

**《计算机组成与设计》
课程实验指导书**

(2016 级)

交大电院

2018 年 04 月

内容提要：

针对电院 IEEE 试点班同学（2016 级）的《计算机组成与设计》课程，该课程实验的前导基础为《数字电路与逻辑设计》，知识点包括：基本电路元件（电阻、电容、电感、二极管、三极管、场效应管）、逻辑门电路、组合逻辑与时序逻辑、小规模集成电路（译码器、多路选择器、存储器等）、大规模可编程逻辑阵列（FPGA）器件技术等，以及利用这些器件进行数字系统设计的逻辑原理与工程方法。

实验中涉及 Verilog HDL 硬件描述语言的使用和 Quartus II（Ver 13.x）与 Modelsim ALTERA（Ver 10.x）等工具软件的使用。

课程实验基本内容包括：熟悉 Verilog HDL 和 Quartus II 设计环境的基于 FPGA 的多功能秒表设计实验、包含 I/O 的单周期和 5 级流水 CPU 设计实验。要求同学们能够较系统地通过实验循序渐进地认识和实践计算机系统设计的基本原理和方法。

目录

第一章	课程实验简介及要求	1
第二章	课程实验	4
2.1	实验一：基于 Verilog 和 FPGA 的多功能秒表设计	6
2.2	实验二：基本单周期 CPU 设计	10
2.3	实验三：5 段流水 CPU 设计	21
2.4	实验四：外部 I/O 及接口扩展实验	26
附件：	35

第一章 课程实验简介及要求

针对电院 IEEE 试点班同学（2016 级）的《计算机组成原理》课程学习，为深入理解计算机组成与设计的基本原理，培养从硬件和软件两方面全局考虑计算机针对问题求解所进行的系统设计思想，掌握针对一定规模系统的设计实践能力，配合课堂教学内容开设以下 3 个基本实验：

1. 基于 Verilog HDL 和 FPGA 的多功能秒表实验；
2. 单周期 CPU 设计实验（含 I/O）；
3. 5 级流水线 CPU 设计实验（含 I/O）；

其中，实验 1 为针对 Verilog HDL 和 Quartus II 实验环境的基本认知和练习实验。

1.1 课程实验目的：

1. 通过实际设计实验，在课堂理论学习的基础上，进一步从“做”中学，深刻理解计算机 CPU 的自动工作原理及其基本设计方法。
2. 通过对计算机 5 大组成部分的设计实践，尤其通过对控制器的设计，建立和深刻理解各组成部件围绕“自动运行存储程序”所运用的全局协调控制思想。
3. 通过设计实践，进一步掌握一定规模系统设计中的组合逻辑和时序逻辑的运用方法。
4. 通过掌握指令流水线的设计和实现方法，领会其它复杂算法的硬件流水实现的设计思想，理解问题求解中软件和硬件的全局平衡（折衷）设计思想。
5. 使同学掌握利用 Verilog HDL 硬件描述语言基于 FPGA 大规模可编程逻辑器件进行一定规模逻辑系统设计的基本设计方法和实现方法。
6. 实验训练所掌握的设计知识和实验技能，是进一步构建 SOPC 系统的理论和实践基础，也是未来从事高级计算机系统结构设计、和系统软件设计的理论和实践基础。
7. 掌握终生受用的基本思想与方法，在系统级上认识计算机硬件和软件，提升对计算问题的认知、分析和解决水平，增强系统能力，体验实现自动计算的乐趣。

1.2 实验要求：

计算机组成原理，是研究计算机是如何构成的、以及各硬件部件是如何自动工作的课程。

三个课程实验，从认识和掌握单周期 CPU 的各个组成部件的工作和设计原理、及其整体自动工作的原理开始，到设计处理能力更强的 5 级流水线 CPU，并扩充输入/输出接口方法，每一个实验都有明确的目的性和对知识点的针对性，要求同学们能够深刻理解和领会每一个实验中的知识点，掌握和积累相应的设计和实验技能。

“创新 = 厚基础 + 善思维 + 常实践”。要求同学们能理论联系实际，将课堂所学知识在实验中进行实际设计应用，并充分发挥自己的想象力和创造力。课程实验要求完成并掌握这三个基本实验内容，但不局限于这三个实验，鼓励同学们设计出更富想象力和创造性的实验。

为了充分有效地利用好有限的实验课时间，要求同学们在实验过程中，注意并遵循以下实验要求。

1.2.1 实验前应做好准备工作：

计算机组成与设计实验虽然是一门专门设置的实践性内容，但毕竟实验学时数有限，大部分实验原理及器件技术不可能在实验课上详细讲解，尤其是 Altera/Intel DE1-SOC 实验平台的详细构成及其上器件的数据手册，以及 Verilog HDL 硬件描述（程序设计）语言，课堂上讲授的主要是设计和运用方法。也就是说，同学们在做实验之前必须要清楚掌握实验原理和方法，并要做到对每一个实验所涉及的硬件部件的工作原理和应用方法都心中有数。这就要求同学们在实验课前必须做好充分的准备：

1. 仔细阅读实验指导书的实验内容及要求，明确实验目的，清楚所要掌握的知识点和实验技能。
2. 按照预习要求，提前阅读指定文档，掌握相关内容和知识点。
3. 写好预习报告。即实验前先完成实验报告中的前三项内容（要求见后），特别是实验任务，必须在课前认真完成，否则不允许进入实验室进行实验。

1.2.2 实验注意事项：

为了在实验中培养学生严谨的科学态度和工作作风，确保人身和设备的安全，顺利有效地完成实验任务，达到预期的实验目的，同学们做实验时应注意以下几点：

1. 严格遵守实验室的管理规范和制度要求。
2. 在进入实验室后，首先检查本次实验所领用的实验器材，包括 DE1-SOC 实验板、稳压电源模块、USB 连接线、（万用表）、（示波器）等，判断其是否正常，同时要掌握其使用方法。

3. 实验前, 应检查实验板电源开关处于断开状态, 确认实验板电源线接入正确, 检查电路板上没有异常状况后, 再与计算机通过 **USB** 接口正确连接。

4. 接好线路、连接稳定后, 一定要认真复查, 确保无误后, 方可给电路板上电。如无把握, 应请老师或助教审查。

5. 只有在下载代码进行调试或测试电路接口功能时才打开电源, 其它情况下应关掉电源, 以免接线错误、或因无意间有导体碰到电路使电路瞬间短路、或带电插接连接器等使器件损坏。

6. 如有损坏实验仪器设备等情况, 必须及时向老师或助教报告, 并写出书面情况说明。

7. 保持实验室整洁、安静。(不乱扔纸屑等)。

8. 实验完成后, 须经指导教师或助教检查实验结果, 然后切断电源, 拆除实验电路, 整理并放置好实验台上的所有仪器设备及工具等, 归还领用设备, 方可离开实验室。

1.2.3 实验报告的要求:

实验结束后认真书写实验报告, 实验报告用纸规定一律用正式用纸, 并加以专门的实验报告封面, 装订整齐。实验报告所含具体内容包括以下几部分:

1. 实验目的。

2. 本次实验所用的实验平台、仪器以及其它实验器材和部件等。

3. 实验任务。这部分是实验报告最主要的内容。根据每个任务要求, 设计出符合要求的实验电路和 **Verilog** 程序代码, 并要求结合实验原理, 写出设计的全过程和实验步骤。

4. 实验总结。把做完的实验内容进行数据整理、归纳。并按实验要求, 回答或解释相应的实验现象或问题。最后再写出自己的心得体会。

第二章 课程实验

该部分的实验教学目的，是要求同学们利用 Verilog 硬件描述语言、基于大规模可编程逻辑阵列器件 FPGA，进行计算机 CPU 的逻辑功能设计，包括 RISC 风格的单周期 CPU 设计、5 级流水 CPU 设计、以及在其上的输入输出部件扩展和功能扩展。

设计实验是利用 Altera/Intel DE1-SOC 实验板进行的，其板上资源详见其用户手册（电子文档）。要求大家利用该实验板上的资源，在 FPGA 中物理实现自己的 CPU，并通过在自己设计的 CPU 上运行设计的程序代码，能够接收外部输入，并能利用板载输出设备或接口，输出程序的运行状态或结果。

第 1 个实验，要求同学设计一个多功能秒表，通过该简单实验的练习，熟练掌握 Verilog HDL 和 Quartus II 针对 FPGA 的开发环境。

第 2 个实验，要求实现单周期 CPU，能够采用查询方式接收板载按键或开关的状态，并产生相应的输出状态。

第 3 个实验，要求采用 5 级流水技术实现实验 1 中的 CPU，其中包含了针对基本的数据冲突和控制冲突的解决方法。同样能够采用查询方式接收板载按键或开关的状态，并产生相应的输出状态。

第 4 个实验部分，属于对 I/O 扩展的实验内容解释，要求其内容贯穿于实验二的单周期 CPU 设计实验和实验三的 5 级流水 CPU 设计实验。

对于希望能够再进行一些基本实验内容要求之外扩展实验的同学，建议同学能够利用 DE1-SOC 实验板上的资源，自由设计自己的作品。如驱动 VGA 接口，在标准显示器上展现出各种信息等。

实验开发环境为 Altera - Quartus II 13.1，软件可从 Altera 网站上进行免费自由下载，网址为 www.altera.com 或 www.altera.com.cn。

具体细化网址为：<http://dl.altera.com/13.1/?edition=web>，勾选的必需和必要软件如图 2-1 所示。

关于实验硬件平台 DE1-SOC 的资料，包括已发送给大家的部分资料等，更多更详细的内容均可以从台湾友晶科技的以下网页下载：

<http://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&CategoryNo=165&No=836&PartNo=1>。

或从其中文网址下载：

<http://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=China&CategoryNo=203&No=870&PartNo=4>。

其上还包括多种应用设计例程下载等。

关于板载硬件资源，如按键接入电路、LED 的连接电路、7 段数码管的连接电路、VGA 接口，以及它们和 FPGA 的连接管脚定义等，更多详细内容参考其用户应用手册。



图 2-1 勾选下载 QuartusII 13.1 及其支持软件包

2.1 实验一：基于 Verilog 和 FPGA 的多功能秒表设计

实验目的：

1. 初步掌握利用 Verilog 硬件描述语言进行逻辑功能设计的原理和方法。
2. 理解和掌握运用大规模可编程逻辑器件进行逻辑设计的原理和方法。
3. 理解硬件实现方法中的并行性，联系软件实现方法中的并发性。
4. 理解硬件和软件是相辅相成、并在设计 and 应用方法上的优势互补的特点。
5. 本实验学习积累的 Verilog 硬件描述语言和对 FPGA/CPLD 的编程操作，是进行后续《计算机组成原理》部分课程实验，设计实现计算机逻辑的基础。

实验内容和任务：

1. 运用 Verilog 硬件描述语言，基于 DE1-SOC 实验板，设计实现一个具有较多功能的计时秒表。
2. 要求将 8 个数码管设计为具有“时：分：秒：毫秒”显示，按键的基本控制动作有 3 个：“计时复位”、“计数/暂停”、“显示暂停/显示继续”。功能能够满足马拉松或长跑运动员的计时需要。
3. 利用示波器观察按键的抖动，设计按键电路的消抖方法。
4. 在实验报告中详细报告自己的设计过程、步骤及 Verilog 代码。

预习内容：

1. 学习和掌握 Verilog HDL 硬件描述语言。
2. 熟悉针对 Altera 公司 FPGA 开发的 Quartus II 13.1 软件开发界面。
3. 掌握利用 Modelsim ALTERA 10.x 仿真软件进行设计功能验证的方法。

实验仪器：

Altera – DE1-SOC 实验板	1 套
示波器	1 台
数字万用表	1 台

实验参考 Verilog HDL 代码：

该部分提供了模块定义及部分代码，仅供参考，并不限定同学们的设计方法。

DE1-SOC 实验板上的 8 个 7 段数码管，其显示原理和具体的硬件连接布线图，

及各控制信号与 FPGA 芯片的管脚对应关系，请参考 DE1-SOC User Manual。

该示例仅使用了 6 个数码管，仅供参考。

图 2-1-1 中，仅示意了部分信号和变量的定义，仅供参考。各构成模块的设计细节没有限定，以方便同学采用自己的设计思想。

```
// =====  
//  
// The counter is designed by a series mode. / asynchronous mode. 即异步进位  
// use "=" to give value to hour_counter_high and so on. 异步操作/阻塞赋值方式  
//  
// 3 key: key_reset/系统复位, key_start_pause/暂停计时, key_display_stop/暂停显示  
//  
// =====  
module stopwatch_01(clk,key_reset,key_start_pause,key_display_stop,  
// 时钟输入 + 3 个按键; 按键按下为 0 。板上利用施密特触发器做了一定消抖, 效果待测试。  
hex0,hex1,hex2,hex3,hex4,hex5,  
// 板上的 6 个 7 段数码管, 每个数码管有 7 位控制信号。  
led0,led1,led2,led3  
);  
// LED 发光二极管指示灯, 用于指示/测试程序按键状态, 若需要, 可增加。 高电平亮。  
  
input clk,key_reset,key_start_pause,key_display_stop;  
output [6:0] hex0,hex1,hex2,hex3,hex4,hex5;  
output led0,led1,led2,led3;  
reg led0,led1,led2,led3;  
  
reg display_work;  
// 显示刷新, 即显示寄存器的值 实时 更新为 计数寄存器 的值。  
reg counter_work;  
// 计数(计时)工作 状态, 由按键“计时/暂停”控制。  
parameter DELAY_TIME = 10000000;  
// 定义一个常量参数。 10000000 ->200ms;  
  
// 定义 6 个显示数据(变量)寄存器:  
  
reg [3:0] minute_display_high;  
reg [3:0] minute_display_low;  
reg [3:0] second_display_high;  
reg [3:0] second_display_low;  
reg [3:0] msecond_display_high;  
reg [3:0] msecond_display_low;  
  
// 定义 6 个计时数据(变量)寄存器:  
  
reg [3:0] minute_counter_high;  
reg [3:0] minute_counter_low;
```

```

reg [3:0] second_counter_high;
reg [3:0] second_counter_low;
reg [3:0] msecond_counter_high;
reg [3:0] msecond_counter_low;

reg [31:0] counter_50M; // 计时用计数器， 每个 50MHz 的 clock 为 20ns。
// 若选择板上的 50MHz 时钟，需要 500000 次 20ns 之后，才是 10ms。

reg      reset_1_time;      // 消抖动用状态寄存器  -- for reset KEY
reg [31:0] counter_reset;    // 按键状态时间计数器
reg      start_1_time;      //消抖动用状态寄存器  -- for counter/pause KEY
reg [31:0] counter_start;    //按键状态时间计数器
reg      display_1_time;    //消抖动用状态寄存器 -- for KEY_display_refresh/pause
reg [31:0] counter_display;  //按键状态时间计数器

reg      start;             // 工作状态寄存器
reg      display;           // 工作状态寄存器

// sevenseg 模块为 4 位的 BCD 码至 7 段 LED 的译码器，
//下面实例化 6 个 LED 数码管的各自译码器。
sevenseg LED8_minute_display_high ( minute_display_high, hex5 );
sevenseg LED8_minute_display_low ( minute_display_low,  hex4 );

sevenseg LED8_second_display_high( second_display_high,  hex3 );
sevenseg LED8_second_display_low ( second_display_low,   hex2 );

sevenseg LED8_msecond_display_high( msecond_display_high, hex1 );
sevenseg LED8_msecond_display_low ( msecond_display_low,  hex0 );

always @ (posedge clk)      //每一个时钟上升沿开始触发下面的逻辑
begin

    //此处功能代码省略，由同学自行设计。

end

endmodule

//其余功能代码，由同学自行设计。

```

//4bit 的 BCD 码至 7 段 LED 数码管译码器模块

```

module sevenseg ( data, ledsegments);
input [3:0] data;
output ledsegments;
reg [6:0] ledsegments;

```

```

always @ (*)
    case(data)
        //      gfe_dcba      // 7 段 LED 数码管的位段编号
        //      654_3210      // DE2 板上的信号位编号
        0: ledsegments = 7'b100_0000; // DE2C 板上的数码管为共阳极接法。
        1: ledsegments = 7'b111_1001;
        2: ledsegments = 7'b010_0100;
        3: ledsegments = 7'b011_0000;
        4: ledsegments = 7'b001_1001;
        5: ledsegments = 7'b001_0010;
        6: ledsegments = 7'b000_0010;
        7: ledsegments = 7'b111_1000;
        8: ledsegments = 7'b000_0000;
        9: ledsegments = 7'b001_0000;
        default: ledsegments = 7'b111_1111; // 其它值时全灭。
    endcase
endmodule

```

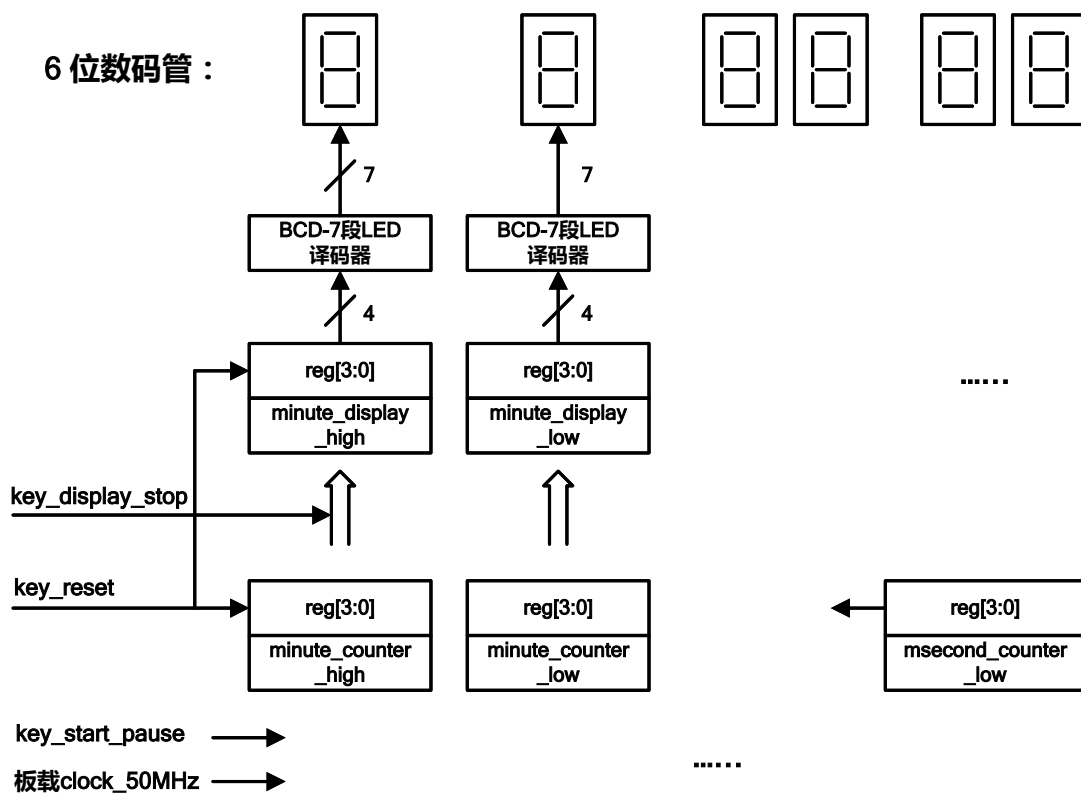


图 2-1-1 实验一 参考逻辑设计

2.2 实验二：基本单周期 CPU 设计

2.2.1 实验目的：

1. 理解计算机 5 大组成部分的协调工作原理，理解存储程序自动执行的原理。
2. 掌握运算器、存储器、控制器的设计和实现原理。重点掌握控制器设计原理和实现方法。
3. 掌握 I/O 端口的设计方法，理解 I/O 地址空间的设计方法。
4. 会通过设计 I/O 端口与外部设备进行信息交互。

2.2.2 实验内容：

1. 采用 Verilog HDL 在 quartus II 中实现基本的具有 20 条 MIPS 指令的单周期 CPU 设计。
2. 利用实验提供的标准测试程序代码，完成仿真测试。
3. 采用 I/O 统一编址方式，即将输入输出的 I/O 地址空间，作为数据存取空间的一部分，实现 CPU 与外部设备的输入输出端口设计。实验中可采用高端地址。
4. 利用设计的 I/O 端口，通过 lw 指令，输入 DE2 实验板上的按键等输入设备信息。即将外部设备状态，读到 CPU 内部寄存器。
5. 利用设计的 I/O 端口，通过 sw 指令，输出对 DE2 实验板上的 LED 灯等输出设备的控制信号（或数据信息）。即将对外部设备的控制数据，从 CPU 内部的寄存器，写入到外部设备的相应控制寄存器（或可直接连接至外部设备的控制输入信号）。
6. 利用自己编写的程序代码，在自己设计的 CPU 上，实现对板载输入开关或按键的状态输入，并将判别或处理结果，利用板载 LED 灯或 7 段 LED 数码管显示出来。
7. 例如，将一路 4bit 二进制输入与另一路 4bit 二进制输入相加，利用两组分别 2 个 LED 数码管以 10 进制形式显示“被加数”和“加数”，另外一组 LED 数码管以 10 进制形式显示“和”等。（具体任务形式不做严格规定，同学可自由创意）。
8. 在实验报告中，汇报自己的设计思想和方法；并以汇编语言的形式，提供采用以上自行设计的指令集的作品应用功能的程序设计代码，并提供程序主要流程图。

2.2.3 预习内容：

1. 实验前仔细阅读 DE1-SOC User Manual 及相关用户应用数据手册，学习并掌握其板载相关资源的工作原理、连接方式、和应用注意事项。
2. 根据课程所讲单周期 CPU 设计原理，提前设计并仿真实现相关设计代码。

2.2.4 实验器材：

DE1-SOC 实验板套件	1 套
万用表	1 台
示波器	1 台

2.2.5 实验参考：

1. 在 Quartus II 中利用 Verilog HDL 进行系统设计时，需首先设计顶层文件。顶层文件也就是从系统最高层看整个系统时，最顶层所呈现的各个组成模块以及它们之间的互联关系。顶层文件中的顶层 **module** 的每一个模块表现为一个 **module** 模块的实例化，**module** 原型可以在该项目工程所包含的其它.v 文件中定义。顶层模块的文件名需要与工程文件名相同，或在建立之初在图 2-2-1 所示界面指定你的顶层文件名，即 **top-level design entry**。

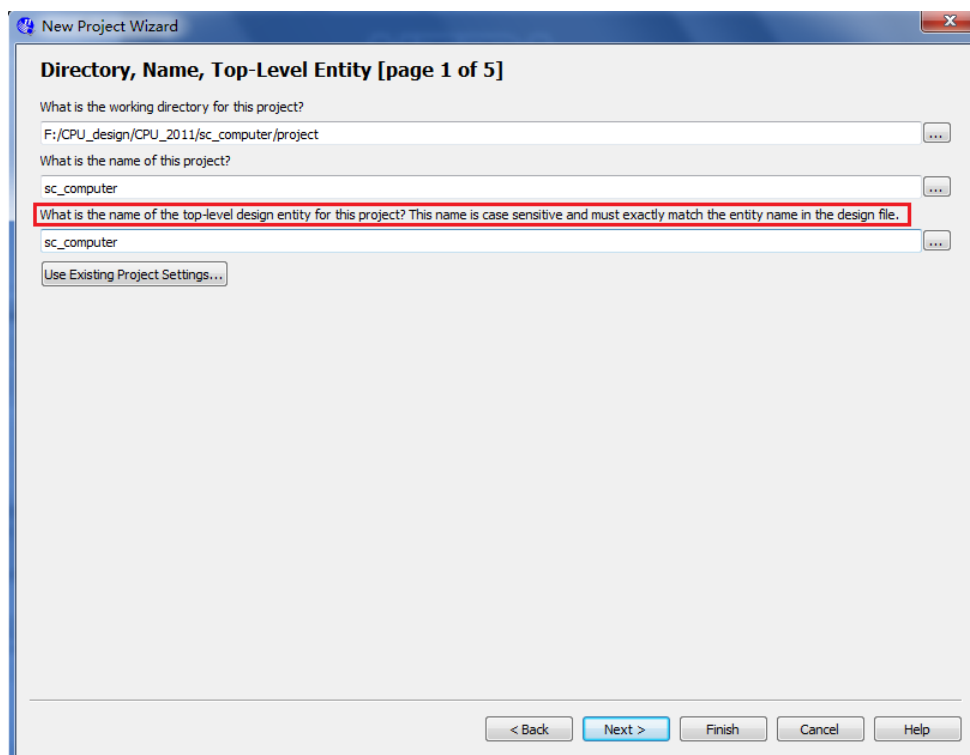


图 2-2-1. 建立工程时对顶层文件入口的设置界面

2. 实验中的顶层文件结构可参考如下所示代码进行设计：（仅用于参考方法，不限定同学自己的设计风格和方法）

顶层代码：

```
module sc_computer ( resetn, clock, mem_clk, pc, inst, aluout, memout, imem_clk, dmem_clk);
//定义顶层模块 sc_computer，作为工程文件的顶层入口，如图 1-1 建立工程时指定。
    input  resetn,clock,mem_clk;
//定义整个计算机 module 和外界交互的输入信号，包括复位信号 resetn、时钟信号
// clock、以及一个频率是 clock 两倍的 mem_clk 信号。这些信号都可以用作仿真验
//证时的输出观察信号。
    output [31:0] pc, inst, aluout, memout;
//模块用于仿真输出的观察信号。缺省为 wire 型。
    output      imem_clk,dmem_clk;
//模块用于仿真输出的观察信号，用于观察验证指令 ROM 和数据 RAM 的读写时序。
    wire  [31:0]  data; // 模块间互联传递数据或控制信息的信号线。
    wire      wmem; // 模块间互联传递数据或控制信息的信号线。
    sc_cpu  cpu (clock,resetn,inst,memout,pc,wmem,aluout,data); // CPU module.
// 实例化了一个 CPU 模块，其内部又包含运算器 ALU 模块、控制器 CU 模块等。
// 在 CPU 模块的原型定义 sc_cpu 模块中，可看到其内部各模块构成。
    sc_instmem  imem (pc,inst,clock,mem_clk,imem_clk); // instruction memory.
// 指令 ROM 存储器 imem 模块。模块原型由 sc_instmem 定义。
// 由于 Altera 的 Cyclone 系列 FPGA 只能支持同步的 ROM 和 RAM，读取操作需要
// 时钟信号。
// 示例代码中采用 Altera 公司 quartus 提供的 ROM 宏模块 lpm_rom 实现的，需要读
// 取时钟，该 imem_clk 读取时钟由 clock 信号和 mem_clk 信号组合而成，具体时序
// 可参考模块内的相应代码。imem_clk 信号作为模块输出信号供仿真器进行观察。
// 宏模块 lpm_rom 的时序要求参见其时序图。
    sc_datamem  dmem (aluout,data,memout,wmem,clock,mem_clk,dmem_clk ); // data memory.
// 数据 RAM 存储器 dmem 模块。模块原型由 sc_datamem 定义。
// 由于 Altera 的 Cyclone 系列 FPGA 只能支持同步的 ROM 和 RAM，读取操作需要
// 时钟信号。
// 示例代码中采用 Altera 公司 quartus 提供的 RAM 宏模块 lpm_ram_dq 实现的，
// 需要读写时钟，该 dmem_clk 读写时钟由 clock 信号和 mem_clk 信号组合而成，
// 具体时序可参考模块内的相应代码。dmem_clk 信号同时作为模块输出信号供
// 仿真器进行观察。
// 宏模块 lpm_ram_dq 的时序要求参见其时序图。
endmodule
```

3. 以上顶层代码表明，该单周期 sc_computer 由 CPU、指令 ROM、数据 RAM 共 3 个模块组成，这 3 个模块间由定义为 wire 型（信号缺省就是 wire 型）连接信号线的多路数据连接信号线通过各模块的输入输出端口（信号）进行互联。同层同名的信号线连通不同模块的端口，一个模块的输出信号，可作为一个或多个模块的输入信号。编写代码时注意定义时的线宽需保持一致。
4. 实验一给同学们提供了几乎全部的实验代码，同学们设计补齐需要填充的代码后，尽快熟悉和精通 quartus 下基于 Verilog HDL 的系统设计方法。

5. 实验代码的工程文件 `sc_computer.qpf` 所包含的相关源代码文件如图 2-2-2 所示，图中右侧为测试输入 `sc_computer_test_wave_01.vwf` 的仿真运行输出。
6. 建立的示例仿真输入波形文件如图 2-2-3 所示，同学们注意 3 个输入信号 `resethn`、`clock`、`mem_clk` 的时序关系。为观察直观，其它的输出观测信号 `imem_clk`、`dmem_clk`、`pc`、`inst`、`aluout`、`memout` 设置为“uninitialized”状态。
7. 同学们需独立自主地采用统一编址的 I/O 端口扩展方法，设计自己针对 DE1-SOC 板载外围设备进行控制接口的控制寄存器，即与外部设备进行互联的端口控制寄存器。提示：可在 `sc_datamem.v` 模块文件中，通过对输入的 `lw` 或 `sw` 指令的地址的判断，实现对数据 RAM 和 IO 控制寄存器组的区分、和分别控制。
8. 该实验的示例代码是学习基础，后续实验二中，流水线的 verilog 代码，主要由同学自行完成，并进行自主的创新设计。希望由你采用你自己设计的流水线 CPU，利用你设计的 I/O 控制端口，通过运行程序，可在 VGA 显示器上展现你的设计。

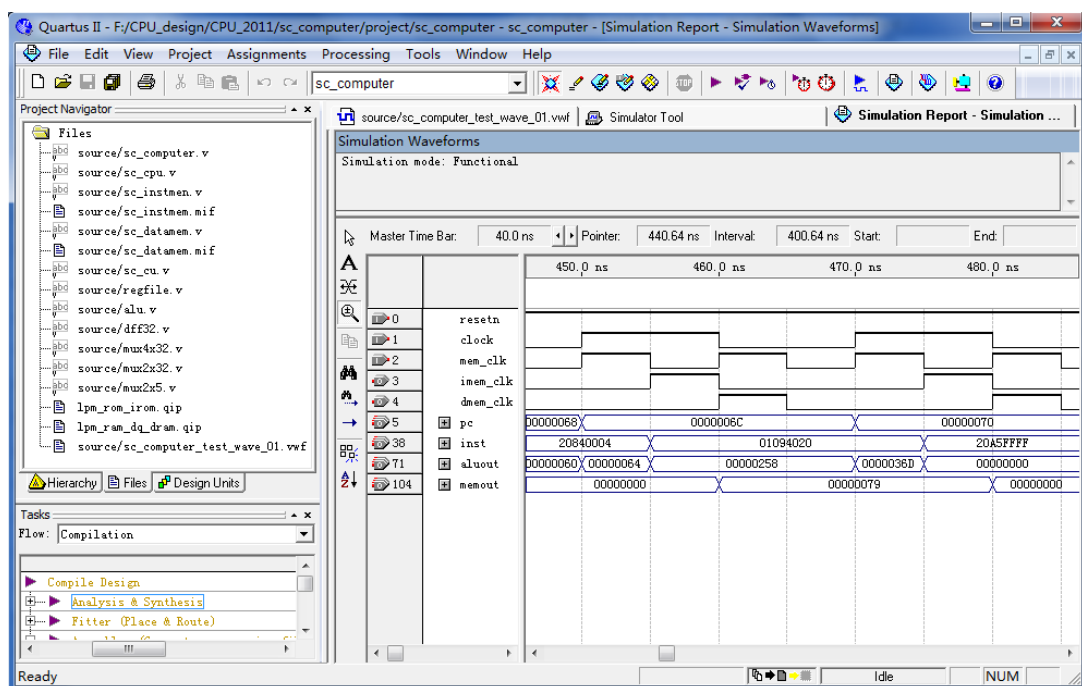


图 2-2-2. 工程文件所包含的相关文件及仿真运行输出示例波形

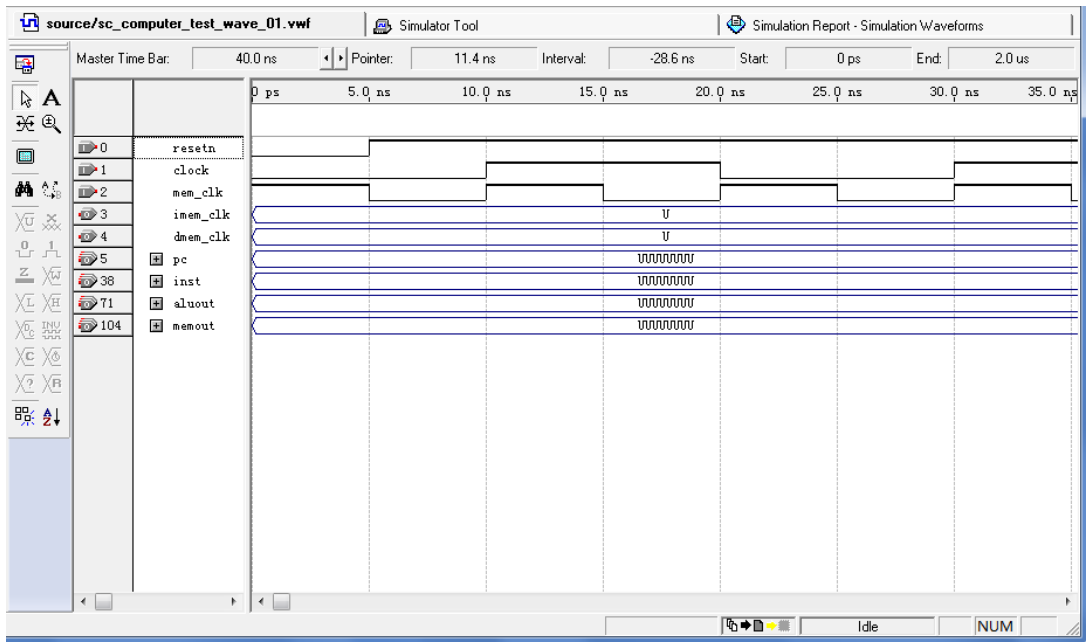


图 2-2-3. 建立的仿真运行测试输入文件波形

2.2.6 设计中的相关问题解答：

单周期 CPU 设计实验中的几个问题：

【问题 1】 如何在设计中加入 Quartus II 中 Altera 提供的逻辑器件库中的模块？

【解答】 如果从最基本的逻辑单元来设计和构建一个大型的逻辑系统，事无巨细的逻辑设计会使工作量巨大，设计效率不高。Quartus II 中 Altera 提供了较为丰富的基本逻辑宏模块设计库，包括如加法器、乘法器、除法器、log 运算器、sqrt 平方根运算器等 ALU 部件，以及多种接口部件等（详见 Quartus II 使用文档）。

另外，Altera 提供有嵌入式的 Nios II 软 CPU 核，可以在 FPGA 中构建 CPU 核，甚至多核，用以实现功能更强大的 SOPC（System On a Programmable Chip）系统。

实验中所设计的指令 ROM 和数据 RAM 存储器，可以利用 Quartus II 中提供的 ROM 和 RAM 宏模块进行设计。由厂家提供的这些宏模块设计，应该是根据其 FPGA 器件的特点，所实现的最优设计。

Quartus II 中所提供的 ROM 和 RAM 设计，均为同步器件，即包含有同步信号，这和单周期 CPU 原理设计中的异步 ROM 和异步 RAM 是有所不同的。采用具有同步信号的设计，能够使信号的变化和传输（延迟）过程等更加明确，增强了电路设计的确定性。

对这些宏模块的应用，在 Quartus II 中可以进行配置并选择生成为 Verilog 的.v 文件的方式，该.v 文件中包含有对所选用宏模块的模块定义，类似于 C 语言中对过程函数的原型声明，这样，其它模块就可以利用该宏模块声明来实例化其具体应用。

在设计中利用 Quartus II 所提供的逻辑器件，其方法及步骤如下：

1. 在“File”菜单中选择“New...”菜单项，出现如图 2-2-4 所示界面，选择“Block Diagram/ Schematic File”，然后点击“OK”按钮，出现图 2-2-5 所示 Block1.bdf 界面。

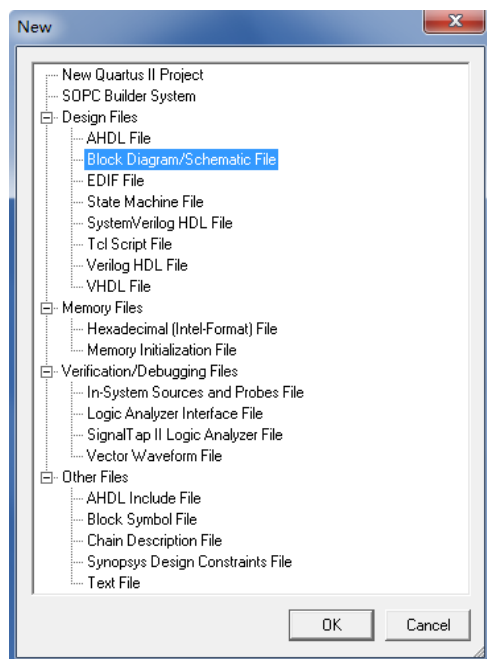


图 2-2-4. 选择建立模块文件界面

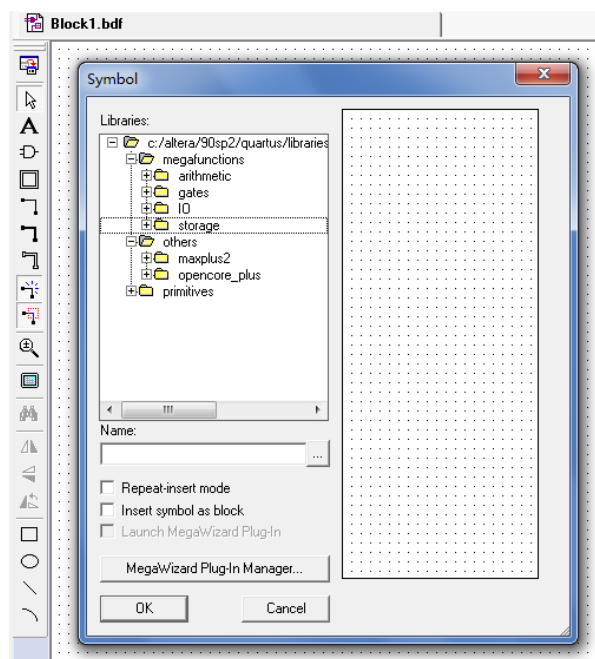


图 2-2-5. 选择库中的宏模块界面

2. 鼠标“双击”图 2-2-5 所示 Block1.bdf 界面视图，出现如图中所示的“Symbol”界面，在其左上方的库模块列表中，就可以选择所需的器件模块。
3. 在图 2-2-5 所示“Symbol”界面中，选择实验中用到的“megafunctions”下“storage”中的“lpm_rom”或“lpm_ram_dq”模块。只读 ROM 相对简单，以同步 RAM 为例，选择“lpm_ram_dq”后的界面如图 2-2-6 所示，右侧显示了该模块的初步设计功能框图，接下来可以对其特性进行配置。

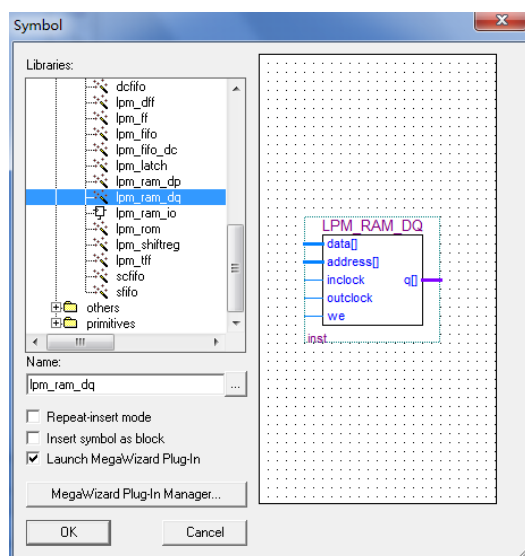


图 2-2-6.选择库中“lpm_ram_dq”宏模块

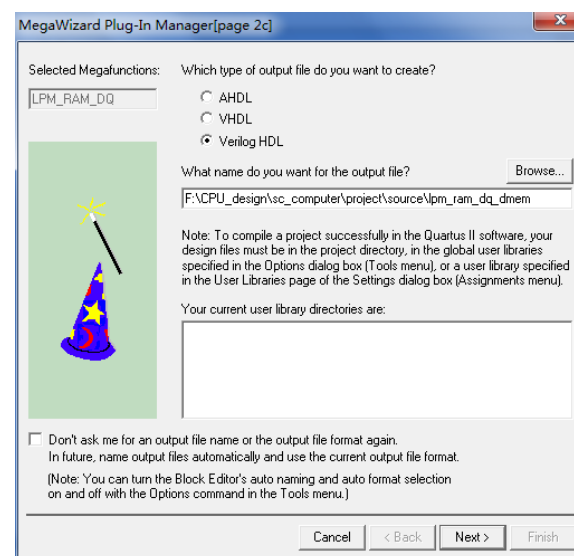


图 2-2-7.选择输出文件形式及其存放位置

4. 在图 2-2-6 所示界面，确认左下角“Launch MegaWizard Plug-In”选项被选，即可点击“OK”按钮，进入对该 RAM 的配置过程。
5. 在如图 2-2-7 所示的“MegaWizard Plug-In Manager”界面中，选择“Verilog

- HDL”文件输出形式，并指定自己接下来要生成的 .v 设计文件的存放位置及文件名。建议和自己设计的其它源文件放在同一目录下，以方便管理。然后点击“Next”。
- 在接下来的配置过程当中，可按实验中的应用需要，依次进行属性选项的配置。如图 2-2-8 至图 2-2-11 所示，依次设置 RAM 位宽为 32bits，RAM 容量为 32words；为便于代码在不同 FPGA 器件间移植，“memory block type”选择“Auto”；选择单一时钟，即 Input 和 Output 逻辑采用相同的时钟；不选“q'output port”，即 RAM 输出不需要加入寄存器缓冲输出级，否则会需要额外的一个时钟周期后才能输出 RAM 中的数据；指定实验中数据 RAM 的初始化文件“sc_datamem.mif”；最后选择生成输出 .v 模块定义文件，以及器件的工作时序波形文件。
 - “Finish”后生成的“lpm_ram_dq_dmem.v”文件中，就包含了对所建 RAM 模块的定义，在其它模块中就可以例化应用。同学可以打开该自动生成文件进行查看其结构并确认，也只有该“lpm_ram_dq_dmem.v”是工程文件中所必需的。
 - 在配置过程中，可选择“documentation”菜单中“Generate Sample Waveforms”功能，生成当时的器件工作波形图进行功能检查，查看是否是自己需要的工作时序，否则可“<back”后进行修改，直至符合自己要求。
 - 在图 2-2-11 中选择生成的器件波形文件“lpm_ram_dq_dmem_waveforms.html”中，是对所建器件功能时序的详细说明，包含有工作波形图，需要仔细查验，看是否是自己想要的。另外的“lpm_ram_dq_dmem_wave*.jpg”文件，是器件工作波形图。这一步检查很重要。
 - 由同学自行学习建立自己的“指令 ROM”存储器。

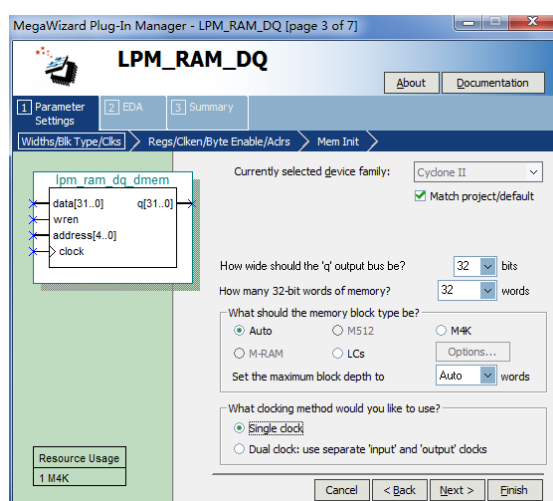


图 2-2-8. 设置 RAM 位宽、容量及时钟

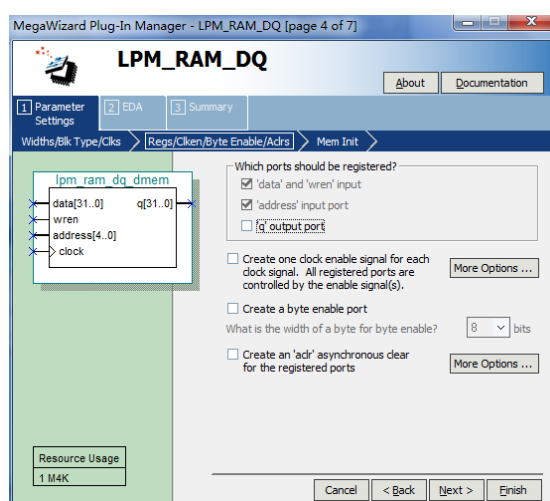


图 2-2-9. RAM 输出级不需要寄存器缓冲结构

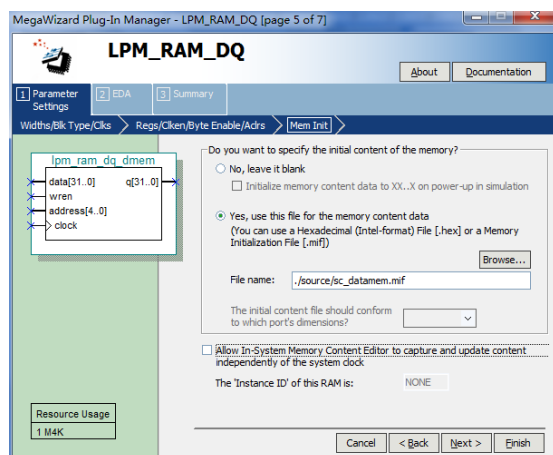


图 2-2-10. 设置 RAM 初始化数据文件

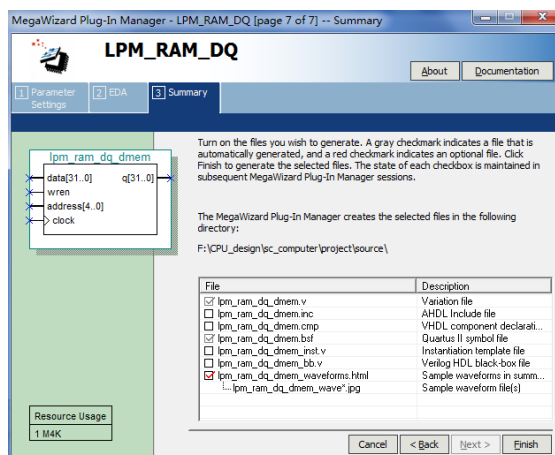


图 2-2-11. 选择输出.v 文件及器件波形图文件

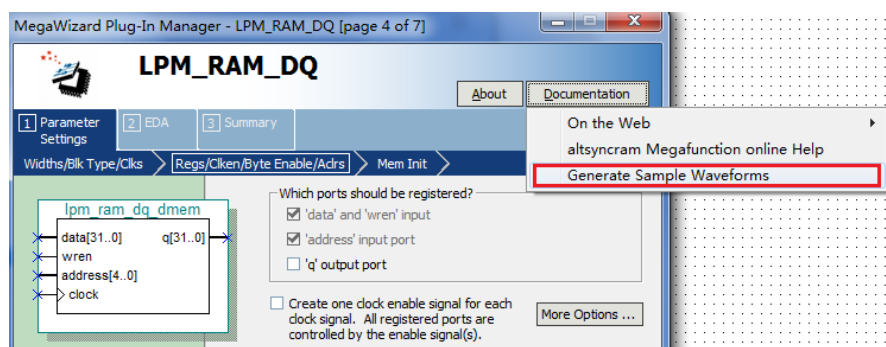


图 2-2-12. 配置过程中可随时生成器件波形文件进行功能检查

【问题 2】 单周期 CPU 设计实验中除了一个基本的 clock 时钟信号外，为什么还设计有一个 2 倍频于 clock 信号、标记为 mem_clk 的时钟信号？

【解答】 这是每个指令周期内，对各逻辑功能部件的工作时间顺序进行控制的需要。

从单周期 CPU 的设计原理上，如果指令 ROM 和数据 RAM 都为异步器件，即只要收到地址就开始送出数据，则在实现中只需要一个 clock 时钟信号控制 PC 的变化、以及整条指令的执行时间（机器周期）就够了。

但在 DE2 实验板上 Altera 的 Cyclone II 系列 FPGA 器件中，只提供同步 ROM 和同步 RAM 宏模块。在【问题 1】的解答中所建立的同步 ROM 和同步 RAM，其读写时序如图 2-2-13 和图 2-2-14 所示。

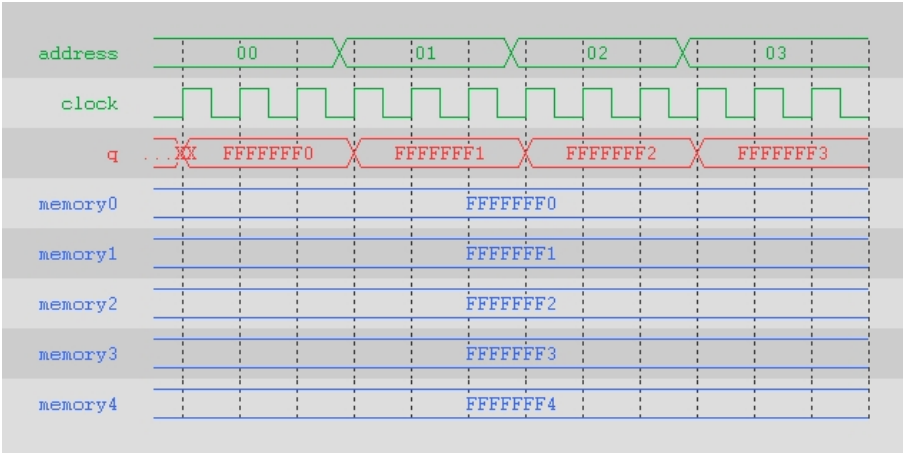


图 2-2-13. 同步 ROM 和同步 RAM 的“读”操作工作时序

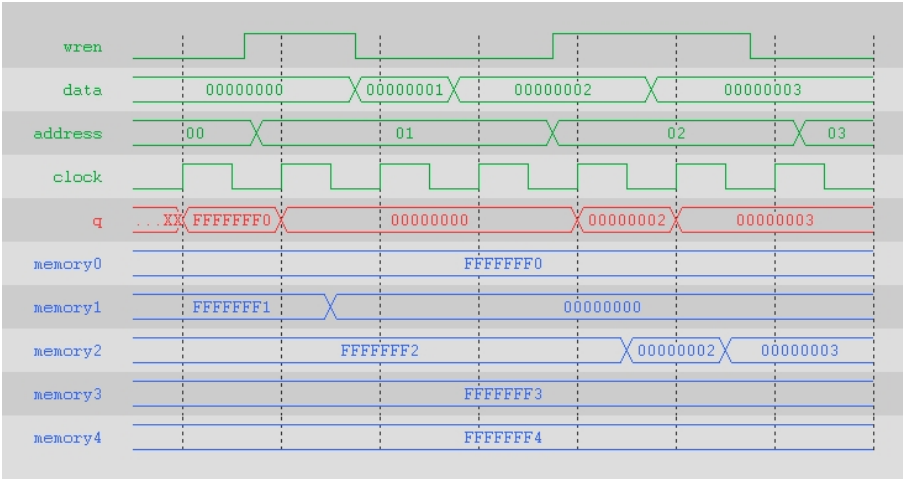


图 2-2-14. 同步 RAM 的“写”操作工作时序

考察图 2-2-13 所示同步读时序，当同步 ROM 或 RAM 接收到数据地址时，不同于异步器件在经过一段组合电路的反应延迟后送出数据，同步器件需要等到同步时钟信号 clock 上升沿时才真正送出数据（这就可以使延迟时间不尽相同的多个器件在输出级达成一致）。

考察图 2-2-14 所示同步写时序，同步 RAM 在接收到地址后，在 wren 信号高有效期间，需要等到同步时钟 clock 上升沿，才开始写动作，并将要写入的新数据马上输出到 q 输出端，即在写时可以同时读输出，满足图 2-2-13 的读时序，并在同步时钟 clock 的下降沿将数据真正写入到存储器中。

对于单周期 CPU 指令执行的一系列过程（5 个大步骤）：

①取指令：PC 在时钟上沿改变新值，按改变后并稳定的新 PC 值作为指令地址，访问指令 ROM 取出指令；②译码：指令的一部分交由控制器 CU 进行译码，另一部分至寄存器堆取操作数；③执行：ALU 等部件执行相应操作；④访问数据 MEM：读或写数据存储器；⑤回写寄存器：回写结果至目的寄存器。

以上各步骤在逻辑上是顺序执行的，在时间上是有先后顺序的，因此，对于采用有同步信号的指令同步 ROM、数据同步 RAM，在一个 CPU 指令周期内，需要安排更精细的时钟信号来协调它们的顺序工作过程。

为此，实验中设计了 2 路时钟信号，一路为表征机器周期的 clock 时钟信号，另一路为 clock 时钟信号 2 倍频的 mem_clk 时钟信号。事实上，在实现中，clock 是 mem_clk 信号的 2 分频信号，分频比倍频在电路上要容易实现的多。信号波形如图 1-15 所示。

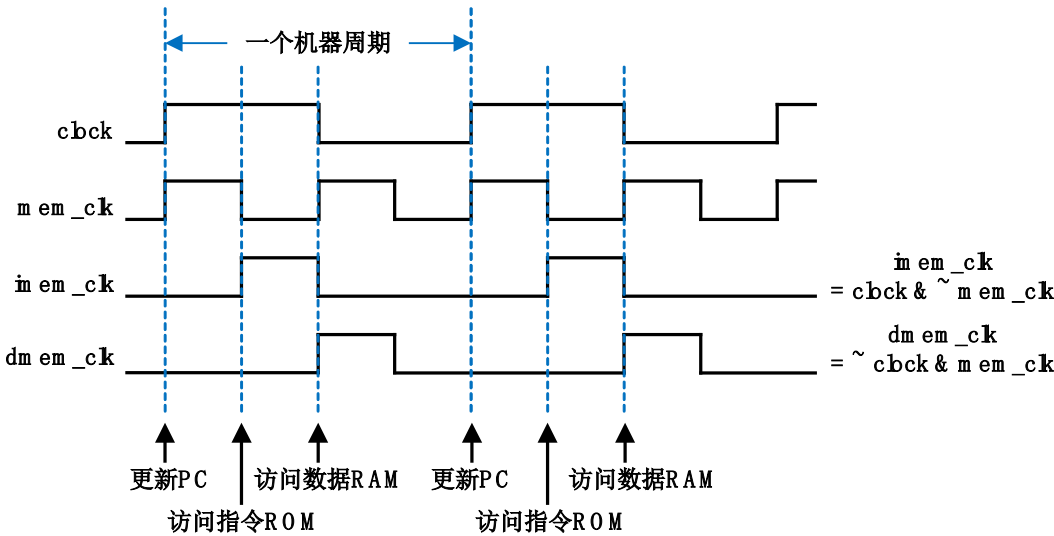


图 2-2-15. 一个机器周期内各执行步骤的执行触发顺序

如图 2-2-15 所示，由输入信号 clock 和 mem_clk 可通过组合逻辑生成 4 个节拍信号，实验中根据时序需要，选取图中标注为 imem_clk、dmem_clk 的两个节拍分别作为指令 ROM 和数据 RAM 的同步时钟。这样，CPU 的工作就在一个机器周期内，有条不紊地顺序执行。各步骤上升沿间的时间间隔，就是留给各阶段组合逻辑的信号传输及延迟时间。

利用机器周期和更高频率（如 8 倍频）信号的组合，可以将一个过程分为更多的节拍（如 8 个节拍），从而安排更多功能部件的协调有序执行。

2.3 实验三：5 段流水 CPU 设计

2.3.1 实验目的：

1. 理解计算机指令流水线的协调工作原理，初步掌握流水线的设计和实现原理。
2. 深刻理解流水线寄存器在流水线实现中所起的重要作用。
3. 理解和掌握流水段的划分、设计原理及其实现方法原理。
4. 掌握运算器、寄存器堆、存储器、控制器在流水工作方式下，有别于实验一的设计和实现方法。
5. 掌握流水方式下，通过 I/O 端口与外部设备进行信息交互的方法。

2.3.2 实验内容：

1. 采用 Verilog 在 quartus II 中实现基本的具有 20 条 MIPS 指令的 5 段流水 CPU 设计。
2. 利用实验提供的标准测试程序代码，完成仿真测试。
3. 采用 I/O 统一编址方式，即将输入输出的 I/O 地址空间，作为数据存取空间的一部分，实现 CPU 与外部设备的输入输出端口设计。实验中可采用高端地址。
4. 利用设计的 I/O 端口，通过 lw 指令，输入 DE2 实验板上的按键等输入设备信息。即将外部设备状态，读到 CPU 内部寄存器。
5. 利用设计的 I/O 端口，通过 sw 指令，输出对 DE2 实验板上的 LED 灯等输出设备的控制信号（或数据信息）。即将对外部设备的控制数据，从 CPU 内部的寄存器，写入到外部设备的相应控制寄存器（或可直接连接至外部设备的控制输入信号）。
6. 利用自己编写的程序代码，在自己设计的 CPU 上，实现对板载输入开关或按键的状态输入，并将判别或处理结果，利用板载 LED 灯或 7 段 LED 数码管显示出来。
7. 例如，将一路 4bit 二进制输入与另一路 4bit 二进制输入相加，利用两组分别 2 个 LED 数码管以 10 进制形式显示“被加数”和“加数”，另外一组 LED 数码管以 10 进制形式显示“和”等。（具体任务形式不做严格规定，同学可自由创意）。
8. 在实验报告中，汇报自己的设计思想和方法；并以汇编语言的形式，提供所设计作品应用功能的程序设计代码，并提供程序主要流程图。

2.3.3 预习内容：

1. 实验前仔细阅读 DE1-SOC User Manual 及相关用户应用数据手册，学习并掌握其板载相关资源的工作原理、连接方式、和应用注意事项。
2. 根据课程所讲 5 段流水 CPU 设计原理，提前设计并仿真实现相关设计代码。

2.3.4 实验器材：

DE1-SOC 实验板套件	1 套
万用表	1 台
示波器	1 台

2.3.5 实验参考：

1. 实验设计原理参考课程教学相关内容及有关实验补充材料。
2. 实验设计的顶层主要代码模块，分别是 5 个流水段的 `module`、以及五级流水段之间的 4 个流水线寄存器 `module`，外加 1 个程序计数器 `module`（也可以当作为最前面一级流水段的输入流水线寄存器），共计 10 个 `module`。
3. 每一个 `module` 的功能，参考课程 5 级流水线的设计原理，即可确定并进行设计。
4. 实验中的顶层文件结构可参考如下所示代码进行设计：（仅用于参考方法，不限定同学自己的设计风格和实现方法）

顶层代码：

```
module pipelined_computer (resetsn, clock, mem_clock, pc, inst, ealu, malu, walu);  
//定义顶层模块 pipelined_computer，作为工程文件的顶层入口，如图 1-1 建立工程时指定。  
    input          resetsn, clock, mem_clock;  
//定义整个计算机 module 和外界交互的输入信号，包括复位信号 resetsn、时钟信号 clock、  
//以及一个和 clock 同频率但反相的 mem_clock 信号。mem_clock 用于指令同步 ROM 和  
//数据同步 RAM 使用，其波形需要有别于实验一。  
//这些信号可以用作仿真验证时的输出观察信号。  
    output [31:0] pc, inst, ealu, malu, walu;  
//模块用于仿真输出的观察信号。缺省为 wire 型。  
    wire [31:0] bpc, jpc, npc, pc4, ins, inst;  
//模块间互联传递数据或控制信息的信号线,均为 32 位宽信号。IF 取指令阶段。  
    wire [31:0] dpc4, da, db, dim;  
//模块间互联传递数据或控制信息的信号线,均为 32 位宽信号。ID 指令译码阶段。  
    wire [31:0] epc4, ea, eb, eimm;  
//模块间互联传递数据或控制信息的信号线,均为 32 位宽信号。EXE 指令运算阶段。  
    wire [31:0] mb, mmo;  
//模块间互联传递数据或控制信息的信号线,均为 32 位宽信号。MEM 访问数据阶段。
```

```

    wire [31:0] wmo,wdi;
//模块间互联传递数据或控制信息的信号线,均为 32 位宽信号。WB 回写寄存器阶段。
    wire [4:0] drn,ern0,ern,mrn,wrn;
//模块间互联,通过流水线寄存器传递结果寄存器号的信号线,寄存器号(32个)为 5bit。
    wire [3:0] daluc,ealuc;
//ID 阶段向 EXE 阶段通过流水线寄存器传递的 aluc 控制信号, 4bit。
    wire [1:0] pcsource;
//CU 模块向 IF 阶段模块传递的 PC 选择信号, 2bit。
    wire wpcir;
//CU 模块发出的控制流水线停顿的控制信号,使 PC 和 IF/ID 流水线寄存器保持不变。
    wire dwreg,dm2reg,dwmem,daluimm,dshift,djal; // id stage
//ID 阶段产生,需往后续流水级传播的信号。
    wire ewreg,em2reg,ewmem,ealuimm,eshift,ejal; // exe stage
//来自于 ID/EXE 流水线寄存器,EXE 阶段使用,或需要往后续流水级传播的信号。
    wire mwreg,mm2reg,mwmem; // mem stage
//来自于 EXE/MEM 流水线寄存器,MEM 阶段使用,或需要往后续流水级传播的信号。
    wire wwreg,wm2reg; // wb stage
//来自于 MEM/WB 流水线寄存器,WB 阶段使用的信号。

    pipepc prog_cnt ( npc,wpcir,clock,resetn,pc );
//程序计数器模块,是最前面一级 IF 流水段的输入。
    pipeif if_stage ( pcsource,pc,bpc,da,jpc,npc,pc4,ins,mem_clock ); // IF stage
//IF 取指令模块,注意其中包含的指令同步 ROM 存储器的同步信号,
//即输入给该模块的 mem_clock 信号,模块内定义为 rom_clk。// 注意 mem_clock。
//实验中可采用系统 clock 的反相信号作为 mem_clock (亦即 rom_clock),
//即留给信号半个节拍的传输时间。
    pipeir inst_reg ( pc4,ins,wpcir,clock,resetn,dpc4,inst ); // IF/ID 流水线寄存器
//IF/ID 流水线寄存器模块,起承接 IF 阶段和 ID 阶段的流水任务。
//在 clock 上升沿时,将 IF 阶段需传递给 ID 阶段的信息,锁存在 IF/ID 流水线寄存器
//中,并呈现在 ID 阶段。
    pipeid id_stage ( mwreg,mrn,ern,ewreg,em2reg,mm2reg,dpc4,inst,
                    wrn,wdi,ealu,malu,mmo,wwreg,clock,resetn,
                    bpc,jpc,pcsource,wpcir,dwreg,dm2reg,dwmem,daluc,
                    daluimm,da,db,dimm,drn,dshift,djal ); // ID stage
//ID 指令译码模块。注意其中包含控制器 CU、寄存器堆、及多个多路器等。
//其中的寄存器堆,会在系统 clock 的下沿进行寄存器写入,也就是给信号从 WB 阶段
//传输过来留有半个 clock 的延迟时间,亦即确保信号稳定。
//该阶段 CU 产生的、要传播到流水线后级的信号较多。
    pipedereg de_reg ( dwreg,dm2reg,dwmem,daluc,daluimm,da,db,dimm,drn,dshift,
                    djal,dpc4,clock,resetn,ewreg,em2reg,ewmem,ealuc,ealuimm,
                    ea,eb,eimm,ern0,eshift,ejal,epc4 ); // ID/EXE 流水线寄存器
//ID/EXE 流水线寄存器模块,起承接 ID 阶段和 EXE 阶段的流水任务。
//在 clock 上升沿时,将 ID 阶段需传递给 EXE 阶段的信息,锁存在 ID/EXE 流水线
//寄存器中,并呈现在 EXE 阶段。

    pipeexe exe_stage ( ealuc,ealuimm,ea,eb,eimm,eshift,ern0,epc4,ejal,ern,ealu ); // EXE stage
//EXE 运算模块。其中包含 ALU 及多个多路器等。
    pipeemreg em_reg ( ewreg,em2reg,ewmem,ealu,eb,ern,clock,resetn,
                    mwreg,mm2reg,mwmem,malu,mb,mrn ); // EXE/MEM 流水线寄存器

```

```

//EXE/MEM 流水线寄存器模块，起承接 EXE 阶段和 MEM 阶段的流水任务。
//在 clock 上升沿时，将 EXE 阶段需传递给 MEM 阶段的信息，锁存在 EXE/MEM
//流水线寄存器中，并呈现在 MEM 阶段。
    pipemem mem_stage ( mwmem,malu,mb,clock,mem_clock,mmo );           // MEM stage
//MEM 数据存取模块。其中包含对数据同步 RAM 的读写访问。// 注意 mem_clock。
//输入给该同步 RAM 的 mem_clock 信号，模块内定义为 ram_clk。
//实验中可采用系统 clock 的反相信号作为 mem_clock 信号（亦即 ram_clk），
//即留给信号半个节拍的传输时间，然后在 mem_clock 上升沿时，读输出、或写输入。

    pipemwreg mw_reg ( mwreg,mm2reg,mmo,malu,mrn,clock,resetn,
                      wwreg,wm2reg,wmo,walu,wrn);           // MEM/WB 流水线寄存器
//MEM/WB 流水线寄存器模块，起承接 MEM 阶段和 WB 阶段的流水任务。
//在 clock 上升沿时，将 MEM 阶段需传递给 WB 阶段的信息，锁存在 MEM/WB
//流水线寄存器中，并呈现在 WB 阶段。
    mux2x32 wb_stage ( walu,wmo,wm2reg,wdi );           // WB stage
//WB 写回阶段模块。事实上，从设计原理图上可以看出，该阶段的逻辑功能部件只
//包含一个多路器，所以可以仅用一个多路器的实例即可实现该部分。
//当然，如果专门写一个完整的模块也是很好的。
endmodule

```

5. 这样，以上的 10 个模块（流水段模块+各段之间的流水线寄存器模块）在计算机系统 clock 的同步协调触发下，就会将指令在 5 段流水线中顺序流水执行。
6. 实验二给同学提供设计顶层代码，实验一的大部分代码可修改后使用，设计完成后，尽快掌握和精通流水线的设计原理和实现方法。并能触类旁通地体会和理解对其它算法逻辑进行流水处理的设计和实现方法。
7. 实验代码的工程文件 pipelined_computer.qpf 所包含的相关源代码文件如图 2-3-1 所示，图中右侧为仿真测试输入 pipelined_computer_test_wave_01.vwf 的输入波形。注意 3 个输入信号 resetn、clock、mem_clock 的时序关系。为观察直观，其它的输出观测信号 pc、ins、inst、ealu、malu、walu 设置为“uninitialized”状态。
8. 图 2-3-2 为采用测试输入 sc_computer_test_wave_01.vwf 运行实验标准测试程序后，流水线的仿真运行输出。
9. 同学需独立自主地采用统一编址的 I/O 端口扩展方法，设计自己针对 DE1-SOC 板载外围设备进行控制接口的控制寄存器，即与外部设备进行互联的端口控制寄存器。**提示：可在 pipemem.v 模块文件中，通过对输入的 lw 或 sw 指令的地址的判断，实现对数据 RAM 和 IO 控制寄存器组的区分、和分别控制。**
10. 利用上述实现的 I/O 功能，同学即可利用自己编写的应用程序，在自己设计的 5 级流水 CPU 上完成实验内容第 6、7 项所要求的内容。

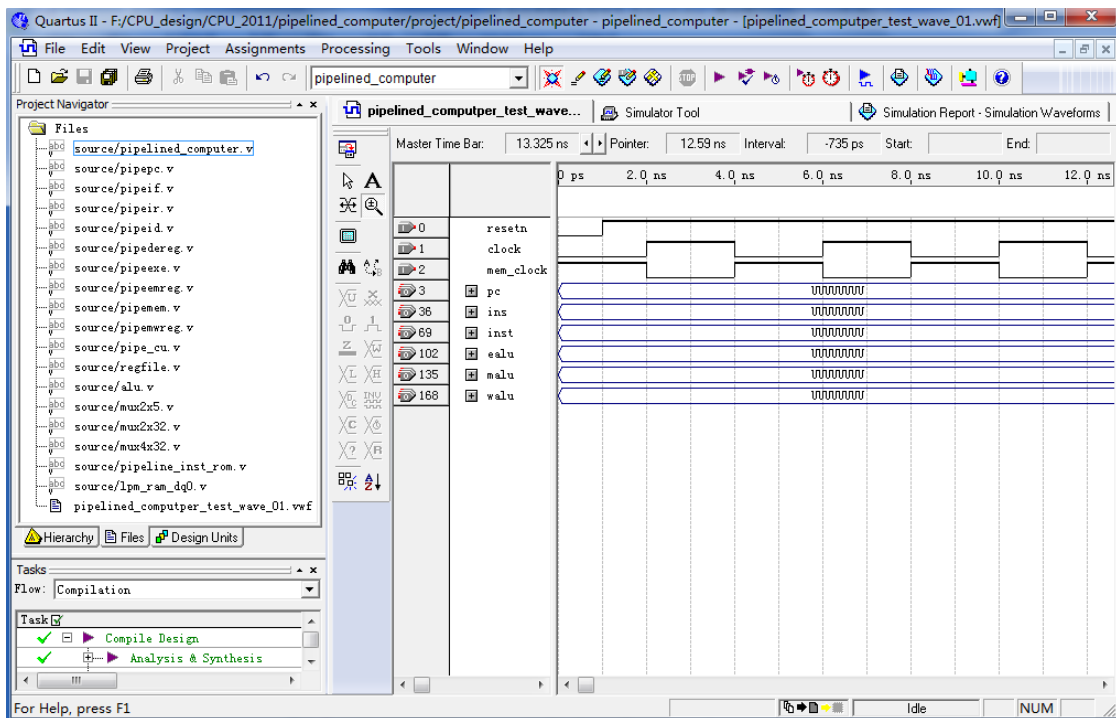


图 2-3-1. 工程文件所包含的相关文件及仿真运行测试输入文件波形

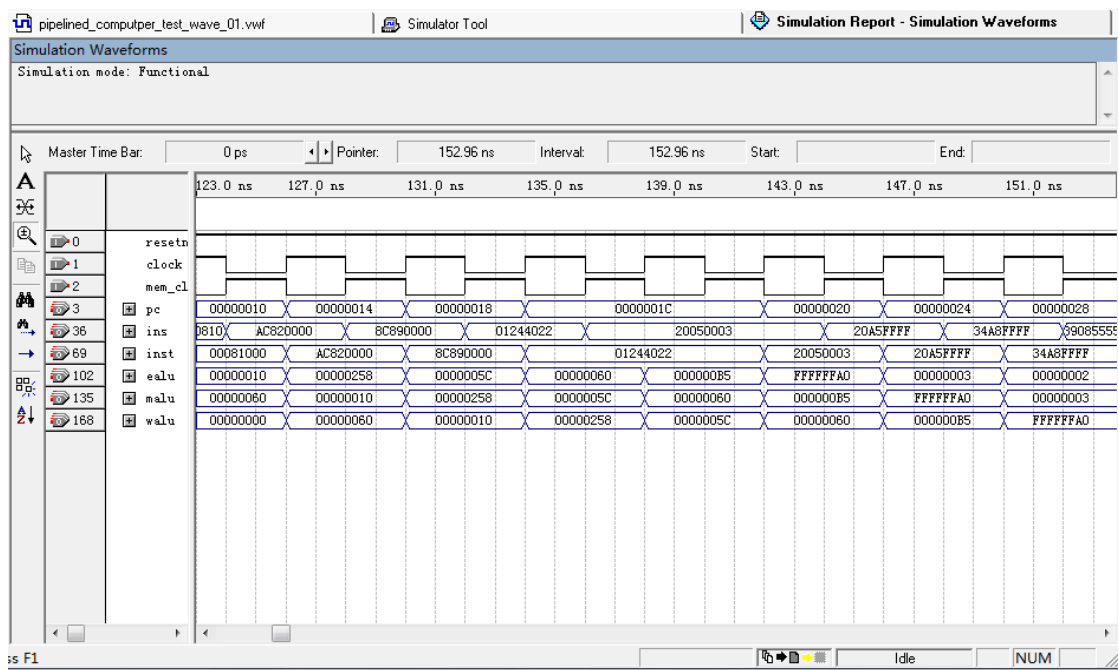


图 2-3-2. 标准测试程序的仿真运行输出波形

2.4 实验四：外部 I/O 及接口扩展实验

2.4.1 实验目的：

1. 理解计算机的外部 I/O 扩展方法原理。
2. 理解外部 I/O 总线的设计原理。
3. 理解 I/O 地址空间的设计方法，掌握 I/O 端口的设计方法。
4. 会通过设计 I/O 端口与外部设备进行信息交互。

2.4.2 实验内容：

1. 在实验二实现的类 MIPS 指令系统的 5 段流水 CPU 设计基础上，采用 I/O 地址和主存统一编址的方式，即将输入输出的 I/O 地址空间，作为主存数据存取空间的一部分，实现 CPU 与外部设备的输入输出端口设计。实验中可采用高端地址。
2. 利用设计的 I/O 端口，通过 lw 指令，输入 DE1-SOC 实验板上的按键等输入设备信息。即将外部设备的状态或数据输入，读入到 CPU 内部寄存器。
3. 利用设计的 I/O 端口，通过 sw 指令，输出对 DE1-SOC 实验板上的 LED 灯等输出设备的控制信号（或数据信息）。即将对外部设备的控制数据，从 CPU 内部的寄存器，写入到外部设备的相应控制寄存器（该控制寄存器输出，可直接连接输出至外部设备，作为对外部设备的控制输入信号）。
4. 能够利用自己编写的程序代码，在自己设计的 CPU 上，利用 DE1-SOC 板载资源，设计实现一个简单的计算器。
5. 利用板载的 VGA 接口资源，设计显示缓存 Framebuffer 存储器、及相应的显示控制逻辑，实现计算结果或其它内容在 VGA 接口显示器上的显示。（扩展内容）
6. 也可以基于板载资源，进行其它多种接口控制器（如 RS232 串行接口、USB 接口、以太网接口等）通过 I/O 总线的互联控制。（扩展内容）
7. 在实验报告中，汇报自己的设计思想和方法；并以汇编语言的形式，提供以上应用功能的程序设计代码，并提供程序主要流程图。

2.4.3 预习内容：

1. 仔细阅读 DE1-SOC User Manual 相关章节、及其它相关器件应用数据手册，学习并掌握其板载相关资源的工作原理、连接方式、和应用注意事项。

2. 重点掌握按键输入、LED 显示输出部分的工作原理和连接方式。
3. 进行 VGA 显示扩展,需重点掌握 DE1-SOC User Manual 中有关板载 VGA 控制器原理和控制电路的部分章节,或其它相关接口控制器章节等。

2.4.4 实验器材:

Altera-DE1-SOC 实验板套件	1 套
数字万用表	1 台
示波器	1 台

2.4.5 实验参考:

2.4.5.1 I/O 端口扩展原理

计算机的 I/O 端口,即和外部设备进行信号交互的 Input/Output 操作地址,有两种典型的设计方法:

一种是采用专门的输入输出指令,如 x86 的 In 指令和 Out 指令,这种方式有自己独立于内存访问地址空间的单独的 I/O 地址空间。

另一种是将 I/O 地址空间设计为内存地址空间的一部分,即 I/O 地址和内存地址统一编址,CPU 在访问内存 MEM 和 I/O 端口时,采用相同的指令,区分是访问 MEM 还是 I/O 端口,由具体的硬件电路通过对它们不同地址(各有自己的范围)的识别进行区分。

采用上述第二种将 I/O 地址和内存地址统一编址的方法,对于课程实验中 RISC 风格的类 MIPS 指令系统,对 I/O 端口的访问(读:输入来自于外部信号;写:输出到外部的信号),采用同访问 MEM 一样的 Lw 指令和 Sw 指令,由硬件通过对地址的识别区分,实现对 MEM 访问和 I/O 访问的区分。

如实验一指导书中的提示:**“提示:可在 sc_datamem.v 模块文件中,通过对输入的 lw 或 sw 指令的地址的判断,实现对数据 RAM 和 IO 控制寄存器组的区分、和分别控制。”** sc_datamem.v 的参考代码、及其实现原理图如下图 2-4-1 及随后的源码所示。

对于实验三的 5 级流水线 CPU,可参考以上 sc_datamem.v 模块示例代码的设计方法,进行相应的输入和输出信号等的对应更改及相关调整后,采用类似原理设计 pipemem.v 模块代码。

(1) I/O 端口扩展原理图解

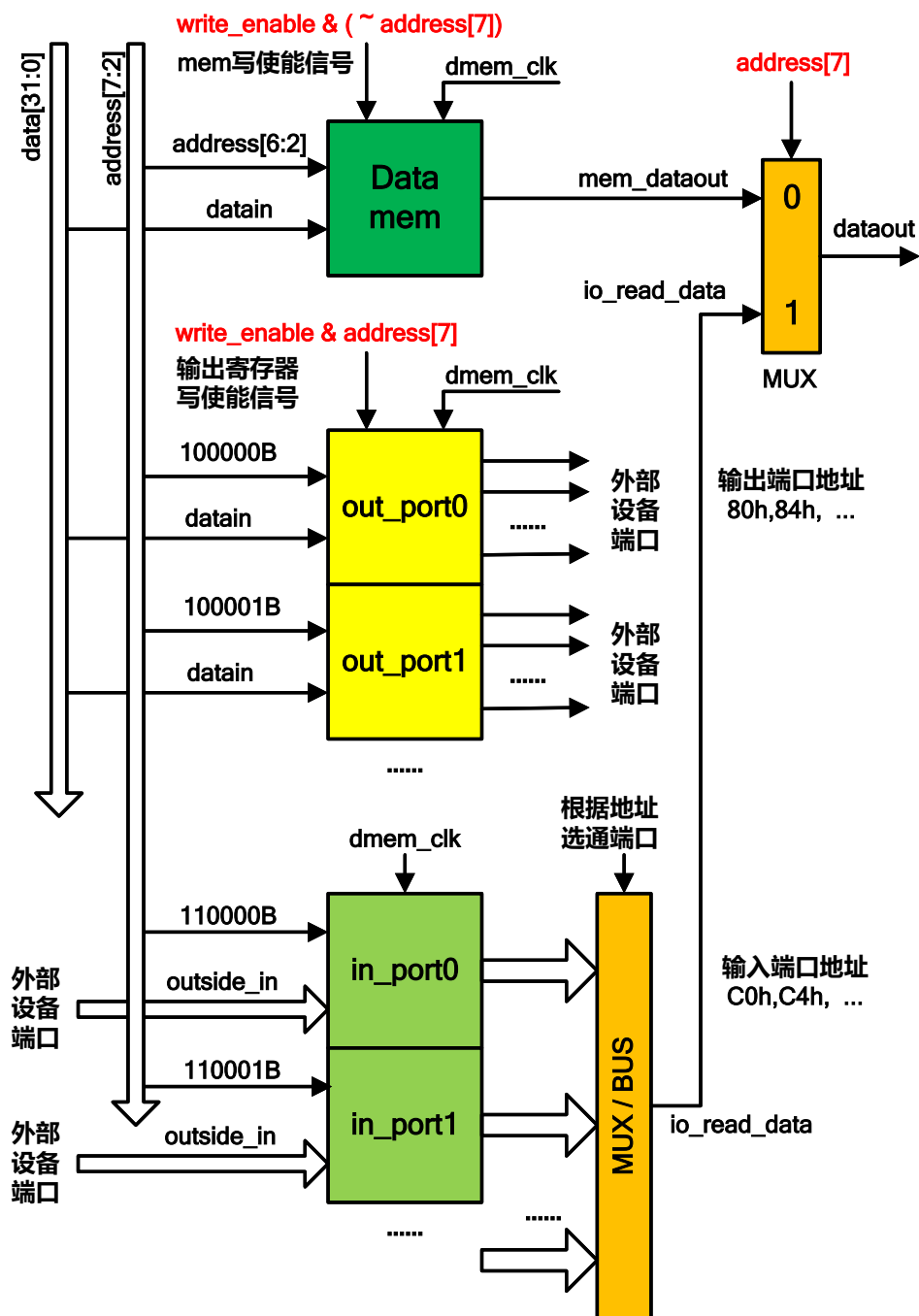


图 2-4-1. 统一编址方式 I/O 端口扩展原理

(2) sc_datamem.v 模块参考代码：

```
module sc_datamem (addr,datain,dataout,we,clock,mem_clk,dmem_clk,clrn,
                  out_port0,out_port1,in_port0,in_port1,mem_dataout,io_read_data);

    input  [31:0]    addr;
    input  [31:0]    datain;
    input          we, clock,mem_clk,clrn;
    input  [31:0]    in_port0,in_port1;

    output [31:0]    dataout;
    output          dmem_clk;
    output [31:0]    out_port0,out_port1;
    output [31:0]    mem_dataout;
    output [31:0]    io_read_data;

    wire [31:0]      dataout;
    wire             dmem_clk;

    wire             write_enable;
    wire             wrire_datamem_enable;
    wire [31:0]      mem_dataout;

    assign           dmem_clk = mem_clk & ( ~ clock) ; //注意
    assign           write_enable = we & ~clock; //注意
    assign           write_datamem_enable = write_enable & ( ~ addr[7]); //注意
    assign           write_io_output_reg_enable = write_enable & ( addr[7]); //注意

    mux2x32 mem_io_dataout_mux(mem_dataout,io_read_data,addr[7],dataout);
// module mux2x32 (a0,a1,s,y);

// when address[7]=0, means the access is to the datamem.
// that is, the address space of datamem is from 000000 to 011111 word(4 bytes)

    lpm_ram_dq dram dram(addr[6:2],dmem_clk,datain,write_datamem_enable,mem_dataout );

// when address[7]=1, means the access is to the I/O space.
// that is, the address space of I/O is from 100000 to 111111 word(4 bytes)

    io_output_reg io_output_regx2 (addr,datain,write_io_output_reg_enable,
                                   dmem_clk,clrn,out_port0,out_port1);
// module io_output_reg (addr,datain,write_io_enable,io_clk,clrn,out_port0,out_port1 );

    io_input_reg io_input_regx2(addr,dmem_clk,io_read_data,in_port0,in_port1);
// module io_input_reg (addr,io_clk,io_read_data,in_port0,in_port1);

endmodule
```


(3) io_output_reg.v 模块参考代码:

```
module io_output_reg (addr,datain,write_io_enable,io_clk,clrn,out_port0,out_port1 );

    input  [31:0]  addr,datain;
    input                write_io_enable,io_clk;
    input                clrn;
                        //reset signal. if necessary,can use this signal to reset the output to 0.
    output [31:0]  out_port0,out_port1;

    reg    [31:0]  out_port0;    // output port0
    reg    [31:0]  out_port1;    // output port1

    always @(posedge io_clk or negedge clrn)
    begin
        if (clrn == 0)
            begin
                // reset
                out_port0 <=0;
                out_port1 <=0;    // reset all the output port to 0.
            end
        else
            begin
                if (write_io_enable == 1)
                    case (addr[7:2])
                        6'b100000: out_port0 <= datain;    // 80h
                        6'b100001: out_port1 <= datain;    // 84h
                        // more ports, 可根据需要设计更多的输出端口。
                    endcase
            end
        end
    end

endmodule
```

(4) io_input_reg.v 模块参考代码:

```
module io_input_reg (addr,io_clk,io_read_data,in_port0,in_port1);

    input  [31:0]  addr;
    input                io_clk;
    input  [31:0]  in_port0,in_port1;
    output [31:0]  io_read_data;

    reg    [31:0]  in_reg0;    // input port0
    reg    [31:0]  in_reg1;    // input port1

    io_input_mux io_input_mux2x32(in_reg0,in_reg1,addr[7:2],io_read_data);

    always @(posedge io_clk)
```

```

begin
    in_reg0 <= in_port0;    // 输入端口在 io_clk 上升沿时进行数据锁存
    in_reg1 <= in_port1;    // 输入端口在 io_clk 上升沿时进行数据锁存

// more ports, 可根据需要设计更多的输入端口。

end
endmodule

module io_input_mux(a0,a1,sel_addr,y);
    input    [31:0]  a0,a1;
    input    [ 5:0]  sel_addr;
    output   [31:0]  y;
    reg      [31:0]  y;

    always @ *
    case (sel_addr)

        6'b110000: y = a0;
        6'b110001: y = a1;

// more ports, 可根据需要设计更多的端口。

    endcase
endmodule

```

(5) 针对 I/O 端口的一段测试例程代码：(sc_instmem_02.mif)

```

DEPTH = 64; % Memory depth and width are required %
WIDTH = 32; % Enter a decimal number %
ADDRESS_RADIX = HEX; % Address and value radices are optional %
DATA_RADIX = HEX; % Enter BIN, DEC, HEX, or OCT; unless %
% otherwise specified, radices = HEX %
CONTENT
BEGIN

0 : 20010050; % (00)  main:  addi $1, $0, 01010000b      # address 50h      %
1 : 20020080; % (04)          addi $2, $0, 10000000b      # address 80h      %
2 : 20030084; % (08)          addi $3, $0, 10000100b      # address 84h      %
3 : 200400c0; % (0c)          addi $4, $0, 11000000b      # address c0h      %
4 : 200500c4; % (10)          addi $5, $0, 11000100b      # address c4h      %
5 : 20060055; % (14)          addi $6, $0, 01010101b      # data    55h      %
6 : 200700aa; % (18)          addi $7, $0, 10101010b      # data    aah      %
7 : 200a0000; % (1c)  loop:  addi $10,$0, 0                # r10 = 0          %
8 : 200b0000; % (20)          addi $11,$0, 0                # r11 = 0          %
9 : 8c2c0000; % (24)          lw   $12,0($1)                # load data[50h]   %
A : ac260000; % (28)          sw   $6, 0($1)                # store 55h to [50] %
B : 8c2c0000; % (2c)          lw   $12,0($1)                # load data[50h]   %
C : ac270000; % (30)          sw   $7, 0($1)                # store aah to [50] %
D : ac460000; % (34)          sw   $6, 0($2)                # output 55h to [80h] %
E : ac660000; % (38)          sw   $6, 0($3)                # output 55h to [84h] %
F : ac470000; % (3c)          sw   $7, 0($2)                # output aah to [80h] %
10 : ac670000; % (40)          sw   $7, 0($3)                # output aah to [84h] %
11 : 8c8a0000; % (44)          lw   $10,0($4)                # input data from [c0h] %
12 : 8cab0000; % (48)          lw   $11,0($5)                # input data from [c4h] %
13 : 8c8a0000; % (4c)          lw   $10,0($4)                # input data from [c0h]* %
14 : 8cab0000; % (50)          lw   $11,0($5)                # input data from [c4h]* %
15 : 214a0000; % (54)          addi $10,$10,0                # just check $10    %
16 : 216b0000; % (58)          addi $11,$11,0                # just check $11    %
17 : 08000007; % (5c)          j     loop                    # loop              %

END ;

```

(6) sc_instmem_02.mif 例程代码的仿真执行结果：

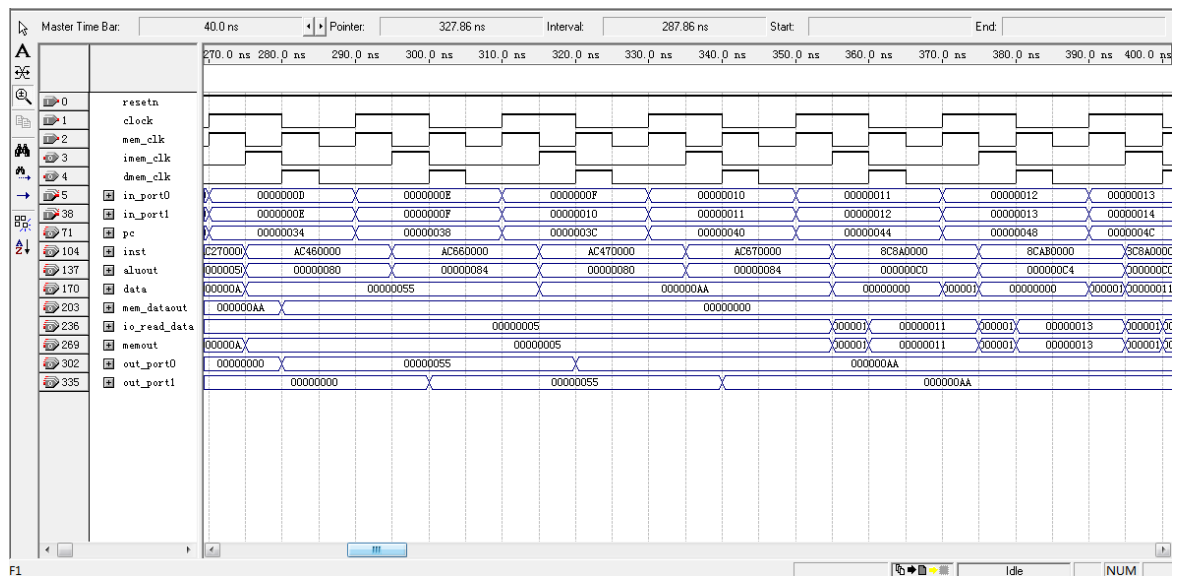
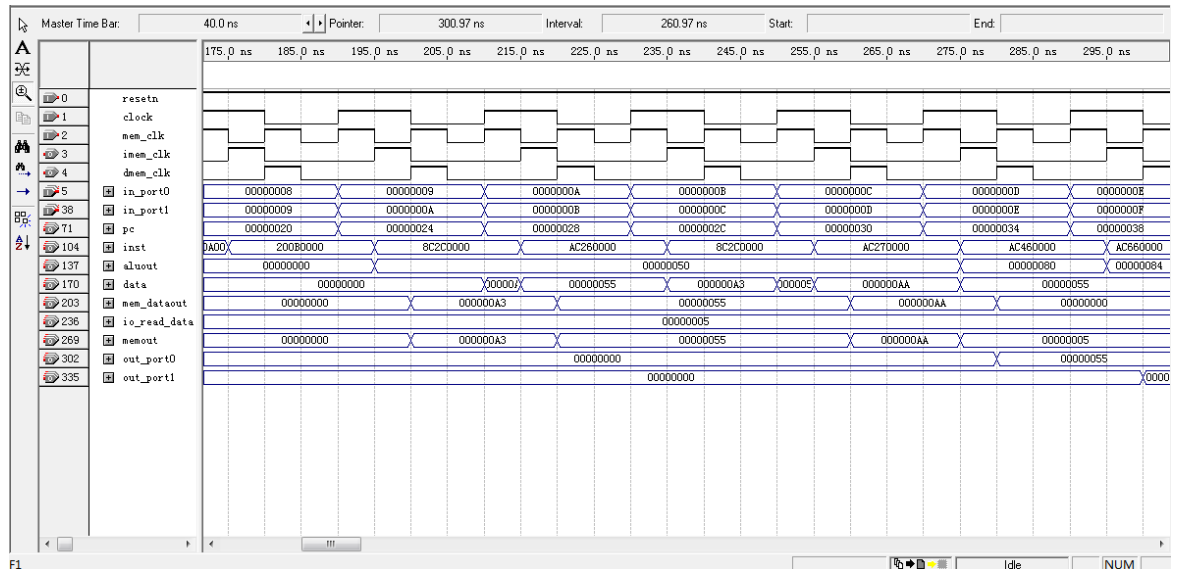


图 3-2. 例程代码的仿真执行结果

2.4.5.2 I/O 端口设计和使用说明

外部设备，如显示器、打印机、键盘、鼠标、以太网等，都是通过相应的各种“接口”和计算机进行连接的，如 VGA 接口、USB 接口、RS232 串行接口、以太网接口、modem 调制解调器等。“接口”是设备和计算机 I/O 总线之间的连接和控制部件，本质上是挂在 I/O 总线上的一种通信控制器。

接口的工作原理示意图如图 2-4-3 所示。CPU 通过 I/O 总线和接口控制器内的若干寄存器进行数据交换，这些寄存器有两类，相对于 CPU，一种是作为 CPU 的输入端口，一种是作为 CPU 的输出端口。这些寄存器按照其功能，可以分为数据寄存器和控制寄存器，数据寄存器里缓冲的是接口通讯线路上真正传输的数据，控制寄存器中放置的是对“通讯电路控制逻辑”的控制数据（或称控制信号），起配置通讯线路数据传输协议（或电信号格式）的作用。

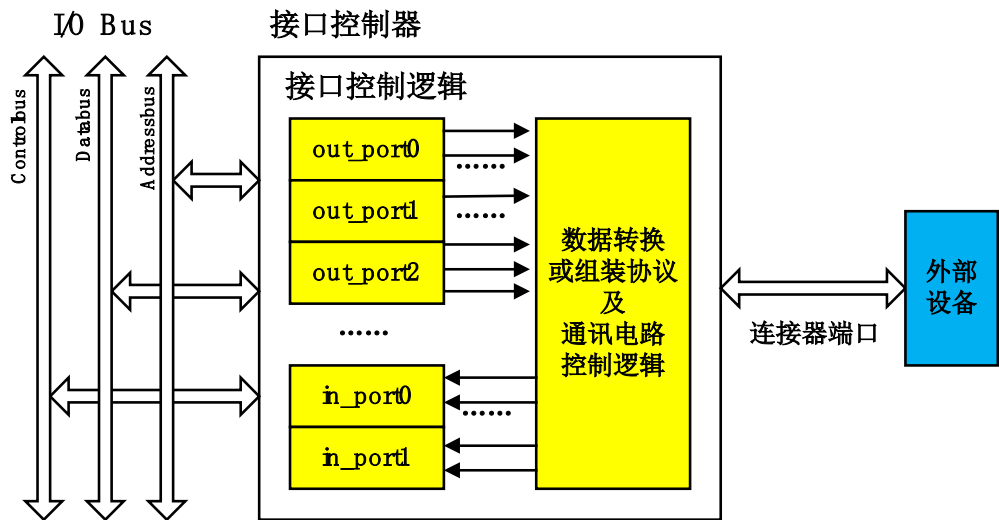


图 2-4-3. 接口控制器原理

接口对于 CPU 可见的是一组输入输出端口寄存器，CPU 也是通过对这一组端口寄存器的“读”和“写”，达到控制“接口”、以及和“接口”交互数据的功能，亦即达到和外设进行交互的功能。

实验中，可通过设计相应的 I/O 端口和相应的接口控制逻辑，达到和多种外设利用其“接口”进行交互的目的。具体设计参考相应的 DE2 板载接口部件说明和设计文档。

附件：

1. Altera-DE1-SOC User Manual。
2. Altera-DE1-SOC 实验板随附资料。最新资料（设计资源、随附光盘资料等）下载网址：

英文：

<http://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&CategoryNo=165&No=836&PartNo=1>

中文：

<http://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=China&CategoryNo=180&No=870>。

3. 实验二参考例程代码。

参考网站：<http://www.altera.com> 或 <http://www.altera.com.cn>