



同濟大學  
TONGJI UNIVERSITY

# 计算机系统结构课程实验 总结报告

实验题目：动态流水线设计与性能定量分析

学号：2152809

姓名：曾崇然

指导教师：陆有军老师

日期：2023-12-30

## 一、实验环境部署与硬件配置说明

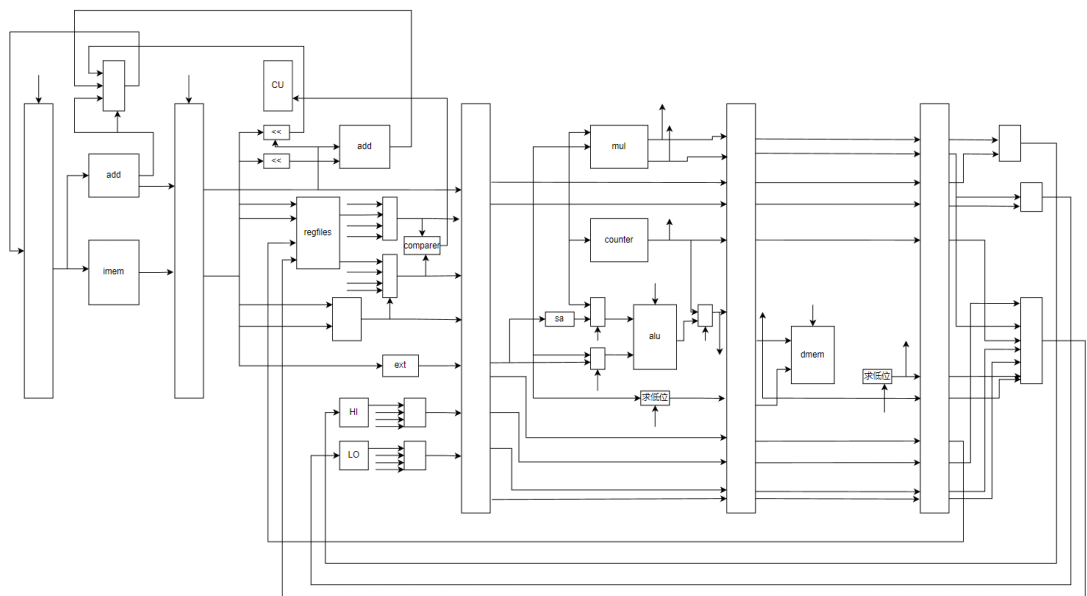
1. 实验环境部署：操作系统：windows11，开发工具：vivado，仿真工具：vivado 自带的仿真工具

2. 硬件配置说明：Xilinx FPGA 器件：Nexys DDR 开发板

## 二、实验的总体结构

实验要求完成至少 31 条 MIPS 指令的动态流水线 CPU 设计，并支持中断。在 CPU 运行验证程序的过程中，由按键或拨动开关产生一个暂停的中断，再次按键或拨动开关结束中断，继续运行后续的运算，并在数码管上动态显示运算值。

### 1、 动态流水线的总体结构



### 2、 指令选取

共选取 32 条 MIPS 指令，具体如下：

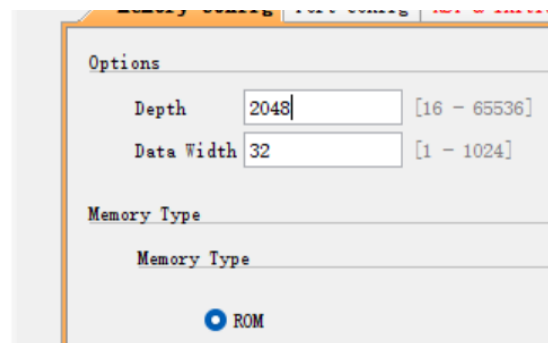
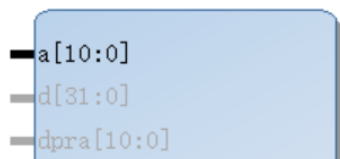
助记符	指令格式					
Bit	31-26	25-21	20-16	15-11	10-6	5-0
add	000000	rs	rt	rd	00000	100000
addu	000000	rs	rt	rd	00000	100001
sub	000000	rs	rt	rd	00000	100010
subu	000000	rs	rt	rd	00000	100011
and	000000	rs	rt	rd	00000	100100
or	000000	rs	rt	rd	00000	100101
xor	000000	rs	rt	rd	00000	100110
nor	000000	rs	rt	rd	00000	100111
slt	000000	rs	rt	rd	00000	101010
sltu	000000	rs	rt	rd	00000	101011
sll	000000	00000	rt	rd	shamt	000000
srl	000000	00000	rt	rd	shamt	000010
sra	000000	00000	rt	rd	shamt	000011
sllv	000000	rs	rt	rd	00000	000100
srlv	000000	rs	rt	rd	00000	000110
srav	000000	rs	rt	rd	00000	000111
jr	000000	rs	00000	00000	00000	001000
addi	001000	rs	rt	immediate		
addiu	001001	rs	rt	immediate		

andi	001100	rs	rt	immediate		
ori	001101	rs	rt	immediate		
xori	001110	rs	rt	immediate		
lui	001111	rs	rt	immediate		
lw	100011	rs	rt	immediate		
sw	101011	rs	rt	immediate		
beq	000100	rs	rt	immediate		
bne	000101	rs	rt	immediate		
slti	001010	rs	rt	immediate		
sltiu	001011	rs	rt	immediate		
j	000010	address				
jal	000011	address				
mul	011100	rs	rt	rd	00000	000010

### 三、总体架构部件的解释说明

#### 1、 动态流水线总体结构部件的解释说明

- a. 指令存储器 imem: 用于存储将要执行的指令



## b. PC 寄存器 PC\_reg

```
module PC_reg(  
    input clk,  
    input rst,  
    input [31:0] data_in,  
    output reg [31:0] data_out  
);
```

c. IF 和 ID 之间的流水寄存器 regs\_IF\_ID: 用于将 IF 输出结果传递给 ID

```
module regs_IF_ID(  
    input clk,  
    input rst,  
    input [31:0] npc_if,  
    input [31:0] instr_if,  
    output reg [31:0] npc_id = 32'b0,  
    output reg [31:0] instr_id = 32'b0  
);
```

d. 专用路径模块 mux: 将 EX 阶段或者 MEM 阶段的输出结果定向前推至 ID 阶段的多路选择器中

```
module mux(  
    input [4:0] rs,  
    input [4:0] rt,  
    input [31:0] rs_wire,  
    input [31:0] rt_wire,  
  
    input [31:0] npc_ex,  
    input [31:0] mul_ex,  
    input [31:0] alu_ex,  
    input [4:0] w_addr_ex,  
    input write_ex,  
    input is_lw_ex,  
    input is_jal_ex,  
    input is_mul_ex,
```

```

input [31:0] npc_mem,
input [31:0] mul_mem,
input [31:0] alu_mem,
input [4:0] w_addr_mem,
input write_mem,
input is_lw_mem,
input is_jal_mem,
input is_mul_mem,

output reg [31:0] rs_mux = 32'b0,
output reg [31:0] rt_mux = 32'b0,
output reg conflict_lw = 1'b0
);

```

e. ID 阶段的控制模块 **controller**: 用于进行指令的解码和产生控制信号

```

module controller(
    input [5:0] operation,
    input [5:0] funct,
    input [31:0] rs_wire,
    input [31:0] rt_wire,

    output is_jump,
    output dm_w_signal_id,
    output write_id,
    output [3:0] aluc_id,

    output [1:0] mux_pc,
    output mux_alu1_id,
    output [1:0] mux_alu2_id,
    output [1:0] mux_waddr_id,

    output is_lw_id,
    output is_jal_id,
    output is_mul_id
);

```

f. 冲突检测模块 **conflict\_judge**: 用于判断指令

是否存在冲突，若存在，则暂停流水

```
module conflict_judge(  
    input [31:0] instr,  
    input is_lw_id,  
    input is_lw_ex,  
    input is_lw_mem,  
    input write_id,  
    input write_ex,  
    input write_mem,  
    input [4:0] w_addr_id,  
    input [4:0] w_addr_ex,  
    input [4:0] w_addr_mem,  
    output is_stall  
);
```

g. ID 和 EX 阶段的流水寄存器 regs\_ID\_EX: 用于将 ID 阶段的输出传递给 EX

```
module regs_ID_EX(  
    input clk,  
    input rst,  
  
    input dm_w_signal_id,  
    input write_id,  
    input is_lw_id,  
    input is_jal_id,  
    input is_mul_id,  
    input mux_alu1_id,  
    input [1:0] mux_alu2_id,  
    input [3:0] aluc_id,  
    input [31:0] npc_id,  
    input [4:0] w_addr_id,  
    input [31:0] shamt_id,  
    input [31:0] simmediate_id,  
    input [31:0] uimmediate_id,  
    input [31:0] rs_wire_id,  
    input [31:0] rt_wire_id,  
    input [31:0] dm_wdata_id,
```

```

output reg dm_w_signal_ex = 1'b0,
output reg write_ex = 1'b0,
output reg is_lw_ex = 1'b0,
output reg is_jal_ex = 1'b0,
output reg is_mul_ex = 1'b0,
output reg mux_alu1_ex = 1'b0,
output reg [1:0] mux_alu2_ex = 2'b0,
output reg [3:0] aluc_ex = 4'b0,
output reg [31:0] npc_ex = 32'b0,
output reg [4:0] w_addr_ex = 5'b0,
output reg [31:0] shamt_ex = 32'b0,
output reg [31:0] simmediate_ex = 32'b0,
output reg [31:0] uimmediate_ex = 32'b0,
output reg [31:0] rs_wire_ex = 32'b0,
output reg [31:0] rt_wire_ex = 32'b0,
output reg [31:0] dm_wdata_ex = 32'b0
);

```

#### h. 运算模块 alu:

```

module alu(
    input [31:0] alu1,
    input [31:0] alu2,
    input [3:0] aluc,
    output reg [31:0] result
);

```

#### i. EX 阶段和 MEM 阶段的流水寄存器

regs\_EX\_MEM: 用于将 EX 阶段的输出传递给 MEM 阶段



```

module regs_EX_MEM(
    input clk,
    input rst,

    input dm_w_signal_ex,
    input write_ex,
    input is_lw_ex,
    input is_jal_ex,
    input is_mul_ex,
    input [4:0] w_addr_ex,
    input [31:0] alu_ex,
    input [31:0] mul_ex,
    input [31:0] npc_ex,
    input [31:0] dm_wdata_ex,

    output reg dm_w_signal_mem = 1'b0,
    output reg write_mem = 1'b0,
    output reg is_lw_mem = 1'b0,
    output reg is_jal_mem = 1'b0,
    output reg is_mul_mem = 1'b0,
    output reg [4:0] w_addr_mem = 5'b0,
    output reg [31:0] alu_mem = 32'b0,
    output reg [31:0] mul_mem = 32'b0,
    output reg [31:0] npc_mem = 32'b0,
    output reg [31:0] dm_wdata_mem = 32'b0
);

```

#### j. 数据存储模块 dmem:

```

module dmem(
    input clk,
    input rst,
    input write,
    input [31:0] addr,
    input [31:0] data_in,
    output [31:0] data_out
);

```

#### k. MEM 阶段和 WB 阶段的流水寄存器 regs\_MEM\_WB，用于将 MEM 阶段的输出传递

## 给 WB 阶段

```
module regs_MEM_WB(  
    input clk,  
    input rst,  
  
    input write_me,  
    input is_lw_me,  
    input is_jal_me,  
    input is_mul_me,  
    input [4:0] w_addr_me,  
    input [31:0] alu_me,  
    input [31:0] mul_me,  
    input [31:0] npc_me,  
    input [31:0] dm_rdata_me,  
  
    output reg write_w = 1'b0,  
    output reg is_lw_w = 1'b0,  
    output reg is_jal_w = 1'b0,  
    output reg is_mul_w = 1'b0,  
    output reg [4:0] w_addr_w = 5'b0,  
    output reg [31:0] alu_w = 32'b0,  
    output reg [31:0] mul_w = 32'b0,  
    output reg [31:0] npc_w = 32'b0,  
    output reg [31:0] dm_rdata_w = 32'b0  
);
```

## I. 寄存器堆模块 regfiles:

```
module regfiles(  
    input clk,  
    input rst,  
    input write,  
    input [4:0] rn1,  
    input [4:0] rn2,  
    input [4:0] w_addr,  
    input [4:0] r_addr,  
    input [31:0] data_in,  
    output [31:0] data_out1,  
    output [31:0] data_out2,  
    output [31:0] data_out  
);
```

## 2、 整体结构解释

- a. 总体采用哈佛结构，指令和数据分开存储，通过数据总线和指令总线和 CPU 进行数据交换。
- b. 一条指令的执行分为 IF、ID、EX、MEM、WB 五个阶段，分别为取址、取值、计算、访存和写回，根据指令类型的不同在不同的阶段完成不同的操作。
- c. 在相邻的两个阶段设置流水寄存器来实现指令的流水执行
- d. 设置专用路径，将之前周期 EX 阶段或者 MEM 阶段输出的结果定向前推至 ID 段向 EX 阶段输出源操作数时，使用控制信号进行多路进行选择。
- e. 采用延迟操的思想解决控制相关的冲突
- f. 增加冲突判断模块，当检测到冲突时使用暂停流水的方式来消除数据相关。

## 四、实验仿真过程

### 1、 动态流水线的仿真过程

- a. 测试文件的编写

```

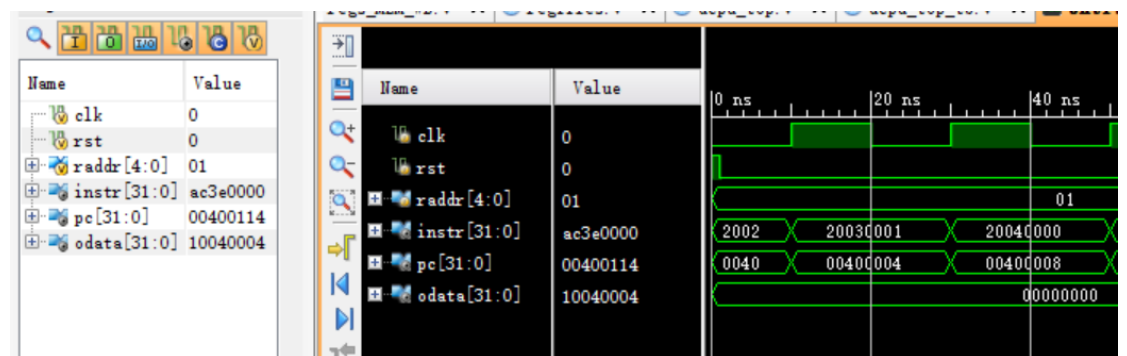
module dcpu_top_tb();
    reg clk;
    reg rst;
    reg [4:0] raddr;
    wire [31:0] instr;
    wire [31:0] pc;
    wire [31:0] odata;
    initial
    begin
        clk = 0;
        rst = 1;
        raddr = 5'b00001;
        #1 rst = 0;
    end

    always
    begin
        #10 clk <= ~clk;
    end

    dcpu_top dcpu_top_inst(clk, rst, raddr, instr, pc, odata);
endmodule

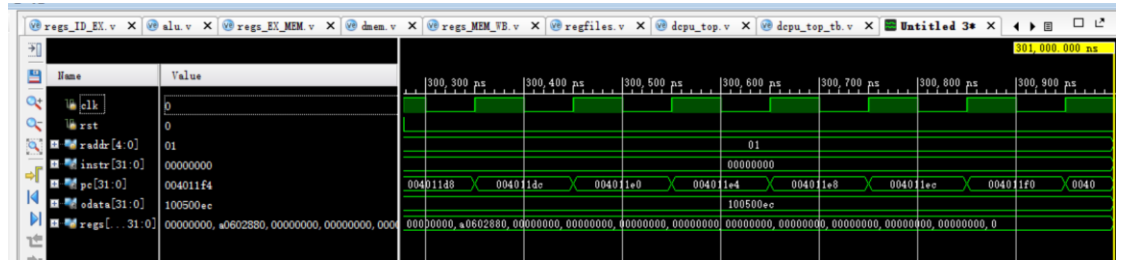
```

## b. 进行仿真

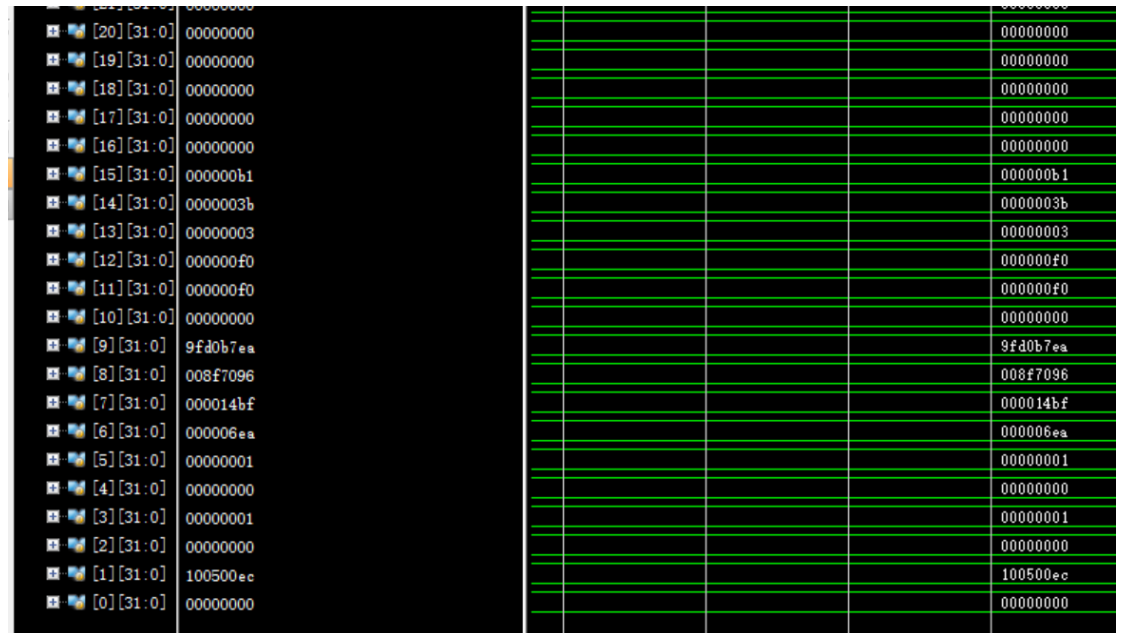


## 五、实验仿真的波形图及某时刻寄存器值的物理意义

### 1、 动态流水线的波形图



## 2、 结束时刻寄存器值的物理意义



- 寄存器 6 中存储了 `a[59]` 的值，十六进制的 `6ea`，即 `1770`
- 寄存器 7 中存储了 `b[59]` 的值，十六进制的 `14bf`，即 `5311`
- 寄存器 8 中存储了 `c[59]` 的值，十六进制的 `8f7096`，即 `9400470`
- 寄存器 9 中存储了 `d[59]` 的值，十六进制的 `9fd0b7ea`，即 `2681255914`
- 其他寄存器存储了一些计算的所需的中间值

## 六、实验验算数学模型及算法程序

```
int a[m],b[m],c[m],d[m];
```

```
a[0]=0;
```

```
b[0]=1;
```

```
a[i]=a[i-1]+i;
```

```
b[i]=b[i-1]+3i;
```

$$c[i] = \begin{cases} a[i], & 0 \leq i \leq 19 \\ a[i] + b[i], & 20 \leq i \leq 39 \\ a[i] * b[i], & 40 \leq i \leq 59 \end{cases}$$
$$d[i] = \begin{cases} b[i], & 0 \leq i \leq 19 \\ a[i] * c[i], & 20 \leq i \leq 39 \\ c[i] * b[i], & 40 \leq i \leq 59 \end{cases}$$

### 1. 编写 C++程序如下:

```
4 int main() {
5     int a[60];
6     int b[60];
7     int c[60];
8     unsigned int d[60];
9
10    for (int i = 0; i < 60; i++) {
11        if (i == 0) {
12            a[0] = 0;
13            b[0] = 1;
14            c[0] = a[0];
15            d[0] = b[0];
16        }
17        else if (i > 0 && i <= 19) {
18            a[i] = a[i - 1] + i;
19            b[i] = b[i - 1] + 3 * i;
20            c[i] = a[i];
21            d[i] = b[i];
22        }
23        else if (i >= 20 && i <= 39) {
24            a[i] = a[i - 1] + i;
25            b[i] = b[i - 1] + 3 * i;
26            c[i] = a[i] + b[i];
27            d[i] = a[i] * c[i];
28        }
29        else {
30            a[i] = a[i - 1] + i;
31            b[i] = b[i - 1] + 3 * i;
32            c[i] = a[i] * b[i];
33            d[i] = b[i] * c[i];
34        }
35    }
```

## 2. 汇编程序如下：

```
.text
main:
addi $2,$0,0 #a[0]
addi $3,$0,1 #b[0]
addi $4,$0,0 #c[0]
addi $5,$0,1 #d[0]
addi $6,$0,0 #a[i]
addi $7,$0,1 #b[i]
addi $8,$0,0 #c[i]
addi $9,$0,1 #d[i]
addi $10,$0,0 #flag for count
addi $11,$0,240 #sum counts
addi $12,$0,4 #counter
addi $13,$0,3 #factor
lui $1,0x1001 #0x10010000 basic address
addu $1,$1,$0
sw $2,0($1)
lui $1,0x1002
addu $1,$1,$0
sw $3,0($1)
lui $1,0x1003
addu $1,$1,$0
sw $4,0($1)
lui $1,0x1004
addu $1,$1,$0
sw $5,0($1)

loop:
srl $14,$12,2 #i=count/4
add $6,$6,$14 #a[i]=a[i-1]+i
lui $1,0x1001
addu $1,$1,$12 #address+count
sw $6,0($1)
mul $15,$13,$14 #3*i
add $7,$7,$15 #b[i]=b[i-1]+3i
lui $1,0x1002
addu $1,$1,$12
sw $7,0($1)
slti $10,$12,80 #i<20?
bne $10,1,op2 #i>=20
lui $1,0x1003 #i<20
addu $1,$1,$12
sw $6,0($1) #c[i]=a[i]
lui $1,0x1004
addu $1,$1,$12
sw $7,0($1) #d[i]=b[i]
addi $8,$6,0
addi $9,$7,0
j nxtloop

op2:
slti $10,$12,160 #i<40?
addi $1,$0,1
bne $10,$1,op3 #i>=40
add $8,$6,$7 #c[i]=a[i]+b[i]
lui $1,0x1003
addu $1,$1,$12
sw $8,0($1)
mul $9,$8,$6 #d[i]=a[i]*c[i]
lui $1,0x1004
addu $1,$1,$12
sw $9,0($1)
j nxtloop

op3:
mul $8,$6,$7 #c[i]=a[i]*b[i]
lui $1,0x1003
addu $1,$1,$12
sw $8,0($1)
mul $9,$8,$7 #d[i]=c[i]*b[i]
lui $1,0x1004
addu $1,$1,$12
sw $9,0($1)

nxtloop:
add $30,$8,$9 #e[i]=c[i]+d[i]
lui $1,0x1005
addu $1,$1,$12
sw $30,0($1)
addi $12,$12,4 #count+4
bne $12,$11,loop
```

## 3. coe 文件：

20020000	200b00f0
20030001	200c0004
20040000	200d0003
20050001	3c011001
20060000	00200821
20070001	ac220000
20080000	3c011002
20090001	00200821
200a0000	ac230000

3c011003	3c011004
00200821	002c0821
ac240000	ac270000
3c011004	20c80000
00200821	20e90000
ac250000	08100042
000c7082	298a00a0
00ce3020	20010001
3c011001	15410009
002c0821	00c74020
ac260000	3c011003
71ae7802	002c0821
00ef3820	ac280000
3c011002	71064802
002c0821	3c011004
ac270000	002c0821
298a0050	ac290000
20010001	08100042
142a0009	70c74002
3c011003	3c011003
002c0821	002c0821
ac260000	ac280000



71074802	3c011005
3c011004	002c0821
002c0821	ac3e0000
ac290000	218c0004
0109f020	158bffd0

## 七、实验验算程序下板测试过程与实现

### 1. 编写下板顶层文件（使用七段数码管显示结果）：

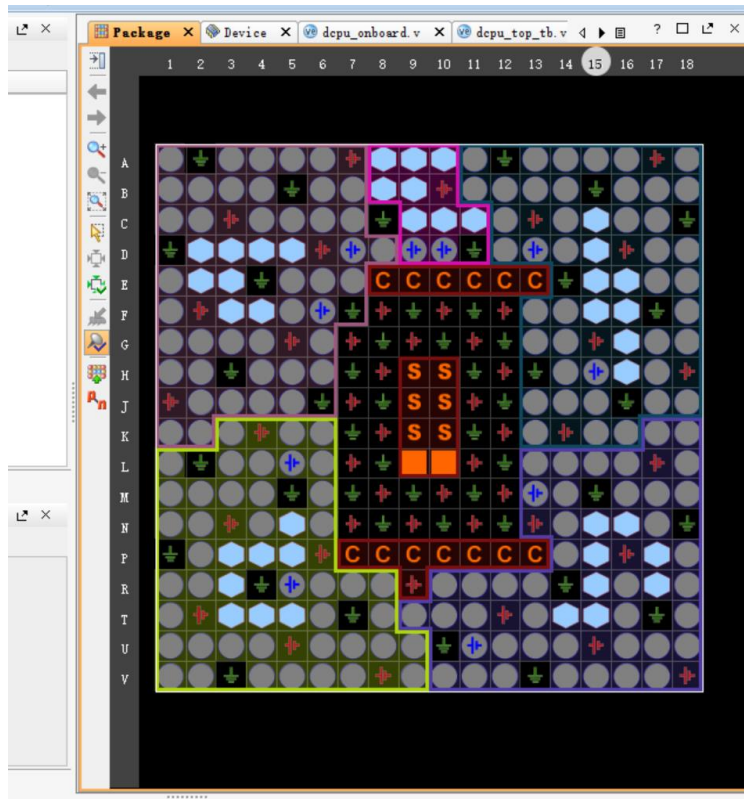
```

module dcpu_onboard(
    input clk_in,
    input enable,
    input reset,
    input stop,
    input [1:0] choice,
    input [4:0] raddr,
    output [7:0] o_seg,
    output [7:0] o_sel
);
    wire [31:0] inst;
    wire [31:0] pc;
    wire [31:0] odata;
    wire [31:0] display;
    reg [24 : 0] cnt;
    always @ (posedge clk_in, posedge reset, posedge stop)
    if (reset)
        cnt <= 0;
    else if (stop)
        cnt <= cnt;
    else
        cnt <= cnt + 1'b1;
    wire clk_cpu = cnt[24];
    assign display = choice[0] ? inst : (choice[1] ? pc : odata);
    seg7x16 seg7x16_inst(clk_in, reset, enable, display, o_seg, o_sel);
    dcpu_top dcpu_top_inst(clk_cpu, reset, raddr, inst, pc, odata);
endmodule

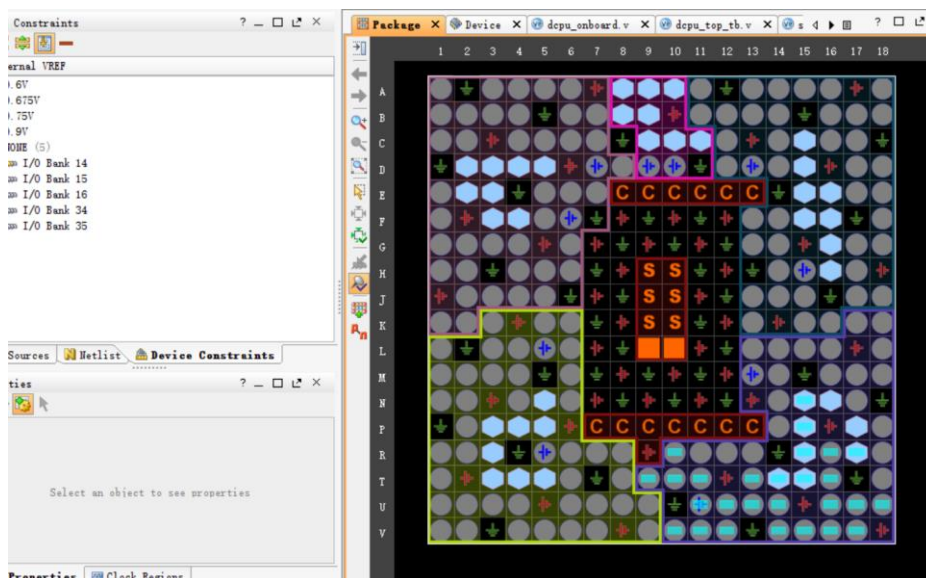
```

## 2. 综合、布线、下板

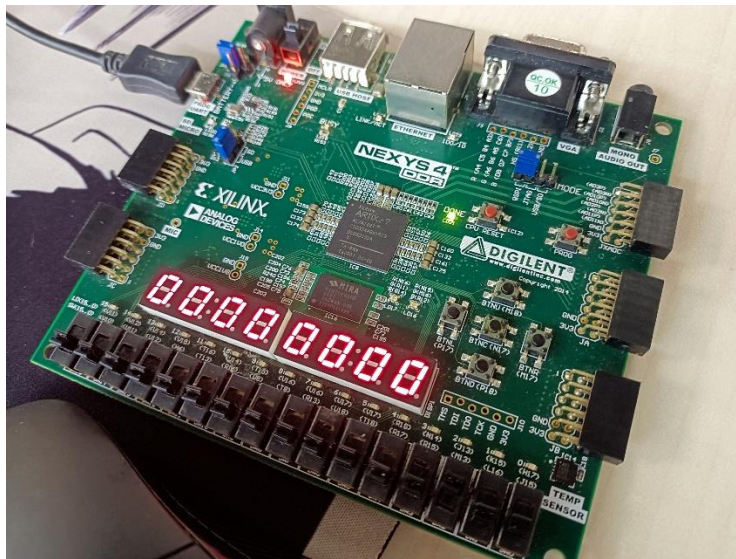
### a. 综合



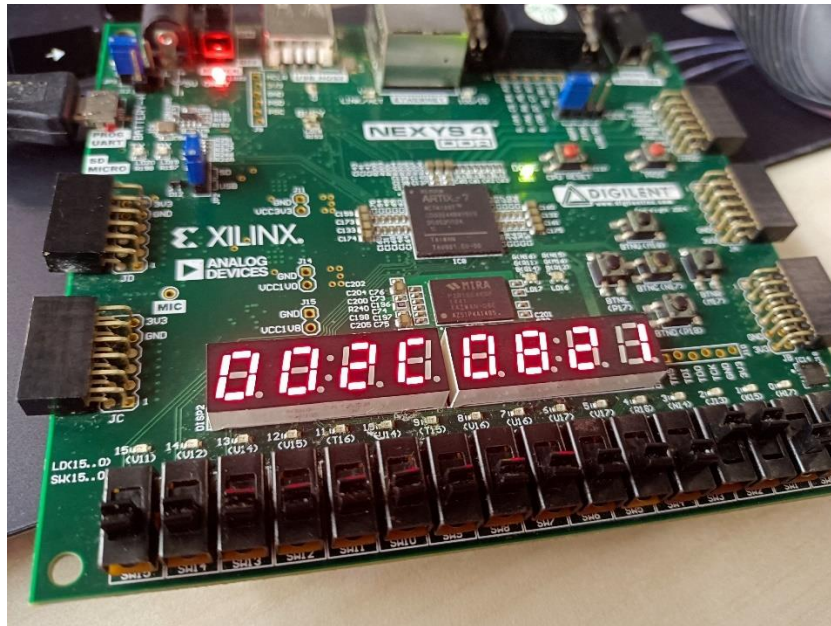
### b. 布线



### c. 下板

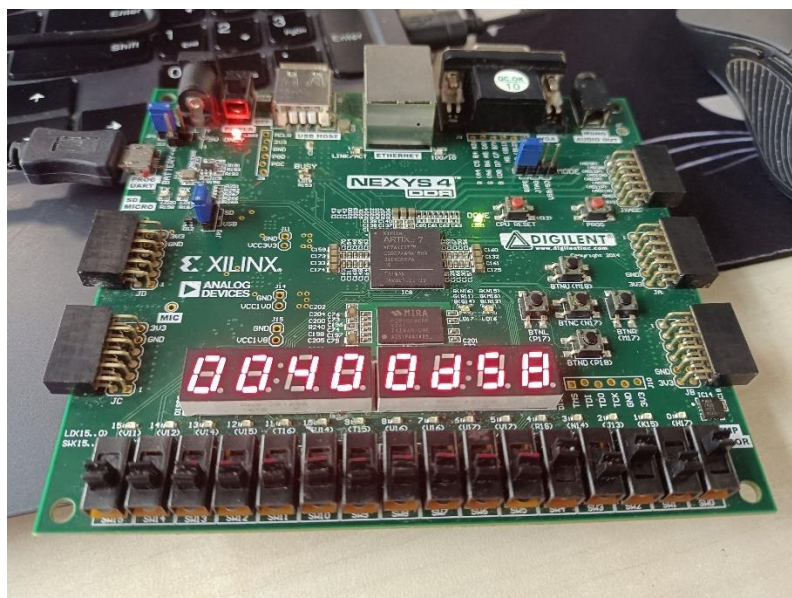


3. 观察现象（通过拨码开关来查看值）
  - a. 程序执行中间某时刻的指令值



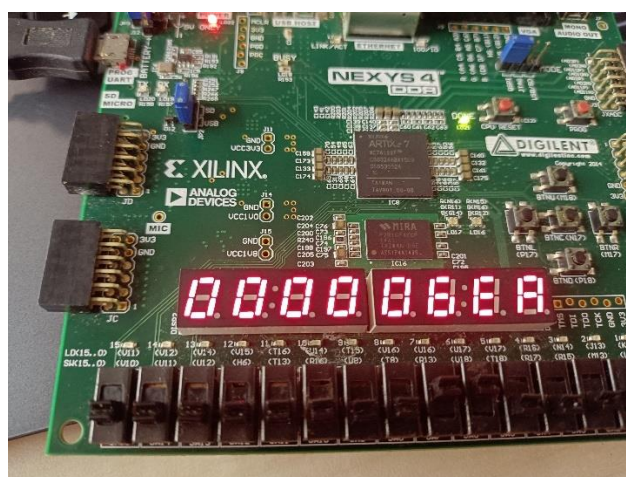
- b. 结束时 PC 的值



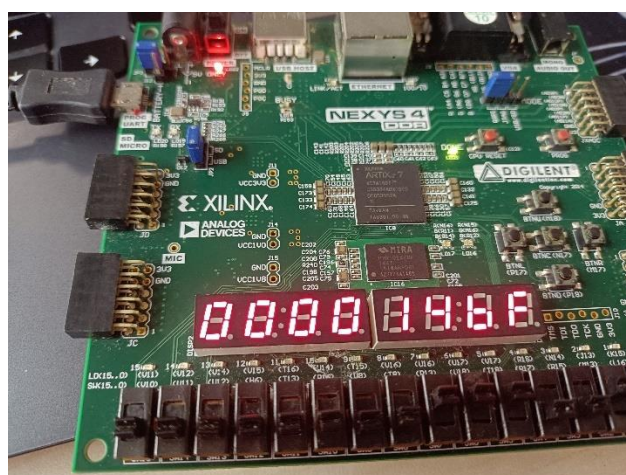


c. 结束时寄存器的值

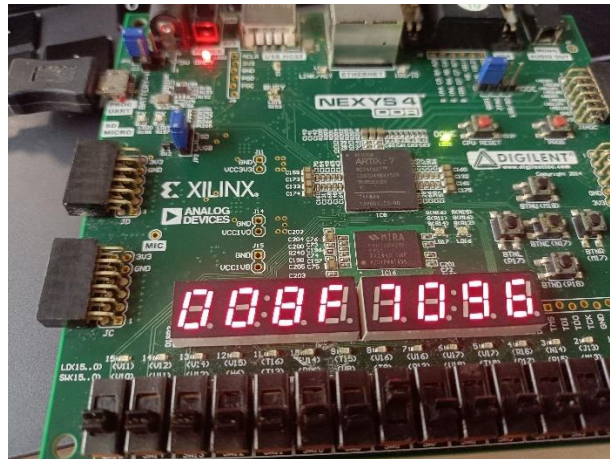
寄存器 6 (a[59]):



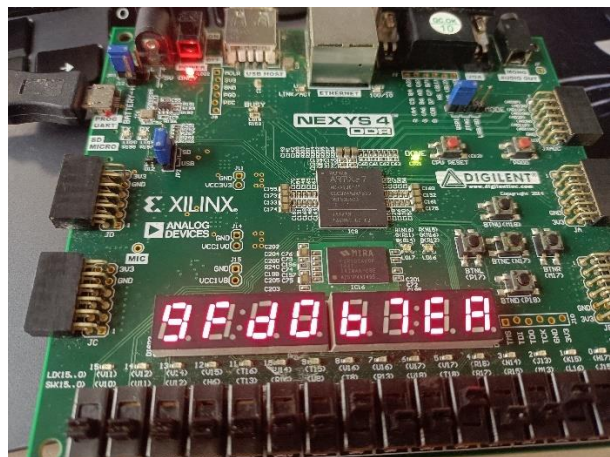
寄存器 7 (b[59]):



寄存器 8 (c[59]):



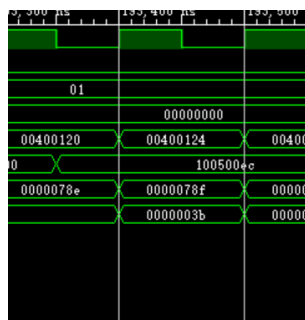
寄存器 9 (d[59]):



八、流水线的性能指标定性分析（包括：吞吐率、加速比、效率及相关与冲突分析）

1、 动态流水线的性能指标定性分析

a. 吞吐率



观察波形，运行完程序一共执行了 78f（16 进

制) 条指令, 空指令的数量为 3b (16 进制), 所以吞吐率为 $(1935-59)/1939$  个时钟周期, 结果为 0.968/时钟周期

b. 加速比

非流水线所使用的时钟周期数量:  $(1935-59)*5$

流水线所使用的时钟周期数量:  $(1935+5-1)$

所以加速比为 4.838

c. 效率

总时空区:  $1939*5$

有效的时空区:  $1876*5$

所以效率为: 0.968

## 2、冲突分析

在本次动态流水线 CPU 的设计中, 相对于静态流水线的 CPU, 添加了专用路径, 延迟槽等机制, 解决了部分的先写后读冲突和控制冲突, 大大减少了流水线 CPU 流水停止的概率, 提高了流水线 CPU 的执行效率。

## 九、总结与体会




本次实验设计了 32 条指令的动态流水线, 并编写了相关的测试程序进行仿真和下板测试, 分析了动态流水线 CPU 的相关性能, 掌握了解决指令冲突的几种方法对于 CPU 性能的显著提升效果。对提升 CPU 性能

的具体方法有了更加深入的了解。


十、附件（所有程序）

1、 动态流水线的设计程序

.v 文件：

 add.v	2023/12/30 15:31	V 文件	4 KB
 alu.v	2023/12/30 15:31	V 文件	3 KB
 conflict_judge.v	2023/12/30 15:26	V 文件	2 KB
 controller.v	2023/12/30 15:25	V 文件	5 KB
 dcpu.v	2023/12/30 15:34	V 文件	7 KB
 dcpu_onboard.v	2023/12/31 11:05	V 文件	2 KB
 dcpu_top.v	2023/12/30 15:40	V 文件	1 KB
 dmem.v	2023/12/30 15:34	V 文件	2 KB
 luislt.v	2023/12/30 15:32	V 文件	2 KB
 mux.v	2023/12/30 15:24	V 文件	5 KB
 PC_reg.v	2023/12/30 15:22	V 文件	1 KB
 regfiles.v	2023/12/30 15:37	V 文件	2 KB
 regs_EX_MEM.v	2023/12/30 15:33	V 文件	2 KB
 regs_ID_EX.v	2023/12/30 15:29	V 文件	3 KB
 regs_IF_ID.v	2023/12/30 15:23	V 文件	1 KB
 regs_MEM_WB.v	2023/12/30 15:36	V 文件	2 KB
 seg7x16.v	2023/12/31 10:30	V 文件	3 KB

测试文件：

 dcpu_top_tb.v	2023/12/31 13:38	V 文件	2 KB
---	------------------	------	------

2、 测试程序文件：

C++：

test.cpp

汇编语言：

test.asm

下板文件:

test.coe