同济大学计算机系

数字逻辑课程实验报告

学　　号 2253156

姓　　名 闫浩扬

专　　业 计算机科学与技术（精英班）

授课老师 郭玉臣

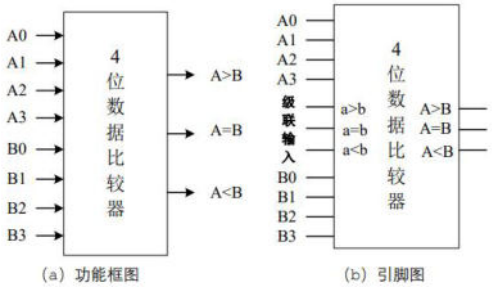
一、实验内容

- **实验介绍**
  在本次实验中，我们将使用 Verilog HDL 语言实现 4 位比较器、8 位比较器、无符号加法器和有符号加法器的设计和仿真。
- **实验目标**
  深入了解比较器和加法器的原理。
  学习使用 Verilog HDL 语言设计 4 位比较器和 8 位比较器。
  学习使用 Verilog HDL 语言设计无符号加法器和有符号加法器。

二、硬件逻辑图

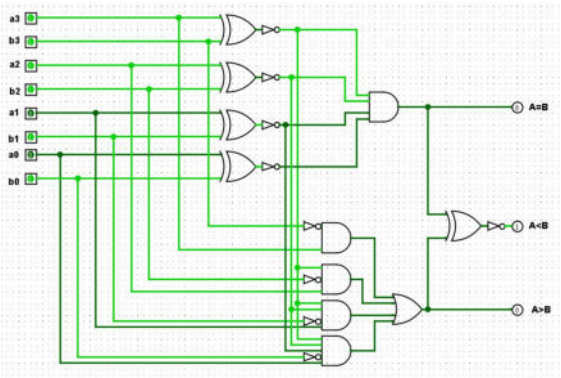
1) 数据比较器 DataCompare4

用来完成两组二进制数大小比较的逻辑电路，称为数据比较器。图 1 给出了 4 位二进制数据比较器的功能框图。输入是一组二进制数 A3 A2 A1 A0（A3 为高位）和另一组二进制数 B3 B2 B1 B0（B3 为高位），两组数比较的结果，只能是输出 A>B，A=B，A<B 三种情况的一种。图 2 给出了 4 位二进制数据比较器芯片 74HC85 的逻辑符号与引脚图，其真值表如表 1 所示。



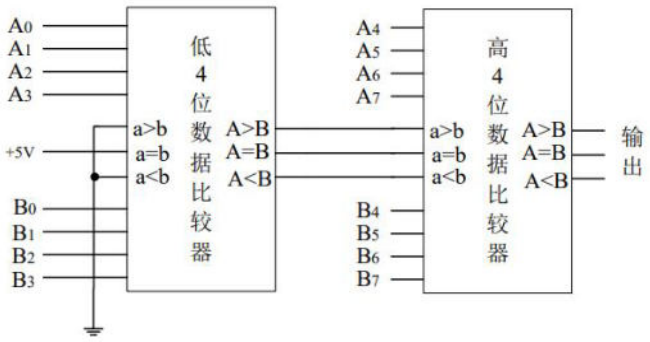
比较输入				级联输入			输出		
a3b3	a2b2	a1b1	a0b0	a>b	a<b	a=b	A>B	A<B	A=B
a3>b3	×	×	×	×	×	×	1	0	0
a3<b3	×	×	×	×	×	×	0	1	0
a3=b3	a2>b2	×	×	×	×	×	1	0	0
a3=b3	a2<b2	×	×	×	×	×	0	1	0
a3=b3	a2=b2	a1>b1	×	×	×	×	1	0	0
a3=b3	a2=b2	a1<b1	×	×	×	×	0	1	0
a3=b3	a2=b2	a1=b1	a0>b0	×	×	×	1	0	0
a3=b3	a2=b2	a1=b1	a0<b0	×	×	×	0	1	0
a3=b3	a2=b2	a1=b1	a0=b0	1	0	0	1	0	0
a3=b3	a2=b2	a1=b1	a0=b0	0	1	0	0	1	0
a3=b3	a2=b2	a1=b1	a0=b0	0	0	1	0	0	1

logisim 逻辑图

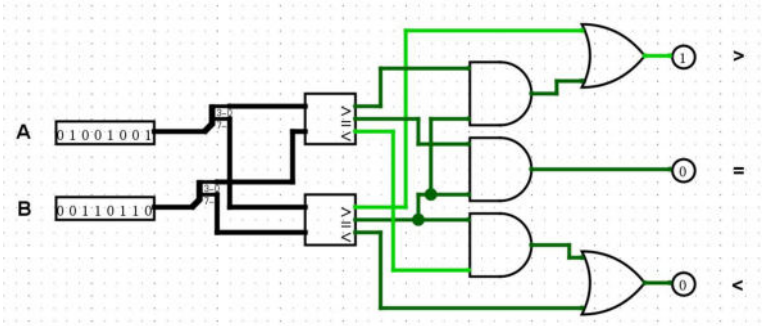


2) 数据比较器 DataCompare8

当比较位数超过 4 位时，可以将两片或多片 74HC85 级联使用。如图所示为 2 片 4 位比较器级联而成的 8 位比较器，此时低 4 位和高 4 位输入信号，分别加到两个比较器的输入端，低 4 位比较器的三个输出分别对应接到高 4 位比较器的三个级联输入端 a>b，a<b，a=b。比较结果由高 4 位比较器输出端输出。

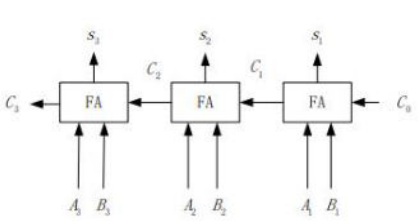


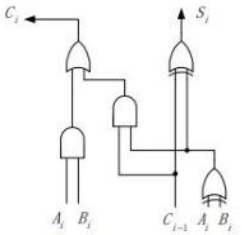
logisim 逻辑图



3) 1 位加法器 FA

加法器是计算机或其他数字系统中对二进制数进行运算处理的组合逻辑构件。本实验主要采用 Verilog HDL 行为描述实现串行加法器，其逻辑结构框图如图所示，它由多个全加器（FA）串行连接而成。每一个全加器是一位加法器，其逻辑图如下，有三个输入（加数Ai，被加数Bi，低位的进位信号Ci−1），两个输出（和数Si，向高 位的进位信号Ci）

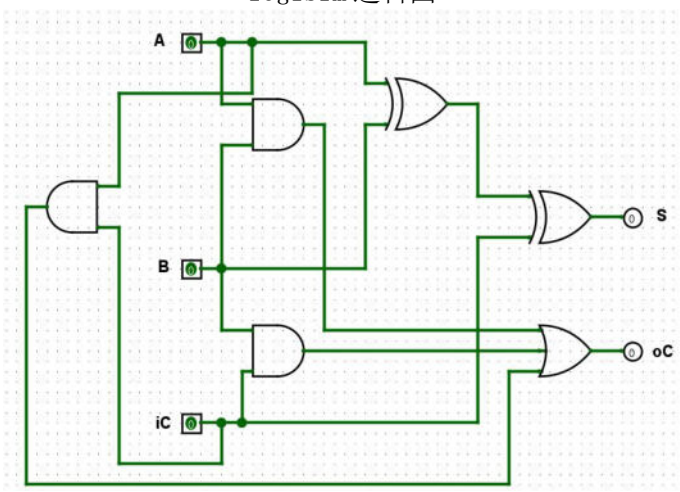




真值表

A	B	CI	S	CO
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

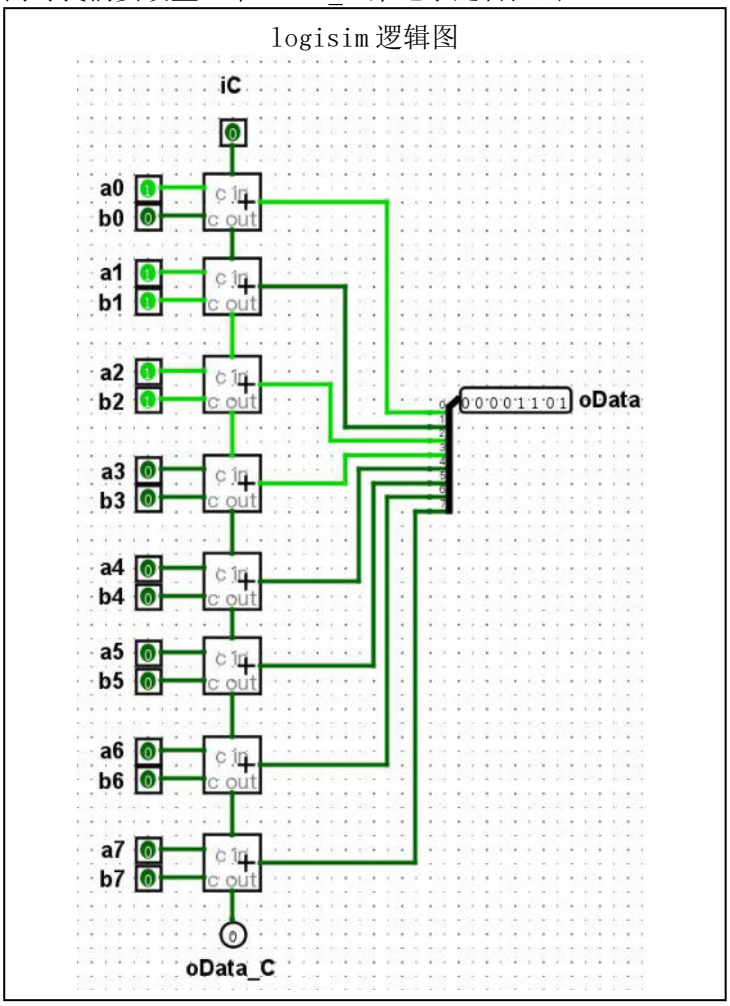
logisim 逻辑图



4) 8 位加法器 Adder

上面我们已经将一位无符号加法器 FA 实现，我们可以将多个 FA 级联起来，从而实现 8 位加法器，同时我们要设置一个 oData_C 来记录是否溢出。

logisim 逻辑图



三、模块建模

（一）4 位比较器 DataCompare4

接口定义

```
module DataCompare4(
input [3:0] iData_a, //输入数据 a
input [3:0] iData_b, //输入数据 b
input [2:0] iData, //级联输入 a>b、a<b，a=b
output [2:0] oData //结果输出 A>B、A<B，A=B
);
```

Verilog 代码描述

```
`timescale 1ns / 1ps
module DataCompare4(
    input [3:0] iData_a,
    input [3:0] iData_b,
    input [2:0] iData,
    output reg [2:0] oData
    );
    always @ *
    begin
        if(iData_a==iData_b)
            oData=iData;
        else if(iData_a<iData_b)
            oData=3'b010;
        else if(iData_a>iData_b)
            oData=3'b100;
        else;
    end
endmodule
```

- **功能描述**

DataCompare 模块使用了行为型描述，实现了一个 4 位二进制数据比较器，通过 if-else 语句比较两组输入数据（iData_a 和 iData_b）的大小关系，并将比较结果输出到 3 位控制信号（oData）中。

XDC 约束

变量	iData_a [0]~[3]	iData_b [0]~[3]	iData [0]~[2]	oData [0]~[2]
N4 板上的管脚	SW0~3 (J15、L16、M13、R15)	SW4~7 (R17、T18、U18、R13)	SW13~15 (U12、U11、V10)	LD0~2 (H17、K15、J13)

（二）8 位数据比较器 DataCompare8

接口定义

```
module DataCompare8(
 input [7:0] iData_a, //输入数据 a
 input [7:0] iData_b, //输入数据 b
output [2:0] oData //结果输出
);
```

Verilog 代码描述

```
`timescale 1ns / 1ps
```

```
module DataCompare8(
    input [7:0] iData_a,
    input [7:0] iData_b,
    output [2:0] oData
);
    wire [2:0] tmp;
    DataCompare4 low(.iData_a(iData_a[3:0]), .iData_b(iData_b[3:0]),
 .iData(3'b001), .oData(tmp));
    DataCompare4 high(.iData_a(iData_a[7:4]), .iData_b(iData_b[7:4]),
 .iData(tmp), .oData(oData));
endmodule
```

功能描述

DataCompare8 模块使用了组合逻辑的数据流型描述。实现了一个 8 位二进制数据比较器。

模块内部调用了上文名为 DataCompare4 的子模块，分别用于对 8 位数据的高 4 位和低 4 位进行比较。首先，将 iData_a 和 iData_b 分别拆分成高 4 位和低 4 位数据，先将低 4 位传递给名为 low 的 DataCompare 模块，并将比较结果传递给 tmp，随后将 tmp 和高 4 位结果传递给名为 high 的 DataCompare 模块，并输出最终的比较结果。

但是，DataCompare 模块稍有修改，将 iData_a==iData_b 时 iData 赋值给 oData，用于级联。通过 iData_a[7:4]切片操作，可以把 8 位中的高位和低位提取出来。

XDC 约束

变量	iData_a[0]~[7]	iData_b[0]~[7]	oData[0]~[2]
N4 板上的管脚	SW0~7 (J15、L16、M13、R15、R17、T18、U18、R13)	SW8~15 (T8、U8、R16、T13、H6、U12、U11、V10)	LD0~2 (H17、K15、J13)

（三）一位加法器 FA

接口定义

```
module FA(
  input iA, //1 位二进制加数
  input iB, //1 位二进制被加数
  input iC, //低位的进位信号
  output oS, //1 位和数
  output oC //向高位的进位信号
);
```

Verilog 代码描述

```
module FA(
    input iA,
    input iB,
    input iC,
    output oS,
    output oC
    );
    wire s1, s2, s3;
    wire tmp1, tmp2;
    xor u1 (tmp1, iA, iB);
    xor u2 (oS, tmp1, iC);
    and u3 (s1, iA, iB);
    and u4 (s2, iB, iC);
```

```
    and u5 (s3, iA, iC);
    or u6 (tmp2, s1, s2);
    or u7 (oC, tmp2, s3);
endmodule
```

功能描述

FA 模块使用了结构型描述，实现了一个全加器。三个输入信号，iA，iB 代表进行加法的数据，iC 用于传输是否有进位要考虑。输出有两个，一个是和 oS，但是仅有一位，另一个是进位输出 oC，用于记录是否有进位。

oS = (iA ^ iB) ^ iC 通过异或实现了加法运算，先处理两个数，再处理进位。s1,s2,s3 分别考虑了各种组合情况，通过这种穷举的办法实现了进位的判断。

XDC 约束

变量	iA	iB	iC	oS	oC
N4 板上的管脚	SW0 (J15)	SW1 (L16)	SW15 (V10)	LD0 (H17)	LD1 (K15)

（四）8 位加法器

接口定义

```
module Adder(
    input [7:0] iData_a, //8  位二进制加数
    input [7:0] iData_b, //8  位二进制被加数
    input iC, //低位的进位信号
    output [7:0] oData, //8  位和数
    output oData_C //向高位的进位信号
);
```

Verilog 代码描述

```
`timescale 1ns / 1ps
module Adder(
    input [7:0] iData_a,
    input [7:0] iData_b,
    input iC,
    output [7:0] oData,
    output oData_C
);
    wire oC1;
    wire oC2;
    wire oC3;
    wire oC4;
    wire oC5;
    wire oC7;
    wire oC6;
    FA A1(.iA(iData_a[0]),.iB(iData_b[0]),.iC(iC),.oS(oData[0]),.oC(oC1));
    FA A2(.iA(iData_a[1]),.iB(iData_b[1]),.iC(oC1),.oS(oData[1]),.oC(oC2));
    FA A3(.iA(iData_a[2]),.iB(iData_b[2]),.iC(oC2),.oS(oData[2]),.oC(oC3));
    FA A4(.iA(iData_a[3]),.iB(iData_b[3]),.iC(oC3),.oS(oData[3]),.oC(oC4));
    FA A5(.iA(iData_a[4]),.iB(iData_b[4]),.iC(oC4),.oS(oData[4]),.oC(oC5));
    FA A6(.iA(iData_a[5]),.iB(iData_b[5]),.iC(oC5),.oS(oData[5]),.oC(oC6));
    FA A7(.iA(iData_a[6]),.iB(iData_b[6]),.iC(oC6),.oS(oData[6]),.oC(oC7));
    FA A8(.iA(iData_a[7]),.iB(iData_b[7]),
          .iC(oC7),.oS(oData[7]),.oC(oData_C));
endmodule
```

功能描述

Adder 模块使用了数据流型描述，通过调用上文的 FA 一位全加器模块，实现了一个 8 位加法器。

模块内部使用了 8 个全加器（FA1 到 FA8）来执行加法操作。每个全加器接收两个输入位（iA 和 iB）和一个进位信号（iC），并输出一个和（oS）和一个进位信号（oC）。每个全加器的输出 oC 用作下一个全加器的进位输入，如此递推形成了级联的加法操作。

通过这个模块，我们可以继续实现 16 位，32 位甚至 64 位的加法运算。同时，本模块仅考虑了无符号位，稍加修改可以改成有符号位的加法器。

XDC 约束

变量	iData_a [0]~[7]	iData_b [0]~[7]	iC	oData [0]~[7]	oData_C
N4 板上的管脚	SW0~7 (J15、L16、M13、R15、R17、T18、U18、R13)	SW8~15 (T8、U8、R16、T13、H6、U12、U11、V10)	BNTR (M17)	LD0~7 (H17、K15、J13、N14、R18、V17、U17、U16)	LD8 (V16)

四、测试模块建模

(一)4 位数据比较器 DataCompare4

```
`timescale 1ns / 1ps
module DataCompare4_tb;
    reg [3:0] iData_a;
    reg [3:0] iData_b;
    reg [2:0] iData;
    wire [2:0] oData;
    DataCompare4 uut(
        .iData_a(iData_a),
        .iData_b(iData_b),
        .iData(iData),
        .oData(oData)
    );
    initial begin
    iData_a = 4'b0000;
    iData_b = 4'b0000;
    iData = 3'b000;
    #10 iData_a = 4'b0010;
    iData_b=4'b0000;
    iData = 3'b100;
```

```
#10 iData_a = 4'b0011;
    iData_b=4'b0011;
    iData = 3'b010;
    #10 iData_a = 4'b0101;
       iData_b = 4'b1100;
       iData = 3'b100;
       #10 iData_a = 4'b1010;
       iData_b = 4'b0010;
       iData = 3'b010;
       #10 iData_a = 4'b1111;
       iData_b = 4'b1111;
       iData = 3'b001;
       #10 iData_a = 4'b1000;
       iData_b = 4'b1000;
       iData = 3'b001;
        #10 iData_a = 4'b0010;
       iData_b = 4'b0010;
       iData = 3'b100;
       #10 iData_a = 4'b0010;
       iData_b = 4'b0010;
       iData = 3'b010;
    end
endmodule
```

(二)8 位数据比较器 DataCompare8

```
 `timescale 1ns / 1ps
module DataCompare8_tb();
    reg [7:0] iData_a;
    reg [7:0] iData_b;
    wire [2:0] oData;
    DataCompare8
uul(.iData_a(iData_a), .iData_b(iData
_b), .oData(oData));
    initial
    begin
         iData_a = 8'b0000_0000;
        iData_b = 8'b0000_0000;
        # 10 iData_a = 8'b0000_1000;
        iData_b = 8'b0000_1010;
        # 10 iData_a = 8'b1010_1001;
        iData_b = 8'b0101_0110;
        # 10 iData_a = 8'b0000_0000;
        iData_b = 8'b0000_0010;
        # 10 iData_a = 8'b1010_0101;
        iData_b = 8'b1010_0101;
        # 10 iData_a = 8'b0111_1111;
        iData_b = 8'b1000_1111;
        # 10 iData_a = 8'b1111_1111;
        iData_b = 8'b1111_0111;
        # 10 iData_a = 8'b0101_0110;
        iData_b = 8'b1110_1001;
    end
endmodule
```

(三)1 位全加器 FA

```
`timescale 1ns / 1ps
module FA_tb();
    reg iA,iB,iC;
    wire oS, oC;
    FA
uul(.iA(iA), .iB(iB), .iC(iC), .oS(oS),
.oC(oC));
    initial
    begin
       iA = 0;
       iB = 0;
       iC = 0;

       # 40 iA = 0;
       iB = 0;
       iC = 1;

       # 40 iA = 0;
       iB = 1;
       iC = 0;

 # 40 iA = 0;
       iB = 1;
       iC = 1;

       # 40 iA = 1;
       iB = 0;
       iC = 0;

       # 40 iA = 1;
       iB = 0;
       iC = 1;

       # 40 iA = 1;
       iB = 1;
       iC = 0;

         # 40 iA = 1;
       iB = 1;
       iC = 1;
    end
endmodule
```

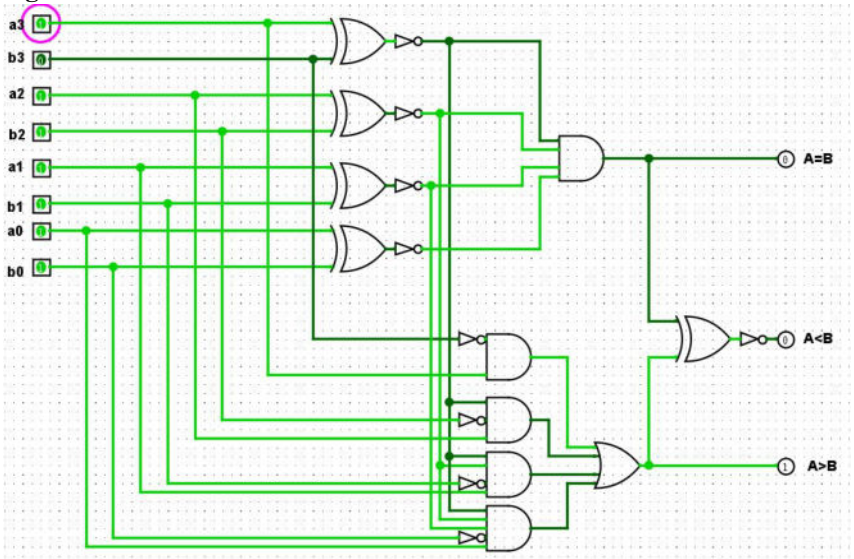
(四)8 位加法器 Adder

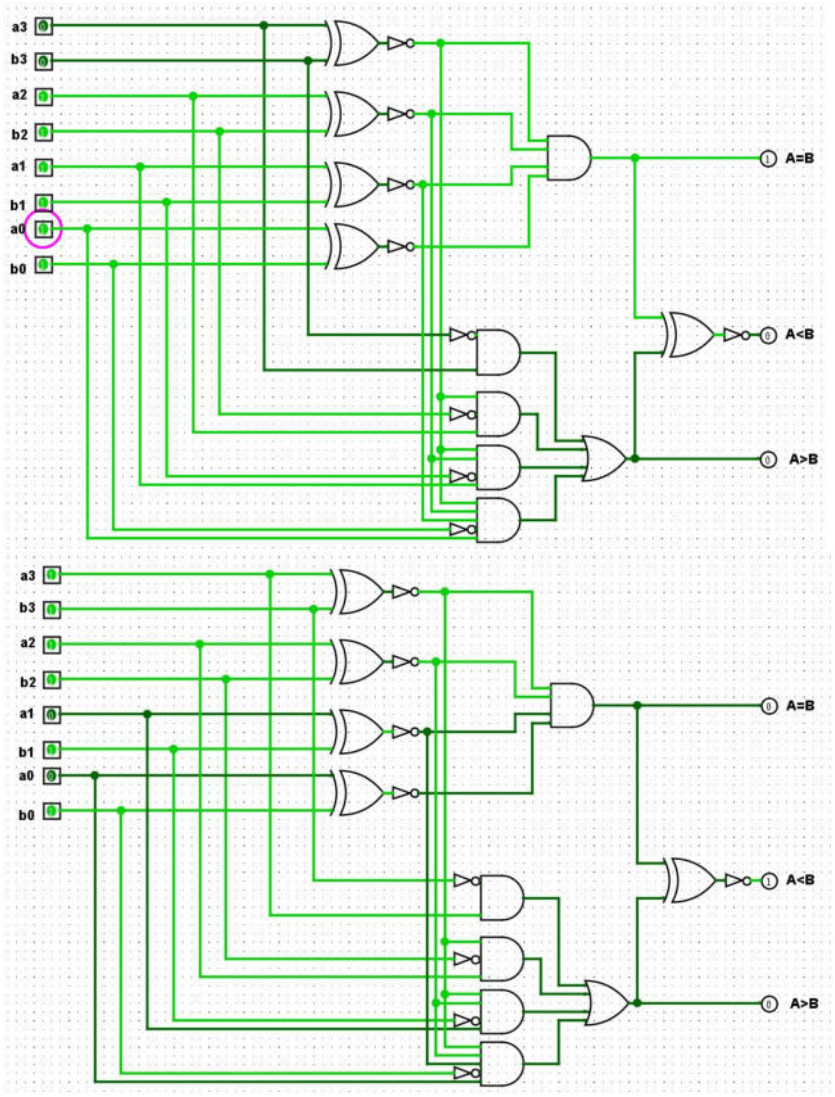
```
`timescale 1ns / 1ps
module Adder_tb();
    reg [7:0] iData_a;
    reg [7:0] iData_b;
    reg iC;
    wire [7:0] oData;
    wire oData_C;
    Adder
uul(.iData_a(iData_a), .iData_b(iData_b),
 .iC(iC), .oData(oData), .oData_C(oData_C
));
    initial
    begin
        iData_a = 8'b0000_1101;
        iData_b = 8'b0000_0010;
        iC = 0;
        # 40
        iData_a = 8'b0000_1101;
        iData_b = 8'b0000_1111;
        iC = 0;
        # 40
        iData_a = 8'b0000_1011;
        iData_b = 8'b0000_1011;
        iC = 1;
        # 40
        iData_a = 8'b0101_0101;
        iData_b = 8'b1010_0110;
        iC = 1;
        # 40
        iData_a = 8'b1010_1101;
        iData_b = 8'b1011_0010;
        iC = 0;
        # 40
        iData_a = 8'b0100_0101;
        iData_b = 8'b1110_0010;
        iC = 1;

        # 40
        iData_a = 8'b1101_0101;
        iData_b = 8'b1010_0010;
        iC = 0;
    end
endmodule
```

五、实验结果

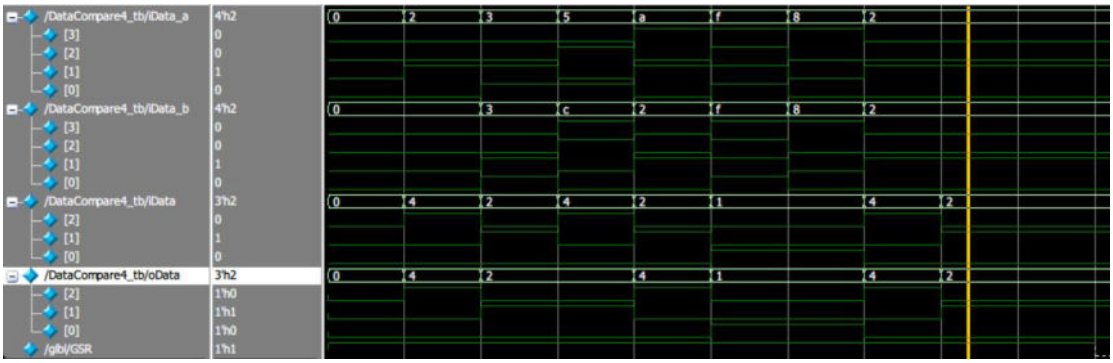
(一)4 位数据比较器

- **logisim 验证图**

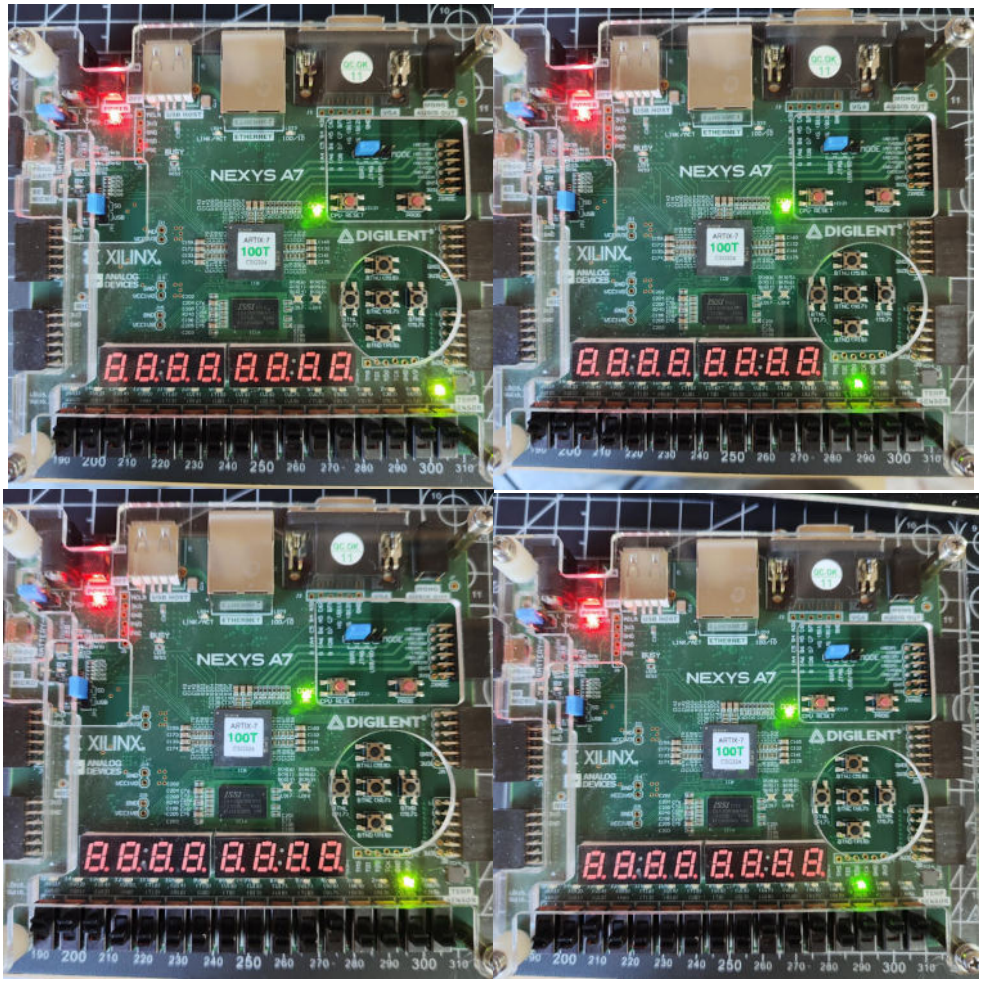




- **modelsim 仿真**

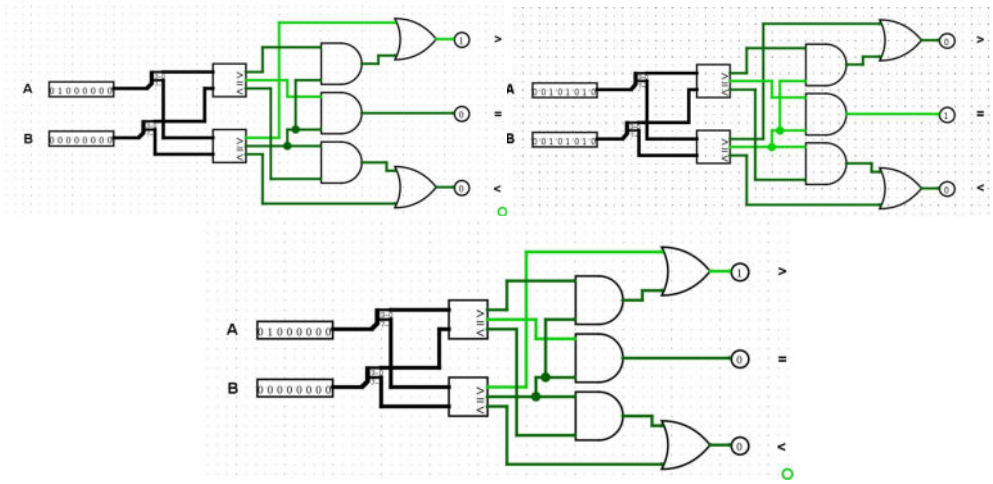


- **实验结果**

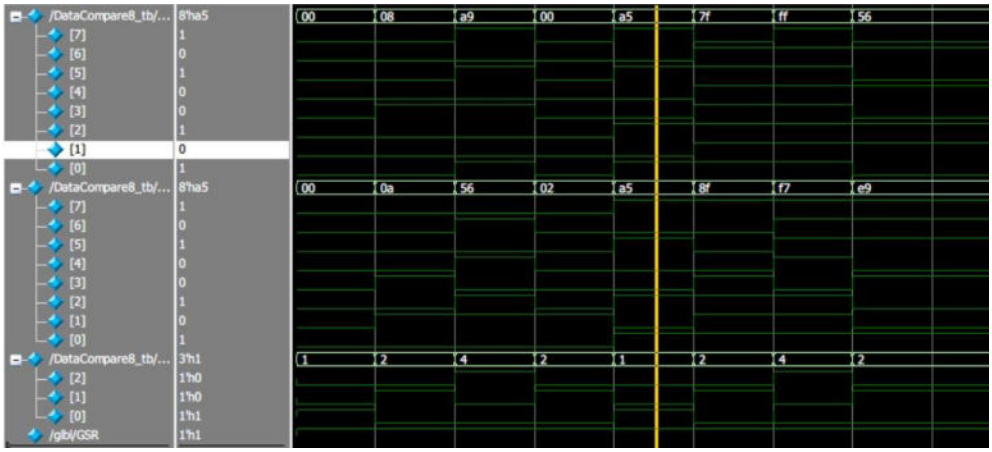


(二)8 位数据比较器

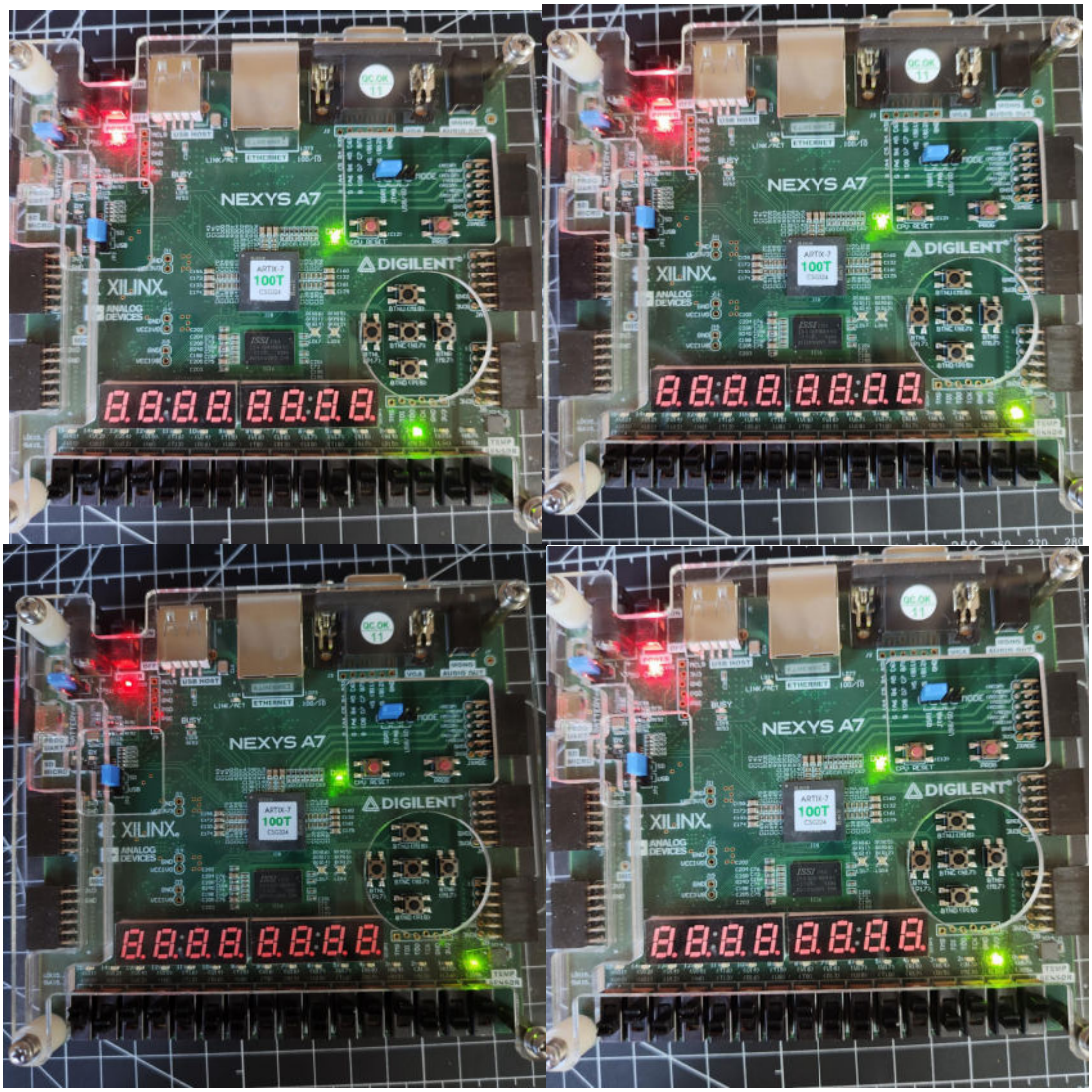
- **logisim 原理图**



- **modelsim 仿真**

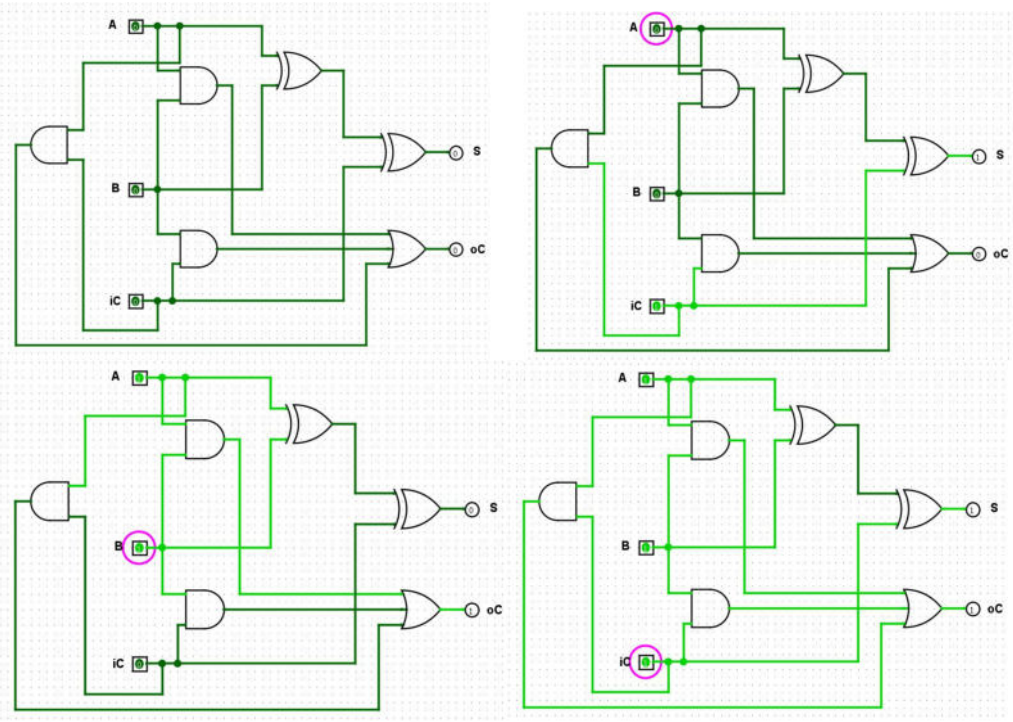


- **实验结果**

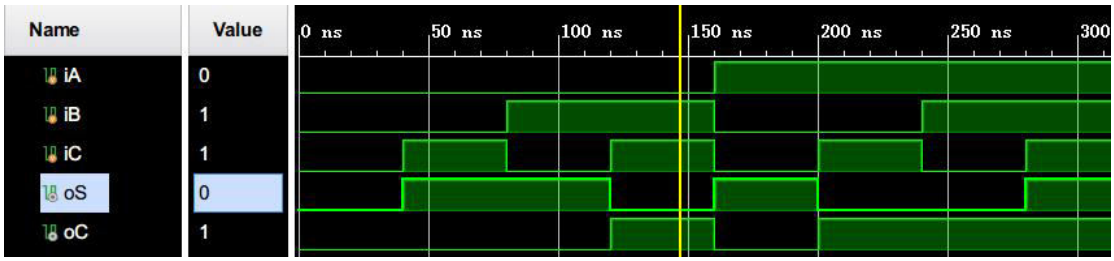


(三)1 位全加器

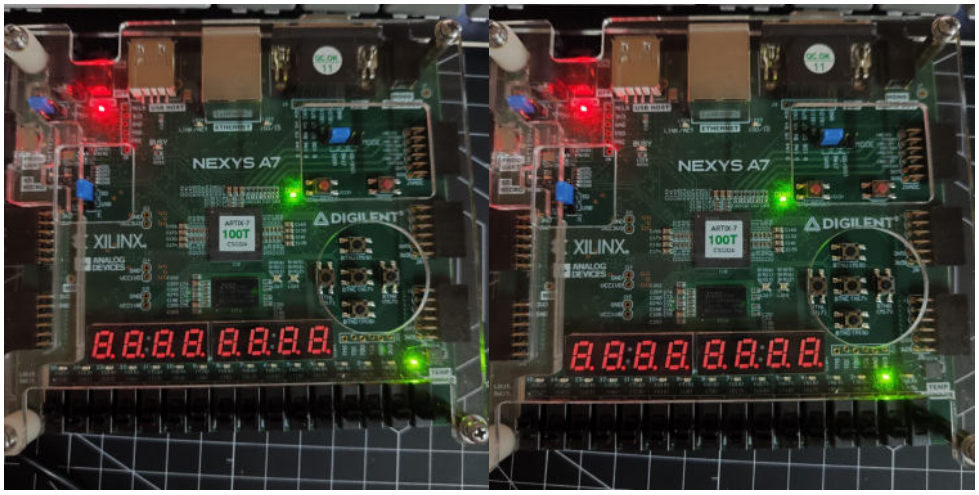
- **logisim 原理图**

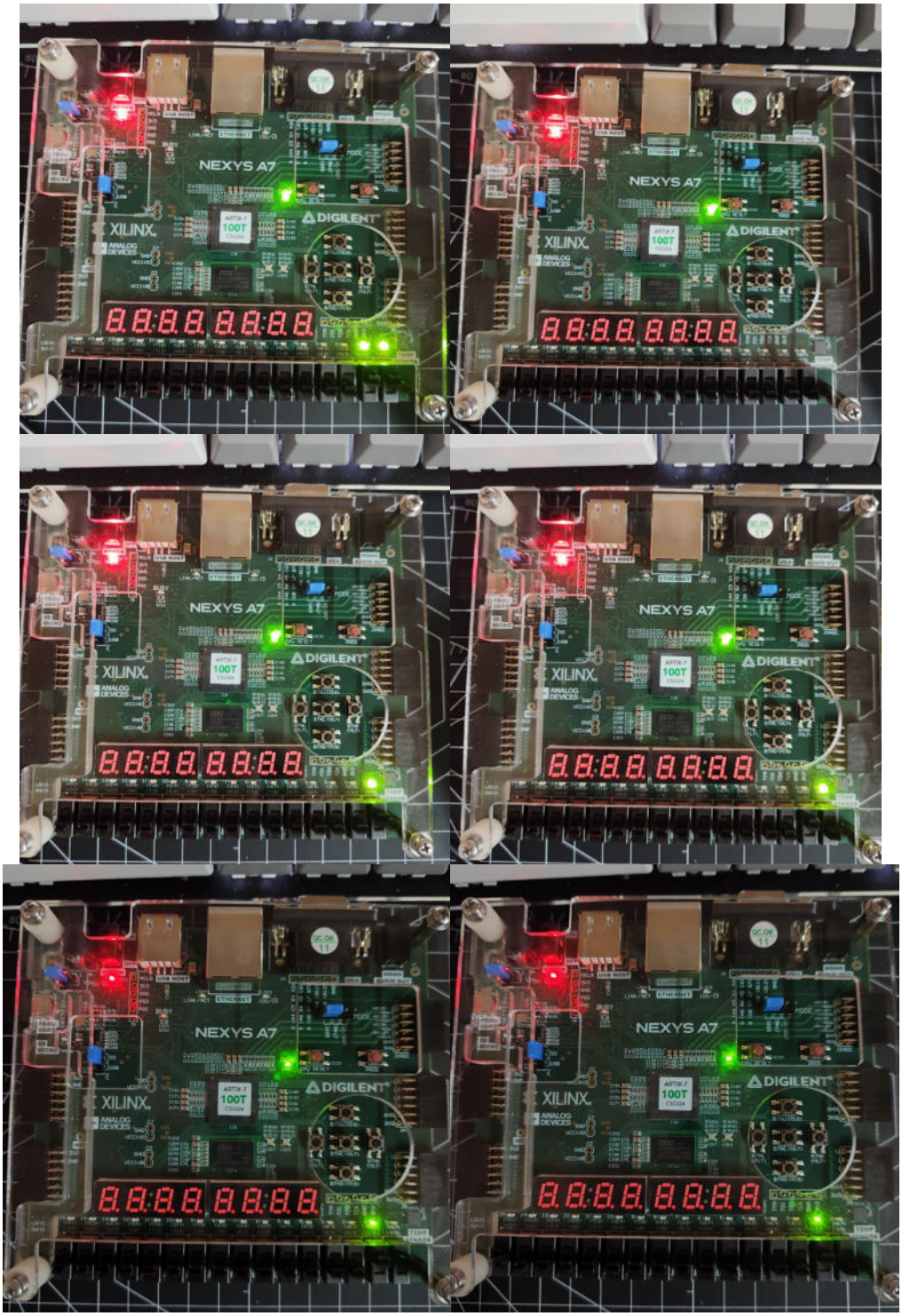


- **modelsim 仿真波形图**



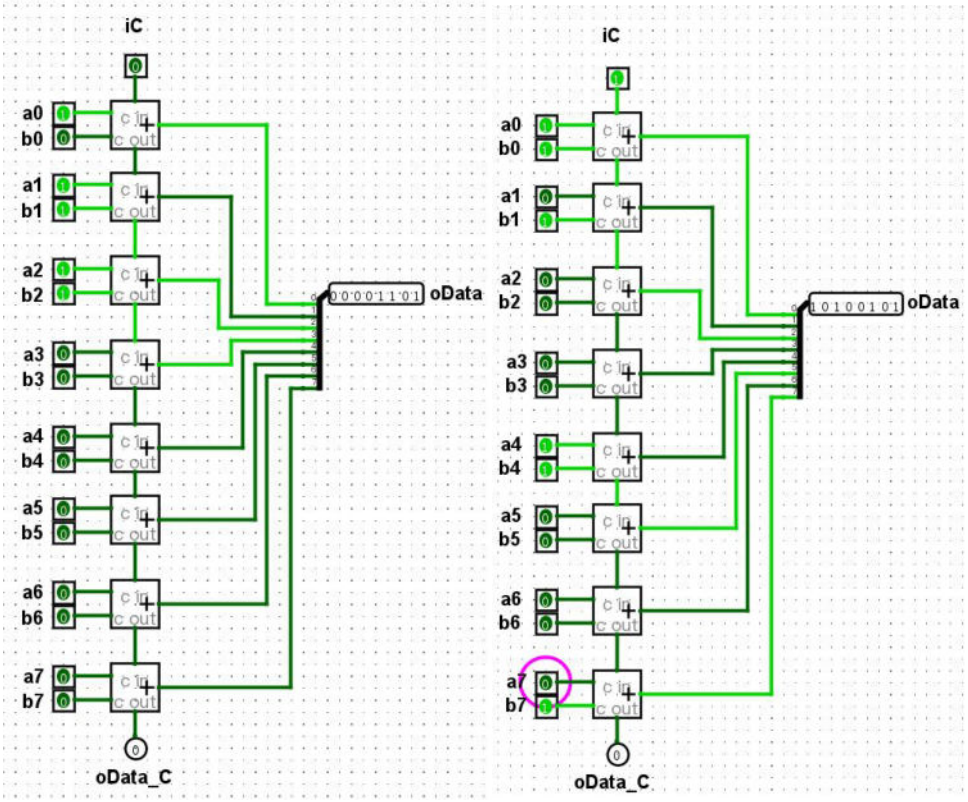
- **实验结果**



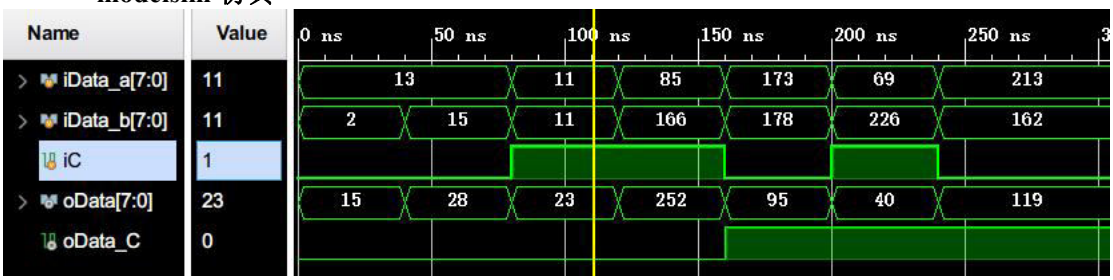


(四)8 位加法器

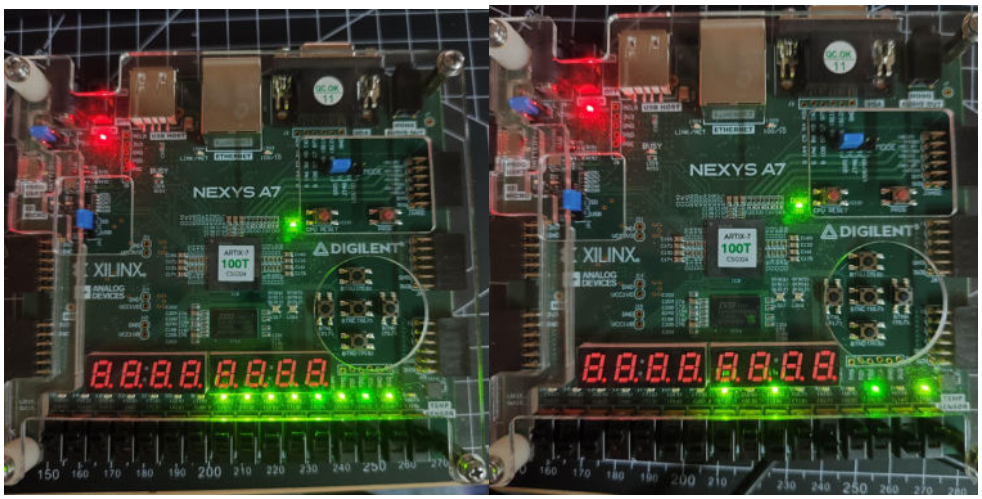
- **logisim 原理图**

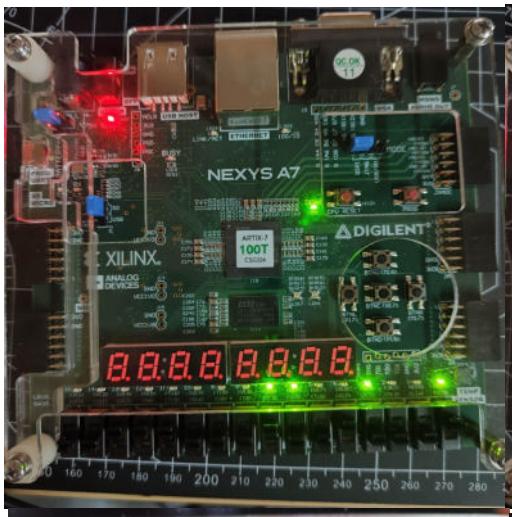


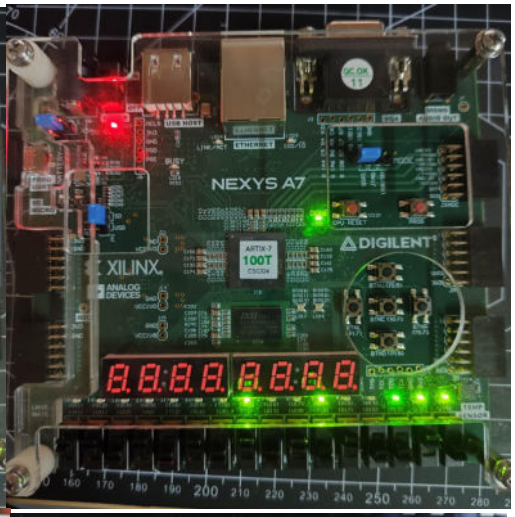
- **modelsim 仿真**

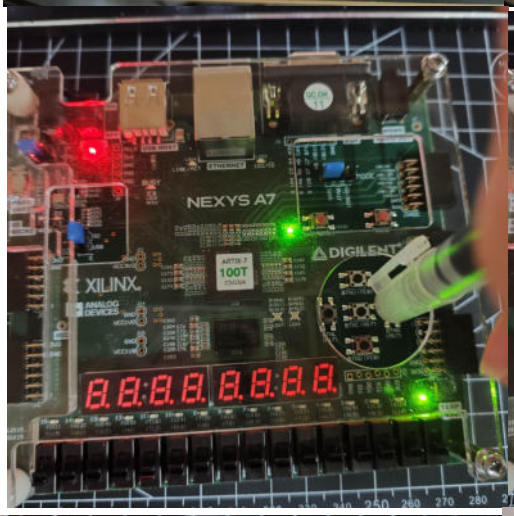


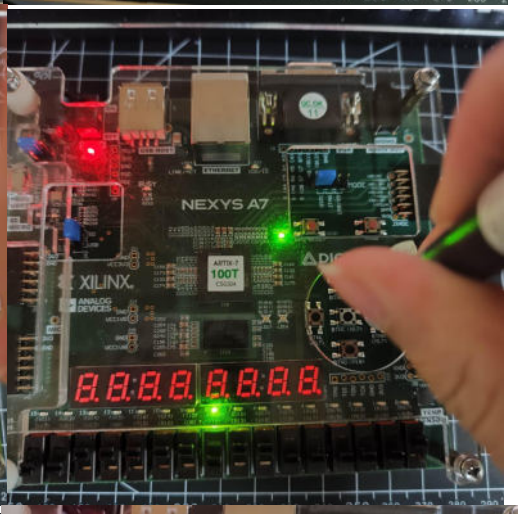
- **实验结果**

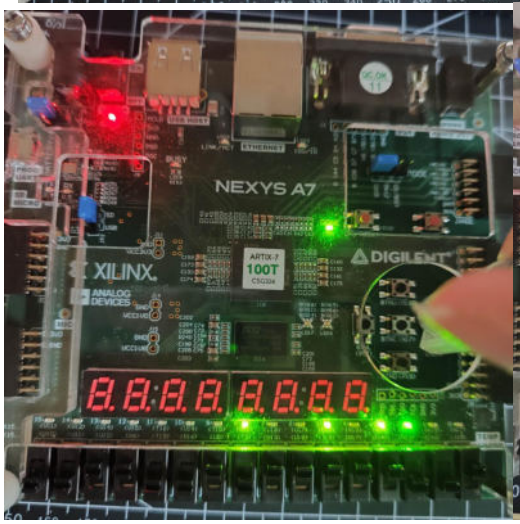


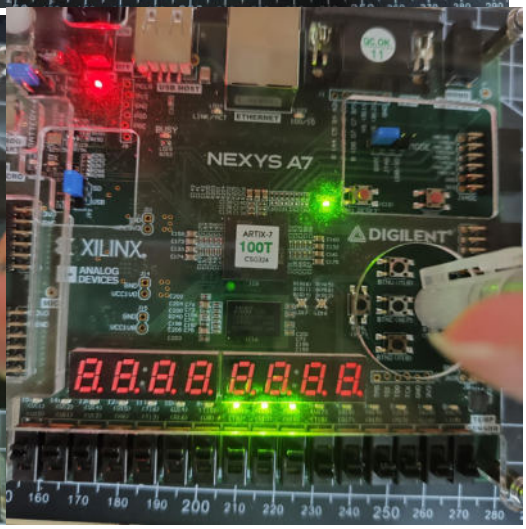

NEXYS A7
XILINX
DIGILENT


NEXYS A7
XILINX
DIGILENT


NEXYS A7
XILINX
DIGILENT


NEXYS A7
XILINX


NEXYS A7
XILINX
DIGILENT


NEXYS A7
XILINX
DIGILENT