EE 314 Experiment 3 - Sequential Design with Registers

Objectives

The purpose of this experiment is to practice your sequential design skills with the combinational design knowledge you already have through one of the most fundamental structures used in computer architectures, a register file. As in the previous experiments, you will use the provided cocotb testbenches to test your design. Finally, in the experiment sessions, you will use your sequential design knowledge with a similar circuit and implement it onto the DE1-SoC Board, equipped with a field programmable gate array (FPGA) using several peripheral units, namely the switch inputs, button inputs, LEDs, and 7-segment outputs.

Please take your time and familiarize yourself with everything present in this preliminary work, as what is taught here will be used throughout the rest of the course.

Although the preliminary work grade is not included in the overall grade of the experiment,

1. **You need to get at least 50% from the report.**
2. **Your codes should pass at least half of the testbenches (1/2 for this experiment).**

Both are required to be eligible to attend the experiment session. We will run the testbenches automatically on your submitted codes for the second requirement.

1 Preliminary Work

In this experiment, you will design and implement a W-bit register file using Verilog. The register file consists of MUXes, a decoder, and register modules with many instances. You will also create a simple ALU to be used in the experiment sessions. The concepts introduced here are foundational and will recur throughout the course, so take the time to master them thoroughly.

You will submit two assignments for the preliminary work: A .pdf report and Verilog codes inside a .zip file.

To fulfill the requirements of this laboratory work, the following tasks should be performed. **Note: If necessary, you should only include important parts of the code in the PDF report. The full codes should be submitted separately as a .zip file.**

Your report must include RTL schematic views of the specified components, along with screenshots showing the successful execution of the testbenches.

All designs must be written in Verilog HDL and included in your submission as a ".v" file wherever required.

1.1 Reading Assignment

The Laboratory Manual, where the regulations and other useful information exist, is available on the ODTUClass course page. **Read the Laboratory Manual thoroughly**. Refer to the combinational Verilog lecture notes and the sequential Verilog lecture notes Part 1 as they contain everything about Verilog you need **for this experiment**. We also assume you are familiar with Chapters 1-5 of the EE348 course, as this experiment is complementary to that.

If you are unfamiliar with Verilog, Quartus, or cocotb, please refer to the course materials on ODTUClass.

1.2 Design Question Part 1: Register, MUX(Multiplexer) and Decoder (60% Credits)

In this part, you will design registers, a 3-to-8 decoder, an 8-to-1 MUX, and an ALU. **You must use the supplied template Verilog files wherever provided; do not change the file names, module names, or module I/O names.**

1.2.1 Creating the Project

1. Download SystemBuilder from the ODTUClass course page if you haven't already.
2. Open SystemBuilder (Section 4 of the DE1-SoC User manual.).
3. Set your project name to EE314exp3.
4. You only need 7-Segmentx6, LEDx10, Buttonx4, and Switchx10 for this experiment. **Note:** 7-segment interface module is provided on ODTUClass inside the "hexto7seg.v" file, it will also be supplied during the laboratory sessions.
5. Generate the project.

Notice that the SystemBuilder also creates a Verilog file with the same name as the project, "EE314exp3" in this case. This file will be useful when uploading your designs to the FPGA boards.

1.2.2 MUX (10% Credits)

Implement an 8-to-1 MUX, which has a W-bit data inputs/output, where W is a module parameter. **Generate the RTL schematic and add the screenshot of the RTL schematic to your report.**

1.2.3 Decoder (10% Credits)

Implement a 3-to-8 decoder. **Generate the RTL schematic and add the screenshot of the RTL schematic to your report**.

1.2.4 Registers (20% Credits)

For this step, you will implement two different W-bit registers, where W is a parameter specifying the data width of the parallel input to the register and output of the register. Note that the **registers should have a clock input**, even though it is not mentioned in the following items. **You should generate RTL schematics and add screenshots of them to your report.**

1. Simple register with synchronous reset (5% Credits): Implement a positive edge-triggered register with parallel load and a synchronous reset. If the reset signal is 1, the content of the register is cleared at the next rising edge of the clock. If the reset signal is 0, the content of the register is loaded with the input data at the next rising edge of the clock. The specifications of the simple register with synchronous reset are provided in Table 1.

Table 1: Simple Register with Reset

Reset	Operation
0	$OUT \leftarrow DATA$
1	$OUT \leftarrow 0$

2. Register with synchronous reset and write enable (15% Credits): Implement a positive-edge triggered register with parallel load, write enable, and synchronous reset. If the reset signal is 1, the contents of the register are cleared at the next rising edge of the clock. If the reset signal is 0 and the write enable signal is 1, the contents of the register are loaded with the input data at the next rising edge of the clock. Finally, the register retains its content if the reset signal is 0 and the write enable signal is 0. The specifications of the register with synchronous reset and write enable are provided in Table 2.

Table 2: Register with synchronous reset and write enable

Reset	Write Enable	Operation
0	0	Retain
0	1	$OUT \leftarrow DATA$
1	X	$OUT \leftarrow 0$

1.2.5 Arithmetic Logic Unit (ALU) (20% Credits)

Implement a W-bit ALU for 2's complement arithmetic using the provided template file "ALU.v", where W is a parameter specifying the data width of its operands. The ALU should have 2 data inputs, two W-bit operands `A` and `B`. It should have 3 operations controlled by a 2-bit control input. It also should have a W-bit result output. The specifications of the ALU operations are provided in Table 3. **You are encouraged to use combinational "always blocks" for the ALU logic.**

1. Generate the RTL schematic and **add the screenshot of the RTL schematic to your report.(10% Credits)**
2. Write a cocotb testbench for the ALU module. Your testbench should be exhaustive. The basic idea of a testbench is to give a test input and compare the acquired output with the expected

true output. You should check the Experiment 2 Manual for the cocotb references. The provided testbenches should also be a good reference point. You can include and utilize the "Log_Design" given in Experiment2Materials function for debugging. **Put your testbench code (without the function declarations for "Log_Design" and "to_hex" if used) in the appendix of your report. (5% Credits)**

3. Verify the module by running your own testbench and **add the screenshot of a successfully passed test to your report. (5% Credits)**

We will check the ALU module with our own testbench for its functional correctness after the submissions.

Table 3: ALU Operation Control

ALU Control [1:0]	**ALU Operation**	**Symbol**
00	Addition	$A + B$
01	SubtractionAB	$A - B$
10	Move Not	$\neg B$
11	Undefined	0

1.3 Design Question Part 2: Register File (40% Credits)

Using the **decoder**, **MUX**, and **register** modules defined in Section 1.2 with the minimum amount of additional logic you need, implement a parameterizable register file consisting of eight registers, each W bits wide. A template file "Register_File.v" is provided. An example 4-register version is shown in Figure 1.

I/O Ports

- `CLK`: clock input (positive-edge triggered)
- `Reset`: synchronous reset (active high, clears all registers)
- `Write_Enable`: write enable
- `Destination_Select[2:0]`: 3-bit address selecting which register to write
- `Source_Select_0[2:0]`, `Source_Select_1[2:0]`: 3-bit addresses selecting which registers to read
- `Data[W-1:0]`: write data input
- `Out_0[W-1:0]`, `Out_1[W-1:0]`: read data outputs (combinational)

Behavioral Specification

1. *Write Operation*: On the rising edge of `CLK`, if `Write_Enable`=1 and `Reset`=0, the value on `Data` is written into the register addressed by `Destination_Select`. All other registers retain their previous contents.
2. *Read Operation*: Independently of `CLK`, whenever `Source_Select_0` or `Source_Select_1` changes, `Out_0` and `Out_1` are driven by the contents of the selected registers. Thus, reads are fully asynchronous.
3. *Reset*: If `Reset`=1 on a rising clock edge, all eight registers are synchronously cleared to zero, regardless of `Write_Enable`.

According to the aforementioned desired operation of the register file:

1. Design and sketch (on paper or digitally) a schematic for a register file design using your decoder, MUX, register modules, and the minimum amount of additional logic. For your sketch, you may present your modules with boxes. **Add your sketch to your report (15% Credits)**

2. Implement your design in Verilog HDL using the provided template.
3. Generate RTL schematics for the "Register_File" using $W = 4$ and **add the screenshot of the RTL schematic to your report. (10% Credits)**
4. Test the "Register_File" module using the testbench supplied on ODTUClass and **add the screenshot of a successfully passed test (a single test screenshot suffices) to your report. You should test with multiple W values (4,8,12) to ensure correctness.** Testbench automatically adjusts for your W value, so you only need to change your ".v" files. **(15% Credits)**

You should include a pen and paper sketch (or digital drawing) of your design in the report.

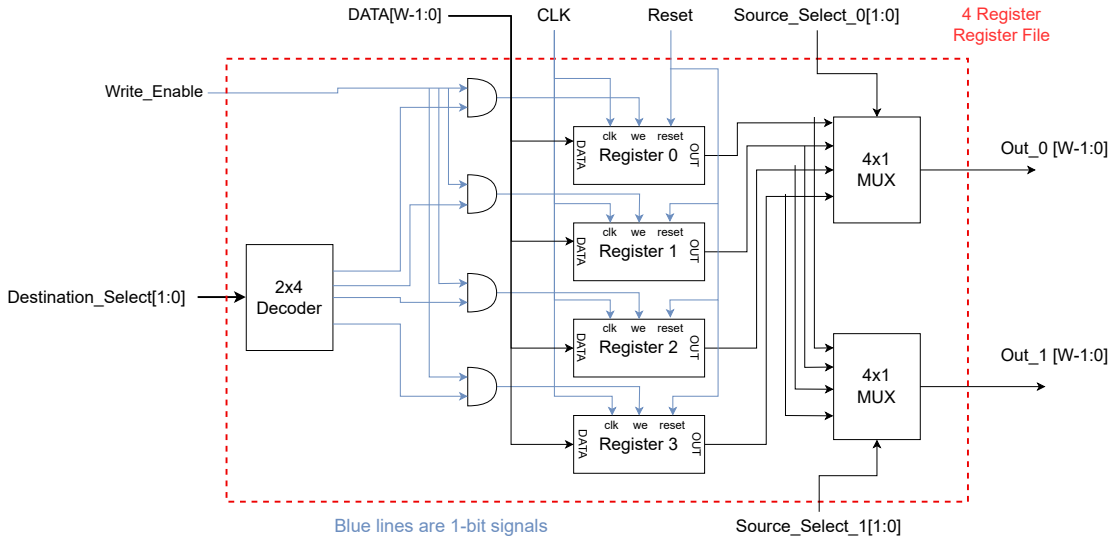

Figure 1: An example 4-register Register File. Note that you are asked to implement an 8-register version.

2 Experimental Work

In the experiment session, you will have two parts, a preliminary work demonstration and a design task.

2.1 Preliminary Work Demonstration (20% Credits)

Run the cocotb testbench for the "Register_File" module once again (with W specified by the lab instructor) and show it to your laboratory instructor.

2.2 Experiment Session Design Task (80% Credits)

The experiment sessions will require implementations of various sequential circuits, using but not limited to all modules from the preliminary work. Details will only be given when the experiment sessions start.

A Parts List

DE1-SoC Board