Cairo University
Faculty Of Engineering
Computer Engineering Department

CMP3020
Fall 2023

Project: Part 2

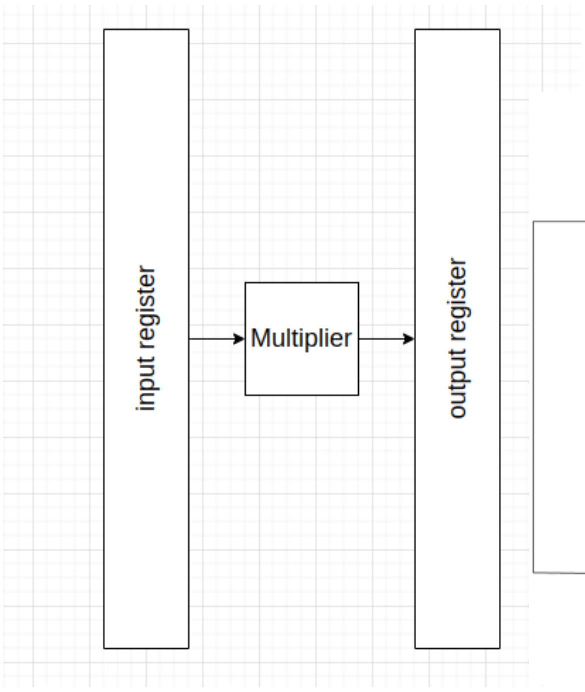
Multipliers Mania

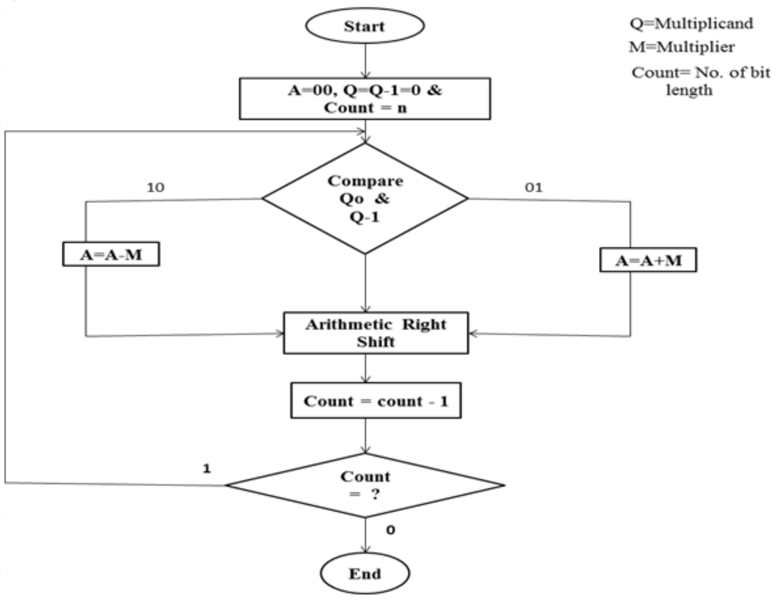
Overview:

Another important block in the computing unit is multipliers. Their performance impacts the whole chip. This is why in this mini project, we will explore different implementations of multipliers and study their characteristics.

Requirement:

1. Design and implement using verilog the following **32-bits** **signed** integer multipliers
   - All multipliers will have an input register before the multiplier and an output register after the multiplier. (left fig)
   - Use the most appropriate adder with the multiplier from the previous mini-project, justifying your choice.
     i. Verilog ('*') version of multiplier //we already took in lab
     ii. Multiplier Tree (combinational)
     iii. Sequential Multiplier using shift & accumulate
     iv. Booth Algorithm (Original algorithm as seen in right figure)





2. Implement a testbench to test the above multipliers: Covering 8 cases:
   - Multiplication of positive and negative number
   - Multiplication of positive and positive number
   - Multiplication of negative and negative number
   - Multiplication of negative and positive number
   - Multiplication by zero
   - Multiplication by 1
   - Additional 2 random test cases.
   - Your testbench should print "TestCase#1: success" on success and should print the "TestCase#1: failed with input X and Y and Output Z and overflow status N", elements in blue should be replaced by your values.
   - Your testbench should also report the total number of success and failure test cases at the end.

3. Synthesis the multipliers and add the following constraints
   i. Set the clock to 2ns.
   ii. Set Input delay to 0.2ns.
   iii. Set load to 10
   iv. Set output load to 0.5ns.
   v. Set Utilization to 60%
   vi. Enable usage of all library cells.
   - Report: Total Area, Max Delay, Max Slack, Min Slack, Total Power, clk.
   - If a design suffers from -ve slack, adjust the timing constraints.
   - Make sure that all designs work on the same constraints after modification.
4. Apply post-synthesis simulation using your testbench **(add sdc file)**.
5. Using the result you got from synthesis, use the most appropriate multiplier (from your point of view) to create a **32-bit floating point multiplier (IEEE-standard)**
6. Repeat the steps from 2-5 for the floating point multiplier.

Deliverables:

- For each multiplier, a folder content the following
  - Code files for Design
  - Code file for testbench.
  - Do file to run and configure wave.
  - Constraints files
  - Scripts used for synthesis
  - Scripts used for Floorplanning, Placement & Routing
  - Generated reports
  - Post-synthesize code
  - Screenshots of 2 simulations: pre-synthesis and post-synthesis.

- Excel Sheet containing the reported results in a tabular form (use python). For example

	Verilog (*)	Multiplier Tree	Sequential Multiplier	Original Booth	Float multiplier
Min Delay					
Max Delay					
Clock					
Total Power					
Area					
Utilization					
Number Latches					

- The Original reports of each design.
- A presentation containing the following
  - Your fullNames and codes (9XXXXXX)
  - Explanation of each adder design (including floating point).
  - Justification of your choice of adder used with floating point.
  - Additional challenges you faced.
  - Roles of each team member and estimate time s/he worked.
- A video for each showcasing your understanding of the work done and results uploaded on google drive with a shared link provided.

Due Dates:

- All files should be zipped together and uploaded on GoogleClassroom by only one team member
- Due Date is the **21st of December**.

Final Remarks:

- Teams will be ranked according to the smallest area (using **the same timing constraints** and that is the design is **<u>full functioning</u>**).
- feel free to ask google for ideas **NOT CODE** .
- **COPYING from the internet will be strongly penalized. (max of 10% of the code is allowed to be copied given you understand it).**
- Don’t use clocks in combinational circuits.
- Don’t generate unnecessary latches.