中国科学院大学计算机组成原理实验课

实 验 报 告

学号：_2016K8009929018_　姓名：_陈灿宇_　专业：_计算机科学与技术_

实验序号：____5-1____　实验名称：_____复杂处理器设计_____

注 1：本实验报告请以 PDF 格式提交。文件命名规则：学号-prjN.pdf，其中学号中的字母“K”为大写，“-”为英文连字符，“prj”和后缀名“pdf”为小写，“N”为 1 至 5 的阿拉伯数字。例如：2015K8009929000-prj1.pdf。PDF 文件大小应控制在 5MB 以内。
注 2：实验报告模板以下部分的内容供参考，可包含但不限定如下条目内容。

一、 逻辑电路结构与仿真波形的截图及说明（比如关键 RTL 代码段{包含注释}及其对应的逻辑电路结构、相应信号的仿真波形和信号变化的说明等）

在选做的实验中我选择了 prj5-1 和 prj5-3，在 prj5-1 中我完成了多周期处理器的设计和 Cache 模块替换算法的设计两部分，分别 push 到了 GitHub 上的 master 和 cache 分支，暂时未完成 pipeline 的设计。

多周期处理器设计部分我基本上复用了 prj4 的代码，改动的地方并不多，基本上只是将译码执行 ID_EX 阶段拆分成了译码阶段 ID 和执行阶段 EX，其他地方都保持不变，例如下面这段代码：

```
always@(*) begin
    if(rst)
        state_next = IF;
    else begin
        case(state)
            IF: state_next = (Inst_Req_Ack)? IW:IF;
            IW: state_next = (Inst_Valid)?   ID:IW;
            ST: state_next = (Mem_Req_Ack)?  IF:ST;
            LD: state_next = (Mem_Req_Ack)?  RDW:LD;
            RDW: state_next = (Read_data_Valid)? WB:RDW;
            WB: state_next = IF;
            ID: state_next = EX;
            EX:  begin
                if((Instruction_i[31:26] == JAL)
                || (Instruction_i[31:26] == J)
                || (Instruction_i[31:26] == SPECIAL && Instruction_i[5:0] == JR_FUNC)
                || (Instruction_i[31:26] == SPECIAL && Instruction_i[5:0] == JALR_FUNC)
                || (Instruction_i[31:26] == BEQ)
                || (Instruction_i[31:26] == BNE)
                || (Instruction_i[31:26] == BLEZ)
                || (Instruction_i[31:26] == BLTZ)
                || (Instruction_i[31:26] == BGTZ)
                || (Instruction_i[31:26] == SPECIAL && Instruction_i[5:0]== MOVN)
                || (Instruction_i[31:26] == SPECIAL && Instruction_i[5:0]== MOVZ))
                    state_next = IF;
```

```
                else if(Instruction_i[31:26] == LBU
                     || Instruction_i[31:26] == LB
                     || Instruction_i[31:26] == LW
                     || Instruction_i[31:26] == LWL
                     || Instruction_i[31:26] == LWR
                     || Instruction_i[31:26] == LH
                     || Instruction_i[31:26] == LHU)
                    state_next = LD;
                else if(Instruction_i[31:26] == SW
                     || Instruction_i[31:26] == SH
                     || Instruction_i[31:26] == SWL
                     || Instruction_i[31:26] == SWR
                     || Instruction_i[31:26] == SB)
                    state_next = ST;
                else
                    state_next = WB;
            end
            default: state_next = IF;
        endcase
    end
end
```

和译码阶段的代码：

```
always @(posedge clk) begin
    case(cu_state_next)
        IF:begin
            {RegWrite, PCWrite} = {1'b0,1'b0};
        end
        IW: begin
            {RegWrite, PCWrite} = {1'b0,1'b0};
        end
        ID: begin
            case(OpCode)
                ADDIU:
                    {Jump, RegDst,ALUSrc,Mem2Reg,RegWrite, Branch, ALUop1,ALUop0, LoadSign, PCWrite, ALUctr} = {8'b0_0100_0_00,4'b0000, 1'b0, 3'b010};//addiu
                SPECIAL: begin
                    case(Func)
                        6'b001_011:
                            {Jump, RegDst,ALUSrc,Mem2Reg,RegWrite, Branch, ALUop1,ALUop0, LoadSign, PCWrite, ALUctr}
                                    = (cu_rdata_2 != 32'd0) ? {8'b0_1001_0_00,4'b0101, 1'b0, 3'b000} : {8'b0_1000_0_00,4'b0101, 1'b0, 3'b000};//movn
                        6'b001_010:
                            {Jump, RegDst,ALUSrc,Mem2Reg,RegWrite, Branch, ALUop1,ALUop0, LoadSign, PCWrite, ALUctr}
                                    = (cu_rdata_2 == 32'd0) ? {8'b0_1001_0_00,4'b0101, 1'b0, 3'b000} : {8'b0_1000_0_00,4'b0101, 1'b0, 3'b000};//movz

                        6'b001_001:
                            {Jump, RegDst,ALUSrc,Mem2Reg,RegWrite, Branch, ALUop1,ALUop0, LoadSign, PCWrite, ALUctr} =   {8'b1_1001_0_00,4'b0010, 1'b1, 3'b000}; //JALR
                        6'b001_000:
                            {Jump, RegDst,ALUSrc,Mem2Reg,RegWrite, Branch, ALUop1,ALUop0, LoadSign, PCWrite, ALUctr} =   {8'b1_0000_0_00,4'b1111, 1'b1, 3'b000}; //JR
                        6'b100_110:
                            {Jump, RegDst,ALUSrc,Mem2Reg,RegWrite, Branch, ALUop1,ALUop0, LoadSign, PCWrite, ALUctr} =   {8'b0_1000_0_00,4'b1011, 1'b0, 3'b000}; //xor
                        6'b100_111:
                            {Jump, RegDst,ALUSrc,Mem2Reg,RegWrite, Branch, ALUop1,ALUop0, LoadSign, PCWrite, ALUctr} =   {8'b0_1000_0_00,4'b1010, 1'b0, 3'b000}; //nor

                        6'b000_011:
                            {Jump, RegDst,ALUSrc,Mem2Reg,RegWrite, Branch, ALUop1,ALUop0, LoadSign, PCWrite, ALUctr} =   {8'b0_1000_0_00,4'b1001, 1'b0, 3'b000}; //sra
                        6'b000_111:
                            {Jump, RegDst,ALUSrc,Mem2Reg,RegWrite, Branch, ALUop1,ALUop0, LoadSign, PCWrite, ALUctr} =   {8'b0_1000_0_00,4'b1001, 1'b0, 3'b000}; //srav
                        6'b000_010:
                            {Jump, RegDst,ALUSrc,Mem2Reg,RegWrite, Branch, ALUop1,ALUop0, LoadSign, PCWrite, ALUctr} =   {8'b0_1000_0_00,4'b1001, 1'b0, 3'b000}; //srl
                        6'b000_110:
                            {Jump, RegDst,ALUSrc,Mem2Reg,RegWrite, Branch, ALUop1,ALUop0, LoadSign, PCWrite, ALUctr} =   {8'b0_1000_0_00,4'b1001, 1'b0, 3'b000}; //srlv
                        6'b000_000:
                            {Jump, RegDst,ALUSrc,Mem2Reg,RegWrite, Branch, ALUop1,ALUop0, LoadSign, PCWrite, ALUctr} =   {8'b0_1000_0_00,4'b1001, 1'b0, 3'b000}; //sll
                        6'b000_100:
                            {Jump, RegDst,ALUSrc,Mem2Reg,RegWrite, Branch, ALUop1,ALUop0, LoadSign, PCWrite, ALUctr} =   {8'b0_1000_0_00,4'b1001, 1'b0, 3'b000}; //sllv

                        6'b101_011:
                            {Jump, RegDst,ALUSrc,Mem2Reg,RegWrite, Branch, ALUop1,ALUop0, LoadSign, PCWrite, ALUctr} =   {8'b0_1000_0_00,4'b1000, 1'b0, 3'b110}; //sltu

                        6'b100_011:
                            {Jump, RegDst,ALUSrc,Mem2Reg,RegWrite, Branch, ALUop1,ALUop0, LoadSign, PCWrite, ALUctr} =   {8'b0_1000_0_00,4'b0000, 1'b0, 3'b110}; //subu

                    6'b100_001:
                        {Jump, RegDst,ALUSrc,Mem2Reg,RegWrite, Branch, ALUop1,ALUop0, LoadSign, PCWrite, ALUctr} =   {8'b0_1000_0_00,4'b0000, 1'b0, 3'b010}; //addu

                    6'b100_101:
                        {Jump, RegDst,ALUSrc,Mem2Reg,RegWrite, Branch, ALUop1,ALUop0, LoadSign, PCWrite, ALUctr} =   {8'b0_1000_0_00,4'b0000, 1'b0, 3'b001}; //move(or)
                    6'b101_010:
                        {Jump, RegDst,ALUSrc,Mem2Reg,RegWrite, Branch, ALUop1,ALUop0, LoadSign, PCWrite, ALUctr} =   {8'b0_1000_0_00,4'b0000, 1'b0, 3'b111}; //slt
                    6'b100_000:
                        {Jump, RegDst,ALUSrc,Mem2Reg,RegWrite, Branch, ALUop1,ALUop0, LoadSign, PCWrite, ALUctr} =   {8'b0_1000_0_00,4'b0000, 1'b0, 3'b010}; //add
                    6'b100_100:
                        {Jump, RegDst,ALUSrc,Mem2Reg,RegWrite, Branch, ALUop1,ALUop0, LoadSign, PCWrite, ALUctr} =   {8'b0_1000_0_00,4'b0000, 1'b0, 3'b000}; //and
                    6'b100_101:
                        {Jump, RegDst,ALUSrc,Mem2Reg,RegWrite, Branch, ALUop1,ALUop0, LoadSign, PCWrite, ALUctr} =   {8'b0_1000_0_00,4'b0000, 1'b0, 3'b001}; //or

                    default:
                        {Jump, RegDst,ALUSrc,Mem2Reg,RegWrite, Branch, ALUop1,ALUop0, LoadSign, PCWrite, ALUctr} =   {8'b0_1000_0_00,4'b0000, 1'b0, 3'b000}; //alu result
                endcase
            end
            BEQ:
                {Jump, RegDst,ALUSrc,Mem2Reg,RegWrite, Branch, ALUop1,ALUop0, LoadSign, PCWrite, ALUctr} = {8'b0_0000_1_01,4'b0000, 1'b1, 3'b110}; //beq
            BNE:
                {Jump, RegDst,ALUSrc,Mem2Reg,RegWrite, Branch, ALUop1,ALUop0, LoadSign, PCWrite, ALUctr} = {8'b0_0000_1_01,4'b0000, 1'b1, 3'b110}; //bne
            REGIMM:
                {Jump, RegDst,ALUSrc,Mem2Reg,RegWrite, Branch, ALUop1,ALUop0, LoadSign, PCWrite, ALUctr} = {8'b0_0000_1_00,4'b1111, 1'b1, 3'b000}; //bgez, bltz
            BLEZ:
                {Jump, RegDst,ALUSrc,Mem2Reg,RegWrite, Branch, ALUop1,ALUop0, LoadSign, PCWrite, ALUctr} = {8'b0_0000_1_00,4'b1111, 1'b1, 3'b000}; //blez
            BGTZ:
                {Jump, RegDst,ALUSrc,Mem2Reg,RegWrite, Branch, ALUop1,ALUop0, LoadSign, PCWrite, ALUctr} = {8'b0_0000_1_00,4'b1111, 1'b1, 3'b000}; //bgtz

            SLTI:
                {Jump, RegDst,ALUSrc,Mem2Reg,RegWrite, Branch, ALUop1,ALUop0, LoadSign, PCWrite, ALUctr} = {8'b0_0100_0_00,4'b0000, 1'b0, 3'b111}; //slti
            SLTIU:
                {Jump, RegDst,ALUSrc,Mem2Reg,RegWrite, Branch, ALUop1,ALUop0, LoadSign, PCWrite, ALUctr} = {8'b0_0100_0_00,4'b0001, 1'b0, 3'b110}; //sltiu

            J:
                {Jump, RegDst,ALUSrc,Mem2Reg,RegWrite, Branch, ALUop1,ALUop0, LoadSign, PCWrite, ALUctr} = {8'b1_0000_0_00,4'b1111, 1'b1, 3'b000}; //j
            JAL:
                {Jump, RegDst,ALUSrc,Mem2Reg,RegWrite, Branch, ALUop1,ALUop0, LoadSign, PCWrite, ALUctr} = {8'b1_0001_0_00,4'b0010, 1'b1, 3'b000}; //jal

            LUI:
                {Jump, RegDst,ALUSrc,Mem2Reg,RegWrite, Branch, ALUop1,ALUop0, LoadSign, PCWrite, ALUctr} = {8'b0_0000_0_00,4'b0100, 1'b0, 3'b000}; //lui

            ANDI:
                {Jump, RegDst,ALUSrc,Mem2Reg,RegWrite, Branch, ALUop1,ALUop0, LoadSign, PCWrite, ALUctr} = {8'b0_0000_0_00,4'b1100, 1'b0, 3'b000}; //andi
            ORI:
                {Jump, RegDst,ALUSrc,Mem2Reg,RegWrite, Branch, ALUop1,ALUop0, LoadSign, PCWrite, ALUctr} = {8'b0_0000_0_00,4'b1101, 1'b0, 3'b000}; //ori
            XORI:
```

```
                {Jump, RegDst,ALUSrc,Mem2Reg,RegWrite, Branch, ALUop1,ALUop0, LoadSign, PCWrite, ALUctr} = {8'b0_0000_0_00,4'b1110, 1'b0, 3'b000}; //xori

            LW:
                {Jump, RegDst,ALUSrc,Mem2Reg,RegWrite, Branch, ALUop1,ALUop0, LoadSign, PCWrite, ALUctr} = {8'b0_0110_0_00,4'b0011, 1'b0, 3'b010}; //lw
            LB:
                {Jump, RegDst,ALUSrc,Mem2Reg,RegWrite, Branch, ALUop1,ALUop0, LoadSign, PCWrite, ALUctr} = {8'b0_0110_0_00,4'b0011, 1'b0, 3'b010}; //lb
            LBU:
                {Jump, RegDst,ALUSrc,Mem2Reg,RegWrite, Branch, ALUop1,ALUop0, LoadSign, PCWrite, ALUctr} = {8'b0_0110_0_00,4'b0011, 1'b0, 3'b010}; //lbu
            LH:
                {Jump, RegDst,ALUSrc,Mem2Reg,RegWrite, Branch, ALUop1,ALUop0, LoadSign, PCWrite, ALUctr} = {8'b0_0110_0_00,4'b0011, 1'b0, 3'b010}; //lh
            LHU:
                {Jump, RegDst,ALUSrc,Mem2Reg,RegWrite, Branch, ALUop1,ALUop0, LoadSign, PCWrite, ALUctr} = {8'b0_0110_0_00,4'b0011, 1'b0, 3'b010}; //lhu
            LWL:
                {Jump, RegDst,ALUSrc,Mem2Reg,RegWrite, Branch, ALUop1,ALUop0, LoadSign, PCWrite, ALUctr} = {8'b0_0110_0_00,4'b0011, 1'b0, 3'b010}; //lwl
            LWR:
                {Jump, RegDst,ALUSrc,Mem2Reg,RegWrite, Branch, ALUop1,ALUop0, LoadSign, PCWrite, ALUctr} = {8'b0_0110_0_00,4'b0011, 1'b0, 3'b010}; //lwr

            SW:
                {Jump, RegDst,ALUSrc,Mem2Reg,RegWrite, Branch, ALUop1,ALUop0, LoadSign, PCWrite, ALUctr} = {8'b0_0100_0_00,4'b0100, 1'b0, 3'b010}; //sw
            SB:
                {Jump, RegDst,ALUSrc,Mem2Reg,RegWrite, Branch, ALUop1,ALUop0, LoadSign, PCWrite, ALUctr} = {8'b0_0100_0_00,4'b0100, 1'b0, 3'b010}; //sb
            SH:
                {Jump, RegDst,ALUSrc,Mem2Reg,RegWrite, Branch, ALUop1,ALUop0, LoadSign, PCWrite, ALUctr} = {8'b0_0100_0_00,4'b0100, 1'b0, 3'b010}; //sh
            SWL:
                {Jump, RegDst,ALUSrc,Mem2Reg,RegWrite, Branch, ALUop1,ALUop0, LoadSign, PCWrite, ALUctr} = {8'b0_0100_0_00,4'b0100, 1'b0, 3'b010}; //swl
            SWR:
                {Jump, RegDst,ALUSrc,Mem2Reg,RegWrite, Branch, ALUop1,ALUop0, LoadSign, PCWrite, ALUctr} = {8'b0_0100_0_00,4'b0100, 1'b0, 3'b010}; //swr

            default:
                {Jump, RegDst,ALUSrc,Mem2Reg,RegWrite, Branch, ALUop1,ALUop0, LoadSign, PCWrite, ALUctr} = {8'b0_0000_0_00,4'b1111, 1'b0, 3'b000}; //swr

        endcase
    end
    EX: begin
        ;
    end
    ST: begin
        {RegWrite, PCWrite} = {1'b0,1'b0};
    end
    LD: begin
        {RegWrite, PCWrite} = {1'b0,1'b0};
    end
    RDW: begin
        {RegWrite, PCWrite} = {1'b0,1'b0};
    end
    WB: begin
        {RegWrite, PCWrite} = {1'b1,1'b0};
    end

      default: begin
          {Jump, RegDst,ALUSrc,Mem2Reg,RegWrite, Branch, ALUop1,ALUop0, LoadSign, PCWrite, ALUctr} = {8'b0_0000_0_00,4'b1111, 1'b0, 3'b000};
          //{RegWrite} = {1'b0};
      end
  endcase

end
```

在 Cache 的替换算法方面我主要参考了 PPT 中的做法实现 Replacing Policy 部分，核心代码如下图：

```
module replacing_policy
#
(
    parameter NUM_WAY = 4
)
(
    input   [NUM_WAY - 1 : 0] valid_flatted_in,
    input   [NUM_WAY - 1 : 0] history_flatted_in,
    output  [NUM_WAY - 1 : 0] replaced_way_out
);

// insert your replacing policy here
    assign replaced_way_out = (valid_flatted_in[0] == 1'b0) ? {1'b0, 1'b0, 1'b0, 1'b1}
                            :((valid_flatted_in[1] == 1'b0) ? {1'b0, 1'b0, 1'b1, 1'b0}
                            :((valid_flatted_in[2] == 1'b0) ? {1'b0, 1'b1, 1'b0, 1'b0}
                            :((valid_flatted_in[3] == 1'b0) ? {1'b1, 1'b0, 1'b0, 1'b0}
                            :((history_flatted_in[0] == 1'b0) ? {1'b0, 1'b0, 1'b0, 1'b1}
                            :((history_flatted_in[1] == 1'b0) ? {1'b0, 1'b0, 1'b1, 1'b0}
                            :((history_flatted_in[2] == 1'b0) ? {1'b0, 1'b1, 1'b0, 1'b0}
                            :((history_flatted_in[3] == 1'b0) ? {1'b1, 1'b0, 1'b0, 1'b0}
                            : {1'b1, 1'b1, 1'b1, 1'b1}))))))));

endmodule
```

在本次实验中我尝试测试了一下我设计的 CPU 可以跑多快，首先下图是我在 100Hz 下测试的数据：

Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	3.324 ns	Worst Hold Slack (WHS):	0.024 ns	Worst Pulse Width Slack (WPWS):	4.427 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:	7132	Total Number of Endpoints:	7132	Total Number of Endpoints:	3654

All user specified timing constraints are met.

我比较了一下在 100Hz,110Hz,120Hz 下我的 CPU 上板测试时跑 advanced benchmark 的所用的周期数，发现 100Hz 到 110Hz 所用的周期数明显增多，但是 110 到 120Hz 周期数变化并不明显，：

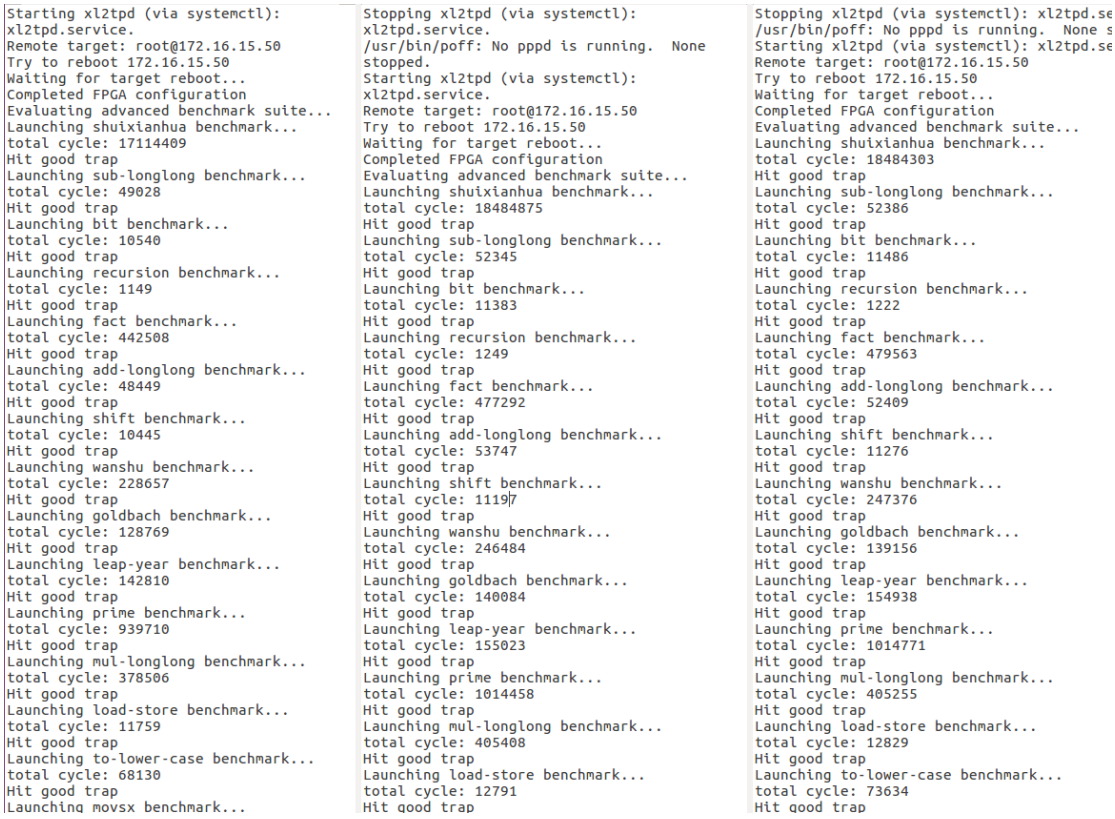


Figure 1:100Hz, 110Hz, 120Hz 下上板测试 advanced benchmark

然后我连续测试了 100Hz,110Hz,120Hz ,150Hz,160Hz,180Hz,190Hz 时的 WNS 的值，发现我的 CPU 最快的极限频率接近 190Hz。

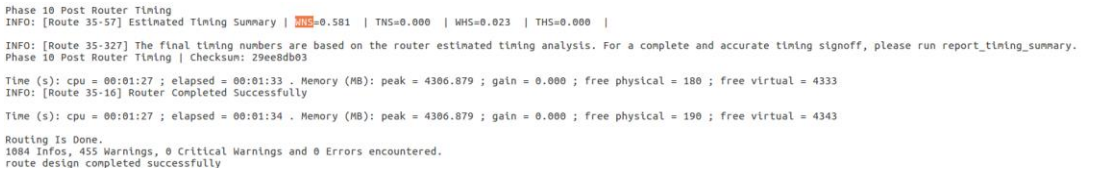


Figure 2：150Hz 下上板测试，WNS>0

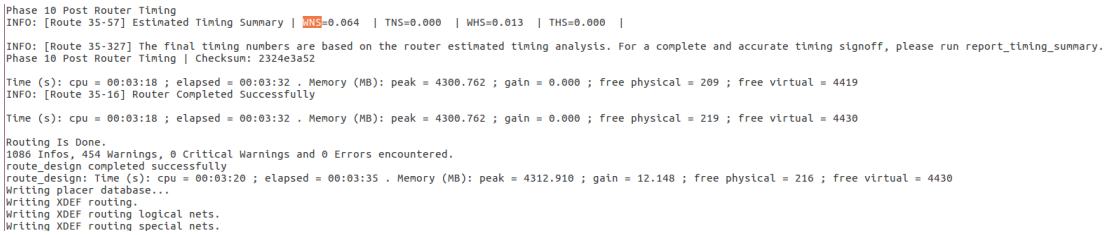


Figure 3：180Hz 下上板测试,WNS>0

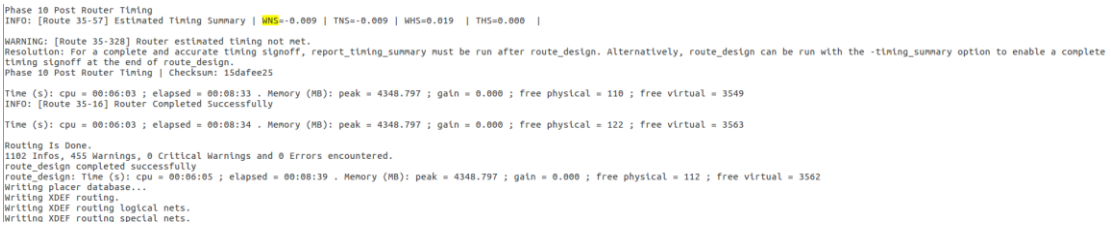
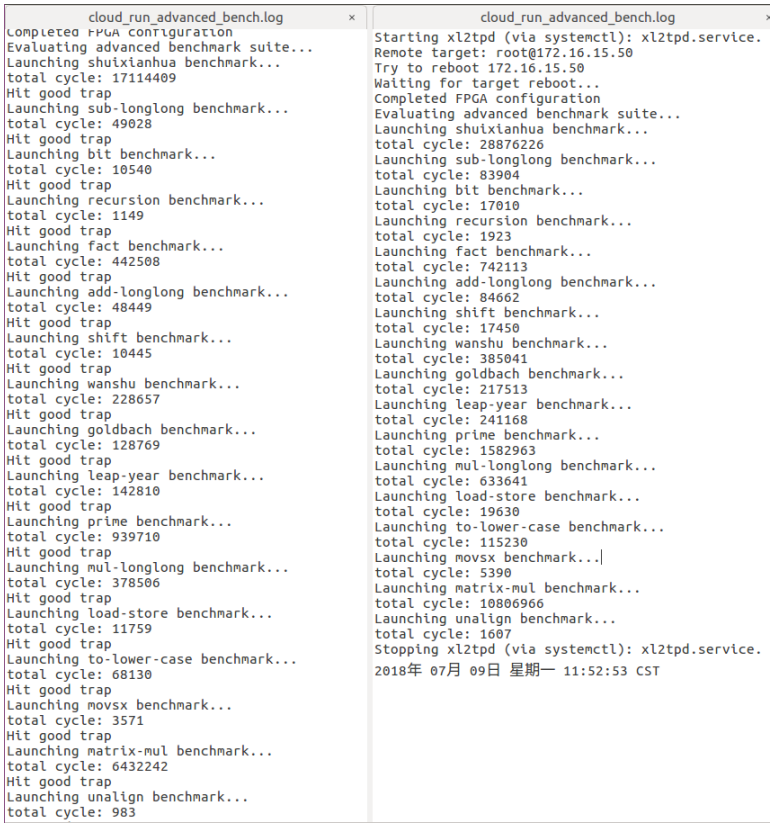


Figure 4：190Hz 下上板测试 WNS<0

然后我比较了一下有 cache 和无 cache 的情况下上，都在 100Hz 下时测试 advanced benchmark，我发现添加 cache 之后上板测试的周期数明显增加，这可以说明 cache 的存在对于访存做了很大的优化，使得 CPU 的频率能进一步提高，CPU 能跑得更快了。



二、 实验过程中遇到的问题、对问题的思考过程及解决方法（比如 RTL 代码中出现的逻辑 bug，仿真、本地上板及云平台调试过程中的难点等）

- 本次实验进行的比较顺利，基本上没有遇到什么 BUG

三、 对讲义中思考题（如有）的理解和回答

- 思考：如果想在main()函数中使用当前C源码文件中未定义声明的bench_prepare()、bench_done()和printf()，还要做什么？

还需要将 perf_cnt.h, printf.h 两个头文件包含进去

```
#include "trap.h"
#include "../../lib/include/perf_cnt.h"
#include "../../lib/include/printf.h"
```

四、 对于此次实验的心得、感受和建议（比如实验是否过于简单或复杂，是否缺少了某些你认为重要的信息或参考资料，对实验项目的建议，对提供帮助的同学的感谢，以及其他想与任课老师交流的内容等）

- 本次实验建立在之前的实验的基础之上，整体难度并不大，让我对于 cache 与 cpu 之间协同工作的原理有了更加深入的认识。