

LCDK6748 DSP LAB

Part I

指导材料

杜伟韬 duweitao@cuc.edu.cn
广播电视数字化工程中心 ECDAV

杨刚 gangy@cuc.edu.cn
信息工程学院 电子工程系

中国传媒大学 Communication University of China

本课程实验内容

- Hello World on DSP
- LED 系列实验
 - 点灯实验-直接配置寄存器
 - 点灯实验-使用GEL函数
 - 点灯实验-使用StarterWare库函数
 - 通过按键中断控制走马灯实验-使用StarterWare库函数
- 音频实时采集回放实验：使用中断、I2C控制器、McASP控制器、AIC31 Codec、DMA控制器和StarterWare库函数
- 存储器布局和访存性能测试实验：使用片内RAM，片外DDR、malloc函数、定时器，观察MAP文件，本实验基于音频采集回放实验
- 使用FIR滤波器的实时音频均衡器实验，本实验基于音频采集回放实验
- DSP/BIOS: Hello World 和系统时间打印实验
- DSP/BIOS: 线程任务（TSK）和信号量（SEM）实验
- DSP/BIOS: 按键中断和点灯实验：使用BIOS中断调度器
- DSP/BIOS: 音频实时采集、回放实验：使用BIOS中断调度器

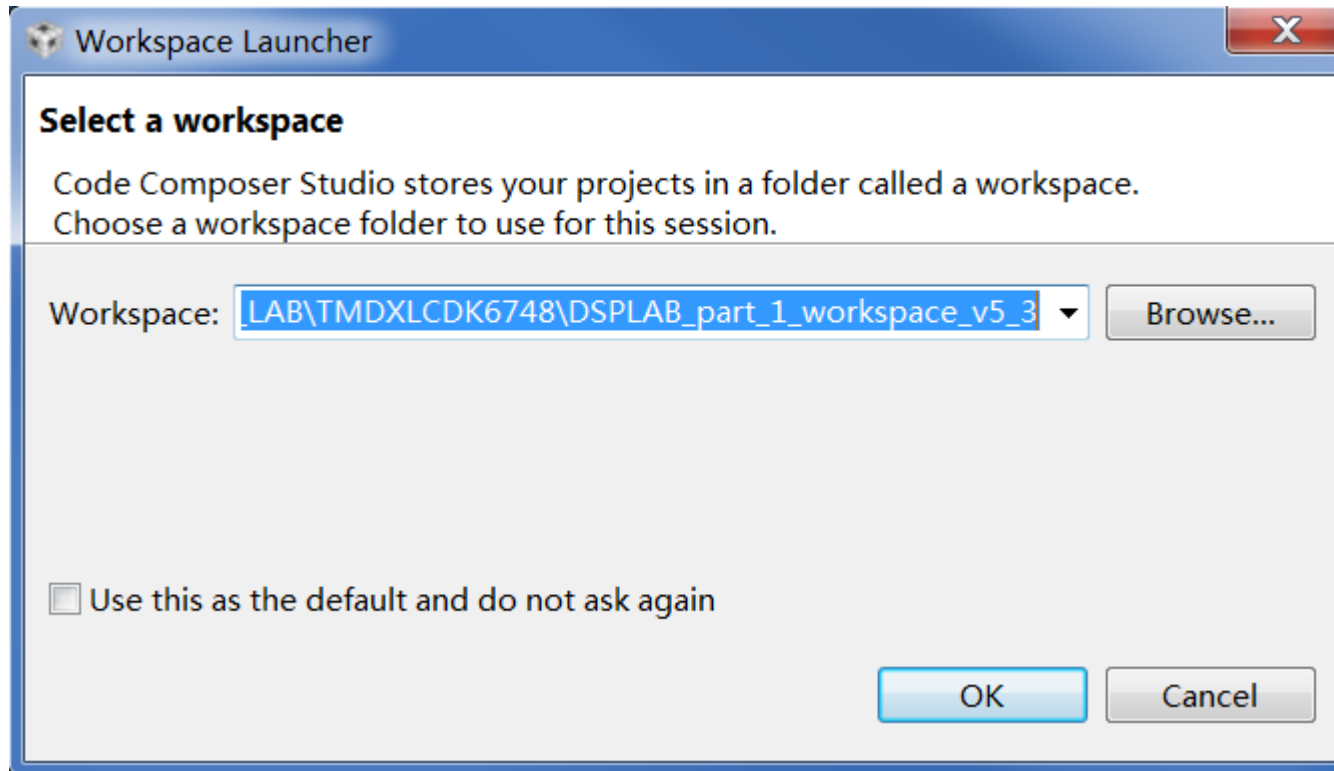


上机操作

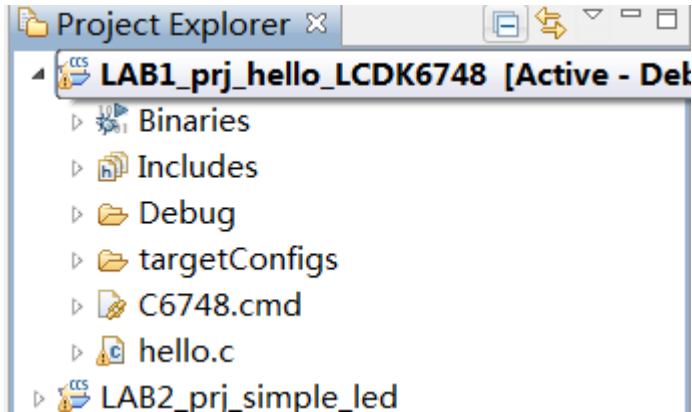
LAB1

Hello World

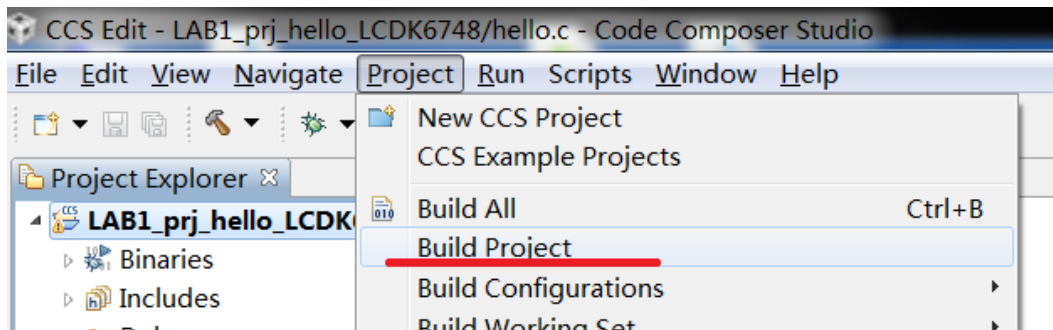
选择实验项目所在的workspace



选中LAB1项目并且编译

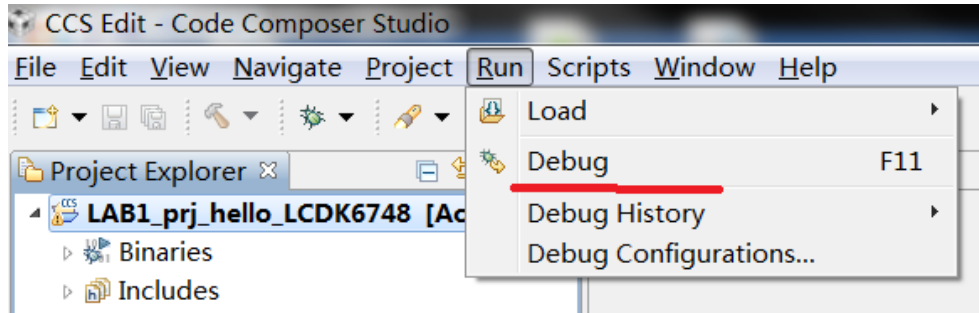


- 双击左侧导航条选中项目



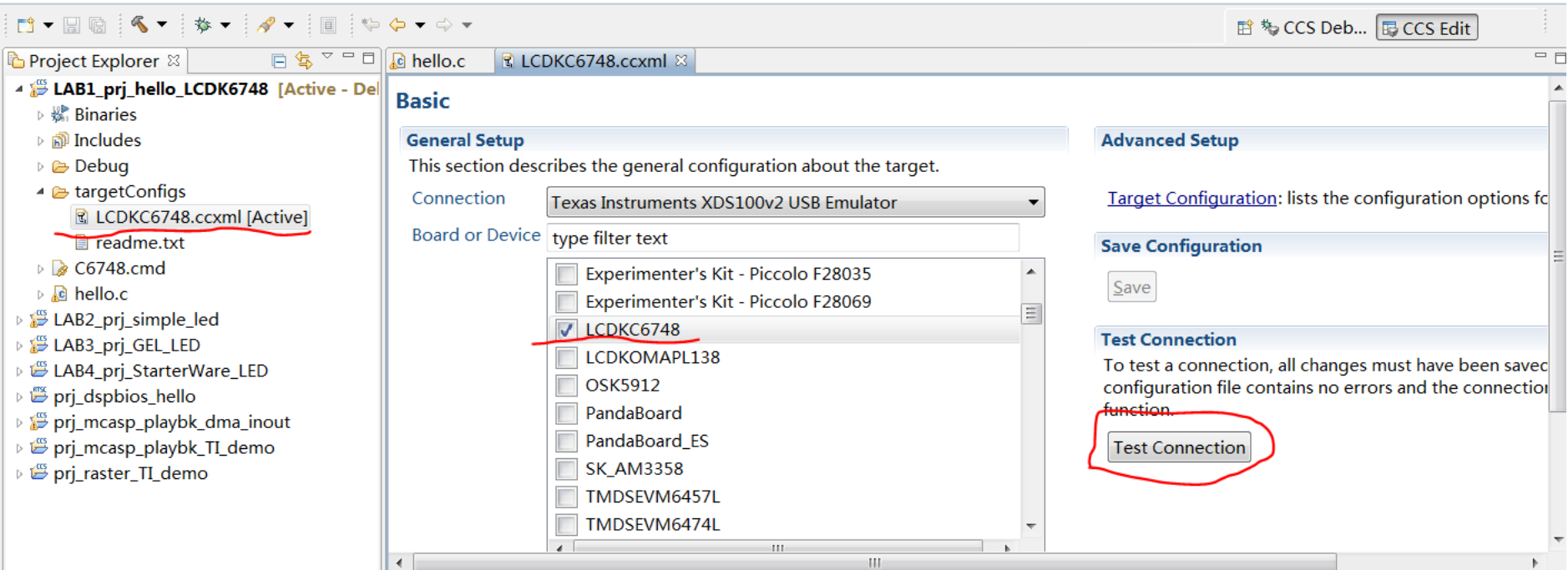
- 可以从菜单项目中选择编译。

连接仿真器到LCDK并下载代码



- 从菜单进入调试模式，会下载代码

- 如果不顺利，按下图检查仿真器连接



仿真器测试成功内容大致如下

- This test will use blocks of 512 32-bit words.
- This test will be applied just once.

- Do a test using 0xFFFFFFFF.
- Scan tests: 1, skipped: 0, failed: 0
- Do a test using 0x00000000.
- Scan tests: 2, skipped: 0, failed: 0
- Do a test using 0xFE03E0E2.
- Scan tests: 3, skipped: 0, failed: 0
- Do a test using 0x01FC1F1D.
- Scan tests: 4, skipped: 0, failed: 0
- Do a test using 0x5533CCAA.
- Scan tests: 5, skipped: 0, failed: 0
- Do a test using 0xAACC3355.
- Scan tests: 6, skipped: 0, failed: 0
- All of the values were scanned correctly.

- The JTAG DR Integrity scan-test has succeeded.

- [End]

- 注意，如果已经开启了某个项目的调试模式，则仿真器的连接通道就被占用，连接测试会失败。

检查启动GEL脚本

- 按照下图检查启动的GEL脚本配置
- 该脚本代码用于初始化DSP的寄存器配置

The screenshot displays the 'Target Configuration' window in a software development environment. On the left, the 'All Connections' tree shows a hierarchy: Texas Instruments XDS100v2 USE, LCDKC6748, C6748_0, ICEPICK_C, Subpath_0, and C674X_0. The 'C674X_0' node is highlighted with a red circle. To the right, the 'Cpu Properties' panel is open, showing the 'initialization script' field with the path '..\..\C6748_LCDK.gel' circled in red. Below the path is a 'Browse...' button. At the bottom of the window, the 'Advanced' tab is selected and circled in red.

Target Configuration

All Connections

- Texas Instruments XDS100v2 USE
 - LCDKC6748
 - C6748_0
 - ICEPICK_C
 - Subpath_0
 - C674X_0**

Cpu Properties

Set the properties of the selected cpu.



Bypass

initialization script **..\..\C6748_LCDK.gel**

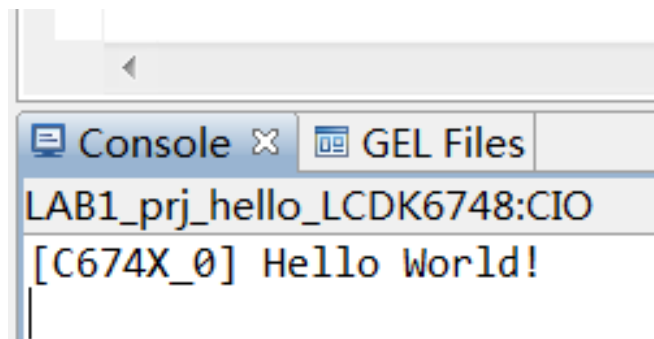
Slave Processor

Basic **Advanced** Source

运行代码

- 运行代码  暂停运行 

- 观察终端窗口的输出



```
Console x GEL Files
LAB1_prj_hello_LCDK6748:CIO
[C674X_0] Hello World!
```

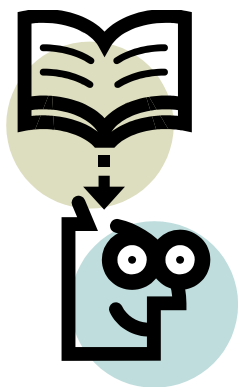
- 注意：printf函数的实时性不好，不要在实时运行的代码中使用printf

LAB1 作业

- 修改代码
- 在打印的内容里面加上你的英文名字。

LAB2

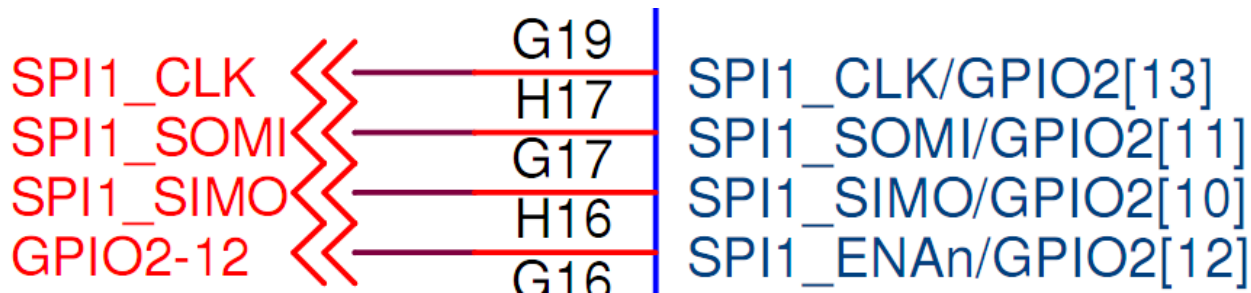
Simple LED



背景知识

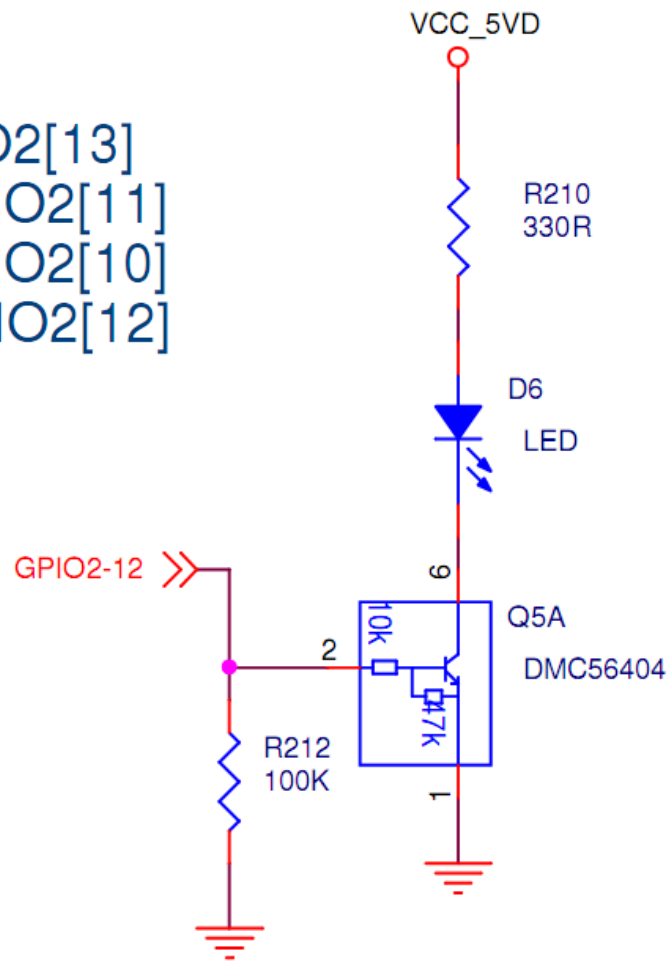
LCDK电路板上的LED

OMAP-L138/C6748 LC Dev Kit Schematic



GPIO2-12 is the pin driving the LED D6

- 首先从电路图中寻找LED和DSP GPIO的对应关系



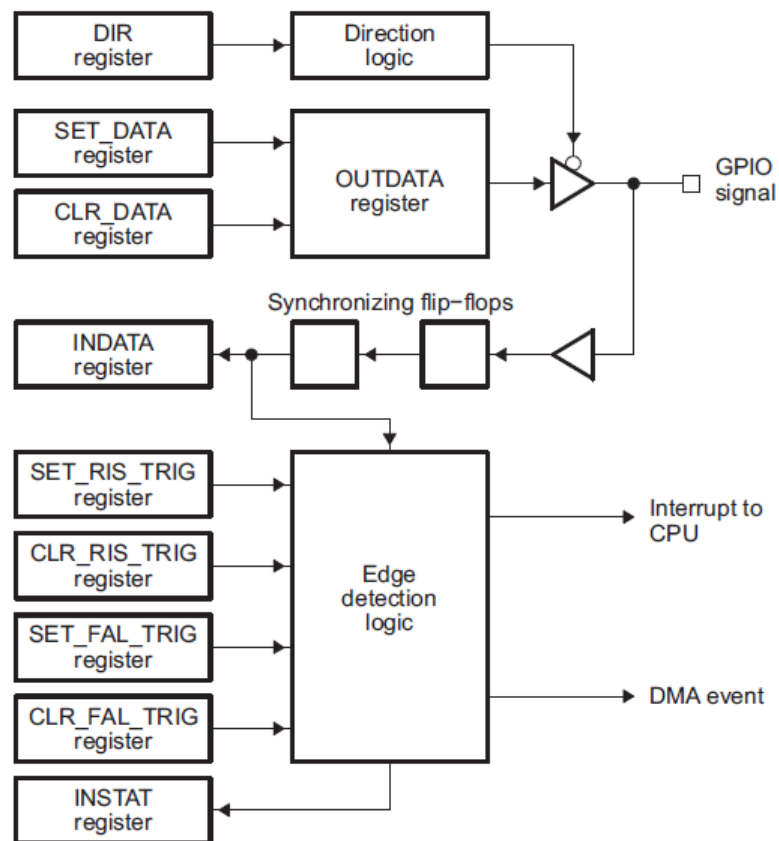
启用 GPIO 功能块

- 启动GPIO所属的PowerDomain供电
- GPIO管脚和其他功能管脚是复用的
- 例如GPIO管脚也可能是个SPI管脚
- 另外，GPIO也可以是输入或输出管脚
- 通过配置寄存器选择GPIO功能

SPI1_ENAn/GPIO2[12]

- First of all, set the pin multiplexing, enable the GPIO function
- Configure the Direction as 'output'
- Set the output data register value

Figure 19-1. GPIO Block Diagram



GPIO 的 Memory Map

From 6748 CPU data sheet

5.32 General-Purpose Input/Output (GPIO)

The GPIO pins are grouped into banks of 16 pins per bank (i.e., bank 0 consists of GPIO [0:15]).

Table 2-29. General Purpose Input Output Terminal Functions (continued)

SIGNAL		NO.	TYPE ⁽¹⁾	PULL ⁽²⁾	POWER GROUP ⁽³⁾
NAME					
GP2					
SPI1_SCS[1] / EPWM1A / PRU0_R30[8] / GP2[15] / TM64P2_IN12		F18	I/O	CP[14]	A
SPI1_SCS[0] / EPWM1B / PRU0_R30[7] / GP2[14] / TM64P3_IN12		E19	I/O	CP[14]	A
SPI1_CLK / GP2[13]		G19	I/O	CP[15]	A
SPI1_ENA / GP2[12]		H16	I/O	CP[15]	A

5.32.1 GPIO Register Description(s)

Table 5-134. GPIO Registers

BYTE ADDRESS	ACRONYM	REGISTER DESCRIPTION
GPIO Banks 2 and 3		
0x01E2 6038	DIR23	GPIO Banks 2 and 3 Direction Register
0x01E2 603C	OUT_DATA23	GPIO Banks 2 and 3 Output Data Register
0x01E2 6040	SET_DATA23	GPIO Banks 2 and 3 Set Data Register
0x01E2 6044	CLR_DATA23	GPIO Banks 2 and 3 Clear Data Register
0x01E2 6048	IN_DATA23	GPIO Banks 2 and 3 Input Data Register
0x01E2 604C	SET_RIS_TRIG23	GPIO Banks 2 and 3 Set Rising Edge Interrupt Register
0x01E2 6050	CLR_RIS_TRIG23	GPIO Banks 2 and 3 Clear Rising Edge Interrupt Register
0x01E2 6054	SET_FAL_TRIG23	GPIO Banks 2 and 3 Set Falling Edge Interrupt Register
0x01E2 6058	CLR_FAL_TRIG23	GPIO Banks 2 and 3 Clear Falling Edge Interrupt Register
0x01E2 605C	INTSTAT23	GPIO Banks 2 and 3 Interrupt Status Register

寄存器定义

TMS320C6748 DSP Technical Reference Manual SPRUH79A Chapter 19 General-Purpose Input/Output (GPIO)

Table 10-3. System Configuration Module 0 (SYSCFG0) Registers

Address	Acronym	Register Description	Access	Section
01C1 4134h	PINMUX5	Pin Multiplexing Control 5 Register	Privileged mode	Section 10.5.9.6

Table 10-26. Pin Multiplexing Control 5 Register (PINMUX5) Field Descriptions

Bit	Field	Value	Description	Type ⁽¹⁾
15-12	PINMUX5_15_12	0	SPIT_ENA/GP2[12] Control Pin is 3-stated.	Z
		1h	Selects Function SPIT_ENA	I/O
		2h-7h	Reserved	X
		8h	Selects Function GP2[12]	I/O
		9h-Fh	Reserved	X

Figure 19-5. GPIO Banks 2 and 3 Direction Register (DIR23)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
GP3P15	GP3P14	GP3P13	GP3P12	GP3P11	GP3P10	GP3P9	GP3P8	GP3P7	GP3P6	GP3P5	GP3P4	GP3P3	GP3P2	GP3P1	GP3P0
R/W-1															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
GP2P15	GP2P14	GP2P13	GP2P12	GP2P11	GP2P10	GP2P9	GP2P8	GP2P7	GP2P6	GP2P5	GP2P4	GP2P3	GP2P2	GP2P1	GP2P0
R/W-1															

LEGEND: R/W = Read/Write; -n = value after reset

Figure 19-10. GPIO Banks 2 and 3 Output Data Register (OUT_DATA23)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
GP3P15	GP3P14	GP3P13	GP3P12	GP3P11	GP3P10	GP3P9	GP3P8	GP3P7	GP3P6	GP3P5	GP3P4	GP3P3	GP3P2	GP3P1	GP3P0
R/W-0															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
GP2P15	GP2P14	GP2P13	GP2P12	GP2P11	GP2P10	GP2P9	GP2P8	GP2P7	GP2P6	GP2P5	GP2P4	GP2P3	GP2P2	GP2P1	GP2P0
R/W-0															

LEGEND: R/W = Read/Write; -n = value after reset

LCDK 4个LED对应的GPIO

说明：以电路板上的LED D4 为例，它属于GPIO6的比特13，需要使用PINMUX13寄存器的 11~8比特段进行管脚复用配置，该比特段配置为0x08后启用GPIO功能，GPIO的方向在DIR67寄存器的比特13进行设置，设为0表示输出，该管脚的输出值通过OUT_DATA67寄存器的比特13进行设置

LED No.	GPIO No.-bit	PIN MUX No.-bit field	PINMUX bit field value	direction config	DIR bit field value	output
D4	GP6-13	PINMUX13_11_8	8h Selects Function GP6[13]	DIR67_13	output bit val = 0	OUT_DATA67_13
D5	GP6-12	PINMUX13_15_12	8h Selects Function GP6[12]	DIR67_12	output bit val = 0	OUT_DATA67_12
D6	GP2-12	PINMUX5_15_12	Selects Function GP2[12]	DIR23_12	output bit val = 0	OUT_DATA23_12
D7	GP0-9	PINMUX0_27_24	8h Selects Function GP0[9]	DIR01_9	output bit val = 0	OUT_DATA01_9

The following is the REG address define in file "C6748_LCDK.gel"

You may need to add some more address definitions.

```
#define GPIO_REG_BASE          (0x01E26000)
#define GPIO_BANK_OFFSET      (0x28)
#define GPIO_DAT_OFFSET       (0x04)
#define GPIO_BANK01_BASE      (GPIO_REG_BASE + 0x10)
#define GPIO_BANK01_DIR        *(unsigned int*)(GPIO_BANK01_BASE)
#define GPIO_BANK01_DAT        *(unsigned int*)(GPIO_BANK01_BASE + GPIO_DAT_OFFSET)
```



上机操作

切换active project 至LAB2

- 项目 LAB2_prj_simple_led
- 编译并Debug该项目
- 在目标板上运行代码
- 观察LED—D6的状态

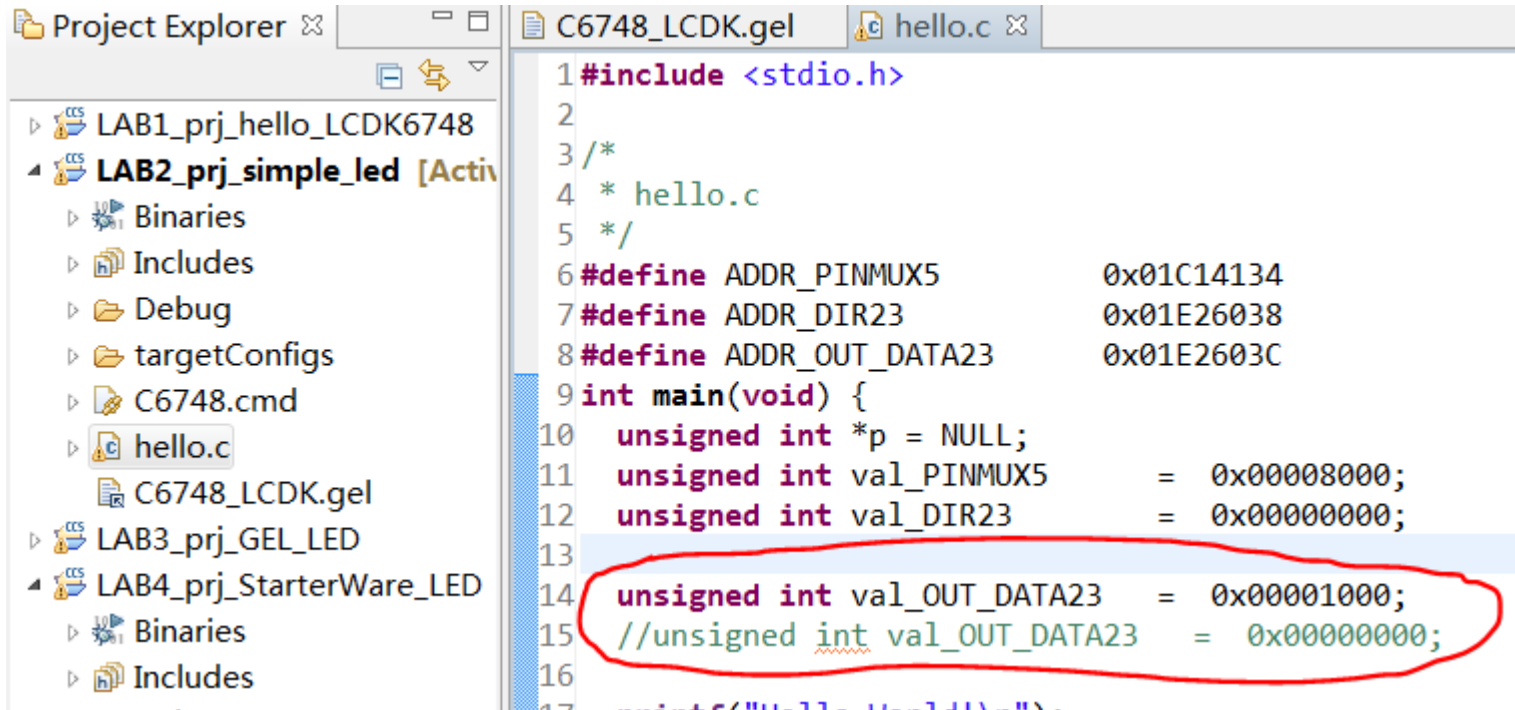
```
p = (unsigned int *)ADDR_PINMUX5 ;  
*p = val_PINMUX5;
```

代码说明：

第一行代码的含义为：把ADDR_PINMUX5 这个宏所表示的整数转换为一个指针类型，并且给指针p赋值。

第二行代码的含义为：指针p所指向的地址，其内容赋值为宏定义val_PINMUX5所表示的值。

修改代码并重新运行



```
1 #include <stdio.h>
2
3 /*
4  * hello.c
5  */
6 #define ADDR_PINMUX5          0x01C14134
7 #define ADDR_DIR23           0x01E26038
8 #define ADDR_OUT_DATA23      0x01E2603C
9 int main(void) {
10     unsigned int *p = NULL;
11     unsigned int val_PINMUX5    = 0x00008000;
12     unsigned int val_DIR23      = 0x00000000;
13
14     unsigned int val_OUT_DATA23 = 0x00001000;
15     //unsigned int val_OUT_DATA23 = 0x00000000;
16
17     printf("val_OUT_DATA23 = %x\n", val_OUT_DATA23);
18 }
```

- 在文件hello.c中，红线圈出的代码用于控制LED D6的亮灭，请尝试不同的值对于LED状态的影响

使用调试变量观察窗口设定LED值

- 如下两图所示，设定不同的变量值并运行，观察LED状态
- 注意，变量的修改和观察，必须在CPU暂停的状态下进行
- 变量值的重新生效必须要重新运行处理器

The image displays two screenshots of the Code Composer Studio (CCS) Debug console, illustrating how to set a variable value during a debug session.

Top Screenshot: The Debug console shows the variable `val_OUT_DATA23` with a value of `4096`. A red circle highlights the value field, and a red line is drawn under the variable name.

Expression	Type	Value
<code>val_OUT_DATA23</code>	unsigned int	4096
+ Add new expression		

Bottom Screenshot: The Debug console shows the same variable `val_OUT_DATA23` with a value of `0`.

Expression	Type	Value
<code>val_OUT_DATA23</code>	unsigned int	0
+ Add new expression		

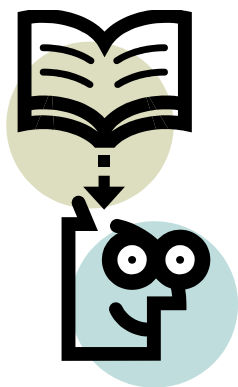
LAB2 作业

- 修改代码，让LED D6重复亮灭
- 提示1，需要重复切换对应的控制数据。
- 提示2，在切换数据的间隔中，需要添加延时函数。
- 延时函数通常使用多次循环来实现，比如

```
void delay(int n){  
    int i=0;  
    for(i=0; i<n; i++){  
        // do something  
    }  
}
```

LAB3

GEL LED



背景知识

GEL的用途

- 什么是GEL?

- Code Composer Studio Help > Tasks > The General Extension Language (GEL)
- 一种脚本语言，和C语言的语法类似
- 使用GEL语法，编写GEL函数，然后加载到CCS IDE

- 为什么用 GEL

- GEL 可以用来完成自动化测试以及定制workspace的工作环境
- 其最方便之处在于，不需要编译，可以直接完成目标处理器的寄存器配置（对于配置外设寄存器非常有用）
- 把GEL代码转换为C代码非常容易

- 如何开始 使用GEL

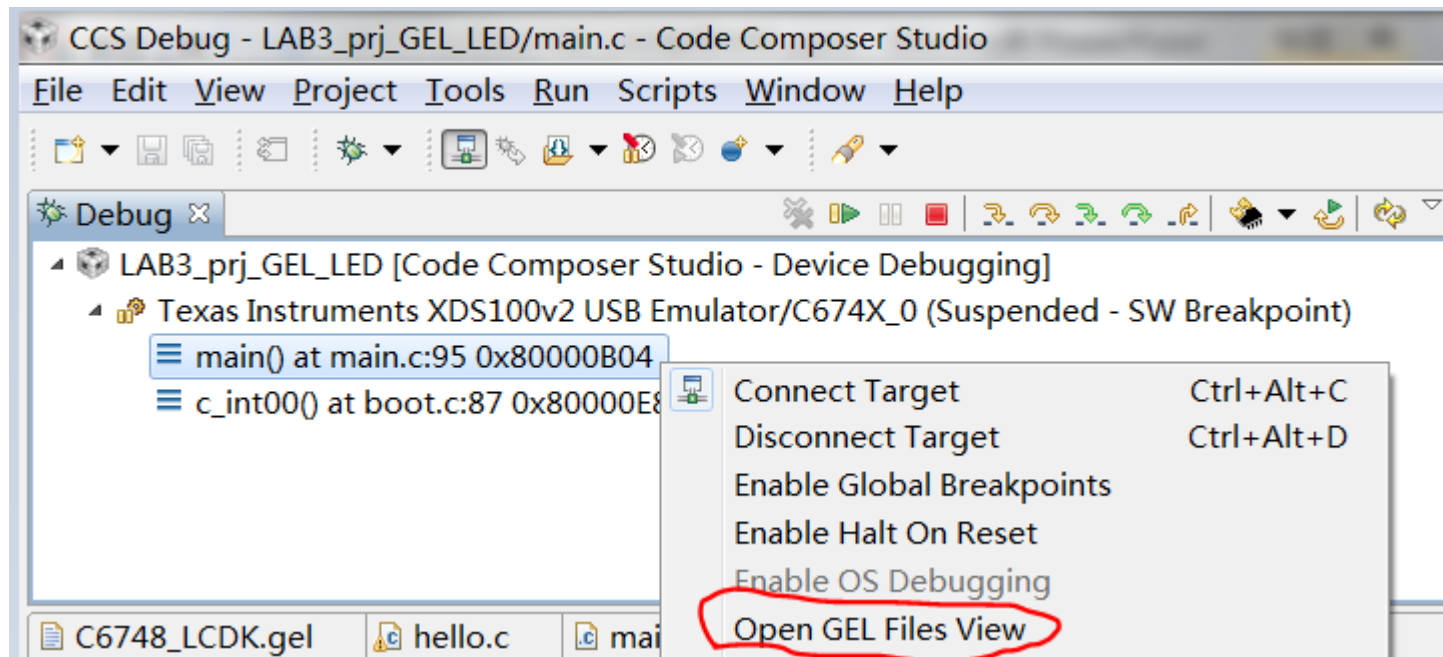
- 从TI的参考代码修改
- 本实验采用的GEL代码修改自“C6748_LCDK.gel”



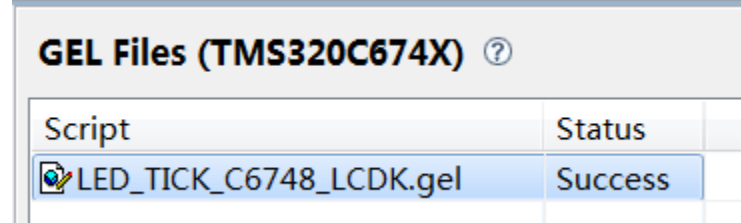
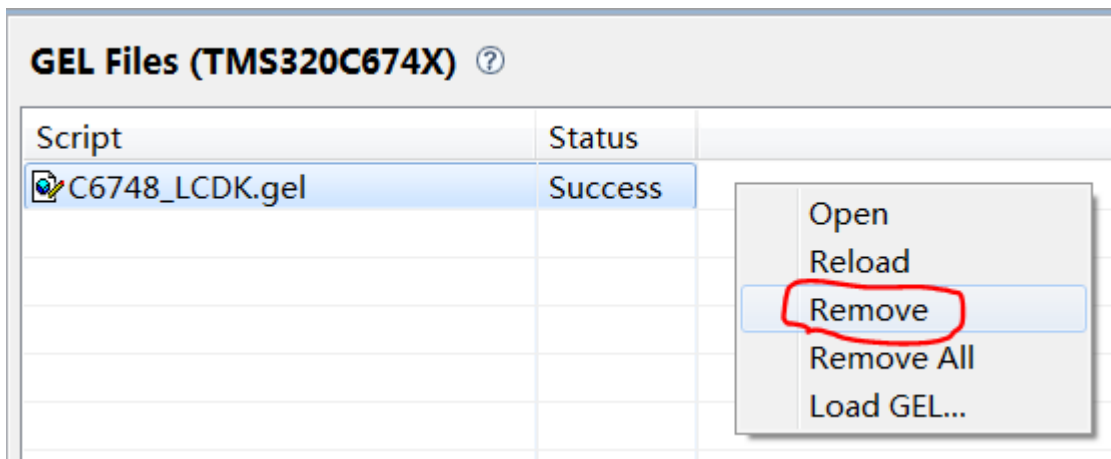
上机操作

切换active project 至LAB3

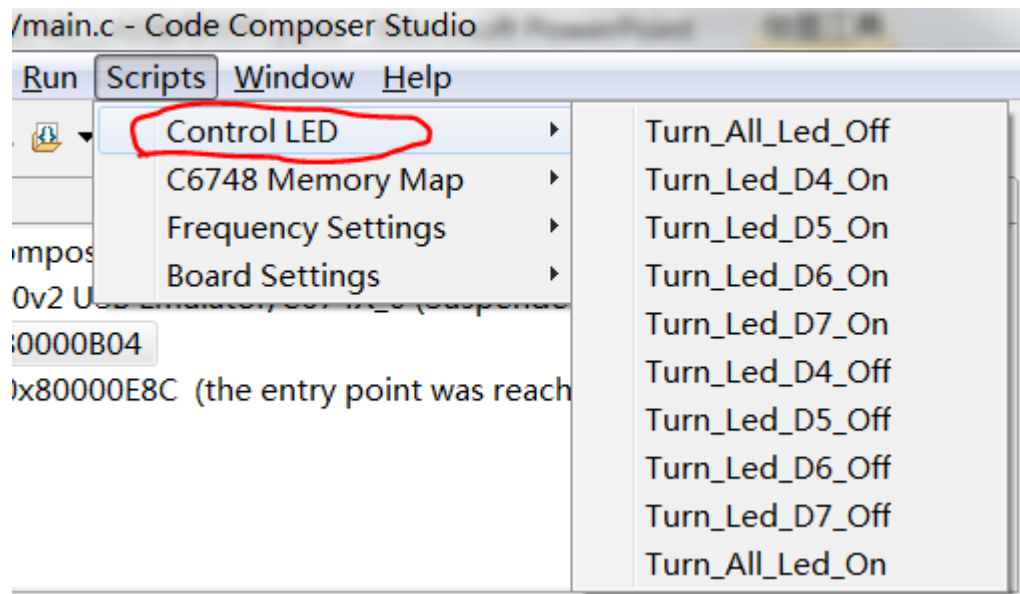
- 项目LAB3_prj_GEL_LED
- 编译并Debug该项目
- 在调试模式下，用鼠标右键，打开GEL窗口



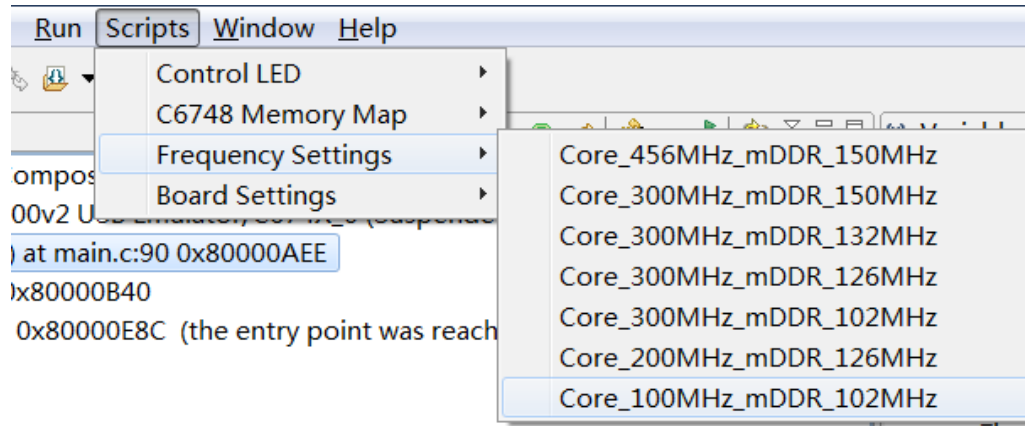
去除原始的GEL，加载新的



- 在CCS菜单中，Scripts菜单选项中，多出“控制LED”的子菜单。
- 请点击不同选项观察结果
- 运行编译后的代码



更改处理器的运行主频



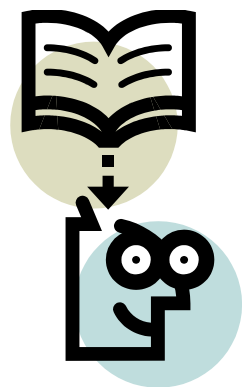
- 在CCS菜单中，Scripts菜单选项中，
- 在Frequency Settings 子菜单中，选择Core 100MHz的选项
- 运行代码
- 把主频换成456MHz
- 再次运行代码，对比LED的闪烁速度
- 由此，可以看到GEL的方便之处，不需编译就修改处理器主频，并且GEL的功能不仅限于修改主频，还可以初始化片外的DDR寄存器

LAB3 作业

- 观察一下GEL代码中，控制LED的部分。
- 再观察一下C代码中控制LED的代码
- 仿照这些代码，编写如下两个LED控制函数
- `LedOddOnEvenOff ()` ，
 - 奇数号码的LED点亮，偶数号码的LED熄灭
- `LedOddOffEvenOn ()`
 - 奇数号码的LED熄灭，偶数号码的LED点亮
- 在C代码的点灯循环中重复运行你编写的函数

LAB4

StartWare LED



背景知识

关于 Starter Ware

- 什么是 Starter Ware

- 用在TI处理器上的一个开源的驱动函数包，应用于没有操作系统的场景
- 包含了一系列用来控制外设的硬件设备抽象层代码 Device Abstraction Layer (DAL) 以及相应的例程
- 早先的处理器平台上对应的函数叫做 CSL (Chip Support Library)

- 为什么使用 Starter Ware

- 使用其预先定义的宏和函数代码进行寄存器配置
- 提高代码在不同处理器平台的重用性，前提是这些处理器都有 Starter Ware库
- 减少自己查阅手册设定寄存器地址和数值的工作量和出错几率



上机操作

切换active project 至LAB4

- 项目 LAB4_prj_StarterWare_LED
- 编译并Debug该项目
- 运行代码，观察LED闪烁情况。

LAB4 作业

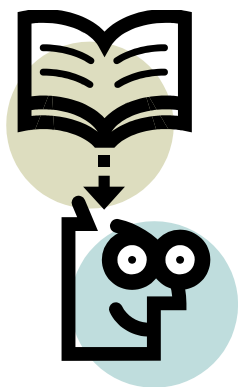
- 观察一下C代码中控制LED的代码，其中的LED写入函数如下：

`GPIOPinWrite(SOC_GPIO_0_REGS, 管脚编号, 0或1数值);`

- 把LAB3的作业中完成的LED闪烁效果，用上述函数来完成。

LAB5

按键中断和LED走马灯



背景知识

DSP的中断选择

Figure 7-1. C674x Megamodule Interrupt Controller Block Diagram

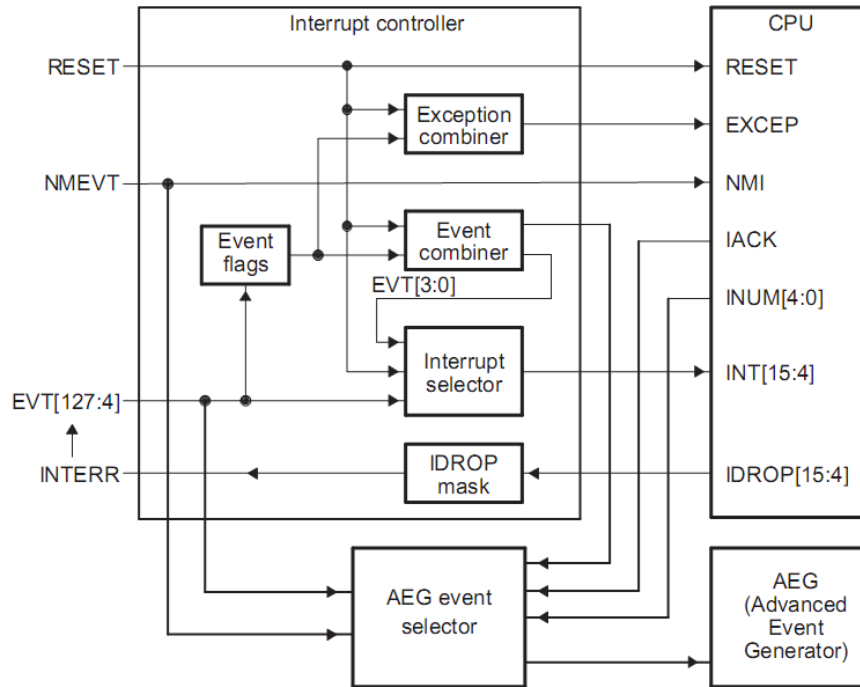
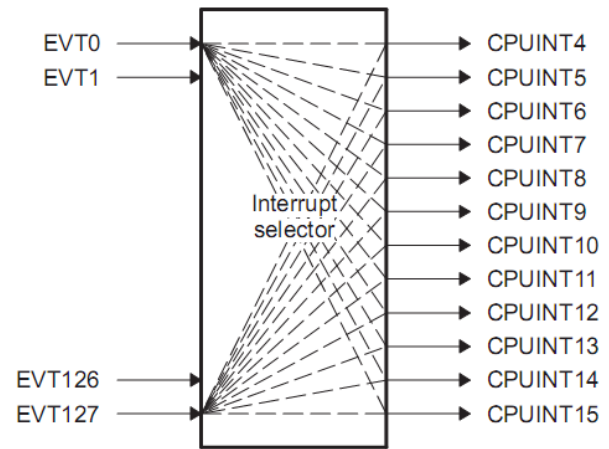


Figure 7-8. Interrupt Selector Block Diagram



TMS320C674x DSP Megamodule
SPRUFK5A – August 2010

- DSP有15个中断信号，同时还有128个中断事件
- 可以通过软件配置某个中断事件触发某个中断信号，对CPU进行中断
- 例如，对于CPUINT4这个中断，可以配置是由按键引起中断，或是由以太网控制器引发中断。

DSP的中断事件

Table 5-6. C6748 DSP Interrupts

EVT#	Interrupt Name	Source
0	EVT0	C674x Int Ctl 0
1	EVT1	C674x Int Ctl 1
2	EVT2	C674x Int Ctl 2
3	EVT3	C674x Int Ctl 3
4	T64P0_TINT12	Timer64P0 - TINT12
5	SYSCFG_CHIPINT2	SYSCFG CHIPSIG Register
6	PRU_EVTOUT0	PRUSS Interrupt
7	EHRPWM0	HiResTimer/PWM0 Interrupt
8	EDMA3_0_CC0_INT1	EDMA3_0 Channel Controller 0 Shadow Region 1 Transfer Completion Interrupt
46	UART_INT1	UART1
47	ECAP1	ECAP1
48	T64P1_TINT34	Timer64P1 Interrupt 34
49	GPIO_B2INT	GPIO Bank 2 Interrupt
50	PRU_EVTOUT7	PRUSS Interrupt
51	ECAP2	ECAP2
52	GPIO_B3INT	GPIO Bank 3 Interrupt

- 请在图中找到GPIO_B2INT，说出其对应的事件编号
- 该表格来自 SPRS590D TMS320C6748 DATASHEET

DSP中断服务程序执行过程

- 每个中断信号产生时，处理器会跳转到固定的地址执行指令。
- 每个中断号拥有0x20字节的指令空间，称为中断服务指令包
- 所有中断的服务指令包称为中断服务表
- 如果中断服务函数代码尺寸超过了取指包的尺寸，则跳转到其他的代码空间执行服务程序。
- 以上过程中的软件工作可以使用StarterWare的库函数完成。

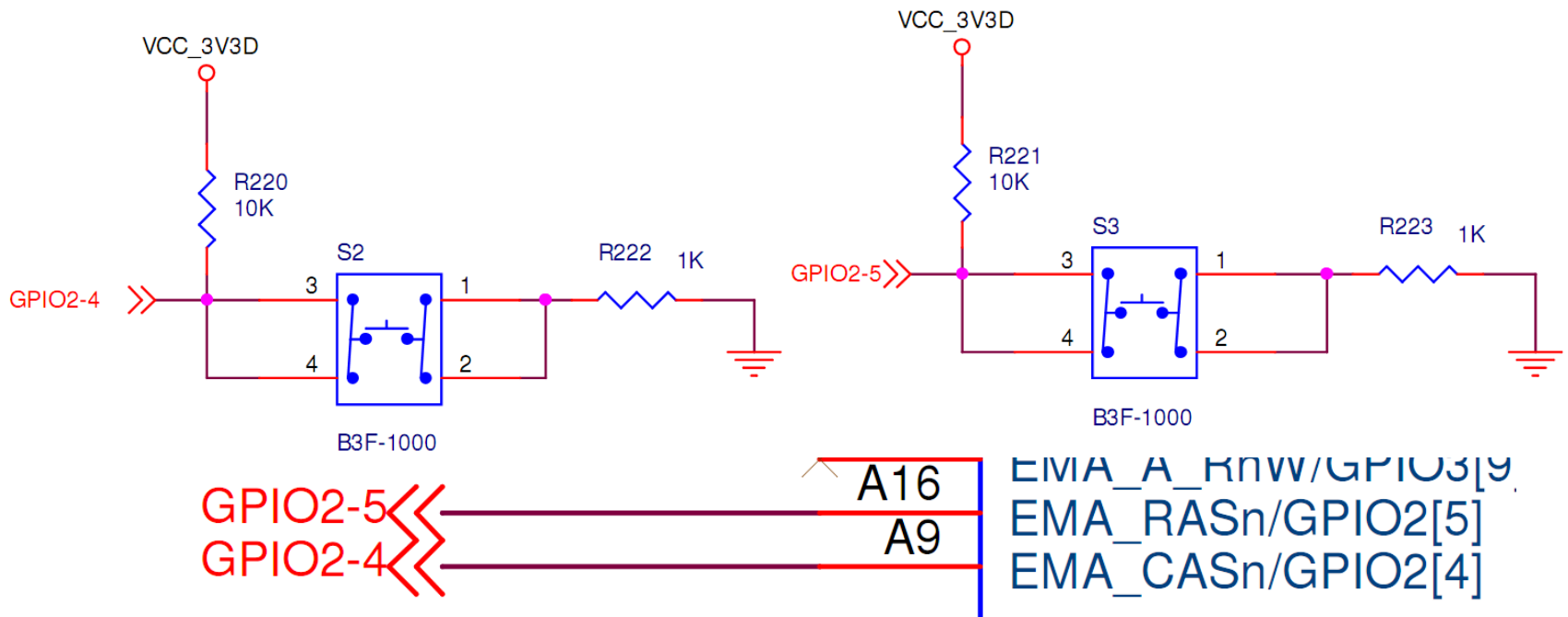
Figure 5-1. Interrupt Service Table

xxxx 000h	RESET ISFP
xxxx 020h	NMI ISFP
xxxx 040h	Reserved
xxxx 060h	Reserved
xxxx 080h	INT4 ISFP
xxxx 0A0h	INT5 ISFP
xxxx 0C0h	INT6 ISFP
xxxx 0E0h	INT7 ISFP
xxxx 100h	INT8 ISFP
xxxx 120h	INT9 ISFP
xxxx 140h	INT10 ISFP
xxxx 160h	INT11 ISFP
xxxx 180h	INT12 ISFP
xxxx 1A0h	INT13 ISFP
xxxx 1C0h	INT14 ISFP
xxxx 1E0h	INT15 ISFP

Program memory

TMS320C674x DSP
CPU and Instruction Set
SPRUFE8B

GPIO 电路连接



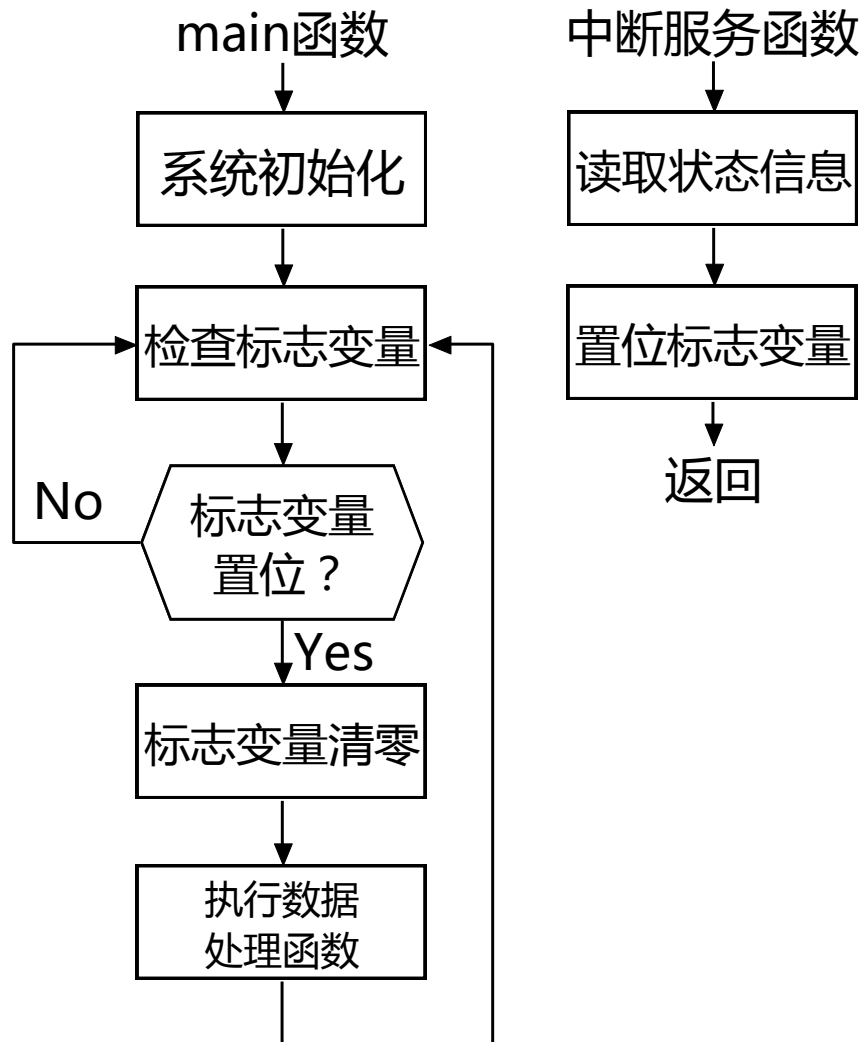
- GPIO 有 9 个 bank，每个 bank 有 16 个管脚
- 管脚编号计算方法 bank 编号 * 16 + bank 内管脚序号 + 1
- GPIO2-4 ~ PIN37，GPIO2-5 ~ PIN38

GPIO的中断设置

- 开启相关的Power Domain
- 设定对应的管脚复用（选择GPIO功能）
- 设定对应的GPIO管脚信号方向为“输入”
- 设定GPIO管脚中断触发类型（上跳、下跳、双向）
- 开启GPIO管脚所属的Bank的中断使能
- 开启全局中断使能
- 注册GPIO的中断服务函数到对应的中断信号
- 映射GPIO Bank 中断事件到对应的中断信号
- 开启对应中断信号的使能

带中断的系统软件流程

- Tip 1. 尽量不要编写庞大的中断服务函数 (ISR) 代码。
- Tip 2. 尽量只在ISR代码里面执行状态查询和标志变量置位的工作。
- Tip 3. 可以在ISR函数内设置断点协助调试。
- Tip 4. 如果有多个中断源，则需要主循环中安排状态变量的查询和任务处理的优先策略，例如优先处理高优先级任务，或是优先处理轻负荷任务。

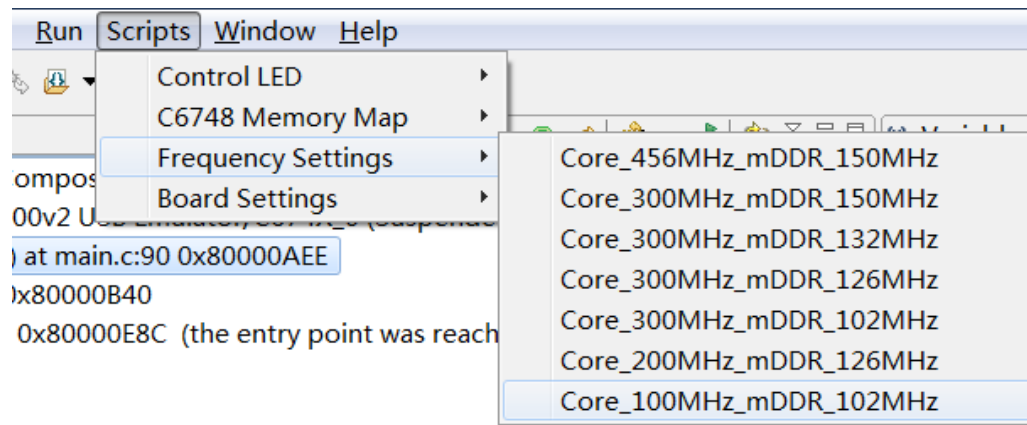




上机操作

切换active project 至LAB5

- 项目 LAB5_prj_GPIO_HWI_StarterWare
- 编译并Debug该项目
- 运行代码，观察LED闪烁情况。
- 按下按键 USER1（图中红圈）
- 观察LED闪烁情况的变化
- 更改处理器主频再次观察按键前后的LED闪烁



LAB5原理

- 本LAB使用了GPIO的双边沿探测中断模式
- 按下按键后产生一次中断
- 按键抬起后产生一次中断
- 通过读取GPIO输入的电平值判断按键状态
- 根据按键状态值调整延时的大小，达到设置不同的闪烁频率

LAB5作业

- 请修改代码完成以下功能
- 作业1：
 - 按下按键后，LED走马灯闪烁频率降低
 - 松开按键后，LED走马灯闪烁频率升高
- 作业2：
 - 在作业1的基础上，按下按键的次数越多，频率降低的越显著
 - 例如，第一次按下按键，闪烁频率是松开按键的 $1/2$ ，第二次按下按键后变成 $1/4$ 等等
 - 具体情况可以自行设置

Thank You
&
Happy DSP Life