CS152B Lab 1 – ALU + Register File

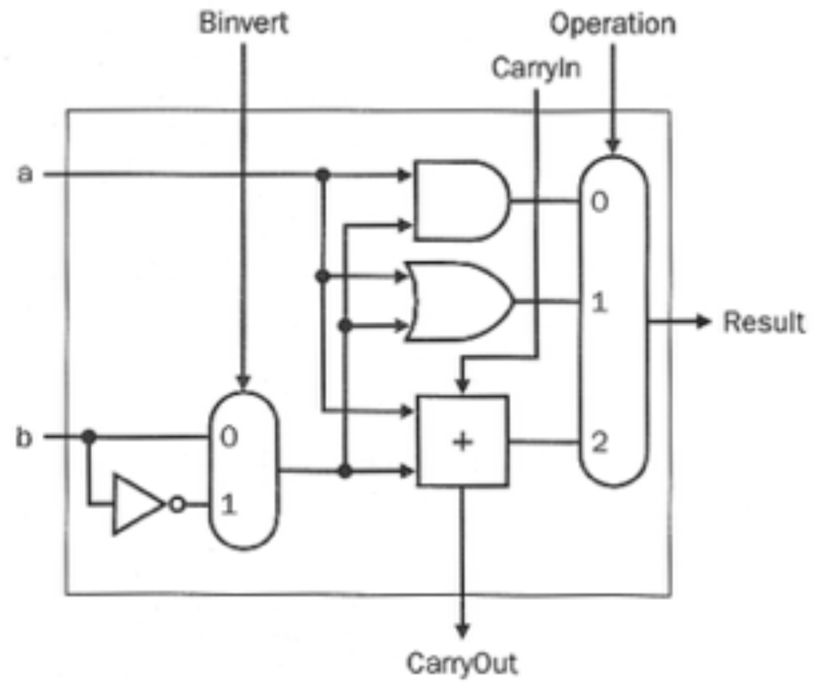
Required Equipment

No FPGA hardware is required for this lab. Simulation only.

PART 1 – 1 BIT ALU

An ALU is a hardware block that can perform one of several operations based on the control signals that are asserted. For example, the ALU could perform an addition of the two inputs (A,B) if the "Operation" input is '0', and would subtract the two inputs of the "Operation" was to set to '1'. This configurable arithmetic unit is very common in digital circuit designs.

In Part 1, you are asked to design and implement a 1-bit asynchronous ALU in the Xilinx Simulation environment. For your convenience, the schematic has been provided below. Remember the steps in the ISE tutorial to create the designs for this lab.



DESIGN: In general, start by considering each of the components you will need for your design and implement each one separately before putting everything together. As previously shown, our ALU consists of 6 components:

AND gate	OR gate	NOT gate	2-1 Mux	3-1 Mux	Full Adder

The AND, OR, and NOT gates are provided by the Xilinx libraries, but the other components are not. Thus, we will need to define hardware modules for each of the remaining components. Your final 1-bit ALU should be a separate standalone module implemented with no behavioral code. Once your design is complete, you must write a comprehensive testbench to evaluate the performance of your ALU with different inputs and operations. However, a demo is not required for Part 1.

2-to-1 Multiplexer: Behavioral Model

Figure 1 – Verilog 2-to-1 MUX. Asynchronous and synchronous (incomplete).

```
module  asynch_mux(in0, in1, sel, out);


input din_0, din_1, sel;
output mux_out;
reg  mux_out;

always @ (sel or din_0 or din_1)
begin
  if (sel == 1'b0) begin
      out <= in0;
  end else begin
      out <= in1;
  end
end

endmodule
```

```
module  synchronous_mux(??);

input din_0, din_1, sel;
output mux_out;
reg  mux_out;

always @ (??)
begin
  if (sel == 1'b0) begin
      out <= in0;
  end else begin
      out <= in1;
  end
end

endmodule
```

This is one possible implementation of a 2-to-1 multiplexer. You can use this to design a large multiplexer hierarchically. In other words, create a 4-to-1 multiplexer using two 2-to-1 multiplexers, and so forth. Note that this particular implementation is asynchronous. Think about what changes would be required to create a synchronous multiplexer.

Full Adder: Structural Model
In this section, you are required to implement the full adder. Create a file called "addbit.v" and add it to your project. Be sure to include the following ports: cin, a, b, s, cout. Below, code is provided for a 1-bit adder.

Figure 2– Verilog 1-bit adder (asic-world.com)

```
module addbit (a, b, ci, sum, co);

input a;
input b;
input ci;
output sum;
output co;
//Port Data types
wire  a;
wire  b;
wire  ci;
wire  sum;
wire  co;

assign {co,sum} = a + b + ci;
endmodule
```

As before, you can use this module to build a structural 16-bit adder using hierarchical design. For example, refer to the code below which shows how you can use multiple 1-bit adders to create a larger adder.

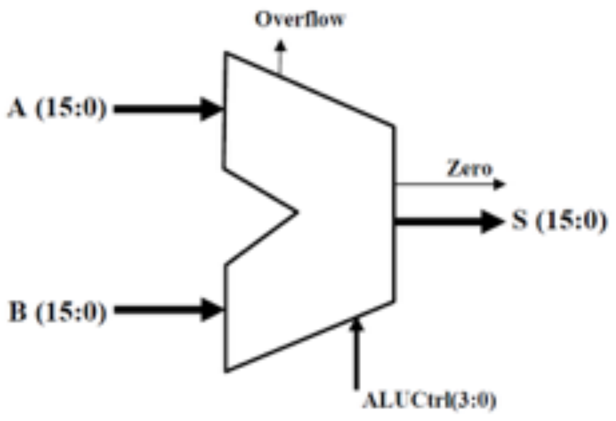
Figure 3– Verilog Hierarchical Adder (asic-world.com)

```
module adder_explicit (result, carry, r1, r2, ci);

// IO Port Declarations
input [3:0] r1;
input [3:0] r2;
input ci;
output [3:0]  result;
output carry;
```

```
wire [3:0] r1;
wire [3:0] r2;
wire ci, carry;
wire [3:0] result;
wire c1, c2, c3;

addbit u0 (.a (r1[0]), .b (r2[0]), .ci (ci), .sum (result[0]), .co (c1));
addbit u1 (.a (r1[1]), .b (r2[1]), .ci (c1), .sum (result[1]), .co (c2));
...
```

PART 2 – 16-BIT ALU

In this section, you are asked to design and build a 16-bit asynchronous ALU using the tools provided by the Xilinx Integrated Software Environment (ISE). Your goal is to implement an ALU that can perform all the operations listed in the table below:



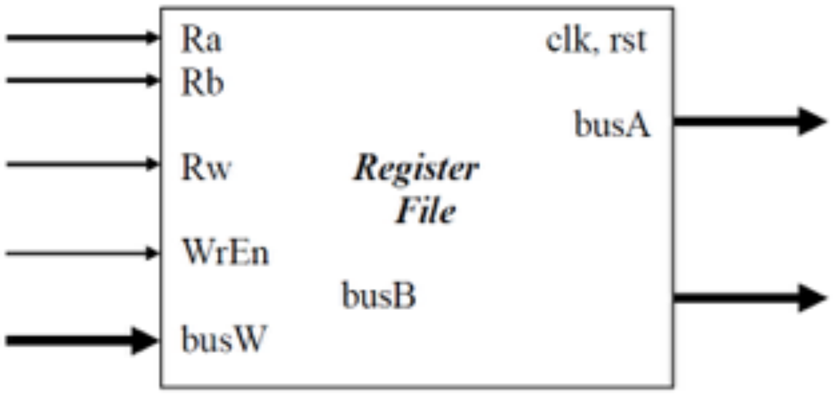
ALU Ctrl	Description
0000	Subtraction
0001	Addition
0010	Bitwise OR
0011	Bitwise AND
0100	Decrement
0101	Increment
0110	Invert
1100	Arithmetic Shift Left
1110	Arithmetic Shift Right
1000	Logical Shift Left
1010	Logical Shift Right
1001	Set on Less than or Equal

Some additional details to keep in mind while working on your design:

- A and B inputs of the ALU are signed numbers coded in the two's complement system
- For all operations, the output signal Zero is '1' when S is equal to zero, otherwise Zero is '0'.
- For arithmetic shift operations, overflow is '1' when shift operation changes the sign of the original number.
- In shift operations, you should shift A by B.
- You may only use behavioral Verilog to design your shifter

Hint: Be sure to note the differences between logical shift and arithmetic shift, paying particular attention to what happens to the sign bit. Make sure to do a research and understand the difference between left/right logical and arithmetic shift operations.

PART 3 – REGISTER FILE



You are asked to build a register file with 32 16-bit registers, two read ports, and a single write port. The register file should be able to support two concurrent read operations and one write operation. For your convenience, the top-level schematic is provided below.

- The signals *busA*, *busB*, and *busW* are 16-bits wide, while the *Ra*, *Rb*, and *Rw* are 5-bits wide.
-Ra, Rb, and Rw represent the address of the register that is being read or written to. The address will be a 5-bit number.
-BusW is the data that we are writing.
- The register file also requires clock and reset pins.
- You may use **either** structural or behavioral Verilog to implement the register file.
- In case of concurrent read/write operations to the same register, the output must reflect the new value.
- The reset signal is a synchronous reset which has higher priority than all write signals. If reset is asserted at the rising edge of the clock, all of the registers are set to all 0's.

DELIVERABLES, DEMO, REPORT

Please demo (1) Your final ALU design and (2) Your register file. Your demo must include a comprehensive and well written testbench showing different use cases and ALU operations.

This lab will also require a final report, based on the specifications outlined in the syllabus. In addition, please answer the following questions in your report:

1) What is the difference between structural and behavioral Verilog? Please provide an example of a structural and behavioral implementation of a multiplexer.
2) What is the difference between an asynchronous and synchronous Multiplexer? Please provide a brief explanation on how you could implement both using behavioral Verilog.
3) What is the difference between an arithmetic and logical shifter?
4) Assuming that you did NOT use Behavioral Verilog to implement an arithmetic shifter, how could you design one from scratch? Please include a simple diagram.