

RISC-V 嵌入式开发实践

—基于 CH32V307 微控制器

王宜怀 杨勇 施连敏 游辉敏 编著

北京航空航天大学出版社

内容简介

本书以沁恒微电子（WCH）的 RSIC-V 内核的 CH32V307 系列微控制器为蓝本、以知识要素为核心、以构件化为基础阐述嵌入式技术基础与实践，同时本书内夹带 AHL-CH32V307 硬件系统，可以满足基础实践需要。全书共 13 章，其中第 1 章简要阐述嵌入式系统的知识体系、学习误区与学习建议；第 2 章给出处理器简介；第 3 章给出 MCU 存储映像、中断源与硬件最小系统。第 4 章以 GPIO 为例给出规范的工程组织框架，阐述底层驱动应用与设计方法；第 5 章阐述嵌入式硬件构件与底层驱动构件基本规范。第 6 章阐述串行通信接口 UART，并给出第一个带中断的实例。1~6 章囊括了学习一个微控制器入门环节的完整要素。7~13 分别给出了 SysTick、Timer、RTC、Flash 在线编程、ADC、DAC、SPI、I2C、TSC、DMA、CAN、USB、以太网及其他模块。第 13 章给出了外接部件、RTOS、嵌入式人工智能等应用案例。

本书提供了电子资源，内含芯片资料、使用文档、硬件说明、源程序等。网上电子资源下载途径：通过百度搜索“苏州大学嵌入式学习社区”官网→“教材”→“AHL-CH32V307”。

本书适用于高等学校嵌入式系统的教学或技术培训，也可供嵌入式系统与物联网应用技术人员作为研发参考。

前 言

嵌入式计算机系统简称为嵌入式系统,其概念最初源于传统测控系统对计算机的需求。随着以微处理器(MPU)为内核的微控制器(MCU)制造技术的不断进步,计算机领域在通用计算机系统与嵌入式计算机系统这两大分支分别得以发展。通用计算机已经在科学计算、通信、日常生活等各个领域产生重要影响。在后PC时代,嵌入式系统的广阔应用是计算机发展的重要特征。一般来说,嵌入式系统的应用范围可以粗略分为两大类:一类是电子系统的智能化(如工业控制、汽车电子、数据采集、测控系统、家用电器、现代农业、嵌入式人工智能及物联网应用等),这类应用也被称为微控制器MCU领域。另一类是计算机应用的延伸(如平板电脑、手机、电子图书等),这类应用也被称为应用处理器MAP领域。不论如何分类,嵌入式系统的技术基础是不变的,即要完成一个嵌入式系统产品的设计,需要有硬件、软件及行业领域相关知识。但是,随着嵌入式系统中软件规模日益增大,对嵌入式底层驱动软件的封装提出了更高的要求,可复用性与可移植性受到特别的关注,嵌入式软硬件构件化开发方法逐步被业界所重视。

2020年以来,RISC-V架构处理器在我国得以快速发展,本书在苏州大学嵌入式团队几十年了积累基础上,基于沁恒微电子的RISC-V架构CH32V307微控制器构建通用嵌入式计算机GEC生态系统,形成了相对完备的教学及实践系统。本书在作者前期撰写的普通高等教育“十一五”、“十二五”国家级规划教材、国家级一流本科课程基础上,以CH32V307微控制器为蓝本重新撰写。同时在南京沁恒微电子及北京航空航天大学出版社的支持下,书中直接配备了可以直接实践的硬件系统AHL-CH32V307,具备简捷、便利、边学边实践等优点,克服了实验箱模式的冗余、不方便带出实验室、不易升级等缺点,逐步探索嵌入式教学新模式。

书中以嵌入式硬件构件及底层软件构件设计为主线,基于嵌入式软件工程的思想,按照“通用知识—驱动构件使用方法—测试实例—构件制作过程”的线条,逐步阐述电子系统智能化嵌入式应用的软件与硬件设计。需要特别说明的是,虽然书籍撰写与教学必须以某一特定芯片为蓝本,但作为嵌入式技术基础,本书试图阐述嵌入式通用知识要素。因此,本书以知识要素为基本立足点设计芯片底层驱动,使得应用程序与芯片无关,具有通用嵌入式计算机(GEC)性质。书中将大部分驱动的使用方法提前阐述,而驱动构件的设计方法后置,目的是先学会使用使用构件进行实际编程,后理解构件的设计方法。因构件设计方法部分有一定难度,对于不同要求的教学场景,也可不要求学生理解全部构件的设计方法,讲解一两个即可。

本书具有以下特点。

(1) 把握通用知识与芯片相关知识之间的平衡。书中对于嵌入式“通用知识”的基本原理,以应用为立足点,进行语言简洁、逻辑清晰的阐述,同时注意与芯片相关知识之间的衔接,使读者在更好地理解基本原理的基础上,理解芯片应用的设计,同时反过来,加深对通用知识的理解。

(2) 把握硬件与软件的关系。嵌入式系统是软件与硬件的综合体，嵌入式系统设计是一个软件、硬件协同设计的工程，不能像通用计算机那样，软件、硬件完全分开来看。特别是对电子系统智能化嵌入式应用来说，没有对硬件的理解就不可能写好嵌入式软件，同样没有对软件的理解也不可能设计好嵌入式硬件。因此，本书注重把握硬件知识与软件知识之间的关系。

(3) 对底层驱动进行构件化封装。书中对每个模块均给出根据嵌入式软件工程基本原则并按照构件化封装要求编制底层驱动程序，同时给出详细、规范的注释及对外接口，为实际应用提供底层构件，方便移植与复用，可以为读者进行实际项目开发节省大量时间。

(4) 设计合理的测试用例。书中所有源程序均经测试通过，并保留测试用例在本书的网上教学资源中，避免了因例程的书写或固有错误给读者带来烦恼。这些测试用例，也为读者验证与理解带来方便。

(5) 网上电子资源提供了所有模块完整的底层驱动构件化封装程序与测试用例。需要使用 PC 机的程序的测试用例，还提供了 PC 机的 C#源程序、芯片资料、使用文档、硬件说明等，网上电子资源的版本将会适时更新。

本书由苏州大学王宜怀担任主编，杨勇、施连敏、游辉敏担任副主编。苏州大学嵌入式系统与物联网研究所的研究生参与程序开发、书稿整理及有关资源建设，他们卓有成效的工作使得本书更加充实。南京沁恒微电子的司云腾、李天培、陶玉凯、陈瑶、刘琪等给予技术支持并校对书稿。苏州大学的刘纯平、赵雷、章晓芳、杨璐、刘晓升等老师，宿迁学院的王志超、陈林、万娟、史洪玮等老师，为本书提出了不少建设性建议，在此一并表示诚挚的感谢。

鉴于作者水平有限，书中难免存在不足和错误之处，恳望读者提出宝贵意见和建议。

苏州大学 王宜怀
2022 年 3 月

硬件资源及网上电子资源

书内夹带的硬件资源（AHL-CH32V307）



网上电子资源：该网上电子资源适时更新，下载路径：百度搜索“苏州大学嵌入式学习社区”官网→“教材”→“AHL-CH32V307”

文件夹		内容
01-Information		内核及芯片文档
02-Document		补充阅读材料、硬件使用说明等
03-Hardware		硬件文档
04-Software	CH01	硬件测试测试程序（含MCU方及PC方程序）
	CH02	认识汇编语句生成的机器码
	CH04	GPIO
	CH06	UART
	CH07	Systick、RTC、Timer、PWM、输入捕捉、输出比较；
	CH08	Flash、ADC、DAC；PC方配套测试程序
	CH09	SPI、I2C、TSC
	CH10	CAN、DMA
	CH11	USB、以太网
	CH11	系统时钟程序的注解、看门狗
	CH13	外接部件、RT-Thread、嵌入式人工智能：物体认知系统
05-Tool	AHL-CH32V307板载TTL-USB芯片芯片驱动程序，C#2019串口测试程序；C#快速应用指南下载导引等	

目 录

第 1 章 概述	1
1.1 初识嵌入式系统	1
1.1.1 运行硬件系统	1
1.1.2 实践体系简介	3
1.1.3 编译、下载与运行第一个嵌入式程序	4
1.2 嵌入式系统的定义、发展简史、分类及特点	6
1.2.1 嵌入式系统的定义	6
1.2.2 嵌入式系统的由来及发展简史	7
1.2.3 嵌入式系统的分类	9
1.2.4 嵌入式系统的特点	10
1.3 嵌入式系统的学习困惑、知识体系及学习建议	11
1.3.1 嵌入式系统的学习困惑	11
1.3.2 嵌入式系统的知识体系	14
1.3.3 基础阶段的学习建议	15
1.4 微控制器与应用处理器简介	17
1.4.1 MCU 简介	17
1.4.2 以 MCU 为核心的嵌入式测控产品的基本组成	18
1.4.3 MAP 简介	19
1.5 嵌入式系统常用术语	20
1.5.1 与硬件相关的术语	21
1.5.2 与通信相关的术语	22
1.5.3 与功能模块相关的术语	23
本章小结	24
习 题	25
第 2 章 RISC-V 架构微处理器	26
2.1 RISC-V 概述与青稞 V4F 微处理器简介	26
2.1.1 RISC-V 概述	26
2.1.2 青稞 V4F 微处理器内部结构概要	28
2.1.3 寄存器通用基础知识及相关基本概念	29
2.1.4 青稞 V4F 微处理器的内部寄存器	30
2.2 寻址方式与机器码的获取方法	31
2.2.1 指令保留字简表与寻址方式	32
2.2.2 机器码的获取方法	33
2.3 RISC-V 基本指令分类解析	35
2.3.1 数据传送类指令	35

2.3.2 数据操作类指令	36
2.3.3 跳转类指令	38
2.3.4 CSR 类指令	39
2.3.5 其他指令	39
2.4 汇编语言的基本语法	40
2.4.1 汇编语言的格式	40
2.4.2 常用伪指令简介	42
本章小结	44
习 题	44
第 3 章 存储器映像、中断源与硬件最小系统	45
3.1 CH32V307 系列 MCU 概述	45
3.1.1 CH32V307 系列 MCU 命名规则	45
3.1.2 CH32V307 存储器映像	46
3.1.3 CH32V307 中断源	47
3.2 CH32V307 的引脚图与硬件最小系统	50
3.2.1 CH32V307 的引脚图	50
3.2.2 CH32V307 硬件最小系统原理图	52
3.3 由 MCU 构建通用嵌入式计算机	54
3.3.1 嵌入式终端开发方式存在的问题与解决办法	55
3.3.2 提出 GEC 概念的时机、GEC 定义与特点	56
3.3.3 由 CH32V307VCT6 构成的 GEC	57
本章小结	62
习 题	63
第 4 章 GPIO 及程序框架	64
4.1 GPIO 通用基础知识	64
4.1.1 GPIO 概念	64
4.1.2 输出引脚的基本接法	64
4.1.3 上拉下拉电阻与输入引脚的基本接法	65
4.2 软件干预硬件的方法	66
4.2.1 GPIO 构件 API	66
4.2.2 第一个 C 语言工程：控制小灯闪烁	68
4.3 认识工程框架	70
4.3.1 工程框架及所含文件简介	70
4.3.2 了解机器码文件及芯片执行流程简析	72
4.4 GPIO 构件的制作过程	74
4.4.1 端口与 GPIO 模块—对外引脚与内部寄存器	75
4.4.2 GPIO 基本编程步骤并点亮一盏小灯	77

4.4.3 GPIO 构件的设计	78
4.5 第一个汇编语言工程：控制小灯闪烁	83
4.5.1 汇编工程文件的组织	84
4.5.2 汇编语言小灯测试工程主程序	85
4.6 实验一 熟悉实验开发环境及 GPIO 编程	89
本章小结	92
习 题	93
第 5 章 嵌入式硬件构件与底层驱动构件基本规范	94
5.1 嵌入式硬件构件	94
5.1.1 嵌入式硬件构件概念与嵌入式硬件构件分类	94
5.1.2 基于嵌入式硬件构件的电路原理图设计简明规则	95
5.2 嵌入式底层驱动构件的概念与层次模型	99
5.2.1 嵌入式底层驱动构件的概念	99
5.2.2 嵌入式硬件构件与软件构件结合的层次模型	99
5.2.3 嵌入式软件构件分类	100
5.3 底层驱动构件的封装规范	101
5.3.1 构件设计的基本思想与基本原则	101
5.3.2 编码风格基本规范	103
5.3.3 头文件的设计规范	107
5.3.4 源程序文件的设计规范	109
5.4 硬件构件及其驱动构件的复用与移植方法	110
5.4.1 硬件构件的复用与移植	110
5.4.2 驱动构件的移植	111
本章小结	112
习 题	113
第 6 章 串行通信模块及第一个中断程序结构	114
6.1 异步串行通信的通用基础知识	114
6.1.1 串行通信的基本概念	114
6.1.2 RS232 和 RS485 总线标准	116
6.1.3 TTL-USB 串口	117
6.1.4 串行通信编程模型	118
6.2 基于构件的串行通信编程方法	119
6.2.1 CH32V307VCT6 芯片 UART 对外引脚	119
6.2.2 UART 构件 API	120
6.2.3 UART 构件 API 的发送测试方法	122
6.2.4 printf 的设置方法与使用	122
6.3 UART 构件的制作过程	123

6.3.1 UART 寄存器概述.....	123
6.3.2 利用直接地址操作的串口发送打通程序.....	124
6.3.3 UART 构件设计.....	127
6.4 中断机制及中断编程步骤.....	130
6.4.1 关于中断的通用基础知识.....	131
6.4.2 RISC-V 非内核模块中断编程结构.....	133
6.4.3 CH32V307VCT6 中断编程步骤——以串口接收中断为例.....	136
6.5 实验二 串口通信及中断实验.....	138
本章小结.....	140
习 题.....	141
第 7 章 定时器相关模块.....	142
7.1 定时器通用基础知识.....	142
7.2 CH32V307VCT6 中的定时器.....	142
7.2.1 青稞 V4F 内核中系统定时器 SysTick.....	143
7.2.2 实时时钟 RTC 模块.....	147
7.2.3 Timer 模块的基本定时功能.....	150
7.3 脉宽调制.....	152
7.3.1 脉宽调制 PWM 通用基础知识.....	152
7.3.2 基于构件的 PWM 编程方法.....	155
7.3.3 脉宽调制构件的制作过程.....	158
7.4 输入捕捉与输出比较.....	162
7.4.1 输入捕捉与输出比较通用基础知识.....	162
7.4.2 基于构件的输入捕捉和输出比较编程方法.....	163
7.5 实验三 定时器及 PWM 实验.....	167
本章小结.....	168
习 题.....	168
第 8 章 Flash 在线编程、ADC 与 DAC.....	170
8.1 Flash 在线编程.....	170
8.1.1 Flash 在线编程的通用基础知识.....	170
8.1.2 基于构件的 Flash 在线编程方法.....	171
8.1.3 Flash 构件的制作过程.....	173
8.2 ADC.....	176
8.2.1 ADC 的通用基础知识.....	176
8.2.2 基于构件的 ADC 编程方法.....	180
8.2.3 ADC 构件的制作过程.....	182
8.3 DAC.....	185
8.3.1 DAC 的通用基础知识.....	185

8.3.2 基于构件的 DAC 编程方法	186
8.4 实验四 ADC 实验	187
本章小结	189
习 题	189
第 9 章 SPI、I2C 与 TSC 模块	190
9.1 串行外设接口 SPI 模块	190
9.1.1 串行外设接口 SPI 的通用基础知识	190
9.1.2 基于构件的 SPI 通信编程方法	193
9.2 集成电路互联总线 I2C 模块	196
9.2.1 集成电路互联总线 I2C 的通用基础知识	196
9.2.2 基于构件的 I2C 通信编程方法	202
9.3 触摸感应控制器 TSC 模块	206
9.3.1 触摸感应控制器 TSC 的基本原理	206
9.3.2 基于构件的 TSC 编程方法	207
9.4 实验五 SPI 通信实验	210
本章小结	211
习 题	212
第 10 章 DMA 与 CAN 总线编程	213
10.1 CAN 总线	213
10.1.1 CAN 总线的通用基础知识	213
10.1.2 基于构件的 CAN 编程方法	216
10.2 DMA	219
10.2.1 DMA 的通用基础知识	219
10.2.2 基于构件的 DMA 编程方法	220
本章小结	223
习 题	223
第 11 章 USB 与嵌入式以太网模块	224
11.1 USB 通用基础知识	224
11.1.1 USB 概述	224
11.1.2 USB 相关基本概念	227
11.1.3 USB 通信协议	232
11.2.4 从设备的枚举看 USB 数据传输	236
11.2 CH32V307 的 USB 模块应用编程方法	241
11.2.1 CH32V307 的 USB 模块简介	241
11.2.2 CH32V307 作为 USB 从机的编程方法	242
11.2.3 CH32V307 作为 USB 主机的编程方法	249

11.3 嵌入式以太网通用基础知识.....	252
11.3.1 以太网的由来与协议模型.....	252
11.3.2 以太网中主要物理设备.....	256
11.3.3 相关名词解释	257
11.4 CH32V307 的以太网模块应用编程方法	261
11.4.1 CH32V307 的以太网模块简介	261
11.4.2 以太网底层驱动构件.....	263
11.4.3 以太网测试实例	264
11.5 本章小结	280
习 题	281
第 12 章 系统时钟与其他功能模块	282
12.1 时钟系统	282
12.1.1 时钟系统概述	282
12.1.2 时钟模块寄存器概要	283
12.1.3 时钟模块编程实例	285
12.2 电源模块与复位模块	287
12.2.1 电源模块	287
12.2.2 复位模块	288
12.3 看门狗.....	289
12.3.1 独立看门狗	289
12.3.2 系统窗口看门狗	291
12.4 数字视频接口与安全数字输入输出	292
12.4.1 数字视频接口	292
12.4.2 安全数字输入输出	293
本章小结	294
习 题	295
第 13 章 应用案例	296
13.1 嵌入式系统稳定性问题	296
13.2 外接传感器及执行部件的编程方法	297
13.2.1 开关量输出类驱动构件	297
13.2.2 开关量输入类驱动构件	300
13.2.3 声音与加速度传感器驱动构件	302
13.3 实时操作系统的简明实例	303
13.3.1 无操作系统与实时操作系统	303
13.3.2 RTOS 中的常用基本概念.....	304
13.3.3 线程的三要素、四种状态及三种基本形式.....	305
13.3.4 RTOS 下编程实例	308

13.4 嵌入式人工智能的简明实例	309
13.4.1 EORS 简介	310
13.4.2 AHL-EORS 的数据采集与训练过程	311
13.4.3 在通用嵌入式计算机 GEC 上进行的推理过程	313
13.5 沁恒 MCU 的其他嵌入式实践资源简介	314
13.5.1 AHL-CH573	314
13.5.2 AHL-CH573-NB-IoT	315
13.5.3 AHL-CH573-CAT1	316
参考文献	318

第1章 概述

本章导读：由于本书配有可实践的硬件体系，作为全书导引，本章首先从运行第一个嵌入式程序开始，使读者直观认识到嵌入式系统就是一个实实在在的微型计算机；接着阐述嵌入式系统的基本概念、由来、发展简史、分类及特点；给出嵌入式系统的学习困惑、知识体系及学习建议；随后给出微控制器与应用处理器简介；最后简要归纳嵌入式系统的常用术语，以便对嵌入式系统的基本词汇有初步认识，为后续内容的学习提供基础。补充阅读材料中简要总结了嵌入式系统常用的 C 语言基本语法概要，以便快速收拢本书所用到的 C 语言知识要素。

1.1 初识嵌入式系统

嵌入式系统，即嵌入式计算机系统的简称，它不仅具有通用计算机的主要特点，还具有自身特点。嵌入式系统不单独以通用计算机面目出现，而是隐含在各类具体的智能产品中，如手机、机器人、自动驾驶系统等。嵌入式系统在嵌入式人工智能、物联网、工厂智能化等产品中起核心作用。

由于嵌入式系统是一门理论与实践密切结合的课程，为了使读者能够更好、更快地学习嵌入式系统，本书随附了苏州大学嵌入式系统与物联网研究所开发的 RISC-V 架构的 AHL-CH32V307 嵌入式开发套件。下面就以运行这个小小的微型计算机为起点，开启嵌入式系统的学习之旅。

1.1.1 运行硬件系统



图1-1 AHL-CH32V307嵌入式开发套件

1. 了解实践硬件

图 1-1 为本书随附的 AHL-CH32V307 嵌入式开发套件，它由主板与一根标准的 Type-C 数据线^①组成，具体内容如表 1-1 所示。

表1-1 AHL-CH32V307嵌入式开发套件

资源类型	名称	数量	备 注
硬件	AHL-CH32V307	1套	① 板载微控制器：沁恒微电子CH32V307VCT6； ② 5V转3.3V电源、红绿蓝三色灯、复位按键等； ③ 两路USB转TTL串口，通过Type-C接口与PC机相连，供程序下载调试及用户串口使用； ④ 引出芯片的所有对外接口引脚，如GPIO、UART、ADC、SPI、PWM、CAN、USB、以太网等； ⑤ 提供一路默认的温度传感器（热敏电阻），可测量环境温度。
	Type-C数据线	1根	标准Type-C数据线，取电与串口通信使用
软件及文档	电子资源：苏州大学嵌入式学习社区→教材→AHL-CH32V307		

AHL-CH32V307 是一个典型的嵌入式系统，虽然体积很小，但它包含了微型计算机的基本要素，也就是俗话所说的，麻雀虽小五脏俱全，可以充分利用这个套件的硬件、软件、文档、开发环境等资源，较好地服务于嵌入式系统入门阶段的学习。

2. 测试实践硬件

出厂时已经将电子资源中的“..\04-Software\CH01 文件夹”下的测试程序灌入这个嵌入式计算机内，只要给它供电，其中的程序就可以运行了，步骤如下。

步骤 1：使用 Type-C 数据线给主板供电。将 Type-C 数据线的小端连接主板，另外一端接工具机^②的 USB 接口。

步骤 2：观察程序运行效果。现象如下：红、绿、蓝各灯每 5s、10s、20s 状态变化，对外表现为三色灯的合成色，其实际效果如图 1-2 所示。即开始时为暗，依次变化为红、绿、黄（红+绿）、蓝、紫（红+蓝）、青（蓝+绿）、白（红+蓝+绿），周而复始。

^① Type-C 数据线是 2014 年面市的基于 USB3.1 标准接口的数据线，没有正反方向的区别，可承受 1 万次反复插拔。

^② 工具机可以是笔记本电脑、个人计算机 PC 等。

合成色				红 + 绿		红 + 蓝	蓝 + 绿	红 + 蓝 + 绿				红 + 绿		红 + 蓝	蓝 + 绿	红 + 蓝 + 绿				红 + 绿
	暗	红	绿	黄	蓝	紫	青	白	暗	红	绿	黄	蓝	紫	青	白	暗	红	绿	黄
蓝灯																				
绿灯																				
红灯																				
	0	5	10	15	20	25	30	35	40	45	50	55	60	65	70	75	80	85	90	95

图1-2 三色灯实际效果

从运行效果可以体会到这小小的嵌入式计算机的功能。实际上，该嵌入式计算机的功能十分丰富，通过编程可以完成智能化领域的许多重要任务，本书将由此带领读者逐步进入嵌入式系统的广阔天地。

接下来，尝试自己下载一个程序到嵌入式计算机中运行，首先需要安装集成开发环境。

1.1.2 实践体系简介

AHL-CH32V307 名称中的首部 AHL 三个字母是“Auhulu”的缩写，中文名字为“金葫芦”，英文名字为“Auhulu”，其含义是“照葫芦画瓢^①”。该开发套件，与一般的嵌入式系统实验箱不同，不仅可以作为嵌入式系统教学使用，还是一套较为完备的嵌入式微型计算机应用开发系统。

AHL-CH32V307 嵌入式开发套件由硬件部分、软件部分、教学资源 3 个部分组成。

1. 硬件部分

AHL-CH32V307 以沁恒微电子的 CH32V307VCT6 微控制器为核心，辅以硬件最小系统，集成红、绿、蓝三色灯，复位按钮，二路 TTL-USB 串口，外接 Type-C 线，从而形成完

^① 照葫芦画瓢：比喻照着样子模仿，简单易行，出自宋·魏泰《东轩笔录》第一卷。古希腊哲学家亚里士多德则说过：“人从儿童时期起就有模仿本能，他们用模仿而获得了最初的知识，模仿就是学习”。孟子则曰：“大匠诲人必以规矩，学者亦必以规矩”，其含义是说高明的工匠教人手艺必定依照一定的规矩，而学习的人也就必定依照一定的规矩。本书借此，期望通过建立符合软件工程基本原理的“葫芦”，为“照葫芦画瓢”提供坚实基础，达到降低学习难度之目标。

整的通用嵌入式计算机（General Embedded Computer，GEC），配合本书及电子资源中的补充阅读材料，可以使读者方便地进行嵌入式系统的学习与开发。随书所附套件为基础型，可以完成本书 90% 的实验。为了满足学校实验室建设要求，还制作了增强型套件，增加了 9 个外接组件，包括声音传感器、加速度传感器、人体红外传感器、循迹传感器、振动马达、蜂鸣器、四按钮模块、彩灯及数码管等，可完成本书所有实验。亦可适用通过主板上的开放式外围引脚外接其他接口模块进行创新性实验。增强型的包装分为盒装式与箱装式，盒装式便于携带，学生可借出实验室，箱装式主要供学生在实验室进行实验。

2. 集成开发环境

嵌入式软件开发有别于个人计算机（Personal Computer，PC）软件开发的一个显著的特点在于：它需要一个交叉编译和调试环境，即工程的编辑和编译所使用的工具软件通常在 PC 上运行，这个工具软件通常称为集成开发环境（Integrated Development Environment，IDE），而编译生成的嵌入式软件的机器码文件则需要通过写入工具下载到目标机上执行。这里的工具机就是人们通常使用的台式个人计算机或笔记本式个人计算机。本书的目标机就是随书所附的 AHL-CH32V307 开发套件。

本书使用的集成开发环境为苏州大学研发的 AHL-GEC-IDE，具有编辑、编译、链接等功能，特别是配合“金葫芦”硬件，可直接运行、调试程序，根据芯片型号不同兼容常用嵌入式集成开发环境。注意：PC 的操作系统需要使用 Windows 10 版本。

针对 AHL-CH32V307 工程，AHL-GEC-IDE 在编辑编译方面，兼容沁恒微电子提供的集成开发环境 MounRiver Studio（MRS）。

3. 下载安装IDE及获得本书的电子资源

（1）下载安装 IDE。可以通过百度搜索“苏州大学嵌入式学习社区”官网下载。AHL-GEC-IDE 下载路径：金葫芦专区→AHL-GEC-IDE。下载后，Windows 10 下安装该开发环境。

（2）获得本书的电子资源。获得路径：教材→AHL-CH32V307。电子资源中包含了芯片资料、开发套件用户手册、补充阅读材料、硬件说明、源程序、硬件测试程序、常用软件工具等。

1.1.3 编译、下载与运行第一个嵌入式程序

在正确安装 AHL-GEC-IDE 及获得本书电子资源的前提下，可以进行第一个嵌入式程序编译、下载与运行，以便直观体会嵌入式程序的运行。

步骤 1：硬件接线。将 Type-C 数据线的小端连接主板的 Type-C 接口，另外一端接通用计算机的 USB 接口。

步骤 2：打开环境，导入工程。打开集成开发环境 AHL-GEC-IDE，单击菜单“文件”→“导入工程”，随后选择电子资源中“..\04-Software\CH01\AHL-CH32V307-Test”

（文件夹名就是工程名。**注意：**路径中不能包含汉字，也不能太深）。导入工程后，左侧为

工程树形目录，右侧为文件内容编辑区，初始显示 main.c 文件内容，如图 1-3 所示。

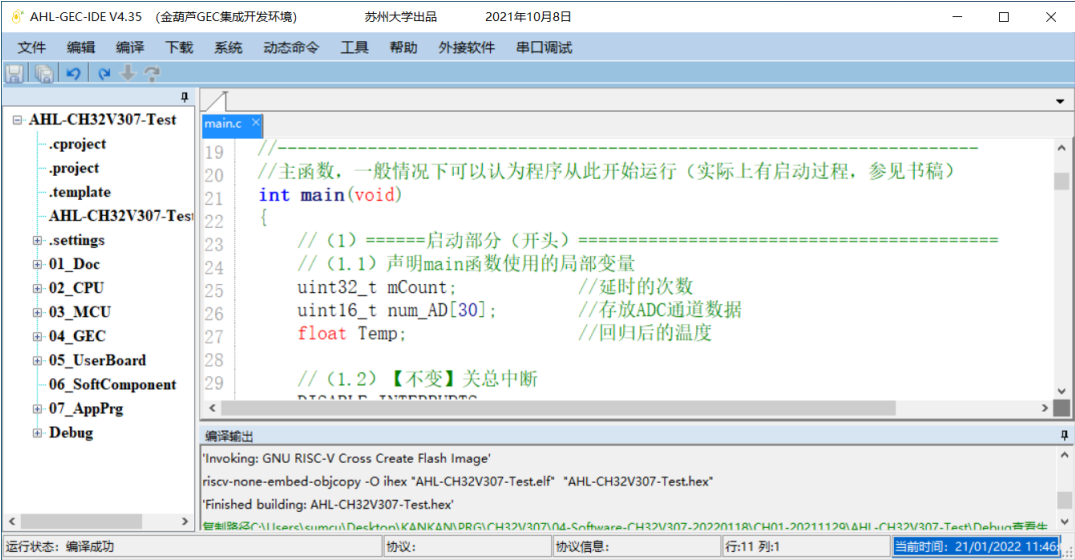


图1-3 IDE界面及编译结果

步骤 3：编译工程。单击菜单“编译”→“编译工程”，就开始编译。正常情况下，编译后会显示“编译成功!”。

步骤 4：连接 GEC。单击菜单“下载”→“串口更新”，将进入更新窗体界面。单击“连接 GEC”查找目标 GEC，若提示“成功连接……”，可进行下一步操作。若连接不成功，则可参阅电子资源中“..\\02-Document”文件夹内的快速指南文档中的“常见问题及解决办法”一节进行解决。

步骤 5：下载机器码。单击“选择文件”按钮，导入被编译工程目录下 Debug 中的.hex 文件，然后单击“一键自动更新”按钮，等待程序自动更新完成。当更新完成之后，程序将自动运行。

步骤 6：观察运行结果。与 1.1.1 节一致，这就是出厂时灌入的程序。

步骤 7：通过串口观察运行情况。① 观察程序运行过程。在 IDE 的顶部菜单栏目，单击“工具”→“串口工具”，选择其中一个串口，波特率设为 115200 并打开，串口调试工具页面会显示三色灯的状态、温度等信息；② 验证串口收发。关闭已经打开的串口，打开另一个串口，波特率选择默认参数，在“发送数据框”中输入字符串，单击“发送数据”按钮。正常情况下，主板会回送数据给 PC，并在接收框中显示，效果如图 1-4 所示。

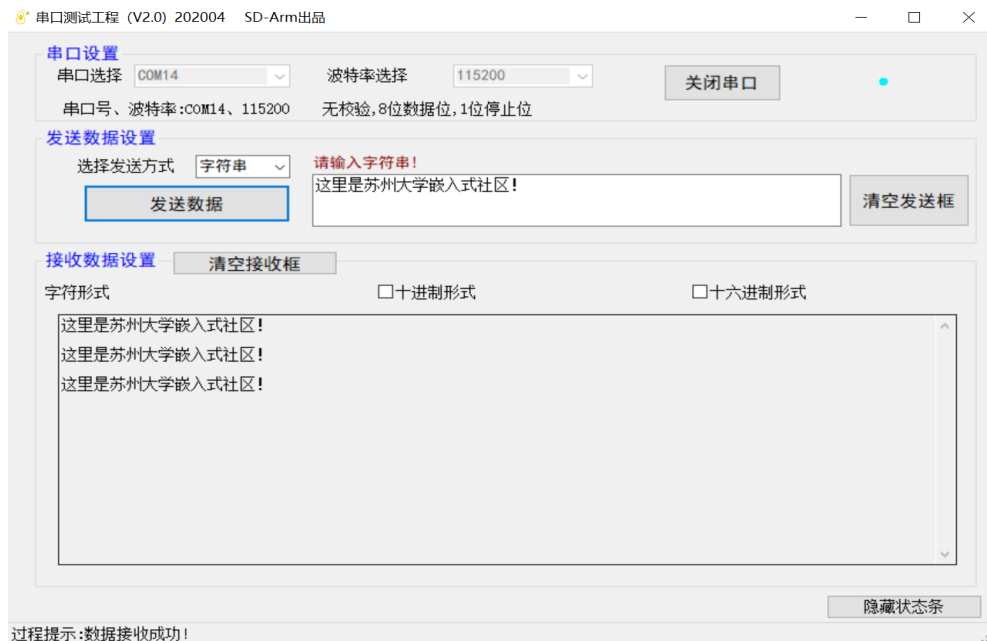


图1-4 IDE内嵌的串口调试工具

若 PC 机安装了 Visual Studio 2019（VS2019）开发环境，还可以运行“..\04-Software\CH01\C#程序(For AHL-CH32V307-Test)”工程，与 MCU 进行串口连接获得温度图示化显示、三色灯图示化颜色变化、语音播报灯体验。

有了这些初步的直观体验，下面开始进入嵌入式系统的学习之旅，首先了解嵌入式系统的定义、发展简史、分类及特点。

1.2 嵌入式系统的定义、发展简史、分类及特点

1.2.1 嵌入式系统的定义

嵌入式系统（Embedded System）有多种多样的定义，但本质是相同的。这里给出美国 CMP Books 出版的 Jack Ganssle 和 Michael Barr 的著作 *Embedded System Dictionary*^① 给出的嵌入式系统的定义：嵌入式系统是一种计算机硬件和软件的组合，也许还有机械装置，用于实现一个特定功能。在某些特定情况下，嵌入式系统是一个大系统或产品的一部分。该词典还给出了嵌入式系统的一些示例，如微波炉、手持电话、数字手表、巡航导弹、全球定位系统接收机、数码相机、遥控器等，难以尽数。通过与通用计算机的对比可以更形象地理解嵌入式系统的定义。该词典给出的通用计算机定义是：计算机硬件和软件

^① Jack Ganssle, Michael Barr. 英汉双解嵌入式系统词典[M]. 马广云, 潘琢金, 彭甫阳, 译. 北京: 北京航空航天大学出版社, 2006 年.

的组合，用作通用计算平台。个人计算机是最流行的现代计算机。

下面将列举其他文献给出的定义，以便读者了解对嵌入式系统定义的不同表述方式，也可看做从不同角度定义嵌入式系统。

中国《国家标准 GB/T 22033 2008 信息技术 嵌入式系统术语》给出的嵌入式系统定义：**嵌入式系统置入应用对象内部起信息处理和控制作用的专用计算机系统**。它是以应用为中心，以计算技术为基础，软件硬件可剪裁，对功能、可靠性、成本、体积、功耗有严格约束的专用计算机系统，其硬件至少包含一个微控制器或微处理器。

美国电气电子工程师学会（Institute of Electrical and Electronics Engineers, IEEE）给出的嵌入式系统定义：嵌入式系统是控制、监视或者辅助装置、机器和设备运行的装置。

维基百科（英文版）给出的嵌入式系统定义：嵌入式系统是一种用计算机控制的、具有特定功能的、较小的机械或电气系统，且经常有实时性的限制，在被嵌入整个系统中时一般包含硬件部件和机械部件。现如今，嵌入式系统控制了人们日常生活中的许多设备，98%的微处理器被用在了嵌入式系统中。

国内对嵌入式系统的定义曾进行过广泛讨论，有许多不同说法。其中，嵌入式系统定义的涵盖面问题是主要争论焦点之一。例如，有的学者认为不能把手持电话叫嵌入式系统，而只能把其中起控制作用的部分叫嵌入式系统，而手持电话可以称为嵌入式系统的应用产品。其实，这些并不妨碍人们对嵌入式系统的理解，因此不必对定义感到困惑。有些国内学者特别指出，在理解嵌入式系统定义时，不要把嵌入式系统与嵌入式系统产品相混淆。实际上，从口语或书面语言角度，不区分“嵌入式系统”与“嵌入式系统产品”，只要不妨碍对嵌入式系统的理解就没有关系。

总地说来，可以从计算机本身的角度概括表述嵌入式系统。嵌入式系统，即嵌入式计算机系统，它是不以计算机面目出现的“计算机”，这个计算机系统隐含在各类具体的产品之中，这些产品中的计算机程序起到了重要作用。

1.2.2 嵌入式系统的由来及发展简史

1. 嵌入式系统的由来

通俗地说，计算机是因科学家需要一个高速的计算工具而产生的。直到 20 世纪 70 年代，电子计算机在数字计算、逻辑推理及信息处理等方面表现出非凡的能力。而在通信、测控与数据传输等领域，人们对计算机技术给予了更大的期待。这些领域的应用与单纯的高速计算要求不同，主要表现在：直接面向控制对象；嵌入具体的应用产品中，而非以计算机的面貌出现；能在现场连续可靠地运行；体积小，应用灵活；突出控制功能，特别是对外部信息的捕捉与丰富的输入输出功能等。由此可以看出，满足这些要求的计算机与满足高速数值计算的计算机是不同的。因此，一种称之为微控制器（单片机）^①的技术得以产生并发展。为了区分这两种计算机类型，通常把满足海量高速数值计算的计算机称为**通用计算机系统**，

^①微控制器与单片机这两个术语的语义是基本一致的，本书后面除讲述历史之外，一律使用微控制器一词。

而把嵌入实际应用系统中,实现嵌入式应用的计算机称为**嵌入式计算机系统**,简称嵌入式系统。可以说,是因为通信、测控与数据传输等领域对计算机技术的需求催生了嵌入式系统的产生。

2. 嵌入式系统的发展简史

1946 年,世界上第一台电子数字积分计算机(The Electronic Numerical Integrator And Calculator, ENIAC)。它由美国宾夕法尼亚大学莫尔电工学院制造,重达 30t,总体积约 90m³,占地 170m²,耗电 140kw/h,运算速度为每秒 5000 次加法,标志着计算机时代开始。其中,最重要的部件是中央处理器(central processing unit, CPU),它是一台计算机的运算和控制核心。CPU 的主要功能是解释指令和处理数据,其内部含有运算逻辑部件,即算术逻辑运算单元(arithmetic logic unit, ALU)、寄存器部件和控制部件等。

1971 年,Intel 公司推出了单芯片 4004 微处理器(micro processor unit, MPU),它是世界上第一个商用微处理器,Busicom 公司就是用它制作电子计算器的,这就是嵌入式计算机的雏形。1976 年,Intel 公司又推出了 MCS-48 单片机(single chip microcomputer, SCM),这个内部含有 1KB 只读存储器(read only memory, ROM),64B 随机存取存储器(random access memory, RAM)的简单芯片成为世界上第一个单片机,开创了将 ROM、RAM、定时器、并行口、串行口及其他各种功能模块等 CPU 外部资源,与 CPU 一起集成到一个硅片上生产的时代。1980 年,Intel 公司对 MCS-48 单片机进行了完善,推出了 8 位 MCS-51 单片机,并获得巨大成功,开启了嵌入式系统的单片机应用模式。至今,MCS-51 单片机仍有较多应用。这类系统大部分应用于一些简单、专业性强的工业控制系统中,早期主要使用汇编语言编程,后来大部分使用 C 语言编程,一般没有操作系统的支持。

20 世纪 80 年代,逐步出现了 16 位、32 位微控制器(micro controller unit, MCU)。1984 年,Intel 公司推出了 16 位 8096 系列。并将其称为嵌入式微控制器,这可能是“嵌入式”一词第一次在微处理机领域出现。这个时期,Motorola、Intel、TI、NXP、Atmel、Microchip、Hitachi、Philips、ST 等公司推出了不少微控制器产品,功能也不断变强,也逐步支持了实时操作系统。

20 世纪 90 年代开始,数字信号处理器(digital signal processing, DSP)、片上系统(system on chip, SoC)得到了快速发展。嵌入式处理器扩展方式从并行总线型发展出各种串行总线,并被工业界所接受,形成了一些工业标准,如集成电路互联(inter integrated circiut, I2C)总线、串行外设接口(serial peripheral interface, SPI)总线。甚至将网络协议的低两层或低三层都集中到嵌入式处理器上,如某些嵌入式处理器集成了(contral area network, CAN)总线接口、以太网接口。随着超大规模集成电路技术的发展,将数字信号处理器、精简指令集计算机、存储器、I/O、半定制电路集成到单芯片的产品 SoC 中。值得一提的是,ARM 微处理器的出现,较快地促进了嵌入式系统的发展。而 RISC-V 架构在极短的时间内便引起了业界的高度关注,迅速在全世界范围内的兴起和风靡,在国内也引起了广泛的关注,同时给嵌入式系统的发展注入了新鲜的血液。

21 世纪开始以来,嵌入式系统芯片制造技术快速发展,融合了以太网与无线射频技术,成为物联网(Internet of Things, IoT)的关键技术基础。嵌入式系统发展的目标应该是实现

信息世界和物理世界的完全融合，构建一个可控、可信、可扩展并且安全高效的信息物理系统（Cyber-Physical Systems, CPS），从根本上改变人类构建工程物理系统的方式。此时的嵌入式设备不仅要具备个体智能（computation, 计算）、交流智能（communication, 通信），还要具备在交流中的影响和响应能力（control, 控制与被控），实现“智慧化”。显然，今后嵌入式系统研究要与网络和高性能计算的研究更紧密地结合。

在嵌入式系统的发展历程中，RISC-V 架构处理器已经具备了替代传统商用嵌入式处理器（譬如 ARM Cortex-M 处理器）的能力。但是由于 RISC-V 诞生时间较短，在很多方面还需要系统而详实的文献资料来帮助初学者快速掌握这一门新兴的处理器架构，本书以 RISC-V 处理器为蓝本阐述嵌入式应用，有助跟踪这一新的发展，有关 RISC-V 的来龙去脉将在第 2 章中阐述。

1.2.3 嵌入式系统的分类

嵌入式系统的分类标准有很多，有的按照处理器位数来分，有的按照复杂程度来分，还有的按其他标准来分，这些分类方法各有特点。从嵌入式系统的学习角度来看，因为应用于不同领域的嵌入式系统，其知识要素与学习方法有所不同，所以可以按应用范围简单地把嵌入式系统分为电子系统智能化（微控制器类）和计算机应用延伸（应用处理器）这两大类。一般来说，微控制器与应用处理器的主要区别在于可靠性、数据处理量、工作频率等方面，相对应用处理器来说，微控制器的可靠性要求更高、数据处理量较小、工作频率较低。

1. 电子系统智能化类（微控制器类）

电子系统智能化类的嵌入式系统，主要用于工业控制、现代农业、家用电器、汽车电子、测控系统、数据采集等，这类应用所使用的嵌入式处理器一般被称为微控制器。这类嵌入式系统产品，从形态上看，更类似于早期的电子系统，但内部计算程序起核心控制作用。如电机控制器、工业监控设备、网络设备、涵养农业系统、智能气象系统、水质监测系统、汽车电子等。从学习与开发角度，电子系统智能化类的嵌入式应用，需要终端产品开发者面向应用对象设计硬件、软件，注重软件、硬件的协同开发。因此，开发者必须掌握底层硬件接口、底层驱动及软硬件密切结合的开发调试技能。电子系统智能化类的嵌入式系统，即微控制器，是嵌入式系统的软硬件基础，是学习嵌入式系统的入门环节，且为重要的一环。从操作系统角度看，电子系统智能化类的嵌入式系统，可以不使用操作系统，也可以根据复杂程度及芯片资源的容纳程度，使用操作系统。电子系统智能化类的嵌入式系统使用的操作系统通常是实时操作系统（Real Time Operating System, RTOS），如 RT-Thread、mbedOS、MQX Lite、FreeRTOS、μCOS-III、μCLinux、VxWorks 和 eCos 等。

2. 计算机应用延伸类（应用处理器类）

计算机应用延伸类的嵌入式系统，主要用于平板电脑、智能手机、电视机顶盒、企业网络设备等，这类应用所使用的嵌入式处理器一般被称为应用处理器（application processor），一般也称为多媒体应用处理器（Multimedia Application Processor, MAP）。这类嵌入式系统产品，从形态上看，更接近通用计算机系统。从开发方式上看，也类似于通用计算机软件

开发方式。从学习与开发角度看，计算机应用延伸类的嵌入式应用，终端产品开发者大多购买厂商制作好的硬件实体在嵌入式操作系统下进行软件开发，或者还需要掌握少量的对外接口方式。因此，从知识结构角度看，学习这类嵌入式系统，对硬件的要求相对较少。计算机应用延伸类的嵌入式系统，即应用处理器，也是嵌入式系统学习中重要的一环。但是，从学习规律角度看，若是要全面学习掌握嵌入式系统，应该先学习掌握微控制器，然后在此基础上，进一步学习掌握应用处理器编程，而不要倒过来学习。从操作系统角度看，计算机应用延伸类的嵌入式系统一般使用非实时嵌入式操作系统，通常称为嵌入式操作系统（Embedded Operation System, EOS），如 Android、Linux、iOS、WindowsCE 等。当然，非实时嵌入式操作系统与实时操作系统也不是明确划分的，只是粗略分类，侧重有所不同而已。现在的 RTOS 的功能也在不断提升，一般的嵌入式操作系统也在提高实时性。

当然，工业生产车间经常看到利用工业控制计算机、个人计算机（PC）控制机床、生产过程等，这些可以说是嵌入式系统的一种形态。因为它们完成特定的功能，且整个系统不被称之为计算机，而是另有名称，如磨具机床、加工平台等。但是，从知识要素角度看，这类嵌入式系统不具备普适意义，本书不讨论这类嵌入式系统。

1.2.4 嵌入式系统的特点

嵌入式系统对不同学者也许有不同的说法，这里从与通用计算机对比的角度来介绍嵌入式系统的特点。

与通用计算机系统相比，嵌入式系统的存储资源相对匮乏、速度较低，对实时性、可靠性、知识综合要求较高。嵌入式系统的开发方法、开发难度、开发手段等，均不同于通用计算机程序，也不同于常规的电子产品。嵌入式系统是在通用计算机发展基础上，面向测控系统逐步发展起来的。因此，从与通用计算机对比的角度来认识嵌入式系统的特点，对学习嵌入式系统具有实际意义。

1. 嵌入式系统属于计算机系统，但不单独以通用计算机的面目出现

嵌入式系统它不仅具有通用计算机的主要特点，又具有自身特点。嵌入式系统也必须要软件才能运行，但其隐含在种类众多的具体产品中。同时，通用计算机种类屈指可数，而嵌入式系统不仅芯片种类繁多，而且由于应用对象大小各异，嵌入式系统作为控制核心，已经融入各个行业的产品之中。

2. 嵌入式系统开发需要专用工具和特殊方法

嵌入式系统不像通用计算机那样，有了计算机系统就可以进行应用软件的开发。一般情况下，微控制器或应用处理器的芯片本身不具备开发功能，必须要有一套与相应芯片配套的开发工具和开发环境。这些开发工具和开发环境一般基于通用计算机上的软硬件设备，以及逻辑分析仪、示波器等。开发过程中往往有工具机（一般为 PC 或笔记本电脑）和目标机（实际产品所使用的芯片）之分，工具机用于程序的开发，目标机作为程序的执行机，开发时需要交替结合进行。编辑、编译、链接生成机器码在工具机完成，通过写入调试器将机器码下载到目标机中，进行运行与调试。

3. 使用MCU设计嵌入式系统，数据与程序空间采用不同存储介质

在通用计算机系统中，程序存储在硬盘上。实际运行时，通过操作系统将要运行的程序从硬盘调入内存（RAM），运行中的程序、常数、变量均在 RAM 中。一般情况下，以 MCU 为核心的嵌入式系统中，其程序被固化到非易失性存储器^①中。变量及堆栈使用 RAM 存储器。

4. 开发嵌入式系统涉及软件、硬件及应用领域的知识

嵌入式系统与硬件紧密相关，嵌入式系统的开发需要硬件和软件的协同设计、协同测试。同时，由于嵌入式系统专用性很强，通常是用在特定应用领域，如嵌入在手机、冰箱、空调、各种机械设备、智能仪器仪表中，起核心控制作用，且功能专用。因此，进行嵌入式系统的开发，还需要对领域知识有一定的理解。当然，一个团队协作开发一个嵌入式产品，其中各个成员可以扮演不同角色，但对系统的整体理解与把握并相互协作，有助于一个稳定可靠嵌入式产品的诞生。

1.3 嵌入式系统的学习困惑、知识体系及学习建议

1.3.1 嵌入式系统的学习困惑

关于嵌入式系统的学习方法，因学习经历、学习环境、学习目的、已有的知识基础等不同，可能在学习顺序、内容选择、实践方式等方面有所不同。但是，应该明确哪些是必备的基础知识，哪些应该先学，哪些应该后学；哪些必须通过实践才能了解；哪些是与具体芯片无关的通用知识，哪些是与具体芯片或开发环境相关的知识。

嵌入式系统的初学者应该通过选择一个具体 MCU 作为蓝本，期望通过学习实践，获得嵌入式系统知识体系的通用知识，其基本原则是：入门时间较快、硬件成本较少、软硬件资料规范、知识要素较多、学习难度较低。

由于微处理器与微控制器种类繁多，也可能由于不同公司、不同机构出于自身的利益，给出一些误导性宣传，特别是国内芯片制造技术的落后及其他相关情况，人们对微控制器及应用处理器的发展，在认识与理解上存在差异，一些初学者有些困惑。下面简要分析初学者可能存在的三个困惑。

（1）**嵌入式系统学习困惑之一：选择入门芯片问题。**在了解嵌入式系统分为微控制器与应用处理器两大类之后，入门芯片选择的困惑表述为：**选微控制器，还是应用处理器作为入门芯片呢？**从性能角度看，与应用处理器相比，微控制器工作频率低、计算性能弱、稳定性高、可靠性强。从使用操作系统角度看，与应用处理器相比，开发微控制器程序一般使用 RTOS，也可以不使用操作系统；而开发应用处理器程序，一般使用非实时操作系统。从知识要素角度看，与应用处理器相比，开发微控制器程序一般更需要了解底层硬件；而开发应用处理器终端程序，一般是在厂商提供的驱动基础上基于操作系统开发，更像开发一般 PC

^①目前，非易失性存储器通常为 Flash 存储器，特点见有关“Flash 在线编程”的内容。

软件的方式。从上述分析可以看出，要想成为一名知识结构合理且比较全面的嵌入式系统工程师，应该选择一个较典型的微控制器作为入门芯片，且从不带操作系统（No Operating System, NOS）学起，由浅入深，逐步推进。

关于学习芯片的选择还有一个困惑，是系统的工作频率。误认为选择工作频率高的芯片进行入门学习，表示更先进。实际上，工作频率高可能给初学者带来学习过程中的不少困难。

实际上，嵌入式系统设计不是追求芯片的计算速度、工作频率、操作系统等因素，而是追求稳定、可靠、维护、升级、功耗、价格等指标。

（3）嵌入式系统学习困惑之二：关于操作系统问题。是 NOS、RTOS 或 EOS？操作系统选择的困惑表述为：开始学习时，是无操作系统（NOS）、实时操作系统（RTOS），还是一般嵌入式操作系统（EOS）？学习嵌入式系统的目的是为了开发嵌入式应用产品，许多人想学习嵌入式系统，不知道该从何学起，具体目标也不明确。于是，看了一些培训广告，看了书店中书架上种类繁多的嵌入式系统的书籍，或上网以“嵌入式系统”为关键词进行搜索，然后参加培训或看书，开始“学习起来”。一些初学者，往往选择一个嵌入式操作系统就开始学习了。不十分恰当的比喻，有点儿像“瞎子摸大象”，只了解其一个侧面。这样难以对嵌入式产品的开发过程有全面了解。针对许多初学者选择“xxx 嵌入式操作系统+xxx 处理器”的嵌入式系统的入门学习模式，本书认为是不合适的。本书的建议是：首先把嵌入式系统软件与硬件基础打好，再根据实际应用需要，选择一种实时操作系统（RTOS）进行实践。读者必须明确认识到，RTOS 是开发某些嵌入式产品的辅助工具和手段，不是目的。况且，一些小型微型嵌入式产品并不需要 RTOS。因此，一开始就学习 RTOS，并不符合“由浅入深、循序渐进”的学习规律。

另外一个问题是：选 RTOS，还是 EOS？面向微控制器的应用，一般选择 RTOS，如 RT-Thread、mbedOS、MQX Lite、FreeRTOS、 μ COS-III 和 μ CLinux 等。RTOS 种类繁多，实际使用何种 RTOS，一般需要工作单位确定。基础阶段主要学习 RTOS 的基本原理，并学习在 RTOS 之上的软件开发方法，而不是学习如何设计 RTOS。面向应用处理器的应用，一般选择 EOS，如 Android、Linux、WindowsCE 等，可根据实际需要进行有选择的学习。

对于嵌入式操作系统，一定不要一开始就学，这样会走很多弯路，也会使读者对嵌入式系统感到畏惧。等读者的软硬件基础打好了，再学习就感到容易理解。实际上，众多 MCU 嵌入式应用，并不一定需要操作系统或只需要一个小型 RTOS，也可以根据实际项目需要再学习特定的 RTOS。一定不要被一些嵌入式实时操作系统培训班宣传所误导，而忽视实际嵌入式系统软件和硬件基础知识的学习。无论如何，以开发实际嵌入式产品为目标的学习者，不要把过多的精力花在设计或移植 RTOS、EOS 上面。正如很多人使用 Windows 操作系统，而设计 Windows 操作系统只有 Microsoft 公司；许多人“研究”Linux 系统，但从来没有使用它开发过真正的嵌入式产品；人的精力是有限的，因此学习必须有所选择。有的学习者，学了很长时间的嵌入式操作系统移植，而不进行实际嵌入式系统产品的开发，最后，做不好一个稳定的嵌入式系统小产品，偏离了学习目标，甚至放弃了嵌入式系统领域。

（3）嵌入式系统学习困惑之三：关于软件与硬件如何平衡问题。以 MCU 为核心的嵌入式技术的知识体系必须通过具体的 MCU 来体现、实践与训练。但是，选择任何型号的 MCU，其芯片相关的知识只占知识体系的 20%左右，剩余 80%左右的是通用知识。但是，

这 80%左右的通用知识，必须通过具体实践才能进行，因此学习嵌入式技术要选择一个系列的 MCU。但是，嵌入式系统均含有硬件与软件两大部分，它们之间的关系如何呢？

有些学者，仅从电子角度认识嵌入式系统，认为“嵌入式系统=MCU 硬件系统+小程序”。这些学者，大多具有良好的电子技术基础知识。实际情况是，早期 MCU 内部 RAM 小、程序存储器外接，需要外扩各种 I/O，没有像现在的 USB、嵌入式以太网等较复杂的接口，因此，程序占总设计量的 50%以下，使人们认为嵌入式系统（MCU）是“电子系统”，以硬件为主、程序为辅。但是，随着 MCU 制造技术的发展，不仅 MCU 内部 RAM 越来越大，Flash 进入 MCU 内部改变了传统的嵌入式系统开发与调试方式，固件程序可以被更方便地调试与在线升级，许多情况与开发 PC 机程序的难易程度相差无几，只不过开发环境与运行环境不是同一载体而已。这些情况使得嵌入式系统的软硬件设计方法发生了根本变化。特别是因软件危机而发展起来的软件工程学科对嵌入式系统软件的发展也产生重要影响，产生了嵌入式系统软件工程。

有些学者，仅从软件开发角度认识嵌入式系统，甚至有的仅从嵌入式操作系统认识嵌入式系统。这些学者，大多具有良好的计算机软件开发基础知识，认为硬件是生产厂商的事，他们没有认识到，嵌入式系统产品的软件与硬件均是需要开发者设计的。本书作者常常接到一些关于嵌入式产品稳定性的咨询电话，发现大多数是由于软件开发对底层硬件的基本原理不理解造成的。特别是，有些功能软件开发，过分依赖底层硬件驱动软件的设计，自己对底层驱动原理知之甚少。实际上，一些功能软件开发，名义上是在做嵌入式软件，但仅是使用嵌入式编辑、编译环境与下载工具而已，本质与开发通用 PC 软件没有两样。而底层硬件驱动软件的开发，若不全面考虑高层功能软件对底层硬件的可能调用，也会使得封装或参数设计的不合理或不完备，导致高层功能软件的调用相对困难。

从上述描述可以看出，若把一个嵌入式系统的开发孤立地分为硬件设计、底层硬件驱动软件设计、高层功能软件设计，一旦出现了问题，就可能难以定位。**实际上，嵌入式系统设计是一个软件和硬件协同设计的工程，不能像通用计算机那样，软件和硬件完全分开来看，要在一个大的框架内协调工作。**在一些小型公司，需求分析、硬件设计、底层驱动、软件设计、产品测试等过程可能是由同一个团队完成的，这就需要团队成员对软件、硬件及产品需求有充分认识，才能协作完成开发。甚至许多实际情况是在一些小型公司这个“团队”可能就是一个人。

面对学习嵌入式系统以软件为主还是以硬件为主，或是如何选择切入点，如何在软件与硬件之间找到平衡。对于这个困惑的建议是：**要想成为一名合格的嵌入式系统设计工程师，在初学阶段，必须重视打好嵌入式系统的硬件与软件基础。**以下是从事嵌入式系统设计二十多年的一个美国学者 John Catsoulis 在 *Designing Embedded Hardware* 一书中关于这个问题的总结：**嵌入式系统与硬件紧密相关，是软件与硬件的综合体，没有对硬件的理解就不可能写好嵌入式软件，同样没有对软件的理解也不可能设计好嵌入式硬件。**

充分理解嵌入式系统软件与硬件相互依存关系，对嵌入式系统的学习有良好的促进作用。一方面，既不能只重视硬件，而忽视编程结构、编程规范、软件工程的要求、操作系统等知识的积累；另一方面，也不能仅从计算机软件角度，把通用计算机学习过程中的概念与方法生搬硬套到嵌入式系统的学习实践中，而忽视嵌入式系统与通用计算机的差异。在嵌入

式系统学习与实践的初始阶段，应该充分了解嵌入式系统的特点，根据自身已有的知识结构，制定适合自身情况的学习计划。其目标应该是打好嵌入式系统的硬件与软件基础，通过实践，为成为良好的嵌入式系统设计工程师建立起基本知识结构。学习过程可以通过具体应用系统为实践载体，但不能拘泥于具体系统，应该有一定的抽象与归纳。例如，有的初学者开发一个实际控制系统，没有使用实时操作系统，但不要认为实时操作系统不需要学习，要注意知识学习的先后顺序与时间点的把握。又例如，有的初学者以一个带有实时操作系统的样例为蓝本进行学习，但不要认为，任何嵌入式系统都需要使用实时操作系统，甚至把一个十分简明的实际系统加上一个不必要的实时操作系统。因此，**片面认识嵌入式系统，可能导致学习困惑**。应该根据实际项目需要，锻炼自己分析实际问题、解决问题的能力。这是一个较长期的、需要静下心来学习与实践过程，不能期望通过短期培训完成整体知识体系的建立，应该重视自身实践，全面地理解与掌握嵌入式系统的知识体系。

1.3.2 嵌入式系统的知识体系

从由浅入深、由简到繁的学习规律来说，嵌入式学习的入门应该选择微控制器，而不是应用处理器，应通过对微控制器基本原理与应用的学习，逐步掌握嵌入式系统的软件与硬件基础，然后在此基础上进行嵌入式系统其他方面知识的学习。

本书主要阐述以 MCU 为核心的嵌入式技术基础与实践。要完成一个以 MCU 为核心的嵌入式系统应用产品设计，需要有硬件、软件及行业领域的相关知识。硬件主要有 MCU 的硬件最小系统、输入输出外围电路、人机接口设计。软件设计有固化软件的设计，也可能含 PC 软件的设计。行业知识需要通过协作、交流与总结获得。

概括地说，学习以 MCU 为核心的嵌入式系统，需要以下软件和硬件基础知识与实践训练，即以 MCU 为核心的嵌入式系统的基本知识体系如下^①。

(1) **掌握硬件最小系统与软件最小系统框架**。硬件最小系统是包括电源、晶振、复位、写入调试器接口等可使内部程序得以运行的、规范的、可复用的核心构件系统^②。软件最小系统框架是一个能够点亮一个发光二极管的，甚至带有串口调试构件的，包含工程规范完整要素的可移植与可复用的工程模板^③。

(2) **掌握常用基本输出的概念、知识要素、构件使用方法及构件设计方法**。如通用 I/O (GPIO)、模数转换 ADC、数模转换 DAC、定时器模块等。

(3) **掌握若干嵌入式通信的概念、知识要素、构件使用方法及构件设计方法**。如串行通信接口 UART、串行外设接口 SPI、集成电路互联总线 I2C, CAN、USB、嵌入式以太网、无线射频通信等。

(4) **掌握常用应用模块的构件设计方法、使用方法及数据处理方法**。如显示模块(LED、LCD、触摸屏等)、控制模块(控制各种设备，包括 PWM 等控制技术)等。数据处理如图

^①有关名词解释详见本章 1.4 节，本书将逐步学习这些内容。

^②将在本书第 3 章阐述。

^③将在本书第 4 章和第 6 章阐述。

形、图像、语音、视频等处理或识别等。

(5) **掌握一门实时操作系统的基本用法与基本原理。**作为软件辅助开发工具的实时操作系统，也可以作为一个知识要素。可以选择一种（如：mbedOS、MQX Lite、 μ C/OS 等）进行学习实践，在没有明确目的的情况下，没必要选择几种同时学习。学好其中一种，在确有必要使用另一种实时操作系统时，再学习，也可触类旁通。

(6) **掌握嵌入式软硬件的基本调试方法。**如断点调试、打桩调试、printf 调试方法等。在嵌入式调试过程中，特别要注意确保在正确硬件环境下调试未知软件，在正确软件环境下调试未知硬件。

这里给出的是基础知识要素，关键还是看如何学习，是他人做好了驱动程序开发人员直接使用，还是开发人员自己完全掌握知识要素，从底层开始设计驱动程序，同时熟练掌握驱动程序的使用。体现在不同层面的人才培养中。而应用中的硬件设计、软件设计、测试等都必须遵循嵌入式软件工程的方法、原理与基本原则。因此，嵌入式软件工程也是嵌入式系统知识体系的有机组成部分，只不过，它融于具体项目的开发过程之中。

若是主要学习应用处理器类的嵌入式应用，也应该在了解 MCU 知识体系的基础上，选择一种嵌入式操作系统（如 Android、Linux 等）进行学习实践。目前，APP 开发也是嵌入式应用的一个重要组成部分，可选择一种 APP 开发进行实践（如 Android APP、iOS APP 等）。

与此同时，在 PC 上，利用面向对象编程语言进行测试程序、网络侦听程序、Web 应用程序的开发及对数据库的基本了解与应用，也应逐步纳入嵌入式应用的知识体系中。此外，理工科的公共基础本身就是学习嵌入式系统的基础。

1.3.3 基础阶段的学习建议

十多年来，嵌入式开发工程师们逐步探索与应用构件封装的原则，把硬件相关的部分封装底层构件，统一接口，努力使高层程序与芯片无关，可以在各种芯片应用系统移植与复用，试图降低学习难度。学习的关键就变成了了解底层构件设计方法，掌握底层构件的使用方式，在此基础上，进行嵌入式系统设计与应用开发。当然，掌握底层构件的设计方法，学会实际设计一个芯片的某一模块的底层构件，也是本科学生应该掌握的基本知识。对于专科类学生，可以直接使用底层构件进行应用编程，但也需要了解知识要素的抽取方法与底层构件基本设计过程。对于看似庞大的嵌入式系统知识体系，可以使用“电子札记”的方式进行知识积累与补缺补漏，任何具有一定理工科基础的学生，通过一段稍长时间的静心学习与实践，都能学好嵌入式系统。

下面针对嵌入式系统的学习困惑，从嵌入式系统的知识体系角度，对广大渴望学习嵌入式系统的读者提出 5 点基础阶段的学习建议：

(1) **遵循“先易后难，由浅入深”的原则，打好软硬件基础。**跟随本书，充分利用本书提供的软硬件资源及辅助视频材料，逐步实验与实践^①；充分理解硬件基本原理、掌握功

^①这里说的实验主要指通过重复或验证他人的工作，其目的是学习基础知识，这个过程一定要经历。实践是自己设计，有具体的“产品”目标。如果你能花 500 元左右自己做一个具有一定

能模块的知识要素、掌握底层驱动构件的使用方法、掌握 1~2 个底层驱动构件的设计过程与方法；熟练掌握在底层驱动构件基础上，利用 C 语言编程实践。理解学习嵌入式系统，必须勤于实践。关于汇编语言问题，随着 MCU 对 C 语言编译的优化支持，可以只了解几个必须的汇编语句，但必须通过第一个程序理解芯片初始化过程、中断机制、程序存储情况等区别于 PC 程序的内容；最好认真理解一个真正的汇编实例。另外，为了测试的需要，最好掌握一门 PC 方面面向对象的编程高级语言（如 C#），本书电子资源中给出了 C#快速入门的方法与实例。

（2）充分理解知识要素、掌握底层驱动构件的使用方法。本书对诸如 GPIO、UART、定时器、PWM、ADC、DAC、Flash 在线编程等模块，首先阐述其通用知识要素，随后给出其底层驱动构件的基本内容。期望读者在充分理解通用知识要素的基础上，学会底层驱动构件的使用方法。即使只有这一点，也要下一番功夫。俗话说，书读百遍，其义自见。有关知识要素涉及硬件基本原理，以及对底层驱动接口函数功能及参数的理解，需反复阅读、反复实践，查找资料，分析、概括及积累。对于硬件，只要在深入理解 MCU 的硬件最小系统基础上，对上述各硬件模块逐个实验理解，逐步实践，再通过自己动手完成一个实际小系统，就可以基本掌握底层硬件基础。同时，这个过程也是软硬件结合学习的基本过程。

（3）基本掌握底层驱动构件的设计方法。对本科学历以上的读者，至少掌握 GPIO 构件的设计过程与设计方法（第 4 章）、UART 构件的设计过程与设计方法（第 6 章），透彻理解构件化开发方法与底层驱动构件封装规范（第 5 章）。从而对底层驱动构件有较好的理解与把握。这是一份细致、静心的任务，力戒浮躁，才能理解其要义。书中的底层驱动构件吸取了软件工程的基本原理，学习时需要注意基本规范。

（4）掌握单步跟踪调试、打桩调试、printf 输出调试等调试手段。在初学阶段，充分利用单步跟踪调试了解与硬件打交道的寄存器值的变化，理解 MCU 软件干预硬件的方式。单步跟踪调试也用于底层驱动构件设计阶段。不进入子函数内部执行的单步跟踪调试，可用于整体功能跟踪。打桩调试主要用于编程过程中，功能确认。一般编写几句程序语句后，即可打桩，调试观察。通过串口 printf 输出信息在 PC 机屏幕显示，是嵌入式软件开发中重要的调试跟踪手段，与 PC 编程中 printf 函数功能类似，只是嵌入式开发 printf 输出是通过串口输出到 PC 屏幕，PC 上需用串口调试工具显示，PC 编程中 printf 直接将结果显示在 PC 屏幕上。

（5）日积月累，勤学好问，充分利用本书及相关资源。有副对联：“智叟何智只顾眼前捞一把，愚公不愚哪管艰苦移二山”。学习嵌入式切忌急功近利，需要日积月累、循序渐进，充分掌握与应用“电子札记”方法。同时，要勤学好问，下真功夫、细功夫。人工智能学科里有个术语叫无教师指导学习模式与有教师指导学习模式，无教师指导学习模式比有教师指导学习模式复杂许多。因此，要多请教良师，少走弯路。此外，本书提供了大量经过打磨的、比较规范的软硬件资源，充分用好这些资源，可以更上一层楼。

以上建议，仅供参考。当然，以上只是基础阶段的学习建议，要成为良好的嵌入式系统设计工程师，还需要注重理论学习与实践、通用知识与芯片相关知识、硬件知识与软件知识

功能的小产品，且能稳定运行 1 年以上，就可以说接近入门了。

的平衡。要在理解软件工程基本原理的基础上，理解硬件构件与软件构件等基本概念。在实际项目中锻炼，并不断学习与积累经验。

1.4 微控制器与应用处理器简介

嵌入式系统的主要芯片为两大类：面向测控领域的微控制器类与面向多媒体应用领域的应用处理器类，本节给出其基本含义及特点。

1.4.1 MCU简介

1. MCU的基本含义

MCU 是单片微型计算机(单片机)的简称，早期的英文名是 Single-chip Microcomputer，后来大多数称之为微控制器（Micro-controller）或嵌入式计算机（Embedded Computer）。现在 Micro-controller 已经是计算机中一个常用术语，但在 1990 年之前，大部分英文词典并没有这个词。我国学者一般使用中文“单片机”一词，而缩写使用“MCU”，来自于英文“Microcontroller Unit”。因此本书后面的简写一律以 MCU 为准。MCU 的基本含义是：在一块芯片内集成了中央处理单元（Central Processing Unit, CPU）、存储器（RAM/ROM 等）、定时器/计数器及多种输入输出（I/O）接口的比较完整的数字处理系统。图 1-5 给出了典型的 MCU 组成框图。

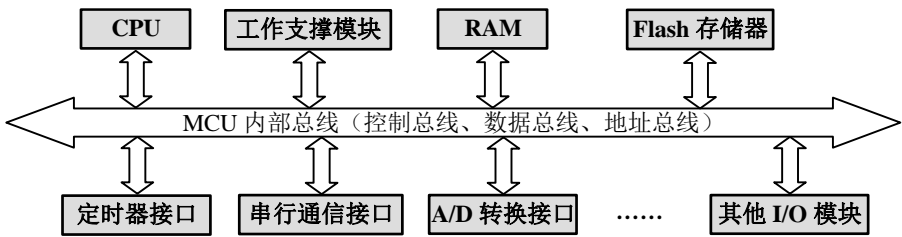


图1-5 一个典型的MCU内部框图

MCU 是在计算机制造技术发展一定阶段的背景下出现的，它使计算机技术从科学计算领域进入智能化控制领域。从此，计算机技术在两个重要领域——通用计算机领域和嵌入式（Embedded）计算机领域都获得了极其重要的发展，为计算机的应用开辟了更广阔的空间。

就 MCU 的组成而言，虽然它只是一块芯片，但包含了计算机的基本组成单元，仍由运算器、控制器、存储器、输入设备、输出设备五部分组成，只不过这些都集成在一块芯片内，这种结构使得 MCU 成为具有独特功能的计算机。

2. 嵌入式系统与MCU的关系

何立民先生说：“有些人搞了十多年的 MCU 应用，不知道 MCU 就是一个最典型的嵌

入式系统”^①。实际上，MCU 是在通用 CPU 基础上发展起来的，MCU 具有体积小、价格低、稳定可靠等优点，它的出现和迅猛发展，是控制系统领域的一场技术革命。MCU 以其较高的性价比、灵活性等特点，在现代控制系统中具有十分重要的地位。大部分嵌入式系统以 MCU 为核心进行设计。MCU 从体系结构到指令系统都是按照嵌入式系统的应用特点专门设计的，它能很好地满足应用系统的嵌入、面向测控对象、现场可靠运行等方面的要求。因此，以 MCU 为核心的系统是应用最广的嵌入式系统。在实际应用时，开发者可以根据具体要求与应用场合，选用最佳型号的 MCU 嵌入实际应用系统中。

3. MCU出现之后测控系统设计方法发生的变化

测控系统是现代工业控制的基础，它包含信号检测、处理、传输与控制等基本要素。在 MCU 出现之前，人们必须用模拟电路、数字电路实现测控系统中的大部分计算与控制功能，这样使得控制系统体积庞大，易出故障。MCU 出现以后，测控系统设计方法逐步产生变化，系统中的大部分计算与控制功能由 MCU 的软件实现。其他电子线路成为 MCU 的外围接口电路，承担输入、输出与执行动作等功能，而计算、比较与判断等原来必须用电路实现的功能，可以用软件取代，大大提高了系统的性能与稳定性，这种控制技术称之为嵌入式控制技术。在嵌入式控制技术中，核心是 MCU，其他部分依次展开。下面给出一个典型的以 MCU 为核心的嵌入式测控产品的基本组成。

1.4.2 以MCU为核心的嵌入式测控产品的基本组成

一个以MCU为核心，比较复杂的嵌入式产品或实际嵌入式应用系统，包含模拟量的输入、模拟量的输出，开关量的输入、开关量的输出及数据通信的部分。而所有嵌入式系统中最为典型的则是嵌入式测控系统。图1-6给出了一个典型的嵌入式测控系统框图。

1. MCU工作支撑电路

MCU 工作支撑电路也就是 MCU 硬件最小系统，它保障 MCU 能正常运行，如电源电路、晶振电路及必要的滤波电路等，甚至可包含程序写入器接口电路。

2. 模拟信号输入电路

实际模拟信号一般来自相应的传感器。例如，要测量室内的温度，就需要温度传感器。但是，一般传感器将实际的模拟信号转成的电信号都比较微弱，MCU 无法直接获得该信号，需要将其放大，然后经过模数转换 ADC 变为数字信号，进行处理。目前许多 MCU 内部包含 ADC 模块，实际应用时也可根据需要外接 ADC 芯片。常见的模拟量有：温度、湿度、压力、重量、气体浓度、液体浓度、流量等。对 MCU 来说，模拟信号通过 ADC 变成相应的数字序列进行处理。

3. 开关量信号输入电路

实际开关信号一般也来自相应的开关类传感器。例如，光电开关、电磁开关、干簧管（磁

^① 《单片机与嵌入式系统应用》，2004 年第 1 期。

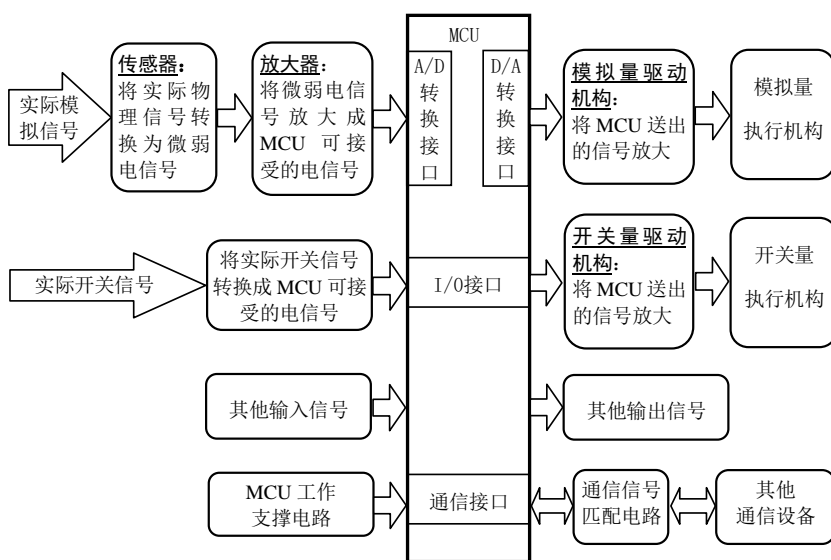


图1-6 一个典型的嵌入式测控系统框图

开关）、声控开关、红外开关等，一些儿童电子玩具中就有一些类似的开关。手动开关也可作为开关信号送到 MCU 中。对 MCU 来说，开关信号就是只有“0”和“1”两种可能值的数字信号。

4. 其他输入信号或通信电路

其他输入信号通过某些通信方式与 MCU 沟通。常用的通信方式有：异步串行（UART）通信、串行外设接口（SPI）通信、并行通信、USB 通信、网络通信等。

5. 输出执行机构电路

在执行机构中，有开关量执行机构，也有模拟量执行机构。开关量执行机构只有“开”、“关”两种状态。模拟量执行机构需要连续变化的模拟量控制。MCU 一般是不能直接控制这些执行机构，需要通过相应的隔离和驱动电路实现。还有一些执行机构，既不是通常的开关量控制，也不是通常数模转换量控制，而是“脉冲”量控制，如控制调频电动机，MCU 则通过软件对其控制。

1.4.3 MAP简介

1. 应用处理器的基本概念及特点

MAP 是在低功耗 CPU 的基础上扩展音视频功能和专用接口的超大规模集成电路。与 MCU 相比，MAP 的最主要特点是：工作频率高；硬件设计更为复杂；软件开发需要选用一个嵌入式操作系统；计算功能更强；抗干扰性能较弱；较少直接应用于控制目标对象；一般情况下，MAP 芯片价格也高于 MCU。

MAP 是伴随着便携式移动设备特别是智能手机而产生的。手机的技术核心是一个语音

压缩芯片，称为基带处理器，发送时对语音进行压缩，接收时解压缩，传输码率只是未压缩的几十分之一，在相同的带宽下可服务更多的用户。而智能手机上除通信功能外还增加了数码相机、音乐播放、视频图像播放等功能，基带处理器已经没有能力处理这些新加的功能。另外，视频、音频（高保真音乐）处理的方法和语音不一样，语音只要能听懂，达到传达信息的目的就可以了，视频要求亮丽的彩色图像，动听的立体声伴音，使人能得到最大的感官享受。为了实现这些功能，需要另外一个协处理器专门处理这些信号，它就是 MAP。

针对便携式移动设备，MAP 的性能需要满足以下 3 点。

（1）低功耗。这是因为 MAP 用在便携式移动设备上，通常用电池供电，节能显得格外重要，使用者给电池充满电后希望使用尽可能长的时间。通常，MAP 的核心电压是 0.9~1.2V，接口电压是 2.5V 或 3.3V，待机功耗小于 3mw，全速工作时 100~300mw。

（2）体积微小。因为 MAP 主要应用在手持式设备中，每 1mm 空间都很宝贵。MAP 通常采用小型 BGA 封装，引脚数有 300~1000 个，锡球直径是 0.3~0.6mm，间距是 0.45~0.75mm。

（3）具备尽可能高的性能。目前的便携式移动设备具备了 DAB（digital audio broadcasting）、蓝牙耳机、无线宽带（WiFi）、GPS 导航、3D 游戏等功能，新的功能仍在积极开发中，这些功能都对 MAP 的性能提出了更高的要求。

2. MAP与MCU的接口比较

MAP 的接口相较于 MCU 更加丰富，除了 MCU 常见的接口，如通用 I/O（GPIO）、模数转换（ADC）、数模转换（DAC）、串行通信接口（UART）、串行外设接口（SPI）、I2C、CAN、USB、嵌入式以太网、LED、LCD 等之外，因 MAP 的场景多有多媒体、与 PC 方便互联等需要，因此其接口通常还包括了 PCI、TU-R 656、TS、AC97、3D、2D、闪存、DDR、SD 等。

3. RISC-V应用处理器架构

RISC-V 架构作为一种指令集架构，相比其他成熟的商业架构，最大的不同在于它是一个模块化的架构。因此 RISC-V 不仅短小精悍，而且其不同的部分还能以模块化的方式组织在一起，从而试图通过一套统一的架构满足各种不同的应用。这种模块化的 RISC-V 架构能够使得用户灵活地选择不同的模块进行组合，以满足不同的应用场景，可以说是应用广泛。例如针对小面积、低功耗的嵌入式场景，用户可以选择 RV32IC 组合的指令集，仅使用机器模式（Machine Mode）；而针对高性能应用操作系统场景，则可以选择例如 RV32IMFDC 的指令集，使用机器模式（Machine Mode）与用户模式（User Mode）两种模式。

1.5 嵌入式系统常用术语

在学习嵌入式应用技术的过程中，经常会遇到一些名词术语。从学习规律的角度，初步了解这些术语有利于随后的学习。因此，本节对嵌入式系统的一些常用术语给出简要说明，以便读者有个初始印象。

1.5.1 与硬件相关的术语

1. 封装

集成电路的封装（package）是指用塑料、金属或陶瓷等材料把集成电路封在其中。封装可以保护芯片，并使芯片与外部世界连接。常用的封装形式可分为通孔封装和贴片封装两大类。

通孔封装主要有：单列直插（Single-In-line Package, SIP）、双列直插（Dual-In-line Package, DIP）、Z 字型直插式封装（Zigzag-In-line Package, ZIP）等。

常见的贴片封装主要有：小外形封装（Small Outline Package, SOP）、紧缩小外形封装（Shrink Small Outline Package, SSOP）、四方扁平封装（Quad-Flat Package, QFP）、塑料薄方封装（Plastic-Low-Profile Quad-Flat Package, LQFP）、塑料扁平组件式封装（Plastic Flat Package, PFP）、插针网格阵列封装（Ceramic Pin Grid Array package, PGA）、球栅阵列封装（Ball Grid Array package, BGA）等。

2. 印刷电路板

印刷电路板（Printed Circuit Board, PCB）是组装电子元件用的基板，是在通用基材上按预定设计形成点间连接及印制元件的印制板，是电路原理图的实物化。PCB 的主要功能是提供集成电路等各种电子元器件固定、装配的机械支撑；实现集成电路等各种电子元器件之间的布线和电气连接（信号传输）或电绝缘；为自动装配提供阻焊图形，为元器件插装、检查、维修提供识别字符和图形等。

3. 动态可读写随机存储器与静态可读写随机存储器

动态可读写随机存储器（Dynamic Random Access Memory, DRAM），由一个 MOS 管组成一个二进制存储位。MOS 管的放电导致表示“1”的电压会慢慢降低。一般每隔一段时间就要控制刷新信息，给其充电。DRAM 价格低，但控制繁琐，接口复杂。

静态可读写随机存储器（Static Random Access Memory, SRAM），一般由 4 个或者 6 个 MOS 管构成一个二进制位。当电源有电时，SRAM 不用刷新，可以保持原有的数据。

4. 只读存储器

只读存储器（Read Only Memory, ROM），数据可以读出，但不可以修改，所以称之为只读存储器。通常存储一些固定不变的信息，如常数、数据、换码表、程序等。ROM 具有断电后数据不丢失的特点。ROM 有固定 ROM、可编程 ROM（即 PROM）和可擦除 ROM（即 EPROM）3 种。

PROM 的编程原理是通过大电流将相应位的熔丝熔断，从而将该位改写成 0，熔丝熔断后不能再次改变，所以只改写一次。

EPROM（Erase PROM）是可以擦除和改写的 ROM，它用 MOS 管代替了熔丝，因此可以反复擦除、多次改写。擦除是用紫外线擦除器来完成的，很不方便。有一种用低电压信号即可擦除的 EPROM 称为电可擦除 EPROM，简称为 E²PROM 或 EEPROM（Electrically

Erasable Programmable Read-Only Memory)。

5. 闪存存储器

闪存存储器简称闪存，是一种新型快速的 E²PROM。由于工艺和结构上的改进，闪存比普通的 E²PROM 的擦除速度更快，集成度更高。闪存相对于传统的 E²PROM 来说，其最大的优点是系统内编程，也就是说不需要另外的器件来修改内容。闪存的结构随着时代的发展而有些变动，尽管现代的快速闪存是系统内可编程的，但仍然没有 RAM 使用起来方便。擦写操作必须通过特定的程序算法来实现。

6. 模拟量与开关量

模拟量是指时间连续、数值也连续的物理量，如温度、压力、流量、速度、声音等。在工程技术上，为了便于分析，常用传感器、变换器将模拟量转换为电流、电压或电阻等电量。

开关量是指一种二值信号，用两个电平（高电平和低电平）分别来表示两个逻辑值（逻辑 1 和逻辑 0）。

1.5.2 与通信相关的术语

1. 并行通信

并行通信是指数据的各位同时在多根并行数据线上进行传输的通信方式，数据的各位同时由源到达目的地；适合近距离、高速通信；常用的有 4 位、8 位、16 位、32 位等同时传输。

2. 串行通信

串行通信是指数据在单线（电平高低表征信号）或双线（差分信号）上，按时间先后一位一位地传送，其优点是节省传输线，但相对于并行通信来说，速度较慢。在嵌入式系统中，串行通信一词一般特指用串行通信接口（UART）与 RS232 芯片连接的通信方式。下面介绍的 SPI、I²C、USB 等通信方式也属于串行通信，但由于历史发展和应用领域的不同，它们分别使用不同的专用名词来命名。

3. 串行外设接口

串行外设接口（Serial Peripheral Interface, SPI）也是一种串行通信方式，主要用于 MCU 扩展外围芯片使用。这些芯片可以是具有 SPI 接口的 A/D 转换、时钟芯片等。

4. 集成电路互联总线

集成电路互联（I²C）总线是一种由 PHILIPS 公司开发的两线式串行总线，有的书籍也记为 IIC 或 I²C，主要用于用户电路板内 MCU 与其外围电路的连接。

5. 通用串行总线

通用串行总线（Universal Serial Bus, USB）是 MCU 与外界进行数据通信的一种新方

式，其速度快、抗干扰能力强，在嵌入式系统中得到了广泛的应用。USB 不仅成为通用计算机上最重要的通信接口，也是手机、家电等嵌入式产品的重要通信接口。

6. 控制器局域网

控制器局域网是一种全数字、全开放的现场总线控制网络，目前在汽车电子中应用最广。

7. 边界扫描测试协议与串行线调试技术

边界扫描测试协议（Joint Test Action Group, JTAG）是由国际联合测试行动组开发，对芯片进行测试的一种方式，可将其用于对 MCU 的程序进行载入与调试。JTAG 能获取芯片寄存器等内容，或者测试遵守 IEEE 规范的器件之间引脚的连接情况。

串行线调试（Serial Wire Debug, SWD）技术使用 2 针调试端口，是 JTAG 的低针数和高性能替代产品，通常用于小封装微控制器的程序写入与调试。

关于通信相关的术语还有嵌入式以太网、无线传感器网络、ZigBee、射频通信等，本章不再进一步介绍。

1.5.3 与功能模块相关的术语

1. 通用输入/输出

通用输入/输出（General Purpose I/O, GPIO），即基本的输入/输出，有时也称并行 I/O。作为通用输入引脚时，MCU 内部程序可以读取该引脚，知道该引脚是“1”（高电平）或“0”（低电平），即开关量输入。作为通用输出引脚时，MCU 内部程序向该引脚输出“1”（高电平）或“0”（低电平），即开关量输出。

2. 模数转换与数模转换

模数转换（Analog to Digital Convert, ADC）的功能是将电压信号（模拟量）转换为对应的数字量。实际应用中，这个电压信号可能由温度、湿度、压力等实际物理量经过传感器和相应的变换电路转化而来。经过 ADC，MCU 就可以处理这些物理量。而与之相反，数模转换（Digital to Analog Convert, DAC）的功能则是将数字量转换为电压信号（模拟量）。

3. 脉冲宽度调制器

脉冲宽度调制器（Pulse Width Modulator, PWM）是一个数模转换器，可以产生一个高电平和低电平之间重复交替的输出信号，这个信号就是 PWM 信号。

4. 看门狗

看门狗（Watch Dog, WDG），是一个为了防止程序跑飞而设计的一种自动定时器。当程序跑飞时，由于无法正常执行清除看门狗定时器，看门狗定时器会自动溢出，使系统程序复位。

5. 液晶显示

液晶显示（Liquid Crystal Display, LCD）是电子信息产品的一种显示器件，可分为字段型、点阵字符型、点阵图形型三类。

6. 发光二极管

发光二极管（Light Emitting Diode, LED）是一种将电流顺向通到半导体 PN 结处而发光的器件。常用于家电指示灯、汽车灯和交通警示灯。

7. 键盘

键盘是嵌入式系统中最常见的输入设备。识别键盘是否有效被按下的方法有查询法、定时扫描法和中断法等。

与功能模块相关的术语很多，这里不再进一步介绍，读者可在学习时逐步积累。

本章小结

1. 关于嵌入式系统的概念、分类与特点

关于嵌入式系统的概念，可以直观表述为嵌入式系统，即嵌入式计算机系统。嵌入式系统是不以计算机面目出现的“计算机”，这个计算机系统隐含在各类具体的产品之中，且在這些产品中，计算机程序起到了重要作用。关于嵌入式系统的分类，可以按应用范围简单地把嵌入式系统分为电子系统智能化（微控制器类）和计算机应用延伸（应用处理器类）这两大类。关于嵌入式系统的特点，可以从与通用计算机比较的角度，可以表述为嵌入式系统是不单独以通用计算机的面目出现的计算机系统，它的开发需要专用工具和特殊方法，使用 MCU 设计嵌入式系统，数据与程序空间采用不同存储介质，开发嵌入式系统涉及软件、硬件及应用领域的知识等。

2. 关于嵌入式系统的学习方法问题

关于芯片选择，建议初学者使用微控制器而不是使用应用处理器作为入门芯片。开始阶段，不学习操作系统，着重打好底层驱动的使用方法、设计方法等软硬件基础。关于硬件与软件平衡的问题，可以描述为：嵌入式系统与硬件紧密相关，是软件与硬件的综合体，没有对硬件的理解就不可能写好嵌入式软件，同样没有对软件的理解也不可能设计好嵌入式硬件。关于学习基本方法，建议遵循“先易后难，由浅入深”的原则，打好软硬件基础；充分理解知识要素、掌握底层驱动构件的使用方法；基本掌握底层驱动构件的设计方法；掌握单步跟踪调试、打桩调试、printf 输出调试等调试手段。

3. 关于MCU的基本含义

MCU 是在一块芯片内集成了 CPU、存储器、定时器/计数器及多种输入输出（I/O）接口的比较完整的数字处理系统。以 MCU 为核心的系统是应用最广的嵌入式系统，是现代测控系统的核心。MCU 出现之前，人们必须用纯硬件电路实现测控系统。MCU 出现以后，测控系统中的大部分计算与控制功能由 MCU 的软件实现，输入、输出与执行动作等通过硬件

实现，带来了设计上的本质变化。MAP 是在低功耗 CPU 的基础上扩展音视频功能和专用接口的超大规模集成电路，其功能与开发方法接近 PC。

4. 关于嵌入式系统的常用术语

对于嵌入式系统的硬件、通信、功能模块等方面的术语，从这里开始认识，后续章节再理解。这里重点认识几个缩写词：GPIO、UART、ADC、DAC、PWM、SPI、I2C、LED 等，记住它们的英文全称、中文含义，有利于随后的学习，这是嵌入式系统的最基本内容。

习 题

1. 简要总结嵌入式系统的定义、由来、分类及特点。
2. 归纳嵌入式系统的学习困惑，简要说明如何消除这些困惑？
3. 简要归纳嵌入式系统的知识体系。
4. 结合书中给出的嵌入式系统基础阶段的学习建议，从个人角度，你认为应该如何学习嵌入式系统？
5. 简要给出 MCU 的定义及典型内部框图。
6. 举例给出一个具体的、以 MCU 为核心的嵌入式测控产品的基本组成。
7. 简要比较中央处理器（CPU）、微控制器（MCU）与应用处理器（MAP）。
8. 列表罗列嵌入式系统常用术语（中文名、英文缩写、英文全称）。

第3章 存储器映像、中断源与硬件最小系统

本章导读：本章首先概述以青稞 V4F 为核心的 CH32V307 系列 MCU，随后给出该 MCU 的存储器映像、中断源与硬件最小系统，并由此构建一种通用嵌入式计算机 GEC（型号为 AHL-CH32V307）作为本书硬件实践平台。MCU 的外围电路简单清晰，它以 MCU 为核心辅以电源电路、晶振电路、复位电路等最基本的电子线路，构成了 MCU 硬件最小系统，使得 MCU 的内部程序可以运行起来。

3.1 CH32V307系列MCU概述

本节简要概述了 CH32V307 系列的 MCU 命名规则、存储映像以及中断源，其中 MCU 命名规则帮助使用者获得芯片信息；CH32V307 存储映像把青稞 V4F 内核之外的模块，用类似存储器编址的方式，统一分配地址，关于存储空间的使用，主要记住片内 Flash 区和片内 RAM 区存储映像；中断源主要包括 CH32V307 中断源的定义及中断源的分类。

3.1.1 CH32V307系列MCU命名规则

CH32V307 系列 MCU 是南京沁恒（WCH）于 2021 年开始陆续推出基于 RISC-V 架构的青稞 V4F 内核处理器的超低功耗微控制器，工作频率达 144MHz，内部硬件模块主要包括 GPIO、UART、Flash、RAM、SysTick、Timer、PWM、RTC、WDG、12 位 A/D、SPI、I2C 与 TKEY、CAN、USB、OPA、RNG、SDIO、FSMC、DVP、ETH 等。该系列包含不同的产品线，如 CH32Vx03 为通用型系列，CH32V307 为互联型系列，满足不同应用的选型需要。

认识一个 MCU，从了解型号含义开始，一般来说，主要包括芯片家族、产品类型、具体特性、引脚数目、Flash 大小、封装类型以及温度范围等。

表3-1 CH32系列芯片命令字段说明

字段	说明	取值
CH32	芯片家族	CH32表示32位MCU
X	产品类型	F表示增强型Cortex-M3；V表示增强型RISC-V
AAA	具体特性	取决于产品系列3xx：主流增强型MCU；
Y	引脚数目	C表示48；R表示64；V表示100
B	Flash大小	6表示32KB；8表示64KB；C表示256KB
T	封装类型	T表示LQFP封装；H 表示BGA； I表示UFBGA； U表示QFN
C	温度范围	6/A表示 - 40℃~+85℃；7/B表示 - 40℃~+105℃；3/C表示 - 40℃~+125℃；D表示 - 40℃~+150℃

CH32 系列芯片的命名格式为：“CH32 X 307 Y B T C”，各字段说明如表 3-1 所示，本书所使用的芯片型号为 CH32V307VCT6。对照命名格式，可以从型号获得以下信息：

属于 32 位的 MCU，超低功耗型，高性能微控制器，引脚数 100，Flash 大小为 480KB^①，封装形式：100 引脚 LQFP 封装；工作范围 - 40℃~+85℃。

3.1.2 CH32V307存储器映像

青稞 V4F 处理器直接寻址空间为 4GB，地址范围是：0x0000_0000~0xFFFF_FFFF。所谓存储器映像是指，把这 4GB 空间当做存储器来看待，分成若干区间，都可安排一些什么实际的物理资源。哪些地址服务什么资源是 MCU 生产厂家规定好的，用户一般只能使用而不能改其性质。

CH32V307 将内核之外的模块进行统一分配编址。在 4G 的存储映器射空间内，片内 Flash、静态存储器 SRAM、系统配置寄存器以及其它外设，均有独立的地址，以便内核进行访问，表 3-2 给出了本书使用的 CH32V307 系列存储器映像的主要常用部分内容。

表3-2 CH32V307存储映器映像表

32位地址范围	对应内容	说明
0x0000_0000~0x0800_0000	Flash或系统存储器的映射	取决于BOOT配置
0x0800_0000~0x0807_7FFF	Flash存储器	480KB
...		
0x2000_0000~0x2005_0000	SRAM	本书默认使用 0x2000_0000~0x2001_0000，64KB ^②
0x2001_0000~0x3FFF_FFFF	保留	
0x4000_0000~0x5005_4000	系统总线和外围总线	GPIO(0x4001_0800~0x4001_1C00)
...		

关于存储空间的使用，主要了解记住片内 Flash 区和片内 SRAM 区的大小及映像地址范围。因为中断向量，程序代码，常数放在片内 Flash 中，在源程序编译后的链接阶段需要使用的链接文件中，需要含有目标芯片 Flash 的地址范围以及用途等信息，才能顺利生成机器码。在产生的链接文件中还需要包含 RAM 的地址范围及用途等信息，以便生成机器码来准确定位全局变量，静态变量的地址及堆栈指针。

1. 片内Flash区存储器映像空间

CH32V307 片内 Flash 大小为 480KB，用于存储中断向量、程序代码、常数等，地址范围是：0x0800_0000-0x0807_7FFF，可分为 1920 扇区（页），每扇区大小 256 字节。

^① 该芯片 Flash 存储器大小实际为 480KB，默认配置为 256KB 被复制到 RAM 中，以便支持更高频率（即 144MHz）下运行，初学者按照 480KB 看待即可，本书芯片初始化运行频率 72MHz。

^② 0x2000_0000~0x2005_0000 共 320KBSRAM 空间，可以分为快速 CODE 区和实际 RAM 区注，可使用四种配置之一：（192，128）、（224，96）、（256，64）、（288，32），单位 KB。本书默认使用（256，64）配置。在 CH32V307 芯片中，为了解决运行 Flash 中程序比放在 SRAM 运行慢的问题，芯片会自动将 Flash 前部的代码复制到 SRAM 后部，转到 SRAM 中运行，速度提高一倍左右。因此，SRAM 中的这部分空间不能作为 RAM 使用，称为快速 CODE 区，例如，当 SRAM 配置成（192，128）时，Flash 中的前 192KB 被复制到 SRAM 区运行，若程序大小超过 192KB，则剩余部分仍在 Flash 中运行。

2. 片内RAM区存储器映像空间

CH32V307 片内 RAM 为静态随机存储 SRAM，用于存储全局变量、静态变量、临时变量（堆栈空间）等。地址范围为：0x2000_0000-0x2001_0000，即 64KB^①，支持字节、半字（2 字节）、全字（4 字节）访问。该芯片的堆栈空间的使用方向是相对方向进行的，因此将堆栈的栈顶设置成 SRAM 地址的最大值。这样栈的生长方向是从 SRAM 的高地址向低地址，堆的生长方向为 SRAM 的低地址向高地址。这样就可以减少重叠错误。

3. 系统启动区存储器映像空间

CH32V307 芯片 Flash 中有 28KB 的引导区（Boot Loader），内有厂家预置的引导程序。用户可以根据 BOOT0、BOOT1 引脚的配置，设置程序复位后的启动模式。BOOT0 引脚为独立的引脚，BOOT1 引脚为 PTB2，用于选择系统启动模式，启动模式引脚硬件连接如表 3-3 所示。

表3-3 启动模式的硬件连接

BOOT0	BOOT1	启动模式	用途
0	x	从程序Flash中启动	一般用户程序
1	0	从系统存储器启动	厂家BOOT程序升级
1	1	从内部SRAM启动	调试模式可以使用

启动模式不同，程序闪存存储器、系统存储器和内部 SRAM 有着不同的访问方式：

CH32V307 芯片从程序闪存存储器启动时，程序闪存存储器地址被映射到 0x00000000 地址区域，同时也能够在原地址区域 0x08000000 访问。从系统存储器启动时，系统存储器地址被映射到 0x00000000 地址区域，同时也能够在原地址区域 0x1FFFF000 访问。从内部 SRAM 启动，只能够从 0x20000000 地址区域访问。

4. 其他存储器映像空间

其他存储映像，如外设区存储映像（GPIO 等），系统保留段存储映像等，只需了解即可，实际使用时，由芯片头文件给出宏定义。

3.1.3 CH32V307中断源

中断是计算机发展中一个重要的技术，它的出现很大程度上解放了处理器，提高了处理器的执行效率。所谓中断，是指 MCU 正常运行程序时，由于 MCU 内核异常或者 MCU 各模块发出请求事件，引起 MCU 停止正在运行的程序，而转去处理异常或执行处理外部事件的程序（又称中断服务例程）。

这些引起 MCU 中断的事件称为中断源，一个 MCU 具有哪些中断源是在芯片设计阶段确定的。CH32V307 的中断源分为两类，一类是内核中断，另一类是非内核中断，如表 3-4 所示，这种表供中断编程时备查。内核中断主要是异常中断，也就是说，当出现错误的时候，这些中断会复位芯片或是做出其他处理。非内核中断是指 MCU 各个模块引起的

^① 必要时可配置为 128KB，配置方法参考芯片应用手册，本书作为基础教学，不涉及该问题。

中断，MCU 执行完中断服务例程后，又回到刚才正在执行的程序，从停止的位置继续执行后续的指令。非内核中断又称可屏蔽中断，这类中断可以通过编程控制开启或关闭该类中断。表 3-4 给出了 CH32V307VCT6 中断源，包含中断请求号（Interrupt Request, IRQ）、优先级、中断源及描述等信息。IRQ 号是从 0 开始编号的，包含内核中断和非内核中断，与样例工程的中断向量表中排列一一对应。

表3-4 CH32V307VCT6中断源

中断类型	IRQ号	优先级	中断源	描述
内核中断	0~1		保留	
	2	-5	NMI	不可屏蔽中断
	3	-4	HardFault	异常中断
	4		保留	
	5	-3	Ecall-M	机器模式回调中断
	6~7		保留	
	8	-2	Ecall-U	用户模式回调中断
	9	-1	BreadPoint	断点回调中断
	10~11		保留	
	12	0	SysTick	系统定时器中断
	13		保留	
	14	1	SW	软件中断
	15		保留	
外部中断	16	2	WWDG	窗口定时器中断
	17	3	PVD	电源电压检测中断（EXTI）
	18	4	TAMPER	侵入检测中断
	19	5	RTC	实时时钟中断
	20	6	Flash	闪存全局中断
	21	7	RCC	复位和时钟控制中断
	22~26	8~12	EXTI0~EXTI4	EXTI线0~4中断
	27~33	13~19	DMA1_CH1~7	DMA1通道1~7全局中断
	34	20	ADC1_2	ADC1和ADC2 全局中断
	35	21	USB_HP或CAN1_TX	USB_HP或CAN1_TX全局中断
	36	22	USB_LP或CAN1_RX0	USB_LP或CAN1_RX0全局中断
	37	23	CAN1_RX1	CAN1_RX1全局中断
	38	24	CAN1_SCE	CAN1_SCE全局中断
	39	25	EXTI9_5	EXTI线[9:5]中断
	40	26	TIM1_BRK	TIM1刹车中断
	41	27	TIM1_UP	TIM1更新中断

42	28	TIM1_TRG_COM	TIM1触发和通信中断
43	29	TIM1_CC	TIM1捕获比较中断
44~46	30~32	TIM2~4	TIM2~4全局中断
47	33	I2C1_EV	I2C1事件中断
48	34	I2C1_ER	I2C1错误中断
49	35	I2C2_EV	I2C2事件中断
50	36	I2C2_ER	I2C2错误中断
51~52	37~38	SPI1~2	SPI1~2全局中断
53~55	39~41	USART1~3	USART1~3全局中断
56	42	EXTI15_10	EXTI线[15:10]中断
57	43	RTCAlarm	RTC闹钟中断（EXTI）
58	44	USBWakeUp	USB唤醒中断（EXTI）
59	45	TIM8_BRK	TIM8刹车中断
60	46	TIM8_UP	TIM8更新中断
61	47	TIM8_TRG_COM	TIM8触发和通信中断
62	48	TIM8_CC	TIM8捕获比较中断
63	49	RNG	RNG全局中断
64	50	FSMC	FSMC全局中断
65	51	SDIO	SDIO全局中断
66	52	TIM5	TIM5全局中断
67	53	SPI3	SPI3全局中断
68~69	54~55	UART4~5	UART4~5全局中断
70~71	56~57	TIM6~7	TIM6~7全局中断
72~76	58~62	DMA2_CH1~5	DMA2通道15~全局中断
77	63	ETH	ETH全局中断
78	64	ETH_WKUP	ETH唤醒中断
79	65	CAN2_T	CAN2_TX全局中断
80	66	CAN2_RX0	CAN2_RX0全局中断
81	67	CAN2_RX1	CAN2_RX1全局中断
82	68	CAN2_SCE	CAN2_SCE全局中断
83	69	OTG_FS	全速OTG中断
84	70	USBHSWakeUp	高速USB唤醒中断
85	71	USBHS	高速USB全局中断
86	72	DVP	DVP全局中断
87~89	73~75	UART6~8	UART7~8全局中断
90	76	TIM9_BRK	TIM9刹车中断

91	77	TIM9_UP	TIM9更新中断
92	78	TIM9_TRG_COM	TIM9触发和通信中断
93	79	TIM9_CC	TIM9捕获比较中断
94	80	TIM10_BRK	TIM10刹车中断
95	81	TIM10_UP	TIM10更新中断
96	82	TIM10_TRG_COM	TIM10触发和通信中断
97	83	TIM10_CC	TIM10捕获比较中断
98~103	84~89	DMA2_CH6~11	DMA2通道6~11全局中断

3.2 CH32V307的引脚图与硬件最小系统

要使一个 MCU 芯片可以运行程序，必须为它做好服务工作，也就是找出哪些引脚需要我们提供服务，如电源与地、晶振、程序写入引脚、复位引脚等。

3.2.1 CH32V307的引脚图

本书以 100 引脚 LQFP 封装的 CH32V307VCT6 芯片为例阐述青稞 V4F 架构的 MCU 的编程和应用，图 3-1 给出的是 100 引脚 LQFP 封装的 CH32V307VCT6 的引脚图，芯片的引脚功能参阅电子资源“..\Information”文件夹中的数据手册第三章。

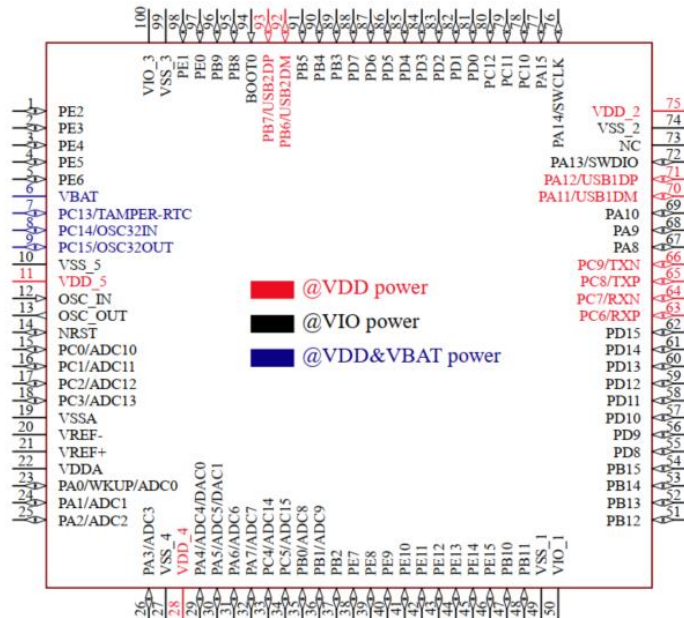


图3-1 100引脚LQFP封装 CH32V307VCT6

可以芯片引脚分为两大部分，一是需要用户为它服务的引脚，另一部分是它为用户服务的引脚。

1. 硬件最小系统引脚

硬件最小系统引脚是指需要为芯片提供服务的引脚，包括电源类引脚，复位引脚，晶振引脚等，表 3-5 中给出了 CH32V307VCT6 的硬件最小系统引脚。CH32V307VCT6 芯片电源类引脚在 LQFP 封装中有 13 个，芯片使用多组电源引脚为内部电压调节器、I/O 引脚驱动、AD 转换电路等电路供电，内部电压调节器为内核和振荡器等供电。为了提供稳定的电源，MCU 内部包含多组电源电路，同时给出多处电源引脚，便于外接滤波电容。为了电源平衡，MCU 提供了内部有共同接地点的多处电源引脚，供电路设计使用。

表3-5 CH32V307VCT6硬件最小系统引脚表

分类	引脚名	引脚号	功能描述
电源输入	VDD	11, 28, 75, 92, 93	电源，典型值：3.3V
	VSS	10, 27, 49, 74, 99	地，典型值：0V
	VSSA	19	AD模块的电源接地，典型值： 0V
	VDDA	22	AD模块的输入电源，典型值： 3.3V
	VBAT	6	内部RTC备用电源引脚
复位	NRST	14	双向引脚，有内部上拉电阻。作为输入，拉低可使芯片复位
晶振	PC14、PC15	8、9	低速无源晶振输入、输出引脚
	OSC_IN、OSC_OUT	12、13	外部高速无源晶振输入、输出引脚
SWD	SWD_DIO/PTA13	72	SWD数据信号线
	SWD_CLK/PTA14	76	SWD 时钟信号线
启动方式	BOOT0	94	程序启动方式控制引脚，BOOT0=0，从内部Flash中程序启动（本书使用）
	BOOT1/PTB2	37	程序启动方式控制引脚，BOOT0=0，BOOT1=0从系统存储器启动
引脚个数统计			硬件最小系统引脚为22个

2. 对外提供服务引脚

除了需要为芯片服务的引脚（硬件最小系统引脚）之外，芯片的其他引脚是向外提供服务的，也可称之为 I/O 端口资源类引脚，见表 3-6，这些引脚一般具有多种复用功能。

表3-6 CH32V307VCT6对外提供I/O端口资源类引脚表

端口号	引脚数	引脚名	硬件最小系统复用引脚
A	16	PTA[0-15]	PTA13、PTA14
B	16	PTB[0-15]	

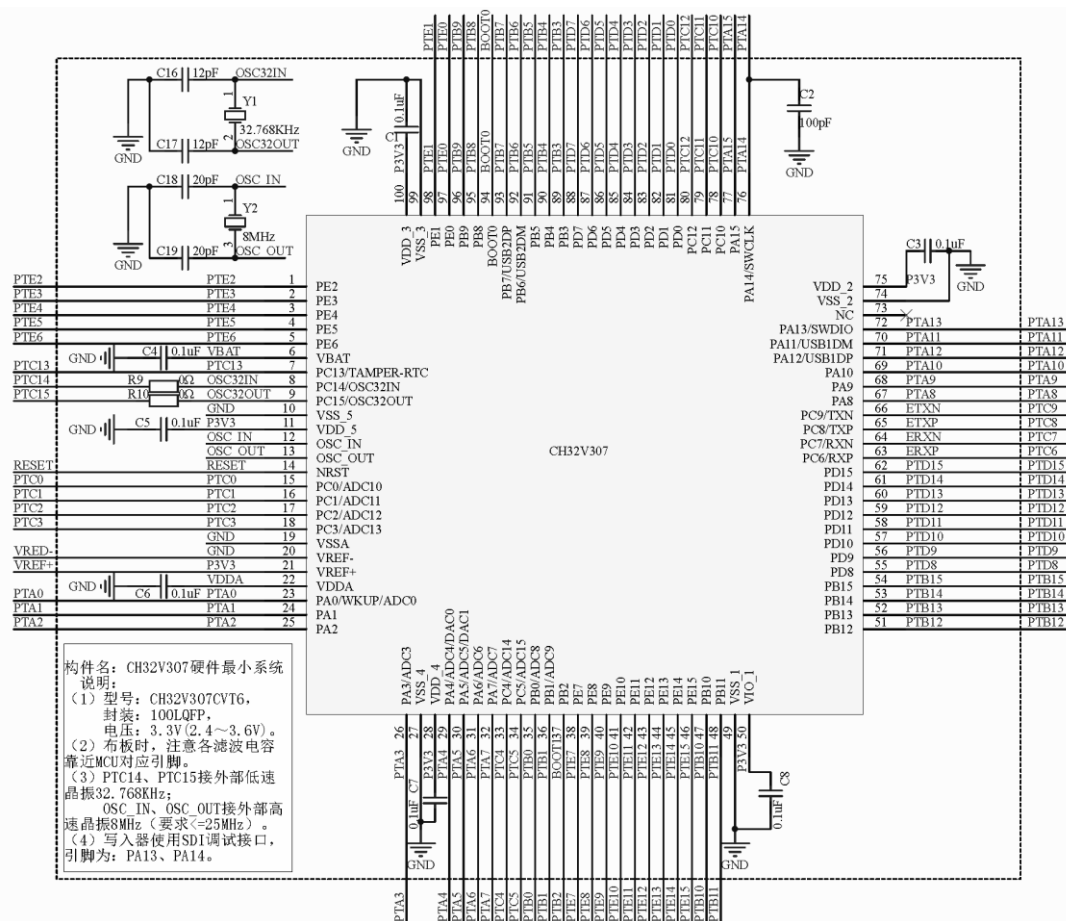
C	16	PTC[0-15]	PC14、PC15
D	16	PTD[0-15]	PTB2
E	16	PTE[0-15]	
合计	80		
说明	本书中所涉及的GPIO端口如PTA引脚与图3-1中的PA引脚同义，均可作为Port A的缩写。		

CH32V307VCT6（100 引脚 LQFP 封装）具有 80 个 I/O 引脚（包含两个 SDI 的引脚，两个外部低速晶振引脚，程序启动方式控制引脚：BOOT0 引脚），这些引脚均具有多个功能，在复位后，会立即被配置为高阻状态，且为通用输入引脚，有内部上拉功能。

【思考一下】 把 MCU 的引脚分为硬件最小系统引脚与对外提供服务引脚对嵌入式系统的硬件设计有何益处？

3.2.2 CH32V307硬件最小系统原理图

MCU 的硬件最小系统是指包括电源，晶振，复位，写入调试器接口等可使内部程序得以运行的、规范的、可复用的核心构件系统，它是设计人员要为芯片提供的硬件方面的最基本的服务。使用一个芯片，必须完全理解其硬件最小系统。当 MCU 工作不正常时，在硬件层面，应该检查硬件最小系统中可能出错的元件。芯片要能工作，必须有电源与工作时钟；至于复位电路则提供不掉电情况下 MCU 重新启动的手段。CH32V307VCT6 芯片的硬件最小系统包括电源电路、复位电路、晶振电路等。图 3-2 给出了 CH32V307VCT6 硬件最小系统原理图，读者需彻底理解该原理图的基本内涵。



则会引起 MCU 复位。一般芯片的复位引脚内部含有上拉电阻，若外部悬空，则上电的一瞬间，引脚为低电平，随后为高电平，这就是上电复位了。若外接一个按钮一端，按钮的另一端接地，这个按钮就称为**复位按钮**，可以从不同角度对复位进行基本分类。

(1) **外部复位和内部复位**。从引起 MCU 复位的内部与外部因素来区分，复位可分为外部复位和内部复位两种。外部复位有上电复位、按下“复位”按钮复位。内部复位有看门狗定时器复位、低电压复位、软件复位等。

(2) **冷复位和热复位**。从复位时芯片是否处于上电状态来区分，复位可分为冷复位和热复位。芯片从无电状态到上电状态的复位属于冷复位，芯片处于带电状态时的复位叫热复位。冷复位后，MCU 内部 RAM 的内容是随机的。而热复位后，MCU 内部 RAM 的内容会保持复位前的内容，即热复位并不会引起 RAM 中内容的丢失。

(3) **异步复位与同步复位**。从 CPU 响应快慢来区分，复位还可分为异步复位与同步复位。异步复位源的复位请求一般表示一种紧要的事件，因此复位控制逻辑会立即有效，不会等到当前总线周期结束后再复位。异步复位源有上电、低电压复位等。同步复位的处理方法与异步复位不同：当一个同步复位源给出复位请求时，复位控制器并不使之立即起作用，而是等到当前总线周期结束之后，这是为了保护数据的完整性。在该总线周期结束后的下一个系统时钟的上升沿时，复位才有效。同步复位源有看门狗定时器、软件等。

【思考一下】 实际编程时，有哪些方式判定热复位与冷复位？

3. 晶振电路

计算机的工作需要一个时间基准，这个时间基准由晶振电路提供。CH32V307VCT6 芯片可使用内部晶振和外部晶振两种方式为 MCU 提供工作时钟。

CH32V307VCT6 系列芯片含有内部高速时钟源，可以通过编程产生最高 144MHz 时钟频率，供系统总线及各个内部模块使用。使用内部时钟源可略去外部晶振电路。

若时钟源需要更高的精度，可自行选用外部晶振，例如，图 3-2 给出外接 8MHz 无源晶振的晶振电路接法，晶振连接在芯片晶振输入引脚（3 脚）与晶振输出引脚（4 脚）之间，根据用户所选晶体负载电容以晶体厂商建议为准。实际上，两个电容有一定偏差，否则晶振不会起振，而电容制造过程总会有一个微小的偏差，满足起振条件。若使用内部内部时钟源，这个外接晶振电路就可以不焊接。

芯片启动时，需要运行芯片时钟初始化程序，随后才能正常工作。这个程序比较复杂，放在第 12 章阐述。从第 4 章开始的所有程序，均有芯片工作时钟初始化过程，我们先用起来，然后再理解其编程细节。

【思考一下】 通过查阅资料，了解一下晶振有哪些类型，简述其工作原理。

3.3 由MCU构建通用嵌入式计算机

嵌入式计算机一般来说是一个微型计算机，目前嵌入式系统开发模式大多数是从“零”做起，也就是硬件从 MCU（或 MPU）芯片做起，软件从自启动开始，增加了嵌入式系统的学习与开发难度，存在软硬件开发存在颗粒度低、可移植性弱等问题。随着 MCU 性能

的不断提高及软件工程概念的普及,给解决这些问题提供了契机。若能向通用计算机那样,把做计算机与用计算机的工作相对分开,可以提高软件可移植性,降低嵌入式系统开发门槛,对嵌入式人工智能、物联网、智能制造等嵌入式应用领域将会形成有力推动。

3.3.1 嵌入式终端开发方式存在的问题与解决办法

1. 嵌入式终端UE开发方式存在的问题

微控制器 MCU 是嵌入式终端 UE 的核心,承担着传感器采样、滤波处理、融合计算、通信、控制执行机构等功能。MCU 生产厂家往往配备一本厚厚的参考手册,少则几百页,多则可达近千页。许多厂家也给出软件开发包 (Software Development Kit, SDK)。但是, MCU 的应用开发人员通常花费太多的精力在底层驱动上,终端 UE 的开发方式存在软硬件设计颗粒度低、可移植性弱等问题。

(1) 硬件设计颗粒度低。以窄带物联网 (Narrow Band Internet of Things, NB-IoT) 终端 (Ultimate-Equipment, UE) 为例说明硬件设计颗粒度问题。在通常 NB-IoT 终端 UE 的硬件设计中,首先选一款 MCU,选一款通信模组,选一家 eSIM 卡,根据终端 UE 的功能,开始了 MCU 最小系统设计、通信适配电路设计、eSIM 卡接口设计及其他应用功能设计,这里有许多共性可以抽取。

(2) 寄存器级编程,软件编程颗粒度低,门槛较高。MCU 参考手册属于寄存器级编程指南,是终端工程师的基本参考资料。例如,要完成一个串行通信,需要涉及波特率寄存器、控制寄存器、状态寄存器、数据寄存器等,一般情况下,工程师针对所使用的芯片,封装其驱动。即使利用厂家给出的 SDK,也需要一番周折。无论如何,有一定技术门槛,花费不少时间。此外,工程师面向个性产品制作,不具备社会属性,常常弱化可移植性。又比如,对 NB-IoT 通信模组,厂家提供的是 AT 指令,要想打通整个通信流程,需要花费一番功夫。

(3) 可移植性弱,更换芯片困难,影响产品升级。一些终端厂家的某一产品使用一个 MCU 芯片多年,有的芯片甚至已经停产,且价格较贵,但由于早期开发可移植性较弱,更换芯片需要较多的研发投入,因此,即使新的芯片性价比高,也较难更换。对于 NB-IoT 通信模组,如何做到更换其型号,而原来的软件不变,是值得深入分析思考的。

2. 解决终端开发方式颗粒度低与可移植性弱的基本方法

针对嵌入式终端 UE 开发方式存在颗粒度低、可移植性弱的问题,必须探讨如何提高硬件颗粒度、如何提高软件颗粒度、如何提高可移植性,做到这三个“提高”,就可大幅度降低嵌入式系统应用开发的难度。

(1) 提高硬件设计的颗粒度。若能将 MCU 及其硬件最小系统、通信模组及其适配电路、eSIM 卡及其接口电路,做成一个整体,则可提高 UE 的硬件开发颗粒度。硬件设计也应该从元件级过度到硬件构件为主,辅以少量接口级、保护级元件,以提高硬件设计的颗粒度。

(2) 提高软件编程颗粒度。针对大多数以 MCU 为核心的终端系统,可以通过面向

知识要素角度设计底层驱动构件，把编程颗粒度从寄存器级提高到以知识要素为核心的构件级。以 GPIO 为例阐述这个问题。共性知识要素是：引脚复用成 GPIO 功能、初始化引脚方向；若定义成输出，设置引脚电平；若定义成输入，获得引脚电平等。寄存器级编程涉及引脚复用寄存器、数据方向寄存器、数据输出寄存器、引脚状态寄存器等。寄存器级编程因芯片不同，其地址、寄存器名字、功能而不同。可以面向共性知识要素编程，把寄存器级编程不同之处封装在内部，把编程颗粒度提高到知识要素级。

(3) 提高软硬件可移植性。特定厂家提供 SDK，也注意可移植性。但是由于厂家之间的竞争关系，其社会属性被弱化。因此，让芯片厂家工程师从共性知识要素角度封装底层硬件驱动，有些勉为其难。科学界必须从共性知识要素本身角度研究这个问题。把共性抽象出来，面向知识要素封装，把个性化的寄存器屏蔽在构件内部，这样才能使得应用层编程具有可移植性。在硬件方面，遵循硬件构件的设计原则，提高硬件可移植性。

3.3.2 提出GEC概念的时机、GEC定义与特点

1. 提出GEC概念的时机

要能够做到提高编程颗粒度、提高可移植性，可以借鉴通用计算机（General Computer）概念与做法，在一定条件下，做通用嵌入式计算机（General Embedded Computer, GEC），把基本输入输出系统（Basic Input and Output System, BIOS）与用户程序分离开来，实现彻底的工作分工。GEC 虽然不能涵盖所有嵌入式开发，但可涵盖其中大部分。

GEC 概念的实质是把面向寄存器编程提高到面向知识要素编程，提高了编程颗粒度。但是，这样做也会降低实时性。弥补实时性降低的方法是提高芯片的运行时钟频率。目前 MCU 的总线频率是早期 MCU 总线频率几十倍，甚至几百倍，因此，更高的总线频率给提高编程颗粒度提供了物理支撑。

另一方面是软件构件技术的发展与认识的普及，也为提出 GEC 概念提供了机遇。嵌入式软件开发人员越来越认识到软件工程对嵌入式软件开发的重要支撑作用，也意识到掌握和应用软件工程的基本原理对嵌入式软件的设计、升级、芯片迭代与维护等方面，具有不可或缺的作用。因此，从“零”开始的编程，将逐步分化为构件制作与构件使用两个不同层次，也为嵌入式人工智能提供先导基础。

2. GEC定义及基本特点

通用嵌入式计算机 GEC 定义。一个具有特定功能的通用嵌入式计算机（General Embedded Computer, GEC），体现在硬件与软件两个侧面。在硬件上，把 MCU 硬件最小系统及面向具体应用的共性电路封装成一个整体，为用户提供 SOC 级芯片的可重用的硬件实体，并按照硬件构件要求进行原理图绘制、文档撰写及硬件测试用例设计。在软件上，把嵌入式软件分为基本输入/输出系统（Basic Input/Output System, BIOS）程序与 User 程序两部分。BIOS 程序先于 User 程序固化于 MCU 内的非易失存储器（如 Flash）中，启动时，BIOS 程序先运行，随后转向 User 程序。BIOS 提供工作时钟及面向知识要素的底层驱动构件，并为 User 程序提供函数原型级调用接口。

与 MCU 对比，GEC 具有硬件直接可测性、用户软件编程快捷性与可移植性三个基本特点。

(1) GEC 硬件的直接可测性。与一般 MCU 不同，GEC 类似 PC 机，通电后可直接可运行内部 BIOS 程序，BIOS 驱动保留使用的小灯引脚，高低电平切换（在 GEC 上，可直接观察到小灯闪烁）。可利用 AHL-GEC-IDE 开发环境，使用串口连接 GEC，直接将 User 序写入 GEC，User 程序中包含类似于 PC 程序调试的 printf 语句，通过串口向 PC 机输出信息，实现了 GEC 硬件的直接可测性。

(2) GEC 用户软件的编程快捷性。与一般 MCU 不同，GEC 内部驻留的 BIOS 与 PC 机上电过程类似，完成系统总线时钟初始化；提供一个系统定时器，提供时间设置与获取函数接口；BIOS 内驻留了嵌入式常用驱动，如 GPIO、UART、ADC、Flash、I2C、SPI、PWM 等，并提供了函数原型级调用接口。利用 User 程序不同框架，用户软件不需要从“零”编起，而是在相应框架基础上，充分应用 BIOS 资源，实现快捷编程。

(3) GEC 用户软件的可移植性。与一般 MCU 软件不同，GEC 的 BIOS 软件由 GEC 提供者研发完成，随 GEC 芯片而提供给用户，即软件被硬件化了，具有通用性。BIOS 驻留了大部分面向知识要素的驱动，提供了函数原型级调用接口。在此基础上编程，只要遵循软件工程的基本原则，GEC 用户软件则具有较高的可移植性。

3.3.3 由CH32V307VCT6构成的GEC

本书以 CH32V307VCT6 为核心构建一种通用嵌入式计算机，命名为 AHL-CH32V307，作为本书的主要实验平台，在此基础上可以构建各种类型的 GEC。

1. AHL-CH32V307硬件系统基本组成

图 3-3 给出了 AHL-CH32V307 硬件图，内含 CH32V307VCT6 芯片及其硬件最小系统、三色灯、复位按钮、温度传感器、触摸区、两路 TTL-USB 串口，基本组成见表 3-7。

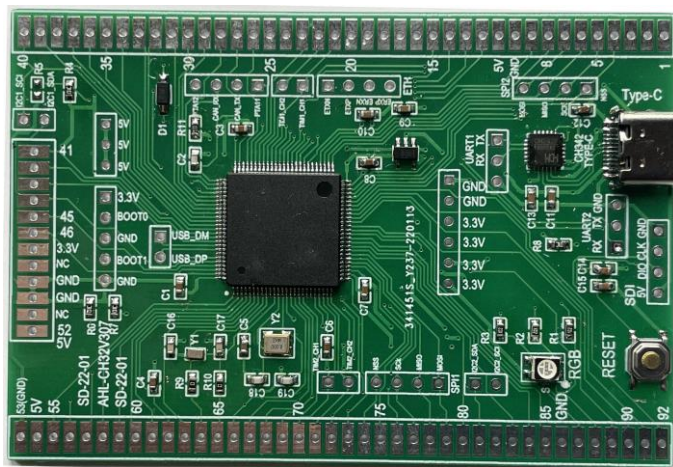


图3-3 AHL-CH32V307硬件图

下面对 AHL- CH32V307VCT6 中的三色灯、TTL-USB 串口等做一个简要说明。

表3-7 AHL- CH32V307VCT6的基本组成

序号	部件	功能说明
1	MCU	CH32V307VCT6芯片
2	三色灯	红、绿、蓝
3	TTL-USB	两路TTL串口电平转USB，与工具计算机通信，下载程序，用户串口
4	复位按钮	用户程序不能写入时，按此按钮6次以上，绿灯闪烁，可继续下载用户程序
5	温度传感器	测量环境温度
6	5V转3.3V电路	实验时通过Type-C线接PC机，5V引入本板，在板上转为3.3V给MCU供电
7	引出脚编号	1~92，把MCU的基本引脚全部再次引出，供应用开发者使用

(1) LED 三色灯。红 (R)、绿 (G)、蓝 (B) 三色灯电路原理图，如图 3-4 所示。三色灯的型号为 1SC3528VGB01MH08，内含红绿蓝三个发光二极管。图中，每个二极管的负极外接 1K Ω 限流电阻后接入 MCU 引脚，只要 MCU 内的程序，控制相应引脚输出低电平，对应的发光二极管就亮起来了，达到软件控制硬件的目的。

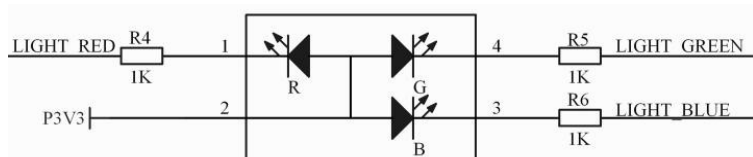


图3-4 三色灯电路图

【思考一下】 上网查一下三色灯 1SC3528VGB01MH08 的芯片手册，根据手册查看其内部发光二极管的额定电流是多少？为了延长三色灯的使用寿命，限流电阻应该适当增大，还是适当减少？限流电阻增大或减少带来的影响是什么？

(2) TTL-USB 串口。这个用于使用 Type-C 线将 GEC 与 PC 机的 USB 连接起来，实质是串行通信连接，PC 机使用 USB 接口模拟串口是为了方便，现在的 PC 机和笔记本电脑已经逐步没有串行通信接口，将在第 6 章对此进行阐述。这个 TTL-USB 串口提供了两路串口，一个用于下载用户与调试程序，一个供用户使用，第 6 章阐述其编程方法。

(3) 复位按钮。图 3-3 的左下部有个按钮，其作用是热复位。特别功能是，在短时间内连续按 6 次以上，GEC 进入 BIOS 运行状态，可以进行用户程序下载，仅用于解决 GEC 与开发环境连接不上时的写入操作问题。

2. AHL-CH32V307的对外引脚

基于 CH32V307VCT6 芯片制作通用嵌入式计算机 GEC 的硬件型号为：AHL-CH32V307，板载芯片硬件最小系统、三色灯、温度传感器等，引出了芯片的全部功能引脚，还增加了一些应用过程可能用到的接口，如表 3-8 所示。大部分是 MCU 的引脚直接引出，有的引脚功能被固定下来，在进行具体应用的硬件系统设计时可查阅此表。

表3-8 AHL-CH32V307的引脚复用功能

编号	特定功能	MCU引脚名	复用功能
1		PTE13	FSMC_D10
2		PTE14	FSMC_D11/OPA2_OUT1
3		PTE15	FSMC_D12/OPA1_OUT1
4	SPI2_NSS	PTB12	SPI2_NSS/I2S2_WS/I2C2_SMBA/USART3_CK/TIM1_BKIN/OPA4_CH0P/CAN2_RX/ETH_MII_TXD0/ETH_RMII_TXD0
5	SPI2_SCK	PTB13	SPI2_SCK/I2S2_CK/USART3_CTS/TIM1_CH1N/OPA3_CH0P/CAN2_TX/ETH_MII_TXD1/ETH_RMII_TXD1
6	SPI2_MISO	PTB14	SPI2_MISO/TIM1_CH2N/USART3_RTS/OPA2_CH0P
7	SPI2_MOSI	PTB15	SPI2_MOSI/I2S2_SD/TIM1_CH3N/OPA1_CH0P
8		PTD10	FSMC_D15
9		PTD11	FSMC_A16
10	GND		
11	5V		
12		PTD12	FSMC_A17
13		PTD13	FSMC_A18
14		PTD14	FSMC_D0
15		PTD15	FSMC_D1
16		RESET	
17	ETH_RXP	PTC6	I2S2_MCK/TIM8_CH1/SDIO_D6/ETH_RXP
18	ETH_RXN	PTC7	I2S3_MCK/TIM8_CH2/SDIO_D7/ETH_RXN
19	ETH_TXP	PTC8	TIM8_CH3/SDIO_D0/ETH_TXP/DVP_D2
20	ETH_TXN	PTC9	TIM8_CH4/SDIO_D1/ETH_TXN/DVP_D3
21	TIM1_CH1	PTA8	USART1_CK/TIM1_CH1/MCO
22	TIM1_CH2	PTA9	USART1_TX/TIM1_CH2/OTG_FS_VBUS/DVP_D0
23		PTA10	USART1_RX/TIM1_CH3/OTG_FS_ID/DVP_D1
24	CAN_RX		
25	CAN_TX		
26		PTA13	TIM10_CH2N
27		PTA14	TIM10_CH3N
28	SWCLK	PTA15	SPI3_NSS I2S3_WS
29		PTC12	UART5_TX/SDIO_CK/TIM10_BKIN/DVP_D9
30		PTD0	FSMC_D2
31		PTD1	FSMC_D3

32		PTD2	TIM3_ETR/UART5_RX/SDIO_CMD/DVP_D11
33		PTD3	FSMC_CLK
34		PTD4	FSMC_NOE
35		PTD5	FSMC_NWE
36		PTD6	FSMC_NWAIT/DVP_D10
37		PTD7	FSMC_NE1 FSMC_NCE2
38	USBHS_DP	PTB7	I2C1_SDA/FSMC_NADV/TIM4_CH2/USBHD_DP/USBHS_DP
39	USBHS_DM	PTB6	I2C1_SCL/TIM4_CH1/USBHD_DM/DVP_D5/USBHS_DM
40		PTB8	TIM4_CH3/SDIO_D4/TIM10_CH1/DVP_D6/ETH_MII_TXD3
41		PTB3	SPI3_SCK I2S3_CK
42		PTB4	SPI3_MISO
43		PTB5	I2C1_SMBA/SPI3_MOSI/I2S3_SD/ETH_MII_PPS_OUT/ETH_RMII_PPS_OUT
44	I2C1_SDA	PTB9	TIM4_CH4/SDIO_D5/TIM10_CH2/DVP_D7
45		PTE0	TIM4_ETR/FSMC_NBL0
46		PTE1	FSMC_NBL1
47	3.3V		
48	NC		
49	GND		
50	GND		
51	NC		
52	5V		
53	GND		
54	5V		
55		PTE2	FSMC_A23
56		PTC11	UART4_RX/SDIO_D3/TIM10_CH4/DVP_D4
57		PTC10	UART4_TX/SDIO_D2/TIM10_ETR/DVP_D8
58		PTE3	FSMC_A19
59		PTE4	FSMC_A20
60		PTE5	FSMC_A21
61		PTE6	FSMC_A22
62		VBAT	
63		PTC13	TAMPER-RTC
64		NC	

65		NC	
66		PTC0	ADC_IN10/TIM9_CH1N/UART6_TX/ETH_RGMII_RXC
67		PTC1	ADC_IN11/TIM9_CH2N/UART6_RX/ETH_MII_MDC/ETH_RMII_MDC/ETH_RGMII_RXCTL
68		PTC2	ADC_IN12/TIM9_CH3N/UART7_TX/OPA3_CH1N/ETH_MII_TXD2/ETH_RGMII_RXD0
69		PTC3	ADC_IN13/TIM10_CH3/UART7_RX/OPA4_CH1N/ETH_MII_TX_CLK/ETH_RGMII_RXD1
70	TIM2_CH1	PTA0	WKUP/USART2_CTS/ADC_IN0/TIM2_CH1/TIM2_ETR/TIM5_CH1/TIM8_ETR/OPA4_OUT0/ETH_MII_CRS_WKUP/ETH_RGMII_RXD2
71		PTA1	USART2_RTS/ADC_IN1/TIM5_CH2/TIM2_CH2/OPA3_OUT0/ETH_MII_RX_CLK/ETH_RMII_REF_CLK/ETH_RGMII_RXD3
72		PTA2	USART2_TX/TIM5_CH3/ADC_IN2/TIM2_CH3/TIM9_CH1/TIM9_ETR/OPA2_OUT0/ETH_MII_MDIO/ETH_RMII_MDIO/ETH_RGMII_GTXC
73		PTA3	USART2_RX/TIM5_CH4/ADC_IN3/TIM2_CH4/TIM9_CH2/OPA1_OUT0/ETH_MII_COL/ETH_RGMII_TXEN
74	SPI1_NSS	PTA4	SPI1_NSS/USART2_CK/ADC_IN4/DAC_OUT1/TIM9_CH3/DVP_HSYNC
75		PTD9	FSMC_D14
76		PTD8	FSMC_D13
77	SPI1_SCK	PTA5	SPI1_SCK/ADC_IN5/DAC_OUT2/OPA2_CH1N/DVP_VSYNC
78	SPI1_MISO	PTA6	SPI1_MISO/TIM8_BKIN/ADC_IN6/TIM3_CH1/OPA1_CH1N/DVP_PCLK
79	SPI1_MOSI	PTA7	SPI1_MOSI/TIM8_CH1N/ADC_IN7/TIM3_CH2/OPA2_CH1P/ETH_MII_RX_DV/ETH_RMII_CRS_DV/ETH_RGMII_TXD0
80		PTC4	ADC_IN14/TIM9_CH4/UART8_TX/OPA4_CH1P/ETH_MII_RXD0/ETH_RMII_RXD0/ETH_RGMII_TXD1
81	I2C2_SDA	PTB11	I2C2_SDA/USART3_RX/OPA1_CH0N/ETH_MII_TX_EN/ETH_RMII_TX_EN
82	I2C2_SCL	PTB10	I2C2_SCL/USART3_TX/OPA2_CH0N/ETH_MII_RX_ER
83		PTC5	ADC_IN15/TIM9_BKIN/UART8_RX/OPA3_CH1P/ETH_MII_RXD1 ETH_RMII_RXD1/ETH_RGMII_TXD2
84		PTB0	ADC_IN8/TIM3_CH3/TIM8_CH2N/OPA1_CH1P/ETH_MII_RXD2 ETH_RGMII_TXD3

85		PTB1	ADC_IN9/TIM3_CH4/TIM8_CH3N/OPA4_CH0N/ETH_MII_RXD3 ETH_RGMII_125IN
86		PTB2	OPA3_CH0N
87		PTE7	FSMC_D4/OPA3_OUT1
88		PTE8	FSMC_D5/OPA4_OUT1
89		PTE9	FSMC_D6
90		PTE10	FSMC_D7
91		PTE11	FSMC_D8
92		PTE12	FSMC_D9

此外，为了方便进行本书的实验，还在板中引出了 SPI、I2C、CAN、USB、ETH、SDI 等接口。

本章小结

1. 关于初识一个MCU

初识一个 MCU 首先要从认识型号标识开始，可以从型号标识中获得芯片家族、产品类型、具体特性、引脚数目、Flash 大小、温度范围、封装类型等信息，这些信息是购买芯片的基本要求；其次了解内部 RAM 及 Flash 的大小、地址范围，以便设置链接文件，为程序编译及写入做好准备；再次了解中断源及中断向量号，为中断编程做准备。

2. 关于硬件最小系统

一个芯片的硬件最小系统是指可以使内部程序运行所必须的最低规模的外围电路，也可以包括写入器接口电路。使用一个芯片，必须完全理解其硬件最小系统。硬件最小系统引脚是我们必须为芯片提供服务的引脚，包括电源、晶振、复位等，做好这些服务之后，其他引脚就为用户提供服务了。硬件最小系统电路中着重掌握电容滤波原理及布板时靠近对应引脚的基本要求。

3. 关于利用MCU构建通用嵌入式计算机

引入通用嵌入式计算机概念的目的不仅仅是为了降低硬件设计复杂度，更重要目的是降低软件开发难度。硬件上，使其只要供电就可工作，关键是其内部有 BIOS。BIOS 不仅可以驻留构件，还可以驻留实时操作系统，提供方便灵活的动态命令^①等等。在最小的硬件系统基础上，辅以各种无线通信等，可以形成不同应用的 GEC 系列，为嵌入式人工智能与物联网的应用提供技术基础。

^① 动态命令用于扩展嵌入式终端的非预设功能，用于深度嵌入式开发中，这里了解即可，不做深入阐述。

习 题

1. 举例说明，对照命名格式，从所用 MCU 芯片的芯片型号标识可以获得哪些信息？
2. 给出所学 MCU 芯片的 RAM 及 Flash 大小、地址范围。
3. 中断的定义是什么？什么是内核中断？什么是非内核中断？给出所学 MCU 芯片的中断个数。
4. 什么是芯片的硬件最小系统，它由哪几个部分组成？简要阐述各部分技术要点。
5. 谈谈你对通用嵌入式计算机 GEC 的理解。
6. 若不用 MCU 芯片的引脚直接连接三色灯，给出 MCU 引脚通过三极管控制三色灯的电路。

第4章 GPIO及程序框架

本章导读：本章是全书的重点和难点之一，需要深入透彻理解，达到快速且规范入门之目的。主要内容有：给出 GPIO 通用基础知识；给出以 GPIO 构件为基础的编程方法，这是最简单的嵌入式系统程序；讲述 GPIO 构件是如何制作出来的，这是第一个基础构件设计样例，有一定难度；给出汇编工程模板，利用汇编程序点亮一盏发光二极管，通过这个例程，可以更透彻地理解软件的软件是如何干预硬件的。

4.1 GPIO通用基础知识

GPIO 是嵌入式应用开发最常用的功能，用途广泛，编程灵活，是嵌入式编程的重点和难点之一，本节对 GPIO 作简要概述。

4.1.1 GPIO概念

输入/输出（Input/Output, I/O）接口，是 MCU 同外界进行交互的重要通道，MCU 与外部设备的数据交换通过 I/O 接口来实现。I/O 接口是一电子电路，其内部有若干专用寄存器和相应的控制逻辑电路构成。接口的英文单词是 interface，另一个英文单词是 port。但有时把 interface 翻译成“接口”，而把 port 翻译成“端口”，从中文字面看，接口与端口似乎有点区别，但在嵌入式系统中它们的含义是相同的。有时把 I/O 引脚称为接口（Interface），而把用于对 I/O 引脚进行编程的寄存器称为端口（Port），实际上它们是紧密相连的。因此，有些书中甚至直接称 I/O 接口（端口）为 I/O 口。在嵌入式系统中，接口种类很多，有显而易见的人机交互接口，如键盘、显示器，也有无人介入的接口，如串行通信接口、USB 接口、网络接口等。

通用 I/O 也记为 GPIO（General Purpose I/O），即基本输入/输出，有时也称并行 I/O，或普通 I/O，它是 I/O 的最基本形式。本书中使用正逻辑，电源（Vcc）代表高电平，对应数字信号“1”；地（GND）代表低电平，对应数字信号“0”。作为通用输出引脚，MCU 内部程序通过端口寄存器控制该引脚状态，使得引脚输出“1”（高电平）或“0”（低电平），即开关量输出。作为通用输入引脚，MCU 内部程序可以通过端口寄存器获取该引脚状态，以确定该引脚是“1”（高电平）或“0”（低电平），即开关量输入。大多数通用 I/O 引脚可以通过编程来设定其工作方式输入或输出，称之为双向通用 I/O。

4.1.2 输出引脚的基本接法

作为通用输出引脚，MCU 内部程序向该引脚输出高电平或低电平来驱动器件工作，即开关量输出，如图 4-1 所示。

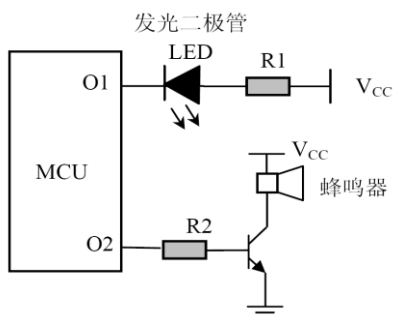


图4-1 通用I/O引脚输出电路

输出引脚 O1 和 O2 采用了不同的方式驱动外部器件，一种接法是 O1 直接驱动发光二极管 LED，当 O1 引脚输出高电平时，LED 不亮；当 O1 引脚输出低电平时，LED 点亮。这种接法的驱动电流一般在 $2\text{mA} \sim 10\text{mA}$ 。另一种接法是 O2 通过一个 NPN 三极管驱动蜂鸣器，当 O2 引脚输出高电平时，三极管导通，蜂鸣器响；当 O2 引脚输出低电平时，三极管截止，蜂鸣器不响。这种接法可以用 O2 引脚上的几个 mA 的控制电流驱动高达 100mA 的驱动电流。若负载需要更大的驱动电流，就必须采用光电隔离外加其他驱动电路，但对 MCU 编程来说，没有任何影响。

4.1.3 上拉下拉电阻与输入引脚的基本接法

芯片输入引脚的外部有三种不同的连接方式：带上拉电阻的连接、带下拉电阻的连接和“悬空”连接。通俗地说，若 MCU 的某个引脚通过一个电阻接到电源（ V_{CC} ）上，这个电阻被称为“上拉电阻”；与之相对应，若 MCU 的某个引脚通过一个电阻接到地（GND）上，则相应的电阻被称为“下拉电阻”。这种做法使得，悬空的芯片引脚被上拉电阻或下拉电阻初始化为高电平或低电平。根据实际情况，上拉电阻与下拉电阻可以取值在 $1\text{K}\Omega \sim 10\text{K}\Omega$ 之间，其阻值大小与静态电流及系统功耗有关。

图 4-2 给出了一个 MCU 的输入引脚的三种外部连接方式，假设 MCU 内部没有上拉

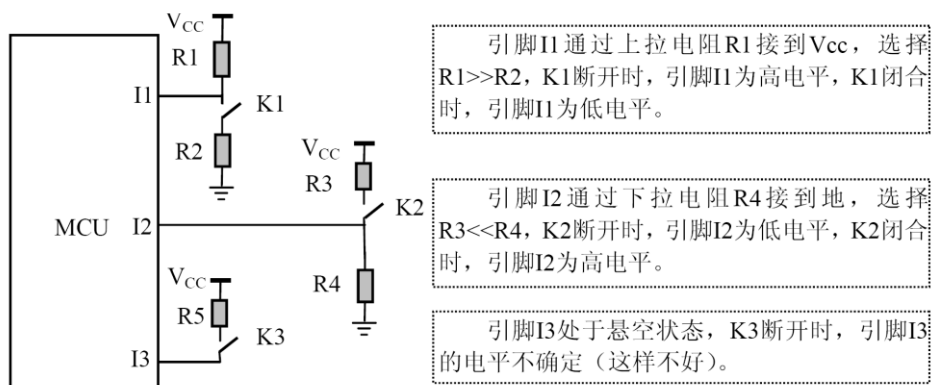


图4-2 通用I/O引脚输入电路接法举例

或下拉电阻，图中的引脚 I3 上的开关 K3 采用悬空方式连接就不合适，因为 K3 断开时，引脚 I3 的电平不确定，图中， $R1 \gg R2$, $R3 \ll R4$ ，各电阻的典型取值为： $R1=10K\Omega$, $R2=200\Omega$, $R3=200\Omega$, $R4=10K\Omega$ 。

【思考一下】 上拉电阻的实际取值如何确定？

4.2 软件干预硬件的方法

本节以 GPIO 构件为基础的样例工程“..\04-Software\CH04\GPIO-Output-Component”为例说明软件如何干预硬件的。关于软硬件构件的概念将在第 5 章阐述。

4.2.1 GPIO构件API

嵌入式系统的重要特点是软件硬件相结合，通过软件获得硬件的状态，通过软件控制硬件的动作。通常情况下，软件与某一硬件模块打交道通过其底层**驱动构件**，也就是封装好的一些函数，编程时通过调用这些函数，干预硬件。这样就把**制作构件**与**使用构件**的工作分成不同过程。就像建设桥梁，先做标准预制板一样，这个标准预制板就是构件。

1. 软件是如何干预硬件？

现在先来看看软件是如何干预硬件的。例如想点亮图 3-4 中的蓝色 LED 小灯，由该电路原理图可以看出，只要使得标识 LIGHT_BLUE 的引脚为低电平，蓝色 LED 就可以亮起来。为了能够做到软件干预硬件，必须将该引脚与 MCU 的一个具有 GPIO 功能的引脚连接起来，通过编程使得 MCU 的该引脚电平为电平（逻辑 0），蓝色 LED 就亮起来了，这就是**软件干预硬件的基本过程**。

而要编程使得具有 GPIO 功能的引脚为低电平，若采用从“零”开始编程的方法，要了解该引脚在哪个端口，端口都有哪些寄存器，每个寄存器相应二进制位的含义，还要了解编程步骤，等等，这个过程对一般读者或初学者十分困难，4.4 节将会描述这个过程。现在，可以利用已经做好的 GPIO 构件，先把 LED 小灯点亮，然后根据不同学习要求，再理解构件是如何做出来的。

通常情况下，应用程序开发人员使用**驱动构件**与具体的硬件打交道。每个驱动构件均含有若干函数，例如 GPIO 构件具有**初始化**、**设定引脚状态**、**获取引脚状态**等函数，使用构件可通过**应用程序接口**（Application Programming Interface, API）使用这些函数，**也就是调用函数名，并使其参数实例化**。所谓**驱动构件的 API**是指应用程序与构件之间的衔接约定，使得应用程序开发人员通过它干预硬件，而无需理解其内部工作细节。

2. GPIO构件的常用函数

GPIO 构件主要 API 有：GPIO 的初始化、设置引脚状态、获取引脚状态、设置引脚中断等等。表 4-1 给出了 GPIO 常用接口函数简明列表，这些函数声明放在头文件 gpio.h 中，构件头文件是构件的使用说明。

表4-1 GPIO常用接口函数简明列表

序号	函数名	简明功能	描述
1	gpio_init	引脚初始化	引脚复用为GPIO功能；定义其为输入或输出；若为输出，还给出其初始状态
2	gpio_set	设定引脚状态	在GPIO输出情况下，设定引脚状态（高/低电平）
3	gpio_get	获取引脚状态	在GPIO输入情况下，获取引脚状态（1/0）
4	gpio_reverse	反转引脚状态	在GPIO输出情况下，反转引脚状态
5	gpio_pull	设置引脚上/下拉	当GPIO输入情况下，设置引脚上/下拉
6	gpio_enable_int	使能引脚中断	当GPIO输入情况下，使能引脚中断
7	gpio_disable_int	关闭引脚中断	当GPIO输入情况下，关闭引脚中断
	...		

3. GPIO构件的头文件gpio.h

头文件 gpio.h 中包含的主要内容有：头文件说明、防止重复包含的条件编译代码结构“#ifndef ...#define ...#endif”、有关宏定义、构件中各函数的 API 及使用说明等。这里给出 GPIO 初始化及设置引脚状态函数的 API，其他函数 API 参见样例工程源码电子文档。

```
// =====
// 文件名称： gpio.h
// 功能概要： GPIO底层驱动构件头文件
// 版权所有：苏州大学嵌入式系统与物联网研究所（http://sumcu.suda.edu.cn）
// 版本更新：20210520-20220219
// 芯片类型：CH32V307
// =====

#ifndef GPIO_H          // 防止重复定义（GPIO_H 开头）
#define GPIO_H

...

// 端口号地址偏移量宏定义
#define PTA_NUM    (0<<8)
#define PTB_NUM    (1<<8)
#define PTC_NUM    (2<<8)
#define PTD_NUM    (3<<8)
#define PTE_NUM    (4<<8)

// GPIO引脚方向宏定义
#define GPIO_INPUT    (0)        // GPIO输入
#define GPIO_OUTPUT    (1)       // GPIO输出
```



```

// =====
// 函数名称: gpio_init
// 函数返回: 无
// 参数说明: port_pin: (端口号)|(引脚号) (如: (PTB_NUM)|(9) 表示为B口9号脚)
//           dir: 引脚方向 (0=输入, 1=输出, 可用引脚方向宏定义)
//           state: 端口引脚初始状态 (0=低电平, 1=高电平)
// 功能概要: 初始化指定端口引脚作为GPIO引脚功能, 并定义为输入或输出, 若是输出,
//           还指定初始状态是低电平或高电平
// =====
void gpio_init(uint16_t port_pin, uint8_t dir, uint8_t state);

// =====
// 函数名称: gpio_set
// 函数返回: 无
// 参数说明: port_pin: (端口号)|(引脚号) (如: (PTB_NUM)|(9) 表示为B口9号脚)
//           state: 希望设置的端口引脚状态 (0=低电平, 1=高电平)
// 功能概要: 当指定端口引脚被定义为GPIO功能且为输出时, 本函数设定引脚状态
// =====
void gpio_set(uint16_t port_pin, uint8_t state);
...
#endif // 防止重复定义 (GPIO_H 结尾)

```

通过底层驱动构件干预硬件的编程相对来说就简单多了, 下面给出如何通过构件点亮一盏小灯。

4.2.2 第一个C语言工程: 控制小灯闪烁

在开发套件的底板上, 有红绿蓝三色灯 (合为一体的), 若使用 GPIO 构件实现蓝灯闪烁, 具体实例可参考 “..\04-Software\CH04\GPIO-Output-Component”, 步骤如下:

第一步, 给小灯起个名字。要用宏定义方式给蓝灯起个英文名 (如 LIGHT_BLUE), 明确蓝灯接在芯片的哪个 GPIO 引脚。根据硬件电路 “..\03-Hardware\03_component.pdf” 文件找到 AHL-CH32V307 中三色灯受 MCU 哪些引脚控制, 由此对灯所接引脚进行宏定义。由于这个工作属于用户程序, 按照 “分门别类, 各有归处” 之原则, 这个宏定义应该写在工程的 05_UserBoard\user.h 文件中。

```

// 指示灯端口及引脚定义
#define LIGHT_BLUE (PTC_NUM|2) // 蓝灯所在引脚

```

第二步, 给灯状态命名。由于灯的亮暗状态所对应的逻辑电平是由物理硬件接法决定, 为了应用程序的可移植性, 需要在工程的 05_UserBoard\user.h 文件中, 对蓝灯的 “亮”、“暗” 状态进行宏定义, 根据三色灯的实际硬件电路, 宏定义如下:

```
// 灯状态宏定义（灯的亮暗对应的逻辑电平，由物理硬件接法决定）
```

```
#define LIGHT_ON 0 // 灯亮
```

```
#define LIGHT_OFF 1 // 灯暗
```

特别说明：对灯的“亮、暗”状态使用宏定义，不仅是为了编程更加直观，也是为了使得软件能够更好地适应硬件。若硬件电路变动了，采用灯的“暗”状态对应低电平，那么只要改变本头文件中的宏定义就可以，而程序源码则不需更改。

【思考一下】 若灯的亮暗不使用宏定义会出现什么情况？有何不妥之处？

第三步，初始化蓝灯。在工程的 07-AppPrg\main.c 文件中，对蓝灯进行编程控制。先将蓝灯初始化为暗，在“用户外设模块初始化”处增加下列语句：

```
gpio_init(LIGHT_BLUE, GPIO_OUTPUT, LIGHT_OFF); // 初始化蓝灯，输出，暗
```

其中 GPIO_OUTPUT 是在 GPIO 构件中，对 GPIO 输出的宏定义，是为了编程直观方便。不然我们很难区分“1”是输出，还是输入。

```
#define GPIO_INPUT (0) // GPIO输入
```

```
#define GPIO_OUTPUT (1) // GPIO输出
```

特别说明：在嵌入式软件设计中，输入还是输出，是站在 MCU 角度，也就是站在 GEC 角度。要控制蓝灯亮暗，对 GEC 引脚来说，就是输出。若要获取外部状态到 GEC 中，对 GEC 来说，就是输入。例如，获取磁开关传感器的状态就需要初始化 GPIO 引脚为输入。

第四步，改变蓝灯亮暗状态。在 main 函数的主循环中，利用 GPIO 构件中的 gpio_set 函数，改变蓝灯状态。工程编译生成可执行文件后，写入目标板，可观察到蓝灯实际闪烁情况，部分程序摘录如下。

```
// (2.3.2) 如灯状态标志mFlag为'L'，灯的闪烁次数+1并显示，改变灯状态及标志
```

```
if (mFlag=='L') //判断灯的状态标志
```

```
{
```

```
    mLightCount++;
```

```
    printf(" mLightCount = %d\r\n",mLightCount);
```

```
    mFlag='A'; //灯的状态标志
```

```
    gpio_set(LIGHT_BLUE,LIGHT_ON); //灯“亮”
```

```
    printf(" LIGHT_BLUE:ON--\r\n"); //串口输出灯的状态
```

```
    printf(" \r\n");
```

```
}
```

```
// (2.3.3) 如灯状态标志mFlag为'A'，改变灯状态及标志
```

```
else
```

```
{
```

```
    mFlag='L'; //灯的状态标志
```

```
    gpio_set(LIGHT_BLUE,LIGHT_OFF); //灯“暗”
```

```
    printf(" LIGHT_BLUE:OFF--\r\n"); //串口输出灯的状态
```

```
}
```

第五步，观察蓝灯运行情况。经过编译生成机器码，通过 AHL-GEC-IDE 软件将 hex 文件下载到目标板中，可观察板载蓝灯每秒闪烁一次，也可在 AHL-GEC-IDE 界面看到蓝灯状态改变的信息，如图 4-3 所示。由此可体会，使用 printf 语句进行调试的好处。



图4-3 GPIO构件的输出测试方法

到这里看到了小灯在闪烁，这就是编程控制的，软件控制了硬件的动作。由此可以体会程序在现代控制系统中的作用。随着逐步深入的学习，可以看到更多更复杂的软件干预硬件的实例。

【思考一下】利用 AHL-GEC-IDE 集成开发环境，对 AHL-CH32V307 硬件上的三色灯编程，使三色灯以紫色形式闪烁。

为了规范地编程，提高程序的可靠性、可移植性与可维护性，我们把每个程序慎重地作为一个工程来对待，既然是个工程，就要有规范的工程框架，下一节将阐述它。

4.3 认识工程框架

4.3.1 工程框架及所含文件简介

嵌入式系统工程包含若干文件，包括程序文件、头文件、与编译调试相关的文件、工程说明文件、开发环境生成文件等，文件众多，合理组织这些文件，规范工程组织，可以提高项目的开发效率、提高阅读清晰度、提高可维护性、降低维护难度。工程组织应体现嵌入式软件工程的基本原则与基本思想。这个工程框架也可被称为软件最小系统框架，因为它包含的工程的最基本要素。软件最小系统框架是一个能够点亮一个发光二管的，甚至带有串口调试构件的，包含工程规范完整要素的可移植与可复用的工程模板。

该工程模板简洁易懂，去掉了一些初学者不易理解或不必要的文件，同时应用底层驱动构件化的思想改进了程序结构，重新分类组织了工程，目的是引导读者进行规范的文件组织与编程。

1. 工程名与新建工程

工程名使用工程文件夹标识工程，不同工程文件夹就区别不同工程。这样工程文件夹内的文件中所含的工程名字不再具有标识意义，可以修改，也可以不修改。建议新工程文件夹使用手动复制标准模板工程文件夹或复制功能更少的旧标准工程的方法来建立，这样，复用的构件已经存在，框架保留，体系清晰。不推荐使用 IDE 或其他开发环境的新建功能来建立一个新工程。

2. 工程文件夹内的基本内容

工程文件夹内编号的共含 7 个下级文件夹，除去 AHL_GEC_IDE 环境保留的文件夹 Debug，分别是 01_Doc、02_CPU、03_MCU、04_GEC、05_UserBoard、06_SoftComponent、07_AppPrg，其简明功能及特点见表 4-2。

表4-2 工程文件夹内的基本内容

名称	文件夹		简明功能及特点
文档文件夹	01_Doc		文档文件夹，工程改动时，及时记录。
CPU 文件夹	02_CPU		CPU 文件夹，与内核相关的文件。
MCU 文件夹	03_MCU	linker_File	链接文件夹，存放链接文件。
		MCU_drivers	MCU 底层构件文件夹，存放芯片级硬件驱动。
		startup	启动文件夹，存放芯片头文件及芯片初始化文件。
GEC 相关文件夹	04_GEC		GEC 芯片相关文件夹，存放引脚头文件。
用户板文件夹	05_UserBoard		用户板文件夹，存放应用构件。
软件构件文件夹	06_SoftComponent		软件构件文件夹，存放与硬件无关的软件构件。
源程序文件夹	07_AppPrg	include.h	总头文件，包含各类宏定义。
		isr.c	中断服务例程文件，存放各中断服务例程。
		main.c	主程序文件，存放芯片启动后的入口函数 main。

3. CPU（内核）相关文件简介

CPU（内核）相关文件（core_riscv.h、core_riscv.c、cpu.h）位于工程框架的“..\02_CPU”文件夹内，它们是沁恒（WCH）公司提供的符合 RISC-V 微控制器的内核相关文件，原则上与具体芯片制造商无关。其中 core_riscv.h 为 RISC-V 内核的外设访问层头文件。对任何使用该 CPU 设计的芯片，该文件夹内容相同。

4. MCU（芯片）相关文件简介

MCU（芯片）相关文件（startup_ch32v30x.s、ch32v30x.h、system_ch32v30x.h、system_ch32v30x.c）位于工程框架的“..\03_MCU\startup”文件夹内。由芯片厂商提供。

芯片头文件 ch32v30x.h 文件中，给出了芯片专用的寄存器地址映射，设计面向直接硬件操作的底层驱动时，利用该文件使用映射寄存器名，获得对应地址。该文件一般由芯片设计人员提供，一般嵌入式应用开发者不必修改该文件，只需遵循其中的命名。

启动文件 startup_ch32v30x.s，包含中断向量表。

系统初始化文件 `system_ch32v30x.h`、`system_ch32v30x.c`，主要存放启动文件 `startup_ch32v30x.s` 中调用的系统初始化函数 `SystemInit()` 及其相关宏常量的定义，此函数实现关闭看门狗及配置系统工作时钟的功能。

5. 应用程序源代码文件—总头文件 `includes.h`、`main.c` 及中断服务例程文件 `isr.c`

在工程的 `07_AppPrg` 文件夹内，放置着总头文件 `includes.h`、`main.c` 及中断服务例程文件 `isr.c`。

总头文件 `includes.h` 是 `main.c` 使用的头文件，内含常量、全局变量声明、外部函数及外部变量的引用。

主程序文件 `main.c` 是应用程序的启动后总入口，`main` 函数即在该文件中实现。在 `main` 函数中包含了一个永久循环，对具体事务过程的操作几乎都是添加在该主循环中。应用程序的执行，一共两条独立的线路，这是一条运行路线。另一条是中断线，在 `isr.c` 文件中编程。若有操作系统，则在这里启动操作系统调度器。

中断服务例程文件 `isr.c` 是中断处理函数编程的地方，有关中断编程问题将在 6.4 节中阐述。

6. 编译链接产生的其他相关文件简介

映像文件 (`.map`) 与列表文件 (`.lst`) 位于工程的 `Debug` 文件夹中，由编译链接产生。`.map` 文件提供了查看程序、堆栈设置、全局变量、常量等存放的地址信息。`.map` 文件中指定的地址在一定程度上是动态分配的（由编译器决定），工程有任何修改，这些地址都可能发生变动。`.lst` 文件提供了函数编译后，机器码与源代码的对应关系，用于程序分析。

4.3.2 了解机器码文件及芯片执行流程简析

这一小节有一点难度，供希望了解完整启动过程的读者阅读，本节对理解启动过程十分有益。若直接从 `main` 函数理解程序运行过程的读者可以跳过本节。

在 `AHL_GEC_IDE` 开发环境，针对 `CH32V307` 系列 MCU，在编译链接过程中生成针对 `RISC-V CPU` 的 `.elf` 格式可执行代码，同时也可生成十六进制 (`.hex`) 格式的机器码。

`.elf` (Executable and Linking Format)，即“可执行链接格式”，最初由 `UNIX` 系统实验室 (`UNIX System Laboratories, USL`) 作为应用程序二进制接口 (`Application Binary Interface, ABI`) 的一部分而制定和发布的。其最大特点在于它有比较广泛的适用性，通用的二进制接口定义使之可以平滑地移植到多种不同的操作环境上。`UltraEdit` 软件工具查看 `.elf` 文件内容。

`.hex` (`Intel HEX`) 文件是由一行行符合 `Intel HEX` 文件格式的文本所构成的 `ASCII` 文本文件，在 `Intel HEX` 文件中，每一行包含一个 `HEX` 记录，这些记录由对应机器语言码（含常量数据）的十六进制编码数字组成。

1. 记录格式

`.hex` 文件中的语句有六种不同类型的语句，但总体格式是一样的，根据表 4-3 格式来

记录。

表4-3 .hex文件记录行语义

	字段1	字段2	字段3	字段4	字段5	字段6
名称	标记	长度	偏移量	类型	数据/信息	校验和
长度	1字节	1字节	2字节	1字节	N字节	1字节
内容	开始标记 “:”		数据类型记录有效；非数据类型，该字段为“0000”。	00-数据记录； 01-文件结束记录； 02-扩展段地址； 03-开始段地址； 04-扩展线性地址； 05-链接开始地址。	取决于记录类型	开始标记之后字段的所有字节之和的补码。 校验和=0xFF-(记录长度+记录偏移+记录类型+数据段)+0x01

2. 实例分析

表4-4 GPIO-Output-Component-CH32V307.hex 文件部分行分解

行	记录标记	记录长度	偏移量	记录类型	数据/信息区	校验和
1	:	02	无效	02	1000	EC
2	:	10	E210	00	13000000130000001300000013000000	B2
...						
710	:	00	0000	01		FF

以“..\04-Software\CH04\GPIO-Output-Component”工程中的.hex 为例，截取该文件中的部分行进行简明分解，如表 4-4 所示。

第 1 行：“:020000021000EC”，行语义分割来看“: 02 0000 02 1000 EC”，以“:”开始，02 表示长度为 2 字节，“0000”对于非数据类型无效，紧接着的“02”代表记录类型为扩展段地址，“1000”，即随后记录的偏移地址需加上 0x10000，才是实际物理地址，“EC”为校验和。

第 2 行：“:10C40006F10200D13000000130000001300000047”，进行语义分割后来看“: 10 C400 00 6F10200D130000001300000013000000 47”，具体分析如下。

以“:”开始，长度为“0x10”（16 个字节），“C400”表示偏移量，实际地址为：0x10000+0xC400=0x1C400，紧接着的“00”代表记录类型为数据类型，接下来的就是数据段“6F10200D130000001300000013000000”，以 4 个字为划分，第一个四字节为“6F10200D”，由于是小端方式存储，这个数按照阅读习惯应写为“0D20106F”。可以到“..\Debug 下的 .lst”文件中，查找到 1C400 地址处，可以看到“1c400: 0d20106f j 1d4d2 <handle_reset>”，就是“..\03_MCU\startup\startup_ch32v30x.S”文件中“j handle_reset”语句，这是复位后要执行的第一个语句，于是转向了 handle_reset 处执行。从这里看到转到了地址 1d4d2 处，也就标号 handle_reset 代表的地址。可以到 startup_ch32v30x.S 文件中看看 handle_reset 标号后是什么语句，那就是由此开始运行程序啦。

第 710 行：即最后一行，为文档的结束记录，记录类型为“0x01”；“0xFF”为本记录的校验和字段内容。

综合分析工程的.map 文件、.ld 文件、.hex 文件、.lst 文件，可以理解程序的执行过程，也可以对生成的机器码进行分析对比。

3. 芯片执行流程简析

芯片复位到 main 函数之前程序运行过程总结如下。

芯片的 BOOT0 与 BOOT1 引脚决定芯片从 Flash 存储器、系统存储器或 SRAM 启动，本书的实例均将 BOOT0 和 BOOT1 接地，即从 Flash 存储器启动。当芯片从 Flash 存储器启动时，Flash 存储器地址被映射到 0x00000000 地址区域。内核寄存器 PC（程序计数器）执行 0x00000000 处指令，启动 BIOS，由于本程序为基于 BIOS 的 User 程序，重新转向 User 的 Flash 中的复位函数 handle_reset 的首地址，因而运行 handle_reset 函数。复位程序 handle_reset 在“..\03_MCU\startup\startup_ch32v30x.S”文件中，首先初始化全局指针（Global Pointer, GP），GP 可以优化±2KB 内全局变量的访问；随后初始化堆栈指针（Stack Pointer, SP）；接着将存有初值的全局变量和静态变量内容从 Flash 复制到 RAM 中，清零未初始化 BSS 数据段；设置运行模式为机器模式并打开中断；最后对芯片的中断向量表、系统时钟及看门狗进行初始化。对于 User 程序，还要判断其是否需要使用 BIOS 的中断服务例程。完成了这些工作，芯片就可以跳转到 main 函数中运行了。一般情况下，认为程序从 main 开始运行。

实际应用中，可根据是否启动看门狗、是否复制中断向量表至 RAM、是否清零未初始化 BSS 数据段等要求来修改此文件。初学者，在未理解相关内容情况下，不建议修改 startup_ch32v30x.s 及 system_ch32v30xx.c 文件内容。

需要说明的是，虽然本书给出的例程基于 BIOS，但不影响基本流程的理解，User 程序只要改变 Flash 首地址、RAM 首地址，即可从空白片写入运行，但不建议这样做，否则 BIOS 就被覆盖。此外，希望深入理解链接文件内容的读者，可参阅电子资源中补充阅读材料。

【思考一下】 综合分析.hex 文件、.map 文件、.lst 文件，在第一个样例工程中找出，SystemInit 函数、main 函数的存放地址，给出各函数前 16 个机器码，并找到其在.hex 文件中位置。

4.4 GPIO 构件的制作过程

这一节阐述 GPIO 构件是如何制作出来的，这是第一个基础构件设计样例，有一定难度，读者可根据所希望达到的学习深度，确定对本节相关内容的学习。构件的制作过程主要是与 MCU 内部模块寄存器（映像寄存器）打交道，大部分细节涉及到寄存器的某一位，程序就是通过寄存器的位干预相应硬件的。

4.4.1 端口与GPIO模块—对外引脚与内部寄存器

CH32V307VCT6 的大部分引脚具有多重复用功能，可以通过对相关寄存器编程来设定使用其中某一种功能，本节给出作为 GPIO 功能时的所用到的寄存器。

1. CH32V307VCT6芯片的GPIO引脚概述

100 引脚封装的 CH32V307VCT6 芯片的 GPIO 引脚分为 5 个端口，标记为 A、B、C、D、E，共含 80 个引脚。端口作为 GPIO 引脚时，逻辑 1 对应高电平，逻辑 0 对应着低电平。GPIO 模块使用系统时钟，从实时性细节来说，当作为通用输出时，高/低电平出现在时钟上升沿。下面给出各口可作为 GPIO 功能的引脚数目及引脚名称：

- (1) A 口有 16 个引脚，分别记为 PTA[0~15]；
- (2) B 口有 16 个引脚，分别记为 PTB[0~15]；
- (3) C 口有 16 个引脚，分别记为 PTC[0~15]；
- (4) D 口有 16 个引脚，分别记为 PTD[0~15]；
- (5) E 口有 16 个引脚，分别记为 PTE[0~15]；

2. GPIO寄存器概述

每个 GPIO 端口包含 7 个 32 位寄存器，分别是低位配置寄存器、高位配置寄存器、输入数据寄存器、输出数据寄存器、置位/复位寄存器、复位寄存器和锁定配置寄存器。如表 4-5 所示。A 口寄存器的基地址为 0x4001_0800，也就是低位配置寄存器的地址，高位配置寄存器的地址则是在低位配置寄存器的地址加 4 字节，其他寄存器的地址顺序加 4 字节。B 口的基地址为 A 口基地址加 0x0000_0400 为 0x4001_0C00，其他口基地址顺推。A、B、C、D 口低位配置寄存器、高位配置寄存器复位值均为 0x44444444。输入数据寄存器为只读寄存器，复位时为 0x0000xxxx；其他口寄存器复位时均为 0x00000000。

表4-5 A口寄存器

类型	绝对地址	寄存器名	R/W	功能简述
配置寄存器	0x4001_0800	低位配置寄存器（R32_GPIOA_CFGLR）	R/W	配高/低位引脚输入
	0x4001_0804	高位配置寄存器（R32_GPIOA_CFGHR）	R/W	输出模式和输出速度
数据寄存器	0x4001_0808	输入数据寄存器（R32_GPIOA_INDR）	R	读取输入引脚电平
	0x4001_080C	输出数据寄存器（R32_GPIOA_OUTDR）	R/W	读取输出引脚电平
其他寄存器	0x4001_0810	置位/复位寄存器（R32_GPIOA_BSHR）	W	置位/复位输出引脚
	0x4001_0814	复位寄存器（R32_GPIOA_BCR）	W	复位输出引脚电平
	0x4001_0818	锁定配置寄存器（R32_GPIOA_LCKR）	R/W	锁定引脚配置
复用寄存器	0x4800_0020	事件控制寄存器（R32_AFIO_ECR）	R/W	内核输出端口和引脚
	0x4800_0024	重映射寄存器（R32_AFIO_PCFR）	R/W	引脚功能复用
	0x4800_0028	外部中断配置寄存器（R32_AFIO_EXTICR1）	R/W	外部中断引脚配置

以下分别介绍这几个重要的寄存器。

3. 配置寄存器

1) GPIO配置寄存器低位 (GPIO_x_CFGLR) (x=A~D)

该寄存器用于配置GPIO端口相应引脚的工作模式,可以配置为输入模式、输出模式、输出速度。

数据位	D31~D30	D29~D28	...	D3~D2	D1~D0
读					
写	CNF7[1:0]	MODE7[1:0]	...	CNF0[1:0]	MODE0[1:0]

D31~D0 (CNFy[1:0], y∈[0,7]): x 端口 y 引脚配置位。这些位通过软件写入,用于配置 I/O 模式。在输入模式时 (MODE=00b) 00: 模拟输入模式; 01: 浮空输入模式; 10: 带有上下拉模式; 11: 保留。在输出模式时 (MODE>00b) 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。D31~D0 (MODE[1:0],y∈[0,7]): x 端口 y 引脚配置位。这些位通过软件写入,用于配置 I/O 模式。00: 输入模式; 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。

2) GPIO配置寄存器高位 (GPIO_x_CFGHR) (x=A~D)

该寄存器用于配置GPIO端口相应引脚的工作模式,可以配置为输入模式、输出模式、输出速度。

数据位	D31~D30	D29~D28	...	D3~D2	D1~D0
读					
写	CNF15[1:0]	MODE15[1:0]	...	CNF0[8:0]	MODE8[1:0]

D31~D0 (CNFy[1:0], y∈[8,15]): x 端口 y 引脚配置位。这些位通过软件写入,用于配置 I/O 模式。在输入模式时 (MODE=00b) 00: 模拟输入模式; 01: 浮空输入模式; 10: 带有上下拉模式; 11: 保留。在输出模式时 (MODE>00b) 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。D31~D0 (MODE[1:0],y∈[8,15]): x 端口 y 引脚配置位。这些位通过软件写入,用于配置 I/O 模式。00: 输入模式; 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。

4. 数据寄存器

1) GPIO端口输入数据寄存器 (GPIO_x_INDR) (x=A~D)

该寄存器用于获取GPIO端口相应引脚的输入电平。1: 代表高电平; 0: 代表低电平。

GPIO_x_INDR 寄存器的 D31~D16 位保留,必须保持复位值,即 0。D15~D0 (IDRy, y∈[0,15]), 端口输入数据位。这些位为只读位,它们包含相应 I/O 引脚的电平信息。

2) GPIO端口输出数据寄存器 (GPIO_x_OUTDR) (x=A~D)

该寄存器用于设置GPIO端口相应引脚的输出电平。1: 代表高电平; 0: 代表低电平。

GPIO_x_OUTDR 寄存器的 D31~D16 位保留,必须保持复位值,即 0。D15~D0 (ODR

y, y \in [0,15])：端口输出数据位。这些位可通过软件读取和写入，该位决定着被配置为输出引脚的电平的高低。若 ODR5=0，5 号引脚为低电平；ODR5=1，5 号引脚为高电平。

5. 其他寄存器

与 GPIO 编程相关的还有一些寄存器，限于篇幅，本书不再介绍，可参阅电子资源的“..\01-Information\CH32FV2x_V3x 系列应用手册”中的“第 10 章 GPIO 及其复用功能(GPIO/AFIO)”。

4.4.2 GPIO基本编程步骤并点亮一盏小灯

本节给出用直接对端口进行编程的方法点亮小灯。

1. GPIO基本编程步骤

要使芯片某一引脚为 GPIO 功能，并定义为输入/输出，随后进行应用，基本编程步骤如下：

(1) 通过外设时钟使能寄存器(RCC_APB2PCENR)设定对应 GPIO 端口外设时钟使能，本例设定 GPIO 的 C 口外设时钟使能。

(2) 通过 GPIO 模块的端口配置寄存器(GPIOx_CFGLR、GPIOx_CFGHR)设定其为 GPIO 功能，可设定为：输入模式、输出模式，本例设定为通用输出模式。

(3) 若是输出引脚，可通过数据输出寄存器(GPIOx_OUTER)设置 GPIO 端口相应引脚的输出状态；也可通过端口复位/置位寄存器(GPIOx_BSHR)设置 GPIO 端口相应引脚的输出状态；也可以通过端口复位寄存器(GPIOx_BCR)设置相应引脚的输出状态为低电平。本例通过 GPIOx_BCR 设置 PTC2 引脚输出低电平。

(4) 若是输入引脚，则通过数据输入寄存器(GPIOx_IDR)获得引脚的状态。若定位为 0，表示当前该引脚上为低电平；若为 1，则为高电平。

2. 用GPIO直接点亮一盏小灯

在开发套件的底板上，有红绿蓝三色灯（合为一体的），分别使用 MCU 的 PTC0、PTC1、PTC2 引脚。现使用 PTC2 引脚点亮蓝灯，步骤如下。

(1) 声明变量并赋值

```
// (1.5.1)声明变量
volatile uint32_t* RCC_AHB2;           // GPIO的C口时钟使能寄存器地址
volatile uint32_t* gpio_ptr;           // GPIO的C口基地址
volatile uint32_t* gpio_mode;          // 引脚模式寄存器地址=口基地址
volatile uint32_t* gpio_bsrr;          // 置位/复位寄存器地址
volatile uint32_t* gpio_brr;           // GPIO位复位寄存器

// (1.5.2) 变量赋值
RCC_AHB2=(uint32_t*)0x40021018;        // GPIO的C口时钟使能寄存器地址
gpio_ptr=(uint32_t*)0x40011000;        // GPIO的C口基地址
```

```

gpio_mode=gpio_ptr;           // 低位配置寄存器地址=口基地址
gpio_bsrr=(uint32_t*)0x40011010; // 置位/复位寄存器地址
gpio_brr=(uint32_t*)0x40011014; // GPIO位复位寄存器

```

(2) 随后对 GPIO 初始化

```

// (1.5.3.1) 使能相应GPIOC的时钟
*RCC_AHB2|= (1<<4);           // GPIO的C口时钟使能
// (1.5.3.2) 定义C口2脚为输出引脚 (令D11、D10=01)方法如下:
*gpio_mode &= ~(3<<8);        // 0b11111111111111111111111111111111;
*gpio_mode |= (1<<8);          // 0b000000000000000000000000100000000;

```

(3) 设置灯为亮

```

*gpio_brr|= (1<<2);           // 设置灯“亮”

```

特别说明：在嵌入式软件设计中，输入还是输出，是站在 MCU 角度，要控制红灯亮暗，就是输出。若要获取外部状态到 MCU 中，对 MCU 来说，就是输入。

这样这个蓝灯就亮起来了。这种编程方法的样例，在本书网上电子资源的“..\04-Software\CH04-GPIO\GPIO-Output-DirectAddress”工程中可以看到。

这样编程只是为了理解 GPIO 的基本编程方法，实际并不使用。不会这样从“零”直接应用程序，而是作为制作构件的第一步，把流程打通，作为封装构件的前导步骤。而制作 GPIO 构件，就是要把对 GPIO 底层硬件操作作用构件把它们封装起来，给出函数名与接口参数，供实际编程时使用。第五章将阐述底层驱动构件封装方法与基本规范。

4.4.3 GPIO构件的设计

1. 设计GPIO驱动构件的必要性

软件构件（Software component）技术的出现，为实现软件构件的工业化生产提供了理论与技术基石。将软件构件技术应用到嵌入式软件开发中，可以大大提高嵌入式开发的开发效率与稳定性。软件构件的封装性、可移植性与可复用性是软件构件的基本特性，采用构件技术设计软件，可以使软件具有更好的开放性、通用性和适应性。特别是对于底层硬件的驱动编程，只有封装成底层驱动构件，才能减少重复劳动，使广大 MCU 应用开发者专注于应用软件稳定性与功能设计上。因此，必须把底层硬件驱动设计好、封装好。

一个芯片有许多引脚可以作为 GPIO 引脚，分布在若干个端口，不可能使用直接地址去操作每个引脚相关寄存器，那样无法实现软件移植与复用。应该把对 GPIO 引脚的操作封装成构件，通过函数调用与传参的方式实现对引脚的干预与状态获取，这样的软件才便于维护与移植，因此设计 GPIO 驱动构件十分必要。同时，底层驱动构件的封装，也为在操作系统下对底层硬件的操作提供了基础。

2. 底层驱动构件封装基本要求

底层驱动构件封装规范见 5.3 节，本节给出概要，以便在认识第一个构件前以及在设计构件时，少走弯路，做出来的构件符合基本规范，便于移植、复用和交流。

1) 底层驱动构件的组成、存放位置与内容

每个构件由头文件(.h)与源文件(.c)文件两个独立文件组成,放在以构件名命名的文件夹中。驱动构件头文件(.h)中仅包含对外接口函数的声明,是构件的使用指南。以构件名命名。例如 GPIO 构件命名为 gpio (使用小写,目的是与内部函数名前缀统一)。基本要求是调用者只看头文件即可使用构件。对外接口函数及内部函数的实现在构件源程序文件(.c)中。同时应注意,头文件中声明对外接口函数的顺序与源程序文件实现对外接口函数的顺序应保持一致。源程序文件中内部函数的声明,放在外接口函数代码的前面,内部函数的实现放在全部外接口函数代码的后面,以便提高可阅读性与可维护性。一个具体的工程中,在本书给出的标准框架下,所有面向 MCU 芯片的底层驱动构件放在工程文件夹下的“..\03_MCU\MCU_drivers”文件夹中,本书所有规范样例工程下的文件组织均是如此。

2) 设计构件的最基本要求

这里摘要给出设计构件的最基本要求。一是使用与移植方便。要对构件的共性与个性进行分析,抽取出构件的属性和对外接口函数。希望做到:使用同一芯片的应用系统,构件不更改,直接使用;同系列芯片的同功能底层驱动移植时,仅改动头文件;不同系列芯片的同功能底层驱动移植时,头文件与源程序文件的改动尽可能少;二是要有统一、规范的编码风格与注释,主要涉及文件、函数、变量、宏及结构体类型的命名规范;涉及空格与空行、缩进、断行等的排版规范;涉及文件头、函数头、行及边等的注释规范,具体要求见 5.3.2 节。三是关于宏的使用限制。宏使用具有两面性,有提高可维护性一面,也有降低阅读性一面,不要随意使用宏。四是关于全局变量问题。构件封装时,应该禁止使用全局变量。

3. GPIO驱动构件封装要点分析

同样以 GPIO 驱动构件为例,进行封装要点分析。即分析应该设计哪几个函数及入口参数。GPIO 引脚可以被定义成输入、输出两种情况:若是输入,程序需要获得引脚的状态(逻辑 1 或 0);若是输出,程序可以设置引脚状态(逻辑 1 或 0)。MCU 的 PORT 模块分为许多端口,每个端口有若干引脚。GPIO 驱动构件可以实现对所有 GPIO 引脚统一编程。GPIO 驱动构件由 gpio.h、gpio.c 两个文件组成,如要使用 GPIO 驱动构件,只需要将这两个文件加入到所建工程中,由此方便了对 GPIO 的编程操作。

1) 模块初始化(gpio_init)

由于芯片引脚具有复用特性,应把引脚设置成 GPIO 功能;同时定义成输入或输出;若是输出,还要给出初始状态。所以 GPIO 模块初始化函数 gpio_init 的参数为哪个引脚、是输入还是输出、若是输出其状态是什么,函数不必有返回值。其中引脚可用一个 16 位数据描述,高 8 位表示端口号,低 8 位表示端口内的引脚号。这样 GPIO 模块初始化函数原型可以设计为:

```
void gpio_init(uint16_t port_pin, uint8_t dir, uint8_t state);
```

其中 `uint8_t` 是无符号 8 位整型的别名，`uint16_t` 是无符号 16 位整型的别名，本书后面不再特别说明。

2) 设置引脚状态 (`gpio_set`)

对于输出，通过函数设置引脚是高电平（逻辑 1）还是低电平（逻辑 0）。入口参数应该是哪个引脚，输出其状态是什么，函数不必有返回值。这样设置引脚状态的函数原型可以设计为：

```
void gpio_set(uint16_t port_pin, uint8_t state);
```

3) 获得引脚状态 (`gpio_get`)

对于输入，通过函数获得引脚的状态是高电平（逻辑 1）还是低电平（逻辑 0），入口参数应该是哪个引脚，函数需要返回值引脚状态。这样设置引脚状态的函数原型可以设计为：

```
uint8_t gpio_get(uint16_t port_pin);
```

4) 引脚状态反转 (`void gpio_reverse`)

类似的分析，可以设计引脚状态反转函数的原型为：

```
void gpio_reverse(uint16_t port_pin);
```

5) 引脚上下拉使能函数 (`void gpio_pull`)

若引脚被设置成输入，可以设定内部上下拉。引脚上下拉使能函数的原型为：

```
void gpio_pull(uint16_t port_pin, uint8_t pullselect);
```

这些函数基本满足了对 GPIO 操作的基本需求。还有中断使能与禁止^①、引脚驱动能力等函数，比较深的内容，可暂时略过，使用或深入学习时参考 GPIO 构件即可。要实现 GPIO 驱动构件的这几个函数，给出清晰的接口、良好的封装、简洁的说明与注释、规范的编程风格等，需要一些准备工作，下一小节给出构件封装基本规范与前期准备。

根据构件生产的基本要求设计的第一个构件—GPIO 驱动构件，由头文件 `gpio.h` 与源程序文件 `gpio.c` 两个文件组成，头文件是使用说明。MCU 的基础构件放在工程的“`..\03_MCU\MCU_drivers`”文件夹下。

在 4.2.1 节中介绍 GPIO 驱动构件时已对头文件做了较为详细的说明，此处不再赘述。

4. GPIO驱动构件源程序文件 (`gpio.c`)

GPIO 驱动构件的源程序文件“`..\03_MCU\MCU_drivers\gpio.c`”中实现的对外接口函数，主要是对相关寄存器进行配置，从而完成构件的基本功能。构件内部使用的函数也在构件源程序文件中定义，下面给出部分函数的源代码。

```
// =====  
// 文件名称: gpio.c
```

^① 关于使能 (Enable) 与禁止 (Disable) 中断，文献中有多种中文翻译，如使能、开启；除能、关闭等，本书统一使用使能中断与禁止中断术语。

```

// 功能概要: GPIO底层驱动构件源文件
// 版权所有: 苏州大学嵌入式系统与物联网研究所 (http://sumcu.suda.edu.cn)
// 版本更新: 20210520-20220219
// 芯片类型: CH32V307
// =====

#include "gpio.h"

// GPIO口基地址放入常数数组GPIO_ARR[0]~GPIO_ARR[5]中
GPIO_TypeDef * GPIO_ARR[] =
    {(GPIO_TypeDef *)GPIOA_BASE,(GPIO_TypeDef *)GPIOB_BASE,
     (GPIO_TypeDef *)GPIOC_BASE,(GPIO_TypeDef *)GPIOD_BASE,
     (GPIO_TypeDef *)GPIOE_BASE};
// ====定义扩展中断IRQ号对应表====
IRQn_Type table_irq_exti[7] = {EXTI0_IRQn, EXTI1_IRQn, EXTI2_IRQn,
    EXTI3_IRQn, EXTI4_IRQn, EXTI9_5_IRQn, EXTI15_10_IRQn};
// 内部函数声明
void gpio_get_port_pin(uint16_t port_pin,uint8_t* port,uint8_t* pin);
void GPIO_EXTILineConfig(uint8_t port, uint8_t pin);
// =====
// 函数名称: gpio_init
// 函数返回: 无
// 参数说明: port_pin: (端口号)(引脚号) (如: (PTB_NUM)(9) 表示为B口9号脚)
//          dir: 引脚方向 (0=输入, 1=输出,可用引脚方向宏定义)
//          state: 端口引脚初始状态 (0=低电平, 1=高电平)
// 功能概要: 初始化指定端口引脚作为GPIO引脚功能, 并定义为输入或输出, 若是输出,
//          还指定初始状态是低电平或高电平
// =====
void gpio_init(uint16_t port_pin, uint8_t dir, uint8_t state)
{
    GPIO_TypeDef * gpio_ptr;    // 声明gpio_ptr为GPIO结构体类型指针
    uint8_t port,pin;           // 声明端口port、引脚pin变量
    uint32_t temp;              // 临时存放寄存器里的值
    // 根据带入参数port_pin, 解析出端口与引脚分别赋给port,pin
    gpio_get_port_pin(port_pin,&port,&pin);
    // 根据入口参数port, 给局部变量gpio_ptr赋值 (GPIO基地址)
    gpio_ptr = GPIO_ARR[port];
    // 使能相应GPIO时钟

```

```

RCC->APB2PCENR |= (RCC_IOPAEN << (port * 1u));
if(pin <= 0x07)
{
    // 清GPIO模式寄存器对应引脚位
    temp = gpio_ptr->CFGLR;
    temp &= ~(GPIO_CFGLR_CNF0 << (pin * 4u));
    // 根据入口参数dir, 定义引脚为输出或输入
    if(dir == 1)    // 定义为输出引脚
    {
        temp |= (GPIO_OUTPUT << (pin * 4u));
        gpio_ptr->CFGLR = temp;
        gpio_set(port_pin,state);    // 调用gpio_set函数, 设定引脚初始状态
    }
    else            // 定义为输入引脚
    {
        temp |= (GPIO_INPUT << (pin * 4u));
        gpio_ptr->CFGLR = temp;
    }
}
if(pin > 0x07)
{
    //清GPIO模式寄存器对应引脚位
    temp = gpio_ptr->CFGHR;
    temp &= ~(GPIO_CFGHR_CNF8 << (port * 4u));
    if(dir == 1)    // 定义为输出引脚 *
    {
        temp |= (GPIO_OUTPUT << (pin * 4u));
        gpio_ptr->CFGHR = temp;
        gpio_set(port_pin,state);
    }
    else            // 定义为输入引脚
    {
        temp |= (GPIO_INPUT << (pin * 4u));
        gpio_ptr->CFGHR = temp;
    }
}
}
}

```

（限于篇幅，省略其他函数实现，见电子资源）

下面对源码中的结构体类型、有关地址、编码的书写问题做简要说明。

1) 结构体类型

在工程文件夹的芯片头文件（“..\03_MCU\startup\ch32v10x.h”）中，有端口寄存器结构体，把端口模块的编程寄存器用结构体类型（GPIO_TypeDef）封装起来。

```
typedef struct
{
    __IO uint32_t  CFGLR;
    __IO uint32_t  CFGHR;
    __IO uint32_t  INDR;
    __IO uint32_t  OUTDR;
    __IO uint32_t  BSHR;
    __IO uint32_t  BCR;
    __IO uint32_t  LCKR;
} GPIO_TypeDef;
```

2) 端口模块及GPIO模块各口基地址

CH32V307VCT6 的 GPIO 模块各口基地址也在芯片头文件（ch32v30x.h）中以宏常数方式给出，本程序直接作为指针常量。

3) 编程与注释风格

读者需要仔细分析本构件的编程与注释风格，从开始就规范起来，这样就会逐步养成好的编程习惯。特别注意，不要编写令人难以看懂的程序，不要把简单问题复杂化，不要使用不必要的宏。

4.5 第一个汇编语言工程：控制小灯闪烁

汇编语言编程给人的第一种感觉就是难，相对于 C 语言编程，汇编在编程的直观性、编程效率、以及可读性等方面都有所欠缺，但掌握基本的汇编语言编程方法是嵌入式学习的基本功，可以增加嵌入式编程者的“内力”。

在本书教学资料中提供的开发环境中，汇编程序是通过工程的方式组织起来的。汇编工程通常包含芯片相关的程序框架文件、软件构件文件、工程设置文件、主程序文件及抽象构件文件等。下面将结合第一个汇编工程实例，讲解上述的文件概念，并简要分析汇编工程的组成、汇编程序文件的编写规范、软硬件模块的合理划分等。读者若能认真分析与实践第一个汇编实例程序，可以达到由此入门的目的。

4.5.1 汇编工程文件的组织

汇编工程的样例在“..\ CH04-GPIO\GPIO-asm”文件夹中。本汇编工程类似 C 工程，仍然按构件方式进行组织。图 4-4 给出了小灯闪烁汇编工程的树型结构，主要包括 MCU 相关头文件夹、底层驱动构件文件夹、Debug 工程输出文件夹、程序文件夹等。读者按照理解 C 工程的方式，理解这个结构。

GPIO-asm-20201110	工程名
.cproject	
.mxproject	
.project	
.settings	
01_Doc	文档文件夹
02_CPU	CPU相关文件夹
03_MCU	MCU相关文件夹
mcu.h	MCU基本信息头文件
Linker_file	链接文件夹，存放链接文件
MCU_drivers	MCU底层构件文件夹，存放芯片级硬件驱动
startup	MCU启动文件夹
04_GEC	芯片底层驱动构件文件夹
05_UserBoard	用户板构件文件夹
06_SoftComponent	软件构件文件夹
07_NosPrg	无操作系统工程主程序文件夹
include.inc	总头文件
isr.s	中断处理程序
main.s	主函数
Debug	工程输出文件夹（编译链接自动生成）

图4-4 小灯闪烁汇编工程的树型结构

汇编工程仅包含一个汇编主程序文件，该文件名固定为 main.s。汇编程序的主体是程序的主干，要尽可能简洁、清晰、明了，程序中的其余功能，尽量由子程序去完成，主程序主要完成对子程序的循环调用。主程序文件 main.s，包含有以下内容：

- （1）**工程描述。**工程名、程序描述、版本、日期等。
- （2）**包含总头文件。**声明全局变量和包含主程序文件中需要的头文件、宏定义等。
- （3）**主程序。**主程序一般包括初始化与主循环两大部分。初始化包括堆栈初始化、系统初始化、I/O 端口初始化、中断初始化等。主循环是程序的工作循环，根据实际需要安排程序段，但一般不宜过长，建议不要超过 100 行，具体功能可通过调用子程序来实现，或由中断程序实现。
- （4）**内部直接调用子程序。**若有不单独存盘的子程序，建议放在此处。这样在主程序总循环的最后一个语句就可以看到这些子程序。每个子程序不要超过 100 行。若有更多的子程序请单独存盘，单独测试。

4.5.2 汇编语言小灯测试工程主程序

1. 小灯测试工程主程序

该工程使用汇编语言来点亮蓝灯，main.s 的代码如下。

```
/* =====  
// 文件名称: main.s  
// 功能概要: 汇编编程调用GPIO构件控制小灯闪烁（利用printf输出提示信息）  
// 版权所有: 苏州大学嵌入式系统与物联网研究所（http://sumcu.suda.edu.cn）  
// 版本更新: 20210916-20220108  
// ===== */  
.include "include.inc" /* 头文件中主要定义了程序中需要使用到的一些常量 */  
/* (0) 数据段与代码段的定义 */  
/* (0.1) 定义数据存储data段开始，实际数据存储RAM中 */  
.section .data  
/* (0.1.1) 定义需要输出的字符串，标号即为字符串首地址，\0为字符串结束标志 */  
hello_information: /* 字符串标号 */  
    .string "-----\n"  
    .string "金葫芦提示: \n"  
    .string "LIGHT:ON--第一次用纯汇编点亮的蓝色发光二极管，太棒了! \n"  
    .string "    这只是万里长征第一步，但是，万事开头难， \n"  
    .string "    有了第一步，坚持下去，定有收获! \n"  
    .string "-----\n\0"  
data_format:  
    .ascii "%d\n\0" /* printf使用的数据格式控制符 */  
light_show1:  
    .ascii "LIGHT_BLUE:ON--\n\0" /* 灯亮状态提示 */  
light_show2:  
    .ascii "LIGHT_BLUE:OFF--\n\0" /* 灯暗状态提示 */  
light_show3:  
    .ascii "闪烁次数mLightCount=\0" /* 闪烁次数提示 */  
/* (0.1.2) 定义变量  
.align 4 /* .word格式四字节对齐 */  
mMainLoopCount: /* 定义主循环次数变量 */  
    .word 0  
mFlag: /* 定义灯的状态标志，1为亮，0为暗 */  
    .byte 'A'  
.align 4
```

```

mLightCount:
    .word 0

/* (0.2) 定义代码存储text段开始，实际代码存储在Flash中 */
.section    .text
.type main function    /* 声明main为函数类型 */
.global main    /* 将main定义成全局函数，便于芯片初始化之后调用 */
.align 2    /* 指令和数据采用2字节对齐 */

/* ----- */
/* main.c使用的内部函数声明处 */
/* ----- */

/* 主函数，一般情况下可以认为程序从此开始运行（实际上有启动过程） */
main:
/* (1) =====启动部分（开头）主循环前的初始化工作===== */
/* (1.1) 声明main函数使用的局部变量 */

/* (1.2) 【不变】关总中断 */

/* (1.3) 给主函数使用的局部变量赋初值 */

/* (1.4) 给全局变量赋初值
    ADDI    sp, sp, -16    /* 分配栈帧 */
    SW     ra, 12(sp)    /* 存储放回地址 */
/* (1.5) 用户外设模块初始化 */
/* 初始化蓝灯， a0、a1、a2是gpio_init的入口参数 */
    LI     a0, LIGHT_RED    /* a0指明端口和引脚 */
    LI     a1, GPIO_OUTPUT    /* a1指明引脚方向为输出 */
    LI     a2, LIGHT_OFF    /* a2指明引脚的初始状态为亮 */
    CALL    gpio_init    /* 调用gpio初始化函数 */
/* 初始化串口UART_User
    LI     a0, UART_User    /* 串口号 */
    LI     a1, UART_BAUD    /* 波特率 */

    CALL    uart_init    /* 调用uart初始化函数 */
    CALL    Delay_Init    /* 初始化延时函数 */
/* (1.6) 使能模块中断 */
    LI     a0, UART_User    /* 串口号 */

```

```

CALL    uart_enable_re_int /* 调用uart中断使能函数 */

/* (1.7) 【不变】开总中断 */

/* 显示hello_information定义的字符串 */
LA      a0, hello_information /* 待显示字符串首地址 */
CALL    printf                /* 调用printf显示字符串 */

/* CALL    . /* 在此打桩(.表示当前地址), 理解发光二极管为何亮起来了? */

/* (1) =====启动部分（结尾）===== */
        LA a5, mMainLoopCount
/* (2) =====主循环部分（开头）===== */
main_loop:                /* 主循环标签（开头） */
/* (2.1) 主循环次数变量mMainLoopCount+1 */
        ADDI a5, a5, 1
/* (2.2) 未达到主循环次数设定值, 继续循环 */
        LI a2, MainLoopNUM
        BLTU a5, a3, main_loop /* 未达到, 继续循环 */
/* (2.3) 达到主循环次数设定值, 执行下列语句, 进行灯的亮暗处理 */
/* (2.3.1) 清除循环次数变量 */
        LA a2, mMainLoopCount /* a2←mMainLoopCount的地址 */
        LI a1, 0
        SW a1, 0(a2)
/* (2.3.2) 如灯状态标志mFlag为'L', 灯的闪烁次数+1并显示, 改变灯状态及标志 */
/* 判断灯的状态标志 */
        LA a2, mFlag
        LH t6, 0(a2)
        LI t5, 'L'
        BNE t6, t5, main_light_off /* mFlag不等于'L'转 */
/* mFlag等于'L'情况 */
        LA a3, mLightCount /* 灯的闪烁次数mLightCount+1 */
        LH a1, 0(a3)
        ADDI a1, a1, 1
        SW a1, 0(a3)
        LA a0, light_show3 /* 显示“灯的闪烁次数mLightCount=” */
        CALL printf
        LA a0, data_format /* 显示灯的闪烁次数值 */

```

```

    LA  a2, mLightCount
    LH  a1, 0(a2)
    CALL printf
    LA  a2, mFlag          /* 灯的状态标志改为'A' */
    LI  t4, #'A'
    SW  t4, 0(a2)
    LI  a0, LIGHT_RED      /* 亮灯 */
    LI  a1, LIGHT_ON
    CALL gpio_set
    LA  a0, light_show1    /* 显示灯亮提示 */
    CALL printf
    /* mFlag等于'L'情况处理完毕，转 */
    J   main_exit
/* (2.3.3) 如灯状态标志mFlag为'A'，改变灯状态及标志 */
main_light_off:
    LA  a2, mFlag          /* 灯的状态标志改为'L' */
    LI  t4, 'L'
    SW  t4, 0(a2)
    LI  a0, LIGHT_RED      /* 暗灯 */
    LI  a1, =LIGHT_OFF
    CALL gpio_set
    LA  a0, light_show2    /* 显示灯暗提示 */
    CALL printf
main_exit:
    LI  a5, 0
    J   main_loop          /* 继续循环 */
/* (2) =====主循环部分（结尾）===== */
/* 释放栈 */
    LW  ra, 12(sp)         /* 恢复返回地址 */
    ADDI sp, sp, 16        /* 释放栈帧 */
    LI  a0, 0              /* 读取返回值0 */
    RET                    /* 返回 */
.end                      /* 整个程序结束标志（结尾） */

```

2. 汇编工程运行过程

当芯片内电复位或热复位后，系统程序的运行过程可分为两部分：**main** 函数之前的运行和 **main** 函数之后的运行。

mian 函数之前的运行过程可以参考 4.3.2 小节加以体会和理解，下面对 **main** 函数之

后的运行过程简要分析。

(1) 进入 `main` 函数后先对所用到的模块进行初始化，比如小灯端口引脚的初始化，小灯引脚复用设置为 `GPIO` 功能，设置引脚方向为输出，设置输出为高电平，这样蓝色小灯就可以被点亮。

(2) 当某个中断发生后，MCU 将转到中断向量表文件 `isr.s` 所指定的中断入口地址处开始运行中断服务例程（ISR，Interrupt Service Routine），因为该小灯程序没有中断向量表文件所以此处就不再描述汇编中断程序，深入学习的读者，不难完成此任务。

4.6 实验一 熟悉实验开发环境及GPIO编程

结构合理、条理清晰的程序结构，有助于提高程序的可移植性与可复用性，有利于程序的维护。学习嵌入式软件编程，从一开始就养成规范编程的习惯，将为未来发展打下坚实基础。这是第一个实验，目的是以通用输入输出为例，达到熟悉实验开发环境、理解规范编程结构、掌握基本调试方法等目的。

1. 实验目的

本实验通过编程控制 LED 小灯，体会 `GPIO` 输出作用，可扩展控制蜂鸣器、继电器等；通过编程获取引脚状态，体会 `GPIO` 输入作用，可用于获取开关的状态，主要目的如下：

- (1) 了解集成开发环境的安装与基本使用方法。
- (2) 掌握 `GPIO` 构件基本应用方法，理解第一个 C 程序框架结构，了解汇编语言与 C 语言的如何相互调用。
- (3) 掌握硬件系统的软件测试方法，初步理解 `printf` 输出调试的基本方法。

2. 实验准备

- (1) 硬件部分。PC 机或笔记本电脑一台、本书随附的 AHL-CH32V307 开发套件一套。
- (2) 软件部分。从苏州大学嵌入式学习社区网站，按照本书 1.1.2 节，下载合适的电子资源。
- (3) 软件环境。按照本书 1.1.2 节，进行有关软件工具的安装。

3. 参考样例

(1) “`..\04-Software\GPIO\GPIO-Output-DirectAddress`”。该程序使用直接地址编程方式，点亮一个发光二极管。从中可了解到，模块的哪个寄存器的哪一位变化使得发光二极管亮了，由此理解硬件是如何干预软件的。但这个程序不作为标准应用编程模板，因为要真正进行规范的嵌入式软件编程，必须封装底层驱动构件，在此基础上进行嵌入式软件开发。

(2) “`..\04-Software\GPIO\GPIO-Output-Component`”。该程序通过调用 `GPIO` 驱动构件方式，使得一个发光二极管闪烁。使用构件方式编程干预硬件是今后编程的基本方式。

而使用直接地址编程方式干预硬件，仅用于底层驱动构件制作过程中的第一阶段（打通硬件），为构件封装做准备。

4. 实验过程或要求

1) 验证性实验

(1) 下载开发环境。

(2) 建立自己的工作文件夹。按照“分门别类，各有归处”之原则，建立自己的工作文件夹。并考虑随后内容安排，建立其下级子文件夹。

(3) 拷贝模板工程并重命名。所有工程可通过拷贝模板工程建立。例如，“..\04-Software\GPIO\GPIO-Output-DirectAddress”工程到自己的工作文件夹，可以改为自己确定的工程名，建议尾端增加“-220109”字样，表示日期，避免混乱。

(4) 导入工程。打开集成开发环境 AHL-GEC-IDE。接着单击“文件”→“导入工程”→导入你拷贝到自己文件夹并重新命名的工程。导入工程后，左侧为工程树形目录，右边为文件内容编辑区，初始显示 main.c 文件的内容。此时，与图 4-5 基本一致。

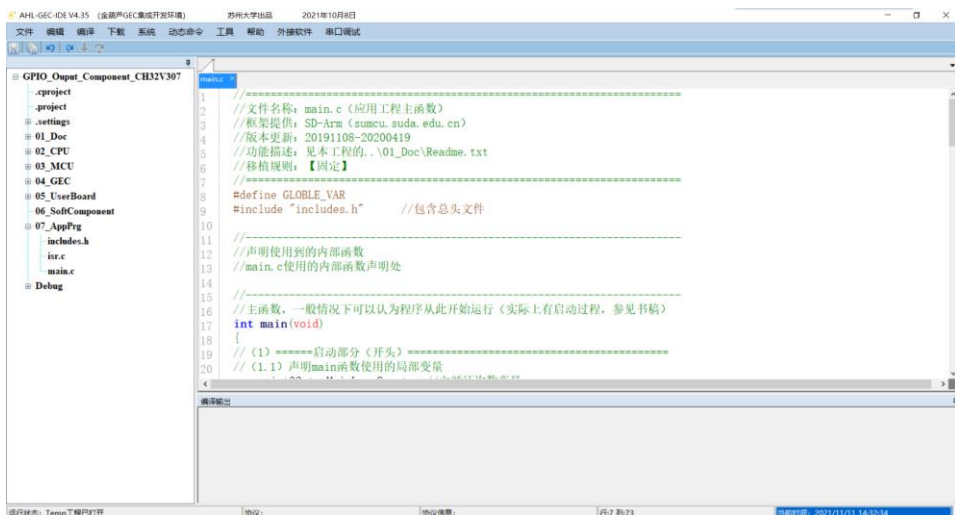


图4-5 AHL-GEC-IDE界面

(5) 编译工程。在打开工程，并显示文件内容前提下，可编译工程。单击“编译”→“编译工程”，则开始编译。

(6) 下载并运行。步骤一，硬件连接：用 Type-C 线连接 GEC 底板上的 Type-C 口与电脑的 USB 口；步骤二，软件连接。单击“下载”→“串口更新”，将进入更新窗体界面。点击“连接 GEC”查找到目标 GEC，则提示“串口号+BIOS 版本号”；步骤三，下载机器码。点击“选择文件”按钮导入被编译工程目录下 Debug 中的.hex 文件，例如：GPIO-Output-DirectAddress_CH32V307.hex 文件，然后单击“一键自动更新”按钮，等待程序自动更新完成。

(7) 观察运行结果与程序的对应。第一个程序运行结果（PC 机界面显示情况）如图 4-6 所示。



图4-6 第一个程序运行结果（PC机界面显示情况）

（8）继续验证其他样例。对于“..\04-Software\CH04”文件夹下提供的每个样例，均进行体验、理解执行过程（从 main 函数为启动理解即可）。特别是，可以使用“for(;;){ }”打个“桩”，这里“桩”特指运行到这里“看结果”，“桩”前面可以放“printf”语句，充分利用本开发环境的下载后立即运行及 printf 函数同步显示功能，进行基本语句功能测试。测试正确之后，删除 printf 语句及“桩”，继续后续编程。相对于更复杂的调试方法，这个方法十分简便。初学时，每编写几个语句，就可利用这种方法进行测试。不要编写过多语句再测试，有时找错花太多时间。

2）设计性实验

自行编程实现开发板上的红、蓝、绿及组合颜色交替闪烁。LED 三色灯电路原理图如图 3-4 所示，对应三个控制端接 MCU 的三个 GPIO 引脚。可以通过程序，测试你使用的开发套件中的发光二极管是否与图中接法一致。

3）进阶实验★

- （1）用直接地址编程方式，实现设计性实验。
- （2）用汇编语言编程方式，实现设计性实验。

5. 实验报告要求

- （1）基本掌握 WORD 文档的排版方法。
- （2）用适当文字、图表描述实验过程。
- （3）用 200~300 字写出实验体会。
- （4）在实验报告中完成实践性问答题。

6. 实践性问答题

（1） $X \&= \sim(1 \ll 3)$ 的目的是什么？ $X |= (1 \ll 3)$ 的目的是什么？给出详细演算过程，举例说明其用途。

-
- (2) volatile 作用是什么？举例说明其使用地方？
 - (3) 给出一个全局变量的地址。
 - (4) 集成的红绿蓝三色灯最多可以实现几种不同颜色 LED 灯的显示，通过实验给出组合列表。
 - (5) 给出获得一个开关量状态的基本编程步骤。

本章小结

本章作为全书的重点和难点之一，给出了 MCU 的 C 语言工程编程框架，对第一个 C 语言入门工程进行了较为详尽的阐述。透彻理解工程的组织原则、组织方式及运行过程，对后续的学习将有很大的铺垫作用。

1. 关于GPIO的基本概念

GPIO 是输入/输出的最基本形式，MCU 的引脚若作为 GPIO 输入引脚，即开关量输入，其含义就是 MCU 内部程序可以获取该引脚的状态，是高电平 1，或是低电平 0。若作为输出引脚，即开关量输出，其含义就是 MCU 内部程序可以控制该引脚的状态，是高电平 1，或是低电平 0。希望掌握开关量输入/输出电路的基本连接方法。

2. 关于基于构件的程序框架

本章通过点亮一盏小灯的过程来开启嵌入式学习之旅，基于从简单到复杂的学习思路，4.2 节给出了一个基于构件点亮小灯的工程样例，并以此为基础讲述程序框架组织以及各文件的功能。嵌入式系统工程往往包含许多文件，有程序文件、头文件、与编译调试相关的文件、工程说明文件、开发环境生成文件等，合理组织这些文件规范工程组织可以提高项目的开发效率和可维护性，工程组织应体现嵌入式软件工程的基本原则与基本思想。本书提供的工程框架主要包括了 01_Doc、02_CPU、03_MCU、04_GEC、05_UserBoard、06_SoftComponent、07_AppPrg 共 7 个文件夹，每个文件夹下存放不同功能的文件，通过文件夹的名称可直接体现出来，用户今后在使用时无需新建工程，复制后改名即为新工程。主程序文件 main.c 是应用程序的启动后总入口，main 函数即在该文件中实现。应用程序的执行，一共两条独立的线路，一条是 main 函数中的永久循环线，另一条是中断线，在 isr.c 文件中编程，将在第 6 章中阐述。若有操作系统，则在这里启动操作系统的调度器。

3. 关于构件的设计过程

为了一开始就进行规范编程。4.4 节给出了 GPIO 驱动构件封装方法与驱动构件封装规范简要说明。在实际工程应用中，为了提高程序的可移植性，不能在所有的程序中都直接操作对应的寄存器，需要将对底层的操作封装成构件，对外提供接口函数，上层只需在调用时传进对应的参数即可完成相应功能，具体封装时用.c 文件保存构件的实现代码，用.h 文件保存需对外提供的完整函数信息及必要的说明。4.4 节中给出了 GPIO 构件的设计方法，在 GPIO 构件中设计了引脚初始化（gpio_init）、设定引脚状态（gpio_set）、获取引脚状态（gpio_get）等基本函数，使用这些接口函数可基本完成对 GPIO 引脚的操作。

4. 关于汇编工程样例

本章 4.5 节给出了一个规范的汇编工程样例，供汇编入门使用，读者可以实际调试理解该样例工程，达到初步理解汇编语言编程之目的。对于嵌入式初学者来说，理解一个汇编语言程序是十分必要的。

习 题

1. 举例给出使用对直接映像地址赋值的方法，实现对一盏小灯编程控制的程序语句。
2. 在第一个样例程序的工程组织图中，哪些文件是由用户编写的？哪些是由开发环境编译链接产生的？
3. 简述第一个样例程序的运行过程。
4. 给出链接文件的功能要点。
5. 说明全局变量在哪个文件声明，在哪个文件中给全局变量中赋初值，举例说明一个全局变量的存放地址。
6. 自行完成一个汇编工程，功能、难易程度自定。
7. 从寄存器角度对 GPIO 编程，GPIO 的输出有推挽输出与开漏输出类型，说明其应用场合。基础的 GPIO 构件中，默认是什么输出类型。
8. 从寄存器角度对 GPIO 编程，GPIO 的输出有输出速度问题，为什么封装基础构件时，不把输出速度作为形式参数？

第6章 串行通信模块及第一个中断程序结构

本章导读：本章阐述 CH32V307 的串行通信构件化编程。主要内容有两个模块：异步串行通信模块及中断模块。首先是异步串行通信模块，给出了异步串行通信的通用基础知识，使读者理解串行通信的基本概念及编程模型；阐述了基于构件的串行通信编程方法，这是一般应用级编程的基本模式，还给出了 UART 构件的基本制作过程。其次是中断模块，给出了青稞 V4F 快速中断机制及 CH32V307 中断编程步骤，阐述了嵌入式系统的中断处理基本方法。最后给出了串口通信及中断实验，读者通过实验熟悉 MCU 的异步串行通信 UART 的工作原理，掌握 UART 的通信编程方法、串口组帧编程方法以及 PC 机的 C#串口通信编程方法。

6.1 异步串行通信的通用基础知识

串行通信接口，简称“串口”、UART 或 SCI。在 USB 未普及之前，串口是 PC 机必备的通信接口之一。作为设备间简便的通信方式，在相当长的时间内，串口还不会消失，在市场上也可很容易的购买到各种电平到 USB 的串口转接器，以便与没有串口但具有多个 USB 口的笔记本电脑或 PC 机连接。MCU 中的串口通信，在硬件上，一般只需要三根线，分别称为发送线（TxD）、接收线（RxD）和地线（GND）；通信方式上，属于单字节通信，是嵌入式开发中重要的打桩调试手段。实现串口功能的模块在一部分 MCU 中被称为通用异步收发器（Universal Asynchronous Receiver-Transmitters, UART），在另一些 MCU 中被称为串行通信接口（Serial Communication Interface, SCI）。

本节简要概述 UART 的基本概念与硬件连接方法，为学习 MCU 的 UART 编程做准备。

6.1.1 串行通信的基本概念

“位”（bit）是单个二进制数字的简称，是可以拥有两种状态的最小二进制值，分别用“0”和“1”表示。在计算机中，通常一个信息单位用 8 位二进制表示，称为一个“字节”（byte）。串行通信的特点是：数据以字节为单位，按位的顺序（例如最高位优先）从一条传输线上发送出去。这里至少涉及 4 个问题：**第一，每个字节之间是如何区分开的？第二，发送一位的持续时间是多少？第三，怎样知道传输是正确的？第四，可以传输多远？**这些问题所需要的知识点涉及到串行通信的基本概念。串行通信分为异步通信与同步通信两种方式，本节主要给出异步串行通信的一些常用概念。正确理解这些概念，对串行通信编程是有益的。主要掌握异步串行通信的格式与波特率，至于奇偶校验与串行通信的传输方式术语了解即可。

1. 异步串行通信的格式

在 MCU 的英文芯片手册上，通常说的异步串行通信的格式是标准不归零传号/空号

数据格式（standard non-return-zero mark/space data forma），该格式采用不归零码(NRZ, Non-Return to Zero)格式。“不归零”的最初含义是：这里采用双极性表示二进制值，如用负电平表示一种二进制值，正电平表示另一种二进制值。在表示一个二进制值码元时，电压均无需回到零，故称不归零码。“mark/space”即“传号/空号”分别是表示两种状态的物理名称，逻辑名称记为“1/0”。对学习嵌入式应用的读者而言，只要理解这种格式只有“1”、“0”两种逻辑值就可以了。UART 串口通信的数据包以帧为单位，常用的帧结构为：1 位起始位+8 位数据位+1 位奇偶校验位（可选）+1 位停止位。图 6-1 给出了 8 位数据、无校验情况的传送格式。



图 6-1 串行通信数据格式

这种格式的空闲状态为“1”，发送器通过发送一个“0”表示一个字节传输的开始，随后是数据位（在 MCU 中一般是 8 位或 9 位，可以包含校验位）。最后，发送器发送位 1 或 2 位的停止位，表示一个字节传送结束。若继续发送下一字节，则重新发送开始位（这就是异步之含义了），开始一个新的字节传送。若不发送新的字节，则维持“1”的状态，使发送数据线处于空闲。从开始位到停止位结束的时间间隔称为一字节帧（Byte Frame）。所以，也称这种格式为字节帧格式。每发送一个字节，都要发送“开始位”与“停止位”，这是影响异步串行通信传送速度的因素之一。

【思考一下】 UART 中每个字节之间是如何区分开的？

2. 串行通信的波特率

位长（Bit Length），也称为位的持续时间（Bit Duration），其倒数就是单位时间内传送的位数。串口通信的速度用波特率来表示，它定义为每秒传输的二进制位数，在这里 1 波特=1 位/s，单位 bps（位/s）。bps 是英文 bit per second 的缩写，习惯上这个缩写不用大写，而用小写。通常情况下，波特率的单位可以省略。只有通信双方的波特率一样时才可以进行正常通讯。

通常使用的波特率有 9600、19200、38400、57600 及 115200 等。如果采用 10 位表示一个字节，包含开始位、数据位以及停止位，很容易计算出，在各波特率下，发送 1KB 所需的时间。显然，这个速度相对于目前许多通信方式而言是慢的，那么，异步串行通信的速度能否提得很高呢？答案是不能。因为随着波特率的提高，位长变小，以至于很容易受到电磁源的干扰，通信就不可靠了。当然，还有通信距离问题，距离小，可以适当提高波特率，但这样毕竟提高的幅度非常有限，达不到大幅度提高的目的。

3. 奇偶校验

在异步串行通信中，如何知道一个字节的传输是否正确？最常见的方法是增加一个位（奇偶校验位），供错误检测使用。由于属于单字节校验，意义不大，实际编程使用较少，因此，奇偶校验的基本含义在电子资源的补充阅读材料中给出。

4. 串行通信传输方式术语

在串行通信中，经常用到全双工、半双工、单工等术语，它们是串行通信的不同传输方式。下面简要介绍这些术语的基本含义。

(1) 全双工 (Full-duplex)：数据传送是双向的，且可以同时接收与发送数据。这种传输方式中，除了地线之外，需要两根数据线，站在任何一端的角度看，一根为发送线，另一根为接收线。一般情况下，MCU 的异步串行通信接口均是全双工的。

(2) 半双工 (Half-duplex)：数据传送也是双向的，但是在这种传输方式中，除地线之外，一般只有一根数据线。任何时刻，只能由一方发送数据，另一方接收数据，不能同时收发。

(3) 单工 (Simplex)：数据传送是单向的，一端为发送端，另一端为接收端。这种传输方式中，除了地线之外，只要一根数据线就可以了。有线广播就是单工的。

6.1.2 RS232和RS485总线标准

现在回答“可以传输多远”这个问题。MCU 引脚输入/输出一般使用晶体管-晶体管逻辑 (Transistor Transistor Logic, TTL) 电平。而 TTL 电平的“1”和“0”的特征电压分别为 2.4V 和 0.4V (目前使用 3V 供电的 MCU 中，该特征值有所变动)，即大于 2.4V 则识别为“1”，小于 0.4V 则识别为“0”。它适用于板内数据传输。若用 TTL 电平将数据传输到 5 米之外，那么可靠性就很值得考究了。为使信号传输得更远，美国电子工业协会 EIA (Electronic Industry Association) 制订了串行物理接口标准 RS232，后来又演化出 RS485。

1. RS232

RS232 采用负逻辑，-15V~-3V 为逻辑“1”，+3V~+15V 为逻辑“0”。RS232 最大的传输距离是 30m，通信速率一般低于 20kbps。当然，在实际应用中，也有人用降低通信速率的方法，通过 RS232 电平，将数据传送到 300m 之外，这是很少见的，且稳定性很不好。目前主要用于几米到几十米范围内的近距离通信。有专门的书籍介绍这 RS232 总线标准最初是为远程数据通信制订的，但个标准，但对于一般的读者，不需要掌握 RS232 标准的全部内容，只要了解本节介绍基本知识就可以使用 RS232。

早期的标准串行通信接口是 25 芯，后来改为 9 芯，目前部分 PC 机带有 9 芯 RS232 串口，其引出脚排列如图 6-2 所示，相应引脚含义若表 6-1 所示。

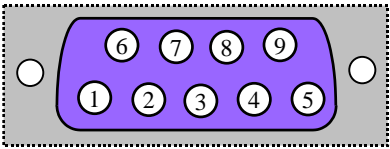


图6-2 9芯串行接口排列

表6-1 计算机中常用的9芯串行接口引脚含义表

引脚号	功 能	引脚号	功 能
1	接收线信号检测	6	数据通信设备准备就绪 (DSR)
2	接收数据线 (RxD)	7	请求发送 (RTS)
3	发送数据线 (TxD)	8	允许发送 (CTS)
4	数据终端准备就绪 (DTR)	9	振铃指示
5	信号地 (SG, 与 GND 一致)		

MCU 的串口通信引脚是 TTL 电平，可通过 TTL-RS232 转换芯片转为 RS232 电平。通常情况，使用精简的 RS232 通信线路，即仅使用 3 根线：RxD（接收线）、TxD（发送线）和 GND（地线），不使用诸如 DTR、DSR、RTS、CTS 等硬件握手信号，直接通过数据线的开始位确定一个字节通信的开始。

2. RS485

此外，为了组网方便，还有一种标准，称为 RS485，它采用差分信号负逻辑，-2V~-6V 表示“1”，+2V~+6V 表示“0”。硬件连接上，采用两线制接线方式，工业应用较多。所谓差分，就是两线电平相减，得出一个电平信号，可以较好地抑制电磁干扰。RS485 标准是为了弥补 RS232 通信距离短、速率低等缺点而产生的，通信距离在 1000 米左右。由于使用差分信号传输，二线的 RS485 通信只能工作于半双工方式，若要全双工通信，必须使用四线。在 MCU 的外围电路中，串口通信要使用 RS485 方式传输，需要使用 TTL-RS485 转换芯片。需要说明的是，上面介绍的 TTL-RS232 转换芯片，以及这里介绍的 TTL-RS485 转换芯片，还有下面将介绍的 TTL-USB 转换芯片，均是硬件电平信号之间的转换，与 MCU 编程无关，MCU 的串口编程是一致的。

【思考一下】为什么差分传输可以较好地抑制电磁干扰？

6.1.3 TTL-USB 串口

由于 USB 接口已经在笔记本电脑及 PC 机标准配置中普及，但是笔记本电脑及 PC 机作为 MCU 程序开发的工具机，需要与 MCU 进行串行通信。于是出现了 TTL-USB 串口芯片，这里介绍南京沁恒微电子股份有限公司生产的一款双路串口转 USB 芯片 CH342。

1. CH342 简介

CH342 是南京沁恒微电子有限公司推出的一款 TTL-USB 串口转接芯片，能够实现两个异步串口与 USB 信号的转换。CH342 芯片有 3 个电源端，内置了产生 3.3V 的电源调节器，工作电压在 1.8V~5V 之间；含有内置时钟电路，支持的通讯波特率在 50bps~3Mbps，工作温度在 -40~+85℃。

2. CH342 与 MCU 芯片引脚的连接电路

CH342 芯片在引脚结构上包括：数据传输引脚、MODEM 联络信号引脚、辅助引脚。如图 6-3 所示，CH342 中的数据传输引脚包括：TXD 引脚和 RXD 引脚，两个电源引脚：VIO 引脚和 VBUS 引脚，UD+ 和 UD- 引脚分别连接 USB 总线上。

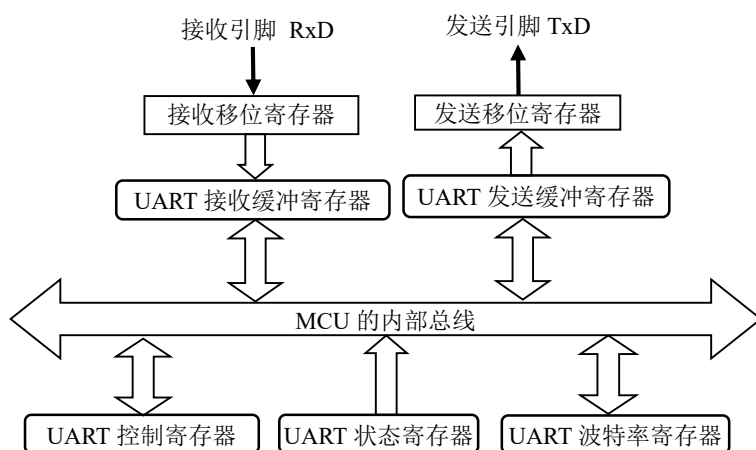


图6-4 UART编程模型

为了设置波特率，UART 应具有波特率寄存器。为了能够设置通信格式、是否校验、是否允许中断等，UART 应具有控制寄存器。而要知道串口是否有数据可收、数据是否发送出去等，需要有 UART 状态寄存器。当然，若一个寄存器不够用，控制与状态寄存器可能多个。而 UART 数据寄存器存放要发送的数据，也存放接收的数据，这并不冲突，因为发送与接收的实际工作是通过“发送移位寄存器”和“接收移位寄存器”完成的。编程时，程序员并不直接与“发送移位寄存器”和“接收移位寄存器”打交道，只与数据寄存器打交道，所以 MCU 中并没有设置“发送移位寄存器”和“接收移位寄存器”的映像地址。发送时，程序员通过判定状态寄存器的相应位，了解是否可以发送一个新的数据。若可以发送，则将待发送的数据放入“UART 发送缓冲寄存器”中就可以了，剩下的工作由 MCU 自动完成：将数据从“UART 接收缓冲寄存器”送到“发送移位寄存器”，硬件驱动将“发送移位寄存器”的数据一位一位地按照规定的波特率移到发送引脚 TxD，供对方接收。接收时，数据一位一位地从接收引脚 RxD 进入“接收移位寄存器”，当收到一个完整字节时，MCU 会自动将数据送入“UART 数据寄存器”，并将状态寄存器的相应位改变，供程序员判定并取出数据。

6.2 基于构件的串行通信编程方法

最基本的 UART 编程涉及初始化、发送和接收三种基本操作。本节主要给出 UART 构件主要 API 接口函数、UART 构件的测试方法以及类似于 PC 机程序调试用的 printf 函数设置与使用方法。

6.2.1 CH32V307VCT6芯片UART对外引脚

CH32V307VCT6 共有八组 UART 引脚，分别标记为 USART1~USART3、UART4~UART8，其中 USART 也可用于同步通信，本章仅给出异步串行通信编程。每个 UART 的

发送数据引脚记为 UARTx_TX, 接收数据引脚记为 UARTx_RX。“x”表示串口模块编号, 表 6-2 给出了本书随附的 AHL-CH32V307 嵌入式开发套件直接引出的 1~3 串口的硬件引脚。

表6-2 UART引脚分布

串行口	MCU引脚号	MCU引脚名	串口号	AHL-CH32V307VT6默认使用
UART1	68	PA9	UART1_TX	编程默认使用 (UART_Debug, BIOS保留使用)
	69	PA10	UART1_RX	
	92	PB6	UART1_TX	
	93	PB7	UART1_RX	
UART2	25	PA2	UART2_TX	编程默认使用 (UART_User)
	26	PA3	UART2_RX	
UART3	47	PB10	UART3_TX	
	48	PB11	UART3_RX	
	78	PC10	UART3_TX	编程默认使用 (保留连接无线通信芯片使用)
	79	PC11	UART3_RX	

这里以 UART1 为例说明一下为什么一个串口有两组或两组以上引脚问题。从表 6-2 中看出 UART1 有两组引脚, 分别是 (68、69) 和 (92、93), 可以从芯片的引脚布局图 (图 3-1) 看出, 这两组属于封装的不同位置, 实际使用时, 用哪一组, 取决于哪边引出方便, 以减少布线长度, 提高稳定性, 这属于芯片设计细节的考量。编程时, 通过相应端口的模式寄存器设置决定使用哪组引脚。

6.2.2 UART构件API

1. UART常用接口函数简明列表

UART 构件主要 API 接口函数有: 初始化、发送一个字节、发送 N 个字节、发送字符串、接收一个字节等等, 如表 6-3 所示。

表6-3 UART常用接口函数

序号	函数名	简明功能	描述
1	uart_init	初始化	传入串口号及波特率, 初始化串口
2	uart_send1	发送一个字节数据	向指定串口发送一个字节数据
3	uart_sendN	发送N个字节数据	向指定串口发送N个字节数据
4	uart_send_string	发送字符串	向指定串口发送字符串
5	uart_re1	接收一个字节数据	从指定串口接收一个字节数据
	...		

2. UART构件的头文件uart.h

UART 构件的文件 uart.h 在工程的 “\03_MCU\MCU_drivers” 文件夹中, 这里给出部分 API 接口函数的使用说明及函数声明。

```
//=====
```

```

//函数名称: uart_init
//功能概要: 初始化uart模块
//参数说明: uartNo—串口号, 如UART_1、UART_2、UART_3
//          baud_rate 一波特率, 可取9600、19200、115200...
//函数返回: 无
//=====
void uart_init(uint8_t uartNo,  uint32_t baud_rate);

//=====

//函数名称: uart_send1
//参数说明: uartNo—串口号:如 UART_1、UART_2、UART_3、...
//          ch—要发送的字节
//函数返回: 函数执行状态, 1表示发送成功; 0表示发送失败
//功能概要: 串行发送1个字节
//=====
uint_8 uart_send1( uint8_t uartNo,  uint8_t ch);

//=====

//函数名称: uart_sendN
//参数说明: uartNo—串口号:如 UART_1、UART_2、UART_3、...
//          buff: 发送缓冲区
//          len:发送长度
//函数返回: 函数执行状态: 1=发送成功; 0=发送失败
//功能概要: 串行 接收n个字节
//=====
uint8_t uart_sendN(uint8_t uartNo ,  uint16_t len ,  uint8_t* buff)

//=====

//函数名称: uart_send_string
//参数说明: uartNo—串口号:如 UART_1、UART_2、UART_3、...
//          buff:要发送的字符串的首地址
//函数返回: 函数执行状态: 1=发送成功; 0=发送失败
//功能概要: 从指定UART端口发送一个以'\0'结束的字符串
//=====
uint8_t uart_send_string(uint8_t uartNo,  uint8_t* buff)

//=====

//函数名称: uart_re1

```

```
//参数说明: uartNo—串口号:如 UART_1、UART_2、UART_3、...
//          *fp—接收成功标志的指针, *fp=1表示接收成功; *fp=0表示接收失败
//函数返回: 返回接收的字节
//功能概要: 串行接收1个字节
//=====
uint_8 uart_re1(uint_8 uartNo, uint_8 *fp);
.....
```

6.2.3 UART构件API的发送测试方法

现在编写 MCU 程序, 通过一个串口把数字“48~100”发送到 PC。在 PC 中, 通过 AHL-GEC-IDE 的“工具”→“串口工具”获得接收信息, 由此体会数据从 MCU 发送出去的过程。

1. MCU方程序的编制

(1) 确定 MCU 串口号、所接 MCU 的引脚。这是硬件制版决定的, UART 构件的头文件 uart.h 中给出了该构件所使用的引脚信息, 。在 user.h 中宏定义本工程使用的串口名为 UART_User, 以便增强编程的可移植性。

(2) 在 main.c 中, 首先确定串口 UART_User 的波特率, 并对其进行初始化, 代码如下。

```
uart_init(UART_User, 115200); //初始化串口模块
```

(3) 在 main.c 的主循环中, 发送数字“48~100”, 代码如下。

```
for (mi=48;mi<=100;mi++)
{
    uart_send1(UART_User, mi);
}
```

2. 编译下载测试

MCU 方的样例工程在“..\CH06\UART_CH32V307_Sent”文件夹中, 可以编译下载测试体会, 并自行练习。

【思考一下】编制程序发送数字 0~255, 若用 8 位无符号数作为循环变量, 注意一下可能遇到的问题。

6.2.4 printf的设置方法与使用

除了使用 UART 驱动构件中封装的 API 函数之外, 还可以使用格式化输出函数 printf 灵活地从串口输出调试信息, 配合 PC 或笔记本电脑上的串口调试工具, 可方便地进行嵌入式程序的调试。上一小节的例程中, 就使用了 printf 函数, 这里说明一下。

printf 函数的实现在工程的“..\05_UserBoard\printf.c”文件中, 同文件夹下的 printf.h 头文件则包含了 printf 函数的声明, 在同文件下的 user.h 头文件中包含 printf.h 头文件, 若

要使用 printf 函数，可在工程的总头文件 “..\ 07_AppPrg \includes.h” 中将 user.h 包含进来，以便其他文件使用。

在使用 printf 函数之前，需要先进行相应的设置来将其与希望使用的串口模块关联起来，设置步骤如下。

(1) 在 printf 头文件 “..\ 05_UserBoard \printf.h” 中宏定义需要与 printf 相关联的调试串口号，例如：

```
#define UART_printf (printf函数使用的串口号) //这里给出具体的串口号
```

(2) 在使用 printf 前，调用 UART 驱动构件中的初始化函数对使用的调试串口进行初始化，配置其波特率。例如：

```
uart_init(UART_printf, 115200); //初始化"调试串口"
```

这样就将相应的串口模块与 printf 函数关联起来了。由于 BIOS 已经对其初始化，因此 User 中可以不再重新初始化。关于 printf 函数的使用方法，参见 printf.h 文件的尾部。

【思考一下】使用 printf 输出一个浮点数，保留 6 位小数。

6.3 UART 构件的制作过程

在第 4 章中介绍过 GPIO 构件的制作过程，这里把制作一个底层驱动构件的基本过程总结一下。首先，要掌握其通用知识；其次，了解是否有对外引脚；第三，了解有哪些寄存器；第四，若能简单实现其基本流程，最好能打通流程；第五，制作构件；第六，测试构件。

6.3.1 UART 寄存器概述

UART 寄存器的基本描述在《CH32FV2X_V3X 系列应用手册》的第 18 章，仔细理解寄存器的基本含义是制作构件的第一环节，这里给出主要寄存器的基本功能概要，如表 6-4 所示。

表6-4 UART寄存器功能概述

寄存器	功能概述
控制寄存器	有三个控制寄存器，用于设定串行通信的格式；设定是否允许接收中断；设定允许发送与接收等
波特率寄存器	设定波特率
状态寄存器	串行口工作时的各种状态标志
数据寄存器	8~0位有效，第8位为奇偶校验位，7~0为数据位

关于 UART 寄存器附加说明如下：

(1) 寄存器地址。UART1 的基地址可查阅《CH32FV2X_V3X 系列应用手册》的第 18.10 节-寄存器描述，查表可知各串口首地址分别是 USART1: 0x4001_3800; USART2: 0x4000_4400; USART3: 0x4000_4800。首地址也是各串行口的基地址，相关寄存器加上各自偏移量即可得其绝对地址

(2)串口类型问题。上述缩写词 USART(Universal Synchronous Receiver Transmitter) 含有同步通信功能, 区别于只有异步通信功能的 UART(Universal Asynchronous Receiver Transmitter), 本书只涉及普通的 UART 方式的编程。

6.3.2 利用直接地址操作的串口发送打通过序

制作 UART 构件, 要考虑到各种通用要素, 如: 串行口的选择、工作方式的选择、寄存器的选择、初始化编程等, 要直接编写一个完整且可稳定运行的构件是很难的, 开发人员一般会考虑先试着发送一个字符至 PC 端, 完整实现串行口正常工作的全过程, 包括: 寄存器赋值、引脚复用的选择、相关标志位的置位或复位等, 然后利用 PC 端能稳定运行的程序接收数据, 如果能成功接收到数据则说明发送过程是可行的。

本节给出用直接对端口进行编程的方法使用 UART 发送单个字节, 这是最简单的串口发送程序, 是制作构件的先导步骤。UART 直接地址的测试工程位于 “..\04-Software\CH06\UART-CH32V307-ADDR” 文件夹。使用 AHL-CH32V307 开发套件上的 UART_User 串口发送数据, 该串口对应的 MCU 硬件引脚见电子资源 “..\03-Hardware\component.pdf” 文件, 软件编程在工程的 “..\03_MCU\MCU_drivers\uart.h” 文件中进行宏定义。

1. 定义地址变量

其中 volatile 是变量修饰符, volatile 关键字可以用来提醒编译器它后面所定义的变量随时有可能改变, 因此编译后的程序每次需要存储或读取这个变量的时候, 都会直接从变量地址中读取数据。如果没有 volatile 关键字, 则编译器可能优化读取和存储, 可能暂时使用寄存器中的值, 如果这个变量由别的程序更新了的话, 将出现不一致的现象。

```
volatile uint32_t* RCC_APB2;    //GPIO的A口时钟使能寄存器地址
volatile uint32_t* RCC_APB1;    //UART2口时钟使能寄存器地址
volatile uint32_t* gpio_mode;   //引脚模式寄存器地址=口基地址
volatile uint32_t* uart_brr;    //UART波特率寄存器地址
volatile uint32_t* uart_isr;    //UART中断和状态寄存器基地址
volatile uint32_t* uart_cr1;    //UART控制寄存器1基地址
volatile uint32_t* uart_cr2;    // UART控制寄存器2地址
volatile uint32_t* uart_cr3;    // UART控制寄存器3基地址
volatile uint32_t* uart_tdr;    // UART发送数据寄存器
uint16_t usartdiv;              //BRR寄存器应赋的值
```

2. 给地址变量赋值

根据《CH32FV2X_V3X 系列应用手册》中查得的地址给相关寄存器赋值, 具体说明一例。如: RCC_APB1PCENR 是外设时钟使能寄存器, 其绝对地址为 0x4002101CUL, 其中, “0x”表示 16 进制数据, “UL”表示表示无符号长整型, 如果不写 UL 后缀, 系统默认为: int, 即有符号整数。

```
//变量赋值, 各寄存器值均可通过芯片参考手册得到
```

```

RCC_APB1= 0x4002101CUL;           //UART时钟使能寄存器地址
RCC_APB2= 0x40021018UL;           //GPIO的A口时钟使能寄存器地址
gpio_mode= 0x40010800UL;           //引脚模式寄存器地址=口基地址
uart_cr1= 0x4000440CUL;            //UART2控制寄存器1地址
uart_brr= 0x40004408UL;            //UART2波特率寄存器地址
uart_isr= 0x40004400UL;            // UART2中断和状态寄存器地址
uart_tdr= 0x40004404UL;            //UART2发送数据寄存器
uart_cr2= 0x40004410UL;            // UART2控制寄存器2地址
uart_cr3= 0x40004414UL;            //UART2控制寄存器3地址

```

3. UART初始化步骤

本例通过 USART2 向 PC 发送字符，所以需要对 PTA3 和 PTA2 进行复用定义，并设置相应波特率参数。

(1)设置引脚复用功能为串口。通过 GPIO 模块的端口模式寄存器(GPIOA_CFGLR)设定引脚为复用功能模式；通过 GPIO 复用功能低位寄存器 (GOPIA_AFIO) 设置设定为 UARTx_TX 和 UARTx_RX。

```

//使能GPIOA和UART2的时钟
*RCC_APB1|=(0x1UL<<17U);           //UART2时钟使能①
*RCC_APB2 |=(0x1UL<<2U);           //GPIOA时钟使能

//将GPIO端口设置为复用功能
//首先将D7、D6、D5、D4清零
*gpio_mode &= ~((0x3UL<<10U)|(0x3UL<<12U));
//然后将D7、D6、D5、D4设为1010，设置PTA2、PTA3为复用功能串行功能。
*gpio_mode |=((0x2UL<<10U)|(0x2UL<<8U)|(0x4UL<<12U));

//暂时禁用UART功能，控制寄存器1的第0位对应的是UE—USART使能位。
//此位清零后，USART预分频器和输出将立即停止，并丢弃所有当前操作。
*uart_cr1 &= ~(0x1UL<<13);

//暂时关闭串口发送与接收功能，控制寄存器1的发送器使能位 (D3)、接收器使能位 (D2)
*uart_cr1 &= ~((0x1UL<<3U)|(0x1UL<<2U));

```

(2)设置波特率。通过 UART 波特率寄存器 (UART_BRR) 设定使用什么速度收发字节，本节设定为 115200。计算时根据 USART_CR1 寄存器中第 15 位对应的过采样模式设置，波特率计算公式有所不同，记系统内核时钟频率为 f_{sysclk} 。

标志位为 1 时：波特率 $= \frac{f_{sysclk} * 2}{115200}$

标志位为 0 时：波特率 $= \frac{f_{sysclk}}{115200}$ 。

此处 $f_{sysclk}=72\text{MHZ}$ ，随后将计算得到的数值写入波特率寄存器。

```

//配置波特率
usartdiv = (uint16_t)((SYSCLK_FREQ_72MHz/(16*2*115200))<<4)|

```

^① 此处可通过关键字“RCC_APB1”查找 CH32FV2x_V3x 系列应用手册，可知，该寄存器第 17 位对应的是 UART2 时钟使能位。

```
(((100000*SYSCCLK_FREQ_72MHz/(16*2*115200))%100000)*16)/100000));
*uart_brr = (uint16_t)usartdiv;
```

(3) 开启 UART 功能。通过 UART 控制寄存器 (UART_CR1, UART_CR2 和 UART_CR3) 开启 UART 功能, 启动串口发送与接收功能。

```
//初始化控制寄存器和中断状态寄存器、清标志位
//关中断
*uart_isr = 0x0UL;
//将控制寄存器2的两个使能位清零。D14—LIN模式使能位、D11—时钟使能位①
*uart_cr2 &= ~((0x1UL<<14U)|(0x1UL<<11U));
//将控制寄存器3的三个使能位清零。D5 (SCEN) —smartcard模式使能位、
//D3 (HDSEL) —半双工选择位、D1 (IREN) —IrDA 模式使能位
*uart_cr3 &= ~((0x1UL<<5U) | (0x1UL<<3U) |(0x1UL<<1U));

//启动串口发送与接收功能
*uart_cr1 |= ((0x1UL<<3U)|(0x1UL<<2U));

//开启UART功能
*uart_cr1 |= (0x1UL<<13U);
```

4. 发送数据

样例中循环发送 ASCII 值为“48~100”的字符至 PC 机显示, 下面为发送代码。

```
for (mi=48;mi<=100;mi++)
{
    //对应uart_send1(UART_User,mi);
    //发送缓冲区为空则发送数据
    for (volatile uint32_t j=0;j<0xFBBB;j++)
    {
        if (*uart_isr & (0x1UL<<7UL))
        {
            *uart_tdr = (mi & USART_Datar_DR);
            break;
        }
    }
}
```

可以看到, 这是个比较复杂的过程, 并且需要在确定硬件正确的前提下, 不断地找错误才能完成, 说明了寄存器级编程的复杂性。

^① 通过关键字“USART2_CTLR2”在 CH32FV2x_V3x 系列应用手册中查找, 可查得其各位的定义。

6.3.3 UART构件设计

1. UART驱动构件封装要点分析

UART 具有初始化、发送和接收三种基本操作。下面分析串口初始化函数的参数应该有哪些。首先应该有串口号，因为一个 MCU 有若干串口，你必须确定使用哪个串口；其次是波特率，因为必须确定串口使用什么速度收发。关于奇偶校验，由于实际使用主要是多字节组成的一个帧，自行定义通信协议，单字节校验意义不大；此外，串口在嵌入式系统中的重要作用是实现类似 C 语言中 `printf` 函数功能，也不宜使用单字节校验，因此就不校验。这样，串口初始化函数就两个参数：串口与波特率。

从知识要素角度，进一步分析 UART 驱动构件的基本函数，与寄存器直接打交道的有：初始化、发送单个字节与接收单个字节的函数，以及使能及禁止接收中断、获取接收中断状态的函数。发送中断不具有实际应用价值，可以忽略。

设计 UART 构件的目的是为了可以实现对所有包含 UART 功能的引脚统一编程。UART 构件是由 `uart.h` 和 `uart.c` 两个文件组成。将这两个文件加到工程的“`..03_MCU\MCU_drivers`”文件夹下，由此方便了对 UART 的编程操作。

(1) 模块初始化 (`uart_init`)。芯片引脚有复用功能，应该将 GPIO 引脚设置为复用功能 `UARTx_TX` 和 `UARTx_RX`。同时通过传入波特率确定收发速度。函数不必有返回值，故 UART 模块的初始化函数原型可以设计为：

```
void uart_init(uint8_t uartNo, uint32_t baud_rate);
```

(2) 发送一个字节 (`uart_send1`)。开发套件发送一个字节，需要确定是由哪一个串口发出，发出的数据是什么，并由返回值告诉用户发送是否成功。故应该有返回值，返回值 0 表示发送失败，1 表示发送成功。这样发送一个字节的函数原型可以设计为：

```
uint8_t uart_send1(uint8_t uartNo, uint8_t ch);
```

(3) 发送 N 个字节、字符串。类似的分析，可以设计发送 N 个字节和字符串函数的原型为：

```
uint8_t uart_sendN(uint8_t uartNo, uint16_t len, uint8_t* buff)
uint8_t uart_send_string(uint8_t uartNo, uint8_t* buff)
```

(4) 其他函数。继续设计接收一个字节、接收 N 个字节、使能串口中断、禁止串口中断等函数原型，基本完成头文件的设计。

2. UART端口寄存器结构体类型

通常在构件设计中把一个模块的寄存器用一个结构体类型封装起来，方便编程时使用，这些结构体存放在工程文件夹的芯片头文件（“`..03_MCU\startup\ch32v30x.h`”）中，串行模块结构体类型为 `USART_TypeDef`。


```
typedef struct
{
    __IO uint16_t  STATR;
    uint16_t      RESERVED0;
    __IO uint16_t  DATAR;
    uint16_t      RESERVED1;
    __IO uint16_t  BRR;
    uint16_t      RESERVED2;
    __IO uint16_t  CTLR1;
    uint16_t      RESERVED3;
    __IO uint16_t  CTLR2;
    uint16_t      RESERVED4;
    __IO uint16_t  CTLR3;
    uint16_t      RESERVED5;
    __IO uint16_t  GPR;
    uint16_t      RESERVED6;
} USART_TypeDef;
```

CH32V307VCT6 的 UART 模块各口基地址也在芯片头文件(ch32v30x.h)中以宏常数方式给出，直接作为指针常量。

3. UART驱动构件源程序的制作

UART 驱动构件的源程序文件中实现的对外接口函数，主要是对相关寄存器进行配置，从而完成构件的基本功能，构件内部使用的函数也在构件源程序文件中定义，构件中函数的制作过程应在已经打通的基本功能基础上（参考 6.3.2 节），先常量后变量，一步一步调试推进，下面给出 `uart_init` 函数源代码。

```
//=====
//文件名称: uart.c
//功能概要: uart 底层驱动构件源文件
//版权所有: 苏州大学嵌入式系统与物联网研究所(sumcu.suda.edu.cn)
//更新记录: 2022-01-06
//=====
#include "uart.h"
USART_TypeDef *USART_ARR[] = {(USART_TypeDef*)USART1_BASE,
(USART_TypeDef*)USART2_BASE, (USART_TypeDef*)USART3_BASE};
//====定义串口 IRQ 号对应表====
IRQn_Type table_irq_uart[3] = {USART1_IRQn, USART2_IRQn, USART3_IRQn};
//内部函数声明
uint_8 uart_is_uartNo(uint_8 uartNo);
//=====
//函数名称: uart_init
//功能概要: 初始化 uart 模块
//参数说明: uartNo:串口号: UART_1、UART_2、UART_3
//          baud:波特率: 4800、9600、19200、115200...
//函数返回: 无
//=====
void uart_init(uint_8 uartNo, uint_32 baud_rate)
{
```

```

uint16_t DIV_M, DIV_F; //BRR 寄存器应赋的值
//判断传入串口号参数是否有误，有误直接退出
if(!uart_is_uartNo(uartNo)) return;
//开启 UART 模块和 GPIO 模块的外围时钟，并使能引脚的 UART 功能
switch(uartNo)
{
case UART_1:    //若为串口 1
#ifdef UART1_GROUP
    //依据选择使能对应时钟，并配置对应引脚为 UART_1
    switch(UART1_GROUP)
    {
case 0:
        //使能 USART1 和 GPIOA 时钟
        RCC->APB2PCENR |= RCC_USART1EN;
        RCC->APB2PCENR |= RCC_AFIOEN;
        RCC->APB2PCENR |= RCC_IOPAEN;
        //使能 PTA9, PTA10 为 USART(Tx, Rx)功能
        GPIOA->CFGHR &= ~(GPIO_CFGHR_MODE9|GPIO_CFGHR_MODE10);
        GPIOA->CFGHR |= ((GPIO_CFGHR_MODE9_1|GPIO_CFGHR_CNF9_1)|
                        (GPIO_CFGHR_CNF10_1));
        AFIO->PCFR1 |= 0;

        break;
case 1:
        //使能 USART1 和 GPIOB 时钟
        RCC->APB2PCENR |= RCC_USART1EN;
        RCC->APB2PCENR |= RCC_IOPBEN;
        RCC->APB2PCENR |= RCC_AFIOEN;
        //使能 PTB6, PTB7 为 USART(Tx, Rx)功能
        GPIOB->CFGLR &= ~(GPIO_CFGLR_CNF6|GPIO_CFGLR_CNF7);
        GPIOB->CFGLR |= ((GPIO_CFGLR_CNF6_1|GPIO_CFGLR_MODE6_1)|
                        GPIO_CFGLR_CNF7_0);
        AFIO->PCFR1 |= AFIO_PCFR1_USART1_REMAP;
        break;
default:
        break;
    }
}
#endif
break;
case UART_2:    //若为串口 2
#ifdef UART2_GROUP
    //依据选择使能对应时钟，并配置对应引脚为 UART_2
    switch(UART2_GROUP)
    {
case 0:
        //使能 USART2 和 GPIOA 时钟
        RCC->APB1PCENR |= RCC_USART2EN;
        RCC->APB2PCENR |= RCC_IOPAEN;
        //使能 PTA2, PTA3 为 USART(Tx, Rx)功能

```

```

        GPIOA->CFGLR &= ~(GPIO_CFGLR_CNF2|GPIO_CFGLR_CNF3);
        GPIOA->CFGLR |= ((GPIO_CFGLR_CNF2_1|GPIO_CFGLR_MODE2_1|
            (GPIO_CFGLR_CNF3_0));

        break;
    default:
        break;
    }
#endif

    break;
case UART_3:    //若为串口 3
    ..... (限于篇幅, 省略其他串口)
    break;
}
//暂时禁用 UART 功能
USART_ARR[uartNo-1]->CTLR1 &= ~USART_CTLR1_UE;
//暂时关闭串口发送与接收功能
USART_ARR[uartNo-1]->CTLR1 &= ~(USART_CTLR1_TE|USART_CTLR1_RE);
//配置串口波特率
if(USART_ARR[uartNo-1]==(USART_TypeDef*)USART1_BASE)
{
    DIV_M = (uint16_t)(SYSCLK_FREQ_72MHz/(16*baud_rate));
    DIV_F = (uint16_t)((((10000*SYSCLK_FREQ_72MHz/(16*baud_rate))%10000)*16)/1000);
    USART_ARR[uartNo-1]->BRR = (uint16_t)(DIV_M<<4|DIV_F);
}
else
{
    DIV_M = (uint16_t)(SYSCLK_FREQ_72MHz/(16*2*baud_rate));
    DIV_F = (uint16_t)((((100000*SYSCLK_FREQ_72MHz/(16*2*baud_rate))%100000)*16)
        /100000);
    USART_ARR[uartNo-1]->BRR = (uint16_t)((DIV_M)<<4|DIV_F);
}
//初始化控制寄存器和中断状态寄存器、清标志位
USART_ARR[uartNo-1]->STATR = 0;
USART_ARR[uartNo-1]->CTLR2 &= ~(USART_CTLR2_LINEN | USART_CTLR2_CLKEN);
USART_ARR[uartNo-1]->CTLR3 &= ~(USART_CTLR3_SCEN | USART_CTLR3_HDSEL |
    USART_CTLR3_IREN);

//启动串口发送与接收功能
USART_ARR[uartNo-1]->CTLR1 |= (USART_CTLR1_TE|USART_CTLR1_RE);
//开启 UART 功能
USART_ARR[uartNo-1]->CTLR1 |= USART_CTLR1_UE;
}
(限于篇幅, 省略其他函数实现, 见电子资源)

```

6.4 中断机制及中断编程步骤

从第 4 章及本章前面的程序可以看出, MCU 启动后跳转到 main 函数执行, 进入一个无限循环, 计算机程序就这样一直运行下去, 但是, 计算机如何处理紧急的任务呢? 这就是中断所要处理的问题。

6.4.1 关于中断的通用基础知识

中断提供了一种程序运行的机制，用来打断当前正在运行的程序，并且保存当前 CPU 状态（CPU 内部寄存器），转而去运行一个中断服务例程，然后恢复 CPU 到运行中断之前的状态，同时使得中断前的程序得以继续运行。

1. 中断的基本概念

1) 中断与异常的基本含义

中断与异常属于同一概念的两种不同产生条件。**异常（exception）**是 CPU 强行从正常的程序运行切换到由某些内部或外部条件所要求的处理任务上去，这些任务的紧急程度优先于 CPU 正在运行的任务。引起异常的外部条件通常来自外围设备、硬件断点请求、访问错误和复位等；引起异常的内部条件通常为指令、不对界错误、违反特权级和跟踪等。一些文献把硬件复位和硬件中断都归类为异常，把硬件复位看作是一种具有最高优先级的异常，而把来自 CPU 外围设备的强行任务切换请求称为**中断（interrupt）**，软件上表现为将程序计数器（PC）指针强制转到中断服务例程入口地址运行。

CPU 对中断与异常具有同样的处理过程，本书随后在谈及这个处理过程时统称为中断。

2) 中断源与中断向量号

可以引起 CPU 中断的外部器件被称为**中断源**。一个 CPU 通常可以识别多个中断源，每个中断源产生中断后，若程序允许，会打断当前正在运行的程序，转而运行相应的**中断服务例程（Interrupt Service Routine, ISR）**。

一个 CPU 能识别多个中断源，芯片制造时，给 CPU 能够识别的各个中断源编个数，就叫**中断向量号**。通常情况下，在程序书写时，中断向量表按中断向量号从小到大的顺序填写中断服务例程 ISR 的首地址，不能遗漏。即使某个中断不需要使用，也要在中断向量表对应的项中填入缺省中断服务例程 ISR 的首地址，因为中断向量表是连续存储区，与连续的中断向量号相对应。有的芯片区分内核中断与非内核中断，把非内核中断的编号称为中断请求（Interrupt Request, IRQ）号，即 IRQ 号，有的芯片不加区分。

中断向量表一般位于芯片工程的启动文件中，以下给出 CH32V307VCT6 的启动文件“startup_ch32v30x.S”中的中断向量表的头部：

```
g_pfnVectors:
    .option norvc;
    j    _start
    .word    0
    j    NMI_Handler           /* NMI Handler */
    j    HardFault_Handler     /* Hard Fault Handler */
    .word    0
```

```
.word    0
...
```

其中，除第一项外的每一项都代表着各个中断服务例程 ISR 的首地址，第一项代表着栈顶地址，一般是程序可用 RAM 空间的最大值。此外，对于未实例化的中断服务例程，由于在程序中不存在具体的函数实现，也就不存在相应的函数地址。因此一般在启动文件内，会采用弱定义的方式，将默认未实例化的中断服务例程 ISR 的起始地址指向一个缺省中断服务例程 ISR 的首地址，这样就保证了所有的中断响应都有一个去处：

```
.weak    NMI_Handler
.weak    HardFault_Handler
.weak    SysTick_Handler
.weak    SW_handler
...
```

其中，这个默认的处理程序一般是一个无限循环语句或是一个直接返回的语句，CH32V307VCT6 采用的方式是无限循环。

给 CPU 能够识别的每个中断源编个号，就叫**中断向量号**。通常情况下，在程序书写时，中断向量表按中断向量号从小到大的顺序填写中断服务例程 ISR 的首地址，不能遗漏。即使某个中断不需要使用，也要在中断向量表对应的项中填入缺省中断服务例程 ISR 的首地址，因为中断向量表是连续存储区，与连续的中断向量号相对应。中断向量号一般从 1 开始，它与 IRQ 中断号一一对应。IRQ 中断号将内核中断与非内核中断稍加区分，对于非内核中断，IRQ 中断号从 0 开始递增，而对于内核中断，IRQ 中断号从-1 开始递减。IRQ 中断号的定义一般位于芯片头文件内，以下给出 CH32V307VCT6 的芯片头文件“ch32v30x.h”中的 IRQ 中断号的部分定义：

```
typedef enum
{
    // RISC-V Processor Exceptions Numbers
    NonMaskableInt_IRQn    = 2,        //!< 2 Non Maskable Interrupt
    EXC_IRQn               = 3,        //!< 4 Exception Interrupt
    SysTicK_IRQn           = 12,       //!< 12 System timer Interrupt
    Software_IRQn           = 14,       //!< 14 software Interrupt
    .....
} IRQn_Type;
```

在“3.1.3 小节”的表 3-4 给出了 CH32V307VCT6 更为而详细的中断源、中断向量号、IRQ 中断号和引用名等信息，这里不再列出。

3) 中断优先级、可屏蔽中断和不可屏蔽中断

在进行 CPU 设计时，一般定义了中断源的优先级。若 CPU 在程序运行过程中，有两个以上中断同时发生，则优先级最高的中断得到最先响应。

根据中断是否可以通过程序设置的方式被屏蔽，可将中断划分为可屏蔽中断和不可屏

蔽中断两种。**可屏蔽中断**是指可以通过程序设置的方式来决定不响应该中断，即该中断被屏蔽了。**不可屏蔽中断**是指不能通过程序方式关闭的中断。

2. 中断处理的基本过程

中断处理的基本过程分为中断请求与中断检测、中断响应与中断处理等过程。

1) 中断请求与中断采样

当某一中断源需要 CPU 为其服务时，它会向 CPU 发出中断请求信号(一种电信号)。中断控制器获取中断源硬件设备的中断向量号^①，并通过识别的中断向量号将对应硬件中断源模块的中断状态寄存器中的“中断请求位”置位，以便 CPU 确定是哪种中断请求。

一般情况下，CPU 在每条指令结束的时候，会检查中断请求或者系统是否满足异常条件，为此，一些 CPU 专门在指令周期中使用了中断周期。在中断周期中，CPU 将会检测系统中是否有中断请求信号，若此时有中断请求信号，则 CPU 将会暂停当前运行的任务，转而去对中断事件进行响应，若系统中没有中断请求信号则继续运行当前任务。

2) 中断响应与中断处理

中断响应的过程是由系统自动完成的，对于用户来说是透明的操作。在中断的响应过程中，首先 CPU 会查找中断源所对应的模块中断是否被允许，若被允许，则响应该中断请求。中断响应的过程要求 CPU 保存当前环境的“上下文(context)”于堆栈中。通过中断向量号找到中断向量表中对应的中断服务例程 ISR，转而去运行该中断服务例程 ISR。中断处理术语中，简单的理解“上下文”即指 CPU 内部寄存器，其含义是在中断发生后，由于 CPU 在中断服务例程中也会使用 CPU 内部寄存器，所以需要在调用 ISR 之前，将 CPU 内部寄存器保存至指定的 RAM 地址(栈)中，在中断结束后再将该 RAM 地址中的数据恢复到 CPU 内部寄存器中，从而使中断前后程序的“运行现场”没有任何变化。

6.4.2 RISC-V非内核模块中断编程结构

1. RISC-V4F中断结构及中断过程

RISC-V4F 中断系统的结构框图，如图 6-5 所示，它由 RISC-V4F 内核、可编程快速中断控制器(Programmable Fast Interrupt Controller, PFIC)及模块中断源组成。其中断过程分为二步，第一步，模块中断源向快速可编程中断控制器 PFIC 发出中断请求信号；第二步，PFIC 对发来的中断信号进行管理，判断该模块中断是否被使能，若使能，通过私有外设总线(Private Peripheral Bus, PPB)发送给 RISC-V4F 内核，由内核进行中断处理。如果同时有多个中断信号到来，PFIC 根据设定好的中断优先级进行判断，优先级高的中断首先响应，优先级低的中断暂时挂起，压入堆栈保存；如果优先级完全相同的多个中断源同时请求，则先响应 IRQ 号较小的，其他的被挂起。例如，当 IRQ4^②的优先级与 IRQ5

^① 设备与中断向量号可以不是一一对应的，如果一个设备可以产生多种不同中断，允许有多个中断向量号。

^② IRQ 中断号为 n，简记为 IRQn

的优先级相等时，IRQ4 会比 IRQ5 先得到响应。

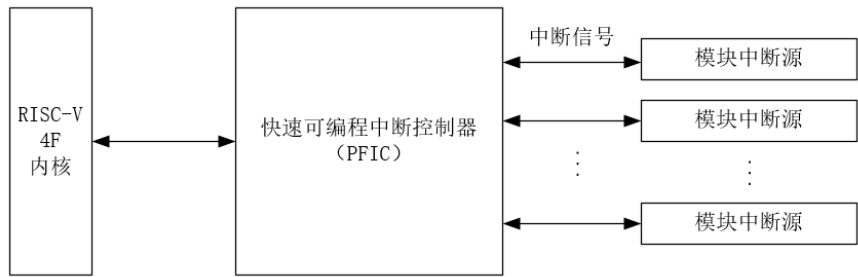


图6-5 V4F中断结构框图

2. PFIC内部寄存器简介

PFIC 模块的基地址（PFIC_BASE）为：0xE000E000，内部用于中断控制的寄存器，如表 6-5 示。在样例工程的 core_riscv.h 文件中定义了一个名为“PFIC_Type”的结构体组织这些寄存器。其中软件触发中断寄存器及中断激活位寄存器比较少用，下面对其他寄存器进行说明。

表6-5 NVIC内各寄存器简表

描 述	地址偏移	使用名称	描 述
中断使能状态寄存器1	0x000	ISR1	只读，1: 31编号以下中断已使能
中断使能状态寄存器2	0x004	ISR2	只读，1: 32编号以上中断已使能
中断挂起状态寄存器1	0x020	IPR1	只读，1: 31编号以下中断已挂起
中断挂起状态寄存器2	0x024	IPR2	只读，1: 32编号以上中断已挂起
中断优先级阈值配置寄存器	0x040	ITHRESDR	只读，设置中断优先级阈值
快速中断服务基地址寄存器	0x044	FIBADDRR	可读/写，快速中断响应的目标跳转地址
中断配置寄存器	0x048	CFGR	可读/写，写1有效
中断全局状态寄存器	0x04C	GISR	只读，判断当前中断状态
快速中断0偏移地址寄存器	0x060	FIOFADDRR0	可读/写，快速中断0进行地址偏移
快速中断1偏移地址寄存器	0x064	FIOFADDRR1	可读/写，快速中断1进行地址偏移
快速中断2偏移地址寄存器	0x068	FIOFADDRR2	可读/写，快速中断2进行地址偏移
快速中断3偏移地址寄存器	0x06C	FIOFADDRR3	可读/写，快速中断3进行地址偏移
中断使能设置寄存器 1	0x100	IENR1	可读/写，31#以下中断只能控制
中断使能设置寄存器 2	0x104	IENR2	可读/写，32#以上中断只能控制
中断使能清除寄存器 1	0x180	IRNR1	可读/写，31#以下中断关闭控制
中断使能清除寄存器 2	0x184	IRNR2	可读/写，32#以上中断关闭控制
中断挂起设置寄存器 1	0x200	IPSR1	可读/写，31#以下中断挂起设置

中断挂起设置寄存器 2	0x204	IPSR2	可读/写，32#以上中断挂起设置
中断挂起清除寄存器 1	0x280	IPRR1	可读/写，31#以下中断挂起清除
中断挂起清除寄存器 2	0x284	IPRR2	可读/写，32#以上中断挂起清除
中断激活状态寄存器 1	0x300	IACTR1	可读/写，31#以下中断执行状态
中断激活状态寄存器 2	0x304	IACTR2	可读/写，32#以上中断执行状态
中断优先级配置寄存器	0x400-0x4FF	IPRIORx (x=0-63)	可读/写，配置中断优先级
系统控制寄存器	0xD10	SCTLR	可读/写，设置系统模式

1) 中断使能寄存器

中断使能设置寄存器（Interrupt Set Enable Register, IENR）有 2 个，中断向量号 IRQ 为 0-31 时使用的是中断使能设置寄存器 1（IENR1），中断向量号 IRQ 在 32-59 之间时使用的是中断使能设置寄存器 2（IENR2）。每个寄存器为 32 位宽，每个位对应于一个中断源，对相应写 1，表示设置对应中断源使能，即允许其中断，写 0，无效。例如，设置 UART1 的接收中断使能，首先在 CH32V307VCT6 中断向量表（表 3-4）中查找 UART1 接收中断的 IRQ 号为 53，对应中断使能寄存器为 IENR 的第 21 位，由于对中断使能寄存器的某一位写 0 无效，则设置 IENR2 的第 21 位=1，用进制表示可以写成：IENR[1]=00000000_00100000_00000000_00000000。这个表达方式写成共性函数见工程的“..\02_CPU\core_riscv.h”文件的 NVIC_EnableIRQ 函数。

```
RV_STATIC_INLINE void NVIC_EnableIRQ(IRQn_Type IRQn)
{
    NVIC->IENR[((uint32_t)(IRQn) >> 5)] = (1 << ((uint32_t)(IRQn) & 0x1F));
}
```

这个函数对于具体的 UART1 接收中断来说，由于 IRQn=53，该函数实参(((uint32_t)IRQn) >> 5)=1，等号右边(uint32_t)(1 << (IRQn& 0x1F))，就是二进制 00000000_00100000_00000000_01000000。

2) 中断使能清除寄存器

中断使能清除寄存器（Interrupt Clear Enable Register, IRER）有 2 个，中断向量号 IRQ 为 0-31 时使用的是中断使能设置寄存器 1（IRER1），中断向量号 IRQ 在 32-59 之间时使用的是中断使能设置寄存器 2（IRER2）。为 32 位宽，每个位对应一个中断源，对相应写 1，表示清除对应中断源的使能（该位变为 0），即禁止其中断，写 0，则无效。“..\02_CPU\core_riscv.h”文件的 NVIC_DisableIRQ 函数可以使用。

3) 中断设置挂起/清除挂起寄存器

当中断发生时，正在处理同级或高优先级中断，或者该中断被屏蔽，则中断不能立即得到响应，此时中断可被暂时挂起。中断的挂起状态通过中断设置挂起寄存器（Interrupt Pending Set Register, IPSR）与中断清除挂起寄存器（Interrupt Pending Clear Register, IPPR）来读取，还可以通过写这些寄存器进行挂起中断。其中，挂起表示排队等待，清除挂起表

示取消此次中断请求。

4) 中断优先级配置寄存器

每个中断都有对应的优先级寄存器，其数量取决于芯片中实际存在的外部中断数，CH32V307VCT6 使用数组元素 IP[0]~[255]表示，每个中断使用 8bit 来设置控制优先级，但只使用高 4 位，可表示 0~255 优先级，优先级数值越小表示优先级越高。只有两级中断嵌套，即只能抢占 1 次。要获得一个芯片实际使用多少位表达优先级，可以用下述方法进行测试：将 0xFF 写入任意中断优先级寄存器，随后将其读回后查看多少位为 1，若设备实际实现了 8 个优先级（3 位），读回值为 0xE0。若不对某一中断的优先级进行配置，默认为 0（最高优先级），在使用实时操作系统时，建议设置外部中断优先级。

【思考一下】在样例工程中，找出表 3-4 中串口（USART）2 的中断使能寄存器的名称、地址。

3. 非内核中断初始化设置步骤

根据本节给出的 RISC-V4F 非内核模块中断编程结构，想让一个非内核中断源能够得到内核响应（或禁止），基本步骤如下：

（1）设置模块中断使能位使能模块中断，使模块能够发送中断请求信号。例如 UART 模式下，在 USART_STATR 中，将中断使能位置 1。

（2）查找芯片中断源表（例如表 3-4）找到对应 IRQ 号，设置可编程快速中断控制器的中断使能寄存器（PFIC_IENR），使该中断源对应位置 1，允许该中断请求。反之，若要禁止该中断，则设置嵌套中断向量控制器的中断禁止寄存器（PFIC_IRER），使该中断源对应位置 1 即可。

（3）若要设置其优先级，可对优先级寄存器编程。

本书电子资源的例程，已经在各外设模块底层驱动构件中封装了模块中断使能与禁止的函数，可直接使用。这里阐述目的是为了使得读者理解其中的编程原理。读者只要选择一个含有中断的构件，理解其使能中断与禁止中断函数即可。

6.4.3 CH32V307VCT6中断编程步骤一以串口接收中断为例

第 3.1 节中给出了 CH32V307VCT6 的中断源及中断向量表。下面以 UART_2 接收中断为例，阐述 CH32V307VCT6 中断编程步骤。样例工程为“..\04-Software\CH06\ UART-CH32V307-ISR”。

1. 准备阶段

在开发板硬件设计阶段确定使用的串口，用它来收发数据，例如 AHL-CH32V307 中的 UART_User，也就是 UART_2。

在“..\03_MCU\startup\startup_ch32v30x.S”文件的中断向量表中，找到串口 2 接收中断服务例程的函数名是 USART2_IRQHandler。同时在“..\05_UserBoard\user.h”文件中，对其宏定义，增强程序的可移植性：

```
#define UART_User          UART_2          //UART_2可用模块宏定义，用户串口
#define UART_User_Handler  USART2_IRQHandler //用户串口中断函数宏定义
```

2. main.c文件中的编程—串口初始化、使能模块中断、开总中断

(1) 在“初始化外设模块”位置调用 `uart` 构件中的初始化函数：

```
uart_init(UART_User 115200); //初始化串口模块，波特率使用 115200
```

(2) 在“初始化外设模块”位置调用 `uart` 构件中的使能模块中断函数：

```
uart_enable_re_int(UART_User); //使能 UART_USER 模块接收中断功能
```

(3) 在“开总中断”位置调用 `cpu.h` 文件中的开总中断宏函数：

```
ENABLE_INTERRUPTS; //开总中断
```

这样，串口接收中断初始化完成。

3. isr.c文件中编程—中断服务例程

紧接着，可以在“`..\07_AppPrg \isr.c`”文件中进行中断服务例程的编程。

```
//=====
//中断名称: UART_User_Handler
//功能概要: UART_UPDATE 接收中断，处理接收到的数据。
//参 数: 无
//返 回: 无
//说 明: 需要启动中断并注册才可使用
//=====
void UART_User_Handler(void)
{
    uint8_t ch;
    uint8_t flag;

    DISABLE_INTERRUPTS; //关总中断
    //接收一个字节的的数据
    ch = uart_re1(UART_2,&flag); //调用接收一个字节的函数，清接收中断位
    if(flag) //有数据
    {
        uart_send1(UART_2,ch); //回发接收到的字节
    }
    ENABLE_INTERRUPTS; //开总中断
}
```

就可在此处进行串口 2 接收中断功能的编程了。这里的函数会取代原来的缺省函数。这样就避免了用户直接对中断向量表进行修改，而 `startup_ch32v10x.S` 文件中采用“弱定义”的方式为用户提供编程接口，既方便用户使用，同时也提高了系统编程的安全性。

中断服务例程的设计与普通构件函数设计是一样的，只是这些程序只有在中断产生时才被运行。为了规范编程，统一将各个中断服务例程，放在工程框架中的“`..\07_AppPrg \isr.c`”文件中。如编写一个 `UART_User` 串口接收中断服务例程，当串口有一个字节的数据到来时产生接收中断，将会执行 `UART_User_Handler` 函数。在这个程序中，首先进入临界区^①，关总中断，接收一个到来的字符，若接收成功，则把这个字符发送回去，退出

^① 有些情况下，一些程序段是需要连续执行而不能被打断的，此时，程序对 CPU 资源的使用是独

临界区。

4. 运行结果

将机器码文件下载到目标开发套件中，在 AHL-GEC-IDE 的“工具”→“串口工具”菜单下，弹出串口测试工程界面，选择好串口，设置波特率为 115200，点击“打开串口”，选择发送方式为“字符串”，在文本框内输入字符内容“A”，点击“发送数据按钮”，则上位机将该字符串发送给 MCU。MCU 接收数据后回发给上位机，如图 6-6 所示。

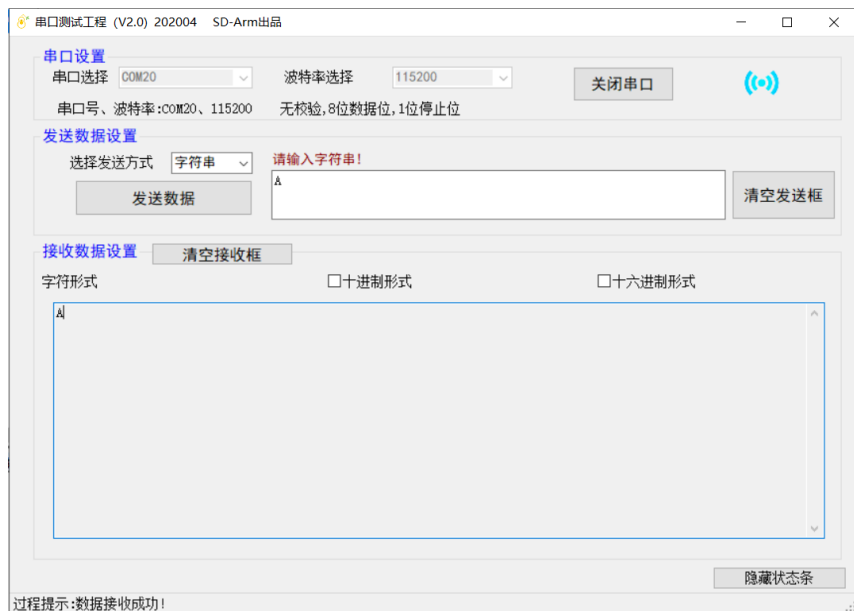


图6-6 通过中断实现串口的收发数据

【思考一下】实现上位机发送“A”，MCU 回发“C”，上位机发送“B”，MCU 回发“D”，…。

6.5 实验二 串口通信及中断实验

串口通信简单方便使用，是最早普及的一种通信方式，也是嵌入式系统学习中简单常用的一种通信技术，可直接与 PC 机通信。其他嵌入式通信方式大多需要通过串口通信与 PC 机连接实现基本调试与现象观察。

1. 实验目的

本次实验内容较多，涉及 UART 通信基本编程、中断编程、组帧解帧，已经 PC 方的 C#串口通信编程方法。掌握了这些知识，为后续的深入学习，打好工具性基础知识。

(1) 以串行接收中断为例，掌握中断的基本编程步骤。

占的，此时称为“临界状态”，不能被打断的过程称为对“临界区”的访问。为防止在执行关键操作时被外部事件打断，一般通过关中断的方式使程序访问临界区，屏蔽外部事件的影响。执行完关键操作后退出临界区，打开中断，恢复对中断的响应能力。

-
- (2) 通过接收多个字节组成一帧，掌握串口通信组帧编程方法。
 - (3) 掌握 PC 机的 C#串口通信编程方法。

2. 实验准备

- (1) 软硬件工具：与实验一相同。
- (2) 运行并理解 “..\04-Software\CH06” 中几个程序。

3. 参考样例

(1) MCU 方样例程序：“..\04-Software\CH06\UART\UART-CH32V307-ISR” 中，以下 MCU 方样例程序均指这个程序。该程序使用 UART 构件，实现串口接收中断编程。MCU 收到一个字节后，进入串口接收中断处理程序，在该程序中，读出该字节，同时直接发送出去。可以利用 PC 机串口通信程序进行测试。

(2) PC 方样例程序：“..\05-Tool\C#2019 串口测试程序” 中。这是 PC 机方串口通信 C#源程序。无论是否学习过 C#语言，可以通过实例顺利理解其执行流程，基本掌握其编程方法，把它作为辅助工作，为学习 MCU 服务。“..\05-Tool” 还给出了 C#快速应用指南的下载方式。

4. 实验过程或要求

1) 验证性实验

验证 MCU 方样例程序，其主要功能是实现开发板上的小灯闪烁、通过 MCU 串口发送字符串、回发接收数据。

(1) 拷贝样例工程并重命名。拷贝 MCU 方样例程序工程到自己的工作文件夹，改为自己确定的工程名，建议尾端增加。

(2) 导入工程、编译、下载到 GEC 中。

(3) 观察实验现象。在开发环境下，使用“工具”→“串口工具”，可进行串口调试。也可利用“..\05-Tool\C#2019 串口测试程序”或其他通用串口调试工具进行测试。在此基础上，理解 main.c 程序和中断服务例程 isr.c。PC 机的 C#界面设计了发送文本框和接收字符型文本框、十进制型文本框、十六进制型文本框，理解接收、发送等程序功能。

(4) 修改程序。MCU 收到的一个字节后，将其减 3，再发送回去，理解其观察到的现象。

2) 设计性实验

(1) 参考 MCU 方样例程序，利用该程序框架实现：通过串口调试工具或“..\05-Tool\C#2019 串口测试程序”，PC 机发送字符‘1’或者‘0’来控制开发板上三色灯中的一个 LED 灯，MCU 的接收到字符‘1’时打开 LED 灯，接收到字符‘0’时关闭 LED 灯。

(2) 参考 MCU 方样例程序，利用该程序框架实现：通过串口调试工具或“..\05-Tool\C#2019 串口测试程序”，PC 机发送字符串“Open”或者“Close”来控制开发板上三色灯中的一个 LED 灯，MCU 的接收到字符串“Open”时打开 LED 灯，接收到字符串“Close”时关闭 LED 灯。

3) 进阶实验★

(1) 参考 MCU 方样例程序, 利用该程序框架实现: 修改编写 MCU 方和 C# 方程序, 利用组帧方法来完成串口任意长度数据的接收和发送。实现 C# 程序发送字符串 “Open” 或者 “Close” 来控制开发板上三色灯中的一个 LED 灯, MCU 的接收到字符串 “Open” 时打开 LED 灯, 接收到字符串 “Close” 时关闭 LED 灯。

提示: 组帧的双方可约定 “帧头+数据长度+有效数据+帧尾” 为数值帧的格式, 帧头和帧尾请自行设定。

(2) 利用上述实验中的组帧方法完成 C# 方和 MCU 方程序功能, C# 方程序实现鼠标单击相应按钮, 控制开发板上的三色灯完成 “红、绿、蓝、青、紫、黄、白、暗” 显示的控制。

5. 实验报告要求

(1) 描述进行串口通信及中断编程实验中遇到的三个以上问题, 给出出现的原因、解决方法及体会。

(2) 用适当文字, 描述接收中断方式下, MCU 方串口通信程序的执行流程, PC 方的 C# 串口通信程序的执行流程。

(3) 在实验报告中完成实践性问答题。

6. 实践性问答题

(1) 分别给出波特率 9600bps 和 115200bps 下发送一个字节需要多少时间?

(2) 有哪些简单的方法可以测试 MCU 串口的 TX 引脚发出了信号?

(3) 串口通信中用电平转换芯片 (RS-485 或 RS-232) 进行电平转换, 程序是否需要修改? 说明原因。

(4) 组帧中如何增加校验字段, 查找资料, 说一说有哪些常用校验方法。

(5) MCU 方的串口接收中断编程, 在 PC 机方的 C# 编程中是如何描述的?

本章小结

本章是全书的重点之一, 串行通信在嵌入式开发中具有特殊地位, 通过串行通信接口与 PC 机相连, 可以借助 PC 机屏幕进行嵌入式开发的调试。本章另一重要内容阐述中断编程的基本方法。至此, 1~6 章已经囊括了学习一个新 MCU 入门环节的完整要素。后续章节将在此规则与框架下学习各知识模块。

1. 关于串口通信的通用基础知识

MCU 的串口通信模块 UART, 在硬件上, 一般只需要三根线, 分别称为发送线 (TxD)、接收线 (RxID) 和地线 (GND), 在通信表现形式上, 属于单字节通信, 是嵌入式开发中重要的打桩调试手段。串行通信数据格式可简要表述为: 发送器通过发送一个 “0” 表示一个字节传输的开始, 随后一般是一个字节的 8 位数据, 最后, 发送器停止位 “1”, 表示一个字节传送结束。若继续发送下一字节, 则重新发送开始位, 开始一个新的字节传送。

若不发送新的字节，则维持“1”的状态，使发送数据线处于空闲。从开始位到停止位结束的时间间隔称为一字节帧。串行通信的速度用波特率表征，其含义是每秒内传送的位数，单位是：位/秒，记为 bps。

2. 关于UART构件的常用对外接口函数

首先应该学会使用 UART 构件进行串口通信的编程，正确理解与使用初始化（`uart_init`）、发送单个字节（`uart_send1`）、发送 N 个字节（`uart_sendN`）、发送字符串（`uart_send_string`）、接收单个字节（`uart_re1`）、使能串口接收中断（`uart_enable_re_int`）等函数。对于 UART 构件的制作，有一定难度，可以根据自己的学习情况确定掌握深度，基本要求是在了解寄存器基础上，理解利用直接地址操作的串口发送打通程序，后续进行构件制作。这里可以看出，使用构件与制作构件的难度差异，这是软件编程的社会分工的重要分界点，利用 GEC 概念，把这两个过程分割开来，做构件与用构件属于不同工作范畴。

3. 关于中断编程问题

任何一个计算机程序原则上可以理解为两条运行线，一条为无限循环线，另一条为中断线。要对一个中断进行编程，要求掌握以下几个环节：（1）中断源、中断 IRQ 号、中断向量号；（2）产生中断的条件；（3）中断初始化；（4）中断处理程序的存放位置及编写中断处理程序。读者可通过串口通信接收中断体会这个过程。

习 题

1. 利用 PC 机的 USB 口与 MCU 之间进行串行通信，为什么要进行电平转换？AHL-CH32V307 开发板中是如何进行这种电平转换的？

2. 设波特率为 115200，使用 NRZ 格式的 8 个数据位、没有校验位、1 个停止位，传输 6KB 的文件最少需要多少时间？

4. 简要给出 CH32V307 中断编程的基本知识要素，以串口通信的接收中断编程为例加以说明。

5. 查阅 UART 构件中对引脚复用的处理方法，说明这种方法的优缺点。

6. 按照 6.3.2 小节的方法，利用直接地址的方法给出开发板上 UART_Debug 串口的发送程序。

7. 简要阐述制作 UART 构件的基本过程。

8. 阐述一下为什么实际串行通信编程中必须对通信内容进行组帧和校验，给出组帧和校验的基本方法描述与实践。

第13章 应用案例

本章导读：本章作为扩展及讲座性内容，给出了嵌入式系统稳定性问题、外接传感器及执行部件的编程方法、实时操作系统的应用、嵌入式人工智能的应用等，这些内容来自实际应用开发的基本概括。目的是了解嵌入式系统实际应用的相关知识及有关领域，为实际应用提供借鉴。

13.1 嵌入式系统稳定性问题

学习到这里，读者基本上具备了进行嵌入式系统开发的软硬件基础，但是在实际开发嵌入式产品远不止于此。稳定性是嵌入式系统的生命线，而实验室中的嵌入式产品在调试、测试、安装之后，最终投放到实际应用，往往还会出现很多故障和不稳定的现象。由于嵌入式系统是一个综合了软件和硬件的复杂系统，因此单单依靠哪个方面都不能完全的解决其抗干扰问题，只有从嵌入式系统硬件、软件以及结构设计等方面进行全面的考虑，综合应用各种抗干扰技术来全面应对系统内外的各种干扰，才能有效提高其抗干扰性能。在这里，作者根据多年来的嵌入式产品开发经验，对实际项目中较常出现的稳定性问题做简要阐述，供读者在进一步学习中参考。

嵌入式系统的抗干扰设计主要包括硬件和软件两个方面。在硬件方面通过提高硬件的性能和功能，能有效的抑制干扰源，阻断干扰的传输信道，这种方法具有稳定、快捷等优点，但会使成本增加。而软件抗干扰设计采用各种软件方法，通过技术手段来增强系统的输入输出、数据采集、程序运行、数据安全等抗干扰能力，具有设计灵活、节省硬件资源、低成本、高系统效能等优点，且能够处理某些用硬件无法解决的干扰问题。

1. 保证CPU运行的稳定

CPU 指令由操作码和操作数两部分组成，取指令时先取操作码后取操作数。当程序计数器 PC 因干扰出错时，程序便会跑飞，引起程序混乱失控，严重时会导致程序陷入死循环或者误操作。为了避免这样的错误发生或者从错误中恢复，通常使用指令冗余、软件拦截技术、数据保护、计算机操作正常监控（看门狗）和定期自动复位系统等方法。

2. 保证通信的稳定

在嵌入式系统中，会使用各种各样的通信接口，以便与外界进行交互，因此，必须要保证通信的稳定。在设计通信接口的时候，通常从通信数据速度、通信距离等方面进行考虑，一般情况下，通信距离越短越稳定，通信速率越低越稳定。例如，对于 UART 接口，通常可选用 9600、38400、115200 等低速波特率来保证通信的稳定性，另外，对于板内通信，使用 TTL 电平即可，而板间通信通常采用 232 电平，有时为了传输距离更远，可以采用差分信号进行传输。

另外，通过为数据增加校验也是增强通信的稳定性的常用方法，甚至有些校验方法不仅具有检错功能，还具有纠错功能。常用的校验方法有奇偶校验、循环冗余校验法(CRC)、

海明码以及求和校验和异或校验等。

3. 保证物理信号输入的稳定

模拟量和开关量都是属于物理信号,它们在传输过程中很容易受到外界的干扰,雷电、可控硅、电机和高频时钟等都有可能成为其干扰源。在硬件上选用高抗干扰性能的元器件可有效的克服干扰,但这种方法通常面临着硬件开销和开发条件的限制。相比之下,在软件上则可使用的方法比较多,且开销低,容易实现较高的系统性能。

通常的做法是进行软件滤波,对于模拟量,主要的滤波方法有限幅滤波法、中位值滤波法、算术平均值法、滑动平均值法、防脉冲干扰平均值法、一阶滞后滤波法以及加权递推平均滤波法等;对于开关量滤波,主要的方法有同态滤波和基于统计计数的判定方法等。

4. 保证物理信号输出的稳定

系统的物理信号输出,通常是通过对相应寄存器的设置来实现的,由于寄存器数据也会因干扰而出错,所以使用合适的办法来保证输出的准确性和合理性也很有必要,主要方法有输出重置、滤波和柔和控制等。

在嵌入式系统中,输出类型的内存数据或输出 I/O 口寄存器也会因为电磁干扰而出错,输出重置是非常有效的办法。定期向输出系统重置参数,这样,即使输出状态被非法更改,也会在很短的时间里得到纠正。但是,使用输出重置需要注意的是,对于某些输出量,如 PWM,短时间内多次的设置会干扰其正常输出。通常采用的办法是,在重置前先判断目标值是否与现实值相同,只有在不相同的情况下才启动重置。有些嵌入式应用的输出,需要某种程度的柔和控制,可使用前面所介绍的滤波方法来实现。

总之,系统的稳定性关系到整个系统的成败,所以在实际产品的整个开发过程中都必须予以重视,并通过科学的方法进行解决,这样才能有效的避免不必要的错误的发生,提高产品的可靠性。

13.2 外接传感器及执行部件的编程方法

本节给出一些常见的嵌入式系统被控单元(传感器)的基本原理、电路接法和编程实践,对应硬件系统为 AHL-CH32V307-EXT,对应没有硬件系统的读者,可以通过阅读了解本节源程序,基本理解应用构件的制作方法及应用方法,达到举一反三之目的。

13.2.1 开关量输出类驱动构件

1. 彩灯

彩灯的控制电路与 RGB 芯片集成在一个 5050 封装的元器件中,构成了一个完整的外控像素点,每个像素点的三基色颜色可实现 256 级亮度显示。像素点内部包含了智能数字接口数据锁存信号整形放大驱动电路、高精度的内部振荡器和可编程定电流控制部分,有效保证了像素点光的颜色高度一致,数据协议采用单线归零码的通讯方式,通过发送具有特定占空比的高电平和低电平来控制彩灯的亮暗。

彩灯的电路原理图及实物图如图 13-1 所示。

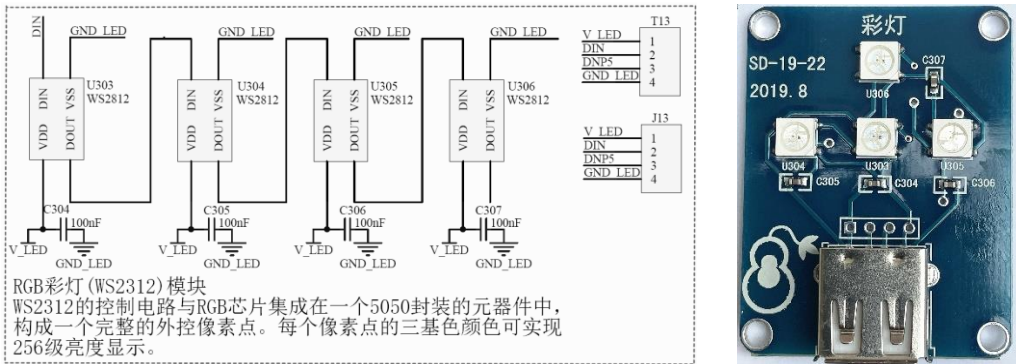


图13-1 彩灯电路原理图及实物图

VDD 是电源端，用于供电，DOUT 是数据输出端，用于控制数据信号输出，VSS 用于信号接地和电源接地，DIN 控制数据信号的输入。彩灯使用串行级联接口，能够通过一根信号线完成数据的接收与解码。

硬件连接参见工程中的文档说明，程序参考“..\04-Software\CH13\WJ01-ColorLight”工程。

2. 蜂鸣器

蜂鸣器输出端电平设置为高电平，蜂鸣器发出声响；输出端电平设置为低电平，蜂鸣器不发出声响或停止发出声响。蜂鸣器初始化默认是低电平，不发出声响。

蜂鸣器的电路原理图及实物图如图 13-2 所示。蜂鸣器通过 P_Beep 引脚来控制输出引脚的高低电平。当 P_Beep 对应的状态值为 1 即高电平时，Q401 导通，蜂鸣器发出声响；反之，当 P_Beep 对应的状态值为 0 即低电平时，Q401 截止，蜂鸣器不发出声响或停止发出声响。

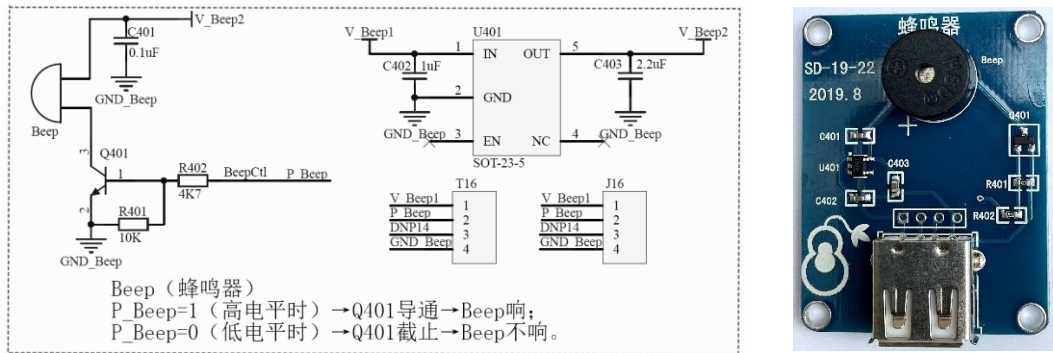


图13-2 蜂鸣器电路原理图及实物图

硬件连接参见工程中的文档说明，程序参考“..\04-Software\CH13\WJ02-BEEF”工程。

3. 马达

输出端电平设置为高电平，马达开始振动；输出端电平设置为低电平，马达不振动或

停止振动；马达初始化默认是低电平，不振动。

马达的电路原理图及实物图如图 13-3 所示。马达通过 AD_SHOCK 引脚来控制输出引脚的高低电平。当 AD_SHOCK 对应的状态值为 1 即高电平时，Q401 导通，马达开始振动；反之，当 AD_SHOCK 对应的状态值为 0 即低电平时，Q401 截止，马达不振动或停止振动。

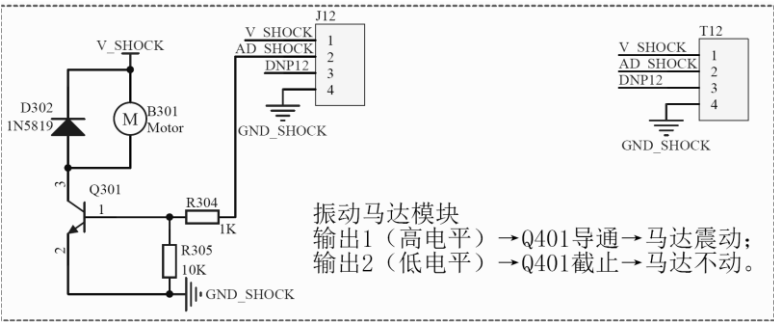


图13-3 马达电路原理图及实物图

硬件连接参见工程中的文档说明，程序参考“..\04-Software\CH13\WJ03-MOTOR”工程。

4. LED

在主函数中通过调用 TM1637_Display (a,a1,b,b1,c,c1,d,d1) 函数可以点亮数码管，其中数码管的数字显示可在调用函数时设置，a、b、c、d 为要显示的 4 位数字大小；而 a1、b1、c1、d1 为四位数字后面的小数点显示，值为 0 则不显示小数点，值为 1 则显示小数点。

数码管的电路原理图及实物图如图 13-4 所示。TM1637 驱动电路，通过 DIO 和 CLK 两个引脚实现对四位数码管的控制。DIO 引脚为数据输入输出，CLK 为时钟输入。数据输入的开始条件是 CLK 为高电平时，DIO 由高变低；结束条件是 CLK 为高电平时，DIO 由低电平变为高电平。

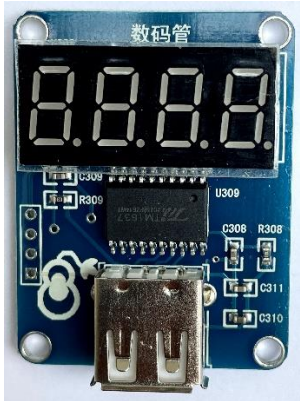
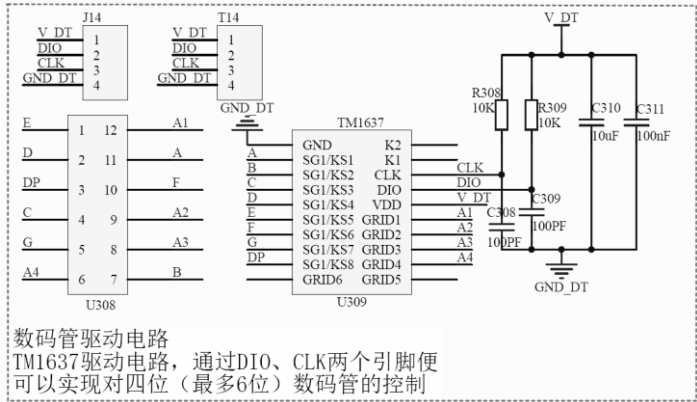


图13-4 LED电路原理图及实物图

硬件连接参见工程中的文档说明，程序参考“..\04-Software\CH13\WJ04- LED ”工程。

13.2.2 开关量输入类驱动构件

1. 红外寻迹传感器

当遮挡物体距离传感器红外发射管 2~2.5cm 之内，发射管发出的红外射线会被反射回来，红外接收管打开，模块输出端为高电平，指示灯亮；反之，若红外射线未被反射回来或反射回的强度不够大时，红外接收管处于关闭状态，模块输出端为低电平，指示灯不亮。

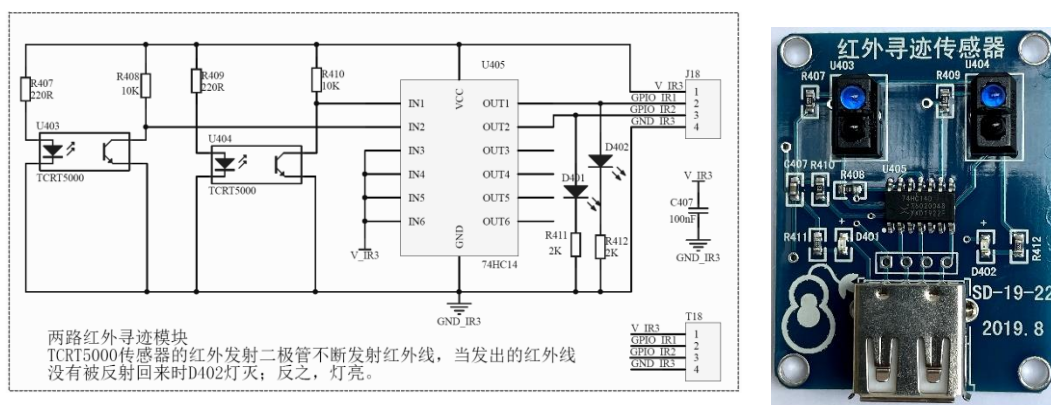


图13-5 红外寻迹传感器电路原理图及实物图

红外寻迹传感器的电路原理图及实物图如图 13-5 所示。其中，V_IR3 引脚为左右两侧红外发射器供电。GPIO_IR1 引脚为右侧的红外输出脚，并控制右侧的小灯亮暗；GPIO_IR2 引脚为左侧的红外输出脚，并控制左侧的小灯亮暗。红外寻迹传感器：用纸张靠近红外循迹传感器，红灯亮；撤掉纸张，红灯灭。

硬件连接参见工程中的文档说明，程序参考“..\04-Software\CH13\WJ05-Ray”工程。

2. 人体红外传感器

任何发热体都会产生红外线，辐射的红外线波长（一般用 μm ）跟物体温度有关，表面温度越高，辐射能量越强。人体都有恒定的体温，所以会发出特定波长 $10\mu\text{m}$ 左右的红外线，人体红外传感器通过检测人体释放的红外信号，判断一定范围内是否有人体活动。默认输出是低电平，当传感器检测到人体运动时，会触发高电平输出，小灯亮（有 3s 左右的延迟）。

人体红外的电路原理图及实物图如图 13-6 所示。其中，V_PIR1 用于供电。REF 为输出引脚。人体红外传感器：当用手靠近靠近人体红外传感器，红灯亮；远离，延迟 3 秒左右，红灯灭。

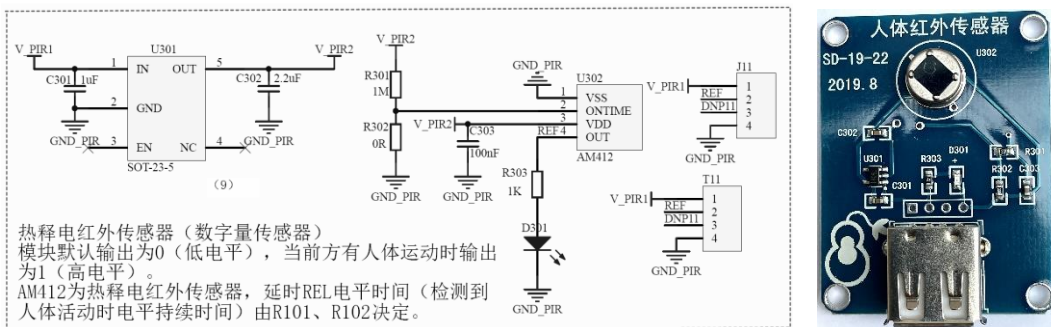


图13-6 人体红外传感器电路原理图及实物图

硬件连接参见工程中的文档说明，程序参考“..\04-Software\CH13\WJ06-RayHuman”工程。

3. 按钮

按钮的工作原理很简单，对于常开触头，在按钮未被按下前，触头是断开的，按下按钮后，常开触头被连通，电路也被接通；对于常闭触头，在按钮未被按下前，触头是闭合的，按下按钮后，触头被断开，电路也被分断。

Btn1、Btn2 初始化为 GPIO 输出，Btn3、Btn4 初始化为 GPIO 输入，并内部拉高（设置为高电平）。改变 Btn1、Btn2 的输出，通过扫描方式获取 Btn3、Btn4 的状态，判断按钮的闭合与断开。若将 Btn1 设置为低电平、Btn2 设置为高电平，则 Btn3 为低电平时，S301 闭合；Btn3 为高电平时，S301 断开。同样，Btn4 为低电平时，S302 闭合；Btn4 为高电平时，S302 断开。若将 Btn1 设置为高电平、Btn2 设置为低电平，则 Btn3 为低电平时，S303 闭合；Btn3 为高电平时，S303 断开。同样，Btn4 为低电平时，S304 闭合，Btn4 为高电平时，S304 断开。

按钮的电路原理图及实物图如图 13-7 所示。按钮：使用连接线接到按钮接口，另一端连接按钮。S301 对应 Btn1 被按下的提示信息，S302 对应 Btn2 被按下的提示信息，S303 对应 Btn3 被按下的提示信息，S304 对应 Btn4 被按下的提示信息。

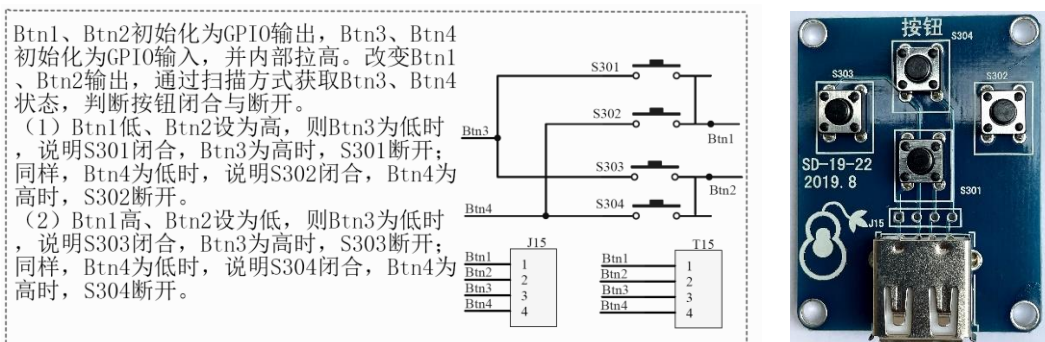


图13-7 按钮电路原理图及实物图

硬件连接参见工程中的文档说明，程序参考“..\04-Software\CH13\WJ07-Btn”工程。

13.2.3 声音与加速度传感器驱动构件

1. 声音传感器

声音传感器内置一个对声音敏感的电容式驻极体话筒（MIC）。声波使话筒内的驻极体薄膜振动，导致电容的变化，而产生与之对应变化的微小电压。这一电压随后被转化成 0-5V 的电压，经过 A/D 转换被数据采集器接受，并发送给计算机。

声音传感器的电路原理图及实物图如图 13-8 所示。对于一个驻极体的声音传感器，内部有一个振膜、垫片和极板组成的电容器。当膜片受到声音的压强时产生振动，从而改变膜片与极板的距离，此时会引起电容的变化。由于膜片上的充电电荷是不变的，所以必然会引起电压的变化，这样就完成了声信号转换成电信号。但由于这个信号非常微弱且内阻非常高，需要通过 U402 电路进行阻抗变化和放大，将放大后的电信号通过 ADSound 采集后被微机处理。

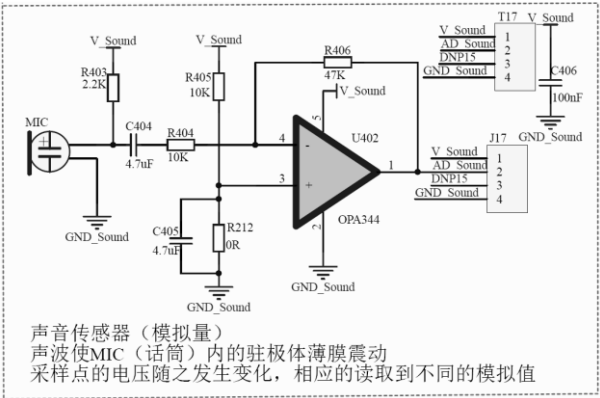


图13-8 声音传感器电路原理图及实物图

硬件连接参见工程中的文档说明，程序参考“..\04-Software\CH13\WJ08- ADSound”工程。

2. 加速度传感器

加速度传感器首先由前端感应器件感测加速度的大小（因为传感器内的差分电容会因为加速度而改变，从而传感器输出的幅度与加速度成正比），然后由感应电信号器件转为可识别的电信号，这个信号首先是模拟信号，然后通过 AD 转换器可以将模拟信号转换为数字信号，再通过串口读取数据。

加速度的电路原理图及实物图如图 13-9 所示。因为传感器内的差分电容会因为加速度而改变，从而传感器输出的幅度与加速度成正比，所以可以通过 SPI 或者 I2C 方法，获得输出的 16 进制数，从而显示出来。

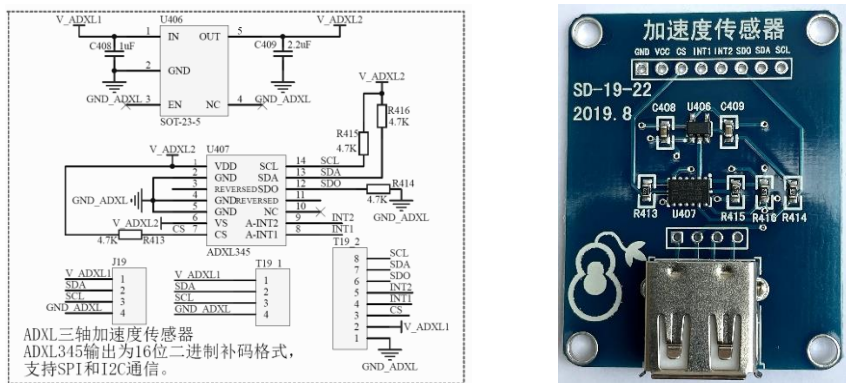


图13-9 加速度传感器电路原理图及实物图

硬件连接参见工程中的文档说明，程序参考“..\04-Software\CH13\WJ09-Acceleration”工程。

13.3 实时操作系统的简明实例

在开发嵌入式应用产品时，根据项目需求、主控芯片的资源状况、软件可移植性要求及开发人员技术背景等情况，可能选用一种实时操作系统作为嵌入式软件设计基础。特别是随着嵌入式人工智能与物联网的发展，对嵌入式软件的可移植性要求不断增强，实时操作系统的应用也将更加普及。

实时操作系统（Real Time Operation System，RTOS）是应用于嵌入式系统中的一种系统软件，在嵌入式产品开发中，可以根据硬件资源、软件复杂程度、可移植性需求、研发人员的知识结构等各个侧面综合考虑是否使用操作系统，若使用操作系统，选择哪种操作系统。

13.3.1 无操作系统与实时操作系统

无操作系统（No Operating System，NOS）的嵌入式系统中，在系统复位后，首先进行堆栈、中断向量、系统时钟、内存变量、部分硬件模块等初始化工作，然后进入“无限循环”，在这个无限循环中，CPU 一般根据一些全局变量的值决定执行各种功能程序（线程），这是**第一条运行路线**。若发生中断，将响应中断，执行中断服务例程（Interrupt Service Routines，ISR），这是**第二条运行路线**，执行完 ISR 后，返回中断处继续执行。从操作系统的调度功能角度理解，NOS 中的主程序，可以被简单地理解为一个 RTOS 内核，这个内核负责系统初始化和调度其它线程。

在基于 RTOS 的编程模式下，有两条线路，一条是线程线，编程时把一个较大工程分解成几个较小工程（被称之为线程或任务），有个调度者，负责这些线程的执行，另一条线路是中断线，与 NOS 情况一致，若发生中断，将响应中断，执行中断服务例程 ISR，然后中断处继续执行。可以进一步理解，RTOS 是一个标准内核，包括芯片初始化、设备驱

动及数据结构的格式化,应用层程序员可以不直接对硬件设备和资源进行操作,而是通过标准调用方法实现对硬件的操作,所有的线程由 RTOS 内核负责调度。也可以这样理解,RTOS 是一段嵌入在目标代码中的程序,系统复位后首先执行它,用户的其他应用程序(线程)都建立在 RTOS 之上。不仅如此,RTOS 将 CPU 时间、中断、I/O、定时器等资源都包装起来,留给用户一个标准的应用程序编程接口(Application Programming Interface, API),并根据各个线程的优先级,合理地在不同线程之间分配 CPU 时间。RTOS 的基本功能可以简单地概括为:RTOS 为每个线程建立一个可执行的环境,方便线程间的传递消息,在中断服务例程 ISR 与线程之间传递事件,区分线程执行的优先级,管理内存,维护时钟及中断系统,并协调多个线程对同一个 I/O 设备的调用。简而言之就是:线程管理与调度、线程间的通信与同步、存储管理、时间管理、中断处理等。

13.3.2 RTOS中的常用基本概念

在 RTOS 基础上编程,芯片启动过程先运行的一段程序代码,开辟好用户线程的运行环境,准备好对线程进行调度,这段程序代码就是 RTOS 的内核。RTOS 一般由内核与扩展部分组成,通常内核的最主要功能是线程调度,扩展部分的最主要功能是提供应用程序编程接口 API。

1. 调度

多线程系统中,RTOS 内核负责管理线程,或者说为每个线程分配 CPU 时间,并且负责线程间的通信。调度就是决定轮到哪个线程该运行了,它是内核最重要职责。每个线程根据其重要程度的不同,被赋予一定的优先级。不同的调度算法对 RTOS 的性能有较大影响,基于优先级的调度算法是 RTOS 常用的调度算法,核心思想是,总是让处于就绪态的、优先级最高的线程先运行。然而何时高优先级线程掌握 CPU 的使用权,由使用的内核类型确定,基于优先级的内核有不可抢占型和可抢占型两种类型。

2. 时钟节拍(时间嘀嗒)

时钟节拍(clock tick),有时中文也直接译为时钟嘀嗒,它是特定的周期性中断,通过定时器产生周期性的中断,以便内核判断是否有更高优先级的线程已进入就绪状态。

3. 线程的基本含义

线程是 RTOS 中最重要概念之一。在 RTOS 下,把一个复杂的嵌入式应用工程按一定规则分解成一个个功能清晰的小工程,然后设定各个小工程的运行规则,交给 RTOS 管理,这就是基于 RTOS 编程的基本思想。这一个个小工程被称之为“线程(Thread)”,RTOS 管理这些线程,被称之为“调度(Scheduling)”。

要给 RTOS 中的线程下一个准确而完整的定义并不十分容易,可以从不同角度理解线程。从线程调度角度理解,可以认为,RTOS 中的线程是一个功能清晰的小程序,是 RTOS 调度的基本单元;从 RTOS 的软件设计角度来理解,就是在软件设计时,需要根据具体应用,划分出独立的、相互作用的程序集合,这样的程序集合就被称之为线程,每个线程都

被赋予一定的优先级；从 CPU 角度理解，在单 CPU 下，某一时刻 CPU 只会处理（执行）一个线程，或说只有一个线程占用 CPU。RTOS 内核的关键功能就是以合理的方式为系统中的每个线程分配时间（即调度），使之得以运行。

实际上，根据特定的 RTOS，线程可能被称之为任务（Task），也可能使用其他名词，含义有可能稍有差异，但本质不变，也不必花过多精力，追究其精确语义，掌握线程设计方法、理解调度过程、提高编程鲁棒性、理解底层驱动原理、提高程序规范性、可移植性与可复用性、提高嵌入式系统的实际开发能力等才是学习 RTOS 的关键。要真正理解与应用线程进行基于 RTOS 的嵌入式软件开发，需要从线程的状态、结构、优先级、调度、同步等角度来认识，将在后续章节中详细阐述。

4. 线程的上下文及线程切换

线程的上下文（Context），即 CPU 内寄存器。当多线程内核决定运行另外的线程时，它保存正在运行线程的当前上下文，这些内容保存在随机存储器（Random Access Memory, RAM）中的线程当前状况保存区，也就是线程自己的堆栈之中。入栈工作完成以后，就把下一个将要运行线程的当前状况从其线程栈中重新装入 CPU 的寄存器，开始下一个线程的运行，这一过程叫做线程切换或上下文切换。

5. 线程间通信

线程间的通信是指线程间的信息交换，其作用是实现同步及数据传输。同步是指根据线程间的合作关系，协调不同线程间的执行顺序。线程间通信的方式主要有事件、消息队列、信号量、互斥量等。

13.3.3 线程的三要素、四种状态及三种基本形式

线程是完成一定功能的函数，但是并不是所有的函数都可以被称为线程。线程有自己的特有的要素以及形式。

1. 线程的三要素：线程函数、线程堆栈、线程描述符

从线程的存储结构上看，线程由三个部分组成：线程函数、线程堆栈、线程描述符，这就是线程的三要素。线程函数就是线程要完成具体功能的程序；每个线程拥有自己独立的线程堆栈空间，用于保存线程在调度时的上下文信息及线程内部使用的局部变量；线程描述符是关联了线程属性的程序控制块，记录线程的各个属性；下面做进一步阐述。

1) 线程函数

一个线程，对应一段函数代码，完成一定功能，可被称之为线程函数。从代码上看，线程函数与一般函数并无区别，被编译链接生成机器码之后，一般存储在 Flash 区。但是从线程自身角度来看，它认为 CPU 就是属于它自己的，并不知道还有其他线程存在。线程函数也不是用来被其他函数直接调用的，而是由 RTOS 内核调度运行。要使线程函数能够被 RTOS 内核调度运行，必须将线程函数进行“登记”，要给线程设定优先级、设置线程堆栈大小、给线程编号等等，不然有几个线程都要运行起来，RTOS 内核如何知道哪个

该先运行呢？由于任何时刻只能有一个线程在运行（处于激活态），当 RTOS 内核使一个线程运行时，之前的运行线程就会退出激活态。CPU 被处于激活态的线程所独占，从这个角度看，线程函数与无操作系统（NOS）中的“main”函数性质相近，一般被设计为“永久循环”，认为线程一直在执行，永远独占处理器。

2) 线程堆栈

线程堆栈是独立于线程函数之外的 RAM，按照“先进后出”策略组织的一段连续存储空间，是 RTOS 中线程概念的重要组成部分。在 RTOS 中被创建的每个线程都有自己私有的堆栈空间，在线程的运行过程中，堆栈用于保存线程程序运行过程中的局部变量、线程调用普通函数时会为线程保存返回地址等参数变量、保存线程的上下文等等。在多线程系统中，每个线程都认为 CPU 寄存器是自己的，一个线程正在运行时，当 RTOS 内核决定不让当前线程运行，而转去运行别的线程，就要把 CPU 的当前状态保存在属于该线程的线程堆栈中，当 RTOS 内核再次决定让其运行时，就从该线程的线程堆栈中恢复原来的 CPU 状态，就像未被暂停过一样。

3) 线程描述符

线程被创建时，系统会为每个线程创建一个唯一的线程描述符（Thread Descriptor, TD），它相当于线程在 RTOS 中的一个“身份证”，RTOS 就是通过这些“身份证”来管理线程和查询线程信息的。这个概念在不同操作系统名称不同，但含义相同，有的称为线程控制块（Thread Control Block, TCB），有的称为任务控制块（Task Control Block, TCB），有的称为进程控制块（Process Control Block, PCB）。线程函数只有配备了相应的线程描述符才能被 RTOS 调度，未被配备线程描述符的驻留在 Flash 区的线程函数代码就只是通常意义上的函数，是不会被 RTOS 内核调度的。

2. 线程的四种状态：终止态、阻塞态、就绪态和激活态

RTOS 中的线程一般有四种状态，分别为：终止态、阻塞态、就绪态和激活态。在任一时刻，线程被创建后所处的状态一定是四种状态之一。

(1) 终止态（Terminated, Inactive）：线程已经完成，或被删除，不再需要使用 CPU。

(2) 阻塞态（Blocked）：又可称为“挂起态”。线程未准备好，不能被激活，因为该线程需要等待一段时间或某些情况发生；当等待时间到或等待的情况发生时，该线程才变为就绪态，处于阻塞态的线程描述符存放于等待列表或延时列表中。

(3) 就绪态（Ready）：线程已经准备好可以被激活，但未进入激活态，因为其优先级等于或低于当前的激活线程，一旦获取 CPU 的使用权就可以进入激活态，处于就绪态的线程描述符存放于就绪列表中。

(4) 激活态（Active, Running）：又称“运行态”，该线程在运行中，线程拥有 CPU 使用权。如果一个激活态的线程变为阻塞态，则 RTOS 将执行切换操作，从就绪列表中选择优先级最高的线程进入激活态，如果有多个具有相同优先级的线程处于就绪态，则就绪列表中的首个线程先被激活。也就是说，每个就绪列表中相同优先级的线程是按执行先进先出（First in First out, FIFO）的策略进行调度的。

3. 线程的基本形式：单次执行、周期执行、资源驱动

线程函数一般分为两个部分：初始化部分和线程体部分。初始化部分实现对变量的定义、初始化以及设备的打开等等，线程体部分负责完成该线程的基本功能。线程一般结构如下：

```
void task ( uint_32 initial_data )
{
    //初始化部分
    //线程体部分
}
```

线程的基本形式主要有单次执行线程、周期执行线程以及事件驱动线程三种。

1) 单次执行线程

单次执行线程是指线程在创建完之后只会被执行一次，执行完成后就会被销毁或阻塞的线程，线程函数结构如下：

```
void task ( uint_32 initial_data )
{
    //初始化部分
    //线程体部分
    //线程函数销毁或阻塞
}
```

单次执行线程由三部分组成：线程函数初始化、线程函数执行以及线程函数销毁。初始化部分包括对变量的定义和赋值，打开需要使用的设备等等；第二部分线程函数的执行是该线程的基本功能实现；第三部分线程函数的销毁或阻塞，即调用线程销毁或者阻塞函数将自己从线程列表中删除。销毁与阻塞的区别在于销毁除了停止线程的运行，还将回收该线程所占用的所有资源，如堆栈空间等；而阻塞只是将线程描述符中的状态设置为阻塞而已。例如，定时复位重启线程就是一个典型的单次执行线程。

2) 周期执行线程

周期执行线程是指需要按照一定周期执行的线程，线程函数结构如下：

```
void task ( uint_32 initial_data )
{
    //初始化部分
    .....
    //线程体部分
    while(1)
    {
        //循环体部分
    }
}
```

```
}  
}
```

初始化部分同上面一样实现包括对变量的定义和赋值，打开需要使用的设备等等，与单次执行线程不一样的地方在于线程函数的执行是放在永久循环体中执行的，由于该线程需要按照一定周期执行，所以执行完该线程之后可能需要调用延时函数 `wait` 将自己放入延时列表中，等到延时的时间到了之后重新进入就绪态。该过程需要永久执行，所以线程函数执行和延时函数需要放在永久循环中。举例来说，在系统中，我们需要得到被监测水域的酸碱度和各种离子的浓度，但并不需要时时刻刻都在检测数据，因为这些物理量的变化比较缓慢，所以使用传感器采集数据时只需要每隔半个小时采集一次数据，之后调用 `wait` 函数延时半个小时，此时的物理量采集线程就是典型的周期执行的线程。

3) 资源驱动线程

除了上面介绍的两种线程类型之外，还有一种线程形式，那就是资源驱动线程，这里的资源主要指信号量、事件等线程通信与同步中的方法。这种类型的线程比较特殊，它是操作系统特有的线程类型，因为只有在操作系统下才导致资源的共享使用问题，同时也引出了操作系统中另一个主要的问题，那就是线程同步与通信。该线程与周期驱动线程的不同在于它的执行时间不是确定的，只有在它所要等待的资源可用时，它才会转入就绪态，否则就会被加入到等待该资源的等待列表中。资源驱动线程函数结构如下：

```
void task ( uint_32 initial_data )  
{  
    //初始化部分  
    .....  
    while(1)  
    {  
        //调用等待资源函数  
        //线程体部分  
    }  
}
```

初始化部分和线程体部分与之前两个类型的线程类似，主要区别就是在线程体执行之前会调用等待资源函数，以等待资源实现线程体部分的功能。仍以刚才的系统为例，数据处理是在物理量采集完成后才能进行的操作，所以在系统中使用一个信号量用于两个线程之间的同步，当物理量采集线程完成时就会释放这个信号量，而数据处理线程一直在等待这个信号量，当等待到这个信号量时，就可以进行下一步的操作。系统中的数据处理线程就是一个典型的资源驱动线程。

13.3.4 RTOS下编程实例

从应用开发角度，只要能够正确使用延时函数、事件、消息队列、信号量、互斥量等，就可以基本使用 RTOS 进行编程，本小节的目的是让读者通过实例，快速了解 RTOS 下编

程与 NOS 下编程的异同，快速了解了解延时函数、事件、消息队列、信号量、互斥量等的应用方法。这些实例基于上海睿赛德电子科技有限公司推出的国产实时操作系统 RT-Thread（Real Time-Thread），如表 13-1 所示。开发环境使用 AHL-GEC-IDE，硬件使用本书随附的 AHL-CH32V307。

表13-1 RTOS下编程实例列表

工程名	知识要素	程序功能
..\CH13\RTOS\RTOS01-Delay	延时函数	软件控制红、绿、蓝各灯每5秒、10秒、20秒状态变化，对外表现为三色灯的合成色，经过分析，即开始时为暗，依次变化为红、绿、黄（红+绿）、蓝、紫（红+蓝）、青（蓝+绿）、白（红+蓝+绿），周而复始
..\CH13\RTOS\RTOS02- Event	事件	当串口接收到一帧数据（帧头3A+四位数据+帧尾0D 0A）即可控制红灯的亮暗
..\CH13\RTOS\RTOS03-MessageQueue	消息队列	每当串口接收到一个字节，就将一条完整的信息放入到消息队列中，消息成功放入队列后，消息队列接收线程（run_messagerecv）会通过串口（波特率设置为115200）打印出消息，以及消息队列中消息的数量
..\CH13\RTOS\RTOS04-Semaphore	信号量	当线程申请、等待和释放信号量时，串口都会输出相应的提示
..\CH13\RTOS\RTOS05-Mutex	互斥量	说明如何通过互斥量来实现线程对资源的独占访问，RTOS01-Delay的样例工程，仍然实现红灯线程每5秒闪烁一次、绿灯线程每10秒闪烁一次和绿灯线程每20秒闪烁一次。在RTOS01-Delay 的样例工程中红灯线程、蓝灯线程和绿灯线程有时会同时亮的情况（出现混合颜色），而本工程通过单色灯互斥量使得每一时刻只有一个灯亮，不出现混合颜色情况

13.4 嵌入式人工智能的简明实例

目前人工智能的算法大多在性能较高的通用计算机上进行，但是，人工智能真正落地的产品却为种类繁多的嵌入式计算机系统。嵌入式人工智能就是指含有基本学习或推理算法的嵌入式智能产品。嵌入式物体认知系统就是嵌入式人工智能的应用实例之一。在此理念的基础上，苏州大学嵌入式人工智能与物联网试验室利用 MCU，设计了一套原理清晰、价格低廉、简单实用的基于图像识别的嵌入式物体认知系统（Embedded Object Recognition System, EORS），命名为 AHL-EORS，可以作为人工智能的快速入门系统。

13.4.1 EORS简介

1. 概述

基于图像识别的嵌入式物体认知系统是利用嵌入式计算机通过摄像头采集物体图像，利用图像识别相关算法进行训练、标记，训练完成后，可进行推理完成对图像的识别。AHL-EORS 主要目标用于嵌入式人工智能入门教学，试图把复杂问题简单化，利用最小的资源、最清晰的流程体现人工智能中“标记、训练、推理”的基本知识要素。同时，提供完整源码、编译及调试环境，期望达到“学习汉语拼音从啊（a）、喔（o）、鹅（e）开始，学习英语从 A、B、C 开始，学习嵌入式人工智能从物体认知系统开始”之目标。学生可通过本系统来获得人工智能的相关基础知识，并真实体会到人工智能的学习快乐，消除畏惧心理，使其敢于自行开发自己的人工智能系统。AHL-EORS 除了用于教学，本身亦可用于数字识别、数量计数等实际应用系统中。

2. 硬件清单

AHL-EORS 硬件清单如表 13-2 所示。

表13-2 AHL-EORS硬件清单

序号	名称	数量	功能描述
1	GEC主机	1	(1) 内含MCU（型号：CH32V307VCT6）、5V转3.3V电源等 (2) 2.8寸（240*320）彩色LCD (3) 接口底板：含光敏、热敏、磁阻等，外设接口UART、SPI、I2C、A/D、PWM等
2	TTL-USB串口线	1	两端标准USB口
3	摄像头	1	获取图像。LCD显示图像的默认设置为112×112（像素）大小

3. 硬件测试导引

产品出厂时已经将测试工程下载到 MCU 芯片中，可以进行 0~9 十个数字识别，测试步骤如下：

步骤一：通电。使用盒内双头一致 USB 线给设备供。电压为 5V，可选择计算机、充电宝等的 USB 口（注意供电要足）。

步骤二：测试。上电后，正常情况下，LCD 彩色屏幕会显示出图像，可识别盒子内“一页纸硬件测试方法”上的 0~9 数字，显示各自识别概率以及系统运行状态等参数。如图 13-10 所示。

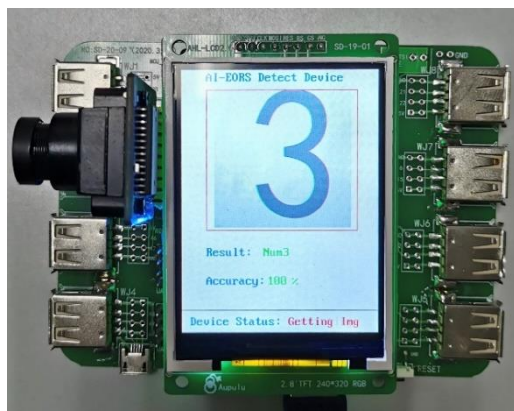


图13-10 AHL-EORS初始上电检测书中“3”正确现象

4. EORS的开发环境与电子资源

本系统的软件资源等都已经打包到电子资源文件夹内,软件下载方式如表 13-3 所示。

表13-3 AHL-EORS软件清单

序号	软件名	备注
1	金葫芦集成开发环境 (AHL-GEC-IDE)	(1) 下载地址: 百度搜索“苏州大学嵌入式学习社区”官网, 随后进入“金葫芦专区”→AHL-GEC-IDE (2) 操作系统: 使用Windows 10版本 (3) 下载完成后, 进入下载地址, 双击打开“AHL-GEC-IDE.exe”, 根据安装界面提示, 进行安装。推荐选择默认安装在D盘, 默认安装文件夹为: D:\AHL-GEC-IDE。
2	EORS电子资源	百度搜索“苏州大学嵌入式学习社区”官网, 随后进入“金葫芦专区”→AHL-EORS下载, 内含说明文档及源程序资源等

13.4.2 AHL-EORS的数据采集与训练过程

以识别字母“A、B、C、D”为例, 用户通过本样例熟悉并掌握完整的 AHL-EORS 中图像数据集采集与标记、模型的训练以及最终在主机上部署模型这三步过程。

1. 利用PC机软件进行图像采集与标记过程

在安装完环境之后, 将串口与 PC 相连, 然后打开电子资源“..\06-Tool\EORS_PC_DataReceive.exe”文件。该程序可以通过串口获取 MCU 上面的摄像头拍摄到的照片, 然后保存到本地计算机, 该过程也是人工智能中的“采集”过程。采集 1 张完整的图像数据后, 系统会显示采集到的这张图像, 如图 13-11 所示。



图13-11 显示数据界面

若显示的图像清晰且无其他干扰，满足采集要求，点击“确认保存”按钮，将本张图像添加到物体数据集中，否则点击“采集下一张”，丢弃本张数据。在采集完成所有的该图像数据集之后，将所有的 txt 文本文件按照类别合并，存放在对应的 txt 格式文件中。最后将文件名改为对应的类别名“A.txt”、“B.txt”、“C.txt”，“D.txt”。

2. 利用PC机软件进行训练过程

采集完成之后便要讲采集到的图片进行训练。点击打开资源文件夹内的“..\06-Tool\EORS_PC_TrainModel\EORS_ModelTrain\ModelTrain_v1.0.exe”可执行文件，打开过程较为缓慢，打开时长大于 10 秒，具体时间与个人计算机性能相关，请耐心等待不要多次点击。

训练模型的第一步是读取数据集，可以先使用我们已经提供的例程，该例程已预先存放在“..\05-Dataset\gray\ABCD”路径下，此时点击对应每个类别的数据集后的“选择文件”按钮，选择对应的数据集文件。在确定每个类别的训练集与测试集之后，我们再继续选择模型构件的保存位置。点击模型生成路径后的“选择路径”按钮，选择模型输出的文件夹。最后点击“开始训练”按钮，系统便开始训练模型。训练结束后，模型的测试准确率将会在提示窗口中显示，如图 13-12 所示。

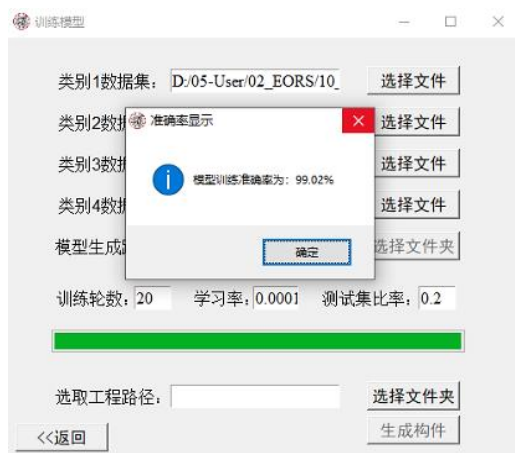


图13-12 训练过程的准确率显示信息

训练完成后，若对模型准确率不满意，可继续点击“开始训练”按钮，继续对模型训练，直到模型准确率趋于平稳或者准确率达到用户预期为止。需要重新训练或选取物体种类时，可点击左下角“返回”按钮，进入上一个界面。注意，返回后将丢失目前的模型和训练进度。

在得到用户满意的模型准确率之后，点击软件界面下方的“选择文件夹”按钮，选择指定的 AHL-EORS 推理工程，选择完毕后再点击“生成构件”按钮更新工程推理模型参数构件，即对本次训练得到的网络模型进行再部署。

13.4.3 在通用嵌入式计算机GEC上进行的推理过程

用户此时可以选择“..\04-Software\Predict_formwork”工程作为自己的样例工程，根据上一小节的所提到的模型参数构件的更新方法，将该工程变为具有识别四个字母功能的嵌入式工程，再再重新编译烧录电子资源，系统便认识了这四个字母。此时系统便“认识”了字母 B，如图 13-13 所示。

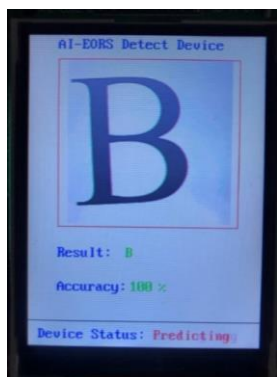


图13-13 检测到字母B

如需想要进一步学习嵌入式物体认知系统的具体实现原理，可以参考 EORS 的电子

资源文件夹内的快速指南。

13.5 沁恒MCU的其他嵌入式实践资源简介

本节以沁恒微电子的 CH573 芯片为例来介绍，包含了 CH573 的基本电路、产品特点以及基于此实现的 NB、CAT1 两种通讯方式，对应硬件系统为 AHL-GEC、AHL-CH573-NB-IoT 和 AHL-CH573-CAT1，若读者没有硬件，可通过阅读本章源程序来进一步理解如何实现 MCU 与通信模组之间的数据传输以及利用通用模组来发送数据等功能。

13.5.1 AHL-CH573

CH573 是沁恒公司推出的集成了 BLE 无线通讯的 32 位 RISC-V 内核微控制器。其片上集成低功耗蓝牙 BLE 通讯模块、全速 USB 主机和设备控制器以及收发器、SPI、4 个串口、ADC、触摸按键检测模块、RTC 等丰富的外设资源。接下来将简单介绍下 CH573 的最小硬件系统。

MCU 的硬件最小系统是指包括电源，晶振，复位，写入调试器接口等可使内部程序得以运行的、规范的、可复用的核心构件系统。使用一个芯片，必须完全理解其硬件最小系统。当 MCU 工作不正常时，在硬件层面，应该检查硬件最小系统中可能出错的元件。芯片要能工作，必须有电源与工作时钟；至于复位电路则提供不掉电情况下 MCU 重新启动的手段。随着 Flash 存储器制造技术的发展，大部分芯片提供了在板或在线系统（On System）的写入程序功能，即把空白芯片焊接到电路板上后，再通过写入器把程序下载到芯片中。这样，硬件最小系统应该把写入器的接口电路也包含在其中。基于这个思路，CH573 芯片的硬件最小系统包括电源电路、复位电路、与写入器相连的 SWD 接口电路及可选晶振电路。图 13-14 给出了 CH573 硬件最小系统原理图。

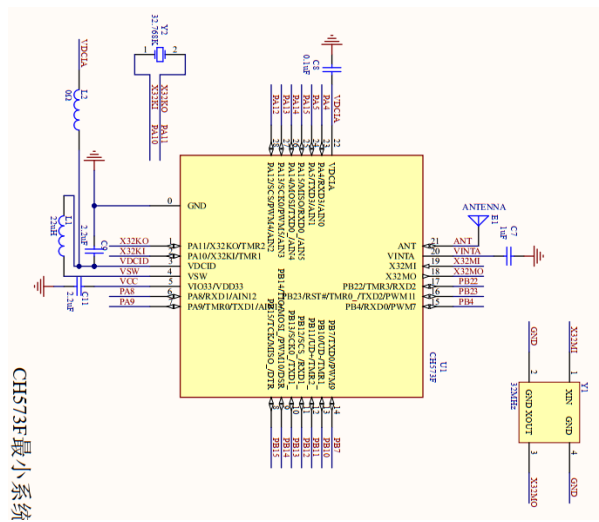


图13-14 CH573最小系统原理图

AHL-CH573 硬件清单如表 13-4 所示。

表13-4 AHL-CH573硬件清单			
序号	名称	数量	功能描述
1	GEC主机	1	(1) 内含MCU（型号：CH573）、5V转3.3V电源等 (2) 接口底板：3.7V电池接口，外设接口UART、SPI、A/D、PWM等
2	Type-C线	1	标准Type-C数据线，主机供电及串口通信使用

硬件连接参见工程中的文档说明，程序参考“..\04-Software\CH13\AHL-CH573”工程。

13.5.2 AHL-CH573-NB-IoT

窄带是物联网中常用的低速率通信方式,金葫芦 NB 开发套件(AHL-CH573-NB-IoT)是一套基于通用嵌入式计算机（General Embedded Computer，GEC）架构的 NB 快速开发套件，不仅可以配合物联网实践教学，还是一套较为完备的 NB 应用开发系统，可以实现面向物联网领域的 NB 应用快速开发。

AHL-CH573-NB-IoT 套件中的 NB 模组使用的是高新物联网的 ME3616 模组，在 NB-IoT 制式下，该模块可以提供最大 66 Kbps 上行速率和 34 Kbps 下行速率。该模块为极小尺寸 LCC 紧凑型封装模块，适用于可穿戴设备等对于模块尺寸有严格要求的应用领域。该套件的软件开发工具为苏州大学自主研发的集成开发环境 AHL-GEC-IDE，可实现串口下载与调试程序，串口 printf 函数跟踪调试等功能；提供了标准化终端软件开源模板，封装了 AT 指令，实现了构件级 NB 通信的收发；提供了云侦听 CS-Monitor、Web 网页、微信小程序开源模板，可以实现 30 分钟内通信收发直观体验，为“照葫芦画瓢”地快速应用开发提供基础。

金葫芦 CAT1 开发套件（AHL-CH573-NB-IoT）硬件清单如表 13-5 所示。

表13-5 AHL-CH573-NB-IoT硬件清单			
序号	名称	数量	功能描述
1	GEC主机	1	(1) 微控制器（CH573）：32位RISC-V处理器；片上集成低功耗蓝牙BLE通讯模块；支持3.3V和2.5V电源；提供4组独立UART。 (2) NB通信模组：ME3616。 (3) 外接天线：FPC贴片天线。 (4) 对外接口：GPIO、UART等。 (5)其他部件：5V转3.3V电源，红绿蓝三色灯，两个Type-C串口，复位等。
2	Type-C线	1	标准Type-C数据线，主机供电及串口通信使用

硬件套件如图 13-15 所示。

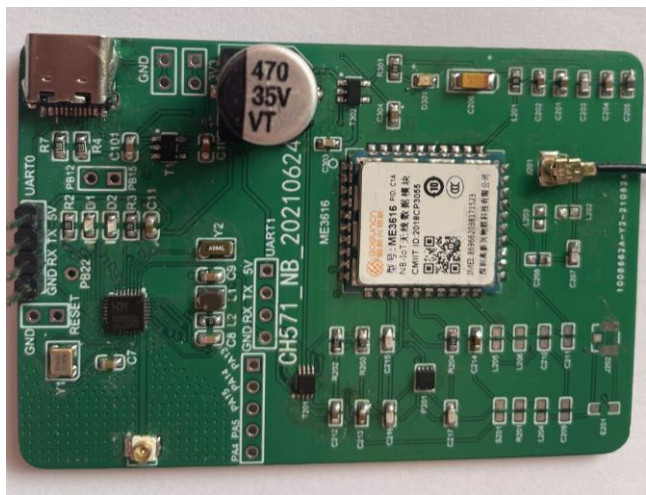


图13-15 AHL-CH573-NB-IoT硬件实物图

13.5.3 AHL-CH573-CAT1

CAT1 是 2020 年左右开始全面推广应用的面向广域网的中速率通信方式，最大上行速率为 5Mbps 左右、下行为 10Mbps 左右，主要目标是取代 GPRS 的物联网应用。金葫芦 CAT1 开发套件（AHL-CH573-CAT1）是一套基于通用嵌入式计算机（General Embedded Computer, GEC）架构的 CAT1 快速开发套件，不仅可以配合物联网实践教学，还是一套较为完备的 CAT1 应用开发系统，可以实现面向物联网领域的 CAT1 应用快速开发。

AHL-CH573-CAT1 套件中的 CAT1 模组使用广和通的 L610-CN-00-MiniPCIE-10（支持 LTE、GSM 双模通信）。该套件的硬件系统以 GEC 架构为基础，实现了将 MCU 与 CAT1 模组有机结合，形成完整的 CAT1 开发体系；软件工具为苏州大学自主研发的集成开发环境 AHL-GEC-IDE，可实现串口下载与调试程序，串口 printf 函数跟踪调试等功能；提供了标准化终端软件开源模板，封装了 AT 指令，实现了构件级 CAT1 通信的收发；提供了云侦听 CS-Monitor、Web 网页、微信小程序开源模板，可以实现 30 分钟内通信收发直观体验，为“照葫芦画瓢”地快速应用开发提供基础。

金葫芦 CAT1 开发套件（AHL-CH573-CAT1）硬件清单如表 13-6 所示。

表13-6 AHL-CH573-CAT1硬件清单

序号	名称	数量	功能描述
1	GEC主机	1	<p>(1) 微控制器（CH573）：32位RISC-V处理器；片上集成低功耗蓝牙BLE通讯模块；支持3.3V和2.5V电源；提供4组独立UART。</p> <p>(2) CAT1通信模组：L610-MINIPCIE。</p> <p>(3) 外接天线：FPC贴片天线。</p> <p>(4) 对外接口：GPIO、UART等。</p> <p>(5) 其他部件：5V转3.3V电源，红绿蓝三色灯，两个Type-C串口，复位等。</p>

2	Type-C线	1	标准Type-C数据线，主机供电及串口通信使用
---	---------	---	-------------------------

硬件套件如图 13-16 所示，电子资源下载地址如下：

<http://sumcu.suda.edu.cn/jhlCAT1kftjwAHLwCAT1wCH573w/list.htm>



图13-16 AHL-CH573-CAT1硬件示意图

参考文献

- [1] Free Software Foundation Inc. Using as The GNU Assembler [Z]. Version 2.11.90.[S.1: s.n.], 2012.
- [2] NATO Communications and Information Systems Agency. NATO Standard for Development of Reusable Software Components[S].[S.1. : s.n.], 1991.
- [3] Andrew Waterman, Yunsup Lee, David Patterson, etc. The RISC-V Instruction Set Manual Volume I: User-Level ISA[Z].[S.1. : s.n.], University of California, Berkeley, 2016.
- [4] Andrew Waterman¹, Krste Asanovic. The RISC-V Instruction Set Manual Volume II: Privileged Architecture[Z].[S.1. : s.n.], University of California, Berkeley, 2019.
- [5] David Patterson, Andrew Waterman. 勾凌睿, 黄成, 刘志刚译. RISC-V 手册: 一本开源指令集的指南[Z].[S.1. : s.n.], 2021.
- [6] 胡振波. RISC-V 架构与嵌入式开发快速入门[M]. 人民邮电出版社, 2019.
- [7] 胡振波. 手把手教你设计 CPU: RISC-V 处理器[M]. 人民邮电出版社, 2018.
- [9] 沁恒微电子. CH32V20x_30x 数据手册[Z].[S.1. : s.n.], 2021.
- [10] 沁恒微电子. CH32FV2x_V3x 系列应用手册[Z].[S.1. : s.n.], 2021.
- [11] 王宜怀, 李跃华, 徐文彬, 施连敏. 嵌入式技术基础与实践(第 6 版)——基于 STM32L431 微控制器[M], 清华大学出版社, 2021.
- [12] 王宜怀, 史洪玮, 孙锦中, 罗喜召. 嵌入式实时操作系统——基于 RT-Thread 的 EAI&IoT 系统开发[M], 机械工业出版社, 2021.
- [13] Gary R. Wright, W. Richard Stevens 著, 陆雪莹、蒋慧等译. TCP/IP 详解卷 1[M], 机械工业出版社, 2007.