DL&CO Experiment 1

*Author:221220089 崔博涵

I. 3-Input Voter

A. Steps

- 列出真值表（见下），写出对应的逻辑函数 $F = Y \cdot Z + X \cdot Z + X \cdot Y$。
- 打开 Logisim 软件，拖入适当数量的逻辑门，输入输出引脚，修改输入端个数。
- 依据逻辑函数连接相应引脚。
- 添加文字和标识符。
- 进行仿真测试，填写对应表。
- 保存电路文件。

B. Truth table

XYZ	F
000	0
001	0
010	0
011	1
100	0
101	1
110	1
111	1

Fig. 1. Truth table

C. Schematic Circuit Diagram

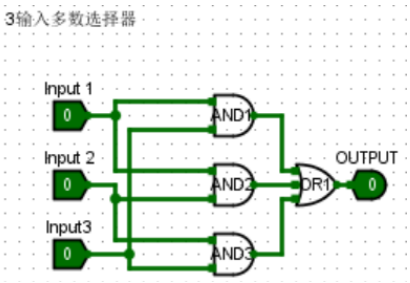

Fig. 2. Circuit Diagram

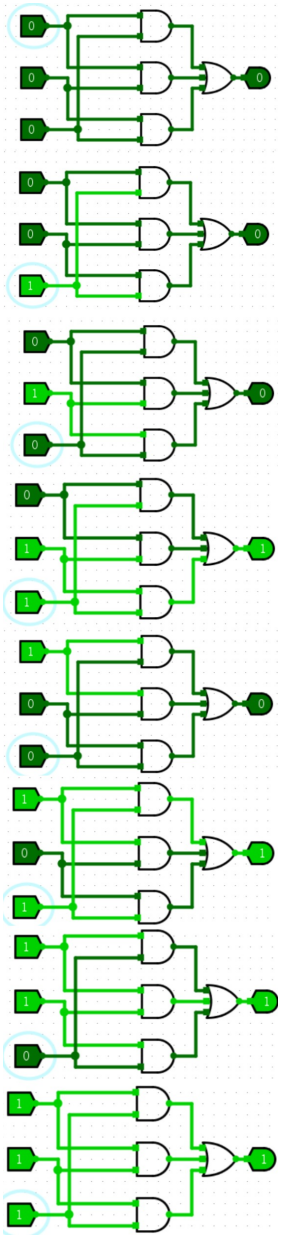

Fig. 3. Simulation

Input(XYZ)	Output
000	0
001	0
010	0
011	1
100	0
101	1
110	1
111	1

Fig. 4. Comparison Table

D. Simulation

Please refer to Fig3 & Fig4.

E. Errors and analysis

No error arose.

II. CMOS to OR-gate

A. Steps

- 根据逻辑电路原理用或非门与非门表示或门。
- 依据二者的 CMOS 晶体管实现加入晶体管，并设置朝向，加入电源和接地，以及引脚。
- 依据原理图连接电路图。
- 添加文字和标识符。
- 进行仿真测试，填写对应表。
- 保存电路文件。

B. Truth table

Input(XY)	Output
00	0
01	1
10	1
11	1

Fig. 5. Truth table

C. Schematic Circuit Diagram

Please refer to Fig6.

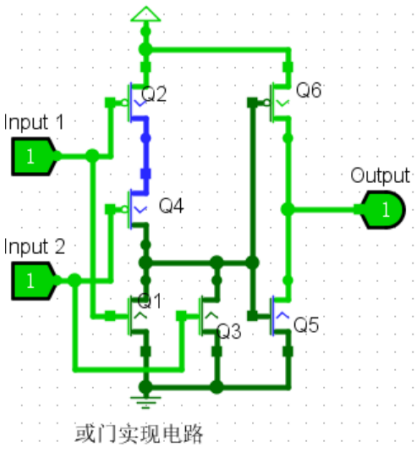

Fig. 6. Circuit Diagram

D. Simulation

Please refer to Fig7 & Fig8.

E. Errors and analysis

No error arose.

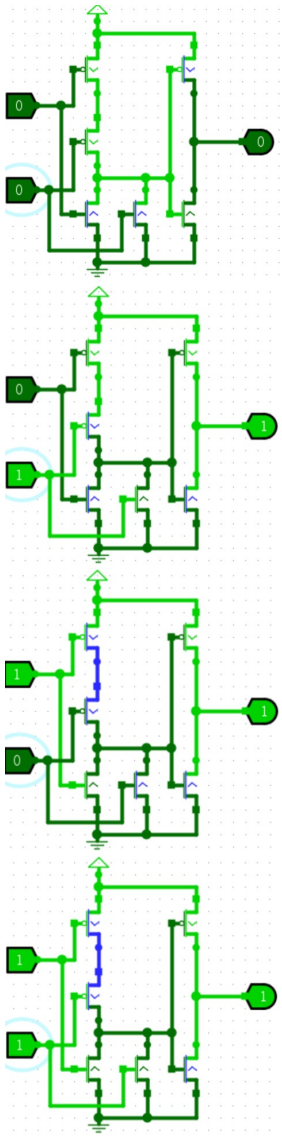

Fig. 7. Simulation

Input(XY)	Output
00	0
01	1
10	1
11	1

Fig. 8. Comparison Table

III. CMOS to 2-1MUX & 4-1MUX(Part 1)

A. Steps

- 根据根据功能写出逻辑函数 $Y = D0 \cdot \bar{S} + D1 \cdot S$。
- 依据逻辑电路的相关原理实现加入晶体管，并设置朝向，加入电源和接地，以及引脚和三态门。
- 连接电路图。
- 添加文字和标识符。
- 进行仿真测试，填写对应表。
- 保存电路文件。

B. Truth table

Input(SDoD1)	Output
000	0
001	0
010	1
011	1
100	0
101	1
110	0
111	1

Fig. 9. Truth table

C. Schematic Circuit Diagram

Please refer to Fig10.

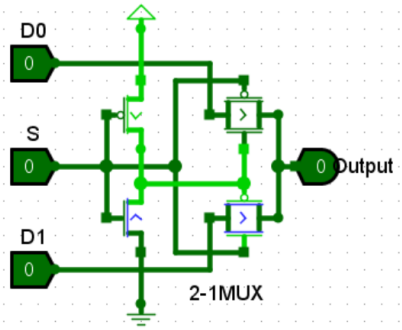

Fig. 10. Circuit Diagram

D. Simulation

Please refer to Fig11 & Fig12.

E. Errors and analysis

No error arose.

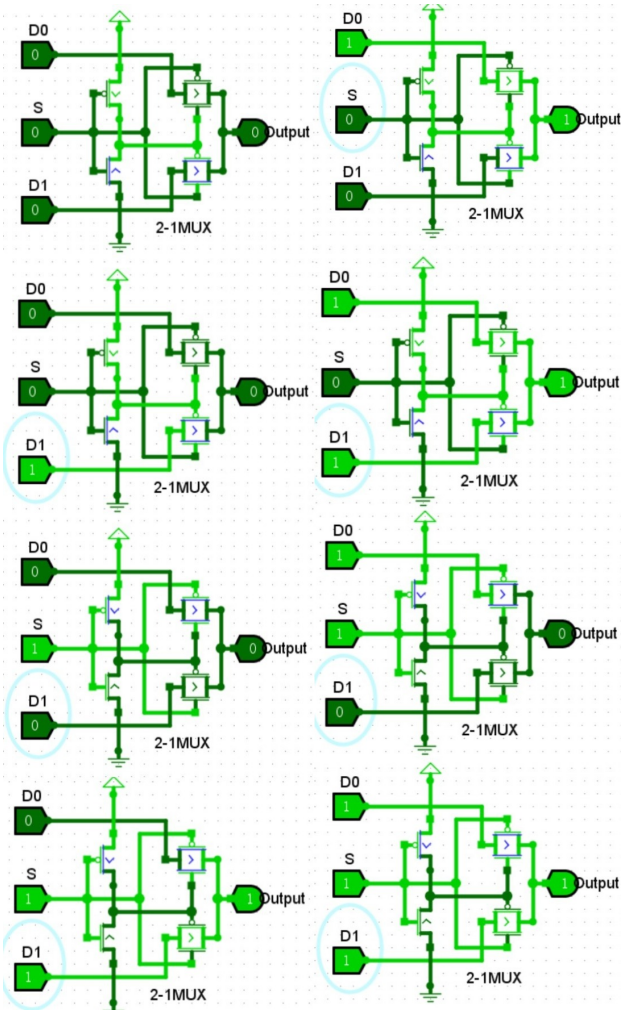

Fig. 11. Simulation

Input(SDoD1)	Output
000	0
001	0
010	1
011	1
100	0
101	1
110	0
111	1

Fig. 12. Comparison Table

IV. CMOS to 2-1MUX & 4-1MUX(Part 2)

A. Steps

- 复用 *Part*1 中制作的 2-1MUX，封装成子电路。
- 改变缺省外观。
- 按照 4-1MUX 的逻辑电路原理，加入 2-1MUX，加入输入和输出引脚。
- 连接电路图。
- 添加文字和标识符。
- 进行仿真测试，填写对应表。
- 保存电路文件。

B. Truth table

Input(SoS1)	Output
00	D0
01	D1
10	D2
11	D3

Fig. 13. Truth table

Please refer to Fig13.

C. Schematic Circuit Diagram

Please refer to Fig14.

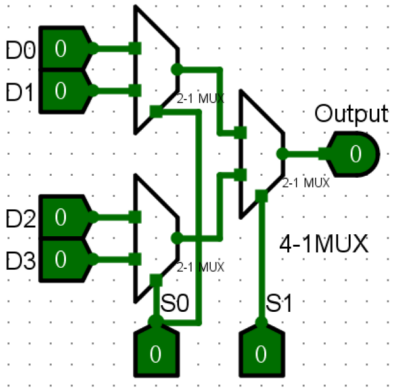

Fig. 14. Circuit Diagram

D. Simulation

Please refer to Fig15 & Fig16.

E. Errors and analysis

No error arose.

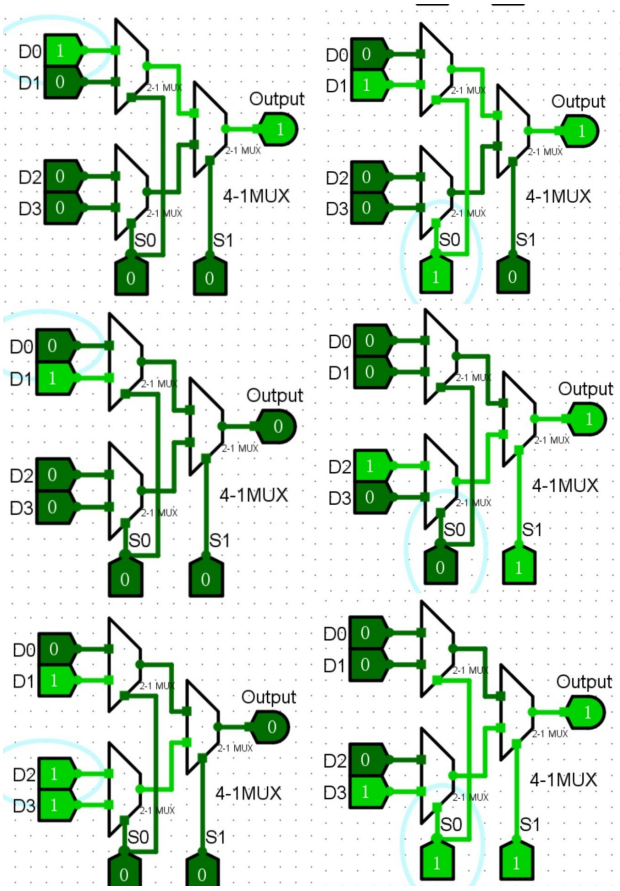

Fig. 15. Simulation (Sample)

Input(SoS1ABCD)	Output
000XXX	0
001XXX	1
01X0XX	0
01X1XX	1
10XX0X	0
10XX1X	1
11XXX0	0
11XXX1	1

Fig. 16. Comparison Table

V. 4-digit Binary Parity Checker

A. Preface

I did't realize what Parity Checker is until turned to TAs. Actually, I should had STFWed it. Mind needs to be altered.

https://www.techopedia.com/definition/1803/parity-check

https://www.youtube.com/watch?v=c8qAti1zBVQ

B. Steps

- 利用亦或门实现奇偶校验。
- 使用组合电路分析功能。
- 分别通过真值表，表达式和最小项生成逻辑电路（完全相同，下只列其一）。
- 进行仿真测试，填写对应表。
- 保存电路文件。

C. Truth table

D0	D1	D2	D3	EVEN	ODD
0	0	0	0	0	1
0	0	0	1	1	0
0	0	1	0	1	0
0	0	1	1	0	1
0	1	0	0	1	0
0	1	0	1	0	1
0	1	1	0	0	1
0	1	1	1	1	0
1	0	0	0	1	0
1	0	0	1	0	1
1	0	1	0	0	1
1	0	1	1	1	0
1	1	0	0	0	1
1	1	0	1	1	0
1	1	1	0	1	0
1	1	1	1	0	1

Fig. 17. Truth table (from Step3)

Please refer to Fig17.

D. Schematic Circuit Diagram

Please refer to Fig18 & Fig19.

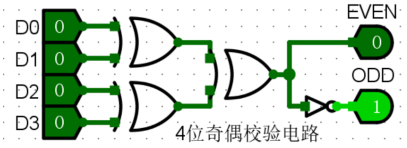

Fig. 18. Circuit Diagram

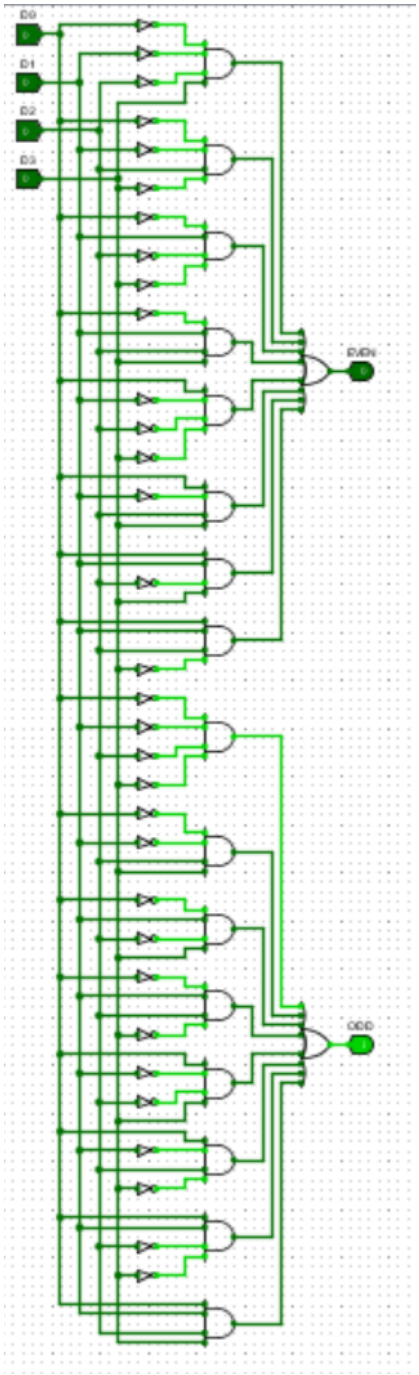

Fig. 19. Circuit Diagram

E. Simulation

Please refer to Fig20 & Fig21.

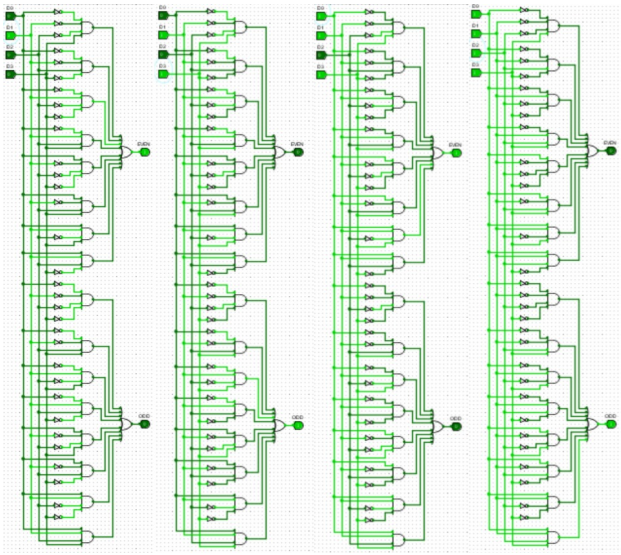

Fig. 20. Simulation (Sample)

Input($D_0D_1D_2D_3$)	Output(EVEN,ODD)
0000	01
0001	10
0010	10
0011	01
0100	10
0101	01
0110	01
0111	10
1000	10
1001	01
1010	01
1011	10
1100	01
1101	10
1110	10
1111	01

Fig. 21. Comparison Table

F. Errors and analysis

No error arose.

VI. Extended Questions

A.

4 种

- 手画。
- 组合电路分析功能中通过真值表生成。
- 组合电路分析功能中通过表达式生成。
- 组合电路分析功能中通过最小项生成。

B.

可以通过子电路功能逐级封装复用，形成一个复杂的电路。

C.

有 9 种输出组件。

- 引脚（Pin）
- 蜂鸣器（Buzzer）
- 发光二极管（LED）
- 彩色发光二极管（RGBLED）
- 数字示波器（Digital0scilloscope）
- 7 段数码管（7-SeementDisplay）
- 16 进制数字显示（HexDigitDisplay）
- LED 点阵（LEDMatri）
- 文本哑终端（TTY）

D.

通过 4-1MUX 与 2-1MUX 的组合，可以实现 8-1MUX。（如果仅能使用 4-1MUX，可以将 2-1MUX 换为 4-1MUX，将一个一个 S 输入端冗余）

电路图如下（已经过校验）。

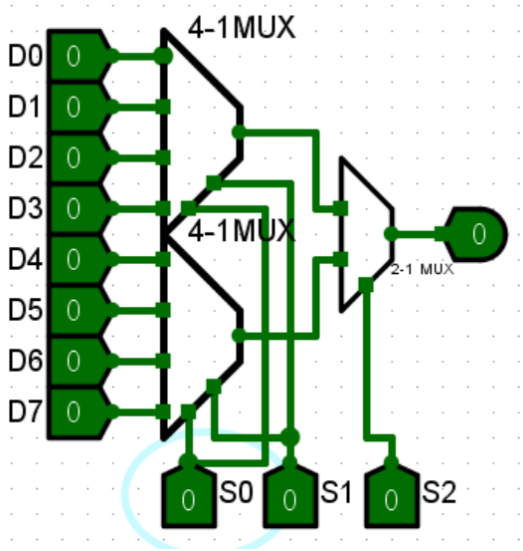

Fig. 22. Comparison Table

DL&CO Experiment 2

*Author:221220089 崔博涵

I. 74X138

A. Steps

- 根据原理图添加逻辑门，输入输出引脚，调整端口数
- 按照原理图连接电路图
- 添加文字和标识符
- 仿真测试，填写对照表
- 保存文件

B. Theoretical Diagram

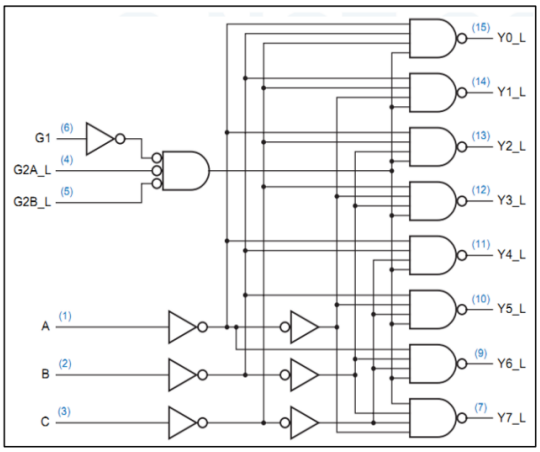

图 2.1　3-8 译码器 74X138 原理图

Fig. 1. Truth Table

C. Truth Table

Input(G,G2A_L,G2B_L)	Input(C,B,A)	Output(Y7~Yo)
x1x	xxx	11111111
xx1	xxx	11111111
0xx	xxx	11111111
100	000	11111110
100	001	11111101
100	010	11111011
100	011	11110111
100	100	11101111
100	101	11011111
100	110	10111111
100	111	01111111

Fig. 2. Truth Table

D. Schematic Diagram

Please refer to Fig.3

E. Simulation

Please refer to Fig.4 & Fig.5

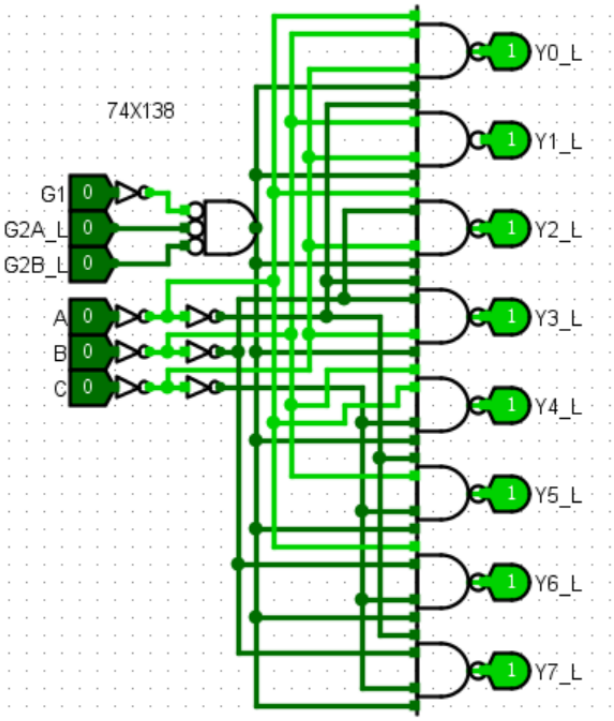

Fig. 3. Schematic Diagram

F. Error and analysis

No error arose.

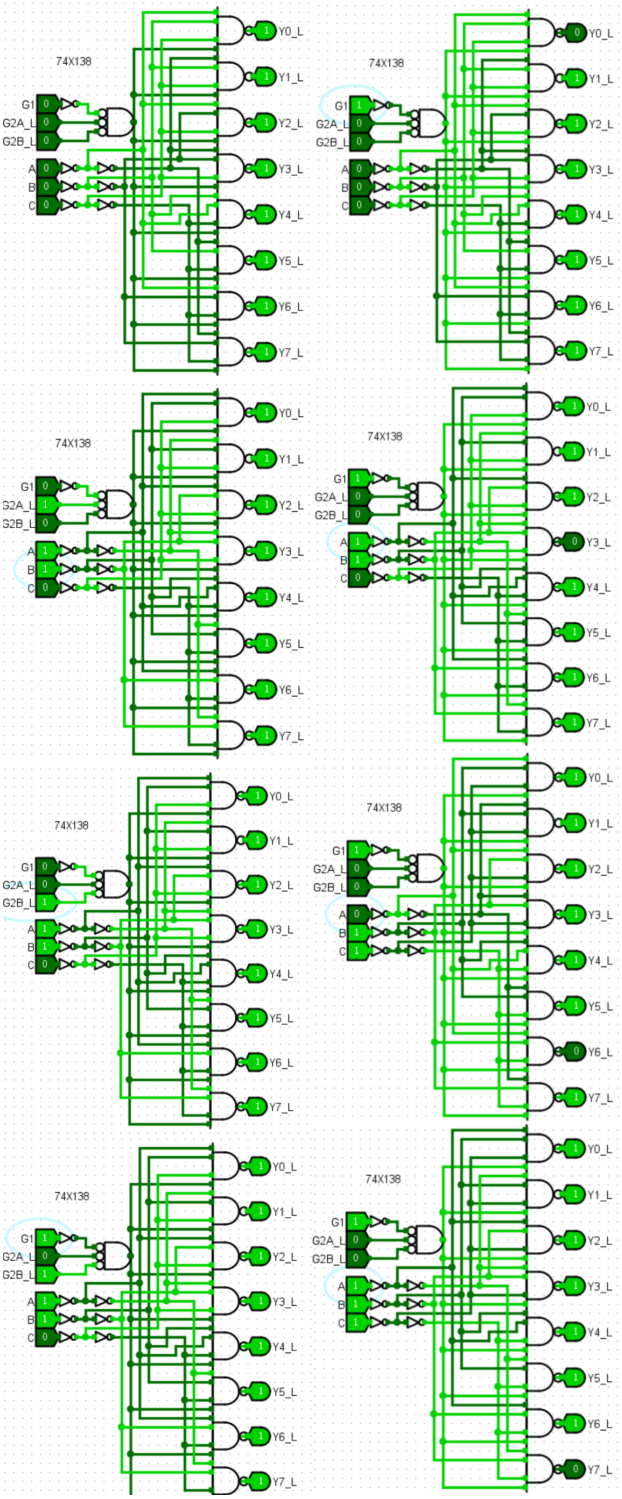

Fig. 4. Simulation

	H	I	J
11			
12	Input(G,G2A_L,G2B_L)	Input(C,B,A)	Output(Y7~Yo)
13	x1x	xxx	11111111
14	xx1	xxx	11111111
15	0xx	xxx	11111111
16	100	000	11111110
17	100	001	11111101
18	100	010	11111011
19	100	011	11110111
20	100	100	11101111
21	100	101	11011111
22	100	110	10111111
23	100	111	01111111

Fig. 5. Comparison Table

II. 8-3 Priority Encoder

A. Steps

- 根据原理图添加逻辑门，输入输出引脚，调整端口数
- 按照原理图连接电路图
- 添加文字和标识符
- 仿真测试，填写对照表
- 保存文件

B. Truth Table

Input(I0~I7)	Output(Q2,Q1,Q0)
1XXXXXXX	000
01XXXXXX	001
001XXXXX	010
0001XXXX	011
00001XXX	100
000001XX	101
0000001X	110
00000001	111

Fig. 6. Truth Table

C. Schematic Diagram

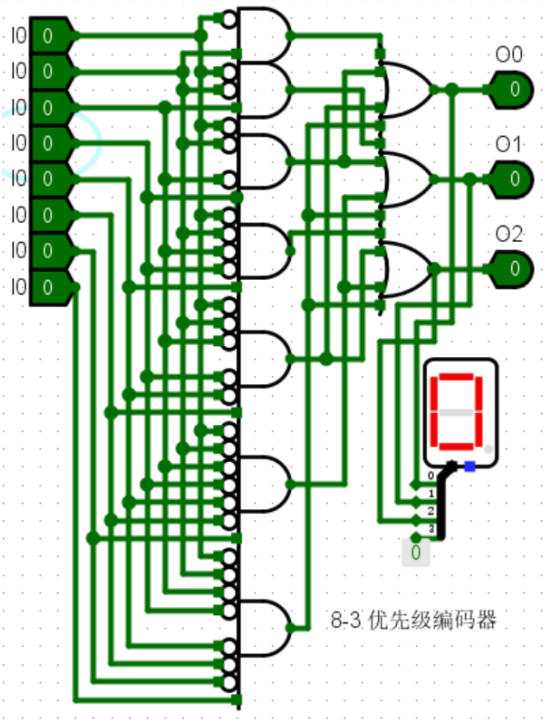

Fig. 7. Schematic Diagram

D. Simulation

Please refer to Fig.8 & Fig.9

E. Error and analysis

No error arose.

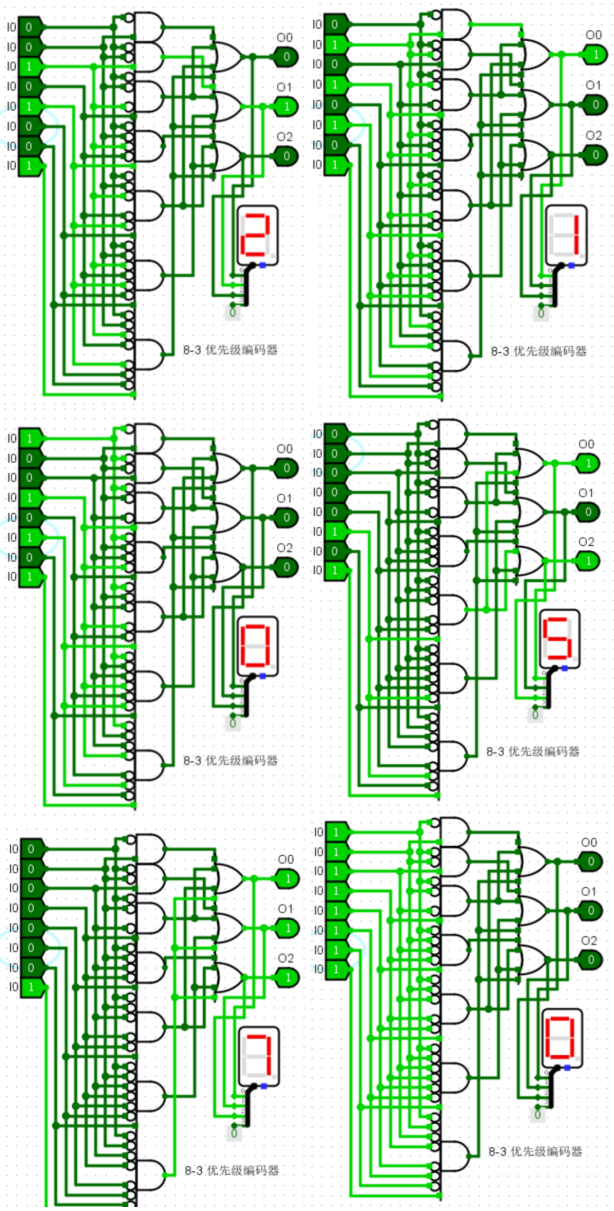

Fig. 8. Simulation

Input(I0~I7)	Output(Q2,Q1,Q0)	Hex
00101001	010	2
10010101	000	0
00000001	111	7
01010101	001	1
00000101	101	5
11111111	000	0

Fig. 9. Comparison Table

III. 4-bit CRA&S

A. Steps

- 设计 RCA 原理图
- 设计并实现全加器
- 根据原理图加入各个元器件
- 根据原理图连接实验电路
- 调整输入输出引脚，数据位宽和端口数
- 添加文字和标识符
- 拓展为加减法器
- 仿真测试，填写对照表
- 保存文件

B. Truth Table

Too complicated to display here.

C. Schematic Diagram

Please refer to Fig.10 & Fig.11 & Fig.12

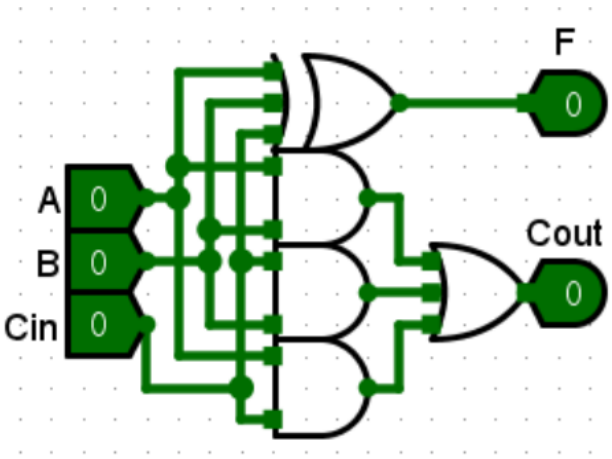

Fig. 10. Full adder

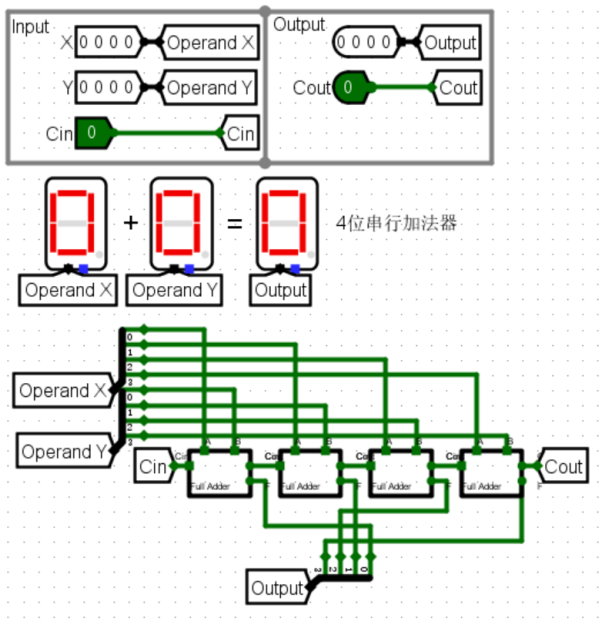

Fig. 11. 4-bit CRA

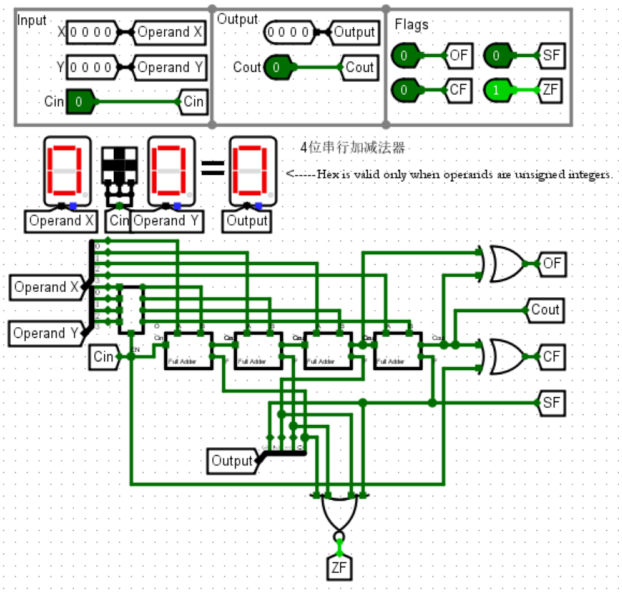

Fig. 12. 4-bit CRA&S

D. Simulation

The flags have been functioning here.
Please refer to Fig.13 & Fig.14

E. Error and analysis

发现 (Unsigned)F+7=0 检验后发现没有调整三输入亦或门的输出逻辑。

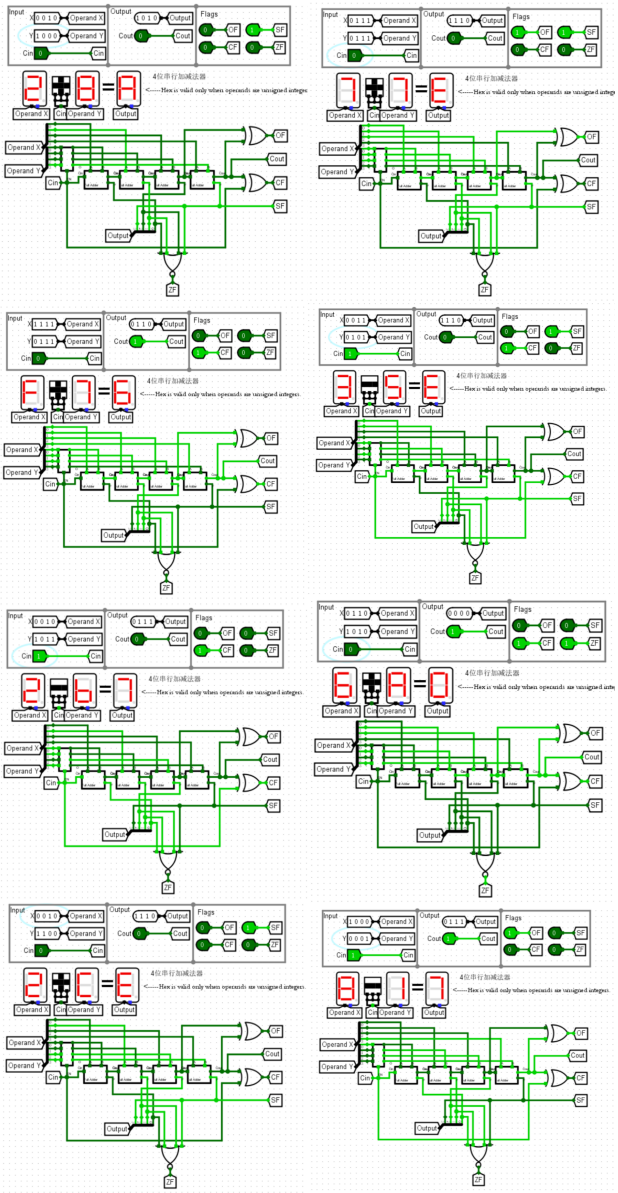

Fig. 13. Simulation

	Cin	Input(X)	Hex	Input(Y)	Hex	Output	Hex	OF	CF	SF	ZF
signed	0	0010	2	1000	8	1010	A	Invalid			
	0	1111	F	0111	7	0110	6				
	1	0010	2	1011	B	0111	7				
unsigned	0	0010(2)	Invalid	1100(-4)	Invalid	1110(-2)	Invalid	0	0	1	0
	0	0111(7)		0111(7)		1110(-2)		1	0	1	0
	1	0011(3)		0101(5)		1110(-2)		0	1	1	0
	0	0110(6)		1010(-6)		0000(0)		0	1	0	1
	1	1000(-7)		0001(1)		0111(7)		1	0	0	0

Fig. 14. Comparison Table

IV. Hamming Code Checker

A. Preface

The introduction on the tutorial is brief, so I found a more detailed explanationof Hamming Code by 3B1B.(Listed below).

https://www.bilibili.com/video/BV1WK411N7kz
https://www.bilibili.com/video/BV1pV411y7E8

B. Steps

- 根据汉明码相关原理得到原理图；
- 导入或实现各个子器件；
- 根据原理图连接实验电路
- 调整输入输出引脚，数据位宽和端口数
- 添加文字和标识符
- 仿真测试，填写对照表
- 保存文件

C. Truth Table

Too complicated to display here.

D. Schematic Diagram

#CAUTION The implemention in the tutorial is inversed, I change it in my experiment, namely, the lower bit is on the right side here.

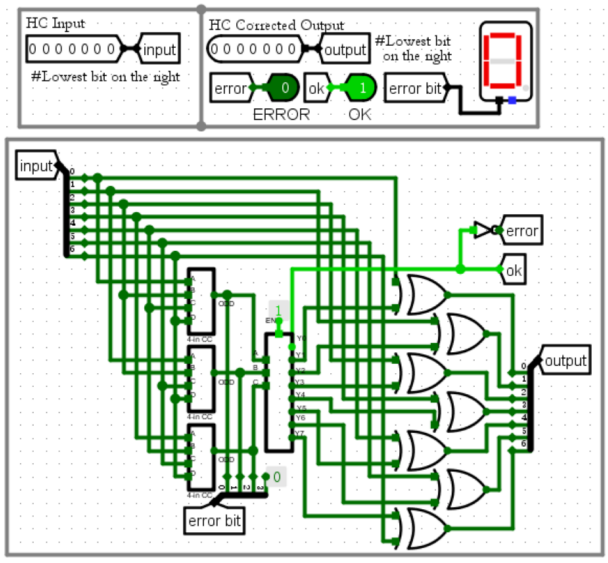

Fig. 15. Schematic Diagram

E. Simulation

Input(High-Low)	Output(High-Low)	Hex	ERROR	OK
0110100	0110100	0	0	1
1101000	1111000	5	1	0
1001101	1001100	1	1	0

Fig. 16. Comparison Table

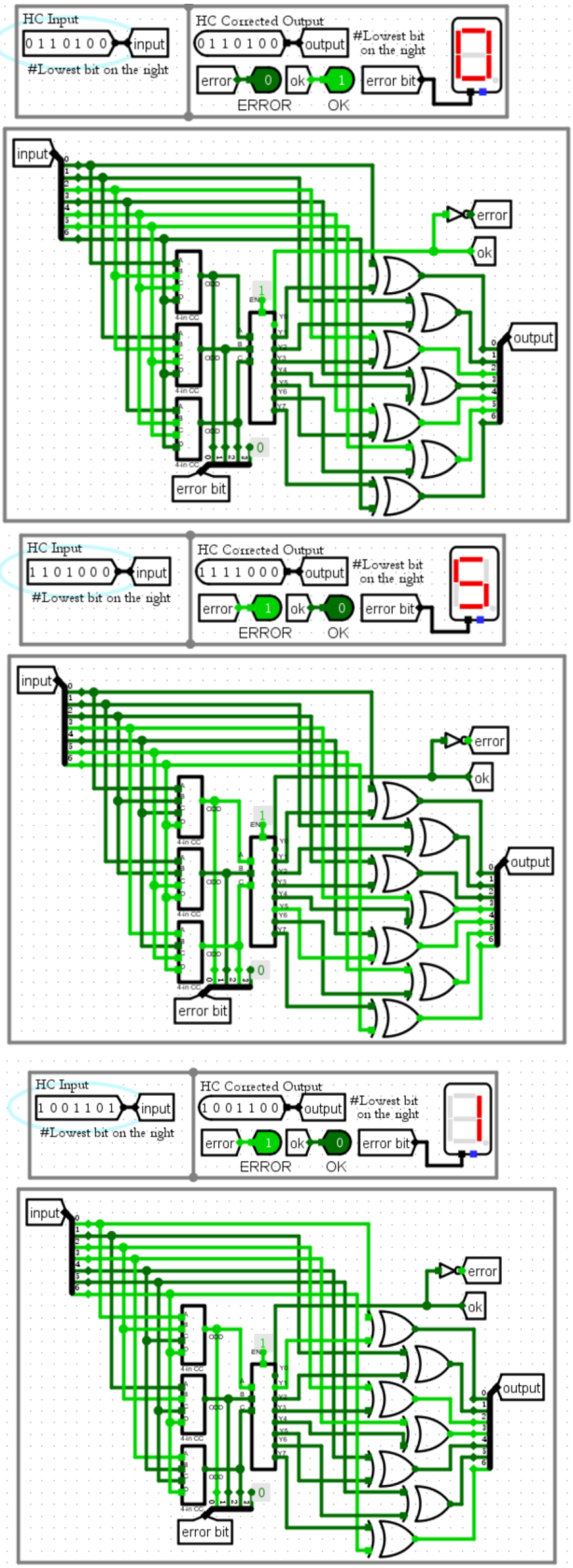

Fig. 17. Simulation

F. Error and analysis

检查后发现是 XOR 门用成了 NXOR 门。

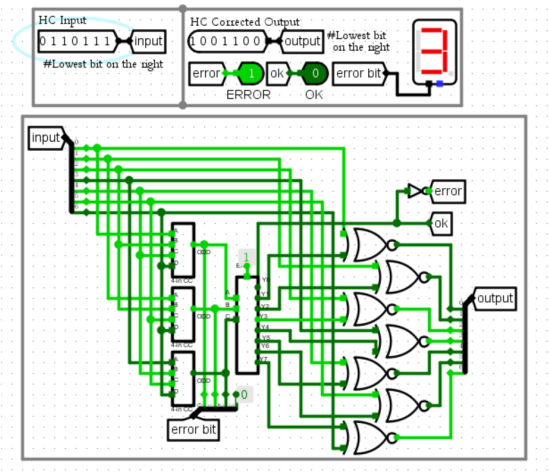

Fig. 18. Error

V. 8-bit Barrel Shifter

A. Steps

- 根据原理图加入各个元器件
- 调整端口数，输入输出引脚，数据位宽
- 根据原理图连接实验电路
- 添加文字和标识符
- 仿真测试，填写对照表
- 保存文件

B. Function Table

To complicated to display here.

C. Schematic Diagram

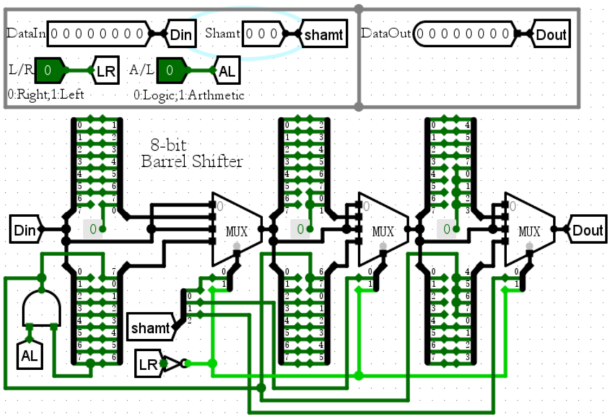

Fig. 19. Schematic Diagram

D. Simulation

Please refer to Fig.20 & Fig.21

Input	Shamt	L/R(R=0,L=1)	A/L(L=0,A=1)	Output
10100100	001	0	0	01010010
10011000	010	0	1	11100110
11111111	110	1	0	11000000
11011011	111	1	1	10000000

Fig. 20. Comparison Table

E. Error and analysis

No error arose.

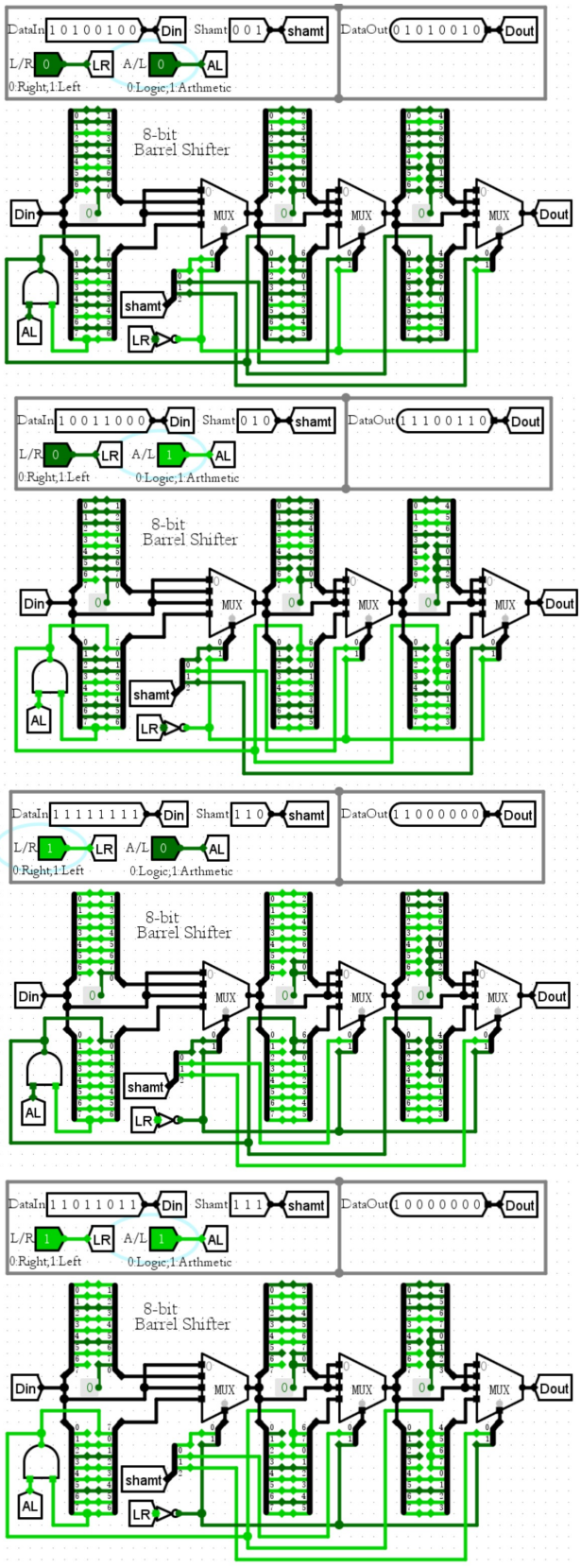

Fig. 21. Simulation

VI. Extended Questions

A. Flags

It has been implemented in the Experiment 3: 4-bit CRA above. Please refer to Page 4.

B. Example of Comparison

According to Fig 22, we compare 3(0011) and 5(0101) with 4-bit Subtracter.

The SF is 1(Negative). Therefore $3 < 5$.

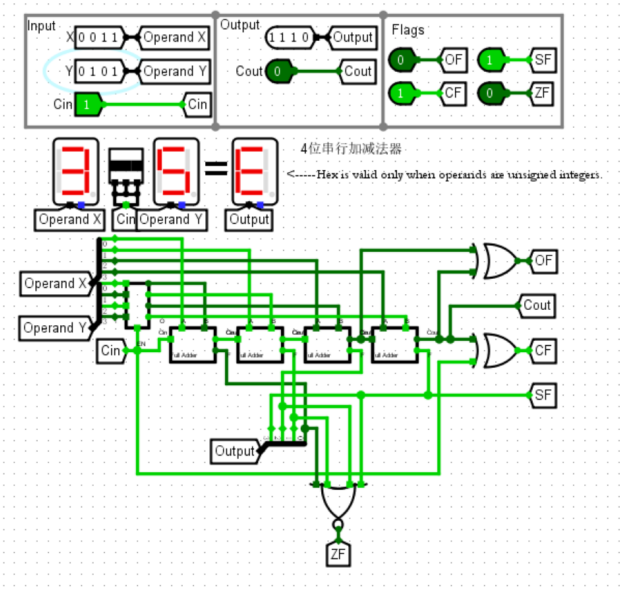

Fig. 22. Example

C. 32-bit Barrel Shifter

My inplementation is like a piece of *.

There might be an alternative to use Carryin and Carryout between the 8-bit Barrel Shifters. But I didn't think of a brief solution for that.

So I wrote a storage edition.

Please refer to Fig.23 & Fig.24 & Fig.25 & Fig.26 & Fig.27.

D. iNan Ideology

Please refer to Fig.28

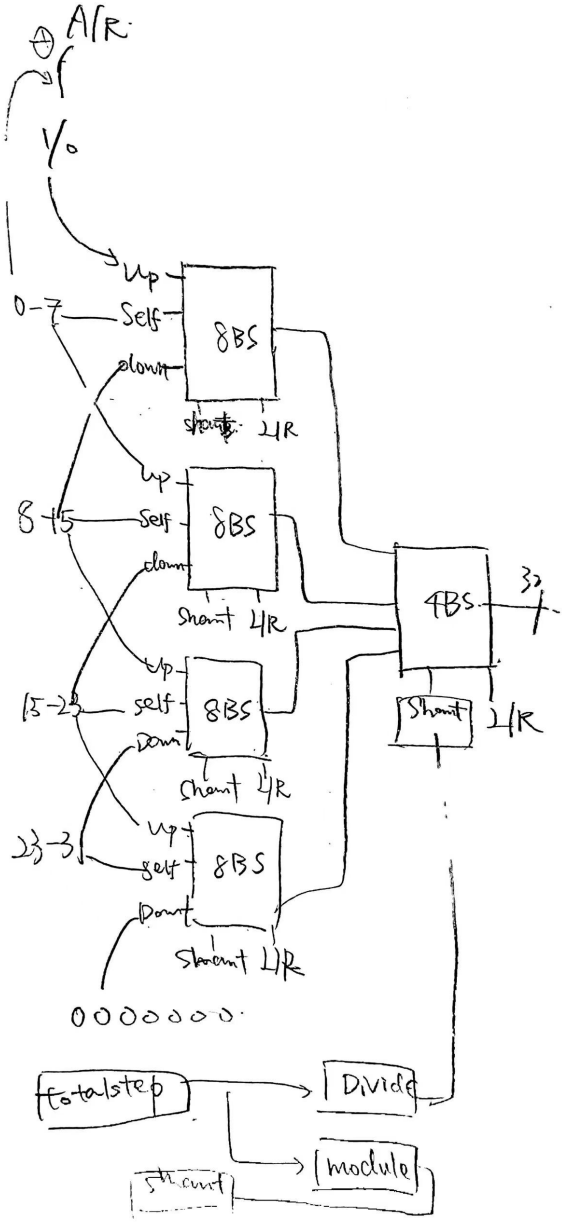

Fig. 23. My Handwriting Draft(doge)

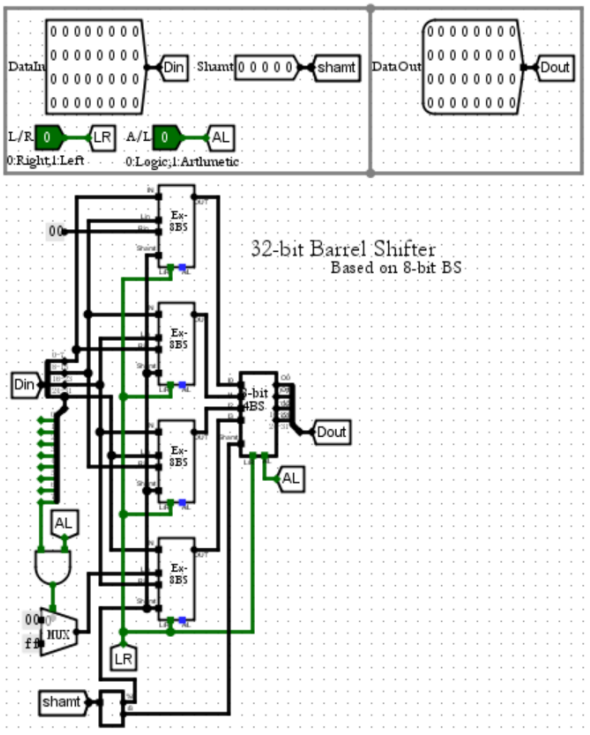

Fig. 24. The Total Diagram

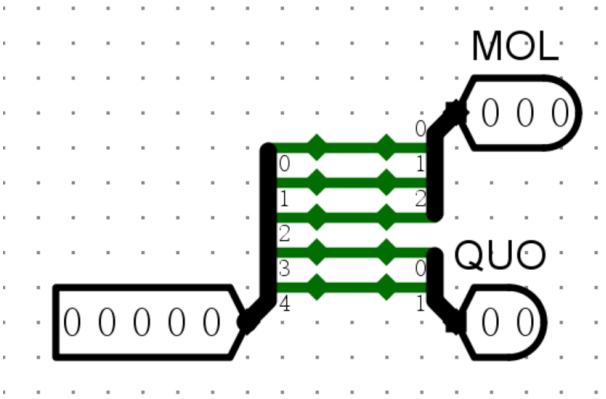

Fig. 26. /8 & %8

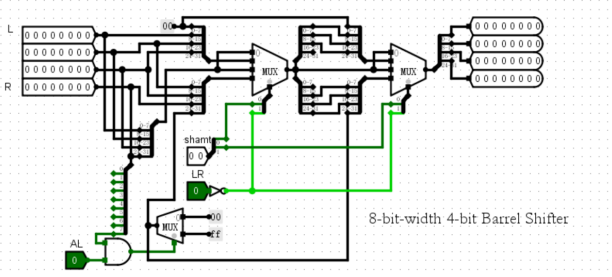

Fig. 27. 8-bit-width 4-bit Barrel Shifter

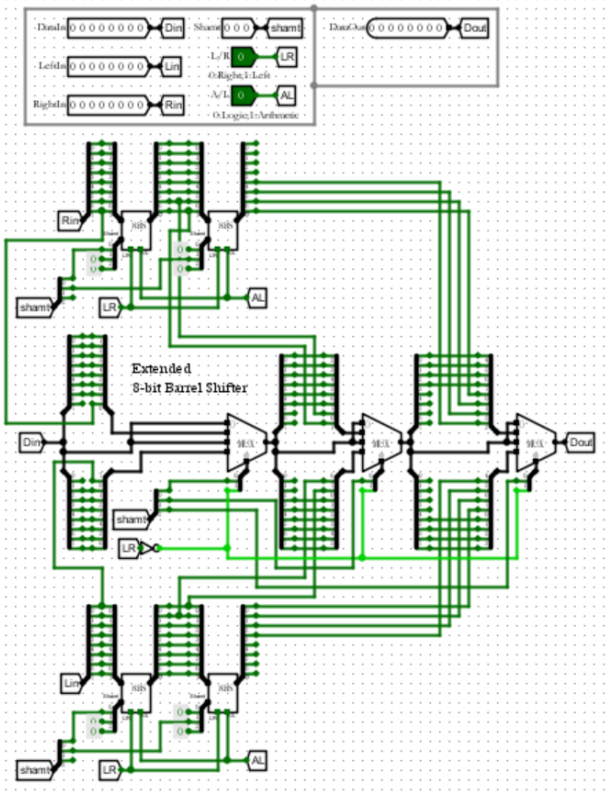

Fig. 25. Ex-8-bit Barrel Shifter

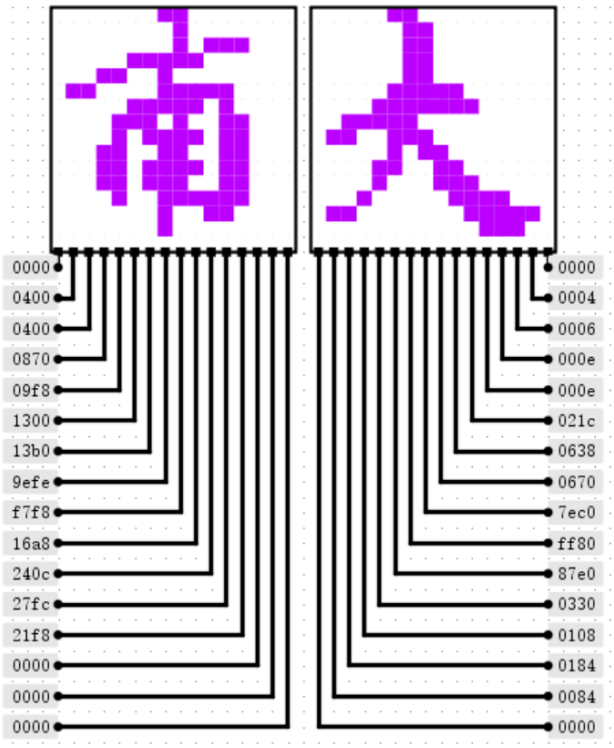

Fig. 28. Eat vege roots, make big noise!

DL&CO Experiment 3

*Author:221220089 崔博涵

I. 4-bit synchronous counter

A. Preface

Here is a Question: Why do we reverse the clock signal twice?

B. Steps

- 根据原理图添加元件
- 按照原理图连接元件，调整触发边沿，端口数和位宽
- 修改封装
- 进行仿真测试
- 保存文件

C. Theoritical Diagram

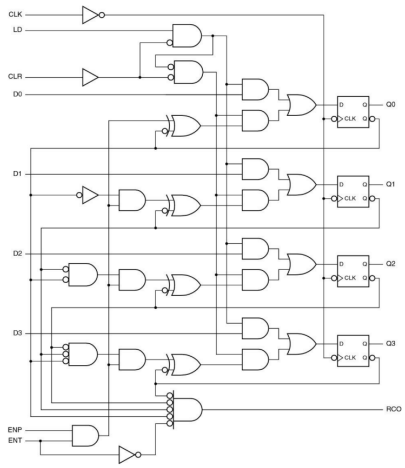

图 3.1 4 位同步二进制计数器原理图

Fig. 1. Theoritical Diagram

D. Function Table

表 3.1 4 位同步二进制计数器功能表

Inputs				Current State				Next State			
CLR	LD	ENT	ENP	Q3	Q2	Q1	Q0	Q3*	Q2*	Q1*	Q0*
1	x	x	x	x	x	x	x	0	0	0	0
0	1	x	x	x	x	x	x	D3	D2	D1	D0
0	0	0	x	x	x	x	x	Q3	Q2	Q1	Q0
0	0	x	0	x	x	x	x	Q3	Q2	Q1	Q0
0	0	1	1	0	0	0	0	0	0	0	1
0	0	1	1	0	0	0	1	0	0	1	0
						...					
0	0	1	1	1	1	0	1	1	1	1	0
0	0	1	1	1	1	1	0	1	1	1	1
0	0	1	1	1	1	1	1	0	0	0	0

Fig. 2. Function Table

Please refer to Fig.2

E. Schematic Diagram

Please refer to Fig.3 & Fig.4

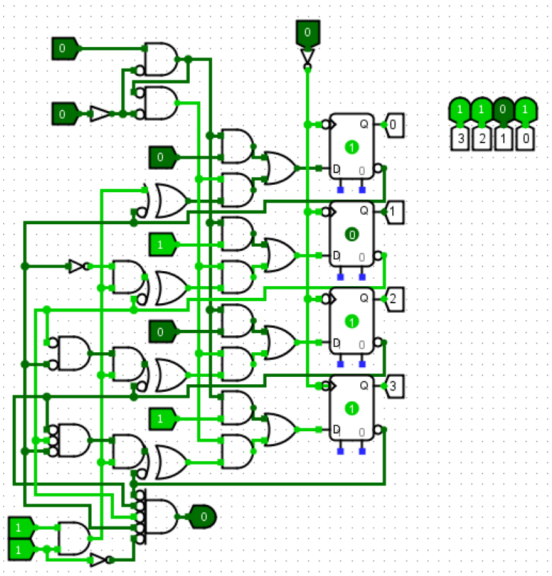

Fig. 3. Schematic Diagram

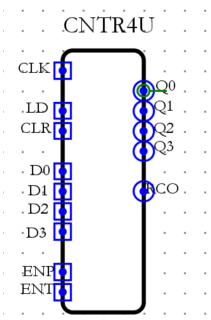

Fig. 4. Encapsulation

F. Simulation

#ATTENTION From this chapter on, the simulation has become so complicated, so I use GIF to display it. I'll attach the portal in this part.(Similarly hereinafter.)(Network needed)

Simulation GIF Click HERE.

G. Error and analysis

No error arose.

II. Clock

A. Preface

There're 2 ways to implement the clock. One is my solution, namely using the Carryin signal to be the ENABLE signal of the higher place(All the places are connected to the same clock). Another is to use the lower place to be the CLOCK signal of the higher place(Offered by my roommate). But the second one has some dilemma in the decision logic.

B. Steps

- 根据要求添加元件
- 调整触发边沿，端口数和位宽
- 按照设计连接原件
- 进行仿真测试
- 保存文件

C. Truth Table

Too complicated to display here.

D. Schematic Diagram

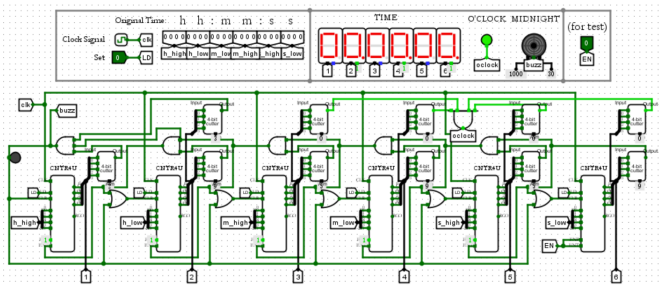

Fig. 5. Schematic Diagram

E. Simulation

Simulation GIF Click HERE.

F. Error and analysis

Return to Eight o'clock after Ten o'clock!

Connect two ports and they are blocked by the edge of the unit! (Such a common mistake in Logisim due to the f**kin' auto-connect function)

III. 4-bit shifting register & Array Generator

A. Steps

- 根据原理图添加元件
- 按照原理图连接元件，调整触发边沿，端口数和位宽
- 修改封装
- 进行仿真测试
- 实现序列生成器
- 进行仿真测试
- 保存文件

B. Function Table

功能	输入			下一个状态			
	CLR	S1	S0	Q3*	Q2*	Q1*	Q0*
清零	0	x	x	0	0	0	0
保持	0	0	0	Q3	Q2	Q1	Q0
右移	0	1	0	RIN	Q3	Q2	Q1
左移	0	0	1	Q2	Q1	Q0	LIN
装载	0	1	1	D3	D2	D1	D0

Fig. 6. Truth Table

C. Schematic Diagram

Please refer to Fig.7 & Fig.8

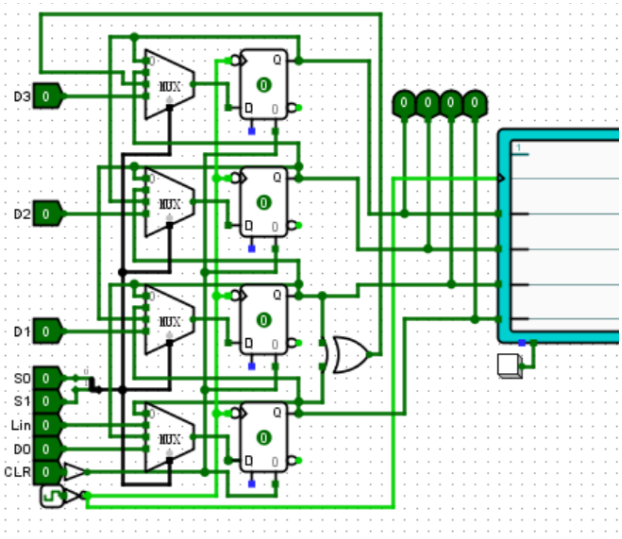

Fig. 7. Schematic Diagram

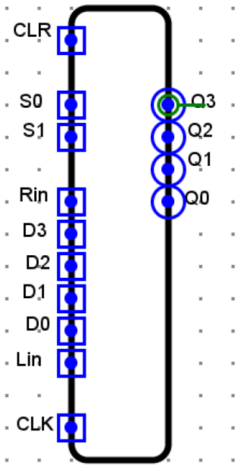

Fig. 8. Encapsulation

D. Simulation

Simulation GIF Click HERE.

E. Error and analysis

No error arose.

IV. 4-bit unsigned multiplier

A. Preface

Each multiplication spents 5 clock cycles in my implemention(1 Set+ 4 Shift & Writein). My roommate use the 9 clock cycles to finish it(1 Set + 4 Shift/ 4 Writein by turns) In my opinion, my solution works well just because the simulator omits all the delay in combinational logic. In sequential logic environment, the 9-cycle solution may perform more robustly.

If we just persue effiency in ideal environment, we could reduce the time to 4-cycle by combine the Set and the first shift into one step.

B. Steps

- 仿照 32 位原理图进行设计
- 根据设计添加元件
- 按照设计连接元件，调整触发边沿，端口数和位宽
- 进行仿真测试
- 保存文件

C. Theoritical Table

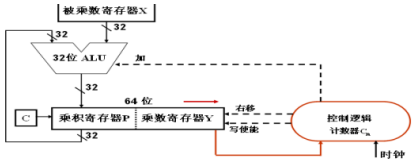

图 3.7 实现 32 位无符号数乘法运算的逻辑结构图

Fig. 9. Theoritical Table

D. Schematic Diagram

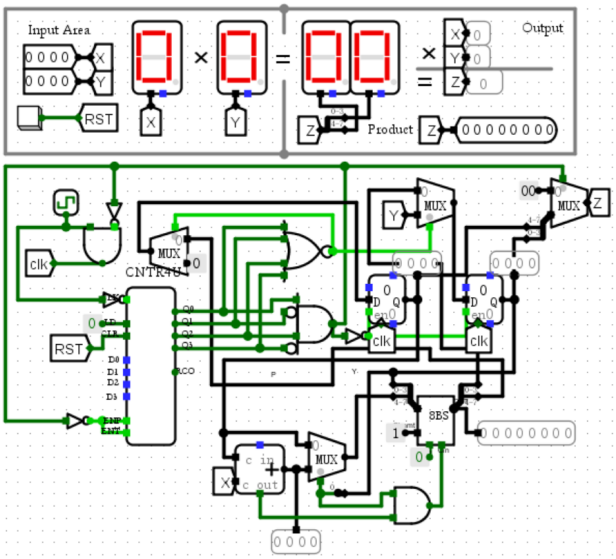

Fig. 10. Schematic Diagram

E. Simulation

Simulation GIF Click HERE.

F. Error and analysis

The output was wrong at first.

Later, I found that the multiplier should end in the 5th time rather than 4th time.

V. 32x32-bit register file

A. Preface

I adjusted the architecture carefully.
Could you feel a sense of beauty? (doge)

B. Steps

- 根据原理图添加元件
- 按照原理图连接元件，调整触发边沿，端口数和位宽
- 进行仿真测试
- 保存文件

C. Theoritical Diagram

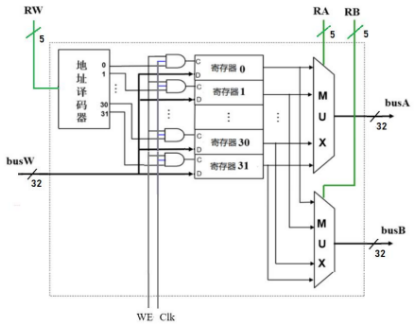

图 3.9 寄存器堆设计原理图

Fig. 11. Theoritical Diagram

D. Schematic Diagram

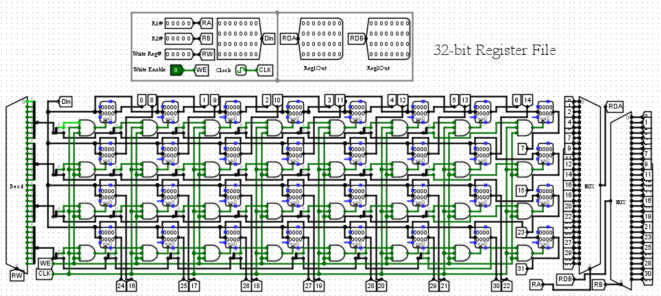

Fig. 12. Schematic Diagram

E. Simulation

Simulation GIF Click HERE.

F. Error and analysis

The output was ERROR at first. However, the ERROR is in the true bit which was stored. And I examined all the circuit without finding the mistake.

Finally, I found that I use the Input pin as the Output. Therefore the superposition of 0 and the stored '1' bit became E in the corresponding bit.

VI. Extended Questions

A. Alarm Clock

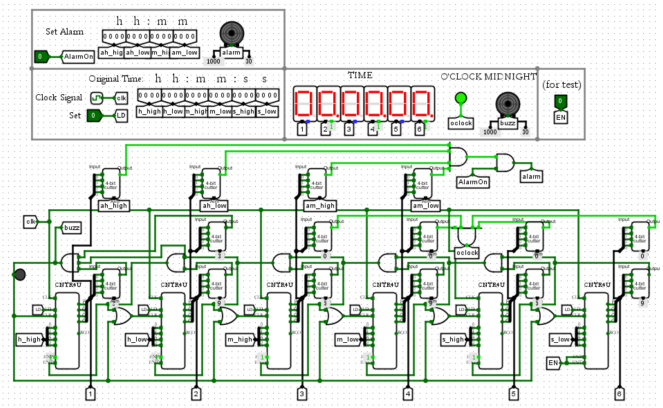

Fig. 13. Ringggggggggg

B. 4-bit Booth Signed Multiplier

Actually, Booth Algorithm seems not to be designed to support signed multiplication, instead, it aims to accelerate the signed multiplication. Signed multiplication could be done just by using the algorithmic shift in multiplier.

However, this point was not mentioned in the textbook or the lecture.

I just understand the Booth Algorithm in the surface, but I summarize some essays in my blog. Portal Here

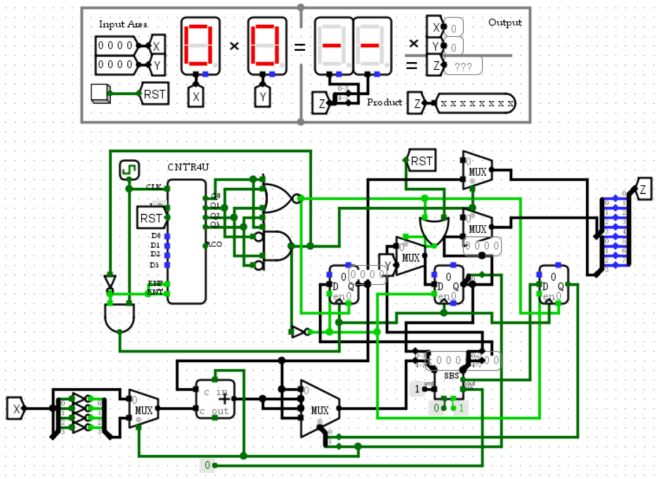

Fig. 14. ×er

C. 4-bit Unsigned Divider

This implemention is 4-bit divide 4-bit Divider. It do not support 8-bit/4-bit, for I did not refer to the textbook and DIYed this solution.(8/4 need Carryin function at the high bit, however 4/4 do not need it.)

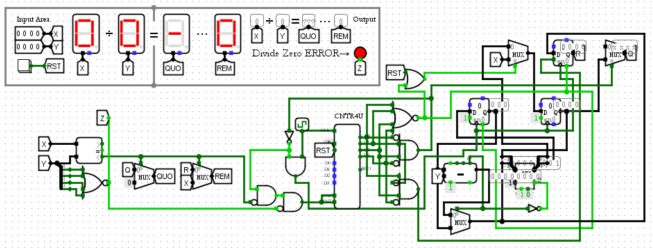

Fig. 15. ÷er

DL&CO Experiment 4

*Author:221220089 崔博涵

I. 4-Bit CLA

A. Steps

- 按照先行进位的计算原理构造 CLU
- 构造全加器
- 用全加器和 CLU 实现 CLA
- 修改封装
- 进行仿真测试，保存文件

B. Theoritical Diagram

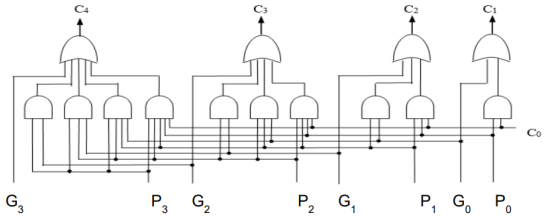

Fig. 1. Theoritical Diagram of CLU

C. Schematic Diagram

Please refer to Fig.2 & Fig.3 & Fig.4.

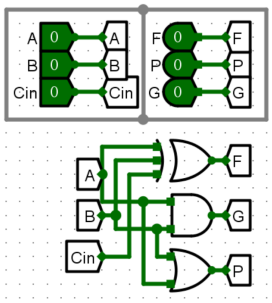

Fig. 2. FA

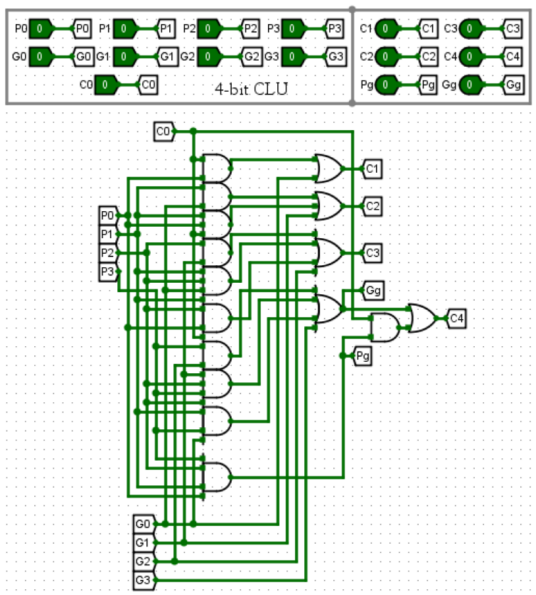

Fig. 3. CLU

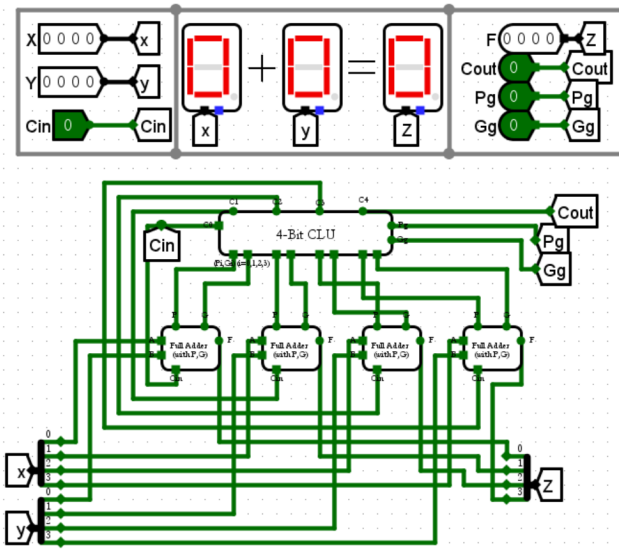

Fig. 4. CLA

D. Simulation

Please refer to Fig.5

E. Error and analysis

At first the outputs are wrong. It is because the default logic of XOR gate in Logisim should be altered.

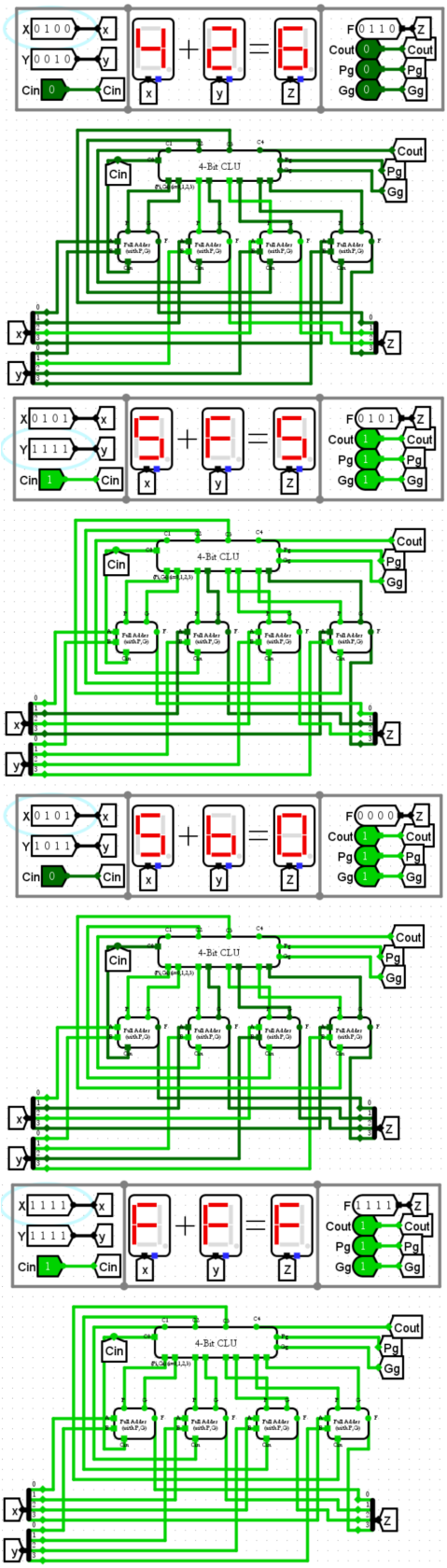

Fig. 5. Simulation

II. 16-Bit BCLA

A. Steps

- 根据原理图级联 4-Bit CLA
- 修改封装
- 进行仿真测试，保存文件

B. Theoritical Diagram

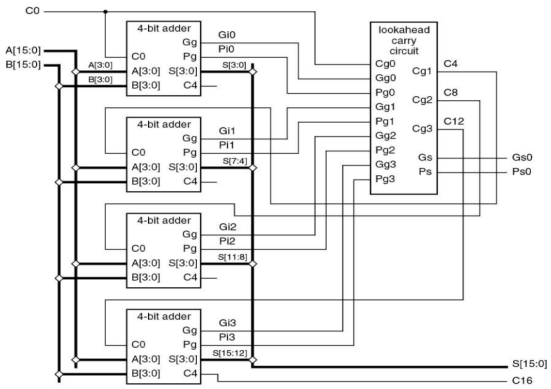

Fig. 6. Theoritical Diagram

C. Schematic Diagram

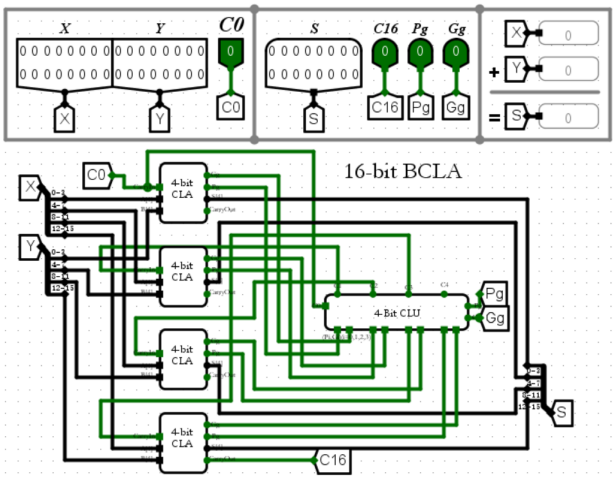

Fig. 7. Schematic Diagram

D. Simulation

Please refer to Fig.8

E. Error and analysis

No error arose.

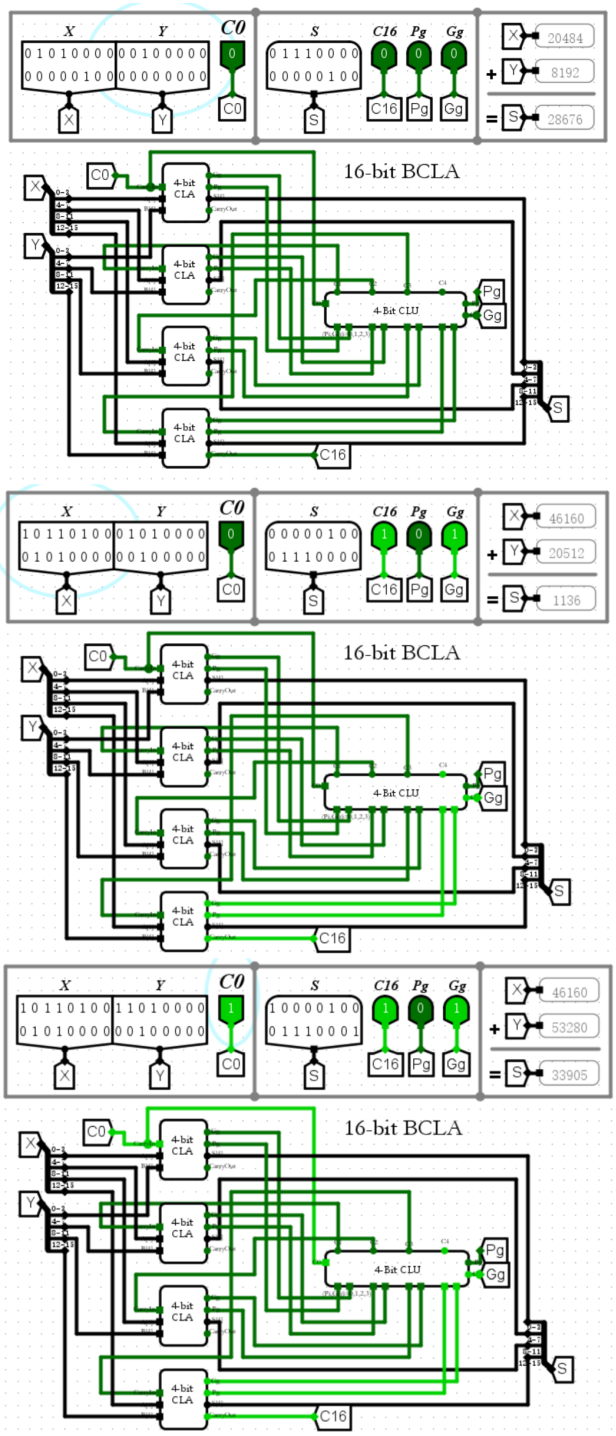

Fig. 8. Simulation

III. 32-Bit Adder(with flag)

A. Steps

- 实现 ZF 判断子电路
- 根据原理图级联 16-Bit BCLA
- 根据原理实现 flag
- 修改封装
- 进行仿真测试，保存文件

B. Theoritical Diagram

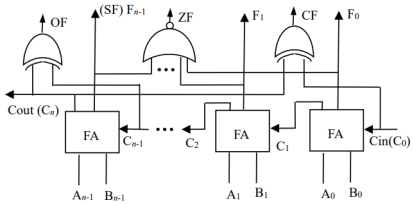

Fig. 9. Theoritical Diagram

C. Schematic Diagram

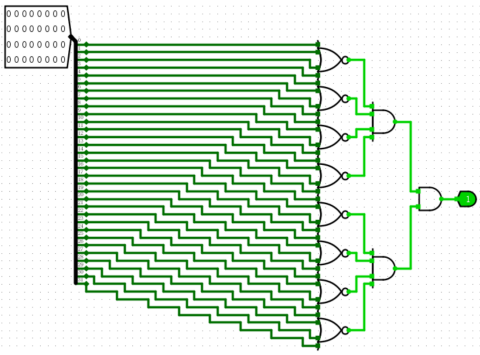

Fig. 10. ZF

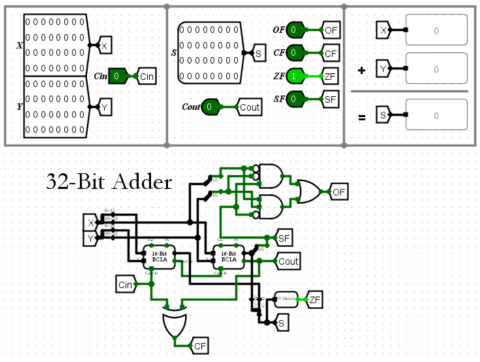

Fig. 11. Schematic Diagram

D. Simulation

Please refer to Fig.12

E. Error and analysis

No error arose.

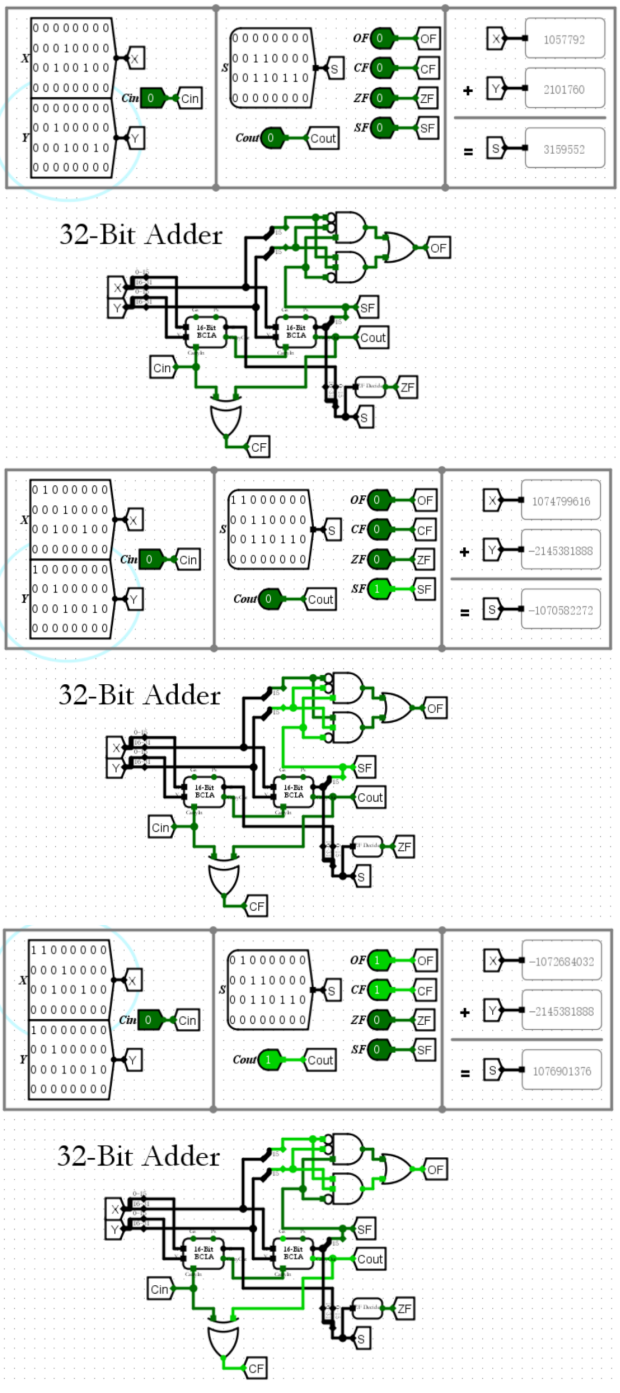

Fig. 12. Simulation

IV. 32-bit ALU

A. Steps

- 根据逻辑表达式实现控制部件（使用“组合电路分析功能”直接生成）
- 导入 32-Bit BS
- 导入 32-Bit Adder
- 按照原理图连接数据通路
- 修改封装
- 进行仿真测试，保存文件

B. Theoritical Diagram

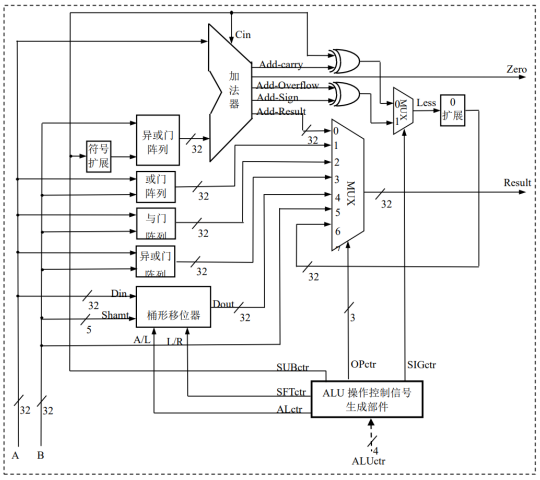

Fig. 13. Theoritical Diagram

C. Schematic Diagram

Please refer to Fig.14 & Fig.15

D. Simulation

#ATTENTION The simulation has become so complicated, so I use GIF to display it. I'll attach the portal in this part.(Similarly hereinafter.)(Network needed)

Algorithmic Manipulations GIF Click HERE.

Logical Manipulations GIF Click HERE.

E. Error and analysis

1. The ops are all in a fuss. Error: In the control unit, the OPctr output pins are in the reverse order.

2. The SUB op is invalid. Error: In combinatorial logic generation, I mistake 1000 for 0100 as 8.

3. The Shift is invalid.(5 to 7 bits) Error: In the 32BS, I overturn the high 4 bits in the third bits transmission.

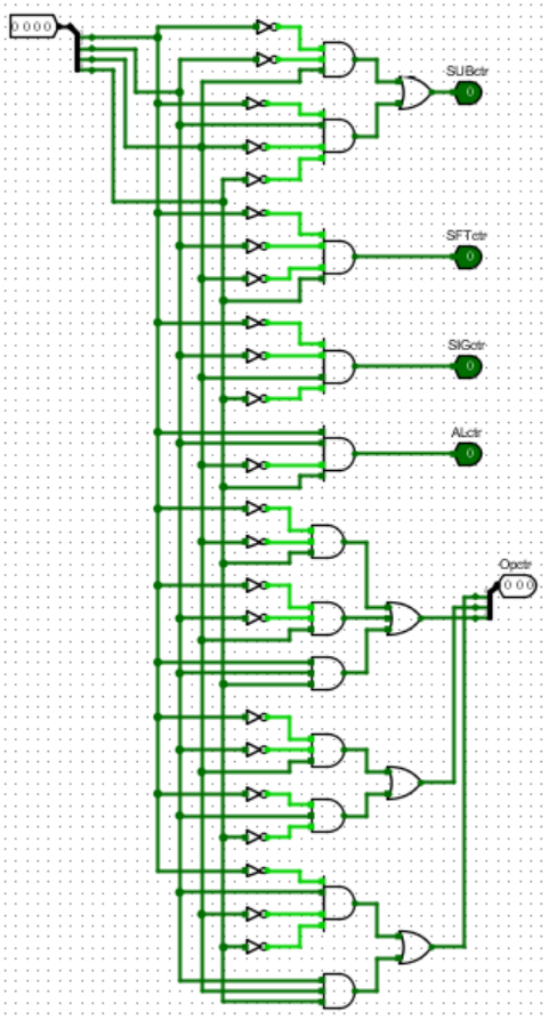

Fig. 14. Control Unit

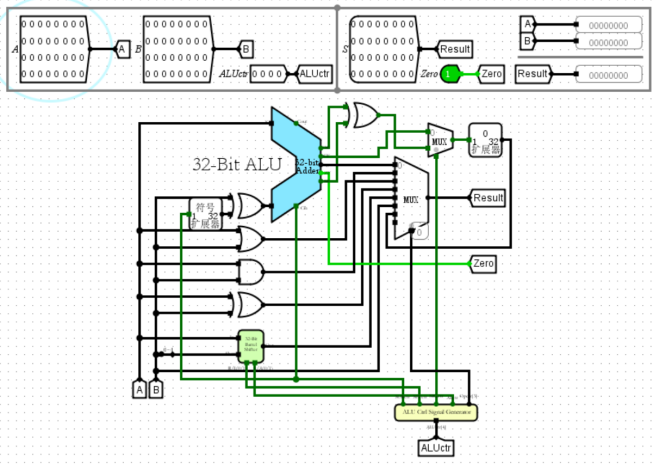

Fig. 15. Schematic Diagram

V. Extended Questions

A. *Exemplify multiplication*

假设常量为 101, 被乘数为 0x00000011(设存在 R1 里)，乘积寄存器 (R2)

读：R1(0x00000011),R2(0x00000000)
操作：add(op:0000)
写：R2
读：R1(0x00000011),imm(0x00000001)
操作：slli(op:0001)
写：R1
读：R1(0x00000022),imm(0x00000001)
操作：slli(op:0001)
写：R1
读：R1(0x00000044),R2(0x00000011)
操作：add(op:0000)
写：R2
结果为 R2(0x00000055)

B. *Additional addition*

并行前缀加法器 (Parellel Prefix Adder)
生成进位的核心是这个逻辑函数

$$c_i = g_{[0,i]} + \left(p_{[0,i]} \cdot c_{-1}\right)$$
$$s_i = p_i \oplus c_i$$

而任何相邻的 g,p 函数均可以合并（满足结合律）

$$g_{[i_0,i_2]} = g_{[i_1,i_2]} + \left(g_{[i_0,i_1-1]} \cdot p_{[i_1,i_2]}\right)$$
$$p_{[i_0,i_2]} = p_{[i_0,i_1-1]} \cdot p_{[i_1,i_2]}$$

因此等价于求数列前缀和，可以通过不同的方式加括号，利用中间结果来求和。（此处是硬件开销，扇入数，延迟的三方权衡）

不同的结构可以用“并行前缀图”来表示

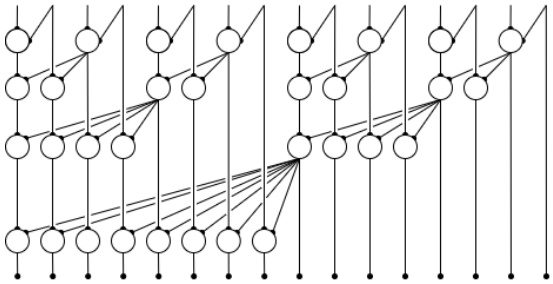

Fig. 16. Ladner-Fischer Structure

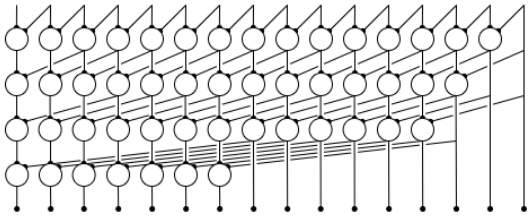

Fig. 17. Kogge-Stone Structure

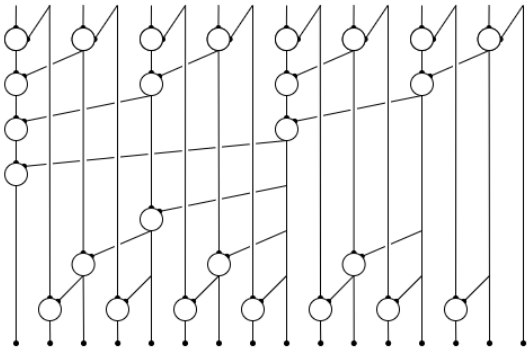

Fig. 18. Brent-Kung Structure

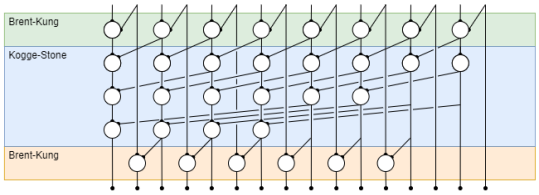

Fig. 19. Han-Carlson Structure(Mixed)

此处实现了一个 4-Bit 的 LF 结构的加法器 (电路文件 4-bit PPA(LF).circ)

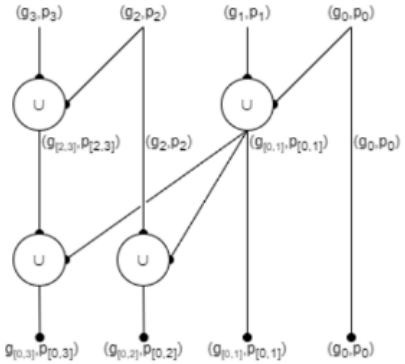

Fig. 20. Prefix Diagram

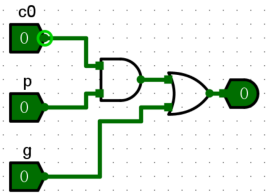

Fig. 21. PG2C

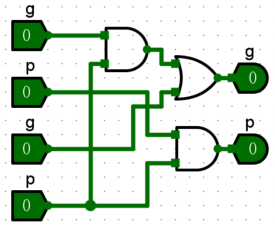

Fig. 22. GP Merge

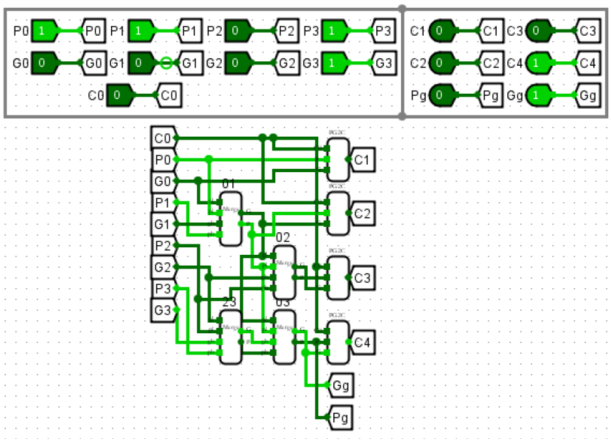

Fig. 23. Control Unit

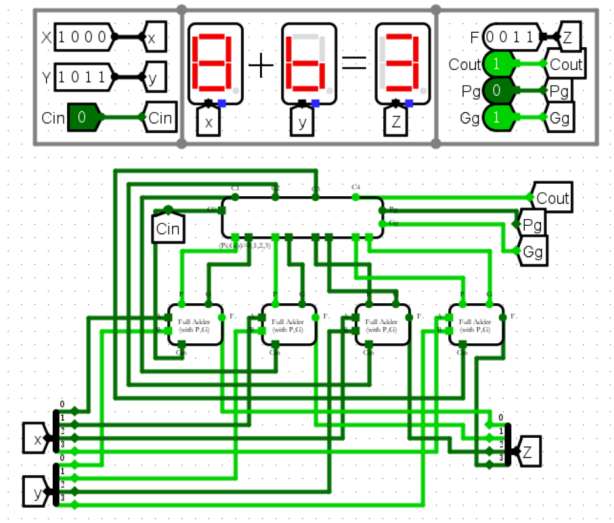

Fig. 24. Schematic Diagram

C. Gate count

加法运算

先考虑加法器计算过程。

1. 全加器算出最小的 p,g 1 级
2.4-bit CLU 算出 4 位的 P,G 2 级
3. 外层 CLU 算出各外层 C 2 级
（此时算出 C16）
（第二个 16 位此时算完第 2 步等待）
5. 第二个 16 位算出各外层 C 2 级
6. 内层 CLU 用 C 算出各小 c 2 级
7. 全加器算出结果 1 级
共 10 级
加上前面的异或和后面的 8-1 选择器，共 12 级

比较运算

带符号数
相比加法，生成 OF 额外需要 2 级，额外的抑或和 2-1 多路选择器再多 2 级
共 16 级

无符号数
在 ALU 中进行到上述步骤 7+ 一级异或门生成 CF+ 外面的异或门 + 外面的 8-1 和 2-1
共 11 级

移位

我的桶形移位器涉及多级嵌套，情形过于复杂
此处计算常规 10 个选择器的桶形移位器。
该种共 11 级门延迟。

DL&CO Experiment 5

*Author:221220089 崔博涵

I. RAM

A. Steps

- 根据指令定义，连接电路
- 写入数据
- 进行仿真测试
- 修改封装
- 保存电路

B. Schematic Diagram

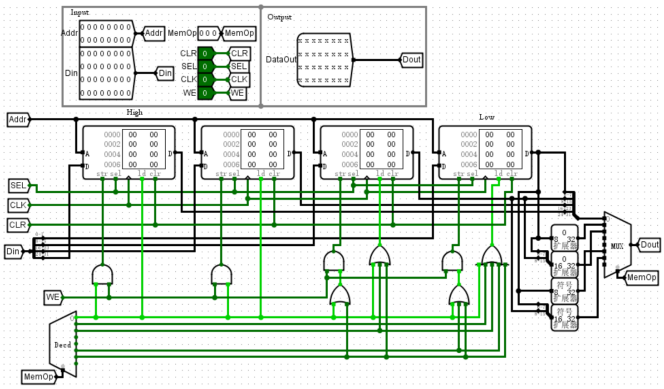

Fig. 1. Schematic Diagram

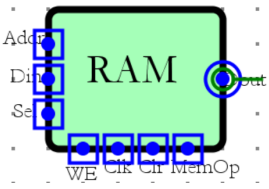

Fig. 2. Encapsulation

C. Simulation

#ATTENTION From this chapter on, the simulation has become so complicated, so I use GIF to display it. I'll attach the portal in this part.(Similarly hereinafter.)(Network needed)

Simulation GIF Click HERE.

D. Error and analysis

No error arose.

II. ASCII

A. Steps

- 根据指令定义，连接电路
- 写入数据
- 进行仿真测试
- 保存电路

B. Schematic Diagram

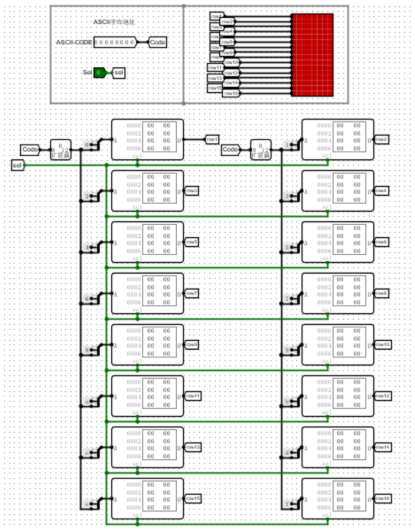

Fig. 3. Schematic Diagram

C. Simulation

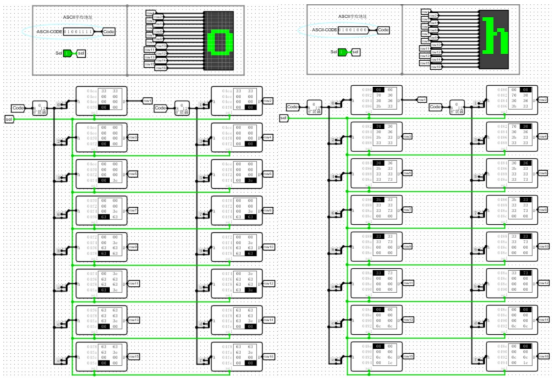

Fig. 4. Schematic Diagram

D. Error and analysis

No error arose.

III. IFU

A. Steps

- 根据指令定义，连接电路
- 写入数据
- 进行仿真测试
- 修改封装
- 保存电路
- 删去 ROM，制作封装版的 IFU

B. Structure

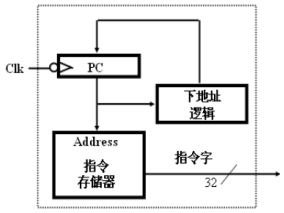

Fig. 5. Structure

C. Schematic Diagram

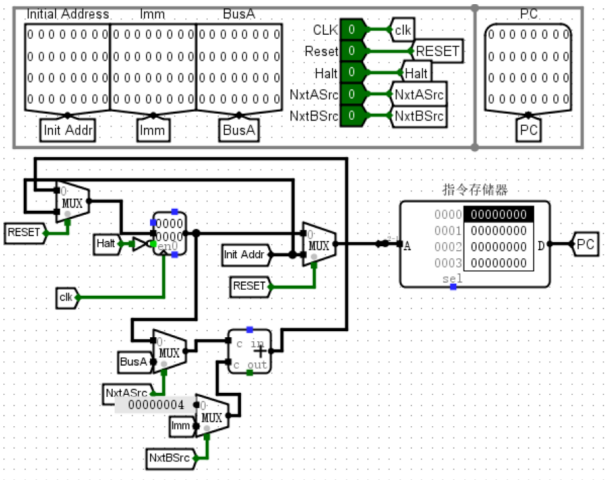

Fig. 6. Schematic Diagram

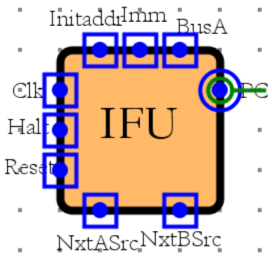

Fig. 7. Encapsulation

D. Simulation

Simulation GIF Click HERE.

E. Error and analysis

No error arose.

IV. Datapath

A. Preface

I thought that the given datapath cannot reveal the notion 'bus', for it connect the bus seperately(And use "DataIn" for "BusB"). So I mark 3 layers of the datapath.(Interior Layer, Bus Layer and Interface Layer)

B. Steps

- 根据指令定义，连接电路，构造立即数拓展器
- 根据指令定义，连接电路，构造 Branch
- 根据指令定义，连接电路，构造 IDU
- 修改封装
- 根据原理图，构造数据通路
- 进行仿真测试
- 保存文件

C. Theoritical Diagram

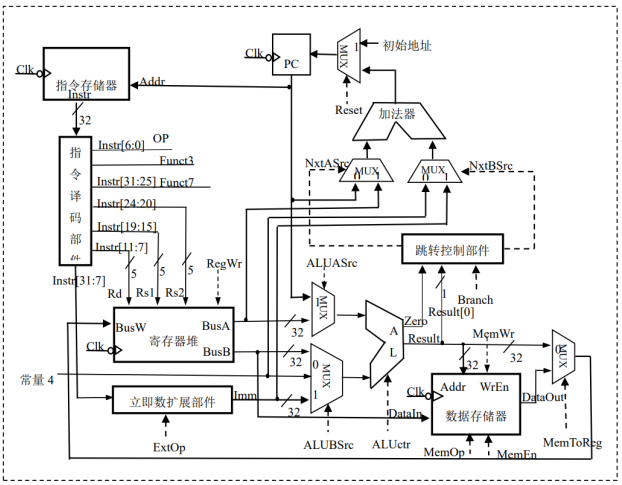

Fig. 8. Theoritical Diagram

D. Schematic Diagram

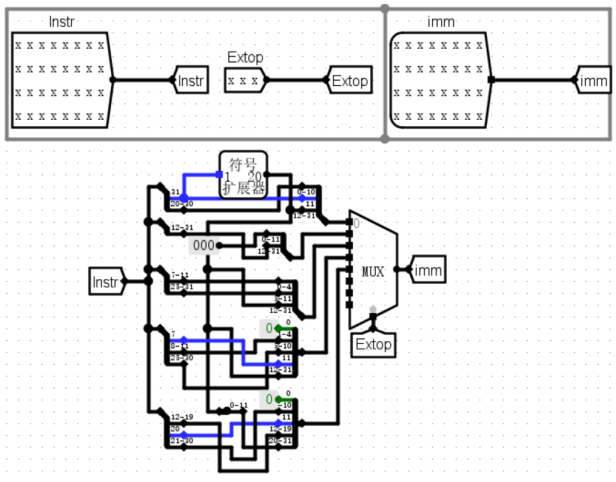

Fig. 9. Schematic Diagram

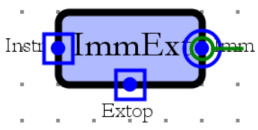

Fig. 10. Encapsulation

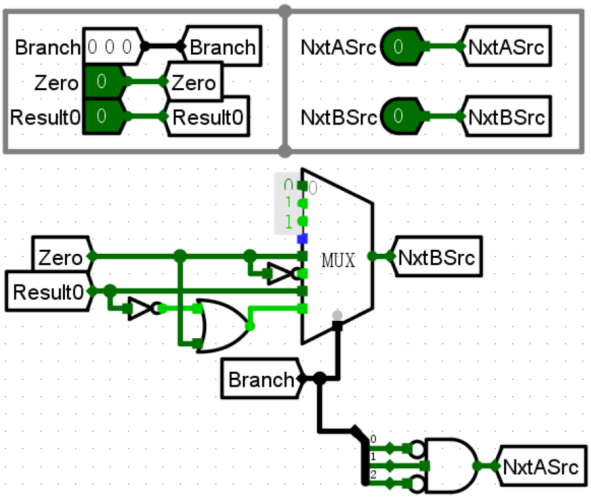

Fig. 11. Schematic Diagram

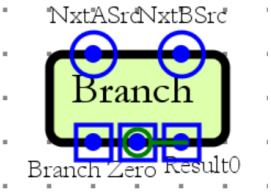

Fig. 12. Encapsulation

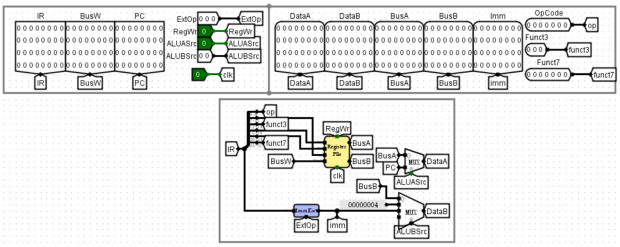

Fig. 13. Schematic Diagram

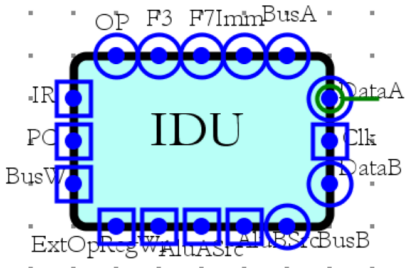

Fig. 14. Encapsulation

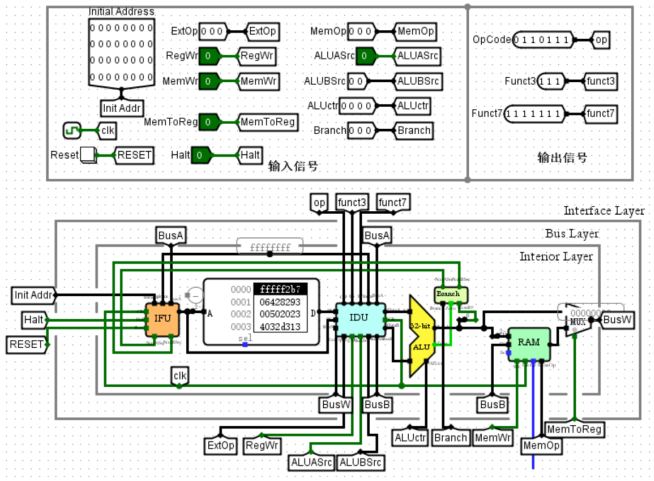

Fig. 15. Schematic Diagram

E. Simulation

The simulation has been done in the lab class. So I check the given sample here. The sample output seem to be wrong in a number.(Mark by red on the pic, being 0xfffffb8 originally.)

BusW	立即数	寄存器堆	数据存储器
0xfffff000	0x fffff000	X5=0xfffff000	
0xffff064	0x0000064	X5=0xfffff064	
0x00000000	0x00000000	X5=0xfffff064	0: 0xfffff064
0xfffffe0c	0x00000403	X6=0xfffffe0c	
0x00000004	0x00000004	X6=0xfffffe0c	1: 0xfffffe0c
0xfffffe0c	0x00000004	X7=0xfffffe0c	
0x000001f3	0xffffffff	X8=0x000001f3	
0x00000008	0x00000008	X8=0x000001f3	2: 0x000001f3
0x00000000	0x00000007	X9=0x00000000	
0x0000000c	0x0000000c		3:0x00000000
0x00000000	0xfffffffd8		
0x00000030	0x00000030	X10=0x00000030	
0x00000010	0x00000010		4: 0x00000030
0x00064034	0x00064000	X11=0x00064034	

Fig. 16. Comparison Table

F. Error and analysis

No error arose.

V. Extended Questions

A. iNJUCS Ideology

A bit weird to shift to the right(doge).

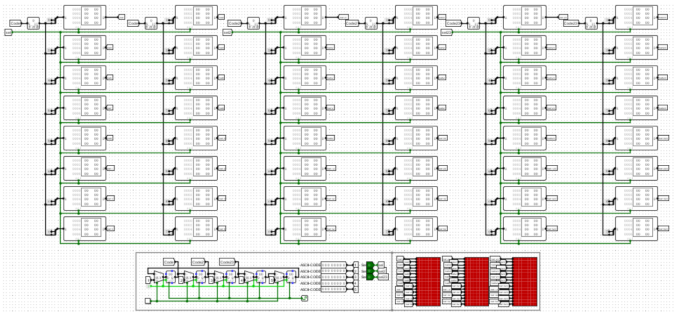

Fig. 17. Schematic Diagram

I choose "CSnb!" to be the content(doge).

Simulation GIF Click HERE.

B. Label

为 -40

$Imm[12:1]$ 为 111111101100,

$Value(Sext(Imm)) = -20$

$bias = -20 \times 2 = -40$

C. Pseudo-instruction

伪指令是指不是真正的指令，其实就是汇编语言中的 syntactic sugar，在汇编过程中会被汇编器替换成对应的一条或多条指令。

举例: ret 会被汇编为 $jalr\ x0, x1, 0$

ret(return) 返回指令此处替换为 jalr(jump&link register) 将 PC+4 存入 x0，因为 RISC-V 中 x0 硬编码为 0，此处即为丢弃，再将 x1 中的值加偏移量 0 存入 PC，即返回到 x1 中的地址，x1 在 RISC-V 中为专用的返回地址寄存器。

D. Halt

其实在实验 6 里已经给出了。在控制部件中增加一个 Halt 信号，当 Halt 信号为 1 时，将 PC 的值保持不变，Halt 信号由 Opcode 全零产生。

DL&CO Experiment 6

*Author:221220089 崔博涵

I. Control Unit

A. Steps

- 根据指令表写出相应的组合逻辑表达式
- 进行化简
- 根据化简后的逻辑表达式连接电路图（部分步骤用组合逻辑电路分析生成子电路）
- 进行仿真测试
- 修改封装
- 保存文件

B. Schematic Diagram

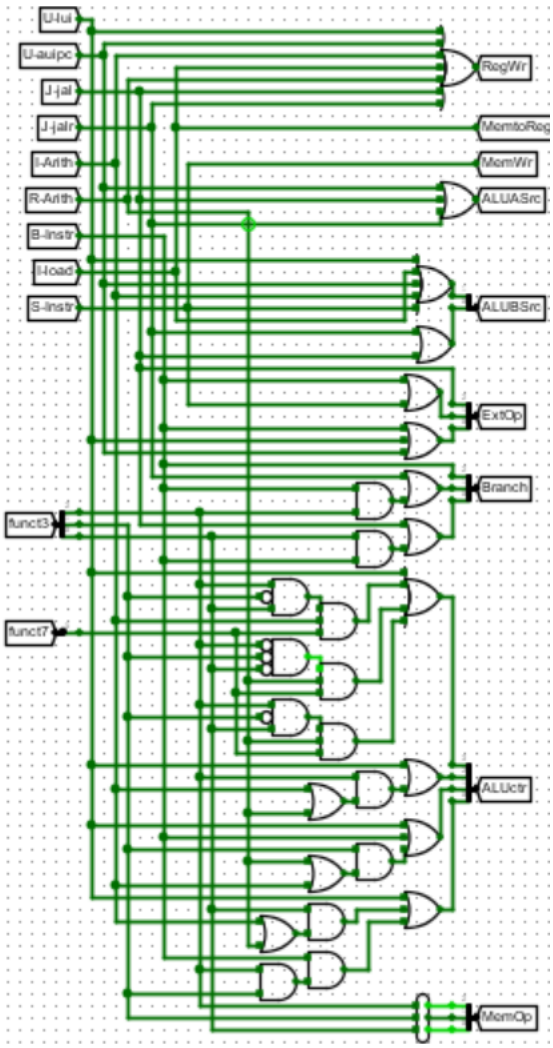

Fig. 1. Schematic Diagram

Fig. 2. Encapsulation

C. Simulation

Please refer to Fig.2

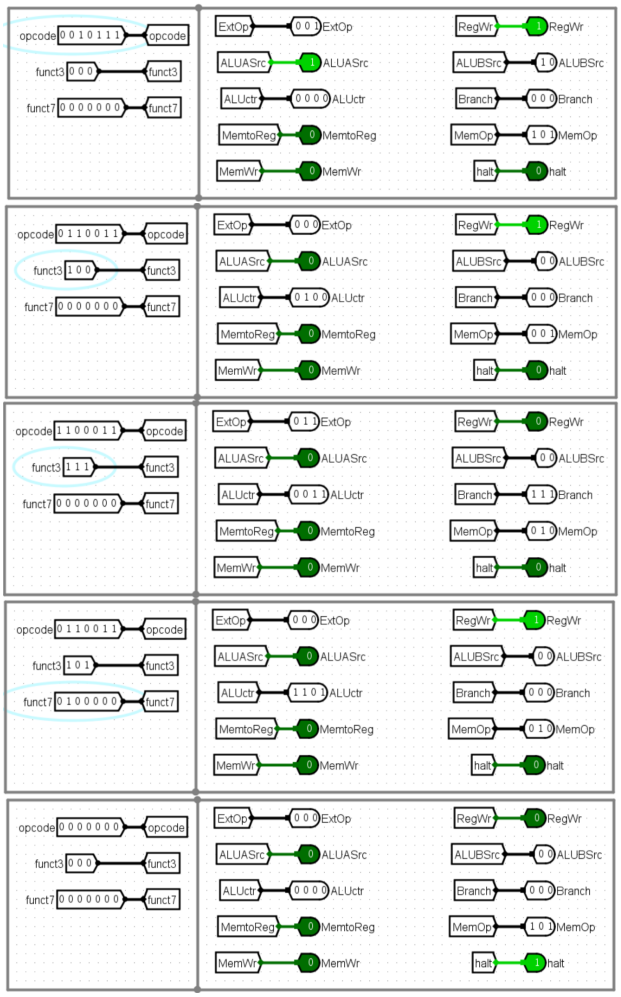

Fig. 3. Simulation

D. Error and analysis

No error arose.

II. CPU

A. Preface

According to the macro-achitecture of the CPU, I encapsulate the 'Datapath' Part individually to reuse the datapath finished in Experiment 5.

B. Steps

- 连接各个信号线
- 导入 ROM, 进行三项仿真测试
- 保存文件

C. Schematic Diagram

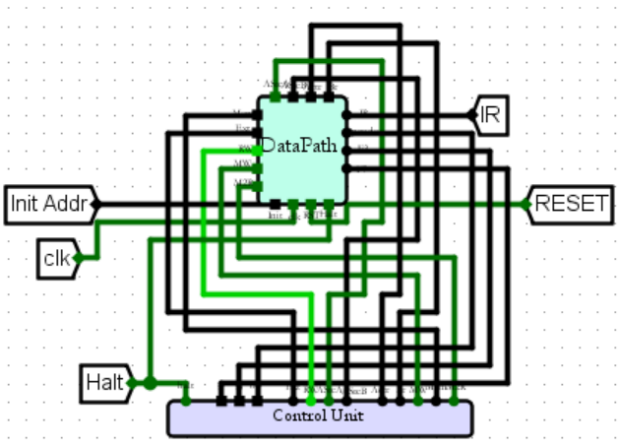

Fig. 4. Schematic Diagram

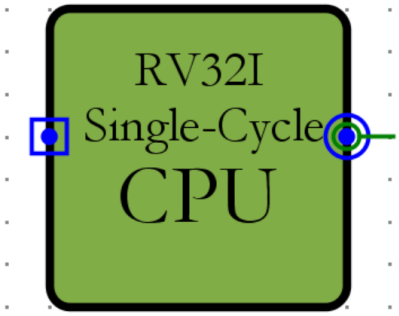

Fig. 5. Encapsulation

D. Simulation

a) Accumulator Test: Please refer to Fig.6

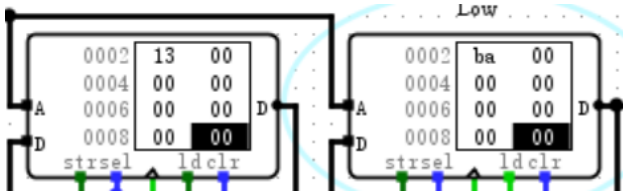

Fig. 6. Final RAM State

0000	0a	02	05	06	07	08	09	12	36	41	5b	00	00	00	00	00

Fig. 7. Final RAM State

b) Bubble Sort Test: Please refer to Fig.7

c) C Test: Please refer to Fig.8

0000	00	00	ff	10	00	00	ff	00	00	0e	13	04	01	00	02	00
0010	00	00	00	00	00	fe	ff	ff	00	00	fc	00	fe	00	00	01
0020	01	00	13	f9	13	00	01	00	00	40	00	00	f6	ff	00	f5
0030	00	f6	01	00	00	40	00	00	f3	fe	00	f2	01	01	01	01
0040	00	00	02	00	18	00	ff	13	00	f4	18	00	00	00	00	00
0000	00	00	c1	c0	05	00	01	10	11	c0	40	c5	30	05	b0	07
0010	47	e6	d7	e7	47	c7	f5	c6	a6	07	df	00	01	91	81	21
0020	31	11	40	df	40	00	40	04	44	85	15	14	1f	34	10	5f
0030	00	5f	40	09	49	85	15	14	1f	94	10	5f	c1	81	41	01
0040	c1	00	01	00	a0	00	01	00	11	df	a0	00	00	00	00	00
0000	04	91	01	00	04	80	01	05	26	00	05	06	05	07	50	a7
0010	a6	d6	a0	a2	87	94	85	07	06	86	f0	80	01	2a	2c	28
0020	26	2e	09	f0	04	04	09	a5	84	05	35	04	f0	14	05	f0
0030	04	f0	04	25	09	05	35	04	f0	14	05	f0	20	24	24	29
0040	29	05	01	80	22	00	01	05	26	f0	22	00	00	00	00	00
0050	00	00	00	00	00	01	03	03	03	04	09	0c	14	1e	22	46
0000	13	17	13	ef	63	67	13	13	23	ef	13	13	93	93	63	03
0010	83	63	23	23	93	e3	93	93	63	13	6f	67	13	23	23	23
0020	23	23	13	ef	93	13	93	03	93	33	13	13	ef	e3	13	ef
0030	13	ef	93	03	13	33	13	13	ef	e3	13	ef	83	03	83	03
0040	83	13	13	67	23	6f	13	13	23	ef	23	6f	00	15	21	27
0050	4c	57	aa	be	c8	90	20	8e	91	bc	97	2a	5f	1f	0c	50
0060	30	01	00	00	00	00	00	00	00	00	00	00	00	00	00	00

Fig. 8. Final RAM State

E. Error and analysis

No error arose.

III. Extended Questions

A. Overflow Detection

Adapt the source code.

```
main:
        lw a0,0(x0)
        addi a2, x0,1
        add a3,x0,x0
loop:
        add a4,a3,a2
        blt a4,a3, finish
        add a3, a4, x0
        beq a2, a0, finish
            addi a2, a2, 1
           jal x0, loop
finish:
        sw a3, 8(x0)
        addi a2, a2, -1
        sw a2, 16(x0)
end:
        jal x0, end
```

Fig. 9. Source code

Due to the hardware of my PC, the frequency of the CPU in Logisim is only 100Hz at the maximum, which means it will execute the programs for about 65535*5/100 s=about 1h (main loop). So I just use the RARS to simulate the program. The outputs are below.

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)
0x00000000	0x00666664	0x00000000	0x7fff8000	0x00000000	0x0000ffff

Fig. 10. RARS output

According to the data in (+8) and (+10), check the output.

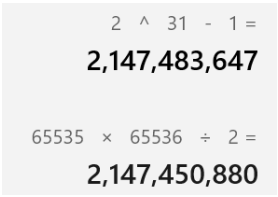

Fig. 11. check

B. Individual comparator

在目前 RV32I 指令集下，B 型指令来源均相同，做如下改动即可：

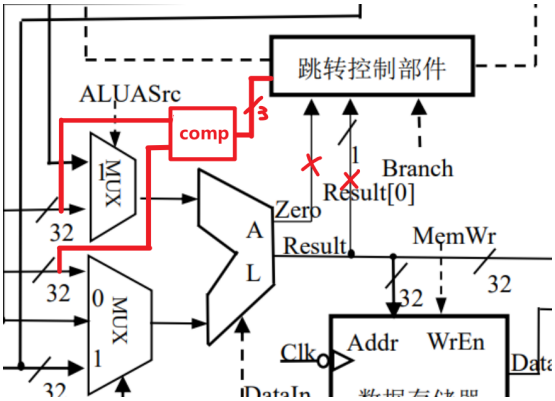

Fig. 12. Adaption

C. I/O

I/O 设备和 CPU 通过接口相互通信。

接口由若干个寄存器组成。

主要有两种方式：

其一是 MMIO，即将 I/O 设备的寄存器映射到内存地址空间中，CPU 通过访问内存地址来访问 I/O 设备。在本实验中，可以通过将 RAM 中的部分与键盘、TTY 组件的寄存器关联来实现。（在 Logisim 中不易实现）

其二是 PMIO，即增加特殊的指令，通过指令来访问独立编址的寄存器。在本实验中，增加额外的控制信号选择主存或键盘、TTY 组件的寄存器即可。

D. Pipeline

实现流水线大致需要完成以下步骤

- 将数据通路分为五个流水段。
- 修改数据通路设置流水线分段寄存器。
- 设计流水线控制器。
- 设置处理数据冒险，结构冒险，控制冒险的修改。（或者进行转发等更高级处理技术的实现）