ECE4530 Digital Hardware Design ***Project 01***

ALU Design

Overview

In this Project you will design an Arithmetic and Logic Unit (ALU) that implements 8 functions, as shown in *Figure 1*. The unit should operate on unsigned numbers only and running at a 25kHz clock. The resulting output will be displayed on the 4-leds on the Zybo board.

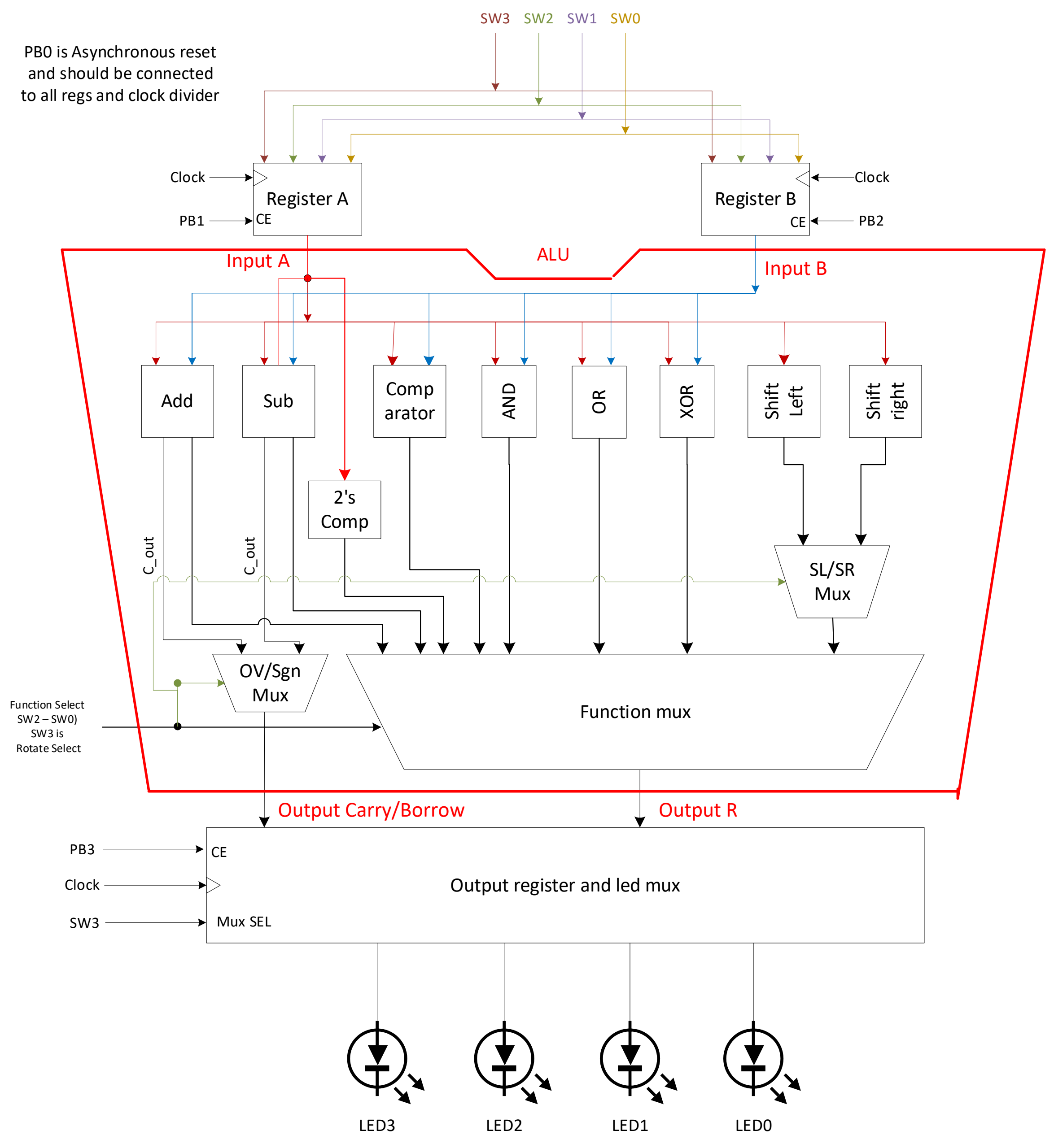

Figure 1 - ALU Block Diagram

Procedure

1. Update your 2-bit adder design from Activity 01 to an N-bit Ripple Carry adder and call it "*n_bit_adder*" using parameters, with a default value of 32. Then use generate statements, explained in the project information document, to generate the 4-bit adder needed in the ALU design.
   - Perform functional simulation for the *n_bit_adder* module to verify its correct functionality before proceeding.
2. Design a Verilog module for the *Two's Complement* module and call it "*n_bit_twos_comp*". Refer to the project information document.
   - Perform functional simulation for *n_bit_twos_comp* module to verify correct functionality before proceeding.
3. Design a Verilog module for a 2's complement-based n-bit subtractor to perform the subtraction operation (A – B) and call it "*n_bit_sub*". This type of subtractor uses an adder, a 2's complement unit, and inverters and a 2-to-1 Mux as explained in the project information document.
   - Perform functional simulation for the *n_bit_sub* module to verify its correct functionality before proceeding.
4. Design a Verilog module for an n-bit comparator and call it "*n_bit_compare*". The module compares the two numbers A and B and generates three signals: EQ signal if A = B; GT signal if A is greater than B, LT signal if A is less than B. Design it as explained in the project design document.
   - Perform functional simulation for the *n_bit_compare* module to verify its correct functionality before proceeding.
5. Design a Verilog module for a generic n-bit register with an active-low asynchronous reset and Clock Enable inputs and call it "*n_bit_reg*"
   - Perform functional simulation for the *n_bit_reg* module to verify correct functionality before proceeding.
6. Create the top module "*alu.v*" that has the same ports as the ALU shown in *Figure 1* (inside the red box). Inputs A and B, and the output R should be defined as generic n-bit wide vectors.
   - Instantiate each one of the components within the *ALU architecture* once. Make sure they are instantiated to only operate on 4 bits. Connect the outputs of the Adder, Subtractor, and two's Complement internal units to the inputs of the *"func_mux*". Make sure to follow the order of the mux inputs as given in *Figure 1*. Connect the overflow bit of the adder and the sign bit of the subtractor modules to the 2-to-1 MUX, and make sure to assign the least significant bit of the *Function Select* ALU input as the select input to the 2-to-1 MUX.
   - For the rotate operations do not use the shift operation provided in Verilog but rather use concatenation and bit assignments. Connect the outputs of the rotate left and rotate right modules to the inputs of 2-to-1 SL/SR Mux. Make sure to assign the most significant bit of the *Function Select* ALU input as the select input to the 2-to-1 MUX.
   - For the remaining logic operations, use continuous assignments then connect appropriate signals to the "*function_mux*" inputs in the same order given in *Table 1* and *Figure 1*.

- Write a testbench for the ALU. Verify that it works as intended before proceeding. Carefully select critical test cases to verify all functions of your design and make sure to comment on them in your testbench with the expected results.

Table 1 - Output Mux Select and ALU Functions

Function Select	Instruction	Function
000	Add	Ripple Carry Adder module
001	Sub	Ripple Carry Subtractor Module
010	Compare	Binary Comparator Module
011	2's Complement	Two's Complement module
100	AND	Output <= A and B;
101	OR	Output <= A or B;
110	XOR	Output <= A xor B;
111	Rotate Right/Left	Rotate input vector 1 bit

7. Write a wrapper and call it *"ALU_Top.v"* that instantiates and interconnects, the Alu of *Figure 1*, a register for input A, another register for input B, an output register (called "*out_reg*"), and LED_Mux called "*led_mux*". The wrapper port definition is:

```
module ALU_Top(
                    input        clk,
                    input  [3:0] pb,
                    input  [3:0] sw,
                    output [3:0] led
                  );
```

- Note: Do not simulate the wrapper.
- Map the parameters for the instantiated ALU, register A, register B and Led Mux to 4 bits; and the "*out_reg*" parameters to 5. The "*led_mux*" is a 2-1 mux that outputs the 4 bits of the function mux or the ov/sgn bit to the LEDs depending on a select signal.
- The ALU will be using a 25 Khz clock. Therefore, include the frequency divider module, you previously designed, and change the parameter during the instantiation to generate this clock signal, then use its output as the operating system clock.
- use the edge detector module designed previously for detecting the positive edge of each Pushbutton. It should be named "*sw#_dge_to_pulse*". ***Make sure that this component work on the negative edge of the clock as previously designed and explained.***

8. Synthesize the design. Review the elaborated schematic to make sure the RTL design synthesized mimics the RTL design shown in *Figure 1*. Annotate it in reference to the designed components

9. Use the *I/O Planning* tool or edit the constrains file of your board to enforce the following connections:
   - Use PB0 as the asynchronous reset (not shown in figure**). Since the PB is normally low and you are using a low-active reset, you need to invert this signal before using it.**
   - [SW3 – SW0] has a dual function
     i. [SW3 – SW0] represent the data input
     ii. SW3 represent the rotate select (0 for SL and 1 for SL)
     iii. [SW2 – SW0] represent the ALU function select
   - Use PB1 and PB2 to load register A and register B with switch data, respectively.
   - Set the function on SW2 – SW0.
   - Use PB3 to load the LED Register, and SW3 to switch the display between the output and ov/sgn. Make sure to select the proper operation for the rotation first.
   - LED3 – LED0 represent the output. The ov/sgn is shown on LED0 when SW3 is set to 1.
10. Program the ZYBO board with the completed wrapper and test it using the following test cases, Take a narrated short video when testing each case, explaining what are you testing such as, entering input A, entering input b, … etc

a)	Add:	1011 + 1010
b)	Subtract:	1010 – 0011 and 0101 - 1011
c)	Two's Complement:	0010 and 1001
d)	AND, OR, XOR:	0101 , 1101
e)	Rotate left:	1101
f)	Rotate right:	1011

Deliverables

Create a readme test file that contains the name of the files submitted and the contents of each file (remember not to submit the whole project). Your Project_01 deliverables should contain the readme file; all commented Verilog design files; the testbench source for the top design only commented and all test cases are included; constraints file; bitstream file; and a report containing all simulations and schematics. Make sure that simulation waveforms are notated and zoomed to be visible for what you are trying to show for the six test cases required, and clearly visible. Also explain the relationship between the resource utilization and the schematic of your design.