COA Assignment Report

Course: Computer Organization and Architecture (COA)

Assignment Title: Verilog Implementation of 16-bit Multipliers

Instructor: Prof. Rajat Sadhukhan

Name: Vishal Kumar Shaw

Roll Number: 24114105

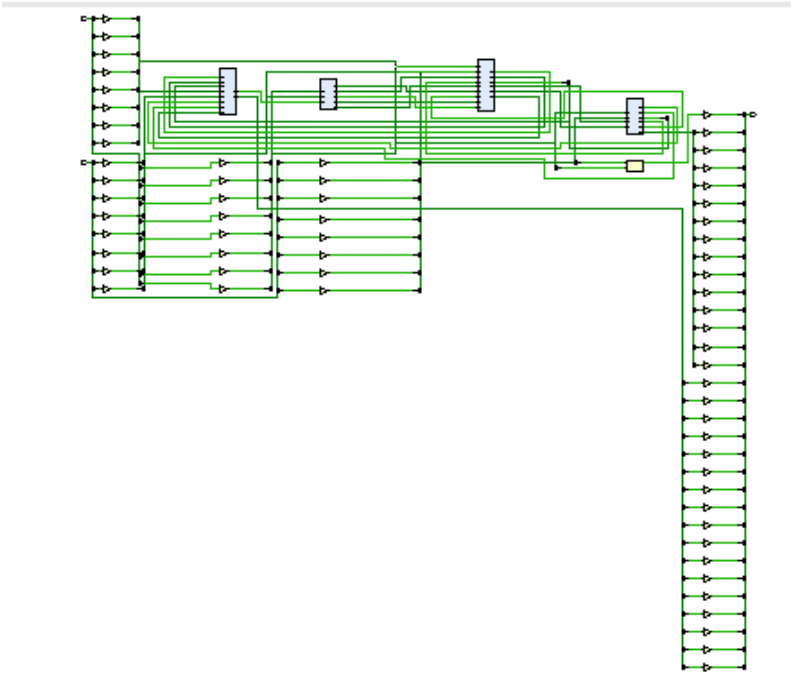
1. Objective

To optimize Wallace tree multiplier and implement it on vivado using Verilog code and compare it with a 16-bit array multiplier. See their design, block diagram and their waveform on different inputs and compare which is faster.

2. Design Description

2.1 Array Multiplier

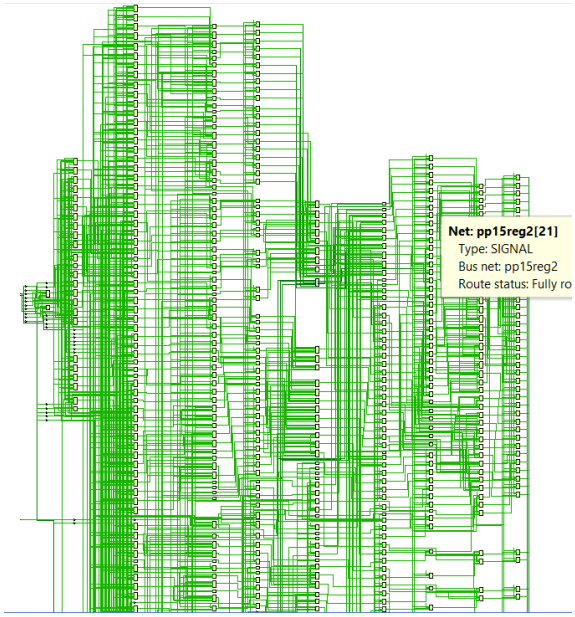
My logic for array multiplier was I am doing divide and conquer method like for 16x16 bit multiplier I'm using 4 8x8 bit multiplier such that first multiplying both 15:8 but of A with 15:8 bit of B than 15:8 bit of A with 7:0 bit of B adding it with 15:8 bit of B with 7:0 bit A and for last 7:0 bit A with 7:0 bit of B and in 8x8 calling 4x4 for the 8 bit multiplier and so on.

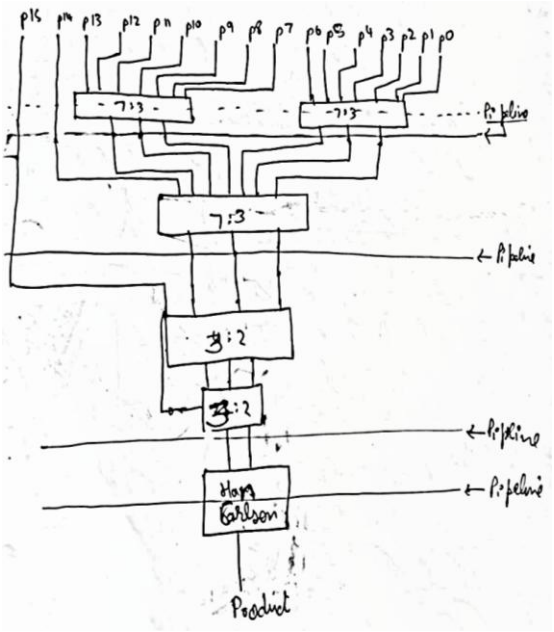


Array Multiplier (From Vivado)

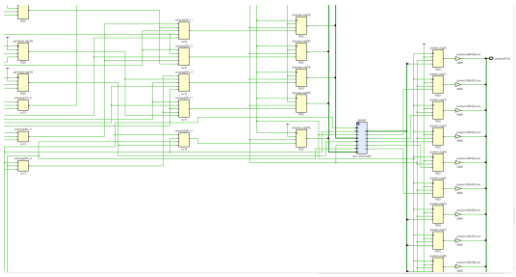
2.2 Wallace-Tree Multiplier

For the 16-bit array multiplier that I have implemented I used 7:3 and 3:2 compressors for the addition of partial product generated and making tree of partial product addition and for the last stage I after learning so many adders like 32-bit CLA, Hans-Carlson adder, Kogge Stone Adder etc. I chose Hans-Carlson Adder as it is fast enough and also requiring less area to achieve that speed.





block diagram of Wallace tree multiplier for a single bit



Wallace Tree Multiplier(From Vivado)

3. Verilog Code

[24114105 Vishal Kumar Shaw - Google Drive]

4. Simulation & Testbench

test_vectors_a[0] = 16'h0000; test_vectors_b[0] = 16'h0000;

test_vectors_a[1] = 16'h0001; test_vectors_b[1] = 16'h0001;

test_vectors_a[2] = 16'h0003; test_vectors_b[2] = 16'h0004;

test_vectors_a[3] = 16'h00FF; test_vectors_b[3] = 16'h000F;

test_vectors_a[4] = 16'h0F0F; test_vectors_b[4] = 16'h00F0;

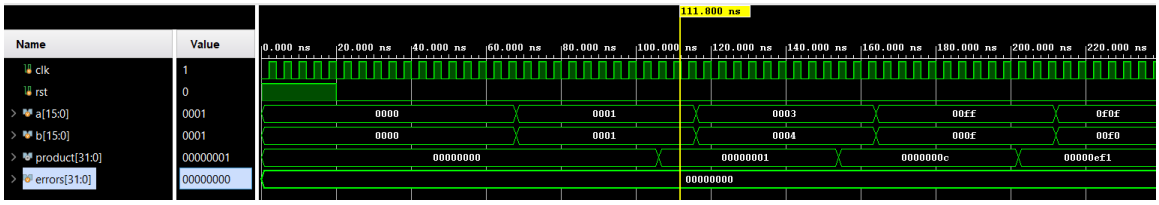
test_vectors_a[5] = 16'hAAAA; test_vectors_b[5] = 16'h5555;

test_vectors_a[6] = 16'h1234; test_vectors_b[6] = 16'h5678;

test_vectors_a[7] = 16'h8000; test_vectors_b[7] = 16'h0002;

test_vectors_a[8] = 16'h7FFF; test_vectors_b[8] = 16'h7FFF;

test_vectors_a[9] = 16'hFFFF; test_vectors_b[9] = 16'hFFFF;



Wallace multiplier waveform having a little latency because of pipeline in code

Array Multiplier Waveform

5. Synthesis Results

Multiplier	LUTs	FFs	DSPs	Max Freq (MHz)	Critical Path Delay (ns)
Array	466	0	0	49.581	20.169ns
Wallace-Tree	572	701	0	256.213	3.903ns

6. Analysis & Discussion

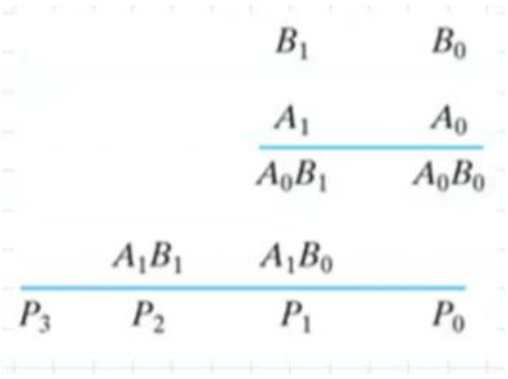
In array multiplier first I use the 16x16 bit multiplier inside that 8x8 bit multiplier is called 4 times inside each call of 8x8 bit multiplier I called 4x4 bit multiplier 4 times and inside each 4x4 multiplier call I called 2x2 bit multiplier where I calculate the product as using this and implemented it using and gates and xor gates when it is computed it returns to 4-bit multiplier and then 4-bit multiplier uses the same principle can calculates its result and then returns the value to 8 bit and then to 16-bit multiplier.

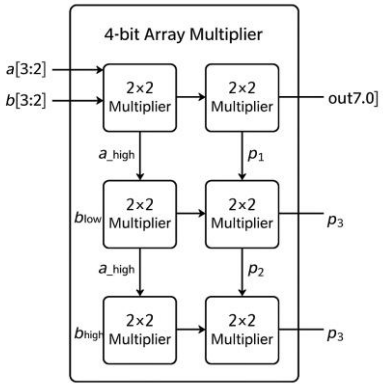
My recurrence relation:

$T(n)=4T(n/2) +O(n)$

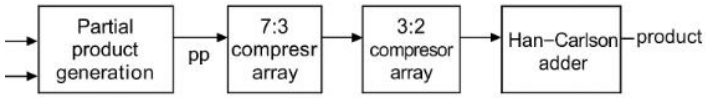
Over all complexity: $O(n^2)$.

Critical Path $O(n*\log n)$

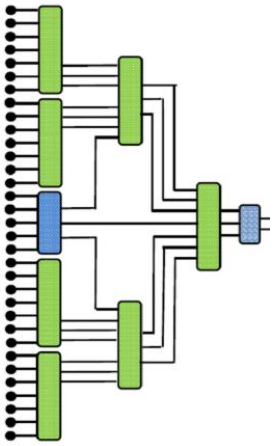




In Wallace tree multiplier we first calculate all the partial product than during the addition of partial product we add them in stages as a tree by using some compressor in my code I used 7:3 compressor and 3:2 compressor, 7:3 compressor take 7 input bit and give 3 out bit one sum bit one carry(<<1) bit and one carry_next (<<2) bit and in one stage what will happen to our 16 bit we make 2 set of 7 bit and use the 7:3 compressor and convert our 16 bits to 8 bit in the next stage what we will do we will take 7 bit from our remaining 8 bit and use the 7:3 compressor and convert it into 4 bit than we will use 3:2 compressor 2 times in the 3rd stage and in each stage I added pipeline to increase the throughput of the multiplier in the last stage we will add the final sum bit and carry bit using some adder in my code I used Hans-Carlson adder which a type of parallel prefix adder which is a hybrid of Kogge-Stone and Brent-Kung making balance between speed and hardware in this we first calculate the propagate bit and generate bit for every bit and then we will build the tree by using the black – grey cell logic.



Stages in my Wallace tree



7:3 compressor critical path

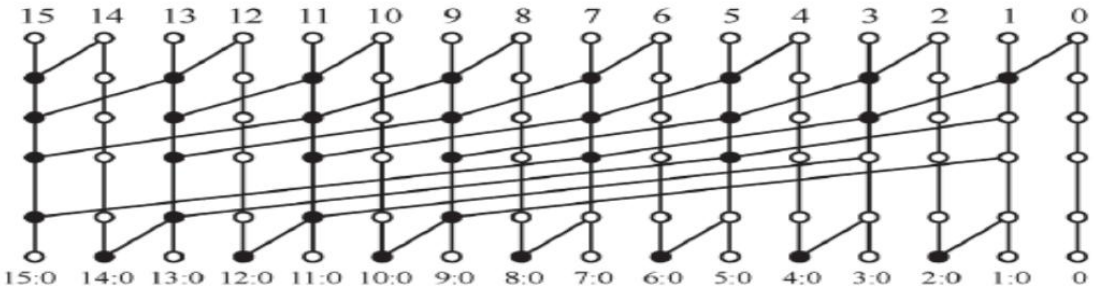


Figure 1: Han-Carlson parallel-prefix topologies. n=16

Near LSB works like Kogge Stone and near MSB works like Brent Kung.

The 4 main stage of Hans-Carlson adder -:

A. Pre-processing -> Compute all generate and propagate bit (G,P).
B. Prefix Tree-> first compute (g',p') for pair than for 4 bit span than continues
C. Carry Propagation -> Each bit carry is computed using prefix tree
D. Post-Processing -> compute sum bit and final carry bit.

And for the pipeline thing I added pipeline after partial product calculation than after using 7:3 compressor than after 7:3 compressor than after 3:2 compressor than inside the Han Carlson adder I also added pipeline ad each stage of Han-Carlson adder as mentioned above.

In the Wallace tree multiplier since I used 7:3 compressor the depth of the tree is $\log_{\frac{7}{3}} n$ and in Han-Carlson the number of stages is $\log_2 n$ and each stage take O(1) to evaluate so the overall time complexity will come out to be $O(\log_{\frac{7}{2}} n + \log_2 n)$

And in Han-Carlson I also added pipeline in between these 4 stages. My data path delay for the Wallace tree is coming out to be 3.906ns.

7. Conclusion

For this I learnt how to code, simulate my code, synthesis my design on Vivado. I read about different types of compressors like **3:2, 4:2,5:3,7:3 and 7:2** and compared to 4:2 or 5:3 compressors, 7:3 reduces tree depth by 30% while maintaining a compact design. So 7:3 is best for Wallace multiplier in my opinion as it makes the depth lesser and also take less hardware to implement. And for the last stage of the Wallace tree multiplier I read about different types of adder such as **CLA, RCA, CSV, Han-Carlson Adder, Kogge Stone Adder, Brent Kung Adder** and I found the **Kogge is the fastest** adder among these and **Brent Kung is slightly slow but take less wiring** and **Han-Carlson is the hybrid of both** it is as fast as Kogge and have less wiring as Brent Kung and CLA and CSV are slow. so, I chose this to maintain balance between speed and cost. And I also introduced **pipeline** in the stages where I can in order to increase the throughput of the design.
So ultimately, I chose **7:3 compressor and Han-Carlson Adder** for last stage adding for better optimization of both time and hardware.
While both multipliers are functionally equivalent, the Wallace Tree achieves 5× speedup at the cost of 20% more LUTs and flip-flops. But for this speed the cost compromise gets validated.

8. References

https://ieeexplore.ieee.org/document/4230999

https://ijiet.com/wp-content/uploads/2016/06/16011.pdf

https://ieeexplore.ieee.org/document/6292146

https://www.elprocus.com/kogge-stone-adder/

https://ieiespc.org/ieiespc/XmlViewer/f415667

https://vlsiverify.com/verilog/verilog-codes/wallace-tree-multiplier/

https://imanagerpublications.com/assets/htmlfiles/JCIR()5929.html

Morris Mano-> https://docs.google.com/file/d/0B8-drkZsESDnN2NmYTQxYjQtYTMwZi00N2IzLTkxNjgtZjI1NTZiN2FjNDli/edit?resourcekey=0-Yk8bAsCt9I5epBNFTG8KMQ

Harris and Harris -> https://books.google.co.in/books?id=5X7JV5-n0FIC&printsec=frontcover&redir_esc=y#v=onepage&q&f=false