Area and Delay Optimized 128-Bit Multiplier using Binary to Excess-1 Converter Based Adder

Gaurav Patidar[1], Prof. Sudha Nair[2]

[1]M-Tech Research Scholar, [2]Research Guide,
RKDF Institute of Science & Technology, Bhopal

Abstract - A multiplier design is solely depends on the adder used in the design. The area, delay and power is mostly utilized by the adder design, so that the adder should be having the efficient area and delay profile to make multiplier design better. Low power consumption and smaller area are some of the most important criteria for the fabrication of DSP systems and high performance systems. Optimizing the speed and area of the multiplier is a major design issue. However, area and speed are usually conflicting constraints so that improving speed results mostly in larger areas. In our project we try to determine the best solution to this problem by comparing a few multipliers. The multiplier architecture proposed in this paper is of 128 bit using binary to excess-1 converter (BEC) based carry select adder(CSLA). The application of BEC instead of Ripple Carry Adder(RCA) is more beneficial in terms of area of the logic, lower delay and the lower power consumption of the circuit.

Keywords - Multiplier, RCA, BEC, Area, Delay.

I. INTRODUCTION

Multipliers are key components of many high performance systems such as FIR filters, microprocessors, digital signal processors, etc. A system's performance is generally determined by the performance of the multiplier because the multiplier is generally the slowest clement in the system. Furthermore, it is generally the most area consuming. Hence, optimizing the speed and area of the multiplier is a major design issue. However, area and speed are usually conflicting constraints so that improving speed results mostly in larger areas. As a result, a whole spectrum of multipliers with different area-speed constraints have been designed with fully parallel. Multipliers at one end of the spectrum and fully serial multipliers at the other end. In between are digit serial multipliers where single digits consisting of several bits are operated on. These multipliers have moderate performance in both speed and area. However, existing digit serial multipliers have been Plagued by complicated switching systems and/or irregularities in design. Radix 2^n multipliers which operate on digits in a parallel fashion instead of bits bring the pipelining to the digit level and avoid most of'the above problems. They were introduced by M. K. Ibrahim in 1993. These structures are iterative and modular. The pipelining done at the digit level brings the benefit of constant operation speed irrespective of the size of' the multiplier. The clock speed *is* only determined by the digit size which is already fixed before the design is implemented.

Carry Select Adder:

In electronics, a carry-select adder is a particular way to implement an adder, which is a logic element that computes the $(n+1)$-bit sum of two n-bit numbers. The carry-select adder is simple but rather fast, having a gate level depth of $O(\sqrt{n})$.

The carry-select adder generally consists of two ripple carry adders and a multiplexer. Adding two n-bit numbers with a carry-select adder is done with two adders (therefore two ripple carry adders) in order to perform the calculation twice, one time with the assumption of the carry being zero and the other assuming one. After the two results are calculated, the correct sum, as well as the correct carry, is then selected with the multiplexer once the correct carry is known.

The number of bits in each carry select block can be uniform, or variable. In the uniform case, the optimal delay occurs for a block size of $\lfloor\sqrt{n}\rfloor$. When variable, the block size should have a delay, from addition inputs A and B to the carry out, equal to that of the multiplexer chain leading into it, so that the carry out is calculated just in time. The $O(\sqrt{n})$ delay is derived from uniform sizing, where the ideal number of full-adder elements per block is equal to the square root of the number of bits being added, since that will yield an equal number of MUX delays.

Basic building block:

Above is the basic building block of a carry-select adder, where the block size is 4. Two 4-bit ripple carry adders are multiplexed together, where the resulting carry and sum bits are selected by the carry-in. Since one ripple carry adder assumes a carry-in of 0, and the other assumes a carry-in of 1, selecting which adder had the correct assumption via the actual carry-in yields the desired result.

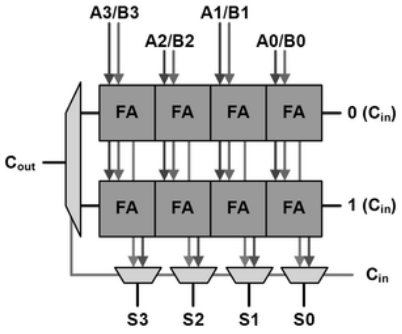

Fig. 1 Basic Building Block

II. BINARY MULTIPLIER

A Binary multiplier is an electronic hardware device used in digital electronics or a computer or other electronic device to perform rapid multiplication of two numbers in binary representation. It is built using binary adders.

The rules for binary multiplication can be stated as follows

1. If the multiplier digit is a 1, the multiplicand is simply copied down and represents the product.
2. If the multiplier digit is a 0 the product is also 0.

For designing a multiplier circuit we should have circuitry to provide or do the following three things:

1. it should be capable identifying whether a bit is 0 or 1.
2. It should be capable of shifting left partial products.
3. It should be able to add all the partial products to give the products as sum of partial products.
4. It should examine the sign bits. If they are alike, the sign of the product will be a positive, if the sign bits are opposite product will be negative. The sign bit of the product stored with above criteria should be displayed along with the product.

From the above discussion we observe that it is not necessary to wait until all the partial products have been formed before summing them. In fact the addition of partial product can be carried out as soon as the partial product is formed.

Array Multiplier :

Array multiplier is well known due to its regular structure. Multiplier circuit is based on add and shift algorithm. Each partial product is generated by the multiplication of the multiplicand with one multiplier bit. The partial product are shifted according to their bit orders and then added. The addition can be performed with normal carry propagate adder. N-1 adders are required where N is the multiplier length.

			x	A3 B3	A2 B2	A1 B1	A0 B0	Inputs
			C	B0 x A3	B0 x A2	B0 x A1	B0 x A0	
		+	B1 x A3	B1 x A2	B1 x A1	B1 x A0		
		C	sum	sum	sum	sum		
	+	B2 x A3	B2 x A2	B2 x A1	B2 x A0			Internal Signals
	C	sum	sum	sum	sum			
+	B3 x A3	B3 x A2	B3 x A1	B3 x A0				
C	sum	sum	sum	sum				
Y7	Y6	Y5	Y4	Y3	Y2	Y1	Y0	Outputs

An example of 4-bit multiplication method is shown below:

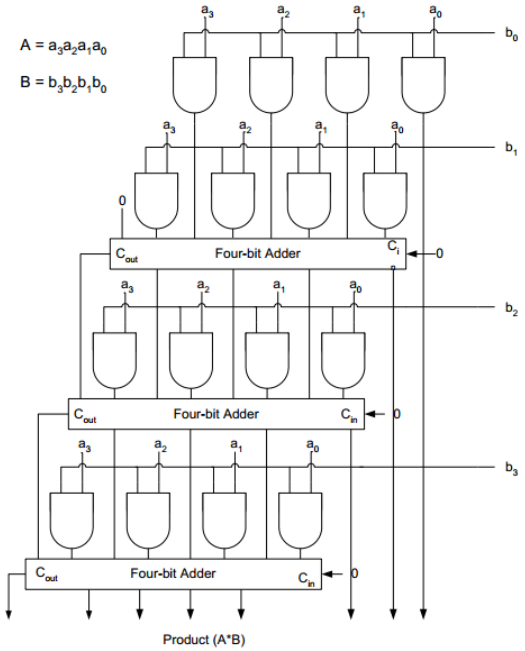

Fig. 2 Four Bit Multiplication

Multiplter For Unsigned Data:

Multiplication involves the generation of partial products, one for each digit in the multiplier, as in Figure3.These partial products are then summed to produce the final product. The multiplication of two n-bit binary integers results in a product of up to 2n bits in length [2]. We used the following algorithm to implement the multiplication operation for unsigned data

III. PROPOSED ARCHITECTURE

Multiplication Algorithm for 128 bit:

Let the product register size be 256 bits. Let the multiplicand registers size be 128 bits. Store the multiplier in the least significant half of the product register. Clear the most significant half of the product register.

Repeat the following steps for 128 times:

1. If the least significant bit of the product register is "1" then add the multiplicand to the most significant half of the product register.

2. Shift the content of the product register one bit to the right (ignore the shifted-out bit.)

3. Shift-in the carry bit into the most significant bit of the product register. Fig. 4. Shows a block diagram for such a multiplier [2].

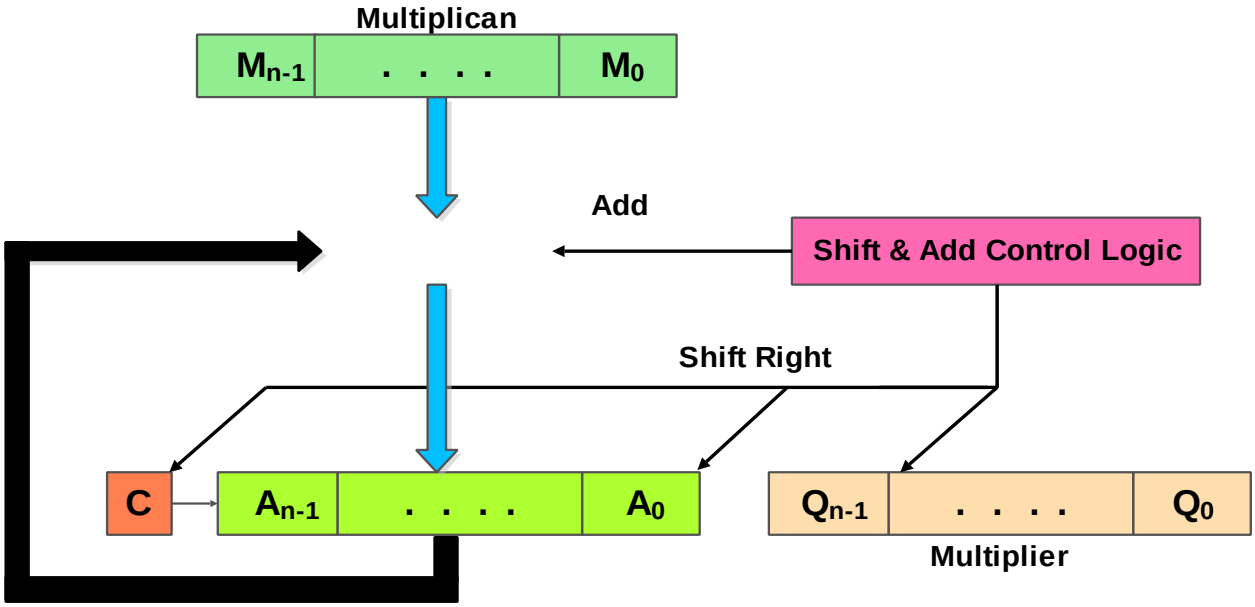

Fig. 4 Two n-Bit Multiplier Design

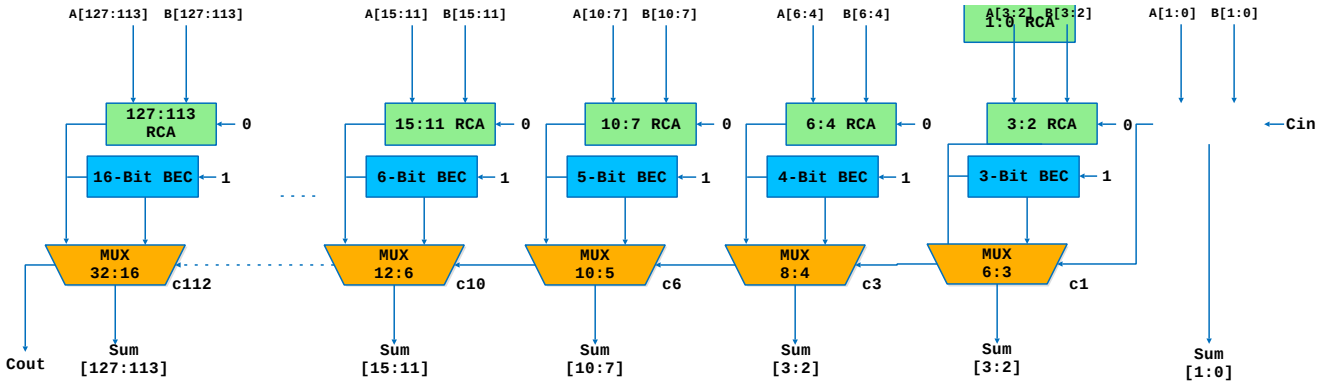

Fig. 5. 128-Bit Area, Delay Efficient BEC Based Adder

IV. SIMULATION RESULTS

The proposed 128-Bit BEC based multiplier architecture is shown in the previous section has better delay profile and the area required. The calculation of the delay and area is shown here. The proposed 128-Bit multiplier design is implemented on FPGA Virtex 7 board. The device utilization summary is given in Table I.

Table I: Device Utilization Summary of Estimated Values

Device Utilization Summary (estimated values)			
Logic Utilization	Used	Available	Utilization
Number of Slices	344	178800	<1%
Number LUT-FF pairs	0	344	0%
Number of bonded IOBs	384	600	64%

```
Device utilization summary:
---------------------------

Selected Device : 7v285tffg1157-2


Slice Logic Utilization:
 Number of Slice LUTs:                  344  out of  178800     0%
    Number used as Logic:               344  out of  178800     0%

Slice Logic Distribution:
 Number of LUT Flip Flop pairs used:    344
   Number with an unused Flip Flop:     344  out of    344   100%
   Number with an unused LUT:             0  out of    344     0%
   Number of fully used LUT-FF pairs:     0  out of    344     0%
   Number of unique control sets:         0

IO Utilization:
 Number of IOs:                         386
 Number of bonded IOBs:                 384  out of    600    64%
```

Fig. 6 Device Utilization Summary

```
Timing Details:
---------------
All values displayed in nanoseconds (ns)

=========================================================================
Timing constraint: Default path analysis
  Total number of paths / destination ports: 75083 / 129
-------------------------------------------------------------------------
Delay:               8.669ns (Levels of Logic = 18)
  Source:            a<1> (PAD)
  Destination:       sum<115> (PAD)

  Data Path: a<1> to sum<115>
                              Gate     Net
    Cell:in->out      fanout   Delay   Delay  Logical Name (Net Name)
    ----------------------------------------  ------------
     IBUF:I->O             2   0.003   0.665  a_1_IBUF (a_1_IBUF)
     LUT5:I0->O            3   0.050   0.365  11/13/Mmux_carry11 (c1)
     LUT5:I4->O            4   0.050   0.370  14/13/Mmux_y11 (n0016<2>)
     LUT5:I4->O            5   0.050   0.376  17/14/Mmux_y11 (n0015<3>)
     LUT5:I4->O            6   0.050   0.381  110/15/Mmux_y11 (n0014<4>)
     LUT5:I4->O            7   0.050   0.387  113/16/Mmux_y11 (n0013<5>)
     LUT5:I4->O            8   0.050   0.392  116/17/Mmux_y11 (n0012<6>)
     LUT5:I4->O            9   0.050   0.398  119/18/Mmux_y11 (n0011<7>)
     LUT5:I4->O            8   0.050   0.562  122/18/Mmux_y11 (n0010<7>)
     LUT6:I3->O           10   0.050   0.403  125/19/Mmux_y11 (n0009<8>)
     LUT6:I5->O           11   0.050   0.579  128/110/Mmux_y11 (n0008<9>)
     LUT6:I3->O           13   0.050   0.421  131/111/Mmux_y11 (n0007<10>)
     LUT5:I4->O           12   0.050   0.414  134/112/Mmux_y11 (n0006<11>)
     LUT6:I5->O           14   0.050   0.425  137/113/Mmux_y11 (n0005<12>)
     LUT6:I5->O           15   0.050   0.640  140/114/Mmux_y11 (n0004<13>)
     LUT6:I2->O           16   0.050   0.743  143/115/Mmux_y1 (n0003<14>)
     LUT6:I1->O            1   0.050   0.339  148/13/Mmux_y11 (sum_115_OBUF)
     OBUF:I->O                 0.002          sum_115_OBUF (sum<115>)
    ----------------------------------------
    Total                      8.669ns (0.805ns logic, 7.864ns route)
                                       (9.3% logic, 90.7% route)
```

Fig. 7 Timing Details of the proposed architecture

V. CONCLUSION AND FUTURE SCOPE

studied about different adders among compared them by different criteria like Area, Time and then Area-Delay Product etc. so that we can judge to know which adder was best suited for situation. After comparing all we came to a conclusion that Carry Select Adders are best suited for situations where Speed is the only criteria. Similarly Ripple Carry Adders are best suited for Low Power Applications. it is suitable for situations where both low power and fastness are a criteria such that we need a proper balance between both as is the case with our Project.

Multipliers are one the most important component of many systems. So we always need to find a better solution in case of multipliers. Our multipliers should always consume less power and cover less power. So through our project we try to determine which of the three algorithms works the best. In the end we determine that radix 4 modified booth algorithm works the best.

REFERENCES

[1] K.H. Tsoi, P.H.W. Leong, "Mullet - a parallel multiplier generator," fpl, pp.691-694, International Conference on Field Programmable Logic and Applications, 2005., 2005

[2] S. Tahmasbi Oskuii, P. G. Kjeldsberg, and O. Gustafsson, “Transition activity aware design of reduction-stages for parallel multipliers,” in Proc. 17th Great Lakes Symp. On VLSI, March 2007, pp. 120–125.

[3] Ayman A. Fayed, Magdy A. Bayoumi, "A Novel Architecture for Low- Power Design of Parallel Multipliers," wvlsi, pp.0149, IEEE Computer Society Workshop on VLSI 2001, 2001

[4] M. 0. Lakshmanan, Alauddin Mohd Ali, "High Performance Parallel Multiplier Using Wallace-Booth Algorithm," IEEE International Conference on Semiconductor Electronics, pp. 433-436, 2002.

[5] Design Compiler User Guide. Synopsys, Inc., Nov. 2000. [6] Z. Huang, “High-Level Optimization Techniques for Low-Power Multiplier Design,” PhD dissertation, Univ. of California, Los Angeles, June 2003.