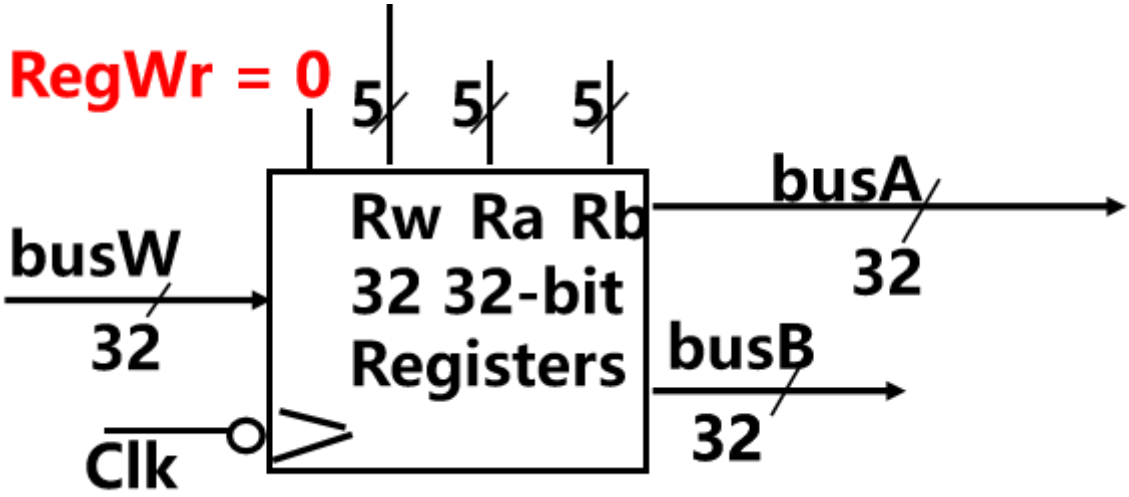
实验一：存储器与寄存器组

设计思路

寄存器组



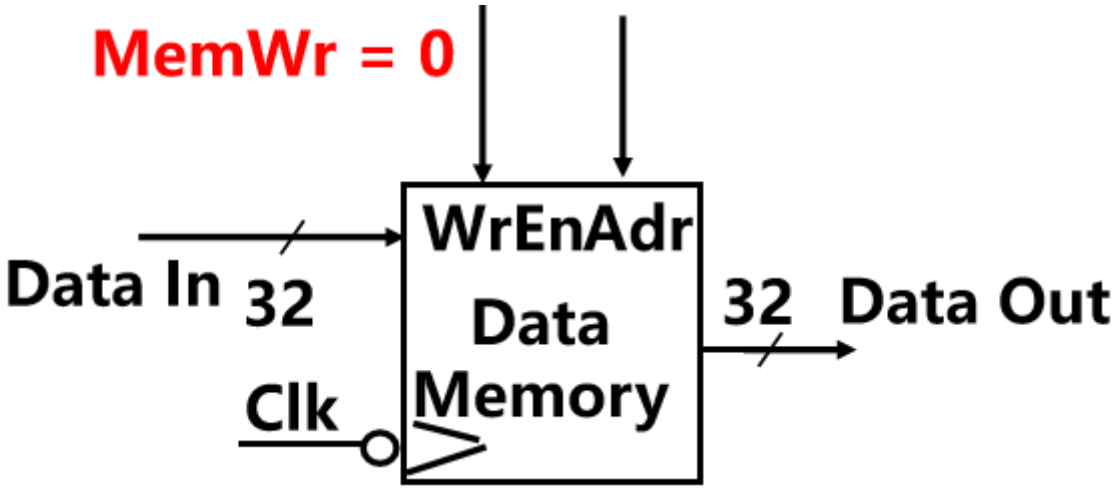
上图所示的为一个通用寄存器组，每个寄存器地址是一个5位的二进制编码，它有两个读口：busA和busB，分别由Ra和Rb给出地址，读操作属于组合逻辑操作，无需时钟信号控制，当地址Ra和Rb到达后，经过一个"取数时间"的延迟，在busA和busB上的信息开始有效。它还有一个写口：busW上的信息写入的地址由Rw指定。写操作属于时序逻辑操作，需要时钟信号CLK的控制，RegWr=1时，时钟触发边沿到来后经过clk-to-Q时间延迟，从busW传来的值开始写入Rw指定的寄存器中。

1. **模块声明：**
   - 模块名：`Register_`
   - 输入信号：
     - `BusW`：用于写入的32位数据总线。
     - `Clk`：时钟信号。
     - `RegWr`：写入使能信号。
     - `Rw`、`Ra`、`Rb`：用于选择寄存器的写入地址、读取地址A和读取地址B。
   - 输出信号：
     - `BusA`、`BusB`：用于输出的32位数据总线。
2. **内部信号：**
   - `registers[31:0]`：32个32位寄存器的数组。
3. **初始化过程：**
   - 使用 `initial` 块，在模拟开始时对 `registers` 数组进行初始化。

- 所有寄存器初始化为32位的0。

4. **数据输出：**
   - 使用 `assign` 语句，根据读取地址 `Ra` 和 `Rb`，将 `registers` 中的数据赋给输出总线 `BusA` 和 `BusB`。
5. **时序逻辑：**
   - 使用 `always @(negedge Clk)` 块，表示在时钟的下降沿进行操作。
   - 在时钟下降沿，检查写使能信号 `RegWr` 是否为1，如果是，将输入总线 `BusW` 的数据写入到选定的寄存器地址 `Rw` 中。

存储器



理想存储器有一个32位数据输入端DataIn，一个32位数据输出端DataOut，还有一个读写公用的地址输入端Address，控制信号有一个写使能信号WE，写操作受时钟信号CLK的控制，假定采用下降沿触发，即在时钟下降沿开始写入信息。读操作是组合逻辑操作，在地址有效后，经过一个“取数时间",数据输出端DataOut上数据有效；写操作时时序逻辑操作，即在WE=1的情况下，当时钟CLK边沿到来时，DataIn开始写入存储单元中。

1. **模块声明：**
   - 模块名：`Cache_`
   - 输入信号：
     - `clk`：时钟信号。
     - `addr`：8位地址总线，用于访问内存。
     - `data_in`：32位数据输入总线，用于写入内存。
     - `write_enable`：写使能信号。
   - 输出信号：
     - `data_out`：32位数据输出总线，用于读取内存。
2. **内部信号：**
   - `memory[0:255]`：256个32位寄存器的数组，模拟 Cache 内存。

3. **初始化过程：**
   - 使用 `initial` 块，在模拟开始时对 `memory` 数组进行初始化。
   - 所有寄存器初始化为32位的0。
4. **写操作过程：**
   - 使用 `always @(negedge clk)` 块，表示在时钟的下降沿进行操作。
   - 检查写使能信号 `write_enable` 是否为1。
   - 如果写使能信号为1，将输入总线 `data_in` 的数据写入到地址总线 `addr` 指定的内存位置。
5. **读操作过程：**
   - 使用 `assign` 语句，根据地址总线 `addr`，将 `memory` 中的数据赋给输出总线 `data_out`。

由于 Cache 是直接从内存中读取数据，没有明确的读使能信号，因此使用 `assign` 语句进行实现。

代码展示

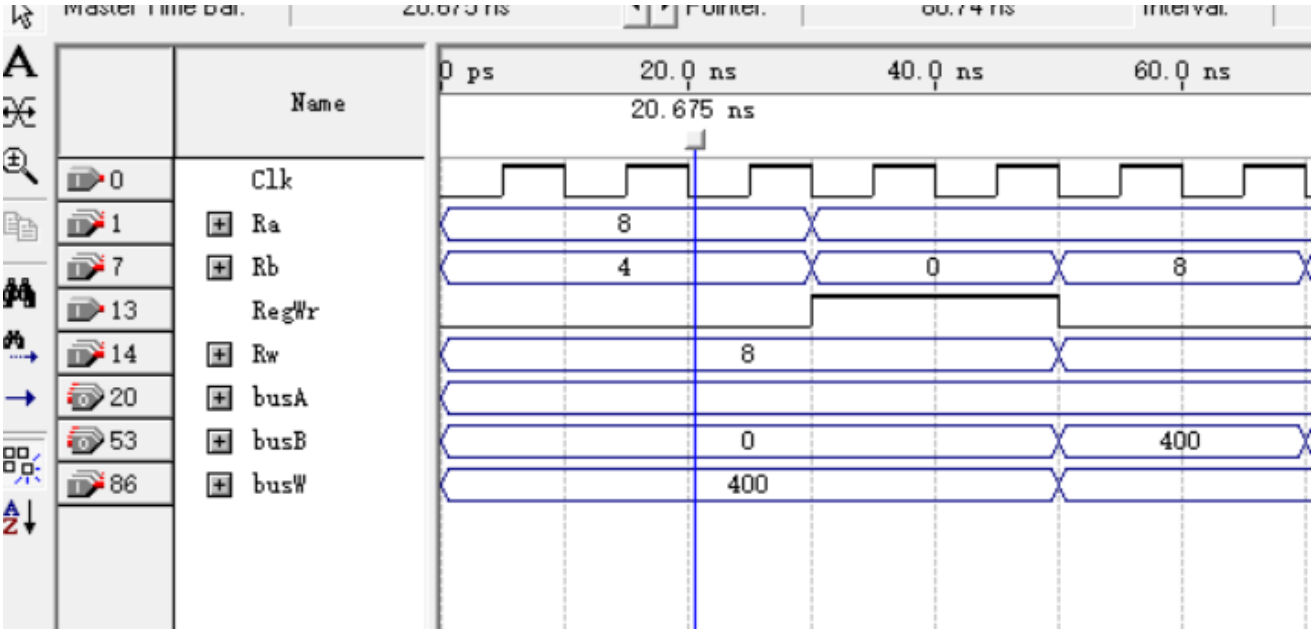
```
module Register_(
    input wire [4:0] Rw,     // 读取寄存器的地址
    input wire [4:0] Ra,     // 读取寄存器A的地址
    input wire [4:0] Rb,     // 读取寄存器B的地址
    input wire [31:0] BusW, // 写入数据
    input wire RegWr,        // 写入使能信号
    input wire Clk,          // 时钟信号

    output wire [31:0] BusA, // 从寄存器A读取的数据
    output wire [31:0] BusB  // 从寄存器B读取的数据
);

reg [31:0] registers [31:0]; // 32个32位寄存器

initial begin
    // 初始化所有寄存器为零
    integer i;
    for (i = 0; i < 32; i = i + 1) begin
        registers[i] <= 32'h00000000;
    end
end


always @(negedge Clk) begin
    if (RegWr) begin
        registers[Rw] <= BusW;
```

```
    end
end

assign BusA = registers[Ra];
assign BusB = registers[Rb];

endmodule
```

```
module Cache_ (
    input wire clk,         // Clock signal
    input wire [7:0] addr, // Address input (8-bit address for a 256 x 32-bit
RAM)
    input wire [31:0] data_in, // Data input (32 bits)
    input wire write_enable, // Write enable control
    output wire [31:0] data_out // Data output
);

parameter MEM_DEPTH = 256; // Memory depth (256 words)
reg [31:0] memory [0:MEM_DEPTH-1];

initial begin
    integer i;
    for (i = 0; i < MEM_DEPTH; i = i + 1) begin
        memory[i] <= 32'h00000000;
    end
end

always @(negedge clk) begin
    if (write_enable) begin
        // Write operation
        memory[addr] <= data_in;
    end
end

assign data_out = memory[addr];
endmodule
```

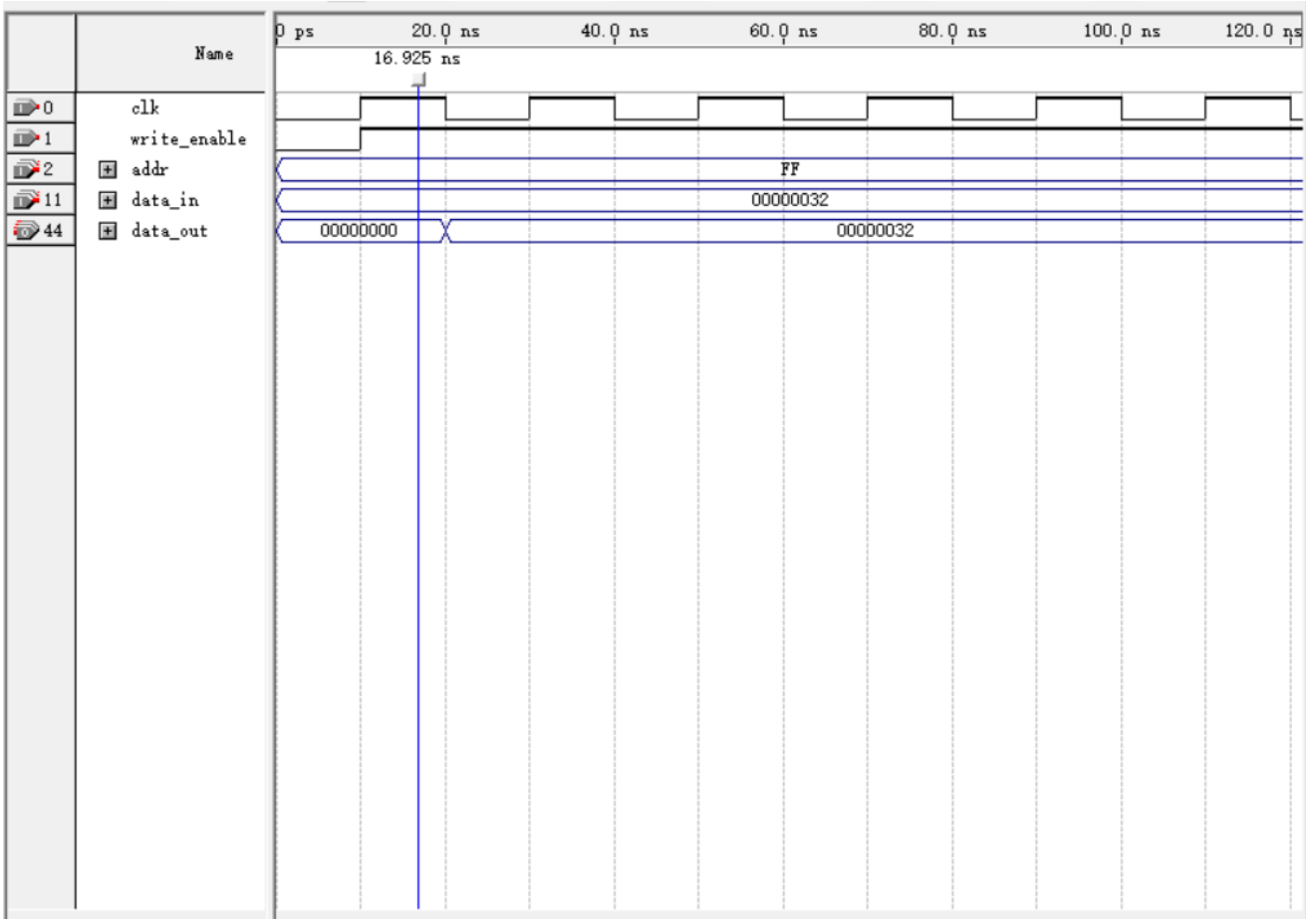
波形分析

寄存器组



如图所示，刚开始写使能RegWr=0，不能写入，此时读出的busA和busB都为初始值0，在RegWr=1后，registers[Rw]=busW，即registers[8]=400,之后在地址为8的寄存器中读出busB为400，写操作、读操作正确。

存储器：



上图所示，时钟下降沿到来之前，读操作data_out的值为初始值0，下降沿到来之后，对地址为0xFF的存储器进行写入0x32，之后再读出地址为0xFF的存储器的值后data_out为0x32。写操作、读操作正确。