第3章　第一个样例程序及CodeWarrior工程组织

本章详细阐述第一个C语言程序和汇编语言程序的结构，完成第一个S08工程的入门。主要内容有：①给出通用I/O接口基本概念及连接方法；②利用GPIO模块编程控制发光二极管作为入门例子，给出S08(C语言与汇编语言)工程组织、框架，阐述各个文件的功能，**主要目的是使读者理解程序框架和工作过程**；③在此基础上进行实际环境的编译、链接生成机器码、将机器码下载芯片内部Flash中，进行调试、运行。重点是透彻理解第一个工程的执行过程。至于C++语言的工程组织与C语言的相仿，在素材之中列出了例程供读者参考。

3.1　通用I/O接口基本概念及连接方法

1. I/O接口的概念

I/O接口，即输入输出接口，是微控制器同外界进行交互的重要通道。这里的接口英文是port，也可以翻译为“端口”，另一个英文单词是interface，也翻译为接口。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，不必深究它们之间的区别。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，接口千变万化，种类繁多，有显而易见的人机交互接口，如操纵杆、键盘、显示器；也有无人介入的接口，如网络接口、机器设备接口。

2. 通用I/O

所谓通用I/O，也记为GPIO(General Purpose I/O)，即基本的输入输出，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”。作为通用输入引脚，MCU内部程序可以通过端口寄存器读取该引脚，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，MCU内部程序通过端口寄存器向该引脚输出“1”(高电平)或“0”(低电平)，即开关量输出。大多数通用I/O引脚可以通过编程来设定工作方式为输入或输出，称为双向通用I/O。

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1kΩ～10kΩ之间，

其阻值大小与静态电流及系统功耗相关。

图 3-1 给出了一个 MCU 的输入引脚的三种外部连接方式,假设 MCU 内部没有上拉或下拉电阻,图中的引脚 I_3 上的开关 K_3 采用悬空方式连接就不合适,因为 K_3 断开时,引脚 I_3 的电平不确定。在图 3-1 中,$R_1 \gg R_2$,$R_3 \ll R_4$,各电阻的典型取值为 $R_1=20k\Omega$,$R_2=1k\Omega$,$R_3=10k\Omega$,$R_4=200k\Omega$。

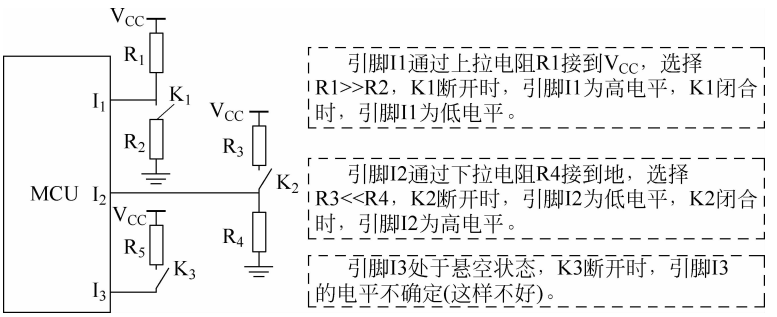

图 3-1　I/O 接口输入电路

4. 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 3-2 所示。

一种接法是 O_1 引脚直接驱动发光二极管 LED,当 O_1 引脚输出高电平时,LED 不亮;当 O_1 引脚输出低电平时,LED 点亮。这种接法的驱动电流一般在 2mA～10mA。

另一种接法是 O_2 引脚通过一个 NPN 三极管驱动蜂鸣器,当 O_2 引脚输出高电平时,蜂鸣器响;O_2 引脚输出低电平时,蜂鸣器不响。这种接法的驱动电流可达 100mA 左右,而 O_2 引脚控制电流可以在几个 mA。

若负载需要更大的驱动电流,就必须使用另外的驱动电路,但对 MCU 编程来说,没有任何影响。

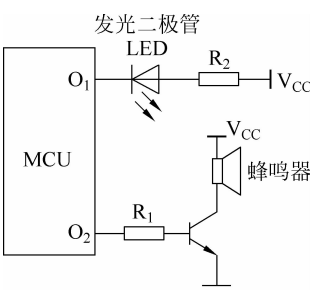

图 3-2　I/O 接口输出电路

3.2　AW60 的 GPIO

3.2.1　AW60 GPIO 编程的基本原理

AW60 的大部分引脚具有多重功能,可以通过编程设定使用其中一种功能。GPIO 作为基本功能,AW60 有 7 个 GPIO 接口。图 3-3 给出了 AW60 的 7 个 GPIO 接口的引脚分布情况。

每个 GPIO 接口的名称由一位英文字母组成,分别是 A、B、C、D、E、F、G。

GPIO 的基本编程方法:

(1) 通过“数据方向寄存器”设置相应引脚为输入或输出。

(2) 若是输出引脚,则设置“端口数据寄存器”引脚输出高电平或低电平。

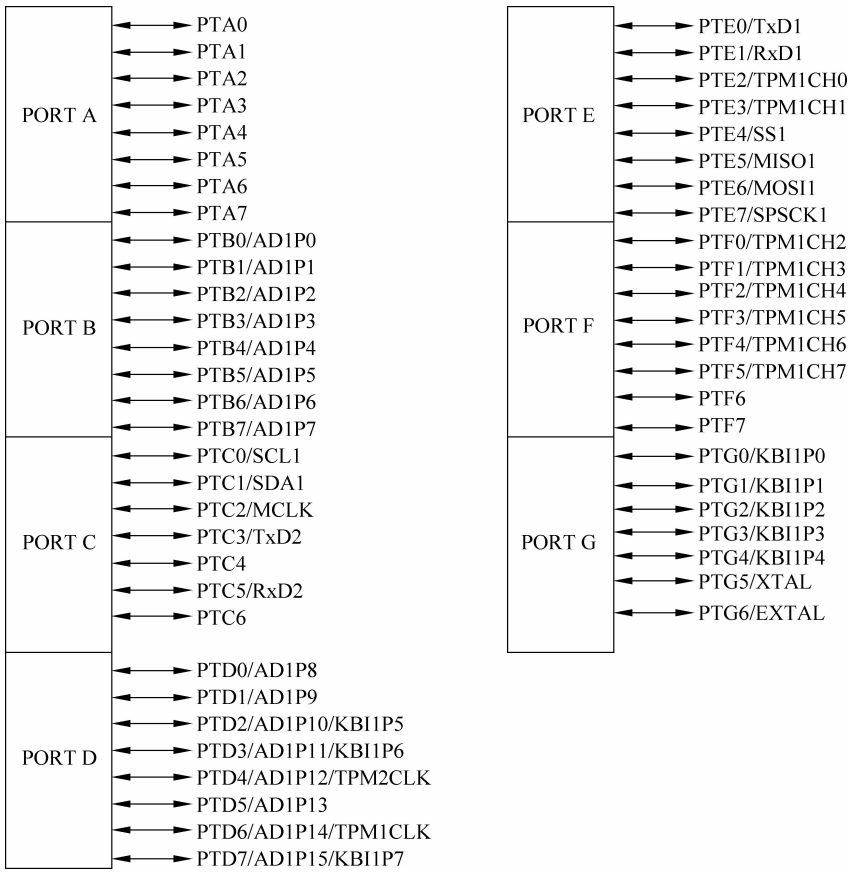

图 3-3　GPIO 模块框图

(3) 若是输入引脚,则通过“端口数据寄存器”获得引脚的状态。

3.2.2　GPIO 模块寄存器与 GPIO 编程的基本方法

GPIO 模块的每个接口最多对应 8 个 GPIO 引脚,但各个 GPIO 接口的编程寄存器均为 8 位,只是没有对应引脚的位无效。

GPIO 模块寄存器的命名有一定的规范,所有寄存器都在 AW60 芯片寄存器及相关位定义头文件 AW60.h 中定义。其中“端口数据寄存器”是 PT+该端口的名称+D。“端口输出方向寄存器”是 PT+该端口的名称+DD。所有寄存器的位编号从 0 开始,且最低位编号为 0。例如,一个 8 位寄存器的最低位编号为 0,最高位编号为 7。例如,A 端口输出数据寄存器记为 PTAD、端口输出方向寄存器记为 PTADD。

1. 操作 GPIO 的基本寄存器

1) 端口数据方向寄存器(Port Data Direction Register,DDR)

这些位分别控制着端口引脚是输入还是输出,若为 0,则引脚为输入;若为 1,则引脚为输出。复位时为 0x00。

2) 端口数据寄存器 PORT(Port Data Register)

若引脚被配置为输出,PORTn 寄存器中每一位数据决定了对应引脚的输出电平。复

位时 PORTn 寄存器的所有被使用的位为 1。

2. GPIO 的基本编程方法举例

在对有关基本寄存器功能有所了解后,下面将以对 D 接口第 3 引脚的 GPIO 功能演示代码为例让读者对上述寄存器有更深的了解,完整工程请参阅素材中本章的例程。

首先,为了程序通用性,对要设置的引脚进行宏定义,如需对第 2 引脚操作,只需将下面的"3"修改为"2"即可。现在,设要使用的引脚名叫"RUNpin",接在 TD 接口的第 3 引脚上,则需使用如下宏定义:

```
#define RUN_PORT     PORT_D          //灯使用的端口
#define RUNpin       3               //用 RUNpin 代替"3"
```

注意:PORT_D 是一个代表 D 端口寄存器的 8 位地址指针,在 GPIO.h 文件中定义。这样,程序员就可以用普通 C 语言对该寄存器进行操作了。

当需要对寄存器的具体位进行设置,而不改变该寄存器中其他位的值时,通常会使用到位操作,例如:

```
RUN_PORT &= 0b11110111;              //表示将 RUN_PORT 的第 3 位设置为 0
RUN_PORT |= 0b00001000;              //表示将 RUN_PORT 的第 3 位设置为 1
```

为了程序通用性,以上两行代码需要写成:

```
RUN_PORT &= ~(0x01<< RUNpin);        //将 RUN_PORT 的第 3 位设置为 0
RUN_PORT |= 0x01<< RUNpin;           //将 RUN_PORT 的第 3 位设置为 1
```

这样做的好处是,当需要改变所使用的引脚时,只需要改变 RUN_PORT 和 RUNpin 的宏定义,这也是软件工程的基本原则的体现。另外,若要对连续几位进行操作,则可以用以下代码来实现:

```
RUN_PORT &= ~(0x03<< RUNpin);        //将 RUN_PORT 的第 3、4 位都设置为 0
RUN_PORT |=  0x03<< RUNpin;          //将 RUN_PORT 的第 3、4 位都设置为 1
```

以上的例子说明了怎样设置具体寄存器的单独 1 位或连续 2 位的位值,若要对连续的 3 位或更多的位进行设置,只需要更改 0x03 为合适的值即可(3 位对应 0x07)。

3.3 开发套件 CodeWarrior 开发环境与 S08/S12/ColdFire 写入器

嵌入式软件开发有别于桌面软件开发的一个显著特点,是它一般需要一个交叉编译和调试环境,即编辑和编译软件在通常的 PC 上进行,而编译好的软件需要通过写入工具下载到目标机上执行,如 AW60 的目标机上。由于主机和目标机处理器的体系结构彼此不同,从而增加了嵌入式软件开发的难度。所以选择一些好的开发套件有助于对目标机的学习与开发。

下面将介绍 Freescale 公司的 CodeWarrior for S08 V6.2 集成开发环境(简称 CW 环境),素材中提供从 Freescale 公司下载的免费使用的 CodeWarrior for S08 V6.2,有文件限

制，可以用于教学。实验可以在苏州大学的“飞思卡尔嵌入式系统开发平台”上进行，也可以在类似的评估系统上进行。写入器使用苏州大学 S08/S12/ColdFire 三合一写入器。

3.3.1　CodeWarrior 开发环境简介与基本使用方法

1. CodeWarrior 环境功能和特点

CodeWarrior 开发环境（简称 CW 环境）是 Freescale 公司研发的面向 Freescale MCU 与 DSP 嵌入式应用开发的商业软件工具，其功能强大，是 Freescale 向用户推荐的产品。

CodeWarrior 分为 3 个版本：特别版（Special Edition）、标准版和专业版。在其环境下可编制并调试 AW60 MCU 的汇编语言、C 语言和 C++语言程序。其中特别版是免费的，用于教学目的，对生成的代码量有一定限制，C 语言代码不得超过 12KB，对工程包含的文件数目也限制在 30 个以内。标准版和专业版没有这种限制。3 个版本的区别在于用户所获取的授权文件（License）不同，特别版的授权文件随安装软件附带，不需要特殊申请，标准版和专业版的授权文件需要付费。CodeWarrior 特别版、标准版和专业版的定义随所支持的微处理器的不同而不同，如 CodeWarrior for S08 V6.2、CodeWarrior for HC12 V4.6、CodeWarrior for ColdFire V6.3 等。

CW 环境包括以下几个功能模块：编辑器、源码浏览器、搜索引擎、构造系统、调试器、工程管理器。编辑器、编译器、连接器和调试器对应开发过程的四个主要阶段，其他模块用于支持代码浏览和构造控制，工程管理器控制整个过程。该集成环境是一个多线程应用，能在内存中保存状态信息、符号表和对象代码，从而提高操作速度；能跟踪源码变化，进行自动编译和连接。

2. CW 环境安装与设置

CW 环境安装没有什么特别之处，在 Windows 操作系统上，只要按照安装向导单击鼠标就可以自动完成。

需要说明的是，安装完毕以后要上网注册以申请使用许可（License Key）。无论是下载的软件还是申请到的免费素材，安装后都要通过因特网注册申请使用许可。这里可通过登录其网站，单击“Request a Key”实现。由于这一注册过程是在网上自动实现的，故只要网络通畅，这个往返过程在数分钟之内即可完成。申请后会通过 E-mail 得到一个 License.dat 文件。将该文件复制到相应目录下即可，例如 C:\Program Files\Freescale\CodeWarrior for S08 V6.2\。对于免费的特别版本，安装好后用 License.dat 覆盖安装目录下的 License.dat。

CW 环境的运行界面如图 3-4 所示。

由于 CodeWarrior IDE 安装后的默认字体是 Courier New，对中文的支持不完善，因此建议修改字体。选择 Edit －＞ Preferences 命令，弹出 IDE Preferences 对话框。在 Font&Tabs 选项卡中设置字体为 Fixedsys，Script 为 CHINESE_GB2312，由于 Tab 在不同文本编辑器解释不同，建议在 Tab Inserts Spaces 前打勾，使 Tab 键插入的是多个空格。

素材中有相关简明使用方法的文档，本章随后的内容也有引导读者如何进行编辑、编译、运行、调试等阐述。

图 3-4 CW 环境运行界面

3.3.2 S08/S12/ColdFire 三合一写入器

开发人员可以通过 S08/S12/ColdFire 三合一写入器对目标板中的 Flash 存储器进行擦除、写入等操作。将机器码下载到 Flash 存储器后,可以进行程序的运行、调试。图 3-5 给出了写入器的实物图。使用该写入器时,一端连接 PC 的 USB 口,另一端连接目标板 BDM 接口。

图 3-5 S08/S12/ColdFire 三合一写入器实物图

素材中含有【(S08-S12-ColdFire BDM)(写入器)安装与使用】文件夹，如图 3-6 所示。该文件夹中提供了写入器安装与使用所需要的软件和使用说明。文件夹中还给出了苏州大学开发的独立写入软件，可直接打开机器码文件下载到目标芯片中，供生产过程中使用。

图 3-6　S08/S12/ColdFire 三合一写入器文件目录

3.3.3　SD-AW60EVB 硬件评估板

SD-AW60EVB 硬件评估板如图 3-7 所示。该硬件评估板使用 64 引脚的 AW60 芯片，带 RS232 串口、一个复位按钮、一个 BDM 写入接口、引出所有 I/O 接口，并带有与苏州大学飞思卡尔嵌入式系统研发中心的 SD-扩展板-D 型的对接口。扩展板外接 12V 直流电源，转成 5V 给 AW60 核心板供电。

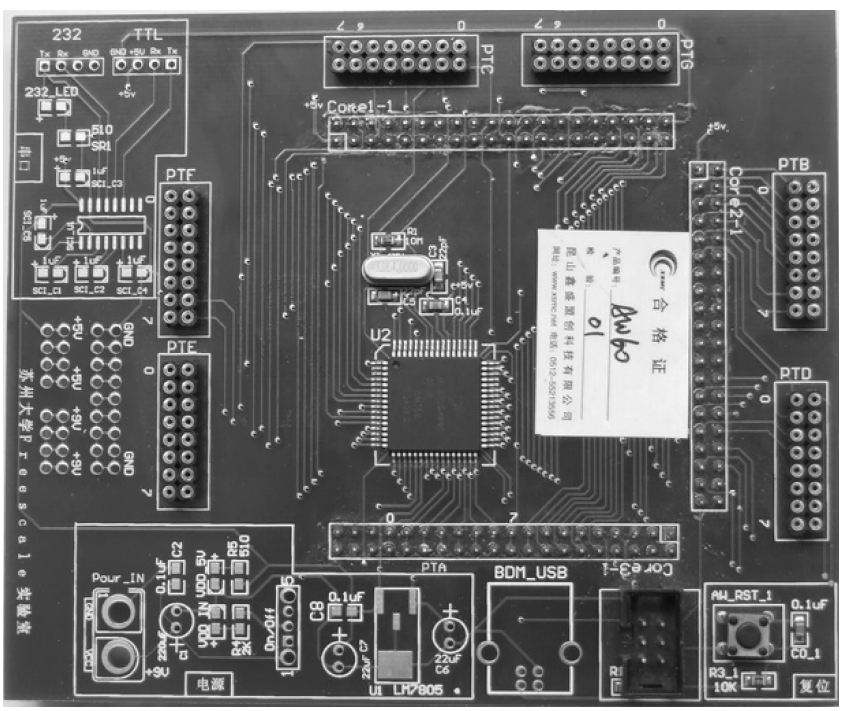

图 3-7　SD-AW60EVB 硬件评估板

3.4　CW 环境 C 语言工程文件的组织

嵌入式系统工程往往包含很多文件，如程序文件、头文件、与编译调试相关的信息文件、工程说明文件以及工程目标代码文件等。工程文件的合理组织对一个嵌入式系统工程犹为重要，它不但会提高项目的开发效率，同时也降低项目的维护难度。

下面将从逻辑结构和物理结构两个层次为读者做一个简要的剖析。逻辑结构是相对于

工程而言的,而物理结构却说明其在存储磁盘介质上的实际分布情况。这部分内容对于初学者而言有一定难度,但一定要有耐心,一时不完全理解也没关系,先建立一个初步印象,待学习过第 5、6 章后再回过头来体会,就会发现良好且规范的工程组织对程序的可复用、可移植,减少重复编码、增强程序稳定性十分有益。

3.4.1 工程文件的逻辑组织结构

嵌入式系统工程的文件逻辑组织方法以硬件构件为核心来展开,有关硬件构件的基本概念将在第 4 章阐述。

一个硬件构件的底层驱动软件由头文件和程序文件组成,在不引起二义性的前提下,也可直接称为硬件构件。**以硬件构件的方式来组织文件,会使工程结构清晰,调试定位方便,后期维护容易,这也是嵌入式系统软件工程的基本思想**。

下面以 3.5 节所述的控制小灯闪烁工程为例,介绍基于 CW 环境的嵌入式**工程文件逻辑组织**方法。图 3-8 给出了该工程相关源文件的树型结构,可分为 5 部分:

- 头文件夹<H>
- C 源程序文件夹<C>
- 库文件文件夹<Libs>
- 工程设置文件夹<Project Settings>
- “工程说明.txt”文件

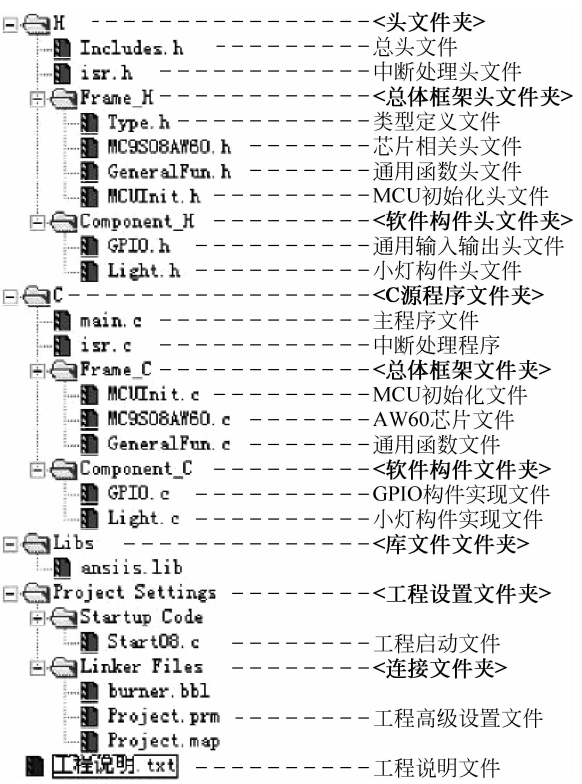

图 3-8 小灯闪烁工程相关源文件的树型(逻辑)结构

其中，“头文件夹＜H＞”又包含“总体框架头文件夹＜Frame_H＞”与“软件构件头文件＜Component_H＞”；相应地，“C 源程序文件夹＜C＞”包含“总体框架文件夹＜Frame_C＞”与“软件构件文件夹＜Component_C＞”。需要说明的是，在 CodeWarrior 工程列表中看到的这些文件夹物理存储器里是不存在的，它只负责逻辑区分，便于工程管理，而不具有任何实际意义。

请特别注意 main.c 和 isr.c 这两个文件，从源程序角度来看，一般嵌入式软件的执行流程如下：系统启动并初始化后，程序转到 main.c 文件中第一个可执行语句执行，随后到 main.c 文件中定义的主循环执行（理论上对于嵌入式系统而言，主程序一定是一个死循环）；当遇到中断请求时，转而执行 isr.c 中定义的相应中断处理程序；中断处理结束后则返回主程序继续执行。由于 main.c 和 isr.c 文件反映了软件系统的整体执行流程，故而在工程文件组织时，将它与其余 C 语言源程序文件分开管理，同样，与这两个文件对应的 Includes.h 和 isr.h 文件也被单独拿出，放在头文件的根目录下。

此外，与总体框架程序相关的头文件和源文件分别放在 Frame_H 和 Frame_C 子文件夹中，以归类使其便于管理。Frame_H 里包含了 Type.h、MC9S08AW60.h、GeneralFun.h、MCUInit.h 四个头文件。Type.h 用于类型别名定义，它将 C 语言中用于变量类型定义的关键字简化定义成比较简短的形式，这样，开发者在定义变量时，可以不必敲入冗长的变量定义关键字，同时，这也为不同编译体系间的代码复用和移植提供了方便。MC9S08AW60.h 是 MC9S08AW60 芯片寄存器及相关位定义头文件，它可被视为芯片的接口文件，没有这个文件，就不可能对该芯片进行任何操作。GeneralFun.h 与 GeneralFun.c 对应，存放与芯片无关的常用且基本的软件功能性子函数，如延时子函数等。MCUInit.h 与 Frame_C 子文件夹中的 MCUInit.c 对应，它定义了系统初始化时的基本参数，如系统时钟等，而 MCUInit.c 文件则包含实际初始化代码。

以上所说，通常是系统正常运行所必需的，它们只是实现了“最小系统”。若要系统能做实际的事情，还必须添加相关功能代码。按照软件构件化原则，这些代码被安置于 Component_H 与 Component_C 子文件夹中。每个功能实体，或称为“构件”，都对应一个.c 文件和一个.h 文件。例如，用于指示灯控制的 Light 构件，就对应了 Light.c 和 Light.h，它们分别放置于 Component_C 和 Component_H 子文件夹中。构件主要是按照功能进行划分的，除了这里定义的 Light 构件和 GPIO 构件，以后的章节还会出现“串行通信”、“键盘”、“LED”、“液晶”等构件，它们都是分别被放置在这两个文件夹里。

另外，嵌入式系统工程框架中还必须包括部分与编译器相关的文件，如连接文件，用于告诉编译器，代码是如何安放在具体的地址空间的。了解该文件的格式，有助于全面理解嵌入式系统的运作。另外，一个好的工程说明文件，对于一个工程的维护而言是相当重要的。

3.4.2　工程文件的物理组织结构

上一节从逻辑层面上阐述了一些重要的内容，这一节将带着读者去看看它们是如何分布在计算机硬盘中的，即在硬盘中的目录分配情况。下面以小灯闪烁程序的文件架构作说明（见图 3-9）。

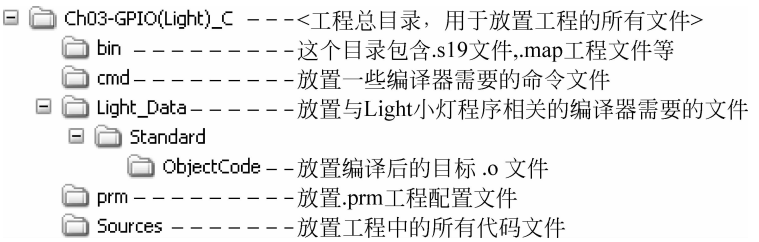

图 3-9 小灯闪烁工程相关源文件的树型(物理)结构

<bin>目录下的.s19文件为工程最后生成文件,可以通过写入工具写入MCU芯片中运行,类似于PC的.exe文件。有的厂家可能直接使用二进制存储,但那样不好直接使用一般文本编辑器阅读。而飞思卡尔公司则使用其定义的文本方式文件(.s19)存储机器码,优点是可直接使用通用的文件编辑器打开查看,缺点是下载到芯片之前,必须转为二进制机器码,当然这是下载工具的事情,一般读者不必关心。关于.s19文件的含义与组成方式见3.4.5节。而bin目录下的.map文件为代码在MCU存储器中的分配提供了证据,建议读者在完成第5章学习后,结合实际例子将.map文件与.s19对比分析一下,.map文件告诉源代码被编译链接后的机器码到底被下载到MCU内存储器中的什么地方,在高级调试时,可能需要用到这些知识。

<cmd>目录下为命令文件,用于配置编译器完成某些特别的功能,例如软件复位,芯片解密等。

<Light_Data/Standard/ObjectCode>目录下存放了本工程各.c文件经过编译后产生的目标代码文件(.o文件)。这些文件经过"链接"后将生成可以下载到芯片中执行的机器码文件(.s19文件)。

<prm>目录下放置.prm文件。关于.prm文件的含义将在3.4.3节阐述。

<Sources>目录下放置了所有程序头文件和源程序文件,但是文件组织没有CodeWarrior中那样的逻辑组织,现在读者应该明白逻辑结构和物理结构的差异了吧。

在工程根目录下,读者只需要关注.mcp文件,这是一个工程的总文件,只需要通过打开它来打开整个工程。一般的编译器都有这样的一个文件,只是后缀名不同而已。

关于工程中的其他文件,一般都由CodeWarrior自动生成,读者无须关心。当然,鼓励读者善于用搜索引擎刨根问底,这里就不再赘述了。

需要一个说明:在阐述工程的逻辑文件结构时使用"文件夹"一词,而在阐述工程的物理结构时使用"目录"一词,没有特别含义,只是为了阐述方便而已。

前面两节分别从逻辑与物理两个层次概述了工程文件的组织情况。下面3.4.3～3.4.5节将对部分文件进行详细的说明,初学者对于这段内容可能感到比较困惑,甚至觉得有点复杂,但认真阅读这些内容并结合理解第一个工程,将有助于学习后面的内容,同时也会对项目有一个比较全面的认识。

3.4.3 系统启动及初始化相关文件

系统启动及初始化相关文件主要指链接文件Project.prm、启动文件start08.c及芯片

映像寄存器头文件。这些文件在一个芯片的工程中一般不再改动。

1. 链接文件 Project.prm

素材中【第 03 章（第一个程序）阅读资料】有“关于 CodeWarrior 中.prm 文件的详细说明”，可参考。这里给出简要说明。

.prm 文件主要实现了芯片的 RAM 和 ROM 的定义，初始化 RAM 中的变量，初始化堆栈的大小，定义复位向量，即应用程序的默认入口。还包含了启动代码，是硬件复位后的函数入口。具体代码如下。

```
//---------------------------------------------------------------------------*
// 文件名:   Project.prm                                                     *
// 说　明:   MC9S08AW60 的链接文件                                           *
//---------------------------------------------------------------------------*

NAMES END     //通过命令行 CodeWarrior 会将所有需要的文件传给链接文件
//但是也需要在这里添加自己的文件

SEGMENTS      //芯片的所有 RAM/ROM 地址在这里被列出.在下面的 PLACEMENT 中被使用
      Z_RAM                 =   READ_WRITE   0x0070 TO 0x00FF;
      RAM                   =   READ_WRITE   0x0100 TO 0x086F;
      ROM                   =   READ_ONLY    0x1860 TO 0xFFAF;
      ROM1                  =   READ_ONLY    0x0870 TO 0x17FF;
      ROM2                  =   READ_ONLY    0xFFC0 TO 0xFFCB;
// INTVECTS                   =   READ_ONLY    0xFFCC TO 0xFFFF;//为中断向量预留
END

PLACEMENT     //这里将所有预定义以及用户段都放置到上面定义的 SEGMENTS 中
      DEFAULT_RAM                          //非 0 页变量
                                           INTO   RAM;

      _PRESTART,                           // 启动代码
      STARTUP,                             // 启动数据结构
      ROM_VAR,                             // 常量
      STRINGS,                             // 串字面值
      VIRTUAL_TABLE_SEGMENT,               // C++有效表段
      DEFAULT_ROM,
      COPY                                 // 复制信息: 如何初始化变量
INTO   ROM; // ROM1,ROM2: 为了更好地使用 ROM1,ROM2, 将选项-OnB=b 传给编译器

      _DATA_ZEROPAGE,                      // 第 0 页变量
      MY_ZEROPAGE                          INTO   Z_RAM;
END

STACKSIZE 0x0100                           //LIBDEF_HEAPSIZE
VECTOR 0 _Startup                          // 复位向量: 这是应用程序的默认入口
```

按所含的信息.prm 文件由五个组成部分构成。

(1) NAMES…END 部分

用于指定在链接时加入除本项目文件列表之外的额外目标代码文件，这种用法不常用，