构建四位加法器

不知从何时起，北航计算机学院 15 级学生高小明同学迷上了计算机硬件的研究。最近他正在探索加法器的设计，在理解了一位加法器的原理之后，他尝试着用 Verilog HDL 设计**四位加法器**.....

提交要求

使用 Verilog 搭建一个四位加法器并提交。具体模块端口定义如下：

信号名	方向	描述
A[3:0]	I	第一个加数
B[3:0]	I	第二个加数
Clk	I	时钟信号
En	I	更新信号
Sum[3:0]	O	两数相加之和
Overflow	O	溢出标志位

模块功能定义如下：

序号	功能名称	功能描述
1	加运算	A 和 B 进行无符号加运算，当需要更新时将结果赋值给 Sum，溢出赋值给 Overflow

注：本题与视频的区别在于视频中是在需要时更新输入，本题要求更新输出。

- 输入： A（4bit）、B（4bit）、Clk（1bit）、En（1bit）
- 输出： Sum(4bit)、 Overflow（1bit）
- **在时钟信号上升沿且 En 为 1 时更新 Sum 和 Overflow，初始的输出设置为 0**
- 加法器做的是无符号运算
- 测试保证时钟上升沿到来时前后短时间内输入值保持恒定不变

- 文件内模块名: adder

首次提交 TIPS

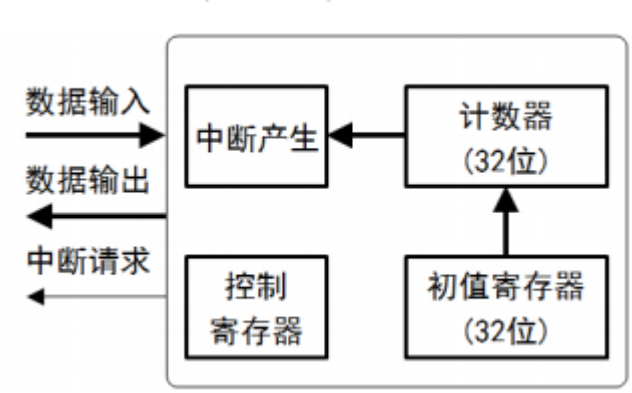
提交 Verilog 程序时一定要注意模块名还有端口名称需与题目要求一致，以及提交相应的.v 文件。例如，此题的要求是文件内模块名为 adder，端口名称在表格中已给出，也就是说我们需要保证端口信息是这样的（顺序无所谓）：

```
module adder(
    input [3:0] A,
    input [3:0] B,
    input Clk,
    input En,
    output [3:0] Sum,
    output Overflow
    );
```

如果端口信息出现错误，评测机就会返回 compile error 的信息。所以在之后的提交中，大家请务必关注端口设置的问题。

计数器

在 P7 中，我们需要设计一个非常重要的部件：计数器。它是定时产生中断信号从而使系统产生一系列既定动作的关键。在这里，我们可以先初步了解一下计数器的内部结构：



这里做一些简要的解释：控制寄存器可以被输入的数据改变其内容，从而实现对计数器的 RESET 操作和 ENABLE 操作。初值寄存器决定了计数器从多大的数开始"倒数"，而计数器在 ENABLE 的前提下会随着时钟周期的推进不断地减小，直到减为 0 后会对外界产生一个中断信号，表示"现在我完成了你让我进行的倒数工作"，而系统就会根据这个信号结合自身当前的状态决定是否要进行别的操作。

计数器的实现还需要规定大量的细节，不过这里我们就不多做解释了。

现在，在对 Verilog 和 ISE 有了初步的了解之后，我们来尝试设计一个经过了大量简化处理的计数器，它只有一些非常简单的功能，端口定义如下：

信号名	方向	描述
Clk	I	时钟信号
Reset	I	复位信号
Slt	I	选择信号
En	I	使能信号
Output0[63:0]	O	计数器 0 当前值
Output1[63:0]	O	计数器 1 当前值

我们要实现的功能非常简单：

1、 在任意一个时钟上升沿到来的时候，如果复位信号有效，则将两个计数器同时清零；

2、 每个时钟上升沿到来的时候，如果使能信号有效，则称其为一个“有效时钟周期”，如果：

a) 选择信号为 0，则将这个有效时钟信号计入计数器 0，每经过 1 个属于计数器 0 的有效时钟周期，计数器 0 累加 1；

b) 选择信号为 1，则将这个有效时钟信号计入计数器 1，每经过 4 个属于计数器 1 的有效时钟周期，计数器 1 累加 1；

3、 在满足 1 时，即使 2 的条件满足，也不必执行 2；

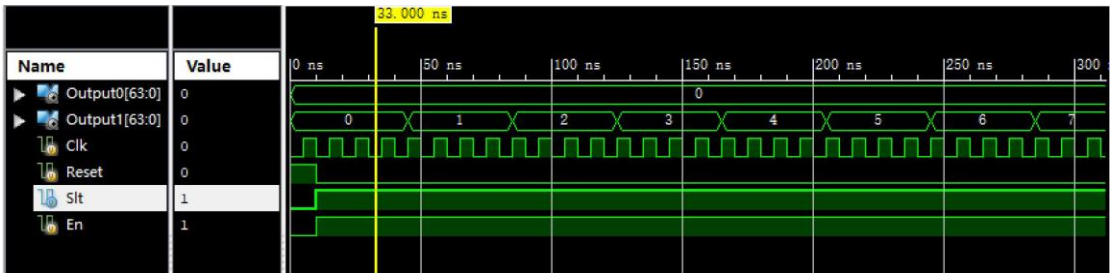
4、 注意复位操作也会复位每个计数器当前的有效时钟周期。

5、 请提交相应的.v 文件

6、 文件内模块名：code

注意初始时每个计数器的值都为 0。

参考波形如
下：



Verilog 流水线电路

在运算电路的设计中，我们通常用流水线的技术来提高元件的运算效率。所谓流水线的技术，就是把组合电路分为若干个顺序执行的阶段，在这些阶段结尾放置寄存器，时钟上升沿到来时，寄存器存储这一阶段的运算结果，并把这一阶段的输出作为下一阶段的输入。

比如用组合电路运算 a*c+b*d，且假设一次”1->0->1”为一个时钟周期，我们可以把运算分为两个阶段，用两个时钟周期来完成。

我们可以在第一个时钟周期运算 a*c 和 b*d，运算完成时时钟从 0 变成 1，它们的结果就存到寄存器里；第二个时钟周期利用第一个时钟周期存在寄存器里的结果运算 a*c+b*d。衔接两个运算阶段的寄存器叫做流水线寄存器。

下面运用了流水线技术来实现 8 个数的加法，我们来从这个例子理解流水线如何工作。

```
21  module eight_digi_adder(
22      input clk,
23      input [31:0] digi1,
24      input [31:0] digi2,
25      input [31:0] digi3,
26      input [31:0] digi4,
27      input [31:0] digi5,
28      input [31:0] digi6,
29      input [31:0] digi7,
30      input [31:0] digi8,
31      output reg [31:0] result = 0
32      );
33      reg [31:0] step1 [4:1]; //第一级流水线寄存器
34      reg [31:0] step2 [2:1]; //第二级流水线寄存器，result可以视为第三级流水线寄存器。
35
36      integer i;
37
38      initial begin
39         for (i = 1;i <= 4;i = i + 1) step1[i] = 0;
40         for (i = 1;i <= 2;i = i + 1) step2[i] = 0;
41      end
42
43      always @(posedge clk) begin
44        step1[1] <= digi1 + digi2;
45        step1[2] <= digi3 + digi4;
46        step1[3] <= digi5 + digi6;
47        step1[4] <= digi7 + digi8;
48
49        step2[1] <= step1[1]+step1[2];
50        step2[2] <= step1[3]+step1[4];
51
52        result <= step2[1] + step2[2];
53      end
54
55  endmodule
56
```

为了理解流水线内部的工作，我们用表格的形式描述流水线内部寄存器的值的变化过程。

第n个时钟上升沿	输入	step1[1]	step1[2]	step1[3]	step1[4]	step2[1]	step2[2]	result
n=1	8个1	2=1+1	2=1+1	2=1+1	2=1+1	x	x	x
n=2	8个2	4=2+2	4=2+2	4=2+2	4=2+2	4=2+2	4=2+2	x
n=3	8个3	6=3+3	6=3+3	6=3+3	6=3+3	8=4+4	8=4+4	8=4+4
n=4	x	x	x	x	x	12=6+6	12=6+6	16=8+8
n=5	x	x	x	x	x	x	x	24=12+12

可以看出，在第一个上升沿，流水线加法器的第一级在处理 n=1 时的输入；它的四个 step1 寄存器储存第一级的结果。

第二个上升沿时，第一级处理 n=2 时的输入；同时，上一个上升沿产生的结果通过 step1 寄存器传递到 step2 并进行运算。

第三个上升沿时，第一级处理 n=3 时的输入；同时来自 n=1 的输入的处理从 step2 传到 step3，来自 n=2 的输入的处理从 step1 传到 step2。

最终，第五个上升沿，n=3 的输入也传到了 step3。

可以看出，流水线技术把 8 个数加法的复杂运算分散为 3 个部分，同时各个部分相对独立，因而一个流水线可以同时处理多个输入的运算。例如流水线加法器就可以同时处理 3 个输入，它在 n=3 时达到满载。

理解并学会应用流水线对于后续的课程是十分有用的。例如我们将在 P5 中用流水线技术实现一个 MIPS CPU。

接下来，请你实现一个简单的流水线部件——流水线二维向量数量积单元。

a. 模块规格

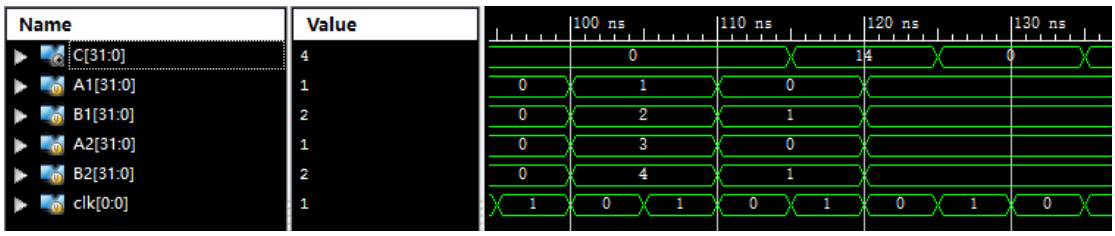
模块名：pipeline

名称	功能	位宽
A1	输入向量A的第一个无符号分量	32
A2	输入向量A的第二个无符号分量	32
B1	输入向量B的第一个无符号分量	32
B2	输入向量B的第二个无符号分量	32
clk	输入时钟信号	1
C	输出A1*B1+A2*B2	32

b. 功能描述

构建一个两级的流水线。其中第一级的流水线寄存器分别存储 A1*B1 和 A2*B2，第二级的流水线寄存器存储 A1*B1+A2*B2。因此你需要在第 n+1 个时钟上升沿输出第 n 个时钟上升沿输入的结果。另外，请把 C 定义为 reg 类型，并且它的初始值规定为 0(`output reg [31:0] C = 0;`)。

波形示例：



测试方式：

我们会用如下风格的 testbench 进行测试，并且请大家提交相应.v 文件。

```
46      //......
47      initial begin
48          // Initialize Inputs
49          A1 = 0;
50          B1 = 0;
51          A2 = 0;
52          B2 = 0;
53          clk = 0;
54
55          // Wait 100 ns for global reset to finish
56          #100;
57
58          A1 = 0;
59          B1 = 1;
60          A2 = 2;
61          B2 = 3;
62          #10;
63          A1 = 3;
64          B1 = 2;
65          A2 = 1;
66          B2 = 0;
67          #10;
68
69      end
```

Verilog 字符自动机

许多人在网上注册账号时，会使用"Jack1996","toad301"这样的用户名。为了分析取这一类“大小写字母+数字”为 ID 的用户的特征，我们需要准确识别这样的字符串。下面请你用 Verilog HDL 编写一个能识别符合这个模式的字符串的电路。我们简称这类字符串为“ID”。ID 即所有符合"大小写字母+数字"(严格来说是符合正则表达式**^[a-zA-Z]+[0-9]+$**)的字符串组成的集合。

a.模块规格：

名称	功能简述	位宽
char	接受串行输入的字符。	8
clk	接受时钟信号。	1
out	状态机输出，详见功能描述。	1

模块名：id_fsm。

b.功能描述：

每个 clk 上升沿到来的瞬间，从 char 读入一个字符 c。注意，这里的 c 是用 ASCII 编码的，也就是说 char 这个 8 位寄存器的值刚好就是 c 的 ASCII 码的 8 位二进制。

设已经读入的字符串为 S，换句话说，这个 S 是由每个 clk 上升沿时读入的 c 拼接而成的。我们需要你在每个 clk 的上升沿完成如下判断：

设读入前的字符串为 S，则读入新字符 c 后字符串为 Sc。假如 Sc 的一个**后缀**符合本题 ID 的定义，out 置为 1；否则 out 置为 0。

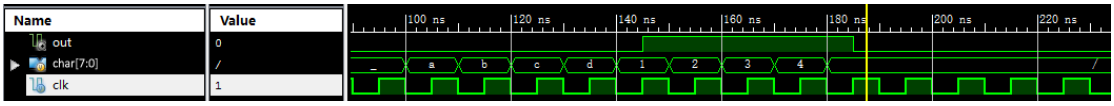
一个字符串 S=c1c2c3...cn 的后缀组成的集合是 P={s=cici+1...cn|1<=i<=n}。

比如已读入的字符串为 **01010abc123**，它的一个后缀 **abc123** 符合 ID 的定义，out 应输出 1；假如是 **01010abc%**，它没有符合要求的后缀（因为所有非空后缀都包含非法符号'%'），因此 out 应输出 0。

波形示例:

输入：char = a b c d 1 2 3 4 /

输出：out = 0 0 0 0 1 1 1 1 0



注意模块名称和端口定义，并且提交相应的.v 文件