

---

# ECE 436 Laboratory 1

---

## REGISTER AND ALU DESIGN

---

### Description:

For this laboratory, you will design a 4-bit register and a 4-bit ALU, larger versions of which will eventually be incorporated into your Jackal processor. You will explore various implementations and tool settings to increase your understanding of design techniques, VHDL, and the CAD tools.

The register should asynchronously reset to all zeros when "reset" goes low, and it should stay all zeros as long as "reset" is low. When "reset" is high, the register should take the value at its input at the rising clock edge when "enable" is high, and the register should hold its value when "enable" is low.

The ALU should be capable of performing all of the arithmetic and logical operations defined in the Jackal ISA. Therefore, the ALU should have control signals (e.g. the instruction opcode) that specify the operation to be performed. For the shift operations, you may implement a single-bit shifter that feeds the output back into the ALU (through a register) for the appropriate number of shifts, or you may implement a barrel shifter capable of shifting any number of bits in a single pass (which requires control signals from the instruction). The former requires little area but multiple clock cycles, and the latter requires more area but only a single cycle. (This is an example of the many tradeoffs that must be considered throughout the design process.)

### Process:

When designing the components, design them both as single components and as bit-sliced components. For example, design the register as a single 4-bit register (using a single VHDL module) and as 4 1-bit registers (e.g. using the schematic editor to insert 4 1-bit registers). Do the same for the ALU.

Feel free to experiment with some of the operators (+, -, etc.) built into VHDL, but make sure you understand how they work and how the synthesis tool treats them. (Look at the references listed on the Synthesizable VHDL handout for more information about operators and other VHDL issues.) Note that the operators only work on `std_logic_vector` signals. For example, you may use the following code to design an adder:

```
entity add4b is
    port
    (
        a    : in  std_logic_vector(3 downto 0);
        b    : in  std_logic_vector(3 downto 0);
        s    : out std_logic_vector(3 downto 0)
    );
end add4b;

architecture behavioral of add4b is
begin
    s <= a + b;
end behavioral;
```

When synthesizing your designs, try different synthesis options. Right-click on "Synthesize" in the Processes for Current Source window and select "Properties...". For "Optimization Goal", try both "Speed" and "Area". (You may also try both "Normal" and "High" for "Optimization Effort".) After each synthesis run, generate a post-synthesis report by double-clicking on "View Synthesis Report" under "Synthesize". In the report, focus especially on the "Device utilization summary" (noting the number of "Slices", "4 input LUTs", and "Slice Flip Flops") and the "Timing Summary" (noting the "Minimum input arrival time before clock", "Maximum output required time after clock", and "Maximum combinational path delay"). Note that not all of this information will be available for every design.

You should use the post-synthesis "View RTL Schematic" tool to see how the synthesis tool treats your designs. This is very helpful in the process of understanding synthesis and how VHDL coding affects implementation.

An optional (but recommended) exercise is to explore various adder types for the ALU. Assuming that the original adder you design is a carry-ripple, consider designs such as carry-lookahead and carry-save. This experiment will be especially interesting given the "fast-carry" logic built into our target FPGAs to help speed-up carry-ripple adders. Given this special circuitry, it will be interesting to see if these other adders still provide performance benefits.

Since you are starting with such simple designs, it may be tempting to work with a "guess and check" approach rather than carefully planning your designs. Synthesis and debugging are fast and easy for such small designs but can take a very long time as your designs get more complex. Start practicing quality design skills now.

### **Deliverables:**

A lab report with the following items is due electronically:

1. VHDL files (and schematics as necessary) for your register and ALU designs (for both the single components AND the bit-sliced components).
2. Post-place-and-route simulation samples for your register and ALU designs (for either the single components OR the bit-sliced components).
3. A table showing the relevant post-synthesis area and delay data for your register and ALU designs (for both the single components AND the bit-sliced components AND for both synthesis "Optimization Goal" settings, ∴ there will be four sets of results for the register and four sets for the ALU).
4. An analysis of the area and delay data. For example, how did bit-slicing and the "Optimization Goal" settings affect the results?
5. An explanation of one thing that you learned about design techniques, VHDL, or the CAD tools during the course of doing the lab.
6. (Optional) A table showing the relevant post-synthesis area and delay data for various adder designs, and an analysis of the results.
7. (Optional) A table showing the relevant post-synthesis area and delay data for a single-bit shifter and a barrel shifter, and an analysis of the results.