Cyclone IV EP4CE115F29 device

Ludovic Provost 300208450

Dexter Day 300192263

Group #16

Laboratory 2: Fixed-Point Arithmetic

CEG3155: Digital Systems II

Prof. Rami Abielmona

Due October 18th 2023

Objectives:

The objective of this laboratory is to design and build a fixed-point signed multiplier and divider in VHDL. Upon completion, the student must be able to:

- Design, realize and test a fixed-point signed multiplier unit;
- Design, realize and test a fixed-point signed divider unit;
- Demonstrate a complete understanding for fixed-point arithmetic.

Problem and Solution Discussion:

The design problem of this lab was to implement a signed fixed-point arithmetic ALU capable of fixed point division and multiplication along with addition and subtraction. In order to complete this top level ALU each individual component first had to be completed. This started with the pre-lab: an addition/subtraction component using a carry-look-ahead system where instead of waiting for each adder to calculate a carry value before the next could do the same, each is calculated simultaneously and returned to the full adders. The implementation in VHDL is shown in the verification section.

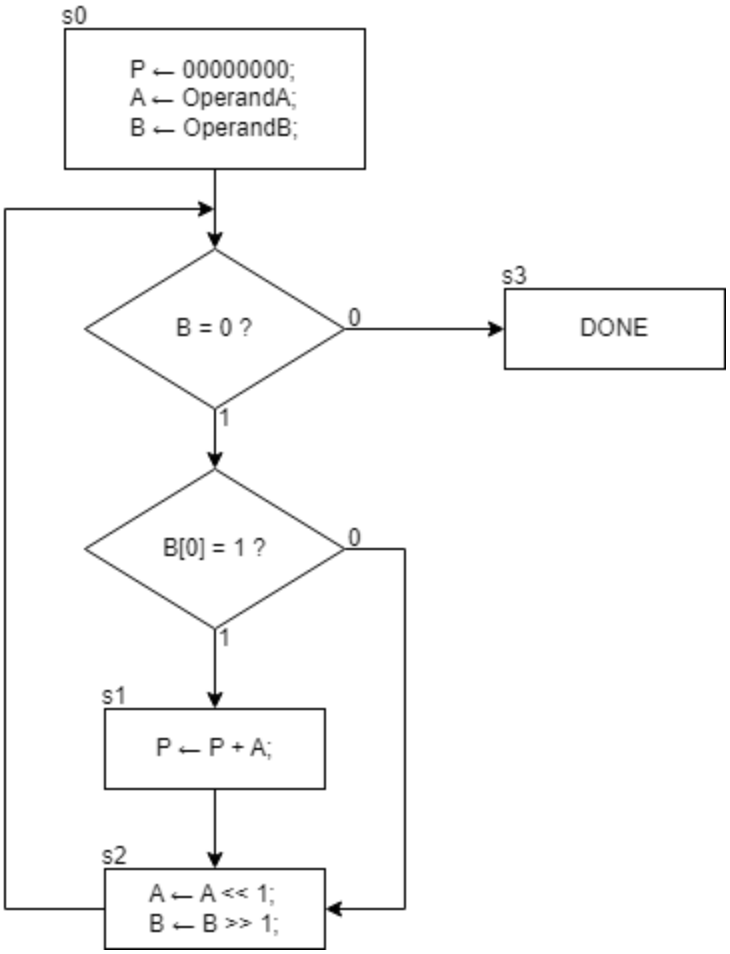
Next the design of the multiplier circuit, which required the ASM design method to be completed. In order to begin designing, the pseudocode for the multiplier first had to be understood. The pseudocode for the design was given in the lab manual as:

$P = 0$;
for $i = 0$ to $n - 1$ do
    if $b_i = 1$ then
        $P = P + A$;
    end if;
    Left-shift A ;
end for;

(b) Pseudo-code

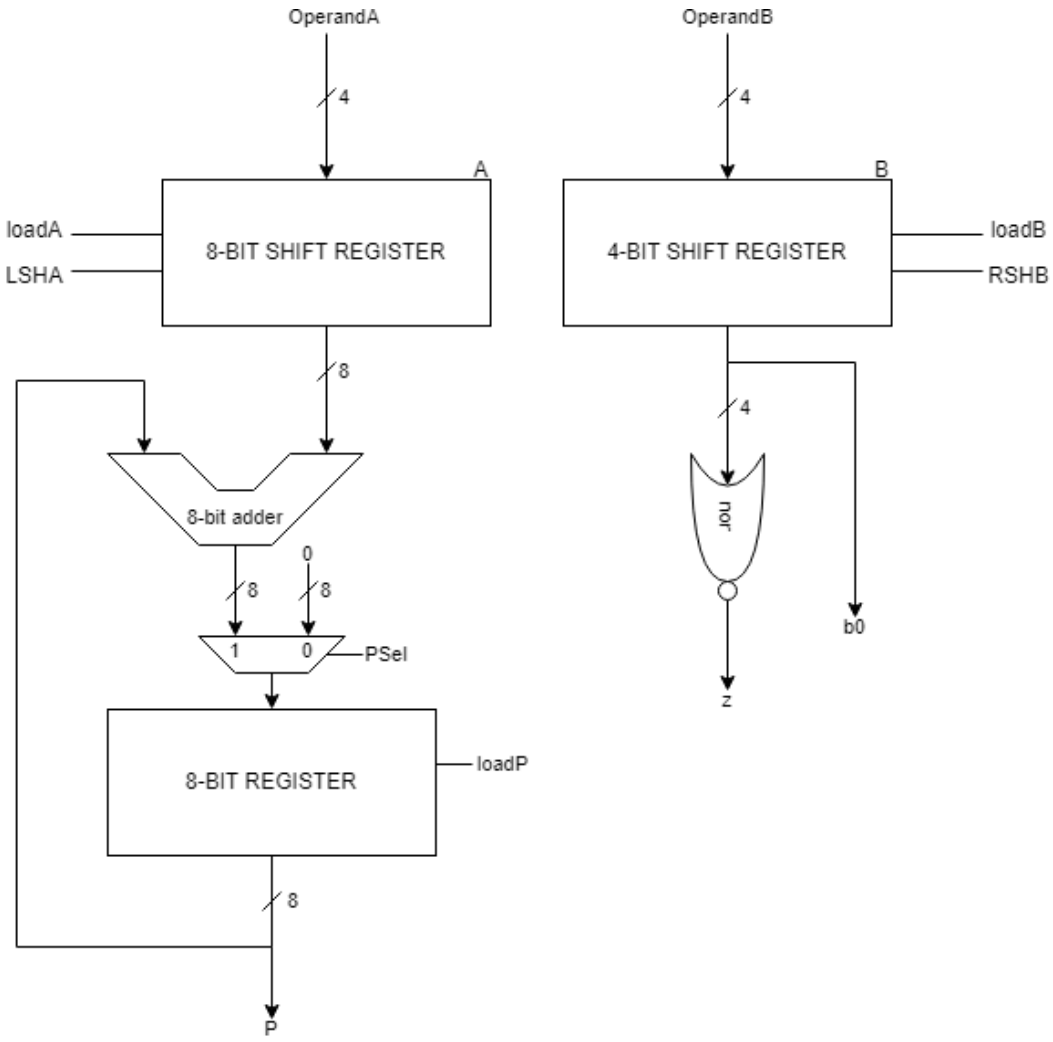
Multiplier Pseudocode

Using this the ASM chart could be developed in order to better understand the design path we were going to take with the multiplier. Below is the ASM chart for the multiplier.



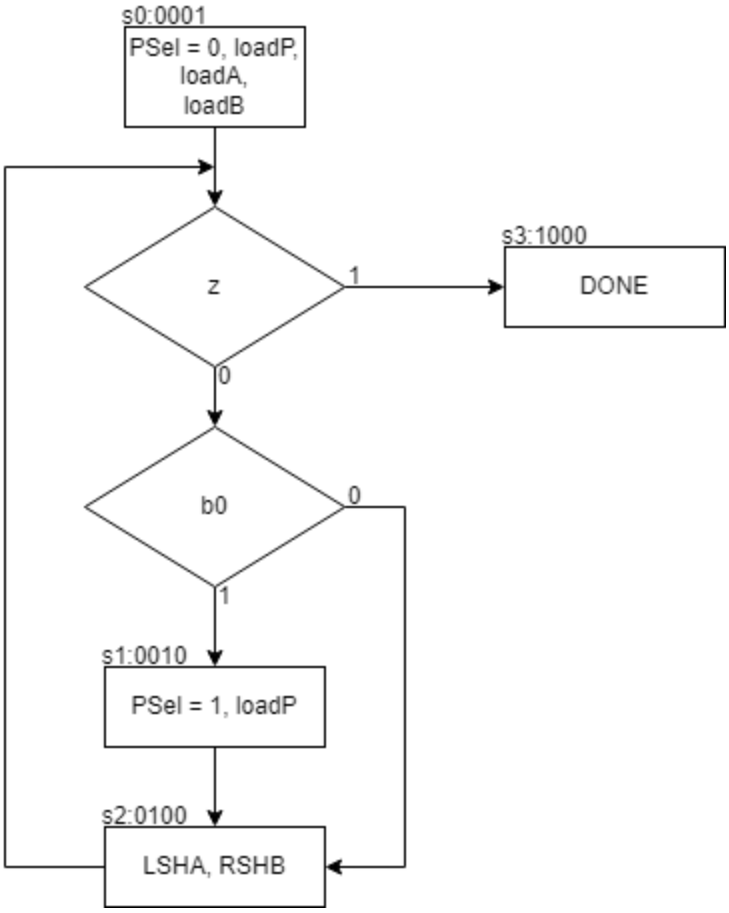
Multiplier ASM chart

Once this was drawn out the components required to complete the multiplier could be further understood, these included shift registers, a 4 bit adder, and storage registers. Using this information the ASM datapath of the multiplier could be imagined.



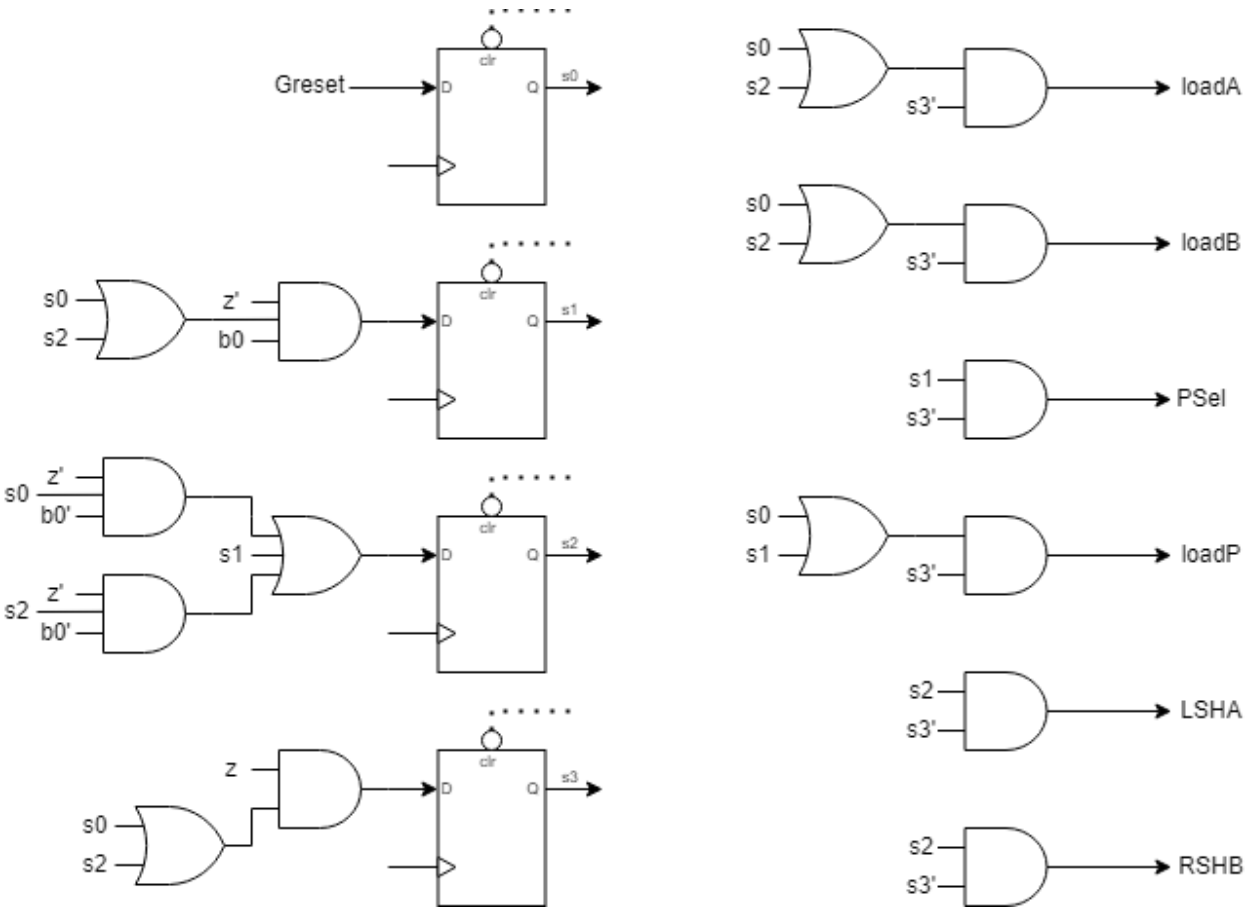
ASM datapath of multiplier

The datapath showing all blocks or components needed to complete the unsigned multiplier is shown above, with all control logic inputs and outputs layed out for the completion of the detailed ASM chart and design of control logic.



ASM detailed chart for multiplier

The more detailed layout of states and control inputs and outputs helps us come to the next step of designing control logic. For the control logic of the multiplier only four states were needed.



ASM Control Logic multiplier

Inputs from the datapath included the zerobit and the LSB of the B operand. These combined with state outputs of each flip flop give us the control logic outputs required to implement the multiplier fully. Most of these outputs are for the registers and the mux taking results from the adder. When implementing the control logic however the clock had to be inverted in order to add a delay to the signals and allow the circuit to function properly.

Final signed multiplier design

Using this the multiplier could be designed, the VHDL code for this is shown in the appendix. But it is still not implemented correctly as a signed fixed point multiplication circuit. In order to do this 2s complement components where required to complement both 4 bit inputs and the 8 bit output depending on the signs of each of the inputs and outputs. If an input was negative i.e. the MSB was 0 then a selection mux would take the complemented version of the input into the multipliers respective register. The output register should only be negated if one of the MSB bits of the input and only one is zero. Meaning that a xor gate can control the output negation.

This is as shown in the VHDL code in the appendix where the selection bit of the mux used for the output is a xor of the MSB of both operand A and operand B.

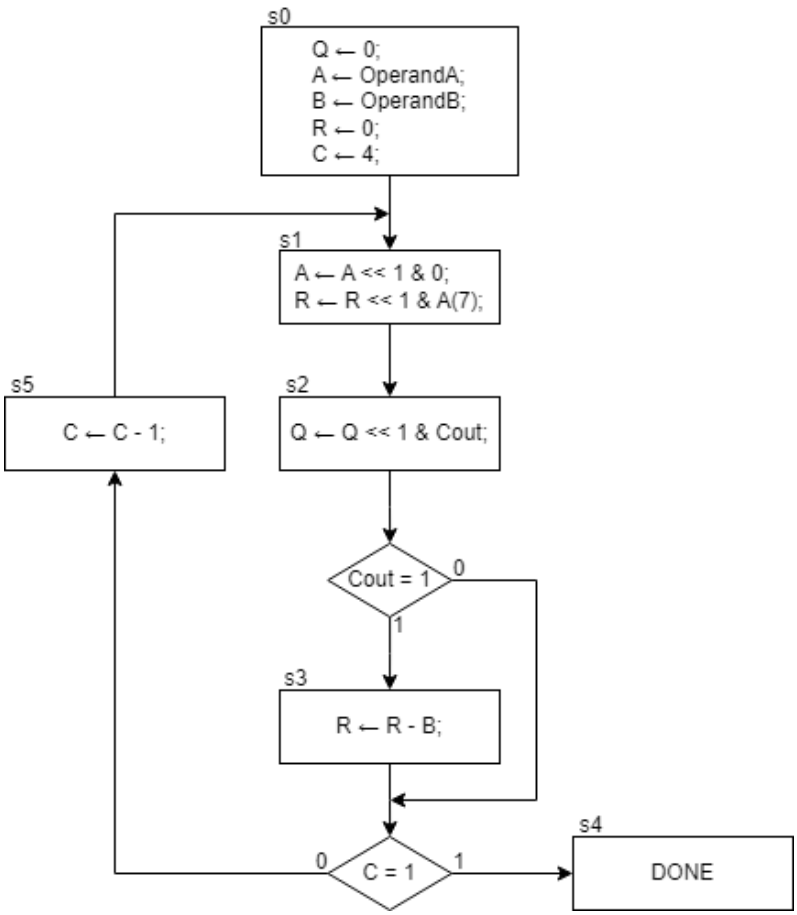
When designing the Divider circuit the exact same methods were followed in order to arrive at a functioning signed divider entity. Starting with understanding the Pseudocode shown below.

```
R = 0;
for i = 0 to n − 1 do
    Left-shift R||A ;
    if R ≥ B then
        q_i = 1;
        R = R − B;
    else
        q_i = 0;
    end if;
end for;
```

(c) Pseudo-code

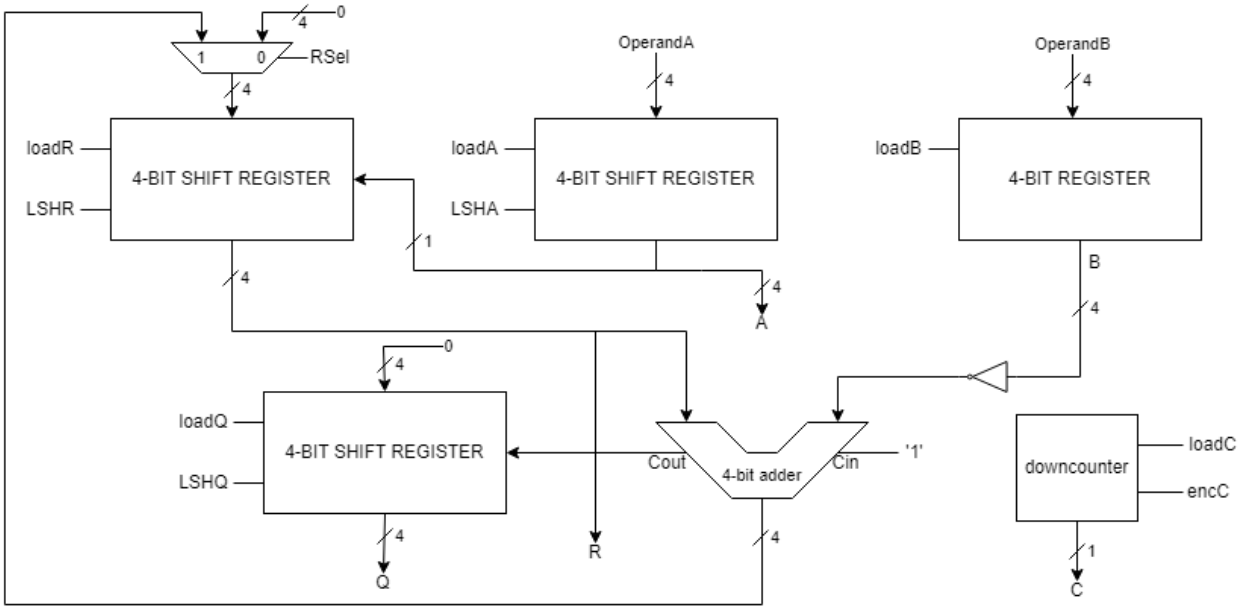
Pseudo code for unsigned divider circuit

Using this pseudocode the ASM chart could be drawn out. This chart is shown below.



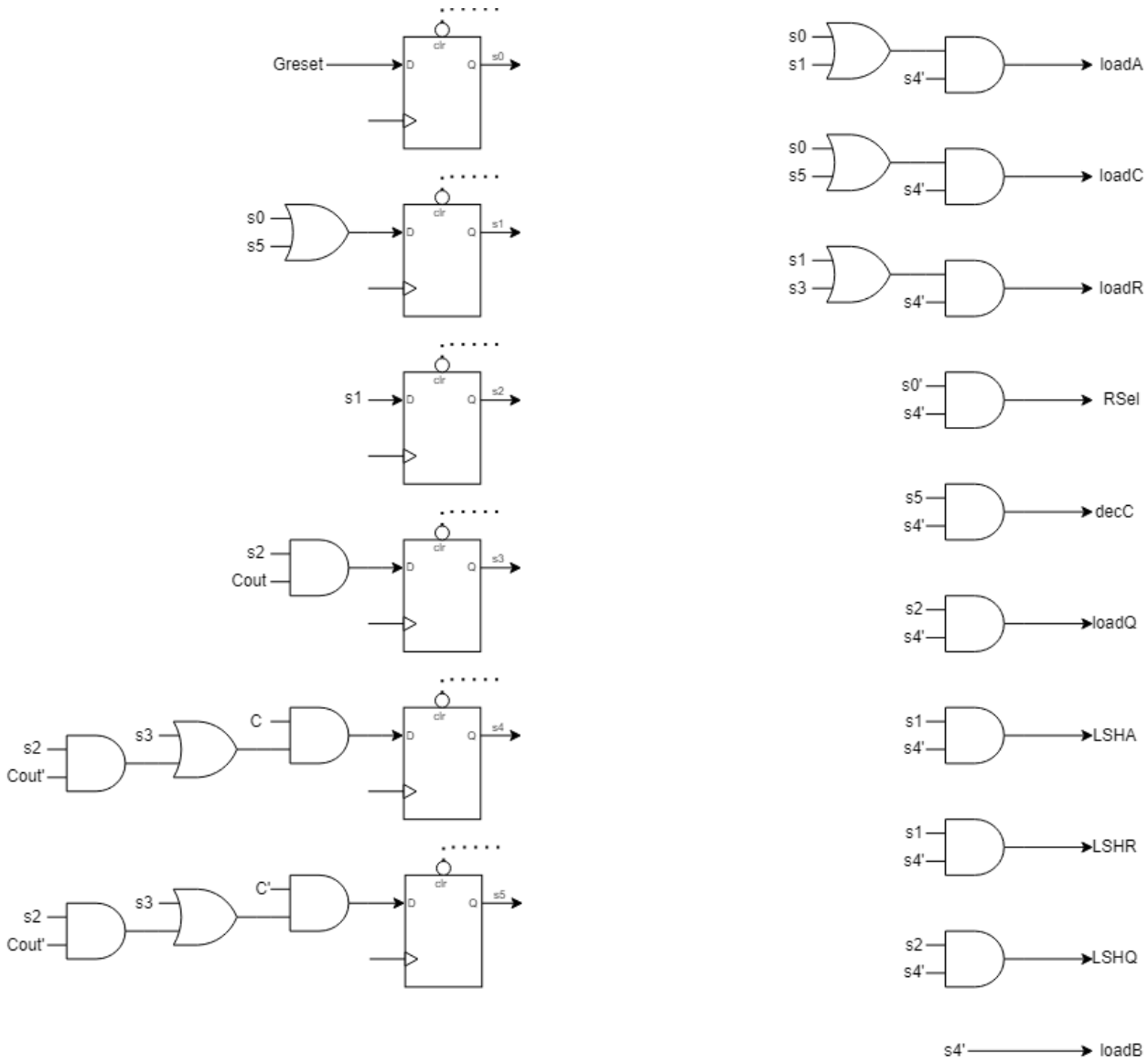
ASM chart divider

Once the operations that had to be performed on the input operands were understood a datapath could be created with all required elements and control signals.



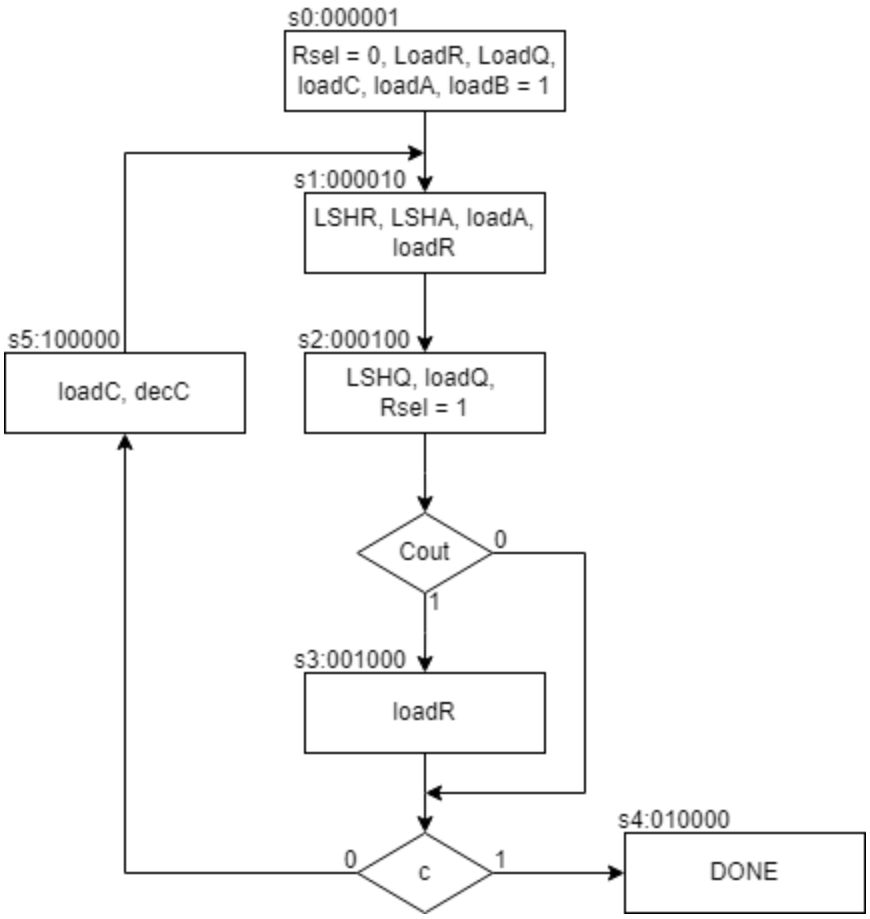
ASM datapath divider

The datapath of the divider is far more complicated than the multiplier as many more actions must take place in order to achieve the correct quotient and remainder. The only two output control signals given by the divider are the Cout signal and the counter completion signal C used in the control logic. Next the control path can be derived from the ASM chart and datapath.



ASM control logic divider

The control logic contains six states and rotates through them depending on the status of the count variable. Once control logic inputs and outputs are understood the detailed asm chart is created in order to show the exact control signals that will be used at each state in the ASM chart.

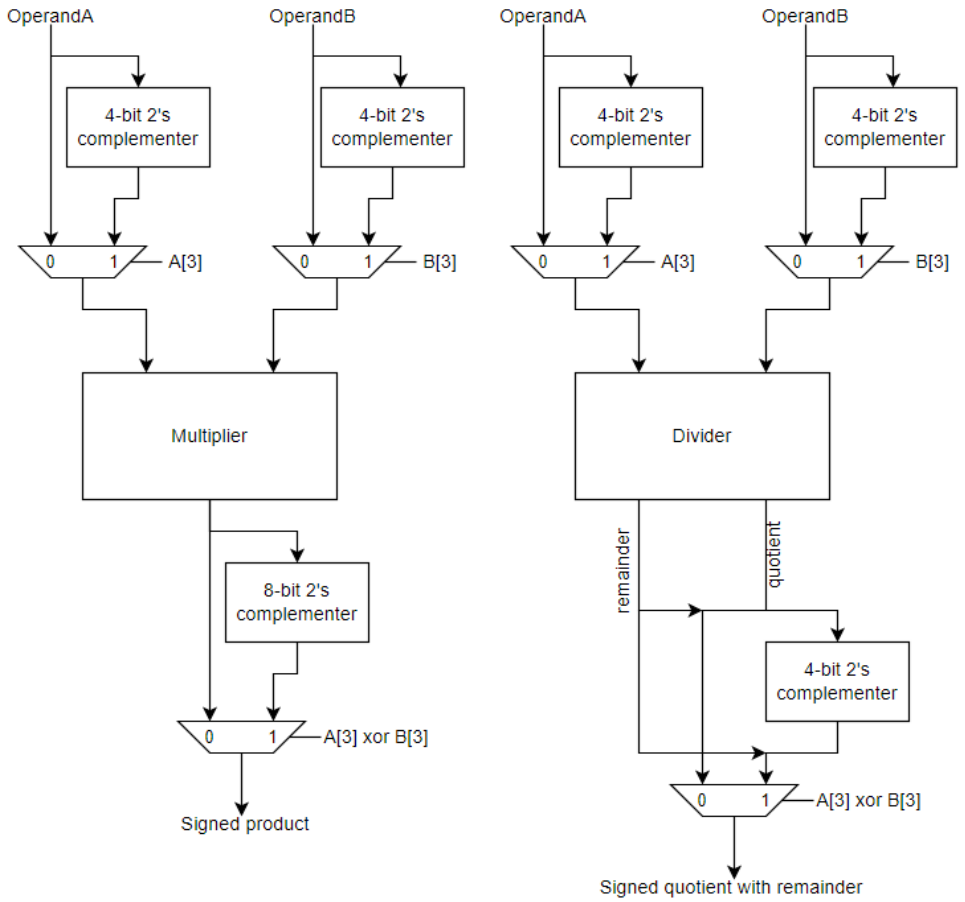


Detailed ASM chart divider

The final detailed ASM chart displayed above shows the control logic signals and the flow of each state of the circuit. The final structural VHDL design of the divider datapath and control logic could then be completed and implemented as shown in the appendix.

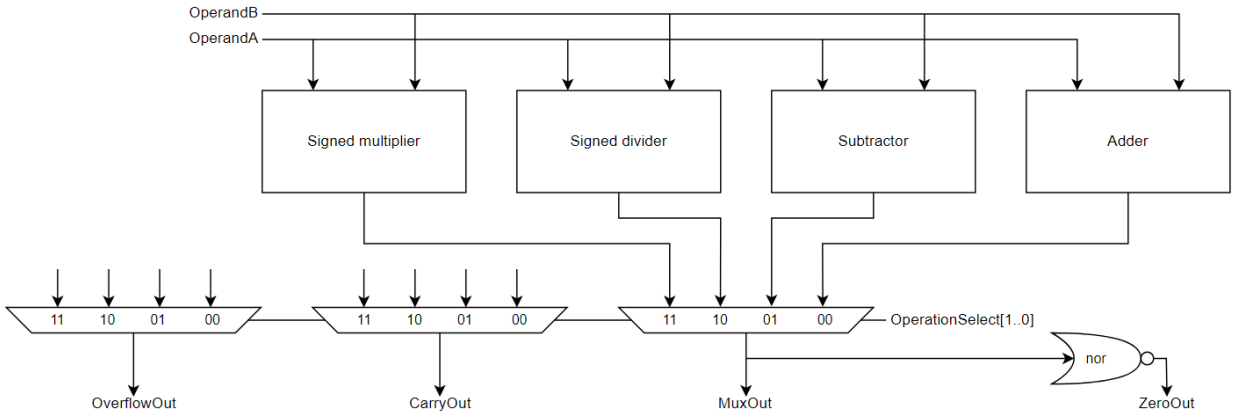
Final signed divider design

Exactly like the multiplier, in order for it to properly function with signed numbers it needed both input and output complementers for the two 4 bit inputs and 8 bit output. Once again each complementer was put in a 2x1 mux with the original input with the selection bit being the MSB of the input operand, only complementing on a negative input. The output complementer only complementing on a xor of the two MSB of inputs. Only complementing when only one of the two numbers was negative. This successfully created a signed divider circuit.



Signed multiplier and signed divider circuit

The final top level design of the fixed point arithmetic ALU used several mux components along with all three of the completed designs. Using the adder/subtractor, signed multiplier and divider with selection bits being operation select inputs. Two other 4x1 mux were used for the overflow selection from each circuit and the carryout. The simple structure of the connected top level design can be seen in the appendix in VHDL.

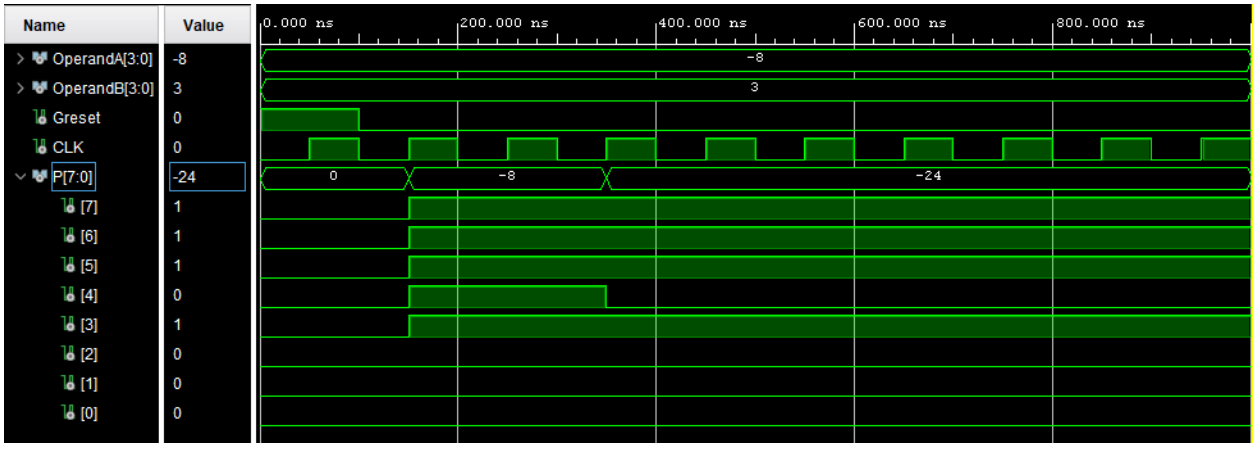


Top level circuit

Verification:

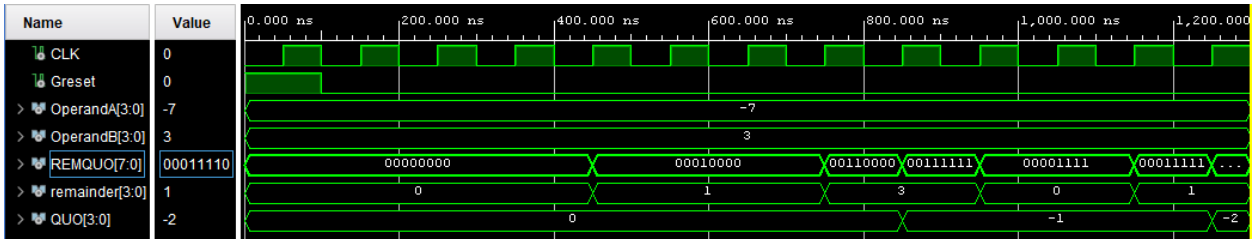
Functional Simulations of major components:

Signed Multiplier:



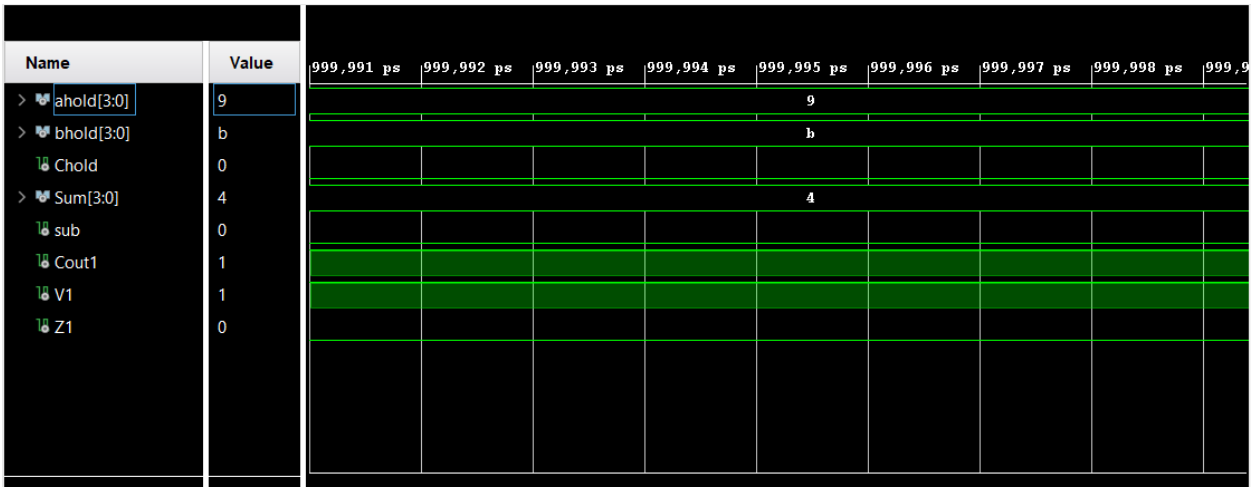
As expected, -8 × 3 = -24. The product is the signal P, which sits at a value of $(11101000)_2 = (-24)_{10}$ when it is done with the multiplication.

Signed Divider:



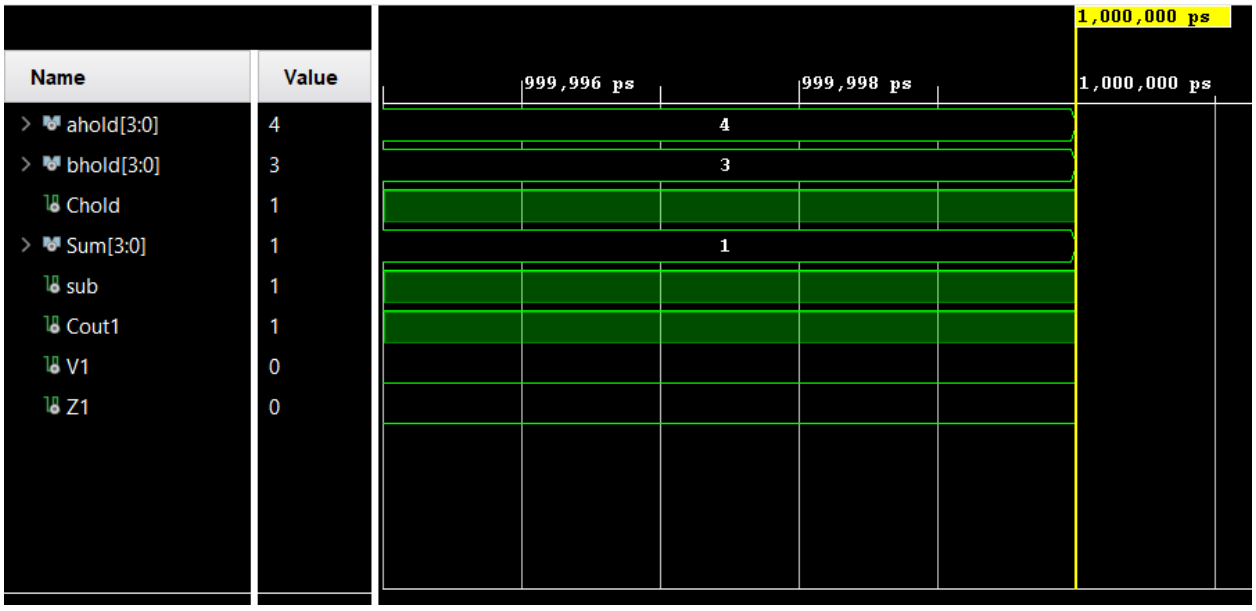
As expected, -7÷3 gives -2 with a remainder of 1. The signal REMQUO is the concatenated remainder and quotient. The 4 MSB of REMQUO are $(0001)_2 = (1)_{10}$ and its 4 LSB are $(1110)_2 = (-2)_{10}$.

Adder:



In this test the inputs for A where 1001 and B 1011, the sign bit produced an overflow with the resulting output being 4 as expected from 1 + 3.

Subtractor:



With the sub bit set to 1 and Cin(Chold) value set to 1 the subtractor is primed with 0100 loaded in A and 0011 loaded in B. The result is 0001 as expected. No overflow is detected.

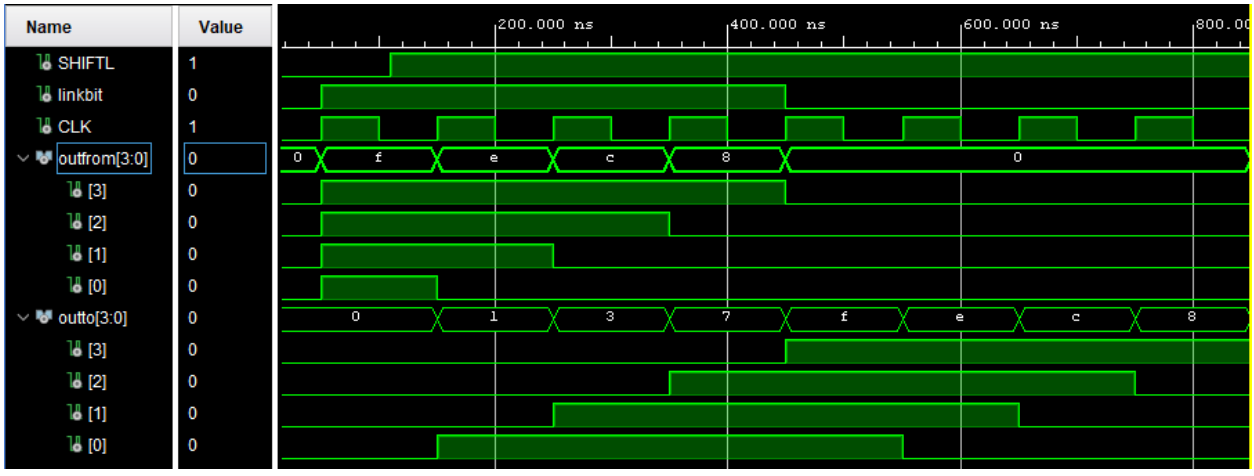
Carry-Look-Ahead System for 4 bit-Adder:

```
entity carrylookahead is
    port(
        P, G: in std_logic_vector(3 downto 0);
        Cin: in std_logic;
        PP, GG : out std_logic;
        Carry : out std_logic_vector(4 downto 0)
    );
    end carrylookahead;

    ARCHITECTURE structural OF carrylookahead IS
      -- put declarations here.
    BEGIN
      -- put concurrent statements here.
        Carry(0) <= Cin;
        Carry(1) <= G(0) or (P(0) and Cin);
        Carry(2) <= G(1) or (P(1) and G(0)) or (P(1) and P(0) and Cin);
        Carry(3) <= G(2) or (P(2) and G(1)) or (P(2) and P(1) and G(0)) or (P(2) and P(1) and P(0) and Cin);
        Carry(4) <= G(3) or (P(3) and G(2)) or (P(3) and P(2) and G(1)) or (P(3) and P(2) and P(1) and G(0)) or

        PP <= P(3) and P(2) and P(1) and P(0);
        GG <= G(3) or (P(3) and G(2)) or (P(3) and P(2) and G(1)) or (P(3) and P(2) and P(1) and G(0));
    END ARCHITECTURE structural; -- Of entity 4bitcarrylookahead
```

For the 4 bit adder/subtractor the carry-look-ahead module was created to take P and G values from each Full adder and calculate the total 5 carry bits involved in each operation. Each of these carry bits were then attached to the full adder's carry_in ports with the final or MSB going to the carryout. This was used for both addition and subtraction operations.

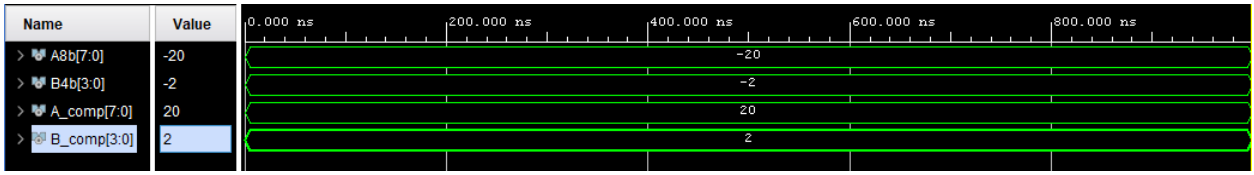
Testbenches Utilized in the Verification Process:

Testbench for Register Pair Shift:

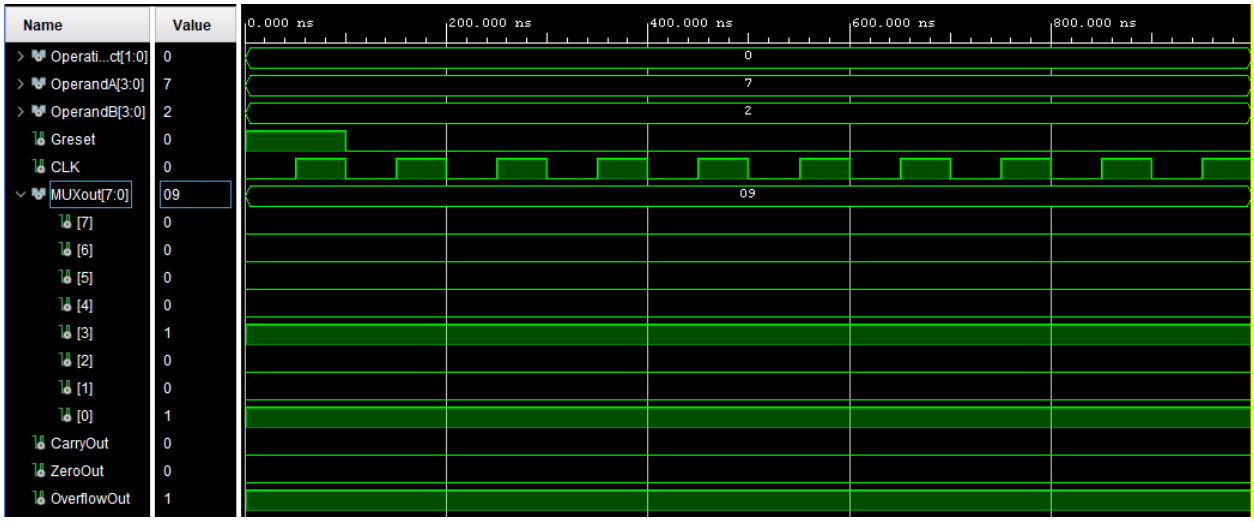


This testbench was to test whether the bit transfer between 2 shifting registers worked or not. This simulates shifting the MSB of the operand A's register to the LSB of the remainder's register in the divider. As we can see starting at time 150 ns, when they both shift, the bit is moved on the same clock edge, as wanted.

Testbench for 2’s complementer:



Top-Level Simulation with Addition Selected:



Since the operation selectors are “00”, MuxOut provides the sum of both operands. 7+2 gives 9, as expected, but since the MSB is for the sign, there is overflow.

Lab Discussion:

The first problem we came across had to do with the control logic. It always took 2 clock edges for the signals from the control logic to make their way to the datapath and then for the components inside the datapath to kick in. Reversing the clock for the control logic only fixed this problem.

When designing the divider, we realized a counter would need to be implemented somehow so that it could count down n-bits. Since efficiency and cost are not our priorities, we decided to reuse components we had already completed. The down counter is simply a left shift register that is initialized at “0001”. When the counter’s decrement signal “decC” is on, it shifts the register’s content to the left so that after 3 left shifts, the MSB in the register is ‘1’ and the counter finally gives the signal to stop.

The remainder and quotient registers in the divider need to be able to take a 1 bit input when shifting, and the A operand register needs to output its MSB when shifting too. We needed registers with new functionalities, so we designed the “fromALShiftReg_4b” and “ALShiftReg_4b”. The first was for the A operand register to be able to output only a single bit, while the latter was used to make the remainder (R) and quotient (Q) registers. This new functionality was tested using a testbench and the result can be seen in the Verification section of this report.

To make our multiplier and divider work with signed numbers, we decided to go with a simple design, which utilizes muxes to select whether the multiplier and register will receive an operand directly or an operand's 2's complement. The selector for these muxes are the MSB of their respective operand. The output of the multiplier or divider also have 2's complementers and a mux, but the selector for this mux is a xor between the signed bits of both operands. Therefore, the complement of the product or quotient will only be selected if the operands have opposite signs. This logic is a little bit different with the divider since we only want to complement the quotient and not the 8-bit concatenated remainder and quotient (see signed divider circuit above), but the idea is the same.

Conclusion and Appendices:

After successful design and verification the desired circuit was produced, able to complete signed multiplication, division and addition/subtraction. Any struggles related to design and programming were solved leading us to a successful laboratory 2. All objectives of lab 2 were completed correctly. This being the ASM design process, VHDL coding, and verification + testing. A full understanding of fixed-point arithmetic was successfully gained after the lab was fully completed. Overall lab 2 was successful and the design was implemented correctly. All final VHDL source files are commented and available here along with the simulation waveforms:
Drive folder of VHDL files for lab 2