实验报告 1

1 实验目的

1.1 熟悉 verilog 编程与调试

通过编写不同层次的激励文件, 对于多层次结构的程序进行调试

1.2 熟悉简单比较器的工作原理

1.3 通过简单模块实例化、连下实现复杂的数字电路

2 实验环境

AMD Vivado2022.2

3 原理说明

3.1 4 位比较器

对于比较器, 采用从高位向地位逐个比较的办法, 如果高位已经比较出结果, 那么低位就不用比较, 逻辑表达式如下

$$\begin{aligned}
out_A_G_B = {} & A_3B_3' + (A_3 \odot B_3)A_2B_2' + (A_3 \odot B_3)(A_2 \odot B_2)A_1B_1' \\
& + (A_3 \odot B_3)(A_2 \odot B_2)(A_1 \odot B_1')A_0B_0' \\
& + (A_3 \odot B_3)(A_2 \odot B_2)(A_1 \odot B_1')(A_0 \odot B_0)in_A_G_B \\
out_A_E_B = {} & (A_3 \odot B_3)(A_2 \odot B_2)(A_1 \odot B_1')(A_0 \odot B_0)in_A_E_B \\
out_A_L_B = {} & (out_A_G_B + out_A_E_B)'
\end{aligned}$$

3.2 16 位数值比较器

通过将四个 4 位比较器串行连接起来, 前一个比较器的结果输出是下一个比较器的结果输入, 以此来实现多位比较

3.3 4 位超前进位加法器

直接写出每一个进位的表达式, 进而可以直接输出每一位的结果, 而不用等上一位才能进行下一位, 尽管电路会更为复杂, 但是用时会更短, 在超前进位加法器中, 引入了两个新的逻辑

变量来简化逻辑表达, 也为实现多位的超前进位加法器提供便利, 逻辑表达式如下

$$
\begin{aligned}
P_i &= A_i \oplus B_i \\
G_i &= A_i B_i \\
CO_i &= G_i + P_i(CI_i) \\
&= G_i + P_i(G_{i-1} + P_{i-1}(CI_{i-1})) \\
&= \cdots
\end{aligned}
$$

3.4 32 位超前进位加法器

利用 11 个 4 位超前加法器搭建一个 32 位超前加法器, 形成一个树状结构, 自下而上进行输入之后, 在自上而下计算进位信号等, 最终输出结果

3.5 门电路图

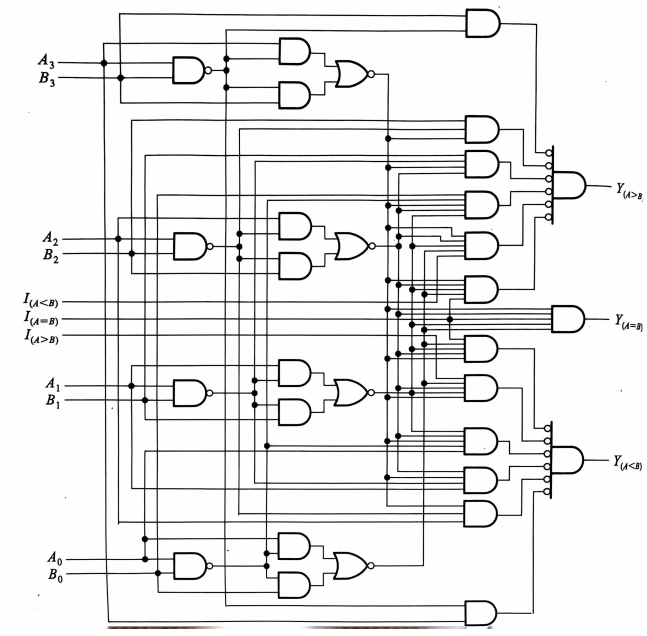

图 1: 4 位数值比较器

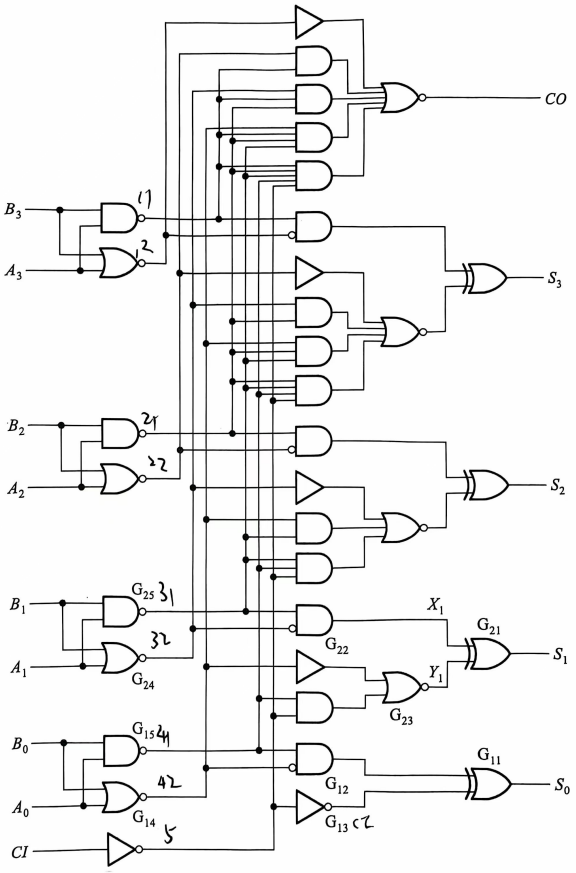

图 2: 4 位超前进位加法器

4 接口定义

4.1 4 位数值比较器

```
module comp_4(
    input [3:0]A,           (4位输入1)
    input [3:0]B,           (4位输入2)
    input in_A_G_B,         (前比较结果A>B输入)
    input in_A_E_B,         (前比较结果A=B输入)
    input in_A_L_B,         (前比较结果A<B输入)
    output reg out_A_G_B,   (比较结果A>B输出)
    output reg out_A_E_B,   (比较结果A=B输出)
    output reg out_A_L_B    (比较结果A<B输出)
);
```

4.2 16 位数值比较器

与 4 位数值比较器的接口定义基本相同, 只是带宽不同, 在此不多赘述

4.3 4 位超前加法器

```
module add(
    input [3:0]A,           (4位输入1)
    input [3:0]B,           (4位输入2)
    input Cin,              (进位输入)
    output [3:0]S,          (本位结果输出)
    output [3:0]C,          *(计算中进位输出)
    output Cout,            (进位输出)
    output p,               *(向上层输出p)
    output g                *(向上层输出g)
);
```

值得注意的是, 在这个四位超前加法器中, 有许多接口是为了实现更高位的超前进位加法器定义的, 比如 C 是用于向下输出进位信号的, p 和 g 是向上输出以产生进位信号的, 单纯的四位超前进位加法器并不需要这些端口

5 调试过程以及结果

5.1 4 位和 16 位数值比较器

在编写 4 位数值比较器时, 唯一遇到的困难是同或的运算如何表示, 经过查询有如下几种表示 $S = A \odot B$

```
1        xnor(S,A,B)
2        assign S=(A ^ B)'
3        assign S=(A ~^ B)
```

波形图如下, check=1 表示结果正确

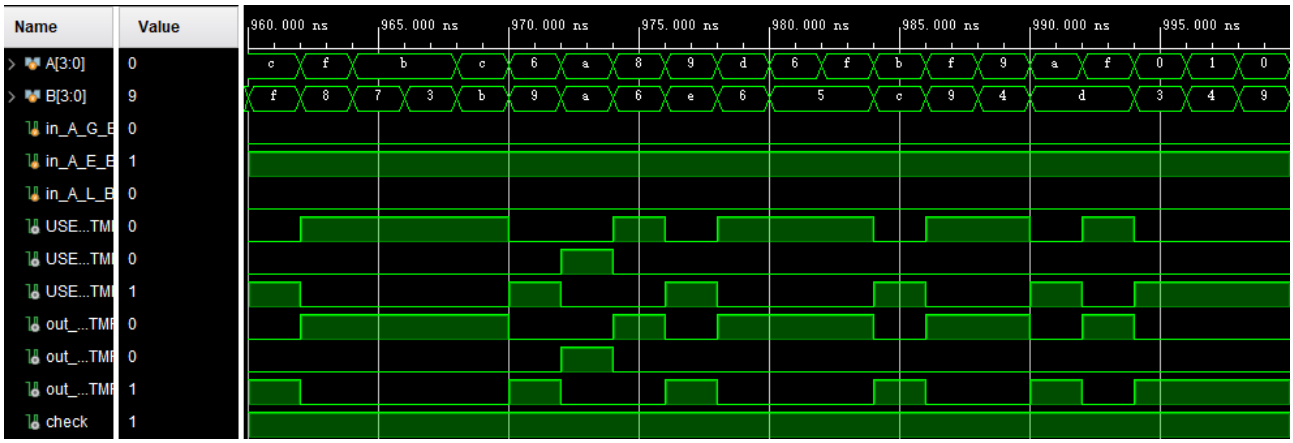

图 3: 4 位数值比较器波形图

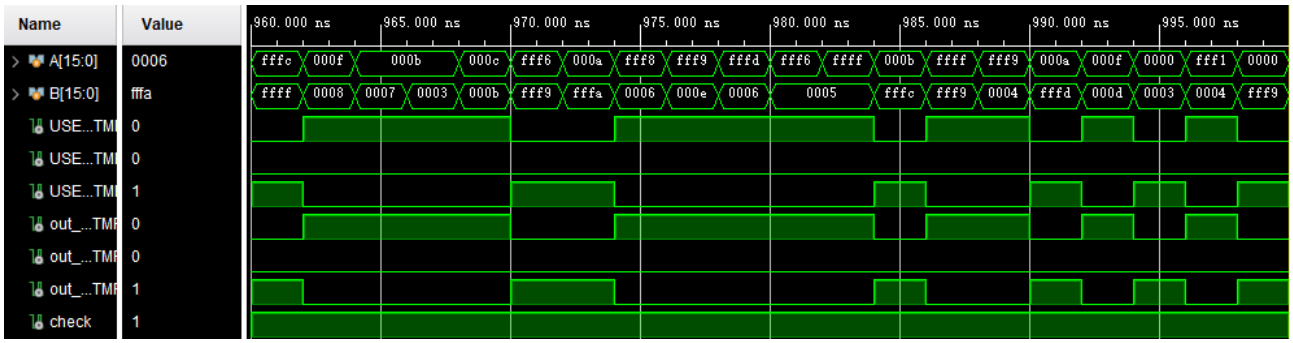

图 4: 16 位数值比较器波形图

5.2 32 位超前进位加法器

利用 4-16-32 的结构层层搭建, 形成一个树状结构, 值得注意的是在讲义中最底层向上输出的 P 和 G 并不是单纯的 P 和 G, 而实际上是如下表达式

$$P = P_3P_2P_1P_0$$

$$G = G_3 + P_3G_2 + P_3P_2G_1 + P_3P_2P_1G_0$$

采用的主要原理是自下而上输入 P、G, 计算出所有进位信号后转而向下输出, 最终计算出结果

本实验中对三个层次都进行了测试, 波形图如下

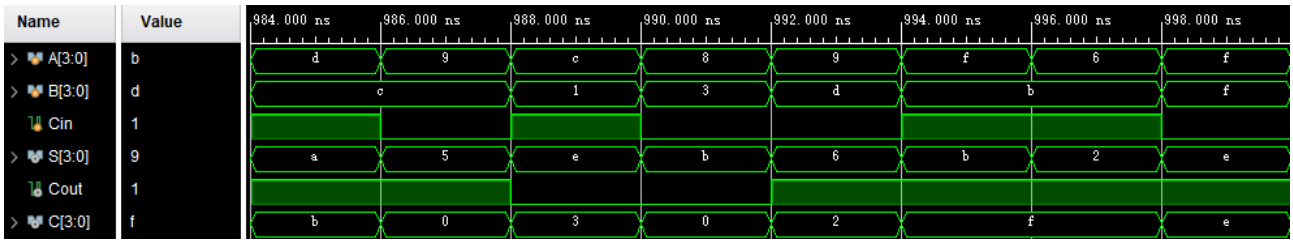

图 5: 4 位超前进位加法器波形图

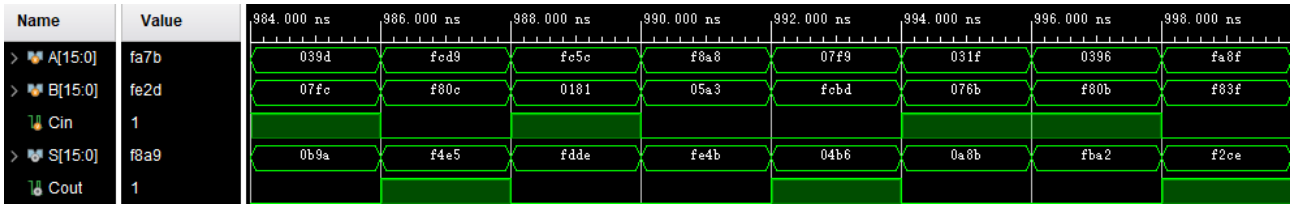

图 6: 16 位超前进位加法器波形图

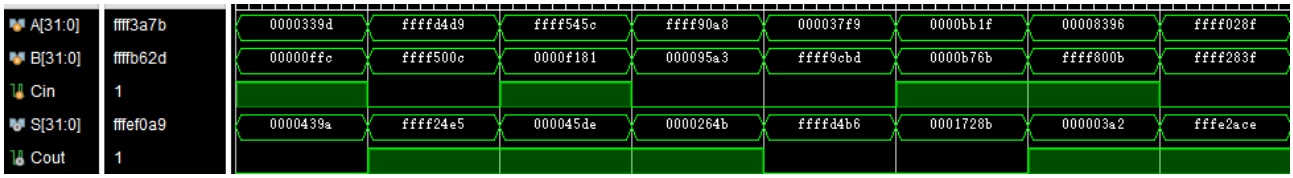

图 7: 32 位超前进位加法器波形图

6 实验总结

6.1 实例化模块

本次实验主要的收获是学会如何实例化模块, 并利用已经写出的模块构建功能更广泛的器件, 另外, 在实例化模块且不需要模块的某些输出时可以不写或者写空括号. 在器件构建中, 最好自上而下、自下而上两个方向分别考虑, 考虑出结构和接口后进行编写

6.2 verilog 调试

在进行这种多层次结构的程序的调试时, 可以从底层到高层撰写多个 testbench 来分别进行模拟, 可以更好的进行调试与 debug

7 源代码

7.1 4 位数值比较器

```
module comp_4(
    input [3:0]A,
    input [3:0]B,
    input in_A_G_B,
    input in_A_E_B,
    input in_A_L_B,
    output reg out_A_G_B,
    output reg out_A_E_B,
    output reg out_A_L_B
    );
    wire [3:0]T;
    xnor(T[0],A[0],B[0]);
    xnor(T[1],A[1],B[1]);
    xnor(T[2],A[2],B[2]);
    xnor(T[3],A[3],B[3]);

    always @(*)begin
    out_A_G_B = (A[3] & ~B[3]) |
                (T[3] & A[2] & ~B[2] ) |
                (T[3] & T[2] & A[1] & ~B[1]) |
                (T[3] & T[2] & T[1] & A[0] & ~B[0]) |
                (T[3] & T[2] & T[1] & T[0] & in_A_G_B);
    out_A_E_B = T[3] & T[2] & T[1] & T[0] & in_A_E_B;
    out_A_L_B = ~(out_A_G_B | out_A_E_B);
    end

endmodule
```

7.2 16 位数值比较器

```
module comp_16(
    input [15:0]A,
    input [15:0]B,
    input in_A_G_B,
```

```
5         output reg out_A_G_B,
6         output reg out_A_E_B,
7         output reg out_A_L_B
8         );
9         wire [3:0]G;
10        wire [3:0]E;
11        wire [3:0]L;
12
13        comp_4 comp0(
14            .A(A[3:0]),
15            .B(B[3:0]),
16            .in_A_G_B(0),
17            .in_A_E_B(1),
18            .in_A_L_B(0),
19            .out_A_G_B(G[0]),
20            .out_A_E_B(E[0]),
21            .out_A_L_B(L[0])
22        );
23
24        comp_4 comp1(
25            .A(A[7:4]),
26            .B(B[7:4]),
27            .in_A_G_B(G[0]),
28            .in_A_E_B(E[0]),
29            .in_A_L_B(L[0]),
30            .out_A_G_B(G[1]),
31            .out_A_E_B(E[1]),
32            .out_A_L_B(L[1])
33        );
34
35        comp_4 comp2(
36            .A(A[11:8]),
37            .B(B[11:8]),
38            .in_A_G_B(G[1]),
39            .in_A_E_B(E[1]),
40            .in_A_L_B(L[1]),
41            .out_A_G_B(G[2]),
```

```
            .out_A_E_B(E[2]),
            .out_A_L_B(L[2])
        );

        comp_4 comp3(
            .A(A[15:12]),
            .B(B[15:12]),
            .in_A_G_B(G[2]),
            .in_A_E_B(E[2]),
            .in_A_L_B(L[2]),
            .out_A_G_B(G[3]),
            .out_A_E_B(E[3]),
            .out_A_L_B(L[3])
        );

        always @(*)begin
            out_A_G_B=G[3];
            out_A_E_B=E[3];
            out_A_L_B=L[3];
        end

    endmodule
```

7.3 4 位超前进位加法器

```
    module add(
        input [3:0]A,
        input [3:0]B,
        input Cin,
        output [3:0]S,
        output [3:0]C,
        output Cout,
        output p,
        output g
        );
        wire [3:0]P;
        wire [3:0]G;
```

```
13
14          assign P[0] = A[0] ^ B[0];
15          assign P[1] = A[1] ^ B[1];
16          assign P[2] = A[2] ^ B[2];
17          assign P[3] = A[3] ^ B[3];
18
19          assign G[0] = A[0] & B[0];
20          assign G[1] = A[1] & B[1];
21          assign G[2] = A[2] & B[2];
22          assign G[3] = A[3] & B[3];
23
24
25          assign C[0] = Cin;
26          assign C[1] = G[0] | P[0]&Cin;
27          assign C[2] = G[1] | (P[1] & (G[0] | (P[0] & Cin)));
28          assign C[3] = G[2] | (P[2] & (G[1] | (P[1] & (G[0] |
29                        (P[0] & Cin)))));
30          assign Cout = G[3] | (P[3] & (G[2] |
31                        (P[2] & (G[1] | (P[1] & (G[0] |
32                        (P[0] & Cin)))))));
33
34
35          assign S[0] = P[0] ^ Cin;
36          assign S[1] = P[1] ^ C[1];
37          assign S[2] = P[2] ^ C[2];
38          assign S[3] = P[3] ^ C[3];
39
40          assign p=P[3]&P[2]&P[1]&P[0];
41          assign g=G[3] | P[3]&G[2] | P[3]&P[2]&G[1] |
42                    P[3]&P[2]&P[1]&G[0];
43
44      endmodule
```

7.4 16 位超前进位加法器

```
1       module add_16(
2           input [15:0]A,
```

```
3         input [15:0]B,
4         input Cin,
5         output Cout,
6         output [15:0]S,
7         output p,
8         output g
9         );
10        wire[3:0]P;
11        wire[3:0]G;
12        wire[3:0]C;
13        wire a;
14        wire b;
15
16        assign C[0]=Cin;
17
18        add pre1(
19            .A(A[3:0]),
20            .B(B[3:0]),
21            .p(P[0]),
22            .g(G[0])
23        );
24
25        add pre2(
26            .A(A[7:4]),
27            .B(B[7:4]),
28            .p(P[1]),
29            .g(G[1])
30        );
31
32        add pre3(
33            .A(A[11:8]),
34            .B(B[11:8]),
35            .p(P[2]),
36            .g(G[2])
37        );
38
39        add pre4(
```

```
            .A(A[15:12]),
            .B(B[15:12]),
            .p(P[3]),
            .g(G[3])
        );

        add4_inputPG outC(
            .P(P[3:0]),
            .G(G[3:0]),
            .Cin(Cin),
            .C(C[3:0]),
            .Cout()
        );

        add add1(
            .A(A[3:0]),
            .B(B[3:0]),
            .Cin(Cin),
            .S(S[3:0]),
            .C(),
            .Cout(),
            .p(),
            .g()
        );

        add add2(
            .A(A[7:4]),
            .B(B[7:4]),
            .Cin(C[1]),
            .C(),
            .S(S[7:4]),
            .Cout(),
            .p(),
            .g()
        );

        add add3(
```

```
            .A(A[11:8]),
            .B(B[11:8]),
            .Cin(C[2]),
            .C(),
            .S(S[11:8]),
            .Cout(),
            .p(),
            .g()
        );

        add add4(
            .A(A[15:12]),
            .B(B[15:12]),
            .Cin(C[3]),
            .C(),
            .S(S[15:12]),
            .Cout(Cout),
            .p(),
            .g()
        );

        assign p=P[3]&P[2]&P[1]&P[0];
        assign g=G[3] | P[3]&G[2] | P[3]&P[2]&G[1] |
                  P[3]&P[2]&P[1]&G[0];

    endmodule
```

7.5 32 位超前进位加法器

```
    module add32(
        input [31:0]A,
        input [31:0]B,
        input Cin,
        output [31:0]S,
        output Cout
        );

```

```
9         wire [1:0]C;wire p0;wire g0;
10
11        assign C[0]=Cin;
12
13        add_16 pre(
14        .A(A[15:0]),
15        .B(B[15:0]),
16        .Cin(C[0]),
17        .S(),
18        .p(p0),
19        .g(g0)
20        );
21
22        assign C[1]=g0 | (p0 & Cin);
23
24        add_16 add1(
25        .A(A[15:0]),
26        .B(B[15:0]),
27        .Cin(C[0]),
28        .S(S[15:0]),
29        .Cout(),
30        .p(),
31        .g()
32        );
33
34        add_16 add2(
35        .A(A[31:16]),
36        .B(B[31:16]),
37        .Cin(C[1]),
38        .S(S[31:16]),
39        .Cout(Cout),
40        .p(),
41        .g()
42        );
43
44    endmodule
```