目 录

1. 实验1 时钟选择与GPIO操作 3

1.1 实验目的 3

1.2 实验主要器材和设备 3

1.3 实验任务的技术解决方案说明 3

1.3.1 实验任务1\_1 3

1.3.2 实验任务1\_2 3

1.4 实验结果自测记录 4

1.4.1 实验任务1\_1现象与分析 4

1.4.2 实验任务1\_2现象与分析 5

1.5 实验核心代码清单 5

1.5.1 实验1\_1核心代码 5

1.5.2 实验1\_2核心代码 5

2. 实验2 A2000TM4扩展板使用方法与SYSTICK中断 6

2.1 实验目的 6

2.2 实验主要器材和设备 6

2.3 实验任务的技术解决方案说明 6

2.3.1 实验任务2\_1 7

2.3.2 实验任务2\_2 7

2.3.3 实验任务2\_3 8

2.3.4 实验任务2\_4 8

2.4 实验结果自测记录 9

2.4.1 实验2\_1实验现象 9

2.4.2 实验2\_2实验现象 9

2.4.3 实验2\_3实验现象 9

2.4.4 实验2\_4实验现象 9

2.5 实验核心代码清单 9

2.5.1 实验2\_1核心代码 9

2.5.2 实验2\_2核心代码 10

2.5.3 实验2\_3核心代码 10

2.5.4 实验2\_4核心代码 10

3. 实验3 UART串行通讯实验 11

3.1 实验目的 11

3.2 实验主要器材和设备 11

3.3 实验任务的技术解决方案说明 11

3.3.1 实验任务3\_1 11

3.3.2 实验任务3\_2 12

3.3.3 实验任务3\_3 12

3.4 实验结果自测记录 13

3.4.1 实验3\_2实验现象 13

3.4.2 实验3\_3实验现象 13

3.5 实验核心代码清单 14

3.5.1 实验3\_2核心代码 14

3.5.2 实验3\_3核心代码 15

4. 附录 学习心得和意见建议 17

4.1 实验1学习思考 17

4.1.1 实验1任务说明书中三道思考题的回答 17

4.1.2 实验1的总结 17

4.2 实验2学习思考 18

4.3 实验3学习思考 18

4.4 关于《工程实践与科技创新II-A》基础实验部分的整体感受 18

# 实验1 时钟选择与GPIO操作

## 实验目的

熟悉ARM的集成开发环境KEIL uVision5，能够自行建立一个实验用工程项目；

理解CPU的时钟信号，理解软件延时的时长与系统时钟(SYSCLK)频率的关系；

掌握GPIO的工作原理，能够结合GPIO的输入与输出进行实验。

## 实验主要器材和设备

电脑，TM4C1294NCPDT实验板卡，数字示波器。

## 实验任务的技术解决方案说明

### 实验任务1\_1

#### 任务概览

实验1\_1部分主要含有两大部分：

（1） 设置不同时钟源信号，利用软件自带DEBUG功能、实验室配备的示波器观测信号频率；

（2） 实现当按下PJ0时，PF0以大约100毫秒为周期快速闪烁；松开PJ0，PF0以6秒为周期缓慢闪烁的外部功能。

#### 实现思路

（1）对于设置不同的时钟源信号，我们仅需不断修正自带函数SysCtlClockFreqSet中的两大参数：第一个参数表示了需要采用的不同时钟源，可供选择的有振荡器源（MOSC、PIOSC）和系统时钟源（OSC、PLL），选中的源之间用“或”（即“|”）表示；第二个参数表示了我们想设置的频率。当进行DEBUG观测时，我们需要打开DEBUG观测窗口，将g\_ui32SysClock添加至WATCH 1，点击“Run to Cursor”至GPIOInit（）处即可。当利用示波器观测时，我们将示波器探头的一侧插至板孔96，另一个插至板孔66，即PF0处，在示波器显示屏幕上观测相关数据；

（2）对于利用按键控制LED的状态，我们在源文件的基础上仅需修改全局变量FastFlashTime为50（50毫秒）、SlowFlashTime为3000（3秒）即可。

### 实验任务1\_2

#### 任务概览

修改源代码，实现如下外部功能：当按下PJ0时，点亮PN1；放开时，熄灭PN1；当按下PJ1时，点亮PN0；放开时，熄灭PN0。

#### 实现思路

利用按键控制LED亮灭的关键在于GPIOPinRead函数以及GPIOPinWrite函数。GPIOPinRead函数的返回值表示按键的状态，若该值为0，则表示按键按下，反之亦然。GPIOPinWrite函数则根据GPIOPinRead的返回值来确定LED的状态。具体代码实现策略如1.5.2所示。

## 实验结果自测记录

### 实验任务1\_1现象与分析

根据表1中的不同时钟源组合，我们不断修改SysCtlClockFreqSet函数的两个参数。经过DEBUG以及4-105的示波器观测可以得到实际观测频率值，具体数值详见表1右边两栏。

**表1 系统时钟频率的设定方式对PF0信号频率的影响**

| 序号 | 外部时钟 | 振荡器源 | | 系统时钟源 | | PLL VCO频率 | | 希望设定的系统时钟频率（MHz） | DEBUG观测到的时钟频率数值(MHz) | USR\_SW1-PJ0为释放状态下，示波器观测的PF0信号频率(Hz) |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| SYSCTL\_XTAL\_25MHZ | MOSC(25M)  SYSCTL\_OSC\_MAIN | PIOSC(16M)  SYSCTL\_OSC\_INT | SYSCTL\_USE\_PLL | SYSCTL\_USE\_OSC | SYSCTL\_CFG\_VCO\_320 | SYSCTL\_CFG\_VCO\_480 |
| 1 |  |  | √ |  | √ |  |  | 16 | 16 | 10.0 |
| 2 |  |  | √ |  | √ |  |  | 12 | 16 | 10.0 |
| 3 |  |  | √ |  | √ |  |  | 8 | 8 | 5.025 |
| 4 | √ | √ |  |  | √ |  |  | 25 | 25 | 10.0 |
| 5 | √ | √ |  |  | √ |  |  | 12 | 12.5 | 5.0 |
| 6 | √ | √ |  |  | √ |  |  | 1 | 1 | 0.40325 |
| 7 | √ | √ |  | √ |  |  | √ | 25 | 24 | 14.881 |
| 8 | √ | √ |  | √ |  |  | √ | 20 | 20 | 12.5 |
| 9 | √ | √ |  | √ |  |  | √ | 8 | 8 | 4.975 |
| 10 |  |  | √ | √ |  |  | √ | 20 | 20 | 12.579 |
| 11 |  |  | √ | √ |  |  | √ | 8 | 8 | 5.0738 |
| 12 |  |  | √ | √ |  |  | √ | 1 | 15 | 9.489 |

由此我们可以归纳得出现象（1）~（2）：

（1）部分预设频率未能实现。

【分析】据了解，系统时钟是由系统时钟源经硬件分频等处理后得到的，这种机制决定了分频系数不能随心所欲的设置。在《工程实践与科技创新II-A》课上，何浩老师曾经讲过，分频系数必须为整数。在表1中，第五行、第七行的两种组合并不具有整的分频系数，故结果与预设相悖。与此同时，一般情况下，系统时钟的值是相对固定的，即使选取的分频系数是整数，可能也无法实现.

（2）DEBUG的观测值与示波器的观测值存在着一定的联系。

【分析】显然，LED亮灭的一个周期内，包含数量相同的系统时钟周期，则示波器的观测值与DEBUG的观测值（若真实反应了系统时钟周期）应存在一定的比例关系。在表1中，一些组合中存在着1.6、2.5两种比例关系。

在实验1\_1的第二个任务中，我们实现了按键控制PJ0不同状态的闪烁情况，我们观察到了如下所述的现象：

（3）在实验1\_1的第二个任务中，当LED灯慢闪时，按下USR\_SW1 ，并没有马上进入快闪模式，而是等待了一会儿。

【分析】我们认为，这是因为按住USR\_SW1的时候系统可能正在执行慢闪的delay，而无法检测到按键key\_value的变化导致的。经过一段时间delay结束后，系统方可检测到按键变化进入快闪模式。软件延时机制导致了微处理器对按键按下信号的响应出现明显迟滞，即阻塞式延时。

### 实验任务1\_2现象与分析

经过编译与运行，我们能够实现预设功能，即：当按下PJ0时，点亮PN1；放开时，熄灭PN1；当按下PJ1时，点亮PN0；放开时，熄灭PN0。

## 实验核心代码清单

### 实验1\_1核心代码

//g\_ui32SysClock=SysCtlClockFreqSet(SYSCTL\_OSC\_INT|SYSCTL\_USE\_OSC,16000000);

//g\_ui32SysClock=SysCtlClockFreqSet(SYSCTL\_OSC\_INT|SYSCTL\_USE\_OSC,12000000);

//g\_ui32SysClock=SysCtlClockFreqSet(SYSCTL\_OSC\_INT,8000000);

//g\_ui32SysClock=SysCtlClockFreqSet(SYSCTL\_USE\_OSC|SYSCTL\_OSC\_MAIN|SYSCTL\_XTAL\_25MHZ,25000000);

//g\_ui32SysClock=SysCtlClockFreqSet(SYSCTL\_USE\_OSC|SYSCTL\_OSC\_MAIN|SYSCTL\_XTAL\_25MHZ,12000000);

//g\_ui32SysClock=SysCtlClockFreqSet(SYSCTL\_USE\_OSC|SYSCTL\_OSC\_MAIN|SYSCTL\_XTAL\_25MHZ,1000000);

//g\_ui32SysClock=SysCtlClockFreqSet(SYSCTL\_USE\_PLL|SYSCTL\_OSC\_MAIN|SYSCTL\_XTAL\_25MHZ|SYSCTL\_CFG\_VCO\_480,25000000);

//g\_ui32SysClock=SysCtlClockFreqSet(SYSCTL\_USE\_PLL|SYSCTL\_OSC\_MAIN|SYSCTL\_XTAL\_25MHZ|SYSCTL\_CFG\_VCO\_480,20000000);

//g\_ui32SysClock=SysCtlClockFreqSet(SYSCTL\_USE\_PLL|SYSCTL\_OSC\_MAIN|SYSCTL\_XTAL\_25MHZ|SYSCTL\_CFG\_VCO\_480,8000000);

//g\_ui32SysClock=SysCtlClockFreqSet(SYSCTL\_USE\_PLL|SYSCTL\_OSC\_INT|SYSCTL\_CFG\_VCO\_480,20000000);

//g\_ui32SysClock=SysCtlClockFreqSet(SYSCTL\_USE\_PLL|SYSCTL\_OSC\_INT|SYSCTL\_CFG\_VCO\_480,8000000);

//g\_ui32SysClock=SysCtlClockFreqSet(SYSCTL\_USE\_PLL|SYSCTL\_OSC\_INT|SYSCTL\_CFG\_VCO\_480,1000000);

### 实验1\_2核心代码

if(!GPIOPinRead(GPIO\_PORTJ\_BASE, GPIO\_PIN\_0))

GPIOPinWrite(GPIO\_PORTN\_BASE, GPIO\_PIN\_0, GPIO\_PIN\_0);

else

GPIOPinWrite(GPIO\_PORTN\_BASE, GPIO\_PIN\_0, 0);

if(!GPIOPinRead(GPIO\_PORTJ\_BASE, GPIO\_PIN\_1))

GPIOPinWrite(GPIO\_PORTN\_BASE, GPIO\_PIN\_1, GPIO\_PIN\_1);

else

GPIOPinWrite(GPIO\_PORTN\_BASE, GPIO\_PIN\_1, 0);

# 实验2 A2000TM4扩展板使用方法与SYSTICK中断

## 实验目的

熟悉LED显示和键盘管理芯片TM1638的功能，学会通过TM4C1294芯片程控该芯片的方法；

初步掌握简易的人机操作界面的设计技巧（利用数码管、LED指示灯、按键阵列等）；

熟悉SYSTICK中断调用方式，掌握相应的编程技能；

学习和掌握在无操作系统支持的条件下，模拟多任务切换式处理的程序设计思想和方法。

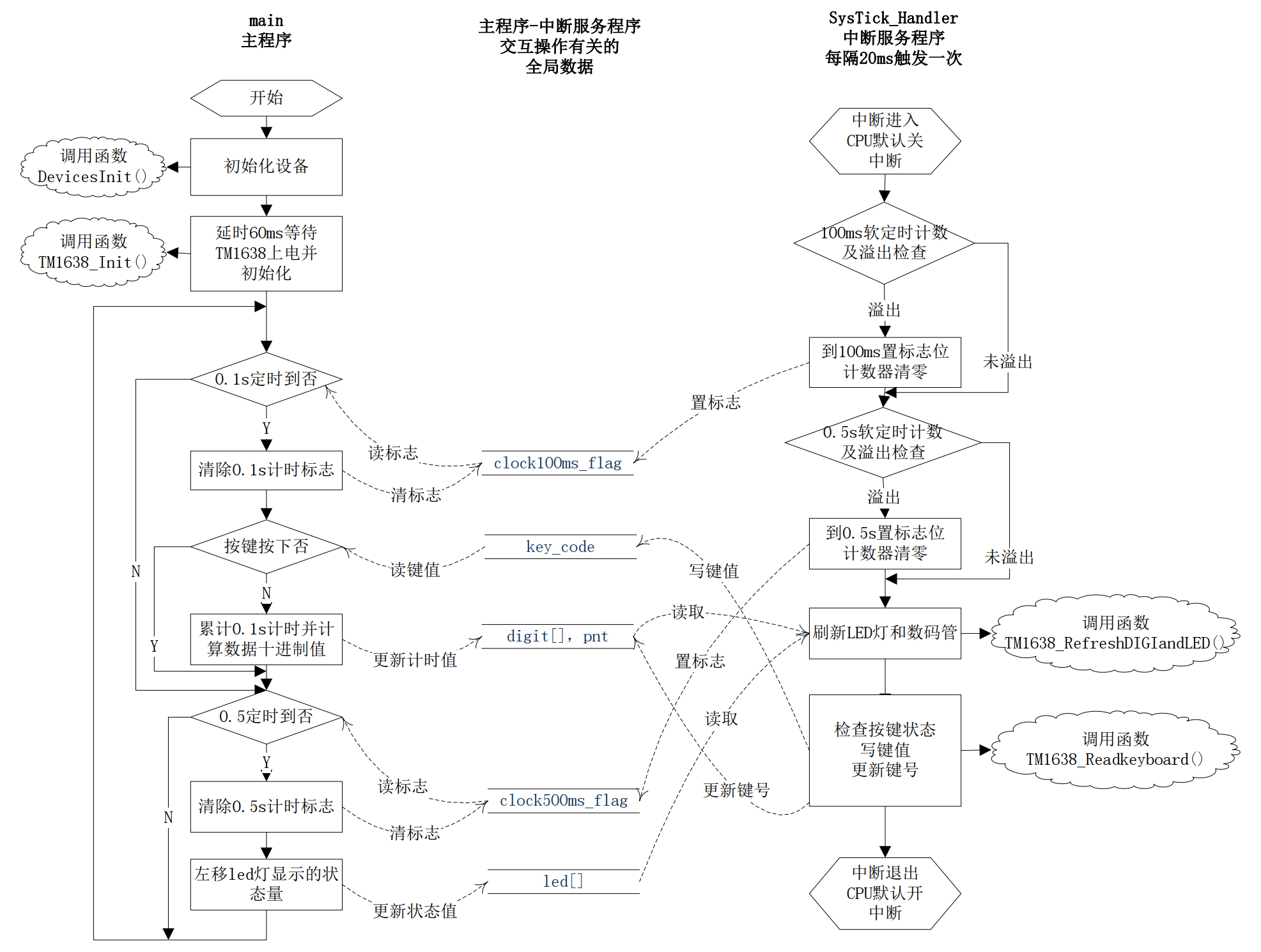
## 实验主要器材和设备

电脑，TM4C1294NCPDT实验板卡，A2000TM4扩展板，数字示波器。

## 实验任务的技术解决方案说明

整体而言，实验二的4项实验项目环环相扣，逐层递进。其中，“中断”是实验二的核心词。由于电子工程系自制的A2000TM4扩展板的按键阵列布置的是机械键盘，因而就难以避免“按键去抖”问题。

本项实验的基础是实验例程exp2\_0.c文件，其算法流程、变量设置等细节如图1所示。其中，“按键去抖”环节位于Systick\_Handler中断服务程序中进行，这是本项实验的重点。



**图1 实验2例程代码流程结构图]**

### 实验任务2\_1

#### 任务概览

实验任务2\_1要求我们在例程exp2\_0.c的基础上进行改进，让系统呈现出3种不同的工况，具体模式如表2所示。模式0为开机或复位后的默认模式，当用户按下键盘阵列中的任意键盘时，系统工况发生一次转变。

**表2 任务2\_1功能要点**

|  |  |
| --- | --- |
| 功能模式 | 功能要点 |
| 0 | 预期功能一：原功能1和功能3有效；功能2暂停（LED灯静止不闪） |
| 1 | 预期功能一：原功能1暂停（计时）；原功能3有效；  预期功能二：8个指示灯以走马灯方式，每0.5秒向右（循环）移动一格 |
| 2 | 预期功能一：原功能1暂停（计时）；原功能3有效；  预期功能二：8个指示灯以走马灯方式，每0.5秒向左（循环）移动一格 |

#### 实现思路

实验任务2\_1的重点在于模式的切换，由于模式的切换由按键的次数有关，因而我们选择设置一个初始值为0的全局变量time用来记录按键次数，每按下一次time数加1。当time对3取模为0，则系统进入工况0；若模值为1，则工况为1；若模值为2，则工况为2。三种工况在main函数中用C语言中经典的switch语句来选择切换。

与此同时，我们需要判断按键的按下与放开。利用TM4C1294自带的key\_code函数，我们可以判断键盘当前按下的键值，若无键被按下，则该值为0。具体判断思路如下所示：若上次该值0，且本次非0，则表示一次按下过程；若上次非0，且本次为0，则表示一次放开。代码实现部分见2.5.1。

### 实验任务2\_2

#### 任务概览

实验任务2\_2继续在例程exp2\_0.c的基础上进行改进，与实验任务2\_1最大的区别在于增加了按键，赋予了按键不同的功能。具体要求如表3所示：

**表3 任务2\_2功能要点**

|  |  |
| --- | --- |
| 功能序号 | 功能要点 |
| 1 | 原功能1继续有效，且受功能3影响 |
| 2 | 若正常计时，LED[7] = 1；否则，LED[7] = 0 |
| 3 | digit[5] = key\_code |
| 4 | 按一次1号键，切换“启动”/“暂停” |
| 5 | 按一次2号键，计数值清零 |

#### 实现思路

在中断服务程序SysTick\_Handler中，我们需要判断按键是否按下以及按键的种类。判断按键是否按下的过程与2.3.1.2中所阐述的一致。同时，我们利用key\_code判断按下的键值，若键值是0，则time值加1，进行功能的切换；若按下的是2，将计数值、数码管清零。

在main函数中，我们对中断服务程序传入的time值进行处理。若按键按下的次数是2的倍数，这说明现工况为启动计时，LED[7]点亮；否则就是暂停，熄灭LED[7]。简单的if-else语句即可实现该功能。

实验任务2\_3的具体代码实现在2.5.2 部分。

### 实验任务2\_3

#### 任务概览

实验任务2\_3的代码基本框架与2\_2完全一致。唯一的区别在于将加法计时改为减法计时，把计时初值从0000改为9999ms，计满1000秒复位从头再来。

#### 实现思路

修改2\_2中的相关初值与复位条件即可。具体代码实现在2.5.3部分。

### 实验任务2\_4

#### 任务概览

实验任务2\_4是2\_2和2\_3的综合应用，3号按键用来切换加法计时与减法计时，默认计时状态为减计时。与此同时，任务要求书中取消了数码管显示“键值” 的功能，用左侧四位数码管显示计时模态，用“\_UU\_”表示加法计时，显示“\_AA\_”表示减法计时。

#### 实现思路

鉴于任务2\_4是前述两个任务的综合，我们仅需加上按键3的部分代码，适当修缮数码管显示处的代码即可。

在中断服务程序SysTick\_Handler中，我们需要加上一段来判断按键 3的按下与否。将按键3按下的次数保存到全局变量time2中，并将time2模2的两种值作为main函数中分辨加法计时还是减法计时的依据。若time2是偶数，则运行任务2\_3中的部分代码，并将数码管的显示置为“\_AA\_”；

反之，则运行2\_2中部分代码，并把数码管置为“\_UU\_”。具体代码实现见2.5.4。

## 实验结果自测记录

### 实验2\_1实验现象

（1）当任意按下一个键时，能够触发一次模式转换；按住不放不会连续改变工作模式，放开动作也不影响工作模式，直至放开后的下一次按键，才能够再次触发转变；

（2）能够实现表2中所述模式0、1、2的既定功能。

### 实验2\_2实验现象

（1）正常计时时，最左边LED灯亮；计时暂停时，最左边的LED灯灭；

（2）按一次1号键，切换一次暂停或计时状态；

（3）按一次2号键，计数清零；

（4）左数第2个数码管显示键值。

### 实验2\_3实验现象

（1）计时从999.9s开始向下计时，计满后重新复位到“9”“9”“9”“9”。

### 实验2\_4实验现象

（1）3号键能够切换控制加减计时模式，默认计时状态为减；

（2）当加法计时时，左边4位数码管显示为“\_UU\_”；减法计时时，左边4为数码管显示为“\_AA\_”。

## 实验核心代码清单

### 实验2\_1核心代码

static uint8\_t ui8KeyCurState = 0;

uint8\_t ui8KeyPreState = ui8KeyCurState;

key\_code = TM1638\_Readkeyboard();

if (key\_code == 0) ui8KeyCurState = 0;

else ui8KeyCurState = 1;

if ((ui8KeyPreState == 0) && (ui8KeyCurState != 0))

{

++time;

}

### 实验2\_2核心代码

if ((prekey\_code != 1) && (key\_code == 1))

{

++time;

}

if ((prekey\_code != 2) && (key\_code == 2))

{

test\_counter = 0;

digit[0] = 0;

digit[1] = 0;

digit[2] = 0;

digit[3] = 0;

}

### 实验2\_3核心代码

if (--test\_counter <= 0) test\_counter = 9999;

……

if ((prekey\_code != 2) && (key\_code == 2))

{

test\_counter = 9999;

digit[0] = 9;

digit[1] = 9;

digit[2] = 9;

digit[3] = 9;

}

### 实验2\_4核心代码

if ((prekey\_code != 3) && (key\_code == 3))

{

++time2;

time1 = 0;

}

……

int main(void)

{

DevicesInit();

while (clock100ms < 3);

TM1638\_Init();

while (1)

{

if (time2%2 == 0)

{

digit[5] = 'A';

digit[6] = 'A';

if (time1%2 == 0)

{

if (clock100ms\_flag == 1)

{

clock100ms\_flag = 0;

if (--test\_counter <= 0) test\_counter = 9999;

digit[0] = test\_counter / 1000;

digit[1] = test\_counter / 100 % 10;

digit[2] = test\_counter / 10 % 10;

digit[3] = test\_counter % 10;

}

led[7] = 1;

}

else led[7] = 0;

}

else

{

digit[5] = 'U';

digit[6] = 'U';

if (time1%2 == 0)

{

if (clock100ms\_flag == 1)

{

clock100ms\_flag = 0;

if (++test\_counter >= 10000) test\_counter = 0;

digit[0] = test\_counter / 1000;

digit[1] = test\_counter / 100 % 10;

digit[2] = test\_counter / 10 % 10;

digit[3] = test\_counter % 10;

}

led[7] = 1;

}

}

}

}

# 实验3 UART串行通讯实验

## 实验目的

了解UART串行通讯的工作原理；

掌握在PC端通过串口调试工具与实验板通过UART通讯的方法；

掌握UART的堵塞式与非堵塞式通讯方法。

## 实验主要器材和设备

电脑，TM4C1294NCPDT实验板卡，A2000TM4扩展板，串口调试软件，数字示波器。

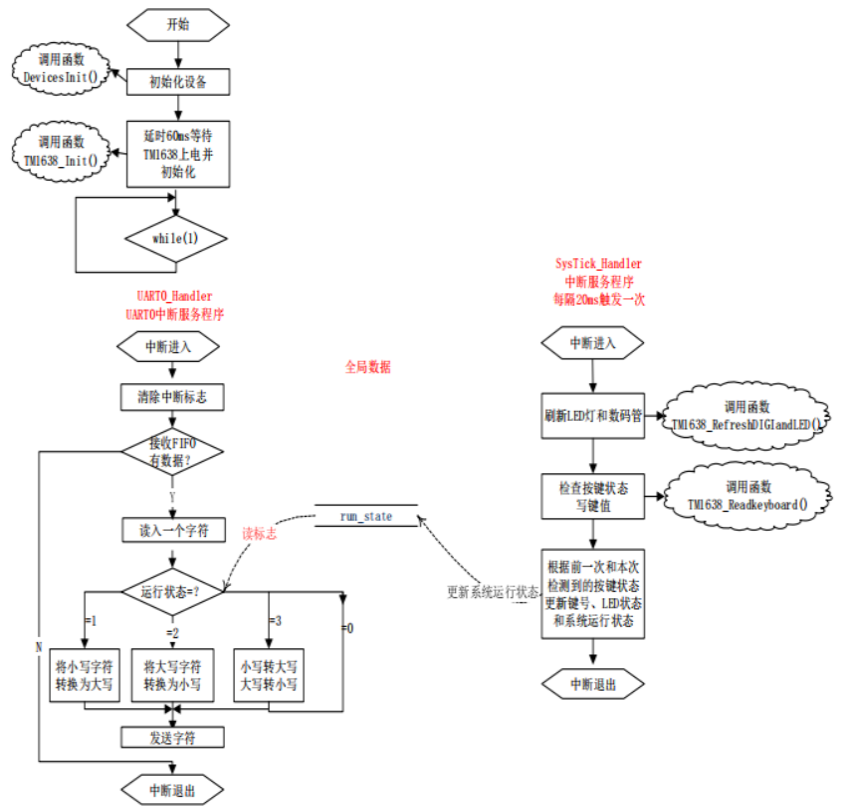
## 实验任务的技术解决方案说明

### 实验任务3\_1

按照实验要求，我们把 exp3\_0.c改写成非阻塞方式，即利用中断的方式进行数据的发送与接收。

首先，我们了解了UART0的波特率、帧格式的设置方式，即利用UARTConfigSetExpClk函数设置。从源代码exp3\_0.c中可以看到，默认波特率是115200bps，帧格式为8个数据位+1个终止位。

其次，我们将实验任务书中的UART0 中断服务程序 void UART0\_Handler(void)复制到例程中，又删除了while循环中的循环体，完成本实验任务。



**图2 实验任务3\_1代码实现流程图**

### 实验任务3\_2

#### 任务概览

实验任务3\_2要求我们在已经掌握3\_0和3\_1的基础上继续进行延伸研究，要求我们利用编程实现一个24小时制的时钟。具体要求为：默认状态下，6 位数码管显示 00.00.00；接收形如“AT+SETXX:XX:XX”格式的对时命令时，能够准确显示；接收形如“AT+INCXX:XX:XX”格式的加时命令时，能够在现有时刻上加上该时间；接收形如“AT+GET”的命令时，能显示当前时间；此外，还需要对语句的有效性进行监测，若有误需返回“Error Command!”。

#### 实验思路

该部分我们需要手动实现几个自定义程序。对于默认归零部分，我们只需设置数组digit[8]与Time为0即可，把对应的数码管值输送到对应位置；对于对时命令部分，为了表述方便及符合我们日常习惯，我们首先增加一个功能函数digitWrite，使得数组序号与数码管序号对应；对于绝对对时、相对对时功能，我们分别设置SetTime、IncTime函数，将板子时间用输入指令的值代替，或把需要增加的时间转成秒数进行操作；对于查询时间部分，我们定义一个长为9的字符型数组，对每一个数组单元赋值时间的对应字符；最后是判断输入语句正确与否的代码，判断语法正确性的指标有两个，一是时分秒间的 “:”隔开；二是时分秒的大小限制。具体代码实现详见3.5.2节。

### 实验任务3\_3

#### 任务概览

实验任务3\_3是实验任务3\_2的延续，任务说明书要求我们实现一个24小时制的时钟。默认状态下，6 位数码管显示 00.00.00； MCU 每隔 1 秒钟向 PC 端发送“现在是中华人民共和国北京时间”的字样和当前时间数值。此外，我们需要根据要求修改UART的波特率，观察PC端的显示情况。

#### 实现思路

该任务相对来说较为简单，我们直接在main函数里的while循环体里增写代码。时钟的实现部分的思路参考了实验3\_2部分；然后我们将字符串转换为时间，把对应数值显示在数码管上并返回到串口软件。

## 实验结果自测记录

### 实验3\_2实验现象

（1）开机复位后， 6 位数码管显示 00.00.00并开始加计时，串口软件显示“Hello, 2A!”；

（2）发送指令“AT+SET00:01:11”后，数码管显示时间为00:01:11并继续计时；

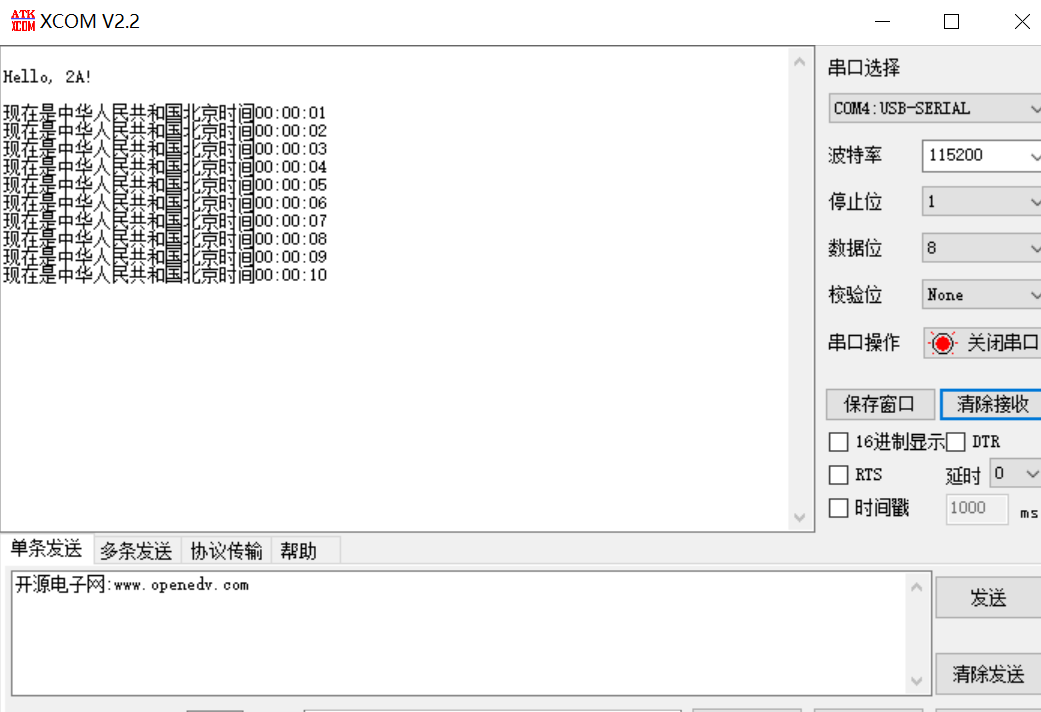
（3）发送指令“AT+INC00:00:12”后，数码管对应的显示时间加上了 12 秒，串口软件显示进行加操作后的时间；

（4）发送指令“AT+GET”后，串口软件返回数码管显示时间；

（5）发送指令“AT+INC26:00:21”、 “AC”等错误指令后，窗口显示“Error Command!”。

### 实验3\_3实验现象

（1）复位后， 6 位数码管显示 00.00.00并开始加计时，串口软件每1秒接收当前时间信息，如图3所示。



**图3 实验任务3\_3串口软件显示**

（2）我们按照表4数据修改波特率，对应结果及相关分析如表4后两栏所示。

**表4改变UART波特率时的实验结果**

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 波特率 | 结果是否正确 | 原因 |
| 1 | 230400 | 正确 | 通信速率足够 |
| 2 | 115200 | 正确 | 通信速率足够 |
| 3 | 9600 | 略慢，偶尔会忽略报时 | 通信速率尚可 |
| 4 | 1200 | 可见逐字显示 | 通信速率较低 |
| 5 | 300 | 逐字显示的现象更加明显，会忽略、误判对时命令 | 通信速率低，1秒内发不完报时信息 |
| 6 | 110 | 显示速度更慢，逐字显示变成每几个字一起显示 | 传输速度过低 |

## 实验核心代码清单

### 实验3\_2核心代码

void SysTick\_Handler(void)

{

if (++clock1000ms >= V\_T1000ms) {

clock1000ms = 0;

board\_time = (board\_time + 1) % 86400;

}

digitWrite(0, board\_time/3600/10);

digitWrite(1, board\_time/3600%10);

digitWrite(2, board\_time%3600/60/10);

digitWrite(3, board\_time%3600/60%10);

digitWrite(4, board\_time%3600%60/10);

digitWrite(5, board\_time%3600%60%10);

TM1638\_RefreshDIGIandLED(digit, pnt, led);

}

void UART0\_Handler(void)

{

int i;

uint32\_t uart0\_int\_status;

char cmd[10], time[10];

if\_read = !if\_read;

if (!if\_read) return;

uart0\_int\_status = UARTIntStatus(UART0\_BASE, true);

UARTIntClear(UART0\_BASE, uart0\_int\_status);

UARTStringGet(UART0\_BASE, str);

// UARTStringPut(UART0\_BASE, str);

for (i = 0; i < 6; ++i) {

cmd[i] = str[i];

}

cmd[6] = '\0';

// UARTStringPut(UART0\_BASE, (const char \*)"command:");

// UARTStringPut(UART0\_BASE, cmd);

for (i = 6; i < 14; ++i) {

time[i-6] = str[i];

}

time[8] = '\0';

// UARTStringPut(UART0\_BASE, (const char \*)"time:");

// UARTStringPut(UART0\_BASE, time);

if (strCmp(cmd, "AT+GET") != 0){

UARTStringPut(UART0\_BASE, (const char \*)"\nGet:");

GetTime(UART0\_BASE);

}

else if (isTimeLegal(time) == 0) {

UARTStringPut(UART0\_BASE, (const char \*)"\nError Command!");

}

else if (strCmp(cmd, "AT+SET") != 0) {

UARTStringPut(UART0\_BASE, (const char \*)"\nSet:");

SetTime(UART0\_BASE, time);

}

else if (strCmp(cmd, "AT+INC") != 0) {

UARTStringPut(UART0\_BASE, (const char \*)"\nInc:");

IncTime(UART0\_BASE, time);

}

else UARTStringPut(UART0\_BASE, (const char \*)"\nError Command!");

}

void digitWrite(int pos, int num){

if (pos < 4) digit[pos+4] = num;

else digit[pos -4] = num;

}

void SetTime(uint32\_t ui32Base, char\* s){

int hour, min, sec;

// char temp[3];

UARTStringPut(UART0\_BASE, s);

hour = 10 \* (s[0] - '0') + s[1] - '0';

min = 10 \* (s[3] - '0') + s[4] - '0';

sec = 10 \* (s[6] - '0') + s[7] - '0';

board\_time = 3600 \* hour + 60 \* min + sec;

// temp[0] = (char) hour / 10 + '0';

// temp[1] = (char) hour % 10 + '0';

// temp[3] = '\0';

// UARTStringPut(UART0\_BASE, temp);

// GetTime(ui32Base);

}

void IncTime(uint32\_t ui32Base, const char\* addition){

int hour = 10 \* (addition[0] - '0') + addition[1] - '0';

int min = 10 \* (addition[3] - '0') + addition[4] - '0';

int sec = 10 \* (addition[6] - '0') + addition[7] - '0';

board\_time += 3600 \* hour + 60 \* min + sec;

GetTime(ui32Base);

}

void GetTime(uint32\_t ui32Base){

int hour = board\_time / 3600 % 24;

int min = board\_time % 3600 / 60;

int sec = board\_time % 3600 % 60;

char s[9];

s[0] = hour / 10 + '0';

s[1] = hour % 10 + '0';

s[2] = ':';

s[3] = min / 10 + '0';

s[4] = min % 10 + '0';

s[5] = ':';

s[6] = sec / 10 + '0';

s[7] = sec % 10 + '0';

s[8] = '\0';

UARTStringPut(UART0\_BASE, s);

}

bool strCmp(const char\* s1, const char\* s2){

while (\*s1 != '\0' && \*s2 != '\0'){

if (\*s1 != \*s2){

return false;

}

++s1;

++s2;

}

if (\*s1 != '\0' || \*s2 != '\0') {

return false;

}

return true;

}

bool isTimeLegal(const char \*s){

int hour, min, sec;

if (s[2] != ':' || s[5] != ':') return false;

hour = 10 \* (s[0] - '0') + s[1] - '0';

min = 10 \* (s[3] - '0') + s[4] - '0';

sec = 10 \* (s[6] - '0') + s[7] - '0';

if (hour >= 0 && hour <= 23 &&

min >= 0 && min <= 59 &&

sec >= 0 && sec <= 59) return true;

else return false;

}

### 实验3\_3核心代码

while (1)

{

if (clock1000ms\_flag == 1)

{

clock1000ms\_flag = 0;

if (++test\_counter >= 86400) test\_counter = 0;

hour=test\_counter /3600;

minute=test\_counter % 3600 / 60;

second=test\_counter % 3600 % 60;

digit[4] = hour / 10;

digit[5] = hour % 10;

digit[6] = minute / 10;

digit[7] = minute % 10;

digit[0] = second / 10;

digit[1] = second % 10;

UARTStringPut(UART0\_BASE, (const char \*)"\r\n现在是中华人民共和国北京时间");

UARTCharPut(UART0\_BASE,(char)(digit[4]+'0'));

UARTCharPut(UART0\_BASE,(char)(digit[5]+'0'));

UARTCharPut(UART0\_BASE,':');

UARTCharPut(UART0\_BASE,(char)(digit[6]+'0'));

UARTCharPut(UART0\_BASE,(char)(digit[7]+'0'));

UARTCharPut(UART0\_BASE,':');

UARTCharPut(UART0\_BASE,(char)(digit[0]+'0'));

UARTCharPut(UART0\_BASE,(char)(digit[1]+'0'));

}

}

# 附录 学习心得和意见建议

## 实验1学习思考

### 实验1任务说明书中三道思考题的回答

【问题1】将经过PLL 后的系统时钟调整至最大值120MHz，观察人眼感受到的LED 灯的工作效果，是否还能清晰辨别其存在亮-灭闪烁？就你的实验观测，人眼能较清晰辨别的LED 灯亮-灭闪烁的最高频率大约为多少？

【回答】不能，这是由于视觉暂留现象。经实验，我们发现大概在大几十Hz人眼就已经不能清晰识别，60Hz以上的闪烁频率人眼基本上就难以察觉；当频率上升至80MHz时，仅能感受到灯的明暗微弱变化。在生活中，这个现象的应用非常广泛。例如，荧光灯以100Hz闪烁，但人眼根本察觉不出。

【问题2】实验1\_1 最后的外部可见功能时，按下USR\_SW1-PJ0，有时要经过明显迟滞，LED 灯才能切换为快闪，这是为什么？（提示：后续实验将提供避免这种迟滞的技术方案）

【回答】在1.4.1（3）节有简要回答。具体来说是因为在本程序中，初始化后进入while死循环。在循环体中，先读取按键值，调用完PF0Flash函数后再读取键值。若在运行PF0Flash时按下按键，则不能即使响应。此现象可以通过采用中断函数的方式来改良。

【问题3】实验1\_1 中关于表1-1 的系统时钟设置操作，特意安排了几种情况，观测实际得到的时钟工作频率与希望设定的系统时钟频率无法正确对应。换言之，系统时钟频率设定取值并不能“随心所欲”。请结合MCU 厂商技术资料，定性分析其中可能的原因。

【回答】在1.4.1（1）节有简要回答。具体说来，一部分谬误产生的原因是因为系统分频时只能除以1-1024间的整数。另一方面，是因为系统本身的局限性。在01-Tiva™ TM4C1294NCPDT Microcontroller Data Sheet第238页Table 5-7Actual PLL Frequency下有这样一条注释：*For a given crystal frequency, N should be chosen such that the reference frequency is within 4 to 30 MHz.*由此可见，当利用PLL设定频率时，频率过大、过小都是不合时宜的。

### 实验1的总结

综上，在实验1中，我们熟悉了KEIL uVision5开发环境；大致掌握了系统时钟控制原理，了解了频率的取值限制；大致了解了GPIO的部分相关函数，掌握了利用按键控制LED的原理与应用；熟悉了DEBUG功能与示波器的使用。整体而言，实验1是ARM编程实战的初探，为后续4个实验奠定了良好基础。

## 实验2学习思考

（1）实验2在实验1的基础上进行了延拓，在实验1中，我们初步了解了TM4C1294控制的TM1638芯片，在实验2中，我们更进一步，学习通过MCU管理A2000TM4扩展板，例如LED灯、小数点、数码管与扩展按键。利用LED灯、小数点、数码管与扩展按键等硬件资源，我们初步感知了人机交互，对简易人机交互界面的设计有了直观认识。

（2）在上半学期《嵌入式系统与接口技术》课程中，我们初步接触到了中断，并掌握了汇编语言8086架构下的中断实现。在实验2中，我们在ARM架构下学习使用Systick中断调用方式，掌握了相关技能，从而实现了相对准确的软定时功能。

## 实验3学习思考

实验3相对来说难度稍大，对我们的代码实现能力提出了更高的要求。例如任务3\_2中，SetTime、IncTime、GetTime等函数的实现，虽然任务明晰、目标明确，但落到实处还是有难度的，尤其是一些细节的把控，常常一不注意就翻车，这是对我们耐心、细心程度以及综合能力的考验。

## 关于《工程实践与科技创新II-A》基础实验部分的整体感受

《工程实践与科技创新II-A》是电子工程系“工程实践与科技创新”系列课程的第二阶段，要求我们综合使用电路焊装/连接技术、编程技术、软件调试工具、专业仪表设备等，能够开发出较复杂工程装置系统。

整体而言，前三个实验相对后两个来说，任务要求较低、时间期限较长，但依然给我们带来了不一样的感受，体会到了很多，例如组内磨合、时间统筹等。这些为后期实验四、实验五的顺利开展奠定了基础。

在此，我们非常感谢电子系各位老师的耐心讲解，包括线下授课、canvas视频的录制与答疑；非常感谢电院提供的实验场地，为组内交流协作提供了极为便利的平台。

当然，我们对课程的设置有一点小小的建议：建议理论课可以多开几节，或者6节理论课可以分散开来，就像《工程问题建模与仿真》一样，根据学生的进度来安排线下理论授课，而不是全部集中在前三周，否则，老师讲的内容会远远超前于学生当前正在做的。在这种情况下，一方面，学生听不太懂；另一方面，老师也付出了努力，这就是典型的“吃力不讨好”。希望老师们以后可以适当考虑考虑我们的建议。

最后，再次感谢各位老师的付出。