实战演练之GPIO读写

一、概述

本教程将介绍如何在 T-Core 开发板上实现 RISC-V 设计——GPIO 读写：使用开发板上的拨码开关 SW 输入高电平或低电平，并赋值给 LED 灯，实现控制 LED 灯的亮灭。

整个程序的实现主要包括：设计思路/原理的分析，使用 Makefile 编译和下载应用程序，或使用 Eclipse 软件打开 Hello World 工程、修改 main.c 主函数、编译并运行 Hello World 工程，在开发板上验证实验结果。

通过本教程，您将会掌握以下知识：

- 巩固学习使用 Eclipse 软件对 T-CORE RISC-V 的应用程序进行开发；
- 巩固学习使用 Makefile 编译和下载应用程序；
- 了解拨码开关（SW）、发光二极管（LED）的工作原理及驱动方法；
- 学习 RISC-V GPIO 结构；
- 掌握 GPIO 读写的实现原理。

二、设备

1. 硬件
    - PC 主机
    - T-Core 开发套件

    （注：T-Core 是一款基于 Intel® MAX 10 FPGA 的开发套件，支持 RISC-V CPU 的板载 JTAG 调试，是学习 RISC-V CPU 设计或嵌入式系统设计的理想平台。如需了解该套件的详情，请访问 Terasic T-Core 官网。）
2. 软件
    - Quartus Prime 19.1 Lite Edition（已安装好 USB Blaster II 驱动）

    （注：Quartus Prime 软件的下载和安装（USB Blaster II 驱动的安装）可参考 "第八讲 RISC-V on T-Core 的开发流程" 文档。）

    - TCORE-RISCV-E203

    （注：TCORE-RISCV-E203-V1.0.tar.gz 可在 Terasic T-Core 官网设计资源 下载，安装可参考 "第八讲 RISC-V on T-Core 的开发流程" 文档。）

三、设计思路

3.1 T-Core 开发板外设工作原理

T-Core 开发板上有四个连接到 FPGA 端的拨码开关，这些开关都未去抖动，可在电路中用作电平触发的数据输入。每个拨码开关都直接单独地连接到 MAX 10 FPGA，当某个拨码开关拨到向上的位置时，会产生一个高电平到 FPGA；当拨到向下的位置时，会产生一个低电平到 FPGA。图 3.1 为 T-Core 开发板拨码开关和 FPGA 之间的连接示意图。

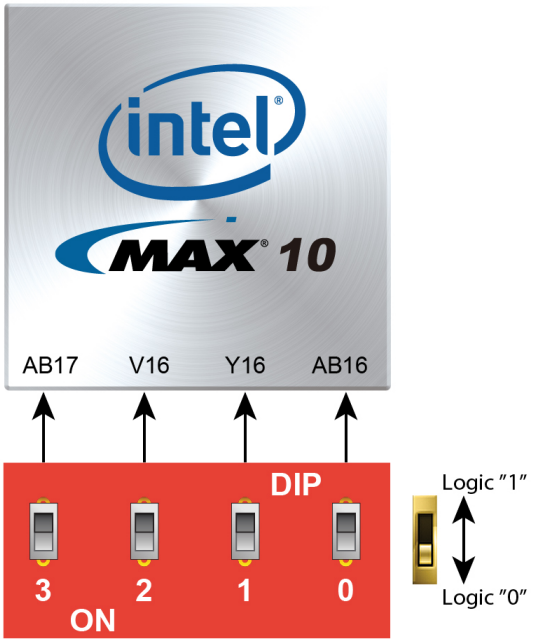

图3.1 T-Core 开发板拨码开关和 FPGA 之间的连接

T-Core 开发板上有四个连接到 FPGA 端的、用户可控的 LED 灯。每个 LED 灯 由 MAX 10 FPGA 直接单独驱动，当 FPGA 输出高电平时，对应 LED 灯点亮；当 FPGA 输出低电平时，对应 LED 灯熄灭。图 3.2 为 T-Core 开发板 LED 灯和 FPGA 之间的连接示意图。

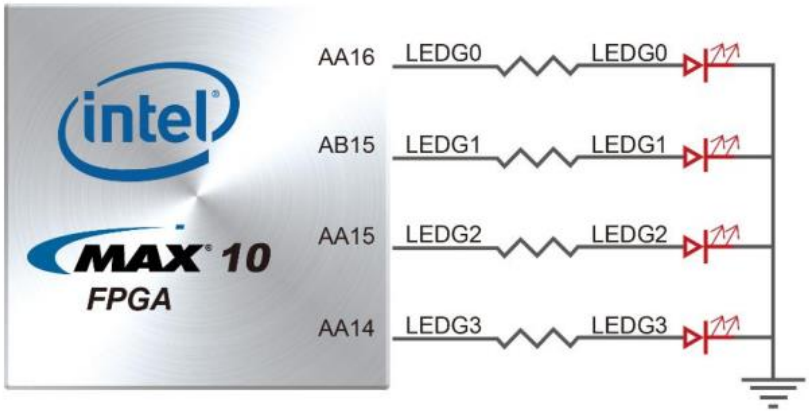

图3.2 T-Core 开发板 LED 灯和 FPGA 之间的连接

3.2 GPIO 结构

GPIO 全称为 General Purpose I/O，GPIO 的 32 个 I/O 结构完全相同，每个独立 I/O 的结构如图 3.3 所示。

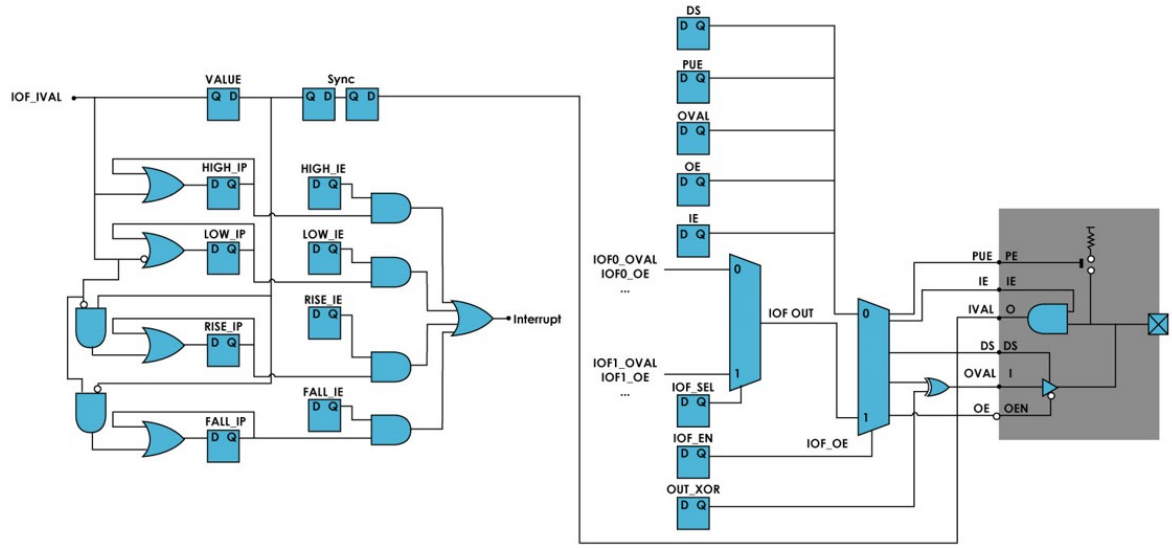

图3.3 GPIO I/O 结构图

由上图可以知道：

1. 每个 I/O 的 Pad 有如下控制信号。
    - PUE：上拉使能
    - IE：输入使能
    - IVAL：输入值
    - DS：输出驱动强度
    - OVAL：输出值
    - OE：输出使能
2. 每个 I/O 具有两种模式，软件控制模式和 IOF（H/W IO Function）控制模式，本教程采用软件控制模式来实现用拨码开关控制 LED 的亮灭。
3. 不管是软件控制模式，还是 IOF 控制模式，都有：
    - GPIO_VALUE 输入值寄存器对应此 I/O 的比特位的值直接来自此 I/O Pad 的 IVAL 控制信号值。
    - 此 I/O Pad 的 PUE 内部上拉控制信号值直接来自 GPIO_PUE 寄存器对应此 I/O 的比特位。
    - 此 I/O Pad 的 DS 驱动强度控制信号值直接来自 GPIO_DS 寄存器对应此 I/O 的比特位。

3.3 GPIO 寄存器列表

本教程主要用到以下寄存器：GPIO_INPUT_VAL 寄存器用于反映 GPIO 的输入值，GPIO_INPUT_EN 寄存器用于在软件控制模式下配置 GPIO 的输入使能，GPIO_OUTPUT_EN 寄存器用于在软件控制模式下配置 GPIO 的输出使能，GPIO_OUTPUT_VAL 寄存器用于在软件控制模式下配置 GPIO 的输出值，GPIO_IOF_EN 寄存器用于选择 H/W IO 的来源，GPIO_OUT_XOR 寄存器用于对输出值进行异或操作控制。

表3.1 GPIO 寄存器列表

寄存器名称	偏移地址	复位默认值	描述
GPIO_INPUT_VAL	0x000	0x0	Pin 的输入值
GPIO_INPUT_EN	0x004	0x0	Pin 的输入使能
GPIO_OUTPUT_EN	0x008	0x0	Pin 的输出使能
GPIO_OUTPUT_VAL	0x00C	0x0	Pin 的输出值
GPIO_IOF_EN	0x038	0x0	H/W IO Function 使能
GPIO_OUT_XOR	0x040	0x0	对输出值进行异或（XOR）操作控制

3.4 软件控制模式

每个 I/O 均可以直接受软件编程的可配置寄存器控制，此模式称为软件控制模式。

当表 3.2 中 GPIO_IOF_EN 寄存器对应此 I/O 的比特位被配置成 0 时，此 I/O 处于软件控制模式，在软件控制模式下：

- 此 I/O Pad 的输入使能 IE 控制信号直接来自 GPIO_INPUT_EN 寄存器对应此 I/O 的比特位。
- 此 I/O Pad 的输出使能 OE 控制信号直接来自 GPIO_OUTPUT_EN 寄存器对应此 I/O 的比特位。
- 此 I/O Pad 的输出值 OVAL 控制信号直接来自 GPIO_PORT 寄存器对应此 I/O 的比特位与 GPIO_OUT_XOR 寄存器对应此 I/O 的比特位进行异或操作的结果。（注意：如果 GPIO_OUT_XOR 寄存器对应此 I/O 的比特位的值为 1，则异或操作等效为取反操作；如果 GPIO_OUT_XOR 寄存器对应此 I/O 的比特位的值为 0，则异或操作等效为不进行操作。）

3.5 GPIO 映射关系

关于 T-Core 的 RISC-V 处理器的 GPIO 与 T-Core 外设 LED 和 SW的映射关系可参考表 3.2。根据 T-Core 的外设与 E203 的 GPIO 映射关系可找到对应的寄存器并进行读写操作。

表3.2 T-Core 外设与 E203 的 GPIO 映射关系

T-Core 的 GPIO 外设	**映射到 E203**
LED0-3	GPIO0-3
SW0-3	GPIO4-7

3.6 程序流程图

本教程是在 GPIO 的软件控制模式下，实现用拨码开关控制 LED 的亮灭。

程序开始时，先配置 LED GPIO 输出使能和 SW GPIO 输入使能，接着开始循环检测 SW 输入的值，并将读取到的值赋给 LED 输出。

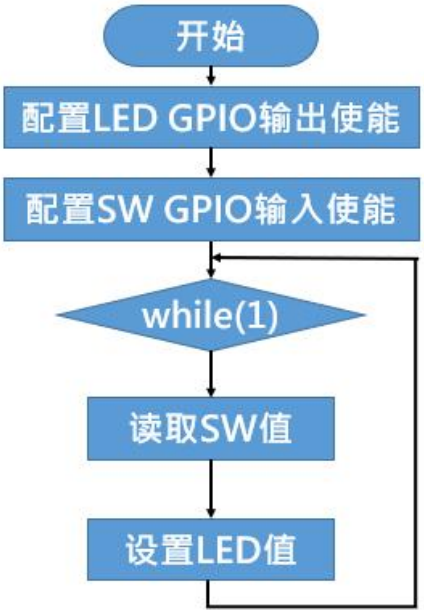

图3.4 GPIO 读写程序流程图

四、操作步骤

4.1 使用 Makefile 编译和下载应用程序（design1）

在本小节中，我们使用地址和宏的方式实现用拨码开关 SW 控制 LED 灯亮灭的功能。

4.1.1 创建并构建工程

1. 创建工程文件夹

   工程通常包含很多例如 .c/.h 或 Makefile 等的设计文件，这些文件通常被存储在同一文件夹下，因此，需要创建一个工程文件夹来存储设计文件和生成文件。

   可以在 "~/Desktop/TCORE-RISCV-E203/TRRV-E-SDK" 的 software 文件夹下创建一个 "demo_gpio_rw1" 文件夹，所以这个文件夹的绝对路径为：

   "~/Desktop/TCORE-RISCV-E203/TRRV-E-SDK/software/demo_gpio_rw1"

2. 创建程序文件（.c文件）

   首先，在 "demo_gpio_rw1" 文件夹下创建一个 "demo_gpio_rw1.c" 的文本文档。

   定义 SW 和 LED 掩码。

```
// LED0-3:bit 0-3
// SW0-3 :bit 4-7
#define TERASIC_LED_MASK   0x0000000F        // led mask
#define TERASIC_SW_MASK    0x000000F0        // switch mask
```

   定义 GPIO 寄存器地址，GPIO_BASE_ADDR 为 GPIO 在系统中的基地址，GPIO_INPUT_VAL_ADDR 寄存器用于反映 GPIO 的输入值，GPIO_INPUT_EN_ADDR 寄存器用于在软件控制模式下配置 GPIO 的输入使能，GPIO_OUTPUT_EN_ADDR 寄存器用于在软件控制模式下配置 GPIO 的输出使能，GPIO_OUTPUT_VAL_ADDR 寄存器用于在软件控制模式下配置 GPIO 的输出值。这里的偏移地址可从 3.3 节中介绍的寄存器列表获得。

```
#define GPIO_BASE_ADDR       (0x10012000)           // gpio base address
#define GPIO_INPUT_VAL_ADDR  (GPIO_BASE_ADDR+0x00)  // gpio input value register addr
#define GPIO_INPUT_EN_ADDR   (GPIO_BASE_ADDR+0x04)  // gpio input enable register addr
#define GPIO_OUTPUT_EN_ADDR  (GPIO_BASE_ADDR+0x08)  // gpio output enable register addr
#define GPIO_OUTPUT_VAL_ADDR (GPIO_BASE_ADDR+0x0C)  // gpio output value register addr
```

到这里准备工作就做好啦！接下来我们在主函数中实现拨码开关 SW 控制 LED 亮灭的功能。

首先配置 LED GPIO 输出使能和 SW GPIO 输入使能。

```
int main(void){
    // Set LED0-3 output
    *(volatile unsigned int *)(GPIO_OUTPUT_EN_ADDR) |= TERASIC_LED_MASK;

    // Set SW0-3 input
    *(volatile unsigned int *)(GPIO_INPUT_EN_ADDR)  |= TERASIC_SW_MASK;
```

定义变量 sw_value 即读取的 SW 值、gpio_output_val 为 GPIO 的输出值、gpio_input_val 为 GPIO 的输入值。

```
    unsigned int sw_value=0, gpio_output_val=0, gpio_input_val=0;
```

要实现每次改变 SW 的值后都能够循环读取并赋值给 LED 输出，我们将下面读取 SW 和写入 LED 的代码部分放在 while 循环中执行。

首先读取 GPIO_INPUT_VAL_ADDR 寄存器获取 GPIO 的输入值 gpio_input_val，将 gpio_input_val 与 SW 掩码进行按位与操作后得到 sw_value（在这里将其右移 4 位是为了方便后面 LED 的赋值操作）。

```
    while(1){
        // Get SW0-3 Value
        gpio_input_val  = *(volatile unsigned int *)(GPIO_INPUT_VAL_ADDR);
        sw_value = (gpio_input_val & TERASIC_SW_MASK)>>4;
```

接着读取 GPIO_OUTPUT_VAL_ADDR 寄存器获取 GPIO 的输出值 gpio_output_val，将 gpio_output_val 与取反后的 LED 掩码进行按位与操作，目的是清零 GPIO 0-3 位，再与 sw_value 进行按位或操作后得到新的 gpio_output_val，即将 sw_value 的值赋值到 GPIO 的 0-3 位，最后将 gpio_output_val 的值存入 GPIO_OUTPUT_VAL_ADDR 寄存器即实现用拨码开关 SW 控制 LED 灯亮灭的功能。

```
        // Set LED0-3 Value
        gpio_output_val = *(volatile unsigned int *)
(GPIO_OUTPUT_VAL_ADDR);
        gpio_output_val = gpio_output_val & (~TERASIC_LED_MASK) |
(sw_value<<0);
        *(volatile unsigned int *)(GPIO_OUTPUT_VAL_ADDR) =
gpio_output_val;
    }   // end of while

}   // end of main
```

3. 创建 **Makefile** 文件

在 "demo_gpio_rw1" 文件夹下创建一个空白文本文档并命名为 "Makefile"，然后在文档中写入如下所示内容。Makefile 文件中制定了 Linux 编译工程的一系列规则，最后编译生成可执行文件。

```
TARGET = demo_gpio_rw1
CFLAGS += -O1

BSP_BASE = ../../bsp

C_SRCS += demo_gpio_rw1.c

include $(BSP_BASE)/tcore-e203/env/common.mk
```

在 Makefile 中："TARGET" 定义了生成的可执行文件名字，这个例子中生成的可执行文件名将为 "demo_gpio_rw1"。

4.1.2 编译工程

1. 使用 Linux 命令 "cd" 切换当前目录至工程路径 "~/Desktop/TCORE-RISCV-E203/TRRV-E-SDK/software"，然后，执行 "make software PROGRAM=demo_gpio_rw1" 命令编译应用程序。如图 4.1.1 所示。

```
cd Desktop/TCORE-RISCV-E203/TRRV-E-SDK/software          # 切换当前目录至工程路径
make software PROGRAM=demo_gpio_rw1                      # 编译应用程序
```

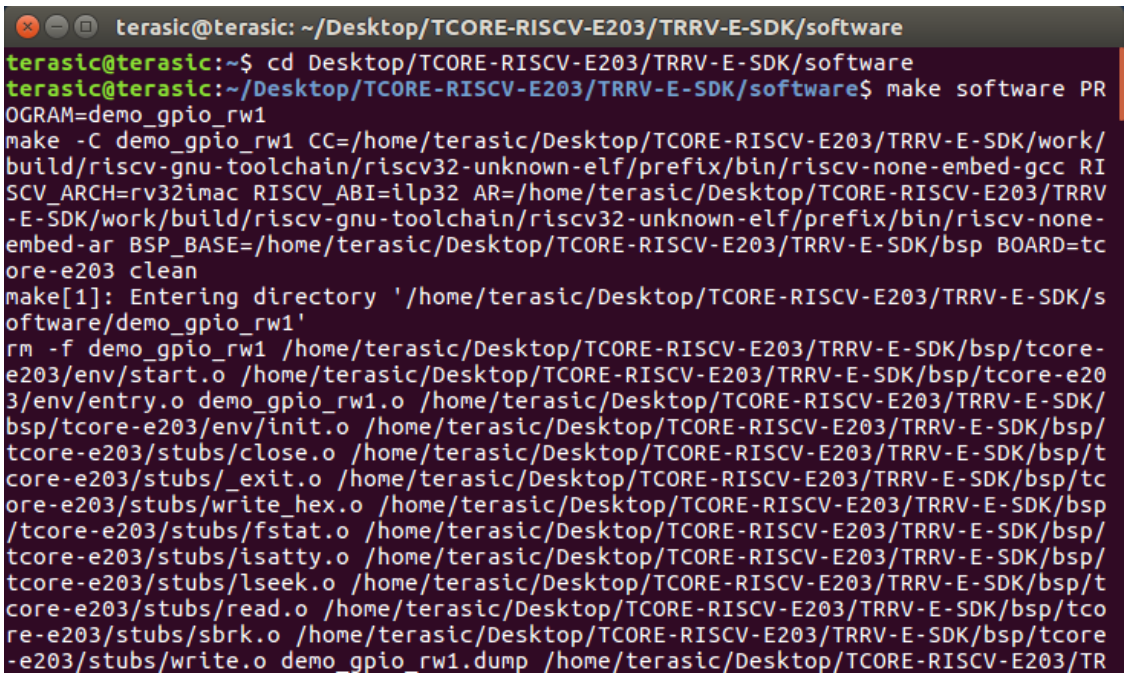

图4.1.1 编译应用程序

2. 工程编译完成之后，可以看到在 "demo_gpio_rw1" 文件夹下生成了可执行文件 "demo_gpio_rw1"，如图 4.1.2 所示。

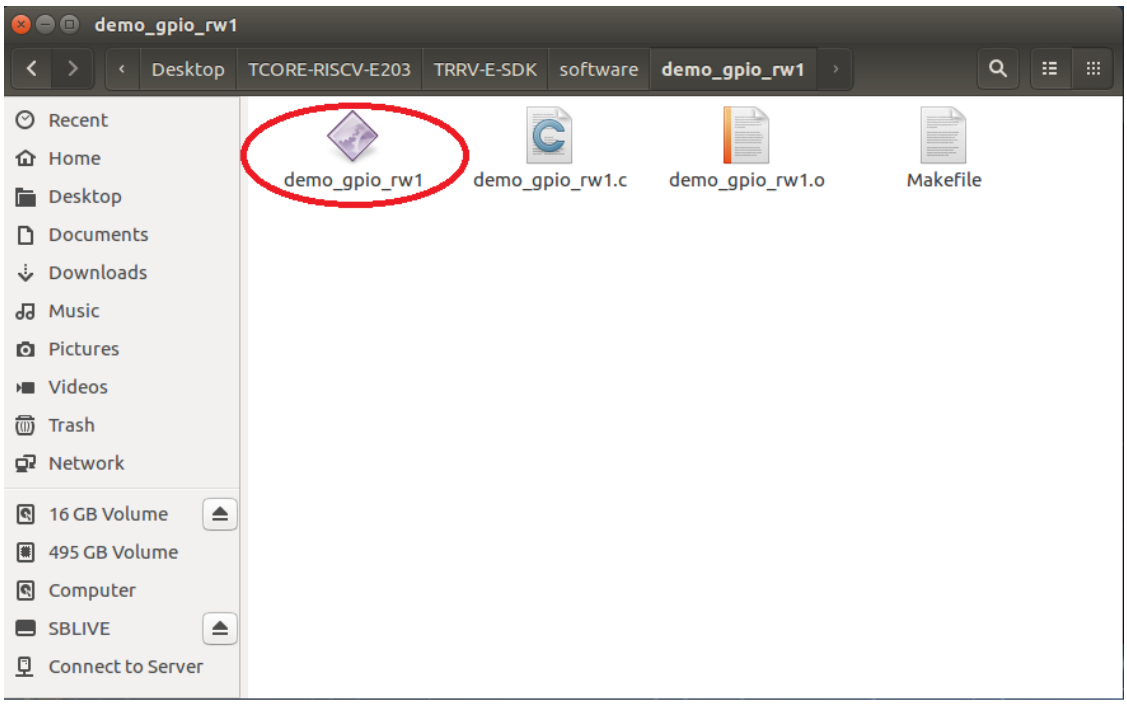

图4.1.2 编译生成二进制文件

4.1.3 执行工程

1. 关闭 T-Core 开发板电源后，将开发板上的 SW2: SW2.1=1，SW2.2=0，选择 RISC-V JTAG 链路。

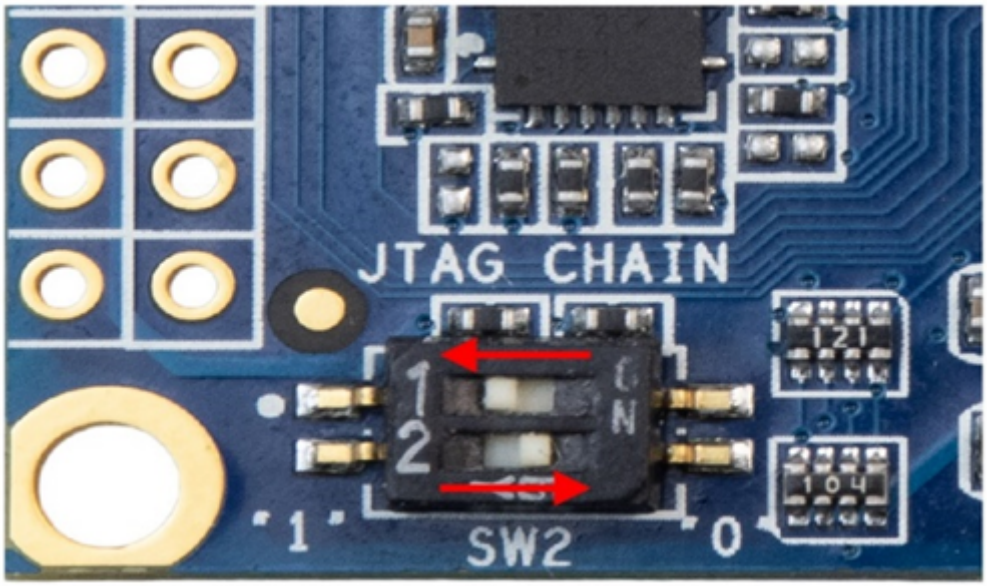

图4.1.3 设置 SW2 开关

2. 将 USB Blaster II 线缆的一端连接到开发板的 USB Blaster 接口（J2），另一端连接至 PC 主机的 USB 接口。

图4.1.4 连接开发板和 PC

3. 使用 "make upload PROGRAM=demo_gpio_rw1" 将可执行文件 "demo_gpio_rw1" 下载到 T-Core 开发板的 QSPI Flash 中。

```
make upload PROGRAM=demo_gpio_rw1
```

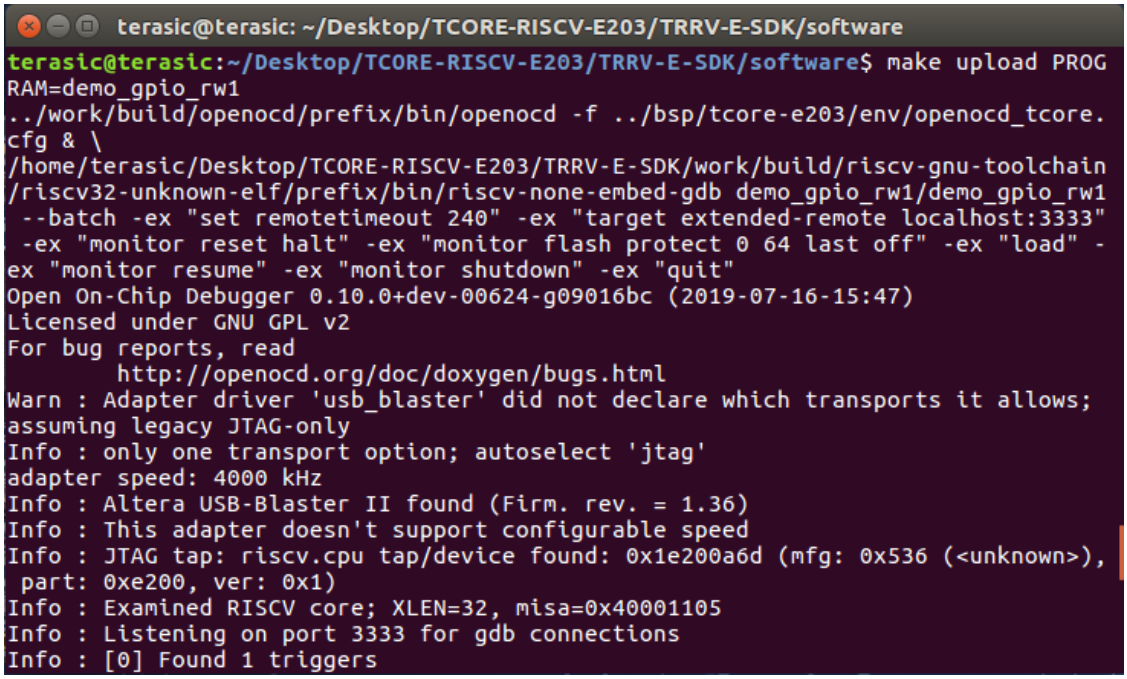

图4.1.5 下载可执行文件

4.1.4 运行结果

程序下载完成后，当将 T-Core 开发板上拨码开关 SW 拨到向上的位置时，可以观察到对应位的 LED 点亮，当将 T-Core 开发板上拨码开关 SW 拨到向下的位置时，可以观察到对应位的 LED 熄灭。

4.2 使用 Makefile 编译和下载应用程序（design2）

在本小节中，我们采用包含头文件的方式实现用拨码开关 SW 控制 LED 灯亮灭的功能。

4.2.1 创建并构建工程

1. 创建工程文件夹

   同样地，在 "~/Desktop/TCORE-RISCV-E203/TRRV-E-SDK" 的 software 文件夹下创建一个 "demo_gpio_rw2" 文件夹。

2. 创建程序文件（.c文件）

   首先，在 "demo_gpio_rw2" 文件夹下创建一个 "demo_gpio_rw2.c" 的文本文档。

   包含 "platform.h" 头文件，这个头文件中包含了 "stdint.h" 头文件和 "gpio.h" 头文件，"stdint.h" 定义了几种扩展的数据类型和宏，"gpio.h" 定义了本教程用到的 GPIO 寄存器。

```
#include "platform.h"
```

   定义 SW 和 LED 掩码。

```
// LED0-3:bit 0-3
// SW0-3 :bit 4-7
#define TERASIC_LED_MASK   0x0000000F        // led mask
#define TERASIC_SW_MASK    0x000000F0        // switch mask
```

到这里准备工作就做好啦！接下来我们在主函数中实现拨码开关 SW 控制 LED 亮灭的功能。

首先配置 GPIO 输出使能和输入使能。

```
int main(void){
    // Set LED0-3 output
    GPIO_REG(GPIO_OUTPUT_EN) |= TERASIC_LED_MASK;

    // Set SW0-3 input
    GPIO_REG(GPIO_INPUT_EN)  |= TERASIC_SW_MASK;
```

定义变量 sw_value 即读取的 SW 值、gpio_output_val 为 GPIO 的输出值、gpio_input_val 为 GPIO 的输入值。

```
    uint32_t sw_value=0, gpio_output_val=0, gpio_input_val=0;
```

要实现每次改变 SW 的值后都能够循环读取并赋值给 LED 输出，我们将下面读取 SW 和写入 LED 的代码部分放在 while 循环中执行。

首先读取 GPIO_INPUT_VAL 寄存器获取 GPIO 的输入值 gpio_input_val，将 gpio_input_val 与 SW 掩码进行按位与操作后得到 sw_value（在这里将其右移 4 位是为了方便后面 LED 的赋值操作）。

```
    while(1){
        // Get SW0-3 Value
        gpio_input_val  = GPIO_REG(GPIO_INPUT_VAL);
        sw_value = (gpio_input_val & TERASIC_SW_MASK)>>4;
```

接着读取 GPIO_OUTPUT_VAL 寄存器获取 GPIO 的输出值 gpio_output_val，将 gpio_output_val 与取反后的 LED 掩码进行按位与操作，目的是清零 GPIO 0-3 位，再与 sw_value 进行按位或操作后得到新的 gpio_output_val，即将 sw_value 的值赋值到 GPIO 的 0-3 位，最后将 gpio_output_val 的值存入 GPIO_OUTPUT_VAL 寄存器即实现用拨码开关 SW 控制 LED 灯亮灭的功能。

```
        // Set LED0-3 Value
        gpio_output_val = GPIO_REG(GPIO_OUTPUT_VAL);
        gpio_output_val = gpio_output_val & (~TERASIC_LED_MASK) |
(sw_value<<0);
        GPIO_REG(GPIO_OUTPUT_VAL) = gpio_output_val;
    }   // end of while

}   // end of main
```

3. 创建 **Makefile** 文件

在 "demo_gpio_rw2" 文件夹下创建一个空白文本文档并命名为 "Makefile"，然后在文档中写入如下所示内容。Makefile 文件中制定了 Linux 编译工程的一系列规则，最后编译生成可执行文件。

```
TARGET = demo_gpio_rw2
CFLAGS += -O1

BSP_BASE = ../../bsp

C_SRCS += demo_gpio_rw2.c

include $(BSP_BASE)/tcore-e203/env/common.mk
```

在 Makefile 中："TARGET" 定义了生成的可执行文件名字，这个例子中生成的可执行文件名将为 "demo_gpio_rw2"。

4.2.2 编译工程

1. 使用 Linux 命令 "cd" 切换当前目录至工程路径 "~/Desktop/TCORE-RISCV-E203/TRRV-E-SDK/software"，然后，执行 "make software PROGRAM=demo_gpio_rw2" 命令编译应用程序。如图 4.2.1 所示。

```
cd Desktop/TCORE-RISCV-E203/TRRV-E-SDK/software          # 切换当前目录至工程路径
make software PROGRAM=demo_gpio_rw2                      # 编译应用程序
```

2. 工程编译完成之后，可以看到在 "demo_gpio_rw2" 文件夹下生成了可执行文件 "demo_gpio_rw2"，如图 4.2.2 所示。

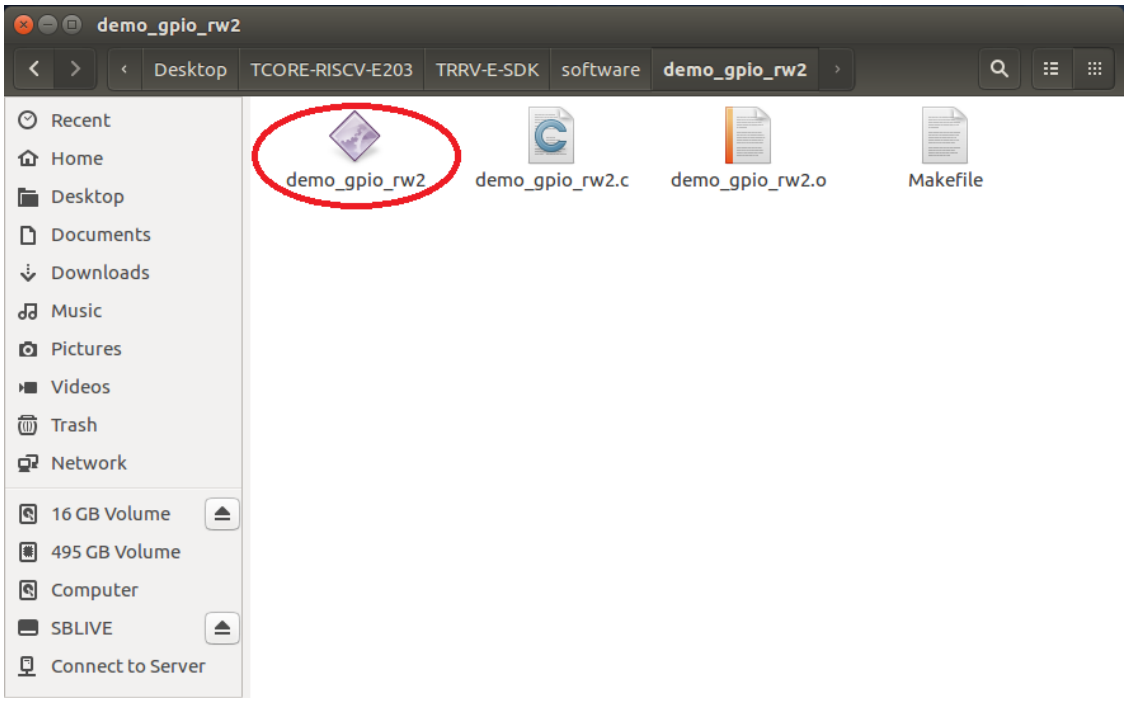

图4.2.2 编译生成二进制文件

4.2.3 执行工程

1. 关闭 T-Core 开发板电源后，将开发板上的 SW2: SW2.1=1，SW2.2=0，选择 RISC-V JTAG 链路。

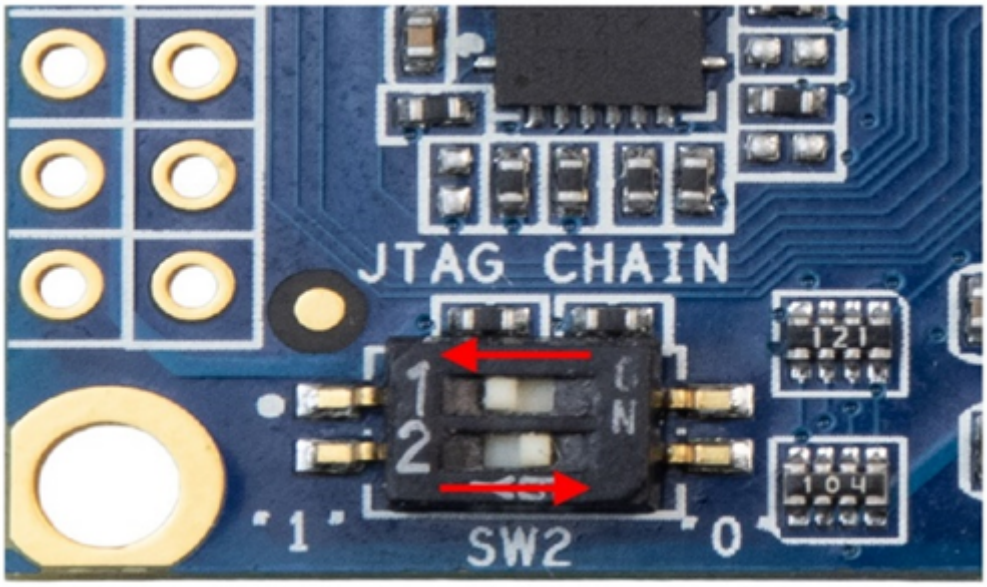

图4.2.3 设置 SW2 开关

2. 将 USB Blaster II 线缆的一端连接到开发板的 USB Blaster 接口（J2），另一端连接至 PC 主机的 USB 接口。
3. 使用 "make upload PROGRAM=demo_gpio_rw2" 将可执行文件 "demo_gpio_rw2" 下载到 T-Core 开发板的 QSPI Flash 中。

4.2.4 运行结果

程序下载完成后，当将 T-Core 开发板上拨码开关 SW 拨到向上的位置时，可以观察到对应位的 LED 点亮，当将 T-Core 开发板上拨码开关 SW 拨到向下的位置时，可以观察到对应位的 LED 熄灭。

4.3 使用 Eclipse 软件编译和下载应用程序

在进行下面的操作前，请先将在第八讲中创建的 blinking_LED 工程复制到 "~/eclipse-workspace" 文件夹。（注：请使用依据 v1.1 及以上版本的第八讲手册创建的 blinking_LED 工程）

4.3.1 打开 Hello World 工程

1. 将文件夹命名由 "blinking_LED" 修改为 "gpio_rw"，如图 4.3.1 所示。

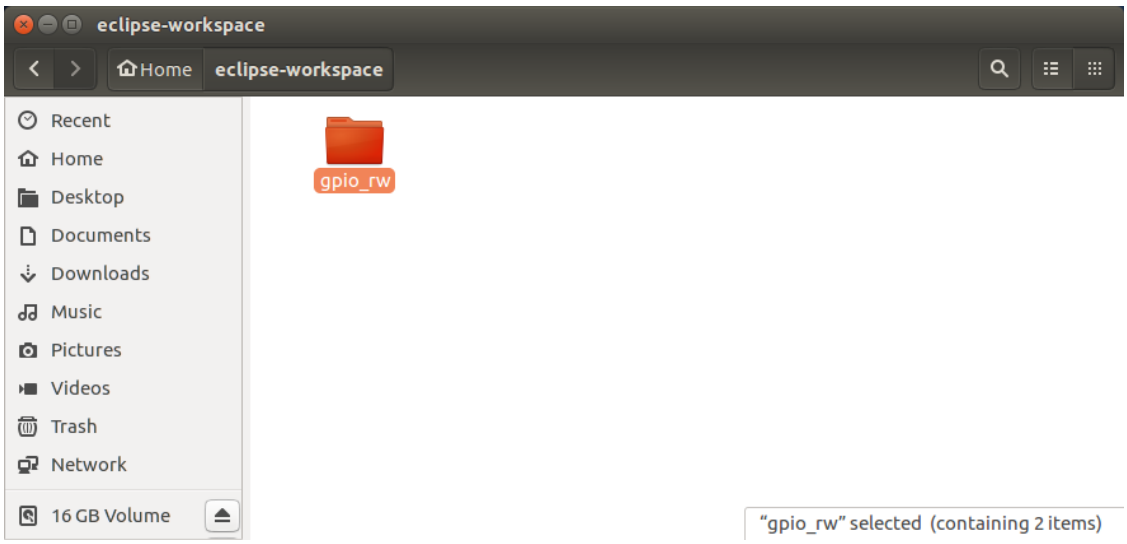

图4.3.1 修改文件名为 "gpio_rw"

2. 双击 GNU_MCU_Eclipse 文件夹中的 eclipse 文件夹下的可执行文件 eclipse，启动 Eclipse 软件。

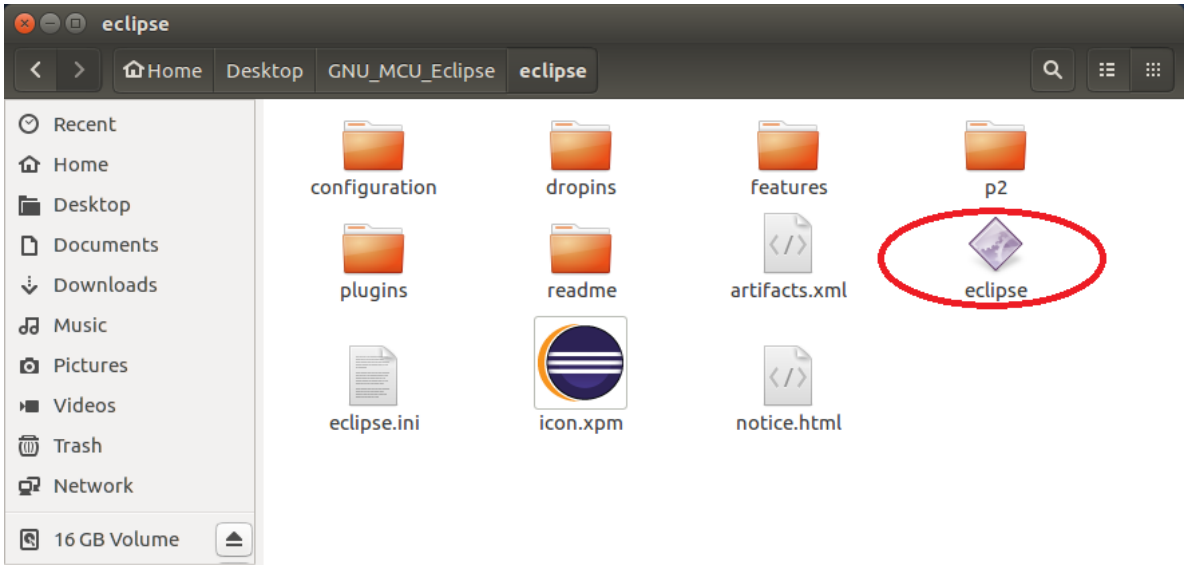

图4.3.2 启动 Eclipse

3. 启动 Eclipse 后，弹出设置 Workspace 的对话框，如图 4.3.3 所示，默认为 home 下的 eclipse-workspace（可根据需要自行设置）。

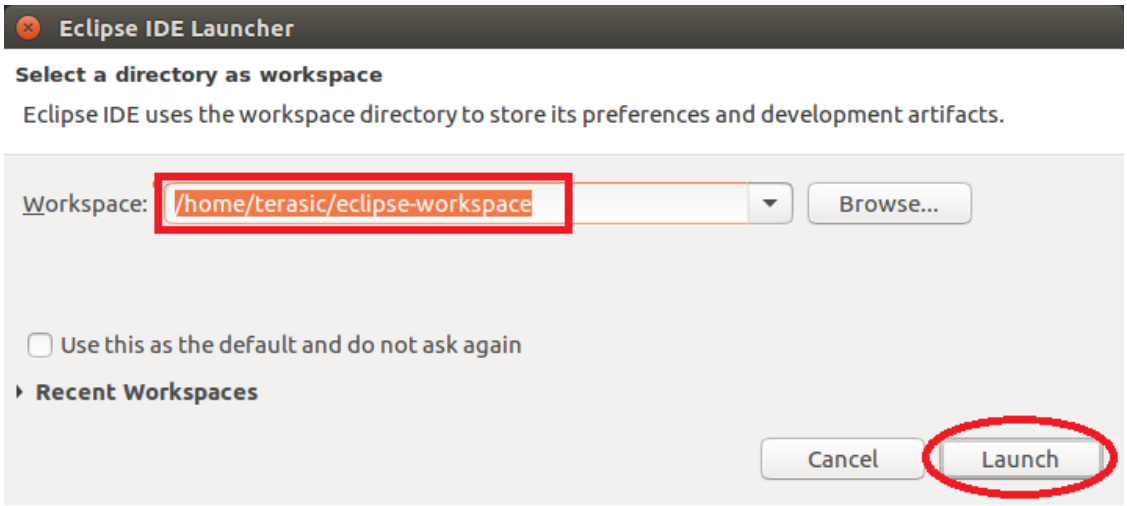

图4.3.3 设置 Workspace

4. 设置好 Workspace 目录后，单击 Launch，将会启动 Eclipse，进入 Welcome 界面，如图 4.3.4 所示。

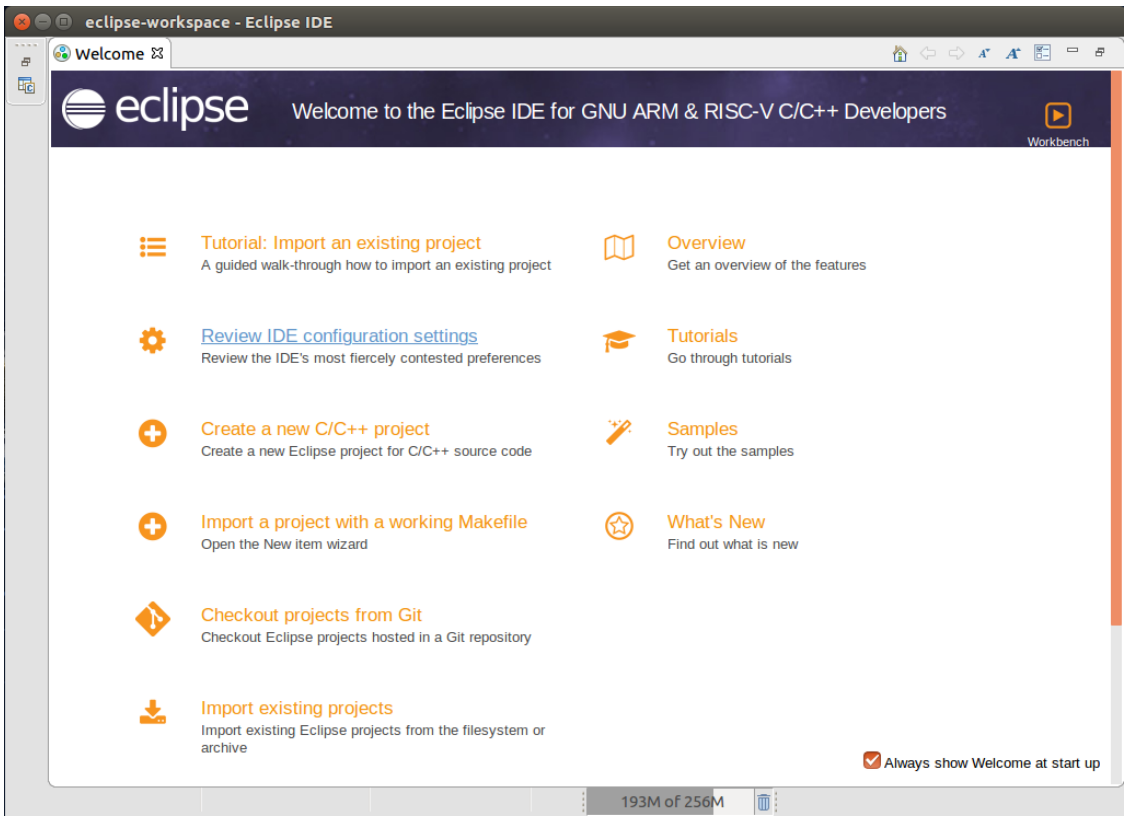

图4.3.4 进入 Eclipse 界面

5. 点击 Welcome 处的叉号，关闭 Welcome 界面，如图 4.3.5 所示。

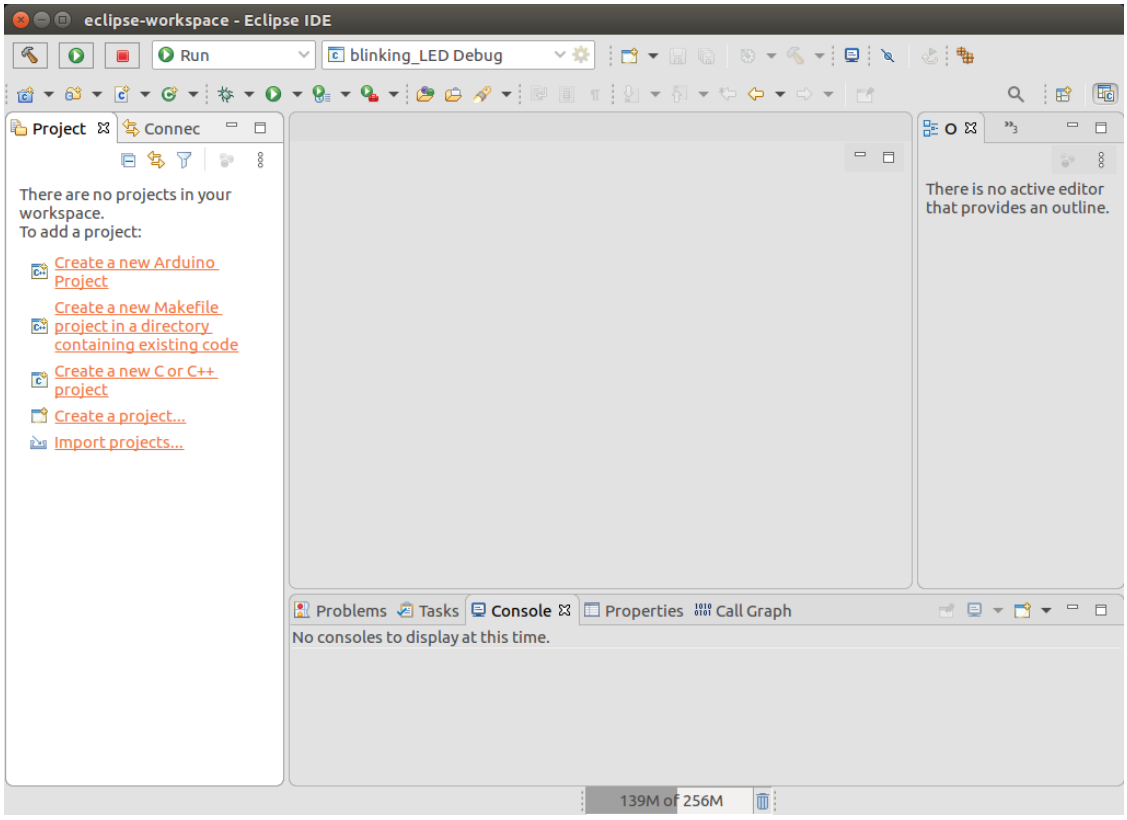

图4.3.5 进入 Eclipse 界面

6. 点击菜单栏 File -> Import... 导入工程，出现如图 4.3.6 所示界面，选择 "Existing Projects into Workspace"，点击 Next。（注：把鼠标移动到顶部菜单栏就会出现）

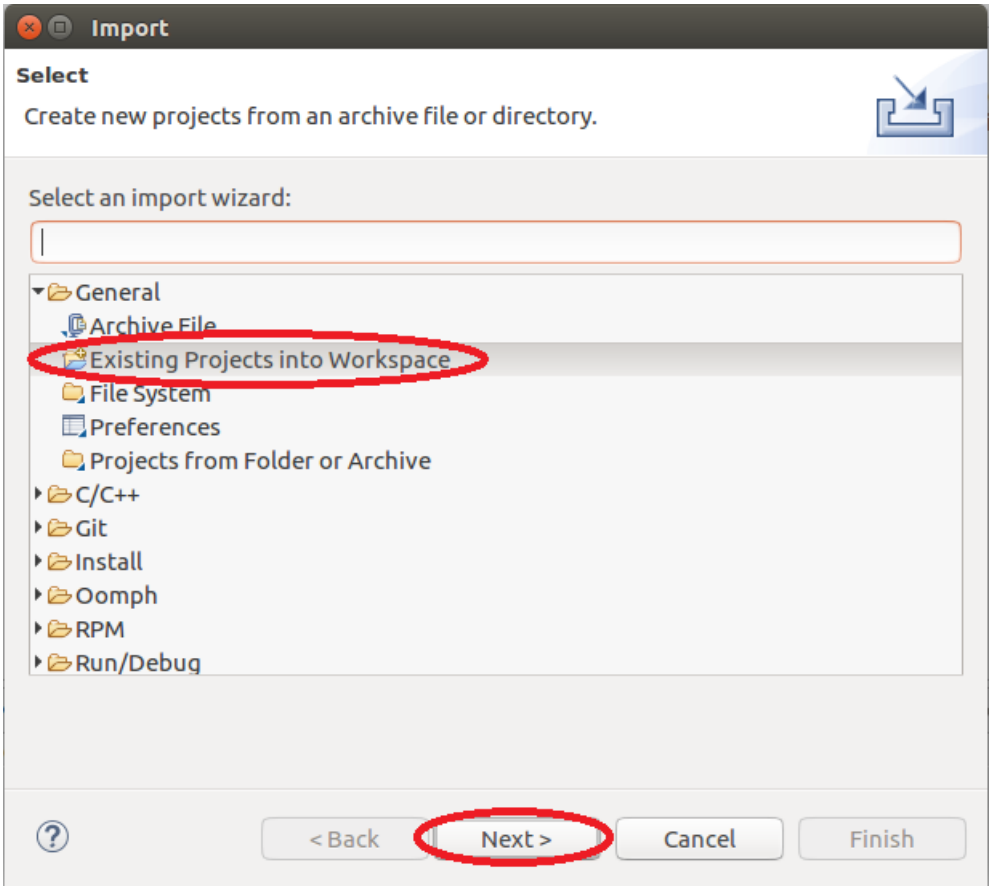

图4.3.6 选择导入工程类型

7. 点击 Browse 导入已有的工程。

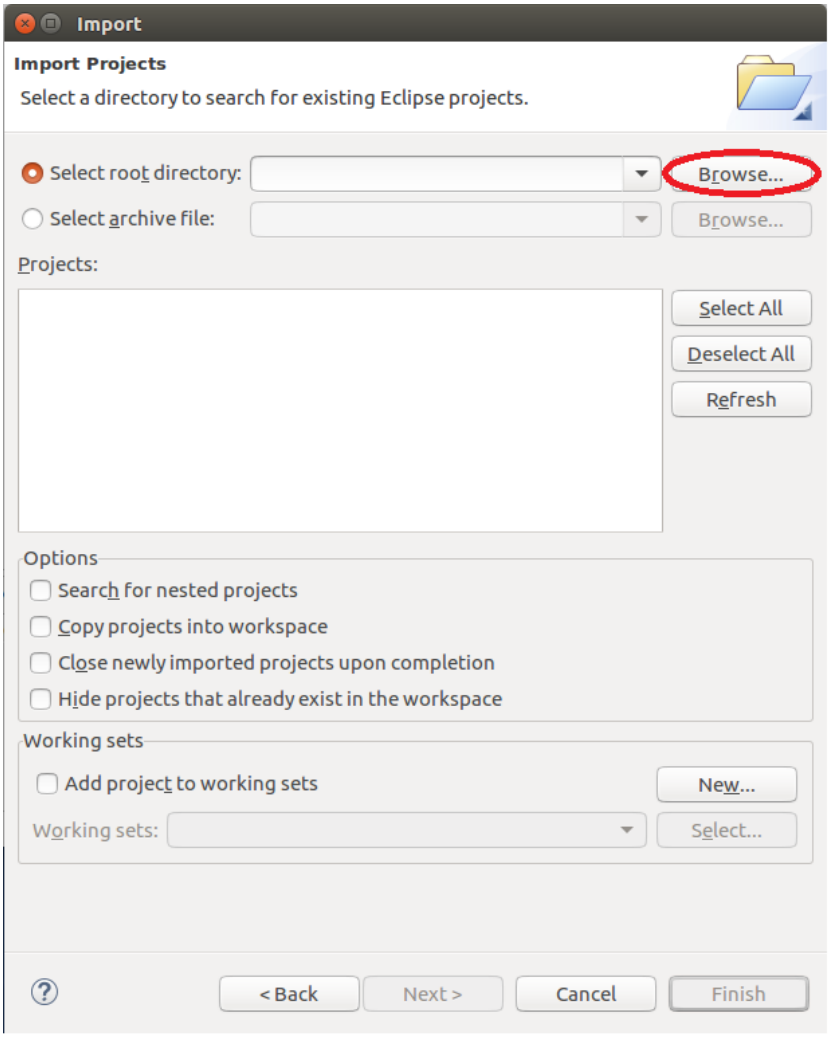

图4.3.7 点击 Browse

8. 选择要添加的 gpio_rw 工程，点击 OK。

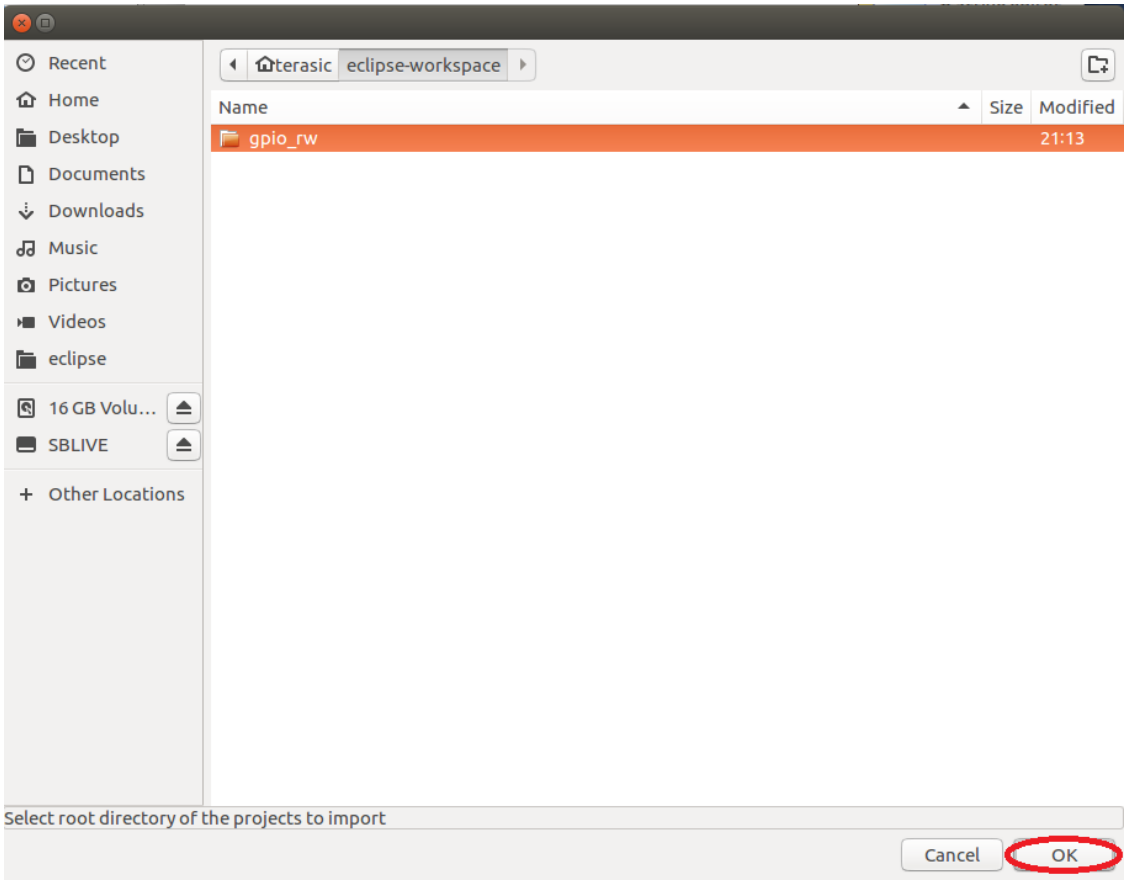

图4.3.8 添加 gpio_rw 工程

9. 勾选 "Add projects to working sets" 将 gpio_rw 工程添加到当前工作空间，点击 Finish。

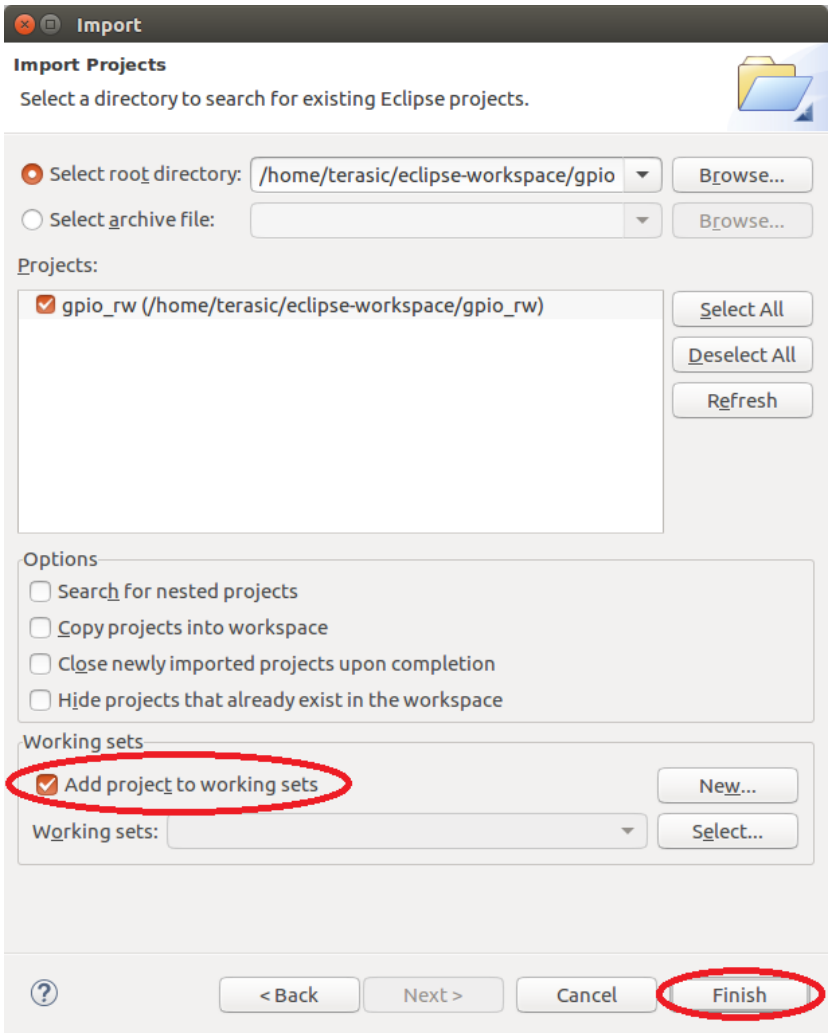

图4.3.9 添加 gpio_rw 工程到工作空间

10. 导入后的工程界面如图 4.3.10 所示。

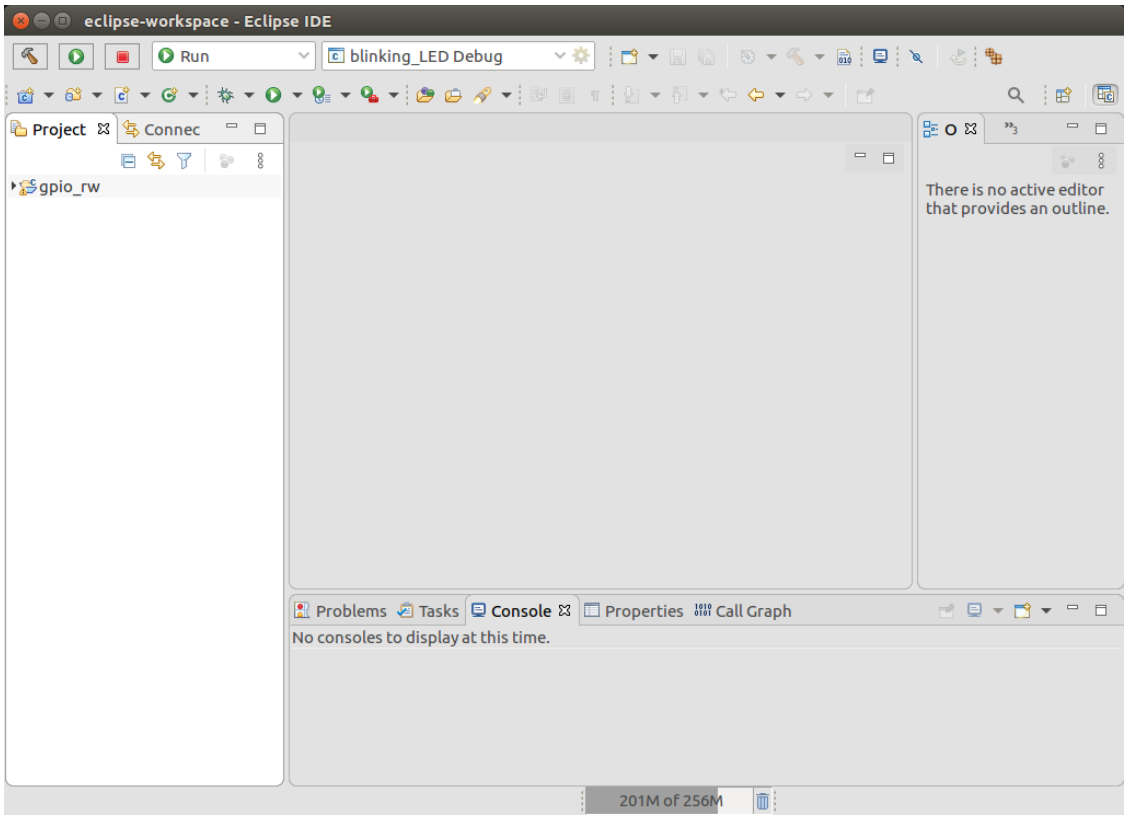

图4.3.10 导入后的工程界面

4.3.2 修改 main.c 文件

点击 gpio_rw --> src --> bsp 下拉框，双击打开 main.c 文件，复制 4.1.1 节中的 demo_gpio_rw1.c 文件中的代码替换掉当前 "main.c" 的代码并保存，如图 4.3.11 所示。

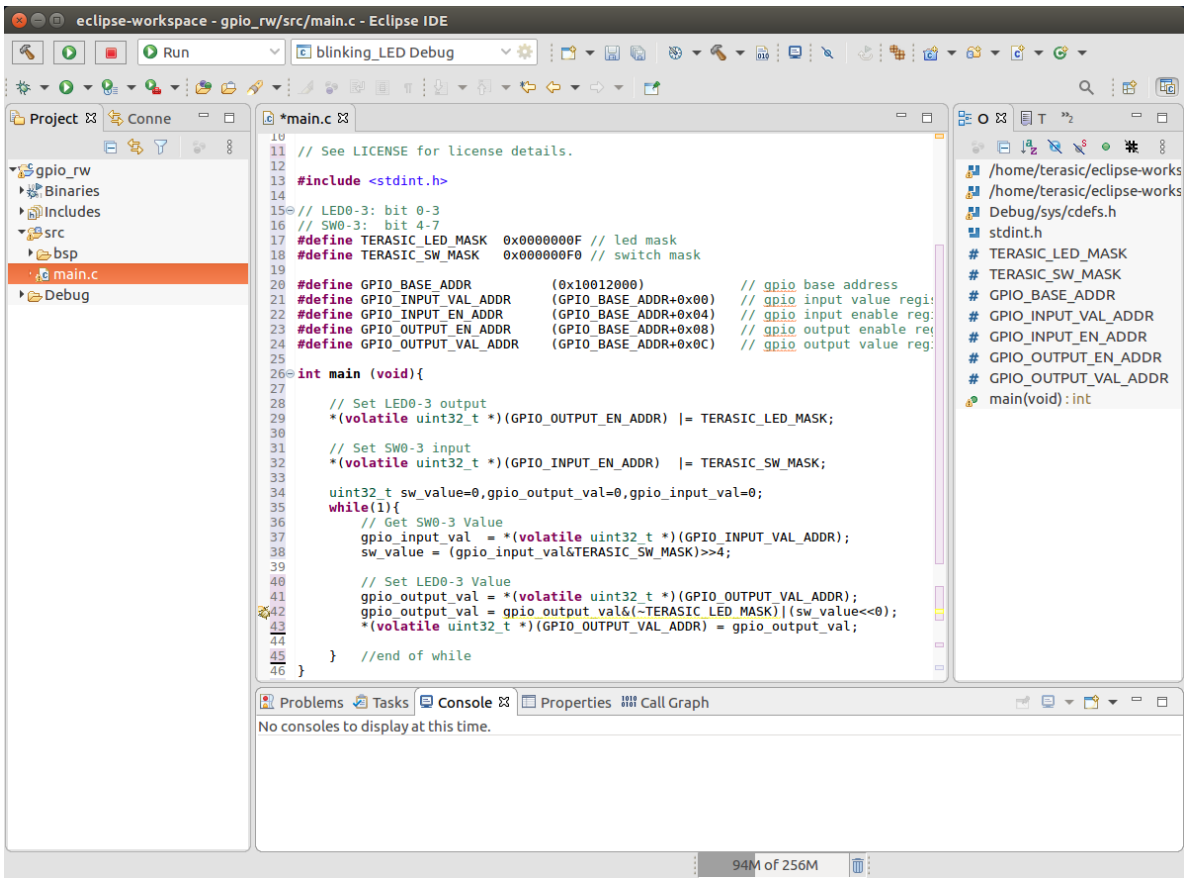

图4.3.11 修改 main.c 文件

4.3.3 编译 gpio_rw工程

1. 在 Eclipse 主界面中，选中 gpio_rw 工程，右键点击 Properties，点击 C/C++ Build 下拉选择 Settings，点击 Tool Settings 选项卡下的 Optimization，修改 Optimization level 为 "Optimize(-O1)"，如图 4.3.12 所示。

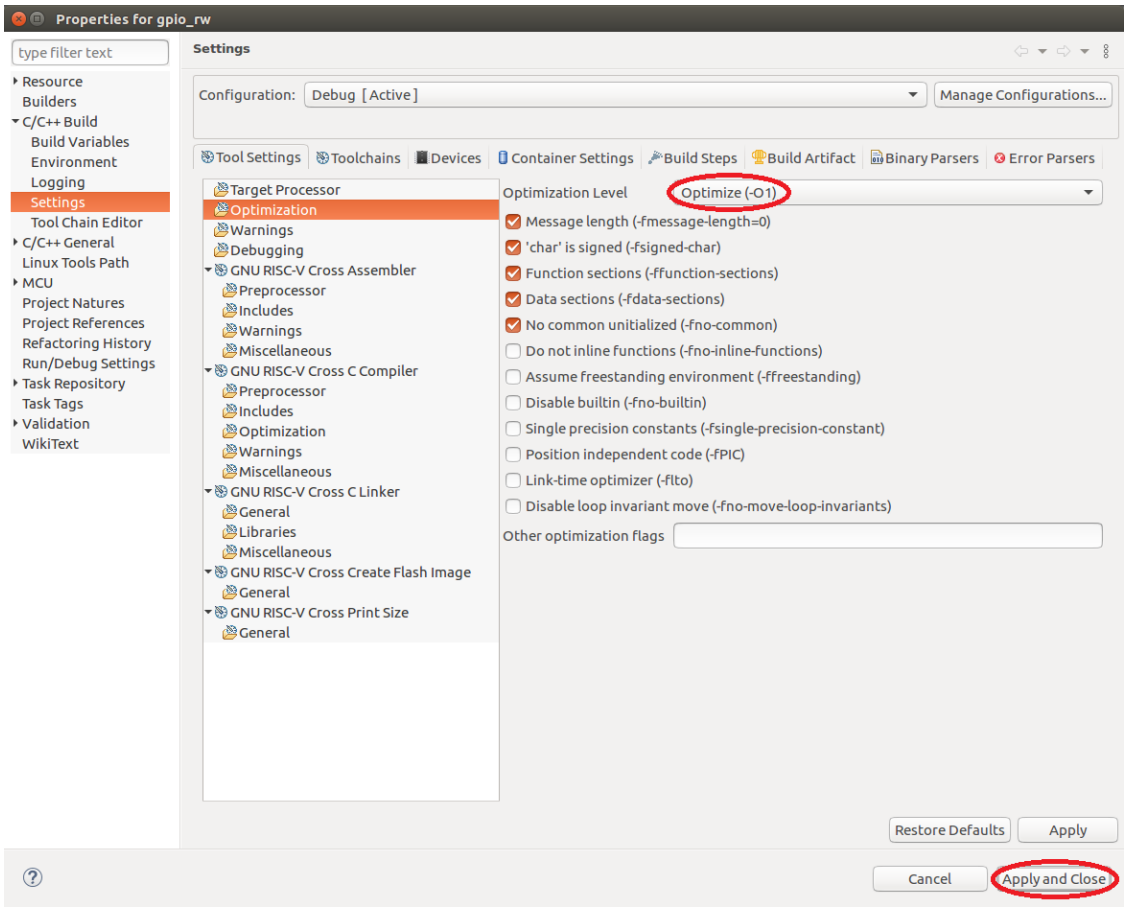

图4.3.12 修改 Optimization level

2. 选中 gpio_rw工程，右键点击 Clean Project；再次选中 gpio_rw工程，右键点击 Build Project，若 gpio_rw工程参照之前的步骤设置正确，则在这一步会编译成功，如图 4.3.13 所示。（注：需要右键点击 Refresh 才可以在 Debug 下拉项中看到生成的 .elf 文件）

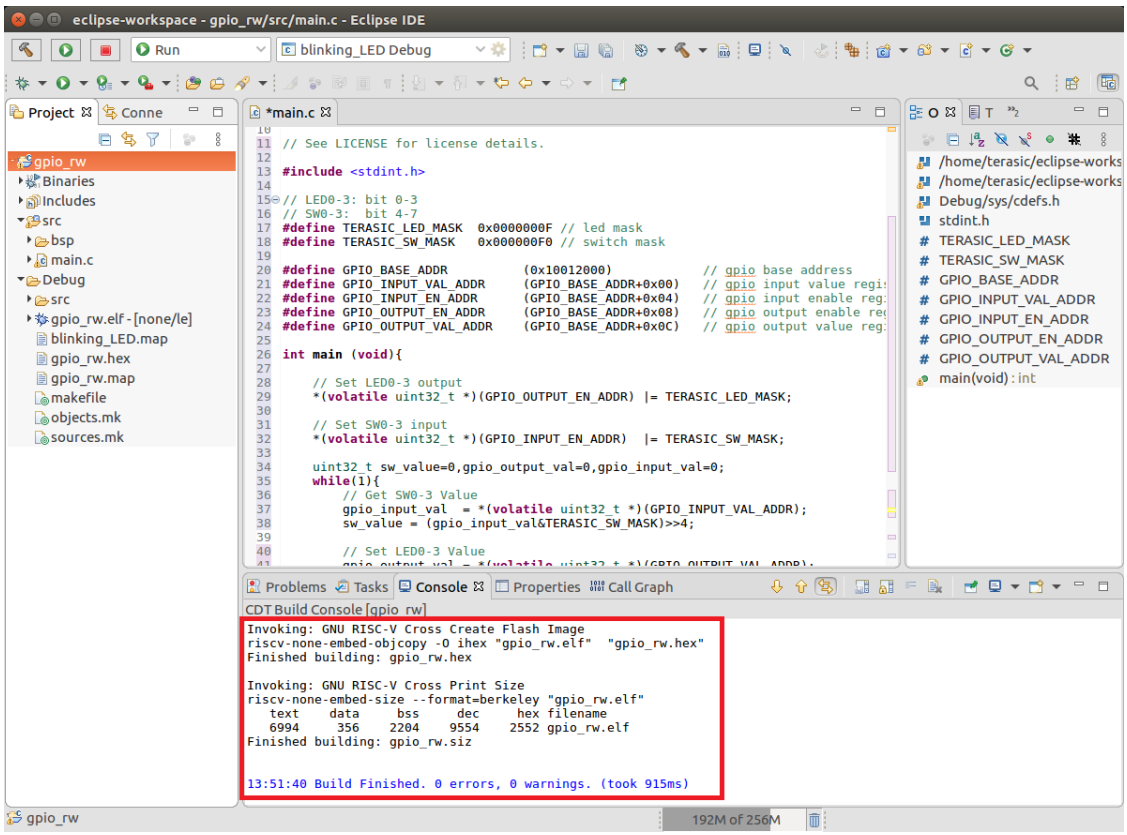

图4.3.13 编译成功

3. 右键 Debug 下拉选项中的 "blinking_LED.map"，点击 Delete，删除完成后如图 4.3.14 所示。

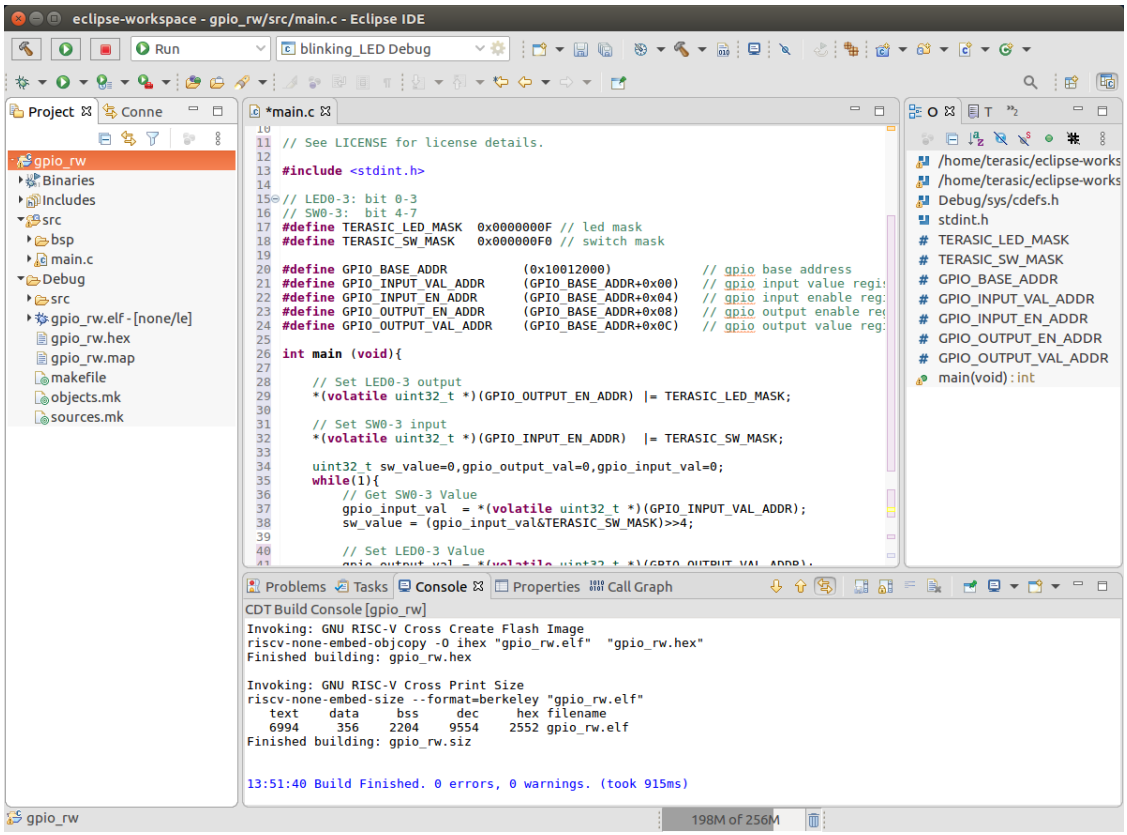

图4.3.14 删除 blinking_LED.map 文件

4.3.4 运行 gpio_rw工程

1. 使用 USB Cable 将 T-Core 开发板与 PC 电脑进行连接来烧录应用程序。具体操作如下：
    - 关闭 T-Core 开发板电源后，将开发板上的 SW2: SW2.1=1，SW2.2=0，选择 RISC-V JTAG 链路。

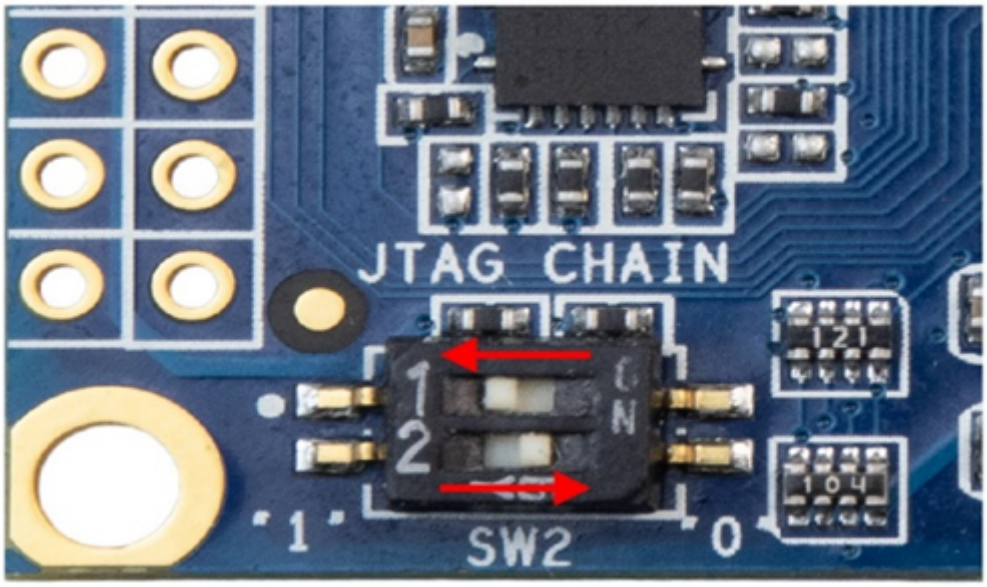

图4.3.15 设置 SW2 开关

- 将 USB Blaster II 线缆的一端连接到开发板的 USB Blaster 接口（J2），另一端连接至 PC 主机的 USB 接口。

2. 选中 gpio_rw工程，右键点击 Run As -> Run Configurations...，双击 GDB OpenOCD Debugging 会出现如图 4.3.16 所示的 gpio_rw Debug 界面，在 Config options 中添加 "-f /home/terasic/Desktop/TCORE-RISCV-E203/TRRV-E-SDK/bsp/tcore-e203/env/openocd_tcore.cfg" 和 "-s /home/terasic/Desktop/TCORE-RISCV-E203/TRRV-E-SDK/bsp/tcore-e203/env/"，在 Commands 中添加 "set arch riscv:rv32"，点击 Run 运行 gpio_rw 工程。

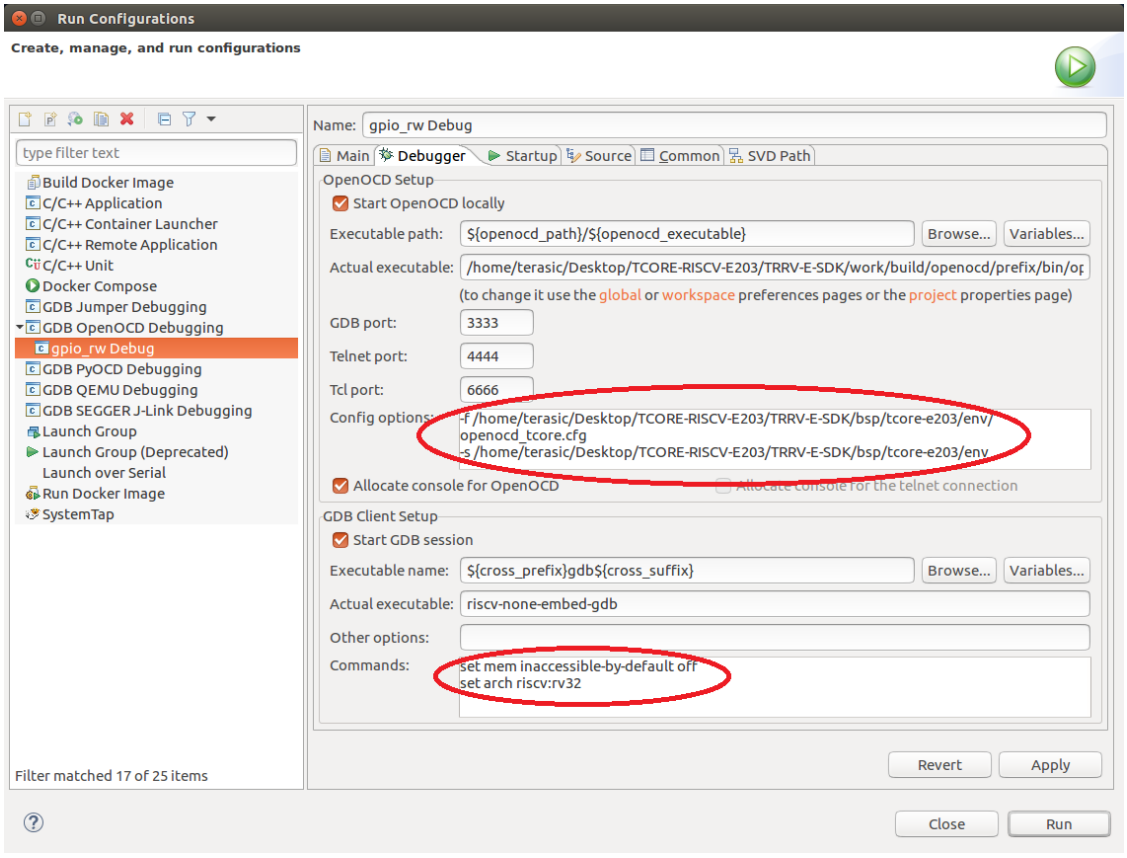

图4.3.16 运行配置

3. 程序下载成功后，如图 4.3.17 所示。

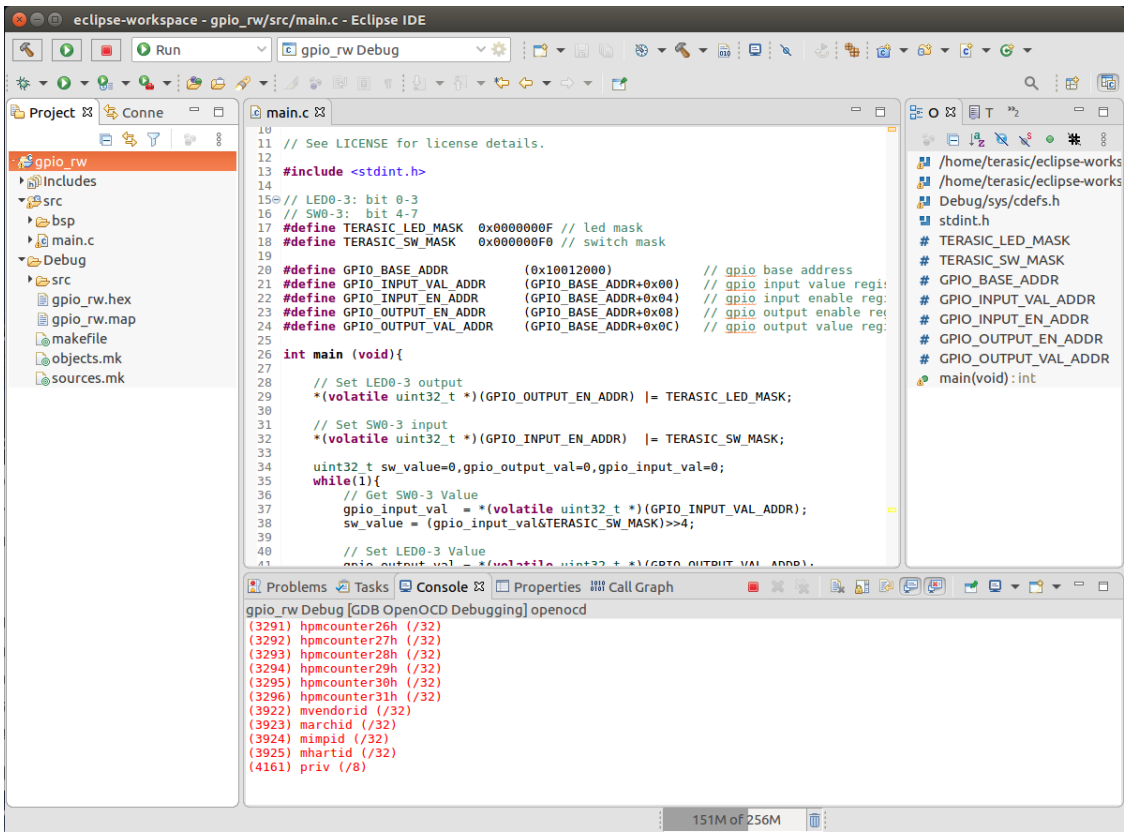

图4.3.17 运行 gpio_rw 工程

4.3.5 运行结果

程序下载完成后，先按 KEY0 键复位。当将 T-Core 开发板上拨码开关 SW 拨到向上的位置时，可以观察到对应位的 LED 点亮，当将 T-Core 开发板上拨码开关 SW 拨到向下的位置时，可以观察到对应位的 LED 熄灭。

您可以将 main.c 文件中的代码替换为 4.2.1 节中的 demo_gpio_rw2.c 文件中的代码并保存，再次执行 4.3.3 和 4.3.4 节中的操作，编译和运行 Hello World 工程，可以观察到同样的现象。

附录

1. 修订历史

版本	时间	修改记录
V1.0	2020.08.01	初始版本

2. 版权声明

3. 获取帮助

如遇到任何问题，可通过以下方式联系我们：

电话：027-87745390

地址：武汉市东湖新技术开发区金融港四路18号光谷汇金中心7C

网址：www.terasic.com.cn

邮箱：support@terasic.com.cn

微信公众号：

订阅号

服务号