嵌入式系统原理及应用

实验指导书

（2016 基础篇）

意见反馈：hyx@nuaa.edu.cn

目录

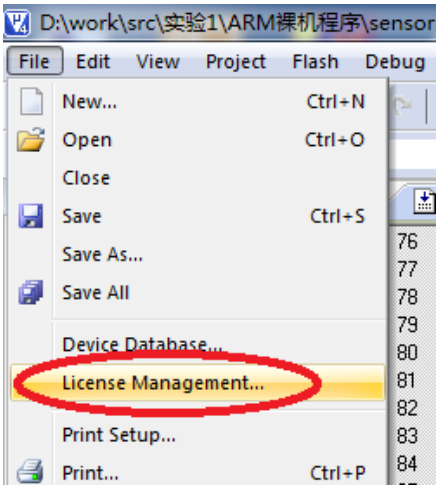
实验1 环境配置-prj

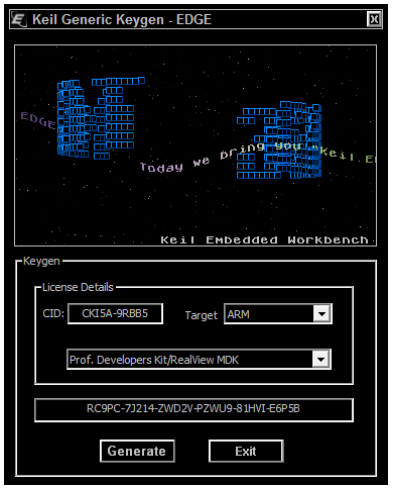
熟悉 KEIL 下工程建立、编写、编译，并使用 Proteus 模拟硬件环境进行调试。

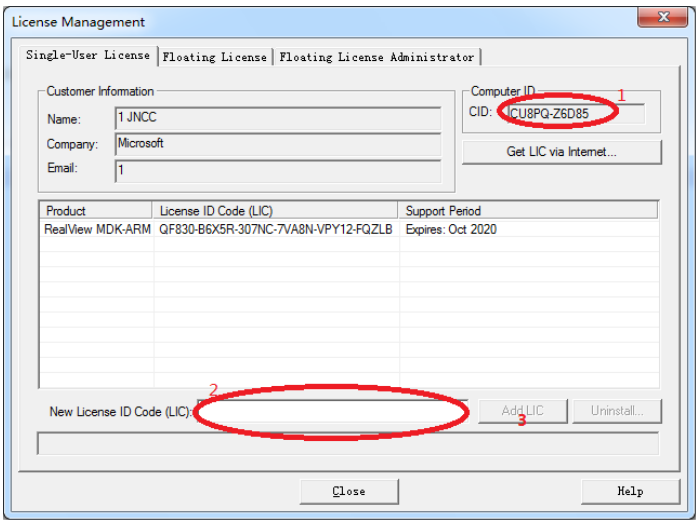
1.1 实验步骤

1.1.1 启动 KEIL，注册

1. 运行桌面“KeiluVision4”程序，File 菜单中选择 LicenseManagement

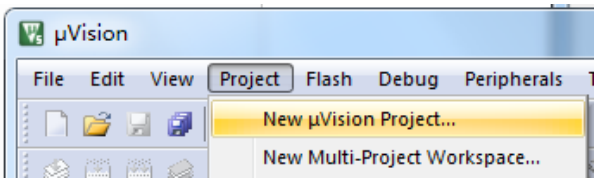




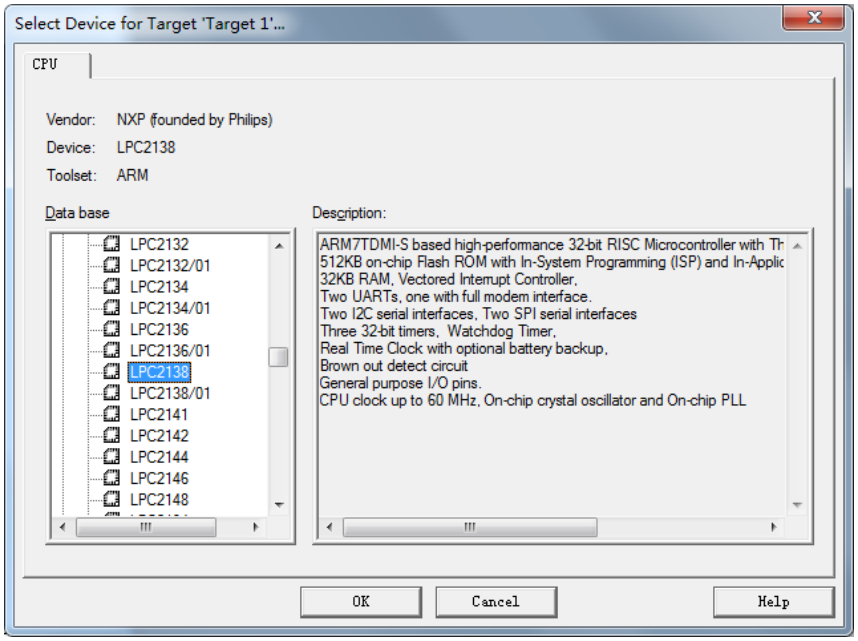


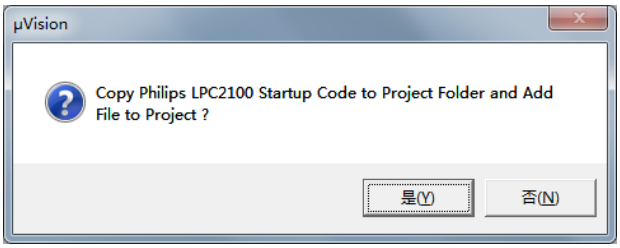
2. 复制 CID 内容①，粘贴至算号器 Keil\KEIL_Lic.exe 程序中(注意大写)，Target 选择 ARM，点击 Generate 生成序列号，复制贴入上一步的 LIC 中②，点击 AddLIC③。

1.1.2 建立 KEIL 新工程



选择 LPC2138 型号 ARM，自动生成启动代码。

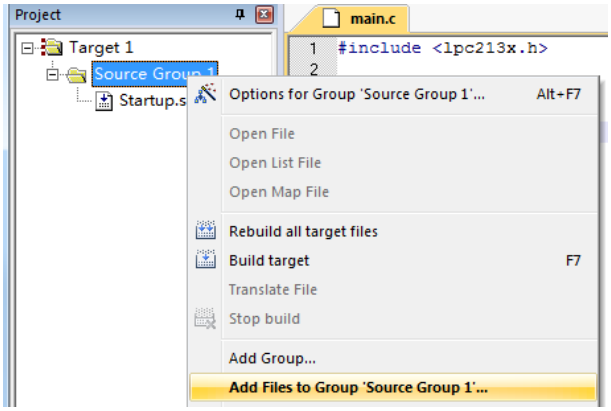




如下新建.c 文件，实现 main()函数并保存（各寄存器可通过 lpc213x.h 文件查看）

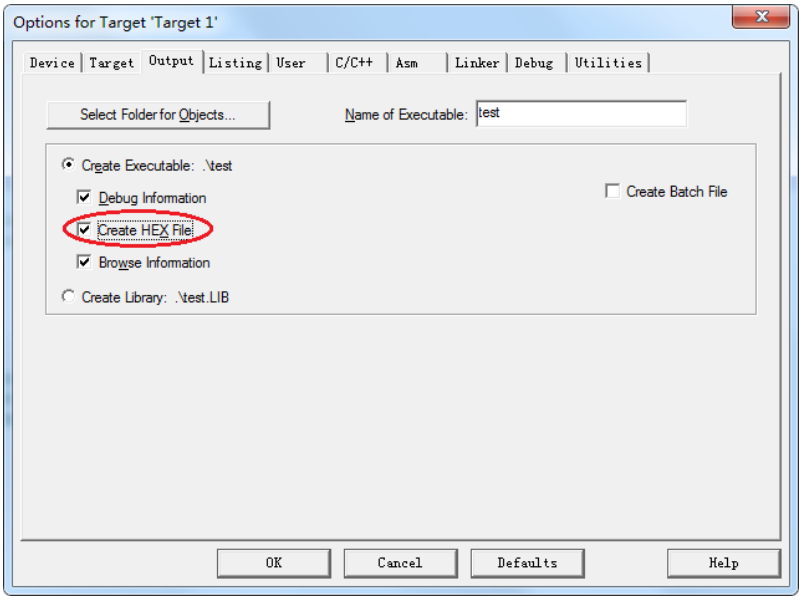
```
#include <lpc213x.h>
int main() { }
```

注意：必须在 Project 中添加新建立的.c 文件才有效。

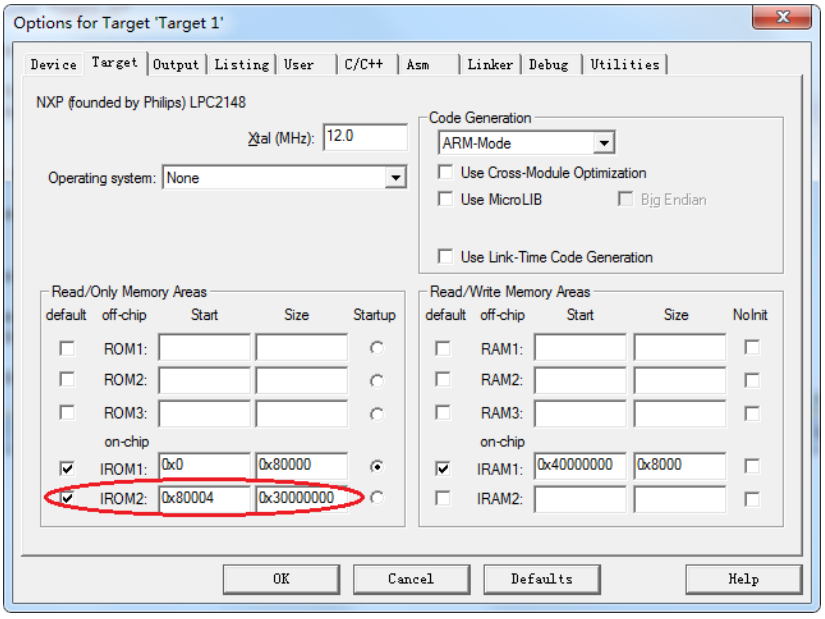


1.1.3 配置 KEIL 工程

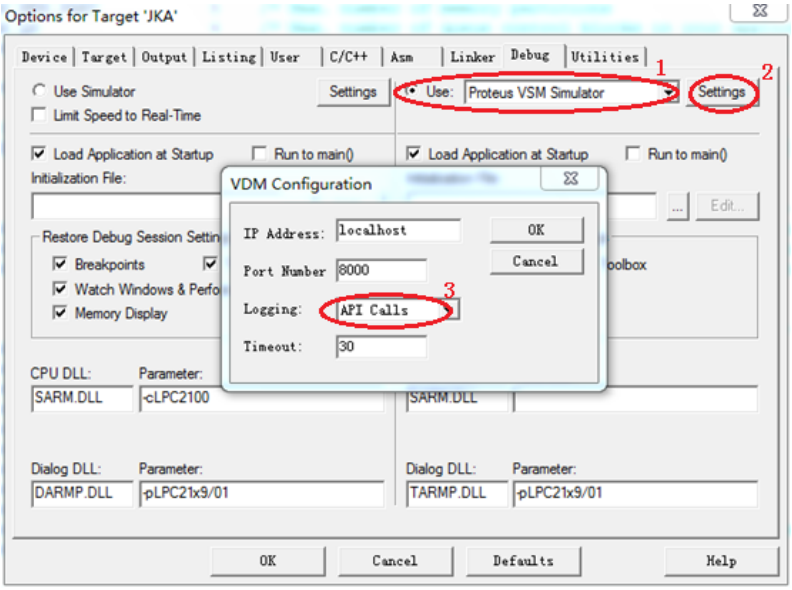
1. 配置 KEIL 工程属性，在 Output 选项中勾选”CreateHEXFile”



2. 修改工程属性，在 Target 选项 Read/OnlyMemeryAreas 中添加 IROM2 项



3. 修改工程属性的 Debug，ProteusVSMSimulator 调试选项如下设置。



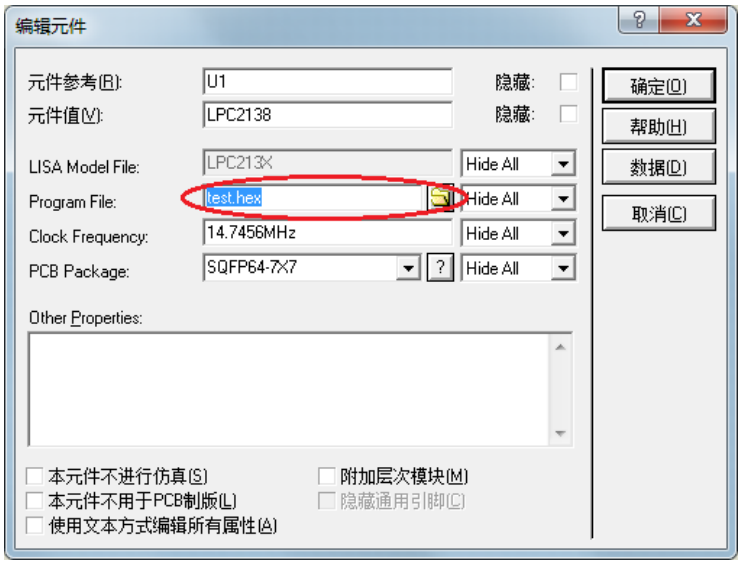
特殊情况下连不上 proteus，可以尝试将 IPAddress“localhost”修改为“127.0.0.1”。

1.1.4 配置 proteus 工程

ProteusISIS 的“debug 调试”菜单栏中，选中“remotedebugmonitor 使用远程调试监控”，即可进行 keil 调试。

1.1.5 proteus 调试使用

双击 proteusISIS 工程中的芯片，在属性中选择 hex 文件，即上一步 keil 工程中设置生成的文件。



ProteusISIS 工程中点击左下角的运行按键就可直接运行生成的 hex 文件。

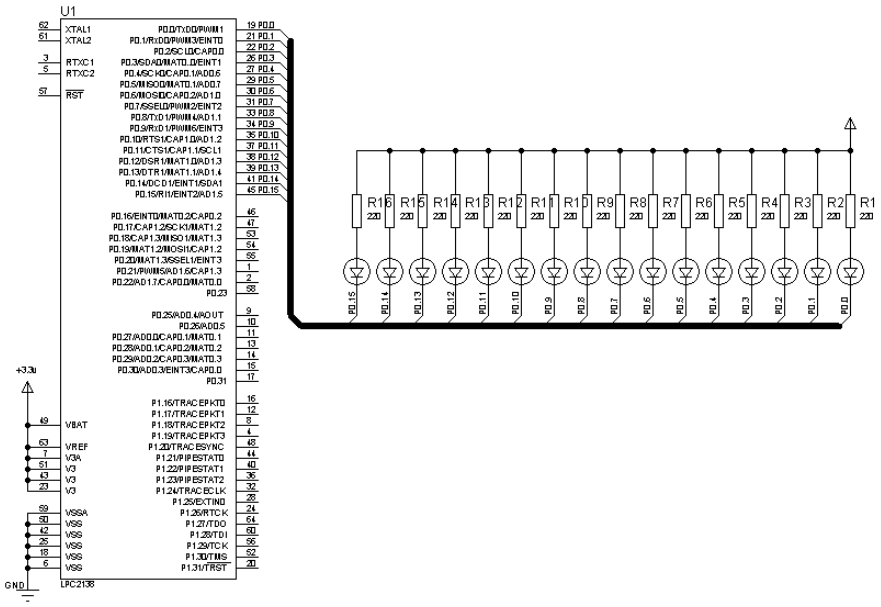
1.1.6 Keil 调试使用

1. 确定 ProteusISIS 工程已打开，且配置正确且没有运行。
2. 确定环境按前文中的“环境配置”执行，在 Keil 工程中执行 StartDebug。

3. 如连接中断后就不能再连上，请检查程序，寄存器操作失误等问题会导致调试联接中断。

实验2 流水灯-gpio

实现多个 LED 灯的流水显示。
器件列表：LPC2138、LED-BLUE、RES



2.1 引脚功能

1. 引脚功能设定

通过管脚功能选择寄存器 PINSEL0、PINSEL1、PINSEL2 设置管脚的功能：

PINSEL0	管脚名称	00	01	10	11	复位值
1:0	P0.0	GPIO P0.0	TxD(UART0)	PWM1	保留	00
3:2	P0.1	GPIO P0.1	RxD(UART0)	PWM3	EINT0	00
5:4	P0.2	GPIO P0.2	SCL0(I^2C)	捕获 0.0(TIMER0)	保留	00
7:6	P0.3	GPIO P0.3	SDA0(I^2C)	匹配 0.0(TIMER0)	EINT1	00
9:8	P0.4	GPIO P0.4	SCK(SPI0)	捕获 0.1(TIMER0)	AD0.6	00
11:10	P0.5	GPIO P0.5	MISO(SPI0)	匹配 0.1(TIMER0)	AD0.7	00
13:12	P0.6	GPIO P0.6	MOSI(SPI0)	捕获 0.2(TIMER0)	AD1.0(LPC2138)	00
15:14	P0.7	GPIO P0.7	SSEL(SPI0)	PWM2	EINT2	00
17:16	P0.8	GPIO P0.8	TxD UART1	PWM4	AD1.1(LPC2138)	00
19:18	P0.9	GPIO P0.9	RxD(UART1)	PWM6	EINT3	00
21:20	P0.10	GPIO P0.10	RTS(UART1) (LPC2138)	捕获 1.0(TIMER1)	AD1.2(LPC2138)	00
23:22	P0.11	GPIO P0.11	CTS(UART1) (LPC2138)	捕获 1.1(TIMER1)	SCL1(I^2C1)	00
25:24	P0.12	GPIO P0.12	DSR(UART1) (LPC2138)	匹配 1.0(TIMER1)	AD1.3(LPC2138)	00
27:26	P0.13	GPIO P0.13	DTR(UART1) (LPC2138)	匹配 1.1(TIMER1)	AD1.4(LPC2138)	00
29:28	P0.14	GPIO P0.14	CD(UART1) (LPC2138)	EINT1	SDA1(I^2C1)	00
31:30	P0.15	GPIO P0.15	RI(UART1) (LPC2138)	EINT2	AD1.5(LPC2138)	00

PINSEL1	管脚名称	00	01	10	11	复位值
1:0	P0.16	GPIO P0.16	EINT0	匹配 0.2(TIMER0)	捕获 0.2(TIMER0)	00
3:2	P0.17	GPIO P0.17	捕获 1.2(TIMER1)	SCK(SSP)	匹配 1.2(TIMER1)	00
5:4	P0.18	GPIO P0.18	捕获 1.3(TIMER1)	MISO(SSP)	匹配 1.3(TIMER1)	00
7:6	P0.19	GPIO P0.19	匹配 1.2(TIMER1)	MOSI(SSP)	匹配 1.3(TIMER1)	00
9:8	P0.20	GPIO P0.20	匹配 1.3(TIMER1)	SSEL(SSP)	EINT3	00
11:10	P0.21	GPIO P0.21	PWM5	AD1.6(LPC2138)	捕获 1.3(TIMER1)	00
13:12	P0.22	GPIO P0.22	AD1.7(LPC2138)	捕获 0.0(TIMER0)	匹配 0.0(TIMER0)	00
15:14	P0.23	GPIO P0.23	保留	保留	保留	00
17:16	P0.24	保留	保留	保留	保留	00
19:18	P0.25	GPIO P0.25	AD0.4	Aout（DAC） (LPC2132/2138)	保留	00
21:20	P0.26	保留				00
23:22	P0.27	GPIO P0.27	AD0.0	捕获 0.1(TIMER0)	匹配 0.1(TIMER0)	00
25:24	P0.28	GPIO P0.28	AD0.1	捕获 0.2(TIMER0)	匹配 0.2(TIMER0)	00
27:26	P0.29	GPIO P0.29	AD0.2	捕获 0.3(TIMER0)	匹配 0.3(TIMER0)	00
29:28	P0.30	GPIO P0.30	AD0.3	EINT3	捕获 0.0(TIMER0)	00
31:30	P0.31	**GPI P0.31**	保留	保留	保留	00

PINSEL2	描述	复位值
1:0	保留，用户软件不应向保留位写入 1。	NA
2	该位为 0 时，P1.36:26 用作 GPIO 管脚。该位为 1 时，P1.31:26 用作一个调试端口。	$\overline{\text{P1.26/RTCK}}$
3	该位为 0 时，P1:25:16 用作 GPIO 管脚。该位为 1 时，P1.25:16 用作一个跟踪端口。	$\overline{\text{P1.20}}$ / $\overline{\text{TRACESYNC}}$
31:4	保留，用户软件不应向保留位写入 1。	NA

GPIOP0 口每两位设置一引脚，如设置为 0 即为通用输入输出 GPIO。

2. 通用 I/O 口设置

当管脚选择 GPIO 功能时，即可通过以下寄存器实现 GPIO 功能，以下寄存器每一位对应一管脚：

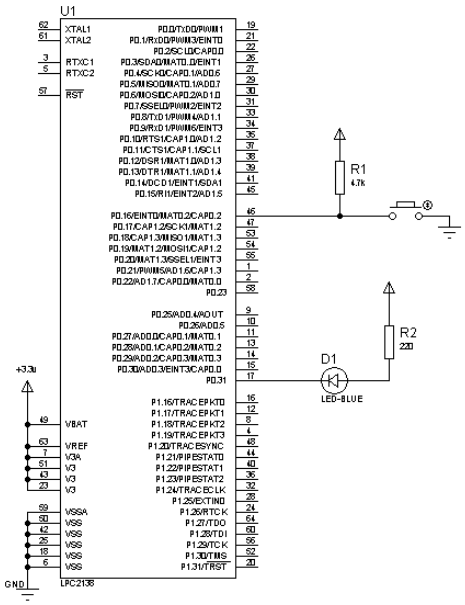
GPIO 管脚值寄存器	IO0PIN、IO1PIN	反映当前管脚状态，高电平为 1，低电平为 0
GPIO 方向寄存器	IO0DIR、IO1DIR	控制管脚的方向，0＝输入，1＝输出
GPIO 输出置位寄存器	IO0SET、IO1SET	控制管脚输出高电平，写入 1 有效，写入 0 无效
GPIO 输出清零寄存器	IO0CLR、IO1CLR	控制管脚输出低电平，写入 1 有效，写入 0 无效

3. 示例：

```
PINSEL0 = 0; //设置引脚 P0.0-P0.15 为 GPIO（16 管脚，32 位寄存器控制）
IO0DIR |= 0xffff; //设置引脚 P0.0-P0.15 为输出（16 管脚，32 位寄存器控制）
If( IO0PIN & (1<<5) ) IO0CLR = 1<<5; //如引脚 P0.5 当前状态为 1，设置引脚 P0.5 为 0
else IO0SET = 1<<5; //如引脚 P0.5 当前状态为低电平，设置引脚 P0.5 为 1，输出高电平
```

实验3 按键中断-int

按下按键，按键弹起时改变 LED 亮灭状态
器件列表：LPC2138、LED-BLUE、RES、BUTTON



3.1 按键开关

按钮开关的触点分为常开触点与常闭触点，当按下按钮开关的时候常开触点闭合，常闭触点断开，手松开后常开触点恢复断开，常闭触点恢复闭合。

3.2 中断控制

1. 向量中断控制器

向量中断控制器（VIC，Vectored Interrupt Controller）具有 32 个中断请求输入，可将其编程分为 3 类：FIQ、向量 IRQ 和非向量 IRQ。可编程分配机制意味着不同外设的中断优先级可以动态分配并调整。
这里讨论具有中等优先级的向量 IRQ（Vectored IRQ），可将 32 个中断源的任一个分配到 16 个向量 IRQslot 中的任意一个（slot0 具有最高优先级），下表为中断源中的 21 个，还有 11 个未使用。

模块	标志	VIC 通道#
WDT	看门狗中断（WDINT）	0
-	保留给软件中断	1
ARM 内核	EmbeddedICE, DbgCommRx	2
ARM 内核	EmbeddedICE, DbgCommTx	3
定时器 0	匹配 0-3（MR0, MR1, MR2, MR3） 捕获 0-3（CR0, CR1, CR2, CR3）	4
定时器 1	匹配 0-3（MR0, MR1, MR2, MR3） 捕获 0-3（CR0, CR1, CR2, CR3）	5
UART0	Rx 线状态（RLS） 发送保持寄存器空（THRE） Rx 数据可用（RDA） 字符超时指示（CTI）	6
UART1	Rx 线状态（RLS） 发送保持寄存器空（THRE） Rx 数据可用（RDA） 字符超时指示（CTI）	7
PWM0	匹配 0-6（MR0, MR1, MR2, MR3, MR4, MR5, MR6）	8
I^2C0	SI（状态改变）	9
SPI0	SPI 中断标志（SPIF） 模式错误（MODF）	10
SPI1 （SSP）	Tx FIFO 至少一半为空（TXRIS） Rx FIFO 至少一半为满（RXRIS） 接收超时条件（RTRIS） 接收溢出（RORRIS）	11
PLL	PLL 锁定（PLOCK）	12
RTC	计数器增加（RTCCIF） 报警（RTCALF）	13
系统控制	外部中断 0（EINT0）	14
	外部中断 1（EINT1）	15
	外部中断 2（EINT2）	16
	外部中断 3（EINT3）	17
A/D0	A/D 转换器 0	18
I^2C1	SI（状态改变）	19
BOD	掉电检测	20

2. 中断设置步骤

以外部中断 INT0 为例，初始化外部中断：

设置 PINSEL1 使 P0.16 为第一功能 EINT0；

设置 EXTMODE 外部中断方式寄存器：bit0-bit3 对应 EINT0-EINT3 触发方式，0-电平触发，1-边缘触发；

设置 EXTPOLAR 外部中断极性寄存器：bit0-bit3 对应 EINT0-EINT3 触发极性，0-低电平有效（下降沿）1-高电平有效（上升沿）。

中断配置：
在 VICIntSelect 中将中断分配为 IRQ 中断；
在 VICVectCntlx 中分配中断通道（优先级）；
在 VICVectAddrx 中设置中断服务程序的地址；
通过 VICIntEnable 使能外设中断。

3. 示例：

外部中断 0 初始化

```
PINSEL1 = (PINSEL1&0x03)|0x1; //设置 P0.16 为 INT0
EXTMODE |= 1<<0; //外部中断 0 设置为边缘触发
EXTPOLAR |= 1<<0; //外部中断 0 设置为上升沿触发
```

中断配置

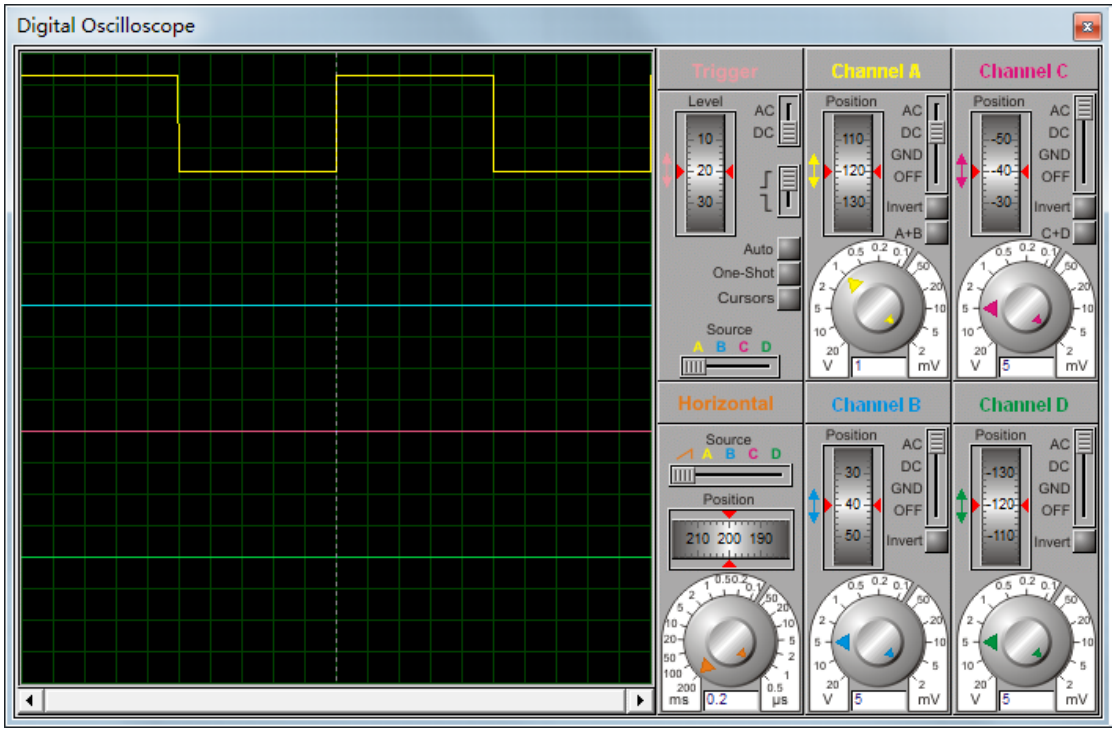
```
VICIntSelect = 0x00000000; //设置所有中断分配为 IRQ 中断
VICVectCntl0 = 0x20|14; //分配外部中断 0VIC:14 到向量中断 0
VICVectAddr0 = (unsignedint)IRQ_Eint0; //设置中断服务程序地址
VICIntEnable = 1<<14;//使能 EINT0 中断
```

中断服务程序

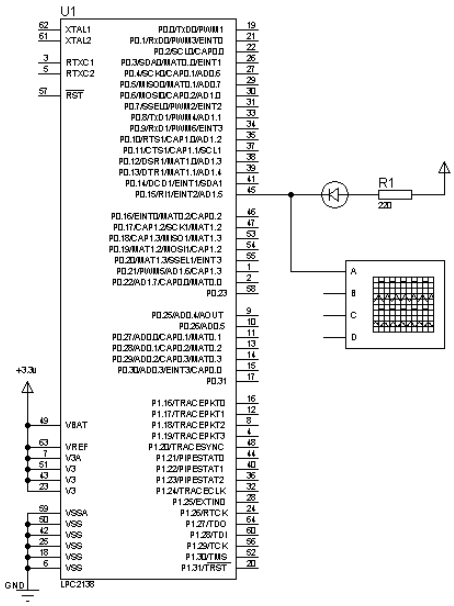
```
void __irq EINT1_IRQ(void)
{
    /*填入中断处理内容*/

    EXTINT = 0x01; //清除 EINT0 中断标志
    VICVectAddr = 0; //写 VICVectAddr，向量中断结束
}
```

实验4 LED 闪烁-timer

使用定时器控制 LED 闪烁，闪烁周期精确到 1s，并使用示波器验证



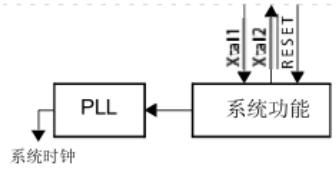
器件列表：LPC2138、LED-BLUE、RES、DigitalOscilloscope



4.1 系统时钟频率

1. 功能描述

在功能框图中能看到以下功能模块，根据 PLL 设置，确定系统时钟。



XAL1、XAL2 引脚连接的振荡器输出频率称为 FOSC，ARM 处理器时钟频率称为 cclk，PLL 时钟频率称为 Fcco，外设时钟频率称为 pclk。

需要设置 PLL 运行连接，FOSC 和 cclk 的值才能通过分频器、倍频器关联。

2. 频率计算

在 KEIL 自动生成的 start.s 中设置 PLLCFG 值，计算 M 倍增器值、P 分频器值

PLLCFG	功能	描述
4:0	MSEL4:0	倍增器值 M=MSEL+1
6:5	PSEL1:0	分频器值 P=2^PSEL

FOSC 时钟频率由引脚上的外部晶振频率决定，仿真中可通过芯片属性的 clock frequency 设置，范围 10MHz~25MHz

计算系统时钟频率 cclk=M*FOSC，范围 10MHz~Fmax（处理器最大允许频率）

计算 PLL 频率 Fcco=cclk*2*P，范围 156MHz~320MHz

外设时钟频率 pclk（VPB 时钟速率）由 VPBDIV 设置值决定：

00：VPB 总线时钟为处理器时钟的 1/4。

01：VPB 总线时钟与处理器时钟相同。

10：VPB 总线时钟为处理器时钟的 1/2。

3. 示例

KEIL 工程默认 start.s 设置：

```
VPBDIV_Val      EQU     0x00000000
PLLCFG_Val      EQU     0x00000024
```

频率计算

M = MSEL + 1 = 4 + 1 = 5; P = 2^PSEL = 2^1 = 2;

FCCLK = M*FOSC(默认 10MHz 晶振) = 50MHz

FCCO = FCCLK*2*P = 200MHz

FPCLK = FCCLK / 4 (根据 VPB 设置) = 12.5MHz

4.2 定时器

1. LPC2138 定时器功能：

定时器主要用于对内部事件进行计数的间隔定时器，通过捕获输入实现脉宽调制，及自由运行的定时器。定时器通过外设时钟（pclk）周期进行计数，可选择产生中断或根据 4 个匹配寄存器的设定，在到达指定的定时值时执行其它动作。它还包括 4 个捕获输入，用于在输入

信号发生跳变时捕获定时器值，并可选择产生中断。以下以定时器 0 为例说明定时的简单使用。

2. 定时器 0 的常用寄存器：

寄存器名称	标识	描述
预分频寄存器	T0PR	32 位预分频寄存器数值，TC 每经过 PR+1 个 pclk 周期加 1
定时器控制寄存器	T0TCR	Bit:0 计数器使能；Bit:1 计数器复位
匹配寄存器	T0MR0	与定时器计数值的比较值，相等触发相应动作
匹配控制寄存器	T0MCR	Bit:0 中断使能(MR0)；Bit:1 定时器计数值复位(MR0)

3. 中断源

模块	标志	VIC 通道#
定时器 0	匹配 0-3（MR0, MR1, MR2, MR3） 捕获 0-3（CR0, CR1, CR2, CR3）	4
定时器 1	匹配 0-3（MR0, MR1, MR2, MR3） 捕获 0-3（CR0, CR1, CR2, CR3）	5

4. 示例

定时器设置：

```
T0PR = 99; //预分频值 100，100000Hz10us
T0MCR = 0x03; //复位，中断配置，TC 和 MR 匹配时产生中断，TC 并复位
T0MR0 = 100000; //匹配值，1s 定时
T0TCR = 0x03; //启动并复位 T0TC
T0TCR = 0x01; //启动 T0
```

中断设置

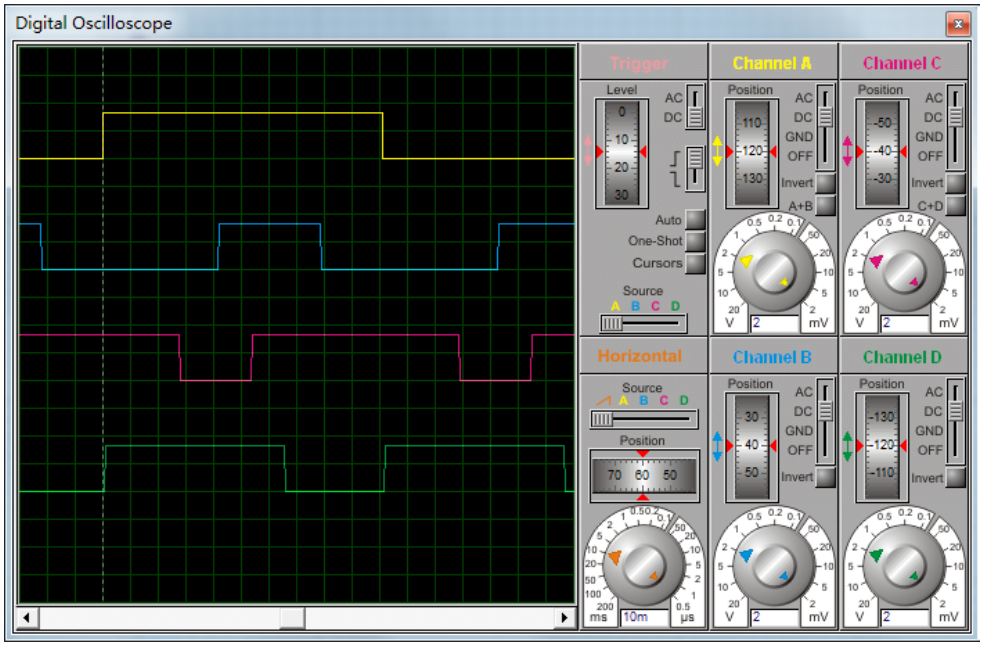
```
VICIntSelect = 0x00; //所有中断设置为 IRQ 中断
VICVectCntl0 = 0x20|4; //定时器 0 设为最高优先级（向量控制 0）使能|中断号
VICVectAddr0 = (unsignedlong)IRQ_Timr0;
VICIntEnable = 1<<4;//使能定时器 0 中断，定时器 0 的 VIC 通道为 4
```

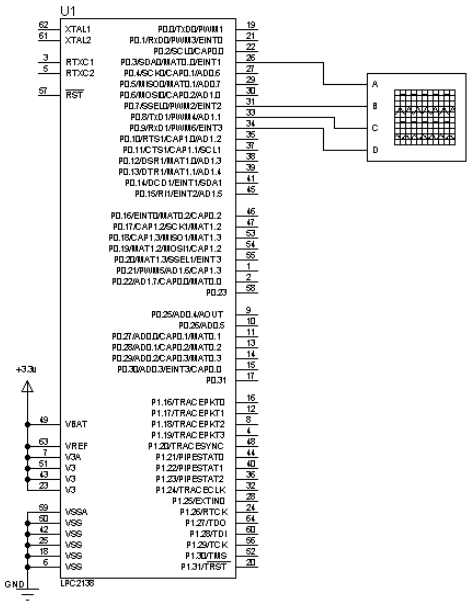
中断处理函数

```
void__irq IRQ_Timr0(void)
{
    /*填入定时处理内容*/

    T0IR = 0x01; //清中断位
    VICVectAddr = 0x00; //通知 VIC 中断处理结束*/
}
```

实验5 脉宽调制-pwm

设置 PWM0、PWM2、PWM4、PWM6 输出如下波形：
PWM0 通过中断控制管脚输出 100ms 周期信号；
PWM2 设置双边沿控制，输出 41ms 上升沿，78ms 下降沿的周期信号；
PWM2 设置双边沿控制，输出 53ms 上升沿，27ms 下降沿的周期信号；
PWM2 设置双边沿控制，输出 0ms 上升沿，65ms 下降沿的周期信号。





5.1 PWM 输出

PWM 基于标准的定时器模块并具有其所有特性。不过 LPC2131 只将其 PWM 功能输出到管脚。定时器对外设时钟(pclk)进行计数，可选择产生中断或基于 7 个匹配寄存器，在到达指定的定时值时执行其它动作（设置为高/低电平、翻转或者无动作）。它还包括 4 个捕获输入，用于在输入信号发生跳变时捕获定时器值，并可选择在事件发生时产生中断。PWM 功能是一个附加特性，建立在匹配寄存器事件基础之上。

1. 双边沿控制 PWM 输出

示例说明 PWM 值与波形输出之间关系：

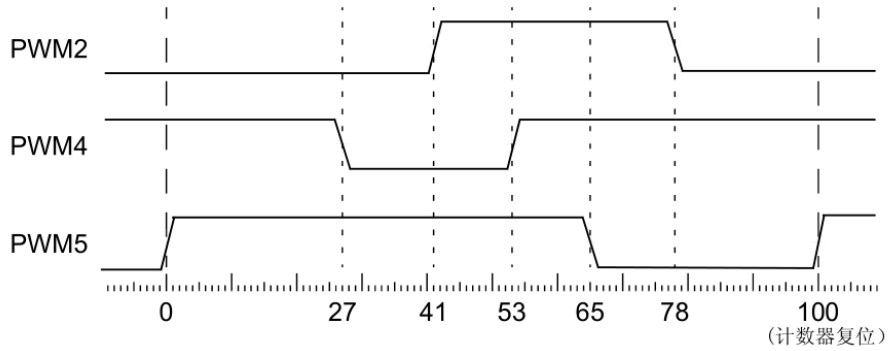
定时器配置为 PWM 模式，匹配寄存器 0 配置为在发生匹配事件时复位定时器/计数器控制位 PWMSEL2 和 PWMSEL4 置位，匹配寄存器值如下：

MR0=100 (PWM 速率)

MR1=41, MR2=78 (PWM2 输出)

MR3=53, MR4=27 (PWM4 输出)

MR5=00, MR6=65 (PWM5 输出)



3. 示例：

双边 PWM 管脚设置

```
PINSEL0 =          (2 << 14) |          // P0.7 连接 PWM2
                   (2 << 16) |          // P0.8 连接 PWM4
                   (2 << 18);           // P0.9 连接 PWM6
```

双边 PWM 初始化

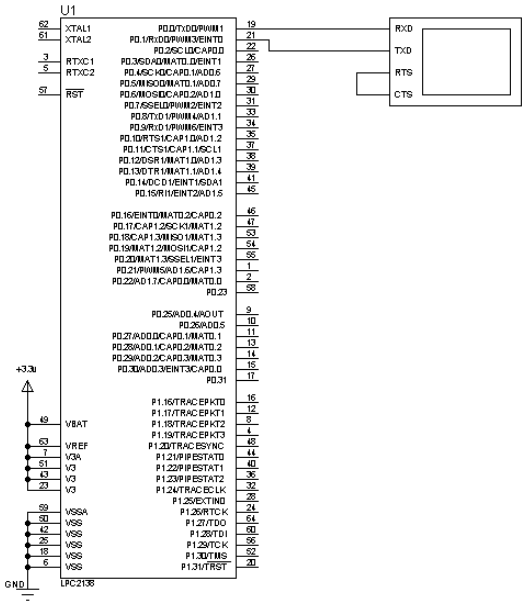
```
PWMTC = 0;       // 定时器设置为 0
PWMPR = 9999; // 时钟分频，tpclk*(9999+1)=0.1us*10000 = 1ms
PWMMCR = 0x03; // 设置 PWMMR0 匹配后复位 PWMTC，并产生中断标志
PWMPCR =           (1 << 2) | // PWM2 双边沿控制
                   (1 << 4) | // PWM4 双边沿控制
                   (1 << 6) | // PWM6 双边沿控制
                   (1 << 10) | // 使能 PWM2 输出
                   (1 << 12) | // 使能 PWM4 输出
                   (1 << 14); // 使能 PWM6 输出
PWMMR0 = 100; // 100ms 定时
```

双边 PWM 的脉宽和位置设置

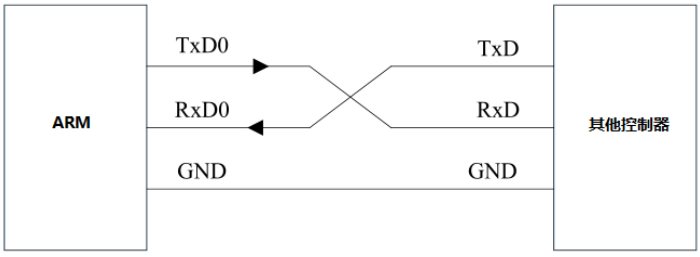
```
    PWMMR1 = 41;              // PWM2 的位置
    PWMMR2 = 78;
    PWMMR3 = 53;              // PWM4 的位置
    PWMMR4 = 27;
    PWMMR5 = 0;               // PWM6 的位置
    PWMMR6 = 65;
    PWMLER = 0x7F; // 锁存所有 PWM 匹配值，更新匹配值后，必须锁存
双边 PWM 输出开启 PWM
    PWMTCR = 0x01; // 使能 PWM
设置 PWM 中断
    VICIntSelect = 0x00000000;     // 所有中断连接 IRQ
    VICVectCntl0 = 0x20 | 8;       // 选择通道 0
    VICVectAddr0 = (int)PWM_Int;     // PWM 中断服务程序地址
    VICIntEnable = (1 << 8);       // 使能 PWM 中断
    while(1);   // 等待中断
```

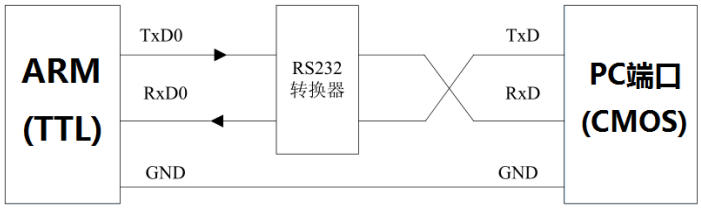
实验6 串口通讯-uart

控制串口 0 输出字符串，在虚拟串口终端上显示，实现 printf()，scanf()功能。
器件列表：LPC2138、虚拟仪器 VIRTUALTERMINAL



6.1 UART 线路连接





6.2 UART0 轮询配置

1. 设置管脚功能连接

   PINSEL0 = (PINSEL0 & (~0x0F)) | 0x05; //P0.0、P0.1 选择 UART0 功能

2. 设置串口

   串口模式设置

U0LCR	功能	描述	复位值
1:0	字长度选择	00：5 位字符长度 01：6 位字符长度 10：7 位字符长度 11：8 位字符长度	0
2	停止位选择	0：1 个停止位 1：2 个停止位（如果 U0LCR[1:0]=00 则为 1.5）	0
3	奇偶使能	0：禁止奇偶产生和校验 1：使能奇偶产生和校验	0
5:4	奇偶选择	00：奇数 01：偶数 10：强制为“1”，奇偶固定 11：强制为“0”，奇偶固定	0
6	间隔控制	0：禁止间隔发送 1：使能间隔发送 当 U0LCR6 为高电平有效时，输出管脚 UART0 TxD 强制为逻辑 0。	0
7	除数锁存访问位	0：禁止访问除数锁存 1：使能访问除数锁存	0

   波特率设置

$$UxDLM, UxDLL = \frac{Fpclk}{16 \times baud}$$

   配置示例

```
unsigned short int fdiv, baud=9600; //波特率 baud
fdiv    = (FPCLK / 16) / baud;
U0LCR = 0x83; //DLAB=1，允许设置波特率
U0DLM= fdiv / 256; //UART 除数锁存 MSB 寄存器
U0DLL = fdiv % 256; //UART 除数锁存 LSB 寄存器
U0LCR = 0x03;
```

3. 发送或接收数据

   发送字节数据：

```
U0THR = data; //data 为要发送的数据
while ((U0LSR & 0x40) == 0); //等待数据发送完毕
```

   接收字节数据

```
while ((U0LSR & 0x01) == 0); //等待有效数据
rcv_dat = U0RBR; //读取数据
```

6.3 KEIL 实现 printf 与 scanf

1. 工程中添加 Retarget.c 文件（文件名自取，一定要添加至工程中），键入以下内容

```
#include <stdio.h>
#include <rt_misc.h>
```

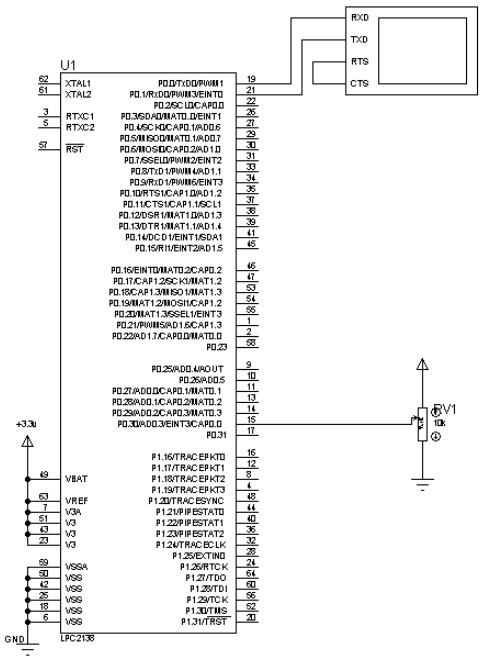
```
#pragma import(__use_no_semihosting_swi)

extern int sendchar (int c);
extern int getkey      (void);

struct __FILE { int handle; /* Add whatever you need here */ };
FILE __stdout;
FILE __stdin;

int fputc(int c, FILE *f) {
   return (sendchar(c));
}

int fgetc(FILE *f) {
   return (getkey());
}

int ferror(FILE *f) {
   /* Your implementation of ferror */
   return EOF;
}

void _ttywrch(int c) {
   sendchar(c);
}

void _sys_exit(int return_code) {
label:    goto label;    /* endless loop */
}
```

2. 通过 uart 功能实现以下函数：

```
int sendchar (int c);
int getkey      (void);
```

实验7 模数转换-adc

通过 ADC 采集滑动变阻器上的电压值，通过串口显示结果
器件列表：LPC2138、POT-HG



7.1 ADC 模数转换器

模数转换器，是把经过与标准量（或参考量）比较处理后的模拟量转换成以二进制数值表示的离散信号的转换器，简称 ADC 或 A/D 转换器。
使用 ADC 模块时，先要将测量通道引脚设置为 AINx 功能，然后通过 ADCR 寄存器设置 ADC 的工作模式，ADC 转换通道，转换时钟(CLKDIV 时钟分频值)，并启动 ADC 转换。可以通过查询或中断的方式等待 ADC 转换完毕，转换数据保存在 ADDR 寄存器中。
使用示例：

1. 设置管脚功能
   PINSEL1 = (PINSEL1 & ~(0x3<<28)) | 1<<28;　　//P0.30 连接到 AD0.3
2. 采样频率及模式设置（FPCLK 为 10MHz）

$$CLKDIV = \frac{Fpclk}{Fadclk} - 1$$

```
AD0CR = (1 << 3) |              // SEL=8,选择通道 3
        ((FPCLK / 1000000 - 1) << 8)  |    //CLKDIV=Fpclk/1000000-1,转换时钟为 1MHz
        (0 << 16)             |           // BURST=0,软件控制转换操作
        (0 << 17)             |           // CLKS=0, 使用 11clock 转换
        (1 << 21)             |           // PDN=1,正常工作模式
        (0 << 22)             |           // TEST1:0=00,正常工作模式
        (1 << 24)             |           // START=1,直接启动 ADC 转换
        (0 << 27);                        // 直接启动 ADC 转换时，此位无效
```

3. 采样控制

```
AD0CR |= 1 << 24;   // 进行第一次转换
while ((AD0DR & 0x80000000) == 0);          // 等待转换结束
```

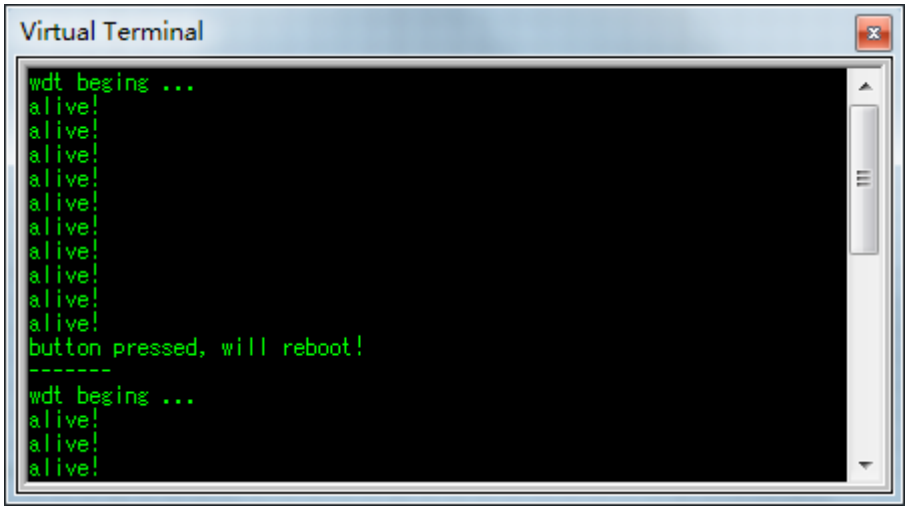
4. 结果计算（满额为 3.30V）

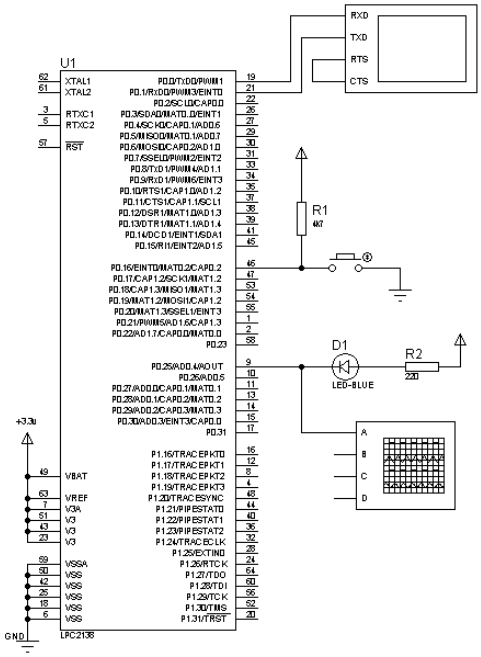
$$Uw1 = \frac{\text{满额电压}}{\text{转换精度}} \times VALUE_{AD0DR} = \frac{3300}{1024} \times VALUE_{AD0DR}(mV)$$

```
ADC_Data = (AD0DR >> 6) & 0x3ff;
ADC_Data = ADC_Data * 3300; // 满额电压 3.30V
ADC_Data = ADC_Data / 1024; // 10 位 AD 转换，1024 级
```

实验8 看门狗-wdt

程序设置看门狗机制，每次喂狗闪烁 LED，当按键按下后停止喂狗，使系统重启。
器件列表：LPC2138、LED-BLUE、RES、BUTTON





8.1 看门狗

为了对处理器运行状态进行实时监测的考虑，产生了一种专门用于监测程序运行状态的机制，俗称“看门狗”，其工作原理是程序周期性操作看门狗复位（喂狗），一旦程序跑飞或进入

死循环状态时，丢失喂狗信号，便会送出一个复位信号，使处理器发生复位，这样便实现了处理器的自动复位功能。

看门狗包括一个 4 分频的预分频器和一个 32 位计数器看门狗定时器的输入时钟是由 pclk 经过 4 分频得到，定时器递减计数，定时器递减的最小值为 0xFF。如果设置一个小于 0xFF 的值，系统会将 0xFF 装入计数器。因此最小看门狗间隔为(tpclk×256×4)，假定用户循环喂狗周期是 M×tpclk，则 WDTC 的最小值是 M/4。要求喂狗周期必须小于看门狗超时值决定的周期才能保证正常喂狗。

看门狗使用步骤

1. 设置看门狗模式

   WDMOD = 0x03; // 设置看门狗模式：中断且复位
2. 设置看门狗定时常数

   WDTC = 10000000;// 设置看门狗定时器参数
3. 喂狗操作

   WDFEED = 0xAA;// 注意：第一次喂狗启动看门狗

   WDFEED = 0x55;

示例：

```
/* 初始化看门狗 */
WDTC = 10000000; // 设置 WDTC，喂狗重装值
WDMOD = 0x03;                // 复位并启动 WDT
WdtFeed(); // 进行喂狗操作

/* 周期性喂狗 */
delay(20000); //约 20ms 周期进行喂狗
WDFEED = 0xAA;
WDFEED = 0x55;
```