INTRODUCTION

Register files are small memories holding temporary values during computation. They are one part of the **central processing unit (CPU)** of a microprocessor. Register files sit at the top of the memory pyramid – the collection of memories that store data in a computer system. While a register holds a single n-bit wide data value, a register file holds multiple n-bit wide values.

An analogy for a register file is a file cabinet. File cabinets have drawers where data – usually paper – is stored. Users of a file cabinet can open a drawer and remove a data set to **read**. Similarly, users of a file cabinet can open a drawer and **place** a new data set into the drawer.

Register files extend the file cabinet concept to voltage storage. The central processing unit **reads** and **writes** data into registers selected with **addresses**. The total number of registers in the register file is fixed to a power-of-2. If a register file provides ***m*** registers, then the number of required address bits is $\log_2 m$.

Consider a typical calculation such as Y = A+B. This calculation requires two pieces of data, **A** and **B**. The central processing unit will first request these **operands** from the main memory of the computer in a **load-to-register** operation. Each operand will be stored in a unique register. Next, the central processing unit will execute an **add** operation that **reads** the operands from the register, completes the arithmetic, and **writes** the result to a third register. A principle in computer architecture called **temporal locality** suggests that the programmer will likely use the calculated value soon in the program. Thus, the central processing unit will not immediately store the calculated result back into main memory. An instruction level view of this calculation using three registers numbered R0, R1, and R2 is:

```
load    R0, Mem[A]     ; comment: R0 = Mem[A]
load    R1, Mem[B]     ; comment: R1 = Mem[B]
add     R2, R0, R1     ; comment: R2 = R0 + R1
```

Each instruction executes in one clock period. The third instruction suggests that the register file should be able to provide two data values to the arithmetic unit in a single clock period. Thus, the register file must have two **read ports**. Similarly, the third instruction suggests that the result should be stored back into the register file in the same clock period**.** Thus, the register file must have one **write port**. The central processing unit divides the instruction into behaviors that occur during the first half of the clock period and behaviors that occur during the second half of the clock period.

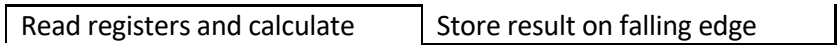

Figure 1: Dividing the clock period into register file behaviors

Standard register files are **2-read/1-write** register files and thus have three address bus inputs: two specifying the read registers and one specifying the write register. Figure 2 shows the functional level symbol for a register file and Figure 3 shows the internal circuit.

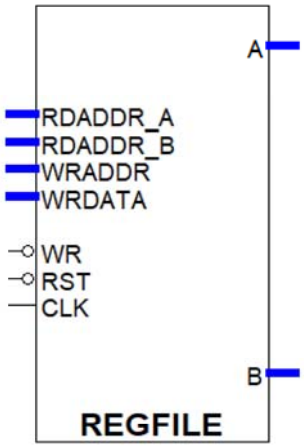

Figure 2: Functional Symbol for a 2-read/1-write register file

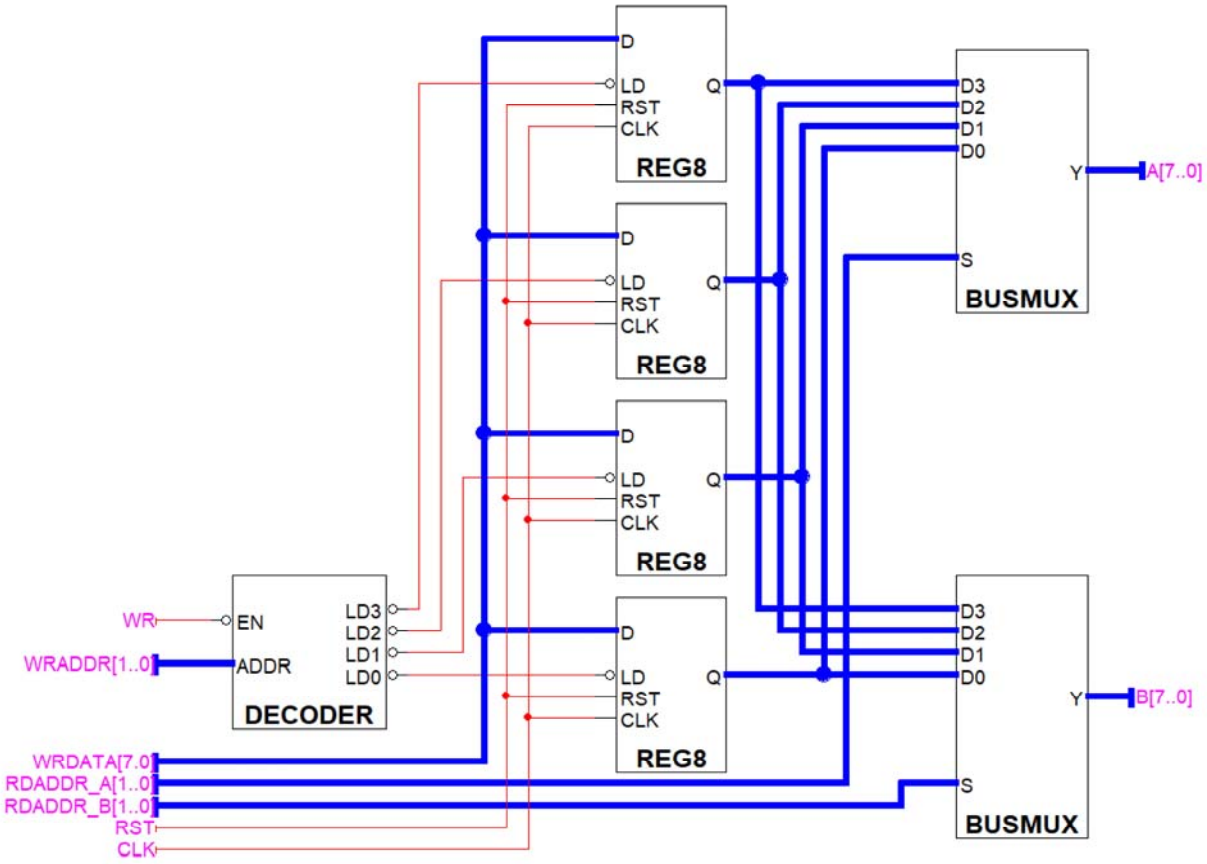

Figure 3: Internal Circuit of a 2-read/1-write Register File

The internal circuit of Figure 3 shows four byte-sized registers sharing a common connection to the clock, to reset, and to the write data. These registers hold data for the central processing unit. Read behavior is provided by two output multiplexers. Each read-port multiplexer has its selection bits attached to one of the read addresses. Similarly, Figure 3 shows an address decoder used to select which register should write data based on the write address. Not every central processing unit instruction will write calculated values back into the register file. Thus, the register file has a **write control signal** called **WR**. This signal is connected to the address decoder. The diagram shows this control signal as **active-low**. Thus, the behavior of the decoder can be summarized as:

- If the WR signal is asserted and if the write address is 0 then LD0 asserts
- If the WR signal is asserted and if the write address is 1 then LD1 asserts
- If the WR signal is asserted and if the write address is 2 then LD2 asserts
- If the WR signal is asserted and if the write address is 3 then LD3 asserts
- If the WR signal is not asserted than no LD signal asserts

Together, the address decoder and the output multiplexers provide the read and write behaviors of the register file.

IMPLEMENTATION STRATEGIES

Standard digital logic components are interconnected to form the register file. This interconnection could be done schematically or described using a hardware description language. The input and output ports of the register file are shown in Figure 2 and eight internal signals can also be seen in Figure 3. These internal signals are listed in Table 1.

Table 1: Internal Signals Visible in Figure 3

INTERNAL SIGNAL	PURPOSE
LD3	command register 3 to load on the falling-edge of the clock
LD2	command register 2 to load on the falling-edge of the clock
LD1	command register 1 to load on the falling-edge of the clock
LD0	command register 0 to load on the falling-edge of the clock
Q3	output of register R3
Q2	output of register R2
Q1	output of register R1
Q0	output of register R0

- Strategy 1: Implement a VHDL bus multiplexer (busmux.vhd), a VHDL byte-sized register (reg8.vhd), and an address decoder (decoder.vhd) as components in a Quartus schematic project (regfile.bdf).
- Strategy 2: Implement a VHDL bus multiplexer (busmux.vhd), a VHDL byte-sized register (reg8.vhd), and an address decoder (decoder.vhd) as components in a Quartus structural VHDL project (regfile.vhd).
- Strategy 3: Implement the register file in a single Quartus behavioral VHDL project (regfile.vhd).

LABORATORY REQUIREMENTS

1. **Implement** the register file in Figure 3 using one of the three strategies.
2. **Write** a VHDL test bench that simulates the register file and verifies operation.
3. **Submit** a laboratory report using your instructor's preferred method.

This laboratory is **simulation only**. You are not required to configure the DE10-Lite FPGA.

TESTBENCH SUGGESTIONS

There are multiple behaviors that should be tested: reset, data write, and data read. A good testbench will simulate all three behaviors multiple times. Here is a suggested data flow.

- Reset behavior: tested at the beginning of the simulation to ensure the registers all zero.
- Read behavior: rdaddr_a = 0, rdaddr_b = 1, verify that R0 and R1 produce 0 on outputs A and B
- Read behavior: rdaddr_a = 2, rdaddr_b = 3, verify that R2 and R3 produce 0 on outputs A and B
- Write behavior: wraddr = 1, wrdata = 9, wr = 0
- Read behavior: rdaddr_a = 0, rdaddr_b = 1, verify that R0 and R1 produce 0 and 9 on A and B
- Read behavior: rdaddr_a = 1, rdaddr_b = 2, verify that R1 and R2 produce 9 and 0 on A and B
- Write behavior: wraddr = 0, wrdata = 13, wr = 0
- Read behavior: rdaddr_a = 0, rdaddr_b = 3, verify that R0 and R3 produce 13 and 0 on A and B
- Write behavior: wraddr = 2, wrdata = 6, wr = 0
- Read behavior: rdaddr_a = 2, rdaddr_b = 1, verify that R2 and R1 produce 6 and 9 on A and B
- Reset behavior: tested at the end of the simulation to ensure the registers all zero.
- Read behavior: rdaddr_a = 0, rdaddr_b = 1, verify that R0 and R1 produce 0 on outputs A and B
- Read behavior: rdaddr_a = 2, rdaddr_b = 3, verify that R2 and R3 produce 0 on outputs A and B

STRATEGY 3 HINTS

- The eight internal signals identified in Table 1 are declared in the signal section.
- Each internal Q signal has a corresponding process that samples WRDATA if its load is asserted.
- The address decoder is a set of with-select statements or when-else statements generating each LD signal.

  with WR & WRADDR select or LD0 <= '0' when WR = ….
  LD0 <= '0' when … ,

- The output multiplexers are with-select statements.

  with RDADDR_A select
  A <= Q3 when B"11", …