# Part 2: Combinational Logic Modules

# Due: March 13th

## Objective:

The project will have combinational or task-oriented operations that have to be created. These can be created first.

## Data Handling:

What data is the cohort going to handle with this project? Dr. Becker wants a minimum of 16-bit integer inputs and a 32-bit integer output. Such a data level is miniscule compared to the different encodings and arithmetic circuits available. This data bit-resolution has to be chosen before the Combinational Logic Modules are designed.

## Combinational Logic Modules

Any circuit that is being used for calculations or manipulations or controls must have a library of useful tools and building blocks. Plan for the complete system, and then decide on building the components. Such code is available from the various textbooks in the course, as well as the web. These can be used as examples to build from.

Draw up a parts list of the modules the cohort will need to complete the project. Then, code the modules in Verilog and make a testbench to test each module to make sure it works.

### Example:

An ALU would have to have the various components created and the different non-sequence operations created.

* Logic:
  + All the logic gates: AND, OR, NOT, XOR, and their compliments exist as routines in a hardware design language. Funneling the parameters from the bus to the right locations is the hard part.
* Mathematics:
  + A Half-Adder can add two one-bit numbers. A Full-Adder can add two one-bit numbers and a one-bit carry. An array of adders can handle much larger combinations of binary integers. And if binary integers can be handled, then fixed-point binary numbers can be handled.
  + If two binary numbers can be added, 2’s compliment can be used to do subtraction and include an overflow bit for error detection.
  + Binary values of integers can be re-routed to an offset position before an addition. The result of an AND followed by an offset followed by an ADD is multiplication.
* Controlling Components
  + A **multiplexor** is a component that allows different results to be fed to different channels from a one-hot selector line. Each of the inputs is a channel, and they are all equally of n-bits long. For example, a system with a 32-bit result would have 32-bit input channels.
  + Multiplexors operate on one-hot signals. A **decoder** changes data from a n-bit binary integer to a 2n-bit one-hot binary number. An **encoder** changes data from a 2n-bit one-hot binary number into an n-bit binary integer. Not only does this help with the multiplexor, but it allows *mutual exclusion* on control lines. Very useful if you do not want a piece of equipment to go forward and backward at the same time.
  + If a cohort is doing something like a robot, they may also need to look into an **Arbiter**, which is either “left” or “right”. A Right Arbiter turns any input into a one-hot, least significant bit. A Left Arbiter turns any input into a one-hot, most significant bit.

## Suggested Parts List:

|  |  |
| --- | --- |
| Decoder, 4-to-16 bit | Encoder, 16-to-4 bit |
| Half-Adder | Full Adder |
| Subtractor, 16-bit | Multiplier |
| Multiplexor | NOT Gate |
| AND Gate | NAND Gate |
| OR Gate | NOR Gate |
| XOR Gate | XNOR Gate |
| Left Arbiter | Right Arbiter |

# WHAT TO TURN IN FOR PART 2:

1. A PDF named <CHORTNAME>.PART2.pdf containing the following
   1. A description of the data being handled, including data type and bit-size.
   2. A list of the parts created in the combinational logic portion. Each entry in the list needs:
      1. The name of the component
      2. The inputs and their bit-size
      3. The outputs and their bit-size
      4. The interfaces and their bit-size
      5. Any controls and their bit-size. (Such as Multiplexor Select)
      6. A one-sentence description if necessary.
2. The code
   1. The code for the components listed in the PDF should be turned in. This should be a zip file named <COHORTNAME>.PART2.zip
   2. Each component should be written into a hardware design language program.
   3. Each component should be used ***at least once*** from the testbench.
3. The output
   1. The output should be saved and turned in as a text file called <COHHORTNAME>.PART2.txt
   2. The testbench should display the results of each component. At this stage, the testbench is just calling individual components, not a full system with a system clock.