**实验18**

**乘法器**

# **实验介绍**

本次实验将基于加法运算的逻辑电路以及有限状态机的内容，实现一个较为简单的乘法模块。

我们可以通过组合逻辑设计一个乘法器，即在一次调用中完成全部计算；也可以通过时序逻辑实现，即在每个时钟周期内完成一次单位运算，经过若干个周期后得到对应乘积。不同的设计对应不同的效率及资源消耗。因此，在设计出一个基础乘法器后，我们需要对其进行优化，以减少空间开支，同时，也要让其能够适配有符号数的情况。

# **实验目标**

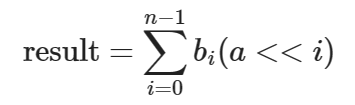
* 进一步掌握时序逻辑电路设计。
* 进一步掌握有限状态机的设计。
* 掌握基本的算术电路的设计方法。

# **实验内容**

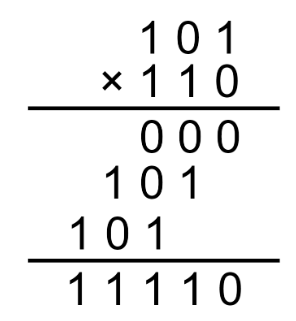
## 1 组合逻辑乘法器

乘法本质上是多次加法运算的叠加。在二进制运算中，乘法则是简单的移位操作。例如 101\_2=5\_{10}，1010\_2=10\_{10}。

如果要将两个无符号 n 位二进制数 a\_{n-1},...,a\_0 和 b\_{n-1},...,b\_0相乘，我们可以将 a的副本移位并加到b中为 1 的位置上。即



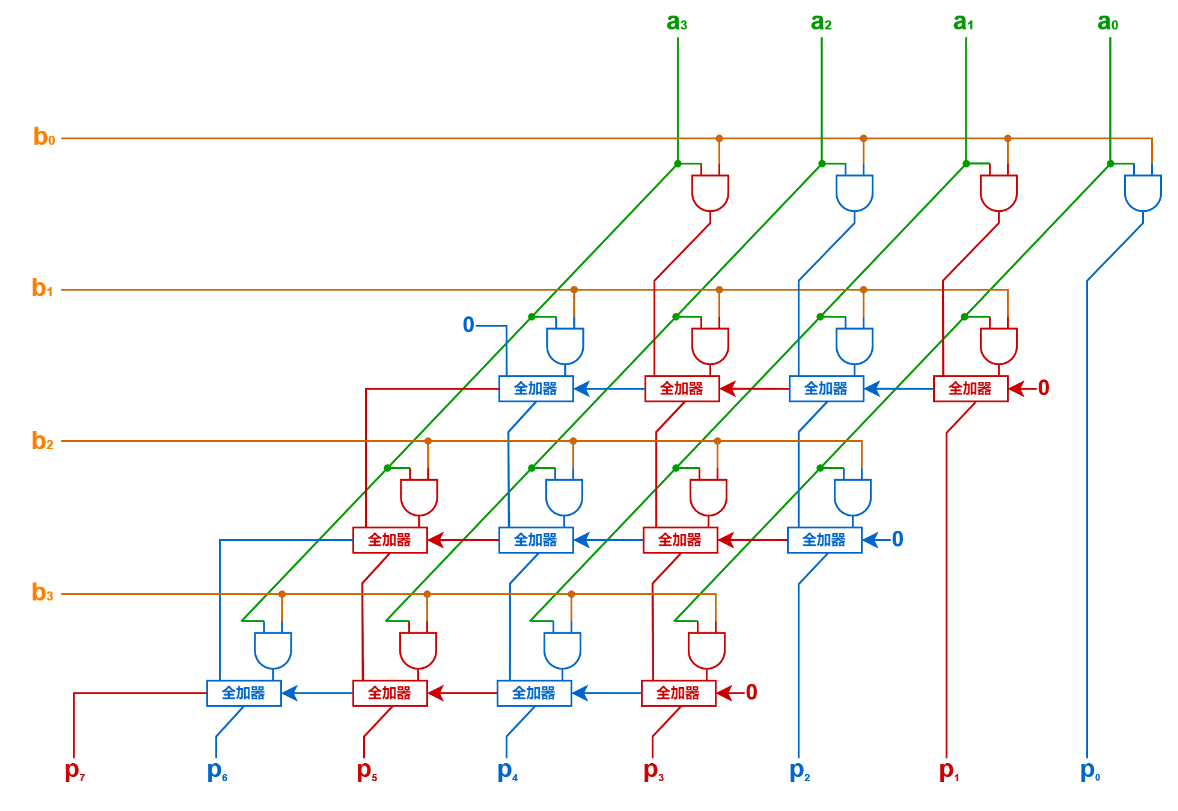
例如：计算 101\_{2}\*110\_2(5\*6))，我们可以用下面的竖式运算过程完成。



可以看到，乘法计算过程共分为三步：

* 第一步计算移位，即将乘数的副本左移 0~n-1 位；
* 第二步将移位结果AND被乘数对应位置的数值；
* 第三步计算每一位的最终和。

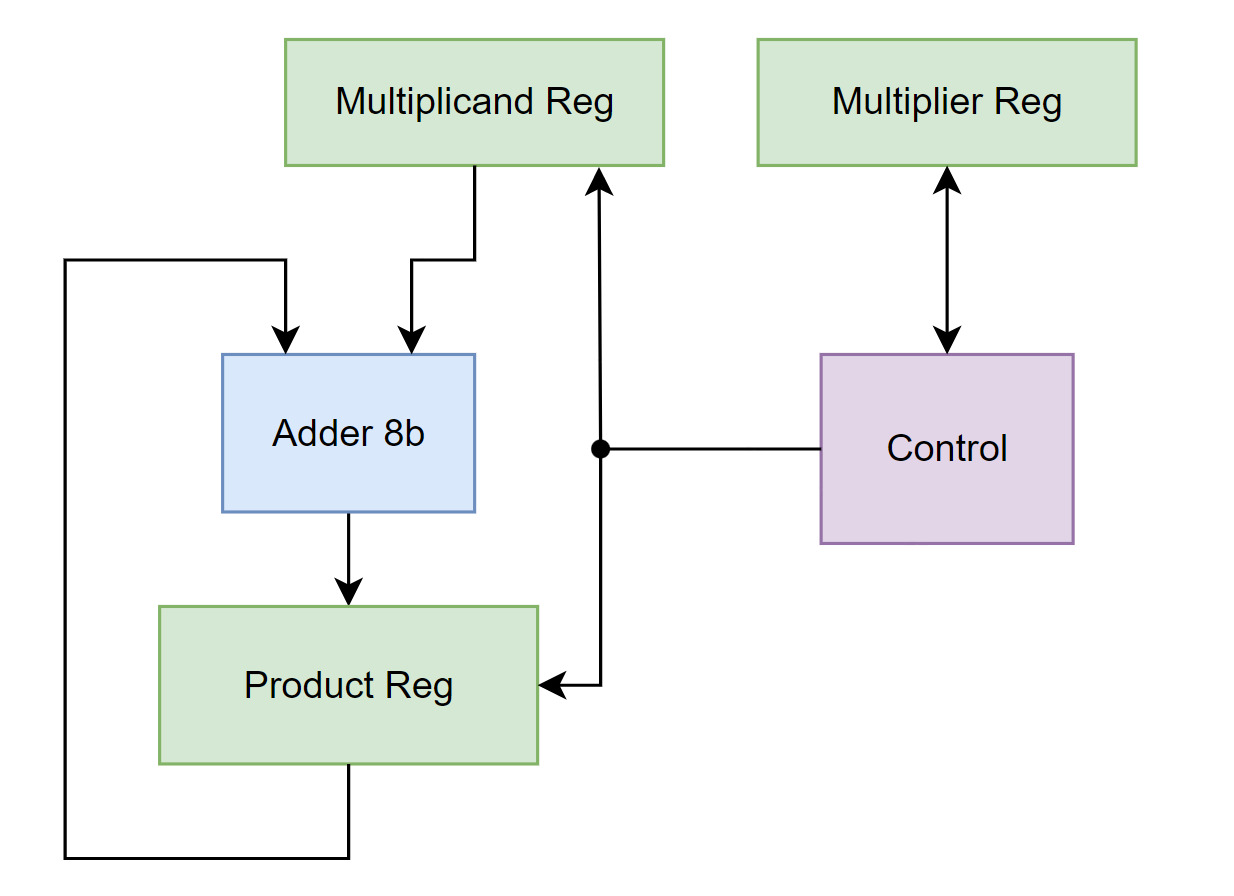
因此，四位无符号二进制数乘法电路的结构可以如下图所示：



## 2 移位乘法器

### ****2.1 基础移位乘法****

.基于组合逻辑实现的乘法器电路较为复杂，且消耗的硬件资源也较多。因此，我们可以采用时序逻辑电路实现乘法运算的过程。下图展示了一个基础的乘法器结构：

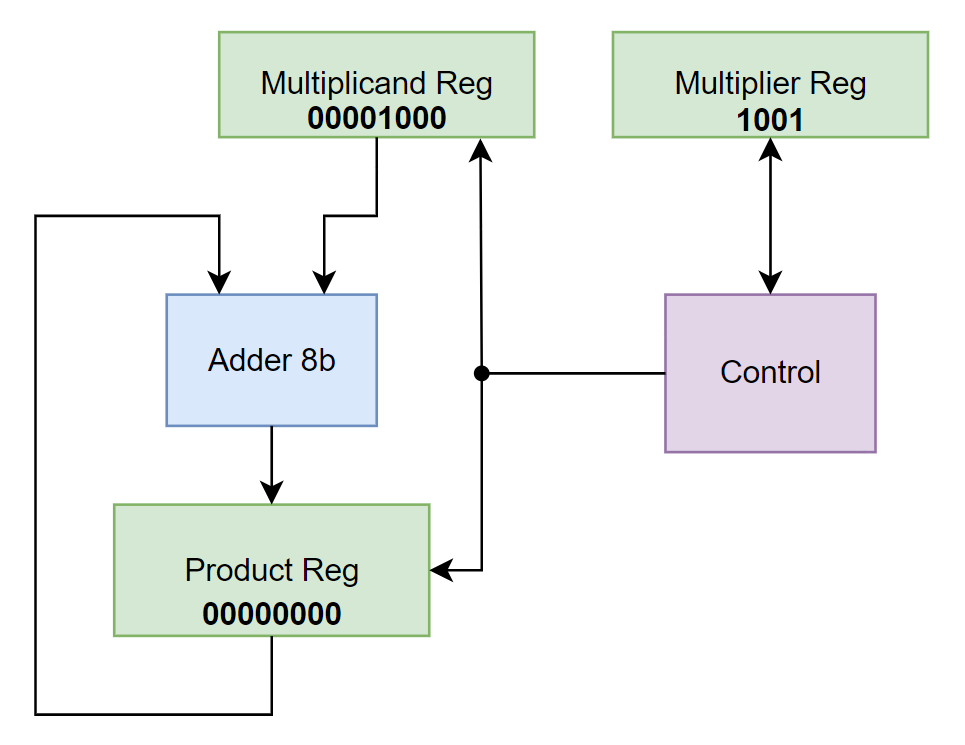
[](https://soc.ustc.edu.cn/Digital/lab7/figs/mul/mul_v1.png)

整个电路包含三个寄存器（被乘数 Multiplicand、乘数 Multiplier、乘积 Product）、一个加法器和一个控制单元。我们用一个简单的二进制乘法作为例子：1000×1001。这是两个四位的二进制数相乘，为此我们要实现一个四位的乘法器。

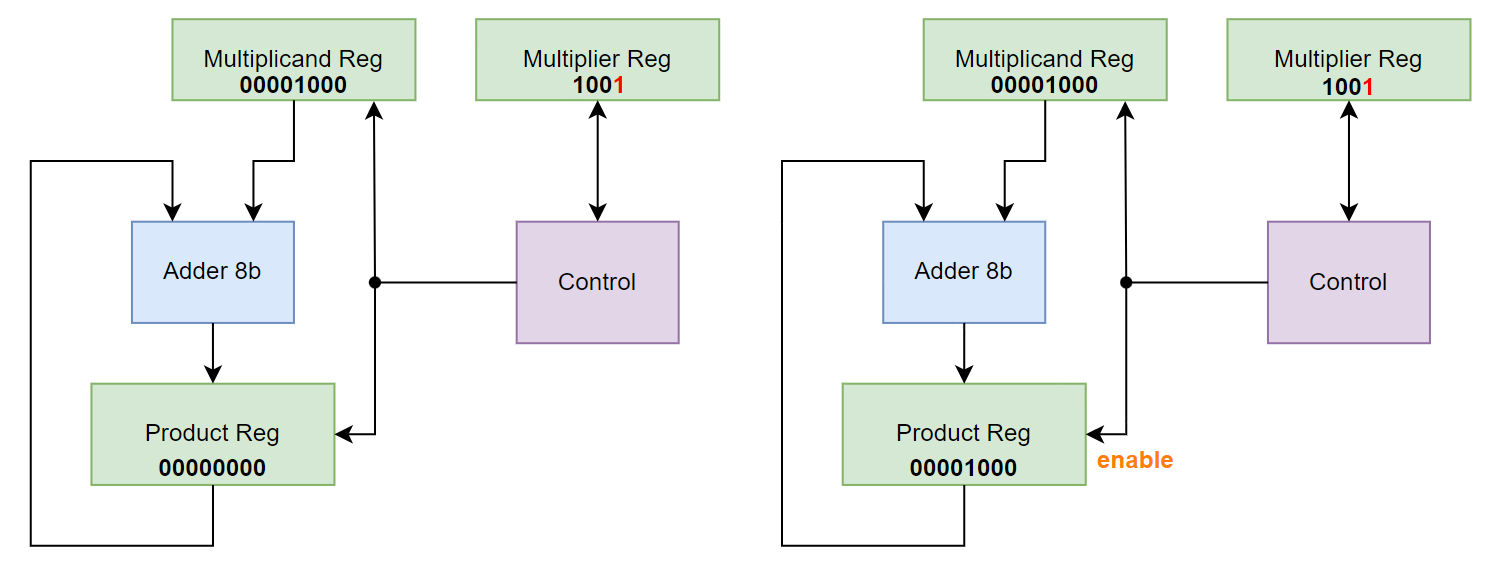
* 被乘数寄存器（Multiplicand Reg）是一个 8 位的寄存器，且带有左移的功能。它有一个左移的控制信号输入，当外部的控制单元（Control）将这个信号置为有效时，在下一个时钟上升沿到来时被乘数寄存器当中的内容就会向左移动一位。
* 乘积寄存器（Product Reg）也是一个 8 位的寄存器，用来保存当前运算的结果。该寄存器包含一个写使能信号，由控制单元给出。
* 乘数寄存器（Multiplier Reg）是一个 4 位的寄存器，且带有右移的功能。该信号同样由外部的控制单元给出。此外，控制单元会读取乘数寄存器最低位的数值，以决定下一阶段的运算过程。

被乘数寄存器和乘积寄存器当中的内容需要进行加法运算，因此我们就需要一个 8 位的加法器，它会将被乘数寄存器和乘积寄存器当中的内容进行相加，并将结果再送到乘积寄存器当中。

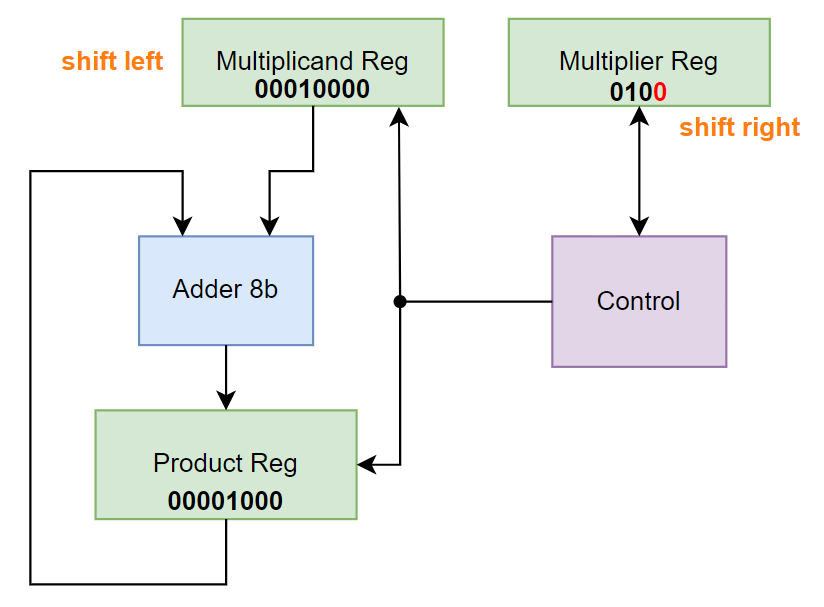
现在，我们就来看一看这样的一个乘法器是如何工作的。首先，我们对乘法器进行初始化。置乘数为 Multiplier = 4'B1001，被乘数为 Multiplicand = 8'B0000\_1000，乘积寄存器设置为 Product = 8'B0000\_0000。

[](https://soc.ustc.edu.cn/Digital/lab7/figs/mul/mul_s0.png)

下面开始计算，请大家回忆我们用纸笔时是如何进行这个运算的。首先，我们检查乘数寄存器的最低位，看其是否为 1。现在发现它是 1，那么就需要将被乘数寄存器当中的内容和当前乘积寄存器当中的内容进行相加。控制单元向乘积寄存器发出写使能信号 enable，乘积寄存器便会写入当前加法器的运算结果（我们用黄色字体表示当前时钟上升沿已完成的操作）。

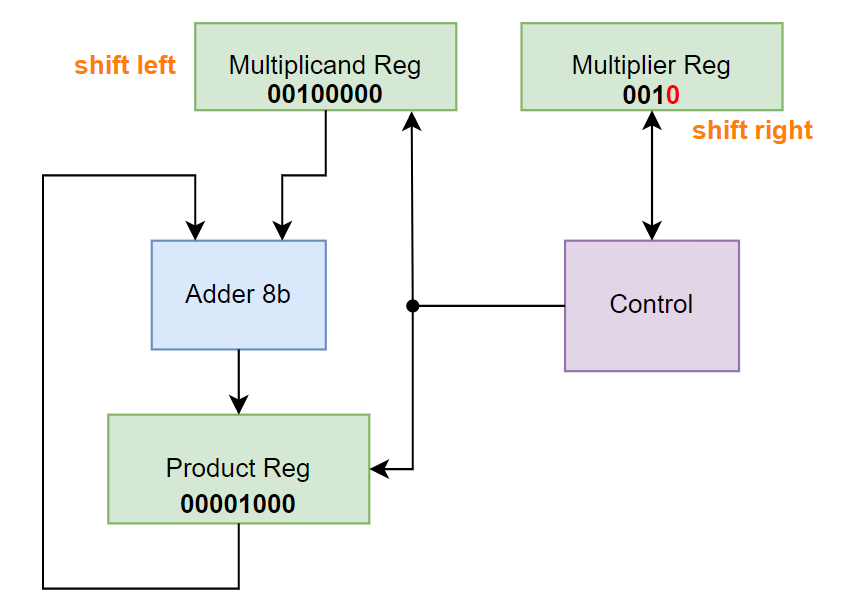


在一步运算结束之后，我们需要将被乘数左移一位，进入下一次计算。

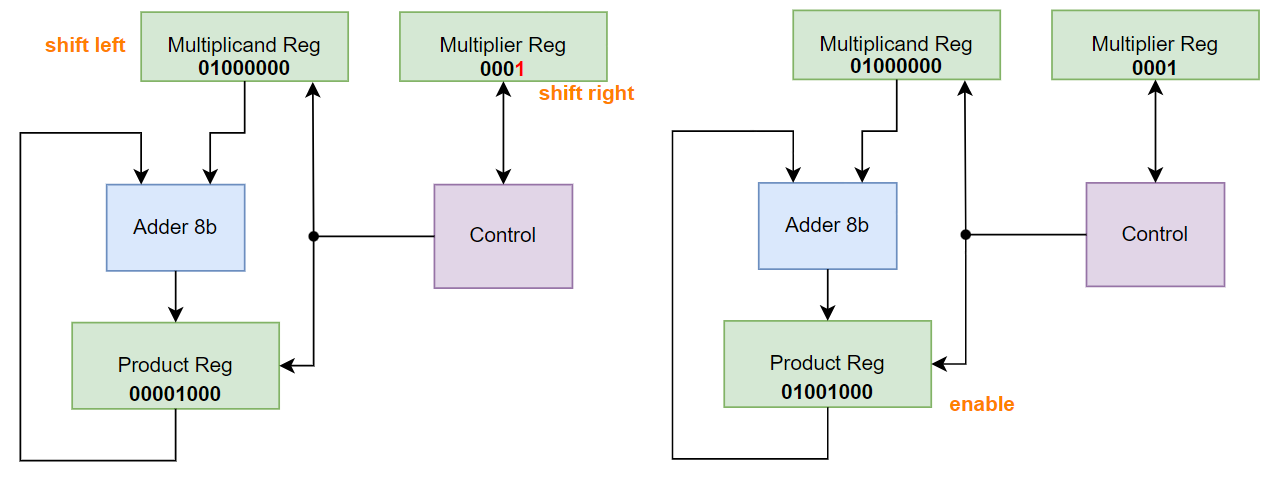
[](https://soc.ustc.edu.cn/Digital/lab7/figs/mul/mul_s2.png)

此时，由于最低位是 0，因此乘积寄存器不会进行写入操作。

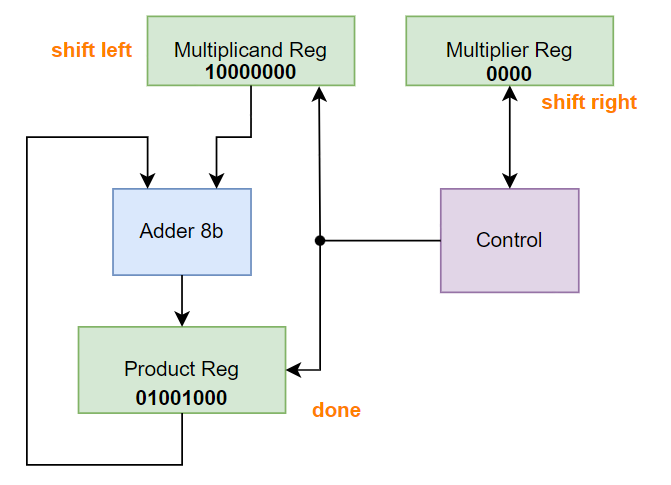
接下来重复上述的操作过程。因为乘数的位数是 4，所以我们需要重复执行上述操作共 4 次，才能在乘积寄存器中读取到最终的结果。

[](https://soc.ustc.edu.cn/Digital/lab7/figs/mul/mul_s3.png)

第三步计算

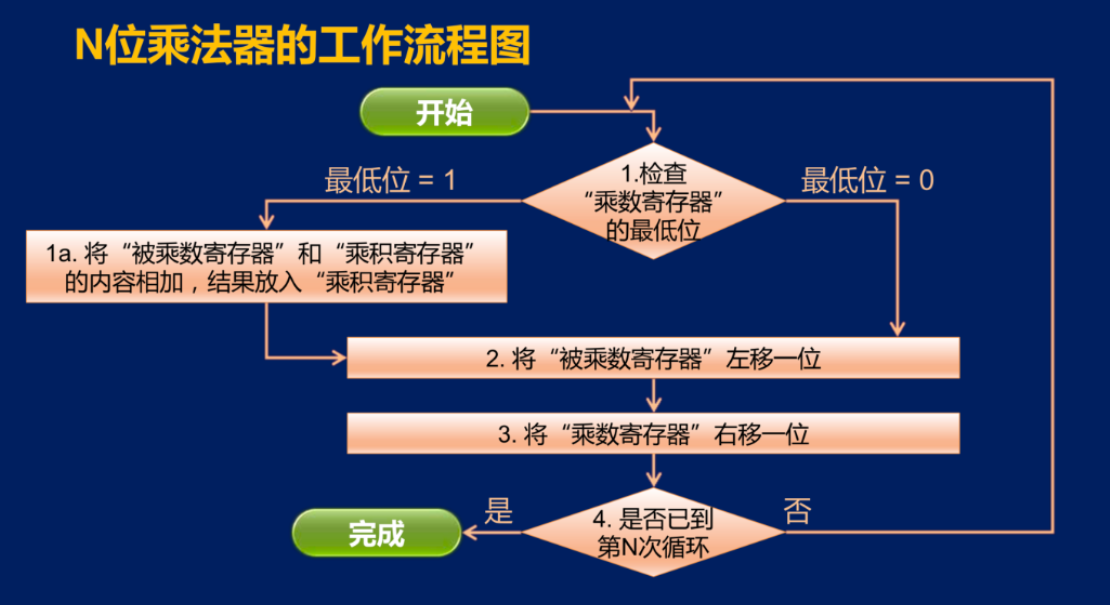
[](https://soc.ustc.edu.cn/Digital/lab7/figs/mul/mul_s4.png)

第四步计算

[](https://soc.ustc.edu.cn/Digital/lab7/figs/mul/mul_s5.png)

最终乘积寄存器的结果为 010010002=72，而 10002×10012=8×9=72，表明我们的运算过程是正确的。

总结一下，N 位乘法器（被乘数和乘数为 N 位，乘积为 2N 位）的工作流程如下：

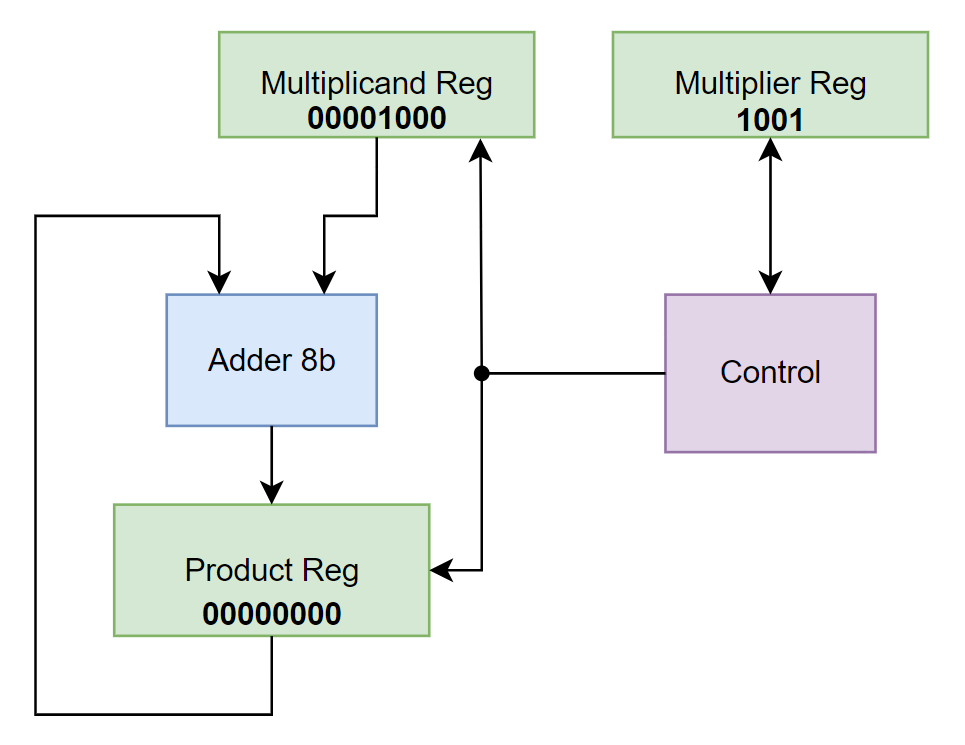


一个简单的判断标准是：被乘数寄存器和乘数寄存器都需要移位 N 次。

### ****2.2 乘法器优化版****

在购买一台计算机时，我们不仅希望它的性能足够好，还希望它的价格足够低。那对于 CPU 这样的集成电路而言，其价格的一个重要因素就是其中晶体管的数量，或者说是芯片的面积。因此，在我们设计各个功能部件的时候，如何减小芯片的面积也是一个重要的优化方向。

那么，上面的移位乘法器有没有可以优化的地方呢？我们再次回到计算的初始状态：

[](https://soc.ustc.edu.cn/Digital/lab7/figs/mul/mul_s0.png)

不难发现：

1. 对于被乘数寄存器（Multiplicand Reg），由于我们的被乘数只有 4 位，因此该寄存器中的高 4 位是无用的。
2. 对于乘数寄存器（Multiplier Reg），虽然一开始是没有浪费的，但是在运算的过程中，每个周期其内部有效的数字便会减少 1 位。因此，这也是一种潜在的浪费。
3. 最后，乘积（Product）确实需要 8 位才能保存，但是乘积寄存器在不断的保存中间结果，而中间结果一开始并不是 8 位的。当被乘数还集中在最低四位的时候，这个运算结果实际的有效数字只有最低的 4 位，只是随着被乘数寄存器不断的左移，乘积寄存器当中的有效数字才不断地增加，最后才达到了 8 位。所以说，对于乘积寄存器，虽然最终它没有浪费，但是在计算的过程中是存在资源浪费的。

我们应当如何基于上述的分析对移位乘法器进行优化呢？

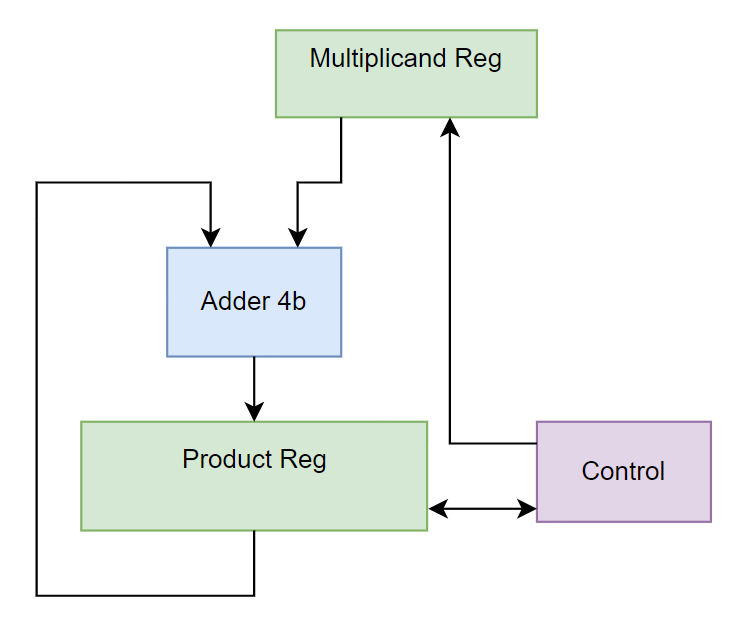
首先，被乘数寄存器是一个 8 位带有左移功能的寄存器，但其内部的有效数字始终只有 4 位。我们可以直接使用一个 4 位的寄存器。并取消其左移功能。这样，被乘数的 4 位就可以一直要参与运算，寄存器也就变成了一个普通的寄存器。

其次，乘积寄存器是 8bits 位宽的，但是初始时有效数字只有 4 位，且每周期增加 1 位。我们不能削减乘积寄存器的宽度，因为乘法运算的结果需要 8bits 位宽。但在运算过程中，我们需要保持成绩寄存器和被乘数寄存器之间的对齐关系。既然被乘数寄存器不能左移了，我们让乘积寄存器右移也可以达到同样的效果。此时，乘积寄存器的初始值就需要放在寄存器的高四位，以便于接下来的右移过程。

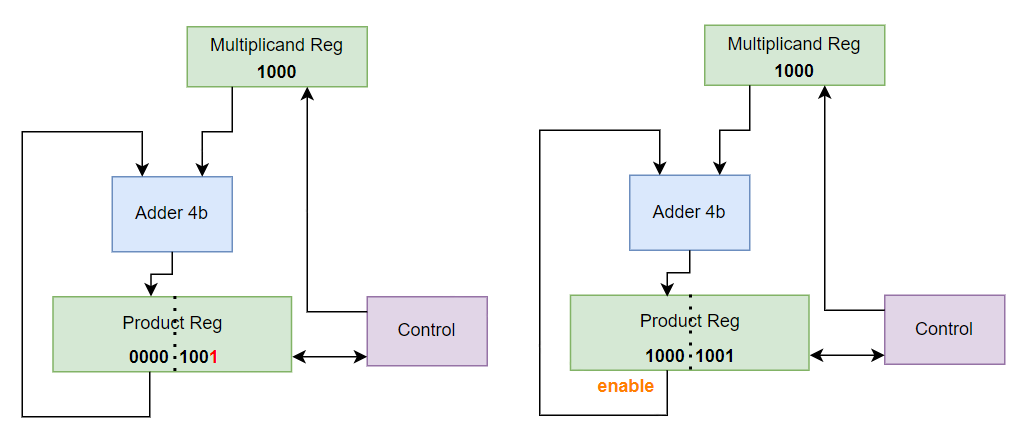
现在，参与加法运算的有效数字实际上只有 4 位（因为被乘数寄存器已经被削减到了 4 位，而乘积寄存器初始值放在了高 4 位的地方），所以我们只需要一个 4 位的 ALU。

最后，我们还需要一个 4 位带有右移功能的乘数寄存器。仔细思考一下，我们真的还有必要设置这个寄存器吗？你可能已经注意到了：乘积寄存器的低四位还是空余的，并且它正好是一个最开始位宽为 4bits 且带有右移功能的寄存器！此外，乘积的结果每个周期会右移 1 位，因此乘积寄存器低位的空余每个周期会减少一位，而这正好是乘数寄存器所需要的功能。所以，我们可以把乘数放在乘积寄存器的低 4 位，从而取消乘积寄存器这个部件。

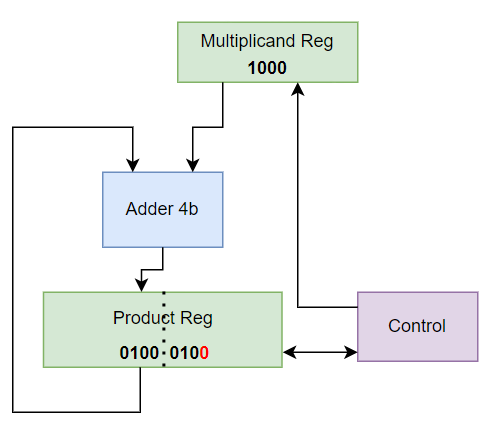
基于上面的分析过程，我们可以得到以下优化版的乘法器，该乘法器相比于基础版减少了一定的空间开支。

[](https://soc.ustc.edu.cn/Digital/lab7/figs/mul/mul_v2.png)

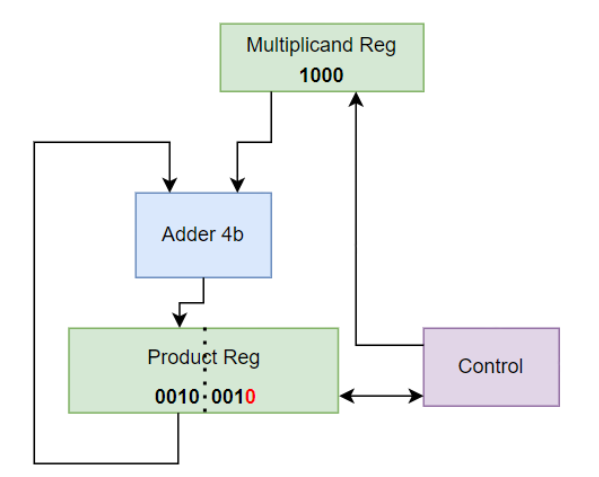
同样的计算过程如下：

[](https://soc.ustc.edu.cn/Digital/lab7/figs/mul/mul_v2s1.png)

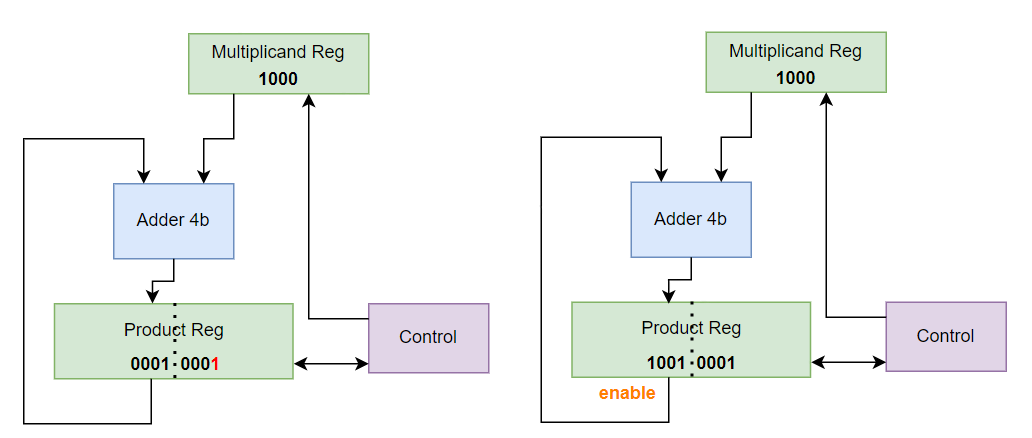
第一步计算

[](https://soc.ustc.edu.cn/Digital/lab7/figs/mul/mul_v2s2.png)

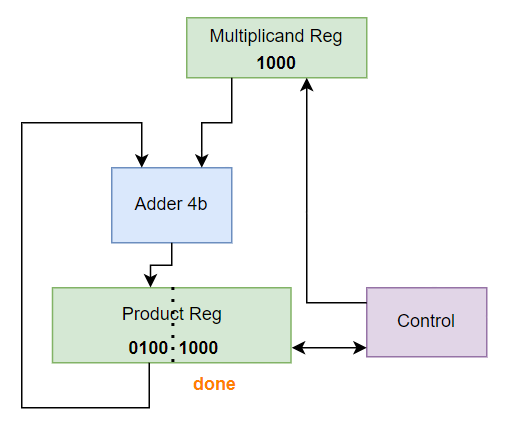
第二步计算



第三步计算

[](https://soc.ustc.edu.cn/Digital/lab7/figs/mul/mul_v2s4.png)

第四步计算

[](https://soc.ustc.edu.cn/Digital/lab7/figs/mul/mul_v2s5.png)

计算完成

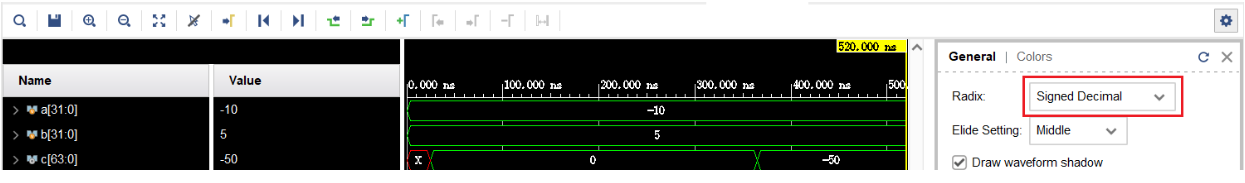
但是！上面介绍的乘法器结构存在一个 Bug。当我们在计算 1111×1111 时就会发现，加法器在加法运算时出现了溢出！如果直接丢弃这些溢出会导致结果运算错误。因此，我们需要在乘积寄存器中额外增加 1 位的空间，用于储存可能溢出的结果。

对于经过优化后的 N 位乘法器。我们需要一个 N 位的寄存器保存被乘数，还需要一个 N 位的加法器进行中间结果的运算，以及一个 2N+1 位的带右移功能的寄存器同时保存乘积和乘数。

### ****2.3 有符号乘法****

到目前为止，我们只处理了无符号数乘法的情况。对于如何处理带符号乘法，最简单的方式是先把乘数和被乘数转换为正数（取反加一），然后记住它们的初始符号。这样，将之前的算法迭代执行 31 次，符号位不参与计算（也可以不做改动执行 32 次，因为乘数不管转换与否符号位总为 0，但会多造成一个周期的资源浪费）。正如我们在小学学到的那样，只有在乘数和被乘数符号相反时，对积取反。

在测试有符号乘法器时，你可以在仿真时将显示的进制设置为有符号十进制（Signed Decimal）。



# 思考与练习

1. 请根据实验文档中给出的移位乘法器设计方法，设计一个基础的参数化移位乘法器。部分信号约定如下：

* start 为开始运算信号，默认值为 0。当 start 由 0 变为 1 时需要初始化内部的被乘数、乘数和乘积寄存器，并从下下个时钟周期开始进行乘法计算（初始化需要一个周期）。
* finish 为运算完成信号，默认值为 0。当 finish 变为 1 时，表明此时乘法器的输出端口 res 为有效的乘积结果。为了保证稳定性，finish 信号需要持续至少一个时钟周期的高电平。
* rst 为复位信号，默认值为 0。当 rst 变为 1 时，乘法器内部的被乘数、乘数和乘积寄存器清零，且状态跳转到 IDLE。
* a 和 b 为被乘数和乘数，均为 WIDTH 位宽的变量。
* res 为最终的乘积，为 2\*WIDTH 位宽的变量。

请通过仿真验证移位乘法器的正确性。本题的框架代码如下：



2. 请根据实验文档实现一个有符号乘法器。