



**KEA128**

# 快速应用指南

Rev2.0, 09/2015

苏州大学飞思卡尔嵌入式中心

<http://sumcu.suda.edu.cn>

---

## 内容提要

本文档为飞思卡尔 KEA128 系列微控制器的快速应用指南，目的是引导使用 KEA128 系列微控制器的用户快速入门，给出阅读 KEA128 系列微控制器其他资料的基本导引。

本文档由苏州大学飞思卡尔嵌入式中心王宜怀负责编写提纲并统稿，李会、范宁宁参与编写，飞思卡尔李越工程师参与讨论、编写及修改。

# 目 录

第 1 章 导读.....	1
1.1 飞思卡尔 Kinetis EA 系列 MCU 的型号命名方式.....	1
1.2 Kinetis EA 系列的主要技术特点.....	1
1.3 参考资料.....	2
1.4 如何使用本文档.....	2
第 2 章 KEA 系列型号、引脚功能及硬件最小系统.....	3
2.1 KEA 系列芯片已出型号及存储器地址范围.....	3
2.1.1 KEA 系列芯片已出型号.....	3
2.1.2 KEA128 芯片的 Flash 及 RAM 地址范围及作用.....	3
2.2 KEA128 引脚引脚图.....	4
2.3 KEA128 引脚基本技术指标.....	4
2.4 KEA128 引脚功能.....	5
2.5 KEA128 硬件最小系统.....	6
第 3 章 底层驱动构件.....	8
3.1 关于底层驱动构件的基本思想.....	8
3.2 KEA128 底层驱动构件的封装方法与使用.....	8
3.2.1 GPIO 驱动构件的制作方法.....	8
3.2.2 UART 驱动构件的制作方法.....	10
第 4 章 工程模板及使用方法.....	14
4.1 统一的工程框架（工程模板）.....	14
4.2 第一个完整的样例程序的测试方法.....	15
4.2.1 样例工程软件环境配置.....	15
4.2.2 工程实例的硬件配置阶段.....	16
4.2.3 工程实例的硬件连接及程序烧录.....	16
4.3 第一个完整的样例程序 main 函数之前执行过程解析.....	18
4.4 如何应用工程模板.....	18
4.4.1 如何增加一个新的驱动.....	18
4.4.2 如何增加一个新的中断.....	19
第 5 章 KEA128 在实时操作系统 MQX-Lite 下的应用.....	20
5.1 MQX-Lite 简介.....	20
5.2 MQX 的工程框架及第一个完整的样例程序.....	20
5.2.1 基于 MQX-Lite 的 KEA128 工程框架.....	20
5.2.2 MQX 的第一个样例工程.....	21
2.....	22
第 6 章 杂记.....	30
附录 A KEA128 最小系统.....	31
附录 B SKEAZ128MLK 引脚功能分配.....	32
附录 C KDS 集成开发环境简明使用方法.....	34
C.1 KDS 下载、安装与配置.....	34
C.1.1 KDS 的下载与安装.....	34
C.1.2 KDS 的配置.....	34
C.2 KDS 简明使用方法.....	36

C.2.1 导入现有工程到开发环境中 .....	36
C.2.2 编译与链接工程—产生可执行的机器码 .....	37
C.2.3 直接下载机器码到目标板—运行 .....	38
C.2.4 在 KDS 下进行跟踪调试 .....	39
C.2.5 KDS 的常用基本操作 .....	40
C.3 常见问题处理 .....	41
附录 D SWD-Programmer 简明使用方法 .....	46
D.1 相关开发工具的安装 .....	46
D.1.1 所需的工具软件清单 .....	46
D.1.2 软件安装过程 .....	46
D.2 使用 JFlash 独立写入软件进行烧录 .....	46
D.2.1 硬件连接 .....	46
D.2.2 烧录步骤 .....	46
D.3 常见问题处理 .....	48
D.3.1 指示灯频闪，不能烧录程序 .....	48
D.3.2 目标板芯片被加密 .....	49
附录 E OpenSDA-Programmer 简明使用方法 .....	50
E.1 驱动的安装 .....	50
E.2 下载机器码与跟踪调试 .....	50
E.2.1 使用 KDS 直接下载机器码 .....	50
E.2.2 使用 KDS 进行跟踪调试 .....	50
E.3 常见问题处理 .....	50
E.3.1 程序下载时，Flash Configurations 界面无法选择机器码文件 .....	50
附录 F printf 格式化输出 .....	51
F.1 printf 调用的一般格式 .....	51
F.1.1 格式字符串 .....	51
F.1.2 输出格式举例 .....	52
附录 G 《KEA128 开发套件》简明测试方法 .....	53
.....	53

## 第1章 导读

Kinetis EA 系列（简称 KEA 系列）MCU 是飞思卡尔针对汽车市场最新开发的 MCU，在汽车电子领域有着广泛的应用，例如，信息娱乐系统连接模块、停车辅助系统、DC/BLDC 电机控制、电子驻车制动、电池管理、泵/风扇控制器、智能无钥匙进入及启动系统等应用。

### 1.1 飞思卡尔 Kinetis EA 系列 MCU 的型号命名方式

飞思卡尔 Kinetis 系列 MCU 的型号众多，但同一子系列的 CPU 核是相同的，多种型号只是为了适用于不同的场合。为了方便实际应用时选型或者订购，需要了解飞思卡尔 MCU 芯片名称的含义。从型号中可以获得系列号、内核属性、Flash 大小、温度范围、封装类型、CPU 最高频率等丰富的信息。

Kinetis EA 系列型号标识格式为：“Q KEA A C FFF M T PP CC N”，各字段含义见表 1-1。

表1-1 Kinetis EA系列芯片命令字段说明

字段	说明	取值
Q	质量状态	S=汽车级；P=工程测试芯片
KEA	Kinetis 汽车系列号	KEA
A	内核属性	Z=M0+内核
C	CAN 总线可用性	N=不包含 CAN 模块；(Blank) =包含 CAN 模块
FFF	程序 Flash 大小	64 = 64KB；128=128KB
M	生产版本	F0=第一版本；F1=第一版本之后的修订版（有时无字段）
T	运行温度范围	M=-40℃~125℃
PP	封装类型	LH =64LQFP(10mm x 10mm)；LK =80LQFP(14mm x 14mm)
CC	CPU 最高频率	4 = 48 MHz（有的芯片无该字段，默认 48 MHz）
N	包装类型	R=卷包装；(空) =盒包装

例如，具体一个芯片型号为：SKEAZ128MLK，从该芯片型号标识可以获得如下信息：该芯片为汽车级、KEA 系列、M0+内核、包含 CAN 模块、程序 Flash 大小为 128KB、运行温度范围是 -40℃~125℃、80 引脚 LQFP 封装、CPU 最高频率为 48 MHz、盒包装。

### 1.2 Kinetis EA 系列的主要技术特点

Kinetis EA 系列 MCU 的主要技术特点有：

(1) MCU 运行频率高达 48MHz，通过 AEC-Q100 一级认证，可满足温度范围为零下 40° C 至零上 125° C 的严苛要求，并且增强了 ESD/EMC 性能。通常情况下，无需接外部晶振，即可满足良好的运行条件。

(2) 可扩展解决方案，包括高达 128KB 的闪存及 80LQFP (71GPIO) 的封装选项，并且可轻松扩展设计，以满足不同的汽车应用需求。

(3) 工作电压范围为 2.7V 至 5.5V，可驱动高电流电机/组件。5V 模拟/传感器组件可轻松连接至系统级设计。

(4) 时钟模块：振荡器 (Oscillator, OSC)，支持 32.768 kHz 晶振或 4 MHz 至 24 MHz 晶体或陶瓷谐振器；内部时钟源 (Internal Clock Source, ICS)，带有内部参考时钟 (ICSIRCLK) 和锁频环 (Frequencylocked-loop, FLL)，为 48 MHz 系统时钟提供 37.5 kHz 的预调整内部参考电压；内部 1 kHz 低功耗振荡器 (Low Power Oscillator, LPO)。

(5) 系统功能：电源管理模块 (Power Management Controller, PMC)，具有 Run（运

行)、Wait(等待)、Stop(停止)三种电源模式;低电压检测(Low-Voltage Detection, LVD),提供可选择的复位或中断触发;具有独立的时钟源看门狗(Watchdog, WDOG);可编程循环冗余校验模块(Cyclic Redundancy Check, CRC);串行线调试接口(Serial Wire Debug, SWD);位操作引擎(Bit Manipulation Engine, BME)。

(6) 人机接口:具有 71 个通用输入/输出口(General-Purpose Input/Output, GPIO);两个 32 位键盘中断模块(Keyboard Interrupt Modules, KBI);外部中断(External interrupt, IRQ)。

(7) 模拟模块:一个 16 通道的 12 位模数转换器(Analog-to-Digital Converter, ADC),可在停止模式下运行,提供硬件触发选项;两个模拟比较器(Analog Comparator, ACMP),包含一个 6 位数模转换器(Digital-to-Analog Converter, DAC)和可编程参考电压输入。

(8) 定时器:柔性定时器(FlexTimer Module, FTM)模块有三个 FTM 模块,其中有一个 6 通道的 FTM 和两个 2 通道的 FTM;一个 2 通道周期中断定时器(Periodic Interrupt Timer, PIT);一个 16 位脉冲宽度定时器(Pulse Width Timer, PWT);一个实时时钟(Real-Time Clock, RTC)。

(9) 通信接口:两个串行外设接口(Serial Peripheral Interface, SPI)模块;三个通用异步接收器/发送器模块(Universal Asynchronous Receiver/Transmitter, UART);两个内部集成电路模块(Inter-Integrated Circuit, I2C);一个 MSCAN 总线模块。可通过这些丰富的通信模块实现车内通讯需求。

(10) 快速原型工具可实现高质量软件开发,包括 Kinetis Design Studio (KDS)、CodeWarrior IDE、Processor Expert 软件建模工具、MQX™ RTOS 和 AUTOSAR。

## 1.3 参考资料

- [1] Freescale. KEA128 Sub-Family Data Sheet Rev 4, 2014. (简称 KEA 数据手册)
- [2] Freescale. KEA128 Sub-Family Reference Manual Rev 2, 2014. (简称 KEA 参考手册)
- [3] Freescale.面向汽车行业的 Kinetis EA 系列 MCU (文档编号: KINETISEAMCUFS REV 1), 2014. (KEA 简介)
- [4] 王宜怀、朱仕浪、郭芸 著. 嵌入式技术基础与实践(第 3 版) — ARM Cortex-M0+Kinetis L 系列微控制器, 清华大学出版社, 2013 年 8 月. (简称王宜怀 M0+书)
- [5] 王宜怀、朱仕浪、姚望舒 著. 嵌入式实时操作系统 MQX 应用开发技术—ARM Cortex-M 微处理器, 电子工业出版社, 2014 年 8 月. (简称王宜怀 MQX 书)

## 1.4 如何使用本文档

本文档为飞思卡尔 KEA128 微控制器的快速应用指南,目的是引导使用 KEA128 微控制器的用户快速入门,给出阅读 KEA128 微控制器其他资料的基本导引。

第 2 章给出了 KEA 系列型号、引脚功能及硬件最小系统,目的是快速了解硬件系统。

第 3 章以 GPIO 和 UART 构件为例介绍了如何制作 KEA 驱动构件。

第 4 章给出了无操作系统(NOS)下的工程框架(工程模板)及测试方法,剖析了 KEA128 从上电至进入主程序 main 函数的芯片启动过程,并给出了在工程框架中新增驱动程序和中断服务例程的方法。

第 5 章给出了 KEA128 在实时操作系统 MQX-Lite 下的应用并给出第一个样例工程。

附录给出了 KDS 集成开发环境简明使用方法、SWD-Programmer 简明使用方法、OpenSDA-Programmer 简明使用方法以及 printf 格式化输出。

## 第2章 KEA系列型号、引脚功能及硬件最小系统

### 2.1 KEA系列芯片已出型号及存储器地址范围

#### 2.1.1 KEA系列芯片已出型号

Kinetis EA 系列 MCU 由六个子系列组成，分别是：KEAZN8、KEAZN16、KEAZN32、KEAZN64、KEAZ64、KEAZ128，表 2-1 给出了 Kinetis EA 系列芯片的简明资源。所有 Kinetis EA 系列 MCU 均具有低功耗与丰富的混合信号控制外设，提供了不同的闪存容量和引脚数量，供实际应用选型。

表2-1 Kinetis EA系列芯片的简明资源

器件	KEAZN8	KEAZN16	KEAZN32	KEAZN64	KEAZ64	KEAZ128
Flash	8KB	16KB	32KB	64KB	64KB	128KB
RAM	1KB	2KB	4KB	4KB	8KB	16KB
CPU 频率	48MHz	40MHz	40MHz	40MHz	48MHz	48MHz
MSCAN	0	0	0	0	1	1
GPIO	22	57	57	57	71	71
UART	1	3	3	3	3	3
SPI	1	2	2	2	2	2
IIC	1	2	2	2	2	2
PWM	1	0	0	0	1	1
ACMP	2	2	2	2	2	2
12 位 ADC	12 通道	16 通道				
16 位 FTM	两个 FTM 模块，分别有 6 个通道和一个 2 个通道	三个 FTM 模块，其中一个 FTM 有 6 个通道和两个 FTM 有 2 个通道				
封装	16TSSOP/24QFN	32/64 引脚 LQFP 封装			64/80LQFP	

#### 2.1.2 KEA128芯片的Flash及RAM地址范围及作用

表 2-2 给出了 KEA128 芯片的存储器映像，此处研究的芯片为 SKEAZ128MLK。

表2-2 KEA128（SKEAZ128MLK）系统存储映像

区域划分	系统 32 位地址范围	功能说明
Flash 区	0x0000_0000~0x07FF_FFFF	Flash 空间，本芯片只使用 0x0000 0000~0x0001 FFFF，128KB，其中前 192B 为中断向量表
	0x0800_0000~0x1FFF_EFFF	保留
片内 RAM 区	0x1FFF_F000~0x1FFF_FFFF	SRAM_L 空间，4KB（普通 RAM 区）
	0x2000_0000~0x2000_2FFF	SRAM_U 空间，12KB（支持位操作 RAM 区）
	0x2000_3000~0x21FF_FFFF	保留
	0x2200_0000~0x2205_FFFF	384KB（映射到 12KB 的 SRAM_U 位带别名区）
	0x2206_0000~0x23FF_FFFF	保留
	0x2400_0000~0x3FFF_FFFF	448MB（位操作引擎 BME 访问 SRAM_U）
外设区	0x4000_0000~0x4007_FFFF	AIPS 外设：串口、定时器、模块配置等
	0x4008_0000~0x400F_FFFF	保留
	0x400F_F000~0x400F_FFFF	通用输入/输出（GPIO）模块
	0x4010_0000~0x43FF_FFFF	保留
	0x4400_0000~0x5FFF_FFFF	448MB（位操作引擎 BME 访问外设 0-127 槽）
外部 RAM、外设区	0x6000_0000~0xDFFF_FFFF	保留
私有外设总线区	0xE000_0000~0xE00F_FFFF	私有外设：系统时钟、中断控制器、调试接口

系统保留区	0xE010_0000~0xF000_1FFF	保留
	0xF000_2000~0xF000_2FFF	ROM 表：存放存储映射信息
	0xF000_3000~0xF000_3FFF	杂项控制单元
	0xF000_4000~0xF7FF_FFFF	保留
	0xF800_0000~0xFFFF_FFFF	IOPORT：GPIO（单周期访问），可被内核直接访问

## 2.2 KEA128引脚引脚图

图 2-1 给出的是 80 引脚 LQFP 封装的 KEA128 的引脚图。每个引脚都可能多个复用功能，有的引脚有两个复用功能，有的有四个复用功能，实际嵌入式产品的硬件系统设计时必须注意只能使用其中的一个功能。进行硬件最小系统设计时，一般以引脚的第一功能作为引脚名进行原理图设计，若实际使用的是其另一功能，可以用括号加以标注，这样设计的硬件最小系统就比较通用。

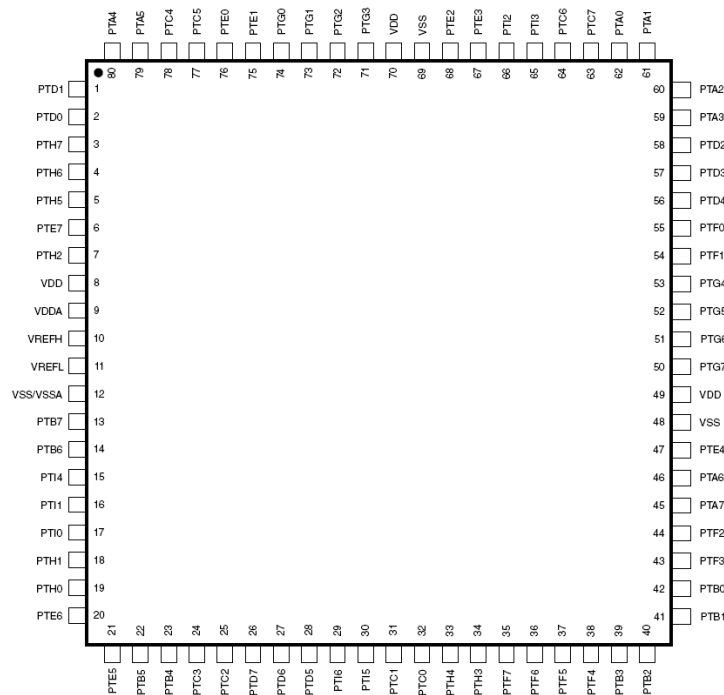


图 2-1 80 引脚 LQFP 封装 KEA128 引脚图

## 2.3 KEA128引脚基本技术指标

### 1. 引脚驱动能力

单个引脚标准驱动能力为 2.5mA。部分引脚可通过编程设定为高驱动模式，其驱动能力为 10mA。

### 2. 内部上下拉电阻

所有数字输入引脚（除了 PTA2 和 PTA3）可配置为内部上拉到 VDD，内部上拉电阻的阻值范围为 30K $\Omega$ ~50K $\Omega$ 。PTA2 和 PTA3 为开漏 I/O 引脚。KEA128 芯片数字输入引脚无内部下拉。

### 3. 未接引脚处理方法

建议通过编程将未使用引脚上拉，或外接上拉电阻。



## 2.4 KEA128引脚功能

KEA128 引脚功能复用表见 KEA128 技术参考手册“KEA128RM.pdf” 139 页的 10.2.1 节。

下面从需求与供给的角度把 MCU 的引脚分为“硬件最小系统引脚”与“I/O 端口资源类引脚”两大类。

### 1. 硬件最小系统引脚

表2-3 KEA128硬件最小系统引脚表

分类	引脚名		引脚号	典型值	功能描述
			LQFP		
电源	电源输入	VDD	8、49	3.3V	电源
		VSS	12、48	0V	地
		VDDA、VSSA	9、12	3.3V、0V	A/D 模块的输入电源
		VREFH、VREFL	10、11	3.3V、0V	A/D 模块的参考电压
复位	RESET		79	复位引脚（双向），作为输入，拉低可使芯片复位；作为输出，上电复位期间有低脉冲输出，输出脉宽最小约 100ns，芯片完成复位。常态下，引脚拉高，可通过复位按键拉低复位。	
晶振	EXTAL、XTAL		13、14	分别为无源晶振输入、输出引脚	
SWD 接口	SWD_CLK		78	SWD 时钟信号线	
	SWD_DIO		80	SWD 数据信号线	
引脚个数统计			LQFP 封装 12 个		

KEA128 硬件最小系统引脚包括电源类引脚、复位引脚、晶振引脚等，如表 2-3 所示。KEA128 芯片电源类引脚，LQFP 封装 7 个。芯片使用多组电源引脚分别为内部电压调节器、I/O 引脚驱动、A/D 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出脚，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。

### 2. I/O端口资源类引脚

除去需要服务的引脚外，其他引脚可以为实际系统提供 I/O 服务。芯片提供服务的引脚也可称为 I/O 端口资源类引脚。KEA128（80 引脚 LQFP 封装）具有 71 个 I/O 引脚。如表 2-4 所示，其中 A 口 8 个，B 口 8 个，C 口 8 个，D 口 8 个，E 口 8 个，F 口 8 个，G 口 8 个，H 口 8 个，I 口 7 个，每个引脚均具有多个功能。这些引脚在复位后，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。

表2-4 KEA128 I/O端口资源类引脚表

端口名	引脚数	引脚名	功能描述
A	8	PTA[0~7]	SWD/FTM/ACMP/ADC/KBI/UART/I2C/RESET/GPIO/IRQ/TCLK
B	8	PTB[0~7]	SPI/FTM/KBI/ADC/UART/ACMP/I2C/NMI/PWT/GPIO
C	8	PTC[0~7]	FTM/ADC/UART/RTC/ACMP/SWD/KBI/CAN/GPIO
D	8	PTD[0~7]	KBI/FTM/SPI/UART/PWT/GPIO
E	8	PTE[0~7]	FTM/SPI/KBI/TCLK/PWT/CAN/GPIO
F	8	PTF[0~7]	KBI/UART/FTM/ADC/GPIO
G	8	PTG[0~7]	KBI/FTM/SPI/GPIO
H	8	PTH[0~7]	KBI/PWT/BUSOUT/FTM/CAN/I2C/GPIO
I	7	PTI[0~6]	IRQ/UART/GPIO

网上光盘的“..\\02-hardware”下给出了 SKEAZ128MLK 芯片的构件化硬件最小系统原理图。

## 1. 电源及其滤波电路

最小系统

PTA0 62  
PTA1 61  
PTA2 60  
PTA3 59  
PTA4 80  
PTA5 70

PTA0/ADC0\_SE0/KBI0\_P0/FTM0\_CH0/I2C0\_4WSCOUT/ACMP0\_IN0  
PTA1/ADC0\_SE1/KBI0\_P1/FTM0\_CH1/I2C0\_4WSDAOUT/ACMP0\_IN1  
PTA2/KBI0\_P2/UART0\_RX/I2C0\_SDA  
PTA3/KBI0\_P3/UART0\_TX/I2C0\_SCL  
PTA4/SWD\_DIO/KBI0\_P4/ACMP0\_OUT  
PTA5

VDD\_8  
VDD\_49  
VDD\_70  
VDDA  
VREFH

KEA1  
KEA128 MCU

PTE4 47  
PTE5 21  
PTE6 20  
PTE7 6

PTE4/KBI1\_P3/SPI0\_FCS  
PTE5/KBI1\_P4  
PTE6/KBI1\_P5  
PTE7/KBI1\_P6  
PTE7/KBI1\_P7/TCLK2/FTM1\_CH1/CAN0\_TX

VSS\_48  
VSS\_69  
VSSVSSA  
VREFL

PTI6/TRQ 29

PTI6

电源（VDDx）与地（VSSx）包括很多引脚，如 VDDA、VSSA、VDD、VSS、VREFH 和 VREFL 等。至于外接电容，是由于集成电路制造技术所限，无法在 IC 内部通过光刻的方法制造这些电容。去耦是指对电源采取进一步的滤波措施，去除两级间信号通过电源互相干扰的影响。图 2-2 中，电源滤波电路用于改善系统的电磁兼容性，降低电源波动对系统的影响，增强电路工作的稳定性。为标识系统通电与否，可以增加一个电源指示灯。

## 2. 复位电路及复位功能

复位电路见图 2-3。复位，意味着 MCU 一切重新开始。复位引脚为 PTA5。若 PTA5 信号有效（低电平）则会引起 MCU 复位。复位电路原理如下：正常工作时复位输入引脚 PTA5 通过一个 10K 的电阻接到电源正极，所以应为高电平。若按下复位按钮，则 PTA5 脚接地为低电平，导致芯片复位。KEA128 的复位引脚是双向引脚，作为输入引脚，拉低可使芯片复位，作为输出引脚，上电复位期间有低脉冲输出，表示芯片已经复位完成。

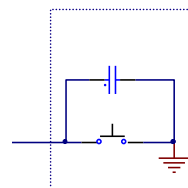


图 2-3 K128 的复位电路

复位引脚 PTA5 有一个内部上拉电阻，大小为 30~50K $\Omega$ 。MCU 上电复位后，系统选项寄存器 0（SIM\_SOPT0）的 RESET 引脚使能字段（RSTPE）为 1，则 PTA5 引脚复用为复位功能，使能内部上拉电阻。复位脉冲最小宽度为 1.5 个总线时钟周期。

## 3. 晶振电路

KEA128 芯片可使用内部的参考时钟或是外部的晶振来提供工作时钟。一般情况下，建议使用内部晶振。若与某些外部器件通信需要更为方便地调整频率，可使用外部晶振。使用外部晶振，需要外部晶振电路加以配合，见图 2-4。作为振荡源的晶体振荡器分为无源晶振（Crystal）和有源晶振（Oscillator）两种类型。有源晶振需要外接电源。无源晶振有两个引脚，由于无极性元件自身无法起振，因此需要借助辅助电路才能产生振荡信号。

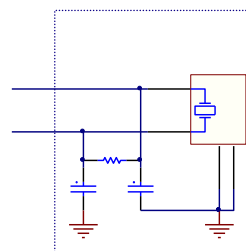


图 2-4 外部晶振电路

## 4. SWD接口电路

KEA128 芯片的调试接口 SWD 是基于 ARM CoreSight 架构，该架构在限制输出引脚和其他可用资源情况下，提供了最大的灵活性。CoreSight 是 ARM 定义的一个开放体系结构，以使 SOC 设计人员能够将其他 IP 内核的调试和跟踪功能添加到 CoreSight 基础结构中。通过 SWD 接口可以实现程序下载和调试功能。SWD 接口只需两根线，数据输入/输出线 SWD\_DIO 和时钟线 SWD\_CLK。如图 2-5 所示。

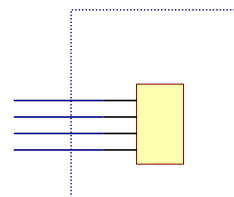


图 2-5 SWD 电路

## 第3章 底层驱动构件

### 3.1 关于底层驱动构件的基本思想

在嵌入式软件领域中，由于软件与硬件紧密联系的特性，使得与硬件紧密相连的底层驱动构件的生产成为嵌入式软件开发的重要内容之一。良好的底层驱动构件具备封装性、描述性、可移植性、可复用性等特性。为了使构件设计满足这些基本要求，嵌入式底层驱动构件的开发，应遵循层次化、易用性、鲁棒性及对内存的可靠使用原则。底层驱动构件封装规范及基本思想可参见王宜怀教授编著的《汽车电子 KEA 系列微控制器-基于 ARM Cortex-M0+ 内核》一书的第五章。

### 3.2 KEA128底层驱动构件的封装方法与使用

#### 3.2.1 GPIO驱动构件的制作方法

##### 1. 了解芯片的GPIO引脚

80 引脚 KEA128 芯片共有 71 个引脚可作为 GPIO 引脚，分别处于 PORTA、PORTB、PORTC、PORTD、PORTE、PORTF、PORTG、PORTH、PORTI 共 9 个端口。每个端口实际可用的引脚数因封装不同而有差异，80 引脚 KEA128 芯片的 71 个引脚名称标识如下：

- (1) PORTA 口有 8 个引脚，引脚名分别为 PTA7~0;
- (2) PORTB 口有 8 个引脚，引脚名分别为 PTB7~0;
- (3) PORTC 口有 8 个引脚，引脚名分别为 PTC7~0;
- (4) PORTD 口有 8 个引脚，引脚名分别为 PTD7~0;
- (5) PORTE 口有 8 个引脚，引脚名分别为 PTE7~0;
- (6) PORTF 口有 8 个引脚，引脚名分别为 PTF7~0;
- (7) PORTG 口有 8 个引脚，引脚名分别为 PTG7~0;
- (8) PORTH 口有 8 个引脚，引脚名分别为 PTH7~0;
- (9) PORTI 口有 7 个引脚，引脚名分别为 PTI6~0。

##### 2. KEA128芯片GPIO构件封装要点分析及提供的函数

以 GPIO 驱动构件为例，进行封装要点分析。即分析应该设计哪些函数及入口参数。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 的引脚统一编程，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。

另外，控制指示灯闪烁的程序中使用了 GPIO 驱动构件。指示灯定义为端口号|引脚号，这样便于用户理解和运用。现在添加一个内部解析函数 gpio\_port\_pin\_resolution，将指示灯的定义进行解析，得出具体端口号与引脚号。

##### 1) 内部解析函数(gpio\_port\_pin\_resolution)

指示灯定义为端口号|引脚号，例如，PORTB|5 表示为 B 口 5 号引脚。内部解析函数将“端口号|引脚号”解析为具体的端口号和引脚号，以便 GPIO 模块函数使用。

```
static void gpio_port_pin_resolution(uint_16 port_pin,uint_8* port,uint_8* pin)
```

其中 `uint_8` 是无符号 8 位整型的别名, `uint_16` 是无符号 16 位整型的别名, 其定义在工程文件夹下的“..\Common”文件夹的 `common.h` 文件中, 本书后面不再特别说明。

## 2) 模块初始化 (gpio\_init)

由于引脚具有复用特性, 把对应引脚设置成 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 模块初始化函数的参数为“端口号|引脚号”、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。这样 GPIO 模块初始化函数原型可以设计为:

```
void gpio_init(uint_16 port_pin, uint_8 dir, uint_8 state)
```

## 3) 设置引脚状态 (gpio\_set)

对于输出, 希望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“端口号|引脚号”, 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(uint_16 port_pin, uint_8 state)
```

## 4) 获得引脚状态 (gpio\_get)

对于输入, 希望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“端口号|引脚号”, 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:

```
uint_8 gpio_get(uint_16 port_pin)
```

## 5) 引脚状态反转 (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(uint_16 port_pin)
```

## 6) 引脚上拉使能函数 (gpio\_pull)

若引脚被设置成输入, 还可以设定内部上拉, KEA128 内部上拉电阻大小为 30~50KΩ。引脚上拉使能函数的原型为:

```
void gpio_pull(uint_16 port_pin, uint_8 pullselect)
```

# 3. GPIO构件的使用举例—制作light构件

GPIO 的应用广泛, 此处给出 GPIO 构件在小灯构件中的具体应用。

对于 LED 灯, 它的亮暗状态是由 GPIO 口输出到 LED 上电平的高低决定的, 因此 LED 灯的驱动构件的设计可基于 GPIO 驱动。具体 LED 灯接到哪个端口哪个引脚, 可使用宏定义设置, 例如:

```
//指示灯端口及引脚定义
#define LIGHT_RED      (PORTF|0)    //红灯使用的端口/引脚
#define LIGHT_GREEN    (PORTF|1)    //绿灯使用的端口/引脚
#define LIGHT_BLUE     (PORTD|4)    //蓝灯使用的端口/引脚
```

LED 灯的亮暗是正逻辑, 即高电平灯亮 (与具体硬件设计相关), 因此需宏定义, 方便记忆及使用:

```
//灯状态宏定义 (灯亮、灯暗对应的物理电平由硬件接法决定)
#define LIGHT_ON       0    //灯亮
#define LIGHT_OFF      1    //灯暗
```

基于以上分析, 该构件设计了以下函数:

- (1) 小灯初始化: **void light\_init**(uint16 port\_pin, uint8 state);
- (2) 小灯控制: **void light\_control**(uint16 port\_pin, uint8 state);

(3) 小灯亮暗反转: **void light\_change**(uint16 port\_pin);

下面给出 **light\_init**(uint16 port\_pin, uint8\_t state) 的函数样例。

```
void light_init(uint16 port_pin, uint8 state)
{
    gpio_init(port_pin, 1, state);
}
```

此外, 其他具体应用中需要对 GPIO 驱动封装的可参见以上分析, 封装好的构件放入工程框架的 06\_App\_Component 中, 这样方便移植使用。

### 3.2.2 UART驱动构件的制作方法

KEA128 芯片 UART 模块采用异步串行通信方式, 通信格式为 NRZ 数据格式, 支持全双工传输方式。UART 模块包括波特率发生器、发送器和接收器模块。UART 发送器和接收器使用相同的波特率发生器, 独立地操作。UART 驱动构件具有初始化、接收和发送三种基本操作。

#### 1. UART硬件结构

UART 模块为异步串行通信接口, 通过两根数据线实现了对串行数据的发送与接收操作, 同时为了便于布线在当前主流芯片中均实现了同一串口多个配置引脚功能, KEA128 中 UART0 具有 2 组可配置引脚, 因而在选择串口功能时在硬件上需要确定串口号和配置引脚。可通过系统集成模块 (System Integration Module, SIM) 提供的引脚选择寄存器 (Pin Selection Register, SIM\_PINSEL) 编程来设定。SIM\_PINSEL1 寄存器如图 3-1 所示。

例: 选择 PTI1、PTI0 分别作为 UART2 的发送和接收脚, 需要做如下配置:

1) 在 SIM\_PINSEL 寄存器中找出控制 UART2 引脚复用选择位

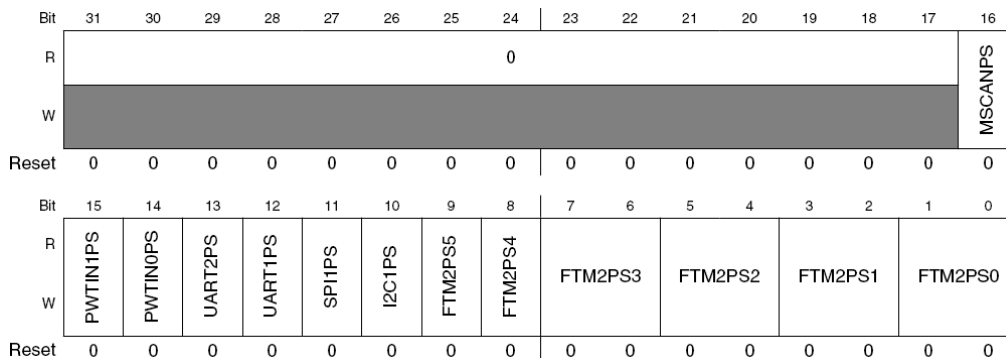


图 3-1 SIM\_PINSEL1 寄存器

查看《KEA 用户参考手册》文档的第 12 章可以找出, SIM\_PINSEL1 寄存器的第 13 位 (UART2PS) 控制 UART2 引脚复用选择。并且当 UART2PS=1 时, UART2\_TX 和 UART2\_RX 分别映射到 PTI1、PTI0。

2) 设置 PTI1、PTI0 的功能为 UART2 的发送和接收脚的代码如下:

```
//复用引脚 PTI0 和 PTI1 分别为 UART2 的接收和发送
SIM_PINSEL1 |= SIM_PINSEL1_UART2PS_MASK; //即 SIM_PINSEL1 |= 0x2000u
```

#### 2. UART知识要素解析

对于 UART 模块进行操作必须要了解以下知识要素, 即串口号, 引脚组号, 波特率, 数据位, 校验位, 停止位, 数据发送, 数据接收。

串口号: KEA128 有 3 个串口, 宏定义 UART\_0~UART\_3 为 0~3;

引脚组号：宏定义为 UARTx\_GROUP，取值范围为 0~3；

波特率：1200~115200 的值；

### 3. UART 构件封装

UART 具有初始化、接收和发送三种基本操作。按照构件的思想，可将它们封装成三个独立的功能函数，初始化函数完成对 UART 模块的工作属性的设定，接收和发送功能函数则完成实际的通信任务。对 UART 模块进行编程，实际上已经涉及到对硬件底层寄存器的直接操作，因此，可将初始化、接收和发送三种基本操作所对应的功能函数共同放置在命名为 `uart.c` 的文件中，并按照相对严格的构件设计原则对其进行封装，同时配以命名为 `uart.h` 的头文件，用来定义模块的基本信息和对外接口。

按照模块所具有的基本操作来确定构件中应该具有哪些功能集合，是很自然也很重要的事情。但是，要实现编程的构件化，对具体的函数原型的设计则是重中之重。函数原型设计的好坏直接影响构件化编程的成败。下面就以 UART 的初始化、接收和发送三种基本操作为例，来说明实现构件化编程的全过程。

需要说明的是，实现构件化编程的 UART 软件模块应当具有以下几个特点：

(1) UART 模块是最底层的构件，它主要向上提供三种服务，分别是 UART 模块的初始化、接收单个字节和发送单个字节，向下则直接访问模块寄存器，实现对硬件的直接操作。

另外，从实际使用角度出发，它还需要封装接收 N 个字节和发送 N 个字节的功能函数。

(2) UART 模块在软件上对应 1 个 `uart.c` 程序源代码文件和 1 个 `uart.h` 头文件，当需要使用 UART 驱动构件时，大多数情况下只需简单拷贝这两个文件即可，无需对源代码文件和头文件进行修改，只有当进行不同芯片之间的移植时，才需要修改头文件中与硬件相关的宏定义。

(3) 上层构件或软件在使用该构件时，严格禁止通过全局变量来传递参数，所有的数据传递都直接通过函数的形式参数来接收。这样做不但使得接口简洁，更加避免了全局变量可能引发的安全隐患。

为了使 UART 驱动适用于所有可复用为 UART 功能的引脚，本驱动在 `uart.h` 文件中设计了相应的宏定义，使用该驱动时只需根据具体用到的引脚修改相应的宏定义。

例如，有两组引脚可复用为串口 1，若复用 PTC7、6 脚为发送、接收功能，则宏定义 `UART_1_GROUP` 为 1 即可。

```
#define UART_1_GROUP 1 //UART_1: 1=PTC7~6 脚,2=PTF3~PTF2 脚
```

下面给出 `void uart_init(uint_8 uartNo, uint_32 baud_rate)` 的函数样例。

```
void uart_init(uint_8 uartNo, uint_32 sel_clk, uint_32 baud_rate)
{
    //局部变量声明
    register uint_16 sbr;
    uint_8 temp;
    //声明 uartch1 为 UART_MemMapPtr 结构体类型指针
    UART_MemMapPtr uartch1 = UART_ARR[uartNo];

    //根据传入参数 uartNo，给局部变量 uartch1 赋值
    switch (uartNo)
    {
        case UART_0:
            #if (UART_0_GROUP==1)
                //复用引脚 PTA2 和 PTA3 分别为 UART0 的接收和发送
                SIM_PINSEL0 |= SIM_PINSEL_UART0PS_MASK;
            #endif

            #if (UART_0_GROUP==2)
```

```

//复用引脚 PTB0 和 PTB1 分别为 UART0 的接收和发送
SIM_PINSEL0 &= ~SIM_PINSEL_UART0PS_MASK;
#endif
SIM_SCGC |= SIM_SCGC_UART0_MASK;           //UART0 使能总线时钟
UART0_C2 |= UART_C2_TE_MASK;                //使能 UART0 发送功能
UART0_C2 |= UART_C2_RE_MASK;                //使能 UART0 接收功能

UART0_C2 |= UART_C2_RIE_MASK;                //使能 UART0 接收中断功能
case UART_1:
//复用引脚 PTC6 和 PTC7 分别为 UART1 的接收和发送
#if (UART_1_GROUP==1)
SIM_PINSEL1 &= ~SIM_PINSEL1_UART1PS_MASK;
#endif
//复用引脚 PTF2 和 PTF3 分别为 UART1 的接收和发送
#if (UART_1_GROUP==2)
SIM_PINSEL1 |= SIM_PINSEL1_UART1PS_MASK;
#endif

SIM_SCGC |= SIM_SCGC_UART1_MASK;           //UART1 使能总线时钟
UART1_C2 |= UART_C2_TE_MASK;                //使能 UART1 发送功能
UART1_C2 |= UART_C2_RE_MASK;                //使能 UART1 接收功能

UART1_C2 |= UART_C2_RIE_MASK;                //使能 UART1 接收中断功能

break;
case UART_2:
//复用引脚 PTF2 和 PTF3 分别为 UART2 的接收和发送
#if (UART_2_GROUP==1)
SIM_PINSEL1 &= ~SIM_PINSEL1_UART2PS_MASK;
#endif
#if (UART_2_GROUP==2)
//复用引脚 PTI0 和 PTI1 分别为 UART2 的接收和发送
SIM_PINSEL1 |= SIM_PINSEL1_UART2PS_MASK;
#endif

SIM_SCGC |= SIM_SCGC_UART2_MASK;           //UART2 使能总线时钟
UART2_C2 |= UART_C2_TE_MASK;                //使能 UART2 发送功能
UART2_C2 |= UART_C2_RE_MASK;                //使能 UART2 接收功能
UART2_C2 |= UART_C2_RIE_MASK;                //使能 UART2 接收中断功能
break;

default:
break; //传参错误, 返回
}

//暂时关闭串口发送与接收功能
uartch1->C2 &= ~(UART_C2_TE_MASK | UART_C2_RE_MASK);

//配置波特率使用 Bus clock = 24M 总线时钟
//配置串口工作模式,8 位无校验模式
uartch1->C1 = 0;
sbr = (uint_16) ((BUS_CLK_KHZ * 1000) / (baud_rate * 16));
temp = UART_BDH_REG(uartch1) & ~(UART_BDH_SBR(0x1F));
UART_BDH_REG(uartch1) = temp | UART_BDH_SBR(((sbr & 0x1F00) >> 8));
UART_BDL_REG(uartch1) = (uint_8) (sbr & UART_BDL_SBR_MASK);

//初始化控制寄存器、清标志位
uartch1->C1 = 0x00;

```



```

uartch1->C3 = 0x00;
uartch1->S1 = 0x1F;
uartch1->S2 = 0x00;
//启动发送接收
uartch1->C2 |= (UART_C2_TE_MASK | UART_C2_RE_MASK);
}

```

通过以上分析，可以设计 UART 驱动构件的 8 个基本功能函数。

- (1) 初始化：uart\_init (uint\_8 uartNo,uint\_32 sel\_clk,uint\_32 baud\_rate);
- (2) 发送单个字节：uint\_8 uart\_send1(uint\_8 uartNo, uint\_8 ch);
- (3) 发送 N 个字节：uint\_8 uart\_sendN (uint\_8 uartNo ,uint\_16 len ,uint\_8\* buff);
- (4) 发送字符串：uint\_8 uart\_send\_string(uint\_8 uartNo, void \*buff);
- (5) 接收单个字节：uint\_8 uart\_re1 (uint\_8 uartNo,uint\_8 \*fp);
- (6) 接收 N 个字节：uint\_8 uart\_reN (uint\_8 uartNo ,uint\_16 len ,uint\_8\* buff);
- (7) 使能串口接收中断：uart\_enable\_re\_int(uint\_8 uartNo);
- (8) 禁止串口接收中断：uart\_disable\_re\_int(uint\_8 uartNo);

#### 4. 调试Printf格式化输出

串口调试是嵌入式开发中最重要的调试手段之一，为了便于调试兼容标准 C 中的 Printf 格式化输出，必须在调用 UART 构件层的基础上封装 Printf 格式化输出函数，以备调试使用，使用方法见附录 F。

## 第4章 工程模板及使用方法

### 4.1 统一的工程框架（工程模板）

图 4-1 给出以 KEA128-Light 工程为例的树形工程结构模板，物理组织与逻辑组织一致。该模板是苏州大学飞思卡尔嵌入式中心为在 KDS 环境下开发 ARM Cortex-M4/M0+ Kinetis Kinetis K/L/EA 系列 MCU 应用工程而制作的。

该工程模板，与 KDS3.0 提供的 Demo 工程模板相比，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。

KEA128_Light	工程名
Binaries	
Includes	
01_Doc	文档文件夹
02_CPU	内核文件
03_MCU	MCU 相关文件
SKEAZ1284.h	KEAZ128 芯片头文件
startup_SKEAZ1284.S	启动代码
system_SKEAZ1284.c	系统初始化源文件
system_SKEAZ1284.h	系统初始化头文件
04_Linker_File	链接文件夹
intflash.ld	链接文件
05_Driver	驱动文件夹
gpio	GPIO 底层构件文件夹
gpio.c	GPIO 底层构件源文件
gpio.h	GPIO 底层构件头文件
06_App_Component	应用构件文件夹
light	小灯构件文件夹
light.c	小灯构件源文件
light.h	小灯构件头文件
07_Soft_Component	软件构件文件夹
08_Sources	源文件夹
includes.h	总头文件
isr.c	中断源文件
isr.h	中断头文件
main.c	主函数
Debug	工程输出

图 4-1 工程框架

#### （1）工程名与新建工程

不必在意工程名，而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工程的方法来建立，这样，复用的构件已经存在。不推荐使用 KDS3.0 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 8 个下级文件夹，除去 KDS 环境保留的文件夹 Includes 与 Debug，分别是 01\_Doc、02\_CPU、03\_MCU、04\_Linker\_File、05\_Driver、06\_App\_Component、07\_Soft\_Component、08\_Source。其简明功能及特点见表 4-1。

表4-1 工程文件夹内的基本内容

编号	文件夹	简明功能及特点
1	01_Doc	说明文档文件夹，工程改动时，及时记录。
2	02_CPU	CMSIS M0+内核文件。
3	03_MCU	MCU 文件夹，存放芯片头文件及芯片初始化文件，MCU 不同时，芯片头文件需更换。
4	04_Linker_File	链接文件夹，放置链接文件。
5	05_Driver	底层驱动文件夹，逐步加入各模块驱动构件。
6	06_App_Component	存放应用构件。应用构件被定义为通过调用底层驱动构件而完成特定功能的构件，例如 led、lcd、电机开关构件。
7	07_Soft_Component	抽象软件构件文件夹，与硬件不直接相关的软件构件，或调用底层构件完成的功能软件构件。
8	08_Source	源程序文件夹，含主程序文件、中断服务例程文件等。这些文件是工程开发人员进行编程的主要对象。

## 4.2 第一个完整的样例程序的测试方法

### 4.2.1 样例工程软件环境配置

#### 1. 集成开发环境KDS3.0.0

第一个样例程序（工程模板）基于 Kinetis Design Studio IDE v3.0.0（后简称 KDS）集成开发环境编写，其下载地址及简明使用方法见附录 C。

#### 2. 写入器

写入器若使用“SWD-Programmer”，见“附录 D SWD-Programmer 简明使用方法”。

若使用 TRK-KEA128 开发板的板载写入器“OpenSDA”，见“附录 E OpenSDA 简明使用方法”。

#### 3. TTL串口转USB接口驱动

若程序调试过程中使用 TTL 串口转 USB 接口线，需安装驱动程序 PL2303\_Prolific\_DriverInstaller\_v1.8.0.exe，该文件位于网上光盘“..\04-tool”下。安装步骤如下：点击 PL2303\_Prolific\_DriverInstaller\_v1.8.0.exe 驱动安装程序（此安装包为 32 位/64 位通用版本），安装过程不需要选择安装路径，点击下一步直到提示安装完成即可。安装完成后连接 TTL-USB 线，便可以在设备管理器的端口中看到类似“Prolific USB-to-Serial Comm Port (COM3)”提示，右击该端口查看属性，在驱动程序选项卡下的驱动程序详细信息中可以看到 PL2303 的两个驱动程序位置。驱动程序文件位置见图 4-2 所示。

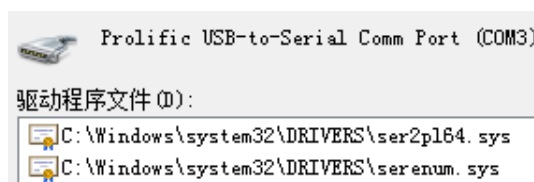


图 4-2 TTL-USB 驱动程序文件位置图

#### 4. 建立自己的工程存放目录及工程实例

进行工程实践时，希望建立自己的工程存放目录，以便合理存放开发的工程软件。如“KEA128-LS”（注意不能含有中文路径）。随后可将模板工程“..\02-software\program\CH06\_KEA128-UART”拷贝至此工作目录下，并更改工程目录名为自己起的名字，如

“CH06\_KEA128-UART-1”，见“..\02-software\program\CH06\_KEA128-UART-1”。

## 4.2.2 工程实例的硬件配置阶段

在 KDS 中打开样例工程，在工程浏览窗口可看到如图 4-1 所示的工程框架。工程实例的硬件是：苏州大学飞思卡尔嵌入式研发中心研制的 KEA128 开发板，板中使用了一个三色灯及 3 个串口，据此修改有关文件。

### 1. 小灯引脚配置

三色灯所在引脚及小灯亮暗的电平的配置可在“..\06\_App\_Component\light\light.h”文件中进行，代码如下：

```
//指示灯端口及引脚定义
#define LIGHT_RED      (PORTF|0)    //红灯使用的端口/引脚
#define LIGHT_GREEN    (PORTF|1)    //绿灯使用的端口/引脚
#define LIGHT_BLUE     (PORTD|4)    //蓝灯使用的端口/引脚

//灯状态宏定义（灯亮、灯暗对应的物理电平由硬件接法决定）
#define LIGHT_ON        0           //灯亮
#define LIGHT_OFF       1           //灯暗
```

### 2. 调试串口的引脚配置

串口的配置可在“..\05\_Driver\UART\uart.h”文件中进行配置，设调试串口为 UART2，使用 PORTD 口的 7、6 脚分别作为 TX、RX 引脚。代码如下：

```
//宏定义串口号
#define UART_0        0
#define UART_1        1
#define UART_2        2

//宏定义串口使用的引脚（由具体硬件板决定）
#define UART_0_GROUP   2    //UART_0: 1=PTA3~2 脚,2=PTB1~0 脚
#define UART_1_GROUP   1    //UART_1: 1=PTC7~6 脚,2=PTF3~PTF2 脚
#define UART_2_GROUP   1    //UART_2: 1=PTD7~6 脚,2=PTI1~0 脚
```

调试串口是将与 printf 函数关联的串口。

调试串口的引脚可在“..\07\_Soft\_Component\printf\printf.h”文件中进行配置。代码如下：

```
#define UART_Debug    UART_2    //printf 函数使用的串口号
```

修改完相应的板级配置后，需要重新编译工程，生成新的机器码文件（工程名.hex）。KDS 下编译工程的方法见附录 C。

## 4.2.3 工程实例的硬件连接及程序烧录

本工程实例所用是苏州大学飞思卡尔嵌入式研发中心研制的 KEA128 开发板，开发板的硬件介绍参见 [KEAQSG\(快速入门文档\).pdf](#)。程序的烧录与调试，可使用附录 D 所示的 SWD-Programmer。

### 1. KEA128硬件实物图

KEA128 硬件实物图见图 4-3。KEA128 硬件的最小系统见附录 A。

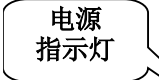
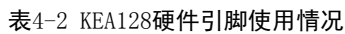


表 4-2 为 KEA128 硬件引脚使用情况。

表 4-2 为 KEA128 硬件引脚使用情况。



## 2. 写入器的连接

使用苏州大学飞思卡尔嵌入式中心自制的 SWD-Programmer, 则按 2.5 节 SWD 接口电路所示将写入器与板上的 SWD 接口相连 (参考附录 D)。

### 3. 调试串口的硬件连接

根据所确定的调试串口，进行调试串口的硬件连接。若使用 TTL-USB，含义见图 4-4。找到板上对应的引脚进行正确连接。TTL-USB 的驱动安装已在 4.2.1 节说明。

### 4. 下载程序

工程编译后生成的机器码文件在工程目录文件夹下的 Debug 子文件夹内，以 xxx.hex 命名（xxx 为工程名）。根据所用写入器的不同，程序烧录步骤参见附录 D 或附录 E。

### 5. 运行程序

程序正确下载后，重新上电直接运行程序，现象在工程目录的“01\_DOC 文件夹”下的 readme.txt 文件中。

给出的样例实验现象是：

- (1) KEA128 板上的三色灯依次闪烁。
- (2) 通过调试串口发送字符串“KEA128 是飞思卡尔针对……”。
- (3) 通过调试串口回发接收数据。

要在 PC 机上获得调试信息，可使用“..\ 03-tool”目录下的“C#2010 串口测试程序”或“串口调试工具.exe”，格式为：波特率 9600、8 位数据、无校验。



RX 接蓝线，TX 接白线，GND 接黑线

图 4-4 TTL-USB

## 4.3 第一个完整的样例程序main函数之前执行过程解析

芯片复位到 main 函数之前程序运行过程如下：

(1) 芯片上电复位后，芯片内部机制首先从 Flash 的 0x00000000 地址中，取出第一个表项的内容，赋给内核寄存器 SP（堆栈指针），完成堆栈指针初始化。例如，在链接文件 intflash.ld 中，链接时将 \_\_StackTop 的值链接到 Flash 的 0x00000000 地址中；

(2) 芯片内部机制将第二个表项的内容，赋给内核寄存器 PC（程序计数器）；

(3) 由于该表项存放启动函数 Reset\_Handler 的首地址，因而转到工程文件夹中“..\03\_MCU\startup.c”文件中，执行函数 startup；

(4) 在函数 startup\_SKEAZ1284.S 中，依次执行关闭所有外设中断并清除所有外设挂起中断标志、关闭看门狗、系统时钟初始化、将 ROM 中的初始化数据拷贝到 RAM 中、清零未初始化 BSS 数据段、进入主函数 main。

## 4.4 如何应用工程模板

工程模板的设计固然重要，但如何将工程模板应用起来，才是开发者真正关心的问题，接下来从“如何增加一个新的驱动”和“如何增加一个新的中断”两个方面介绍工程模板的应用。

### 4.4.1 如何增加一个新的驱动

在实际工程开发时，需要用到不同外设，添加相应的驱动程序必不可少。下面介绍如何在工程模板中增加一个新驱动。例如，增加一个 MSCAN 模块的驱动。

首先，在工程的“05\_Driver”下新建一个文件夹，命名为“can”，再将写好的 can 驱动函数拷贝到该文件夹下。此时，可以在程序中直接调用 can 驱动函数，但相对复杂，因此可以根据具体的需要对 can 驱动进行适当的封装，方便以后调用。封装的过程及方法参见 3.4 节，与具体硬件相关的可放在 06\_App\_Component 文件夹中。最后，要在工程配置中添加相应的文件夹路径，这样在工程中就可以正常使用添加或新封装的驱动了。

#### 4.4.2 如何增加一个新的中断

在实际工程开发时，也要用到不同的中断，下面介绍如何在工程模板中增加一个新中断。

增加一个新的中断需要使能相应模块的中断，注册及编写相应的中断服务例程。

KEA128 的中断向量表存放在 startup\_SKEAZ1284.S 中，是使用弱符号（weak）定义的，即只要工程中其他地方定义相应的中断服务例程，就可将之前定义成弱符号的中断服务例程覆盖掉而不会出重复定义的错误。换言之，工程需要用到什么中断，只需在中断向量表中找出其中断服务例程名，在 isr.c 中定义同名的中断服务例程函数，例如：

```
//串口 2 接收中断服务例程
void isr_uart2_re(void)
{
    uint_8 ReData;
    uint_8 flag;

    DISABLE_INTERRUPTS;    //关总中断

    flag=1;
    g_uart_num=2;          //标记当前使用的串口号

    ReData = uart_re1(UART_2, &flag);
    if (0 == flag)
    {
        uart_send1(UART_2, ReData);
    }

    ENABLE_INTERRUPTS;    //开总中断
}
```

主函数部分则调用开串口接收中断函数 void uart\_enable\_re\_int 开启相应的中断。

## 第5章 KEA128在实时操作系统MQX-Lite下的应用

消息队列执行（Message Queue eXecutive，MQX）是飞思卡尔公司提供的开源嵌入式 RTOS，其具有实时性高、内核精简、内核免费且有技术支持、开发工具成熟、外设驱动丰富及性价比高等特点。

在标准的 MQX RTOS 之外，飞思卡尔公司还提供了 MQX 的简化版，即轻量级 MQX（MQX-Lite）。MQX-Lite 所占用的芯片存储资源较标准 MQX 更少，但功能也相对标准 MQX 进行了部分裁剪。有关 MQX 的详细信息可参见王宜怀教授编著的《嵌入式实时操作系统 MQX 应用开发技术-ARM Cortex-M 微处理器》一书。

### 5.1 MQX-Lite简介

MQX-Lite 实时操作系统是标准 MQX<sup>1</sup>（Message Queue eXecutive，消息队列执行）实时操作系统的轻量级版本，在组成与功能上都是标准 MQX 的子集，主要针对资源有限的 MCU，支持应用以低于 4KB 的 RAM 空间运行。

MQX-Lite 并不随标准的 MQX 一起独立发布，而是打包作为处理器专家（Processor Expert，后简称 PE）的一个组件提供，目前的最新版本为 MQX-Lite V1.1.1。

MQX-Lite 由 MQX-Lite 内核与 PSP（处理器支持包）组成。表 14-1 列举出了 MQX-Lite 内核的所有组件及其功能。

表5-1 MQX-Lite内核组件

类型	名称	内容及作用
核心组件	初始化	初始化及创建自启动任务
	任务管理	任务的创建、管理和终止
	调度	基于优先级的抢占式 FIFO 任务调度
	轻量级信号量	实现轻量级信号量同步机制
	中断和异常处理	处理所有的硬件中断
可选组件	轻量级事件	实现轻量级事件同步机制
	互斥量	实现互斥量同步机制
	轻量级消息队列	实现任务间通信
	轻量级定时器	用于定时或周期性地调用应用函数
	内核日志	记录 MQX-Lite 的活动
	轻量级存储分配	动态存储分配

由表 5-1 可知，使用 MQX-Lite 实时操作系统，可以实现诸如多任务处理、基于优先级的抢占式任务调度、共享资源的同步访问、任务间的通信及中断处理等功能。

### 5.2 MQX的工程框架及第一个完整的样例程序

#### 5.2.1 基于MQX-Lite的KEA128工程框架

本节中基于 MQX-Lite 的 KEA128 工程框架是在 4.4.1 小节中提出的无操作系统工程组织框架的基础之上搭建起来的。与无操作系统下的工程差别仅在于添加了名为“09\_MQXLite”的文件夹，此文件夹中存放了 MQX-Lite 内核、PSP（内核支持包）及用户的应用任务代码。

<sup>1</sup> 关于标准版 MQX 实时操作系统的使用，可以参考王宜怀等著《嵌入式实时操作系统 MQX 应用开发技术》



MQX Lite 文件夹内包含了 app、bsp、config、include、kernel（内核）、psp 共 6 个下级文件夹。表 5-2 展示了 MQX Lite 文件夹下的文件与目录组成。

其中<app>文件夹是按照 MQX-Lite 实时操作系统任务设计的要求增加的应用程序目录，用户开发的应用程序任务主要在此目录下进行操作，该文件夹所包含的文件及其内容见表 5-3。它包含了应用软件的总头文件 app\_inc.h、任务模板列表文件 task\_templates.c 和自启动任务文件 task\_main.c，这三个文件的内容可根据工程要求进行修改，但文件名及文件内容布局不再更改，这是每个工程的共性部分，有利于工程的移植。而“task\_XXX.c”为自定义任务，“XXX”代表具体任务名。

<bsp>文件夹下包含开发板相关的配置及初始化，可参照完整版的 MQX 中提供的 BSP 模板，由用户自行编写。<config>文件夹下存放了 user\_config.h 配置文件，用于配置、裁剪 MQX-Lite 的功能。而剩下的<include>、<kernel>及<psp>文件夹下的文件是从 PE 安装目录下的 MQX-Lite 子目录<..\KDS\_3.0.0\eclipse\ProcessorExpert\lib\mqxlite\V1.1.1\source>中提取来的。

表5-2 MQX Lite文件夹的下级文件夹组成

文件夹	内含文件说明
app	app_inc.h 任务公共头文件
	task_templates.c 任务模板列表文件
	task_main.c 自启动任务文件
	task_XXX.c 用户应用任务文件
bsp	开发板相关配置文件，由用户自行编写
config	user_config.h 用于配置文件，配置、裁剪 MQX-Lite 功能
include	存放 MQX-Lite 系统的所有源码头文件
kernel	存放 MQX-Lite 内核功能源代码
psp	与芯片相关的调度、中断等汇编源程序

表5-3 app文件夹的内容

文件名	名称	内容
app_inc.h	任务公共头文件	用于定义任务模板号、任务的声明，文件名不变，内容可修改。
task_main.c	MQX 自启动任务	用于创建用户自定义的任务，文件名不变，内容随着用户任务变动而修改。
task_templates.c	任务模板列表	用于向 MQX-Lite 操作系统登记任务，在模板列表中，任务模板编号、任务函数体、任务栈大小、任务优先级、任务启动属性等；文件名不变，内容随着用户任务变动而修改。
task_XXX.c	自定义任务	该函数用于实现自定义任务的功能，“XXX”被实际任务名取代。

与无操作系统下的编程相比，原来放在 main.c 函数中的应用程序在 MQX-Lite 中被划分成了各个不同功能的应用任务函数（task\_XXX.c），而 main.c 中只需调用 \_mqxlite\_init() 及 \_mqxlite() 函数来初始化并启动 MQX-Lite，让操作系统来进行任务的管理与调度。

## 5.2.2 MQX 的第一个样例工程

### 1. 样例工程的功能

KEA128 在 MQX-Lite 下的第一个样例工程以上节给出的框架为模板，完成指示灯的闪烁与串口接收发送数据功能，具体要求如下：

硬件连接：

（1）红灯、绿灯、蓝灯硬件连接于 KEA128 开发板上的 S1，由 KEA128 芯片的 PTF0、PTF1 和 PTD4 引脚控制，高电平灯灭，低电平灯亮。

（2）串口 2 作为缺省通信设备与外界通信，硬件连接于 KEA128 芯片的 PTD6、PTD7

引脚，作为串口 2 的收、发引脚。

软件功能：

样例工程包含 5 个任务和一个中断服务例程，简要说明如下：

- (1) 任务 1：自启动任务，初始化外设，创建其他任务；
- (2) 任务 2：小灯 1（红灯）每隔 2 秒亮 1 秒；
- (3) 任务 3：小灯 2（绿灯）每隔 2 秒亮 1 秒；
- (4) 任务 4：小灯 3（蓝灯）每隔 2 秒亮 1 秒；
- (5) 任务 5：使用串口 2 作为通信端口，当接收到完整的数据帧时，将整个数据帧内容发送至 PC 机。

(7) 串口 2 中断：接收串口数据组成数据帧。接收数据的格式：“帧头（'A'） + 数据长度（1 字节） + 有效数据 + 帧尾（'D'）”。数据长度是指“有效数据”的字节数。串口 3 初始化波特率为 9600，1 位停止位，无校验。

## 2. 样例工程任务设计

依据样例工程的功能要求，需在工程 09\_MQXLite\app 目录中添加相关的任务函数，包括：小灯闪烁任务函数 task\_A()、task\_B()、task\_C()、串口 4 发送数据到 PC 机的 task\_D() 任务函数、自启动任务函数 task\_main()、应用任务公共头文件 app\_inc.h 以及任务模板列表文件 task\_templates.c 等。

### 1) 小灯闪烁任务函数 task\_A()

小灯 2 任务函数 task\_B() 和小灯 1、小灯 3 的任务函数基本功能和实现方法一样，区别在于任务循环体前的延时时长不同。小灯 1~3 都是每隔 2 秒亮 1 秒，小灯 2~3 在首次亮暗前依次多延时 1 秒。小灯使用的引脚可在工程目录..\06\_App\_Component\light\light.h 文件中进行配置。下面仅对小灯 2 的程序实现进行分析。这里给出任务函数 task\_B() 的源码：

```
//=====
//任务名称：task_B
//功能概要：light2 每隔 2 秒亮 1 秒，首次亮暗前延时 1 秒
//参数说明：未使用
//=====
void task_B(uint_32 initial_data)
{
    _time_delay_ticks(DELAY_TIMES);
    while (TRUE)
    {
        printf("---Light2 On---\r\n");
        light_control(LIGHT_2,LIGHT_ON);    // 小灯 2 亮
        _time_delay_ticks(DELAY_TIMES);    // 延时 DELAY_TIMES(放弃 CPU 控制权)
        light_control(LIGHT_2,LIGHT_OFF);  // 小灯 2 灭
        _time_delay_ticks(DELAY_TIMES*2);  // 延时 DELAY_TIMES(放弃 CPU 控制权)
    }
}
```

在 while(TRUE) 永久循环体内，通过 \_time\_delay\_ticks 函数实现时间的延时，DELAY\_TIMES 为在 app\_inc.h 中定义的宏，值为 200，即小灯亮 200\*1=200 个时钟滴答，灭 200\*2=400 个时钟滴答（程序中默认 1 个时钟滴答为 5ms，可在 bsp.h 中修改，200\*5ms=1s，400\*5ms=2s），以实现 3 盏小灯轮流点亮，在延时期操作系统可以调度执行其他的任务。

### 2) 帧数据发送任务函数 task\_D()

任务中使用 UART2 作为调试串口 UART\_Debug，可在..\09\_MQX\app\app\_inc.h 中修改

为其他串口模块，而 UART2 使用的收发引脚可在..\05\_Driver\uart\uart.h 中进行配置。对于串口 2 通信的实现包含接收中断的功能，依据 ISR 设计的基本原则，中断服务例程的设计尽可能简短，以便其它异步中断事件能得到响应。在串口 2 接收中断服务例程 UART2\_RX\_ISR()中只完成数据接收到全局变量数组 g\_UART\_ISR\_buffer[]中，再由 task\_D()任务函数将全局变量中的数据通过串口发送到 PC 机。中断服务例程 UART2\_RX\_ISR()位于工程\08\_source\isr.c 文件中，这是兼容无操作系统的中断服务例程进行布置的。实现代码如下：

```
#include "app_inc.h"    //应用任务公共头文件

//=====
//任务名称: task_D
//功能概要: 将串口 2 中断接收的完整数据帧发送至 PC 机
//参数说明: 未使用
//备    注: 使用全局变量 lwevent_group1, UART2_RecData_Event,g_UART_ISR_buffer
//=====
void task_D (uint32_t initial_data )
{
    int itemp;
    //进入主循环
    while (TRUE)
    {
        //等待串口 2 接收完整数据帧事件位 (Event_UART2_ReData)
        _lwevent_wait_ticks(&lwevent_group1, Event_UART2_ReData, FALSE, 0);
        g_UART_FrameCount++;           //接收的帧数加 1

        printf("串口 2 接收的完整数据帧: (--0x%02X--)",g_UART_FrameCount);

        //发送帧数据
        for(itemp=0;itemp<g_UART_ISR_buffer[1]+3;itemp++)
        {
            printf("%x",g_UART_ISR_buffer[itemp]);
        }
        printf("\r\n");
        //清除串口 2 接收完整数据帧事件位 (Event_UART2_ReData)
        _lwevent_clear(&lwevent_group1, Event_UART2_ReData);
    }
}
```

### 3) 任务模板和任务公共头文件

编写好以上各任务后，必须在任务模板列表文件 task\_templates.c 中按任务模板的格式进行登记，设置任务栈的大小、任务优先级以及任务属性等，为任务创建做好准备。并在任务公共头文件 app\_inc.h 中静态分配好任务栈，定义任务模板索引号，对各个任务进行申明。**需要注意的是任务模板列表的最后一个任务模板必须为{0}**，它表示任务模板列表的结束。在任务创建过程中，经常需从任务模板列表的第一条信息，按模板索引号进行查找，若找到的模板索引号为 0，表示已经到模板列表最后一项，查找过程结束。

task\_templates.c:

```
#include "app_inc.h"
// 为自启动任务注册任务栈
const uint_8 *mqx_task_stack_pointers[] =
{
    task_main_stack,
    NULL
};
```

```
// 任务模板列表
const TASK_TEMPLATE_STRUCT MQX_template_list[] =
{
    //任务编号,    任务函数,        任务栈大小,    优先级,任务名,任务属性
    {TASK_MAIN, task_main, TASK_MAIN_STACK_SIZE, 9, "main", MQX_AUTO_START_TASK},
    {TASK_A,    task_A,    TASK_A_STACK_SIZE,    10, "task_A", 0},
    {TASK_B,    task_B,    TASK_B_STACK_SIZE,    10, "task_B", 0},
    {TASK_C,    task_C,    TASK_C_STACK_SIZE,    10, "task_C", 0},
    {TASK_D,    task_D,    TASK_D_STACK_SIZE,    8, "task_D", 0},
    {0}
};
```

app\_inc.h:

```
#ifndef __APP_INC_H_
#define __APP_INC_H_
//-----

//1.包含头文件
#include "isr.h"
#include "bsp.h"
#include "mqxlite.h"
#include "mqx_inc.h"
#include "light.h"
#include "lwevent.h"
#include "printf.h"

//2.声明全局变量
//防止全局变量重复声明的前缀处理方法
#ifdef GLOBLE_VAR                //GLOBLE_VAR 在 task_main.c 文件中宏定义
#define G_VAR_PREFIX             //task_main.c 文件中使用全局变量不加“extern”前缀
#else
#define G_VAR_PREFIX extern //其他文件中使用全局变量自动加“extern”前缀
#endif
//声明全局变量（全局变量类型前一律前缀 G_VAR_PREFIX）
G_VAR_PREFIX uint8_t g_UART_ISR_buffer[100];    //存放 UART2 接收的数据帧
G_VAR_PREFIX uint8_t g_UART_FrameCount;        //UART2 接收的数据帧的个数
G_VAR_PREFIX LWEVENT_STRUCT lwevent_group1;    //轻量级事件组
#define Event_UART2_ReData ((1uL)<<(6)) //串口 2 接收完整数据帧事件位
#define DELAY_TIMES      200            // 每个 tick 对应 5ms

//3.登记任务模板编号
#define TASK_MAIN          1
#define TASK_A              2
#define TASK_B              3
#define TASK_C              4
#define TASK_D              5

//4.为任务创建任务栈
#define TASK_MAIN_STACK_SIZE (sizeof(TD_STRUCT) + 1024 + PSP_STACK_ALIGNMENT + 1)
#define TASK_A_STACK_SIZE    (sizeof(TD_STRUCT) + 512 + PSP_STACK_ALIGNMENT + 1)
#define TASK_B_STACK_SIZE    (sizeof(TD_STRUCT) + 512 + PSP_STACK_ALIGNMENT + 1)
#define TASK_C_STACK_SIZE    (sizeof(TD_STRUCT) + 512 + PSP_STACK_ALIGNMENT + 1)
#define TASK_D_STACK_SIZE    (sizeof(TD_STRUCT) + 512 + PSP_STACK_ALIGNMENT + 1)

G_VAR_PREFIX uint_8 task_main_stack[TASK_MAIN_STACK_SIZE];
G_VAR_PREFIX uint_8 task_A_stack[TASK_A_STACK_SIZE];
G_VAR_PREFIX uint_8 task_B_stack[TASK_B_STACK_SIZE];
G_VAR_PREFIX uint_8 task_C_stack[TASK_C_STACK_SIZE];
```

```
G_VAR_PREFIX uint_8 task_D_stack[TASK_D_STACK_SIZE];
```

```
//5.声明任务函数
```

```
void task_main(uint32_t initial_data);
```

```
void task_A(uint32_t initial_data);
```

```
void task_B(uint32_t initial_data);
```

```
void task_C(uint32_t initial_data);
```

```
void task_D(uint32_t initial_data);
```

```
//6.声明中断服务例程 ISR
```

```
void UART2_RX_ISR(pointer user_isr_ptr);
```

```
//-----
```

```
#endif //app_inc.h
```

#### 4) 自启动任务及任务的创建

自启动任务 `task_main` 是 MQX-Lite 启动后执行的第一个应用任务，它是在 MQX-Lite 系统初始化时由 `_mqxlite()` 函数创建。自启动任务的功能就是调用 `_task_create_at()` 函数创建其它的任务。根据给定的任务索引号，`_task_create_at()` 函数从模板列表查找任务的相关信息创建任务，即把任务的相关信息填写到任务描述符节点，并把该节点加入到任务就绪队列，供任务调度器调度。

```
#define GLOBLE_VAR
```

```
#include "app_inc.h"
```

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

```
//任务名称: task_main
```

```
//功能概要: MQX 自启动任务，主要实现全局变量初始化、创建其他任务、安装用户 ISR
```

```
//参数说明: 未使用
```

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

```
void task_main(uint32_t initial_data)
```

```
{
```

```
    //1.全局变量初始化
```

```
    _lwevent_create(&lwevent_group1,0); //创建轻量级事件组
```

```
    g_UART_FrameCount=0; //接收的帧数
```

```
    //2. 关总中断
```

```
    DISABLE_INTERRUPTS;
```

```
    //3.外设初始化
```

```
    light_init(LIGHT_1,LIGHT_OFF); //小灯 1 初始化
```

```
    light_init(LIGHT_2,LIGHT_OFF); //小灯 2 初始化
```

```
    light_init(LIGHT_3,LIGHT_OFF); //小灯 3 初始化
```

```
    uart_init(UART_Debug,9600); //串口 2 使用总线时钟 20MHz
```

```
    printf("Hello Uart_%d!\r\n",UART_Debug); //串口发送初始化提示
```

```
    //4.创建其他任务
```

```
    _task_create_at(0, TASK_A, 0, task_A_stack, TASK_A_STACK_SIZE);
```

```
    _task_create_at(0, TASK_B, 0, task_B_stack, TASK_B_STACK_SIZE);
```

```
    _task_create_at(0, TASK_C, 0, task_C_stack, TASK_C_STACK_SIZE);
```

```
    _task_create_at(0, TASK_D, 0, task_D_stack, TASK_D_STACK_SIZE);
```

```
    //5.安装用户 ISR
```

```
    _int_install_isr(UART0_IRQn+UART_Debug+16,UART2_RX_ISR,NULL); //注册调试串口的 ISR
```

```

//6.使能模块中断及总中断
uart_enable_re_int(UART_Debug); //使能调试串口接收中断

//7.开总中断
ENABLE_INTERRUPTS;

//-----执行完毕，本任务进入阻塞态-----
_task_block();
}

```

### 5) 串口2中断服务例程

串口2中断服务例程从串口接收数据组建数据帧，并回显接收到的数据。定义的串口数据帧格式为：帧头（1B）+数据长度（1B）+数据内容（NB）+帧尾（1B），此处使用字符“A”（0x41）作为帧头，字符“D”（0x44）作为帧尾。当接收到一个完整的数据帧时，置接收成功事件位，唤醒帧数据发送任务函数 task\_E()。若最后一个帧尾数据不为“D”，则认为前面收到的数据均为无效数据重新开始接收计数。

```

//=====
//文件名称: isr.c
//功能概要: 中断底层驱动构件源文件
//版权所有: 苏州大学飞思卡尔嵌入式中心(sumcu.suda.edu.cn)
//版本更新: 2015-4-26 V1.0
//芯片类型: KEA128
//=====

#include "includes.h"
#include "app_inc.h"

#define FrameHead (0x41) //帧头
#define FrameTail (0x44) //帧尾
//内部函数声明
uint8_t CreateFrame(uint8_t Data,uint8_t * buffer);

////=====中断函数服务例程=====

//=====
//中断例程: UART2_RX_ISR
//功能概要: 串口2中断服务例程，每收到一个字节执行一次本程序，发送回去，并组帧
//参数说明: user_isr_ptr（未使用）
//备 注: 使用全局变量 g_UART_ISR_buffer[]
//=====
void UART2_RX_ISR( pointer user_isr_ptr)
{
    uint8_t Redata; //存放接收来的数据
    uint_8 flag = 1;

    Redata = uart_re1(UART_2, &flag); //接收的一个字节数据
    if (flag!=0) goto UART2_RX_ISR_End;
    else uart_send1(UART_2, Redata); //将收到的一个字节数据发送回去
    //组帧成功,置“串口2接收完整数据帧事件位”
    if(CreateFrame(Redata,g_UART_ISR_buffer)!=0)
    {
        _lwevent_set(&lwevent_group1,Event_UART2_ReData);
    }

    UART2_RX_ISR_End: ;
}

```

//内部调用函数

```
//=====
//ISR 名称: Createbuffer
//功能概要: 组建数据帧, 将待组帧数据加入到数据帧中
//参数说明: Data:待组帧数据
//          buffer:数据帧变量
//函数返回: 组帧状态    0-组帧未成功, 1-组帧成功
//备    注: 十六进制数据帧格式
//          帧头      + 数据长  + 有效数据  + 帧尾
//          FrameHead + len    + 有效数据  + FrameTail
//=====
uint8_t CreateFrame(uint8_t Data,uint8_t * buffer)
{
    static uint8_t frameCount=0;    //组帧计数器
    uint8_t frameFlag;              //组帧状态
    frameFlag=0;
    //根据静态变量 frameCount 组帧
    switch(frameCount)
    {
        case 0:    //第一个数据
        {
            if (Data==FrameHead)    //收到数据是帧头 FrameHead
            {
                buffer[0]=Data;
                frameCount++;
                frameFlag=0;          //组帧开始
            }
            break;
        }
        case 1:    //第二个数据, 该数据是随后接收的数据个数
        {
            buffer[1]=Data;
            frameCount++;
            break;
        }
        default:    //其他情况
        {
            //第二位数据是有效数据长度,根据它接收余下的数据直到帧尾前一位
            if(frameCount>=2 && frameCount<=(buffer[1] + 1))
            {
                buffer[frameCount]=Data;
                frameCount++;
                break;
            }

            //若是末尾数据则执行
            if(frameCount>=(buffer[1]+2))
            {
                if (Data==FrameTail)    //若是帧尾
                {
                    buffer[frameCount]=Data; //将帧尾 0x44 存入缓冲区
                    frameFlag=1;           //组帧成功
                }
                frameCount=0;              //计数清 0, 准备重新组帧
                break;
            }
        }
    }
}
```

```

    }
} //switch_END
return frameFlag; //返回组帧状态
}

```

### 3. 样例工程的执行流程及运行结果

表5-4 样例任务

任务函数	优先级	任务属性	任务功能
task_main	9	自启动	打印输出唤醒消息，启动其它任务
task_A~task_C	10	无	小灯 1~3 轮流闪烁
task_D	8	无	通过串口 2 接收 PC 机数据，回发相应数据的功能

样例工程中的任务信息如表 5-4 所示。样例工程在启动后进入 main()函数，在其中调用 \_mqxlite\_init 函数及 \_mqxlite 函数来初始化并启动 MQX-Lite 操作系统，在进行一系列的初始化过程之后，进入自启动任务 task\_main。

在自启动任务 task\_main 中，初始化所用到的外部设备，并依次创建其他任务 task\_A、task\_B、task\_C 及 task\_D，最后调用 task\_block()函数将自身阻塞。此后 MQX Lite 将按照所创建任务的优先级对任务进行调度。

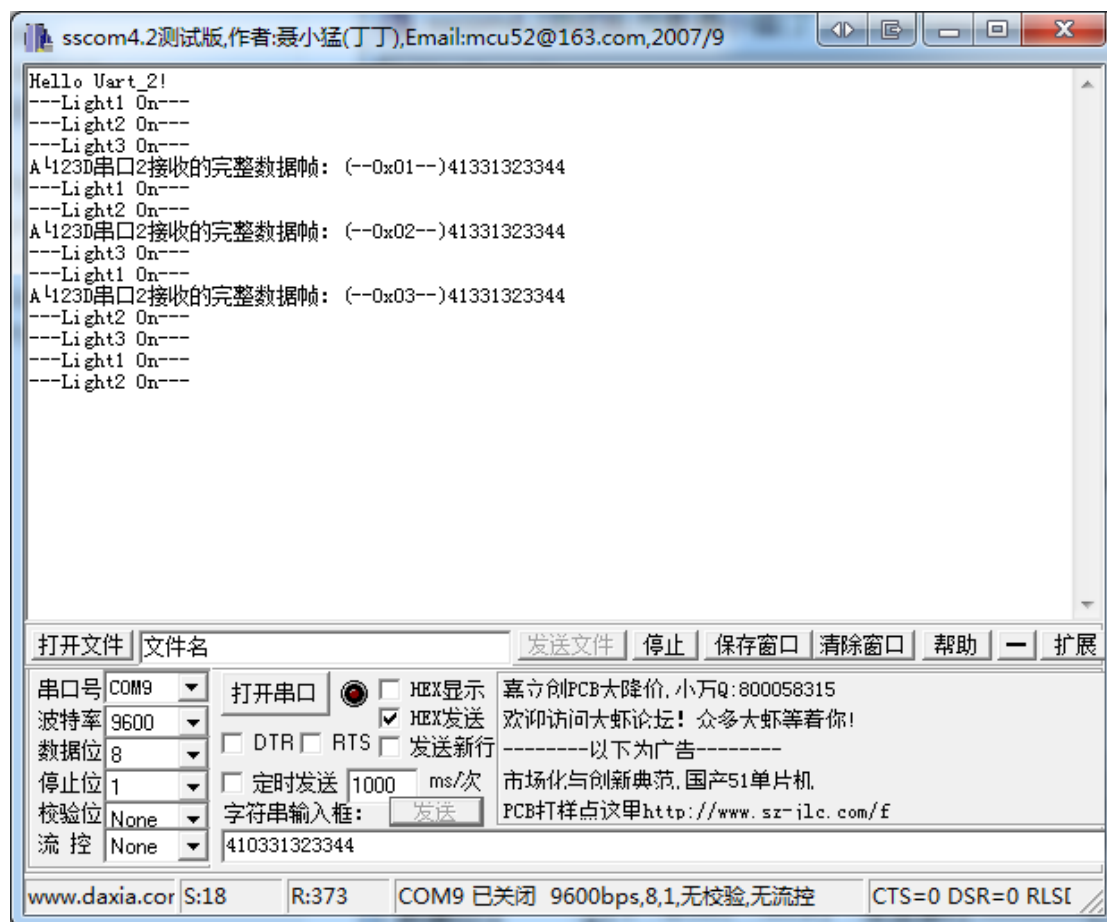


图 5-1 样例工程运行结果

需要注意的是，在创建其他任务的过程中，任务 task\_A 到 task\_C 的优先级（都为 10）低于自启动任务 task\_main 的优先级（9），所以在任务创建后这两个任务会进入各自优先级的任务就绪队列中等待调度；而在创建任务 task\_D 后，由于其任务优先级（8）高于自启动任务 task\_main（9），所以其会在下一个系统调度时刻（1 个 systick 的时间）到来时抢占当

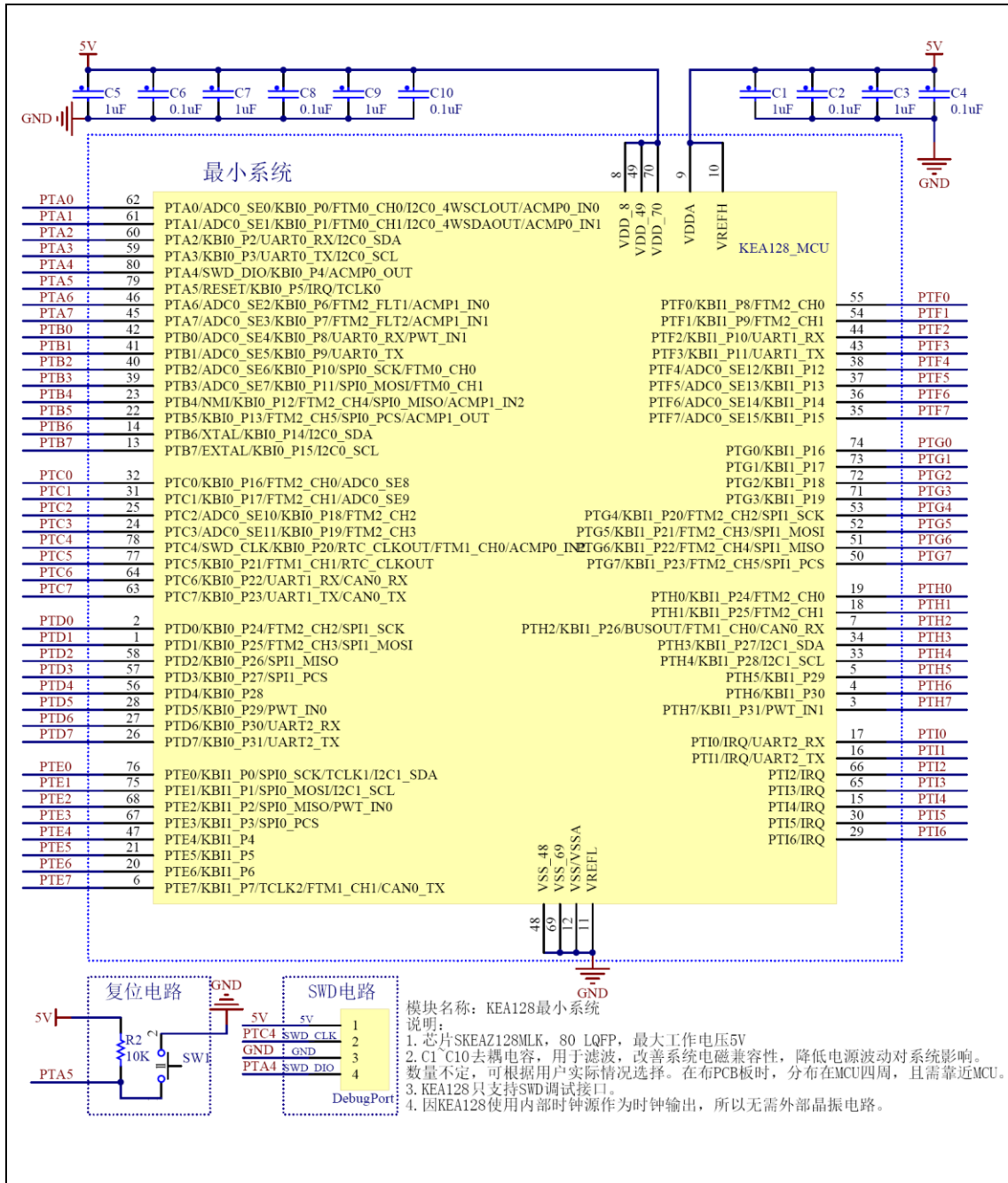


前任务开始执行。而由于在 `task_D` 的任务函数循环体开始时使用了轻量级事件来同步任务，在串口 2 没有接收到完整的数据帧时，`task_D` 被轻量级事件阻塞，系统会重新回到 `task_main` 任务继续执行。

将样例工程编译下载到目标板，重新上电后，可以观察小灯 1~3 按设定的方案不停闪烁。通过串口 2 与 PC 机连接，在串口调试器中向目标板发送数据，可看到其回显到数据接收框中。从串口调试器以 16 进制发送“410331323344”一帧数据，在显示窗口回送了数据帧的内容，如图 5-1 所示。

## 第6章 杂记

## 附录A KEA128最小系统



## 附录B SKEAZ128MLK引脚功能分配

引脚号	引脚名	缺省功能	优先级低->优先级高	ALT0	ALT1	ALT2	ALT3	ALT4	ALT5	ALT6
1	PTD1	DISABLED	PTD1	KBIO_P25	FTM2_CH3	SPI1_MOSI				
2	PTD0	DISABLED	PTD0	KBIO_P24	FTM2_CH2	SPI1_SCK				
3	PTH7	DISABLED	PTH7	KB11_P31	PWT_IN1					
4	PTH6	DISABLED	PTH6	KB11_P30						
5	PTH5	DISABLED	PTH5	KB11_P29						
6	PTE7	DISABLED	PTE7	KB11_P7	TCLK2			FTM1_CH1	CAN0_TX	
7	PTH2	DISABLED	PTH2	KB11_P26	BUSOUT			FTM1_CH0	CAN0_RX	
8	VDD	VDD								VDD
9	VDDA	VDDA							VREFH	VDDA
10	VREFH	VREFH								VREFH
11	VREFL	VREFL								VREFL
12	VSS/ VSSA	VSS/VSSA							VSSA	VSS
13	PTB7	EXTAL	PTB7	KBIO_P15	I2C0_SCL					EXTAL
14	PTB6	XTAL	PTB6	KBIO_P14	I2C0_SDA					XTAL
15	PT14	DISABLED	PT14		IRQ					
16	PT11	DISABLED	PT11		IRQ	UART2_TX				
17	PT10	DISABLED	PT10		IRQ	UART2_RX				
18	PTH1	DISABLED	PTH1	KB11_P25	FTM2_CH1					
19	PTH0	DISABLED	PTH0	KB11_P24	FTM2_CH0					
20	PTE6	DISABLED	PTE6	KB11_P6						
21	PTE5	DISABLED	PTE5	KB11_P5						
22	PTB5	DISABLED	PTB5	KBIO_P13	FTM2_CH5	SPI0_PCS	ACMP1_OUT			
23	PTB4	NMI_b	PTB4	KBIO_P12	FTM2_CH4	SPI0_MISO	ACMP1_IN2	NMI_b		
24	PTC3	ADC0_SE11	PTC3	KBIO_P19	FTM2_CH3		ADC0_SE11			
25	PTC2	ADC0_SE10	PTC2	KBIO_P18	FTM2_CH2		ADC0_SE10			
26	PTD7	DISABLED	PTD7	KBIO_P31	UART2_TX					
27	PTD6	DISABLED	PTD6	KBIO_P30	UART2_RX					
28	PTD5	DISABLED	PTD5	KBIO_P29	PWT_IN0					
29	PT16	DISABLED	PT16	IRQ						
30	PT15	DISABLED	PT15	IRQ						
31	PTC1	ADC0_SE9	PTC1	KBIO_P17	FTM2_CH1		ADC0_SE9			
32	PTC0	ADC0_SE8	PTC0	KBIO_P16	FTM2_CH0		ADC0_SE8			
33	PTH4	DISABLED	PTH4	KB11_P28	I2C1_SCL					
34	PTH3	DISABLED	PTH3	KB11_P27	I2C1_SDA					
35	PTF7	ADC0_SE15	PTF7	KB11_P15			ADC0_SE15			
36	PTF6	ADC0_SE14	PTF6	KB11_P14			ADC0_SE14			
37	PTF5	ADC0_SE13	PTF5	KB11_P13			ADC0_SE13			
38	PTF4	ADC0_SE12	PTF4	KB11_P12			ADC0_SE12			
39	PTB3	ADC0_SE7	PTB3	KBIO_P11	SPI0_MOSI	FTM0_CH1	ADC0_SE7			
40	PTB2	ADC0_SE6	PTB2	KBIO_P10	SPI0_SCK	FTM0_CH0	ADC0_SE6			
41	PTB1	ADC0_SE5	PTB1	KBIO_P9	UART0_TX		ADC0_SE5			
42	PTB0	ADC0_SE4	PTB0	KBIO_P8	UART0_RX	PWT_IN1	ADC0_SE4			
43	PTF3	DISABLED	PTF3	KB11_P11	UART1_TX					
44	PTF2	DISABLED	PTF2	KB11_P10	UART1_RX					
45	PTA7	ADC0_SE3	PTA7	KBIO_P7	FTM2_FLT2	ACMP1_IN1	ADC0_SE3			
46	PTA6	ADC0_SE2	PTA6	KBIO_P6	FTM2_FLT1	ACMP1_IN0	ADC0_SE2			
47	PTE4	DISABLED	PTE4	KB11_P4						
48	VSS	VSS								VSS

49	VDD	VDD							VDD
50	PTG7	DISABLED	PTG7	KB11_P23	FTM2_CH5	SPI1_PCS			
51	PTG6	DISABLED	PTG6	KB11_P22	FTM2_CH4	SPI1_MISO			
52	PTG5	DISABLED	PTG5	KB11_P21	FTM2_CH3	SPI1_MOSI			
53	PTG4	DISABLED	PTG4	KB11_P20	FTM2_CH2	SPI1_SCK			
54	PTF1	DISABLED	PTF1	KB11_P9	FTM2_CH1				
55	PTF0	DISABLED	PTF0	KB11_P8	FTM2_CH0				
56	PTD4	DISABLED	PTD4	KB10_P28					
57	PTD3	DISABLED	PTD3	KB10_P27	SPI1_PCS				
58	PTD2	DISABLED	PTD2	KB10_P26	SPI1_MISO				
59	PTA3	DISABLED	PTA3	KB10_P3	UART0_TX	I2C0_SCL			
60	PTA2	DISABLED	PTA2	KB10_P2	UART0_RX	I2C0_SDA			
61	PTA1	ADCO_SE1	PTA1	KB10_P1	FTM0_CH1	I2C0_4WSD AOUT	ACMP0_IN1	ADCO_SE1	
62	PTA0	ADCO_SE0	PTA0	KB10_P0	FTM0_CH0	I2C0_4WSC LOUT	ACMP0_IN0	ADCO_SE0	
63	PTC7	DISABLED	PTC7	KB10_P23	UART1_TX			CAN0_TX	
64	PTC6	DISABLED	PTC6	KB10_P22	UART1_RX			CAN0_RX	
65	PT13	DISABLED	PT13	IRQ					
66	PT12	DISABLED	PT12	IRQ					
67	PTE3	DISABLED	PTE3	KB11_P3	SPI0_PCS				
68	PTE2	DISABLED	PTE2	KB11_P2	SPI0_MISO	PWT_IN0			
69	VSS	VSS							VSS
70	VDD	VDD							VDD
71	PTG3	DISABLED	PTG3	KB11_P19					
72	PTG2	DISABLED	PTG2	KB11_P18					
73	PTG1	DISABLED	PTG1	KB11_P17					
74	PTG0	DISABLED	PTG0	KB11_P16					
75	PTE1	DISABLED	PTE1	KB11_P1	SPI0_MOSI		I2C1_SCL		
76	PTE0	DISABLED	PTE0	KB11_P0	SPI0_SCK	TCLK1	I2C1_SDA		
77	PTC5	DISABLED	PTC5	KB10_P21		FTM1_CH1		RTC_CLKOUT	
78	PTC4	SWD_CLK	PTC4	KB10_P20	RTC_CLKOUT	FTM1_CH0	ACMP0_IN2	SWD_CLK	
79	PTA5	RESET_b	PTA5	KB10_P5	IRQ	TCLK0	RESET_b		
80	PTA4	SWD_DIO	PTA4	KB10_P4		ACMP0_OUT	SWD_DIO		

## 附录C KDS集成开发环境简明使用方法

### C.1 KDS下载、安装与配置

#### C.1.1 KDS的下载与安装

KDS (Kinetis Design Studio) 是飞思卡尔于 2014 年开始推出的面向 ARM Cortex-M 内核的 Kinetis 系列微控制器的嵌入式集成开发环境, 下载的安装文件为“KDS-v3.0.0.exe” (更新日期为 2015 年 5 月 6 日), 安装包大小为 658MB。KDS 具有编辑、编译、下载程序、调试等功能。KDS 下载地址为: [http://www.freescale.com/zh-Hans/webapp/sps/site/prod\\_summary.jsp?code=KDS\\_IDE&nodeId=0152101E8C1EB4&fsp=1&tab=Design\\_Tools\\_Tab#nogo](http://www.freescale.com/zh-Hans/webapp/sps/site/prod_summary.jsp?code=KDS_IDE&nodeId=0152101E8C1EB4&fsp=1&tab=Design_Tools_Tab#nogo)。KDS 支持的操作系统有: Windows 7/8 (32 and 64-bit) 及 Linux (Ubuntu、Redhat、Centos)。在 WindowsXP 系统下也能安装, 但可能会出现意想不到的问题, 一般推荐在 Windows7 系统下安装 KDS。

安装方法: 运行 KDS 安装文件 “Kinetis Design Studio installer for Microsoft Windows 3.0.0.exe”, 根据提示安装即可。

KDS 安装好后, 根据用户使用的写入器型号, 再安装有关写入器驱动软件。

#### C.1.2 KDS的配置

##### 1. 增加.hex和.map文件

KDS 在编译过程中默认建立.elf 文件, 如果需要增加.hex 和.map 文件, 选中工程, 点击菜单栏中 Project->Properties 进入工程属性设置, 进入 C/C++Build-->Settings-->Toolchains 选项卡, 勾选 Create flash image 及 Create extended listing, 点击 OK。此时工程在编译过程中就会在 DEBUG 中出现.hex 和.map 文件。具体操作见图 A-1 所示。

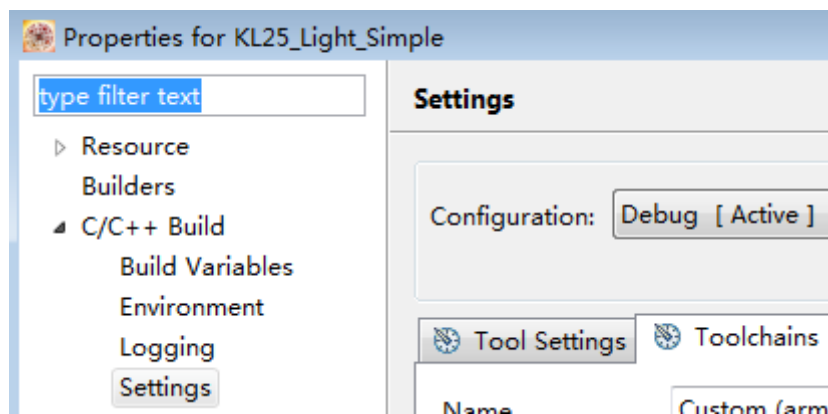


图 A-1 Toolchains 所在位置图

设置 Create flash image 及 Create extended listing, 见图 A-2 所示。

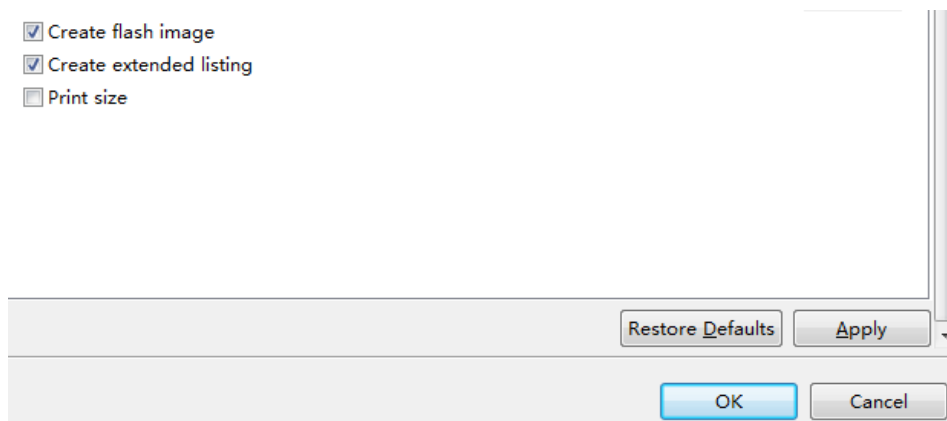


图 A-2 增加.hex 和.map 文件设置图

.hex 文件是由符合 Intel HEX 文件格式的文本构成的 ASCII 文本文件。具体知识见《嵌入式技术基础与实践（第三版）—ARM Cortex-M0+ Kinetis L 系列微控制器》中第 87 页。

.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。

## 2. 增加或更改文件的查看方式

KDS 中默认的文件打开方式为 C/C++ Editor，只能识别.c/h/.s 等文件，对于工程中的.ld 等文件不能识别，因此需要更改 KDS 中对某种文件的打开方式，可以点击菜单栏中的 Window->Preferences，点击 General->Editors->File Associations，在右侧选项卡中的 File types 中选择需要更改的文件类型（若没有，可点击右侧的 Add 按钮手动添加），在下方 Associated editors 中会显示此类文件当前的打开方式，点击右侧的 Add 按钮添加新的打开方式，选中它后点击右侧的 Default 按钮可将其设置为默认打开方式，点击 OK 即可完成修改。

例如：需打开链接文件（.ld 文件），但是无法查看文件中代码，可以点击 Window->Preferences->General->Editors->File Associations，在右侧选项卡中的 File types 中选择\*.ld 文件类型，在下方 Associated editors 中点击 Add，选择“C/C++ Editor”打开方式，点击 OK 即可查看文件中代码。

## 3. 更改字体属性

若需改变文本编辑区文字的字体属性，可以点击菜单栏中的 Window->Preferences，点击 General->Appearance->Colors and Fonts，在右侧选项卡中的 Colors and Fonts 里选择 Basic->Text Font，点击 Edit 按钮即可进入字体属性界面。设置字体属性见图 A-3 所示。

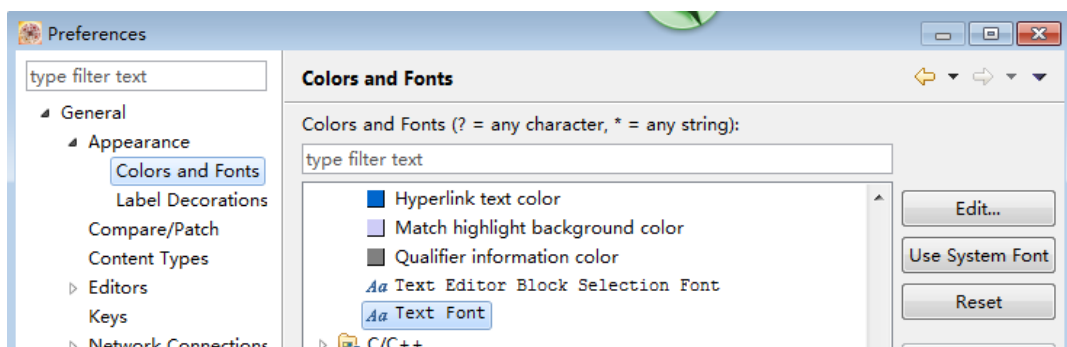


图 A-3 字体属性设置图

点击  即可修改文字属性。

#### 4. 取消自动编译

编译时勾选当前工程，在 KDS 中 Project 菜单下的 Build Automatically 项前的 √ 点击后取消，取消自动编译（建议的配置方式），如图 A-4 所示。

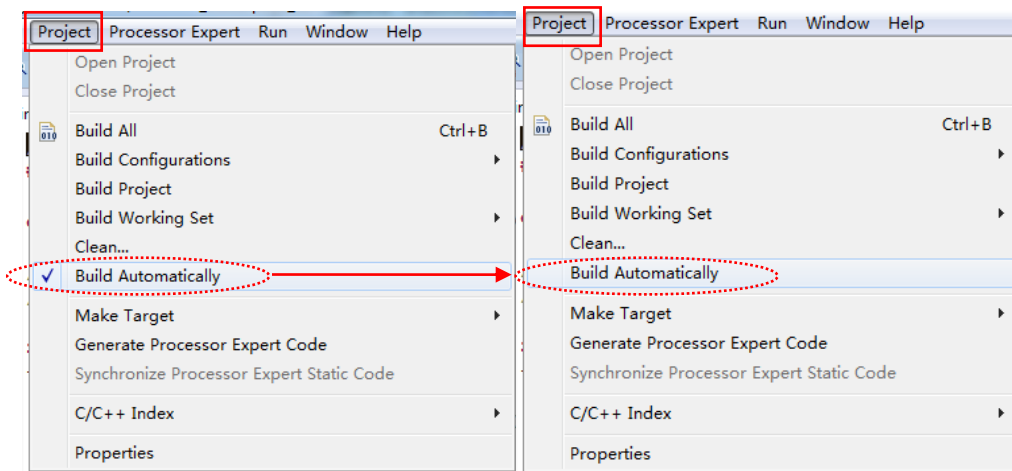


图 A-4 取消自动编译

## C.2 KDS 简明使用方法

### C.2.1 导入现有工程到开发环境中

#### 1. 什么是 KDS 工程目录？

如图 A-5 “MAPS\_K64-uart\_20150205” 就是一个 KDS 工程目录。注：Debug 文件夹是工程进行构建后自动生成的文件夹。工程框架见图 A-5 所示：

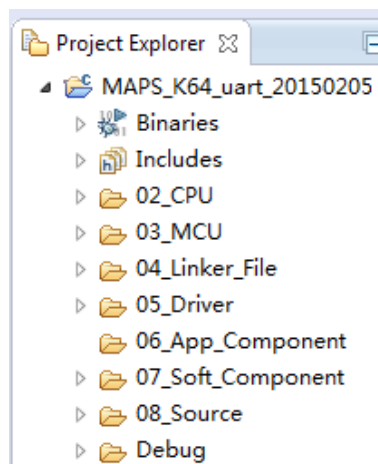


图 A-5 KDS 工程框架

#### 2. 什么是 KDS 工程面板

如图 A-6 所示，是 KDS 的工程面板。若没有，可以通过点击菜单栏中 Window->Show View->Project Explorer，显示出 KDS 的工程面板。未导入工程的 KDS 工程面板如图 A-6 所示。



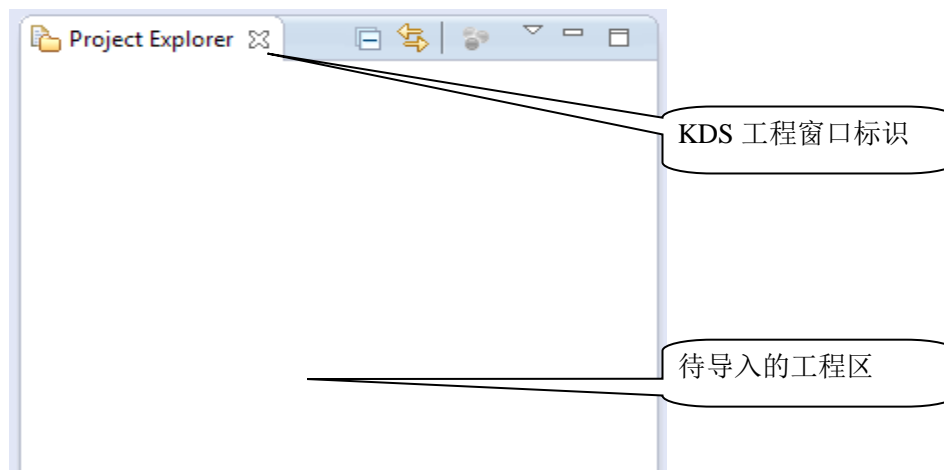


图 A-6 KDS 的工程面板

### 3. 工程导入方法

在 KDS 中需要打开工程时，选择菜单栏中 **File->Import**，在弹出的对话框中选择 **General->Existing Projects into Workspace**，点击 **Next**，接着点选 **Select root directory**，点击右侧的 **Browse**，选择需要打开的工程文件夹，点击 **Finish** 完成，导入过程如图 A-7 所示：

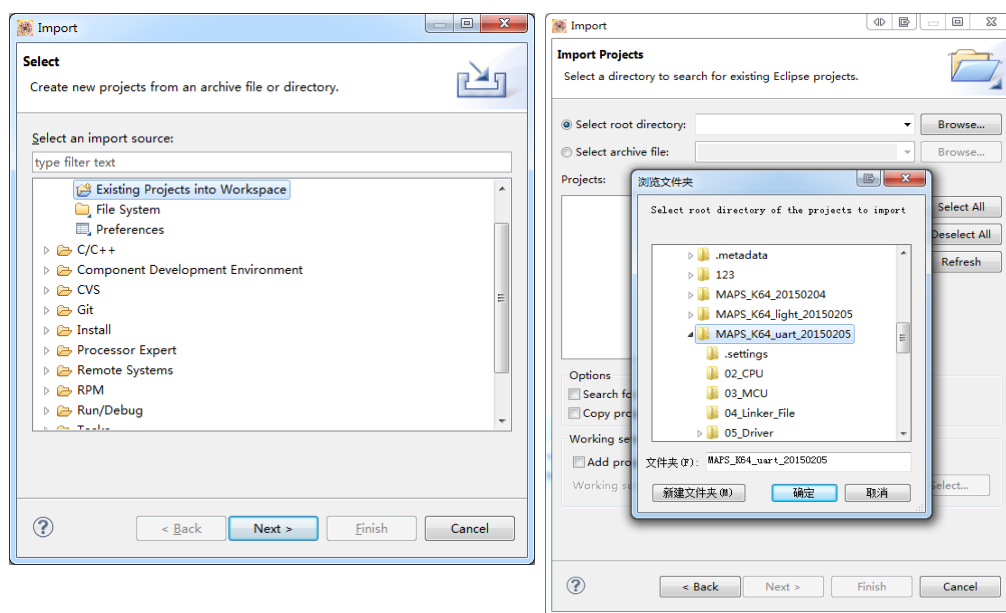


图 A-7 导入工程流程


### C.2.2 编译与链接工程—产生可执行的机器码

编译前，需在左侧的工程目录视图中点选需要编译的工程，然后在菜单栏中选择 **Project->Build Project** 或点击工具栏上的 图标，即可编译所选工程。

若需清理工程，则在菜单栏中选择 **Project->Clean** 即可。若需在清理工程后自动编译工程，则先需要将 **Project->Build Automatically** 的勾去掉，此时在清理工程后会自动对工程进行重新编译。建议使用此种方式对工程进行编译。

具体的编译过程信息可以查看下方的 **Console** 窗口。编译过程结束后可以在下方的

Problems 窗口中查看编译警告及错误信息。

在左侧工程栏的 DEBUG 中，工程默认编译生成的机器码为“.elf”文件，如“ MAPS\_K64.elf - [arm/le]”，若需要生成其他机器码，如.hex 文件及.lst 文件，可以选择工程，点击菜单栏中 Project->Properties 进入工程属性设置，进入 C/C++Build-->Settings-->Toolchains 选项卡，勾选 Create flash image 及 Create extended listing，点击 OK。具体过程见附录 C.1.2 下“1. 增加.hex 和.map 文件”。

若 Problems 中没有 ERRORS，则编译通过。可以进行程序的运行或调试。

## C.2.3 直接下载机器码到目标板—运行

### 1. 使用独立写入软件下载机器码

在存在机器码的情况下，便可使用独立写入软件将机器码直接下载到目标板，无需使用 KDS 等开发环境，这为开发人员带来便利，同时节省时间、提高效率。

对于支持 SWD 调试协议的目标板（如：KEA128、K64），可以使用 JFlash 独立写入软件，具体操作方法详见附录 D。

### 2. 在KDS下直接下载机器码

#### 1) 导入样例工程

在 KDS 中需要打开工程时，选择菜单栏中 File->Import，在弹出的对话框中选择 General->Existing Projects into Workspace，点击 Next，接着点选 Select root directory，点击右侧的 Browse，选择需要打开的工程文件夹，点击 Finish 完成。

#### 2) 编译工程

编译前，需在左侧的工程目录视图中选择需要编译的工程，然后在菜单栏中选择 Project->Build Project 或点击工具栏上的图标，即可编译所选工程。

若需清理工程，则在菜单栏中选择 Project->Clean Project 即可。若需在清理工程后自动编译工程，则先需要将 Project->Build Automatically 的勾去掉，此时在清理工程后会自动对工程进行重新编译。

具体的编译过程信息可以查看下方的 Console 窗口。编译过程结束后可以在下方的 Problems 窗口中查看编译警告及错误信息。

#### 3) 烧录配置

点击菜单栏下方工具栏内的黄色闪电型图标会进入烧录界面。

首次使用时，需双击左侧调试类型窗口中的 GDB SEGGER J-Link Debugging，会在其下方自动新建一个以当前工程命名的 JLink Programmer 连接，单击其进入烧录配置界面。

接着，在烧录配置界面中选择 Main 选项卡，在 Project 中选择烧录文件所在工程，在 C/C++ Application 中选择需要烧录的文件(.elf 或.hex)，可以点击 Search Project（相对路径，只能选择.elf 文件）或 Browse（绝对路径，.elf、.hex 均可）来选取烧录文件。

然后，进入烧录配置界面中的 Debugger 选项卡，填写目标板芯片名称 Device Name。具体的目标板芯片名称可以点击 Device Name 后面的 Supported device names 链接，进入 segger 的官方网站查看。芯片的 Device Name 如下：

K64: MK64FN1M0xxx12

K60: MK60DN512xxx10

Commands 框内默认为 set mem inaccessible-by-default off，保留并回车输入 set backtrace limit 20，该语句可以避免断点设置在中断内时的调试问题。

**Commands:** `set mem inaccessible-by-default off`  
`set backtrace limit 20`

同时，取消勾选 Startup 选项卡中的 Enable SWO。

最后确认好写入器与目标板已正确连接且目标板通电，点击 Flash 按钮即可开始向目标芯片烧录程序。

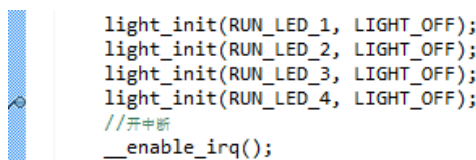
## C.2.4 在KDS下进行跟踪调试

### 1. 设置和取消断点

设置断点可以让程序直接运行到断点处，可以查看此时程序的状态，如变量值、寄存器状态等，一般用于程序快速排错和深入理解程序执行流程等。

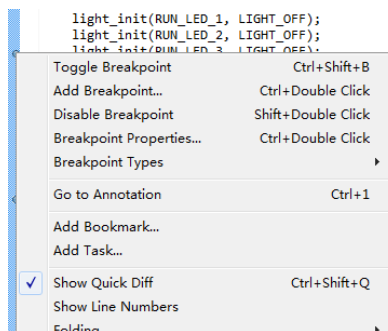
在 KDS 下设置断点有两种方式，使用时任选其一即可。注意：断点一般设置在源文件中，如.c 文件和.s 文件。

方法一：打开需要设置断点的文件，双击需要设置断点语句的左侧蓝色区域，双击后便会出现一个蓝色原点，表示已经在该语句上设置了断点。再次双击便可取消此处断点。



```
light_init(RUN_LED_1, LIGHT_OFF);
light_init(RUN_LED_2, LIGHT_OFF);
light_init(RUN_LED_3, LIGHT_OFF);
light_init(RUN_LED_4, LIGHT_OFF);
//开中断
__enable_irq();
```

方法二：同样打开需要设置断点的文件，找到需要设置断点的语句，在该语句左侧蓝色区域右击鼠标，选择“Toggle Breakpoint”，便可在该语句上设置断点。再次选择这个选项，便可取消此处断点。







### 2. 调试配置

在 KDS 下直接下载机器码时的烧录配置即为调试配置，因此无需再另行配置，详见附录 C.2.3 中的“3) 烧录配置”。

### 3. 执行调试

确认好写入器与目标板已正确连接且目标板通电，点击调试按钮“ ”下的“Debug

Configurations...”选项，即可进入单步调试主界面，如图 A-8 所示。按 F5  或者 F6  开始单步向前调试，F5 为进入函数体调试，F6 为跳过函数体调试。为了节省单步调试的时间，可使用 F6 单步调试快捷键。

点击按钮“ ”能够重新启动调试，点击按钮“ ”能够中止当前调试，点击按钮

“” 可以从当前程序执行的位置直接跳转到下一个断点处。

同时，单步调试主界面右上方的视图显示调试过程中各变量、寄存器的值等。

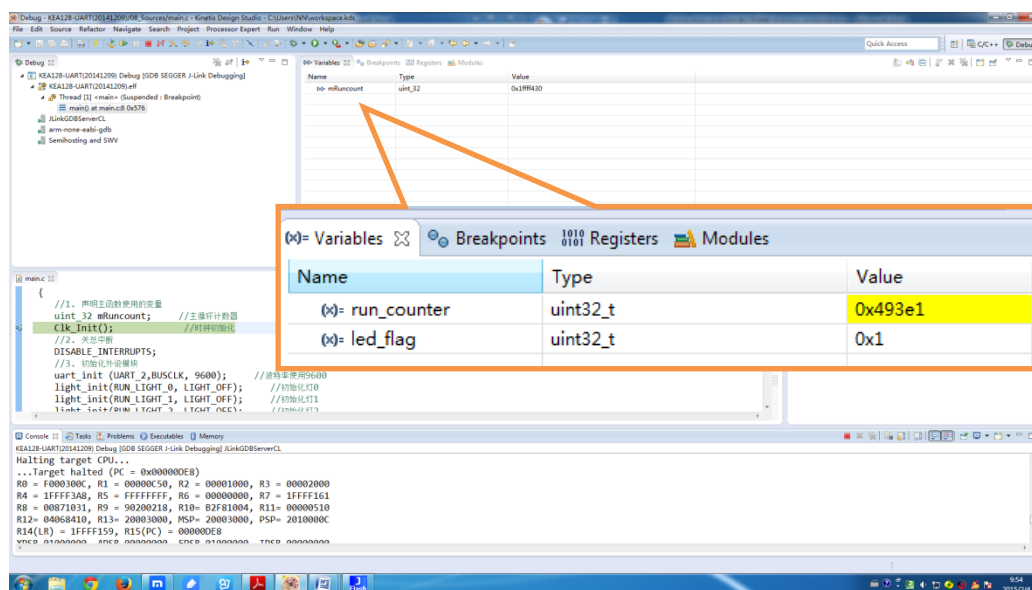


图 A-8 串口单步调试主界面

## C.2.5 KDS的常用基本操作

### 1. 变量与函数声明的定位

在文本编辑区里，将鼠标指针悬停在某个宏常量、变量或是函数上时，会弹出文本框显示对应的宏展开、变量或函数声明。

若想跳转到相应的声明处，可以右键单击相应宏常量、变量或是函数，在弹出的菜单中选择 **Open Declaration**（或左键单击文本，按 F3 快捷键；也可以按住 Ctrl 键不放，左键单击相应文本）追踪该变量或函数、宏常量的上层定义位置，继续该操作直到最早定义位置（有些定义是存放到 .ld 文件中，需按照前面附录 C.1.2 中“1. 增加或更改文件的查看方式”设置关联 .ld 文件或其他文件。此外，可以用 Alt+- 及 Alt+> 快捷键来后退或前进到前一个或后一个光标所在处。由此可以查看函数或变量的属性，用于更好的理解程序。

有以下特殊情况无法索引到变量或函数声明，可使用“搜索与替换”中“在所有打开的工程文件中搜索/替换关键字”的方法进行搜索定位。

- （1）变量或函数声明所在的文件不属于当前工程。
- （2）变量或函数声明所在的文件是链接文件或汇编文件。

### 2. 搜索与替换

若只需在单个文件中搜索/替换关键字，可以点击菜单栏中的 **Edit->Find/Replace**（快捷键 Ctrl+F）。

若需要在所有打开的工程文件中搜索/替换关键字，可以点击菜单栏中的 **Search->File**（也可使用快捷键 Ctrl+H，然后点击 **File Search** 选项卡），在 **Containing text** 里填入搜索关键字，在 **File name patterns** 里填入需要搜索的文件类型（一般填写 \*.\*），点击 **Search** 或 **Replace** 来进行搜索与替换。搜索结果会在屏幕下方的 **Search** 窗口中显示。



### 3. 添加文件/文件夹

在工程中添加软件构件时选择其中一种方法即可。

方法一：点击工程菜单右键选择“Import”，在对话框中选择“File System”，点击“Next”，选择需要添加的文件/文件夹路径，确定后选中需要选中的文件/文件夹，如果添加的是文件夹，下方 Options 选项卡中要勾选“Create top-level folder”，点击 Finish 即可；如果添加的是文件，直接点击 Finish 即可。

方法二：将要添加的文件复制到要放入的工程目录下，然后在 KDS 下，右击工程名点击“Refresh”，文件便会自动加进工程。推荐使用第二种方法添加一个文件/文件夹。

### 4. 添加文件夹引用

虽然添加文件夹将文件夹加进了工程，KDS 并未将文件夹“引用”，需要添加工程应用，这样头文件才能将其包含至工程。在 KDS 下右击工程名点击“Properties”，在左边栏点击“C/C++ Build”下选择“Settings”。在“Setting”右边的选项卡内选择“Tool Settings”下的“Cross ARM C Compiler”中的“Includes”项，然后在“Include Paths (-I)”单击  添加工程路径单击“OK”便可。单击  可以删除某一路径引用。

### 5. 设置/取消“鼠标悬停提示”

“鼠标悬停提示”的优点是，当鼠标悬停时，会有相应的提示；“鼠标悬停提示”的缺点是，不希望提示时，干扰视线，影响阅读程序的效率。在 KDS 环境下，可以通过以下方法设置/取消“鼠标悬停提示”：

Window->Preferences->C/C++->Editor->Hovers，将 Combined Hover 前面的对号设置或去掉。

### 6. KDS组件更新

如果 KDS 不支持当前连接的目标板，则需要对 KDS 开发环境进行组件更新，以获得更多的芯片型号支持。

在菜单栏下单击“Help”的“Check for Updates”，如果有新的更新，则选择 Help 菜单下的“Install New Software”，在对话框中选择要升级文件即可。注：组件更新不等于 KDS 软件版本更新，如果飞思卡尔官网有新的 KDS 版本，仍需在官网上进行下载安装，同时可以卸载旧版本。

## C.3 常见问题处理

1) 在编译程时会提示“writing to APSR without specifying a bitmask is deprecated”错误

答：这个提示在 Console 窗口中为警告，但在 Problems 窗口中变为了错误，这是 KDS 的 bug，可以忽略，不影响最终生成.elf 及.hex 文件。

2) 在打开工程时，选择好工程所在文件夹后无法点击 Finish 按钮导入工程，工程导入向导中的 Projects 栏里虽然出现了工程所在路径，但却是灰色的无法勾选

答：这是因为在 KDS 中已经打开了同名工程。可以在工程浏览器里将打开的同名工程关闭，或是将已打开的工程重命名（右键点击工程，选择 Rename），就可以将另外一个同名工程导入了。

3) 将核心板通过写入器端口用 USB 线连接到电脑后没有任何反应，电脑右下方也没有弹出“发现新硬件”的提示

答：右击“我的电脑→管理”，显示“计算机管理”视图，找到“系统工具→设备管理

器”，其中有带问号的疑问设备，如图 A-9 所示。



图 A-9 待安装驱动界面

右击选择“更新驱动程序”，系统弹出“发现新硬件”的提示，并弹出“找到新的硬件向导”对话框，选择“从列表或指定位置安装（高级）”选项。如图 A-10 所示。

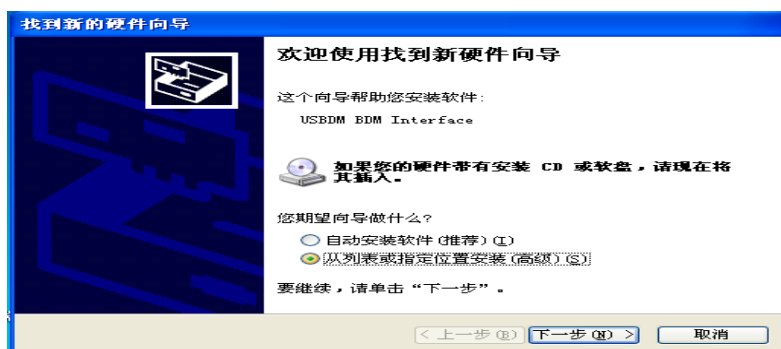


图 A-10 发现 USBDM

单击“**下一步 (N) >**”，如图 A-11 所示。

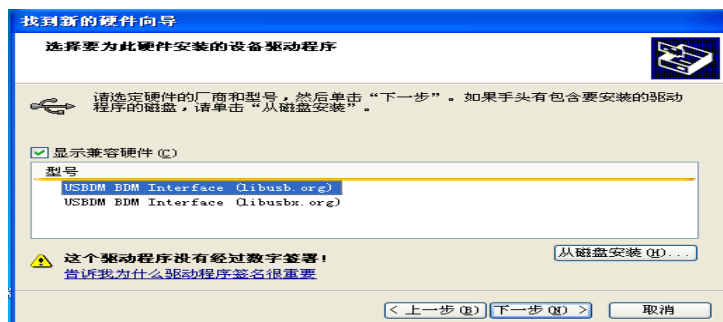


图 A-11 选择要安装的驱动

选择“**从磁盘安装 (D) ...**”，然后选择驱动程序的路径，即“C:\Program Files\pgo\USBDM Drivers 1.0.1\Drivers\BDM\_Driver”文件夹（XP 系统）。“C:\Program Files (X86)\pgo\USBDM Drivers 1.0.1\Drivers\BDM\_Driver”文件夹（WIN7 系统）。如图 A-12 所示。



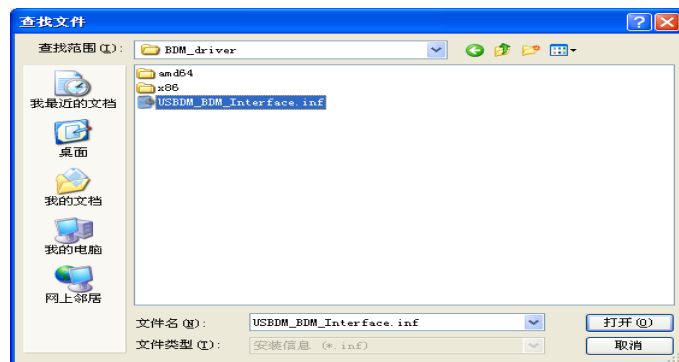


图 A-12 驱动程序路径

单击“打开(O)”按钮，进行驱动安装，最后单击“完成”将完成 USB 驱动的安装。驱动程序是 USBDM\_BDM\_Interface.inf。

若在 64 位 WIN7 系统下，安装 USBDM\_Drivers\_1\_0\_1\_Win\_x64.msi 后，将核心板的写入器端口与 PC 机的 USB 口相连，系统会自动安装好驱动。视不同系统而定。

**4) 系统中已经存在某些驱动文件，导致冲突，以上处理方法无效，USBDM 驱动安装不成功**

答：首先，确定驱动安装在系统中的文件名。找一台已经安装好 USBDM 驱动的机器，在设备管理器中找到 USBDM 的设备条目，右击选择“属性”，选择“驱动程序→驱动程序详细信息”，如图 A-13 所示。



图 A-13 USBDM 驱动文件信息框

从 USBDM 驱动文件信息框可知道驱动程序在系统中的文件为 WdCoInstaller01011.dll，在操作系统的“Windows\system32”目录下搜索该文件，得到结果如图 A-14 所示。

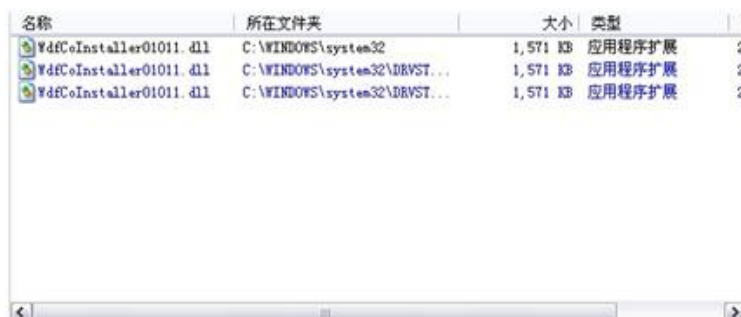


图 A-14 USBDM 驱动文件的位置

删除这些文件（但最好从 WINDOWS 系统的“开始→控制面板→卸载程序”，打开卸载程序窗口，找到 USBM Driver 驱动程序，将其卸载。有的系统这些文件是难以直接删除的。），重新安装 USBDM 的驱动：[USBDM\\_Drivers\\_1\\_0\\_1\\_Win\\_x32.msi](#)（32 位操作系统下使用）

或 USBDM\_Drivers\_1\_0\_1\_Win\_x64.msi (64 位操作系统下使用)。驱动安装完成后, 重启计算机, 问题得以解决。

**5) 有时 USB 接头插入计算机 USB 接口时无反应**

答: 可能原因是笔记本电脑或台式机的前置 USB 口供电不足, 换其他 USB 口重试。

**6) 编程下载对话框中的选项出现异常, 无法下载程序**

答: 断开 BDM 与 PC 机的连接, 将当前工程关闭。再重新连接 BDM, 打开工程, 一般可以消除异常执行正常的下载操作。注意此步骤是在写入器固件程序升级的基础上执行的。

**7) 工程无误, 但编译时提示错误找不到路径包含头文件路径**

答: 可能是工程使用了中文目录, 应保证工程的路径为全英文字符。

**8) 导入现有工程后直接编译不通过**

答: 可以清除当前工程之前的生成文件, 重新创建生成文件即可。

**9) 打开工程后看不到工程目录结构, 或者误删了工程框栏**

答: 选择 “Window” → “Show View” → “Project Explorer”, 可在左侧面板显示工程目录结构。

**10) 编程下载对话框中的选项出现异常, 无法下载程序**

答: 断开 BDM 与 PC 机的连接, 将当前工程关闭。再重新连接 BDM, 打开工程, 一般可以消除异常执行正常的下载操作。注意此步骤是在写入器固件程序升级的基础上执行的。

**11) 打开 KDS 环境时出现 “Failed to create the Java Virtual Machine” (如图 A-15 所示) 或者 “Out of memory” 问题**

答:



图 A-15 错误实例

首先关闭如 360 等杀毒软件, 再进行尝试。如果还是出现上述问题, 则需要重新设置开发环境的最高堆栈大小, 找到 KDS 安装目录\eclipse\kinetis-designstudio.ini, 打开 ini 文件, 默认是-Xmx512m, 意味着为 KDS 分配的最高堆栈大小为 512MB, 若出现 “Failed to create the Java Virtual Machine” 问题, 则降低该值 (如 256MB), 若出现 “Out of memory”, 则提高该值 (如 1024MB)。

**12) KDS-2.0 下的工程向 KDS-3.0 移植, 编译出错**

答: KDS-3.0 环境下, 在该工程名下右击选择 “Properties”。选择 “C/C++ Build” → “Setting” → “Cross ARM C++ Linker” → “Miscellaneous”, 做如图 A-16 所示的配置。



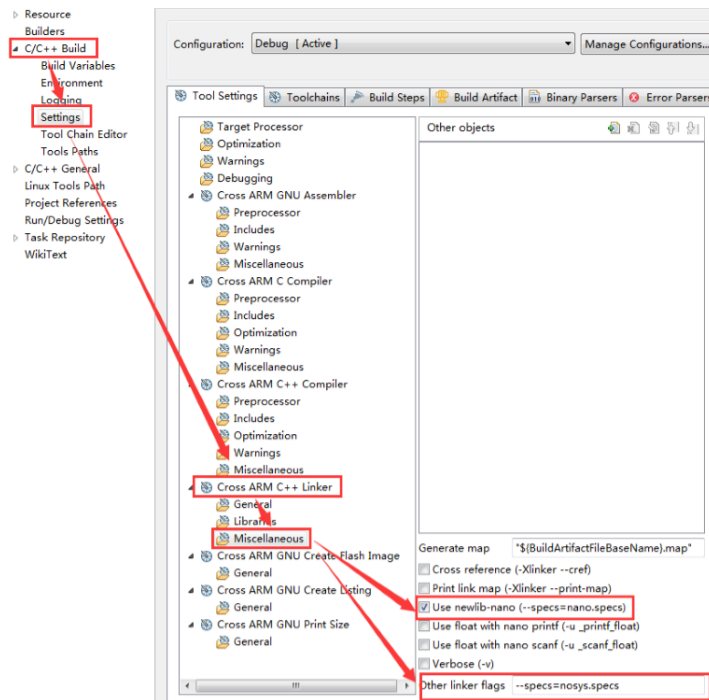


图 A-16 配置示意图

13) 修改工程名完成后, 重新编译生成的目标文件的名称仍是修改前的名称

答: KDS-3.0 环境下, 在该工程名下右击选择“Properties”。选择“C/C++ Build”→“Setting”→“Build Artifact”, 修改“Artifact name”为“\${ProjName}”。如图 A-17 所示。

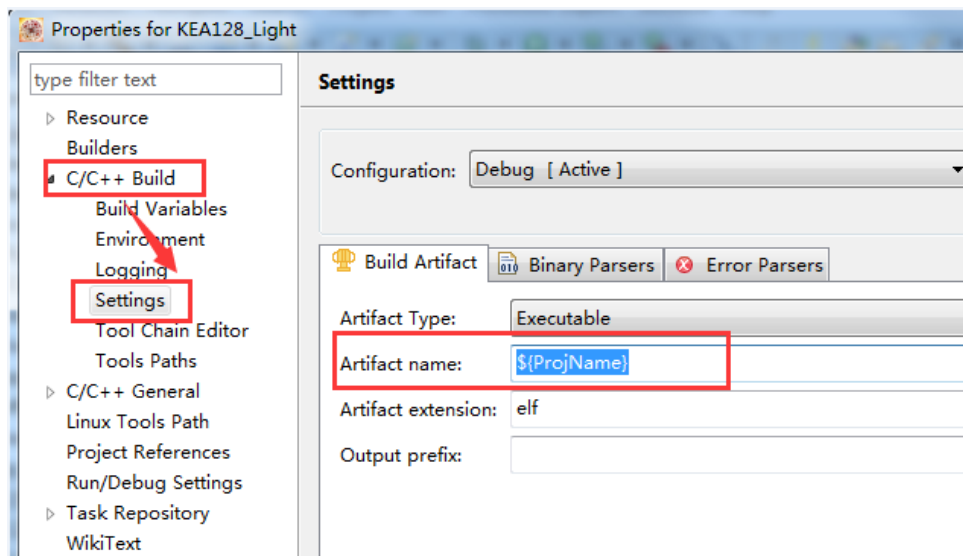


图 A-17 配置示意图

答: KDS-3.0 环境下, 在该工程名下右击选择“Properties”。选择“C/C++ Build”→“Setting”→“Build Artifact”, 修改“Artifact name”为“\${ProjName}”。如图 A-17 所示。

## 附录D SWD-Programmer简明使用方法

### D.1 相关开发工具的安装

#### D.1.1 所需的工具软件清单

使用 SWD-Programmer 时，可能会用到下述软件：

(1) 集成开发环境 KDS (Kinetis Design Studio)：具有编辑、编译、下载程序、调试等功能；

(2) JLink 驱动程序：JFlash 独立写入软件需要使用 JLink，因而需要事先安装 JLink 驱动；

(3) JFlash 独立写入软件：JFlash.exe，用于将 KDS 环境生成的 .hex 文件通过 SWD-Programmer 烧录至目标板（如：KW01、KL25、KL26、K60、K64 等）。

#### D.1.2 软件安装过程

##### 1. 集成开发环境KDS的安装

KDS 相关的安装与配置，请参考附录 A。

##### 2. JLink驱动程序的安装

将 SWD-Programmer 插入 PC 机的 USB 口，PC 机会自动安装 JLink 驱动程序，安装成功之后，会在 PC 机的“设备管理器”→“通用串行总线控制器”目录下增加一个“JLink driver”的图标。

##### 3. JFlash的安装

在 KDS 安装时，会自动安装 JFlash，可以在安装目录下的 segger 目录中找到，没有安装 KDS 的用户，请自行到官网上 <https://www.segger.com/jlink-software.html> 下载最新版本。


### D.2 使用JFlash独立写入软件进行烧录

#### D.2.1 硬件连接


SWD-Programmer 的 USB 端接到 PC 机，另一端接到目标板上。SWD-Programmer 的四个脚“SWDIO、GND、SWDCLK、5V”分别接在板上对应的“TMS/DIO、GND、TCK/CLK、5V”四个口中，即棕线接底板的 TCK/CLK、红线接底板的 GND、橙线接底板的 TMS/DIO、单线接底板的 5V。可参见对应目标板硬件最小系统原理图中的 SWD 调试接口图进行连接。

注意：指示灯常亮表示 SWD-Programmer 正常运行。若指示灯存在其他状态请参考“D.3 常见问题处理”进行解决。

#### D.2.2 烧录步骤

(1) 打开 JFlash 软件，设置模板。首次使用时  后面的内容为“Other”，所以需要设置模板。

## (2) 设置模板

选择  **Create a new project.** , 然后点击菜单栏上 Options->Project settings, Target Interface 选择 SWD, CPU 栏的 Device 选择相应的芯片, 见图 B-1。下次打开 JFlash 可直接选 Open exiting project 中的模板。

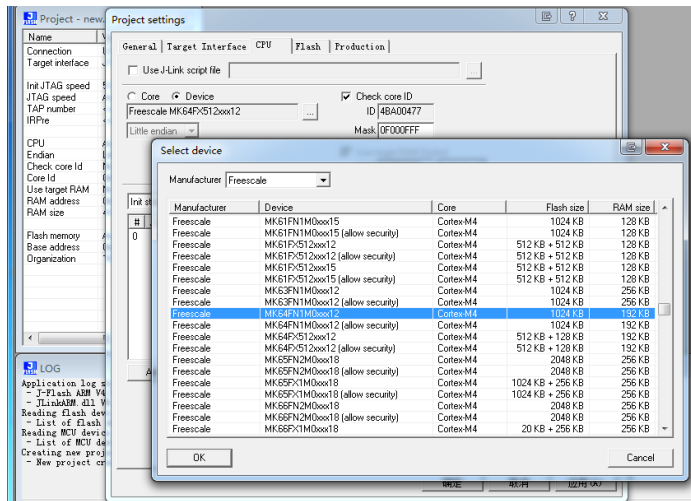


图 B-1 设置模板

## (3) 配置完成后, 载入工程生成的.hex 文件

“File”——“Open data file”, 选择.hex 文件, 点击“打开”, 见图 B-2 和图 B-3。若没有.hex 文件请打开 KDS, 导入工程后选中工程, 点击菜单栏中 Project->Properties 进入工程属性设置, 进入 C/C++Build-->Settings-->Toolchains 选项卡, 勾选 Create flash image 及 Create extended listing, 点击 OK, 便可生成.hex 文件。

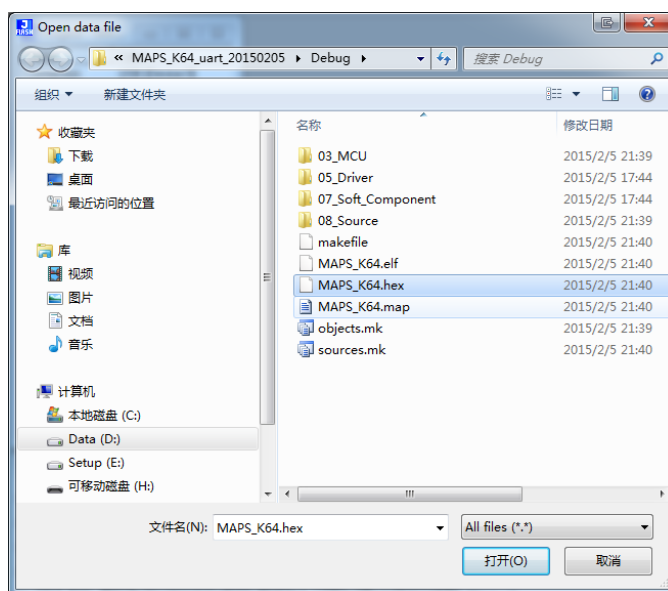


图 B-2 载入十六进制文件

打开之后, 载入成功之后的界面见图 B-3。

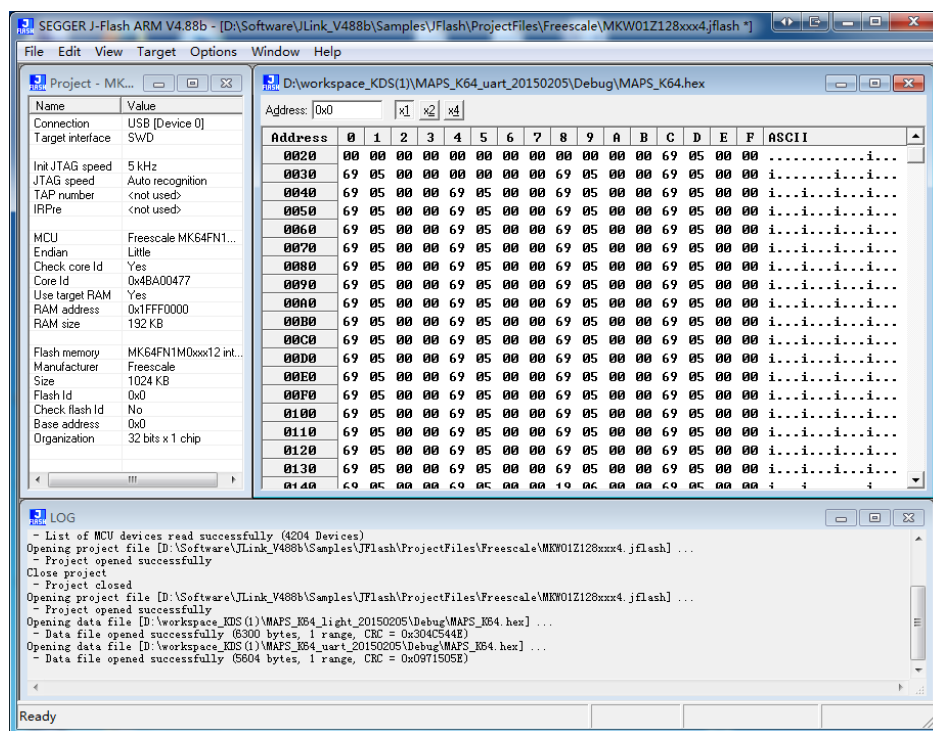


图 B-3 载入成功界面

#### (4) 烧录程序

点击菜单栏中“Target”——“Connect”，连接成功后，再点击菜单栏的“Target”——“Auto”，自动烧录程序。烧录结束后会弹出一个对话框表示烧录成功，见图 B-4。需要注意的是，使用 SWD-Programmer 只能烧录程序，不能对目标板进行在线调试，若需调试请使用 KDS，详见附录 A。

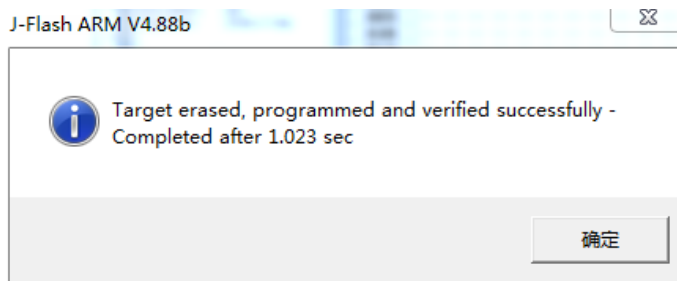


图 B-4 烧录成功提示框

## D.3 常见问题处理

### D.3.1 指示灯频闪，不能烧录程序

指示灯频闪表示 SWD-Programmer 没有连接好，重新连接即可。

如果在程序烧录过程中出现图 B-5 的问题，同样表示 SWD-Programmer 没有连接好，重新连接即可。

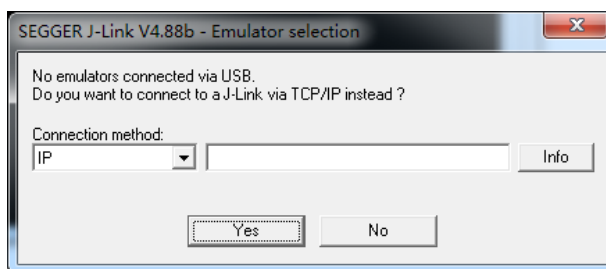
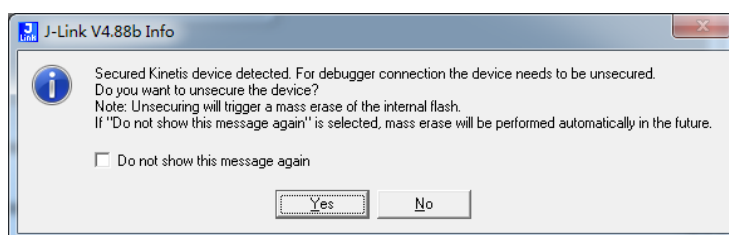


图 B-5 写入器频繁复位提示

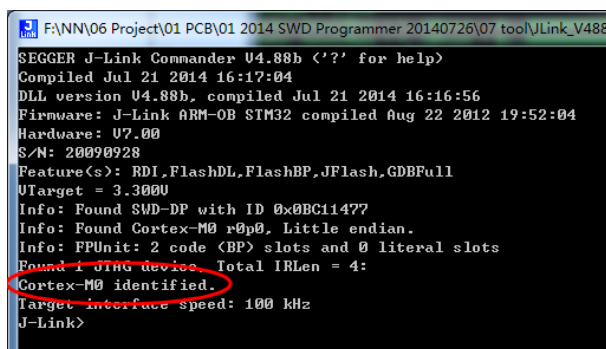
### D.3.2 目标板芯片被加密

软件操作不当、硬件连接错误都有可能导致芯片加密，而芯片加密后无法正常工作，所以需要进行芯片解密。芯片加密后，烧录程序时会出现下图的提示：



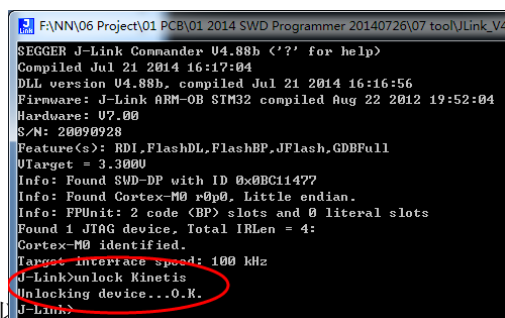
芯片加密问题是无法完全避免的，因此芯片解密操作显得尤为重要。芯片解密步骤如下：

(1) 按照 D.2.1 节进行硬件连接，接好后还需将目标板的 reset 引脚接到 GND，打开 JLink 软件（如果提示固件升级，请升级）



JLink 软件自动识别到目标板的内核，表示连接正确。

接下来输入“unlock Kinetis”命令，界面显示“Unlocking device...O.K.”，即表示芯片成功解密。



注意：若芯片可以烧录程序，说明芯片有可能被加密，利用上述方法解锁。


## 附录E OpenSDA-Programmer简明使用方法

### E.1 驱动的安装

KEA128 板载写入器为 OpenSDA, 只需连接 KEA128 开发板的 Micro USB 口(USB\_SDA)和电脑的 USB 口, 驱动即可自动安装。安装好后, 可在设备管理器中查看, 端口中多出了 OpenSDA 及虚拟串口。若驱动没有自动安装, 用户可前往 <http://www.pemicro.com/opensda/> 下载并安装驱动“PEDrivers\_install.exe”或安装 KDS 集成开发环境。

### E.2 下载机器码与跟踪调试

#### E.2.1 使用KDS直接下载机器码

首先连好线, 然后点击“”, 需双击左侧调试类型窗口中的 GDB PEMicro Interface Debugging, 会在其下方自动新建一个以当前工程命名的烧录连接, 单击其进入烧录配置界面。

接着, 在烧录配置界面中选择 Main 选项卡, 在 Project 中选择烧录文件所在工程, 在 C/C++ Application 中选择需要烧录的文件, 点击 Search Project (相对路径, 只能选择.elf 文件), 若无机器码可选, 请参见 E.3.1 进行解决。

然后, 进入烧录配置界面中的 Debugger 选项卡, PEMicro Interface Settings 的 Interface 项选择“OpenSDAC Embeddwd Debug – USB Port”, Port 项会显示相应的“USB1 - OpenSDA”, Device Name 项中选择设备名“KEAZ128M4”, 此时直接点“Flash”即可。

#### E.2.2 使用KDS进行跟踪调试

点击“”的下拉单, 按照 E.2.1 进行配置, 然后直接点“Debug”即可。

### E.3 常见问题处理

#### E.3.1 程序下载时, Flash Configurations界面无法选择机器码文件

首先检查工程是否被编译, 若无请参照附录 E.2.2 进行操作, 再进行尝试; 若已编译, 请刷新当前工程。刷新方法如下 (二选一即可):

- (1) 选中当前工程, 按“F5”按钮;
- (2) 右击当前工程, 选择“Refresh”选项。

## 附录F printf格式化输出

### F.1 printf调用的一般格式

printf 函数是一个标准库函数，它的函数原型在头文件“stdio.h”中。但作为一个特例，不要求在使用 printf 函数之前必须包含 stdio.h 文件。

printf 函数调用的一般形式为：printf(“格式控制字符串”，输出表列)。

其中格式控制字符串用于指定输出格式。格式控制串可由格式字符串和非格式字符串两种组成。格式字符串是以%开头的字符串，在%后面跟有各种格式字符，以说明输出数据的类型、形式、长度、小数位数等。如：

” %d”表示按十进制整型输出；

” %ld”表示按十进制长整型输出；

” %c”表示按字符型输出等。

非格式字符串原样输出，在显示中起提示作用。输出表列中给出了各个输出项，要求格式字符串和各输出项在数量和类型上应该一一对应。

#### F.1.1 格式字符串

格式字符串的一般形式为：[标志][输出最小宽度][精度][长度]类型，其中方括号[]中的项为可选项。以下说明各项的意义。

##### 1. 类型

类型字符用以表示输出数据的类型，其格式符和意义如表 D-1 所示。

表D-1 类型字符

格式字符	意义
d	以十进制形式输出带符号整数(正数不输出符号)
o	以八进制形式输出无符号整数(不输出前缀 0)
x,X	以十六进制形式输出无符号整数(不输出前缀 0x)
u	以十进制形式输出无符号整数
f	以小数形式输出单、双精度实数
e,E	以指数形式输出单、双精度实数
g,G	以%f或%e中较短的输出宽度输出单、双精度实数
c	输出单个字符
s	输出字符串

##### 2. 标志

标志字符为 -、+、# 和空格四种，其意义如表 D-2 所示。

表D-2 标志字符

标志	意义
-	结果左对齐，右边填充空格
+	输出符号(正号或负号)

##### 3. 输出最小宽度

用十进制整数来表示输出的最少位数。若实际位数多于定义的宽度，则按实际位数输出，若实际位数少于定义的宽度则补以空格或 0。

##### 4. 精度

精度格式符以“.”开头，后跟十进制整数。本项的意义是：如果输出数字，则表示小数

的位数；如果输出的是字符，则表示输出字符的个数；若实际位数大于所定义的精度数，则截去超过的部分。

## 5. 长度

长度格式符为 h、l 两种，h 表示按短整型量输出，l 表示按长整型量输出。

### F.1.2 输出格式举例

实验测试用例：

```
char c,s[20];
int a;
float f;
double x;
a=1234;
f=3.14159322;
x=0.123456789123456789;
c='A';
strcpy(s,"Hello,World");

printf("苏州大学嵌入式实验室 KEA128-printf 构件测试用例!\n");
//整数数据类型数据输出测试
printf("整型数据输出测试:\n");
printf("整数 a=%d\n", a);           //按照十进制整数格式输出，显示 a=1234
printf("整数 a=%d%\n", a);          //输出%号 结果 a=1234%
printf("整数 a=%6d\n", a);          //输出 6 位十进制整数 左边补空格，显示 a= 1234
printf("整数 a=%06d\n", a);         //输出 6 位十进制整数 左边补 0，显示 a=001234
printf("整数 a=%2d\n", a);          //a 超过 2 位，按实际输出 a=1234
printf("整数 a=%-6d\n", a);         //输出 6 位十进制整数 右边补空格，显示 a=1234
printf("\n");
//浮点数据类型数据输出测试
printf("浮点型数据输出测试:\n");
printf("浮点数 f=%f\n", f);          //浮点数有效数字是 7 位，结果 f=3.14159297
printf("浮点数 f = %6.4f\n", f);     //输出 6 列，小数点后 4 位，结果 f=3.1416
printf("double 型数 x=%lf\n", x);    //输出长浮点数 x=0.12345678912345678
printf("double 型数 x=%18.15lf\n", x); //输出 18 列，小数点后 15 位，x=0.123456789123456
printf("\n");
//字符类型数据输出测试
printf("字符类型数据输出测试:\n");
printf("字符型 c=%c\n", c);          //输出字符 c=A
printf("ASCII 码 c=%x\n", c);        //以十六进制输出字符的 ASCII 码 c=41
printf("字符串 s[]=%s\n", s);        //输出数组字符串 s[]=Hello,World
printf("字符串 s[]=%6.9s\n", s);     //输出最少 6 列，最多 9 个字符的字符串 s[]=Hello,Wor
```

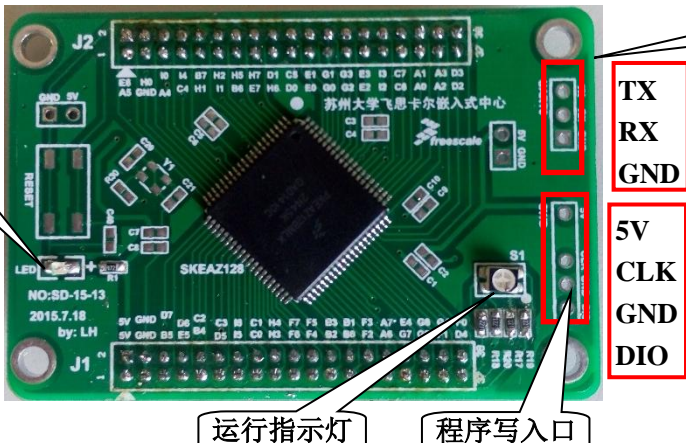

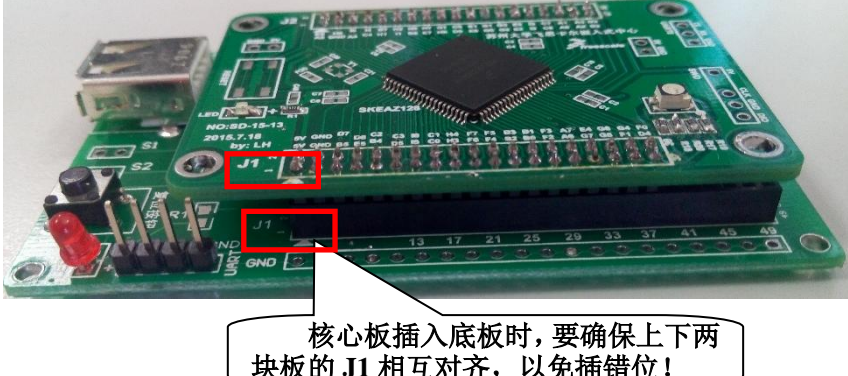

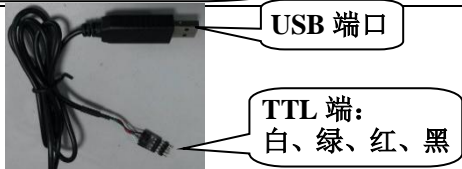




## 附录 G 《KEA128 开发套件》简明测试方法




## 1. 物件清单:

核心板	
底板	
核心板 + 底板	
写入器	 <div> <div>USB-TTL 串口线</div>  </div>

2. 测试方法: (出厂时, 已经将 KEA128 (网上光盘) 中.. \ 04-Other\KEA128-Test \Debug \KEA128\_Test.hex” 机器码文件写入 KEA128 芯片中)。

**第一步: 观察三色灯变化:** 将 KEA128 核心板正确插入底座中, 用套件中的任意 USB

线将核心板接 5V 电源，现象：核心板上的小灯按红、绿、蓝、白次序闪烁，则 KEA128 运行正常（若不运行，可重新插入 USB 线，再试一下）。

**第二步：串口通信测试：**（1）在 PC 机安装 USB-TTL 驱动（网上光盘“..\03-tool”中） PL2303\_Prolific\_DriverInstaller\_v1.5.0.exe；（2）将“USB-TTL 串口线”的“USB 端口”接 PC 机的 USB 口，串口线的 TTL 端接底板上的串口（4 根，TX 接白线，RX 接绿线，5V 接红线，GND 接黑线，见上图。注意：5V 与 GND 不能接反，否则会烧坏芯片）；（3）在 PC 机，运行“SerialPort.exe”（网上光盘“..\03-tool \C#2010 串口测试程序\bin\Debug”）或“..\03-Tool\串口调试工具.exe”，波特率默认选择“9600”，选择正确的串口号并打开串口，按核心板上的“复位按钮”，串口测试窗口显示“KEA128 是……”。说明核心板串口发送正常。在字符串输入框内容“abcdefg1234567h”，单击“发送”按钮，若出现回显，说明核心板串口接收正常。

**注意：**核心板与底板上引出的串口不同，核心板上串口为 UART0，底板上串口为 UART2。KEA128-Test 测试程序中使用的是 UART2。因此使用测试程序测试串口通信时，“USB-TTL 串口线”的 TTL 端要与底板上的串口 UART2 相接。

其它详细资料请参考网上光盘：<http://sumcu.suda.edu.cn> → “KEA128 书” → “SD-FSL-KEA-CD”。