

浙江大学

本科实验报告

课程名称: 计算机组成与设计

姓名: 张晋恺

学院: 竺可桢学院

系: 所在系

专业: 计算机科学与技术

学号: 3230102400

指导教师: 刘海风

2024 年 10 月 21 日

浙江大学实验报告

课程名称: 计算机组成与设计 实验类型: 综合

实验项目名称: IP 核运用与连线

学生姓名: 张晋恺 专业: 计算机科学与技术 学号: 3230102400

同组学生姓名: _____ 指导老师: 刘海风

实验地点: 东 4-512 实验日期: 2024 年 10 月 10 日

一、操作方法与实验步骤

导入 IP 核

首先按照要求将附件中的 IP 核导入到 Vivado 中。

1. 以 IP 工程提供的模块采用 IP Catalog 添加调用 (生成 ROM 和 RAM)。
2. 以 EDF (.v) 文件提供的模块采用直接添加源文件的方式调用。
3. VGA 模块和七段数码管 Seg_7 模块采用直接添加源文件的方式调用。

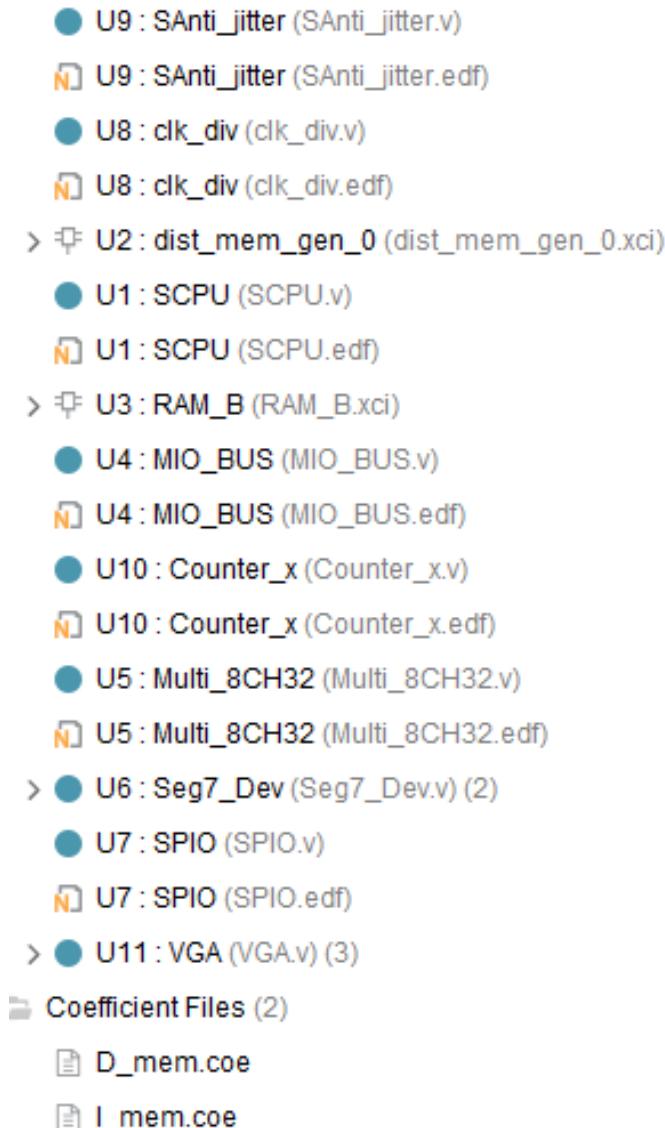


图 1: 导入 IP 核

根据电路图设计顶层模块

CSSTE.v 文件代码如下

```

1 `timescale 1ns / 1ps
2 module CSSTE(
3     input          clk_100mhz,
4     input          RSTN,
5     input [3:0]    BTN_y,
6     input [15:0]   SW,
```

```

7      output [3:0] Blue,
8      output [3:0] Green,
9      output [3:0] Red,
10     output      HSYNC,
11     output      VSYNC,
12     output [15:0] LED_out,
13     output [7:0] AN,
14     output [7:0] segment
15   );
16
17 //U9
18 wire [3:0] U9_BTN_OK;
19 wire [15:0] U9_SW_OK;
20 wire U9_rst;
21 SAnti_jitter U9(
22   .clk(clk_100mhz),
23   .RSTN(RSTN),
24   .Key_y(BTN_y),
25   .SW(SW),
26   .BTN_OK(U9_BTN_OK),
27   .SW_OK(U9_SW_OK),
28   .rst(U9_rst)
29 );
30 //
31
32 //U8
33 wire [31:0] U8_clk_div;
34 wire U8_Clk_CPU;
35 clk_div U8(
36   .clk(clk_100mhz),
37   .rst(U9_rst),
38   .SW2(U9_SW_OK[2]),
39   .SW8(U9_SW_OK[8]),
40   .STEP(U9_SW_OK[10]),
41   .clkdiv(U8_clk_div),
42   .Clk_CPU(U8_Clk_CPU)
43 );
44 //
45
46 //U2
47 wire [31:0] U2_spo;
48 dist_mem_gen_0 U2(
49   .a(U1_PC_out[11:2]),
50   .spo(U2_spo)
51 );

```

```

52    //
53
54    //U1
55    wire U1_MemRW;
56    wire [31:0] U1_Addr_out;
57    wire [31:0] U1_Data_out;
58    wire [31:0] U1_PC_out;
59    SCPU U1(
60        .clk(U8_Clk_CPU),
61        .rst(U9_rst),
62        .Data_in(U4_Cpu_data4bus),
63        .inst_in(U2_spo),
64        .MemRW(U1_MemRW),
65        .Addr_out(U1_Addr_out),
66        .Data_out(U1_Data_out),
67        .PC_out(U1_PC_out)
68    );
69    //
70
71    //U3
72    wire [31:0] U3_douta;
73    RAM_B U3(
74        .clka(~clk_100mhz),
75        .wea(U4_data_ram_we),
76        .addr(U4_ram_addr),
77        .dina(U4_ram_data_in),
78        .douta(U3_douta)
79    );
80    //
81
82    //U4
83    wire [31:0] U4_Cpu_data4bus,
84                U4_ram_data_in;
85    wire [9:0] U4_ram_addr;
86    wire U4_data_ram_we,
87          U4_GPIOf00000000_we,
88          U4_GPIOe00000000_we,
89          U4_counter_we;
90    wire [31:0] U4_Peripheral_in;
91    MIO_BUS U4(
92        .clk(clk_100mhz),
93        .rst(U9_rst),
94        .BTN(U9_BTN_OK),
95        .SW(U9_SW_OK),
96        .mem_w(U1_MemRW),

```

```

97     .Cpu_data2bus(U1_Data_out),
98     .addr_bus(U1_Addr_out),
99     .ram_data_out(U3_douta),
100    .led_out(U7_LED_out),
101    .counter_out(U10_counter_out),
102    .counter0_out(U10_counter0_OUT),
103    .counter1_out(U10_counter1_OUT),
104    .counter2_out(U10_counter2_OUT),
105    .Cpu_data4bus(U4_Cpu_data4bus),
106    .ram_data_in(U4_ram_data_in),
107    .ram_addr(U4_ram_addr),
108    .data_ram_we(U4_data_ram_we),
109    .GPIOf00000000_we(U4_GPIOf00000000_we),
110    .GPIOe00000000_we(U4_GPIOe00000000_we),
111    .counter_we(U4_counter_we),
112    .Peripheral_in(U4_Peripheral_in)
113  );
114  //
115
116 //U10
117 wire U10_counter0_OUT;
118 wire U10_counter1_OUT;
119 wire U10_counter2_OUT;
120 wire [31:0] U10_counter_out;
121 Counter_x U10(
122     .clk(~U8_Clk_CPU),
123     .rst(U9_rst),
124     .clk0(U8_clk_div[6]),
125     .clk1(U8_clk_div[9]),
126     .clk2(U8_clk_div[11]),
127     .counter_we(U4_counter_we),
128     .counter_val(U4_Peripheral_in),
129     .counter_ch(U7_counter_set),
130     .counter0_OUT(U10_counter0_OUT),
131     .counter1_OUT(U10_counter1_OUT),
132     .counter2_OUT(U10_counter2_OUT),
133     .counter_out(U10_counter_out)
134  );
135  //
136
137 //U5
138 wire [7:0] U5_point_out,
139             U5_LE_out;
140 wire [31:0] U5_Displ_num;
141 Multi_8CH32 U5(

```

```

142     .clk(~U8_Clk_CPU),
143     .rst(U9_rst),
144     .EN(U4_GPIOe00000000_we),
145     .Test(U9_SW_OK[7:5]),
146     .point_in({U8_clk_div[31:0],U8_clk_div[31:0]}),
147     .LES(64'b0),
148     .Data0(U4_Peripheral_in),
149     .data1({2'b0,U1_PC_out[31:2]}),
150     .data2(U2_spo),
151     .data3(U10_counter_out),
152     .data4(U1_Addr_out),
153     .data5(U1_Data_out),
154     .data6(U4_Cpu_data4bus),
155     .data7(U1_PC_out),
156     .point_out(U5_point_out),
157     .LE_out(U5_LE_out),
158     .Disp_num(U5_Dispatcher)
159   );
160   //
161
162   //U6
163   Seg7_Dev U6(
164     .disp_num(U5_Dispatcher),
165     .point(U5_point_out),
166     .les(U5_LE_out),
167     .scan(U8_clk_div[18:16]),
168     .AN(AN),
169     .segment(segment)
170   );
171   //
172
173   //U7
174   wire [1:0] U7_counter_set;
175   wire [15:0] U7_LED_out;
176   assign LED_out=U7_LED_out;
177   SPIO U7(
178     .clk(~U8_Clk_CPU),
179     .rst(U9_rst),
180     .Start(U8_clk_div[20]),
181     .EN(U4_GPIOf00000000_we),
182     .P_Data(U4_Peripheral_in),
183     .counter_set(U7_counter_set),
184     .LED_out(U7_LED_out)
185   );
186   //

```

```

187
188 //U11
189 VGA U11(
190     .clk_25m(U8_clk_div[1]),
191     .clk_100m(clk_100mhz),
192     .rst(U9_rst),
193     .pc(U1_PC_out),
194     .inst(U2_spo),
195     .alu_res(U1_Addr_out),
196     .mem_wen(U1_MemRW),
197     .dmem_o_data(U3_douta),
198     .dmem_i_data(U4_ram_data_in),
199     .dmem_addr(U1_Addr_out),
200     .hs(HSYNC),
201     .vs(VSYNC),
202     .vga_r(Red),
203     .vga_g(Green),
204     .vga_b(Blue)
205 );
206 //
207 endmodule
208

```

处理 VGA

1. 修改文件路径

```

1 initial $readmemh("K://Computer_organization//Lab2-attachment//Lab2_"
2   ↵ -attachment//0Exp02-IP2SOC//vga_debugger.mem",
2   ↵ display_data);
2 initial $readmemh("K://Computer_organization//Lab2-attachment//Lab2_"
2   ↵ -attachment//0Exp02-IP2SOC//font_8x16.mem",
2   ↵ fonts_data);

```

2. 修改 VGA.v 中 module VGA 的端口描述（增加若干 debug 信号输入），将增加的输入接到 VGA 模块的 vga_debugger 实例；

```

1 module VGA(
2   input wire clk_25m,

```

```

3  input wire clk_100m,
4  input wire rst,
5  input wire [31:0] pc,
6  input wire [31:0] inst,
7  input wire [31:0] alu_res,
8  input wire mem_wen,
9  input wire [31:0] dmem_o_data,
10 input wire [31:0] dmem_i_data,
11 input wire [31:0] dmem_addr,
12 //add
13 input wire [4:0] rs1,
14 input wire [31:0] rs1_val,
15 input wire [4:0] rs2,
16 input wire [31:0] rs2_val,
17 input wire [4:0] rd,
18 input wire [31:0] reg_i_data,
19 input wire reg_wen,
20 input wire is_imm,
21 input wire is_auipc,
22 input wire is_lui,
23 input wire [31:0] imm,
24 input wire [31:0] a_val,
25 input wire [31:0] b_val,
26 input wire [3:0] alu_ctrl,
27 input wire [2:0] cmp_ctrl,
28 input wire cmp_res,
29 input wire is_branch,
30 input wire is_jal,
31 input wire is_jalr,
32 input wire do_branch,
33 input wire [31:0] pc_branch,
34 input wire mem_ren,
35 input wire csr_wen,
36 input wire [11:0] csr_ind,
37 input wire [1:0] csr_ctrl,
38 input wire [31:0] csr_r_data,
39 input wire [31:0] x0,
40 input wire [31:0] ra,
41 input wire [31:0] sp,
42 input wire [31:0] gp,
43 input wire [31:0] tp,
44 input wire [31:0] t0,
45 input wire [31:0] t1,
46 input wire [31:0] t2,
47 input wire [31:0] s0,

```

```

48  input wire [31:0] s1,
49  input wire [31:0] a0,
50  input wire [31:0] a1,
51  input wire [31:0] a2,
52  input wire [31:0] a3,
53  input wire [31:0] a4,
54  input wire [31:0] a5,
55  input wire [31:0] a6,
56  input wire [31:0] a7,
57  input wire [31:0] s2,
58  input wire [31:0] s3,
59  input wire [31:0] s4,
60  input wire [31:0] s5,
61  input wire [31:0] s6,
62  input wire [31:0] s7,
63  input wire [31:0] s8,
64  input wire [31:0] s9,
65  input wire [31:0] s10,
66  input wire [31:0] s11,
67  input wire [31:0] t3,
68  input wire [31:0] t4,
69  input wire [31:0] t5,
70  input wire [31:0] t6,
71  input wire [31:0] mstatus_o,
72  input wire [31:0] mcause_o,
73  input wire [31:0] mepc_o,
74  input wire [31:0] mtval_o,
75  input wire [31:0] mtvec_o,
76  input wire [31:0] mie_o,
77  input wire [31:0] mip_o,
78  //
79  output wire hs,
80  output wire vs,
81  output wire [3:0] vga_r,
82  output wire [3:0] vga_g,
83  output wire [3:0] vga_b
84 );
85 VgaDebugger vga_debugger(
86     .clk          (clk_100m      ),
87     .display_wen  (display_wen   ),
88     .display_w_addr(display_w_addr),
89     .display_w_data(display_w_data),
90     .pc           (pc          ),
91     .inst         (inst         ),
92     .rs1          (rs1          ),

```

```

93     .rs1_val      (rs1_val          ),
94     .rs2         (rs2          ),
95     .rs2_val      (rs2_val          ),
96     .rd          (rd          ),
97     .reg_i_data  (reg_i_data        ),
98     .reg_wen     (reg_wen          ),
99     .is_imm      (is_imm          ),
100    .is_auipc   (is_auipc         ),
101    .is_lui      (is_lui          ),
102    .imm         (imm          ),
103    .a_val       (a_val          ),
104    .b_val       (b_val),
105    .alu_ctrl    (alu_ctrl          ),
106    .cmp_ctrl    (cmp_ctrl          ),
107    .alu_res     (alu_res          ),
108    .cmp_res     (cmp_res          ),
109    .is_branch   (is_branch         ),
110    .is_jal      (is_jal          ),
111    .is_jalr     (is_jalr         ),
112    .do_branch   (do_branch         ),
113    .pc_branch   (pc_branch         ),
114    .mem_wen    (mem_wen          ),
115    .mem_ren    (mem_ren          ),
116    .dmem_o_data (dmem_o_data        ),
117    .dmem_i_data (dmem_i_data        ),
118    .dmem_addr   (dmem_addr          ),
119    .csr_wen     (csr_wen          ),
120    .csr_ind     (csr_ind          ),
121    .csr_ctrl    (csr_ctrl          ),
122    .csr_r_data  (csr_r_data         ),
123    .x0          (x0          ),
124    .ra          (ra          ),
125    .sp          (sp          ),
126    .gp          (gp          ),
127    .tp          (tp          ),
128    .t0          (t0          ),
129    .t1          (t1          ),
130    .t2          (t2          ),
131    .s0          (s0          ),
132    .s1          (s1          ),
133    .a0          (a0          ),
134    .a1          (a1          ),
135    .a2          (a2          ),
136    .a3          (a3          ),
137    .a4          (a4          ),

```

```

138     .a5          (a5          ),
139     .a6          (a6          ),
140     .a7          (a7          ),
141     .s2          (s2          ),
142     .s3          (s3          ),
143     .s4          (s4          ),
144     .s5          (s5          ),
145     .s6          (s6          ),
146     .s7          (s7          ),
147     .s8          (s8          ),
148     .s9          (s9          ),
149     .s10         (s10         ),
150    .s11         (s11         ),
151    .t3          (t3          ),
152    .t4          (t4          ),
153    .t5          (t5          ),
154    .t6          (t6          ),
155    .mstatus_o   (mstatus_o   ),
156    .mcause_o    (mcause_o    ),
157    .mepc_o      (mepc_o      ),
158    .mtval_o     (mtval_o     ),
159    .mtvec_o     (mtvec_o     ),
160    .mie_o       (mie_o       ),
161    .mip_o       (mip_o       )
162  );

```

程序实现的功能

观察 I_mem.pdf

| | PC | Machine Code | Basic Code | Original Code | Result |
|----|------|--------------|---------------|--------------------|---------------|
| 1 | 0x0 | 0x00100093 | addi x1 x0 1 | loop: addi x1,x0,1 | #x1=00000001 |
| 2 | 0x4 | 0x00102133 | slt x2 x0 x1 | slt x2,x0,x1 | #x2=00000001 |
| 3 | 0x8 | 0x002101B3 | add x3 x2 x2 | add x3,x2,x2 | #x3=00000002 |
| 4 | 0xc | 0x00218233 | add x4 x3 x2 | add x4,x3,x2 | #x4=00000003 |
| 5 | 0x10 | 0x003202B3 | add x5 x4 x3 | add x5,x4,x3 | #x5=00000005 |
| 6 | 0x14 | 0x00428333 | add x6 x5 x4 | add x6,x5,x4 | #x6=00000008 |
| 7 | 0x18 | 0x005303B3 | add x7 x6 x5 | add x7,x6,x5 | #x7=0000000d |
| 8 | 0x1c | 0x00638433 | add x8 x7 x6 | add x8,x7,x6 | #x8=00000015 |
| 9 | 0x20 | 0x007404B3 | add x9 x8 x7 | add x9,x8,x7 | #x9=00000022 |
| 10 | 0x24 | 0x00848533 | add x10 x9 x8 | add x10,x9,x8 | #x10=00000037 |

```

12 0x28 0x009505B3 add x11 x10 x9 add x11,x10,x9 #x11=00000059
13 0x2c 0x00A58633 add x12 x11 x10 add x12,x11,x10 #x12=00000090
14 0x30 0x00B606B3 add x13 x12 x11 add x13,x12,x11 #x13=000000E9
15 0x34 0x00C68733 add x14 x13 x12 add x14,x13,x12 #x14=00000179
16 0x38 0x00D707B3 add x15 x14 x13 add x15,x14,x13 #x15=00000262
17 0x3c 0x00E78833 add x16 x15 x14 add x16,x15,x14 #x16=000003DB
18 0x40 0x00F808B3 add x17 x16 x15 add x17,x16,x15 #x17=000006D3
19 0x44 0x01088933 add x18 x17 x16 add x18,x17,x16 #x18=00000A18
20 0x48 0x011909B3 add x19 x18 x17 add x19,x18,x17 #x19=000010EB
21 0x4c 0x01298A33 add x20 x19 x18 add x20,x19,x18 #x20=00001B03
22 0x50 0x013A0AB3 add x21 x20 x19 add x21,x20,x19 #x21=00003bEE
23 0x54 0x014A8B33 add x22 x21 x20 add x22,x21,x20 #x22=000046F1
24 0x58 0x015B0BB3 add x23 x22 x21 add x23,x22,x21 #x23=000080DF
25 0x5c 0x016B8C33 add x24 x23 x22 add x24,x23,x22 #x24=0000C9D0
26 0x60 0x017C0CB3 add x25 x24 x23 add x25,x24,x23 #x25=00014AAF
27 0x64 0x018C8D33 add x26 x25 x24 add x26,x25,x24 #x26=0001947F
28 0x68 0x019D0DB3 add x27 x26 x25 add x27,x26,x25 #x27=0012DF2E
29 0x6c 0x01AD8E33 add x28 x27 x26 add x28,x27,x26 #x28=001473AD
30 0x70 0x01BE0EB3 add x29 x28 x27 add x29,x28,x27 #x29=002752DB
31 0x74 0x01CE8F33 add x30 x29 x28 add x30,x29,x28 #x30=003BC688
32 0x78 0x01DF0FB3 add x31 x30 x29 add x31,x30,x29 #x31=00621963
33 0x7c 0xF80002E3 beq x0 x0 -124 beq x0,x0,loop #pc---->loop
34

```

可以看到这个程序实现的是一个计算斐波那契数列的功能；

- PC: 程序计数器 (Program Counter), 表示每条指令所在的内存地址。
- Machine Code: 机器码, 以 16 进制表示的每条指令。
- Basic Code: 基本汇编指令, 展示 RISC-V 的操作, 包括立即数加法 (addi) 和寄存器加法 (add)。
- Original Code: 是 Basic Code 的等效解释, 显示了指令的上下文 (如循环结构)。
- Result: 每条指令执行后目标寄存器的值 (以 # 号后的 16 进制数表示)。
- 起始指令 (PC = 0x0): addi x1 x0 1, 将立即数 1 加到寄存器 x0, 结果存入寄存器 x1, x0 为零寄存器, 总是为 0。因此 x1 的结果为 1。
- 第二条指令 (PC = 0x4): slt x2 x0 x1, 比较 x0 和 x1, 由于 x0=0, x1=1, 因此 x2 会被置为 1。
- 后续指令: 从 x3 到 x31, 每次将两个寄存器相加, 结果存入另一个寄存器。例如, add x3 x2 x1 计算的是 x2 + x1 的结果, 依次类推, 得到新的斐

波那契数

- 最后一条指令 ($PC = 0x7c$) 是 $beq\ x0\ x0\ -124$, 即如果 $x0$ 等于 $x0$ (这总是成立), 程序跳转回 $PC = 0x0$, 形成一个无限循环。

二、实验结果与分析

下板

| SW 组合 | 功能描述 |
|---------------|---------------------------------|
| SW[8]SW[2]=00 | CPU 全速时钟 100MHz |
| SW[8]SW[2]=01 | CPU 自动单步时钟 |
| SW[8]SW[2]=1X | CPU 手动单步时钟 |
| SW[10] | 手动时钟模式，拨一下 SW[10] 增加一个周期并执行一条指令 |
| SW[7:5]=001 | 显示下一条要执行的指令地址 PC_out[31:2] |
| SW[7:5]=010 | 显示正在执行的指令 Inst_in |
| SW[7:5]=100 | 显示数据存储地址 addr_bus |
| SW[7:5]=101 | 显示数据输出 Cpu_data2bus |
| SW[7:5]=110 | 显示数据输入 Cpu_data4bus |
| SW[7:5]=111 | 显示指令字地址 PC_out |

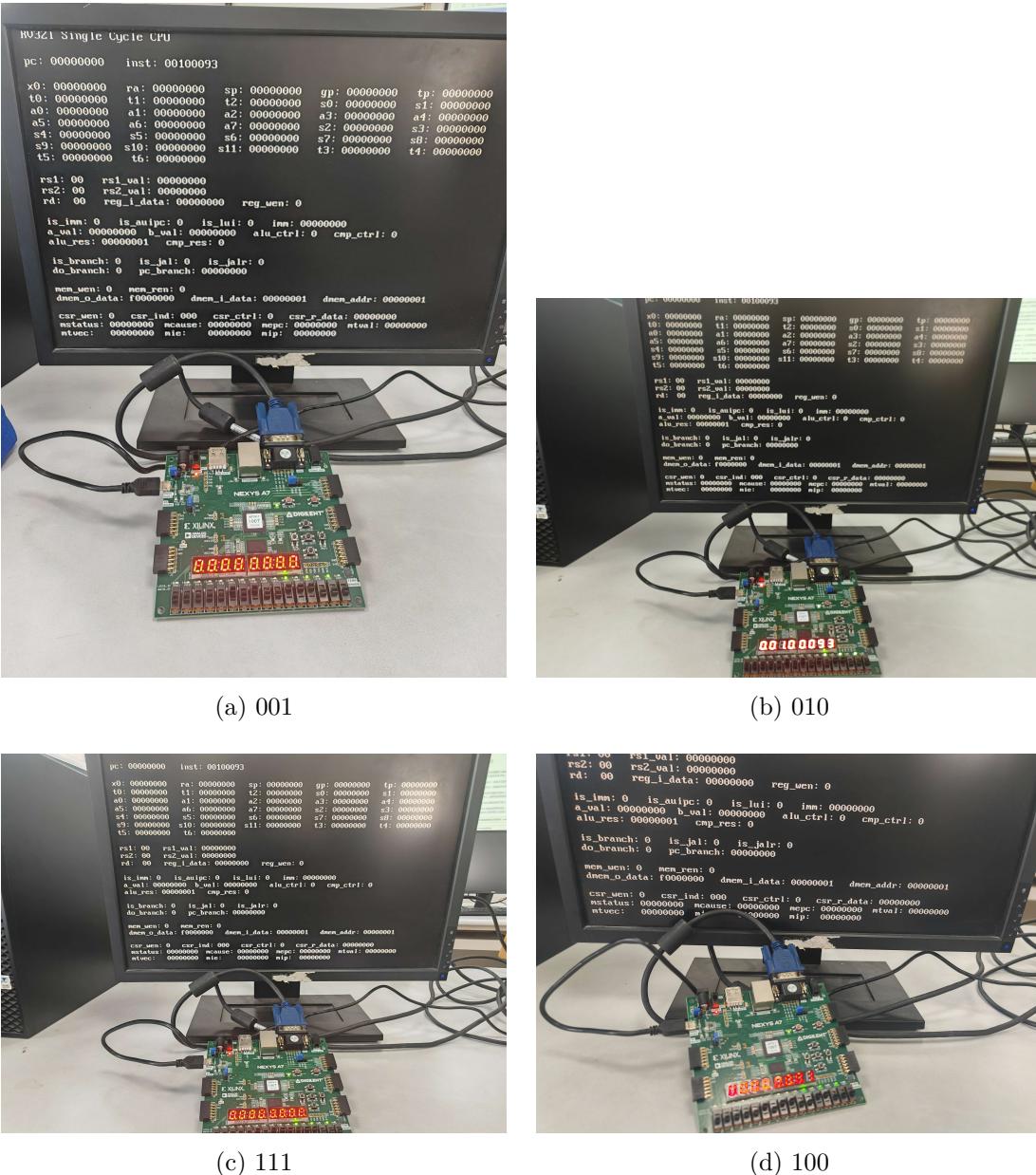


图 2: 第一条指令的执行情况

需要注意的是，当开关波动到 001 时，此时下一个指令是 0x4，在二进制表示中最后两位是 10，所以显示的字地址在舍去最后两位后只剩下全 0 了；其他的情况可以看到开关拨动到对应的位置后，显示的数据是正确的，与 VGA 上的显示一致，也与 Pdf 文件中显示的一致。

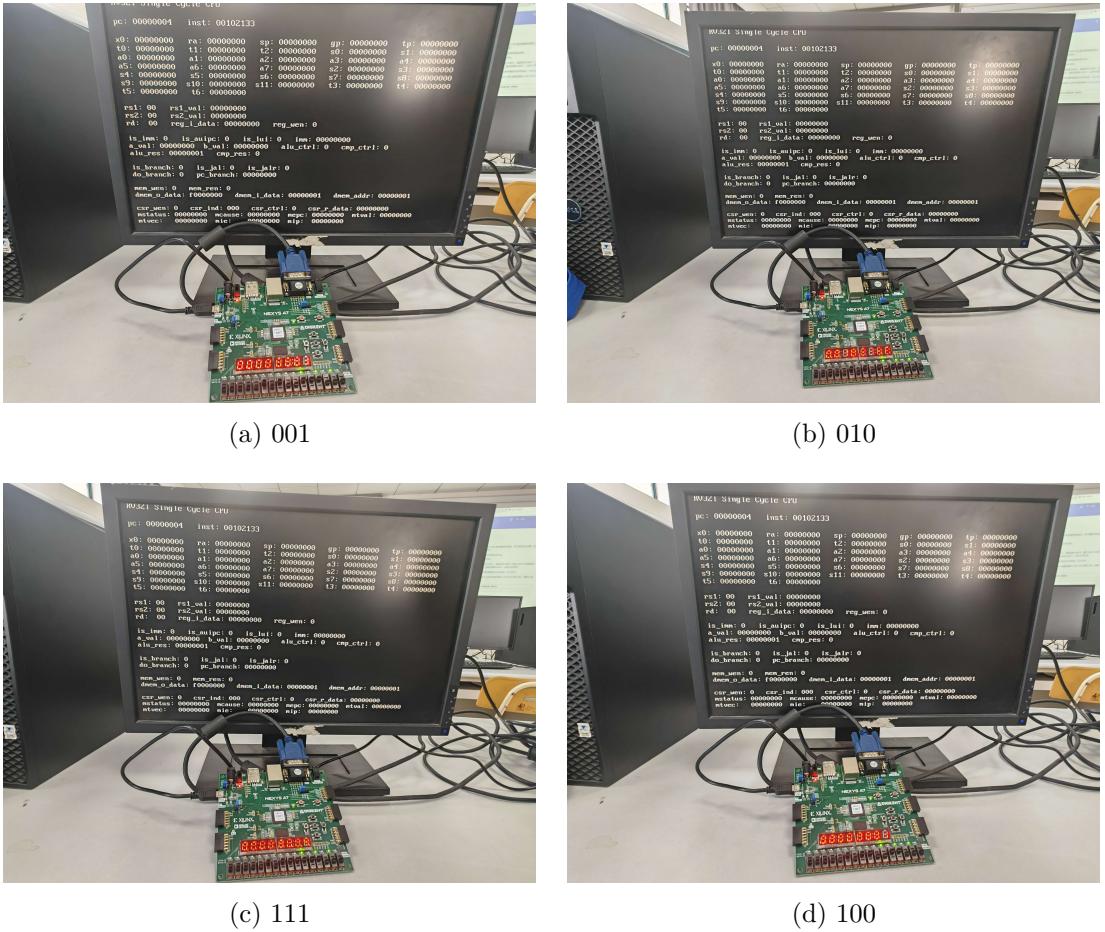


图 3: 第二条指令的执行情况

当开关波动到 001 时，此时下一个指令是 0x8，在二进制表示中是两位是 100，所以显示的字地址在舍去最后两位后只剩下最低为的 1 了；其他的情况可以看到开关拨动到对应的位置后，显示的数据是正确的，与 VGA 上的显示一致，也与 Pdf 文件中显示的一致。

其它数据的正确性

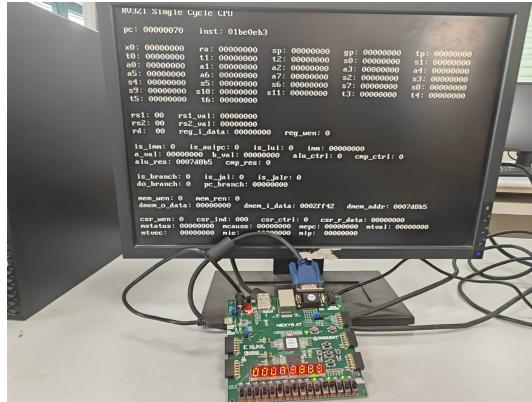


图 4: 循环指令的执行情况

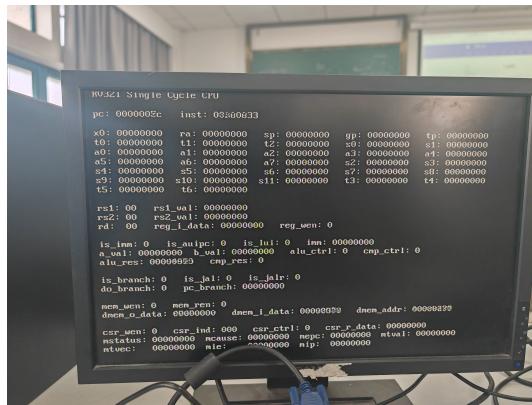


图 5: 全速时钟的执行情况

此时由于时钟很快，VGA 上的显示无法看清，闪烁得很快；



图 6: 显示数据输出的执行情况

可以看到此时的显示数据与 dmem_i_data 一致；观察我们的代码

.dmem_i_data(U4_ram_data_in), , 是与 Cpu_data2bus 连在一起的

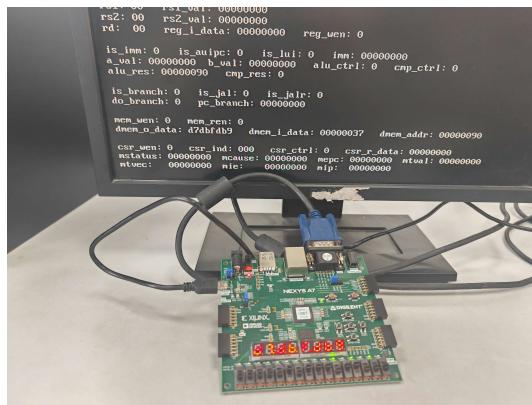


图 7: 显示数据输入的执行情况

可以看到此时的显示数据与 dmem_o_data 一致；观察我们的代码

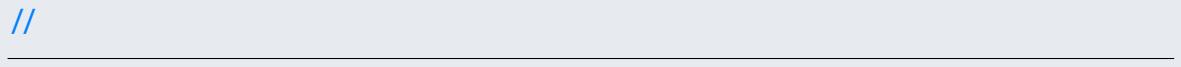
.dmem_o_data(U3_douta), , 是 RAM 的输出

三、讨论与实验心得

心得

本次实验的难度与工作量主要在于连线；连线的时候，我主要关注实例化的时候，模块的输入是唯一的，模块的输出可以接到其它很多模块上面，所以每次实例化一个模块之前；我都会先声明它的输出 Wire，并在前面加上它的模块名字；例如 Ux_clk_out；这样在以后的输入需要用到这个信号的时候，我在 CSSTE.pdf 图中发现这个输入来自 Ux 模块；就可以只输入 Ux，然后 Vscode 编辑器就会自动跳出可选的信号名字，直接选用即可；这样可以减少很多的输入错误，提高了工作效率。例如

```
1  wire [3:0] U9_BTN_OK;
2  wire [15:0] U9_SW_OK;
3  wire U9_rst;
4  SAnti_jitter U9(
5      .clk(clk_100mhz),
6      .RSTN(RSTN),
7      .Key_y(BTN_y),
8      .SW(SW),
9      .BTN_OK(U9_BTN_OK),
10     .SW_OK(U9_SW_OK),
11     .rst(U9_rst)
12 );
13 //
14
15 //U8
16 wire [31:0] U8_clk_div;
17 wire U8_Clk_CPU;
18 clk_div U8(
19     .clk(clk_100mhz),
20     .rst(U9_rst),
21     .SW2(U9_SW_OK[2]),
22     .SW8(U9_SW_OK[8]),
23     .STEP(U9_SW_OK[10]),
24     .clkdiv(U8_clk_div),
25     .Clk_CPU(U8_Clk_CPU)
26 );
```



还有一个需要关注的点是 VGA 的文件路径的修改，我看到很多同学对于 Win 系统的文件路径写法需要一个个尝试，但是其实如果用的是 VScode 编辑器，可以直接右键文件然后选择 copy path，然后粘贴到代码中即可；不需要遍历各种斜杠的组合，这样可以减少很多的时间浪费。

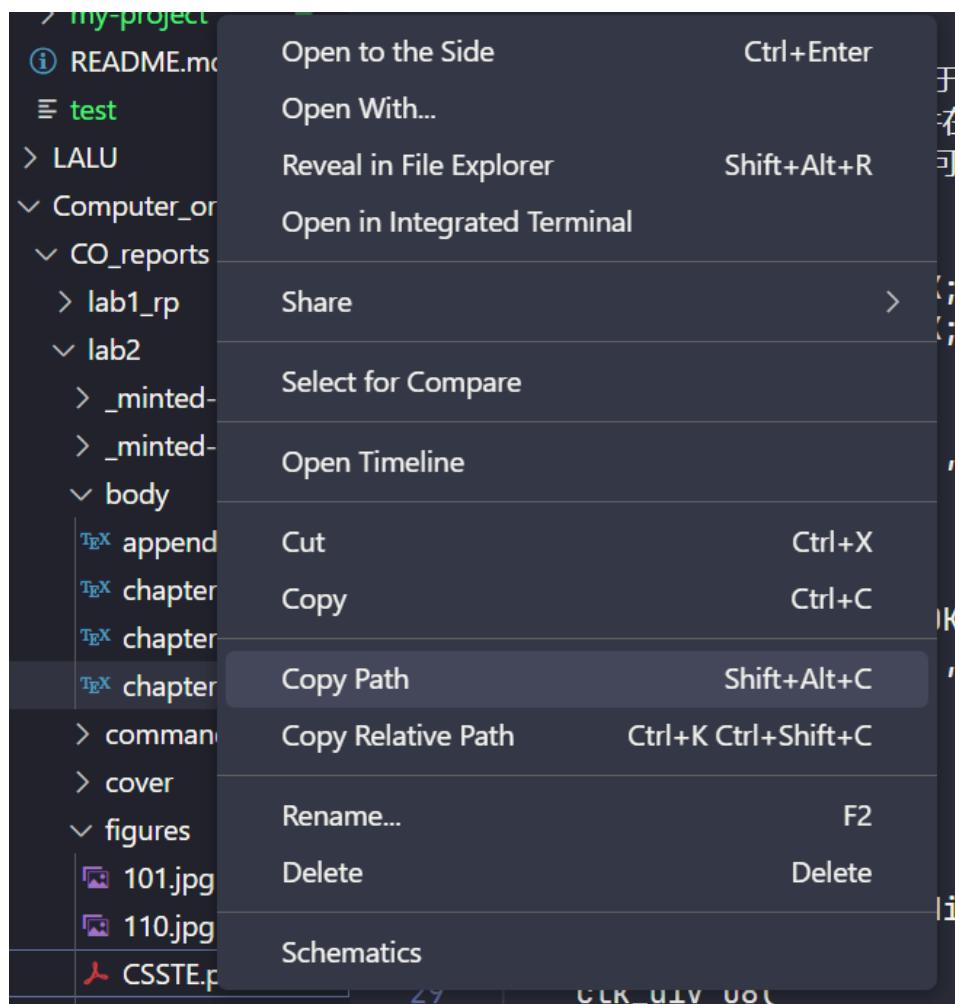


图 8: VScode copy path

对于 VGA 的端口修改，我并没有一个个修改，而是将 VGAdisplay 的端口复制到了 VGA 的模块中，然后注释掉；让 copilot 读了之后，就能让它自动补全，然后检查一下稍加修改就可以了，完全无痛；

思考题

本次实验没有思考题。以下是 CSSTE 文件。

