Exploring Design and Performance of Carry-Save Multipliers

Nicholas Marks

I. Abstract

This paper investigates a carry-save multiplier design and its associated performance when implemented and simulated using 45 nm CMOS technology in Cadence. Prior work such as that in [1] achieve power draws of roughly 20 μW with an overall time delay of approximately 1.5 ps, using a low gigahertz frequency clock, so the proposed specifications are within similar margins to this and other prior work.

II. Introduction

Multiplication of a 4×12 product $A \times B$ can be modelled as the addition of four 12-digit numbers left-shifted to account for their position in the multiplicand:

$$\begin{array}{rr} & 1\,2\,3\,4\,5\,6\,7\,8\,9 \\ \times & \qquad\qquad 1\,2\,3\,4 \\ \hline & 4\,9\,3\,8\,2\,7\,1\,5\,6 \\ & 3\,7\,0\,3\,7\,0\,3\,6\,7 \\ & 2\,4\,6\,9\,1\,3\,5\,7\,8 \\ 1\,2\,3\,4\,5\,6\,7\,8\,9 & \\ \hline 1\,5\,2\,3\,4\,5\,6\,7\,7\,6\,2\,6 & \end{array}$$

The same method applies for multiplication in binary. The most basic form of multiplier architecture is one that uses a ripple-adder to compute the piece-wise products, however whilst being hardware-efficient, this produces bad propagation delay on the order $\mathcal{O}(n)$ where n is the number of bits in the multiplicand. Beetter is the carry lookahead adder which reduces the delay to $\mathcal{O}(\log n)$ however it does so at the cost of increased complexity. Furthermore, as the size of n increases, the distance the signals have to travel on the chip via interconnects increases proportionally to n and hence so does the propagation delay.

As far as how carry-save works, the idea is to compute the sum term S_i and carry-term C_i independently of one another. When adding three numbers together, a, b, c in binary, one can either compute $a + b$ and then the tertiary sum $(a + b) + c$, however this relies on waiting for the final term to propagate from lower bits. Instead, one computes S_i as $a_i \oplus b_i \oplus c_i$ and C_i as $(a_i b_i) \vee (b_i c_i) \vee (a_i c_i)$ using the formulae proposed originally by John von Neumann and subsequently in later work.

III. Proposed Technique

This paper presents a multiplier architecture using carry-save adders. It implements logic gates in 45nm CMOS technology and assumes throughout $V_{DD} = 1.2V$. It exploits DCVSL for:

- Minimum power draw, particularly static power draw
- Minimising hardware resources compared against complimentary CMOS

The design consists of a $(4 + 1) \times 12$ array of multiplying adder blocks, similar to that of [2]:

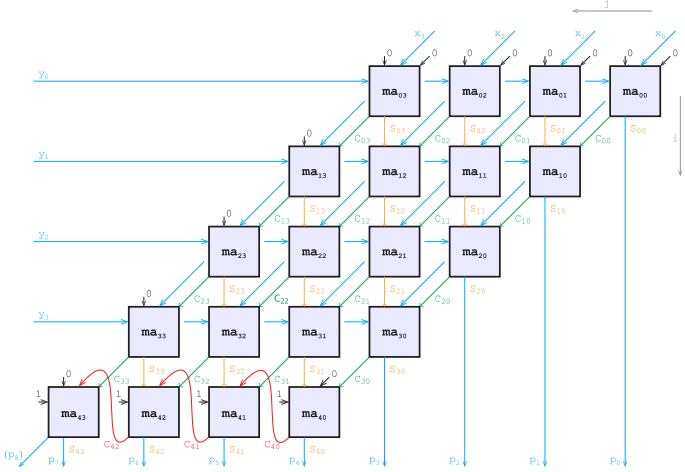

Fig. 1. Visualising a 4×4 bit carry-save multiplier

Each `ma` block consists of a 1-bit Manchester adder using a delete/propagate/generate schema for the sum S_i and carry-out $C_{0,i}$ bits. Assuming inputs a_i and b_i, we have

$$p_i = a_i \oplus b_i \tag{1}$$
$$g_i = a_i \wedge b_i \tag{2}$$
$$d_i = \overline{a_i} \wedge \overline{b_i} = \overline{a_i \vee b_i} \tag{3}$$

And then

$$S_i = p_i \oplus C_{in,i} \tag{4}$$
$$C_{0,i} = g_i + p_i C_{in,i} \tag{5}$$

This adder can be easily converted for multiplication of two numbers x and y at bit position i of the product under the mapping $a_i \to S_{in,i}$ and $b_i \to x_i \wedge y_i$.

A. Optimisations

Inspection of 1 and 4 shows that the gates $\texttt{ma}_{0i}$ can be replaced with simple AND gates since the carry-in and sum-in for this round of multiplication is 0. Furthermore, the bottom left-most cell can be dropped sincethe net carry-out is always 0 when S is 16 bits, and the final row of ripple adders can be replaced with a faster carry-lookahead adder as proposed in [3]. Refer to Table I for comparison of designs.

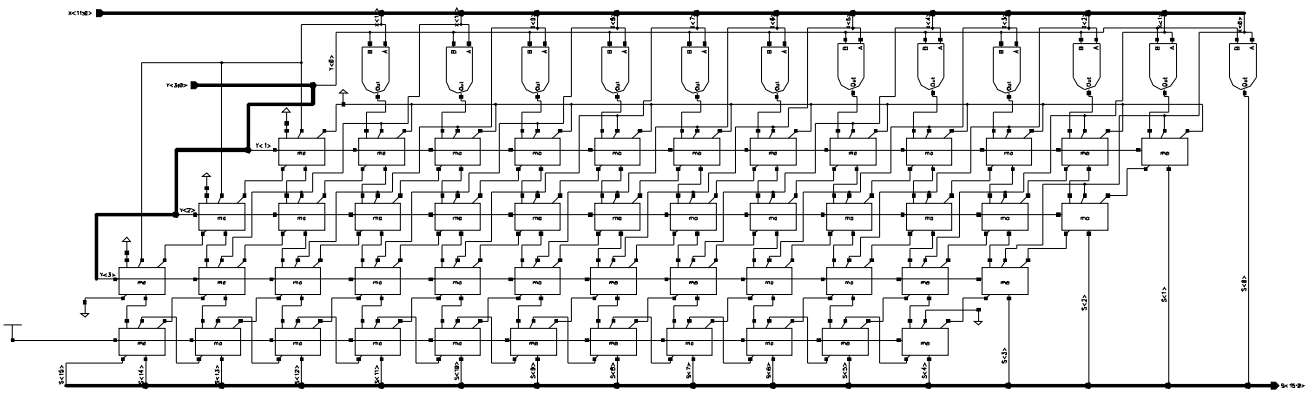

Fig. 2. The full 4×12 bit adder. The 4-bit value is fed horizontally across the columns running from top to bottom ($Y_0 \rightarrow Y_3$) whilst the 12-bit multiplicand runs across the rows ($X_0 \rightarrow X_{11}$ from right to left)

B. Schematic Design

We now present the transistor-level implementation of the 4×12 adder, starting from the top-level and working down. The full carry-save multiplier can be found in Figure 2 below. From here on we refer to the computation of product $X \times Y$ where $X[11:0]$ $Y[3:0]$.

We first consider each mulitplier block $\texttt{ma}_{i,j}$ representing computing the product of X and Y_j that has been left-shifted i positions.

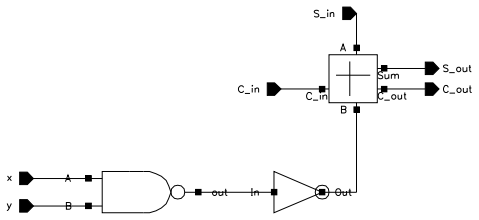

Fig. 3. The `ma` block

The block in the upper right above of Figure 3 designates a 1-bit Manchester adder that looks as follows:

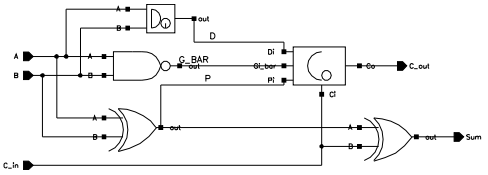

Fig. 4. The 1-bit adder for computing each sum and carry-out term

The blocks above labelled $\texttt{D}_0$ and $\texttt{C}_0$ compute the delete bit and carry-out bit respectively and are produced thusly:

And

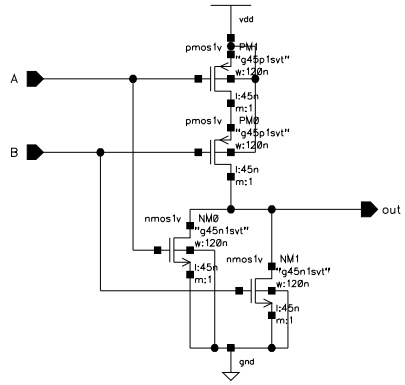

Fig. 5. The Delete block

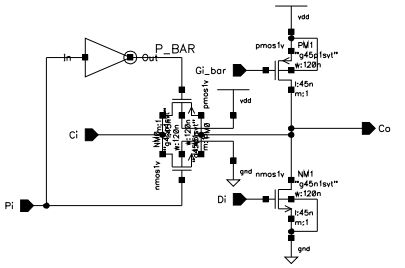

Fig. 6. The Carry-out block

Finally, we consider two different implementations of the XOR and NAND gates underlying Figures 3, 4 and 5 above, one using standard complimentary CMOS logic and the other ratioed DCVSL.

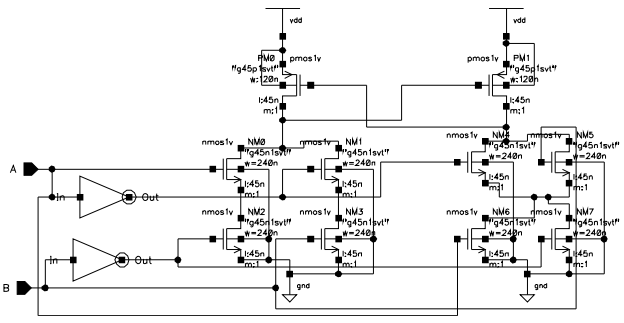

Fig. 7. DCVSL XOR Gate

Transistors for the DCVSL implementation have been sized accordingly. The complimentary CMOS follows a standard design, whilst the DCVSL is shown below.

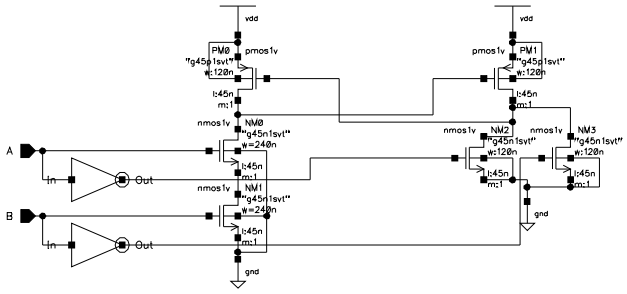

Fig. 8. DCVSL NAND Gate

IV. Simulation Results

Using standard complimentary CMOS logic for the CSA produces the following results. The design can be run at a clock of 1.11 GHz (periodicity of 0.9 ns) before erroneous outputs are produced.

The average propagation delay can be calculated from the graph as 173 ps, and the average power dissipation for the testbench is given below:

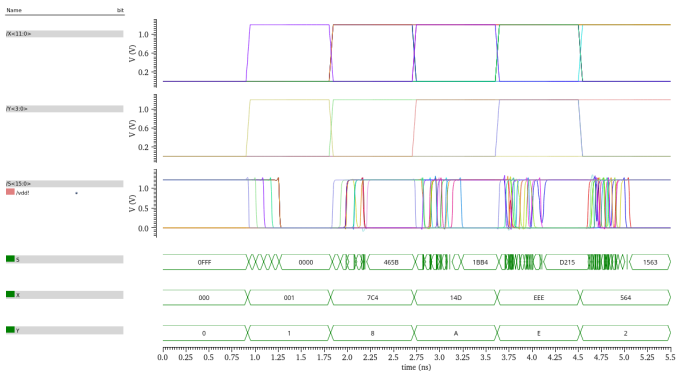

Fig. 9. Simulation results for 4x12 adder at 1.11 GHz

Fig. 10. Power dissipation for unoptimised complimentary CMOS giving incorrect results when running beyond 1.11 GHz

However, once we convert to using DCVSL logic and implement the optimisations sugested in the section on III, then the device can be run at a marginally higher clock ($\approx$ 1.25 GHz), though at a slightly higher power draw. Note that the same test values of X and Y were provided as above:

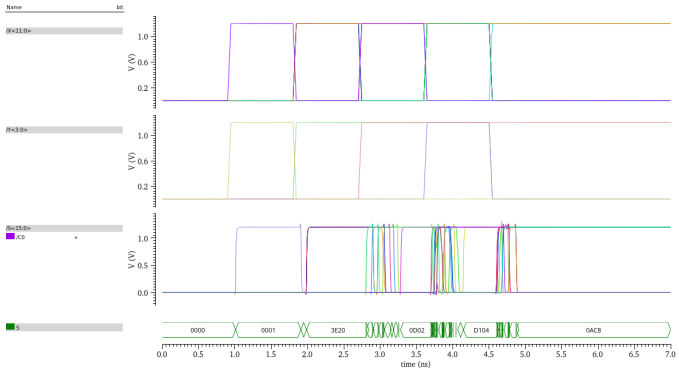

Fig. 11. The optimised multiplier running at 1.25 GHz

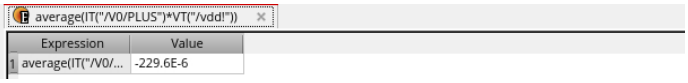

Fig. 12. The power draw of the optimised device at 1.25 GHz

Observation of the above figures shows that the DCVSL, optimised implementation allows for faster throughput and in general less instability in the transition region of `Sum` generation, however it does come at the cost of increased power draw, perhaps owing to static power consumption of the DCVSL design.

In both cases, the worst-case propagation delay was caused when the carry bits had to propagate across the entire multiplication, such as when switching from $0x000 \times 0x0$ to $0xEEE \times 0xE$.

TABLE I
A Summary of Multiplier Performance

Logic type	Bit size	Delay (ns)	Power (μW)
Complimentary CMOS	4×12	0.173	132.8
DCVSL optimised	4×12	0.142	229.6
Sub-threshold ([4])	8×8, $V_{DD} = 0.862V$	0.3	0.101
CSA of [3]	32×32	54.1×10^3	21.8×10^3

References

[1] R. A. Javali, R. J. Nayak, A. M. Mhetar, and M. C. Lakkannavar, "Design of high speed carry save adder using carry lookahead adder," in *International Conference on Circuits, Communication, Control and Computing*. IEEE, 2014, pp. 33–36.

[2] C. Vonk, "Carry-save array multiplier using logic gates," *Coert Vonk*, October 2015. [Online]. Available: https://coertvonk.com/hw/building-math-circuits/faster-parameterized-multiplier-in-verilog-30774

[3] R. P. P. Singh, P. Kumar, and B. Singh, "Performance analysis of 32-bit array multiplier with a carry save adder and with a carry-look-ahead adder," *International Journal of Recent Trends in Engineering*, vol. 2, no. 6, p. 83, 2009.

[4] B. C. Paul, H. Soeleman, and K. Roy, "An 8× 8 sub-threshold digital cmos carry save array multiplier," in *Proceedings of the 27th European Solid-State Circuits Conference*. IEEE, 2001, pp. 377–380.