CMPE 260 Laboratory Exercise 2

Register File

Andrei Tumbar
Performed: February 14th
Submitted: February 23rd

Lab Section: 1
Instructor: Moskal
TA: Jacob Meyerson

Lecture Section: 2
Professor: Cliver

By submitting this report, you attest that you neither have given nor have received any assistance (including writing, collecting data, plotting figures, tables or graphs, or using previous student reports as a reference), and you further acknowledge that giving or receiving such assistance will result in a failing grade for this course.

Your Signature: _____

Abstract

In this laboratory exercise, a parameterized register file was created. Generic parameters were used to control the number and size of registers included in the register file. A test-bench was written to assert the functionality of the register by creating a table of inputs and expected outputs. Expected outputs were checked against simulation outputs and comparison failures tripped an assertion failure.

Design Methodology

A block diagram of a 3-bit `LOG_PORT_DEPTH` 8-bit `BIT_DEPTH` register was created to simplify the design of the generic parameter register file.

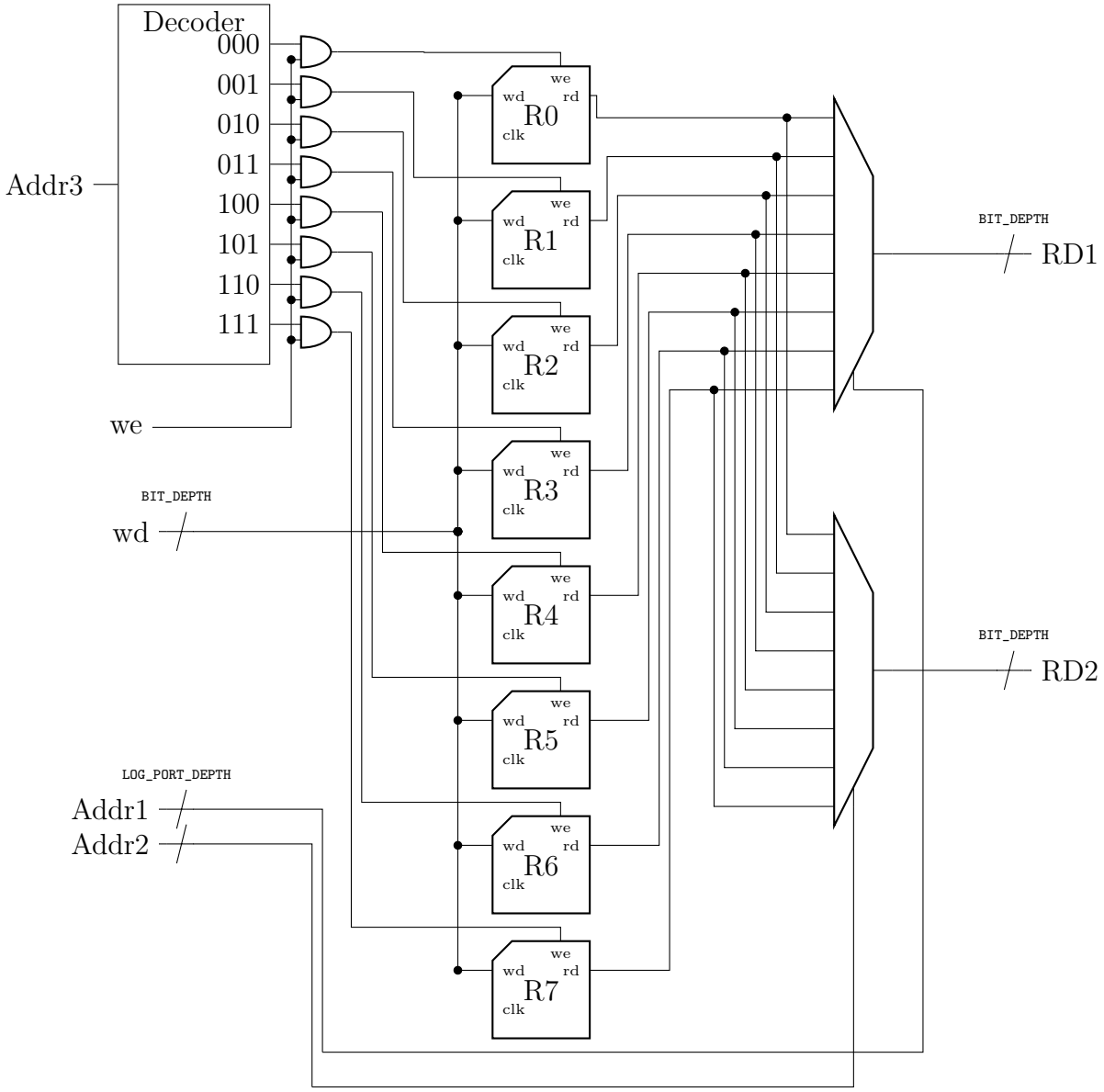

Figure 1: Register File block diagram

Each register in figure 1 work with 3 input signals. When the `we` signal is `HIGH`, the `wd` bus will be written to the register. The `clk` signal will drive the D-flip-flops inside the register that are used as memory. This signal is falling-edge driven. The register output signal `rd` will output the contents of the register on a falling-edge of the clock signal.

Figure 1 shows a block diagram of a register file with 8 registers. `Addr1` and `Addr2` control the data in the outputs `RD1` and `RD2`. These inputs select which register will we read from and appear in the output bus. The `Addr3` input will control which register to write to when `we` (write enable) is `HIGH`. The data written to the selected register will be the `wd`.

This register file can be implemented in VHDL by creating a table (array of arrays). The number of columns will be controlled by generic parameter `BIT_DEPTH`. Each row in the table will represent a single register. The number of registers is controlled by the `LOG_PORT_DEPTH` which is number of bits to address a single register. Therefore the number of registers will be $2^{LOG\_PORT\_DEPTH}$.
The zero register `R0` is designed to not be writable and always output `0` when read from.

Results & Analysis

To test the VHDL source code, a test-bench was written to analyse the behaviour of the circuit and all of its operations. A table was created with inputs and expected outputs. The expected were checked against simulation outputs to verify the fidelity of the register file implementation. When expected results differ from simulation results, an assertion failure is tripped marking the simulation with failure.
A behavioural waveform was generated.

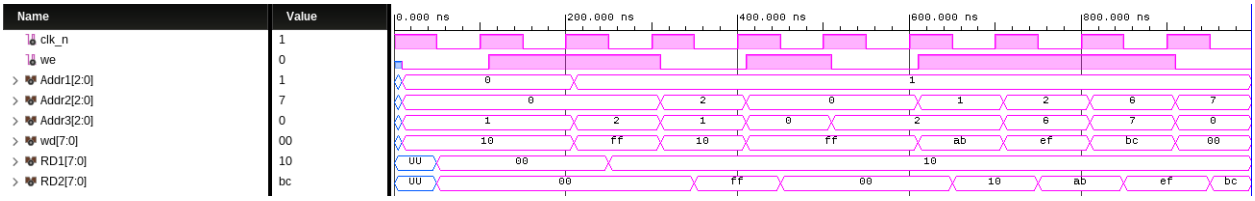

Figure 2: Screen capture of register file behavioural simulation

Figure 2 shows the waveforms generated from running a behavioural simulation of the VHDL test-bench. A table illustrating the inputs and outputs used to test the register was generated.

Table 1: Test cases used to test register file implementation

Addr1	Addr2	Addr3	we	wd	RD1	RD2
000	000	001	0	0x10	0x00	0x00
000	000	001	1	0x10	0x00	0x00
001	000	010	1	0xff	0x10	0x00
001	010	001	0	0x10	0x10	0xff
001	000	000	1	0xff	0x10	0x00
001	000	010	0	0xff	0x10	0x00
001	001	010	1	0xab	0x10	0x10
001	010	110	1	0xef	0x10	0xab
001	110	111	1	0xbc	0x10	0xef
001	111	000	0	0x00	0x10	0xbc

Looking at read addresses (`Addr1` and `Addr2`) in Table 1, whenever the address is `0`, the resulting read value is `0x00`. This meets the design specification as the zero-register should not be able to be written to and should always output `0`. Looking at the other test-cases, because they are run sequentially, the values of the registers hold the correct values after a write operation occurs.
Following the passing of all of the register file test cases, a post-synthesis timing waveform was generated.

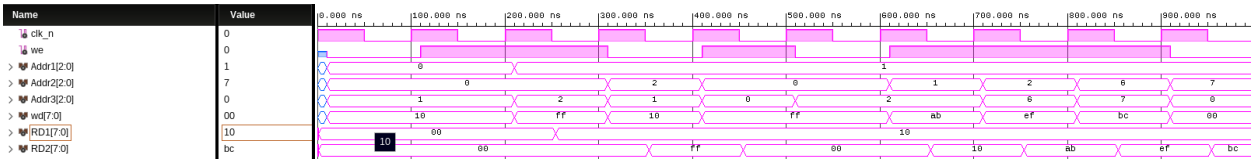

Figure 3: Post-synthesis timing simulation

The post-synthesis timing simulation uses the same test cases as the behavioural simulation. No assertion failure was tripped meaning that the results were valid.

Following the post-synthesis timing simulation, a post-implementation timing simulation was performed to test the design for the Baysis 3 FPGA.

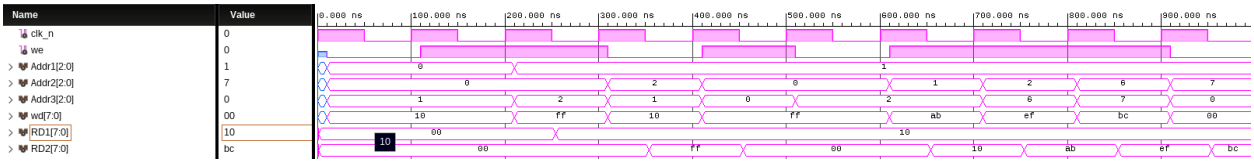

Figure 4: Post-implementation timing simulation

To map the pins to real IO pins on the Baysis board, a constraints file was written to map input pins to input signals in the register files. Output signals were mapped to LEDs.

Conclusion

This laboratory exercise introduced the concept of the register file and implemented a generically sized register file. The register file could read from two registers at once and write to one register when enabled. Behavioural and timing simulations were performed while also asserting the output from the simulation to verify integrity. This exercise was successful as a working implementation was created meeting all of the hardware specifications of the register file.

Demo results

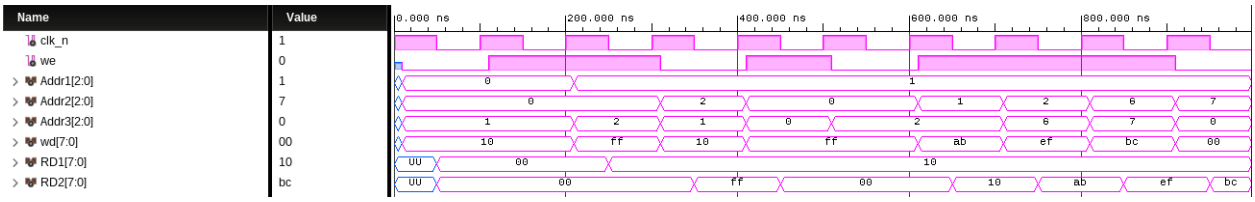

Figure 5: Behavioural simulation of register file

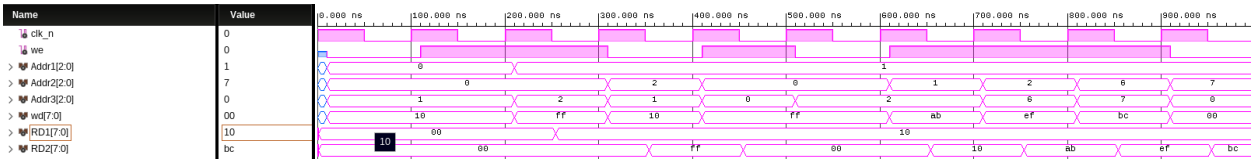

Figure 6: Post-synthesis timing simulation

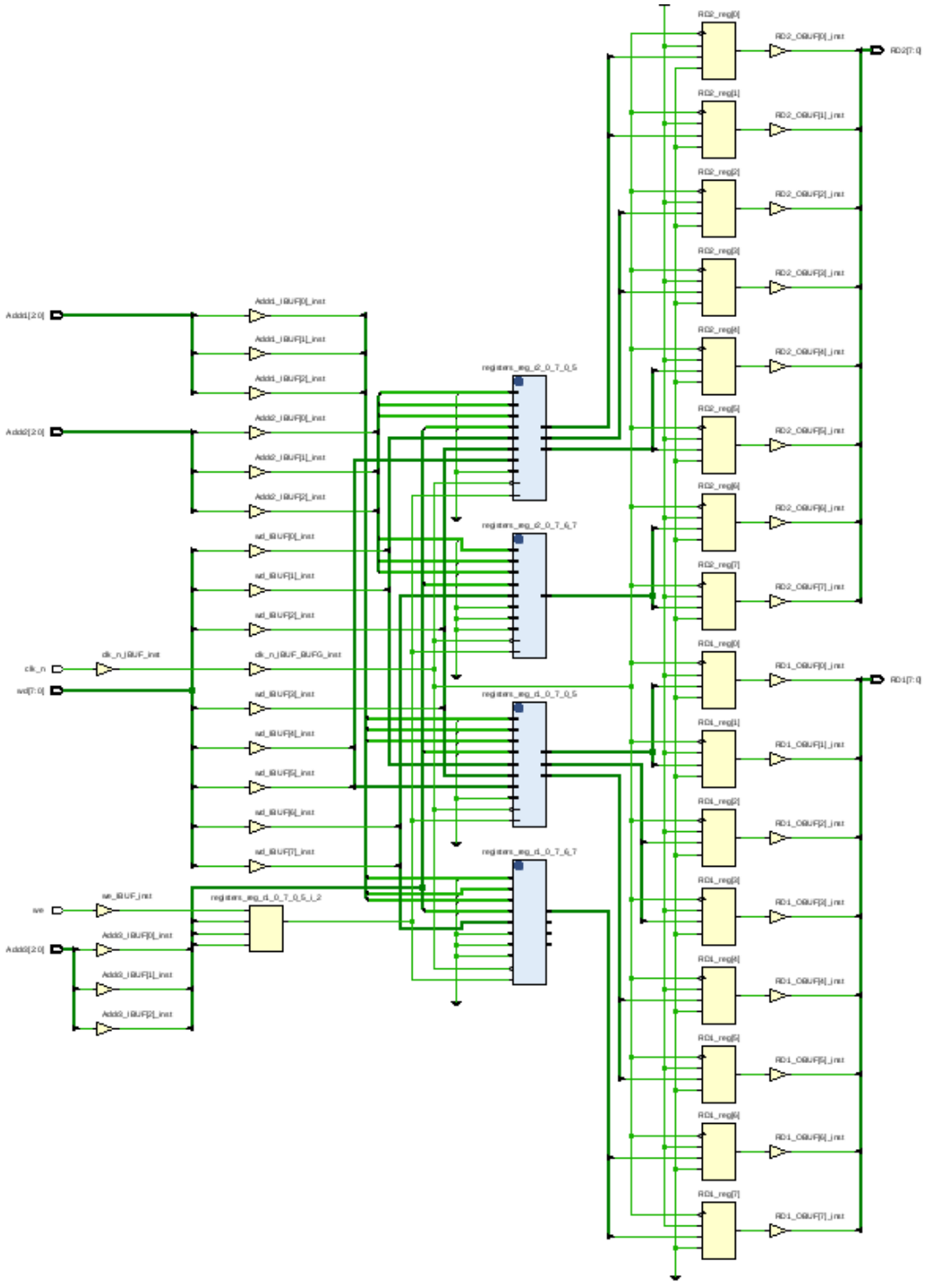

Figure 7: Synthesis Schematic

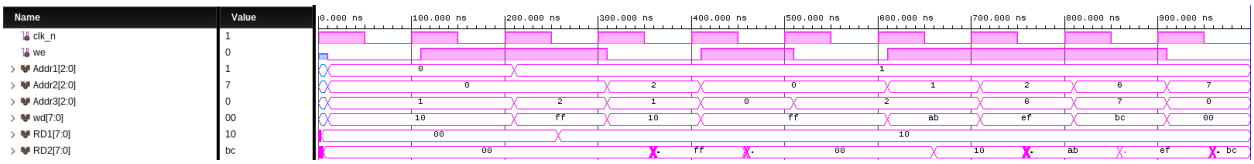

Figure 8: Post-Implementation Timing simulation

Resource	Utilization	Available	Utilization %
LUT	17	20800	0.08
LUTRAM	16	9600	0.17
FF	16	41600	0.04
IO	35	106	33.02

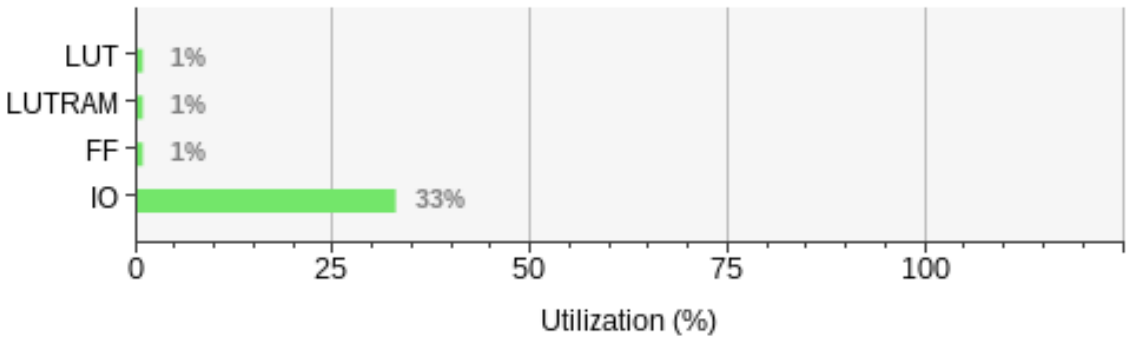

Figure 9: Implementation Utilization Report

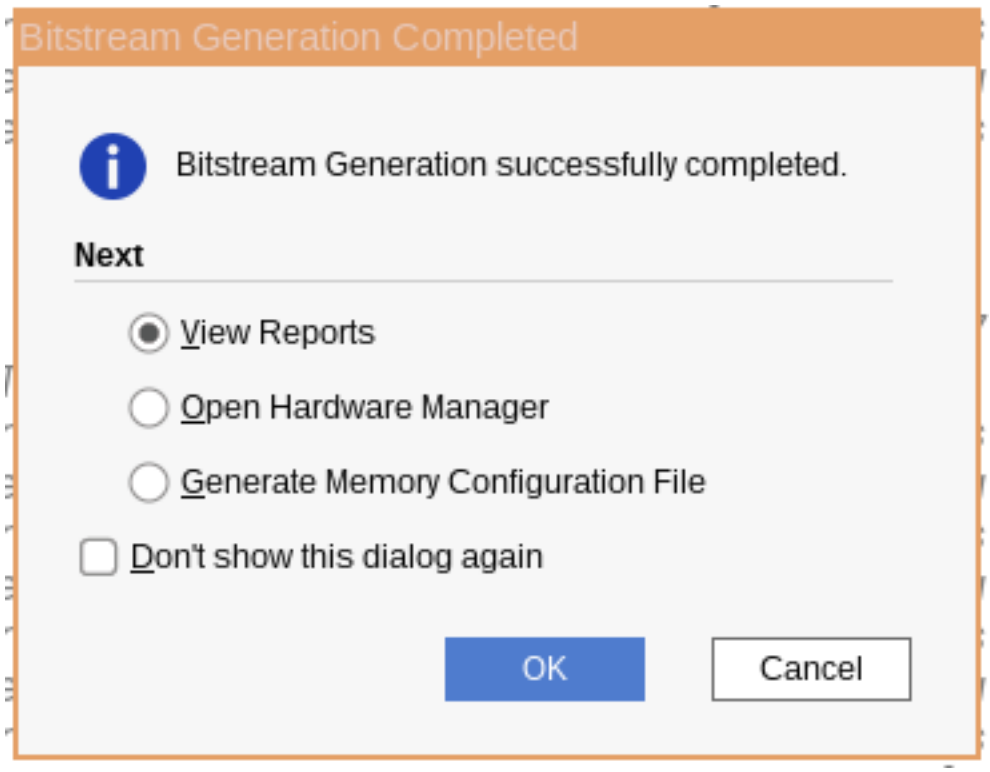

Figure 10: Screen-capture of successful bitstream generation