实战演练之Timer中断

一、概述

本教程将介绍如何在 T-Core 开发板上实现 RISC-V 设计——Timer 中断：通过配置定时器中断相关寄存器，每隔 1s 打开定时器中断，控制 LED 实现保持点亮 1s，保持熄灭 1s 的效果。

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 软件创建 demo_timer 工程、修改 main.c 主函数、编译并运行 demo_timer 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；
- 了解发光二极管（LED）的工作原理及驱动方法；
- 学习 RISC-V CLINT 模块与定时器的结构和工作原理；
- 掌握 RISC-V 定时器中断的实现原理。

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-V1.2.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、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 3.1 为 T-Core 开发板 LED 灯和 FPGA 之间的连接示意图。

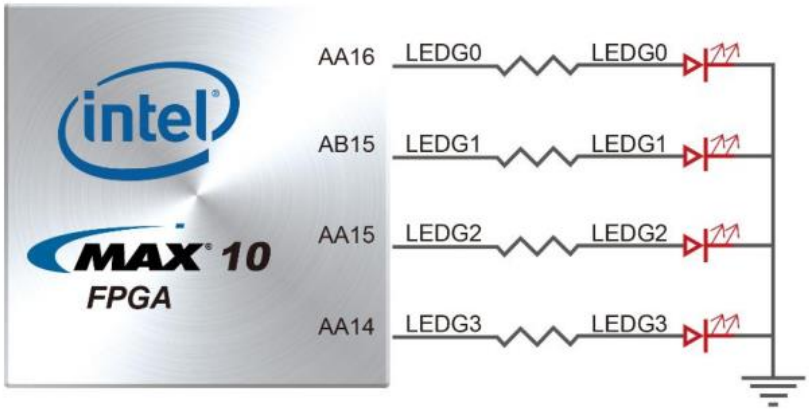

图3.1 T-Core 开发板 LED 灯和 FPGA 之间的连接

3.2 GPIO 寄存器列表

本教程主要用到以下 GPIO 寄存器：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。

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_OUTPUT_EN	0x008	0x0	Pin 的输出使能
GPIO_OUTPUT_VAL	0x00C	0x0	Pin 的输出值

3.3 GPIO 映射关系

关于 T-Core 的 RISC-V 处理器的 GPIO 与 T-Core 外设 LED 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

3.4 RTC

RTC 全称 Real-Time Clock，是 MCU 中常用的模块。RTC 通常在 MCU 系统中处于电源常开域，以固定不变的低速时钟频率运行，因此可以提供绝对精准的时间参考，所以被称为实时时钟。

表3.3 RTC 寄存器列表

寄存器名称	地址	复位默认值	描述
RTCCFG	0x1000_0040	0x0	RTC 配置寄存器
RTCLO	0x1000_0048	0x0	RTC 计数器计数值寄存器
RTCHO	0x1000_004C	0x0	RTC 计数器计数值寄存器
RTCS	0x1000_0050	0x0	RTC 计数器比较值寄存器
RTCCMP	0x1000_0060	0xFFFFFFFF	RTC 比较器寄存器

RTCCFG 寄存器可以用于对 RTC 进行配置，RTCCFG 寄存器的格式如图 3.2 所示。

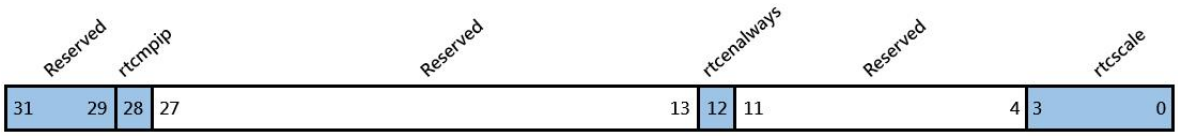

图3.2 RTCCFG 寄存器格式

表3.4 RTC 寄存器各比特域

域名	比特域	读写属性	复位默认值	描述
rtccmpip	28	只读	0x0	此域用于反映 RTC 产生的中断等待（Pending）状态
rtcenalways	12	可读可写	0x0	如果该域的值被配置为 1，则计数器永远进行计数
rtcscale	3:0	可读可写	0x0	该域用于指定从 RTC 计数器计数值中取出 32 位（作为 RTCS 寄存器的值）的低位起始位置

若 RTCCFG 寄存器 rtcscale 域的值为最大值 15，则意味着将 RTC 计数器计数值除以 2^{15} 作为 RTCS 寄存器的值，由于 RTC 处于常开域的时钟域，而常开域的低速时钟频率为 32.768 kHz，因此 RTCS 寄存器的值便是每一秒自增一，计算过程为：

$$(1/32768) \times 2^{15} = 1$$

3.5 CLINT 模块生成定时器中断

CLINT 的全称为处理器核局部中断控制器（Core Local Interrupts Controller），主要用于产生定时器中断（Timer Interrupt）和软件中断（Software Interrupt）。（本教程仅介绍定时器中断）

CLINT 是一个存储器地址映射的模块，寄存器的地址区间如表 3.1 所示。注意：CLINT 的寄存器只支持操作尺寸为 32 位的读写访问。

表3.5 CLINT 寄存器的存储器映射地址

地址	寄存器名称	复位默认值	功能描述
0x0200_4000	mtimecmp_lo	0xFFFFFFFF	配置定时器比较值的低32位
0x0200_4004	mtimecmp_hi	0xFFFFFFFF	配置定时器比较值的高32位
0x0200_BFF8	mtime_lo	0x00000000	反映定时器的低32位值
0x0200_BFFF	mtime_hi	0x00000000	反映定时器的高32位值

可通过 mtime 和 mtimecmp 寄存器生成定时器中断，其结构如图 3.3 所示。

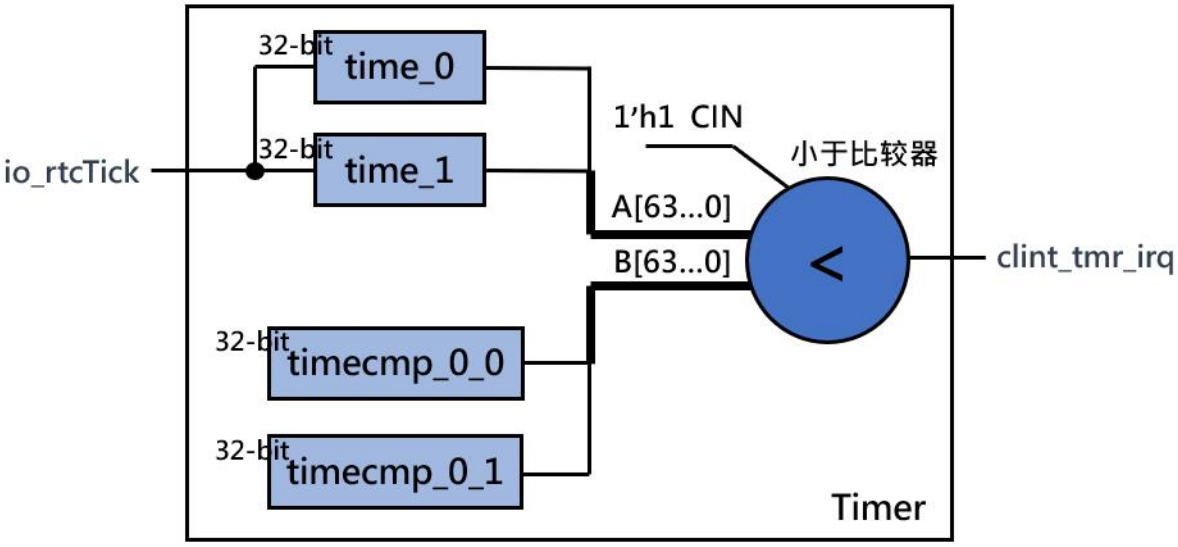

图3.3 定时器中断结构

- CLINT 中实现了一个 64 位的 mtime 寄存器，该寄存器反映了 64 位定时器的值。定时器根据低速的输入节拍信号进行计时，定时器默认是打开的，因此会一直进行计数。注意：由于 CLINT 的定时器上电后默认会一直进行计数，为了在某些特殊情况下关闭此定时器计数，可以通过自定义的 CSR 寄存器 mcounterstop 中的 TIMER 域进行控制。
- CLINT 中实现了一个 64 位的 mtimecmp 寄存器，该寄存器作为定时器的比较值。假设定时器的值 mtime 大于或者等于 mtimecmp 的值，则产生定时器中断。软件可以通过改写 mtimecmp 的值（使得其大于 mtime 的值）来清除定时器中断。注意：只有全局中断使能和定时器中断局部使能被打开后，才能响应此软件中断。

3.6 中断相关寄存器

1. **mie**

   RISC-V 架构定义了 CSR 寄存器机器模式中断使能寄存器 mie（Machine Interrupt Enable Registers），可以用于控制中断屏蔽。在本教程中，将介绍 mie 与定时器中断相关的部分。mie 寄存器的详细格式如图 3.4 所示，其中每一个比特域用于控制每个单独的中断使能。

图3.4 mie 寄存器格式

   在 mie 寄存器中，MTIE 域控制机器模式下定时器中断的屏蔽：

   - 当 MTIE 域被设置为 0，则意味着将定时器中断屏蔽，处理器无法响应定时器中断。
   - 当 MTIE 域被设置为 1，处理器可以响应定时器中断。

2. **mstatus**

mstatus 寄存器是机器模式下的状态寄存器。如图 3.5 所示，该寄存器包含若干不同的功能域，其中 MIE 域表示全局中断使能。

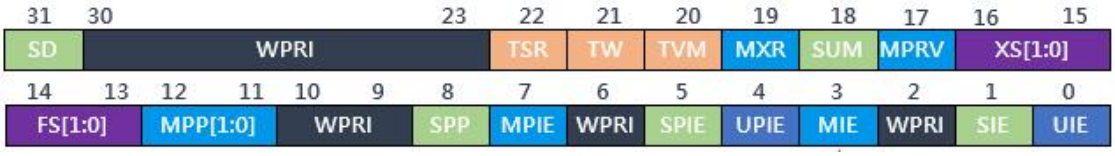

图3.5 mstatus 寄存器格式

- 当 MIE 域的值为 1 时，表示所有中断的全局开关打开。
- 当 MIE 域的值为 0 时，表示全局关闭所有的中断。

3.7 程序流程图

通过配置定时器中断相关寄存器，每隔 1s 打开定时器中断，控制 LED 实现保持点亮 1s，保持熄灭 1s 的效果。

在主程序中，程序开始时，先配置 LED GPIO 输出使能，并将默认输出值设为 0，再对定时器进行初始化，屏蔽定时器中断，设置 1s 的中断间隔，打开定时器中断和全局中断，接着使用 while(1) 语句使程序无限循环，如图 3.6 所示。

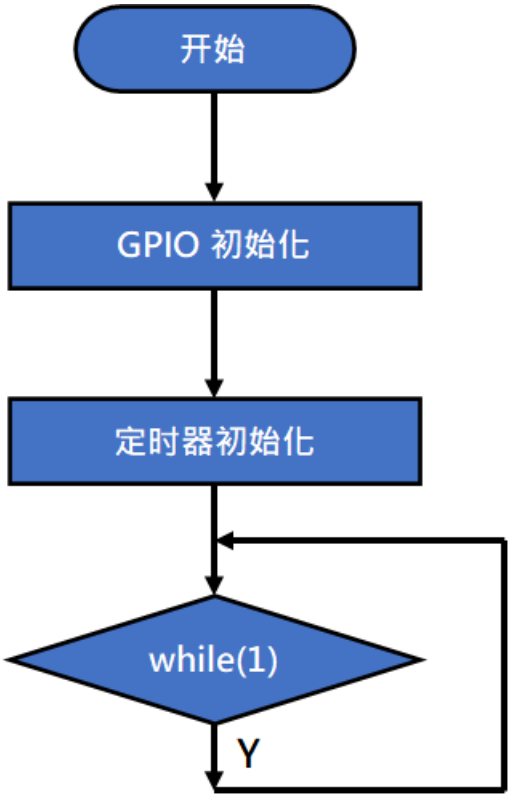

图3.6 Timer 中断主程序流程图

打开定时器中断后，响应中断程序，首先屏蔽定时器中断，然后重新设置 mtimecmp 寄存器的值，再使 LED 状态翻转，最后打开定时器中断。

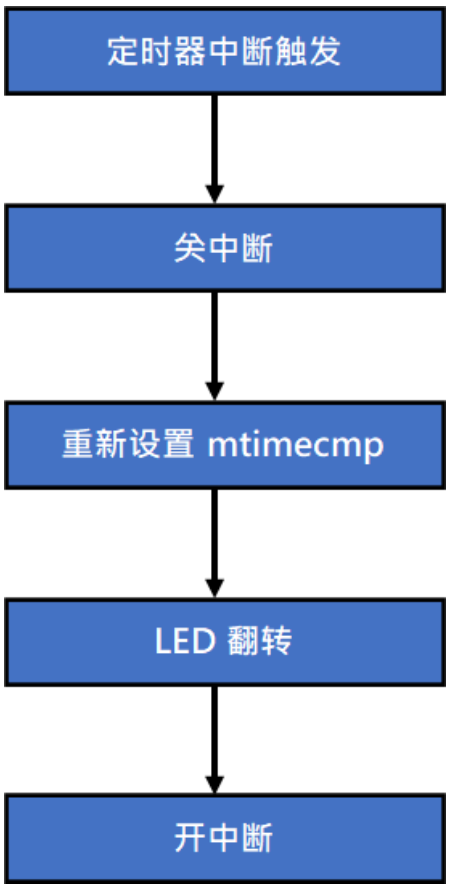

图3.7 Timer 中断响应程序流程图

四、操作步骤

4.1 使用 Makefile 编译和下载应用程序

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_timer" 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_timer"

2. 创建程序文件（.c文件）

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

   包含头文件，在头文件 platform.h 中，定义了一些常用的外设基地址，并包含了外设相关的头文件，在 plic_driver.h 中则定义了和中断相关的寄存器的基地址和偏移，在 encoding.h 中定义了 CSR 寄存器相关的寄存器和寄存器操作函数。

```
#include "platform.h"
#include "plic/plic_driver.h"
#include "encoding.h"
```

   定义 LED 掩码。

```
#define TERASIC_LED_MASK  0x0000000F        // led mask
```

定义定时器中断处理函数 handle_m_time_interrupt，首先将 mie 寄存器中与定时器中断相关的 MTIE 域清 0，屏蔽定时器中断，然后读取 mtime 寄存器的计数值，将 mtimecmp 的值设置为在当前计数值的基础上增加 "RTC_FREQ"，（这个值在 board.h 中被定义为 32768），由 3.4 节的讲解可知，即每隔 1s 打开定时器中断。再将 LED 对应的 GPIO 的值与 LED 掩码进行异或操作，即将 LED 的状态取反，在中断响应完成后，将 mie 的 MTIE 域设为 1，打开定时器中断。

```
// timer interrupt handler
void handle_m_time_interrupt(){
    // disable timer interrupt
    clear_csr(mie, MIE_MTIE);

    // Reset the timer for 1s in the future.
    // This also clears the existing timer interrupt.
    volatile uint64_t * mtime       = (uint64_t*) (CLINT_CTRL_ADDR + CLINT_MTIME);
    volatile uint64_t * mtimecmp    = (uint64_t*) (CLINT_CTRL_ADDR + CLINT_MTIMECMP);
    uint64_t now  = *mtime;
    uint64_t then = now + 1 * RTC_FREQ;
    *mtimecmp = then;

    // change led[3:0] value
    GPIO_REG(GPIO_OUTPUT_VAL) ^= (TERASIC_LED_MASK);

    // enable timer interrupt.
    set_csr(mie, MIE_MTIE);
}
```

定义定时器初始化函数 timer_init，前面的操作在定时器中断处理函数中已经做过介绍，在此不再赘述。

接着将 mstatus 的 MIE 域设为 1，表示将所有中断的全局开关打开。

```
void timer_init(){
    // disable timer interrupt
    clear_csr(mie, MIE_MTIE);

    // setup timer
    // set the machine timer to go off in 1 seconds.
    volatile uint64_t * mtime       = (uint64_t*) (CLINT_CTRL_ADDR + CLINT_MTIME);
    volatile uint64_t * mtimecmp    = (uint64_t*) (CLINT_CTRL_ADDR + CLINT_MTIMECMP);
    uint64_t now  = *mtime;
    uint64_t then = now + 1 * RTC_FREQ;
    *mtimecmp = then;

    // enable timer interrupt
    set_csr(mie, MIE_MTIE);

    // enable global interrupt
    set_csr(mstatus, MSTATUS_MIE);
}
```

定义 GPIO 的初始化函数，使能 LED0-3 对应的 GPIO 输出，并将 LED 默认输出值设为 0（即 LED 全灭）。

```
void gpio_init(){
    // set LED0-3 output
    GPIO_REG(GPIO_OUTPUT_EN) |= TERASIC_LED_MASK;

    // set led default value 0
    GPIO_REG(GPIO_OUTPUT_VAL) &=~TERASIC_LED_MASK;
}

```

最后定义程序的主函数，在 main 函数中调用 GPIO 初始化函数和定时器初始化函数，并使用 while(1) 语句使程序无限循环。

```
int main(int argc, char **argv){
    // initialize led
    gpio_init();

    // initialize timer
    timer_init();

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

3. 创建 **Makefile** 文件

在 "demo_timer" 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_timer
CFLAGS += -O1

BSP_BASE = ../../bsp

C_SRCS += demo_timer.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_timer"。

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_timer" 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_timer                         # 编译应用程序
```

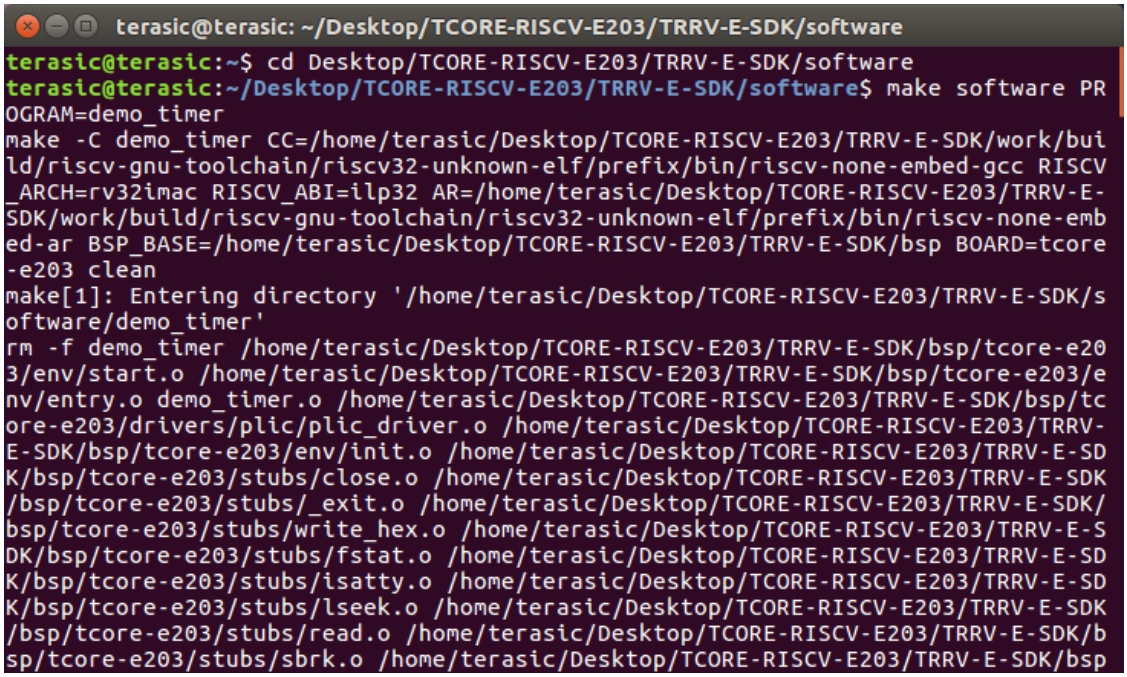

图4.1.1 编译应用程序

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

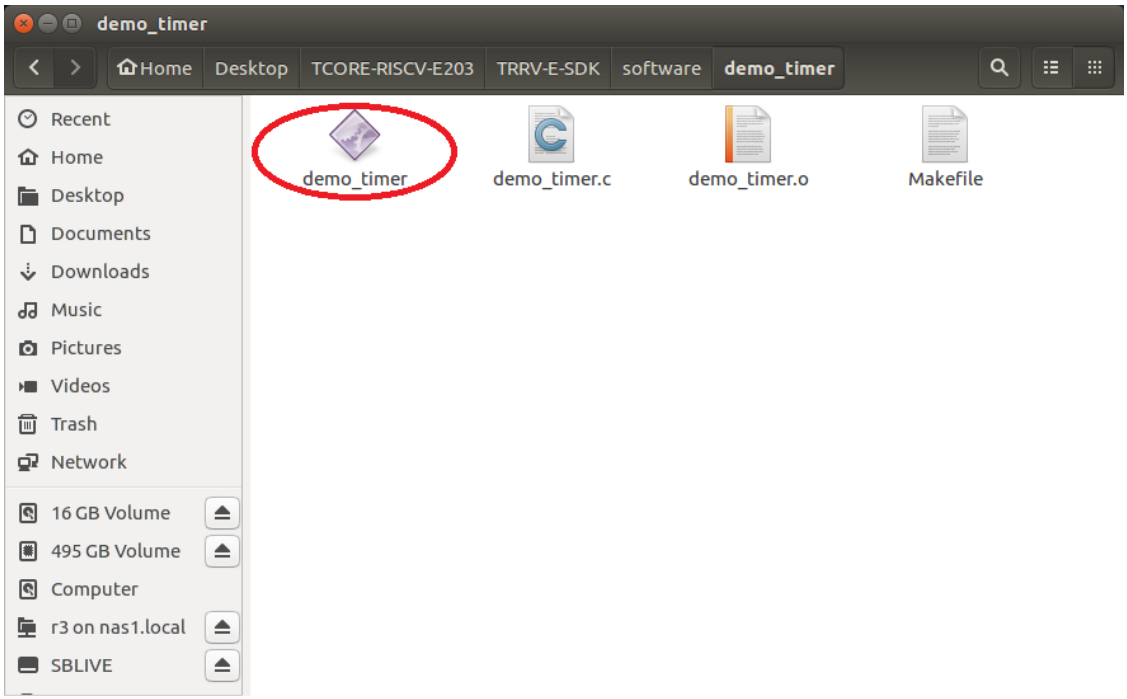

图4.1.2 编译生成二进制文件

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。

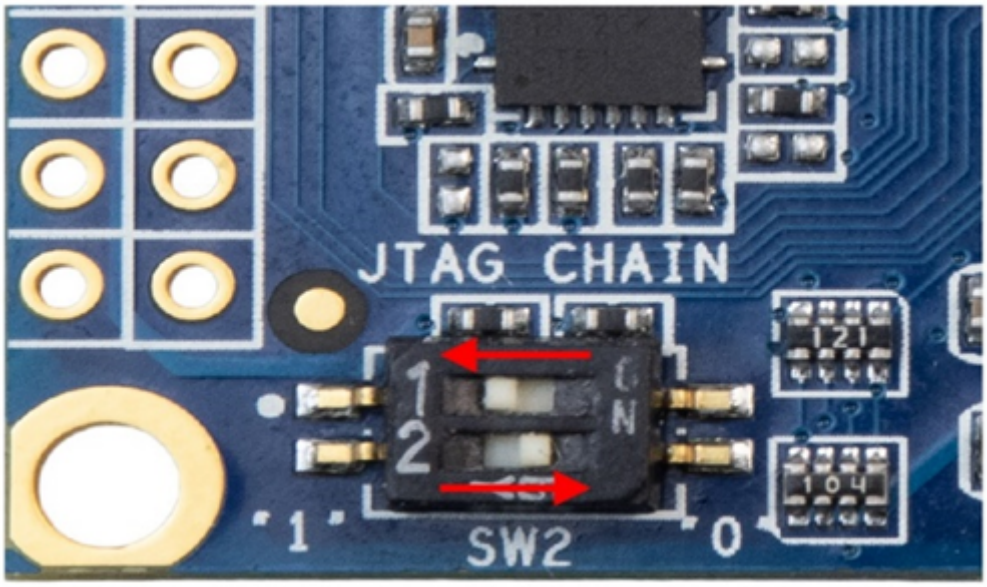

图4.1.3 设置 SW2 开关

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_timer" 将可执行文件 "demo_timer" 下载到 T-Core 开发板的 QSPI Flash 中。

```
make upload PROGRAM=demo_timer
```

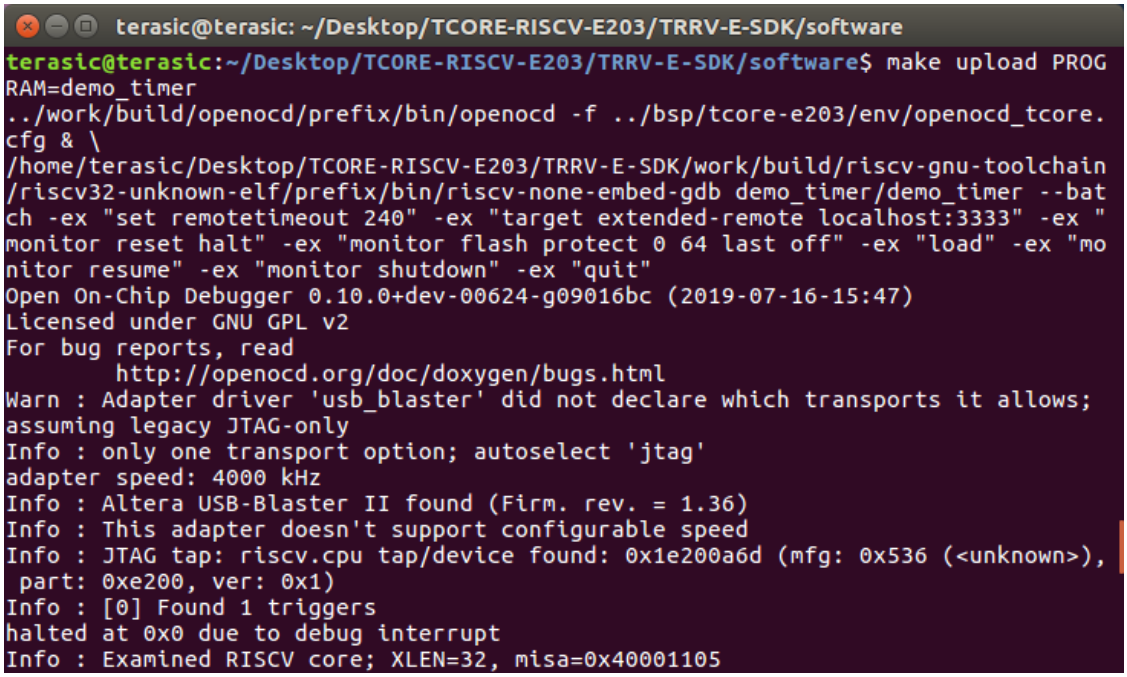

图4.1.5 下载可执行文件

4.1.4 运行结果

程序下载完成后，LED0-3 首先为熄灭状态，在 1s 后 LED 被点亮，1s 后再次熄灭，循环交替。

4.2 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）

4.2.1 创建 demo_timer 工程

1. 将文件夹命名由 "blinking_LED" 修改为 "demo_timer"，如图 4.2.1 所示。

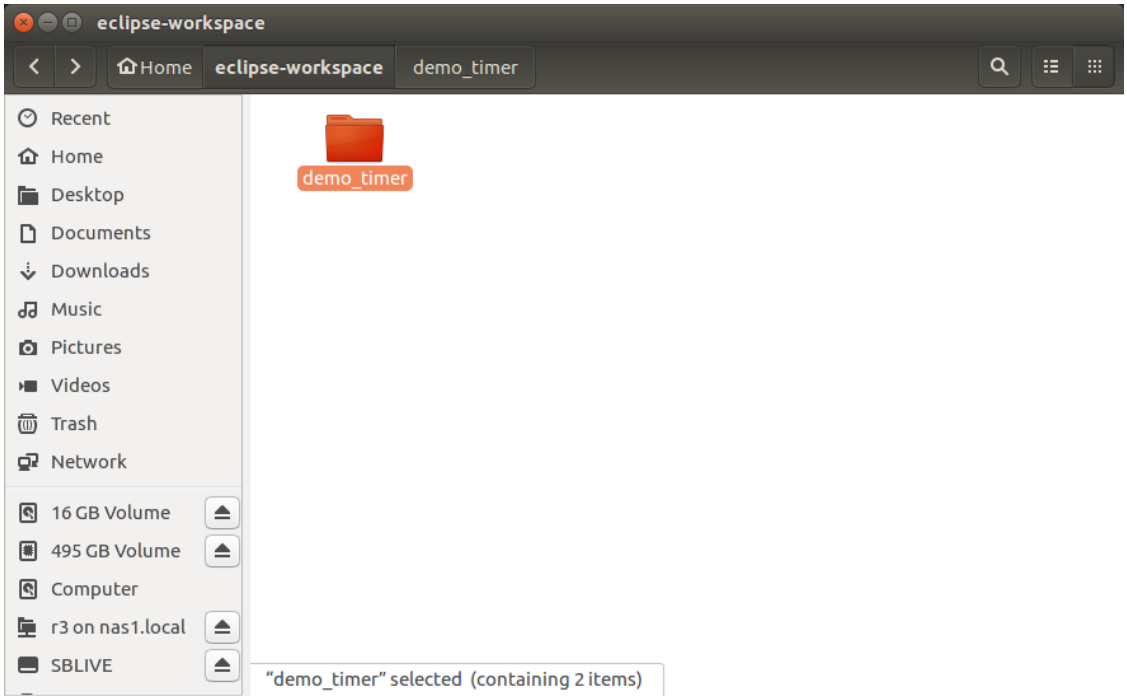

图4.2.1 修改文件名为 "demo_timer"

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。

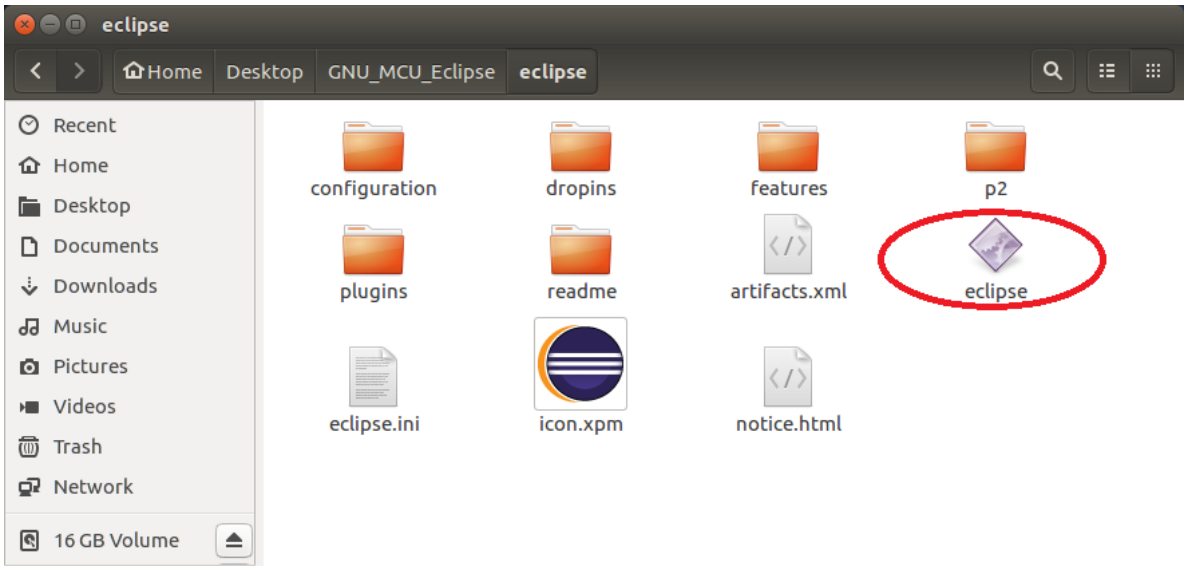

图4.2.2 启动 Eclipse

3. 启动 Eclipse 后，弹出设置 Workspace 的对话框，如图 4.2.3 所示，默认为 home 下的 eclipse-workspace（可根据需要自行设置）。

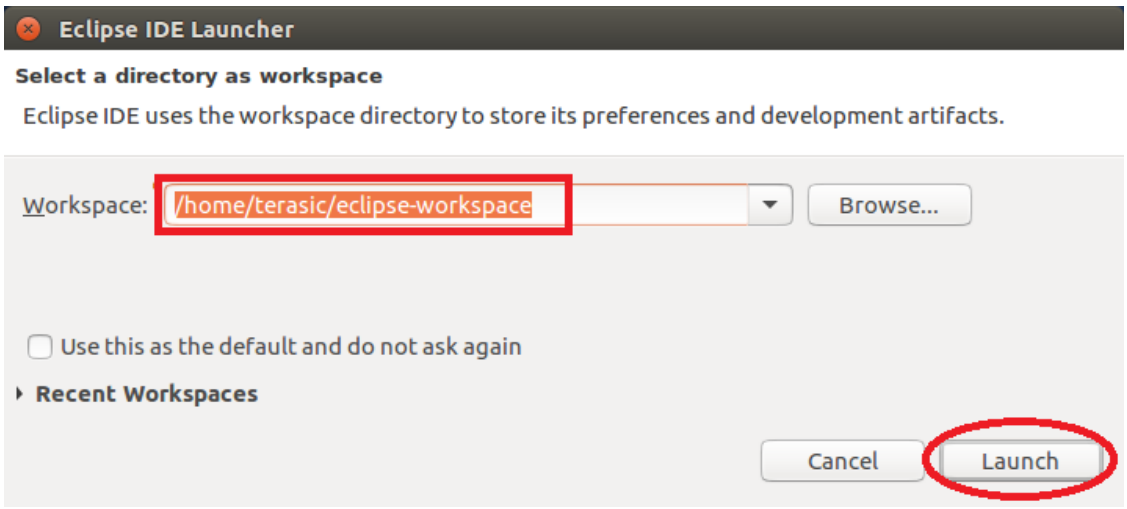

图4.2.3 设置 Workspace

4. 设置好 Workspace 目录后，单击 Launch，将会启动 Eclipse，进入 Welcome 界面，如图 4.2.4 所示。

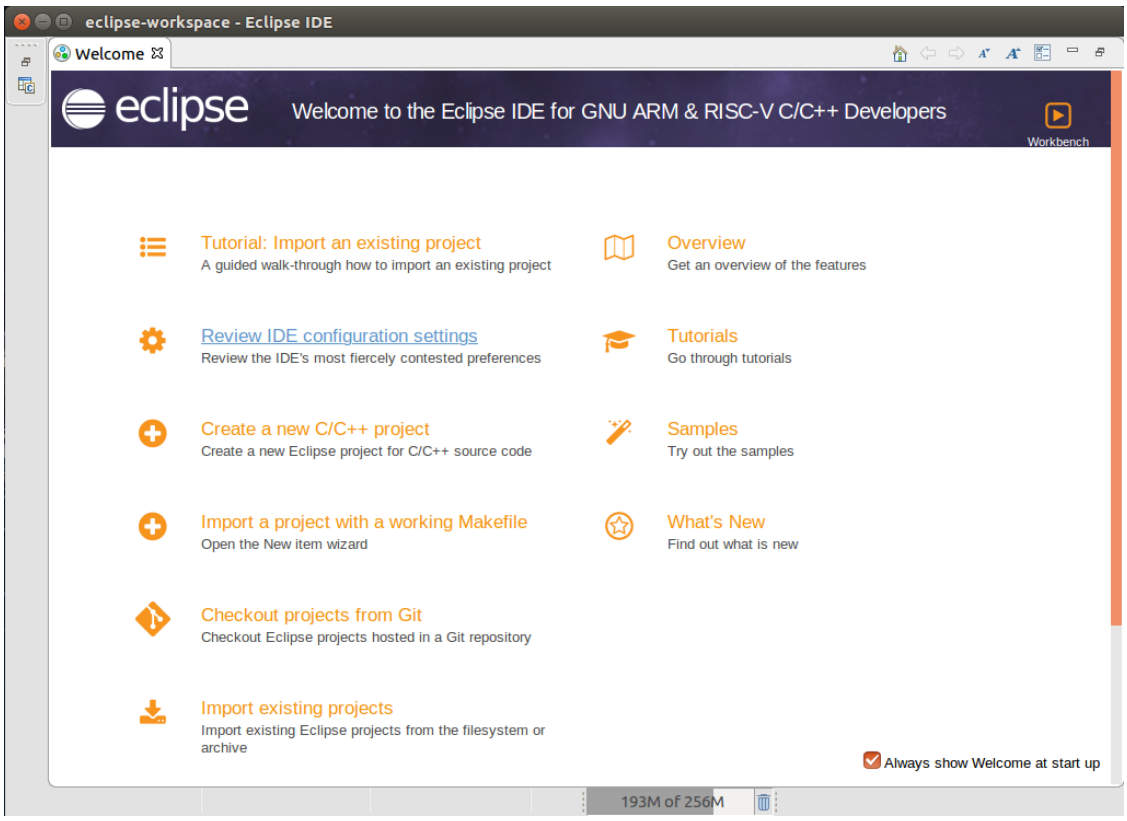

图4.2.4 进入 Eclipse 界面

5. 点击 Welcome 处的叉号，关闭 Welcome 界面，如图 4.2.5 所示。

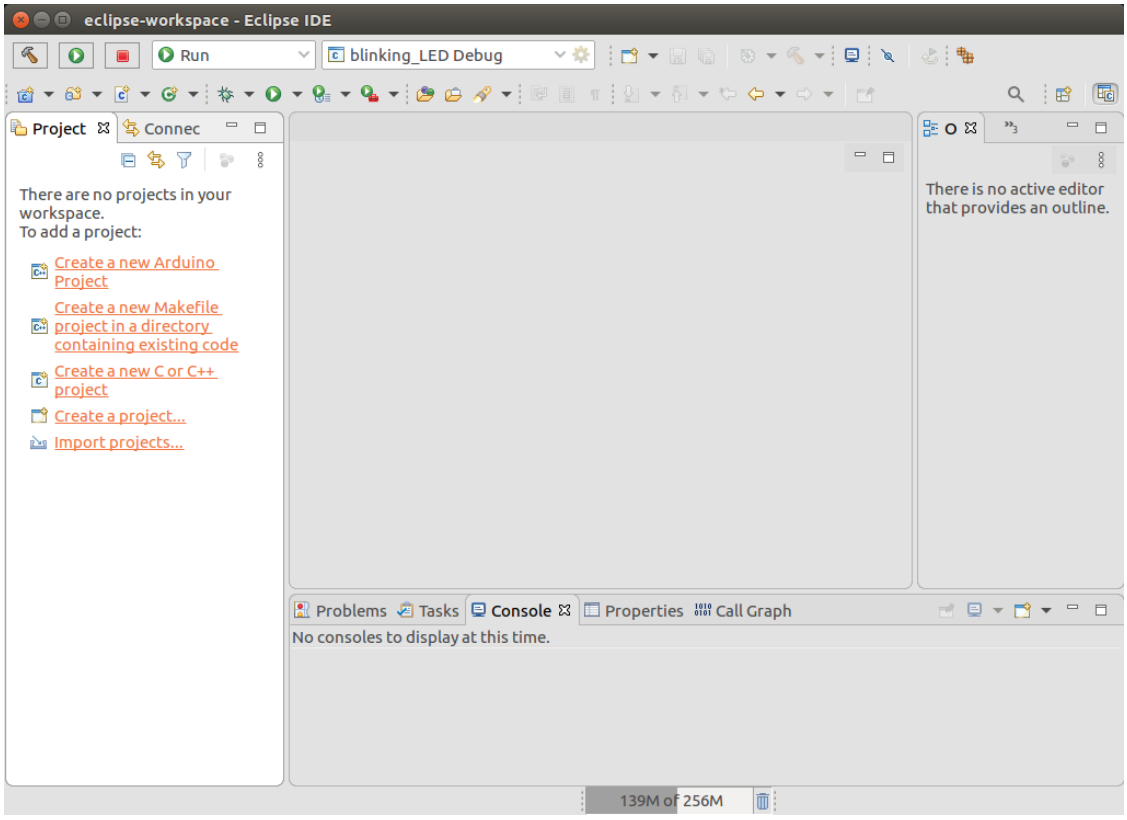

图4.2.5 进入 Eclipse 界面

6. 点击菜单栏 File -> Import... 导入工程，出现如图 4.2.6 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看到 File 选项）

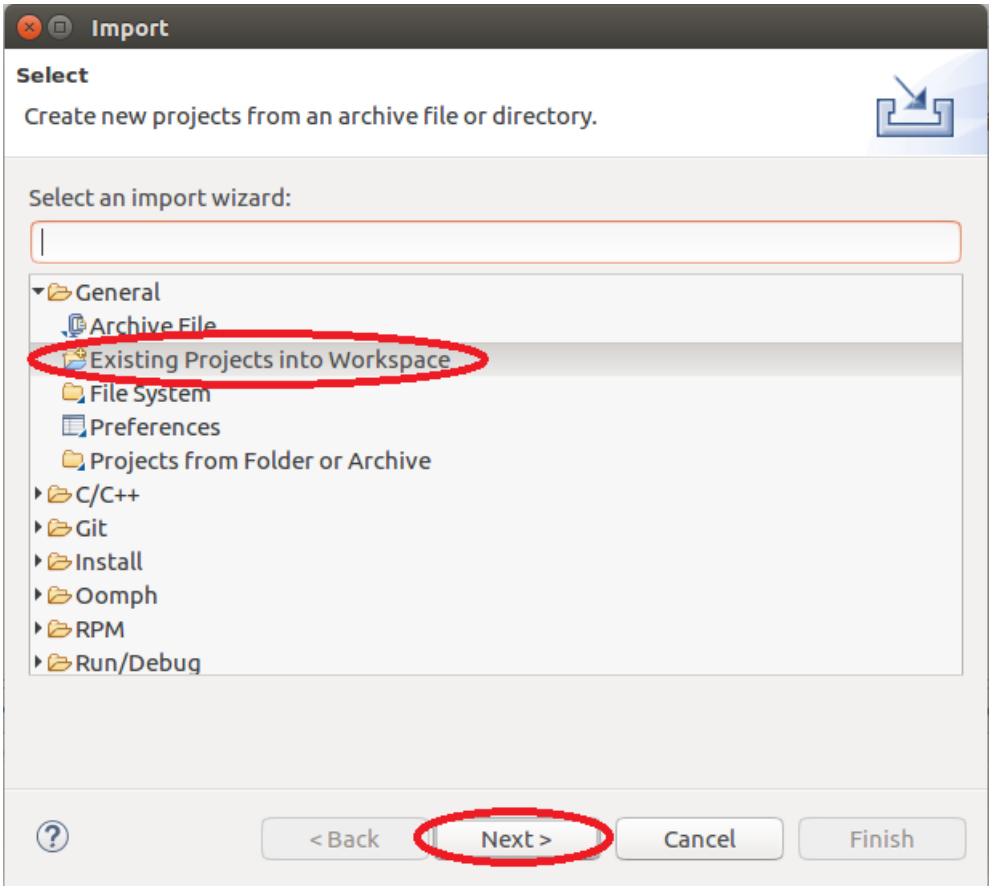

图4.2.6 选择导入工程类型

7. 点击 Browse 导入已有的工程。

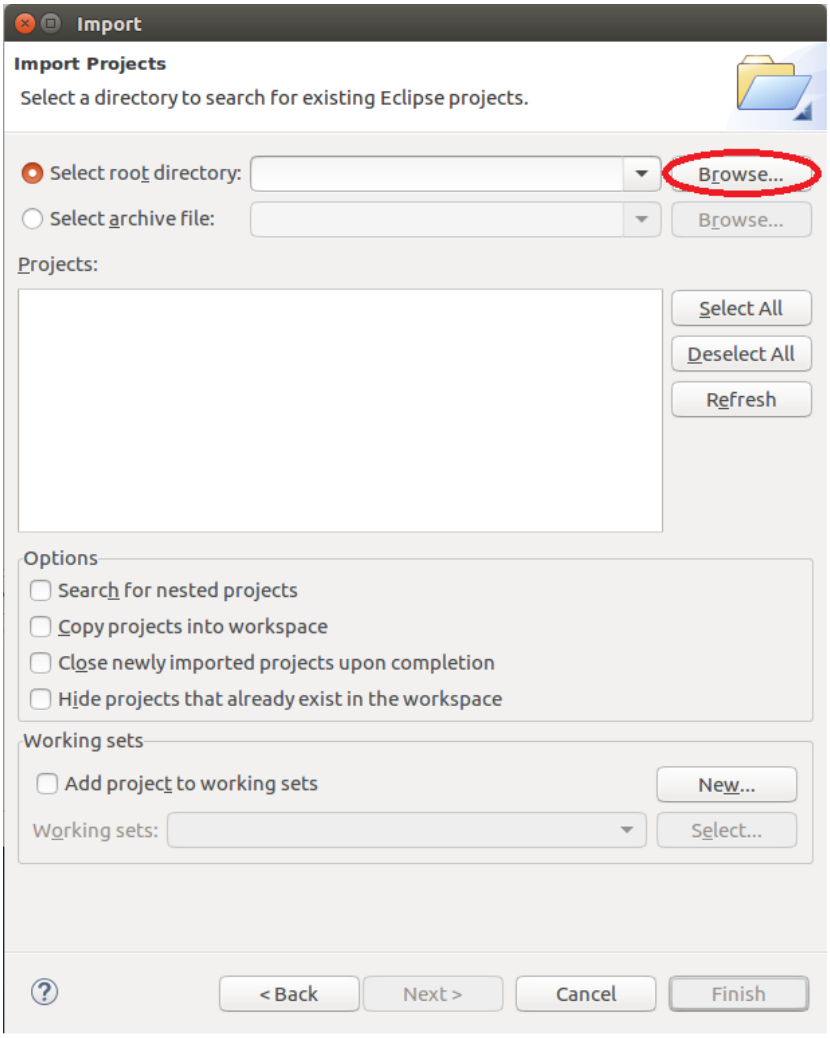

图4.2.7 点击 Browse

8. 选择要添加的 demo_timer 工程，点击 OK。

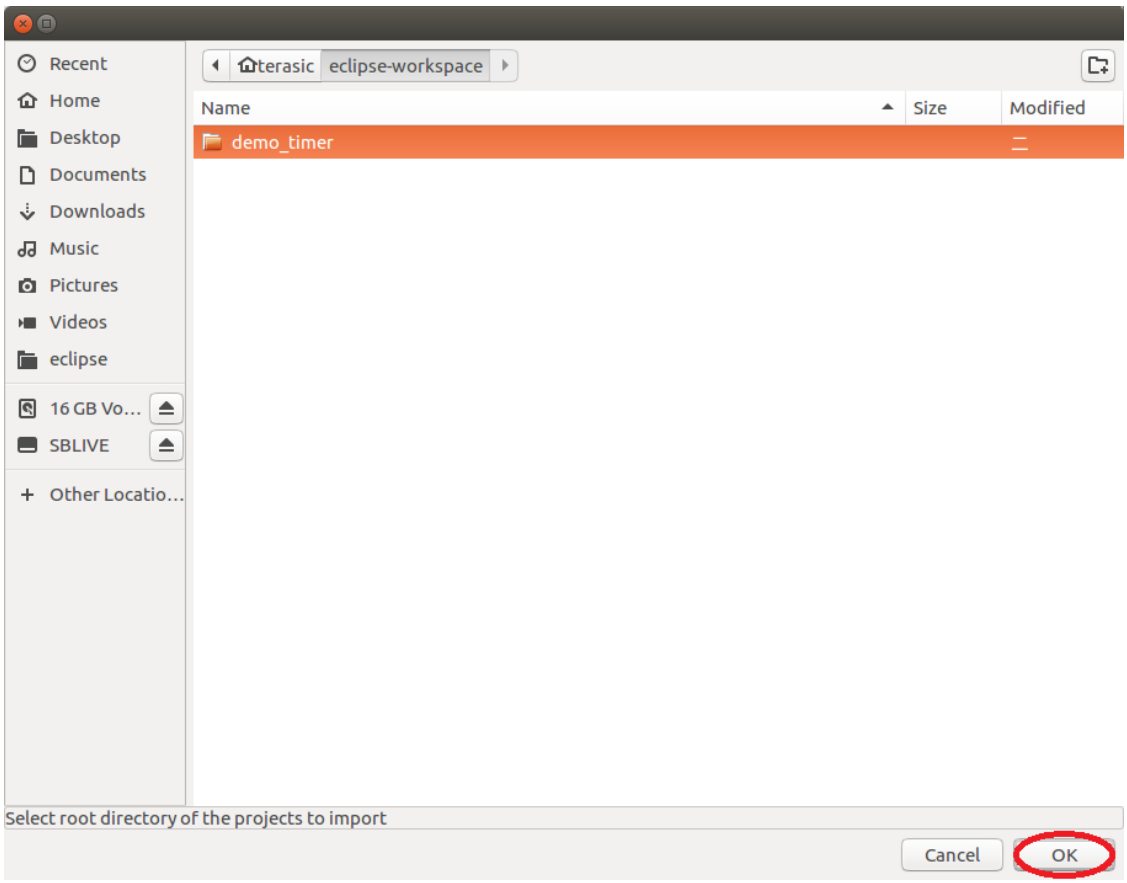

图4.2.8 添加 demo_timer 工程

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

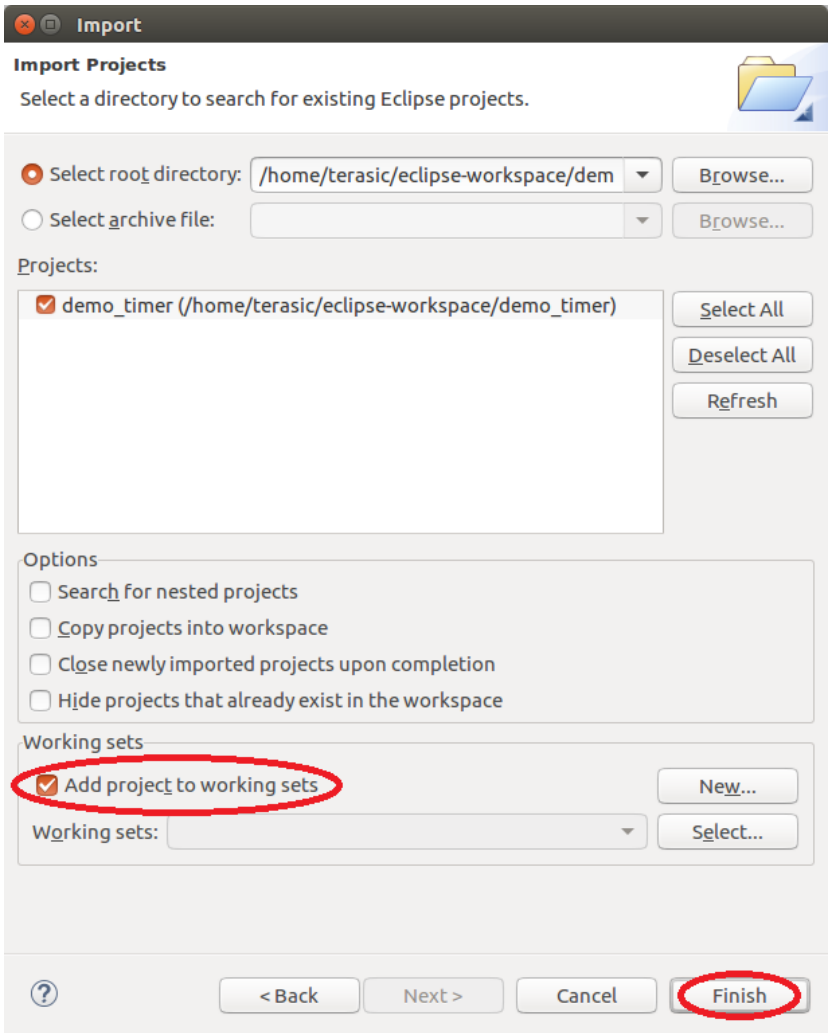

图4.2.9 添加 demo_timer 工程到工作空间

10. 导入后的工程界面如图 4.2.10 所示。

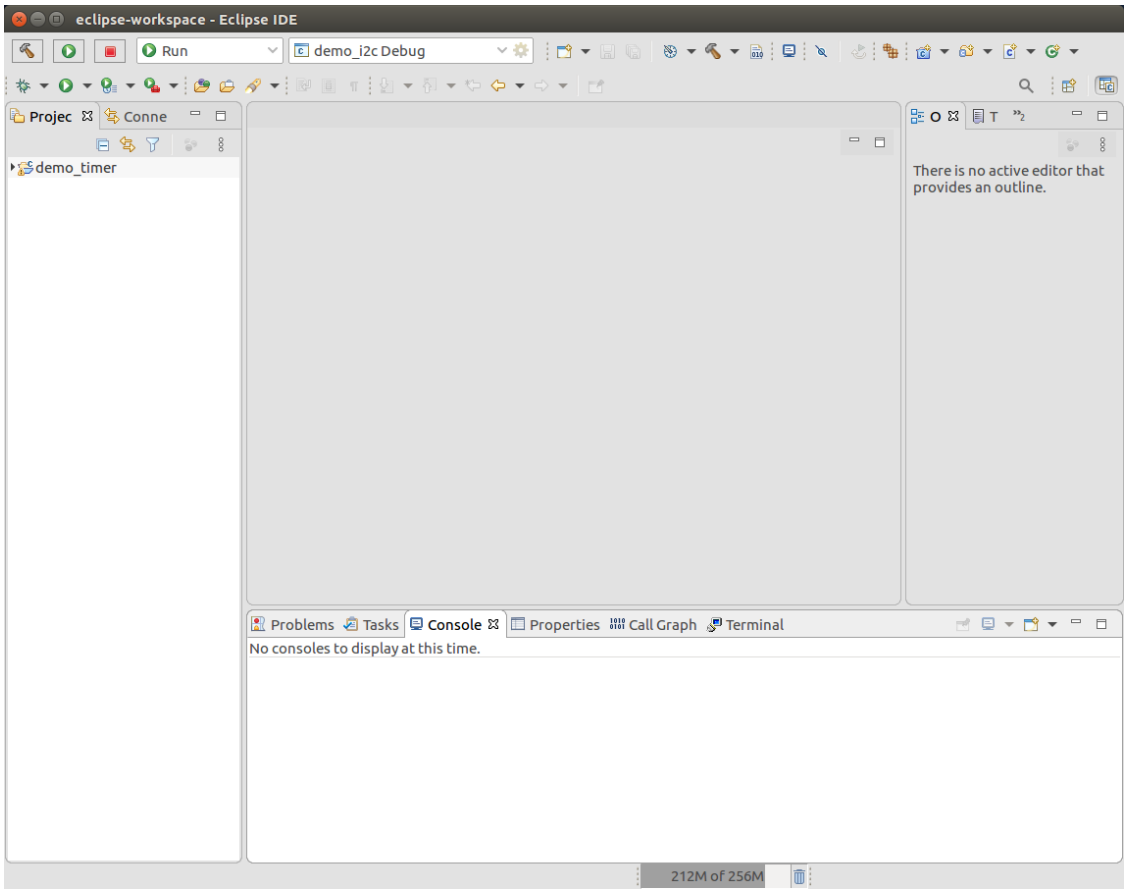

图4.2.10 导入后的工程界面

4.2.2 修改 main.c 文件

点击 demo_timer --> src 下拉框，双击打开 main.c 文件，复制 4.1.1 节中的 demo_timer.c 文件中的代码替换掉当前 "main.c" 的代码并保存，如图 4.2.11 所示。

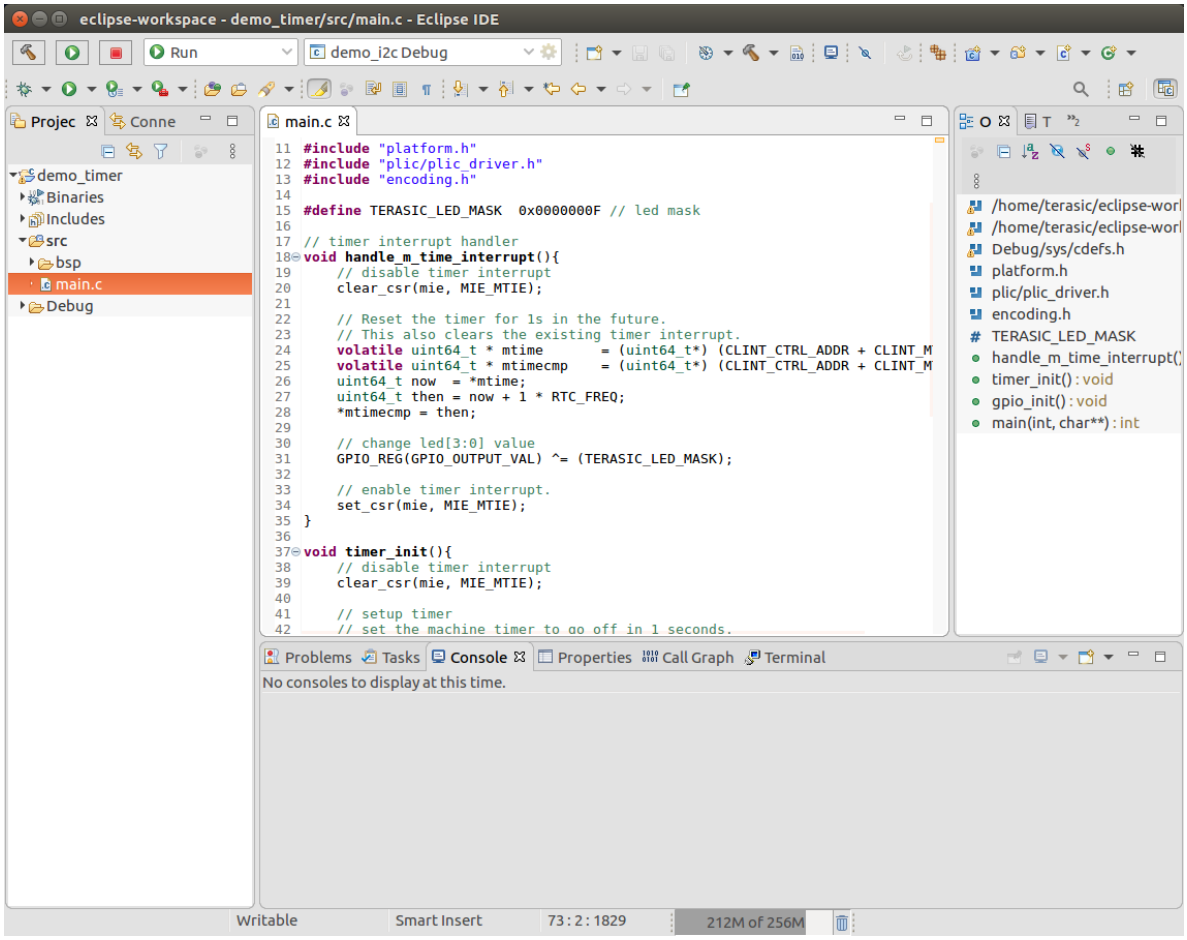

图4.2.11 修改 main.c 文件

4.2.3 编译 demo_timer 工程

1. 在 Eclipse 主界面中，选中 demo_timer 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 4.2.12 所示。

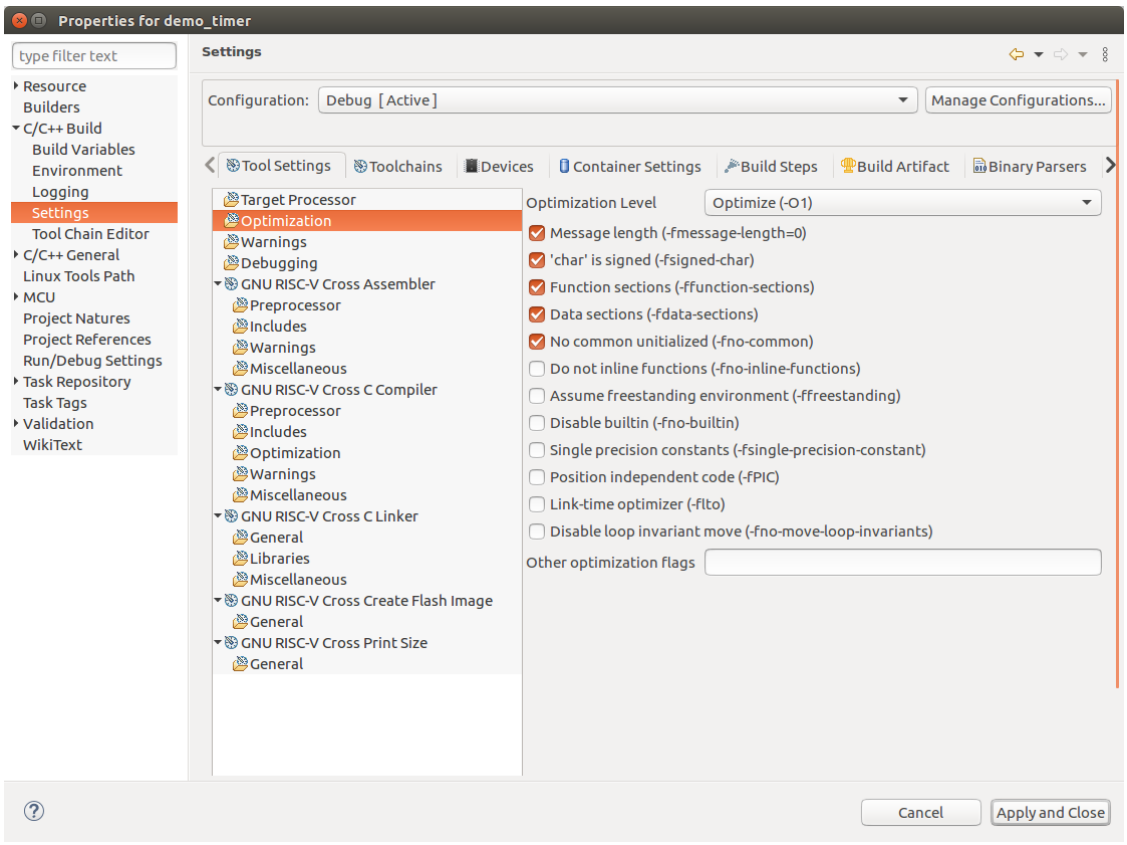

图4.2.12 修改 Optimization level

2. 选中 demo_timer 工程，右键点击 Clean Project；再次选中 demo_timer 工程，右键点击 Build Project，若 demo_timer 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 4.2.13 所示。需要右键点击 Refresh 在 Debug 下拉项中看到生成的 .elf 文件。

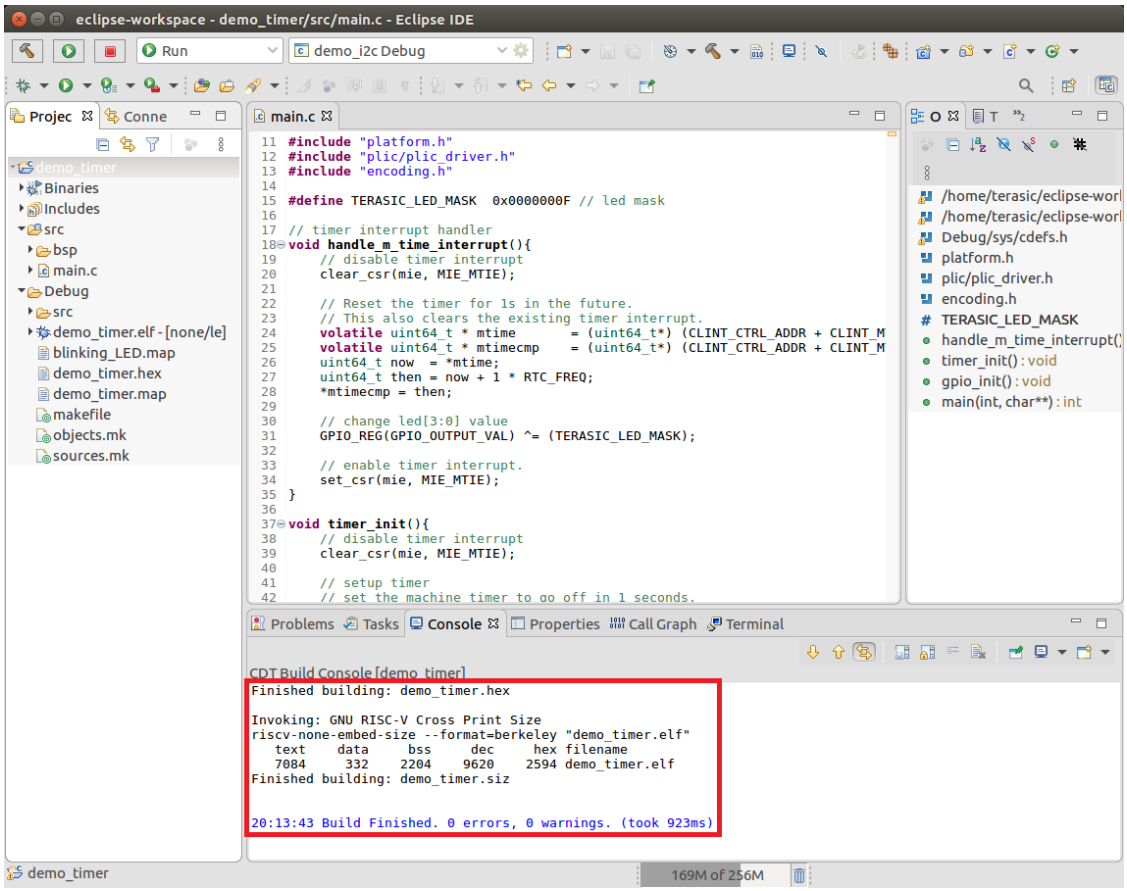

图4.2.13 编译成功

3. 右键 Debug 下拉选项中的 "blinking_LED.map"，点击 Delete，删除完成后如图 4.2.14 所示。

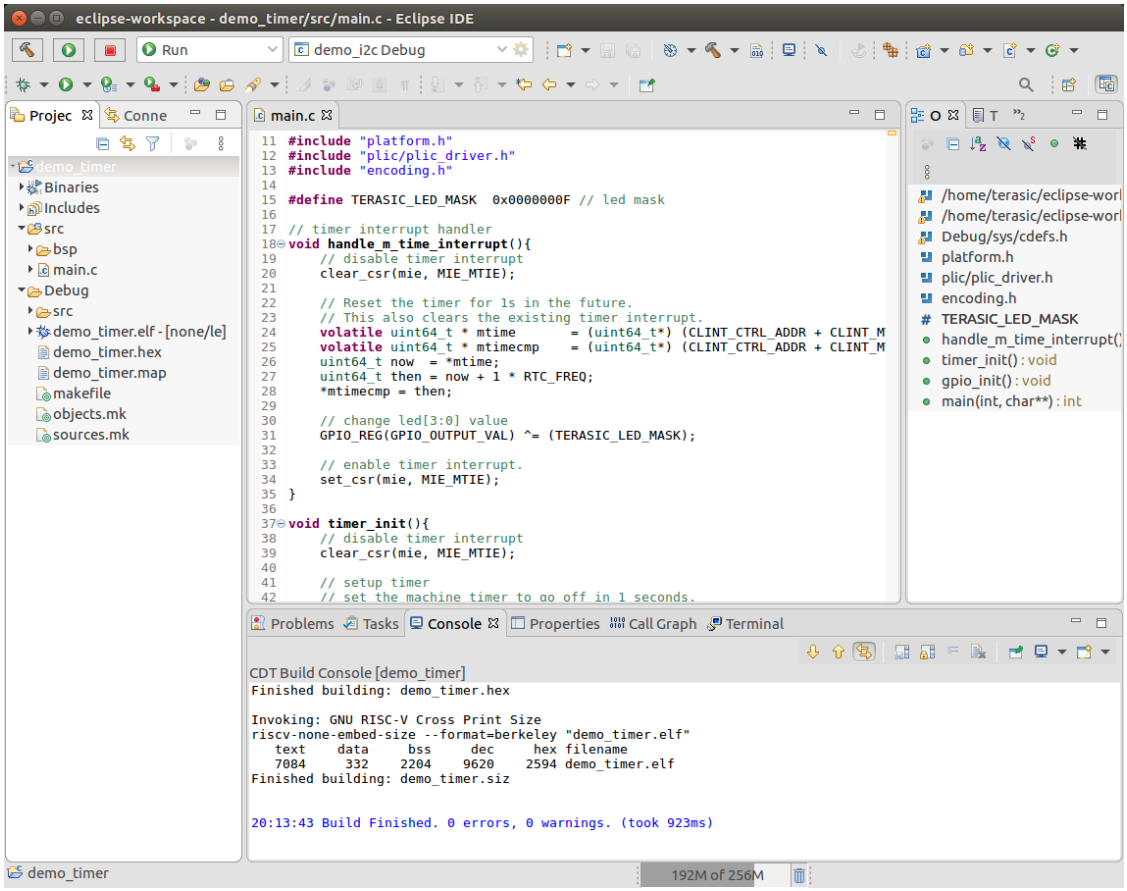

图4.2.14 删除 blinking_LED.map 文件

4.2.4 运行 demo_timer 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。

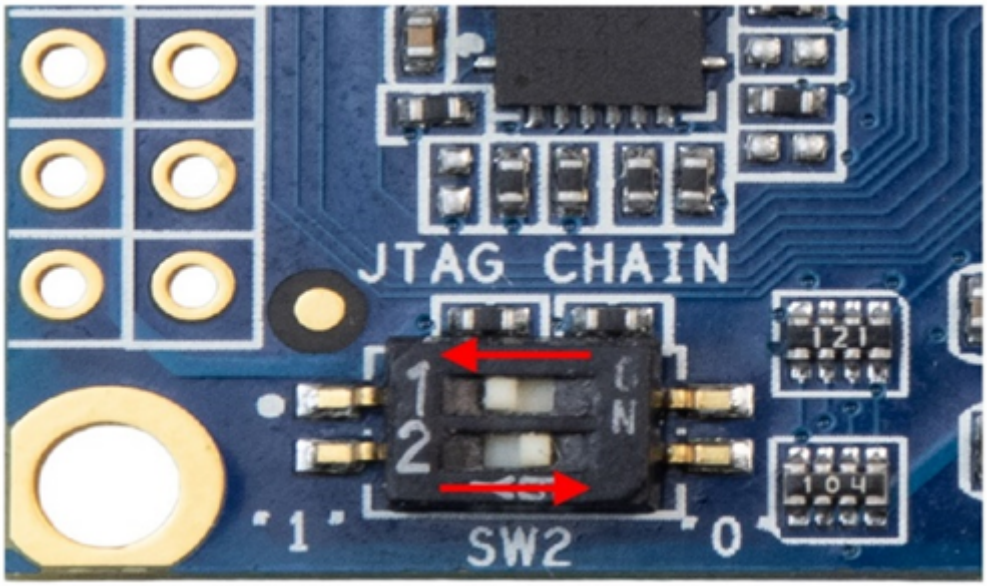

图4.2.15 设置 SW2 开关

   - 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 demo_timer 工程，右键点击 Run As -> Run Configurations...，双击 GDB OpenOCD Debugging 会出现如图 4.2.16 所示的 demo_timer 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 demo_timer 工程。

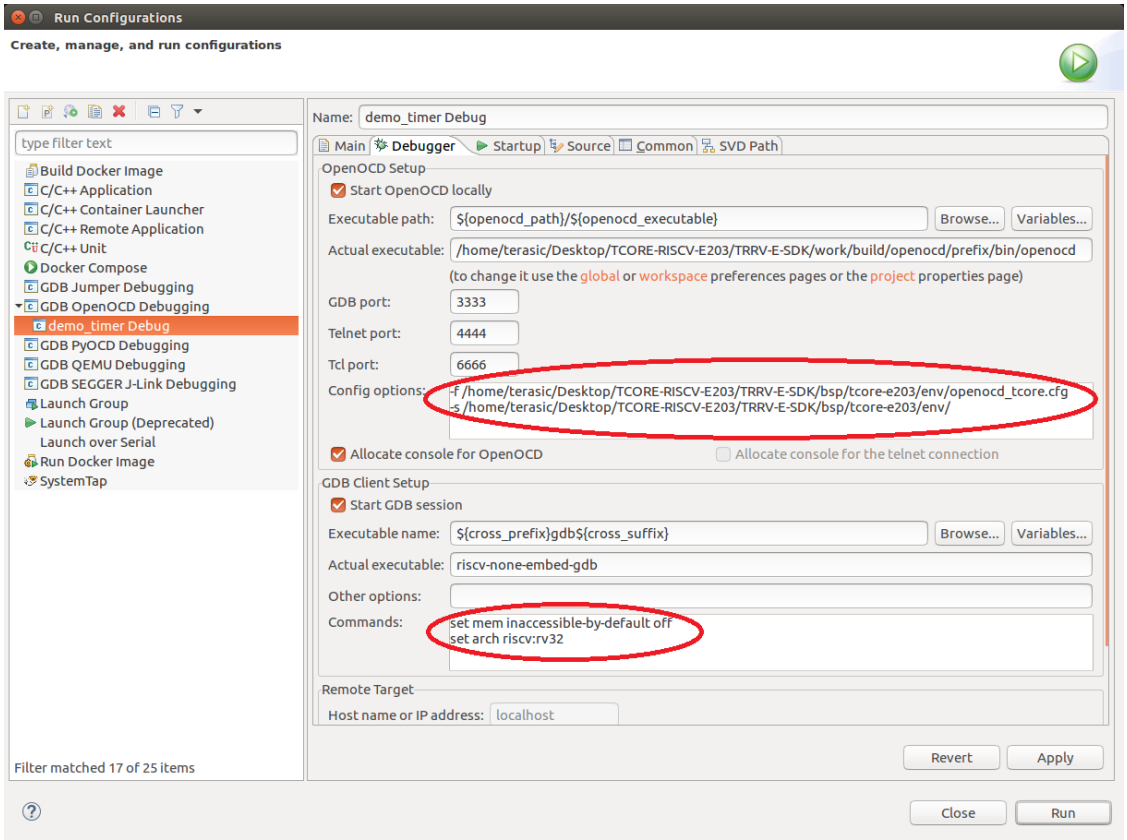

图4.2.16 运行配置

3. 程序下载成功后，如图 4.2.17 所示。

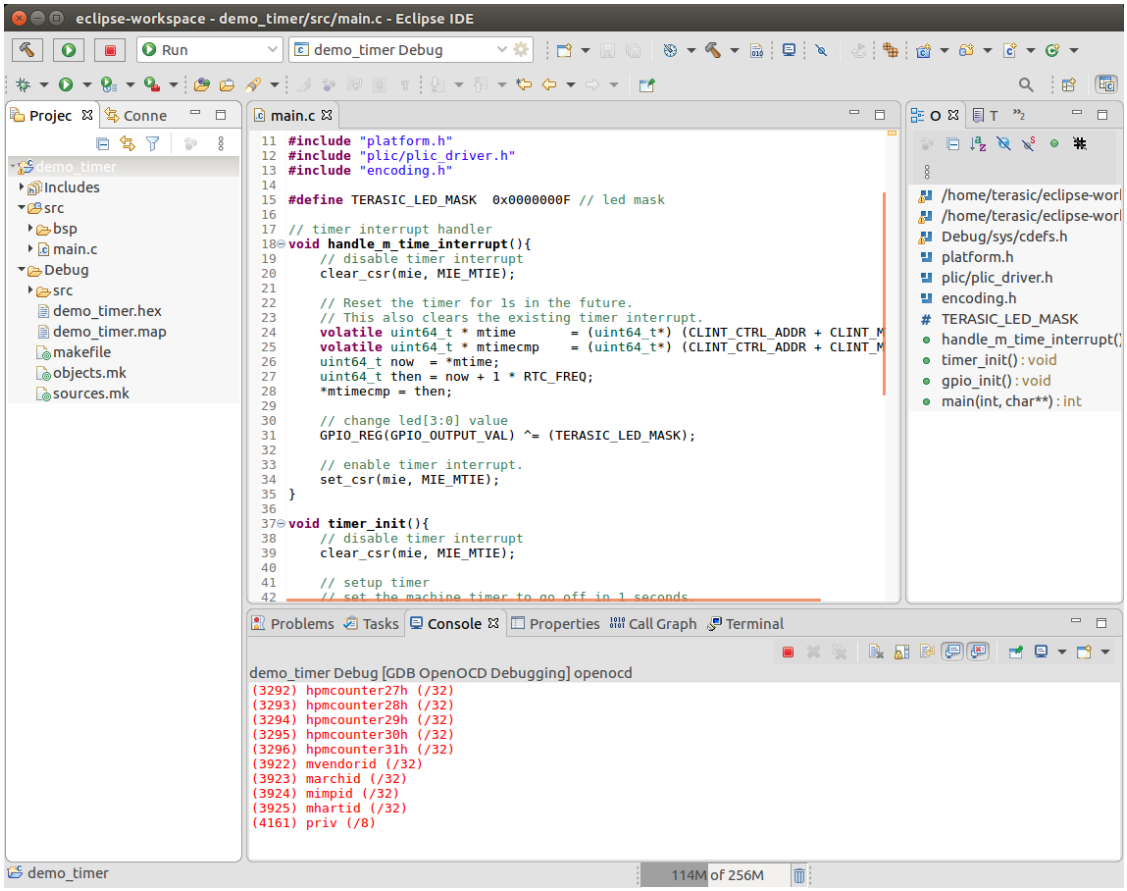

图4.3.17 运行 demo_timer 工程

4.2.5 运行结果

程序下载完成后，先按 KEY0 键复位。LED0-3 首先为熄灭状态，在 1s 后 LED 被点亮，1s 后再次熄灭，循环交替。

附录

1. 修订历史

版本	时间	修改记录
V1.0	2020.08.10	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