# 实验三：4位加法器设计与实现

1. **实验目的**
   * + 1. 通过本次实验，需要学生掌握加法器的运作原理以及基本设计思路
2. **实验内容**
   * + 1. 使用Verilog语言编写逻辑实现两个4bits有符号数或无符号数补码的输入（输入使用拨号开关）
       2. 两数进行加减运算与其4bits运算结果的输出，并将运算数与运算结果分别显示在开发板的7段数码管上
       3. 运算过程产生的标志位（本次实验包括SF、ZF、CF、OF）送LED显示
3. **实验原理**
   * + 1. 有符号数：两个有符号数的加法可以表示为两数补码的加法运算；两个有符号数的减法，可以先将减数进行取反加一得到减数对应取负的数的补码，进而再将被减数的补码与其相加，便可得到原两数相减结果的补码；
       2. 无符号数：两个无符号数的加法为简单的二进制加法运算；对于两个无符号数之间的减法，例如：A-B，可以表示为：A + (2^n – B) - 2^n，其中n为A与B的位数，此时(2^n-B)可看作为B的补码，最终可将问题转换为1中的两数补码的相加（A为无符号数，其补码看作为其本身）；
       3. 如何表示加减法：设定符号位M，令M为0时表示两数的加法运算；M为1时表示两数之间的减法运算。根据1与2中的分析，同时运用异或操作的特性：A异或1为A取反；A与0异或为A本身，可以得到以下设计思路：只要将两个操作数其一的每一位与M进行异或，便可表示两个操作数之间的加法与减法操作；
       4. SF：sign flag的设置取决于结果的符号位，也即是结果的最高位，最高位为1置1，反之置0；
       5. ZF：zero flag的设置取决于结果是否为0，为0时置1，反之置0；
       6. CF：carry flag的设置取决于无符号数运算时是否在最高位发生进位或借位，可将其表示为C3异或M（其中C3为两数补码相加最高位的进位）；
       7. OF：overflow flag的设置取决于有符号数运算时是否产生符号位溢出，可将其表示为C3异或C2（其中C3与C2分别为两数补码相加最高位与次最高位的进位）；
4. **实验器材**
   * + 1. BASY3开发板
       2. 装有Xilinx Vivado软件的电脑一台
5. **实验过程纪录**

**INPUT：**

<R2>拨号开关：有符号加减法运算（default）

<T1>拨号开关：无符号加减法运算

<W13><W14><V15><W15>拨号开关：NUM1

<W17><W16><V16><V17>拨号开关：NUM2

<T18> button：M置1

<U17> button：M置0

<W19> button：isResult置1

<T17> button ：isResult置0

**OUTPUT：**

七段数码管：isResult为1时显示运算结果；反之显示两个操作数（负数无负号表示）

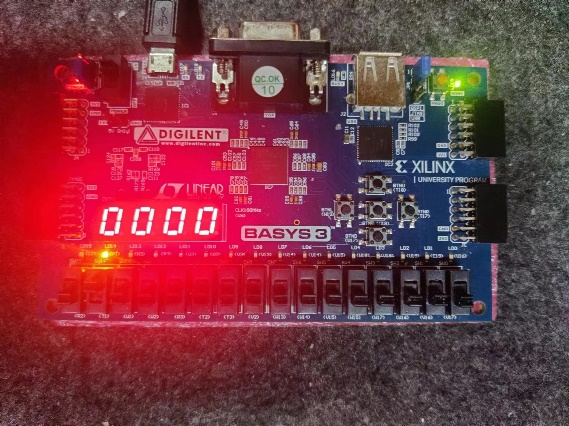
<L1>：SF

<P1>：ZF

<N3>：CF

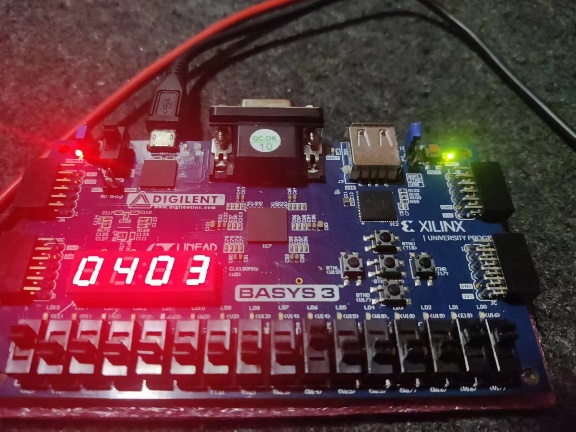
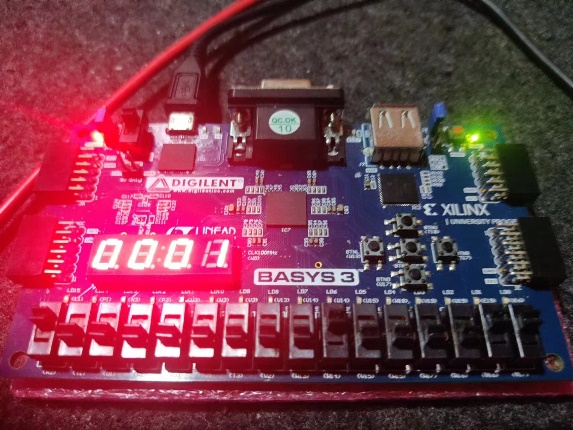
<P3>：OF

**开发板初始状态：**



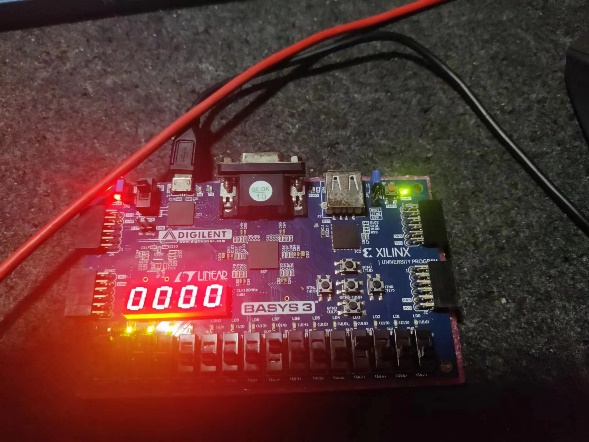
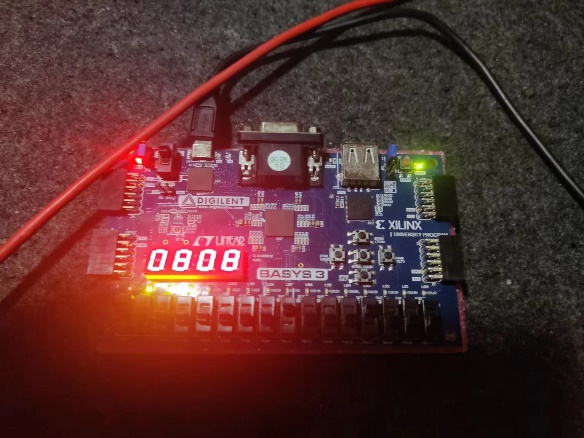
**计算有符号加法：**

4 + (-3) = 1



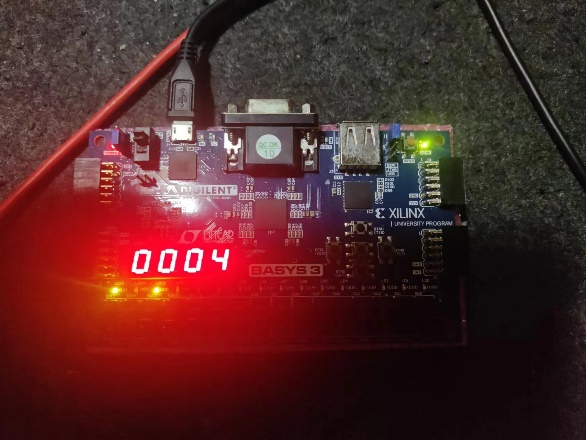
4 + 4 = -8（正溢出）

(-8) + (-8) = 0 (负溢出)



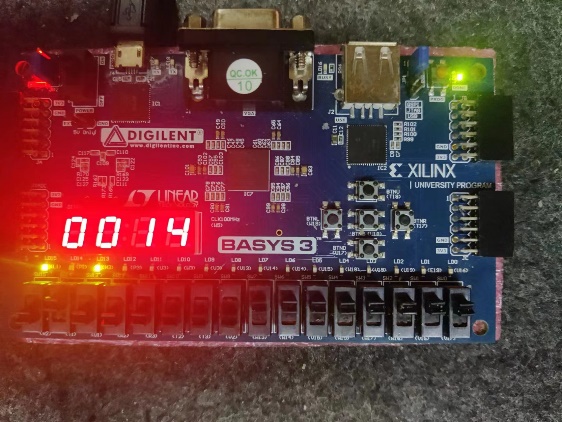
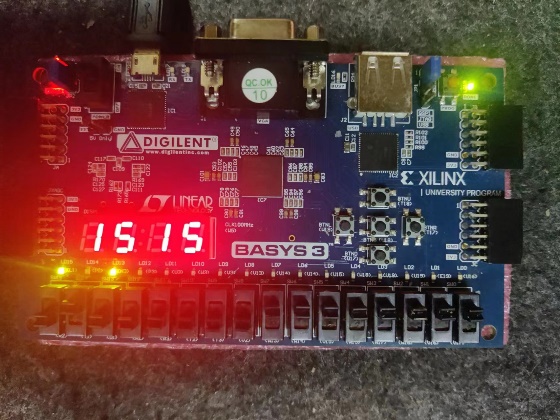
**计算有符号减法：**

0 – 4 = (-4)

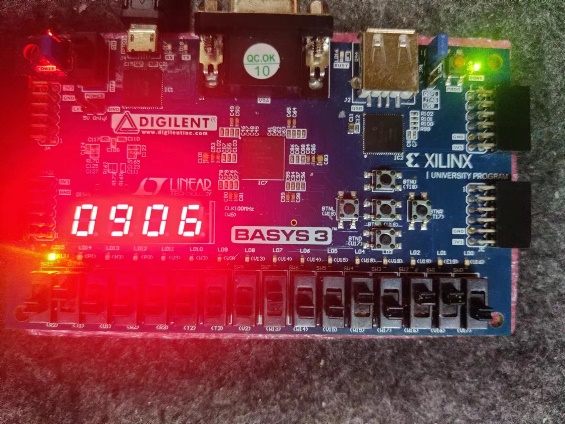
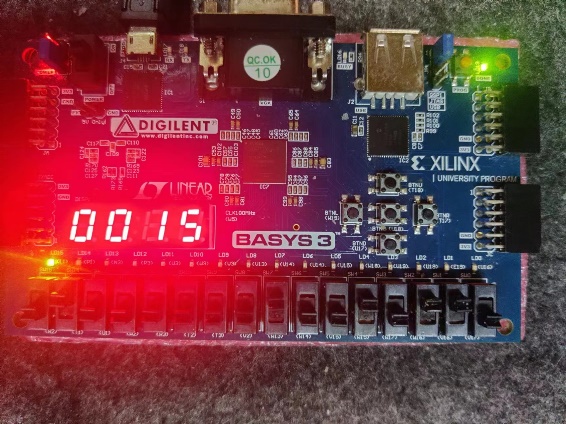


**计算无符号加法：**

15 + 15 = 14（最高位进位）

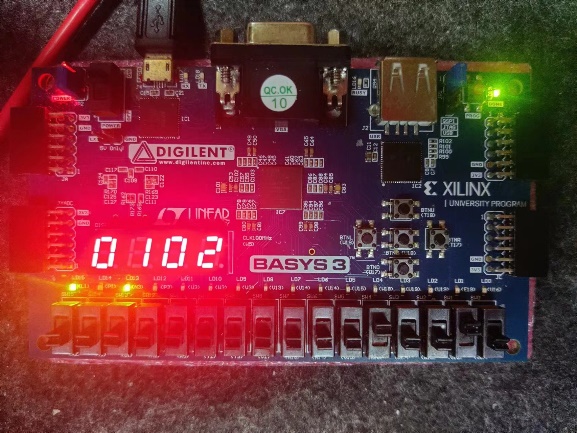
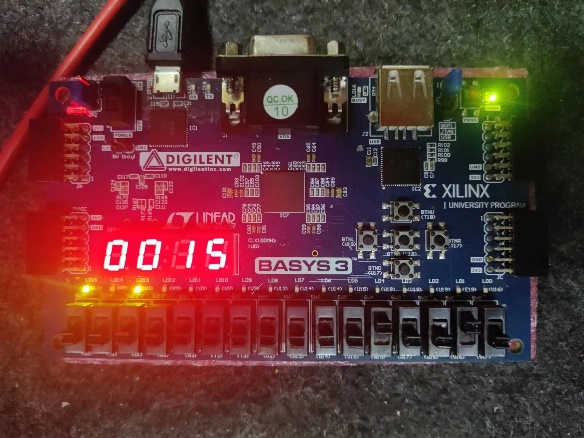


6 + 9 = 15

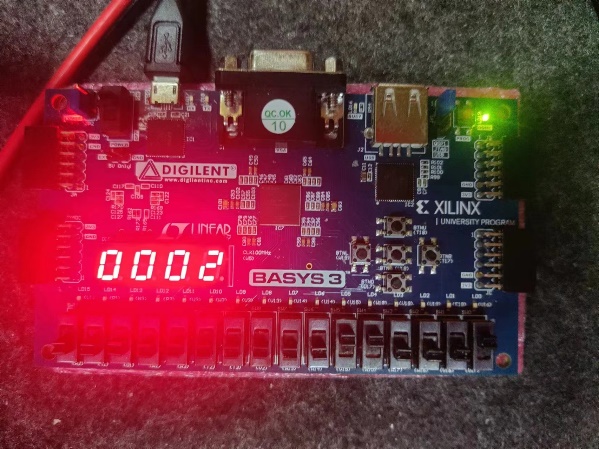
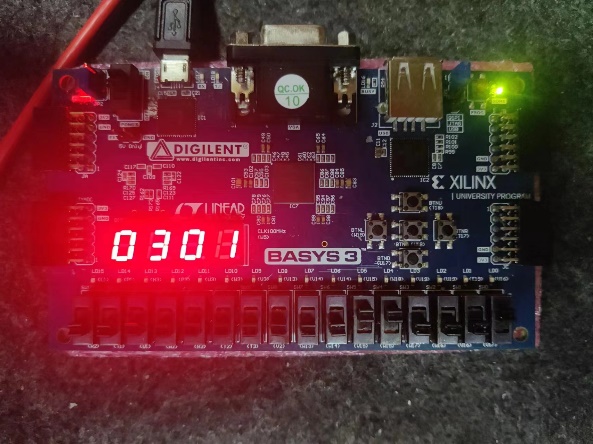


**计算无符号减法：**

1 – 2 = 15（最高位借位）



3 – 1 = 2



实现代码：

**Top.v:**

`timescale 1ns / 1ps

*//////////////////////////////////////////////////////////////////////////////////*

*// Company:*

*// Engineer:*

*//*

*// Create Date: 2023/10/11 21:04:00*

*// Design Name:*

*// Module Name: Top*

*// Project Name:*

*// Target Devices:*

*// Tool Versions:*

*// Description:*

*//*

*// Dependencies:*

*//*

*// Revision:*

*// Revision 0.01 - File Created*

*// Additional Comments:*

*//*

*//////////////////////////////////////////////////////////////////////////////////*

module Top(

    input CLK,

    input [3:0] Num1,

    input [3:0] Num2,

    input [7:0] Control,

    input button1, *// button1 and button2 control the value of isResult*

    input button2,

    input button3, *// button3 and button4 control the value of M*

    input button4,

    output SF,

    output ZF,

    output CF,

    output OF,

    output [3:0] seg,

    output [6:0] a\_to\_g

    );

    wire CLK\_wire;

    wire [3:0] Result;

    wire [3:0] ALUcontrol;

    wire [7:0] mux2\_result;

    reg isResult = 0;

    reg isSigned = 0;

    reg M = 0;

    always @(\*) begin

        if (button1 == 1)

            isResult <= 1;

        else if (button2 == 1)

            isResult <= 0;

        if (button3 == 1)

            M <= 1;

        else if (button4 == 1)

            M <= 0;

        isSigned <= Control[7];

    end

    CLK\_div \_CLK\_div (

        .CLK\_in(CLK),

        .CLK\_out(CLK\_wire)

    );

    ShowNum \_showNum (

        .CLK(CLK\_wire),

        .\_show\_num(mux2\_result),

        .isResult(isResult),

        .isSigned(isSigned),

        .SF(SF),

        .seg(seg),

        .a\_to\_g(a\_to\_g)

    );

    toALUcontrol \_toALUcontrol (

        .Control(Control),

        .ALUcontrol(ALUcontrol)

    );

    ALU \_ALU (

        .Num1(Num1),

        .Num2(Num2),

        .Control(ALUcontrol),

        .M(M),

        .SF(SF),

        .ZF(ZF),

        .CF(CF),

        .OF(OF),

        .Result(Result)

    );

    Mux2 \_mux2 (

        .Data1({Num1, Num2}),

        .Data2({4'b0000, Result}),

        .Sel(isResult),

        .Result(mux2\_result)

    );

endmodule

**ALU.v**

`timescale 1ns / 1ps

*//////////////////////////////////////////////////////////////////////////////////*

*// Company:*

*// Engineer:*

*//*

*// Create Date: 2023/10/11 20:56:30*

*// Design Name:*

*// Module Name: ALU*

*// Project Name:*

*// Target Devices:*

*// Tool Versions:*

*// Description:*

*//*

*// Dependencies:*

*//*

*// Revision:*

*// Revision 0.01 - File Created*

*// Additional Comments:*

*//*

*//////////////////////////////////////////////////////////////////////////////////*

module ALU(

    input [3:0] Num1,

    input [3:0] Num2,

    input [3:0] Control,

    input M, *// 0 for add, 1 for sub*

    output SF,

    output ZF,

    output CF,

    output OF,

    output reg [3:0] Result

    );

    reg [3:0] c\_out; *// carry out*

    reg c\_in; *// carry in*

    reg cg, cp; *// carry generate, carry propagate*

    reg [3:0] i;

    always @(\*) begin

*// calculate carry*

        if (M == 0) begin

            c\_in = 0;

            for (i = 0; i < 4; i = i + 1) begin

                cg = Num1[i] & Num2[i];

                cp = Num1[i] | Num2[i];

                c\_out[i] = cg | (cp & c\_in);

                c\_in = c\_out[i];

            end

        end

        else begin

            c\_in = 1;

            for (i = 0; i < 4; i = i + 1) begin

                cg = Num1[i] & ~Num2[i];

                cp = Num1[i] | ~Num2[i];

                c\_out[i] = cg | (cp & c\_in);

                c\_in = c\_out[i];

            end

        end

        case (Control)

            4'b0000, 4'b0001: begin *// b0000 for unsigned, b0001 for signed*

                if (M == 0)

                    Result = Num1 + Num2;

                else

                    Result = Num1 + ~Num2 + 1;

            end

            default: Result = 4'b0000;

        endcase

    end

    assign CF = c\_out[3] ^ M;

    assign OF = c\_out[3] ^ c\_out[2];

    assign SF = Result[3];

    assign ZF = (Result == 0);

endmodule

**toALUcontrol.v**

`timescale 1ns / 1ps

*//////////////////////////////////////////////////////////////////////////////////*

*// Company:*

*// Engineer:*

*//*

*// Create Date: 2023/10/11 21:05:17*

*// Design Name:*

*// Module Name: toALUcontrol*

*// Project Name:*

*// Target Devices:*

*// Tool Versions:*

*// Description:*

*//*

*// Dependencies:*

*//*

*// Revision:*

*// Revision 0.01 - File Created*

*// Additional Comments:*

*//*

*//////////////////////////////////////////////////////////////////////////////////*

module toALUcontrol(

    input [7:0] Control,

    output reg [3:0] ALUcontrol

    );

always @(Control) begin

    case (Control)

        8'b10000000: ALUcontrol = 4'b0000; *// signed*

        8'b01000000: ALUcontrol = 4'b0001; *// unsigned*

        default: ALUcontrol = 4'b0000;

    endcase

end

endmodule

**Mux2.v**

`timescale 1ns / 1ps

*//////////////////////////////////////////////////////////////////////////////////*

*// Company:*

*// Engineer:*

*//*

*// Create Date: 2023/10/11 21:04:29*

*// Design Name:*

*// Module Name: Mux2*

*// Project Name:*

*// Target Devices:*

*// Tool Versions:*

*// Description:*

*//*

*// Dependencies:*

*//*

*// Revision:*

*// Revision 0.01 - File Created*

*// Additional Comments:*

*//*

*//////////////////////////////////////////////////////////////////////////////////*

module Mux2(

    input [7:0] Data1,

    input [7:0] Data2,

    input Sel,

    output reg [7:0] Result

    );

    always @(Data1 or Data2 or Sel) begin

        if (Sel == 0)

            Result = Data1;

        else

            Result = Data2;

    end

endmodule

**CLK\_div.v**

`timescale 1ns / 1ps

*//////////////////////////////////////////////////////////////////////////////////*

*// Company:*

*// Engineer:*

*//*

*// Create Date: 2023/10/11 21:04:44*

*// Design Name:*

*// Module Name: CLK\_div*

*// Project Name:*

*// Target Devices:*

*// Tool Versions:*

*// Description:*

*//*

*// Dependencies:*

*//*

*// Revision:*

*// Revision 0.01 - File Created*

*// Additional Comments:*

*//*

*//////////////////////////////////////////////////////////////////////////////////*

module CLK\_div #(parameter N = 99999) (

    input CLK\_in,

    output CLK\_out

    );

    reg [31:0] cnt = 0;

    reg out = 0;

    always @(posedge CLK\_in) begin

        if (cnt == N) begin

            cnt <= 0;

            out <= ~out;

        end

        else begin

            cnt <= cnt + 1;

        end

    end

    assign CLK\_out = out;

endmodule

**ShowNum.v**

`timescale 1ns / 1ps

*//////////////////////////////////////////////////////////////////////////////////*

*// Company:*

*// Engineer:*

*//*

*// Create Date: 2023/10/11 21:05:01*

*// Design Name:*

*// Module Name: ShowNum*

*// Project Name:*

*// Target Devices:*

*// Tool Versions:*

*// Description:*

*//*

*// Dependencies:*

*//*

*// Revision:*

*// Revision 0.01 - File Created*

*// Additional Comments:*

*//*

*//////////////////////////////////////////////////////////////////////////////////*

module ShowNum(

    input CLK,

    input [7:0] \_show\_num,

    input isResult,

    input isSigned,

    input SF,

    output reg [3:0] seg,

    output reg [6:0] a\_to\_g

    );

    reg [7:0] show\_num = 0;

    reg [3:0] seg\_num = 0;

    reg [3:0] x = 0;

    reg sign1, sign2;

    reg [3:0] num1;

    reg [3:0] num2;

    always @(posedge CLK) begin

        if (isResult)  begin

            if (isSigned && SF)

                show\_num = ((~\_show\_num) + 1) & 8'h0f; *// convert to 2's complement and only keep last 4 bits*

            else

                show\_num = \_show\_num & 8'h0f;

            case (seg\_num)

                0: begin x = show\_num / 1000; seg = 4'b0111; end

                1: begin x = show\_num / 100 % 10; seg = 4'b1011; end

                2: begin x = show\_num / 10 % 10; seg = 4'b1101; end

                3: begin x = show\_num % 10; seg = 4'b1110; end

            endcase

        end

        else begin

            if (isSigned) begin

                sign1 = \_show\_num[7]; *// sign bit of num1*

                sign2 = \_show\_num[3]; *// sign bit of num2*

                if (sign1)

                    num1 = (~\_show\_num[7:4] + 1);

                else

                    num1 = \_show\_num[7:4];

                if (sign2)

                    num2 = (~\_show\_num[3:0] + 1);

                else

                    num2 = \_show\_num[3:0];

                show\_num = {num1, num2};

            end

            else begin

                show\_num = \_show\_num;

            end

            case (seg\_num)

                0: begin x = show\_num[7:4] / 10; seg = 4'b0111; end

                1: begin x = show\_num[7:4] % 10; seg = 4'b1011; end

                2: begin x = show\_num[3:0] / 10; seg = 4'b1101; end

                3: begin x = show\_num[3:0] % 10; seg = 4'b1110; end

            endcase

        end

        seg\_num = (seg\_num + 1) % 4;

    end

    always @(\*) begin

        case (x)

            0: a\_to\_g = 7'b0000001;

            1: a\_to\_g = 7'b1001111;

            2: a\_to\_g = 7'b0010010;

            3: a\_to\_g = 7'b0000110;

            4: a\_to\_g = 7'b1001100;

            5: a\_to\_g = 7'b0100100;

            6: a\_to\_g = 7'b0100000;

            7: a\_to\_g = 7'b0001111;

            8: a\_to\_g = 7'b0000000;

            9: a\_to\_g = 7'b0000100;

            default: a\_to\_g = 7'b1111111;

        endcase

    end

endmodule