第3章 STM32F1系列处理器

3.1 STM32F1 系列处理器简介

STM32 系列处理器是意法半导体(ST)公司推出的基于 Cortex-M3 内核的 32 位 MCU。目前这个系列包含多个子系列,分别是 STM32 小容量产品、STM32 中容量产品、STM32 大容量产品和 STM32 互联型产品。在众多 STM32 系列产品中,STM32F1 系列基础型处理器无疑占据了极其重要的地位。该系列产品功能强大,满足了工业、医疗和消费类市场的各种应用需求。凭借该系列的产品,意法半导体公司在全球 ARM Cortex-M 微控制器领域处于领先地位。

STM32F1 系列处理器包含 5 个产品线,分别为 STM32F100-24MHz、STM32F101-36MHz、STM32F102-48MHz、STM32F103-72MHz 和 STM32F105/107-72MHz。它们的引脚、外设和软件均兼容。STM32 系列产品命名规则如图 3.1 所示。

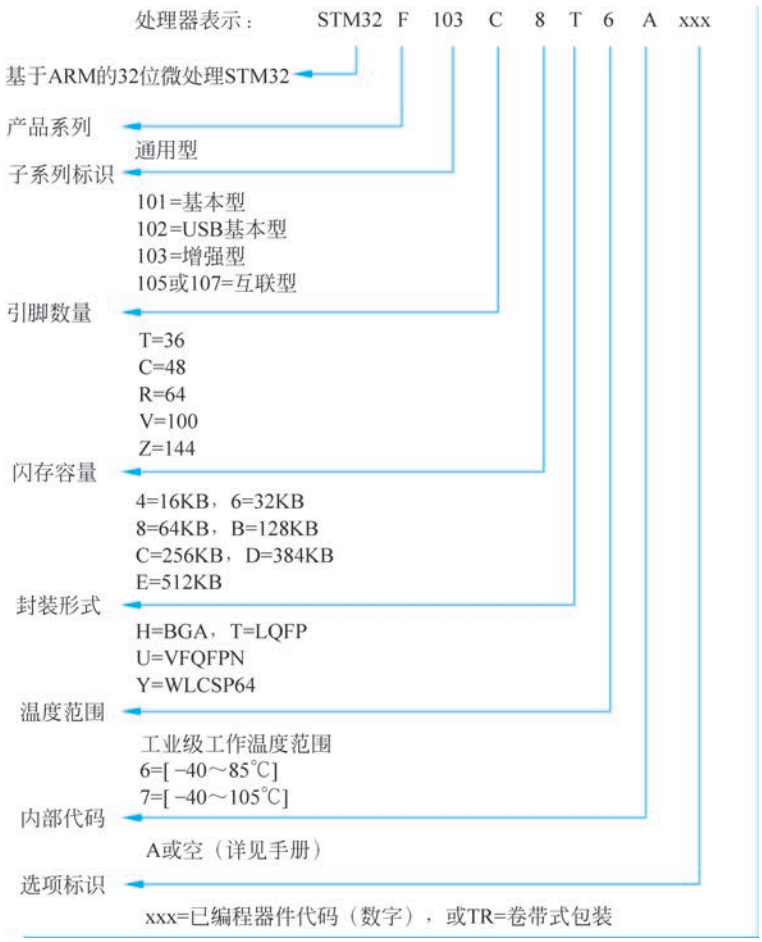

图 3.1 STM32 系列处理器产品命名规则

3.2 STM32F103ZET6 处理器架构和主要特性

3.2.1 芯片和引脚定义

STM32F103ZET6 处理器采用 LQFP144 封装，其引脚图如图 3.2 所示。从封装类别和引脚图均可以看出，STM32F103ZET6 芯片总共引出了 144 个引脚，就引脚数量而言比 51 系列的单片机复杂很多。

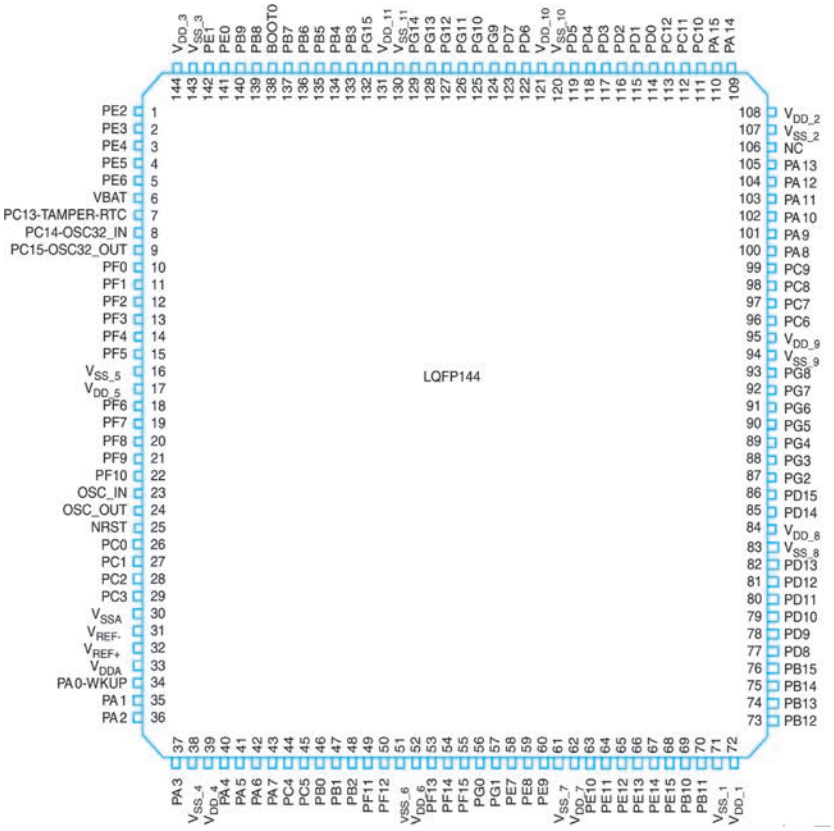

图 3.2 STM32F103ZET6 的引脚图

STM32F103ZET6 的各个引脚都有其特定的功能及特性。由于其引脚数量较多，故在此仅列出其中编号为 1～50 的引脚定义，具体如表 3.1 所示。如果需要使用其他的引脚或者查找详细的引脚信息，请自行查阅意法半导体公司的官方手册和说明书。

表 3.1 STM32F103ZET6 的部分主要引脚定义

引脚编号	引脚名称	类型	I/O 电平	复位后的主要功能	复用功能	
					默认情况	重映射后
1	PE2	I/O	FT	PE2	TRACECK/FSMC_A23	—
2	PE3	I/O	FT	PE3	TRACED0/FSMC_A19	—

续表

引脚编号	引脚名称	类型	I/O电平	复位后的主要功能	复用功能	
					默认情况	重映射后
3	PE4	I/O	FT	PE4	TRACED1/FSMC_A20	—
4	PE5	I/O	FT	PE5	TRACED2/FSMC_A21	—
5	PE6	I/O	FT	PE6	TRACED3/FSMC_A22	—
6	VBAT	S	—	VBAT	—	—
7	PC13-TAMPER-RTC	I/O	—	PC13	TAMPER-RTC	—
8	PC14-OSC32_IN	I/O	—	PC14	OSC32_IN	—
9	PC15-OSC32_OUT	I/O	—	PC15	OSC32_OUT	—
10	PF0	I/O	FT	PF0	FDMC_A0	—
11	PF1	I/O	FT	PF1	FDMC_A1	—
12	PF2	I/O	FT	PF2	FDMC_A2	—
13	PF3	I/O	FT	PF3	FDMC_A3	—
14	PF4	I/O	FT	PF4	FDMC_A4	—
15	PF5	I/O	FT	PF5	FDMC_A5	—
16	VSS_5	S	—	VSS_5	—	—
17	VDD_5	S	—	VDD_5	—	—
18	PF6	I/O	—	PF6	ADC3_IN4/FSMC_NIORD	—
19	PF7	I/O	—	PF7	ADC3_IN5/FSMC_NREG	—
20	PF8	I/O	—	PF8	ADC3_IN6/FSMC_NIOWD	—
21	PF9	I/O	—	PF9	ADC3_IN7/FSMC_CD	—
22	PF10	I/O	—	PF10	ADC3_IN8/FSMC_INTR	—
23	OSC_IN	I	—	OSC_IN	—	—
24	OSC_OUT	I	—	OSC_OUT	—	—
25	NRST	I/O	—	NRST	—	—
26	PC0	I/O	—	PC0	ADC123_INI0	—
27	PC1	I/O	—	PC1	ADC123_INI1	—
28	PC2	I/O	—	PC2	ADC123_INI2	—
29	PC3	I/O	—	PC3	ADC123_INI3	—
30	VSSA	S	—	VSSA	—	—
31	VREF−	S	—	VREF−	—	—
32	VREF+	S	—	VREF+	—	—
33	VDDA	S	—	VDDA	—	—
34	PA0-WKUP	I/O	—	PA0	WKUP/USART2_CTS ADC123_IN0 TIM2_CH1_ETR TIM5_CH1/TIM8_ETR	—

续表

引脚编号	引脚名称	类型	I/O电平	复位后的主要功能	复用功能	
					默认情况	重映射后
35	PA1	I/O	—	PA1	USART2_RTS ADC123_IN1 TIM5_CH2/TIM2_CH2	—
36	PA2	I/O	—	PA2	USART2_TX/TIM5_CH3 ADC123_IN2/TIM2_CH3	—
37	PA3	I/O	—	PA3	USART2_RX/TIM5_CH4 ADC123_IN3/TIM2_CH4	—
38	VSS_4	S	—	VSS_4	—	—
39	VDD_4	S	—	VDD_4	—	—
40	PA4	I/O	—	PA4	SPI1_NSS USART2_CK DAC_OUT1/ADC12_IN4	—
41	PA5	I/O	—	PA5	SPI1_SCK DAC_OUT2/ADC12_IN5	—
42	PA6	I/O	—	PA6	SPI1_MISO TIM8_BKIN/ADC12_IN6 TIM3_CH1	TIM1_BKIN
43	PA7	I/O	—	PA7	SPI1_MOSI TIM8_CH1N/ADC12_IN7 TIM3_CH2	TIM1_CH1N
44	PC4	I/O	—	PC4	ADC12_IN14	—
45	PC5	I/O	—	PC5	ADC12_IN15	—
46	PB0	I/O	—	PB0	ADC12_IN8/TIM3_CH3 TIM8_CH2N	TIM1_CH2N
47	PB1	I/O	—	PB1	ADC12_IN9/TIM3_CH4 TIM8_CH3N	TIM1_CH3N
48	PB2	I/O	FT	PB2/Boot1	—	—
49	PF11	I/O	FT	PF11	FSMC_NIOS16	—
50	PF12	I/O	FT	PF12	FSMC_A6	—

注意：(1) I=输入(Input)；O=输出(Output)，S=电源(Supply)。

(2) FT=可容忍5V电压。

如表3.1所示，处理器在复位后通常保留引脚的默认功能，即大部分都是作为普通的输入/输出引脚使用。芯片提供的强大的引脚复用功能是其灵魂所在，比如可以用作串口通信引脚、ADC采集引脚以及定时器通道引脚等。在STM32F103ZET6这款处理器芯片中，复用功能还可分为默认情况下的复用功能和重映射后的复用功能。这些复用功能的使用都需要提前对引脚的输入/输出模式以及时钟等进行正确的配置，而重映射的配置则比默认复用功能的配置多一些步骤。可能有读者了解过STC12或者STC15系

列的单片机，这些单片机的引脚在使用时也需要对输入/输出模式等进行配置，但是配置寄存器数目相对较少，就其复杂程度和功能而言远不如 STM32 系列处理器。

3.2.2 系统架构

STM32F103ZET6 的系统架构如图 3.3 所示，其主要由以下部件构成：

（1）Cortex-M3 内核 DCode 总线和系统总线。

（2）通用 DMA1 和通用 DMA2。

（3）内部 SRAM。

（4）内部闪存存储器。

（5）FSMC。

（6）AHB 到 APB 的桥（AHB2APBx），它连接所有的 APB 设备。

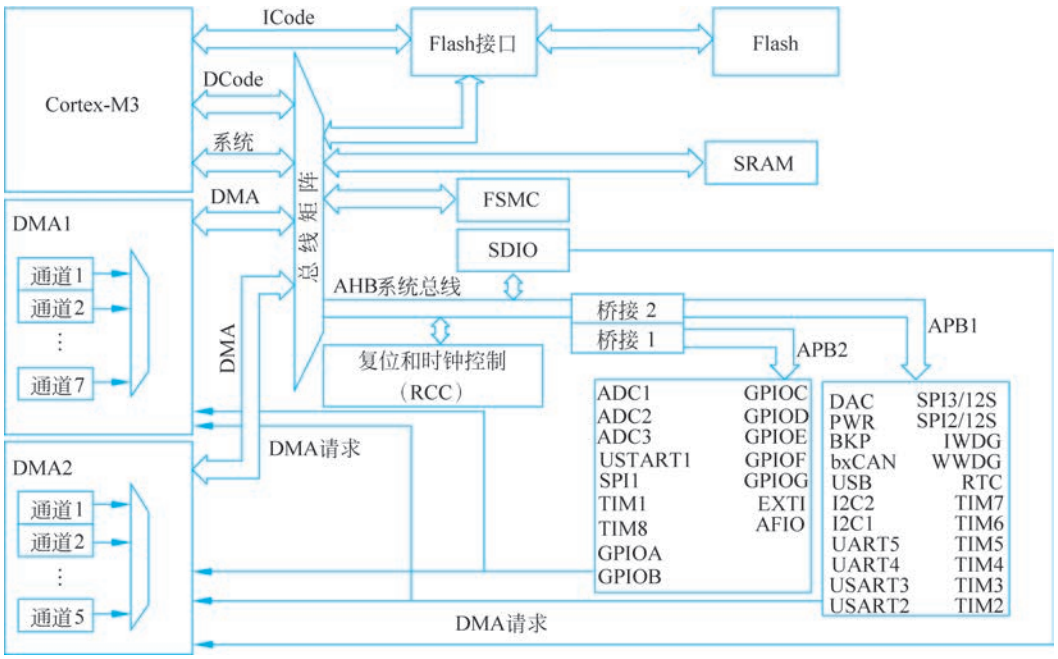

图 3.3 STM32F103ZET6 的系统架构

这些部件都是通过一个多级的 AHB 总线架构相互连接的，该总线架构的主要组成部分及其功能如表 3.2 所示。

表 3.2 AHB 总线架构的主要组成部分及其功能

名　称	功　能
ICode 总线	此总线将 Cortex-M3 内核的指令总线与闪存指令接口相连接。指令预取在此总线上完成
DCode 总线	此总线将 Cortex-M3 内核的 DCode 总线与闪存存储器的数据接口相连接（用于常量加载和调试访问）

续表

名　　称	功　　能
系统总线	此总线连接 Cortex-M3 内核的系统总线(外设总线)到总线矩阵,总线矩阵负责协调内核和 DMA 间的访问
DMA 总线	此总线将 DMA 的 AHB 主控接口与总线矩阵相连,总线矩阵负责协调 CPU 的 DCode 和 DMA 到 SRAM、闪存和外设的访问
总线矩阵	总线矩阵协调内核系统总线和 DMA 主控总线之间的访问仲裁,仲裁采用轮换算法。在互联型产品中,总线矩阵包含 5 个驱动部件(CPU 的 DCode、系统总线、以太网 DMA、DMA1 总线和 DMA2 总线)和 3 个从部件(闪存接口 FLITF、SRAM 和 AHB2APB 桥)。在其他产品中总线矩阵包含 4 个驱动部件(CPU 的 DCode、系统总线、DMA1 总线和 DMA2 总线)和 4 个被动部件(FLITF、SRAM、FSMC 和 AHB2APB 桥)
AHB/APB 桥(APB)	两个 AHB/APB 桥在 AHB 和两个 APB 总线间提供同步连接。APB1 操作速度限于 36MHz,APB2 全速操作,最高为 72MHz

3.2.3　主要特性

STM32F103ZET6 的处理性能优越,应用广泛。下面列出该处理器的主要特性。

(1) 内核。

① 基于 ARM 的 Cortex-M3 架构的 32 位处理器,最高工作频率为 72MHz。

② 能做单周期乘法和硬件除法。

(2) 存储器。

① 512KB 的闪存。

② 64KB 的 SRAM。

③ 带有 4 个片选信号的灵活的静态存储器控制器(FSMC),可支持 Compact Flash、SRAM、PSRAM、NOR 和 NAND 存储器扩展。

(3) 时钟、复位和电源管理。

① 芯片和 I/O 引脚的供电电压为 2.0～3.6V。

② 上电/断电复位(POR/PDR)、可编程电压检测器(PVD)。

③ 4～16MHz 晶体振荡器。

④ 内嵌有经过出厂调校的 8MHz 的 RC 振荡器。

⑤ 内嵌有带校准的 40kHz RC 振荡器。

⑥ 带校准功能的 32kHz RTC 振荡器。

(4) 低功耗。

① 支持睡眠、停机和待机模式。

② V_{BAT} 为 RTC 和后备寄存器供电。

(5) 拥有 3 个 12 位模数转换器(ADC)。

(6) 拥有 2 个 12 位数模转换器(DAC)。

(7) DMA 控制器。

① 12 通道 DMA 控制器。

② 支持的外设包括定时器、ADC、DAC、SDIO、I2S、SPI、I2C 和 USART。

(8) 调试模式。

① 串行单线调试(SWD)和 JTAG 接口。

② Cortex-M3 嵌入式跟踪宏单元(ETM)。

(9) 快速 I/O 接口(PA～PG)。

拥有多达 7 个快速 I/O 接口,每个接口包含 16 根 I/O 引脚,所有 I/O 引脚都可以映像到 16 个外部中断。几乎所有接口均可容忍 5V 信号。

(10) 多达 11 个定时器。

拥有 4 个 16 位通用定时器、2 个 16 位 PWM 定时器、2 个看门狗定时器、系统滴答定时器和 2 个 16 位基本定时器。

(11) 多达 13 个通信接口。

拥有 2 个 I2C 接口、5 个 USART 接口、3 个 SPI 接口、1 个 CAN 接口、1 个 USB 2.0 全速接口和 1 个 SDIO 接口。

(12) 拥有循环校验码(CRC)计算单元。

(13) 工作温度:－40～＋105℃。

由上述特性可以看出,STM32F103ZET6 处理器的内部资源非常丰富,尤其是定时器和通信接口等资源。STM32F103ZET6 芯片由于功能强大,且具有的低功耗等特点,非常适合用于医疗器械和工业控制等应用领域。

3.3 STM32F103ZET6 的时钟树

时钟结构是处理器整体结构的重要组成部分。STM32 系列处理器共有 5 个时钟源,分别为高速内部时钟(High Speed Internal,HSI)、高速外部时钟(High Speed External,HSE)、低速内部时钟(Low Speed Internal,LSI)、低速外部时钟(Low Speed External,LSE)和锁相环倍频输出(Phase Locked Loop,PLL)。由于 STM32F103ZET6 的时钟系统为树状结构,故也称为时钟树,具体如图 3.4 所示。在 STM32F103ZET6 处理器中主要的时钟及其简要说明如表 3.3 所示。由如图 3.4 所示的时钟树结构,可得出以下结论:

(1) HSI、HSE 或 PLL 可用来驱动系统时钟(SYSCLK)。

(2) LSI、LSE 作为二级时钟源。40kHz 低速内部 RC 时钟(LSI)可以用于驱动独立看门狗和通过程序选择驱动 RTC。

(3) 用户可通过多个预分额器配置 AHB、高速 APB(APB2)和低速 APB(APB1)的频率。其中,AHB 和 APB2 的最高频率是 72MHz,APB1 的最高频率是 36MHz。

(4) RCC 将 AHB 时钟(HCLK)8 分频后作为 Cortex 系统定时器(SysTick)的外部时钟。通过对 SysTick 控制与状态寄存器的设置,可选择上述时钟或 Cortex(HCLK)时钟作为 SysTick 时钟。

(5) ADC 时钟由高速 APB2 时钟经 2 分频、4 分频、6 分频或 8 分频后获得。

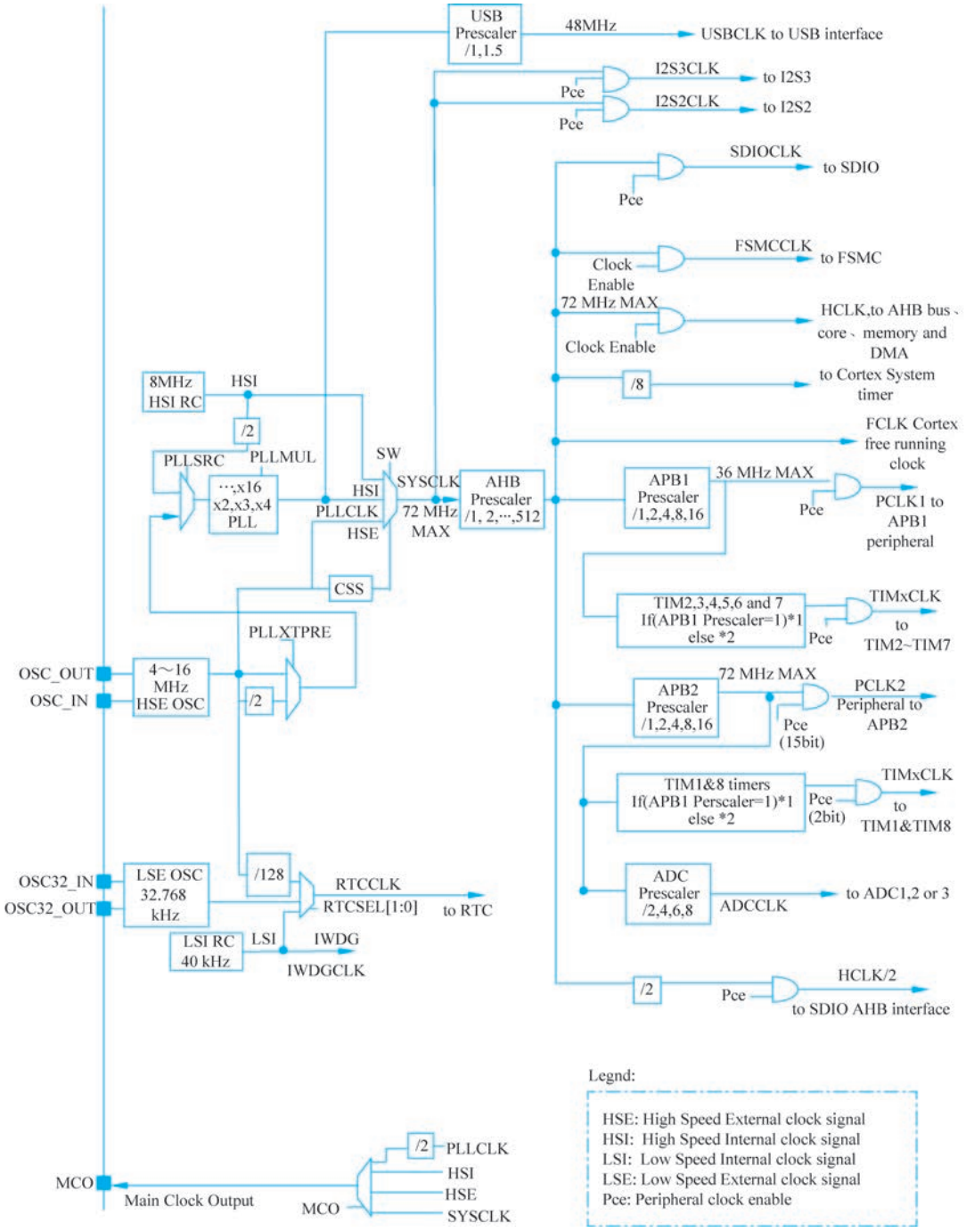

图 3.4　时钟内部结构与原理图

（6）SDIO 接口的时钟频率固定为 HCLK/2。

（7）定时器时钟频率分配由硬件分为两种情况自动设置。如果相应的 APB 预分额系数是 1，定时器的时钟频率与所在 APB 总线频率一致；否则定时器的时钟频率被设为

与其相连的 APB 总线频率的 2 倍。

（8）当某个部件不被使用时，任一个时钟源都可被独立地启动或关闭，由此可优化系统功耗。

表 3.3 STM32F103ZET6 的主要时钟及其简要说明

时钟名称	简要说明
HSE 时钟	高速外部时钟（HSE）可以由外部晶体或陶瓷谐振器产生，也可以由用户外部时钟产生。一般采用外部晶体或陶瓷谐振器产生 HSE 时钟。在 OSC_IN 和 OSC_OUT 引脚之间连接 4～16MHz 外部振荡器为系统提供精确的主时钟
HSI 时钟	HSI 时钟由内部 8MHz 的 RC 振荡器产生，可直接作为系统时钟或在 2 分频后作为 PLL 输入。HSI RC 振荡器能够在不需要任何外部器件的条件下提供系统时钟，并且它的启动时间比 HSE 晶体振荡器短。然而由于其时钟频率精度较差，HSI 时钟常作为备用时钟源
PLL 时钟	内部 PLL 可以用来倍频 HSI RC 的输出时钟或 HSE 晶体输出时钟。PLL 的设置必须在其被激活前完成。在 PLL 被激活之后，这些参数不能被改动。如果需要在应用中使用 USB 接口，那么 PLL 必须被设置为输出 48MHz 或 72MHz 时钟，用于提供 48MHz 的 USBCLK 时钟
LSE 时钟	LSE 晶体是一个 32.768kHz 的低速外部晶体或陶瓷谐振器。它为实时时钟或者其他定时功能提供一个低功耗且精确的时钟源
LSI 时钟	LSI RC 担当着低功耗时钟源的角色，它可以在停机和待机模式下保持运行，为独立看门狗和自动唤醒单元提供时钟。LSI 时钟频率约为 40kHz（在 30～60kHz）
系统时钟（SYSCLK）	系统复位后，HSI 振荡器被选为系统时钟。当时钟源被直接或通过 PLL 间接作为系统时钟时，它将不能被停止。只有当目标时钟源准备就绪了（经过启动稳定阶段的延迟或 PLL 稳定），从一个时钟源到另一个时钟源的切换才会发生。在被选择时钟源没有就绪时，系统时钟的切换不会发生
RTC 时钟	通过设置备份域控制寄存器（RCC_BDCR）中的 RTCSEL[1:0]位，RTCCLK 时钟源可由 HSE/128、LSE 或 LSI 时钟提供
看门狗时钟	如果独立看门狗已经由硬件选项或软件启动，LSI 振荡器将被强制在打开状态，并且不能被关闭。在 LSI 振荡器稳定后，时钟供应给 IWDG

由图 3.4 和表 3.3 可看出，STM32F103ZET6 处理器具有多个时钟频率，分别供给内核和不同外设模块使用。其中高速时钟用于中央处理器等高速设备，低速时钟用于外设等低速设备。STM32 处理器因为低功耗的需要，各个模块可分别独立开启时钟。因此，当需要使用某个外设模块时，就要先使能该模块对应的时钟。这一点也需要格外注意，在实际开发中很容易因时钟设置的疏忽而导致外设不能工作。

3.4 STM32F103ZET6 存储器组织及其映像

程序存储器、数据存储器、寄存器和输入/输出接口被组织在同一个 4GB 的线性地址空间范围内。可访问的存储器空间被分成 8 个主要块，每个块大小为 512MB，具体如表 3.4 所示。其他没有分配给片上存储器和外设的存储器空间都是保留的地址空间。

同时数据字节以小端格式存放在存储器中。一个字中的最低地址字节是该字的最低有效字节，而最高地址字节是最高有效字节。

表 3.4　4G 存储器空间划分

序　　号	用　　途	地 址 范 围
Block0	Flash	0x0000 0000～0x1FFF FFFF(512MB)
Block1	SRAM	0x2000 0000～0x3FFF FFFF(512MB)
Block2	片上外设	0x4000 0000～0x5FFF FFFF(512MB)
Block3	FSMC bank1 & bank2	0x6000 0000～0x7FFF FFFF(512MB)
Block4	FSMC bank3 & bank4	0x8000 0000～0x9FFF FFFF(512MB)
Block5	FSMC 寄存器	0xA000 0000～0xCFFF FFFF(512MB)
Block6	未使用	0xD000 0000～0xDFFF FFFF(512MB)
Block7	Cortex-M3 内部外设	0xE000 0000～0xFFFF FFFF(512MB)

8 个主要块中最重要的是 Block0、Block1 和 Block2，Block0 用于设计 Flash。STM32F03ZET6 处理器的 Flash 最大为 512KB，包含 Flash、预留、系统存储器、选项字节、boot 设置。Block1 用于设计 SRAM，STM32F103ZET6 处理器的 SRAM 最大为 64KB；Block2 用于片内外设，包括 APB1、ABP2、AHB 和部分预留空间。STM32F103ZET6 中内置外设的地址范围如表 3.5 所示。除去部分保留未使用的地址空间外，大部分地址范围均可以供给用户开发使用。

表 3.5　内置外设的地址范围

地 址 范 围	外　　设	总　　线
0x5000 0000～0x5003 FFFF	USB OTG 全速	AHB
0x4003 0000～0x4FFF FFFF	保留	
0x4002 8000～0x4002 9FFF	以太网	
0x4002 3400～0x4002 3FFF	保留	
0x4002 3000～0x4002 33FF	CRC 循环校验码	
0x4002 2000～0x4002 23FF	闪存存储器接口	
0x4002 1400～0x4002 1FFF	保留	
0x4002 1000～0x4002 13FF	复位和时钟控制(RCC)	
0x4002 0800～0x4002 0FFF	保留	
0x4002 0400～0x4002 07FF	DMA2	
0x4002 0000～0x4002 03FF	DMA1	
0x4001 8400～0x4001 7FFF	保留	
0x4001 8000～0x4001 83FF	SDIO	
0x4001 4000～0x4001 7FFF	保留	APB2
0x4001 3C00～0x4001 3FFF	ADC3	
0x4001 3800～0x4001 3BFF	USART1	
0x4001 3400～0x4001 37FF	TIM8 定时器	
0x4001 3000～0x4001 33FF	SPI1	
0x4001 2C00～0x4001 2FFF	TIM1 定时器	
0x4001 2800～0x4001 2BFF	ADC2	

续表

地址范围	外设	总线
0x4001 2400～0x4001 27FF	ADC1	APB2
0x4001 2000～0x4001 23FF	GPIO 接口 G	
0x4001 2000～0x4001 23FF	GPIO 接口 F	
0x4001 1800～0x4001 1BFF	GPIO 接口 E	
0x4001 1400～0x4001 17FF	GPIO 接口 D	
0x4001 1000～0x4001 13FF	GPIO 接口 C	
0x4001 0C00～0x4001 0FFF	GPIO 接口 B	
0x4001 0800～0x4001 0BFF	GPIO 接口 A	
0x4001 0400～0x4001 07FF	EXTI	
0x4001 0000～0x4001 03FF	AFIO	
0x4000 7800～0x4000 FFFF	保留	APB1
0x4000 7400～0x4000 77FF	DAC	
0x4000 7000～0x4000 73FF	电源控制(PWR)	
0x4000 6C00～0x4000 6FFF	后备寄存器(BKP)	
0x4000 6800～0x4000 6BFF	bxCAN2	
0x4000 6400～0x4000 67FF	bxCAN1	
0x4000 6000～0x4000 63FF	USB/CAN 共享的 512B SRAM	
0x4000 5C00～0x4000 5FFF	USB 全速设备寄存器	
0x4000 5800～0x4000 5BFF	I2C2	
0x4000 5400～0x4000 57FF	I2C1	
0x4000 5000～0x4000 53FF	UART5	
0x4000 4C00～0x4000 4FFF	UART4	
0x4000 4800～0x4000 4BFF	USART3	
0x4000 4400～0x4000 47FF	USART2	
0x4000 4000～0x4000 3FFF	保留	
0x4000 3C00～0x4000 3FFF	SPI3/I2S3	
0x4000 3800～0x4000 3BFF	SPI2/I2S2	
0x4000 3400～0x4000 37FF	保留	
0x4000 3000～0x4000 33FF	独立看门狗(IWDG)	
0x4000 2C00～0x4000 2FFF	窗口看门狗(WWDG)	
0x4000 2800～0x4000 2BFF	RTC	
0x4000 1800～0x4000 27FF	保留	
0x4000 1400～0x4000 17FF	TIM7 定时器	
0x4000 1000～0x4000 13FF	TIM6 定时器	
0x4000 0C00～0x4000 0FFF	TIM5 定时器	
0x4000 0800～0x4000 0BFF	TIM4 定时器	
0x4000 0400～0x4000 07FF	TIM3 定时器	
0x4000 0000～0x4000 03FF	TIM2 定时器	

表 3.5 中每个地址范围的第一个地址为对应外设的首地址，该外设的相关寄存器地址都可以用“首地址＋偏移量”的方式找到其绝对地址。

3.5 最小系统

处理器 CPU 的最小系统就是让处理器能正常工作并发挥其功能时所必需的组成部分，也可理解为处理器能正常运行的最小环境。最小系统一般分为 5 个部分，分别为：

（1）处理器芯片。运行任务程序及执行相应的控制动作。

（2）时钟电路。为处理器运行程序提供时钟源。

（3）复位电路。使处理器内部各个模块处于确定的初始状态。

（4）系统电源。提供系统工作电源。

（5）调试电路。提供运行程序的下载和运行监控。

要使 STM32 处理器能正常运行，必须具备以上 5 个电路。由于 STM32 处理器内部已经集成了时钟电路和调试电路，所以 STM32 处理器只需带有复位电路和提供工作电源，便可正常运行。但是为了让 STM32 处理器能提供灵活、可靠、稳定、抗干扰性较强的控制动作，最小系统可能还需具备其他附加电路。在此以 STM32F103ZET6 处理器为例，介绍一个完整的最小系统实例。

3.5.1 复位电路

如图 3.5 所示，最简单的复位电路可由电容串联电阻构成，其基本原理为电容的充放电过程。我们知道，电容的电压不能突变，当系统上电时处理器的 RST 脚将会出现一个持续的高电平，并且这个高电平持续的时间由电路的电容值的充电过程决定。STM32 处理器的 RST 脚检测持续 20μs 以上的高电平后，就会对处理器进行复位操作。所以适当组合 *RC* 的取值就可以保证系统可靠的复位。

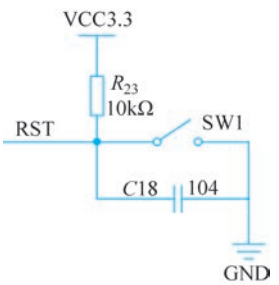

图 3.5 复位电路

3.5.2 时钟电路

如图 3.6 所示，为了提供更为精准的时钟信号，处理器可采用外置时钟电路，其主要由晶振、电容和电阻构成。处理器内部振荡器在外部接续晶振和电容的作用下产生自激振荡，为处理器提供 8MHz 的正弦信号。时钟电路相当于处理器的心脏，它的每次跳动，也称振荡节拍，都控制着处理器执行代码的工作节奏。振荡节拍慢时系统工作速度就慢，振荡节拍快时系统工作速度就快。

在很多控制系统中都要用到实时时钟（Real Time Clock，RTC）电路，而确保 RTC 工作计时准确的关键部分就是 32.768kHz 的晶体振荡电路。STM32F1 处理器除了具有主时钟电路外，还内置了 RTC 时钟。只需要在 OSC32IN 和 OSC32OUT 两引脚间外接晶振即可实现。如图 3.7 所示，可采用基于 EPSON 的 FC-135/FC-135R 贴片音叉晶振或使用 C2 系列圆柱体晶振（32.768kHz）搭建的晶振电路产生 32.768kHz 的正弦波以提供实时时钟信号。

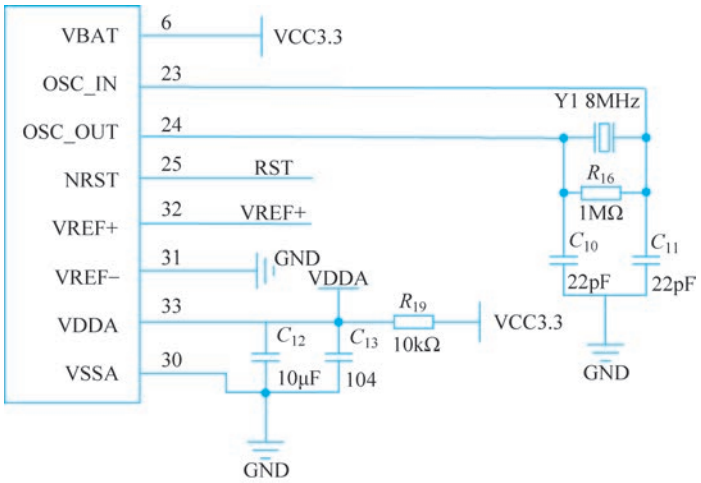

图 3.6　主时钟电路

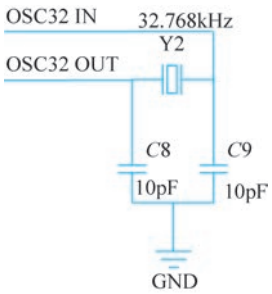

图 3.7　32.768kHz RTC 晶振外接电路图

3.5.3　电源 DC-DC 转换电路

由于 STM32F1 处理器电源采用 3.3V 供电。一般常用的电源是 5V。如 TTL 电平也为 5V，市场售卖的电源和 USB 接口提供的均是 5V，所以需要对电压进行降压处理。此 DC-DC 电路使用了 AMS1117-3.3 DC-DC 转换芯片，可将 5V 转换为 3.3V 提供处理器使用。其具体 DC-DC 转换电路如图 3.8 所示。

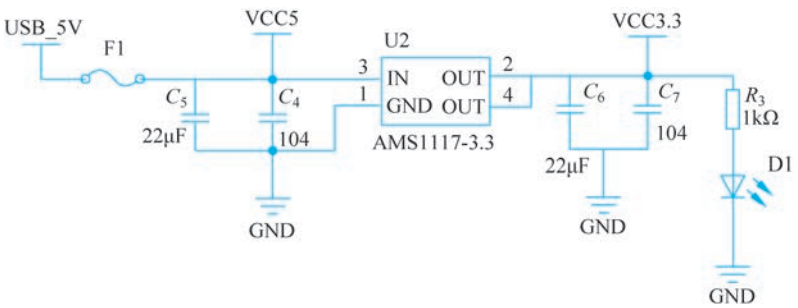

图 3.8　直流 DC-DC 电源转换电路

3.5.4 系统调试电路 JTAG

系统调试电路JTAG(Joint Test Action Group,联合测试行动小组)是一种国际标准测试协议,主要用于芯片内部测试以及对系统进行仿真和调试。JTAG 技术是一种嵌入式调试技术,它在处理器芯片内部封装了专门的测试电路 TAP(Test Access Port,测试访问口),通过专用的 JTAG 测试工具对内部节点进行测试。目前大多数比较复杂的器件都支持 JTAG 协议,如 ARM、DSP 和 FPGA 器件等均支持 JTAG。标准的 JTAG 接口具有 4 线数据信号:TMS、TCK、TDI 和 TDO,分别对应为测试模式选择、测试时钟、测试数据输入和测试数据输出。JTAG 测试也允许多个器件通过 JTAG 接口串联在一起联调,形成一个 JTAG 链,能分别实现对各个器件的测试。JTAG 接口还常用于实现 ISP(In-System Programmable,在系统编程)功能,如对 Flash 器件进行编程等。通过 JTAG 接口,可对芯片内部的所有部件进行访问,因而它是开发调试嵌入式系统的一种简洁高效的手段。目前 JTAG 接口的连接器有两种标准,即 14 针接口和 20 针接口连接标准,其具体引脚定义分别如图 3.9 与图 3.10 以及表 3.6 与表 3.7 所示。

VREF	p	1	2	p	GND
TRST_N	i	3	4	p	GND
TDI	i	5	6	p	GND
TMS	i	7	8	p	GND
TCK	i	9	10	p	GND
TDO	o	11	12	od	SRST_N
VREF	p	13	14	p	GND

图 3.9 JTAG 14 针接口定义和信号名称

VREF	p	1	2	nc	—
TRST_N	i	3	4	p	GND
TDI	i	5	6	p	GND
TMS	i	7	8	p	GND
TCK	i	9	10	p	GND
—	nc	11	12	p	GND
TDO	o	13	14	p	GND
SRST_N	od	15	16	p	GND
—	nc	17	18	p	GND
—	nc	19	20	p	GND

20针不带RTCK

VREF	p	1	2	nc	—
TRST_N	i	3	4	p	GND
TDI	i	5	6	p	GND
TMS	i	7	8	p	GND
TCK	i	9	10	p	GND
RTCK	o	11	12	p	GND
TDO	o	13	14	p	GND
SRST_N	od	15	16	p	GND
—	nc	17	18	p	GND
—	nc	19	20	p	GND

20针带RTCK

图 3.10 JTAG 20 针接口定义引脚

表 3.6 JTAG 14 针接口引脚定义

引脚和信号	信 号 定 义
1、13	VREF 接电源
2、4、6、8、10、14	GND 接地
3 TRST_N	测试系统复位信号
5 TDI	测试数据串行输入
7 TMS	测试模式选择
9 TCK	测试时钟
11 TDO	测试数据串行输出
12 SRST_N	目标系统复位信号

表 3.7 JTAG 20 针接口定义引脚名称描述

引脚和信号	信 号 定 义
1 VREF	目标板参考电压,接电源
2 nc	未连接
3 TRST_N	测试系统复位信号
4、6、8、10、12、14、16、18、20	GND 接地
5 TDI	测试数据串行输入
7 TMS	测试模式选择
9 TCK	测试时钟
11 RTCK	测试时钟返回信号
13 TDO	测试数据串行输出
15 SRST_N	目标系统复位信号
17、19 nc	未连接

3.5.5 其他辅助电路

通常在设计控制系统时,为保证系统稳定工作,常在所有的电源引脚旁边放置一个 0.1μF 的电容滤波,用来滤除电源的噪声杂波。具体连接如图 3.11 所示。

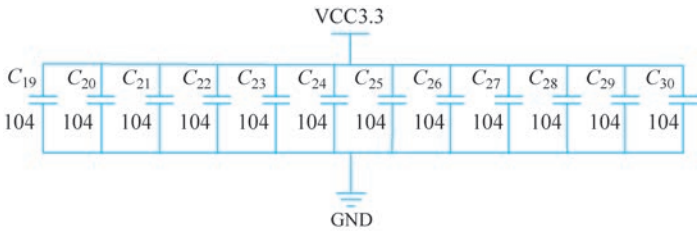

图 3.11 电源高频噪声杂波滤波电路

在复位后程序启动,如图 3.12 所示,通常可使用 Boot0/Boot1 均置为 0,即均为低电平。Cortex-M3 内核的器件有 3 种启动方式,Cortex-M4 有 4 种,可通过 Boot0、Boot1 引脚的电平进行选择。如第 2 章所述,STM32 的 3 种启动模式对应的存储介质均是芯片内置的,它们是:用户闪存(芯片内置的 Flash);SRAM(芯片内置的 RAM 区),也称为内存;系统存储器,即芯片内部一块特定的区域,芯片出厂时在这个区域预置了一段 Bootloader,也就是通常说的 ISP 程序。这个区域的内容在芯片出厂后没有人能够修改或擦除,即它是一个 ROM 区,它是使用 USART1 作为通信接口。

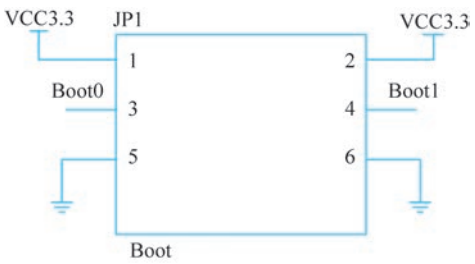

图 3.12 Boot0/Boot1 设置和对应启动模式

应用系统通常还需要设计状态 LED 指示电路。如图 3.13 所示，两盏状态指示灯 LED1 和 LED2 进行系统工作状态的显示。LED1 和 LED2 分别与主芯片的 GPIO 引脚连接，串联电阻为限流电阻，防止电流过大损坏发光二极管。LED 指示灯数量可根据应用需求增减。

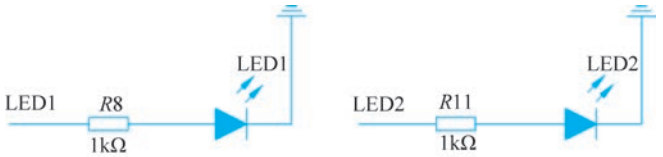

图 3.13　状态指示灯 LED1 和 LED2 电路

键盘是用于操作设备运行的一种指令和数据输入装置，也指经过系统安排操作一台机器或设备的一组功能键。键盘是计算机最常用也是最主要的输入设备，通过键盘可以将英文字母、数字以及标点符号等输入到系统中，从而向处理器发出命令以及输入数据等。如图 3.14 所示，键盘的设计非常简单，但是必须配以处理器驱动程序才能完成相关功能。另外，为防止按键抖动，常常还配以防抖动辅助电路。防抖动辅助功能可以采用硬件电路或软件来实现。

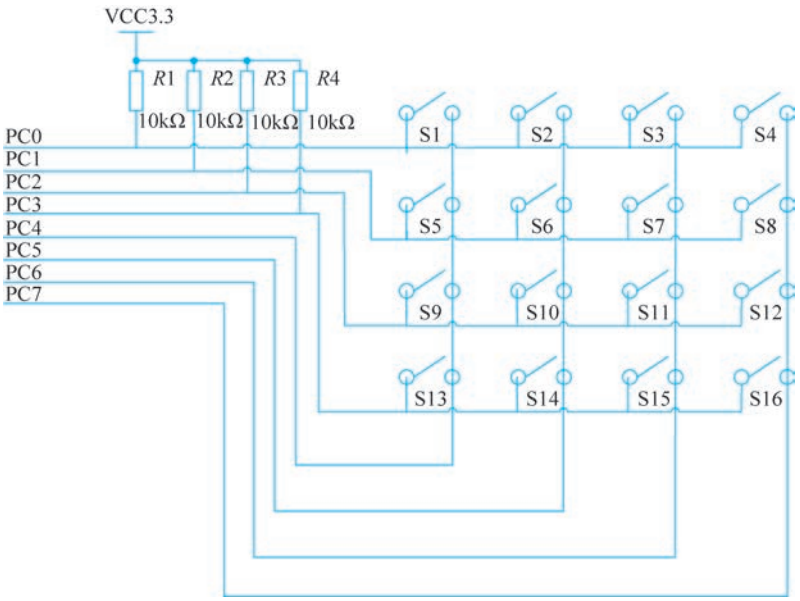

图 3.14　键盘 4×4 矩阵电路示意图

数码管显示电路也称作辉光管，是一种可以显示数字和其他信息的电子设备。玻璃管中包括一个金属丝网制成的阳极和多个阴极。大部分数码管阴极的形状为数字。管中充以低压气体，通常大部分为氖加上一些汞和/或氩。给某一个阴极充电，数码管就会发出颜色光，由管内的气体种类决定，一般都是橙色或绿色。如图 3.15 所示，数码管动态显示器是应用系统中最为广泛的一种显示器，常用数码管引脚定义如图 3.15 所示。数码管的 8 个显示笔画 A、B、C、D、E、F、G、DP，用于显示日常生活和工作中实用的数字

和小数点。也可利用多个数码管排列显示各种数值。

如图 3.15 中的 A、B、C、D、E、F、G、DP 分别可与处理器的 GPIO 接口或驱动 IC 集成电路相连，用来控制显示数字的形状。Q3、Q4、Q5、Q6、Q7、Q8 这 6 个三极管是用来驱动和片选数码管，用于打开或关闭某一位数码管的显示。RA1、RA0、RA3、RA2、RA5、RA4 分别接在处理器的其他 GPIO 接口上，通过控制这些三极管的基极电平来打开或关闭数码管的显示，即起到“使能”作用。其电路原理如图 3.16 所示。

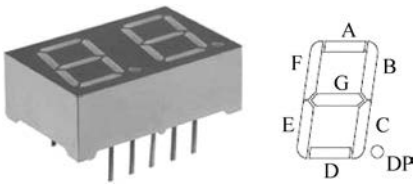

图 3.15　数码管示意图

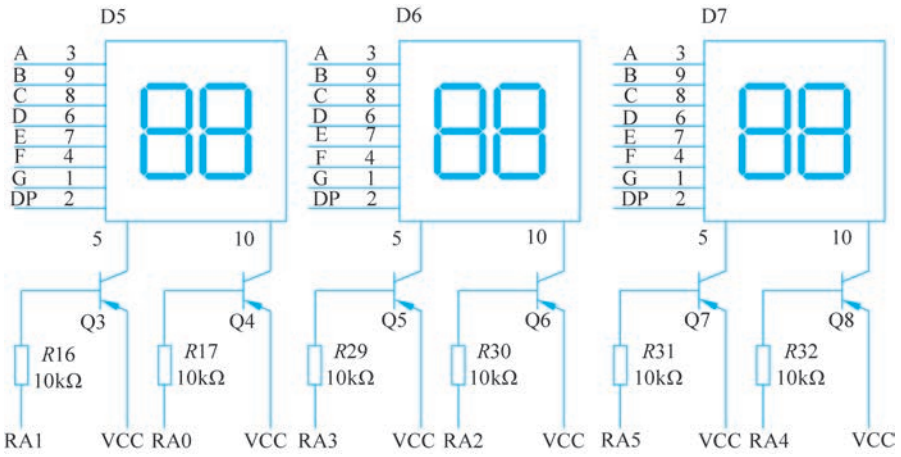

图 3.16　多位数码管设计电路

3.6 STM32 最小系统和拓展实验平台

如图 3.17 所示为一款简单实用完整的最小系统。在该最小系统电路中，除处理器 CPU 电路、晶振电路、电源电路等必要的电路外，还设计了 LED 电路、数码管电路、按键电路和 LCD 电路等拓展辅助功能电路。

LED 电路包含 12 个可编程使用的 LED 灯，其分别连接于 CPU 芯片的 PE5、PE6、PF0～PF8 和 PF12 接口引脚。另外，LED 灯可以根据颜色分为 3 组，分别对应红、绿、黄这 3 种颜色，可用于编程模拟交通灯的功能。由电路图可以看出，发光二极管 LED0～LED11 的正极均通过限流电阻连接于 3.3V 电源，负极则直接连在 GPIO 引脚上。因此在用户编程时，若使某个 GPIO 引脚输出信号为低电平，则对应的 LED 灯亮；反之输出高电平，则 LED 灯灭。

数码管电路包含两个可供编程使用的数码管，每个数码管通过移位寄存器 74HC595 芯片连接于 CPU 芯片的 GPIO 引脚。其中 74HC595 芯片的引脚功能描述如表 3.8 所示。例如，数码管 P1 连接 74HC595 芯片 U6 的 1～7 脚，U6 的 11、12、14 脚连接 CPU 芯片的 PA7、PA6、PA5。如此便可以通过 CPU 芯片来控制 74HC595 芯片的输出，再以此

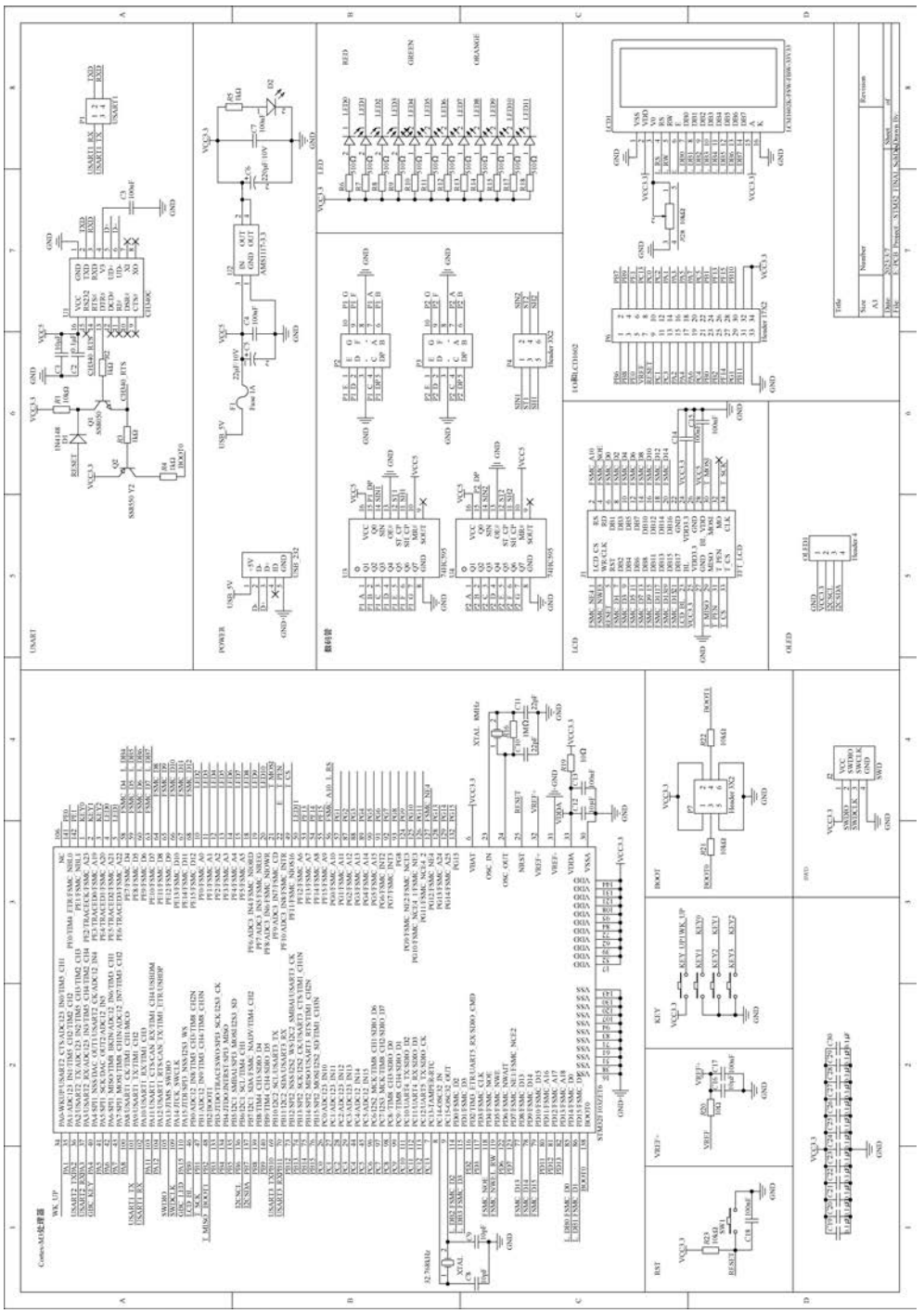

图 3.17　实验最小系统电路图

输出来驱动数码管显示。此外，还需要注意，此电路中使用的是共阴极数码管，因此其公共端需要接地处理。

表 3.8　74HC595 IC 引脚功能描述

符　　号	引　　脚	描　　述
Q0～Q7	15 脚、1～7 脚	8 位并行数据输出
GND	8 脚	地
SOUT	9 脚	串行数据输出
MR＃	10 脚	主复位(低电平有效)
SH_CP	11 脚	数据输入时钟线
ST_CP	12 脚	输出存储器锁存时钟线
OE＃	13 脚	输出有效(低电平有效)
SIN	14 脚	串行数据输入
VCC	16 脚	电源

按键电路包含 4 个独立按键，即 KEY_UP、KEY1、KEY2、KEY3。由电路图可以看出，4 个按键的一端分别连接于 CPU 芯片的 PA0、PE2、PE3、PE4 引脚，而另一端的接法稍有区别。其中，按键 KEY_UP 的另一端直接连接于 3.3V 电源，而按键 KEY1～KEY3 的另一端则直接接地。因此，当我们使用按键时，对于电路接法不同的按键常常要使用不同的初始化配置。就 STM32F103ZET6 这款芯片而言，如果需要使用这 4 个按键，则要提前将 PA0 配置为下拉输入模式，并将 PE2～PE4 配置为上拉输入模式。LCD 电路提供了一组 LCD 接口用于外接 LCD 显示屏。当需要使用 LCD 模块时，只需要将特定的 LCD 模块插入如图 3.17 所示的 LCD 接口，然后即可通过 GPIO 引脚初始化和控制 LCD 模块的显示，操作起来非常方便实用。

3.7 STM32 实验环境构建

由于 STM32 开发板上已经设计了一键下载电路，因此硬件实验环境构建特别简单，主要分为以下几步：

(1) 置启动模式。Boot1 接 GND，Boot0 接 VCC，配置从系统存储器启动。Boot 原理图如图 3.18 所示，也就是将 1 脚和 3 脚、4 脚和 6 脚分别短接。一般来说，也就是为了从串口下载程序。

(2) 连接 USB 转串口芯片与主芯片的对应引脚。串口接线原理如图 3.19 所示。转串口芯片的 TXD 和 RXD 这两个引脚并未直接与 STM32F103ZET6 的 USART1 对应引脚相连，中间隔了一个标准的排针接口。因此，如果想要利用 USART1 下载程序，需要将 1 脚和 2 脚、3 脚和 4 脚分别短接。

(3) 用 USB 线连接开发板与 PC 端。USB 接口原理图如图 3.20 所示。由于使用了一键下载电路，我们只需要利用一根 USB 线就可以实现供电和通信这两大功能，接线特别方便、简单，不需要额外连接串口线。开发板的 USB 接口采用的是 mini USB，该接口和信号线的实物图分别如图 3.21 和图 3.22 所示。

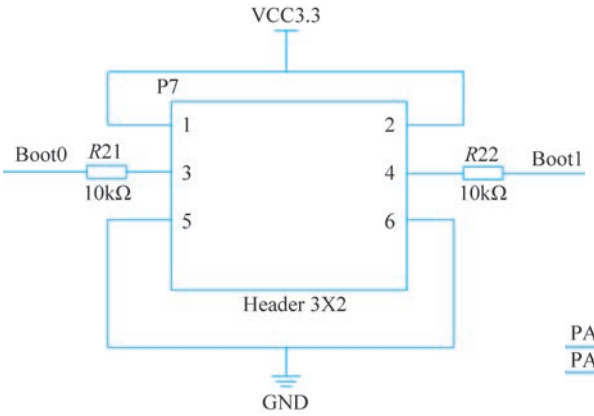

图 3.18　Boot 原理图

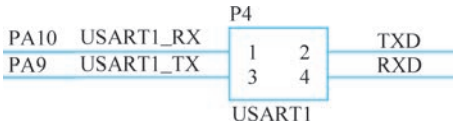

图 3.19　串口接线原理图

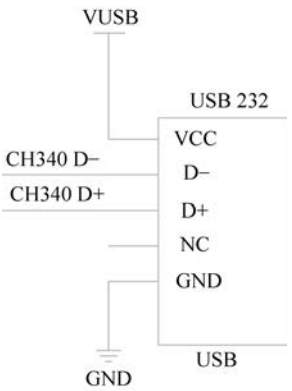

图 3.20　USB 接口原理图

图 3.21　mini USB 接口

经过上述的几个步骤以后，STM32 的硬件实验环境已经构筑完成，下面只需要进行软件开发环境的配置。实际开发系统的实物如图 3.23 所示。

图 3.22　mini USB 信号连接线

图 3.23　实验系统连接与开发示意图

3.8　STM32 软件开发与仿真环境构建

1. MDK 软件安装

首先从 Keil 官网获取 MDK 软件安装包。安装包是一个扩展名为.exe 的可执行文

件，本节中均以 MDK5.27 版本为例。下载完成之后，单击 mdk527.exe 文件进行安装，如图 3.24 所示。

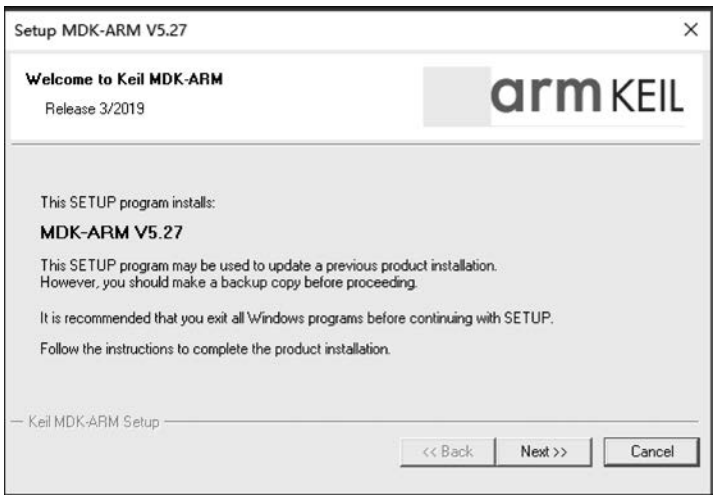

图 3.24　MDK5.27 安装界面

然后单击 Next 按钮，进入 License Agreement 界面，如图 3.25 所示，选择同意协议内容。

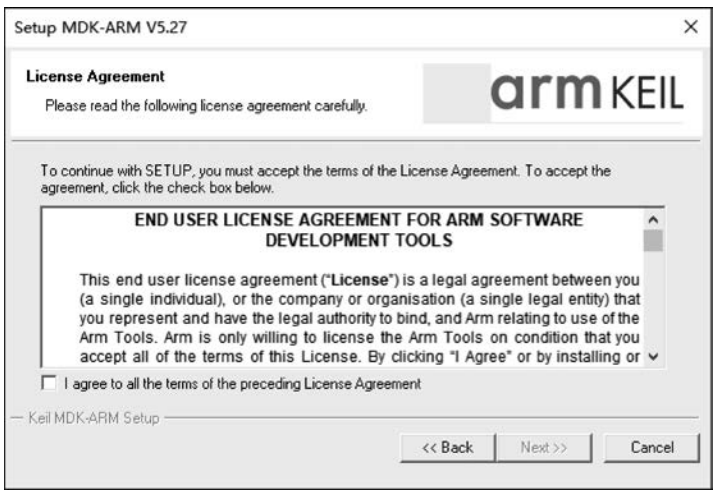

图 3.25　License Agreement 界面

接着单击 Next 按钮，选择软件安装路径，如图 3.26 所示。这里选择创建一个 Keil_v5 文件夹，同时应注意文件夹不要命名为中文或带有空格。

再单击 Next 按钮，填写个人信息，如图 3.27 所示。

填完信息之后再单击 Next 按钮就正式开始了软件安装，安装界面如图 3.28 所示。

软件安装成功后单击 Finish 按钮会弹出 Pack Installer 界面，如图 3.29 所示。这里单击关闭按钮即可，因为我们后面会通过已下载好的包进行安装。

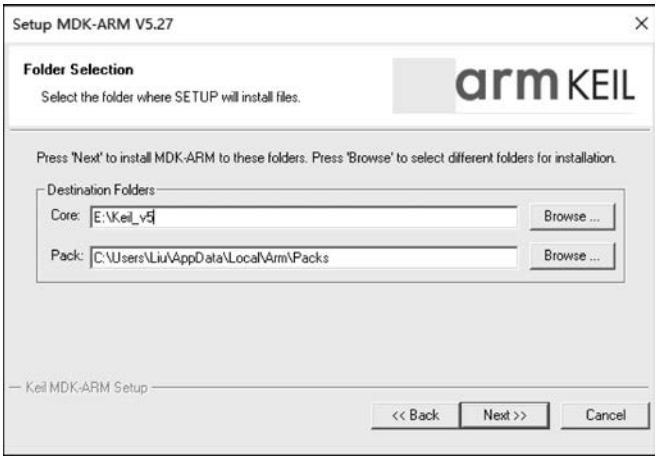

图 3.26　安装路径选择

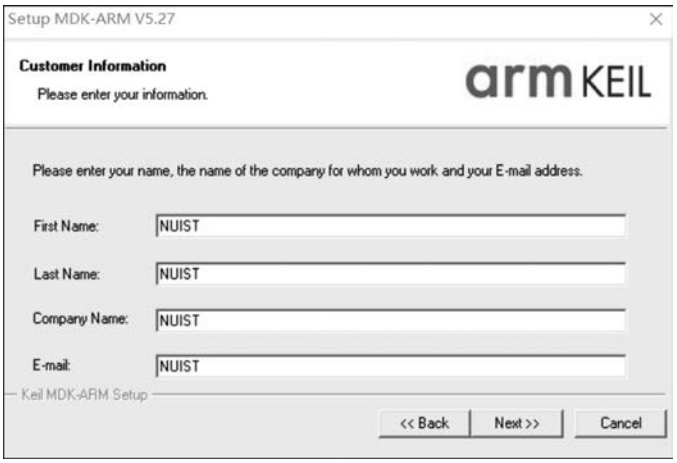

图 3.27　填写个人信息

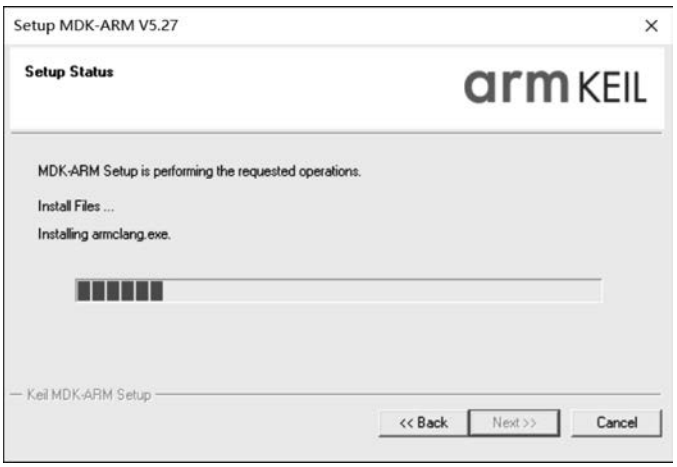

图 3.28　软件安装界面

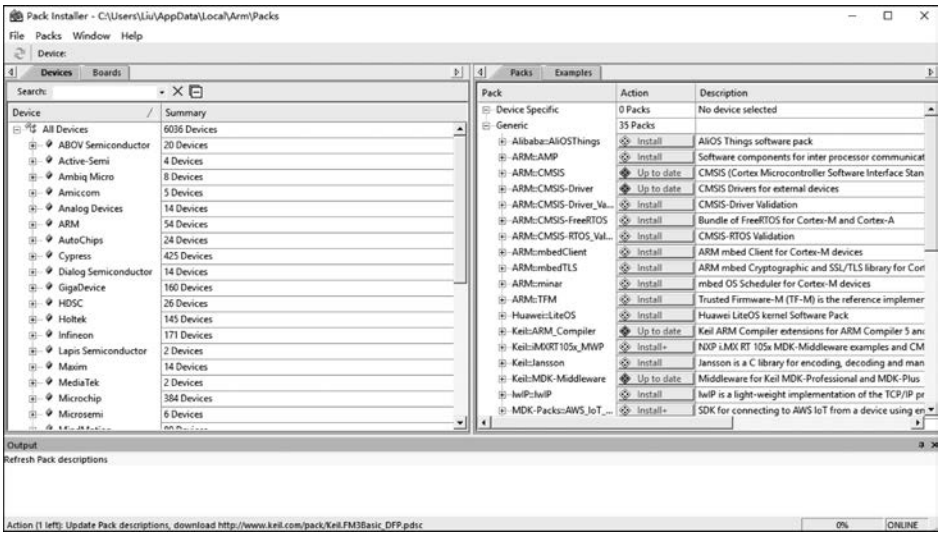

图 3.29　Pack Installer 界面

2. STM32F1 器件包安装

我们需要从 Keil 官网获取 STM32F1 系列芯片的器件包，下载完成后如图 3.30 所示。双击其中的 Keil.STM32F1xx_DFP.2.3.0 文件即可进行安装。安装界面如图 3.31 所示，单击其中的 Next 按钮开始安装，过程也比较简单。

图 3.30　STM32F1 系列芯片的器件包

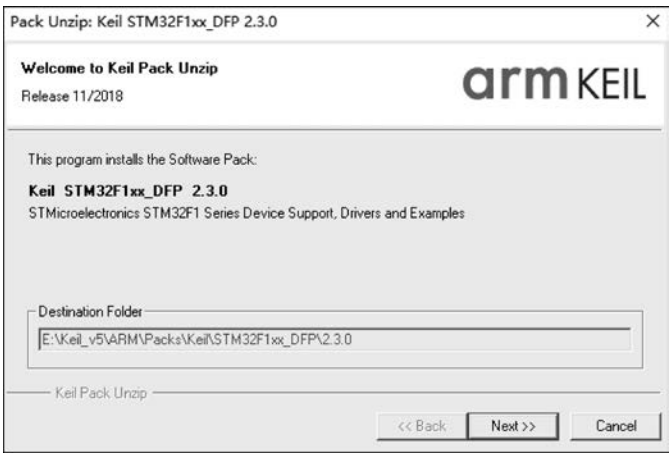

图 3.31　STM32F1 器件包安装

在用 Keil 创建实际工程时都会弹出如图 3.32 所示的设备选择界面。如果器件包安装成功了,其中就会有 STM32F1 系列芯片供选择。

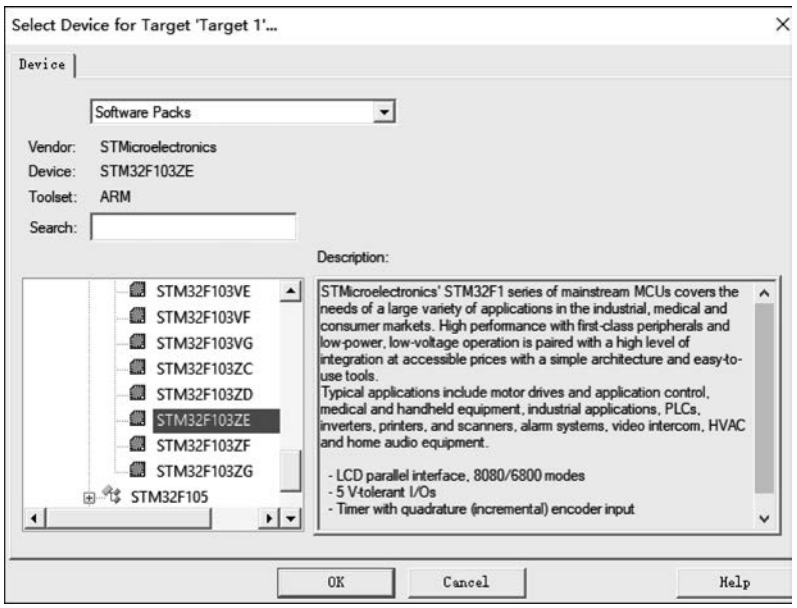

图 3.32　设备选择界面

3. 安装 USB 转串口驱动

本节中使用的转串口芯片是 CH340 系列,驱动文件可以从沁恒微电子官网获取。下载成功后得到一个名为 CH341SER.INF 的文件,双击该文件进行安装。驱动安装界面如图 3.33 所示,单击“安装”按钮即可。如果没有安装驱动会导致计算机无法识别串口,也就无法通过串口下载程序。

安装成功后会弹出提示对话框,如图 3.34 所示。

图 3.33　驱动安装

图 3.34　驱动安装成功

4. FlyMcu 软件配置

通常使用 FlyMcu 软件利用串口将程序下载至 STM32 中。其中 FlyMcu 是免安装的,直接双击打开即可,软件界面如图 3.35 所示。同时还需要打开 STM32 的电源,这样串口才能被检测到并用于下载程序。

图 3.35　FlyMcu 软件界面

此外，使用该软件时有以下几点需要注意：

（1）对于 STM32F1 系列芯片来说波特率可设置为任意值。

（2）选中“校验”和“编程后执行”这两个选项，不要选中“编程到 FLASH 时写选项字节”。

（3）在界面下方选择“DTR 高电平复位，RTS 高电平进 BootLoader”。当单击打开 FlyMcu 的下拉列表框时，可以看到有很多不同的配置方案，如图 3.36 所示。这里之所以这样选择，其实是与一键下载电路的电路结构有关，如图 3.37 所示。FlyMcu 通过 USB 信号线控制 CH340 转串口芯片的 DTR 和 RTS 引脚的输出，这两个输出进而控制 Q1 和 Q2 的导通与截止，从而进一步影响 RESET 和 BOOT0 这两个引脚的电平。这样就实现了一键下载功能。

图 3.36　DTR 和 RTS 配置

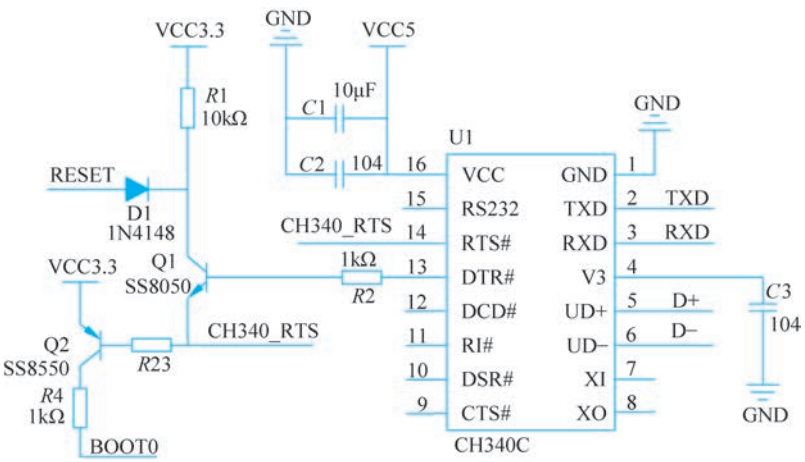

图 3.37　一键下载电路原理图

5. 固件库文件下载

固件库文件可以从ST的官网处获取，用于移植到STM32工程中。本节中使用的是V3.5.0版本，固件库解压后文件夹名称为STM32F10x_StdPeriph_Lib_V3.5.0，其目录结构如图3.38所示。

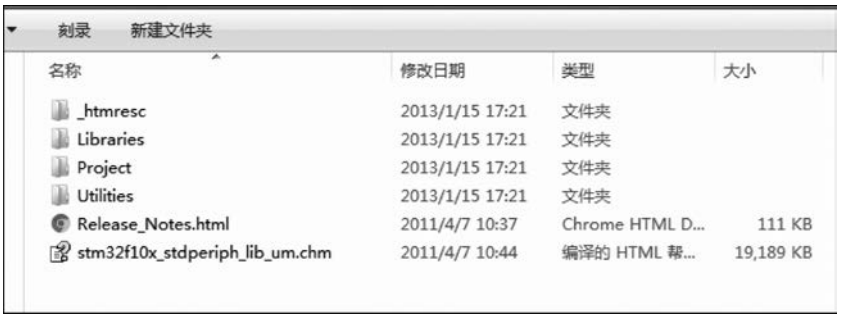

图3.38 固件库文件目录结构

其中，_htmresc文件夹中是一些官方的logo图片，Project文件夹中是官方模板工程示例，Utilities文件夹中是官方评估模板例程，Libraries文件夹中则是移植工程时需要用到的固件库的相关源码。

6. 新建工程模板

(1) 首先新建一个文件夹，后面所建立的工程都可以放在这个文件夹中，这里将之命名为STM32_Project。

(2) 单击Project→New μVision Project菜单命令，如图3.39所示。然后将目录定位到刚才建立的STM32_Project文件夹之下。先在这个文件夹下建立子文件夹Project，再定位到Project文件夹下，将工程命名为Template，单击“保存”按钮，如图3.40所示。

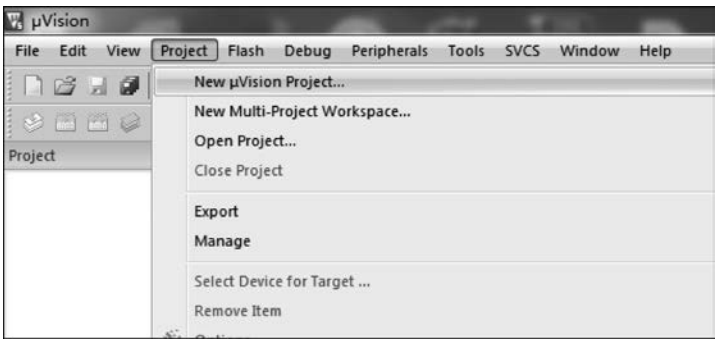

图3.39 新建工程

接下来会出现一个选择CPU的界面，也就是选择芯片型号，如图3.41所示。因为使用的主芯片CPU为STM32F103ZET6，所以在这里选择STMicroelectronics→STM32F1 Series→STM32F103→STM32F103ZET6。

(3) 单击OK按钮，MDK会弹出Manage Run-Time Environment对话框，如图3.42所示。

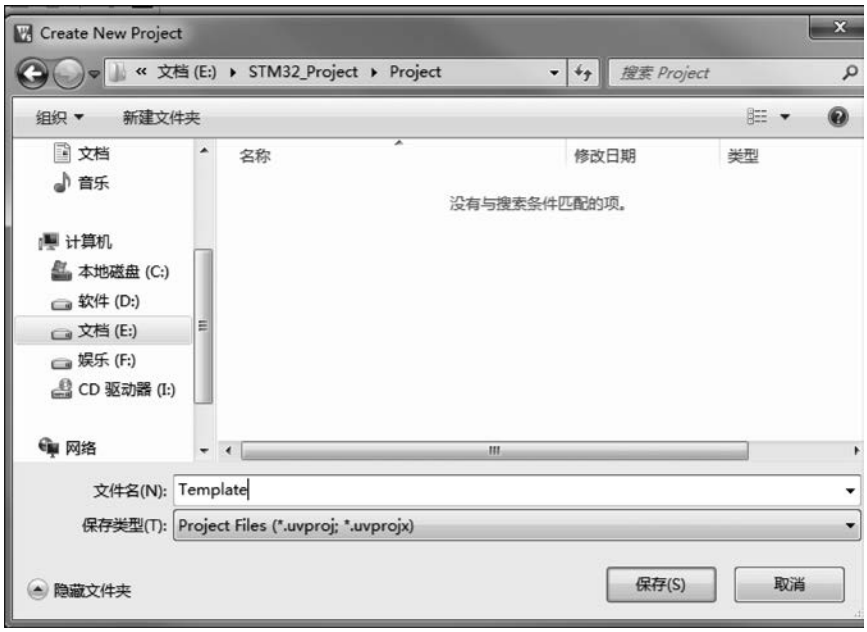

图 3.40　定义工程名称

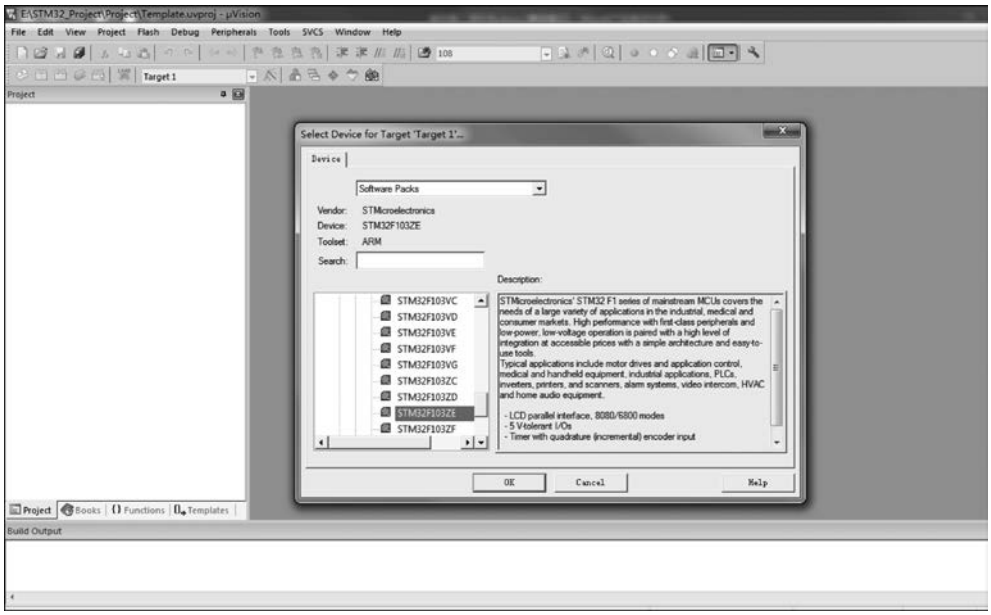

图 3.41　选择芯片型号

这是 MDK5 新增的一个功能，在这个界面中可以添加自己需要的组件，从而方便构建开发环境。在这里直接单击 Cancel 按钮即可，然后得到如图 3.43 所示的界面。

至此，我们只是创建了一个框架，还需要添加启动代码，以及相关的.c 文件等。

(4) 现在可以看到 Project 文件夹下包含 2 个文件夹和 2 个文件，如图 3.44 所示。

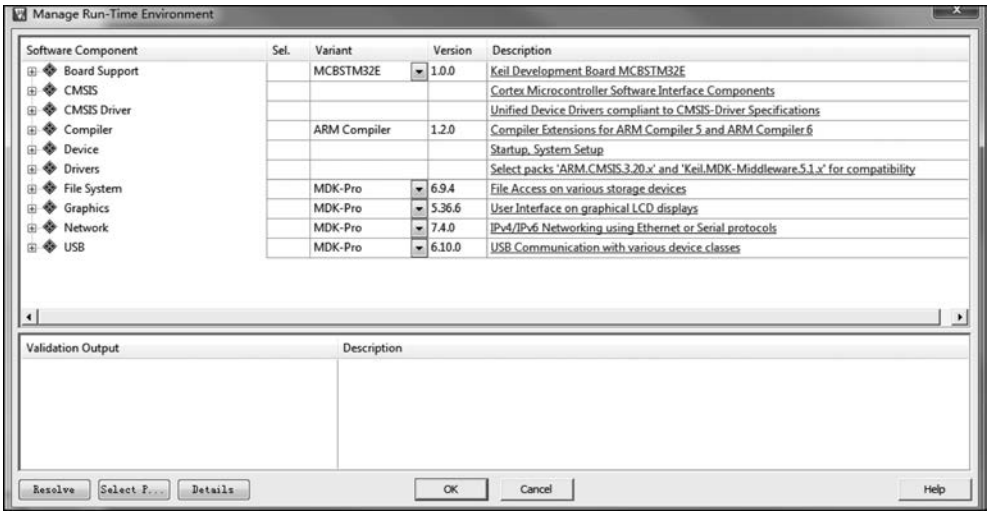

图 3.42　Manage Run-Time Environment 对话框

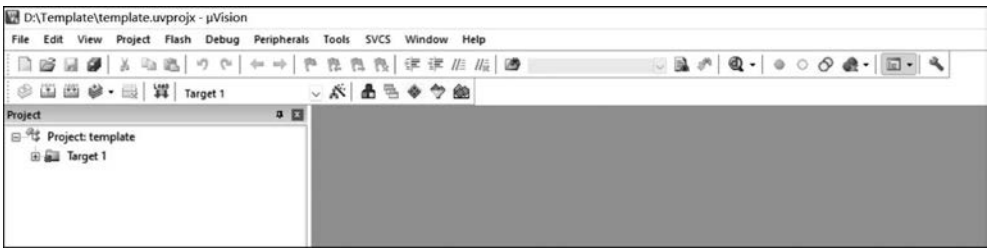

图 3.43　工程初步建立

图 3.44　工程 Project 目录文件

在此必须说明，Template. uvoptx 是工程文件，不能轻易删除。Listings 和 Objects 文件夹是 MDK 自动生成的文件夹，用于存放编译过程产生的中间文件。为了与 MDK5. 1 之前版本工程兼容，在此把两个文件夹删除。新建一个 OBJ 文件夹，用来存放编译中间文件。

（5）在 STM32_Project 文件夹下新建 3 个文件夹，分别是 CORE、OBJ 和 STM32F10x_FWLib，如图 3. 45 所示。CORE 用来存放核心文件和启动文件，OBJ 用来存放编译过程文件以及. hex 文件，STM32F10x_FWLib 用来存放 ST 官方提供的库函数源码文件。已有的 Project 文件夹除了用来放工程文件外，还用来存放主函数文件 main. c 以及 system_

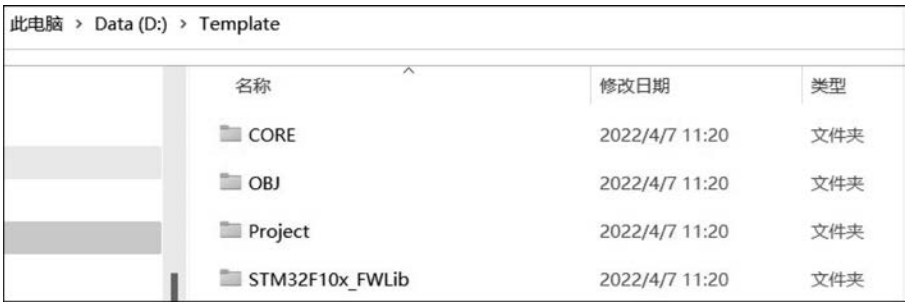

图 3.45　工程目录

stm32f10x.c 等其他文件。

(6) 将官方固件库中的源码文件复制到建立的工程目录下面。打开官方固件库包，定位到之前准备好的固件库包的目录 STM32F10x_StdPeriph_Lib_V3.5.0\Libraries\STM32F10x_StdPeriph_Driver 下面，将该目录下面的 src、inc 文件夹复制到刚才建立的 STM32F10x_FWLib 文件夹下面，如图 3.46 所示。其中 src 存放固件库的相关.c 文件，inc 存放对应的.h 文件。

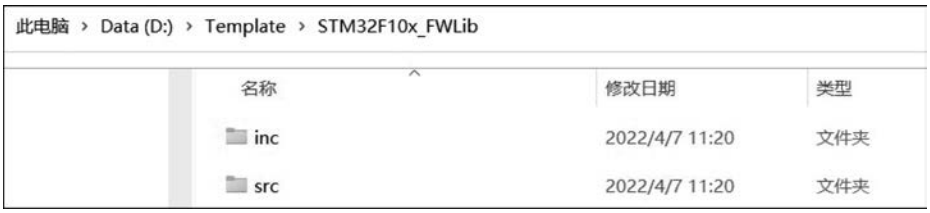

图 3.46　官方库源码文件夹

(7) 将固件库中相关的启动文件复制到工程目录 CORE 之下，具体如表 3.9 所示。复制成功后 CORE 文件夹如图 3.47 所示。

表 3.9　CORE 文件夹所需文件

源文件所在目录	文　件　名
STM32F10x_StdPeriph_Lib_V3.5.0\Libraries\CMSIS\CM3\CoreSupport	core_cm3.c、core_cm3.h
STM32F10x_StdPeriph_Lib_V3.5.0\Libraries\CMSIS\CM3\DeviceSupport\ST\STM32F10x\startup\arm	startup_stm32f10x_hd.s

图 3.47　CORE 文件夹

(8) 将固件库中的相关文件复制到 Project 文件夹下,具体如表 3.10 所示。复制成功后 Project 文件夹如图 3.48 所示。

表 3.10 Project 文件夹所需文件

源文件所在目录	文 件 名
STM32F10x_StdPeriph_Lib_V3.5.0\Libraries\CMSIS\CM3\DeviceSupport\ST\STM32F10x	stm32f10x.h、system_stm32f10x.c、system_stm32f10x.h
STM32F10x_StdPeriph_Lib_V3.5.0\Project\STM32F10x_StdPeriph_Template	main.c、stm32f10x_conf.h、stm32f10x_it.c、stm32f10x_it.h

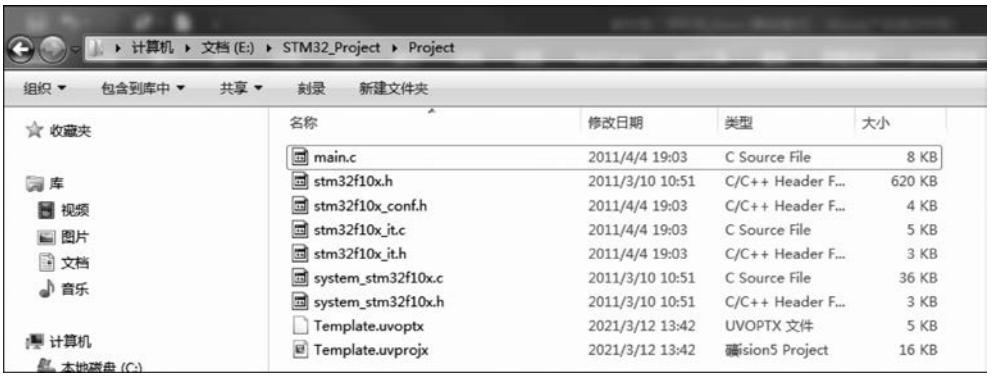

图 3.48 Project 文件夹中的文件

(9) 前面 8 个步骤,将需要的固件库相关文件复制到了自己的工程目录下面。下面再将这些文件加入自己的工程中去。右击 Target1,选择 Manage Project Items,如图 3.49 所示。

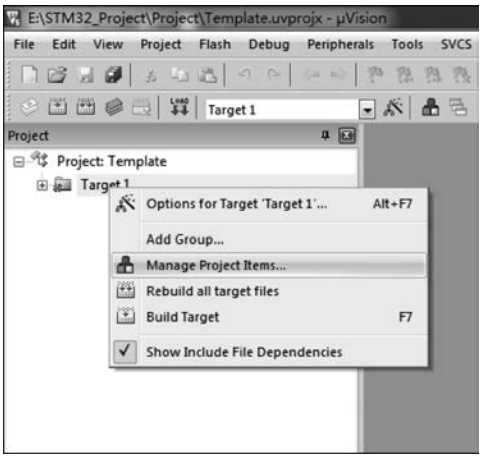

图 3.49 选择 Manage Project Items

(10) 弹出如图 3.50 所示界面后,在 Project Targets 一栏中将 Target 名字修改为 Template,然后在 Groups 一栏删掉 SourceGroup1,新建 3 个分组:PROJECT、CORE 和

FWLIB。最后单击 OK 按钮，可以看到自己的 Target 名字以及 Groups 的情况，如图 3.51 所示。

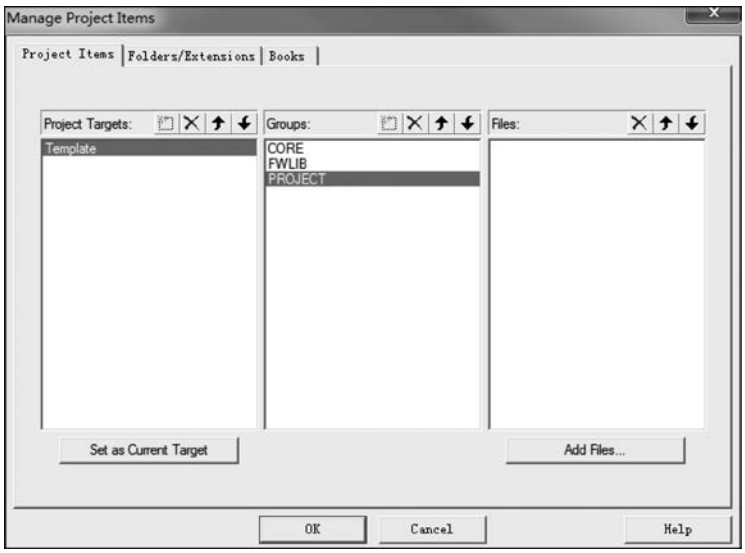

图 3.50　新建分组

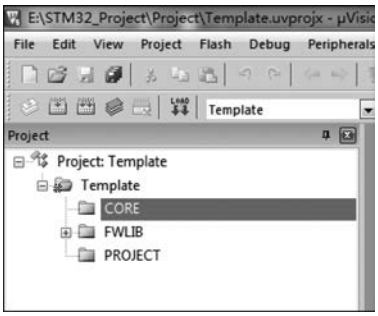

图 3.51　工程主界面

(11) 下面向分组中添加自己工程需要的文件。按照步骤(10)的方法，右击 Template，选择 Manage Project Items，然后选择需要添加文件的分组。这里第一步选择 FWLIB，然后单击右下角的 Add Files 按钮，定位到刚才建立的目录 STM32F10x_FWLib/src，将里面所有的文件选中并单击 Add 按钮，最后单击 Close 按钮。可以看到 Files 列表中包含了添加的文件，如图 3.52 所示。

(12) 用同样的方法，定位到 CORE 和 PROJECT 分组，添加需要的文件。这里 CORE 下面需要添加的文件为 core_cm3.c、startup_stm32f10x_hd.s(注意，默认添加时文件类型为.c，也就是添加 startup_stm32f10x_hd.s 启动文件时，需要选择文件类型为 All files 才能看得到这个文件)，PROJECT 下面需要添加的文件为 main.c、stm32f10x_it.c 和 system_stm32f10x.c，分别如图 3.53 和图 3.54 所示。如此就把需要添加的文件添加到自己的工程中了，最后单击 OK 按钮，回到工程主界面。最终的工程结构如图 3.55 所示。

(13) 接下来选择编译中间文件存放目录。右击 Template，单击 Options for Target 'Template'，如图 3.56 所示。然后在如图 3.57 所示的界面中，选择上面新建的 OBJ 文件夹。注意，如果不设置 Output 路径，那么默认的编译中间文件存放位置就是 MDK 自动生成的 Objects 文件夹和 Listings 文件夹。另外，需选中 Create HEX File 复选框，这样编译项目代码时才会生成.hex 文件。

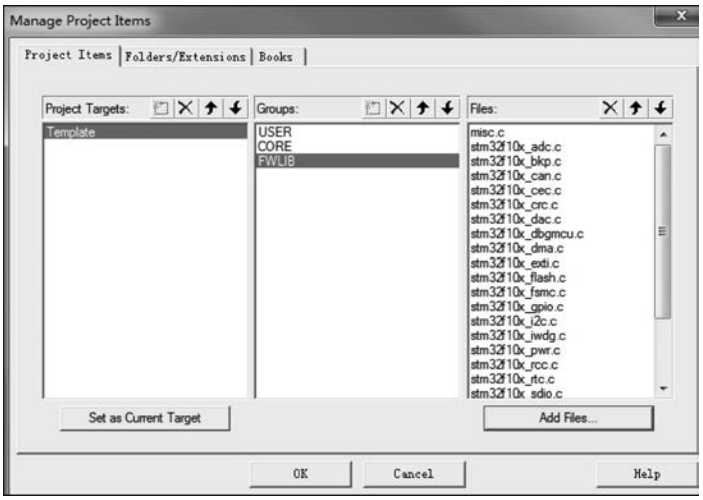

图 3.52　添加文件到 FWLIB 分组

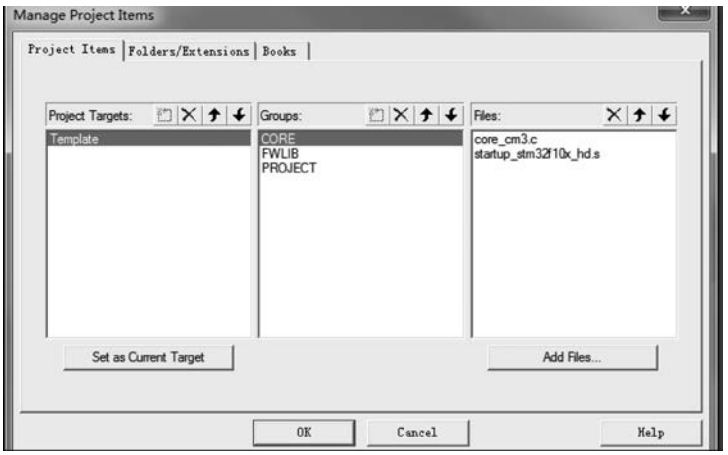

图 3.53　添加文件到 CORE 分组

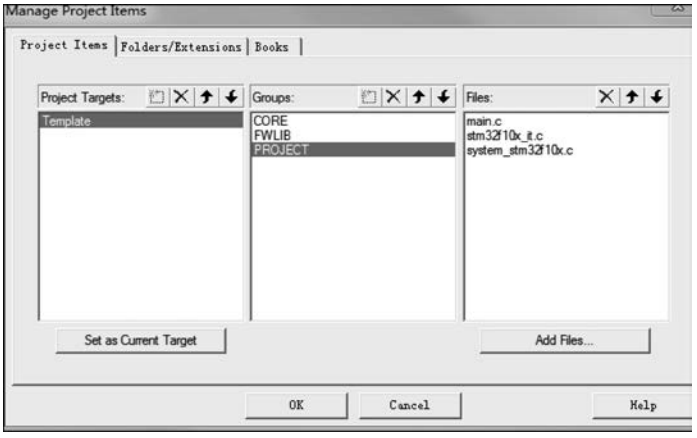

图 3.54　添加文件到 PROJECT 分组

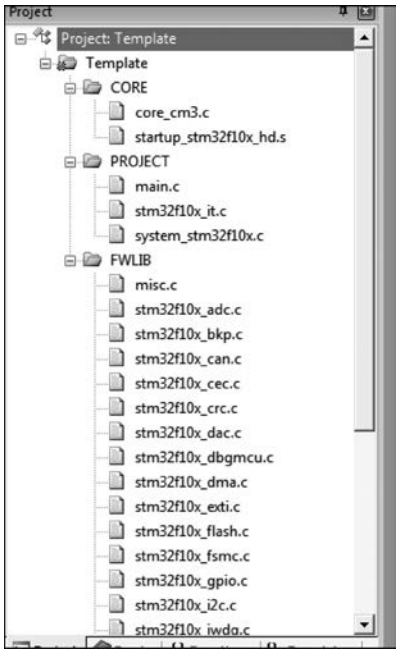

图 3.55　工程结构

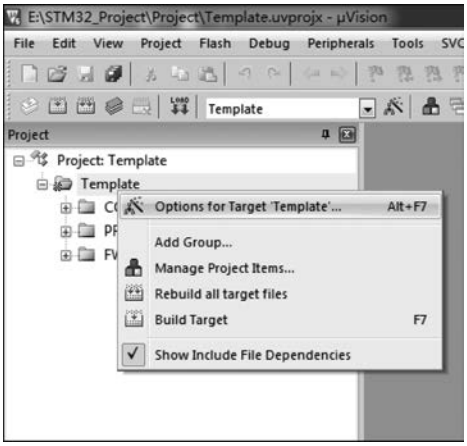

图 3.56　单击 Options for Target 'Template'

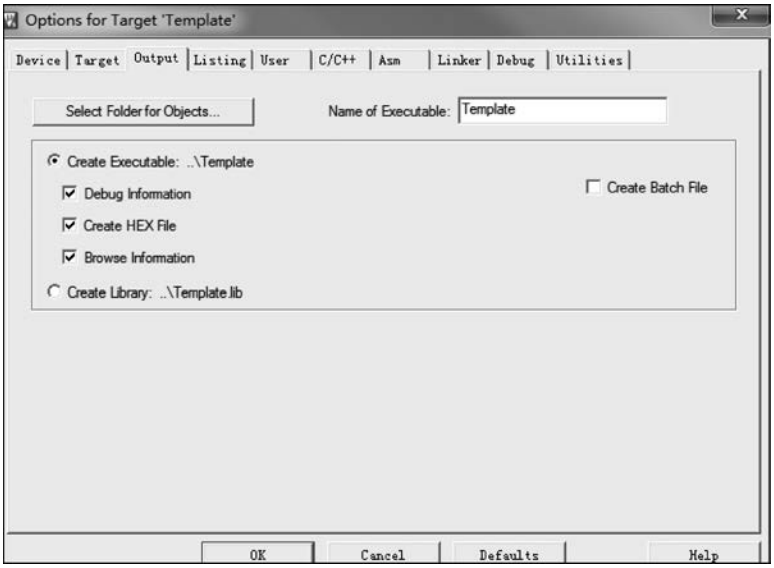

图 3.57　选择编译后的存放目录

(14) 接下来还需要配置工程代码的头文件路径。回到工程主菜单，右击 Template，单击 Options for Target 'Template'。然后选择 C/C++选项卡，单击 Include Paths 右边的“…”按钮，如图 3.58 所示。接着会弹出一个添加路径的对话框，如图 3.59 所示，将图上面的 3 个目录添加进去，最后单击 OK 按钮。

(15) 配置宏定义变量。因为 Version 3.5 版本的库函数在配置和选择外设时是通过

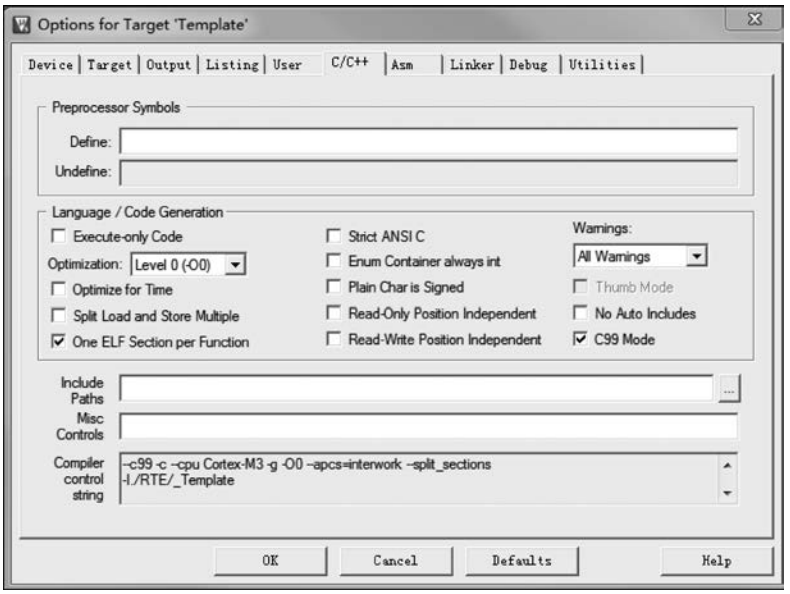

图 3.58　C/C++选项卡

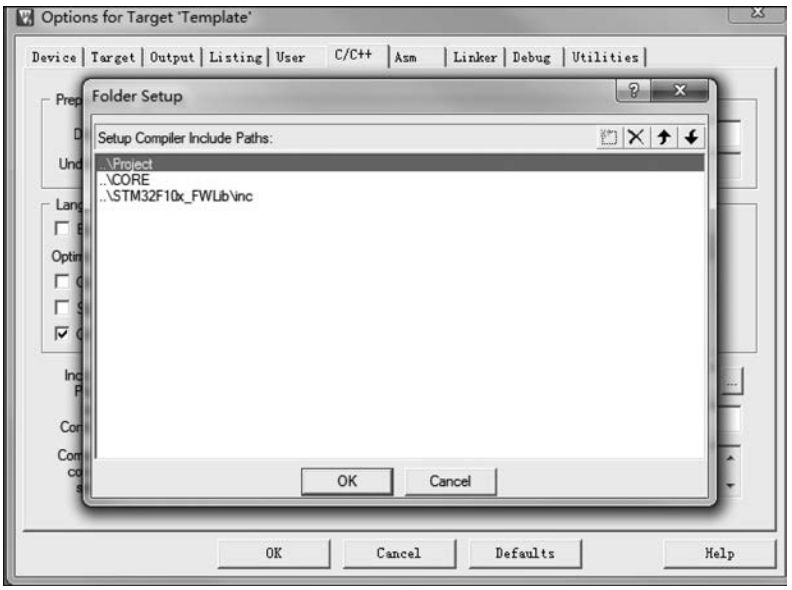

图 3.59　添加头文件路径

宏定义完成的，所以需要配置一个全局的宏定义变量。按照步骤(14)，定位到 C/C++选项卡，然后填写“STM32F10X_HD，USE_STDPERIPH_DRIVER”到 Define 输入框中，如图 3.60 所示。最后单击 OK 按钮。

经过上述的步骤后，工程模板建立完成。下面就可以开始编写代码和调试程序工作了。

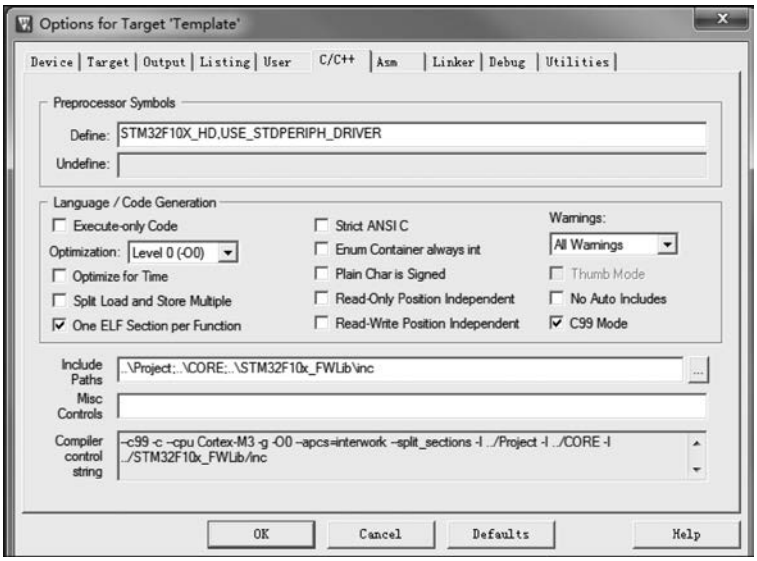

图 3.60　添加全局宏定义标识符到 Define 输入框

7. 编译和下载程序

在进行编译之前，需要打开 Project 下的 main.c 文件，编写好主程序，之后可以进行编译。编译结果如图 3.61 所示，结果显示没有错误。

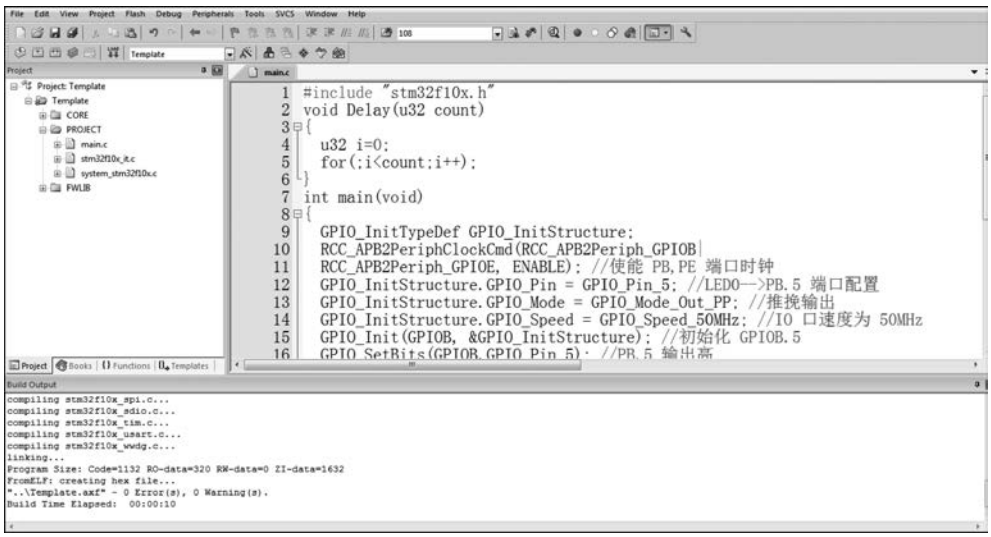

图 3.61　编译代码

工程编译成功以后，就可以下载程序到开发板了。下载程序时，只需要在 FlyMcu 中选择好对应的.hex 文件，并且确保配置正确，然后单击“开始编程”按钮就可以进行下载了。如图 3.62 所示，可以从进度条看到程序下载成功。

运行程序，结果如图 3.63 所示，可控制点亮 12 个 LED 显示灯。

图 3.62　下载程序

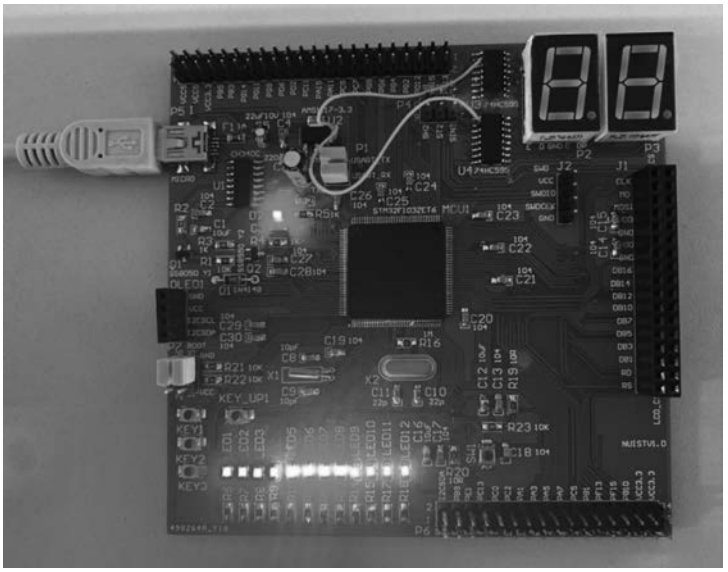

图 3.63　在基础开发实验板上观看程序运行结果