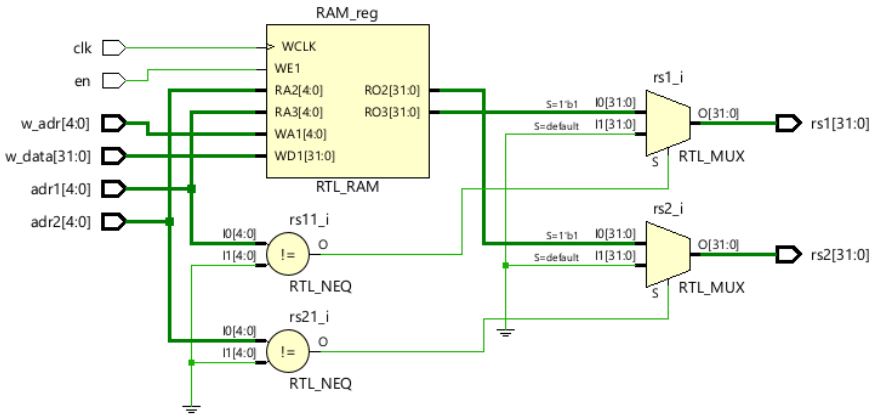
CPE 233 HW 3

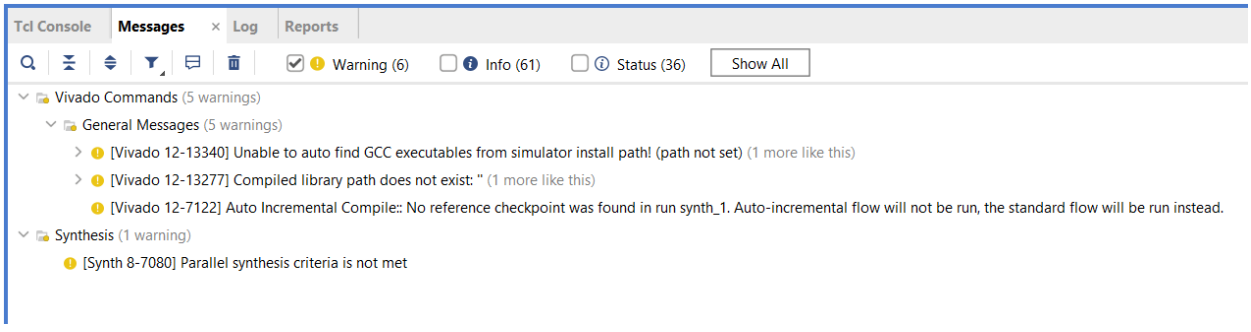
Wyatt Tack

1. Behavior Description: The RegFile module in the 32-bit OTTER MCU functions as the RAM of the hardware, being essentially a matrix of 32 separate 32-bit registers to store memory. The RegFile acts as a file cabinet, in which only one file (one register) can be written over (changed) at a time, provided that the input allowing for write enable is on. The register write over is synchronous, so if en (write enable) is on, and the clock triggers high, the data in w_data (write data) will pe placed into the register with the index chosen by w_adr (write address). The outputs are asynchronous dual read, which means that at all times, the data in the registers with the index of adr1 and adr2 will be output to the 32-bit outputs rs1 and rs2 respectively. The 3 address inputs are each 5 bits, counting 0 to 31, being able to pinpoint each of the 32 registers in the RegFile RAM.

2. Structural Design:



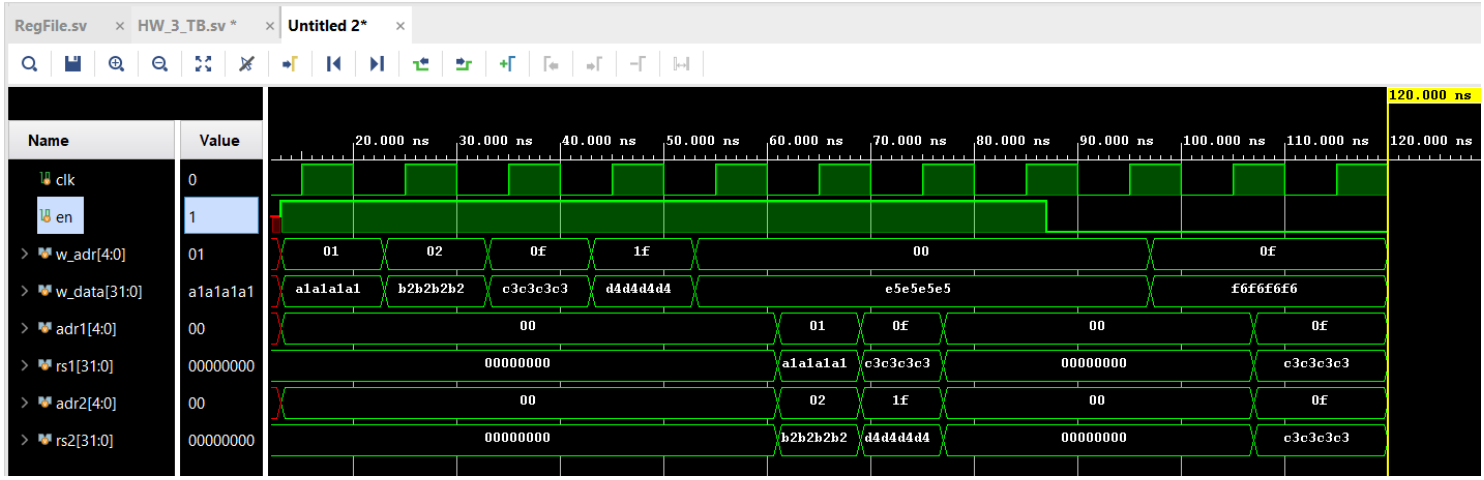
3. Synthesis Warnings Listing:



4. Verification:

Test Case Input	Test Case Output	Test Case Reasoning
en = 1 adr1 = 0 adr2 = 0 w_adr = 1 w_data = 0xa1a1a1a1 w_adr = 2 w_data = 0xb2b2b2b2 w_adr = 15 w_data = 0xc3c3c3c3 w_adr = 31 w_data = 0xd4d4d4d4 w_adr = 0 w_data = 0xe5e5e5e5	(no outputs, just initializing data into registers, outputs rs1 and rs2 are still 0)	Standard operation of inputting data into multiple registers to be used throughout test cases
adr1 = 1 adr2 = 2	rs1 = 0xa1a1a1a1 rs2 = 0xb2b2b2b2	Proves asynchronous dual read works
adr1 = 15 adr2 = 31	rs1 = 0xc3c3c3c3 rs2 = 0xd4d4d4d4	Proves asynchronous dual read works at mid and high end of registers
adr1 = 0 adr2 = 0	rs1 = 0x00000000 rs2 = 0x00000000	Shows register x0 will always output 0 (was written with data in initializing stage)
en = 0 w_adr = 15 w_data = f6f6f6f6f6 adr1 = 15 adr2 = 15	rs1 = 0xc3c3c3c3 rs2 = 0xc3c3c3c3	Ensures no data will be written over registers when en is off

Simulation:



5. System Verilog Source Code:

For RegFile:

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////
// Company: Cal Poly SLO
// Engineer: Wyatt Tack
//
// Create Date: 01/16/2024
// Design Name: RegFile
// Project Name: OTTER MCU
// Target Devices: Basys 3 Board
// Description: Array of 32x32 registers for RAM
//              on OTTER MCU
//
//////////////////////////////////////////////////////////////////

module RegFile(
input en,
input [4:0] adr1,  //two asyncrhonous register reads
input [4:0] adr2,  //and one address, write, and enable
input [4:0] w_adr, //synchronous destination register
input [31:0] w_data,
input clk,
output logic[31:0] rs1,
output logic[31:0] rs2
    );
logic [31:0] RAM [0:31]; //set up 32x32 registers
always_ff@(posedge clk) //write data,
begin                   //functions as register ffs
    if(en == 1)         //syncrhonously
        RAM [w_adr] <= w_data;
end
always_comb             //asynchronous read from RAM
begin                   //with 0 condition (x0 always = 0)
    if (adr1 != 0)
        rs1 = RAM [adr1];
    else
        rs1 = 0;
    if (adr2 != 0)
        rs2 = RAM [adr2];
    else
        rs2 = 0;
end
endmodule
```

Simulation Test Bench:

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////
// Company: Cal Poly SLO
// Engineer: Wyatt Tack
//
// Create Date: 01/16/2024
// Design Name: RegFile TB
// Project Name: OTTER MCU
// Target Devices: Basys 3 Board
// Description: Array of 32x32 registers for RAM
//              on OTTER MCU simulation source
//
//////////////////////////////////////////////////////////////////
module HW_3_TB();
logic en;
logic [4:0] adr1;
logic [4:0] adr2;
logic [4:0] w_adr;
logic [31:0] w_data;
logic clk = 0;
logic[31:0] rs1;
logic[31:0] rs2;
RegFile UUT (.en(en), .adr1(adr1), .adr2(adr2), .w_adr(w_adr),
            .w_data(w_data), .clk(clk), .rs1(rs1), .rs2(rs2));
always begin
#5              //initialize connection and clock pulses
clk = ~clk;
end
always begin
#13 en = 1;     //set read adresses to 0 and start
    adr1 = 0;   //loading registers with data
    adr2 = 0;
    w_adr = 1;
    w_data = 32'ha1a1a1a1;
#10 w_adr = 2;
    w_data = 32'hb2b2b2b2;
#10 w_adr = 15;
    w_data = 32'hc3c3c3c3;
#10 w_adr = 31;
    w_data = 32'hd4d4d4d4;
#10 w_adr = 0;
    w_data = 32'he5e5e5e5;
#8  adr1 = 1;  //prove asyncrhonous read works
    adr2 = 2;
#8  adr1 = 15;
    adr2 = 31;
#8  adr1 = 0;
    adr2 = 0;
#10 en = 0;   //prove en works (data in x15 doesnt change)
#10 w_adr = 15;
    w_data = 32'hf6f6f6f6;
#10 adr1 = 15;
    adr2 = 15;
end
endmodule
```