CONCORDIA UNIVERSITY
Department of Electrical and Computer Engineering
COEN 316 Computer Architecture
Lab 2
Register File
Fall 2017

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 out-putted 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 speci-fied 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.

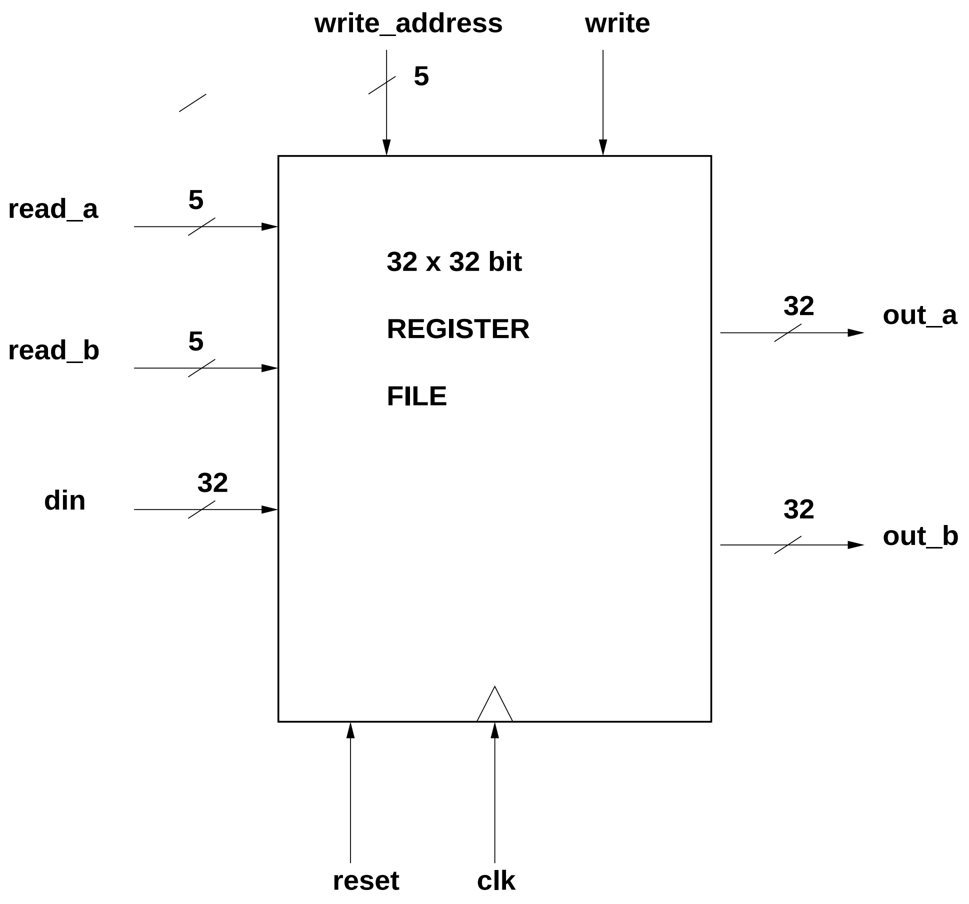

Figure 1: Register file block diagram.

Procedure

Design the register file using VHDL. You may use any style of synthesizeable VHDL (i.e. structural with port maps, processes, CSA statements, etc.) 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 Precision RTL. Implement the design using the Xilinx ISE and Impact tools to generate the .ace file. For this lab, it is not necessary to program the XUPV2Pro FPGA board due to the limited number of inputs and outputs available on the board. However, you should carefully review the log files produced by the Xilinx tools to ensure that the design was properly implemented. Demonstrate the operation (the Modelsim simulation) of the register to your lab demonstrator.

Requirements

1. Modelsim simulation results for the design. Use the following Modelsim DO file:

```
# do file to test the  regfile

add wave reset
add wave clk
add wave din
add wave write
add wave read_a
add wave read_b
add wave write_address
add wave out_a
add wave out_b

# reset
force  reset 1
force clk 0
force din X"FAFA3B3B"
force  write 0
force   write_address "00001"
force read_a "00000"
force read_b "00001"
run 2


#  deassert reset and write into R1

force reset 0
run 2
force write 1
run 2
force clk 1
run 2
force clk 0
run 2


# write X"FAFA3B3B" into R31 and sent R31 to both out ports
force write_address "11110"
run 2
force read_a "11110"
force read_b "11110"
force clk 1
run 2
```

```
force clk 0
run 2


# deassert write, and try to write new data into R31, shouild be
no change
force write 0
run 2
force clk 1
run 2
force clk 0
run 2


# write data into R1-R31 consecutively
force write 1
force write_address “00001”
force din X”00000001”
force clk 1
run 2
force clk 0
run 2

force write_address “00010”
force din X”00000002”
force clk 1
run 2
force clk 0
run 2

force write_address “00011”
force din X”00000003”
force clk 1
run 2
force clk 0
run 2

force write_address “00100”
force din X”00000004”
force clk 1
run 2
force clk 0
run 2

force write_address “00101”
force din X”00000005”
force clk 1
run 2
force clk 0
run 2

force write_address “00110”
force din X”00000006”
force clk 1
run 2
force clk 0
run 2

force write_address “00111”
force din X”00000007”
force clk 1
run 2
force clk 0
run 2


force write_address “01000”
force din X”00000008”
force clk 1
run 2
force clk 0
run 2


force write_address “01001”
force din X”00000009”
force clk 1
run 2
force clk 0
run 2

force write_address “01010”
force din X”0000000A”
force clk 1
run 2
force clk 0
run 2

force write_address “01011”
force din X”0000000B”
force clk 1
run 2
```

```
force clk 0
run 2

force write_address “01100”
force din X”0000000C”
force clk 1
run 2
force clk 0
run 2

force write_address “01101”
force din X”0000000D”
force clk 1
run 2
force clk 0
run 2

force write_address “01110”
force din X”0000000E”
force clk 1
run 2
force clk 0
run 2

force write_address “01111”
force din X”0000000F”
force clk 1
run 2
force clk 0
run 2



# second hald of registers

force write_address “10000”
force din X”00000010”
force clk 1
run 2
force clk 0
run 2



force write_address “10001”
force din X”00000011”
force clk 1
run 2
force clk 0
run 2

force write_address “10010”
force din X”00000012”
force clk 1
run 2
force clk 0
run 2

force write_address “10011”
force din X”00000013”
force clk 1
run 2
force clk 0
run 2

force write_address “10100”
force din X”00000014”
force clk 1
run 2
force clk 0
run 2

force write_address “10101”
force din X”00000015”
force clk 1
run 2
force clk 0
run 2

force write_address “10110”
force din X”00000016”
force clk 1
run 2
force clk 0
run 2

force write_address “10111”
force din X”00000017”
force clk 1
run 2
force clk 0
run 2
```

```
force write_address “11000”
force din X”00000018”
force clk 1
run 2
force clk 0
run 2


force write_address “11001”
force din X”00000019”
force clk 1
run 2
force clk 0
run 2

force write_address “11010”
force din X”0000001A”
force clk 1
run 2
force clk 0
run 2

force write_address “11011”
force din X”0000001B”
force clk 1
run 2
force clk 0
run 2

force write_address “11100”
force din X”0000001C”
force clk 1
run 2
force clk 0
run 2

force write_address “11101”
force din X”0000001D”
force clk 1
run 2
force clk 0
run 2
```

```
force write_address “11110”
force din X”0000001E”
force clk 1
run 2
force clk 0
run 2


force write_address “11111”
force din X”0000001F”
force clk 1
run 2
force clk 0
run 2


# now deassert the write and read out consecutively asynchro-
nously

force write 0
force read_a “00000”
force read_b “00001”
run 2

force read_a “00001”
force read_b “00010”
run 2

force read_a “00011”
force read_b “00100”
run 2

force read_a “00100”
force read_b “00101”
run 2

force read_a “00110”
force read_b “00111”
run 2

force read_a “00111”
force read_b “01000”
run 2


# next quarter
```

```
force read_a “01001”
force read_b “01010”
run 2

force read_a “01011”
force read_b “01100”
run 2

force read_a “01101”
force read_b “01110”
run 2

force read_a “01111”
force read_b “10000”
run 2



# second half of reads


force read_a “10001”
force read_b “10010”
run 2

force read_a “10011”
force read_b “10100”
run 2

force read_a “10101”
force read_b “10110”
run 2

force read_a “10111”
force read_b “11000”
run 2

force read_a “11001”
force read_b “11010”
run 2

force read_a “11011”
force read_b “11100”
```

```
run 2


# next 3 quarter


force read_a “11101”
force read_b “11110”
run 2

force read_a “11111”
force read_b “11111”
run 2
```

2. RTL schematic diagram of the synthesized circuit.

3. The precision.log file highlighting any messages/warnings. Discuss the importance of any messages or warnings.

Ted Obuchowicz
Sept. 29, 2016