MIPS 流水线大作业实验报告

何源 电子工程系 2018011182

一、设计方案

1. 流水线处理器设计方案

该部分设计方案见附件《MIPS 流水线架构文档》。

2. 外设设计方案

1) BCD7

BCD7 段数码管为该处理器的显示外设。在处理器中内存地址 0x40000010 对应该七段数码管。

中断软件程序中，对 Systick 取低 4 位，通过汇编进行查表然后将对应的 BCD7_32bit 写入到七段数码管对应的内存地址中。所查表如下：

Systick[3:0]	Hex	BCD7	BCD7_32bit
0000	0	11111100	32'h000001fc
0001	1	01100000	32'h00000160
0010	2	11011010	32'h000001da
0011	3	11110010	32'h000001f2
0100	4	01100110	32'h00000166
0101	5	10110110	32'h000001b6
0110	6	10111110	32'h000001be
0111	7	11100000	32'h000001e0
1000	8	11111110	32'h000001fe
1001	9	11110110	32'h000001f6
1010	A	11101111	32'h000001ef
1011	B	11111111	32'h000001ff
1100	C	10011101	32'h0000019d
1101	D	11111101	32'h000001fd
1110	E	10011111	32'h0000019f
1111	F	10001111	32'h0000018f

2) 定时器

定时器内置在数据存储器中，通过汇编访问对应地址进行控制。本次实验中对该定时器的功能做了一些修改，具体如下：

地址	功能	描述
0x40000000	定时器 TH	每当 TL == TH 时，将 TL 的值置为 0.
0x40000004	定时器 TL	定时器计数器，TL 值随时钟递增
0x40000008	定时器控制 TCON	0bit: 定时器使能控制， 1-enable, 0-disable 1bit: 定时器中断控制，1-enable, 0-disable 2bit: 定时器中断状态，当 TL == 1 时，将 2bit 置零

3. 软件设计方案

本次实验的软件由汇编语言编写，大体流程如下：

1) 启动定时器
2) 记录起始时刻 Systick
3) 快速排序
4) 读取结束时刻 Systick，得到排序运行时间
5) 定时器 Timeout，触发中断
6) 执行中断程序，将 Systick 第四位译为 BCD 码，存入七段数码管对应内存中。

由于设计 hazard 检测在软件阶段进行，因此在编写汇编程序时应当遵守规范。关于 hazard 检测详细内容见《MIPS 流水线架构文档》第六部分。

二、关键代码

1. Register

Register 实现先写后读的方法：Read 采用组合逻辑，Write 采用时序逻辑。

```
module Register(
    input reset,
    input clk,
    input [4:0] Write_Addr,
    input [4:0] Read_Addr_1,
    input [4:0] Read_Addr_2,
    input [31:0] Write_Data,
    output [31:0] Read_Data_1,
    output [31:0] Read_Data_2,
    input RegWrite,
    input [31:0] Register_26,
    input PCBackup
);

    reg [31:0] Register_Data [31:1];

    assign Read_Data_1 = (Read_Addr_1 == 5'b00000) ? 32'h00000000:Register_Data[Read_Addr_1];
    assign Read_Data_2 = (Read_Addr_2 == 5'b00000) ? 32'h00000000:Register_Data[Read_Addr_2];

    integer i;
    always @(posedge reset or posedge clk) begin
        if (reset) begin
            for (i = 1; i < 32; i = i+1)
                if (i == 29)
                    Register_Data[i] <= 32'h00000800;
                else
                    Register_Data[i] <= 32'h00000000;
        end else begin
            if (RegWrite && (Write_Addr != 5'b00000))
                Register_Data[Write_Addr] <= Write_Data;
```

```
        if (PCBackup)
            Register_Data[26] <= Register_26;
    end
end
endmodule : Register
```

2. Data Memory

Data Memory 也使用和 Register 相同的先写后读方法。此外，Data Memory 中还有对于外设 Timer 的实现。128 个 32 位随机数使用 Python 生成后复制粘贴于 Verilog 代码中，在 Data Memory 初始化时存入其中。

```
module Data_Memory(
    input reset,
    input clk,
    input [31:0] Read_Addr,
    input [31:0] Write_Addr,
    input [31:0] Write_Data,
    output [31:0] Read_Data,
    input MemRead,
    input MemWrite,
    output reg [31:0] LEDs,
    output reg [31:0] BCD7,
    output reg [31:0] SysTick,
    output Irq
);

    parameter RAM_SIZE=512;
    parameter RAM_SIZE_BIT=9;

    reg [31:0] RAM_data [RAM_SIZE-1:0];
    reg [31:0] Timer [2:0];

    assign Read_Data =
        MemRead ? (Read_Addr < 32'h000007ff) ? RAM_data[Read_Addr[RAM_SIZE_BIT+1:2]]:
            (Read_Addr == 32'h40000000) ? Timer[0]:
                (Read_Addr == 32'h40000004) ? Timer[1]:
                    (Read_Addr == 32'h40000008) ? Timer[2]:
                        (Read_Addr == 32'h4000000c) ? LEDs:
                            (Read_Addr == 32'h40000010) ? BCD7:
                                (Read_Addr == 32'h40000014) ? SysTick:
                                    32'h00000000:
            32'h00000000;
    assign Irq = Timer[2][2];
```

```
integer i;
always @(posedge reset or posedge clk) begin
    if (reset) begin
            RAM_data[0] <= 32'h8351D612;
            RAM_data[1] <= 32'h836A9DF7;
            RAM_data[2] <= 32'h069897D9;
            //••• 此处省略若干行
            RAM_data[126] <= 32'hEAC15BCE;
            RAM_data[127] <= 32'hDA1E623C;
        for (i = 128; i < RAM_SIZE; i = i+1)
           RAM_data[i] <= 32'h00000000;
        LEDs <= 32'h00000000;
        BCD7 <= 32'h00000000;
        SysTick <= 32'h00000000;
        for (i = 0; i < 3; i = i + 1)
            Timer[i] <= 32'h00000000;
    end
    else begin
        if (MemWrite) begin
            if (Write_Addr < 32'h000007ff)
                RAM_data[Write_Addr[RAM_SIZE_BIT+1:2]] <= Write_Data;
            else if (Write_Addr == 32'h4000000c)
                LEDs <= Write_Data;
            else if (Write_Addr == 32'h40000010)
                BCD7 <= Write_Data;
        end
        if (MemWrite && Write_Addr == 32'h40000000)
            Timer[0] <= Write_Data;
        else if (MemWrite && Write_Addr == 32'h40000004)
            Timer[1] <= Write_Data;
        else if (MemWrite && Write_Addr == 32'h40000008)
            Timer[2] <= Write_Data;
        else begin
            if (Timer[2][0]) begin
                if (Timer[1] == Timer[0]) begin
                    Timer[1] <= 32'h00000000;
                    if (Timer[2][1])
                        Timer[2][2] <= 1'b1;
                end
                else
                    Timer[1] <= Timer[1] + 32'h00000001;
                    if (Timer[1] == 32'h00000001)
                        Timer[2][2] <= 1'b0;
            end
```

```
            end

            if (MemWrite && Write_Addr == 32'h40000014)
                  SysTick <= Write_Data;
            else
               SysTick <= SysTick + 32'h00000001;
        end
    end
endmodule
```

三、综合情况：

资源占用：

Name	Slice LUTs (10400)	Slice Registers (20800)	F7 Muxes (16300)	F8 Muxes (8150)	Bonded IOB (210)	BUFGCTRL (32)
CPU	6805	17942	2478	1090	14	1
execution (EX)	484	167	0	0	0	0
forwarding (Forward)	18	0	0	0	0	0
instruction_decode (ID)	845	1055	257	0	0	0
instruction_fetch (IF)	249	32	45	2	0	0
memory (MEM)	5209	16650	2176	1088	0	0
write_back (WB)	0	38	0	0	0	0

时序性能：

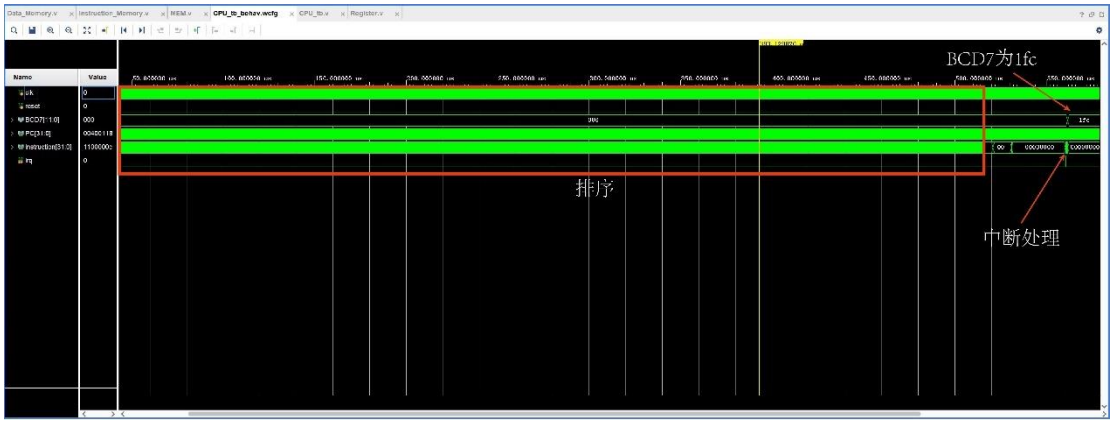
Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	-3.446 ns	Worst Hold Slack (WHS):	0.142 ns	Worst Pulse Width Slack (WPWS):	4.500 ns
Total Negative Slack (TNS):	-48006.812 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	16608	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	35477	Total Number of Endpoints:	35477	Total Number of Endpoints:	17943

Timing constraints are not met.

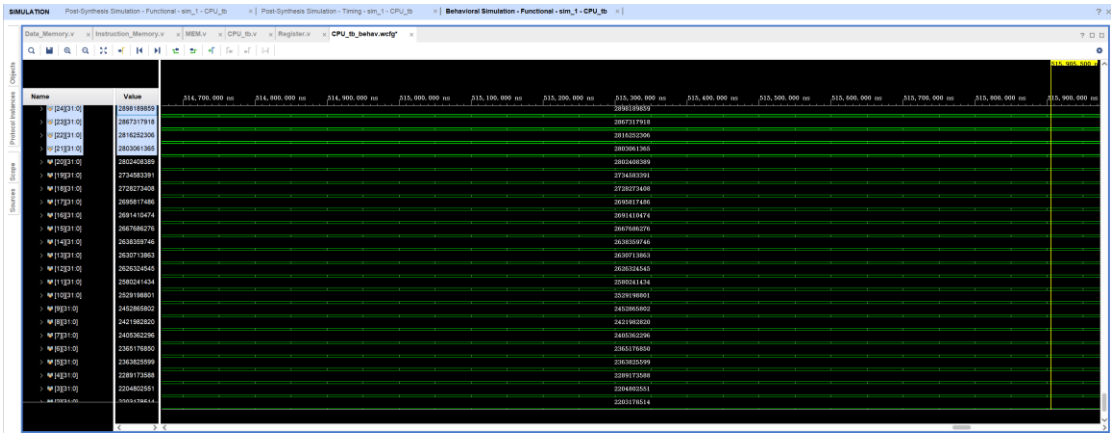
综合采用的时钟周期为 10ns，此处报告最坏的 setup slack 为-3.446ns，因此该处理器能工作的最短时钟周期为 13.446ns，最高时钟频率为 74.4MHz.

四、后仿真情况和分析



如图，后仿真输出的 BCD7 为 1fc，查前表可知排序用时最低位为 0.

由于后仿中 Data_Memory 中的内部元件被综合成了若干零散的寄存器，不便于查看排序结果，故以前仿排序结果代替：



检查 Data Memory 中的数据，发现排序结果正确。
排序共用了 25760 个周期。Mars 仿真共用了 20344 周期。CPI 约为 1.266.
排序所用时间最少为 0.346ms = 25760 * 13.446ns
前一学期完成的单周期 MIPS 处理器最短周期为 30.606ns，所用时间最少为：0.6226ms = 20344 * 30.606ns.
性能提升了接近一半。

五、经验体会

经验体会分为整理的经验体会和局部经验知识的收获。

先说整体的经验体会。这次 MIPS 流水线处理器的开发工作是 9 月 15 日才开始的，于 9 月 20 日结束。在这 6 天中，我的开发时间分配是 3 天半的时间设计架构，1 天的时间完成代码，最后一天的时间完成仿真。根据以前软件开发的经验，架构设计永远是开发中最重要的事情。要在动代码之前将所有的代码怎么写全都想清楚，否则最后的代码写出来往往是一锅粥。因此我花了大量时间研究流水线的课件，绘制了 Architecture 和 Modules 两张图，还有《MIPS 流水线架构文档》，整个庞大的系统在眼前愈发清晰明了。每一个元件的输入输出端口的定义，关键元件的逻辑，以及架构中应当特别注意之处，也都在各个文件中有所表达。这样做虽然不能穷尽所有的细节，但是为之后的开发提供了极大的便利。一是因为有了仔细的考虑后再动代码，只会出现一些粗心导致的错误，而几乎不会出现要大改的设计错误。二是因为即便出现了一些设计中考虑不周到的地方，所有的一切也都在框架中了，修改之处一目了然，不会出现改的不彻底的情况。三是因为架构清晰的代码便于软件进行综合，不至于因为 Vivado 的“企业级理解”而出现的各种难以解释的错误。我这次的开发和仿真极为顺利，在排完粗心导致的一些错误之后，后仿一遍就过，和我本次在架构设计上下的功夫是分不开的。

再说一些细节上的经验。

1) 在 always 中尽量使用非阻塞赋值，且应保证一个变量只赋值一次；
2) 在一个时钟周期内要跑完的电路不能出现时序逻辑电路的串接；
3) 架构图设计完之后，在对着结构图写代码的过程中，应该将架构图打印出来放在手边，及时动笔标画；
4) 应当找一个靠谱的 IDE，Vivado 缺少一些好用的开发功能，会降低开发效率，增加出低级错误的概率；

六、文件清单

/	codes	assembly	Code.asm	汇编代码
		constains	CPU.xdc	约束文件
		design_source	ALU.v	各设计图同名原件的 Verilog 描述性文件
			ALU_Ctrl.v	
			Compare.v	
			CPU.v	
			Ctrl.v	
			Data_Memory.v	
			EX.v	
			EX_MEM.v	
			Exception_Unit.v	
			Forward.v	
			ID.v	
			ID_EX.v	
			IF.v	
			IF_ID.v	
			Instruction_Memory.v	
			MEM.v	
			PC.v	
			Register.v	
			Signal_Extend.v	
			WB.v	
		simulation	CPU_tb.v	仿真文件
	design	Architecture.ai		架构图（矢量图）
		Architecture.jpg		架构图（位图）
		MIPS 流水线架构文档.pdf		设计文档
		Module.ai		模块图（矢量图）
		Module.jpg		模块图（位图）
	MIPS 流水线大作业实验报告.pdf			实验报告