CPRE 381 Lab 6
The Arithmetic Logic Unit

In this lab you will construct a design for an arithmetic logic unit, the core of a computer processor. More details can be found in appendix B in your book.

Part 1 – 1-bit ALU

1. Design a 1-bit ALU to implement the following functions:
   a. Logical AND (operation 00)
   b. Logical OR (operation 01)
   c. Addition (operation 10)
   d. Subtraction (operation 10 with binvert)
   e. Set-on-Less-Than (operation 11)
2. The ALU should have the following inputs:
   a. A
   b. B
   c. B-invert
   d. Carry-in
   e. Less
   f. Operation (2-bits)
3. The ALU should have the following outputs:
   a. Result
   b. Set (Only the MSB will use this output but for modular design and reuse, you can build this into the basic 1-bit ALU and simply not use it for lesser bits in multi-bit ALU designs)
   c. p (propagate for CLA)
   d. g (generate for CLA)
   e. Carry-out (used when building ripple carry instead of CLA)
4. Implement your 1-bit design in verilog.
5. Simulate and verify your 1-bit design.

Note that this 1-bit ALU design can be used for constructing an ALU based on ripple-carry addition or carry-look-ahead. The same basic block can be used for either design by simply ignoring either the p and g outputs in ripple-carry design or ignoring the carry-out in CLA design. This will require a minor redesign of your 1-bit adder subtractor.

Also note, that overflow is not incorporated into the 1-bit design because it is only ever required by the MSB, and in contrast to the Set output, requires additional logic that would be wasted for all other bits (unless an optimizing compiler recognizes the unused logic and removes it).

Part 2 – 64-bit ALU

Now you will reuse your 1-bit ALU to construct four different implementations of a 64-bit ALU.

All designs will have the following inputs and outputs:

A[64:0] input
B[64:0] input
Op[2:0] input where the MSB is binvert, and the two LSBs are the operation code.
Result[64:0] output
Zero output flag denoting that the result is equal to zero
Overflow output flag denoting that an overflow has occurred

Ripple-carry

Construct a ripple-carry 64-bit ALU where the Carry-out bits are chained to the carry-in bits of the next higher bits.

One-level CLA

Construct a carry-look-ahead 64-bit ALU with one level of CLA for every 4-bits, that are then chained in ripple-carry fashion.

Two-level CLA

Construct a carry-look-ahead 64-bit ALU with two levels of CLA, one for every 4-bits, and one to build 16-bit blocks from 4-bit blocks. Chain four 16-bit blocks in ripple-carry fashion to construct a 64-bit ALU.

Three-level CLA

Construct a carry-look-ahead 64-bit ALU with three levels of CLA, one for every 4-bits, one to build 16-bit blocks from 4-bit blocks, and one to build a 64-bit ALU from 16-bit blocks.

1. Implement all four ALU designs in verilog.
2. Simulate and verify the functionality of all four designs (for all operations).
3. What two values when supplied to A and B, and what operation yields the longest critical path?
4. Supply these values and determine the propagation delays of all four ALUs. (Select an Altera Cyclone device for simulation).
5. View the log files and record the resources used by each of the ALU designs.
6. Based on the results of timing and resource usage, which ALU do you recommend for use in FPGA synthesis? Is this what you expected?