1 Introduction

1.1 Referenced documents

- Shiann-Rong Kuang, "**Modified Booth Multipliers With A Regular Partial Product Array**", *IEEE Transactions On Circuits And Systems—Ii:* Express Briefs, Vol. 56, No. 5, May 2009.
- Jung-Yup Kang, "**A Simple High-Speed Multiplier Design**", *IEEE Transactions On Computers*, Vol. 55, No.10, October 2006.
- Kuang SR, Wang JP. "**Design of power-efficient configurable booth multiplier"**. *IEEE Transactions on Circuits and Systems* I: Regular Papers 2010; 57 (3): 568-580. doi: 10.1109/TCSI.2009.2023763.
- R. P. Rajput and M. N. S. Swamy, "**High Speed Modified Booth Encoder Multiplier for Signed and Unsigned Numbers,**" *2012 UKSim 14th International Conference on Computer Modelling and Simulation*, Cambridge, UK, 2012, pp. 649-654, doi: 10.1109/UKSim.2012.99.
- Neil Weste and David Mooney Harris, *CMOS VLSI design : a circuits and systems perspective*, fourth edition. Pearson Education, Inc., 2011.
- Max Koretskyi, **How to round binary numbers**. inDepthDev. Retrieved Oct 8, 2023, from <https://indepth.dev/posts/1017/how-to-round-binary-numbers>.

1.2 Design library name

uk65lscllmvbbr_100c25_tc_ccs

1.3 People involved in the block

1. Rahul (2023EEN2238)

2 Function

2.1 Brief overview

- The multiplier stage uses a wallace tree multiplier with a radix-4 booth encoder decoder and a 32 bit carry select adder to multiply the two 16 bit signed fractional inputs (1 sign + 3 integer + 12 fractional).
- The output of multiplier is accumulated in the Accumulator register using a 40 bit carry select adder (1 sign + 15 integer + 24 fractional)
- The number in accumulator is then rounded off to 11 fractional bits using "to the nearest; ties to even" rule and stored in 27 bit output register (1 sign + 15 integer + 11 fractional).

- To achieve an operating frequency of 150MHz, pipelining has been done which results in a latency of 3 clock cycle or 20ns.

2.2 Interfaces

Table 1: Description of I/Os used in the 16-bit MAC.

Signal Name	Input/Output (I/O)	Description
MD <15:0>	Input	16 - bits of multiplicand (1 sign + 3 integer + 12 fractional)
MR <15:0>	Input	16 - bits of multiplier (1 sign + 3 integer + 12 fractional)
OUT_ROUNDED <26:0>	Output	27-bit rounded off output of MAC (1 sign + 15 integer + 11 fractional)
V_{DD}	Input/Output	DC supply voltage
V_{SS}	Input/Output	Ground

2.3 Architecture

- 16-bit radix 4 booth encoded wallace tree multiplier consists of following blocks:
    1. 9 18-bit partial product generator
    2. 7 32-bit carry save adder
    3. 1 32-bit carry select adder

- Accumulator block comprises of a 40 bit Carry Select adder.

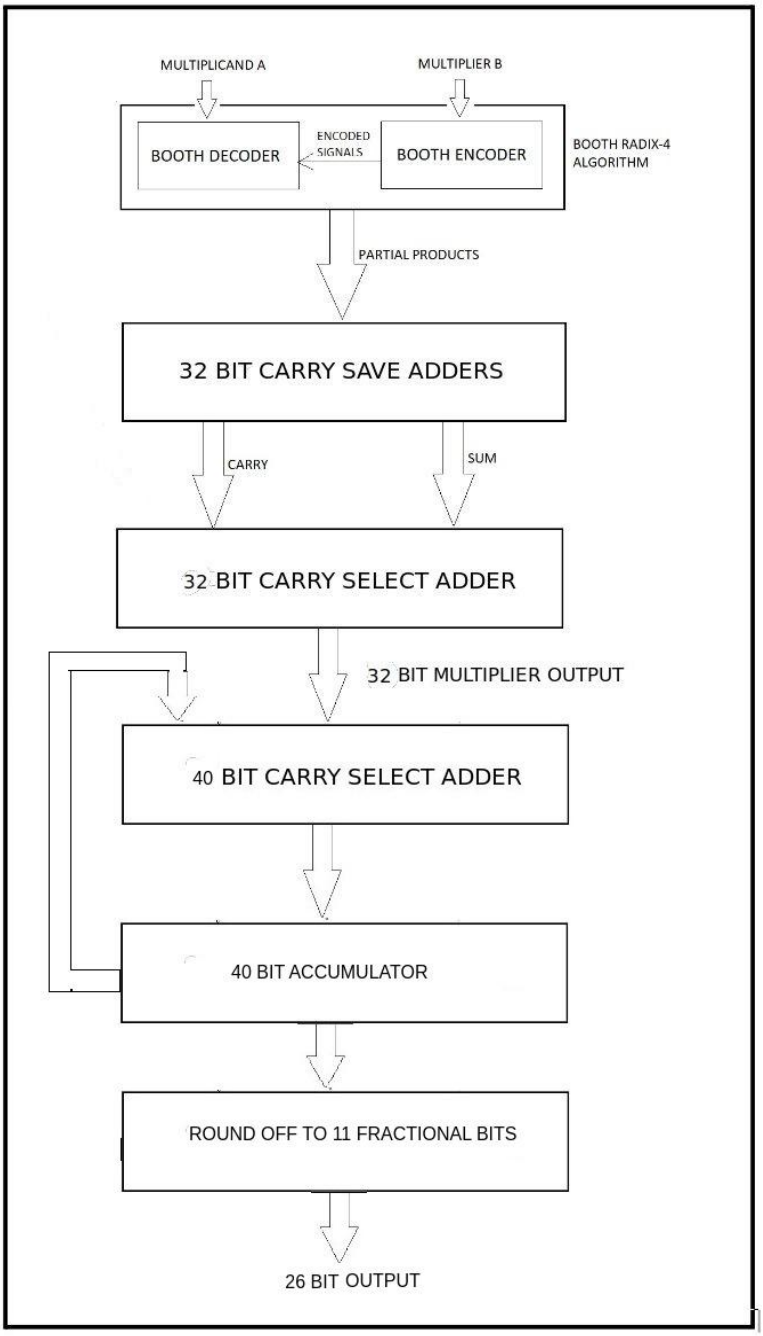
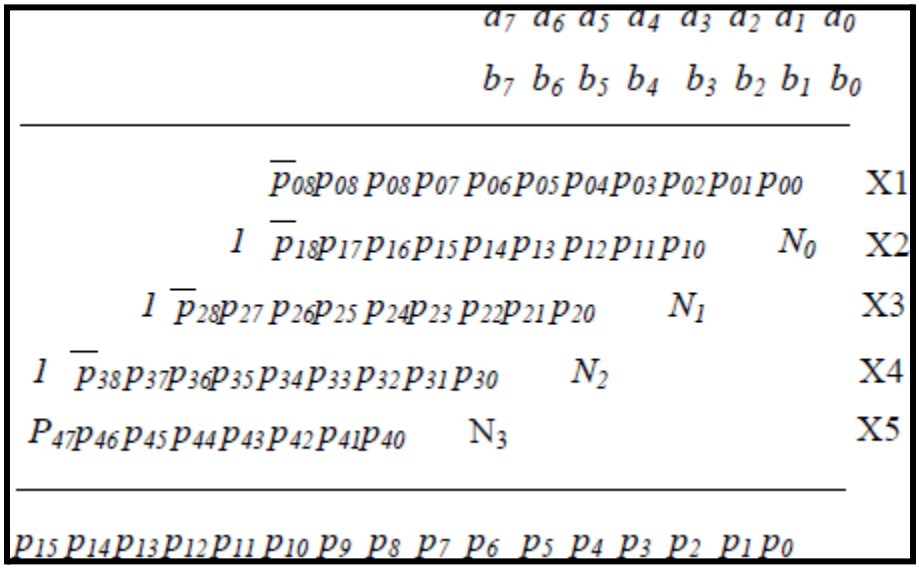


Fig 1: Architecture of radix 4 Booth Encoded Wallace Tree Multiplier.

2.4 Detailed functional description

1. 18-bit Partial Product Generator

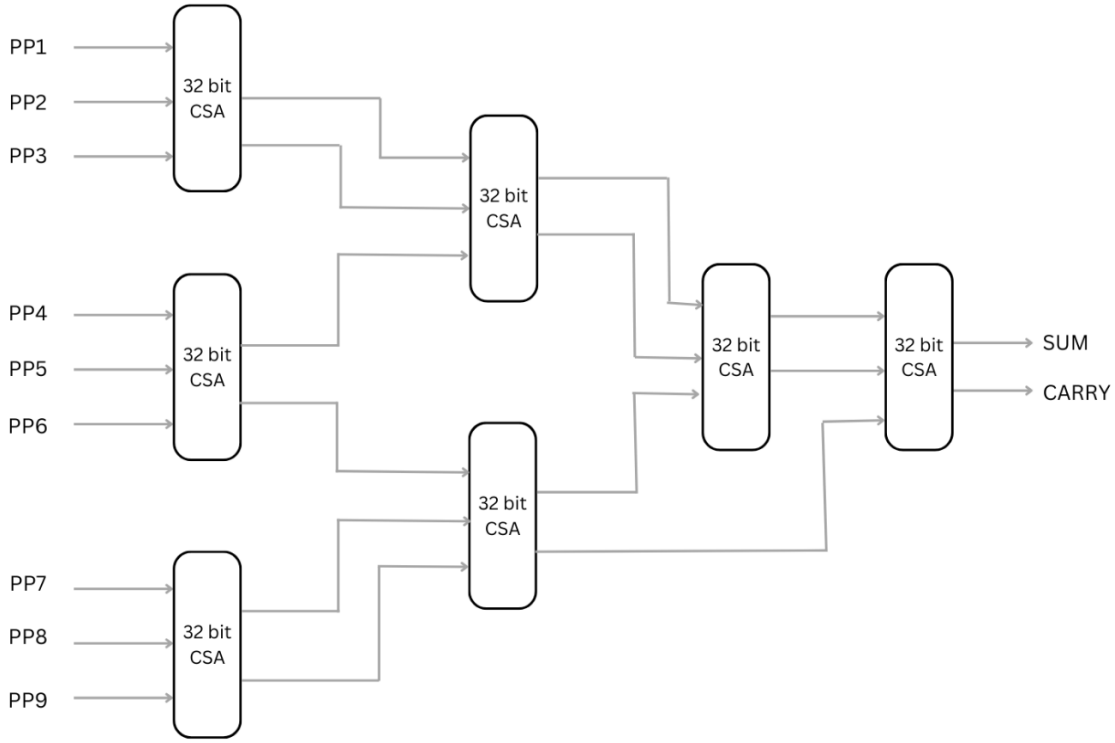
- This is responsible for generating the partial products that needs to be added together to get the final output by taking 8 bit multiplier and multiplicand as input. Since for a radix-4, 8-bit booth multiplier (N/2 + 1) partial products are generated, 9 such partial product generator are needed.
- Both multiplier and multiplicand are sign extended with 2 bits at MSB and 1 zero at LSB so that the partial product generator can give the correct output.
- The MSB bit of the output tells us whether the partial product is in 2's complement form or not. If MSB is 1, then it's 2's complement form otherwise a positive number.
- The MSB bit is extended to all the other remaining bits to get a 32 bit partial product.
- One thing to note here is that the partial product generator do not perform the complete operating of taking 2's complement of multiplicand but only inverts the multiplicand. The 1 is added to LSB by putting Negate bit N (0 for +ve PP and 1 for -ve PP) in the next partial product just below the LSB of previous partial product as shown below:



2. 32-bit Carry Save Adder

- Carry save adder has been used here to minimize the time required to add all the 9 partial products. In a carry save adder, N number of full adders are used for N bit carry save adder.
- Each full adder works independently and takes 3 bits from 3 numbers that needs to be added and gives the sum and carry at the output.
- By taking all the sum bits as one number and carry bits as another number, we can add them using any of the available adders like RCA, CLA, CseLA, etc to get the final output.

- Since we need to add 9 partial products, 7 carry save adders are connected in the following configuration to get the final sum and carry bits.



3. 4-bit Carry Lookahead Adder

Two 4 bit signed/unsigned number with carry in is fed as input. The two 4 bit numbers are represented as A and B whose individual 4 bits are A3, A2, A1, A0 and B3, B2, B1, B0 respectively. Using Half adders Carry Generator and Carry Propagator term is calculated.

$Pi = Ai \oplus Bi$, where P_i is carry propagator
$Gi = Ai \cdot Bi$, where G_i is carry generator

The sum output Si and carry output C_{i+1} can be expressed in terms of Pi and Gi as:
$Si = Pi \oplus Ci$
$Ci+1 = Gi + CiPi$

$C1 = G0 + C0P0$
$C2 = G1 + P1G0 + P1P0C0$
$C3 = G2 + P2G1 + P2P1G0 + P2P1P0C0$
$C4 = G3 + P3G2 + P3P2G1 + P3P2P1G0 + P3P2P1P0C0$

- Such 4-bit CLA's are used to implement various carry select adders.

4. **Rounding off fractional bits**

- The general rule when rounding binary fractions to the n-th place prescribes to check the digit following the n-th place in the number. If it's 0, then the number should always be rounded down.
- If, instead, the digit is 1 and any of the following digits are also 1, then the number should be rounded up.
- If, however, all of the following digits are 0's, then a tie breaking rule must be applied and it's 'ties to even' in our case.

3 Design parameters

3.1 Performance Requirements

- Operating frequency required: 150MHz
- Able to perform signed operations

3.2 Clock Distribution

Frequency: 150MHz
Duty cycle: 50%
Rise time: 0.13ns
Fall time: 0.13ns
Clock uncertainty: 0.01ns

3.3 Reset

Polarity: Active Low

At reset, the registers are assigned the following values:

```
acc <= 0;
MD <= 0;
MR <= 0;
out_rounded <= 0;
setup_count <= 0;
setup_completed <= 0;
count <= 0;
```

```
PP0_reg <= 0;
PP1_reg <= 0;
PP2_reg <= 0;
PP3_reg <= 0;
PP4_reg <= 0;
PP5_reg <= 0;
PP6_reg <= 0;
PP7_reg <= 0;
PP8_reg <= 0;
```

3.4 Timing Description

Latency: 20ns
Update conditions: At every positive edge of clock.

4 Verification Strategy

4.1 Objectives

The 16-bit MAC takes 2 16-bit signed inputs (1 sign + 3 integer + 12 fractional) and multiply them and accumulate it in the accumulator register. Accumulation is done for 256 cycles and then is reset. After every 256 inputs, we need to wait for 3 clock cycles to send next set of 256 inputs to let the MAC setup to give correct output.

Following things needs to be verified:

- Being able to correctly reset.
- Correct output is produced after every new set of 256 inputs.
- All corner cases are correct.
- Able to perform signed operations.

4.2 Tools and Version

- Syntheisis - Genus
- Place and Route - Innovus
- Simulation - ModelSim

4.3 Checking mechanisms

The testbench tests the 3 corner cases {(-8)*(-8) , 7.999*7.999, (-8)*7.999} and outputs whether the accumulation after 256 cycles is correct or not as shown below:

```
#   MD: -8.000000   MR: -8.000000   Simulation Output: 14912.000000
#   MD: -8.000000   MR: -8.000000   Simulation Output: 14976.000000
#   MD: -8.000000   MR: -8.000000   Simulation Output: 15040.000000
#   MD: -8.000000   MR: -8.000000   Simulation Output: 15104.000000
#   MD: -8.000000   MR: -8.000000   Simulation Output: 15168.000000
#   MD: -8.000000   MR: -8.000000   Simulation Output: 15232.000000
#   MD: -8.000000   MR: -8.000000   Simulation Output: 15296.000000
#   MD: -8.000000   MR: -8.000000   Simulation Output: 15360.000000
#   MD: -8.000000   MR: -8.000000   Simulation Output: 15424.000000
#   MD: -8.000000   MR: -8.000000   Simulation Output: 15488.000000
#   MD: -8.000000   MR: -8.000000   Simulation Output: 15552.000000
#   MD: -8.000000   MR: -8.000000   Simulation Output: 15616.000000
#   MD: -8.000000   MR: -8.000000   Simulation Output: 15680.000000
#   MD: -8.000000   MR: -8.000000   Simulation Output: 15744.000000
#   MD: -8.000000   MR: -8.000000   Simulation Output: 15808.000000
#   MD: -8.000000   MR: -8.000000   Simulation Output: 15872.000000
#   MD: -8.000000   MR: -8.000000   Simulation Output: 15936.000000
#   MD: -8.000000   MR: -8.000000   Simulation Output: 16000.000000
#   MD: -8.000000   MR: -8.000000   Simulation Output: 16064.000000
#   MD: -8.000000   MR: -8.000000   Simulation Output: 16128.000000
#   MD: -8.000000   MR: -8.000000   Simulation Output: 16192.000000
#   MD: -8.000000   MR: -8.000000   Simulation Output: 16256.000000
#   MD: -8.000000   MR: -8.000000   Simulation Output: 16320.000000
#   MD: -8.000000   MR: -8.000000   Simulation Output: 16384.000000
#
#
#
#   MD: -8.000000   MR: -8.000000   Simulation Output: 16384.000000  Correct Output: 16384  Test: Successful
#   MD: 7.999756   MR: -8.000000   Simulation Output: -16383.500000  Correct Output: -16383.5  Test: Successful
#   MD: 7.999756   MR: 7.999756   Simulation Output: 16383.000000  Correct Output: 16383  Test: Successful

VSIM 13>
```

5 Functional Checklist

- Able to perform signed operation
- Gives correct output on 3 corner cases {(-8)*(-8) , 7.999*7.999, (-8)*7.999}

5 Testbench

5.1 Overview

The testbench will test 3 corner cases {(-8)*(-8) , 7.999*7.999, (-8)*7.999} and before each corner case it will reset the MAC and compare the final result with actual result and displac in transcript/console whether test is successful or not.

5.2 Architecture

- At begining, reset signal is sent to the MAC to initialize all registers to 0
- In next clock cycle, reset is disabled and first corner case is sent.
- Waits for 256 cycles and outputs in the transcript/console whether the test is successful or not
- Reset signal is sent.
- 2nd corner cases is sent as input.
- Waits for 256 cycles and outputs in the transcript/console whether the test is successful or not

- Reset signal is sent.
- 3rd corner cases is sent as input.
- Waits for 256 cycles and outputs in the transcript/console whether the test is successful or not

6 Tests Specification

Simulation time: 7860ps

First corner case:

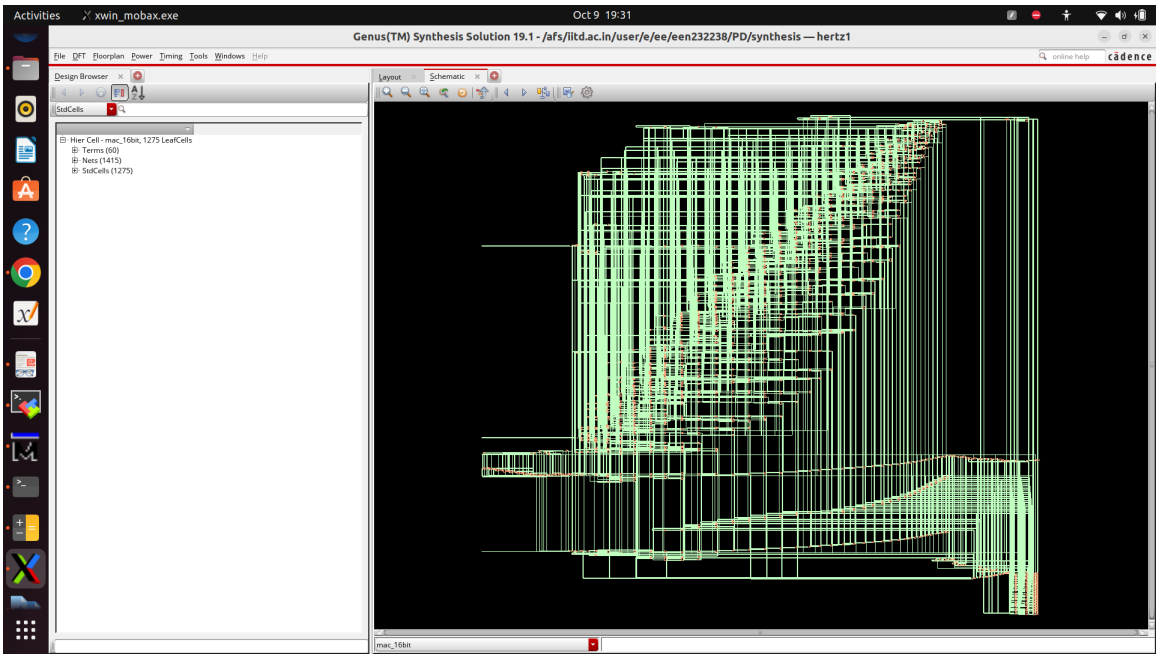
a = 1000.000000000000 (-8)
b = 1000.000000000000 (-8)

Second corner case:

a = 0111.111111111111 (7.999755859375)
b = 1000.000000000000 (-8)

Third corner case:
a = 0111.111111111111 (7.999755859375)
b = 0111.111111111111 (7.999755859375)

7 Design Microarchitecture



7.1 Top Level Interface

Not Applicable

7.2 Sub-Block Description

- Sub-blocks used in the overall design:
  1. 18-bit Partial Product Generators using the Booth Radix-4 algorithm.
  2. 32-bit Carry save adders.
  3. 40-bit Carry Select adder using Carry look ahead adders and 2:1 Mux.
  4. Rounding off circuit

1. Partial Product Generators

- The following logic implementation is used to generate one bit of the partial product, corresponding to each bit of the extended multiplicand, based on the 3-bit grouping of the multiplier as per the Booth Radix-4 algorithm.

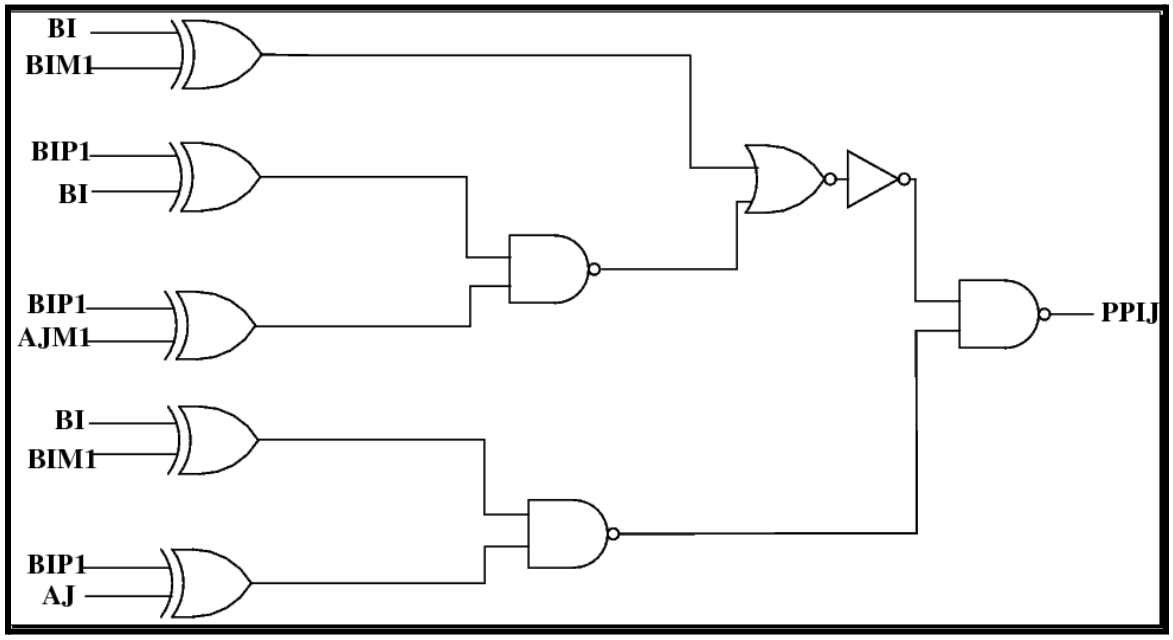


Fig 3: Schematic of Partial Product Generator

- A 10-bit partial product is to be generated corresponding to each 3-bit grouping of the multiplier. Hence, a total of 5 such 10-bit partial product generators are used to create the {(N/2)+1} partial products.
- Each block of Partial product generator has the following interfaces:
  1. 3 bits of the multiplier as input.
  2. 8 bits of the multiplicand as input.
  3. 10 bits of partial product as output.

2. **Carry Save Adders**
   - A Carry save adder is the most useful adder for our application.
   - It is a parallel ensemble of 16 Full adders without any horizontal connection.
   - Its main function is to perform the additions on the partial products.
   - The shifting of partial products is taken care of implicitly while routing.

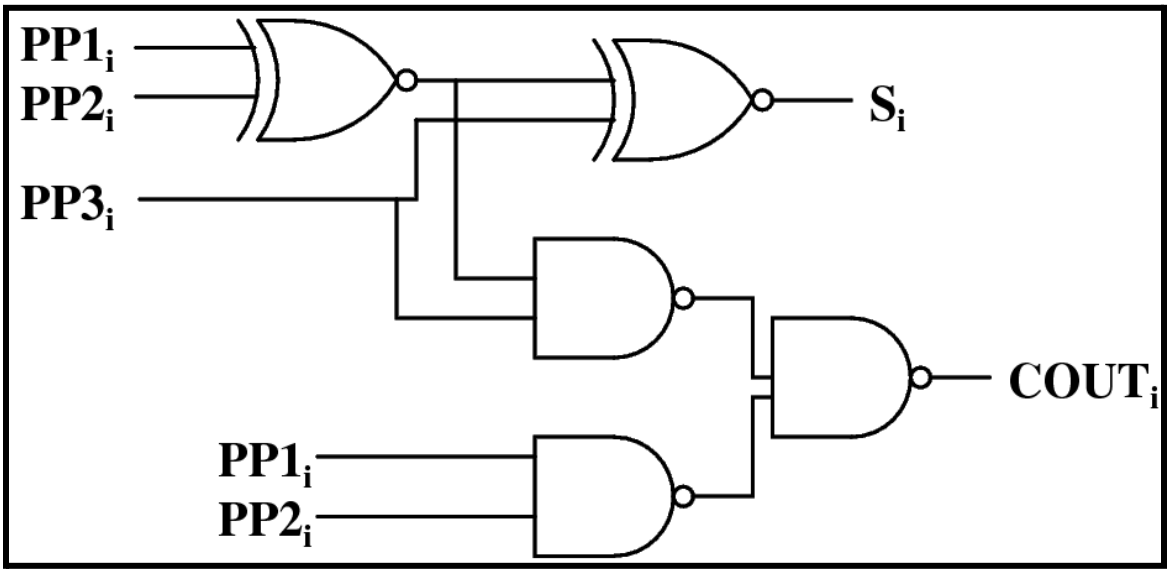


Fig 4: Schematic of Carry Save Adder

3. **Carry Look Ahead Adder**
   - Addition of the final Sum and Carry output from the cascade CSA stages were performed using 4 Carry look ahead adders.
   - The basic logic implementation of a 4-bit carry look ahead adder using the carry propagate and carry generate terms is shown below:

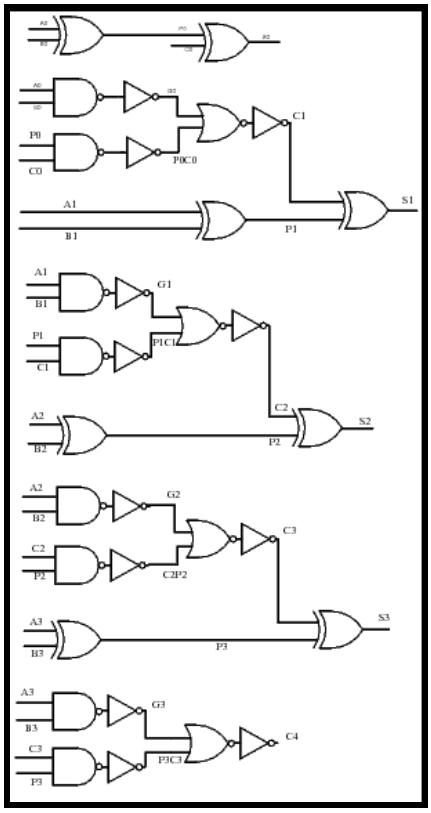
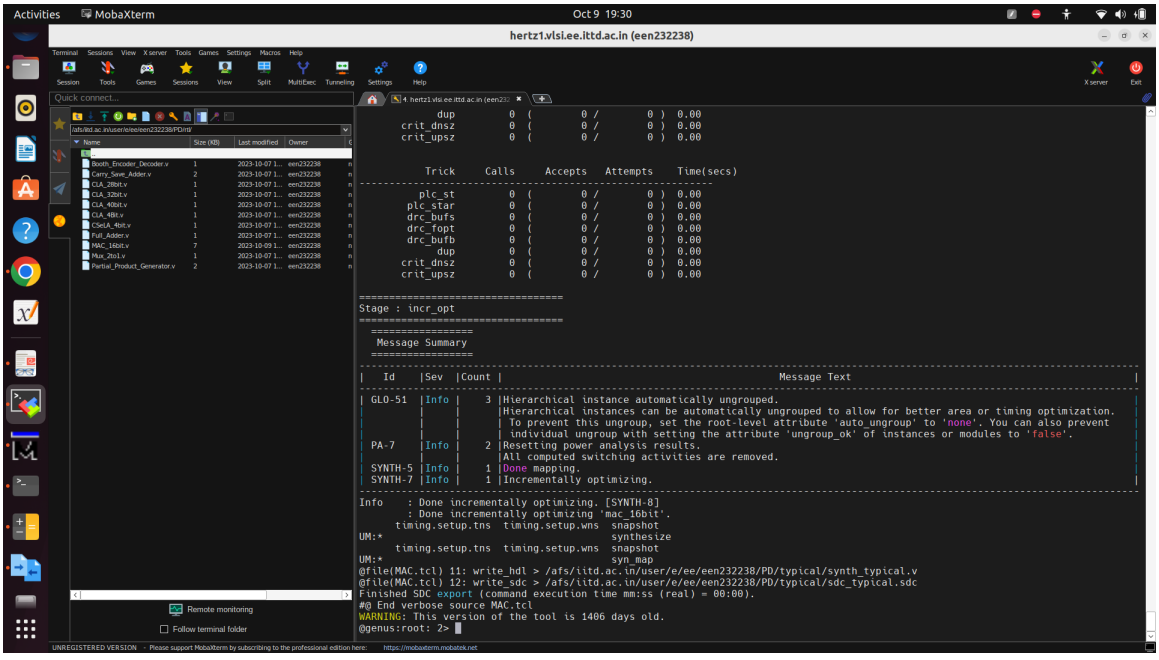


Fig - Schematic of Carry Look Ahead Adder

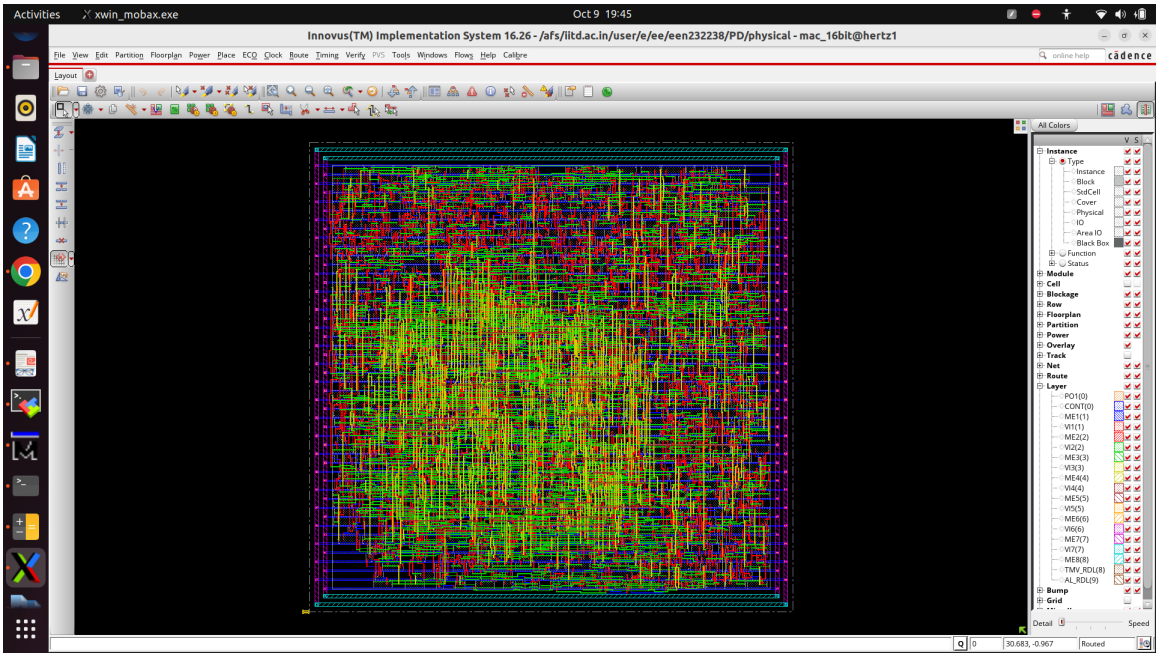
7.3 Structural Mapping Process

Genus was used to synthesise the RTL code, ann encounter to make the layout.



8 Physical hierarchy

8.1 Floorplanning



Area: 5186.52 um2
Aspect Ratio: 1:1
Shape: Square

8.2 Clocktree insertion

Not Applicable

8.3 Layout Strategy

- Core to Die Boundary from all sides: 4.5um
- Power planning:
    1. Ring
       Width: 0.7
       Spacing: 0.985
    2. Stripe:
       Width: 0.3
       Spacing: 0.4

- Routing: Timing driven and SI driven

9 Results

9.1 Area

Block Name	Standard Cells Count	Dimensions (W*L)	Layout Area(um2)
16 bit MAC	1275	72*72 um	5186.2

10.2 Timing

All timing constraints have been met and following worst negative slack has been observed:

WNS	Setup	Hold
Pre-CTS	+0.061	+0.042
Post-CTS	+0.042	+0.006

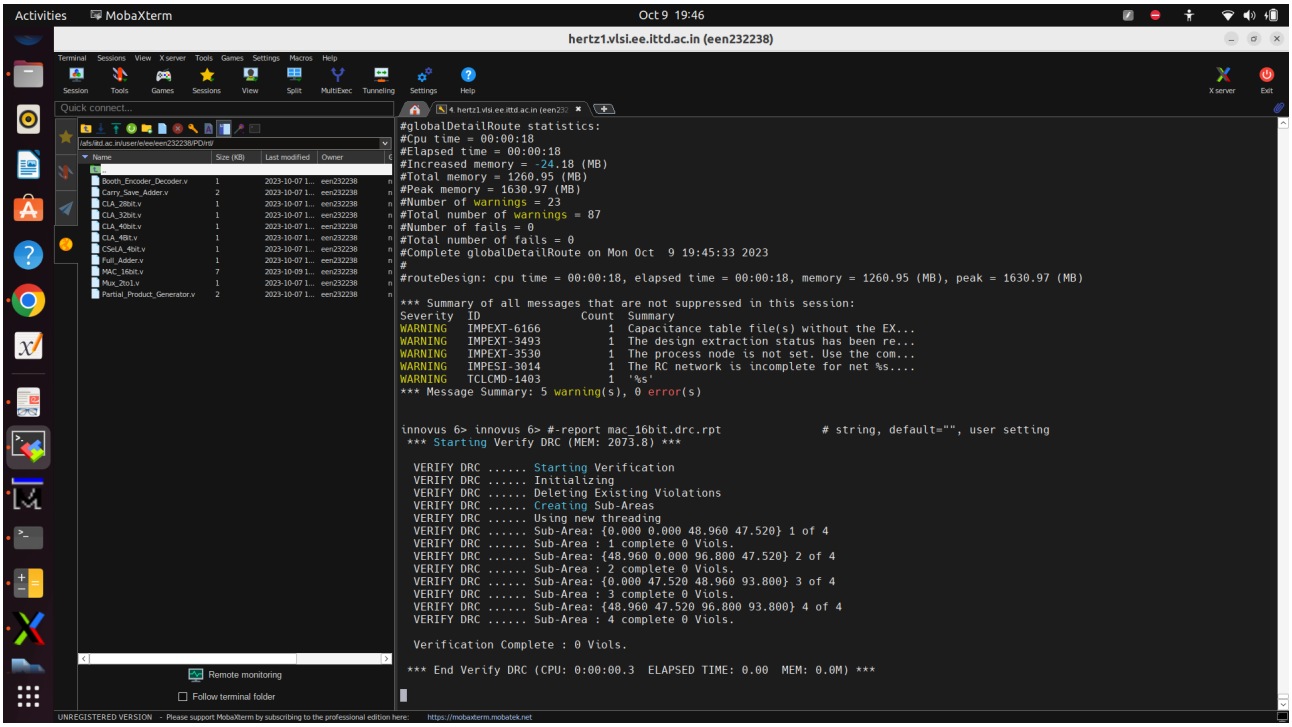
10.3 Testability analysis

Test signals

1. clk
2. rst

10.4 DRC rule violations

None.



11 Bugs known at submission date

None.