**《计算系统设计与实现》**

实验指导书

**哈尔滨工业大学计算学部**

**2025 年 3 月**

**目 录**

[实验基础 1](#bookmark2)

[1.1 实验平台 1](#bookmark4)

[1.2 CCSv10 简介 8](#bookmark6)

[1.3 TivaWare 简介 12](#bookmark8)

[1.4 新建一个 CCS 工程 16](#bookmark10)

[1.5 TivaWare 的使用 25](#bookmark12)

[实验项目 32](#bookmark14)

[实验 1 环境建立与 GPIO 实验 32](#bookmark16)

[实验 2 I2C 数据传送设计 35](#bookmark18)

[实验 3 RTOS 的移植与应用 39](#bookmark20)

[实验 4 PWM 的生成与分析 47](#bookmark22)

[实验 5 三色呼吸灯实验 53](#bookmark24)

[实验 6 综合设计实验 55](#bookmark26)

**实验基础**

1.1 实验平台

**1. Tiva LaunchPad**简介

Tiva LaunchPad 是基于德州仪器（Texas Instruments，简称 TI）Tiva™ C 系列的 TM4C123G 微处理器的开发板。它是一款低成本且具有强大扩展性的 ARM 微处理器开发板。

TM4C**123**GH6PM是TI公司推出的一款32位基于ARM Cortex-M4的处理器， 主频80MHz， 256kB Flash， 32kB SRAM，具有USB Host， Device和OTG的能力。

Tiva C LaunchPad是基于TM4C123GH6PM控制器的实验板卡，自带仿真器， 连接上USB即 可进行Corte-M4的学习。

**主要特点**

1. **板载调试器 ICDI**

o 开发板自带板载调试接口电路（In-Circuit Debug Interface，简称 ICDI）， 无 需外接调试器即可进行开发和调试。

o 开发板通过白色分隔线分为上下两部分：上半部分为 ICDI，下半部分为 TM4C123GH6PMI 处理器组成的最小系统。

2. **性能强大的处理器**

o 核心处理器为 TM4C123GH6PMI，具有以下特性：

 CPU： ARM Cortex-M4。

 Flash： 256 KB。

 SRAM： 32 KB。

 最大工作频率： 80 MHz。

 提供丰富的外设接口， 包括 PWM、 QEI、 GPIO、 USB OTG、 SPI、 I2C、 UART、 ADC 等。

3. **低成本且易于入手的板上资源**

o 提供两个通用按键、 一个复位按键、 一个 RGB 三色 LED 和一个用于通信的 USB 接口。

o 这些资源配合 ICDI 和计算机软件， 用户可以以最低的硬件成本实现对 M4 处理器的基本编程练习， 快速入门 ARM M4 处理器的开发。

4. **易于扩展的 BoosterPack 兼容接口**

o 提供统一的扩展接口，支持 BoosterPack 模块。

o BoosterPack 是基于这种扩展接口开发的各种功能小模块，用户可以根据需要 选择不同的模块进行扩展（例如传感器模块、通信模块等）。

o TI 官方网站提供了多种设计好的 BoosterPack 模块供选购。

5. **丰富的在线资源**

o 用户可以访问 TI 官方网站 （[http://www.ti.com/launchpad）获取所需的软](http://www.ti.com/launchpad%EF%BC%89%E8%8E%B7%E5%8F%96%E6%89%80%E9%9C%80%E7%9A%84%E8%BD%AF%E4%BB%B6%E3%80%81%E5%8F%82%E8%80%83%E6%89%8B%E5%86%8C%E5%8F%8A%E4%BB%A3%E7%A0%81%E5%BA%93%E3%80%82) [件、参考手册及代码库。](http://www.ti.com/launchpad%EF%BC%89%E8%8E%B7%E5%8F%96%E6%89%80%E9%9C%80%E7%9A%84%E8%BD%AF%E4%BB%B6%E3%80%81%E5%8F%82%E8%80%83%E6%89%8B%E5%86%8C%E5%8F%8A%E4%BB%A3%E7%A0%81%E5%BA%93%E3%80%82)

o 提供了详细的开发文档和技术支持，方便开发者学习和开发。

整个板卡如图1-1所示， 板子上带2个用户按键和1个三色的LED灯； 对外引出的IO口符合 BoosterPack 40Pin标准定义（详见TI的文档），也就是说可以兼容MSP430 LaunchPad板卡。

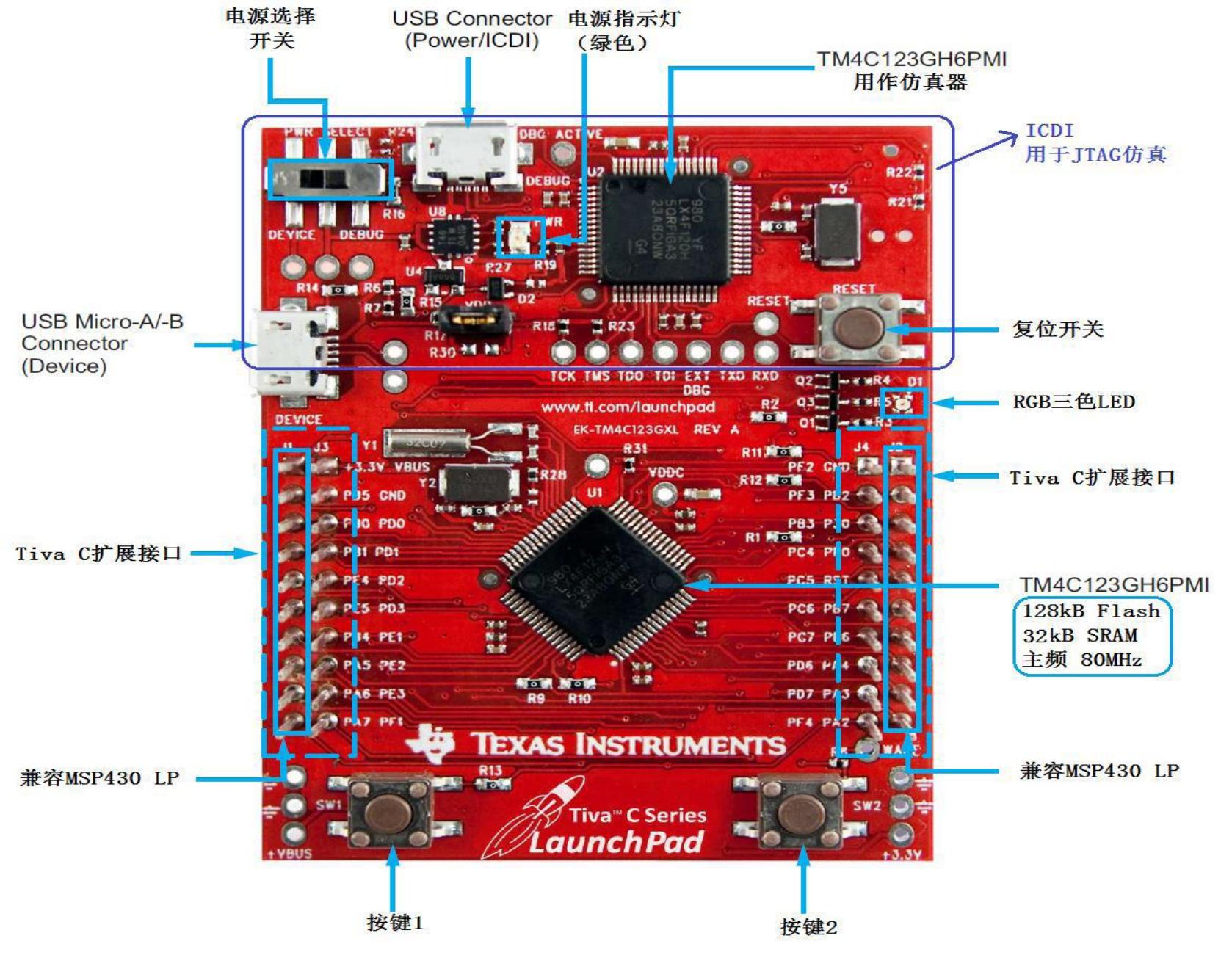


图 1-1 Tiva C LaunchPad 板卡

**1.1** 电源

板子通过USB口供电。有两个USB口（Device和Debug； USB座子边上有丝印字符表示）， 这两个USB口都可以给板子供电， 用开关来选择。

板子上有一个绿色的LED灯，用来指示3.3V供电。 对外输出两路电压：

l 3.3VDC 最大300mA；

l 5.0VDC 输出能力23mA~323mA，与3.3VDC的使用有关。 3.3V用的多了， 供这边使用 的就少了。

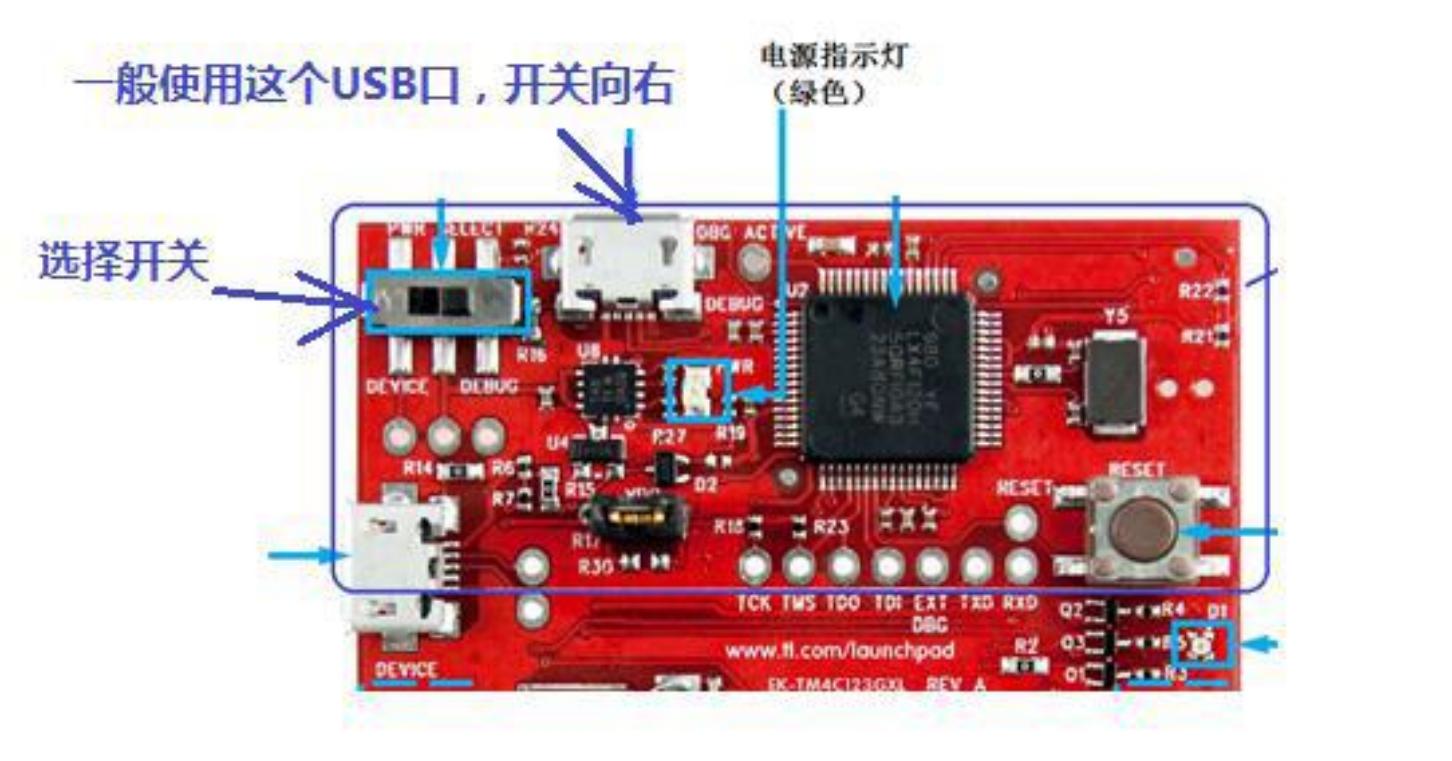


图 1-2 电源接口

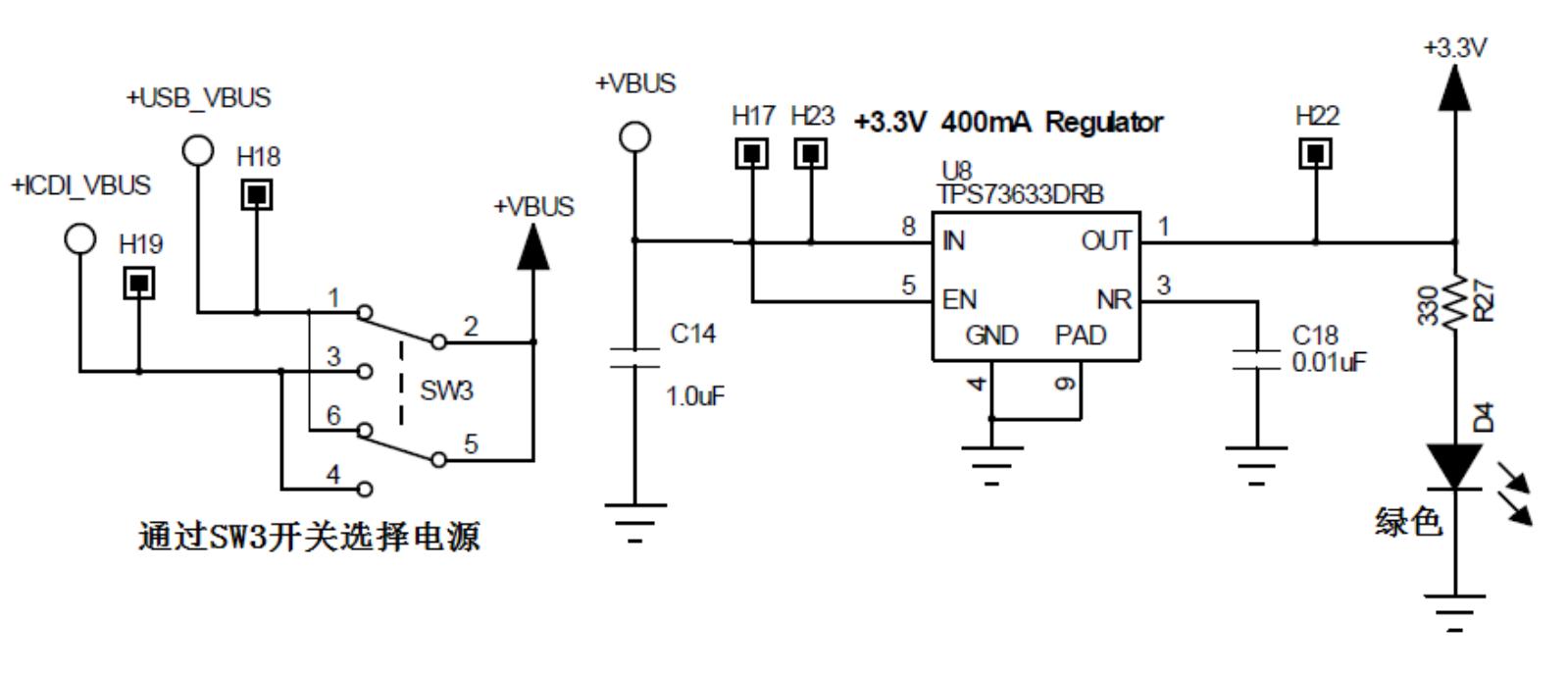


图 1-3 电源电路

**1.2** 复位电路

低电平有效复位。当按键按下时， 为低电平。



图 1-4 复位按键

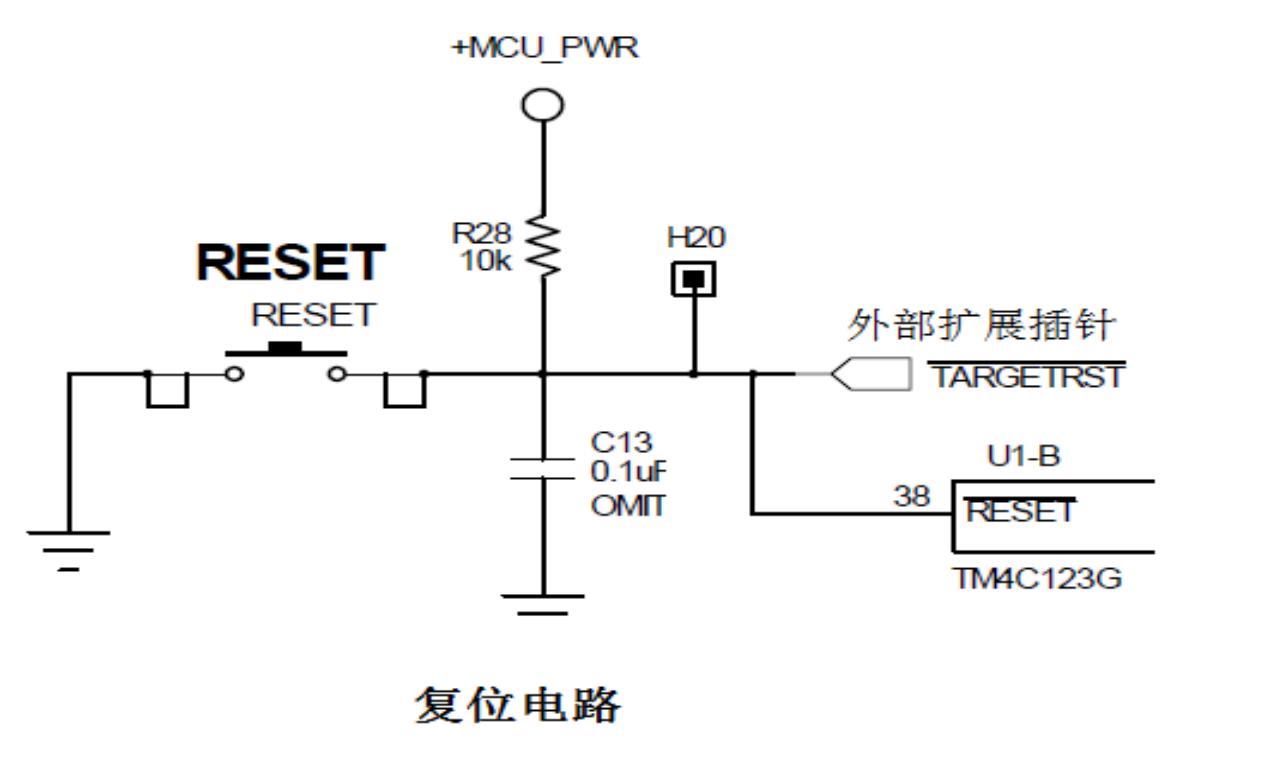


图 1-5 复位电路

**1.3** 时钟电路

板卡上有两个晶振。 16MHz晶振（Y2） 提供给处理器， 通过内部PLL， 倍频后再分频给内 核和外设使用； 晶振32.768KHz （Y1）用于休眠系统的时钟源， 休眠系统也是一大特色。

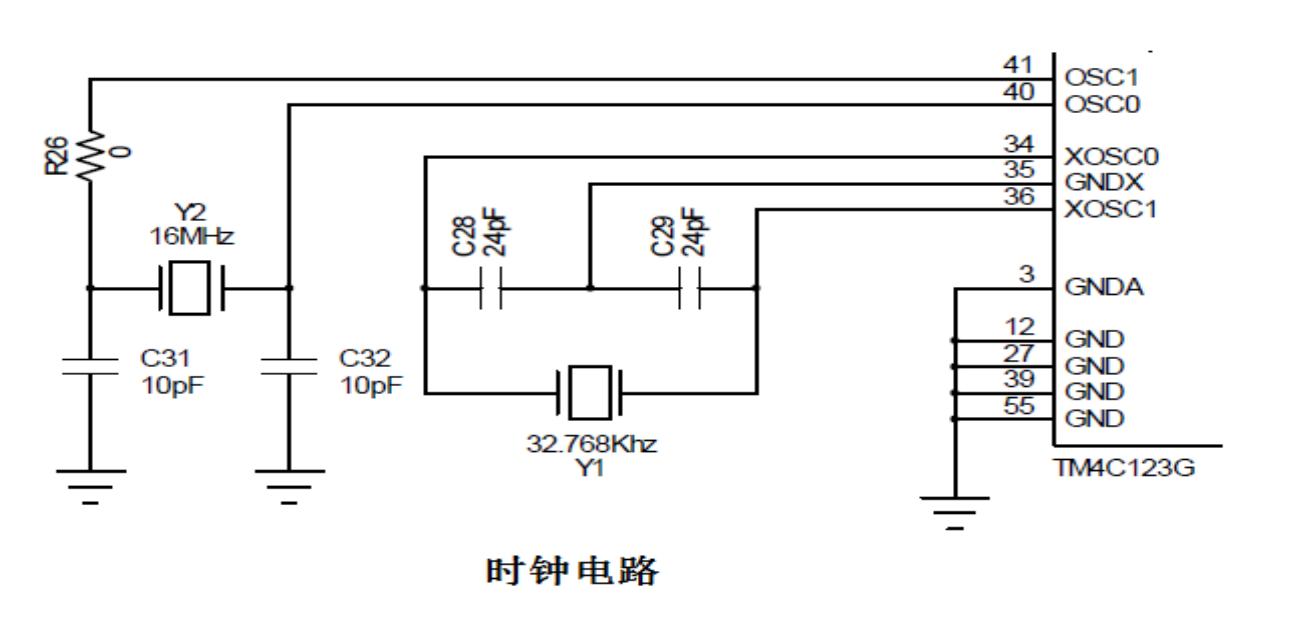


图 1-6 时钟电路

**1.4** 用户按键

按键按下时， IO口接地， 为低电平；

l GPIO-PF4---->按键SW1

l GPIO-PF0---->按键SW2



图1-7 用户按键

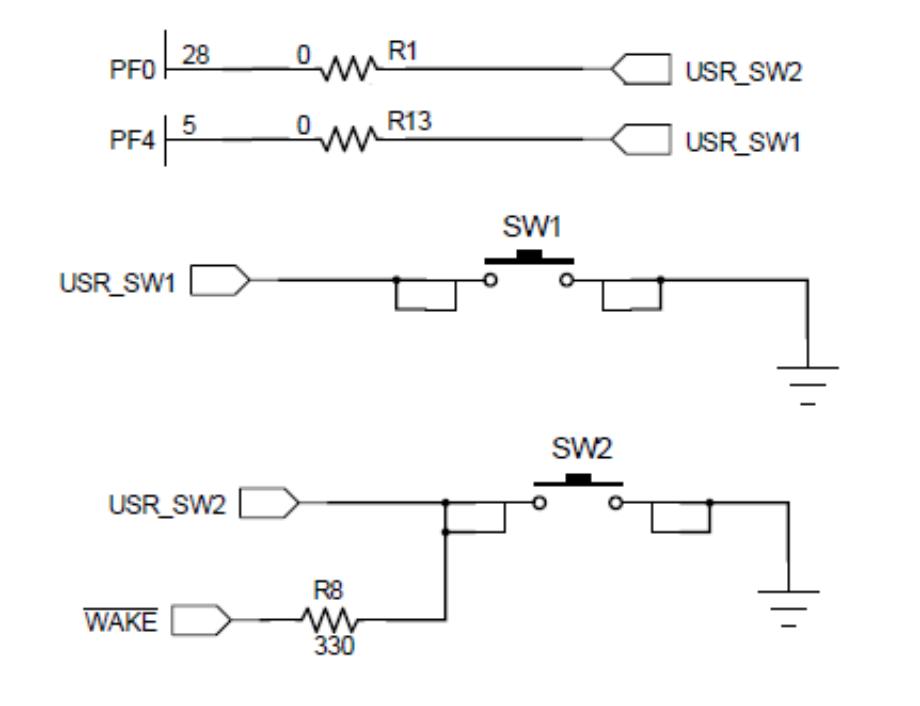


图1-8 用户按键电路

**1.5** 三色**LED**灯显示

LaunchPad 板子上带一个三色的 LED 灯， 位于复位按键下边。 IO 口输出高电平时，三

极管导通， LED 灯亮。

l GPIO-PF1---->RGB LED (红色)

l GPIO-PF2---->RGB LED (蓝色)

l GPIO-PF3---->RGB LED (绿色)

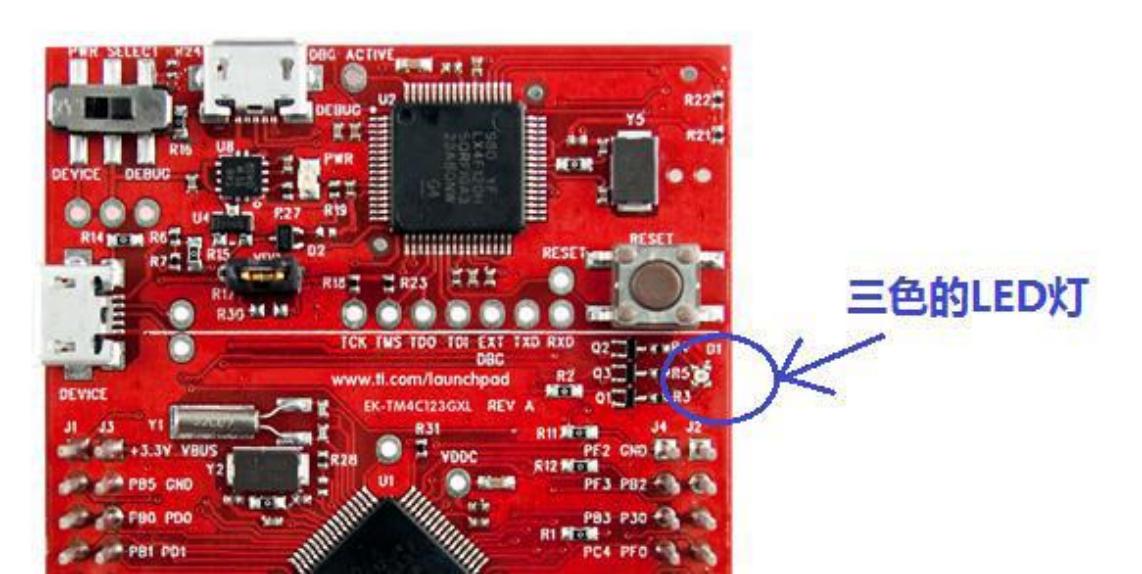


图 1-9 LED 灯

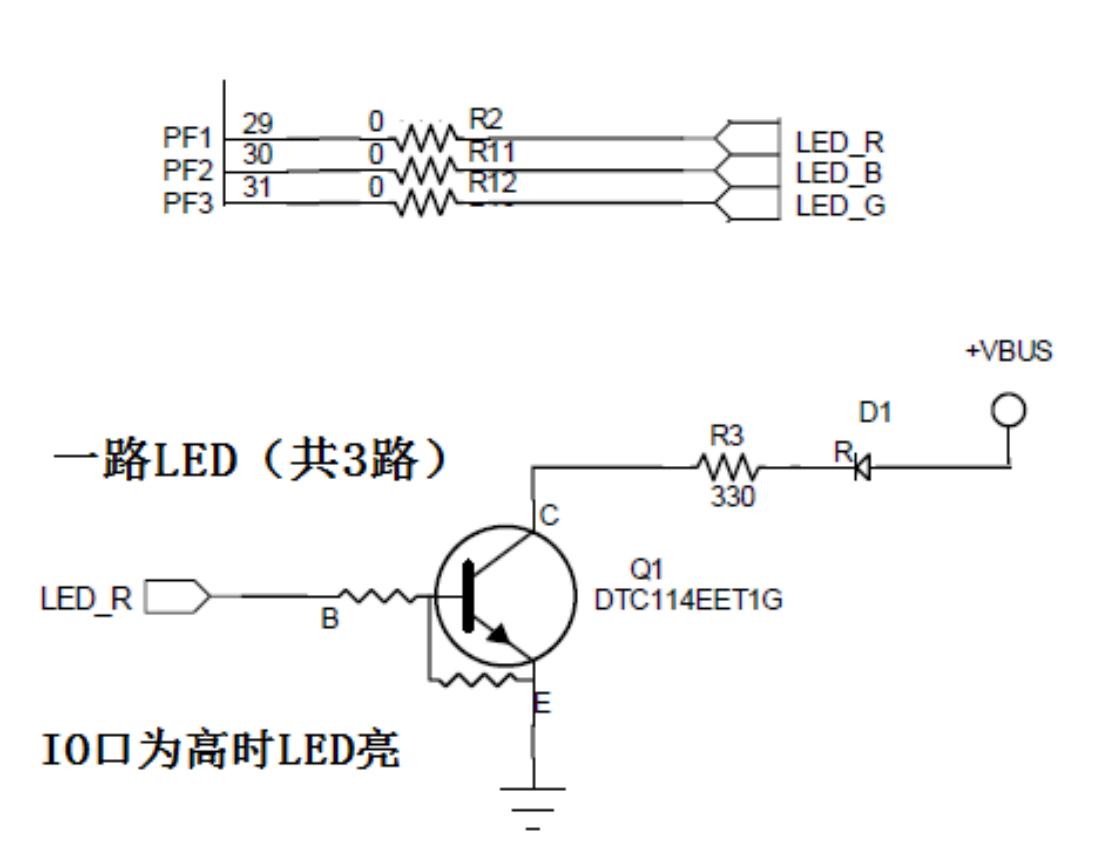


图 1-10 LED 接口电路

1.6 虚拟串口

当LaunchPad板卡连接到电脑时， 作为调试器的同时， 也会有一个虚拟的串口可以使用。 处理器上使用的串口引脚为：

l PA0 ---->U0RX

l PA1 ---->U0TX

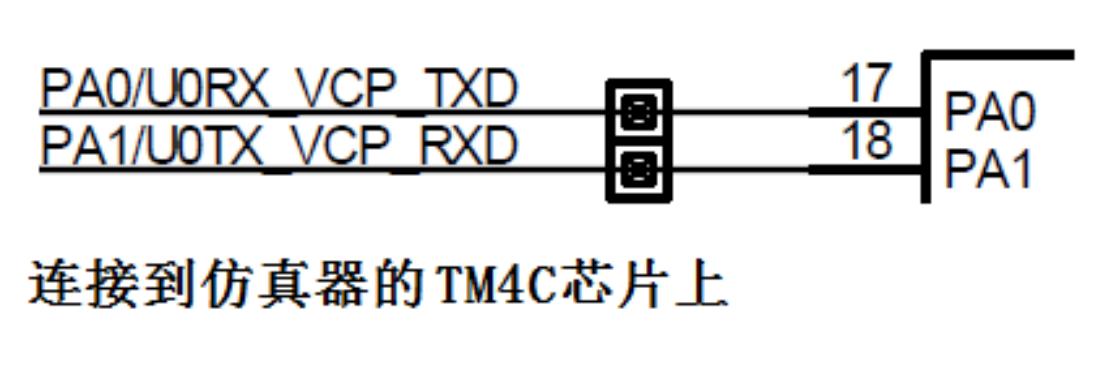


图 1-11 虚拟串口

1.7 调试接口

LaunchPad板自带一个板上仿真接口ICDI （In-Circuit Debug Interface）， 通过USB先连接电 脑后即可使用。 ICDI可用于编程或者调试TM4C123GH6PM。支持LM Flash Programmer或者兼 容的工具。

其它的调试接口可以连接到SWD （Serial Wire Debug）和SWO。注意， ICDI仅支持JTAG调 试。

l PC0---->TCK/SWCLK

l PC1---->TMS/SWDIO

l PC2---->TDI

l PC3---->TDO/SWO

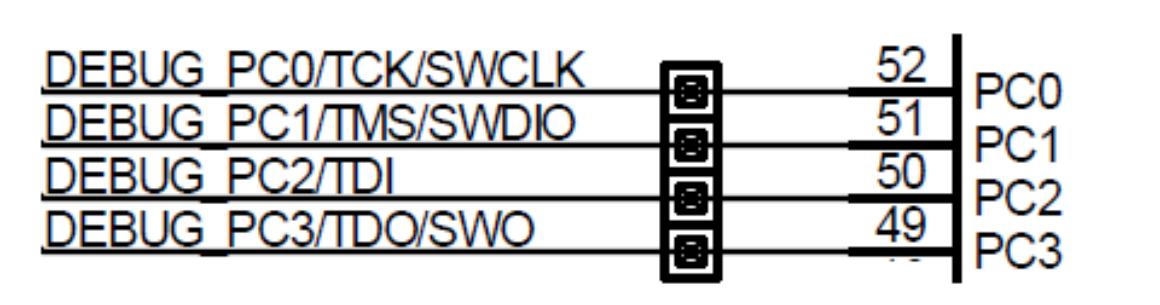


图1-12 调试接口

**1.8 USB**接口电路

一般情况，可以直接连接USB接口电路。

要使能 OTG 功能， 需要焊接板子上的电阻 R25 和 R29（两个均为 0 欧）。电阻将 USB ID、 USB Vbus 信号分别连接到芯片的 PB0、 PB1 上； 同时， 引脚 PB0 和 PB1 必须配置成 USB 引脚 模式。因为 PB0、 PB1 也作为扩展引脚上，为了避免器件损坏，当 R25 和 R29 电阻焊接时， 外接 BoosterPack 要特别注意 PB0 和 PB1 这两跟引脚不能使用。

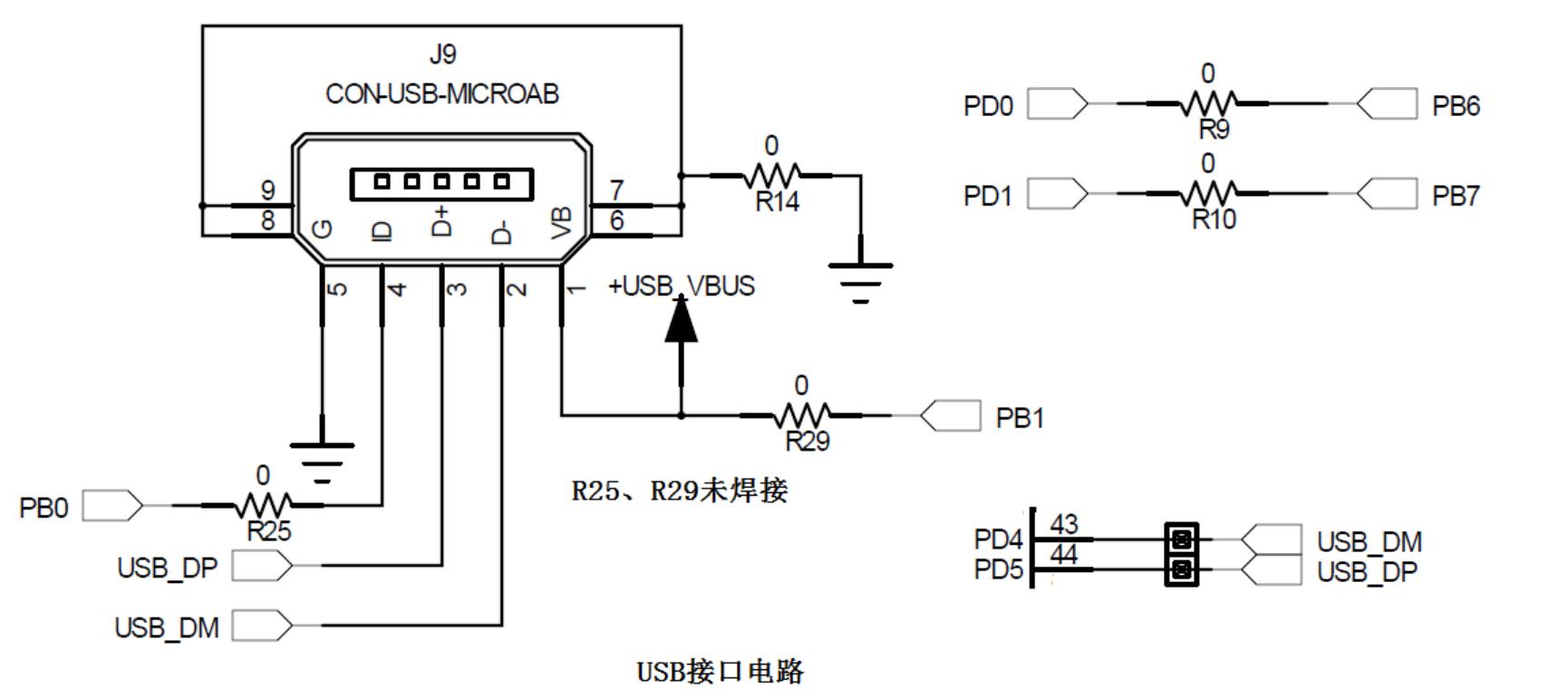


图 1-13 USB 接口电路

1.2 CCSv10 简介

**2.1 集成开发环境简介**

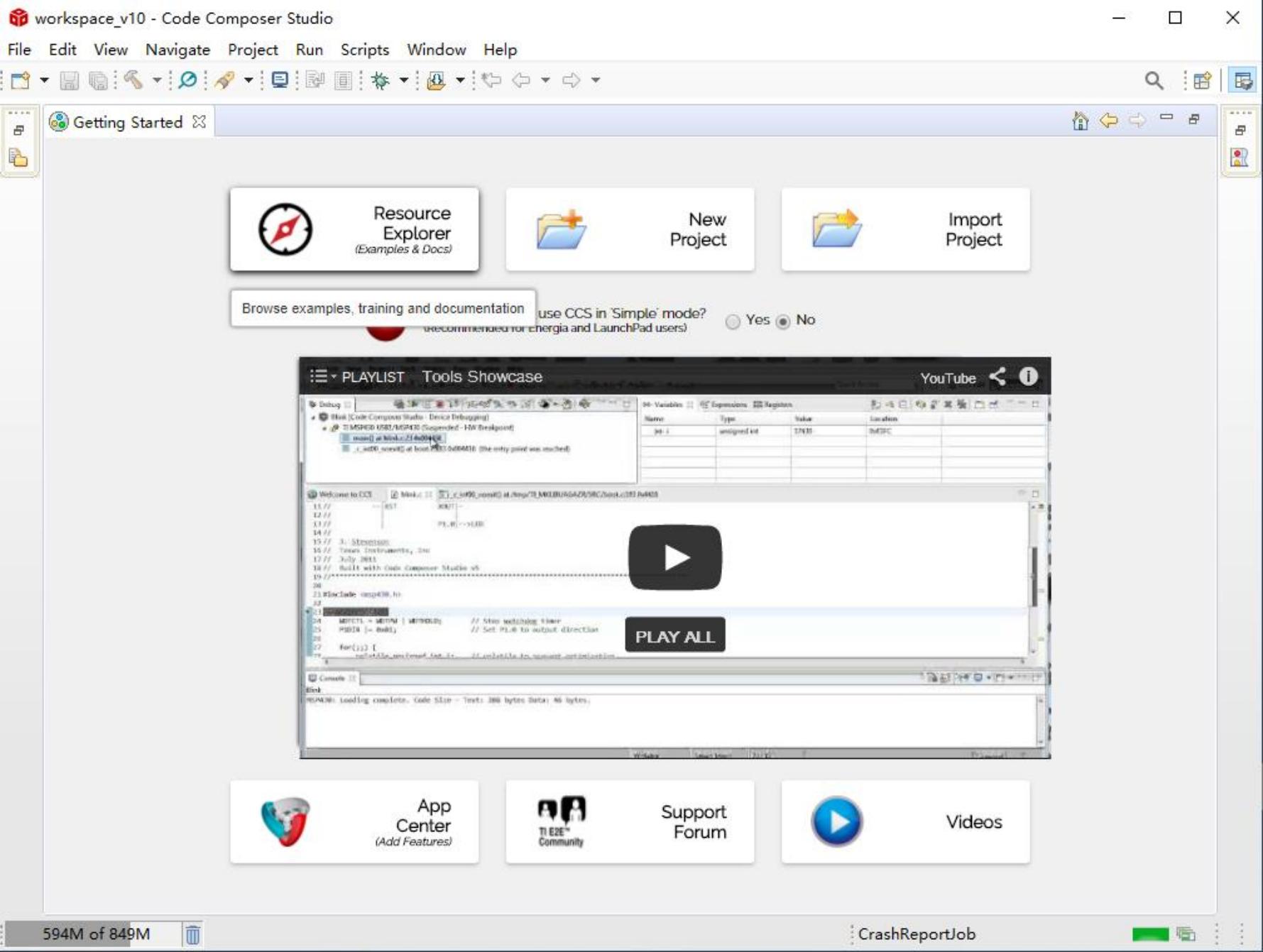


图 2.1 CCSv10 开发环境

Code Composer Studio™ (CCStudio) 是用于德州仪器 (TI) 嵌入式处理器系列的集成 开发环境 (IDE)。 CCStudio 包含一整套用于开发和调试嵌入式应用的工具。它包含适用于 每个 TI 器件系列的编译器、源码编辑器、项目构建环境、调试器、描述器、仿真器、实时 操作系统以及多种其他功能。直观的 IDE 提供了单个用户界面，可帮助用户完成应用开发 流程的每个步骤。借助于精密的高效工具， 用户能够利用熟悉的工具和界面快速上手并将功 能添加至他们的应用。

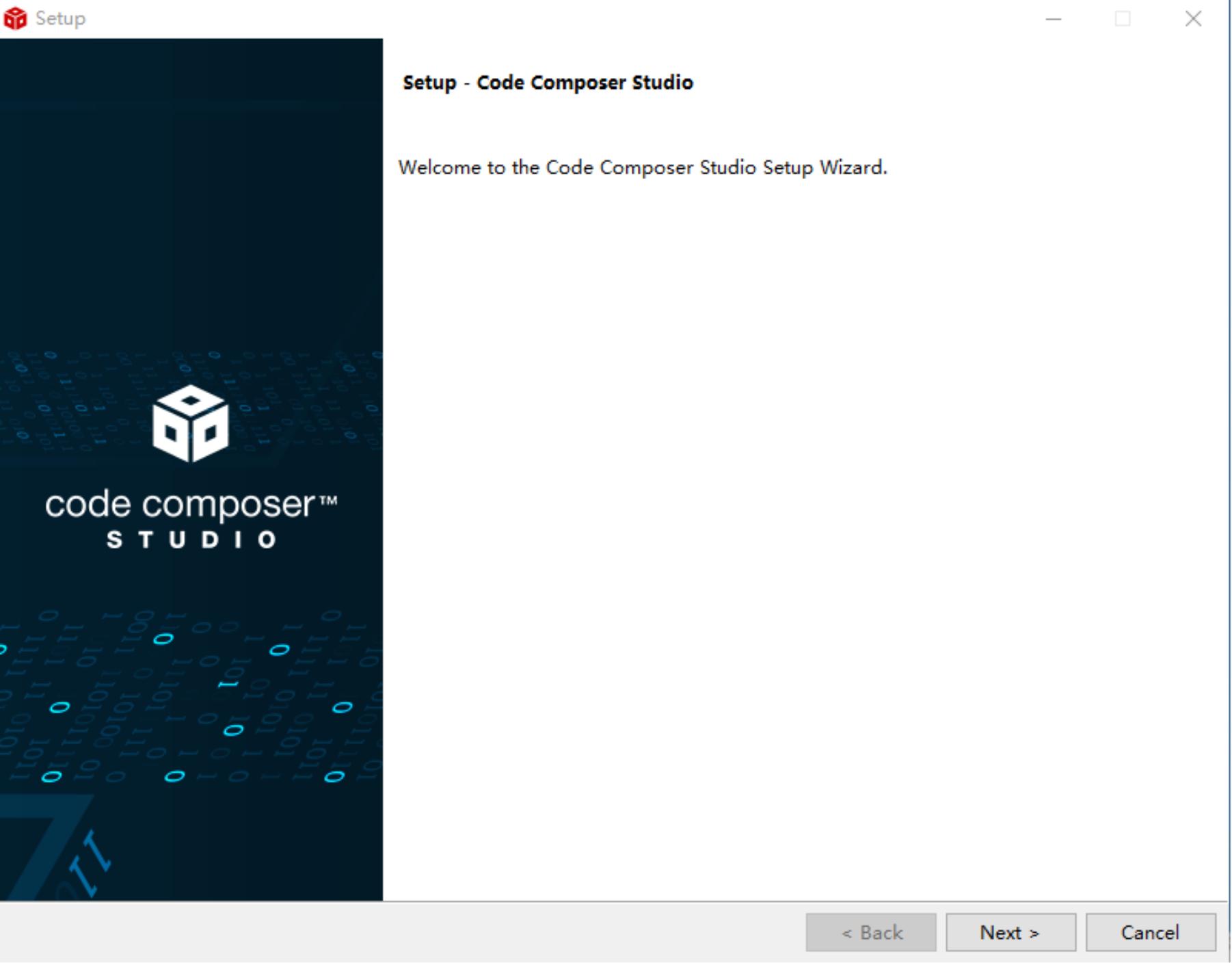
Code Composer Studio 以 Eclipse 开源软件框架为基础。 Eclipse 软件框架最初作为 创建开发工具的开放框架而被开发。 Eclipse 为构建软件开发环境提供了出色的软件框架， 并且逐渐成为备受众多嵌入式软件供应商青睐的标准框架。CCStudio 将 Eclipse 软件框架 的优点和 TI 先进的嵌入式调试功能相结合， 为嵌入式开发人员提供了一个引人注目、功能

丰富的开发环境。

Code Composer Studio 可在 Windows 和 Linux PC 上运行。并非所有功能或器件都与 Linux 兼容，详细信息请参见 Linux 主机支持。

**2.2 集成开发环境的安装**

CCS 的安装方式为常规软件安装方式，这里不再具体介绍。可在 TI 官网搜索 code composer studio，可以找到相关下载链接。可参考 TI 网页上的 WIKI 中有关 CCStudio 安 装的描述。



**2.3 Stellaris ICDI 驱动安装**

由于 CCS 安装完毕后已经支持 TI ARM 系列处理器，所以在安装完毕后可以采用自动 安装 Stellaris ICDI 驱动程序。流程如下：

1. 将 launchpad 板上方 USB 接口通过 USB 线连接电脑，同时将供电选择开关拨到 DEBUG 位置；

2. 完成步骤 1， windows 将自动安装驱动程序，如图 2.2：



图 2.2 ICDI 设备驱动安装

3. 安装完成后，打开控制面板—>硬件和声音—>设备和打印机，可以看到 ICDI 设备 已经存在，如图 2.3：

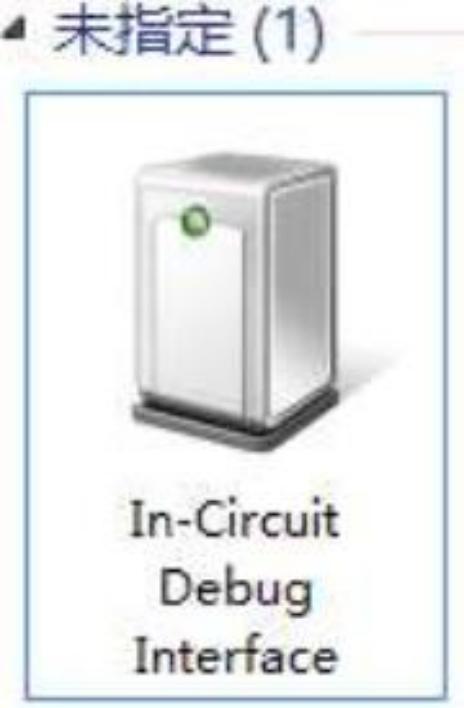


图 2.3 ICDI 设备驱动安装完成

4. 打开 ICDI 设备，可以观察到该设备已经运转正常，如图 2.4.



图 2.4 ICDI 设备运转正常

1.3 TivaWare 简介

**3.1 TivaWare 简介**

由于 Tiva 系列是基于 ARM 核的 MCU, 它提供资源丰富， 足以在其上运行一个简单的 操作系统。运行了操作系统的 MCU，对开发复杂功能的应用带来更多便利，比如代码的可 重用性、可维护性以及代码的模块化程度都得到了提高，但对开发人员也提出更高的要求， 比如要熟悉操作系统的特性，熟悉驱动程序等。

TivaWare 是以源码包的形式发布所有 Tiva 设备的驱动程序和代码实例的一个程序开发包， 它同时包含了 Tiva 系列 MCU 开发所需的头文件。 Tiva 驱动程序库的初衷是为基于操作系 统的应用开发准备的， 但它也可以在无操作系统的方式应用，为开发提供方便。

**3.2 TivaWare 的安装**

请按照以下步骤安装 TivaWare:

(1) 下载 TivaWare 软 件 包； 从 <http://www.ti.com/tool/sw-tm4c> 上 下 载 最 新 版 本 的 TivaWare 软件包。

(2) 双击下载的应用程序，进入 TivaWare 安装界面，如图 3-1；

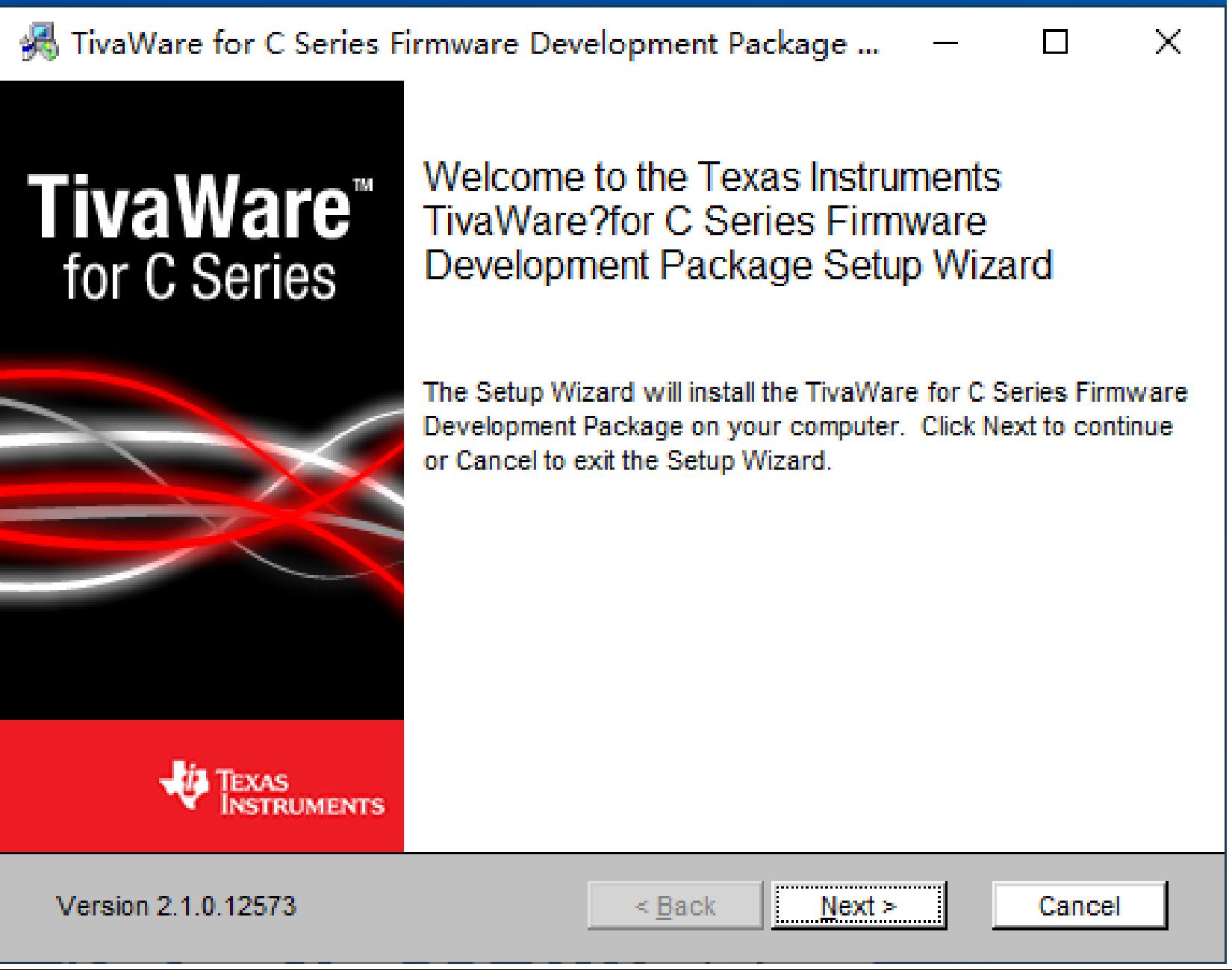


图 3.1 TivaWare 安装步骤 1

(3) 点击 next， 出现 TivaWare 的版本信息，如图 3-2；

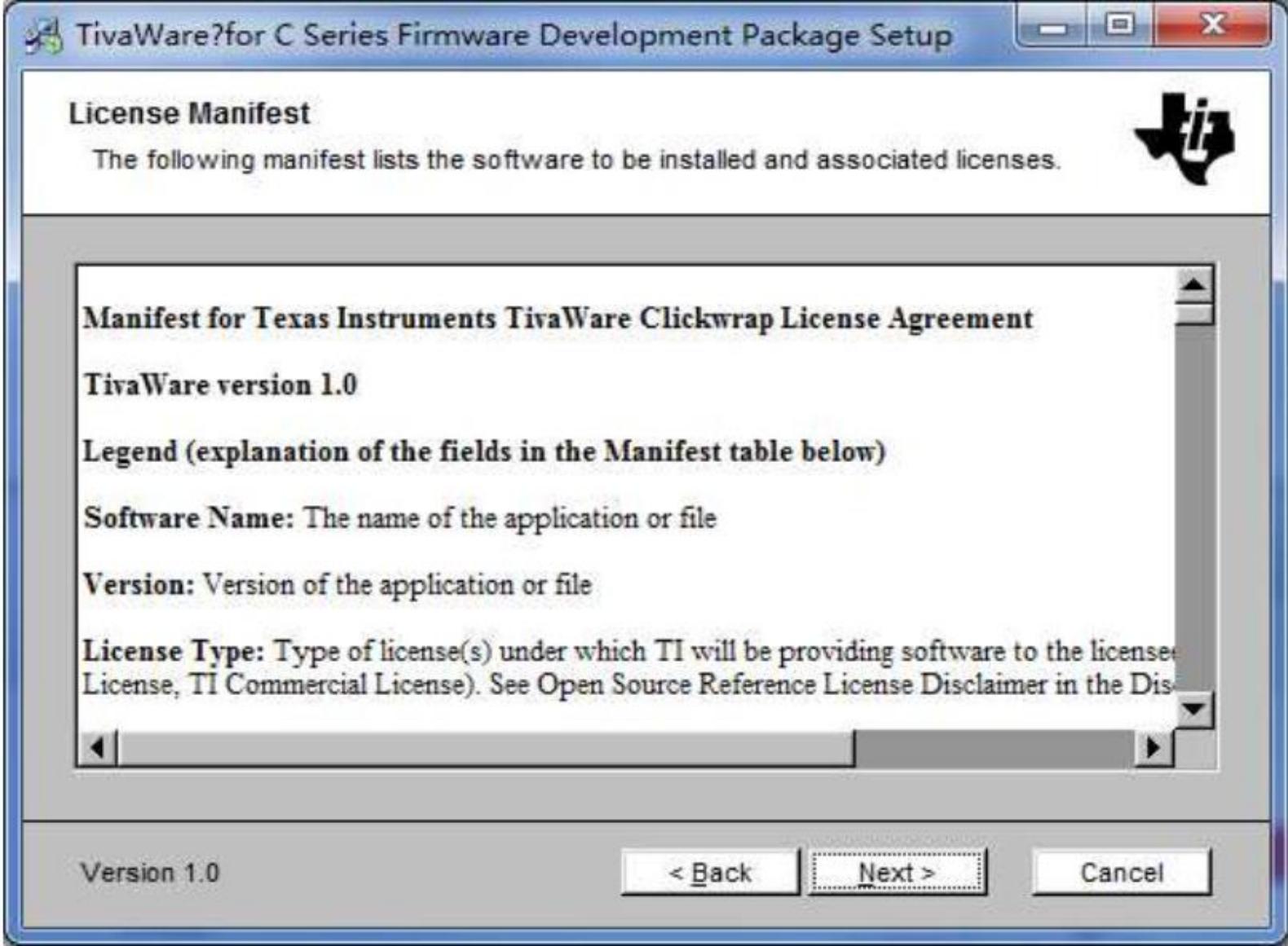


图 3.2 TivaWare 安装步骤 2

(4) 点击 next， 选择 I agree to the terms in the License Agreement，如图 3-3；



图 3.3 TivaWare 安装步骤 3

(5) 点击 next，选择软件包安装目录，如图 3-4。注意：你可以选择将 TivaWare 安装到

你自定义的目录下，但请记住安装路径，在后需加载系统文件目录时要到此安装目录下查找；

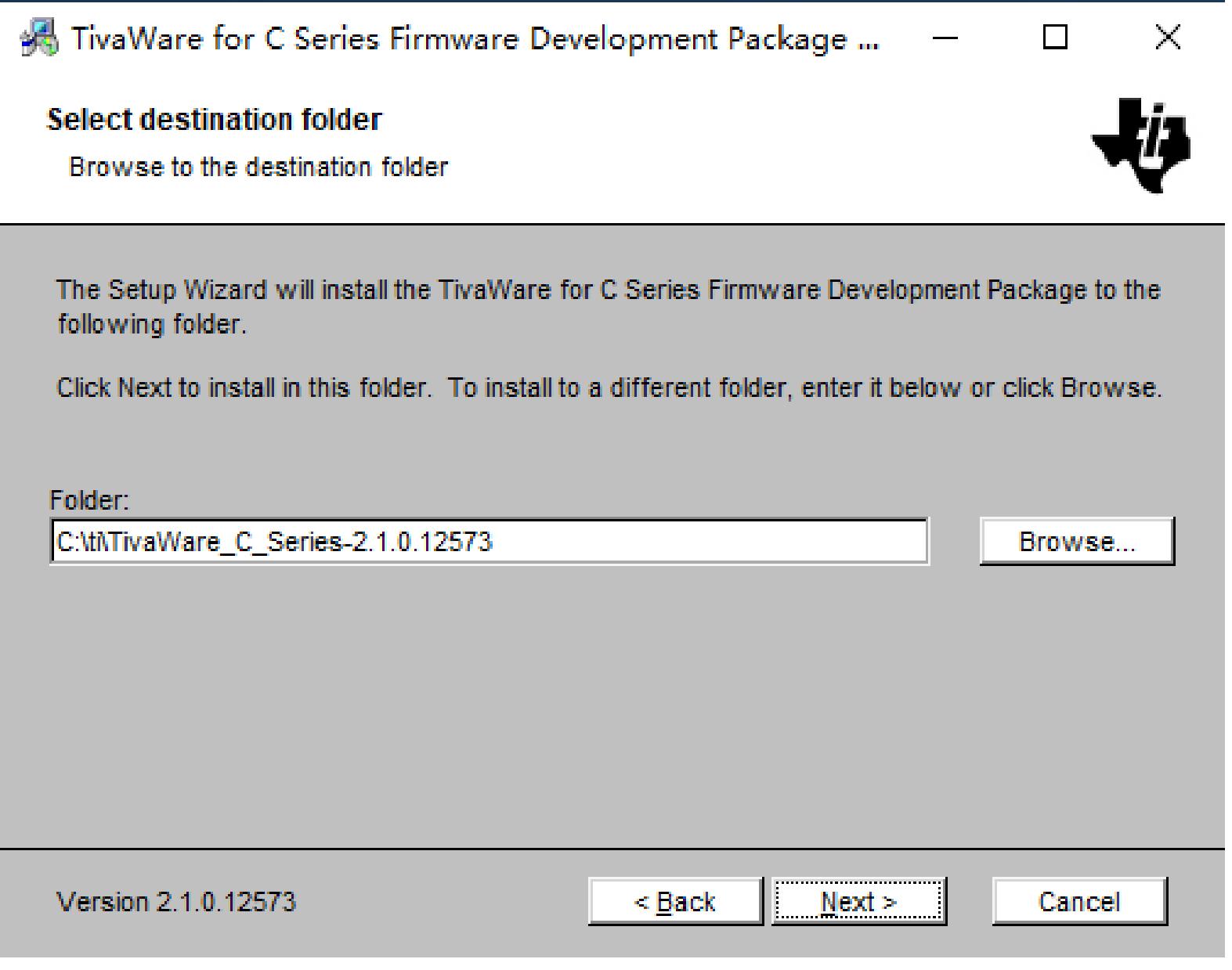


图 3.4 TivaWare 安装步骤 4

(6) 点击 next， 最后一步点击 install，如图 3-5；

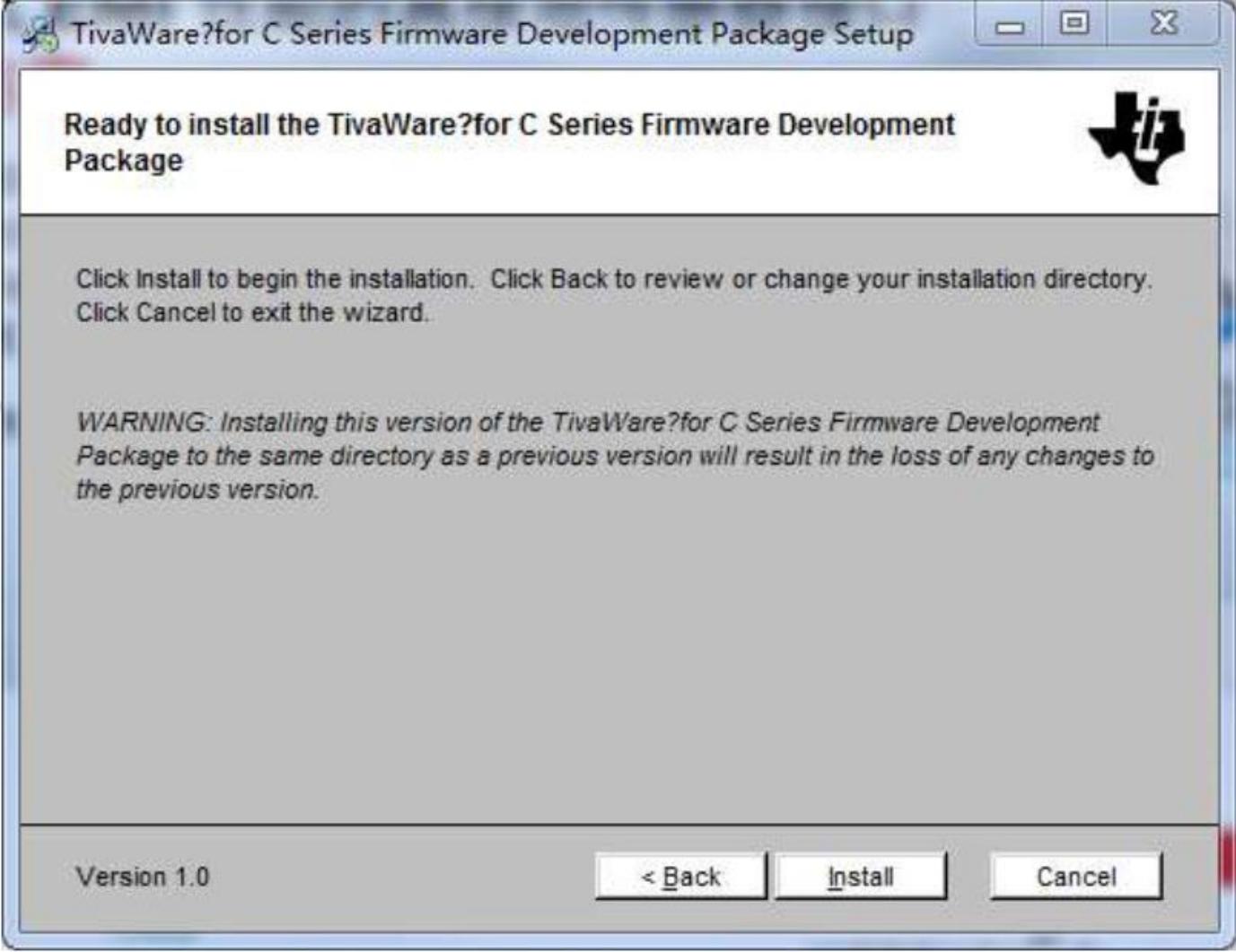


图 3.5 TivaWare 安装步骤 5

(7) 开始安装， 如图 3-6；

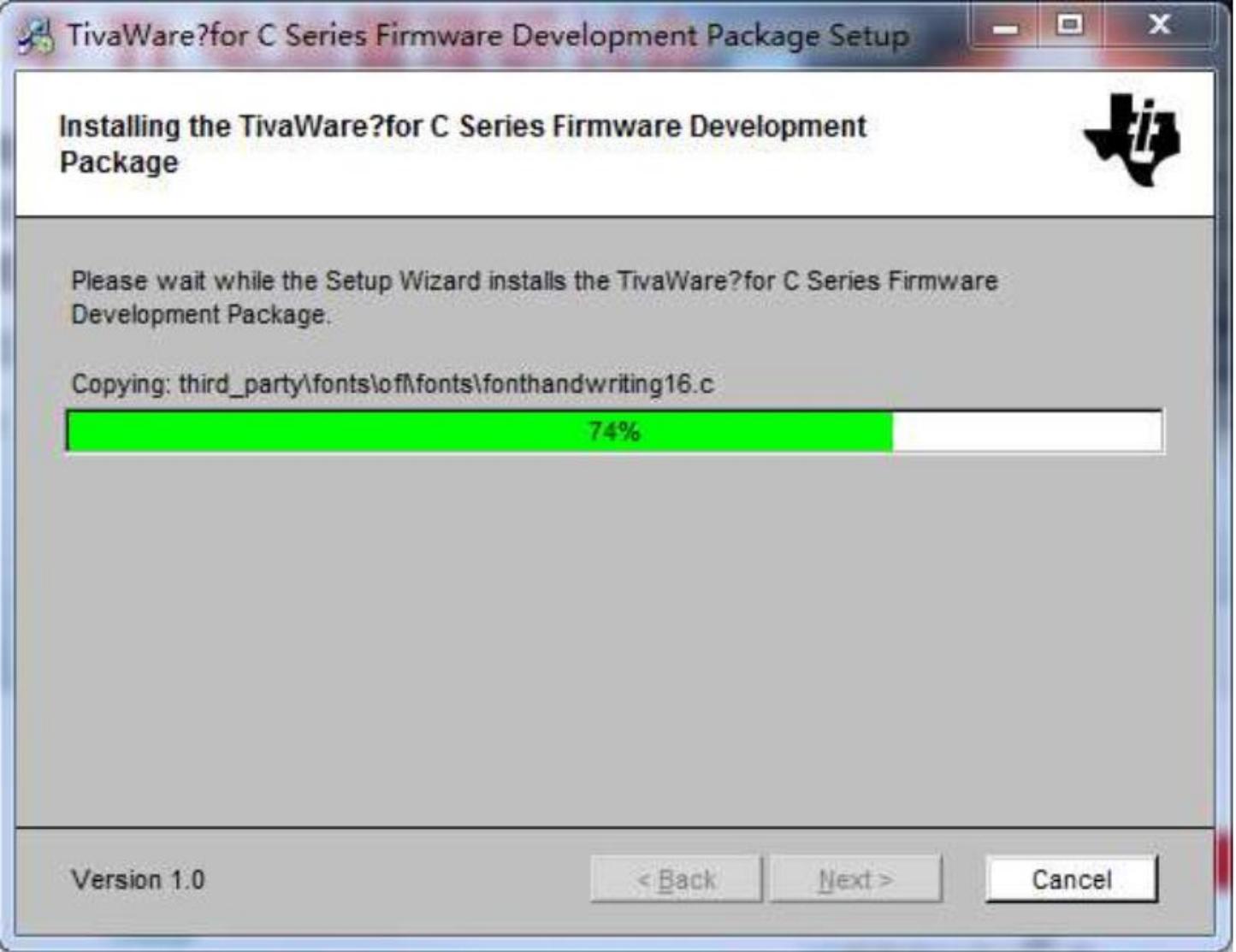


图 3.6 TivaWare 安装步骤 6

(8) 点击 Finish 完成安装，如图 3-7；



图 3.7 TivaWare 安装步骤 7

1.4 新建一个 CCS 工程

CCS 中的开发是以工程（Project） 为单元来组织管理的， 理解工程的概念对初学者来 说有一定的难度， 但至少可以先把工程理解为一个以工程名命名的文件夹， 这个文件夹包含 用户编写的一个或多个源代码文件、工程的配置信息文件、调试过程生成的中间文件、生成 的可用于下载到 MCU 中的输出文件以及其它与编码、调试、生成有关的其它文件。以工程 的方式组织管理方便用户修改、编译、 生成、调试、移植等工作都在一种工具里完成， 所以 我们称 CCS 为 IDE （集成开发环境）。

**4.1 新建 CCS 工程**

第一次运行 CCS 时会要求用户定义工作目录， 如图 4.1。新建的工程会放到这个 工作目录下。

**注意**： 改变这个目录时务必保证目录名里不出现中文（或其它非 ASCII 码表示的语种）， 不然会造成编译出错。

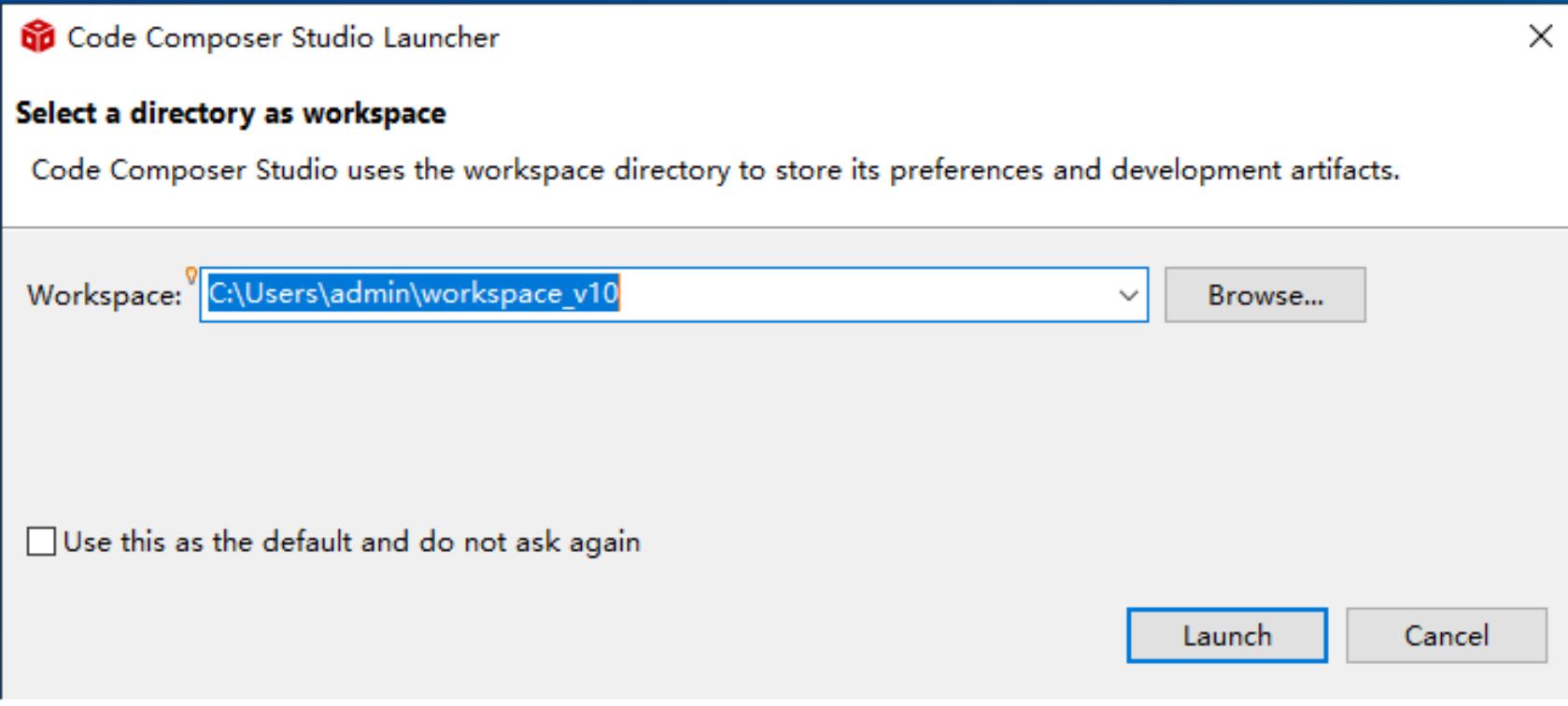


图 4.1 选取工作空间

下面开始新建一个以 Tiva TM4C123G 为目标设备的工程。在 CCS 中选择菜单项 File->New->CCS Project，将弹出如图 4.2 所示对话框，对应图中的标号，做如下说明：

（1） **为工程名**， 与工程相关的文件都会保存在以工程名命名的文件夹中， 请在

Project Name 中输入新建工程的名称，示例用“FirstProject”为例， 请勿用中文命名；

（2） **Output Type** 是指工程最终生成的结果的类型， 可以是可执行代码

（Executable）， 它是下载到设备上执行的代码； 也可以是静态库（Static library）， 它

通常作为功能模块被其它工程使用。在此我们先选择： Executable；

（3） **Location** 是工程文件夹的存放路径；

（4） 图中的 **Device** 部分是用来选择工程的目标设备（器件种类）和调试设备的， 因为 CCS 支持 TI 的全系列处理器，所以目标设备的列表会非常长， 为了方便用户 选择，按器件的大类（Family）、系列 （Variant）和目标设备的具体型号进行分类， 请在 Family 选择 ARM；

（5） **Variant** （目标设备所属的系列）， 选择 Tiva C Series；

（6） **目标设备的型号** （指开发板上 MCU 的型号）， 选择 Tiva TM4C123GH6PM；

（7） **目标设备与计算机的联接方式** （Connection）选择 Stellaris In-Circuit Debug Interface；

（8） 选择 **空工程** （带 main.c）， 然后单击 Finish 完成新工程的创建。

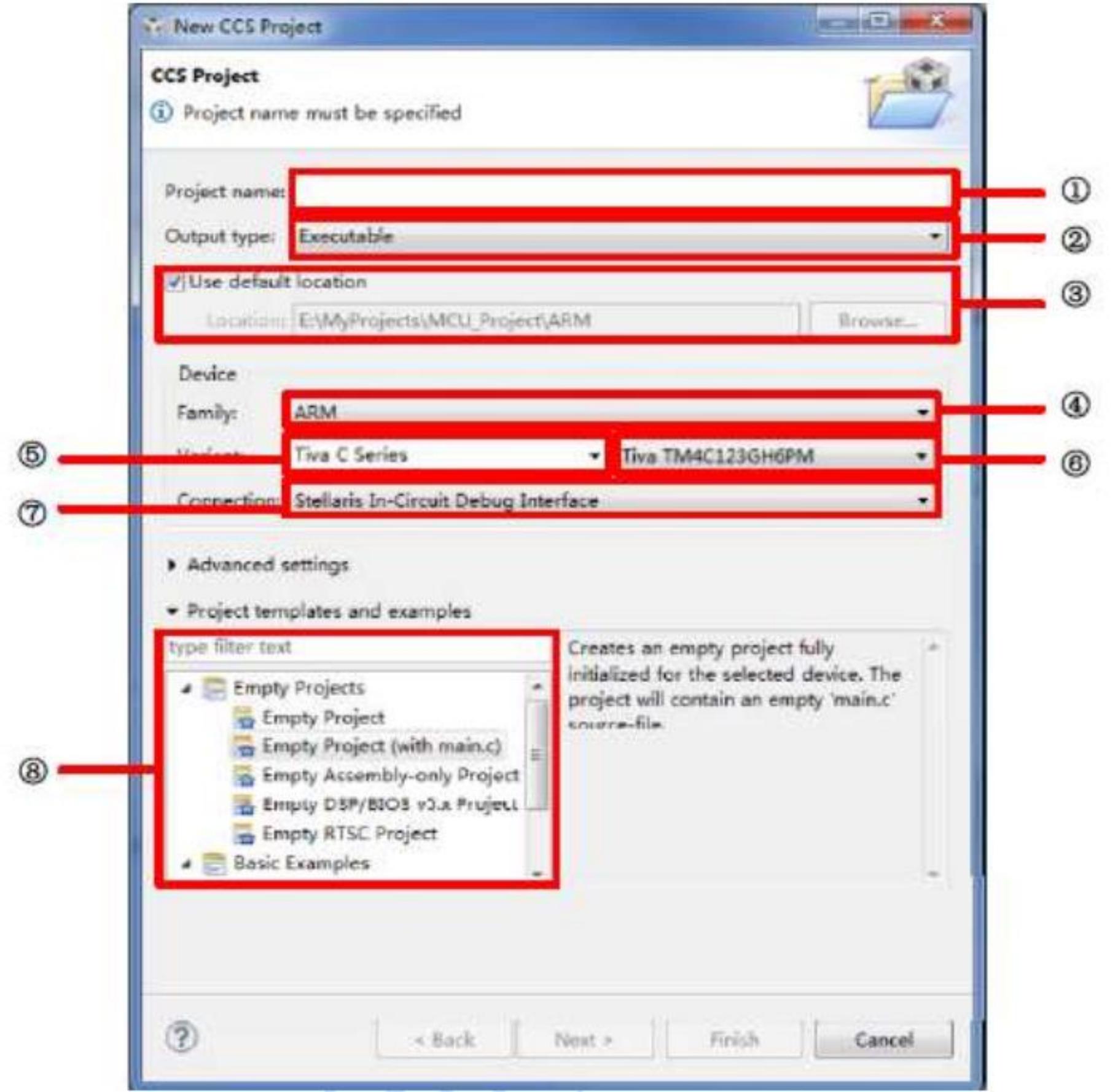


图 4.2 项目配置

工程创建后 CCS 进入如图 4-3 所示窗口。窗口左侧显示为项目的文件组成， 右侧 为 文 件 内 容。 在 工 程 上 右 键 单 击， 选 择 Properties ，出现如图 4-4 界面。选择 Build->ARMCompiler->Include Options。

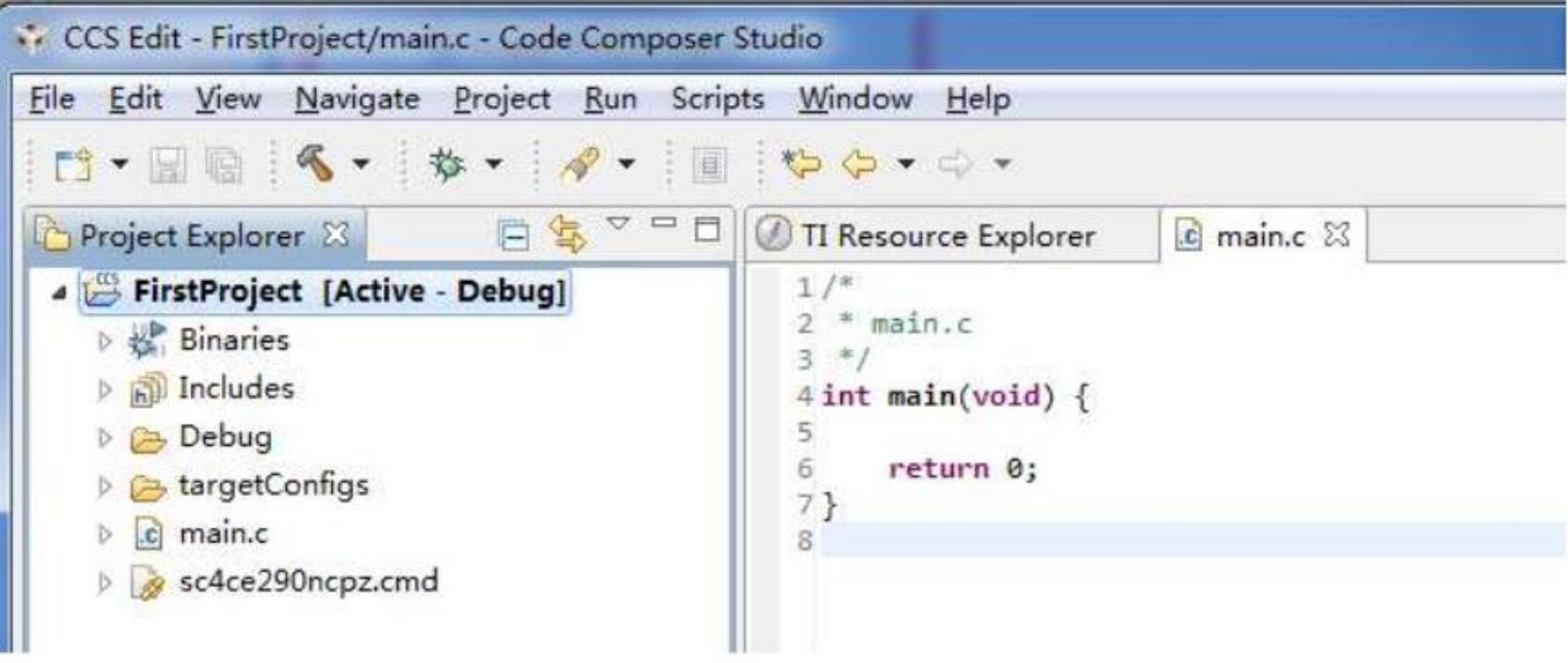


图 4.3 初始工程

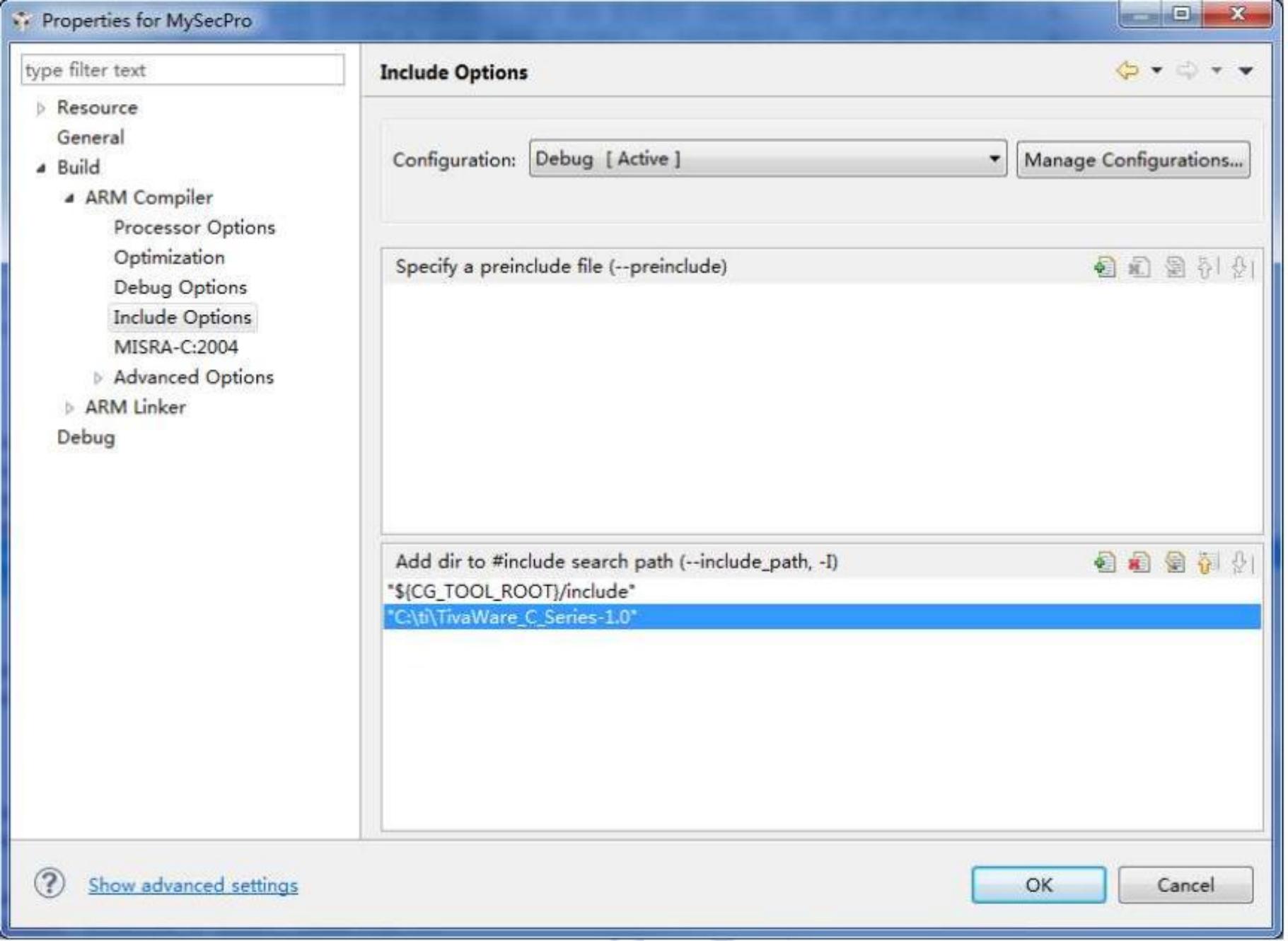


图 4.4 添加 include 路径

因为开发过程中会包含各种不同的头文件，这些头文件在用户的机器在 **Add dir to #include search path** 中添加路径“C:\ti\TivaWare\_C\_Series-2.1.0. 12573”，（注意： 您 安装 TivaWare 的路径有可能与示例的不同， 请按您的安装路径来设置）， 该文件夹内中

包含了开发所需的头文件。

至此，初始工程建立完毕。

**4.2 CCS 常用按钮**

**4.2.1 编译界面常用按钮**

图 4.1 为构建 一 个新的项目或者导入 一 个已经存在的项目时的编辑界面，此时，

图标处在激活状态（底色为白色）。

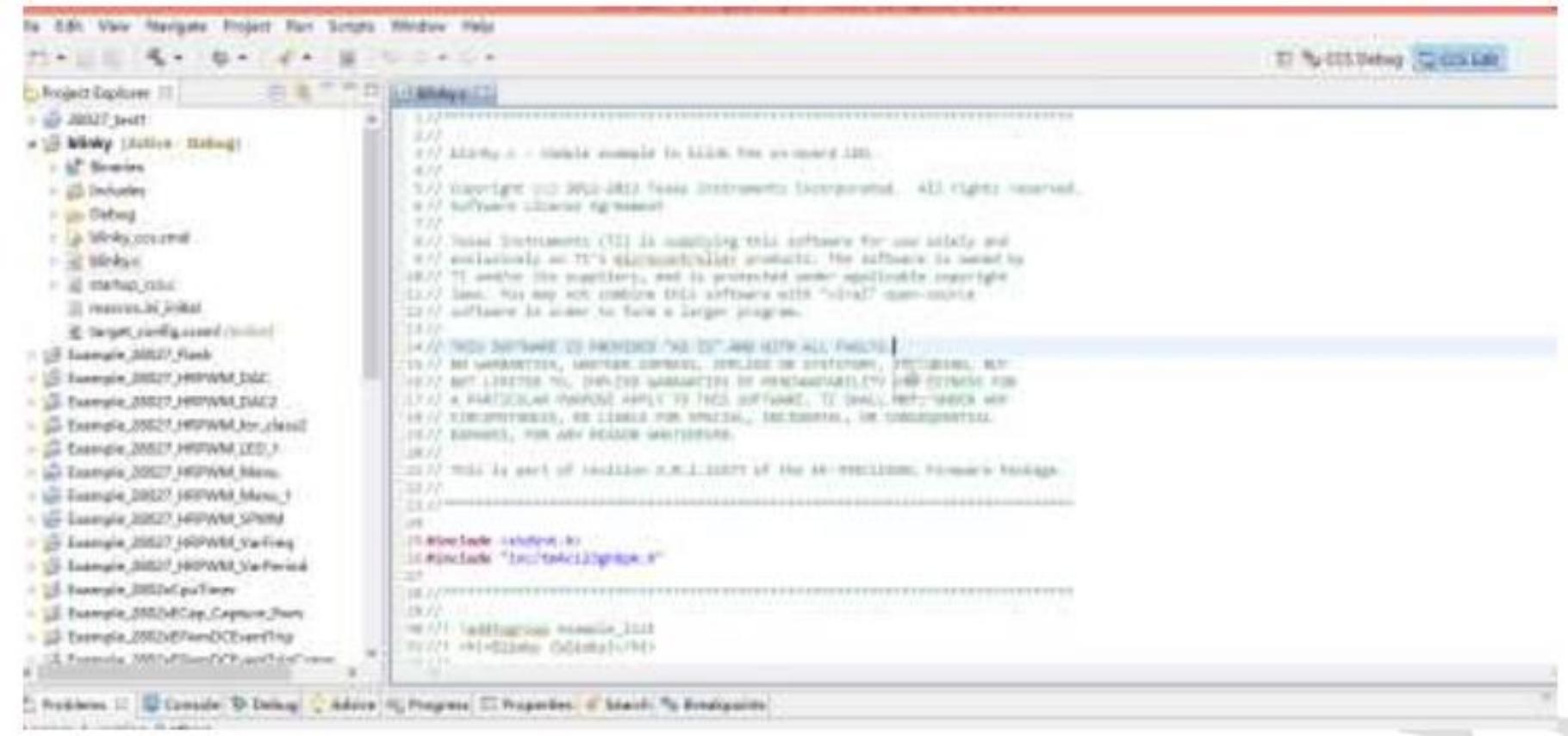


图 4.1 项目编辑页面

表 4.1 为 CCSv5 编译状态下的常用按钮。

|  |  |  |
| --- | --- | --- |
| **图标** | **名称** | **说明** |
|  | Build  ‘debug’ | 根据实际使用的情况， 该按钮对自建项目有效， 其下拉菜单 有两个可选项： Debug 允许对自建项目进行编译； Release 将自建项目释放成配置前的状态，此时编译将不能通过。 |
|  | Debug | 编译调试按钮， 该按钮在项目激活时有效， 其下拉菜单有历 次编译记录，可激活选定的项目并进行编译。 |
|  | Search | 搜索按钮，其下拉菜单有 3 种搜索方式： |

|  |  |  |  |
| --- | --- | --- | --- |
| **图标** | **名称** | **说明** | |
|  |  | File Search  c\c++ Search Git Search | 文件搜索； c\c++ 搜索； Git 搜索 |
|  | Back to ... | 记录通过项目管理器（Project Explorer） 导入的项目（或文 件）， 按该钮时， 将根据后进先出的顺序激活对应的项目 | |
|  | Last Edit Location | 记录通过按钮依次导入的项目或文件， 按该钮时， 将根据后 进先出的顺序激活对应的项目。当最后一个被激活后， 该按 钮关闭 | |
|  | CCS Debug | 切换到当前的调试界面 | |
|  | CCS Edit | 切换到当前的编辑界面 | |

**4.2.2 调试界面常用按钮**

在编译状态下， 按调试按钮， 当编译链接全部通过后， 可进入图 4.2 所示的项 目调试界面。上部椭圆形红框内为调试状态下的常用按钮， 下部椭圆形红框标注的阴影 条左侧有一个箭头表示程序将从这一行开始运行。

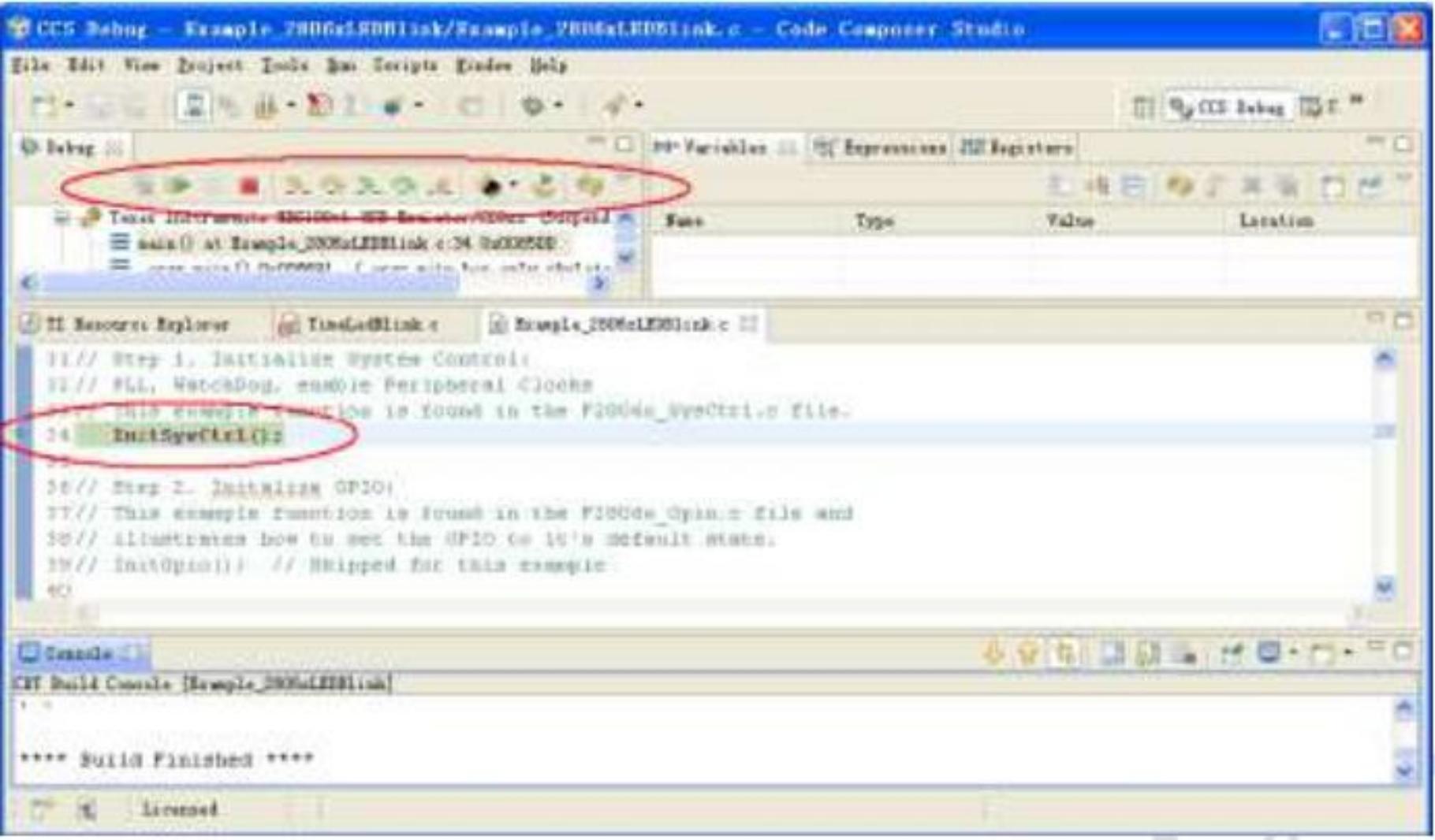


图 4.2 项目的调试界面 表 4.2 为调试界面的常用按钮。

表 4.2 CCSv5 调试界面的常用按钮

|  |  |  |
| --- | --- | --- |
| **图标** | **名称** | **说明** |
|  | Resume | 运行按钮 |
|  | Suspend | 暂停按钮 |
|  | Terminate | 结束按钮 |
|  | Step Into | 步入按钮 |
|  | Step Over | 步出按钮 |

|  |  |  |
| --- | --- | --- |
| **图标** | **名称** | **说明** |
|  | Assembly Step Into | 汇编步入按钮 |
|  | Assembly Step Over | 汇编步出按钮 |
|  | Step Return | 单步返回按钮 |
|  | Reset CPU or Reset Emulator | 复位 CPU 或者复位仿真机按钮 |
|  | Restart | 重新开始 |
|  | Refresh | 刷新按钮 |
|  | Real Time Mode | 实时模式按钮 |
|  | Continuous Reflash | 连续刷新按钮 |
|  | Load | 下载按钮 |
|  | Show logical structure | 逻辑结构显示按钮 |
|  | Collapse all | 全部折叠 |

|  |  |  |
| --- | --- | --- |
| **图标** | **名称** | **说明** |
|  | Open new View | 打开新的观察窗 |
|  | Pinto debug Context | 固定到调试上下文 |
|  | Open Perspective | 打开透视窗 |
|  | CCS edit Perspective | CCS 编辑透视窗 |
|  | disconnect hardware | 断开硬件 |

1.5 TivaWare 的使用

本章将会介绍 TIVAWARE 的使用， 即通过 CCS 导入 TIVAWARE 工程， 同时可以验 证 ICDI 驱动程序等是否已安装完成。

**5.1 导入 TIVAWARE**

TIVAWARE 中包含了 TI 官方的示例程序及相关驱动库， 通过使用这些驱动库和学习 相关例程，将大大加快开发的步骤。导入 TIVAWARE 的流程如下：

1. 打开 CCS，选择菜单栏 **View—>TI Resource Explorer**， 即可打开资源页面，如图 5.1；

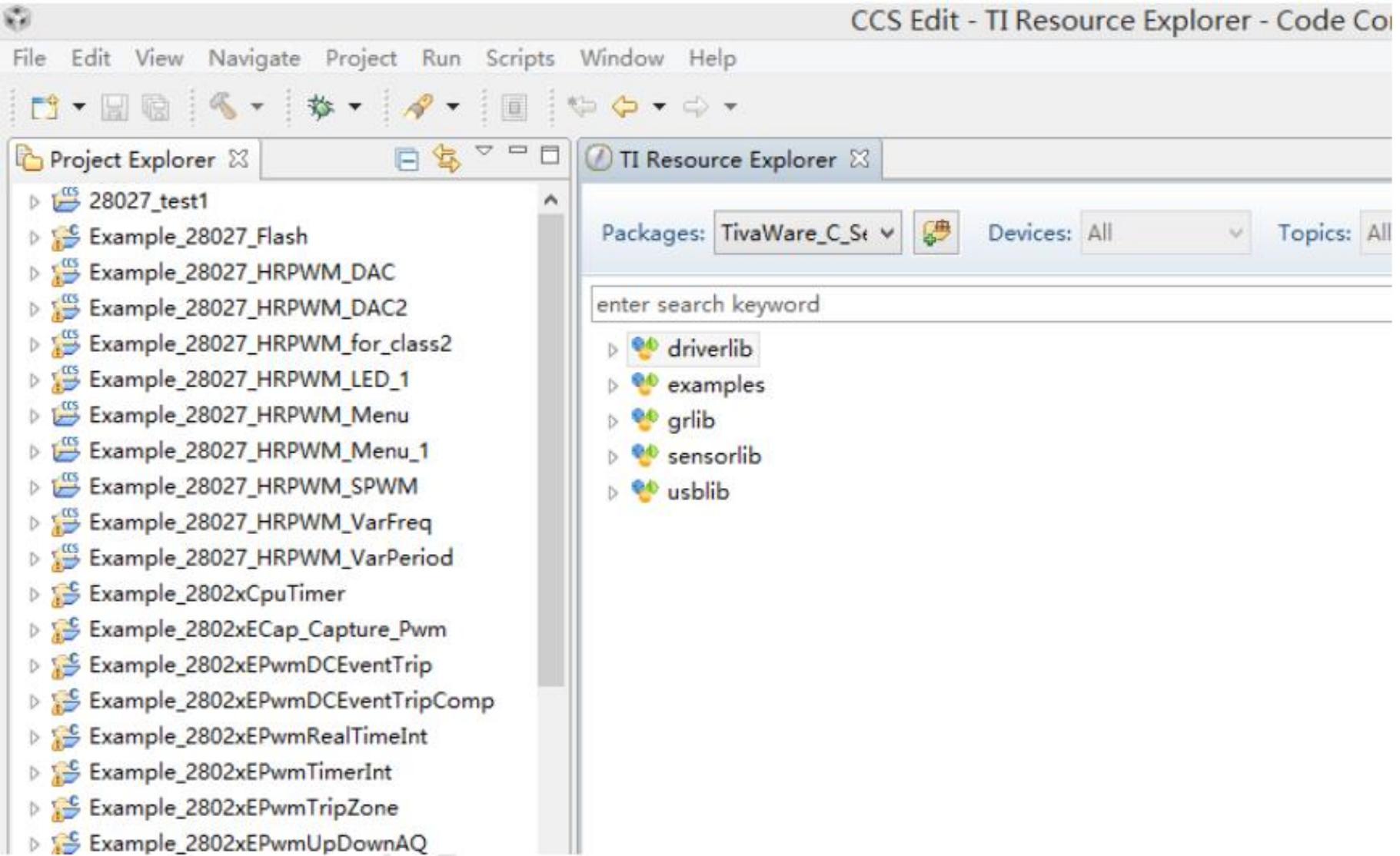


图 5.1 TI Resource Explorer

2. 初 次 使 用 TIVAWARE 需 要 首 先 导 入 安 装 的 TIVAWARE ，点击 explorer 中 **packages** 右边的添加按钮，如图 5.2；

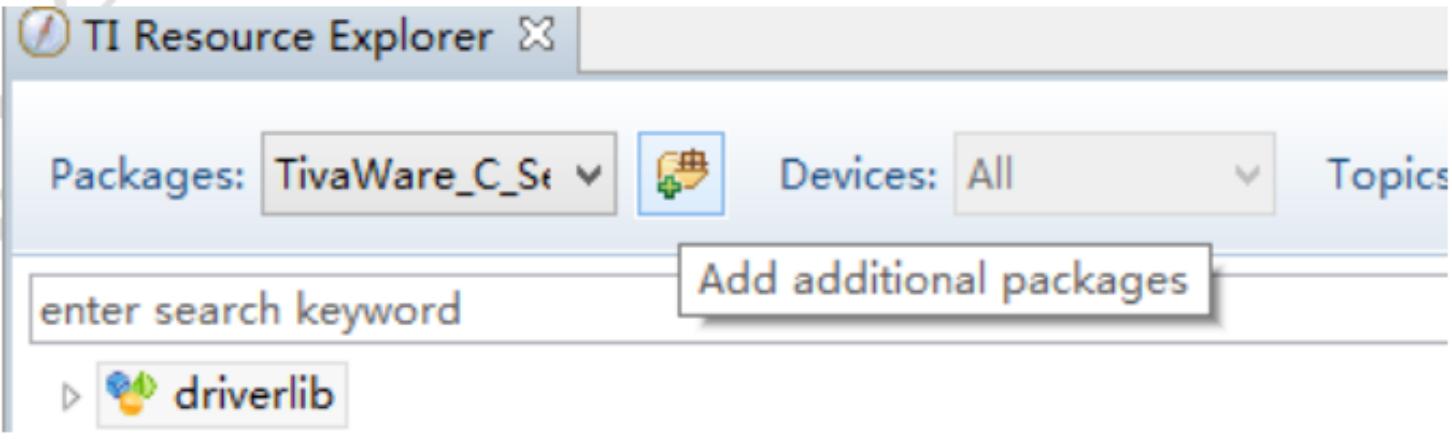


图 5.2 添加 packages

3. 在点击添加后， 在弹出的 **package configuration** 中添加安装好的 tivaware， 选取其 根目录即可，如图 5.3；

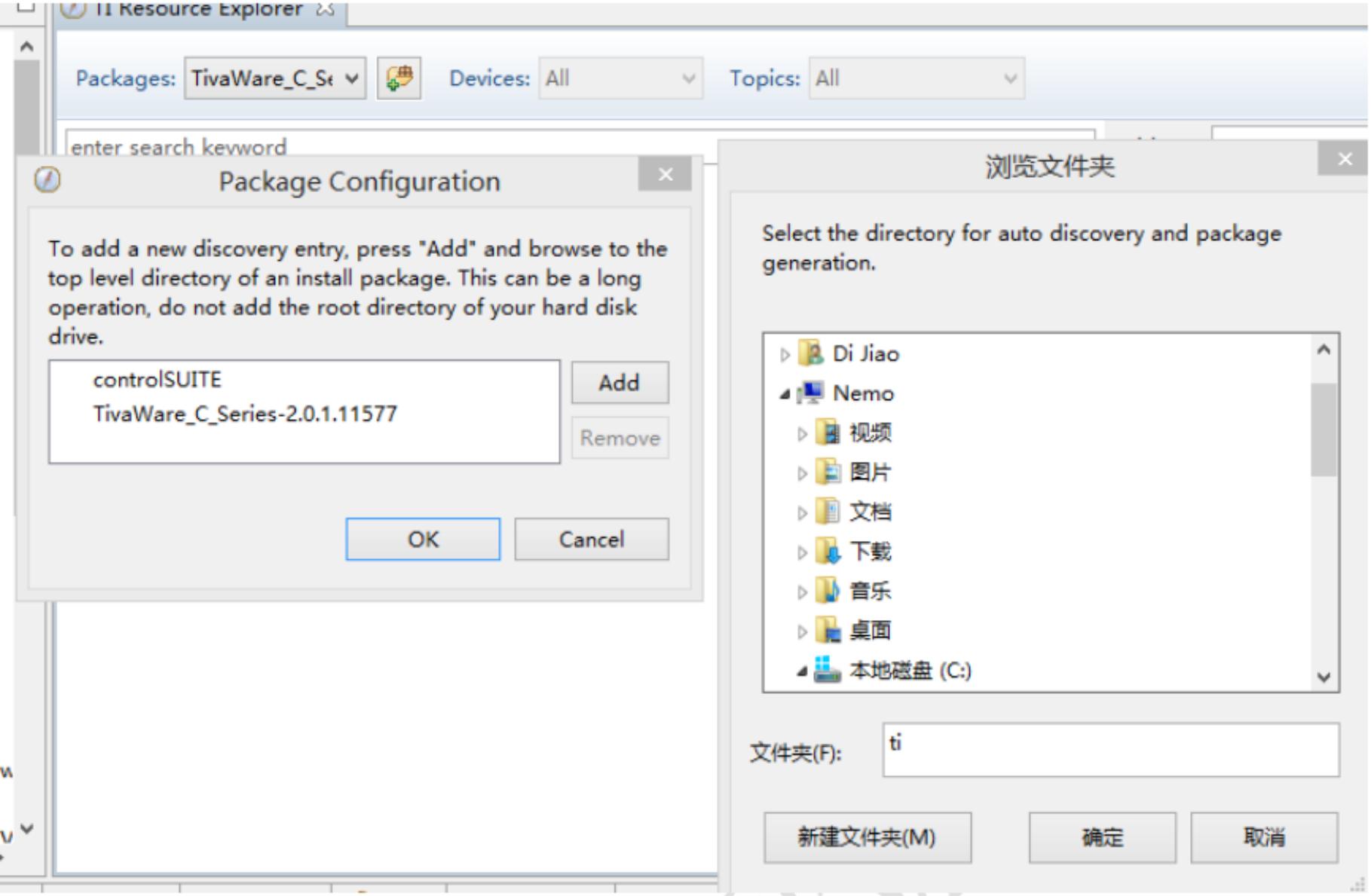


图 5.3 添加 packages

4. 添加后可打开 tivaware， 可观察其包含的资源， 如图 5.4；

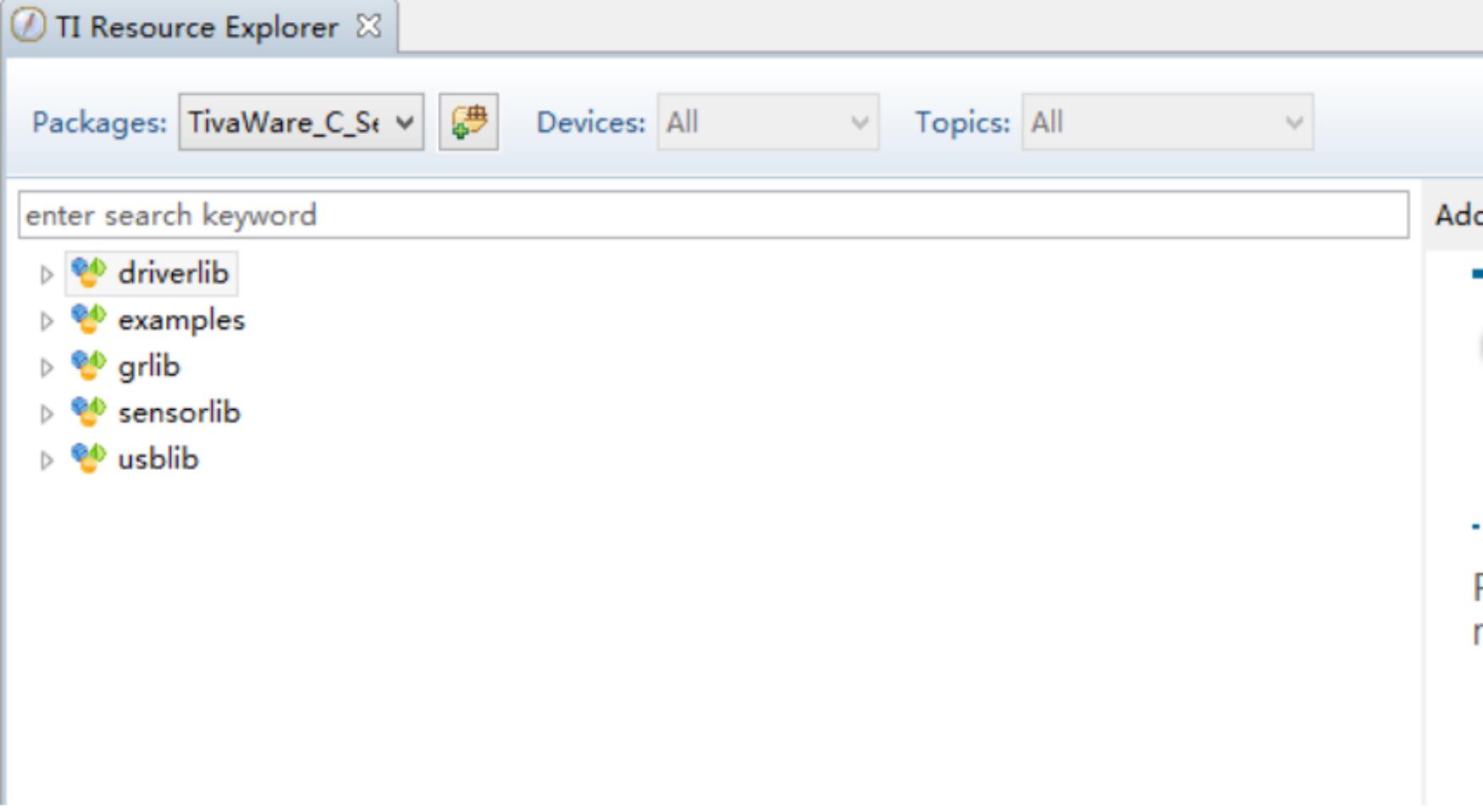


图 5.4 tivaware 资源

5. 至此， tivaware 导入完毕。

**5.2 导入示例工程**

这里我们以导入 tivaware 工程中的示例程序为例，流程如下：

1. 打开 TI Resource Explorer 中 tivaware， 选择列表中的 **example**， 如图 5.5；

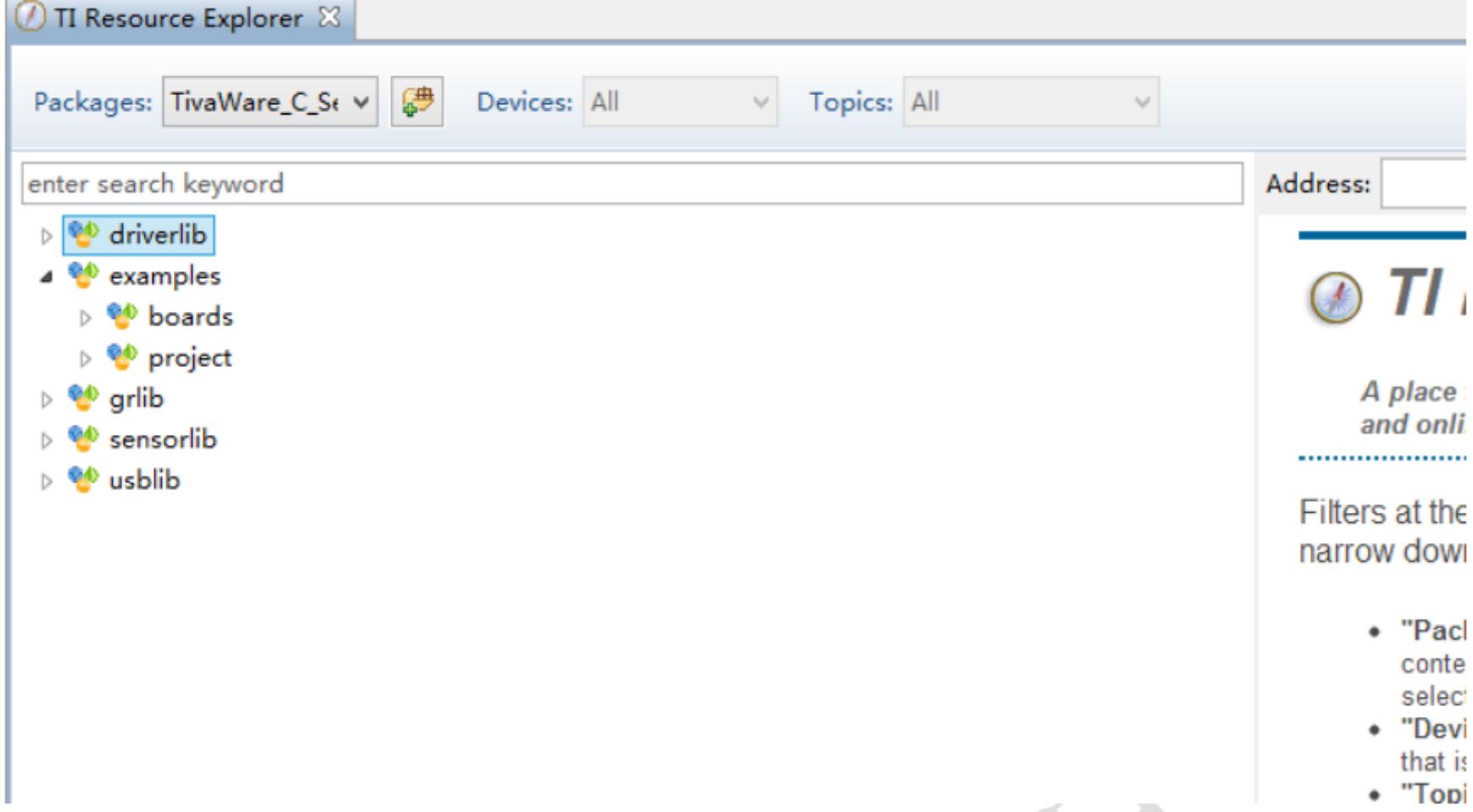


图 5.5 tivaware 资源

2. 选择 **examples—>boards—>ek-tm4c123gxl—>blinky**，如图 5.6；

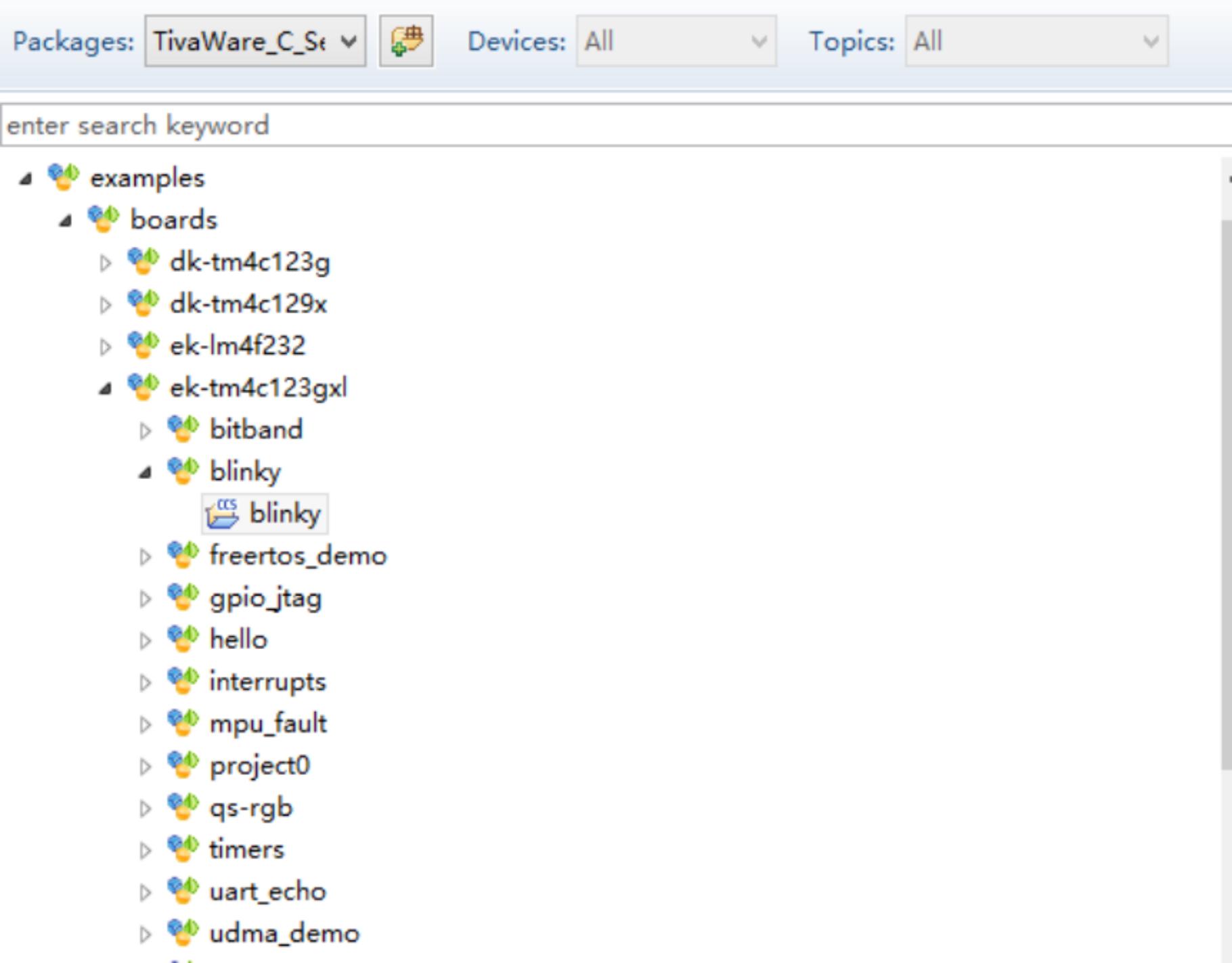


图 5.6 tivaware 资源

3. 此时，可以看到 **blinky 工程** 已出现在 explorer 右侧，如图 5.7；

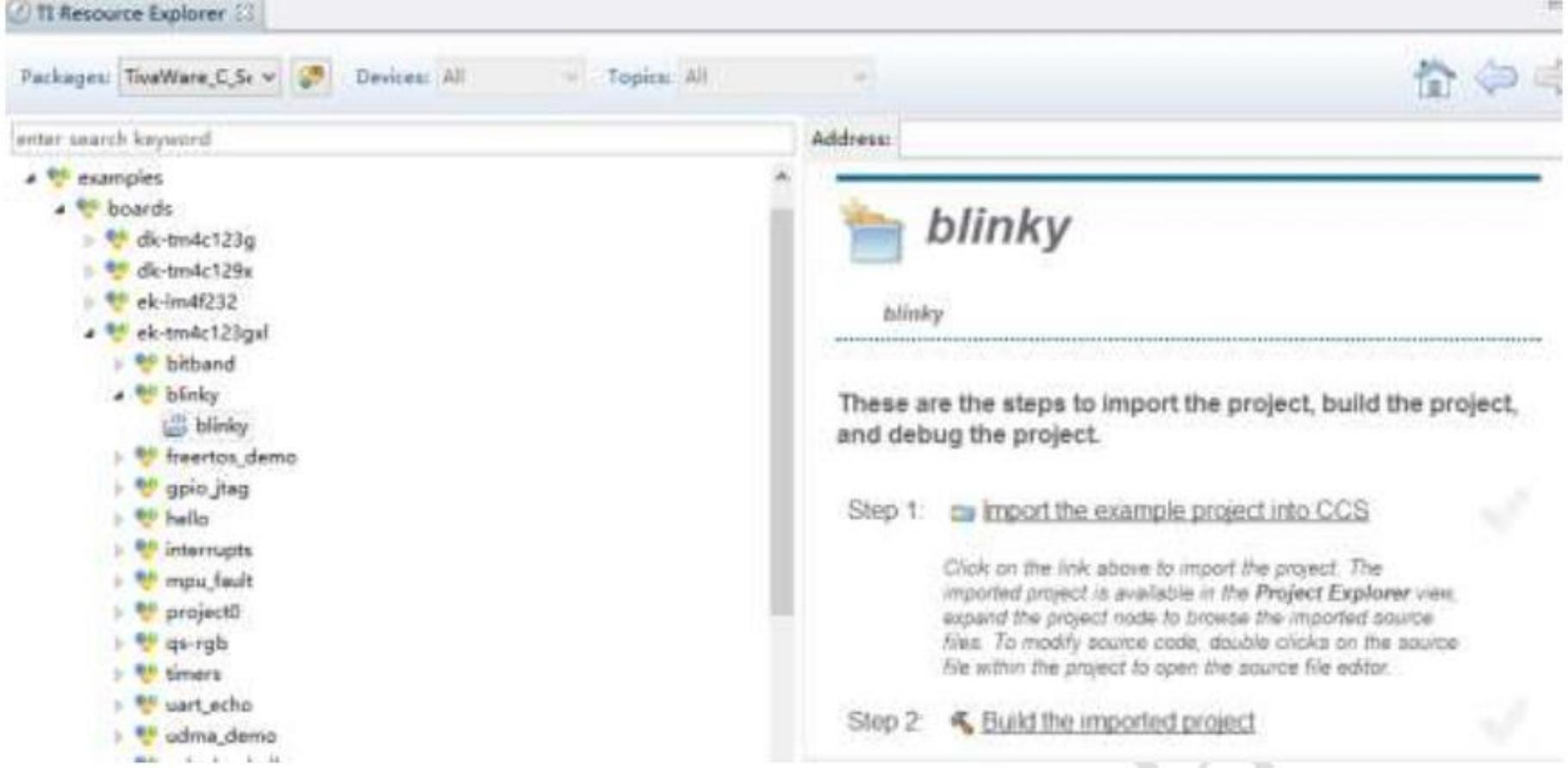


图 5.7 导入工程

4. 也可不经选择 **import the example project into CCS**，即可完成工程的导入，此时，

在 CCS 界面 **project explorer** 中可以看到 **blinky 工程** 已经存在，如图 5.8。

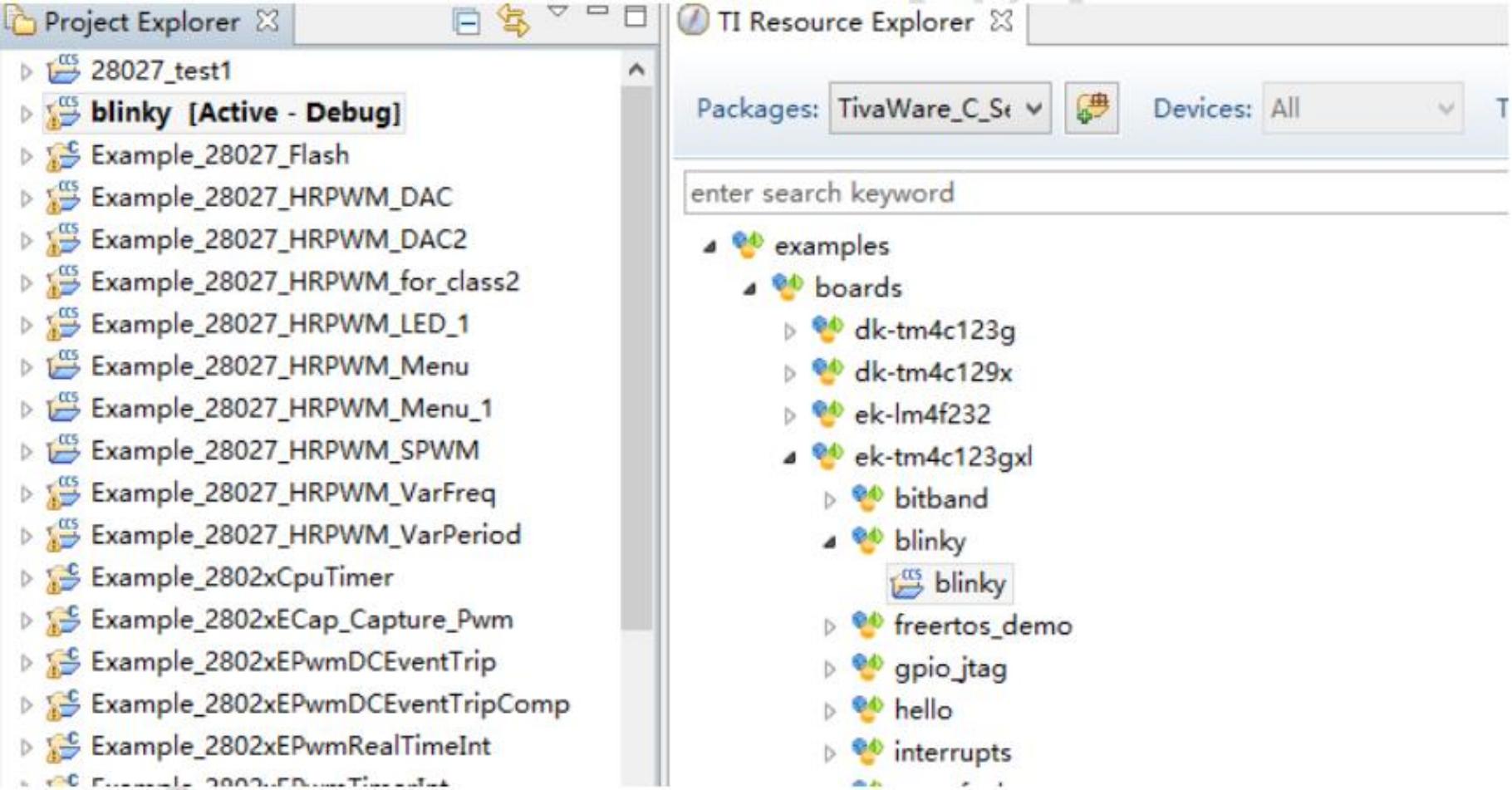


图 5.8 导入工程

5. 也可不经过 TI Resource Explorer， 直接导入已存在的工程。 在 CCS 界面选择菜单项 **Projects**， 选择 **Import CCS Projects…**， 选择搜索文件夹， 通过浏览按钮找到 C:\ti\TivaWare C Series-2.1.0.12573\**examples\boards\ek-tm4c123gxl\blinky 目录**， 点 击完成搜索， 即可将 **blinky** 工程导入 **Project Explorer** 中**。**

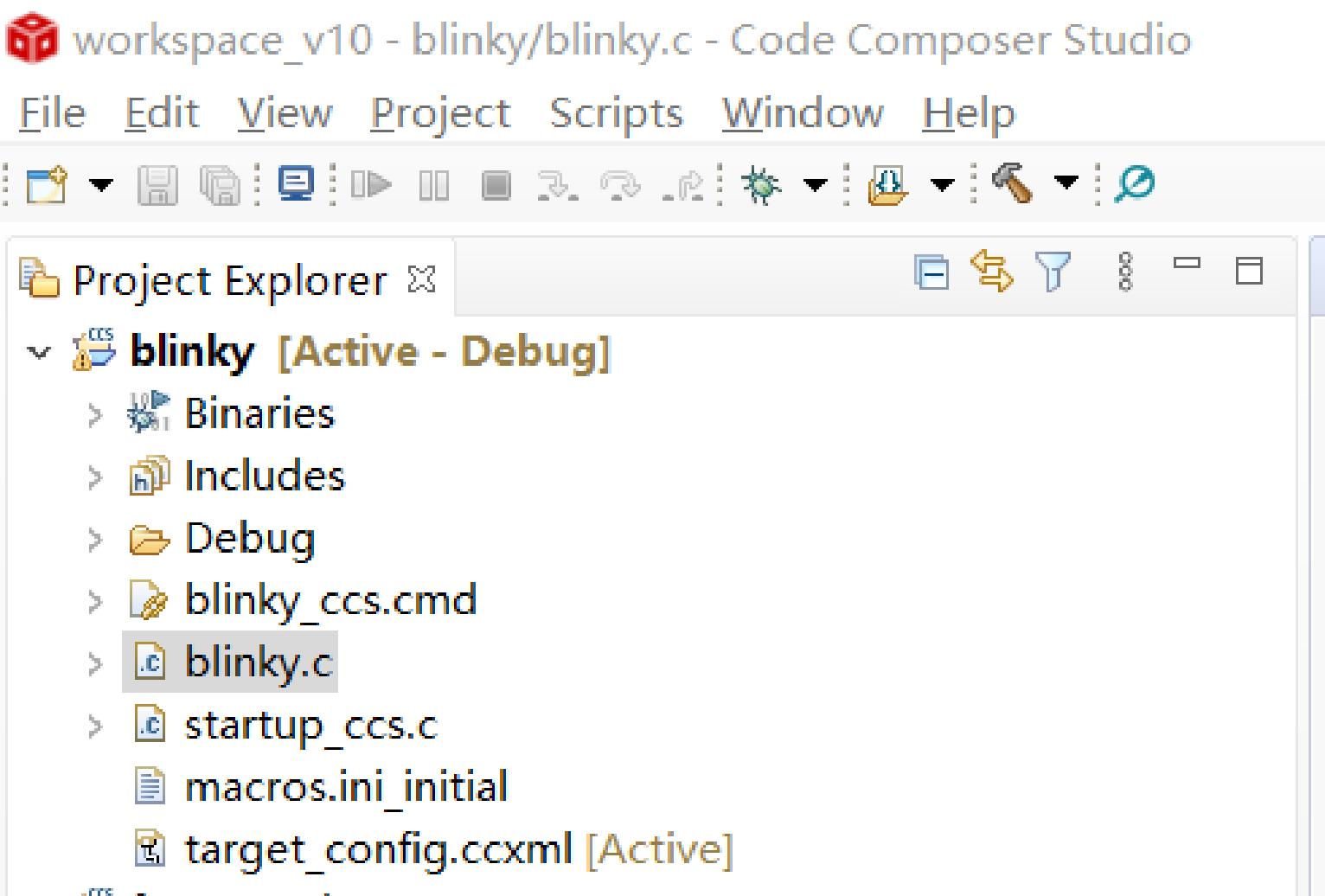


图 5.9 导入 **blinky** 工程

**5.3 运行导入的示例工程**

这里我们仍以导入的工程 **blinky** 为例， 流程如下：

1. 选择 **blinky 工程**， 使其处于 **active-debug** 状态，如图 5.10；

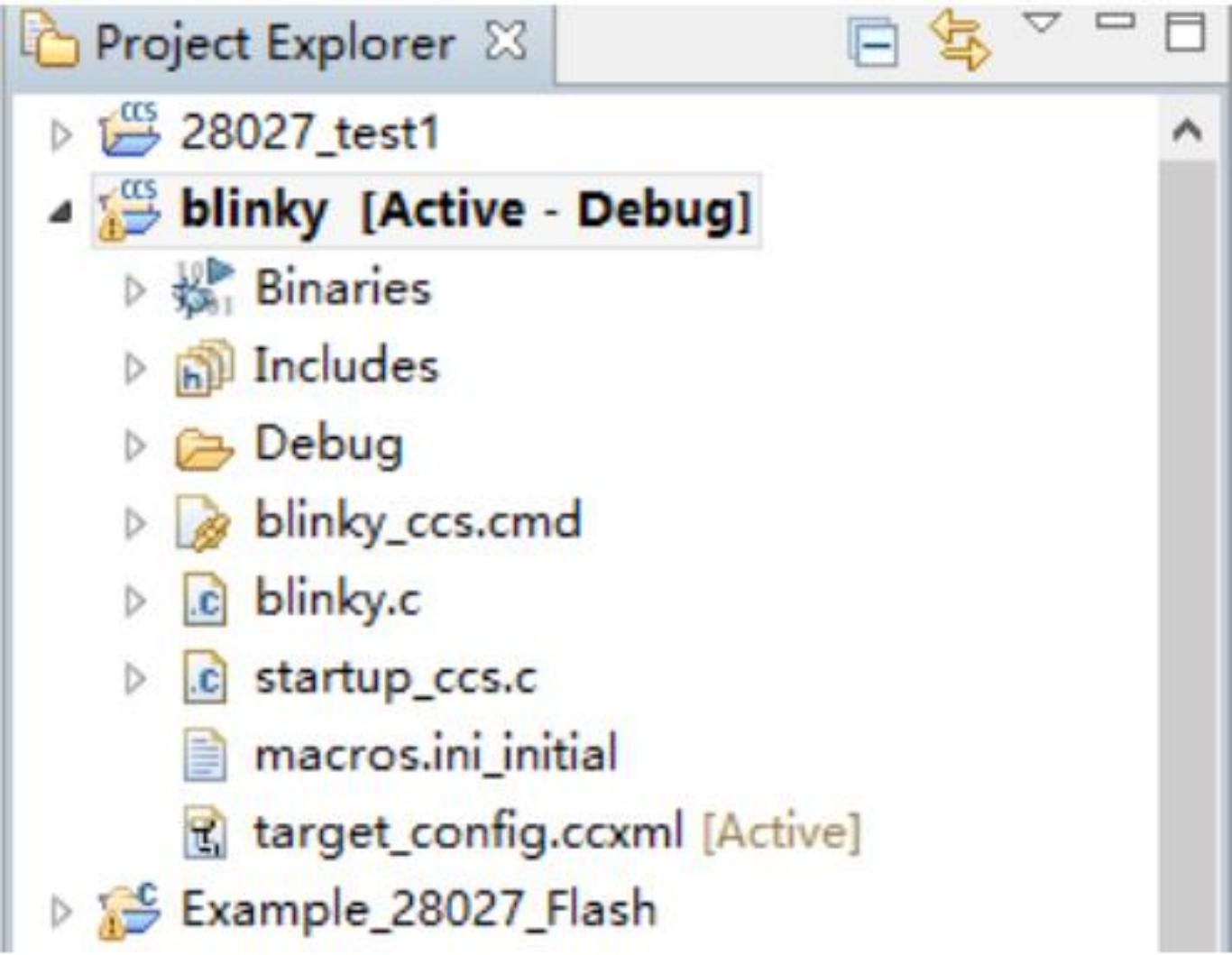


图 5.10 运行 blinky 工程

2. 选择工具栏 **debug** 按钮，程序即可烧入开发板，此时，程序尚未运行，程序指向 **main** 中第一行语句， 如图 5.11；

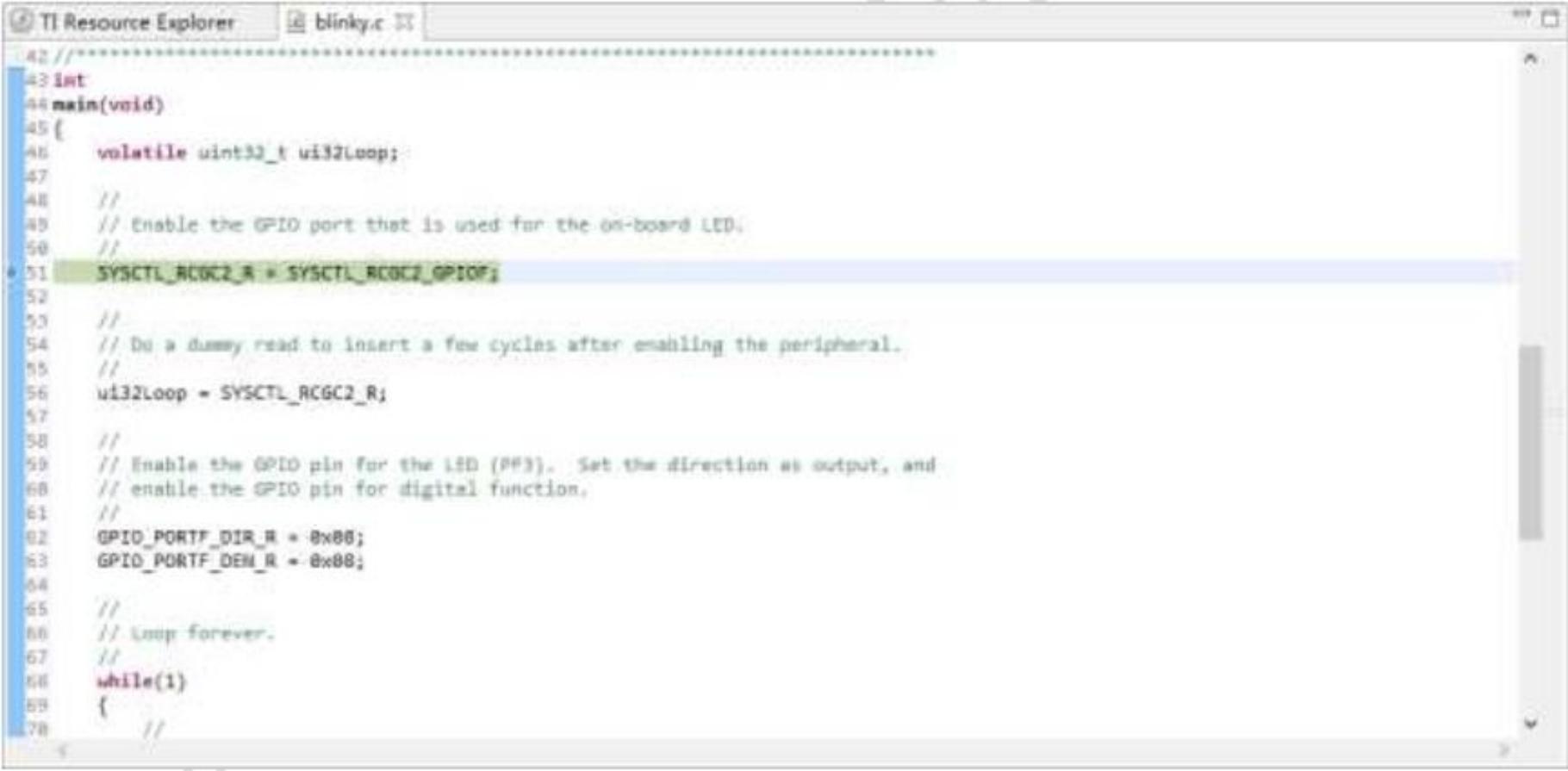


图 5.11 运行 blinky 工程

3. 选择 **debug** 下的第一个按钮，即 **运行**，程序即可正常运行；

4. 此时可以观察到开发板 **D1 灯快速闪烁**，如图 5.12。

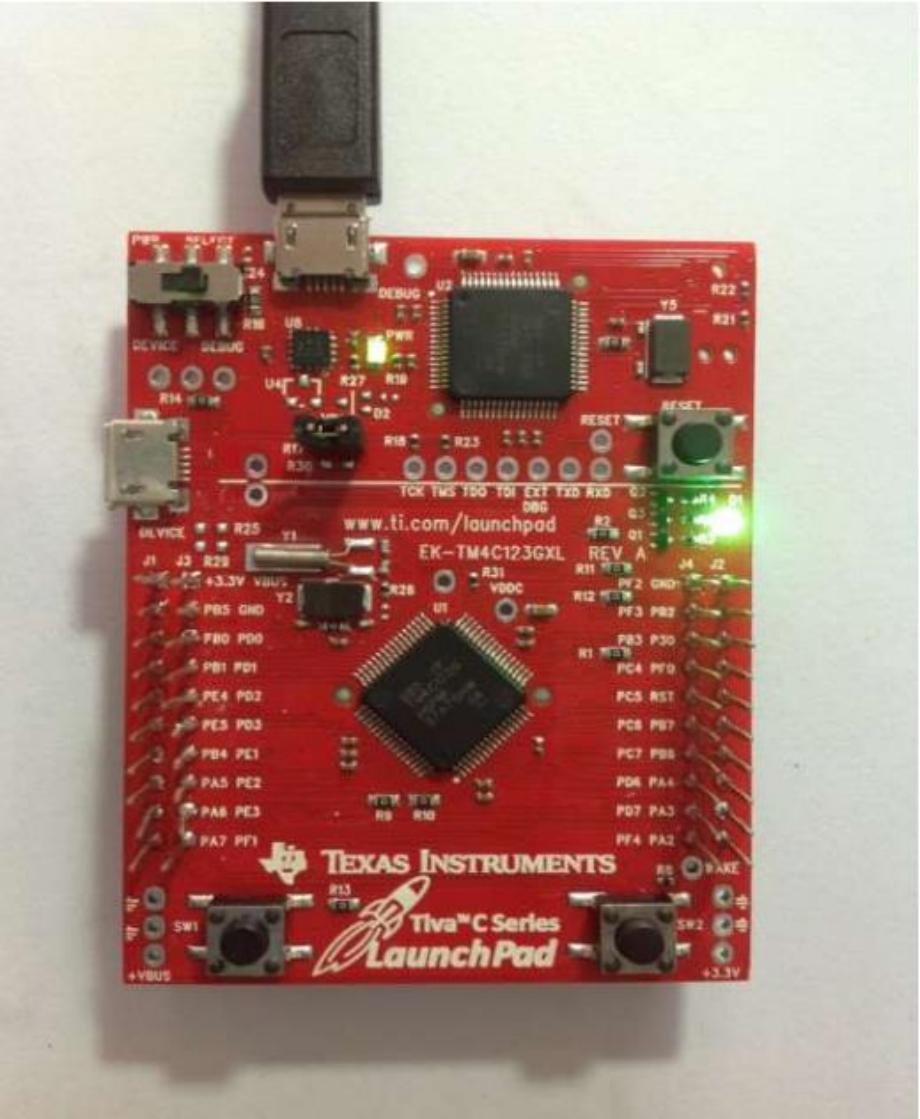


图 5.12 运行 blinky 工程

**实验项目**

实验 1 环境建立与 GPIO 实验

**1.1 实验目的**

1、 掌握 GPIO （通用输入输出）引脚的基本配置方法。

2、 学会使用 GPIO 控制 LED 灯的闪烁和按键输入。

3、 熟悉 Tivaware 程序库的使用， 简化 GPIO 操作。

**1.2 实验环境**

1、 Tiva C LaunchPad实验卡。

2、 CCS 集成开发环境。

**1.3 实验内容**

1. **安装 CCS、 TivaWare， 建立开发环境**。

2. **实验一：控制 LED 灯交替闪烁**

o **任务**： 通过 GPIO 控制开发板上的三色 LED 灯交替闪烁。 o **步骤**：

1. 设置 GPIO 端口时钟。

2. 配置端口 F 的引脚 1 和引脚 2 为数字输出模式。

3. 编写程序，使 LED 灯交替闪烁。

3. **实验二：利用按键控制LED 灯**

o **任务**： 通过按键控制 LED 灯的亮灭，长按按键 1 秒 （近似） 点亮 LED 灯， 再 长按 1 秒 （近似） 熄灭 LED 灯。

o **步骤**：

1. 使能 GPIO 端口 F 的时钟。

2. 配置端口 F 的引脚 2 为数字输出模式。

3. 配置端口 F 的引脚 4 为数字输入模式，并设置为上拉输入。

4. 编写程序，根据按键状态控制 LED 灯的亮灭。

**1.4 实验步骤**

1. **硬件连接**：

o 确保开发板与计算机通过 USB 线连接。

o 检查开发板上的 LED 灯和按键是否正常。

2. **软件配置**：

o 打开开发环境（如 CCS）。

o 创建新的工程， 选择 TM4C123GH6PM 芯片。

o 添加必要的头文件和库文件（如 tm4c123gh6pm.h 和 Tivaware 库）。

3. **代码编写**：

o **实验一代码**：

#include <stdint.h>

#include "inc/tm4c123gh6pm.h"

int main(void) {

volatile uint32\_t ui32Loop; // 1. 设置时钟

// 2. 配置端口 F 的引脚 1 和引脚 2 为数字输出模式

// 3. 编写程序， 使 led 红蓝闪烁

o **实验二代码**：

#include <stdint.h>

#include "inc/tm4c123gh6pm.h"

int main(void) {

**volatile** uint32\_t ui32Loop; uint8\_t ui8flag = 0;

// 使能GPIO端口F的时钟

// 配置PF2为输出（连接到LED）

// 配置PF4为输入（连接到按钮）

//编写程序， 完成按键控制led的亮灭任务

}

4. **调试与运行**：

o 编译程序，确保无错误。

o 下载程序到开发板。

o 观察 LED 灯的闪烁情况和按键控制效果。

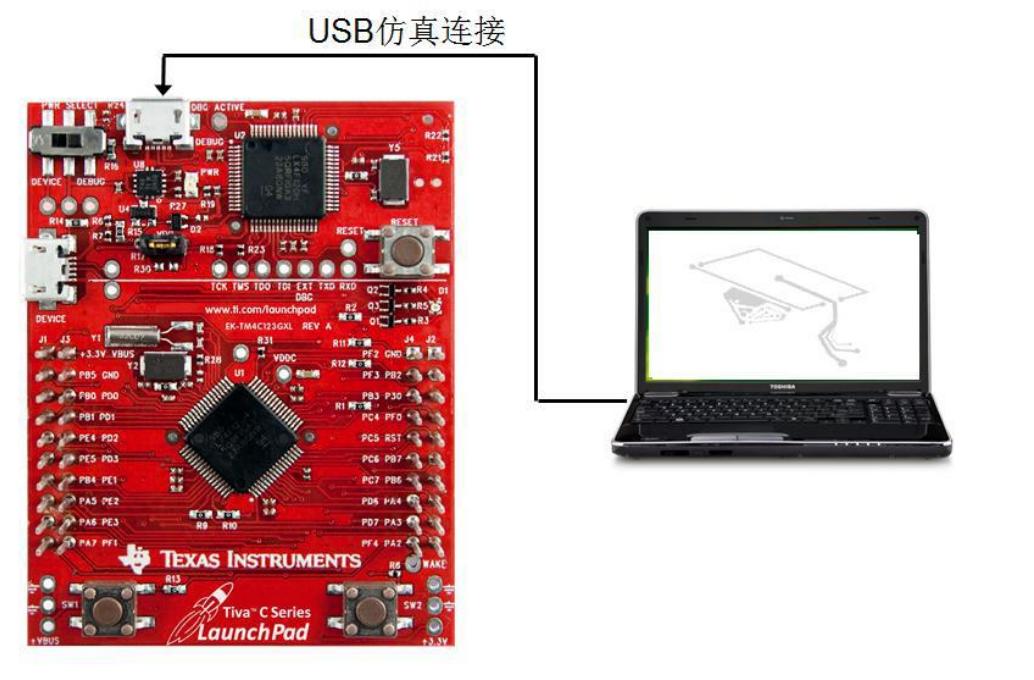


图 目标板与 PC 机的连接

**1.5 实验要求**

1. 完成实验一和实验二的代码编写和调试。

2. 记录实验过程中的问题及解决方法。

3. 撰写实验报告， 总结实验内容和结果。

实验 2 I2C 数据传送设计

**2.1 实验目的**

1. 掌握 I2C 总线的基本原理和通信协议。

2. 学会使用嵌入式开发环境（如 TivaWare 和 CCS）进行 I2C 通信的编程。

3. 通过 I2C 回送实验验证主从设备之间的数据传输功能。

**2.2 实验环境**

1. Tiva LaunchPad 开发板

2. 计算机（安装有 Code Composer Studio v5 及以上版本）

3. 串口调试助手软件

4. USB 数据线

**2.3 实验内容**

I2C （Inter-Integrated Circuit） 总线是一种用于连接低速设备的串行通信协议，具有简单、可 靠、成本低等特点。在本实验中， I2C主机模块与从机模块通过SDL （数据线） 和SCL （时钟 线） 连接， 主机发送数据给从机， 从机再将接收到的数据回送给主机， 以验证数据传输的正 确性。

**2.4 实验步骤**

**1. I2C初始化及配置**

1. 使能I2C时钟。

2. 使能相应的GPIO模块的时钟。

3. 启用相应GPIO口的复用功能。

4. 配置I2CSDA引脚为开漏模式。

5. 为相应的引脚配置为I2C信号接口。

6. 确定I2C模块工作在主机模式还是从机模式，以及是否进入内部回送模式。

7. 设置所需的SCL时钟速率。

**2. I2C主机与从机模式配置**

• **主机模式**： 将要与之通信的从机地址写入I2CMSA寄存器， 并通过该寄存器配置主机

是发送还是接收模式。

• **从机模式**：写入本身从机地址到I2CSOAR寄存器。若从机为发送模式， 则写入数据

到I2CSDR寄存器；若从机为接收模式， 则从I2CSDR寄存器读取接收到的数据。

**3. 编写I2C回送实验程序**

1. 初始化I2C主机和从机模块。

2. 配置UART用于调试信息输出。

3. 实现主机向从机发送数据，并接收从机回送的数据。

4. 通过串口助手工具显示主从机的发送和接收数据， 验证数据传输是否正确。

**4. 运行实验程序**

1. 将开发板通过USB线连接到计算机。

2. 在CCS中编译并下载程序到开发板。

3. 使用串口助手工具连接到开发板的UART外设， 设置波特率为115200。

4. 观察串口助手的输出， 验证I2C回送操作是否成功。

**5. 以下是I2C回送实验的头文件：**

#include <stdbool.h>

#include <stdint.h>

#include "inc/hw\_i2c.h"

#include "inc/hw\_memmap .h" #include "inc/hw\_types.h"

#include "driverlib/gpio.h"

#include "driverlib/i2c.h"

#include "driverlib/pin\_map .h" #include "driverlib/sysctl.h"

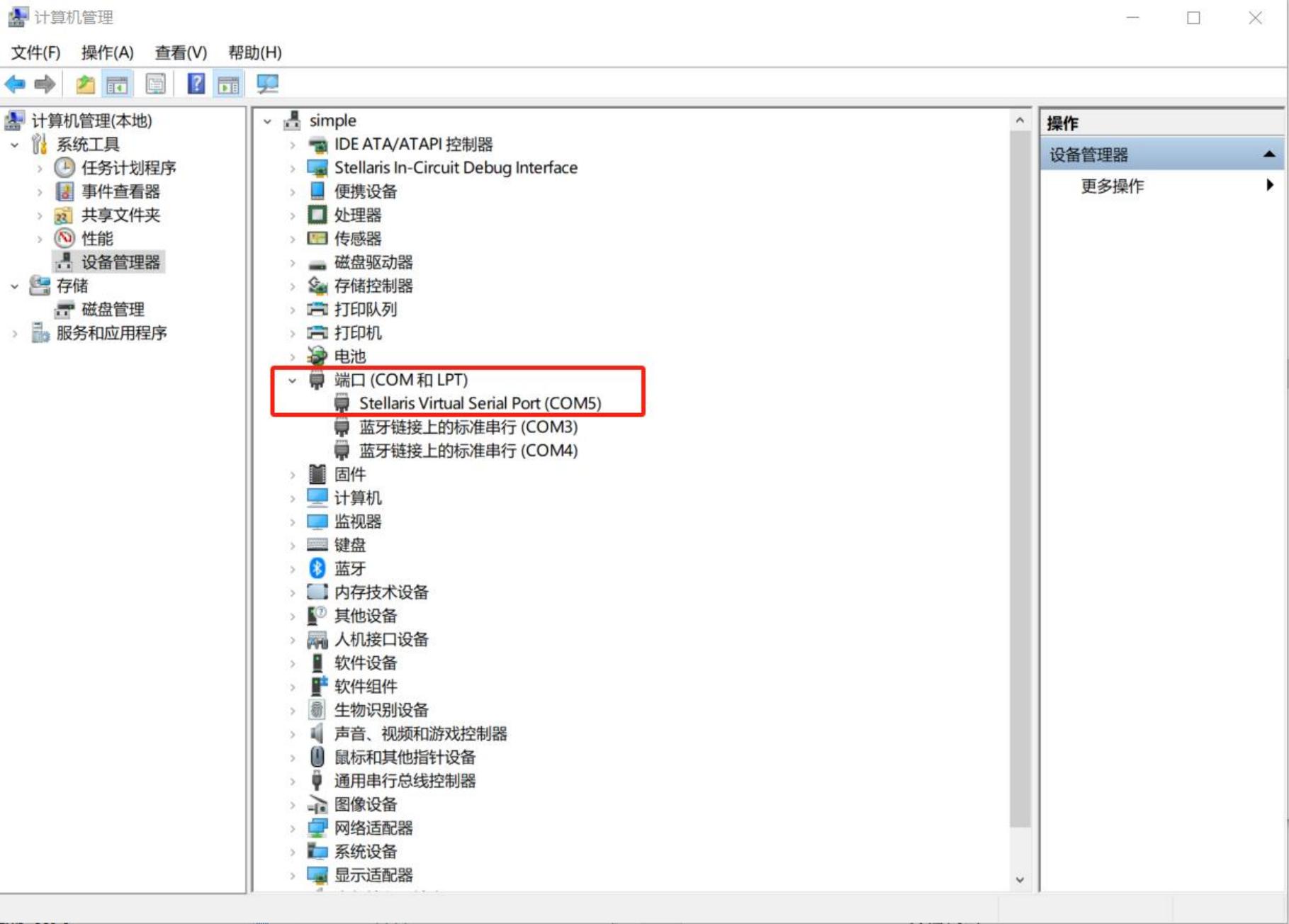
#include "driverlib/uart.h" #include "utils/uartstdio.h"

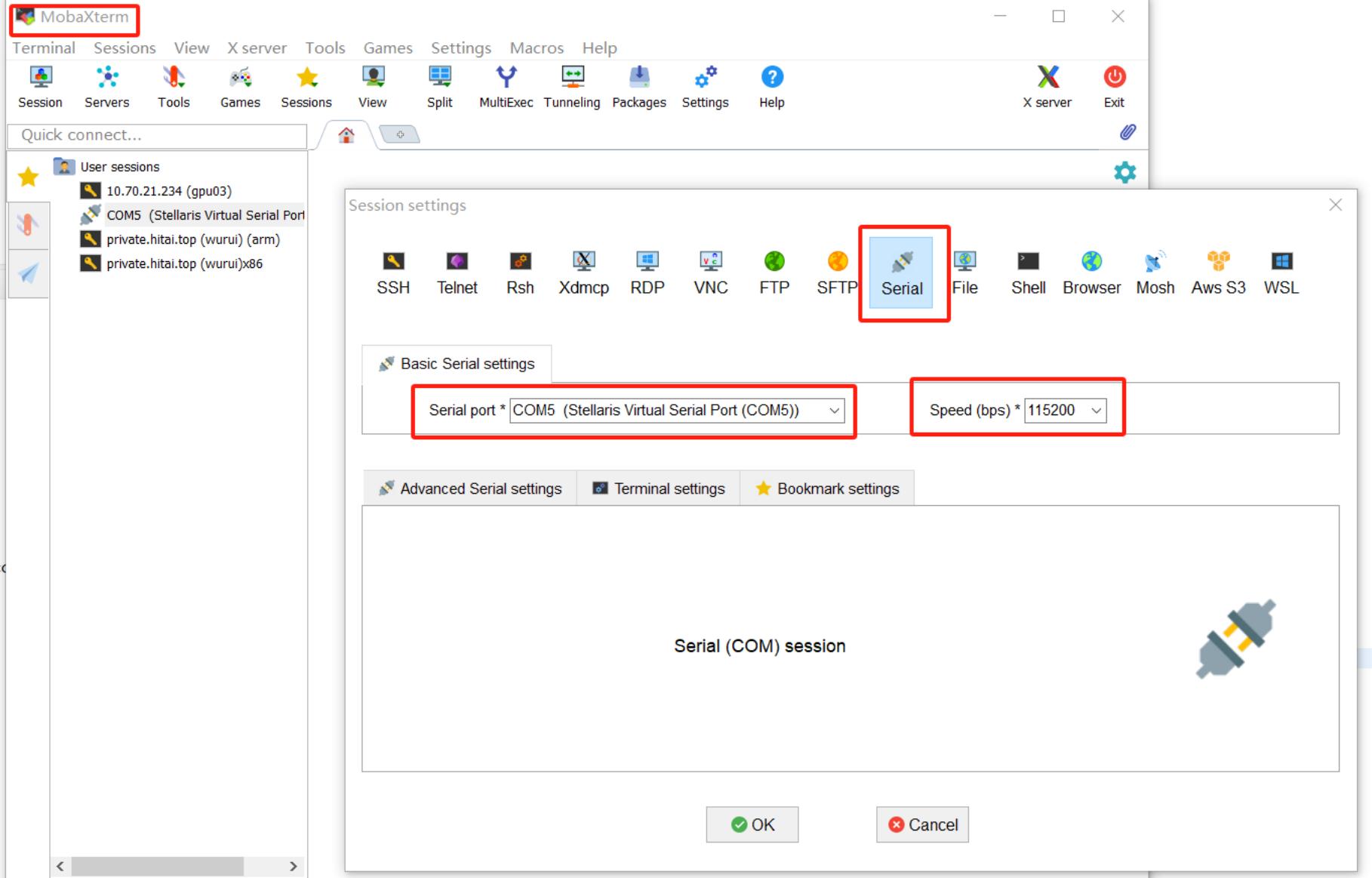
**6. 串口助手设置**

（1） 首先在设备管理器中查看系统连接实验板的虚拟串口号， 如COM5；

（2） 打开串口助手， 如MobaXterm， 新建Session， 选择串口serial， 配置串口参数： 串口号 为COM5， 波特率为115200。

（3） 点OK， 即可进入串口通信状态， 等待串口信息。





**2.5 实验要求**

1. **硬件连接**：将 Tiva LaunchPad 开发板通过 USB 线连接到计算机， 并确保设备管理 器中识别到开发板。

2. **软件配置**：在 CCS 中创建一个新的工程，并导入 TivaWare 库。

3. **代码编写**： 根据实验指导书中的步骤， 编写 I2C 回送实验程序。

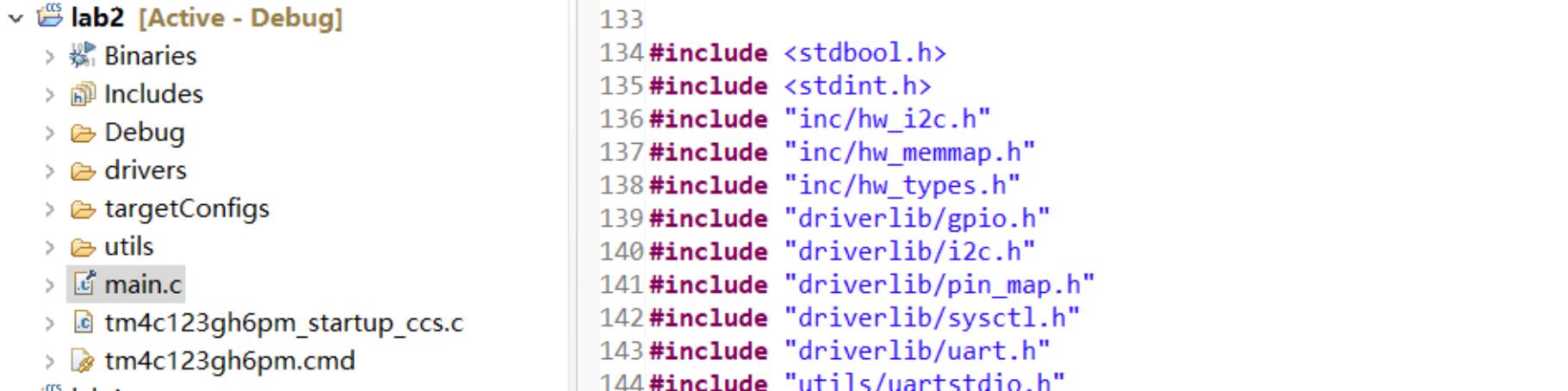
4. **程序下载**： 将编译好的程序烧录到开发板中。

5. **数据传输测试**： 使用串口助手工具连接到开发板的 UART 外设，观察数据传输过 程。

6. **撰写实验报告**，总结实验内容和结果。

**2.6 实验提示**

参考 C:\ti\TivaWare C Series-2.1.0.12573\examples\peripherals\i2c 目录下的示例代码。 新建 CCS 工程后， 参考示例代码完成主程序编码， 继续构建相关代码， 完善 CCS 工程文件。



实验 3 RTOS 的移植与应用

**3.1 实验目的**

1. 掌握 FreeRTOS 的基本特性及其移植方法。

2. 学会将 FreeRTOS 移植到 EK-TM4C123GXL 开发板。

3. 理解 RTOS 中任务调度与定时器中断的协同工作。

4. 测量中断响应时间， 评估 FreeRTOS 对中断的处理性能。

**3.2 实验环境**

1. 嵌入式开发板（如 EK-TM4C123GXL，基于 ARM Cortex-M4F 处理器）。

2. 计算机 （安装有 Code Composer Studio v9 及以上版本）。

3. USB 数据线。

4. 串口调试助手软件。

**3.3 实验内容**

FreeRTOS 是一个轻量级的实时操作系统内核，广泛应用于嵌入式系统开发。它具有以 下特点：

• **免费开源**： 提供完整的源代码，可免费用于商业产品。

• **高效简洁**：内核仅包含 3 个 C 源文件，典型占用 6KB ROM 和 12KB RAM。

• **高实时性**： 支持多任务调度、信号量、队列等实时功能。

• **可移植性**： 易于移植到各种微控制器平台。 移植 FreeRTOS 到目标平台主要涉及以下文件：

• **portmacro.h**：数据类型重定义和与堆栈、时间片、中断相关的宏定义。

• **port.c**：实现 portmacro.h 中的函数，涉及底层寄存器配置。

• **portasm.s** （可选）：用汇编语言实现的函数。

• **FreeRTOSConfig.h**：全局配置文件，用于定义系统参数。

（1）将 FreeRTOS 移植到 EK-TM4C123GXL 开发板， 分别建立按键检测和 LED 显 示任务， 基于 FreeRTOS 进行多任务调度， 分析任务调度效率， 观察实验现象。

（2） 配置不同的中断源， 如 GPIO 中断、定时器中断等， 测量从中断发生到相应 任务被调度执行的时间，评估 FreeRTOS 对中断的处理性能。

**3.4 实验步骤**

**1. 环境准备**

1. **硬件连接**：

o 将 EK-TM4C123GXL 开发板通过 USB 线连接到计算机。

o 确保开发板在设备管理器中被正确识别。

2. **软件安装**：

o 安装 Code Composer Studio （CCS）。

o 安装 TivaWare 库（包含 FreeRTOS 移植示例）。

o 安装串口调试助手软件。

**2. FreeRTOS 移植**

1. **创建工程**：

o 打开 CCS， 创建一个新的工程， 目标设备选择 EK-TM4C123GXL。

o 选择“Empty Project”并添加 main.c 文件。

2. **导入 FreeRTOS 库**：

o 将 TivaWare 中的 FreeRTOS 目录（如

C:\ti\TivaWare\_C\_Series-x.x.x.x\examples\boards\ek-tm4c1 23gxl\freertos\_demo）复制到工程目录。

o 在 CCS 中将相关文件（如 FreeRTOSConfig.h、 portmacro.h、 port.c 等）添加到工程中。

3. **配置 FreeRTOS**：

o 打开 FreeRTOSConfig.h 文件，根据目标平台配置系统参数，如任务优先 级、 堆栈大小等。

o 修改 portmacro.h 和 port.c 文件， 以适配目标微控制器的硬件特性。

**3. 编写内容 1 任务代码 （也可采用 freertos\_demo 中的参考例程）**

**（1） 创建任务**：

o 在 main.c 中编写两个任务： LEDTask 和 SwitchTask。

o LEDTask 任务控制开发板上的 LED 灯闪烁。

o SwitchTask 任务检测按键输入， 并切换 LED 闪烁的延时。

void LEDTask(void\* pvParameters) { while (1) {

// 点亮 LED

GPIOPinWrite(GPIO\_PORTF\_BASE, GPIO\_PIN\_2, GPIO\_PIN\_2); vTaskDelay(pdMS\_TO\_TICKS(500)); // 延时 500ms

// 熄灭 LED

GPIOPinWrite(GPIO\_PORTF\_BASE, GPIO\_PIN\_2, 0); vTaskDelay(pdMS\_TO\_TICKS(500)); // 延时 500ms

} }

void SwitchTask(void\* pvParameters) { uint32\_t delay = 500;

while (1) {

if (GPIOPinRead(GPIO\_PORTF\_BASE, GPIO\_PIN\_4) == 0) { delay += 500; // 按键按下，增加延时

}

vTaskDelay(pdMS\_TO\_TICKS(delay)); }

}

**（2） 初始化任务并启动调度器**：

int main(void) {

// 系统初始化

SysCtlClockSet(SYSCTL\_SYSDIV\_5 | SYSCTL\_USE\_PLL | SYSCTL\_XTAL\_ 16MHZ |

SYSCTL\_OSC\_MAIN);

SysCtlPeripheralEnable(SYSCTL\_PERIPH\_GPIOF);

GPIOPinTypeGPIOOutput(GPIO\_PORTF\_BASE, GPIO\_PIN\_2); GPIOPinTypeGPIOInput(GPIO\_PORTF\_BASE, GPIO\_PIN\_4);

GPIOPadConfigSet(GPIO\_PORTF\_BASE, GPIO\_PIN\_4, GPIO\_STRENGTH\_2MA, GPIO\_PIN\_TYPE\_STD\_WPU);

// 创建任务

xTaskCreate(LEDTask, "LEDTask", configMINIMAL\_STACK\_SIZE, NULL,

tskIDLE\_PRIORITY + 1, NULL);

xTaskCreate(SwitchTask, "SwitchTask", configMINIMAL\_STACK\_SIZE, NULL,

tskIDLE\_PRIORITY + 1, NULL);

// 启动调度器

vTaskStartScheduler();

// 不会执行到这里 return 0;

}

**4. 内容 2： 中断性能测试**

使用 GPIO 引脚作为外部中断源， 并通过定时器记录时间戳来测量中断响应时间。 具体地， 使用 一个 GPIO 引脚（如 PF0）作为外部中断源。 使用另一个 GPIO 引脚（如 PF1）作为输出信号， 用于记录中断服务函数的执行时间。 实验步骤：

**（1） 硬件连接：**

- 将一个按键连接到 PF0 引脚，用于触发中断。

- 将一个 LED 连接到 PF1 引脚， 用于指示中断服务函数的执行。

**（2） 参考代码**

#include <stdint.h> #include <stdbool.h>

#include "inc/hw\_types.h"

#include "inc/hw\_memmap.h" #include "driverlib/sysctl.h"

#include "driverlib/gpio.h"

#include "driverlib/pin\_map.h" #include "driverlib/timer.h"

#include "FreeRTOS.h"

#include "task.h"

#include "queue.h" #include "semphr.h"

volatile uint32\_t g\_ui32TimerValue = 0;

// 定时器中断处理函数

void TimerIntHandler(void) { // 清除定时器中断标志

TimerIntClear(TIMER0\_BASE, TIMER\_TIMA\_TIMEOUT); // 更新定时器值

g\_ui32TimerValue++; }

// 初始化定时器

void TimerInit(void) {

// 使能定时器 0 的时钟

SysCtlPeripheralEnable(SYSCTL\_PERIPH\_TIMER0);

// 配置定时器 0 为 32 位周期模式，计数频率为 1MHz

TimerConfigure(TIMER0\_BASE, TIMER\_CFG\_32\_BIT\_PER);

TimerLoadSet(TIMER0\_BASE, TIMER\_A, SysCtlClockGet() / 1000000 - 1);

// 使能定时器中断

IntEnable(INT\_TIMER0A);

// 使能定时器 A 超时中断

TimerIntEnable(TIMER0\_BASE, TIMER\_TIMA\_TIMEOUT);

// 注册定时器中断处理函数

TimerIntRegister(TIMER0\_BASE, TIMER\_A, TimerIntHandler); // 启用定时器

TimerEnable(TIMER0\_BASE, TIMER\_A); }

// 中断服务函数

void GPIOPortFIntHandler(void) { // 清除中断标志

GPIOIntClear(GPIO\_PORTF\_BASE, GPIO\_INT\_PIN\_0);

// 记录中断触发时间

uint32\_tui32InterruptTime = g\_ui32TimerValue;

// 记录中断服务函数入口时间

uint32\_tui32EntryTime = g\_ui32TimerValue;

// 执行中断服务函数逻辑 // 例如： 点亮 LED

GPIOPinWrite(GPIO\_PORTF\_BASE, GPIO\_PIN\_ 1, GPIO\_PIN\_ 1);

// 记录中断服务函数出口时间

uint32\_tui32ExitTime = g\_ui32TimerValue;

// 计算中断响应时间（从触发到入口）

uint32\_tui32ResponseTime = ui32EntryTime - ui32InterruptTime;

// 计算中断服务函数执行时间

uint32\_tui32ExecutionTime = ui32ExitTime - ui32EntryTime;

// 打印结果（可以使用串口或其他调试工具）

// printf("Interrupt Response Time: %dus\n", ui32ResponseTime); // printf("ISR Execution Time: %dus\n", ui32ExecutionTime);

// 熄灭 LED

GPIOPinWrite(GPIO\_PORTF\_BASE, GPIO\_PIN\_ 1, 0); }

// 初始化 GPIO 中断

void GPIOInit(void) {

// 使能 GPIOF 的时钟

SysCtlPeripheralEnable(SYSCTL\_PERIPH\_GPIOF);

// 等待时钟稳定

while (!SysCtlPeripheralReady(SYSCTL\_PERIPH\_GPIOF)) {}

// 配置 PF0 为输入， PF1 为输出

GPIOPinTypeGPIOInput(GPIO\_PORTF\_BASE, GPIO\_PIN\_0); GPIOPinTypeGPIOOutput(GPIO\_PORTF\_BASE, GPIO\_PIN\_1);

// 配置 PF0 为中断源

GPIOIntTypeSet(GPIO\_PORTF\_BASE, GPIO\_PIN\_0, GPIO\_FALLING\_EDGE);

// 注册中断处理函数

GPIOIntRegister(GPIO\_PORTF\_BASE, GPIOPortFIntHandler);

// 启用 PF0 的中断

GPIOIntEnable(GPIO\_PORTF\_BASE, GPIO\_INT\_PIN\_0); // 启用中断

IntMasterEnable(); }

int main(void) {

// 初始化系统时钟

SysCtlClockSet(SYSCTL\_SYSDIV\_4 | SYSCTL\_USE\_PLL | SYSCTL\_XTAL\_ 16MHZ |

SYSCTL\_OSC\_MAIN);

// 初始化定时器 TimerInit();

// 初始化 GPIO GPIOInit();

// 创建任务（如果有其他任务） // xTaskCreate(...);

// 启动调度器

vTaskStartScheduler();

// 如果调度器停止， 进入无限循环 while (1) {

} }

**（3） 按下连接到 PF0 的按键，触发外部中断， 记录时间。**

**（4） 多次触发中断， 取平均值以获得更准确的结果。**

**3.5 实验要求**

**编译与下载**

1. 在 CCS 中编译工程，确保无错误。

2. 将编译好的程序烧录到开发板。

3. 使用串口调试助手连接到开发板的 UART 接口， 波特率设置为 115200。

**观察实验现象**

1. 观察开发板上的 LED 灯闪烁情况。

2. 按下开发板上的按键，观察 LED 闪烁延时的变化。

3. 通过串口助手输出调试信息，验证任务的运行情况。

4. 按照实验报告中的要求， 完成实验内容。

**实验 4 PWM 的生成与分析**

**4.1 实验目的**

1. 掌握 PWM 信号的生成和配置方法。

2. 利用示波器观察 PWM 信号占空比。

3. 通过 PWM 中断方式实现动态调整 PWM 占空比。

**4.2 实验环境**

1. EK-TM4C123GXL 开发板。

2. 计算机（安装有 Code Composer Studio v9 及以上版本）。

3. USB 数据线。

4. 串口调试助手软件 （可选）。

**4.3 实验内容**

（1） 基于 EK-TM4C123GXL 开发板生成指定频率指定占空比的 PWM 信号。

（2） 利用示波器对 PWM 信号进行观察、 验证、 分析。

（3） 通过 PWM 中断方式实现动态调整 PWM 占空比， 完成 PWM 中断实验。

**4.4 实验准备**

1. **硬件连接**：

o 将 EK-TM4C123GXL 开发板通过 USB 线连接到计算机。

o 确保开发板在设备管理器中被正确识别。

2. **软件安装**：

o 安装 Code Composer Studio （CCS）。

o 安装 TivaWare 库（包含 PWM 模块的驱动程序）

**4.5 实验步骤**

**1、 PWM 初始化与配置**

**（1） 使能 PWM 时钟**：

o 设置系统时钟 SysCtlClockSet；

o 通过 SysCtlPeripheralEnable 函数使能 PWM 模块的时钟；

o 设置 PWM 的时钟分频系数（如 PWM\_SYSCLK\_DIV\_64）。

**（2） 配置 GPIO 引脚**：

o 使能相应 GPIO 模块的时钟；

o 将 PWM 信号分配到合适的引脚上；

o 使能引脚的复用功能。

**（3） 设置 PWM 参数**：

o 设置 PWM 的计数方式（如上下计数模式） ;

o 设置 PWM 信号的周期和占空比。 （PWM 信号频率 100Hz， 占空比 25%）

o 使能 PWM 输出和 PWM 发生器的定时器。

**2、 程序主循环体设计**

o 按键 1 按下时 PWM 信号反转;

o 按键 2 按下时 PWM 信号停止输出。

**3、 利用示波器观察、分析 PWM 输出波形 .**

**4、 PWM 中断实验 （可新建主程序）**

利用 EK-TM4C123GXL 开发板上的 PWM 模块在加载装载值时产生中断，设计一个占空比 由 0.1%逐渐变大到 75%的 PWM。 若系统时钟频率为 16MHz， PWM 时钟频率 16MHz， PWM 信 号频率 250Hz。 则 PWM 模块的转载值： PWM 时钟频率/PWM 信号频率 -1 = 63999。

实验步骤为

**（1） PWM 初始化**：（略）

**（2） 配置 PWM 中断**：

o 使能 PWM 中断。

o 注册中断服务程序（ISR）， 在 ISR 中动态调整 PWM 占空比。

**（3） 中断服务程序**：

o 在中断中逐步增加占空比，直到达到最大值后重置。

占空比从 0.1%逐渐增加到 75%， 步长为 01.%， 则相应的匹配值从 63 逐渐增加到 47999， 步长为 64.

**4.6 实验要求**

1. 在 CCS 中编译工程，确保无错误。

2. 将编译好的程序烧录到开发板。

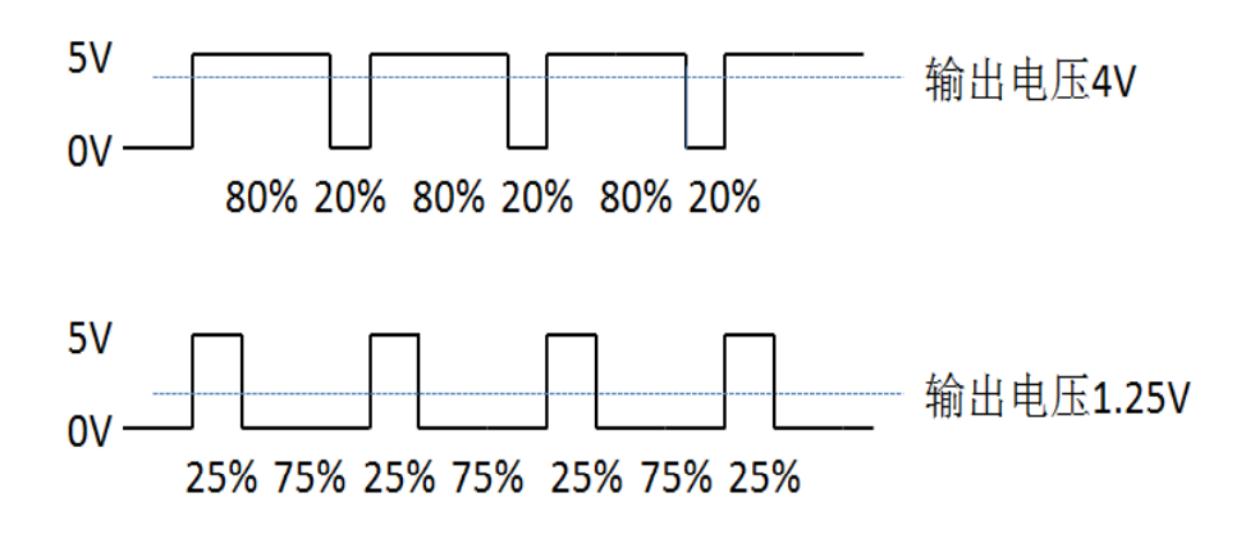
3. 利用示波器观测波形。

4. 观察 PWM 中断实验中占空比的变化。

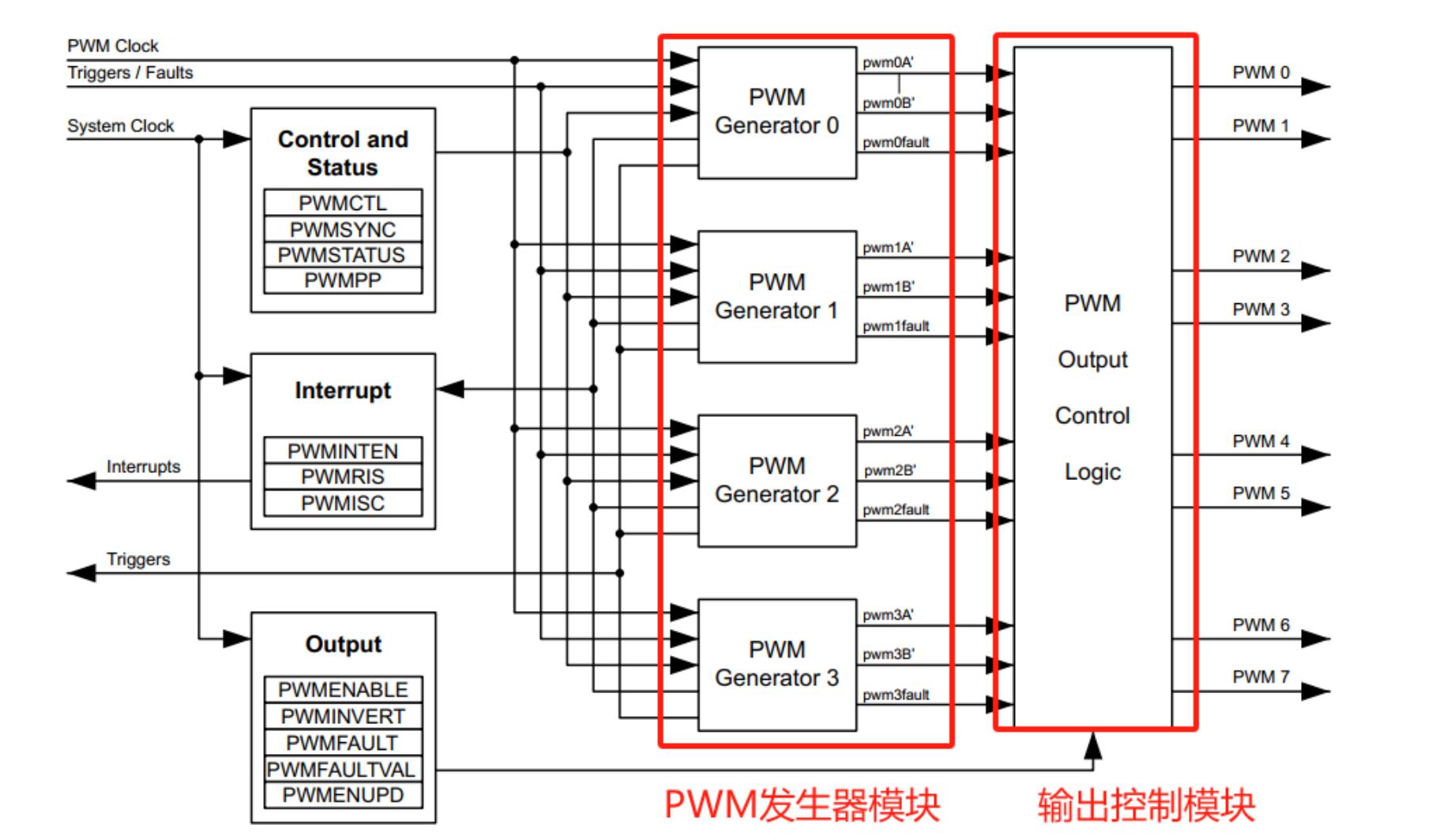
5. 按实验报告中的内容要求逐项完成。

**4.7 实验提示**

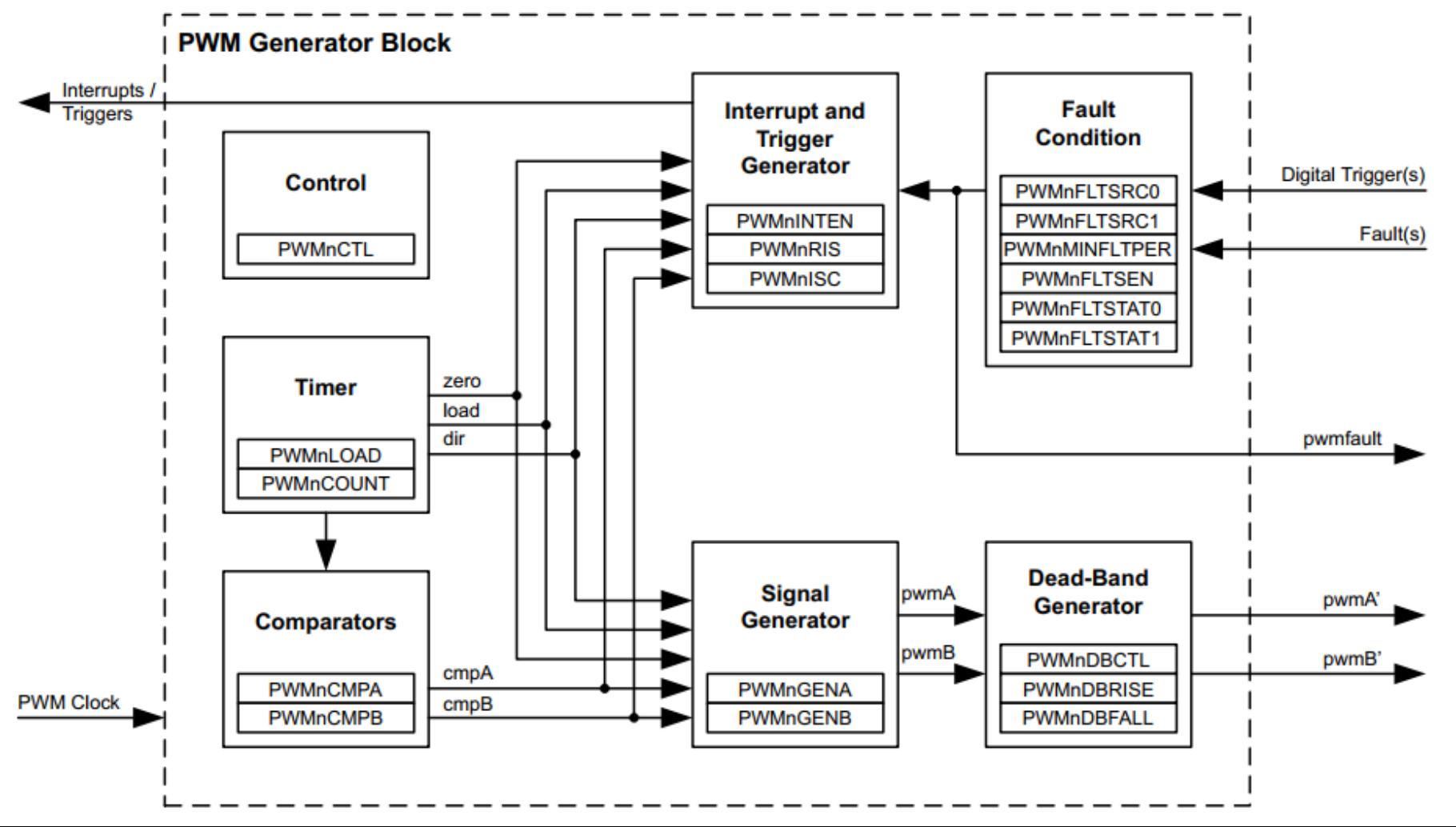
1. PWM 的输出电压大小是通过占空比来控制的。 输出电压=占空比 X 最大电压值



2. TM4C123GH6PM 中的 PWM 模块。 PWM 模块结构图如下



PWM 发生器模块结构图如下



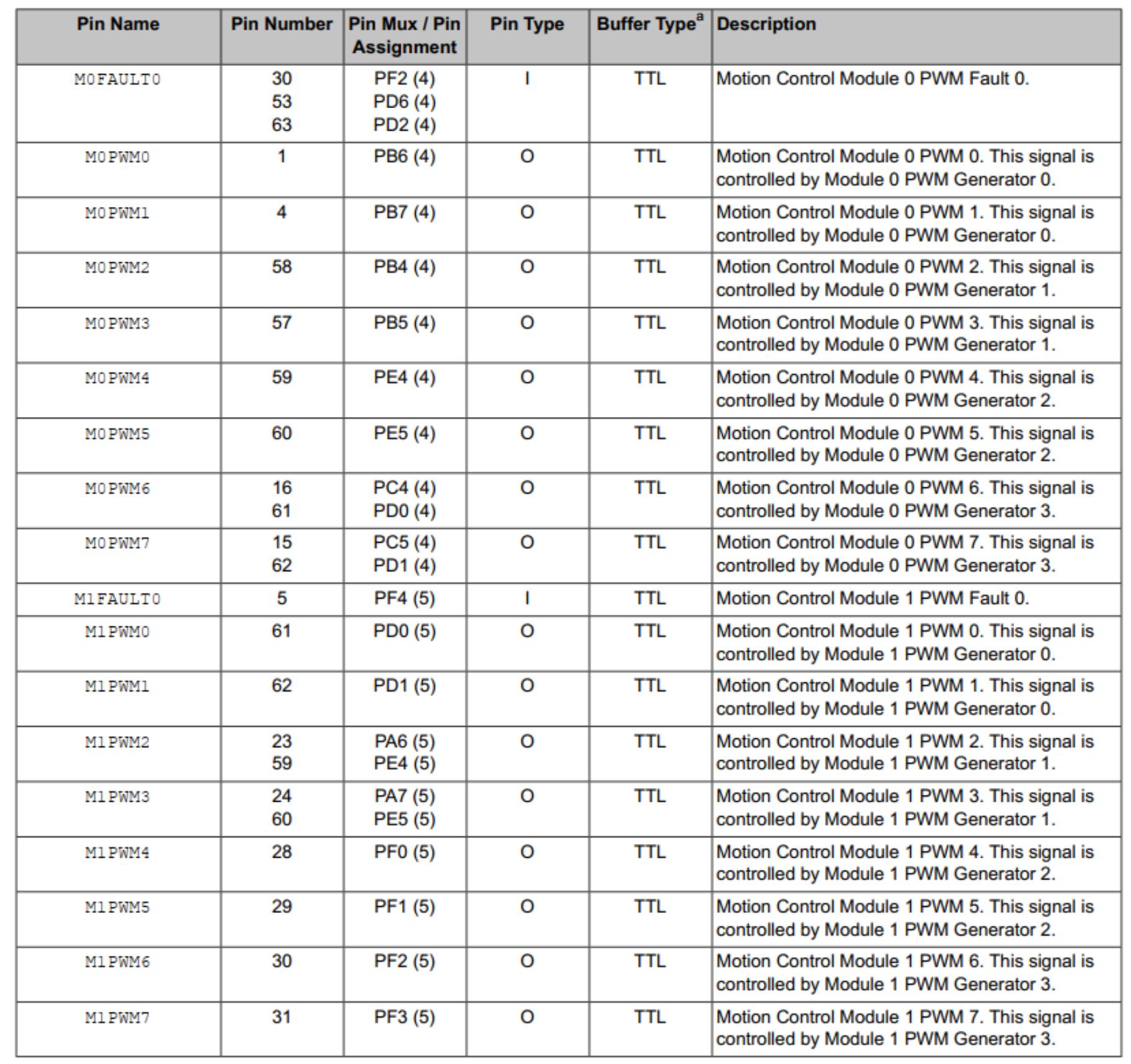
PWM 发生器：

• 可以编程产生 2 路独立的 PWM 信号，或者编程产生一对插入死区延迟的互补信号。

• PWM 信号的频率由装载值控制， 脉宽则由比较器的值决定。

• TM4C123GH6PM 微控制器的 PWM 发生器有系统时钟和预分频系统时钟两个时钟源选择。 • 在使能 PWM 发生器之前， 必须配置好时钟、计数方式、 定时器的装载值、 两个比较器 的值， 以及 PWM 信号受什么事件控制等。

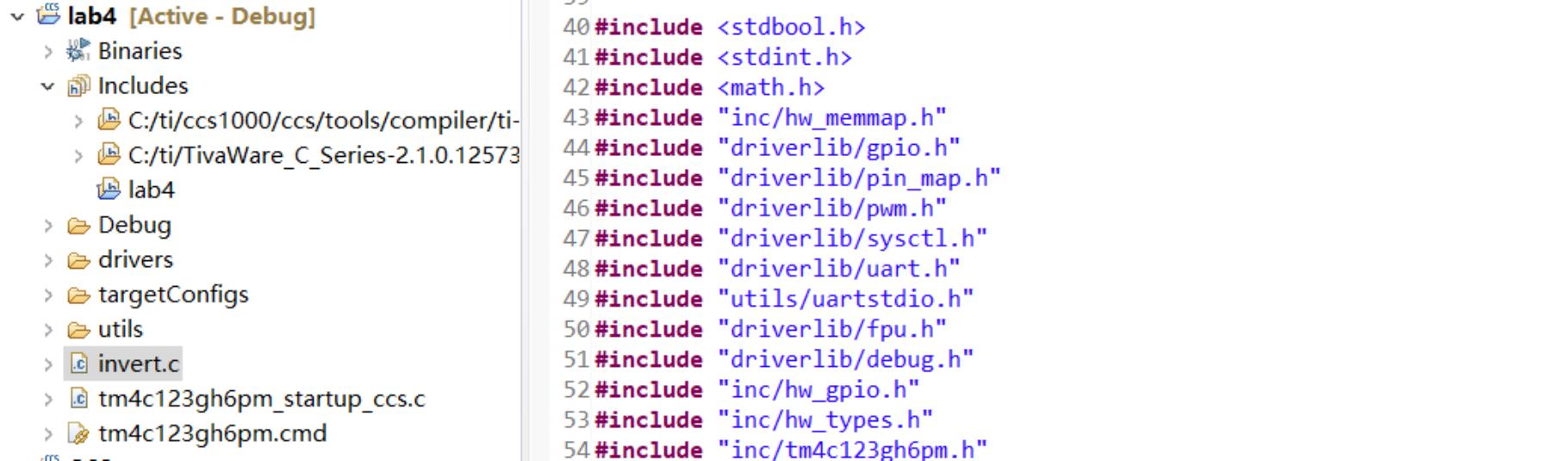
3.根据开发板扩展接口的管脚分布， 选择合适的 PWM 通道， 以实现合适的输出， 方便测试。



4.PWM 定时器工作模式， 可进一步参考 TM4C123GH6PM 的 datasheet。

5、 实验中可新建 CCS 工程后， 可参考

C:\ti\TivaWare C Series-2.1.0.12573\examples\peripherals\pwm 下的例程作为主程序， 继续构建相 关代码， 完善 CCS 工程文件。



实验 5 三色呼吸灯实验

**5.1 实验目的**

1.实现呼吸灯效果，通过 PWM 信号控制 LED 的亮度变化。 2.掌握动态调整 PWM 占空比的方法。

**5.2 实验环境**

1. EK-TM4C123GXL 开发板。

2. 计算机（安装有 Code Composer Studio v9 及以上版本）。

3. USB 数据线。

4. 串口调试助手软件（可选）。

**5.3 实验准备**

1. **硬件连接**：

o 将 EK-TM4C123GXL 开发板通过 USB 线连接到计算机。

o 确保开发板在设备管理器中被正确识别。

2. **软件安装**：

o 安装 Code Composer Studio （CCS）。

o 安装 TivaWare 库（包含 PWM 模块的驱动程序）

**5.4 实验内容**

采用 EK-TM4C123GXL 开发板上的 PWM 模块产生占空比逐渐变化的 PWM 信号， 驱动 3 色 LED 灯。

**5.5 实验提示**

1. **参考实验 4 的基础上完成 PWM 的初始化与配置， 注意 3 色 LED 的 GPIO 配置**；

2. **动态调整占空比**：

o 使用正弦函数 （sinf） 生成占空比的变化曲线。

o 占空比在 0%~100%之间变化， 对应匹配值为： 从 1 到装载值再到 1 变化， 步长由 sin 函数控制， 共 256 阶。

基于 sin 函数的占空比计算

**#define** APP\_PI 3.1415926535897932384626433832795f **#define** STEPS 256

**volatile uint32\_tui32Index = 0;**

**float fAngle = ui32Index \* (2.0f \* APP\_PI / STEPS);**

**uint32\_tui32BlueLevel = (uint32\_t)(xxxx.0f \* (1 + sinf(fAngle)));**

**xxxx** 为装载值的一半， 如： 若装载值为 18740， 则 **xxxx** 为 9370.0f

3. **在主循环中动态更新 PWM 的占空比**。

**5.6 实验要求**

1. 在 CCS 中编译工程，确保无错误。

2. 将编译好的程序烧录到开发板。

3. 使用串口调试助手连接到开发板的 UART 接口， 波特率设置为 115200。（可选）

4. 观察开发板上的 3 色 LED 灯实现呼吸灯效果。

实验 6 综合设计实验

**6.1 实验目的**

1.掌握红外对管、 舵机的使用与控制方法。

2.模拟实现一个具有信号检测、 动作执行、 状态显示及通信交互的控制系统。

**6.2 实验环境**

1. EK-TM4C123GXL 开发板。

2. 计算机（安装有 Code Composer Studio v9 及以上版本）。

3. USB 数据线。

4. 红外对管一个、 舵机一个。

5. 杜邦线 6 根（公对母 3 根， 母对母 3 根）。

6. 串口调试助手软件。

**6.3 实验内容**

模拟一个自动控制系统， 如危险区域自动监控系统。 想定的场景： 当有人或物体靠近 某个禁止区域时， 监控系统控制监控相机转向特定角度实施定向监控， 并发出灯光报 警， 同时将异常信息发送给主控台。 各功能模块具体内容如下：

**事件检测模块**： 通过红外对管检测入侵事件， 对管被遮挡时表示有人靠近。

**相机转向控制模块**： 通过舵机实现相机的转向控制， 包括巡视模式 （0 ° ~180 °来回 巡视） 和定向模式（固定指向 90 °方向）。 舵机正常处于巡视模式， 当有入侵事件 时舵机转入定向模式。

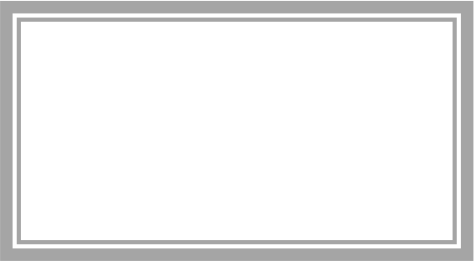
**灯光显示模块**： 当有入侵事件时， 控制 LED 等发出亮度变化 （呼吸灯） 的彩色光表 示报警。

**交互模块**： 通过串口实现主控台（串口调试助手） 与控制系统的交互， 包括工作模式 设定与工作状态发送。 系统工作前， 通过串口对监控系统的启动/关闭等工作模式进 行设定， 系统工作中， 实时将异常事件发送至主控台。 自动控制系统框图如下图所示：

IR

LED

PWM



EK-TM4C123GXL 开发板

UART

主控

（串口助手）

图 6.1 危险区域自动监控系统框图

SG90 （舵机）



GPIO PWM

**6.4 实验要求**

1. 对实验内容进行分析， 对各项功能进行设计， 编写相应的代码。

2. 对开发板资源进行分配， 选择合适的管脚进行连线。

3. 事件检测模块（即红外对管检测） 以中断方式工作。

4. 使用串口调试助手连接到开发板的 UART 接口， 波特率设置为 115200。

5. 观察开发板上的实验效果。

6. 按照实验报告中的内容与要求， 完成实验报告。

**6.5 实验提示**

**1.红外对管**

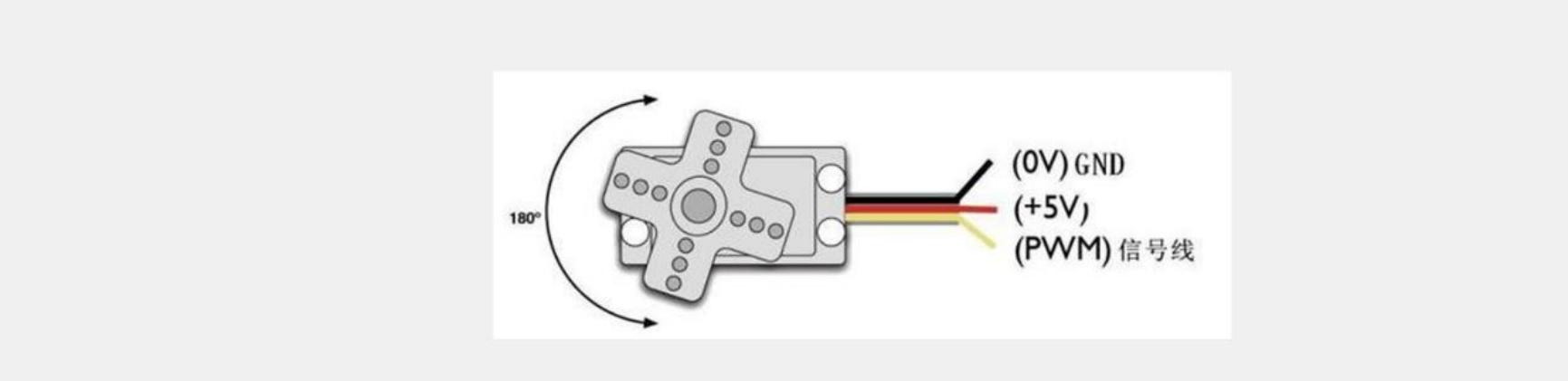
红外对管由红外发射管和红外接收管组成，通常用于检测物体是否遮挡红外光。当有物体遮 挡时， 红外接收管的输出会发生变化。本实验中通过母对母的杜邦线将对管的三个管脚按其名称 分别与 Tiva LaunchPad 的对应管脚相连， 即 VCC 连接到 Tiva LaunchPad 的 3.3V 引脚， GND 连接到 Tiva LaunchPad 的 GND 引脚，IR 引脚（OUT）连接 Tiva LaunchPad 的某个 GPIO 脚（如 PF0）， 通常需要一个上拉电阻连接在 PF0 和 3.3V 之间， 以确保稳定的高电平。

**2、 舵机 SG90**

SG90 舵机是一种常见的微型伺服电机， 广泛应用于机器人、航模、智能家居等领域。 SG90 舵机通过脉冲宽度调制（PWM） 信号进行控制。 PWM 信号的周期为 20ms， 脉冲宽度在 1ms 到 2ms 之间变化，对应舵机输出轴的角度从 0°到 180 °。



图 6.2 SG90 舵机



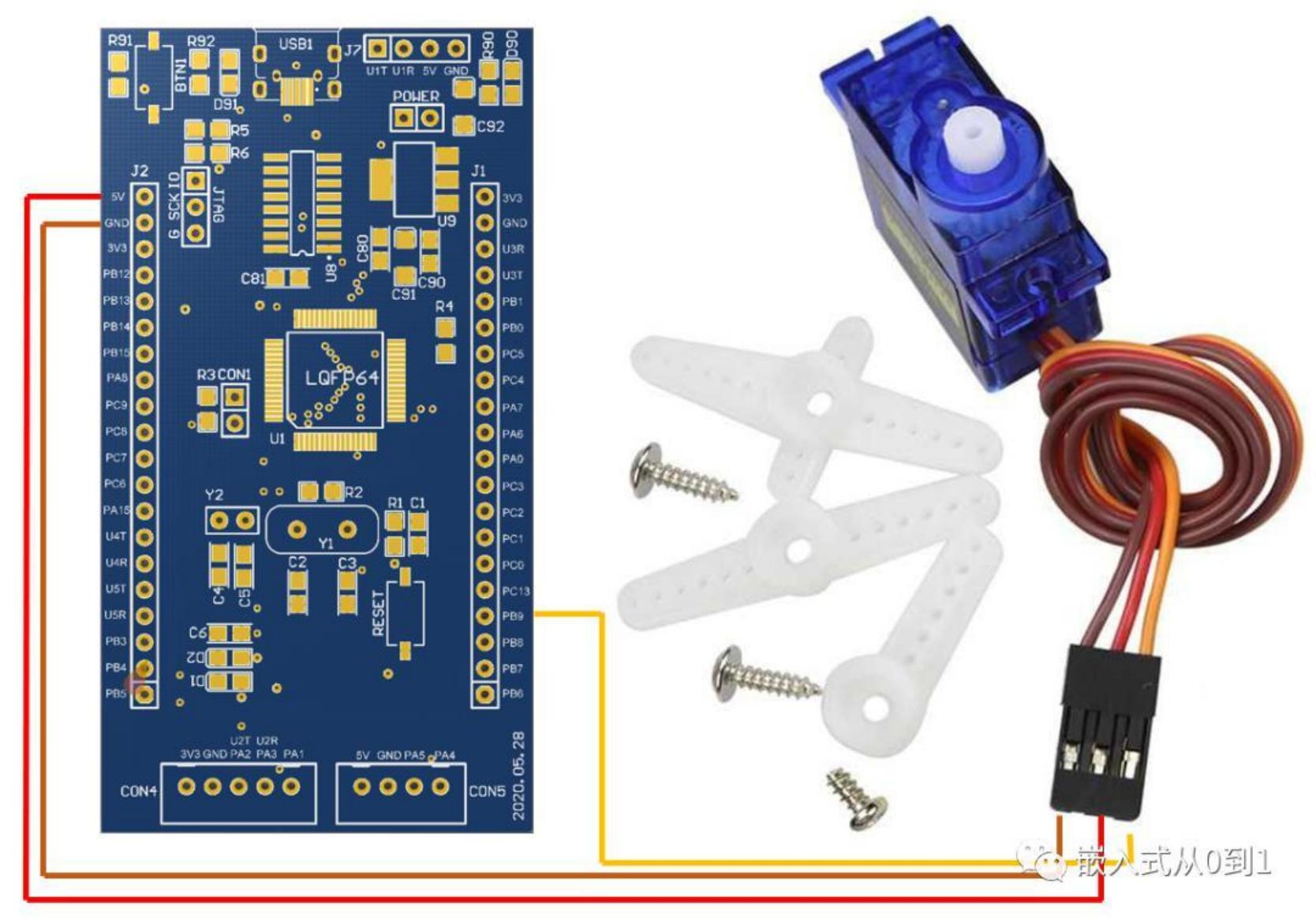


图 6.3 SG90 舵机硬件连接示意图

**硬件连接**：

- 将 SG90 舵机的红色引脚连接到电源（+5V）。

- 将棕色引脚连接到地（GND）。

- 将黄色引脚（PWM 信号线）连接到 TM4C123GH6PM 的 PWM 输出引脚（如PF2）。

**PWM 参数**：

SG90 舵机需要接收 50Hz 的 PWM 信号， 周期为 20ms。 根据舵机的角度， PWM 信 号的高电平持续时间在 1ms 到 2ms 之间变化：

- 0 ° ：高电平持续时间约为 1ms。

- 90 ° ：高电平持续时间约为 1.5ms。

- 180 ° ：高电平持续时间约为 2ms。 **线性关系**：

舵机的角度和 PWM 脉冲宽度之间通常存在线性关系。下面是设置舵机角度的参考程序。 #define PWM\_PERIOD 16000000 / 50 // PWM 周期（20ms， 对应 50Hz）

// 设置舵机角度（0°到 180°)

void SetServoAngle(float angle) { if (angle < 0) angle = 0;

if (angle > 180) angle = 180;

// 计算 PWM 脉冲宽度（1ms 到 2ms 之间）

uint32\_t pulse\_width = (uint32\_t)((1.0 + (angle / 180.0) \* 1.0) \* (PWM\_PERIOD / 20.0)); PWMPulseWidthSet(PWM1\_BASE, PWM\_OUT\_6, pulse\_width);

}

**3、 触发中断**

如果需要在遮挡事件发生时立即响应，可以使用 GPIO 中断。 通过配置 GPIO 中断，并 在中断服务例程中处理事件。