HOMEWORK II - REPORT

Şeyma ÇALIŞKAN

Intern at GSTL

Introduction

This report presents the design and verification of a 16-bit multiplier using various modules such as **Partial Product Generator (PPG)**, **Partial Product Accumulator (PPA)**, **4:2 Compressor**, and **Ripple Carry Adder (RCA)**. The implementation follows the Wallace Tree structure for efficient accumulation of partial products and is synthesized using OpenLane.

Multiplier Architecture

The multiplier consists of the following components:

- **PPG (Partial Product Generator):** Generates 16 partial products from the 16-bit multiplicand (X) and multiplier (Y).
- **4:2 Compressor:** Used to reduce the number of partial products in Wallace Tree.
- **PPA (Partial Product Accumulator):** Accumulates the partial products using the compressor.
- **RCA (Ripple Carry Adder):** Final adder stage to compute the product.

Design and Implementation

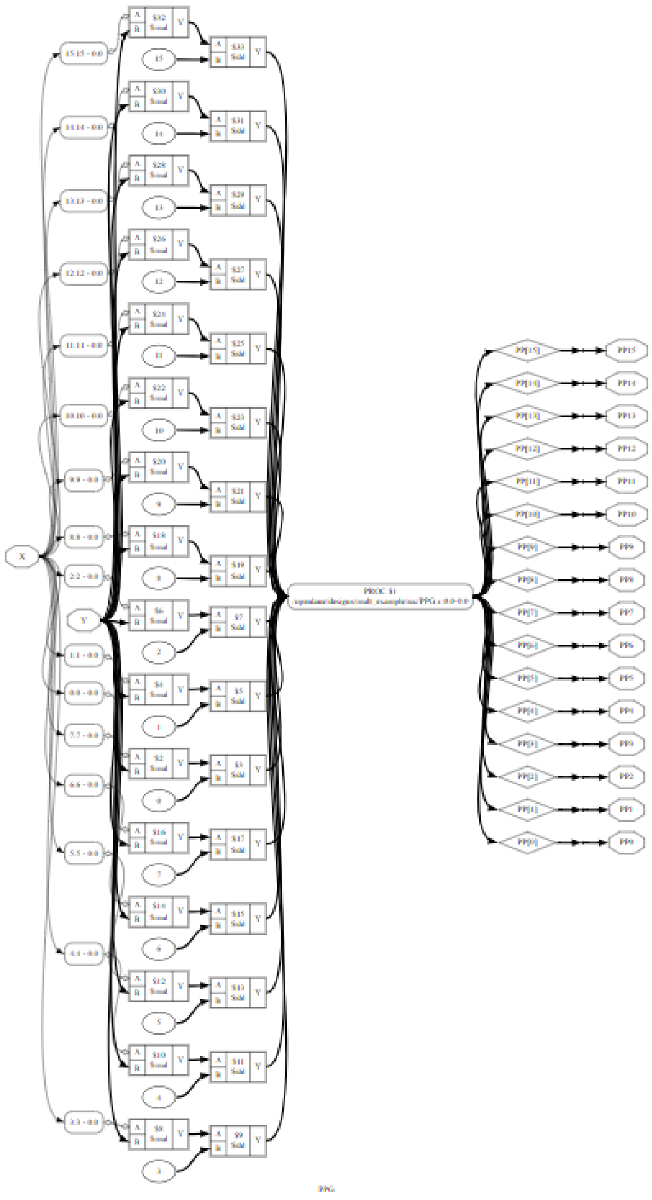
1) Partial Product Generator (PPG)

The `PPG.v` module generates the partial products by AND-ing each bit of the multiplier with the multiplicand.

OpenLane Synthesis

- The design was synthesized using OpenLane.
- The resulting netlist and gate-level schematic were obtained.
- Observations: The circuit consists mainly of AND gates and buffers.

Dot Schematics:



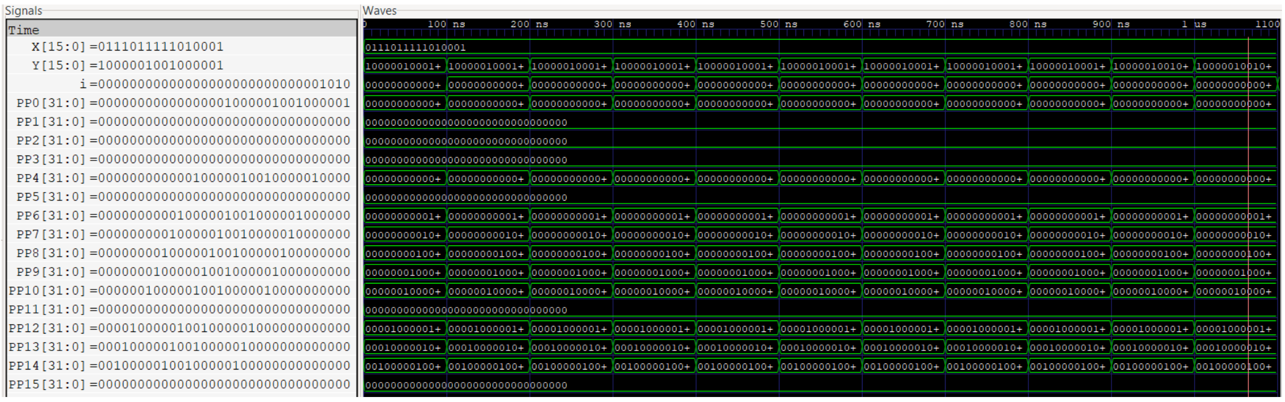
Behavioral Simulation (Icarus Verilog)

- A testbench was written to verify the correctness.
- Inputs were generated using the equation:

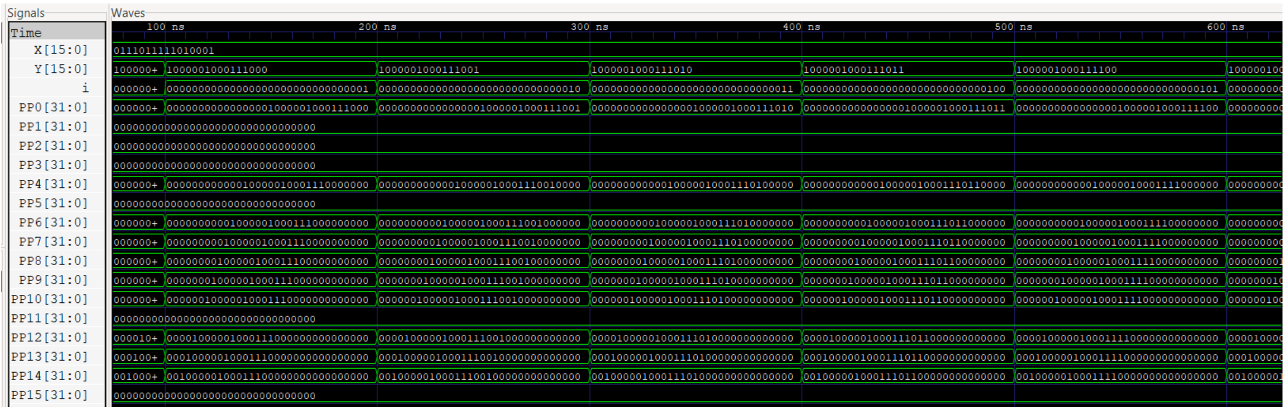
$$\text{For } 0 \le i \le 10,\ \{X, Y\} = ((\text{Student ID number} + i) \bmod 2^{32})_2$$

- The simulation results matched the expected outputs.

Behavioral Simulation Results:



```
Time
    X[15:0] =0111011111010001
    Y[15:0] =1000001001000001
          i =00000000000000000000000000001010
  PP0[31:0] =00000000000000001000001001000001
  PP1[31:0] =00000000000000000000000000000000
  PP2[31:0] =00000000000000000000000000000000
  PP3[31:0] =00000000000000000000000000000000
  PP4[31:0] =00000000000010000010010000010000
  PP5[31:0] =00000000000000000000000000000000
  PP6[31:0] =00000000001000001001000001000000
  PP7[31:0] =00000000010000010010000010000000
  PP8[31:0] =00000000100000100100000100000000
  PP9[31:0] =00000001000001001000001000000000
 PP10[31:0] =00000010000010010000010000000000
 PP11[31:0] =00000000000000000000000000000000
 PP12[31:0] =00001000001001000001000000000000
 PP13[31:0] =00010000010010000010000000000000
 PP14[31:0] =00100000100100000100000000000000
 PP15[31:0] =00000000000000000000000000000000
```



```
Time        100 ns                              200 ns                              300 ns                              400 ns                              500 ns                              600 ns
X[15:0]     0111011111010001
Y[15:0]     100000+  1000001000111000                  1000001000111001                  1000001000111010                  1000001000111011                  1000001000111100                  100000100
i           000000+  00000000000000000000000000000001  00000000000000000000000000000010  00000000000000000000000000000011  00000000000000000000000000000100  00000000000000000000000000000101  000000000
PP0[31:0]   000000+  00000000000000001000001000111000  00000000000000001000001000111001  00000000000000001000001000111010  00000000000000001000001000111011  00000000000000001000001000111100  000000000
PP1[31:0]   00000000000000000000000000000000
PP2[31:0]   00000000000000000000000000000000
PP3[31:0]   00000000000000000000000000000000
PP4[31:0]   000000+  00000000000010000010001110000000  00000000000010000010001110010000  00000000000010000010001110100000  00000000000010000010001110110000  00000000000010000010001111000000  000000000
PP5[31:0]   00000000000000000000000000000000
PP6[31:0]   000000+  00000000001000001000111000000000  00000000001000001000111001000000  00000000001000001000111010000000  00000000001000001000111011000000  00000000001000001000111100000000  000000000
PP7[31:0]   000000+  00000000010000010001110000000000  00000000010000010001110010000000  00000000010000010001110100000000  00000000010000010001110110000000  00000000010000010001111000000000  000000000
PP8[31:0]   000000+  00000000100000100011100000000000  00000000100000100011100100000000  00000000100000100011101000000000  00000000100000100011101100000000  00000000100000100011110000000000  000000001
PP9[31:0]   000000+  00000001000001000111000000000000  00000001000001000111001000000000  00000001000001000111010000000000  00000001000001000111011000000000  00000001000001000111100000000000  000000010
PP10[31:0]  000000+  00000010000010001110000000000000  00000010000010001110010000000000  00000010000010001110100000000000  00000010000010001110110000000000  00000010000010001111000000000000  000000100
PP11[31:0]  00000000000000000000000000000000
PP12[31:0]  000010+  00001000001000111000000000000000  00001000001000111001000000000000  00001000001000111010000000000000  00001000001000111011000000000000  00001000001000111100000000000000  000010000
PP13[31:0]  000100+  00010000010001110000000000000000  00010000010001110010000000000000  00010000010001110100000000000000  00010000010001110110000000000000  00010000010001111000000000000000  000100000
PP14[31:0]  001000+  00100000100011100000000000000000  00100000100011100100000000000000  00100000100011101000000000000000  00100000100011101100000000000000  00100000100011110000000000000000  001000001
PP15[31:0]  00000000000000000000000000000000
```

2) 4:2 Compressor

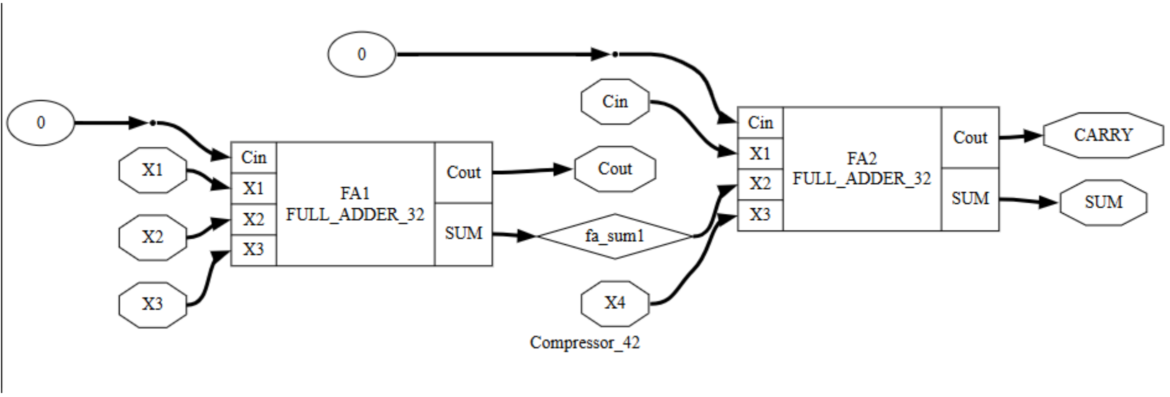
Verilog Code Implementation

The Compressor_42.v module adds five 32-bit inputs (X1, X2, X3, X4, Cin) and produces three 32-bit outputs (CARRY, SUM, Cout).

OpenLane Synthesis

- The synthesized netlist confirmed that adders and multiplexers were used.
- Gate-level representation was examined.

Dot Schematics:



Behavioral Simulation

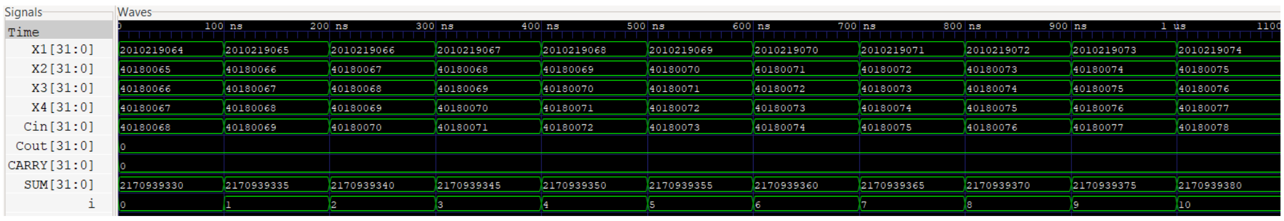
- A testbench was created.
- Inputs were generated using:

For $0 \leq i \leq 10$, $1 \leq j \leq 4$, $x_j = ((\text{Student ID number} + i + j) \bmod 2^{32})_2$

For $0 \leq i \leq 10$, $c_{in} = ((\text{Student ID number} + i + 5) \bmod 2^{32})_2$

- Waveforms verified that the compressor worked as expected.

Behaveriol Design Results



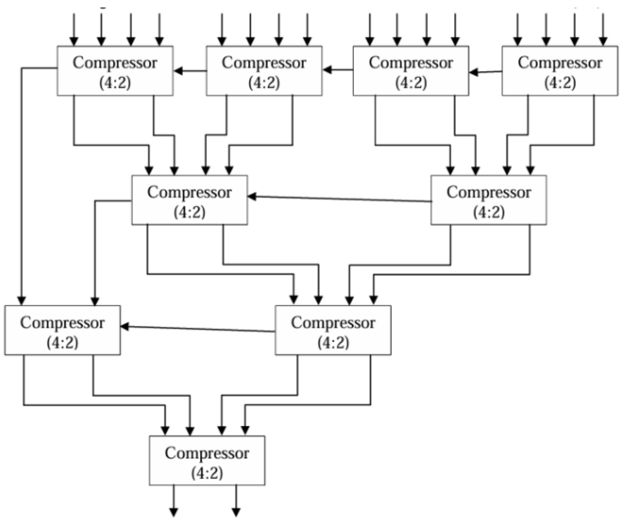
```
Signals         Waves
Time            0           100 ns      200 ns      300 ns      400 ns      500 ns      600 ns      700 ns      800 ns      900 ns      1 us        1100
  X1[31:0]      2010219064  2010219065  2010219066  2010219067  2010219068  2010219069  2010219070  2010219071  2010219072  2010219073  2010219074
  X2[31:0]      40180065    40180066    40180067    40180068    40180069    40180070    40180071    40180072    40180073    40180074    40180075
  X3[31:0]      40180066    40180067    40180068    40180069    40180070    40180071    40180072    40180073    40180074    40180075    40180076
  X4[31:0]      40180067    40180068    40180069    40180070    40180071    40180072    40180073    40180074    40180075    40180076    40180077
 Cin[31:0]      40180068    40180069    40180070    40180071    40180072    40180073    40180074    40180075    40180076    40180077    40180078
Cout[31:0]      0
CARRY[31:0]     0
 SUM[31:0]      2170939330  2170939335  2170939340  2170939345  2170939350  2170939355  2170939360  2170939365  2170939370  2170939375  2170939380
         i      0           1           2           3           4           5           6           7           8           9           10
```

3) Partial Product Accumulator (PPA)

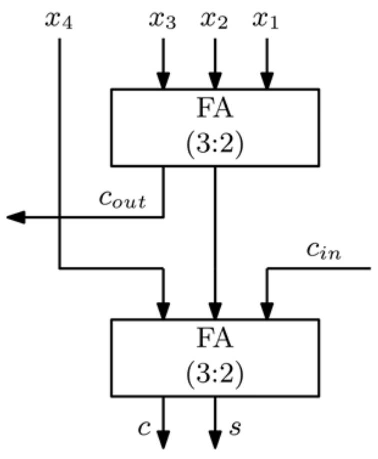
Verilog Code Implementation

The PPA.v module accumulates the partial products using Compressor_42.v as a submodule.

The structure of the requested PPA modüle:



Wallece Tree Structure

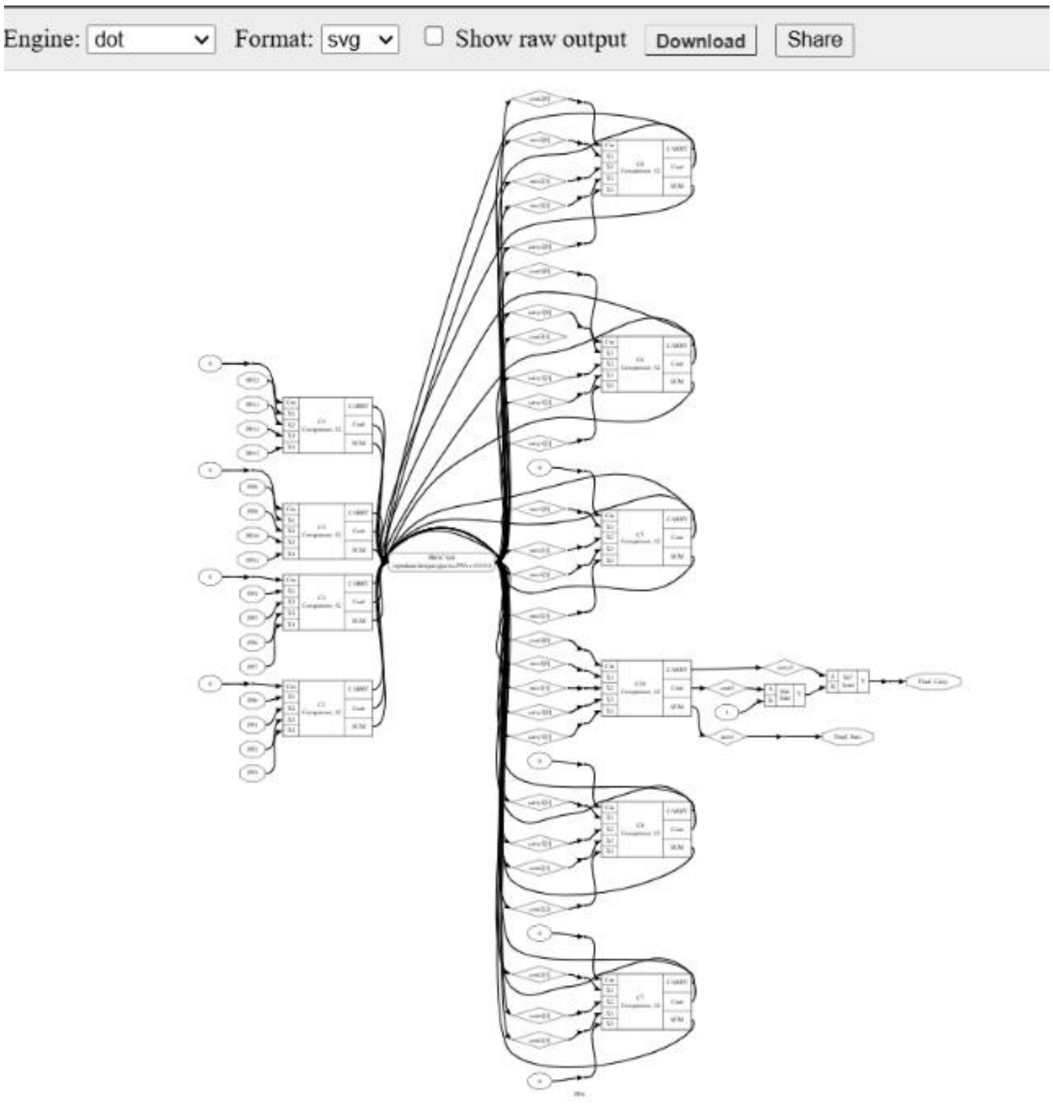


A basic 4:2 compressor circuit

OpenLane Synthesis

- The synthesized circuit included compressors and buffers.
- The hierarchy dot schematic was examined.

Dot Schematics:



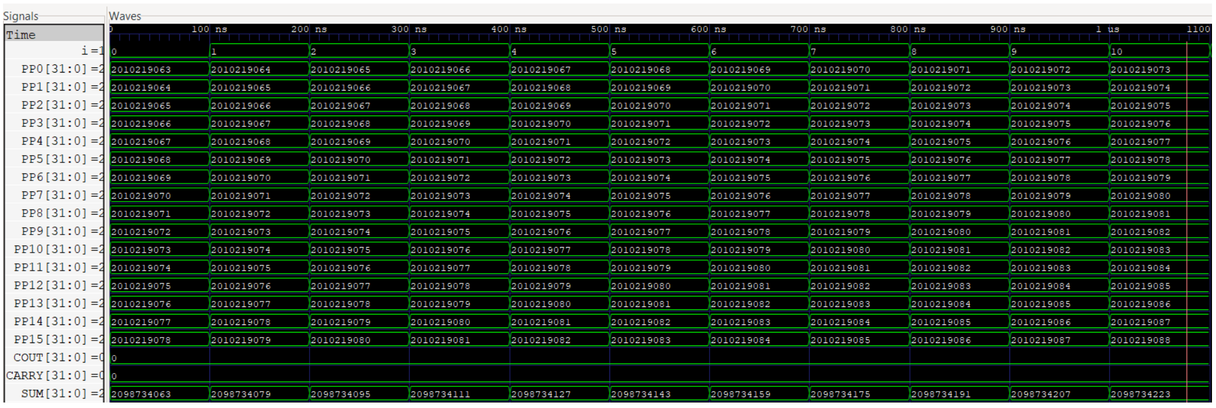
Behavioral Simulation

- A testbench was created.
- Inputs were generated using:

For $0 \le i \le 10$, $0 \le j \le 15$, $PP_j=((\text{Student ID number}+i+j) \bmod 2^{32})_2$

- The simulation confirmed that the partial product accumulation was correct.

Behavioral Simulation Results:



```
Signals          Waves
Time             0          100 ns     200 ns     300 ns     400 ns     500 ns     600 ns     700 ns     800 ns     900 ns     1 us       1100
i =1             0          1          2          3          4          5          6          7          8          9          10
PP0[31:0] =2     2010219063 2010219064 2010219065 2010219066 2010219067 2010219068 2010219069 2010219070 2010219071 2010219072 2010219073
PP1[31:0] =2     2010219064 2010219065 2010219066 2010219067 2010219068 2010219069 2010219070 2010219071 2010219072 2010219073 2010219074
PP2[31:0] =2     2010219065 2010219066 2010219067 2010219068 2010219069 2010219070 2010219071 2010219072 2010219073 2010219074 2010219075
PP3[31:0] =2     2010219066 2010219067 2010219068 2010219069 2010219070 2010219071 2010219072 2010219073 2010219074 2010219075 2010219076
PP4[31:0] =2     2010219067 2010219068 2010219069 2010219070 2010219071 2010219072 2010219073 2010219074 2010219075 2010219076 2010219077
PP5[31:0] =2     2010219068 2010219069 2010219070 2010219071 2010219072 2010219073 2010219074 2010219075 2010219076 2010219077 2010219078
PP6[31:0] =2     2010219069 2010219070 2010219071 2010219072 2010219073 2010219074 2010219075 2010219076 2010219077 2010219078 2010219079
PP7[31:0] =2     2010219070 2010219071 2010219072 2010219073 2010219074 2010219075 2010219076 2010219077 2010219078 2010219079 2010219080
PP8[31:0] =2     2010219071 2010219072 2010219073 2010219074 2010219075 2010219076 2010219077 2010219078 2010219079 2010219080 2010219081
PP9[31:0] =2     2010219072 2010219073 2010219074 2010219075 2010219076 2010219077 2010219078 2010219079 2010219080 2010219081 2010219082
PP10[31:0] =2    2010219073 2010219074 2010219075 2010219076 2010219077 2010219078 2010219079 2010219080 2010219081 2010219082 2010219083
PP11[31:0] =2    2010219074 2010219075 2010219076 2010219077 2010219078 2010219079 2010219080 2010219081 2010219082 2010219083 2010219084
PP12[31:0] =2    2010219075 2010219076 2010219077 2010219078 2010219079 2010219080 2010219081 2010219082 2010219083 2010219084 2010219085
PP13[31:0] =2    2010219076 2010219077 2010219078 2010219079 2010219080 2010219081 2010219082 2010219083 2010219084 2010219085 2010219086
PP14[31:0] =2    2010219077 2010219078 2010219079 2010219080 2010219081 2010219082 2010219083 2010219084 2010219085 2010219086 2010219087
PP15[31:0] =2    2010219078 2010219079 2010219080 2010219081 2010219082 2010219083 2010219084 2010219085 2010219086 2010219087 2010219088
COUT[31:0] =0    0
CARRY[31:0] =0   0
SUM[31:0] =2     2098734063 2098734079 2098734095 2098734111 2098734127 2098734143 2098734159 2098734175 2098734191 2098734207 2098734223
```

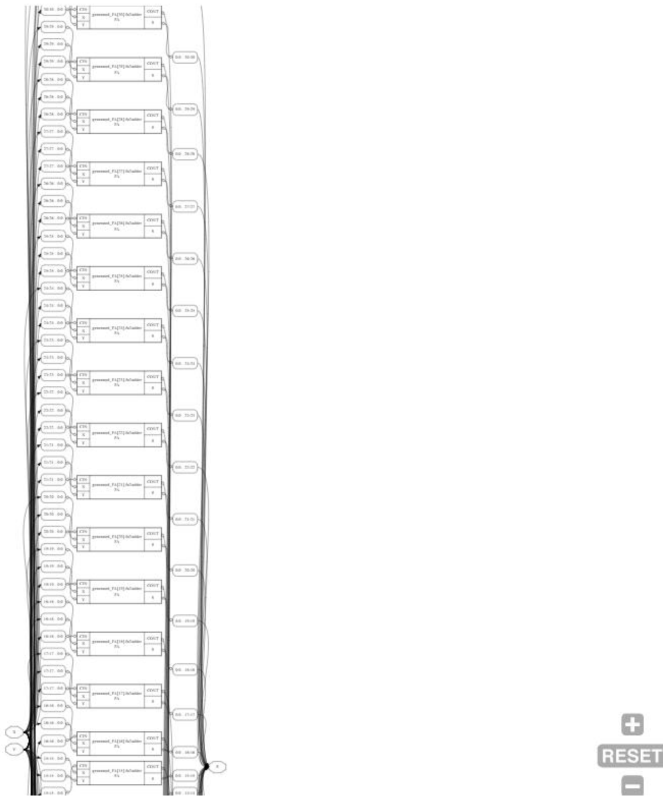
4) Ripple Carry Adder (RCA)

Verilog Code Implementation

The RCA.v module implements an n-bit ripple carry adder for the final stage.

OpenLane Synthesis

- The post-technology schematic showed the use of full adders and buffers.



Behavioral Simulation

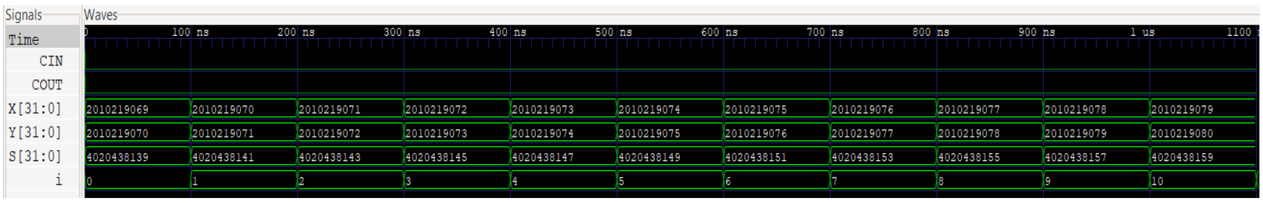
- A testbench was created.
- Inputs were generated using:

For $0 \le i \le 10$, $x=((\text{Student ID number}+i+6) \bmod 2^{32})_2$

For $0 \le i \le 10$, $y=((\text{Student ID number}+i+7) \bmod 2^{32})_2$

- The simulation confirmed the correctness of the addition.

Behavioral Simulation Results:



```
Signals   Waves
Time      0          100 ns     200 ns     300 ns     400 ns     500 ns     600 ns     700 ns     800 ns     900 ns     1 us       1100
CIN
COUT
X[31:0]   2010219069 2010219070 2010219071 2010219072 2010219073 2010219074 2010219075 2010219076 2010219077 2010219078 2010219079
Y[31:0]   2010219070 2010219071 2010219072 2010219073 2010219074 2010219075 2010219076 2010219077 2010219078 2010219079 2010219080
S[31:0]   4020438139 4020438141 4020438143 4020438145 4020438147 4020438149 4020438151 4020438153 4020438155 4020438157 4020438159
i         0          1          2          3          4          5          6          7          8          9          10
```

5) Multiplier (MUL)

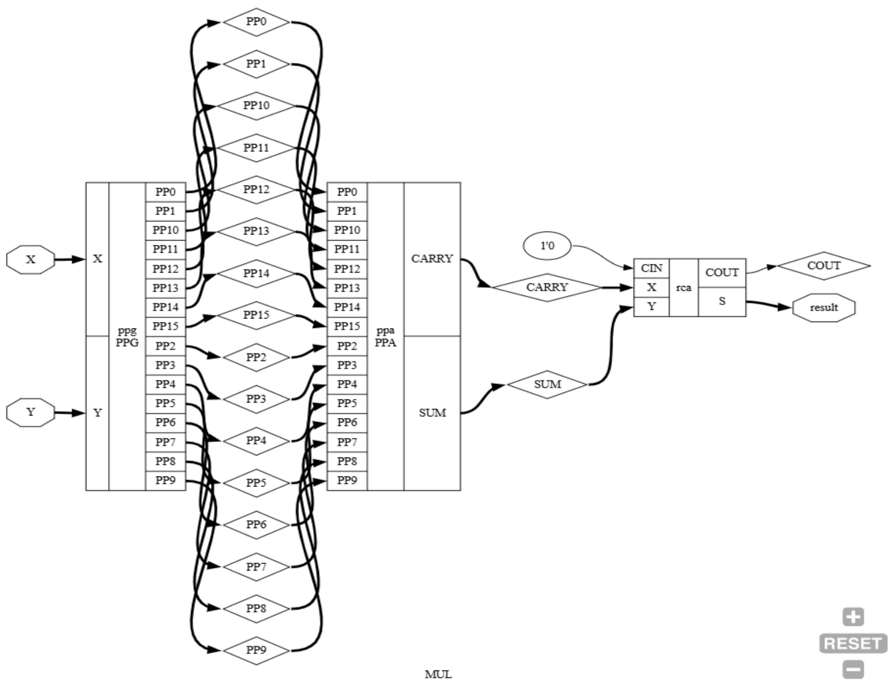
Verilog Code Implementation

The MUL.v module integrates PPG.v, PPA.v, and RCA.v.

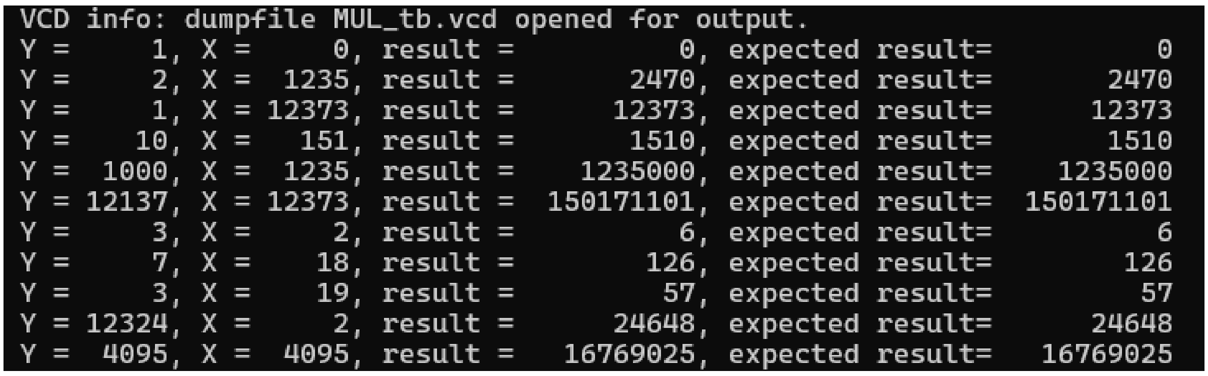
OpenLane Synthesis

- The final synthesized design showed a hierarchical structure.
- The netlist confirmed the correct instantiation of submodules.

Dot Schematics:



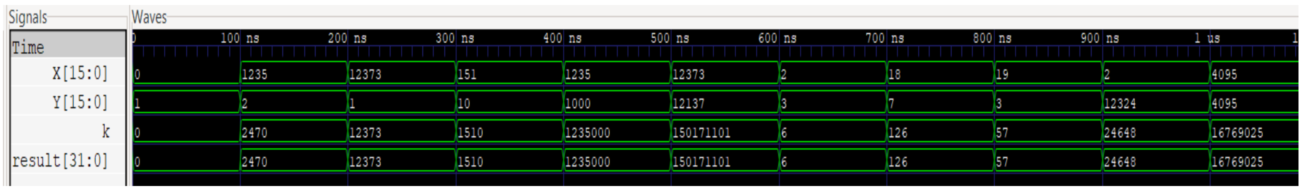
The output of the module in the console:



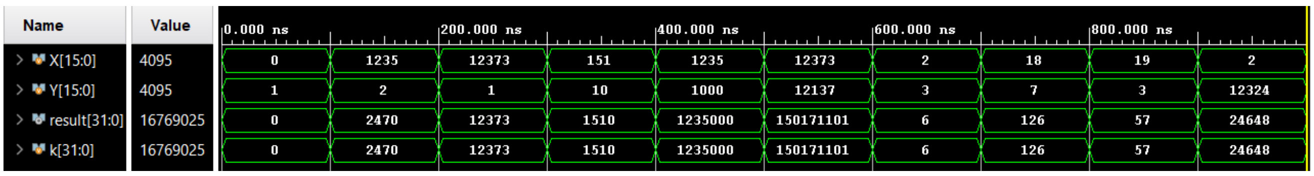
```
VCD info: dumpfile MUL_tb.vcd opened for output.
Y =     1, X =     0, result =          0, expected result=          0
Y =     2, X =  1235, result =       2470, expected result=       2470
Y =     1, X = 12373, result =      12373, expected result=      12373
Y =    10, X =   151, result =       1510, expected result=       1510
Y =  1000, X =  1235, result =    1235000, expected result=    1235000
Y = 12137, X = 12373, result =  150171101, expected result=  150171101
Y =     3, X =     2, result =          6, expected result=          6
Y =     7, X =    18, result =        126, expected result=        126
Y =     3, X =    19, result =         57, expected result=         57
Y = 12324, X =     2, result =      24648, expected result=      24648
Y =  4095, X =  4095, result =   16769025, expected result=   16769025
```

Simulation Results:

- **iVerilog**



- **Vivado**



Results and Analysis

- **PPG:** Successfully generated 16 partial products.
- **Compressor 4:2:** Efficiently reduced partial products in Wallace Tree.
- **PPA:** Accumulated partial products correctly.
- **RCA:** Final addition stage produced correct results.
- **Multiplier (MUL):** Full 16-bit multiplication was successfully implemented and verified.

Conclusion

This report presented the design and verification of a 16-bit multiplier using a structured hierarchy of PPG, PPA, and RCA. The Wallace Tree structure was effectively used to speed up accumulation. The design was synthesized in OpenLane and verified using Icarus Verilog. The simulation results confirmed the correctness of the implementation.

References

1. Hardware algorithms for arithmetic modules, http://www.aoki.ecei.tohoku.ac.jp/arith/mg/algorithm.html#ppa
2. Shirshendu Roy, “Alternative Techniques for Partial Product Accumulation”, https://digitalsystemdesign.in/alternative-techniques-for-partial-product-accumulation/