同济大学

TONGJI UNIVERSITY

《计算机系统实验》

实验报告

实验名称	μC/OS-II 系统移植
实验成员	罗劲桐（1951443）
日　　期	2022 年　6 月　8 日

1、实验目的

- 回顾编译原理、操作系统等课程；
- 贯通计算机相关课程

2、实验内容

按照《自己动手写 CPU》思路，增加 Wishbone 总线，GPIO、UART、Flash 控制器、SDRAM 控制器。利用 Ubuntu 上建立交叉编译环境对 μC/OSII 系统进行改写、编译。最终成功将 μC/OSII 系统移植到 Digilent NEXYS4 DDR 开发板上并验证。

实验过程中，主要做出以下修改：

- 修改 Flash 控制器、SDRAM 控制器，以适配开发板上资源
- 修改 bootloader 模块，以适配 CPU 频率和 UART 频率
- 修改 μC/OSII 系统，适配 CPU 频率和存储器

3、实验步骤

3.1. 实现 89 条 MIPS CPU

在实验一中，参考《自己动手写 CPU 》，已经实现了这部分功能。

3.2. 总线模块修改

3.2.1. 增加 7 段数码管显示

通过 sw[3:0]选通 7 段数码管输出数据。将多个 32 位测试输出连接到 7 段数码管输入数据，并用 sw[3:0]作为 16 选 1 选通信号输出对应的数据。

```
1.    reg [31:0] show_data;
2.    seg7x16 SEG(
3.         .clk(CLK100MHZ),
4.         .reset(rst),
5.         .cs(1),
6.         .i_data(show_data),
7.         .o_seg(o_seg),
8.         .o_sel(o_sel)
9.     );
10.   always@(*)
11.    begin
12.            case(sw[3:0])
13.            4'd0:if(m1_addr_i!=32'b0)
14.                                show_data<=m1_addr_i;        //inst addr
15.            4'd1:if(m0_addr_i!=32'b0)
16.                                show_data<=m0_addr_i;        //data addr
17.            4'd2:if(m1_data_o!=32'b0)    //ack
18.                                show_data<=m1_data_o;        //inst
```

```
19.             4'd3:if(m0_data_o!=32'b0)     //ack
20.                                 show_data<=m0_data_o;         //data
21.             4'd4:if(gpio_o!=32'b0)
22.                                 show_data<=gpio_o;            //gpio out
23.             4'd5:if(s1_data_o!=32'b0)
24.                                 show_data<=s1_data_o;         //uart out
25.             default:show_data<=32'hfedc1234;
26.             endcase
27.     end
```

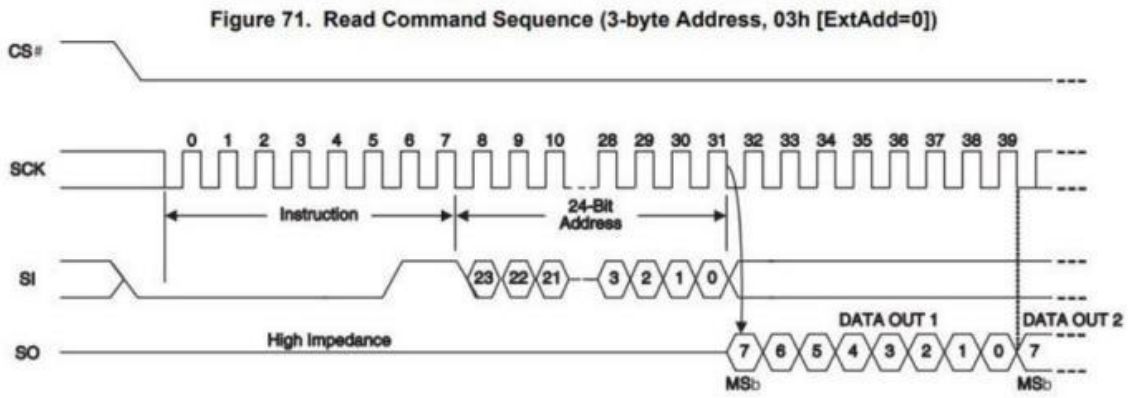
3.2.2. 增加 Flash 控制器

参考 CSDN 的博客，实现了 wishbone B2-spi flash 读写模块。在 wb_flash.v 模块中解析 wishbone 信号、写总线输出寄存器、写总线信号并控制 flash 执行数据读操作。在 flash.v 模块中，实现 flash 读写信号。

3.2.2.1. 接口信号

```
module flash(
    input           clk,            //100MHZ 时钟
    input           rst,            //异步复位信号
    input [31:0] rd_addr_i,         //读地址
    output reg   [31:0] rd_data_o,      //32 位数据输出
    output          finish,         //数据读取结束信号
    output reg   led, // used to debug

    // Ports for SPI Flash
    output reg   cs_n,              //片选信号低电平有效
    input           sdi,            //输入到主设备的信号线
    output reg   sdo,               //输入到 flash 的信号线
    output          wp_n,           //不使用
    output          hld_n           //不使用
);
```



3.2.2.2. flash.v 模块

首先向 spi flash 在 SI 数据线发送指令 8'h03，表示为读操作，然后从 MSB 到 LSB 发送 24 位地址。随后从 SO 数据线连续读出 32 位输出，保存在 rd_data_o 中。

```
READ_DATA:
begin     // get the first data from flash
if(~sdo_count[0]) begin
    datain_shift <= {datain_shift[6:0],sdi};
end

if(sdo_count == 4'd1) begin
    datain <= {datain_shift, sdi};
                        case (page_count)
                                16'h1:    rd_data_o[31:24] <= {datain_shift, sdi};
                                16'h2:    rd_data_o[23:16] <= {datain_shift, sdi};
                                16'h3:    rd_data_o[15:8] <= {datain_shift, sdi};
                                16'h4:    rd_data_o[7:0] <= {datain_shift, sdi};

                                default: begin end
                        endcase
end

            if(sdo_count != 4'd15) begin
                        sdo_count <= sdo_count + 4'd1;
            end
            else begin
    page_count <= page_count + 16'd1;
                        sdo_count  <= 4'd0;
                        next_state <= (page_count < (rd_cnt)) ? READ_DATA : ENDING;
            end
 end
```

SCK 引脚在下板过程中被占用，需要使用 STARTUPE2 模块重新连接 SCK。

```
STARTUPE2
#(
.PROG_USR("FALSE"),
.SIM_CCLK_FREQ(10.0)
)
STARTUPE2_inst
(
  .CFGCLK     (),
  .CFGMCLK    (),
  .EOS        (),
  .PREQ       (),
  .CLK        (1'b0),
  .GSR        (1'b0),
  .GTS        (1'b0),
  .KEYCLEARB  (1'b0),
  .PACK       (1'b0),
  .USRCCLKO   (sck),     // First three cycles after config ignored, see AR# 52626
```

```
18.   .USRCCLKTS  (1'b0),     // 0 to enable CCLK output
19.   .USRDONEO   (1'b1),     // Shouldn't matter if tristate is high, but generates a
    warning if tied low.
20.   .USRDONETS  (1'b1)      // 1 to tristate DONE output
21. );
```

3.2.2.3. wb_flash.v 模块

接受总线信号 cyc&stb=1 时，开始 wishbone 读写历程。如果 we=0，则为从 flash 中读取 32 位数据保存到 wb_dat_o。总线输出数据准备好后，ack=1。随后在 stb=0 后，ack 置为 0，结束一次总线访问。

```
1.      always @(posedge wb_clk_i) begin
2.          if( wb_rst_i == 1'b1 ) begin
3.              waitstate <= 4'h0;
4.              wb_ack_o <= 1'b0;
5.              flash_rst<=1'b1;
6.              is_end<=4'b0;
7.          end else if(wb_acc == 1'b0) begin
8.              waitstate <= 4'h0;
9.              wb_ack_o <= 1'b0;
10.             wb_dat_o <= 32'h00000000;
11.             flash_rst<=1'b1;
12.             is_end<=4'b0;
13.         end else if(waitstate == 4'h0) begin
14.             wb_ack_o <= 1'b0;
15.             if(wb_acc) begin
16.               waitstate <= waitstate + 4'h1;
17.             end
18.                       flash_addr_i <= {10'b0000000000,wb_adr_i[21:2],2'b0};
19.             flash_rst<=1'b0;
20.             is_end<=4'b0;
21.         end else begin
22.             if(flash_finish) begin
23.                 wb_dat_o<=flash_data_o;
24.                 flash_rst<=1'b1;
25.                 wb_ack_o <= 1'b1;
26.                 is_end<=4'b1;
27.             end
28.             else begin
29.                 waitstate <= waitstate + 4'h1;
30.             end
31.
32.             if(is_end==4'b1&&wb_stb_i==1'b0) begin
33.                 wb_ack_o <= 1'b0;
34.             end
35.          end
36.       end
```

3.2.3. 增加 SDRAM 控制器

使用板载 ddr2 内存和 mig 控制接口实现 ddr2 读写历程。

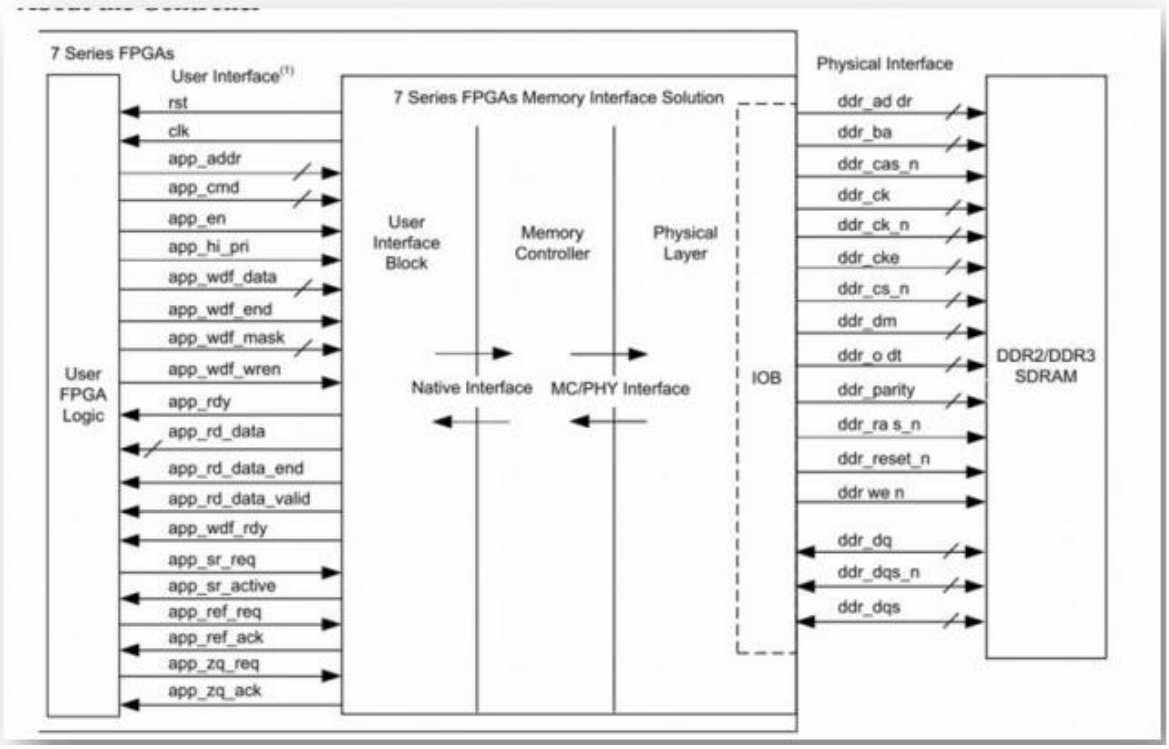
3.2.3.1. 接口配置

module ddr2_signal_controller(

input clk, //100MHZ 时钟

```
input rst,                  //异步复位信号
input clk_ref_i,            //200MHZ 参考时钟
output ddr2_ck_p,           //与 ddr2 相连
output ddr2_ck_n,           //与 ddr2 相连
output ddr2_cke,            //与 ddr2 相连
output ddr2_cs_n,           //与 ddr2 相连
output ddr2_ras_n,          //与 ddr2 相连
output ddr2_cas_n,          //与 ddr2 相连
output ddr2_we_n,           //与 ddr2 相连
output [1:0] ddr2_dm,       //与 ddr2 相连
output [2:0] ddr2_ba,       //与 ddr2 相连
output [12:0] ddr2_addr,    //读写地址，与 ddr2 相连
inout [15:0] ddr2_dq,       //与 ddr2 相连
inout [1:0] ddr2_dqs_p,     //与 ddr2 相连
inout [1:0] ddr2_dqs_n,     //与 ddr2 相连
output [1:0] rdqs_n,        //useless
output ddr2_odt,            //与 ddr2 相连

input [31:0] addr_in,       //输入 32 位地址
input [127:0] data_in,      //写数据，通过 app_wdf_mask 处理后写入 SDRAM
input stb_in,               //使能总线 stb
output reg ack_out,         //总线回复 ack
output [127:0] app_rd_data, //128 位读取的数据
output app_rd_data_valid,   //数据读写结束信号
output app_rdy              //初始化结束信号
);
```



3.2.3.2. 发送 mig 读写指令

在写数据的时候必须检测 app_rdy 和 app_wdf_rdy 信号是否同时有效，否则写入命令无法成功写入到 DDR 控制器的命令 FIFO。读操作时检测 app_rdy 有效，表示读到的数据。

```
1.      always @(posedge ui_clk or posedge rst)
2.      begin
3.          if(rst)
4.          begin
5.              app_en_pe <= 0;
6.              app_addr_pe<=0;
7.
8.                          app_cmd_wr <= 3'b1;
9.              app_en_wr <= 1'b0;
10.             app_wdf_data <= 128'h0;
11.             app_addr_wr <= 27'h0;
12.             app_wdf_end <= 1'b0;
13.             app_wdf_wren <= 1'b0;
14.             ack_out <= 0;
15.             state <= IDLE;
16.         end else if(ui_clk_sync_rst)
17.         begin
18.             app_en_pe <= 0;
19.             app_addr_pe<=0;
20.
21.                         app_cmd_wr <= 3'b1;
22.             app_en_wr <= 1'b0;
23.             app_wdf_data <= 128'h0;
24.             app_addr_wr <= 27'h0;
25.             app_wdf_end <= 1'b0;
26.             app_wdf_wren <= 1'b0;
27.             ack_out <= 0;
```

```
28.             state <= IDLE;
29.         end
30.         else if(~stb_in)
31.              begin
32.                       app_en_wr <= 0;
33.             app_wdf_wren <= 0;
34.             app_wdf_end <= 0;
35.             ack_out <= 0;
36.             case(state)
37.                                 IDLE:
38.                                 begin
39.                                           app_en_pe <= 1;
40.                                           app_addr_pe<={addr_in,3'b0};
41.                                           app_cmd_pe <= 3'b001;
42.                                           state <= READ;
43.                                 end
44.                                 READ:
45.                                 begin
46.                                           if(app_rdy) begin
47.                                                     app_en_pe <= 1'b0;
48.                                                     app_addr_pe<={addr_in,3'b0};
49.                                                     state <= IDLE;
50.                                           end
51.                                 end
52.                                 default:state <= IDLE;
53.             endcase
54.         end
55.         else
56.              begin
57.             app_en_pe <= 1'b0;
58.                       case(state)
59.                 IDLE:
60.                 begin
61.                     if(app_rdy & app_wdf_rdy)
62.                                           begin
63.                         app_wdf_data <= data_in;
64.                         app_cmd_wr <= 3'b0;
65.                         app_addr_wr <= {addr_in,3'b0};
66.                         app_wdf_wren <= 1'b1;
67.                         app_wdf_end <= 1'b1;
68.                         app_en_wr <= 1'b1;
69.                         ack_out <= 0;
70.                         state <= WRITE;
71.                     end
72.                     else state <= IDLE;
73.                 end
74.                 WRITE:
75.                                 begin
76.                     app_en_wr <= 1'b0;
77.                     app_cmd_wr <= 3'b1;
78.                     ack_out <= 1;
79.                     app_wdf_wren <= 1'b0;
80.                     app_wdf_end <= 0;
81.                     state <= IDLE;
82.                 end
```

```
83.                 default:state <= IDLE;
84.             endcase
85.         end
86.     end
```

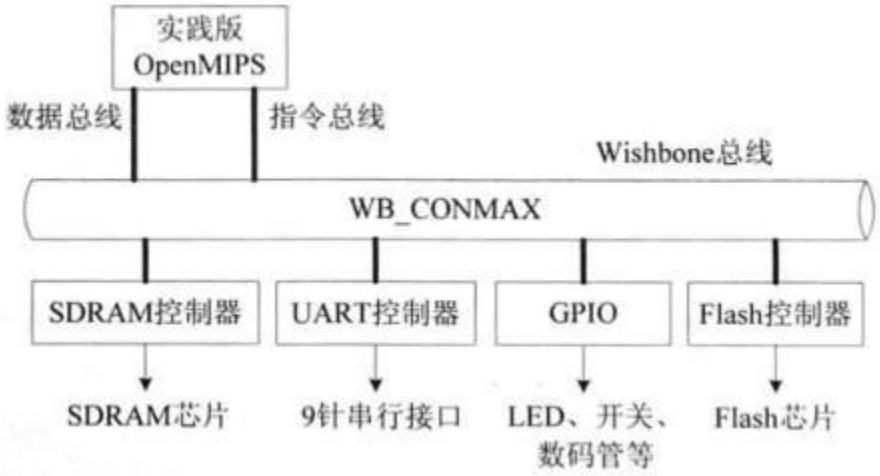
3.2.3.3. wishbone 总线交互

接受总线信号 cyc&stb=1 时，开始 wishbone 读写历程。根据 we 判断读写请求，从 ddr2 中读取 32 位数据保存到 wb_dat_o，或通过 buffer，改写其中 32 位数据并写入 ddr2。总线输出数据准备好后，ack=1。随后在 stb=0 后，ack 置为 0，结束一次总线访问。

```
1.    always@(posedge CLK100MHZ or posedge rst)
2.    begin
3.        if(rst)
4.        begin
5.            s_ack<=1'b0;
6.            s_err<=1'b0;
7.            s_rty<=1'b0;
8.            ddr_we<=1'b0;
9.
10.           start_insist_count<=16'b0;
11.           read_insist_count<=16'b0;
12.           write_insist_count<=16'b0;
13.
14.           state<=INIT;
15.           return_state<=INIT;
16.       end
17.       else
18.       begin
19.           case(state)
20.           INIT:begin
21.               s_ack<=1'b0;
22.               ddr_we<=1'b0;
23.               state<=IDLE;
24.           end
25.           IDLE:begin
26.               s_ack<=1'b0;
27.               ddr_we<=1'b0;
28.               start_insist_count<=16'b0;
29.               read_insist_count<=16'b0;
30.               write_insist_count<=16'b0;
31.               if(s_acc) begin
32.                       if(s_rd) begin
33.                           state<=READ;
34.                       end else if (s_wr) begin
35.                           state<=WRITE;
36.                       end
37.                       else begin
38.                           state<=IDLE;
39.                       end
40.               end else begin
41.                   state<=IDLE;
42.               end
43.           end
```

3.2.4. 增加 Wishbone 总线

参考《自己动手写 CPU 》，使用 wishbone B2 总线接口 IP 核，使用的是 OpenCores 站点提供的开源项目 WB_CONMAX ，这是一个 Wishbone 总线互联矩阵，采用的是交叉互联方式，允许多对主从设备同时进行通信。。Wishbone 总线使用有交叉互联方式连接所有设备，其中主设备为 CPU 地址接口和数据接口，从设备为 GPIO、UART、FLASH、DDR2。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。




```
1.    wb_conmax_top wb_conmax_top0(
2.              .clk_i(CLK100MHZ),
3.              .rst_i(rst),
4.
5.        // Master 0 Interface
6.        .m0_data_i(m0_data_i),
7.        .m0_data_o(m0_data_o),
8.        .m0_addr_i(m0_addr_i),
9.        .m0_sel_i(m0_sel_i),
10.       .m0_we_i(m0_we_i),
11.       .m0_cyc_i(m0_cyc_i),
12.       .m0_stb_i(m0_stb_i),
13.       .m0_ack_o(m0_ack_o),
14.  ......
15.       // Slave 0 Interface
16.       .s0_data_i(s0_data_i),
17.       .s0_data_o(s0_data_o),
18.       .s0_addr_o(s0_addr_o),
19.       .s0_sel_o(s0_sel_o),
20.       .s0_we_o(s0_we_o),
21.       .s0_cyc_o(s0_cyc_o),
22.       .s0_stb_o(s0_stb_o),
23.       .s0_ack_i(s0_ack_i),
24.       .s0_err_i(1'b0),
25.       .s0_rty_i(1'b0),
26.  ......
```

3.2.5. 增加 GPIO

此次实验中，采用 OpenCores 站点提 供的开源项目 GPIO IP Core ，添加整个系统中。

然后在 sw[3:0]选通信号中输入 GPIO 输出，以在 7 段数码管显示 GPIO 输出。

```
1.    gpio_top gpio_top0(
2.      .wb_clk_i(CLK100MHZ),
```

```
3.              .wb_rst_i(rst),
4.              .wb_cyc_i(s2_cyc_o),
5.              .wb_adr_i(s2_addr_o[7:0]),
6.              .wb_dat_i(s2_data_o),
7.              .wb_sel_i(s2_sel_o),
8.              .wb_we_i(s2_we_o),
9.              .wb_stb_i(s2_stb_o),
10.     .wb_dat_o(s2_data_i),
11.             .wb_ack_o(s2_ack_i),
12.             .wb_err_o(),
13.             .wb_inta_o(gpio_int),
14.             .ext_pad_i(gpio_i_temp),
15.             .ext_pad_o(gpio_o),
16.             .ext_padoe_o()
17.   );
```

3.2.6. 增加 UART

在此次实验中，采用的 OpenCores 站点提供的开源项目 UART16550 IP Core。添加了 UART 控制器之后，就可以通过串口与 计算机进行通信了。

```
1.    uart_top  uart_top0(
2.       .wb_clk_i(CLK100MHZ),
3.       .wb_rst_i(rst),
4.       .wb_adr_i(s1_addr_o[4:0]),
5.       .wb_dat_i(s1_data_o),
6.       .wb_dat_o(s1_data_i),
7.       .wb_we_i(s1_we_o),
8.       .wb_stb_i(s1_stb_o),
9.       .wb_cyc_i(s1_cyc_o),
10.      .wb_ack_o(s1_ack_i),
11.      .wb_sel_i(s1_sel_o),
12.      .int_o(uart_int),
13.      .stx_pad_o(uart_out),
14.      .srx_pad_i(uart_in),
15.      .cts_pad_i(1'b0),
16.      .dsr_pad_i(1'b0),
17.      .ri_pad_i(1'b0),
18.      .dcd_pad_i(1'b0),
19.      .rts_pad_o(),
20.      .dtr_pad_o()
21.   );
```

3.3. μC/OSII 系统修改

3.3.1. 修改 bootloader 模块

修改了 UART 分频频率，按照实验参考上修改后，实际分频后的波特率为 19200，需要同步修改串口测试器的接收波特率，才能接收到正确信息。

```
1.  _start:
2.
3.     lui $1,0x1000
4.     ori $1,$1,0x0003
5.     ori $2,$0,0x80
6.     sb  $2,0x0($1)
7.
8.     lui $1,0x1000
```

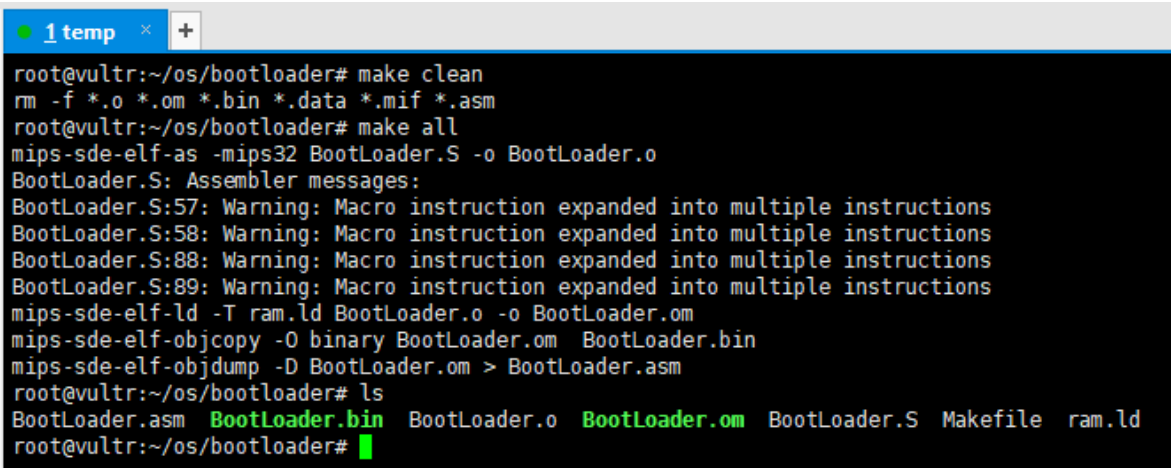
```
9.     ori $1,$1,0x0001
10.    ori $2,$0,0x01
11.    sb  $2,0x0($1)    # MSB of divisor latch
12.
13.    lui $1,0x1000
14.    ori $1,$1,0x0000
15.    ori $2,$0,0x45
16.    sb  $2,0x0($1)    # LSB of divisor latch
17.
18.    lui $1,0x1000
19.    ori $1,$1,0x0003
20.    ori $2,$0,0x03
21.    sb  $2,0x0($1)    # 8bit, no parity, 1 stop bit
```

3.3.2. 修改 openmips.h

修改时钟频率和UART波特率，这里与实验指导做法不同，由于实际接受的波特率为19200，需要对时钟频率和 UART 波特率，做出同步修改。

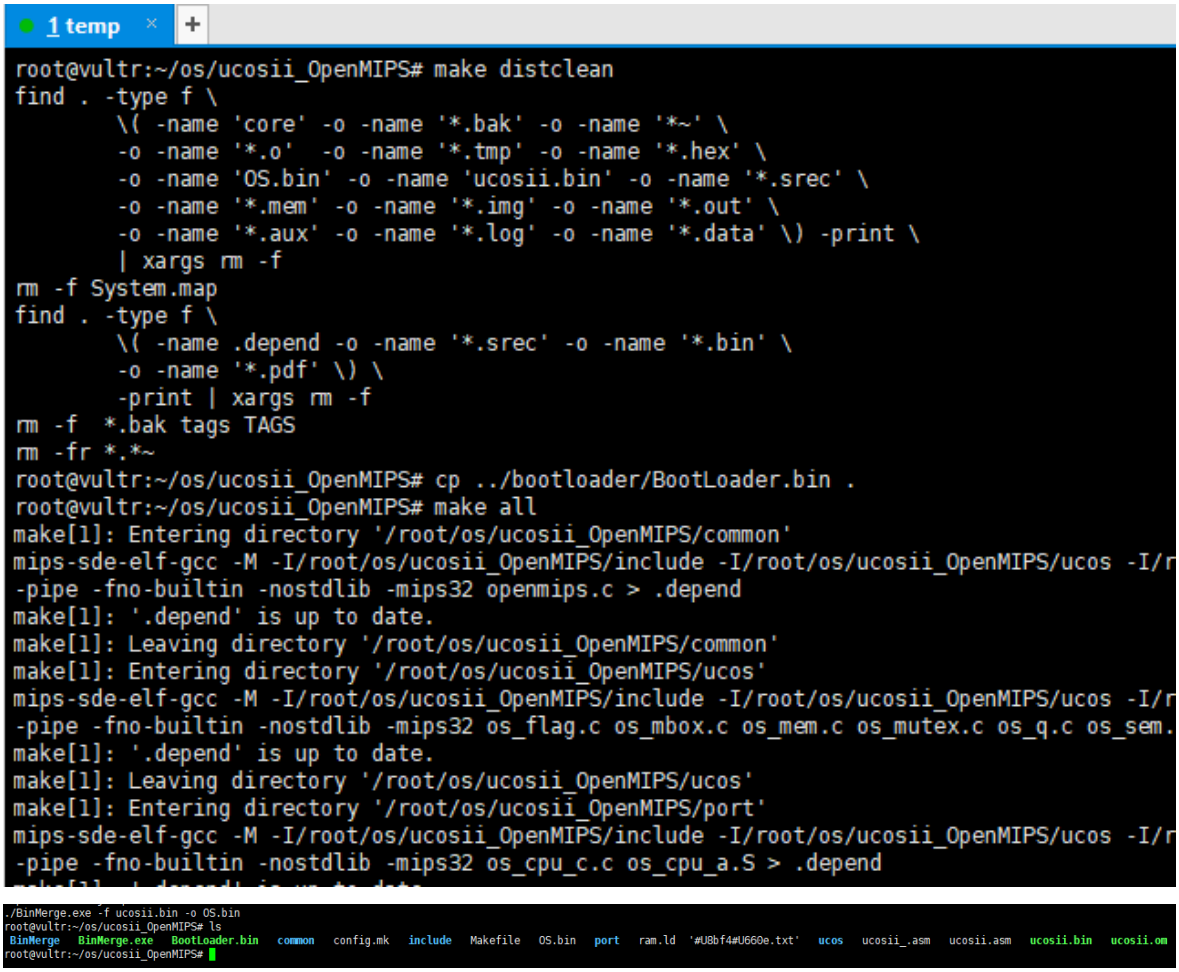
```
1.  #define IN_CLK 100000000            /* ʿɬʰ27ˇ┬MHz */
2.
3.  /*****************************************************************
4.
5.                          ´®¿ز²ĭ
6.
7.  *****************************************************************/
8.
9.  #define UART_BAUD_RATE  19200         /* ´®¿ڂز´Ź600bps */
10. #define UART_BASE      0x10000000
11. #define UART_LC_REG    0x00000003    /* Line Control Register */
12. #define UART_IE_REG     0x00000001    /* Interrupt Enable Register */
13. #define UART_TH_REG    0x00000000    /* Transmitter Holding Register */
14. #define UART_LS_REG     0x00000005    /* Line Status Register */
15. #define UART_DLB1_REG   0x00000000    /* Divisor Latch Byte 1(LSB) */
16. #define UART_DLB2_REG   0x00000001    /* Divisor Latch Byte 2(MSB) */
```

3.3.3. 编译 bootloader



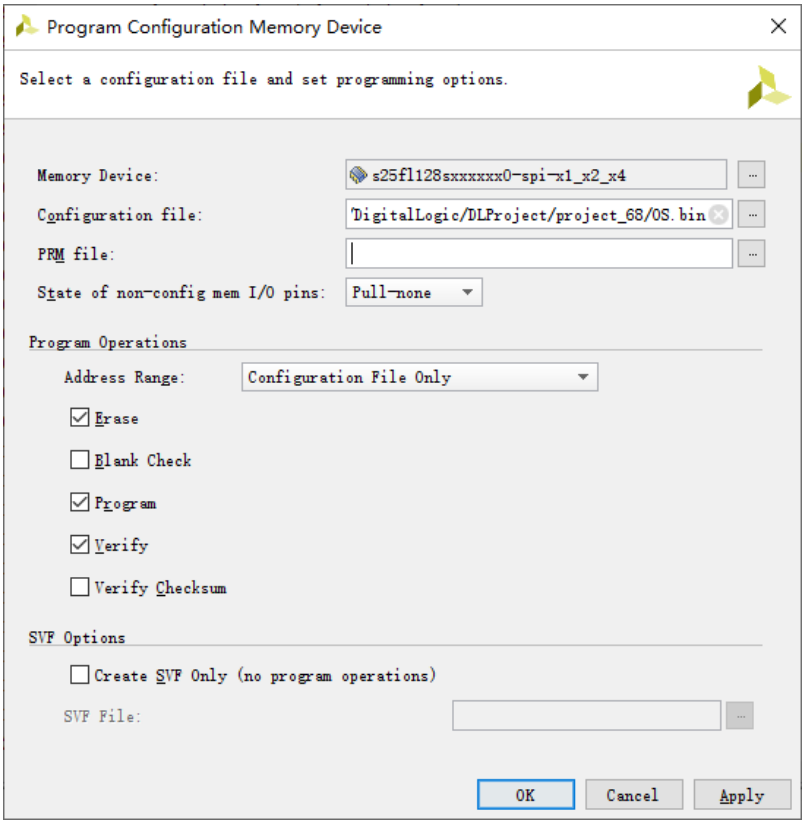
3.3.4. 编译 μC/OSII 系统

由于 makefile 文件构建.depend 文件，需要先进行 distclean，清除依赖。



4、实验结果

4.1. 配置 spi flash 数据，写入 μC/OS II 操作系统的二进制程序文件。



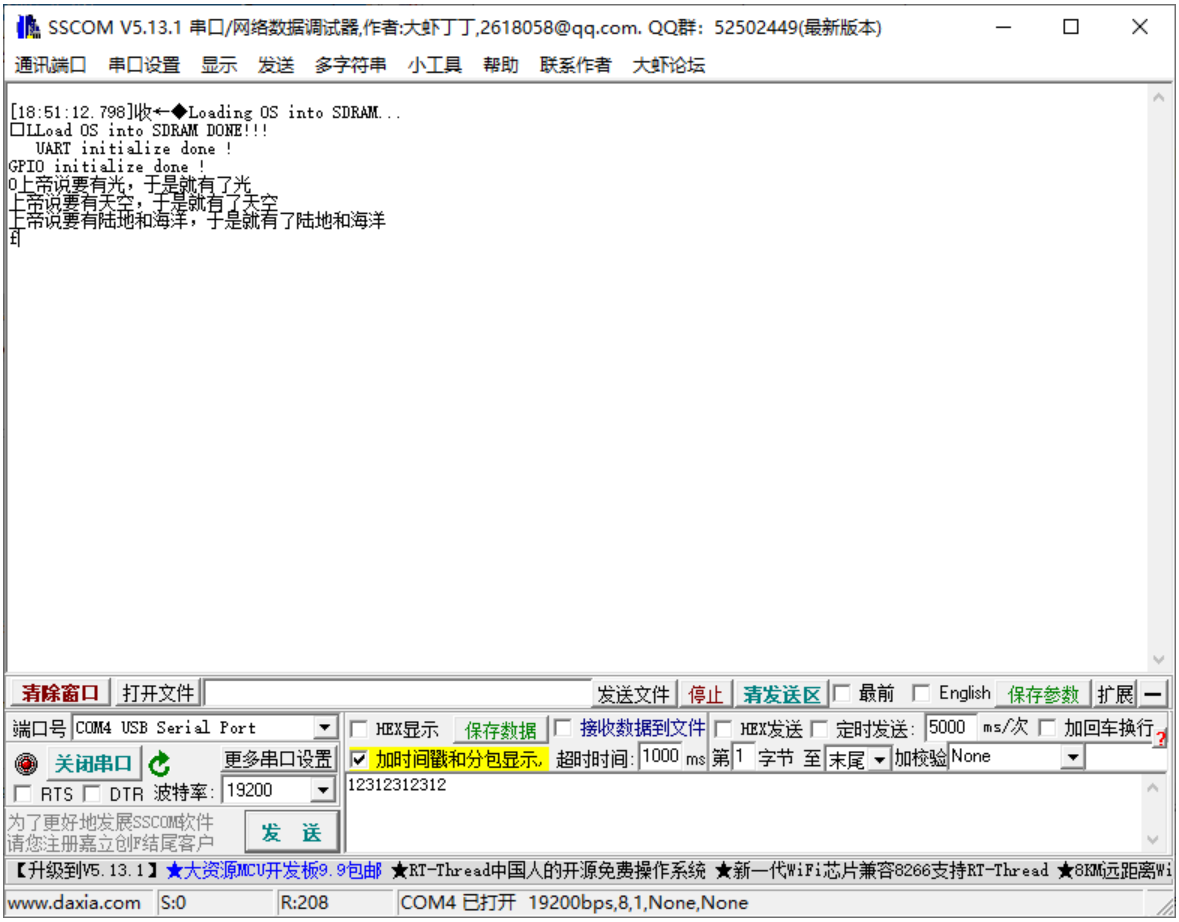
装

订

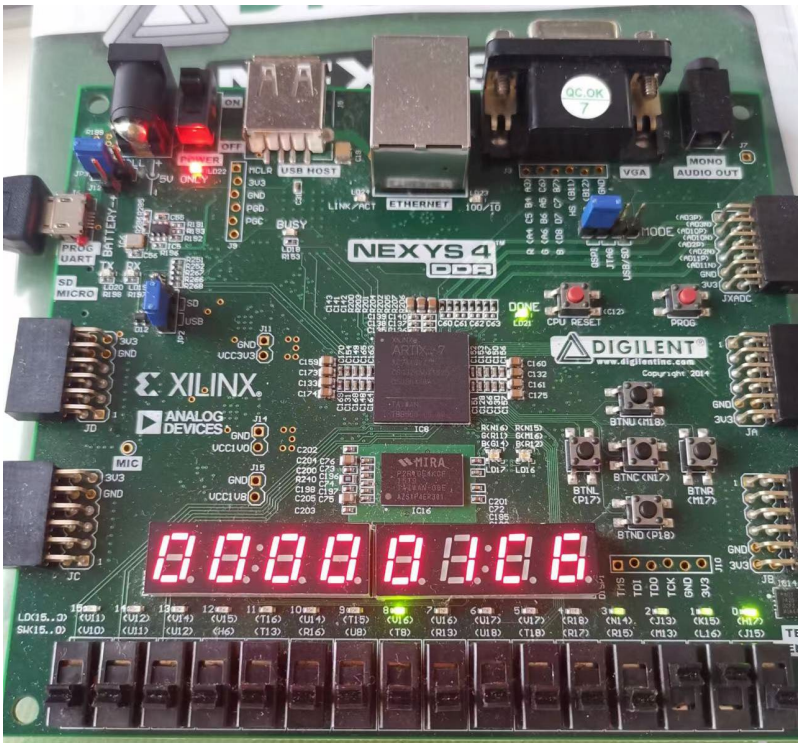
线

4.2. 串口结果

打开串口调试，设置波特率 19200bps，8 位数据位、没有奇偶校验位，1 位停止位。将 bit 流下板，观察串口通信



4.3. 数码管 GPIO 输出，sw[3:0]=6，查看 gpio 输出。



5、实验总结

在本次μC/OS-II 系统移植实验中，设计并实现了 gpio、uart、spi flash、ddr2 总线接口模块，并使用 wishbone B2 连接这些模块和 89 条 MIPS 动态流水线 CPU，能够输出 gpio 到 7 段数码管，uart 输入输出，从 flash 中读取 bootloader 并在 ddr2 中读取和写入。

设计系统移植的 cpu 过程中，需要考虑到不同模块对时序的不同要求，因此，不能直接使用总线数据访问各个模块，容易造成时序错误，导致前仿真结果与下板表现不一致的情况。对于这种情况，在统一总线时钟的情况下为不同模块设计总线接口和功能接口，减少时序不吻合的情况。通过这部分的学习，方便了程序的阅读和快速修改，对于代码的可读性和健壮性有很大帮助，我从中也学到了很多。

另外，89 条 MIPS 动态流水线 CPU 涉及非对齐地址的内存访问汇编代码。在这次 CPU 改造实验中，在 ddr2 写入过程中，通过 sel 指定写入对应的字节，对于 32 位输入中部分字节执行写入操作。