Lab 1

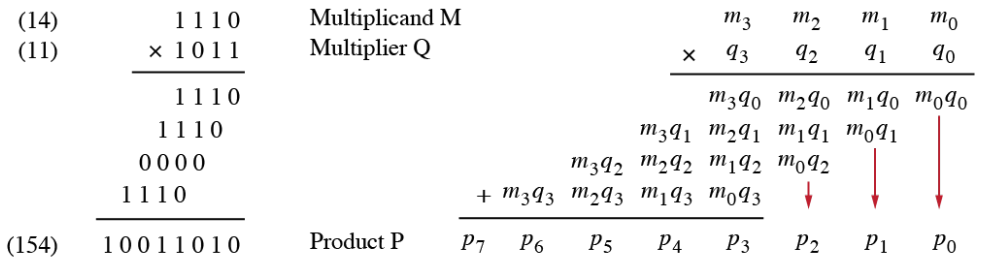
Building a Multiplier Circuit and a Simple Calculator

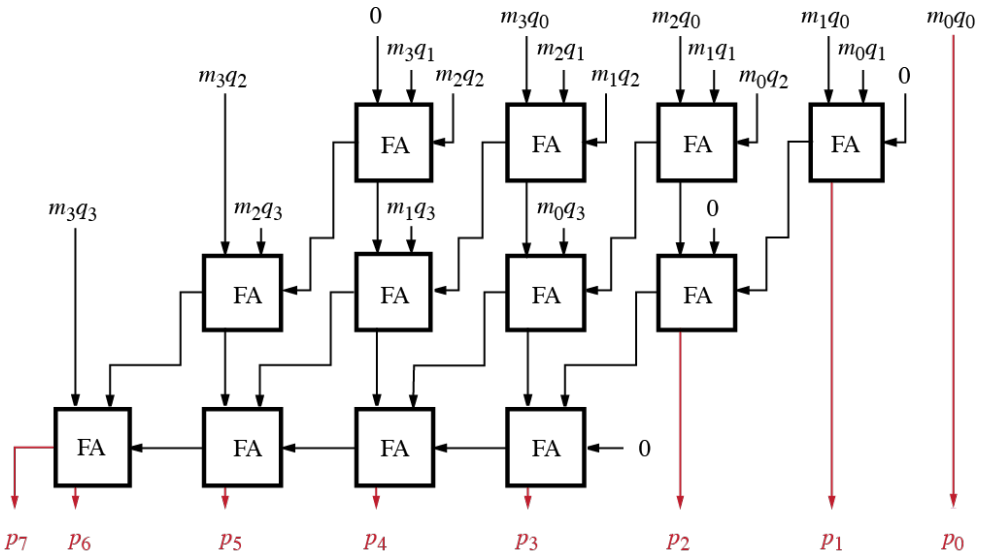
The purpose of this lab is to build a binary multiplier circuit, verify its functionality using a testbench, then combine it with an adder/subtractor circuit to create a basic calculator.

Part 1 (Building a multiplier circuit)

Two binary numbers can be multiplied in the same method used to multiply two decimal numbers. The figure below shows an example of multiplying 14x11=154. In this part, we focus on building an unsigned number multiplier, because signed number multiplication is more complex.

A binary multiplier can be implemented using several stages of full-adder (FA) blocks connected in a ripple-carry configuration. However, the delay caused by the carry signal rippling through the different FA blocks and the different stages can cause a significant impact on the time needed to generate a correct product result. The *carry-save array* configuration, is a binary multiplier that reduces the delay effect and produces correct results faster. The diagram below shows a 4-bit multiplier circuit arranged in a *carry-save array* configuration.





In this part, you will build the *carry-save array* multiplier circuit shown in the figure above, then you will write a test bench to verify the functionality of your implmentation. In all your steps in this part, assume the numbers used are 4-bit unsigned integers.

The following steps should guide you through the implementation:

1. Verify the functionality of the circuit by tracing the multiplications 14x11 and 9x5 direclty on the digram. In other words, (with a pencil and paper) trace the patterns of 1's and 0's propagating through the different circuit elements, and verify they produce the correct results
2. To reduce clutter and make your code easier to read, debug, and modify, you will use hierarchical design.
3. $m_i q_j$ can be simply implemented using a 2-input *and* gate. Write a verilog module (call it `mq_4bit`) that takes a 4-bit signal `m`, 1-bit signal `q`, and generates a 4-bit signal `mq`. Specifically, the module's output should be anding the different bits of `m` with the signle bit of `q`. You might find the concatenation operator useful in this step
4. Import all files included with this guide. Open `full_adder.v` and make sure you understand how to use it.
5. Write a verilog module (call it `csa_multiplier`) to describe the circuit shown in the figure above. You should utilize the `full_adder` and `mq_4bit` modules and instantiate as many of them as necessary
6. Verify the functionality of `csa_multiplier` by writing a testbench (`csa_multiplier_tb`). You should use the following test vectors (0x10, 5x5, 9x5, 12x13, 15x10) to verify the multiplier is working correctly. You can add more test vectors if you want.
7. Include a screenshot of the simulation output with your submission, you might want to figure out how to embed it in your README.md file

Part 2 (Building a simple calculator)

In this part, you will combine the `csa_multiplier` and `adder_subtractor` (provided with this guide) to build a simple calculator. The calculator should perform 4-bit addition, subtraction, and multiplication. You will also verify the functionality of your calculator by implementing it on the FPGA board.

1. Write Verilog code to describe an 8-bit 2x1 multiplexer. This MUX will be used to control what is displayed at the output of the calculator
2. Write a Verilog module (`simple_calc`)
    a. The module should accept two 4-bit inputs `X, Y`
    b. The module should accept a 2-bit operator select input (`op_sel`).
        i. `op_sel` = 00 (add)
        ii. `op_sel` = 01 (subtract)
        iii. `op_sel` = 1x (multiply)
    c. The module should output an 8-bit signal (`result`)

d. Draw a block diagram of the calculator. Show the different circuit elements and how they are connect. Specifically, your diagram should show a multiplier, adder/subtractor, and 8-bit 2x1 MUX circuit:
    i. Instantiate an instance of a 4-bit adder/subtractor and connect `X, Y` and the appropriate `add_n` signal to it
    ii. Instantiate an instance of the 4-bit multiplier circuit you built in part 1. Connect `X, Y` to it
    iii. Instantiate an 8-bit 2x1 MUX you wrote in the step above. Connect the output of the multiplier to one input and the output of the adder/subtractor to the other input. You might also want to connect `4'b0` to the most significant bits of the input connected to the adder/subtractor.
e. Verify the functionality of your simple calculator by implementing in on the FPGA board using the following IO specifications:
    i. SW3 ← SW0 for the input `X`
    ii. SW7 ← SW4 for the input `Y`
    iii. SW15 ← SW14 for the `op_sel`
    iv. LED7 ← LED0 for the output `result`
    v. LED14 to display the `carry_out` of the adder/subtractor
    vi. LED15 to display the `overflow` of the adder/subtractor

Submission check list:

[ ] All Verilog code you generated or modified
[ ] All testbenches written
[ ] Include a screenshot of your testbench output, preferably embedded in your README.md
[ ] Include a sketch of the block diagram used in Part 2, preferably embedded in your README.md
[ ] Short video demonstrating a working calculator