

**本科实验报告**

|  |  |
| --- | --- |
| 课程名称： | 计算机组成 |
| 姓 名： | TANG ANNA YONGQI |
| 学 院： | 计算机科学与技术学院 |
| 专 业： | 计算机科学与技术（中加班）留学生 |
| 学 号： | 3180300155 |
| 生活照： |  |
| 指导教师： | 刘海风，洪奇军 |

2020年 5月17日

Lab 9 – Multicycle CPU Design

**Name:** Anna Yongqi Tang **ID:** 3180300155 **Major:** 计算机科学与技术（中加班）留学生

**Course:** Computer Organization

**Date:** 2020-05-17 **Instructor:** 洪奇军

1. **Method and Experimental Steps**

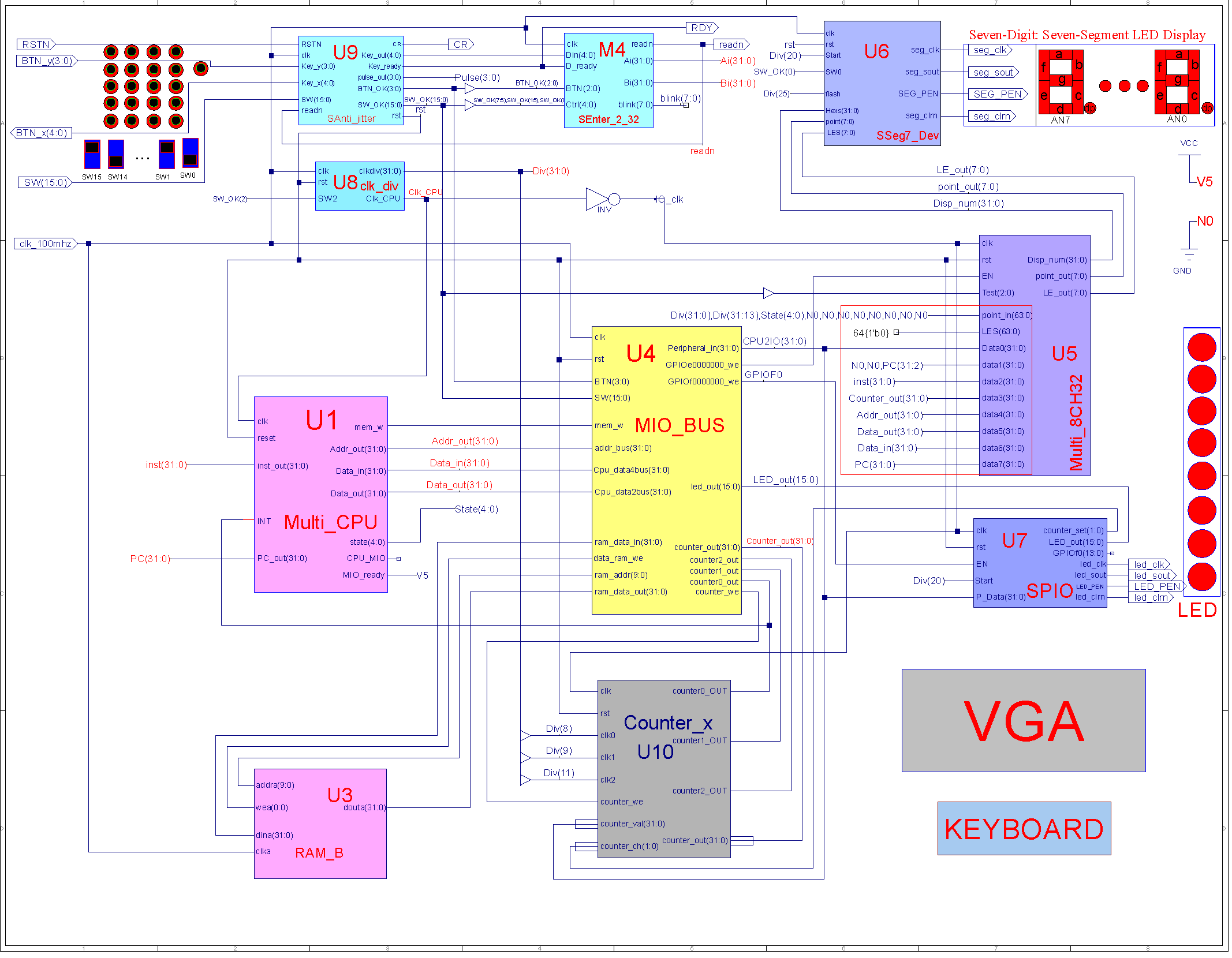


Figure 1 - topMod.sch

In lab 9, I designed a multicycle CPU using modules from previous experiments and ngc cores. The top level (pictured above) was designed off of lab 3, but a HDL implementation was used instead of a schematic. The CPU module is replaced with a multicycle CPU, and only a RAM was used. Supposedly, we were to implement a new coe file in the ram. This lab’s purpose was to better understand the structure of a multicycle implementation and explore the different module’s functions. I believe that within the next coming experiments, we will rebuild some of the modules from scratch instead of using ngc cores. The .ucf for this program came from the provided courseware, and is linked to topMod.v. Synthesis had minimal warnings, and implementation was successful. A programmable file has been generated and is ready for testing on the SWORD board. The above diagram was retrieved from the lab PowerPoint. No schematic has been drawn for this experiment.

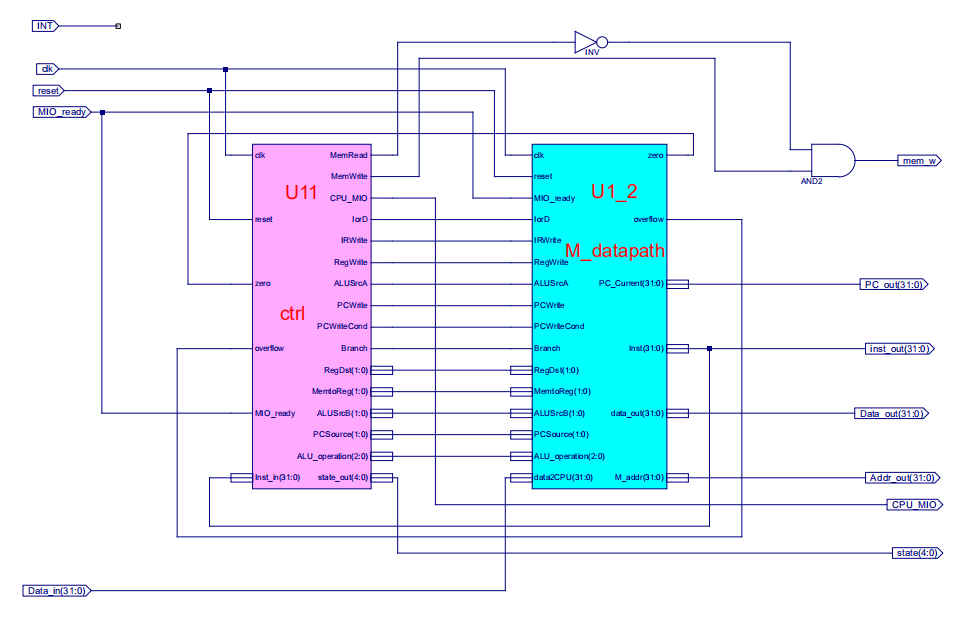


Figure 2 – Multi\_CPU.sch

The multicycle CPU is made up of two modules, the datapath and the controller. All the code and schematics here were retrieved from the provided courseware. We use a RISC implementation, where memory is accessed through specific instructions rather than as a part of most instructions. The datapath processes the instruction and performs memory accesses based on the what is provided by the PC. It also takes care of the different control signals that were explored in chapter 5. The control module tells the datapath what to do to by asserting and deasserting the control signals. The above diagram was also retrieved from the provided courseware.



Figure 3 – 3180300155\_TANGANNAYONGQI\_09

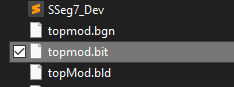
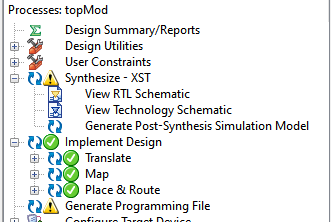


Figure 4 - .bit file generation Figure 5 - .bit file generated in directory

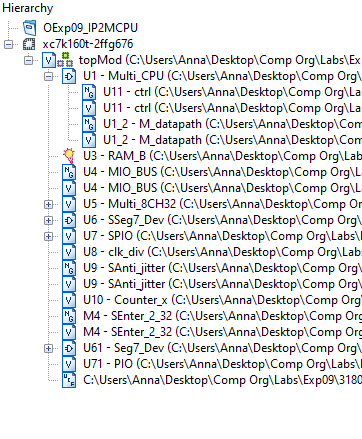


Figure 6 - file hierarchy

1. **Simulations and Observations**

This lab requires a MIPS program (demo.coe) to be tested on the multicyle CPU. The demo will be performed later in a future exercise. Since all modules were implemented with ngc cores, no simulation was performed for this lab.

**3. Conclusion**

This week’s lab was just a brief overview and introduction to a multicycle CPU implementation. It is quite similar in structure to the single cycle implementation, but a lot more efficient in runtime and hardware. I look forward to simulating the datapath and comparing the results to what was produced in a single cycle implementation. I am also looking forward to rebuilding some of the modules used from scratch and examining the differences in hardware.

1. **Source Code**

All modules and components were either retrieved from previous labs or directly taken from the given files.

*Top Module - topMod.v*

module topMod(

input RSTN,

input [3:0] BTN\_y,

input [4:0] BTN\_x,

input [15:0] SW,

input clk\_100mhz,

output CR,

output RDY,

output readn,

output seg\_clk,

output seg\_sout,

output seg\_clrn,

output SEG\_PEN,

output led\_clk,

output led\_sout,

output LED\_PEN,

output led\_clrn,

output [7:0] SEGMENT,

output [3:0] AN,

output [7:0] LED,

output Buzzer

);

wire V5, N0;

assign V5 = 1'b1;

assign N0 = 1'b0;

assign Buzzer = 1'b1;

wire Clk\_CPU, mem\_w, data\_ram\_we, IO\_clk, GPIOE0, GPIOF0, counter0\_out, counter1\_out, counter2\_out, counter\_we;

wire[1:0] counter\_set;

wire[3:0] BTN\_OK, Pulse;

wire[4:0] Key\_out, state;

wire[7:0] point\_out, LE\_out, blink;

wire[9:0] ram\_addr;

wire[15:0] SW\_OK, LED\_out;

wire[31:0] inst, PC, Addr\_out, Data\_in, Data\_out, ram\_data\_in, ram\_data\_out, CPU2IO, Counter\_out, Div, Disp\_num, Ai, Bi;

assign IO\_clk = ~Clk\_CPU;

Multi\_CPU U1(

.clk(Clk\_CPU),

.reset(rst),

.inst\_out(inst),

.INT(counter0\_out),

.PC\_out(PC),

.mem\_w(mem\_w),

.Addr\_out(Addr\_out),

.Data\_in(Data\_in),

.Data\_out(Data\_out),

.state(state),

.CPU\_MIO(),

.MIO\_ready(V5)

);

RAM\_B U3(

.addra(ram\_addr),

.wea(data\_ram\_we),

.dina(ram\_data\_in),

.clka(clk\_100mhz),

.douta(ram\_data\_out)

);

MIO\_BUS U4(

.clk(clk\_100mhz),

.rst(rst),

.BTN(BTN\_OK),

.SW(SW\_OK),

.mem\_w(mem\_w),

.Cpu\_data2bus(Data\_out),

.addr\_bus(Addr\_out),

.ram\_data\_out(ram\_data\_out),

.led\_out(LED\_out),

.counter\_out(Counter\_out),

.counter0\_out(counter0\_out),

.counter1\_out(counter1\_out),

.counter2\_out(counter2\_out),

.Cpu\_data4bus(Data\_in),

.ram\_data\_in(ram\_data\_in),

.ram\_addr(ram\_addr),

.data\_ram\_we(data\_ram\_we),

.GPIOf0000000\_we(GPIOF0),

.GPIOe0000000\_we(GPIOE0),

.counter\_we(counter\_we),

.Peripheral\_in(CPU2IO)

);

Multi\_8CH32 U5(

.clk(IO\_clk),

.rst(rst),

.EN(GPIOE0),

.Test(SW\_OK[7:5]),

.point\_in({Div, Div[31:13], state, N0, N0, N0, N0, N0, N0, N0, N0}),

.LES(64'b0),

.Data0(CPU2IO),

.data1({N0,N0,PC[31:2]}),

.data2(inst),

.data3(Counter\_out),

.data4(Addr\_out),

.data5(Data\_out),

.data6(Data\_in),

.data7(PC),

.point\_out(point\_out),

.LE\_out(LE\_out),

.Disp\_num(Disp\_num)

);

SSeg7\_Dev U6(

.clk(clk\_100mhz),

.rst(rst),

.Start(Div[20]),

.SW0(SW\_OK[0]),

.flash(Div[25]),

.Hexs(Disp\_num),

.point(point\_out),

.LES(LE\_out),

.seg\_clk(seg\_clk),

.seg\_sout(seg\_sout),

.SEG\_PEN(SEG\_PEN),

.seg\_clrn(seg\_clrn)

);

SPIO U7(

.clk(IO\_clk),

.rst(rst),

.Start(Div[20]),

.EN(GPIOF0),

.GPIOf0(),

.P\_Data(CPU2IO),

.counter\_set(counter\_set),

.LED\_out(LED\_out),

.led\_clk(led\_clk),

.led\_sout(led\_sout),

.led\_clrn(led\_clrn),

.LED\_PEN(LED\_PEN)

);

clk\_div U8(

.clk(clk\_100mhz),

.rst(rst),

.SW2(SW\_OK[2]),

.clkdiv(Div),

.Clk\_CPU(Clk\_CPU)

);

SAnti\_jitter U9(

.clk(clk\_100mhz),

.RSTN(RSTN),

.readn(readn),

.Key\_y(BTN\_y),

.Key\_x(BTN\_x),

.SW(SW),

.Key\_out(Key\_out),

.Key\_ready(RDY),

.pulse\_out(Pulse),

.BTN\_OK(BTN\_OK),

.SW\_OK(SW\_OK),

.CR(CR),

.rst(rst)

);

Counter\_x U10(

.clk(IO\_clk),

.rst(rst),

.clk0(Div[8]),

.clk1(Div[9]),

.clk2(Div[10]),

.counter\_we(counter\_we),

.counter\_val(CPU2IO),

.counter\_ch(counter\_set),

.counter0\_OUT(counter0\_out),

.counter1\_OUT(counter1\_out),

.counter2\_OUT(counter2\_out),

.counter\_out(Counter\_out)

);

SEnter\_2\_32 M4(

.clk(clk\_100mhz),

.BTN(BTN\_OK[2:0]),

.Ctrl({SW\_OK[7:5],SW\_OK[15],SW\_OK[0]}),

.D\_ready(RDY),

.Din(Key\_out),

.readn(readn),

.Ai(Ai),

.Bi(Bi),

.blink(blink)

);

Seg7\_Dev U61(

.Scan({SW\_OK[1],Div[19:18]}),

.SW0(SW\_OK[0]),

.flash(Div[25]),

.Hexs(Disp\_num),

.point(point\_out),

.LES(LE\_out),

.SEGMENT(SEGMENT),

.AN(AN)

);

PIO U71(

.clk(IO\_clk),

.rst(rst),

.EN(GPIOF0),

.counter\_set(),

.GPIOf0(),

.PData\_in(CPU2IO),

.LED\_out(LED)

);

endmodule