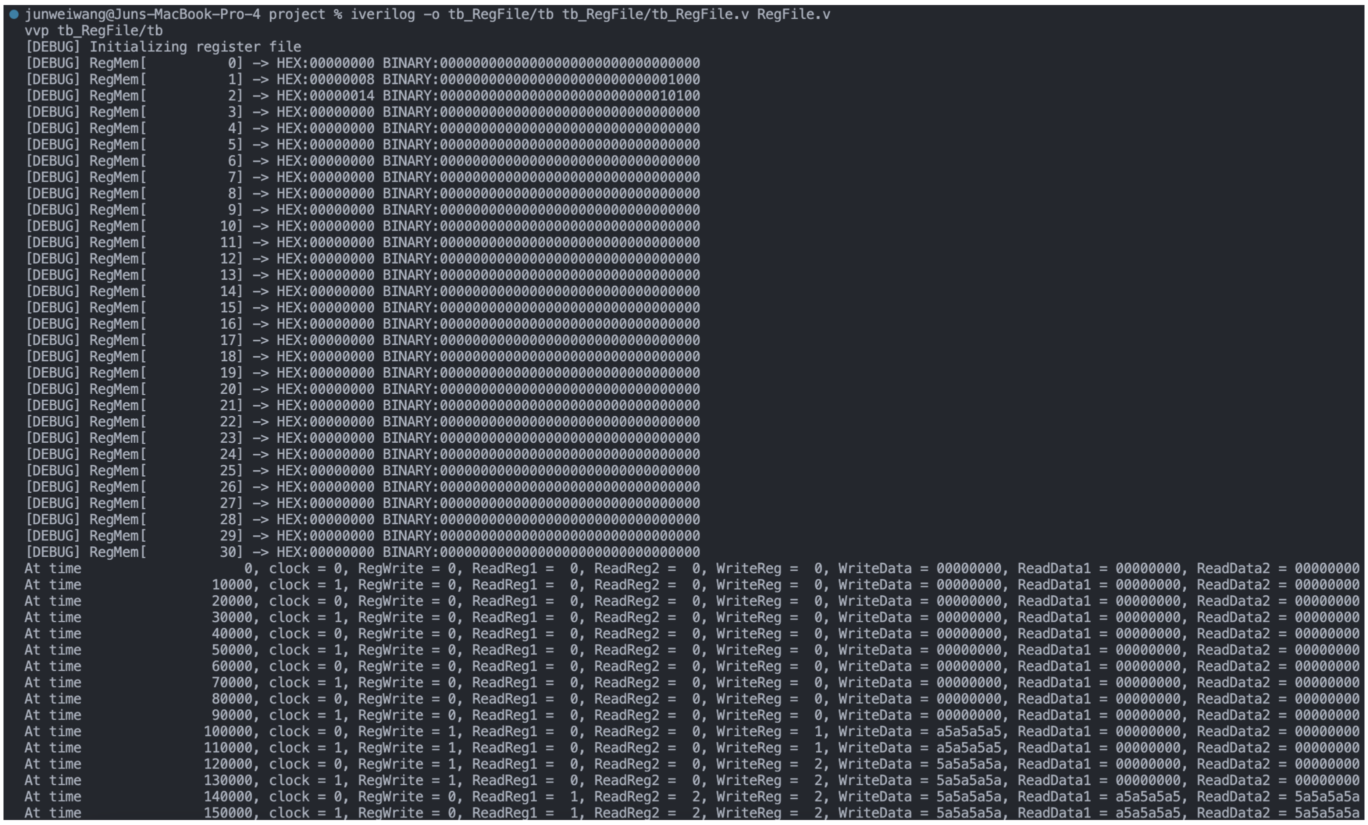
PC (Program Counter)

```
iverilog -o tb_RegFile/tb tb_RegFile/tb_RegFile.v RegFile.v
vvp tb_RegFile/tb

# open gtkwave to view the waveform
gtkwave tb_RegFile/tb_reg_file.vcd &
```

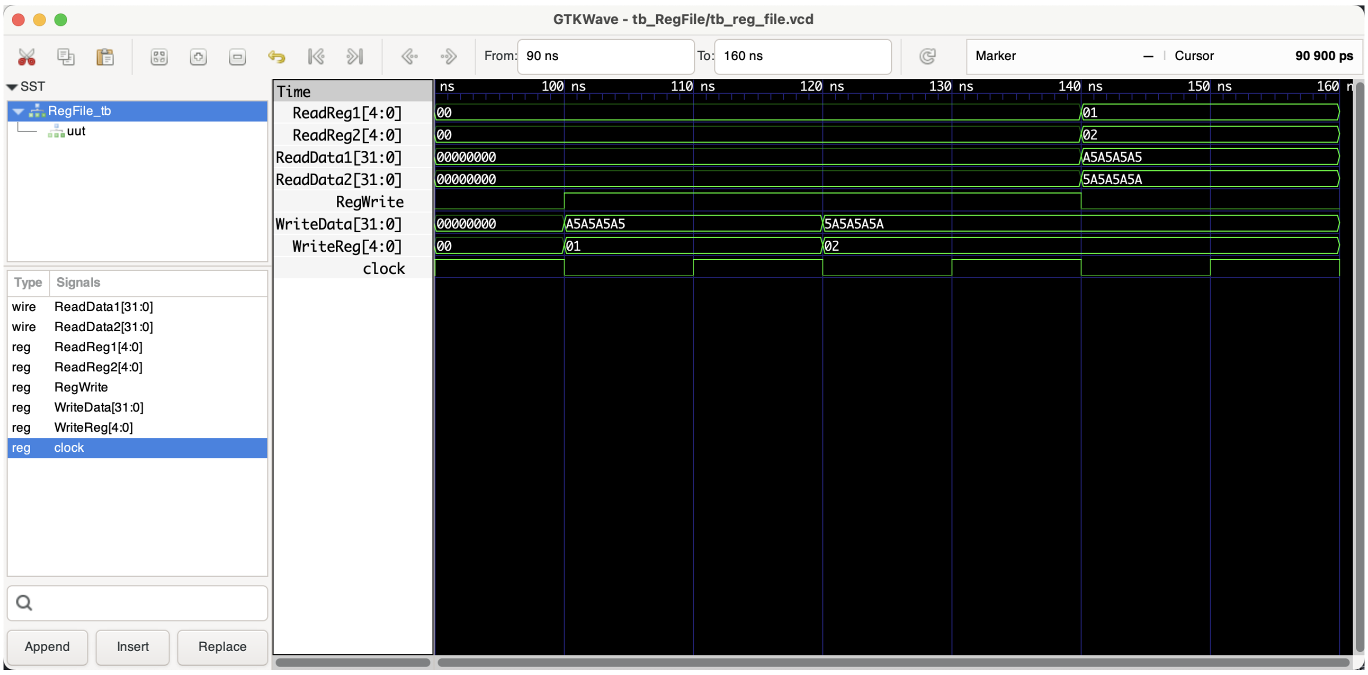
Benchmark (verifying the correctness)

Shell output



```
junweiwang@Juns-MacBook-Pro-4 project % iverilog -o tb_RegFile/tb tb_RegFile/tb_RegFile.v RegFile.v
vvp tb_RegFile/tb
[DEBUG] Initializing register file
[DEBUG] RegMem[          0] -> HEX:00000000 BINARY:00000000000000000000000000000000
[DEBUG] RegMem[          1] -> HEX:00000008 BINARY:00000000000000000000000000001000
[DEBUG] RegMem[          2] -> HEX:00000014 BINARY:00000000000000000000000000010100
[DEBUG] RegMem[          3] -> HEX:00000000 BINARY:00000000000000000000000000000000
[DEBUG] RegMem[          4] -> HEX:00000000 BINARY:00000000000000000000000000000000
[DEBUG] RegMem[          5] -> HEX:00000000 BINARY:00000000000000000000000000000000
[DEBUG] RegMem[          6] -> HEX:00000000 BINARY:00000000000000000000000000000000
[DEBUG] RegMem[          7] -> HEX:00000000 BINARY:00000000000000000000000000000000
[DEBUG] RegMem[          8] -> HEX:00000000 BINARY:00000000000000000000000000000000
[DEBUG] RegMem[          9] -> HEX:00000000 BINARY:00000000000000000000000000000000
[DEBUG] RegMem[         10] -> HEX:00000000 BINARY:00000000000000000000000000000000
[DEBUG] RegMem[         11] -> HEX:00000000 BINARY:00000000000000000000000000000000
[DEBUG] RegMem[         12] -> HEX:00000000 BINARY:00000000000000000000000000000000
[DEBUG] RegMem[         13] -> HEX:00000000 BINARY:00000000000000000000000000000000
[DEBUG] RegMem[         14] -> HEX:00000000 BINARY:00000000000000000000000000000000
[DEBUG] RegMem[         15] -> HEX:00000000 BINARY:00000000000000000000000000000000
[DEBUG] RegMem[         16] -> HEX:00000000 BINARY:00000000000000000000000000000000
[DEBUG] RegMem[         17] -> HEX:00000000 BINARY:00000000000000000000000000000000
[DEBUG] RegMem[         18] -> HEX:00000000 BINARY:00000000000000000000000000000000
[DEBUG] RegMem[         19] -> HEX:00000000 BINARY:00000000000000000000000000000000
[DEBUG] RegMem[         20] -> HEX:00000000 BINARY:00000000000000000000000000000000
[DEBUG] RegMem[         21] -> HEX:00000000 BINARY:00000000000000000000000000000000
[DEBUG] RegMem[         22] -> HEX:00000000 BINARY:00000000000000000000000000000000
[DEBUG] RegMem[         23] -> HEX:00000000 BINARY:00000000000000000000000000000000
[DEBUG] RegMem[         24] -> HEX:00000000 BINARY:00000000000000000000000000000000
[DEBUG] RegMem[         25] -> HEX:00000000 BINARY:00000000000000000000000000000000
[DEBUG] RegMem[         26] -> HEX:00000000 BINARY:00000000000000000000000000000000
[DEBUG] RegMem[         27] -> HEX:00000000 BINARY:00000000000000000000000000000000
[DEBUG] RegMem[         28] -> HEX:00000000 BINARY:00000000000000000000000000000000
[DEBUG] RegMem[         29] -> HEX:00000000 BINARY:00000000000000000000000000000000
[DEBUG] RegMem[         30] -> HEX:00000000 BINARY:00000000000000000000000000000000
At time                    0, clock = 0, RegWrite = 0, ReadReg1 =  0, ReadReg2 =  0, WriteReg =  0, WriteData = 00000000, ReadData1 = 00000000, ReadData2 = 00000000
At time                10000, clock = 1, RegWrite = 0, ReadReg1 =  0, ReadReg2 =  0, WriteReg =  0, WriteData = 00000000, ReadData1 = 00000000, ReadData2 = 00000000
At time                20000, clock = 0, RegWrite = 0, ReadReg1 =  0, ReadReg2 =  0, WriteReg =  0, WriteData = 00000000, ReadData1 = 00000000, ReadData2 = 00000000
At time                30000, clock = 1, RegWrite = 0, ReadReg1 =  0, ReadReg2 =  0, WriteReg =  0, WriteData = 00000000, ReadData1 = 00000000, ReadData2 = 00000000
At time                40000, clock = 0, RegWrite = 0, ReadReg1 =  0, ReadReg2 =  0, WriteReg =  0, WriteData = 00000000, ReadData1 = 00000000, ReadData2 = 00000000
At time                50000, clock = 1, RegWrite = 0, ReadReg1 =  0, ReadReg2 =  0, WriteReg =  0, WriteData = 00000000, ReadData1 = 00000000, ReadData2 = 00000000
At time                60000, clock = 0, RegWrite = 0, ReadReg1 =  0, ReadReg2 =  0, WriteReg =  0, WriteData = 00000000, ReadData1 = 00000000, ReadData2 = 00000000
At time                70000, clock = 1, RegWrite = 0, ReadReg1 =  0, ReadReg2 =  0, WriteReg =  0, WriteData = 00000000, ReadData1 = 00000000, ReadData2 = 00000000
At time                80000, clock = 0, RegWrite = 0, ReadReg1 =  0, ReadReg2 =  0, WriteReg =  0, WriteData = 00000000, ReadData1 = 00000000, ReadData2 = 00000000
At time                90000, clock = 1, RegWrite = 0, ReadReg1 =  0, ReadReg2 =  0, WriteReg =  0, WriteData = 00000000, ReadData1 = 00000000, ReadData2 = 00000000
At time               100000, clock = 0, RegWrite = 1, ReadReg1 =  0, ReadReg2 =  0, WriteReg =  1, WriteData = a5a5a5a5, ReadData1 = 00000000, ReadData2 = 00000000
At time               110000, clock = 1, RegWrite = 1, ReadReg1 =  0, ReadReg2 =  0, WriteReg =  1, WriteData = a5a5a5a5, ReadData1 = 00000000, ReadData2 = 00000000
At time               120000, clock = 0, RegWrite = 1, ReadReg1 =  0, ReadReg2 =  0, WriteReg =  2, WriteData = 5a5a5a5a, ReadData1 = 00000000, ReadData2 = 00000000
At time               130000, clock = 1, RegWrite = 1, ReadReg1 =  0, ReadReg2 =  0, WriteReg =  2, WriteData = 5a5a5a5a, ReadData1 = 00000000, ReadData2 = 00000000
At time               140000, clock = 0, RegWrite = 0, ReadReg1 =  1, ReadReg2 =  2, WriteReg =  2, WriteData = 5a5a5a5a, ReadData1 = a5a5a5a5, ReadData2 = 5a5a5a5a
At time               150000, clock = 1, RegWrite = 0, ReadReg1 =  1, ReadReg2 =  2, WriteReg =  2, WriteData = 5a5a5a5a, ReadData1 = a5a5a5a5, ReadData2 = 5a5a5a5a
```

Waveform



Documentation

Module Declaration:

```
module RegFile (
  clock, WriteEnable,
  ReadReg1, ReadReg2,
  WriteReg, WriteData,
  ReadData1, ReadData2
);
```

This line declares the module's name `RegFile` and specifies its interface, which consists of inputs for the clock, write enable, register addresses, and data to write, as well as outputs for the data read from the registers.

Port Definitions:

- `input clock;`: The clock signal input. It synchronizes the writing of data to the registers.
- `input RegWrite;`: A signal indicating if writing to a register is enabled (1 for enabled, 0 for disabled).
- `input [4:0] ReadReg1, ReadReg2;`: 5-bit inputs specifying which registers to read from.
- `input [4:0] WriteReg;`: A 5-bit input specifying which register to write to.
- `input [31:0] WriteData;`: A 32-bit input specifying the data to be written into the register specified by `WriteReg`.
- `output [31:0] ReadData1, ReadData2;`: 32-bit outputs providing the data read from the registers specified by `ReadReg1` and `ReadReg2`.

Register Array:

```
reg [31:0] reg_mem [0:31];
```

This line declares a 32-element array of 32-bit registers, representing the register file in a MIPS CPU. Each element corresponds to one of the MIPS architecture's 32 general-purpose registers.

Initialization:

The initial block sets all register values to 0 and then assigns some random initial values for demonstration purposes. It also includes debug messages to display the initial state of the register file.

Reading Registers:

The `ReadData1` and `ReadData2` outputs are continuously assigned the values of `reg_mem[ReadReg1]` and `reg_mem[ReadReg2]`, respectively. This means that they always reflect the current contents of the specified registers.

Writing to Registers:

The always block is triggered on the rising edge of the clock. If `RegWrite` is enabled, it writes the value of `WriteData` into the register specified by `WriteReg`.

Usage:

The `RegFile` module is a fundamental component in the MIPS CPU, providing the necessary mechanism for storing and accessing the CPU's general-purpose registers. It's used in nearly every instruction to read operand values and write results.

Testing and Verification:

- Ensure the register file correctly returns the stored values for any specified registers.
- Verify that writing to a register updates its value only when `RegWrite` is enabled and the clock is triggered.
- Check that the register file maintains its state correctly over multiple read and write operations.

Conclusion:

The `RegFile` module is an essential part of the MIPS CPU architecture, serving as the primary storage for the processor's working data. Its design must be efficient and reliable to ensure accurate and fast access to register values. Proper documentation and testing are crucial for maintaining and understanding this component's role within the larger CPU architecture.