目 录

1. 实验1 时钟选择与GPIO操作 1

1.1 实验目的 1

1.2 实验主要器材和设备 1

1.3 实验任务的技术解决方案说明 1

1.3.1 实验任务1\_1 1

1.3.2 实验任务1\_2 3

1.4 实验结果自测记录 4

1.5 实验核心代码清单 4

1.5.1 实验任务1\_1 4

1.5.2 实验任务1\_2 5

1.6 实验小结 5

2. 实验2 A2000TM4扩展板使用方法与SYSTICK中断 7

2.1 实验目的 7

2.2 实验主要器材和设备 7

2.3 实验任务的技术解决方案说明 7

2.3.1 实验任务2\_1 7

2.3.2 实验任务2\_2 8

2.3.3 实验任务2\_3 8

2.3.4 实验任务2\_4 9

2.4 实验结果自测记录 9

2.5 实验核心代码清单 9

2.5.1 实验任务2\_1 9

2.5.2 实验任务2\_2 10

2.5.3 实验任务2\_3 12

2.5.4 实验任务2\_4 15

2.6 实验小结 19

3. 实验3 UART串行通讯实验 20

3.1 实验主要器材和设备 20

3.2 实验任务的技术解决方案说明 20

3.2.1 实验任务3\_1 20

3.2.2 实验任务3\_2 20

3.2.3 实验任务3\_3 21

3.3 实验结果自测记录 22

3.4 实验核心代码清单 22

3.4.1 实验任务3\_1 22

3.4.2 实验任务3\_2 25

3.4.3 实验任务3\_3 32

3.5 实验小结 38

4. 参考文献 39

5. 附录 学习心得和意见建议 40

# 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

#### 任务要求

修改程序exp0.c，另存命名为exp1\_1.c，完成以下任务：

1) 分别使用16M内部精密时钟（PIOSC）、25M外部时钟（MOSC）以及PLL时钟，作为程控延时等待的时钟源信号，在本程序中调节D4-PF0闪烁频率的快慢；

2) 进入集成开发环境KEIL uVision5的DEBUG模式，设置断点，显示和观察时钟频率返回值，辅助验证程序功能；

3) 使用示波器观测PF0引脚的信号波形，测量该信号频率，辅助验证程序功能；

4) 实现如下外部可见功能：初始时，D4-PF0以大约6秒为周期缓慢闪烁；当按下USR\_SW1-PJ0，D4-PF0以大约100毫秒为周期快速闪烁；松开USR\_SW1-PJ0，D4-PF0恢复以6秒为周期缓慢闪烁。

#### 实现思路

从技术文档中获取需要的信息，包括：

（1）TM4C1294NCPDT微控制器的Datasheet[1]的第5.2.5节提供的有关系统时钟控制的信息；

（2）Tiva系列MCU的外设驱动库（Peripheral Driver Library）用户手册[2]的第26.2节介绍的与系统时钟设置相关的API函数的用法；

（3）EK-TM4C1294XL板卡的用户手册[3]给出的插针（Pin）-引脚（Port）对应关系，以及板卡上的按键、LED灯等资源与端口的连接关系；

（4）课程自学材料和实验例程[4][5]给出的补充信息和编程思路。

#### 实验结果和分析

我们根据上述技术文档，结合本课程提供的自学材料，完成了实验任务。

为了探究系统时钟频率的设定方式对PF0信号频率的影响，我们进行了一系列实验，结果示于表1.1和图1.1。设计实验时，控制的变量有振荡器源（MOSC或PIOSC）和系统时钟源（OSC或PLL），共有四种组合方式，相应地可将这些实验分为四组。

观察实验现象，结合表1.1的数据，我们得出了以下结论。

1. 希望设定的时钟频率未必可实现。

查阅技术文档后，我们认为，这是因为实际的系统时钟是由系统时钟源（PLL或VCO）经硬件分频等处理后得到的（参见Datasheet[1]的第5.2.5节），这种机制决定了分频系数不能任意。分频系数必须为整数，例如实验序号5、7不具有整数分频系数，因而无法实现。然而，即使希望设定的系统时钟频率具有整数的分频系数，也未必可实现，如实验序号12就是反例。事实上，驱动库手册[2]的第26.1节已指出*When using the PLL, the input clock frequency is constrained to specific frequencies that are specified in the device data sheet*.

2. DEBUG观测到的时钟频率与实际的时钟频率成正比，比例系数可能与振荡器源和系统时钟源有关。

从表1.1可见，在实验序号第1至3和第7至12中，比例系数

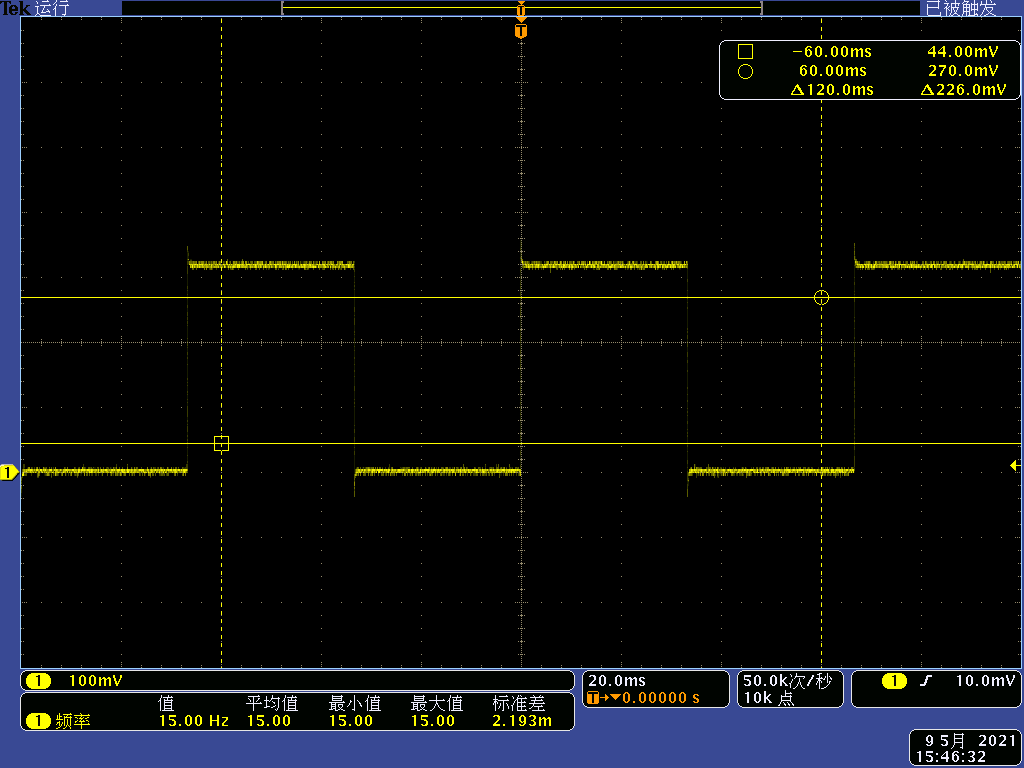


而在实验序号第4至6中，该比例系数约为2.5。

根据程序流程，可认为D4-PF0的每个亮-灭周期包含相同数量的系统时钟周期，于是示波器观测的PF0频率正比于系统时钟频率。若DEBUG观测到的时钟频率与实际的时钟频率相同，则比例系数应是一个与振荡器源和系统时钟源无关的常数，然而实验结果表明并非如此。要解释这种现象，仍需查阅更多资料。

**表1.1 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 | 5.0 |
| 4 | √ | √ |  |  | √ |  |  | 25 | 25 | 10.0 |
| 5 | √ | √ |  |  | √ |  |  | 12 | 12.5 | 5.0 |
| 6 | √ | √ |  |  | √ |  |  | 1 | 1 | 0.4 |
| 7 | √ | √ |  | √ |  |  | √ | 25 | 24 | 15.0 |
| 8 | √ | √ |  | √ |  |  | √ | 20 | 20 | 12.5 |
| 9 | √ | √ |  | √ |  |  | √ | 8 | 8 | 5.0 |
| 10 |  |  | √ | √ |  |  | √ | 20 | 20 | 12.6 |
| 11 |  |  | √ | √ |  |  | √ | 8 | 8 | 5.0 |
| 12 |  |  | √ | √ |  |  | √ | 1 | 15 | 9.5 |

**图1.1 示波器观测的PF0信号。左图：实验序号1，右图：实验序号7**

3. 系统时钟（SYSCLK）频率越高，软件延时的时长越短。

我们认为，这是因为软件延时的原理是通过执行空循环体，用多个指令周期消耗时间；时钟频率越高，MCU的时钟周期、总线周期和指令周期就越短，所以延时的时长就越短。

4. 软件延时机制导致MCU对按键按下信号的响应出现明显迟滞。

我们注意到，在实验任务1\_1最后的外部可见功能中，按下USR\_SW1-PJ0，有时要经过明显迟滞，LED灯才能切换为快闪。我们认为，这是由于软件延时导致的。从前述软件延时原理可见，这是一种阻塞式延时：在延时期间，MCU一直在执行空循环体，只有在延时结束后才可响应按键信号。例如，若按下USR\_SW1-PJ0的时刻正好位于长延时周期的初始，将会导致约一个长延时的迟滞时间。

要解决响应迟滞的问题，可引入中断机制模拟多任务处理，详见后续章节将介绍的实验2和实验3。

### 实验任务1\_2

#### 任务要求

修改程序代码并另存命名为exp1\_2.c，完成实验任务1\_2。

外部可见功能要求如下：

(i) 当按下USR\_SW1-PJ0时，点亮D1-PN1；放开时，熄灭D1-PN1；

(ii) 当按下USR\_SW2-PJ1时，点亮D2-PN0；放开时，熄灭D2-PN0。

#### 实现思路

实验任务1\_1已实现了通过USR\_SW1-PJ0控制D4-PF0的功能，类似地可实现本实验任务。

使用SysCtlPeripheralEnable、SysCtlPeripheralReady等API函数使能端口N，用GPIOPinTypeGPIOOutput函数将PN0、PN1配置为输出引脚，用GPIOPinTypeGPIOInput函数将PJ0、PJ1设置为输入引脚，用GPIOPadConfigSet函数将PJ0、PJ1引脚设置为“推挽上拉”类型。各API函数的用法参见驱动库手册[2]，或源文件sysctl.c的注释。

在主循环体中，用以下语句实现USR\_SW1-PJ0、USR\_SW2-PJ1对D1-PN1、D2-PN0的控制：

// 读取 PJ0、PJ1 键值. 0-按下 1-松开. Any bit that is not specified by ui8Pins is returned as a 0.

ui8KeyValue = GPIOPinRead(GPIO\_PORTJ\_BASE, GPIO\_PIN\_0 | GPIO\_PIN\_1);

// 若 PJ0/1 为 1（松开），则熄灭 PN1/0 （写入0）；否则.

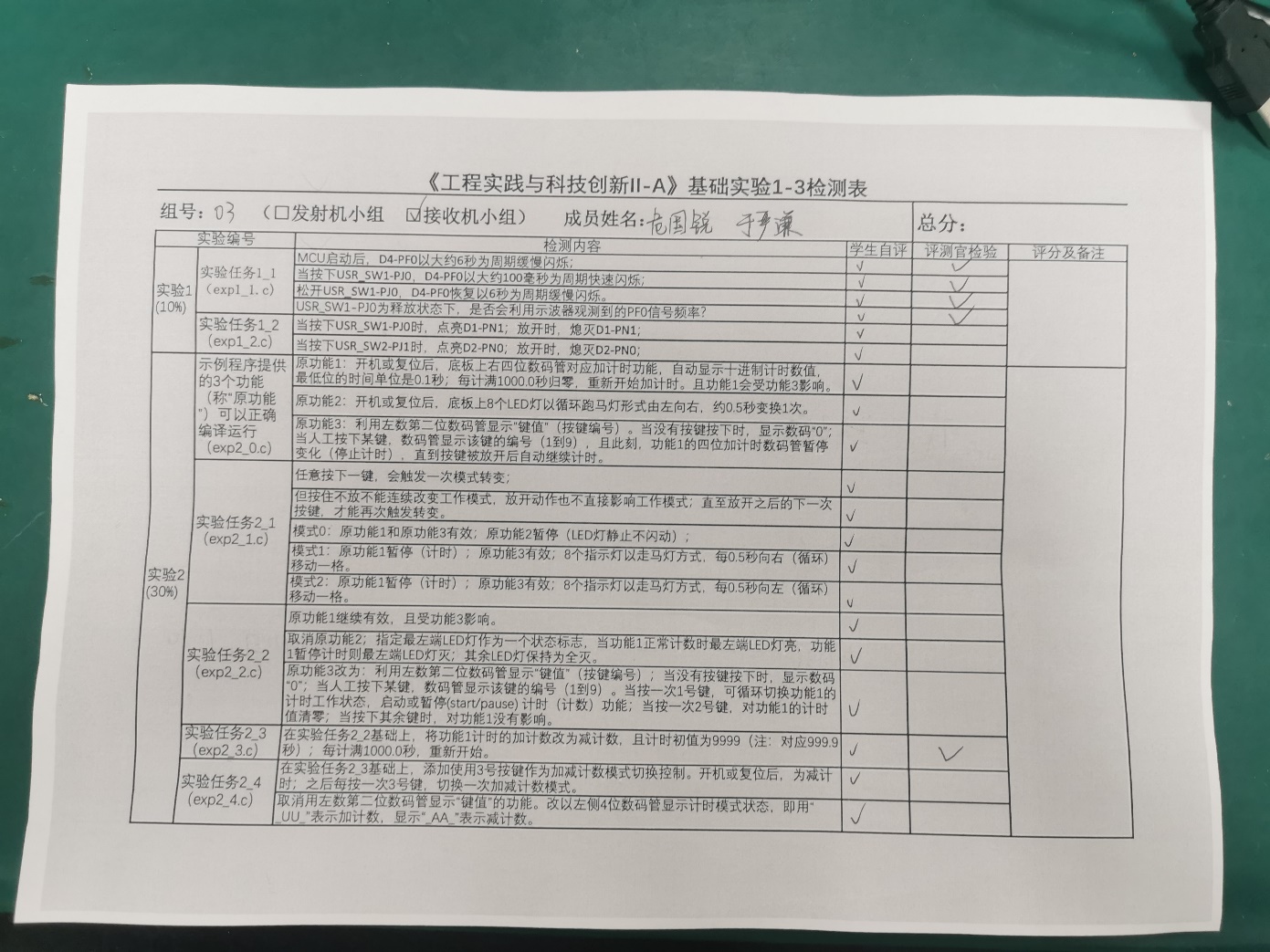
GPIOPinWrite(GPIO\_PORTN\_BASE, GPIO\_PIN\_0 | GPIO\_PIN\_1, ~((ui8KeyValue << 1) | (ui8KeyValue >> 1)));

#### 实验结果和分析

实验任务已正确实现。

在驱动库的API函数中，常用bit-packed方式表示引脚及各种状态，例如本实验使用的int32\_t GPIOPinRead(uint32\_t ui32Port, uint8\_t ui8Pins) 函数。利用C语言提供的移位操作，我们写出了上述简单清晰的与按键-LED控制相关的语句。

## 实验结果自测记录



## 实验核心代码清单

### 实验任务1\_1

|  |  |
| --- | --- |
| 1 | //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* |
| 2 | // |
| 3 | // 函数原型：void PF0Flash(uint8\_t ui8KeyValue) |
| 4 | // 函数功能：根据传入的按键值，决定PF0快闪或慢闪. 0-快闪，1-慢闪 |
| 5 | // 函数参数：ui8KeyValue：按键值 |
| 6 | // |
| 7 | //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* |
| 8 | void PF0Flash(uint8\_t ui8KeyValue) |
| 9 | { |
| 10 | uint32\_t ui32DelayTime; |
| 11 |  |
| 12 | if (ui8KeyValue == 0) // PUSH1(USR\_SW1-PJ0) 按下（低有效） |
| 13 | ui32DelayTime = FASTFLASHTIME; |
| 14 | else // PUSH1(USR\_SW1-PJ0) 松开 |
| 15 | ui32DelayTime = SLOWFLASHTIME; |
| 16 |  |
| 17 | GPIOPinWrite(GPIO\_PORTF\_BASE, GPIO\_PIN\_0, GPIO\_PIN\_0); // 点亮 LED4(D4-PF0) |
| 18 | DelayMilliSec(ui32DelayTime);                          // 延时ui32DelayTime毫秒 |
| 19 |  |
| 20 | GPIOPinWrite(GPIO\_PORTF\_BASE, GPIO\_PIN\_0, 0x0); // 关闭 LED4(D4-PF0) |
| 21 | DelayMilliSec(ui32DelayTime);                   // 延时ui32DelayTime毫秒 |
| 22 | } |
| 23 |  |
| 24 | /\*\* |
| 25 | \* @brief 表格 1-1 |
| 26 | \* |
| 27 | \* @param number 序号 |
| 28 | \* @return uint32\_t The actual configured system clock frequency in Hz or zero if the value could not be changed |
| 29 | due to a parameter error or PLL lock failure. |
| 30 | \*/ |
| 31 | uint32\_t exp1\_1\_SysClockSel(uint8\_t number) |
| 32 | { |
| 33 | uint32\_t ui32SysClock; |
| 34 | switch (number) |
| 35 | { |
| 36 | case 1: |
| 37 | ui32SysClock = SysCtlClockFreqSet(SYSCTL\_OSC\_INT | SYSCTL\_USE\_OSC, 16000000); |
| 38 | break; |
| 39 | case 2: |
| 40 | ui32SysClock = SysCtlClockFreqSet(SYSCTL\_OSC\_INT | SYSCTL\_USE\_OSC, 12000000); |
| 41 | break; |
| 42 | case 3: |
| 43 | ui32SysClock = SysCtlClockFreqSet(SYSCTL\_OSC\_INT | SYSCTL\_USE\_OSC, 8000000); |
| 44 | break; |
| 45 | case 4: |
| 46 | ui32SysClock = SysCtlClockFreqSet(SYSCTL\_XTAL\_25MHZ | SYSCTL\_OSC\_MAIN | SYSCTL\_USE\_OSC, 25000000); |
| 47 | break; |
| 48 | case 5: |
| 49 | ui32SysClock = SysCtlClockFreqSet(SYSCTL\_XTAL\_25MHZ | SYSCTL\_OSC\_MAIN | SYSCTL\_USE\_OSC, 12000000); |
| 50 | break; |
| 51 | case 6: |
| 52 | ui32SysClock = SysCtlClockFreqSet(SYSCTL\_XTAL\_25MHZ | SYSCTL\_OSC\_MAIN | SYSCTL\_USE\_OSC, 1000000); |
| 53 | break; |
| 54 | case 7: |
| 55 | ui32SysClock = SysCtlClockFreqSet(SYSCTL\_XTAL\_25MHZ | SYSCTL\_OSC\_MAIN | SYSCTL\_USE\_PLL | SYSCTL\_CFG\_VCO\_480, 25000000); |
| 56 | break; |
| 57 | case 8: |
| 58 | ui32SysClock = SysCtlClockFreqSet(SYSCTL\_XTAL\_25MHZ | SYSCTL\_OSC\_MAIN | SYSCTL\_USE\_PLL | SYSCTL\_CFG\_VCO\_480, 20000000); |
| 59 | break; |
| 60 | case 9: |
| 61 | ui32SysClock = SysCtlClockFreqSet(SYSCTL\_XTAL\_25MHZ | SYSCTL\_OSC\_MAIN | SYSCTL\_USE\_PLL | SYSCTL\_CFG\_VCO\_480, 8000000); |
| 62 | break; |
| 63 | case 10: |
| 64 | ui32SysClock = SysCtlClockFreqSet(SYSCTL\_OSC\_INT | SYSCTL\_USE\_PLL | SYSCTL\_CFG\_VCO\_480, 20000000); |
| 65 | break; |
| 66 | case 11: |
| 67 | ui32SysClock = SysCtlClockFreqSet(SYSCTL\_OSC\_INT | SYSCTL\_USE\_PLL | SYSCTL\_CFG\_VCO\_480, 8000000); |
| 68 | break; |
| 69 | case 12: |
| 70 | ui32SysClock = SysCtlClockFreqSet(SYSCTL\_OSC\_INT | SYSCTL\_USE\_PLL | SYSCTL\_CFG\_VCO\_480, 1000000); |
| 71 | break; |
| 72 | default: |
| 73 | ui32SysClock = SysCtlClockFreqSet(SYSCTL\_OSC\_INT | SYSCTL\_USE\_PLL | SYSCTL\_CFG\_VCO\_480, 8000000); |
| 74 | break; |
| 75 | } |
| 76 | return ui32SysClock; |
| 77 | } |
| 78 |  |

### 实验任务1\_2

|  |  |
| --- | --- |
| 1 | int main(void) |
| 2 | { |
| 3 | volatile uint8\_t ui8KeyValue; |
| 4 | uint32\_t ui32SysClock = SysCtlClockFreqSet(SYSCTL\_XTAL\_25MHZ | SYSCTL\_OSC\_MAIN | SYSCTL\_USE\_PLL | SYSCTL\_CFG\_VCO\_320, 16000000); |
| 5 | GPIOInit(); // GPIO初始化 |
| 6 |  |
| 7 | while (1) // 无限循环 |
| 8 | { |
| 9 | // 读取 PJ0、PJ1 键值. 0-按下 1-松开. Any bit that is not specified by ui8Pins is returned as a 0. |
| 10 | ui8KeyValue = GPIOPinRead(GPIO\_PORTJ\_BASE, GPIO\_PIN\_0 | GPIO\_PIN\_1); |
| 11 |  |
| 12 | // 若 PJ0/1 为 1（松开），则熄灭 PN1/0 （写入0）；否则. |
| 13 | GPIOPinWrite(GPIO\_PORTN\_BASE, GPIO\_PIN\_0 | GPIO\_PIN\_1, ~((ui8KeyValue << 1) | (ui8KeyValue >> 1))); |
| 14 | } |
| 15 | } |
| 16 |  |

## 实验小结

在本实验中，我们的主要工作如下：

（1）建立并熟悉开发环境。我们使用ARM的集成开发环境KEIL uVision5，建立了实验用工程项目；

（2）使用IDE的Debug功能进行在线调试。我们利用添加断点，单步执行，使用watch窗口观察变量的值等调试功能，完成了实验任务；

（3）学习TM4C1294NCPDT微控制器的系统时钟控制原理。我们通过查阅技术文档，结合课程提供的自学材料，实现了编程设置MCU的系统时钟。我们通过实验，探讨了系统时钟频率的取值限制，软件延时的时长与系统时钟（SYSCLK）频率的关系，以及按键操作有时要经过明显迟滞才能得到响应的原因；

（4）学习GPIO的工作原理，结合GPIO的输入与输出进行实验。我们通过GPIO编程，实现了通过按键USR\_SW1-PJ0、USR\_SW2-PJ1控制发光二极管D1-PN1、D2-PN0。

（5）使用示波器观测PF0信号频率，与希望设定的系统时钟频率和DEBUG观测到的时钟频率数值作比较。

# 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

A2000TM4扩展板采用的机械按键，具有“按键抖动”的固有缺陷，因此需要做“消抖”处理。在实验例程中，按键状态检测被安排在Systick中断服务程序中进行，而该中断每20ms进入一次，事实上已实现“定时扫描法”消抖，详见自学材料[7]。

### 实验任务2\_1

#### 任务要求

将exp2\_0.c另存为exp2\_1.c。以exp2\_0.c的功能为基础，修改程序，完成实验任务2\_1。

在实验任务2\_1中，要让电路呈现三种工作模式（见表2.1）。通过人工按动一次任意键，使电路做一次模式切换，在三种工作模式中轮转。开机或复位运行，以模式0为默认工作模式。

**表2.1 工作模式**

|  |  |
| --- | --- |
| **模式** | **功能描述** |
| 0 | 原功能1和功能3有效；功能2暂停（LED灯静止不闪动） |
| 1 | 原功能1暂停（计时）；原功能3有效；  8个指示灯以走马灯方式，每0.5秒向右（循环）移动一格 |
| 2 | 原功能1暂停（计时）；原功能3有效；  8个指示灯以走马灯方式，每0.5秒向左（循环）移动一格 |

一次人工按键操作应包括按下和放开两个过程。任意按下一键，会触发一次模式转变，但按住不放不能连续改变工作模式，放开动作也不直接影响工作模式；直至放开之后的下一次按键，才能再次触发转变。

#### 实现思路

建立一个全局变量mode，初始值为0，用来记录当前工作模式（相应于表2.1）。建立一个静态局部变量，用来暂存上次读到的键值。

在主程序的循环体中，将本次读到的键值与上次读到的键值作比较：若上次读到0（表示无键按下，参见技术文档[6]对全局变量key\_code的说明），且本次读到非0值，表示检测到一次按下过程；若上次读到非0值，且本次读到0值，表示检测到一次放开过程。

实验要求仅按下过程会直接改变工作模式，因此每检测到一次按下过程，就令变量mode自增1再模3，其余情况不改变mode的值。在主程序和Systick的ISR中，根据mode的取值执行相应的控制代码。

#### 实验结果和分析

实验任务已正确实现。

可将上述实验思路理解为一个简单的有限状态机（FSM）。共有三个状态，状态转移条件是一次按下事件。

### 实验任务2\_2

#### 任务要求

将exp2\_0.c另存命名为exp2\_2.c。以exp2\_0.c的功能为基础，修改程序，完成实验任务2\_2。

exp2\_2.c的功能要点如下。

1）原功能1继续有效，且受功能3影响。

2）取消原功能2；指定最左端LED灯作为一个状态标志，当功能1正常计数时最左端LED灯亮，功能1暂停计时则最左端LED灯灭；其余LED灯保持为全灭。

3）原功能3改为：利用左数第二位数码管显示“键值”（按键编号）；当没有按键按下时，显示数码“0”；当人工按下某键，数码管显示该键的编号（1到9）。当按一次1号键，可循环切换功能1的计时工作状态，启动或暂停（start / pause）计时（计数）功能；当按一次2号键，对功能1的计时值清零；当按下其余键时，对功能1没有影响。

#### 实现思路

建立一个静态全局变量cur\_state，用于记录当前状态号；变量的存储方式采取静态方式，是为了暂存上一状态。建立全局变量pre\_state，用于记录上一状态号。

采用有限状态机（FSM）实现实验任务。FSM规则如下：

（a）状态定义。0-计时进行；1-计时暂停；2-计时清零。

（b）状态转移条件。(C01, C10)[[1]](#footnote-1) 检测到1号键按下；(C02, C12) 检测到2号键按下；(C20, C21)完成一次状态动作。

（c）状态转移动作。(C02, C12) 保存当前状态号（通过更新全局变量pre\_state），计数值清零；(C01, C10) 保存当前状态号。

（d）状态动作。0-计时进行，最左端LED灯亮，其余全灭；1-计时暂停，LED灯全灭；2-更新数码管显示（通过修改全局变量digit[8]，参见技术文档[6]），然后返回上一状态号 (pre\_state)。

状态转移、状态动作都在主程序的循环体中实现，以精简Systick的ISR，保证其在一个中断周期（已设定为20ms）内执行完毕。

#### 实验结果和分析

实验任务已正确实现。

使用FSM的思想，可使程序逻辑清晰，易于维护。

### 实验任务2\_3

#### 任务要求

在实验任务2\_2基础上，修改程序exp2\_2.c代码并另存为exp2\_3.c，完成实验任务2\_3。

将功能1计时的加计数改为减计数，且计时初值为9999（注：对应999.9秒）；每计满1000.0秒，重新开始。

#### 实现思路

在实验任务2\_2的基础上做相应修改即可。

最好避免uint32\_t类型的计数器变量test\_counter发生下溢。

#### 实验结果和分析

实验任务已正确实现。

### 实验任务2\_4

#### 任务要求

在实验任务2\_3基础上，修改程序exp2\_3.c代码并另存命名为exp2\_4.c，完成实验任务2\_4。

添加使用3号按键作为加减计数模式切换控制。开机或复位后，为减计时；之后每按一次3号键，切换一次加减计数模式。

取消用左数第二位数码管显示“键值” 的功能。改以左侧4位数码管显示计时模式状态，即用“\_UU\_”表示加计数，显示“\_AA\_”表示减计数。

#### 实现思路

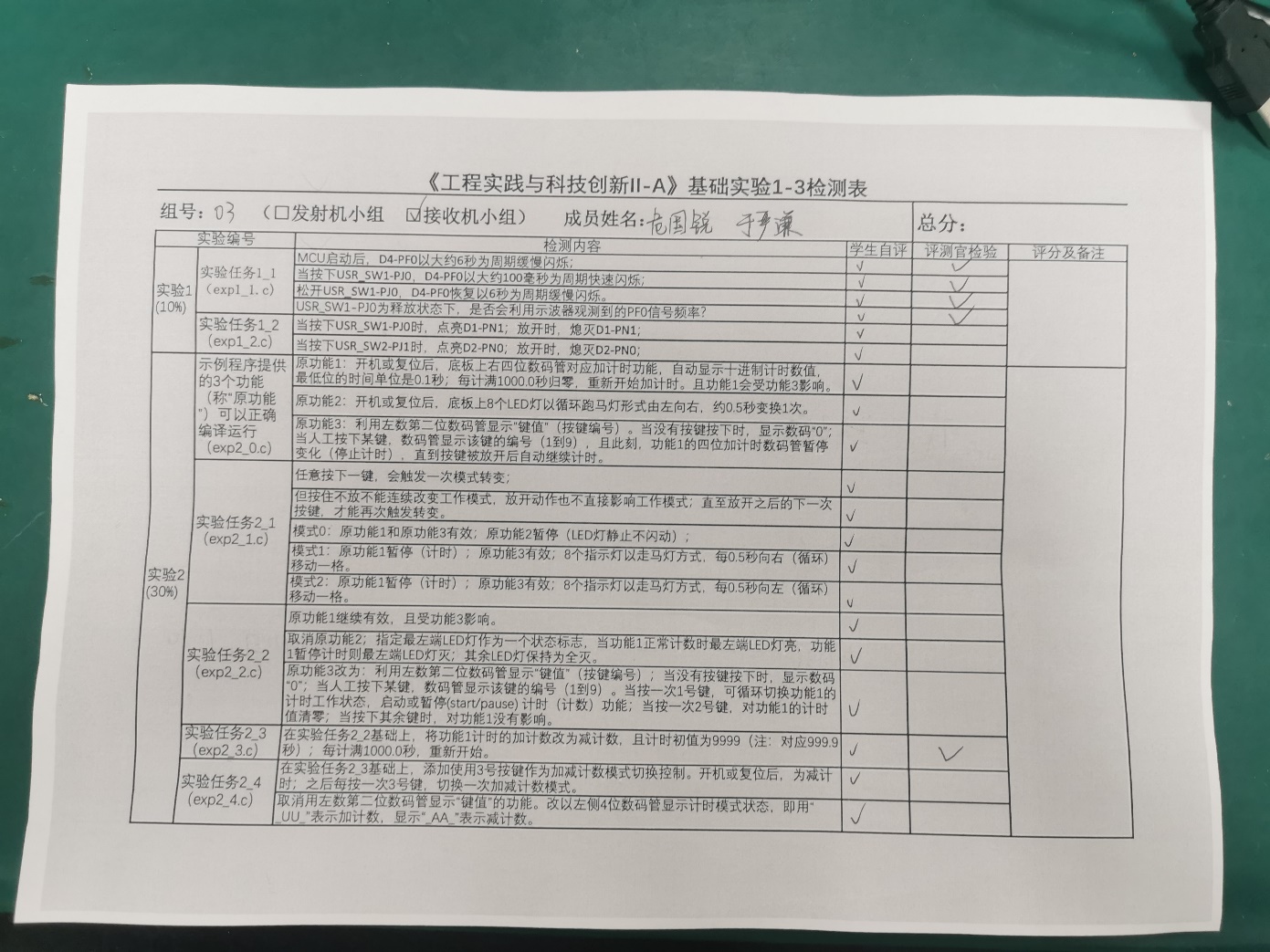
增加全局变量flag\_DecCnt表示计数方向。

按调用关系，逐级上查TM1638芯片相关API函数TM1638\_RefreshDIGIandLED、 TM1638\_DigiSegment的源代码和注释，可知相关API函数的用法，得知可向全局变量digit[8]里存'\_'，'A'，'U' 等字符。

#### 实验结果和分析

实验任务已正确实现。

## 实验结果自测记录



## 实验核心代码清单

### 实验任务2\_1

|  |  |
| --- | --- |
| 1 | int main(void) |
| 2 | { |
| 3 | uint8\_t temp, i; |
| 4 |  |
| 5 | DevicesInit(); //  MCU器件初始化 |
| 6 |  |
| 7 | while (clock100ms < 3) |
| 8 | ;          // 延时>60ms,等待TM1638上电完成 |
| 9 | TM1638\_Init(); // 初始化TM1638 |
| 10 |  |
| 11 | while (1) |
| 12 | { |
| 13 | UpdateMode(); |
| 14 |  |
| 15 | if (clock100ms\_flag == 1) // 检查0.1秒定时是否到 |
| 16 | { |
| 17 | clock100ms\_flag = 0; |
| 18 | // [仅模式 0 时有效]每0.1秒累加计时值在数码管上以十进制显示，有键按下时暂停计时 |
| 19 | if (!mode && !key\_code) |
| 20 | { |
| 21 | if (++test\_counter >= 10000) |
| 22 | test\_counter = 0; |
| 23 | digit[0] = test\_counter / 1000;     // 计算百位数 |
| 24 | digit[1] = test\_counter / 100 % 10; // 计算十位数 |
| 25 | digit[2] = test\_counter / 10 % 10;  // 计算个位数 |
| 26 | digit[3] = test\_counter % 10;       // 计算百分位数 |
| 27 | } |
| 28 | } |
| 29 |  |
| 30 | if (clock500ms\_flag == 1) // 检查0.5秒定时是否到 |
| 31 | { |
| 32 | clock500ms\_flag = 0; |
| 33 | // [仅模式 1 有效] 8个指示灯以走马灯方式，每0.5秒向右（循环）移动一格 |
| 34 | if (mode == 1) |
| 35 | { |
| 36 | temp = led[0]; |
| 37 | for (i = 0; i < 7; i++) |
| 38 | led[i] = led[i + 1]; |
| 39 | led[7] = temp; |
| 40 | } |
| 41 |  |
| 42 | // [仅模式 2 有效] 8个指示灯以走马灯方式，每0.5秒向左（循环）移动一格 |
| 43 | if (mode == 2) |
| 44 | { |
| 45 | temp = led[7]; |
| 46 | for (i = 0; i < 7; i++) |
| 47 | led[7 - i] = led[6 - i]; |
| 48 | led[0] = temp; |
| 49 | } |
| 50 | } |
| 51 | } |
| 52 | } |
| 53 |  |
| 54 | //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* |
| 55 | // |
| 56 | // 函数原型：void SysTick\_Handler(void) |
| 57 | // 函数功能：SysTick中断服务程序 |
| 58 | // 函数参数：无 |
| 59 | // 函数返回值：无 |
| 60 | // |
| 61 | //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* |
| 62 | void SysTick\_Handler(void) // 定时周期为20ms |
| 63 | { |
| 64 | // 0.1秒钟软定时器计数 |
| 65 | if (++clock100ms >= V\_T100ms) |
| 66 | { |
| 67 | clock100ms\_flag = 1; // 当0.1秒到时，溢出标志置1 |
| 68 | clock100ms = 0; |
| 69 | } |
| 70 |  |
| 71 | // 0.5秒钟软定时器计数 |
| 72 | if (++clock500ms >= V\_T500ms) |
| 73 | { |
| 74 | clock500ms\_flag = 1; // 当0.5秒到时，溢出标志置1 |
| 75 | clock500ms = 0; |
| 76 | } |
| 77 |  |
| 78 | // 刷新全部数码管和LED指示灯 |
| 79 | TM1638\_RefreshDIGIandLED(digit, pnt, led); |
| 80 |  |
| 81 | // 检查当前键盘输入(因在 20ms 中断中，故已消抖)，0代表无键操作，1-9表示有对应按键 |
| 82 | // 键号显示在一位数码管上 |
| 83 | key\_code = TM1638\_Readkeyboard(); |
| 84 |  |
| 85 | digit[5] = key\_code; |
| 86 | } |
| 87 |  |
| 88 | /\*\* |
| 89 | \* @brief 更新当前模式，会修改全局变量 mode. |
| 90 | \* |
| 91 | \* 任意按下一键，会触发一次模式转变，按 0->1->2->0 轮转. |
| 92 | \* 但按住不放不能连续改变工作模式，放开动作也不直接影响工作模式； |
| 93 | \* 直至放开之后的下一次按键，才能再次触发转变. |
| 94 | \* |
| 95 | \*/ |
| 96 | void UpdateMode() |
| 97 | { |
| 98 | static uint8\_t cur\_key\_code;         // 当前按键状态 |
| 99 | uint8\_t pre\_key\_code = cur\_key\_code; // 上一按键状态 |
| 100 | cur\_key\_code = key\_code; |
| 101 | if (!pre\_key\_code && cur\_key\_code) |
| 102 | { |
| 103 | mode = (mode + 1) % 3; |
| 104 | } |
| 105 | } |
| 106 |  |

### 实验任务2\_2

|  |  |
| --- | --- |
| 1 | int main(void) |
| 2 | { |
| 3 | uint8\_t i; |
| 4 |  |
| 5 | DevicesInit(); //  MCU器件初始化 |
| 6 |  |
| 7 | while (clock100ms < 3) |
| 8 | ;          // 延时>60ms,等待TM1638上电完成 |
| 9 | TM1638\_Init(); // 初始化TM1638 |
| 10 |  |
| 11 | while (1) |
| 12 | { |
| 13 | static uint8\_t cur\_key\_code; |
| 14 | uint8\_t pre\_key\_code = cur\_key\_code; |
| 15 | cur\_key\_code = key\_code; |
| 16 |  |
| 17 | /\*\* |
| 18 | \* FSM. |
| 19 | \* |
| 20 | \* 当前状态号. 0-计时进行；1-计时暂停；2-计时清零. |
| 21 | \* 状态转移规则：(C01, C10) 当状态0/1时，按一次1号键，转移至状态1/0；(C02, C12) 当状态0/1时，按一次2号键，转移至状态2；(C2\*) 当状态2时，完成一次状态动作后，返回至上一状态. |
| 22 | \* 状态转移动作：(C02, C12) 保存当前状态号（通过更新 pre\_state），将计数值清零；(C01, C10) 保存当前状态号（通过更新 pre\_state）. |
| 23 | \* 状态动作：0-计时进行，最左端LED灯亮，其余全灭；1-计时暂停，LED灯全灭；2-立即刷新数码管的值，然后返回上一状态号(pre\_state). |
| 24 | \*/ |
| 25 | switch (cur\_state) |
| 26 | { |
| 27 | case 0: |
| 28 | // C01: 当状态0/1时，按一次1号键，转移至状态1/0；保存当前状态号（通过更新 pre\_state） |
| 29 | if (pre\_key\_code != 1 && cur\_key\_code == 1) |
| 30 | { |
| 31 | pre\_state = 0; |
| 32 | cur\_state = 1; |
| 33 | } |
| 34 |  |
| 35 | // C02: 保存当前状态号（通过更新 pre\_state），将计数值清零 |
| 36 | if (pre\_key\_code != 2 && cur\_key\_code == 2) |
| 37 | { |
| 38 | test\_counter = 9999; |
| 39 | pre\_state = 0; |
| 40 | cur\_state = 2; |
| 41 | } |
| 42 |  |
| 43 | // 状态动作0-计时进行，最左端LED灯亮，其余全灭 |
| 44 | led[7] = 1; |
| 45 | for (i = 0; i < 7; i++) |
| 46 | { |
| 47 | \*(led + i) = 0; |
| 48 | } |
| 49 | break; |
| 50 | case 1: |
| 51 | // C10: 当状态0/1时，按一次1号键，转移至状态1/0；保存当前状态号（通过更新 pre\_state） |
| 52 | if (pre\_key\_code != 1 && cur\_key\_code == 1) |
| 53 | { |
| 54 | pre\_state = 1; |
| 55 | cur\_state = 0; |
| 56 | } |
| 57 |  |
| 58 | // C12: 保存当前状态号（通过更新 pre\_state），将计数值清零 |
| 59 | if (pre\_key\_code != 2 && cur\_key\_code == 2) |
| 60 | { |
| 61 | test\_counter = 9999; |
| 62 | pre\_state = 1; |
| 63 | cur\_state = 2; |
| 64 | } |
| 65 |  |
| 66 | // 状态动作1-计时暂停，LED灯全灭 |
| 67 | for (i = 0; i < 8; i++) |
| 68 | { |
| 69 | \*(led + i) = 0; |
| 70 | } |
| 71 | break; |
| 72 | case 2: |
| 73 | // 状态动作2-立即刷新数码管的值，然后返回上一状态号(pre\_state). |
| 74 | digit[0] = digit[1] = digit[2] = digit[3] = 0; |
| 75 | cur\_state = pre\_state; |
| 76 | break; |
| 77 | default: |
| 78 | break; |
| 79 | } |
| 80 |  |
| 81 | if (clock100ms\_flag == 1) // 检查0.1秒定时是否到 |
| 82 | { |
| 83 | clock100ms\_flag = 0; |
| 84 | // [仅状态0有效] 每0.1秒累加计时值在数码管上以十进制显示 |
| 85 | if (cur\_state == 0) |
| 86 | { |
| 87 | if (++test\_counter >= 10000) |
| 88 | test\_counter = 0; |
| 89 | digit[0] = test\_counter / 1000;     // 计算百位数 |
| 90 | digit[1] = test\_counter / 100 % 10; // 计算十位数 |
| 91 | digit[2] = test\_counter / 10 % 10;  // 计算个位数 |
| 92 | digit[3] = test\_counter % 10;       // 计算百分位数 |
| 93 | } |
| 94 | } |
| 95 |  |
| 96 | if (clock500ms\_flag == 1) // 检查0.5秒定时是否到 |
| 97 | { |
| 98 | clock500ms\_flag = 0; |
| 99 | } |
| 100 | } |
| 101 | } |
| 102 |  |
| 103 | //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* |
| 104 | // |
| 105 | // 函数原型：void SysTick\_Handler(void) |
| 106 | // 函数功能：SysTick中断服务程序 |
| 107 | // 函数参数：无 |
| 108 | // 函数返回值：无 |
| 109 | // |
| 110 | //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* |
| 111 | void SysTick\_Handler(void) // 定时周期为20ms |
| 112 | { |
| 113 | // 0.1秒钟软定时器计数 |
| 114 | if (++clock100ms >= V\_T100ms) |
| 115 | { |
| 116 | clock100ms\_flag = 1; // 当0.1秒到时，溢出标志置1 |
| 117 | clock100ms = 0; |
| 118 | } |
| 119 |  |
| 120 | // 0.5秒钟软定时器计数 |
| 121 | if (++clock500ms >= V\_T500ms) |
| 122 | { |
| 123 | clock500ms\_flag = 1; // 当0.5秒到时，溢出标志置1 |
| 124 | clock500ms = 0; |
| 125 | } |
| 126 |  |
| 127 | // 刷新全部数码管和LED指示灯 |
| 128 | TM1638\_RefreshDIGIandLED(digit, pnt, led); |
| 129 |  |
| 130 | // 检查当前键盘输入，0代表无键操作，1-9表示有对应按键 |
| 131 | // 键号显示在一位数码管上 |
| 132 | key\_code = TM1638\_Readkeyboard(); |
| 133 |  |
| 134 | digit[5] = key\_code; |
| 135 | } |
| 136 |  |

### 实验任务2\_3

|  |  |
| --- | --- |
| 1 | #include <stdint.h> |
| 2 | #include <stdbool.h> |
| 3 | #include "inc/hw\_memmap.h"       // 基址宏定义 |
| 4 | #include "inc/hw\_types.h"        // 数据类型宏定义，寄存器访问函数 |
| 5 | #include "driverlib/debug.h"     // 调试用 |
| 6 | #include "driverlib/gpio.h"      // 通用IO口宏定义 |
| 7 | #include "driverlib/pin\_map.h"   // TM4C系列MCU外围设备管脚宏定义 |
| 8 | #include "driverlib/sysctl.h"    // 系统控制定义 |
| 9 | #include "driverlib/systick.h"   // SysTick Driver 原型 |
| 10 | #include "driverlib/interrupt.h" // NVIC Interrupt Controller Driver 原型 |
| 11 |  |
| 12 | #include "tm1638.h" // 与控制TM1638芯片有关的函数 |
| 13 |  |
| 14 | //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* |
| 15 | // |
| 16 | // 宏定义 |
| 17 | // |
| 18 | //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* |
| 19 | #define SYSTICK\_FREQUENCY 50 // SysTick频率为50Hz，即循环定时周期20ms |
| 20 |  |
| 21 | #define V\_T100ms 5  // 0.1s软件定时器溢出值，5个20ms |
| 22 | #define V\_T500ms 25 // 0.5s软件定时器溢出值，25个20ms |
| 23 |  |
| 24 | //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* |
| 25 | // |
| 26 | // 函数原型声明 |
| 27 | // |
| 28 | //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* |
| 29 | void GPIOInit(void);    // GPIO初始化 |
| 30 | void SysTickInit(void); // 设置SysTick中断 |
| 31 | void DevicesInit(void); // MCU器件初始化，注：会调用上述函数 |
| 32 | //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* |
| 33 | // |
| 34 | // 变量定义 |
| 35 | // |
| 36 | //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* |
| 37 |  |
| 38 | // 软件定时器计数 |
| 39 | uint8\_t clock100ms = 0; |
| 40 | uint8\_t clock500ms = 0; |
| 41 |  |
| 42 | // 软件定时器溢出标志 |
| 43 | uint8\_t clock100ms\_flag = 0; |
| 44 | uint8\_t clock500ms\_flag = 0; |
| 45 |  |
| 46 | // 测试用计数器 |
| 47 | uint32\_t test\_counter = 0; |
| 48 |  |
| 49 | // 8位数码管显示的数字或字母符号 |
| 50 | // 注：板上数码位从左到右序号排列为4、5、6、7、0、1、2、3 |
| 51 | uint8\_t digit[8] = {' ', ' ', ' ', ' ', '\_', ' ', '\_', ' '}; |
| 52 |  |
| 53 | // 8位小数点 1亮  0灭 |
| 54 | // 注：板上数码位小数点从左到右序号排列为4、5、6、7、0、1、2、3 |
| 55 | uint8\_t pnt = 0x04; |
| 56 |  |
| 57 | // 8个LED指示灯状态，0灭，1亮 |
| 58 | // 注：板上指示灯从左到右序号排列为7、6、5、4、3、2、1、0 |
| 59 | //     对应元件LED8、LED7、LED6、LED5、LED4、LED3、LED2、LED1 |
| 60 | uint8\_t led[] = {1, 1, 1, 1, 1, 1, 1, 0}; |
| 61 |  |
| 62 | // 当前按键值 |
| 63 | uint8\_t key\_code = 0; |
| 64 |  |
| 65 | // 系统时钟频率 |
| 66 | uint32\_t ui32SysClock; |
| 67 |  |
| 68 | // 当前状态号. 0-计时进行；1-计时暂停；2-计时清零. |
| 69 | // 状态转移规则：(C01, C10) 当状态0/1时，按一次1号键，转移至状态1/0；(C02, C12) 当状态0/1时，按一次2号键，转移至状态2；(C2\*) 当状态2时，完成一次状态动作后，返回至上一状态. |
| 70 | // 状态转移动作：(C02, C12) 保存当前状态号（通过更新 pre\_state），将计数值清零；(C01, C10) 保存当前状态号（通过更新 pre\_state）. |
| 71 | // 状态动作：0-计时进行，最左端LED灯亮，其余全灭；1-计时暂停，LED灯全灭；2-立即刷新数码管的值，然后返回上一状态号(pre\_state). |
| 72 | uint8\_t cur\_state = 0; |
| 73 | uint8\_t pre\_state = 0; // 上一状态号. |
| 74 |  |
| 75 | //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* |
| 76 | // |
| 77 | // 主程序 |
| 78 | // |
| 79 | //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* |
| 80 | int main(void) |
| 81 | { |
| 82 | uint8\_t i; |
| 83 |  |
| 84 | DevicesInit(); //  MCU器件初始化 |
| 85 |  |
| 86 | while (clock100ms < 3) |
| 87 | ;          // 延时>60ms,等待TM1638上电完成 |
| 88 | TM1638\_Init(); // 初始化TM1638 |
| 89 |  |
| 90 | while (1) |
| 91 | { |
| 92 | static uint8\_t cur\_key\_code; |
| 93 | uint8\_t pre\_key\_code = cur\_key\_code; |
| 94 | cur\_key\_code = key\_code; |
| 95 |  |
| 96 | /\*\* |
| 97 | \* FSM. |
| 98 | \* |
| 99 | \* 当前状态号. 0-计时进行；1-计时暂停；2-计时清零. |
| 100 | \* 状态转移规则：(C01, C10) 当状态0/1时，按一次1号键，转移至状态1/0；(C02, C12) 当状态0/1时，按一次2号键，转移至状态2；(C2\*) 当状态2时，完成一次状态动作后，返回至上一状态. |
| 101 | \* 状态转移动作：(C02, C12) 保存当前状态号（通过更新 pre\_state），将计数值清零；(C01, C10) 保存当前状态号（通过更新 pre\_state）. |
| 102 | \* 状态动作：0-计时进行，最左端LED灯亮，其余全灭；1-计时暂停，LED灯全灭；2-立即刷新数码管的值，然后返回上一状态号(pre\_state). |
| 103 | \*/ |
| 104 | switch (cur\_state) |
| 105 | { |
| 106 | case 0: |
| 107 | // C01: 当状态0/1时，按一次1号键，转移至状态1/0；保存当前状态号（通过更新 pre\_state） |
| 108 | if (pre\_key\_code != 1 && cur\_key\_code == 1) |
| 109 | { |
| 110 | pre\_state = 0; |
| 111 | cur\_state = 1; |
| 112 | } |
| 113 |  |
| 114 | // C02: 保存当前状态号（通过更新 pre\_state），将计数值清零 |
| 115 | if (pre\_key\_code != 2 && cur\_key\_code == 2) |
| 116 | { |
| 117 | test\_counter = 9999; |
| 118 | pre\_state = 0; |
| 119 | cur\_state = 2; |
| 120 | } |
| 121 |  |
| 122 | // 状态动作0-计时进行，最左端LED灯亮，其余全灭 |
| 123 | led[7] = 1; |
| 124 | for (i = 0; i < 7; i++) |
| 125 | { |
| 126 | \*(led + i) = 0; |
| 127 | } |
| 128 | break; |
| 129 | case 1: |
| 130 | // C10: 当状态0/1时，按一次1号键，转移至状态1/0；保存当前状态号（通过更新 pre\_state） |
| 131 | if (pre\_key\_code != 1 && cur\_key\_code == 1) |
| 132 | { |
| 133 | pre\_state = 1; |
| 134 | cur\_state = 0; |
| 135 | } |
| 136 |  |
| 137 | // C12: 保存当前状态号（通过更新 pre\_state），将计数值清零 |
| 138 | if (pre\_key\_code != 2 && cur\_key\_code == 2) |
| 139 | { |
| 140 | test\_counter = 0; |
| 141 | pre\_state = 1; |
| 142 | cur\_state = 2; |
| 143 | } |
| 144 |  |
| 145 | // 状态动作1-计时暂停，LED灯全灭 |
| 146 | for (i = 0; i < 8; i++) |
| 147 | { |
| 148 | \*(led + i) = 0; |
| 149 | } |
| 150 | break; |
| 151 | case 2: |
| 152 | // 状态动作2-立即刷新数码管的值，然后返回上一状态号(pre\_state). |
| 153 | digit[0] = digit[1] = digit[2] = digit[3] = 9; |
| 154 | cur\_state = pre\_state; |
| 155 | break; |
| 156 | default: |
| 157 | break; |
| 158 | } |
| 159 |  |
| 160 | if (clock100ms\_flag == 1) // 检查0.1秒定时是否到 |
| 161 | { |
| 162 | clock100ms\_flag = 0; |
| 163 | // [仅状态0有效] 每0.1秒累加计时值在数码管上以十进制显示 |
| 164 | if (cur\_state == 0) |
| 165 | { |
| 166 | // 要避免uint类型的变量发生下溢，因可能引发异常. |
| 167 | if (!test\_counter) |
| 168 | { |
| 169 | test\_counter = 10000; |
| 170 | } |
| 171 | --test\_counter; |
| 172 | digit[0] = test\_counter / 1000;     // 计算百位数 |
| 173 | digit[1] = test\_counter / 100 % 10; // 计算十位数 |
| 174 | digit[2] = test\_counter / 10 % 10;  // 计算个位数 |
| 175 | digit[3] = test\_counter % 10;       // 计算百分位数 |
| 176 | } |
| 177 | } |
| 178 |  |
| 179 | if (clock500ms\_flag == 1) // 检查0.5秒定时是否到 |
| 180 | { |
| 181 | clock500ms\_flag = 0; |
| 182 | } |
| 183 | } |
| 184 | } |
| 185 |  |
| 186 | //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* |
| 187 | // |
| 188 | // 函数原型：void SysTick\_Handler(void) |
| 189 | // 函数功能：SysTick中断服务程序 |
| 190 | // 函数参数：无 |
| 191 | // 函数返回值：无 |
| 192 | // |
| 193 | //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* |
| 194 | void SysTick\_Handler(void) // 定时周期为20ms |
| 195 | { |
| 196 | // 0.1秒钟软定时器计数 |
| 197 | if (++clock100ms >= V\_T100ms) |
| 198 | { |
| 199 | clock100ms\_flag = 1; // 当0.1秒到时，溢出标志置1 |
| 200 | clock100ms = 0; |
| 201 | } |
| 202 |  |
| 203 | // 0.5秒钟软定时器计数 |
| 204 | if (++clock500ms >= V\_T500ms) |
| 205 | { |
| 206 | clock500ms\_flag = 1; // 当0.5秒到时，溢出标志置1 |
| 207 | clock500ms = 0; |
| 208 | } |
| 209 |  |
| 210 | // 刷新全部数码管和LED指示灯 |
| 211 | TM1638\_RefreshDIGIandLED(digit, pnt, led); |
| 212 |  |
| 213 | // 检查当前键盘输入，0代表无键操作，1-9表示有对应按键 |
| 214 | // 键号显示在一位数码管上 |
| 215 | key\_code = TM1638\_Readkeyboard(); |
| 216 |  |
| 217 | digit[5] = key\_code; |
| 218 | } |
| 219 |  |

### 实验任务2\_4

|  |  |
| --- | --- |
| 1 | #include <stdint.h> |
| 2 | #include <stdbool.h> |
| 3 | #include "inc/hw\_memmap.h"       // 基址宏定义 |
| 4 | #include "inc/hw\_types.h"        // 数据类型宏定义，寄存器访问函数 |
| 5 | #include "driverlib/debug.h"     // 调试用 |
| 6 | #include "driverlib/gpio.h"      // 通用IO口宏定义 |
| 7 | #include "driverlib/pin\_map.h"   // TM4C系列MCU外围设备管脚宏定义 |
| 8 | #include "driverlib/sysctl.h"    // 系统控制定义 |
| 9 | #include "driverlib/systick.h"   // SysTick Driver 原型 |
| 10 | #include "driverlib/interrupt.h" // NVIC Interrupt Controller Driver 原型 |
| 11 |  |
| 12 | #include "tm1638.h" // 与控制TM1638芯片有关的函数 |
| 13 |  |
| 14 | //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* |
| 15 | // |
| 16 | // 宏定义 |
| 17 | // |
| 18 | //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* |
| 19 | #define SYSTICK\_FREQUENCY 50 // SysTick频率为50Hz，即循环定时周期20ms |
| 20 |  |
| 21 | #define V\_T100ms 5  // 0.1s软件定时器溢出值，5个20ms |
| 22 | #define V\_T500ms 25 // 0.5s软件定时器溢出值，25个20ms |
| 23 |  |
| 24 | //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* |
| 25 | // |
| 26 | // 函数原型声明 |
| 27 | // |
| 28 | //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* |
| 29 | void GPIOInit(void);    // GPIO初始化 |
| 30 | void SysTickInit(void); // 设置SysTick中断 |
| 31 | void DevicesInit(void); // MCU器件初始化，注：会调用上述函数 |
| 32 | //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* |
| 33 | // |
| 34 | // 变量定义 |
| 35 | // |
| 36 | //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* |
| 37 |  |
| 38 | // 软件定时器计数 |
| 39 | uint8\_t clock100ms = 0; |
| 40 | uint8\_t clock500ms = 0; |
| 41 |  |
| 42 | // 软件定时器溢出标志 |
| 43 | uint8\_t clock100ms\_flag = 0; |
| 44 | uint8\_t clock500ms\_flag = 0; |
| 45 |  |
| 46 | // 测试用计数器 |
| 47 | uint32\_t test\_counter = 0; |
| 48 |  |
| 49 | // 8位数码管显示的数字或字母符号 |
| 50 | // 注：板上数码位从左到右序号排列为4、5、6、7、0、1、2、3 |
| 51 | uint8\_t digit[8] = {' ', ' ', ' ', ' ', '\_', ' ', '\_', ' '}; |
| 52 |  |
| 53 | // 8位小数点 1亮  0灭 |
| 54 | // 注：板上数码位小数点从左到右序号排列为4、5、6、7、0、1、2、3 |
| 55 | uint8\_t pnt = 0x04; |
| 56 |  |
| 57 | // 8个LED指示灯状态，0灭，1亮 |
| 58 | // 注：板上指示灯从左到右序号排列为7、6、5、4、3、2、1、0 |
| 59 | //     对应元件LED8、LED7、LED6、LED5、LED4、LED3、LED2、LED1 |
| 60 | uint8\_t led[] = {1, 1, 1, 1, 1, 1, 1, 0}; |
| 61 |  |
| 62 | // 当前按键值 |
| 63 | uint8\_t key\_code = 0; |
| 64 |  |
| 65 | // 系统时钟频率 |
| 66 | uint32\_t ui32SysClock; |
| 67 |  |
| 68 | // 当前状态号. 0-计时进行；1-计时暂停；2-计时清零. |
| 69 | // 状态转移规则：(C01, C10) 当状态0/1时，按一次1号键，转移至状态1/0；(C02, C12) 当状态0/1时，按一次2号键，转移至状态2；(C2\*) 当状态2时，完成一次状态动作后，返回至上一状态. |
| 70 | // 状态转移动作：(C02, C12) 保存当前状态号（通过更新 pre\_state），将计数值清零；(C01, C10) 保存当前状态号（通过更新 pre\_state）. |
| 71 | // 状态动作：0-计时进行，最左端LED灯亮，其余全灭；1-计时暂停，LED灯全灭；2-立即刷新数码管的值，然后返回上一状态号(pre\_state). |
| 72 | uint8\_t cur\_state = 0; |
| 73 | uint8\_t pre\_state = 0; // 上一状态号. |
| 74 | bool flag\_DecCnt = 1;  // 计数方向标志. 1-减计数，0-加计数 |
| 75 |  |
| 76 | //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* |
| 77 | // |
| 78 | // 主程序 |
| 79 | // |
| 80 | //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* |
| 81 | int main(void) |
| 82 | { |
| 83 | uint8\_t i; |
| 84 |  |
| 85 | DevicesInit(); //  MCU器件初始化 |
| 86 |  |
| 87 | while (clock100ms < 3) |
| 88 | ;          // 延时>60ms,等待TM1638上电完成 |
| 89 | TM1638\_Init(); // 初始化TM1638 |
| 90 |  |
| 91 | while (1) |
| 92 | { |
| 93 | static uint8\_t cur\_key\_code; |
| 94 | uint8\_t pre\_key\_code = cur\_key\_code; |
| 95 | cur\_key\_code = key\_code; |
| 96 |  |
| 97 | // 更新计数方向标志 flag\_DecCnt |
| 98 | if (pre\_key\_code != 3 && cur\_key\_code == 3) |
| 99 | { |
| 100 | flag\_DecCnt = !flag\_DecCnt; |
| 101 | } |
| 102 |  |
| 103 | // 更新左侧4位数码管(digit[4~7])的显示 |
| 104 | digit[4] = digit[7] = '\_'; |
| 105 | digit[5] = digit[6] = (flag\_DecCnt ? 'A' : 'U'); |
| 106 |  |
| 107 | /\*\* |
| 108 | \* FSM. |
| 109 | \* |
| 110 | \* 当前状态号. 0-计时进行；1-计时暂停；2-计时清零. |
| 111 | \* 状态转移规则：(C01, C10) 当状态0/1时，按一次1号键，转移至状态1/0；(C02, C12) 当状态0/1时，按一次2号键，转移至状态2；(C2\*) 当状态2时，完成一次状态动作后，返回至上一状态. |
| 112 | \* 状态转移动作：(C02, C12) 保存当前状态号（通过更新 pre\_state），将计数值清零；(C01, C10) 保存当前状态号（通过更新 pre\_state）. |
| 113 | \* 状态动作：0-计时进行，最左端LED灯亮，其余全灭；1-计时暂停，LED灯全灭；2-立即刷新数码管的值，然后返回上一状态号(pre\_state). |
| 114 | \*/ |
| 115 | switch (cur\_state) |
| 116 | { |
| 117 | case 0: |
| 118 | // C01: 当状态0/1时，按一次1号键，转移至状态1/0；保存当前状态号（通过更新 pre\_state） |
| 119 | if (pre\_key\_code != 1 && cur\_key\_code == 1) |
| 120 | { |
| 121 | pre\_state = 0; |
| 122 | cur\_state = 1; |
| 123 | } |
| 124 |  |
| 125 | // C02: 保存当前状态号（通过更新 pre\_state），将计数值清零 |
| 126 | if (pre\_key\_code != 2 && cur\_key\_code == 2) |
| 127 | { |
| 128 | test\_counter = (flag\_DecCnt ? 0 : 9999); |
| 129 | pre\_state = 0; |
| 130 | cur\_state = 2; |
| 131 | } |
| 132 |  |
| 133 | // 状态动作0-计时进行，最左端LED灯亮，其余全灭 |
| 134 | led[7] = 1; |
| 135 | for (i = 0; i < 7; i++) |
| 136 | { |
| 137 | \*(led + i) = 0; |
| 138 | } |
| 139 | break; |
| 140 | case 1: |
| 141 | // C10: 当状态0/1时，按一次1号键，转移至状态1/0；保存当前状态号（通过更新 pre\_state） |
| 142 | if (pre\_key\_code != 1 && cur\_key\_code == 1) |
| 143 | { |
| 144 | pre\_state = 1; |
| 145 | cur\_state = 0; |
| 146 | } |
| 147 |  |
| 148 | // C12: 保存当前状态号（通过更新 pre\_state），将计数值清零 |
| 149 | if (pre\_key\_code != 2 && cur\_key\_code == 2) |
| 150 | { |
| 151 | test\_counter = (flag\_DecCnt ? 0 : 9999); |
| 152 | pre\_state = 1; |
| 153 | cur\_state = 2; |
| 154 | } |
| 155 |  |
| 156 | // 状态动作1-计时暂停，LED灯全灭 |
| 157 | for (i = 0; i < 8; i++) |
| 158 | { |
| 159 | \*(led + i) = 0; |
| 160 | } |
| 161 | break; |
| 162 | case 2: |
| 163 | // 状态动作2-立即刷新数码管的值，然后返回上一状态号(pre\_state). |
| 164 | digit[0] = digit[1] = digit[2] = digit[3] = (flag\_DecCnt ? 9 : 0); |
| 165 | cur\_state = pre\_state; |
| 166 | break; |
| 167 | default: |
| 168 | break; |
| 169 | } |
| 170 |  |
| 171 | if (clock100ms\_flag == 1) // 检查0.1秒定时是否到 |
| 172 | { |
| 173 | clock100ms\_flag = 0; |
| 174 | // [仅状态0有效] 每0.1秒累加计时值在数码管上以十进制显示 |
| 175 | if (cur\_state == 0) |
| 176 | { |
| 177 | // 减计数 |
| 178 | // 要避免uint类型的变量发生下溢，因可能引发异常. |
| 179 | if (flag\_DecCnt) |
| 180 | { |
| 181 | if (!test\_counter) |
| 182 | { |
| 183 | test\_counter = 10000; |
| 184 | } |
| 185 | --test\_counter; |
| 186 | } |
| 187 |  |
| 188 | // 加计数 |
| 189 | if (!flag\_DecCnt && ++test\_counter >= 10000) |
| 190 | { |
| 191 | test\_counter = 0; |
| 192 | } |
| 193 |  |
| 194 | // 刷新数码管显示 |
| 195 | digit[0] = test\_counter / 1000;     // 计算百位数 |
| 196 | digit[1] = test\_counter / 100 % 10; // 计算十位数 |
| 197 | digit[2] = test\_counter / 10 % 10;  // 计算个位数 |
| 198 | digit[3] = test\_counter % 10;       // 计算百分位数 |
| 199 | } |
| 200 | } |
| 201 |  |
| 202 | if (clock500ms\_flag == 1) // 检查0.5秒定时是否到 |
| 203 | { |
| 204 | clock500ms\_flag = 0; |
| 205 | } |
| 206 | } |
| 207 | } |
| 208 |  |
| 209 | //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* |
| 210 | // |
| 211 | // 函数原型：void SysTick\_Handler(void) |
| 212 | // 函数功能：SysTick中断服务程序 |
| 213 | // 函数参数：无 |
| 214 | // 函数返回值：无 |
| 215 | // |
| 216 | //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* |
| 217 | void SysTick\_Handler(void) // 定时周期为20ms |
| 218 | { |
| 219 | // 0.1秒钟软定时器计数 |
| 220 | if (++clock100ms >= V\_T100ms) |
| 221 | { |
| 222 | clock100ms\_flag = 1; // 当0.1秒到时，溢出标志置1 |
| 223 | clock100ms = 0; |
| 224 | } |
| 225 |  |
| 226 | // 0.5秒钟软定时器计数 |
| 227 | if (++clock500ms >= V\_T500ms) |
| 228 | { |
| 229 | clock500ms\_flag = 1; // 当0.5秒到时，溢出标志置1 |
| 230 | clock500ms = 0; |
| 231 | } |
| 232 |  |
| 233 | // 刷新全部数码管和LED指示灯 |
| 234 | TM1638\_RefreshDIGIandLED(digit, pnt, led); |
| 235 |  |
| 236 | // 检查当前键盘输入，0代表无键操作，1-9表示有对应按键 |
| 237 | // 键号显示在一位数码管上 |
| 238 | key\_code = TM1638\_Readkeyboard(); |
| 239 | } |
| 240 |  |

## 实验小结

在本实验中，我们的主要工作如下：

（1）学习有限状态机（FSM）的思想，用其完成了实验任务。本实验要求实现的功能包含多个彼此联系的工作模式，故采用FSM的设计思路是比较自然的。

（2）学习通过TM4C1294微控制器程控TM1638芯片，通过TM1638管理A2000TM4扩展板上资源（包括LED灯、数码管和键盘）的方法。TM4C1294微处理器通过自身GPIO引脚向TM1638芯片发出控制信号。课程组提供了封装好的TM1638控制程序段供本实验调用。

（3）熟悉Systick中断调用方式，学习相应的编程技能。利用Systick中断，我们实现了相对准确的软定时功能。

（4）学习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。中断机制的引入，在主程序和不同的中断服务程序（ISR）中安排不同的任务，可在一定程度上实现多任务处理。

（5）初步掌握了简易的人机操作界面的设计技巧。在本实验中，我们利用A2000TM4扩展板上的数码管、LED指示灯、按键阵列等资源，实现了简单的人机交互功能。

# 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

#### 任务要求

修改程序 exp3\_0.c，同时另存命名为 exp3\_1.c，完成实验任务 3\_1。实验要求把例程 exp3\_0.c 改写成非阻塞方式，即中断方式进行发送与接收。

#### 实现思路

修改UART初始化程序，增加UART中断允许以及相应的端口使能函数；

删除示例程序原本的实现部分，将其转移到提供的UART中断服务程序中即可；

R-FIFO与T-FIFO的触发深度均选择默认的1/2深度。

#### 实验结果和分析

实验结果已正确满足要求。

### 实验任务3\_2

#### 任务要求

编写实现一个时钟（24 小时制）：

1) 开机或复位后，从左到右 6 位数码管显示 00.00.00，表示 00:00:00(零点零分零秒)。

2) 当 PC 端发送绝对对时命令，如 AT+SET15:04:34，自动将当前时间同步到 15:04:34,数码管 显示为 15.04.34。

3) 当 PC 端发送相对对时命令，如 AT+INC00:00:34，自动将当前时间加 34 秒，并回之以当前 时间。PC 端显示格式为 hh:mm:ss。要求检测时间格式的有效性。

4) 当 PC 端发送查询时间命令，如 AT+GET，自动回之以当前时间。PC 端显示格式为 hh:mm:ss

5) 对 PC 端发送的命令要进行格式有效性的检测,要求 hh 取值 00-23；mm 取值 00-59；ss 取值 00-59。如果输入错误命令或格式不对的命令，微控制器回之以 Error Command!。

#### 实现思路

采用全局变量counter\_hh, counter\_mm, counter\_ss来记录当前系统存储的时间。

实现自动走时时，利用示例程序设定完成的时长20ms的Systick，引入全局变量clock1000ms来记录程序进入系统时钟中断的次数，计数满50次则现实时间过去1s，counter\_ss加1，以此完成系统的自动计时。

在UART串行通信方面同实验2的思路，针对UART收发构建如下FSM：

|  |
| --- |
| 1) 状态定义. 0-IDLE（空闲）；1-BUSY\_RX（接收忙）；2-BUSY\_TX（处理或发送忙）. |
| 2) 状态动作. 0-无；1-接收时间窗计数进行，把进入 UART Rx FIFO 的数据全部存入命令接收缓存区（直至缓存区满）；2-执行命令（若合法）、决定 UART 回传内容、将回传内容（若有）写入命令发送缓存区、置命令发送缓存区相关标志并关写允许（若有写入）、以非阻塞方式将命令发送缓存区的数据（若有）送到 UART Tx FIFO. |
| 3) 状态转移条件. 状态转移条件. C01-发生 `UART\_RX` 或 `UART\_RT` 中断；C12-接收计时结束；C20-命令（若合法）执行完毕，且回传内容（若有）已全部送到 UART Tx FIFO. |
| 4) 状态转移动作. C01-接收时间计数器复位、接收缓存区复位、将第一个字符存入接收缓存区；C12-关命令接收缓存区写允许，发送缓存区复位（包括开写允许）；C20-无. |

中断触发深度为1/8（TX），7/8（RX）。

在处理UART串行获得命令时，首先利用C语言标准库的strcmp函数比对获得命令的帧头是否合规，然后使用条件语句逐个判断具体命令，最后利用sscanf函数获得“hh:mm:ss”格式的字符串，sscanf函数会检测是否匹配并返回匹配的对数，即可以由此判断输入的时间信息是否合规并提取获得的时间信息，在命令格式以及时间格式合规后，进行相应的操作。

#### 实验结果和分析

实验结果已正确满足要求，此外由于状态机的引入，程序能够正确，有序响应高频率的输入。

### 实验任务3\_3

#### 任务要求

编写实现一个时钟（24 小时制）：

1) 开机或复位后，从左到右 6 位数码管显示 00.00.00，表示 00:00:00(零点零分零秒)。

2) 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”和当前时间。如当前时间

为 15:04:34，则 PC 端显示为“现在是中华人民共和国北京时间 15:04:34”。

3) 按照下表修改 UART 的波特率，观察 PC 端的显示结果是否正确，并分析原因。

不同的波特率对 UART 通信的影响

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | UART波特率  （Baud） | 显示结果  是否正确 | 原因 |
| 1 | 230400 |  |  |
| 2 | 115200 |  |  |
| 3 | 9600 |  |  |
| 4 | 1200 |  |  |
| 5 | 300 |  |  |
| 6 | 110 |  |  |

#### 实现思路

对实验3\_2的程序加以改造，在状态机中加入相应消息的发送条件，每当计时器满1秒就执行一次发送。

设定UART发送端与接收端的编码格式均为GBK，波特率默认为115200，进行通信。

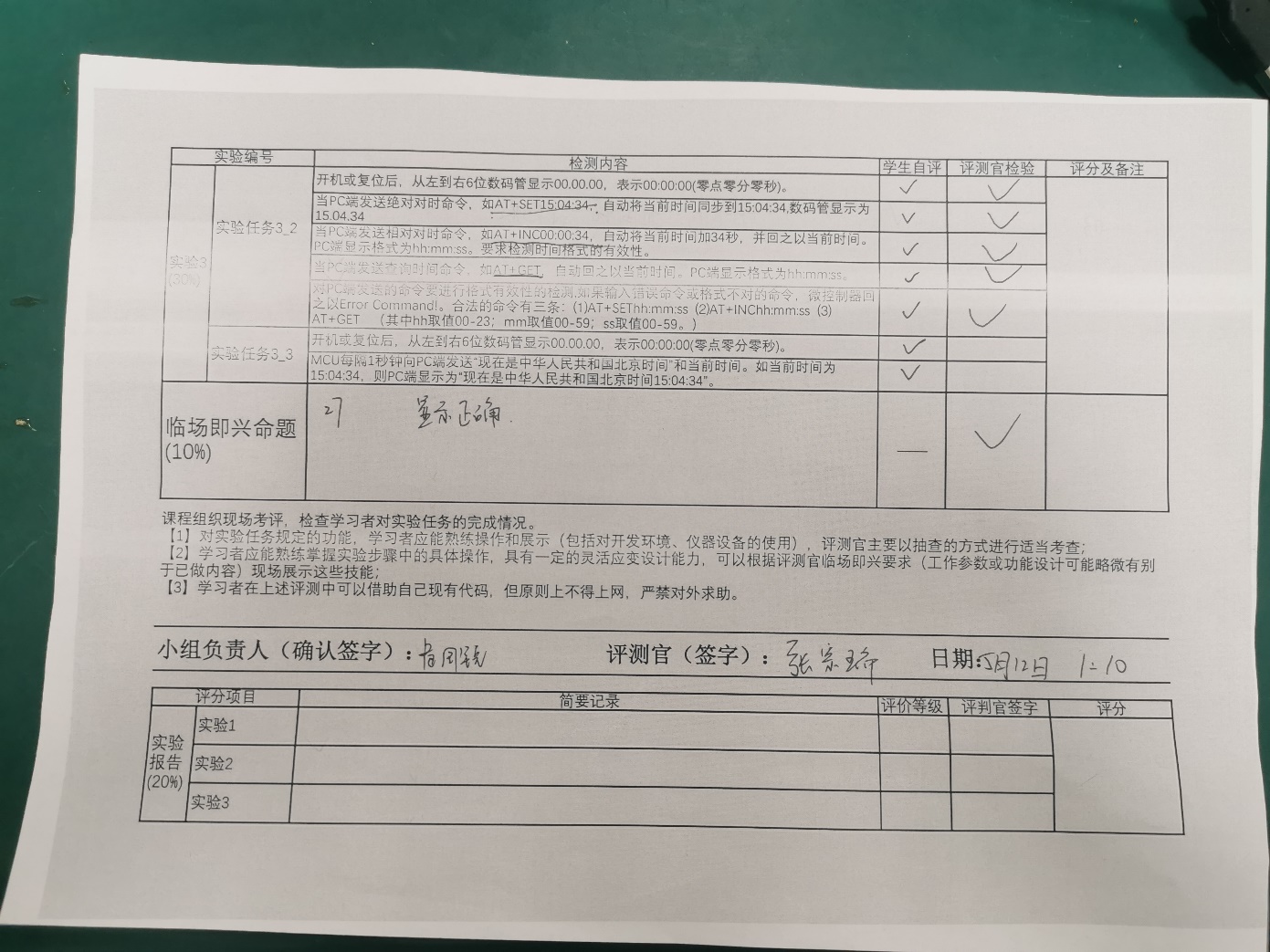
#### 实验结果和分析

成功实现了目标需求。

**表3.1 不同的波特率对UART通信的影响**

| **序号** | **UART波特率**  **（Baud）** | **显示结果**  **是否正确** | **原因** |
| --- | --- | --- | --- |
| 1 | 230400 | 是 |  |
| 2 | 115200 | 是 |  |
| 3 | 9600 | 是 |  |
| 4 | 1200 | 否 | 波特率太低，由于程序的发送机制，无法做到每秒钟返回一次时间信息。 |
| 5 | 300 | 否 | 同上。 |
| 6 | 110 | 否 | 同上。 |

## 实验结果自测记录



## 实验核心代码清单

### 实验任务3\_1

|  |  |
| --- | --- |
| 1 | #ifndef PART\_TM4C1294NCPDT |
| 2 | #define PART\_TM4C1294NCPDT // pin\_map.h 需要 |
| 3 | #endif |
| 4 |  |
| 5 | //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* |
| 6 | // |
| 7 | // 头文件 |
| 8 | // |
| 9 | //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* |
| 10 | #include <stdint.h> |
| 11 | #include <stdbool.h> |
| 12 | #include "inc/hw\_memmap.h"       // 基址宏定义 |
| 13 | #include "inc/hw\_types.h"        // 数据类型宏定义，寄存器访问函数 |
| 14 | #include "driverlib/debug.h"     // 调试用 |
| 15 | #include "driverlib/gpio.h"      // 通用IO口宏定义和函数原型 |
| 16 | #include "driverlib/pin\_map.h"   // TM4C系列MCU外围设备管脚宏定义 |
| 17 | #include "driverlib/sysctl.h"    // 系统控制定义 |
| 18 | #include "driverlib/systick.h"   // SysTick Driver 原型 |
| 19 | #include "driverlib/interrupt.h" // NVIC Interrupt Controller Driver 原型 |
| 20 | #include "driverlib/uart.h"      // 与UART有关的宏定义和函数原型 |
| 21 | #include "tm1638.h"              // 与控制TM1638芯片有关的宏定义和函数原型 |
| 22 | #include "inc/tm4c1294ncpdt.h"   // TM4C1294NCPDT Register Definitions |
| 23 |  |
| 24 | //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* |
| 25 | // |
| 26 | // 宏定义 |
| 27 | // |
| 28 | //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* |
| 29 |  |
| 30 | #define SYSTICK\_FREQUENCY 50 // SysTick频率为50Hz，即循环定时周期20ms |
| 31 | #define V\_T100ms 5           // 0.1s软件定时器溢出值，5个20ms |
| 32 | //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* |
| 33 | // |
| 34 | // 函数原型声明 |
| 35 | // |
| 36 | //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* |
| 37 | void GPIOInit(void);                                         // GPIO初始化 |
| 38 | void SysTickInit(void);                                      // 设置SysTick中断 |
| 39 | void UARTInit(void);                                         // UART初始化 |
| 40 | void DevicesInit(void);                                      // MCU器件初始化，注：会调用上述函数 |
| 41 | void UARTStringPut(uint32\_t ui32Base, const char \*cMessage); // 向UART发送字符串 |
| 42 | //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* |
| 43 | // |
| 44 | // 变量定义 |
| 45 | // |
| 46 | //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* |
| 47 |  |
| 48 | // 8位数码管显示的数字或字母符号 |
| 49 | // 注：板上数码位从左到右序号排列为4、5、6、7、0、1、2、3 |
| 50 | uint8\_t digit[8] = {' ', ' ', ' ', ' ', '\_', 0, '\_', ' '}; |
| 51 |  |
| 52 | // 8位小数点 1亮  0灭 |
| 53 | // 注：板上数码位小数点从左到右序号排列为4、5、6、7、0、1、2、3 |
| 54 | uint8\_t pnt = 0x00; |
| 55 |  |
| 56 | // 8个LED指示灯状态，0灭，1亮 |
| 57 | // 注：板上指示灯从左到右序号排列为7、6、5、4、3、2、1、0 |
| 58 | //     对应元件LED8、LED7、LED6、LED5、LED4、LED3、LED2、LED1 |
| 59 | uint8\_t led[] = {0, 0, 0, 0, 0, 0, 0, 1}; |
| 60 |  |
| 61 | // 当前按键值 |
| 62 | volatile uint8\_t key\_code = 0; |
| 63 |  |
| 64 | // 用于记录前一次按键检测时的键盘状态，0表示无键按下，1有键按下 |
| 65 | volatile uint8\_t key\_state = 0; |
| 66 |  |
| 67 | // 记录系统运行状态 |
| 68 | volatile uint8\_t run\_state = 0; |
| 69 |  |
| 70 | // 系统时钟频率 |
| 71 | uint32\_t ui32SysClock; |
| 72 |  |
| 73 | //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* |
| 74 | // |
| 75 | // 主程序 |
| 76 | // |
| 77 | //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* |
| 78 | int main(void) |
| 79 | { |
| 80 | DevicesInit(); //  MCU器件初始化 |
| 81 |  |
| 82 | SysCtlDelay(60 \* (ui32SysClock / 3000)); // 延时>60ms,等待TM1638上电完成 |
| 83 | TM1638\_Init();                           // 初始化TM1638 |
| 84 |  |
| 85 | while (1) |
| 86 | { |
| 87 | } |
| 88 | } |
| 89 |  |
| 90 | //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* |
| 91 | // |
| 92 | // 函数原型：void UARTStringPut(uint32\_t ui32Base, const char \*cMessage) |
| 93 | // 函数功能：向UART模块发送字符串 |
| 94 | // 函数参数：ui32Base：UART模块 |
| 95 | //          cMessage：待发送字符串 |
| 96 | // 函数返回值：无 |
| 97 | // |
| 98 | //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* |
| 99 | void UARTStringPut(uint32\_t ui32Base, const char \*cMessage) |
| 100 | { |
| 101 | while (\*cMessage != '\0') |
| 102 | UARTCharPut(ui32Base, \*(cMessage++)); |
| 103 | } |
| 104 |  |
| 105 | //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* |
| 106 | // |
| 107 | // 函数原型：void UARTInit(void) |
| 108 | // 函数功能：UART初始化。使能UART0，设置PA0,PA1为UART0 RX,TX引脚； |
| 109 | //          设置波特率及帧格式。 |
| 110 | // 函数参数：无 |
| 111 | // 函数返回值：无 |
| 112 | // |
| 113 | //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* |
| 114 | void UARTInit(void) |
| 115 | { |
| 116 | // 引脚配置 |
| 117 | SysCtlPeripheralEnable(SYSCTL\_PERIPH\_UART0); // 使能UART0模块 |
| 118 | SysCtlPeripheralEnable(SYSCTL\_PERIPH\_GPIOA); // 使能端口 A |
| 119 | while (!SysCtlPeripheralReady(SYSCTL\_PERIPH\_GPIOA)) |
| 120 | ; // 等待端口 A准备完毕 |
| 121 |  |
| 122 | GPIOPinConfigure(GPIO\_PA0\_U0RX); // 设置PA0为UART0 RX引脚 |
| 123 | GPIOPinConfigure(GPIO\_PA1\_U0TX); // 设置PA1为UART0 TX引脚 |
| 124 |  |
| 125 | // 设置端口 A的第0,1位（PA0,PA1）为UART引脚 |
| 126 | GPIOPinTypeUART(GPIO\_PORTA\_BASE, GPIO\_PIN\_0 | GPIO\_PIN\_1); |
| 127 |  |
| 128 | // 波特率及帧格式设置 |
| 129 | UARTConfigSetExpClk(UART0\_BASE, |
| 130 | ui32SysClock, |
| 131 | 115200,                  // 波特率：115200 |
| 132 | (UART\_CONFIG\_WLEN\_8 |    // 数据位：8 |
| 133 | UART\_CONFIG\_STOP\_ONE |  // 停止位：1 |
| 134 | UART\_CONFIG\_PAR\_NONE)); // 校验位：无 |
| 135 |  |
| 136 | IntEnable(INT\_UART0);                                 // UART0中断允许 |
| 137 | UARTIntEnable(UART0\_BASE, UART\_INT\_RX | UART\_INT\_RT); // 使能UART0 RX,RT中断 |
| 138 |  |
| 139 | // 初始化完成后向PC端发送"Hello, 2A!"字符串 |
| 140 | UARTStringPut(UART0\_BASE, (const char \*)"\r\nHello, 2A!\r\n"); |
| 141 | } |
| 142 |  |
| 143 | //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* |
| 144 | // |
| 145 | // 函数原型：void SysTick\_Handler(void) |
| 146 | // 函数功能：SysTick中断服务程序 |
| 147 | // 函数参数：无 |
| 148 | // 函数返回值：无 |
| 149 | // |
| 150 | //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* |
| 151 | void SysTick\_Handler(void) // 定时周期为20ms |
| 152 | { |
| 153 | uint8\_t i; |
| 154 |  |
| 155 | // 刷新全部数码管和LED指示灯 |
| 156 | TM1638\_RefreshDIGIandLED(digit, pnt, led); |
| 157 |  |
| 158 | // 检查当前键盘输入，0代表无键操作，1-9表示有对应按键 |
| 159 | // 键号显示在一位数码管上 |
| 160 | key\_code = TM1638\_Readkeyboard(); |
| 161 |  |
| 162 | // 按键操作在SysTick中断服务程序中的状态转移处理程序 |
| 163 |  |
| 164 | // key\_state用于记录前一次按键检测时的键盘状态，0表示无键按下，1有键按下 |
| 165 | switch (key\_state) |
| 166 | { |
| 167 | case 0:               // 前一次按键检测时无键按下 |
| 168 | if (key\_code > 0) // 本次按键检测有键按下 |
| 169 | { |
| 170 | key\_state = 1; |
| 171 |  |
| 172 | if (key\_code >= 1 && key\_code <= 3) // 编号1-3按键按下 |
| 173 | { |
| 174 | run\_state = key\_code; // 修改系统运行状态 |
| 175 | for (i = 0; i < 8; ++i) |
| 176 | led[i] = 0; // 修改LED灯显示 |
| 177 | led[7 - run\_state] = 1; |
| 178 | digit[5] = run\_state; // 修改显示键号 |
| 179 | } |
| 180 | else // 编号4-9按键按下 |
| 181 | { |
| 182 | run\_state = 0; // 修改系统运行状态 |
| 183 | for (i = 0; i < 8; ++i) |
| 184 | led[i] = 0; // 修改LED灯显示 |
| 185 | led[7] = 1; |
| 186 | digit[5] = 0; // 修改显示键号 |
| 187 | } |
| 188 | } |
| 189 | break; |
| 190 | case 1:                // 前一次按键检测时有键按下 |
| 191 | if (key\_code == 0) // 本次按键检测时无键按下 |
| 192 | { |
| 193 | key\_state = 0; |
| 194 | } |
| 195 | break; |
| 196 | default: |
| 197 | key\_state = 0; |
| 198 | break; |
| 199 | } |
| 200 | } |
| 201 |  |
| 202 | /\*\* |
| 203 | \* @brief UART0 中断服务程序 |
| 204 | \* |
| 205 | \*/ |
| 206 | void UART0\_Handler(void) |
| 207 | { |
| 208 | int32\_t uart0\_int\_status; |
| 209 | uint8\_t uart\_receive\_char; |
| 210 | uart0\_int\_status = UARTIntStatus(UART0\_BASE, true); // 取中断状态 |
| 211 | UARTIntClear(UART0\_BASE, uart0\_int\_status);         // 清中断标志 |
| 212 | while (UARTCharsAvail(UART0\_BASE))                  // 重复从接收 FIFO 读取字符 |
| 213 | { |
| 214 | uart\_receive\_char = UARTCharGetNonBlocking(UART0\_BASE); // 读入一个字符 |
| 215 | switch (run\_state) |
| 216 | { |
| 217 | case 1: // 小写转大写 |
| 218 | if (uart\_receive\_char >= 'a' && uart\_receive\_char <= 'z') |
| 219 | { |
| 220 | uart\_receive\_char = uart\_receive\_char - 'a' + 'A'; |
| 221 | } |
| 222 | break; |
| 223 | case 2: // 大写转小写 |
| 224 | if (uart\_receive\_char >= 'A' && uart\_receive\_char <= 'Z') |
| 225 | { |
| 226 | uart\_receive\_char = uart\_receive\_char - 'A' + 'a'; |
| 227 | } |
| 228 | break; |
| 229 | case 3: // 大小写互换 |
| 230 | if (uart\_receive\_char >= 'a' && uart\_receive\_char <= 'z') |
| 231 | { |
| 232 | uart\_receive\_char = uart\_receive\_char - 'a' + 'A'; |
| 233 | } |
| 234 | else |
| 235 | { |
| 236 | if (uart\_receive\_char >= 'A' && uart\_receive\_char <= 'Z') |
| 237 | { |
| 238 | uart\_receive\_char = uart\_receive\_char - 'A' + 'a'; |
| 239 | } |
| 240 | } |
| 241 | break; |
| 242 | default: |
| 243 | break; |
| 244 | } |
| 245 | // 发送转换好的字符 |
| 246 | UARTCharPutNonBlocking(UART0\_BASE, uart\_receive\_char); |
| 247 | if (uart\_receive\_char == '\r') // 如果发现'\r'补发一个回车 |
| 248 | { |
| 249 | UARTCharPutNonBlocking(UART0\_BASE, '\n'); |
| 250 | } |
| 251 | } |
| 252 | } |
| 253 |  |

### 实验任务3\_2

|  |  |
| --- | --- |
| 1 | #ifndef PART\_TM4C1294NCPDT |
| 2 | #define PART\_TM4C1294NCPDT // pin\_map.h 需要 |
| 3 | #endif |
| 4 |  |
| 5 | //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* |
| 6 | // |
| 7 | // 头文件 |
| 8 | // |
| 9 | //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* |
| 10 | #include <stdint.h> |
| 11 | #include <stdbool.h> |
| 12 | #include "inc/hw\_memmap.h"       // 基址宏定义 |
| 13 | #include "inc/hw\_types.h"        // 数据类型宏定义，寄存器访问函数 |
| 14 | #include "driverlib/debug.h"     // 调试用 |
| 15 | #include "driverlib/gpio.h"      // 通用IO口宏定义和函数原型 |
| 16 | #include "driverlib/pin\_map.h"   // TM4C系列MCU外围设备管脚宏定义 |
| 17 | #include "driverlib/sysctl.h"    // 系统控制定义 |
| 18 | #include "driverlib/systick.h"   // SysTick Driver 原型 |
| 19 | #include "driverlib/interrupt.h" // NVIC Interrupt Controller Driver 原型 |
| 20 | #include "driverlib/uart.h"      // 与UART有关的宏定义和函数原型 |
| 21 | #include "tm1638.h"              // 与控制TM1638芯片有关的宏定义和函数原型 |
| 22 | #include "inc/tm4c1294ncpdt.h"   // TM4C1294NCPDT Register Definitions |
| 23 | #include <ctype.h>               // Character handling functions |
| 24 | #include <string.h>              // C Strings |
| 25 | #include <stdio.h>               // C library to perform Input/Output operations |
| 26 | #include <stdlib.h>              // C Standard General Utilities Library |
| 27 |  |
| 28 | //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* |
| 29 | // |
| 30 | // 宏定义 |
| 31 | // |
| 32 | //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* |
| 33 | #define SYSTICK\_FREQUENCY 50            // SysTick频率为50Hz，即循环定时周期20ms |
| 34 | #define V\_T1000ms 50                    // 1.0s软件定时器溢出值，50个20ms |
| 35 | const uint8\_t CMD\_RX\_BUF\_MAX\_SIZE = 60; // 命令接收缓存区最大容量 |
| 36 | #define CMD\_TX\_BUF\_MAX\_SIZE 60          // 命令发送缓存区最大容量 |
| 37 | #define V\_CMD\_RX\_TIMEOUT 1              // 命令接收超时阈值，1个20ms |
| 38 |  |
| 39 | //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* |
| 40 | // |
| 41 | // 函数原型声明 |
| 42 | // |
| 43 | //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* |
| 44 | void GPIOInit(void);                                         // GPIO初始化 |
| 45 | void SysTickInit(void);                                      // 设置SysTick中断 |
| 46 | void UARTInit(void);                                         // UART初始化 |
| 47 | void DevicesInit(void);                                      // MCU器件初始化，注：会调用上述函数 |
| 48 | void UARTStringPut(uint32\_t ui32Base, const char \*cMessage); // 向UART发送字符串 |
| 49 | void UART\_clock(void);                                       // 基于 UART 的时钟处理函数 |
| 50 |  |
| 51 | //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* |
| 52 | // |
| 53 | // 变量定义 |
| 54 | // |
| 55 | //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* |
| 56 |  |
| 57 | // 软件定时器计数 |
| 58 | volatile uint8\_t clock1000ms = 0; |
| 59 |  |
| 60 | // 软件定时器溢出标志 |
| 61 | volatile uint8\_t clock1000ms\_flag = 0; |
| 62 |  |
| 63 | // 时分秒计数器 |
| 64 | volatile uint8\_t counter\_hh = 0, counter\_mm = 0, counter\_ss = 0; |
| 65 |  |
| 66 | // 8位数码管显示的数字或字母符号 |
| 67 | // 注：板上数码位从左到右序号排列为4、5、6、7、0、1、2、3 |
| 68 | uint8\_t digit[8] = {' ', ' ', ' ', ' ', '\_', 0, '\_', ' '}; |
| 69 |  |
| 70 | // 8位小数点 1亮  0灭 |
| 71 | // 注：板上数码位小数点从左到右序号排列为4、5、6、7、0、1、2、3 |
| 72 | uint8\_t pnt = 0xA2; |
| 73 |  |
| 74 | // 8个LED指示灯状态，0灭，1亮 |
| 75 | // 注：板上指示灯从左到右序号排列为7、6、5、4、3、2、1、0 |
| 76 | //     对应元件LED8、LED7、LED6、LED5、LED4、LED3、LED2、LED1 |
| 77 | uint8\_t led[] = {0, 0, 0, 0, 0, 0, 0, 0}; |
| 78 |  |
| 79 | // 当前按键值 |
| 80 | volatile uint8\_t key\_code = 0; |
| 81 |  |
| 82 | // 系统时钟频率 |
| 83 | uint32\_t ui32SysClock; |
| 84 |  |
| 85 | /\*\* |
| 86 | \* @brief 命令收发缓存区 |
| 87 | \* |
| 88 | \*/ |
| 89 | struct cmd\_buf\_t |
| 90 | { |
| 91 | /\*\* |
| 92 | \* @brief 命令接收缓存区结构体 |
| 93 | \* |
| 94 | \*/ |
| 95 | struct cmd\_Rx\_buf\_t |
| 96 | { |
| 97 | volatile uint8\_t size;                                // 当前缓存区内容长度 |
| 98 | const uint8\_t max\_size;                               // 缓存区最大容量 |
| 99 | volatile unsigned char data[CMD\_RX\_BUF\_MAX\_SIZE + 1]; // 缓存区数据 |
| 100 | volatile bool WriteEnable;                            // True-写允许；False-提示缓存区中有一个待处理的命令 |
| 101 | } Rx;                                                     /\* = {0, CMD\_RX\_BUF\_MAX\_SIZE, {'\0'}, true}; \*/ |
| 102 |  |
| 103 | /\*\* |
| 104 | \* @brief 命令发送缓存区结构体 |
| 105 | \* |
| 106 | \*/ |
| 107 | struct cmd\_Tx\_buf\_t |
| 108 | { |
| 109 | volatile uint8\_t size;                                // 当前缓存区内容长度 |
| 110 | const uint8\_t max\_size;                               // 缓存区最大容量 |
| 111 | volatile unsigned char data[CMD\_TX\_BUF\_MAX\_SIZE + 1]; // 数据 |
| 112 | volatile uint8\_t next\_trans\_index;                    // 下一次将发送的数据的下标. 当达到size时，表示已发送完成. |
| 113 | volatile bool WriteEnable;                            // True-写允许；False-提示缓存区中有未发送完毕的数据 |
| 114 | } Tx;                                                     /\* = {0, cmd\_buf.Tx\_MAX\_SIZE, {'\0'}, 0, true}; \*/ |
| 115 |  |
| 116 | /\*\* |
| 117 | \* FSM 规则： |
| 118 | \* 1. 状态定义. 0-IDLE（空闲）；1-BUSY\_RX（接收忙）；2-BUSY\_TX（处理或发送忙）. |
| 119 | \* 2. 状态动作. 0-无；1-接收时间窗计数进行，把进入 UART Rx FIFO 的数据全部存入命令接收缓存区（直至缓存区满）；2-执行命令（若合法）、决定 UART 回传内容、将回传内容（若有）写入命令发送缓存区、置命令发送缓存区相关标志并关写允许（若有写入）、以\*\*非阻塞方式\*\*将命令发送缓存区的数据（若有）送到 UART Tx FIFO. |
| 120 | \* 3. 状态转移条件. 状态转移条件. C01-发生 `UART\_RX` 或 `UART\_RT` 中断；C12-接收计时结束；C20-命令（若合法）执行完毕，且回传内容（若有）已全部送到 UART Tx FIFO. |
| 121 | \* 4. 状态转移动作. C01-接收时间计数器复位、接收缓存区复位、将第一个字符存入接收缓存区；C12-关命令接收缓存区写允许，发送缓存区复位（包括开写允许）；C20-无. |
| 122 | \*/ |
| 123 | volatile uint8\_t state;          // 当前状态 |
| 124 | volatile uint8\_t Rx\_timeout\_cnt; // 接收时间计数器 |
| 125 | const uint8\_t V\_Rx\_timeout;      // 接收时长 |
| 126 |  |
| 127 | } cmd\_buf = {{0, CMD\_RX\_BUF\_MAX\_SIZE, {'\0'}, true}, {0, CMD\_TX\_BUF\_MAX\_SIZE, {'\0'}, 0, true}, 0, 0, V\_CMD\_RX\_TIMEOUT}; |
| 128 |  |
| 129 | //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* |
| 130 | // |
| 131 | // 主程序 |
| 132 | // |
| 133 | //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* |
| 134 | int main(void) |
| 135 | { |
| 136 | DevicesInit(); //  MCU器件初始化 |
| 137 |  |
| 138 | SysCtlDelay(60 \* (ui32SysClock / 3000)); // 延时>60ms,等待TM1638上电完成 |
| 139 | TM1638\_Init();                           // 初始化TM1638 |
| 140 |  |
| 141 | while (1) |
| 142 | { |
| 143 | if (clock1000ms\_flag == 1) // 检查1.0秒定时是否到 |
| 144 | { |
| 145 | clock1000ms\_flag = 0; |
| 146 | if (++counter\_ss > 59) |
| 147 | { |
| 148 | counter\_ss = 0; |
| 149 | if (++counter\_mm > 59) |
| 150 | { |
| 151 | counter\_mm = 0; |
| 152 | if (++counter\_hh > 23) |
| 153 | { |
| 154 | counter\_hh = 0; |
| 155 | } |
| 156 | } |
| 157 | } |
| 158 | } |
| 159 |  |
| 160 | // 更新数码管显示 |
| 161 | digit[4] = counter\_hh / 10; |
| 162 | digit[5] = counter\_hh % 10; |
| 163 | digit[6] = counter\_mm / 10; |
| 164 | digit[7] = counter\_mm % 10; |
| 165 | digit[0] = counter\_ss / 10; |
| 166 | digit[1] = counter\_ss % 10; |
| 167 |  |
| 168 | UART\_clock(); // 基于 UART 的时钟处理函数 |
| 169 | } |
| 170 | } |
| 171 |  |
| 172 | //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* |
| 173 | // |
| 174 | // 函数原型：void UARTStringPut(uint32\_t ui32Base, const char \*cMessage) |
| 175 | // 函数功能：向UART模块发送字符串 |
| 176 | // 函数参数：ui32Base：UART模块 |
| 177 | //          cMessage：待发送字符串 |
| 178 | // 函数返回值：无 |
| 179 | // |
| 180 | //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* |
| 181 | void UARTStringPut(uint32\_t ui32Base, const char \*cMessage) |
| 182 | { |
| 183 | while (\*cMessage != '\0') |
| 184 | UARTCharPut(ui32Base, \*(cMessage++)); |
| 185 | } |
| 186 |  |
| 187 | //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* |
| 188 | // |
| 189 | // 函数原型：void UARTInit(void) |
| 190 | // 函数功能：UART初始化。使能UART0，设置PA0,PA1为UART0 RX,TX引脚； |
| 191 | //          设置波特率及帧格式。 |
| 192 | // 函数参数：无 |
| 193 | // 函数返回值：无 |
| 194 | // |
| 195 | //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* |
| 196 | void UARTInit(void) |
| 197 | { |
| 198 | // 引脚配置 |
| 199 | SysCtlPeripheralEnable(SYSCTL\_PERIPH\_UART0); // 使能UART0模块 |
| 200 | SysCtlPeripheralEnable(SYSCTL\_PERIPH\_GPIOA); // 使能端口 A |
| 201 | while (!SysCtlPeripheralReady(SYSCTL\_PERIPH\_GPIOA)) |
| 202 | ; // 等待端口 A准备完毕 |
| 203 |  |
| 204 | GPIOPinConfigure(GPIO\_PA0\_U0RX); // 设置PA0为UART0 RX引脚 |
| 205 | GPIOPinConfigure(GPIO\_PA1\_U0TX); // 设置PA1为UART0 TX引脚 |
| 206 |  |
| 207 | // 设置端口 A的第0,1位（PA0,PA1）为UART引脚 |
| 208 | GPIOPinTypeUART(GPIO\_PORTA\_BASE, GPIO\_PIN\_0 | GPIO\_PIN\_1); |
| 209 |  |
| 210 | // 波特率及帧格式设置 |
| 211 | UARTConfigSetExpClk(UART0\_BASE, |
| 212 | ui32SysClock, |
| 213 | 115200,                  // 波特率：115200 |
| 214 | (UART\_CONFIG\_WLEN\_8 |    // 数据位：8 |
| 215 | UART\_CONFIG\_STOP\_ONE |  // 停止位：1 |
| 216 | UART\_CONFIG\_PAR\_NONE)); // 校验位：无 |
| 217 |  |
| 218 | IntEnable(INT\_UART0);                                               // UART0中断允许 |
| 219 | UARTIntEnable(UART0\_BASE, UART\_INT\_RX | UART\_INT\_RT | UART\_INT\_TX); // 使能UART0 RX,RT,TX中断 |
| 220 | UARTFIFOLevelSet(UART0\_BASE, UART\_FIFO\_TX1\_8, UART\_FIFO\_RX7\_8);     // Sets the FIFO level at which interrupts are generated. |
| 221 |  |
| 222 | // 初始化完成后向PC端发送"Hello, 2A!"字符串 |
| 223 | //UARTStringPut(UART0\_BASE, (const char \*)"\r\nHello, 2A!\r\n"); |
| 224 | } |
| 225 |  |
| 226 | //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* |
| 227 | // |
| 228 | // 函数原型：void SysTick\_Handler(void) |
| 229 | // 函数功能：SysTick中断服务程序 |
| 230 | // 函数参数：无 |
| 231 | // 函数返回值：无 |
| 232 | // |
| 233 | //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* |
| 234 | void SysTick\_Handler(void) // 定时周期为20ms |
| 235 | { |
| 236 | // 1.0秒钟软定时器计数 |
| 237 | if (++clock1000ms >= V\_T1000ms) |
| 238 | { |
| 239 | clock1000ms\_flag = 1; // 当1.0秒到时，溢出标志置1 |
| 240 | clock1000ms = 0; |
| 241 | } |
| 242 |  |
| 243 | // 命令接收缓存区计数 |
| 244 | if (cmd\_buf.state == 1 && ++cmd\_buf.Rx\_timeout\_cnt >= cmd\_buf.V\_Rx\_timeout) |
| 245 | { |
| 246 | // C12 |
| 247 | cmd\_buf.Rx.WriteEnable = false; |
| 248 | cmd\_buf.Tx.WriteEnable = true; |
| 249 | memset((void \*)cmd\_buf.Tx.data, 0, cmd\_buf.Tx.max\_size); |
| 250 | cmd\_buf.Tx.next\_trans\_index = cmd\_buf.Tx.size = 0; |
| 251 | cmd\_buf.state = 2; |
| 252 | } |
| 253 |  |
| 254 | // 刷新全部数码管和LED指示灯 |
| 255 | TM1638\_RefreshDIGIandLED(digit, pnt, led); |
| 256 |  |
| 257 | // 检查当前键盘输入，0代表无键操作，1-9表示有对应按键 |
| 258 | // 键号显示在一位数码管上 |
| 259 | key\_code = TM1638\_Readkeyboard(); |
| 260 | } |
| 261 |  |
| 262 | /\*\* |
| 263 | \* @brief UART0 中断服务程序 |
| 264 | \* |
| 265 | \*/ |
| 266 | void UART0\_Handler(void) |
| 267 | { |
| 268 | int32\_t uart0\_int\_status; |
| 269 |  |
| 270 | uart0\_int\_status = UARTIntStatus(UART0\_BASE, true); // 取中断状态 |
| 271 | UARTIntClear(UART0\_BASE, uart0\_int\_status);         // 清中断标志 |
| 272 |  |
| 273 | switch (uart0\_int\_status) |
| 274 | { |
| 275 | case UART\_INT\_RT:                      // Receive Timeout Interrupt |
| 276 | case UART\_INT\_RX:                      // Receive Interrupt |
| 277 | while (UARTCharsAvail(UART0\_BASE)) // 将 UART Rx FIFO 中的数据全部读入命令接收缓存区 |
| 278 | { |
| 279 | uint8\_t uart\_receive\_char = UARTCharGetNonBlocking(UART0\_BASE); // 读入一个字符 |
| 280 | if (!cmd\_buf.state) |
| 281 | { |
| 282 | cmd\_buf.state = 1; |
| 283 |  |
| 284 | // C01 |
| 285 | cmd\_buf.Rx\_timeout\_cnt = 0; |
| 286 | memset((void \*)cmd\_buf.Rx.data, 0, cmd\_buf.Rx.max\_size); |
| 287 | cmd\_buf.Rx.size = 0; |
| 288 | cmd\_buf.Rx.WriteEnable = true; |
| 289 | cmd\_buf.Rx.data[cmd\_buf.Rx.size++] = uart\_receive\_char; |
| 290 | } |
| 291 | else if (cmd\_buf.state == 1) |
| 292 | { |
| 293 | // 若命令接收缓存区已满，则提前终止接收 |
| 294 | if (cmd\_buf.Rx.size >= cmd\_buf.Rx.max\_size) |
| 295 | { |
| 296 | cmd\_buf.Rx.data[cmd\_buf.Rx.max\_size - 1] = '\0'; |
| 297 |  |
| 298 | // C12 |
| 299 | cmd\_buf.Rx.WriteEnable = false; |
| 300 | cmd\_buf.Tx.WriteEnable = true; |
| 301 | memset((void \*)cmd\_buf.Tx.data, 0, cmd\_buf.Tx.max\_size); |
| 302 | cmd\_buf.Tx.next\_trans\_index = cmd\_buf.Tx.size = 0; |
| 303 | cmd\_buf.state = 2; |
| 304 | } |
| 305 | else |
| 306 | { |
| 307 | cmd\_buf.Rx.data[cmd\_buf.Rx.size++] = uart\_receive\_char; |
| 308 | } |
| 309 | } |
| 310 | } |
| 311 | break; |
| 312 | case UART\_INT\_TX: // Transmit Interrupt |
| 313 | // 尝试用命令发送缓存区的待发送数据（若有）填满 Tx FIFO |
| 314 | while (cmd\_buf.state == 2 && !cmd\_buf.Tx.WriteEnable && cmd\_buf.Tx.size && UARTSpaceAvail(UART0\_BASE)) |
| 315 | { |
| 316 | if (UARTCharPutNonBlocking(UART0\_BASE, cmd\_buf.Tx.data[cmd\_buf.Tx.next\_trans\_index])) |
| 317 | { |
| 318 | // 若全部发送完毕，则清空命令发送缓存区，开写允许 |
| 319 | if (++cmd\_buf.Tx.next\_trans\_index >= cmd\_buf.Tx.size) |
| 320 | { |
| 321 | // C20 |
| 322 | cmd\_buf.state = 0; |
| 323 | } |
| 324 | } |
| 325 | } |
| 326 | break; |
| 327 | default: |
| 328 | break; |
| 329 | } |
| 330 | } |
| 331 |  |
| 332 | /\*\* |
| 333 | \* @brief 基于 UART 的时钟处理函数 |
| 334 | \* |
| 335 | \* 首先判断命令接收缓存区 cmd\_buf.Rx 是否存在已接收完毕的待处理命令， |
| 336 | \* 若有，则作出相应处理，以非阻塞方式向命令发送缓存区 cmd\_buf.Tx 填数据； |
| 337 | \* 然后，以非阻塞方式将命令发送缓存区的数据送往 UART 端口的 Tx FIFO. |
| 338 | \*/ |
| 339 | void UART\_clock(void) |
| 340 | { |
| 341 | // 当有接收事件（状态2），且命令发送缓存区 cmd\_buf.Tx 允许写入，则开始处理接收缓存区的数据 |
| 342 | if (cmd\_buf.state == 2 && cmd\_buf.Tx.WriteEnable) |
| 343 | { |
| 344 | // 1. 接收到查询时间命令 |
| 345 | if (!strcmp("AT+GET", (const char \*)cmd\_buf.Rx.data)) |
| 346 | { |
| 347 | // char \*src\_str = (char \*)calloc(cmd\_buf.Tx.max\_size, sizeof(unsigned char)); |
| 348 | char src\_str[CMD\_RX\_BUF\_MAX\_SIZE] = {'\0'}; |
| 349 |  |
| 350 | // 1.1 准备写入命令发送缓存区的数据 |
| 351 | sprintf(src\_str, "%.2hhu:%.2hhu:%.2hhu", |
| 352 | counter\_hh, counter\_mm, counter\_ss); |
| 353 |  |
| 354 | // 1.2 向命令发送缓存区写入数据，并置相应的标志 |
| 355 | strcpy((char \*)cmd\_buf.Tx.data, (const char \*)src\_str); |
| 356 | cmd\_buf.Tx.size = strlen((const char \*)cmd\_buf.Tx.data); |
| 357 | cmd\_buf.Tx.next\_trans\_index = 0; |
| 358 | cmd\_buf.Tx.WriteEnable = false; // 向命令发送缓存区写入完毕，关写允许 |
| 359 |  |
| 360 | // free(src\_str); |
| 361 | } |
| 362 | // 2. 接收到（疑似）绝对对时命令 |
| 363 | else if (!strncmp("AT+SET", (const char \*)cmd\_buf.Rx.data, 6) && strlen((const char \*)cmd\_buf.Rx.data) == strlen("AT+SET12:34:56")) |
| 364 | { |
| 365 | uint8\_t hh, mm, ss; |
| 366 |  |
| 367 | // 2.1 若解析不成功，或解析成功但取值不合规，则准备发送报错信息 |
| 368 | if (sscanf((const char \*)cmd\_buf.Rx.data, |
| 369 | "AT+SET%2hhu:%2hhu:%3hhu", &hh, &mm, &ss) < 3 || |
| 370 | hh > 23 || mm > 59 || ss > 59) |
| 371 | { |
| 372 | strcpy((char \*)cmd\_buf.Tx.data, "Error Command!"); |
| 373 | cmd\_buf.Tx.size = strlen((const char \*)cmd\_buf.Tx.data); |
| 374 | cmd\_buf.Tx.next\_trans\_index = 0; |
| 375 | cmd\_buf.Tx.WriteEnable = false; // 向命令发送缓存区写入完毕，关写允许 |
| 376 | } |
| 377 | // 2.2 匹配成功，执行绝对对时动作 |
| 378 | else |
| 379 | { |
| 380 | counter\_hh = hh; |
| 381 | counter\_mm = mm; |
| 382 | counter\_ss = ss; |
| 383 |  |
| 384 | // C20 |
| 385 | cmd\_buf.state = 0; |
| 386 | } |
| 387 | } |
| 388 | // 3. 接收到（疑似）相对对时命令 |
| 389 | else if (!strncmp("AT+INC", (const char \*)cmd\_buf.Rx.data, 6) && strlen((const char \*)cmd\_buf.Rx.data) == strlen("AT+INC12:34:56")) |
| 390 | { |
| 391 | uint8\_t inc\_hh, inc\_mm, inc\_ss; |
| 392 |  |
| 393 | // 3.1 若解析不成功，或解析成功但取值不合规，则准备发送报错信息 |
| 394 | if (sscanf((const char \*)cmd\_buf.Rx.data, |
| 395 | "AT+INC%2hhu:%2hhu:%2hhu", &inc\_hh, &inc\_mm, &inc\_ss) < 3 || |
| 396 | inc\_hh > 23 || inc\_mm > 59 || inc\_ss > 59) |
| 397 | { |
| 398 | strcpy((char \*)cmd\_buf.Tx.data, "Error Command!"); |
| 399 | cmd\_buf.Tx.size = strlen((const char \*)cmd\_buf.Tx.data); |
| 400 | cmd\_buf.Tx.next\_trans\_index = 0; |
| 401 | cmd\_buf.Tx.WriteEnable = false; // 向命令发送缓存区写入完毕，关写允许 |
| 402 | } |
| 403 | // 3.2 匹配成功，执行相对对时动作，然后向PC端发回当前时间 |
| 404 | else |
| 405 | { |
| 406 | // char \*src\_str = (char \*)calloc(cmd\_buf.Tx.max\_size, sizeof(char)); |
| 407 | char src\_str[CMD\_RX\_BUF\_MAX\_SIZE] = {'\0'}; |
| 408 |  |
| 409 | // 3.2.1 执行相对对时动作 |
| 410 | if ((counter\_ss += inc\_ss) > 59) |
| 411 | { |
| 412 | counter\_mm += counter\_ss / 60; |
| 413 | counter\_ss %= 60; |
| 414 | } |
| 415 | if ((counter\_mm += inc\_mm) > 59) |
| 416 | { |
| 417 | counter\_hh += counter\_mm / 60; |
| 418 | counter\_mm %= 60; |
| 419 | } |
| 420 | if ((counter\_hh += inc\_hh) > 23) |
| 421 | { |
| 422 | counter\_hh %= 24; |
| 423 | } |
| 424 |  |
| 425 | // 3.2.2 准备写入命令发送缓存区的数据 |
| 426 | sprintf(src\_str, "%.2hhu:%.2hhu:%.2hhu", |
| 427 | counter\_hh, counter\_mm, counter\_ss); |
| 428 |  |
| 429 | // 3.2.3 向命令发送缓存区写入数据，并置相应的标志 |
| 430 | strcpy((char \*)cmd\_buf.Tx.data, (const char \*)src\_str); |
| 431 | cmd\_buf.Tx.size = strlen((const char \*)cmd\_buf.Tx.data); |
| 432 | cmd\_buf.Tx.next\_trans\_index = 0; |
| 433 | cmd\_buf.Tx.WriteEnable = false; // 向命令发送缓存区写入完毕，关写允许 |
| 434 |  |
| 435 | // free(src\_str); |
| 436 | } |
| 437 | } |
| 438 | // 4. 接收到非法命令 |
| 439 | else |
| 440 | { |
| 441 | strcpy((char \*)cmd\_buf.Tx.data, "Error Command!"); |
| 442 | cmd\_buf.Tx.size = strlen((const char \*)cmd\_buf.Tx.data); |
| 443 | cmd\_buf.Tx.next\_trans\_index = 0; |
| 444 | cmd\_buf.Tx.WriteEnable = false; // 向命令发送缓存区写入完毕，关写允许 |
| 445 | } |
| 446 | } |
| 447 |  |
| 448 | // 若命令发送缓存区已写入完毕（此时写允许关闭，且缓存区非空）， |
| 449 | // 则尝试用命令发送缓存区的数据（如果有）填满 Tx FIFO |
| 450 | while (cmd\_buf.state == 2 && !cmd\_buf.Tx.WriteEnable && cmd\_buf.Tx.size && UARTSpaceAvail(UART0\_BASE)) |
| 451 | { |
| 452 | if (UARTCharPutNonBlocking(UART0\_BASE, cmd\_buf.Tx.data[cmd\_buf.Tx.next\_trans\_index])) |
| 453 | { |
| 454 | // 若全部发送完毕，则转移至状态0 |
| 455 | if (++cmd\_buf.Tx.next\_trans\_index >= cmd\_buf.Tx.size) |
| 456 | { |
| 457 | // C20 |
| 458 | cmd\_buf.state = 0; |
| 459 | } |
| 460 | } |
| 461 | } |
| 462 | } |
| 463 |  |

### 实验任务3\_3

|  |  |
| --- | --- |
| 1 | #ifndef PART\_TM4C1294NCPDT |
| 2 | #define PART\_TM4C1294NCPDT // pin\_map.h 需要 |
| 3 | #endif |
| 4 |  |
| 5 | //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* |
| 6 | // |
| 7 | // 头文件 |
| 8 | // |
| 9 | //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* |
| 10 | #include <stdint.h> |
| 11 | #include <stdbool.h> |
| 12 | #include "inc/hw\_memmap.h"       // 基址宏定义 |
| 13 | #include "inc/hw\_types.h"        // 数据类型宏定义，寄存器访问函数 |
| 14 | #include "driverlib/debug.h"     // 调试用 |
| 15 | #include "driverlib/gpio.h"      // 通用IO口宏定义和函数原型 |
| 16 | #include "driverlib/pin\_map.h"   // TM4C系列MCU外围设备管脚宏定义 |
| 17 | #include "driverlib/sysctl.h"    // 系统控制定义 |
| 18 | #include "driverlib/systick.h"   // SysTick Driver 原型 |
| 19 | #include "driverlib/interrupt.h" // NVIC Interrupt Controller Driver 原型 |
| 20 | #include "driverlib/uart.h"      // 与UART有关的宏定义和函数原型 |
| 21 | #include "tm1638.h"              // 与控制TM1638芯片有关的宏定义和函数原型 |
| 22 | #include "inc/tm4c1294ncpdt.h"   // TM4C1294NCPDT Register Definitions |
| 23 | #include <ctype.h>               // Character handling functions |
| 24 | #include <string.h>              // C Strings |
| 25 | #include <stdio.h>               // C library to perform Input/Output operations |
| 26 | #include <stdlib.h>              // C Standard General Utilities Library |
| 27 |  |
| 28 | //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* |
| 29 | // |
| 30 | // 宏定义 |
| 31 | // |
| 32 | //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* |
| 33 | #define SYSTICK\_FREQUENCY 50            // SysTick频率为50Hz，即循环定时周期20ms |
| 34 | #define V\_T1000ms 50                    // 1.0s软件定时器溢出值，50个20ms |
| 35 | const uint8\_t CMD\_RX\_BUF\_MAX\_SIZE = 60; // 命令接收缓存区最大容量 |
| 36 | #define CMD\_TX\_BUF\_MAX\_SIZE 60          // 命令发送缓存区最大容量 |
| 37 | #define V\_CMD\_RX\_TIMEOUT 1              // 命令接收超时阈值，1个20ms |
| 38 |  |
| 39 | //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* |
| 40 | // |
| 41 | // 函数原型声明 |
| 42 | // |
| 43 | //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* |
| 44 | void GPIOInit(void);                                         // GPIO初始化 |
| 45 | void SysTickInit(void);                                      // 设置SysTick中断 |
| 46 | void UARTInit(void);                                         // UART初始化 |
| 47 | void DevicesInit(void);                                      // MCU器件初始化，注：会调用上述函数 |
| 48 | void UARTStringPut(uint32\_t ui32Base, const char \*cMessage); // 向UART发送字符串 |
| 49 | void UART\_clock(void);                                       // 基于 UART 的时钟处理函数 |
| 50 |  |
| 51 | //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* |
| 52 | // |
| 53 | // 变量定义 |
| 54 | // |
| 55 | //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* |
| 56 |  |
| 57 | // 软件定时器计数 |
| 58 | volatile uint8\_t clock1000ms = 0; |
| 59 |  |
| 60 | // 软件定时器溢出标志 |
| 61 | volatile uint8\_t clock1000ms\_flag = 0; |
| 62 |  |
| 63 | // 时分秒计数器 |
| 64 | volatile uint8\_t counter\_hh = 12, counter\_mm = 34, counter\_ss = 56; |
| 65 |  |
| 66 | // 8位数码管显示的数字或字母符号 |
| 67 | // 注：板上数码位从左到右序号排列为4、5、6、7、0、1、2、3 |
| 68 | uint8\_t digit[8] = {' ', ' ', ' ', ' ', '\_', 0, '\_', ' '}; |
| 69 |  |
| 70 | // 8位小数点 1亮  0灭 |
| 71 | // 注：板上数码位小数点从左到右序号排列为4、5、6、7、0、1、2、3 |
| 72 | uint8\_t pnt = 0xA2; |
| 73 |  |
| 74 | // 8个LED指示灯状态，0灭，1亮 |
| 75 | // 注：板上指示灯从左到右序号排列为7、6、5、4、3、2、1、0 |
| 76 | //     对应元件LED8、LED7、LED6、LED5、LED4、LED3、LED2、LED1 |
| 77 | uint8\_t led[] = {0, 0, 0, 0, 0, 0, 0, 0}; |
| 78 |  |
| 79 | // 当前按键值 |
| 80 | volatile uint8\_t key\_code = 0; |
| 81 |  |
| 82 | // 系统时钟频率 |
| 83 | uint32\_t ui32SysClock; |
| 84 |  |
| 85 | /\*\* |
| 86 | \* @brief 命令收发缓存区 |
| 87 | \* |
| 88 | \*/ |
| 89 | struct cmd\_buf\_t |
| 90 | { |
| 91 | /\*\* |
| 92 | \* @brief 命令接收缓存区结构体 |
| 93 | \* |
| 94 | \*/ |
| 95 | struct cmd\_Rx\_buf\_t |
| 96 | { |
| 97 | volatile uint8\_t size;                                // 当前缓存区内容长度 |
| 98 | const uint8\_t max\_size;                               // 缓存区最大容量 |
| 99 | volatile unsigned char data[CMD\_RX\_BUF\_MAX\_SIZE + 1]; // 缓存区数据 |
| 100 | volatile bool WriteEnable;                            // True-写允许；False-提示缓存区中有一个待处理的命令 |
| 101 | } Rx;                                                     /\* = {0, CMD\_RX\_BUF\_MAX\_SIZE, {'\0'}, true}; \*/ |
| 102 |  |
| 103 | /\*\* |
| 104 | \* @brief 命令发送缓存区结构体 |
| 105 | \* |
| 106 | \*/ |
| 107 | struct cmd\_Tx\_buf\_t |
| 108 | { |
| 109 | volatile uint8\_t size;                                // 当前缓存区内容长度 |
| 110 | const uint8\_t max\_size;                               // 缓存区最大容量 |
| 111 | volatile unsigned char data[CMD\_TX\_BUF\_MAX\_SIZE + 1]; // 数据 |
| 112 | volatile uint8\_t next\_trans\_index;                    // 下一次将发送的数据的下标. 当达到size时，表示已发送完成. |
| 113 | volatile bool WriteEnable;                            // True-写允许；False-提示缓存区中有未发送完毕的数据 |
| 114 | } Tx;                                                     /\* = {0, cmd\_buf.Tx\_MAX\_SIZE, {'\0'}, 0, true}; \*/ |
| 115 |  |
| 116 | /\*\* |
| 117 | \* FSM 规则： |
| 118 | \* 1. 状态定义. 0-IDLE（空闲）；1-BUSY\_RX（接收忙）；2-BUSY\_TX（处理或发送忙）. |
| 119 | \* 2. 状态动作. 0-无；1-接收时间窗计数进行，把进入 UART Rx FIFO 的数据全部存入命令接收缓存区（直至缓存区满）；2-执行命令（若合法）、决定 UART 回传内容、将回传内容（若有）写入命令发送缓存区、置命令发送缓存区相关标志并关写允许（若有写入）、以\*\*非阻塞方式\*\*将命令发送缓存区的数据（若有）送到 UART Tx FIFO. |
| 120 | \* 3. 状态转移条件. C01-发生 `UART\_RX` 或 `UART\_RT` 中断；C12-接收计时结束；C20-命令（\*\*若有且\*\*合法）执行完毕，且回传内容（若有）已全部送到 UART Tx FIFO；C02-定时1秒到. |
| 121 | \* 4. 状态转移动作. C01-接收时间计数器复位、接收缓存区复位、将第一个字符存入接收缓存区；C12-关命令接收缓存区写允许，发送缓存区复位（包括开写允许）；C20-无；C02-关命令接收缓存区写允许、发送缓存区复位（包括开写允许）、将报时信息写入发送缓存区、关发送缓存区写允许. |
| 122 | \*/ |
| 123 | volatile uint8\_t state;          // 当前状态 |
| 124 | volatile uint8\_t Rx\_timeout\_cnt; // 接收时间计数器 |
| 125 | const uint8\_t V\_Rx\_timeout;      // 接收时长(\*20ms) |
| 126 |  |
| 127 | } cmd\_buf = {{0, CMD\_RX\_BUF\_MAX\_SIZE, {'\0'}, true}, {0, CMD\_TX\_BUF\_MAX\_SIZE, {'\0'}, 0, true}, 0, 0, V\_CMD\_RX\_TIMEOUT}; |
| 128 |  |
| 129 | //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* |
| 130 | // |
| 131 | // 主程序 |
| 132 | // |
| 133 | //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* |
| 134 | int main(void) |
| 135 | { |
| 136 | DevicesInit(); //  MCU器件初始化 |
| 137 |  |
| 138 | SysCtlDelay(60 \* (ui32SysClock / 3000)); // 延时>60ms,等待TM1638上电完成 |
| 139 | TM1638\_Init();                           // 初始化TM1638 |
| 140 |  |
| 141 | while (1) |
| 142 | { |
| 143 | char src\_str[CMD\_RX\_BUF\_MAX\_SIZE] = {'\0'}; |
| 144 | if (clock1000ms\_flag == 1) // 检查1.0秒定时是否到 |
| 145 | { |
| 146 | clock1000ms\_flag = 0; |
| 147 | if (++counter\_ss > 59) |
| 148 | { |
| 149 | counter\_ss = 0; |
| 150 | if (++counter\_mm > 59) |
| 151 | { |
| 152 | counter\_mm = 0; |
| 153 | if (++counter\_hh > 23) |
| 154 | { |
| 155 | counter\_hh = 0; |
| 156 | } |
| 157 | } |
| 158 | } |
| 159 | if (!cmd\_buf.state) |
| 160 | { |
| 161 | // C02 |
| 162 | cmd\_buf.Rx.WriteEnable = false; |
| 163 | cmd\_buf.Tx.WriteEnable = true; |
| 164 | memset((void \*)cmd\_buf.Tx.data, 0, cmd\_buf.Tx.max\_size); |
| 165 | cmd\_buf.Tx.next\_trans\_index = cmd\_buf.Tx.size = 0; |
| 166 |  |
| 167 | // 3.2.2 准备写入命令发送缓存区的数据 |
| 168 | sprintf(src\_str, "现在是中华人民共和国北京时间%.2hhu:%.2hhu:%.2hhu\n", |
| 169 | counter\_hh, counter\_mm, counter\_ss); |
| 170 |  |
| 171 | // 3.2.3 向命令发送缓存区写入数据，并置相应的标志 |
| 172 | strcpy((char \*)cmd\_buf.Tx.data, (const char \*)src\_str); |
| 173 | cmd\_buf.Tx.size = strlen((const char \*)cmd\_buf.Tx.data); |
| 174 | cmd\_buf.Tx.next\_trans\_index = 0; |
| 175 | cmd\_buf.Tx.WriteEnable = false; // 向命令发送缓存区写入完毕，关写允许（将触发一个发送事件） |
| 176 |  |
| 177 | cmd\_buf.state = 2; |
| 178 | } |
| 179 | } |
| 180 |  |
| 181 | // 更新数码管显示 |
| 182 | digit[4] = counter\_hh / 10; |
| 183 | digit[5] = counter\_hh % 10; |
| 184 | digit[6] = counter\_mm / 10; |
| 185 | digit[7] = counter\_mm % 10; |
| 186 | digit[0] = counter\_ss / 10; |
| 187 | digit[1] = counter\_ss % 10; |
| 188 |  |
| 189 | UART\_clock(); // 基于 UART 的时钟处理函数 |
| 190 | } |
| 191 | } |
| 192 |  |
| 193 | //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* |
| 194 | // |
| 195 | // 函数原型：void UARTStringPut(uint32\_t ui32Base, const char \*cMessage) |
| 196 | // 函数功能：向UART模块发送字符串 |
| 197 | // 函数参数：ui32Base：UART模块 |
| 198 | //          cMessage：待发送字符串 |
| 199 | // 函数返回值：无 |
| 200 | // |
| 201 | //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* |
| 202 | void UARTStringPut(uint32\_t ui32Base, const char \*cMessage) |
| 203 | { |
| 204 | while (\*cMessage != '\0') |
| 205 | UARTCharPut(ui32Base, \*(cMessage++)); |
| 206 | } |
| 207 |  |
| 208 | //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* |
| 209 | // |
| 210 | // 函数原型：void UARTInit(void) |
| 211 | // 函数功能：UART初始化。使能UART0，设置PA0,PA1为UART0 RX,TX引脚； |
| 212 | //          设置波特率及帧格式。 |
| 213 | // 函数参数：无 |
| 214 | // 函数返回值：无 |
| 215 | // |
| 216 | //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* |
| 217 | void UARTInit(void) |
| 218 | { |
| 219 | // 引脚配置 |
| 220 | SysCtlPeripheralEnable(SYSCTL\_PERIPH\_UART0); // 使能UART0模块 |
| 221 | SysCtlPeripheralEnable(SYSCTL\_PERIPH\_GPIOA); // 使能端口 A |
| 222 | while (!SysCtlPeripheralReady(SYSCTL\_PERIPH\_GPIOA)) |
| 223 | ; // 等待端口 A准备完毕 |
| 224 |  |
| 225 | GPIOPinConfigure(GPIO\_PA0\_U0RX); // 设置PA0为UART0 RX引脚 |
| 226 | GPIOPinConfigure(GPIO\_PA1\_U0TX); // 设置PA1为UART0 TX引脚 |
| 227 |  |
| 228 | // 设置端口 A的第0,1位（PA0,PA1）为UART引脚 |
| 229 | GPIOPinTypeUART(GPIO\_PORTA\_BASE, GPIO\_PIN\_0 | GPIO\_PIN\_1); |
| 230 |  |
| 231 | // 波特率及帧格式设置 |
| 232 | UARTConfigSetExpClk(UART0\_BASE, |
| 233 | ui32SysClock, |
| 234 | 115200,                  // 波特率：115200 |
| 235 | (UART\_CONFIG\_WLEN\_8 |    // 数据位：8 |
| 236 | UART\_CONFIG\_STOP\_ONE |  // 停止位：1 |
| 237 | UART\_CONFIG\_PAR\_NONE)); // 校验位：无 |
| 238 |  |
| 239 | IntEnable(INT\_UART0);                                               // UART0中断允许 |
| 240 | UARTIntEnable(UART0\_BASE, UART\_INT\_RX | UART\_INT\_RT | UART\_INT\_TX); // 使能UART0 RX,RT,TX中断 |
| 241 | UARTFIFOLevelSet(UART0\_BASE, UART\_FIFO\_TX1\_8, UART\_FIFO\_RX7\_8);     // Sets the FIFO level at which interrupts are generated. |
| 242 |  |
| 243 | // 初始化完成后向PC端发送"Hello, 2A!"字符串 |
| 244 | UARTStringPut(UART0\_BASE, (const char \*)"\r\nHello, 2A!\r\n"); |
| 245 | } |
| 246 |  |
| 247 | //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* |
| 248 | // |
| 249 | // 函数原型：void SysTick\_Handler(void) |
| 250 | // 函数功能：SysTick中断服务程序 |
| 251 | // 函数参数：无 |
| 252 | // 函数返回值：无 |
| 253 | // |
| 254 | //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* |
| 255 | void SysTick\_Handler(void) // 定时周期为20ms |
| 256 | { |
| 257 | // 1.0秒钟软定时器计数 |
| 258 | if (++clock1000ms >= V\_T1000ms) |
| 259 | { |
| 260 | clock1000ms\_flag = 1; // 当1.0秒到时，溢出标志置1 |
| 261 | clock1000ms = 0; |
| 262 | } |
| 263 |  |
| 264 | // 命令接收缓存区计数 |
| 265 | if (cmd\_buf.state == 1 && ++cmd\_buf.Rx\_timeout\_cnt >= cmd\_buf.V\_Rx\_timeout) |
| 266 | { |
| 267 | // C12 |
| 268 | cmd\_buf.Rx.WriteEnable = false; |
| 269 | cmd\_buf.Tx.WriteEnable = true; |
| 270 | memset((void \*)cmd\_buf.Tx.data, 0, cmd\_buf.Tx.max\_size); |
| 271 | cmd\_buf.Tx.next\_trans\_index = cmd\_buf.Tx.size = 0; |
| 272 | cmd\_buf.state = 2; |
| 273 | } |
| 274 |  |
| 275 | // 刷新全部数码管和LED指示灯 |
| 276 | TM1638\_RefreshDIGIandLED(digit, pnt, led); |
| 277 |  |
| 278 | // 检查当前键盘输入，0代表无键操作，1-9表示有对应按键 |
| 279 | // 键号显示在一位数码管上 |
| 280 | key\_code = TM1638\_Readkeyboard(); |
| 281 | } |
| 282 |  |
| 283 | /\*\* |
| 284 | \* @brief UART0 中断服务程序 |
| 285 | \* |
| 286 | \*/ |
| 287 | void UART0\_Handler(void) |
| 288 | { |
| 289 | int32\_t uart0\_int\_status; |
| 290 |  |
| 291 | uart0\_int\_status = UARTIntStatus(UART0\_BASE, true); // 取中断状态 |
| 292 | UARTIntClear(UART0\_BASE, uart0\_int\_status);         // 清中断标志 |
| 293 |  |
| 294 | switch (uart0\_int\_status) |
| 295 | { |
| 296 | case UART\_INT\_RT:                      // Receive Timeout Interrupt |
| 297 | case UART\_INT\_RX:                      // Receive Interrupt |
| 298 | while (UARTCharsAvail(UART0\_BASE)) // 将 UART Rx FIFO 中的数据全部读入命令接收缓存区 |
| 299 | { |
| 300 | uint8\_t uart\_receive\_char = UARTCharGetNonBlocking(UART0\_BASE); // 读入一个字符 |
| 301 | if (!cmd\_buf.state) |
| 302 | { |
| 303 | cmd\_buf.state = 1; |
| 304 |  |
| 305 | // C01 |
| 306 | cmd\_buf.Rx\_timeout\_cnt = 0; |
| 307 | memset((void \*)cmd\_buf.Rx.data, 0, cmd\_buf.Rx.max\_size); |
| 308 | cmd\_buf.Rx.size = 0; |
| 309 | cmd\_buf.Rx.WriteEnable = true; |
| 310 | cmd\_buf.Rx.data[cmd\_buf.Rx.size++] = uart\_receive\_char; |
| 311 | } |
| 312 | else if (cmd\_buf.state == 1) |
| 313 | { |
| 314 | // 若命令接收缓存区已满，则提前终止接收 |
| 315 | if (cmd\_buf.Rx.size >= cmd\_buf.Rx.max\_size) |
| 316 | { |
| 317 | cmd\_buf.Rx.data[cmd\_buf.Rx.max\_size - 1] = '\0'; |
| 318 |  |
| 319 | // C12 |
| 320 | cmd\_buf.Rx.WriteEnable = false; |
| 321 | cmd\_buf.Tx.WriteEnable = true; |
| 322 | memset((void \*)cmd\_buf.Tx.data, 0, cmd\_buf.Tx.max\_size); |
| 323 | cmd\_buf.Tx.next\_trans\_index = cmd\_buf.Tx.size = 0; |
| 324 | cmd\_buf.state = 2; |
| 325 | } |
| 326 | else |
| 327 | { |
| 328 | cmd\_buf.Rx.data[cmd\_buf.Rx.size++] = uart\_receive\_char; |
| 329 | } |
| 330 | } |
| 331 | } |
| 332 | break; |
| 333 | case UART\_INT\_TX: // Transmit Interrupt |
| 334 | // 尝试用命令发送缓存区的待发送数据（若有）填满 Tx FIFO |
| 335 | while (cmd\_buf.state == 2 && !cmd\_buf.Tx.WriteEnable && cmd\_buf.Tx.size && UARTSpaceAvail(UART0\_BASE)) |
| 336 | { |
| 337 | if (UARTCharPutNonBlocking(UART0\_BASE, cmd\_buf.Tx.data[cmd\_buf.Tx.next\_trans\_index])) |
| 338 | { |
| 339 | // 若全部发送完毕，则回到状态0 |
| 340 | if (++cmd\_buf.Tx.next\_trans\_index >= cmd\_buf.Tx.size) |
| 341 | { |
| 342 | // C20 |
| 343 | cmd\_buf.state = 0; |
| 344 | } |
| 345 | } |
| 346 | } |
| 347 | break; |
| 348 | default: |
| 349 | break; |
| 350 | } |
| 351 | } |
| 352 |  |
| 353 | /\*\* |
| 354 | \* @brief 基于 UART 的时钟处理函数 |
| 355 | \* |
| 356 | \* 首先判断命令接收缓存区 cmd\_buf.Rx 是否存在已接收完毕的待处理命令， |
| 357 | \* 若有，则作出相应处理，以非阻塞方式向命令发送缓存区 cmd\_buf.Tx 填数据； |
| 358 | \* 然后，以非阻塞方式将命令发送缓存区的数据送往 UART 端口的 Tx FIFO. |
| 359 | \*/ |
| 360 | void UART\_clock(void) |
| 361 | { |
| 362 | // 当有接收事件（状态2，且命令发送缓存区 cmd\_buf.Tx 允许写入），则开始处理接收缓存区的数据 |
| 363 | if (cmd\_buf.state == 2 && cmd\_buf.Tx.WriteEnable) |
| 364 | { |
| 365 | // 1. 接收到查询时间命令 |
| 366 | if (!strcmp("AT+GET", (const char \*)cmd\_buf.Rx.data)) |
| 367 | { |
| 368 | // char \*src\_str = (char \*)calloc(cmd\_buf.Tx.max\_size, sizeof(unsigned char)); |
| 369 | char src\_str[CMD\_RX\_BUF\_MAX\_SIZE] = {'\0'}; |
| 370 |  |
| 371 | // 1.1 准备写入命令发送缓存区的数据 |
| 372 | sprintf(src\_str, "%.2hhu:%.2hhu:%.2hhu", |
| 373 | counter\_hh, counter\_mm, counter\_ss); |
| 374 |  |
| 375 | // 1.2 向命令发送缓存区写入数据，并置相应的标志 |
| 376 | strcpy((char \*)cmd\_buf.Tx.data, (const char \*)src\_str); |
| 377 | cmd\_buf.Tx.size = strlen((const char \*)cmd\_buf.Tx.data); |
| 378 | cmd\_buf.Tx.next\_trans\_index = 0; |
| 379 | cmd\_buf.Tx.WriteEnable = false; // 向命令发送缓存区写入完毕，关写允许 |
| 380 |  |
| 381 | // free(src\_str); |
| 382 | } |
| 383 | // 2. 接收到（疑似）绝对对时命令 |
| 384 | else if (!strncmp("AT+SET", (const char \*)cmd\_buf.Rx.data, 6) && strlen((const char \*)cmd\_buf.Rx.data) == strlen("AT+SET12:34:56")) |
| 385 | { |
| 386 | uint8\_t hh, mm, ss; |
| 387 |  |
| 388 | // 2.1 若解析不成功，或解析成功但取值不合规，则准备发送报错信息 |
| 389 | if (sscanf((const char \*)cmd\_buf.Rx.data, |
| 390 | "AT+SET%2hhu:%2hhu:%3hhu", &hh, &mm, &ss) < 3 || |
| 391 | hh > 23 || mm > 59 || ss > 59) |
| 392 | { |
| 393 | strcpy((char \*)cmd\_buf.Tx.data, "Error Command!"); |
| 394 | cmd\_buf.Tx.size = strlen((const char \*)cmd\_buf.Tx.data); |
| 395 | cmd\_buf.Tx.next\_trans\_index = 0; |
| 396 | cmd\_buf.Tx.WriteEnable = false; // 向命令发送缓存区写入完毕，关写允许 |
| 397 | } |
| 398 | // 2.2 匹配成功，执行绝对对时动作 |
| 399 | else |
| 400 | { |
| 401 | counter\_hh = hh; |
| 402 | counter\_mm = mm; |
| 403 | counter\_ss = ss; |
| 404 |  |
| 405 | // C20 |
| 406 | cmd\_buf.state = 0; |
| 407 | } |
| 408 | } |
| 409 | // 3. 接收到（疑似）相对对时命令 |
| 410 | else if (!strncmp("AT+INC", (const char \*)cmd\_buf.Rx.data, 6) && strlen((const char \*)cmd\_buf.Rx.data) == strlen("AT+INC12:34:56")) |
| 411 | { |
| 412 | uint8\_t inc\_hh, inc\_mm, inc\_ss; |
| 413 |  |
| 414 | // 3.1 若解析不成功，或解析成功但取值不合规，则准备发送报错信息 |
| 415 | if (sscanf((const char \*)cmd\_buf.Rx.data, |
| 416 | "AT+INC%2hhu:%2hhu:%2hhu", &inc\_hh, &inc\_mm, &inc\_ss) < 3 || |
| 417 | inc\_hh > 23 || inc\_mm > 59 || inc\_ss > 59) |
| 418 | { |
| 419 | strcpy((char \*)cmd\_buf.Tx.data, "Error Command!"); |
| 420 | cmd\_buf.Tx.size = strlen((const char \*)cmd\_buf.Tx.data); |
| 421 | cmd\_buf.Tx.next\_trans\_index = 0; |
| 422 | cmd\_buf.Tx.WriteEnable = false; // 向命令发送缓存区写入完毕，关写允许 |
| 423 | } |
| 424 | // 3.2 匹配成功，执行相对对时动作，然后向PC端发回当前时间 |
| 425 | else |
| 426 | { |
| 427 | // char \*src\_str = (char \*)calloc(cmd\_buf.Tx.max\_size, sizeof(char)); |
| 428 | char src\_str[CMD\_RX\_BUF\_MAX\_SIZE] = {'\0'}; |
| 429 |  |
| 430 | // 3.2.1 执行相对对时动作 |
| 431 | if ((counter\_ss += inc\_ss) > 59) |
| 432 | { |
| 433 | counter\_mm += counter\_ss / 60; |
| 434 | counter\_ss %= 60; |
| 435 | } |
| 436 | if ((counter\_mm += inc\_mm) > 59) |
| 437 | { |
| 438 | counter\_hh += counter\_mm / 60; |
| 439 | counter\_mm %= 60; |
| 440 | } |
| 441 | if ((counter\_hh += inc\_hh) > 23) |
| 442 | { |
| 443 | counter\_hh %= 24; |
| 444 | } |
| 445 |  |
| 446 | // 3.2.2 准备写入命令发送缓存区的数据 |
| 447 | sprintf(src\_str, "%.2hhu:%.2hhu:%.2hhu", |
| 448 | counter\_hh, counter\_mm, counter\_ss); |
| 449 |  |
| 450 | // 3.2.3 向命令发送缓存区写入数据，并置相应的标志 |
| 451 | strcpy((char \*)cmd\_buf.Tx.data, (const char \*)src\_str); |
| 452 | cmd\_buf.Tx.size = strlen((const char \*)cmd\_buf.Tx.data); |
| 453 | cmd\_buf.Tx.next\_trans\_index = 0; |
| 454 | cmd\_buf.Tx.WriteEnable = false; // 向命令发送缓存区写入完毕，关写允许 |
| 455 |  |
| 456 | // free(src\_str); |
| 457 | } |
| 458 | } |
| 459 | // 4. 接收到非法命令 |
| 460 | else |
| 461 | { |
| 462 | strcpy((char \*)cmd\_buf.Tx.data, "Error Command!"); |
| 463 | cmd\_buf.Tx.size = strlen((const char \*)cmd\_buf.Tx.data); |
| 464 | cmd\_buf.Tx.next\_trans\_index = 0; |
| 465 | cmd\_buf.Tx.WriteEnable = false; // 向命令发送缓存区写入完毕，关写允许 |
| 466 | } |
| 467 | } |
| 468 |  |
| 469 | // 若命令发送缓存区已写入完毕（此时写允许关闭，且缓存区非空）， |
| 470 | // 则尝试用命令发送缓存区的数据（如果有）填满 Tx FIFO |
| 471 | while (cmd\_buf.state == 2 && !cmd\_buf.Tx.WriteEnable && cmd\_buf.Tx.size && UARTSpaceAvail(UART0\_BASE)) |
| 472 | { |
| 473 | if (UARTCharPutNonBlocking(UART0\_BASE, cmd\_buf.Tx.data[cmd\_buf.Tx.next\_trans\_index])) |
| 474 | { |
| 475 | // 若全部发送完毕，则转移至状态0 |
| 476 | if (++cmd\_buf.Tx.next\_trans\_index >= cmd\_buf.Tx.size) |
| 477 | { |
| 478 | // C20 |
| 479 | cmd\_buf.state = 0; |
| 480 | } |
| 481 | } |
| 482 | } |
| 483 | } |
| 484 |  |

## 实验小结

在本实验中，我们的主要工作如下：

（1）进一步利用有限状态机（FSM）的思想，实现了接收与发送的有序管理；

（2）学习UART串行通讯的使用方法与设计思想，完成了PC端与开发板的串口通讯，了解阻塞式与非阻塞式两种发送方式，并学会利用UART中断来进行通讯；

（3）利用Systick循环50次的软计时，来完成秒计时的时钟系统，进一步延续了实验2的思想；

（4）了解不同文本编码方式对于串行通讯的影响；

（5）学习使用串口通讯软件，一定程度上了解了辅助软件程序的应用技巧。

# 参考文献

[1] Tiva™ TM4C1294NCPDT Microcontroller DATA SHEET ( DS-TM4C1294NCPDT-15863.2743 SPMS433B )[R]. Texas Instruments Incorporated, 2014.

[2] TivaWare™ Peripheral Driver Library USER’S GUIDE ( SW-TM4C-DRL-UG-2.1.4.178 )[R]. Texas Instruments Incorporated, 2017.

[3] Tiva™ C Series TM4C1294 Connected LaunchPad Evaluation Kit EK-TM4C1294XL User's Guide” ( SPMU365B )[R]. Texas Instruments Incorporated, 2017.

[4] 上海交大电子工程系. 实验1\_时钟选择与GPIO操作[EB/OL]. <https://oc.sjtu.edu.cn/courses/30811/files/2446875?module_item_id=431458>.

[5] 上海交大电子工程系. 工程实践与科技创新II-A\_1.5常用调试方法[EB/OL]. <https://vshare.sjtu.edu.cn/play/b9c32627cc259b7309177662d3e6f0ce>.

[6] 上海交大电子工程系. 实验2\_A2000TM4扩展板使用方法与SYSTICK中断实验[EB/OL]. https://oc.sjtu.edu.cn/courses/30811/files/2446876?module\_item\_id=431459.

[7] 上海交大电子工程系. 工程实践与科技创新II-A\_2.2定时扫描法检测按键[EB/OL]. <https://vshare.sjtu.edu.cn/play/cbc1b104598c0259b9a4de6087421c08>.

[8] 上海交大电子工程系. 工程实践与科技创新II-A\_3.1UART串口通信[EB/OL]. https://vshare.sjtu.edu.cn/play/f1717b5a1c71ecd3d7eaacd6d6ffac31.

1. C01表示从状态0转移到状态1，以此类推。 [↑](#footnote-ref-1)