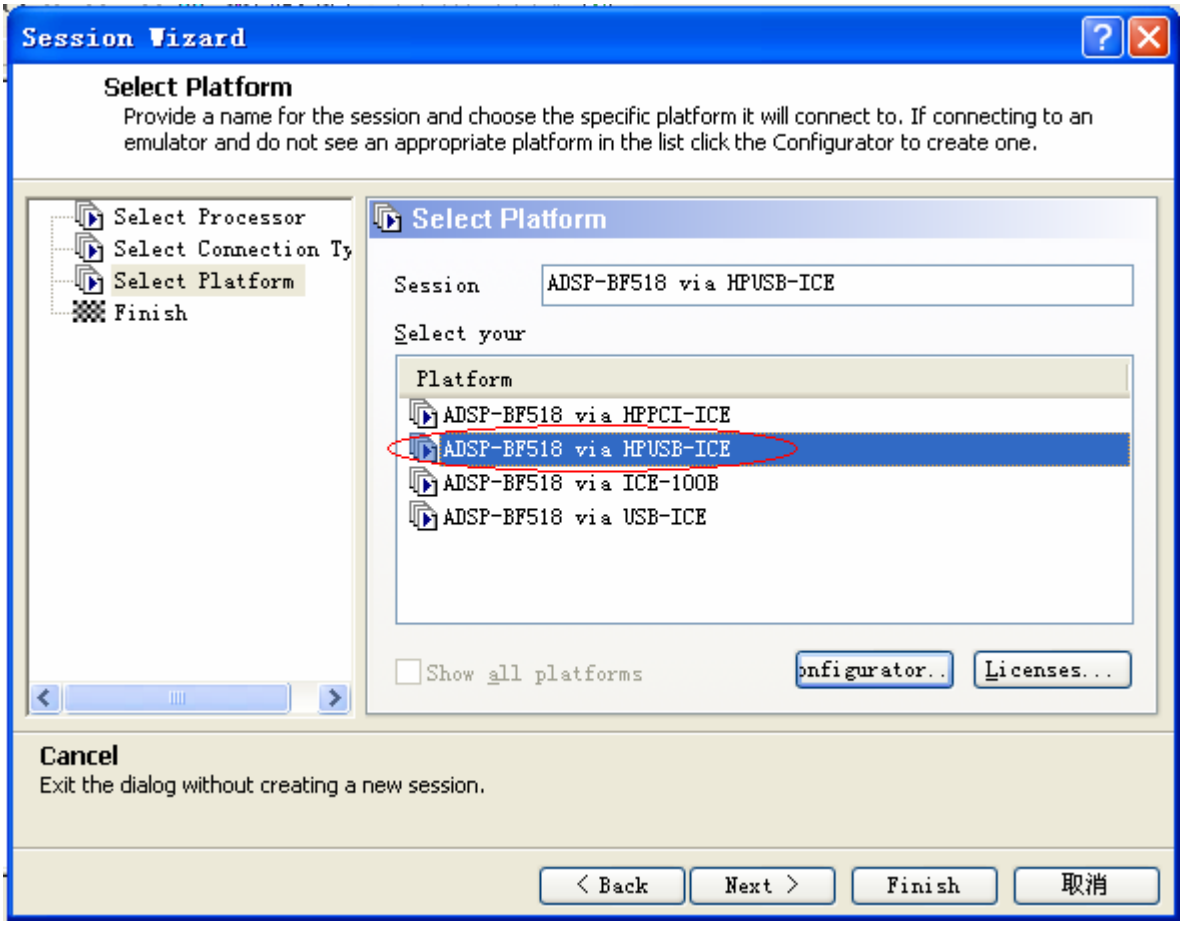
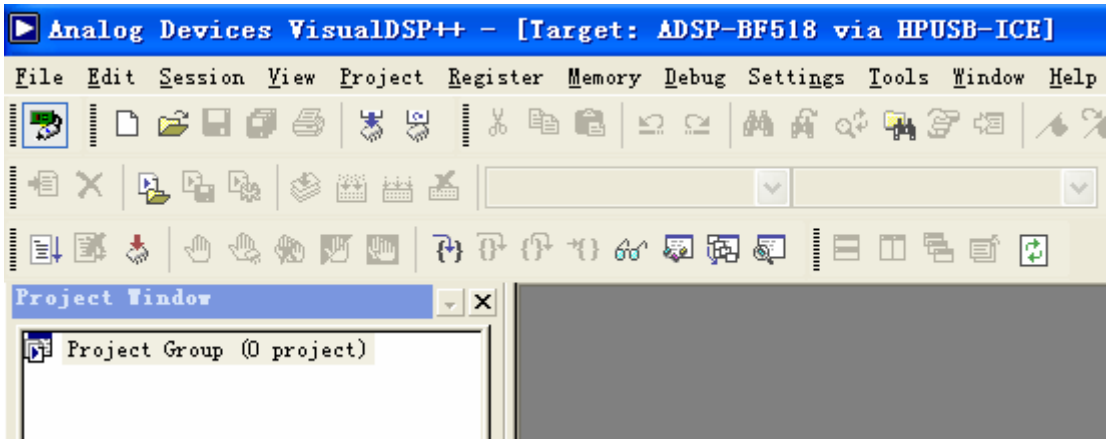


连接成功后在 VisualDSP++5.0 软件界面上可以看到连接状态，该状态代表目前仿真器在连接着板卡



3.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，以最基础的示例来诠释 Blackfin 的魅力。
本章教程参考代码位于“InterfaceCode”文件夹下。

3.1 BF51x_GPIO

3.11 接口功能介绍

ADSP-BF51x 处理器上有 40 个 GPIO 接口，分布在 PF0~PF15，PG0~PG15，PH0~PH7 端口上，通过寄存器配置，这些接口可以输出电平和感知接口电平，并可作为外部中断接口使用。

由于 BF51x 处理器接口复用功能很多，在使用 BF51x 端口之前，必须要对端口的功能作配置，以告知处理器使用的是什么接口。

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 IO 接口。

3.12 接口寄存器说明

PF 端口主要寄存器功能与使用方法

PF 端口寄存器	功能
PORTFIO	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
PORTFIO_CLEAR	清除标志位寄存器：将该寄存器内写 1，相对应的 PF 管脚被清除，电平置为 0。写 0 则无效
PORTFIO_SET	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
PORTFIO_MASKA	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
PORTFIO_MASKA_CLEAR	中断屏蔽清除寄存器 A：清除 PF 管脚的中断功能，写 1 则对应管脚关闭中断，写 0 则无效
PORTFIO_MASKA_SET	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
PORTFIO_MASKB	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
PORTFIO_MASKB_CLEAR	中断屏蔽清除寄存器 B：清除 PF 管脚的中断功能，写 1 则对应管脚关闭中断，写 0 则无效
PORTFIO_MASKB_SET	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
PORTFIO_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
PORTFIO_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
PORTFIO_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
PORTFIO_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
PORTFIO_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

3.13 例子代码分析

接口功能配置：

将 PF0 接口配置为 IO 功能。

```
*pPORTF_FER      &= ~PF0;
```

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。

```
*pPORTFIO_DIR        &= ~PF0;      //设置 PF0 为输入
*pPORTFIO_INEN       |=   PF0; //输入使能
i = *pPORTFIO;                     //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。

```
*pPORTFIO_DIR |= PF0;     //设置 PF0 为输出
*pPORTFIO_SET |= PF0;          //PF0 脚置高
*pPORTFIO_CLEAR |= PF0;            //PF0 脚置低
*pPORTFIO |= PF0;              //PF0 脚置高
*pPORTFIO &= ~PF0;             //PF0 脚置低
```

3.14 代码实现功能

由于 PF0 接口和网卡 MII 接口复用，如使用开发板扩展接口的 PF0 接口测试，需将拨码开关 SW5 和 SW6 全部拨向 OFF，以断开网口对 PF0 接口的影响。

工程 BF51x_GPIO_IN.dpj 实现了读取 PF0 接口状态并打印出 PF 接口状态数据。

工程 BF51x_GPIO_OUT.dpj 实现了通过 PF0 接口不断的输出高低变化的电平。

3.15 测试结果

工程 BF51x_GPIO_IN.dpj：运行代码后将 PF0 接口的电平状态打印在 VDSP 上。

```
Load complete.
PF0 data is 0
PF0 data is 0
PF0 data is 0
PF0 data is 0
PF0 data is 1
PF0 data is 1
PF0 data is 1
PF0 data is 1
PF0 data is 0
PF0 data is 0
PF0 data is 0
PF0 data is 0
PF0 data is 0
PF0 data is 0
```

工程 BF51x_GPIO_OUT.dpj： 运行代码后 PF0 将不断变换高低电平。

3.2 BF51x_GPIO_INTERRUPT

3.21 接口功能介绍

ADSP-BF51x 的 PF，PG，PH 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。

PORTFIO_MASKA 和 PORTFIO_MASKB：用来为 PF 管脚设置中断源，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。

通过 VDSP 的帮助文件找到中断配置表，在表中找到所使用的中断源 Port F interrupt A，从表中可以知道，其默认中断优先等级为 IVG13，配置寄存器是 SIC_IAR5 的 Bit20~Bit23，使用的中断屏蔽寄存器是 SIC_IMASK1，其使能位是 Bit13。

Table 5-4. Peripheral Interrupt Events (Part 2)

Peripheral ID Number	Bit Position for SIC_ISR1, SIC_IMASK1, SIC_IWR1	SIC_IAR7–4	Interrupt Source	Default Mapping
49	Bit 17	SIC_IAR6[7:4]	Reserved	IVG7
48	Bit 16	SIC_IAR6[3:0]	SPI1 status	IVG7
47	Bit 15	SIC_IAR5[31:28]	SPI0 status	IVG7
46	Bit 14	SIC_IAR5[27:24]	Port F interrupt B	IVG13
45	Bit 13	SIC_IAR5[23:20]	Port F interrupt A	IVG13
44	Bit 12	SIC_IAR5[19:16]	Watchdog timer	IVG13
43	Bit 11	SIC_IAR5[15:12]	MDMA1	IVG13
42	Bit 10	SIC_IAR5[11:8]	MDMA0	IVG13
41	Bit 9	SIC_IAR5[7:4]	Port G interrupt B	IVG12

中断优先等级为 IVG13，通过查询下表，获得 IVG13 对应的值是 6，所以 SIC_IAR5 的 Bit20~Bit23 应写入 6。

Table 5-2. IVG Select Definition

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

函数：

register_handler(ik_ivg13, FlagA_ISR);

中断等级注册函数，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 13 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。

3.22 接口寄存器说明

寄存器	功能
SIC_IARx	中断等级设置寄存器
SIC_IMASKx	中断屏蔽寄存器

3.23 例子代码分析

PF 口设置使用外部中断：

```
*pPORTF_FER       &= ~(PF1|PF0);    //配置 PF0 和 PF1 管脚为 IO 功能
*pPORTFIO_DIR     &= ~(PF1|PF0);    //配置 PF0 和 PF1 管脚为输入
*pPORTFIO_INEN    |=  (PF1|PF0);    //配置 PF0 和 PF1 管脚输入使能

*pPORTFIO_EDGE    |= (PF1|PF0) ;    //配置为沿触发模式
*pPORTFIO_POLAR   |= (PF1|PF0);     //配置为下降沿触发
```

```
    *pPORTFIO_MASKA_SET  |= (PF1|PF0);        //中断源采用 Port F interrupt A
```

配置外部中断：

```
    iar5 |= 0x00060000;
    iar5 &= 0xfff6ffff;
    *pSIC_IAR5 = iar5;                        //配置中断等级
    register_handler(ik_ivg13, FlagA_ISR);    //注册中断
    imask1 = *pSIC_IMASK1;
    imask1 |= 0x00002000;
    *pSIC_IMASK1 = imask1;                    //使能中断
```

中断函数：

```
EX_INTERRUPT_HANDLER(FlagA_ISR)       //设置中断函数标志为 FlagA_ISR
{
    if((*pPORTFIO&PF0) == PF0)            //判断中断管脚
    {
        printf("interrupt is PF0!\n");
    }
    else if((*pPORTFIO&PF1) == PF1)       //判断中断管脚
    {
        printf("interrupt is PF1!\n");
    }
    *pPORTFIO_CLEAR = PF1|PF0;            //清除中断
}
```

3.24 代码实现功能

代码实现了通过 PF0 和 PF1 管脚触发外部中断的功能。板卡的扩展接口上引出了 PF0 和 PF1 管脚，但这两个管脚没有上拉电阻，所以必须在外部上拉电阻后，才能使用该代码进行外部中断测试。

PF 管脚和网口复用，所以使用该管脚作为外部中断时，需将板卡上的拨码开关 SW5 和 SW6 全部拨到 OFF，断开与网口芯片的连接。

代码通过 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。

3.25 测试结果

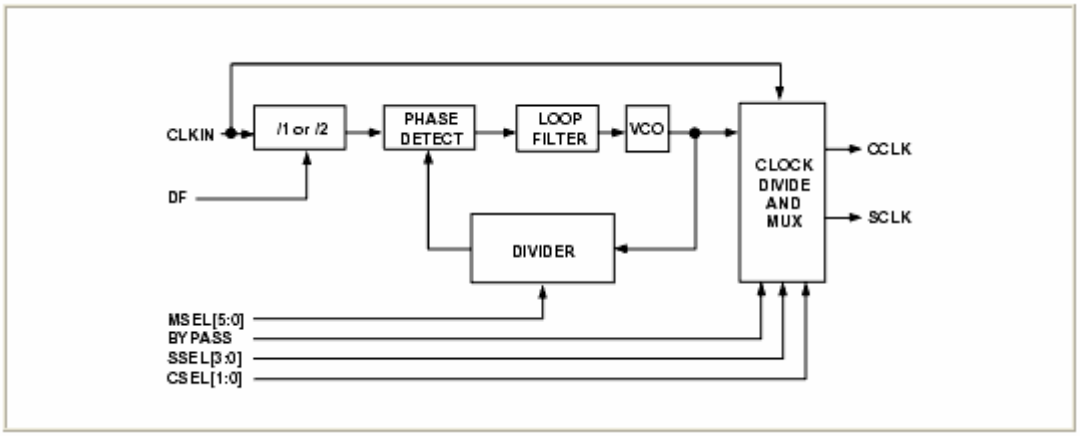
运行代码后，当 PF0 或 PF1 管脚有中断触发，会进入中断函数，判断中断后，打印出中断信息。

```
Load complete.
interrupt is PF0!
interrupt is PF0!
interrupt is PF0!
interrupt is PF0!
interrupt is PF1!
interrupt is PF1!
interrupt is PF1!
interrupt is PF1!
interrupt is PF0!
interrupt is PF1!
```

3.3 BF51x_PLL

3.31 接口功能介绍

PLL(Phase Locked Loop)是 ADSP-BF51x 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-BF51x 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 100MHz。

3.32 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

3.33 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。

3.34 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 4 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 400/4=100MHz。

3.35 测试结果

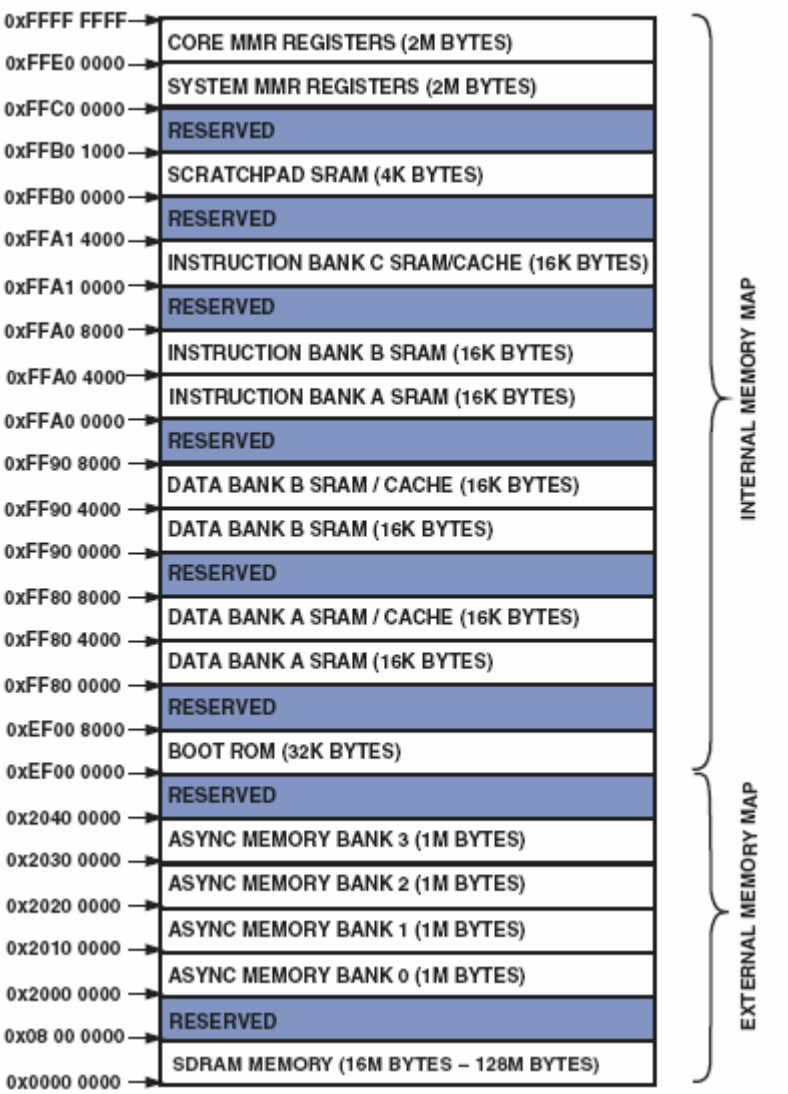
运行代码后，处理器的内核时钟会运行在 400MHz，系统时钟运行在 100MHz。

3.4 BF51x_EBIU

3.41 接口功能介绍

EBIU 接口是 ADSP-BF51x 的外部总线接口，ADSP-BF51x 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-BF51x 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-BF51x 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。

3.42 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

3.43 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
```

3.44 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。

3.45 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
```

3.5 BF51x_SPI

3.51 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-BF51x 上共有 2 个 SPI 接口。支持主机模式和从机模式，在主机模式下，可以通过 SPISEL 接口挂载 SPI 设备，支持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
SPIx_MOSI	主输入从输出接口，根据主机和设备模式确定功能
SPIx_MISO	从输入主输出接口，根据主机和设备模式确定功能
SPIx_SCK	SPI 时钟

SPIx_SELx	SPI 设备选则接口
SPIx_SS	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)

3.52 接口寄存器说明

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

3.53 例子代码分析

```
*pSPI1_BAUD=2;                  //配置速率为 1/4 系统时钟   SPI 速率  = SCLK/2*SPI_BAUD
*pSPI1_FLG |=FLS2;              //选择 SPI1SEL2 接口
*pSPI1_CTL = 0x1001|CPHA| CPOL;  //配置模式为手动片选模式
*pSPI1_CTL = (*pSPI_CTL | SPE);  //使能 SPI1 接口

*pSPI1_FLG &= ~FLG2;            //将 SPI1SEL2 拉到 0
while(!(*pSPI1_STAT & SPIF));   //查看 SPI1 传输状态是否完成
*pSPI1_TDBR = 0x55;              //将数据送入 SPI1 传输数据寄存器
*pSPI1_FLG |= FLG2;              //将 SPI1SEL2 拉到 1，完成数据传输

*pSPI1_FLG &= ~FLG2;
while(*pSPI1_STAT & RXS)        //查看 SPI1 传输状态是否有数据需要接收
i = *pSPI1_RDBR;                //读取数据
*pSPI1_FLG |= FLG2;
```

ADSP-BF51x 的 SPI 接口支持手动片选和自动片选两种模式，通过 SPIx_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。

3.54 代码实现功能

代码实现了采用 SPI1 接口发送 0x55 数据和读取 SPI1 接口数据。
由于没有相关硬件为 SPI1 发送数据，所以代码只是为了学习 SPI1 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。

3.55 测试结果

SPI1 接口发送数据 0x55 后读取 SPI1 接口数据。

3.6 BF51x_Timer

3.61 接口功能介绍

ADSP-BF51x 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）

3.62 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

3.63 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 = 0xfffffff5;                    //配置中断等级数据为 4
register_handler(ik_ivg12, TIMER0_ISR);     //注册中断等级为 12，标识符为 TIMER0_ISR
*pSIC_IMASK1 = 0x00000001;                  //开启中断

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
}
```

3.64 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。

3.65 测试结果

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

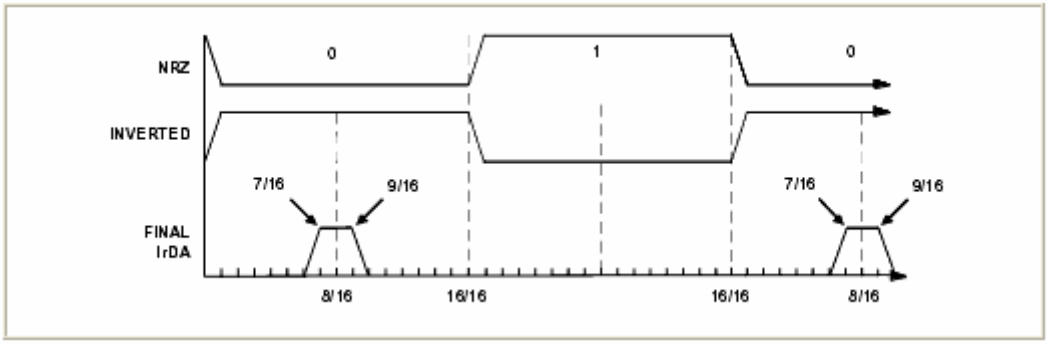
3.7 BF51x_UART

3.71 接口功能介绍

UART（Universal Asynchronous Receiver/Transmitter 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-BF51x 上有两个 UART 接口，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)

3.72 接口寄存器说明

寄存器	功能
UARTx_THR	UART 传输数据寄存器
UARTx_RBR	UART 接收缓存寄存器
UARTx_DLL	UART 波特率配置低 8 位寄存器
UARTx_DLH	UART 波特率配置高 8 位寄存器
UARTx_IER	UART 中断使能寄存器
UARTx_IIR	UART 中断识别寄存器
UARTx_LCR	UART 线路控制寄存器
UARTx_MCR	UART 调制控制寄存器
UARTx_LSR	UART 线路状态寄存器
UARTx_SCR	UART 暂存寄存器
UARTx_GCTL	UART 全局控制寄存器

3.73 例子代码分析

代码请参考板卡驱动代码的 RS232 程序。

```
div = SYSCLK/rate/16;              //计算波特率配置值
*pUART0_GCTL=0x0009;               //配置 UART 工作模式
*pUART0_LCR=0x0080;// DLAB=1  允许访问 DLL  和 DLH
*pUART0_DLL=div;
*pUART0_DLH=div>>8; //DLL DLH 分别赋值
*pUART0_LCR=0x0003;//    允许访问  RBR THR  和 IER
*pUART0_IER=0x0001;//    接收中断允许
```

```
*pSIC_IAR0 = 0xffffffff;
*pSIC_IAR1 = 0xffffffff;                    // UART 中断等级定义
*pSIC_IAR2 = 0xf3ffffff;
register_handler(ik_ivg10, UART0_ISR);     //注册中断
*pSIC_IMASK0 = 0x00400000;                  //使能中断

*pUART0_THR=TXbuf[i];                       //向 UART0 传输数据寄存器写数据
while(!(*pUART_LSR&0x0020));                //等待传输完成

EX_INTERRUPT_HANDLER(UART0_ISR)     //UART 接收数据中断函数
{
    if(*pUART0_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]=*pUART0_RBR;    //读取数据
        cont++;
    }
}
```

3.74 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。

3.75 测试结果

代码实现了通过 UART0 接口发送测试数据，并且通过接收中断函数获取接收到的数据。

3.8 BF51x_SPORT

3.81 接口功能介绍

ADSP-BF51x 上有两个 SPORT 口，SPORT（synchronous serial ports）接口是 ADSP-BF51x 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

3.82 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

3.83 例子代码分析

```
*pSPORT1_TCLKDIV  =    TCLKDIV;              //配置 SPORT 传输接口的时钟频率
```

```
*pSPORT1_TFSDIV     =     TFSDIV;                  //配置 SPORT 传输接口的同步频率
*pSPORT1_TCR1       =     ITFS|TFSR|ITCLK;         //配置 SPORT 传输工作模式
*pSPORT1_TCR2       =     31;                      //配置 SPORT 以 32Bit 数据传输

*pDMA6_PERIPHERAL_MAP = 0x6000;                    //设置 SPORT 传输接口 DMA
*pDMA6_CONFIG = WDSIZE_32 | DI_EN |FLOW_1;        //设置 DMA 工作模式
*pDMA6_START_ADDR = (void *)iTxBuffer;            //设置 DMA  传输数据起始地址
*pDMA6_X_COUNT = 1000;                            //设置 DMA 传输次数
*pDMA6_X_MODIFY = 4;                              //设置 DMA 每次地址增量变化

*pDMA6_CONFIG       = (*pDMA6_CONFIG | DMAEN);    //使能传输 DMA
*pSPORT1_TCR1  = (*pSPORT1_TCR1 | TSPEN);         //使能传输 SPORT

*pSIC_IAR2 = 0xffff32ff;                          //配置 SPORT DMA 中断等级
register_handler(ik_ivg9, Sport1_RX_ISR);         //注册接收中断
register_handler(ik_ivg10, Sport1_TX_ISR);        //注册传输中断
*pSIC_IMASK0 = 0x000C0000;                        //打开 SPORT 传输和接收中断

EX_INTERRUPT_HANDLER(Sport1_TX_ISR)               //传输 DMA 中断函数
{
    *pDMA6_IRQ_STATUS = 0x0001;                   //清除中断标志位
    printf("SPORT TX DMA Done!\n");               //打印信息
    *pSIC_IMASK0 &= ~0x00080000;                  //屏蔽接收中断
}
```

3.84 代码实现功能

代码实现了通过 SPORT1 接口利用 SPORT1 DMA 传输数据和接收数据，SPORT1 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 SPORT1 接口使用 DMA 传输时常用的配置，由于没有和其他设备做通讯，所以看不到接收的实际数据。也可以将扩展接口上 SPORT1 的 DT1PRI 和 DR1PRI 两个接口短接，实现环路测试功能，通过接收数据 Buffer 查看收到的数据。

3.85 测试结果

```
Load complete.
SPORT TX DMA Done!
SPORT RX DMA Done!
```

3.9 BF51x_PPI

3.91 接口功能介绍

PPI（Parallel Peripheral Interface ）接口在 ADSP-BF51x 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-BF51x 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。

PPI 接口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。

3.92 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

3.93 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

*pTIMER0_PERIOD     = 525;  //配置行同步信号产生的周期
*pTIMER0_WIDTH      = 41;   //配置行同步信号宽度
*pTIMER0_CONFIG     = 0x00a9;//配置行同步信号工作模式
```

```
*pTIMER1_PERIOD      = 150150;//配置列同步信号产生的周期
*pTIMER1_WIDTH       = 5250;   //配置列同步信号宽度
*pTIMER1_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|= 0x0003;       //使能行场同步信号
asm("ssync;");                 //系统同步
```

PPI 的行场同步信号与 TIMER0 和 TIMER1 复用，所以要配置 TIMER 寄存器来启动 PPI 的同步信号。

3.94 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。

3.95 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，如需要看结果，可以运行板卡驱动下的液晶屏代码，观察传输的图像数据。

3.10 BF51x_MDMA

3.101 模块功能介绍

MDMA 全称是 memoryDMA ，是内存到内存搬运数据的 DMA。在 DSP 做算法时，经常会遇到数据重组或者搬移，如果用 core 搬运这些数据，是对 DSP 资源的一种浪费，此时就可以用到 MDMA 进行数据搬移。

3.102 接口寄存器说明

寄存器	功能
MDMA_S0_START_ADDR	MDMA 源地址寄存器
MDMA_S0_X_COUNT	MDMA 源地址 X 计数寄存器
MDMA_S0_X_MODIFY	MDMA 源地址 X 修改寄存器
MDMA_S0_PERIPHERAL_MAP	MDMA 源地址通道配置寄存器