# 计算机系统实验课程报告

## ——操作系统移植实验


院　　系　计算机科学与技术学院

专　　业　计算机科学与技术

姓　　名　张宇

学　　号　2251745

指导老师　郭玉臣

完成日期　2025 年 6 月 13 日

# 目录

# 一、 实验目的

本实验的主要目标是将 μC/OS−II 操作系统移植到 NEXYS DDR4 开发板上，为后续的程序开发提供基础。移植过程遵循《自己动手写 CPU》一书中的方法。

移植的操作系统必须是嵌入式实时操作系统，其设计目标是执行特定的任务，以实现系统的小型化和成本降低。同时，操作系统需要能够及时响应外界事件和数据变化，并在规定时间内处理任务，确保任务之间的协调一致性，因此快速响应和高可靠性是其核心特点。

μC/OS−II 操作系统在多个领域得到广泛应用，包括通信设备、电力系统、飞行器等，并且通过了美国航空管理局的认证，证明其具有极高的可靠性。该系统采用 ANSI C 语言编写，并辅以少量汇编代码，支持多种微处理器架构。

本实验将在已有的 89 条指令集 CPU 基础上，进一步添加 Wishbone 总线、GPIO、UART 控制器、Flash 控制器和 SDRAM 控制器等硬件组件，并通过在 Ubuntu 上建立交叉编译环境，对 μC/OS−II 进行修改和编译，最终成功将其移植到 NEXYS 4 DDR 开发板上并进行验证。

# 二、 实验内容

本次实验要求在 Windows 11 操作系统环境下，使用 Vivado 集成开发环境，基于 Xilinx 公司提供的 NEXYS4 DDR 开发板进行操作系统的移植。上一阶段的实验已成功完成了 89 条 MIPS 指令集的 CPU 改造，并验证了实验结果的正确性。然而，尽管该 CPU 已成功实现了基本的 MIPS 指令，它仍然缺少操作系统移植所需的核心组件，如内存管理单元，因此无法完成操作系统的完整移植。

为了实现操作系统的移植，本次实验选择了能够嵌入到 FPGA 中的 MicroBlaze 软核作为处理器，并采用了支持 MicroBlaze 软核的 $\mu$C/OS-II 操作系统。该操作系统具有简单的架构，适合嵌入式开发，并能够顺利支持 MicroBlaze 软核的功能。

# 三、 实验环境

## 3.1 硬件环境

Xilinx Nexys 4 DDR Artix-7 开发板

## 3.2 软件环境

操作系统：Windows 11 (64-bit)

实验环境：Vivado v2019 (64-bit)

移植的操作系统：$\mu$C/OS -II

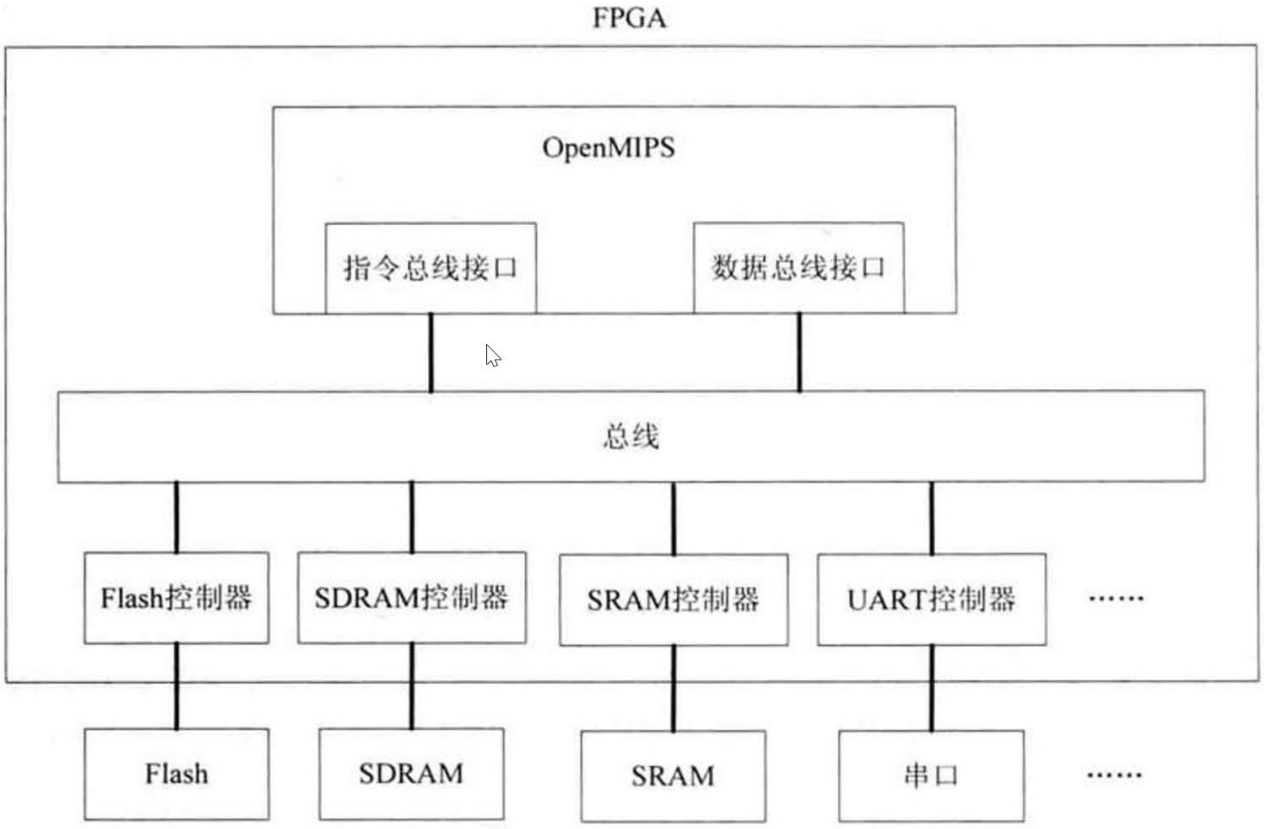
CPU 软核 IP：Micro Blaze

# 四、 实验步骤

## 4.1 实现 89 条 OpenMIPS

在实验一 CPU 改造实验中完成了 89 条 MIPS CPU 的实现，成功实现了 CP0 协处理器和异常处理功能。

为了方便的接入新设备而不需要频繁修改 CPU 的接口，可以将 CPU 通过总线接口模块挂载到总线上，各种设备也可以通过总线进行挂载。本次实验的总框架如下图所示：




具体实现步骤包括：

1. 修改处理器接口，将指令存储器和数据存储器的接口替换为 WishBone 总线接口。

2. 实现 WishBone 总线接口模块，将处理器对数据和指令的访问信号转化为标准的 WishBone 总线信号，模块在 wishbone_bus_if.v 文件中定义。

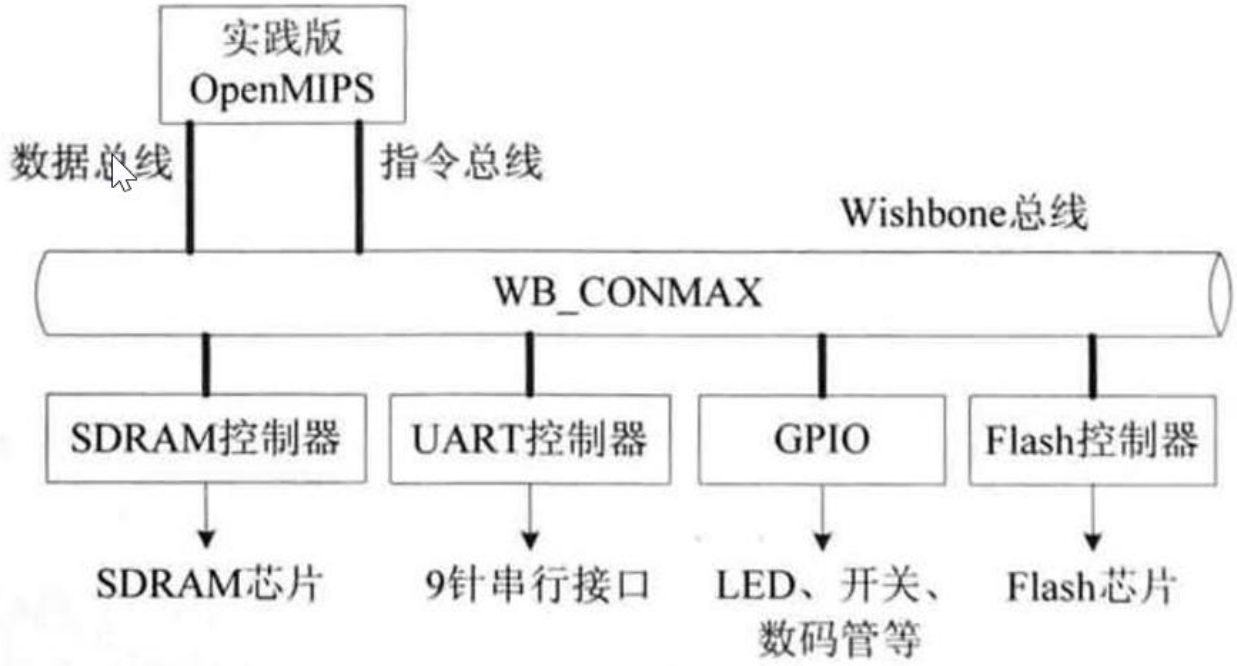
3. 修改处理器内部的一些模块和连接，不再通过 PC 直接访问指令存储器，而是通过 WishBone 总线接口来访问，这样可以使从 flash 和 SDRAM 中获取指令和数据的时间得到合理控制，避免了访存冲突问题。

## 4.2 为 CPU 添加 Wishbone 总线

PC 机一般都提供 PCI 插槽，各种板卡（包括显卡、语音卡、网卡甚至是用户自制的板卡），只要组装 PCI 接口标准，就可以直接插在 PC 机的 PCI 插槽中，十分方便。同样的道理，目前有很多 IP 核的开发者或公司，为了方便研究或公司 IP 核能够连接，就要求这些 IP 核共

享的接口标准。在系统（SoC）中，处理器与其他 IP 核通过总线互联，这些 IP 核必需遵守同样的总线规则。总线规范定义了 IP 核之间的接口。当前常见的总线规范有 ARM 公司的 AMBA，IBM 公司的 CoreConnect，以及这些公司所定义的 Wishbone。Wishbone 总线规范是 Silicore 公司最近推出的，由于其开源性，既适合不少用户群，特别是一些免费的 IP 核，大多数都采用 Wishbone 规范。Wishbone 除了开放、免费，还很简单、灵活、轻便，还支持用户自定义签名的特性。目前有 B4 版的规范，OR1200 中运信的就是 Wishbone B2 与 B3 版本的规范，用户口自配更是采用 B2 还是 B3 规范。Wishbone 有多种互联方式：点对点、数据传输、共享总线，及互联方式等。OR1200 内核使用的都是点对点连接方式。在点对点连接方式中，有一个主控制器，一个只有一个连接关系的从端所采用。

在此次实验中，采用交叉互联方式建立 Wishbone 总线，其结构如下图所示。在 Wishbone 总线上挂接了五个模块：OpenMIPS 处理器、GPIO、UART 控制器、Flash 控制器、SDRAM 控制器。其中 Wishbone 总线使用的是 OpenCores 站点提供的开源项目 WB_CONMAX，这是一个 Wishbone 总线互联阵列，采用的是交叉互联方式，允许多个主机设备同时进行通信。




## 4.3 添加 GPIO 模块

GPIO 是以位为单位进行数字输入输出的 IO 接口，作为单纯的通用输入/输出 IO，输入时从外部读取输入信号，输出时将写入的值输出到外部，处理器通过 GPIO 可以与各种设备相连接，例如 LED、开关、七段数码管等。在实践版 OpenMIPS 设计过程中，采用 OpenCores 站点提供的开源项目 GPIOIPCore，添加整个系统中。

## 4.4 添加 UART 控制器

UART（通用异步收发器）是广泛使用的串行数据传输协议，主要功能是将并行的有效数据字节转换为串行数据进行发送，同时将接收到的串行数据转化为并行数据。数据的传输格式包括起始位、数据位、奇偶校验位和停止位。传输开始时，首先会发出一个低电平信号作

为起始位，接着是有效的数据位，这些数据位可以是 4、5、6、7 或 8 位，并且按最低位优先的顺序传输。数据位后会有一个奇偶校验位，用于判断数据的传输是否正确，确保数据中“1”的个数为偶数或奇数（取决于奇偶校验类型）。最后是停止位，用来标识数据帧的结束。

UART 采用波特率来表示传输速率，波特率即单位时间内的数据变化速率，常见的波特率有 9600、19200、38400 等。接收器的采样频率通常是发送器波特率的 16 倍，以确保数据正确接收。UART 控制器通过高频采样来判断数据的每一位，逐位接收数据并通过校验位来验证传输的正确性。

在实验设计中，UART 控制器的实现采用了 OpenCores 提供的开源项目 uart16550 IP Core。该控制器支持标准的 UART 通讯格式，并通过 Wishbone 总线接口与其他模块互联，保证了数据的正确传输和系统的稳定运行。

## 4.5 添加 Flash 控制器

在嵌入式系统中，我们通常需要一个非易失性存储器来存储 BootLoader 程序和操作系统的二进制代码。Flash 存储在系统中扮演着与 CPU 协同工作的重要角色，它与其他模块不同，通常用于存储代码和数据。此次实验中使用了 Nexys4 DDR 模块的 SPI FLASH 模块。

在实现 OpenMIPS 的设计过程中，加入了一个 SPI 接口的 Flash 控制器。该控制器实现了与 Flash 存储器的通信，主要用于读取数据，包括 BootLoader 程序和操作系统的二进制文件。Nexys4 DDR 使用的 SPI Flash 芯片的具体型号为 S25FL128S。

SPI 是一种串行总线协议，通过时钟信号（sck）、片选信号（cs）、数据输入（si）和数据输出（so）实现数据的传输。SPI 通信中的时钟信号控制数据的同步传输，cs 信号用于选择通信的设备，si 信号是输入数据，so 信号用于输出数据。Flash 存储器通常用来保存大量的非易失性数据，并可用于存储程序代码。

Flash 存储器内部包含控制逻辑、数据路径、SRAM 存储区等模块，采用 EEPROM 类似的结构，适用于频繁读写操作的数据存储。Flash 的操作指令包括读取、擦除和写入等，控制器根据这些指令执行相关的操作。

由于本次实验中仅需要读取数据，主要实现了读取指令：READ。根据 S25FL128S 的文档，Flash 芯片的读取过程需要特定的时序。在操作中，首先是片选信号被拉低，然后数据通过 SPI 总线进行传输，读取操作在 50MHz 的频率下进行。

## 4.6 添加 SDRAM 控制器

SDRAM(Synchronous Dynamic RandomAccess Memory)）是同步动态随机访问存储器，同步是指 Memory 工作需要同步时钟，内部命令的发送与数据的传输都以它为基准; 动态是指存储阵列需要不断地刷新以保证数据不丢失; 随机访问是指数据不是线性依次读写，而是可以自由指定地址进行读/写。与《自己动手写 CPU》书上实现的 SDRAM 不同，此次实验中使用的是 Nexys4DDR 板载的 DDR2 模块。此次实验中，使用 DDR2 模拟一块一次最小读写宽度 8bits，最大读写宽度 32bits 的内存。

NEYXS4DDR 里的 DDR2SDRAM 型号为 MT47H64M16HR-25:H，由于 DDR 协议比较复杂，Xilinx 提供了一个简化控制 DDR 的内存控制器 IP 核，MemoryInterface Generator(MIG)，此次实验中就使用这个 IP 核来进行操作。

DDR 是 Nexys4 板上的 Nexys4 板子上的一个资源，容量为 128M。DDR17 的操作非常复杂，需要借助一个 IP 核 MemoryInterfaceGenerator，简称 MIG（当然用了 MIG 还是非常复杂，需要多级封装）。MIG 能够封装 DDR 的物理层信号，用户不需要对物理信号有所了解，只需要关注应用信号。MIG 需要按照资料中指导的操作生成。但是由于 DDR 相较于 cpu 属于慢速的设备，所以要协调工作的话还要靠下面的 MIG 控制器。IP 核生成后，需要将 DDR 的物理接口添加到顶层模块，但无需再在 XDC 文件中配置端口，因为生成过程中已经配置好了。

FPGA 如果需要对 DDR 进行读写，则需要一个 DDR 的控制器。DDR 控制器的时序主要有控制信号、写操作和读操作三个。

## 4.7 利用 Ubuntu 建立交叉编译环境

在这里与《自动写 CPU》方法不同，将在 Ubuntu 下配置支持 MIPS 编译的 GNU 对应 uC/OS-II 操作系统进行编译，生成对应的 OS.bin，支持实践版 OpenMIPS 执行的二进制文件。其构建交叉编译环境的具体过程如下：

1. 将《自动写 CPU》tools 下提供的 GNU 工具链安装文件 mips-sde-elf-i686-pc-linux-gnu.tar 复制到 Ubuntu 系统的 /opt 目录并解压；复制文件到 Ubuntu 系统中。

文件包含：Bootloader 文件夹，ucosii_OpenMIPS 文件及编译工具。

2. 修改环境变量，在 .bashrc 中加入编译工具 PATH，确保系统能够访问到该编译工具。
3. 重启 Ubuntu 系统，完成源码编译；安装 lib32z1、libc6-dev-i386 库。输入 mips-sde-elf 并双击 Tab，出现 MIPS 编译指令说明 GNU MIPS 工具安装成功。
4. 完成上述操作，Ubuntu 下的交叉编译环境配置成功。

## 4.8 对 μC/OS−II 操作系统改写编译

1. 修改 Bootloader.S 并编译

首先，修改 Bootloader.S 文件，然后进行编译生成对应的 BootLoader.o 文件。执行 make all 命令时，编译过程会将 BootLoader.S 转换为可执行的 BootLoader.asm 文件。make all 对 BootLoader 进行编译，生成可执行文件 BootLoader.o，将其转换为执行文件 BootLoader.asm（在 make all 中已经实现）

2. 修改 openmips.h 中的时钟频率并编译

接下来，需要修改 openmips.h 文件中的时钟频率，并进行编译。由于 Ubuntu 系统权限控制的原因，需要更新 BinMerge.exe 文件的执行权限，然后执行 make all 进行编译。chmod +x BinMerge.exe 同时，由于依赖文件中的路径没有被修改，所以需要将 ucosii_OpenMIPS/.depend 中的路径进行修改，然后再次执行 make all。

至此，得到二进制文件，将 OS.bin 写入板子上的 Flash，当 OpenMIPS 运行时，会首

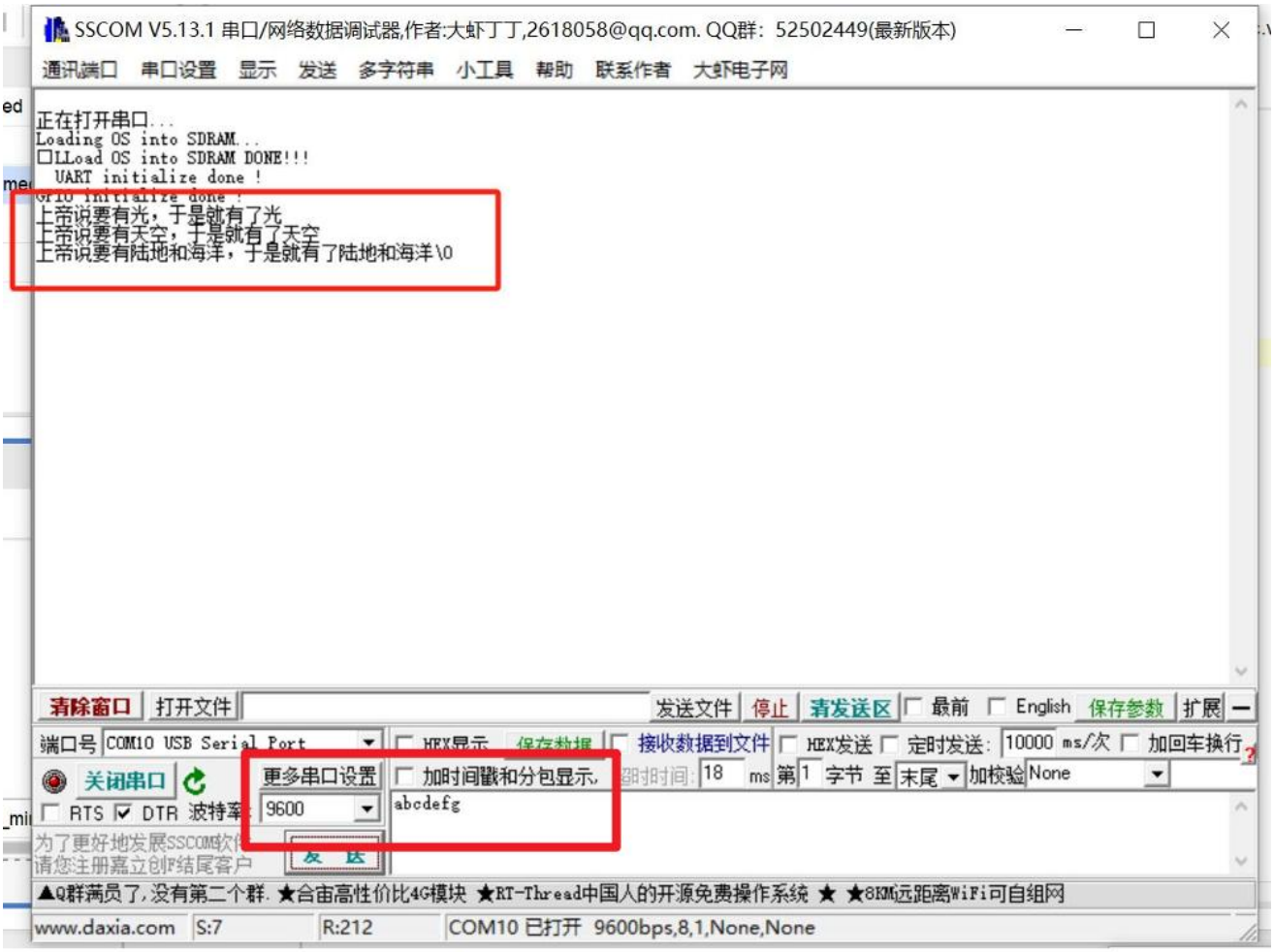
先运行 BootLoader，后者将 μC/OC-的代码复制到 SDRAM 中，然后跳转到 SDRAM，把控制权交给 μC/OS−II，于是 μC/OS−II 就运行起来了。

# 五、 实验验证

将文件 OS.bin 写入板子上的 SPIflash 中，打开 PC 上的串口程序，将参数设置为 9600bps、8 位数据位、没有奇偶校验位、1 位停止位。拨动开关复位 OpenMIPS，然后启动 OpenMIPS，串口程序将得到结果，汉字每隔 100ms 显示一个，另外 4 个 7 段数码管的显示大概每隔 100ms 变化一次。由此可知，BootLoader 加载 μC/OS−II 成功，μC/OS−II 工作正常、移植成功。具体结果展示如下。

## 5.1 串口通讯

配置 SPIflash 中的数据为 CμC/OS−II 操作系统的二进制文件 OS.bin，将 bit 文件下板，打开串口工具，设置波特率为 9600，取消加时间戳和分包显示。串口通信成功




## 5.2 数码管显示

观察开发板，点击开发板 reset，观察 PC 中串口监视工具 SSCOM 是否接收到信息


## 六、 实验总结

本次实验的主要目标是将 μC/OS−II 操作系统成功移植到 NEXYS 4 DDR 开发板上，并在 FPGA 上实现 OpenMIPS 处理器的设计。实验过程中，我们首先完成了 89 条 MIPS 指令集的 CPU 改造，并为操作系统的移植添加了多个硬件组件，最终实现了 μC/OS−II 操作系统的完整移植和验证。

在硬件设计方面，本次实验依赖于 Xilinx 提供的 NEXYS 4 DDR 开发板，使用了 FPGA 上的软核 MicroBlaze 处理器进行实验。实验过程中，首先完成了 MIPS 指令集的改造，通过添加 CP0 协处理器来实现异常处理功能。为了将不同外设接入系统，我们引入了 Wishbone 总线协议，并通过该总线连接了多个外设模块，包括 GPIO、UART 控制器、Flash 控制器以及 SDRAM 控制器等。

通过在 FPGA 上实现这些模块，使得 CPU 不仅具备了执行基本指令的能力，还能够通过总线与外部设备进行高效的数据交换。尤其是在增加 UART 和 GPIO 控制器之后，系统能够与外部世界进行有效的串行数据通信，并能够直接控制连接到 GPIO 的外设如 LED、开关以及七段数码管等。

在软件部分，实验首先在 Ubuntu 系统中配置了交叉编译环境。使用的是 MIPS 架构的 GNU 工具链，通过下载并解压安装了相应的编译工具，设置环境变量，使得系统能够支持

μC/OS−II 操作系统的编译。交叉编译环境的配置是移植过程中的关键步骤，它确保了我们能够在开发板上运行特定的操作系统，同时避免了直接在开发板上编译带来的性能和资源限制。

接下来，我们对 μC/OS−II 操作系统进行了一系列的修改和调整，主要包括修改 Bootloader 以支持从 Flash 存储器中加载操作系统，并调整时钟频率配置以匹配目标硬件。在编译过程中，遇到了一些路径和权限问题，需要通过修改路径配置和更新文件权限来确保编译能够顺利完成。最终，我们成功地将编译生成的 OS.bin 文件写入到开发板上的 SPI Flash 中，并通过串口通信验证了系统是否能够正常启动和运行。

完成操作系统的移植后，我们在开发板上进行了充分的验证。通过将 OS.bin 文件写入开发板的 Flash 存储器，并通过串口工具进行实时监测，能够看到操作系统正常启动。具体表现为，在串口监控工具中，系统每 100ms 显示一个汉字，并且 4 个七段数码管的显示内容在每 100ms 内发生变化。这表明 μC/OS−II 操作系统已经成功地从 Flash 启动，并能够在开发板上稳定运行。

验证过程中，我们还检查了串口通信的稳定性，确保数据能够正确地传输并显示在计算机端的串口工具中。此外，七段数码管也成功显示了不同的数字，证明硬件模块的连接和功能正常。

通过本次实验，我深入理解了嵌入式操作系统的移植过程以及硬件和软件之间的密切配合。在硬件设计方面，添加 Wishbone 总线以及各个外设模块（如 UART、GPIO 和 Flash 控制器）增强了系统的功能，使得系统能够与外部设备进行交互。这些模块的集成不仅提升了系统的扩展性，还让我们能够在操作系统层面进行更加复杂的功能实现。

在软件方面，通过配置交叉编译环境，我进一步理解了如何将操作系统移植到特定硬件平台上，特别是如何修改操作系统源代码以适应硬件的要求。操作系统的修改和编译过程让我深刻认识到，操作系统不仅仅是一个独立的软件系统，它与硬件的交互是密切而复杂的，必须根据硬件平台的特性进行相应的调整。

此外，整个实验过程中遇到的编译环境配置、路径问题、时钟配置等问题，也让我积累了宝贵的调试经验。通过这些问题的解决，我对嵌入式系统的开发流程有了更深入的理解。

本实验通过将 μC/OS−II 操作系统移植到 NEXYS 4 DDR 开发板上，为嵌入式系统的学习和开发提供了一个实用的实践案例。通过本次实验，我不仅提升了硬件设计和软件编程的能力，还学会了如何协调硬件和软件，确保系统能够高效运行。移植过程中遇到的各种问题和挑战，也为我后续的嵌入式开发奠定了更为坚实的基础。未来，我希望能将所学知识应用到更复杂的嵌入式系统开发中，进一步提高自己的开发能力。