

答案：

1~9列：

0x22222222

0x55555555

0x88888888

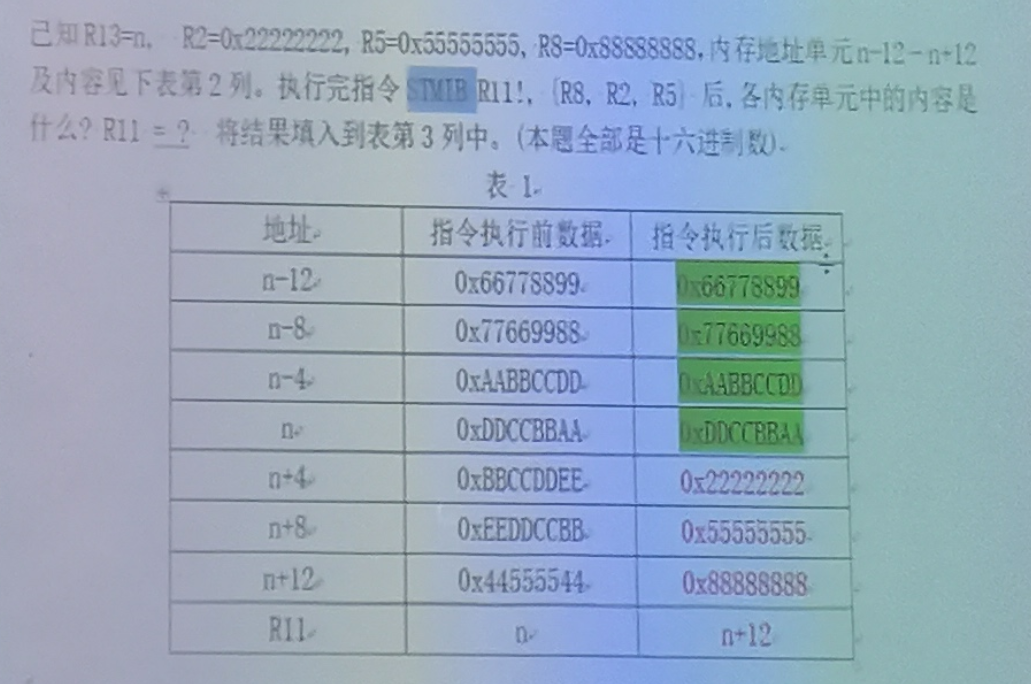
DDCCBBAA

BBCCDDEE

EEDDCCBB

44555544

n-12



将 R0 寄存器的内容乘以 9

ADD R0,R0,R0 ,LSL #3

将 R4 寄存器的 3、5、7 位清 0

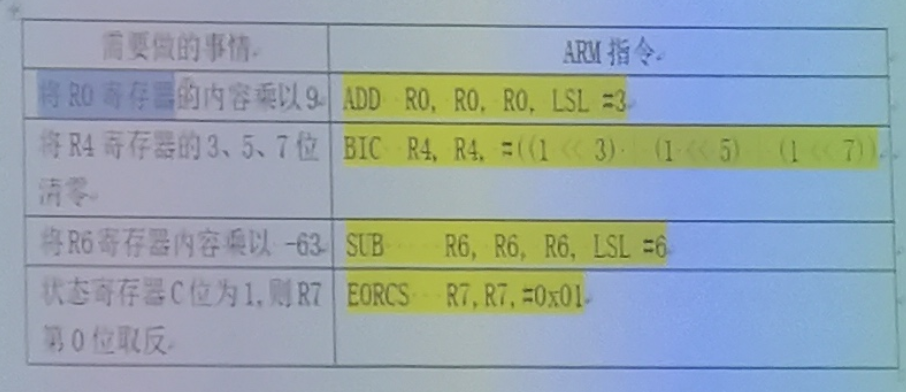
BIC R4,R4 #((1<<3)|(1<<5)|(1<<7))

若当前状态寄存器 C 位为 1,则 R7 第 0 位取反

EORCS R7,R7,#0X01

将当前状态寄存器的内容复制到 R0

MRS R0,CPSR



1.复位电路在上电以及复位的时候控制 CPU 回到原初状态，复位电路设计的好坏，直接影响到整个系统的可靠性。

1)请依据你所掌握的专业知识，谈谈如果系统没有可靠复位可能产生的故障现象；

2)某工程师设计的 RC 复位电路如下图所示，系统实际工作中偶尔出现上电后软件不能正常执行的现象，但按下按键却 可以正常工作，请分析原因并提出改进方案。

答：1）

· 上电异常启动：系统在上电时可能不会进入正确的工作状态，导致软件无法正常执行或执行错误的操作。

· 随机故障：系统在运行过程中可能会出现随机的、不确定的故障，表现为间歇性的崩溃、死机或不响应。

· 数据损坏：如果系统在未完全复位的情况下开始工作，可能会导致寄存器、内存或外设中的数据不一致或损坏。

· 系统不稳定：系统可能会表现出不稳定的行为，例如频繁重启、不可预测的行为或无法执行预期的功能。

· 安全隐患：在某些安全关键应用中，复位不可靠可能会导致系统无法进入安全状态，增加安全风险。

2）RC复位电路的分析与改进方案

分析原因

1. 复位信号不可靠：RC复位电路在上电时可能没有提供足够长的复位信号，使得系统在复位信号还没有完全建立之前就已经开始执行代码。
2. 电容充放电时间不一致：RC电路的充电时间常数可能不合适，导致复位信号的有效时间不足，特别是在上电瞬间，电源电压上升较快时，电容可能没有足够时间充电到阈值电压。
3. 电源波动：在上电过程中，电源电压可能存在波动，导致复位信号不稳定。

改进方案

1. 使用专用复位IC：可以使用专用的复位IC（如Maxim、Microchip等公司的复位芯片），这些IC能够提供可靠的复位信号，并具有固定的复位时间和电压监测功能。
2. 调整RC电路参数：如果继续使用RC复位电路，可以通过调整电阻和电容的值，增加RC时间常数，以确保复位信号持续足够长的时间。通常，增加电容的容量或电阻的阻值可以延长复位信号的持续时间。
3. 增加二极管和施密特触发器：在RC电路中可以增加一个二极管来加速电容的放电，并使用施密特触发器来提供稳定的复位信号。施密特触发器能够将缓慢变化的信号转换为干净的数字信号。
4. 复位按键去抖动：如果使用按键复位，确保按键去抖动电路的设计，以避免按键抖动引起的复位信号不稳定。
5. 电源良好检测：增加一个电源良好（Power Good）检测电路，确保电源稳定后再产生复位信号，这可以避免电源波动引起的复位信号不可靠问题。

六、编程题（2 小题，每小题 12 分，共 24 分）

1、C 语言中有一个标准库函数 strcat,原型为：void strcat(char \*str1, char \*str2);它的用途是将字符串 str2 复 制到 str1 的末尾。请用 ARM 汇编语言编写一个完整的替换程序，实现字符串的连接。需要将 my\_strcat 编写为函数， 同时编写一个 main 函数调用 my\_strcat 函数。

答：

main：

int main(void){

char str\_src[]="Hello world!";

char str\_dst[64]

my\_strcpy(str\_dst,str\_src);

printf("Destination:%s\n",str\_dst);

return 0;

}

汇编：

AREA MyCODE,CODE,READONLY

EXPORT my\_strcpy

my\_strcpy

LDRB R2,[R1],#1

STRB R2,[R0],#1

CMP R2,#0

BNE my\_strcpy

MOV PC,LR

END

void my\_strcpy(char \*str1, char \*str2) {

char \*ptr = str1;

// 移动 ptr 到 str1 的末尾

while (\*ptr != '\0') {

ptr++;

}

// 复制 str2 到 str1 的末尾

while (\*str2 != '\0') {

\*ptr = \*str2;

ptr++;

str2++;

}

// 添加字符串结束标志 '\0'

\*ptr = '\0';

}

void my\_strcat2(char \*str1, char \*str2){

strcat(str1,str2);

}

2、嵌入式设计时为了节省 GPIO 资源，常采用矩阵键盘设计方式，又称为行列式键盘，它是由若干个分别位于行和列

的按键组成的开关矩阵，如下图所示。

1）请提供你的按键驱动程序设计思路；（4 分）

2）以 S3C44B0X 为处理器,F 口对应引脚为键盘接口，给出你的驱动程序相关代码。（

矩阵键盘的设计目的是通过少量的GPIO引脚来读取大量的按键状态。设计思路如下：

硬件连接：

行引脚连接到GPIO端口的输出引脚。

列引脚连接到GPIO端口的输入引脚。

扫描原理：

每次将一行引脚设为低电平（其余设为高电平）。

读取所有列引脚的状态，如果某个列引脚为低电平，说明相应行列的按键被按下。

重复上述过程直到扫描完所有行引脚。

去抖动处理：

读取到按键按下状态后，等待一段时间再次读取确认按键状态，避免因机械按键抖动导致的误判。

#include <stdint.h>

#include <stdio.h>

#include "s3c44b0x.h" // 假设有一个头文件定义了寄存器和GPIO宏

#define ROWS 4

#define COLS 4

// 定义行和列的引脚对应关系（假设F口用于键盘接口）

uint8\_t row\_pins[ROWS] = {0, 1, 2, 3}; // F0, F1, F2, F3

uint8\_t col\_pins[COLS] = {4, 5, 6, 7}; // F4, F5, F6, F7

// 硬件初始化

void keypad\_init(void) {

// 配置行引脚为输出，列引脚为输入

for (int i = 0; i < ROWS; i++) {

GPIOF->OUTENB |= (1 << row\_pins[i]);

GPIOF->DAT |= (1 << row\_pins[i]); // 默认高电平

}

for (int i = 0; i < COLS; i++) {

GPIOF->OUTENB &= ~(1 << col\_pins[i]);

}

}

// 读取列引脚状态

uint8\_t read\_cols(void) {

uint8\_t cols = 0;

for (int i = 0; i < COLS; i++) {

cols |= ((GPIOF->DAT >> col\_pins[i]) & 1) << i;

}

return cols;

}

// 扫描按键

void scan\_keypad(void) {

for (int row = 0; row < ROWS; row++) {

// 设置当前行低电平，其他行高电平

for (int i = 0; i < ROWS; i++) {

if (i == row) {

GPIOF->DAT &= ~(1 << row\_pins[i]);

} else {

GPIOF->DAT |= (1 << row\_pins[i]);

}

}

// 延时，确保电平稳定

delay(1);

// 读取列状态

uint8\_t cols = read\_cols();

if (cols != 0x0F) { // 0x0F 表示所有列都是高电平

for (int col = 0; col < COLS; col++) {

if ((cols & (1 << col)) == 0) {

printf("Key pressed at row %d, col %d\n", row, col);

// 可在此处进行去抖动处理、状态记录等

}

}

}

}

}

// 主函数

int main(void) {

// 初始化

keypad\_init();

// 主循环

while (1) {

scan\_keypad();

// 适当延时避免占用过多CPU时间

delay(50);

}

return 0;

}

// 简单的延时函数（可根据实际情况调整）

void delay(int ms) {

for (volatile int i = 0; i < ms \* 1000; i++) {

// 简单延时

}

}

7.、请根据以下要求完成嵌入式应用系统设计：系统工作时，LED 灯周期性交替闪烁并监听串口输入；当用户通过串口 从 PC 端输入’P’则 LED 暂停闪烁；暂停状态下输入’C’则 LED 继续闪烁；可输入数值改变 LED 闪烁周期；

1）请陈述你的主要设计方案，绘制硬件功能框图和软件主流程，（6 分）

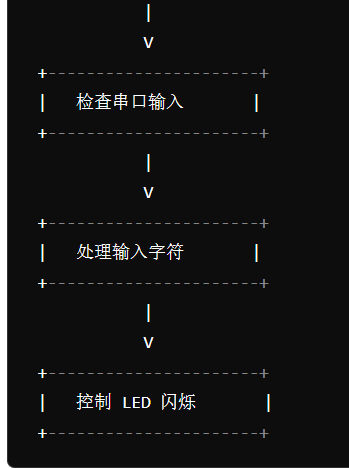
2）请编写软件主程序、UART 的初始化及接收函数、LED 及其他主要软件功能代码；（8 分）

3）请说明你在开发调试过程中使用的主要软硬件工具及方法，并给出你的系统测试方案。（6 分）

1)本设计实现了一个嵌入式系统，通过串口监听用户输入以控制 LED 的闪烁行为。系统的基本功能包括：

* 周期性地交替闪烁 LED。
* 监听串口输入，当接收到特定字符时控制 LED 的闪烁行为。
* 可以通过串口输入数值来改变 LED 闪烁的周期。





2)

主程序：

#include <stdint.h>

#include <stdbool.h>

#include "uart.h"

#include "gpio.h"

#include "timer.h"

volatile bool led\_paused = false;

volatile uint32\_t led\_period = 500; // 默认闪烁周期 500ms

void main() {

SystemInit(); // 系统初始化

UART\_Init(); // UART 初始化

LED\_Init(); // LED 初始化

Timer\_Init(); // 定时器初始化

while (1) {

char input = UART\_Read();

if (input == 'P') {

led\_paused = true; // 暂停 LED 闪烁

} else if (input == 'C') {

led\_paused = false; // 继续 LED 闪烁

} else if (input >= '0' && input <= '9') {

led\_period = (input - '0') \* 100; // 改变 LED 闪烁周期

}

}

}

UART 初始化及接收函数：

#include "uart.h"

#include "stm32f4xx.h"

void UART\_Init() {

// 配置 UART 外设

RCC\_APB2PeriphClockCmd(RCC\_APB2Periph\_USART1, ENABLE);

// 配置引脚、波特率、数据位等

// 略去详细配置代码，假设已配置完毕

}

char UART\_Read() {

while (USART\_GetFlagStatus(USART1, USART\_FLAG\_RXNE) == RESET);

return USART\_ReceiveData(USART1);

}

LED 控制及定时器中断：

#include "gpio.h"

#include "timer.h"

#include "stm32f4xx.h"

void LED\_Init() {

// 配置 LED GPIO 引脚

RCC\_AHB1PeriphClockCmd(RCC\_AHB1Periph\_GPIOA, ENABLE);

GPIO\_InitTypeDef GPIO\_InitStruct;

GPIO\_InitStruct.GPIO\_Pin = GPIO\_Pin\_5;

GPIO\_InitStruct.GPIO\_Mode = GPIO\_Mode\_OUT;

GPIO\_InitStruct.GPIO\_Speed = GPIO\_Speed\_50MHz;

GPIO\_Init(GPIOA, &GPIO\_InitStruct);

}

void Timer\_Init() {

// 配置定时器外设

RCC\_APB1PeriphClockCmd(RCC\_APB1Periph\_TIM2, ENABLE);

TIM\_TimeBaseInitTypeDef TIM\_TimeBaseInitStruct;

TIM\_TimeBaseInitStruct.TIM\_Period = 1000 - 1;

TIM\_TimeBaseInitStruct.TIM\_Prescaler = 84 - 1;

TIM\_TimeBaseInit(TIM2, &TIM\_TimeBaseInitStruct);

TIM\_ITConfig(TIM2, TIM\_IT\_Update, ENABLE);

TIM\_Cmd(TIM2, ENABLE);

NVIC\_EnableIRQ(TIM2\_IRQn);

}

void TIM2\_IRQHandler() {

if (TIM\_GetITStatus(TIM2, TIM\_IT\_Update) != RESET) {

TIM\_ClearITPendingBit(TIM2, TIM\_IT\_Update);

if (!led\_paused) {

GPIO\_ToggleBits(GPIOA, GPIO\_Pin\_5);

}

}

}

#### 3）主要软硬件工具

1. **开发板：** STM32 开发板（如 STM32F4xx 系列）。
2. **调试工具：** JTAG/SWD 调试器（如 ST-Link）。
3. **开发环境：** Keil MDK 或 STM32CubeIDE。
4. **串口调试工具：** 用于在 PC 上发送和接收串口数据。

#### 调试方法

1. **代码调试：** 使用 Keil MDK 或 STM32CubeIDE 进行代码编写和调试，通过 JTAG/SWD 接口进行单步调试。
2. **串口调试：** 使用串口调试工具（如 Putty 或 SecureCRT）与嵌入式系统进行交互，验证串口通信功能。
3. **LED 测试：** 观察 LED 的闪烁状态，验证定时器和 GPIO 控制功能。

### 系统测试方案

1. **上电测试：** 系统上电后，观察 LED 是否按预定周期闪烁。
2. **串口通信测试：** 通过串口发送字符 'P'，观察 LED 是否停止闪烁；发送字符 'C'，观察 LED 是否继续闪烁。
3. **周期改变测试：** 通过串口发送数字字符（0-9），观察 LED 闪烁周期是否相应改变。
4. **综合测试：** 将上述测试组合起来，验证系统整体功能是否正常。