EE5331: DSP Architecture and Embedded Systems Project

High-Performance Approximate Multipliers for FPGA-Based Hardware Accelerators

by
Kumar Rohan : EE22S003
Ganga Sagar Tripathi : EE23M087

Department of Electrical Engineering

April 24, 2024

PROBLEM STATEMENT

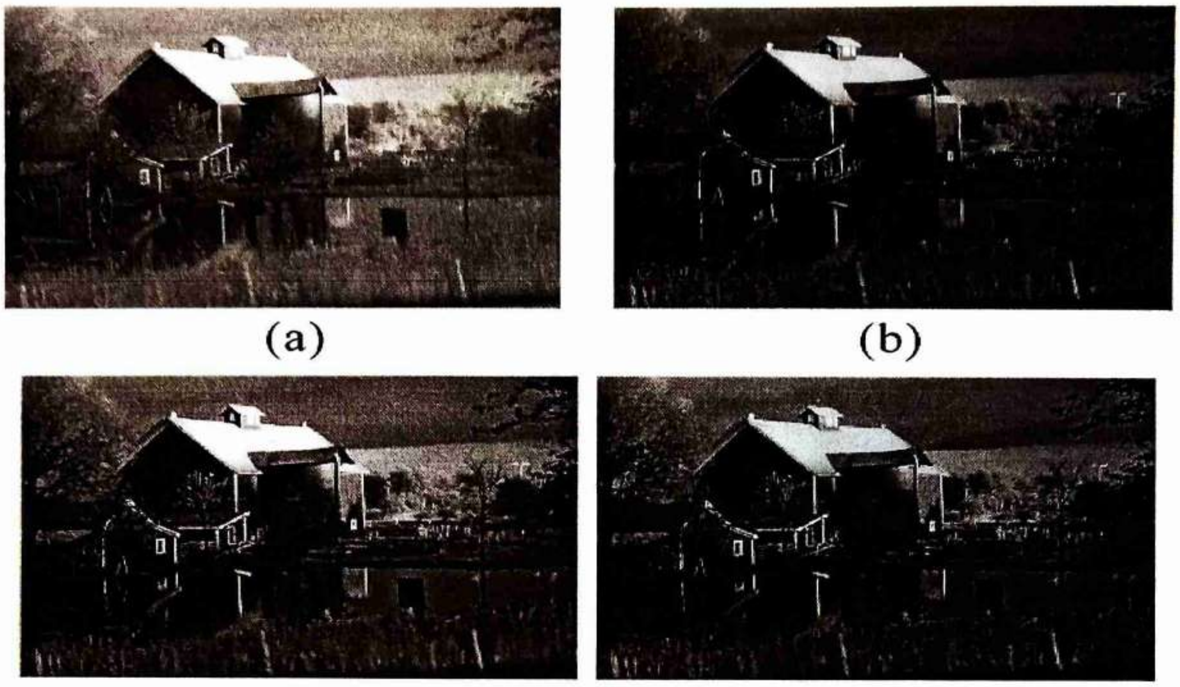
To design high performance, area optimized, low-latency approximate softcore multiplier by exploiting the underlying architectural features of FPGA.

Motivation

* A wide range of applications in image processing and Machine learning do not require accurate intermediate computations. Hence, their operation can be improved further by using approximate multipliers.-
* FPGA veddors provide high performance multipliers in the form of DSP blocks but these multipliers may prove inefficient for smaller bit-width multiplications. However, these soft multipliers still need better designs for high-performance and resource efficiency.
* Most of the accurate and approximate multipliers. architectu consider only ASIC based systems. Thus a full control over resource utilization is possible.
* For FPGA based hardware accelerators, therefore, we should do LUT based optimation for significant performance gains.

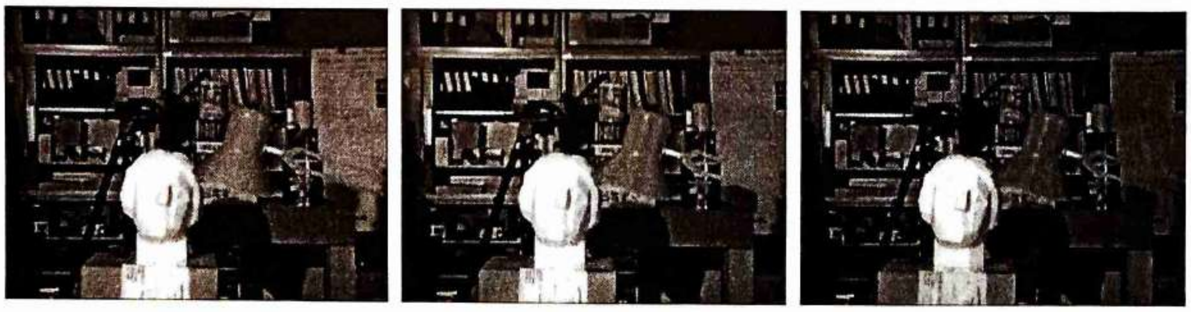
IMAGE BLENDING APPLICATION (MULTIPLY MODE)

(a) Original Image (b) Accurate



(c) Ca (d) Cc

ACCURATE AND APPROXIMATE MULTIPLIERS-BASED SUSAN IMAGE SMOOTHING ACCELERATOR OUTPUT

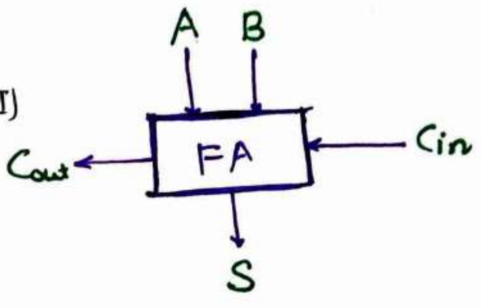


BACKGROUND

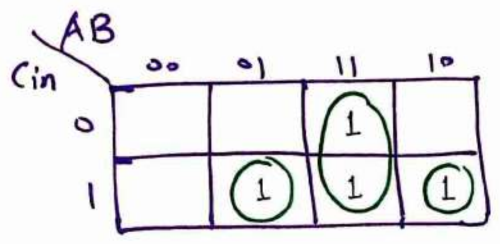
We are familier with the Full Adder Architecture. i.e.

Sum $S = A \oplus B \oplus C_{in}$ —— (I)

carry $C_{out} = AB + BC_{in} + C_{in}A$. —— (II)



Let's modify the Cout:



after rearranging the product terms in K-MAP:

$$C_{out} = AB + \bar{A}BC_{in} + A\bar{B}C_{in}$$

$$\Rightarrow \quad = AB + (\bar{A}B + A\bar{B}) \cdot C_{in}$$

$$= AB(AB + \bar{A}\bar{B}) + (\bar{A}B + A\bar{B}) \cdot C_{in}$$

$$= AB(\overline{A \oplus B}) + (A \oplus B) \cdot C_{in} \quad \text{—— (III)}$$

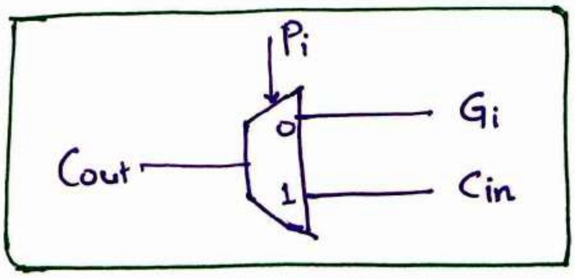
Say, $A_i \cdot B_i = G_i$ (Generate bit)

and $A_i \oplus B_i = P_i$ (Propage signal).

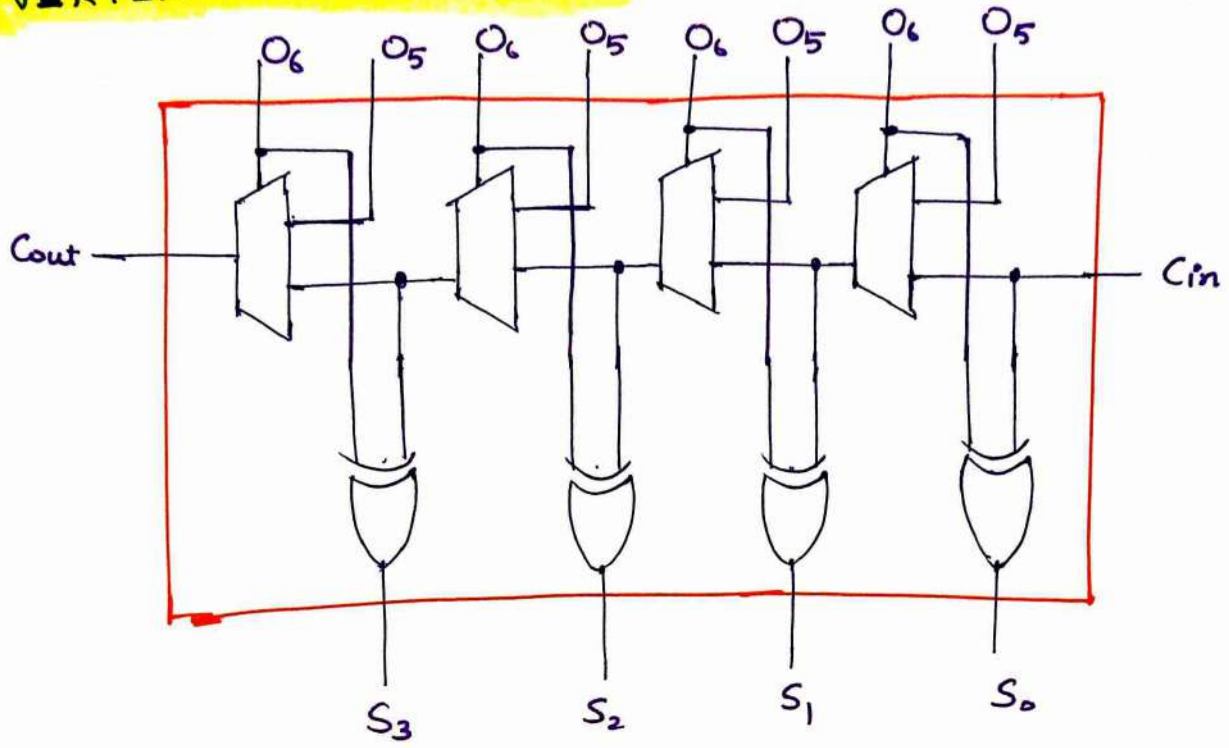
Hence, Sum and Carry be written as:

$$S = P_i \oplus C_{in}$$

$$C_{out} = G_i\overline{P_i} + C_{in}P_i$$ (This essentially implements a MUX)



VIRTEX-7 CARRY CHAIN



- Xilinx and Intel, use 6-input LUTs to implement combinational and sequential circuits.

★ A slice in the configurable logic block of xilinx's 7-series FPGAs have four 6 inputs LUTs and a single 4-bit long carry chain.

APPROXIMATE MULTIPLIERS ARCHITECTURE

1. APPROXIMATE DESIGN OF 4x2 MULTIPLIER

An accurate 4x2 multiplier generates a 6-bit output with the following optimized logic equations for:

Multiplicand A	:	$A_3\ A_2\ A_1\ A_0$
multiplier B	:	$B_1\ B_0$
Product P	:	$P_5\ P_4\ P_3\ P_2\ P_1\ P_0$

$$\begin{array}{ccccccl}
 & & A_3 & A_2 & A_1 & A_0 & \\
 & & & & B_1 & B_0 & \\
\hline
 & & A_3B_0 & A_2B_0 & A_1B_0 & A_0B_0 & \rightarrow \text{Partial Product 0 } (PP_0) \\
 & A_3B_1 & A_2B_1 & A_1B_1 & A_0B_1 & \downarrow & \rightarrow \text{Partial Product 1 } (PP_1) \\
\hline
P_5 & P_4 & P_3 & P_2 & P_1 & P_0 & \\
\hline
\end{array}$$

$P_0 = A_0B_0$;

$$P_1 = A_1B_0 \oplus A_0B_1 = A_1B_0 \cdot \overline{A_0B_1} + \overline{A_1B_0}\, A_0B_1$$

$$= A_1B_0(\bar{A_0} + \bar{B_1}) + (\bar{A_1} + \bar{B_0}) A_0B_1$$

$$= A_1B_0\bar{A_0} + A_1B_0\bar{B_1} + \bar{A_1}A_0B_1 + \bar{B_0}A_0B_1$$

Similarly, other optimized logic equations for product P[5:0] are as following:

$$P_0 = B_0A_0$$

$$P_1 = \bar{B_1}B_0A_1 + B_1\bar{B_0}A_0 + B_1\bar{A_1}A_0 + B_0A_1\bar{A_0}$$

$$P_2 = \bar{B_1}B_0A_2 + B_1\bar{B_0}A_1 + B_0A_2\bar{A_1} + B_1\bar{A_2}A_1\bar{A_0} + B_1A_2A_1A_0$$

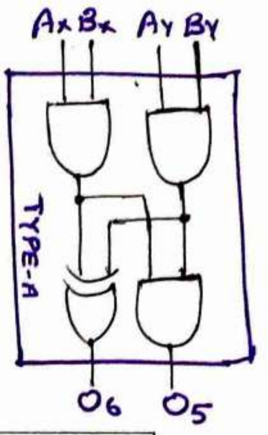
$$P_3 = \bar{B_1}B_0A_3 + B_1\bar{B_0}A_2 + B_1\bar{A_3}A_2\bar{A_1} + B_0A_3\bar{A_2}\bar{A_1} + B_1B_0\bar{A_3}\bar{A_2}A_1A_0 + B_0A_3A_2A_1 + B_0A_3A_1\bar{A_0}$$

$$P_4 = B_1\bar{B_0}A_3 + B_1A_3\bar{A_2}\bar{A_1} + B_1A_3\bar{A_2}\bar{A_0} + B_1B_0\bar{A_3}A_2A_1$$

$$P_5 = B_1B_0A_3A_2 + B_1B_0A_3A_1A_0$$

P_0, P_1 and P_2 depends on less than six variables. Hence any two of these three can be generated using a single LUT 6_2 by accomodating all six product bits in a single slice (of four LUTs), we can optimize on area and energy efficiency. For this, P_0 is neglected. P_1 and P_2 are generated using 1 LUT and remaing use 1 each. Truncation of P_0 limits the error to LSB with output accuracy of 75% and max. error magnitude = 1.

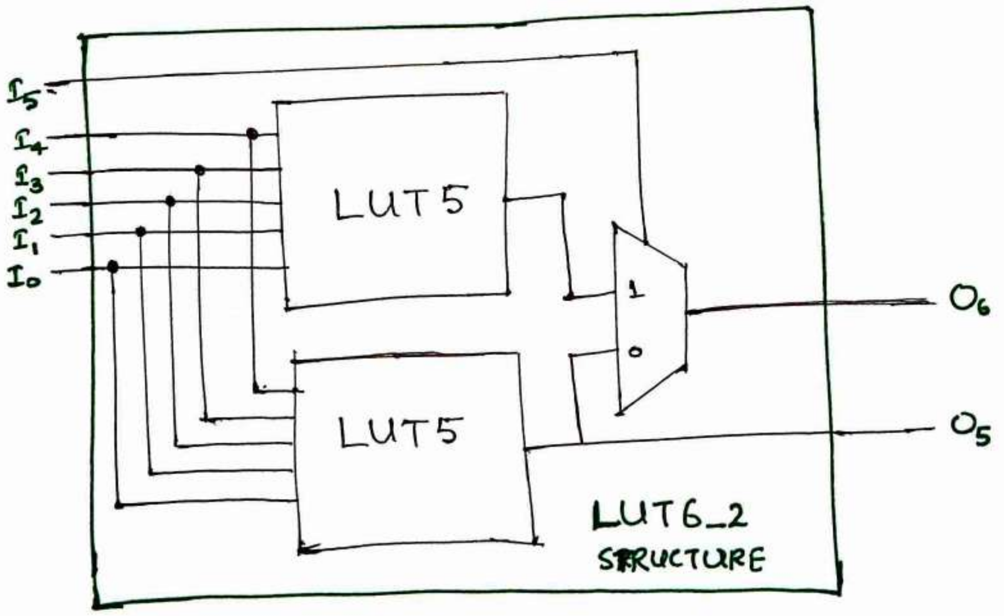
LUT CONFIGURATION



The table defines generation (O_5) and summation (O_6) of partial product bits A_YB_Y and A_xB_x.
The value $O5 = 0\times8000$ and $O_6 = 0\times7888$.

A_Y	B_Y	A_X	B_X	A_XB_X	A_YB_Y	$A_XB_X + A_YB_Y$		O6 (Hex)	O5 (Hex)
						Sum (O6)	Carry (O5)		
0	0	0	0	0	0	0	0	8	0
0	0	0	1	0	0	0	0		
0	0	1	0	0	0	0	0		
0	0	1	1	1	0	1	0		
0	1	0	0	0	0	0	0	8	0
0	1	0	1	0	0	0	0		
0	1	1	0	0	0	0	0		
0	1	1	1	1	0	1	0		
1	0	0	0	0	0	0	0	8	0
1	0	0	1	0	0	0	0		
1	0	1	0	0	0	0	0		
1	0	1	1	1	0	1	0		
1	1	0	0	0	1	1	0	7	8
1	1	0	1	0	1	1	0		
1	1	1	0	0	1	1	0		
1	1	1	1	1	1	0	1		

accomodates only four input values. The INIT value for LUT6-2 to produce $O5 = 0\times8000$ and $O6 = 0\times7888$ will be 0x7888 7888 8000 8000



2. APPROXIMATE Design of 4×4 MULTIPLIER

This is implemented by combining two 4×2 multipliers. To multiply A ($A_3A_2A_1A_0$) and B($B_3B_2B_1B_0$), the first multiplier takes A ($A_3A_2A_1A_0$) and B (B_1B_0), while the second 4×2 multiplier takes A($A_3A_2A_1A_0$) and B (B_1B_0).

The partial products obtained are shown below:

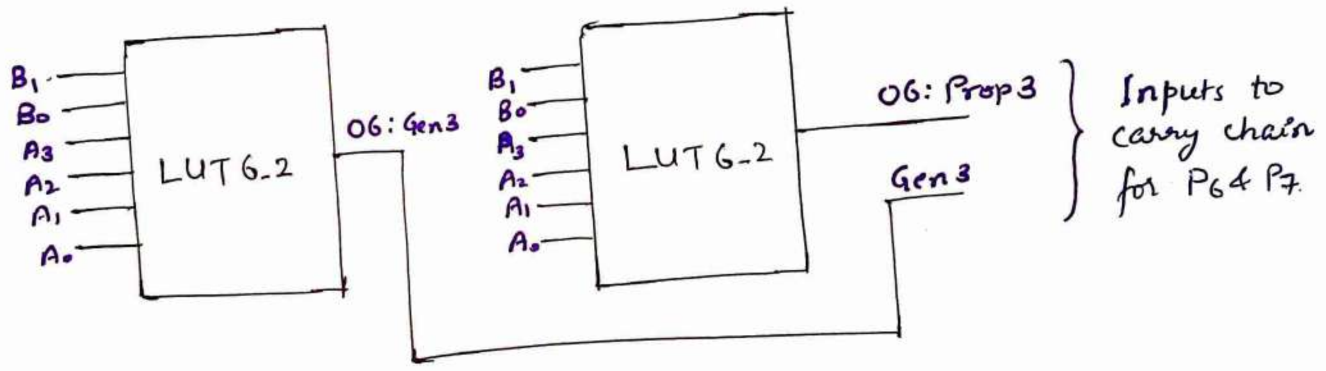
$$\begin{array}{c|cccc|ccc}
 & & PP_{05} & PP_{04} & PP_{03} & PP_{02} & PP_{01} & PP_{00} \\
PP_{15} & PP_{14} & PP_{13} & PP_{12} & PP_{11} & PP_{10} & & \\
\hline
P_7 & P_6 & P_5 & P_4 & P_3 & P_2 & P_1 & P_0
\end{array}$$

In order to minimize the no. of LUTs used, we implicitly generate PP_{15} & PP_{14} to calculate P_7 and P_6, with P_6 = sum and P_7 = carry_out of the carry chain. This requires two LUTs, one for generate and one for propagate. P_3, P_4 and P_5 are computed using a single carry chain whose Cout is then used in the computation of P_6 & P_7. This requires 3 LUTs. Calculating P_0 & P_2 requires one LUT. Two more LUTs are required to compute PP_{11}, PP_{12} and PP_{13}. Thus, only a total of 12 LUTs are required to do an approximate 4×4 multiplication whereas it was 16 LUTs before optimation.

Not only does this give area gains but also provides significant reduction in total number of error cases by having only six erroneous outputs out of all the combinations of 4x4 multiplication.

Multiplier	Multiplicand	Accurate Product	Approximate Result	Difference
5	15	75	67	8
6	7	42	34	8
7	15	105	97	8
6	15	90	82	8
13	13	169	161	8
15	5	75	67	8

P6 and P7 are obtained using Gen3 and Prop3 signals. which are generated from input bits as shown below:



Examples:

(i)
```
        0101   :5
        1111   :15
   ----------------
        1111
       1111
   ----------------
    1000011    :67
```

(ii)
```
        0110   :6
        0111   :7
   ----------------
       10010
      0110
   ----------------
      101010 : 42
```

* In (i) example, ∵ Carry is generate from P_2 stage which is discarded and hence produces error of 8 (difference).
* In (ii) example: ∵ No carry is generated at P_2 hence No error.

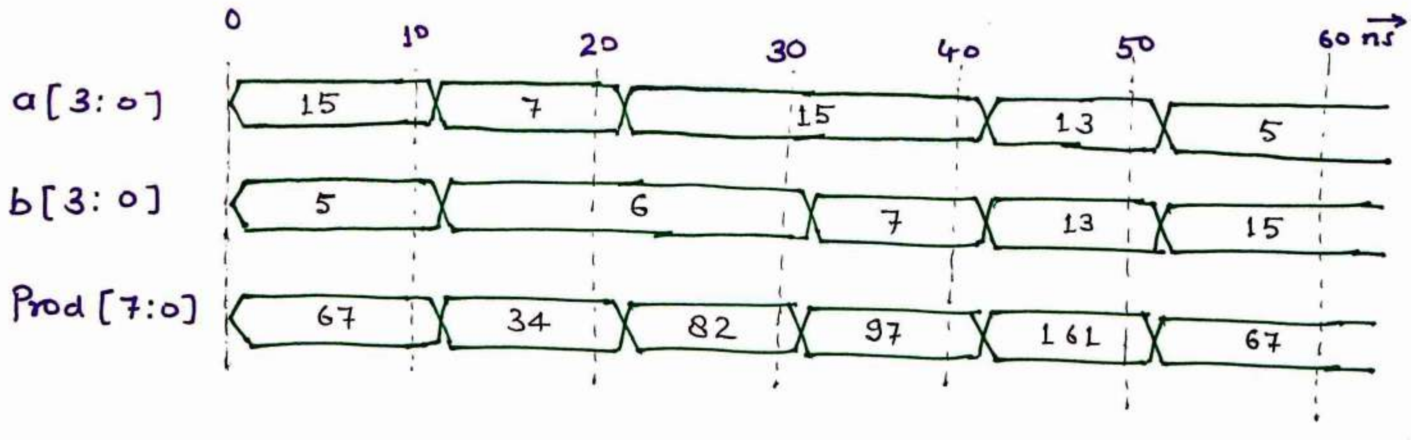
LUTS' INPUTS AND OUTPUTS PINS CONFIGURATION FOR APPROXIMATE 4 × 4 MULTIPLIER

LUT	LUT Input Pins Configuration						INIT value (Hex)	LUT Output Pins Configuration	
	I5	I4	I3	I2	I1	I0		O6	O5
LUT_0	1	B_1	B_0	A_2	A_1	A_0	*B4CCF00066AACC00*	PP_0<2>	PP_0<1> = P_1
LUT_1	B_1	B_0	A_3	A_2	A_1	A_0	*C738F0F0FF000000*	PP_0<3>	
LUT_2	B_1	B_0	A_3	A_2	A_1	A_0	*07C0FF0000000000*	PP_0<4>	
LUT_3	B_1	B_0	A_3	A_2	A_1	A_0	*F800000000000000*	PP_0<5>	
LUT_4	1	B_3	B_2	A_2	A_1	A_0	*B4CCF00066AACC00*	PP_1<2>	PP_1<1>
LUT_5	B_3	B_2	A_3	A_2	A_1	A_0	*C738F0F0FF000000*	PP_1<3>	
LUT_6	B_3	B_2	A_3	A_2	A_1	A_0	*F800000000000000*	Gen_3	
LUT_7	1	1	PP_0<2>	B_2	B_0	A_0	*5FA05FA088888888*	P_2	P_0
LUT_8	1	PP_1<1>	PP_0<3>	B_2	A_0	PP_0<2>	*007F7F80FF808000*	$Prop_0$	Gen_0
LUT_9	1	1	1	1	PP_1<2>	PP_0<4>	*6666666688888880*	$Prop_1$	Gen_1
LUT_{10}	1	1	1	1	PP_1<3>	PP_0<5>	*6666666688888880*	$Prop_2$	Gen_2
LUT_{11}	B_3	B_2	A_3	A_2	A_1	A_0	*07C0FF0000000000*	$Prop_3$	

Higher order multipliers can be produced by combining more than one 4×4 multipliers. This modular approach provides a broader design space by using various accurate / approximate submultipliers.

RESULTS

The diagram below shows different combinations of 4×4 approximate results and their outputs.



References

[1] *7 Series DSP48E1 Slice*, Xilinx, San Jose, CA, USA, 2018. [Online]. Available: https://www.xilinx.com/support/documentation/user_guides/ug479_7Series_DSP48E1.pdf

[2] *LogiCORE IP V12.0*, Xilinx, San Jose, CA, USA, 2015. [Online]. Available: https://www.xilinx.com/support/documentation/ip_documentation/mult_gen/v12_0/pg108-mult-gen.pdf

[3] (2020). *Integer Arithmetic IP Cores User Guide*. [Online]. Available: https://www.altera.com/en_US/pdfs/literature/ug/ug_lpm_alt_mfug.pdf

[4] N. Brunie, F. de Dinechin, M. Istoan, G. Sergent, K. Illyes, and B. Popa, "Arithmetic core generation using bit heaps," in *Proc. 23rd Int. Conf. Field Program. Logic Appl.*, Porto, Portugal, 2013, pp. 1–8.

[5] J.-L. Beuchat and J.-M. Muller, "Automatic generation of modular multipliers for FPGA applications," *IEEE Trans. Comput.*, vol. 57, no. 12, pp. 1600–1613, Dec. 2008.

[6] A. Kakacak, A. E. Guzel, O. Cihangir, S. Gören, and H. F. Ugurdag, "Fast multiplier generator for FPGAs with LUT based partial product generation and column/row compression," *Integration*, vol. 57, pp. 147–157, Mar. 2017.

[7] M. Kumm, J. Kappauf, M. Istoan, and P. Zipf, "Resource optimal design of large multipliers for FPGAs," in *Proc. IEEE 24th Symp. Comput. Arithmetic (ARITH)*, London, U.K., 2017, pp. 131–138.

[8] E. G. Walters, "Array multipliers for high throughput in Xilinx FPGAs with 6-input LUTs," *Computers*, vol. 5, no. 4, p. 20, 2016.

[9] M. Kumm, S. Abbas and P. Zipf, "An efficient softcore multiplier architecture for Xilinx FPGAs," in *Proc. IEEE 22nd Symp. Comput. Arithmetic*, Lyon, France, 2015, pp. 18–25.