Lab 4 – Register File

Objectives

In this lab, we will develop a Register File for a processor, and will simulate its behavior in ModelSim.

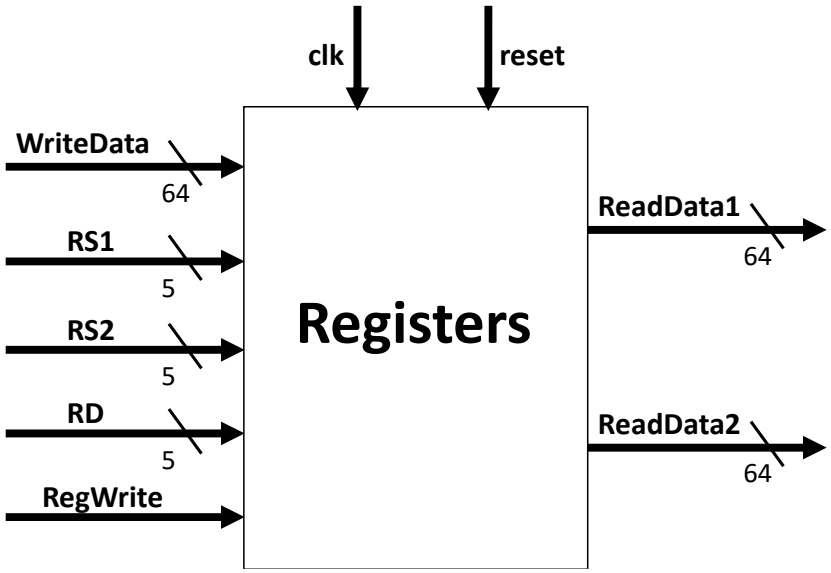
Section	
a) Introduction A brief overview of a Register Files.	**05**
b) Implementation In this section, you will implement the Register File module.	**60**
Exercise Small exercise to practice further.	**60**

a. Introduction

Register files are necessary for computer memory. For a computer to be able to function, it needs to have some form of memory. There are two different forms of memory devices: external memory devices and local memory devices. External memory devices are items such as RAM, ROM, disk drives, etc. Registers provide local memory, which allows for temporary storage of data that is about to be processed. For this lab, registers will be used to save and store numbers.

b. Implementation

Create a module named, *`registerFile`*, and define 18 64-bit registers. The module should have the input address ports for reading register 1, register 2, and for writing data in a register. Furthermore, it should have a 64-bits data input port, two 64-bits data output ports to read the values from two different registers. Finally, it should have a clk, reset, and an enable signal to write the data. The top-level diagram of this module is shown in the figure below.



Create a test bench to simulate its behavior in ModelSim.

The following command creates an `Array` of type `reg` containing 10 elements each of which are 5 bits wide.

```
reg [4:0] Array [9:0]
```

Exercise

Create a `top` module having 32-bit input, named `instruction`, and two 64-bit outputs, named `ReadData1` and `ReadData2`. Now, instantiate `instruction parser` module developed in lab 3 and `registerFile` module developed in this lab. Make appropriate connections.

Create a testbench and pass any 32-bit `instruction` in `top` module, and observe if the data of correct source registers are obtained.