

目录

- 1、概述.....1
- 2、功能介绍.....1
  - 2.1 计算器功能.....1
  - 2.2 计算器功能细节 .....2
  - 2.3 游戏功能 .....2
  - 2.4 功能展示 .....3
    - 2.4.1 计算器功能展示.....3
    - 2.4.2 游戏功能展示.....5
- 3、模块结构.....5
  - 3.1 模块化设计.....5
  - 3.2 主功能模块及其子模块.....6
    - 3.3.1 加法器模块 MODULE\_Adder.....6
    - 3.3.2 游戏模块 MODULE\_1.....7
    - 3.3.3 游戏模块 MODULE\_2.....8
- 4、实现方法.....9
  - 4.1 键盘模块 .....9
    - 4.1.1 输入与输出.....9
    - 4.1.2 实现方法.....11
  - 4.2 加法器模块.....14
    - 4.2.1 输入与输出.....14
    - 4.2.2 功能实现.....14

# 1、概述

基于 FPGA 设计并开发出了一款带有游戏功能的计算器，游戏名为《拆弹专家》。计算器功能和游戏功能由用户在开发板上通过拨码开关选择，当工作于计算器模式下时，能够实现 6 位十进制的加法功能。共有两种游戏模式，提供不同的游戏难度。用户的主要操作在矩阵键盘上进行，信息由数码管和 LED 灯进行反馈。开发环境为 Vivado，开发语言采用 Verilog，整个工程采用模块化实现。经过多次测试，所有的功能均能实现，且工作稳定。

## 2、功能介绍

### 2.1 计算器功能

打开拨码开关 sw11，计算器打开，等待输入。使用矩阵键盘输入加数，显示于数码管，加数输入完成后，按“+”键，数码管清空，等待输入被加数，被加数输入完成后，按下“=”键，两数之和显示于数码管。用 LED9 的亮灭表示最高位的进位，亮，则表示进位为 1，否则进位为 0。

在键入过程中如果需要修改数字，可以通过“DEL”键退格。

计算完成后，若想再次计算，可通过“R”键重启，再次输入加数与被加数，过程同上。

## 2.2 计算器功能细节

用户输入的数字最高到第 6 位，如果此时继续输入，计算器不会有反应，也不会影响退位（无论超出多少次，按一次退位键也能从 6 位退至 5 位）；

用户在没有任何键入时按退位键，不会有任何影响；

本计算器不支持连加操作，如果用户以连加的方式输入，多次按下加号，只计算收尾两个数相加的结果； $(a + b + c + d = a + d)$

用户能在任何时候按下“R”键，重新计算

## 2.3 游戏功能

游戏名为《拆弹专家》，所谓“拆弹”的过程实则是指玩家需要在有限时间内计算出任意给出的两个数的和。采用蜂鸣器发出响声的变化以及 LED 灯红光闪烁的变化来模拟真实的定时炸弹，增加游戏的趣味性和紧张感。有两种游戏难度，简单模式下计算的是个位数加法，困难模式下计算的是十位数加法。下面具体说明其功能和玩法。

打开拨码开关 sw10 或 sw9（两种游戏难度，sw10 为简单，sw9 为困难），进入战场。作为拆弹专家的你首先需要获取炸弹，按下矩阵键盘上的指定按键，此时数码管将显示一个随机数，请最好默记这个数。再次按下该键，获取第二个随机数，炸弹立刻被触发，滴滴声响起，伴随同步的红光闪烁，你需要抓紧时间口算出这两个

数之和（也可以笔算，但没有时间）。随着时间推进，滴滴声越来越快，红光闪烁得越来越急促，你越来越紧张。

你的结果需要在键盘上输入并按下“=”。若未在时间范围内输入，炸弹爆炸；若输入错误的结果，炸弹提前爆炸；若及时输入正确的结果，炸弹被成功拆除。

如果炸弹爆炸，蜂鸣器将持续发出低音 do，原先闪烁的红光保持亮；

如果炸弹被拆除，原先的红光熄灭，4 位流水灯开始流动，蜂鸣器播放欢快的小星星片段。

## 2.4 功能展示

### 2.4.1 计算器功能展示

(1) 计算  $120000+240000$

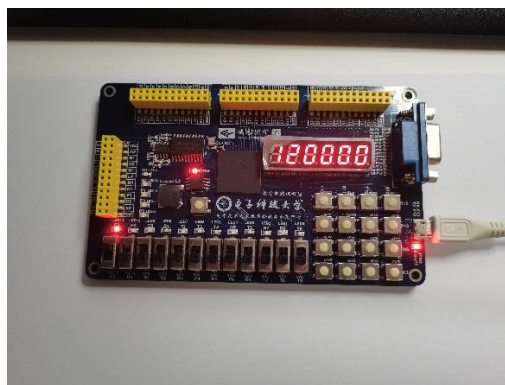


Figure 1

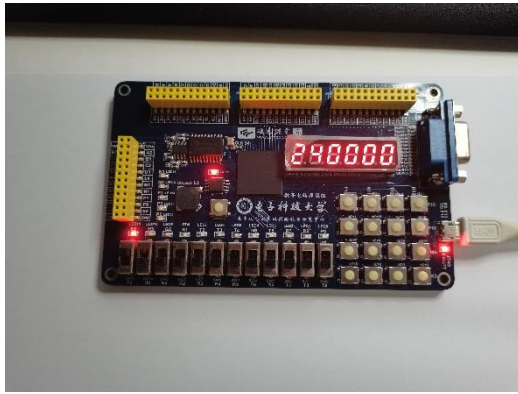


Figure 2

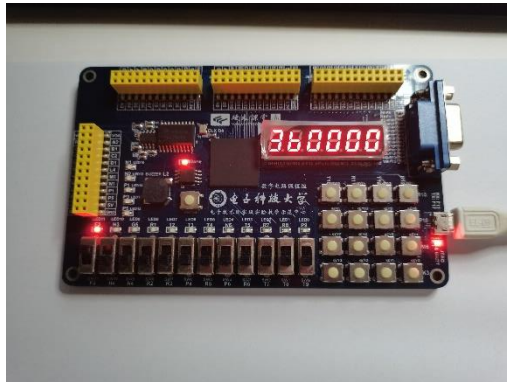


Figure 3

(2) 计算  $967765+256879$

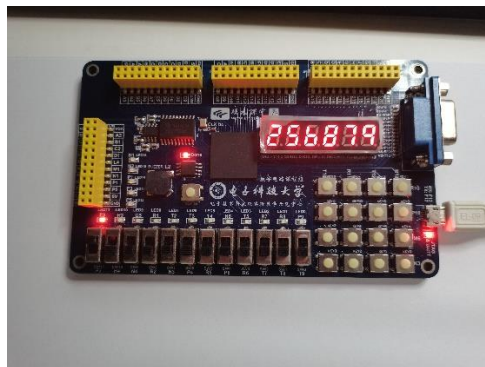


Figure 4

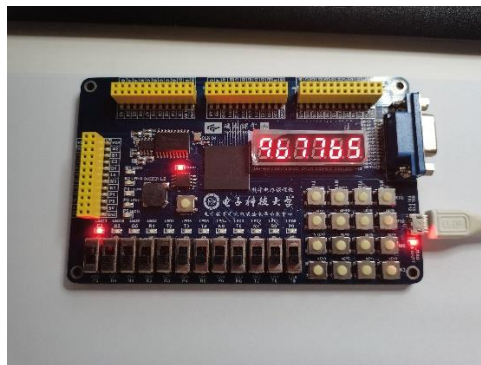


Figure 5

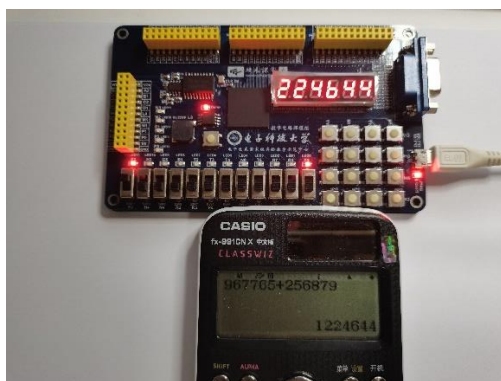


Figure 6

## 2.4.2 游戏功能展示

游戏功能涉及声音、倒计时、LED 闪烁等变化，限制于静态图片展示效果较差，游戏功能部分仅在视频中展示，这里不做展示。

# 3、模块结构

## 3.1 模块化设计

该计算器涵盖了大量逻辑器件，组合逻辑的如比较器、编码器、译码器、多路复用器等，时序逻辑的如计数器、移位寄存器等。各个模块之间的信息转递、信息的处理等带来了工程的复杂性，因此必须严格采用模块化的思想，自顶向下地设计，把系统划分成几个功能模块，每个功能模块再划分成下一层的子模块。在 Vivado 中，每个模块对应一个 module，一个 module 设计成一个 Verilog 程序文件，顶层模块分别调用各个功能块，即实例化的过程。

## 3.2 主功能模块及其子模块

有三个主功能模块，一个加法器模块和两个游戏模块，两个游戏模块分别实现同一个游戏，只是难度不同，因此内容大致相同。三个主功能模块在顶层模块中实例化并被调用，顶层模块负责开发板上的 FPGA 与其外设的连接以及各个模块信息的传递和接收。由于数码管译码器在所有功能中均会涉及，因此也包装成模块放在顶层模块中，并不作为主功能模块。

```
▼ test_top (test_top.v) (4)
  > MODULE_2_u: MODULE_2 (MODULE_2.v) (5)
  > MODULE_1_u: MODULE_1 (MODULE_1.v) (5)
  > MODULE_Adder_u: MODULE_Adder (MODULE_Adder.v) (2)
  led_disp_u: led_disp (led_disp.v)
```

Figure 7

### 3.3.1 加法器模块 MODULE\_Adder

加法器模块 MODULE\_Adder 下有两个子模块 KeyIn 和 add10，KeyIn 负责处理矩阵键盘的输出信号，而 add10 实现两个 6 位十进制数相加的功能。两个模块 KeyIn 和 add10 在其上层模块中建立了联系，KeyIn 输出的两个加数的数据信息作为 add10 模块的输入。

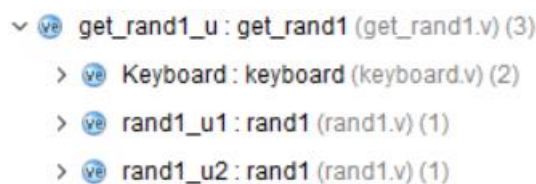
```
▼ MODULE_Adder_u: MODULE_Adder (MODULE_Adder.v) (2)
  > KeyIn_u: KeyIn (KeyIn.v) (5)
  > add10_u: add10 (add10.v) (3)
  led_disp_u: led_disp (led_disp.v)
```

Figure 8

### 3.3.2 游戏模块 MODULE\_1

游戏模块 MODULE\_1 下有 5 个子模块，分别为随机数获取模块 (get\_rand1)、比较判断模块 (Judge2)、加法器模块 (add10)、计时器模块 (Timer) 以及蜂鸣闪烁模块 (Buzzer\_shine)。

随机数获取模块实现在用户的控制下分别输出两个随机数，随机数的随机性实际上与用户控制之间具有紧密的联系，因为这里的随机数由不断流动计数的计数器截取得到，用户在两个不同的时间发出截取指令，就获得了两个可能相同也可能不同的随机数。由于涉及到用户控制，也就涉及到矩阵键盘问题，因此其子模块中还包



```
▼ get_rand1_u : get_rand1 (get_rand1.v) (3)
  > Keyboard : keyboard (keyboard.v) (2)
  > rand1_u1 : rand1 (rand1.v) (1)
  > rand1_u2 : rand1 (rand1.v) (1)
```

Figure 9

在判断比较模块中，用户键入的和被接收，并与实际的和相比较，反馈出相等或是不相等的信息。由于涉及到接收用户的键入，因此其子模块还涉及到矩阵键盘模块。



```
▼ Judge2_u : Judge2 (Judge2.v) (3)
  KEYNUM_u : KEYNUM (KEYNUM.v)
  Compare2_u : Compare2 (Compare2.v)
  > keyboard_u : keyboard (keyboard.v) (2)
```

Figure 10

加法模块，该模块在前面提及的计算器模块中同样被使用，功能也相同，即实现两个 6 位十进制数相加的功能。这里的两个加数



即随机数获取模块传出的两个随机数，相加的结果将传入判断比较模块中作为比较的标准。



Figure 11

计时器模块在游戏中的倒计时环节发挥作用，根据其计时，可以判断用户是否已经超时，也可以作为蜂鸣闪烁模块的输入，调控蜂鸣和闪烁的频率。

蜂鸣闪烁模块负责控制所谓“定时炸弹”的外部表现，控制LED是否闪烁，如何闪烁以及蜂鸣器是否发出声响，发出怎样的声响。针对蜂鸣器发出的不同声响，几个蜂鸣器模块也被设计，在该模块种调用，当需要发出某种声响的时候，只需要对其使能即可。



Figure 12

### 3.3.3 游戏模块 MODULE\_2

游戏模块 MODULE\_2 和 MODULE\_1 大体相同，只是由于是两位十进制数的加法，涉及到的某些数据线位宽、移位寄存器位数等与 MODULE\_1 中的相比有所不同。

## 4、实现方法

前面从整体介绍了三个主要功能模块的功能以及它们所包含的子模块，可以看出，底层模块传出信号在顶层模块中被进行一系列处理，而且各个底层模块也在顶层模块中建立联系，逻辑功能也就相应地产生了。那么，这些逻辑功能具体是怎么实现的呢？

本节着重介绍两个重要的功能模块，键盘模块和加法模块，这两个模块在工程中多次被调用。键盘模块检测用户对矩阵键盘的操作并提供相应的输出，基本的输出如矩阵键盘被按压的位置。加法模块实现任意两个 6 位十进制的加法，其内部实际上采用 BCD 码进行运算，也包含一些具有特定功能的子模块。

### 4.1 键盘模块

#### 4.1.1 输入与输出

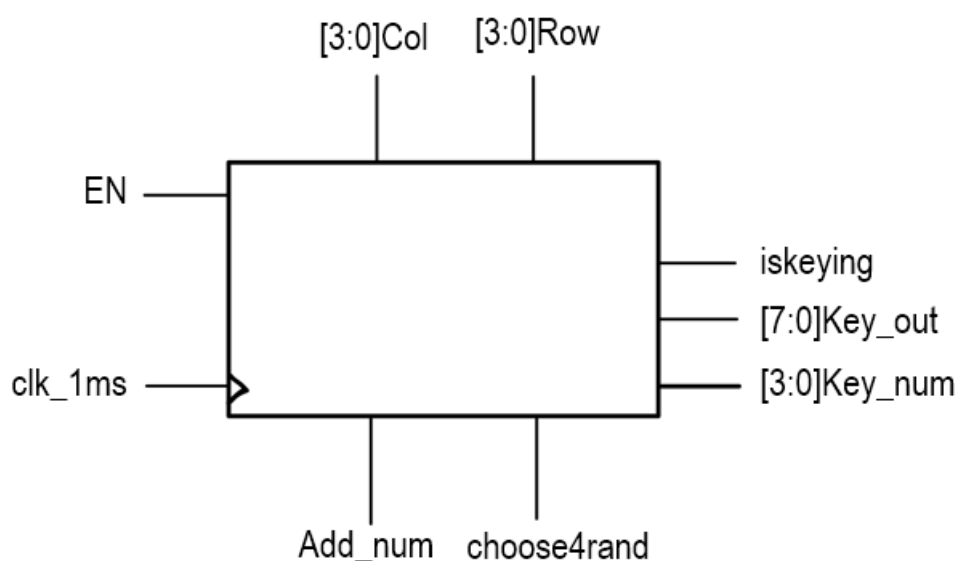


Figure 13

(1) [3:0]Col 和 [3:0]Row 是分别代表矩阵键盘列和行的电平，涉及到矩阵键盘的工作原理，这里不做详细讨论；

[3:0]Row	[3:0]Col
0111	1111
1011	1111 (1101)
1101	1111
1110	1111

Figure 14

(2) EN：键盘使能。用户在操作键盘的过程中，不同阶段键盘可能工作或不工作，工作时的功能也可能不同（对应的上层模块不同），这就意味着不同的键盘子模块需要有使能端，在合适的情况下进行使能。要注意，在不使能的状态下，键盘模块中寄存器所存的值应该都还原为初始值，以便下一次调用不会出现不期望的错误。

(3) Iskeying：用户是否正在键入。在加法功能中，用户正在输入加数或被加数时，键盘和数码管连通，数码管显示键入的内容或显示退格，当用户输入完成，即按下“+”键或者“=”键时，用户停止输入，此时数码管要么不显示，要么显示相加结果。所以，在功能实现的过程中，“用户是否在输入”这个信息比较重要，因此把它作为键盘模块的逻辑输出对于上层而言是有用的。

(4) Key\_out：用户输出的键位。它的位宽为 8 位，由用户按下的[3:0]Col 和 [3:0]Row 组成，当用户按下后松开，它仍能保持上一次所按，知道用户再一次按下的瞬间。显而易见，它的作用是

很重要的，比如，数码管显示需要知道它的输出，随机数获取时的控制信号也来源于它的输出。

(5) Add\_num: 用户按下加号的次数。这个输出用在加法器模块中。加法模块中用户输入的两个加数分别存于两个寄存器，用户输入一串数字，这些数字存于哪个寄存器就要根据用户按下加法的次数来判断，这就是设计此输出的原因。

(6) Choose4rand: 随机数发生器的选择。这个输出用在随机数发生模块中。根据游戏模块随机产生的过程，两个随机数分别在用户第一次和第二次按相应键的时候产生，这就涉及随机数发生器的选择，用户第一次按的时候选择第一个随机数发生器，第二次按的时候选择第二个随机数发生器，于是用该输出来表示随机数发生器的选择。

### 4.1.2 实现方法

逻辑器件的组合可以实现一些较为复杂的功能，下面就一些键盘模块中功能的实现方法展开讨论

#### (1) 输出按下的键位

由矩阵键盘的原理可知，在行 Row 不断扫描的时候，如果出现不全为高电平的列 Col，说明有键被按下，所按下键的位置可以由此时的各行列电压确定。

```

if(EN == 1) begin
    if(Col != 4'b1111)begin    //在键盘使能的情况下，如果有行扫描到有键按下，改变输出键位
        Key_out <= Key;
    end
    else Key_out <= Key_out;    //保持输出键位
end

```

Figure 15

## (2) 截取按键按下的瞬间

截取按键按下的瞬间很重要，比如我们可以通过它来获得按键按下的次数。首先，如果连续 4 个时钟，行扫描下，列都没有出现 0，说明没有按下，反之则说明有按下，于是可以考虑用 4bit 的移位寄存器来存 4 个时钟的“IN”，这里的 IN 定义为各个列电压的乘积

```

wire IN;
assign IN = Col[3]*Col[2]*Col[1]*Col[0];    //当前行扫描下是否有列电压为0
wire [3:0]Q;    //存储连续四个行扫描的IN
reg_4bit reg_4bit_u(
    .clk(clk_1ms),
    .IN(IN),
    .Q(Q)
);

```

Figure 16

当 IN 为 1 时，该时钟下列电压四个其中的一个为 0；当列为 0 时，该时钟下列电压四个全为 1。IN 作为移位寄存器的输入，在寄存器中不断移位，于是就可以判断是否由键按下，用 ispress 表示。

```

if(Q == 4'b1111) ispress <= 0;    //如果当前列电压全为1，说明没有按
else ispress <= 1;    //否则有按

```

Figure 17

已经知道了是否有键被按下，要想截取按下瞬间，就涉及 ispress 从 1 变为 0 或者从 0 变为 1，所以还需要一个 2bit 的移位寄存器，其状态用大写的 ISPRESS 表示

```
reg ispress;    //根据Q判断按键是否按下，按键按下为1，没有按下为0
wire [1:0] ISPRESS;
reg_2bit reg_2bit_u(
    .clk(clk_1ms),
    .IN(ispress),
    .Q(ISPRESS)
);
```

Figure 18

当 ISPRESS 为 10 时，表示松开的瞬间；当 ISPRESS 为 01 时，表示按下的瞬间

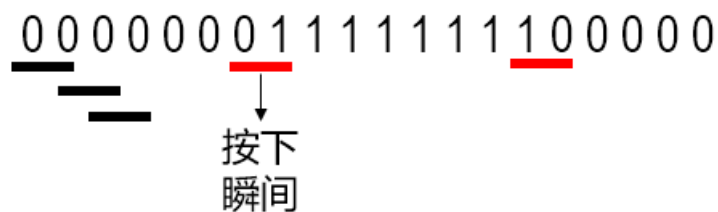


Figure 19

于是我们就截取到了按键按下的瞬间。很多操作都得以在按键按下的瞬间进行

```
if(ISPRESS==2'b01) begin    //按键按下的瞬间
```

Figure 20

### (3) DEL 操作的实现

DEL 操作是键盘模块和数码管相互配合实现的。前面介绍过键盘模块的输出 Key\_num 表示数码管上“应该”出现的数字个数，Key\_out 表示用户按下的键位。当用户按数字 0~9 数字时，

Key\_num+1；当用户按下 DEL 时，Key\_num-1。根据 Key\_num 和 Key\_out，如果按下键位是数字，Key\_num 对应的数码管显示该数字；如果按下的键位是“DEL”，Key\_num+1 对应的数码管不显示

## 4.2 加法器模块

### 4.2.1 输入与输出



Figure 21

(1) [23:0]NUMBER1：表示 6 位十进制加数。采用 BCD 编码，所以每个十进制数字需要 4bit，一共 24bit。

(2) [23:0]NUMBER2：表示 6 位十进制被加数。

(3) S ：输出 6 位十进制和。

(4) Hi：表示最高位进位。

### 4.2.2 功能实现

该加法器是时序逻辑，进位输入输出采用反馈的形式。用 BCD 码进行运算。即将十进制数用 BCD 码编码，模块内部进行 4bit 二进制的加法。

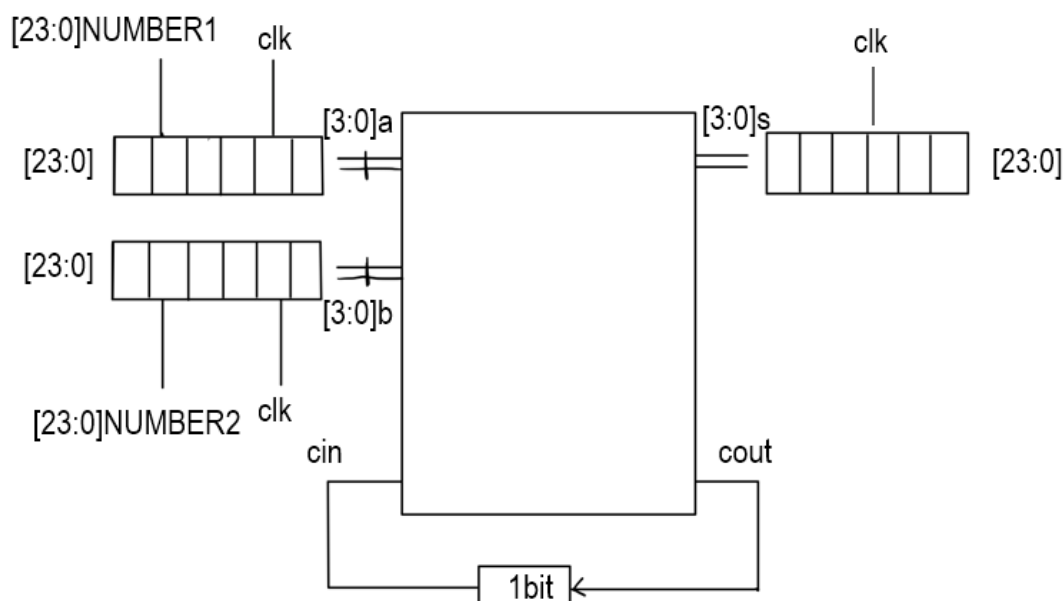


Figure 22

先将加数和被加数存入寄存器，每个时钟寄存器移出一个 4bit 的二进制数，模块内部进行二进制数的加法，当两个二进制数相加小于十进制 10 时，结果直接输入用于存放结果的移位寄存器，进位 0 存入一个 1bit 的寄存器中；当两个二进制数相加大于或等于十进制 10 时，结果减去十进制 10 后存入移位寄存器，并将进位 1 存入 1bit 寄存器。1bit 寄存器的作用是将前一位相加后的进位输出作为后一位相加时的进位输入。代码描述如下



```

wire [3:0] s;
reg [4:0] sum;
assign s = sum[3:0];
reg [3:0] count = 4'b0000; assign test = count;
always @ (posedge clk) begin
    if(count != 4'b1111) count <= count + 1;
    else begin
        count <= 4'b0000;
    end

    if(count == 4'b0000) begin
        cin <= 0;
        cout <= 0;
    end

    if(count == 4'b0111) begin
        hi <= cin;
    end
    if(a+b+cin>=5'd10)begin
        cout = 1;
        sum = a+b+cin-5'd10;
        cin = cout;
    end
    else begin
        cout = 0;
        sum = a + b + cin;
        cin = cout;
    end
end
end

```