Chapter 1

Introduction

The tremendous advancements in VLSI technologies in the past few years have fueled the need for intricate tradeoffs among speed, power dissipation and area. With gigahertz range microprocessors becoming common place along with the perennial increments in power dissipation, the emphasis is even more on pushing the speeds to their extreme while minimizing power dissipation and die area. The complexity of computation intensive circuits like dedicated DSP processors, CDMA systems, RSA algorithms have increased greatly in the past three decades instigating the need for large integration density, high-speed operation, low-power dissipation and low costs. The focus of our research, here at the Virginia Tech VLSI for Telecommunications (VTVT) research laboratory is two fold. The first aspect of is the analysis of a relatively unexplored logic style called MOS Current Mode Logic (MCML), which has been shown in the past to aid in the design of high performance arithmetic circuits with minimal power dissipation. The second goal is to design high-speed arithmetic circuits, in particular, gigahertz range multipliers that make use of many attractive features of the MCML logic style.

Multipliers are critical components of many computational intensive circuits, that include real time signal processing and arithmetic based systems. The increasing demand for fast arithmetic units in floating-point co-processors, graphic processing units, CDMA systems and DSP chips has shaped the need for highly integrated, very high speed multipliers.

Traditionally multiplier architectures fall in into one of the following two categories, viz. array multipliers and tree multipliers. Whereas in array multipliers, the latency is a linear function of the word length of the multiplier, *O(n)*, in the case of tree multipliers, the latency is a logarithmic function of the word length, *O[log(n)]*. Hence tree structures are most suitable for high speed multiplier designs and have been adopted in our research. To further enhance the performance of our multiplier, we adopted pipelined multiplier design by inserting a register stage after every compressor cell.

As mentioned earlier, the unique aspect of our work involves taking advantage of many attractive features of the relatively new logic style called MOS Current Mode Logic (MCML) in developing very high speed pipelined tree multipliers capable of performance in the Gigahertz range. A small library of logic gates including NAND/AND, XOR3/XNOR3, 3x2 and 4x2 Compressors and Master-slave Flip-flops that form the core components of the multiplier were designed and optimized for high speed operation using this logic technique. As would be evident in the ensuing sections, simulation results place our Multiplier design among the fastest found in contemporary literature.

This thesis is organized as follows: Chapter 2 introduces the basic concepts and operation of MOS Current Mode Logic (MCML) gates. The operation of the most basic gate, an inverter-buffer is dealt with in detail. The design of other gates like XOR3/XNOR3, AND/NAND of our optimized library are also explained in this chapter. Also dealt with in detail are the various design parameters and the challenges involved in optimizing them. The methodology followed for optimizing the design parameters of the

MCML gates are also discussed in this chapter. Chapter 3 gives an overview of the different multiplier architectures in common use today. A comparison of different multipliers found in contemporary literature and various design bottlenecks that cripples high speed operation are discussed. Chapter 4 explains in detail the architecture of the proposed multiplier designs. The simulation setup and simulation results are mentioned and discussed in detail. Chapter 5 gives the results of the simulations performed on all the gates and the top-level multiplier design. Further discussions include the comparison of obtained results with expected results and with comparable multiplier designs found in contemporary literature. Chapter 6 concludes the thesis with a summary of our research and pointers for further work in this area.

Chapter 2

Design of Basic MCML Gates

This chapter explains the principles of MOS Current Mode Logic and the operation of a basic MCML gate. First, the operation of an Inverter/Buffer is discussed in detail to understand designing gates in this logic style. Next, the designs of other simple MCML gates like NAND, XOR, Flip-flop etc. are explained. Next, the MCML design metrics that include delay, power dissipation, and voltage swing ratio and other performance criteria are discussed. Finally, the different MCML design parameters such as bias voltages, transistor lengths and their effect on performance parameters are discussed.

2.1 MCML Basics

The operation of an MCML gate may be understood with the help of the basic structure of an MCML gate, shown in Figure 2.1 [2]. The main parts of the MCML gate are: the load resistances R_L, the differential pull-down network (PDN) with complementary sets of inputs and outputs, and a constant current source.

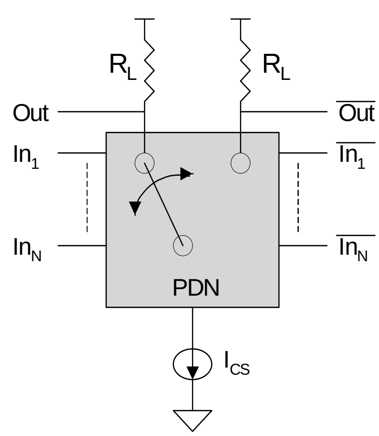


Figure 2.1 ***Basic MCML gate***

The differential inputs are fed to the pull down network (PDN). The design of the PDN takes a tree-like differential structure, similar to a CMOS circuit technique called Differential Cascode Voltage Switch (DCVS) [1]. The output and its complement are available at the two arms as indicated in Figure 2.1. The PDN is grounded through a constant current source I_{CS}, which is usually an NMOS transistor. The differential PDN steers the current between the two pull up resistance and through the constant current source. The differential tree like structure of the MCML gate would be more apparent in the subsequent discussions on logic gates.

The total voltage swing at the output and its complement is $\Delta V = I_{SC} \times R_L$ which is usually controlled by setting the value of the current source I_{SC} usually an NMOS transistor, and the effective value of R_L, which is usually a PMOS device. The value of voltage swing is of the order of a few hundred mV and is a very crucial leverage factor in high-speed operation. The equations for the total propagation delay, power dissipation and power delay product of an MCML logic circuit and its CMOS counterpart may be given as shown in Table 2.1 [2]:

Table 2.1 ***Expressions for Delay, PD, PDP and EDP for MCML & CMOS logic styles.***

Parameter	MCML Logic Style	Conventional CMOS Logic
Propagation Delay (τ)	$\boldsymbol{\tau}_{\mathbf{MCML}} = \dfrac{C_L \times \Delta V \times N}{I_{SC}}$	$\boldsymbol{\tau}_{\mathbf{CMOS}} = \dfrac{C_L \times V_{DD} \times N}{\frac{\beta}{2} \times (V_{DD} - Vt)^2}$
Power Dissipated (PD)	$\mathbf{PD_{MCML}} = V_{DD} \times I_{SC} \times N$	$\mathbf{PD_{CMOS}} = N \times C_L \times V_{DD}^2 \times f$

Power Delay Product (PDP)	**PDP_{MCML}** $= N^2 \times C_L \times \Delta V \times V_{DD}$	**PDP_{CMOS}** $= N \times C_L \times V_{DD}^2$
Energy Delay Product (EDP)	**EDP_{MCML}** $= \dfrac{N^3 \times C_L^2 \times \Delta V^2 \times V_{DD}}{I_{SC}}$	**EDP_{CMOS}** $= \dfrac{N^2 \times C_L^2 \times V_{DD}^2}{\frac{\beta \times (V_{DD} - Vt)^2}{2}}$

Where N is the logic depth of the MCML logic circuit, f (**=1/ τ_{CMOS}**) is the frequency of CMOS logic circuit, β is transistor gain.

As can be seen from Table 2.1, the delay of an MCML logic circuit varies linearly with voltage swing ΔV and does not vary with the supply voltage V_{DD} as is the case with conventional CMOS logic circuits. The power dissipated in an MCML logic circuit varies linearly with the supply V_{DD} and is independent of the operating frequency [2], [3], unlike conventional CMOS circuits where power dissipation depends linearly on operating frequency and has a square-law dependence on supply voltage. Since the delay depends linearly on ΔV and is independent of supply $V_{DD,}$ it can be minimized by determining an acceptable and a low voltage swing ΔV while simultaneously minimizing power dissipation by reducing the supply voltage V_{DD}, which has little effect on the delay. Thus, as the power dissipated is independent of the operating frequency, MCML circuits may be operated at very high speeds with minimal power consumption in contrast to conventional CMOS circuits.

Another important issue concerning MCML logic circuits is the need for a shallow logic depth. As in any other logic technique, signal regeneration is of prime importance in MCML gates. The DC voltage gain is usually fixed close to 1.4 to account for process variations and the logic depth is minimized to prevent signal deterioration thereby enhancing regeneration and stability [2]. Also, reducing the logic depth (N) optimizes the energy delay product (EDP) as EDP_{MCML} varies cubically with N unlike EDP_{CMOS} that varies with square law dependence. Further, keeping the signal swing (ΔV) at the output at low values of about few hundred mV helps in minimizing EDP_{MCML} that has a square law dependence on ΔV, unlike EDP_{CMOS} that depends not on the signal swing but on the supply voltage itself.

2.2 The MCML Inverter/Buffer

After understanding the basic structure and design of an MCML gate, it is now appropriate to look deeper into its transistor level implementation. The most basic MCML gate is the inverter buffer shown in Figure 2.2.

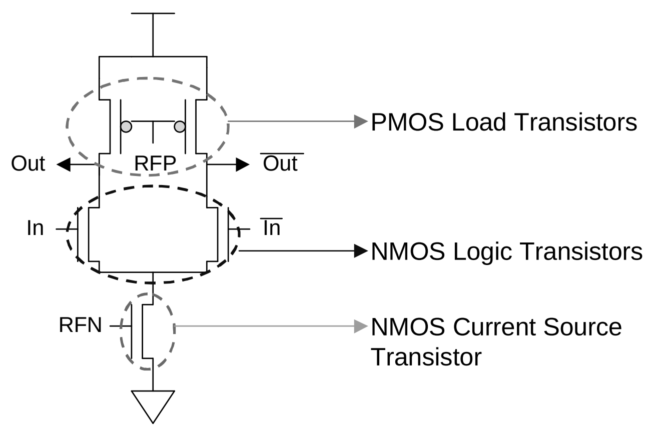


Figure 2.2 ***The MCML Inverter/Buffer***

The two PMOS transistors at the head of the gate are modeled as load resistances. The value of the load resistance may be adjusted by varying the dimensions of the PMOS transistors and ensuring operation in the triode region by controlling the bias voltage, RFP. The pull down network consists of two NMOS transistors to which the differential input is fed. The NMOS transistor at the tail acts as a constant current source and provides a regulated path for the current to the ground. The value of the current may be adjusted by fixing the value of the current source bias, RFN. It is usually a non-minimal length device that provides higher output impedance and good current matching. The current source transistor is biased for operation in saturation region. The current source, in fact, may be implemented in several topologies (e.g. cascaded current mirrors) but for our purposes, a single NMOS was decided keeping in mind the digital nature of our design and to effect as minimal an area as possible.

The operation of an ideal MCML Inverter/Buffer gate is as follows: suppose the input ‘In’ is logic high; this causes the transistor of the left arm of the PDN to conduct, while transistor in the right arm remains cut-off. Ideally, all current flows from the PMOS load through the left arm and drains through the tail current source, pulling down the ‘Out’ node to logic low and its complement to logic high. In reality, unlike the ideal gate described above, some current always flows through the ‘OFF’ path degrading the signal voltage to a value below V_{DD}. This happens because the transistors in the ‘OFF’ path are not fully turned off but rather are less saturated than those in the ‘ON’ path. Although this degrades the signal value, nevertheless, this would enable achieving high switching speeds. Thus, in effect, the differential PDN switches the current provided by the current source between the two arms to realize the logic at the two output nodes.

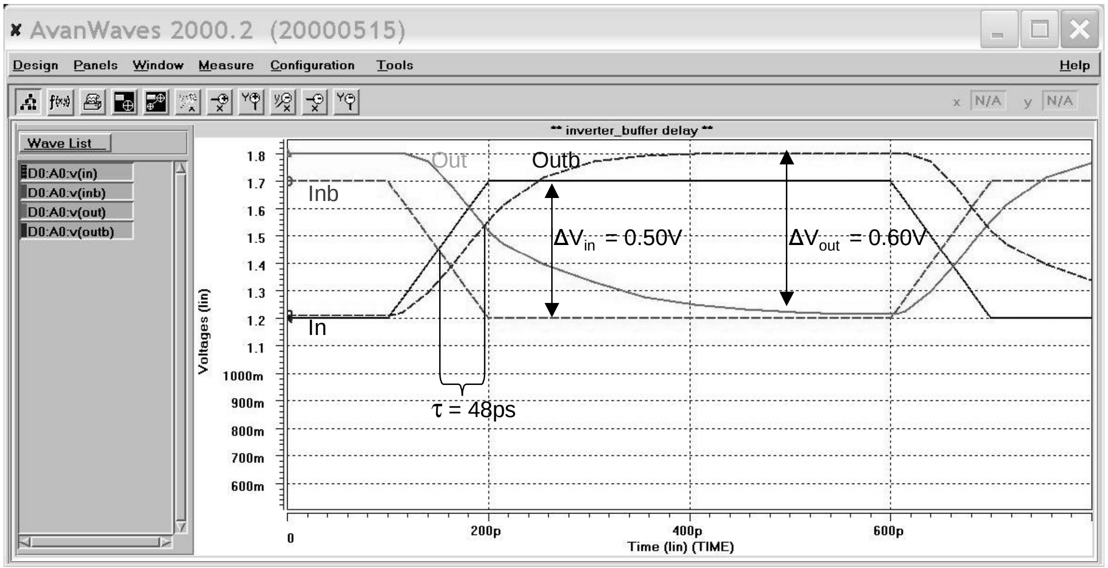


Figure 2.3 ***Simulation results of Inverter/Buffer.*** ***V_{DD}=1.8V, ΔV = 0.5 V, I=72 μA***

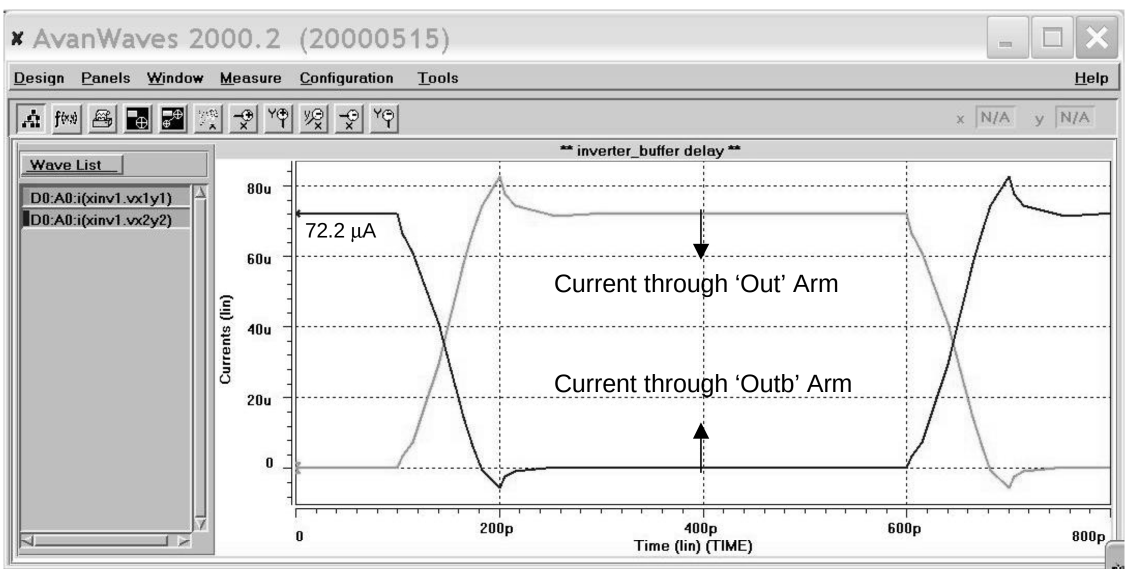


Figure 2.4 ***Inverter/Buffer: Waveform showing current through two arms of gate.***

To understand the operation of a simple MCML Inverter/Buffer, the simulation waveforms are provided in Figure 2.3 and Figure 2.4. In Figure 2.3, signals 'In' and 'Inb' are the two complementary inputs signals. 'Out' (Inverted output) and 'Outb' (Buffered output) are the two complementary output signals. Figure 2.3 shows the current through the two arms of the Inverter/Buffer. When the input 'In' rises, the transistor in the left arm conducts and routes the current (Green line in Figure 2.4) from the supply to the current source, causing the 'Out' signal (Buff line in Figure 2.3) to go low. It may be noted that the propagation delay is calculated from the point where the two inputs cross each other and the point where the two outputs cross unlike the conventional 50% rule in case of CMOS circuits.

2.3 Other MCML Gates

Apart from the Inverter/Buffer introduced in the previous section, other MCML gates designed and optimized include AND/NAND, XOR3, Majority function, D-latch and Master-Slave Flip-flops. The goal of designing these gates is to create a small library of optimized gates for high speed operation with little concern for power dissipated and area, to serve the purpose of achieving our ultimate goal: design of high speed multipliers. In other words, this library is not a generally optimized one but rather, one that has been customized for high-speed operation for use only in our multiplier design, to achieve operation in the gigahertz range.

The design of differential pull down networks of the MCML AND/NAND, XOR3 and the majority function gates are based on CMOS logic techniques called Differential Cascode Voltage Switch (DCVSL) and Binary Decision Diagrams (BDD). A complete overview of DCVSL or BDD is beyond the scope of this report. More information on these techniques may be found in [1]. However, this design structure would be more apparent in the ensuing discussions. The MCML AND/NAND gate is shown in Figure 2.5. The differential nature of the gate is apparent in the way in which the pull down network is derived from the binary decision diagram of the gate shown in Figure 2.5(b).

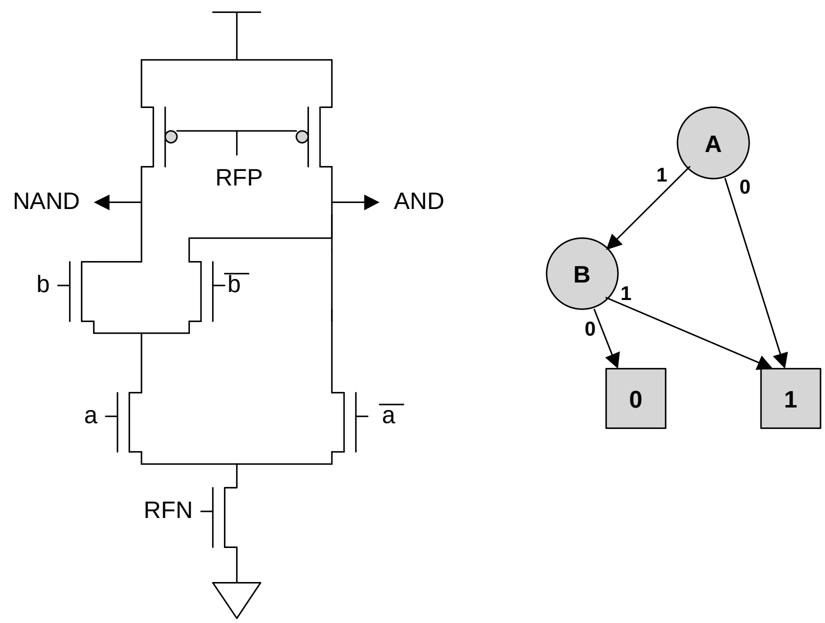


Figure 2.5 (a) ***An MCML AND/NAND gate*** **(b)** ***and its Binary Decision Diagram(BDD)***

The MCML XOR3 gate shown in Figure 2.6 is based on the DCVSL design proposed by Chu and Pulfrey in [1]. The general structure of an n-input XOR gate is shown in Figure 2.7. The DCVS XOR3 gate structure proposed by Chu and Pulfrey that has been adapted to our MCML design has two transistors less than the BDD design proposed by Musicer et al. in [2]. Further, simulation results confirm that the former was found be faster than the latter with the added advantage of savings in area.

The pull down network of the XOR gate consists of a pair of interrelated binary decision trees (or DCVS trees) [1]. The general DCVS tree of the XOR gate (Figure 2.7) is properly designed such that: 1) when the input vector $x = (x_1, \ldots, x_n)$ is the true vector of the switching function $Q(x)$, node Q is isolated from ground and node Q is grounded by a unique conducting path through the tree; and 2) when $x = (x_1, \ldots, x_n)$ is the false vector of $Q(x)$, the reverse holds. The three input XOR gate shown in Figure 2.6 is designed by replicating the building block twice in the general structure. The functionality of this circuit may be easily verified by trying all the possible combinations of input vectors.

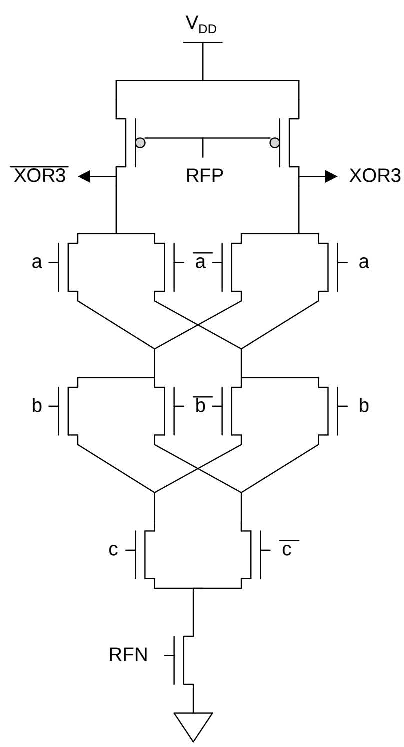


Figure 2.6 ***MCML XOR3 Gate***

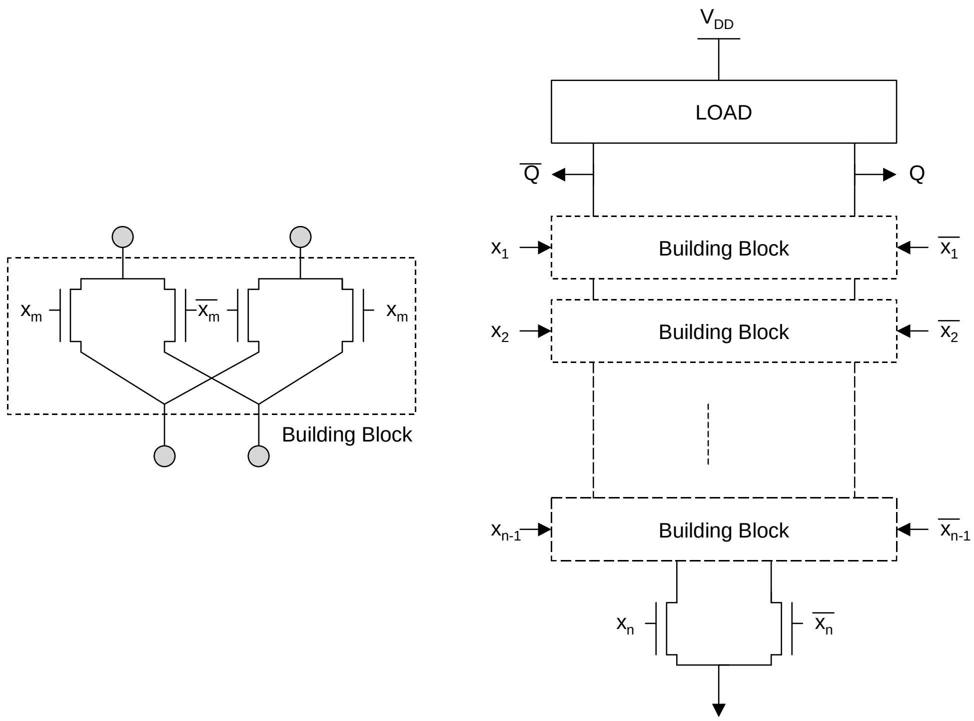


Figure 2.7 ***DCVSL structure for a general n-input XOR gate [1]***

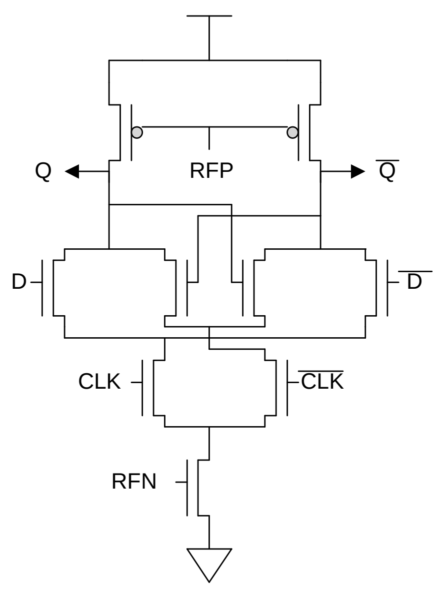
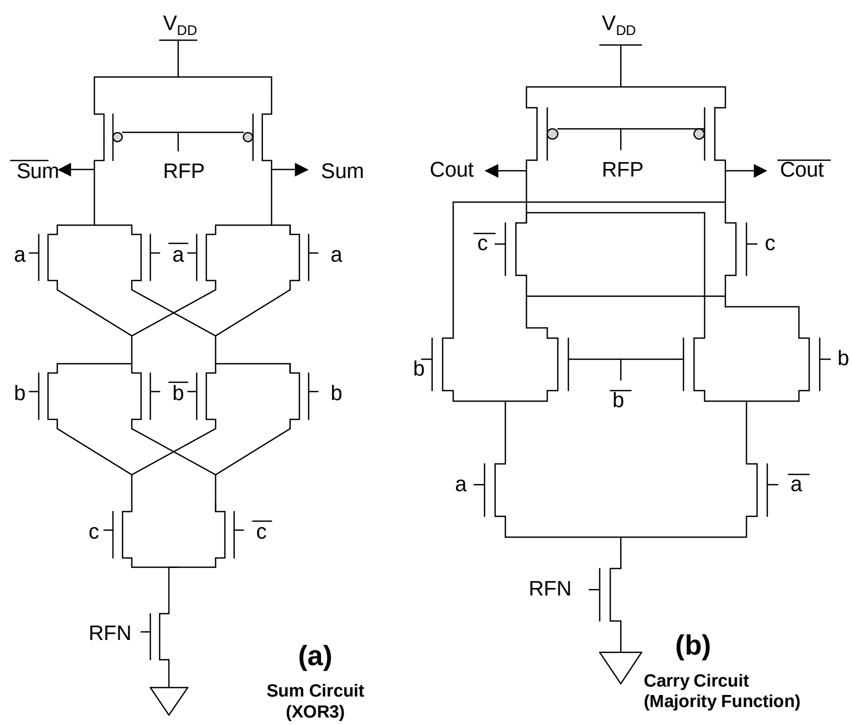


Figure 2.8 ***The MCML D-Latch***

The basic MCML storage element, D-latch is shown in Figure 2.8 [3]. It has a simple cross-coupled structure. A positive edge triggered Master-Slave D flip-flop is created by cascading two such D-latches making the master latch sensitive to logic low and the slave latch sensitive to logic high. The D flip-flop is used for pipelining purposes in our multiplier architecture. Next, we shall see the design of the MCML full adder used in our high-speed multiplier design.

The complete MCML full adder (or a 3x2 compressor) is shown in Figure 2.9. It consists of a three input XOR gate as described in the previous sections and a carry generate gate, which is nothing but a majority function. Simulation results explained in subsequent sections show these designs to be faster and minimal transistor ones than those proposed by Musicer in [2]&[4] using binary decision diagrams.



Sum = abc + a’b’c + ab’c’ + a’bc’

Sumb = a’b’c’ + a’bc + ab’c + abc’

Cout = ab + ab’c + a’bc

CoutB = a’b’ + a’bc’ + ab’c’

Figure 2.9 ***The MCML Full Adder (or 3x2 Compressor)***

All the MCML gates introduced in this section were designed and simulated using Cadence and HSPICE using TSMC's 0.18 μm technology. The details of the simulation setup and the simulation results are provided and discussed in detail in Chapter 5. Now that we have a good understanding of the differential MCML logic style and the design of basic gates, we shall next consider the various parameters that affect the performance of the gates before going further to discuss the optimization methodology.

2.4 MCML Design Metrics

In this section, we shall discuss in detail the different performance criteria of an MCML gate such as voltage swing ratio, voltage gain, RFP, RFN bias voltage limits among others that may be used as leverage to optimize the performance of our small library of MCML gates and ultimately the MCML multiplier. Following this, we show how the different input parameters such as transistor dimensions, input swing voltage etc. affect these performance criteria.

2.4.1 Gain

The gain of any circuit is an important design criterion that ensures regenerative property and bi-stability of the circuit. Once of the main criterion that ensures proper functionality in digital circuits is that there exists a point in the DC voltage transfer curve where the gain is larger than one. In conventional CMOS circuits, this requirement directly affects the robustness of the circuit and thereby the signal regeneration and bi-stability. Regenerativity ensures that a disturbed signal (due to noise in the circuit) gradually converges back to one of the nominal voltage levels after passing through a number of logical stages [5]. Bi-stability refers to the existence of only two stable logic states in sequential circuits like latches and flip-flops.

Unlike traditional CMOS circuits that have an inherently high value of DC gain, MCML circuits have naturally low values of DC gain. Large values of gain can still be achieved but with a compromise in area, delay and power. On the brighter side, the differential nature of MCML circuit obviates the requirements of very high gains. That is, most of the noise that affects conventional CMOS circuits becomes common mode noise in MCML circuits and is rejected by the differential logic. Thus, low switching noise operation is very suitable for low gain operation.

For our research purposes, we set the limit on DC voltage gain to 1.2 [2]. This value that is slightly greater than unity was chosen to account for the leeway required to meet process variations and impedance matching conditions.

2.4.2 Voltage Swing Ratio (VSR)

The voltage swing ratio is defined as the ratio of the output signal swing voltage and the input signal swing voltage [4]. In ideal MCML gates, no current flows in the 'OFF' path and all the current flows to the current source through the 'ON' path. However, in typical MCML gates, some current does flow through the 'OFF' path causing degradation in the output swing voltage. It is thus possible that in a chain of MCML gates, the signal dies as it progresses through the chain and all that is seen at the output is noise. However, this typically does not happen as just like some gates degrade the output swing, some do regenerate the swing thus compensating for the loss. In practical MCML circuit design, the value of voltage swing ratio is maintained close to unity.

2.4.3 Delay, Area and Power Dissipation

These metrics carry their common interpretations and are used throughout this report in analyzing the performance of the MCML circuits. Since the goal of our research is to develop gigahertz range multipliers, optimizing delay takes prime importance in this work.

2.5 Design Parameters

In this section, the input design parameters of an MCML gate are analyzed in detailed and the tradeoffs involved in the optimization process are described. The design parameters of a general MCML gate are summarized in Table 2.2.

Table 2.2 ***MCML Design parameters***

Parameter	Description
V_{DD}	Supply Voltage.
ΔV	Input and output swing voltage.
I_{SC}	Current source current.
W, L	Width and length of NMOS logic transistors in pull down network.
W_{RFP}, L_{RFP}	Width and length of load PMOS transistors.

W_{RFN}, L_{RFN}	Width and length of NMOS current source transistors.
RFP,RFN	Bias voltages of PMOS load transistor and NMOS current source transistor respectively.

We describe how the parameter in the table above and see how they affect the performance.

2.5.1 Supply Voltage (V_{DD})

The upper limit of the supply voltage is set by the technology used for our work. For our research, we have used TSMC 0.18 μm technology where the maximum supply voltage is limited to 1.8 V. From the point of view of low power dissipation, it is very desirable to operate the MCML circuit at reduce voltages. However, though the supply voltage does not directly affect the delay (Table 2.1), it does reduce the voltage swing ratio and the mid swing voltage gains due to reduced operating headroom for the transistors. This reduce voltage swing at the output adversely affects the delay of the gate. Since the goal of our research is to develop high-speed multipliers, we set the supply voltage at its maximum value of 1.8 V for all our gates.

2.5.2 Input and Output Voltage Swings (ΔV)

Input and output voltage swings values are two of the most important parameters of MCML circuits that have serious impact on the performance of the gates. Since the delay is linearly dependent on voltage swing, it is imperative to maintain it at as low a value as possible, usually of the order of few hundred mV. The lower limit on voltage swing is set by the DC voltage gain requirements and the quality of current switching. The DC voltage gain is usually set to a value of 1.4 [4] while a lower voltage swing affects the quality of switching current and the voltage swing ratio (VSR).

2.5.3 Current Source Current (I_{SC})

The current source current is one of the parameters of an MCML gate that may be varied over a wide range and has serious impacts on both speed and power of the gate. The value of the current is usually set by varying the transistor dimensions and according to the speed and power requirements based on simulations. It may be recalled from Table 2.1 that the delay of an MCML gate is inversely proportional to I_{SC} while the power increases linearly with I_{SC}. For our goal of realizing very high-speed multipliers with little concern for power, it is desirable to maintain this value on the higher side, usually in the order of a few micro-amps. A high value of current may be achieved by appropriately sizing the transistors in the MCML gate. The effect of varying sizes of the logic, load and current source transistors may be understood from the discussions in the subsequent sections.

2.5.4 NMOS Logic Transistor Sizes (W, L)

The sizes of the logic transistors used in the differential pull down network of an MCML gate affect almost all performance criteria. The transistor sizing provides a great degree of freedom for the designer to make decisive tradeoffs between different performance criteria. In general, the widths of all the logic transistors in a differential pair are kept equal and the lengths are kept at their minimal values (0.18 μm) as there is no benefit from increasing them.

Usually, increasing the widths of logic transistors increases the voltage gain along with the undesirable effects of increasing power dissipation and delay. This leads to direct tradeoffs between higher speeds, power dissipation and voltage gain. It is thus desirable for MCML circuits to operate with low voltage gain to maintain performance. For our goal of realizing very high-speed multipliers, we usually set the widths of logic transistors to twice its minimal value (0.54 μm) and the lengths at its minimal value (0.18 μm) to maintain a low but acceptable voltage gain of 1.2.

2.5.5 Current Source NMOS Transistor Sizes (W_{RFN}, L_{RFN})

It is usually desirable to use nonminimal length transistors for the current mirror to increase the output impedance and to strengthen its robustness. Hence, the major tradeoff is between large areas and robustness. For attaining very high speeds and to enable a low voltage design, the value of W_{RFN}/L_{RFN} is increased. A large W_{RFN}/L_{RFN} also helps in setting high values of the current I_{SC} that improves the speed, trading off power. For our gate library and high-speed multiplier designs we have used W_{RFN} = 0.9 μm and L_{RFN} = 0.36 μm.

2.5.6 PMOS Load Transistor Sizes (W_{RFP}, L_{RFP})

The task of sizing the PMOS load transistors is a complex one involving tradeoffs between many performance parameters. The goal of sizing the PMOS transistors is to model them as closes as possible to load resistors. To this end, it is desirable to operate these transistors in the triode region. The bias voltage RFP is maintained at 0.3 V to enable operation in triode region. Non-minimal length transistors are used in some cases to increase the voltage gain of the gate.

Increasing the width (W_{RFN}) or the length (L_{RFP}) of the load devices helps in reducing the effective load resistance, thereby improving the propagation delay [2]. However, this also results in improved output load capacitance negating the improvement in delay and further causes the non-linearity of the effective load resistance to improve. The transistor sizes are further governed by the value of the bias voltage RFP. In practice, the value of W_{RFN} and L_{RFP} are so chosen that the linearity of the effective resistance is not compromised.

The several trends and tradeoffs discussed above leads to a fairly complex optimization problem. After several simulations and careful analysis of the tradeoffs involved, the W_{RFN} and L_{RFP} for our gates were set at 0.45 μm and 0.18 μm.

2.6 Summary

In this chapter, we introduced the new logic style, MOS Current Mode Logic and also saw how different gates may be designed using this logic technique. In the subsequent chapters, we shall see how these gates are optimized and used intelligently to design very high-speed multipliers. The next two chapters, Chapter 3 and Chapter 4 deal with the design of high-speed multipliers and our proposed MCML multiplier architectures respectively. The optimization methodology and the simulation results for the various gates of our design library and the overall multiplier are discussed in Chapter 5.

Chapter 3

Design of High Speed Multipliers

A multiplier is an essential unit in any digital signal processing circuit and often constitutes the critical path in DSP and floating-point units. The ever-increasing demand for higher calculation speeds in applications such as three-dimensional computer graphics (3DCG) and digital video filtering requires really high-speed multipliers.

3.1 Binary Multiplication

The Webster's dictionary defines multiplication as "a mathematical operation that at its simplest is an abbreviated process of adding an integer to itself a specified number of times". In other words, a number called the multiplicand is added to itself a number of times as specified by another number which is the multiplier to form the result which is the product. Starting from its least significant digit, the multiplicand is multiplied by each digit of the multiplier to yield the partial products. The rows of partial products are then placed one on top of the other, offset by one digit to align digits of the same weight. The final product is obtained by vertical summation of all the partial products [7]. Although multiplication is generally thought in terms of base-10, the same procedure also applies to base-2 (binary) numbers.

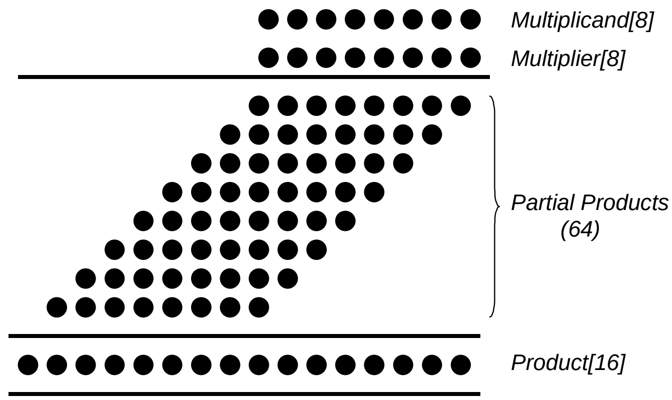


Figure 3.1 ***Dot diagram showing 8x8 multiplication***

The simplest method of multiplication is the add-and-shift multiplication algorithm illustrated with the help of Figure 3.1. This figure shows the data flow in an 8 x 8 multiplication process in the form of a dot diagram. Each black dot is a placeholder for a single bit, which is either a zero or one. Each horizontal row of dots represents a single copy of the multiplicand, which is acted upon by one bit of the multiplier. In the dot diagram for the 8 x 8 multiplication shown in Figure 3.1, there are 64 (8x8) partial products that are place one atop the other offset by one bit and the calculated product, which is 16 bits long, is placed right at the bottom.

In a binary multiplication process, the result of multiplying any one binary bit by another bit is either a zero or a one which is essentially the logic AND operation of the two bits [7]. Generating the partial products involves logical ANDing of two bits and may be carried out in a simple and efficient fashion. However, summing these partial products is a time consuming task and often is the main cause for speed bottlenecks in implementing multipliers. This process of summing may be carried out by software that runs on processors that do not have dedicated multipliers. However, present day applications require calculation speeds that are much higher than those achieved with such software algorithms and that calls for multipliers that are directly implemented in hardware.

Hardware implementation of the add-and-shift multiplication algorithm is faster than software synthesis but nevertheless, the demand for higher speed result in various multiplier architectures to push the limit. A plain add-and-shift algorithm is slow in hardware because as each additional partial product is summed, a carry must be propagated from the least significant bit to the most significant bit. This carry propagation is the main speed bottleneck and several techniques have evolved that target the optimization of the carry propagation delay.

The main architectural techniques employed in contemporary multipliers target the optimization of latency and throughput of the multiplier. The latency is the number of clock cycles from the time when the two inputs (Multiplicand and Multiplier) are applied to the multiplier and to the time when the product is available at its output. The throughput of a multiplier is defined as the number of multiplications that a multiplier can perform in one second. One method to increase the multiplier performance is to use encoding techniques that reduce the number of partial products to be summed. A common encoding method is *Booth's algorithm* [6]. Hardware implementations use a slightly modified version of Booth's algorithm referred to as *Modified Booth's algorithm* [6]. In general, for an n x n multiplier that uses the 2-bit version of this algorithm (Booth 2), there are [n(n+2)/2] partial products in contrast to n^2 partial products when no such encoding technique is used. Other encoding techniques in vogue today in hardware multiplier architectures include Redundant Booth, Booth with bias and Booth 3 with partially redundant partial products.

In order to achieve even higher performance, advanced hardware multiplier architectures have evolved that reduce the latency by using faster and more efficient methods for summing the partial products. The idea behind reducing the latency is to do away with the main design bottleneck: the carry propagate addition. The method most commonly used is the *carry-save addition* [6]. In carry-save addition, the carry propagation is done in the last step, while in all other intermediate steps a sum and carry are generated for each bit position. When two bits are added together, the carry propagates only to the next bit position and no ripple carry propagation occurs. One common method that has been developed for summing rows of partial products using a carry-save representation is the *Array multiplier*. Another method that uses simultaneous additions to reduce the number of summing stages employs *Tree structures* to realize extremely high-speed multipliers. Subsequent sections in this chapter discuss the advantages and disadvantages of these two techniques, viz. array multipliers (using carry-save arrays) and tree multipliers (using tree structures) in detail and analyze their performance.

3.2 Array Structures

Common array multiplier architectures are realized using rows of carry-save adders (CSA). A portion of an array multiplier along with the internal signal routings is shown in Figure 3.2 1. It may be recalled that MCML circuits unlike conventional CMOS, have differential inputs and output. To avoid clutter, these complementary signals are not shown in Figure 3.2 1 as well as in other block level architectural design diagrams in this thesis.

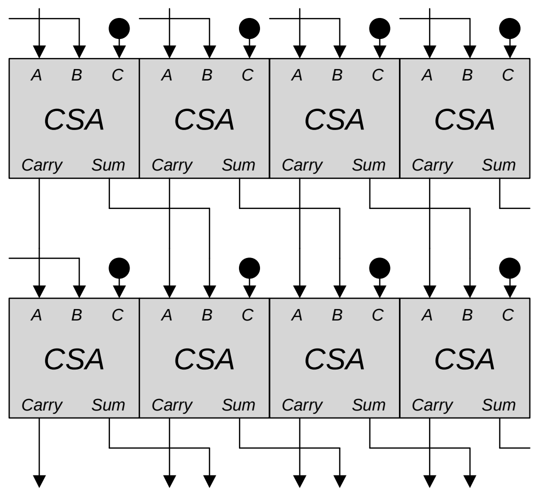


Figure 3.2 ***Two Rows of an Array Multiplier***

In a linear array multiplier, as the data propagates through the arrays, each row of CSA's add one additional partial product to the partial sum [7]. Hence, the carry does not propagate through the multiplier but is rather saved in every step of the partial product summation. This implies that the delay of the array multiplier is independent of the partial product width but depends on the depth of the array.

Figure 3.2 2 shows the design of a 5 x 5 multiplier in terms of full-adder cells and two input AND gates [8]. The sum outputs are connected diagonally, while the carry outputs are linked vertically, except in the last row, where they are chained from right to left. This design assumes unsigned numbers, but it can be easily extended to a 2's complement array multiplier using the Baugh-Wooley method.

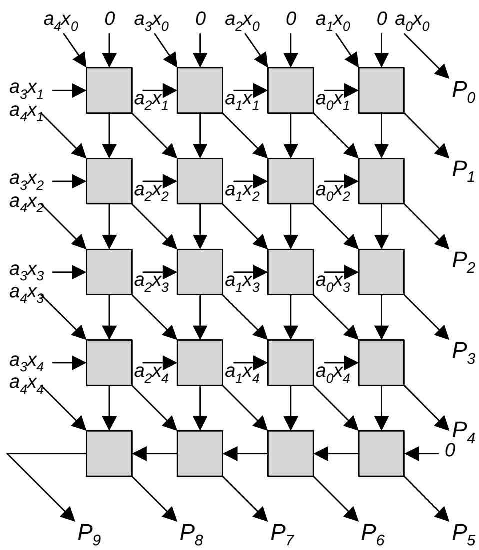


Figure 3.3 ***Design of a 5 x 5 array multiplier using full adder blocks***

The critical path though any n x n array multiplier goes through the main (top left to bottom right) diagonal and proceeds horizontally in the last row, in the case of a 5 x5 multiplier, to P_9 as shown in Figure 3.2 2. Thus the main speed bottleneck that makes this architecture one of the slowest is the last row carry propagate adder. Although the array architecture is much slower (larger latency) in comparison to the tree structure (to be discussed in subsequent sections), it is still a preferred design technique in the industry due to various reasons. An array multiplier is regular in its structure and uses only short wires that go from one full adder to horizontally, vertically, or diagonally adjacent full adders [8]. Thus it has a simple and efficient VLSI layout in contrast to tree structures that are highly irregular and have complex routing networks. Furthermore, it can be easily and efficiently pipelined by inserting latches or flip-flops after every CSA or after every few rows.

3.3 Tree Structures

Before we discuss the architecture of multipliers with tree structures, it is imperative to understand two terms that are widely used when dealing with tree multipliers, namely, *counters* and *compressors*. The carry-save adder or the basic full adder is also referred to as a (3, 2) counter [6] (Figure 3.3 1). Higher level counters such as (7, 3) counters are usually realized from (3, 2) counters. Compressors are a special form of counters that are designed for use in tree topology implementations of multipliers. Compressors usually have two outputs (or powers of 2), ignoring the intercolumn carries. The most common compressor is the [4:2] compressor. It is a particular form of (5, 3) counter (Figure 3. 3 1) with one carry entering and one leaving the compressor column. Compressors used in tree structures result in more regular VLSI layouts for multiplier architectures in contrast to counters.

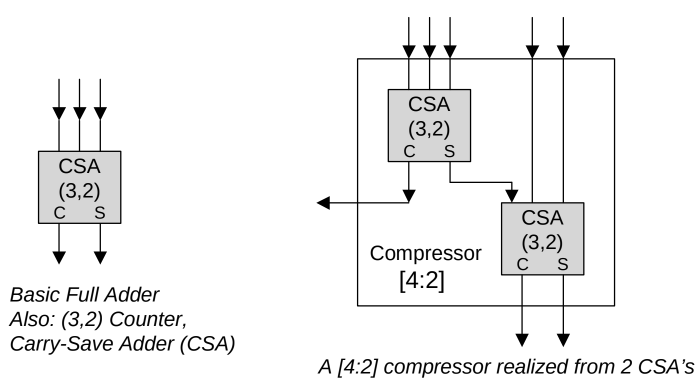


Figure 3.4 ***A (3, 2) Counter and a [4:2] Compressor***

A trees structure is a fast approach for summing partial products. Here, all the bits in a partial product bit column are summed concurrently using counters or compressors. In other words, the counters and compressors are connected mostly in parallel as shown in Figure 3.3 2. Although trees are faster than arrays, they both use the same number of counters and compressors to reduce the partial products (PPs) [6]. However, the difference in their topologies lies in the interconnections between the counters. Trees create a three dimensional structure that needs to be flattened for IC implementation. This flattening is achieved by placing the counters and compressors in a linear fashion.

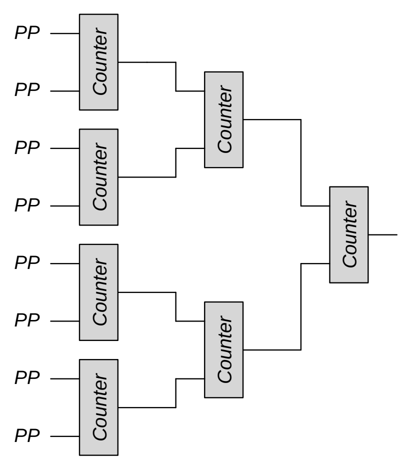


Figure 3.5 ***Parallel connection of counters in trees***

The two conventional tree architectures that are used for realizing high speed multipliers are the Wallace tree and Dadda tree structures. Figure 3.3 3 shows the dataflow through a 4 x 4 Wallace tree multiplier in the form of a dot diagram indicating the partial product summation at each stage. It may be recalled that each dot represents a '0' or a '1'. The sixteen partial products are spread across seven columns with 1,2,3,4,3,2,1 bits in each column respectively. In the first stage, 4 CSA's are required to reduce the partial products into seven columns with 1,3,2,3,2,1,1 bits in each column respectively. Again, in the second stage, 4 CSAs are required to further reduce the number of bits in seven columns to 2,2,2,2,1,1,1 bits respectively. Finally, a 4-bit carry propagate adder (or a ripple carry adder) is used to obtain the 8-bit product. Thus, this partial product reduction technique requires 8 CSAs in the first 2 stages and a 4-bit carry propagate adder (CPA) in the end. From Figure 3.3 3, it can be seen how every CSA or FA (Full adder) computes a sum at every stage and saves the carry to propagate it down the multiplier tree.

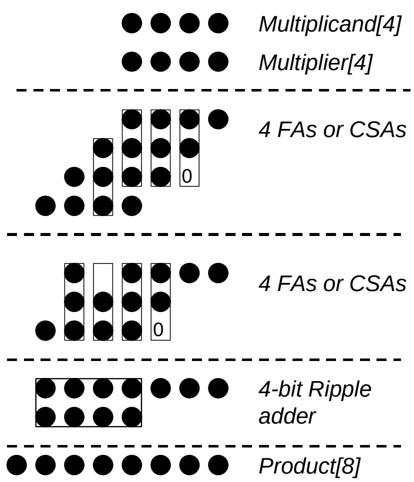


Figure 3.6 ***Dataflow through a 4 x 4 Wallace tree multiplier***

From Figure 3.3 3, it is apparent that the CSAs are used simultaneously in the Wallace tree structures to achieve a logarithmic depth reduction. In other words, in the case of tree multipliers, the latency is a logarithmic function of the word length, *O[log(n)]* whereas, in the case of array multipliers, the latency is a linear function of the word length of the multiplier, *O(n)*. This leads to fast multiplier implementations using tree structures rather than array topologies.

However, when the CSAs of the tree structure are laid out, the outputs of some counters may be inputs to nonadjacent counters. This leads to a highly irregular structure that makes its design and layout quite difficult. Regularity is an important issue in high speed arithmetic circuits' VLSI implementation. Regular structures may be constructed from building blocks that are laid out once and then tiled together [7]. The reuse of building blocks leads to efficient designs and reduced layout time. But the irregular nature of tree structures involving complex routing networks leads to highly inefficient designs. In order to reduce the irregularity, higher order compressors like [4:2] and [9:2] compressors are generally used for designing tree structures but with a compromise on increased area. As we have discussed in this section, the design of tree structures is a complex process involving intricate tradeoffs between performance, area and ease of IC implementation.

3.4 Multiplier Performance

The performance metrics of a multiplier is usually its latency and its throughput. Throughput is an important metric when it comes to high speed digital processing and other computation intensive circuits and latency is a more general purpose measure. It may be recalled that in a pipelined multiplier, latency is the number of clock cycles from the time when the multiplier and the multiplicand are input to the multiplier and when the product is available at its output. Throughput is a measure of the maximum speed at which a multiplier may be clocked. In other words, it gives the number of multiplications that a multiplier is capable of performing per second, which is usually of the order of billions of multiplication per second.

As we have seen, tree multipliers have a definite advantage over array multipliers for designing very high-speed multipliers. The main reason for this is due to the logarithmic reduction of partial products in tree multipliers in contrast to a linear reduction in array multipliers as shown in Figure 3.4 1. The [4:2] compressor reduces the partial products in logarithmic time, because of its 2-to-1 reduction ratio. A binary tree reduces N partial products using $\log_2(N/2)$ [4:2] compressor stages.

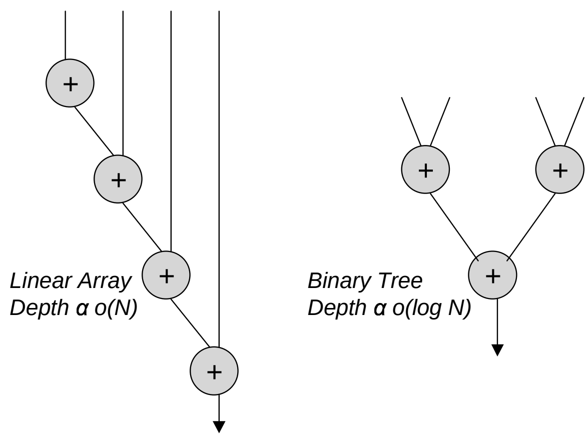


Figure 3.7 ***Array versus Tree Multipliers***

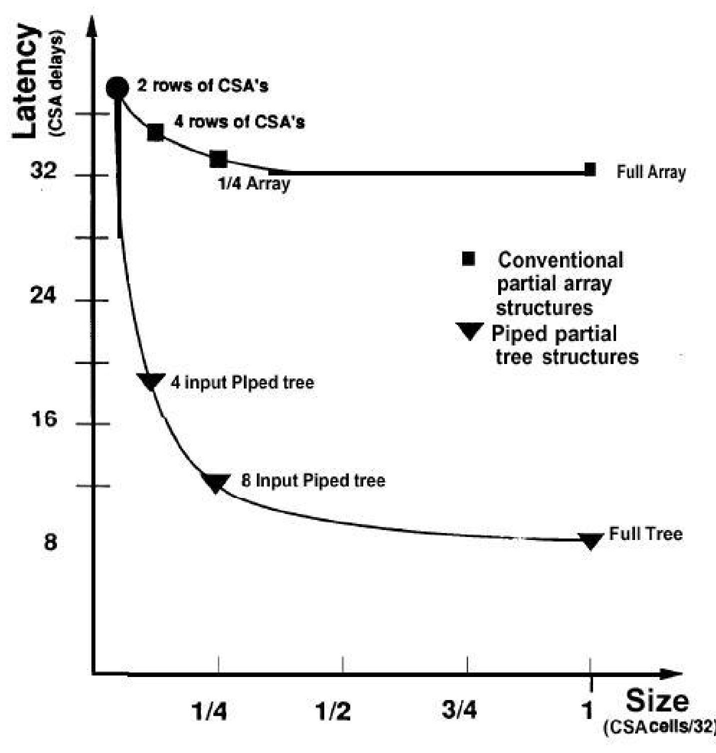


Figure 3.8 ***Comparison of architectures [7]***

Figure 3.4 2 shows the latency versus size tradeoff diagram for different sized partial, pipelined 4-2 trees over conventional linear arrays [7]. Partial tree structures are generally designed to save tree sizes by iteratively accumulating the partial products. This design technique was first used in the IBM 360 Model 91 floating point unit that uses partial 3-2 trees and a carry-save accumulator to iteratively accumulate the partial product [7]. The plot in Figure 3.4 2 shows a clear tradeoff between the size of the multiplier and its performance for the two techniques. Adding hardware to form larger partial linear arrays results in little performance improvements in contrast to adding hardware to make the partial trees into full trees, where the performance improves dramatically. It may be observed that the full tree multiplier is almost 4 times as fast as the full array multiplier with little differences in area. Thus, tree multipliers have a definitive advantage over array multipliers when it comes to realizing high-speed multiplier architecture designs.

3.5 Summary

In this chapter, we described the how the basic process of binary multiplication works. Following this, we learned about the two most common methods of multiplication, namely, array and tree multiplication. In tree multiplication, the latency is a logarithmic function of the partial products while it is a linear function of the partial product in array multiplier. This makes tree structures highly suitable for high-speed multiplier architectures. However, one of the major drawbacks of tree structures include highly irregular layouts because of the complex interconnect networks involved. Finally, we learned about the tradeoffs between latency and area for the array and tree structures.

Chapter 4

Architecture of Proposed High-Speed MCML Multipliers

In the previous chapter, we discussed the two most common methods of implementing hardware multipliers, namely, array and tree multiplication algorithms. We also saw the obvious advantage of tree algorithm over the array algorithm in terms of lower latency, thus making it a good choice for high-speed hardware multiplier implementations. Consequently, it was decided to adopt the tree structure for implementing our MCML multipliers, and draw upon the low-latency advantage from using tree structures and high-speed-operation advantage from the use of MCML logic style. Here, we propose three 8-bit MCML multiplier architectures: 4-2 tree multiplier design with ripple carry adders (RCAs), 3-2 tree multiplier design with ripple carry adders (RCAs) and 4-2 tree multiplier design with carry look-ahead adders (CLAs). The 4-2 tree architecture with carry look-ahead adders is used for comparison purposes; to prove that ripple carry adders are best suited for MCML high speed multiplier architectures. In the first three sections of this chapter, we deal with the three aforementioned architectures respectively and in the final section, we compare the three architectures.

4.1 Proposed 4-2 Tree MCML Multiplier architecture (using Ripple Carry Adder)

The architecture of the 8-bit 4-2 tree MCML multiplier may be understood from the top-level block diagram shown in Figure 4.1. The main components of the overall multiplier are: the 64-bit partial product generator to which the 8-bit multiplier and the 8-bit multiplicand are input, the binary tree slices, the deskew registers and the ripple carry adder. We shall now see in detail, the circuit design of each of these blocks used to build the overall multiplier. It may be recalled that to avoid clutter, the complementary signals of the MCML gates are not shown in the figures.

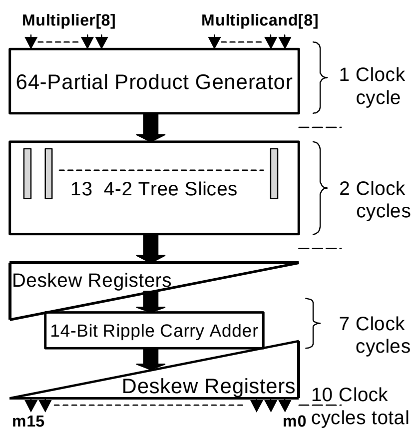


Figure 4.1 ***Architecture of an 8-bit MCML Multiplier using 4x2-Trees & Ripple Carry Adders***

4.1.1 The Partial Product Generator

The partial product generator is the first block of the multiplier to which the 8-bit multiplier, the multiplicand and their respective complements are input. At this juncture, the basic process of multiplication may be recalled from Figure 3.1. In this figure, each horizontal row of dots represents a single copy of the multiplicand which is multiplied by one bit of the multiplier resulting in a total of 64 partial products (8 rows of 8 partial products offset by one bit). The result of multiplying any one binary bit by another bit is either a zero or a one which is simply the logic ANDing of the two bits. Thus, the partial product generator block may be designed using 64 MCML AND/NAND gates as shown in Figure 4.2. The transistor-level circuit of the AND/NAND gate has been discussed in Section 2.3.

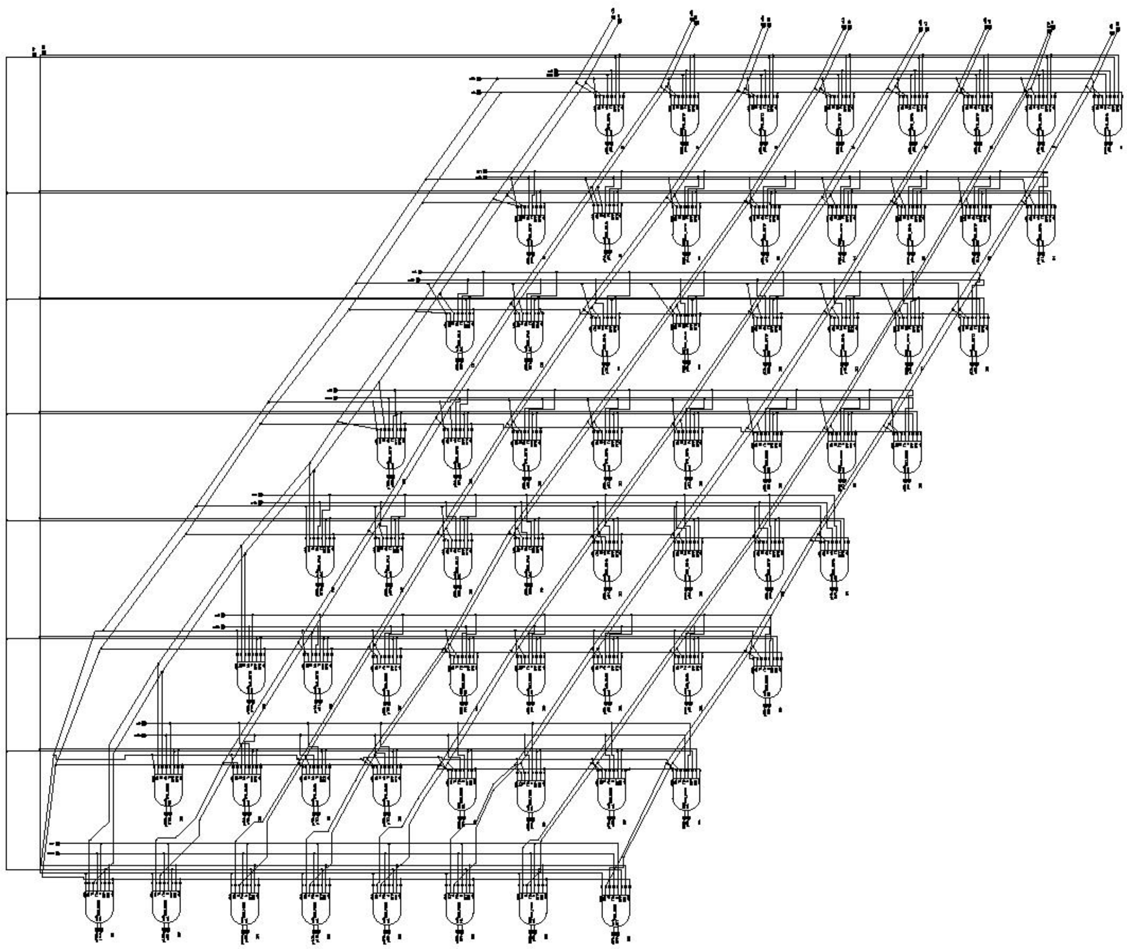


Figure 4.2 Snapshot of schematic of *64-Partial Product Generator (Cadence-Composer)*

The output of each AND/NAND gate is registered using a flip-flop. Hence, one delay stage or one clock cycle of overall latency of the multiplier can be attributed to the partial product generator.

4.1.2 The 4-2 Tree Slice

The rows of 4-2 tree slices form the next block of the multiplier architecture. It is so called because it uses [4:2] compressors for formation of the tree structure. The construction of a single 4-2 tree slice is shown in Figure 4.3.

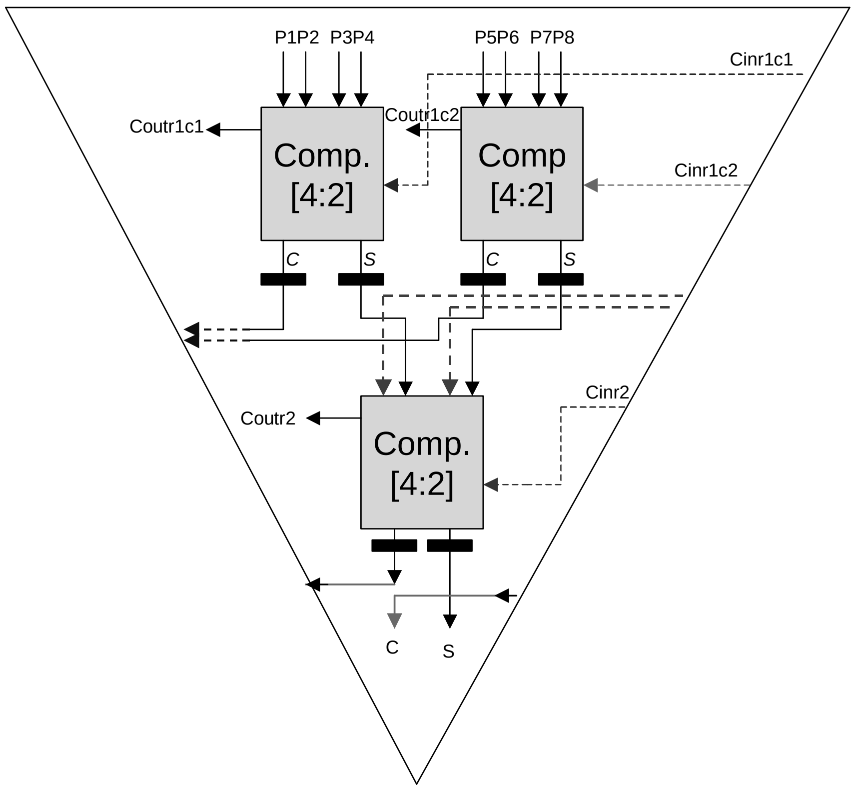


Figure 4.3 ***A single 4-2 tree slice with two stage pipelining.***

The 4-2 tree slice is designed with three [4:2] compressors, two in the first stage (row 1) and one in the second stage (row 2). The sum and carry outputs of each stage are pipelined with the help of flip-flops indicated in the figure as solid rectangular blocks. At this juncture, the circuit designs of the [4:2] compressor and the D Flip-flop may be recalled from Section 3.3 and Chapter 2, respectively. From the 64 partial products that are generated in the previous stage, one column of 8 partial products is input to the first row of the tree slice. The 8 partial products are reduced to 4 in the first stage and into 2 at the end of the second stage. In general, for 4-2 tree structures, it takes $\log_2[N/2]$ stages to reduce N partial products. Each tree slice contributes two delay stages towards the final delay of the multiplier.

Thirteen such 4-2 tree slices are used for the reduction of partial products generated in the first stage of the multiplier. A three dimensional view of the thirteen tree slices is shown in Figure 4.4.

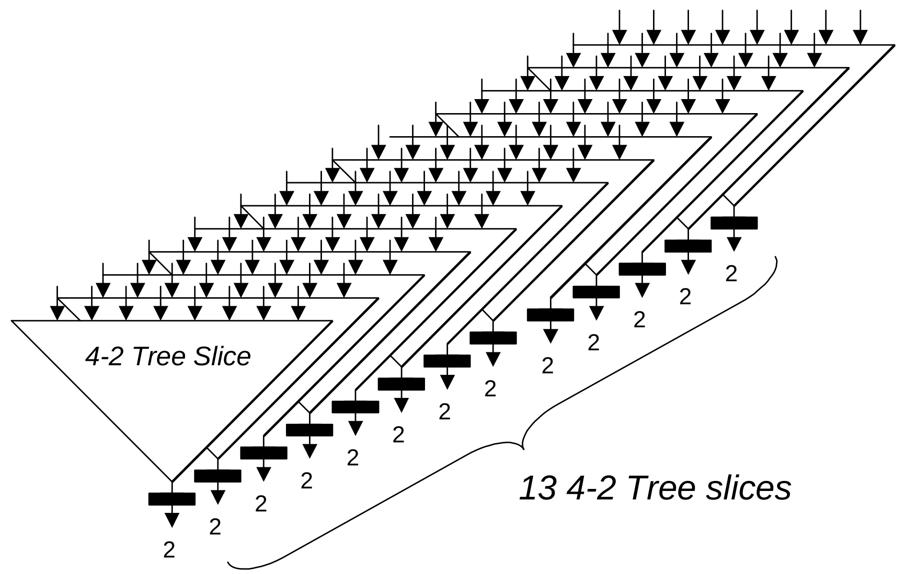


Figure 4.4 ***A three dimensional view of the 13 4-2 tree slices used for partial product reduction.***

The carry interconnections between two adjacent 4-2 trees may be easily understood with the help of Figure 4.5. Since the output carry of the [4:2] compressor has a weight of 2 as opposed to the inputs and the sum output that have a weight of 1, it has to be routed to the next bit slice. The carry in and carry out of the [4:2] compressors are routed to the corresponding [4:2] compressors in the next tree slice as shown in Figure 4.5. It may be noted that the 64 partial products are reduced to sum and carry outputs (indicated by a ‘2’ in Figure 4.4) and are available at the same clock cycle after three delay stages (one delay stage contributed by the partial product generator and two delay stages contributed by the 4-2 tree slice).

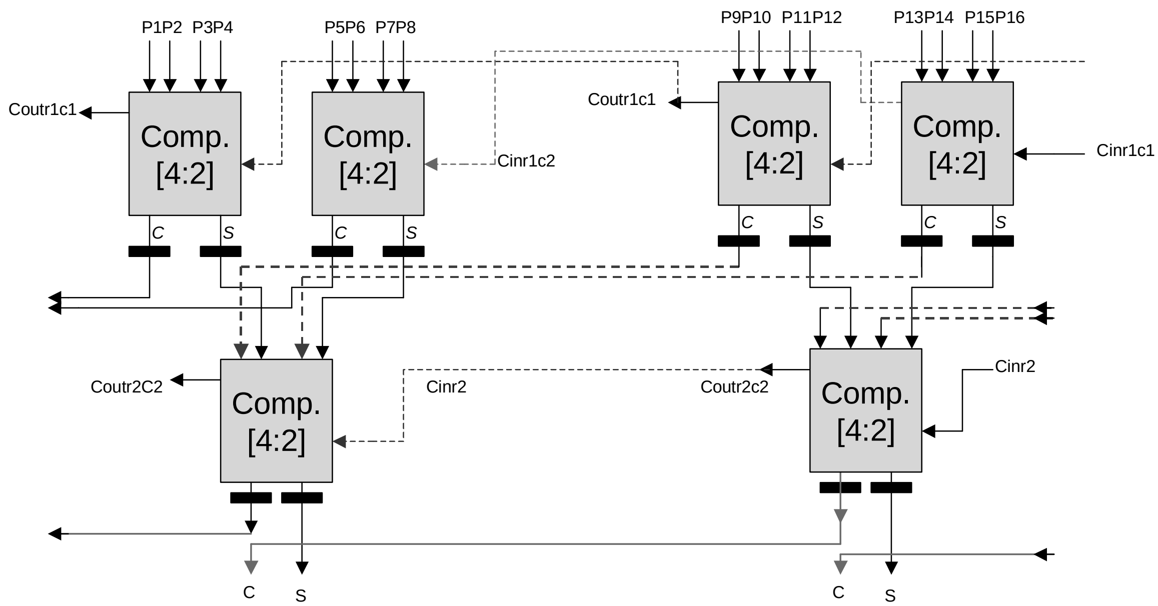


Figure 4.5 ***Two adjacent 4-2 tree slices showing carry interconnections.***

4.1.3 The 14-bit Ripple Carry Adder (RCA) and Deskew Registers

At the end of the previous stage, we had thirteen tree slices with sum and carry outputs arriving at the same time. A ripple carry adder (RCA) is required for parallel addition of these bits. The ripple carry adder is designed as shown in Figure 4.6. Two full adders, FA [3x2], are cascaded to construct a FA_2, where the carry out of the first full adder is given to the carry in of the second. A 14-bit RCA is then designed using 7 such FA_2’s and inserting one flip-flop between each of them. Thus, the ripple carry adder is two-stage pipelined or, in other words, one flip-flop is inserted between every two full adders.

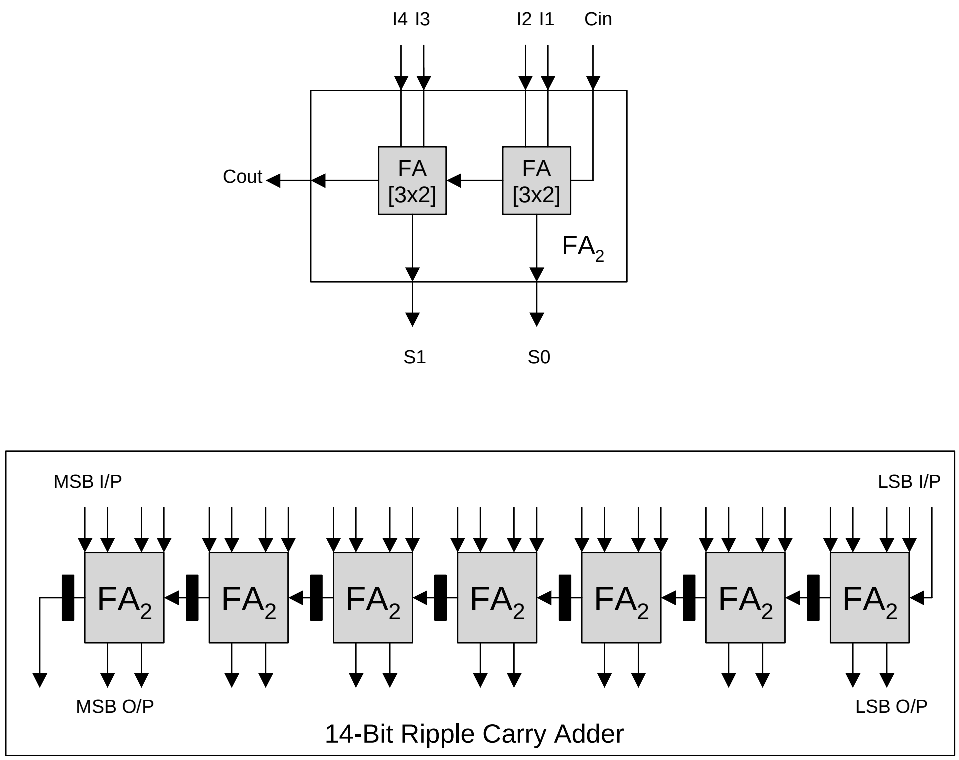


Figure 4.6 A 14-bit Ripple Carry Adder

Since all inputs arrive at the same clock cycle, they should be delayed successively and applied to the ripple carry adder. Hence, the first four inputs are delayed by one clock cycle, the next four by two and so on. Similarly, the first output is delayed by seven clock cycles, the second by six and so on. This delay balancing at the inputs and outputs of the ripple carry adder is done with the help of deskewing registers. Deskewing registers are simply columns of flip-flops used to insert appropriate delays at the input and output. This ensures that all the outputs of the ripple carry adder, which form a part of the final product, appear at the same clock cycle.

The complete dataflow during the process of multiplication may be summarized with the help of the dot-diagram shown in Figure 4.7. The 64 blue dots represent the partial products generated at the end of delay stage 1. Starting from the third column from left, the thirteen columns of 8 inputs are fed to thirteen 4-2 tree slices respectively. The inputs are fed in sets of four to the appropriate row and column of the tree as indicated in the figure. The grey and green dots represent the carry and sum outputs of row 1 of the tree that are input to row 2. The outputs of the tree are indicated by purple dots. The total delay at this stage is three. These outputs form the inputs of the 14-bit ripple carry adder that contributes seven delay stages to the total delay. The pink dots indicate the final 16-bit product that is obtained after ten delay stages. Hence the total latency of the multiplier is ten clock cycles.

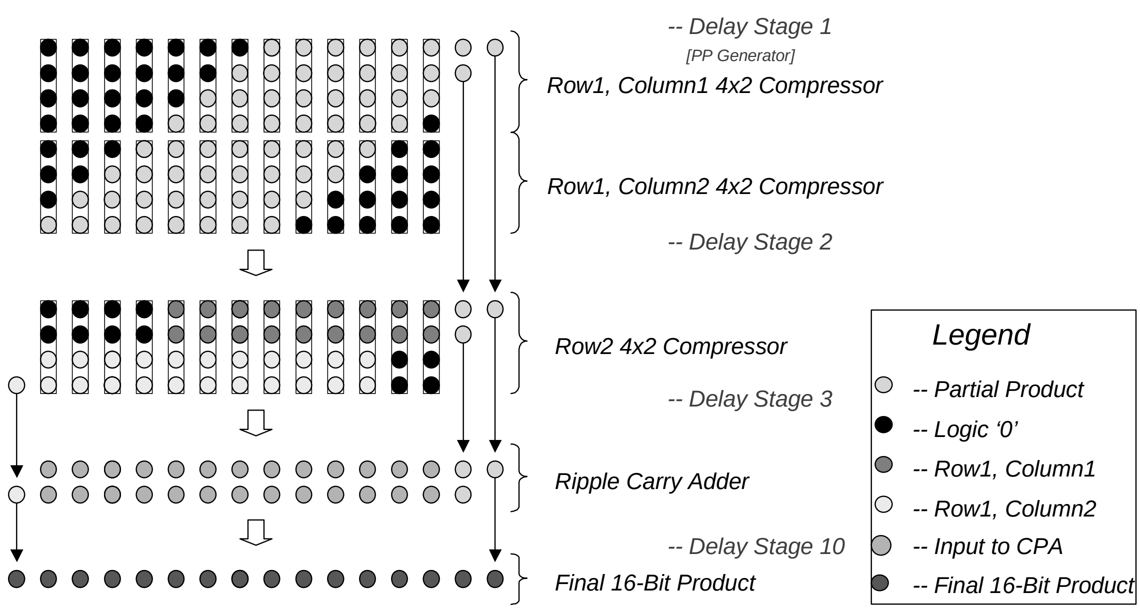


Figure 4.7 ***Dot-diagram indicating dataflow in multiplication process using 4-2 trees.***

A snapshot of the final top-level schematic (Cadence-Composer) of the 8x8 MCML 4-2 tree multiplier is shown in Figure 4.8.

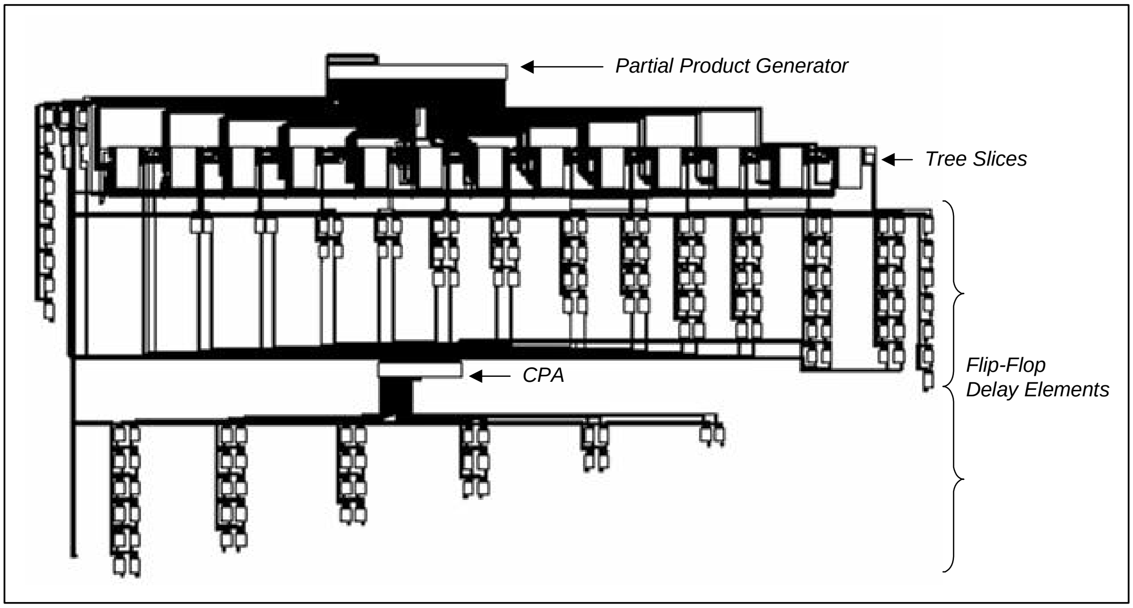


Figure 4.8 ***Top-level schematic (Cadence-Composer) of 8x8 MCML 4-2 tree multiplier***

4.2 Proposed 3-2 Tree MCML Multiplier Architecture

The 3-2-tree multiplier is designed on the same lines as the 4-2 tree multiplier, but using 3-2 compressors or full adders instead of [4:2] compressors and making appropriate changes to balance the delays. The architecture of the 3-2 tree multiplier indicating the delays at the different stages is shown in Figure 4.9. The block level diagram shown is essentially similar to that of the previous design with changes made to the design of the components found in the blocks. The partial product generator, however, is common to all three designs proposed in this work. The variation occurs in the design of the tree structure, the final parallel adder and the deskew registers.

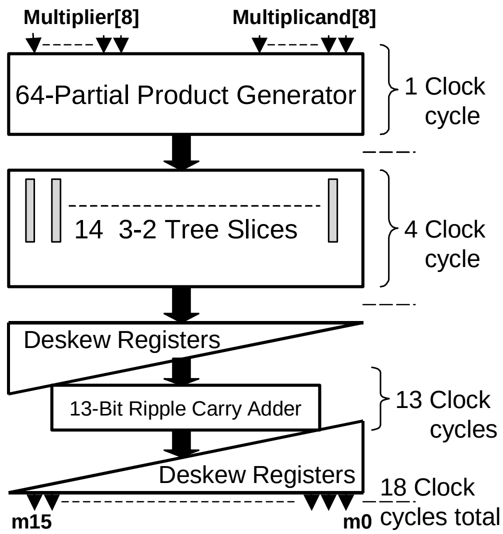


Figure 4.9 ***Architecture of an 8-bit MCML multiplier using 3-2 trees and Ripple Carry Adder***

4.2.1 The 3-2 Tree Slice

A snapshot of the schematic of a single 3-2 tree slice is shown in Figure 4.10. There are a total of seven full adders in the tree pipelined with the help of flip-flops that contribute four clock delays to the total multiplier delay. The eight partial products that are input to the 3-2 tree are reduced to two outputs (sum and carry) in four delay stages as shown in the figure. In general, in the case of 3-2 tree designs, it takes $\log_{1.5}[N/2]$ stages to reduce N partial products in to sum and carry outputs, unlike 4-2 tree designs where it takes $\log_2[N/2]$ stages for the same reduction. As we have seen in the design of the 4-2 trees, the carries produced in the intermediate stages are routed to the adjacent tree slices in a similar way in the case of 3-2 trees also.

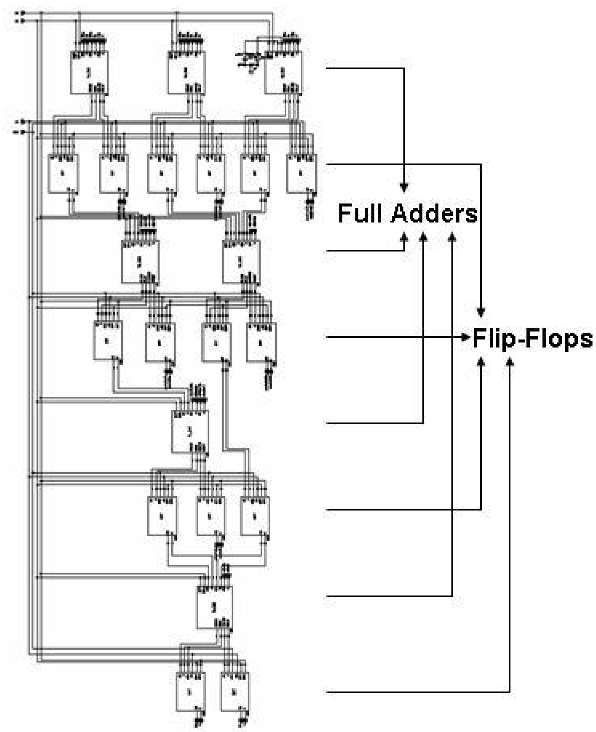


Figure 4.10 ***A snapshot of schematic (Cadence-Composer) of a 3-2 tree slice.***

14 such tree slices are used in the reduction of the 64 partial products generated in the previous stage of the multiplier architecture. The sum and carry outputs of each tree are available at the same time (of the clock) after five delay stages (one delay stage contributed by the partial product generator and four contributed by the 3-2 tree slice).

4.2.2 *The 13-bit Ripple Carry Adder (RCA) and Deskew Registers*

At the end of the previous stage, we had 14 tree slices with sum and carry outputs arriving at the same time. A ripple carry adder (RCA) is required for parallel addition of these bits. The ripple carry adder is designed as shown in Figure 4.11.

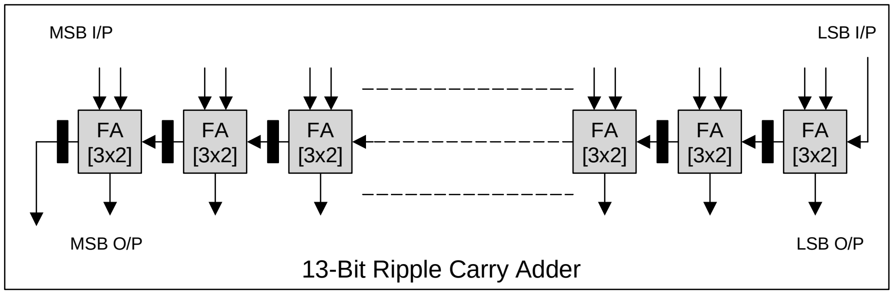


Figure 4.11 ***Block Diagram of a 13-bit Ripple Carry Adder.***

The ripple carry adder is designed using thirteen full adders with the carry out of one full adder routed to the carry in of the next full adder after being registered in a flip-flop. Thus the RCA is pipelined by inserting one flip-flop between every two full adders. Since all inputs arrive at the same clock cycle, they have to be delayed successively and fed to the ripple carry adder. The first two inputs (carry and sum) are delayed by one clock cycle, the next two by two clock cycles and so on. Similarly, the first output of the RCA is delayed by thirteen clock cycles, the second by twelve and so on. This delay balancing at the inputs and outputs of the ripple carry adder is done with the help of deskewing registers. Deskewing registers are columns of flip-flops used to insert appropriate delays at the input and output. This ensures that all the outputs of the ripple carry adder, that form a part of the final product, appear at the same time (of the clock), after thirteen clock cycles.

Thus, the total latency or the number of delay stages of the multiplier is eighteen clock cycles: one clock cycle is attributed to the partial product generator, four clock cycles attributed to the 3-2 tree slice and thirteen attributed to the ripple carry adder.

4.3 Proposed 4-2 Tree MCML Multiplier architecture (using Carry Look-ahead Adder)

The architecture of the 8-bit 4-2 tree MCML multiplier that uses carry look-ahead adders may be understood from the top-level block diagram shown in Figure 4.12. The architecture of this multiplier is similar to that of one proposed in Section 4.1 but makes use of Carry Look-ahead Adders (CLAs) as opposed to Ripple Carry Adders for the parallel addition of the sum and carry bits generated at the end of the tree stages.

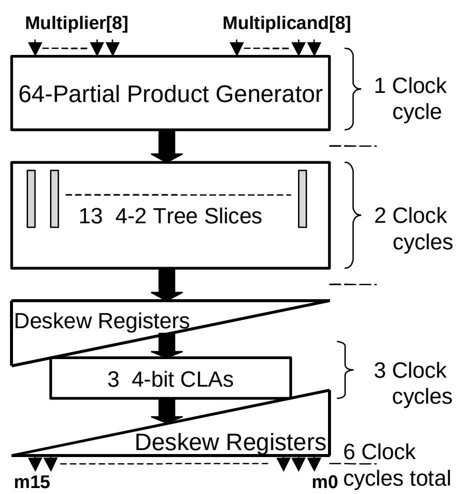


Figure 4.12 ***Architecture of 8-bit MCML multiplier using 4-2 trees and Carry Look-ahead Adder***

The 64-bit partial product generator block is same as the one used for the architectures proposed in Sections 4.1 and 4.2. The design of the 13 4-2-tree-slices is implemented in the same lines as of the 4-2-tree slices used in the multiplier architecture proposed in Section 4.1. The only difference between the 8-bit multiplier architecture proposed in Section 4.1 and the one proposed in this section is the usage of pipelined 4-bit Carry Look-ahead Adders (CLAs) instead of Ripple Carry Adders used in former. The parallel adder is pipelined by inserting one flip-flop between every 4-bit CLA. Deskew registers similar to the ones used in Section 4.1 and 4.2 are used to attain the same function of balancing the delays of the input and output of the parallel adder in order to make available the 16-bit product at the same clock cycle.

4.3.1 *The 4-bit Carry Look-ahead Adder (CLA)*

The linear growth of adder carry-delay with the input word size for an n-bit adder may be improved by calculating the carries to each stage in parallel using a Carry Look-ahead Adder [23]. The carry of the i^{th} stage, C_i, may be expressed as,

$$C_i = G_i + P_i \times C_{i-1}$$

where,

the generate signal G is $G_i = A_i \times B_i$ and

the propagate signal P is $P_i = A_i + B_i$

Expanding this yields,

$$C_i = G_i + P_i \times G_{i-1} + P_i \times P_{i-1} \times G_{i-2} + \ldots + P_i \ldots P_1 C_0$$

The sum S_i is generated by $S_i = C_{i-1} \oplus A_i \oplus B_i$

The size and fan-in of the gates needed to implement this carry look-ahead scheme grows too fast. As a result, the number of stages of look-ahead is usually limited to about four. Figure 4.13 shows the block diagram of a carry look-ahead adder. The PG generation and SUM generation circuits surround a carry-generate block. The PC generation block consists of an AND gate and an OR gate for each pair of inputs. The SUM block is an XOR gate that takes the P input from the PG block and the carry from the carry generator block and produces the SUM output.

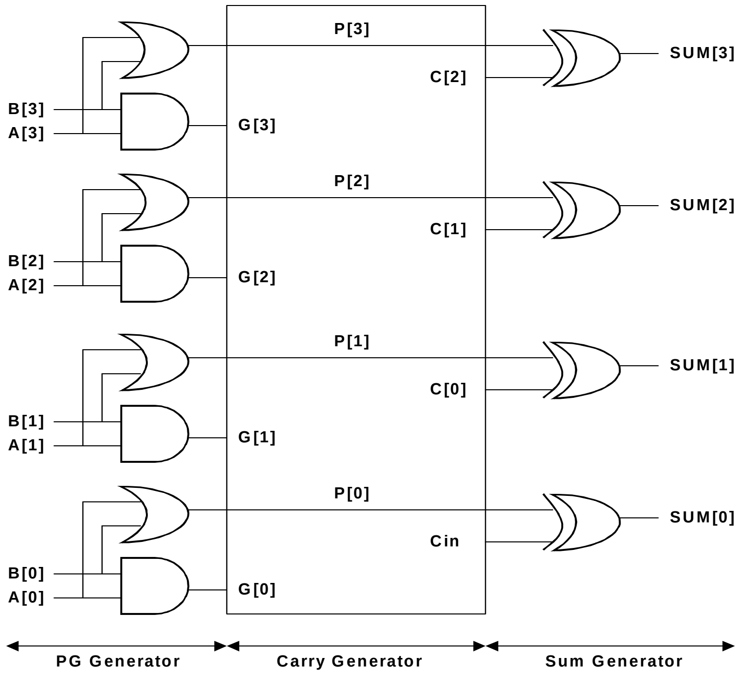


Figure 4.13 ***Block diagram of a 4-bit Carry Look-ahead Adder***

The transistor level circuit diagram of the AND/NAND and XOR MCML gates have already been described in Section 2.3, while the circuit diagram of the MCML OR gate and the carry generator gate ($C_i = G_i + P_i \times C_{i-1}$) is shown in Figure 4.14.

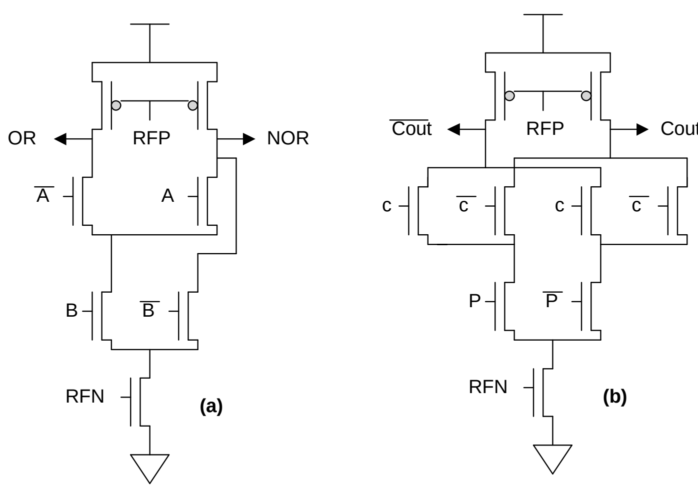


Figure 4.14 ***(a) MCML OR Gate (b) Carry generator gate ($C_i = G_i + P_i \times C_{i-1}$)***

The parallel adder section of the multiplier is pipelined by inserting one flip-flop between every 4-bit carry look-ahead adder. Since all inputs to the parallel adder arrive at the same time, they should be delayed successively and applied to the ripple carry adder. The first four inputs (carry and sum) are delayed by one clock cycle, the next two by two clock cycles and so on. Similarly, the first output of the RCA is delayed by three clock cycles, the second by two and so on. This delay balancing at the inputs and outputs of the CLA is done with the help of deskewing registers. Deskewing registers are nothing but columns of flip-flops used to insert appropriate delays at the input and output. This ensures that all the outputs of the pipelined CLA arrive at the same clock cycle.

Thus, the total latency or the number of delay stages in the multiplier is six clock cycles: one clock cycle is attributed to the partial product generator, two clock cycles attributed to the 4-2 tree slice and three attributed to the pipelined CLA.

4.4 Summary

In this chapter, we proposed three 8-bit MCML multiplier architectures. The three 8-bit MCML multiplier architectures are: 4-2 tree multiplier design with ripple carry adders (RCAs), 3-2 tree multiplier design with ripple carry adders (RCAs) and 4-2 tree multiplier design with carry look-ahead adders (CLAs).

Chapter 5

Optimization and Simulation Results

In this thesis, we have seen the design of several MCML gates and the architecture of three 8-bit multipliers designed using them, thus far. This chapter provides an insight in to the simulation setup; the optimization technique adopted for maximizing the performance of these gates and also provides the simulation results/data of all the gates and the overall multiplier architectures.

5.1 Experimental Simulation Setup

Before we proceed to explain the technique for optimization of the various performance parameters, it is appropriate to establish a simulation setup and enable a common ground for evaluation of all the gates designed thus far. The major part of the design entry was done at the transistor schematic level. The layout of few basic gates where also done in an attempt at estimating the area of the overall multiplier.

Designs were entered using the Cadence Composer tool for transistor level schematics and Cadence Virtuoso for layouts targeting TSMC 0.18 μm CMOS technology. The netlists were extracted from the schematics using Affirma Analog Artist and simulated using HSPICE.

The HSPICE simulations were carried out in an effort to characterize all the basic gates discussed in Chapter 2 for delay and power consumption. For delay modeling, three kinds of delays were measured: the average delay, Fan4 delay and the actual delay. The average delay measurements were carried out with sets of input patterns that caused maximum delay in the critical path of the circuit. The Fan4 delay is simply the propagation delay incurred in the gate with a fan-out of four, which is when the output of the gate drives four similar gates. The actual delay is the propagation delay incurred in the gate when it is simulated with load conditions that would be present if it were used in the architecture of our proposed MCML multipliers. The realistic delay is the most accurate and relevant value of the delay of our gates that are applicable to the proposed multiplier architectures. It should be noted that the delay of the flip-flop is determined as the sum of its setup time and clock-to-q delays. It can further be observed from Figure 5.1 that the measurement of delay for MCML gates is done from the time of input crossover of complementary inputs to the time of crossover of the complementary outputs. This is in contrast to conventional CMOS gates where the delay measurement is between 50% transition points of the input and output waveforms.

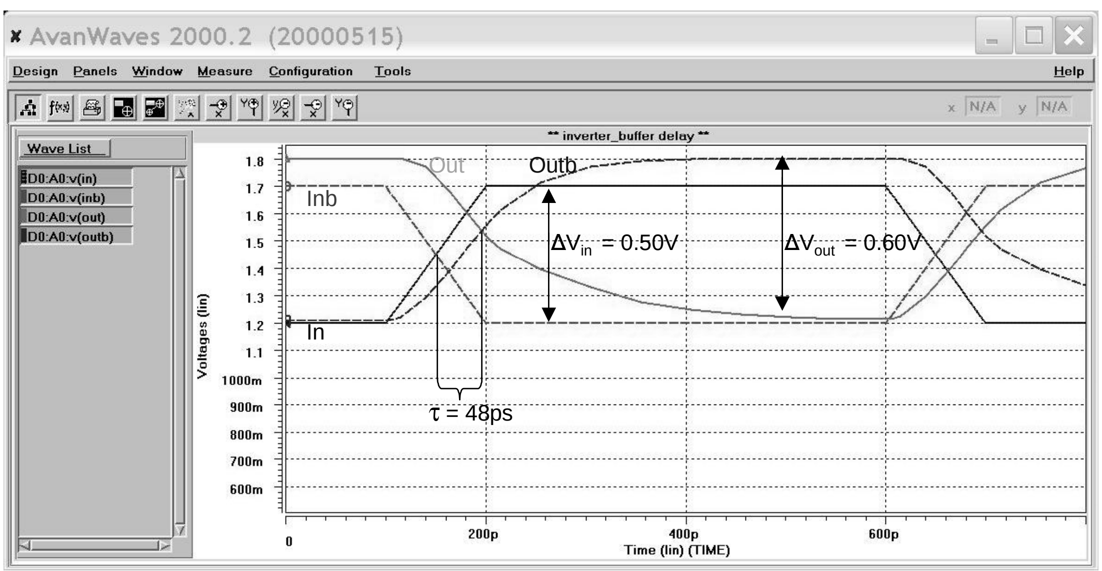


Figure 5.1 ***MCML Inverter/Buffer simulation waveform showing delay measurements.***

To find the worst-case delays of MCML gates, exhaustive simulations were performed for cells with fewer than four inputs; for gates with four or more inputs, the number of patterns needed in the sequence used for exhaustive simulations become excessive. Note that for finding delays of an N-input gate, it is not adequate to simply apply all the 2N possible inputs in a random order. The reason is that the delay depends on the exact transition that occurs, instead of simply the final input state. For instance, for a full adder with three inputs (A, B, Carry-in), the input transition from 000 to 111 and the transition 110 to 111 may cause different delays, even though in both cases, the input settles at 111. Hence, whenever exhaustive simulations are needed, we may have to apply a sequence containing all possible input transitions in sequence instead of merely applying all possible inputs.

For N inputs, there are 2^N input patterns, and every input pattern could transition to any of the other (2^N-1) input patterns, resulting in a total of $2^N(2^N-1)$ possible transitions. Hence a sequence of at least $[2^N(2^N-1)+1]$ patterns should be generated for exhaustive analysis.

The other important performance parameter measured was the power dissipation of the MCML gates. The average power consumption is taken for the test vectors used, which is the same as the test vectors used for delay estimation.

Since the chief goal of our research was to achieve the maximum speed by minimizing the delay so as to maximize the multiplier throughput, the area of the design was compromised. The value of the area of the overall multiplier is estimated from the layout of the basic gates in an effort to deduce an understanding of the tradeoffs involved among speed, area and power dissipation in the proposed multiplier architectures.

5.2 Gate optimization technique

The major motivation behind optimizing the MCML gates is to attain high operation speeds. In this endeavor, the area and power dissipation of the gates is compromised. The expressions for delay and power dissipation in a general MCML gate may be recalled from Table 2.1 of Chapter 2. It is stated here again for convenience.

Propagation Delay $\boldsymbol{\tau}_{\mathbf{MCML}} = \dfrac{C_L \times \Delta V \times N}{I_{SC}}$ --- (E 5.1)

Power Dissipation $\mathbf{PD_{MCML}} = V_{DD} \times I_{SC} \times N$ --- (E 5.2)

Since the delay depends linearly on ΔV and is independent of supply $V_{DD,}$ it can be minimized by determining an acceptable and a low voltage swing ΔV, while simultaneously minimizing power dissipation by reducing the supply voltage V_{DD}, which has little effect on the delay. However, it is apparent that in order to minimize the propagation delay, the driving current though the gate I_{SC} should be maximized while minimizing the output voltage swing and maintaining a shallow logic depth N. It is also obvious that an increase in the source current I_{SC} leads to an increase in power dissipation, that has to be compromised for high-speed operation.

In order to optimize the library gates to operate in the gigahertz range, the following approach was adopted. The maximum current through the logic transistors is determined through several simulation runs and was found to be in the order of a few hundred microamperes. Fixing the bias voltages RFP and RFN and using the current value from the previous simulations, the dimensions of the load transistors and the current source transistor are determined by further simulations. Optimizing the PMOS load transistor is one of the most crucial and challenging tasks in an MCML gate design, involving fine tradeoffs between non-linearity and signal strengths. Whereas a high W/L ratio improves the delay by decreasing the resistance, it increases the non-linearity of the resistance leading to degradation in output swings.

For all the gates in our library, the following design parameters were set the values indicated: the supply voltage (VDD) was set at 1.8 V, which is the maximum for the 0.18 μm technology, the input swing (ΔV) was set at 0.5 V with a low swing of 1.2 V and a high swing of 1.7 V and the load transistor bias voltage (RFP) and the current source transistor bias voltage (RFN) were set at 0.3 V and 1.1 V, in order to work the load and current source transistors in linear and saturation regions respectively.

5.3 Simulation Results

5.3.1 MCML Gate Library

The performance metrics of the optimized library of MCML gates are shown in Table 5.1. It may be observed that the delays of all the gates are on an average between 50 and 100 ps, which are expected values for the MCML logic style. The actual delay that is determined under the load conditions similar to when it is used in a multiplier is as expected, greater than the average delay (when the gate is unloaded) and lesser than Fan4 delay (when the gate is loaded with four similar gates).

Table 5.1 ***MCML gate library performance metrics.***

Gate \ Performance Metric	Propagation Delay (ps)			Average Power (mW)	Area (μm^2)
	Average	Fan4	Actual		
NAND/AND	53.96	104.85	62.93	0.446	86.7
Master-Slave DFF	49	74	55	0.260	206.34
XOR3 ('Sum' of FA)	57	113	66.4	0.445	147.07
Maj. Function ('Carry' of FA)	76	138	88	0.445	147.07
[4:2] Compressor	148	238	159	1.498	558.09

5.3.2 8-bit MCML Multiplier Architectures

The performance metrics of the three 8-bit MCML multiplier architectures proposed in this work is shown in Table 5.2. Architecture I achieves the highest throughput among the three multipliers, but it incurs the largest latency (in terms of clock cycle count as well as absolute delay) and area. It operates with a throughput of 4.76 GHz; latency of 18 clock cycles and dissipates 261 mW.

Table 5.2 ***Performance metrics of MCML multiplier architectures***

Multiplier Architectures \ Performance Metrics	Throughput		Latency		Power (mW)	Area Estimate (mm^2)
	Clock Period (ns)	Billions of Multiplications per second	Clock Cycles	Absolute Delay (ns)		
[I] 3-2-Tree MCML Multiplier with RCAs	0.21	4.76	18	3.78	261.20	0.1490
[II] 4-2-Tree MCML Multiplier with RCAs	0.30	3.33	10	3.00	124.47	0.0548
[III] 4-2-Tree MCML Multiplier with CLAs	0.50	2.00	6	3.00	105.20	0.0389

The high throughput of Architecture I is due to the high degree of pipelining achieved, while the high area is due to the large number of flip-flops used for delay balancing. In order to achieve the high degree of pipelining, one flip-flop is inserted between every two compressor stages, which, in this case, is a full adder or the (3x2) counter. This also leads to the undesirable effect of increase in the number of latency clock cycle count. Architecture II on the other hand is more efficient than architecture I when it comes to latency clock cycle count as well as the absolute delay but looses out on the throughput. This is because; the basic compressor cell in this architecture is the [4:2] compressor that has a higher delay in contrast to the (3x2) counter which is the basic compressor cell in architecture I. Figure 5.2 brings forth obvious tradeoffs between throughput and latency in high-speed pipelined multiplier architectures proposed in this work.

Architecture III on the other hand, has the least latency clock cycle count because of its lower degree of pipelining, but it has a low throughput of just 2 GHz. This is because the longer critical path of the pipelined CLA decreases the throughput to under that of the pipelined RCA architectures (architectures I and II). In other words, the CLA is more difficult to pipeline than the RCA, and the delay of a 4-bit CLA exceeds that of a 2-bit adder; hence, since the 4-bit CLA is situated in the critical path, the maximum clock speed decreases.

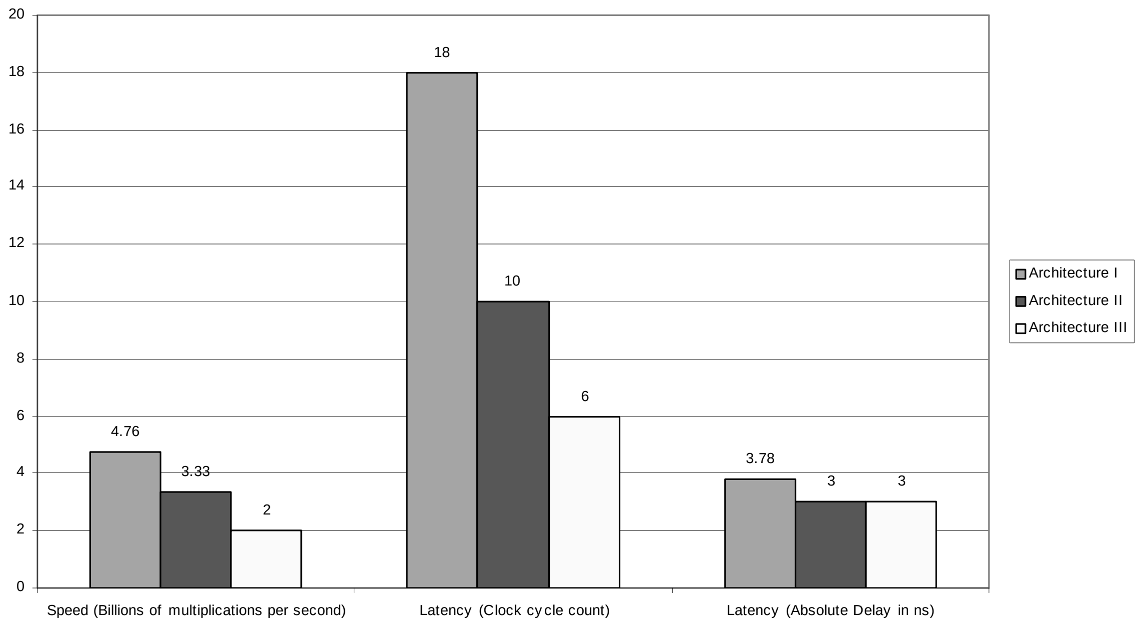


Figure 5.2 ***Graph showing tradeoff between multiplier throughput and latency***

Figure 5.3 shows the power consumption and die area estimates of the three proposed multiplier architectures. We observe that architecture I dissipates more power than architecture II, which, in turns, dissipates more power than architecture III. This is in accordance with the expected results because architecture I has more pipeline stages than architecture II, which, in turn, has more pipeline stages than architecture III and this results in decreased overall signal switching in the architectures in that order.

The area estimates of the complete multiplier are calculated manually from the layout area of the individual gates. We observe from Figure 5.3 that architecture I consumes the largest die area and architecture III consumes the least. This is again due to the fact that architecture I uses the highest number of compressor cells (full adders) for partial product reduction and deskew registers for delay balancing, followed by architecture II and architecture III.

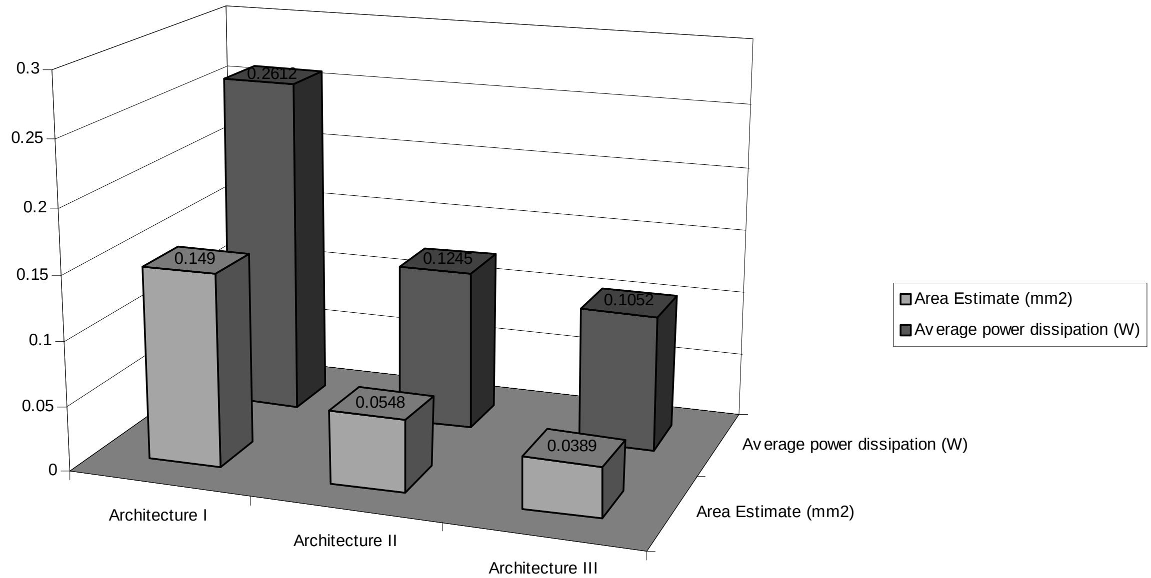


Figure 5.3 ***Graph showing area and power estimates for MCML multiplier architectures.***

5.4 Comparison of performance of multiplier architectures

Comparison of various multiplier architectures with the proposed designs is not an easy task. One of the reasons is that multipliers are often designed with different performance goals; some multipliers are optimized for throughput, compromising on latency, while some others are optimized for latency, compromising on throughput. We selected ten designs for comparison with the proposed architectures. Since the goal of our research is to attain high throughput, it was decided to base our comparison on this performance metric alone without considering die area and power dissipation.

In order to make the comparison a fair one, the throughput of the multipliers found in literature were normalized using estimated FO4 delays. For this purpose, the following equation was used.

$$\text{Normalized throughput} = \text{Estimated Throughput} \times \frac{FO4_{Estimated}}{FO4_{MCML}}$$

Table 5.3 ***Comparison of multiplier architectures.***

	Year	$T_{CLK,MIN}$ (ns)	Estimated FO4 Delay	Feature Size (μm)	Word Length	Estimated Throughput (GHz)	Normalized Throughput (GHz)
[9]	2000	1.6000	216	0.6	8	0.625	3.000
[10]	1995	19.9940	1300	1.2	16	0.500	1.444
[11]	1990	3.7998	180	0.5	16	0.263	1.052
[12]	1993	10.9980	450	1	16	0.910	0.910
[13]	1996	10.4985	450	1	24	0.950	0.950
[14]	2001	2.2504	126	0.35	32	0.444	1.243
[15]	2001	1.5801	65	0.18	54	0.633	0.914
[16]	1998	2.5002	90	0.25	54	0.400	0.800
[17]	2002	0.6499	41.4	0.13	54	1.538	1.414
[18]	2002	0.9998	41.4	0.13	59	1.000	0.920
Proposed Architecture [I]	2003	0.2100	45	0.18	8	4.760	4.760
Proposed Architecture [2]	2003	0.3000	45	0.18	8	3.300	3.300
Proposed Architecture [3]	2003	0.5000	45	0.18	8	2.000	2.000

Table 5.3 shows the comparison of throughputs of various multiplier architectures. It may be observed that proposed architecture [1] and [2] have throughputs higher than the designs found in contemporary literature while proposed architecture [3] does not lag far behind with a throughput of 2 GHz.

5.5 Summary

In this chapter, we discussed the simulation setup; the optimization technique adopted for maximizing the performance of the library of MCML gates and also provided the simulation results/data of all the gates and the overall multiplier architectures. The 3-2-tree design operates with a maximum throughput of 4.76 GHz (4.76 Billion multiplications per second) and a latency of 3.78 ns. The 4-2-tree architecture that uses ripple carry adder operates with a throughput of 3.3 GHz and a latency of 3 ns and the 4-2-tree architecture that uses carry look-ahead adder operates with a throughput of 2 GHz and a latency of 3 ns.

Chapter 6

Conclusions

In this work, we proposed three multiplier architectures capable of operating in the gigahertz-range, based on MOS Current Mode Logic (MCML). The ability to build logic gates that function at very high speeds, dissipating relatively less power compared to other logic styles including CMOS, makes MCML a promising technique for designing gigahertz-range arithmetic circuits. Traditionally multiplier architectures fall in into one of the following two categories, viz. array multipliers and tree multipliers. Whereas in array multipliers, the latency is a linear function of the word length of the multiplier, *O(n)*, the latency is a logarithmic function of the word length, *O[log(n)]* for tree multipliers. Hence, a tree structure requires fewer numbers of stages for partial product reduction compared to an array structure and is more suitable for high-speed multiplier designs. We investigate three different 8-bit MCML binary-tree multiplier architectures in TSMC 0.18 μm technology and compare their performance in terms of latency, throughput (number of multiplications per second) and power consumption.

A small library of MCML gates consisting of NAND/AND, XOR/XNOR, 3x2 counter (Full Adder), [4:2] Compressor, and Flip-flop were designed and optimized for high-speed operation. These gates were used to build the three different types of multiplier architectures investigated in this thesis.

The three 8-bit MCML multiplier architectures proposed are as follows: 3-2-tree architecture with ripple carry adder (Architecture I), 4-2-tree design with ripple carry adder (Architecture II) and 4-2-tree design with carry look-ahead adders (Architecture

III). We design and simulate our multiplier architectures in TSMC 0.18 μm CMOS technology. According to our simulation results, architecture I operates with a maximum throughput of 4.76 GHz (4.76 Billion multiplications per second) and a latency of 3.78 ns. Architecture II has a maximum throughput of 3.3 GHz and a latency of 3 ns and Architecture III that uses carry look-ahead adder has a maximum throughput of 2 GHz and a latency of 3 ns. Architecture I achieves the highest throughput among the three multipliers, but it incurs the largest area and latency, in terms of clock cycle count as well as absolute delay. Although it is difficult to compare the speed of our multiplier with existing ones due to the use of different technology and different optimization goals, we believe our multipliers are among the fastest found in contemporary literature.

Bibliography

1. K.M. Chu and D.I. Pulfrey, "Design and Procedures for Differential Cascode Voltage Switch Circuits", IEEE Journal of Solid-State Circuits, vol. Sc-21, No. 6, December 1986.
2. J. Musicer, "An Analysis of MOS Current Mode Logic for Low Power and High Performance Digital Logic" A research project at the University of California, Berkeley, 2000.
3. M.Mizuno, M.Yamashina et al. "A GHz MOS Adaptive Pipeline Technique Using MOS Current-Mode Logic" IEEE Journal of Solid-State Circuits Vol. 31, No. 6, June 1996.
4. J.Musicer and J.Rabaey, "MOS Current Mode Logic for Low Power, Low Noise CORDIC Computation in Mixed-Signal Environments", Proc., ISLPED 2000, pp. 102-107, 2000
5. J Rabaey, A Chandrakasan, B Nikolic, "Digital Integrated Circuits – A Design Perspective, 2nd Ed.", New Jersey : Prentice-Hall Inc, c2003
6. MJ Flynn and SF Oberman, "Advanced Arithmetic Computer Design", New York : J. Wiley, c2001.
7. Santoro M R, 'Design and Clocking of VLSI Multipliers', Technical Report No. CSL-TR-89-397, October 1989, Computer Systems Laboratory, Departments of Electrical Engineering and Computer Science, Stanford University, Stanford, CA 94305
8. B Parhami, "Computer Arithmetic: Algorithms and Hardware Designs", New York : Oxford University Press, c2000
9. J S. Wang, P H Yang, and D Sheng, "Design of a 3-V 300-MHz Low-Power 8-b×8-b Pipelined Multiplier Using Pulse-Triggered TSPC Flip-Flops", IEEE Journal of Solid-State Circuits, vol. 35 issue 4, Apr 2000, pp. 583 –592
10. W Amendola Jr., H R Srinivas, and K K Parhi, "A 16-bit×16-bit 1.2 μ CMOS Multiplier with Low Latency Vector Merging", Proceedings of the 8th International Conference on VLSI Design, Jan 1995, pp. 398 –402

11. K Yano, T Yamanaka, T Nishida, M Saito, K. Shimohigashi, and A. Shimizu, "A 3.8-ns CMOS 16×16-b Multiplier Using Complementary Pass-Transistor Logic", IEEE Journal of Solid-State Circuits, vol. 25 issue 2, Apr 1990, pp. 388 –395
12. M C Park, B W Lee, G M Kim, and D H Kim, "Compact and Fast Multiplier Using Dual Array Tree Structure", Proceedings of IEEE International Symposium on Circuits and Systems, 3-6 May 1993, vol. 3, pp. 1817 –1820
13. V G Oklobdžija, D Villeger, and S S Liu, "A Method for Speed Optimized Partial Product Reduction and Generation of Fast Parallel Multipliers Using An Algorithmic Approach", IEEE Transactions on Computers, vol. 45 issue 3, Mar 1996, pp. 294 –306
14. M Olivieri, "Design of Synchronous and Asynchronous Variable-Latency Pipelined Multipliers", IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 9 issue 2, Apr 2001, pp. 365 –376
15. N Itoh, Y Naemura, H Makino, Y Nakase, T Yoshihara, Y Horiba, "A 600-MHz 54×54-bit Multiplier with Rectangular-Styled Wallace Tree", IEEE Journal of Solid-State Circuits, vol. 36 issue 2, Feb 2001, pp. 249 –257
16. Y Hagihara, S Inui, A Yoshikawa, S Nakazato, S Iriki, R Ikeda, Y Shibue, T Inaba, M Kagamihara, and M Yamashina, "A 2.7 ns 0.25 μm CMOS 54×54 b Multiplier", Digest of Technical Papers, 45th IEEE International Solid-State Circuits Conference, 5-7 Feb 1998, pp. 296 –297
17. Intel Corp, *Intel Pentium 4 Optimization Guide*, Intel Corp, Santa Clara, CA, 2002
18. R Rogenmoser, L O'Donnell, and S Nishimoto, "A Dual-issue Floating-Point Coprocessor with SIMD Architecture and Fast 3D Functions", Digest of Technical Papers, 2002 IEEE International Solid-State Circuits Conference, 2002, vol. 1, pp. 414 -415
19. L Wirbel, "ISSCC: Developers Creep Towards 40-Gbits/s Chip", EE Times, February 10 2000
20. "High Speed CMOS Integrated Circuits: 20 Gbits/s and beyond", ResearchTrends, A publication by Corporate Research (Infineon Technologies AG), Special Edition / October 2002.

21. M Yamashina and H Yamada, "An MOS Current Mode Logic (MCML) Circuit for Low-Power GHz Processors", NEC Res. & Develop., Vol 36, No. 1 January 1995
22. M H Anis and M I Elmasry, "Power Reduction Via An MTCMOS Implementation of MOS Current Mode Logic", ASIC/SOC Conference, 2002. 15th Annual IEEE International, 25-28 Sept. 2002, Page(s): 193 –197
23. N H Weste and K Eshragian, "Principles of CMOS VLSI Design – A System Perspective", Reading, Mass. : Addison-Wesley, c1985

The vita has been removed from the scanned document