**组成原理课程第 三 次实报告**

**实验名称：CPU设计实战三次实践**

学号： 2312141 姓名： 张德民 班次：李涛老师班

## 实验目的

根据《CPU设计实战》书中的第三章讲解，完成Lab2的三个实验，深入认识寄存器堆，了解同步RAM与异步RAM，并且学习使用vivado进行调试代码，发现bug。

## 实验内容说明

请根据《CPU设计实战》书中的第三章讲解，完成Lab2的三个实验，并撰写实验报告。

1、针对任务一寄存器堆实验，完成仿真，在感想收获中思考并回答问题：为什么寄存器堆要设计成“两读一写”？

2、针对任务二同步ram和异步ram实验，可以参考实验指 导手册中的存储器实验，注意同步和异步需要分开建工程，然后仿真，在感想收获中分析同步ram和异步ram各自的特点和区别。

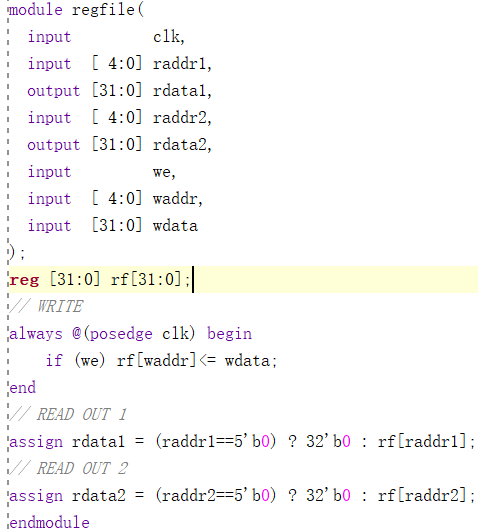
3、针对任务三，重点介绍清楚发现bug、修改bug和验证的过程，在感想收获中总结使用vivado调试的经验步骤。

## 实验步骤

### 任务一：寄存器堆仿真实验

代码分析：由于代码过多，因此主要分析核心的部分。

#### regfile模块的实现：



这个模块定义了两个读地址raddr1和raddr2，用于规定要读取的寄存器，寄存器是32个，所以读地址是5位，2^5=32。还用两个读数据堆，rdata1和rdata2，用于存储从寄存器里读到的数据。we(write enable)是写使能信号，只有当其为1时，才能向寄存器中写入。

waddr用于确定向哪一个寄存器写入，wdata用于确定写入什么数据。然后是32个32位的寄存器。

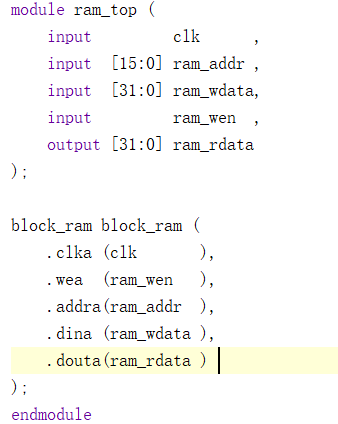
之后使用always语句，使得在时钟上升沿，如果we为1，那么就向指定寄存器写入指定数据。

然后读取数据，规定若raddr为0，则读取0，否则读取相应寄存器的值。

### 任务二：同步RAM与异步RAM仿真

#### 同步RAM仿真实验

##### block\_ram\_top模块的实现

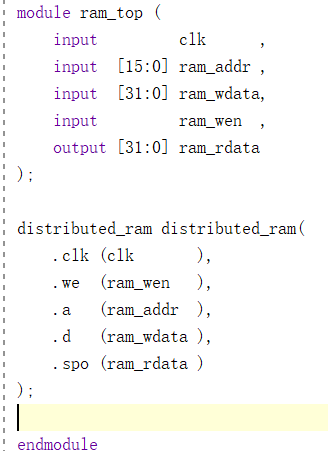


这个模块有四个输入端口，clk是时钟端口，ram\_addr是ram寄存器存取的地址，ram\_wdata是写入的数据，ram\_rdata用来存储读出的数据，ram\_wen是使能信号，为0时代表读，为1时代表写。

并且为了保持一致性，所有的读写操作都是在时钟上升沿进行的。

#### 异步RAM仿真实验

##### distributed\_ram\_top模块的实现

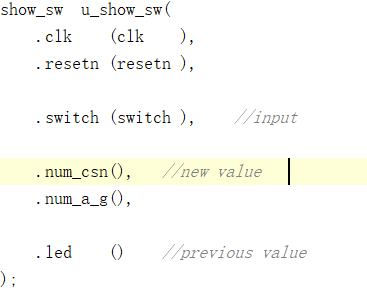


和同步ram的代码基本一致，只有名称换了。核心的修改点在于使用的RAM IP不同，异步RAM与同步RAM不同，它进行读写操作时不需要等到时钟上升沿，而是立即进行的。同时也没有写后读的机制。

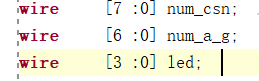
### 任务三：数字逻辑电路的设计与调试

这里有五个bug，我们根据书中的提示了一个一个寻找。

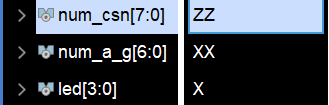
#### Bug1：波形为Z



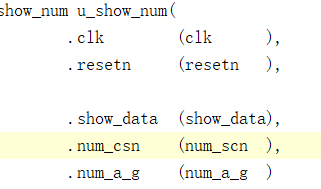
我发现，问题是在tb代码里调用模块时，部分端口没有定义，也没有连接。要进行修改。



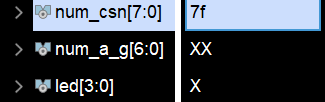
我们在tb里加入这些定义，并连接。



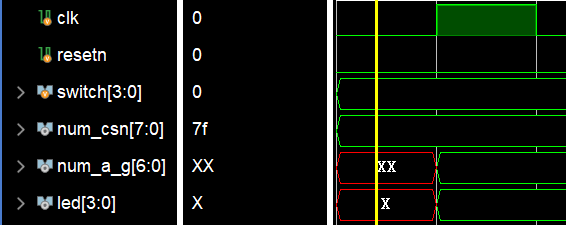
波形图显示了这三个值，但是num\_csn的值是Z,应该是没有连接，而num\_a\_g和led是X不定值，应该是没有进行赋值。



这里我们发现，num\_csn在连接的时候拼错了，这应该是导致Z值的原因，我们修改回来。

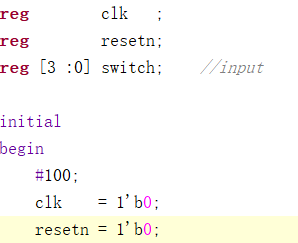


可以看见，这样num\_csn就正常了。但是num\_a\_g和led还是不定值，这是因为它们最初还没有被赋值。

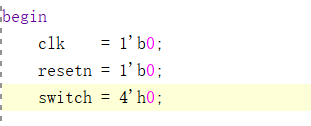


但是可以看见，在第一个时钟上升沿后，二者就赋值了，因此这不算bug，没有必要进行赋值修改。

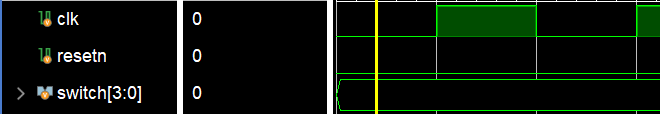
#### Bug2：波形为X



产生的原因是reg类型的端口clk，resetn以及switch最初都没有赋值，clk和resetn在100ns后赋值为0，但是switch一直没有，这就导致了这三个值一直是X不定值。

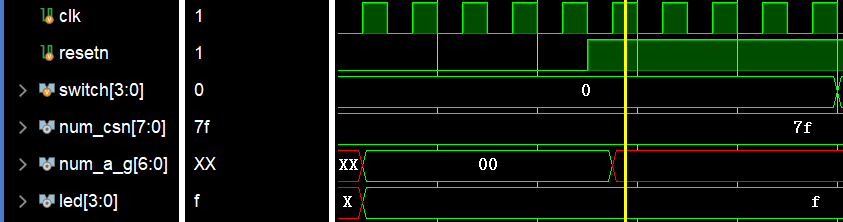


修改方法是删除等待100ns，并给switch赋值为0。

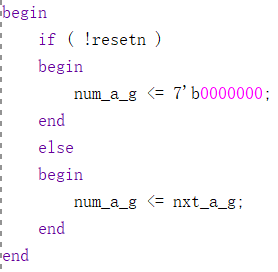


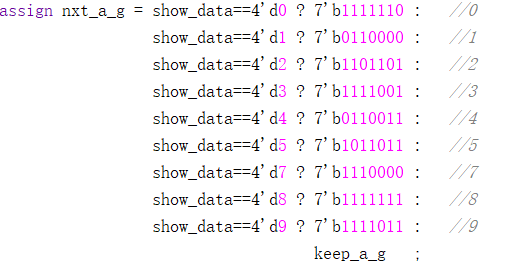
这样处理过后，最初就没有X的问题了。

#### Bug3：波形停止



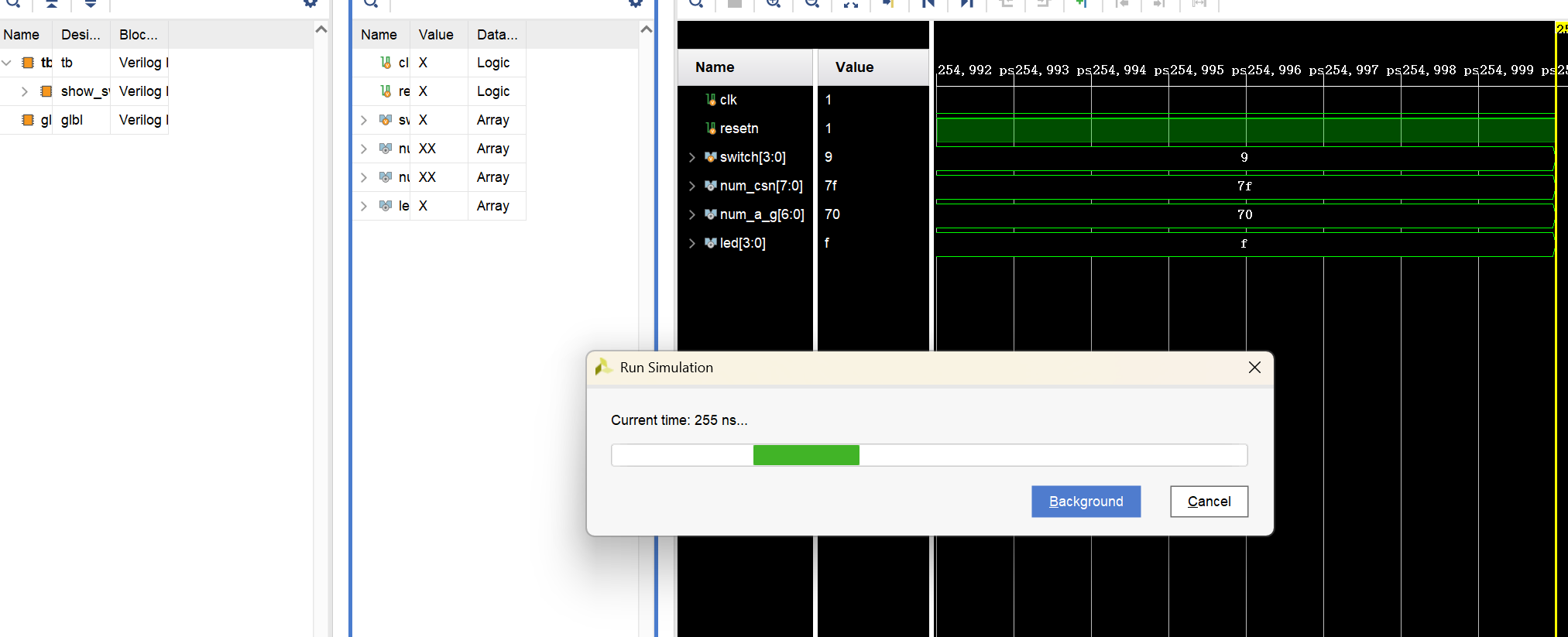
我们观察到当resetn变成1时，num\_a\_g一直是X，这出现了问题。因此我们去查看num\_a\_g的代码。







我们发现num\_a\_g的值由nxt\_a\_g决定，而nxt\_a\_g的值由show\_data决定，而show\_data的赋值语句被注释掉了，我们去除注释再仿真。



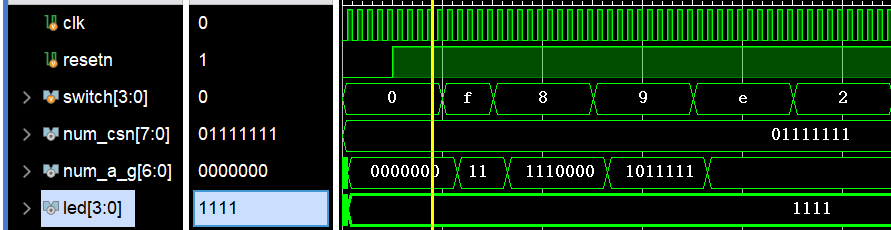
这时我发现num\_a\_g不再是X，但是出现了波形停止的bug。我进行综合来debug。这

种错误往往是RTL里存在组合环路导致的。



系统显示这里有问题，我分析后发现，这一步的目的的当show\_data大于9的时候，显示之前的数据不变，因此这句的逻辑就不正确，只需要保留一个num\_a\_g就可以了。原来的代码里nxt\_a\_g表达式里有keep\_a\_g，二者相互包含，因此引发了波形停止的bug。修改之后，仿真恢复正常。

#### Bug4：越前采样



我发现，led的值全程都是1111没有改变。因此我去反向追溯。



我发现led由prev\_data决定，



prev\_data由show\_data\_r决定，



show\_data\_r由show\_data决定，这里我发现了问题，没有使用非阻塞延迟。修改后led正常变化。

#### Bug5：功能bug

对于nxt\_a\_g的赋值，缺少了6，我们将其添加上去。

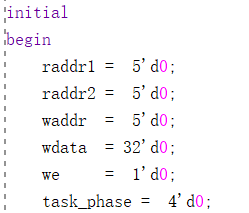


这样程序就正常了。

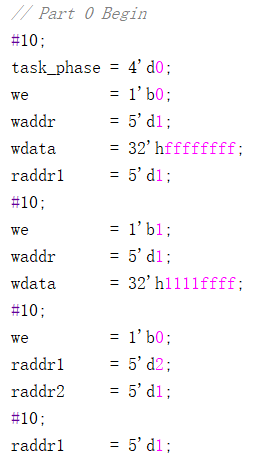
## 实验结果分析

### 任务一：寄存器堆仿真实验

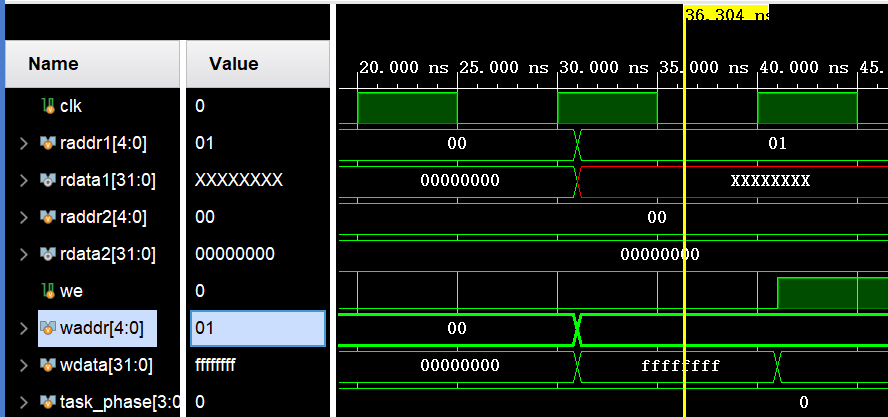
本次结果分析是基于testbench代码设定的数据。



首先，所有值先初始化为0。

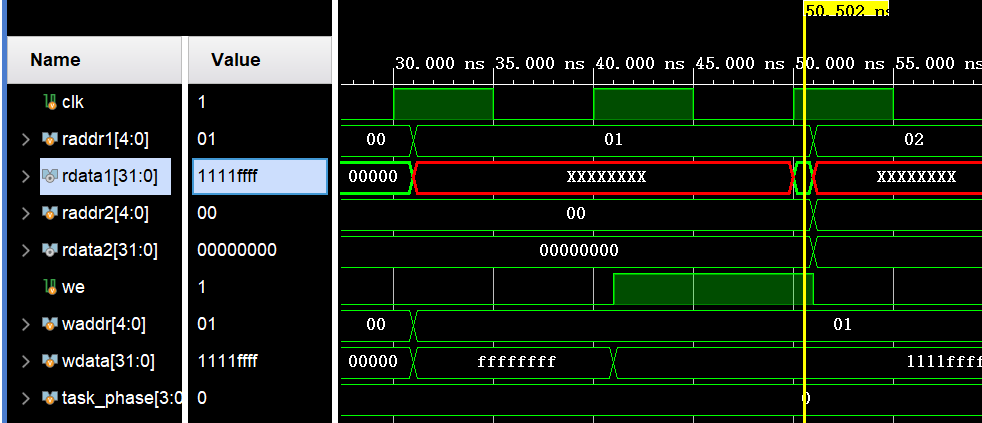


然后开始第0部分，首先要将waddr赋值为1，wdata赋值为0xffffffff，raddr1赋值为1，意味着向1号寄存器写入数据0xffffffff，1号堆要读取的寄存器是1号，不过由于we为0，所以暂时还没有进行写入。



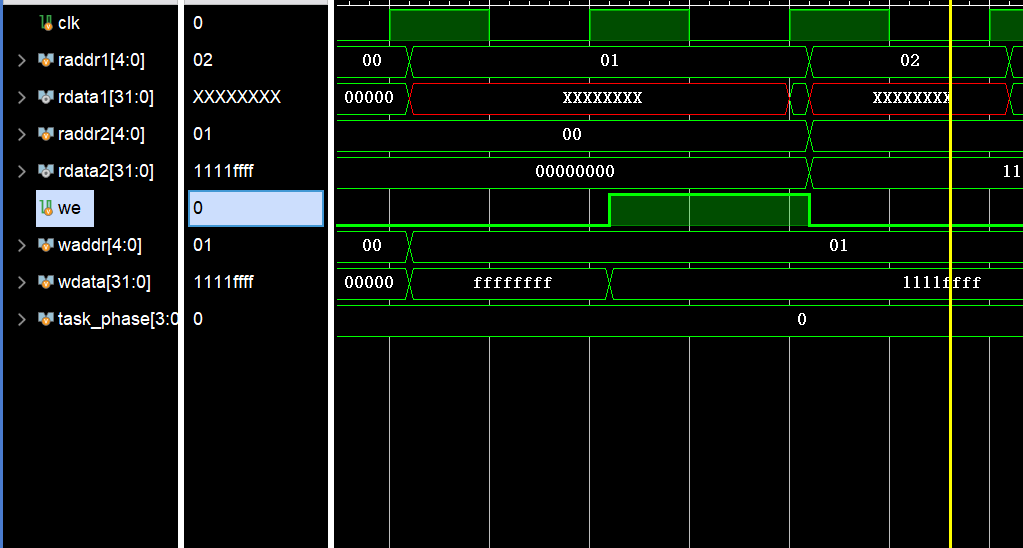
如图，此时waddr为1，wdata为0xffffffff，1号堆读取了1号寄存器，但是由于还没有写入，所以是X不定值。

接下来we赋值为1，waddr赋值为1，wdata赋值为0x1111ffff。



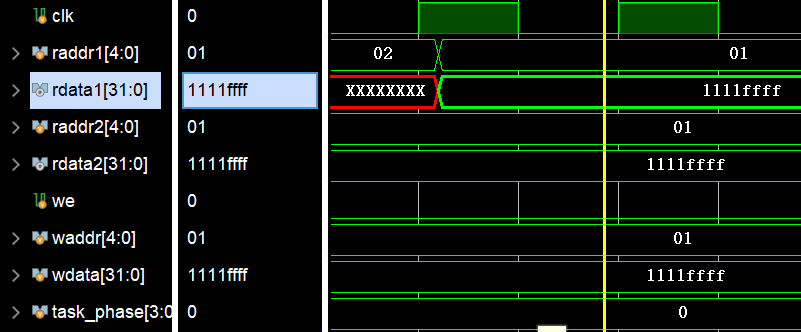
可以看见，we，waddr，wdata的赋值同时进行，因为读取操作只有当时钟上升沿才会触发，所以当在一个时钟上升沿时，rdata1变成了寄存器1的值1111ffff。

然后将we赋值为0，raddr1赋值为2，raddr2赋值为1,即1号堆要存储2号寄存器的值，2号堆存储1号寄存器的值。

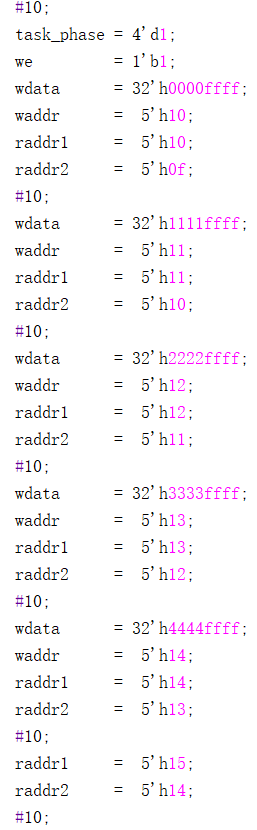


如图，rdata1变成了不定值，rdata2变成了1111ffff。

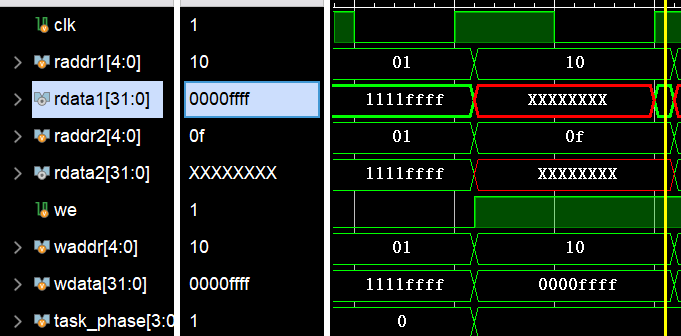
再之后又把addr1改为1号寄存器。所以1号堆也要存1111ffff。



接下来开始第1部分：

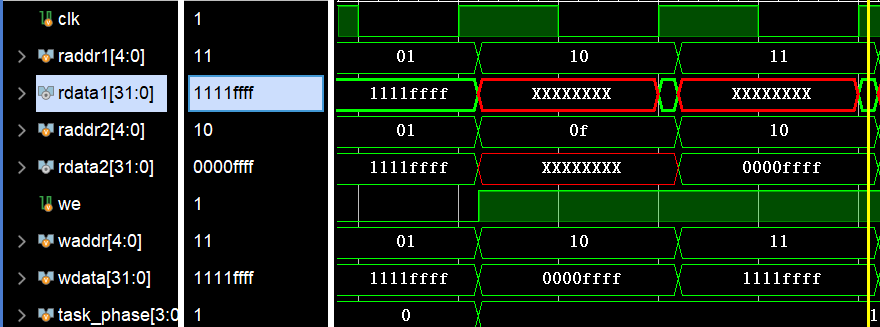


首先向16号寄存器写入0x0000ffff，然后让1号堆读取16号寄存器，2号堆读取15号寄存器。



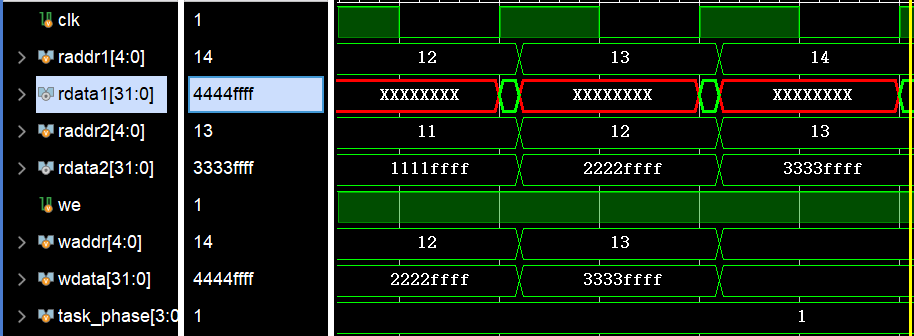
此时一个上升沿以后，1号堆存入了16号寄存器的值0000ffff，而由于15号寄存器没有写入，所以2号堆存入的是不定值X。

然后向17号寄存器写入0x1111ffff，然后让1号堆读取17号寄存器，2号堆读取16号寄存器。



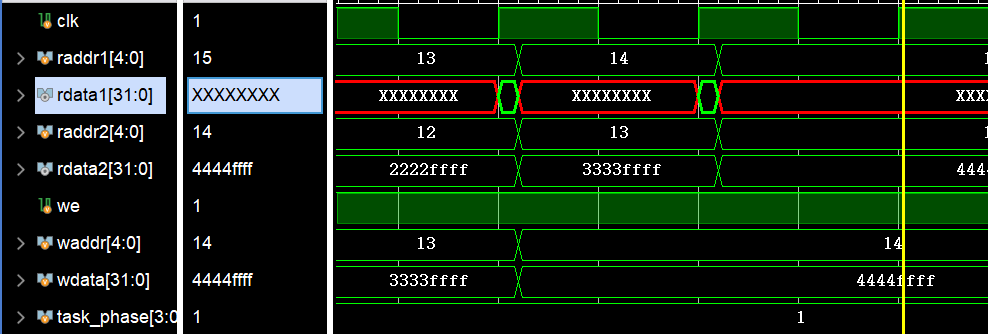
可见，此时1号堆存入了17号寄存器的值1111ffff，2号堆存入了16号寄存器的值0000ffff。

之后由按照这种格式往后类推，到向20号寄存器写入0x4444ffff，然后让1号堆读取20号寄存器，2号堆读取19号寄存器。



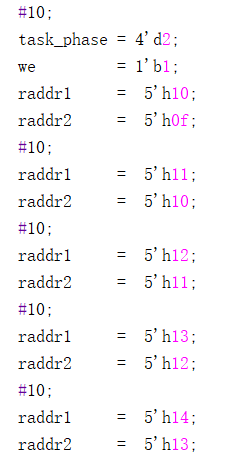
结果与之前的类似。

然后，raddr1改为21号寄存器，raddr2改为20号寄存器，这是21号寄存器没有写入，所以1号堆存入不定值X。

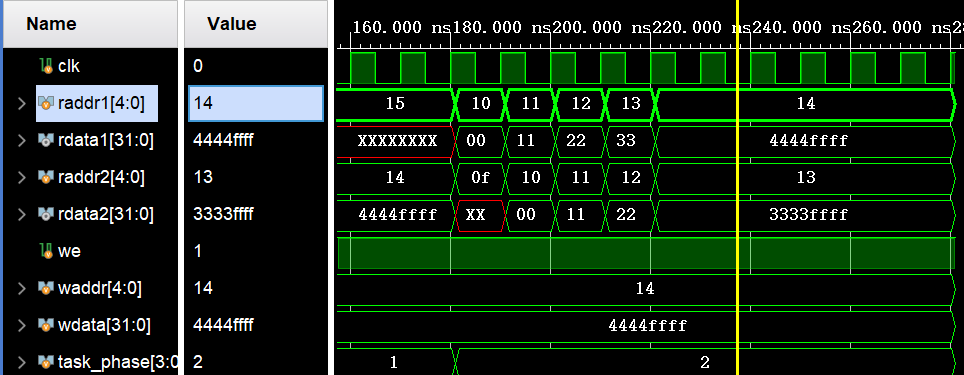


可见，1号堆为不定值X，2号堆是20号寄存器的值4444ffff。

最后开始第2部分：



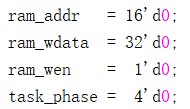
首先，让1号堆读取16号寄存器，2号堆读15号，之后让1号堆读取17号寄存器，2号堆读16号，让1号堆读取18号寄存器，2号堆读17号，让1号堆读取19号寄存器，2号堆读18号，让1号堆读取20号寄存器，2号堆读19号。除了15号寄存器没有写入，其余寄存器都是之前已经写入的。



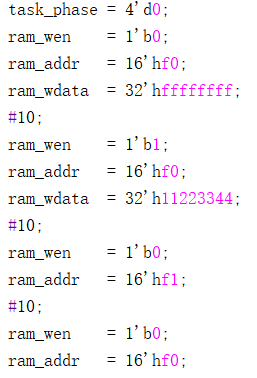
如上图，这里读取的数据和第1部分写入的一致，与我们分析的一致，实验正确。

### 任务二：同步RAM与异步RAM仿真

#### 同步RAM仿真实验



首先所有端口初始化为0。

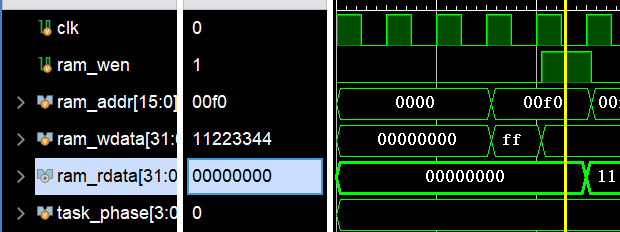


开始第0部分，先指定ram中的地址是f0，然后写入的数据是0xffffffff。ram\_wen是0，代表此时进行读操作，因此无法写入。



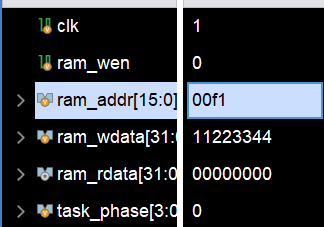
如上图，各个端口数据成功修改。此时ram\_rdata是全零，因为还没有写入。

接下来令ram\_wen是1，开始写操作，地址是0xf0，写入的值是0x11223344.



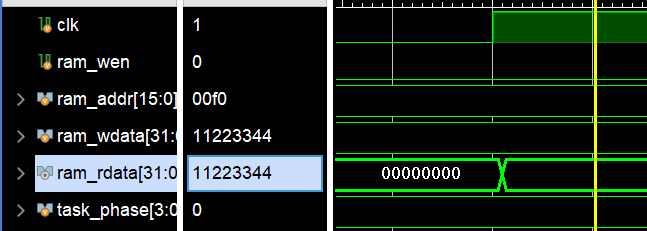
如图，ram\_wen为1，写入的数据是11223344。因为还没有读取，所有ram\_rdata是全0。

接下来令ram\_wen是0，开始读操作，然后修改读取地址为0xf1。这个地址还没有写入，因此应该是全0。



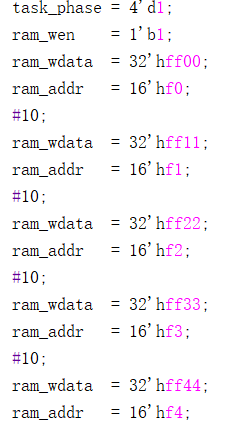
如图，与分析的一致。

接下来继续读取，地址改为0xf0，这是各个写入0x11223344的地址。

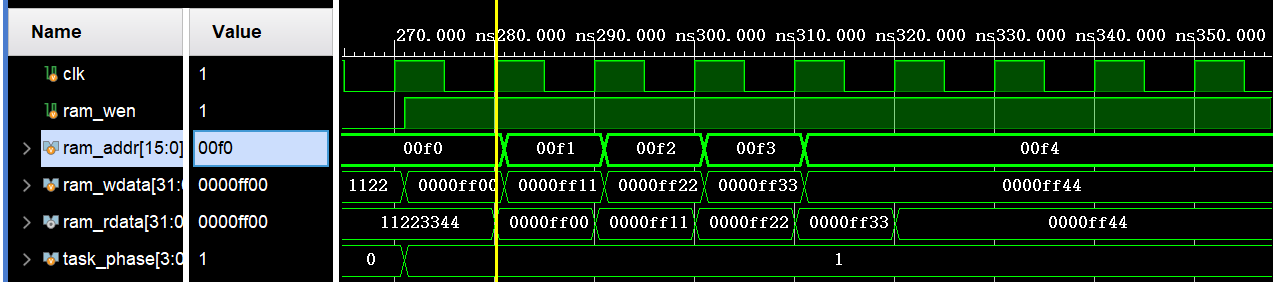


如图，读取的是我们预估的值，正确。

然后开始第1部分。

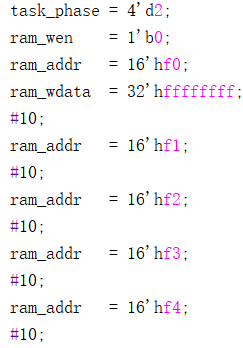


这里进行了连续的写操作，在0xf0写入0xff00，在0xf1写入0xff11，在0xf2写入0xff22，在0xf3写入0xff33，在0xf4写入0xff44。

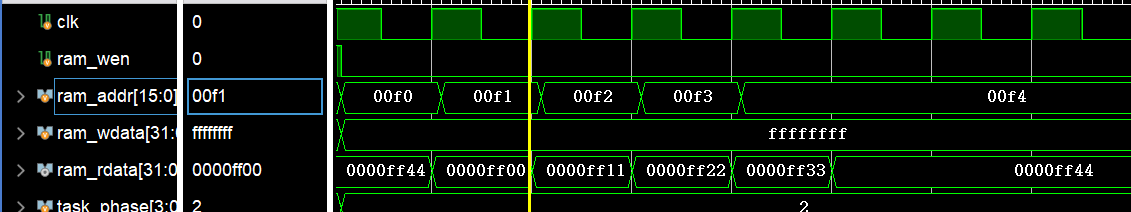


如 上图，写入的顺序与预估的基本一致，但是需要注意到的是，我们的RAM默认设定的是写后读模式，即在时钟上升沿触发时，ram\_rdata会直接保存此时ram\_addr写入的值。这与波形图一致。

接下来开始第2阶段。

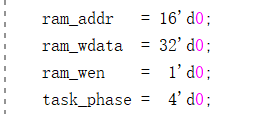


此时开始了连续读操作，ram\_wen赋值为0，写入的数据改为0xffffffff，当然并不会真的进行写入。开始连续读取0xf0,0xf01,xf02,xf03,xf4五个地址的值。

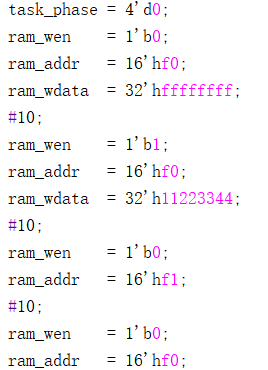


如图，每当时钟上升沿的时候，ram\_rdata就会存入读取到的值，波形图的值与我们刚刚写入的值一致。

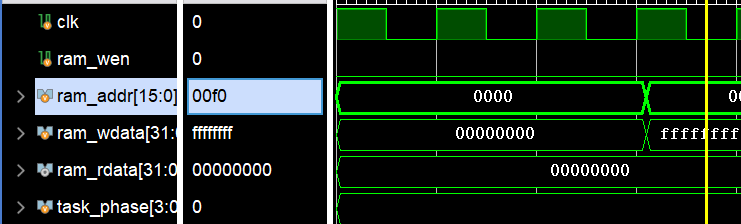
#### 异步RAM仿真实验



首先全部初始化为0。

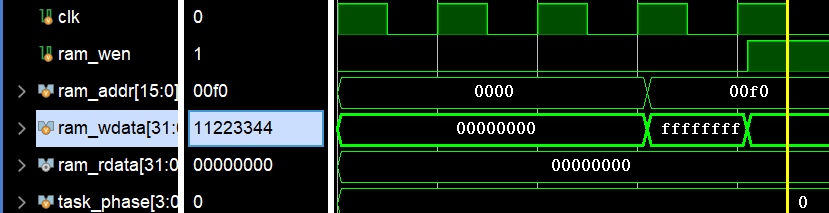


开始第0部分，先指定ram中的地址是f0，然后写入的数据是0xffffffff。ram\_wen是0，代表此时进行读操作，所以无法写入。



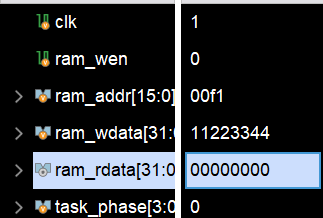
如上图，各个端口数据成功修改。此时ram\_rdata是全零，因为还没有写入。

接下来令ram\_wen是1，开始写操作，地址是0xf0，写入的值是0x11223344.

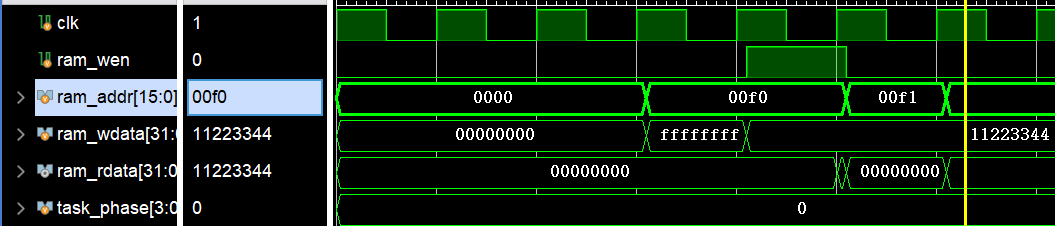


如图，ram\_wen为1，写入的数据是11223344。因为还没有读取，所有ram\_rdata是全0。

接下来令ram\_wen是0，开始读操作，然后修改读取地址为0xf1。这个地址还没有写入，因此应该是全0。

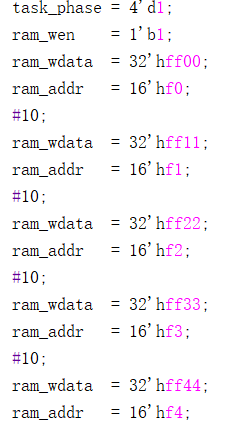


接下来继续读取，地址改为0xf0，这是各个写入0x11223344的地址。

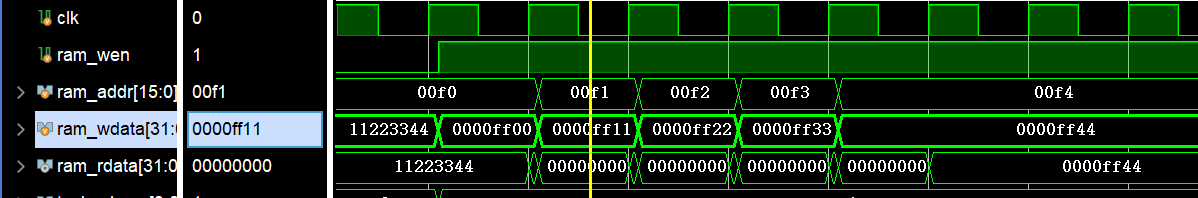


可以看见，此时的ram\_rdata是我们写入的数据，但是与之前不同的一点是，它并没有等到时钟上升沿才进行读取，而是在使能端口改变的瞬间就开始读取。

然后开始第1部分。

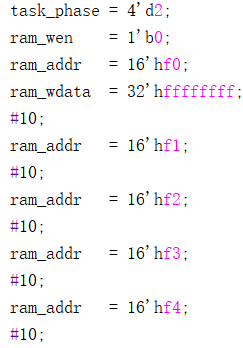


这里进行了连续的写操作，在0xf0写入0xff00，在0xf1写入0xff11，在0xf2写入0xff22，在0xf3写入0xff33，在0xf4写入0xff44。

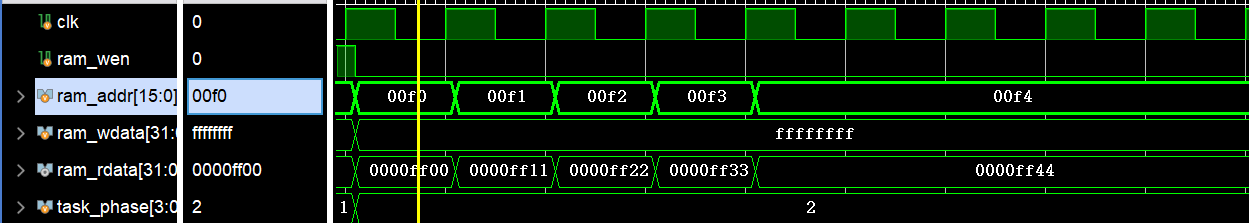


如上图，写入的顺序与预估的基本一致，但是需要注意到的是，我们的异步RAM没有默认设定的写后读模式，因此ram\_rdata一致保持全0不变。这与波形图一致。

最后是第2阶段。

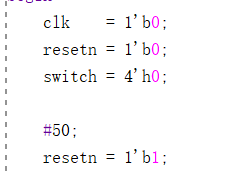


此时开始了连续读操作，ram\_wen赋值为0，写入的数据改为0xffffffff，当然并不会真的进行写入。开始连续读取0xf0,0xf01,xf02,xf03,xf4五个地址的值。



如图，与同步ram不同的是，一旦地址修改，不需要等到时钟上升沿，ram\_rdata就会存入读取到的值，波形图的值与我们刚刚写入的值一致。

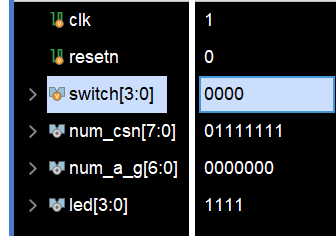
### 任务三：数字逻辑电路的设计与调试



首先初始化，然后令resetn为1，停止清零。

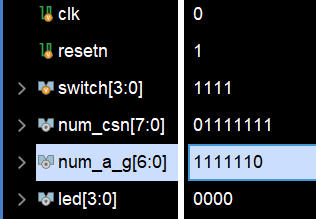


然后等了100ns，由于原先switch为0，取反后就是f。



此时num\_a\_g为00000000,不亮，led1111，代表不亮(这里的led灯为0时候亮，为1的时候不亮，这是因为switch和我们想输入的值是反的，所以led灯也是反的)。

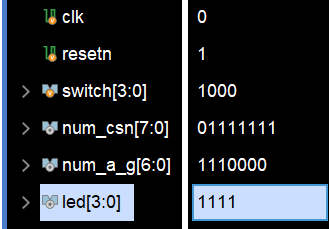




这时候num\_a\_g为1111110,代表0，led0000，为之前的值f。



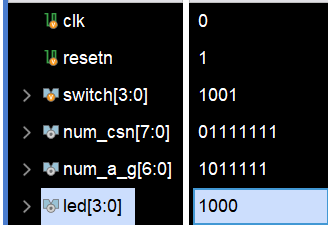
接下来switch为8，那么输入的值应该是8取反，也就是7。



这时候num\_a\_g为1110000,代表7，led1111，为之前的值0。



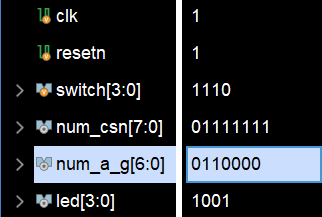
接下来switch为9，那么输入的值应该是9取反，也就是6。



这时候num\_a\_g为1011111,代表6，led1000，为之前的值7。



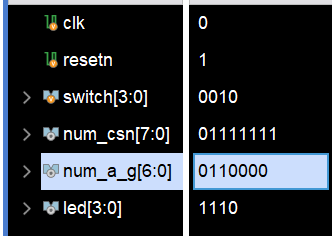
接下来switch为e，那么输入的值应该是e取反，也就是1。



这时候num\_a\_g为0110000,代表1，led1001，为之前的值6。



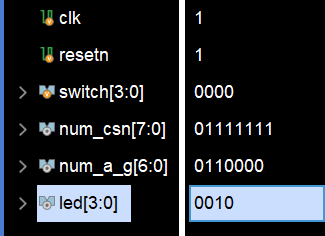
接下来switch为2，那么输入的值应该是2取反，也就是d。



由于d大于9，因此这时候num\_a\_g为0110000,代表1，不变，led1110，为之前的值1。



接下来switch为0，那么输入的值应该是0取反，也就是f。



由于f大于9，因此这时候num\_a\_g为0110000,代表1，不变，led1101，为之前的值d。

综上，验证成功，代码功能正确。

接下来使用实验箱来验证。

首先按下复位键：



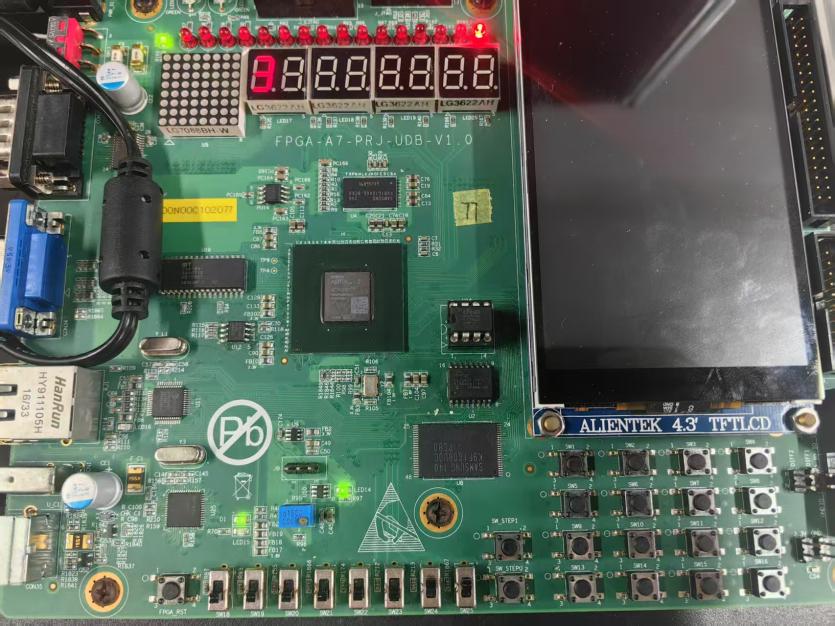
这时候led灯不亮，数码屏显示为0.

接下来拨码调为1：



可以看见，数码屏变成了1，而led灯显示上一个数字0。

接下来拨码调为3：



可以看见，数码屏变成了3，而led灯显示上一个数字1。

接下来拨码调为7：



可以看见，数码屏变成了7，而led灯显示上一个数字3。

接下来我把四个拨码全部拨到上边，代表15，大于9：



可以看见，此时的数码屏显示的还是之前的7，而led灯也变成了之前的数字7。

接下来我把第一个拨码拨下去，代表数字14，还是大于9：



因为14仍然大于9，所以数码屏幕依旧不变，还是7，而led灯显示上一个数据15。

综上所述，在实验箱上验证的功能也正常。

## 总结感想

### 任务一：寄存器堆仿真实验

为什么寄存器堆要设计为两读一写？

主要是为了提高cpu的运行效率，因为在MIPS体系下，对于add，sub等操作指令往往涉及到两个源寄存器和一个目标寄存器，我们需要从两个源寄存器里边读取到我们需要的数据，经过相应运算之后把结果写入到一个目标寄存器里边，这样就需要寄存器堆可以同时读取两个寄存器的值，并且向一个寄存器写入结果，这样可以提高cpu的处理效率。如果只能一读的话，寄存器堆需要读取两次，降低了速度，但如果设计更多的读取接口的话，会增大成本，让电路太复杂，得不偿失。

### 任务二：同步RAM与异步RAM仿真

同步RAM与异步RAM的特点与不同。

同步RAM的特点：

首先，对于同步RAM来说，每一次的读写操作都必须在时钟的上升沿触发，以此来保持同步性，第二点，同步RAM具有默认的读后写的功能，即当使能端为1，进行写操作时，在时钟上升沿，同步RAM也会自动读取刚刚写入的值。这比较适合性能好，运行速度快的设备。

异步RAM的特点：

首先，对于异步RAM来说，它的每一次读写操作不需要等待时钟信号，而是随着使能端，地址等端口的变化而随时进行。并且，异步RAM并没有读后写的功能，读写操作严格按照使能端的信号进行。异步RAM比较适合慢速的设备，因为在读写时不需要等待时钟信号。

同步RAM与异步RAM的不同：

第一点，同步RAM的读写操作在时钟上升沿进行，而异步RAM是立即进行。第二点，同步RAM具有读后写的功能，而异步RAM没有。

### 任务三：数字逻辑电路的设计与调试

使用vivado的调试经验：

1. 调试时候最主要的方法就是看波形图，如果遇见Z波形，那么首先去模块里找wire类型的变量，看看它们有没有被正确赋值，并且要检查一下tb里边调用模块的时候有没有正确连接。
2. 遇见X波形的时候，就用看看是不是reg类型的量出了问题，有必要的时候可以在always模块里边进行一项一项的追溯。
3. 遇见波形停止的情况，这里就需要自己写代码的时候小心一点，利用vivado的实现功能去找出bug，搞清楚自己需要的逻辑，然后进行修改。
4. 遇见越沿采样的情况，这应该是发现波形图里部分数据一直错误，然后去一步一步追溯检查，然后才能发现。
5. 最后，如果是功能bug，那就需要中找出波形图里是哪一个数据出了问题，然后去模块里边进行追溯，找出问题。