开方器

姓名：尤艺霖 学号：2018011324 日期：2020.6.5

我同意授权将本工程代码完全开源。

1. 实验目的

掌握行为描述的乘法器
掌握结构描述的比较器
练习元件例化和状态机的使用，并结合实际算法
练习利用软件仿真对电路的功能进行验证和分析

2. 实验任务

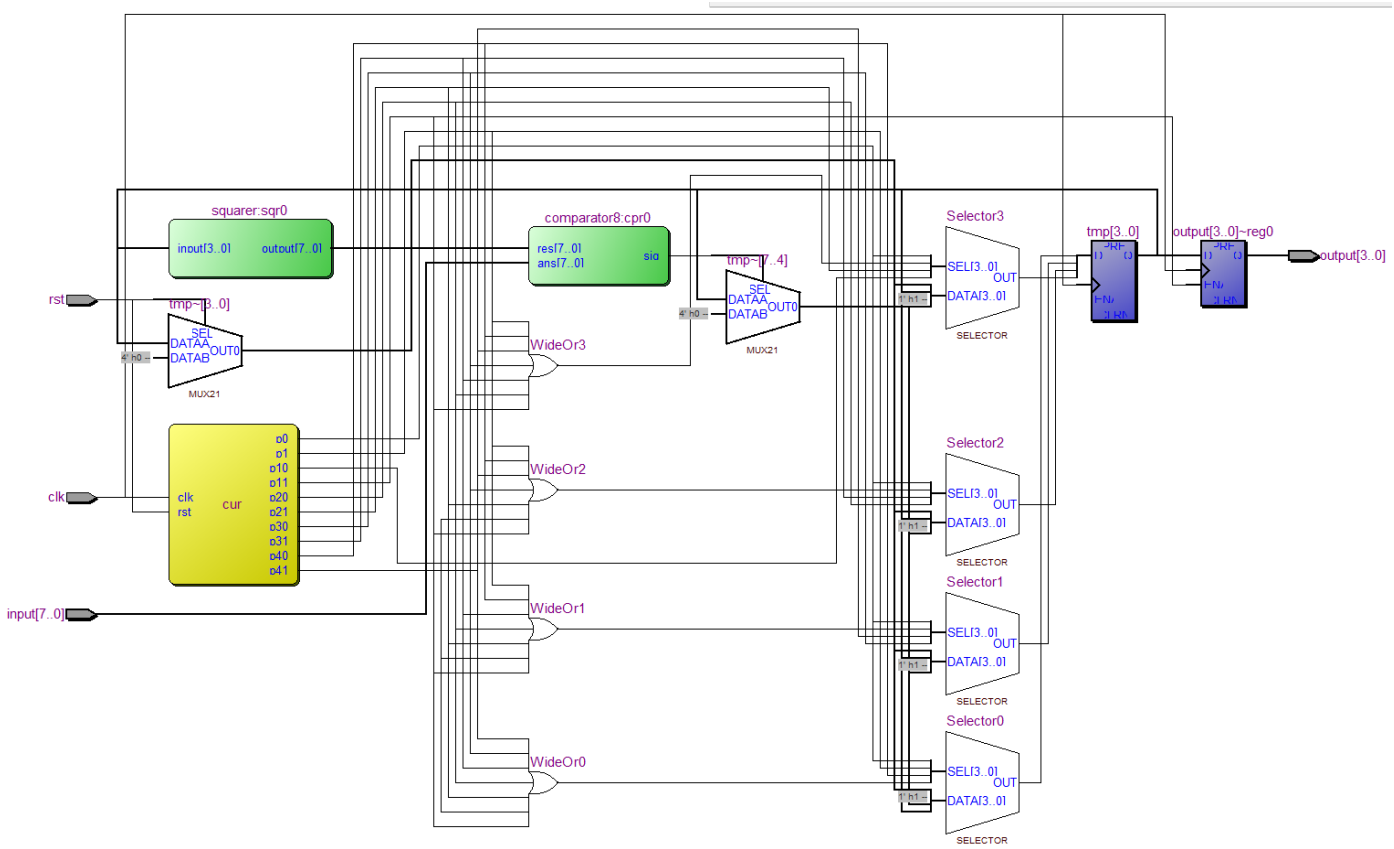
设计了一个基于二分法的开方器，允许输入 8 位的二进制数，输出将其开平方根后下取整的值的二进制形式。并允许 Reset 从而重复使用开方功能。

3. 实验原理

算法上使用二分法。数字在电路中都是二进制表示，适合使用二分法。具体步骤是每次将未处理的最高位置 1，然后判断当前答案的平方是否大于输入，如果是则将最高位置 0，否则保留最高位的 1，然后进入下一位再置 1、计算平方，以此类推直到最低位，即有答案。

使用元件例化的方法，分别实现平方器和比较器，在每次改变二分值得时候直接通过两个元件计算出当前答案对应得平方值是否大于输入，如果大于则将对应位置 0，否则不动，然后进入下一位。对于二分法所带有的时序，采用状态机的设计方法解决。

RTL 视图如下：



其中黄色的为状态机，左边绿色为平方器，右边绿色为比较器，tmp 为暂时存储当前答案的变量。

4. 实现方法

使用 Jielab 上的接口实现。其中 rst 和 clk 两个信号直接有对应的接口，除此之外 Jielab 上总共有 16 个接口，理论上最多可以支持 10 位二进制数的开方，但考虑到 8 位更方便实现比较器等元件，故选择实现 8 位的开方器，因而需要 8 个接口输入，4 个接口输出，总共用到了 12 个接口，以及 rst 和 clk。除了板子以外还需要一块 clock 负责提供时钟信号，三块 4 位 switch 提供输入的按钮和对输入输出的显示，和最后一个 switch 提供对 rst 的控制。

5. 设计过程

整体的时序由状态机控制，每次二分需要两个状态，开始和结束还各需要一个状态，以下是状态说明：

p0：开始状态，当 rst 为 1 时将存储当前答案的临时变量置 0，然后进入 p1

p1：将临时变量第 3 位置 1，然后进入 p40，平方器和比较器会自动计算出当前答案的平方是否大于输入

p40：如果当前答案平方大于输入，则将临时变量第 3 位置 0。同时进入 p41

p41：与 p1 类似，区别在于处理的是第 2 位（这里认为最高位是第 3 位），然后进入 p30

p30：与 p40 类似，区别在于处理的是第 2 位，同时进入 p31

p31：与 p1 类似，区别在于处理的是第 1 位，然后进入 p20

p20：与 p40 类似，区别在于处理的是第 1 位，同时进入 p21

p21：与 p1 类似，区别在于处理的是第 0 位，然后进入 p10

p10：与 p40 类似，区别在于处理的是第 0 位，同时进入 p11

p11：将当前答案赋值给输出。如果 rst 为 0，则回到 p0 等待下一轮计算。

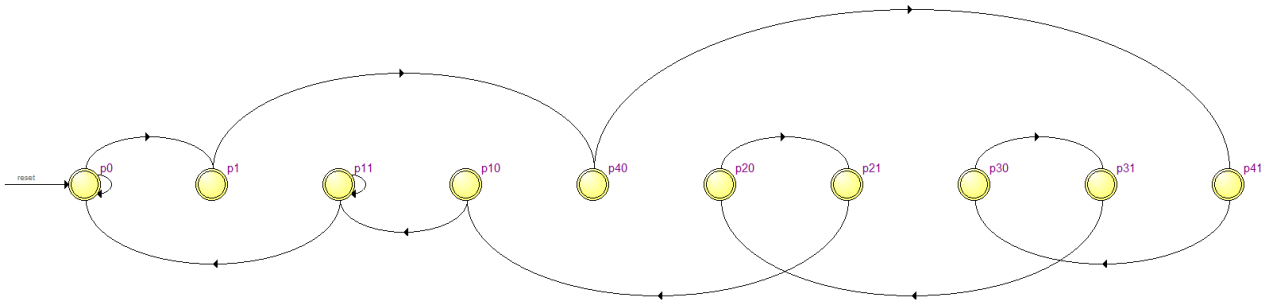
注意在到达终态前改变输入会使结果不正确，但通常使用的是 1MHz 的时钟信号，理论上不会有上述情况。

平方器的具体实现采用了行为描述的方法。依次判断每一位是否是 1，如果是 1 的话则用一个临时变量存下这个数左移对应次数之后的数，最后将所有临时变量加起来，就完成了平方的操作。

八位比较器是两个四位比较器组合而成的。四位比较器按照数电理论课上的式子来实现：对每一位计算出是否大于、是否等于，然后利用这些来判断整体是否大于、是否等于。这里的八位比较器只需要输出是否大于的信号，而不用判断相等，故可以节省一个输出。

同时，这样对 rst 的利用方式也有一定鲁棒性，无论状态的初值如何，只要 rst 为 0 就会迅速到达并停留在 p0。

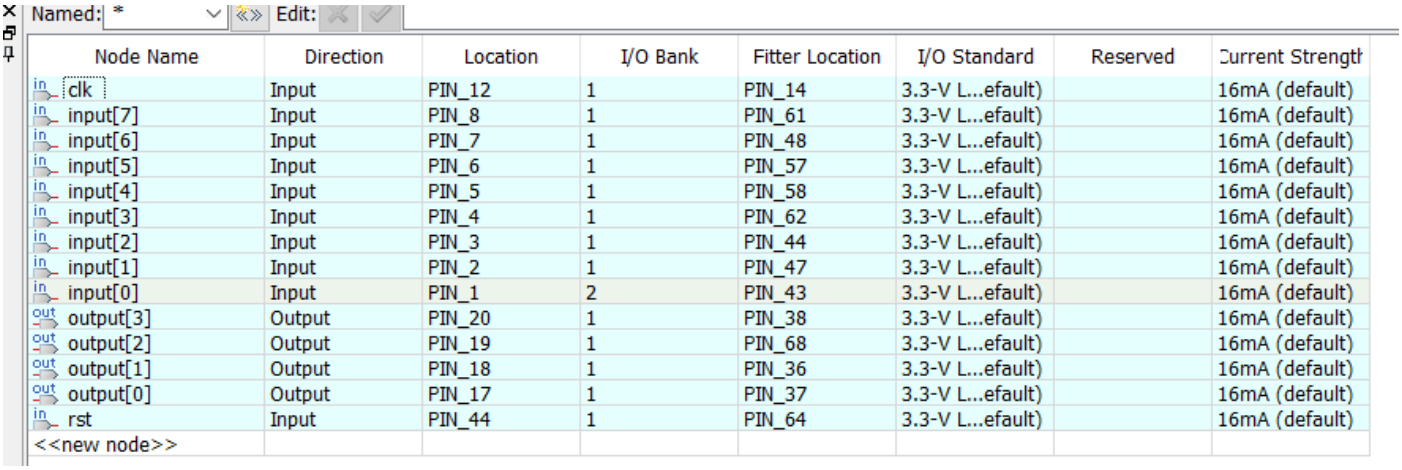
最后附上状态机截图：



6. 实验结果

1） 管脚分配

软件中分配的管脚如下图：



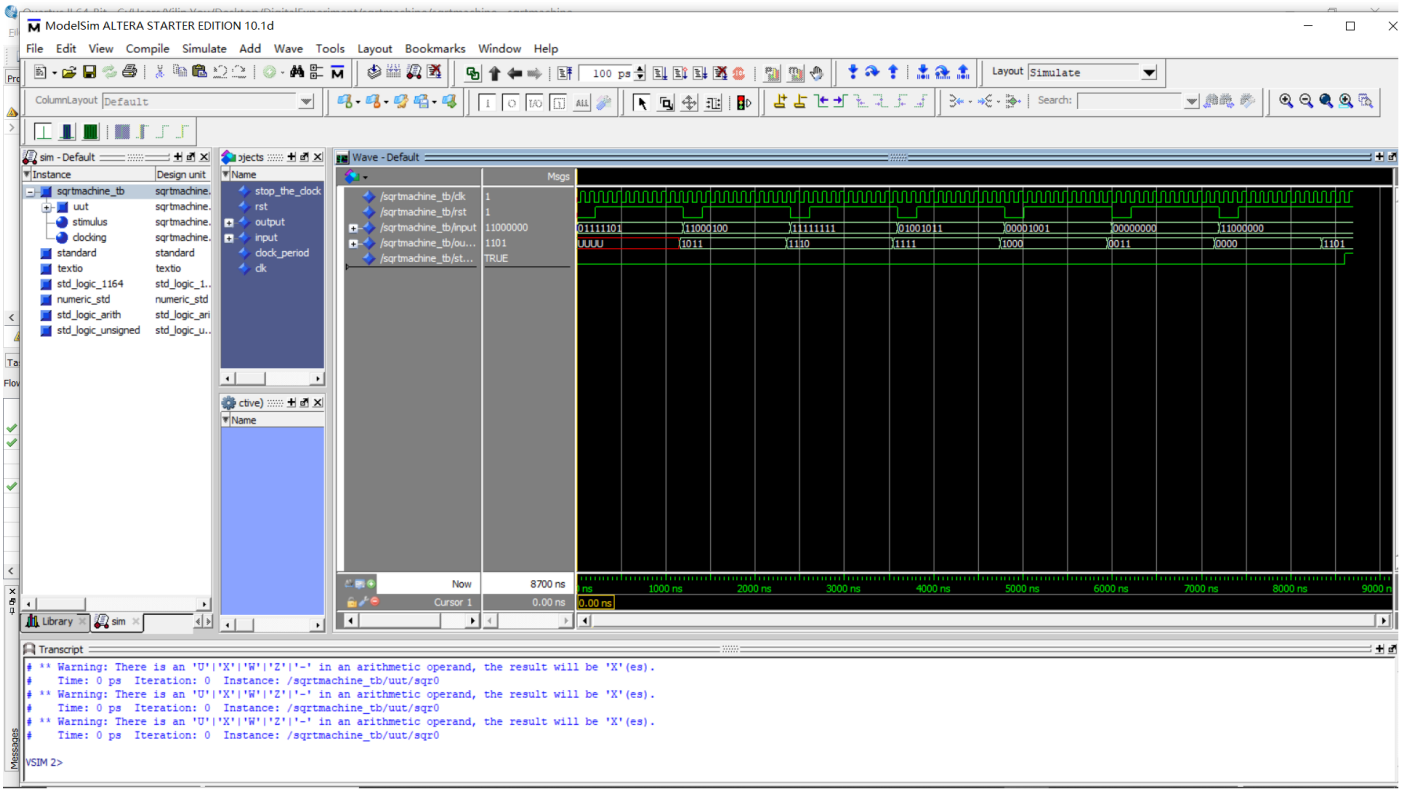
Named: * Edit:

Node Name	Direction	Location	I/O Bank	Fitter Location	I/O Standard	Reserved	Current Strengtl
clk	Input	PIN_12	1	PIN_14	3.3-V L...efault)		16mA (default)
input[7]	Input	PIN_8	1	PIN_61	3.3-V L...efault)		16mA (default)
input[6]	Input	PIN_7	1	PIN_48	3.3-V L...efault)		16mA (default)
input[5]	Input	PIN_6	1	PIN_57	3.3-V L...efault)		16mA (default)
input[4]	Input	PIN_5	1	PIN_58	3.3-V L...efault)		16mA (default)
input[3]	Input	PIN_4	1	PIN_62	3.3-V L...efault)		16mA (default)
input[2]	Input	PIN_3	1	PIN_44	3.3-V L...efault)		16mA (default)
input[1]	Input	PIN_2	1	PIN_47	3.3-V L...efault)		16mA (default)
input[0]	Input	PIN_1	2	PIN_43	3.3-V L...efault)		16mA (default)
output[3]	Output	PIN_20	1	PIN_38	3.3-V L...efault)		16mA (default)
output[2]	Output	PIN_19	1	PIN_68	3.3-V L...efault)		16mA (default)
output[1]	Output	PIN_18	1	PIN_36	3.3-V L...efault)		16mA (default)
output[0]	Output	PIN_17	1	PIN_37	3.3-V L...efault)		16mA (default)
rst	Input	PIN_44	1	PIN_64	3.3-V L...efault)		16mA (default)
<<new node>>							

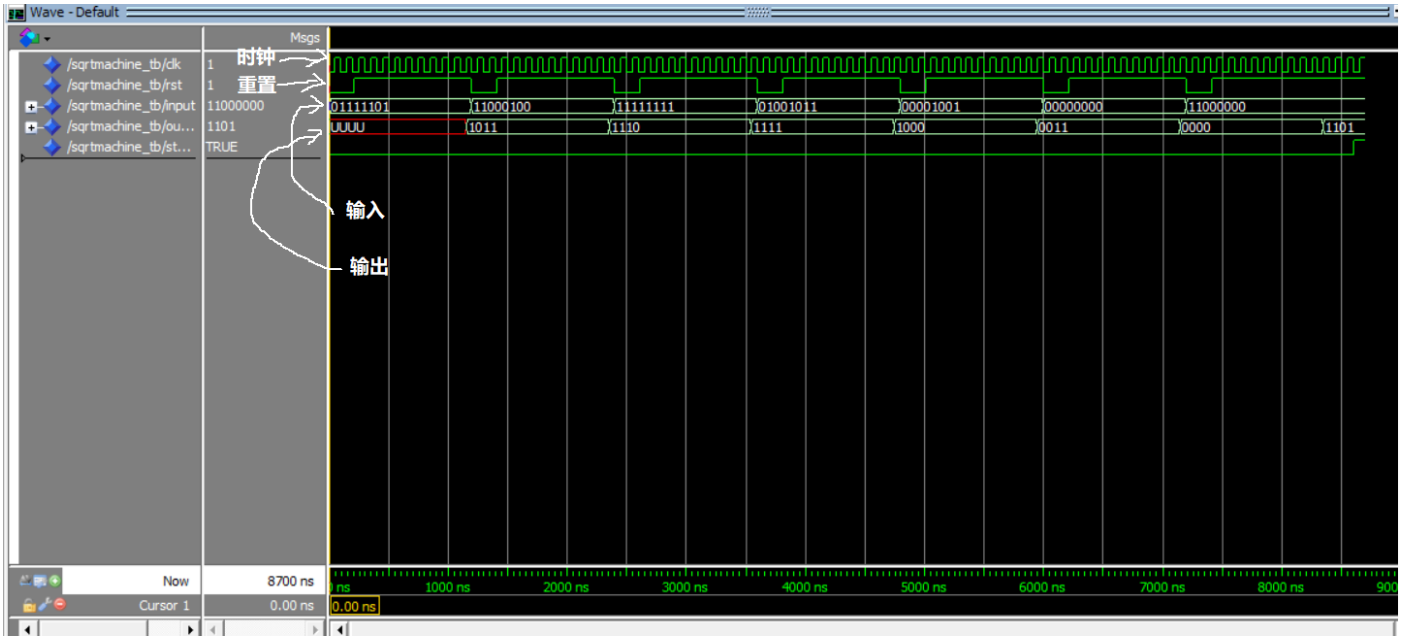
具体解释如下：

信号	管脚	说明
clk	Pin12	按照指示设置时钟信号
rst	Pin44	按照指示设置 rst 信号
input[0]	Pin1	输入的最低位
input[1]	Pin2	输入的第二位
input[2]	Pin3	输入的第三位
input[3]	Pin4	输入的第四位
input[4]	Pin5	输入的第五位
input[5]	Pin6	输入的第六位
input[6]	Pin7	输入的第七位
input[7]	Pin8	输入的最高位
output[0]	Pin17	输出的最低位
output[1]	Pin18	输出的第二位
output[2]	Pin19	输出的第三位
output[3]	Pin20	输出的最高位

2） 测试样例的仿真功能截图



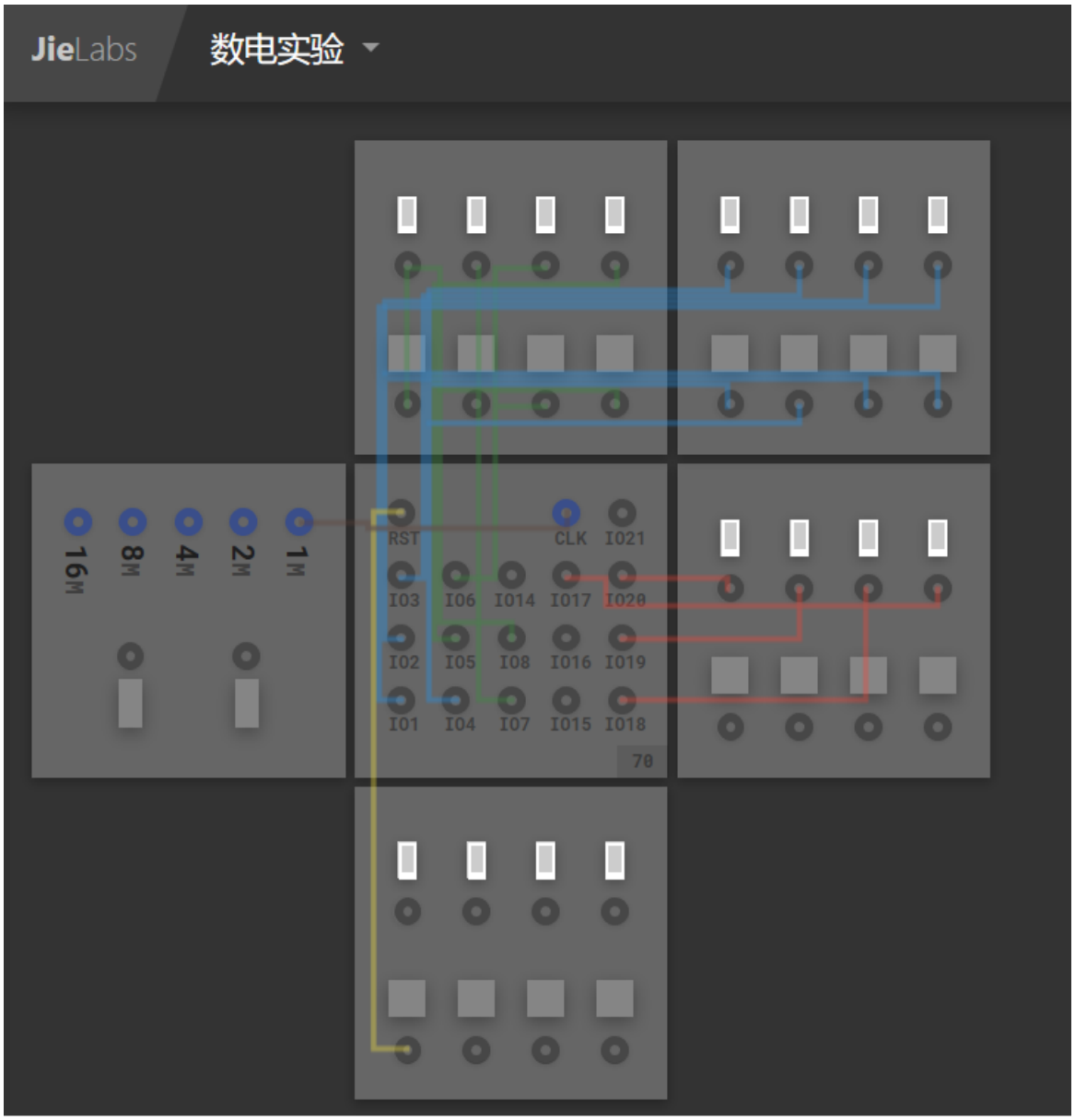
以上为截屏原图，下面为带解释的细节图：



可以看到其中包含了 7 个测例，每个测例都是在输入之后若干个时钟周期之后才产生。
同时这张图也可以证明 rst 功能是完整的。
七个测例分别为：
floor(sqrt(125))=11
floor(sqrt(196))=14
floor(sqrt(255))=15
floor(sqrt(75))=8
floor(sqrt(9))=3
floor(sqrt(0))=0
floor(sqrt(192)=13

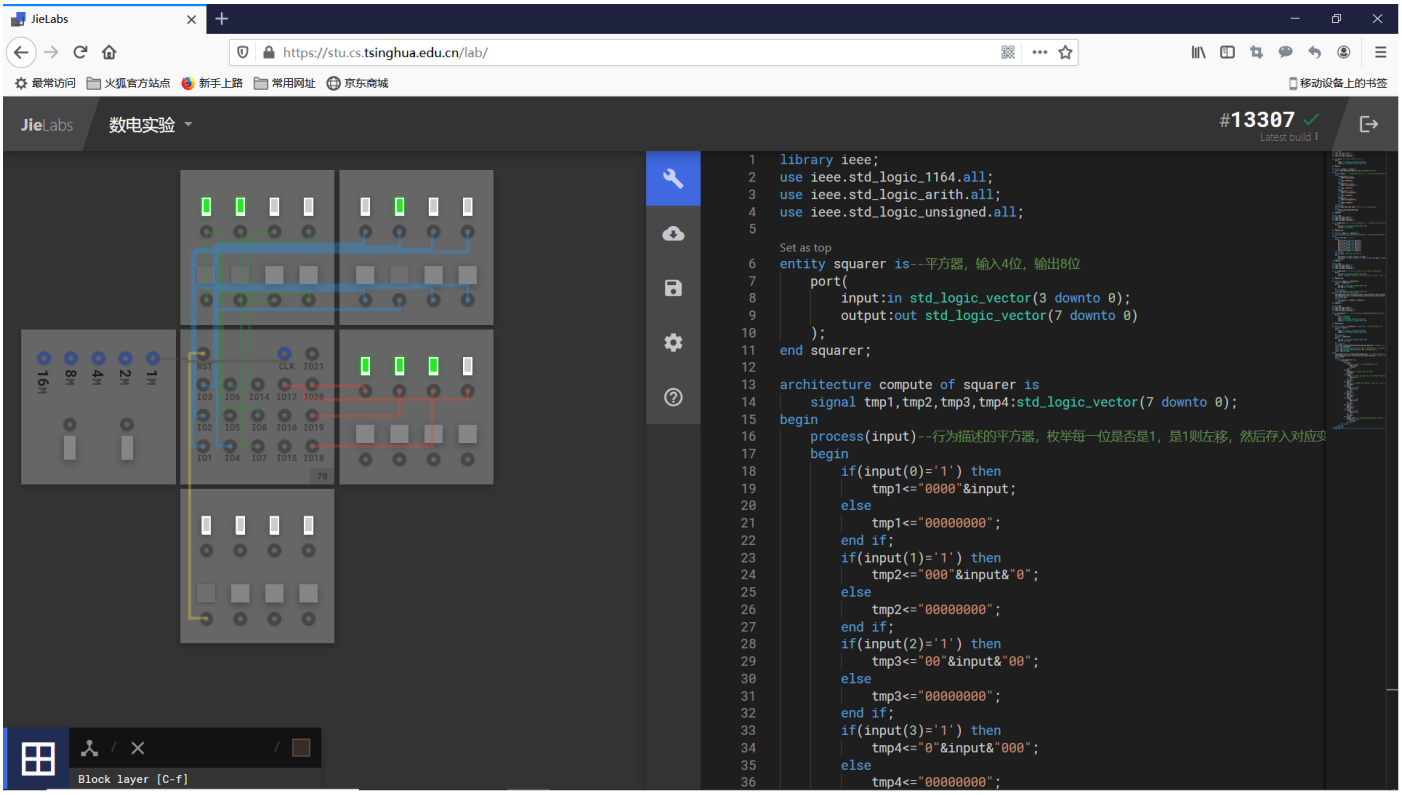
3） 实验操作

首先将代码复制到 Jielab 中，然后按照上面的管脚分配表分配管脚，按照如下方式连线：

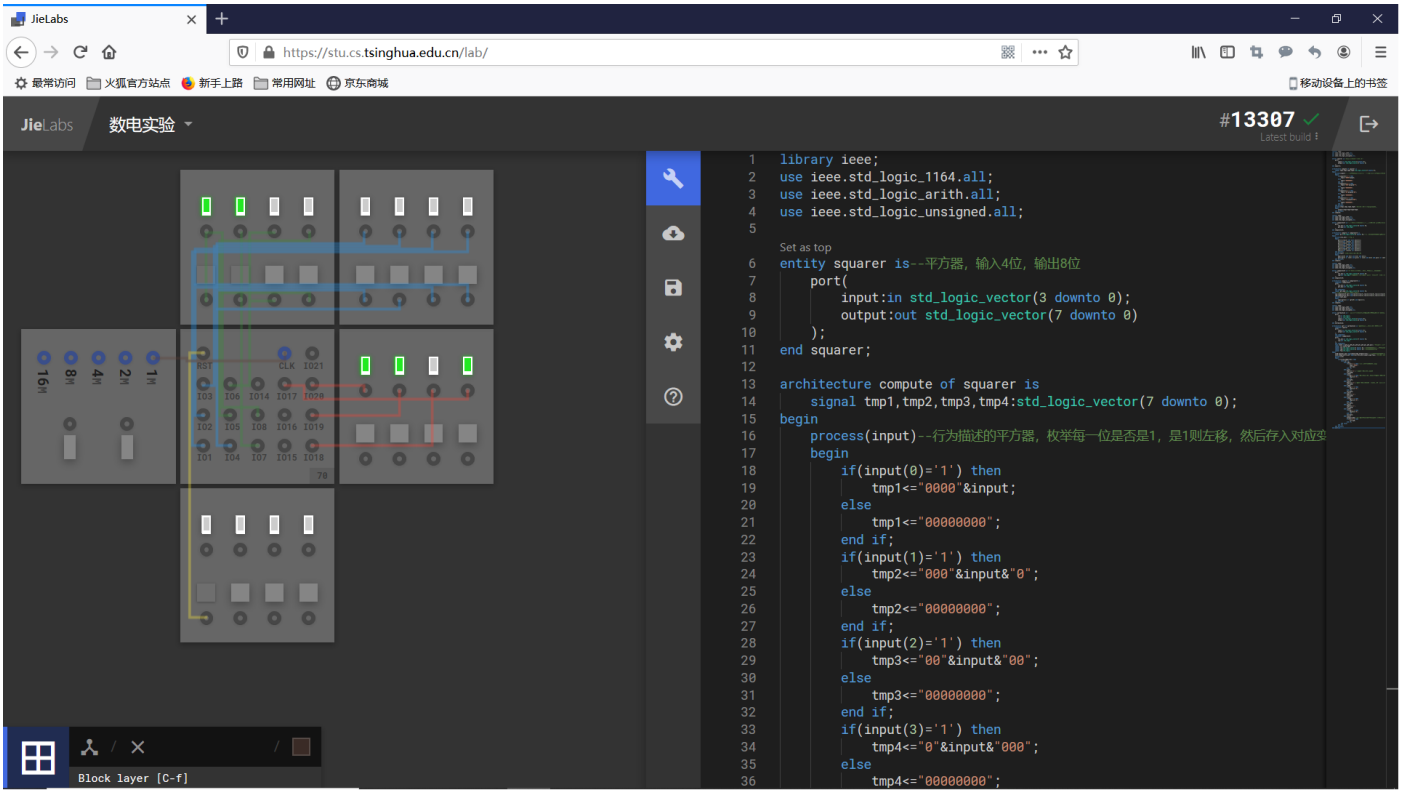


说明：
棕色连线连接了时钟信号
黄色连线连接了 rst 信号
红色连线连接了输出信号
蓝色连线连接了低四位输入信号
绿色连线连接了高四位输入信号
其中蓝绿色连线还连上了灯泡，用来让输入更显眼

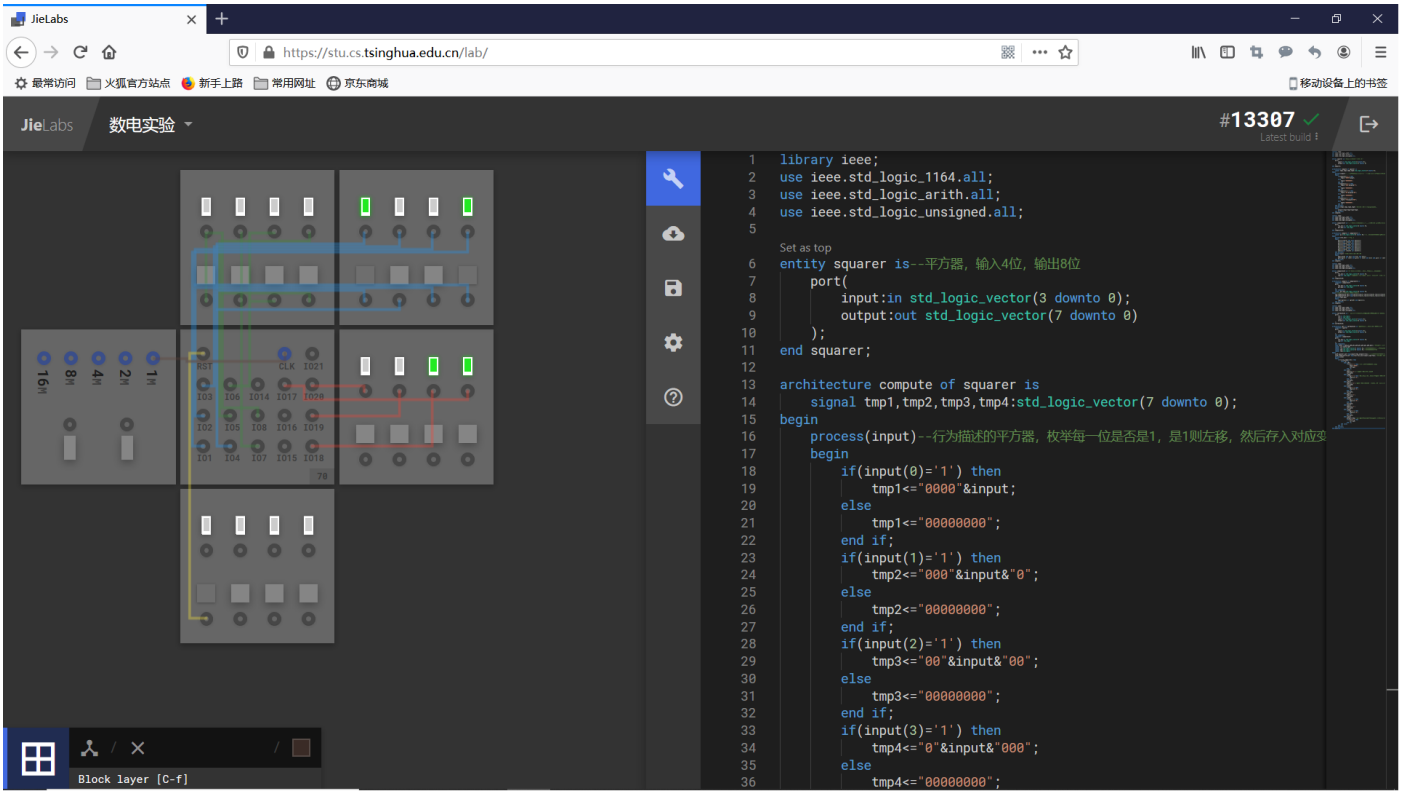
使用时先将底下的黄色连线按钮置 0
然后在输入部分按下按钮安排输入的数
然后将黄色按钮置 1，即可在输出部分看到输出的数的二进制形式
然后将 rst 置 0 即可开始新一轮测试
几个样例放在接下来的几页：



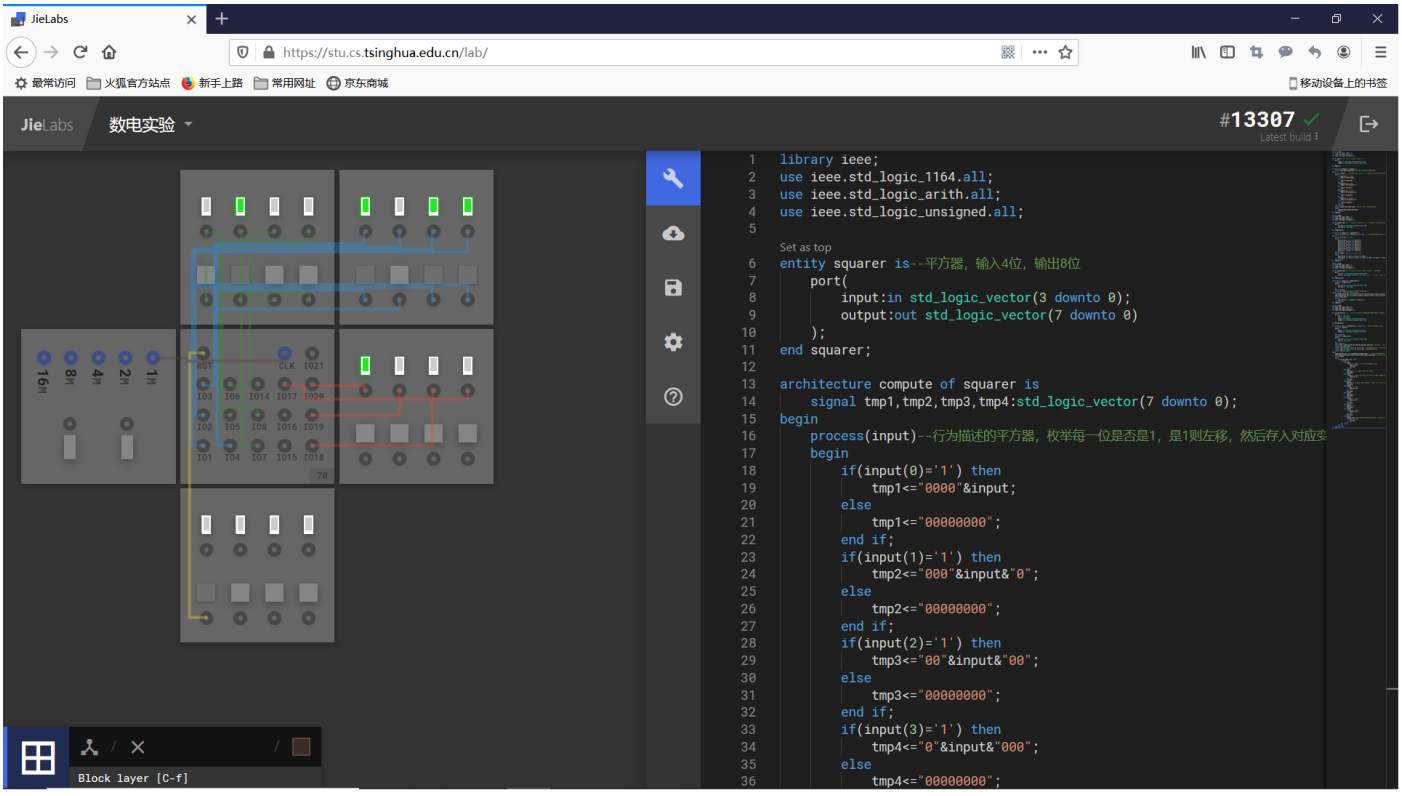
以上样例表示的是 floor(sqrt(196))=14



以上样例表示的是 floor(sqrt(192))=13



以上样例表示的是 floor(sqrt(9))=3



以上样例表示的是 floor(sqrt(75))=8