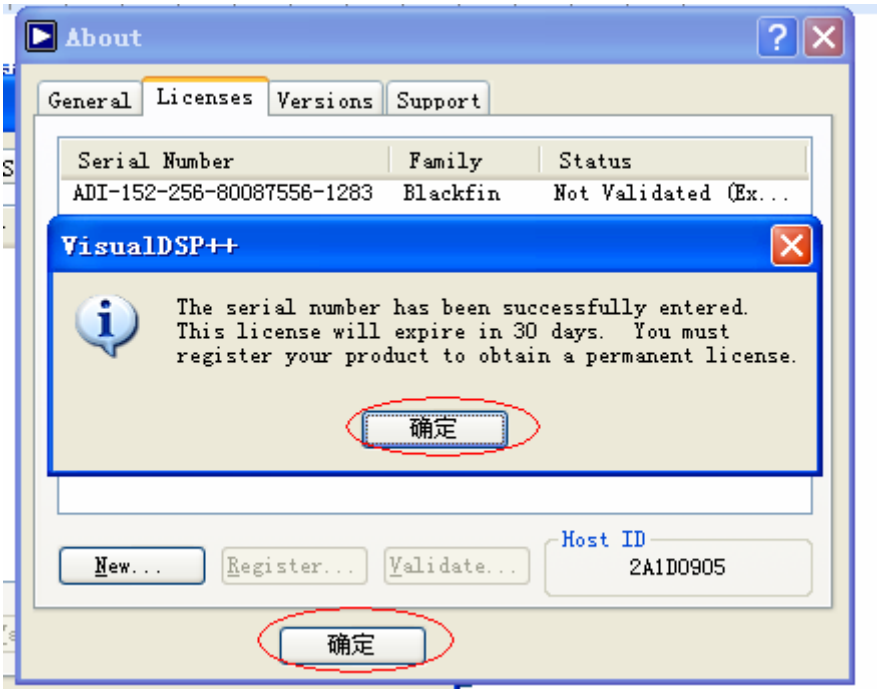
选择“New…”



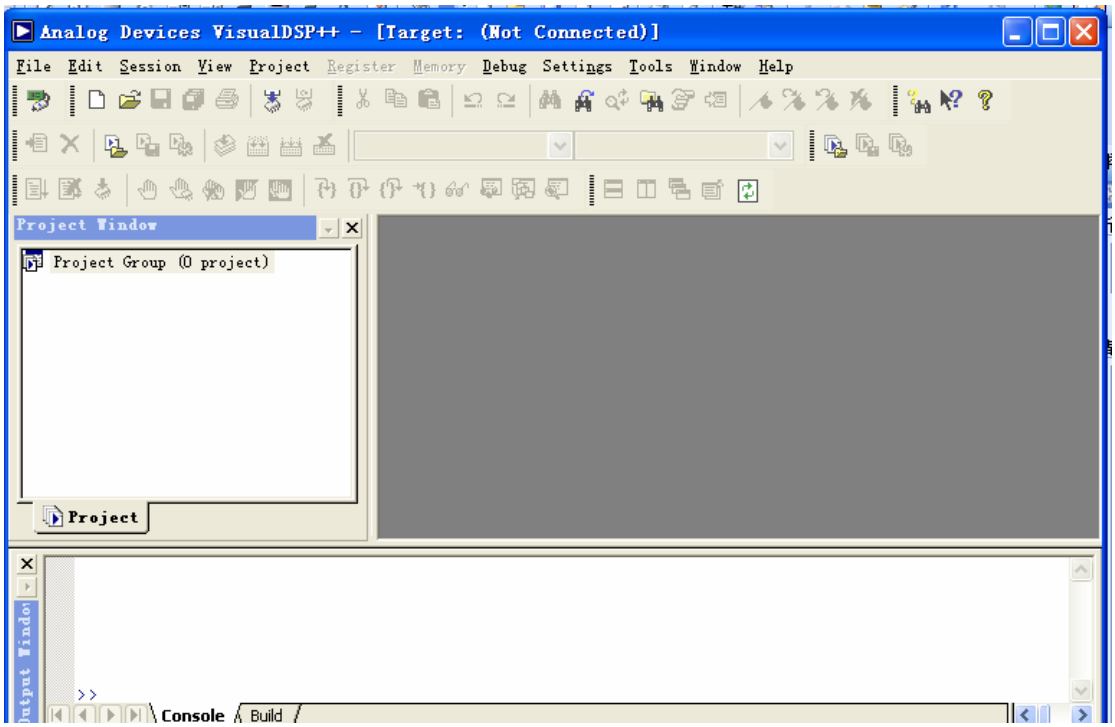
输入序列号“ADI-152-256-80087556-1283”,点“OK”



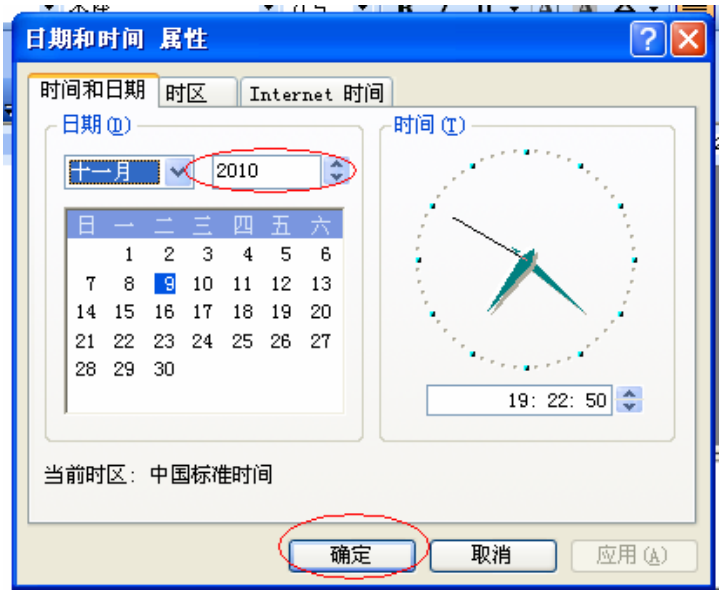
弹出会话框提示可以使用 30 天，点“确定”



即可进入软件界面

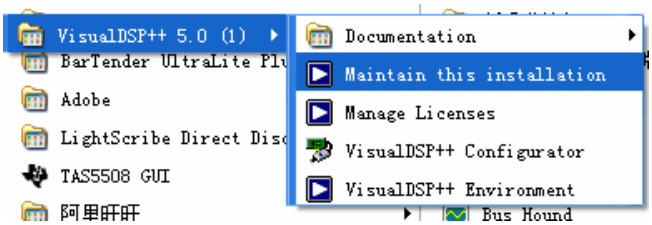


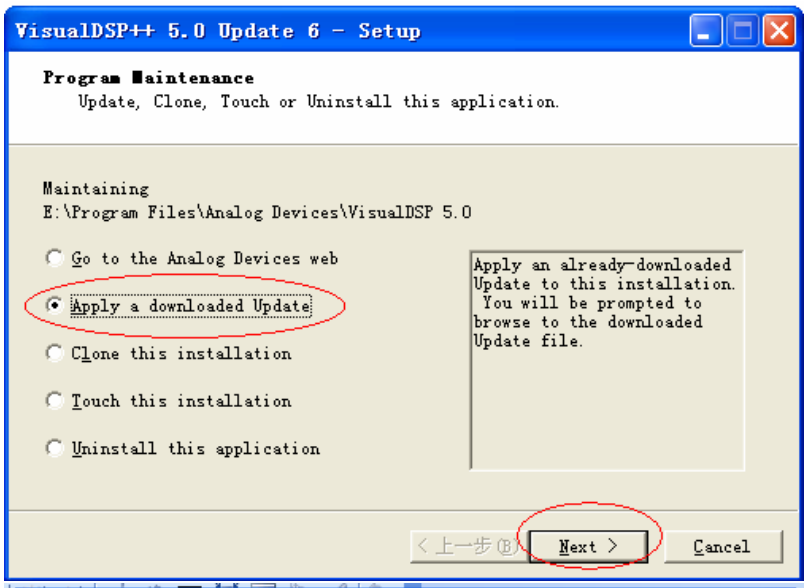
将计算机的时间改为当前时间

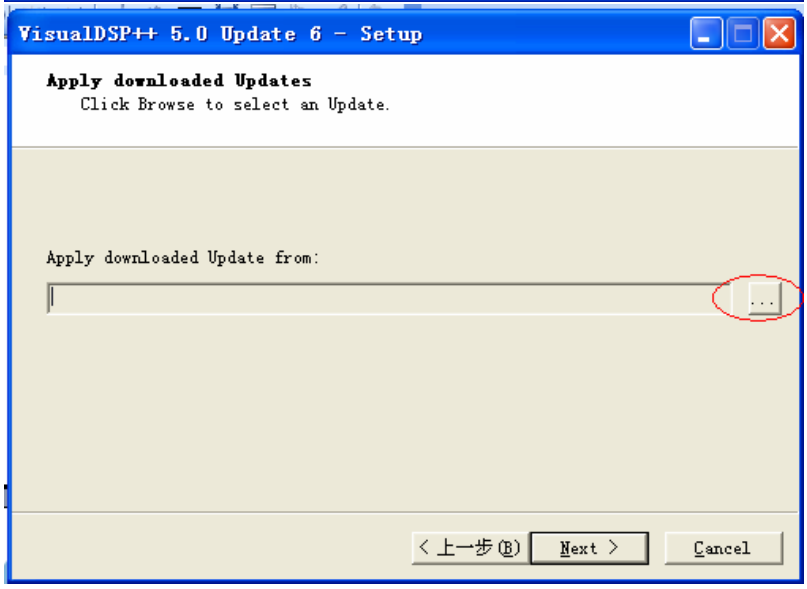


Visual DSP++ 5.0 升级包（Update）的安装

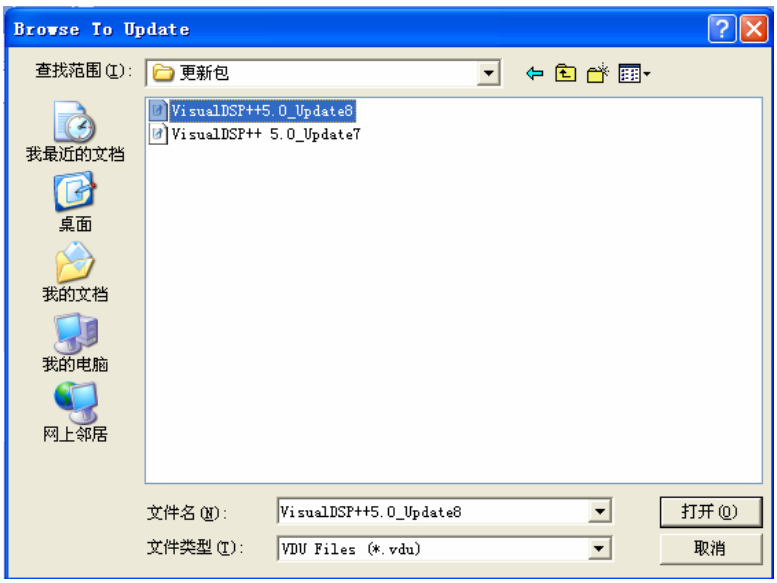
关闭 Visual DSP++ 5.0 软件，找到开始菜单中如图图标，运行。

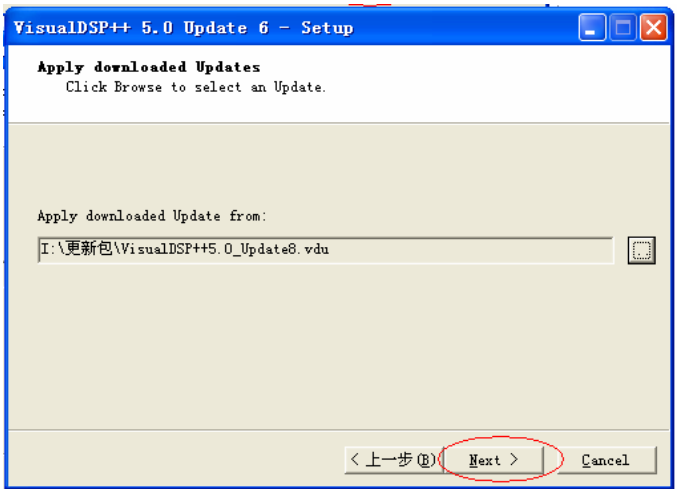

VisualDSP++ 5.0 (1)
Documentation
Maintain this installation
Manage Licenses
VisualDSP++ Configurator
VisualDSP++ Environment
Adobe
LightScribe Direct Disc
TAS5508 GUI
Bus Hound

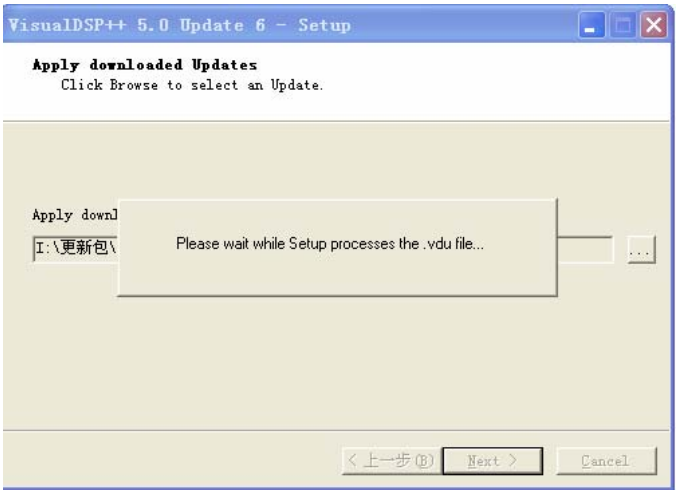

VisualDSP++ 5.0 Update 6 - Setup
Program Maintenance
Update, Clone, Touch or Uninstall this application.
Maintaining
E:\Program Files\Analog Devices\VisualDSP 5.0
Go to the Analog Devices web
Apply a downloaded Update
Clone this installation
Touch this installation
Uninstall this application
Apply an already-downloaded Update to this installation. You will be prompted to browse to the downloaded Update file.
< 上一步(B)
Next >
Cancel


VisualDSP++ 5.0 Update 6 - Setup
Apply downloaded Updates
Click Browse to select an Update.
Apply downloaded Update from:
...
< 上一步(B)
Next >
Cancel

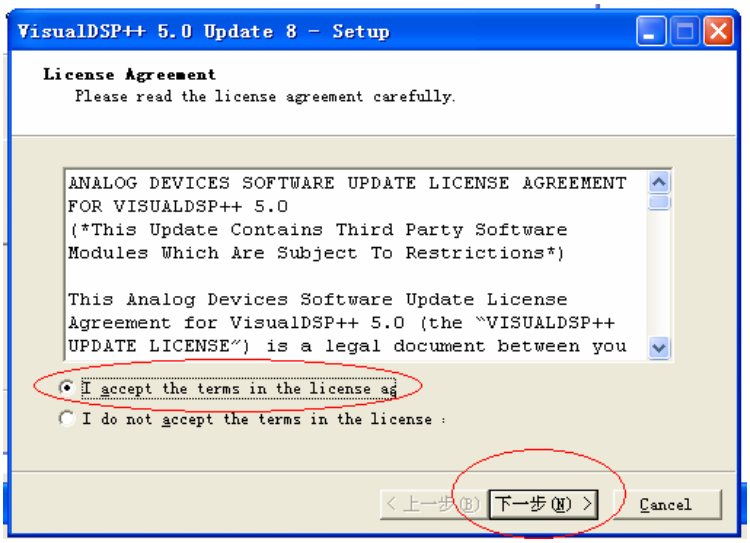
找到更新包

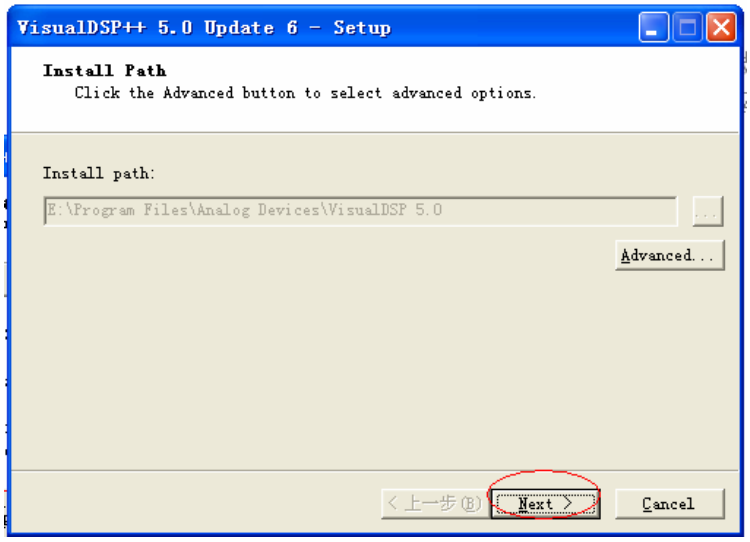

Browse To Update
查找范围(I):
更新包
VisualDSP++5.0_Update8
VisualDSP++ 5.0_Update7
我最近的文档
桌面
我的文档
我的电脑
网上邻居
文件名(N):
VisualDSP++5.0_Update8
打开(O)
文件类型(T):
VDU Files (*.vdu)
取消

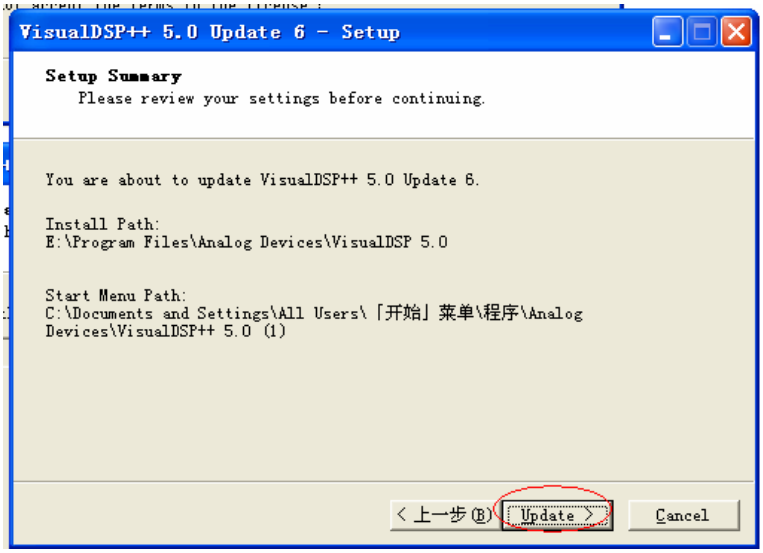

VisualDSP++ 5.0 Update 6 - Setup
Apply downloaded Updates
Click Browse to select an Update.
Apply downloaded Update from:
I:\更新包\VisualDSP++5.0_Update8.vdu
< 上一步(B)
Next >
Cancel

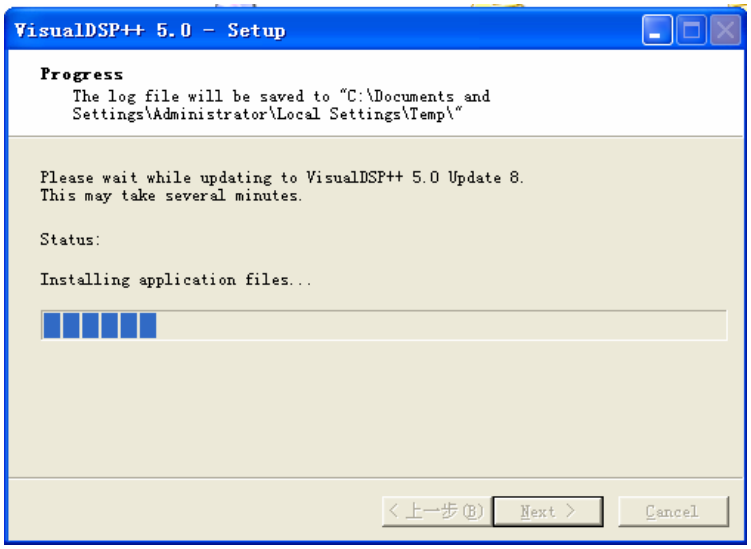

VisualDSP++ 5.0 Update 6 - Setup
Apply downloaded Updates
Click Browse to select an Update.
Apply downl
I:\更新包\
Please wait while Setup processes the .vdu file...
< 上一步(B)
Next >
Cancel

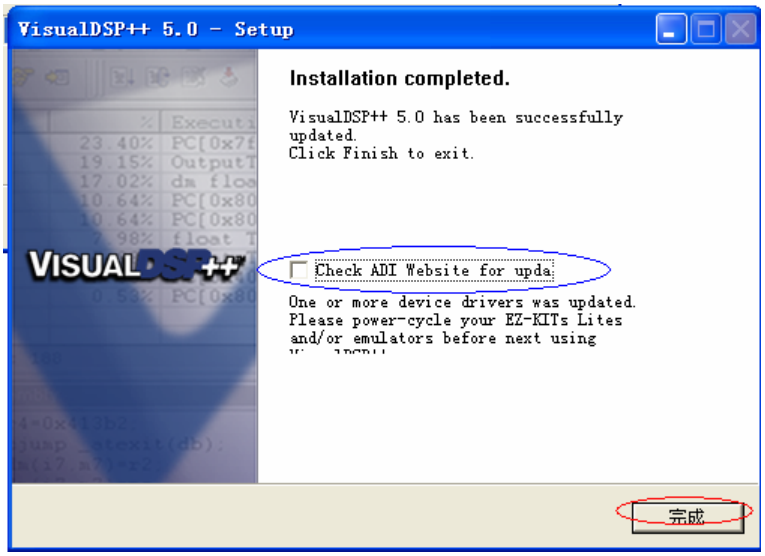

Setup
An updated installer was found. The installer will now restart using this updated version.
Press "OK" to continue.
确定


VisualDSP++ 5.0 Update 8 - Setup
License Agreement
Please read the license agreement carefully.
ANALOG DEVICES SOFTWARE UPDATE LICENSE AGREEMENT
FOR VISUALDSP++ 5.0
(*This Update Contains Third Party Software
Modules Which Are Subject To Restrictions*)
This Analog Devices Software Update License
Agreement for VisualDSP++ 5.0 (the "VISUALDSP++
UPDATE LICENSE") is a legal document between you
I accept the terms in the license ag
I do not accept the terms in the license
< 上一步(B)
下一步(N) >
Cancel


VisualDSP++ 5.0 Update 6 - Setup
Install Path
Click the Advanced button to select advanced options.
Install path:
E:\Program Files\Analog Devices\VisualDSP 5.0
Advanced...
< 上一步(B)
Next >
Cancel


VisualDSP++ 5.0 Update 6 - Setup
Setup Summary
Please review your settings before continuing.
You are about to update VisualDSP++ 5.0 Update 6.
Install Path:
E:\Program Files\Analog Devices\VisualDSP 5.0
Start Menu Path:
C:\Documents and Settings\All Users\「开始」菜单\程序\Analog Devices\VisualDSP++ 5.0 (1)
< 上一步(B)
Update >
Cancel


VisualDSP++ 5.0 - Setup
Progress
The log file will be saved to "C:\Documents and Settings\Administrator\Local Settings\Temp\"
Please wait while updating to VisualDSP++ 5.0 Update 8.
This may take several minutes.
Status:
Installing application files...
< 上一步(B)
Next >
Cancel


VisualDSP++ 5.0 - Setup
Installation completed.
VisualDSP++ 5.0 has been successfully updated.
Click Finish to exit.
Check ADI Website for upda
One or more device drivers was updated.
Please power-cycle your EZ-KITs Lites and/or emulators before next using
完成

仿真器与板卡的连接

ADI DSP 的仿真器有很多种，根据仿真器不同，在连接时选择的选项也不同。文档以 AD-HP510ICE-FULL 仿真器和 AD-HP560ICE-FULL 仿真器为例，来连接 ADSP-EDU-BF53x 开发板。

AD-HP510ICE-FULL

采用 AD-HP510ICE-FULL 仿真器连接 ADSP-EDU-BF53x 开发板，步骤如下：

上电顺序

1. 确保目标板和仿真器都未上电
2. 将仿真接口连接 ADSP-EDU-BF53x 开发板，板卡上的仿真接口为 14Pin 双排插针接口，其中 Pin3 脚被剪去，其目的是防反插。AD-HP510ICE-FULL 仿真器的仿真接口中 Pin3 孔被堵上，反向不能插入板卡。
3. 仿真器上电
4. 目标板上电

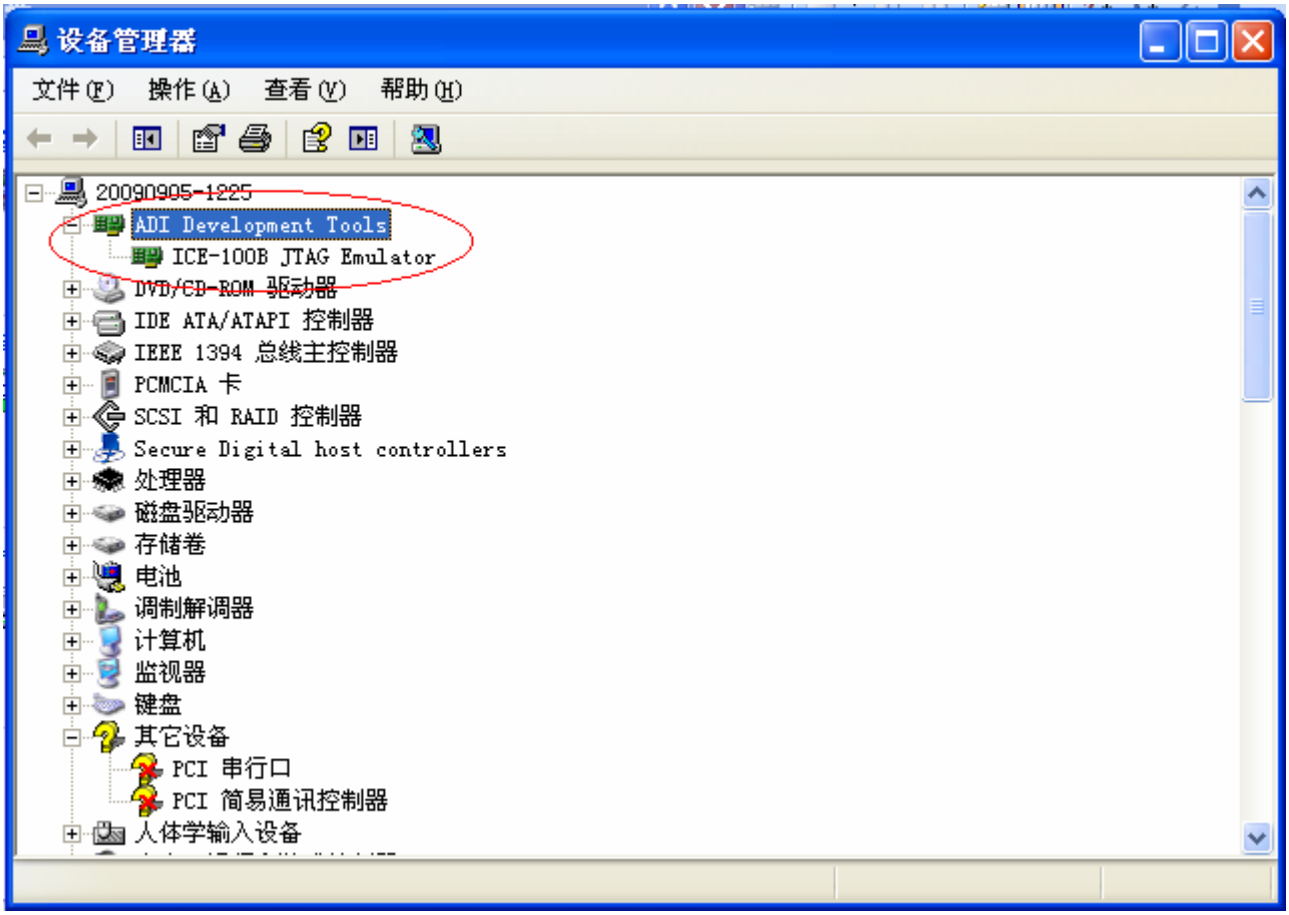
仿真器上电，板卡上电，仿真器 USB 线连接计算机，计算机端弹出驱动安装页面，选择“自动安装软件”

找到驱动程序

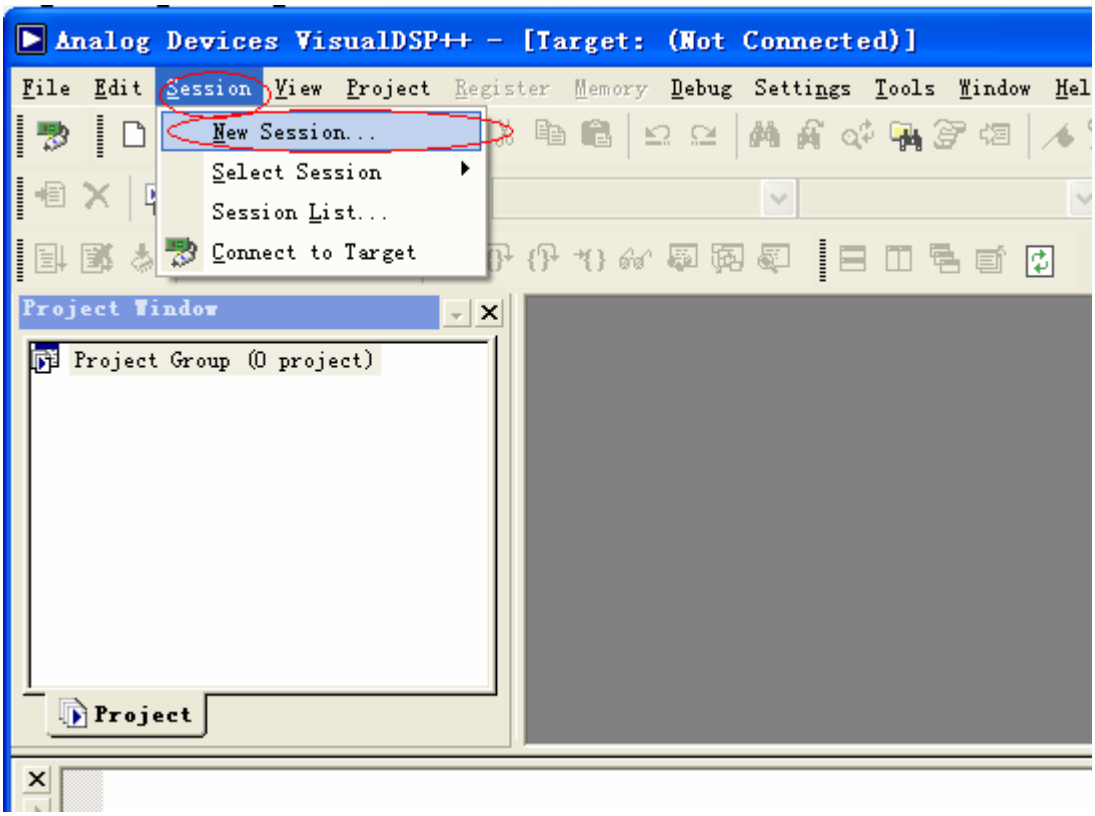
AD-HP510ICE-FULL 仿真器与 ADI 公司原产的 ICE-100B 仿真器完全兼容，所以安装后驱动会显示为 ICE-100B JTAG Emulator 的设备接入。

完成驱动安装

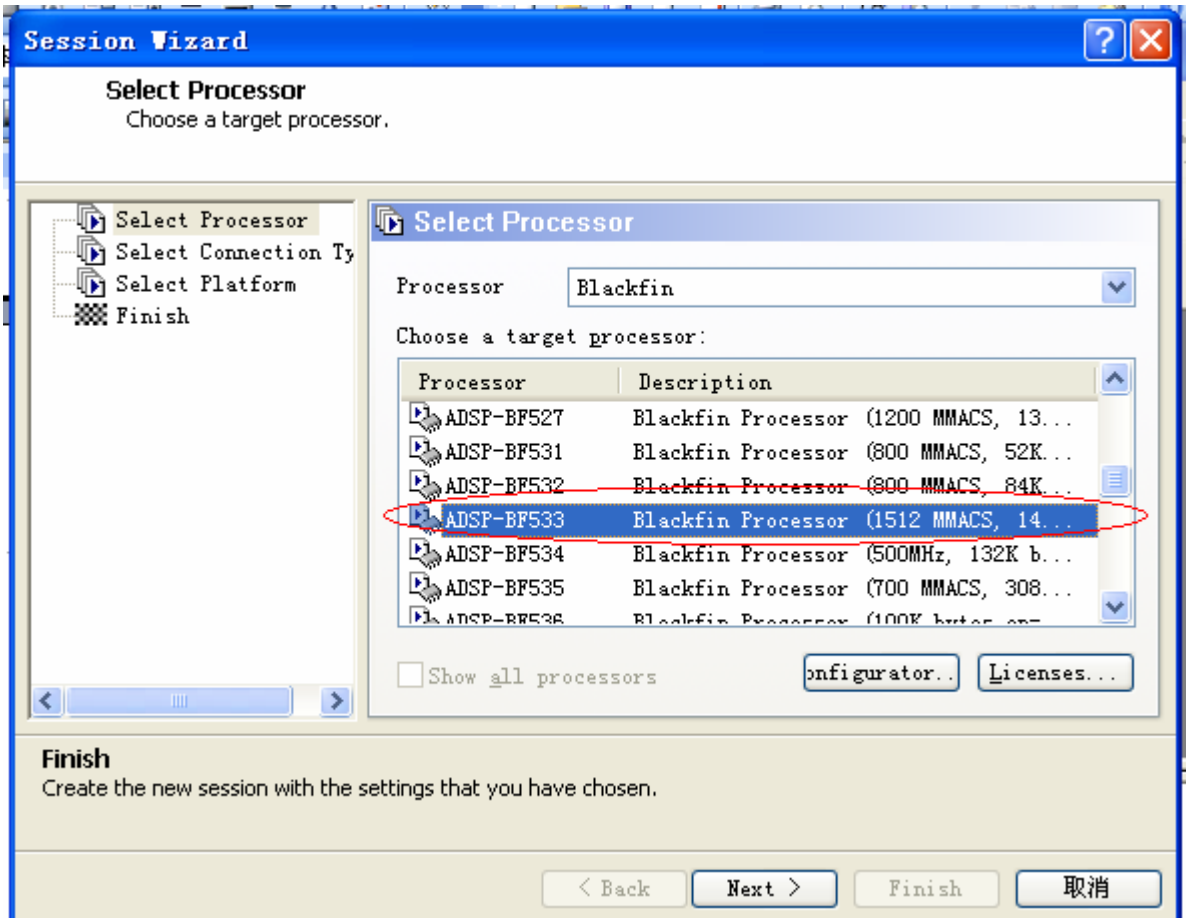
设备管理器中显示驱动安装状态



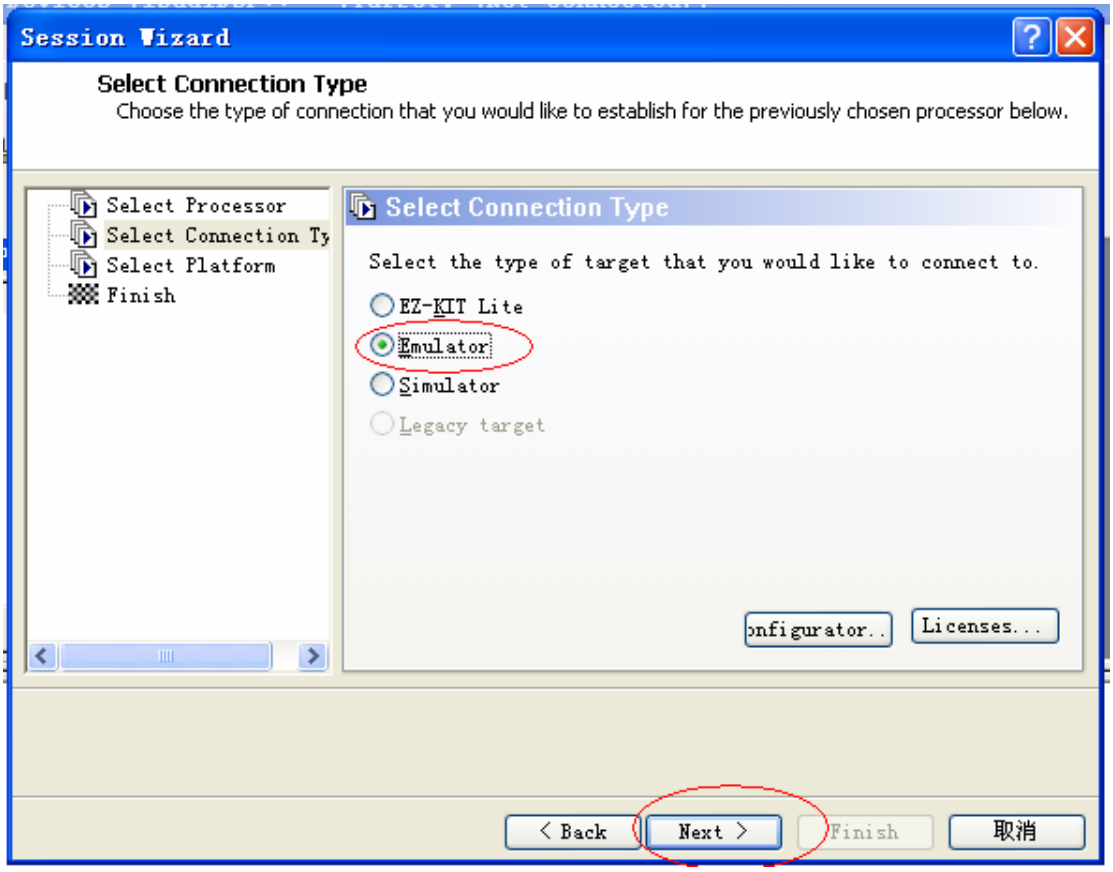
运行 VisualDSP++5.0 软件界面上，选择“New Session..”


Analog Devices VisualDSP++ - [Target: (Not Connected)]
File Edit Session View Project Register Memory Debug Settings Tools Window Hel
New Session...
Select Session
Session List...
Connect to Target
Project Window
Project Group (0 project)
Project

选择红框圈的选项

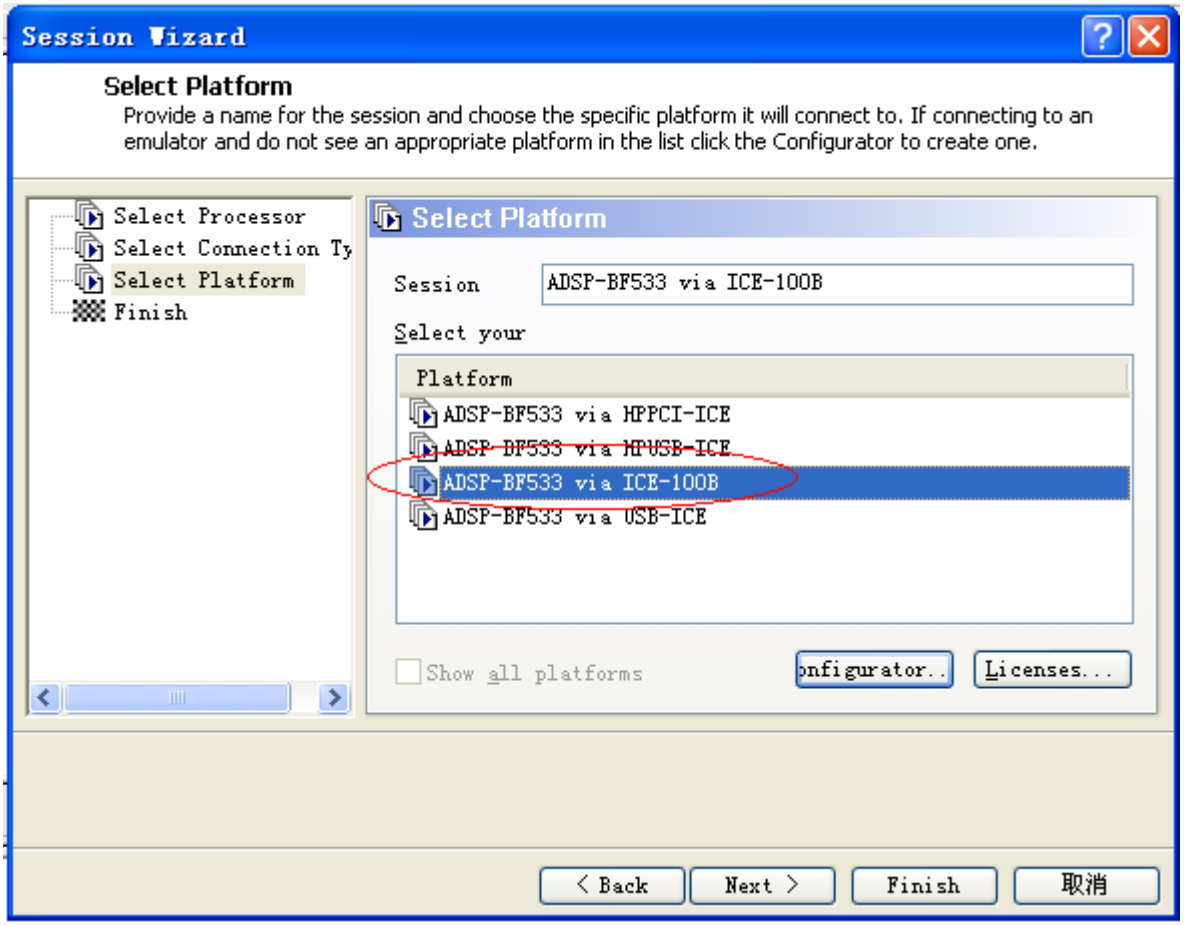


选择“Emulator”后，点“Next>”

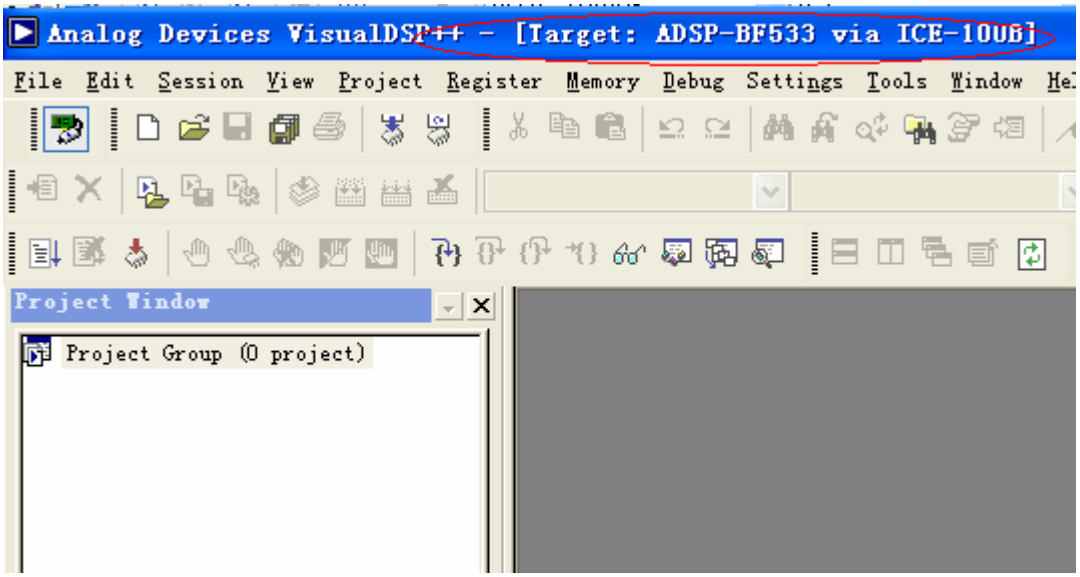


选择“ADSP-BF533 via ICE-100B”后，点“Finish”

ADSP-EDU-BF53x 开发板以后会根据客户需求，处理器会使用 BF531,BF532,BF533，但我们的所有工程代码都是以 ADSP-BF533 Session 下开发，BF531,BF532,BF533 有着相同的内核和外设，管脚完全兼容，所以不管开发板上焊接的芯片是哪一款，都应以 ADSP-BF533 的 Session 开发，把板卡上的处理器都作为 BF533　400MHz 的主频来使用。



连接成功后在 VisualDSP++5.0 软件界面上可以看到连接状态，该状态代表目前仿真器在连接着板卡



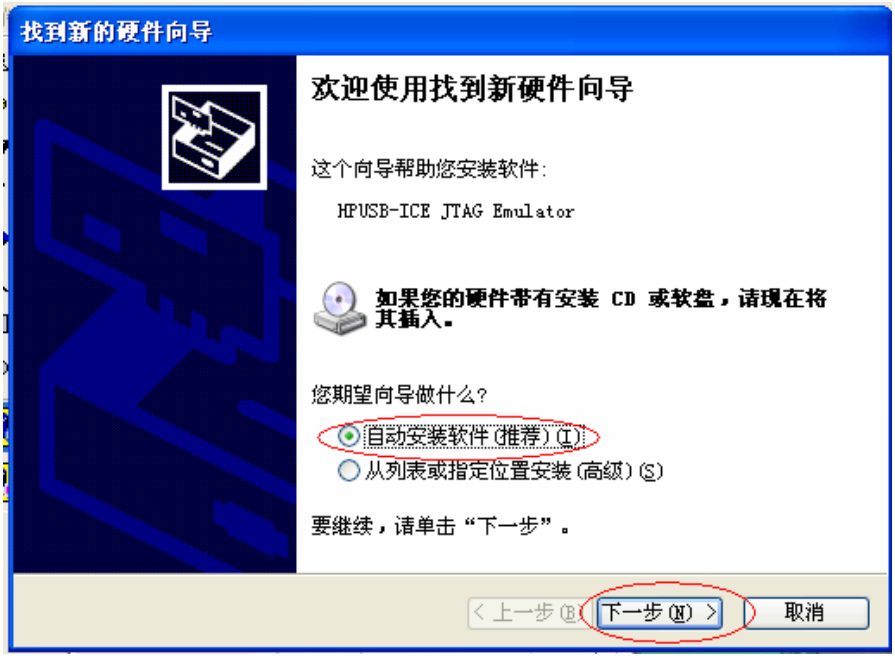
AD-HP560ICE-FULL

采用 AD-HP560ICE-FULL 仿真器连接 ADSP-EDU-BF53x 开发板，步骤如下：

上电顺序：

1. 确保目标板和仿真器都未上电；
2. 将仿真接口连接 ADSP-EDU-BF53x 开发板，板卡上的仿真接口为 14Pin 双排插针接口，其中 Pin3 脚被剪去，其目的是防反插。AD-HP560ICE-FULL 仿真器的仿真接口中 Pin3 孔被堵上，反向不能插入板卡；
3. 仿真器上电；
4. 目标板上电；

仿真器上电，板卡上电，仿真器 USB 线连接计算机，计算机端弹出驱动安装页面，选择“自动安装软件”

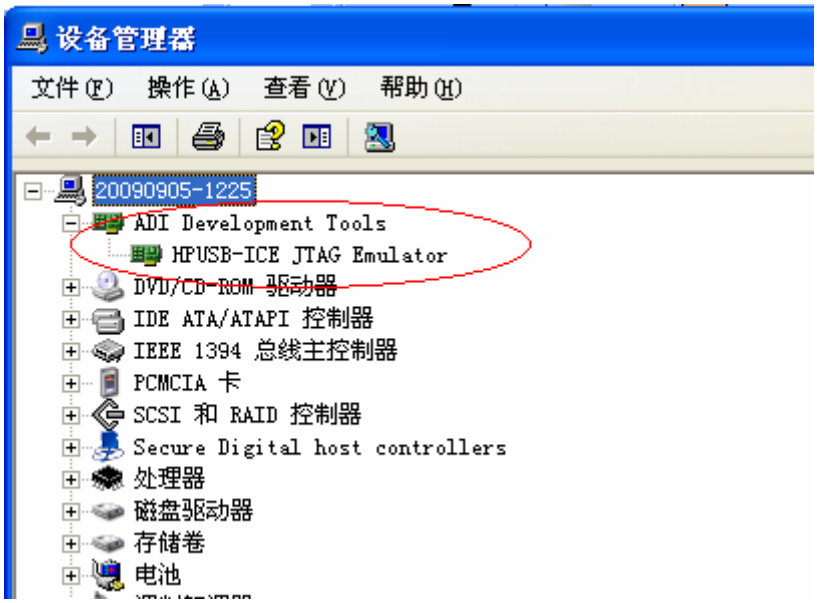


找到驱动程序

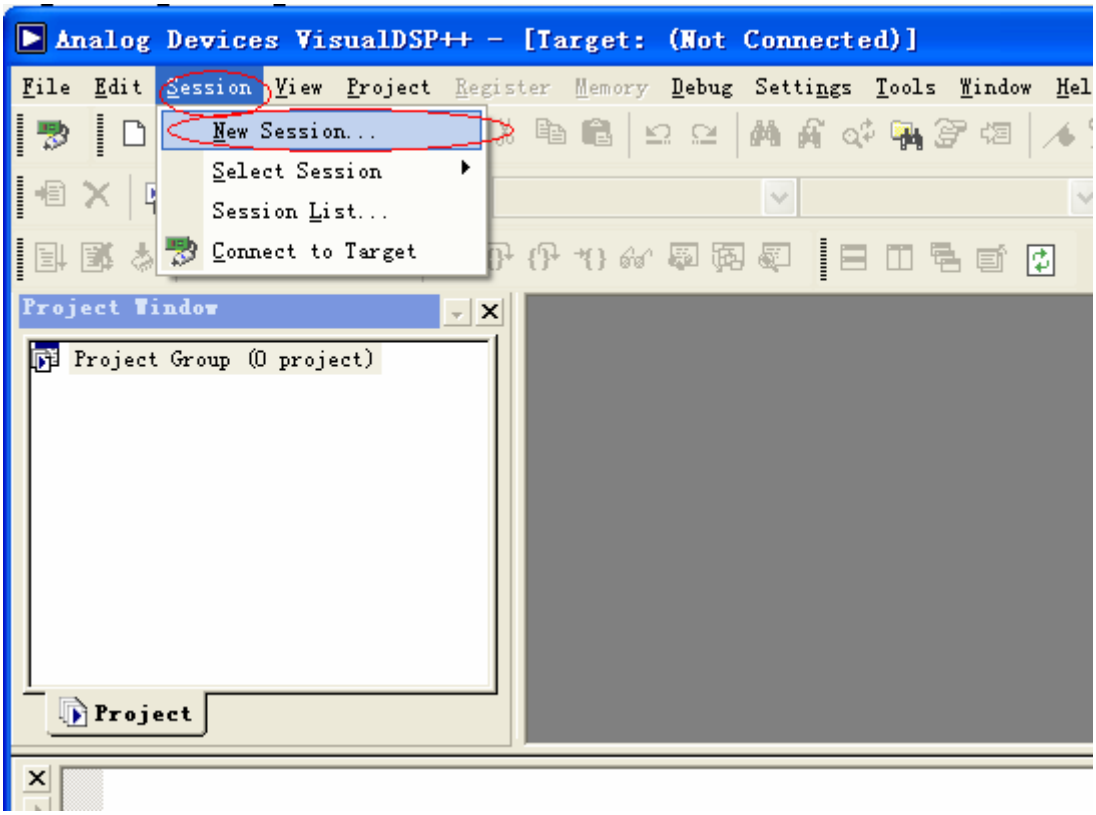
AD-HP560ICE-FULL 仿真器与 ADI 公司原产的 ADSP-HPUSB-ICE 仿真器完全兼容，所以安装后驱动会显示为 HPUSB-ICE 的设备接入。

完成驱动安装

设备管理器中显示驱动安装状态

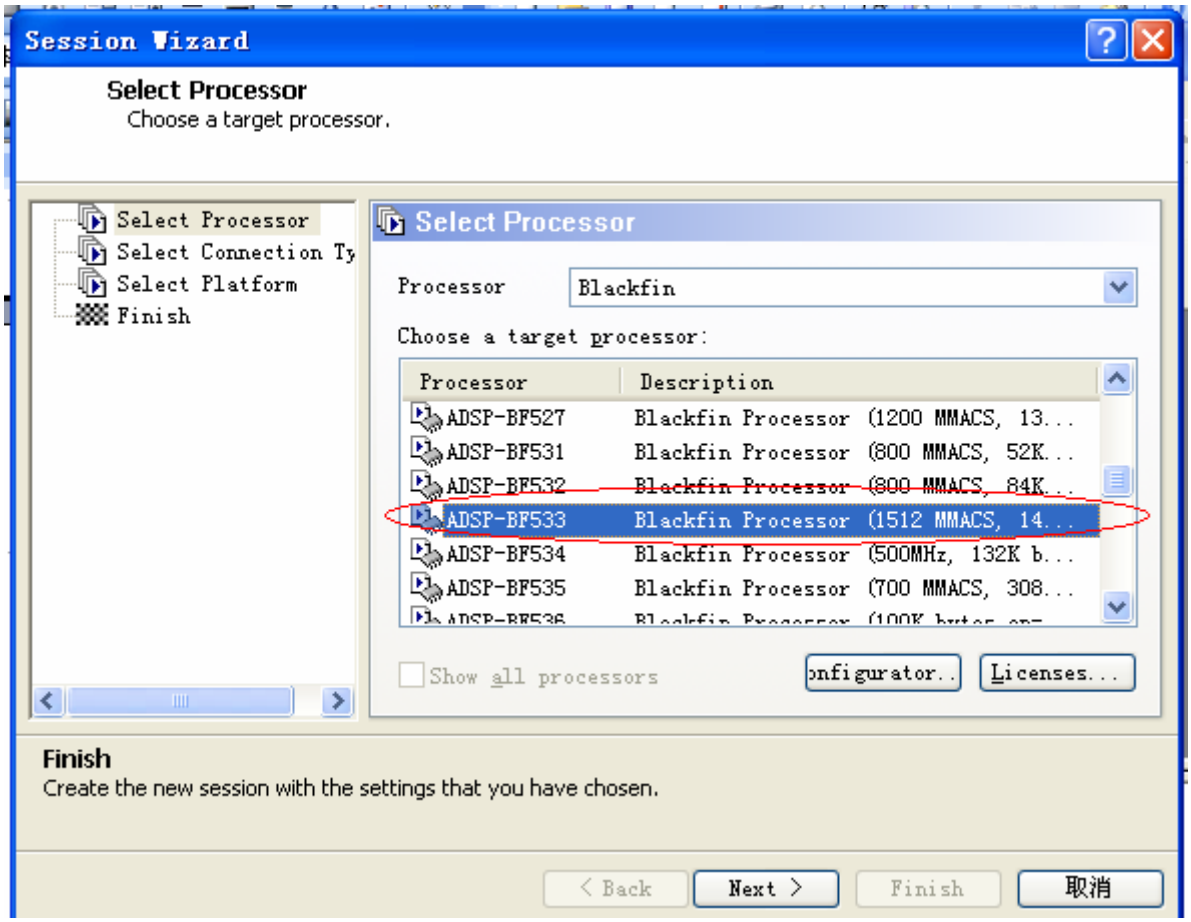


运行 VisualDSP++5.0 软件界面上，选择“New Session..”

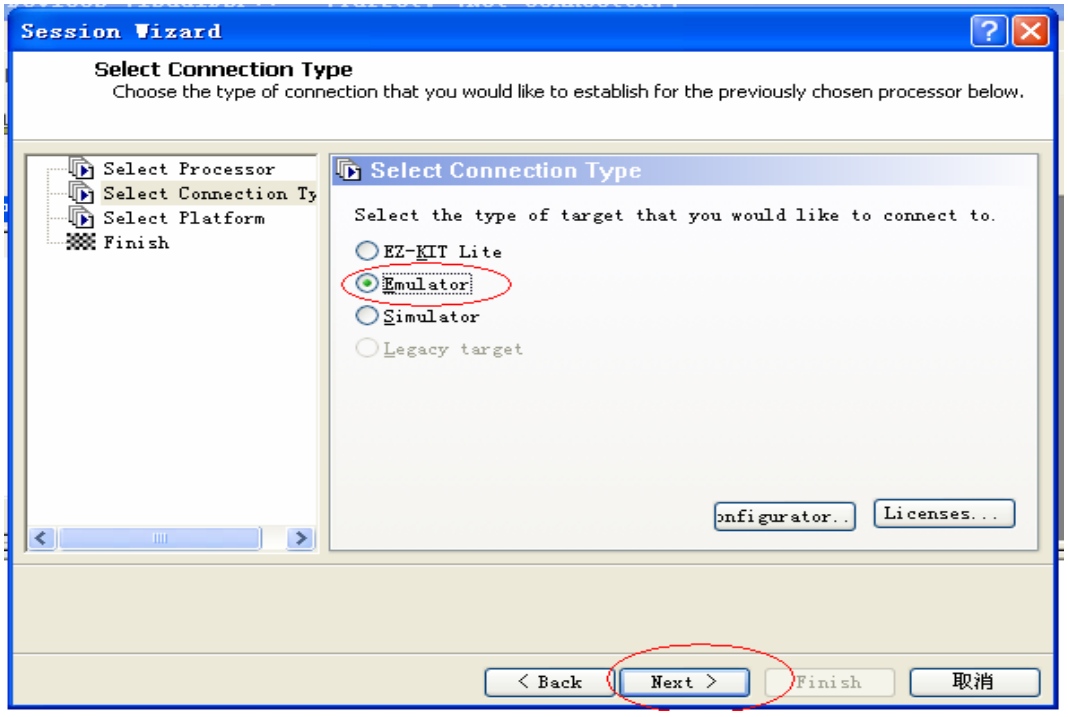


选择红框圈的选项

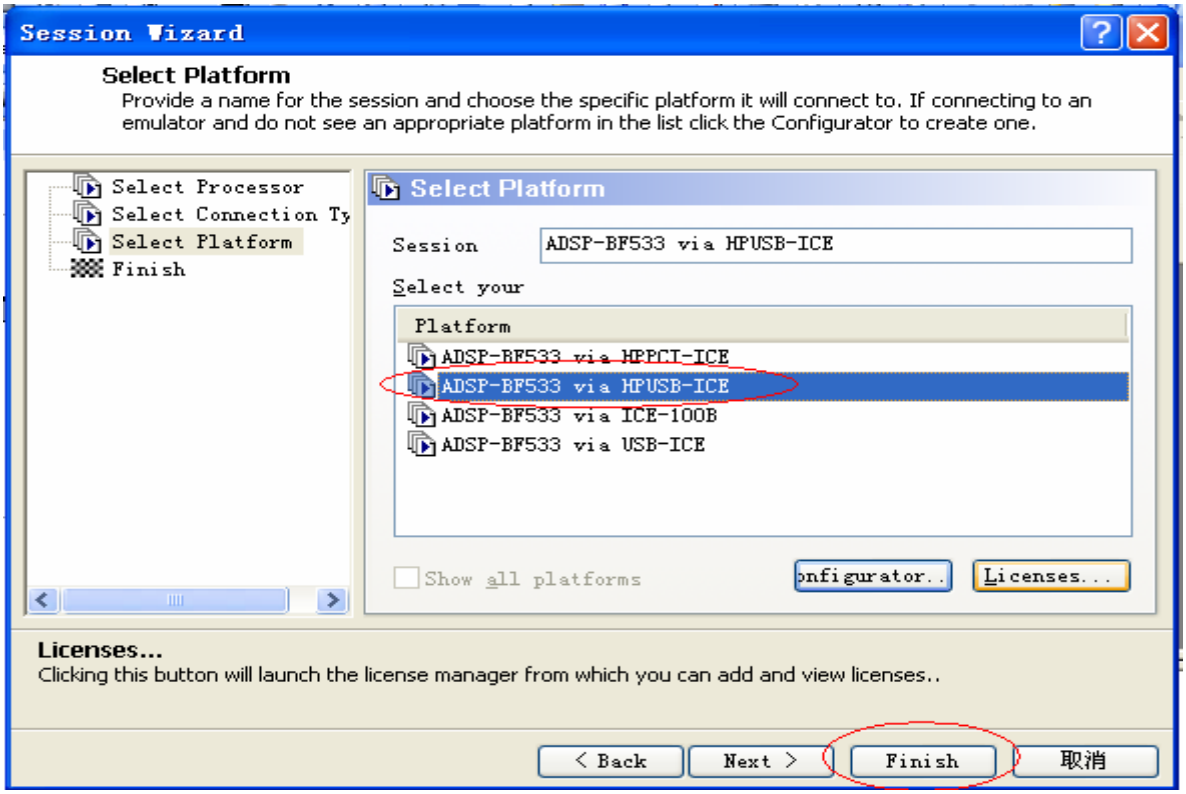
ADSP-EDU-BF53x 开发板以后会根据客户需求，处理器会使用 BF531,BF532,BF533，但我们的所有工程代码都是以 ADSP-BF533 Session 下开发，BF531,BF532,BF533 有着相同的内核和外设，管脚完全兼容，所以不管开发板上焊接的芯片是哪一款，都应以 ADSP-BF533 的 Session 开发，把板卡上的处理器都作为 BF533 400MHz 的主频来使用。



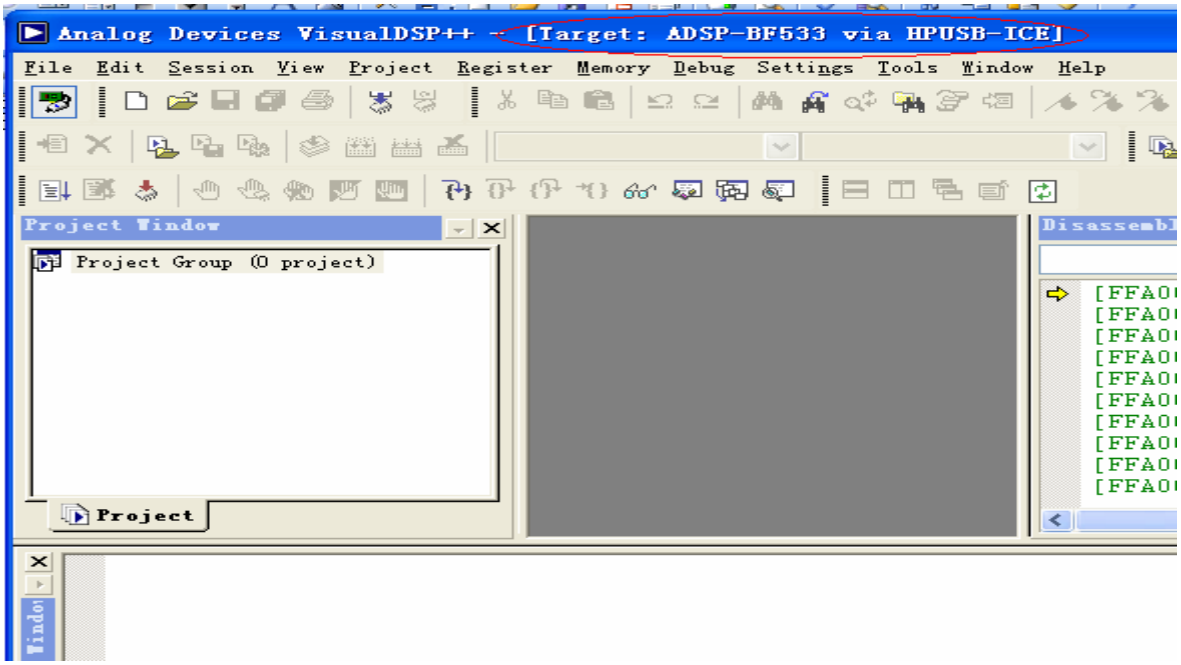
选择“Emulator”后，点“Next>”



选择“ADSP-BF533 via HPUSB-ICE”后，点“Finish”



连接成功后在 VisualDSP++5.0 软件界面上可以看到连接状态，该状态代表目前仿真器在连接着板卡



Blackfin 入门教程

Blackfin 系列处理器的初学教程很少，很多初学者不知如何来上手这款 DSP。
为方便初学者更快的学会使用 DSP，成为 DSP 高手，本章节将详细介绍 DSP 的接口使用。

BF53x_GPIO

接口功能介绍

ADSP-BF53x 处理器上有 16 个 Programmable Flag 接口，简称 PF 接口，这些接口就是通常所有的 IO 接口，通过寄存器配置，可以输出电平和感知接口电平，每一个 PF 接口都可以作为外部中断接口。

在单片机上，通常如果设置一个 IO 接口输出时，直接将输出信号值付给该接口，如果作为输入时，直接通过该接口读取即可。Blackfin 处理器的 IO 使用与单片机不同，在使用前必须对该接口进行初始化，如告知接口的方向，如配置为输出接口，则直接配置输出接口电平信号，如配置为输入接口，需打开输入使能开关，配置输出信号触发方式，是否中断触发，是否双极性触发等等。初始化完成后，才能使用 PF 接口。

接口寄存器说明

PF 接口主要寄存器功能与使用方法

PF 寄存器	功能
FIO_FLAG_D	数据寄存器：通过该寄存器写入值设置接口电平和读取该寄存器值获取接口电平
FIO_FLAG_C	清楚标志位寄存器：将该寄存器内写 1，相对应的 PF 管脚被清楚，电平置为 0。写 0 则无效
FIO_FLAG_S	设置标志位寄存器：将该寄存器内写 1，相对应的 PF 管脚被设置，电平置为 1。写 0 则无效
FIO_MASKA_D	中断屏蔽数据寄存器 A：设置 PF 管脚是否使用中断功能，写 1 则使用中断，写 0 则关闭中断
FIO_MASKA_C	中断屏蔽清楚寄存器 A：清楚 PF 管脚的中断功能，写 1 则对应管脚关闭中断，写 0 则无效
FIO_MASKA_S	中断屏蔽设置寄存器 A：设置 PF 管脚的中断功能，写 1 则对应管脚使用中断，写 0 则无效
FIO_MASKB_D	中断屏蔽数据寄存器 B：设置 PF 管脚是否使用中断功能，写 1 则使用中断，写 0 则关闭中断
FIO_MASKB_C	中断屏蔽清楚寄存器 B：清楚 PF 管脚的中断功能，写 1 则对应管脚关闭中断，写 0 则无效
FIO_MASKB_S	中断屏蔽设置寄存器 B：设置 PF 管脚的中断功能，写 1 则对应管脚使用中断，写 0 则无效
FIO_DIR	方向设置寄存器：写 1 则对应管脚为输出，写 0 则对应管脚为输入
FIO_POLAR	极性设置寄存器：写 1 则 0 电平触发或下降沿触发，写 0 则高电平触发或上升沿触发
FIO_EDGE	沿触发寄存器：写 1 则对应管脚设置为沿触发，写 0 则对应管脚设置为电平触发
FIO_BOTH	双沿设置寄存器：写 1 则对应管脚设置为双沿触发，写 0 则对应管脚设置为单沿触发
FIO_INEN	输入使能寄存器：写 1 则对应管脚使能输入功能，写 0 则对应管脚关闭输入功能

例子代码分析

输入接口配置：

将 PF0 接口配置为输入接口，并且读出接口电平状态。

```
*pFIO_DIR &= ~PF0;          //设置 PF0 为输入
*pFIO_INEN |= PF0;          //输入使能
i = *pFIO_FLAG_D;           //读取数据
```

输出接口配置：

将 PF0 接口配置为输出接口，使用两种方式设置 PF0 输出高低电平。

```
*pFIO_DIR |= PF0;           //设置 PF0 为输出
*pFIO_FLAG_S |= PF0;        //PF0 脚置高
*pFIO_FLAG_C |= PF0;        //PF0 脚置低
*pFIO_FLAG_D |= PF0;        //PF0 脚置高
*pFIO_FLAG_D &= ~PF0;       //PF0 脚置低
```

代码实现功能

工程 BF53x_GPIO_IN.dpj 实现了读取 PF0 接口状态并打印出 PF 接口状态数据。

```
PF data is 1
PF data is 1
PF data is 1
PF data is 1
PF data is 1
PF data is 1
PF data is 1
PF data is 1
PF data is 0
PF data is 0
PF data is 0
PF data is 0
PF data is 1
PF data is 1
```

工程 BF53x_GPIO_OUT.dpj 实现了通过 PF0 接口不断的输出高低变化的电平。

测试结果

工程 BF53x_GPIO_IN.dpj：运行代码后将 PF0 接口的电平状态打印在 VDSP 上。

工程 BF53x_GPIO_OUT.dpj： 运行代码后 PF0 将不断变换高低电平。

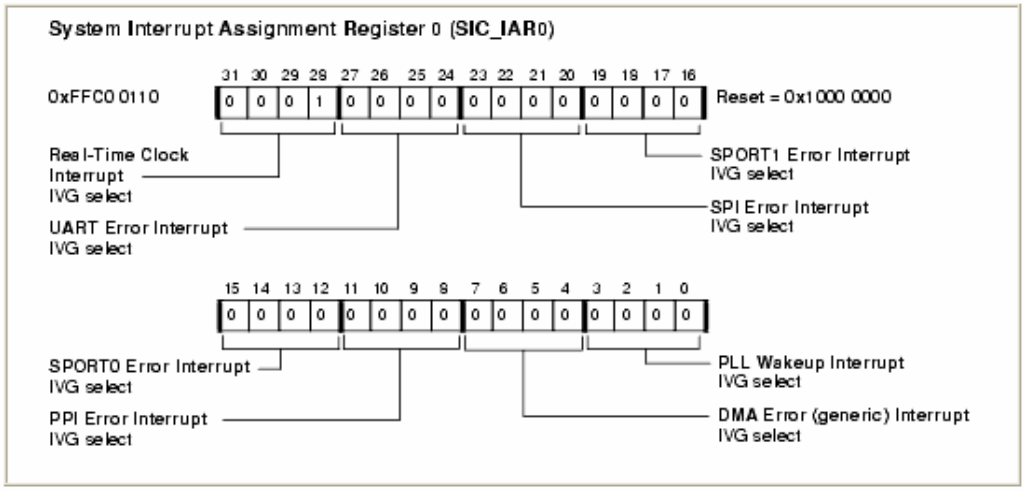
BF53x_GPIO_INTERRUPT

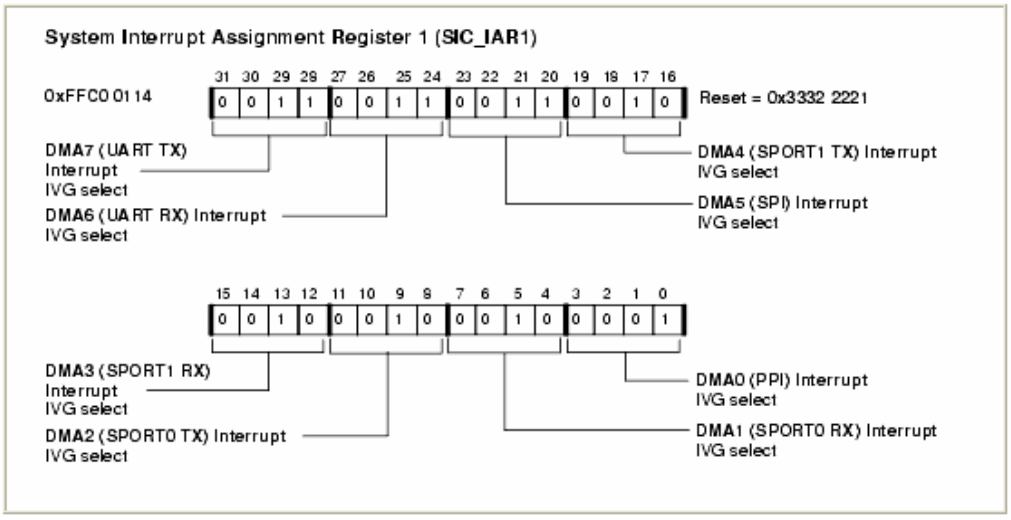
接口功能介绍

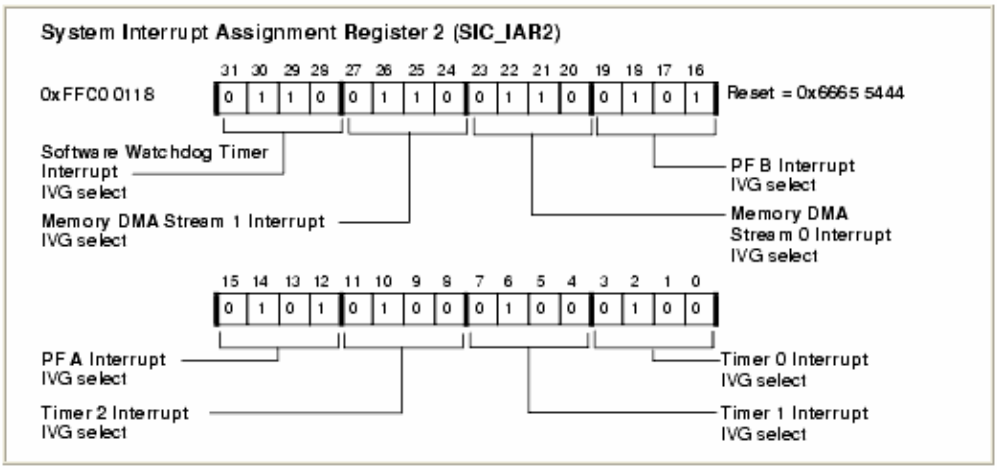
ADSP-BF53x 的 16 个 PF 接口都可以做为外部中断来使用。要使用 PF 的外部中断，需要为 PF 脚选择一个中断源，设置中断触发方式，为中断设置一个中断优先级，并且使能中断。

FIO_MASKA_D 和 FIO_MASKB_D：用来为 PF 管脚设置中断源，ADSP-BF53x 共有 PFA 和 PFB 两个中断源，通过选择配置这两个寄存器，使用不同的中断源。

SIC_IARx：设置中断优先等级。每个中断源都有一个默认的优先等级，如不对该寄存器配置，则可以使用默认的中断优先等级配置中断源。







从表中可以看出 PF 管脚相关的中断源 PFA 和 PFB 位于 SIC_IAR2，其默认配置值都为 5，根据其配置值，通过下表获知其对应的中断等级为 IVG12，如将 SIC_IAR2 配置值改为下表中的数值，则中断等级变为该数值对应的中断等级。

General-purpose Interrupt	Value in SIC_IAR
IVG7	0
IVG8	1
IVG9	2
IVG10	3
IVG11	4
IVG12	5
IVG13	6
IVG14	7
IVG15	8

SIC_IMASK：中断屏蔽寄存器，使能中断使用。

函数：

register_handler(ik_ivg12, FlagA_ISR)；

中断等级管理函数，该函数在头文件 “exception.h”中定义，定义该头文件后直接可以使用，其功能是告知中断管理器定义的中断标识符为 FlagA_ISR 和中断等级为 12 级。

EX_INTERRUPT_HANDLER(FlagA_ISR)

中断函数，该函数在头文件 “exception.h”中定义，当触发中断后，会进入该函数执行。

接口寄存器说明

寄存器	功能
SIC_IARx	中断等级设置寄存器

SIC_IMASK	中断屏蔽寄存器

例子代码分析

PF 口设置使用外部中断：

```
*pFIO_INEN      |= PF0|PF1;        //设置 PF0，PF1 输入使能
*pFIO_DIR       &=~(PF0|PF1);     //设置 PF0，PF1 为输入接口
*pFIO_EDGE      |= PF0|PF1;        //设置为沿触发
*pFIO_MASKA_D|= PF0|PF1;          //使用中断源为 PFA，使能 PF 外部中断
```

配置外部中断：

```
*pSIC_IAR0 = 0xffffffff;
*pSIC_IAR1 = 0xffffffff;
*pSIC_IAR2 = 0xffff5fff;          //设置中断等级参数为 5
register_handler(ik_ivg12, FlagA_ISR);      //告知中断管理器使用中断等级为 12，中断标志为 FlagA_ISR
*pSIC_IMASK = 0x00080000;         //使能外部中断
```

中断函数：

```
EX_INTERRUPT_HANDLER(FlagA_ISR)       //设置中断函数标志为 FlagA_ISR
{
        if(*pFIO_FLAG_D == PF0)               //判断是否为 PF0 中断
        {
            printf("interrupt is PF0!\n");
        }
        else if(*pFIO_FLAG_D == PF1)          //判断是否为 PF1 中断
        {
            printf("interrupt is PF1!\n");
        }
        *pFIO_FLAG_C = PF0|PF1;               //清楚中断标志

}
```

代码实现功能

通过 PF0 和 PF1 接口作为外部中断信号触发管脚，当有下降沿出发时进入中断函数，在中断函数中判断是哪一个 PF 脚设置了中断，打印出中断 PF 脚信息。

测试结果

运行代码后，利用接地的导线触发 PF0 和 PF1 管脚，会进入相应的中断，打印出中断信息。

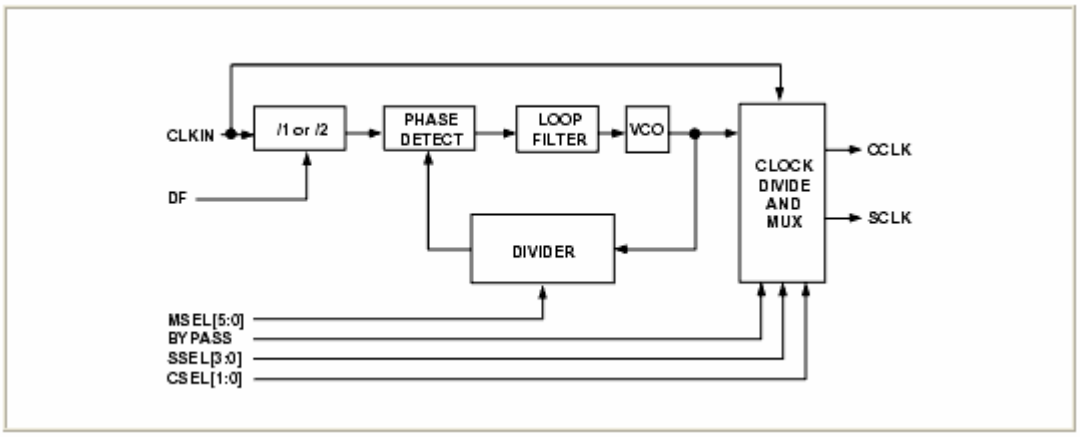
```
Loading: "D:\file\file\CY\2011UDC\bf531\cjcode\bf53x_
Load complete.
interrupt is PF0!
interrupt is PF1!
interrupt is PF0!
interrupt is PF0!
interrupt is PF1!
```

BF53x_PLL

接口功能介绍

PLL(Phase Locked Loop)是 ADSP-BF53x 的内核和时钟设置的机制，叫做锁相环。通过 PLL 配置当前处理器工作的内核和系统时钟。

PLL 机制如图：



输入时钟送给 ADSP-BF53x 后，通过 DF 设置是否对输入时钟分频，然后将根据 MSEL 的值对时钟进行倍频，倍频后将时钟送给 VCO，由 VCO 根据设置的分频系数，分出内核时钟和系统时钟。

Signal name MSEL[5:0]	VCO Frequency DF = 0	DF = 1
0	64x	32x
1	1x	0.5x
2	2x	1x
N = 3–62	Nx	0.5Nx
63	63x	31.5x

MSEL 占用 6Bit，最大可设置 64 倍倍频。通常情况下，该倍频频率不要超过芯片允许的最大频率。

Signal Name CSEL[1:0]	Divider Ratio VCO/CCLK	Example Frequency Ratios (MHz) VCO	CCLK
00	1	300	300
01	2	600	300
10	4	600	150
11	8	400	50

内核时钟分频系数占 2Bit，最大可设置 8 倍分频，当为 00 时，内核时钟等于 VCO 时钟。设置的内核时钟不要超过芯片允许的最高频率。

Signal Name SSEL[3:0]	Divider Ratio VCO/SCLK	Example Frequency Ratios (MHz) VCO	SCLK
0000	Reserved	N/A	N/A
0001	1:1	100	100
0010	2:1	200	100
0011	3:1	400	133
0100	4:1	500	125
0101	5:1	600	120
0110	6:1	600	100
N = 7–15	N:1	600	600/N

系统时钟分频系数占 4bit，最大进行 15 倍的分频。设置的系统时钟不要超过 133MHz。

接口寄存器说明

寄存器	功能
PLL_DIV	PLL 分频寄存器，设置系统时钟和内核时钟分频系数

PLL_CTL	PLL 控制寄存器，设置 VCO 倍频系数和一些控制开关
PLL_STAT	PLL 状态寄存器，获取芯片当前工作的状态
PLL_LOCKCNT	PLL 计数器，用于设置计数时钟

例子代码分析

```
*pPLL_DIV = pssel;                  //设置系统时钟分频系数，内核不做分频
asm("ssync;");                      //系统同步
new_PLL_CTL = (pmsel & 0x3f) << 9;      //将 VCO 倍频系数移位至需设置的位置
*pSIC_IWR |= 0xffffffff;            //将系统中断唤醒使能
if (new_PLL_CTL != *pPLL_CTL)       //判断是否已经配置过倍频系数
{
    *pPLL_CTL = new_PLL_CTL;    //配置倍频系数
    asm("ssync;");                  //系统同步
    asm("idle;");                   //将处理器设置为空闲
}
```

配置完 PLL 后，系统必须将系统设置为空闲后，系统再一次唤醒后，设置的值才会生效。

代码实现功能

代码实现了将内核时钟配置为 16 倍倍频，将系统时钟配置为 3 倍分频。板卡上输入时钟为 25MHz，所以 VCO 时钟配置后为 25*16 =400MHz，内核时钟没有做分频，所以内核时钟等于 VCO 时钟，也为 400MHz，系统时钟为 400/3=133MHz。

测试结果

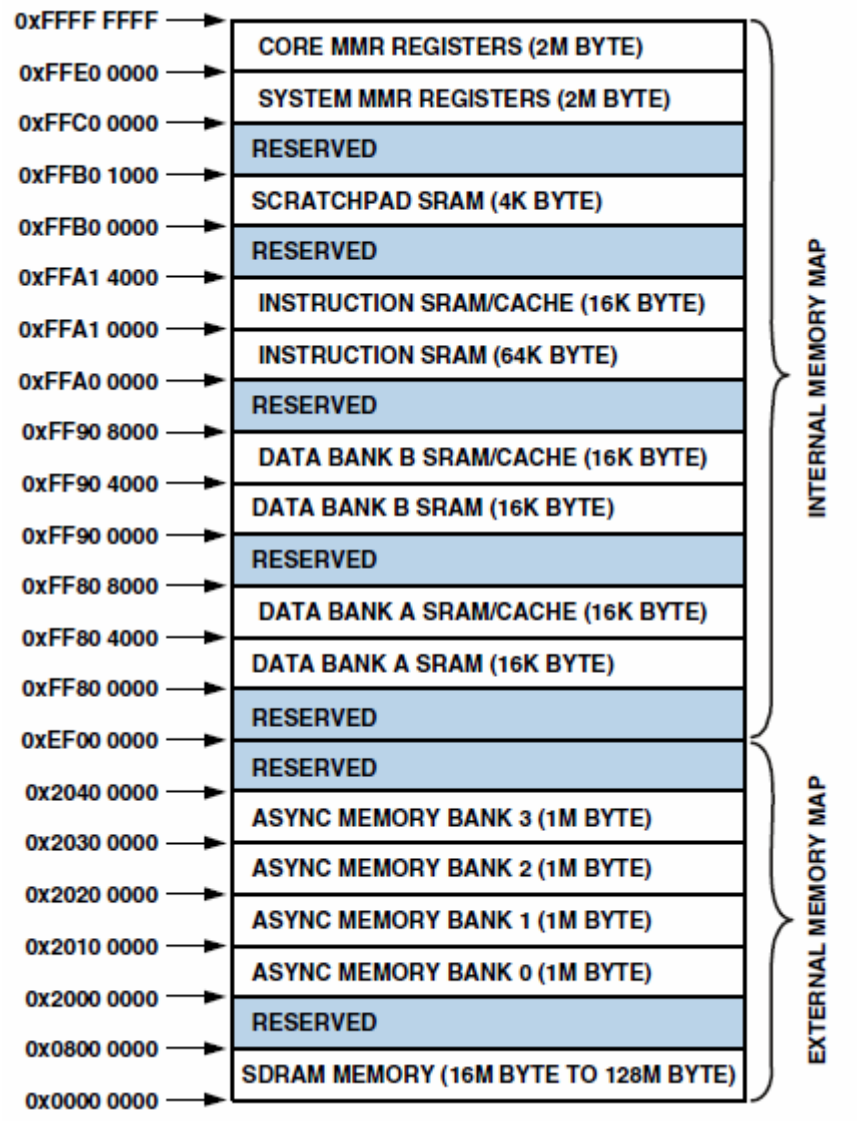
运行代码后，处理器的内核时钟会运行在 400MHz，系统时钟运行在 133MHz。

BF53x_EBIU

接口功能介绍

EBIU 接口是 ADSP-BF53x 的外部总线接口，ADSP-BF53x 的 EBIU 接口共有 16 根数据线，19 根地址线，支持同步的 SDRAM 接入和异步的总线外设接入，ADSP-BF53x 的异步 EBIU 接口共有 4 个 BANK，每个 BANK 1MByte，支持各种总线接口设备。

EBIU 接口采用指针方式访问，通过宏定义出要访问的地址，然后通过指针进行读写数据操作。



上图是 ADSP-BF53x 处理器的内存分配表，其中地址 0~0x08000000 为 SDRAM 地址，地址 0x20000000~0x203fffff 为 EBIU 的异步 Bank 地址。

接口寄存器说明

寄存器	功能
EBIU_AMBCTL0	BANK0，BANK1 时序配置寄存器
EBIU_AMBCTL1	BANK2，BANK3 时序配置寄存器
EBIU_AMGCTL	EBIU 使能寄存器

例子代码分析

```
#define pADDR     (volatile unsigned short *)0x1000    //定义一个指针，地址指向 0x1000

*pADDR = 0x1234;                    //向 0x1000 地址里写入数据 0x1234
i = *pADDR;                         //读出该地址数据
printf("addr is %x\n",pADDR);       //打印出当前访问的地址
printf("data is %x\n",i);           //打印出当前地址中的数据
  *pADDR = 0xaa55;                  //向 0x1000 地址里写入数据 0xaa55
i = *pADDR;                         //读出该地址数据
printf("addr is %x\n",pADDR);       //打印出当前访问的地址
printf("data is %x\n",i);           //打印出当前地址中的数据
```

代码实现功能

代码实现了通过 EBIU 接口访问 SDRAM 空间地址 0x1000，向 0x1000 地址中写入数据并读出，打印出访问的地址和读出的数据。

测试结果

```
Loading: "D:\file\file\CY\2011UDC\bf531\cjcode\bf53x_eb
Load complete.
addr is 1000
data is 1234
addr is 1000
data is aa55
```

BF53x_SPI

接口功能介绍

SPI 接口是 4 线串口，可以连接 SPIFLASH，SPI 接口的 AD，DA 等等。ADSP-BF53x 的 SPI 接口支持主机模式和从机模式，它有 7 个 SPI 从机片选，在主机模式下，它同时可以挂载 7 个 SPI 设备，还可以在主机模式或从机模式下进行 BOOT 启动。

SPI 管脚定义：

管脚定义	功能
MOSI	主输入从输出接口，根据主机和设备模式确定功能
MISO	从输入主输出接口，根据主机和设备模式确定功能
SCK	SPI 时钟

SPISELx	SPI 设备选则接口
SPISS	SPI 从机片选接口

SPI 接口时钟最快可以到系统时钟的 1/4，其配置公式为：

SCK Frequency = (Peripheral clock frequency SCLK)/(2 x SPI_BAUD)

接口寄存器说明

寄存器	功能
SPI_CTL	SPI 控制寄存器，配置 SPI 工作模式及相位等
SPI_FLG	SPI 从机选择寄存器，用于选择使用哪一个片选控制设备
SPI_STAT	SPI 状态寄存器，获取 SPI 当前工作状态
SPI_TDBR	SPI 数据传输寄存器
SPI_RDBR	SPI 数据接收寄存器
SPI_SHADOW	SPI_RDBR 的影子寄存器，可用于读取数据

例子代码分析

```
*pSPI_BAUD=2;          //配置速率为 1/4 系统时钟   SPI 速率  = SCLK/2*SPI_BAUD
*pSPI_FLG |=FLS2;      //选择 SPISEL2 接口
*pSPI_CTL = 0x1001|CPHA| CPOL; //配置模式为手动片选模式
*pSPI_CTL = (*pSPI_CTL | SPE);       //使能 SPI 接口

*pSPI_FLG &= ~FLG2;        //将 SPISEL2 拉到 0
while(!(*pSPI_STAT & SPIF));   //查看 SPI 传输状态是否完成
*pSPI_TDBR = 0x55;          //将数据送入 SPI 传输数据寄存器
*pSPI_FLG |= FLG2;         //将 SPISEL2 拉到 1，完成数据传输



*pSPI_FLG &= ~FLG2;        //将 SPISEL2 拉到 0
while(*pSPI_STAT & RXS)//查看 SPI 传输状态是否有数据需要接收
i = *pSPI_RDBR;            //读取数据
*pSPI_FLG |= FLG2;         //将 SPISEL2 拉到 1，完成数据传输
```

ADSP-BF53x 的 SPI 接口支持手动片选和自动片选两种模式，通过 SPI_CTL 寄存器的 CPHA 和 CPOL 位配置，

例子代码采用的是手动片选模式，每次读取数据和数据读取结束后需要通过代码来选通和关闭片选，自动片选的例子可以参考板卡驱动程序中的 SD 卡驱动代码。

代码实现功能

代码实现了采用 SPI 接口发送 0x55 数据和读取 SPI 接口数据。
由于没有相关硬件为 SPI 发送数据，所以代码只是为了学习 SPI 接口的使用，实现了读取和传输数据的功能，并不能查看发送数据和读取数据的结果。

测试结果

SPI 接口发送数据 0x55 后读取 SPI 接口数据。

BF53x_Timer

接口功能介绍

ADSP-BF53x 上有 3 个通用定时器，每个定时器有三种模式：

1. 脉冲宽度调制模式（PWM_OUT）
2. 脉冲宽度计数捕获模式（WDTH_CAP）
3. 外部事件模式（EXT_CLK）

接口寄存器说明

寄存器	功能
TIMERx_CONFIG	定时器配置寄存器，用于设置定时器工作模式
TIMERx_WIDTH	定时器宽度寄存器，设置输出波形脉冲宽度
TIMERx_PERIOD	定时器周期寄存器，设置输出波形的周期
TIMERx_COUNTER	定时器计数寄存器，读取捕获的脉冲数量
TIMER_ENABLE	定时器使能寄存器
TIMER_DISABLE	定时器关闭寄存器
TIMER_STATUS	定时器状态寄存器

例子代码分析

```
*pTIMER0_CONFIG         = 0x0019;          //配置定时器为 PWM 模式
```

```
*pTIMER0_PERIOD      = 0x00800000;    //设置周期为 0x00800000 个系统时钟
*pTIMER0_WIDTH       = 0x00400000;    //设置脉宽为 0x00400000 个系统时钟
*pTIMER_ENABLE       = 0x0001;        //使能 Timer0

*pSIC_IAR0 = 0xffffffff;
*pSIC_IAR1 = 0xffffffff;
*pSIC_IAR2 = 0xfffffff4;              //配置中断等级数据为 4
register_handler(ik_ivg11, TIMER0_ISR);   //注册中断等级为 11，标识符为 TIMER0_ISR
*pSIC_IMASK = 0x00010000;

EX_INTERRUPT_HANDLER(TIMER0_ISR) //标识符为 TIMER0_ISR 的中断函数
{
    *pTIMER_STATUS = 0x0001;          //清除定时器中断标志
    printf("timer0 interrupt !\n");   //打印信息
}
```

代码实现功能

代码实现了将定时器配置为 PWM_OUT 模式，通过定时器中断来定时一个 0x00800000 个系统的时间长度，定时完成后，在中断内打印信息。
定时器没有单独的计时功能，所以如果计时，可以采用 PWM_OUT 模式，利用定时器中断来进行计时，同时在芯片的 TIMER0 管脚上，会有 PWM 波形输出。

测试结果

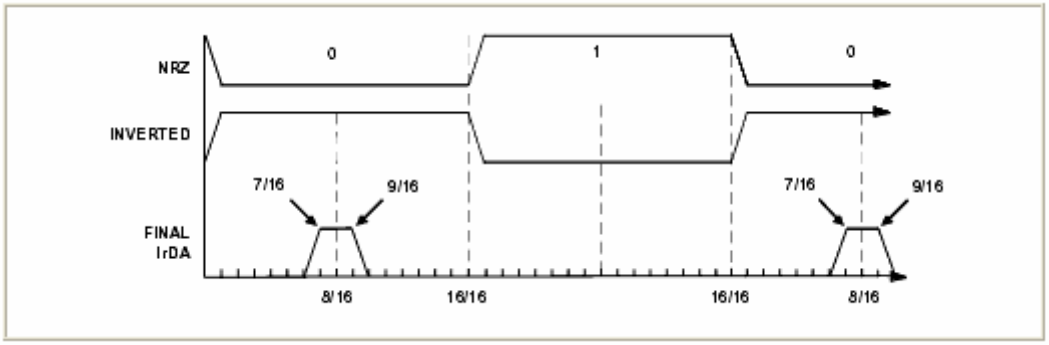
```
Loading: "D:\file\file\CY\2011UDC\bf531\cj
Load complete.
timer0 interrupt !
timer0 interrupt !
timer0 interrupt !
```

BF53x_UART

接口功能介绍

UART（Universal Asynchronous Receiver/Transmitter (UART) port）接口，是全双工通用的串行接口，由 RX 和 TX 两根线组成，扩展 RS232 芯片可以直接和计算机串口通讯，通常作为调试用的命令和数据通讯接口。
ADSP-BF53x 的 UART 接口，除了支持标准串口功能外，还支持 IrDA 模式，在硬件上增加一个红外通讯模块可以

进行红外数据传输。
当设置 IrDA 模式后，输出的波形会与原数据相反，且信号宽度变窄，下图是 IrDA 模式下和正常模式下的比较。



UART 接口通讯的波特率配置值可以通过下面公式进行计算：

BAUD RATE = SCLK/(16 x Divisor)

接口寄存器说明

寄存器	功能
UART_THR	UART 传输数据寄存器
UART_RBR	UART 接收缓存寄存器
UART_DLL	UART 波特率配置低 8 位寄存器
UART_DLH	UART 波特率配置高 8 位寄存器
UART_IER	UART 中断使能寄存器
UART_IIR	UART 中断识别寄存器
UART_LCR	UART 线路控制寄存器
UART_MCR	UART 调制控制寄存器
UART_LSR	UART 线路状态寄存器
UART_SCR	UART 暂存寄存器
UART_GCTL	UART 全局控制寄存器

例子代码分析

```
*pUART_GCTL=0x0009;
*pUART_LCR=0x0080;// DLAB=1  允许访问 DLL  和 DLH
*pUART_DLL=div;      //将变量 div 的值写入波特率配置寄存器
*pUART_DLH=div>>8; //DLL DLH 分别赋值
*pUART_LCR=0x0003;//    允许访问  RBR THR  和 IER
*pUART_IER=0x0001;//    接收中断允许
```

```
*pSIC_IAR0 = 0xffffffff;
*pSIC_IAR1 = 0xf3ffffff;        // UART  中断等级设置
*pSIC_IAR2 = 0xffffffff;
register_handler(ik_ivg10, UART_ISR);            // 注册 UART 中断等级为 10，标志为 UART_ISR
*pSIC_IMASK = 0x00004000;     //使能 UART 中断

*pUART_THR=TXbuf[i];            //向 UART 传输数据寄存器写数据
while(!(*pUART_LSR&0x0020));       //等待传输完成

EX_INTERRUPT_HANDLER(UART_ISR)            //UART 接收数据中断函数
{
    if(*pUART_LSR&DR)                //判断是否有新的数据。
    {
        if(cont>512)                //防止 buff 溢出，测试代码，将接收到的数据重复写入 512 字节的 buff
        cont = 0;
        RXbuf[cont]=*pUART_RBR;        //读取数据
        cont++;
    }
}
```

代码实现功能

代码实现了配置波特率为 9600，设定了数据接收中断，运行代码后，会将数组 Txbuf 中的字符串通过串口发送出，当接收到数据后，会进入中断函数读取数据。

测试结果

BF53x_SPORT

接口功能介绍

ADSP-BF53x 上有两个 SPORT 口，SPORT（synchronous serial ports）接口是 ADSP-BF53x 上速度最快的串口，其速度可以达到系统时钟的 1/2， 每一个 SPORT 口有两根接收数据线和两根传输数据线，支持全双工模式传输。
SPORT 接口通常用做一些高速的数据传输，它支持 I2S 模式，通常将 SPORT 接口连接音频的编解码器芯片，作为音频数据输出接口。
SPORT 时钟频率配置：

SPORTx_TCLK frequency = (SCLK frequency)/(2 x (SPORTx_TCLKDIV + 1))

SPORTx_RCLK frequency = (SCLK frequency)/(2 x (SPORTx_RCLKDIV + 1))

SPORT 同步信号频率配置：

SPORTxTFS frequency = (TSCLKx frequency)/(SPORTx_TFSDIV + 1)

SPORTxRFS frequency = (RSCLKx frequency)/(SPORTx_RFSDIV + 1)

不同模式下，寄存器配置值：

Bit Field	Stereo Audio Serial Scheme		
	I2S	Left-Justified	DSP Mode
RSFSE	1	1	0
RRFST	0	0	0
LARFS	0	1	0
LRFS	0	1	0
RFSR	1	1	1
RCKFE	1	0	0
SLEN	2 – 31	2 – 31	2 – 31
RLSBIT	0	0	0
RFSDIV (If internal FS is selected.)	2 – Max	2 – Max	2 – Max
RXSE (Secondary Enable is available for RX and TX.)	X	X	X

接口寄存器说明

寄存器	功能
SPORTx_TX_CONFIG	SPORTx 传输配置寄存器
SPORTx_RX_CONFIG	SPORTx 传输配置寄存器
SPORTx_TX	SPORTx 传输寄存器
SPORTx_RX	SPORTx 接收寄存器
SPORTx_TSCLKDIV	SPORTx 传输时钟配置寄存器
SPORTx_RSCLKDIV	SPORTx 接收时钟配置寄存器
SPORTx_TFSDIV	SPORTx 传输同步信号配置寄存器
SPORTx_RFSDIV	SPORTx 接收同步信号配置寄存器

SPORTx_STAT	SPORTx 状态寄存器

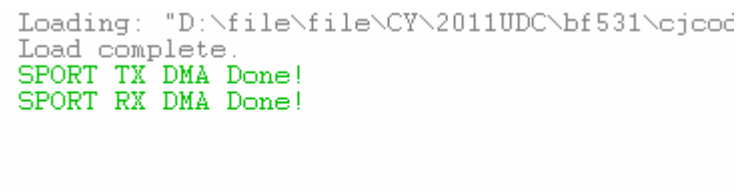
例子代码分析

```
*pSPORT0_TCLKDIV  =    TCLKDIV;          //配置 SPORT 传输接口的时钟频率
*pSPORT0_TFSDIV        =    TFSDIV;       //配置 SPORT 传输接口的同步频率
*pSPORT0_TCR1      =    ITFS|TFSR|ITCLK;//配置 SPORT 传输工作模式
*pSPORT0_TCR2      =    31;  //配置 SPORT 以 32Bit 数据传输

*pDMA2_PERIPHERAL_MAP = 0x2000;              //设置 SPORT 传输接口 DMA
*pDMA2_CONFIG = WDSIZE_32 | DI_EN |FLOW_1;    //设置 DMA 工作模式
*pDMA2_START_ADDR = (void *)iTxBuffer;       //设置 DMA  传输数据起始地址
*pDMA2_X_COUNT = 1000;                       //设置 DMA 传输次数
*pDMA2_X_MODIFY = 4;                         //设置 DMA 每次地址增量变化

*pDMA2_CONFIG      = (*pDMA2_CONFIG | DMAEN);     //使能传输 DMA
*pSPORT0_TCR1  = (*pSPORT0_TCR1 | TSPEN);         //使能传输 SPORT

*pSIC_IAR0 = 0xffffffff;
*pSIC_IAR1 = 0xfffff32f;                     //配置 SPORT DMA 中断等级
*pSIC_IAR2 = 0xffffffff;
register_handler(ik_ivg9, Sport0_RX_ISR);    //注册接收中断
register_handler(ik_ivg10, Sport0_TX_ISR);   //注册传输中断
*pSIC_IMASK = 0x00000600;                    //使能中断

EX_INTERRUPT_HANDLER(Sport0_TX_ISR)          //传输 DMA 中断函数
{
    *pDMA2_IRQ_STATUS = 0x0001;              //清楚中断标志位
    printf("SPORT TX DMA Done!\n");          //打印信息
    *pSIC_IMASK &= ~0x00000400;              //关闭传输中断
}
```

代码实现功能

代码实现了通过 SPORT 0 接口利用 SPORT0 DMA 传输数据和接收数据，SPORT 接口时钟和同步信号采用内部由系统时钟配置分频获取。
代码描述了 SPORT 接口使用 DMA 传输时常用的配置，没有和其他设备做通讯，所以看不到接收的实际数据。

测试结果



BF53x_PPI

接口功能介绍

PPI（Parallel Peripheral Interface ）接口在 ADSP-BF53x 上常用于视频信号和同步数据的传输，是半双工接口，支持数据的采集和数据的传输。

ADSP-BF533x 上有一个 16Bit 的 PPI 接口，最高速度可以到系统时钟的 1/2，有视频信号传输使用的行、列、场是三个同步信号，支持 ITU656,ITU601 等模式，可兼容大部分视频相关的芯片。

PPI 接口自身不能产生时钟信号，所以 PPICLK 信号必须由外部设备或者晶振提供，它没有专门的行，列同步信号管脚，在使用 PPI 时，需采用与其复用的 Timer1 和 Timer2 管脚来作为行列同步信号管脚，PPI 接口的场同步管脚 FS3 与 PF3 脚复用，该信号是在传输电视视频信号时，指示当前传输的信号是奇场还是偶场信号，在通常不使用的情况下，该管脚必须下拉。

PPI 接口与其他接口不同，他没有发送和接收数据的寄存器，不能采用 Core 来操作数据，只能采用 DMA 传输。

PPI 接口管脚与复用定义：

Signal Name	Function	Direction	Alternate Function
PPI15	Data	Bidirectional	PF4, SPI Enable Output
PPI14	Data	Bidirectional	PF5, SPI Enable Output
PPI13	Data	Bidirectional	PF6, SPI Enable Output
PPI12	Data	Bidirectional	PF7, SPI Enable Output
PPI11	Data	Bidirectional	PF8
PPI10	Data	Bidirectional	PF9
PPI9	Data	Bidirectional	PF10
PPI8	Data	Bidirectional	PF11
PPI7	Data	Bidirectional	PF12
PPI6	Data	Bidirectional	PF13
PPI5	Data	Bidirectional	PF14
PPI4	Data	Bidirectional	PF15
PPI3	Data	Bidirectional	N/A
PPI2	Data	Bidirectional	N/A
PPI1	Data	Bidirectional	N/A
PPI0	Data	Bidirectional	N/A
PPI_FS3	Frame Sync3/Field	Bidirectional	PF3, SPI Enable Output
PPI_FS2	Frame Sync2/VSYNC	Bidirectional	Timer 2
PPI_FS1	Frame Sync1/HSYNC	Bidirectional	Timer 1
PPI_CLK	Up to SCLK/2	Input Clock	N/A

接口寄存器说明

寄存器	功能
PPI_CONTROL	PPI 控制寄存器，用于配置 PPI 工作模式
PPI_STATUS	PPI 状态寄存器
PPI_COUNT	PPI 传输计数寄存器，设置图像一条线由多少数据组成
PPI_DELAY	PPI 延时计数寄存器，设置在传输时延时多少个时钟开始采数据
PPI_FRAME	PPI 帧寄存器，用来设置一幅完整图像一帧的线条数

例子代码分析

```
*pDMA0_START_ADDR = 0;      //配置 PPIDMA 数据起始地址
*pDMA0_X_COUNT = 480;       //配置 DMA 一行要传输多少次数据
```

```
*pDMA0_X_MODIFY = 2;        //配置每次传输行地址的增量
*pDMA0_Y_COUNT = 286;       //配置要传输多少行数据
*pDMA0_Y_MODIFY = 2;        //配置每次列数据地址的增量
*pDMA0_CONFIG = 0x1034;     //配置 DMA 工作模式

*pPPI_CONTROL = 0x781e;     //配置 PPI 工作偶是
*pPPI_DELAY = 0;            //配置时钟延时为 0
*pPPI_COUNT = 479;          //配置 PPI 每行要传输 480 次
*pPPI_FRAME = 286;          //配置每帧图像有 286 行

*pTIMER1_PERIOD     = 525;    //配置行同步信号产生的周期
*pTIMER1_WIDTH      = 41;     //配置行同步信号宽度
*pTIMER1_CONFIG     = 0x00a9;//配置行同步信号工作模式
*pTIMER2_PERIOD     = 150150;//配置列同步信号产生的周期
*pTIMER2_WIDTH      = 5250;   //配置列同步信号宽度
*pTIMER2_CONFIG     = 0x00a9; //配置列同步信号工作模式

*pDMA0_CONFIG |= 0x1;          //使能 DMA
asm("ssync;");                 //系统同步                    /
*pPPI_CONTROL |= 0x1;          //使能 PPI
asm("ssync;");                 //系统同步
*pTIMER_ENABLE|= 0x0006;       //使能行场同步信号
asm("ssync;");                 //系统同步
```

PPI 的行场同步信号与 TIMER1 和 TIMER2 复用，所以要配置 TIMER 寄存器来启动 PPI 的同步信号。

代码实现功能

代码实现了 PPI 连续发送 525*286 尺寸图像的数据，其中图像有效数据尺寸为 480*286。

测试结果

PPI 接口传输设置的数据。
该代码实现了使用 PPIDMA 传输数据的功能，没有实际的设备与其通讯来观察结果，如需要看接口，可以运行板卡驱动下的液晶屏代码，观察传输的图像数据。

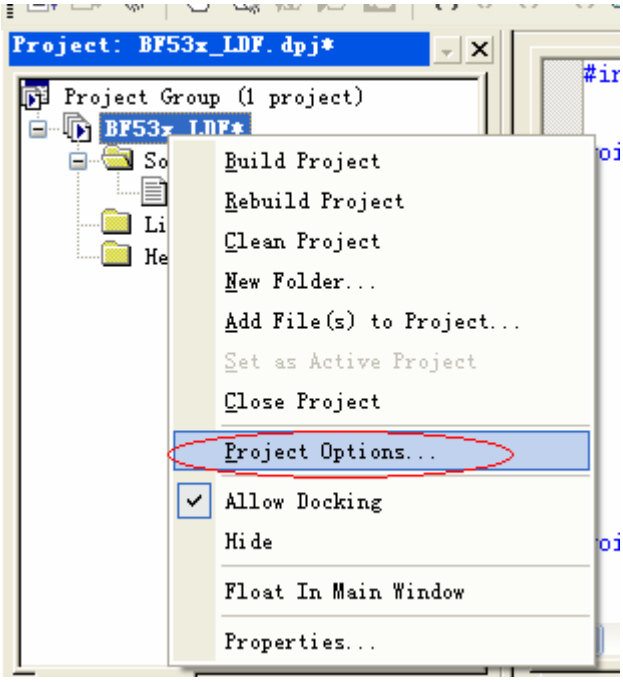
BF53x_LDF

模块功能介绍

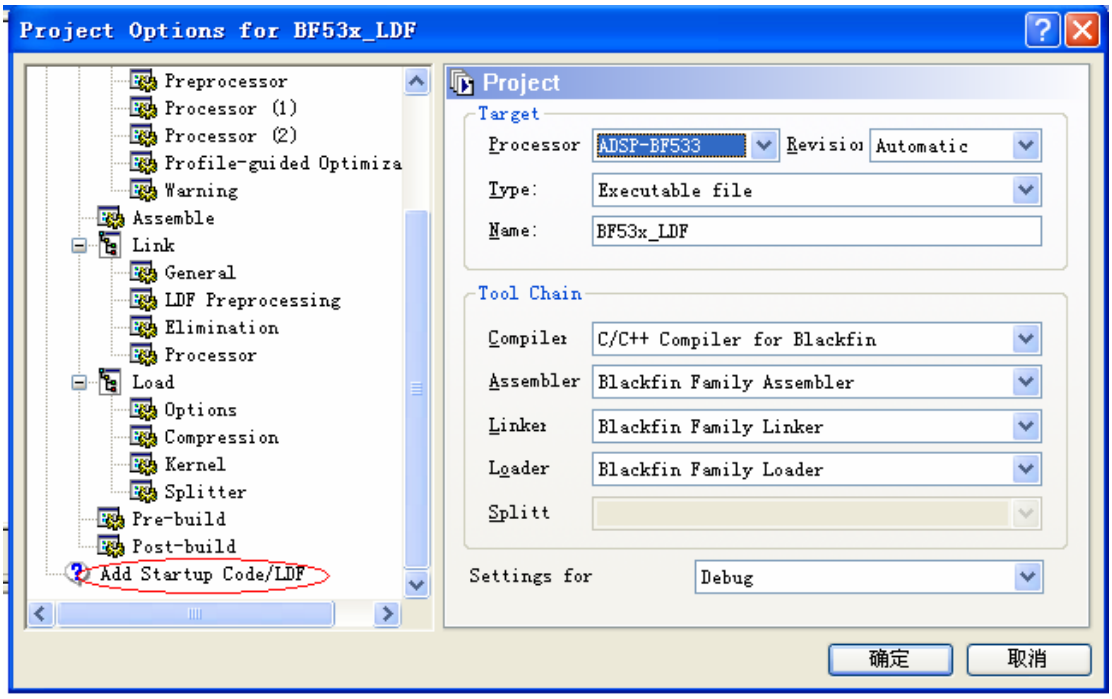
ADSP上的LDF（Linker Description Files）连接器描述文件是处理器用来进行资源分配的文件，通过对LDF文件的修改，可以分配自己需要的空间片断。通常情况下，不需要对LDF文件进行修改，通过简单指令即可使用LDF文件分配的空间，本章介绍一下如何通过LDF文件指定代码使用内存。

LDF 文件的生成

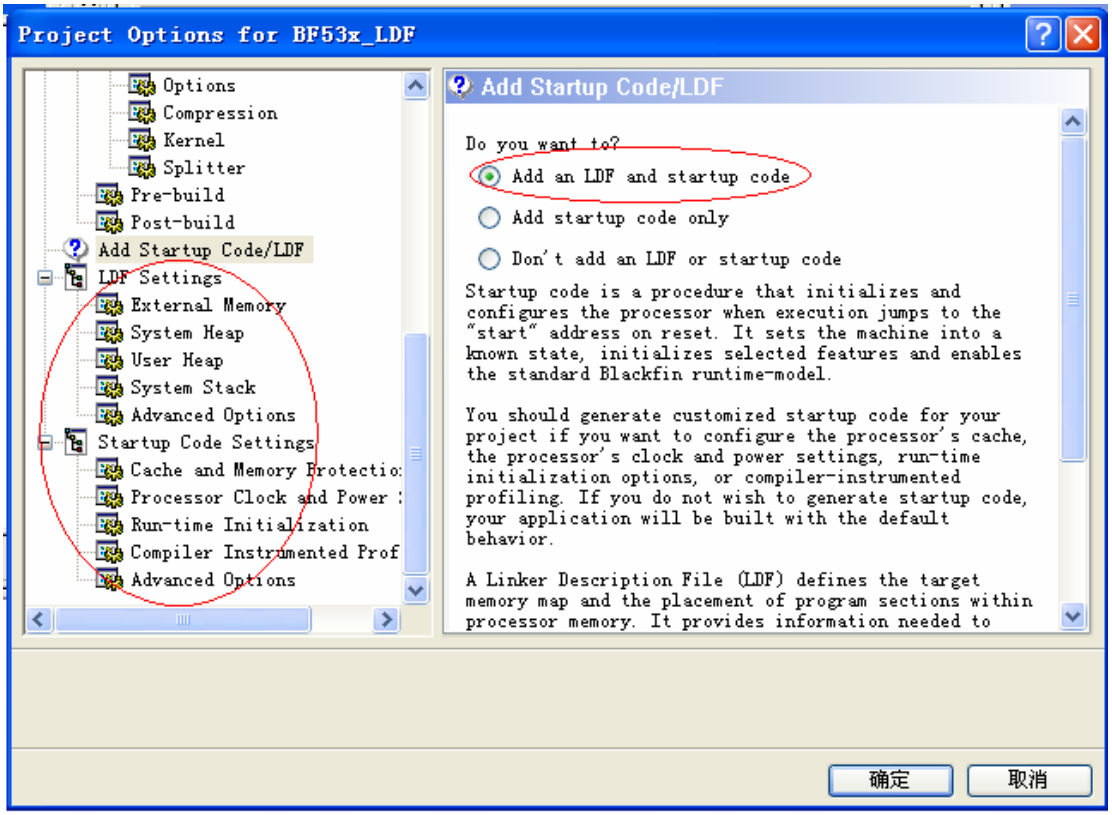
用 Visual DSP++ 5.0　软件，连接好板卡，打开要添加 LDF 文件的工程，在工程名上按鼠标右键，选择“工程选项”。



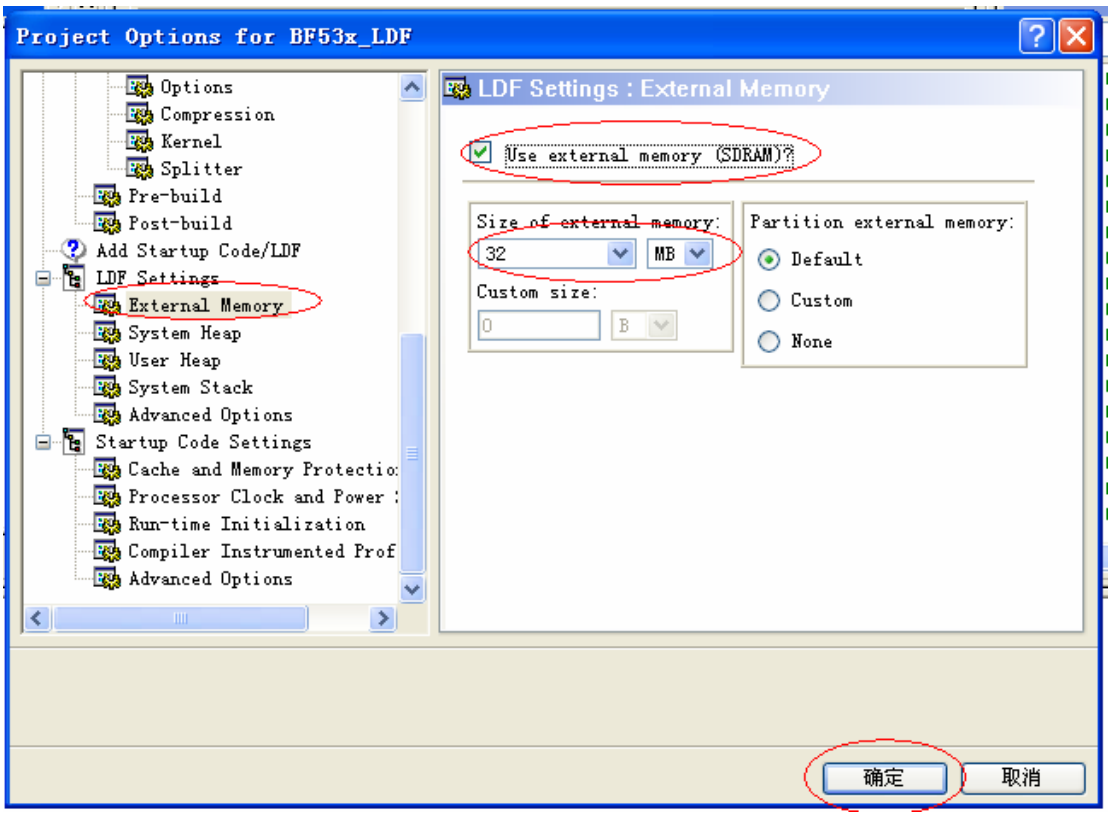
在弹出窗口中选择“add Startup Code/LDF”



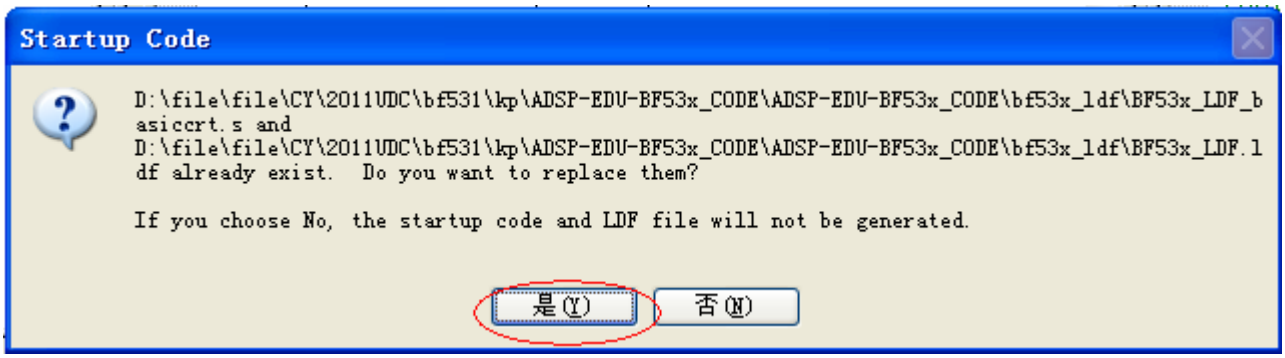
选择“Add an LDF and startup code”后，在左边窗口会出现关于 LDF 的选项操作。



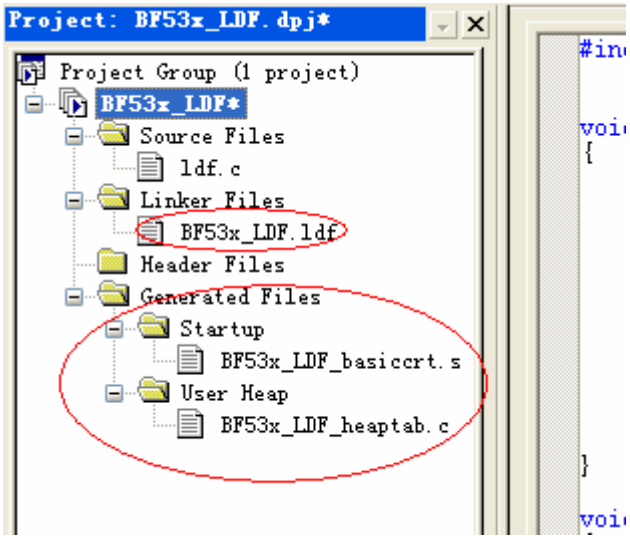
如图，把使用外部 SDRAM 选项勾上，选择与板卡 SDRAM 容量匹配的选项，ADSP-EDU-BF533 板卡的 SDRAM 容量为 32MB。选好后点“确定”。



在弹出的选择框上选择“是”



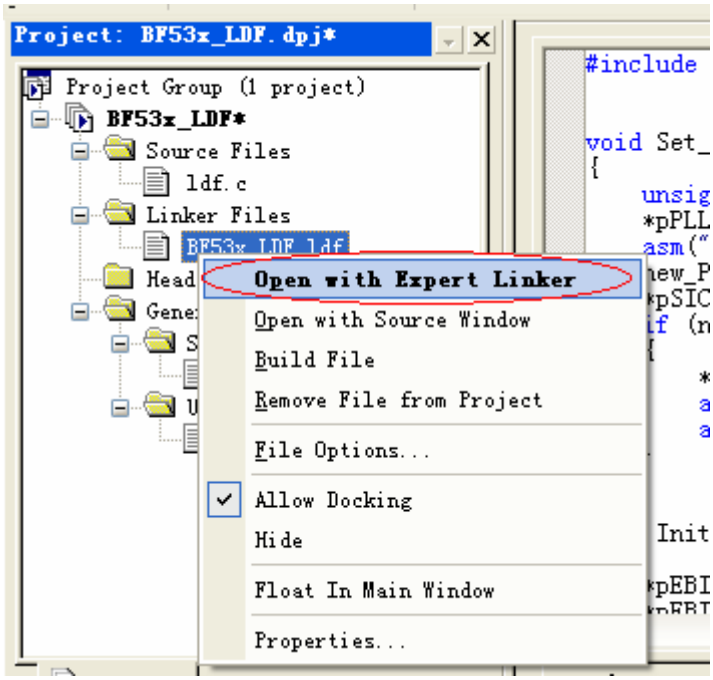
在工程里会自动添加 LDF 及其相关文件



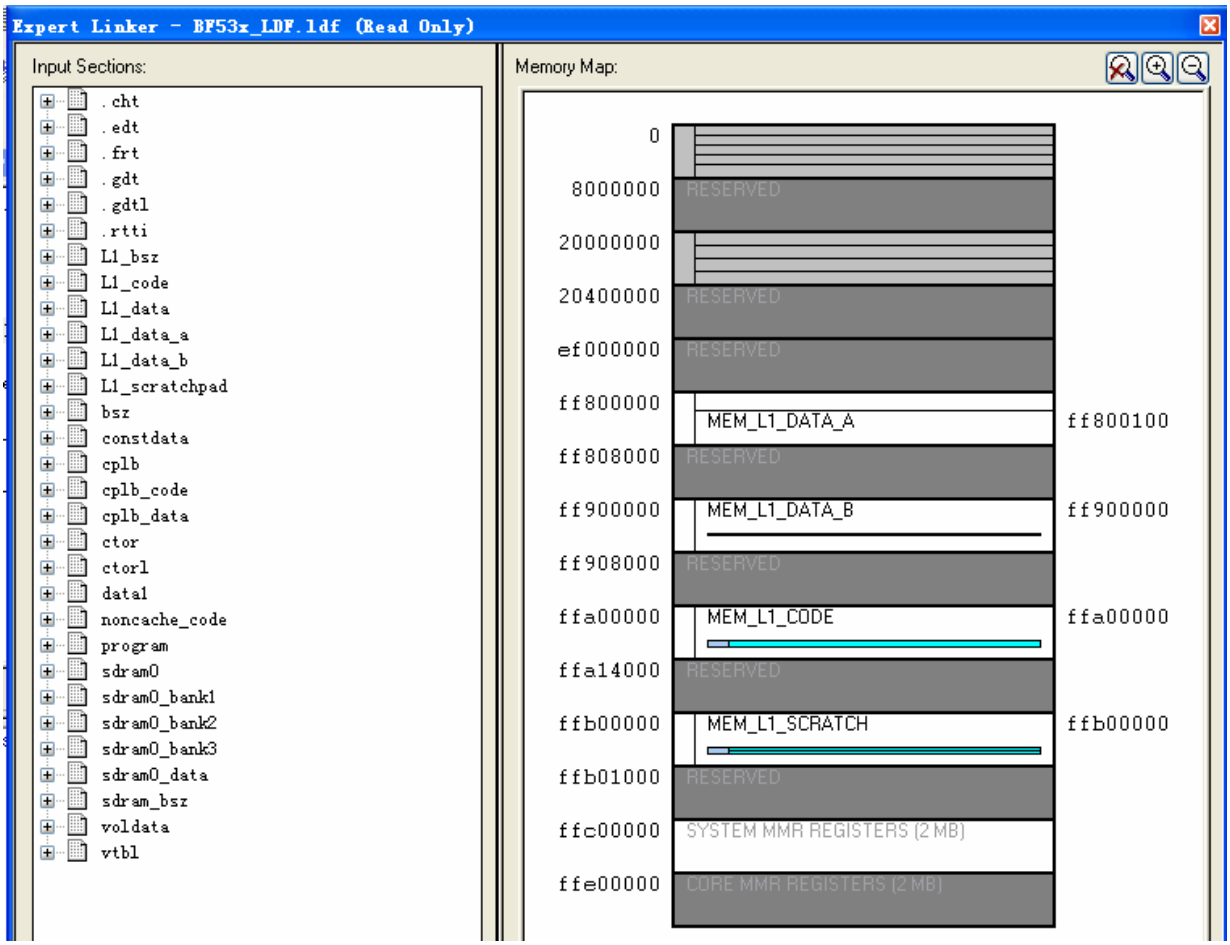
LDF 文件说明

使用图形打开

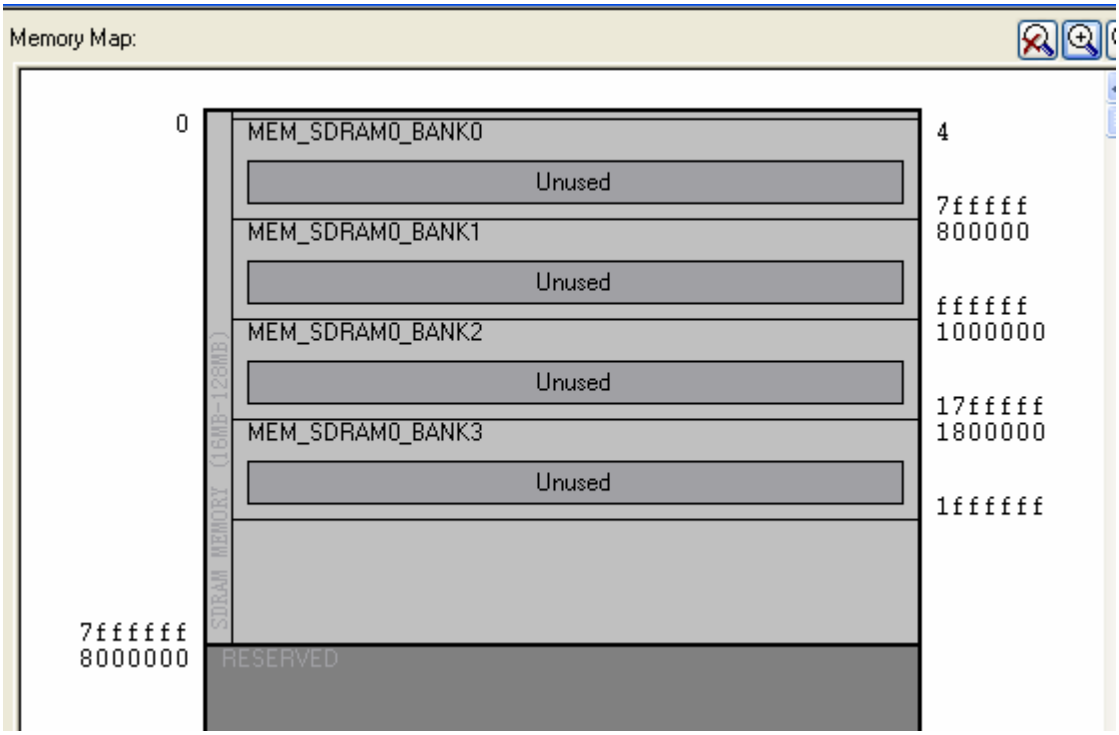
在 LDF 文件上进行双击鼠标左键，或者按右键在弹出菜单中按下图选择，会弹出 LDF 文件的图形界面。



图形界面分左右两个窗口，左边为 LDF 文件分配的空间片断的标识符。右边为 DSP 整个空间的分配地址和使用情况。从左边的标识符中可以看到 SDRAM 区域分为 sdram0,sdram0_bank1，sdram0_bank2，sdram0_bank3 等区域，如果要使用 SDRAM 的空间，就可以通过这些描述符来定义。

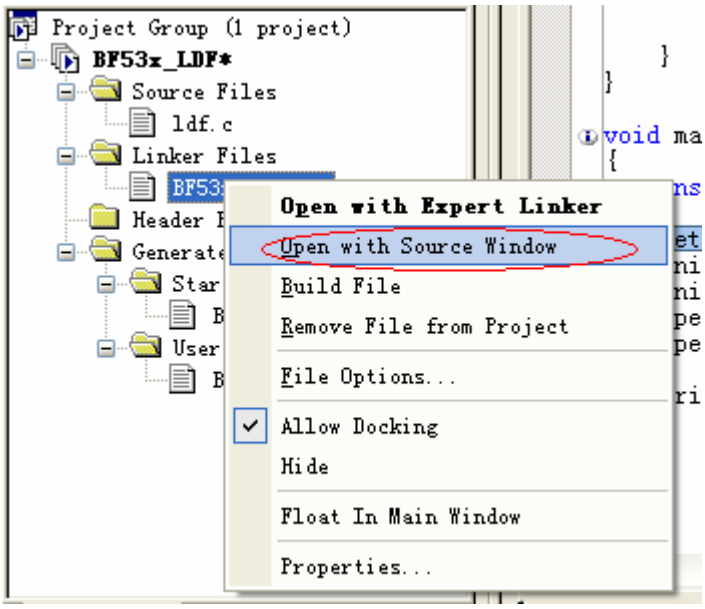


可以点击右上角放大按钮，将内存分布放大，观察当前内存使用情况。



使用代码打开

在 LDF 文件上按右键在弹出菜单中按下图选择，会以代码方式打开文件。



在代码中，可以找到如下图代码，该代码是指定 DSP 所使用空间的大小分配表，可以根据需要修改其空间分配的大小。

```
MEM_L1_SCRATCH          { TYPE(RAM) START(0xFFB00000) END(0xFFB00FFF) WIDTH(8) }
MEM_L1_CODE             { TYPE(RAM) START(0xFFA00000) END(0xFFA13FFF) WIDTH(8) }
MEM_L1_DATA_B           { TYPE(RAM) START(0xFF900000) END(0xFF907FFF) WIDTH(8) }
MEM_ARGV                { TYPE(RAM) START(0xFF800000) END(0xFF8000FF) WIDTH(8) }
MEM_L1_DATA_A           { TYPE(RAM) START(0xFF800100) END(0xFF807FFF) WIDTH(8) }
MEM_ASYNC3              { TYPE(ASYNC3_MEMTYPE) START(0x20300000) END(0x203FFFFF) WIDTH(8) }
MEM_ASYNC2              { TYPE(ASYNC2_MEMTYPE) START(0x20200000) END(0x202FFFFF) WIDTH(8) }
MEM_ASYNC1              { TYPE(ASYNC1_MEMTYPE) START(0x20100000) END(0x201FFFFF) WIDTH(8) }
MEM_ASYNC0              { TYPE(ASYNC0_MEMTYPE) START(0x20000000) END(0x200FFFFF) WIDTH(8) }
MEM_SDRAM0_BANK0        { TYPE(RAM) START(0x00000004) END(0x007fffff) WIDTH(8) }
MEM_SDRAM0_BANK1        { TYPE(RAM) START(0x00800000) END(0x00ffffff) WIDTH(8) }
MEM_SDRAM0_BANK2        { TYPE(RAM) START(0x01000000) END(0x017fffff) WIDTH(8) }
MEM_SDRAM0_BANK3        { TYPE(RAM) START(0x01800000) END(0x01ffffff) WIDTH(8) }
```

搜索“MEM_SDRAM0_BANK1”，可以找到如下代码，该代码中调用了很多库函数，来定义空间功能，最后对该空间片断做了标识符映射，其标识符为“sdram0_bank1”，通过调用该标识符，可以使用该空间。如果自己定义一个空间片断，可以在上面的空间地址中定义一个空间片断的地址区域，注意不要使空间重复，然后复制下面的代码，建立一个自定义空间的功能描述，修改为自己定义的描述符即可。

```
sdram0_bank1
{
    INPUT_SECTION_ALIGN(4)
    INPUT_SECTIONS($OBJECTS(sdram0) $LIBRARIES(sdram0))
    INPUT_SECTIONS($OBJECTS(sdram0_bank1) $LIBRARIES(sdram0_bank1))
    INPUT_SECTIONS($OBJECTS(sdram0_data) $LIBRARIES(sdram0_data))

    /*$VDSG<insert-input-sections-at-the-start-of-sdram0_bank1>  */
    /* Text inserted between these $VDSG comments will be preserved */
    /*$VDSG<insert-input-sections-at-the-start-of-sdram0_bank1>  */

    INPUT_SECTIONS($OBJECTS(cplb) $LIBRARIES(cplb))
    INPUT_SECTIONS($OBJECTS(data1) $LIBRARIES(data1))
    INPUT_SECTIONS($OBJECTS(voldata) $LIBRARIES(voldata))
    INPUT_SECTIONS($OBJECTS(constdata) $LIBRARIES(constdata))
    INPUT_SECTIONS($OBJECTS(cplb_data) $LIBRARIES(cplb_data))
    INPUT_SECTIONS($OBJECTS(vtbl) $LIBRARIES(vtbl))
    INPUT_SECTIONS($OBJECTS(.frt) $LIBRARIES(.frt))
    INPUT_SECTIONS($OBJECTS(.rtti) $LIBRARIES(.rtti))
    INPUT_SECTIONS($OBJECTS(.edt) $LIBRARIES(.edt))
    INPUT_SECTIONS($OBJECTS(.cht) $LIBRARIES(.cht))

    /*$VDSG<insert-input-sections-at-the-end-of-sdram0_bank1>  */
    /* Text inserted between these $VDSG comments will be preserved */
    /*$VDSG<insert-input-sections-at-the-end-of-sdram0_bank1>  */

} > MEM_SDRAM0_BANK1
```

在 LDF 文件定义空间时，常在空间片断描述符后面看到“ZERO_INIT”标志，该标志表示在编译代码时，将该段空间清为 0，也可以不对该空间操作，其标识符写作“NO_INIT”

将 sdram0_bank0 初始化时数据清为 0：

```
sdram0_bank0 ZERO_INIT
{
    INPUT_SECTION_ALIGN(4)
    INPUT_SECTIONS($OBJECTS(program) $LIBRARIES(program))
    INPUT_SECTIONS($OBJECTS(sdram0) $LIBRARIES(sdram0))

    /*$VDSG<insert-input-sections-at-the-start-of-sdram0_bank0>  */
    /* Text inserted between these $VDSG comments will be preserved */
    /*$VDSG<insert-input-sections-at-the-start-of-sdram0_bank0>  */

} > MEM_SDRAM0_BANK0
```

将 sdram0_bank0 初始化时保持原数据：

```
sdram0_bank0 NO_INIT
{
    INPUT_SECTION_ALIGN(4)
    INPUT_SECTIONS($OBJECTS(program) $LIBRARIES(program))
    INPUT_SECTIONS($OBJECTS(sdram0) $LIBRARIES(sdram0))

    /*$VDSG<insert-input-sections-at-the-start-of-sdram0_bank0>  */
    /* Text inserted between these $VDSG comments will be preserved */
    /*$VDSG<insert-input-sections-at-the-start-of-sdram0_bank0>  */

} > MEM_SDRAM0_BANK0
```

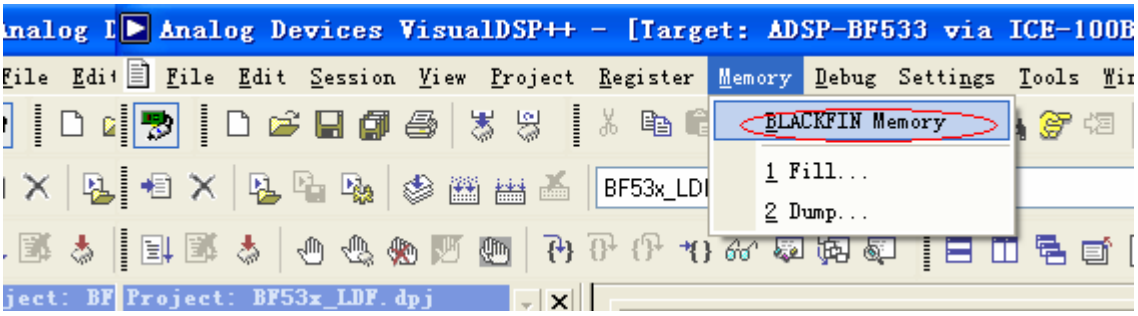
在正常使用时，原则上不建议大家来修改 LDF 的代码，以免因为 LDF 文件问题，为开发后期来带麻烦。

如何使用 LDF 文件定义的空间

要在代码中使用 LDF 文件定义的空间，可以通过“section(“ *** ”)”指令来定义，“***”代表空间片断标识符，即用图形界面打开，在左边窗口中看到的空间片断名称。
如定义一个数组“buffer[10000]”,将其放到“sdram0_bank1”的空间地址中,可以定义如下：

```
#include <cdefBF533.h>

section("sdram0_bank1") unsigned char buffer[100000];
```

在 VDSP 软件上，打开 memory 查看窗口



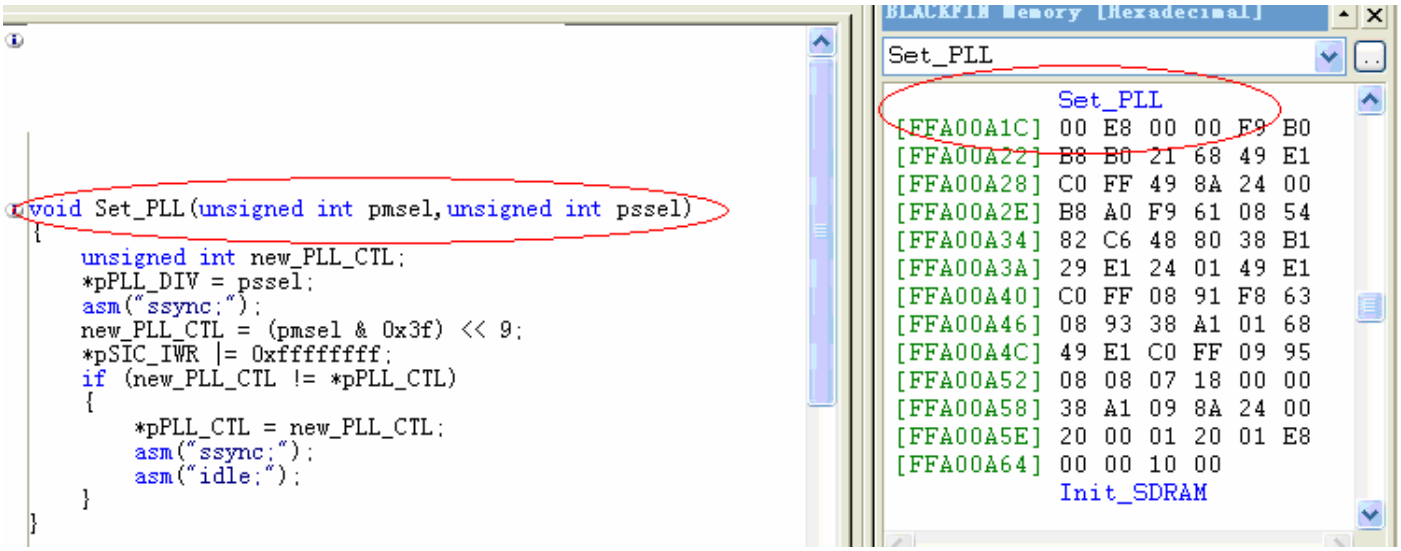
在窗口中输入定义的数组名称，按“回车”键，找到内存中该数组定义的位置。

从上图中可以看到，buffer 这个数组被定义在了 0x800000 这个地址区域，通过下图可以看出，这个地址位于 sdram0_bank1 的起始地址。

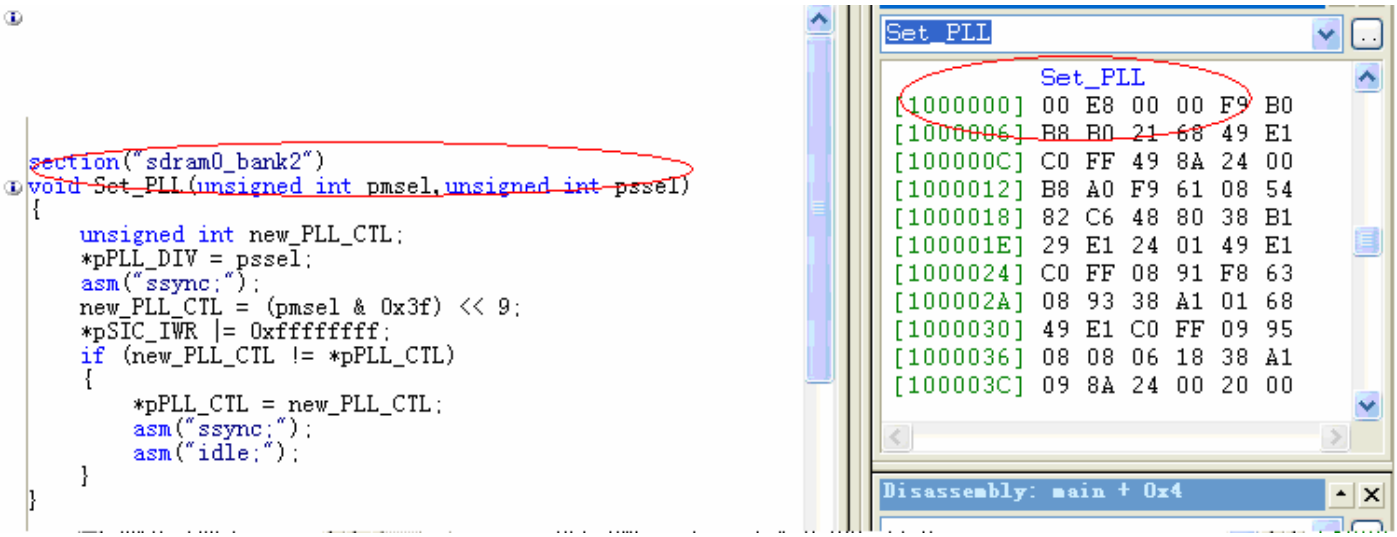
```
MEM_L1_SCRATCH          { TYPE(RAM) START(0xFFB00000) END(0xFFB00FFF) WIDTH(8) }
MEM_L1_CODE             { TYPE(RAM) START(0xFFA00000) END(0xFFA13FFF) WIDTH(8) }
MEM_L1_DATA_B           { TYPE(RAM) START(0xFF900000) END(0xFF907FFF) WIDTH(8) }
MEM_ARGV                { TYPE(RAM) START(0xFF800000) END(0xFF8000FF) WIDTH(8) }
MEM_L1_DATA_A           { TYPE(RAM) START(0xFF800100) END(0xFF807FFF) WIDTH(8) }
MEM_ASYNC3              { TYPE(ASYNC3_MEMTYPE) START(0x20300000) END(0x203FFFFF) WIDTH(8) }
MEM_ASYNC2              { TYPE(ASYNC2_MEMTYPE) START(0x20200000) END(0x202FFFFF) WIDTH(8) }
MEM_ASYNC1              { TYPE(ASYNC1_MEMTYPE) START(0x20100000) END(0x201FFFFF) WIDTH(8) }
MEM_ASYNC0              { TYPE(ASYNC0_MEMTYPE) START(0x20000000) END(0x200FFFFF) WIDTH(8) }
MEM_SDRAM0_BANK0        { TYPE(RAM) START(0x00000004) END(0x007fffff) WIDTH(8) }
MEM_SDRAM0_BANK1        { TYPE(RAM) START(0x00800000) END(0x00ffffff) WIDTH(8) }
MEM_SDRAM0_BANK2        { TYPE(RAM) START(0x01000000) END(0x017fffff) WIDTH(8) }
MEM_SDRAM0_BANK3        { TYPE(RAM) START(0x01800000) END(0x01ffffff) WIDTH(8) }
```

除了将一个数组定义在指定的空间片断，还可以指定一个函数，定义在指定的片断，其定义方法是在函数前面加入定义使用空间的指令。

通过 memory 查看器，可以看到在默认情况下，“Set_PLL” 函数是定义在地址 0xffa00a1c 地址，该代码断位于“L1_CODE”地址。



现指定“Set_PLL”函数将其放入“sdram0_bank2”空间



在 Visual DSP++ 5.0 up 8 软件上，如果定义的数组过大，超过了 L1 的空间，即使不使用“section”指令指定放置在 SDRAM 空间，系统也会自动将该数组放置在 SDRAM 空间中，但前提是必须使用该工程生成 LDF 文件。对于早期的 VDSP 软件版本，则没有该功能，必须指定数组放置的空间。

如果该工程没有自动生成 LDF 文件，也可以拷贝其他的 LDF 文件到该工程文件夹下，然后添加到该工程中，同样可以通过“section”指令指定使用。

代码实现功能

代码实现了通过工程生成 LDF 文件，通过“section”指令定义一个数组和一个函数的空间，通过 memory 查看器查看该数组和函数定义的空间地址。运行代码，代码会将变量 i 生成的数据写入 buffer 数组内。

测试结果

可以看到通“section”指令指定的数组和函数位于内存中位置的变化。

板卡驱动说明

BF53X_AUDIO

ADSP-EDU-BF53X 音频实验。

硬件实现原理

音频是采用 TI 公司的 TLV320AIC23B 音频 Codec 芯片，TLV320AIC23B 支持 1 路 MICIN，1 路 LINEIN, 1 路 OUT，1 路 HPOUT。硬件设计中将 1 路 LINEIN 和 1 路 HPOUT 通过接口引出。

BF53x 处理器的 SPORT 接口支持 IIS 协和和 TDM 协议，可直接与 TLV320AIC23B 的接口连接。通过 BF53x 处理器的 PF0 和 PF1 接口分别模拟 IIC 的 SCL 和 SDA 总线，用来初始化 TLV320AIC23B 芯片。TLV320AIC23B 支持多种音频采样格式，硬件设计中为其提供的时钟为 12MHz，可参考 TLV320AIC23B 数据手册将其配置为 USB 模式采样。

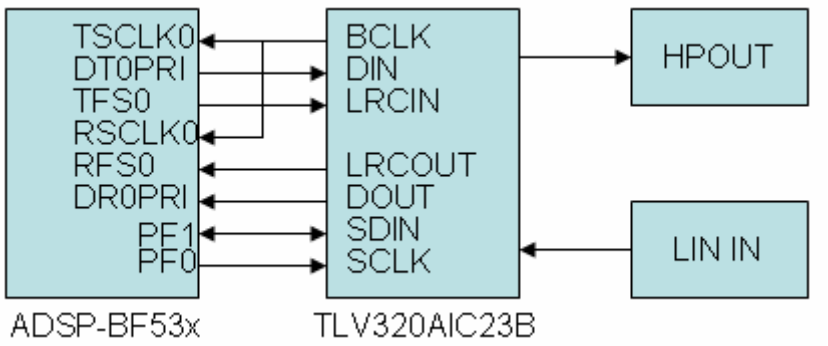
3.3.2.1 USB-Mode Sampling Rates (MCLK = 12 MHz)

In the USB mode, the following ADC and DAC sampling rates are available:

SAMPLING RATE†		FILTER TYPE	SAMPLING-RATE CONTROL SETTINGS				
ADC (kHz)	DAC (kHz)		SR3	SR2	SR1	SR0	BOSR
96	96	3	0	1	1	1	0
88.2	88.2	2	1	1	1	1	1
48	48	0	0	0	0	0	0
44.1	44.1	1	1	0	0	0	1
32	32	0	0	1	1	0	0
8.021	8.021	1	1	0	1	1	1
8	8	0	0	0	1	1	0
48	8	0	0	0	0	1	0
44.1	8.021	1	1	0	0	1	1
8	48	0	0	0	1	0	0
8.021	44.1	1	1	0	1	0	1

† The sampling rates are derived from the 12-MHz master clock. The available oversampling rates do not produce exactly 8-kHz, 44.1-kHz, and 88.2-kHz sampling rates, but 8.021 kHz, 44.117 kHz, and 88.235 kHz, respectively. See Figures 3–17 through 3–34 for filter responses

硬件连接示意图



初始化配置

TLV320AIC23B 的器件地址可通过接口上的 CS 引脚进行选择，如下表：

CS 状态	TLV320AIC23B 写器件地址	TLV320AIC23B 读器件地址

0	0x34	0x35
1	0x36	0x37

TLV320AIC23B 需要通过 IIC 接口配置初始化，所以需要通过配置板卡上的 CPLD 寄存器，将 PF0 和 PF1 配置为 IIC 总线模式，该配置映射在 CPLD 的 DEVICE_OE 寄存器，其配置功能为：

DEVICE_OE 寄存器地址：0x20320000

DEVICE_OE 寄存器（写唯一）：

DEVICE_OE 寄存器设置硬件设备上一些控制管脚的电平状态，该寄存器只能写入数据，不能读出当前数据。

DEVICE_OE 寄存器位功能：

Bit 位	7	6	5	4	3	2	1	0
功能	保留	INTERRUPT_OE	SPI_OE	SYNCINT_OE	PTS_OE	LCD_OE	SPI_SEL	LED_OE

INTERRUPT_OE：

1：关闭 I2C_SCL 输入信号，使能 PF0 中断信号

0：使能 I2C_SCL 输入信号，关闭 PF0 中断信号

使用 IIC 配置 TLV320AIC23B 时，需将 INTERRUPT_OE 设置为 0，关闭中断，待 IIC 配置完成后，再将中断 INTERRUPT_OE 打开。

代码实现功能

代码实现了一个音频输入播放的功能，将一个声源通过图 3 中 LINEIN 接口输入，将一音响或耳机连接在图 3 中 HPOUT 接口，运行代码后，音响中能听到输入声源的声音。

代码实现原理

代码通过 IIC 初始化完 TLV320AIC23B 后，TLV320AIC23B 开始通过 LINEIN 接口采集模拟音频数据，并将采集到的数据通过 ADSP-BF53x 的 SPORT 口传送给 ADSP-BF53x，ADSP-BF53x 将数据做内存交换后，再通过 SPORT 口传送给 TLV320AIC23B，TLV320AIC23B 将数据转为模拟信号后通过 HPOUT 接口输出。

测试实验步骤

1. 将仿真器（ICE）与 ADSP-EDU-BF53x 开发板和计算机连接好。
2. 将测试的音源通过标准音频连接线接入开发板的 Lin IN 接口（蓝色接口）接入，音箱连接 HPOUT 接口（绿色接口）。
3. 先给仿真器（ICE）上电再为 ADSP-EDU-BF53x 开发板上电。
4. 运行 VisualDSP++ 5.0 软件，选择合适的 BF533 的 session 将仿真器与软件连接。
5. 加载 VisualDSP++ 5.0 工程文件 BF53x_AUDIO.dpj，编译并全速运行。

测试结果

在音响或耳机中可以听到输入声源的声音。

BF53x_INTERRUPT

ADSP-EDU-BF53X 中断实验

硬件实现原理

ADSP-EDU-BF53x 开发板上的中断资源连接到了 CPLD，并通过 CPLD 将中断信号连接到 PF0 触发，通过 CPLD 映射的寄存器读取中断源数据。

中断功能映射到 CPLD 寄存器中的 DEVICE_OE 和 INTERRUPT_DAT 两个寄存器，其映射内容如下：

DEVICE_OE 寄存器（写唯一）：

DEVICE_OE 寄存器地址：0x20320000

DEVICE_OE 寄存器设置硬件设备上一些控制管脚的电平状态，该寄存器只能写入数据，不能读出当前数据。

DEVICE_OE 寄存器位功能：

Bit 位	7	6	5	4	3	2	1	0
功能	保留	INTERRUPT_OE	SPI_OE	SYNCINT_OE	PTS_OE	LCD_OE	SPI_SEL	LED_OE

INTERRUPT_OE：

1：关闭 I2C_SCL 输入信号，使能 PF0 中断信号

0：使能 I2C_SCL 输入信号，关闭 PF0 中断信号

使用中断时，将 INTERRUPT_OE 设置为 1 。

INTERRUPT_DAT 寄存器（读唯一）：

INTERRUPT_DAT 寄存器地址：0x20360000

INTERRUPT_DAT 寄存器是板卡上所有中断资源的中断源数据寄存器，可以通过该寄存器数据判断出当前中断是哪一个设备产生的。

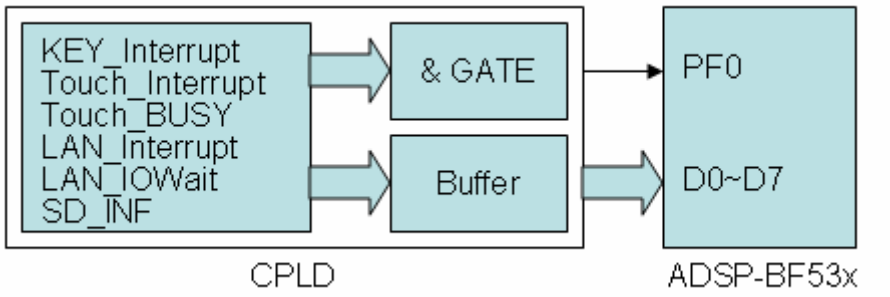
INTERRUPT_DAT 寄存器位功能：

Bit 位	7	6	5	4	3	2	1	0
功能	保留	保留	SD_INF	LAN_INT	LAN_IOWAIT	TOUCH_BUSY	TOUCH_INT	KEY_INT

当中断未触发时，读取的 Bit 位值为 1，当中断触发时，读取的 Bit 位值为 0，根据 Bit 为数据。

SD_INF 为 SD 卡插入查询位，该 bit 位不会触发中断，只能通过读取该寄存器来查询 SD 卡是否插入。

硬件连接示意图



代码实现功能

代码实现了利用板卡的中断机制，当有中断触发时，PF0 脚产生中断，然后进入中断函数，通过 INTERRUPT_DAT 寄存器查询触发中断的中断源，并打印中断源信息。

测试实验步骤

1. 将仿真器（ICE）与 ADSP-EDU-BF53x 开发板和计算机连接好。
2. 先给仿真器（ICE）上电再为 ADSP-EDU-BF53x 开发板上电。
3. 运行 VisualDSP++ 5.0 软件，选择合适的 BF533 的 session 将仿真器与软件连接。
4. 加载 VisualDSP++ 5.0 工程文件 BF53x_INTERRUPT.dpj，编译并全速运行。
5. 点击触摸屏或者按按键测试中断触发。

测试结果

点击触摸屏或者按按键测试中断触发，代码打印出触发中断的中断源。

```
Loading: "D:\file\file\CY\2011UDC\bf531\code\ADSP-EDU-
Load complete.
The interrupt is touch interrupt!

The interrupt is keyboard!

Loading: "D:\file\file\CY\2011UDC\bf531\code\ADSP-EDU-
Load complete.
The interrupt is touch interrupt!

SDCard inserted

The interrupt is keyboard!

SDCard inserted
```

未插入 SD 卡和插入 SD 卡中断信息比较。

BF53x_KEY

ADZS-EDU-BF53X 按键实验。

硬件实现原理

ADSP-EDU-BF53x 开发板上的按键连接到了 CPLD，通过 CPLD 将按键信号与 EBIU 总线和中断管脚 PF0 连接，并将中断数据地址映射在 CPLD 的当有按键按下后，会触发 PF0 中断信号，同时可以通过 EBIU 总线访问中断功能映射到 CPLD 寄存器中的 INTERRUPT_DAT 寄存器，通过访问该寄存器，可以获取键盘的中断源数据，通过读取 CPLD 的 KEYBOARD_DAT 寄存器，可以获取当前触发中断的按键的键值。

KEYBOARD_DAT 寄存器（读唯一）：

KEYBOARD_DAT 寄存器地址：0x20380000

KEYBOARD_DAT 寄存器是按键数据寄存器，通过该寄存器可以读取当前按键键值，通过键值判断当前哪个按键按下。

INTERRUPT_DAT 寄存器（读唯一）：

INTERRUPT_DAT 寄存器地址：0x20360000

INTERRUPT_DAT 寄存器是板卡上所有中断资源的中断源数据寄存器，可以通过该寄存器数据判断出当前中断是哪一个设备产生的。

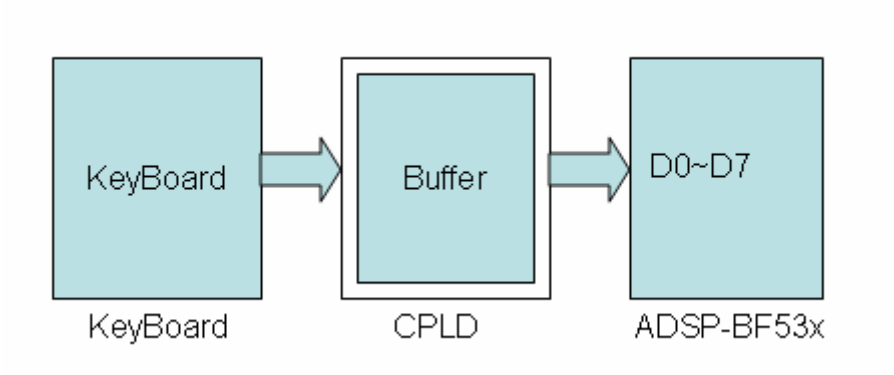
INTERRUPT_DAT 寄存器位功能：

Bit 位	7	6	5	4	3	2	1	0
功能	保留	保留	SD_INF	LAN_INT	LAN_IOWAIT	TOUCH_BUSY	TOUCH_INT	KEY_INT

当中断未触发时，读取的 Bit 位值为 1，当中断触发时，读取的 Bit 位值为 0，根据 Bit 为数据。

SD_INF 为 SD 卡插入查询位，该 bit 位不会触发中断，只能通过读取该寄存器来查询 SD 卡是否插入。

硬件连接示意图



代码实现功能

代码实现了利用查询法读取按键键值，并将读到键值与按键对应，打印出按键信息。运行代码后，代码会不停的读取键值，当读取键值与单个按键触发的键值一致时，打印出该按键信息。

测试步骤

1. 将仿真器（ICE）与 ADSP-EDU-BF53x 开发板和计算机连接好。
2. 先给仿真器（ICE）上电再为 ADSP-EDU-BF53x 开发板上电。
3. 运行 VisualDSP++ 5.0 软件，选择合适的 BF533 的 session 将仿真器与软件连接。
4. 加载 VisualDSP++ 5.0 工程文件 BF53x_KEY.dpj，编译并全速运行。
5. 按下按键。

测试结果

代码会打印出触发按键的信息。

```
Loading: "D:\file\file\CY\2011UDC\bf531\code\ADSP-EDU
Load complete.
Please press the Keyboard!
The key is right->down

The key is right->up

The key is left->right

The key is left->down

The key is left->left
```

BF53x_LAN

ADSP-EDU-BF53X 网口实验。

硬件实现原理

ADSP-EDU-BF53x 开发板上的网卡模块采用 DM9000EP 实现，DM9000EP 为 DAVICOM 公司生产的网络芯片，它集成了网卡的 MAC 和 PHY，支持 10M/100M 速度。支持 16Bit/32Bit 总线访问带宽。

ADSP-BF53x 通过 EBIU 总线采用 16Bit 总线方式与 DM9000EP 连接，其映射地址为 ADSP-BF53x 的 BANK2 地址，其映射寄存器如下：

DM9000_PPTR 寄存器（写唯一）：

DM9000_PPTRT 寄存器地址：0x20200000

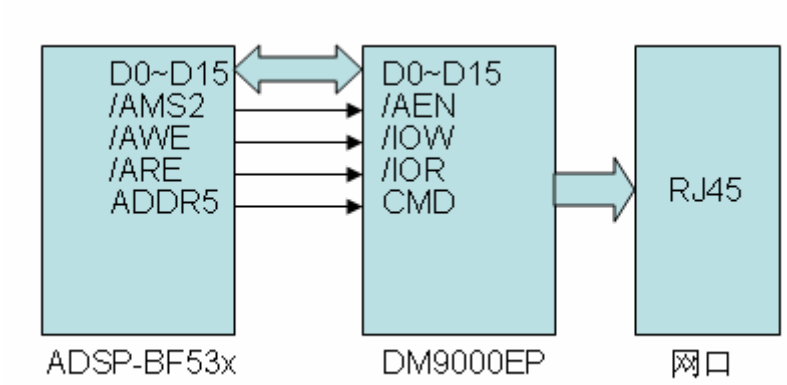
DM9000_PPTR 寄存器是 DM9000EP 的指令寄存器，用于为 DM9000EP 写入指令。

DM9000_PDATA 寄存器（读写）：

DM9000_PDATA 寄存器地址：0x20200020

DM9000_PDATA 寄存器是 DM9000EP 的数据寄存器，通过该寄存器读取和发送数据包。

硬件连接示意图



代码实现功能

代码实现了通过网络接口发送数据包的功能，没有包含任何网络协议。通过网线与计算机连接，使用计算机上的抓包工具，抓取数据包可以查看数据包的数据内容。

测试步骤

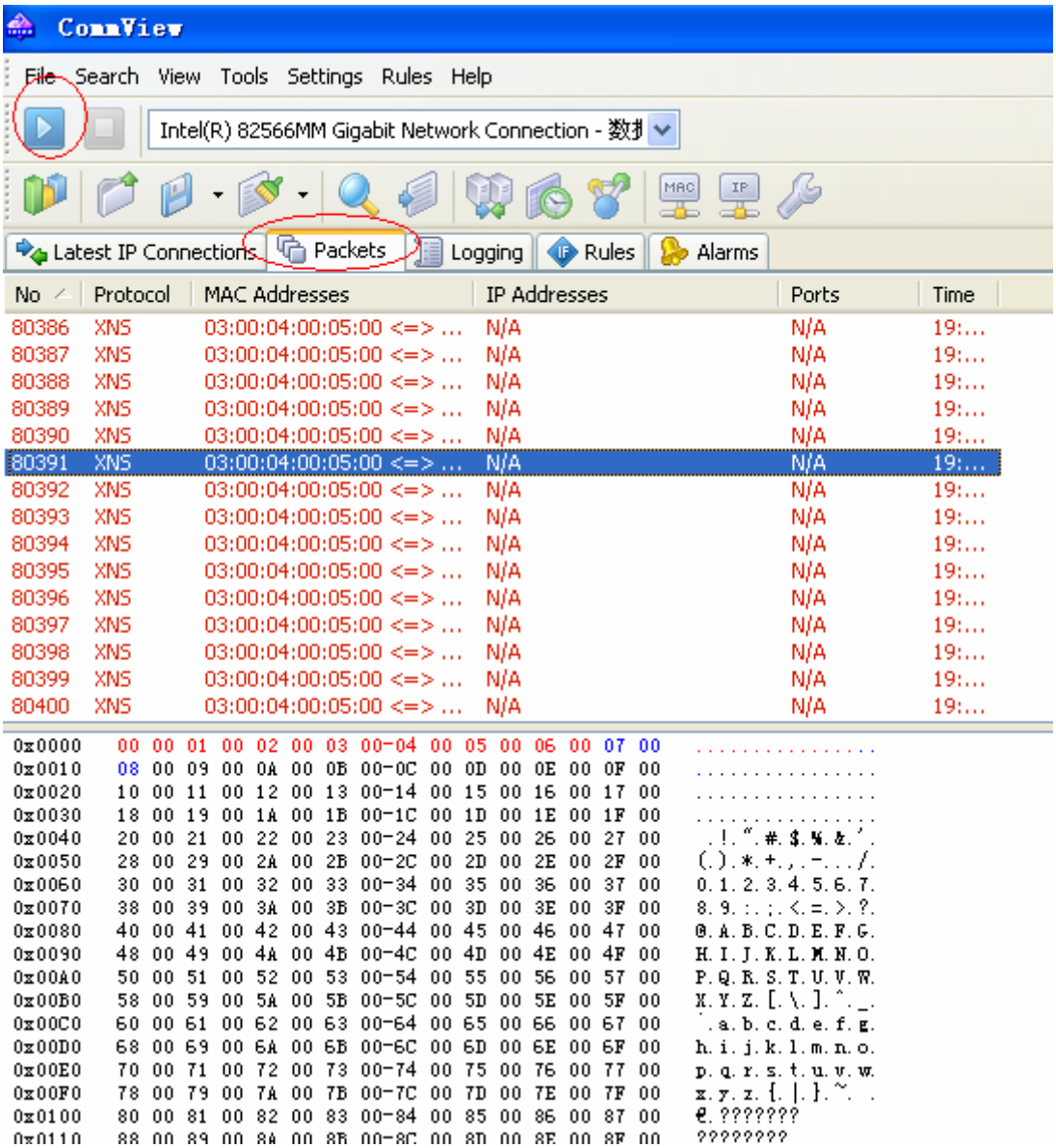
1. 将仿真器（ICE）与 ADSP-EDU-BF53x 开发板和计算机连接好。
2. 先给仿真器（ICE）上电再为 ADSP-EDU-BF53x 开发板上电。
3. 将交叉网线连接于计算机网口和开发板网口。
4. 运行 VisualDSP++ 5.0 软件，选择合适的 BF533 的 session 将仿真器与软件连接。
5. 加载 VisualDSP++ 5.0 工程文件 BF53x_LAN.dpj，编译并全速运行。
6. 运行抓包软件 Commview.exe，通过软件抓取网卡发送的数据包。

测试结果

通过抓包软件 Commview.exe 的 Pack 窗口可以看到抓取的数据包的数据信息。

```
Loading: "D:\file\file\CY\2011UDC\bf53
Load complete.
DM9000E ID is 900a 46

DM9000 work in 16 bus width
Link on ethernet at:100 Mbps
```

计算机端打印的网卡信息



使用 COMMVIEW 软件抓取的数据包信息。

BF53x_LED

ADSP-EDU-BF53X LED 灯控制实验。

硬件实现原理

ADSP-EDU-BF53x 开发板上共设计了 8 个 LED，这些 LED 直接连接于 CPLD，通过灌电流方式接入，需要点亮时，将该位数据写 0，通过配置 CPLD 映射的 DEVICE_OE 寄存器和 LED_DAT 寄存器，可以对 LED 灯进行控制。

DEVICE_OE 寄存器（写唯一）：

DEVICE_OE 寄存器地址：0x20320000

DEVICE_OE 寄存器设置硬件设备上一些控制管脚的电平状态，该寄存器只能写入数据，不能读出当前数据。

DEVICE_OE 寄存器位功能：

Bit 位	7	6	5	4	3	2	1	0
功能	保留	INTERRUPT_OE	SPI_OE	SYNCINT_OE	PTS_OE	LCD_OE	SPI_SEL	LED_OE

LED_OE：

1：使能 LED 灯模块

0：关闭 LED 灯模块

LED_DAT 寄存器（写唯一）：

LED_DAT 寄存器地址：0x20340000

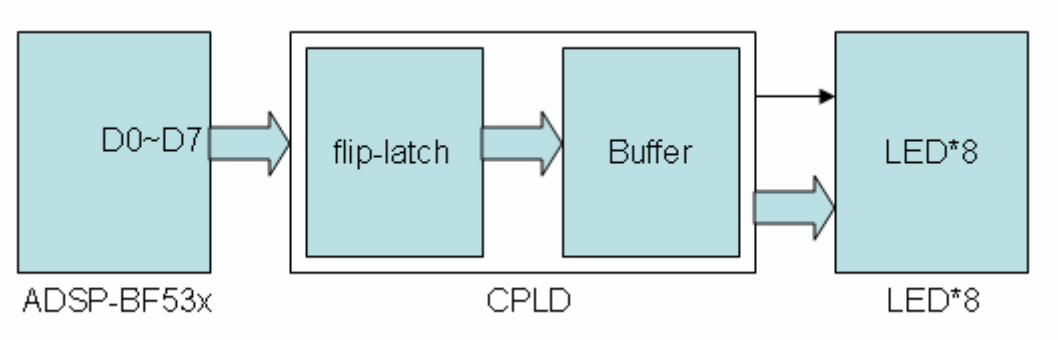
LED_DAT 寄存器是 LED 模块的数据寄存器，该寄存器的 8Bit 分别对应板卡上 8 个 LED 指示灯，通过对寄存器 Bit 位设置点亮其中一个 LED 指示灯。

LED_DAT：

1：熄灭 LED 灯

0：点亮 LED 灯

硬件连接示意图



代码实现功能

代码实现了通过逐次配置 LED_DAT 寄存器，实现了 LED 跑马灯功能。

测试步骤

1. 将仿真器（ICE）与 ADSP-EDU-BF53x 开发板和计算机连接好。
2. 先给仿真器（ICE）上电再为 ADSP-EDU-BF53x 开发板上电。
3. 运行 VisualDSP++ 5.0 软件，选择合适的 BF533 的 session 将仿真器与软件连接。
4. 加载 VisualDSP++ 5.0 工程文件 BF53x_LED.dpj，编译并全速运行。

测试结果

板卡上的 LED 逐个点亮，实现跑马灯功能。

BF53x_RS232

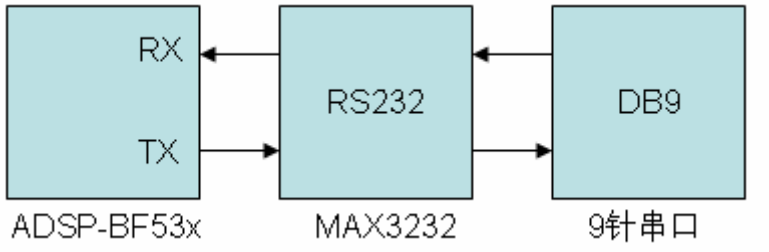
ADSP-EDU-BF53X 串口实验。

硬件实现原理

ADSP-EDU-BF53x 开发板上设计了一个 RS232 接口，该接口通过 ADSP-BF53x 上的 UART 接口，扩展 RS232 协议的芯片实现。通过串口延长线连接计算机可实现串口通讯功能。

UART 接口的通讯波特率是通过系统时钟分频实现的，系统时钟是通过输入晶振频率通过 PLL 后实现的，板卡上采用的晶振频率为 24.576MHz，采用 9600 波特率通讯时可正常打印信息，其它波特率如 115200，会出现较大的误码率，打印字符会有错误，如需使用其它波特率，需重新配置 UART 波特率的分频系数或更换为 25MHz 晶振来实现。

硬件连接示意图



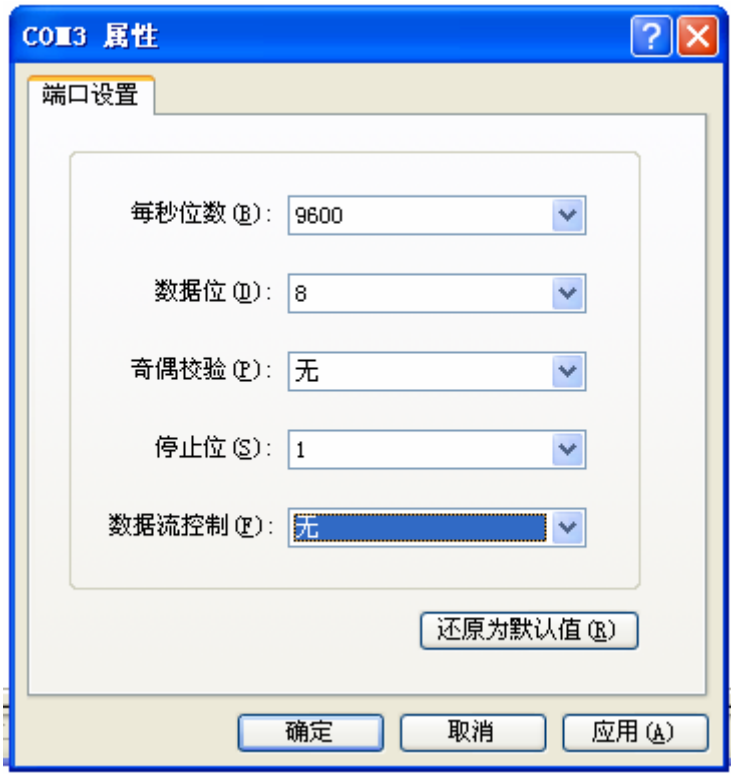
代码实现功能

代码实现了 UART 接口的自动波特率检测功能，运行 Windows 上自带的超级中断软件，其路径在：“开始”菜单 ->所有程序->附件->通讯->超级终端。将超级终端设置的波特率为 9600。编译并运行 VisualDSP++ 5.0 工程文件 BF53x_RS232.dpj 的代码，打开超级中断窗口，通过计算机键盘输入”@”，这时板卡会自动识别当前超级终端通讯的波特率，与当前超级终端建立连接，并将超级中断发送的键值返回打印在超级终端窗口上。

测试步骤

1. 将仿真器（ICE）与 ADSP-EDU-BF53x 开发板和计算机连接好。

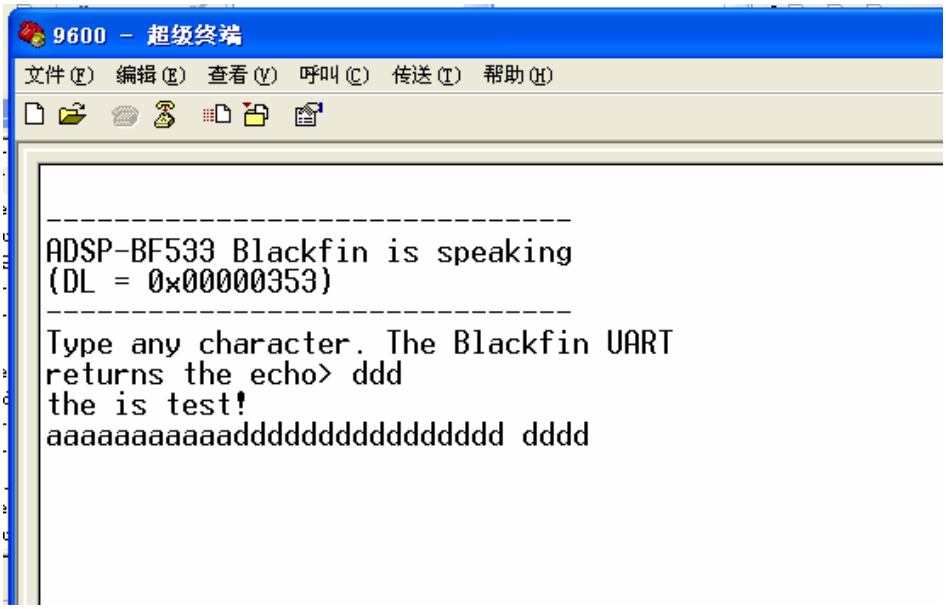
2. 先给仿真器（ICE）上电再为 ADSP-EDU-BF53x 开发板上电。
3. 运行 VisualDSP++ 5.0 软件，选择合适的 BF533 的 session 将仿真器与软件连接。
4. 加载 VisualDSP++ 5.0 工程文件 BF53x_RS232.dpj，编译并全速运行。
5. 打开 Windows 自带的超级中断软件，按下图设置，点确定后建立连接。



6. 通过计算机键盘输入”@”，这时板卡会自动识别当前超级终端通讯的波特率，与当前超级终端建立连接，并将超级中断发送的键值返回打印在超级终端窗口上。

测试结果

通过计算机键盘输入数据，超级终端窗口上打印当前按键信息。



超级终端输入“@”后，通过键盘输入的信息。

BF53x_TFT_480_272

ADSP-EDU-BF53X 液晶屏显示实验。

硬件实现原理

ADSP-EDU-BF53x 开发板上的液晶屏为 480*272 尺寸真彩 TFT 液晶屏，其型号为 WXCAT43-TG3#001R。WXCAT43-TG3#001R 为 24Bit 的液晶屏，数据输入格式为 RGB888。ADSP-BF53x 的 PPI 接口只有 16 根数据线，所以接入方式为 RGB565，将 RGB 的低位数据线直接接地，以匹配数据线的不足。

WXCAT43-TG3#001R 的背光开关由 CPLD 控制，其映射于 CPLD 的 DEVICE_OE 寄存器。其背光亮度由 Timer0 控制，通过配置 Timer0 输出 PWM 脉冲的脉宽来改变亮度。

DEVICE_OE 寄存器（写唯一）：

DEVICE_OE 寄存器地址：0x20320000

DEVICE_OE 寄存器设置硬件设备上一些控制管脚的电平状态，该寄存器只能写入数据，不能读出当前数据。

DEVICE_OE 寄存器位功能：

Bit 位	7	6	5	4	3	2	1	0
功能	保留	INTERRUPT_OE	SPI_OE	SYNCINT_OE	PTS_OE	LCD_OE	SPI_SEL	LED_OE

LCD_OE：

1：关闭 TFT 液晶屏背光

0：使能 TFT 液晶屏背光

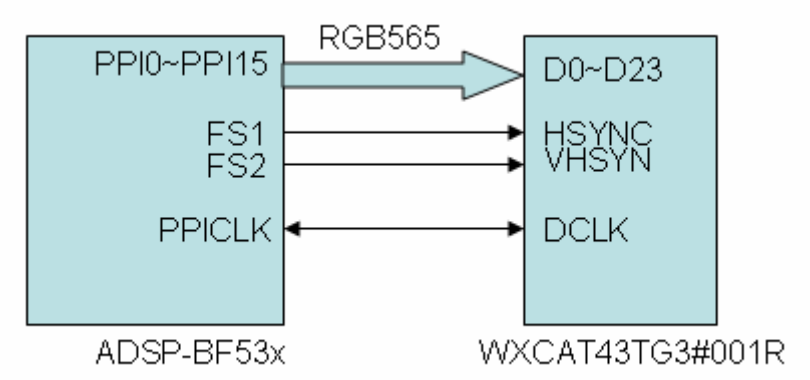
Timer0 寄存器配置：

TIMER0_CONFIG 寄存器：配置 Timer 工作模式。

TIMER0_WIDTH 寄存器：配置脉冲宽度。

TIMER0_PERIOD 寄存器：配置波形周期。;

硬件连接示意图



代码实现功能

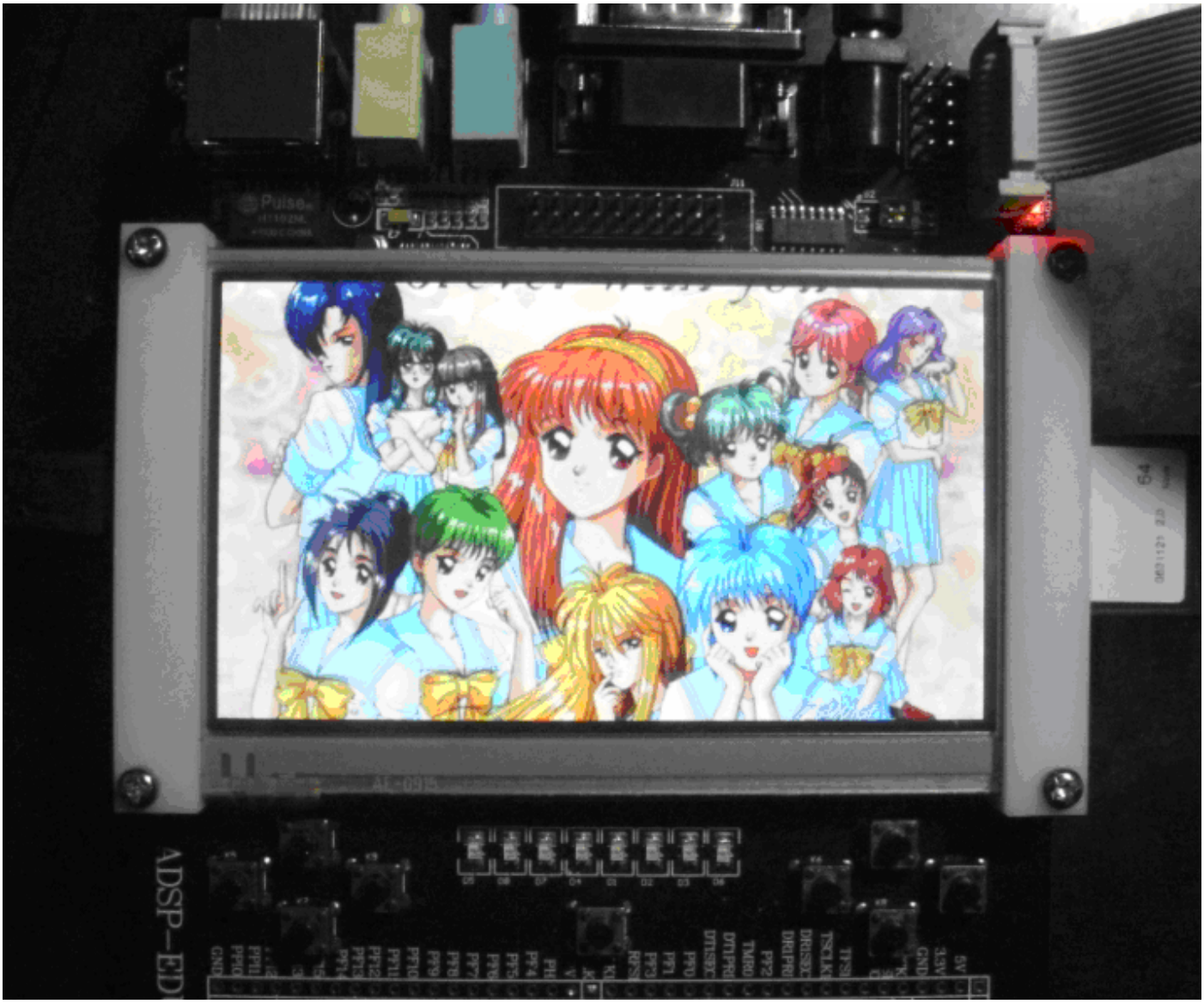
代码实现了通过文件系统读出一幅 BMP24 格式 480*272 点阵的位图图片数据，将数据做转换后变为 RGB888 格式数据，再将 RGB888 格式数据转为 RGB565 格式，通过 PPI 接口使用 PPIDMA 将数据送给液晶屏。通过 Init_Timers0h 函数可以配置背光亮度。代码中的 color_bar 函数，可以产生彩条数据，在液晶屏上产生彩条。

测试步骤

1. 将仿真器（ICE）与 ADSP-EDU-BF53x 开发板和计算机连接好。
2. 先给仿真器（ICE）上电再为 ADSP-EDU-BF53x 开发板上电。
3. 运行 VisualDSP++ 5.0 软件，选择合适的 BF533 的 session 将仿真器与软件连接。
4. 加载 VisualDSP++ 5.0 工程文件 BF53x_TFT_480_272.dpj，编译并全速运行。

测试结果

板卡上的 TFT 显示一幅卡通图像。



液晶屏上显示的 BMP24 文件的图像。

BF53x_RTC

ADSP-EDU-BF53X RTC 实验。

硬件实现原理

ADSP-BF53x 处理器上集成了一个实时时钟（RTC）模块，板卡上设计了一个专门用于 RTC 时钟源的晶体 32.768KHz，通过配置 ADSP-BF53x 处理器的 RTC 寄存器，实现时间的读取。

代码实现功能

代码实现了配置 ADSP-BF53x 处理器的 RTC 寄存器，为其设定一个初始时间，通过打印将当前的时间信息打印在软件窗口中。更新时间后，RTC 会延迟一秒钟后更新为新的时间。

测试步骤

1. 将仿真器（ICE）与 ADSP-EDU-BF53x 开发板和计算机连接好。
2. 先给仿真器（ICE）上电再为 ADSP-EDU-BF53x 开发板上电。
3. 运行 VisualDSP++ 5.0 软件，选择合适的 BF533 的 session 将仿真器与软件连接。
4. 加载 VisualDSP++ 5.0 工程文件 BF53x_RTC.dpj，编译并全速运行。

测试结果

在 VisualDSP++ 5.0 软件上看打印当前的时间。

```
Loading: "D:\file\file\CY\2011UDC\bf531\code\ADSP-EDU-BF53x_CODE1231\ADSP-EI
Load complete.
time is 10543 day 6:27:24
time is 10543 day 6:27:26
time is 10543 day 6:27:26
time is 10543 day 6:27:26
time is 0 day 12:15:31
time is 0 day 12:15:31
time is 0 day 12:15:31
time is 0 day 12:15:32
time is 0 day 12:15:32
time is 0 day 12:15:32
time is 0 day 12:15:33
time is 0 day 12:15:33
time is 0 day 12:15:34
time is 0 day 12:15:34
time is 0 day 12:15:34
time is 0 day 12:15:34
time is 0 day 12:15:35
time is 0 day 12:15:35
time is 0 day 12:15:36
time is 0 day 12:15:36
time is 0 day 12:15:36
time is 0 day 12:15:37
time is 0 day 12:15:37
time is 0 day 12:15:37
```

BF53x_SD_MMC

ADSP-EDU-BF53X SD 卡实验。

硬件实现原理

ADSP-EDU-BF53x 板卡上设计了一个 SD/MMC 接口，可以通过该接口实现对 SD/MMC 卡数据的访问。

SD 卡有两种工作模式：SDIO 模式和 SPI 模式。SDIO 模式为半字节读写模式，该模式采用四根数据线，每次可对半个字节作操作。SPI 模式有 4 根控制线：MOSI，MISO，SPICLK,SPISS.因为每次只能对一位数据作操作。SD/MMC 卡的寻址方式是按字节寻址的，为方便使用，将其寻址方式定义成连续的存储单元寻址方式。每个存储单元为 512 个字节。每个存储单元地址唯一，通过访问存储单元地址来读取数据。

ADSP-EDU-BF53x 板卡上采用 PF2 实现对 SPI 设备的控制，由于板卡上有 SD 卡和触摸屏控制器两个 SPI 设备，所以通过 CPLD 将 PF2 接口做了个 2 选 1 切换，通过配置 CPLD 的 DEVICE_OE 寄存器，来使能和选通 PF2 连接哪一个 SPI 设备。

SD 卡插入后，不会触发中断信号，但可以通过中断数据寄存器读取其插入信息。

DEVICE_OE 寄存器（写唯一）：

DEVICE_OE 寄存器地址：0x20320000

DEVICE_OE 寄存器设置硬件设备上一些控制管脚的电平状态，该寄存器只能写入数据，不能读出当前数据。

DEVICE_OE 寄存器位功能：

Bit 位	7	6	5	4	3	2	1	0
功能	保留	INTERRUPT_OE	SPI_OE	SYNCINT_OE	PTS_OE	LCD_OE	SPI_SEL	LED_OE

SPI_SEL：

1：SPI_SEL2 选通触摸屏模块

0：SPI_SEL2 选通 SD 卡模块

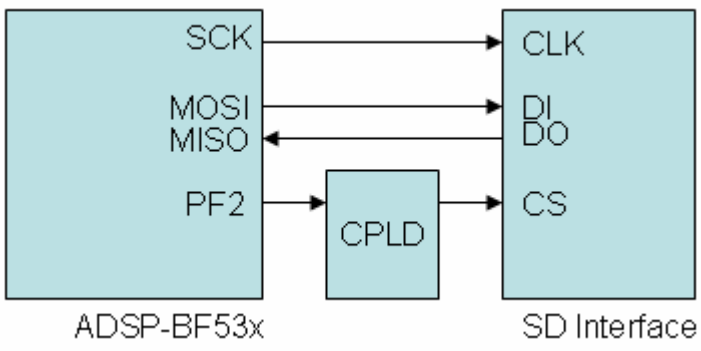
SPI_OE:

1：关闭 SPISEL 选通开关

0：使能 SPISEL 选通开关

在使用 SD/MMC 卡接口时时，需将 SPI_OE 位设置为 0，将 SPI_SEL 位设置为 0。

硬件连接示意图



代码实现功能

代码实现了对 SD/MMC 卡扇区的写入和读取，并将读出的数据与写入的数据做比较，并打印比较结果。

测试步骤

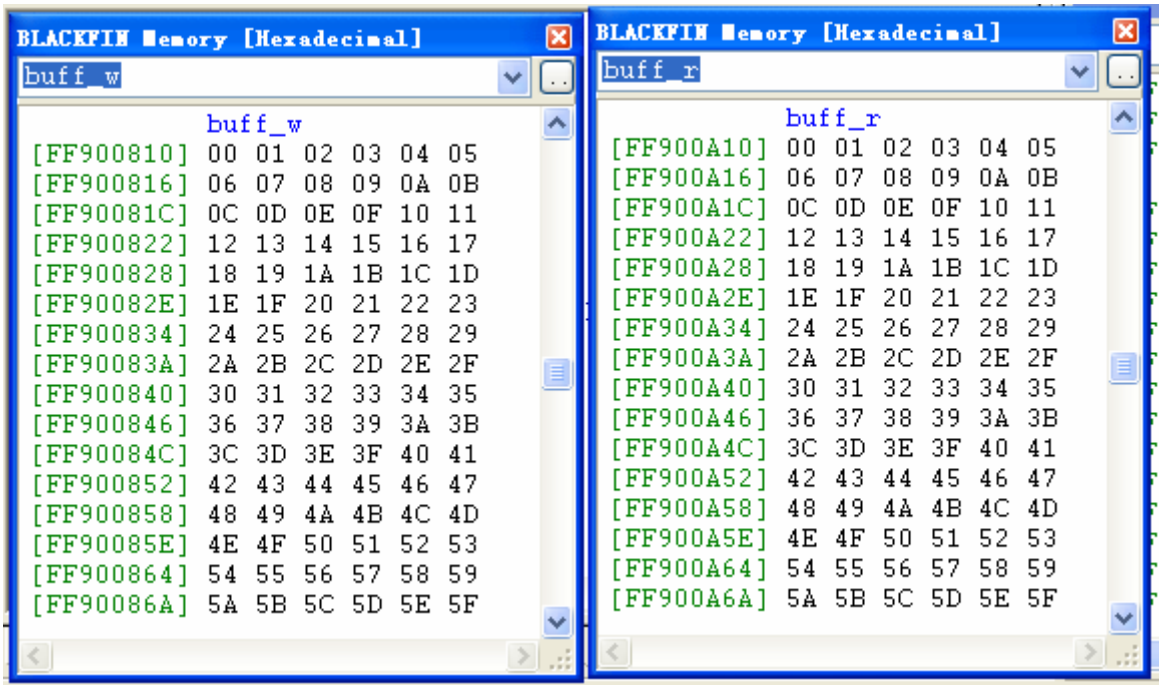
1. 将仿真器（ICE）与 ADSP-EDU-BF53x 开发板和计算机连接好。
2. 将 SD 或 MMC 卡插入 SD 卡接口。
3. 先给仿真器（ICE）上电再为 ADSP-EDU-BF53x 开发板上电。
4. 运行 VisualDSP++ 5.0 软件，选择合适的 BF533 的 session 将仿真器与软件连接。
5. 加载 VisualDSP++ 5.0 工程文件 BF53x_SD_MMC.dpj，编译并全速运行。

测试结果

在 VisualDSP++ 5.0 软件上打印校验数据的结果。

```
Loading: "D:\file\file\CY\2011UDC\bf531\cc
Load complete.
SD/MMC_INIT is OK!
SD/MMC TEST OK!
SD/MMC Check END!
```

SD 卡校验完成



写入的数据和读出的数据内存比较。

BF53x_SDRAM

ADSP-EDU-BF53X SDRAM 模块实验。

硬件实现原理

ADSP-EDU-BF53x 板卡上采用的 SDRAM 型号为 MT48LC16M16A2,容量为 32Mbyte，采用 16Bit 模式连接 ADSP-BF53x。通过配置 EBIU 的 SDRAM 控制寄存器对其进行初始化。

代码实现功能

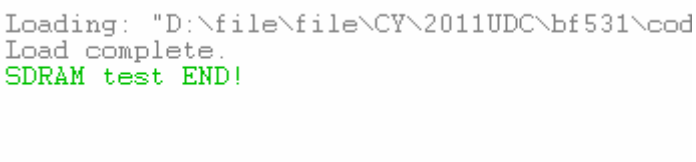
代码实现了对 SDRAM 数据进行写入和读出遍历操作，并将读出的数据与写入的数据做比较，判断内存是否正常，可用。

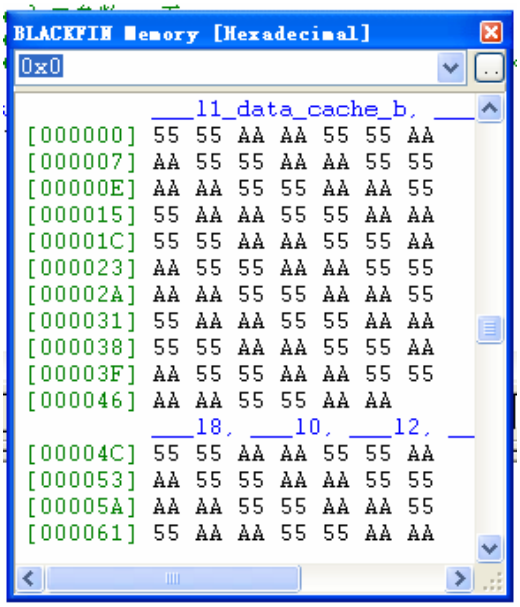
测试步骤

1. 将仿真器（ICE）与 ADSP-EDU-BF53x 开发板和计算机连接好。
2. 将 SD 或 MMC 卡插入 SD 卡接口。
3. 先给仿真器（ICE）上电再为 ADSP-EDU-BF53x 开发板上电。
4. 运行 VisualDSP++ 5.0 软件，选择合适的 BF533 的 session 将仿真器与软件连接。
5. 加载 VisualDSP++ 5.0 工程文件 BF53x_SDRAM.dpj，编译并全速运行。

测试结果

如内存出错，在 VisualDSP++ 5.0 软件上打印出错数据与元数据值，如数据没有出错，完成读写遍历后打“SDRAM test END!”





SDRAM 中的测试数据。

BF53x_TOUCH

ADSP-EDU-BF53X 触摸屏实验。

硬件实现原理

ADSP-EDU-BF53x 板卡的 TFT 液晶屏上，覆盖了一层触摸屏(Touch)，触摸屏连接触摸屏控制器芯片 ADS7843, ADSP-BF53x 通过 SPI 接口连接该控制器，采用 PF2 实现控制器设备的控制，由于板卡上有 SD 卡和触摸屏控制器两个 SPI 设备，所以通过 CPLD 将 PF2 接口做了个 2 选 1 切换，通过配置 CPLD 的 DEVICE_OE 寄存器，来使能和选通 PF2 连接哪一个 SPI 设备。

DEVICE_OE 寄存器（写唯一）：

DEVICE_OE 寄存器地址：0x20320000

DEVICE_OE 寄存器设置硬件设备上一些控制管脚的电平状态，该寄存器只能写入数据，不能读出当前数据。

DEVICE_OE 寄存器位功能：

Bit 位	7	6	5	4	3	2	1	0
功能	保留	INTERRUPT_OE	SPI_OE	SYNCINT_OE	PTS_OE	LCD_OE	SPI_SEL	LED_OE

SPI_SEL：

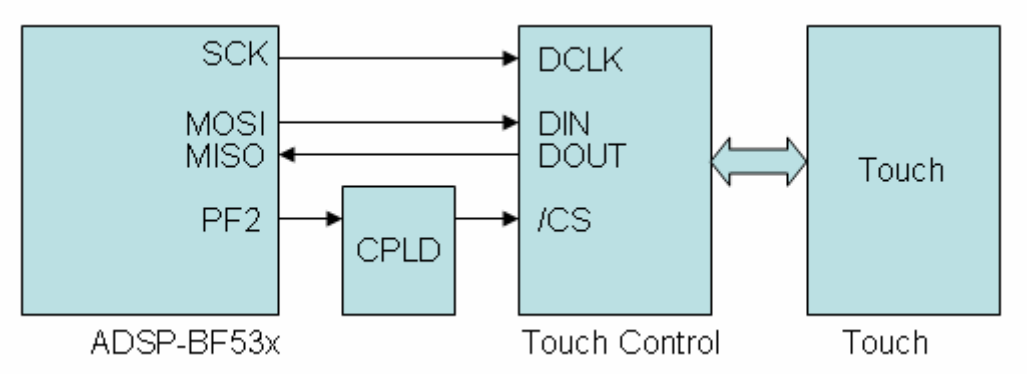
1：SPI_SEL2 选通触摸屏模块

0：SPI_SEL2 选通 SD 卡模块

SPI_OE:

1：关闭 SPISEL 选通开关
0：使能 SPISEL 选通开关
在使用触摸屏时，需将 SPI_OE 位设置为 0，将 SPI_SEL 位设置为 1。

硬件连接示意图



代码实现功能

代码实现了 12Bit 的触摸屏控制器对触摸屏坐标的读取，代码采用中断触发方式，当中断触发后，读出中断数据寄存器，判断确认为触摸屏中断，然后调用触摸屏读坐标函数分别读取 X 坐标和 Y 坐标。并将坐标打印在 VisualDSP++ 5.0 软件上。

测试步骤

1. 将仿真器（ICE）与 ADSP-EDU-BF53x 开发板和计算机连接好。
2. 先给仿真器（ICE）上电再为 ADSP-EDU-BF53x 开发板上电。
3. 运行 VisualDSP++ 5.0 软件，选择合适的 BF533 的 session 将仿真器与软件连接。
4. 加载 VisualDSP++ 5.0 工程文件 BF53x_TOUCH，编译并全速运行。

测试结果

用手点击触摸屏，在 VisualDSP++ 5.0 软件上打印被触发的触摸屏坐标。

```
Loading: "D:\file\file\CY\2011UDC\bf531\code\ADSP-EDU-BF5
Load complete.
Please press the touch screen!
x is 9f0
y is 508
x is 738
y is a04
x is e2b
y is 210
x is d38
y is ee4
x is 1b9
y is 1ac
x is d31
y is cf8
x is da7
y is 4ab
```

BF53x_USB

ADSP-EDU-BF53X USB 模块实验。

硬件实现原理

ADSP-EDU-BF53x 板卡上设计了一个 USB1.1 标准的 USB 设备芯片，其型号为 PDIUSBD12,该芯片有 6 个数据传输断点，其控制命令简单，寄存器少，适合于广大初学者学习 USB 协议。ADSP-BF53x 处理器通过 EBIU 接口的 BANK3 连接 PDIUSBD12 芯片，通过 CPLD 为其分配访问地址，通过 CPLD 的 USB_DAT 寄存器和 USB_CMD 寄存器进行访问。

USB_DAT 寄存器（读写）：

USB_DAT 寄存器地址：0x20300000

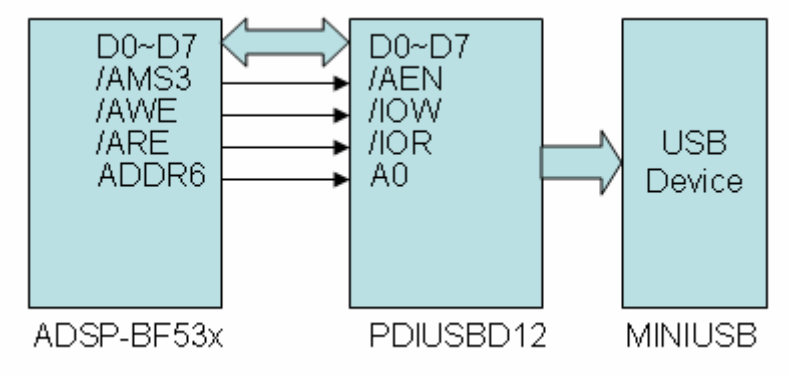
USB_DAT 寄存器：该寄存器用于发送和读出 PDIUSBD12 端点的数据。

USB_CMD 寄存器（写唯一）：

USB_CMD 寄存器地址：0x20300040

USB_CMD 寄存器：该寄存器用于发送 PDIUSBD12 的指令。

硬件连接示意图



代码实现功能

代码实现了 USB 设备的枚举和大容量存储器盘符出现的功能。运行代码后，将 MINIUSB 连接线连接板卡的 MINIUSB 接口，这时计算机端会出现有 USB 设备接入，板卡上的 USB GL 指示灯会不停的闪烁，待枚举完成后，在计算机端会出现一个磁盘符号。由于代码还没有做完善，我们不能对该磁盘符号做任何的操作。
将串口设置波特率 9600，打开超级终端，可以看到 USB 与计算机通讯的数据信息。也可使用 USB 抓包工具抓举更详细的数据信息。

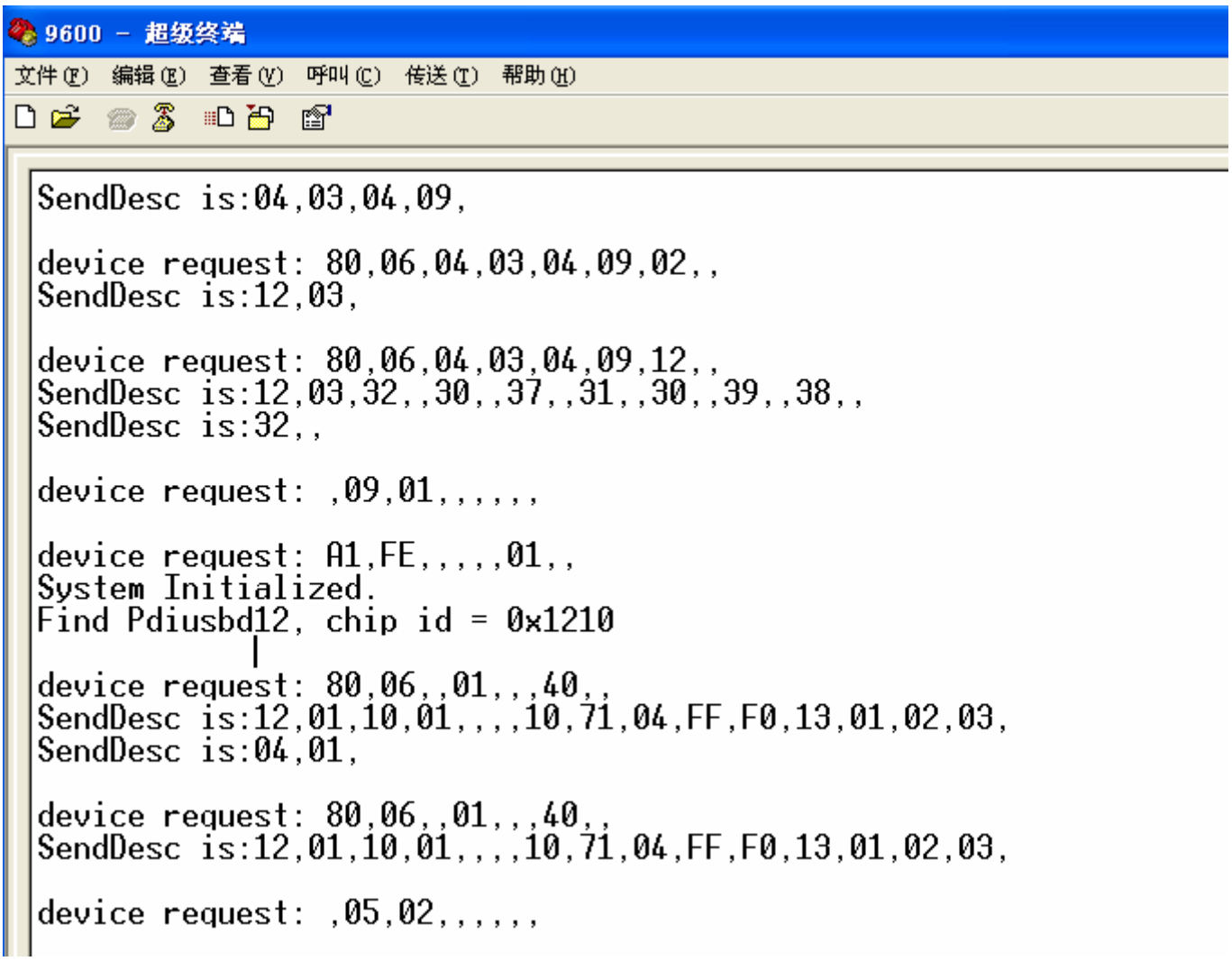
注意：必须先运行代码，然后再将 USB 线接入计算机，否则不会产生硬件响应。

测试步骤

1. 将仿真器（ICE）与 ADSP-EDU-BF53x 开发板和计算机连接好。
2. 先给仿真器（ICE）上电再为 ADSP-EDU-BF53x 开发板上电。
3. 运行 VisualDSP++ 5.0 软件，选择合适的 BF533 的 session 将仿真器与软件连接。
4. 加载 VisualDSP++ 5.0 工程文件 BF53x_USB，编译并全速运行。
5. 采用 MINIUSB 线连接计算机和板卡。

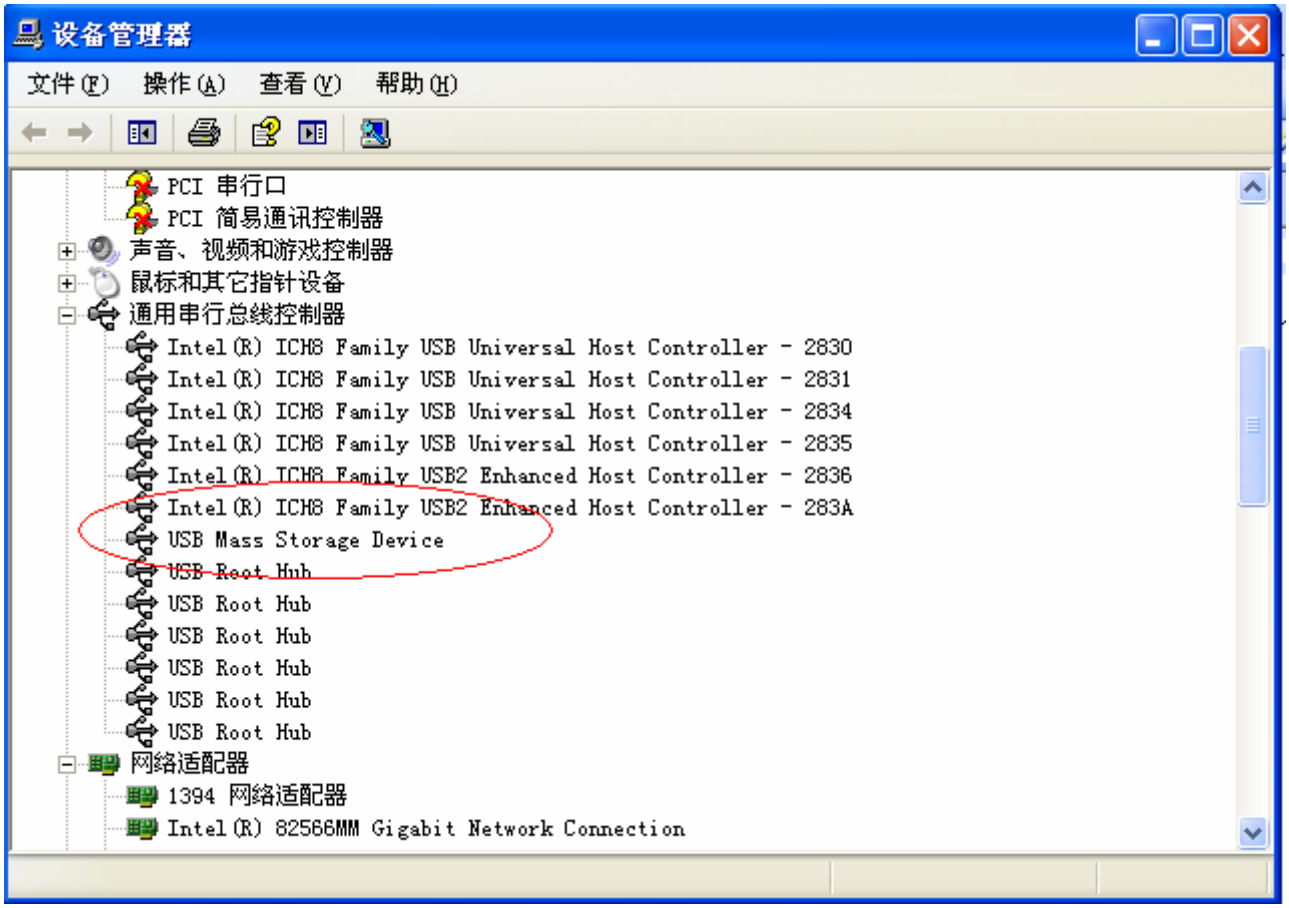
测试结果

在计算机端能看到有 USB 设备接入的提示，等待片刻后能在计算机端看到有 USB 设备接入。

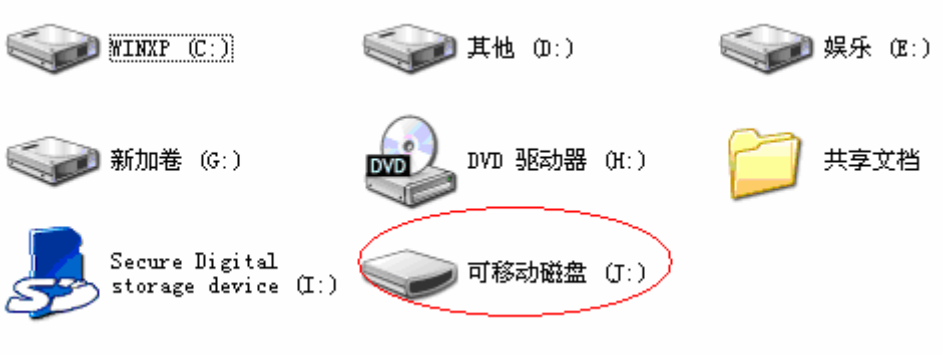


超级终端上打印的 USB 数据包信息

计算机端发现 USB 设备接入，识别出是一个磁盘驱动器。



设备管理器中大容量存储器被识别。



计算机端显示盘符。

BF533EzFlashDriver

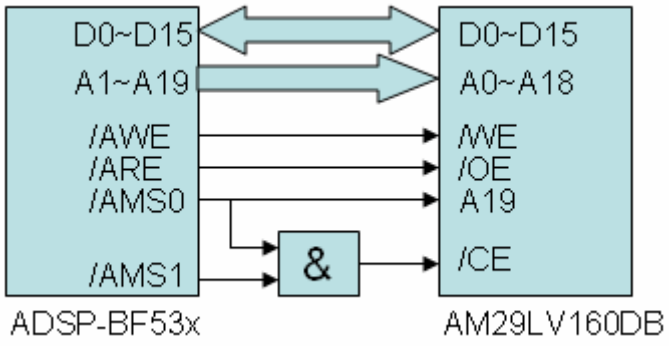
ADSP-EDU-BF53X NOR FLASH 驱动实验。

硬件实现原理

ADSP-EDU-BF53x 开发板上的 NorFlash 采用的是 AM29LV160DB。 NorFlash 主要用于存储 ADSP-BF53x 启动的代码，做好软件后可以将生成的 LDR 文件通过 VisualDSP++5.0 软件写入 NorFlash，让代码在板卡上运行。

ADSP-BF53x 共有 4 个异步 BANK，每个 BANK 共 1MB 空间，AM29LV160DB 是 2MB 的 Flash，所以硬件设计上，采用两个 BANK 连接该 FLASH，将其映射于 ADSP-BF53x 的 BANK0 和 BANK1。

硬件连接示意图



代码实现功能

代码实现了 NorFlash 在 VisualDSP++5.0 软件下的驱动，运行软件会生成能在 VisualDSP++5.0 软件的 FLASH 烧写工具中直接挂载使用的.Dxe 文件。该工程下已经提供了一个 BF533EzFlashDriver.dxe，使用这个.Dxe 文件直接

可以挂载在 VisualDSP++5.0 软件的 FLASH 烧写工具中使用，无需重新生成。

测试步骤

参考“Flash 编程”章节。

测试结果

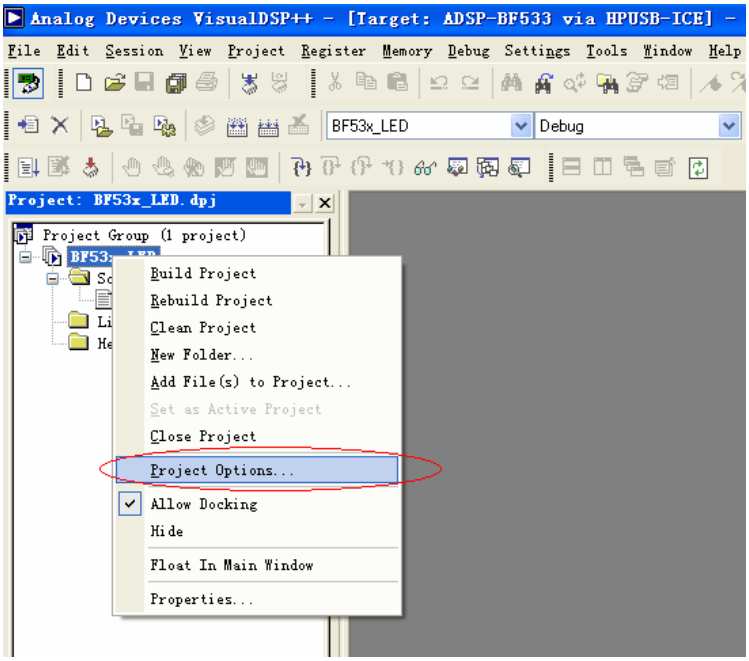
参考“Flash 编程”章节。

Blackfin FLASH 烧写说明

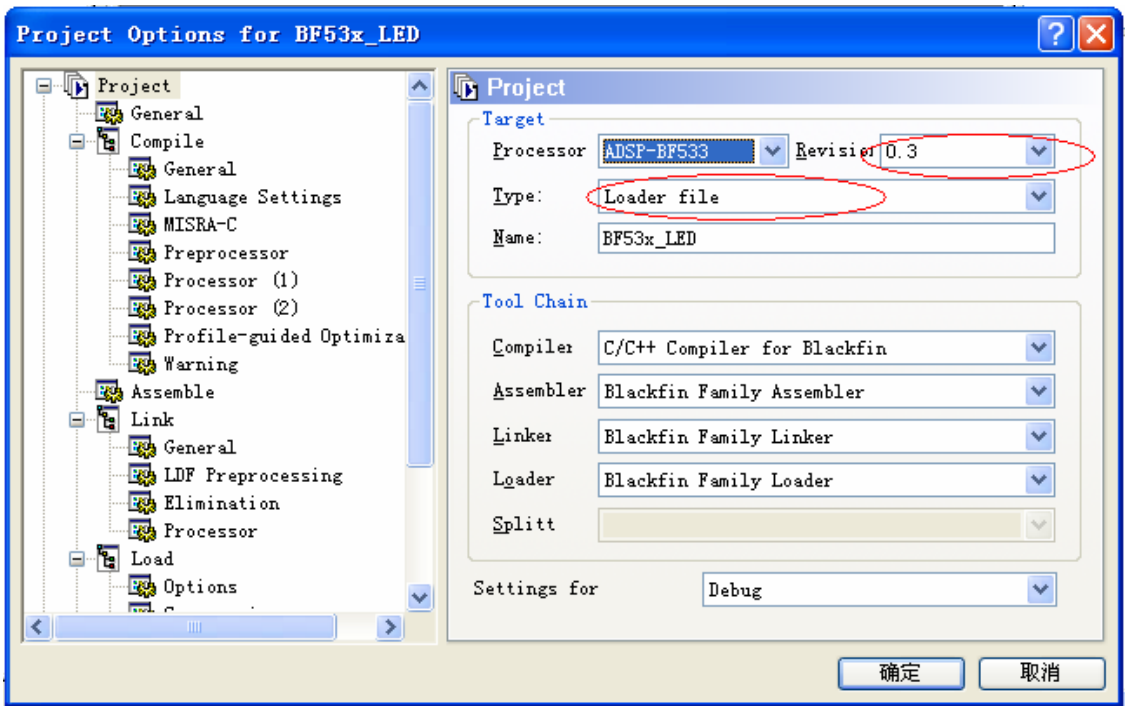
FLASH 烧写文件的生成

Blackfin 系列处理器的烧写文件尾缀是“.ldr”，这个文件通过可以通过代码工程生成，以 BF53x_LED 工程为例。将 Visual DSP++5.0 软件连接 ADSP-EDU-BF53x 开发板，通过 Visual DSP++5.0 软件下“File->open->Project..”选项将 BF53x_LED.dpj 工程载入 Visual DSP++5.0 软件。

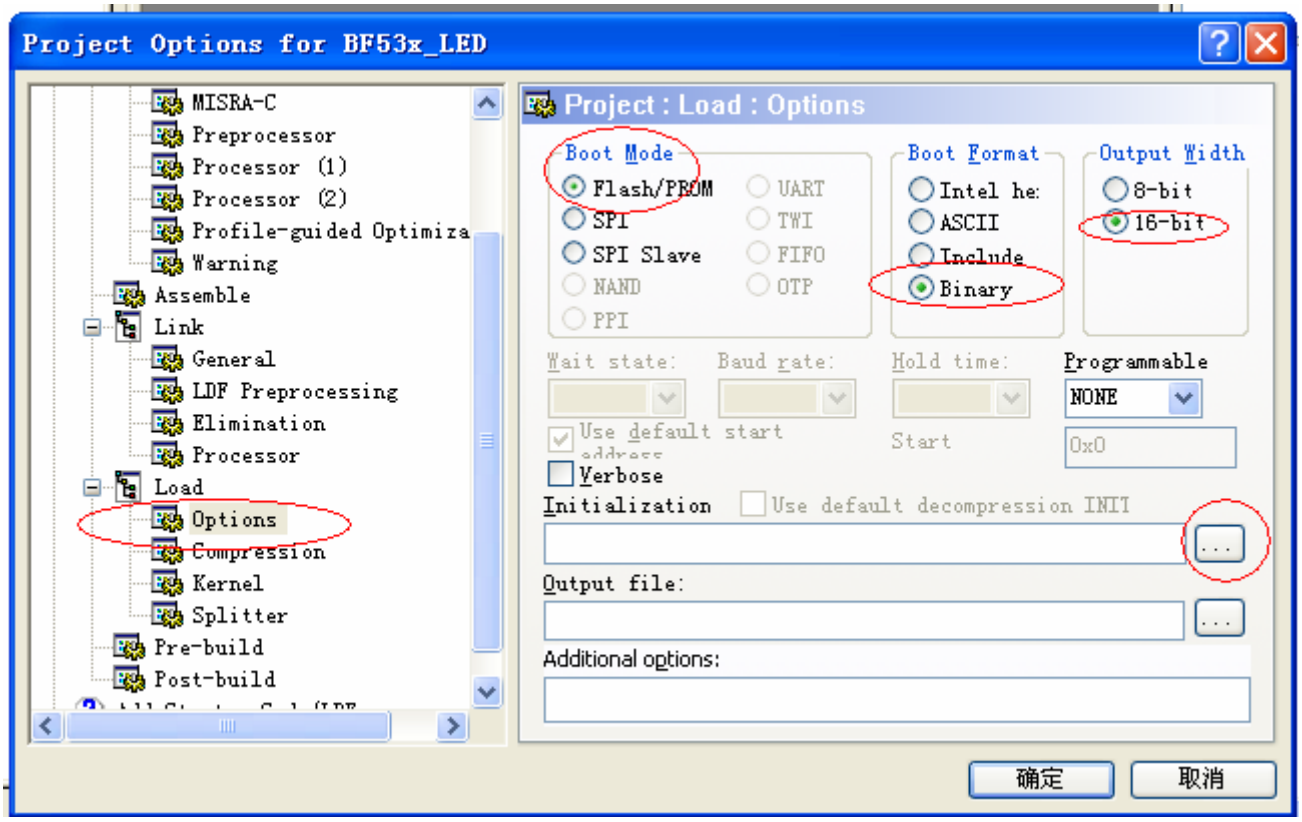
在工程名上按鼠标右键，选择“Project Options..”，



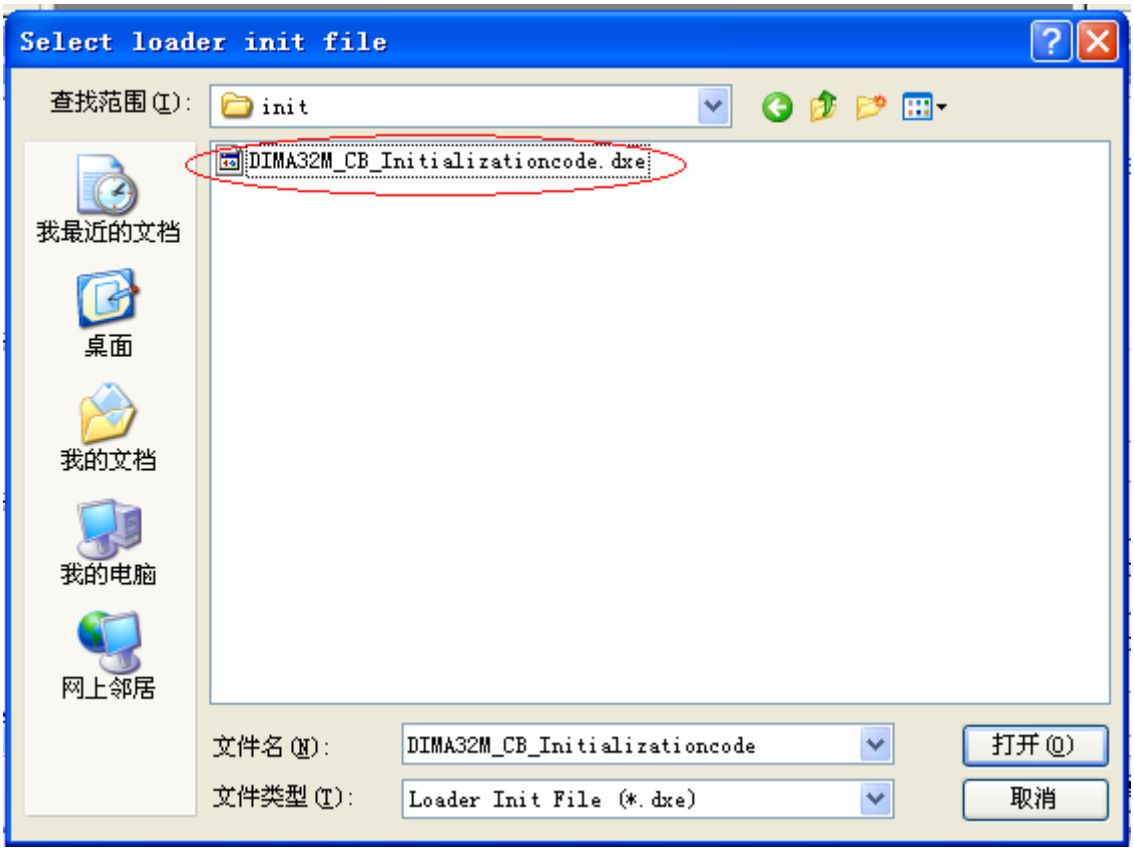
根据芯片的实际版本，为工程选择一个芯片版本，将“Type”选为“Loader File”。



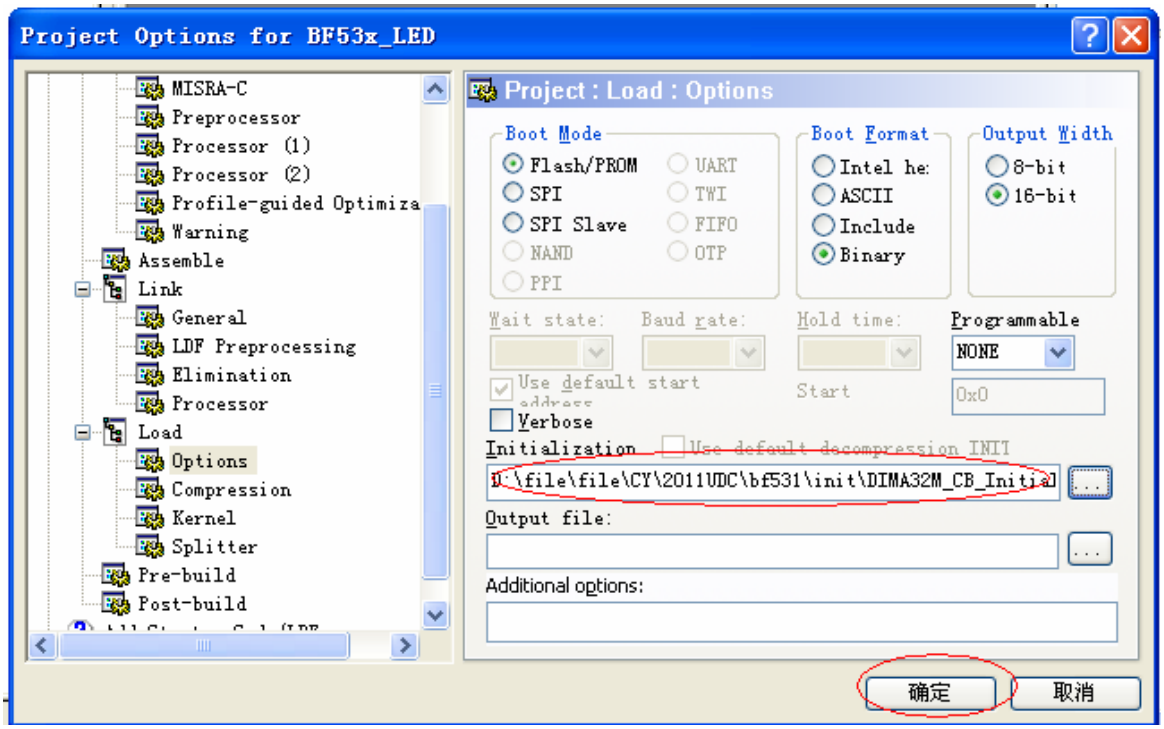
按下图为生成的 LDR 文件选择格式。最后要为生成的文件加上 initialization 文件，这个文件主要是初始化板卡上的 SDRAM，板卡上电后会最先执行该文件，不加载此文件板卡将不能正常运行烧写的程序。点击 initialization 选项后面的选择按钮。



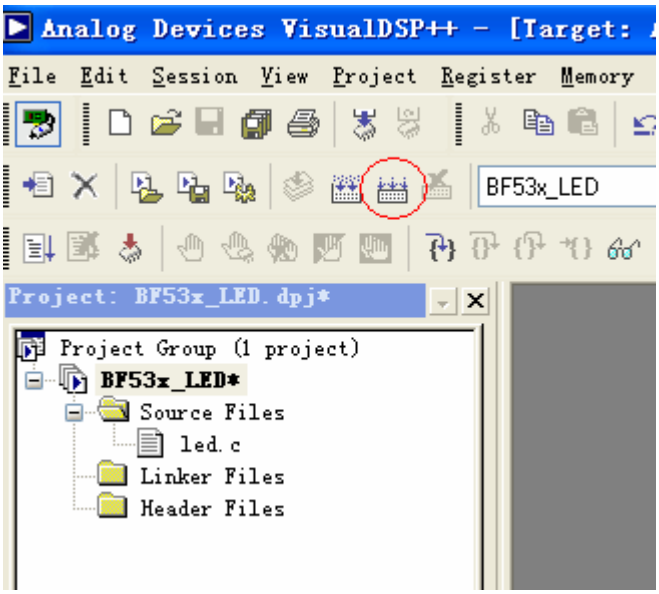
找到“DIMA32M_CB_Initializationcode.dxe”文件。



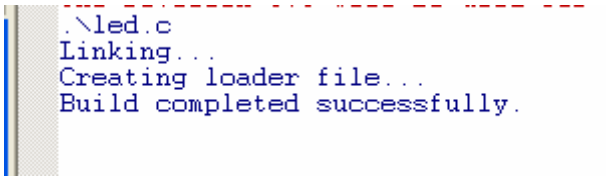
加载完文件后点“确定”



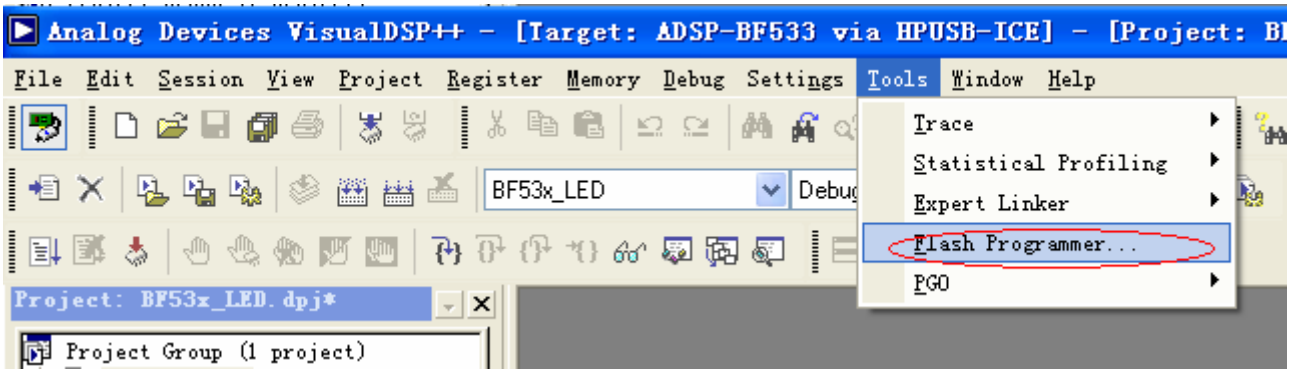
选择“ReBuild All“按钮全编译工程。



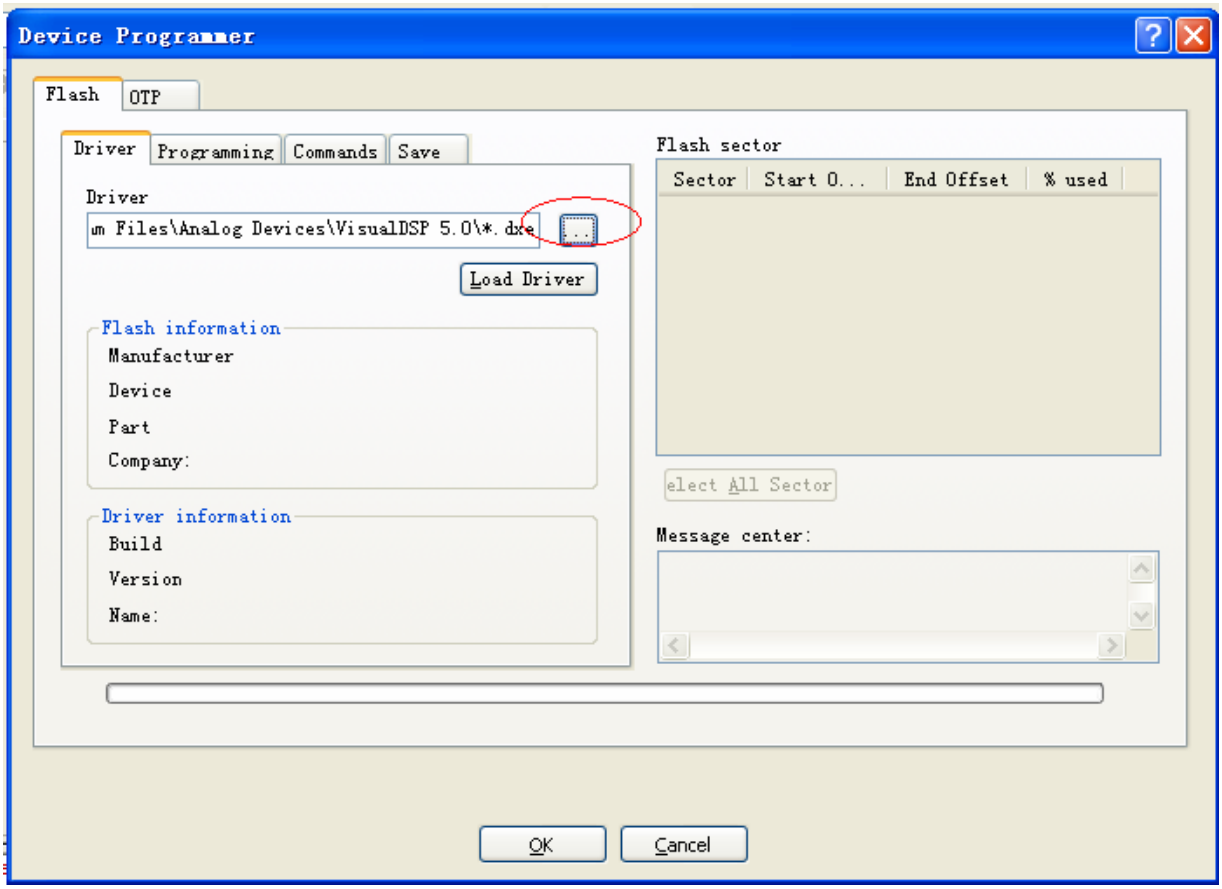
编译完成后，会看到生成文件提示。该文件默认生成地址为当前工程的 Debug 文件夹下。



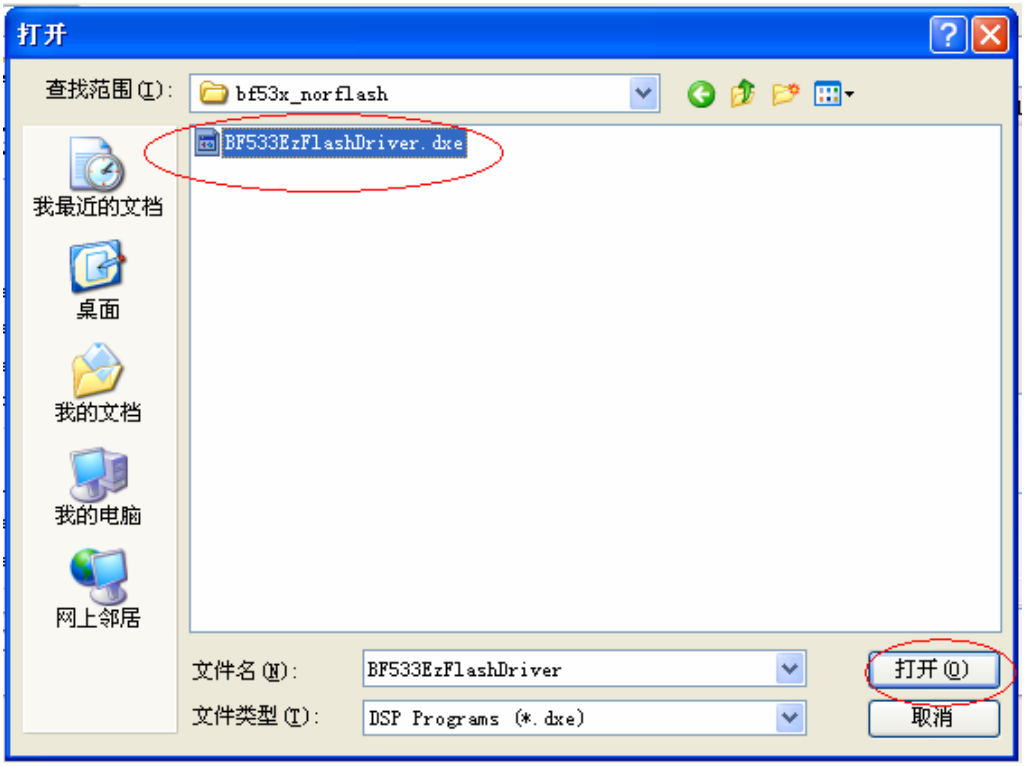
FLASH 编程



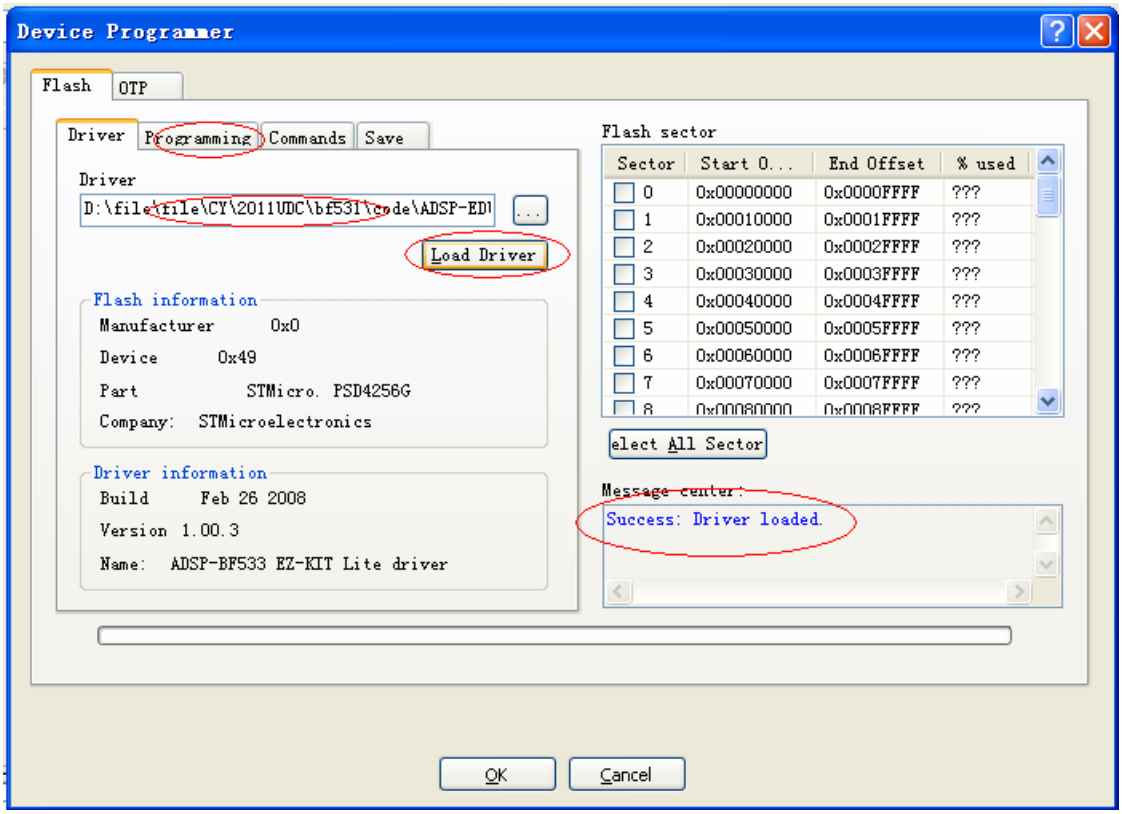
为 Flash 加载一个驱动文件，这个文件在例子代码下“bf53x_norflash”文件夹下。



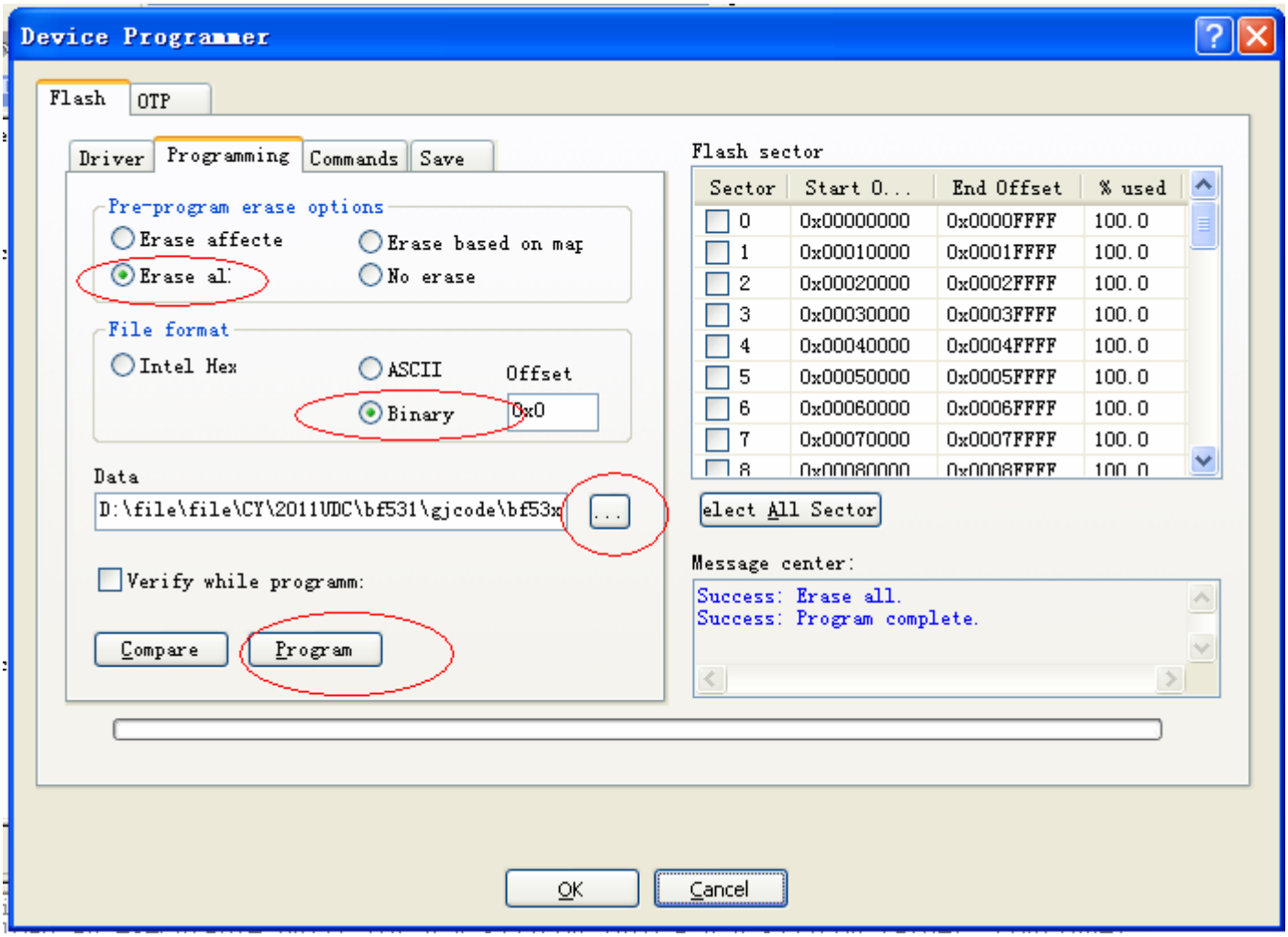
找到“BF533EzFlashDriver.dxe”文件



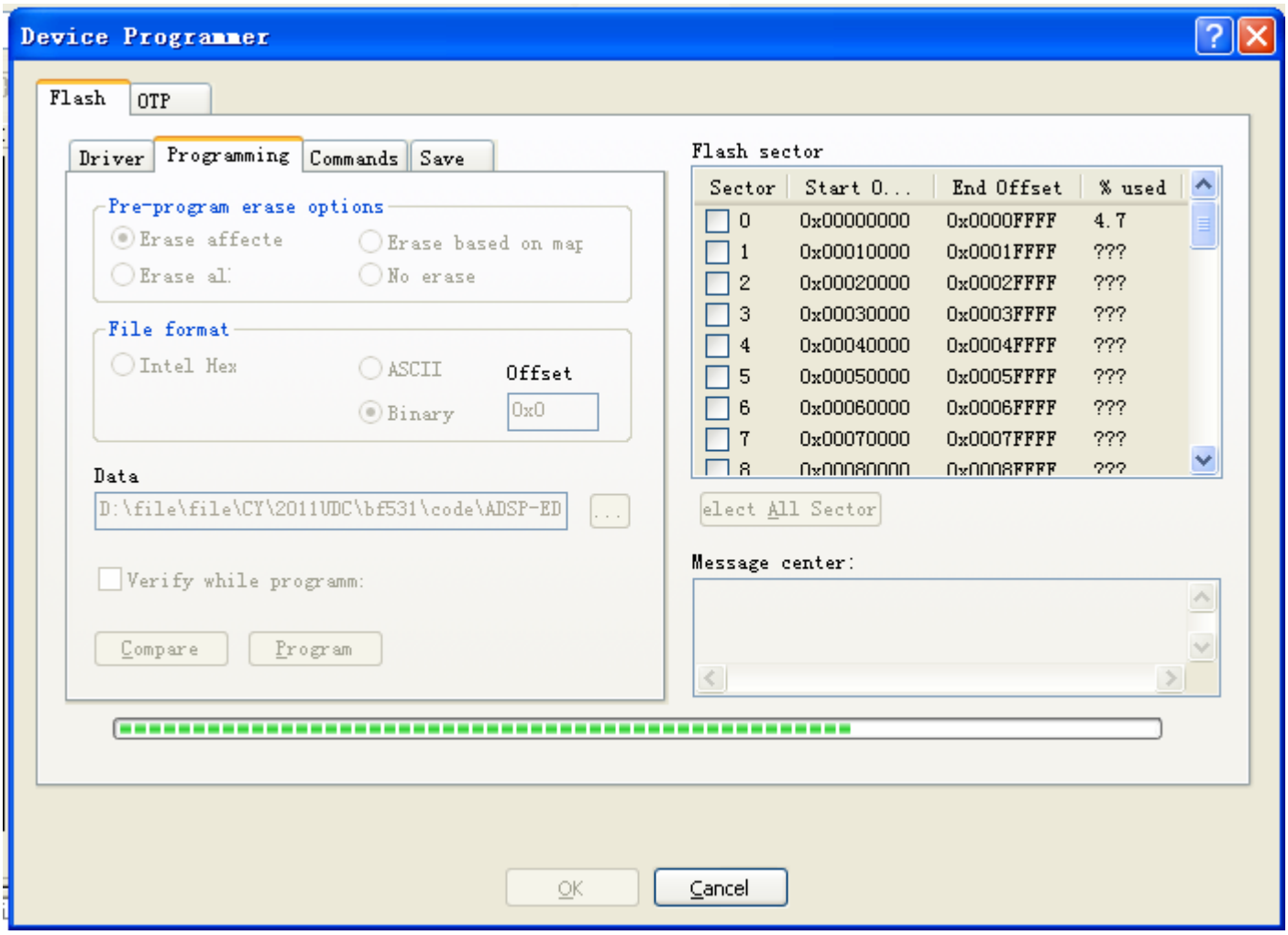
选定驱动文件后，点“Load Driver”，成功加载后，能看到右边窗口提示“Success:Dricer loaded”，然后点“Programming”页面。



按下图选择选项，然后点 data 后面的按钮，找到 BF53x_LED 工程下 Debug 文件夹下刚才生成的“BF53x_LED.ldr”文件，加载后，点“Program”按钮。



点“Program”后，软件会将 FLASH 整个擦除，然后再将下载代码写入，擦除 FLASH 时需耐心等待。



完后编程后，点“OK”按钮，退出编程。

关闭 Visual DSP++ 5.0 软件，为板卡断电。

将板卡上的拨码开关 S2 拨至“1-ON，2-OFF”，为板卡重新上电，观察 LED 灯。

CPLD 资源分配表

CPLD 控制着 ADSP-EDU-BF53x 开发板上多个模块的资源，该资源分配如下：

总线地址：

资源	地址	读写状态	说明
USB_DAT	0x20300000	读写	USB 芯片数据地址
USB_CMD	0x20300040	写唯一	USB 芯片指令地址
DEVICE_OE	0x20320000	写唯一	设备使能地址
LED_DAT	0x20340000	写唯一	LED 指示灯数据地址
INTERRUPT_DAT	0x20360000	读唯一	中断源数据寄存器

KEYBOARD_DAT	0x20380000	读唯一	按键数据寄存器

USB_DAT 寄存器（读写）：
USB_DAT 寄存器是 USB 模块的数据寄存器，通过该寄存器对 USB 芯片进行数据读写。

USB_CMD 寄存器（写唯一）：
USB_CMD 寄存器 USB 模块的指令寄存器，通过该寄存器对 USB 芯片进行指令写入。

DEVICE_OE 寄存器（写唯一）：
DEVICE_OE 寄存器设置硬件设备上一些控制管脚的电平状态，该寄存器只能写入数据，不能读出当前数据。
DEVICE_OE 寄存器位功能：

Bit 位	7	6	5	4	3	2	1	0
功能	保留	INTERRUPT_OE	SPI_OE	SYNCINT_OE	PTS_OE	LCD_OE	SPI_SEL	LED_OE

LED_OE：
1：使能 LED 灯模块
0：关闭 LED 灯模块
SPI_SEL：
1：SPI_SEL2 选通触摸屏模块
0：SPI_SEL2 选通 SD 卡模块
LCD_OE：
1：关闭 TFT 液晶屏背光
0：使能 TFT 液晶屏背光
PTS_OE：
1：功能保留
0：功能保留
SYNCINT_OE：
1：功能保留
0：功能保留
SPI_OE:
1：关闭 SPISEL 选通开关
0：使能 SPISEL 选通开关
INTERRUPT_OE：
1：关闭 I2C_SCL 输入信号，使能 PF0 中断信号
0：使能 I2C_SCL 输入信号，关闭 PF0 中断信号

LED_DAT 寄存器（写唯一）：

LED_DAT 寄存器是 LED 模块的数据寄存器，该寄存器的 8Bit 分别对应板卡上 8 个 LED 指示灯，通过对寄存器 Bit 位设置点亮其中一个 LED 指示灯。
LED_DAT：

1：熄灭 LED 灯
0：点亮 LED 灯

INTERRUPT_DAT 寄存器（读唯一）：
INTERRUPT_DAT 寄存器是板卡上所有中断资源的中断源数据寄存器，可以通过该寄存器数据判断出当前中断是哪一个设备产生的。
INTERRUPT_DAT 寄存器位功能：

Bit 位	7	6	5	4	3	2	1	0
功能	保留	保留	SD_INF	LAN_INT	LAN_IOWAIT	TOUCH_BUSY	TOUCH_INT	KEY_INT

当中断未触发时，读取的 Bit 位值为 1，当中断触发时，读取的 Bit 位值为 0，根据 Bit 为数据。
SD_INF 为 SD 卡插入查询位，该 bit 位不会触发中断，只能通过读取该寄存器来查询 SD 卡是否插入。

KEYBOARD_DAT 寄存器（读唯一）：
KEYBOARD_DAT 寄存器是按键数据寄存器，通过该寄存器可以读取当前按键键值，通过键值判断当前哪个按键按下。

CPLD 寄存器赋值：
CPLD 寄存器，除了 USB_DAT 寄存器可以读写外，其它寄存器都是单项操作，所以在配置这些寄存器时，只能采用一次性赋值，不能采取回读赋值的方法。如需回读，可以采用临时变量作为回读的介体，进行赋值操作。
正确赋值方法：

```
*pDEVICE_OE = 0x24;
```

或

```
Unsigned char temp;
temp = 0x24;
temp |= 0x01 ;
*pDEVICE_OE = temp;
```

错误的赋值方法：

```
*pDEVICE_OE = 0x24;
*pDEVICE_OE &= ~0x4;
```

具体的操作方法可以参考每一个模块的驱动代码。

高级实验代码说明

部分代码会使用.xml 文件进行初始化，为保证所有高级代码都能正常运行，请将光盘提供的 ADSP-BF533-proc.xml 与 Visual DSP ++软件安装路径下：Analog Devices\VisualDSP 5.0 \System\ArchDef\ADSP-BF533-proc.xml 的同名文件进行替换。

BF53x_SD_FS

代码实现功能

代码实现了通过文件系统读取 SD 卡上的文件，并对文件和文件夹做读、写、删除、建立、文件列表、文件搜索等功能，代码以将文件系统和 SD 卡驱动挂载在一起，直接运行代码可对格式化过的 SD 卡进行文件建立等操作。
编译运行代码，代码会对 SD 卡根目录下的文件进行文件列表，然后将第一个文件进行复制，并保存为“2.txt”，通过计算机可以查阅复制的文件。

代码使用说明

工程下有两个文件夹：bf53x_sd_mmc_lib 和 fat32_lib 。 这两个文件夹分别是 SD 卡驱动的库源代码和文件系统的库源代码，直接运行这两个文件夹下的工程，会在各自的 DEBUG 文件夹下生成*.dlb 文件，这个文件就是库文件。在工程文件夹的根目录下，已经有这两个库文件，这两个库文件被 BF53x_SD_FS.dpj 工程调用，实现文件系统功能。
SD 卡速度的配置：

```
void MMC_HardwareInitial(void)
{
    *pFIO_DIR = PF2;
    *pFIO_FLAG_S = PF2;
    *pSPI_FLG = FLS2;
    *pSPI_BAUD = 4;
    *pSPI_CTL = 0x0000;
    *pSPI_CTL = 0x0001 | MSTR ;
    *pSPI_CTL = (*pSPI_CTL | SPE);
}
```

修改*pSPI_BAUD 的值进行配置，当为 2 时，速度最快，配置值不能小于 2。
当使用 SD 卡时，需通过 CPLD 打通 SD 卡链路，配置如下：

```
*pDEVICE_OE     = ~SD_SEL;
```

文件列表函数：将“/”目录下所有文件和文件夹进行列表，并将文件和文件夹的名称和数量存入指定数组。

```
scan_files("/",file_name,file_count,dir_name,dir_count);
```

代码实验步骤

1. 将 SD 卡格式化，并在 SD 卡根目录下存一些.txt 文件。
2. 将 SD 卡插入开发板的 SD 卡接口，编译并运行代码。
3. 待代码运行完成后，将 SD 卡插入计算机，查看建立的 2.txt 文件。

代码实验结果

在 SD 卡根目录下将文件列表的第一个文件行复制并将文件名改为 2.txt。

BF53x_TOUCH_LINE

代码实现功能

代码实现了读取触摸屏坐标，并将触摸屏坐标换算为液晶屏的显示坐标，将像素点显示到触摸坐标的位置，实现了通过触摸屏在液晶屏上进行简单的线条绘画。
代码设置了防触摸屏坐标误触发机制，通过算法尽量减少触摸屏野点。使触发坐标更准确。
运行代码，通过用手指或者其他光滑的物体在液晶屏上划动，可以看到液晶屏上显示出划动的轨迹。

代码使用说明

触摸屏野点算法：
为确保触摸屏显示坐标准确，通常采用野点处理算法，代码中每次采取 9 个坐标点，然后对其取平均值和差值，排除偏离实际坐标较大的坐标值，将准确的坐标输出。
触摸屏坐标与液晶屏坐标换算：
X 坐标换算：　　tem_x = (tem_x-0xb0)*100/763;
Y 坐标换算：　　tem_y = (tem_y-0x150) *100/1288;
如感觉坐标定位不准确，可适当修改上面的换算参数值。
在指定坐标显示一个点：
因液晶屏最终显示的数据格式为 RGB565，所以显示点的数据为 16bit ,由两个 8bit 的数据分别送入。

```
void play_point(unsigned int x,unsigned int y,unsigned char *pdispbuf)
{
    pdispbuf[2*x+0+y*960] = 0xff ;
    pdispbuf[2*x+1+y*960] = 0xff;
}
```

代码实验步骤

1. 编译并运行代码
2. 用手指在液晶屏上划动
3. 观察液晶屏显示

代码实验结果

液晶屏上会显示出手指划动的轨迹。

BF53x_TOUCH_MOUSE

代码实现功能

代码实现了读取触摸屏坐标，并将触摸屏坐标换算为液晶屏的显示坐标，将光标数据叠加到背景数据上，然后显示出来，再次触动触摸屏，光标数据会重新刷新计算叠加，不会再背景上留下上次光标的拖影，实现数据实时叠加运算功能。

代码使用说明

代码主要实现了光标叠加功能，当有触摸屏触发，会计算出触发坐标，然后将背景图像送入显示内存，然后再将光标数据叠加到内存上，这样保证每次刷新光标，背景上都不会有残留的光标数据。

```
tem_x = (tem_x-0xb0)*100/800;
tem_y = (tem_y-0x150) *100/1288;
memcpy(DisplayBuffer_565,TempBuffer_img,261120);
Mouse(tem_x,tem_y,DisplayBuffer_565);
```

将 X 和 Y 的坐标换算成液晶屏显示坐标，通过 memcpy 函数将背景数据从存放的数组 TempBuffer_img 中拷贝到显示数组 DisplayBuffer_565 中，拷贝数据大小为 261120 字节。再将光标叠加到显示区域 DisplayBuffer_565 中。

代码实验步骤

1. 编译并运行代码
2. 用手指在液晶屏上划动
3. 观察液晶屏显示

代码实验结果

液晶屏上会看到有个光标会随着手指划动，跟随显示。

BF53x_ZIKU

代码实现功能

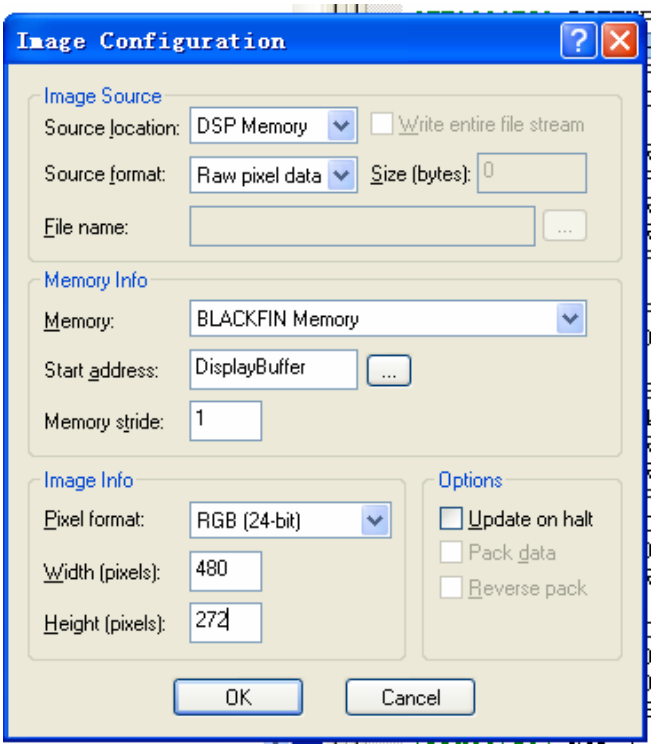
代码实现了汉字库和 ASIC II 码 字库的功能，代码会在编译时，将字库文件加载到内存中，所以必须使用 xml 文件对内存进行初始化，所以必须用光盘中的 ADSP-BF533-proc.xml 文件把 Visual DSP++ 5.0 下的 ADSP-BF533-proc.xml 文件替换掉，才能保证字库文件正确加载。
代码实现了在指定的坐标处显示出指定的汉字，字母，光标及平铺区域显示，可以对文字的颜色进行修该。
编译并运行代码，会将指定的文字数据写入指定的内存区域，通过使用 image view 工具进行查看。

代码使用说明

清除内存：
memset(DisplayBuffer,'\0',391680);
将 DisplayBuffer 内存清为空，清楚大小为 391680 字节。
ASICII 码显示：
Glib_disp_ascii16x8_v(130,0,"A",0x0000ff);
在坐标 130，0 位置显示字母 A，颜色为蓝色。
汉字显示：
Glib_disp_hzk16_v(130,200,"北京",0xffffff);
在坐标 130，200 位置显示汉字“北京”，颜色为白色。
填充区域：
Rect(100,100,200,100,0xff0000);
在坐标 100，100 位置填充一个宽为 200，高为 100 的区域，填充色为红色。
光标显示：
Mouse(240,135,0x00ff00);
在坐标 240，135 位置显示绿色的光标。

代码实验步骤

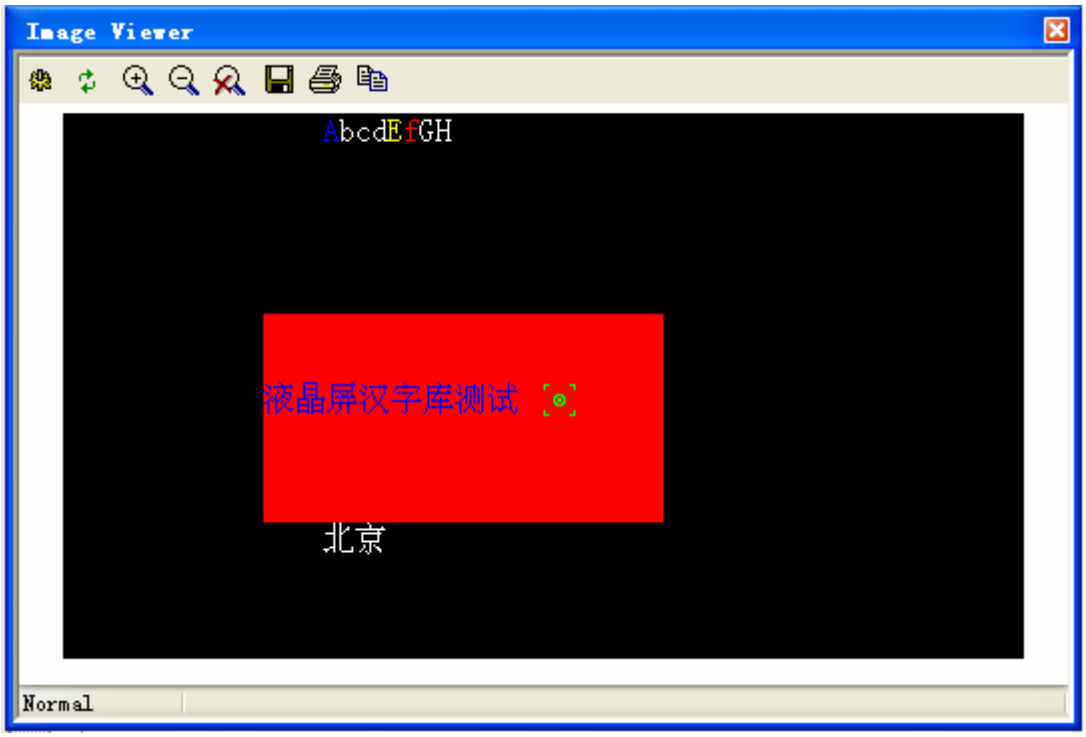
1. 编译并运行代码
2. 待代码运行完成后，选择 Visual DSP++5.0 菜单下“View -->DebugWindows-->image viewer…”选项。
3. 按下图配置选项：



4. 点“OK”后观察输出的内存图像。

代码实验结果

在 image view 窗口上可以看到如下图像：



BF53x_LCD_ZIKU

代码实现功能

代码实现了将字库信息显示到液晶屏上

代码使用说明

增加了液晶屏驱动，将内存数据显示到液晶屏上。
内存数据转换：
RGB888_RGB565(DisplayBuffer,391680,DisplayBuffer_565);
将 DisplayBuffer 数组中的 RGB888 格式数据转为 RGB565 格式，存在 DisplayBuffer_565 中，数据转换大小为 391680 字节。

代码实验步骤

1. 编译并运行代码

2. 观察液晶屏显示

代码实验结果

在液晶屏上可以看到内存的数据显示出。

BF53x_LCD_TXT

代码实现功能

代码实现了读取 SD 卡中“/txt/test.txt”路径下的 TXT 文件，将 TXT 文件内容显示到液晶屏上，通过按键“Lift->UP”和“Lift->Down”进行上下翻页。
代码读取 TXT 文档后，会根据读取的数据信息，将数据信息及格式进行判断，将内容通过调用字库显示到液晶屏上，并将每一页的地址保存在页数组中，在上下翻页时，会根据保存的信息找到以前的页面。

代码使用说明

代码主要调用了文件系统函数和 TXT 文档解析函数，文件系统将 SD 卡内指定路径下的测试文件打开并读取，TXT 解析函数将数据进行解析，并将相应的文字调入液晶显示内存中，并做了自动换行、换页、向上翻页的机制。

display_txt(unsigned char *pbuff,WORD len,int color)

将 pbuff 指针指向的地址数据调用并解析显示，调用长度参数为 len，颜色参数为 color。

代码实验步骤

1. 将 SD 卡根目录下的 TXT 文件夹中，保存一个叫做”test.txt”的文件，文件中写入要显示的内容或小说。
2. 将 SD 卡插入开发板 SD 卡接口。
3. 编译并运行代码，待提示“开始阅读电子书，按按键翻页查看”时，按下“Lift->Down”按键，通过“Lift->Down”和“Lift->UP”按键可进行上下翻页。

代码实验结果

在液晶屏上可以看到 test.txt 文件的文字内容。

BF53x_JPEG_DECODE

代码实现功能

代码实现了将 480*272 尺寸的 JPEG 数据解码为 RGB888 数据功能，调用了 JPEG 解码库函数。
JPEG 图像数据以.dat 文件的方式，通过编译代码时，自动加载到 windowsBuffer_t 空间中，运行代码会将 JPEG 格式的数据解码并存入 TempBuffer 数组中。

代码使用说明

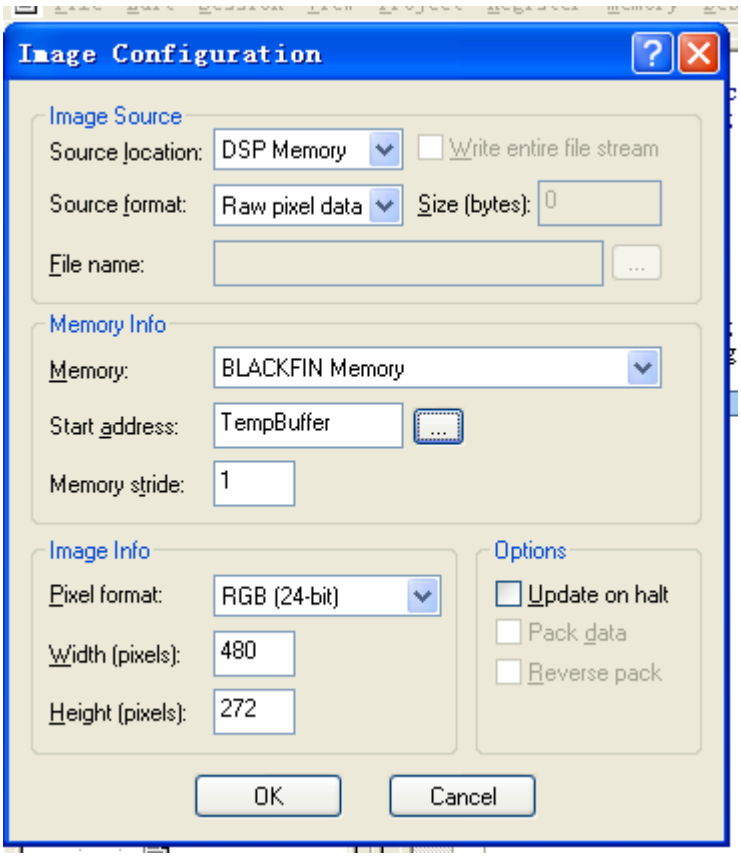
代码调用了一个 JPG 解码库，该解码库可以解 480*272 尺寸的 JPG 文件，将 JPG 文件解码为同尺寸的 RGB888 格式的数据文件。
JPG 解码函数：

jpg_scaling_rgb24(TempBuffer,480,272,windowsBuffer_t);

将 windowsBuffer_t 内存中的 JPEG 数据解码后存入 DisplayBuffer 中，解码图像的尺寸为 480*272.

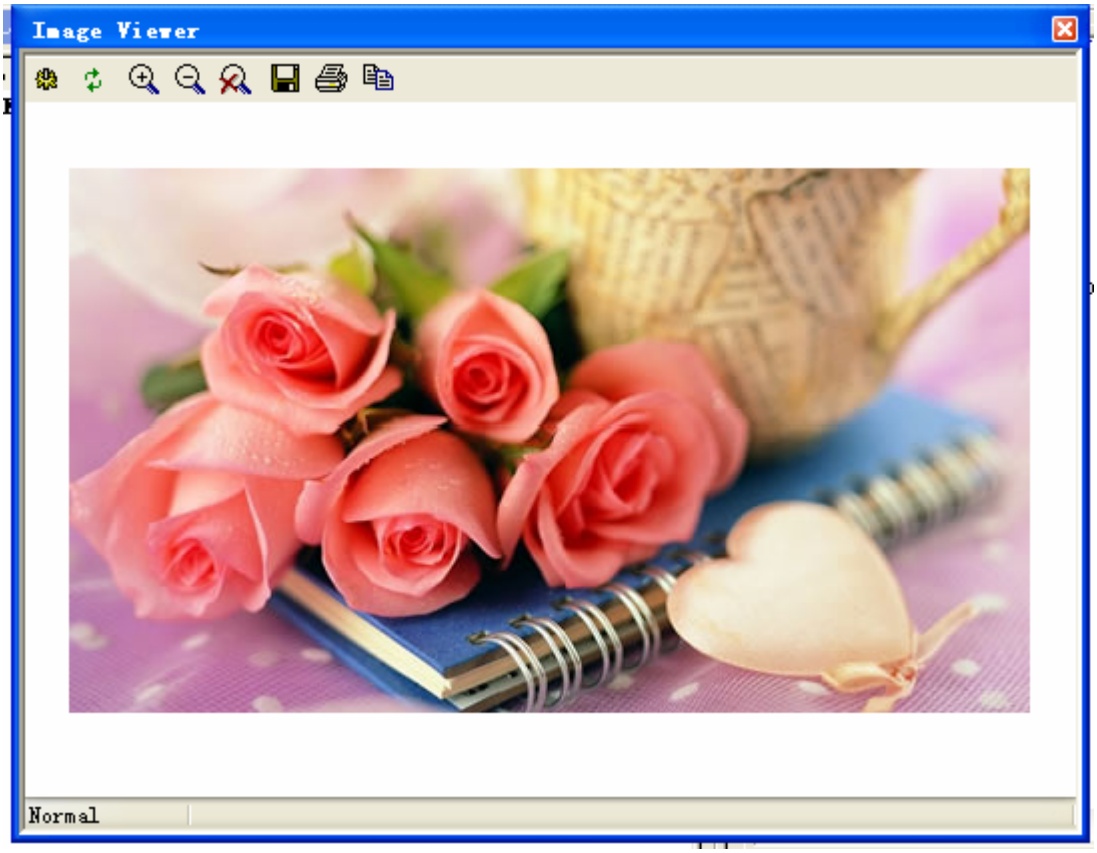
代码实验步骤

1. 编译并运行代码
2. 待代码运行结束后，选择 Visual DSP++5.0 菜单下“View -->DebugWindows-->image viewer…”选项。



代码实验结果

在 image view 工具中看到解码后的数据图像：



BF53x_JPEG_LCD_FS

代码实现功能

代码实现了将 SD 卡根目录下的所有文件进行文件列表，然后将 480*272 尺寸的 JPEG 文件进行 JPEG 解码，将解码后的数据显示到液晶屏上，延时后切换下一张图片，并且循环显示所有图片，实现电子相册功能。

代码使用说明

代码调用了一个 JPG 解码库，该解码库可以解 480*272 尺寸的 JPG 文件，将 JPG 文件解码为同尺寸的 RGB888 格式的数据文件，再将数据由 RGB888 转为 RGB565 送到屏上显示。

JPG 解码函数：

jpg_scaling_rgb24(DisplayBuffer,480,272,windowsBuffer_t);

将 windowsBuffer_t 内存中的 JPEG 数据解码后存入 DisplayBuffer 中，解码图像的尺寸为 480*272.

代码实验步骤

3. 将 480*272 尺寸的“.jpg”以短文件名命名后存放在 SD 卡根目录下，将卡插入板卡 S D 卡接口
4. 编译并运行代码
5. 观察液晶屏上的图片显示

代码实验结果

在液晶屏上看到循环显示的 JPEG 图片。

BF53x_AUDIO_MP3DECODE

代码实现功能

代码实现了读取工程目录下的一个 MP3 文件，对文件进行解码后，将生成的数据以 PCM 文件的形式保存在工程目录下。代码调用了 MP3 解码库，该解码库可对采样率 44.1KHz，位速 128Kbps 的 MP3 文件进行解码。解码后文件以 PCM 格式保存，通过 GoldWave 软件可对文件进行播放。

代码使用说明

主要函数说明：

```
int find_head(unsigned char *start,unsigned char *end)
```

参数：start ： mp3 数据的起始内存地址
End ： mp3 数据的终止内存地址
返回值：音频帧的起始地址，相对 start 的偏移量。
作用：从 start 到 end 的内存进行音频帧的查询，得到距离 start 地址最近的帧位置。

```
int check_data(char * pMp3Stream,char *pPCMStream,int *in_size,int *out_size,FILE *save_file)
```

参数：pMp3Stream ： mp3 数据帧的起始位置，可参照 demo 的使用（至少有 0x20 个数据
字节）：mp3 数据的起始地址+帧的偏移地址；
pPCMStream ： mp3 解码数据的存储地址，必须要保证 4920 字节长度。
in_size ： 返回针对本 mp3 音频格式的每次解码所需的数据帧大小
out_size ： 返回针对本 mp3 音频格式的每次解码后的数据大小
save_file：如果进行文件保持，该项传递要保存文件的指针。
如果不用进行文件保存，则传递 0x0 参数。
返回值：1：表示检测过程真确，得到了文件的信息，并保证返回的参数有效；

-1：表示本次检测失败；原因：pMp3Stream 的地址是否是帧起始地址；该地址开始是否存储有 mp3 数据 0x20 个字节。

作用：通过对 mp3 的帧数据进行检测，得到 mp3 文件的相关信息，并计算出该文件的帧大小和解码后数据的大小。

```
int MP3DEC_decode( unsigned char *pInData,
                   unsigned long inSize,
                   unsigned char *pOutData,
                   unsigned short *pGetSize,
                   MP3DEC_Params *pDecParams)
```

参数：pInData：mp3 的单帧起始地址

inSize ：单帧的数据大小

pOutData：单帧解码后的数据存储起始地址

pGetSize ：已解码的数据大小

pDecParams：解码的相关参数

返回值：1：表示该帧数据解码的数据有效

其它：表示该帧数据解码数据无效

作用：对需要解码的 mp3 数据帧进行解码；

.

片断分析：

```
    pMp3Stream = mp3_stream;
    pPCMStream = dataA;
    MP3DEC_init();     //初始化相关的数据
  output_file = fopen("../test.wav","wb");//打开解码结果的存储文件
  input_file = fopen("../test.mp3","rb");//打开需要进行解码的文件
  result_num = fread(mp3_stream,1,0x300000-0x1000,input_file);
//读取的大小 0x300000-0x1000 是针对本 demo 代码只给予输入 mp3 数据最大的内存容量
//来决定的
  asm("ssync;");
  mp3_stream_bufend = &mp3_stream[0] + result_num;//mp3 数据的结束地址
  head_flag = find_head(pMp3Stream,mp3_stream_bufend); //寻找 mp3 数据的音频同步
//帧的位置
  decode_flag = pMp3Stream+head_flag;
  result = check_data(decode_flag,pPCMStream,&input_mp3,&output_mp3,output_file);
  //检查音频帧和数据流
  if(result == -1)
  {
      printf("error mp3 file format \n");
  }
  while(1)
  {
```

```
        result = MP3DEC_decode(decode_flag,input_mp3,pPCMStream,&decodedSize_t,&myDecParams_t);
//进行每一音频帧的解码操作
          if(result == 1)
          {
          frame_num++;
          pPCMStream += output_mp3;
          if(frame_num>0x100)
          {
                fwrite(dataA,1,output_mp3*frame_num,output_file);//保存解码结果
                asm("ssync;");
                frame_num = 0;
                pPCMStream = dataA;
          }
    }
        decode_flag += input_mp3;
        if(decode_flag+input_mp3 >= mp3_stream_bufend)//判断解码是否完成
        {
         fwrite(dataA,1,output_mp3*frame_num,output_file);
         asm("ssync;");
                break;
        }
    }
fclose(output_file);
fclose(input_file);
MP3DEC_exit();
```

注意事项：

1. 函数只能对 128kbps、44.1kHz 存储格式的 mp3 文件进行解码；
2. 用户在自己生成工程中使用该解码库，必须向工程中添加 mp3dec_lib.dlb， testbf533.ldf。
3. 如果用户更改了工程属性，那么该操作会反映给 LDF 文件，那么需要把原始的 LDF 再次拷贝到工程下。

代码实验步骤

1. 编译代码
2. 运行代码，查看原工程下生成的 PCM 文件，正常解码后，刷新时能看到 PCM 文件大小会慢慢增加。如，没有慢慢增加或过早退出工程，说明解码失败，可以尝试重新编译代码，然后按 F10 键，单步方式将代码运行到 MP3 循环解码的函数里，然后再按 F5 键连续运行。
3. 待代码运行结束后，确认代码已执行过代码中的 fclose(output_file)函数，然后可以使用 GoldWave 软件，打开 PCM 文件。当提示文件打开格式时，按如图配置：

4. 加载完数据后，即可播放解码后的 PCM 数据流。

代码实验结果

工程下生成 MP3 文件解码后的 PCM 文件，用 GoldWave 软件可以对其进行播放，听到解码后数据音乐。

BF53x_AUDIO_PCM

代码实现功能

代码实现了打开 SD 卡根目下"../music/test.pcm"路径中的 PCM 文件，并读取 6MB 的数据到内存中，然后将内存中的数据进行循环播放，实现播放 PCM 音乐的功能。将耳机插入绿色的音频接口，可以听到正在播放的音乐。

代码使用说明

代码采用了 SPORT 接口的描述符 DMA 实现，通过 SPORT 口以 I2S 方式，将音频数据送给音频解码芯片，由音频解码芯片将数据转为音乐信号输出。

代码实验步骤

1. 将 SD 卡根目录下的 music 文件夹内存放一个名为 test.pcm 的声音文件，该文件可以由提供的 MP3 解码代码生成，也可以由 GoldWave 软件生成，文件格式必须为立体声 44.1KHz128Kbps 格式。
2. 将 SD 卡插入开发板，将耳机接入开发板上绿色的音频接口，编译并运行代码
3. 等待文件系统加载 PCM 数据，待数据加载完后，通过耳机能听到音乐输出。

代码实验结果

通过耳机听到 PCM 数据音乐输出。

BF53x_TOUCH_ORGAN

代码实现功能

代码实现了触摸屏电子琴的功能，运行代码后，系统会从ＳＤ卡的 organ 文件夹下加载声音文件，然后液晶屏上看到一个电子琴按键的界面，将耳机插入绿色的音频接口，用手触摸液晶屏，可以听到按下的按键声音。

该代码支持中音声音，如果感觉声音不好听，可以将ＳＤ卡 organ 文件夹下的文件替换，声音文件为 44.1KHz 128Kbps 立体声 PCM 数据文件，注意更换后文件名不要改变。

代码使用说明

代码通过文件系统将音频文件读入内存，然后根据触摸屏触摸的区域，将相应的音频文件进行播放，实现电子琴功能。

代码实验步骤

1. 将 SD 卡的 organ 文件夹下拷入提供的音频文件，将 SD 卡接入开发板，将耳机接入开发板绿色音频接口中。
2. 编译运行代码
3. 当液晶屏上看到琴键时，触摸液晶屏上的琴键。

代码实验结果

耳机中能听到触摸的按键的声音。

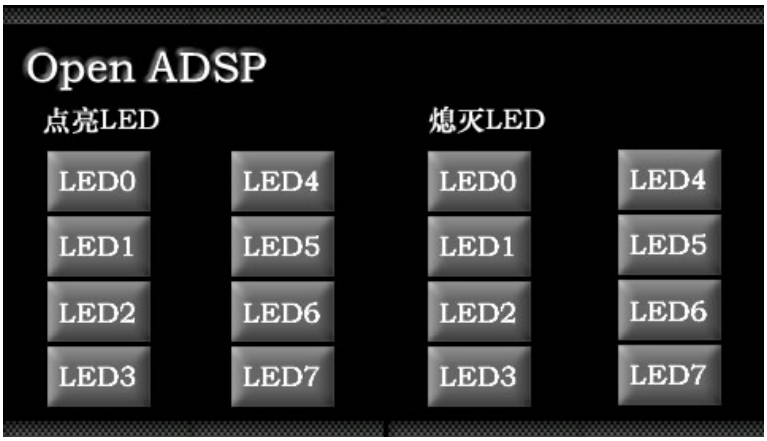
BF53x_TOUCH_LED

代码实现功能

代码实现了通过触摸屏界面控制板卡上 8 个 LED 指示灯的点亮和熄灭。

代码使用说明

代码调用了一幅 LED 控制界面的图片，根据图片上的功能按钮，划分出了触摸屏功能坐标区域，根据判断触发的区域，来实现相应的点亮和熄灭 LED 灯的功能。



代码实验步骤

1. 编译运行代码
2. 待液晶屏上出现 LED 控制界面，用手点击相应的控制按钮，观察 LED 指示灯的变化。

代码实验结果

LED 指示灯会根据液晶屏上的操作进行点亮和熄灭。

滤波器代码说明

BF53x_FIR

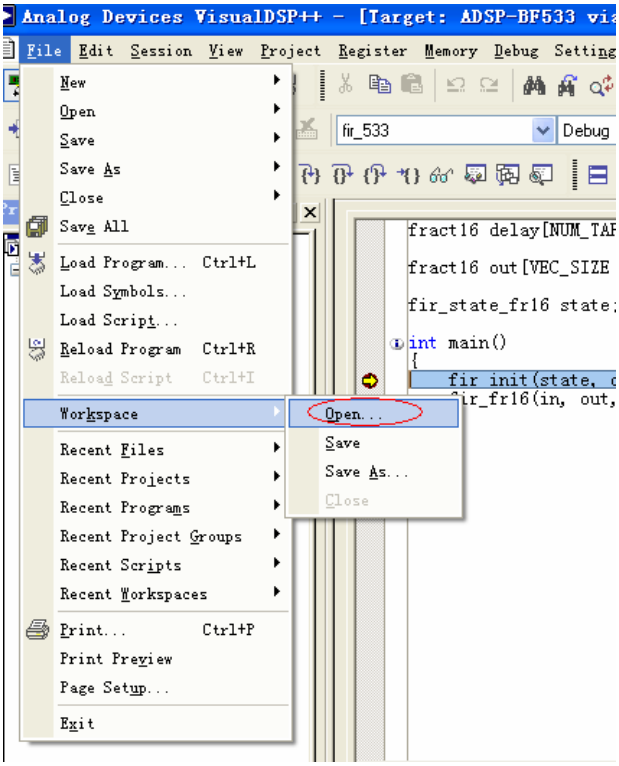
代码实现功能

FIR(Finite Impulse Response)滤波器：有限长单位冲激响应滤波器，是数字信号处理系统中最基本的元件，它可以在保证任意幅频特性的同时具有严格的线性相频特性，同时其单位抽样响应是有限长的，因而滤波器是稳定的系统。因此，FIR 滤波器在通信、图像处理、模式识别等领域都有着广泛的应用。

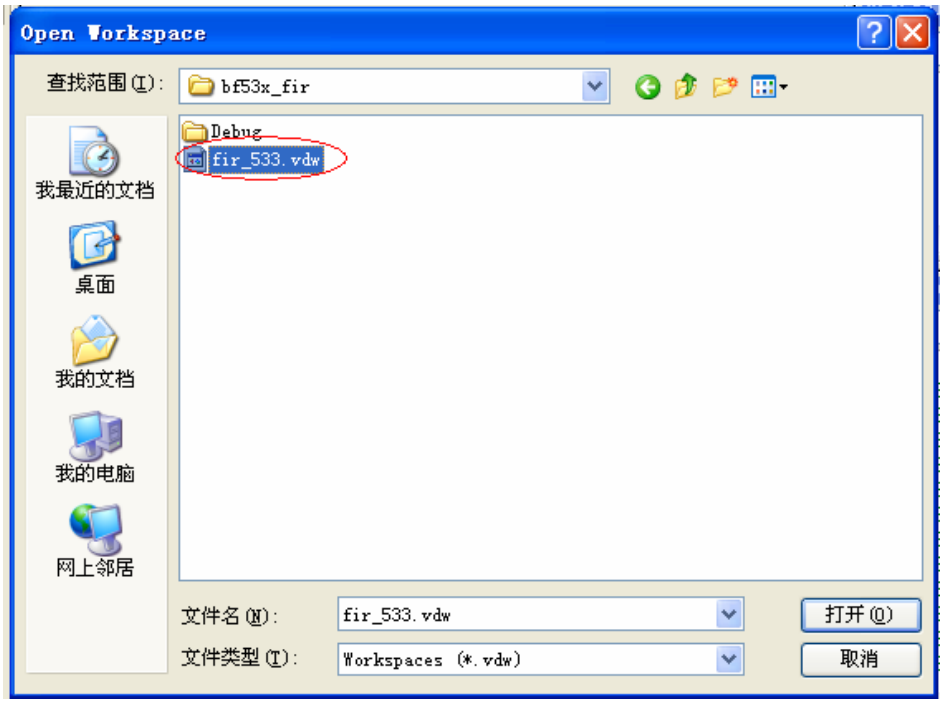
代码实现了通过算法实现 FIR 滤波器，对输入的波形进行滤波后输出，通过 Visual DSP 的 Polt 工具可以查看输入的波形和滤波后产生的波形。

实验步骤

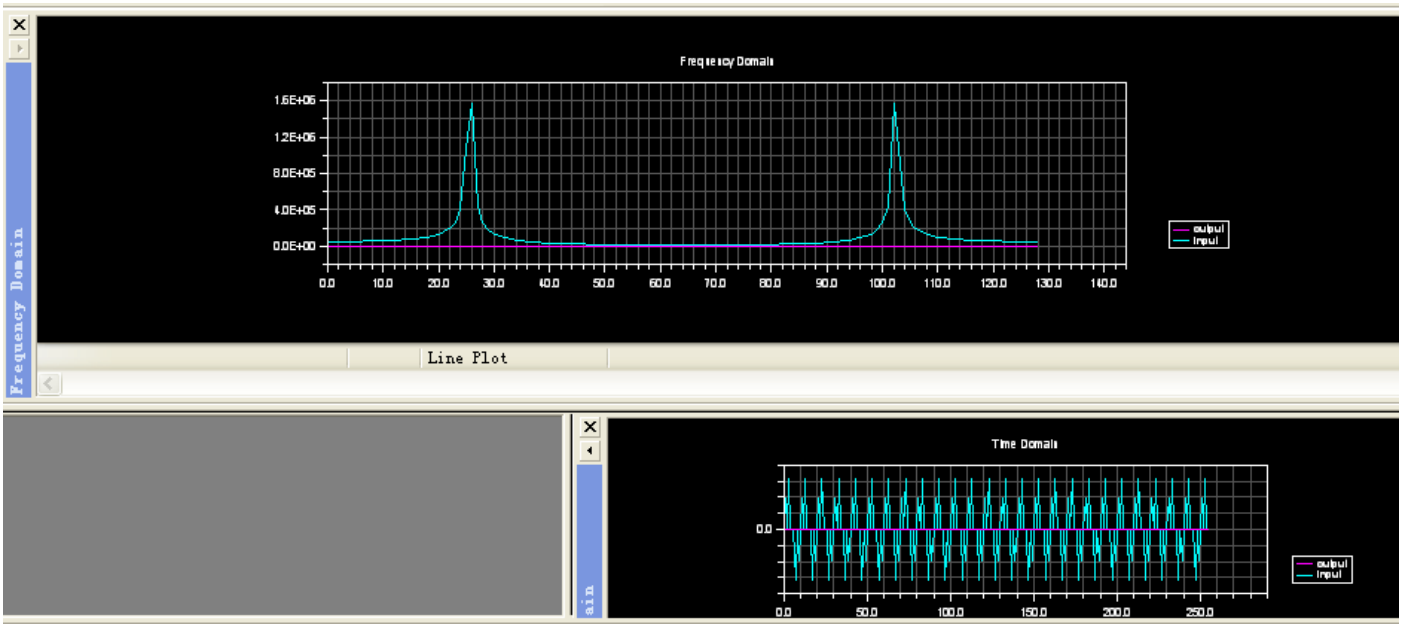
1. 将工程文件载入 visual DSP 软件，将软件与板卡连接。
2. 将按下图打开工程下保存的.vdw 文件



3. 在工程路径下找到 fir_533.vdw 文件。



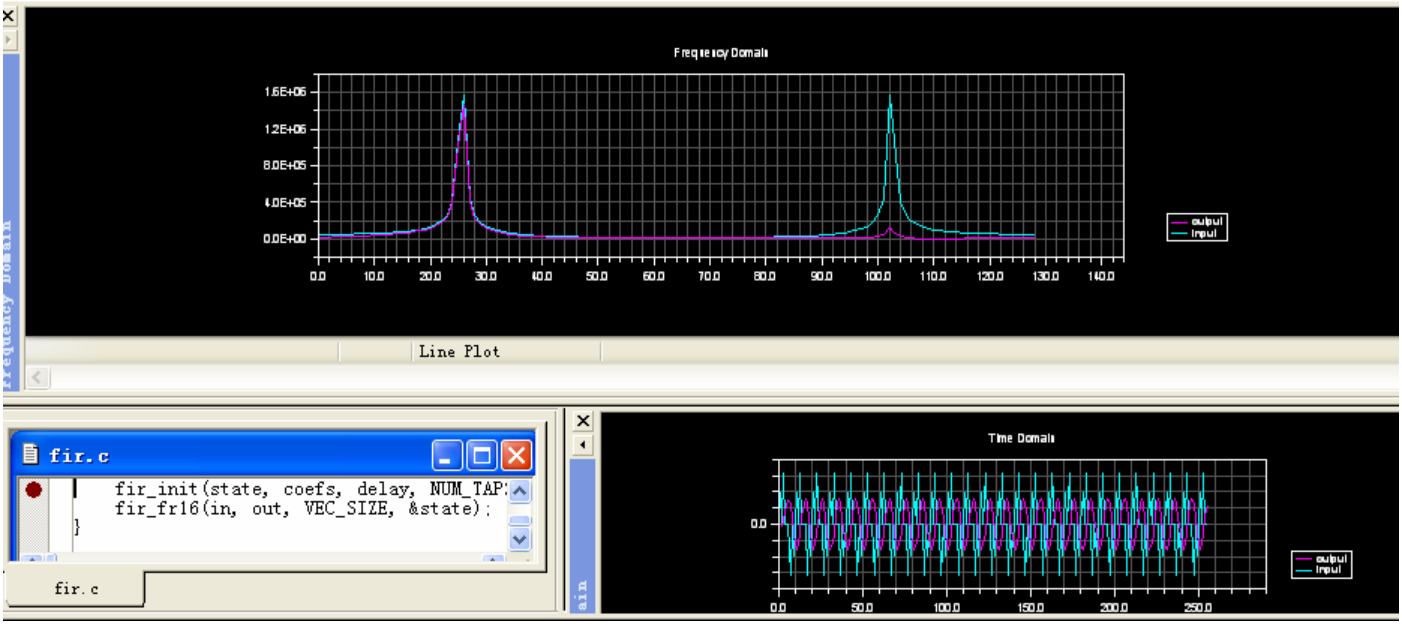
4. 打开文件后能看到会弹出下面的窗口，蓝色线表示待滤波的数据，红色线标识滤波后的数据。



5. 编译代码，并全速运行代码。

实验结果

代码运行后，会看到代表滤波后的数据的红线输出滤波后的波形。



BF53x_FFT

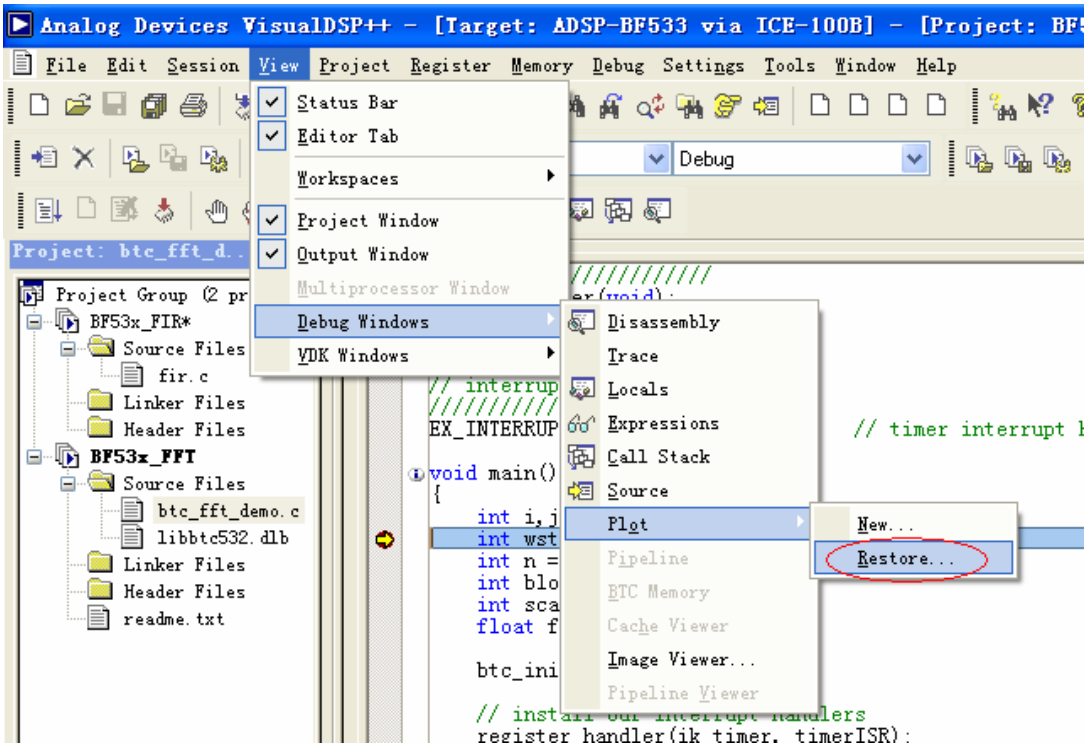
代码实现功能

FFT（Fast Fourier Transformation），即为快速傅氏变换，是离散傅氏变换的快速算法，它是根据离散傅氏变换的奇、偶、虚、实等特性，对离散傅立叶变换的算法进行改进获得的。

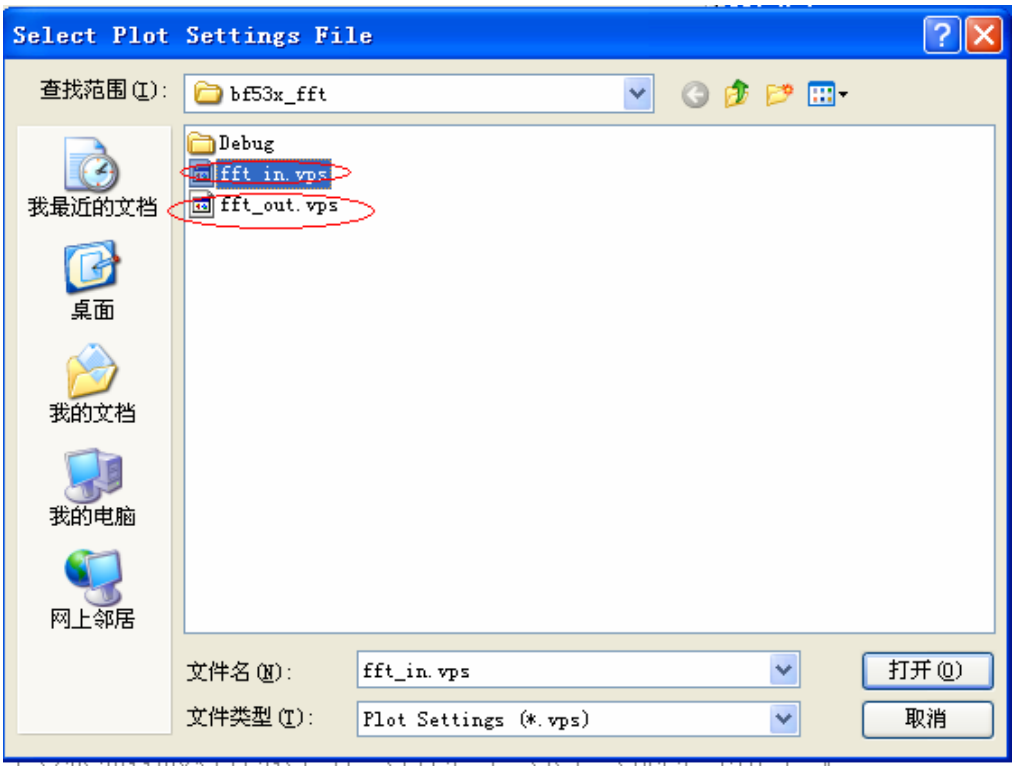
代码实现了通过 Visual DSP++ 软件的 BTC（Background Telemetry Channels）功能进行后台监测，FFT 算法产生一个输入的波形数据，经计算后产生输出的波形数据，然后将输入和输出波形数据由 BTC 控制，通过视图实时输出显示。

实验步骤

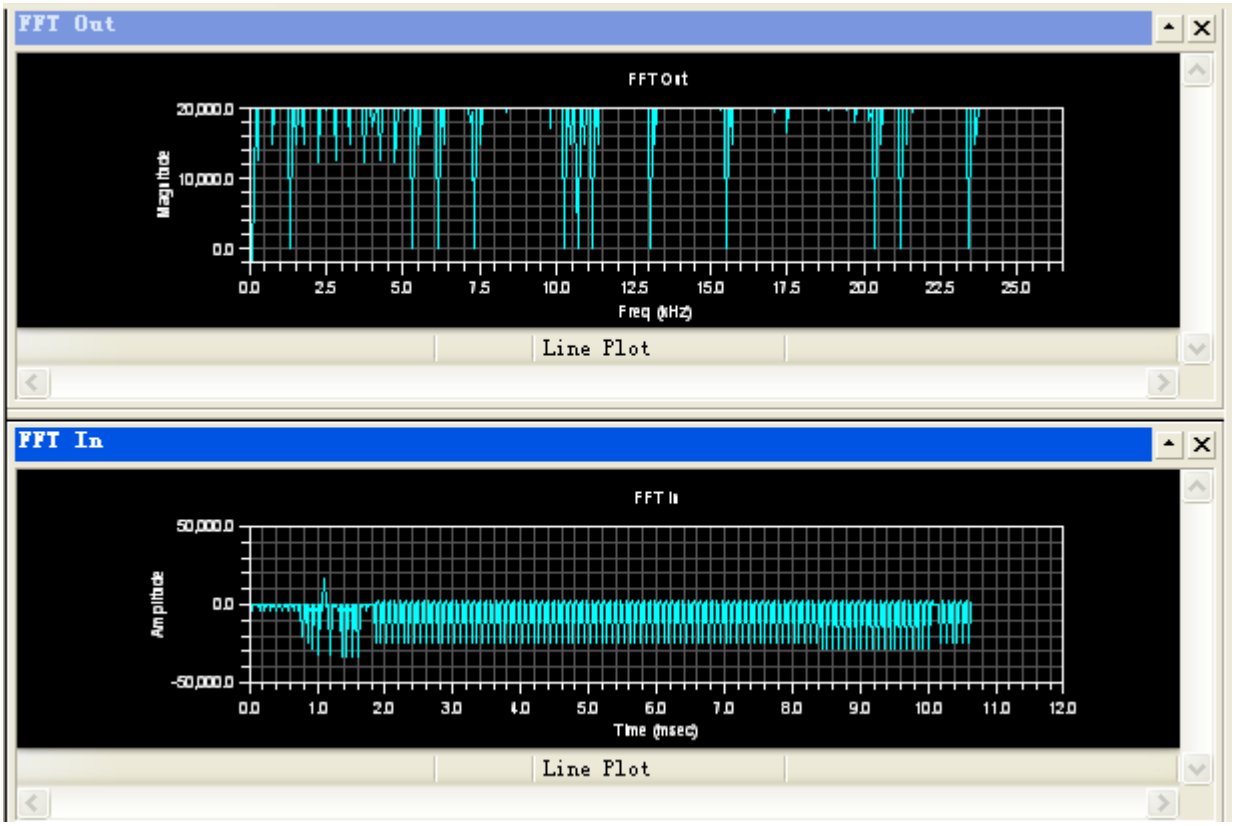
1. 按下图，选择“Restore”菜单。



2. 在弹出会话框上找到工程路径下 fft_in.vps 文件，打开。
3. 用同样的方法打开工程路径下 fft_out.vps 文件。

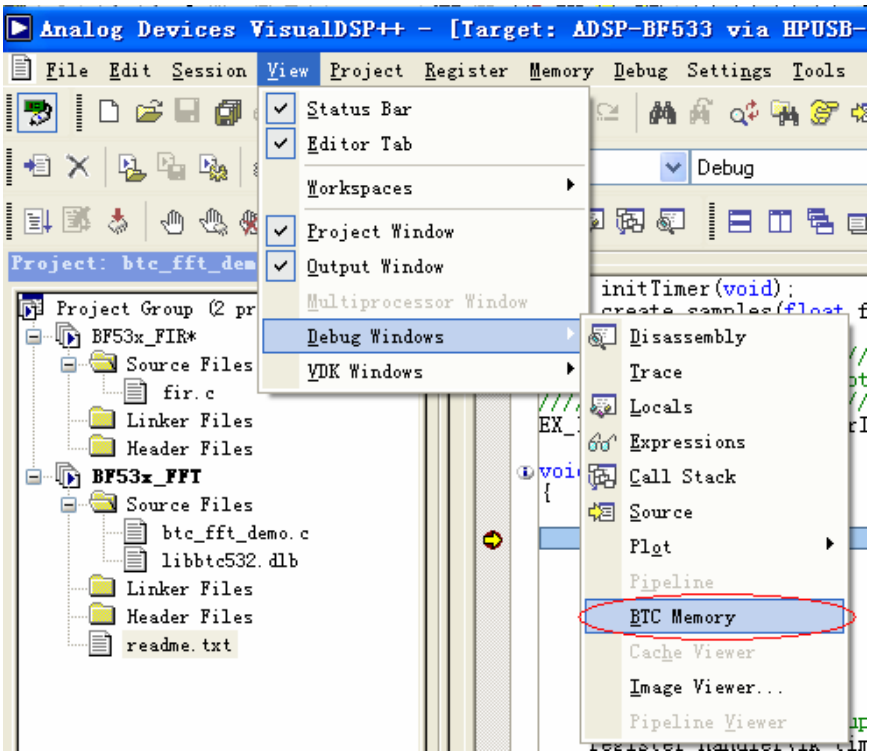


4. 打开后在 Visual DSP++软件下可以看到两个波形窗口。

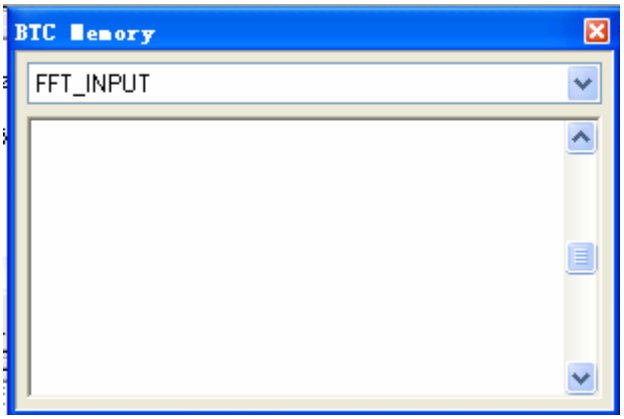


5. 打开 BTC Memory 窗口

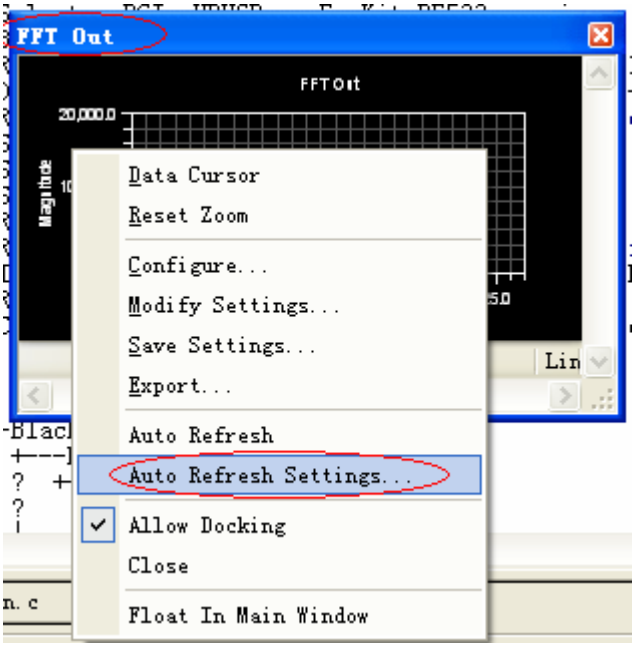
BTC Memory 窗口必须采用 AD-HP560ICE-FULL 仿真器和 ADI 原厂的 HP-ICE 仿真器时，才能使用。AD-HP510ICE-FULL 仿真器不支持该功能，该选项为灰色。



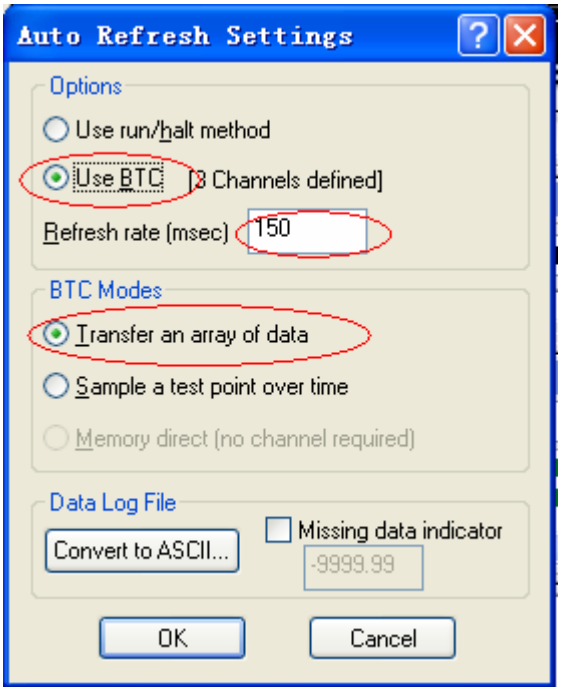
6. 打开后 BTC Memory 如图



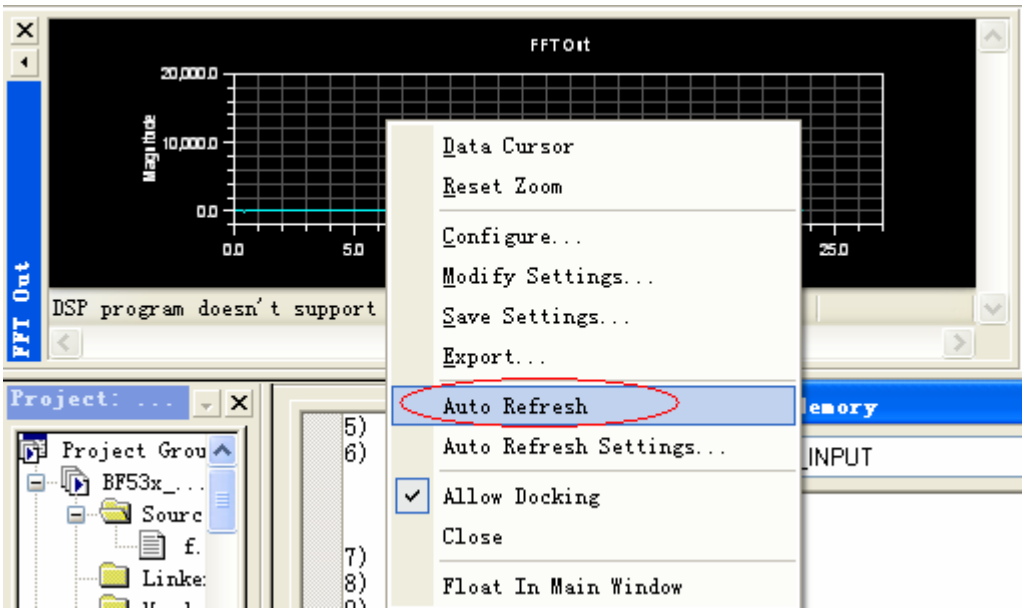
7. 选中 FFT OUT 波形视图框，按鼠标右键，选择“Auto Refresh Settings..”选项。



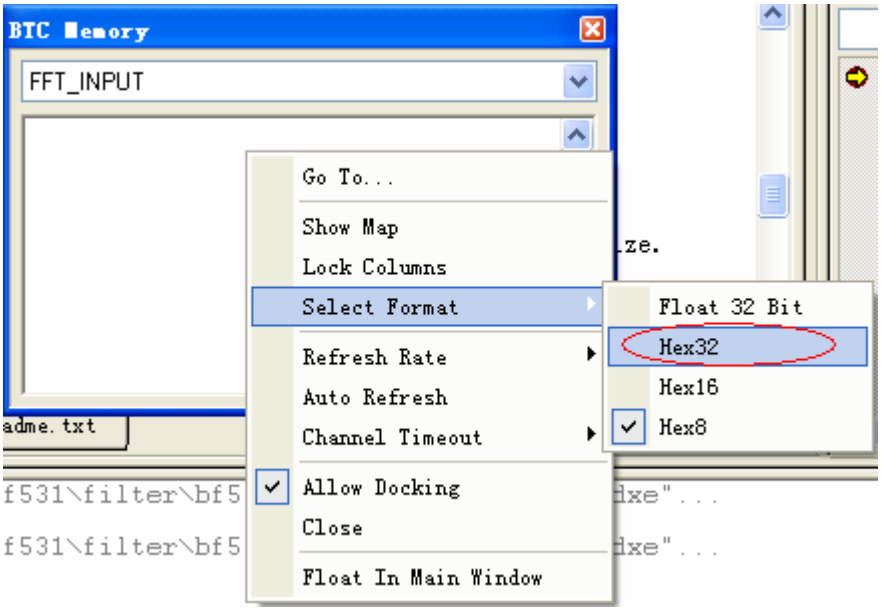
8. 在弹出会话框中，按如下设置，完后后点“OK”



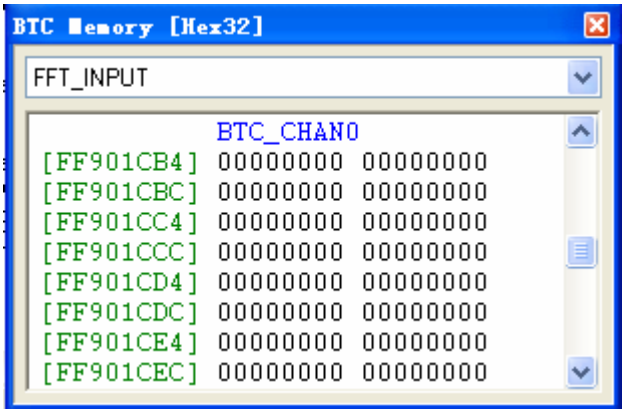
9. 再次调出鼠标右键菜单，选择“Auto Refresh”



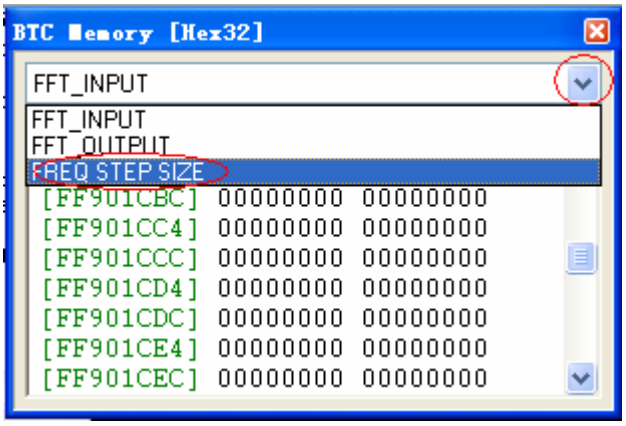
10. 用同样的方法，对 FFT IN 波形视图框进行设置。
11. 选中 BTC Memory 窗口，鼠标右键调出菜单，选择格式为“Hex32”



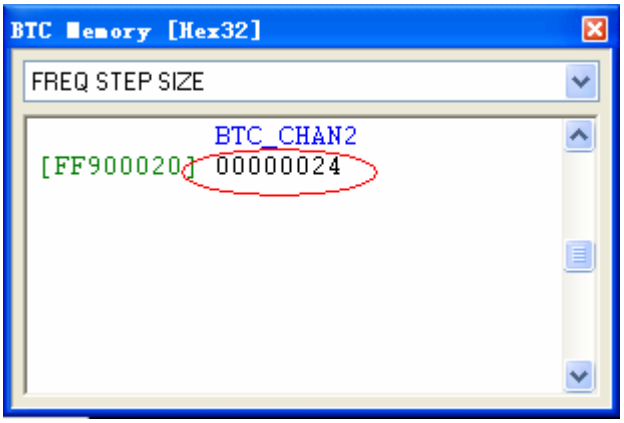
12. 选中后 BTC Memory 窗口以 32bit 显示。



13. 选中 BTC Memory 下拉菜单，选择“FREQ STEP SIZE”。



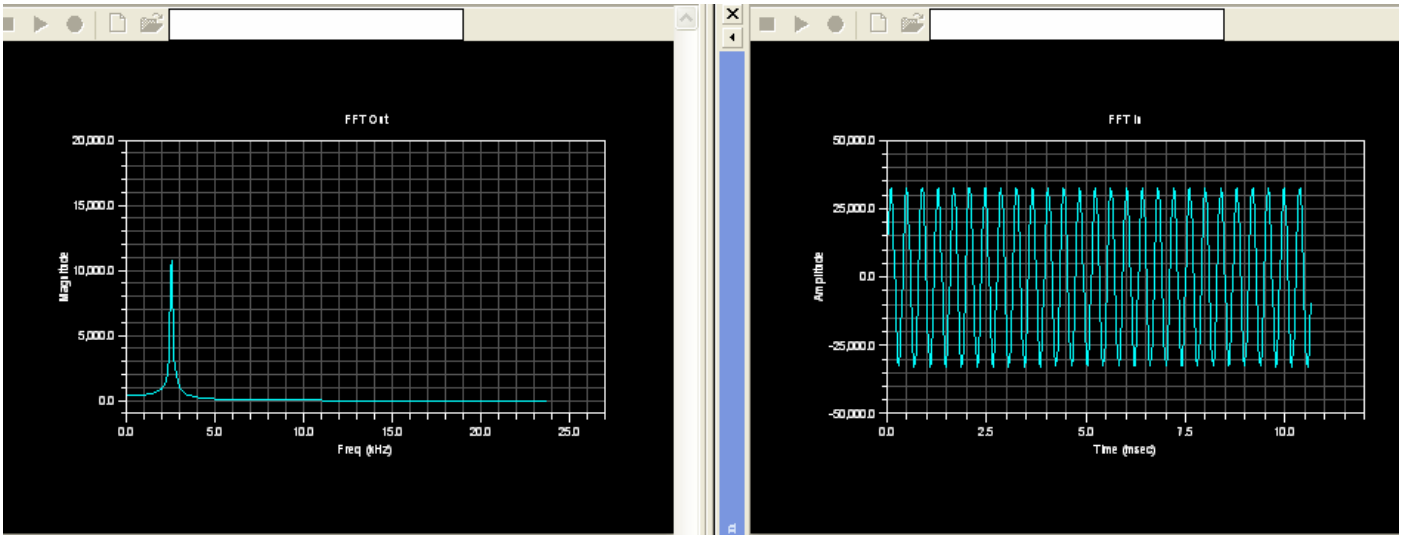
14. 在下面的地址中输入 10~100 任意数据，来设置改变波形和数据刷新的频率。



15. 编译并全速运行代码。

实验结果

运行代码后，在两个波形窗口中可以看到变化的波形图像。



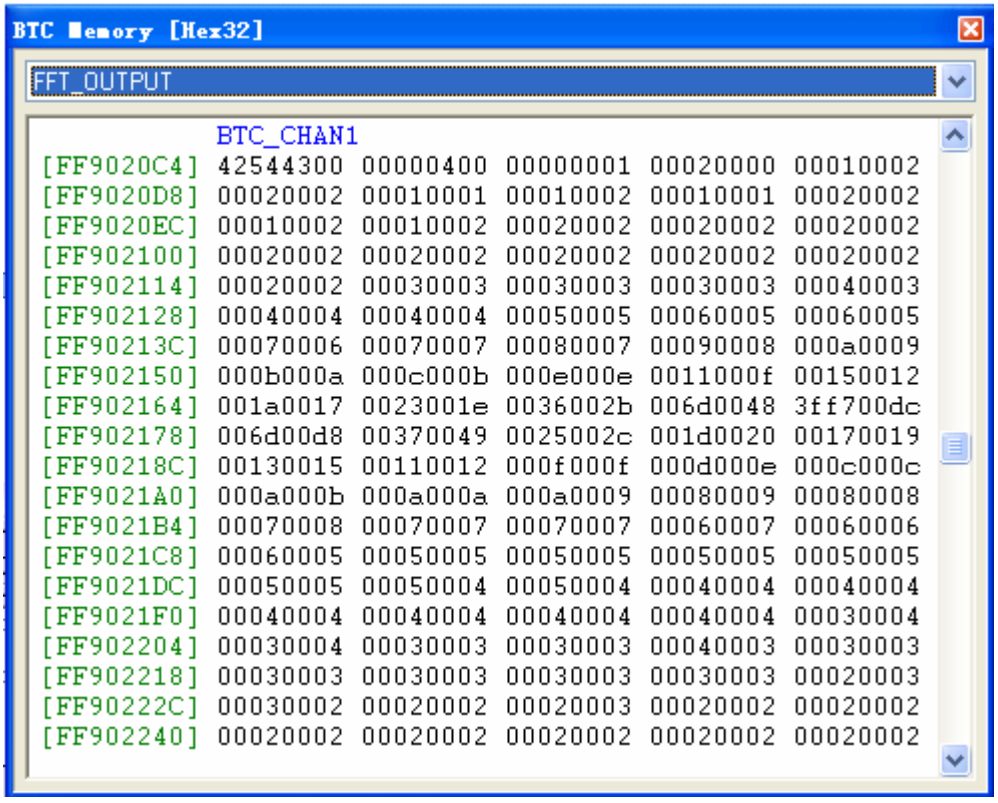
BTC 窗口检测到的 FFT_INPUT 的数据：

BTC Memory [Hex32]

FFT_INPUT

	BTC_CHAN0				
[FF901CB4]	42544300	00000400	00000001	71300000	f15c699d
[FF901CC8]	9f5988ba	7bcc1d17	d4d8566b	b4f2814f	7fec38a8
[FF901CDC]	ba963eb4	ce798087	7d595142	a3f723b3	ea98866c
[FF901CF0]	7436659b	922a06d4	07d592ae	64fd74a1	861de99b
[FF901D04]	24aaa4aa	507b7d8c	8072cd8c	3f93bb6f	37c17fe4
[FF901D18]	8175b423	5729d5ca	1c1c7b89	89189eb2	6a2df25c
[FF901D2C]	feff70b7	96f58e59	77a20fa3	e1ef5ffd	aa5383f4
[FF901D40]	7ed42c1a	c6714a3d	c22d800e	7f614641	adf83099
[FF901D54]	dd4482dc	79435cbb	99ca146a	fa2d8c36	6ccb6e5a
[FF901D68]	8af6f72a	17629ba1	5aa17a31	8243da60	3360b04e
[FF901D7C]	43b77fa2	8026bf8e	4cacc927	29427e64	84bba81c
[FF901D90]	61f5e4df	0ca47688	8fc49544	721c0202	ef5d6877
[FF901DA4]	a0ae8803	7c4a1f0b	d2f554ed	b696810b	7ff63a74
[FF901DB8]	b8e83cf1	d05580b8	7cee52cd	a29421c4	ec948710
[FF901DCC]	735b66d0	912504d2	09d693bd	63be7571	8584e7a1
[FF901DE0]	2696a616	4ee87dec	804ccbb5	4150bd24	35f07fce
[FF901DF4]	81c5b286	589ed7b2	1a257aff	89da9d66	6b49f45c
[FF901E08]	fcfd6fc0	981f8d70	785511a2	dffc5ea6	abd4837a
[FF901E1C]	7f142dfb	c4a84897	c3f28009	7f2c47ed	ac6f2ebc
[FF901E30]	df34834b	789b5e1b	9897126d	fc2f8d15	6bb96f5b

BTC 窗口检测到的 FFT_OUTPUT 的数据：



BF53x_NES_128K

代码实现功能

代码实现了 NES 游戏模拟器在 BF53x 平台上的运行。运行代码后，会通过文件系统打开工程目录下 ROM 内的 NES 游戏 ROM，然后运行游戏，在液晶屏上显示出游戏内容。通过开发板上的按键可进行游戏娱乐。目前代码解码支持 128K 及以下的 ROM 文件，大于 128K 的 ROM 文件不支持运行。模拟器音频部分没有做。

代码使用说明

代码通过文件系统打开 ROM：

```
fp = fopen("../rom/hundouluo.nes", "rb");
lenth = fread(rom_file, 1, 0x200000, fp);
fclose(fp);
```

打开液晶屏驱动：

```
LCDBK_OFF();
InitDMA();
```

```
    InitPPI();
    InitTimer();
    PPI_TMR_DMA_Enable();
    LCDBK_OE();
    Init_Timers0(1999);//1~1999  控制背光亮度
    Enable_Timers0();
```

运行模拟器：

```
    InfoNES_ReadRom();
    InfoNES_Reset();
    InfoNES_Main();
```

液晶屏数据更新：

```
void InfoNES_LoadFrame(void)
{
    int x,y;
    u16 color_data;
    u8 color_r,color_g,color_b;

    for(x = 0;x < 240;x++)
    {
        for(y = 0;y < 256;y++)
        {
        color_data =    ChColor(WorkFrame[x*256+y]);
        color_r = (color_data>>11)& 0x1f;
        color_g = (color_data>>5)& 0x2f;
        color_b = (color_data)& 0x1f;

        color_data = (color_r) | (color_g<<5) | (color_b<<11);

        DisplayBuffer_565[x][y*2+220] = (color_data & 0xff) ;
        DisplayBuffer_565[x][y*2+221] = ((color_data>>8) & 0xff);
        }
    }
}
```

模拟器按键操作：

```
u8 ReadKey(u8 KeyCodeLast)
{
    u8 KeyCode,i;
    u8 temp_code = 0;
    static u8 KeyDly[10]={3,3,3,3,3,3,3,3,3,3};
```

```
    temp_code = ~*pKEY_DAT;
    KeyCode = ((temp_code>>0)&0x1)<<4 ;
    KeyCode |= ((temp_code>>1)&0x1)<<6;
    KeyCode |= ((temp_code>>2)&0x1)<<5 ;
    KeyCode |= ((temp_code>>3)&0x1)<<7;
    KeyCode |= ((temp_code>>4)&0x1)<<3;
    KeyCode |= ((temp_code>>5)&0x1)<<1;
    KeyCode |= ((temp_code>>6)&0x1)<<0;
    KeyCode |= ((temp_code>>7)&0x1)<<2;

     for (i = 0; i < 7; i++)
     {
          if ((board_input&(1<<i))!=(KeyCode&(1<<i)))
          {
               KeyDly[i] = 3;
          }
          else
          {
               if ((KeyCode&(1<<i)))
               {
                    if (KeyDly[i] < 6)
                    {
                         KeyDly[i]++;
                    }
                    else
                         return (KeyCode);
               }
               else
               {
                    if (KeyDly[i] > 0)
                    {
                         KeyDly[i]--;
                    }
                    else
                         return (KeyCode);
               }
          }
     }
     return(KeyCodeLast);
}
```

按键设置说明：

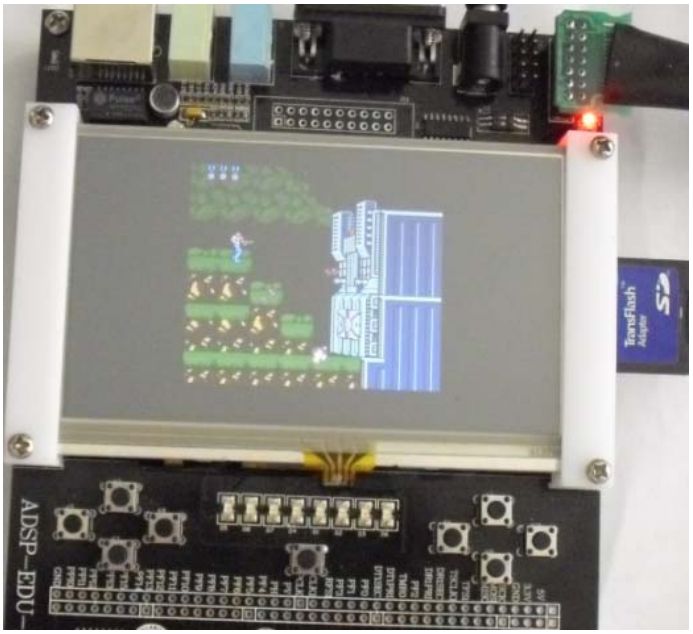
板卡按键	模拟器功能键
left->left	左
left->down	下
left->right	右
left->up	上
right->left	B
right->down	A
right->right	SELECT
right->up	START

代码实验步骤

1. 编译运行代码
2. 待屏幕出现魂斗罗画面，通过按键输入：上上下下左右左右 BABA，按 START，开始游戏。
3. 30 条命，祝游戏愉快。

代码实验结果

在液晶屏上出现魂斗罗画面，通过按键控制可进行游戏。



注：板卡上的按键使用寿命有限，仅限于运行游戏测试，请误长期使用板卡玩游戏，以免按键疲劳损坏。

更多应用正在添加中!

产品视频:

http://v.youku.com/v_show/id_XMjcxNzgyMDcy.html

http://v.youku.com/v_show/id_XMjg0ODMyNjM2.html

联系我们:

- 联系人:陈工
- 联系电话:15011475977
- 电子邮箱:sale@openadsp.com
- 传真:010-64811482

网站:www.openadsp.com

淘宝店铺:http://dsp-tools.taobao.com/