实验四 存储器阵列设计

**一、实验目的**

1 掌握Verilog语言和Vivado、Logisim开发平台的使用；

2 掌握存储器和寄存器组的设计和测试方法。

**二、实验内容**

1 存储器设计与测试

2 寄存器组设计与测试

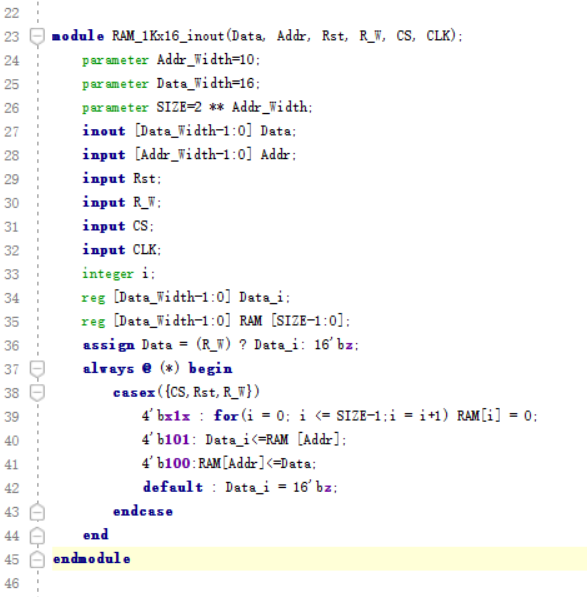
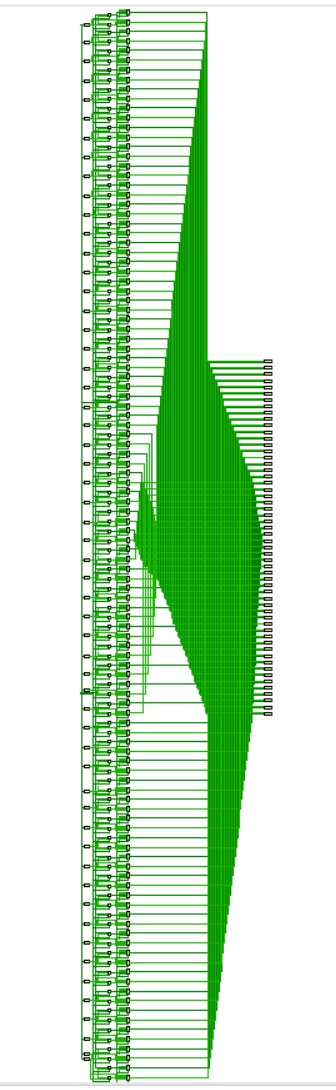
**三、实验要求**

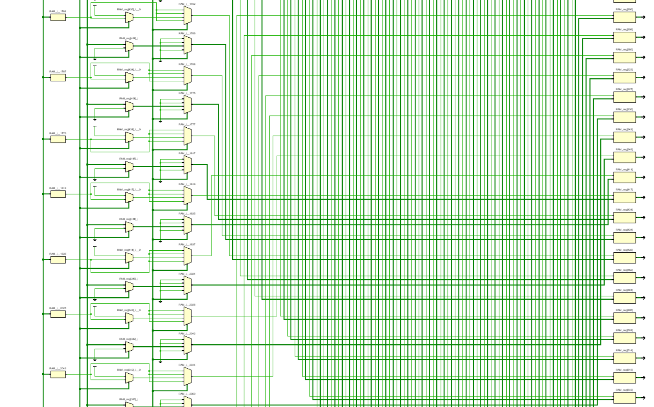
1 掌握Vivado或Logisim开发工具的使用，掌握以上电路的设计和测试方法；

2 记录设计和调试过程（Verilog代码/电路图/表达式/真值表，Vivado仿真结果，Logisim验证结果等）；

3 分析Vivado仿真波形/Logism验证结果，注重输入输出之间的对应关系。

**四、实验过程及分析**

存储器设计：



这Verilog模块实现了一个1K x 16位的双向RAM（Random Access Memory）。

Data\_i寄存器用于存储从RAM中读取的数据。

Data：双向端口，用于连接RAM的数据总线。

Addr：输入端口，用于指定RAM的读写地址。

Rst：输入端口，用于复位整个RAM，将存储单元清零。

R\_W：输入端口，用于控制RAM的读写操作。

CS：输入端口，用于片选，通过此信号控制RAM的操作是否生效。

CLK：输入端口，时钟信号。

使用状态机实现不同的操作：

当 CS, Rst, R\_W 组合为 4'bx1x 时，执行循环将整个RAM清零。

当 CS, Rst, R\_W 组合为 4'b101 时，将RAM中地址为 Addr 的数据赋给 Data\_i。

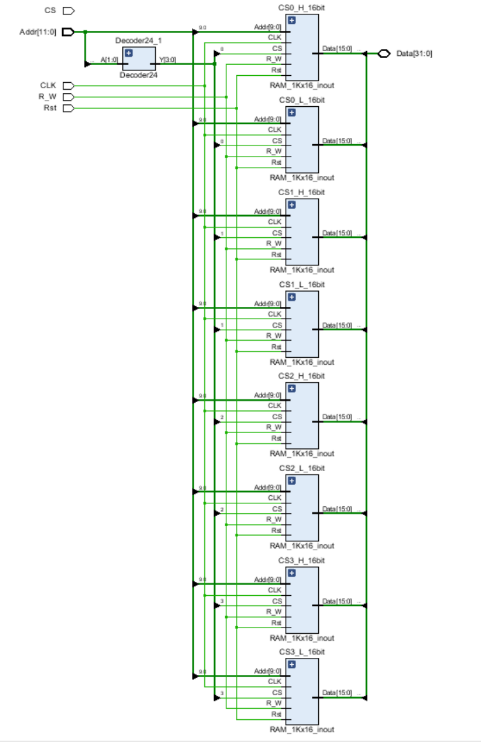
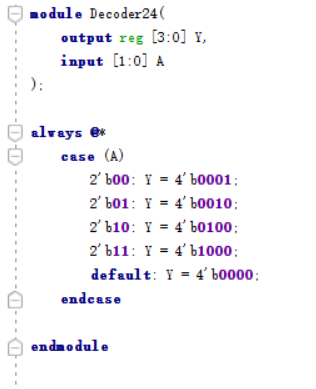
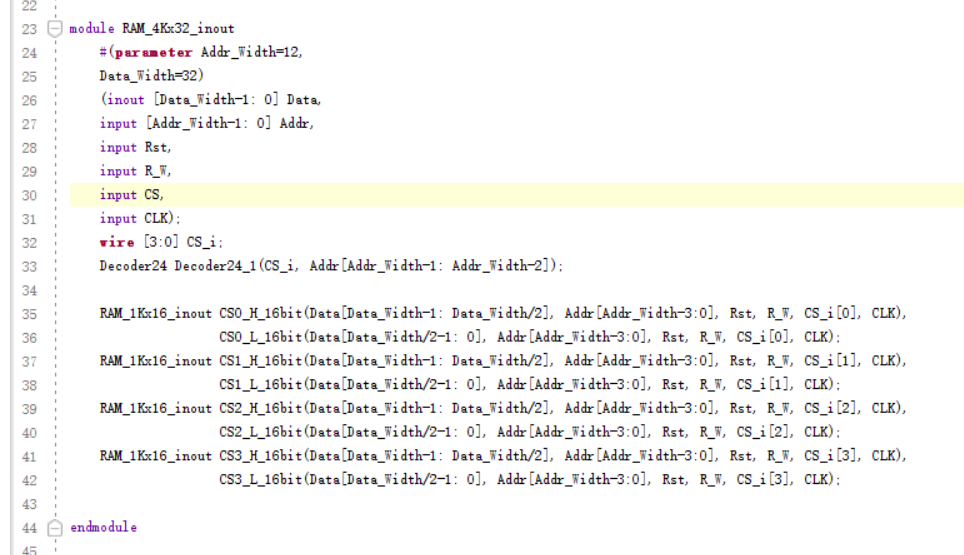
当 CS, Rst, R\_W 组合为 4'b100 时，将 Data 的值写入RAM中地址为 Addr 的位置。

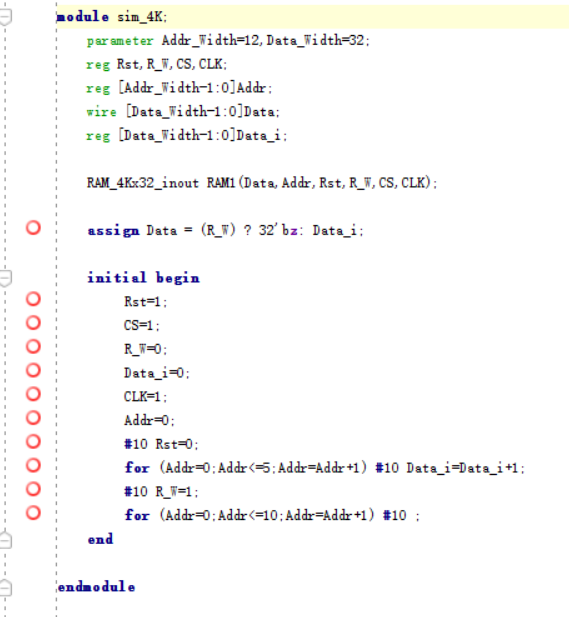
对于其他组合，将 Data\_i 置为高阻态。

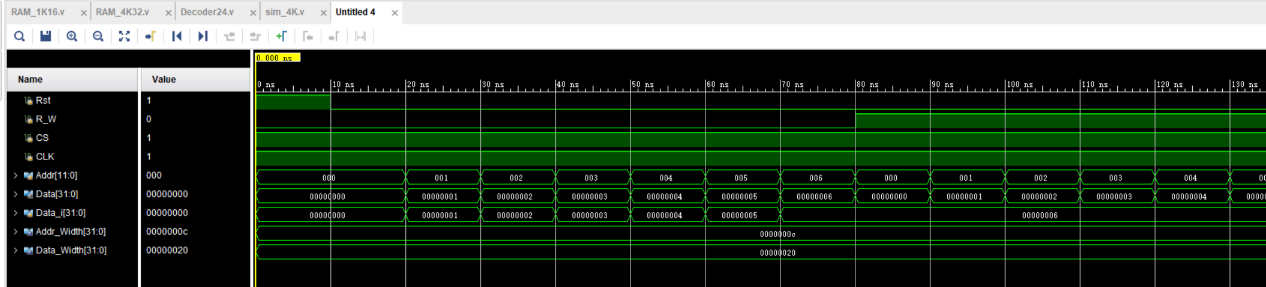
使用 assign Data = (R\_W) ? Data\_i : 16'bz; 根据读写控制信号 R\_W 的状态，将 Data\_i 的值赋给 Data（读操作），或者将 Data 置为高阻态（写操作）。

这个模块允许在指定的地址执行读写操作，可以通过 Rst 信号将整个RAM清零，同时支持片选信号 CS 控制是否执行RAM操作。

字扩展与位扩展：

****

****

****

Decoder24:

声明了一个Verilog模块，模块名为Decoder24，有两个端口：输出端口Y（4位寄存器）和输入端口A（2位寄存器）。

使用always @\*表示组合逻辑，根据输入A的值进行不同的解码操作：

当A为2'b00时，输出Y为4'b0001。

当A为2'b01时，输出Y为4'b0010。

当A为2'b10时，输出Y为4'b0100。

当A为2'b11时，输出Y为4'b1000。

对于其他情况，使用default语句将输出Y置为4'b0000。

总结：

这个模块实现了一个2-4译码器，将2位的输入编码为4位的输出。根据不同的输入值，输出的4位二进制数Y有不同的值。此模块适用于数字电路设计中的译码器应用，可以将输入的2位二进制数映射到4位输出。

**RAM\_4K32:**

声明了一个Verilog模块，模块名为RAM\_4Kx32\_inout，有六个端口，包括数据端口Data、地址端口Addr、复位端口Rst、读写控制端口R\_W、片选端口CS和时钟端口CLK。此外，通过参数Addr\_Width和Data\_Width指定了地址宽度和数据宽度。

通过实例化Decoder24模块，将Addr的高两位用于片选信号CS\_i的解码。

实例化了八个RAM\_1Kx16\_inout模块，用于处理每个片选信号对应的数据。每个模块接收相应的高位和低位的数据，并使用相应的CS\_i信号，通过相同的时钟信号CLK进行操作。

这个模块实现了一个4K x 32位的双向RAM，通过实例化八个1K x 16位的RAM子模块，通过Decoder24将地址的高两位解码成四个片选信号。每个子模块对应一个片选信号，用于处理相应的数据。这样的设计使得4K x 32位的RAM可以分为八个1K x 16位的RAM，并且每个部分可以独立地进行读写操作。

**Sim\_4K:**

声明了一个Verilog模块，模块名为sim\_4K，并定义了两个参数：Addr\_Width为12，Data\_Width为32。

声明了一些寄存器和线网，其中Rst、R\_W、CS、CLK、Addr为输入端口，Data为输出端口，Data\_i为内部寄存器。

实例化了一个RAM\_4Kx32\_inout模块，通过端口连接与sim\_4K模块的相应信号和数据。

根据读写控制信号R\_W的状态，将Data连接到Data\_i或置为高阻态。

在仿真开始时，进行了一系列的初始化操作：

Rst被置为1，表示复位。

CS被置为1，表示不选中RAM。

R\_W被置为0，表示写入操作。

Data\_i被置为0，作为写入数据的起始值。

CLK被置为1，表示时钟信号。

Addr被置为0，作为起始地址。

然后，通过#10的延时，进行了一些状态的切换：

Rst被置为0，表示结束复位。

使用for循环，从地址0开始，每隔10个时间单位，Data\_i增加1，总共执行6次。

R\_W被置为1，表示读取操作。

使用for循环，从地址0开始，每隔10个时间单位，执行了11次。

总结：

这个模块实现了对RAM\_4Kx32\_inout模块的简单仿真。在初始化过程中，进行了一系列的状态设置和数据写入操作，以模拟RAM的读写过程。

时序输出关系：

Rst在初始时为1，表示RAM处于复位状态，此时不进行读写操作。

CS在初始时为1，表示不选中RAM，也就是RAM不响应读写操作。

R\_W在初始时为0，表示写入操作。在for循环中，每隔10个时间单位，Data\_i递增，相应地写入到RAM中。这模拟了对RAM的写入过程。

在R\_W被置为1后，表示读取操作。在for循环中，每隔10个时间单位，没有进行实际的写入或读取操作，因为for循环中没有相关的操作。这阶段用于观察读取操作的影响。

**综上所述，仿真的时序输出关系为：**

**初始时，RAM被置为复位状态，不进行读写操作。**

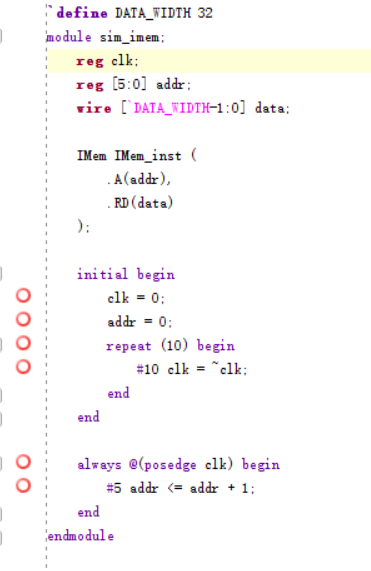
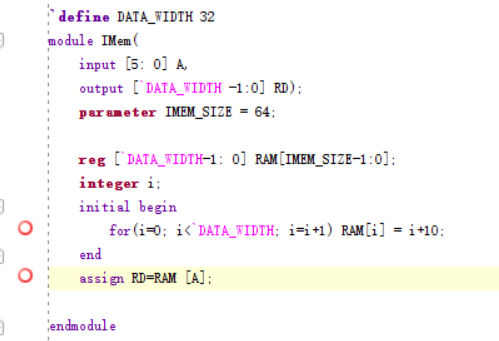
**随后，进行了一系列的写入操作，模拟了写入过程。**

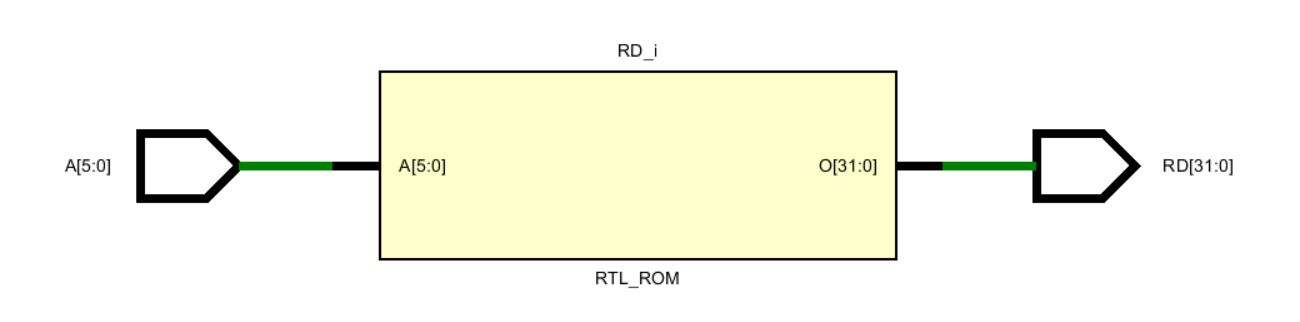
**最后，进行了一系列的读取操作，模拟了读取过程。**

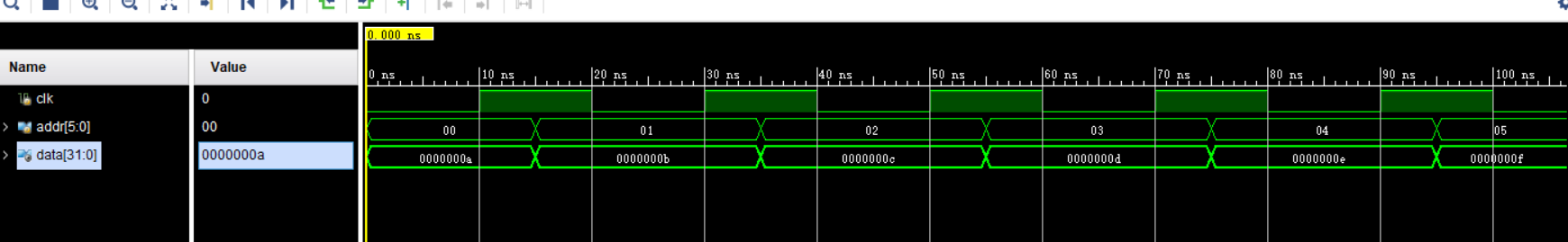
**可以看到时序图所示：**

**地址addr在循环递增，一开始R\_W置于写入状态，向data中写入数据，后来R\_W置于读取状态，data读出前面写入的addr处的数值，而data\_i保持不变。**

**指令存储器设计：**

****

****

****

这段 Verilog 代码定义了一个指令存储器模块 IMem

模块声明：模块名为 IMem，有一个 6 位输入端口 A 用于指定读取的存储器地址，和一个 32 位输出端口 RD 用于输出从存储器读取的数据。

参数声明：定义了参数 IMEM\_SIZE，表示存储器的大小为 64 个 32 位的存储单元。

内部存储器声明：声明了一个名为 RAM 的内部存储器，是一个大小为 IMEM\_SIZE 的数组，每个元素是 32 位宽度。

初始化块：在仿真开始时，使用 for 循环，将 RAM 的前 DATA\_WIDTH 个元素初始化为递增的值，即 RAM[0] 初始化为 10，RAM[1] 初始化为 11，以此类推。

数据输出：通过 assign 语句，将存储器中地址为 A 的数据赋给输出端口 RD。这表示在每个时钟周期，从指定地址读取存储器中的数据。

这个模块的作用是模拟一个简单的指令存储器，可以根据输入的地址 A 从存储器中读取相应的 32 位指令，并将其输出到 RD 端口。在初始化时，存储器的前 DATA\_WIDTH 个位置被初始化为递增的值。

定义了一个名为 sim\_imem 的仿真模块，包含了一个时钟信号 clk、一个6位宽的地址信号 addr 以及一个32位宽的数据线 data。实例化了 IMem 模块，并将其输入输出端口与仿真模块的信号相连接。

在仿真开始时，初始化了时钟信号 clk 为 0，地址信号 addr 为 0。然后通过 repeat 循环，每次循环都在时钟上升沿（每隔10个时间单位）切换 clk 的状态，即从 0 切换到 1 或从 1 切换到 0。这样，通过 clk 的变化来模拟时钟信号。

在时钟上升沿发生时，addr 的值递增 1。通过 #5 的延时，表示在下一个时钟上升沿前进行这个操作。这段代码模拟了在时钟上升沿时不断递增地址的行为。

**总结时序输出关系：**

**初始时，clk 为 0，addr 为 0。**

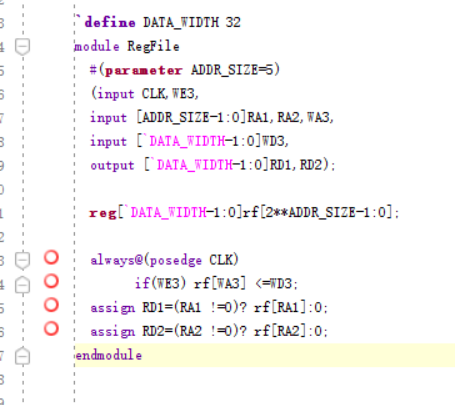
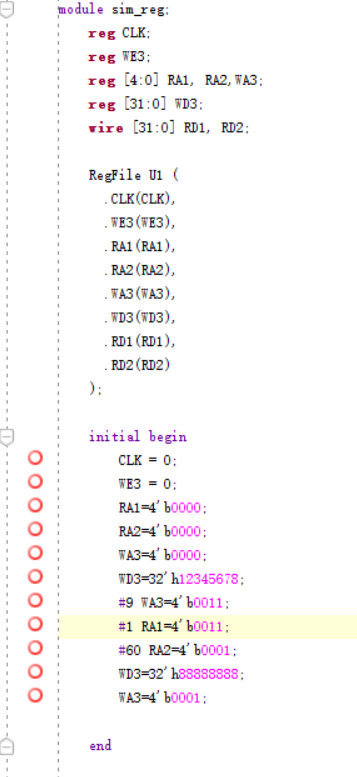
**通过 repeat 循环，clk 在每个循环中发生状态切换（0->1或1->0）。**

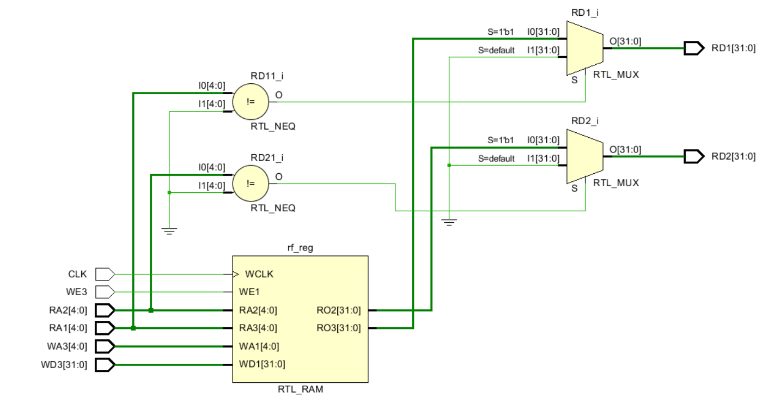
**在每个时钟上升沿，addr 递增 1。**

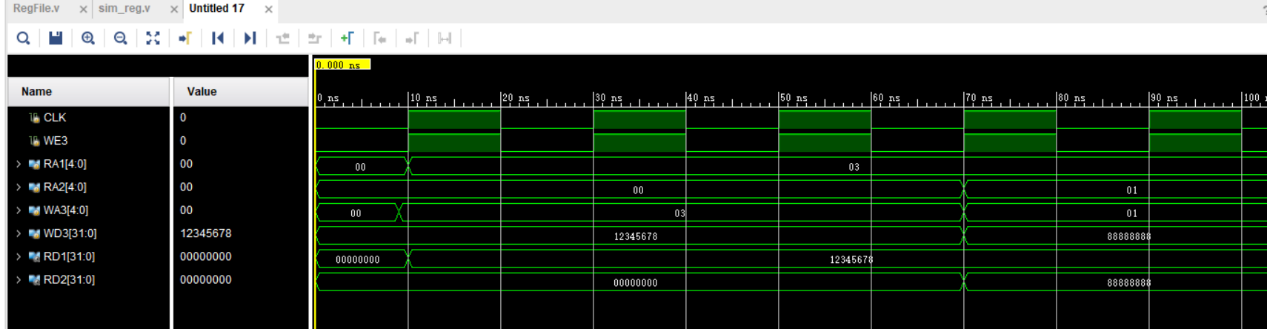
**通过 IMem 模块实例，data 会根据 addr 的值从指令存储器中读取相应的 32 位数据。**

**由时序图可以看出每过15ns，会从ram中读出一个addr地址处的数据，在design文件中可以看出ram初始化的赋值，所以从a开始输出，一直递增。**

**寄存器文件设计：**

****

****

****

声明了一个名为 RegFile 的 Verilog 模块，其中包含了一系列输入和输出端口。CLK 是时钟信号，WE3 是写使能信号，RA1 和 RA2 是两个读地址输入，WA3 是写地址输入，WD3 是写数据输入，RD1 和 RD2 是两个读数据输出。

使用 parameter 关键字声明了一个参数 ADDR\_SIZE，默认值为 5。这个参数指定了地址的位宽。

声明了一个名为 rf 的内部寄存器数组，每个寄存器是 DATA\_WIDTH 位宽，数组的大小为 2^ADDR\_SIZE。这实现了一个简单的寄存器文件。

在时钟的上升沿，如果写使能信号 WE3 为 1，将写数据 WD3 写入到寄存器文件的相应写地址 WA3 处。

通过 assign 语句，实现了读数据的输出逻辑。如果读地址 RA1 不为 0，则将寄存器文件中的相应位置的数据赋给 RD1，否则 RD1 被赋值为 0。同样的逻辑应用在 RD2 上。

总结：

这个模块实现了一个带有写使能的寄存器文件，可以进行读和写操作。在每个时钟的上升沿，如果写使能信号为 1，将写数据写入到指定的写地址。读操作通过比较读地址是否为 0 来确定是否进行读取。

**仿真：**

实例化了 RegFile 模块，并将其输入输出端口连接到仿真模块的信号。CLK 是时钟信号，WE3 是写使能信号，RA1 和 RA2 是两个读地址输入，WA3 是写地址输入，WD3 是写数据输入，RD1 和 RD2 是两个读数据输出。

在仿真开始时，初始化了时钟信号 CLK 为 0，写使能信号 WE3 为 0，读写地址和写数据信号也被初始化。然后在仿真过程中，通过 # 符号的延时，改变了各个信号的值，模拟了时钟周期内的变化。

在仿真的每个10个时间单位的时钟周期内，时钟信号 CLK 发生状态切换（0->1或1->0）。这个语句模拟了时钟的边沿生成。

**时序输出关系：**

**时钟信号 CLK：在每个10个时间单位的时钟周期内，时钟信号 CLK 发生状态切换。**

**写使能信号 WE3：在每个10个时间单位的时钟周期内，写使能信号 WE3 发生状态切换。**

**读写地址和写数据信号：在仿真开始时，读写地址和写数据信号被初始化。在仿真过程中，通过延时和指定的时间序列，这些信号的值被改变，模拟了读写地址和写数据的变化。**

**读数据信号 RD1 和 RD2：在仿真过程中，通过 RegFile 模块实例进行读操作，读数据信号 RD1 和 RD2 的值会随着读地址的变化而改变。在 #1 RA1 = 4'b0011; 处，RA1 的值变为 4'b0011，因此 RD1 的值将对应于寄存器文件中相应位置的数据。在 #60 RA2 = 4'b0001; 处，RA2 的值变为 4'b0001，因此 RD2 的值将对应于寄存器文件中相应位置的数据。**

**写数据信号 WD3：在仿真过程中，通过 WD3 = 32'h88888888; 和 WA3 = 4'b0001; 的操作，将写数据信号 WD3 写入到寄存器文件的相应写地址 WA3 处。**

**五、调试和心得体会**

在本次计算机组成实验中，我深入学习了硬件描述语言（Verilog）的基本概念和用法，并通过设计和仿真实际的硬件电路，深入理解了计算机组成原理的一些关键概念。

首先，我了解了Verilog语言的基础知识，包括模块化设计、数据类型、参数化、内部存储器的使用等。Verilog的模块化设计允许我将电路划分为不同的模块，提高了代码的可读性和可维护性。

其次，通过实现RAM模块和寄存器文件模块，我加深了对存储器和寄存器的理解。在RAM模块中，我学会了如何使用Verilog描述存储器的读写过程，以及如何使用casex语句处理不同的控制信号。在寄存器文件模块中，我了解了时序逻辑的实现，以及如何通过时钟信号和写使能信号进行同步写入操作。

实验中，我还学到了如何编写测试文件进行仿真。通过对各个模块的功能进行测试，我能够验证设计的正确性，发现潜在的问题并进行调试。

最后，在整个实验的过程中，我深刻体会到硬件设计的精确性和时序的重要性。每一个模块都需要按照时序要求正确地执行，否则可能导致电路不稳定或不正确的结果。同时，我对计算机组成原理的知识有了更深层次的理解，通过实际设计和仿真，我更好地掌握了理论知识的应用。

总体而言，这次计算机组成实验为我提供了一个很好的机会，让我在实际操作中学到了很多关于硬件设计和Verilog语言的知识。这不仅增强了我对计算机组成原理的理解，也提升了我的实际问题解决能力。通过这次实验，我对硬件描述语言和计算机硬件设计有了更深刻的认识，这对我的学术和职业发展都具有重要的意义。