四位二进制数的可控加法/减法

徐康彦 2017 年 4 月

实验基本要求

用拨码开关输入两个四位二进制数，按下运算钮后，以 led 的方式显示运算的结果。

实验扩展要求

1．在基本功能实现前提下，增加一个清零功能。

2．更换显示方式，结果用数码管进行显示。

3．对溢出值的考虑。如果发生溢出现象，则数码管显示“E”。

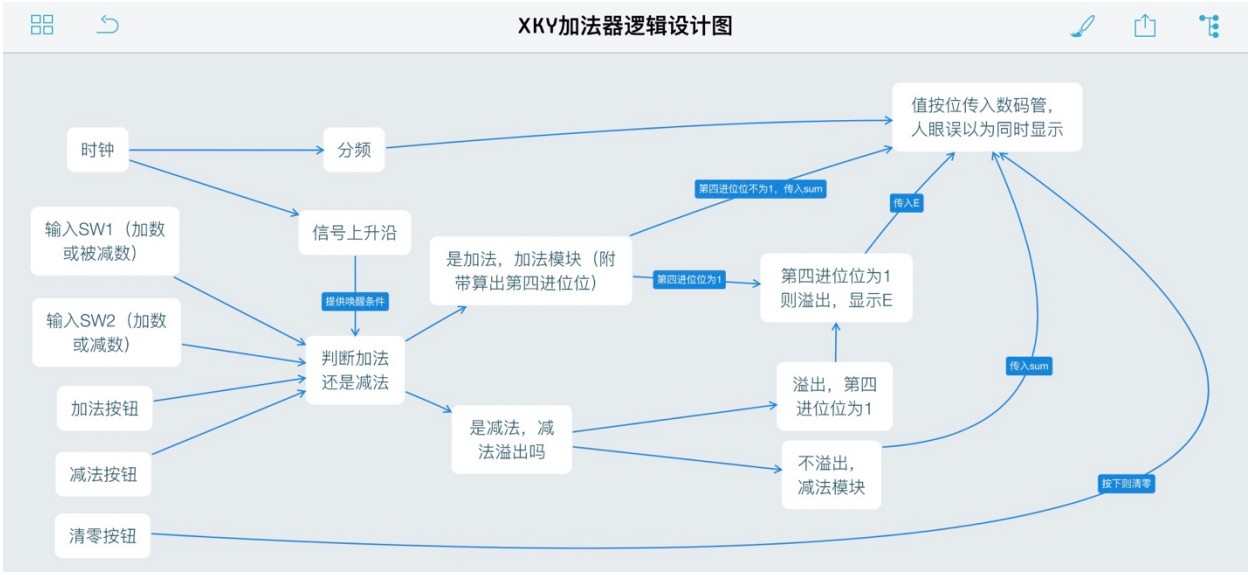
4．放弃拨码开关的输入方式，按照算式方法输入。两个微动开关代表“1”和“0”，两个开关代表“＋”和“－”，还有一个开关代表“＝”。比如要算 0000＋1111，就按顺序输入这些数字和运算符，同时数码管动态显示输入值，等按下“＝”后显示结果。用点阵显示运算符。

一、实验设计

本实验中完成了扩展要求的 1、2、3。

加法器是数电课程的教学内容。本实验中的加法设计依照于此：输入两个待相加的位，输出一个 sum 位与进位位。当第四个进位位为 1 则说明溢出。而减法较为复杂，需要将减数先取反码，然后与被减数相加，再与 0001 相加。由于在无符号数的减法中用进位位判断溢出有困难，所以直接通过比较被减数与减数每一位的大小来判断溢出。

二、逻辑设计图



三、源程序

1、底层模块

```
module XKY_homework_adder_sub(
    input[1:0]NUM,                          选 2 位二进制，因为只有 0、1、E 三种可能
    output reg[6:0]a_to_g);
    always@(*)
    begin
```

```
    case(NUM)
    0:a_to_g=7'b0000001;
    1:a_to_g=7'b1001111;
    2:a_to_g=7'b0110000;
    default:a_to_g=7'b1111111;  //total dark
    endcase
    end
endmodule
```

2、顶层模块

```
module XKY_homework_adder_top(
   input wire[3:0]SW1,
   input wire[3:0]SW2,     输入 SW1（被减数）(左 4 开关）与 SW2（减数）(右 4 开关）
   input add,              左 1 按钮
   input sub,              左 2 按钮
   input clr,              左 4 按钮
   input clk,
   output reg[3:0]an,
   output[6:0]a_to_g);
   reg[1:0] seg_choose;          数码管上要显示的对应的 0、1、E
   reg[3:0] c;                   4 个进位位
   reg[3:0] sum;                 和或差（不带进位位）
   reg[3:0] sumsubs;             转存上述 sum
   reg[15:0] count_for_clk=0;  //65536 分频 50MHz

always@(posedge clk)
    begin
```

2.1 分频模块

```
    count_for_clk<=count_for_clk+1;   计数器功能

    case(count_for_clk[15:14])
    0: seg_choose=sum[0];           //选数码管 0 送入和或差第 0 位
    1: seg_choose=sum[1];           //选数码管 1 送入和或差第 1 位
    2: seg_choose=sum[2];           //选数码管 2 送入和或差第 2 位
    3: seg_choose=sum[3];           //选数码管 3 送入和或差第 3 位
    endcase
```

//0to16383 选一段，16384to32767 选 2 段，32768to49151 选三段，49152to65535 选 4 段，每段 16384*20=327680ns,3052Hz,人眼会认为四段常亮。

```
    case(count_for_clk[15:14])
    0: an=4'b0111;                //选通数码管 0
    1: an=4'b1011;                //选通数码管 1
    2: an=4'b1101;                //选通数码管 2
```

```
    3: an=4'b1110;                    //选通数码管 3
    endcase
//管教定义时注意顺序
```

2．2 清零模块

```
    if(clr)
    begin
    sum=4'b0000;      送入 0000
    c[3]=0;           进位位也清零
    end
```

2.3 加法模块

```
    if(add)
    begin
    sum[0]=SW1[0]^SW2[0];
    c[0]=SW1[0]&SW2[0];
    sum[1]=SW1[1]^SW2[1]^c[0];
    c[1]=(SW1[1]&SW2[1])|(c[0]&(SW1[1]^SW2[1]));
    sum[2]=SW1[2]^SW2[2]^c[1];
    c[2]=(SW1[2]&SW2[2])|(c[1]&(SW1[2]^SW2[2]));
    sum[3]=SW1[3]^SW2[3]^c[2];
    c[3]=(SW1[3]&SW2[3])|(c[2]&(SW1[3]^SW2[3]));
    end
```

2.4 减法模块

```
if(sub&((SW1[3]>SW2[3])|((SW1[3]==SW2[3])&(SW1[2]>SW2[2]))|((SW1[3]==SW2[3])&(S
W1[2]==SW2[2])&(SW1[1]>SW2[1]))|((SW1[3]==SW2[3])&(SW1[2]==SW2[2])&(SW1[1]==SW2
[1])&(SW1[0]>=SW2[0]))))

//按下减法且被减数比减数大时
    begin
    sum[0]=SW1[0]^~SW2[0];
    c[0]=SW1[0]&~SW2[0];
    sum[1]=SW1[1]^~SW2[1]^c[0];
    c[1]=(SW1[1]&~SW2[1])|(c[0]&(SW1[1]^~SW2[1]));
    sum[2]=SW1[2]^~SW2[2]^c[1];
    c[2]=(SW1[2]&~SW2[2])|(c[1]&(SW1[2]^~SW2[2]));
    sum[3]=SW1[3]^~SW2[3]^c[2];
//与减数反码相加
    assign sumsubs[0]=sum[0];
    assign sumsubs[1]=sum[1];
    assign sumsubs[2]=sum[2];
    assign sumsubs[3]=sum[3];
//转存相加后的结果
```

```
    sum[0]=1^sumsubs[0];
    c[0]=1&sumsubs[0];
    sum[1]=0^sumsubs[1]^c[0];
    c[1]=(0&sumsubs[1])|(c[0]&(0^sumsubs[1]));
    sum[2]=0^sumsubs[2]^c[1];
    c[2]=(0&sumsubs[2])|(c[1]&(0^sumsubs[2]));
    sum[3]=0^sumsubs[3]^c[2];
    end
//再与 0001 相加
```

2．5 减法溢出判断模块

```
if(sub&~((SW1[3]>SW2[3])|((SW1[3]==SW2[3])&(SW1[2]>SW2[2]))|((SW1[3]==SW2[3])&(
SW1[2]==SW2[2])&(SW1[1]>SW2[1]))|((SW1[3]==SW2[3])&(SW1[2]==SW2[2])&(SW1[1]==SW
2[1])&(SW1[0]>=SW2[0]))))
//按下减法而且被减数比减数小
    begin
    c[3]=1; //让第四进位位为 1，这样与加法判断溢出统一起来
    end
```

2.6 溢出显示 E 模块

```
if(c[3])
    begin
    an=4'b0111;
    seg_choose=2'b10;  //显示 E
    end

end //always 模块结束

XKY_homework_adder_sub A1(.NUM(seg_choose),
                          .a_to_g(a_to_g));

endmodule
```

四、调试中所碰到的问题与解决方法

1.“=”与“<=”混用会报错，于是改为“=”。

2.外部输入 SW1 与 SW2 后便不可再赋值,不然报错,故删除了类似于“SW1[0]=~SW1[0]”的语句。

3.本来想将 seg_choose 以 integer 的形式传入 NUM 中，然而无法显示 E，故后改用两位二进制来传递。

4.不将 if 语句写入 always 块会报错。

5．一开始直接将“按下减法而且被减数比减数小”作为判断溢出的条件，导致只在长按 sub 按钮时才显示溢出 E，后通过对 2.5、2.6 模块的修改解决了问题。

五、实验心得

每次面对一种新的语言，我都采用“少看书，先实战”的方法。这次的Verilog语言也是如此。仅依靠三页的《Verilog 语法快速指南》，或是细心研究课堂代码，不但效果不明显，反倒大有可能被那繁杂的语句吓住，“灭自己威风”。还不如先在ISE中写出代码的整体框架，再慢慢“添砖加瓦”、小修小补。

代码的核心其实很容易，无非是半加器的原理，以及减法用补码、考虑溢出的判断。麻烦的无非是一些分频模块、还有变量的设置。这些只要慢慢调试，碰到bug上网去搜索，一般常见的bug网上都有解决方案，依此修改代码便可。

然而就算编译通过，写入到FPGA板上也会有各种问题，往往是由于写代码时考虑欠佳所致，这些逻辑问题就需要通过重新思考代码来解决了。

不足之处在于虽已实现要求，但代码仍有冗余。

最后，感谢本课程提供了了解FPGA、Verilog语言的机会，也感谢老师的悉心教导。