Mini Project#2

Multipliers Mania

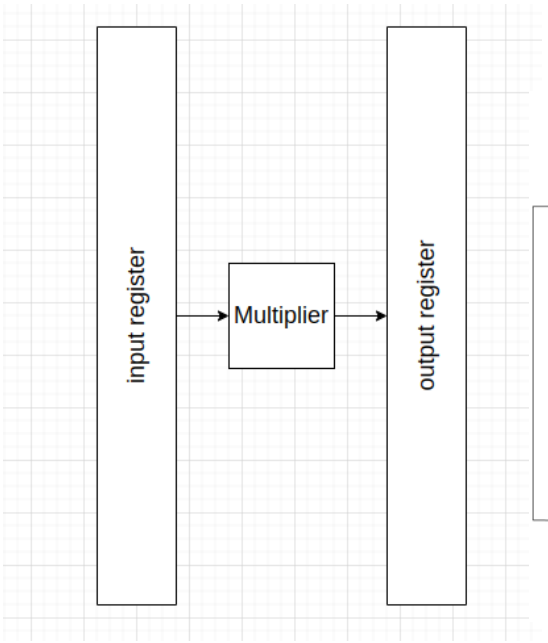
Overview:

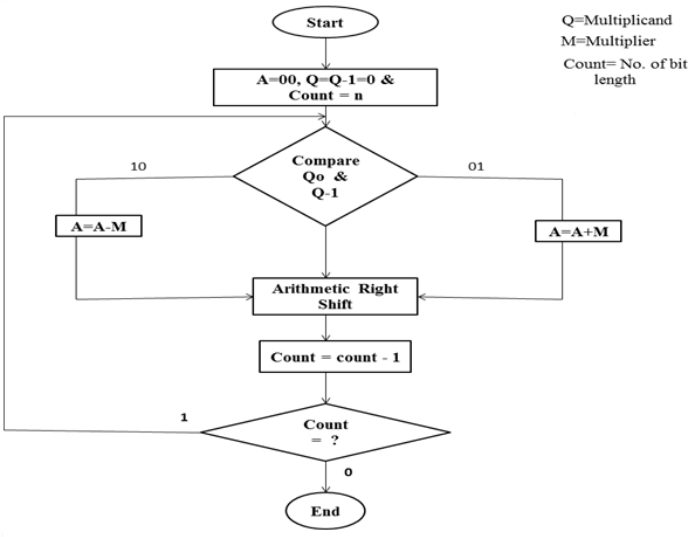
Another important block in “compute” unit is multipliers. Their performance impacts the whole chip. This is why in this mini project, we will explore different implementations of multipliers and study their characteristics.

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)
        v. Radix-4 Booth Algorithm.
        vi. Modified Booth Algorithm (for 5th member in the team only)





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 testcases at the end.

3. Synthesis the multipliers and add the following constraints
    i. Set 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 previously made testbench.
5. Place and route the multipliers with similar constraints to synthesis.
    - Constraint clock skew to 0.2ns
    - Only use vertical strips
    - Report: Total Area, Utilization, Max Delay, Min Delay, worst 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.
6. Apply post-routing simulation using your previously made testbench. **(include your sdf file)**
7. Generate the final GDS file for each design.
8. 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)**
9. Repeat the steps from 2-7 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
  - Oasys generated reports
  - Nitro generated reports
  - Post-synthesize code
  - Post-routing code
  - Sdf file
  - GDS
  - Final saved database from Nitro
  - Screenshot from oasys netlists(showing details not just high level)
  - Screenshot from Nitro final layout.
  - Screenshots of 3 simulations, pre-synthesis, post-synthesis, post-routing

- Excel Sheet containing the reported results in a tabular form. For example

	Verilog (*)	Multiplier Tree	Sequential Multiplier	Original Booth	Radix-4 Booth	Float multiplier
Min Delay						
Max Delay						
Clock						
Total Power						
Area						
Utilization after routing (not after export)						
Number Latches/Flip Flops.						

- A presentation containing the following
  - Your fullNames
  - Explanation of each multiplier design (including floating point, you can use netlist from oasys to show block diagram of each module).
  - Justification of your choice of adder/multiplier used with floating point.
  - Add screenshots of layout for the generated multipliers showing power routing/ detailed routing/ placement, …etc.
  - Additional challenges you faced.
  - **Roles of each team member** and estimate time s/he worked.

Due Dates:

- All files should be zipped together and uploaded on GoogleClassroom by only one team member
- Due Date is the 20th **of December at midnight**. Discussion will be with Phase three isA.

Final Remarks:

- Teams will be ranked according to the smallest area (using **the same timing constraints** and that is the design is **full functioning**).
- 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).**
- You will be asked to explain the multipliers, make sure you do understand what you implement.
- Don't generate unnecessary latches. (please)
- Different members in the same team can have different grades, make sure your work is balanced with others.