

目 录

第 1 章	LPC2103 与 EasyARM2103 简介	1
1.1	概述.....	1
1.2	LPC2103 特性	1
1.3	EasyARM2103 简介	1
第 2 章	TKStudio 与 EasyARM2103 快速入门	3
2.1	TKStudio 集成开发环境简介	3
2.2	工程的编辑.....	4
2.2.1	建立工程.....	4
2.2.2	使用直接新建工程方式建立工程.....	5
2.2.3	使用从工程模板新建工程方式建立工程.....	6
2.2.4	建立文件.....	6
2.2.5	添加文件到工程.....	7
2.2.6	编译连接工程.....	8
2.2.7	打开旧工程.....	10
2.2.8	工程的调试.....	11
2.3	使用 GNU ARM LPC2103 工程模板	15
2.3.1	使用 GNU ARM LPC2103 工程模板建立工程	15
2.3.2	GNU ARM LPC2103 工程模板适用范围	16
2.4	EasyJTAG-H 简介	16
2.4.1	EasyJTAG-H 安装	16
2.4.2	H-JTAG 配置	17
2.5	在 TKStudio 中使用 EasyJTAG-H 仿真器.....	20
第 3 章	实验例程	24
3.1	TKStudio 应用示例	24
3.1.1	实验目的.....	24
3.1.2	实验内容.....	24
3.1.3	实验步骤.....	24
3.1.4	出现的问题与解决方法.....	31
3.2	GPIO 实验	32
3.2.1	普通低速 GPIO 实验.....	32
3.2.2	高速 GPIO 实验	33
3.3	外部中断实验.....	35
3.3.1	实验目的.....	35
3.3.2	实验内容.....	35
3.3.3	实验要求.....	35
3.3.4	实验思路.....	35
3.3.5	实验步骤.....	35
3.3.6	GCC 中断服务程序编写说明.....	35
3.3.7	实验参考程序.....	37
3.4	定时器实验.....	38
3.4.1	定时器实验 1.....	38
3.4.2	定时器实验 2.....	40

3.4.3	定时器实验 3.....	42
3.4.4	定时器实验 4.....	43
3.5	UART 实验	45
3.5.1	UART 实验 1	45
3.5.2	UART 实验 2	48
3.6	I ² C 接口实验	52
3.6.1	实验目的.....	52
3.6.2	实验内容.....	52
3.6.3	实验要求.....	52
3.6.4	实验思路.....	53
3.6.5	实验步骤.....	53
3.6.6	实验参考程序.....	53
3.7	SPI 接口实验.....	54
3.7.1	实验目的.....	54
3.7.2	实验要求.....	54
3.7.3	实验思路.....	54
3.7.4	实验步骤.....	54
3.7.5	实验参考程序.....	54
3.8	SSP 实验	55
3.9	RTC 实验	55
3.9.1	RTC 实验 1	55
3.9.2	RTC 实验 2	60
3.10	模数转换器实验.....	61
3.10.1	实验目的.....	61
3.10.2	实验内容.....	61
3.10.3	实验要求.....	61
3.10.4	实验思路.....	61
3.10.5	实验步骤.....	62
3.10.6	实验参考程序.....	62
3.11	WDT 实验.....	64
3.11.1	实验目的.....	64
3.11.2	实验内容.....	64
3.11.3	实验要求.....	64
3.11.4	实验思路.....	64
3.11.5	实验步骤.....	64
3.11.6	烧写目标文件到 FLASH 步骤	64
3.11.7	实验参考程序.....	66
3.12	PLL 设置实验.....	67
3.12.1	实验目的.....	67
3.12.2	实验内容.....	67
3.12.3	实验要求.....	68
3.12.4	实验思路.....	68
3.12.5	实验步骤.....	68
3.12.6	实验参考程序.....	68

第1章 LPC2103 与 EasyARM2103 简介

1.1 概述

LPC2103 是一个基于支持实时仿真的 16/32 位 ARM7TDMI-S CPU 的微控制器, 并带有 32KB 的嵌入高速 Flash 存储器, 128 位宽度的存储器接口和独特的加速结构使 32 位代码能够在最大时钟速率下运行。

较小的封装和极低的功耗使 LPC2103 适用于访问控制器和 POS 机等小型应用系统中; 由于内置了宽范围的串行通信接口(2 个 UART、SPI、SSP 和 2 个 I²C)和 8KB 的片内 SRAM, LPC2103 也适合用在通信网关和协议转换器中。32/16 位定时器、增强型 10 位 ADC、定时器输出匹配 PWM 特性、多达 13 个边沿、电平触发的外部中断、32 条高速 GPIO, 使得 LPC2103 微控制器特别使用于工业控制和医疗系统中。

1.2 LPC2103 特性

- 16/32 位 ARM7TDMI-S 微控制器, 超小 LQFP48 封装;
- 8KB 的片内静态 RAM 和 32KB 的片内 Flash 程序存储器。128 位宽度接口/加速器可实现高达 70MHz 工作频率;
- 通过片内 boot 装载程序实现在系统/在应用编程 (ISP/IAP)。单个 Flash 扇区或整片擦除时间为 100ms, 256 字节编程时间为 1ms;
- 嵌入式 ICE RT 通过片内 RealMonitor 软件提供实时调试;
- 10 位 A/D 转换器提供 8 路模拟输入 (每个通道的转换时间低至 2.44us), 以及特定的结果寄存器来最大限度地减少中断开销;
- 2 个 32 位定时器/外部事件计数器 (带 7 路捕获和 7 路比较通道);
- 2 个 16 位定时器/外部事件计数器 (带 3 路捕获和 7 路比较通道);
- 低功耗实时时钟 (RTC) 具有独立的电源和特定的 32KHz 时钟输入;
- 多个串行接口, 包括 2 个 UART(16C550 协议标准)、2 个高速 I²C 总线(400Kbit/s)、SPI 和具有缓冲作用和数据长度可变功能的 SSP;
- 向量中断控制器 (VIC), 可配置优先级和向量地址;
- 多达 32 个通用 I/O 口 (可承受 5V 电压);
- 多达 13 个边沿、电平触发的外部中断管脚;
- 通过一个可编程的片内 PLL (100us 的设置时间) 可实现最大为 70MHz 的 CPU 操作频率, 其具有 10MHz~25MHz 的输入频率;
- 片内集成振荡器与外部晶体的操作频率范围为 1~25MHz;
- 低功耗模式包括空闲模式、带 RTC 的睡眠模式和掉电模式;
- 可通过个别使能/禁止外围功能和外围时钟分频来优化额外功耗;
- 通过外部中断或 RTC 将处理器从掉电模式中唤醒。

1.3 EasyARM2103 简介

EasyARM2103 开发板是广州周立功单片机发展有限公司设计的 EasyARM 系列开发套件之一, 它采用了 NXP 公司基于 ARM7TDMI-S 核、LQFP48 封装的 LPC2103 芯片, 具有 JTAG 仿真调试和 ISP 编程功能。

EasyARM2103 开发板上提供了按键、发光二极管等常用的功能器件，具有 RS-232 接口电路和 I²C 存储器电路。用户可以更换兼容的 CPU 进行仿真调试，如 LPC2101 和 LPC2102 芯片等。开发板上所有的 I/O 口全部引出，灵活的跳线组合，极大的方便用户进行 32 位 ARM 嵌入式系统的开发实验。

功能特点：

- 采用“底板+PACK 板”的形式构成 EasyARM2103 开发套件，PACK 板的主芯片为 LPC2103；
- 板上所有的功能器件与 LPC2103 的引脚可通过跳线来连接；
- 配套有详细的开发板实验教程；
- I/O 口全部引出，方便用户连接外部电路进行开发；
- 可进行 GPIO 的输入输出实验，如按键输入、发光二极管输出等；
- 按键、发光二极管分别可用于外部中断、GPIO 输出等；
- 具有 RS-232 转换电路，可与上位机进行通信，完成 UART 通信实验；
- 具有 I²C 接口和 SPI/SSP 接口输出；
- 提供基于 PC 的人机界面，方便调试实时时钟和串口通信等；
- 可进行外部中断实验，学习向量中断控制器（VIC）；
- 定时器实验，如定时器输出和定时器捕获等；
- 复位芯片 CAT1025，完成 I²C 总线实验；
- A/D 转换实验；
- 实时时钟控制实验；
- WDT 看门狗实验；
- 详细的实验教程和实验例程完整地验证了芯片的所有功能外设。

第2章 TKStudio 与 EasyARM2103 快速入门

2.1 TKStudio 集成开发环境简介

TKStudio IDE 集成开发环境是广州致远电子有限公司开发的一个微处理软件开发平台，它是一款具有强大内置编辑器的多内核编译/调试环境，支持 8051、ARM v4/v5/v7、AVR 等内核，可以完成从工程建立和管理、编译、链接、目标代码的生成，到软件仿真，硬件仿真(挂接 TKS 系列仿真器的硬件)等完整的开发流程。

TKStudio 集成开发环境特点：

- 支持众多主流内核，全部操作界面高度统一，免除开发环境熟悉周期；
- 傻瓜式的成员提示及代码模板功能，提升编码速度及准确度；
- 智能的代码导航系统，方便阅读和分析代码；
- 方便的原型对比功能，快速追踪分析代码结构；
- 专业的文件比较/差异合并功能，显示处理不同版本的细微差别；
- 支持超大文件及十六进制编辑功能；
- 完善的工程管理，多目标模式，多工具链编译；
- 强大的专业编辑器，可配置语法高亮，独立元素颜色、大小突出显示，代码智能格式化和折叠，列编辑功能等；
- 完美的兼容性，支持 uV2、uV3、ADS1.2 工程导入；
- 支持系统级调试，配备大量调试工具。

TKStudio 主窗口如图 2.1 所示。

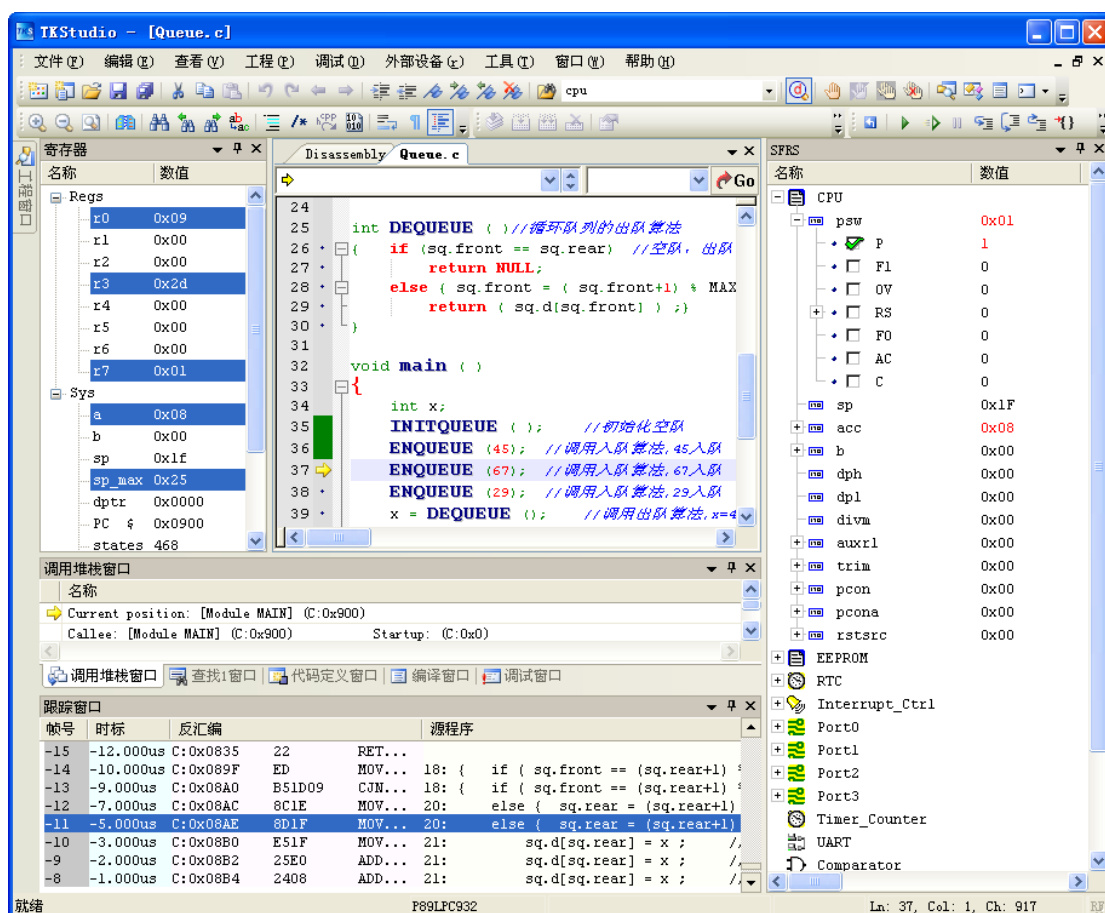


图 2.1 TKStudio 集成开发环境

2.2 工程的编辑

2.2.1 建立工程

在 TKStudio 集成开发环境中，可以通过两种方式建立工程：直接新建工程和从工程模板新建工程。采用直接新建工程方式建立工程，要选择相应的编译工具链和目标设备；采用从工程模板新建工程方式建立工程，则不需要选择编译工具链和目标设备，直接采用相应工程模板中的配置。

选择 Windows 操作系统的“开始”→“程序”→“TKStudio”→“TKStudio”命令，启动 TKStudio 集成开发环境，或双击 TKStudio 快捷方式启动。启动 TKStudio 如图 2.2 所示。

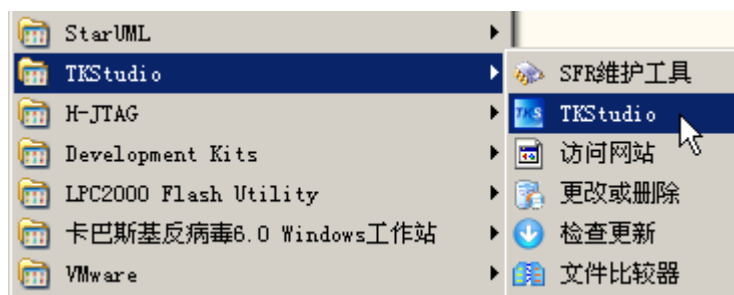


图 2.2 启动 TKStudio

2.2.2 使用直接新建工程方式建立工程

单击“工程”菜单，选择“新建工程”菜单项即弹出<新建工程>对话框。如图 2.3 所示。



图 2.3 新建工程

在“保存在”项中选择工程存放路径，并在“文件名”项中输入工程名称，并单击【保存】按钮，将弹出<选择目标 CPU>对话框。如图 2.4 所示。

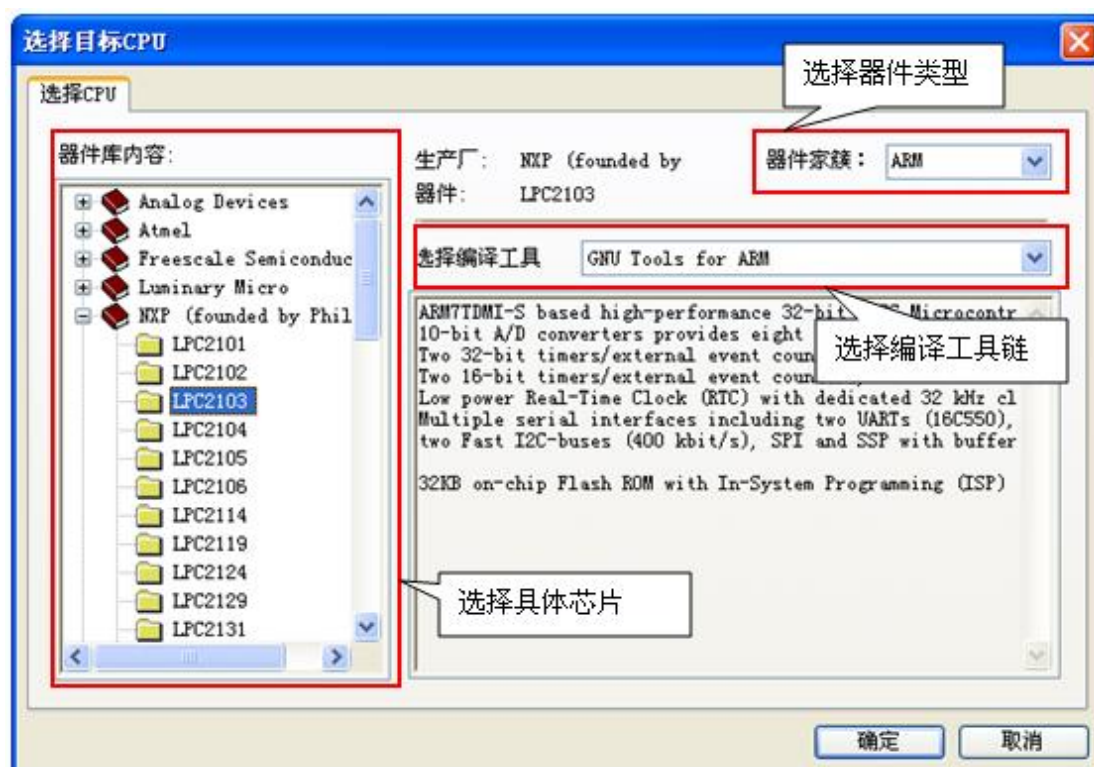


图 2.4 选择目标 CPU

在“器件家族”项中选择器件类型，在“选择编译工具”项选择编译工具链，并在“器件库内容”项中选择具体芯片，然后单击“确定”按钮即可建立相应工程，工程文件名后缀

为.xmp。

2.2.3 使用从工程模板新建工程方式建立工程

单击“工程”菜单，选择“从工程模板新建工程”菜单项即弹出<新建>对话框。如图 2.5 所示。

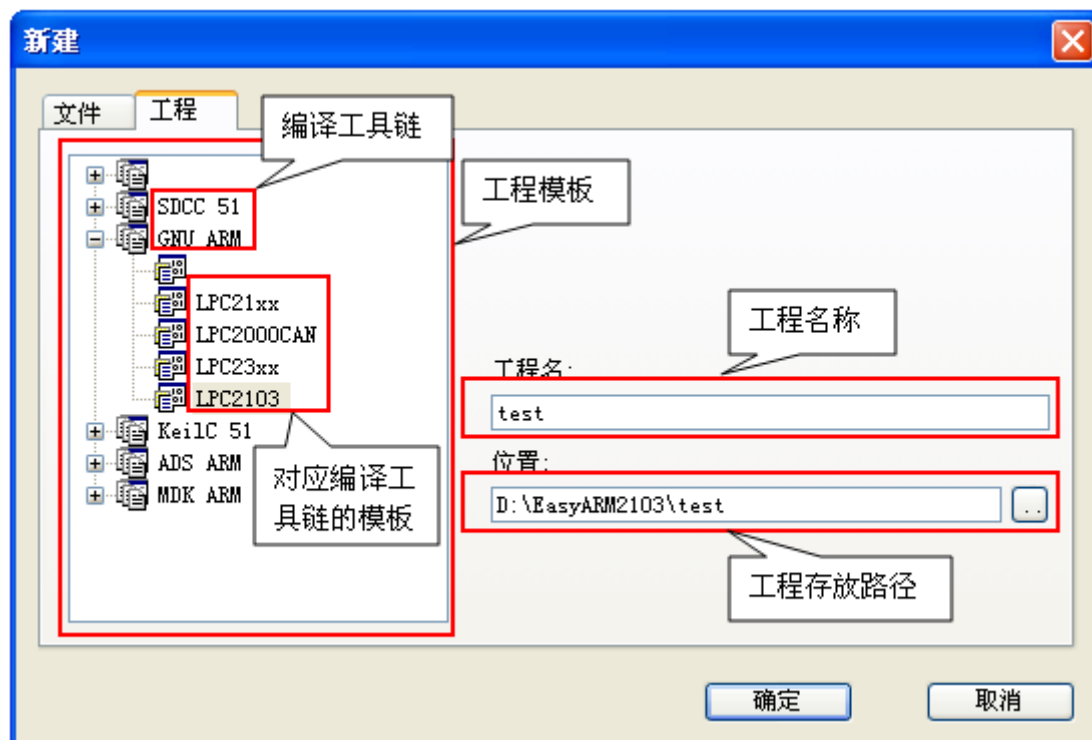


图 2.5 从模板新建工程

在左边工程模板中，展开对应的工具链项，选择对应的工程模板，在“工程名”项中输入工程名称，然后在“位置”项中选择工程存放路径，单击“确定”按钮即可建立相应工程，工程后缀名为.xmp。

2.2.4 建立文件

建立一个文本文件，以便输入用户程序。点击“新文件”图标按钮，如图 2.6 所示。



图 2.6 新文件图标按钮

然后在新建的文件中编写程序，单击保存图标按钮将文件存盘（或在“文件”菜单中执行保存命令），输入文件全名，如 Test.c。注意,请将文件保存到相应工程目录下，以便于管理和查找。

您也可以通过选择“文件”→“新建”命令或点击“新建”图标按钮弹出<新建>对话框建立文件。如图 2.7 所示。

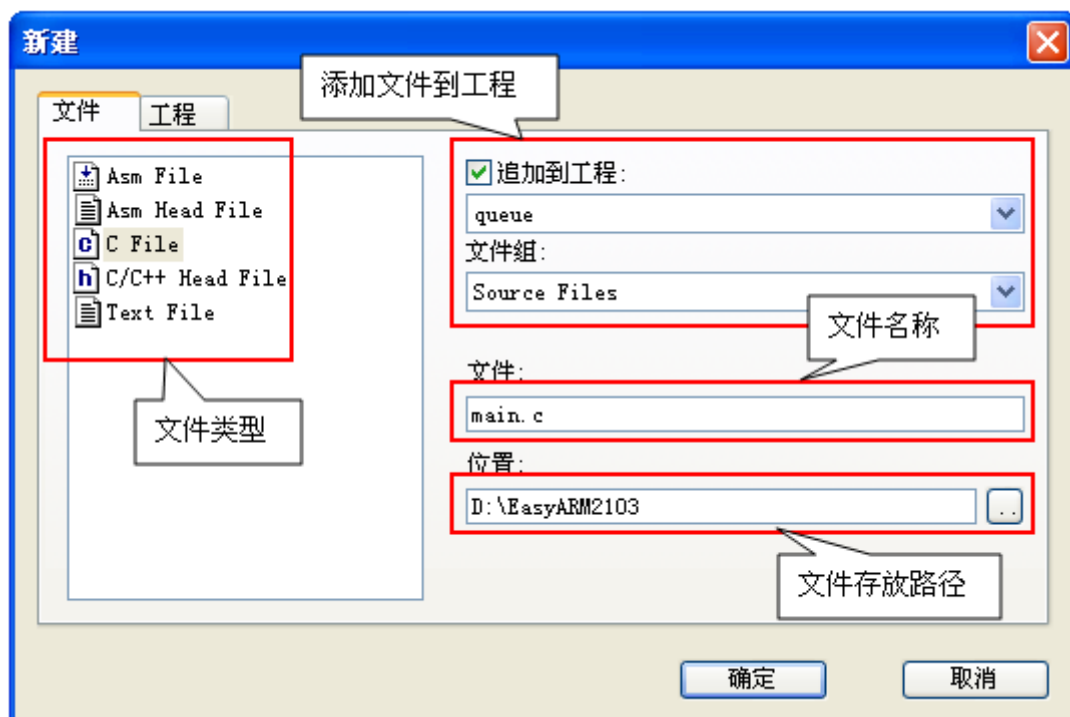


图 2.7 新建文件

在左边文件类型列表中，选择所要建立的文件类型，如要将新建的文件添加到工程中，选中“追加到工程”复选框，然后选择对应的工程和文件组。在“文件”项中输入文件名，可以不包含文件后缀，然后在“位置”项中选择文件存放路径，单击“确定”按钮即可。

2.2.5 添加文件到工程

如图 2.8 所示，在工程窗口中，选择对应的文件组，右键单击，弹出浮动菜单，选择“追加文件到文件组”命令即可弹出打开对话框。如图 2.9 所示。

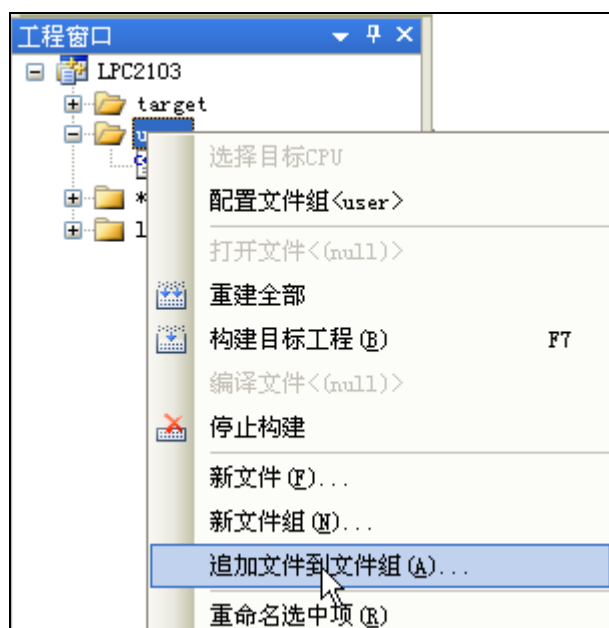


图 2.8 添加文件菜单

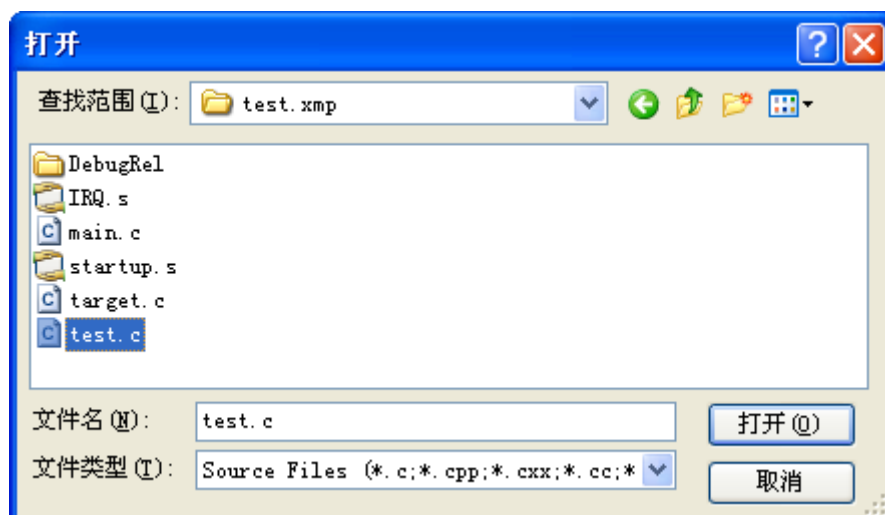


图 2.9 选择源文件

在打开对话框中，选择要添加的文件，然后单击“打开”按钮即完成文件添加操作。

2.2.6 编译连接工程

1. 编译工具配置

在进行编译、构建工程或重新构建工程之前，需要进行编译工具链的配置工作。若已进行编译工具配置，则不再需要进行重复的配置工作。

如图 2.10 所示，选择“工程”→“编译工具设置”命令即弹出<编译工具设置>对话框。如图 2.11 所示。

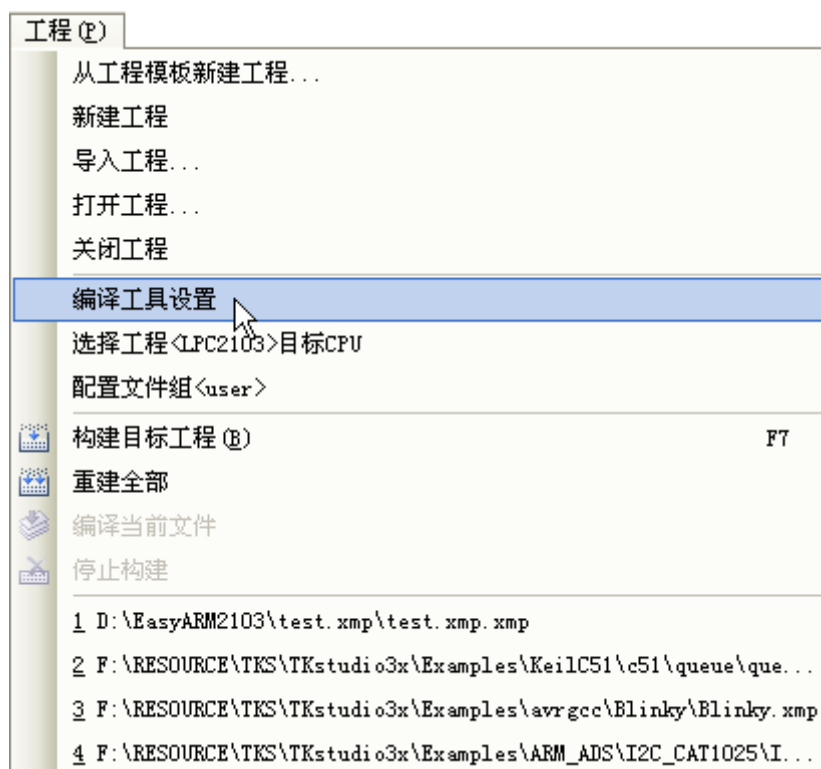


图 2.10 编译工具设置菜单

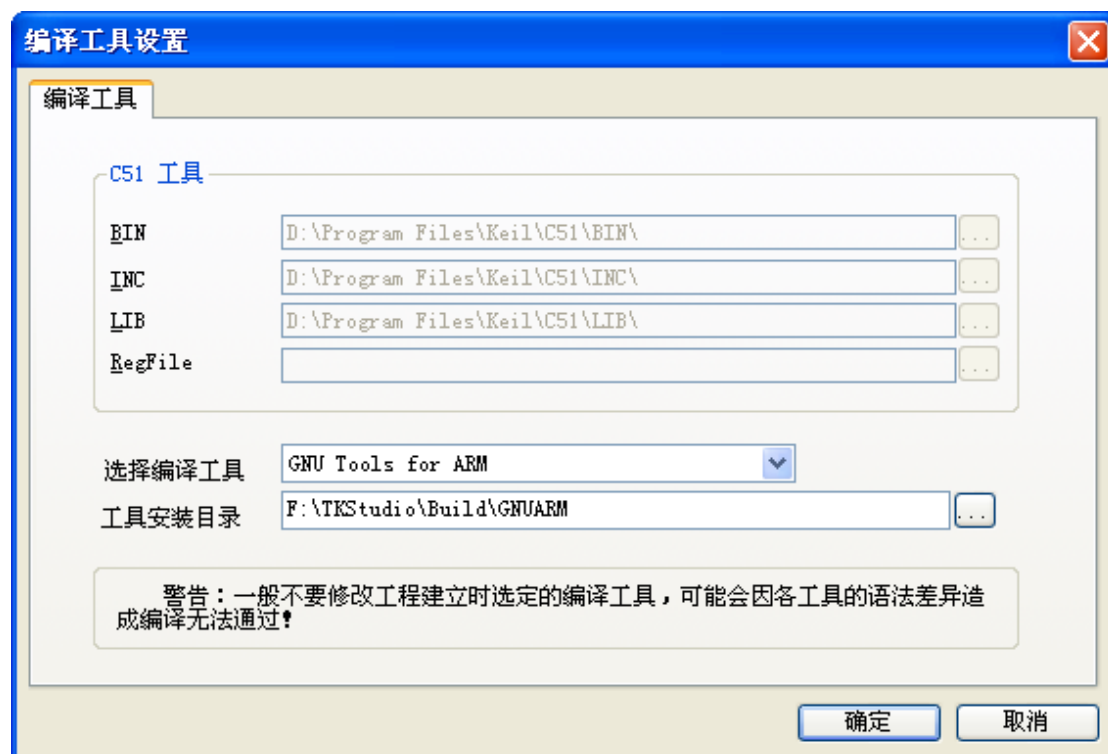


图 2.11 编译工具设置

在“选择编译工具”项中选择使用的编译工具，并在“工具安装目录”项中选择编译工具所在目录（一般为 bin 文件夹），单击“确定”按钮即完成编译工具设置。

TKStudio 中已自带了 GNU ARM 的编译工具，若选择“GNU Tools for ARM”编译工具，则不需要另外安装 GNU ARM 工具链。

2. 编译连接

如图 2.12 所示为 TKStudio 的图标按钮，通过这些图标按钮，您可以快速的进行工程配置、编译和连接。它们从左到右分别为：

- 编译当前文件：只对当前文件进行编译，不进行连接；
- 构建：对所有的目标文件进行连接，不进行编译；
- 重建所有工程文件：对所有工程文件进行编译，然后进行连接；
- 停止构建：中止正在执行构建操作；
- 工程配置：如地址设置、输出文件设置和编译选项等。其中，DebugRel 为当前工程配置名。

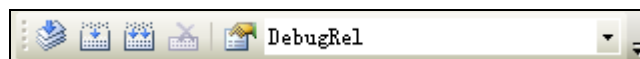


图 2.12 编译连接图标按钮

单击工程配置图标按钮，即可进行工程的地址设置、输出文件设置和编译项设置等。选择不同的编译工具，会弹出不同的工程配置对话框。若选择“GNU Tools for ARM”编译工具，则工程配置对话框如图 2.13 所示。

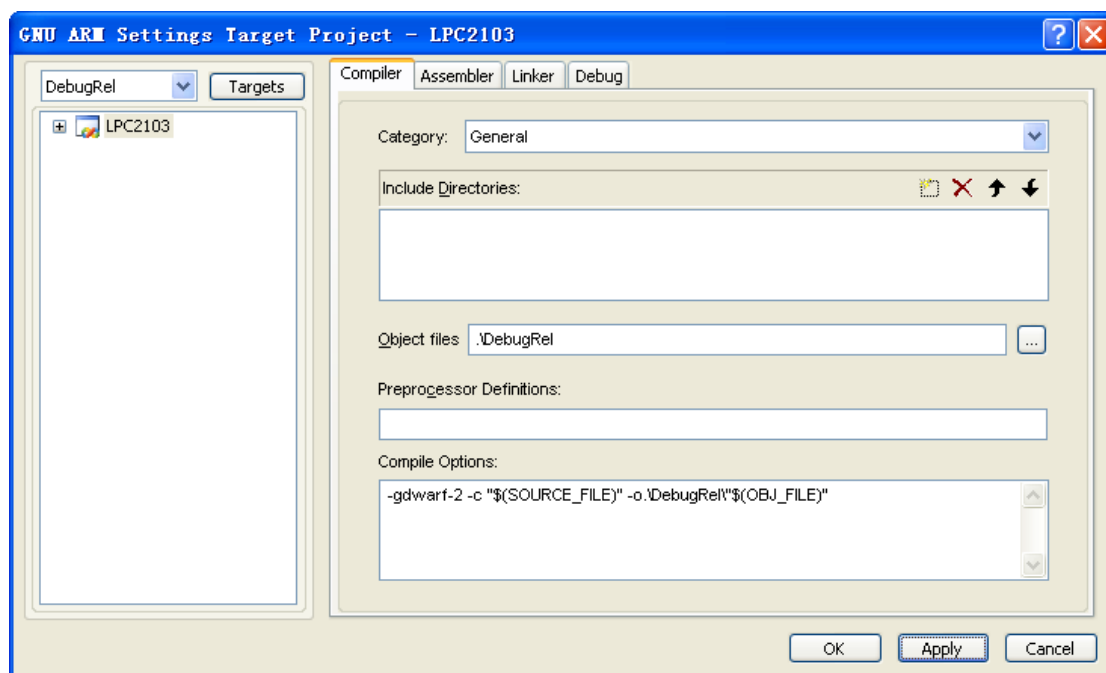


图 2.13 GNU ARM 工程配置

在工程配置对话框中，设置好编译选项后，点击编译图标按钮或构建工程按钮即可对工程进行编译或连接。若编译出错，会有相应的出错提示，双击出错提示行信息，编译窗口即会定位到当前出错的源代码行，编译连接输出窗口如图 2.14 所示。

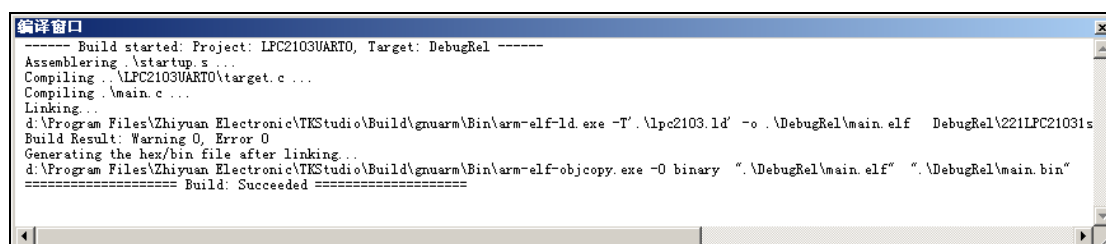


图 2.14 编译窗口

2.2.7 打开旧工程

单击“工程”菜单，选择“打开工程”命令即可弹出<打开>对话框，找到相应的工程文件 (*.xmp)，单击“打开”按钮即可。

如果您想打开最近打开的工程，单击“工程”菜单，在菜单下面的最近打开工程中选择即可。如图 2.15 所示。



图 2.15 打开最近工程

2.2.8 工程的调试

1. 选择调试目标

调试主要有两种方式：软件仿真和硬件调试。在 TKStudio 中，通过工程配置对话框可选择不同的调试方式，并设置调试参数。不同的工具链下会有不同的工程配置对话框。如选择“GNU Tools for ARM”编译工具时，调试设置界面如图 2.16 所示。

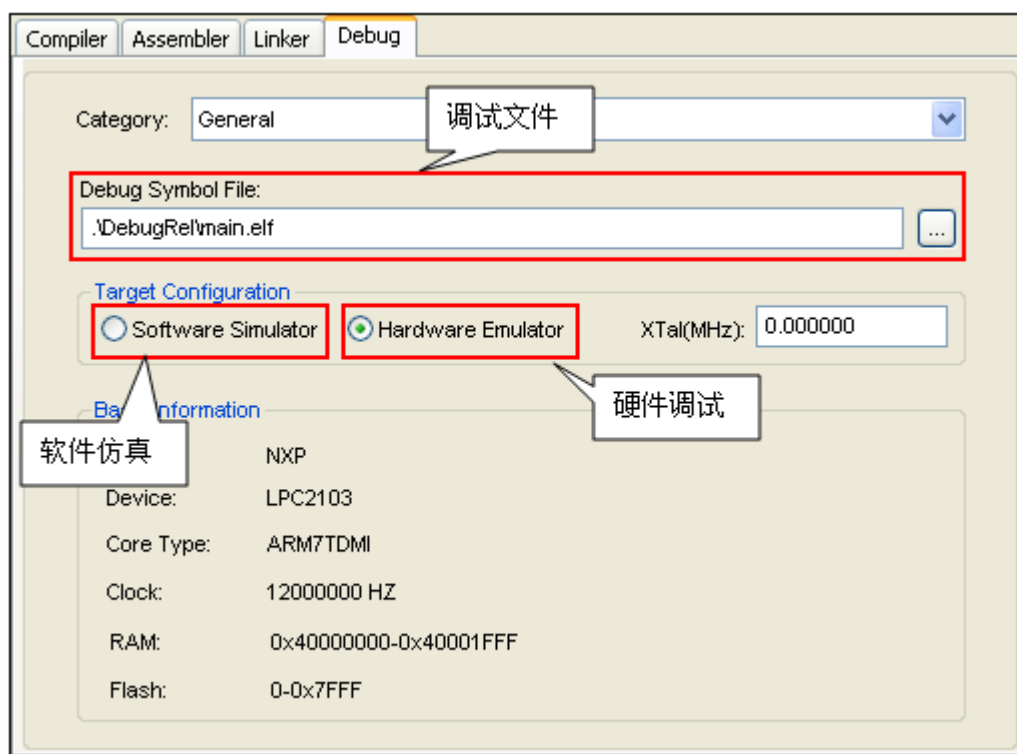


图 2.16 调试方式设置

2. 调试参数设置

调试参数主要包括：运行到 main 函数、加载调试信息、初始化脚本文件、存储器配置（软件仿真）、调试之前更新目标（硬件调试）和仿真器驱动设置（硬件调试）。各参数说明如下：

- 运行到 main：软件仿真开始后运行到 main 函数后暂停，等待用户操作；
- 加载调试信息：启动调试后，加载调试信息。在调试过程中，用户须将此项选上，否则不能进行正常的调试；
- 初始化脚本文件：设置调试时连接的初始化脚本文件；
- 存储器配置：配置存储器的类型、开始地址、大小和加载文件；
- 调试之前更新目标：调试之前，将代码重新加载到目标板上，刷新存储器的内容；
- 仿真器驱动：设置仿真器驱动。硬件调试时，TKStudio 调用该驱动，通过仿真器与设备进行通讯，仿真调试。

选择“GNU Tools for ARM”编译工具时，软件仿真调试设置界面如图 2.17 图 2.18 所示，硬件调试设置如图 2.18 所示。

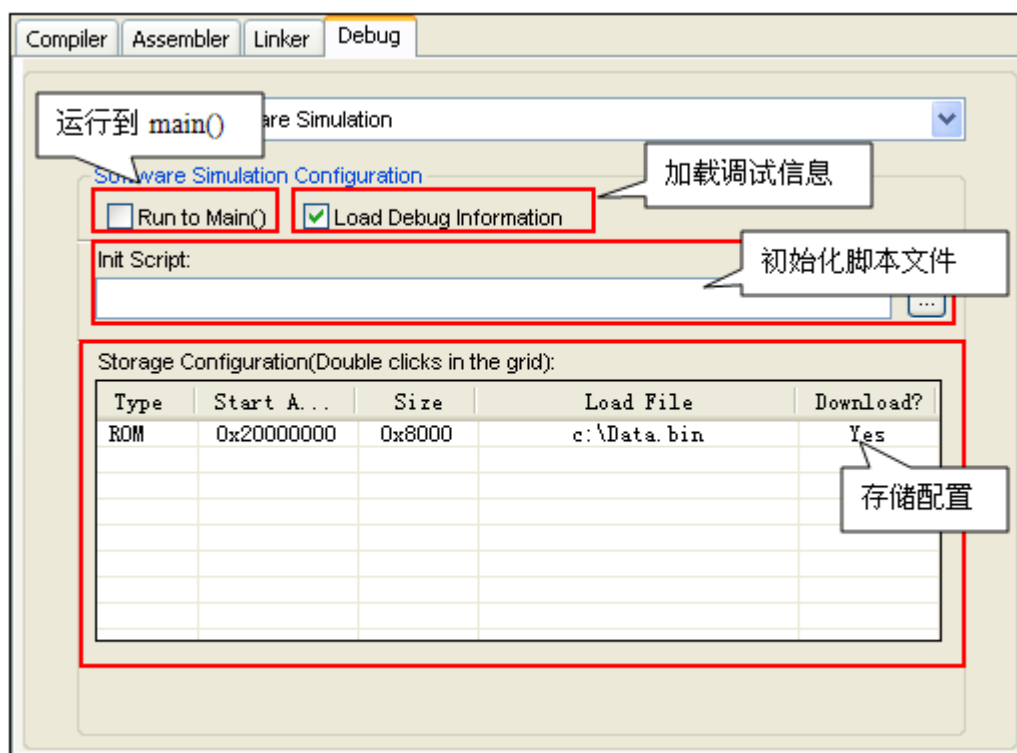


图 2.17 GNU 软件仿真设置

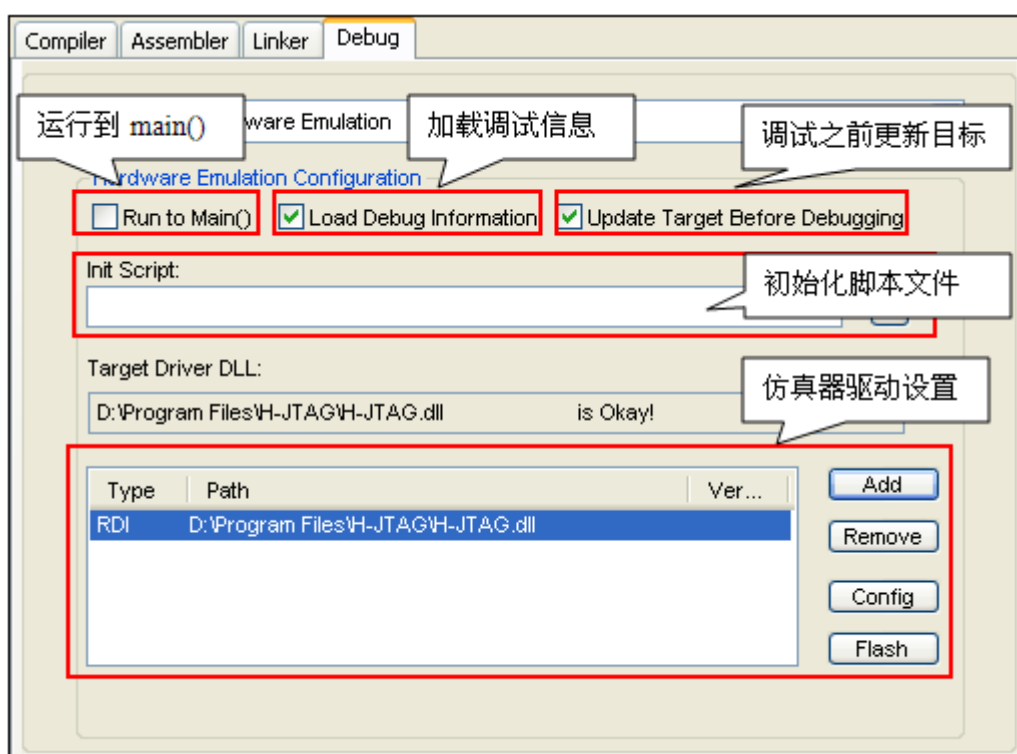


图 2.18 GNU 硬件调试设置

当工程编译连接通过后，单击“启动/停止调试”图标按钮，即可启动 TKStudio 进行调试。TKStudio 启动调试和断点操作工具条如图 2.19 所示，调试工具条如图 2.20 所示。

 显示或隐藏串行监视窗口。

 显示或隐藏存储器窗口。

 显示或隐藏寄存器窗口。

 显示或隐藏特殊功能寄存器（SFR）窗口。

 显示或隐藏跟踪窗口。

2.3 使用 GNU ARM LPC2103 工程模板

在介绍如何新建工程时，我们已经接触了 TKStudio 集成开发环境提供的几个工具链下的工程模板，如 ADS ARM 工具链的 LPC2103 工程模板、LPC21xx 工程模板，GNU ARM 工具链的 LPC2000CAN 工程模板、LPC2103 工程模板等等。在本教程中，所有实验例程基于 TKStudio 中的 GNU ARM 工具链的 LPC2103 工程模板创建。

2.3.1 使用 GNU ARM LPC2103 工程模板建立工程

启动 TKStudio 集成开发环境，选择“工程”→“从工程模板新建工程”命令，即弹出<新建>对话框，如图 2.21 所示。

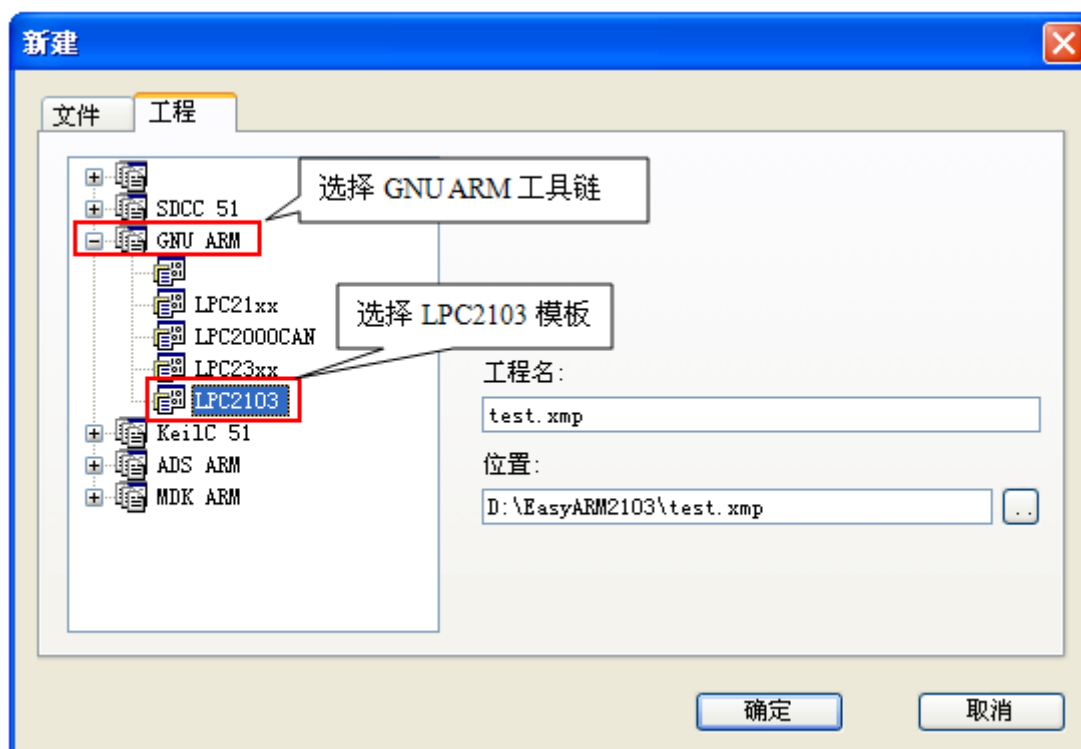


图 2.21 使用 GNU ARM LPC2103 工程模板

在工程模板列表中，展开 GNU ARM 工具链项，选择“LPC2103”工程模板，输入工程名，并选择工程存放路径，单击【确定】按钮即完成新工程的建立。使用 GNU ARM LPC2103

工程建立新工程如图 2.22 所示。

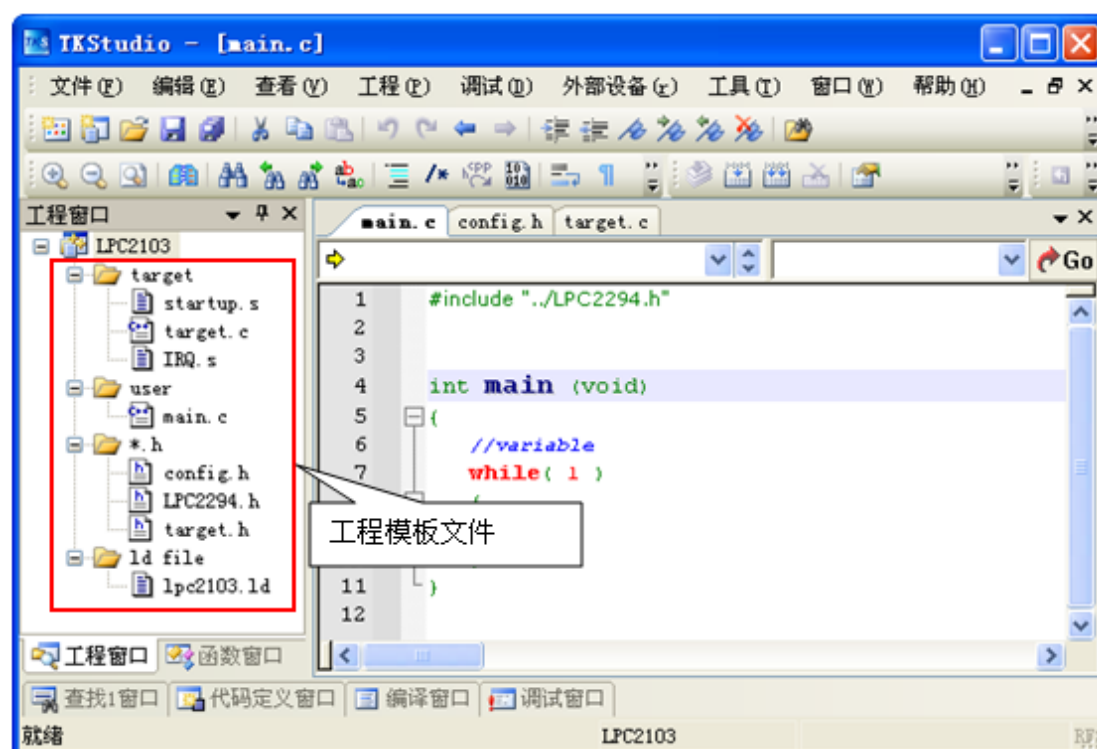


图 2.22 LPC2103 工程模板新工程

GNU ARM LPC2103 工程模板包含了 LPC2200 系列的 ARM7 微控制器的启动文件，包括 startup.s 和 target.c；还包含了 LPC2200 系列的 ARM7 微控制器的头文件（LPC2294.h，LPC2294 的寄存器是向下兼容的），连接脚本文件等等。采用该模板建立工程后，这些文件自动被添加到工程目录中。

2.3.2 GNU ARM LPC2103 工程模板适用范围

GNU ARM LPC2103 工程模板采用 GNU ARM 编译工具链的默认编译连接参数，连接文件使用 lpc2103.ld 连接脚本文件，调试入口为 0x00000000，即片内 Flash 调试。若用户需要更改调试方式，须更改 lpc2103.ld 连接脚本文件，GNU 连接脚本知识参看相关说明文档。

2.4 EasyJTAG-H 简介

EasyJTAG-H 仿真器是一款新型的仿真器。目前，可以支持 ARM7 微控制器和部分 ARM9 芯片，支持 TKStudio 集成开发环境，支持单步、全速及断点等调试功能，支持下载程序到片内 FLASH 和特定型号的片外 FLASH，采用 ARM 公司提出的标准 20 脚 JTAG 仿真器调试接口。这款仿真器需要 H-JTAG 软件（调试代理）的支持。

2.4.1 EasyJTAG-H 安装

将 EasyJTAG-H 仿真器的 25 针接口通过并口延长线与 PC 机的并口连接，将 EasyJTAG-H 仿真器的 20 针接口通过 20 PIN 连接电缆接到 EasyARM2103 教学平台的 JTAG 接口上，再使用配套的变压器（9V）给开发板供电。

由于 EasyJTAG-H 仿真器需要 H-JTAG 软件的支持, 所以使用 EasyJTAG-H 之前, 必须先安装 H-JTAG。将 H-JTAG 压缩包解压, 然后运行安装文件 H-JTAG.exe, 如图 2.23 所示, 根据安装提示完成安装即可。

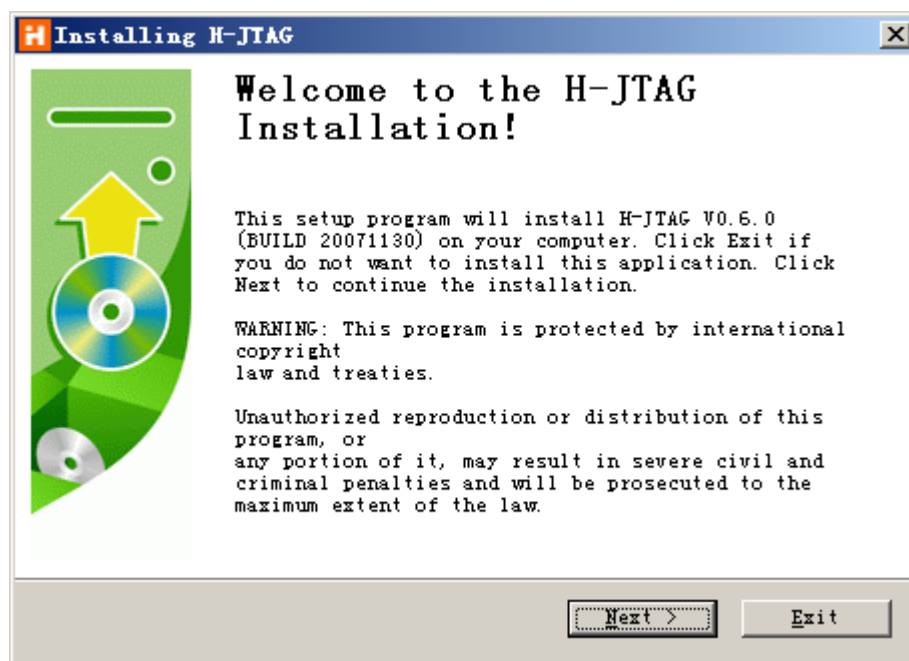


图 2.23 H-JTAG 安装界面

安装好的 H-JTAG 软件包含有 H-JTAG Server (下文简称为 H-JTAG) 和 H-Flasher, 在桌面上有它们的快捷图示, 如图 2.24 所示。



图 2.24 H-JTAG 快捷图示

2.4.2 H-JTAG 配置

H-JTAG 设置较简单, 只要进行以下两步操作, 其它采用默认设置即可。

(1) 单击任务栏的 H 提示图标, 将看见 H-JTAG 的主窗口, 如图 2.25 所示。单击“放大镜”图标按钮后, 能看见调试代理搜索到 ARM7 处理器。

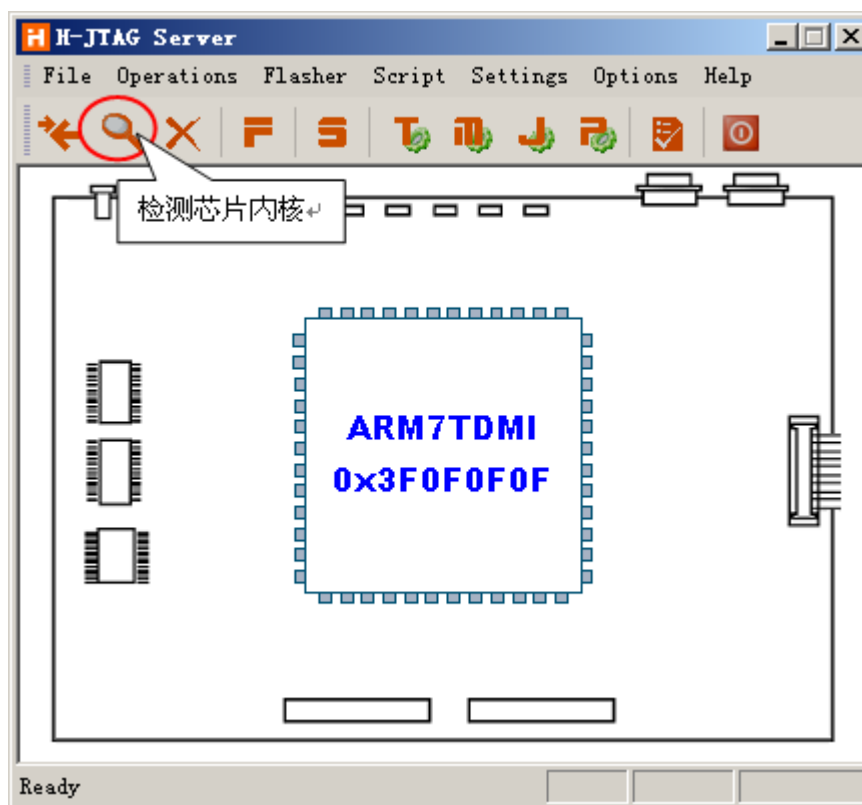


图 2.25 H-JTAG Server 检测到芯片内核

(2) 选择【Flasher】->【Auto Download】选择自动下载项。

注意，在 Flash 中调试时必须要选择“Auto Download”，而在 RAM 中调试可以不选择。

H-Flasher 主要用来完成 Flash 的烧写。如果需要手动下载 hex/bin 文件，需要按照下面的步骤进行操作：

- 在 1 Flash Selection 中选择 CPU 的型号。如图 2.26 所示；

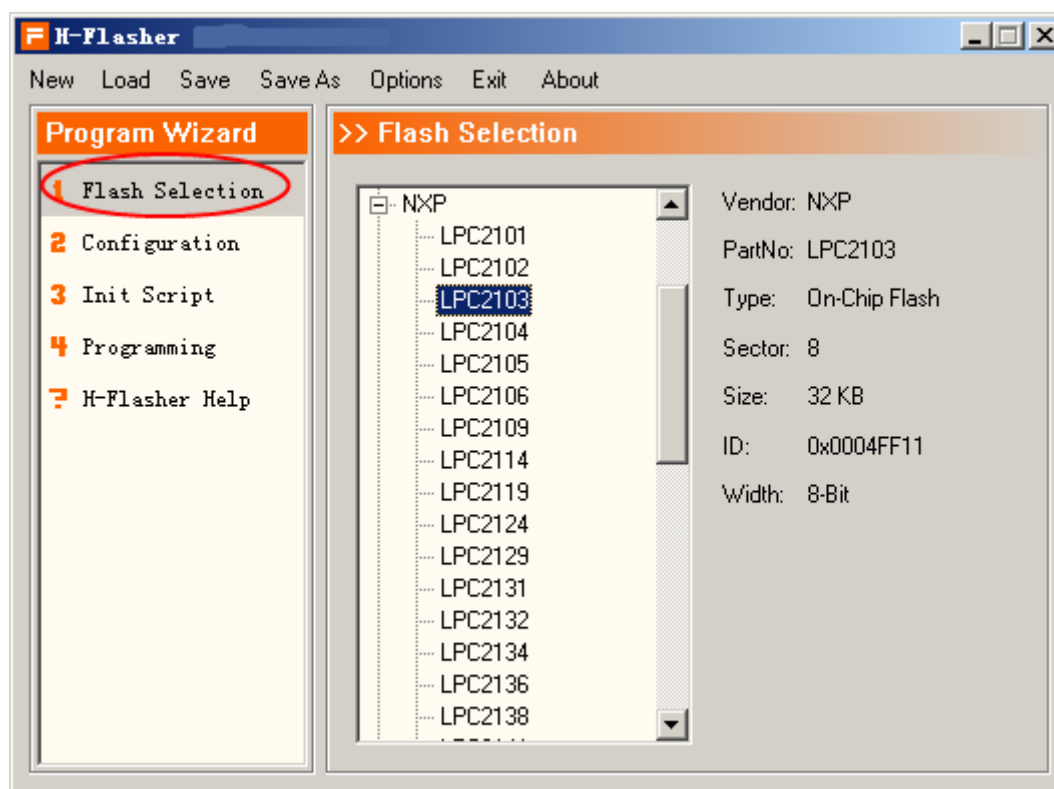


图 2.26 选择芯片型号

- 打开 4 Programming 选项，如图 2.27 所示。

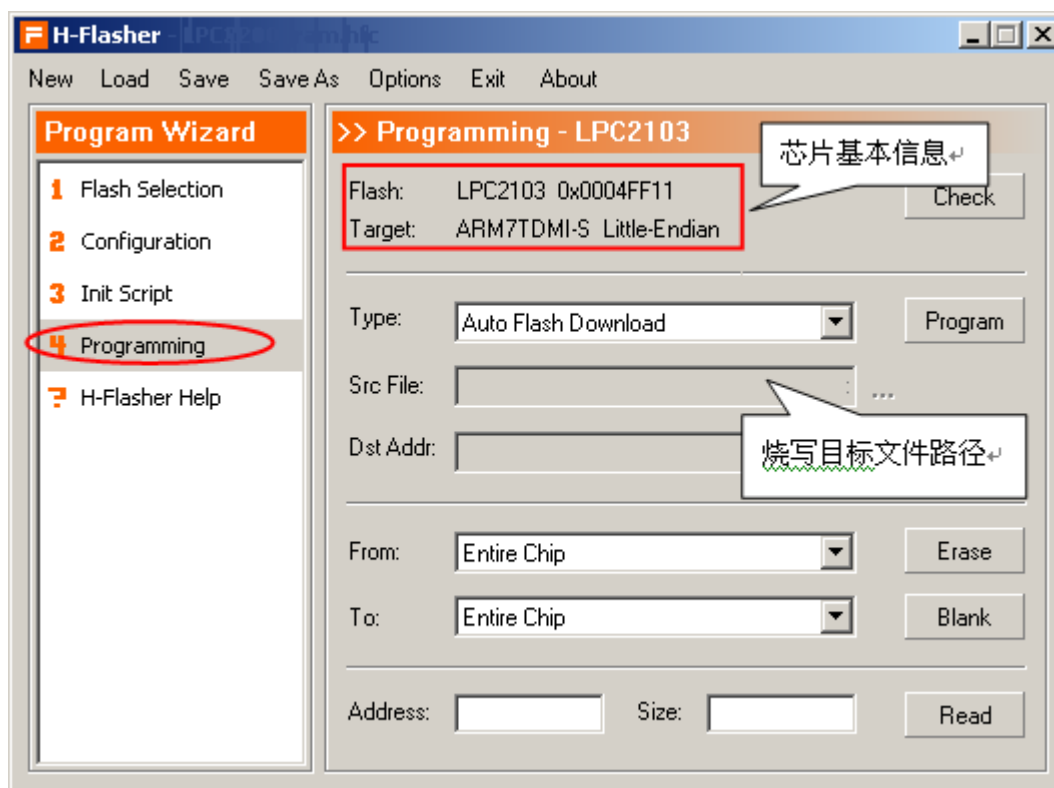


图 2.27 Flash 编程选项

Check: 检测芯片内核。如果 EasyJTAG-H 连接正确，且芯片型号正确，则 **Check** 后会

显示芯片的基本信息。

Type: 烧写文件的类型。

Auto Flash Down: 自动下载模式；

Intel Hex Format: 下载 Hex 文件；

Plain Binary Format: 下载 Bin 文件。

Src File: 烧写文件的路径，如果 **Type** 选择为 **Auto Flash Down** 时，该项无效。

Dst Addr: 目标地址信息，只有 **Type** 选择为 **Plain Binary Format** 时，该项才有效。

Program: 对芯片进行编程操作。

Erase: 对选中（由“**From ~ To**”指定）的扇区进行擦除操作。

工具栏中各选项功能说明：

New: 新建一个配置文件。

Load: 载入配置文件。**H-Flasher** 在启动时，总是自动载入最近一次的配置信息。

Save: 将当前的配置信息保存为一个文件。

Save As: 将配置信息另存。

Option: 调试程序时，是否使能自动计算向量表前 32 字的累加和。默认为使能。

2.5 在 TKStudio 中使用 EasyJTAG-H 仿真器

TKStudio 集成开发环境支持 EasyJTAG-H 仿真器。在 TKStudio 使用 EasyJTAG-H 仿真器进行硬件仿真调试，只需进行 H-JTAG 配置。TKStudio 支持多个编译工具链，每个编译工具链的工程配置窗体有所不同。在本教程中，以“GNU Tools for ARM”编译工具链下的 H-JTAG 配置为例，讲解在 TKStudio 中如何配置 H-JTAG。

（1）新建工程，选择编译工具为“GNU Tools for ARM”或打开现有的采用“GNU Tools for ARM”编译工具的工程，单击【工程配置】按钮或选择菜单“工程”->“配置目标工程”命令，即弹出工程配置对话框，并选择【Debug】项，如图 2.28 所示。

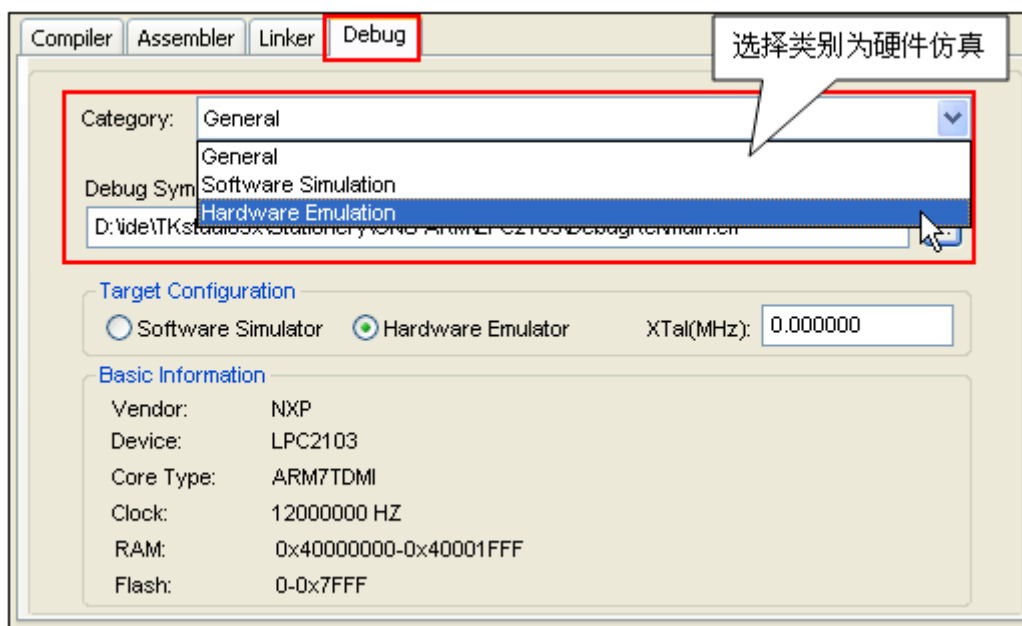


图 2.28 工程配置-Debug

在“Category”项选择为“Hardware Emulation”，显示界面如图 2.29 所示。

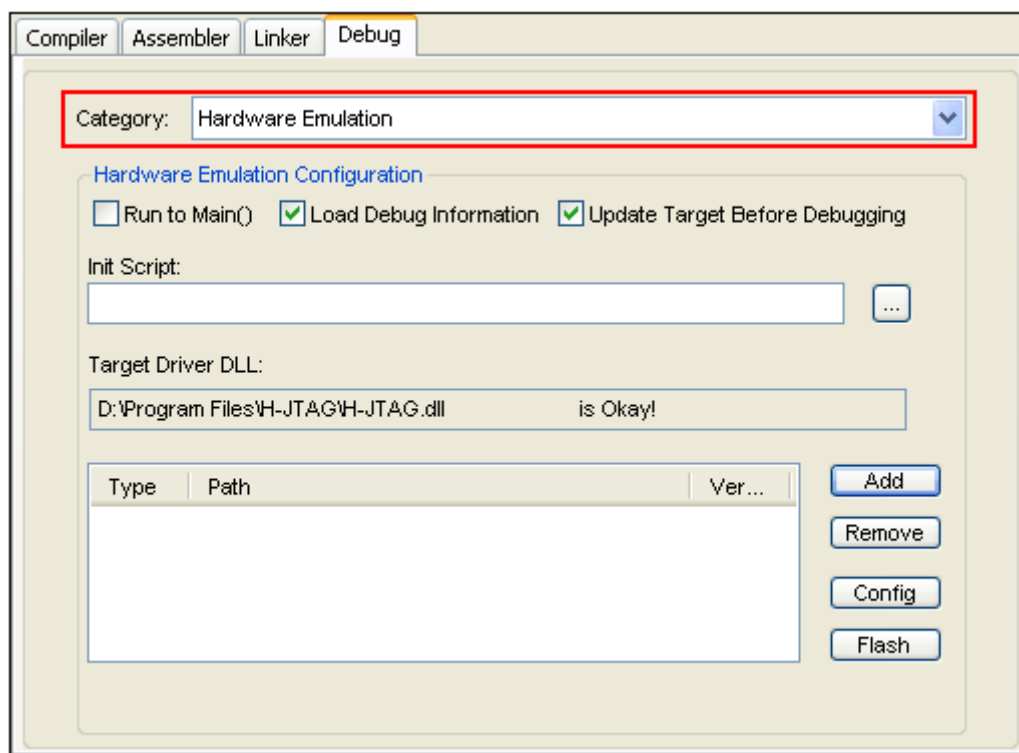


图 2.29 硬件调试设置

(2) 单击【Add】按钮，将弹出选择驱动类型对话框，如图 2.30 所示。

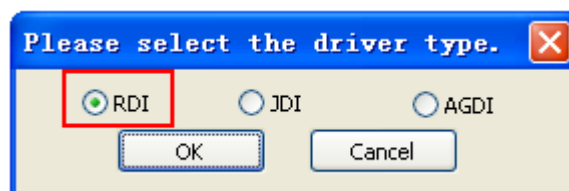


图 2.30 驱动类型选择

在如图 2.30 所示对话框中，选中 RDI 单选按钮（H-JTAG 是 RDI 接口），然后单击【OK】按钮，将弹出<打开>对话框。在打开对话框中，选择 H-JTAG 安装目录下的“H-JTAG.dll”文件，如图 2.31 所示。

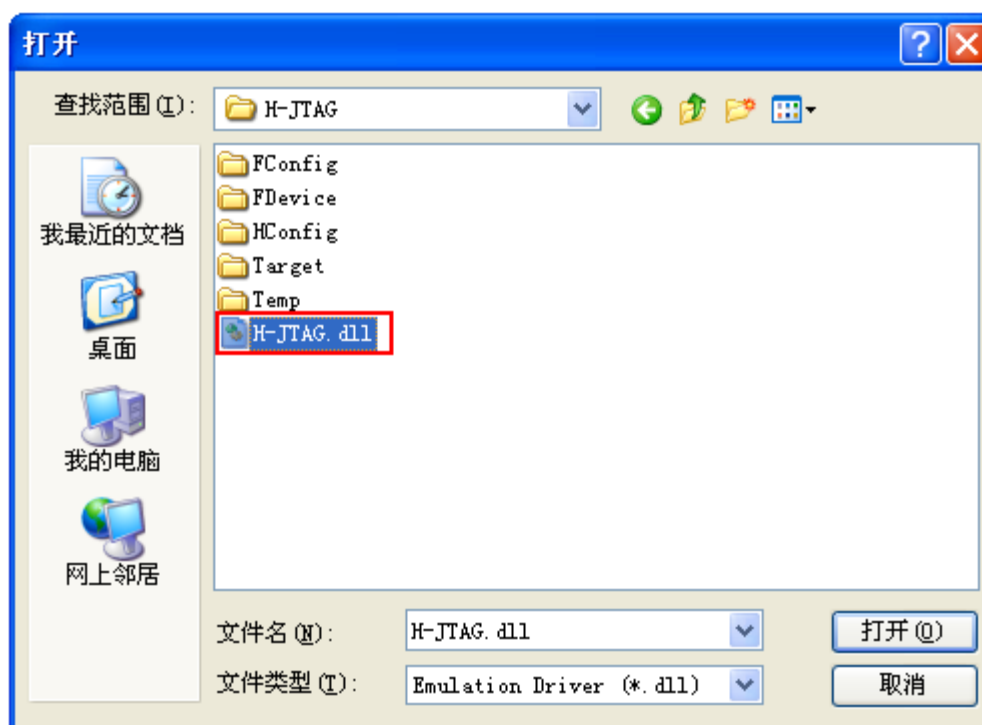


图 2.31 选择 H-JTAG.dll 文件

点击【打开】按钮后，H-JTAG 驱动将会添加到驱动列表中，并显示加载状态。如图 2.32 所示。

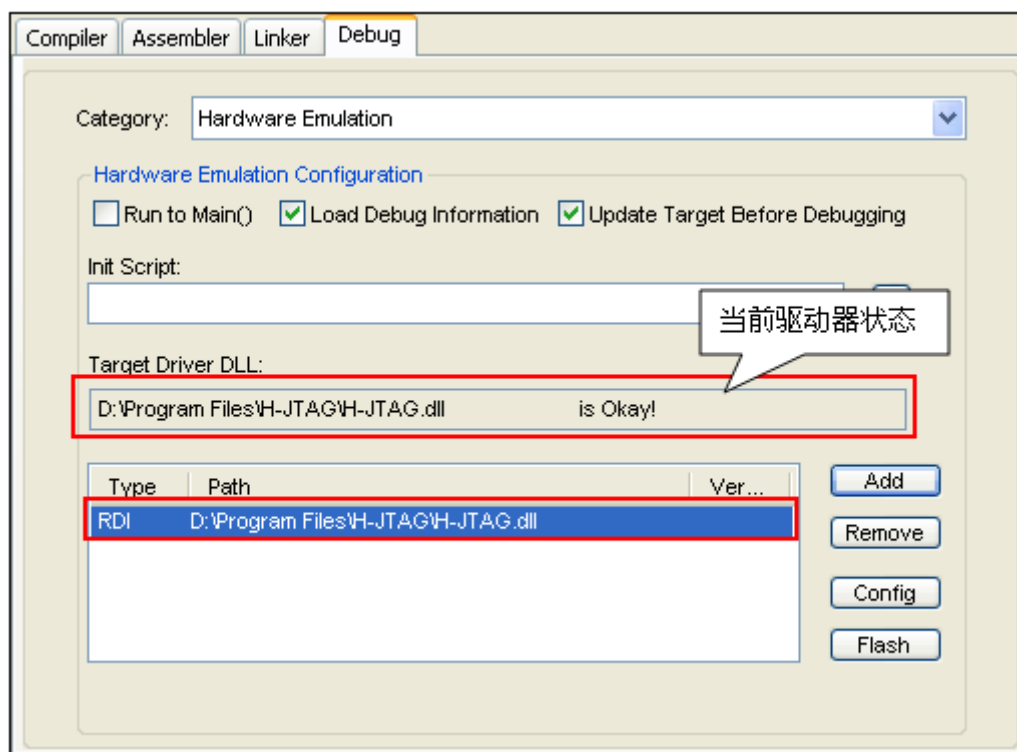


图 2.32 添加 H-JTAG.dll 到列表

(3) 双击列表中的 H-JTAG 项或单击【config】按钮，还可以单击【flash】按钮，若弹

出如图 2.33 所示对话框，说明 H-JTAG 的配置已成功。在 TKStudio 中，各个工具链采用相同的 H-JTAG 配置。即一个工具链配置好了，其他的工具链就不需再进行重复的配置工作。

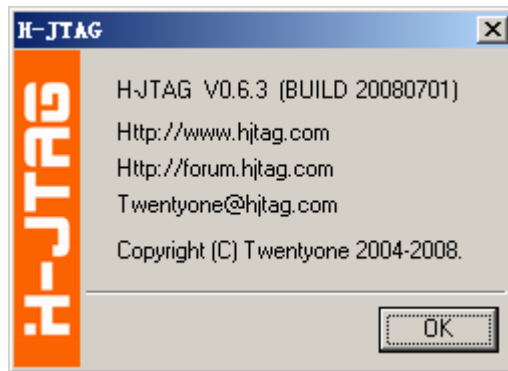


图 2.33 配置成功提示

第3章 实验例程

3.1 TKStudio 应用示例

3.1.1 实验目的

- 了解 TKStudio 集成开发环境的使用方法；
- 了解在 TKStudio 集成开发环境中使用 GNU ARM 编译方法；
- 了解模板文件使用方法。

3.1.2 实验内容

运用 TKStudio 工程模板建立一个工程，添加文件，编写 main()函数，使程序能够从汇编代码运行到 C 程序的 main()函数。

3.1.3 实验步骤

1. 启动 TKStudio 集成开发环境，选择【工程】->【从工程模板新建工程】，将弹出工程模板选择对话框。在对话框【工程】页中，展开“GNU ARM”节点，选择“LPC2103”工程模板，输入工程名为“Test”，选择工程存放路径，然后点击【确定】按钮，如图 3.1 所示。

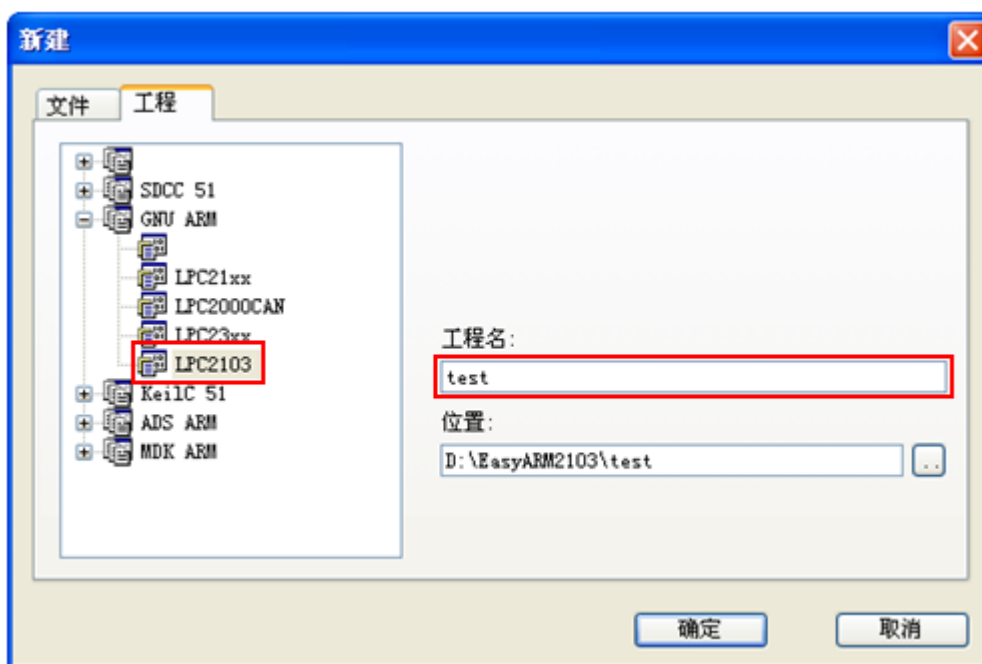


图 3.1 新建工程

经过这个简单的步骤，一个新的工程就建好了。建立好的 Test 工程如图 3.2 所示。

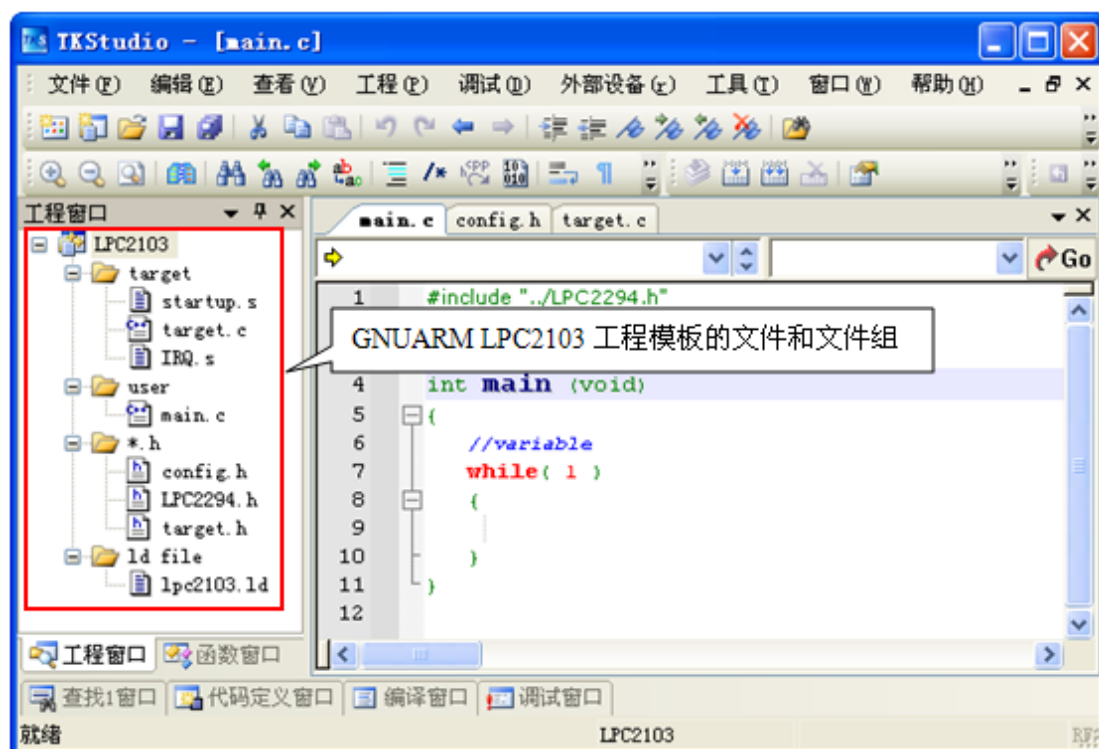


图 3.2 建立工程完成

2. 在 main.c 文件中编写 main()函数.这里 main()函数什么都不做, 只是为了测试, 里面是一个无限循环, 还有一个永远不会执行的 return 语句。如程序清单 3.1。

程序清单 3.1 main 函数

```
#include "../config.h" /* 包含头文件 */
int main(void)
{
    while(1);
    return 0;
}
```

3. 启动 H-JTAG Server 软件。在 H-JTAG Flash 软件主界面中, 选中 Flash Seletion 选项, 在右边列表中选择 NXP-LPC2103, 然后选择 Programing 项, 点击 Check 按钮。如图 3.3 所示。

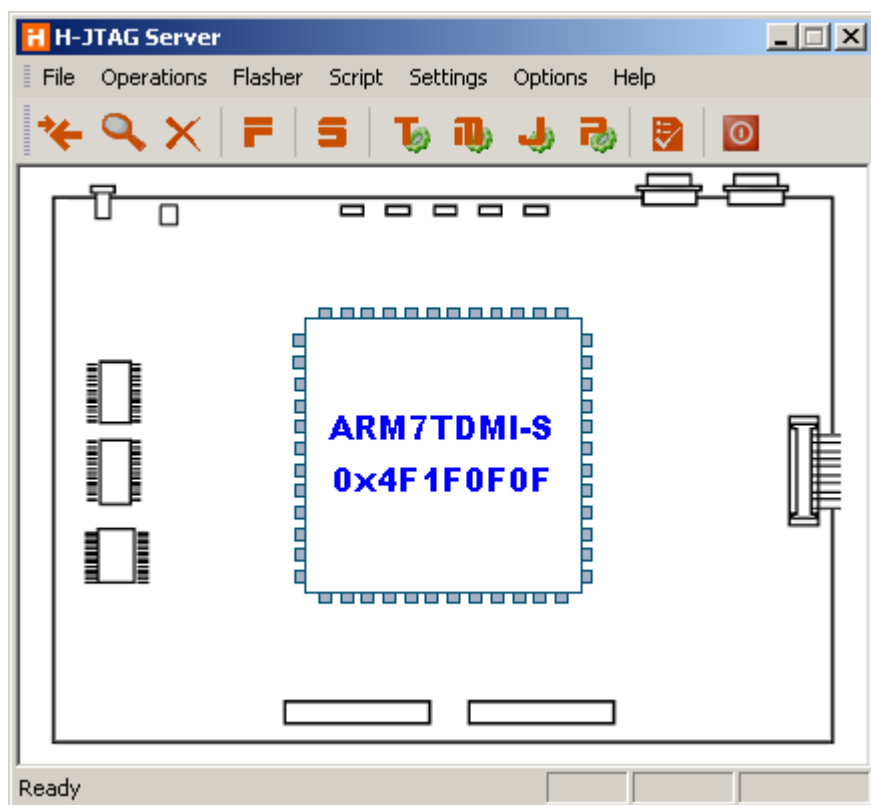


图 3.3 H-JTAG Server 启动

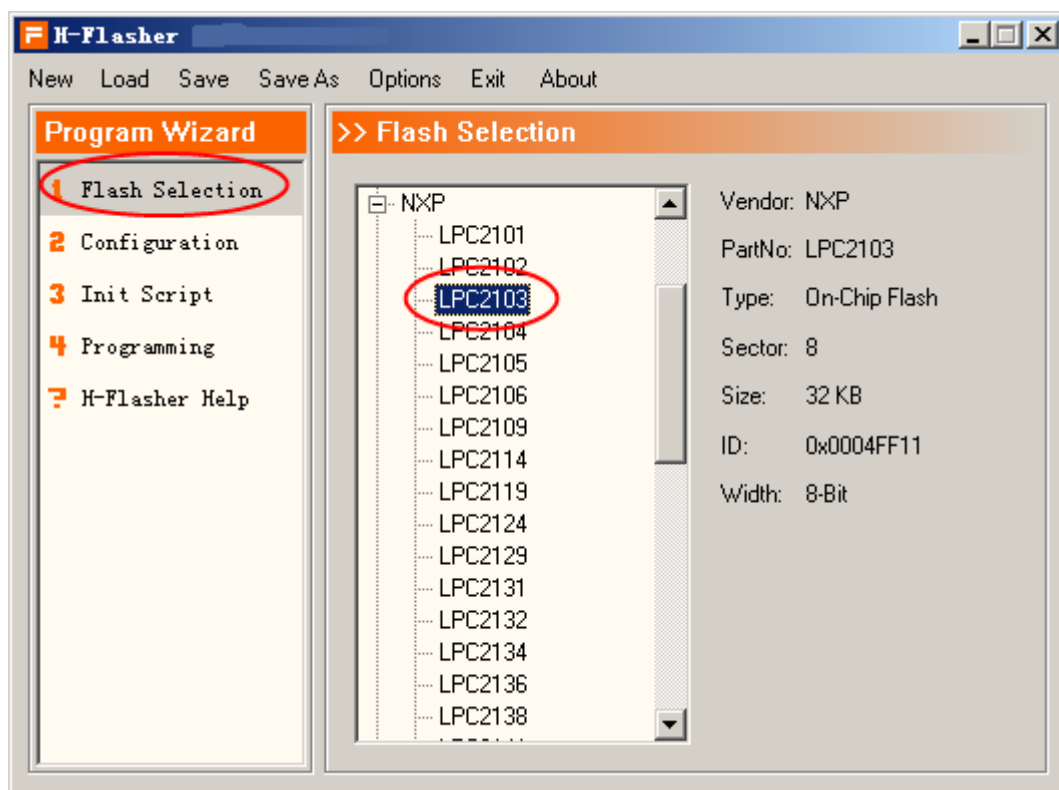


图 3.4 选择 CPU 型号

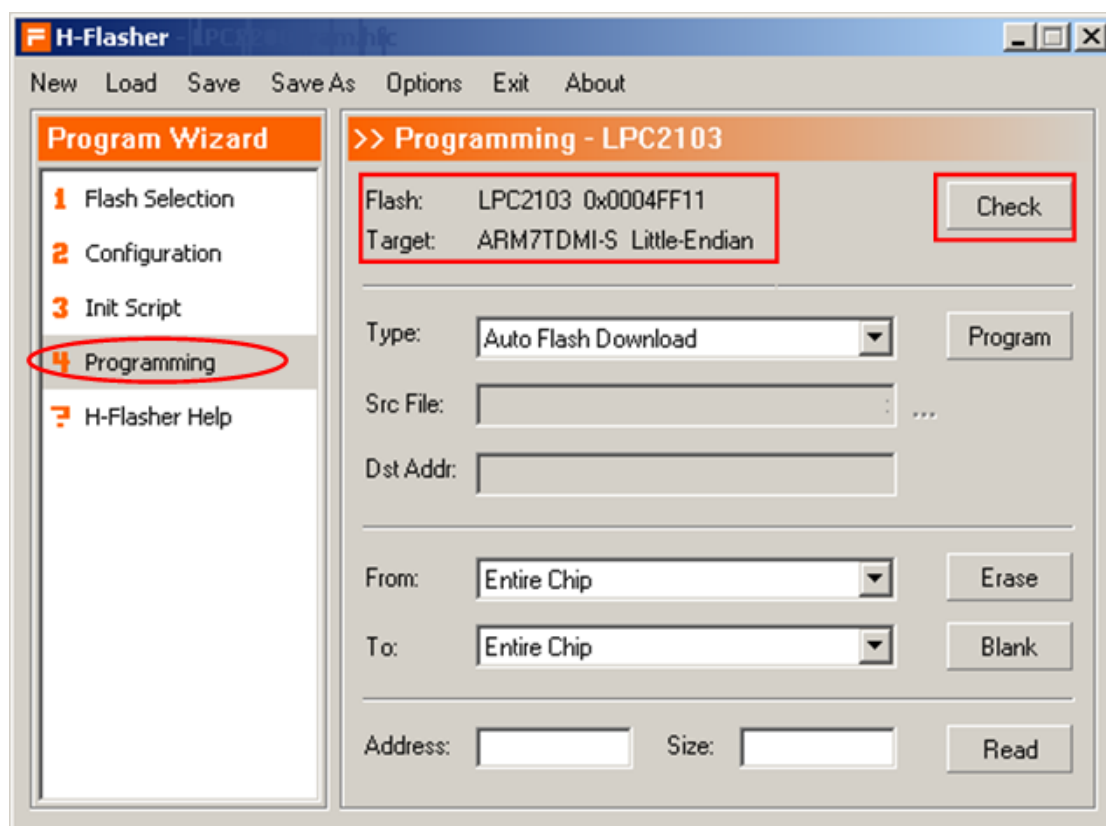


图 3.5 Check

如果显示出处理器的基本信息，说明连接正常，调试接口可以使用；如果显示出错信息，请检查 EasyARM2103 电源是否连接，PC 机并口是否与 H-JTAG 仿真器连接等等。

4. 点击工程配置图标 ，也可以从工程—配置目标工程启动工程配置命令。在配置对话框中，linker 项的 linker script file 选择分散加载描述文件：lpc2103.ld，debug 项的 Category 为 General 下，选择 Target Swicth 为 Hardware Emulator.，然后点击 Apply；在 Category 的下拉列表中选择 Hardware Emulator，将 Run to main()前的钩去掉（这里是为了观察从汇编代码开始运行），然后，测试 H-JTAG 驱动是否正常。如图 3.6 所示。

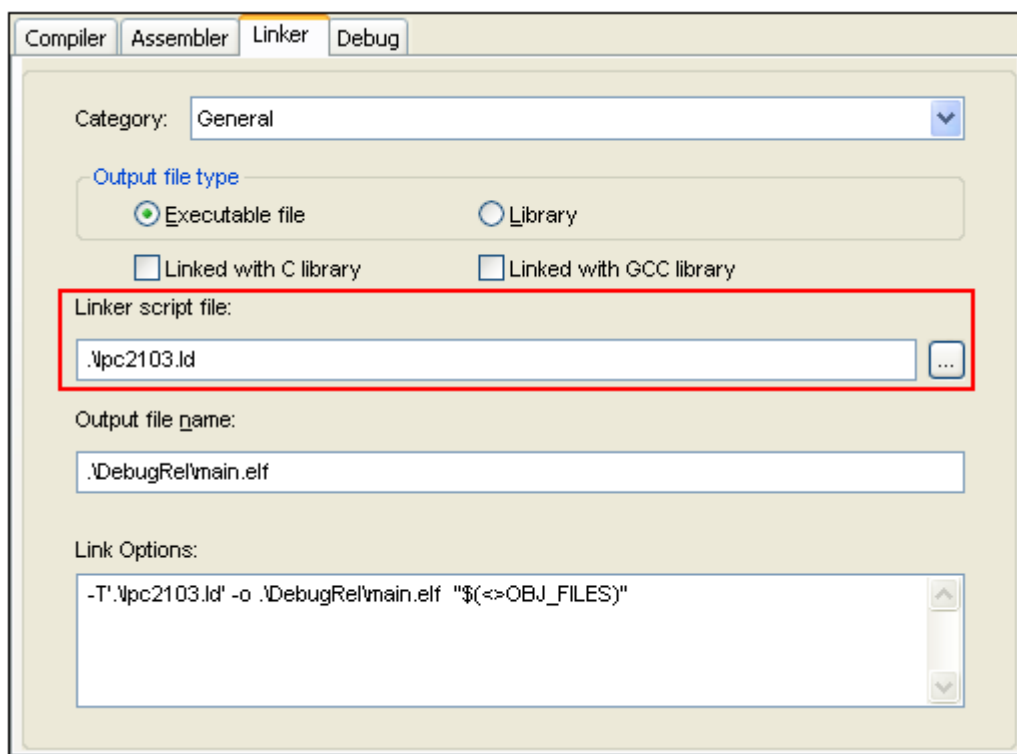


图 3.6 选择链接描述文件

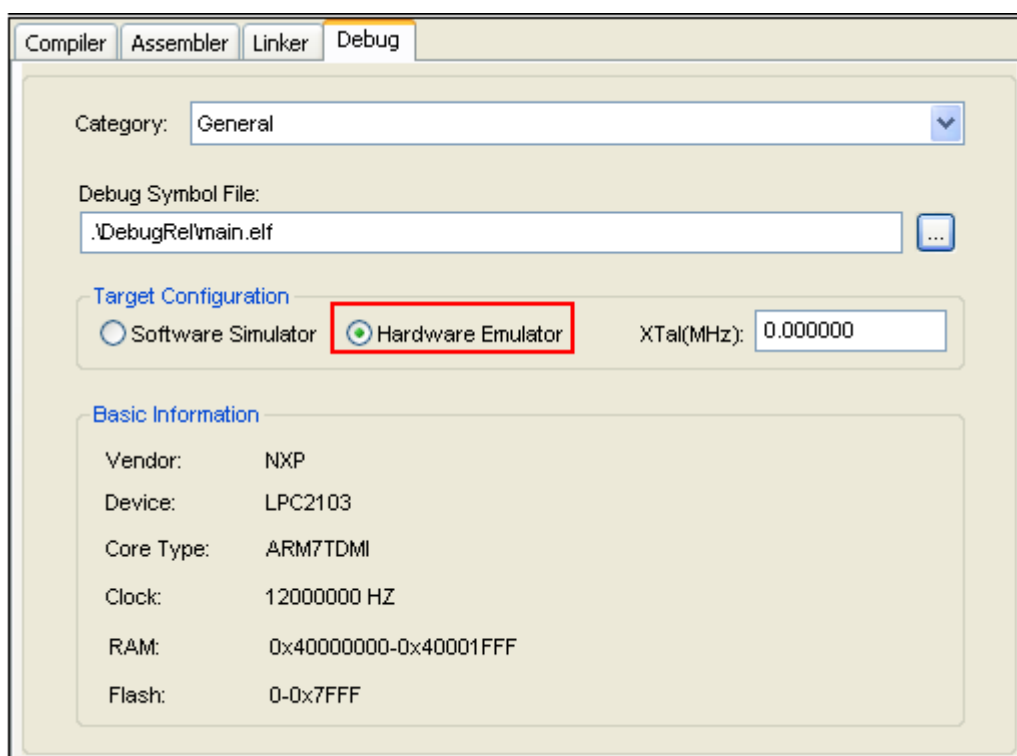


图 3.7 选择硬件仿真

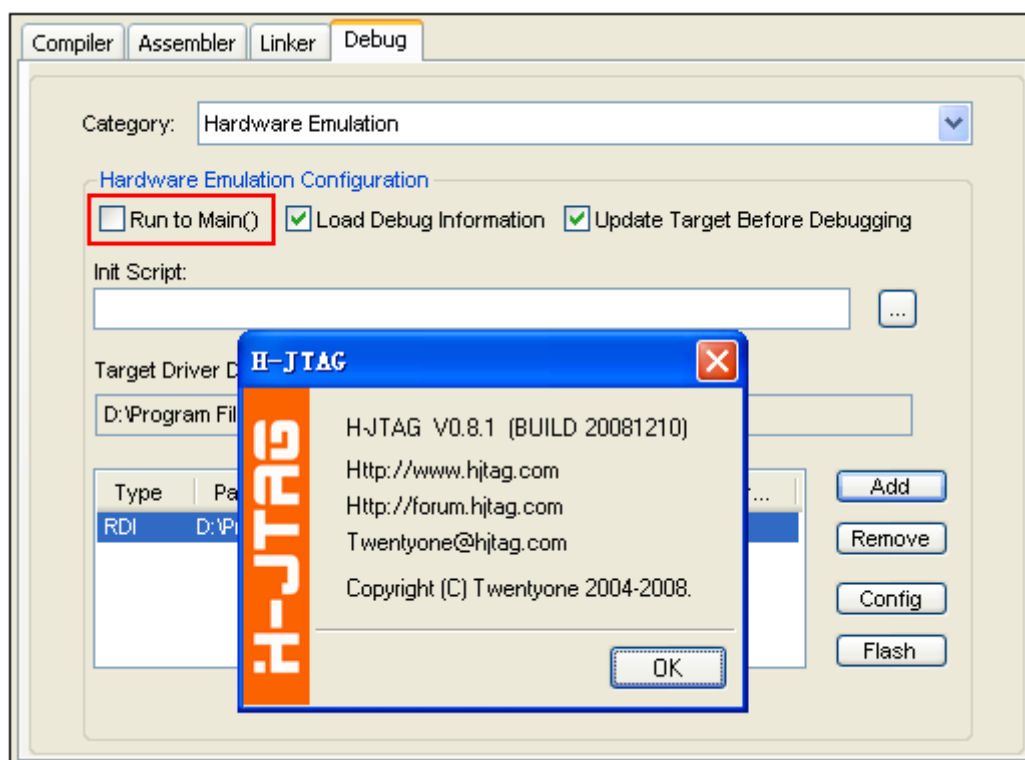


图 3.8 将钩去掉和测试 H-JTAG

5. 完成了以上工作就可以开始编译了。编译过程很简单，只需点击编译按钮即可。
6. 如果编译没有提示错误的话，就可以开始硬件仿真调试了。点击启动/停止调试按钮（也可以按快捷键 shift+F5），工程第一次调试时一般会弹出如下对话框：



图 3.9 调试对话框

此时单击 Locate Debug File 按钮，在弹出的对话框中选择工程目录\DebugRel 目录下的 main.elf 文件，单击打开按钮，如图 3.10 所示，然后点击【Retry】按钮，将会出现下载进度条，如图 3.11 所示。

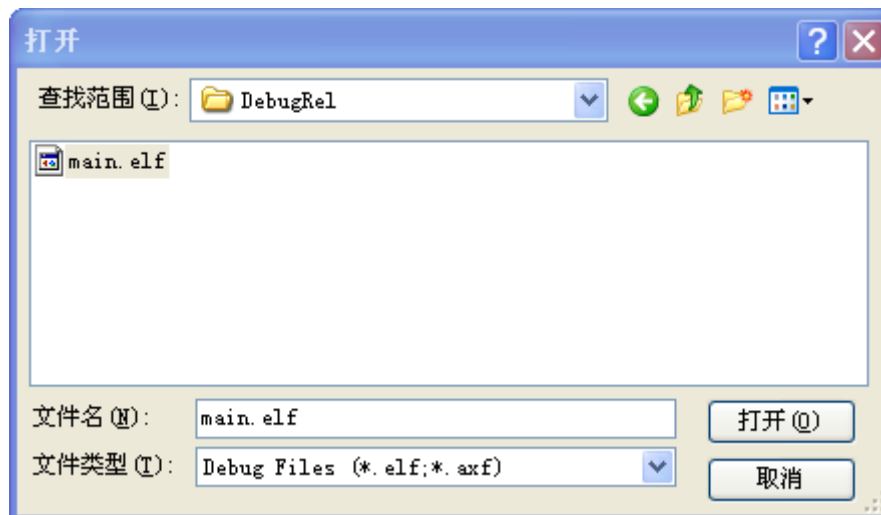


图 3.10 选择调试文件

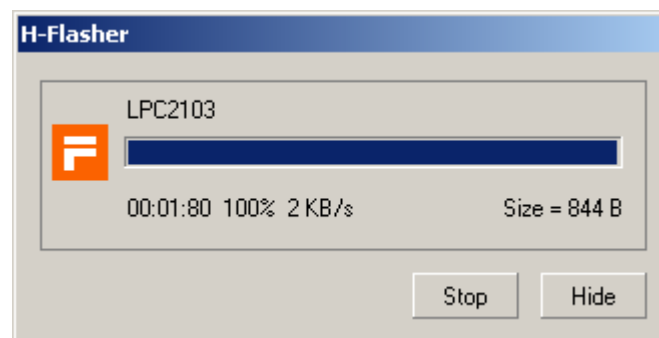


图 3.11 下载进度条

7. 单步运行程序(F11), 观察程序的运行过程。程序程序从汇编代码运行到 C 代码 main() 函数处。

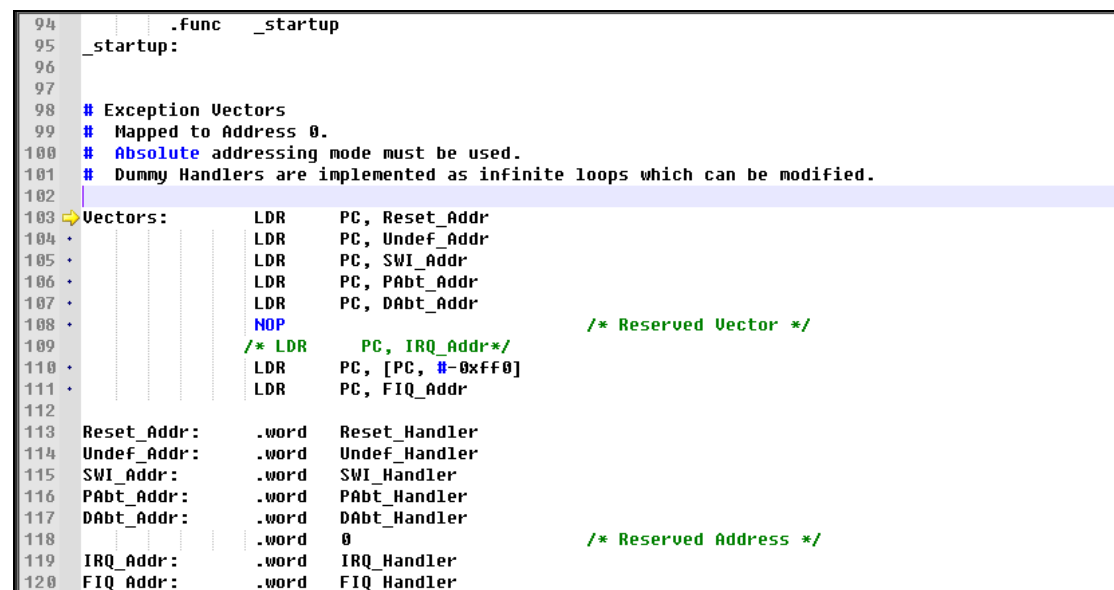


图 3.12 程序开始运行

```

259 •      LDRLO    R0, [R1], #4
260 •      STRLO    R0, [R2], #4
261 •      BLO      LoopRel
262
263
264 •      # Clear .bss section (Zero init)
265 •      MOV      R0, #0
266 •      LDR      R1, =__bss_start__
267 •      LDR      R2, =__bss_end__
268 •      LoopZI:  CMP      R1, R2
269 •      STRLO    R0, [R1], #4
270 •      BLO      LoopZI
271
272
273 •      # Enter the C code
274 •      ADR      LR, __main_exit
275 •      LDR      R0, =main
276 •      BX       R0
277
278 •      __main_exit: B      __main_exit
279
280
281 •      .size    _startup, . - _startup
282 •      .endfunc
283
284
285 •      .end

```

图 3.13 程序加载 main 函数地址

```

1
2      #include "../config.h" /*包含头文件*/
3
4      int main(void)
5      {
6          while(1);
7          return 0;
8      }

```

图 3.14 程序跳转到 main 函数执行

3.1.4 出现的问题与解决方法

1. 编译时提示: undefined reference to `_data'错误。

问题原因: 分散加载描述文件没有链接进来。

解决方法: 在工程配置对话框的 linker 选项中的 linker script file 中选择分散加载描述文件。

2. 启动调试时提示:

RDI fatal error:

Could not find h-jtag server. Please make sure that h-jtag server has been started properly.

问题原因: H-JTAG 软件没有启动。

解决方法: 启动 H-JTAG 软件, 并选择 NXP 的 LPC2103 处理器, 然后 Check。

3. 调试时, 程序直接运行到 main()函数, 看不到在汇编代码运行的过程。

问题原因: 工程设置了程序调试时运行到 main()。

解决方法: 在工程配置对话框的硬件仿真中, 将 Run to main()前的钩去掉。

3.2 GPIO 实验

3.2.1 普通低速 GPIO 实验

1. 实验目的

- 熟悉 TKStudio 集成开发环境的使用方法；
- 熟悉 LPC2103 ARM7 微控制器的 GPIO 控制。

2. 实验内容

用普通方式控制 EasyARM2103 开发板上 LED1 灯闪烁。

3. 实验要求

- 仔细阅读 LPC2103 芯片手册 GPIO 部分内容；
- 仔细阅读 EasyARM2103 电路手册，了解硬件结构。

4. 实验思路

- 1) 设置 P0.17 引脚为 GPIO 功能（P0.17 引脚控制 LED1），通过设置 PINSEL1 位 3:2=00B 实现；
- 2) 设置 P0.17 引脚为输出（设置 IO0DIR 位 17 为 1）；
- 3) 向 P0.17 引脚输出低电平，点亮 LED1（设置 ID0CLR 位 17 为 1）；
- 4) 向 P0.17 引脚输出高电平，熄灭 LED1（设置 ID0SET 位 17 为 1）。

5. 实验步骤

- 1) 短接跳线 JP4；
- 2) 其他步骤同第一个实验；
- 3) 在 main.c 中编写 GPIO 控制 LED1 显示代码。

6. 实验参考程序

程序清单 3.2 普通低速 GPIO 实验

```
#include "config.h"

#define LED1 1 << 17 /* P0.17 控制 LED1 */
/*****

** 函数名称: DelayNS
** 函数功能: 延时函数
** 输入参数: uiDly 值越大，延时时间越长
** 输出参数: 无
*****/

void DelayNS (uint32 uiDly)
{
    uint32 i;
    for (; uiDly > 0; uiDly--){
        for(i = 0; i < 50000; i++);
    }
}

/*****
** 函数名称: main
```

```

** 函数功能: 跳线 JP4 短接, LED1 闪烁
** 输入参数: 无
** 输出参数: 无
*****/
int main (void)
{
    PINSEL1 = PINSEL1 & ~(0x03 << 2);          /* 将 P0.17 设置为 GPIO */
    IO0DIR  = LED1;                             /* 设置 LED 控制口为输出 */
    IO0SET  = LED1;                             /* LED1 熄灭 */

    while (1) {
        IO0SET = LED1;                          /* LED1 熄灭 */
        DelayNS(50);                            /* 延时 */
        IO0CLR = LED1;                          /* LED1 点亮 */
        DelayNS(50);                            /* 延时 */
    }
    return 0;
}

```

3.2.2 高速 GPIO 实验

1. 实验目的

- 熟悉 TKStudio 集成开发环境的使用方法;
- 熟悉 LPC2103 ARM7 微控制器的高速 GPIO 控制。

2. 实验内容

用高速方式控制 EasyARM2103 开发板上 LED1、LED2、LED3、LED4 灯交替闪烁。

3. 实验要求

- 仔细阅读 LPC2103 芯片手册 GPIO 部分内容;
- 仔细阅读 EasyARM2103 电路手册, 了解硬件结构。

4. 实验思路

- 1) 设置 P0.17、P0.18、P0.19、P0.20 引脚为 GPIO 功能 (P0.17-LED1, P0.18-LED2, P0.19-LED3, P0.20-LED4), 通过设置 PINSEL1 寄存器位 9: 2 值为 0 实现;
- 2) 将系统控制和状态标志寄存器 SCS 位 0 置 1, 选择高速 GPIO 模式;
- 3) 设置 P0.17、P0.18、P0.19、P0.20 引脚为输出, 通过设置 FIO0DIR 寄存器实现;
- 4) 向引脚输出高电平, 灯熄灭 (将 FIO0SET 对应位置位); 向引脚输出低电平, 灯点亮 (将 FIO0CLR 对应位清 0); 控制 LED1、LED3 和 LED2、LED4 交替显示, 实现交替闪烁功能。

5. 实验步骤

- 1) 短接跳线 JP4;
- 2) 其他步骤同第一个实验;
- 3) 在 main.c 中编写 GPIO 控制 LED1、LED2、LED3、LED4 交替显示代码。

6. 实验参考程序

程序清单 3.3 高速 GPIO 控制

```
#include "config.h"

# define    LED1    1 << 17                                /* P0.17 控制 LED1 */
# define    LED2    1 << 18                                /* P0.18 控制 LED2 */
# define    LED3    1 << 19                                /* P0.19 控制 LED3 */
# define    LED4    1 << 20                                /* P0.20 控制 LED4 */

/*****
** 函数名称: DelayNS
** 函数功能: 延时函数
** 输入参数: uiDly  值越大, 延时时间越长
** 输出参数: 无
*****/

void DelayNS (uint32 uiDly)
{
    uint32 i;
    for (; uiDly > 0; uiDly--){
        for(i = 0; i < 50000; i++);
    }
}

/*****
** 函数名称: main
** 函数功能: 跳线 JP4 短接, LED1 闪烁
** 输入参数: 无
** 输出参数: 无
*****/

int main (void)
{
    PINSEL1 = PINSEL1 & ~(0xFF << 2);                    /* 将 P0.17-P0.20 设置为 GPIO */
    SCS      = 0x01;                                       /* 设定为高速 GPIO 模式 */
    FIO0DIR = LED1 | LED2 | LED3 | LED4;                  /* 设置 LED 控制口为输出 */
    FIO0SET = LED1 | LED2 | LED3 | LED4;                  /* LED1 熄灭 */

    while (1) {
        FIO0SET = LED1 | LED3;                            /* LED1、LED3 熄灭 */
        FIO0CLR = LED2 | LED4;                            /* LED2、LED4 点亮 */
        DelayNS(50);                                       /* 延时 */
        FIO0CLR = LED1 | LED3;                            /* LED1、LED3 点亮 */
        FIO0SET = LED2 | LED4;                            /* LED2、LED4 熄灭 */
        DelayNS(50);                                       /* 延时 */
    }
}
```

```
return 0;
}
```

3.3 外部中断实验

3.3.1 实验目的

- 熟悉 TKStudio 集成开发环境的使用方法；
- 掌握向量中断控制器（VIC）的设置；
- 掌握外部中断引脚功能设置及外部中断的工作模式设置；
- 了解 GCC 编译环境下的中断服务函数的编写。

3.3.2 实验内容

用外部中断方式控制 LED1 显示（GPIO 方式为普通低速方式），按下 KEY1 时，点亮；再次按下时熄灭；再按下时点亮；如此循环控制。

3.3.3 实验要求

- 仔细阅读 LPC2103 芯片手册外部中断输入部分内容；
- 仔细阅读 EasyARM2103 电路手册，了解硬件结构。

3.3.4 实验思路

- 1) 编写中断服务程序，通过向 P0.17 输出高电平或者低电平控制 LED1 的亮与灭。每次进入中断服务函数，取 P0.17 的状态，然后取反输出；
- 2) 设置 P0.16 为外部中断 0 管脚，通过配置 PINSEL1 寄存器位 1: 0=01B 实现；
- 3) 设置 P0.17 为 GPIO 方式，通过配置 PINSEL1 寄存器位 3: 2=00B 实现，注意不要直接写，要采用读-修改-回写方式。设置为输出方式。这是控制 LED 的操作方法；
- 4) 使能 IRQ 中断；
- 5) 设置外部中断模式为低电平触发（EXTMODE 位 0 设置为 0 选择外部中断 0 为电平触发，EXTPOLAR 位 0 设置为 0 选择外部中断 0 为低电平触发）；
- 6) 配置 VIC 中断控制器。选择外部中断 0 中断方式为 IRQ（VICIntSelect 对应位设置为 0），为外部中断 0 分配向量中断 0（设置 VICVectCntl0），设置中断服务程序地址（设置 VICVectAddr0，注意与分配向量中断一致），然后使能外部中断 0（设置 VICIntEnable 位 14 值为 1）；
- 7) 清除外部中断 0 中断标志（设置 EXTINT 值为 1），等待中断到来。

3.3.5 实验步骤

1. 短接跳线 JP3；
2. 其他步骤同第一个实验；
3. 在 main.c 中编写外部中断控制程序。中断服务程序写法参见 3.3.6；
4. 在中断服务程序设置断点，调试时，观察按键 KEY1 按下时，程序是否执行到断点处。

3.3.6 GCC 中断服务程序编写说明

1. 寄存器保护与恢复

在执行中断服务函数前，都要对用户模式下的通用寄存器（r0-r12）、链接寄存器、程序

状态寄存器的内容进行保护,即将这些内容先保存到中断栈,等中断服务程序执行完成再从栈中进行恢复。有些编译器提供一些关键字供用户使用,编译器在编译的时候自动在中断服务函数第一条指令前加入将当前寄存器内容入栈中断栈的指令,在函数末尾加入恢复指令,如 ADS 编译器提供 `__irq` 关键字。而 GCC 编译器没有自动维护这些寄存器,这就要求用户自己来保存与恢复这些寄存器的内容。

在模板文件 `target.h` 中,我们已经编写了两个宏来对这些寄存器的内容进行保存与恢复。

```
#define ISR_ENTRY() asm volatile(" sub    lr,lr,#4\n" \
                                " stmfd sp!,{r0-r12,lr}\n" \
                                " mrs    r1, spsr\n" \
                                " stmfd sp!,{r1}")
```

宏 `ISR_ENTRY()`的作用是将通用寄存器 (r0-r12)、链接寄存器、程序状态寄存器内容保存到中断栈。宏 `ISR_ENTRY()`写在中断服务程序第一条语句前,即作为中断服务程序的第一条语句。

```
#define ISR_EXIT()  asm volatile(" ldmfd sp!,{r1}\n" \
                                " msr    spsr_c,r1\n" \
                                " ldmfd sp!,{r0-r12,pc}^")
```

宏 `ISR_EXIT()`作用是中断服务程序执行完成后将通用寄存器 (r0-r12)、链接寄存器、程序状态寄存器内容进行恢复,执行宏 `ISR_ENTRY()`的反操作。宏 `ISR_EXIT()`写在中断服务程序末尾,即作为中断服务程序的最后一条语句。

注意: 宏 `ISR_ENTRY()`与宏 `ISR_EXIT()`一定要配对使用。

2. 中断服务函数声明

GCC 中一般函数在执行用户第一条语句前都做了什么工作?

现在我们通过反汇编代码来查看一下究竟。以下是反汇编后函数名后,用户第一条语句前的汇编代码。

程序清单 3.4 反汇编代码

```
MOV    IP, SP
STMEA  SP!, {FP-IP,LR-PC}
SUB    FP, IP, #4          ; 0x4
SUB    SP, SP, #8          ; 0x8
STR    R0, [FP, #-20]      ; 0x14
;以下为函数末尾代码
SUB    SP, FP, #12         ; 0xc
LDMFD  SP, {FP,SP,PC}
```

从以上汇编代码可知,在执行用户第一条语句前程序对 FP、IP、LR、PC 寄存器内容进行了保存,最后在函数结束时对这些寄存器进行恢复。LR 寄存器保存的是函数返回地址。如果该函数是中断服务函数会发生什么情况呢?此时的 LR 是中断模式下的链接地址。正确做法是将 LR 减去偏移量才是中断返回地址,而编译器直接将 LR 入栈,等中断服务函数执行完成的时候,恢复 LR,那么程序返回到中断发生处执行就是错误的了。要避免这种情况的发生,就是不要编译器自动添加这些代码。GCC 提供这样的编译开关,就是 `__attribute__((naked))`。在函数声明时加入这条语句,就是告诉编译器,这个函数的寄存器与返回地址的保存动作由用户来完成,不用编译器管理了。这种函数也称为“裸函数”。中断

处理函数声明方式如下:

```
void XXXX_ISR(void) __attribute__((naked));
```

__attribute__((naked))只能在函数声明处添加,在函数实现代码中不能添加。中断处理函数定义形式如下:

```
void XXXX_ISR(void)
{
    ISR_ENTRY(); //加入这个宏
    //你的中断处理代码
    ISR_EXIT();
}
```

3.3.7 实验参考程序

程序清单 3.5 外部中断

```
#include "config.h"
#define LED 1 << 17
void Eint0IRQ(void) __attribute__((naked));

/*****
** 函数名称: Eint0IRQ
** 函数功能: 外部中断 0 服务程序
** 输入参数: 无
** 输出参数: 无
*****/

void Eint0IRQ(void)
{
    ISR_ENTRY(); /* 进入中断取反 LED */
    if ((IO0PIN & (1 << 17)) == 0) {
        IO0SET = 1 << 17; /* 熄灭发光二极管 */
    }
    else {
        IO0CLR = 1 << 17; /* 点亮发光二极管 */
    }

    while((IO0PIN & (1 << 16)) == 0); /* 等待按键松开 */

    EXTINT = 0x01; /* 清中断标志 */
    VICVectAddr = 0x00; /* 通知 VIC 中断处理结束 */
    ISR_EXIT();
}

/*****
** 函数名称: main
** 函数功能: P0.16 低电平触发外部中断主函数
** 输入参数: 无
** 输出参数: 无
*****/
```

```

*****/
int main (void)
{
    PINSEL1 = PINSEL1 & (~0x03);
    PINSEL1 = PINSEL1 | 0x01;                /* 设置 P0.16 为外部中断 0 管脚 */
    PINSEL1 = PINSEL1 & ~(0x03 << 2);        /* 设置 P0.17 为 GPIO 功能 */
    IO0DIR  = LED;                          /* 设置 P0.17 为输出 */
    IO0SET  = LED;                          /* 设置输出为高电平 */
    IRQEnable();                            /* IRQ 中断使能 */

    EXTMODE= 0x00;                          /* 设置外部中断为低电平触发 */
    EXTPOLAR = 0x00;

    VICIntSelect = 0 << 14;                  /* 选择 EINT0 为 IRQ 中断 */
    VICVectCntl0 = 0x20 | 14;                /* 将外部中断 0 分配给向量中断 0 */
    VICVectAddr0 = (uint32)Eint0IRQ;          /* 设置中断服务程序地址 */
    VICIntEnable = 1 << 14;                  /* 使能 EINT0 中断 */

    EXTINT = 0x01;                          /* 清除 EINT0 中断标志 */
    while(1);
    return 0;
}

```

3.4 定时器实验

3.4.1 定时器实验 1

1. 实验目的

- 熟悉 TKStudio 集成开发环境的使用方法；
- 熟悉 LPC2103 微控制器的定时器的基本设置及应用。

2. 实验内容

采用定时器 0 匹配产生中断，在中断服务程序中控制 LED1 闪烁。

3. 实验要求

- 仔细阅读 LPC2103 芯片手册定时器部分内容；
- 仔细阅读 EasyARM2103 电路手册，了解硬件结构。

4. 实验思路

- 1) 编写中断服务程序，通过向 P0.17 输出高电平或者低电平控制 LED1 的亮与灭。每次进入中断服务函数，取 P0.17 的状态值，然后取反输出；
- 2) 设置 P0.17 引脚为 GPIO 功能（设置 PINSEL1 位 3: 2=00B），设置 P0.17 为输出（设置 IO0DIR 位 17 值为 1）；
- 3) 初始化定时器 0。不设置时钟分频（设置 T0PR = 0），设置匹配后复位方式，并产生中断（设置 T0MCR），设置匹配值（设置 T0MR0），然后清除定时器 0 中断标志（设置 T0IR），并启动定时器 0（设置 T0TCR）；
- 4) 使能 IRQ 中断；

- 5) 配置 VIC 中断控制器。选择定时器 0 中断方式为 IRQ (VICIntSelect 对应位设置为 0)，为定时器 0 分配向量中断 0 (设置 VICVectCntl0)，设置中断服务程序地址 (VICVectAddr0，注意与分配向量中断一致)，然后使能定时器 0 中断 (设置 VICIntEnable)。

5. 实验步骤

- 1) 短接跳线 JP4;
- 2) 其他步骤同第一个实验;
- 3) 在 main.c 中编写定时器 0 匹配产生中断程序，中断服务程序写法参见 3.4.6;
- 4) 调试时，在中断服务程序设置断点，观察是否执行到断点处。

6. 实验参考程序

程序清单 3.6 定时器 0 匹配产生中断

```
#include "../config.h"

#define LED (1 << 17)

void Timer0ISR (void) __attribute__((naked)); /*GCC 中断处理函数声明 */
/*****

** 函数名称: Timer0_ISR
** 函数功能: 定时器 0 中断主程序
** 输入参数: 无
** 输出参数: 无
*****/

void Timer0ISR (void)
{
    ISR_ENTRY();
    if ((IO0PIN & LED) == 0) {
        IO0SET = LED; /* 点亮发光二极管 */
    }
    else IO0CLR = LED; /* 熄灭发光二极管 */
    T0IR = 0x01; /* 清除中断标志 */
    VICVectAddr = 0x00; /* 中断向量结束 */
    ISR_EXIT();
}
/*****

** 函数名称: Timer0Init
** 函数功能: 定时器 0 初始化
** 输入参数: 无
** 输出参数: 无
*****/

void Timer0Init(void)
{
    T0TCR = 0x02; /* 定时器 0 复位 */
}
```

```

TOPR    = 0;                                /* 不设时钟分频          */
TOMCR   = 0x03;                             /* 匹配后复位 TC，并产生中断 */
TOMR0   = Fpclk / 2;                        /* 设置 0.5 秒匹配值      */
TOIR    = 0x01;                             /* 清除中断标志          */
TOTCR   = 0x01;                             /* 启动定时器 0          */
}

/*****

** 函数名称: main
** 函数功能: 定时器 0 匹配产生中断
** 输入参数: 无
** 输出参数: 无

*****/

int main (void)
{
    PINSEL1 = PINSEL1 & 0xFFFFFFF3;         /* 选择 P0.17 的 GPIO 功能 */
    IO0DIR  = LED;                           /* 设置 P0.17 为输出      */
    IO0SET  = LED;                           /* 设置输出高电平        */
    Timer0Init();                            /* 定时器 0 初始化        */
    IRQEnable();                             /* IRQ 中断使能          */

    VICIntSelect = VICIntSelect & ~(1 << 4); /* 定时器 0 分配为 IRQ 中断 */
    VICVectCntl0 = 0x20 | 4;                 /* 定时器 0 分配为向量 IRQ 通道 0 */
    VICVectAddr0 = (uint32) Timer0ISR;        /* 分配中断服务程序地址  */
    VICIntEnable = 1 << 4;                   /* 定时器 0 中断使能      */

    while(1);
    return 0;
}

```

3.4.2 定时器实验 2

1. 实验目的

- 熟悉 TKStudio 集成开发环境的使用方法；
- 熟悉 LPC2103 微控制器的定时器的基本设置及应用。

2. 实验内容

定时器 1 匹配输出翻转，控制 LED3 闪烁。

3. 实验要求

- 仔细阅读 LPC2103 芯片手册定时器部分内容；
- 仔细阅读 EasyARM2103 电路手册，了解硬件结构。

4. 实验思路

- 1) 选择 P0.19 引脚为 MAT1.2 功能（设置 PINSEL1 位 7：6=10B）。P0.19 引脚的电平变化将会使 LED3 闪烁；
- 2) 初始化定时器 1。不设时钟分频（设置 T1PR=0），设置 T1MR1 匹配后复位 T1TC，设置匹配翻转（设置 T1EMR），设置设置匹配值（设置 T1MR0），然后清除中断标

志（设置 T1IR），启动定时器 1；

3) 等待匹配输出翻转。

5. 实验步骤

1) 短接跳线 JP4；

2) 其他步骤同第一个实验；

3) 在 main.c 中编写定时器匹配输出程序。

6. 实验参考程序

程序清单 3.7 定时器匹配输出

```
#include "../config.h"

/*****
** 函数名称: Timer1Init
** 函数功能: 定时器 1 初始化
** 输入参数: 无
** 输出参数: 无
*****/

void Timer1Init(void)
{
    T1TCR = 0x02;          /* 定时器 1 复位          */
    T1PR = 0;              /* 不设时钟分频          */
    T1MCR = 0x02;          /* 设置 T1MR1 匹配后复位 T1TC */
    T1EMR = 0x03 << 8;     /* 匹配翻转              */
    T1MR0 = Fpclk / 2;     /* 设置 0.5 秒匹配值      */
    T1IR = 0x01;           /* 清除中断标志          */
    T1TCR = 0x01;          /* 启动定时器 1          */
}

/*****
** 函数名称: main
** 函数功能: 定时器 1 匹配输出翻转主程序
** 输入参数: 无
** 输出参数: 无
*****/

int main (void)
{
    PINSEL1 = PINSEL1 & (0x03 << 6); /* 选择 MAT1.2 输出      */
    PINSEL1 = PINSEL1 | (0x02 << 6); /* 选择 MAT1.2 输出      */
    Timer1Init();                 /* 定时器 1 初始化        */
    while(1);
    return 0;
}
```

3.4.3 定时器实验 3

1. 实验目的

- 熟悉 TKStudio 集成开发环境的使用方法;
- 熟悉 LPC2103 微控制器的定时器的基本设置及应用。

2. 实验内容

定时器 1PWM 输出, 控制 LED3, LED4 显示。

3. 实验要求

- 仔细阅读 LPC2103 芯片手册定时器部分内容;
- 仔细阅读 EasyARM2103 电路手册, 了解硬件结构。

4. 实验思路

- 1) 选择 P0.19 引脚为 MAT1.2 输出功能、P0.20 引脚为 MAT1.3 输出功能(通过配置 PINSEL1 寄存器位 7: 6=10b,位 9: 8=10B 实现)。P0.19-LED3, P0.20-LED4;
- 2) 定时器 1 初始化。不设时钟分频(设置 T1PR=0),使能 PWM 输出(设置 PWM1CON),设置匹配后复位(设置 T1MCR),设置设置匹配值(设置 T1MR0),设置 PWM 输出的周期(设置 T1MR0),设置 PWM1.2 输出占空比,设置 PWM1.3 输出占空比,启动定时器 1;
- 3) 等待 PWM 输出。

5. 实验步骤

- 1) 短接跳线 JP4;
- 2) 其他步骤同第一个实验;
- 3) 在 main.c 中编写定时器 1PWM 输出程序。

6. 实验参考程序

程序清单 3.8 定时器 1PWM 输出

```
#include "../config.h"

/*****

** 函数名称: Timer1Init
** 函数功能: 定时器 1 初始化
** 输入参数: 无
** 输出参数: 无

*****/

void Timer1Init(void)
{
    T1TCR    = 0x02;                /* 定时器 0 复位          */
    T1PR     = 0;                   /* 不设时钟分频          */
    PWM1CON  = 0x0C;                /* 使能 PWM 输出          */
    T1MCR    = 0x02;                /* 设置 T0MR0 匹配后复位 T0TC */
    T1MR0    = Fpclk / 2000;        /* 设置 PWM 输出的周期    */
    T1MR2    = (Fpclk / 2000) / 2;  /* 设置 PWM1.2 输出占空比为 50% */
    T1MR3    = ((Fpclk / 2000) / 4) * 3; /* 设置 PWM1.3 输出占空比为 25% */
    T1TCR    = 0x01;                /* 启动定时器 1          */
}
```

```

}
/*****
** 函数名称: main
** 函数功能: 定时器 1PWM 输出
** 输入参数: 无
** 输出参数: 无
*****/

int main (void)
{
    PINSEL1 = (PINSEL1 & ~(0x03 << 6)) | (0x02 << 6);    /* 选择 MAT1.2 输出      */
    PINSEL1 = (PINSEL1 & ~(0x03 << 8)) | (0x02 << 8);    /* 选择 MAT1.3 输出      */
    Timer1Init();                                          /* 定时器 1 初始化      */
    while(1);
    return 0;
}

```

3.4.4 定时器实验 4

1. 实验目的

- 熟悉 TKStudio 集成开发环境的使用方法；
- 熟悉 LPC2103 微控制器的定时器的基本设置及应用。

2. 实验内容

定时器 2 捕获产生中断,捕获 P0.27 引脚。按键 KEY3 按下时 LED1 亮,再按下时 LED1 熄灭。

3. 实验要求

- 仔细阅读 LPC2103 芯片手册定时器部分内容；
- 仔细阅读 EasyARM2103 电路手册,了解硬件结构。

4. 实验思路

- 1) 编写定时器 2 中断服务函数,如果 LED1 亮就熄灭,否则点亮;
- 2) 选择 P0.17 引脚为 GPIO 功能(设置 PINSEL1 位 3: 2=00B),设置 P0.17 为输出(设置 IO0DIR 位 17 为 1)。P0.17 控制 LED1;
- 3) 定时器 2 初始化。选择 P0.27 为 CAP2.0 功能(设置 PINSEL1 位 23: 22=10b),复位定时器 2(设置 T2TCR = 0x02),清除中断标志(设置 T2IR),设置上升沿捕获,并产生中断(设置 T2CCR),启动定时器 2(设置 T2TCR);
- 4) IRQ 中断使能;
- 5) 配置 VIC 向量中断控制器。定时器 2 分配为 IRQ 中断(设置 VICIntSelect 位 26 为 0),定时器 2 分配为向量 IRQ 通道 0(设置 VICVectCntl0),分配中断服务程序地址(设置 VICVectAddr0 值为中断服务程序地址),定时器 2 中断使能(设置 VICIntEnable 位 26 为 1);
- 6) 按下按键 KEY3,观察结果是否正确。

5. 实验步骤

- 1) 短接跳线 JP4,断开跳线 JP3;

- 2) 用一根线将 P0.27 与 KEY3 跳线左端接线相连接;
- 3) 其他步骤同第一个实验;
- 4) 在 main.c 中编写定时器 2 中断服务程序和捕获输出程序;
- 5) 测试程序是否正确。

6. 实验参考程序

程序清单 3.9 定时器 2 捕获中断

```
#include "../config.h"
#define LED 1 << 17
void Timer2ISR (void) __attribute__((naked));

/*****
** 函数名称: Timer2ISR
** 函数功能: 中断服务函数
** 输入参数: 无
** 输出参数: 无
*****/

void Timer2ISR (void)
{
    ISR_ENTRY();
    if ((IO0PIN & LED) == 0) {
        IO0SET = LED; /* 熄灭发光二极管 */
    }
    else IO0CLR = LED;
    T2IR = 0x10; /* 清除中断标志 */
    VICVectAddr = 0x00; /* 中断向量结束 */
    ISR_EXIT();
}

/*****
** 函数名称: Timer2Init
** 函数功能: 定时器 2 初始化
** 输入参数: 无
** 输出参数: 无
*****/

void Timer2Init(void)
{
    PINSEL1 = PINSEL1 & ~(0x03<<22) | (0x02<<22); /* 选择 CAP2.0 功能 */
    T2TCR = 0x02; /* 复位定时器 2 */
    T2IR = 0x10; /* 清除中断标志 */
    T2CCR = 0x06; /* 设置上升沿捕获, 并产生中断 */
    T2TCR = 0x01; /* 启动定时器 */
}

/*****
** 函数名称: main
** 函数功能: 定时器捕获中断主程序
*****/
```

```

** 输入参数: 无
** 输出参数: 无
*****/
int main (void)
{
    PINSEL1 = PINSEL1 & 0xFFFFFFF3;          /* 选择 P0.17 的 GPIO 功能 */
    IO0DIR  = LED;                            /* 设置 P0.17 为输出 */
    IO0SET  = LED;                            /* 设置输出高电平 */
    Timer2Init();                             /* 定时器 2 初始化 */
    IRQEnable();                              /* IRQ 中断使能 */

    /* * 设置向量中断控制器 */
    VICIntSelect = VICIntSelect & ~(1 << 26); /* 定时器 2 分配为 IRQ 中断 */
    VICVectCntl0 = 0x20 | 26;                 /* 定时器 2 分配为向量 IRQ 通道 0 */
    VICVectAddr0 = (uint32) Timer2ISR;         /* 分配中断服务程序地址 */
    VICIntEnable = 1 << 26;                   /* 定时器 2 中断使能 */
    while(1);
    return 0;
}

```

3.5 UART 实验

3.5.1 UART 实验 1

1. 实验目的

- 熟悉 TKStudio 集成开发环境的使用方法;
- 掌握 UART 各个控制寄存器的设置, 并能使用串口向 PC 机发送数据。

2. 实验内容

使用 UART0 从 PC 机接收 20 个字符, 然后向 PC 机发送接收到的字符串。UART0 使用查询方式通讯, 通信波特率为 115200, 8 位数据位, 1 位停止位, 无奇偶校验。

3. 实验要求

- 仔细阅读 LPC2103 芯片手册 UART0 部分内容;
- 仔细阅读 EasyARM2103 电路手册, 了解硬件结构。

4. 实验思路

- 1) 设置 I/O 连接到 UART0 (设置 PINSEL0 位 3: 0=0101B);
- 2) UART0 初始化。设置为 8 位数据位, 1 位停止位, 无奇偶校验 (U0LCR), 波特率为 115200 (U0DLM, U0DLL);
- 3) 等待接收标志置位 (U0LSR), 然后读取数据 (U0RBR);
- 4) 在 PC 机上向 COM1 口发送 20 个字符数据;
- 5) 读取 20 个字符后, 向 PC 机发送字符串。通过写 U0THR 实现, 写一个字符, 查询一次 PC 是否已接收; 已接收, 继续发送, 直至完成发送。

5. 实验步骤

- 1) 用一根串口延长线将 PC 串口与 EasyARM UART0 相连接;

- 2) 其他步骤同第一个实验;
- 3) 在 main.c 中编写 UART0 发送与接收程序;
- 4) 启动调试;
- 5) 在 PC 机上向串口发送 20 个字符数据;
- 6) 观察 PC 机是否接收到 UART0 发送过来的数据。

6. 实验参考程序

程序清单 3.10 UART0 发送与接收数据

```
#include "../config.h"
#include "string.h"
#include <stdio.h>
#define UART_BPS 115200 /* 串口通信波特率 */
/*****
** 函数名称: DelayNS
** 函数功能: 延时函数
** 输入参数: uiDly 值越大, 延时时间越长
** 输出参数: 无
*****/
void DelayNS (uint32 uiDly)
{
    uint32 i;
    for (; uiDly > 0; uiDly--){
        for(i = 0; i < 50000; i++);
    }
}
/*****
** 函数名称: UARTInit
** 函数功能: 串口初始化, 设置为 8 位数据位, 1 位停止位,
无奇偶校验, 波特率为 115200
** 输入参数: uiDly 值越大, 延时时间越长
** 输出参数: 无
*****/
void UARTInit (void)
{
    uint16 uiFdiv;
    U0LCR = 0x83; /* 允许设置波特率 */
    uiFdiv = (Fpclk / 16) / UART_BPS; /* 设置波特率 */
    U0DLM = uiFdiv / 256;
    U0DLL = uiFdiv % 256;
    U0LCR |= 0x03; /* 锁定波特率 */
}
/*****
** 函数名称: UART0GetByte
```

```

** 函数功能: 从串口接收 1 字节数据, 使用查询方式接收
** 输入参数: 无
** 输出参数: 无
***/
uint8 UART0GetByte (void)
{
    uint8 uiRcvData;
    while ((U0LSR & 0x01) == 0);          /* 等待接收标志置位 */
    uiRcvData = U0RBR;                     /* 读取数据 */
    return (uiRcvData);
}

** 函数名称: UART0GetStr
** 函数功能: 串口接收字符串
** 输入参数: uiStr 指向接收数据数组的指针
**          uiNum 接收数据的个数
** 输出参数: 无
***/
void UART0GetStr(uint8 *uiStr, uint32 uiNum)
{
    for (; uiNum > 0; uiNum--){
        *uiStr++ = UART0GetByte ();
    }
}

** 函数名称: UART0SendByte
** 函数功能: 向串口发送子节数据, 并等待数据发送完成, 使用查询方式
** 输入参数: uiDat 要发送的数据
** 输出参数: 无
***/
void UART0SendByte (uint8 uiDat)
{
    U0THR = uiDat;                         /* 写入数据 */
    while ((U0LSR & 0x40) == 0);          /* 等待数据发送完毕 */
}

** 函数名称: UART0SendStr
** 函数功能: 向串口发送字符串
** 输入参数: uiStr 要发送的字符串指针
** 输出参数: 无
***/
void UART0SendStr(uint8 const *uiStr)
{
    while (1){

```

```

        if (*uiStr == '\0')break;                /* 遇到结束符，退出 */
        UART0SendByte (*uiStr++);
    }
}
int main (void)
{
    uint8    uiBuf[30];
    PINSEL0 = PINSEL0 & (~0x0F);
    PINSEL0 = PINSEL0 | 0x05;                    /* 设置 I/O 连接到 UART */
    UARTInit ();                                /* 串口初始化 */
    UART0GetStr(uiBuf, 20);                      /* 从串口接收字符串 */
    DelayNS(10);
    UART0SendStr (uiBuf);                        /* 向串口发送字符串 */
    DelayNS(10);
    while (1);
    return 0;
}

```

3.5.2 UART 实验 2

1. 实验目的

- 熟悉 TKStudio 集成开发环境的使用方法；
- 掌握 UART 各个控制寄存器的设置，UART 中断程序的设计，并能使用串口向 PC 机发送数据，能够理解发送 FIFO 和接收 FIFO 的功能。

2. 实验内容

使用 UART0 中断通信方式接收数据，用查询方式发送数据。

3. 实验要求

- 仔细阅读 LPC2103 芯片手册 UART0 部分内容；
- 仔细阅读 EasyARM2103 电路手册，了解硬件结构。

4. 实验思路

- 1) 编写中断服务程序。首先判断是否有中断挂起（读 U0IIR），然后判断中断标志，如果是接收数据中断则置接收新数据标志，连续接收多个字符；如果是字符超时中断，则判断字符是否接受完毕；
- 2) 设置 I/O 连接到 UART0（设置 PINSEL0 位 3：0=0101B）；
- 3) UART0 初始化。设置为 8 位数据位，1 位停止位，无奇偶校验（设置 U0LCR），波特率为 115200（设置 U0DLM 和 U0DLL）。使能 FIFO，设置触发点（设置 U0FCR），使能接收中断（设置 U0IER）；
- 4) 使能 IRQ 中断；
- 5) 配置 VIC 向量中断控制器。UART0 分配为 IRQ 中断（设置 VICIntSelect 位 6 为 0），UART0 分配为向量 IRQ 通道 0（设置 VICVectCntl0），分配中断服务程序地址（设置 VICVectAddr0 值为中断服务程序地址），UART0 中断使能（设置 VICIntEnable 位 6 为 1）；
- 6) 等待数据到来，判断是否有新数据；

7) 有则向 PC 机发送接收到的数据。

5. 实验步骤

- 1) 用一根串口延长线将 PC 串口与 UART0 相连接;
- 2) 其他步骤同第一个实验;
- 3) 在 main.c 中编写 UART0 发送与接收程序;
- 4) 启动调试;
- 5) 在 PC 机上向串口发送字符数据;
- 6) 观察 PC 机是否接收到 UART0 发送过来的数据。

6. 实验参考程序

程序清单 3.11 UART0 中断方式接收数据

```
#include "../config.h"

# define    UART_BPS    115200                                /* 串口通信波特率 */
volatile    uint8      uiGRcvNew;                             /* 串口接收新数据的标志 */
uint8      uiGRcvBuf[30] = {0};                               /* 串口接收数据缓冲区 */
uint32     uiGNum;                                             /* 串口接收数据的个数 */

void  UART0_IRQ (void)  __attribute__((naked));

/*****
** 函数名称: DelayNS
** 函数功能: 延时函数
** 输入参数: uiDly  值越大, 延时时间越长
** 输出参数: 无
*****/

void DelayNS (uint32 uiDly)
{
    uint32 i;
    for (; uiDly > 0; uiDly--){
        for(i = 0; i < 50000; i++);
    }
}

/*****
** 函数名称: UART0_IRQ
** 函数功能: 串口中断服务函数
** 输入参数: 无
** 输出参数: 无
*****/

void  UART0_IRQ (void)
{

```

```

ISR_ENTRY();

uiGNum = 0;

while ((U0IIR & 0x01) == 0){                                     /* 判断是否有中断挂起 */
    switch (U0IIR & 0x0E){                                       /* 判断中断标志 */

        case 0x04:                                             /* 接收数据中断 */
            uiGRcvNew = 1;                                       /* 置接收新数据标志 */
            for (uiGNum = 0; uiGNum < 8; uiGNum++){           /* 连续接收 8 个字节 */
                uiGRcvBuf[uiGNum] = U0RBR;
            }
            break;

        case 0x0C:                                             /* 字符超时中断 */
            uiGRcvNew = 1;
            while ((U0LSR & 0x01) == 0x01){                 /* 判断数据是否接收完毕 */
                uiGRcvBuf[uiGNum] = U0RBR;
                uiGNum++;
            }
            break;

        default:
            break;
    }
}

VICVectAddr = 0x00;
ISR_EXIT();
}

/*****
** 函数名称: UARTInit
** 函数功能: 串口初始化, 设置为 8 位数据位, 1 位停止位,
              无奇偶校验, 波特率为 115200
** 输入参数: uiDly 值越大, 延时时间越长
** 输出参数: 无
*****/

```

```

void UARTInit (void)
{
    uint16 uiFdiv;
    U0LCR  = 0x83;                                /* 允许设置波特率          */
    uiFdiv = (Fpclk / 16) / UART_BPS;              /* 设置波特率              */
    U0DLM  = uiFdiv / 256;
    U0DLL  = uiFdiv % 256;
    U0LCR  = 0x03;                                /* 锁定波特率              */
}

/*****

** 函数名称: UART0SendByte
** 函数功能: 向串口发送字节数据, 并等待数据发送完成, 使用查询方式
** 输入参数: uiDat    要发送的数据
** 输入参数: 无

*****/

void UART0SendByte (uint8 uiDat)
{
    U0THR = uiDat;                                /* 写入数据                */
    while ((U0LSR & 0x20) == 0);                  /* 等待数据发送完毕        */
}

/*****

** 函数名称: UART0SendStr
** 函数功能: 向串口发送字符串
** 输入参数: uiStr     要发送的字符串指针
**           uiNum     要发送的数据个数
** 输出参数: 无

*****/

void UART0SendStr(uint8 const *uiStr, uint32 uiNum)
{
    uint32 i;
    for (i = 0; i < uiNum; i++){
        UART0SendByte (*uiStr++);                /* 发送指定个字节数据      */
    }
}

/*****

** 函数名称: main
** 函数功能: 跳线 JP6 短接, 打开串口调试软件, 串口 0 中断方式通信

```

```

** 输入参数: 无
** 输出参数: 无
*****/

int main (void)
{
    PINSEL0 = PINSEL0 & (~0x0F);
    PINSEL0 = PINSEL0 | 0x05;          /* 设置 I/O 连接到 UART */
    uiGRcvNew = 0;
    UARTInit ();                      /* 串口初始化 */
    U0FCR = 0x81;                     /* 使能 FIFO, 设置 8 个字节触发点 */
    U0IER = 0x01;                     /* 使能接收中断 */

    IRQEnable ();

    VICIntSelect = 0x00000000;        /* 设置所有中断为向量中断 */
    VICVectCntl0 = 0x20 | 0x06;       /* 设置串口中断为最高优先级 */
    VICVectAddr0 = (uint32)UART0_IRQ; /* 设置向量地址 */
    VICIntEnable = 1 << 0x06;        /* 使能串口中断 */

    while (1){
        if (uiGRcvNew == 1){          /* 判断是否有新数据 */
            uiGRcvNew = 0;            /* 清除标志 */
            UART0SendStr (uiGRcvBuf, uiGNum); /* 向串口发送数据 */
        }
    }

    return 0;
}

```

3.6 I²C 接口实验

3.6.1 实验目的

- 熟悉 TKStudio 集成开发环境的使用方法;
- 掌握 I²C 总线协议内容, 学会使用 I2C 总线。

3.6.2 实验内容

使用 I²C 总线将数据从 EEPROM 中写入和读出。

3.6.3 实验要求

- 仔细阅读 LPC2103 芯片手册 UART0 部分内容;
- 仔细阅读 EasyARM2103 电路手册, 了解硬件结构。

3.6.4 实验思路

- 1) 编写 I²C 总线协议通信方面的程序，包括初始化，发送接收数据等；
- 2) 参照实验 6 写 UART 通信程序；
- 3) 通过 I²C 向 EEPROM 写一个字节的数据；
- 4) 从 EEPROM 读去一个字节的数据；
- 5) 比较两者的结果，看是否相当，如果相等通过 UART 发送数据，同时循环点亮 LED 灯。

3.6.5 实验步骤

- 1) 参照实验 6 接 UART 引脚跳线；
- 2) 短接条线 JP4；
- 3) 打开 PC 机串口调试软件，设置波特率为 115200；
- 4) 下载调试程序，查看结果。

3.6.6 实验参考程序

程序清单 3.12 I²C 实验

```
/******  
** 函数名称: main  
** 函数功能: 向 EEPROM 写入数据，然后读出判断是否正确写入  
** 输入参数: 无  
** 输出参数: 无  
*****/  
  
int main(void)  
{  
    BYTE t='a';  
    status ret=ERROR;  
  
    TargetInit();  
    PortInit();  
    uart_init();  
    I2C_init();  
  
    ret=send_byte(CAT1025,0,t);  
    if(ret==SUCESS)  
    {  
        uHALr_printf("sucess\r\n");  
        LED();  
    }  
  
    t='A';  
    rcv_byte(CAT1025,0,&t);  
    uHALr_printf("Get:");
```

```

uart_SendByte(t);
uHALr_printf("\r\n");

while(1)
{
    if(t=='a')
        LED();
}
return 0;
}

```

3.7 SPI 接口实验

3.7.1 实验目的

- 熟悉 TKStudio 集成开发环境的使用方法；
- 掌握 SPI 总线协议内容，学会使用 SPI 总线。

3.7.2 实验要求

阅读 LPC2103 手册中有关 SPI 方面的内容，并掌握有关 SPI 的使用。

3.7.3 实验思路

使用 LPC2103 自带的两个 SPI 接口(其中一个为 SSP，其内容与 SPI 差不多)，一个做主机，一个做从机，进行数据的发送与接收，并将接收到的数据通过串口发送到 PC 机上。

- 1) 初始化串口。包括设置波特率，数据位，停止位，奇偶校验位以及 I/O 引脚选择；
- 2) 初始化主 SPI，其中需要拿一个 IO 引脚作为选择从机信号；
- 3) 初始化从 SPI；
- 4) 编写双方数据发送和接收功能程序；
- 5) 连线进行通信。

3.7.4 实验步骤

- 1) 参照 UART 实验，对 UART I/O 进行跳线；
- 2) 参照 LPC2103 数据手册进行连线。其中主 SPI 的 MOSI 接从 MOSI，主 SPI 的 MISO 接从机的 MISO。两根 SCK 连在一起。另外选用 GPIO0 的第 13 脚作为从机选择信号，请与从机的 SSEL 连接。

3.7.5 实验参考程序

最后需要注意的是如果程序没有输出结果，请确保引脚已经正确连接。

程序清单 3.13 SPI 接口实验

```

/*****
** 函数名称: main
** 函数功能: SPI 做主机、SSP 做从机进行数据通信，并将 SPI 发送的数据通过串
              口发送到 PC 机显示
** 输入参数: 无
** 输出参数: 无

```

```

/*****
int main(void)
{
    uint8 i;
    uint8 j;
    uint8 uiRcvData;

    PINSEL0 = PINSEL0 & ~(0x03 << 26));          /* 设置 GPIO 引脚连接          */

    IO0DIR  = SLAVE_CS;                          /* 设置 SSP 片选为输出          */
    IOSET   = SLAVE_CS;

    UARTInit();
    MSPInit();                                  /* 初始化 SPI 接口              */
    SSPInit();                                  /* 初始化 SSP 接口              */

    for(;;){
        for ( i = 0; i < 8; i++){                /* SPI 发送数据                  */
            MSPISendData(uiBuf[i]);
            DelayNS(10);
            uiRcvData = SSPRcvByte();             /* SSP 接收数据                  */
            DelayNS(10);

            for ( j = 0; j < 8; j++){
                PCDISPChar (j, uiRcvData);        /* 将 SSP 接收到的数据发送到 PC 机 */
            }
            DelayNS(50);
        }
    }
    return 0;
}

```

3.8 SSP 实验

本章实验内容与实验 8 的实验内容类似，这里将不再作过多的说明。只是 SSP 实验使用 SSP 作为主机，SPI 作为从机。连线：主 SSP 的 MOSI 接从 MOSI，主 SSP 的 MISO 接从机的 MISO。两根 SCK 连在一起。另外选用 GPIO0 的第 13 脚作为从机选择信号，请与从机的 SSEL 连接。

3.9 RTC 实验

3.9.1 RTC 实验 1

1. 实验目的

- 熟悉 TKStudio 集成开发环境的使用方法；
- 掌握 RTC 在不同时钟系统下的分频设置，掌握 RTC 日期和时间值的设置及读取。

2. 实验内容

读取 LPC2103 时钟值向 PC 串口发送，在 PC 机显示。

3. 实验要求

- 仔细阅读 LPC2103 芯片手册 UART0 部分内容；
- 仔细阅读 LPC2103 芯片手册 RTC 部分内容；
- 仔细阅读 EasyARM2103 电路手册，了解硬件结构。

4. 实验思路

- 1) UART0 初始化。设置 UART0 波特率，数据位，停止位，奇偶校验；
- 2) RTC 初始化。设置基准时钟分频器（设置 PREINT），禁止时间计数器（设置 CCR），设置年（设置 YEAR）、月（设置 MONTH）、日（设置 DOM）、小时（设置 HOUR）、分（设置 MIN）、秒值（设置 SEC）。设置秒值的增量产生 1 次中断（设置 CIIR），启动 RTC（设置 CCR）；
- 3) 等待 RTC 增量中断发生，然后清除中断标志；
- 4) 读取完整的时钟寄存器，然后分别分离出年月日小时分秒向 PC 串口发送；
- 5) 在 PC 端启动串口接收软件，观察接收到的数据。

5. 实验步骤

- 1) 用一根串口延长线将 PC 串口与 UART0 相连接；
- 2) 其他步骤同第一个实验；
- 3) 在 main.c 中编写程序，引用 UART0 发送字符函数；
- 4) 启动调试；
- 5) 观察 PC 机是否接收到 UART0 发送过来的数据。

6. 实验参考程序

程序清单 3.14 RTC 读取时间

```
#include "config.h"

typedef struct UartMode{                                /* 定义串口模式设置的数据结构*/
    uint8 datab;                                       /* 字长度 5/6/7/8 */
    uint8 stopb;                                       /* 停止位 1/2 */
    uint8 parity;                                       /* 奇偶校验 */
}UARTMODE;

#define UART_BPS 115200                                /* 串口通信波特率 */

/*****
** 函数名称: UARTInit
** 函数功能: 串口初始化，设置为 8 位数据位，1 位停止位，无奇偶校验，波特率为 115200
** 输入参数: uiDly 值越大，延时时间越长
** 输出参数: 无
*****/

void UARTInit (void)
{
```

```

uint16 uiFdiv;

PINSEL0 = (PINSEL0 & 0xFFFFFFF0) | 0x00000005;          /* 串口引脚设置          */

U0LCR = 0x83;          /* 允许设置波特率          */
uiFdiv = (Fpclk / 16) / UART_BPS;          /* 设置波特率          */
U0DLM = uiFdiv / 256;
U0DLL = uiFdiv % 256;
U0LCR = 0x03;          /* 锁定波特率          */
U0FCR = 0x01;          /* FIFO 使能          */
}

/*****
** 函数名称: UART0SendByte
** 函数功能: 向串口发送子节数据, 并等待数据发送完成, 使用查询方式
** 输入参数: uiDat 要发送的数据
** 输出参数: 无
*****/

void UART0SendByte (uint8 uiDat)
{
    U0THR = uiDat;          /* 写入数据          */
    while ((U0LSR & 0x40) == 0);          /* 等待数据发送完毕          */
}

/*****
** 函数名称: PC_Dispatch
** 函数功能: 向 PC 机发送显示字符
** 输入参数: x 显示字符的横坐标
**           y 显示字符的纵坐标
**           chr 显示的字符, 不能为 ff
**           color 显示的状态, 包括前景色、背景色、闪烁位。
**           与 DOS 字符显示一样: 0~3,前景色, 4~6, 背景色, 7, 闪烁位
** 输出参数: 无
*****/

void PCDispChar (uint8 uiX, uint8 uiChr)
{
    UART0SendByte(0xFF);          /* 起始字符          */
    UART0SendByte(0x81);
    UART0SendByte(uiX);
    UART0SendByte(uiChr);
    UART0SendByte(0x00);
}

uint8 const uiSHOWTABLE[10] = {0x3f,0x06,0x5b,0x4f,0x66,0x6d,0x7d,0x07,0x7f,0x6f};

/*****
** 函数名称: SendTimeRtc

```

```

** 函数功能: 将 RTC 时间值发送到串口显示
** 输入参数: uiDat 要发送的数据
** 输出参数: 无
*****/
void SendTimeRtc (void)
{
    uint32 uiDats;
    uint32 uiTimes;
    uint32 bak;

    uiTimes = CTIME0;          /* 读取完整的时钟寄存器 */
    uiDats = CTIME1;
    bak = (uiDats >> 16) & 0xfff;    /* 获取年 */
    PCDispChar (0, uiSHOWTABLE [bak / 1000]);
    bak = bak % 1000;
    PCDispChar (1, uiSHOWTABLE [bak / 100]);
    bak = bak % 100;
    PCDispChar (2, uiSHOWTABLE [bak / 10]);
    PCDispChar (3, uiSHOWTABLE [bak % 10]);

    bak = (uiDats >> 8) & 0x0f;    /* 获取月 */
    PCDispChar (4, uiSHOWTABLE [bak / 10]);
    PCDispChar (5, uiSHOWTABLE [bak % 10]);

    bak = uiDats & 0x1f;          /* 获取日 */
    PCDispChar (6, uiSHOWTABLE [bak / 10]);
    PCDispChar (7, uiSHOWTABLE [bak % 10]);

    bak = (uiTimes >> 24) & 0x07;    /* 获取星期 */
    PCDispChar (8, uiSHOWTABLE [bak]);

    bak = (uiTimes >> 16) & 0x1f;    /* 获取小时 */
    PCDispChar (9, uiSHOWTABLE [bak / 10]);
    PCDispChar (10, uiSHOWTABLE [bak % 10]);

    bak = (uiTimes >> 8) & 0x3f;    /* 获取分钟 */
    PCDispChar (11, uiSHOWTABLE [bak / 10]);
    PCDispChar (12, uiSHOWTABLE [bak % 10]);

    bak = uiTimes & 0x3f;          /* 获取 秒钟 */
    PCDispChar (13, uiSHOWTABLE [bak / 10]);
    PCDispChar (14, uiSHOWTABLE [bak % 10]);
}
*****/

```

```

** 函数名称: RTCInit
** 函数功能: 初始化实时时钟
** 输入参数: 无
** 输出参数: 无
*****/
void RTCInit (void)
{
    PREINT  = Fpclk / 32768 - 1;          /* 设置基准时钟分频器          */
    PREFRAC = Fpclk - (Fpclk / 32768) * 32768;
    CCR = 0x00;                          /* 禁止时间计数器          */
    YEAR   = 2008;
    MONTH  = 04;
    DOM     = 07;
    DOW     = 4;
    HOUR    = 15;
    MIN     = 52;
    SEC     = 59;
    CHIR    = 0x01;                      /* 设置秒值的增量产生 1 次中断 */
    CCR     = 0x01;                      /* 启动 RTC                  */
}
/*****
** 函数名称: main
** 函数功能: 主函数，通过串口发送到 PC 机显示当前 RTC 时间
** 输入参数: 无
** 输出参数: 无
*****/
int main (void)
{
    UARTMODE uart0_set;
    uart0_set.datab  = 8;
    uart0_set.stopb  = 1;
    uart0_set.parity = 0;

    UARTInit();          /* 串口初始化          */
    RTCInit();           /* RTC 初始化          */
    while (1){
        while (0 == (ILR & 0x01)); /* 等待 RTC 增量中断 */
        ILR = 0x01;          /* 清除中断标志          */
        SendTimeRtc();        /* 发送到串口显示          */
    }
    return 0;
}

```

3.9.2 RTC 实验 2

1. 实验目的

- 熟悉 TKStudio 集成开发环境的使用方法；
- 掌握 RTC 使用方法。

2. 实验内容

RTC 中断唤醒掉电 CPU，唤醒 CPU 后，LED1 闪烁。

3. 实验要求

- 仔细阅读 LPC2103 芯片手册 RTC 部分内容；
- 仔细阅读 EasyARM2103 电路手册，了解硬件结构。

4. 实验思路

- 1) 选择 P0.17 引脚为 GPIO 功能（设置 PINSEL1 位 3: 2=00B），LED1-P0.17；
- 2) RTC 初始化。设置使用独立的振荡器（设置 CCR），设置秒值的增量产生一次中断（设置 CIIR），清除增量中断标志（设置 ILR），启动 RTC（设置 CCR），允许 RTC 唤醒 CPU（设置 INTWAKE）；
- 3) 使 CPU 进入掉电状态（设置 PCON），清除增量中断标志；
- 4) 等待 CPU 唤醒，取 P0.17 状态值，然后取反，达到闪烁目的。

5. 实验步骤

- 1) 短接跳线 JP4；
- 2) 其他步骤同第一个实验；
- 3) 在 main.c 中编写程序；
- 4) 启动调试。

6. 实验参考程序

程序清单 3.15 RTC 中断唤醒掉电 CPU

```
#include "config.h"

#define LED (1 << 17) /* P0.17 控制 LED */
/*****

** 函数名称: RTCInit
** 函数功能: 初始化实时时钟
** 输入参数: 无
** 输出参数: 无
*****/

void RTCInit (void)
{
    CCR = 0x12; /* 使用独立振荡器 */
    CIIR = 0x01; /* 设置秒值的增量产生一次中断 */
    ILR = 0x03; /* 清除增量中断标志 */
    CCR = 0x11; /* 启动 RTC */
    INTWAKE = 1 << 15; /* 允许 RTC 唤醒 CPU */
}
```

```

}
/*****
** 函数名称: main
** 函数功能: RTC 中断唤醒掉电 CPU 实验
** 输入参数: 无
** 输出参数: 无
*****/

int main (void)
{
    PINSEL1 = PINSEL1 & ~(0x03 << 2);
    IO0DIR = LED;
    IO0SET = LED;

    RTCInit ();                                /* RTC 初始化 */
    while(1) {
        PCON = 0x02;                          /* CPU 进入掉电模式 */
        ILR = 0x01;                          /* 清除增量中断标志 */
        if((IO0PIN & LED) == 0) {             /* 唤醒 CPU 后, 取反 LED */
            IO0SET = LED;
        }
        else IO0CLR = LED;
    }
    return 0;
}

```

3.10 模数转换器实验

3.10.1 实验目的

- 熟悉 TKStudio 集成开发环境的使用方法;
- 掌握模/数 (A/D) 转换器的应用设置, 进行电压信号的测量。

3.10.2 实验内容

利用 ADC 对通道 0 的电压进行采样, 并把值通过 UART0 发送到 PC 机显示。

3.10.3 实验要求

- 仔细阅读 LPC2103 芯片手册 A/D 部分内容;
- 仔细阅读 EasyARM2103 电路手册, 了解硬件结构。

3.10.4 实验思路

- 1) 设置 P0.22 引脚为 AD 功能 (设置 PINSEL1);
- 2) 进行 ADC 模块设置。选择通道 0, 转换时钟, 软件控制转换操作, 选择正常工作模式, 设置直接启动模式;
- 3) 启动 AD 转换。将转换结果通过 UART0 发送给 PC 端显示。

3.10.5 实验步骤

- 1) 用一根串口延长线将 PC 串口与 UART0 相连接;
- 2) 其他步骤同第一个实验;
- 3) 在 main.c 中编写程序, 引用 UART0 发送字符函数;
- 4) 启动调试;
- 5) 观察 PC 机是否接收到 UART0 发送过来的数据。

3.10.6 实验参考程序

程序清单 3.16 AD 转换

```
#include "config.h"
#include <stdio.h>
#include "UART_Display\UART.h"

/*****
** 函数名称: DelayNS
** 函数功能: 延时函数
** 输入参数: uiDly 延时值
** 输出参数: 无
*****/

void DelayNS (uint32 uiDly)
{
    uint32 i;
    for(; uiDly > 0; uiDly--){
        for(i=0; i<50000; i++);
    }
}

/*****
** 函数名称: UiToStr
** 函数功能: 将无符号 32 数据转换为字符串
** 输入参数: uiValue--需要转换的值
** 输出参数: pStr--转换后的结果
*****/

void UiToStr(uint32 uiValue,char *pStr)
{
    uint8 i,j,uiSize = 0;
    char *pTStr = pStr;
    while(uiValue != 0)
    {
        *pTStr++ = uiValue%10 + '0';
        uiValue = uiValue/10;
        uiSize++;
    }
    *pTStr = '\0';
    j = uiSize >> 1;
```

```

    for (i = 0; i < j ;i++)
    {
        uiSize --;
        char cTmp = pStr[i];
        pStr[i] = pStr[uiSize];
        pStr[uiSize] = cTmp;
    }
}

/*****

** 函数名称: Main
** 函数功能: 利用 ADC 对通道 0 的电压进行采样,
并把值发送到 PC 机显示出来
** 输入参数: 无
** 输出参数: 无

*****/

int main(void)
{
    uint32 uiVal;
    char    cStr[20];
    PINSEL1 = PINSEL1 & ~(0x03 << 12);          /* 将 P0.22 设置为 AD 功能      */
    PINSEL1 = PINSEL1 | (0x03 << 12);
    UARTInit();                                  /* 串口初始化                  */
    AD0CR = (1 << 0) |                            /* 选择通道 0                  */
            (((Fpclk / 1000000) - 1) << 8) |      /* 转换时钟为 1MHz            */
            (0 << 16) |                            /* BURST = 0,软件控制转换操作 */
            (0 << 17) |                            /* CLKS = 0,使用 11clock 转    */
            (1 << 21) |                            /* PDN = 1, 正常工作模式      */
            (0 << 22) |                            /* TEST = 0, 正常工作模式     */
            (1 << 24) |                            /* 设置直接启动模式           */
            (0 << 27);                             /* 设置模式, 直接启动模式下无效 */

    while(1) {
        AD0CR |= (1 << 24);
        while ((AD0STAT & 0x01) == 0);           /* 读取 AD0STAT 的通道 0 的 Done */
        AD0CR |= (1 << 24);
        while ((AD0STAT & 0x01) == 0);           /* 读取 AD0STAT 的通道 0 的 Done */
        DelayNS(10);
        uiVal = AD0DR0;
        uiVal = (uiVal >> 6) & 0x3FF;
        uiVal = (uiVal * 3300)/1024;              /* ADC 的参考电压为 3300mV      */

        UiToStr(uiVal,cStr);                      /* 格式化显示数据              */
        ISendStr (0,0,0x30,cStr);                /* 将数据发送到 PC 机显示      */
        DelayNS(50);
    }
}

```

```
    DelayNS(50);  
}  
return (0);  
}
```

3.11 WDT 实验

3.11.1 实验目的

- 熟悉 TKStudio 集成开发环境的使用方法;
- 熟悉 LPC2103 ARM7 微控制器的 WDT 功能及使用方法。

3.11.2 实验内容

演示看门狗溢出复位, 按键 KEY1 被按下后使得看门狗溢出复位, LED 闪烁一次。

3.11.3 实验要求

- 仔细阅读 LPC2103 芯片手册 WDT 部分内容;
- 仔细阅读 EasyARM2103 电路手册, 了解硬件结构。

3.11.4 实验思路

- 1) 设置 LED1 显示配置 (参看前面实验设置方法);
- 2) 看门狗初始化。设置 WDT 定时值 (设置 WDTC), 设置 WDT 工作模式 (设置 WDMOD), 启动 WDT;
- 3) 编写喂狗程序;
- 4) 检测按键是否按下, 按下就停止喂狗, 让硬件复位; 没有按下, 等待。

3.11.5 实验步骤

- 1) 短接跳线 JP3, JP4;
- 2) 其他步骤同第一个实验;
- 3) 在 main.c 中编写程序;
- 4) 编译后, 将目标文件 (main.bin) 烧写到 FLASH 中, 然后复位;
- 5) 等 LED1 熄灭后, 按下按键 KEY1, 观察 LED 是否点亮。

3.11.6 烧写目标文件到 FLASH 步骤

(1) 单击任务栏下的  图标, 将弹一个 H-Flasher 窗口, 如图 3.15 所示。如找不到  图标, 说明 H-Flasher 软件没有启动, 启动 H-Flasher 软件。

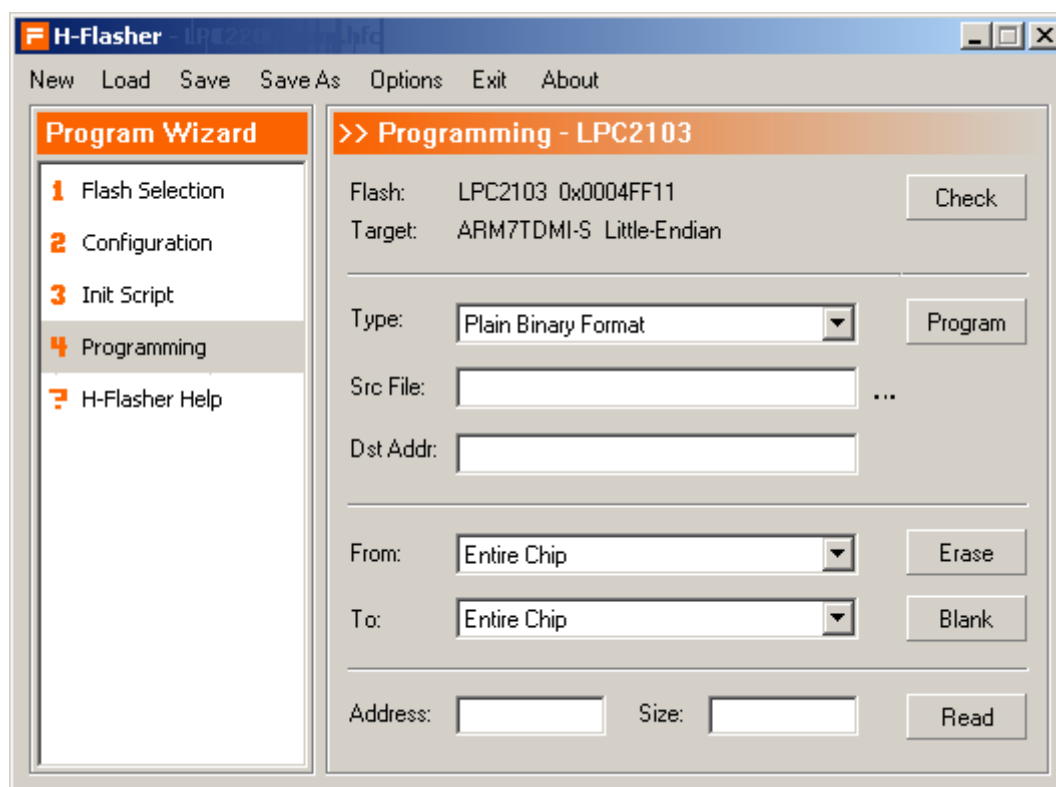


图 3.15 H-Flasher

(2) 选择【4 programming】项，右边的【Type】项选择“Plain Binary Format”，【Src File】项选择为工程目录\DebugRef 目录下的后缀名为.bin 的文件；【Dst Addr】填入要烧写的内存地址，本例子烧写地址为 0，然后点击【Program】按钮即可。如图 3.16 所示，此时将弹出一个烧写进度框。如图 3.17 所示。

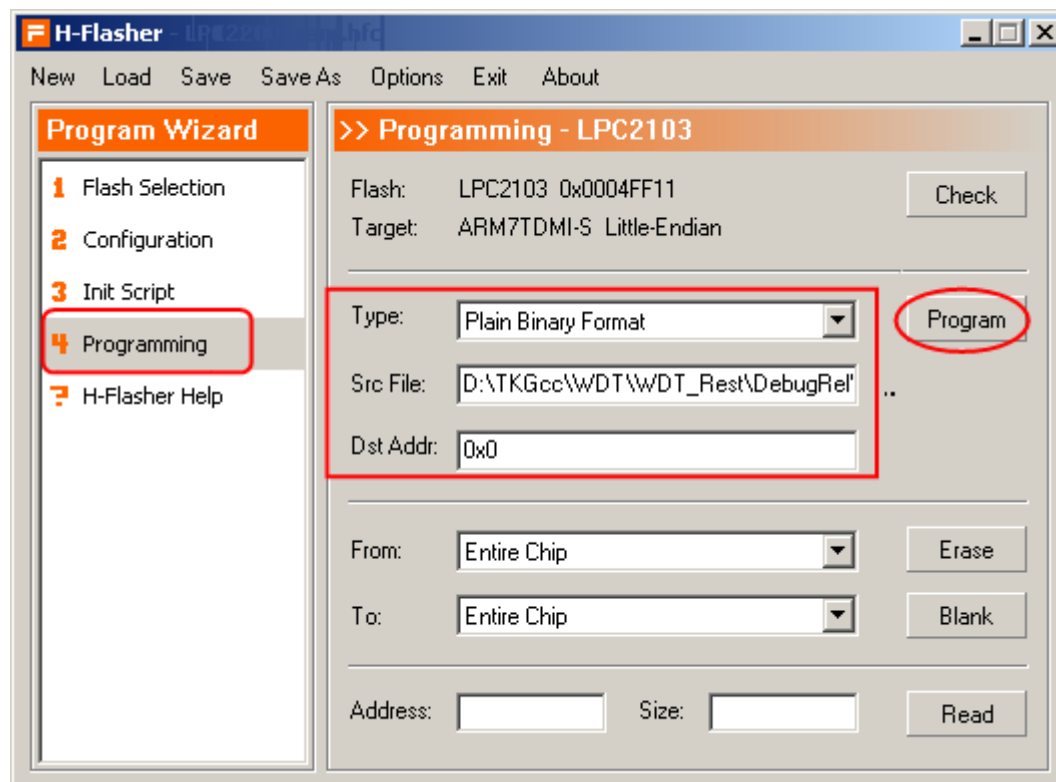


图 3.16 文件选择与烧写地址

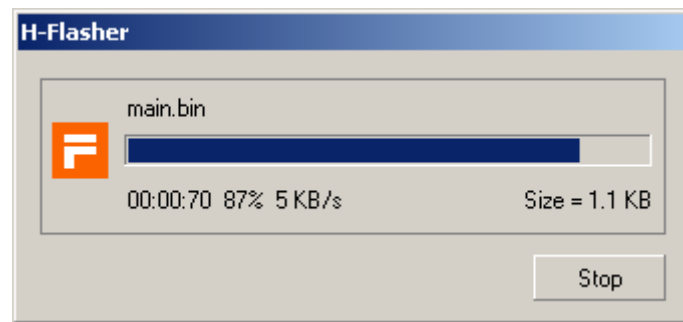


图 3.17 烧写进度

3.11.7 实验参考程序

程序清单 3.17 WDT 复位

```
#include    "../config.h"

#define    LED1      1 << 17                /* P0.17 控制 LED1          */
#define    KEY1      1 << 16                /* P0.16 产生看门狗溢出按键 */

/*****

** 函数名称: DelayNS
** 函数功能: 延时函数
** 输入参数: uiDly    延时值
** 输出参数: 无

*****/

void DelayNS(uint32 uiDly)
{
    uint32 i;
    for(; uiDly > 0; uiDly--)
    {
        for(i=0; i<50000; i++);
    }
}

/*****

** 函数名称: WDTInit
** 函数功能: 看门狗初始化
** 输入参数: uiTime    设置看门狗定时器参数
** 输出参数: 无

*****/

void WDTInit(uint32 uiTime)
{
    WDTC      = uiTime;                /* 设置看门狗定时器参数      */
    WDMOD     = 0x03;                  /* 设置看门狗模式: 溢出后复位 */
    WDFEED    = 0xAA;                  /* 第一次喂狗启动 WDT        */
    WDFEED    = 0x55;                  /* 喂狗序列                    */
}
```

```

/*****
** 函数名称: FeedDog
** 函数功能: 喂狗函数
** 输入参数: 无
** 输出参数: 无
*****/

void FeedDog(void)
{
    IRQDisable ();                /* 禁止中断 */
    WDFEED = 0xAA;                /* 喂狗序列 */
    WDFEED = 0x55;
    IRQEnable ();                 /* 喂狗序列 */
}

/*****
** 函数名称: main
** 函数功能: 演示看门狗溢出复位, 按键 KEY1 被按下后使得
看门狗溢出复位, LED 闪烁一次
** 输入参数: 无
** 输出参数: 无
*****/

int main (void)
{
    uint32 i;
    PINSEL1  =  0x00000000;        /* 管脚连接 GPIO */
    IO0DIR   =  LED1;              /* LED1 控制口输出 */
    IO0CLR   =  LED1;              /* 点亮 LED1 */
    DelayNS(100);
    IO0SET   =  LED1;              /* 熄灭 LED1 */
    WDTInit(0x200000);             /* 看门狗初始化 */
    while(1){
        while((IO0PIN & KEY1) == 0); /* 如果按键 1 按下, 就停止喂狗 */
        for(i = 0; i < 0xFF; i++);    /* LED1 闪烁周期 */
        FeedDog();                    /* 喂狗 */
    }
    return 0;
}

```

3.12 PLL 设置实验

3.12.1 实验目的

- 熟悉 TKStudio 集成开发环境的使用方法;
- 熟悉 LPC2103 ARM7 微控制器的 PLL 设置。

3.12.2 实验内容

设置 PLL, 在中断中打开 PLL, 让 LED1 闪烁一次。

3.12.3 实验要求

- 仔细阅读 LPC2103 芯片手册 PLL（锁相环）部分内容；
- 仔细阅读 EasyARM2103 电路手册，了解硬件结构。

3.12.4 实验思路

- 1) 编写中断服务程序。实现 PLL 使能和连接（设置 PLLCON 值为 3），发送 PLL 馈送序列（设置 PLLFEED）。点亮 LED1，延时，熄灭 LED1，然后禁止 PLL 中断；
- 2) 设置 LED1 显示电路（设置 PINSEL1）；
- 3) 使能 PLL（设置 PLLCON），设置 M、P 的值（设置 PLLCFG），发送 PLL 馈送序列（设置 PLLFEED）；
- 4) 使能 IRQ 中断；
- 5) 配置 VIC 向量中断控制器，设置 PLL 中断为 IRQ 中断（设置 VICIntSelect），中断优先级为 0（设置 VICVectCntl0），设置中断服务函数地址，然后使能 PLL 中断（设置 VICIntEnable）；
- 6) 等待 PLL 中断发生。

3.12.5 实验步骤

- 1) 短接跳线 JP4；
- 2) 其他步骤同第一个实验；
- 3) 在 main.c 中编写程序；
- 4) 启动调试。

3.12.6 实验参考程序

程序清单 3.18 PLL 设置

```
#include "config.h"

#define LED (1 << 17) /* P0.17 控 LED 闪烁 */
void PLLINT(void) __attribute__((naked));
/*****
** 函数名称: DelayNS
** 函数功能: 延时函数
** 输入参数: uiDly 延时值
** 输出参数: 无
*****/
void DelayNS(uint32 uiDly)
{
    uint32 i;
    for(; uiDly > 0; uiDly--)
    {
        for(i=0; i<50000; i++);
    }
}
```

```

/*****
** 函数名称: PLL_INT
** 函数功能: 中断服务函数
** 输入参数: 无
** 输出参数: 无
*****/

void PLLINT(void)
{
    ISR_ENTRY();
    PINSEL1 = 0x00000000; /* 设置所有管脚连接 GPIO */
    IO0DIR = LED; /* 设置 LED 控制口输出 */

    PLLCON = 3; /* PLL 使能和连接 */
    PLLFEED = 0xAA; /* 发送 PLL 馈送序列 */
    PLLFEED = 0x55;

    IO0CLR = LED; /* LED 闪烁 1 次 */
    DelayNS(200);
    IO0SET = LED;

    VICIntEnClr = 1 << 12; /* 禁能 PLL_INT 中断 */
    VICVectAddr = 0x00; /* 向量中断处理结束 */
    ISR_EXIT();
}

/*****
** 函数名称: main
** 函数功能: 在中断中打开 PLL, LED 闪烁一次
** 输入参数: 无
** 输出参数: 无
*****/

int main(void)
{
    PINSEL0 = 0x00000000; /* 设置管脚连接 GPIO */
    IO0SET = LED; /* LED 控制口输出 */
    PLLCON = 1; /* 使能 PLL */
    PLLCFG = 0x25; /* 设置 M 为 6, P 为 2 */
    PLLFEED = 0xAA; /* 发送 PLL 馈送序列 */
    PLLFEED = 0x55;

    IRQEnable(); /* 使能 IRQ 中断 */
    VICIntSelect = 0x00000000; /* 设置所有的通道为 IRQ 中断 */
    VICVectCntl0 = 0x20 | 12; /* PLL_INT 分配到最高优先级 */
    VICVectAddr0 = (uint32)PLLINT; /* 设置 PLL_INT 向量地址 */
    VICIntEnable = 1 << 12; /* 使能 PLL_INT 中断 */
}

```

```
while (1);  
return 0;  
}
```