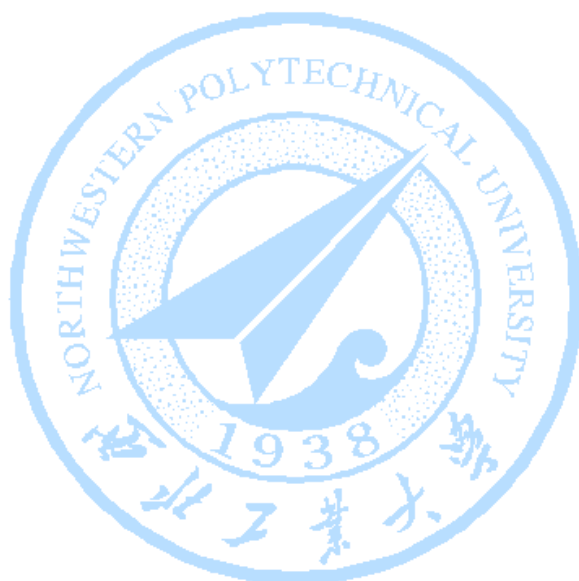


# 实 验 报 告



课程名称: 嵌入式系统及应用

专    业: 信息安全

班    级: 09061401

姓    名: Higor

学    号: 2014xxxxxx

自 动 化 学 院 实 验 中 心

2017 年 5 月 4 日

实 验 项 目 列 表

序 号	实 验 项 目 名 称			备 注
1	基础程序设计实验			
2	LED 灯定时控制实验			
3	温度采样通信实验			
基础实验成绩			指导教师	XXX

# 实验一报告

## 一、实验名称

基础程序设计实验

## 二、实验目的

- 1、熟悉 LPC2378 系统套件的实验环境；
- 2、熟悉 KEIL 集成开发环境，掌握工程的生成方法；
- 3、熟悉汇编指令，掌握系统启动文件的汇编编写方法；
- 4、根据要求能够实现 C 语言的程序设计；
- 5、掌握 LPC2378 技术手册的使用方法；

## 三、实验内容

- 1、学习 LPC2378 系统套件的组成及注意事项；
- 2、掌握 KEIL 集成开发环境，构建 LPC2378 工程文件；
- 3、定位系统启动文件参数，对系统时钟、存储区及外设进行配置；
- 4、以“随机数生成”程序为例，编译链接生成可执行文件，并下载运行程序；
- 5、运用 KEIL 集成开发环境的调试工具观察结果；
- 6、按照实验要求编写程序；

## 四、实验例程与实验要求

已给出 100 以内随机数生成例程，实现对生成的随机数进行排序，并记录排序后数据在原数据所处的位置，观察、对比排序前后的数据变化。

## 五、程序框图

产生随机数并记录其原始位置



对随机数进行冒泡排序，排序过程中进行数据交换时同时交换数据的位置

## 六、核心程序

```
#define NUM 100




unsigned int data[NUM];
unsigned int sorted[NUM];
unsigned int index[NUM];

int main (void) {
    unsigned int n;
    int i,j,swaped;
    //生成100位0-100的随机数
    for(n=0;n<NUM;n++)
    {
        data[n]=rand()%(NUM+1);
        sorted[n] = data[n];
        index[n] = n;
    }
    //对随机数进行排序,同时记录排序后数据在原数据中的位置
    for(i=NUM;i>1;i--)
    {
        swaped = 0;
        for(j=0;j<i-1;j++)
        {
            if(sorted[j]>sorted[j+1])
            {
                sorted[j] += sorted[j+1];
                sorted[j+1] = sorted[j]-sorted[j+1];
                sorted[j] -= sorted[j+1];
                index[j] += index[j+1];
                index[j+1] = index[j]-index[j+1];
                index[j] -= index[j+1];
                swaped = 1;
            }
        }
        if(swaped==0)break;
    }
    while (1);
}
```

七、实验结果

通过增加对随数的排序程序后，将程序编译下载到 LPC2378 开发板运行，所得到的随机产生的随机数和排序后数据以及排序后数据在原数据中的位置如表格 1 所示。

表格 1 实验结果展示

排序前数据		排序后数据		排序后数据在原数据中的位置	
 data	0x40000...	 sorted	0x40000...	 index	0x40000...
[0]	41	[0]	0	[0]	30
[1]	78	[1]	1	[1]	54
[2]	14	[2]	2	[2]	13
[3]	11	[3]	2	[3]	57
[4]	32	[4]	4	[4]	34
[5]	70	[5]	4	[5]	77
[6]	97	[6]	8	[6]	83
[7]	40	[7]	11	[7]	3
[8]	23	[8]	12	[8]	36
[9]	86	[9]	14	[9]	2
[10]	38	[10]	14	[10]	42
[11]	37	[11]	14	[11]	48
[12]	59	[12]	14	[12]	55
[13]	2	[13]	15	[13]	86
[14]	99	[14]	17	[14]	66
[15]	79	[15]	18	[15]	37
[16]	89	[16]	19	[16]	39
[17]	88	[17]	19	[17]	51
[18]	24	[18]	22	[18]	32
[19]	87	[19]	22	[19]	33
[20]	62	[20]	23	[20]	8
[21]	29	[21]	24	[21]	18
[22]	41	[22]	27	[22]	74
[23]	98	[23]	27	[23]	79
[24]	47	[24]	28	[24]	43
[25]	99	[25]	28	[25]	92
[26]	29	[26]	29	[26]	21
[27]	95	[27]	29	[27]	26
[28]	65	[28]	30	[28]	52
[29]	36	[29]	31	[29]	41
[30]	0	[30]	32	[30]	4
[31]	74	[31]	36	[31]	29
[32]	22	[32]	36	[32]	38
[33]	22	[33]	37	[33]	11
[34]	4	[34]	38	[34]	10
[35]	46	[35]	38	[35]	94

◆ [36]	12	◆ [36]	39	◆ [36]	89
◆ [37]	18	◆ [37]	40	◆ [37]	7
◆ [38]	36	◆ [38]	40	◆ [38]	40
◆ [39]	19	◆ [39]	41	◆ [39]	0
◆ [40]	40	◆ [40]	41	◆ [40]	22
◆ [41]	31	◆ [41]	42	◆ [41]	44
◆ [42]	14	◆ [42]	42	◆ [42]	62
◆ [43]	28	◆ [43]	42	◆ [43]	63
◆ [44]	42	◆ [44]	43	◆ [44]	71
◆ [45]	87	◆ [45]	44	◆ [45]	59
◆ [46]	75	◆ [46]	44	◆ [46]	68
◆ [47]	89	◆ [47]	46	◆ [47]	35
◆ [48]	14	◆ [48]	47	◆ [48]	24
◆ [49]	63	◆ [49]	48	◆ [49]	60
◆ [50]	75	◆ [50]	49	◆ [50]	95
◆ [51]	19	◆ [51]	50	◆ [51]	84
◆ [52]	30	◆ [52]	51	◆ [52]	69
◆ [53]	64	◆ [53]	53	◆ [53]	70
◆ [54]	1	◆ [54]	53	◆ [54]	73
◆ [55]	14	◆ [55]	53	◆ [55]	93
◆ [56]	100	◆ [56]	55	◆ [56]	98
◆ [57]	2	◆ [57]	59	◆ [57]	12
◆ [58]	82	◆ [58]	59	◆ [58]	76
◆ [59]	44	◆ [59]	62	◆ [59]	20
◆ [60]	48	◆ [60]	63	◆ [60]	49
◆ [61]	81	◆ [61]	63	◆ [61]	90
◆ [62]	42	◆ [62]	64	◆ [62]	53
◆ [63]	42	◆ [63]	64	◆ [63]	80
◆ [64]	92	◆ [64]	64	◆ [64]	87
◆ [65]	69	◆ [65]	65	◆ [65]	28
◆ [66]	17	◆ [66]	65	◆ [66]	78
◆ [67]	87	◆ [67]	65	◆ [67]	91
◆ [68]	44	◆ [68]	67	◆ [68]	97
◆ [69]	51	◆ [69]	68	◆ [69]	72
◆ [70]	53	◆ [70]	69	◆ [70]	65
◆ [71]	43	◆ [71]	70	◆ [71]	5
◆ [72]	68	◆ [72]	74	◆ [72]	31

◆ [73]	53	◆ [73]	75	◆ [73]	46
◆ [74]	27	◆ [74]	75	◆ [74]	50
◆ [75]	81	◆ [75]	76	◆ [75]	82
◆ [76]	59	◆ [76]	78	◆ [76]	1
◆ [77]	4	◆ [77]	79	◆ [77]	15
◆ [78]	65	◆ [78]	80	◆ [78]	88
◆ [79]	27	◆ [79]	81	◆ [79]	61
◆ [80]	64	◆ [80]	81	◆ [80]	75
◆ [81]	98	◆ [81]	81	◆ [81]	85
◆ [82]	76	◆ [82]	82	◆ [82]	58
◆ [83]	8	◆ [83]	86	◆ [83]	9
◆ [84]	50	◆ [84]	87	◆ [84]	19
◆ [85]	81	◆ [85]	87	◆ [85]	45
◆ [86]	15	◆ [86]	87	◆ [86]	67
◆ [87]	64	◆ [87]	88	◆ [87]	17
◆ [88]	80	◆ [88]	89	◆ [88]	16
◆ [89]	39	◆ [89]	89	◆ [89]	47
◆ [90]	63	◆ [90]	89	◆ [90]	99
◆ [91]	65	◆ [91]	92	◆ [91]	64
◆ [92]	28	◆ [92]	95	◆ [92]	27
◆ [93]	53	◆ [93]	97	◆ [93]	6
◆ [94]	38	◆ [94]	98	◆ [94]	23
◆ [95]	49	◆ [95]	98	◆ [95]	81
◆ [96]	99	◆ [96]	99	◆ [96]	14
◆ [97]	67	◆ [97]	99	◆ [97]	25
◆ [98]	55	◆ [98]	99	◆ [98]	96
◆ [99]	89	◆ [99]	100	◆ [99]	56

可以看出，原始数据是随机产生的 100 以内的随机数，排序后的数据这是将原始随机数排序后得到的数据，同时排序后的数据在原数据中的位置也被记录下来。通过检查，验证了排序和记录位置的正确性。

## 八、实验心得

通过本次基础程序设计实验，我熟悉了 LPC2378 系统套件的实验环境以及学习使用 KEIL 集成开发环境构建 LPC2378 的工程文件。在对随机生成的数据进行排序的过程中，我不仅熟悉了利用 KEIL 进行仿真实验，也学会了如何将程序下载到开发板中运行，同时掌握了利用 KEIL 提供的调试工具对 LPC2378 开发板进行在线调试，包括在线查看数据、断点设置等。

## 实验二报告

### 一、实验名称

LED 灯定时控制实验

### 二、实验目的

- 1、了解 LED 灯与 LPC2378 系统的连接关系；
- 2、熟悉 GPIO 额度结构，掌握 GPIO 寄存器的使用方法；
- 3、掌握正确配置引脚的方法，实现指定功能；
- 4、熟悉定时器的机制，掌握定时器的寄存器的使用方法；
- 5、熟悉系统中断机制，能够设定中断寄存器；
- 6、实现 LED 流水灯的程序设计；

### 三、实验内容

- 1、运行 GPIO 示例程序，正确配置 P2.0~P2.7 引脚，实现输出功能；
- 2、运行定时器中断示例程序，正确配置定时器 0 和向量中断控制器，实现 1 秒定时；
- 3、按照实验要求编写应用程序；

### 四、实验例程与实验要求

已给出 GPIO 例程和定时器中断例程。例程实现了定时器的 1s 定时中断，并在中断中改变 GPIO 的状态，实现 LED 灯的闪烁效果。

实验要求为实现对 8 个 LED 灯的控制，要求运行间隔 1s 的 LED 流水灯。

LPC2378 系统与 LED 灯的连接关系如图 1 中的原理图所示，由原理图可知，8 个 LED 灯分别与 LPC2378 系统的引脚 P2.0~P2.7 连接，且当引脚为低电平时，LED 灯将被点亮。



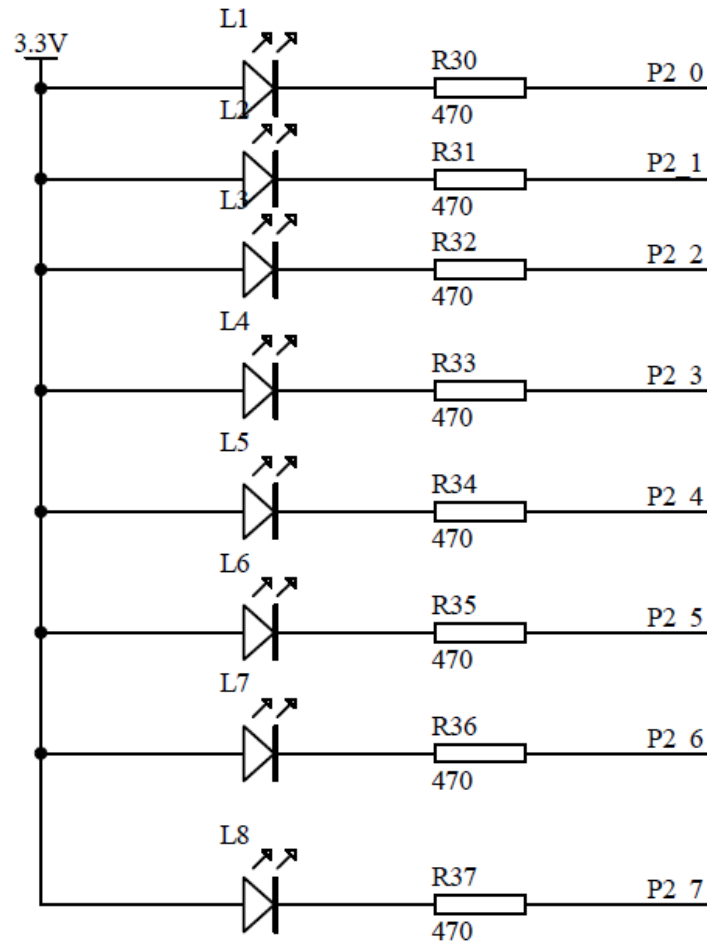


图 1 LED 灯与 LPC2378 的连接关系

## 五、主要外设及其功能寄存器

本实验使用的主要外设包括：GPIO、定时器 TIMER0、中断向量控制器 VIC。

### GPIO 相关寄存器：

**PINSEL4：**管脚功能选择寄存器，控制管脚的功能，本实验需要设置 PINSEL4 选择 P2.0~P2.7 为 GPIO 功能。

**FIO2DIR：**高速 GPIO 端口方向控制寄存器。该寄存器控制端口 2 管脚的方向，本实验需要设置 FIO2DIR 使 P2.0~P2.7 为输出方向。

**FIO2MASK：**端口的高速屏蔽寄存器。写、置位、清零和读端口改变或返回该寄存器中仅由 0 使能的位。

**FIO2SET：**高速端口置位寄存器。该寄存器控制输出管脚的状态。写 1 使相应的端口管脚产生高电平。写 0 没有影响。

**FIO2CLR：**高速端口输出清零寄存器。该寄存器控制输出管脚的状态。写 1 到相应端口的管脚产生低电平。写 0 没有影响。

**FIO2PIN：**高速端口管脚值寄存器。数字端口管脚的当前状态可从该寄存器中读出。

### 定时器 TIMER0 相关寄存器:

T0TC: 定时器 0 计数器。32 位的 TC 每隔(PR+1)个 PCLK 周期递增一次。

T0PC: 预分频计数器。32 位的 PC 是一个计数器, 它会增加到与 PR 中存放的值相等。当达到了 PR 的值时, TC 增加, PC 被清除。

T0MCR: 匹配控制寄存器。MCR 用来控制在匹配出现时是否产生中断和 TC 是否复位。

T0MR0: 匹配寄存器 0。MR0 可通过 MCR 设定为在匹配时复位 TC, 停止 TC 和 PC 和/或产生中断。

T0TCR: 定时器控制寄存器。TCR 用来控制定时器计数器功能。定时器计数器可以通过 TCR 来禁能或复位。

T0IR: 中断寄存器。可向 IR 写入相应的值来清除中断。

### 中断向量控制器 VIC 相关寄存器:

VICIntSelect: 中断选择寄存器。该寄存器将 32 个中断请求的每个都分配为 FIQ 或 IRQ。

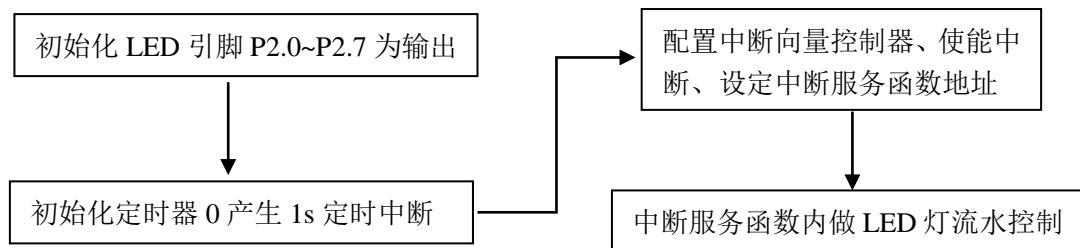
VICVectPriority4: 向量优先级 4 寄存器。指定了相应的向量 IRQ 的优先级。

VICVectAddr4: 向量地址 4 寄存器。保存了相应的向量 IRQ 的中断服务程序 (ISR)地址。

VICIntEnable: 中断使能寄存器。该寄存器控制将 32 个中断请求和软件中断中的哪几个使能为 FIQ 或 IRQ。

VICVectAddr: 向量地址寄存器。当发生 IRQ 中断时, 向量地址寄存器保存当前有效中断的地址。

## 六、程序框图



## 七、核心程序

```
//LED 初始化
void LED_Init(void)
{
    PINSEL10 = 0;           //禁止 ETM
    PINSEL4  &= 0xFFFFFFFF; //选择 P2.0~P2.7 为 GPIO
    FIO2DIR  |= 0x000000FF; //设置 P2.0~P2.7 为输出
    FIO2MASK &= 0xFFFFFFFF;
    FIO2SET   = 0x000000FF; //熄灭所有 LED
}
```

```

//LED 配置函数，相应位为 1 表示灯亮
void LED_Config(uint8_t led_status)
{
    uint32_t status = 0x00000000;
    status |= led_status;
    FIO2CLR = status;
    status = 0x00000000;
    status |= ~led_status;
    FIO2SET = status;
}

//TIMER0 中断服务函数
void __irq IRQ_Timer0 (void)
{
    uint8_t LED = 0x01;
    static uint8_t count = 0;
    LED_Config(LED<<count);
    count++;
    if(count>7) count=0;

    T0IR = 0x01;           //清除中断标志
    VICVectAddr = 0x00;    //通知 VIC 中断处理结束，更新 VIC
}

//TIMER0 与 VIC 初始化
void Timer_Init(void)
{
    /* 定时器 0 设定 */
    T0TC = 0;              //定时器 0 的计数器清零
    T0PR = 0;              //时钟不预分频
    T0MCR = 0x03;          //设置 T0MR0 匹配后复位 T0TC，并产生中断标志
    T0MR0 = 12000000;       //TIM0 的 PCLK=CCLK/4=48/4=12M
                           //设定为 12000000，则每 1s LED 状态变换一次
    T0TCR = 0x01;          //启动定时器

    /* 设置 IRQ TIM0 中断对应的是 VIC 中断通道的中断向量 4 */
    VICIntSelect = VICIntSelect & ~(1<<4); //定时器 0 中断通道设置为 IRQ 中断
    VICVectPriority4 = 0x00;                 //设置中断向量 4 的优先级
    VICVectAddr4 = (unsigned int)IRQ_Timer0; //设置中断向量 4 的中断程序地址为
                                           //TIM0 中断程序
    VICIntEnable |= 1 << 0x04;              //使能中断向量 4，即允许定时器 0 中断
}

```

```
#include "LPC23xx.H"
#include "led.h"
#include "timer.h"

int main (void) {
    LED_Init();
    Timer_Init();
    while (1){;}
}
```

## 八、实验结果

通过将程序编译并下载至 LPC2378 开发板上，观察到 8 个 LED 灯以 1s 的时间间隔循环亮起，实现了间隔为 1s 的 LED 流水灯。

实验现象表明，程序中对 GPIO、TIMER0、VIC 的配置是正确的，P2.0~P2.7 正确地设置为 GPIO 且为输出方向可以控制 LED 灯。定时器 TIMER0 正确地产生了 1s 间隔的计数周期且结合 VIC 产生了中断。在中断服务函数内对 LED 灯进行了流水灯控制。

## 九、实验心得

通过本次 LED 灯定时控制实验，我熟悉了 LPC2378 系统与 LED 灯的连接关系，熟悉了 GPIO、定时器 0、向量中断控制器 VIC 相关寄存器的配置。掌握了利用定时器定时产生中断并在中断中控制 LED 灯状态的方法与配置流程。

# 实验三报告

## 一、实验名称

温度采样通信实验

## 二、实验目的

- 1、了解热敏电阻与温度的对应关系，熟悉温度采样硬件电路；
- 2、熟悉 AD 的结构，掌握 AD 寄存器的使用方法；
- 3、掌握正确配置引脚的方法，实现指定功能；
- 4、熟悉串口的机制，掌握串口寄存器的使用方法；
- 5、实现温度采样，并将采样结构通过串口上传至 PC 端；

## 三、实验内容

- 1、运行 AD 示例程序，正确配置 P0.23 引脚，实现 AD 采样功能；
- 2、运行 UART 示例程序，正确配置串口 0，实现数据通过串口上传至 PC；
- 3、按照实验要求编写应用程序；

## 四、实验例程与实验要求

已给出 AD 例程和 UART 串口例程。AD 例程的功能是对电压进行采样，并将采样结果反应在 8 个 LED 灯上。UART 例程实现了串口回传功能，将串口收到的数据原样返回。

实验要求实现对环境温度的采样上传功能，要求采样位数为 10 位，上传速率为 0.5s。

LPC2378 系统与热敏电阻的连接关系如图 2 中的原理图所示，由原理图可知，由于温度变化引起的热敏电阻阻值的变化将改变 AD 采样点 P0.23 引脚的电位，因此通过 AD 采样并转换电压就可以得到环境温度。

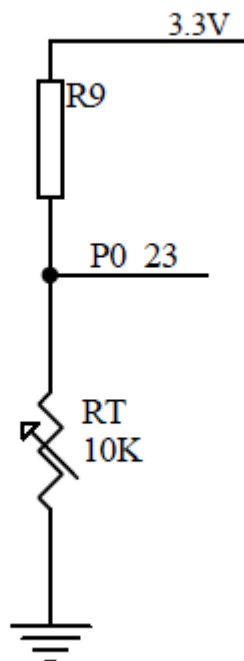


图 2 热敏电阻与 LPC2378 的连接关系

## 五、主要外设及其功能寄存器

本实验使用的主要外设包括：GPIO、数模转换器 AD、异步串口 UART、定时器 TIMER0、中断向量控制器 VIC。

### GPIO 相关寄存器：

**PINSEL0：**管脚功能选择寄存器，控制管脚的功能，本实验需要设置 PINSEL0 选择 P0.2、P0.3 分别为 UART 的 TXD0、RXD0。

**PINSEL1：**管脚功能选择寄存器，控制管脚的功能，本实验需要设置 PINSEL1 选择 P0.23 为 AD 功能。

### 数模转换器 AD 相关寄存器：

**AD0CR：**A/D 控制寄存器。A/D 转换开始前，必须写入 AD0CR 寄存器来选择工作模式。本实验需设置 AD0CR 实现 10 位 AD 采样。

**AD0GDR：**A/D 全局数据寄存器，它包含最近一次 A/D 转换的结果。通过读取此寄存器可以获取最近一次 AD 转换的结果。

**AD0STAT：**A/D 状态寄存器。该寄存器包含所有 A/D 通道的 DONE 标记和 OVERRUN 标记，以及 A/D 中断标记。在开始 AD 转换后，需要检查此寄存器的 DONE 位直到转换完成。

**AD0DR0：**A/D 通道 0 数据寄存器。该寄存器包含在通道 0 上完成的最近一次转换的结果。

### **异步串口 UART 相关寄存器：**

**U0LCR:** UART0 线控制寄存器，U0LCR 决定发送和接收数据字符的格式。本次实验设施 U0LCR 设置格式为 8N1，8 位，无校验，1 停止位。

**U0DLL:** UART0 除数锁存 LSB 寄存器。除数锁存是波特率发生器的一部分，它保存了用于产生波特率时钟的 APB 时钟（PCLK）分频值。波特率时钟必须是目标波特率的 16 倍。U0DLL 和 U0DLM 寄存器一起构成一个 16 位除数。

**U0DLM:** UART0 除数锁存 MSB 寄存器。本实验结合 U0DLL、U0DLM 设置波特率为 57600。

**U0IER:** UART0 中断使能寄存器。设置 UART 相关事件的中断的禁止与使能。如接收、发送、线状态中断。

**U0IIR:** UART0 中断标识寄存器，U0IIR 提供状态代码用于指示挂起中断的中断源和优先级。

**U0LSR:** UART0 线状态寄存器。U0LSR 为只读寄存器，它提供 UART0 Tx 和 Rx 模块的状态信息。

**U0RBR:** UART0 接收缓存寄存器。U0RBR 是 UART0 RX FIFO 的最高字节。它包含了最早接收到的字符，我们可以通过总线接口读取它。

**U0THR:** UART0 发送保持寄存器。U0THR 是 UART0 TX FIFO 的最高字节。它包含了 TX FIFO 中最新的字符，我们可以通过总线接口对它进行写操作。本实验将需要通过串口发送的数据写入此寄存器。

### **定时器 TIMER0 相关寄存器：**

**T0TC:** 定时器 0 计数器。32 位的 TC 每隔(PR+1)个 PCLK 周期递增一次。

**T0PC:** 预分频计数器。32 位的 PC 是一个计数器，它会增加到与 PR 中存放的值相等。当达到了 PR 的值时，TC 增加，PC 被清除。

**T0MCR:** 匹配控制寄存器。MCR 用来控制在匹配出现时是否产生中断和 TC 是否复位。

**T0MR0:** 匹配寄存器 0。MR0 可通过 MCR 设定为在匹配时复位 TC，停止 TC 和 PC 和/或产生中断。

**T0TCR:** 定时器控制寄存器。TCR 用来控制定时器计数器功能。定时器计数器可以通过 TCR 来禁能或复位。

**T0IR:** 中断寄存器。可向 IR 写入相应的值来清除中断。

### **中断向量控制器 VIC 相关寄存器：**

**VICIntSelect:** 中断选择寄存器。该寄存器将 32 个中断请求的每个都分配为 FIQ 或 IRQ。

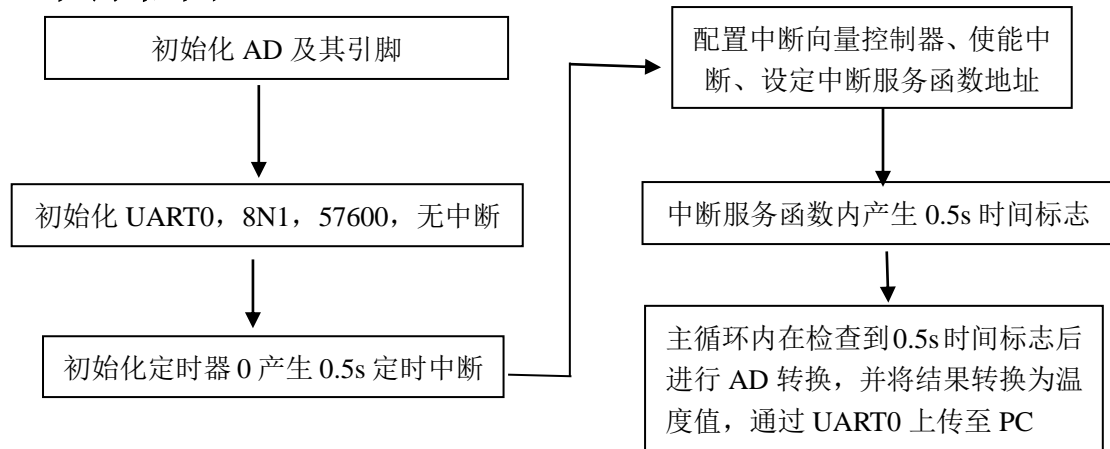
**VICVectPriority4:** 向量优先级 4 寄存器。指定了相应的向量 IRQ 的优先级。

**VICVectAddr4:** 向量地址 4 寄存器。保存了相应的向量 IRQ 的中断服务程序 (ISR)地址。

**VICIntEnable:** 中断使能寄存器。该寄存器控制将 32 个中断请求和软件中断中的哪几个使能为 FIQ 或 IRQ。

**VICVectAddr:** 向量地址寄存器。当发生 IRQ 中断时，向量地址寄存器保存当前有效中断的地址。

## 六、程序框图



## 七、核心程序

```
#include <stdlib.h>
#include <stdio.h>
#include "LPC23xx.H"
#include "ad.h"
#include "timer.h"
#include "uart.h"
volatile uint8_t time_500ms_flag = 0;
int main (void) {
    uint32_t temperatureAD;
    uint8_t temperature;
    //初始化 AD
    AD_Init();
    //UART 初始化, 波特率 57600
    UART_Init(57600);
    //初始化 TIMER0, 产生 0.5s 定时中断
    Timer_Init(500);
    while (1){
        if(time_500ms_flag){
            time_500ms_flag = 0;
            temperatureAD = AD_Get();
            printf("AD 采样值: %d\n",temperatureAD); //已重定向至 UART0
            temperature = Get_Temp(temperatureAD);

            printf("温度: %d°C\n",temperature); //已重定向至 UART0
        }
    }
}
```



```

//AD 初始化
void AD_Init(void)
{
    PINSEL10 = 0;           //禁止 ETM
    PCONP |= (1<<12);      // AD 外设 ON
    //硬件连接关系: AD0.0 即 P0.23 接入热温电阻
    PINSEL1=(PINSEL1&0xFFFF3FFF)|0x00004000;    //P0.23 功能设定为 AD0.0
}

//获取 AD 采样结果
uint32_t AD_Get(void)
{
    uint32_t addata;
    AD0CR = 0x01000001 | 0x00200300;           // A/D: 10 位 AD 时钟为
PCLK/(3+1)=12/4=3MHz 启动 AD
    do {
        addata = AD0GDR;                       // 读 AD 全局寄存器
    } while ((addata & 0x80000000) == 0);      // 判断 AD 是否转换完成
    AD0CR &= ~0x01000001;                     // 停止 AD 运行
    addata = (addata >> 6) & 0x03FF;          //获取 AD 值
    return addata;
}

//10K 的热敏电阻 0-100 度所对应的码表值
const unsigned int Temp_code[]=
{
    // 0-25

    2843,2718,2599,2486,2379,2277,2180,2087,1999,1916,1836,1760,1688,1619,155
    3,1490,1430,1373,1319,1266,1217,1169,1124,1081,1039,1000,
    // 26--50

    962,925,891,857,826,795,766,738,711,686,661,638,615,593,573,553,534,515,4
    97,480,464,449,433,419,405,
    // 51-75

    392,379,366,355,343,332,321,311,301,292,283,274,265,257,249,242,234,227,2
    20,214,208,201,195,190,184,
    // 76--100

    179,174,169,164,159,155,150,146,142,138,134,131,127,124,120,117,114,111,1
    08,105,102,100,97,94,92
};

```

```

/*****
//查电阻值所对应的温度是多少
*****/
uint8_t check_code(uint32_t k)
{
    unsigned char i;
    if(k<=92) return(100); //大于100度以上都显示为100度
    for(i=0;k<Temp_code[i];i++);
    return (i-1);
}

/*****
//R=30K 或 20K
//RT=10K
*****/
uint32_t Totemp(uint32_t AD)
{
    unsigned int Itemp;
    unsigned int Tk;
    Tk=1023-AD;

    Itemp=(uint32_t)((uint32_t)1000*AD)/Tk; //扩大100倍
    return(Itemp);
}

uint8_t Get_Temp(uint32_t AD)
{
    return (check_code(Totemp(AD))); //转换温度
}

//UART 初始化、只发不收
void UART_Init(uint32_t baudrate)
{
    uint32_t Fdiv;
    PINSEL10 = 0; //禁止 ETM
    PINSEL0 = (PINSEL0&0xFFFFF0F)|0x00000050; //RxD0 and TxD0
    U0LCR = 0x83; //8 bits, no Parity, 1 Stop bit 允许访问
除数锁存
    Fdiv = ( Fpclk / 16 ) / baudrate ; /*baud rate */
    U0DLM = Fdiv / 256; //除数高8位
    U0DLL = Fdiv % 256; //除数低8位
    U0LCR = 0x03; /* 禁止访问除数锁存*/
    U0FCR = 0x07; //FIFO 使能, 清空读写缓存
}

```

```

//加入以下代码,支持 printf 函数
#if 1
#pragma import(__use_no_semihosting)
//标准库需要的支持函数
struct __FILE
{
    int handle;
};
FILE __stdout;
//定义 _sys_exit() 以避免使用半主机模式
_sys_exit(int x)
{
    x = x;
}
//重定义 fputc 函数
int fputc(int ch, FILE *f)
{
    while((U0LSR&LSR_THRE)==0); //循环发送,直到发送完毕
    U0THR = (uint8_t) ch;
    return ch;
}
#endif
//TIMER0 中断服务函数
void __irq IRQ_Timer0 (void)
{
    time_500ms_flag = 1;
    T0IR = 0x01; //清除中断标志
    VICVectAddr = 0x00; //通知 VIC 中断处理结束,更新 VIC
}
//TIMER0 与 VIC 初始化
void Timer_Init(uint32_t time_ms)
{
    /* 定时器 0 设定 */
    T0TC = 0; //定时器 0 的计数器清零
    T0PR = 0; //时钟不预分频
    T0MCR = 0x03; //设置 T0MR0 匹配后复位 T0TC,并产生中断标志
    T0MR0 = (Fpclk/1000)*time_ms; //TIM0 的 PCLK=CCLK/4=48/4=12M
    T0TCR = 0x01; //启动定时器
    /* 设置 IRQ TIM0 中断对应的是 VIC 中断通道的中断向量 4 */
    VICIntSelect = VICIntSelect & (~(1<<4)); //定时器 0 中断通道设置为 IRQ 中断
    VICVectPriority4 = 0x00; //设置中断向量 4 的优先级
    VICVectAddr4 = (unsigned int)IRQ_Timer0; //设置中断向量 4 的中断程序地址为
TIM0 中断程序
    VICIntEnable |= 1 << 0x04; //使能中断向量 4,即允许定时器 0 中断
}

```

## 八、实验结果

通过将程序编译并下载至 LPC2378 开发板上，连接开发板的 RS232 串口至 PC。PC 端打开串口工具，观察到每 0.5s 收到一次温度的采样结果与转换后的温度。如图 3 所示。

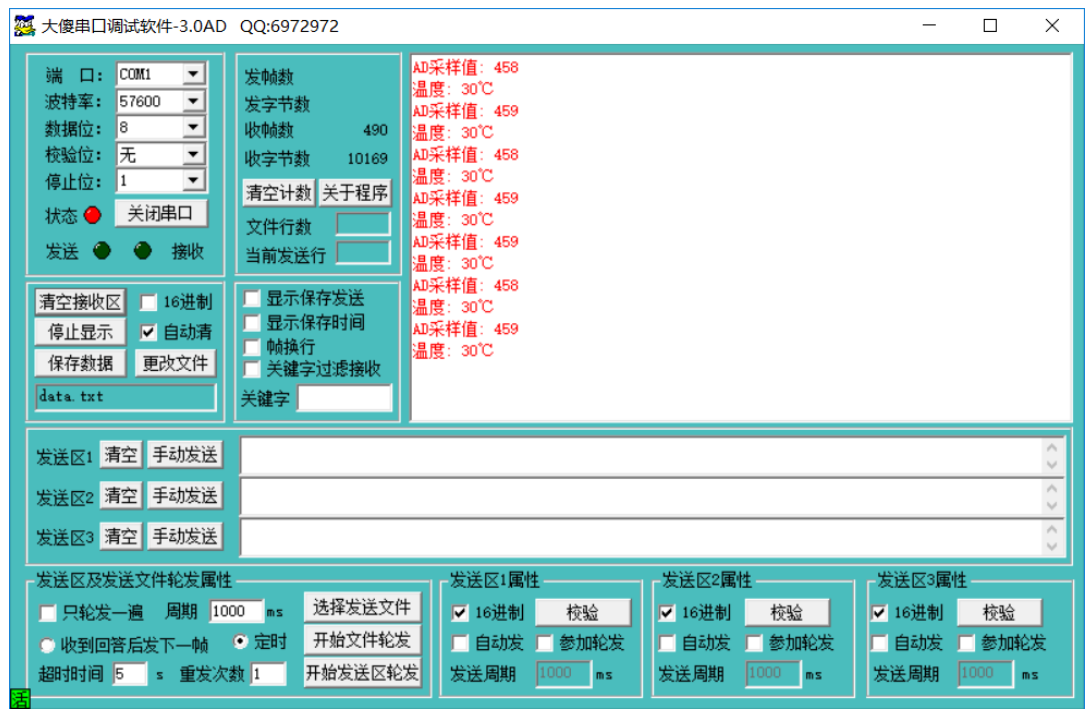


图 3 串口助手收到开发板上传的数据

上传到 PC 的数据包括原始的 AD 采样值和转换后的温度值。温度的转换是根据 10K 的热敏电阻 0-100℃所对应的码表值确定的。通过查表得出 AD 采样值对应的环境温度。

实验现象表明，程序中对 GPIO、AD、UART、TIMER0、VIC 的配置是正确的，P0.23 被配置为 AD 功能，P0.2、P0.3 被配置为 UART0 的 TXD、RXD，并成功实现与 PC 的通信。定时器 TIMER0 正确地产生了 0.5s 间隔的计数周期且结合 VIC 产生了中断，这个 0.5s 的定时作为 AD 转换与串口上传的时间基准。

## 九、实验心得

通过本次温度采样通信实验，我了解了热敏电阻与温度的对应关系，熟悉温度采样的硬件电路。熟悉了 GPIO、模数转换器 AD、异步串口 UART、定时器 0、向量中断控制器 VIC 相关寄存器的配置。掌握了通过 AD 对电压进行采样、通过 UART 与 PC 或其他串口设备通信的方法。利用定时器定时产生中断作为时间基准，控制 AD 转换的开启与结果读取，并将转换结果通过 UART 上传至 PC。