

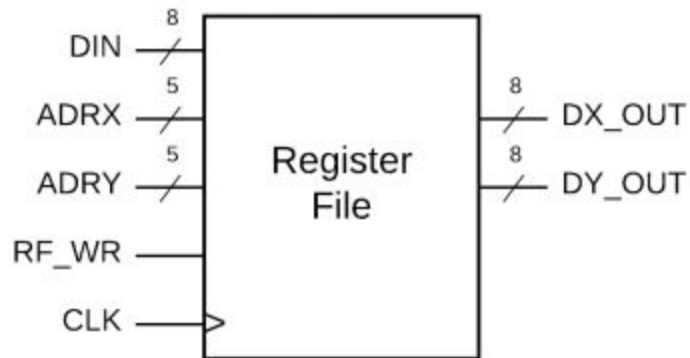
Black Box Diagram :

Figure 1 : Register File

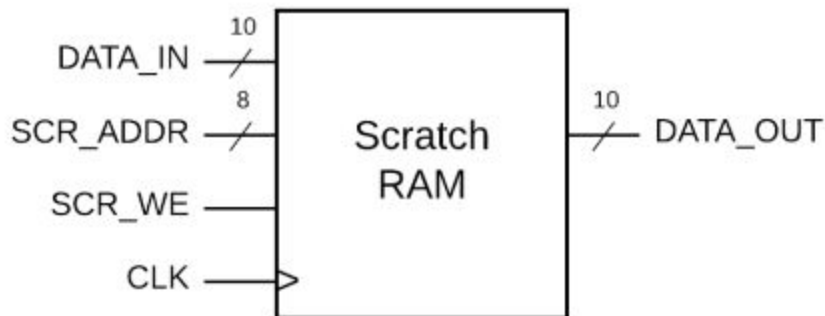


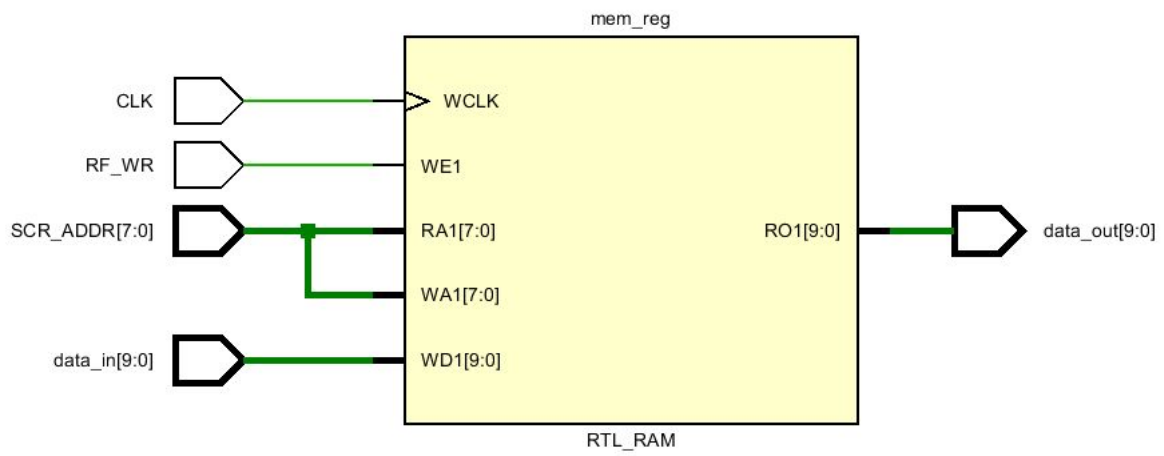
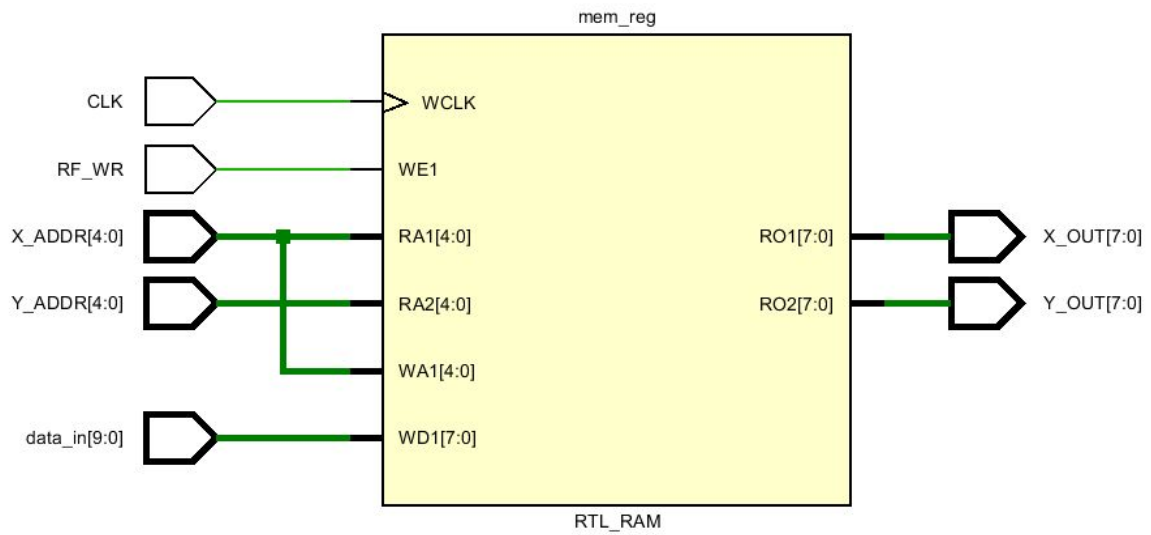
Figure 2 : Scratch RAM

Behavior Design :

The register file designed in Figure 1 is composed of multiple registers, with each register holding a specific spot in the register file. To represent this in our code, we used an array. The register file allows for reading from 2 ports asynchronously, and we read from the array index depending on the two addresses. However, only a single register can be written to per clock cycle, and only if the RF_WR flag is set. This means data is saved synchronously.

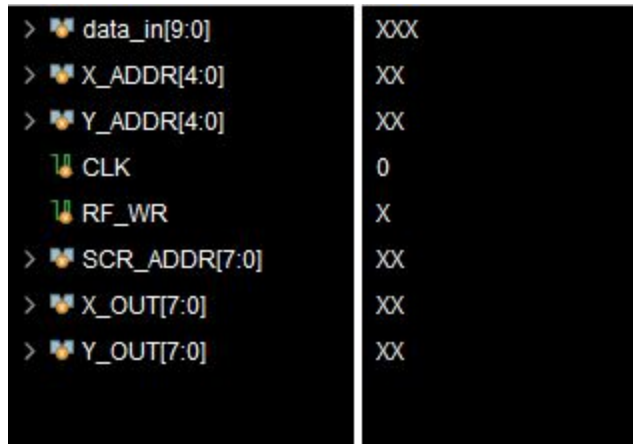
The scratch ram is used to temporarily hold data that the register file can not, and is used as a stack. However, data can be transferred from the scratch ram to the register file. It is single port, so only one address can be read and written to. The data can only be written if the SCR_WE flag is set. Once again the data is read asynchronously while the data is written synchronously.

Structure Design :



Verification :

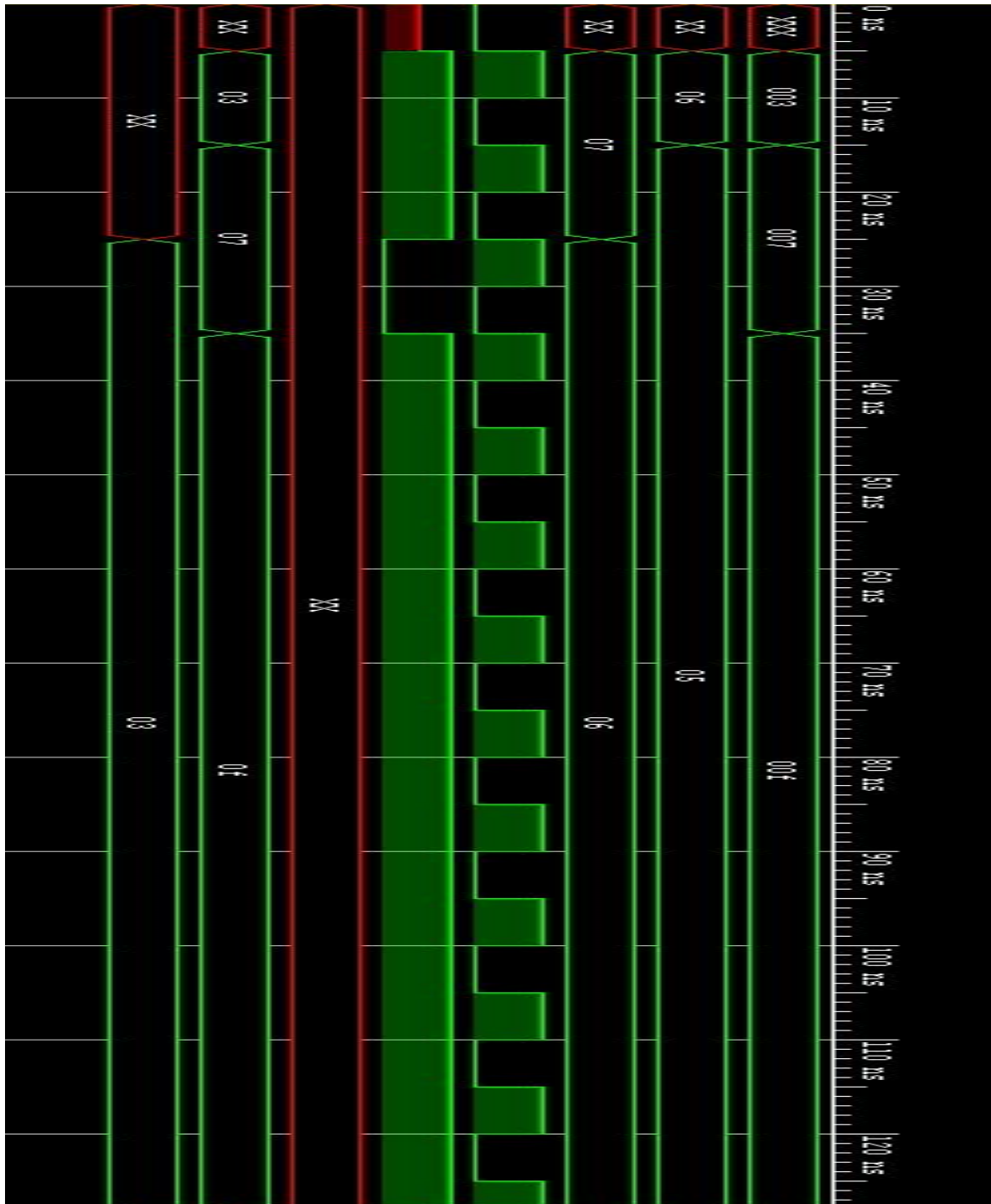
Register File Values:

A screenshot of a simulation console with a black background and white text. It displays a list of register names and their current values. The registers are: data_in[9:0], X_ADDR[4:0], Y_ADDR[4:0], CLK, RF_WR, SCR_ADDR[7:0], X_OUT[7:0], and Y_OUT[7:0]. The values are: XXX, XX, XX, 0, X, XX, XX, and XX respectively. Each register name is preceded by a green icon of a circuit board.

> data_in[9:0]	XXX
> X_ADDR[4:0]	XX
> Y_ADDR[4:0]	XX
CLK	0
RF_WR	X
> SCR_ADDR[7:0]	XX
> X_OUT[7:0]	XX
> Y_OUT[7:0]	XX

In this set of simulations, our team would read and write to certain indexes in the register file. From the chart you can see we write and read from registers 5 and 6. First we write a value of 3 to register 5. We then overwrite that value twice to 7 and then 15. You can also see how we write the value of 3 to register 6 and read from it. All of this happening on the positive clock edge. All these test cases make sure we can read from multiple registers, and write to multiple registers.

Chart for RegisterFile

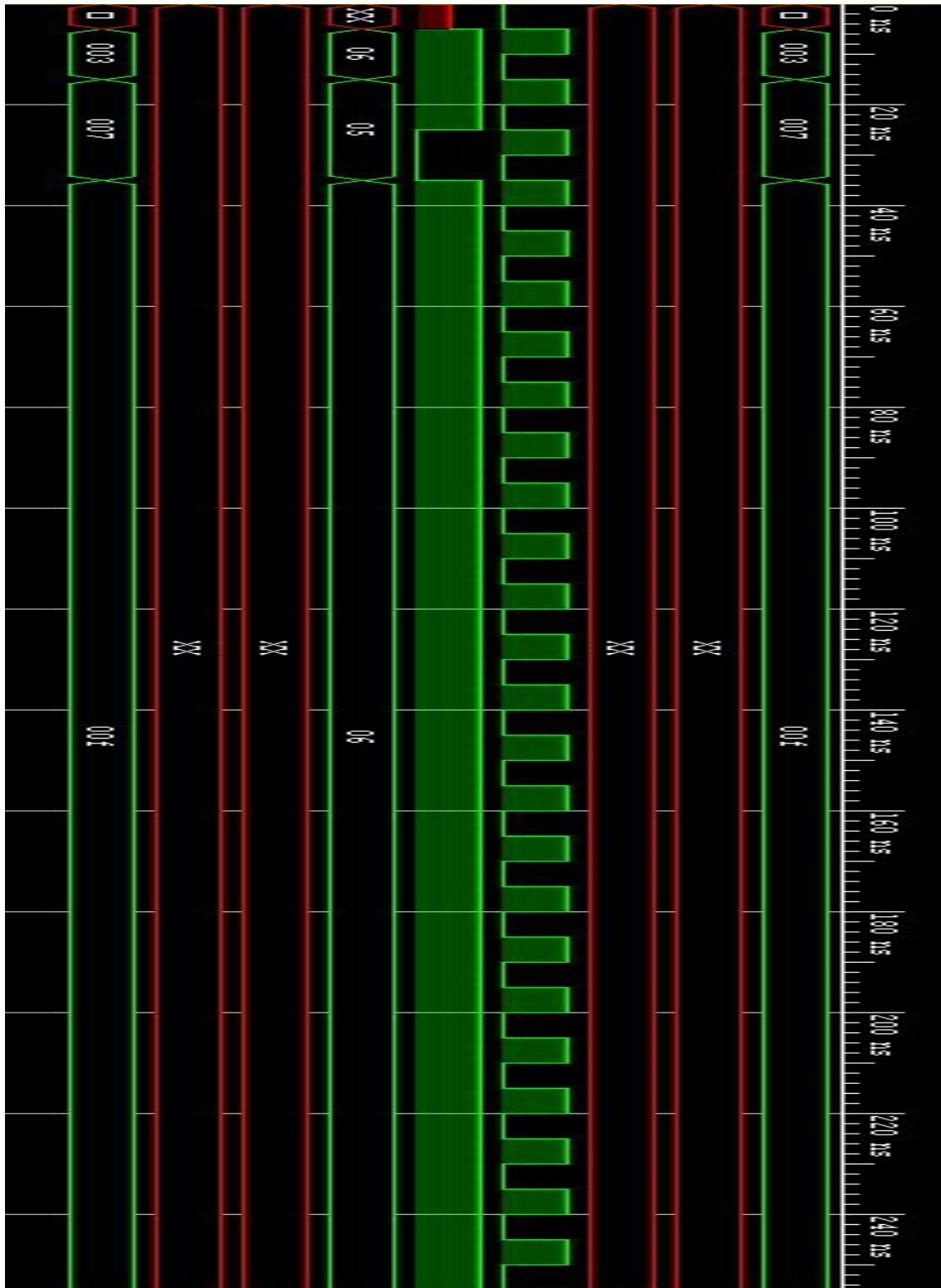


Scratch File Values:

> 🗨 data_in[9:0]	XXX
> 🗨 X_ADDR[4:0]	XX
> 🗨 Y_ADDR[4:0]	XX
🕒 CLK	0
🕒 RF_WR	X
> 🗨 SCR_ADDR[7:0]	XX
> 🗨 X_OUT[7:0]	XX
> 🗨 Y_OUT[7:0]	XX
> 🗨 data_out[9:0]	XXX

In this set of simulations our team would read and write to memory addresses in the scratch memory. We started by writing a value of 3 to register 6, then a value of 7 to register 5, then a value of 15 to register 6. This shows that we can accurately read to a certain index, write to a certain index, and write a value to an index that already has a value. All these test cases make sure we can read from multiple registers, and write to multiple registers.

Scratch File Chart:



System Verilog Source Code :

```

module Scratch(data_in, SCR_ADDR, CLK, RF_WR, data_out);

    input  [9:0] data_in;
    input  [7:0] SCR_ADDR;
    input  CLK, RF_WR;

    output [9:0] data_out;

    logic [3:0] mem [0:255];

    initial begin
        mem = '{1024{'h0}};
    end

    always_ff @ (posedge CLK) begin
        if (RF_WR)
            mem[SCR_ADDR] = data_in;
        end

    assign data_out = mem[SCR_ADDR];

endmodule

module RegisterFile(data_in, X_ADDR, Y_ADDR, CLK, RF_WR, X_OUT,
Y_OUT);

    input  [9:0] data_in;
    input  [4:0] X_ADDR, Y_ADDR;
    input  CLK, RF_WR;

    output [7:0] X_OUT, Y_OUT;

    logic [3:0] mem [0:31];

    initial begin
        mem = '{1024{'h0}};
    end

    always_ff @ (posedge CLK) begin
        if (RF_WR)

```

```
        mem[X_ADDR] = data_in;
    end

    assign X_OUT = mem[X_ADDR];
    assign Y_OUT = mem[Y_ADDR];

endmodule
```