2023 春计算机组成课程实验大作业实验报告

黄奔皓*

上海交通大学

目录

*感谢陈老师的悉心指导

1 程序执行周期数的采样模块设计与实现

1.1 增加 I/O 端口口的设计思路

在 lab4 已有代码的基础上，我们需要增加一个地址为 0x88 的 I/O 端口 inport2。为此，我们需要做如下修改：

- 我们需要对 io_input.v 进行修改，为 io_input 模块增加 inport2 的端口，并增加 reg 类型变量 in_reg2, 用于存储 inport2 的值。同时我们还需要修改 io_input_mux 多选器模块，增加一种选择，与 inport2 配合。
- 同时，我们需要修改 sc_datamem.v 文件，以增加 inport2 端口，并修改在其中例化的 io_input 模块。
- 修改 sc_computer_main.v 模块，增加 inport2 端口，并修改在其中例化的 sc_datamem 模块。
- 修改顶层文件 sc_cpu_iotest.v, 在其中声明 wire 类型变量 inport2，并修改在其中例化的 sc_computer_main 模块。

1.2 增加一个计数器模块

计数器的实现，即每次时钟上升沿时对 count 变量 +1；若有 rst 的下降沿信号，那么我们清零 count。具体计数器代码设计如下：

```
1  module sys_clk_counter (
2    input wire sys_clk_in,       // 系统时钟输入
3    input wire sys_rst_n,        // 异步复位信号输入
4    output reg [31:0] count      // 32位计数器输出
5  );
6
7  always @(posedge sys_clk_in or negedge sys_rst_n) begin
8    if (~sys_rst_n) begin        // 异步复位
9      count <= 0;
10   end else begin              // 正常计数
11     count <= count + 1;
12   end
13 end
14
15 endmodule
```

Figure 1: 具有异步清零功能的 32bit 计数器 sys_clk_counter

我们添加计数器到 lab4 的顶层模块 sc_cpu_iotest.v, 然后在其中例化计数器模块，并将计数器输出连接到 inport2 即可。

1.3 lab2.1 测试程序修改

1.3.1 分析端口的读写方法，修改 coe 文件

结合程序数据通路进行分析知，我们需要利用 lw 指令将 inport2 的值读入 x20 寄存器中，然后用 sw 指令将 x20 的值读入 outport2 中。同时，为了让七段数码管的 2，3 位显示 x18 寄存器十进制值的最后两位，我们还要将 x18 寄存器的值读入 outport1 中。outport2 的地址为 0x88=136，outport1 的地址为 0x84=132。于是我们需要添加如下几条汇编命令：

```
xori x12,x0,0x88
xori x13,x0,0x84
xori x14,x0,0x80
lw x20,0(x12) //从inport2中读取到x20中
sw x20,0(x12) //从x20读取到outport2中
sw x18,0(x13) //从x18中读取到outport1中
lw x22,0(x14) //从inport0中读取到x22中
sw x22,0(x14) //从x22中读取到outport0中
fi:
j fi
```

Figure 2: 新增指令

将上述三条指令输入 RIPES，得到对应的机器码。然后我们对 coe 文件进行相应的修改，在 00f95913 和 0000006f 之间添加上述三条指令的机器码即可。具体的 coe 代码见附录。

1.3.2 学号展示

为了让七段数码管最高两位显示学号，我们需要在 sc_cpu_iotest.v 增加 wire [3:0] HEX4b7,HEX4b6，然后声明变量 wire [31:0] student_num 变量用于存储学号，然后例化 out_port_hex2dec 模块: out_port_hex2dec dec76(student_num, HEX4b7,HEX4b6)

1.4 更新设计完成验证

1.4.1 程序执行周期数的验证

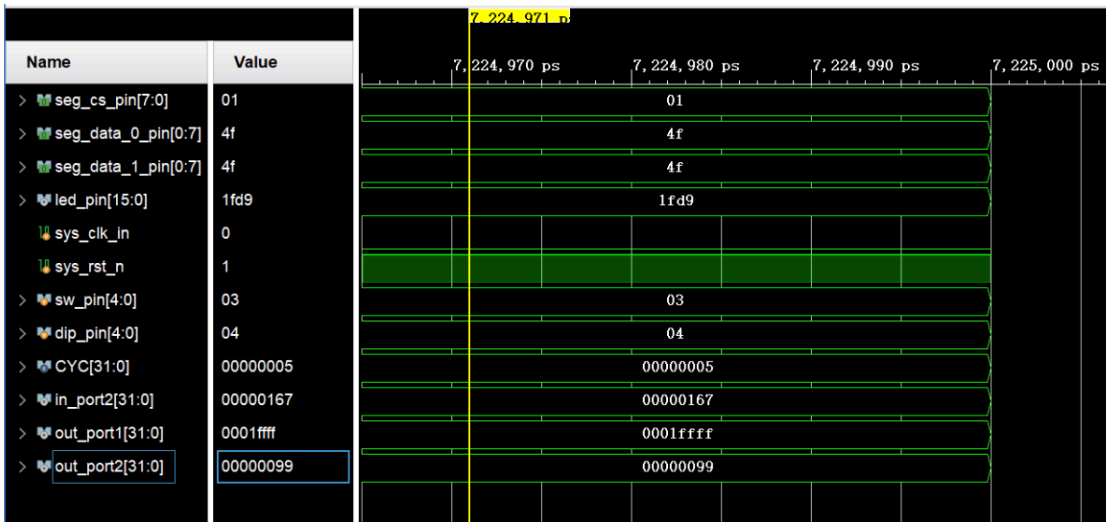

Figure 3: 程序仿真得到的时钟周期数和端口值

从 simulation 中可以看到，程序执行完毕后，寄存器 x20 的值存入了 out_port2，其值为程序执行到将 counter 的值存入 x20 所用的时钟数，为 0x99=153。而寄存器 x18 的值存在 out_port1 中，为 0x1ffff。

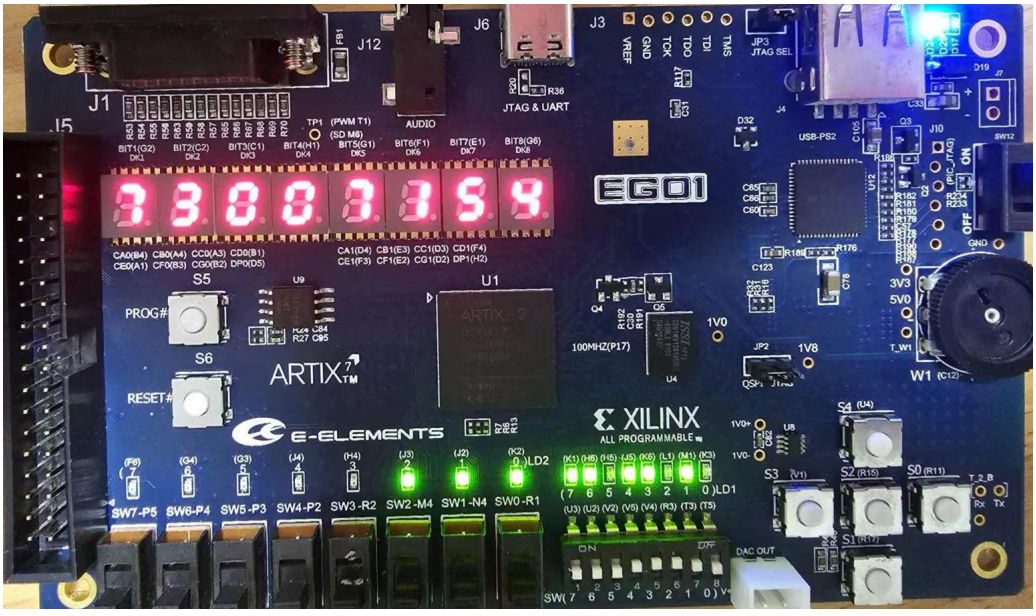

Figure 4: 时钟周期数板级验证

七段数码管的最高两位显示了我的学号末两位数字；七段数码管最后两位（0，1 位）显示了 out_port2 的十进制最后两位为 54，与仿真结果相差 1；第 2，3 位显示了程序

执行后寄存器 x18 值的十进制后两位数，0x1ffff=131071, 最后两位为 71，与板上显示的数字一致。时钟周期的误差可能与总周期数为奇数有关，询问老师后，得知有一定的波动是正常的。

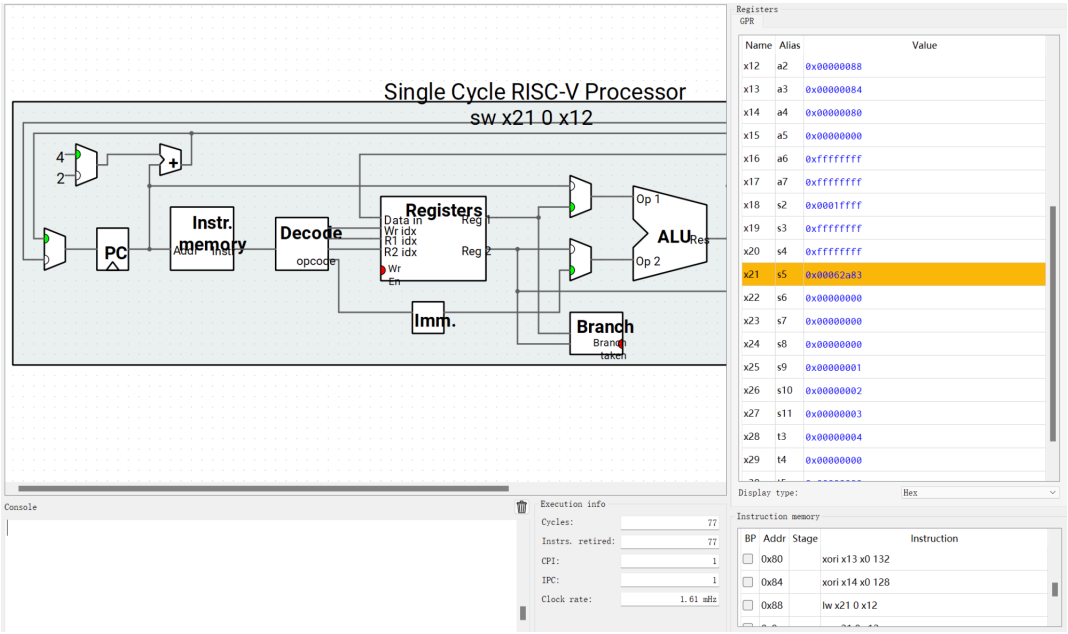

Figure 5: 程序在 RIPES 上运行的时钟周期数和寄存器值

从 RIPES 上进行仿真的结果中看出，当执行完将 counter 的值存入 x20 的指令后，使用的时钟周期为 77。而 $2 \times 77 = 154$, 这与板级验证的结果相符，程序执行周期数的验证完毕。

2 基于自定义指令或已有的 22 条指令的极值搜索程序的设计仿真和板级验证

2.1 修改模块代码

为了实现极值搜索程序，我们在 22 条指令的基础上添加 blt 分支控制指令。为此，我们需要修改 sc_cu 文件，加入 i_blt SB 类型指令 wire i_blt = (op == 7'b1100011) & (func3 == 3'b100)，修改相应的 aluc 控制信号,aluc[3] = aluc[3] | i_blt。此外，由于 blt 需要判断 ra 和 rb 的大小，所以我们在 alu 模块中增加端口 lz，lz=1 说明 ra<rb, 否则 lz=0，在此基础上修改 pcsource = pcsource[0] | (i_blt & lz)

```
wire i_bne  =  (op == 7'b1100011) & (func3 == 3'b001)  ;
wire i_blt =   (op == 7'b1100011) & (func3 == 3'b100)  ; // new inst
//U type
wire i_lui  =  (op == 7'b0110111);
//UJ type
wire i_jal  =  (op == 7'b1101111);

assign pcsource[1] = i_jalr | i_jal;
assign pcsource[0] = ( i_beq & z ) | (i_bne & ~z) | (i_blt & lz) | i_jal ;

assign wreg = i_add | i_sub | i_and | i_or   | i_xor  |
              i_sll | i_srl | i_sra | i_addi | i_andi |
              i_ori | i_xori | i_srli | i_slli | i_srai | i_lw | i_jalr | i_lui  | i_jal | i_hamd;


assign aluc[3] = i_sub | i_srai | i_beq | i_bne | i_hamd | i_blt;
assign aluc[2] = i_and | i_or | i_xor | i_srl | i_sra | i_andi | i_ori| i_xori | i_srli | i_srai | i_h
assign aluc[1] = i_and | i_or | i_andi | i_ori |i_lui | i_hamd;
assign aluc[0] = i_and | i_sll | i_srl | i_sra | i_andi| i_slli | i_srli | i_srai | i_hamd;

assign aluimm  = i_addi | i_andi | i_ori | i_xori| i_slli| i_srli | i_srai | i_lw | i_jalr |i_sw |i_lu
// Here srai is included simply because its inst[31] has been considered
assign sext    = i_addi | i_lw | i_jalr | i_sw | i_beq | i_bne | i_blt | i_jal | i_andi |i_ori |i_xori
assign wmem    = i_sw;
```

Figure 6: sc_cu.v 文件中针对 blt 新定义指令的修改

```
module alu (a, b, aluc, s, z, lz);
   input [31:0] a, b;
   input [3:0] aluc;
   output [31:0] s;
   output        z;
   output        lz;
                                  end
   if (s == 0 )  z = 1;
   else z = 0;
   if (a < b) lz = 1;
   else lz = 0;
```

Figure 7: alu.v 中增加 lz 端口，用来记录 ra 和 rb 的大小情况

2.2 极值搜索的汇编源程序

Assembly language ▾ 自动换行 | 复制

```
.data
v:  .4byte 0x52,0x10,0x30,0x91,0x00,0x73
.4byte 0x37,0x00,0x19,0x03,0x01,0x25
.text
ori x11, x0,0 # max
ori x12, x0,99 # min
ori x20,x0,12  # counter
lui x10, 0x10000
ori x4, x10, 0 # pointer

loop: beq x20,x0, finish # 如果x20等于0，则程序结束

lw x13, 0(x4) # 载入当前数

bgt x13,x11,max # 如果当前数比max大，就跳转
call_1:
    bgt x12,x13,min # 如果当前数比min小，就跳转
call_2:
    addi x20, x20, -1
    addi x4,x4,0x4
    j loop

max:
    addi,x11,x13,0 # 更新max
    j call_1 # 返回
min:
    addi,x12,x13,0 # 更新min
    j call_2 # 返回
finish:
j finish
```

Figure 8: 极值搜索程序版本 1

Execution info

Cycles:	100
Instrs. retired:	100
CPI:	1
IPC:	1
Clock rate:	-1.01 Hz

Figure 9: 极值搜索程序版本 1 需要 100 个时钟

我们可以对以上程序进行优化。版本 1 中，没有很好地利用比较 max 和 min 两个过程潜在的的串行性，其中跳转的方向是从前往后再回到前，存在冗余性。因此，我们

可以把“如果输入值小于 min 则跳转，更新 min 后返回跳转指令的下一条指令，如果输入值小于 max 则跳转，更新 max 后返回，进入下一轮循环”改成如果“输入值不小于 min，则跳过更新 min；反之更新 min。接着判断输入值是否大于 max，与 min 同理。最后返回 loop”。后者比前者少了分支跳转后的返回，起到了节省时钟的作用。

```
.data
v:  .4byte 0x52,0x10,0x30,0x91,0x00,0x73,0x37,0x00,0x19,0x03,0x01,0x25
.text

# 初始化寄存器
lui x10, 0x10000
addi x4, x10, 0     # pointer
lw x18, 0(x4)        # min = 第一个元素
lw x19, 0(x4)        # max = 第一个元素
addi x20, x0, 11     # counter = 11

loop:
    addi x4, x4, 0x4 # Increase pointer by 4 bytes (32 bits)

    lw x21, 0(x4)    # 载入下一个数

    blt x18, x21, min_skip # if x13 >= x11, skip updating min
    addi x18, x21,0     # Update min
min_skip:
    blt x21, x19, max_skip # 如果 x13 <= x12, skip updating max
    addi x19, x21,0      # Update max
max_skip:
    addi x20, x20, -1 # counter --
    bne x20,x0, loop    # counter不为0, 继续循环
lw x22,136(x0)
sw x22,136(x0)
sw x19,128(x0)
sw x18,132(x0)
finish:
    j finish

```

Figure 10: 极值搜索程序版本 2

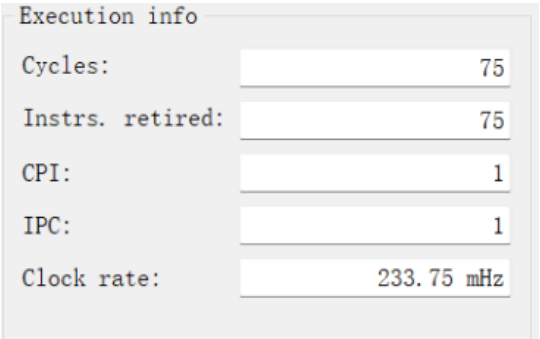

Figure 11: 极值搜索程序版本 2 需要 75 个时钟

2.3 Vivado 仿真结果

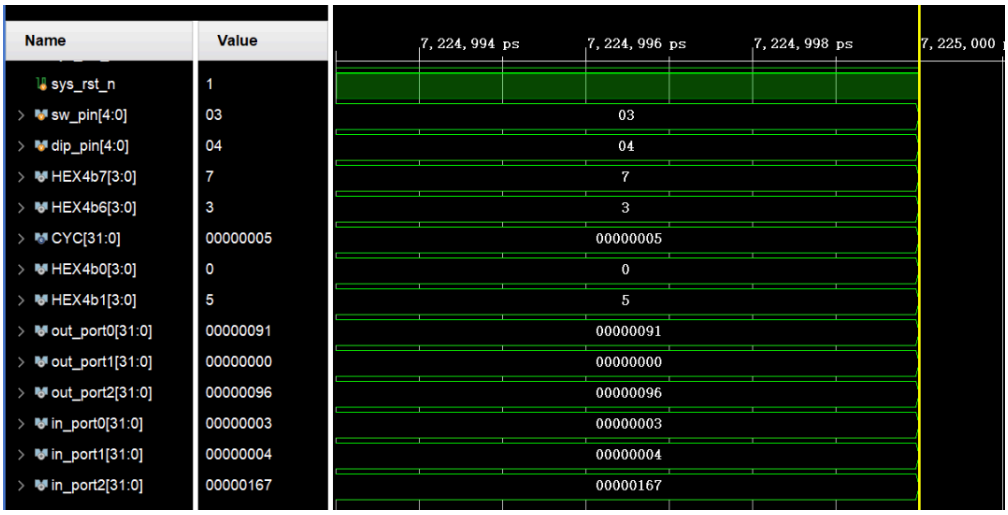

Figure 12: 极值搜索程序 vivado 仿真

图中 out_port0 显示最大值 91，out_port1 显示最小值 0，out_port2 显示的是时钟周期数，为 0x96 = 150 = 2 × 75，符合我们的预期。

2.4 板级执行结果

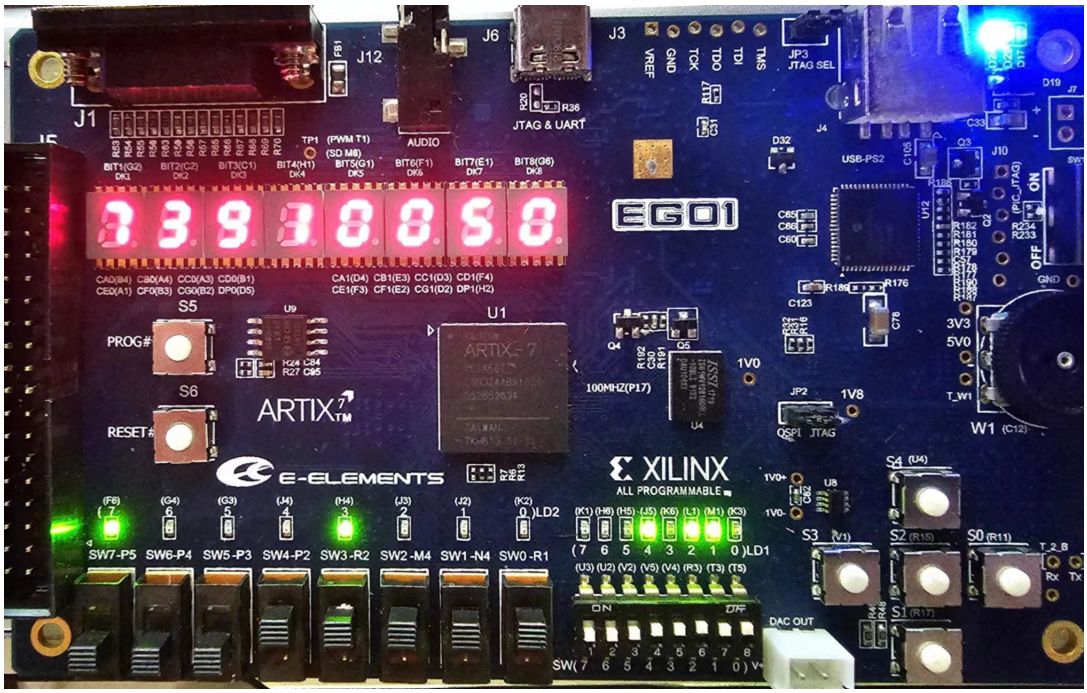

Figure 13: 极值搜索程序板级验证

极值搜索程序版本 2 的板级验证结果如图所示。其中，7，6 两位显示的为我的学号 521030910073 的末两位 73，5，4 两位显示的是最大值 91，3，2 两位显示的是最小值 0。1，0 位显示了执行所用时钟周期数十进制下的末两位。根据 ripes 上的仿真结果，一共需要 75 个周期，2*75=150，与板上显示的 50 相符。

附录

Listing 1: 问题 1: 修改后的 coe 文件

```
memory_initialization_radix=16;
memory_initialization_vector=00000537
00056213
00100c93
00200d13
00300d93
00400e13
01922023
01a22223
01b22423
01c22623
00400293
074000ef
00c22023
00022983
40c98933
00300293
 fff28293
 fff2e913
55594913
 fff00993
 fff9fa13
013a6833
013a4933
010a78b3
00028463
 fddff06f
 fff00293
00f29913
```

```
01091913
41095913
00f95913
08804613
08404693
08004713
00062a03
01462023
0126a023
00072b03
01672023
0000006f
00000933
00022983
00420213
01390933
 fff28293
fe0298e3
00091613
00008067
;
```