Design Implementation

I started by making a module called propGen that takes in two 1-bit inputs and outputs the propagate and generate functions for those inputs. I then made a fullAdder module to calculate the sum of two 1-bit inputs and a carry-in bit using a three input XOR gate. The next module called propGen8 takes in 8 bit inputs and calculates the propagate and generate function at each bit level using the propGen module. The propGen8 module also calculates the group propagation and group generation of the 8-bit inputs by using the already calculate propagation and generation results at each bit level and plugging them into formulas given on slide 26 of lecture 6. The next module CLA8 calculates the non-RCA sum of two 8-bit inputs. The module also takes the propagation and generation values (calculated at each bit level through the propGen8 module) and a carry-in bit. The propagation and generation values along with the initial carry-in are used in the formula provided at the bottom of slide 22 of Lecture 6 to derive the carry-in at each bit level of the 8-bit values under consideration. A for loop inside the generate statement then calculates the sum at each bit level using the fullAdder module which takes in the two input bits along with the already calculate carry-in bit at that particular level.

The next module, named 'LCU', Lookahead Carry Unit. This unit takes in the group propagation and group generation from 4 blocks of propGen8 where each block calculates the propagation and generation for eight bits of the 32 bit inputs. The LCU also takes the original carry-in at the first bit level. The LCU then calculates the carry-in at the eighth, sixteenth, and twenty fourth bit by using the formulas provided on slide 26 of Lecture 6. Finally the 'CLA' module acts as a carry-lookahead adder. The CLA uses propGen8 to calculate the required group propagate and group generate vales. These vales are inputted into a LCU to get the carry-in at the above mentioned bit locations. Then 8 bit blocks of input along with the corresponding carry-in values are used to get the sum for the corresponding 8 bits giving us a 32 bit final sum.

The next modules constructed were the 1-2 decoder (one-hot output) and the 2:1 multiplexer (one hot input). These modules were built using the method employed in the previous homework. The input and output bit widths for the decoder were made changeable using parameterization. A 3:1 multiplexer was also created and two such multiplexers were combined to get a 6:1 multiplexer. The module 'lshifter' was created next. The module had a parameter value which could be change to reflect the shift amount. The module took in a 32 bit input and used the parameter value to concatenate the input appropriately and assign each bit values that represented a left logical shift. Similar parameterization and concatenation was used in the 'rshifter' module to create a right arithmetic shift. The 'sll' module was written next. The module takes in 32 bit input and a 5 bit input which represents a shift amount. Each bit of the shift amount input is decoded using a 1-2 decoder to get a one-hot select signal for that bit level. The lshifter module and the 2-1 module is then used to see whether the input should be shifted left by 16 bit and/or 8 bits and/or 4 bits and/or 2 bits and/or 1 bit using the barrel shifter design given in lecture 7. A similar approach is used to get a barrel shifted right arithmetic output in 'sra' module. The key difference between sra and sll modules is that sra uses rshifters while sll use lshifter.

The next module, called 'bitwise', takes in two 32-bit inputs and outputs the bitwise AND, OR, and NOT values. The final ALU module combines the already constructed modules. The ctrl_ALUopcode is decoded using a 3-6 one hot decoder. The decoder takes in the lower three bits of the opcode since the upper two bits are always 0 for the required functions. The bitwise module calculates the AND, OR, and NOT of the two 32 bit inputs. The CLA module computes the result of addition of the two inputs (with carry-in of 0) and the result of subtraction (using NOT of the second input along with a carry-in of 1). The sll module computes the left logical shift and the sra module computes the right arithmetic shift using the provided shift amount. A 6-1 multiplexer selects outputs the data_result using the decoded 6 hot select signal. Finally AND and OR operators are used bitwise to calculate isNotEqual and isLessThan output.