2023《数字逻辑与处理器基础》处理器大作业

2023/05/23

李栋庭 2020011222 无16

一、 实验目的

1. 掌握 MIPS 单周期处理器的控制通路和数据通路的设计原理和 RTL 实现方法；
2. 掌握处理器与外部设备的通信原理和 RTL 实现方法。

二、 指令集

1. MIPS 指令集子集：

lw, sw, lui, add, addu, sub, subu, addi, addiu, and, or, xor, nor, andi, sll, srl, sra, slt, sltu, slti, sltiu, beq, j, jal, jr, jal

2. MIPS 指令格式：

Instruction	OpCode[5:0]	Rs[4:0]	Rt[4:0]	Rd[4:0]	Shamt[4:0]	Funct[5:0]
lw rt, offset (rs)	0x23	rs	rt	offset		
sw rt, offset (rs)	0x2b	rs	rt	offset		
lui rt, imm	0x0f	0	rt	imm		
add rd, rs, rt	0	rs	rt	rd	0	0x20
addu rd, rs, rt	0	rs	rt	rd	0	0x21
sub rd, rs, rt	0	rs	rt	rd	0	0x22
subu rd, rs, rt	0	rs	rt	rd	0	0x23
addi rt, rs, imm	0x08	rs	rt	imm		
addiu rt, rs, imm	0x09	rs	rt	imm		
and rd, rs, rt	0	rs	rt	rd	0	0x24
or rd, rs, rt	0	rs	rt	rd	0	0x25
xor rd, rs, rt	0	rs	rt	rd	0	0x26
nor rd, rs, rt	0	rs	rt	rd	0	0x27
andi rt, rs, imm	0x0c	rs	rt	imm		
sll rd, rt, shamt	0	0	rt	rd	shamt	0
srl rd, rt, shamt	0	0	rt	rd	shamt	0x02
sra rd, rt, shamt	0	0	rt	rd	shamt	0x03
slt rd, rs, rt	0	rs	rt	rd	0	0x2a
sltu rd, rs, rt	0	rs	rt	rd	0	0x2b
slti rt, rs, imm	0x0a	rs	rt	imm		
sltiu rt, rs, imm	0x0b	rs	rt	imm		
beq rs, rt, label	0x04	rs	rt	offset		
j target	0x02	target				
jal target	0x03	target				
jr rs	0	rs	0			0x08
jalr rd, rs	0	rs	0	rd	0	0x09

三、 实验内容

1. MIPS **单周期** CPU：*single-cycle* 文件夹中，是单周期处理器的 RTL 实现。请阅读 各个基础功能模块的 Verilog 代码，理解每个模块的输入输出接口和基本功能。

   1) 根据对各个控制信号的理解，完成 MIPS 指令集子集与控制信号的真值表(如下表所示，填 0、1、2 、x 等)，并根据填写的真值表完成 *single-cycle* 文件夹中控制器模块 Control.v 的 Verilog 代码实现。

	PCSrc [1:0]	Branch	RegWrite	RegDst [1:0]	MemRead	MemWrite	MemtoReg [1:0]	ALUSrc1	ALUSrc2	ExtOp	LuOp
lw	00	0	1	00	1	0	01	0	1	1	0
sw	00	0	0	x	0	1	x	0	1	1	0
lui	00	0	1	00	0	0	00	0	1	x	1
add	00	0	1	01	0	0	00	0	0	x	0
addu	00	0	1	01	0	0	00	0	0	x	0
sub	00	0	1	01	0	0	00	0	0	x	0
subu	00	0	1	01	0	0	00	0	0	x	0
addi	00	0	1	00	0	0	00	0	1	1	0
addiu	00	0	1	00	0	0	00	0	1	1	0
and	00	0	1	01	0	0	00	0	0	x	0
or	00	0	1	01	0	0	00	0	0	x	0
xor	00	0	1	01	0	0	00	0	0	x	0
nor	00	0	1	01	0	0	00	0	0	x	0
andi	00	0	1	00	0	0	00	0	1	0	0
sll	00	0	1	01	0	0	00	1	0	x	0
srl	00	0	1	01	0	0	00	1	0	x	0
sra	00	0	1	01	0	0	00	1	0	x	0
slt	00	0	1	01	0	0	00	0	0	x	0
sltu	00	0	1	01	0	0	00	0	0	x	0
slti	00	0	1	00	0	0	00	0	1	1	0
sltiu	00	0	1	00	0	0	00	0	1	1	0
beq	00	1	0	x	0	0	00	0	0	x	0
j	01	0	0	x	0	0	x	0	x	x	0
jal	01	0	1	10	0	0	10	0	x	x	0
jr	10	0	0	x	0	0	x	0	x	x	0
jalr	10	0	1	01	0	0	10	0	0	x	0

   2) 阅读 MIPS Assembly 1 中的指令代码。这段程序运行足够长时间后，寄存器 $v0, $v1, $a0, $a1, $a2, $a3, $t0, $t1, $t2, $t3 (分别对应 2- 11 号寄存器)中的值应该是多少？

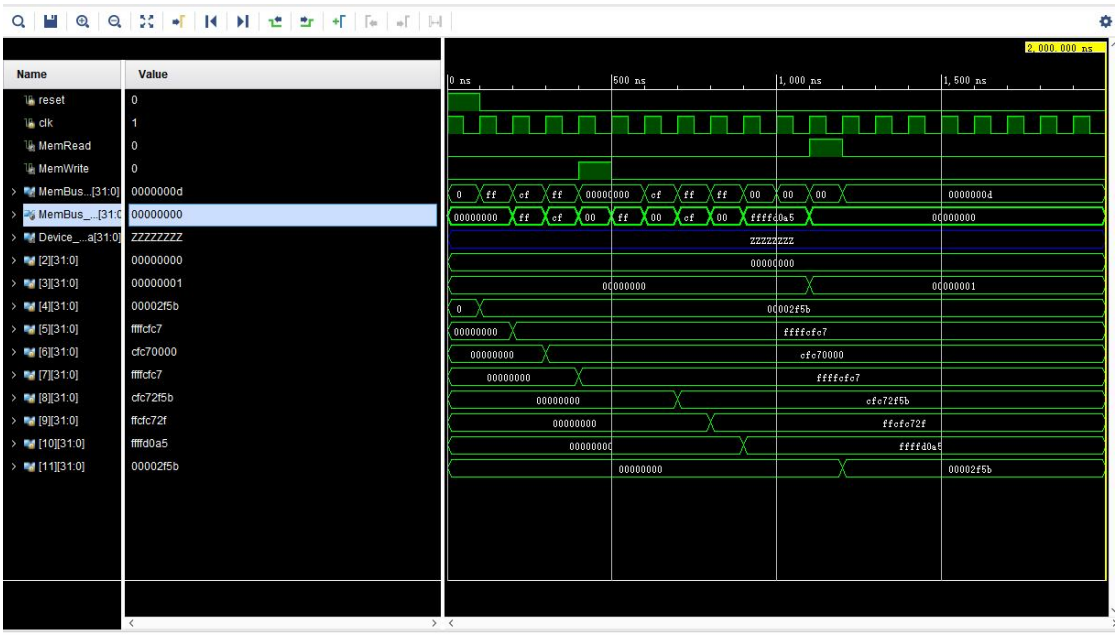
如下表所示：

编号	寄存器	十进制	16进制
2	$v0	0	0x00000000
3	$v1	1	0x00000001
4	$a0	12123	0x00002f5b
5	$a1	-12345	0xffffcfc7

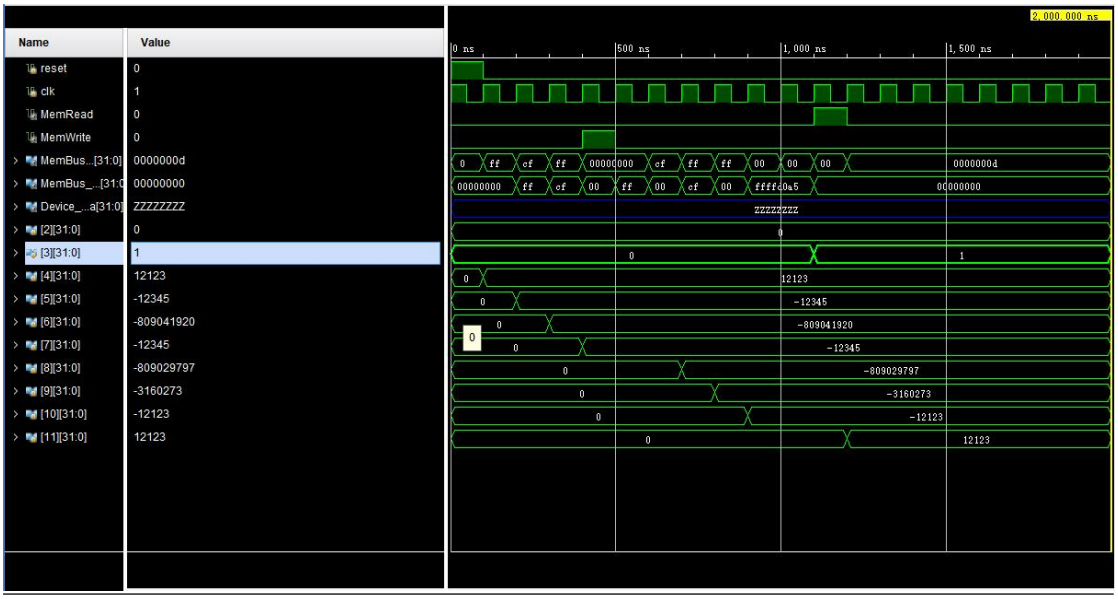
6	$a2	-809041920	0xcfc70000
7	$a3	-12345	0xffffcfc7
8	$t0	-809029797	0xcfc72f5b
9	$t1	-3160273	0xffcfc72f
10	$t2	-12123	0xffffd0a5
11	$t3	12123	0x00002f5b

3) 将 Inst-q1.txt 中的代码(对应 MIPS Assembly 1) 粘贴至 InstructionMemory.v 的 相应位置，使用 ModelSim 或 Vivado 等仿真软件进行仿真，顶层仿真模块为 test_cpu.v 。请给出 b 问中所有寄存器的仿真波形图，验证 b 问中的计算结果是否与仿真结果一致，验证单周期处理器的功能正确性。

16进制：



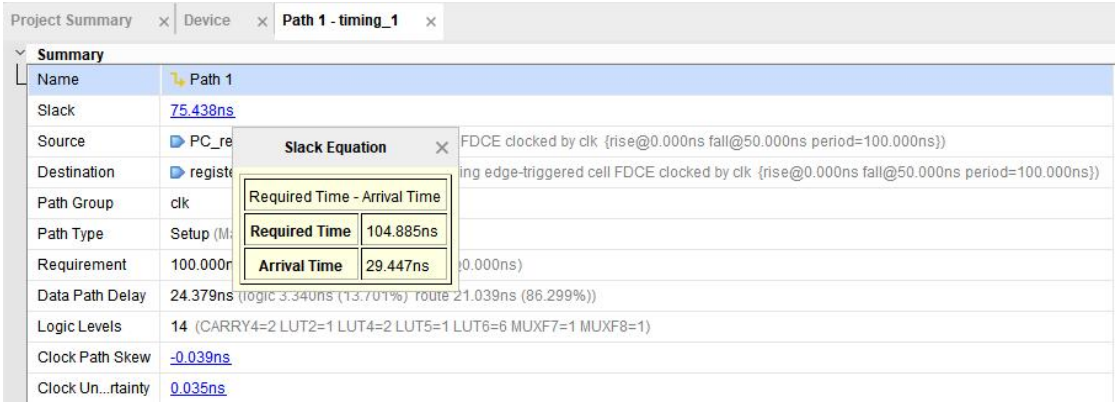
十进制：



如上面两个图所示，b 问中的计算结果与仿真结果一致，验证了单周期处理器的功能正确性。

4) 面向“数逻实验课”所采用的 FPGA 板卡(参见其他说明-5)，基于 Vivado 工具进行综合并开展静态时序分析。根据 Vivado 的资源和时序分析报告，分析说明 CPU 所可能达到的最高时钟频率、所使用的硬件资源开销，附上 Vivado 的综合分析资源和时序报告截图。

如下图是关键路径的部分时序参数：



Slack = Required Time - Arrival Time = 75.438ns

Required Time = Tcycle + Tclk2 - Tsu = 104.885ns

Tsu = 0.109ns

Arrival Time = Tclk1 + Tco + Tlogic = 29.447ns

Tco + Tlogic = Data Path Delay = 24.379ns

Tskew = -0.039ns

由于 Tlogic + Tco + Tsu <= Tcycle_min + Tskew

Tcycle_min = Tlogic + Tco + Tsu + Tun - Tskew = 24.379ns + 0.109ns + 0.035ns + 0.039ns = 24.562ns

事实上 Tcycle_min = Tcycle - Slack = 24.562ns

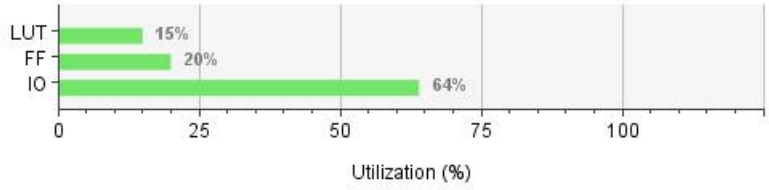
所以CPU 所可能达到的最高时钟频率为 $f_{max} = \frac{1}{Tcylce\_min} \approx 40.7MHZ$。

硬件资源开销中，数据存储器所消耗硬件资源较多，之后是寄存器。

综合分析资源：

Name	Slice LUTs (20800)	Slice Registers (41600)	F7 Muxes (16300)	F8 Muxes (8150)	Bonded IOB (106)	BUFGCTRL (32)
CPU	3170	8425	1024	456	68	1
alu1 (ALU)	8	0	0	0	0	0
data_memory1 (DataM...	2234	8192	1024	456	0	0
register_file1 (Register...	922	224	0	0	0	0

Resource	Utilization	Available	Utilization %
LUT	3170	20800	15.24
FF	8425	41600	20.25
IO	68	106	64.15



时序报告：

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	75.438 ns	Worst Hold Slack (WHS):	0.307 ns	Worst Pulse Width Slack (WPWS):	49.500 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	16841	Total Number of Endpoints:	16841	Total Number of Endpoints:	8426

All user specified timing constraints are met.

Intra-Clock Paths - clk - Setup

Name	Slack	Levels	Routes	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement	Source Clock	Destination Clock	Exception
Path 1	75.438	14	13	117	PC_reg[5]/C	register_file1/R...ta_reg[7][29]/D	24.379	3.340	21.039	100.0	clk	clk	
Path 2	75.649	14	12	123	PC_reg[5]/C	register_file1/R...ta_reg[7][27]/D	24.171	3.280	20.891	100.0	clk	clk	
Path 3	76.076	14	13	117	PC_reg[5]/C	register_file1/R...ta_reg[5][29]/D	23.697	3.340	20.357	100.0	clk	clk	
Path 4	76.091	13	13	127	PC_reg[5]/C	register_file1/R...data_reg[6][5]/D	23.765	3.359	20.406	100.0	clk	clk	
Path 5	76.147	13	13	127	PC_reg[5]/C	register_file1/R...ta_reg[11][5]/D	23.721	3.359	20.362	100.0	clk	clk	
Path 6	76.150	14	13	117	PC_reg[5]/C	register_file1/R...ta_reg[6][29]/D	23.713	3.340	20.373	100.0	clk	clk	
Path 7	76.158	13	13	127	PC_reg[5]/C	register_file1/R...data_reg[7][6]/D	23.684	3.359	20.325	100.0	clk	clk	
Path 8	76.161	13	13	127	PC_reg[5]/C	register_file1/R...data_reg[5][5]/D	23.715	3.359	20.356	100.0	clk	clk	
Path 9	76.177	14	13	117	PC_reg[5]/C	register_file1/R...ta_reg[5][30]/D	23.615	3.340	20.275	100.0	clk	clk	
Path 10	76.185	14	13	117	PC_reg[5]/C	register_file1/R...ta_reg[4][30]/D	23.606	3.340	20.266	100.0	clk	clk	

2. **实现乘法指令：**我们现在需要计算函数f(x, y) = g(x) − h(y)，其中$g(x) = (x^4 + x^3 + x^2 + x + 1)$, $h(y) = (y^2 + y)$。为了计算这个函数，我们决定在该单周期 CPU 中额外支持乘法指令 mul rd, rs, rt。该指令为 R 型指令，实现的功能为实现 rd = rs * rt (rs 和 rt 是有符号 32 位整数，使用补码表示，不必考虑溢出，直接取乘法结果的低 32 位即可) 指令格式如下所示：

Instruction	OpCode[5:0]	Rs[4:0]	Rt[4:0]	Rd[4:0]	Shamt[4:0]	Funct[5:0]
mul rd, rs, rt	0x1c	rs	rt	rd	0	0x2

1) 在前一题的基础上，在 Control.v 和 ALUControl.v 文件中(如有需要，也可以在其他文件中进行相应修改) 补充 mul 指令的相关控制逻辑的 RTL 实现，并在 ALU.v 文件中实现 32 位乘法。请简要写出你的设计思路，并把新增的关键代码粘贴在实验报告中。

设计思路：

在 Control.v 文件中补充上 OpCode == 6'h1c的控制信号；在 ALUControl.v 文件中补充参数 aluMUL 并且在 ALUOp == 3'b110 时，使得 ALUCtl = aluMUL；在 ALU.v 文件中补充上对应 ALUCtl 下的两个操作数的乘法运算。

代码如下：

Control.v:

```
assign RegDst[1:0] =
        (OpCode == 6'h00 || OpCode == 6'h1c)? 2'b01://R-type
        (OpCode == 6'h03)? 2'b10:
        2'b00;//I-type

assign ALUSrc2 =
        (OpCode == 6'h00 || OpCode == 6'h04 || OpCode == 6'h1c)? 1'b0://rt
        1'b1;//imm

assign ALUOp[2:0] =
        (OpCode == 6'h00)? 3'b010://R-type
        (OpCode == 6'h04)? 3'b001://beq
        (OpCode == 6'h0c)? 3'b100://and
        (OpCode == 6'h0a || OpCode == 6'h0b)? 3'b101://slt
        (OpCode == 6'h1c)? 3'b110://mul
        3'b000;//add
```

ALUControl.v:

```
parameter aluMUL = 5'b11010;
always @(*)
        case (ALUOp[2:0])
            3'b000: ALUCtl <= aluADD;
            3'b001: ALUCtl <= aluSUB;
            3'b100: ALUCtl <= aluAND;
```

```
            3'b101: ALUCtl <= aluSLT;
            3'b010: ALUCtl <= aluFunct;
            3'b110: ALUCtl <= aluMUL;
            default: ALUCtl <= aluADD;
        endcase
```

ALU.v:

```
always @(*)
        case (ALUCtl)
            5'b00000: out <= in1 & in2;
            5'b00001: out <= in1 | in2;
            5'b00010: out <= in1 + in2;
            5'b00110: out <= in1 - in2;
            5'b00111: out <= {31'h00000000, Sign? lt_signed: (in1 < in2)};
            5'b01100: out <= ~(in1 | in2);
            5'b01101: out <= in1 ^ in2;
            5'b10000: out <= (in2 << in1[4:0]);
            5'b11000: out <= (in2 >> in1[4:0]);
            5'b11001: out <= ({{32{in2[31]}}, in2} >> in1[4:0]);
            5'b11010: out <= in1 * in2;
            default: out <= 32'h00000000;
        endcase
```

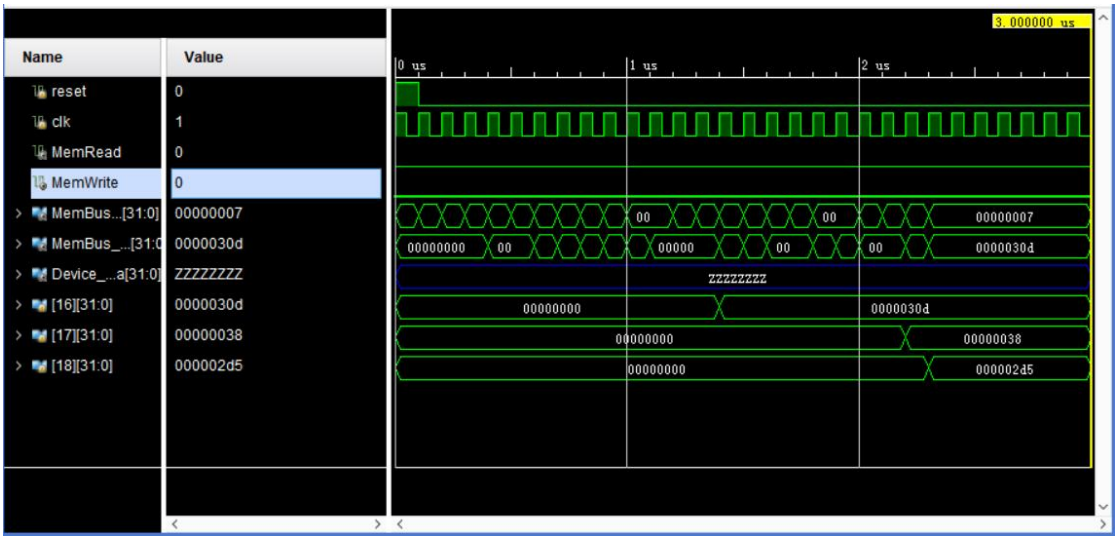
2) 阅读 MIPS Assembly 2 中的指令代码。该代码实现了计算f(x, y)的功能。这段程序运行足够长时间后，寄存器 $s0, $s1, $s2 中的值应该是多少？

如下表所示：

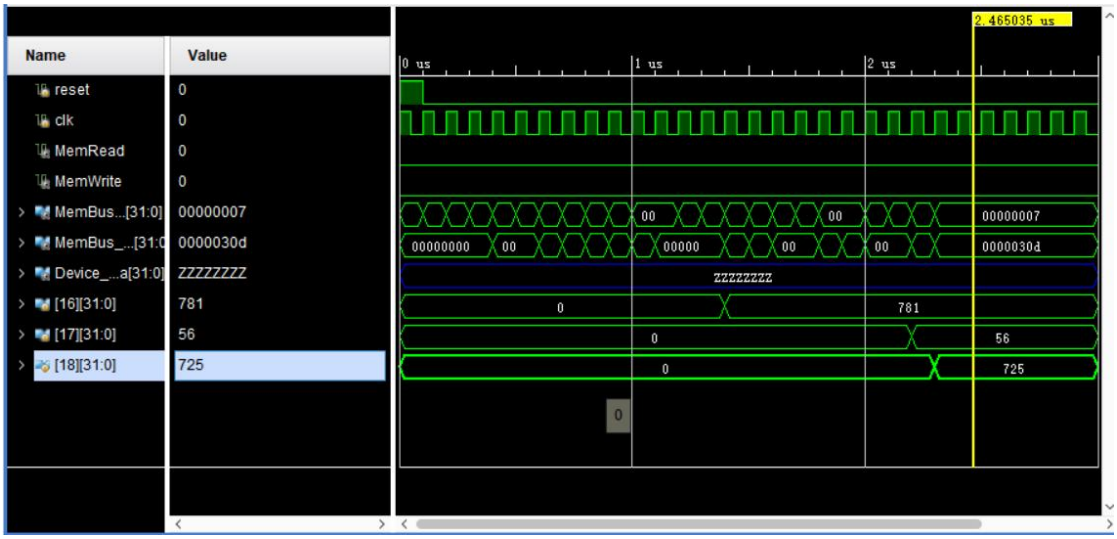
编号	寄存器	十进制	16进制
16	$s0	781	0x0000030d
17	$s1	56	0x00000038
18	$s2	725	0x000002d5

3) 将 Inst-q2.txt 中的代码(对应 MIPS Assembly 2) 粘贴至 InstructionMemory.v 的相应位置，使用 ModelSim 或 Vivado 等仿真软件进行仿真，顶层仿真模块为 test_cpu.v 。请给出 b 问中所有寄存器的仿真波形图，验证 b 问中的计算结果是否与仿真结果一致，验证乘法指令的功能正确性。

16进制：



十进制：



如上面两个图所示，b 问中的计算结果与仿真结果一致，验证了单周期处理器的功能正确性。

3. **实现设备**：为了加速计算，减轻 CPU 的负担，我们计划把计算 g(x)的操作使用外部设备实现(Device.v)。注意设备是通过内存总线挂载到 CPU 上的，所以CPU 并不 需要额外的指令，只需要通过 sw 写入操作数，设备在收到写入指令和数据时初始化有限状态机，并独立于 CPU 完成后续运算。设备中包含：操作数寄存器 reg_op 、结果寄存器 reg_ans、工作状态寄存器 reg_start 和有限状态机。其中 reg_op、reg_ans 、reg_start 均为 32bit 的寄存器，并映射在 CPU 的内存地址空间中，CPU 可以通过使用 sw 和 lw 访问对应的地址写入或读取寄存器的值，通过内存的地址区别访问的是内存还是该设备。

   1) 在前两题的基础上，在 Device.v 文件中实现该设备的功能(如有需要，也可以在等其他文件中进行相应修改)。请简要写出你的设计思路，并把关键代码粘贴在实验报告中。

设计思路：

在 Device.v 文件中：当 MemWrite == 1时，考虑 MemBus_Address 的值，如果为 32'h40000000 或者 32'h40000008，reg_op，reg_start 分别读入；如果 reg_start 非零，将其设为1，进入有限状态机，在5个时钟周期后完后运算，结果放在 Device_Read_Data 中。在 CPU.v 文件中：得到 MemBus_Read_Data 之前，考虑地址，如果地址为 32'h40000004，MemBus_Read_Data = Device_Read_Data。

代码如下：

Device.v:

```
always @(*)
        if(MemWrite == 1)
            case (MemBus_Address)
               32'h40000000: reg_op <= MemBus_Write_Data;
               32'h40000008: begin
                    reg_start <= MemBus_Write_Data;
                    if (reg_start != 0)
                        reg_start = 32'h00000001;
               end
            endcase


    assign Device_Read_Data = reg_ans;

    reg [31:0] t;
```

```
always @(posedge reset or posedge clk)
        if (reset) begin
            reg_op <= 32'h00000000;
            reg_start <= 32'h00000000;
            reg_ans <= 32'h00000000;
        end
        else begin
            case (reg_start)
                32'h00000001: begin
                    reg_ans <= 32'h00000001;
                    t <= reg_op;
                    reg_start <= 32'h00000002;
                end
                32'h00000002: begin
                    reg_ans <= reg_ans + t;
                    t <= t * reg_op;
                    reg_start <= 32'h00000003;
                end
                32'h00000003: begin
                    reg_ans <= reg_ans + t;
                    t <= t * reg_op;
                    reg_start <= 32'h00000004;
                end
                32'h00000004: begin
                    reg_ans <= reg_ans + t;
                    t <= t * reg_op;
                    reg_start <= 32'h00000005;
                end
                32'h00000005: begin
                    reg_ans <= reg_ans + t;
                    reg_start <= 32'h00000000;
                end
            endcase
        end
```

CPU.v:

```
assign MemBus_Read_Data     = (ALU_out == 32'h40000004) ? Device_Read_Data: Memory_Read_Data;
```

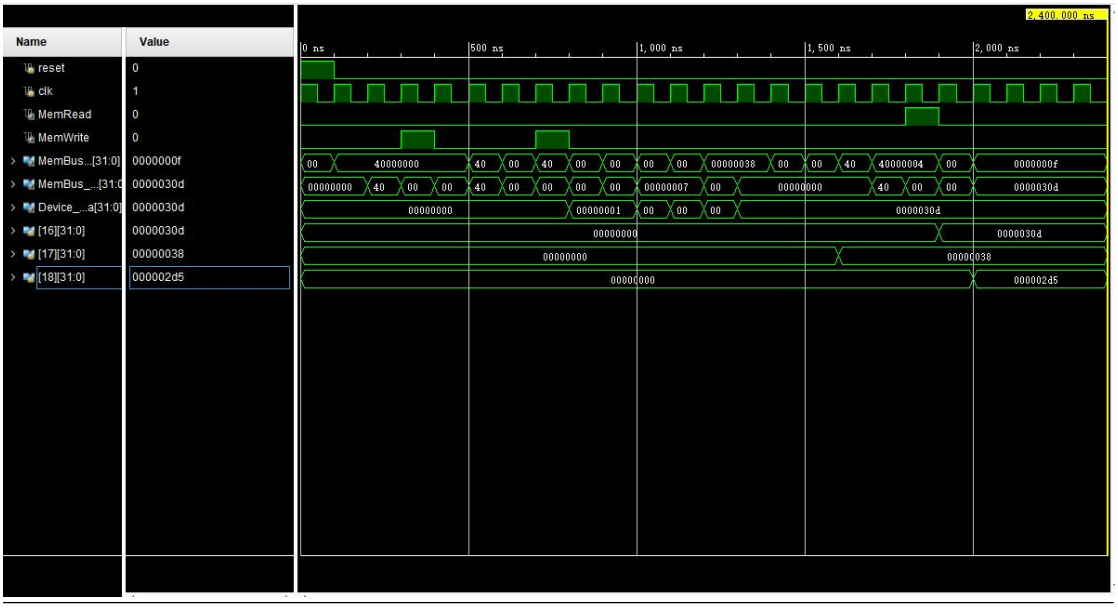
2) 阅读 MIPS Assembly 3 中的指令代码。该代码使用设备实现了 g(x) 的计算(蓝色部分代码)。这段程序运行足够长时间后，寄存器 $s0, $s1,$s2 中的值应该是多少?

如下表所示：

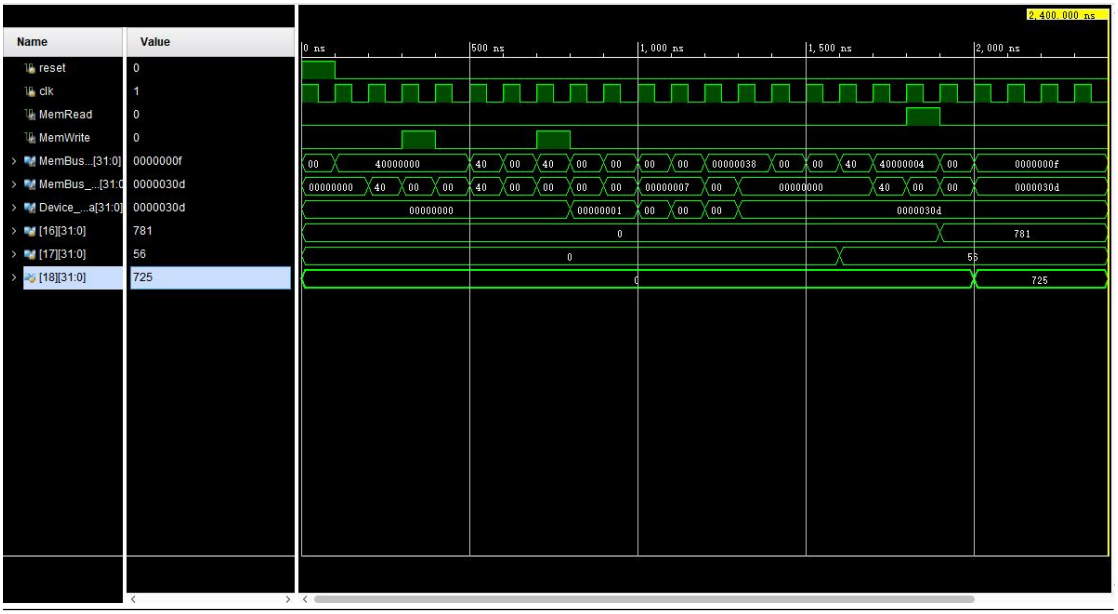
编号	寄存器	十进制	16进制
16	$s0	781	0x0000030d
17	$s1	56	0x00000038
18	$s2	725	0x000002d5

3) 将 Inst-q3.txt 中的代码(对应 MIPS Assembly 3) 粘贴至 InstructionMemory.v 的相应位置，使用 ModelSim 或 Vivado 等仿真软件进行仿真，顶层仿真模块为 test_cpu.v 。请给出 b 问中所有寄存器的仿真波形图，验证 b 问中的计算结果是否与仿真结果一致，验证乘法指令的功能正确性。

16进制：



十进制：



如上面两个图所示，b 问中的计算结果与仿真结果一致，验证了单周期处理器的功能正确性。

4) 请比较修改 CPU 以支持新功能(第 2 问)与通过在内存总线上挂载设备实现新功能(第 3 问)这两种实现方式的区别与特点。

修改CPU以支持新功能：这种方式是在CPU的数据通路和控制部件中添加乘法器模块，根据乘法指令的格式和功能，生成相应的控制信号和数据信号，完成乘法运算。这种方式的优点是可以利用FPGA片上的DSP资源或组合逻辑资源实现高效的乘法器，缺点是需要修改CPU的结构和逻辑，增加设计的复杂度，在实现复杂运算时指令多导致花费时间较大。

通过在内存总线上挂载设备实现新功能：这种方式是在CPU外部添加一个乘法器设备，通过内存映射的方式访问该设备，将操作数写入设备的寄存器，然后读取设备的寄存器得到结果。这种方式的优点是几乎不需要修改CPU的结构和逻辑，只需要增加一个外部设备的接口，并且由于多线程计算所使用时间较少，缺点是需要占用内存地址空间。