浙江大学

本科实验报告

课程名称：	计算机组成与设计
姓　　名：	熊子宇
学　　院：	竺可桢学院
专　　业：	混合班
指导教师：	姜晓红/叶大源
报告日期：	2022 年 3 月 9 日

实验一 搭建 CPU 测试框架

一、实验目的

1. 了解计算机系统的基本结构和各组件的基本功能
2. 导入已有组件，完成顶层模块中 MACtrl 部分的数据通路连线
3. 设计 MemoryAccessController，并仿真验证其功能
4. 生成 CPU 测试框架的 bit 流，上板验证

二、实验方法与步骤

1 了解计算机系统的基本结构和各组件的基本功能

冯诺依曼的计算机模型由 CPU、Memory 和 I/O 设备构成。

CPU Core: 负责指令执行和运算。其内部主要有 PC 和 IR 两个寄存器，PC 存下一条指令的地址，IR 存当前指令的内容。

Memory：冯氏体系的计算机将数据和指令都存放在内存中。

Memory 有两个重要的寄存器 MAR 和 MDR。

- 在 Load 状态下，MAR 存放所需读入的内存的地址，MDR 存放 MAR 对应地址内的内容。
- 在 Store 状态下，MAR 存放需要写入的内存地址，MDR 存放需要写入的内容。

在本实验中 Memory 分为 IMem（Instruction Memory,存放指令）和 DMem（Data Memory，存放数据）两部分。对于内存的读写，则需要 MemoryAccessController 来完成调度（这也是本实验的主要内容）。

I/O 设备：在 Sword 平台上，输入设备主要是开关和阵列键盘，输出设备主要是 LED 和 VGA。

Input：

- Switch：一共 16 个机械开关
- InputAntiJitter: 阵列键盘的输入和防抖动模块(集成在一起)

Output：

- VGA: 板上显示屏的驱动模块
- LEDCtrl：驱动板上 LED，在本实验中使用地址 xFE000000 映射到 LEDCtrl

MemoryAccessController：连接 CPU、Memory 和 Output 设备的通路。控制数据来源与去向。接收 CPU 发来的内存访问地址和写使能信号，从相应组件获取数据后返回给 CPU。

- 指令读取：获取 instruction address，传给 IMem，从 IMem 获取 instruction data 后返回给 CPU。
- 数据读写：获取 data address，如果地址不是 xFE000000 则表示为内存的真实地址，传给 DMem。
  - 若 WE=0，表示读取数据，从 DMem 获取 data 后返回给 CPU。

- 若 WE=1，表示写入数据，把从 CPU 获得的数据写入 DMem 对应地址中。

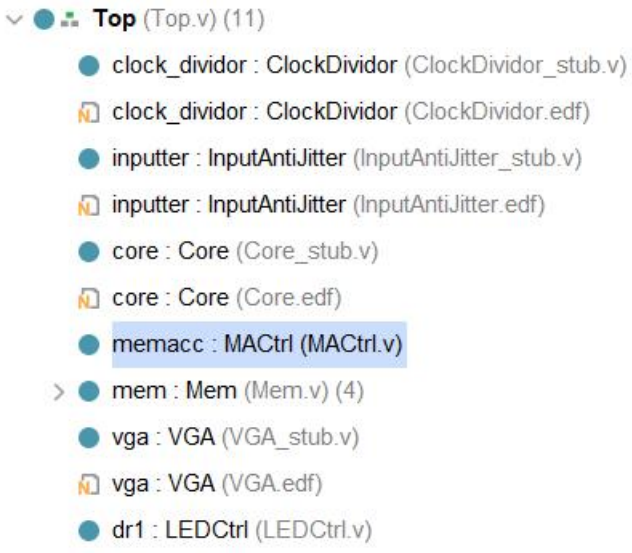
- I/O 设备读写：获取 data address，如果地址为 xFE000000 则表示为映射的输出设备，传给 LEDCtrl 组件（输出设备的一个例子）。

  - 若 WE=0，表示读取数据，从 LEDCtrl 获取 data 后返回给 CPU。
  - 若 WE=1，表示写入数据，把从 CPU 获得的数据写入 LEDCtrl 对应地址中。

2 导入已有组件，完成顶层模块中 MACtrl 部分的数据通路连线

2.1 建立 SCPU 工程，导入已有组件

点击主菜单上的 New Project -> RTL Project -> Add Files，把提供的文件全部导入，选择板子型号，进入主界面。文件层级结构如下图（除了 MACtrl.v 为后续新建）



2.2 完成顶层模块中 MACtrl 部分的数据通路连线

MACtrl 和 Mem 部分代码如下：

```
wire [31:0] o_imem_addr;
wire [31:0] i_imem_o_data;
wire [31:0] o_dmem_addr;
wire [31:0] i_dmem_o_data;
wire [31:0] o_dmem_i_data;
wire o_dmem_wen;

MACtrl memacc(
    //facing core
    .i_iaddr(imem_addr),
    .o_idata(imem_o_data),
    .i_dwen(dmem_wen),
    .i_daddr(dmem_addr),
    .i_d_idata(dmem_i_data),
    .o_d_odata(dmem_o_data),
    //facing IMem
    .o_iaddr(o_imem_addr),
    .i_idata(i_imem_o_data),
    //facing DMem
    .o_dwen(o_dmem_wen),
    .o_daddr(o_dmem_addr),
```

```
        .o_d_idata(o_dmem_i_data),
        .i_d_odata(i_dmem_o_data),
        //facing DR1
        .o_dr1wen(dr1_wen),
        .o_dr1_idata(dr1_i_data),
        .i_dr1_odata(dr1_o_data)
    );
    Mem mem(
        .i_addr(o_imem_addr),
        .i_data(i_imem_o_data),
        .clk(~clk_100mhz),
        .d_wen(o_dmem_wen),
        .d_addr(o_dmem_addr),
        .d_i_data(o_dmem_i_data),
        .d_o_data(i_dmem_o_data)
    );
```

在助教给出的 Top.v 中，Core 和 Mem 是直接相连的，而现在要设计的 MACtrl 要起到中转调度的作用，因此 Top.v 中相连的部分要断开，即新建一组变量，替换到 Mem 的参数表中。

根据第 1 节对 MemoryAccessController 的需求分析，主要有三部分功能的连线：

- 指令读取：iaddr 和 idata 的同名端口应该与 IMem 访问有关。i_iaddr 和 o_iaddr 应当分别为接收 CPU 传来的 imem_addr 并传递给 IMem。IMem 获取指令内容后，再通过 imem_o_data 返回给 CPU。

- 数据读写：dwen, daddr, d_idata 和 d_odata 的同名端口分别为数据写使能信号，数据地址，写入数据和传出数据，与 DMem 访问有关，因此应该连接 dmem_系列的线。dwen, daddr, d_idata 从 CPU 传给 MACtrl，然后传递给 DMem。DMem 读出数据后，通过 i_dmem_o_data 返回给 CPU。

- I/O 设备读写：与读写 DMem 比较类似，除了映射的地址固定为 xFE000000，所以不需要 addr。连接 drl_系列的线即可。

3 设计 MemoryAccessController，并仿真验证其功能

3.1 设计 MACtrl.v

MACtrl.v 代码如下：

```
module MACtrl(
        //facing core
        input [31:0]i_iaddr,
        output [31:0]o_idata,
        input i_dwen,
        input [31:0]i_daddr,
        input [31:0]i_d_idata,
        output [31:0]o_d_odata,
        //facing IMem
        output [31:0]o_iaddr,
        input [31:0]i_idata,
        //facing DMem
        output o_dwen,
        output [31:0]o_daddr,
        output [31:0]o_d_idata,
        input [31:0]i_d_odata,
        //facing DR1
        output o_dr1wen,
        output [31:0]o_dr1_idata,
```

```
        input [31:0]i_dr1_odata
    );

    assign o_iaddr = i_iaddr;
    assign o_idata = i_idata;
    assign o_dr1wen = (i_daddr != 32'hFE000000) ? 0 : i_dwen;
    assign o_dwen = (i_daddr != 32'hFE000000) ? i_dwen : 0;
    assign o_daddr = (i_daddr != 32'hFE000000) ? i_daddr : o_daddr;
    assign o_d_idata = (i_daddr != 32'hFE000000) ? i_d_idata : o_d_idata;
    assign o_d_odata = (i_daddr != 32'hFE000000) ? i_d_odata : i_dr1_odata;
    assign  o_dr1_idata  =  (i_daddr  !=  32'hFE000000)  ?  o_dr1_idata  :
i_d_idata;
endmodule
```

代码思路如下：

首先是写形参表。把 top.v 中的接口拷贝过来，以 i 开头的变量设为 input，以 o 开头的变量设为 output。地址线和数据线都是 32 位的，而写使能信号只有 1 位。

然后写 assign 语句。基本思路是给所有 output 都找到对应的赋值。

- 读入指令是单独的部分，直接用 assign 连接同名变量即可。

- 数据读写和向 LED 读写的区别仅在于 CPU 给的 daddr 不同。因此根据 data address 是否为 xFE000000，使用条件表达式对 d 系列的变量赋值。冒号左边都是访问 DMem，冒号右边都是访问 LED。

  访问 DMem：

  DR 设备的写使能 o_dr1wen= 0。

  DMem 的写使能 o_dwen = CPU 送来的控制信号 i_dwen。

  DMem 接收到的地址 o_daddr = CPU 送来的地址 i_daddr。

  DMem 接收到的写的内容 = CPU 送来的写的内容 i_d_idata。

  CPU 输出的内容 o_d_odata = DMem 从该地址访问到的内容 i_d_odata。

  DR 设备接收到的写的内容保持不变。

- 访问 DR 设备的情形非常类似，不做赘述。

3.2 仿真验证

新建一个工程 MACtrl，把 SCPU 工程中的 MACtrl.v, MEM 核和 LEDCtrl.v 拷贝过来，并且新建仿真文件 MACtrl_tb.v。（新建工程而不是在原工程仿真的原因将在实验心得部分讨论）

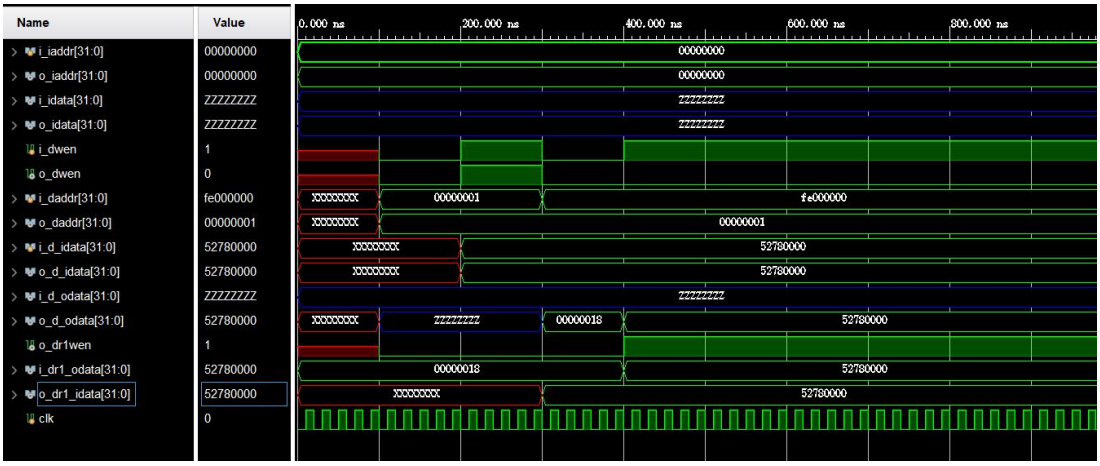
MACtrl_tb.v 代码如下：

```
module MACtrl_tb;
       //facing core
       reg [31:0]i_iaddr;
       wire [31:0]o_idata;
       reg i_dwen;
       reg [31:0]i_daddr;
       reg [31:0]i_d_idata;
       wire [31:0]o_d_odata;
       //facing IMem
       wire [31:0]o_iaddr;
       wire [31:0]i_idata;
       //facing DMem
```

```
        wire o_dwen;
        wire [31:0]o_daddr;
        wire [31:0]o_d_idata;
        wire [31:0]i_d_odata;
        //facing DR1
        wire o_dr1wen;
        wire [31:0]o_dr1_idata;
        wire [31:0]i_dr1_odata;
        reg clk;

    MACtrl test_MAC(...);//omit details.

    Mem m0(
        //for IMem
        .i_addr(o_iaddr),
        .i_data(i_idata),
        //for DMem
        .clk(clk),
        .d_wen(o_dwen),
        .d_addr(o_daddr),
        .d_i_data(o_d_idata),
        .d_o_data(i_d_odata)
    );

    LEDCtrl dr1(
        .wen(o_dr1wen),
        .i_data(o_dr1_idata),
        .o_data(i_dr1_odata)
    );

    always begin clk = ~clk; #10; end
    initial begin
       clk = 0;
       //Imem 读指令
       i_iaddr = 0;
       #100;

       //对 DMem 只读
       i_dwen = 0;
       i_daddr = 1;
       #100;

       //对 DMem 改写
       i_dwen = 1;
       i_d_idata = 32'h52780000;
       #100;

       //对 LED 只读
       i_dwen = 0;
       i_daddr = 32'hFE000000;
       #100;

       //对 LED 改写
       i_dwen = 1;
       #100;
    end
endmodule
```

仿真代码分成三部分。首先定义要测试的变量，然后引用 MACtrl、Mem 和 LEDCtrl 组件。MACtrl 组件形参抄写 MACtrl.v 即可，把 input 的变量改为 reg，把 output 变量改为 wire。分析 Mem 组件的形参表可知，从 MACtrl 中获取 iaddr, dwen, daddr, d_idata，然后把读取到的指令或数据内容传回 MACtrl。LEDCtrl 组件同理。

第三部分是仿真测试。因为一共有读指令、读数据、改写数据、读 LED、改写 LED 五个功能，所以我将测试分成相应的五个部分进行。结合仿真波形图分析如下：



0~#100，读指令。输入 i_addr=0, o_addr = i_addr = 0，而 Mem 中各个地址内的内容尚未定义，所以都是高阻态，因此 o_idata = i_idata = ZZZZZZZZ。

#100~#200，读数据。输入 i_daddr = 1, o_daddr = i_daddr = 1，访问到 i_d_odata = ZZZZZZZZ，所以 o_d_odata 也是高阻态。

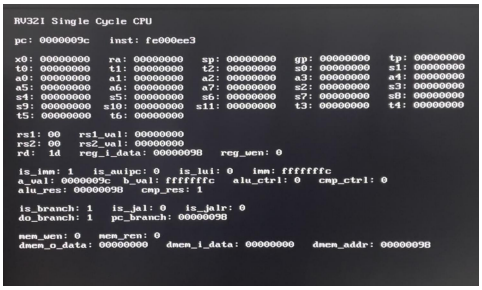
#200~#300，写数据。置 i_dwen = 1, o_dwen = i_dwen = 1。输入要写入的数据 i_d_idata=52780000, o_d_i = i_d_idata = 52780000。但是 Mem 核是黑匣子，我不知道其内部是怎样将数据写入地址的，由 i_d_odata = ZZZZZZZZ 可见 Mem 并没有写入成功，从而 o_d_odata = i_d_odata 也显示为高阻态。这里是仿真中的一个小 Bug。

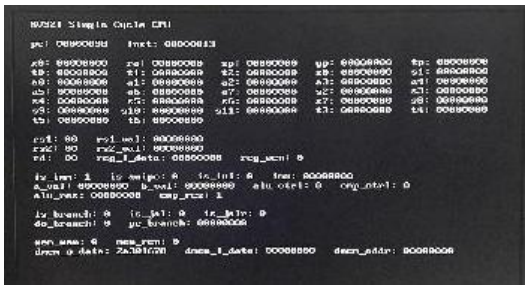
#300~#400，读 DR。输入 i_dwen = 0，i_daddr = 32'hFE000000，访问到 i_dr1_odata = 32’h00000018，所以 o_d_odata = i_dr1_odata = 32’h00000018。

#400~#500，写 DR。置 i_dwen = 1, o_dr1wen = i_dwen = 1。要写入的数据 i_d_idata=52780000。LED 模块成功写入后 i_dr1_odata = 52780000，从而 o_d_odata = 52780000。

4 生成 CPU 测试框架的 bit 流，上板验证

点击 Generate Bit Stream 后，工程会依次进行 Synthesis, Run Implementation 和 Generate Bit Stream 三步。生成 bit 流后点击 Open Hardware Manger，点击 Auto Connect，Device 选中 Sword 板。上板后，电脑屏幕出现如下画面，说明 bit 流文件没有问题。





三、实验心得

这一部分主要叙述我做 Lab1 时遇到的 Bug 或困难。

困难一：信息获取不足，代码理解有困难。由于我学过 ICS，所以我对 MemoryAccessController 的需求理解得比较快，虽然把 memory 拆成了 instruction 和 data 两部分让我不太适应。但是我因为最开始没有注意到 PPT 最后一页的提示（i 表示 input，o 表示 output），所以当实际阅读代码时，对 i 和 o 的同名变量感到疑惑，不知道它们分别要干什么。此外我注意到在 Top.v 当中 CPU 已经和 Memory 直接相连，所以觉得 MACtrl 并没有起到什么实际作用（后来才知道实验要求之一就是要把 Top 中原有的部分线断开）。经过与同学的讨论和反复研究 PPT 上的需求，我顿悟了要把 i、o 的同名变量相连。

困难二：Verilog 语法不熟悉。MACtrl 最初的代码是用 if 语句写的：

```
if (i_daddr != 32'hFE000000) begin//这种语法为什么会错误?报错：'i_daddr' is
not a constant
       assign o_dr1wen = 0;
       assign o_dwen = i_dwen;
       assign o_daddr = i_daddr;
       assign o_d_idata = i_d_idata;
       assign o_d_odata = i_d_odata;
       end
   else begin
       assign o_dwen = 0;
       assign o_dr1wen = i_dwen;
       assign o_dr1_idata = i_d_idata;
       assign o_d_odata = i_dr1_odata;
       end
```

但是会报错'i_daddr' is not a constant。经过同学提醒得知 if 语句等都必须嵌套在 always 语句当中才可以正常使用。所以在独自实现代码时我改用了 assign+条件表达式。

困难三：仿真过程的种种困难。

首先，不能对整个 Top 仿真，因为 Top 的 input 和 output 当中根本没有我们需要测试的 addr 和 data 系列变量。

其次，在原工程中我尝试了对 MACtrl 进行单独仿真。但由于我一开始不知道要将 MACtrl_tb.v 设为顶层模块，所以每次仿真默认仿真 Top.v，所以我不得已又创建了一个 MACtrl 工程单独仿真。

第三，仿真形式。我最开始仅对 MACtrl 仿真，由仿真程序模拟 CPU、Mem、LEDCtrl 的行为，但是发现没有办法模拟写入的过程。所以我认为需要同时引入 MACtrl、Mem 和 LEDCtrl 三个组件，由仿真程序模拟 CPU 的行为。

第四，Mem 无法写入。Mem 是封装好的无源文件的 IP 核，所以不知道具体实现。由仿真结果来看，似乎每一个地址的内容都是高阻态 ZZZZZZZZ。而且 Mem 无法执行写入数据操作。

困难四：生成 Bit Stream 时报错 Multiple Driven

该 Error 信息与困难一中的理解有关。如果不修改 Top 中 CPU 和 Mem 的直接连线，同时新加了 MACtrl 的输入输出，那么 addr、data 等变量就同时有两个 input 来源，于是发生 Multiple Driven。只要新增一组同名变量，即可消除此问题。

困难五：VGA 必须要连接至电脑才能显示调试信息。在 Sword 板上会显示 No Sync。