CONCORDIA UNIVERSITY
Department of Electrical and Computer Engineering
COEN 316 Computer Architecture
Lab 2
Register File
Fall 2021

Introduction

In this lab, a multi-port register file will be designed. The following VHDL entity specification is to be used for the design of the register file:

```
-- 32 x 32 register file
-- two read ports, one write port with write enable

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

entity regfile  is
port( din   : in std_logic_vector(31 downto 0);
      reset : in std_logic;
      clk   : in std_logic;
      write : in std_logic;
      read_a : in std_logic_vector(4 downto 0);
      read_b : in std_logic_vector(4 downto 0);
      write_address : in std_logic_vector(4 downto 0);
      out_a  : out std_logic_vector(31 downto 0);
      out_b  : out std_logic_vector(31 downto 0));
end regfile ;
```

The register file consists of 32 registers (R0-R31), each register consisting of 32 bits. The entire register file can be reset (each register cleared to 0) by asserting the **asynchronous active-high** `reset` input. There are two read ports (`out_a` and `out_b`). The 5 bit addresses presented on the input ports `read_a` and `read_b` determine which of the 32 registers are made available at the output ports `out_a` and `out_b` respectively. For example, if `read_a` = "00101" and if `read_b` = "01111" the contents of register R5 is outputted on `out_a` and register R15 is outputted on the `out_b` port. Note that the reading is done asynchronously (i,.e. independent of the clock input). The register file contains a single input port (din) which is used to provide 32 bit data to be written into a specific register. The 5 bit address presented on the `write_address` input determines which of the 32 registers is to be written to. The writing of the data to the specified register is synchronous (i.e. occurs with a rising clock edge) and is also controlled by the `write` signal which is an active high signal (in order for a write to occur, the `write` signal must be '1' and the clock input must change from '0' to '1').

Figure 1 gives the block diagram of the register file showing its input and output ports.

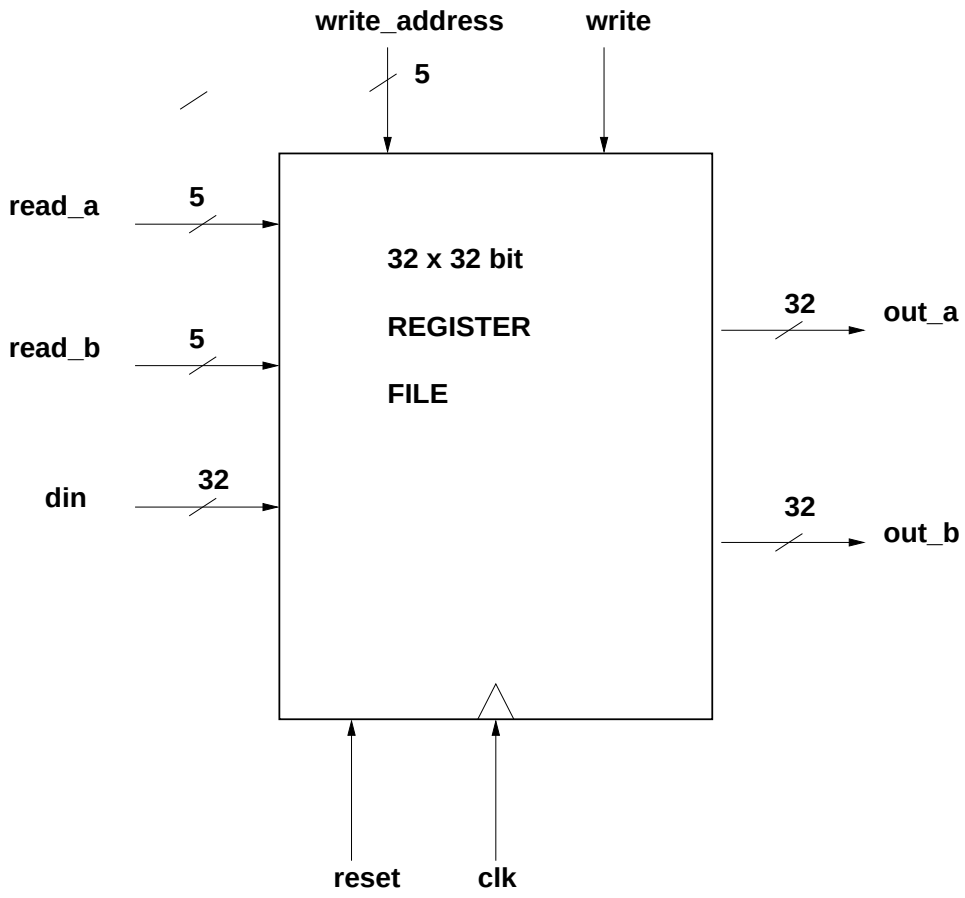

Figure 1: Register file block diagram.

Procedure

Design the register file using VHDL. You may use any style of VHDL (i.e. structural with port maps, processes, CSA statements, etc.) as you wish. Simulate your design with the Modelsim simulator to verify correct functioning for all the possible operations supported by the register file for typical input values. Synthesize your VHDL code with the Xilinx Vivado tools.

Board Implementation

Similar to Lab 1, only a subset of the various input and output ports will be mapped to switches and LEDs via a .xdc file. You are to simulate the full 32-bit design and to create a “board wrap-

per" VHDL file for implementation. Use the following entity specification for the "board wrapper" VHDL code:

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

entity regfile  is
port( din_in   : in std_logic_vector(3 downto 0);
      reset : in std_logic;
      clk   : in std_logic;
      write : in std_logic;
      read_a_in : in std_logic_vector(1 downto 0);
      read_b_in : in std_logic_vector(1 downto 0);
      write_address_in : in std_logic_vector(1 downto 0);
      out_a_out  : out std_logic_vector(3 downto 0);
      out_b_out  : out std_logic_vector(3 downto 0));
end regfile ;
```

Requirements

1. Modelsim simulation results (including your DO file) for the design. It is suggested that one uses the `X"` notation to specify hexadecimal values in a `force` statement as in:

```
force din X"FAFA3B3B"
```

Similar notation may be used within a DO file. Within the Waveform window, one can select a particular signal and specify the radix as hexadecimal. This will allow for easier understanding of the displayed values.

You are to test your register file design for several different input combinations. Write values into several registers, then read out the values. Document clearly in the lab report (with the use of a table or other means) the test cases that you have selected and the expected values and whether your design simulates as expected. Your test inputs should first reset all the registers, and then proceed to write values into certain registers and then read out the registers which you have written into and also read out other registers which have NOT been written into.

2. VHDL code for the register file (for both the simulation and "board-wrapper" versions).

3. The Xilinx Vivado `runme.log` files for the synthesis and implementation runs. Discuss the importance of any messages or warnings. The `runme.log` files are found within the `synth_1` and `impl_1` subdirectories of your Vivado project directory.

4. Demonstrate the operation of your register file on the FPGA board to your lab TA (you may demo at the beginning of your next lab session).

Ted Obuchowicz
Sept. 29, 2016

Revised: Oct. 15, 2020. Mofidied for command line simulation and TCL Vivado synthesis script due to onlines labs due to COVID-19.

Revised: Oct. 1, 2021. Modified for use with Vivado GUI tools and implementation of subset of i/o bits on Nexys A7 FPGA board.