同濟大學
TONGJI UNIVERSITY

计算机系统实验课程实验报告

实验题目：应用程序开发实验报告

学号：2151769

姓名：吕博文

指导教师：郭玉臣

日期：2024.6.28

一、 实验目的

本次实验的主要目标是加深学生对微处理器设计的理解，通过实际的设计和实现过程，掌握现代处理器的关键技术和原理。在实验中，学生将学习到如何为处理器设计和实现指令集架构，并将其与存储器和其他外围设备进行集成。此外，实验还涵盖了数字逻辑设计、嵌入式系统开发和硬件描述语言（HDL）的应用，让学生在实践中理解硬件与软件的互动及其在系统性能中的作用。

通过本实验，学生不仅能够熟悉硬件编程语言，如 Verilog 或 VHDL，还将掌握使用现代 FPGA 开发板和 EDA 工具（如 Xilinx Vivado）的技能。实验的过程中，将引导学生理解微处理器内部的复杂交互和通信机制，增强对计算机科学和工程领域跨学科知识的应用能力。

此外，实验也旨在培养学生的创新思维和解决问题的能力，使他们能够根据不同的应用需求设计和优化计算机系统。通过深入学习和实践，学生将能更好地准备进入高技术领域的职业生涯，为未来的学术或工业研究打下坚实的基础。

二、 实验内容

本次实验的目标是基于第二次实验移植成功的操作系统，设计一个应用程序，通过更改操作系统中的.c 文件使得操作系统移植在开发板上时能够实现新的应用程序功能。本实验猜拳应用小程序，通过用户在开发板上控制开关表示自己的选择，而电脑则随机给出它的结果并判定比赛的结果输出。

为完成上述目标，在实验二的基础上，具体步骤如下：

- 修改μC/OS-II 操作系统的用户自定义程序 openmips.c 程序，实现应用程序开发
- 重新编译μC/OS-II 操作系统得到包含应用程序功能的 OS.bin 文件
- 结合上次实验生成的 bit 流与新的 OS.bin 文件进行下板实验，调整串口程序观察输出结果

在完成本实验的各个阶段之前，我们需要对系统环境进行适当的配置和调整，确保所有工具和资源都能顺利运行。例如，需要在 Ubuntu 系统上配置 MIPS 编译环境，这要求实验者具备一定的 Linux 操作和编程基础；在 Windows 10 系统上，需要配置 COM 串口及其相关驱动程序，而不同的驱动可能会影响通信效果；同时，Vivado 软件的不同版本可能与某些 OpenCores 提供的 IP 核存在兼容性问题，这要求实验者在选择软件版本时必须谨慎。

本实验的核心目标是通过对μC/OS-II 操作系统的移植和改造，加深对计算机系统结构、总线工作原理及串行通信协议如 UART 的理解。借助 Ubuntu 系统的交叉编译环境，本实验不仅强化了软硬件间的互动理解，也促进了从理论到实践的知识转化。

此外，本实验也旨在提升我们对 Verilog HDL 的编程技能，以及对 Vivado、ModelSim 等 EDA 工具的熟练操作。通过接触和使用 OpenCores、Xilinx 等社区资源，学生将更好地了解这些平台的基本规则和操作方法。实验的过程中，对时序问题的关注和理解将被强化，为未来的软件开发和更高级的硬件设计打下坚实的基础。

三、 实验环境

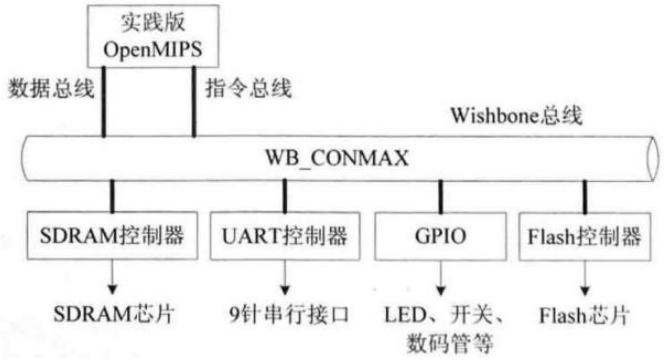
本次实验涉及的软件环境和硬件配置包括：

- **操作系统**：使用 Windows 11 位系统，版本号为 22621.3593。
- **处理器规格**：3th Gen Intel(R) Core(TM) i9-13900H 2.60 GHz
- **开发板**：选用 Digilent Nexys4 DDR 开发板。
- **设计与编程软件**：
- Vivado v2019.1（64-bit）用于 FPGA 设计与编程。
- Visual Studio Code 1.66.0 用于代码编辑。
- **仿真软件**：
- ModelSim PE 10.4c 和 Vivado ML v2021.1（64-bit）用于仿真验证。
- **辅助工具**：
  - VMware Workstation 16.1.2 build-17966106 用于运行 Ubuntu 虚拟机。
  - Sscom5.13.1 用于实现 COM 串口通信。
- **交叉编译环境**：在 VMware Workstation 16 Pro 上运行的 Ubuntu 22 操作系统中配置 MIPS 交叉编译环境。
- **IP 核使用**：
  - 使用 Memory Interface Generator 来配置 DDR2 存储器接口。
  - 使用 Clocking Wizard 来实现 DDR2 的时钟分频。

这些配置和工具将支持实验的各项需要，确保从硬件设计、软件编程到仿真测试的全面覆盖，为实验的顺利进行提供坚实的技术支持基础。

四、 实验步骤

4.1 修改 MIPS CPU，移植总线模块和操作系统



我们在实验一二中已经完成了上述总线模块的编写（总体架构如上图）和操作系统的移植。

4.2 用户任务文件修改

修改 openmips.c 的用户任务函数

TaskStart（）函数是整个操作系统主要的任务函数，在第二次实验中我们在该函数中完成了对应文本的输出，故本项目中也是着重对 TaskStart（）函数做出改变。我们在该函数中分别实现了 UI 输出、uart 输入和输出、用户任务逻辑模块。其中 UI 输出采用 uart 输出方式实现，wart 输入输出通过调用系统函数完成。用户任务逻辑模块包括通过 uart 读取开关状态，判断用户选择的猜拳结果，电脑部分随机产生猜拳结果，并通过两者之间的比对输出最终猜拳的结果。

```
void TaskStart(void *pdata)
{
    INT32U count = 0;
    INT32U hasExecuted = 0;   //  添加一个标志变量
    pdata = pdata;              /* Prevent compiler warning                    */
    OSInitTick();               /* don't put this function in main() */
    // Program START
    for (;;)
    {
        //  输出游戏规则和说明->存到  Info[]  里, GBK 编码
        if(count <= 100)
        {
            uart_putc(Info[count]);
           // uart_putc(Info[count+1]);
        }
```

```
else if (!hasExecuted)  // 只在未执行过时运行一次
        {
            INT32U data;
            data = gpio_in();
            INT32U ready = data << 31;
            INT32U choice = data >> 1;

            if(ready)  // 用户开始->按下 N17
            {
                // 闪灯提示
                uart_print_str("\n");
                gpio_out(0x80000000);

                // 获得用户输入
                INT32U u_stone = choice & 0x00000004;
                INT32U u_scissor = choice & 0x00000002;
                INT32U u_paper = choice & 0x00000001;

                // 输出用户选择
                INT32U gamer = 0;
                if(u_stone)
                {
                    gamer = 2;
                    uart_print_str("- Play:\n     (You)Rock");
```

```
        else if(u_scissor)
                        {
                            gamer = 0;
                            uart_print_str("- Play:\n     (You)Scissor");
                        }
                        else
                        {
                            gamer = 1;
                            uart_print_str("- Play:\n     (You)Paper");
                        }

                        // 电脑选择
                        INT32U computer = count % 3;   // Generate a random number between 0
and 2
                        if(computer == 0)computer = 1;
                        if (computer == 0)
                        {
                            uart_print_str(" vs (Computer)Scissor\n     Game Result:\0");
                        }
                        else if (computer == 1)
                        {
                            uart_print_str(" vs (Computer)Paper\n     Game Result:\0");
                        }
                        else
```

```
    {
                uart_print_str(" vs (Computer)Rock\n     Game Result:\0");
            }

            // 输出结果
            if (gamer == computer)
            {
                uart_print_str("     @ Tied.\n");
            }
            else if ((gamer == 0 && computer == 2) ||
                     (gamer == 1 && computer == 0) ||
                     (gamer == 2 && computer == 1))
            {
                uart_print_str("     @ Computer Won.\n");
            }
            else
            {
                uart_print_str("     @ You Won!!!\n");
            }

            hasExecuted = 1;  // 设置标志，确保只执行一次
        }

    }
```

```
            count = count + 1;
            // OSTimeDly(10);
        }
    }
```

4.4 Ubuntu 下重新编译

在本次实验中，我们使用 Ubuntu 22 系统来搭建 MIPS 编译环境。根据指导书的步骤，需要将 MIPS 编译包复制到 /opt 文件夹中。由于 /opt 文件夹默认权限可能限制写入，因此需要进行以下操作以配置环境：

- sudo chmod /opt 777 以修改文件夹访问权限；
- 将 Windows 中 mips-sde-elf-i686-pc-linux-gnu.tar.tar 拷贝到某个可访文件夹；
- cp 指令将拷贝过来的文件拷贝到 /opt 文件夹下；
- cd /opt 进入文件夹；
- tar mips-sde-elf-i686-pc-linux-gnu.tar.tar 搭建编译环境；
- cd .. 回到上一级文件夹；
- vi .bashrc 使用 vim 编辑器修改当前用户的环境变量文件；
- 具体的 Vim 指令的使用方式上网搜索，不再赘述；
- source .bashrc 对文件重新编译，就可以获得相关的编译环境。

```
extreme1228@LAPTOP1228:~/data/cpu_data/ucosii_OpenMIPS$ mips-sde-elf-
mips-sde-elf-addr2line  mips-sde-elf-cpp        mips-sde-elf-gdbtui     mips-sde-elf-ranlib
mips-sde-elf-ar         mips-sde-elf-g++        mips-sde-elf-gprof      mips-sde-elf-readelf
mips-sde-elf-as         mips-sde-elf-gcc        mips-sde-elf-ld         mips-sde-elf-run
mips-sde-elf-c++        mips-sde-elf-gcc-4.3.2  mips-sde-elf-nm         mips-sde-elf-size
mips-sde-elf-c++filt    mips-sde-elf-gcov       mips-sde-elf-objcopy    mips-sde-elf-strings
mips-sde-elf-conv       mips-sde-elf-gdb        mips-sde-elf-objdump    mips-sde-elf-strip
```

因为涉及了操作系统文件的更改，我们需要先进行`make clean`操作清除缓存文件。

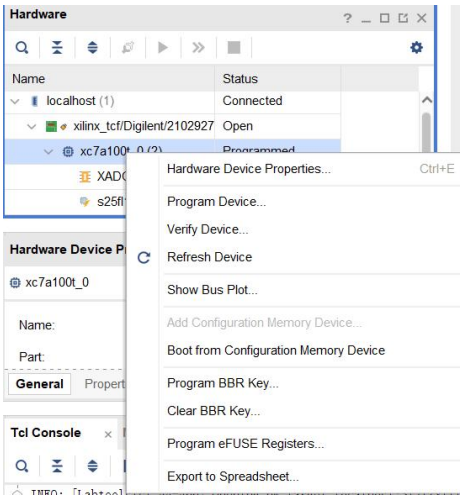
```
extreme1228@LAPTOP1228:~/data/cpu_data/ucosii_OpenMIPS$ make clean
find . -type f \
        \( -name 'core' -o -name '*.bak' -o -name '*~' \
        -o -name '*.o'  -o -name '*.tmp' -o -name '*.hex' \
        -o -name 'OS.bin' -o -name 'ucosii.bin' -o -name '*.srec' \
        -o -name '*.mem' -o -name '*.img' -o -name '*.out' \
        -o -name '*.aux' -o -name '*.log' -o -name '*.data' \) -print \
        | xargs rm -f
rm -f System.map
```

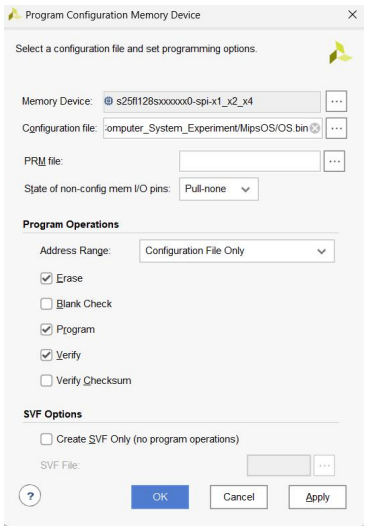
之后我们进行`make all`重新编译生成`OS.bin`文件

```
extreme1228@LAPTOP1228:~/data/cpu_data/ucosii_OpenMIPS$ make all
make[1]: Entering directory '/home/extreme1228/data/cpu_data/ucosii_OpenMIPS/common'
make[1]: '.depend' is up to date.
make[1]: Leaving directory '/home/extreme1228/data/cpu_data/ucosii_OpenMIPS/common'
make[1]: Entering directory '/home/extreme1228/data/cpu_data/ucosii_OpenMIPS/ucos'
make[1]: '.depend' is up to date.
make[1]: Leaving directory '/home/extreme1228/data/cpu_data/ucosii_OpenMIPS/ucos'
make[1]: Entering directory '/home/extreme1228/data/cpu_data/ucosii_OpenMIPS/port'
make[1]: '.depend' is up to date.
make[1]: Leaving directory '/home/extreme1228/data/cpu_data/ucosii_OpenMIPS/port'
make[1]: Entering directory '/home/extreme1228/data/cpu_data/ucosii_OpenMIPS/common'
make[1]: Nothing to be done for 'all'.
make[1]: Leaving directory '/home/extreme1228/data/cpu_data/ucosii_OpenMIPS/common'
make[1]: Entering directory '/home/extreme1228/data/cpu_data/ucosii_OpenMIPS/ucos'
make[1]: Nothing to be done for 'all'.
make[1]: Leaving directory '/home/extreme1228/data/cpu_data/ucosii_OpenMIPS/ucos'
make[1]: Entering directory '/home/extreme1228/data/cpu_data/ucosii_OpenMIPS/port'
make[1]: Nothing to be done for 'all'.
make[1]: Leaving directory '/home/extreme1228/data/cpu_data/ucosii_OpenMIPS/port'
mips-sde-elf-gcc -Tram.ld -o ucosii.om common/common.o ucos/ucos.o port/port.o -nostdlib -lgcc -e 256
mips-sde-elf-objcopy -O binary ucosii.om ucosii.bin
mips-sde-elf-objdump -D ucosii.om > ucosii.asm
./BinMerge.exe -f ucosii.bin -o OS.bin
```

4.5 uc/OS 操作系统移植

编译完成后，将 OS.bin 文件放入 Vivado 的 Memory Device 中

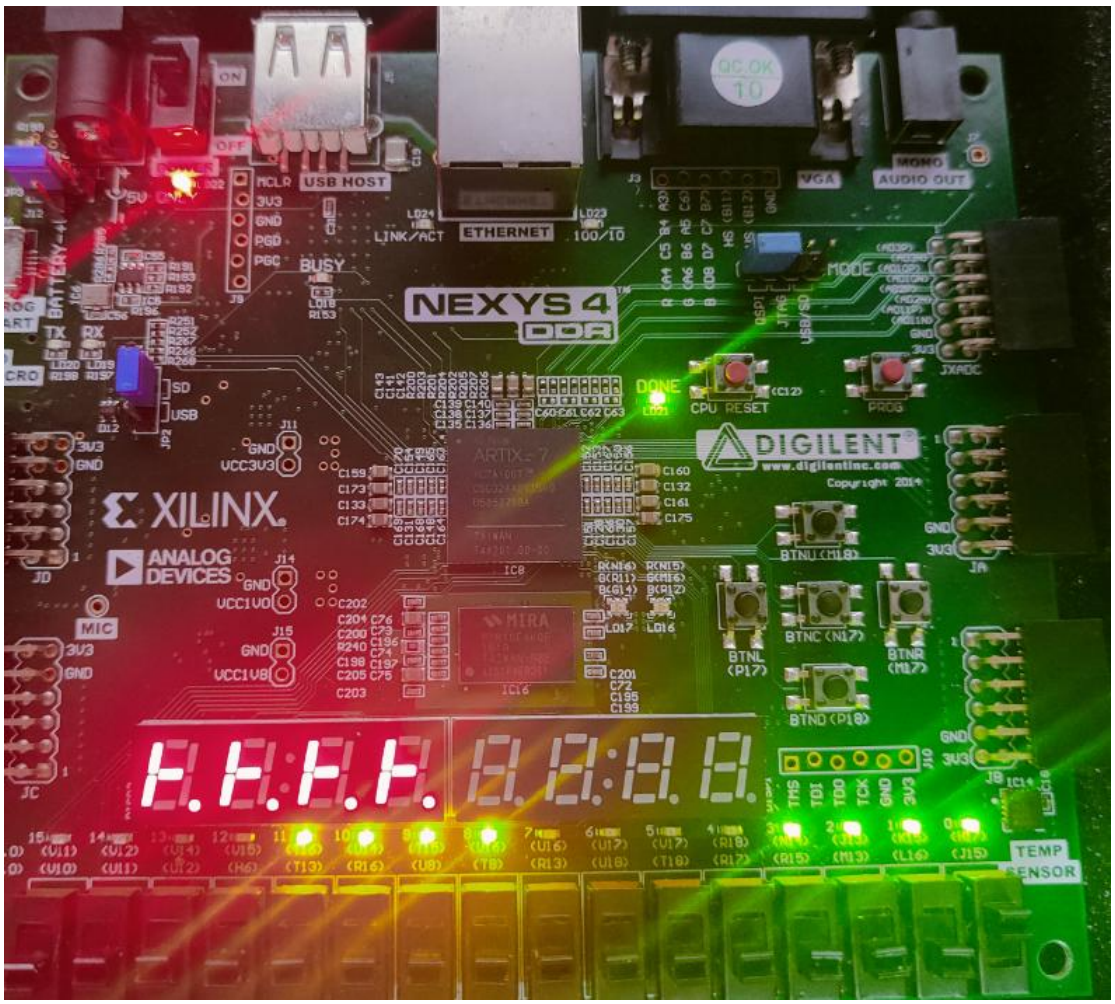




之后打开串口调试程序，调节波特率与下板系统一致（9600），选择对应的 USB 端口，下板后拨动开关即可看到串口调试程序中得到对应结果。

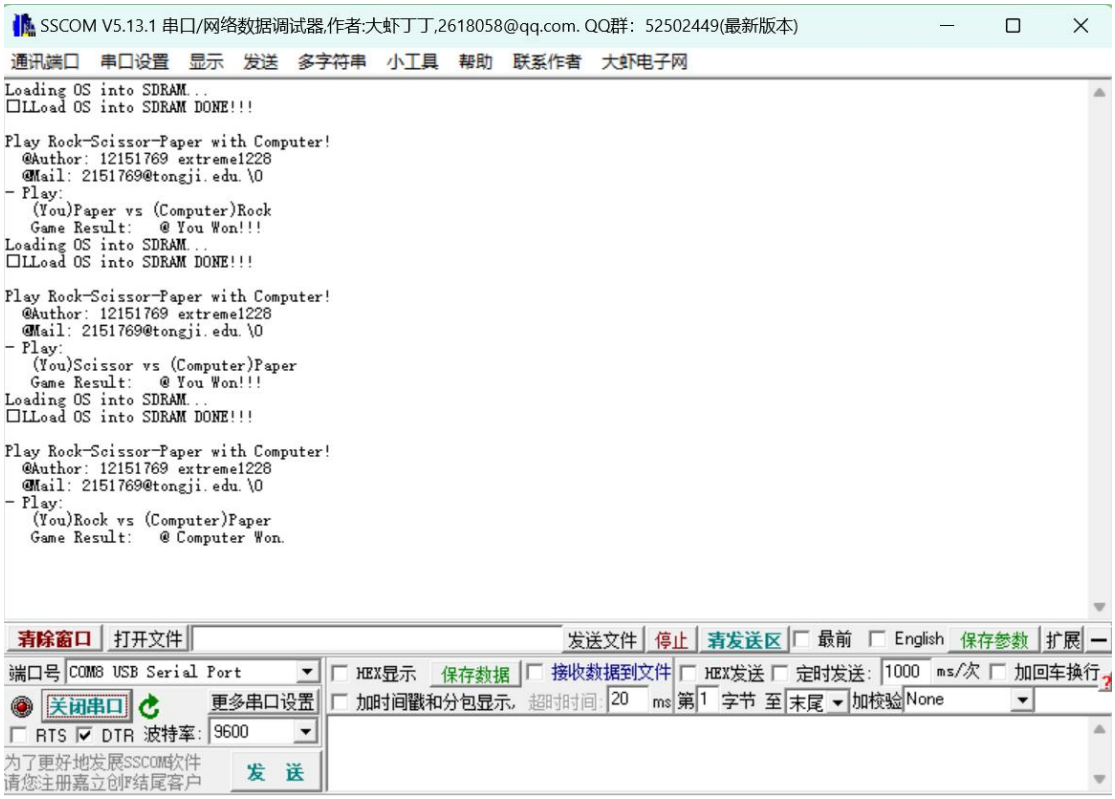
五、 实验结果

实验中，在最终下板的顶层文件中添加了总线分配的指示器，可以验证总线与其 他设备通信是否正常，在测试中结果如下：



本次实验项目中，最右侧开关（J15）表示是否开始游戏，之后向左三个开关（L16、M13、R15）分别表示剪刀、石头、布，（N17 开关代表开始游戏）数码灯亮表示游戏处于就绪状态。

我们在开发板上进行操作，串口调试程序中可以看到的结果如下：



六、 总结与体会

本次实验通过对 MIPS CPU 的移植与总线模块及操作系统的集成，成功实现了猜拳游戏应用程序。通过对操作系统用户任务函数的修改，应用程序能够在开发板上实现交互功能，用户可以通过拨动开关进行选择，电脑则随机产生结果并判定比赛结果，输出到串口调试程序中显示。

在实验过程中，我们经历了以下几个关键步骤：

- 修改 MIPS CPU 与移植总线模块：在实验一二中，我们完成了总线模块的编写及操作系统的移植，为本次实验奠定了基础。
- 用户任务文件修改：对操作系统的 TaskStart 函数进行修改，增加了用户选择与电脑选择的逻辑，实现了基本的猜拳游戏功能。
- Ubuntu 下重新编译：搭建了 MIPS 编译环境，进行了操作系统文件的修改和重新编译，生成了包含应用程序功能的 OS.bin 文件。
- uc/OS 操作系统移植：将编译生成的 OS.bin 文件放入 Vivado 的 Memory Device 中，并通过串口调试程序进行验证。
- 实验结果验证：通过开发板的开关操作和串口调试程序的输出，验证了猜拳游戏应用程序的正确性。

在实验过程中，我们不仅掌握了 MIPS CPU 和总线模块的设计与实现方法，还通过对操作系统文件的修改，深入理解了操作系统的工作原理。同时，通过对 Verilog HDL 编程和 Vivado、ModelSim 等 EDA 工具的使用，提升了我们对软硬件协同设计的能力。

本次实验的顺利完成，使我们对计算机系统结构和总线工作原理有了更深刻的理解，并增强了我们在跨学科知识应用方面的能力。通过理论与实践的结合，我们不仅巩固了所学知识，也为未来从事高技术领域的职业生涯打下了坚实的基础。