第七章 STM32 处理器基本接口应用

本章将以 STM103V100 评估板为对象，分别介绍 STM32 处理器各种片上常见外围接口，每个接口都会给出相应的应用实例，以帮助读者掌握使用 MDK 进行 STM32 处理器的开发及应用。

7.1 通用 I/O 端口

7.1.1 通用 I/O 端口（GPIO）功能描述

STM32F10x 处理器上共有 7 个 I/O 端口：A、B、C、D、E、F、G，每个 16 个管脚。每个 GPIO 端口都对应有两个 32 位配置寄存器（GPIOx_CRL，GPIOx_CRH）、两个 32 位数据寄存器（GPIOx_IDR，GPIOx_ODR）、一个 32 位置位/复位寄存器（GPIOx_BSRR）、一个 16 位复位寄存器（GPIOx_BRR）和一个 32 位锁定寄存器（GPIOx_LCKR）。注意，I/O 寄存器必须以 32 位字形式访问（不允许半字或字节访问）。

通过配置寄存器可很容易配置每组端口来满足不同系统以及设计的需要，端口的每一位都可以设置为：输入浮空、输入上拉、输入下拉、模拟输入、开漏输出、推挽式输出、复用功能（推挽式）、复用功能（开漏式）等模式之一。在运行程序之前必须对每个用到的管脚功能进行设置，如果某些管脚的复用功能没有使用，可以先将该管脚设置为 I/O 口。I/O 端口的基本结构如图 7-1 所示。所有 GPIO 引脚有一个内部弱上拉和弱下拉电阻，当配置为输入时，它们可以被激活也可以不被激活。

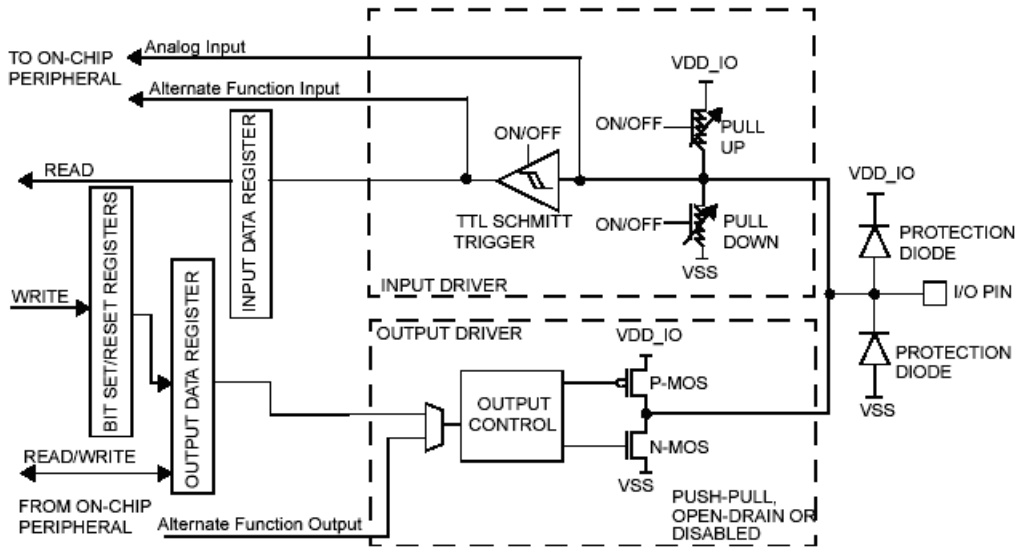

图 7-1 I/O 基本结构图

I/O 口除了通用的输入、输出外，还有很多其它可选复用功能，如外部中断，如图 7-2 所示，可以配置 A、B、C、D、E 口第 0 个引脚连接 0 号外部中断线。在复位后，将默认不激活可选的复用功能，I/O 口被配置成浮空输入模式。当端口配置为输入时，在每个 APB2 时钟周期会将捕获的端口

数据送到输入寄存器（GPIOx_IDR） (x=A..G)，在输入模式下，输出是断开的；当配置成输出模式时，写到输出寄存器（GPIOx_ODR） (x=A..G)的值被传给端口对应的 I/O 引脚，在输出模式下，输入是允许的。

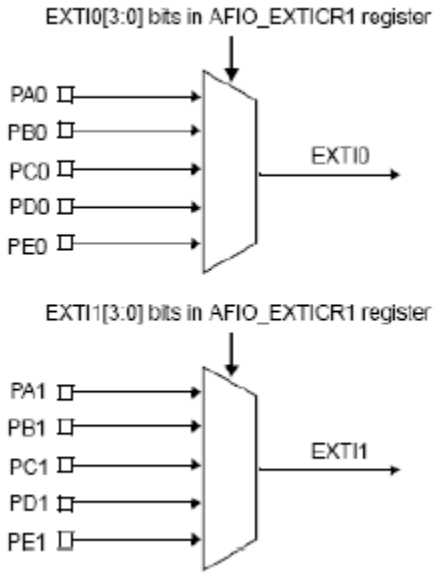

图 7-2 I/O 外部中断

7.1.2 相关功能寄存器

■ **端口配置寄存器（低位）（GPIOA_CRL- GPIOG_CRL）**

在 STM32F10x 处理器中大多数的管脚都可以复用，因此必须对每个管脚进行配置。低位配置寄存器用来设置端口低 8 位的工作模式。端口配置寄存器（低位）如图 7-3 所示，各位的含义如表 7-1 所列。该寄存器的地址偏移量为 00h（所有 GPIO 控制寄存器的地址都从 0x4001 0800 处开始，可参考表 3-1），复位值为 4444 4444h。

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
CNF7[1:0]		MODE7[1:0]		CNF6[1:0]		MODE6[1:0]		CNF5[1:0]		MODE5[1:0]		CNF4[1:0]		MODE4[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CNF3[1:0]		MODE3[1:0]		CNF2[1:0]		MODE2[1:0]		CNF1[1:0]		MODE1[1:0]		CNF0[1:0]		MODE0[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

图 7-3 端口配置寄存器（低位）

其中 MODE0[1:0]－MODE7[1:0]用于配置输入或输出，CNF0[1:0]－CNF7[1:0]用于配置输入或输出的具体模式。这些寄存器只能以字的形式进行读写。在图 7-3 中各位域下的标志分别表示

该位域的读写特性，其中：r 表示只读；rw 表示可读写；w 表示只写，读将获得复位值；rc 表示可读和可以清除该位；rc_w1 表示可读，写 1 可以清除该位，写 0 无效；rc_w0 表示可读，写 0 可以清除该位，写 1 无效；rs 表示可读可设置该位，写 0 无效；t 表示写 1 使得该位翻转，写 0 无效。本书所有寄存器的介绍均采用以上标志方法。

表 7-1 端口配置寄存器（低位）的定义

位 31:30、27:26、23:22、19:18、15:14、11:10、7:6、3:2	CNFx[1:0]：配置第x位(x = 0…7)，软件通过这些位来配置I/O端口相应的位。 在输入模式(MODE[1:0]=00)： 00：模拟输入模式； 01：浮空输入模式(复位后的状态)； 10：上拉/下拉输入模式； 11：保留。 在输出模式(MODE[1:0]>00)： 00：通用推挽输出模式； 01：通用开漏输出模式； 10：复用功能（推挽）输出模式； 11：复用功能（开漏）输出模式。
位9:28、25:24、21:20、17:16、13:12、9:8、5:4、1:0	MODEx[1:0]：配置第x位(x = 0…7)，软件通过这些位来配置I/O端口相应的位。 00：输入模式(复位后的状态)； 01：输出模式，最大速度10MHz； 10：输出模式，最大速度2MHz； 11：输出模式，最大速度50MHz。

■ **端口配置寄存器（高位）（GPIOA_CRH- GPIOG_CRH）**

高位配置寄存器用来设置端口高 8 位的工作模式。端口配置寄存器（高位）如图 7-4 所示，各位的含义如表 7-2 所列。该寄存器的地址偏移量为 04h，复位值为 4444 4444h。这些寄存器只能以字的形式进行读写。

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
CNF15[1:0]		MODE15[1:0]		CNF14[1:0]		MODE14[1:0]		CNF13[1:0]		MODE13[1:0]		CNF12[1:0]		MODE12[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CNF11[1:0]		MODE11[1:0]		CNF10[1:0]		MODE10[1:0]		CNF9[1:0]		MODE9[1:0]		CNF8[1:0]		MODE8[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

图 7-4 端口配置寄存器（高位）

表 7-2 端口配置寄存器（高位）的定义

位 31:30、27:26、23:22、19:18、15:14、11:10、7:6、3:2	CNFx[1:0]：配置第x位(x = 8…15)，软件通过这些位来配置I/O端口相应的位。 在输入模式(MODE[1:0]=00)： 00：模拟输入模式； 01：浮空输入模式(复位后的状态)； 10：上拉/下拉输入模式； 11：保留。 在输出模式(MODE[1:0]>00)： 00：通用推挽输出模式； 01：通用开漏输出模式；

	10：复用功能（推挽）输出模式； 11：复用功能（开漏）输出模式。
位9:28、25:24、21:20、17:16、13:12、9:8、5:4、1:0	MODEx[1:0]：配置第x位(x = 8…15)，软件通过这些位来配置I/O端口相应的位。 00：输入模式(复位后的状态)； 01：输出模式，最大速度10MHz； 10：输出模式，最大速度2MHz； 11：输出模式，最大速度50MHz。

■ **端口数据输入寄存器（GPIOA_IDR- GPIOG_IDR）**

如果端口被配置成了输入端口，可以从 GPIOx_IDR (x=A..G)相应位读数据，如图 7-5 所示，其低 16 位分别对应每个引脚，高 16 位保留。该寄存器地址偏移为 08h，复位值为 00000000h。这些寄存器只能以字的形式进行读取。

图 7-5 端口数据输入寄存器

■ **端口数据输出寄存器（GPIOA_ODR- GPIOG_ODR）**

如果端口被配置成了输出端口，可以向 GPIOx_ODR (x=A..G)相应位读或写数据。如图 7-6 所示，其低 16 位分别对应每个引脚，高 16 位保留。该寄存器的地址偏移为 0Ch，复位值为 0000 0000h。这些寄存器只能以字的形式进行读写。

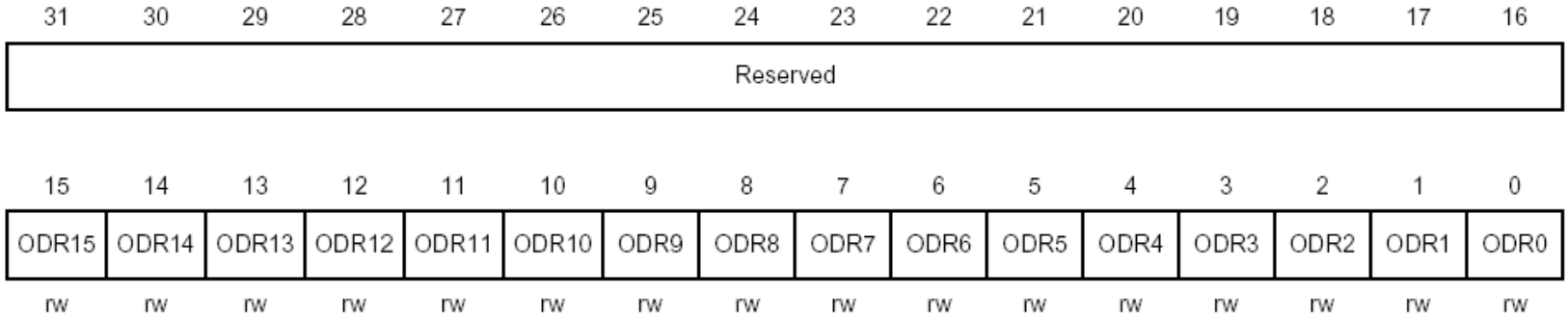

图 7-6 端口数据输入寄存器

■ **端口置位复位寄存器（GPIOA_BSRR- GPIOG_ BSRR）**

通过置位复位寄存器，可以对端口数据输出寄存器（GPIOA_ODR- GPIOG_ODR）每一位进行置 1 和复位操作，如图 7-7 所示，寄存器各位的定义见表

7-3。该寄存器的地址偏移为 10h，复位值为 0000 0000h。

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
BR15	BR14	BR13	BR12	BR11	BR10	BR9	BR8	BR7	BR6	BR5	BR4	BR3	BR2	BR1	BR0
w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BS15	BS14	BS13	BS12	BS11	BS10	BS9	BS8	BS7	BS6	BS5	BS4	BS3	BS2	BS1	BS0
w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w

图 7-7 端口置位复位寄存器

表 7-3 端口复位置位寄存器的定义

位31:16	**BRx**: 清除位x (x = 0…15) 这些位只能写入并只能以字的形式操作。 0：对对应的ODRx位不产生影响 1：清除对应的ODRx位为0 注：如果同时设置了BSx和BRx的对应位，BSx位起作用。
位15:0	**BSx**: 设置位x (x = 0…15) 这些位只能写入并只能以字的形式操作。 0：对对应的ODRx位不产生影响 1：设置对应的ODRx位为1

■ **端口复位寄存器（GPIOA_BRR- GPIOG_ BRR）**

通过复位寄存器，可以对端口数据输出寄存器（GPIOA_ODR- GPIOG_ODR）每一位进行复位操作，如图 7-8 所示，寄存器各位的定义见表 7-4。该寄存器的地址偏移为 14h，复位值为 0000 0000h。

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BR15	BR14	BR13	BR12	BR11	BR10	BR9	BR8	BR7	BR6	BR5	BR4	BR3	BR2	BR1	BR0
w	w	w	w	w	w	w	w	w	w	w	w	w	w	w	w

图 7-8 端口复位寄存器

表 7-4 端口复位寄存器的定义

位31:16	保留。
位15:0	**BRx**: 清除位x (x = 0…15) 这些位只能写入并只能以字的形式操作。 0：对对应的ODRx位不产生影响

	1：清除对应的ODRx位为0 注：如果同时设置了BSx和BRx的对应位，BSx位起作用。

■ **端口配置锁定寄存器（GPIOA_LCKR- GPIOG_ LCKR）**

端口配置锁定寄存器的每一位用来锁定控制寄存器(CRL, CRH)相应的 4 位。端口配置锁定寄存器如图 7-9 所示，寄存器定义见表 7-5，当执行正确的写序列设置了位 LCKK 时，该寄存器用来锁定端口位的配置。位[15:0]用于锁定 GPIO 端口的配置。在规定的写入操作期间，不能改变 LCKR[15:0]。当对相应的端口位执行了 LOCK 序列后，在下次系统复位之前将不能再更改端口位的配置，直到下次复位后解锁。该寄存器的地址偏移为 18h，复位值为 0000 0000h。

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															LCKK
															rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
LCK15	LCK14	LCK13	LCK12	LCK11	LCK10	LCK9	LCK8	LCK7	LCK6	LCK5	LCK4	LCK3	LCK2	LCK1	LCK0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

图 7-9 端口配置锁定寄存器

表 7-5 端口配置锁定寄存器的定义

位31:17	保留
位16	**LCKK**，LCKx(x=1~15)的锁键，该位可随时读出，它只可通过锁键写入序列修改。 0：端口配置锁键位激活 1：端口配置锁键位被激活，下次系统复位前GPIOx_LCKR寄存器被锁住。 锁键的写入序列：写1 -> 写0 -> 写1 -> 读0 -> 读1。 最后一个读可省略，但可以用来确认锁键已被激活。 注：在操作锁键的写入序列时，不能改变LCK[15:0]的值。操作锁键写入序列中的任何错误将不能激活锁键。
位15:0	**LCKx**: 锁位x (x = 0…15) 这些位可读可写，但只能在LCKK位为0时写入。 0：不锁定端口的配置 1：锁定端口的配置

7.1.3 应用实例

7.1.3.1 设计要求

开发板上有四个发光二极管 LED1、LED2、LED3、LED4，使它们有规律的点亮，具体顺序如下：LED1 亮->LED2 亮->LED3 亮->LED4 亮，如此反复，当按下开发板上的 Key 按钮时，灯全部熄灭，2 秒钟后发光二极管全部点亮，再过一秒钟后返回被中断前的状态，有规律的依次点亮。

7.1.3.2 硬件电路设计

在评估板上，LED1、LED2、LED3、LED4 分别与 PC6、PC7、PC8、PC9 相连，按键 Key 与 PB9 相连，如图 7-10 所示。

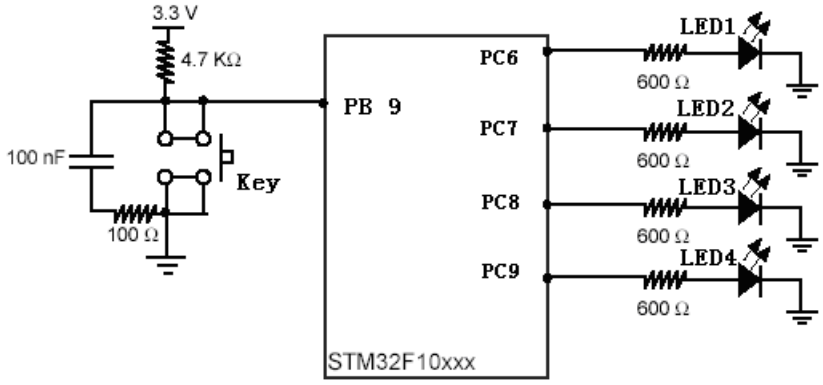

图 7-10 GPIO 接口应用实例电路图

7.1.3.3 软件程序设计

根据任务要求，程序内容主要包括：

(1) 配置 PB 口第 9 个引脚作为外部中断，下降延触发；

(2) 读取端口数据输出寄存器 GPIOC_ODR[15:0]的值，因为 C 口[9:6]位和四个 LED 灯连通。

(3) 初始化时，LED 依次点亮，当按下 Key 键时产生中断；

(4) 中断服务子程序内容：灯全部熄灭，2 秒钟后发光二极管全部点亮，再过 1 秒钟后中断返回。

整个工程包含 3 个源文件：STM32F10x.s、stm32f10x_it.c 和 main.c，其中 STM32F10x.s 为启动代码，所有中断服务子程序均在 stm32f10x_it.c 中，其它函数则在 main.c 中。下面分别介绍相关的函数，具体程序清单见参考程序。

GPIO_Configuration 函数负责配置 GPIO 端口，其中 GPIO_InitStructure 数据结构包含所有 GPIO 端口配置所需各项，函数 GPIO_Init 则实现某个端口的配置。在 STM32F10xR.LIB 和 STM3210xD.LIB 中均已定义，读者可通过查阅 5.3 节所述的 ST 公司发布的《ARM®-based 32-bit MCU STM32F101xx and STM32F103xx Firmware Library》手册来了解该数据结构以及相关驱动函数。

NVIC_Configuration 函数用于配置嵌入式中断控制器，相关内容将在后面章节介绍，此处需要将 PB9 引脚配置为外部中断。

EXTI9_5_IRQHandler 函数是 PB9 的中断服务子程序，当按下 Key 按钮之后触发 EXTI9_5 中断，在中断服务子程序中先将灯全部熄灭，延迟 2 秒后全部点亮，再延迟 1 秒之后退出中断服务子程序。

SysTick_Configuration 函数用于配置和允许系统时钟中断，系统时钟中断服务子程序 SysTickHandler 函数则用于产生 1 毫秒的延时，这样 Delay 函数就可以通过开关系统时钟计数器来实现精确延时了。

系统时钟配置函数 RCC_Configuration 等基本配置函数则不再冗述了。

参考程序：

- main.c

```
/*******************************************************
* File Name          : main.c
* Author             :Wuhan R&D Center, Embest
* Date First Issued    : 02/05/2007
* 描述               : Main program body
```

```
*******************************************************************************/

/* Includes ------------------------------------------------------------------*/
#include "stm32f10x_lib.h"
#define ADC1_DR_Address    ((u32)0x4001244C)

unsigned short int ADC_ConvertedValue;
GPIO_InitTypeDef GPIO_InitStructure;
ADC_InitTypeDef   ADC_InitStructure;
DMA_InitTypeDef   DMA_InitStructure;
EXTI_InitTypeDef EXTI_InitStructure;
ErrorStatus HSEStartUpStatus;
extern vu32 TimingDelay;
/* Private function prototypes -----------------------------------------------*/
void RCC_Configuration(void);
void NVIC_Configuration(void);
void GPIO_Configuration(void);
void Delay(vu32 nTime);
void SysTick_Configuration(void);
void SetupLED (void) ;
extern void SetupADC    (void);

/* Private functions ---------------------------------------------------------*/

/*******************************************************************************
* 函数名      : main
* 描述        : 主程序
* 输入        :无
* 输出        :无
* 返回        :无
*******************************************************************************/
int main(void)
{
#ifdef DEBUG
  debug();
#endif

  /* 配置系统时钟 */
  RCC_Configuration();
  SysTick_Configuration();

  /* 配置 NVIC */
  NVIC_Configuration();

  /* 配置 GPIO 端口 */
  GPIO_Configuration();

  /* 将 PB.9 配置为 EXTI Line9 */
  GPIO_EXTILineConfig(GPIO_PortSourceGPIOB, GPIO_PinSource9);

  /* EXTI Line9 下降沿产生中断 */
  EXTI_InitStructure.EXTI_Line = EXTI_Line9;
  EXTI_InitStructure.EXTI_Mode = EXTI_Mode_Interrupt;
  EXTI_InitStructure.EXTI_Trigger = EXTI_Trigger_Falling;
```

```
  EXTI_InitStructure.EXTI_LineCmd = ENABLE;
  EXTI_Init(&EXTI_InitStructure);

  /* 通过向 PC 口输出，循环点亮 LED1—LED4 */
   for(;;)
   {
    GPIOC->ODR = 0xfffffc4f;
    Delay(800);
    GPIOC->ODR = 0xfffffc8f;
    Delay(800);
    GPIOC->ODR = 0xfffffd0f;
    Delay(800);
    GPIOC->ODR = 0xfffffe0f;
    Delay(800);
    }

}

/*******************************************************************************
* 函数名  ：RCC_Configuration
* 描述    ：配置不同的系统时钟
* 输入    ：无
* 输出    ：无
* 返回：  ：无
*******************************************************************************/
void RCC_Configuration(void)
{
  /* RCC system reset(for debug purpose) */
  RCC_DeInit();

  /* Enable HSE */
  RCC_HSEConfig(RCC_HSE_ON);

  /* Wait till HSE is ready */
  HSEStartUpStatus = RCC_WaitForHSEStartUp();

  if(HSEStartUpStatus == SUCCESS)
  {
    /* HCLK = SYSCLK */
    RCC_HCLKConfig(RCC_SYSCLK_Div1);

    /* PCLK2 = HCLK */
    RCC_PCLK2Config(RCC_HCLK_Div1);

    /* PCLK1 = HCLK/2 */
    RCC_PCLK1Config(RCC_HCLK_Div2);

    /* Flash 2 wait state */
    FLASH_SetLatency(FLASH_Latency_2);
    /* Enable Prefetch Buffer */
    FLASH_PrefetchBufferCmd(FLASH_PrefetchBuffer_Enable);

    /* PLLCLK = 8MHz * 9 = 72 MHz */
    RCC_PLLConfig(RCC_PLLSource_HSE_Div1, RCC_PLLMul_9);
```

```
    /* 允许 PLL */
    RCC_PLLCmd(ENABLE);

    /* 等待 PLL 有效 */
    while(RCC_GetFlagStatus(RCC_FLAG_PLLRDY) == RESET)
    {
    }

    /* 以 PLL 作为系统时钟源 */
    RCC_SYSCLKConfig(RCC_SYSCLKSource_PLLCLK);

    /* 等待 PLL 成为系统时钟源有效 */
    while(RCC_GetSYSCLKSource() != 0x08)
    {
    }
  }

  /* Enable GPIOB, GPIOC and AFIO clocks */
  RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOB | RCC_APB2Periph_GPIOC |
                         RCC_APB2Periph_AFIO, ENABLE);
}

/*******************************************************************
* 函数名  : NVIC_Configuration
* 描述    : 配置中断系统.
* 输入    : 无
* 输出    : 无
* 返回:   : 无
*******************************************************************/
void NVIC_Configuration(void)
{
 NVIC_InitTypeDef NVIC_InitStructure;

#ifdef  VECT_TAB_RAM
  /* Set the Vector Table base location at 0x20000000 */
  NVIC_SetVectorTable(NVIC_VectTab_RAM, 0x0);
#else  /* VECT_TAB_FLASH  */
  /* Set the Vector Table base location at 0x08000000 */
  NVIC_SetVectorTable(NVIC_VectTab_FLASH, 0x0);
#endif

  /* Configure one bit for preemption priority */
  NVIC_PriorityGroupConfig(NVIC_PriorityGroup_1);

  /* 允许 EXTI9_5 中断 */
  NVIC_InitStructure.NVIC_IRQChannel = EXTI9_5_IRQChannel;
  NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;
  NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;
  NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
  NVIC_Init(&NVIC_InitStructure);
}

/*******************************************************************
```

```
* 函数名  ：SysTick_Configuration
* 描述    ：配置系统时钟，每 1 毫秒产生 1 次中断.
* 输入    ：无
* 输出    ：无
* 返回：  ：无
*******************************************************************************/
void SysTick_Configuration(void)
{
  /*选择 AHB 时钟(HCLK)作为 SysTick 时钟源 */
  SysTick_CLKSourceConfig(SysTick_CLKSource_HCLK);

  /*将 SysTick 的优先级设置为 3 */
  NVIC_SystemHandlerPriorityConfig(SystemHandler_SysTick, 3, 0);

  /* 每 1ms 发生一次 SysTick 中断（对应 HCLK 运行于 72MHz） */
  SysTick_SetReload(72000);

  /* 允许 SysTick 中断 */
  SysTick_ITConfig(ENABLE);
}

/*******************************************************************************
* 函数名  ：GPIO_Configuration
* 描述    ：配置 GPIO 端口
* 输入    ：无
* 输出    ：无
* 返回：  ：无
*******************************************************************************/
void GPIO_Configuration(void)
{
  GPIO_InitTypeDef GPIO_InitStructure;

  /* 配置 PC 口为输出（推挽），速度 50MHz*/
  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_6|GPIO_Pin_7|GPIO_Pin_8|GPIO_Pin_9;
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
  GPIO_Init(GPIOC, &GPIO_InitStructure);

  /* 配置 PB9 为输入 (EXTI Line9) */
  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_9;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN_FLOATING;
  GPIO_Init(GPIOB, &GPIO_InitStructure);
}

/*******************************************************************************
* 函数名  ：Delay
* 描述    ：插入延迟时间
* 输入    ：nTime，延迟时间，以毫秒为单位
* 输出    :无
* 返回    :无
*******************************************************************************/
void Delay(u32 nTime)
{
```

```
  /* 允许 SysTick 计数器 */
  SysTick_CounterCmd(SysTick_Counter_Enable);

  TimingDelay = nTime;

  while(TimingDelay != 0);

  /* 禁止 SysTick 计数器*/
  SysTick_CounterCmd(SysTick_Counter_Disable);
  /* SysTick 计数器清零 */
  SysTick_CounterCmd(SysTick_Counter_Clear);
}

#ifdef  DEBUG
/*******************************************************************************
* 函数名  : assert_failed
* 描述    : 当程序出错时，返回出错的文件名及在源程序中的行号
* 输入          : - file:指向文件名的指针
*           - line:在源程序中的行号
* 输出    : 无
* 返回:   : 无
*******************************************************************************/
void assert_failed(u8* file, u32 line)
{
  /* 用户可以在此根据需要添加内容，例如报告出错的文件名和行号,
     ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) */

  /* 无限循环 */
  while (1)
  {
  }
}
#endif/
```

- stm32f10x_it.c

```
#include "stm32f10x_it.h"
vu32 TimingDelay = 0;
extern  void Delay(vu32 nTime);

/*******************************************************************************
* 函数名  : EXTI9_5_IRQHandler( )
* 说明    : 向端口数据输出寄存器 GPIOC_ODR 写数据。
* 输入    : 无
* 输出    : 无
* 返回:   : 无
*******************************************************************************/
void EXTI9_5_IRQHandler(void){
 int i;
 if(EXTI_GetITStatus(EXTI_Line9) != RESET)
  {
       //发光二极管全部熄灭
       for(i=0;i<=8000000;i++)
      GPIOC->ODR = 0xfffffC3f;
       //延时
```

```
        for(i=0;i<=2000000;i++)
       //发光二极管全部点亮
        GPIOC->ODR = 0xffffffff;
        //延时
        for(i=0;i<=1000000;i++)
    //清除 9 号外部中断屏蔽位
    EXTI_ClearITPendingBit(EXTI_Line9);
  }
}

/*******************************************************************
* 函数名  : SysTickHandler
* 描述    : 处理 SysTick 中断
* 输入    : 无
* 输出    : 无
* 返回:   : 无
*******************************************************************/
void SysTickHandler(void)
{
        TimingDelay--;
}
```

7.1.3.4 运行过程

(1) 使用 Keil uVision3 通过 ULINK 2 仿真器连接实验板，或使用两根 USB 电缆分别将 CN1 和 CN5 与 PC 机 USB 端口相连（使用板上自带的 UlinkMe）；打开实验例程目录 GPIO_TEST 子目录下的 GPIO.Uv2 例程，编译链接工程；

(2) 选择软件调试模式，点击 MDK 的 Debug 菜单，选择 Start/Stop Debug Session 项或 Ctrl+F5 键，在逻辑分析仪中添加 GPIOC_ODR.6、GPIOC_ODR.7、GPIOC_ODR.8、GPIOC_ODR.9，点击 Run 按钮即可在逻辑分析仪中看到如图 7-11；

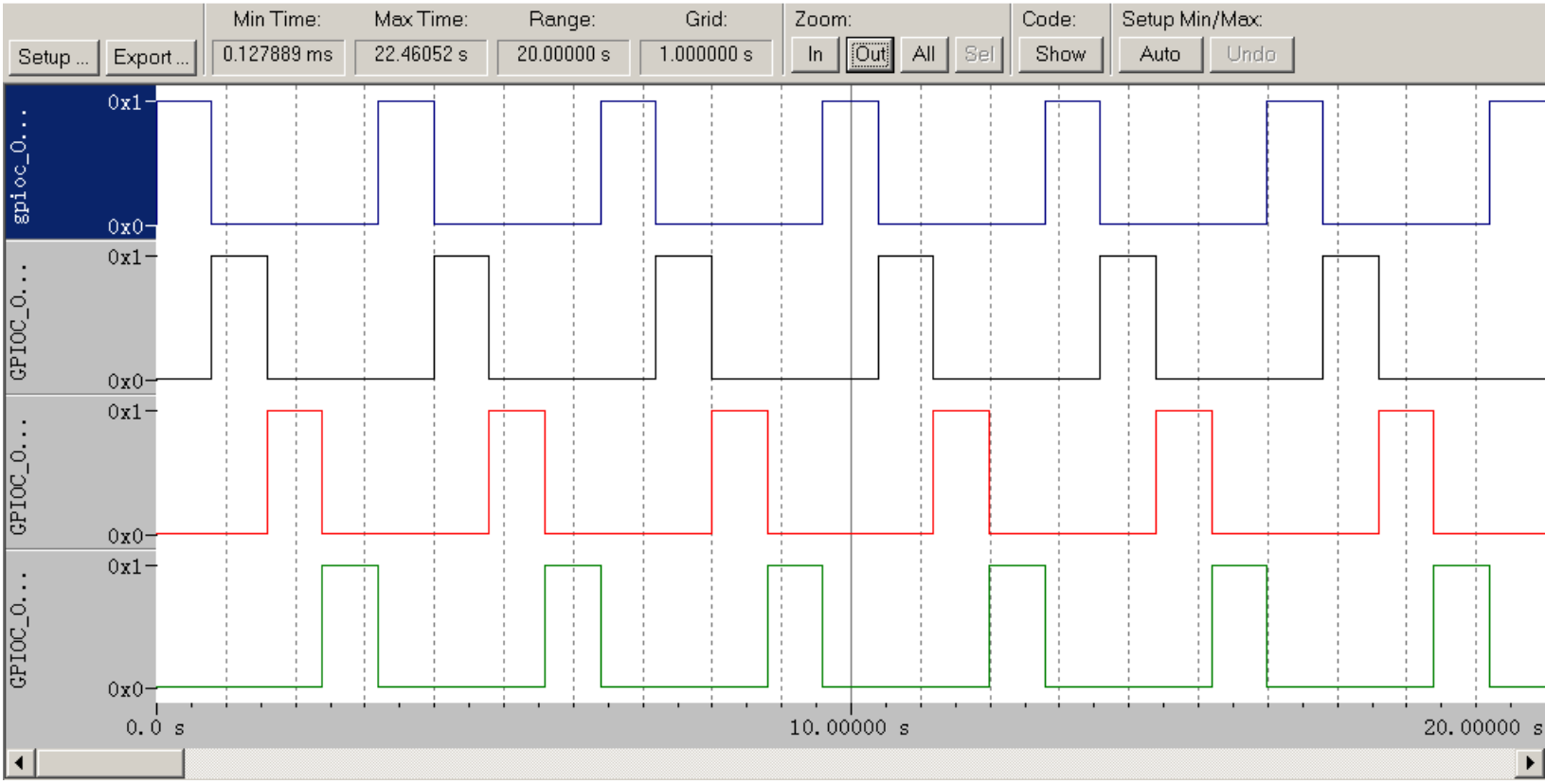

图7-11 GPIO接口应用实例软件仿真结果

(3) 选择硬件调试模式，选择 Start/Stop Debug Session 项或 Ctrl+F5 键，下载程序并运行，观察 LED 灯的变化情况；当程序运行在 while 循环体内时，按 Key 键，程序进入中断服务子程序 EXTI9_5_IRQHandler()，单步运行，观察 LED 的变化情况。

(4) 退出 Debug 模式，打开 Flash 菜单>Download，将程序下载到开发板的 Flash 中，按 RESET 键复位，观察 LED 灯的情况，正常情况应如表 7-6 所列。

表 7-6 LED 灯状态

LED 灯状态				说明
LED1	LED2	LED3	LED4	
亮	灭	灭	灭	程序正常运行发光二极管依次点亮
灭	亮	灭	灭	
灭	灭	亮	灭	
灭	灭	灭	亮	
灭	灭	灭	灭	外部信号输入