成绩

中国农业大学

课程设计

(2023 -2024 学年夏季学期)

题　　目： 计算机系统综合训练-任务 1+任务 2

课程名称： 计算机系统工程综合实践

任课教师： 王耀君, 黄岚, 段青玲, 史银雪

班　　级： 计算机 211

学　　号： 2020301010225

姓　　名： 夏婉可

计算机系统工程综合实践 ——任务 1（接口技术）

一：课程设计任务与实验设备、开发工具链

（1）课程设计任务

1：对 MIPS 处理器软核的片上系统分析。

2：完善 C 语言程序，完成对 GPIO 的控制。

3：完成实验报告，包括 GPIO 原理图、带注释的 C 语言程序、可行的调试方案和步骤、课程设计总结等部分。

（2）实验设备

Minisys 开发板、Bus Blaster 开发板、带 Windows 7 操作系统的主机、设备连接线、电源线等。

（3）开发工具链

OpenOCD、Codescape、Vivado 2014.4、串口调试助手等。

二：总体设计思路

硬件基础任务的总体设计主要包括 2 大方面，如上图所示。

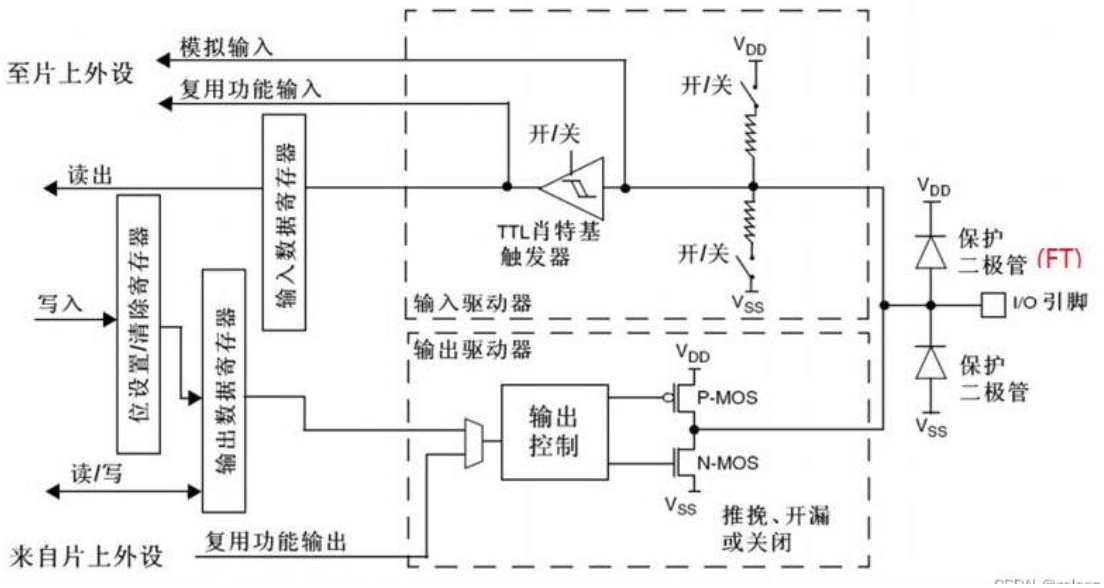
第一点是连接 Minisys 板，加载 MIPS fpga project，分析接口地址，下载片上系统的 bit 文件。

第二点是采用 OpenOCD、Codescape 工具链接完后交叉开发，在无操作系统裸板上运行程序，完成 GPIO 和串口控制。

三：硬件基础任务的设计与实现

（1）GPIO 的原理图

GPIO 的模拟电路设计图，如下图所示。



GPIO 全称为通用输入/输出（General Purpose Input Output），是负责拨码开关和 LED 灯访问的模块。

在本实验中，Minisys 板具有 4 个控制 LED 灯和拨码开关的 GPIO 寄存器，包括 2 个 GPIO 端口读写的寄存器和 2 个 GPIO 输入输出方向的设置寄存器。其地址偏移量和功能如下图所示。

Register	Address offset	Function
DATA	0x00	If pin is input, reads to bit read pin value. Writes are discarded. If pin is output, reads to bit read the current output value on pin, write to bit write the value on the pin.
TRIS	0x04	Selects which bit is input (1) or output (0).

（2）C 语言源程序（含注释）

实现 GPIO 控制的 C 语言程序，如下表所示。

main.c 文件内容

```
/*
 * main.c for the MIPSfpga core running on Nexys4 DDR board.
 *
 * This program:
 *   (1) reads the switches on the Nexys4 DDR board and
 *   (2) flashes the value of the switches on the LEDs
```

```
 */

void delay();

int main() {
  // volatile：防止编译器优化变量
  volatile int *IO_iotr1= (int*)0xb0600004;
  volatile int *IO_iotr2= (int*)0xb0c00004;
  /*
    [virtual address for tris register]

    input is 1(f...f), output is 0(0...0), which are set later

    IO_iotr1 is 0 => output => LEDR
    IO_iotr2 is 1 => input => SWITCHES

    offset = 0x04
    [IO_iotr1] = [LEDR] + offset
    [IO_iotr2] = [SWITCHES] + offset
  */

  volatile int *IO_SWITCHES = (int*)0xb0c00000;
  volatile int *IO_LEDR = (int*)0xb0600000;
  /*
    [virtual addresses for data register]

    io_switches：拨码开关的存储映射 IO 地址
    io_ledr：LED RED 灯的存储映射 IO 地址
    只是给定了 GPIO 的输入&&输出地址，控制地址需要确定

    返回低 16 位：拨码开关的值
  */
  volatile unsigned int switches;

  *IO_iotr1=0x00000000;
  *IO_iotr2=0xffffffff;
  /*
    32bit
    IO_iotr1 is input
    IO_iotr2 is output
  */

  while (1) {
    switches =*IO_SWITCHES; // 读取拨码开关地址，确定开关状态
    delay();                // 延时
    *IO_LEDR = switches;    // 显示值到 LED 灯上
```

```
    *IO_LEDR = 0;            // turn off LEDs
    delay();                 // 延时
  }
  return 0;
}

void delay() {
  volatile unsigned int j;
  // delay for 100,0000 times
  for (j = 0; j < (1000000); j++) ; // delay
}
```

程序整体主要包括主函数 main 和延时函数 delay。

主函数部分设置了与拨码开关和 LED 灯有关的虚拟地址，包括方向控制寄存器的地址和数据寄存器的地址。在 minisys 开发板中，各组成部分的虚拟地址和物理地址的映射关系，如下图所示。

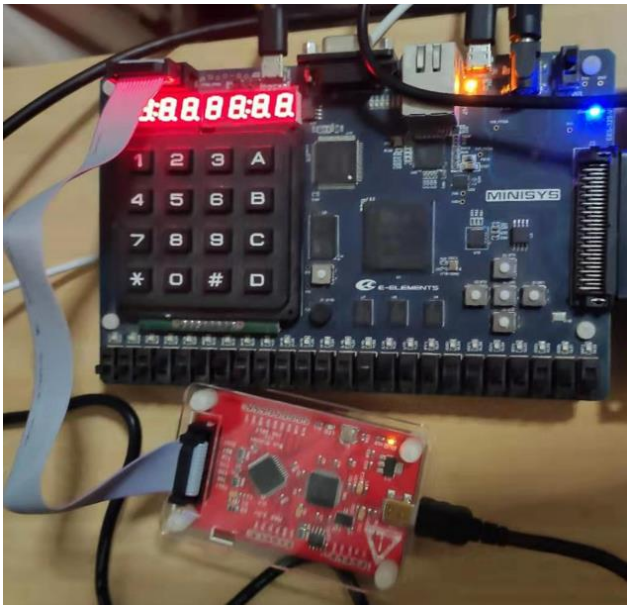
表 5. FPGA 板卡内存映射 I/O 地址

虚拟地址	物理地址	信号名	Minisys	DE2-115
0xb0600000	0x10600000	IO_LEDR	LEDs	Red LEDs
0xb0600004	0x10600004	IO_LEDG	N/A	Green LEDs
0xb0c0 0000	0x10c0 0000	IO_SW	switches	switches
0xb0c0 00040	0x10c0 0004	IO_PB	U, D, L, R, C pushbuttons	pushbuttons

四：具体调试方案

1：补充和修改 main.c 文件的代码，并将工程文件拷贝到主机。

2：检查 minisys 和 Blaster 的线路连接情况，包括 Blaster 与主机、Blaster 与 minisys 板、minisys 和主机、minisys 和电源等。三个硬件之间的连接结果，如下图所示。

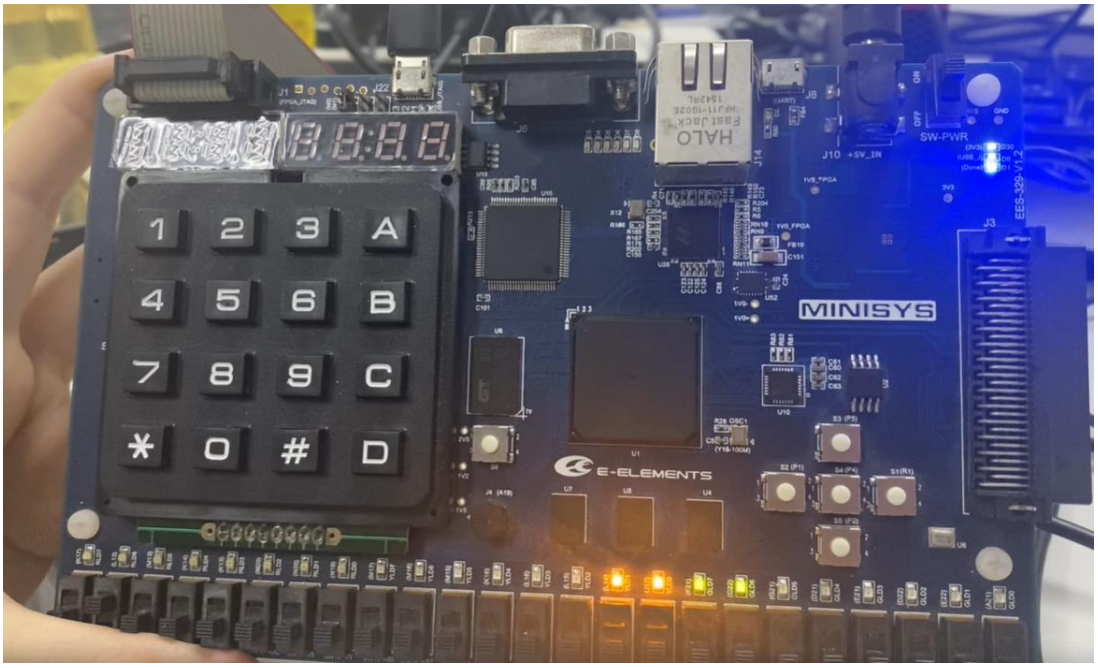


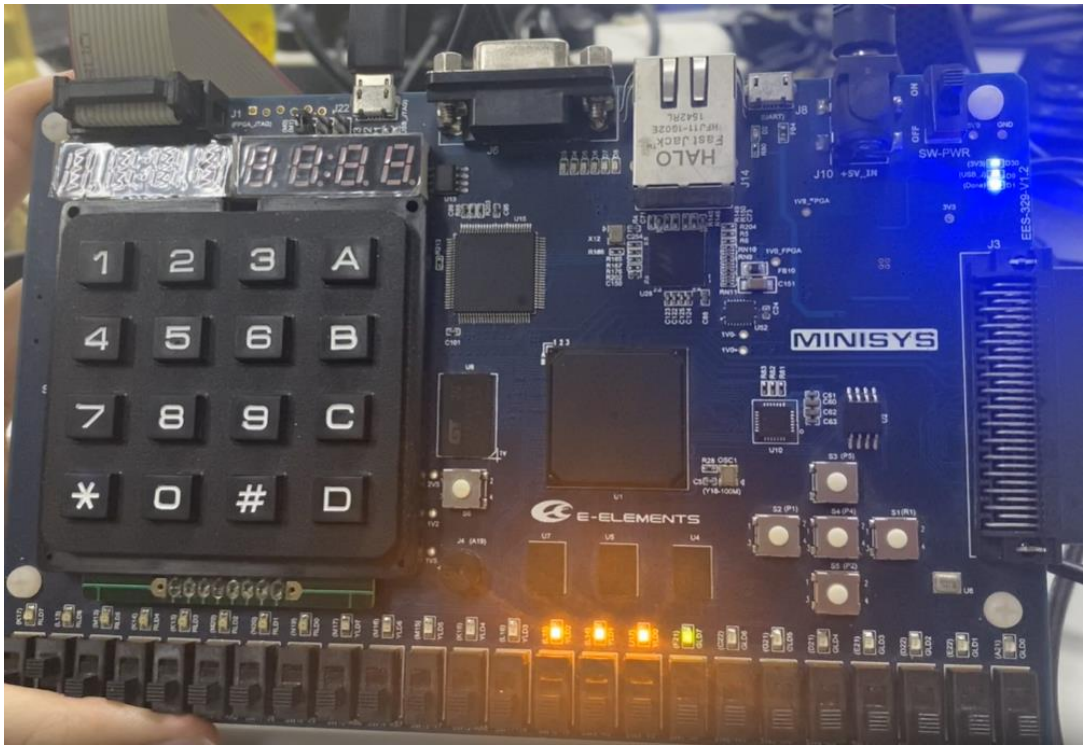
3：打开 vivado 2015，选择 Flow，选择 Open Hardware Manager。

4：选择 Open target，选择 Auto Connect。

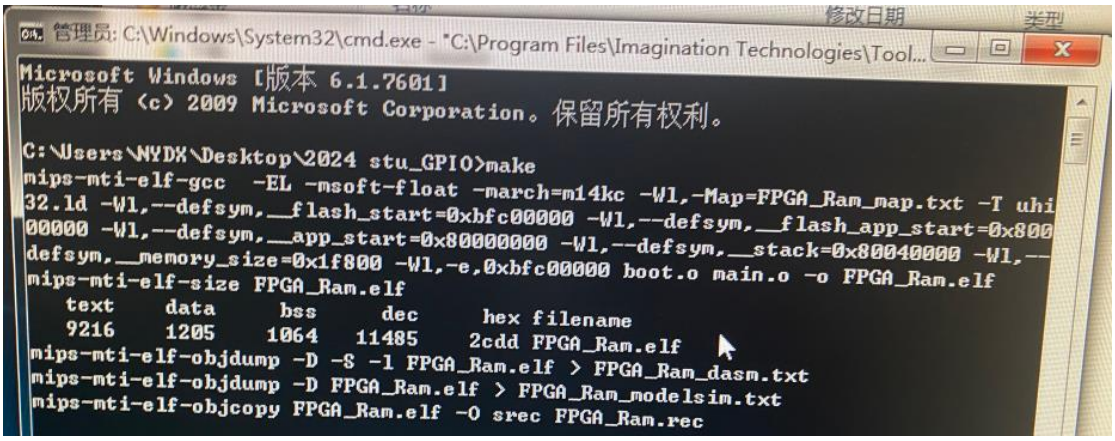
5：选择 Program device，在弹出窗口的 Bitstream file 中选择比特流文件 2019_design_1_wrapper.bit，然后选择 Program。同时，注意该比特流文件所在路径不能包含中文。

6：比特流下载完成后，按下 minisys 上的 reset 键（s6），比特流包含的左移亮灯程序成功演示。如下图所示，每次亮起 4 个灯，延时一段时间后向左移动 1 个灯的位置。

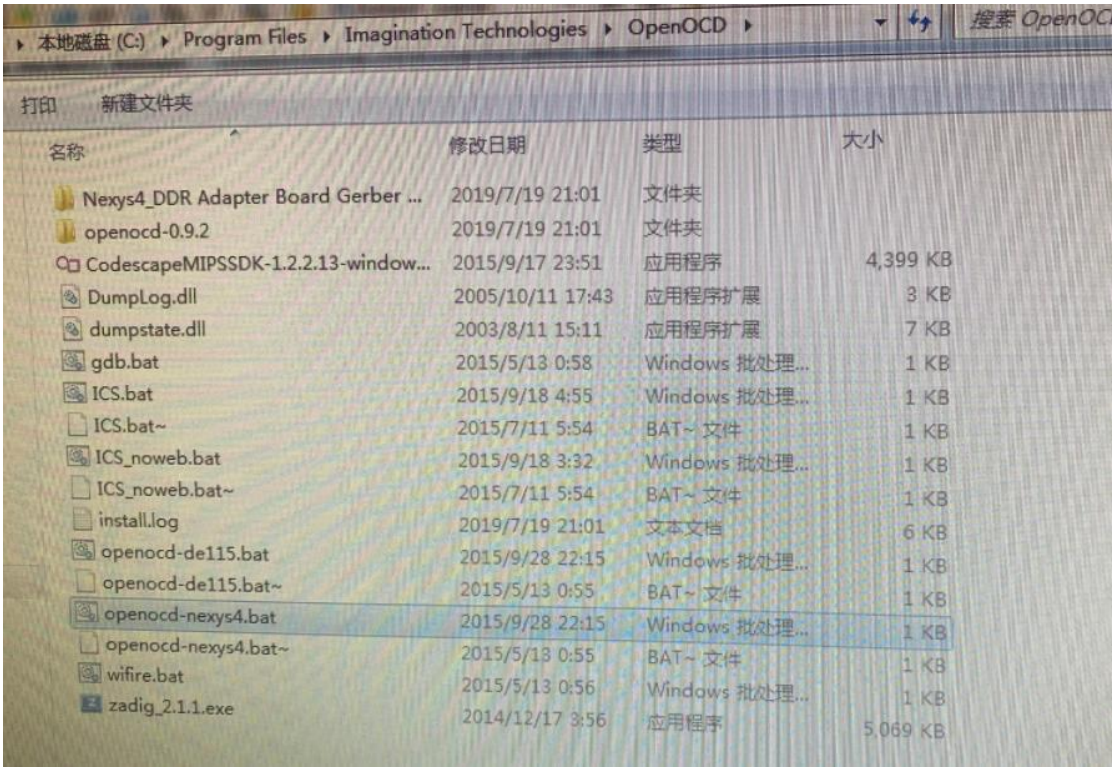


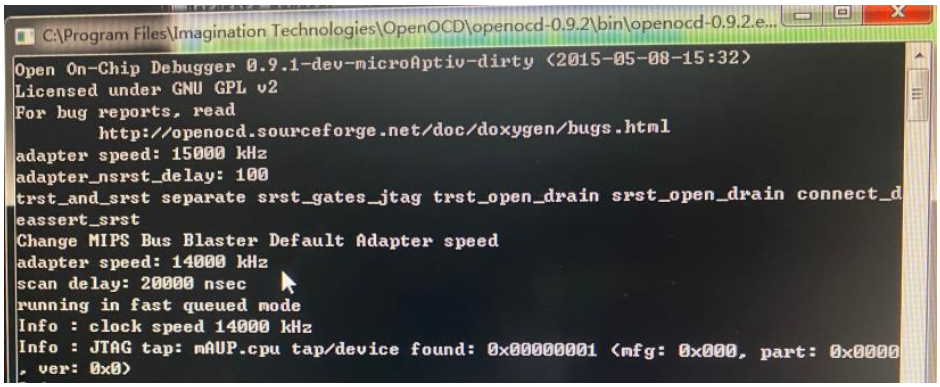


7：编译 C 语言程序，在 main.c 文件的目录下打开 CMD 终端，输入 make 进行编译。结果如下图所示。

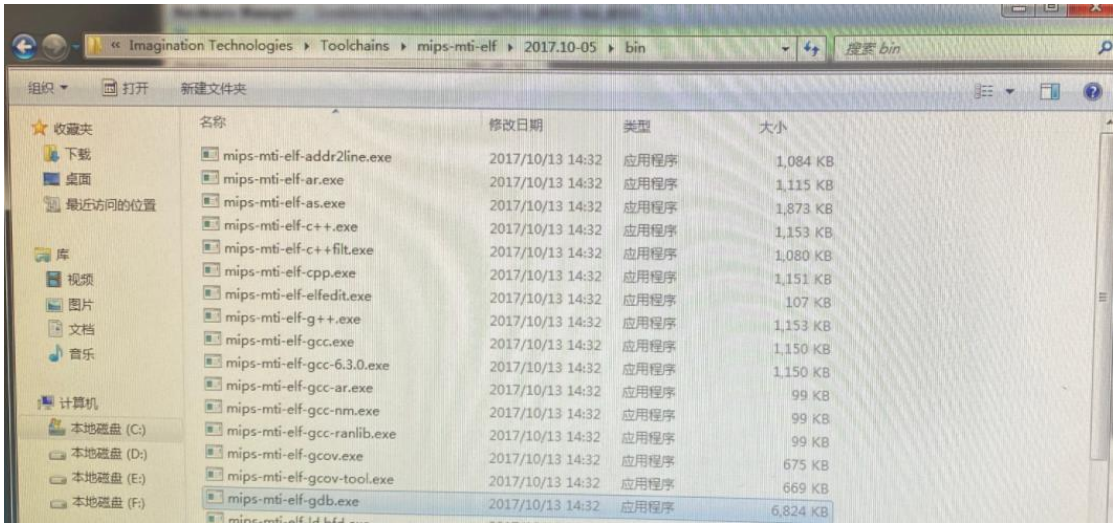


8：在 C:\\Program Files\\Imagination Technologies\\OpenOCD 路径下运行运行批处理文件 openocd-nexys4.bat。结果如下图所示。

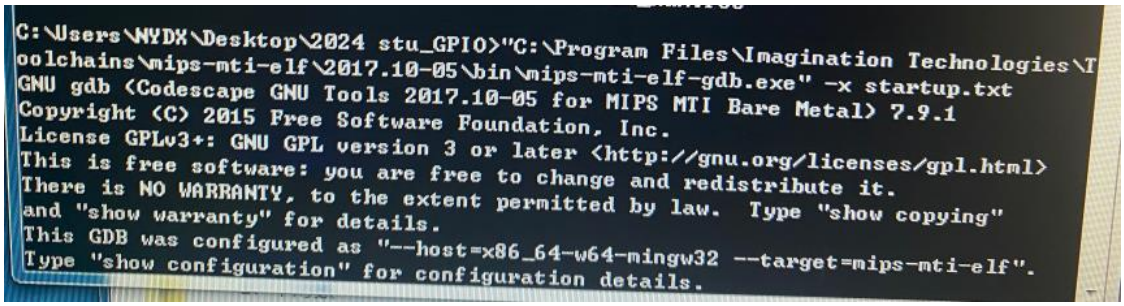


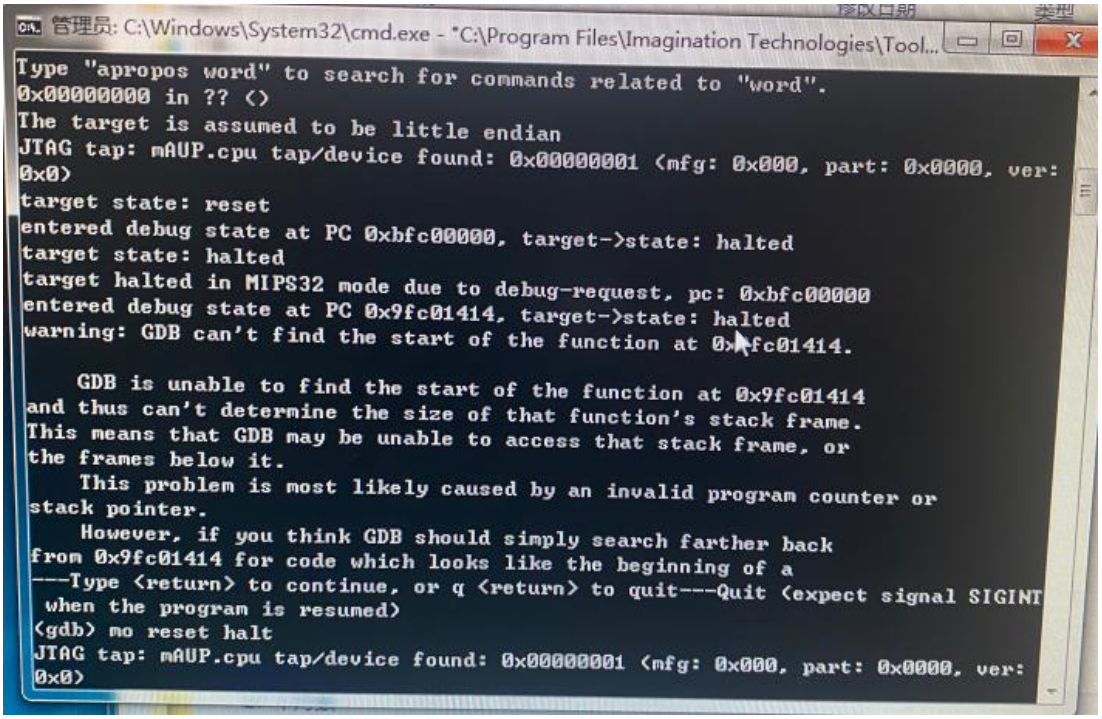


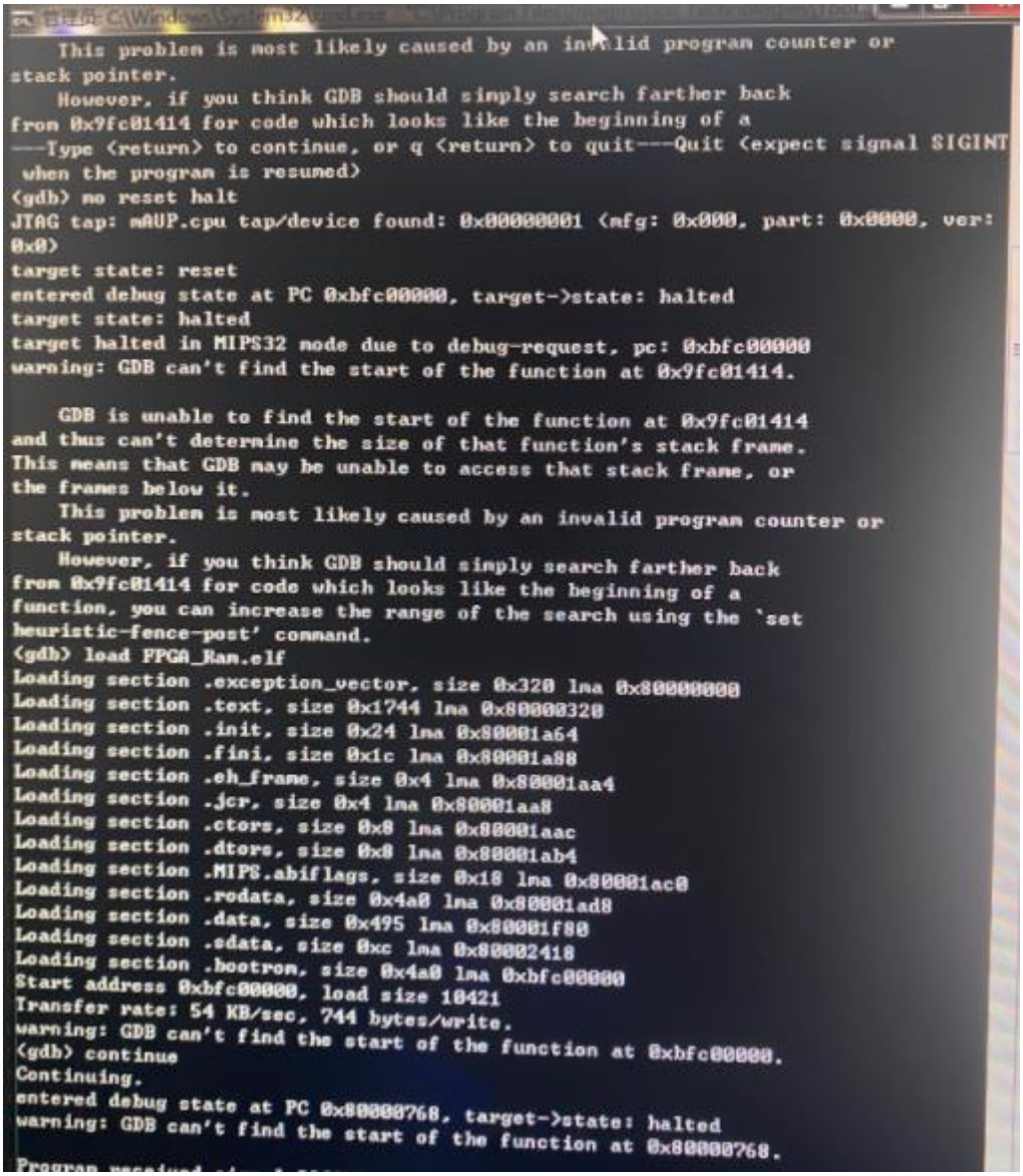
9：在之前的 CMD 窗口下，执行命令"C:\Program Files\Imagination Technologies\Toolchains\mips-mti-elf\2017.10-05\bin\mips-mti-elf-gdb.exe" -x startup.txt。该 exe 文件的路径，如下图所示。



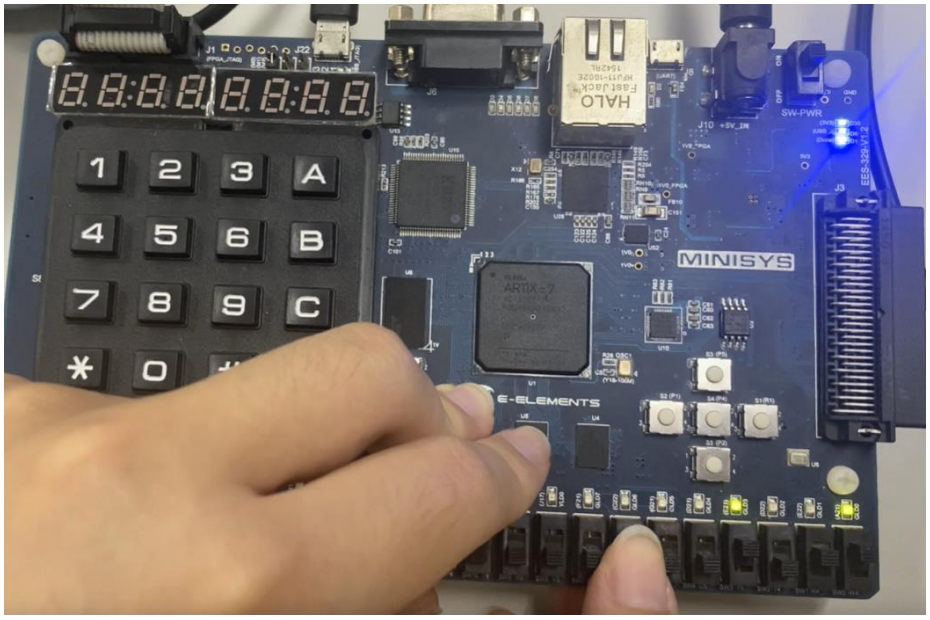
而后，按下两次 ctrl+c，在 gdb 提示符下，先执行命令 mo reset halt，再执行命令 load FPGA_Ram.elf，最后执行命令 continue。在 CMD 终端执行的过程，如下图所示。







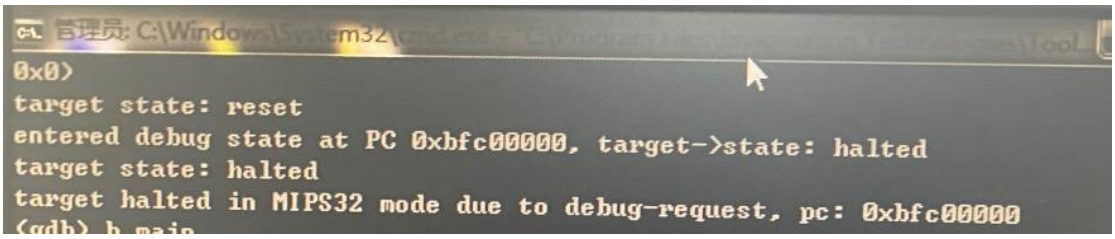
10：在 minisys 上进行拨码开关的操作，观察 LED 灯的闪烁情况。发现拨上的拨码开关所对应的 LED 灯均有闪烁，如下图所示。



五：调试过程与结果分析

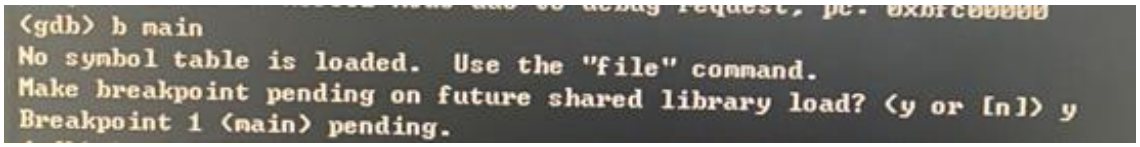
在 gdb 命令窗口。按顺序输入表 1 中的命令来停止处理器工作、设置断点、查看变量和寄存器的值等。

1：停止并复位处理器，执行命令 mo reset halt。

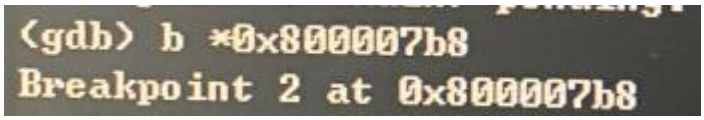


2：在 main 函数处设置一个断点，执行命令 b main。

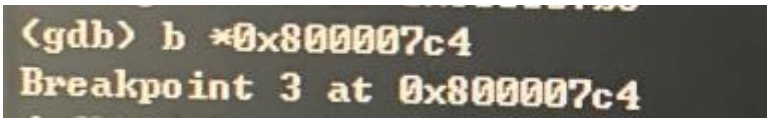
在选择设置【让断点在将来加载共享库时等待？】时，选择确定【y】。



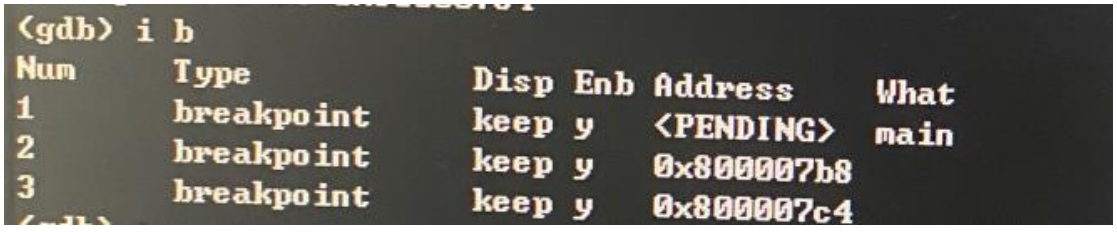
3：在 0x800007b8 指令地址上设置一个断点，执行命令 b *0x800007b8。



4：在 0x800007c4 指令地址上设置一个断点，执行命令 b *0x800007c4。



5：列出所有断点，执行命令 i b。



6：让处理器继续运行，执行命令 c。

```
(gdb) c
Continuing.
entered debug state at PC 0x800007b8, target->state: halted
warning: GDB can't find the start of the function at 0x800007b8.

[Remote target] #1 stopped.
0x800007b8 in ?? ()
```

7：打印从当前正在执行的指令开始的三条指令，执行命令 x/3i $pc。

```
(gdb) x/3i $pc
=> 0x800007b8:  sw      v0,0(s0)
   0x800007bc:  jal     0x80000734
   0x800007c0:  nop
```

8：以 16 进制的形式打印从当前正在执行的指令开始的三条指令，执行命令 x/3x $pc。

```
(gdb) x/3x $pc
0x800007b8:     0xae020000      0x0c0001cd      0x00000000
(gdb) c
```

9：继续执行到第二个断点 0x800007b8 处，执行命令 c。

```
(gdb) c
Continuing.
entered debug state at PC 0x800007bc, target->state: halted
warning: GDB can't find the start of the function at 0x800007bc.
entered debug state at PC 0x800007c4, target->state: halted
warning: GDB can't find the start of the function at 0x800007c4.

[Remote target] #1 stopped.
0x800007c4 in ?? ()
```

10：执行一条指令，执行命令 si。

```
(gdb) si
entered debug state at PC 0x800007c8, target->state: halted
warning: GDB can't find the start of the function at 0x800007c8.
0x800007c8 in ?? ()
```

11：执行一条指令，执行命令 si。

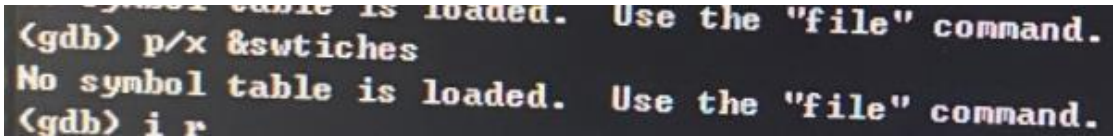
```
(gdb) si
entered debug state at PC 0x80000734, target->state: halted
warning: GDB can't find the start of the function at 0x80000734.
0x80000734 in ?? ()
```

12：打印 switches 这个变量的值，执行命令 p switches。

```
(gdb) p switches
No symbol table is loaded.  Use the "file" command.
(gdb) p/x swtiches
```

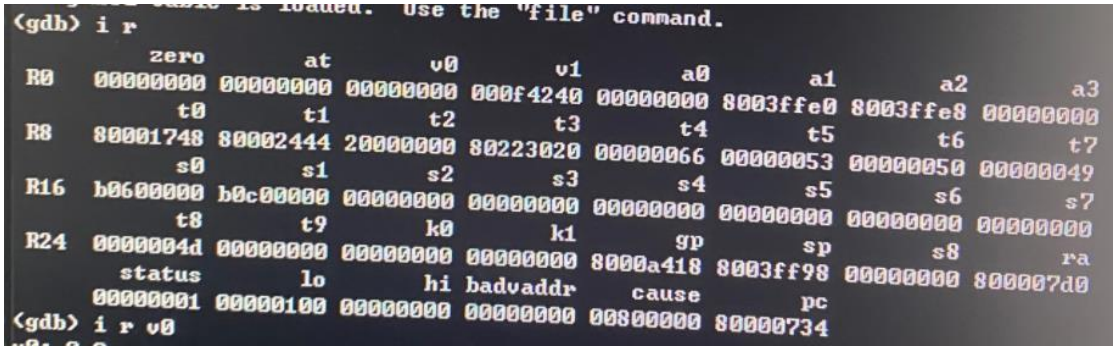
13：以 16 进制的形式打印 switches 变量，执行命令 p/x switches。

```
No symbol table is loaded.  Use the "file" command.
(gdb) p/x swtiches
No symbol table is loaded.  Use the "file" command.
(gdb) p/x &swtiches
```

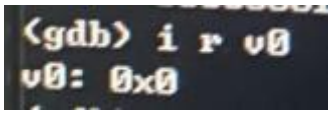


14：打印 switches 变量的地址，执行命令 p/x &switches。

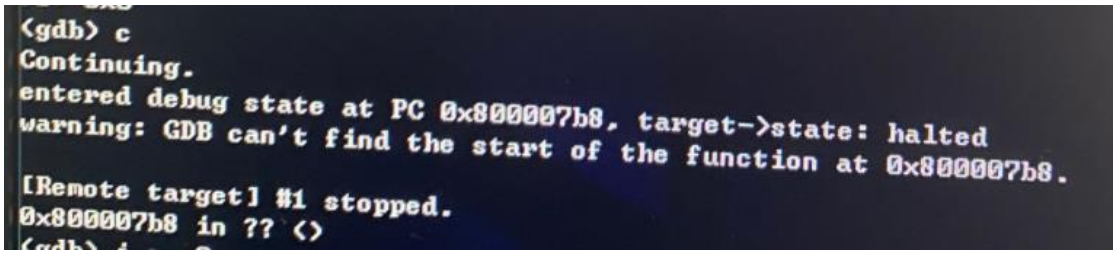
15：打印所有寄存器的值，执行命令 i r。



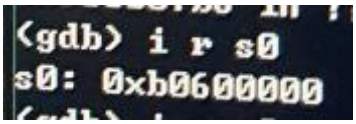
16：打印 v0 寄存器的值，执行命令 i r v0。



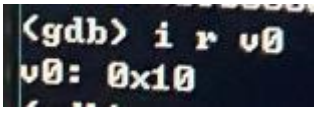
17：继续执行程序，执行命令 c。



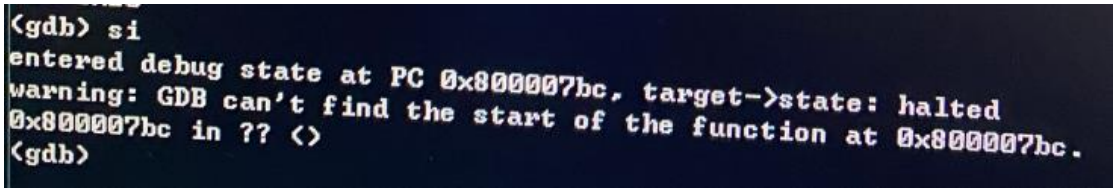
18：打印 s0 寄存器的值，执行命令 i r s0。



19：打印 v0 寄存器的值，执行命令 i r v0。



20：执行存数指令，将看到 LED 灯更新为拨码开关的值，执行命令 si。



21：删除断点 1，执行命令 d 1。

22：复位并运行处理器，执行命令 mo reset run。

完整的 gdb 命令序列，如下表所示。

命令	描述
monitor reset halt	停止并复位处理器，程序也会停止运行。运行此命令后，**以便** load 新程序 缩写：mo reset halt。
b main	在 main 函数处设置一个断点（"break main".的简写)。 注意断点设置在了 0x800007b4，刚好在代码中 delay 函数和栈操作的后面（地址在 0x8000075c - 0x800007b0) 你可以在处理器运行的过程中设置断点，但是只有处理器在停止工作之后（mo reset halt）这个断点才起作用。
b *0x800007b8	在 0x800007b8 指令地址上设置一个断点。ReadSwitches C 程序在这个地址对应的指令为读取拨码开关的值（参考 MIPSfpga_Fundamentals\Xilinx\Lab02_C\ReadSwitches\FPGA_Ram_dasm.txt)。 800007b8: 8e020008 lw v0,8(s0) 这个命令还有另外一个版本：b 20，在 main.c 的第 20 行设置断点。
b *0x800007c4	在 0x800007c4 指令地址上设置一个断点。ReadSwitches C 程序在这个地址对应的指令为把值写到 LED 灯上（参考 MIPSfpga_Fundamentals\Xilinx\Lab02_C\ReadSwitches\FPGA_Ram_dasm.txt)。 800007c4: ae020000 sw v0,0(s0)

i b	列出所有的断点 (info breakpoint 的简写)。经过前面几步之后这个命令会列出前面所设置的三个断点 0x800007b4 (main), 0x800007b8, and 0x800007c4。
c	让处理器继续运行。(continue 的简写), 这个命令会停在第一个断点, 在这里会让 main 程序开始执行 (指令地址 0x800007b4)。
x/3i $pc	打印从当前正在执行的指令开始的三条指令(程序计数器$pc 存的是当前指令的地址) 800007b4: lui s0,0xbf80 800007b8: lw v0,8(s0) 800007bc: sw v0,16(sp)
x/3x $pc	以 16 进制的形式打印从当前正在执行的指令开始的三条指令。
c	继续执行到第二个断点 0x800007b8 处。
stepi	执行一条指令。如果这样做的话你就会发现 PC 已经加到了 0x800007bc. 简写: si
si	再执行一条指令。(你可以简单的用回车键 (Enter) 来重复上一条 gdb 命令)
p switches	现在拨码开关的状态已经读进来, 我们可以使用这个命令来打印 switches 这个变量的值来检测其是否正确。(print switches 的缩写)。如果把拨码开关的最低三位置为 1 (开关处于上面的位置) 那这个 switches 的值就应该是 7。
p/x switches	以 16 进制的形式打印 switches 变量。
p/x &switches	打印 switches 变量的地址。

i r	打印所有寄存器的值。(Info registers 的简写。)
i r v0	打印 v0 寄存器的值。这时候 v0 中存的是拨码开关的状态，这个值将会通过 0x800007c4 地址上的 sw 指令写到 LED 灯上。
c	继续执行程序（continue 的简写）。当前执行的指令是地址 0x800007c4 上的 sw 指令，这条 sw 指令把拨码开关的值送给 LED 灯。
i r s0	打印 s0 寄存器的值，现在 s0 中的值为 LED 灯的存储映射地址: 0xbf800000。

六：本组设计的特色

测试了断点运行，观察了寄存器值的变化（从灯亮到灯灭）。

七：本次设计的总结

1：学习了 minisys 中物理地址和虚拟地址的对应关系，其中包含 LED 灯、拨码开关、按钮这三个部件的地址信息。可以发现：物理地址 + 0xa0000000 = 虚拟地址。

2：在使用 vivado 软件在载入文件时，路径中不应该包含中文字样，否则会报错加载错误。

3：load 命令应该载入 FPGA_Ram.elf 文件，该文件是编译生成的可执行和可链接格式文件，用于把程序下载到处理器系统的内存上。

4：blaster 的作用是将 C 语言程序载入到 minisys 板子上，用于调试功能。

5：vivado 的作用是将比特流文件下载到 minisys 板子上，用于生成软核 CPU。

6：软核 CPU 是利用 HDL 语言描述的处理器功能代码，用于实现处理器的所需要的各种功能。

八：参考文献

1：计算机系统工程综合实践任务一指导手册.pdf。

2：MIPSfpga SOC Starter Tutorial.pdf。

3：GPIO-led-switch_任务一 2024v2.docx。

4：GPIO 实验 IO 端口地址.pdf。

5：中文指南修改版 2019-在 MINISYS 移植 MIPSFPGA.pdf。

6：中文指南-MIPSfpga Getting Started Guide.1.2-CHN.pdf。

计算机系统工程综合实践——任务2（操作系统）

一：课程设计任务与实验设备、开发工具链

（1）课程设计任务

1：在虚拟机下完成Linux操作系统裁剪和编译。验证开发板的连接、驱动程序的安装，利用相关工具和命令完成操作系统的工作。

2：Minisys平台操作系统的移植。通过OpenOCD调试开发板，将裁剪后的Linux加载到开发板上，并确保Linux正常运行。

3：程序设计控制LED灯和开关。

【1】在移植到开发板的 Linux 环境下使用 Shell 编程，将自己的学号后四位以16位2进制的方式显示在开发板的16个LED灯上（亮表示1，灭表示0，每个数字对应4盏灯）。

【2】编写 C 程序，并在开发板上运行，实现2个功能。

功能1：根据 16 个开关的拨动情况控制对应的 16 个 LED 灯的亮或灭。

功能2：本组号为初始亮灯，从左往右移动实现跑马灯，超出左边高位的部分要补足到右侧低位的灯上，每2秒移动一位。

（2）实验设备

Minisys开发板、Bus Blaster开发板、网络通信板、带Windows 7操作系统的主机、设备连接线、电源线等。

（3）开发工具链

OpenOCD、Codescape、Vivado 2014.4、串口调试助手、Dev CPP、记事本等。

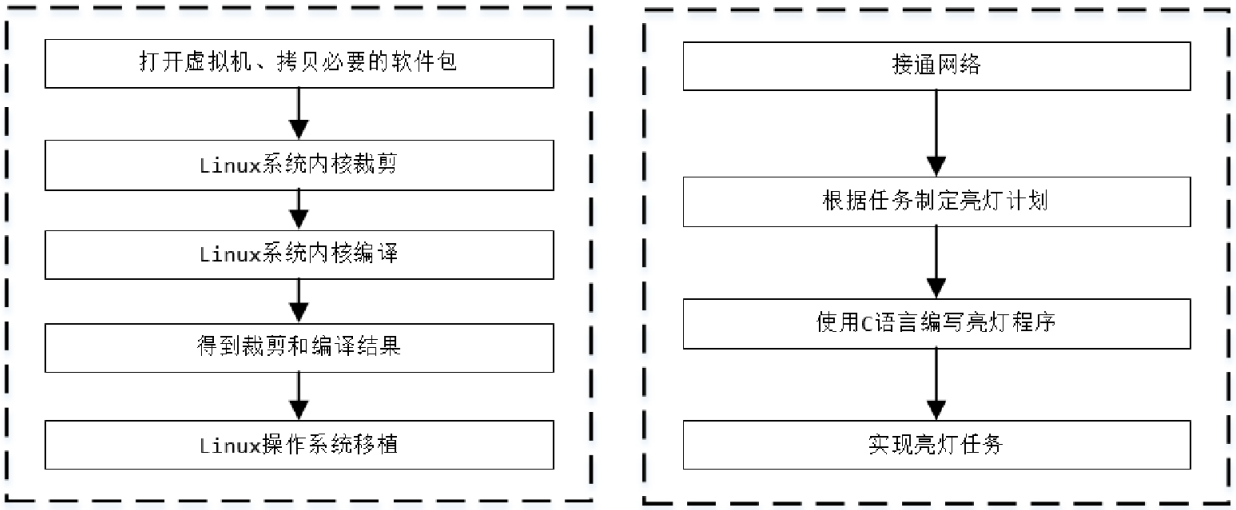
二：总体设计思路

硬件基础任务的总体设计主要包括2大方面，如下图所示。

第一点是移植操作系统。

第二点是基于操作系统，完成对 GPIO 的驱动。

三：操作系统任务的设计与实现



基于操作系统的移植任务的设计，如上图左侧所示。主要包括五个步骤，分别是【1】打开虚拟机、拷贝必要的软件包。【2】Linux 系统内核裁剪。【3】Linux 系统内核编译。【4】得到裁剪和编译结果。【5】Linux 操作系统移植。

基于操作系统的驱动任务的设计，如上图右侧所示。主要包括四个步骤，分别是【1】接通网络。【2】根据任务制定亮灯计划。【3】使用 C 语言编写亮灯程序。【4】实现亮灯任务。

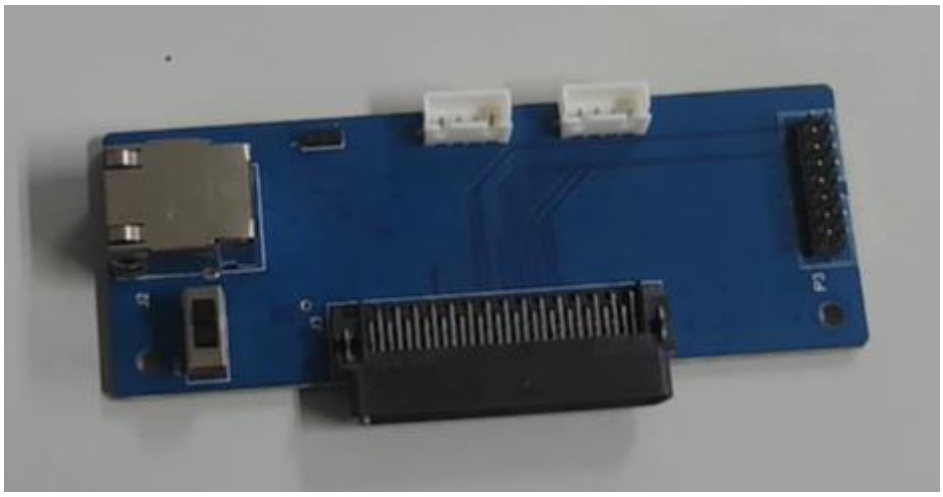
四：具体调试方案

1：认识硬件。

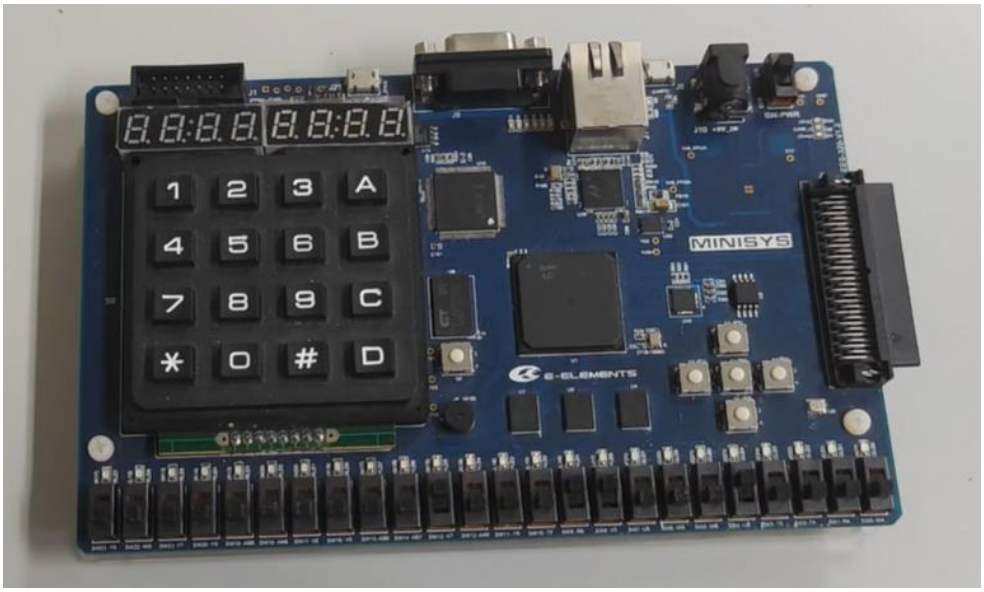
除了在任务 1 中的基本硬件之外，本任务额外需要网络通信板、网线等硬件设备。

网络通信板需要和 minisys 开发板的右侧直接通过卡槽进行连接。

网络通信板的硬件结构，如下图所示。

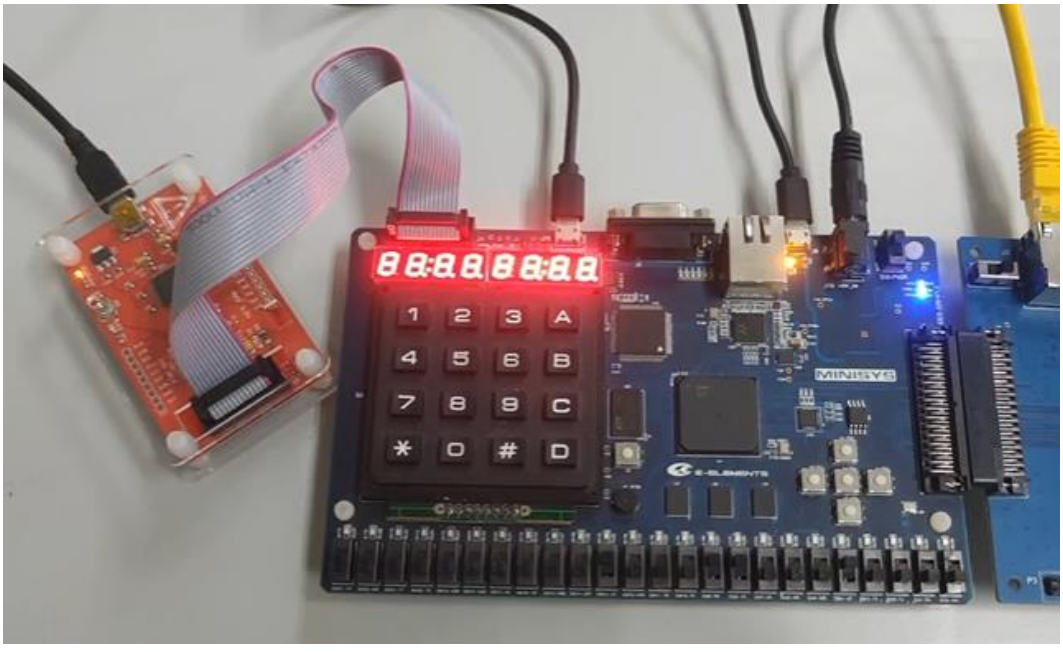


minisys 开发板的硬件结构，如下图所示。



2：连接硬件。

将任务所需的所有硬件按照次序连接，如下图所示。



3：烧写 mips fpga 到开发板。

打开 Vivado 软件，按照下述步骤将 design_1_wrapper.bit 比特流文件烧写到 minisys 开发板。

选择【open hardware manager】。

选择【open target】下的【auto connect】。

选择【program device】，在弹出窗口的【bitstream file】处选择 design_1_wrapper.bit 比特流文件。

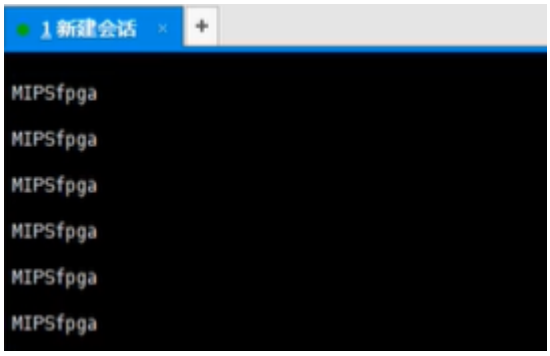
选择【program】，等待烧写完成。

4：观察 mipsfpga 下板子现象。

按下 minisys 开发板上的 reset 键，可以看到 4 个 LED 亮灯按照一定延时从右边向左边移动。

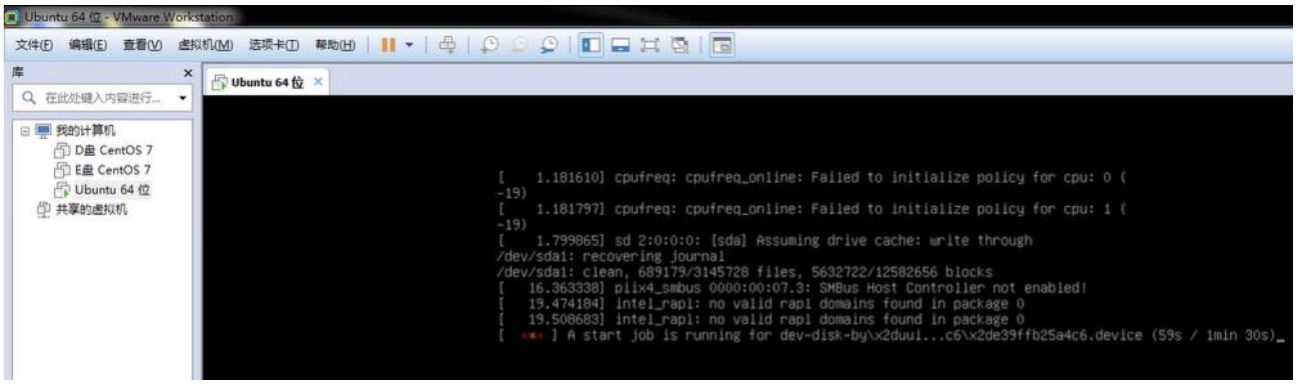
5：通过 XShell 验证现象。

打开 XShell 软件，新建一个与 minisys 开发板连接的会话。协议选择【serial】，端口号与设备管理器中的硬件端口号一致（例如都是【COM3】），连接速率选择【115200】。设置完成后确定，并在终端看到屏幕持续输出【MIPSfpga】字样。



6：启动 Linux 虚拟机。

在 Ubuntu 64 位目录下，打开【Ubuntu 64 位.vmx】，进入虚拟机界面。



7：拷贝必要文件到工作目录。

需要的安装包和软件包已经保存在路径【/home/zpf/resources】。

创建工作目录 mipsfpga，并将 Linux 的源代码和 CodeScape 工具链软件包拷贝到这个目录下。

配置 ToolChain，在 mipsfpga 下新创建 toolchain 目录，将 CodeScape 工具链解压到这个目录下。

配置 BuildRoot，在 mipsfpga 下新创建 buildroot 目录，拷贝 BuildRoot 到这个目录下。

8：Linux 系统裁剪。

执行编译命令，如下图所示。

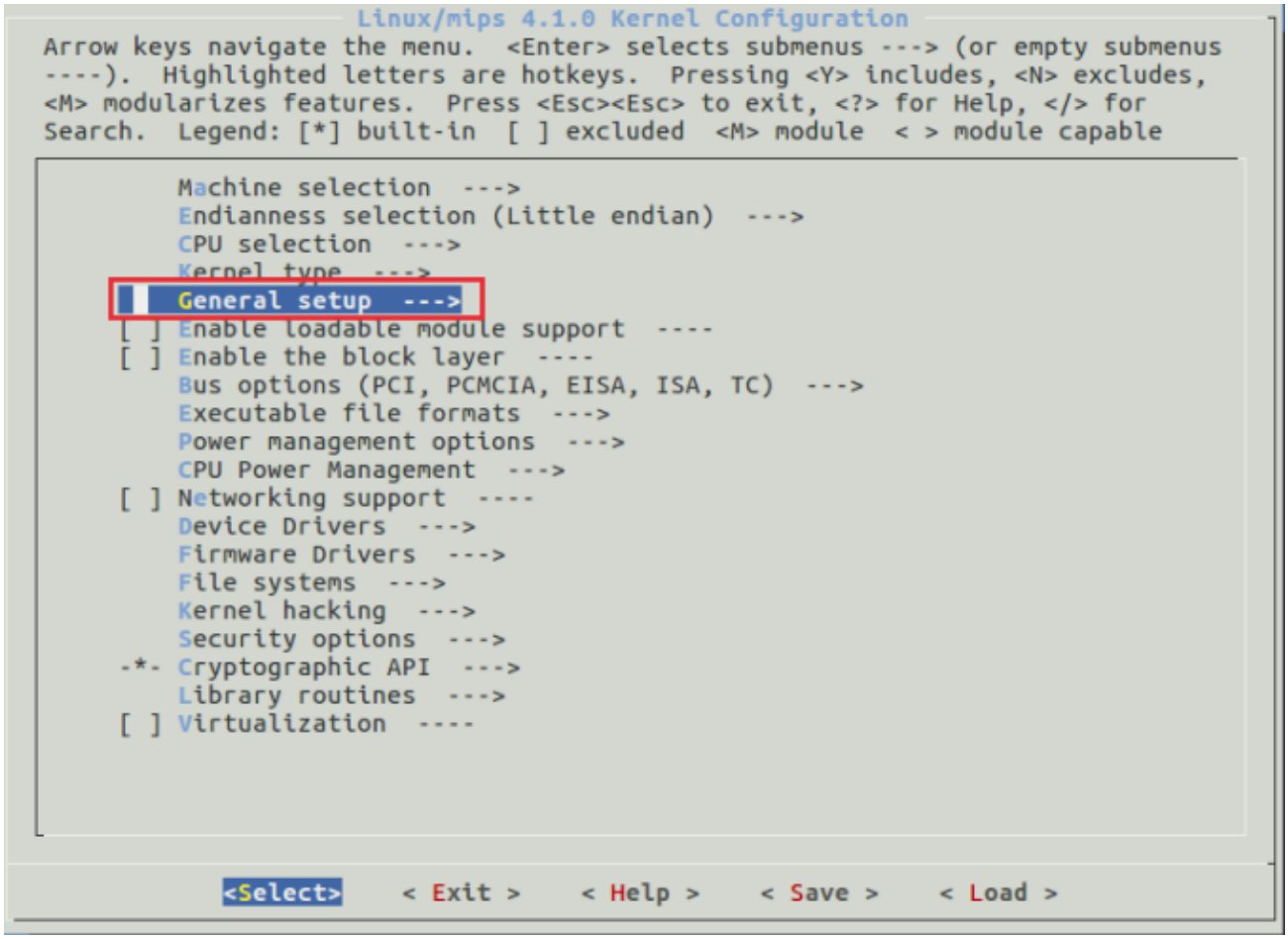
```
make xilfpga_static_defconfig
make
```

回到 mipsfpga 目录下，进入内核 kernel 目录。

执行打开 menuconfig 命令，如下图所示。

```
make ARCH=mips menuconfig
```

进入配置界面，如下图所示。按照视频完成 Linux 系统的裁剪。



9：对裁剪后的系统进行编译。

执行重新编译 linux 内核的命令，如下图所示。

编译过程时间长，编译完成后会生成 3 个文件，分别是 vmlinux、vmlinux.o、vmlinuz。

其中，vmlinux 文件是我们裁剪需要的结果。

```
make ARCH=mips CROSS_COMPILE=~/mipsfpga/toolchain/mips-mti-linux-gnu/2015.06-04/bin/mips-mti-linux-gnu-
```

10：移植 Linux 系统到开发板。

在 OpenOCD 中运行 openocd-nexys4.bat 批处理文件，进入终端窗口并保存窗口。

在 Starter_Tutorial 目录下，开启 CMD 终端，执行以下命令。

```
"E:\Program Files\Imagination Technologies\Toolchains\mips-mti-elf\2017.10-05\bin\mips-mti-elf-gdb.exe" -x startup.txt
```

按 ctrl+c，中断 GDB 程序的死循环。

将裁剪的结果放入 Starter_Tutorial 目录下，在终端执行以下命令。

```
Load vmlinux
```

在 gdb 中执行命令【c】，XShell 终端显示欢迎界面，并启动 Linux。

启动完成后，在 XShell 终端输入【root】，进入命令提示符界面。

11：Shell 编程。

12：观察 Shell 编程现象。

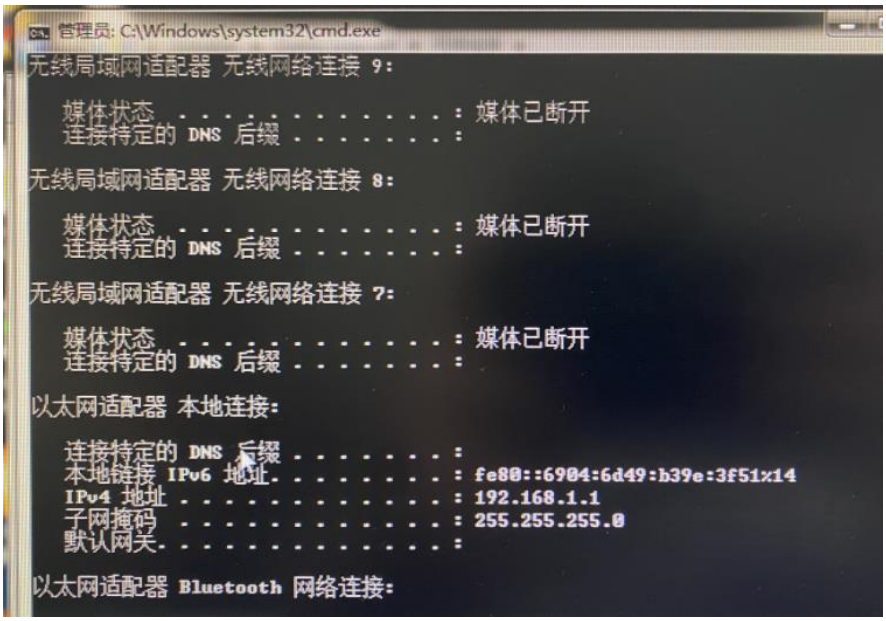
13：C 语言网络联通。

采用网线直连电脑和开发板的方式实现网络互连。

确保网线一头连接板子右侧的网卡，另一边连接计算机的网口，并且两个网口的灯都要点亮。

配置本地连接的 IPv4 中的 IP 地址为【192.168.1.1】，子网掩码为【255.255.255.0】，默认网关不进行设置。

保存后，在 cmd 窗口下执行命令【ipconfig】，确认是否配置成功，如下图所示。



在 XShell 终端执行命令【ifconfig eth0 192.168.1.2】，配置开发板的 IP 地址。

```
# ifconfig eth0 192.168.1.2
# xilinx_emaclite 10a00000.ethernet eth0: Link is Up - 100Mbps/Full - flow control rx/tx

# ifconfig eth0 192.168.1.2
# ifconfig eth0
eth0      Link encap:Ethernet  HWaddr 08:86:4C:0D:F7:09
          inet addr:192.168.1.2  Bcast:192.168.1.255  Mask:255.255.255.0
          UP BROADCAST RUNNING  MTU:1500  Metric:1
          RX packets:112 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:10006 (9.7 KiB)  TX bytes:0 (0.0 B)
          Interrupt:8 Memory:10a00000-10a0ffff

#
```

在 XShell 终端执行命令【ping 192.168.1.1】，检测开发板与主机之间的网络是否联通。

```
# ping 192.168.1.1
PING 192.168.1.1 (192.168.1.1): 56 data bytes
64 bytes from 192.168.1.1: seq=0 ttl=64 time=39.634 ms
64 bytes from 192.168.1.1: seq=1 ttl=64 time=13.537 ms
64 bytes from 192.168.1.1: seq=2 ttl=64 time=10.631 ms
64 bytes from 192.168.1.1: seq=3 ttl=64 time=10.618 ms
```

14：C 语言程序编译。

15：观察 C 语言编程现象。

五：调试过程与结果分析

（1）Shell 编程

红色灯	黄色灯	绿色灯
不用	498-505	490-497

LED 灯和 echo 命令中的编号之间的关系，如上图所示。

最右侧的绿色灯的编号是 490，最左侧的绿色灯的编号是 497。

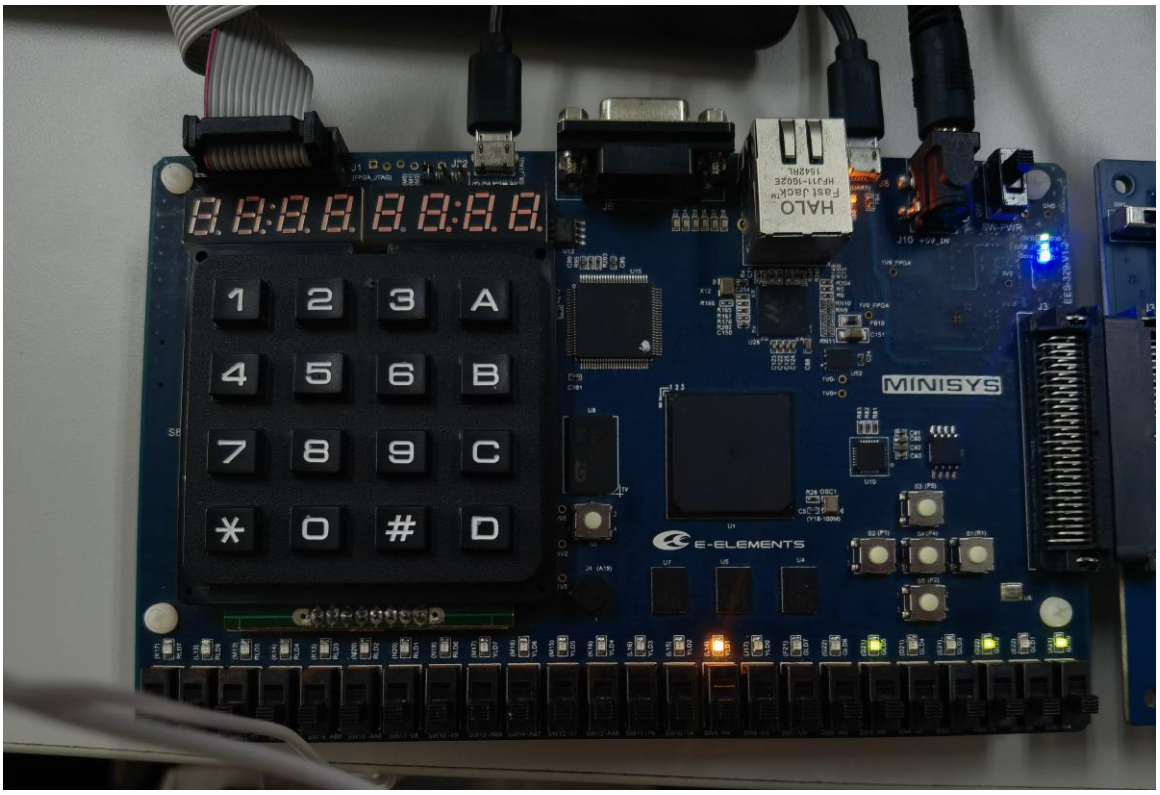
最右侧的黄色灯的编号是 498，最左侧的黄色灯的编号是 505。

在 XShell 终端执行命令【cd /sys/class/gpio/】，进入 GPIO 控制的目录下。

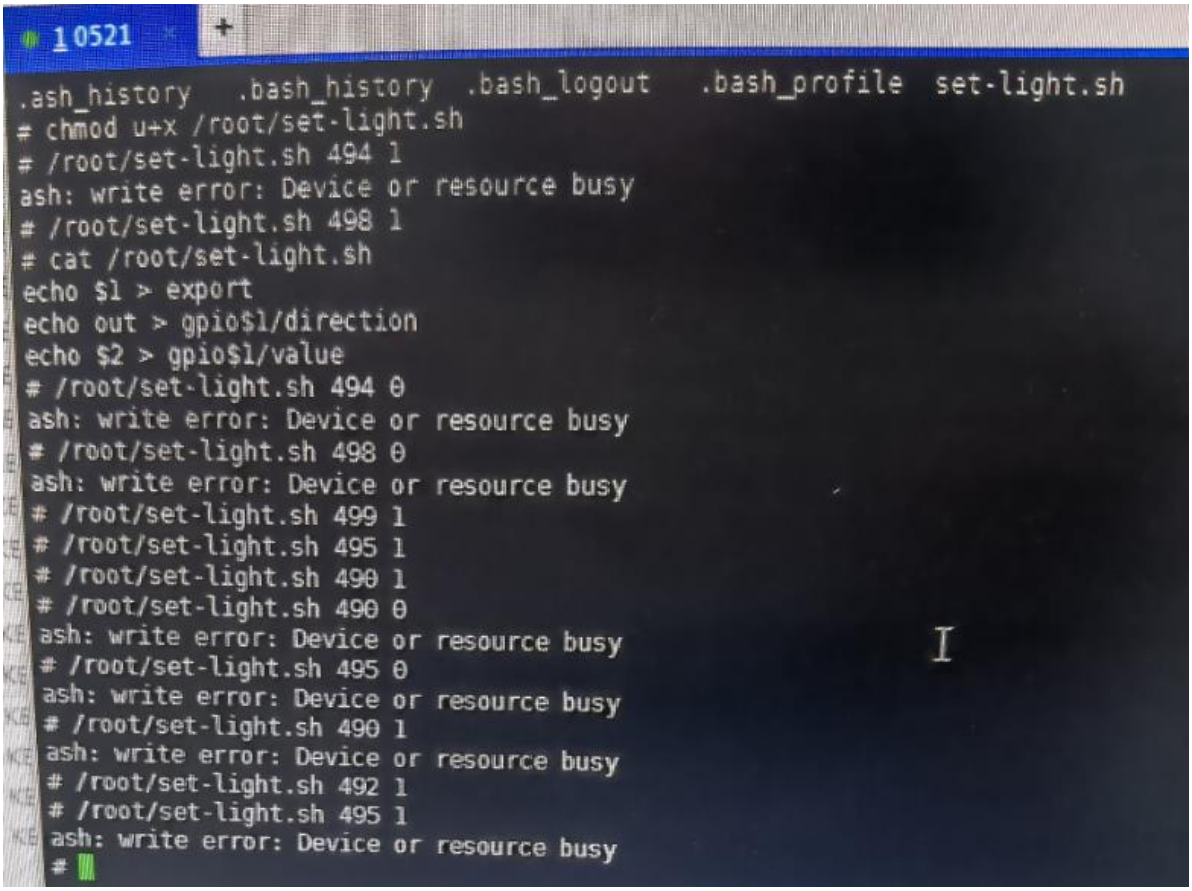
而后执行命令【ls】，可以查看 gpiochip，如下图所示。

```
# cd /sys/class/gpio/
# ls
export       gpiochip490  unexport
```

按照学号最后 4 位（0225）设置亮灯的程序。亮灯的结果，如下图所示。



在 XShell 终端执行的亮灯命令，如下图所示。



（2）C 语言编程

1：拨码开关控制 LED 灯

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <fcntl.h>   //define O_WRONLY and O_RDONLY
#include <sys/types.h>
#include <sys/stat.h>
#include <stdint.h>
#include <sys/mman.h>

volatile unsigned int * gpio;
//定义芯片引脚:
#define SYSFS_GPIO_EXPORT           "/sys/class/gpio/export"
#define SYSFS_GPIO_RST_DIR_VAL      "out"
#define SYSFS_GPIO_RST_VAL_H        "1"
#define SYSFS_GPIO_RST_VAL_L        "0"



char SYSFS_GPIO_RST_PIN_VAL[16][25] = { //端口号
    "505", "504", "503", "502", "501",
    "500", "499", "498", "497", "496",
    "495", "494", "493", "492", "491", "490"};
char SYSFS_GPIO_RST_DIR[16][50]={       //端口方向（开关输入 IN，灯输出 OUT）
    "/sys/class/gpio/gpio505/direction",
    "/sys/class/gpio/gpio504/direction",
    "/sys/class/gpio/gpio503/direction",
    "/sys/class/gpio/gpio502/direction",
    "/sys/class/gpio/gpio501/direction",
    "/sys/class/gpio/gpio500/direction",
    "/sys/class/gpio/gpio499/direction",
    "/sys/class/gpio/gpio498/direction",
    "/sys/class/gpio/gpio497/direction",
    "/sys/class/gpio/gpio496/direction",
    "/sys/class/gpio/gpio495/direction",
    "/sys/class/gpio/gpio494/direction",
    "/sys/class/gpio/gpio493/direction",
    "/sys/class/gpio/gpio492/direction",
    "/sys/class/gpio/gpio491/direction",
    "/sys/class/gpio/gpio490/direction"};

char SYSFS_GPIO_RST_VAL[16][50] = {  //端口值文件
```

```
    "/sys/class/gpio/gpio505/value",
    "/sys/class/gpio/gpio504/value",
    "/sys/class/gpio/gpio503/value",
    "/sys/class/gpio/gpio502/value",
    "/sys/class/gpio/gpio501/value",
    "/sys/class/gpio/gpio500/value",
    "/sys/class/gpio/gpio499/value",
    "/sys/class/gpio/gpio498/value",
    "/sys/class/gpio/gpio497/value",
    "/sys/class/gpio/gpio496/value",
    "/sys/class/gpio/gpio495/value",
    "/sys/class/gpio/gpio494/value",
    "/sys/class/gpio/gpio493/value",
    "/sys/class/gpio/gpio492/value",
    "/sys/class/gpio/gpio491/value",
    "/sys/class/gpio/gpio490/value"};

//函数功能：十进制转换成二进制数存入字符串
void int_to_chars(char* str) {
    int num = gpio[0];
    for (int i = 15; i >= 0; i--) {
        str[i] = num % 2 + '0';
        num /= 2;
    }
}

//打开开关设备，并设置控制方向
int init() {
    int fd = open("/dev/uio0", O_RDWR);
    if (fd < 1) {
        printf("could not open /dev/uio0\n");
        exit(EXIT_FAILURE);
    }

    //虚拟地址到物理地址的映射
    gpio = (unsigned int *)mmap(NULL, getpagesize(), PROT_READ|PROT_WRITE, MAP_SHARED,
fd, 0);

    //Controller，控制所有开关，4 位 16 进制数对应 16 个二进制位（16 个开关）
    gpio[1]  = 0xFFFF;
    int result = gpio[0]; //开关的值

    //关闭设备
    close(fd);

    return result;
```

```
}

//初始化灯全灭
int offled() {
    for(int i = 0; i < 16; i++) {
        //打开端口/sys/class/gpio# echo 48 > export
        int fd = open(SYSFS_GPIO_EXPORT, O_WRONLY);
        if(fd == -1) {
            printf("ERR: Radio hard reset pin open error.\n");
            return EXIT_FAILURE;
        }
        char *temp_str;
        temp_str = (char*)malloc(sizeof(char)*(sizeof(SYSFS_GPIO_RST_PIN_VAL[i])+1));
        strncpy(temp_str, SYSFS_GPIO_RST_PIN_VAL[i],
sizeof(SYSFS_GPIO_RST_PIN_VAL[i]));
        write(fd, temp_str ,sizeof(temp_str));
        close(fd);

        //设置端口方向/sys/class/gpio/gpio48# echo out > direction
        char *temp_str1;
        temp_str1=(char*)malloc(sizeof(char)*(sizeof(SYSFS_GPIO_RST_DIR[i])+1));
        strncpy(temp_str1,SYSFS_GPIO_RST_DIR[i],sizeof(SYSFS_GPIO_RST_DIR[i]));
///sys/class/gpio/gpio505/direction
        fd = open(temp_str1, O_WRONLY);
        if(fd == -1) {
            printf("ERR: Radio hard reset pin direction open error.\n");
            return EXIT_FAILURE;
        }
        write(fd, SYSFS_GPIO_RST_DIR_VAL, sizeof(SYSFS_GPIO_RST_DIR_VAL));  //out
        close(fd);

        //灯全部初始化为熄灭
        char *temp_str2;
        temp_str2=(char*)malloc(sizeof(char)*(sizeof(SYSFS_GPIO_RST_VAL[i])+1));
        strncpy(temp_str2,SYSFS_GPIO_RST_VAL[i],sizeof(SYSFS_GPIO_RST_VAL[i]));
///sys/class/gpio/gpio505/value
        fd = open(temp_str2, O_RDWR);
        if(fd == -1) {
            printf("ERR: Radio hard reset pin value open error.\n");
            return EXIT_FAILURE;
        }
        write(fd, SYSFS_GPIO_RST_VAL_L, sizeof(SYSFS_GPIO_RST_VAL_L));      //0 低电平
        close(fd);
    }
}
```

```c
//点亮灯
int onled(char* number){
    int i=0;
    char *temp_str2;
    for(i = 0; i < 16; i++) {
        if(number[i] == '1'){   //如果该位数字为 1，则对应的灯需要点亮
            int fd;
            temp_str2=(char*)malloc(sizeof(char)*(sizeof(SYSFS_GPIO_RST_VAL[i])+1));
            strncpy(temp_str2, SYSFS_GPIO_RST_VAL[i], sizeof(SYSFS_GPIO_RST_VAL[i]));
            fd = open(temp_str2, O_RDWR);
            if(fd == -1){
                   printf("ERR: Radio hard reset pin value open error.\n");
                   return EXIT_FAILURE;
            }
            write(fd, SYSFS_GPIO_RST_VAL_H, sizeof(SYSFS_GPIO_RST_VAL_H));  //高电平
            close(fd);
        }

        else{               //为 0 则该灯灭
            int fd;
            temp_str2=(char*)malloc(sizeof(char)*(sizeof(SYSFS_GPIO_RST_VAL[i])+1));
            strncpy(temp_str2,SYSFS_GPIO_RST_VAL[i],sizeof(SYSFS_GPIO_RST_VAL[i]));
            fd = open(temp_str2, O_RDWR);
            if(fd == -1){
                   printf("ERR: Radio hard reset pin value open error.\n");
                   return EXIT_FAILURE;
            }
            write(fd, SYSFS_GPIO_RST_VAL_L, sizeof(SYSFS_GPIO_RST_VAL_L));
            close(fd);
        }
    }
}

void light(char* number) {                  //跑马灯代码
    int k = 0;
    while(k <= 99) {
        onled(number);              //点亮灯
        //数字走马，左边第一位等于最后一位，其余取其左值
        char temp = number[15];
        for(int i = 15; i > 0; i--) {
            number[i] = number[i-1];
        }
        number[0] = temp;
        usleep(3000000); //亮灯维持 3 秒，单位是微秒
        k++;
    }
```

```
}

int main() {
    int sw = 0;
    sw = init();  //打开开关设备，并设置控制方向
    offled();     //初始化灯全灭
        while(1){
        int_to_chars(number);   //进制转换，将 gpio 结果放到 number
        onled(number);
    }       //点亮灯
    return 0;
}
```

最终的实现结果，如下图所示。

2：跑马灯

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <fcntl.h>   //define O_WRONLY and O_RDONLY
#include <sys/types.h>
#include <sys/stat.h>
#include <stdint.h>
#include <sys/mman.h>

volatile unsigned int * gpio;
//定义芯片引脚:
#define SYSFS_GPIO_EXPORT          "/sys/class/gpio/export"
#define SYSFS_GPIO_RST_DIR_VAL     "out"
#define SYSFS_GPIO_RST_VAL_H       "1"
#define SYSFS_GPIO_RST_VAL_L       "0"



char SYSFS_GPIO_RST_PIN_VAL[16][25] = { //端口号
    "505", "504", "503", "502", "501",
    "500", "499", "498", "497", "496",
    "495", "494", "493", "492", "491", "490"};
char SYSFS_GPIO_RST_DIR[16][50]={       //端口方向（开关输入 IN，灯输出 OUT）
    "/sys/class/gpio/gpio505/direction",
    "/sys/class/gpio/gpio504/direction",
    "/sys/class/gpio/gpio503/direction",
    "/sys/class/gpio/gpio502/direction",
    "/sys/class/gpio/gpio501/direction",
    "/sys/class/gpio/gpio500/direction",
    "/sys/class/gpio/gpio499/direction",
    "/sys/class/gpio/gpio498/direction",
    "/sys/class/gpio/gpio497/direction",
    "/sys/class/gpio/gpio496/direction",
    "/sys/class/gpio/gpio495/direction",
    "/sys/class/gpio/gpio494/direction",
    "/sys/class/gpio/gpio493/direction",
    "/sys/class/gpio/gpio492/direction",
    "/sys/class/gpio/gpio491/direction",
    "/sys/class/gpio/gpio490/direction"};

char SYSFS_GPIO_RST_VAL[16][50] = {  //端口值文件
    "/sys/class/gpio/gpio505/value",
    "/sys/class/gpio/gpio504/value",
    "/sys/class/gpio/gpio503/value",
```

```
    "/sys/class/gpio/gpio502/value",
    "/sys/class/gpio/gpio501/value",
    "/sys/class/gpio/gpio500/value",
    "/sys/class/gpio/gpio499/value",
    "/sys/class/gpio/gpio498/value",
    "/sys/class/gpio/gpio497/value",
    "/sys/class/gpio/gpio496/value",
    "/sys/class/gpio/gpio495/value",
    "/sys/class/gpio/gpio494/value",
    "/sys/class/gpio/gpio493/value",
    "/sys/class/gpio/gpio492/value",
    "/sys/class/gpio/gpio491/value",
    "/sys/class/gpio/gpio490/value"};

//函数功能：十进制转换成二进制数存入字符串
void int_to_chars(char* str) {
    int num = gpio[0];
    for (int i = 15; i >= 0; i--) {
        str[i] = num % 2 + '0';
        num /= 2;
    }
}

//打开开关设备，并设置控制方向
int init() {
    int fd = open("/dev/uio0", O_RDWR);
    if (fd < 1) {
        printf("could not open /dev/uio0\n");
        exit(EXIT_FAILURE);
    }

    //虚拟地址到物理地址的映射
    gpio = (unsigned int *)mmap(NULL, getpagesize(), PROT_READ|PROT_WRITE, MAP_SHARED,
fd, 0);

    //Controller，控制所有开关，4 位 16 进制数对应 16 个二进制位（16 个开关）
    gpio[1]  = 0xFFFF;
    int result = gpio[0]; //开关的值

    //关闭设备
    close(fd);

    return result;
}

//初始化灯全灭
```

```
int offled() {
    for(int i = 0; i < 16; i++) {
        //打开端口/sys/class/gpio# echo 48 > export
        int fd = open(SYSFS_GPIO_EXPORT, O_WRONLY);
        if(fd == -1) {
            printf("ERR: Radio hard reset pin open error.\n");
            return EXIT_FAILURE;
        }
        char *temp_str;
        temp_str = (char*)malloc(sizeof(char)*(sizeof(SYSFS_GPIO_RST_PIN_VAL[i])+1));
        strncpy(temp_str, SYSFS_GPIO_RST_PIN_VAL[i],
sizeof(SYSFS_GPIO_RST_PIN_VAL[i]));
        write(fd, temp_str ,sizeof(temp_str));
        close(fd);

        //设置端口方向/sys/class/gpio/gpio48# echo out > direction
        char *temp_str1;
        temp_str1=(char*)malloc(sizeof(char)*(sizeof(SYSFS_GPIO_RST_DIR[i])+1));
        strncpy(temp_str1,SYSFS_GPIO_RST_DIR[i],sizeof(SYSFS_GPIO_RST_DIR[i]));
///sys/class/gpio/gpio505/direction
        fd = open(temp_str1, O_WRONLY);
        if(fd == -1) {
            printf("ERR: Radio hard reset pin direction open error.\n");
            return EXIT_FAILURE;
        }
        write(fd, SYSFS_GPIO_RST_DIR_VAL, sizeof(SYSFS_GPIO_RST_DIR_VAL));  //out
        close(fd);

        //灯全部初始化为熄灭
        char *temp_str2;
        temp_str2=(char*)malloc(sizeof(char)*(sizeof(SYSFS_GPIO_RST_VAL[i])+1));
        strncpy(temp_str2,SYSFS_GPIO_RST_VAL[i],sizeof(SYSFS_GPIO_RST_VAL[i]));
///sys/class/gpio/gpio505/value
        fd = open(temp_str2, O_RDWR);
        if(fd == -1) {
            printf("ERR: Radio hard reset pin value open error.\n");
            return EXIT_FAILURE;
        }
        write(fd, SYSFS_GPIO_RST_VAL_L, sizeof(SYSFS_GPIO_RST_VAL_L));      //0 低电平
        close(fd);
    }
}

//点亮灯
int onled(char* number){
    int i=0;
```

```
    char *temp_str2;
    for(i = 0; i < 16; i++) {
        if(number[i] == '1'){   //如果该位数字为 1，则对应的灯需要点亮
            int fd;
            temp_str2=(char*)malloc(sizeof(char)*(sizeof(SYSFS_GPIO_RST_VAL[i])+1));
            strncpy(temp_str2, SYSFS_GPIO_RST_VAL[i], sizeof(SYSFS_GPIO_RST_VAL[i]));
            fd = open(temp_str2, O_RDWR);
            if(fd == -1){
                   printf("ERR: Radio hard reset pin value open error.\n");
                   return EXIT_FAILURE;
            }
            write(fd, SYSFS_GPIO_RST_VAL_H, sizeof(SYSFS_GPIO_RST_VAL_H));  //高电平
            close(fd);
        }

        else{               //为 0 则该灯灭
            int fd;
            temp_str2=(char*)malloc(sizeof(char)*(sizeof(SYSFS_GPIO_RST_VAL[i])+1));
            strncpy(temp_str2,SYSFS_GPIO_RST_VAL[i],sizeof(SYSFS_GPIO_RST_VAL[i]));
            fd = open(temp_str2, O_RDWR);
            if(fd == -1){
                   printf("ERR: Radio hard reset pin value open error.\n");
                   return EXIT_FAILURE;
            }
            write(fd, SYSFS_GPIO_RST_VAL_L, sizeof(SYSFS_GPIO_RST_VAL_L));
            close(fd);
        }
    }
}

void light(char* number) {                  //跑马灯代码
    int k = 0;
    while(k <= 99) {
        onled(number);              //点亮灯
        //数字走马，左边第一位等于最后一位，其余取其左值
        char temp = number[15];
        for(int i = 15; i > 0; i--) {
            number[i] = number[i-1];
        }
        number[0] = temp;
        usleep(2000000); //亮灯维持 2 秒，单位是微秒
        k++;
    }
}

int main() {
```

```
    int sw = 0;
    char number[16] = "0000000000000111";   //组号(7)
    sw = init();  //打开开关设备，并设置控制方向
//    int_to_chars(number);   //进制转换，将 gpio 结果放到 number
    onled(number);  //点亮灯
    light(number);  //跑马灯
    return 0;
}
```

最终的实现结果，如下图所示。

六：本组设计的特色

在 C 语言代码的开始阶段定义了 GPIO 端口及其对应的文件路径、方向、状态等信息。同时定义了一些常量，用于控制 GPIO 端口的方向和电平状态（高电平或低电平）。

实现关闭 LED 灯时依赖于 offled 函数。遍历所有 GPIO 端口，依次打开并设置其方向为输出（“out”），然后将所有 LED 灯的状态初始化为熄灭（低电平）。

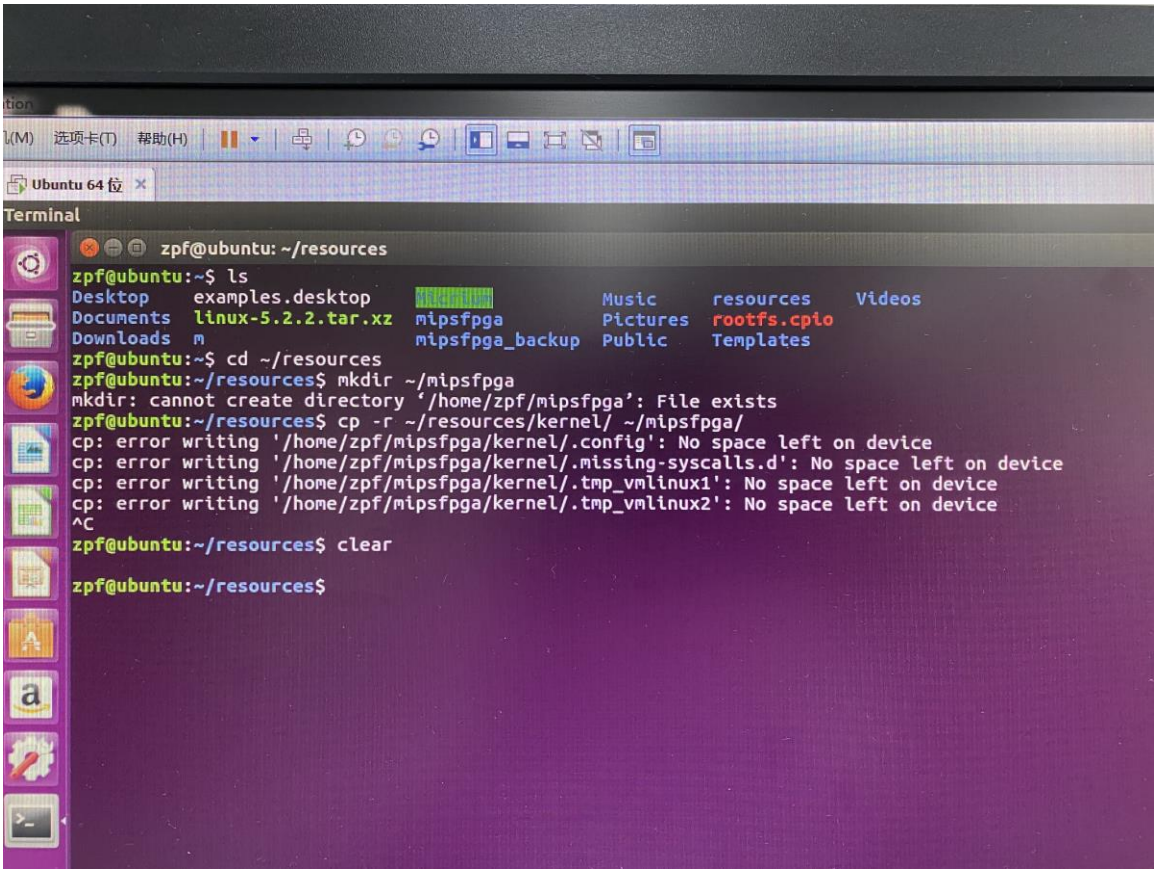
实现点亮 LED 灯时依赖于 onled 函数。根据传入的二进制字符串（如

“0000000000000111”），判断哪些位为‘1’（需要点亮对应的 LED 灯）或‘0’（保持熄灭），并相应地控制 GPIO 端口的电平状态（高电平或低电平）。

实现跑马灯效果时依赖于 light()函数。在一个循环中，不断调用 onled 函数点亮当前指定的 LED 灯组，然后将字符串中各位按顺序左移，实现“跑马灯”效果。每次循环后灯光保持亮 2 秒，然后继续循环。

七：本次设计的总结

1：在实验过程中，硬件设备可能会出现问题，应该及时进行更换。例如：minisys 的卡槽缺少焊丝、网线存在接触不良或断裂、网络通信板的灯不亮、主机的 USB 接口无法接入、主机的虚拟机启动后内存爆炸且无法删除垃圾箱中的文件（如下图所示）等等问题。



2：按照视频进行 Linux 操作系统的裁剪、编译所生成的 vmlinux 无法使用，后续需要 load 文件夹中已有的 vmlinux。

3：在 shell 编程中，echo 指令后应该紧跟从根目录开始的绝对地址，而不能用 gpio 目录下的相对地址，否则会报错。例如：echo out > /sys/class/gpio/gpio47/direction，而不是 echo out > /gpio47/direction。

4：配置完主机的 IP 地址后，应该使用 ipconfig 进行检查，有可能存在未配置成功的现象，

此时就应该检查网线等硬件是否正确地紧密连接。

八：参考文献

1：MIPSfpga SoC Docs Overview.pdf。

2：MIPSfpga SOC Starter Tutorial.pdf。

3：01_LinuxKernel.pdf。

4：02_LinuxBuildroot.pdf。

5：03_LinuxIntC.pdf。

6：04_LinuxIICEthernet.pdf。

7：05_LinuxUIO.pdf。

8：计算机系统工程综合实践-操作系统任务指导书（学生版）.pdf。

9：计算机系统工程综合实践课程-操作系统任务书-2024.pdf。

10 ：https://blog.csdn.net/wsclinux/article/details/43230499?spm=1001.2014.3001.5506

完成时间	2024/09/05	验收时间	2024/09/05	报告成绩	
所在组序号	10 组				
本组成员情况					
姓　名	学　号	承担的任务			
李盈	2021308250221	接口硬件测试，操作系统的裁剪和移植，C 语言跑马灯编程			
王馨怡	2021308250205	接口硬件测试，操作系统的裁剪和移植，C 语言亮灯编程			
夏婉可	2020301010225	接口硬件测试，操作系统的裁剪和移植，Shell 编程			
阮琰杰	2021308130110	接口硬件测试，操作系统的裁剪和移植，C 语言亮灯编程			

验 收 报 告

MIPSfpga SOC	□通过 □未通过	操作系统裁剪	□通过 □未通过
调试成绩		实验成绩	
接口电路控制	□通过 □未通过	操作系统移植	□通过 □未通过
调试成绩		实验成绩	
接口电路驱动	□通过 □未通过	LED 灯控制	□通过 □未通过
调试成绩		实验成绩	
综合成绩		综合成绩	
存在问题			
验收结论	□优秀 □良好 □中等 □及格 □不及格		
教师签名			

*100-90 优秀 89-80 良好 79-70 中等 69-60 及格 59-0 不及格