# CS M152A Lab 1

## Sequencer: An Example Project

*In this lab, you will study an example project and do exercises based on the project.*

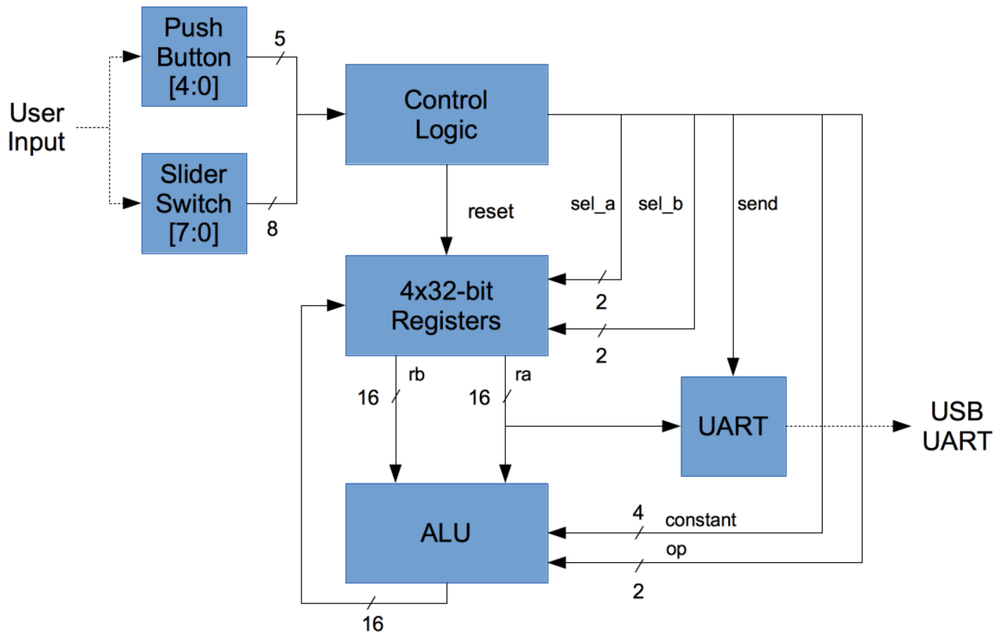
## Introduction

The first part of Lab 1 helps you get familiar with the FPGA design flow. You will get a general idea of how an FPGA design project is created and developed in Xilinx ISE. You will go through important steps in FPGA design flow including simulation, synthesis, place and route, and bit-stream generation. You will then interact with the programmed FPGA and finish a simple task based on it.

The second part of Lab 1 involves studying the Verilog module and test bench in the example project. The project is a good reference for future design projects, as you can learn the styles, formats, structures of code, etc. from the project. You can also learn useful techniques, like clock dividers and debouncers, from the project. There will be questions based on the code, and there will also be a few tasks that require modifying the original code.

## Example Project Introduction

The example project is an adder/multiplier sequencer. It has four 16-bit general purpose registers. It can perform add or multiply instructions using registers as operands. The results are stored into register files.



Project Diagram

## Sequencer Operation

To operate the sequencer, the user enters instructions using the switches and push buttons:

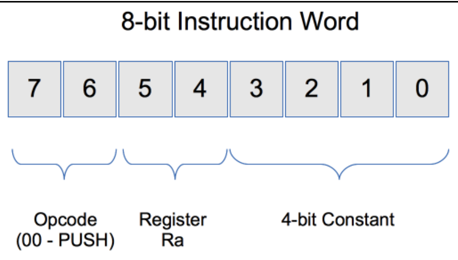
- 8 switch positions represent a single 8-bit instruction (up: 1, down: 0)
- Push center button to enter and execute the instruction
- The push button BTNR (right button) resets values of all registers to 0

In addition, the 8 LED indicators shows the number of instructions executed since the last reset, in binary

## Sequencer Instructions

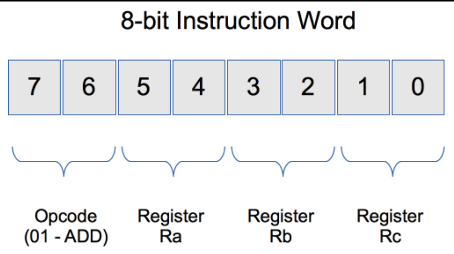
PUSH (00) instruction left-shifts the target register by 4 bits and "pushes" the new constant in. Example:

- R0 starts with value 0x55AA
- PUSH R0 0xB
- Now R0 is 0x5AAB



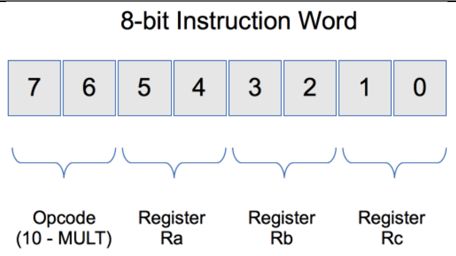
ADD (01) instruction adds Ra and Rb and stores the result to Rc. Addition is performed in unsigned integer arithmetic. Example:

- ADD R0 R1 R3
- R0 + R1 à R3



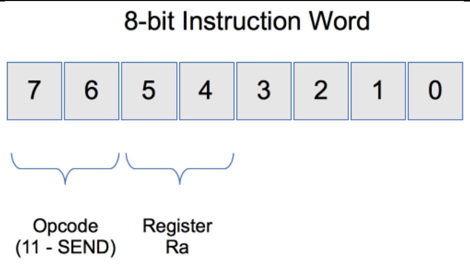
MULT (10) instruction multiplies Ra and Rb and stores the result to Rc. Inputs are assumed to be unsigned integers. Only the lower 16-bit results are retained. Example:

- MULT R3 R0 R2
- R3 * R0 à R2



SEND (11) instruction sends the content of Rt to the UART for display. The 16-bit value is converted to ASCIIHEX and appended with a newline character. Example:

- R1 has a value of 0x1234
- SEND R1 causes the UART to send "1234\n"



## Exercises

1. Implement the project according to the *FPGA Design and Implementation Fundamentals*. Please explain in your report how the steps in the FPGA design workflow correspond to the steps in the actual implementation using the Xilinx ISE and simulation tools.
2. Translate the following "program" into binary instructions. Demo the UART console output to your TA after you finish.
   - PUSH R0 0x4
   - PUSH R0 0x0
   - PUSH R1 0x3
   - MULT R0 R1 R2
   - ADD R2 R0 R3
   - SEND R0
   - SEND R1
   - SEND R2
   - SEND R3
3. Write a "program" (instruction) using file input to print out the first 10 numbers of the Fibonacci series. Demo the simulation and the UART console output to your TA after you finish.
4. Finish exercise 1. Include answers in the report.
5. Finish exercise 2. Include answers in the report.

## Report

The report format described in the syllabus does not apply to Lab 1. The report for Lab 1 should include only the exercises from the section above.

## CS152A Lab 1 Exercise 1

Go through the tutorial on FPGA design and implementation, and went through the whole process using Xilinx's toolchain. In this section, you will explore the example's simulation process to learn more about behavioral simulation, debugging, and design.

Answer the following questions as much as you can and include the answers in your lab report.

### Clock Enable

1. In the nexys3.v file, there is a signal/reg named clk_en. Read the section of code that's relevant to clk_en and try to understand what this signal does.
2. Add clk_en to the simulation's waveform tab and then run the simulation again. Use the cursor to find the periodicity of this signal. Capture a waveform picture that shows two occurrences of clk_en, and include it in the lab report. Indicate the exact period of the signal in the report.
3. What is the value of clk_dv singal during the clock cycle that clk_en is high?
4. Draw a simple schematic/diagram that illustrates the logical relationship amongst clk_dv, clk_en, and clk_en_d signals. Ask your TA for an example if you have never drawn a digital system diagram before.

### Instruction Valid

1. Now move on to the signal inst_vld. Read the relevant code and use the simulation as your aid, answer the following questions in your lab report.
2. Write down the first simulation time interval (exact number) during which the expression inst_vld = ~step_d[0] & step_d[1] & clk_en_d evaluates to 1.
3. What is the purpose of clk_en_d signal when used in expression ~step_d[0] & step_d[1] & clk_en_d? Why don't we use clk_en?
4. Include a waveform capture that clearly shows the timing relationship between clk_en, step_d[1], step_d[0], btnS, clk_en_d, and inst_vld.
5. Draw a simple schematic/diagram to illustrate the logical relationship amongst the signals above.

### Register File

1. The sequencer's register file is located in a file called seq_rf.v. It stores the values of the four registers. Take a look at the source code and see if you can understand how it is implemented. Answer the following questions in the lab report.
2. Find the line of code where a register is written a non-zero value. Is this sequential logic or combinatorial logic?
3. Find the lines of code where the register values are read out from the register file. Is this sequential or combinatorial logic? If you were to manually implement the readout logic, what kind of logic elements would you use?
4. Draw a circuit diagram of the register file block.

5. Capture a waveform that shows the first time register 3 is written with a non-zero value.

## CS152A Lab 1 Exercise 2

In the previous exercise you were given a tutorial of FPGA design and implementation, and went through the whole process using Xilinx's toolchain. In this session, you will focus more on extending the existing testbench that's provided to you.

### Nicer UART Output

The existing testbench program contains a uart model that announces each byte that's received by the model. While generic, the output of this model is somewhat hard to read. Modify model_uart.v such that the per-byte output is surpressed. Instead, model_uart.v shall output one line at a time after a carriage-return character ('\r') is received. For example, print out "0003\n" instead of "0\n"  "0\n" "0\n" and "3\n" for the first line of UART output.

### An Easier Way to Load Sequencer Program

The existing sequencer testbench sends a static sequence of instructions to the UUT (Unit Under Test) after the reset. In the last session you have observed this in the waveform viewer how the instructions are sent. Answer the following questions in your lab report.

1. Identify the part of the tb.v where the instructions are sent to the UUT.
2. Which user tasks are called in this process?

Now we would like to change the static set of instructions. Instead, we will be loading instructions from a text file. The format of the file is the following:

1. The name of the file is "seq.code"
2. The file is up to 1024 lines long.
3. The first line of the file contains a binary number that indicates how many instructions are included in this file.
4. Each of the remaining (n-1) lines contains a single instruction in binary.

For example, here's the file-equivalent of the simple sequencer instructions currently in use:

Line 1: 1001

Line 2: 00000100

Line 3: 00000000

Line 4: 00010011

Line 5: 10000110

Line 6: 01100011

Line 7: 11000000

Line 8: 11010000

Line 9: 11100000

Line 10: 11110000

**Modify the testbench such that it loads seq.code into an array, and executes every instruction in the file.**

Hint: for file I/O, you may use the built-in Verilog system task $readmemb (google Verilog quick reference), or the c-like $fopen and $fscanf tasks.

## Fibonacci Numbers

Now that you have an easier way to program the sequencer from simulation, design a sequence of instructions such that the first 10 numbers of the Fibonacci series is printed from the UART.