同濟大學
TONGJI UNIVERSITY

计算机系统实验课程实验报告

实验题目：操作系统移植实验报告

学号：2151769

姓名：吕博文

指导教师：郭玉臣

日期：2024.5.30

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已经实现的五级动态流水线 CPU，进一步扩展其功能，包括集成 Wishbone 总线、GPIO 从设备、UART 串行通信接口、Flash 或 SD 卡控制器，以及 SDRAM 控制器。实验将在 Ubuntu 系统中建立 MIPS 交叉编译环境，对 μC/OS-II 操作系统进行必要的修改和编译，包括添加 Boot Loader，以支持系统在 Digilent NEXYS-4 DDR 开发板上的运行。此外，实验还包括通过电脑的 COM 串口与板载系统进行通信的功能实现。

为完成上述目标，实验的具体步骤如下：

- 修改现有 CPU 的取指和数据地址逻辑，以适应新的硬件结构；
- 在 CPU 中集成 Wishbone 总线接口，确保模块间的高效通信；
- 在总线结构中添加 GPIO 和 UART 通信模块，并处理相关的中断逻辑；
- 实现 Flash 控制器（推荐使用 SPI Flash 串行控制器），或集成 SD 卡控制器（支持 SDHC 或更高版本）；
- 集成 DDR2 内存控制器，以支持更大的存储需求和更快的访问速度；
- 在 Ubuntu 系统中配置和测试电脑的 COM 串口，确保与开发板之间的通信；
- 进行 Wishbone 总线的功能测试，编译并运行 GPIO 和 UART 的测试程序；
- 安装并测试 SimpleOS 操作系统，执行交叉编译及 Boot 加载流程；
- 对 μC/OS-II 操作系统进行交叉编译，完成操作系统的板载移植和验证。

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

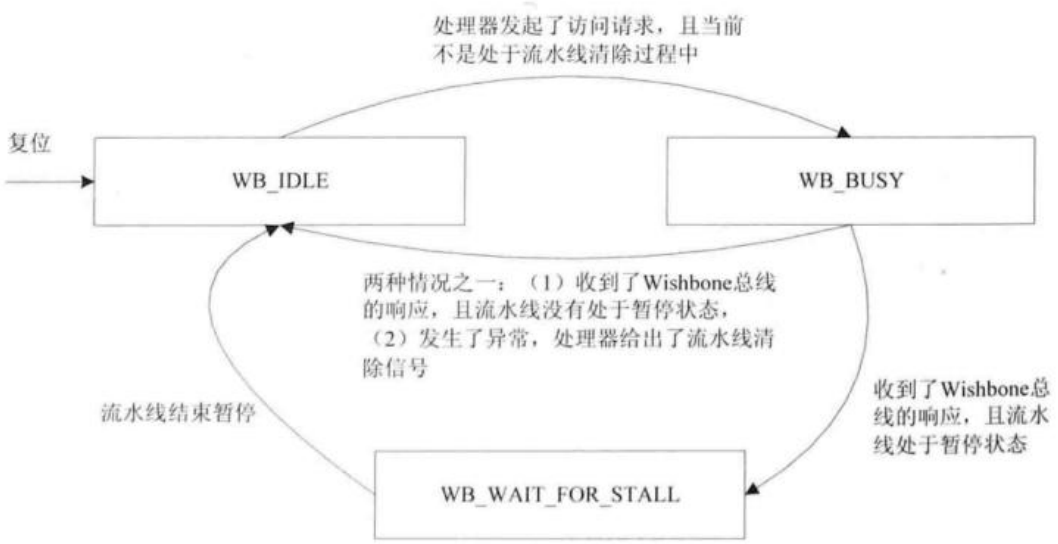
实验采用的 Wishbone 总线遵循了 Wishbone B2 版本规范，并采用了交叉互联的连接方式，以实现主从设备之间的有效通信。在此实验设置中，重点信号包括 CYC（周期信号）、STB（选通信号）以及 ACK（确认信号）。由于实验中的总线实现仅包括了 ACK 信号，而未包括 ERR（错误信号）和 RTY（重试信号），这可能会在调试过程中带来一些不便。

在集成 Wishbone 模块的过程中，还需要对 CPU 的 CTRL 控制模块和顶层模块进行相应的修改。这主要涉及按照文档或教材中描述的步骤，添加必要的控制信号和数据路径。具体的实施细节在此不作详细阐述。

4.1 WishBone 总线与 GPIO 模块

至于 Wishbone 总线的内部状态机，它是总线协议的核心，负责管理数据传输过程中的状态转换和信号控制，确保数据在主设备和从设备之间正确且高效地传输。通过精确地管理这些信号和状态，Wishbone 总线能够在各种硬件组件之间提供稳定且灵活的接口，促进模块间的无缝数据交换。

下图是总线内部的交换机图示：



在本实验中，Wishbone 总线的配置包括两个主设备和多个从设备。主设备 0 直接与 CPU 的数据接口和 DMEM（数据存储器）接口连接，而主设备 1 则连接到 CPU 的指令接口和 IMEM（指令存储器）接口。由于指令存储器（IMEM）从 Flash 存储中获取数据，因此指令存储器的起始地址需要与 Flash 连接的从设备的起始地址匹配。

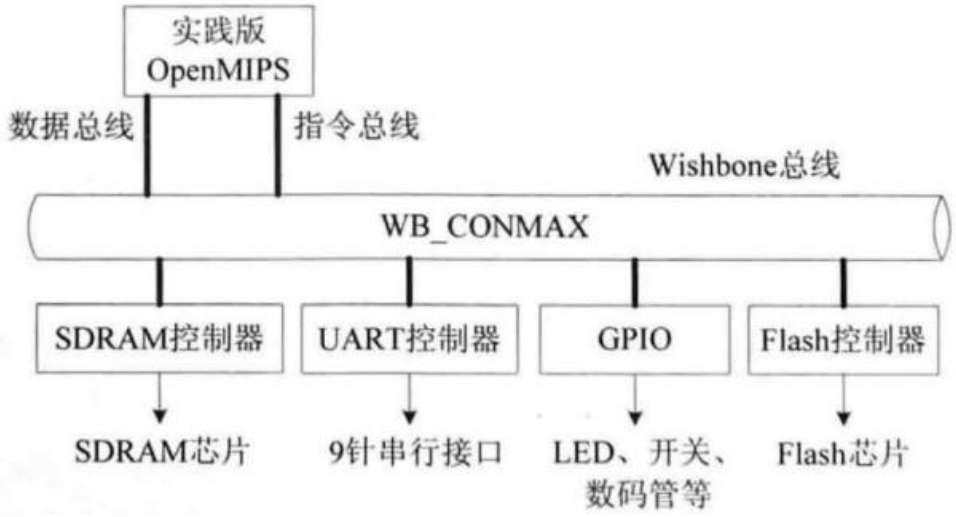
对于 Wishbone 总线的从设备配置如下：

- 从设备 0 连接到 SDRAM 及其 DDR2 存储控制器，主要用于存储数据。
- 从设备 1 负责与 UART 串口通信接口连接，用于实现串行通信。
- 从设备 2 连接到 GPIO 设备，用于通用输入输出功能。

- 从设备 3 连接到 Flash 存储控制器，此控制器同时负责指令的存储和管理。

这样的配置确保了各种存储和通信功能能够通过统一的总线接口高效地与 CPU 交互，从而提高系统的整体性能和灵活性。此外，确保各从设备地址分配合理，对于保证系统的稳定运行和优化性能至关重要。

总线和各个模块之间的关系图如下：



在本实验中，Wishbone 总线采用固定优先级的仲裁机制，可配置为 1、2 或 4 种优先级，支持多个主设备和从设备之间的并行通信。此总线的交叉互联架构基于 OpenCores 提供的开源 IP 核——WB_CONMAX，其详细逻辑和实现原理在此不再详述。

GPIO（通用输入/输出）接口是用于数字信号输入和输出的基础硬件接口，每个信号位可以独立控制。它允许处理器读取外部输入或向外部发送输出信号，从而与各种硬件设备进行交互，例如 LED 灯、开关、七段显示器等，甚至可以与采用位序列传输的设备如麦克风等通信。在本实验中，我们利用了 OpenCores 的开源 IP 核——GPIO IP Core，该核心与 Wishbone B2 总线协议兼容，主要通过 RGPIO_IN 和 RGPIO_OUT 寄存器实现，这使得 CPU 能够与外界的简单设备进行有效的数据交换。

UART（通用异步接收/发送器）是一种常用的串行通信协议，主要功能是将并行数据转换为串行数据进行发送，或将接收的串行数据转换回并行数据。通信的速率通过波特率来度量，波特率的计算不仅包括数据位，还涉及起始位、奇偶校验位和停止位等，因此波特率与实际的数据传输速率并不完全相同。在本实验中，设定的波特率为 9600 波特，实验所用的 UART 通信模块是基于 OpenCores 提供的开源项目 UART16550 IP Core。

4.2 SPI Flash 控制器

在本次实验中，设计的小型系统级芯片（SOPC）程序被存储在 Flash 存储器中，用于测试时存放测试程序，而在最终移植阶段，则存储 Bootloader 和操作系统程序。在这一阶段，CPU 首先从 Flash 中加载程序，通过 Bootloader 程序将操作系统复制到 DDR2 内存中，然后 CPU 在 DDR2 中执行操作系统。这一过程模仿了真实计算机系统中操作系统的加载过程。

实验使用的 Nexys4 DDR 开发板配备了型号为 S25FL128S 的 Quad-SPI FLASH 模块。主要通信接口包括 SDI（串行数据输入）、SDO（串行数据输出）以及 CS（片选信号），而 WP（写保护）和 HLD（暂停）信号虽然可用，但在本实验中基本未被利用。Flash 控制器与总线接口的设计细节如下：

```
module flash(

    input wire wb_clk_i,  // Wishbone 时钟输入

    input wire wb_rst_i,  // Wishbone 复位输入

    input wire wb_cyc_i,  // Wishbone 周期有效信号

    input wire wb_stb_i,  // Wishbone 选通信号

    input wire wb_we_i,   // Wishbone 写使能信号

    input wire [3:0] wb_sel_i,  // Wishbone 字节选择信号

    input wire [23:0] wb_adr_i,  // Wishbone 地址输入

    input wire [31:0] wb_dat_i,  // Wishbone 数据输入

    output reg [31:0] wb_dat_o,  // Wishbone 数据输出

    output reg wb_ack_o,  // Wishbone 应答输出

    output reg cs_n,      // SPI 片选信号

    input sdi,            // SPI 数据输入

    output reg sdo,       // SPI 数据输出

    output reg wp_n,      // SPI 写保护信号

    output reg hld_n      // SPI 暂停信号
```

在从板载 SPI Flash 读取数据的过程中，首先需要将片选信号 cs_n 设为低电平以启用通信，随后保证时钟信号 sck 维持在适当的工作频率范围内。本实验板的 SCK 信号不需要额外配置，直接使用板载的 50MHz 时钟频率。当 SDO 信号线串行接收到指令 0x03 后，Flash 开始接收接下来的 24 位地址信息。随后，Flash 芯片将通过 SDI 信号线分四次输出，每次输出 8 个字节的数据，因此状态机设计中至少包括等待状态、发送 READ 指令状态、发送地址状态、四个数据接收状态以及结束状态。

在使用 SD 卡控制器时，需要特别注意 SD 卡的版本和通信协议。市场上常见的 SD 卡通常支持 SDHC 或更高版本的协议，而 SD v1.0 协议的卡几乎已不在市场上流通。初次使用 SD 控制器时，可能会遇到 SD 卡持续返回 busy 信号的问题，这通常与协议处理不当有关。SDHC 卡在初始化阶段需要通过 CMD8 命令确认协议版本，接收到 CMD8 命令后，SD 卡会停止发送 busy 信号，开始正常通信。相比之下，旧版 SD 卡仅需通过 CMD41 和 CMD55 的命令组合即可建立通信。因此，使用针对旧协议设计的 SD 控制器 IP 核可能无法与现代 SD 卡进行有效通信。在本实验中，我们采用 GitHub 上的开源项目 SimpleSDHC 的 sd_spi.vhd 文件作为 SD 卡的底层通信模块，并使用 OpenCores 提供的开源 SD Cores 作为总线接口封装。

SD 卡主要用到的接口有 Clock（时钟信号）、Select（选择信号）、MOSI（输入信号），以及 MISO（输出信号），其模块设计接口如下：

```
module sd_controller(
    input wire wb_clk_i,          // Wishbone 时钟输入
    input wire wb_rst_i,          // Wishbone 复位输入
    input wire [31:0] wb_adr_i, // Wishbone 地址输入
    output wire [31:0] wb_dat_o, // Wishbone 数据输出
    input wire [31:0] wb_dat_i, // Wishbone 数据输入
    input wire [3:0] wb_sel_i,   // Wishbone 选择信号
    input wire wb_we_i,           // Wishbone 写使能信号
    input wire wb_stb_i,          // Wishbone 选通信号
    input wire wb_cyc_i,          // Wishbone 周期有效信号
```

```
    output reg wb_ack_o,          // Wishbone 应答信号输出
        input wire flash_clk_i,     // 专用于 Flash 的时钟输入
        input wire locked,          // 锁相环锁定信号
        input wire SD_CD,           // SD 卡检测信号
        output wire SD_RESET,       // SD 卡复位输出
        output wire SD_SCK,         // SD 卡时钟输出
        output wire SD_CMD,         // SD 卡命令信号线输出
        input wire SD_DAT0,         // SD 卡数据 0 输入
        inout [3:0] SD_DAT          // SD 卡数据线（双向）
    );
```

在实验过程中，最初使用的 SDHC 卡控制器负责指令存储的任务。尽管该控制器在硬件下板测试和总线连接方面表现良好，但在后续的实验步骤中，特别是在进行 COM 串口通信测试和操作系统移植时，出现了数据读取错误和系统异常挂起的问题。这些问题似乎是由于 DDR2 模块添加后产生的时序冲突所致。

为了解决这些问题，我们决定替换存储方案，使用 SPI Flash 控制器来实现指令存储功能。这一更改不仅解决了与 SDHC 卡相关的时序问题，也稳定了系统的运行，确保了实验的顺利进行。

4.3 DDR2 控制器添加

我们在实验中使用的板载 DDR2 SDRAM 型号为 MT47H64M16HR-25 E，这是一种双倍数据率的同步动态随机存储器（DDR SDRAM），提供比常规 RAM 更高的读取速率。此型号的 DDR2 内存配置为单一 Rank 和 16 位数据宽度，并且仅需要一个片选信号进行操作。由于 DDR2 的协议较为复杂，在使用时通常会借助 IP 核，如 Memory Interface Generator 来简化设计流程，并利用官方提供的 DDR-to-SRAM 模块来实现数据在 SRAM 和 DDR 之间的转换。

在具体使用时，DDR 一次最小的读写单元为 8 位，而最大读写宽度可达到 32 位。需要注意的是，不同版本的 IP 核之间可能存在不兼容的问题，且不同的 DDR IP 核文件也难以协调。因此，在使用 Vivado 设计工具时，建议明确指定使用的 IP 核版本，并尽可能一次性生成所需的 Memory Interface Generator，

避免在修改过程中出现问题导致项目无法顺利完成。同时需要注意的是过低版本的 Vivado 可能不能正确配置 DDR2 IP 核

关于 DDR2 的总线接口，我们参考与 NEXYS-4 DDR 开发板配套的官方项目代码，以确保正确实施和集成。

```
module DDR(
input wire wb_clk_i,
input wire wb_rst_i,
input wire wb_cyc_i,
input wire wb_stb_i,
input wire wb_we_i,
input wire [3:0] wb_sel_i,
input wire [26:0] wb_adr_i, input wire [31:0] wb_dat_i,
output reg [31:0] wb_dat_o, output reg wb_ack_o,
output reg init_calib_complete,
//memory signals
output [12:0] ddr2_addr,
output [2:0] ddr2_ba,
output ddr2_ras_n,
output ddr2_cas_n,
output ddr2_we_n,
output ddr2_ck_p,
output ddr2_ck_n,
output ddr2_cke,
output ddr2_cs_n,
output [1:0] ddr2_dm,
output ddr2_odt,
inout [15:0] ddr2_dq,
inout [1:0] ddr2_dqs_p, inout [1:0] ddr2_dqs_n
);
```

4.4 Ubuntu 环境搭建

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
```

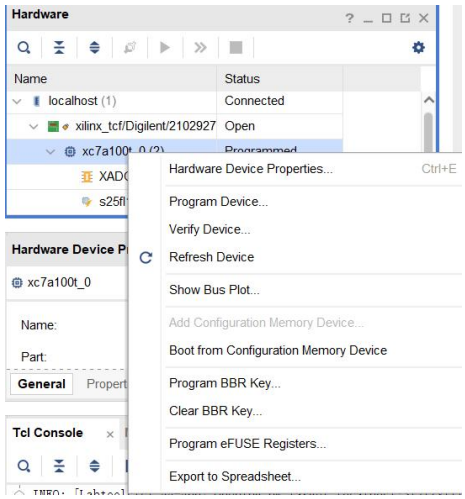
4.5 uc/OS 操作系统移植与编译

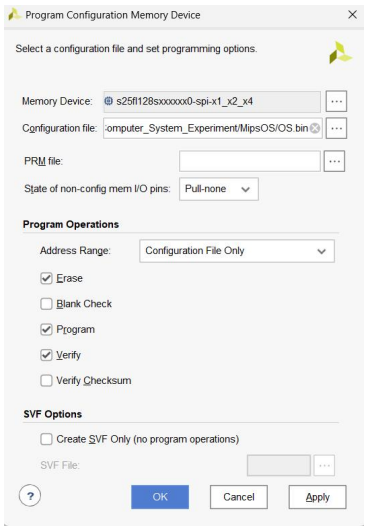
我们首先修改 Bootloader 中存储 UART 波特率的两个寄存器的值为 0x028B，并 对它重新编译，生成 BootLoader.bin：

之后修改 openmips.h 中的波特率和时钟频率，并对文件所有项目工程进行编译， 生成与 Bootloader.bin 结合好的可以移植的 OS.bin 文件：

```
-reduce -O2 -g -pipe -fno-builtin -nostdlib -mips32   -c -o os_cpu_a.o os_cpu_a.
S
mips-sde-elf-ld -r -o port.o os_cpu_c.o os_cpu_a.o
make[1]: Leaving directory '/home/ray/vivado/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
```

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



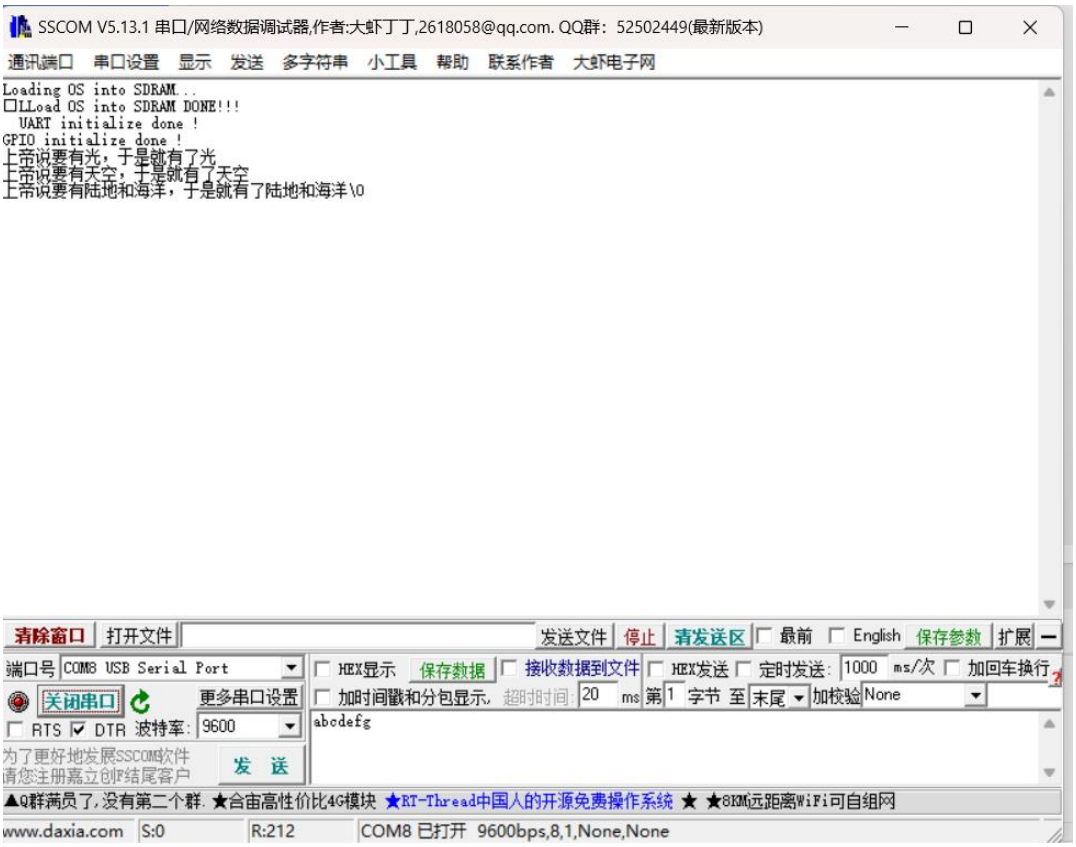


之后打开串口调试程序，调节波特率与下板系统一致（9600），选择对应的 USB 端口，下板后拨动开关即可看到串口调试程序中得到对应结果。

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：

图中 LED 灯亮起且变化，表示总线分配和响应也在不断变化。七段数码管显示 PC 值和传输数据值等，可见系统运行正常。将 μC/OS-II 操作系统下板后，打开串口 调试程序，将板子复位后，可以获得如下结果：



六、 总结与体会

在本次操作系统移植实验中，我独立完成了从系统设计到实施的全过程，这不仅验证了我的理论知识，还极大地提升了我的实践技能。实验中，我深入探索了微处理器的结构，并成功集成了多种硬件模块，包括 Wishbone 总线、GPIO、UART 接口、Flash 及 SDRAM 控制器等。

实验使我对如何处理硬件与软件的交互有了更深刻的理解，尤其是在交叉编译和系统移植过程中遇到的各种具体问题。通过对μC/OS-II 操作系统的移植，我从中获得了操作系统工作流程的实际经验，包括 Bootloader 的配置、内核的移植以及系统的调试等。这些知识和技能对我未来从事更高级的系统集成和开发工作具有重要的指导价值。

此外，通过独立解决实验中的问题，如 SD 卡的时序问题和 DDR2 的配置问题，我体会到了理论知识与实际应用之间存在的差异，以及自主学习和调试的重要性。每一个挑战都促使我更深入地研究相关技术和解决方案，极大地增强了我的问题解决能力。

总体来说，这次操作系统移植实验不仅让我掌握了必要的技术技能，还锻炼了我分析和解决复杂问题的能力。独立完成这样一个复杂的项目，让我对自己的专业知识和技能有了更加充分的信心，也为我的学习和职业生涯的未来奠定了坚实的基础。