Register File Module Explained

Purpose

The **register file** is a crucial component of any CPU. In RISC-V, the register file consists of 32 general-purpose, 32-bit registers (`x0` to `x31`). It provides operands to instructions and stores results. This module supports:

- **Two simultaneous asynchronous register reads**
- **One synchronous register write** per clock cycle

Port	Direction	Width	Description
clk	input	1 bit	Clock signal for synchronous writes
we	input	1 bit	Write enable signal
rs1	input	5 bits	Read register 1 address (0–31)
rs2	input	5 bits	Read register 2 address (0–31)
rd	input	5 bits	Write register address (0–31)
wdata	input	32 bits	Data to write to register `rd`
rdata1	output	32 bits	Data read from register `rs1`
rdata2	output	32 bits	Data read from register `rs2`

Register Array Declaration

verilog

```
logic [31:0] regs [31:0];
```

- Declares an array of 32 registers, each 32 bits wide.

Asynchronous Read Logic

verilog

```
assign rdata1 = (rs1==0) ? 0 : regs[rs1];
assign rdata2 = (rs2==0) ? 0 : regs[rs2];
```

- **Reads are asynchronous**: Outputs (`rdata1`, `rdata2`) reflect the value of the addressed register instantly, without a clock edge.
- **Register x0 is hardwired to zero**: RISC-V specifies that register `x0` is always zero and cannot be written to. If either `rs1` or `rs2` is `0`, the output is forced to zero.

Synchronous Write Logic

verilog

```
always_ff @(posedge clk) begin
    if (we && rd!=0) regs[rd] <= wdata;
end
```

- **Writes are synchronous**: Data is written to the register file **only** on the rising edge of the clock.
- **Write enable (`we`)**: If `we` is 1 and `rd` is not zero, `wdata` is written to register `rd`.
- **Cannot write to register x0**: Any attempt to write to register 0 is ignored, preserving its constant zero value.

Diagramatic Representation

text

```
regfile
  +-------------------+
  |      x0 = 0       |
  |  x1 ... x31       |
  +-------------------+
    ↗               ↖
  rs1   ...     rs2     (async reads)
```

Context in CPU Pipeline

- **ID (Instruction Decode) Stage**:
  This module is used to fetch the source register values for the executing instruction.
- **WB (Write Back) Stage**:
  The result of an operation (from the EXE stage) is written back to the destination register here.

Key Points

- **Asynchronous Reads**: Allows quick supply of operands to the ALU without waiting for the clock.
- **Synchronous Writes**: Ensures orderly update of register values in sync with the pipeline.
- **x0 Behavior**: RISC-V architectural requirement: `x0` always reads as zero, cannot be overwritten.
- **Multi-porting**: Two independent read addresses, one write address (common in RISC architectures for performance).