1 Hello,World 实验初试

工程源码	|-ACZ702 v2.0 开发板 |-- 无
相关视频课程	无

章节导读

这一章节中，我们将在 Ubuntu 上编写第一个实验代码——"hello，world"。这个实验不仅是学习 Linux C 编程的第一步，更是开启嵌入式 Linux 世界大门的钥匙。通过简单的"Hello, World"程序，将了解到如何在 Linux 环境下编写代码、编译程序以及运行程序。

1.1 前言

前面已经完成软件的安装和开发环境搭建，现在可以开始第一个实验“hello，world”。相信大家对此实验应该很熟悉了，在初学 C 语言中，一般第一个实验也是它。

```
#include <stdio.h>
int main() {
    printf("Hello, World\n");
    return 0;
}
```

但是在 Linux 中，代码的编写、编译、甚至框架与 windows 平台都有所不同；简单的看向下列代码，与上面不同的是 main 里面添加了参数（让程序接收命令行参数），这也是本小节重点之一。

```
#include <stdio.h>
int main(int argc, char *argv[]) {
    printf("Hello, World\n");
    return 0;
}
```

为了方便大家更容易理解 Linux C 代码中参数的应用，将上面代码修改成如下，让大家更加容易理解参数的传递:

```
#include <stdio.h>
int main(int argc, char *argv[]) {
```

```
    int i;
    printf("This program has received %d arguments:\n", argc);
    for(i = 0; i < argc; i++) {
        printf("argument %d: %s\n", i, argv[i]);
    }
    return 0;
}
```

1.1 远程连接 Ubuntu

现在打开 VMware 中的 Ubuntu 虚拟机，使用 mobaxterm 软件远程连接，下面所有的操作全程在 mobaxterm 软件中进行。(如果没有安装的，回到软件安装章节)

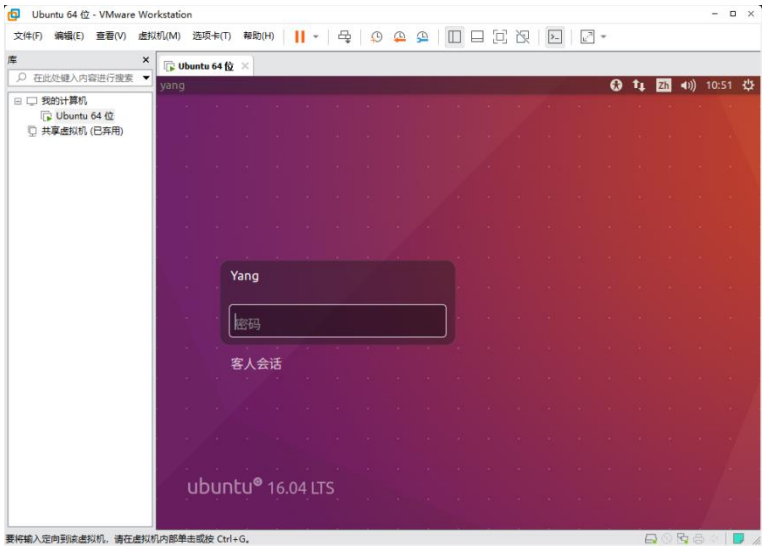

图 1-1 启动 Ubuntu

如果全程跟随教程没有遗漏的话，那么在 mobaxterm 软件界面右侧双击 Ubuntu，即可远程连接虚拟机。

图 1-2 连接 Ubuntu

1.2 代码编写流程

1.2.1 创建工程文件夹

在终端界面中，输入命令“cd FPGA_Linux/ZYNQ_Prj/”，进入 ZYNQ_Prj 目录下；继续输入命令“mkdir 1_hello_world”，创建工程文件夹。

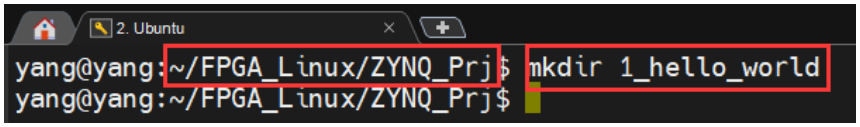

图 1-3 创建工程文件夹

继续输入命令“cd 1_hello_world”，进入 1_hello_world 文件夹下。

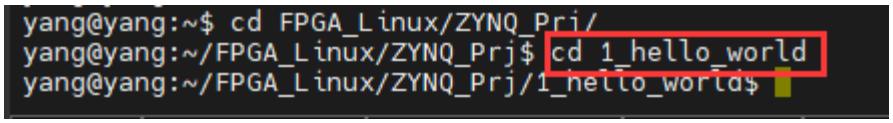

图 1-4 进入 1_hello_world 文件夹

1.2.2 创建源文件

输入命令“vim hello_world.c”，创建.c 文件。

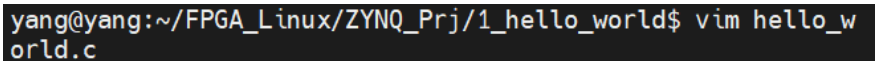

图 1-5 输入 vim 命令

1.2.3 编辑代码

此时，会进入一个新界面，如果你现在输入代码，它是不会写入文件，需要先进入编辑模式；在英文输入法下，输入字母‘i’，就可以继续敲键盘了，我们将代码第一行“#include <stdio.h>”输入其中，如图 1-7 所示。

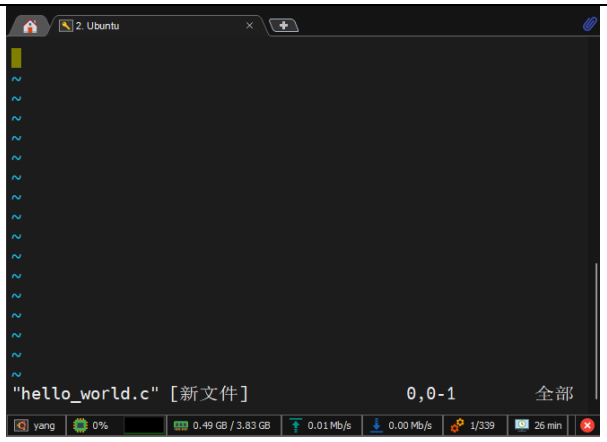

图 1-6vim 打开文件

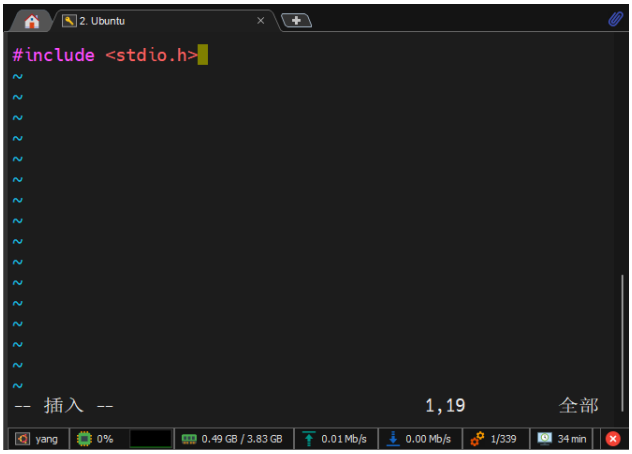

图 1-7 进入 vim 编辑模式

继续将下面代码补充完整，建议自己敲代码，不要粘贴复制，代码虽然简单，但是自己手动敲代码也是熟悉 Linux 平台的一个过程。

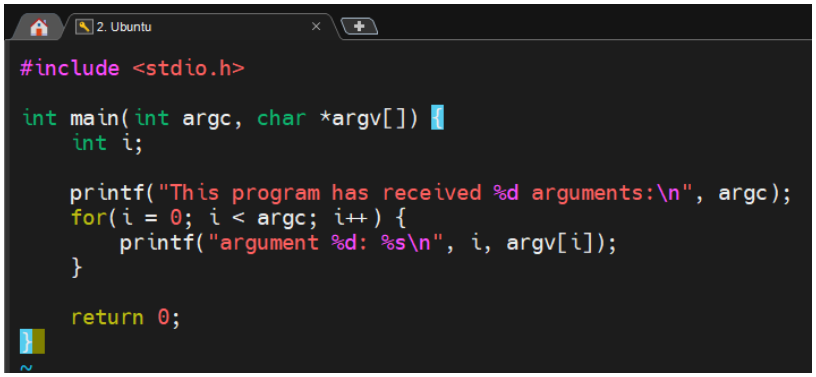

图 1-8 添加代码

1.2.4 添加上行号

现在手动敲写的代码量不大，只有十几行，所以看上去不会显着凌乱；但是当我们的代码量达到几十行，甚至几百行时；这样看着会非常痛苦，这时就

需要添加上行号，方便浏览阅读。

先在键盘上按下 ESC，然后输入命令“:set number”，如图 1-9 所示

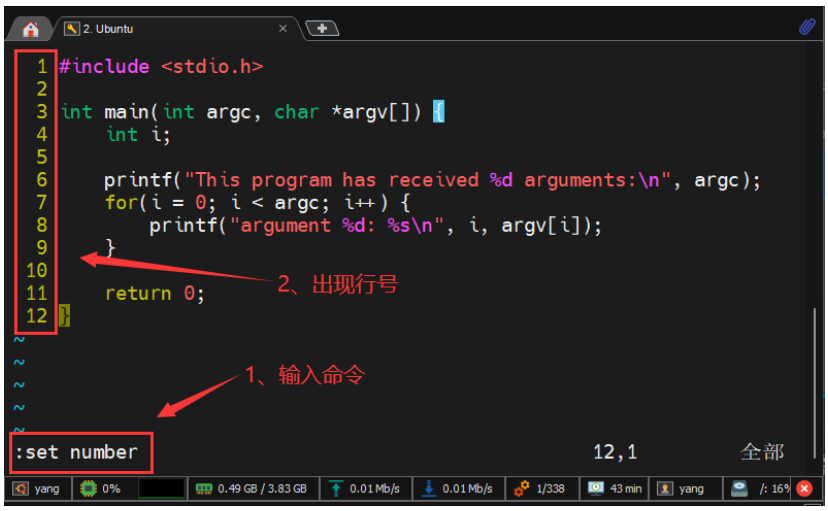

图 1-9 添加行号

1.2.5 退出 VIM

继续按下 ESC，然后输入命令“:wq”，保存文档并退出。

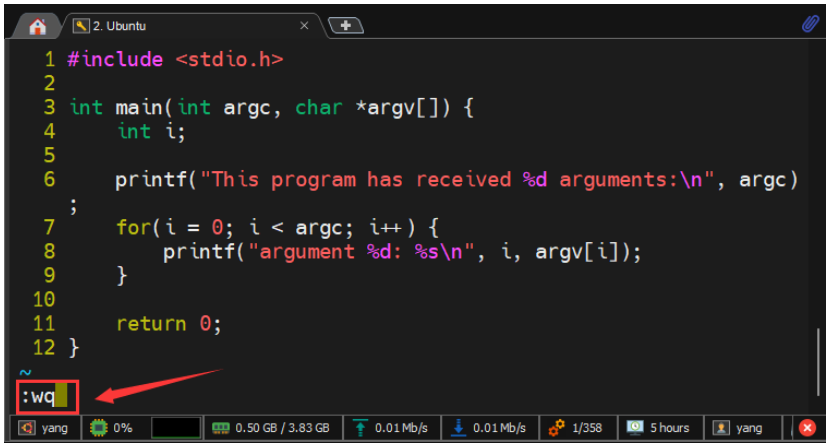

图 1-10 保存并退出

1.2.6 更换编辑器（不推荐）

大部分人长期使用 windows 平台上的编辑器，可能在使用 vim 的过程中难受，以及适应不了，那么可以尝试更换其他编辑器（再次说明强烈不推荐，只建议过渡用下，后期还是 VIM 操作）；在终端界面中输入命令“gedit hello_world.c”；该命令的作用是，用 gedit 软件打开 hello_world.c 文件。

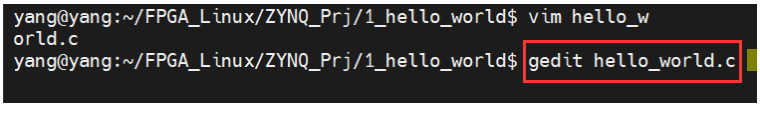

图 1-11 启动 gedit 编辑器

此时会出现新的编辑器界面，其中交互操作与 windows 平台上的编辑器基本一致。

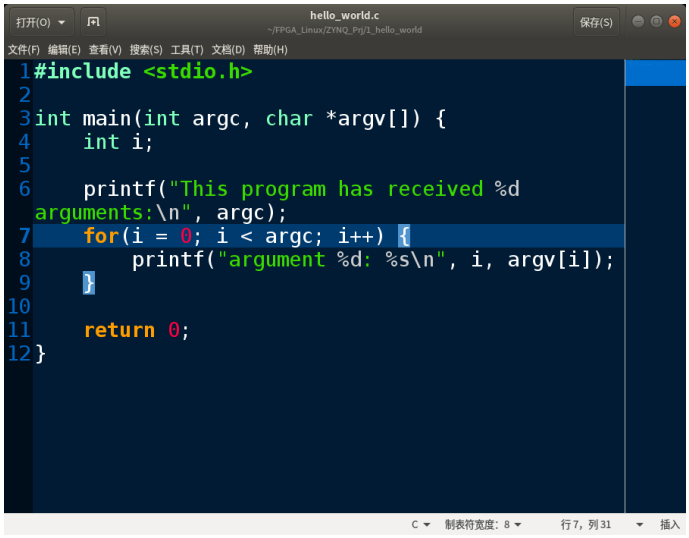

图 1-12 编辑器界面

支持编辑器配置，可以在首选项中，配置界面背景、文字大小与颜色，行号和概览图等等。

图 1-13 编辑器配置

1.3 代码编译流程

在 Windows 平台上，有许多 IDE（集成开发环境）如 Visual Studio 2019、Eclipse 等等，提供了便捷的"一键编译"按钮，使得编译过程变得相对直观和简洁。而在 Linux 平台上，通常更多是采用“手工”方式，使用更加底层和灵活的命令行工具，如 gcc 编译器，make 等等，来完成编译链接。

1.3.1 生成可执行文件

在终端界面中，输入命令“gcc hello_world.c -o hello_world”，如图所示。

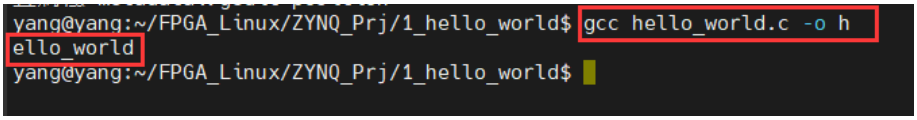

图 1-14gcc 命令

该条命令作用为：

1. gcc: 代表了你正在使用 GCC 编译器。

2. hello_world.c：代表要被编译的 C 语言源文件。

3. -o：命令行选项，代表"输出"的意思。在其后你需要跟上输出文件的名称。

4. hello_world：这是输出文件的名称，这条命令会把编译结果输出为这个文件名。（可以自由指定自己需要的名称）

然后在终端界面中输入命令“ls”，显示所有文件；此时在界面中，可以看到出现新的文件，即应用文件“hello_world”。

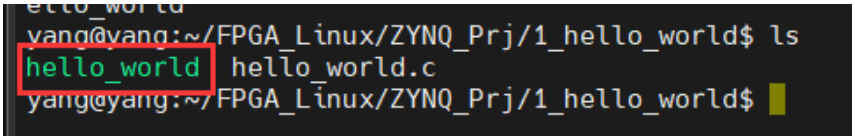

图 1-15 显示当前目录下文件

1.3.2 运行程序

在终端界面中输入“./hello_world 1 2 3”，将会看到如下图的输出：

命令讲解：

其中，`.`代表当前目录，`/`是目录分隔符。因此，`./hello_world`表示在当前目录下执行名为“hello_world”的可执行文件。

接下来的`1 2 3`则被视为命令行参数，这些参数会被传递给 hello_world 程序。在 C 程序中，你可以通过 main 函数的两个参数——int argc, char *argv[]来接收这些参数。

1. argc：参数计数，表示命令行参数的数量。在“./hello_world 1 2 3”这个命令中，argc 的值为 4。

2. argv：参数向量，是一个指向字符指针的指针，可用于访问命令行参数。argv[0]通常是程序的名称（在这里是“./hello_world”），argv[1]到 argv[argc-1]则是命令行传递给程序的参数（在这里是 1, 2, 3），argv[argc]是一个 NULL 指

针。

可以看到，我们编写的程序运行成功，传递的1、2、3，都接收到并打印出。

图 1-16 运行应用程序

1.4 Makefile

1.4.1 Makefile 简介

强调：Makefile 文件编写流程、Makefile 规则、Makefile 函数等等篇幅颇大，对于初学者很难短时间掌握，而如果前期花费大量时间在这方面，会打击学习热情以及迟迟难以进入驱动与应用的学习，甚至学了后面忘记了前面，故更加详细的 Makefile 使用教学，请翻阅**【格外篇之 Makefile】**，下面只简单介绍：

Makefile 是一种组织代码编译的方式。它是一个文本文件，该文件内含有一系列规则（rules），这些规则具体规定了如何依据不同的依赖关系（prerequisites）来更新或创建一个或多个目标文件（targets）。通过这些规则集，Makefile 能够有效地指导构建过程。

一个完整的规则由三个主要部分组成：

1. 目标（Target）：规则的目标通常指你要构建的文件或伪目标。

2. 依赖（Prerequisites）：这是目标所依赖的文件集合。如果依赖文件的修改时间晚于目标文件，则认为目标过时，需要重新构建。

3. 命令（Commands）：当目标需要被构建（即目标不存在，或者其依赖的文件有比目标更新的修改时间）时，make 会执行这一组命令来生成或更新目标文件。(命令开头是 Tab，不是空格)

因此，一个典型的 Makefile 规则结构是这样的：

```
目标(target): 依赖(prerequiries)
    命令(command)
```

Makefile 的主要优势是可以非常准确地跟踪源文件和目标文件之间的依赖关系，有效地减少不必要的编译和链接。

1.4.2 Makefile 自动化编译

现在我们继续尝试使用 Makefile 文件自动化编译 hello_world.c

(1) 输入命令“vim Makefile”

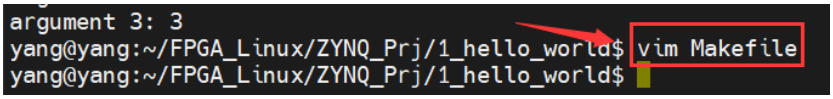

图 1-17vim 创建并编辑 Makefile

(2) 按下”i“，进入编辑模式

图 1-18 进入编辑模式

(3) 输入下面代码文件

```
all: hello_world

hello_world: hello_world.c
	gcc -o hello_world hello_world.c
clean:
	rm hello_world
```

(4) 按下“ESC”，退出编辑模式；输入命令“:wq”

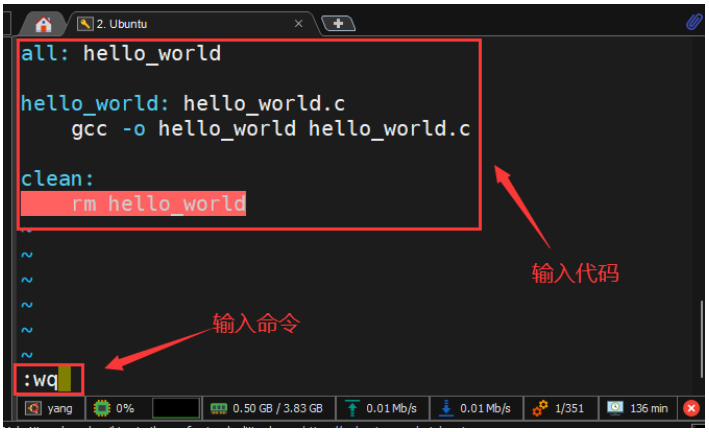

图 1-19 退出编辑模式

(5) 现在删除以前生成可执行文件，在终端中输入命令“rm hello_world”并回车。

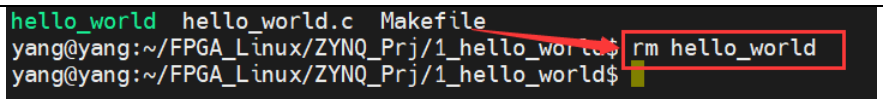

图 1-20 删除文件

输入命令“ls”，可以看到当前目录中存在的文件；这时“hello_world”已经消失不见。

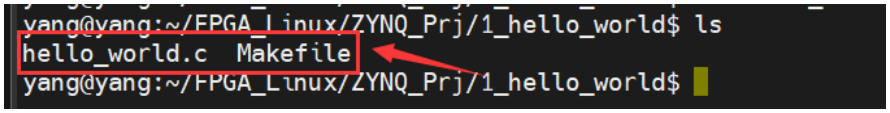

图 1-21 列出文件

(6) 继续输入 make，然后它竟出现报错，如下图所示。

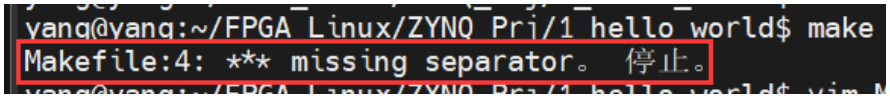

图 1-22 输入 make

注意：这里**故意写错代码**，就是为了告诉大家 Makefile 命令中 tab 键的重要！！！以后千万不要忘记命令前面是 tab，不是 4 个空格。

(7) 代码改正

终端界面中，输入命令“vim Makefile”，紧接着输入“i”，移动光标到命令前，按删除键，将所有空格删除，并按下 tab 键代替。现在退出 vim，按下 ESC 键，输入“:wq”；

提示：在 Vim 编辑器中，移动光标的命令非常直观，除了可以使用“上下左右”按键外，还可以使用以下的一些命令：

h - 左移一个字符

j - 下移一行

k - 上移一行

l - 右移一个字符

图 1-23 代码改正

(8) 执行 make

在命令行继续输入 make，出现下图输出文字，说明编译成功。

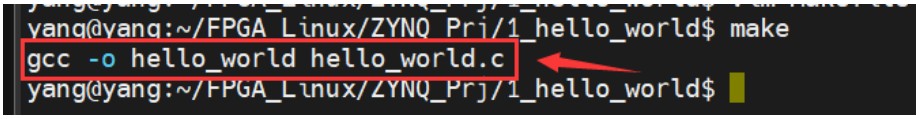

图 1-24 重新执行 make

接下来，输入命令“ls”，可以看到可执行程序“hello_world”又回来了。

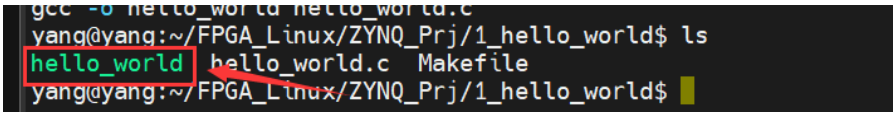

图 1-25 列出所有文件

现在可以执行在终端界面中输入命令“./hello_world 1 2 3”执行程序，结果与上面“运行程序”小节中一致。

1.5 gcc 编译器

1.5.1 gcc 简介

强调：考虑到大家为初学者，如果一开始就大篇幅介绍 gcc 这些新东西，会很容易陷入”陷阱“，打击学习热情以及迟迟难以进入驱动与应用的学习，甚至学了后面忘记了前面，故更加详细的 gcc 使用教学，请翻阅**【格外篇之 gcc 编译器】**，下面只简单介绍：

GCC 编译器，全称 GNU Compiler Collection，是 GNU 开源项目中的一个重要部分，起初仅是 GNU 的 C 语言编译器，后续发展成了一个支持多种编程语言（如 C++、Java、Objective-C、Fortran、Ada 等）的强大编译器。

以下列出的是 GCC 编译器的主要功能：

1. 编译代码：GCC 的最基本的作用是编译源代码，将高级语言编写的源代码转换为机器代码，从而可以在计算机上执行。

2. 代码优化：GCC 提供了各种不同级别的代码优化选项，可以通过编译器做到程序的优化，提高程序的运行效率。

3. 跨平台支持：GCC 支持各种不同的处理器架构以及操作系统，可以在多种不同的环境下编译出可以在不同平台运行的代码。

4. 错误和警告提示：GCC 在编译过程中会检查代码中的语法错误，逻辑错误以及潜在的问题，并且提供易于理解的错误信息或者警告信息，帮助开发者找出并修复问题。

5. 调试信息：GCC 可以生成帮助调试的符号信息，方便使用 GDB 等调试工具对程序进行调试。

1.6 总结

通过前面的学习，我们完成在嵌入式 Linux 平台开发环境的搭建，奠定了程序开发的基础。接着，利用 C 语言编写了一段简洁的"Hello, World"代码，展示了代码的基本结构及功能。

然后，我们进行了程序的编译，体验了在 Linux 环境下利用 gcc 编译器实现程序编译的过程，以及如何解决可能出现的编译问题。

最后，我们成功地在 Linux 上运行了"Hello_World"程序。通过这一切，我们开始理解嵌入式 Linux 平台的工作方式，为接下来更加复杂的实验和开发工作奠定了坚实的基础。