COEN316 – Lab 2

Name: Christopher Dubuc-Pesetti

Student ID: 40037815

Date submitted: October 15, 2018

"I hereby certify that this submission is my original work and meets the Faculty's Expectations of Originality."

Name: Christopher Dubuc-Pesetti ID: 40037815

Objectives:

The objective of the lab is to design a multi-port (32 ports) 32-bit register file in VHDL. The register file consists of a data in port, two read ports, one write address port, one write enable, a reset port and two output ports. The register file is synchronously clocked.

Results & Discussion:

```
-- create 32 position array of 32-bit registers initialized to all 0's
type registerFile is array (31 downto 0) of std_logic_vector(31 downto 0);
signal registers : registerFile := (others => (others => '0'));
```

An array of vectors was used to store the register values for simplicity's sake. In this case, they are all initialized to zeroes.

```
out_a <=        registers(0) when (read_a = "00000") else
                registers(1) when (read_a = "00001") else
                registers(2) when (read_a = "00010") else
                registers(3) when (read_a = "00011") else
                registers(4) when (read_a = "00100") else
                registers(5) when (read_a = "00101") else
                registers(6) when (read_a = "00110") else
                registers(7) when (read_a = "00111") else
                registers(8) when (read_a = "01000") else
                registers(9) when (read_a = "01001") else
                registers(10) when (read_a = "01010") else
                registers(11) when (read_a = "01011") else
                registers(12) when (read_a = "01100") else
                registers(13) when (read_a = "01101") else
                registers(14) when (read_a = "01110") else
                registers(15) when (read_a = "01111") else
                registers(16) when (read_a = "10000") else
                registers(17) when (read_a = "10001") else
                registers(18) when (read_a = "10010") else
                registers(19) when (read_a = "10011") else
                registers(20) when (read_a = "10100") else
                registers(21) when (read_a = "10101") else
                registers(22) when (read_a = "10110") else
                registers(23) when (read_a = "10111") else
                registers(24) when (read_a = "11000") else
                registers(25) when (read_a = "11001") else
                registers(26) when (read_a = "11010") else
                registers(27) when (read_a = "11011") else
                registers(28) when (read_a = "11100") else
                registers(29) when (read_a = "11101") else
                registers(30) when (read_a = "11110") else
                registers(31);
```

(discussion of above code on next page)

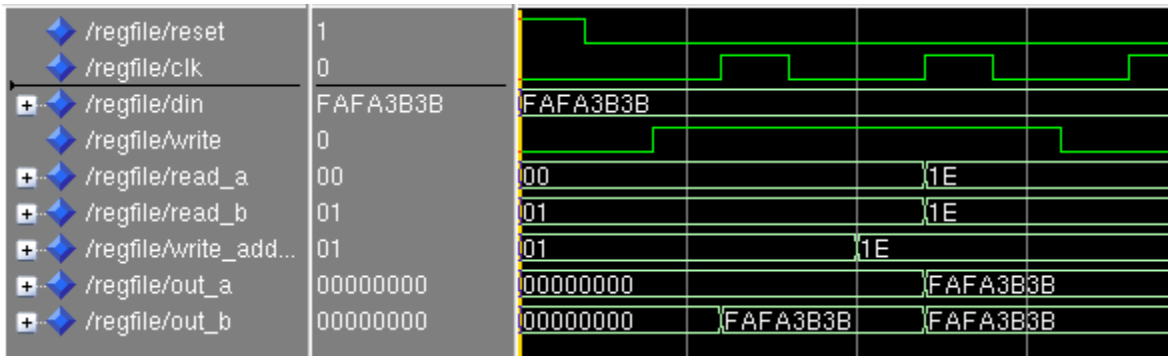
For the read output (code snippet seen on previous page), using the “to_integer” instruction was not possible since we are forced to use the std_unsigned library and are not allowed to use the numeric_std library. Because of this, I used a nested if-else branch to designate which registers were to be written to based on the binary value of the ‘read_a’ and ‘read_b’ inputs. The read_b code block is identical apart from the signal name.

```
process(clk,reset)
begin
        if (reset = '1') then
                registers <= (others=>(others=>'0'));
        elsif (clk'event and clk = '1') then
                if (write = '1') then
                        if (write_address = "00000") then
                                registers(0) <= din;
                        elsif (write_address = "00001") then
                                registers(1) <= din;
                        elsif (write_address = "00010") then
                                registers(2) <= din;
                        elsif (write_address = "00011") then
                                registers(3) <= din;
                        elsif (write_address = "00100") then
                                registers(4) <= din;
                        elsif (write_address = "00101") then
                                registers(5) <= din;
                        elsif (write_address = "00110") then
                                registers(6) <= din;
                        elsif (write_address = "00111") then
                                registers(7) <= din;
                        elsif (write_address = "01000") then
                                registers(8) <= din;
                        elsif (write_address = "01001") then
                                registers(9) <= din;
                        elsif (write_address = "01010") then
                                registers(10) <= din;
                        elsif (write_address = "01011") then
                                registers(11) <= din;
                        elsif (write_address = "01100") then
                                registers(12) <= din;
                        elsif (write_address = "01101") then
                                registers(13) <= din;
                        elsif (write_address = "01110") then
                                registers(14) <= din;
                        elsif (write_address = "01111") then
                                registers(15) <= din;
                        elsif (write_address = "10000") then
                                registers(16) <= din;
                        elsif (write_address = "10001") then
                                registers(17) <= din;
                        elsif (write_address = "10010") then
                                registers(18) <= din;
```

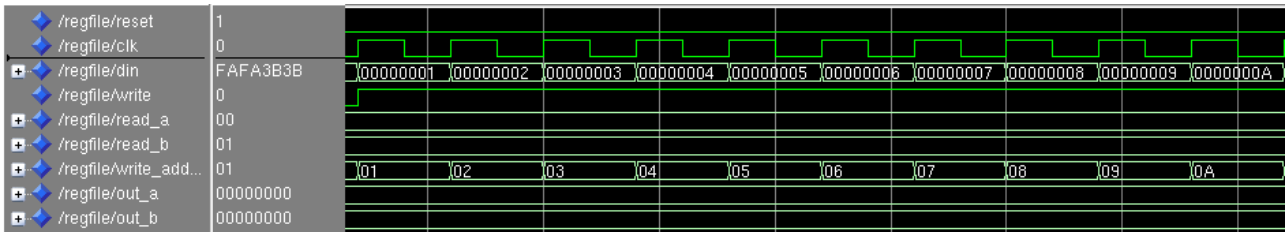
Above is the code snippet of the process used for writing to the registers using a synchronized clock. The code continues until register(31) but I cut the image off due to space limitations.

As with the output reading, a nested if-else branch was used instead of using the simpler "to_integer" instruction to convert the binary write_address to an integer value.

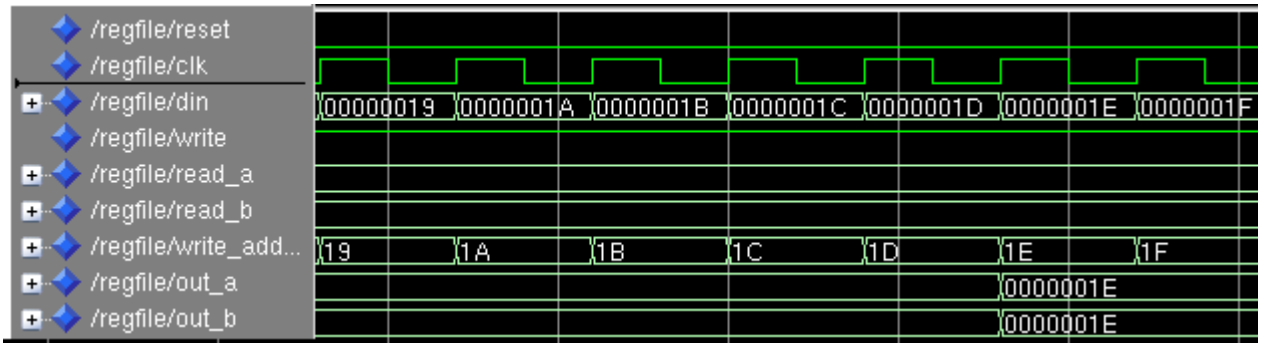
Test case for the simulation provided by the lab manual.



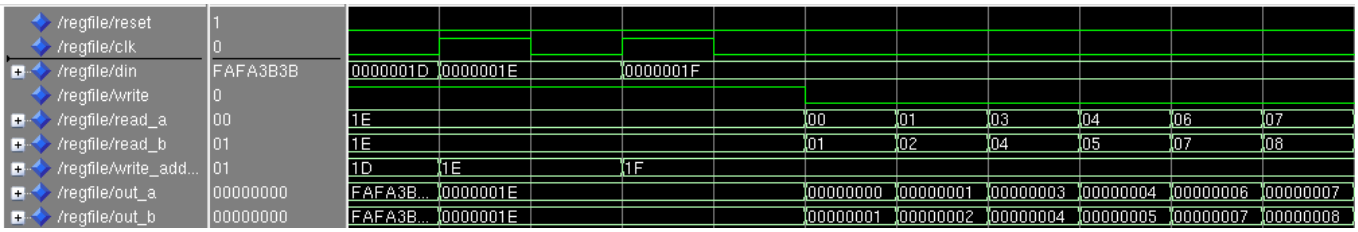
Above is the simulation result of the first 4 tests. Registers are reset, din set to "FAFA3B3B" and written to register 01. Output of R01 then output to both out_a and out_b.



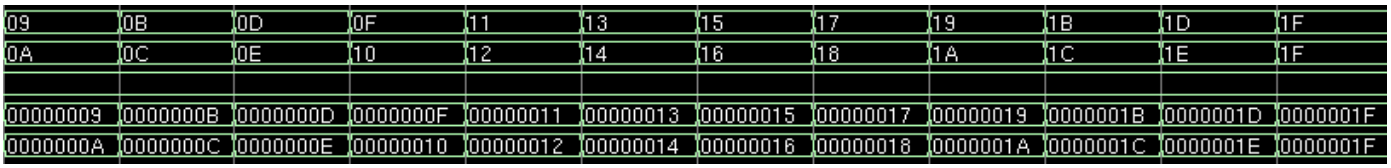
Above is the case where the registers are filled with data corresponding to their address number. Above is the lower half of the registers (cut off slightly due to space limitations).



This continues into the upper half of the registers (shortened due to space). Since the value of read_a and read_b were kept at 0x1E from the start, the corresponding change in register value was output to the output ports.



Above is the reading-from-registers lower portion of the test case. Read_a and read_b are incremented and the values of the registers are placed on those ports.



Above is the upper half of the register reading test case.

Conclusion:

Relatively simple design process for this portion of the CPU. Using the numeric_std library instead of std_unsigned would simplify and shorten the code. Simulation and demo performed successfully.

Appendix