实验 2　16 位比较器实验

Chen-Yuanmeng*

2024/10/31

1　实验目的

1. 熟悉 Verilog 编程、调试
2. 熟悉简单比较器的工作原理
3. 通过简单模块例化、连线实现复杂的数字电路

2　实验环境

- Microsoft Windows 10.0.19045.5073
- Vivado v2017.4 (64-bit)
- 玉泉路一机房

3　原理说明

3.1　16 位比较器

16 位比较器由 4 个 4 位比较器相连而成. 一个 4 位比较器有高位输入, 本位数字输入两种输入, 输出为大于/等于/小于三种可能. 具体情况如下:

1. 当高位输入为 “>” 时, 输出为 “>”;
2. 当高位输入为 “=” 时:
   (a) 当本位数字 $A > B$ 时, 输出为 “>”;
   (b) 当本位数字 $A = B$ 时, 输出为 “=”;
   (c) 当本位数字 $A < B$ 时, 输出为 “<”.
3. 当高位输入为 “<” 时, 输出为 “<”.

要将 4 位比较器扩展到 16 位比较器, 则需要先比较 16 位数字中的最高 4 位, 再依次比较更低位. 故, 高位的输出应与低位的输入部分相连, 最高位输入应为 “=”, 最低位输出即为 16 位比较器的输出.

在实际设计过程中, 我们需要在 `comp_16` 模块中**例化** 4 个 `comp_4` 模块的示例, 进行对其的调用. 这是模块化设计思想的体现, 可以让代码复用减少, 增加程序的可读性和易维护性.

3.2　4 位超前进位加法器

超前进位加法器相比串行加法器, 可以实现更快速的计算. 其原理是, 加到第 i 位的进位输入信号是两个加数第 i 位以前各位的函数，可在相加前由 A, B 两数确定.

*Email: chenyumeng23@mails.ucas.ac.cn

设 $G_iA_iB_i$ 为进位生成函数, $P_iA_i + B_i$ 为进位传递函数. 此时可以得到

$$\begin{aligned}(CO)_i &= G_i + P_i(CI)_i \\ &= G_i + P_i(G_{i-1} + P_{i-1}(CI)_{i-1}) \\ &= G_i + P_iG_{i-1} + P_iP_{i-1}(G_{i-2} + P_{i-2}(CI)_{i-2}) \\ &= \cdots \\ &= G_i + P_iG_{i-1} + P_iP_{i-1}G_{i-2} + \cdots + P_iP_{i-1}\cdots P_1G_0 + P_iP_{i-1}\cdots P_1P_0(CI)_0,\end{aligned}$$

而

$$S_i = A_i \oplus B_i \oplus (CI)_i.$$

据此即可得到每一位的进位输入, 进而进行超前进位.

4 接口定义

4.1 4 位比较器

4 位比较器的接口定义如下:

```
module comp_4(
    input [3:0] A,       // 输入的待比较数字 A
    input [3:0] B,       // 输入的待比较数字 B
    input in_A_G_B,      // 高位输入, 表示高位A>B
    input in_A_E_B,      // 高位输入, 表示高位A=B或没有数字
    input in_A_L_B,      // 高位输入, 表示高位A<B
    output out_A_G_B,    // 输出, 为1时表示 A>B
    output out_A_E_B,    // 输出, 为1时表示 A=B
    output out_A_L_B     // 输出, 为1时表示 A<B
);
```

4.2 16 位比较器

16 位比较器的接口定义如下:

```
module comp_16(
    input [15:0] A,      // 输入的待比较数字 A
    input [15:0] B,      // 输入的待比较数字 A
    output out_A_G_B,    // 输出, 为1时表示 A>B
    output out_A_E_B,    // 输出, 为1时表示 A=B
    output out_A_L_B     // 输出, 为1时表示 A<B
);
```

4.3 4 位超前进位加法器

16 位比较器的接口定义如下:

```
module cla_adder_4(
    input [3:0] A,       // 加数 A
    input [3:0] B,       // 加数 B
    input Cin,           // 输入进位
    output [3:0] S,      // 输出位
```

```
    output Cout          // 输出进位
);
```

5 调试过程及结果

点击左侧 “SIMULATION” 下的 “Run Simulation”, 以默认设置进行模拟调试. 调试结果分别如图所示:

5.1 4 位比较器

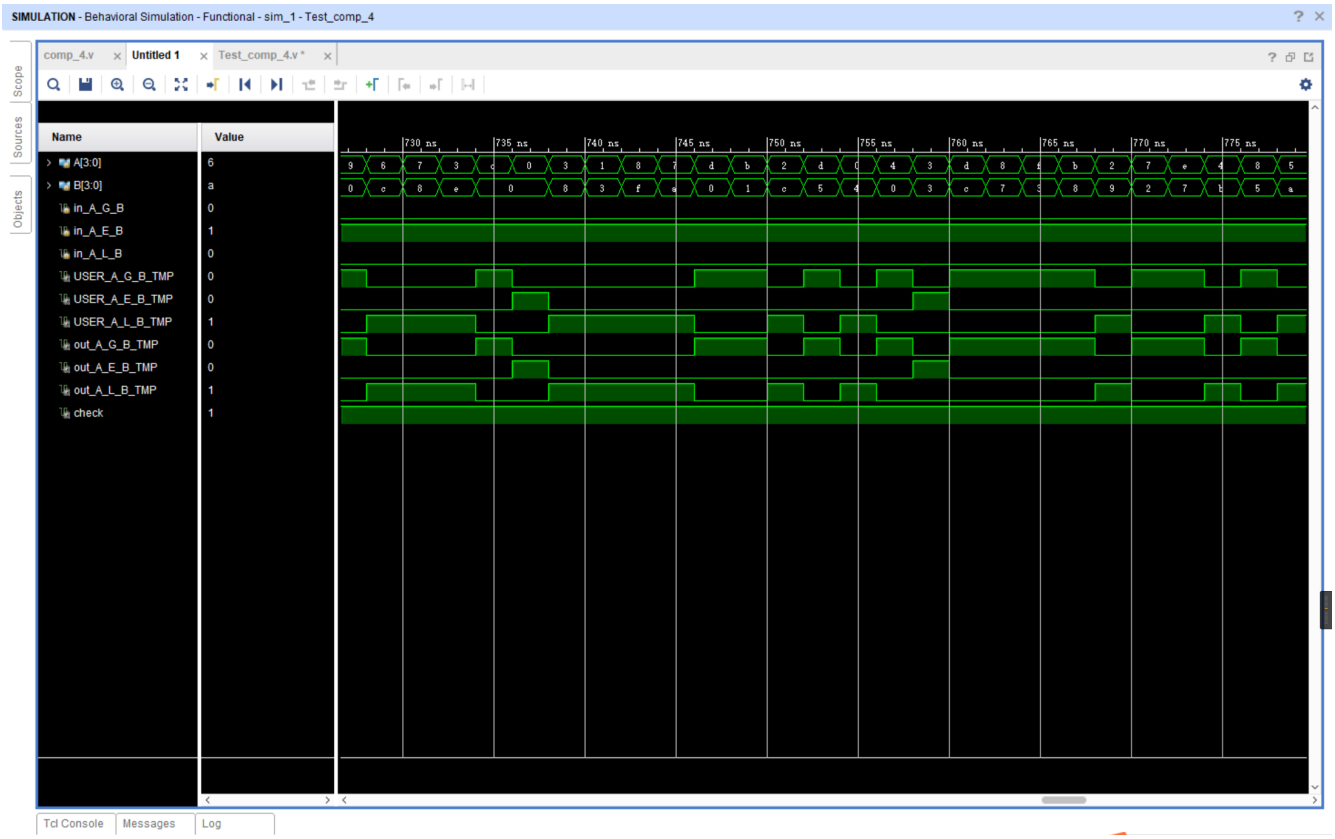

图 1: 4 位比较器 Simulation 结果

5.2 16 位比较器

图 2: 全加器 Simulation 结果

5.3 4 位超前进位加法器

图 3: 3–8 译码器 Simulation 结果

6 实验总结

通过本次实验, 我了解并实践了 Vivado 模块化设计的思想, 并对比较器和超前进位加法器的结构有了更深刻的认识. 此外, 我纠正了对于 `always` 语句的错误认识, 对组合逻辑电路和时序逻辑电路在 Verilog 设计中的不同有了更深刻的了解.

7 源代码

7.1 16 位比较器

7.1.1 4 位比较器

File: comp_4.v

```
`timescale 1ns / 1ps

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 out_A_G_B,
    output out_A_E_B,
    output out_A_L_B
    );

    reg A_G_B, A_E_B, A_L_B;

    always @(*) begin
    if (in_A_G_B == 1 && in_A_E_B == 0 && in_A_L_B == 0) begin
        A_G_B = 1;
        A_E_B = 0;
        A_L_B = 0;
        end
    else if (in_A_G_B == 0 && in_A_E_B == 1 && in_A_L_B == 0) begin
        A_G_B = 0;
        A_E_B = 0;
        A_L_B = 0;
        if (A>B)
            A_G_B = 1;
        else if (A == B)
            A_E_B = 1;
        else
            A_L_B = 1;
        end
    else begin
        A_G_B = 0;
        A_E_B = 0;
        A_L_B = 1;
```

```
        end
    end

    assign out_A_G_B = A_G_B;
    assign out_A_E_B = A_E_B;
    assign out_A_L_B = A_L_B;

endmodule
```

File: test_comp_4.v

```
`timescale 1ns / 1ps

module Test_comp_4(

    );

// ports you get
reg [3:0] A;
reg [3:0] B;
reg in_A_G_B;
reg in_A_E_B;
reg in_A_L_B;
wire USER_A_G_B_TMP;
wire USER_A_E_B_TMP;
wire USER_A_L_B_TMP;

// ports you needn't care at writing
wire out_A_G_B_TMP;
wire out_A_E_B_TMP;
wire out_A_L_B_TMP;
wire check = (USER_A_G_B_TMP === out_A_G_B_TMP) && (USER_A_E_B_TMP === out_A_E_B_TMP) &&
     (USER_A_L_B_TMP === out_A_L_B_TMP);

TEMPLATE_COMP_4 inst_comp_4_0 (
    .A(A),
    .B(B),
    .in_A_G_B(in_A_G_B),
    .in_A_E_B(in_A_E_B),
    .in_A_L_B(in_A_L_B),
    .out_A_G_B(out_A_G_B_TMP),
    .out_A_E_B(out_A_E_B_TMP),
    .out_A_L_B(out_A_L_B_TMP)
    );

// instantiate your module
/****************************** WRITE YOUR CODE HERE *******************************/
comp_4 inst_comp_4(
    .A(A),
    .B(B),
    .in_A_G_B(in_A_G_B),
    .in_A_E_B(in_A_E_B),
    .in_A_L_B(in_A_L_B),
```

```
    .out_A_G_B(USER_A_G_B_TMP),
    .out_A_E_B(USER_A_E_B_TMP),
    .out_A_L_B(USER_A_L_B_TMP)
);
/****************************** WRITE YOUR CODE HERE *******************************/

initial begin
    A = 4'd1;
    B = 4'd0;
    in_A_G_B=1'b0;
    in_A_E_B=1'b1;
    in_A_L_B=1'b0;
end

always begin
    #2;
    A = $random() % 16;
    B = $random() % 16;
end

endmodule
```

7.1.2 16 位比较器

File: comp_4.v 同上 `comp_4.v`.

File: comp_16.v

```
`timescale 1ns / 1ps

module comp_16(
    input [15:0] A,
    input [15:0] B,
    output out_A_G_B,
    output out_A_E_B,
    output out_A_L_B
    );

    wire mid_A_G_B_1, mid_A_E_B_1, mid_A_L_B_1;
    wire mid_A_G_B_2, mid_A_E_B_2, mid_A_L_B_2;
    wire mid_A_G_B_3, mid_A_E_B_3, mid_A_L_B_3;

    comp_4 comp_4_1 (
        .A(A[15:12]),
        .B(B[15:12]),
        .in_A_G_B(1'b0),
        .in_A_E_B(1'b1),
        .in_A_L_B(1'b0),
        .out_A_G_B(mid_A_G_B_1),
        .out_A_E_B(mid_A_E_B_1),
        .out_A_L_B(mid_A_L_B_1)
    );
```

```
    comp_4 comp_4_2 (
        .A(A[11:8]),
        .B(B[11:8]),
        .in_A_G_B(mid_A_G_B_1),
        .in_A_E_B(mid_A_E_B_1),
        .in_A_L_B(mid_A_L_B_1),
        .out_A_G_B(mid_A_G_B_2),
        .out_A_E_B(mid_A_E_B_2),
        .out_A_L_B(mid_A_L_B_2)
    );

    comp_4 comp_4_3 (
        .A(A[7:4]),
        .B(B[7:4]),
        .in_A_G_B(mid_A_G_B_2),
        .in_A_E_B(mid_A_E_B_2),
        .in_A_L_B(mid_A_L_B_2),
        .out_A_G_B(mid_A_G_B_3),
        .out_A_E_B(mid_A_E_B_3),
        .out_A_L_B(mid_A_L_B_3)
    );

    comp_4 comp_4_4 (
        .A(A[3:0]),
        .B(B[3:0]),
        .in_A_G_B(mid_A_G_B_3),
        .in_A_E_B(mid_A_E_B_3),
        .in_A_L_B(mid_A_L_B_3),
        .out_A_G_B(out_A_G_B),
        .out_A_E_B(out_A_E_B),
        .out_A_L_B(out_A_L_B)
    );

endmodule
```

File: test_comp_16.v

```
`timescale 1ns / 1ps

module Test_comp_16(

    );

    // ports you get
    reg [15:0] A;
    reg [15:0] B;
    wire USER_A_G_B_TMP;
    wire USER_A_E_B_TMP;
    wire USER_A_L_B_TMP;

    // ports you needn't care at writing
    wire out_A_G_B_TMP;
    wire out_A_E_B_TMP;
```

```
    wire out_A_L_B_TMP;
    wire check = (USER_A_G_B_TMP === out_A_G_B_TMP) && (USER_A_E_B_TMP === out_A_E_B_TMP) &&
        (USER_A_L_B_TMP === out_A_L_B_TMP);

    TEMPLATE_COMP_16 inst_comp_16_0 (
    .A(A),
    .B(B),
    .A_G_B(out_A_G_B_TMP),
    .A_E_B(out_A_E_B_TMP),
    .A_L_B(out_A_L_B_TMP)
    );

    // instantiate your module
    /****************************** WRITE YOUR CODE HERE *******************************/
    comp_16 inst_comp_16 (
    .A(A),
    .B(B),
    .out_A_G_B(USER_A_G_B_TMP),
    .out_A_E_B(USER_A_E_B_TMP),
    .out_A_L_B(USER_A_L_B_TMP)
    );
    /****************************** WRITE YOUR CODE HERE *******************************/

    initial begin
        A = 16'd1;
        B = 16'd0;
    end

    always begin
        #2;
        A = $random() % 16;
        B = $random() % 16;
    end

endmodule
```

7.2 4 位超前进位加法器

File: cla_adder_4.v

```
`timescale 1ns / 1ps

module cla_adder_4(
    input [3:0] A,
    input [3:0] B,
    input Cin,
    output [3:0] S,
    output Cout
    );

    wire [3:0] G;
    wire [3:0] P;
```

```
    wire [3:0] C;

    assign G = A & B;
    assign P = A ^ B;

    assign C[0] = G[0] | (P[0] & Cin);
    assign C[1] = G[1] | (P[1] & G[0]) | (P[1] & P[0] & Cin);
    assign C[2] = G[2] | (P[2] & G[1]) | (P[2] & P[1] & P[0] & Cin);
    assign C[3] = G[3] | (P[3] & G[2]) | (P[3] & P[2] & P[1] & P[0] & Cin);

    assign S = A ^ B ^ {C[2:0], Cin};

    assign Cout = C[3];
endmodule
```

File: test_cla_adder_4.v

```
`timescale 1ns / 1ps

module test_cla_adder_4();

    reg [3:0] A, B;
    reg Cin;
    wire Cout;
    wire [3:0] S;

    cla_adder_4 inst_cla_add_4 (
        .A(A),
        .B(B),
        .Cin(Cin),
        .S(S),
        .Cout(Cout)
    );

    initial begin
    A = 1'b0;
    B = 1'b0;
    Cin = 1'b0;
    end

    always begin
    #2;
    A = $random() % 16;
    B = $random() % 16;
    Cin = $random() % 2;
    end

endmodule
```