CE3001 Lab2. Register File(RF)

Lu Shengliang

SLU001

School of Computer Engineering

Nanyang Technological University

SLU001@e.ntu.edu.sg

September 27, 2013

Abstract

A register file(RF) is an array of data storage registers while the central processing unit(CPU) is working on it. The RF designed in this report is 16*16-bit with data read and controlled write processes.

1 Introduction

1.1 Description of RF Specification

All the registers in the RF are positive edge triggered flip-flops. The specification of port lists is described in Table 1.

1.2 Structure of the rest of the paper

The rest of the paper first describes the Verilog implementation of RF in *Section 2*. *Section 3* presents the experimental results using testbench, which valid the functionality of our RF. *Section 4* presents our conclusions and discussions.

Port	Direction	Size	Description
RAddr1	Input	4-bit	Address for first Read Port
RAddr2	Input	4-bit	Address for second Read Port
WAddr	Input	4-bit	Address for Write Port
WData	Input	16-bit	Data for Write Port
Wen	Input	1-bit	Active-High enable signal for Write Port
Clock	Input	1-bit	Clock signal for the circuit
Reset	Input	1-bit	Reset signal. Synchronous active-low.
RData1	Output	16-bit	Data for first Read Port
RData2	Output	16-bit	Data for second Read Port

Note: Register 0 is always 0; All registers are reset to 0x0000 when *reset* is asserted; The RF supports simultaneous reads and write.

Table 1: RF Port List Specification

2 Implementation

2.1 Verilog Code RF.v

```
module Reg_File (RAddr1, RAddr2, WAddr, WData,
                 Wen, Clock, Reset, RData1, RData2);
  input [3:0]  RAddr1, RAddr2;
  input [3:0]  WAddr;
  input [15:0] WData;
  input        Wen, Clock, Reset;

  output reg [15:0] RData1, RData2;
         reg [15:0] regFile [0:15];
  integer           i;

  always @(!Reset) begin
    for (i = 0; i < 8; i = i + 1) begin
      regFile[i] <= 16'b0;
    end
  end
  always @(posedge Clock) begin
    if (Reset) begin
      if (Wen) regFile[WAddr] <= WData;
      regFile[0] = 0;
      RData1 = regFile[RAddr1];
      RData2 = regFile[RAddr2];
    end
  end
endmodule // Reg_File
```

Our implementing strategies are using *always* block to test the *Reset* signal. If *Reset* is asserted, the RF will be flushed to *zero*. Besides, whenever there is a posedge of the *Clock* signal, the *Reset* is not asserted and the write enable signal *Wen* is asserted, *WData* will be written into RF at the location of *WAddr*.

Two words will be read from RF as the result of no *Reset* signal.
The implementation results will be listed in *Section 3*.

3 Evaluation

3.1 Testbench Code 1 RF_tb.v

```
`define RSIZE 4
`define DSIZE 16
`timescale 1ns / 100ps
module RF_tb ();
  reg clk, rst, wen;
  reg [`RSIZE-1:0] RAddr1, RAddr2;
  reg [`RSIZE-1:0] WAddr;
  reg [`DSIZE-1:0] WData;

  wire [`DSIZE-1:0] RData1, RData2;
  integer           i;
  Reg_File RF_inst (
                    .Clock(clk), .Reset(rst), .Wen(wen),
                    .RAddr1(RAddr1), .RAddr2(RAddr2),
                    .WAddr(WAddr), .WData(WData),
                    .RData1(RData1), .RData2(RData2)
                    );
  always #5 clk = ~clk;

  initial begin
    clk = 0;
    rst = 1;
    wen = 0;
    #5 rst = 0;
    #10 rst = 1;
    wen = 1;

    for (i=0; i<16; i=i+1)  //write to RF
      begin
        #5 WAddr = i;
           WData = i+16;
        #10  $display("Write: WAddr=%h, WData=%h\n",WAddr ,WData);
      end
    $display("============================");

    for (i=0; i<16; i=i+2)    // read from RF
      begin
        #5 RAddr1 = i; RAddr2 = i+1;
        #10 $display("Read: RData1=%h, RData2=%h\n",RData1, RData2);
      end
    $display("============================");

    for (i=0; i<14; i=i+2)    //2 reads and 1 write in one cycle
      begin                   //simultaneous
        #5 RAddr1 = i; RAddr2 = i+1;
           WAddr = i+2;
           WData = 5;
        #10 $display("Read: RData1=%h, RData2=%h\n",RData1, RData2);
      end
    #5 RAddr1 = 14; RAddr2 = 15;
    #10 $display("Read: RData1=%h, RData2=%h\n",RData1, RData2);
    #1000 $finish;
  end
endmodule // end of RF_tb
```

A testbench has been designed to test the *RF.v* file. A simulation function is provided by *ModelSim* software. The Testbench uses three *for* loops. First *for* loop is used to write initial value into RF. Second *for* loop is used to check what was grated in to the RF. The last *for* simultaneously reads twice and write once. The console output will be provided below.

3.2 Testbench result of RF_tb.v

```
# Writing: WAddr=0, WData=0010#
# Writing: WAddr=1, WData=0011#
# Writing: WAddr=2, WData=0012#
# Writing: WAddr=3, WData=0013#
# Writing: WAddr=4, WData=0014#
# Writing: WAddr=5, WData=0015#
# Writing: WAddr=6, WData=0016#
# Writing: WAddr=7, WData=0017#
# Writing: WAddr=8, WData=0018#
# Writing: WAddr=9, WData=0019#
# Writing: WAddr=a, WData=001a#
# Writing: WAddr=b, WData=001b#
# Writing: WAddr=c, WData=001c#
# Writing: WAddr=d, WData=001d#
# Writing: WAddr=e, WData=001e#
# Writing: WAddr=f, WData=001f#
# ============================
# Reading: RData1=0000, RData2=0011#
# Reading: RData1=0012, RData2=0013#
# Reading: RData1=0014, RData2=0015#
# Reading: RData1=0016, RData2=0017#
# Reading: RData1=0018, RData2=0019#
# Reading: RData1=001a, RData2=001b#
# Reading: RData1=001c, RData2=001d#
# Reading: RData1=001e, RData2=001f#
# ============================
# Reading: RData1=0000, RData2=0011#
# Reading: RData1=0005, RData2=0013#
# Reading: RData1=0005, RData2=0015#
# Reading: RData1=0005, RData2=0017#
# Reading: RData1=0005, RData2=0019#
# Reading: RData1=0005, RData2=001b#
# Reading: RData1=0005, RData2=001d#
# Reading: RData1=0005, RData2=001f#
```

Simulations results are intuitive and clear for the implementation of RF. The first set of output is displaying what is written into RF. The second set of output is displaying the reading results from RF. For the last set, the testbench tests the simultaneously read and write Implementation by read first 2 words' data and flush the next register into a certain value 5.

4 Conclusions and Future Work

The RF design works properly based on two sets testing results.
For this report, the timing delays inside testbench should be paid attention, because some delay can cause instruction missing. The value of register 0, which is located at address 0, should always be 0.
One improvement is the approaching a versions that handles read-after-write hazards. Another one is the combination with ALU and later laboratory implementations.