

Exploring realizations of large integer multipliers using embedded blocks in modern FPGAs



Shreesha Srinath

- **Motivation**
 - Problem proposal
 - Related work
- **Design description**
 - High level design description
- **Example and results**
 - Performance comparison
- **Conclusions**



The Problem:

- HPCWire Article “*A paper-and-pencil analysis of FPGA peak floating-point performance*”
- Modern FPGAs provide the designer with hardwired multiplier blocks.
- How to build large integer multiplier using these embedded blocks?

Solution:

- Provide a systematic approach for implementation of multiplication.
- Design equations which can be converted into constraint graphs which the designer can analyze given the resources.

- Optimised realisations of large integer multipliers and squarers using embedded blocks – Gao et. al
 - Systematic approach for “symmetric blocks”
- Large multipliers with less DSP blocks – LIP research report.
 - “Karatsuba-Ofman” Algorithm
 - Non-Standard Multiplier Tiling
 - *Limited to double-precision floating point multiplication*
- The Modern FPGAs provide “**asymmetric multiplier**” blocks
- ***Solution addresses these resources in a general fashion***



- Assume two numbers X & Y which are k-bit wide
- Assume a basic asymmetric block of size m x n bits with m<n;
- We need to compute the multiplication

$$Z = X*Y$$

- Step 1: Split the numbers into m and n sized chunks



Step 1 : Obtaining the chunks

- Split X into m -sized chunks and Y into n -sized chunks
- Number of chunks for $X = \text{ceiling}(k/n) = p1$
- Number of chunks for $Y = \text{ceiling}(k/m) = p2$

- $X = [X_{p1-1} \ X_{p1-2} \ \dots \ X_0]_{2^n}$

- $Y = [Y_{p2-1} \ Y_{p2-2} \ \dots \ Y_0]_{2^m}$

- $X = 2^{(n*(p1-1))} X_{p1-1} + 2^{(n*(p1-2))} X_{p1-2} + \dots + X_0$

- $Y = 2^{(n*(p2-1))} Y_{p2-1} + 2^{(n*(p2-2))} Y_{p2-2} + \dots + Y_0$



Step 2 : Multiplication

- Computing the multiplication :

$$Z = X * Y$$

- Substituting the chunks we get

$$Z = ([X_{p1-1} \ X_{p1-2} \ \dots \ X_0]_{2^n}) * ([Y_{p2-1} \ Y_{p2-2} \ \dots \ Y_0]_{2^m})$$

$$Z = (2^{(n*(p1-1))} X_{p1-1} + 2^{(n*(p1-2))} X_{p1-2} + \dots + X_0) * (2^{(m*(p2-1))} Y_{p2-1} + 2^{(m*(p2-2))} Y_{p2-2} + \dots + Y_0)$$

- Each product is of length $m+n-1$ bits and in general :

$$PP_{ij} = X_{p1-i} * Y_{p2-j} = (2^{(n*(p1-i)+(m*(p2-j))}) X_{p1-i} * Y_{p2-j}$$



- The partial products obtained by the above computation are then grouped and added to obtain the result of the computation.
- How do we group them? And does this matter?
- Grouping:
 - Horizontally
 - Vertically
 - Diagonally



- Consider numbers X & Y such that,

$$X = [x_2 \ x_1 \ x_0]$$

$$Y = [y_3 \ y_2 \ y_1 \ y_0]$$

- $Z = X * Y$

$$= [x_2 \ x_1 \ x_0] * [y_3 \ y_2 \ y_1 \ y_0]$$

$$= (2^{2n}x_2 + 2^n x_1 + x_0) * (2^{3m}y_3 + 2^{2m}y_2 + 2^m y_1 + y_0)$$

$$= 2^{2n} \cdot (x_2 \cdot y_0) + 2^{2n} \cdot (x_1 \cdot y_0) + (x_0 \cdot y_0)$$

$$+ 2^{(2n+m)} \cdot (x_2 \cdot y_1) + 2^{(n+m)} \cdot (x_1 \cdot y_1) + 2^{(m)} \cdot (x_0 \cdot y_0)$$

$$+ 2^{(2n+2m)} \cdot (x_2 \cdot y_2) + 2^{(n+2m)} \cdot (x_1 \cdot y_2) + 2^{(2m)} \cdot (x_0 \cdot y_2)$$

$$+ 2^{(2n+3m)} \cdot (x_2 \cdot y_3) + 2^{(n+3m)} \cdot (x_1 \cdot y_3) + 2^{(3m)} \cdot (x_0 \cdot y_3)$$

$$= a_1 + a_2 + a_3$$

$$+ b_1 + b_2 + b_3$$

$$+ c_1 + c_2 + c_3$$

$$+ d_1 + d_2 + d_3$$

- Partial Products Obtained are:

$$\text{PP1: } (a_1 + a_2 + a_3)$$

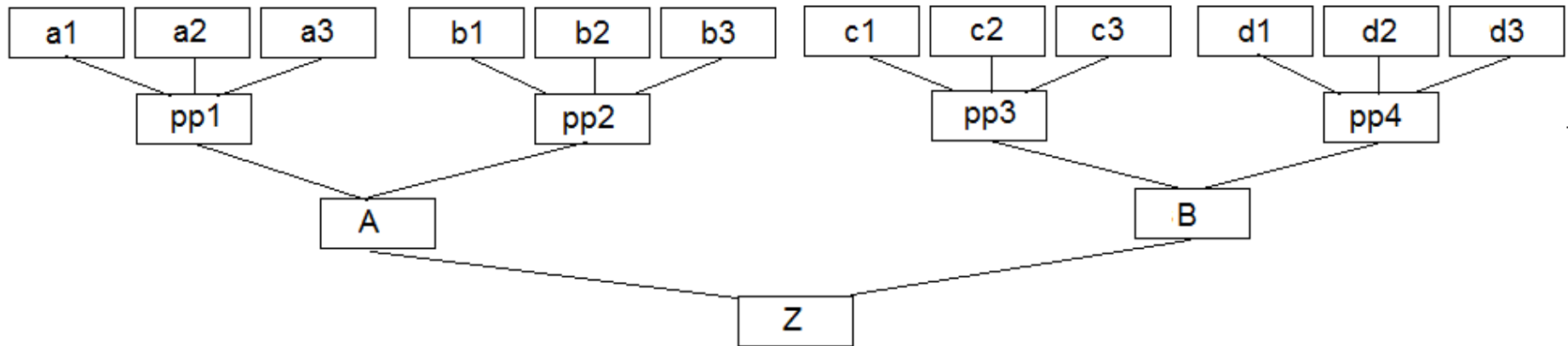
$$\text{PP2: } (b_1 + b_2 + b_3)$$

$$\text{PP3: } (c_1 + c_2 + c_3)$$

$$\text{PP4: } (d_1 + d_2 + d_3)$$

$$Z = \text{PP1} + \text{PP2} + \text{PP3} + \text{PP4}$$





- Organization for timing
- Assuming the cycles for multiplication and addition to be 1
- Cost of the implementation is
 - 12 multiplications
 - 12 additions
- Result obtained after 7 cycles

Partial Products Obtained are:

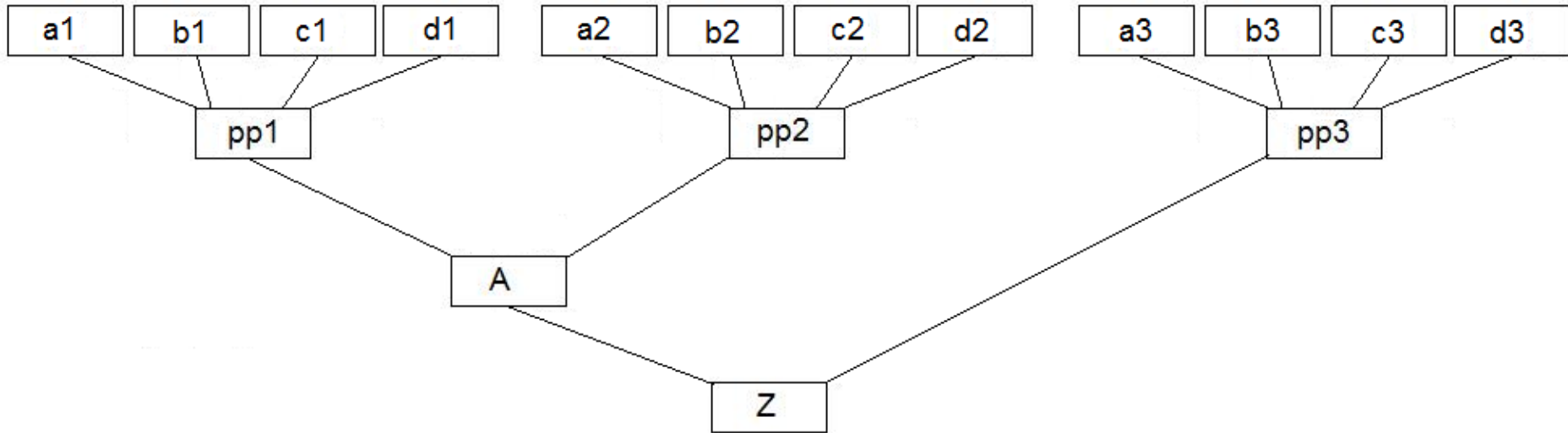
$$\text{PP1: } (a_1 + b_1 + c_1 + d_1)$$

$$\text{PP2: } (a_2 + b_2 + c_2 + d_2)$$

$$\text{PP3: } (a_3 + b_3 + c_3 + d_3)$$

$$Z = \text{PP1} + \text{PP2} + \text{PP3}$$





- Organization for timing
- Assuming the cycles for multiplication and addition to be 1
- Cost of the implementation is
12 multiplications
12 additions but larger adders!
- Result obtained after 7 cycles

Partial Products Obtained are:

PP1: (a1)

PP2: (b1 + a2)

PP3: (c1 + b2 + a3)

PP4: (d1 + c2 + b3)

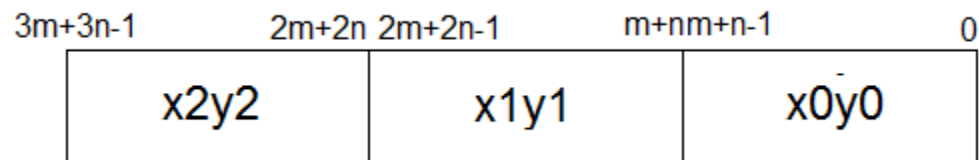
PP5: (d2 + c3)

PP6: (d3)

$$Z = PP1 + PP2 + PP3 + PP4 + PP5 + PP6$$

Grouping the products diagonally is not an **addition** it is merely a **concatenation**.

$$\begin{aligned} \text{PP3: } & (c_1 + b_2 + a_3) \\ & = x_2y_2 + x_1y_1 + x_0y_0 \end{aligned}$$



What other optimization can be done for better timing and area?

Deferred Parallel Carry addition of partial product

- To optimize the multiplier with an emphasis on area saving. The idea here is a set of carry bits generated from various levels of additions are combined and processed later.
- Keep note of the position where carry needs to be generated and insert zeros in between
- **Caveat** : If the position of carry generated by any two levels overlap then if the number is two handled by simple hardware and if more the optimization is not useful



Another advantage of going diagonally.

- **The adder size at each level increases logarithmically.**

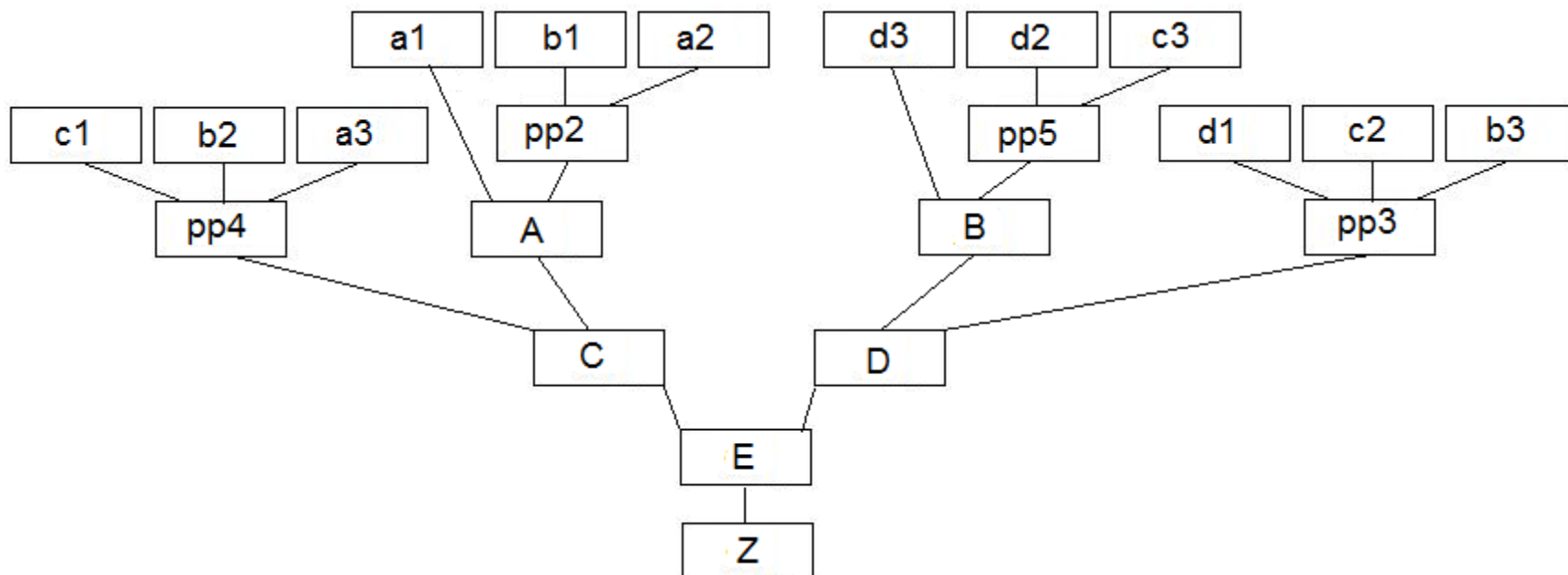
Stage 1: Largest adder – $m+n$

Stage 2: Largest adder – $2(m+n)$

Stage 3: Largest adder – $3(m+n)$

•
•
•
•
•

Stage Q: Largest adder – $Q(m+n)$



- Best Organization for **timing and area**
- Assuming the cycles for multiplication and addition to be 1
- Cost of the implementation is 12 multiplications
6 additions but lesser in area!
- Result obtained after 5 cycles

Three design implemented targeted for Xilinx Virtex-5 family of devices

The multiplier block available of size : 24 x 17

Multiplier capable of handling multiplications of size : 52-68 bits

- Baseline design with no deferred carry and horizontal grouping
- Horizontal grouping with deferred carry additions
- Diagonal grouping with deferred carry additions



Design	Number of Slice Registers	Number of LUTs	Frequency (MHz)
Baseline	498	736	147.22
Horizontal	506	601	239.2
Diagonal	517	551	254.05

Conclusions

- Contributions
 - Systematic methodology presented to build larger multipliers using **asymmetric** building blocks
 - Design equations when transformed to constraint graphs help the designer decide the best schedule
 - The design can further yield better frequencies when pipelined
 - The proposed approach provides the best design for timing and area considerations

