DESIGN EXPLORATIONS OF VLSI ARITHMETIC CIRCUITS

THESIS

Submitted in partial fulfillment

of the requirements for the degree of

DOCTOR OF PHILOSOPHY

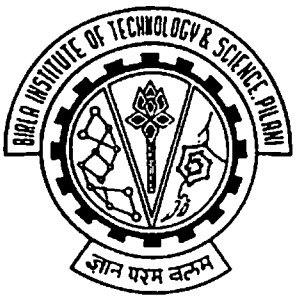
By

ANU GUPTA

1995PHXF405

Under the supervision of

Dr. CHANDRA SHEKHAR



BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE

PILANI (RAJASTHAN) INDIA

2002

BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE
PILANI RAJASTHAN

CERTIFICATE

This is to certify that the thesis entitled **DESIGN EXPLORATIONS OF VLSI ARITHMETIC CIRCUITS** and submitted by **Ms. ANU GUPTA, ID. No. 1995PHXF405**, for award of Ph. D. Degree of the institute, embodies original work done by her under my supervision.

Chandra Shekhar

Signature
in full of the Supervisor

DR. CHANDRA SHEKHAR

Name
in capital block letters

SCIENTIST 'G', CEERI, PILANI

Designation

Date: 11-07-02

ACKNOWLEDGMENTS

I wish to convey my sincere gratitude and thankfulness to my guide **Dr. Chandra Shekhar,** Scientist G_1, CEERI, Pilani, for introducing me to the area of low power designing of arithmetic circuits, providing me all the guidance, helping me understand the concepts, and for his continuous encouragement and moral support.

I thank **Prof. S. Venkateswaran,** Director, BITS, Pilani for providing me the necessary infrastructure and facilities.

I am sincerely indebted to **Prof. L. K. Maheshwari,** Deputy Director, BITS, for his keen interest in the work and constructive suggestions offered from time to time.

I thank **Prof. K. E. Raman,** Deputy Director, BITS, **Prof. Ravi Prakash,** Dean (Research and Consultancy Division), **Prof. R. K. Patnaik,** Dean (Instruction Division), **Prof. G. Raghurama,** Dean (Faculty Division II), and **Prof. G. P. Srivastava,** Dean (Educational Development Division), for providing necessary administrative help.

I wish to express my deep gratitude to **Prof. Rajiv Gupta,** BITS, for providing me moral support and help in programming and in learning various software tools used in the present work.

I wish to express my sincere thanks to **Prof. S. Gurunaraynan,** BITS, for providing me help in various forms for the completion of the present work.

I express my sincere thanks to **Prof. S. Balasubramanian,** Group Leader (EEE), BITS, and **Prof. R. C. Jain,** BITS, for giving invaluable thought provoking suggestions from time to time.

I wish to express my sincere thanks to **Dr. S. C. Bose,** Scientist, CEERI, Pilani, for providing me excellent references on low power designing.

I thank all the **Staff members** of EEE, and Instrumentation for their constant support in completion of my work.

I am thankful to **Mr. S. D. Pohekar,** Lecturer and Ph.D in-charge (Research and Consultancy Division), BITS, for making necessary arrangements for seminars and providing guidelines for proper organization of the work.

I am thankful to **Mr. Bahadur,** Maintenance in-charge, VLSI CAD LAB, BITS, for his invaluable help in the maintenance of simulation data in the present work.

Lastly, I express my thanks to all those who directly or indirectly contributed to the completion of this work.

Anu Gupta

ABSTRACT

The objective of this thesis is to explore various adder architectures using different logic design styles and transistor sizes for different operand sizes and to obtain the optimal adder designs in terms of speed and energy consumption per addition for each operand size. Results obtained have been analyzed from a theoretical point of view to gain an understanding of the contributions of architecture, logic design style, and transistor sizing to the performance of the adder, and degradation of performance caused by layout parasitics. A predictive model has accordingly been built to propose the most optimal adder design for a given performance. The work has been carried out in two parts. In the first part, simulation results were generated using five different architectures; each designed using four logic design styles for three different transistor sizes. A standard cell layout format was used in each case. The designs were simulated to generate the values of worst-case propagation delay and energy consumption per addition.

This information is then used for validating the delay and energy consumption per addition in the second part. The approach used for modeling the delay was based on the determination of 'gates' on the critical delay path, average fan-out per gate in critical delay path, time-constant of the technology, and parasitic degradation factor. The energy consumption per addition was modeled by taking product of energy consumed in driving one gate capacitance (C_g) load, average load capacitance per node in units of C_g in an adder design, glitch factor, average number of gate output transitions in a design, parasitic degradation factor, and the average weight-factor per gate for energy consumption due to switching of internal nodes of a gate.

The work concludes with the development of tools, which can be used to predict an optimum adder design for a given application based on the speed and energy consumption constraints of the application.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

Adders of various bit-widths are frequently required in very large-scale integrated circuits (VLSI) from processors to application specific integrated circuits (ASICs). Addition is one of the fundamental arithmetic operations. Adders are essential not only for addition, but also for subtraction, multiplication, and division. Even for programs that don't do explicit arithmetic, addition must be performed to increment the program counter and to calculate addresses. Floating-point operations too eventually reduce to integer operations. Thus increasing the speed of integer operations will also lead to faster floating-point operations. As a result, the speed of microprocessors and computers heavily depends upon the speed of the adders. Adder logic is thus of obvious importance, and has received due attention from computer designers. The most important and widely accepted metrics for measuring the quality of adder designs in the past were propagation delay, and area. Efforts in the past were focused towards increasing the speed of computing systems. As a result high-speed computation has become an expected norm for the average user.

Minimizing area and delay has always been important, but reducing power consumption has gained importance more recently - both because of increasing levels of integration and the desire for portability. There is an increasing demand for portable applications requiring high throughput and vastly increased capabilities like notebook, laptop computers and personal communication services (PCS's) without the need to be connected to a wired network. Though improvements in battery technology are being made, the progress there is slow as compared to the advances in electronic circuits and it is unlikely to provide a solution to the power problem [Powers 1995]. It has thus become imperative to develop integrated circuits and systems that use less energy - without greatly sacrificing computational throughput. The situation has been further aggravated by the fact that microprocessor on-chip clock rates have already crossed 1 GHz mark, leading to a substantial increase in dynamic (switching) power consumption. Furthermore energy efficient circuits are also needed in high performance desktops, AC powered systems in which sinking large amount of heat through packages is becoming a

difficult problem. Thus designing a low-power processor is as important as designing a high performance one.

There are several degrees of freedom available in the design of low-power high-performance circuits (here adders) and systems. These are: process technology, circuit design style, architecture, and algorithm. [Chandrakasan et al. 1992, Bellouar et al. 1995, Sinencio et al. 1999] Complementary metal oxide semiconductor (CMOS) technology, the vehicle for VLSI, offers the combination of large noise margins, ruggedness of design, low power consumption, scalability of technology and validity of the logic design style at scaled down technologies. Within the CMOS technology, the designers have the freedom of choosing the architecture, the circuit design approach and the transistor sizes for implementing various arithmetic functions. Besides these, technology scaling including threshold voltage scaling, and supply voltage scaling constitute other techniques that can be used in low-power digital design.

Despite the simplicity of addition, there isn't a single best way to perform high-speed addition. [Hennessy et al. 1996] The adder architectures are available from the simple but slow ripple carry adder to the fairly complex but fast carry look-ahead adder. The performance of adders also depends upon the choice of logic design style and the transistor sizes used. Hence there exist numerous possibilities of making changes in the adder designs because of which many designs can be developed and the performance of every design will differ from other designs. Consequently, it is important to study, design, and simulate a range of CMOS adders of different bit-widths; each built using several different architectures, logic design styles and transistor sizes. Using this information, it should be possible to develop a model that can be used for selecting the optimal adder design for a given application.

CHAPTER 2

A REVIEW OF HIGH-PERFORMANCE LOW-POWER CMOS ADDER DESIGNS

Over the years an important aspect of arithmetic circuit design for most applications has been the minimization of their delay. In a classical ripple carry adder, the carry propagates in a time proportional to the bit-width of the adder. The advantage of this adder is its simplicity and its economical use of hardware. This, however, is obtained at the cost of an increased carry propagation delay. The sum and output carry for this adder are given by relations

$$s_i = a_i \oplus b_i \oplus c_{i-1}$$

$$c_i = p_i.c_{i-1} + a_i.b_i$$

where $p_i = a_i \oplus b_i$

As c_i is dependent on c_{i-1}, it can be thought that the problem of adding two n-bit numbers should be intrinsically linear. But it is recognized that if both operands are equal (a_i=b_i=0; a_i=b_i=1), there is no need to know c_{i-1} to obtain c_i. Thus it is possible to build an adder whose average time of computation would be proportional to the average size of the longest chain of differing bits of operands a_i and bi. Burks et al. [1946] have shown that this average size is upper bounded by $\log_2 n$.

For instance, in the following example, all the blocks separated by slashes can be added in parallel:

1010|0001|1001001101|001|101

1101|0110|1110110010|010|110

From a practical point of view (particularly for use in synchronous systems) the adder must be based on the "worst-case delay". [Lehman et al. 1960] Thus the worst-case time of addition must be minimized instead of the average time. Various solutions for speeding-up addition have been proposed to address this problem

Weinberger et al. [1956] in their simultaneous carry circuit (or carry look-ahead circuit) have taken advantage of the fact that the recursive form of the full carry-function may be

expressed in non-recursive form. This implies that output carry c_i need not depend explicitly upon lower order carry c_{i-1}, but can be expressed as a function of only the relevant augend and addend bits - thus reducing carry-propagation delay.

The carry skip technique [Morgan et al. 1959] exploits the occurrences $a_i=b_i$. But large chains of consecutive bits 'i' may arise, such that $a_i \neq b_i$. So, adder design is divided in blocks of ripple carry adders, where a special circuit associated with each block detects if all the bits to be added are different ($p_i = 1$ in all the blocks). In this case the input carry (c_{in}) of the block directly bypasses it and is fed to the next block. This technique is further refined in many ways. Lehman et al. [1961] have addressed the issue of optimum size of equal groups, and the effect of unequal groups on the speed of addition. They have shown that for an adder of length (m+1) bits, which is to be divided into k-skip groups each of n-bits so that:

$nk=m+1$,

the time (T) required for worst-case carry propagation is

$T = [1 + (n-1) + (k-2) + (n-1)]$

Minimization of T with respect to n gives

$$n = \sqrt{\frac{m+1}{2}}$$

or

$$k = 2n.$$

Thus, optimum integral values of k, and n can be determined. It also has been shown that worst-case propagation time can be decreased further by reducing the size of the least significant group and increasing the sizes of succeeding groups by one stage for each group up to the middle group, and then decreasing in the same way so that the dependence of propagation time on the chain length is almost eliminated. It has been shown that such a design when applied to 60-bit adder gives a speed-up of 20 % without any increase in the amount of the hardware required. The adder has been analyzed assuming that the ratio (r) of the time needed by the carry to skip a block to the time needed by the carry to pass through a full-adder cell is 1. Barnes et al. [1985] have given a strategy for finding optimal sizes of blocks if the ratio (r) is an integer with $2 \leq r \leq 7$. Guyot et al. [1987] have considered the general case of this ratio (r) to be any non-

negative number. By reducing the task of minimizing time '*T*' to a geometric problem, they have developed an algorithm for two level variable carry skip adder designed using restoring logic to obtain the optimum sizing of the blocks. Majerski's [1967] suggestions for a multilevel implementation of variable-size carry skip adders are for improvement in speed through focusing on reducing the number of skips. Chan et al. [1990] have used Manchester carry skip adder using dynamic logic and analyzed it using RC timing model. Based on it a polynomial algorithm has been developed to determine near optimal blocks for one level skip.

In the carry select adder proposed by Bedrij [1962], the addend and augend are divided into subaddend and subaugend sections that are added twice to produce two subsums. One addition is done with a carry bit forced to '1'and the other addition is done with a carry bit forced to '0'. The selection of the correct subsum from each of the adder sections depends upon whether or not there actually is a carry into that adder section. Sklansky's conditional sum adder [1960 b] is based on determination of conditional sums and output-carries that can arise from all possible distributions of input carries for groups of addend and augend. By passing the appropriate information through a series of selecting switches, the true sum and true final carry-out are selected. Gosling [1971], and Sklansky [1960 a] have also given a review and evaluation of high-speed addition techniques.

With the advent of battery-operated applications like portable computing and personal communication systems, energy efficient circuits are needed because of the difficulty in providing adequate cooling to high-density chips and for increasing the battery lifetime [Chandrakasan et al. 1992, Bellouar 1995, Roy et al. 2000]. Hence, the designers need to estimate the delay as well as the average power dissipation accurately before the circuit goes to fabrication. Accurate timing analysis has been the subject of numerous investigations over the years. Auvergne et al. [1986] have defined delays t_{HL} (t_{LH}) in enhancement-depletion MOS logic gate as the time spent by the output to fall (rise) from the static high level (static low level) to $\frac{1}{2}V_{DD}$ respectively. For actual loading conditions, the time spent by the input to fall from V_{IH} down to $\frac{1}{2}V_{DD}$ or to rise from V_{IL} up to $\frac{1}{2}V_{DD}$ is added to it. The difference of currents of pull-up and pull-down averaged over the end points of the voltage transition is calculated to include the influence of the finite input slope on the output current using which t_{HL} (t_{LH}) is evaluated. Auvergne et al.

[1987] have generalized the above results to CMOS inverters and showed that delay time in unidirectional cells is composed of time spent by the input drive to cut off the PMOS or NMOS transistor and time corresponding to step response of the on transistor. Deschacht et al. [1988] have developed models for estimation of propagation delays of general CMOS structures (CMOS data paths) taking into account device size, process parameters, layout parasitics, and realistic output loading conditions. The approach used is to partition the data path and real delays are then obtained through temporal evaluation of unidirectional (driver-load gates) and bi-directional elements (transmission gates). The model is further improved upon including slow input ramp effects in delay evaluation. [Auvergne 1990] It is shown that for a fast ramp the delay increases with slew time of the ramp, and then decreases with very high values of the input slew. Lee et al. [1984] have analyzed the single stage CMOS gate delay for small (fall-time and rise-time of the output voltages shorter than the rise-time and fall-time of the input voltage), medium (fall-time and rise-time of the output voltages nearly equal to the rise-time and fall-time of the input voltage) and large (fall-time and rise-time of the output voltages longer than the rise-time and fall-time of the input voltage) loading conditions. Based on the results, they have presented a minimum delay time algorithm that investigates the technology parameters and the load capacitance and calculates the minimum delay time and suggests the number of stages required in the critical path of any integrated circuit. Also, the channel widths of all transistors are calculated which yield the minimum silicon area for a required delay time. Hedenstierna et al. [1987] have developed a model based on analytical solution for the CMOS inverter output response to an input ramp. The model is improved by considering the propagation delay as a function of step-response delay of the previous stage. Tsao et al. [1986] have presented timing model which uses a switch-level state predictor for determining the steady-state and then uses a forward Euler prediction method to predict the transient time between two adjacent voltage levels: V_{SS}, $\frac{1}{2}V_{DD}$, and V_{DD}. Hafed et al. [2001] have presented a technique to compute delay of a CMOS inverter driving R-C load. They have used the concept of determination of effective capacitance for estimating supply current and output voltage transitions while the charging/discharging capacitor is in saturation because, in this region, inverter behavior is insensitive to resistive component of the interconnect. The current and voltage transitions for linear mode operation of transistor are estimated by computing its effective resistance. The model has been shown to be accurate for 0.8μm, 5v and 0.24μm, 2.5v CMOS technologies.

There are three major sources of power dissipation in digital CMOS circuits: (1) switching component of power, (2) direct-path short-circuit current, and (3) leakage current. Veendrick [1984] has analyzed the short-circuit dissipation of CMOS inverter with and without load. He has shown that the short-circuit power dissipation is only a fraction (< 20%) of the total dissipation for equal rise and fall times of input and output signals. The dominant term in power dissipation is the switching component given by

$$P_{dynamic} = \frac{1}{2} C_L V_{DD}^2 f_{clk} \eta$$

Low-power designs, thus, aim at minimizing the power consuming transitions (switching activity factor 'η'), power supply (V_{DD}), and load capacitance (C_L). [Bellaouar et al. 1995] An accurate estimation of average power dissipation is required to estimate battery life [Chandrakasan et al.2000], while the peak power dissipation has a bearing on the circuit reliability and the proper design of power and ground lines. [Chowdhury et al. 1990, Wu et al. 2001, Evmorfopoulos et al. 2002] Since, the largest component of power dissipation is due to the signal transitions at circuit nodes, an accurate estimation of switching activity at the internal circuit nodes is required. [Najm 1994] The simplest approach is to use circuit simulation technique for a large number of input patterns and determine the current waveforms from the supply voltage. [Roy et al. 2000] Average power is then calculated by determining the average current from the power supply. Another method is the use of pattern independent probabilistic techniques for estimating switching activity. In these techniques, the input signal distribution is determined in terms of some probability values. Then an analysis tool is used to determine the average power dissipation based on input signal distributions. Najm [1993] has proposed the propagation of transition density (average switching rate) to avoid simulation for large number of input patterns. An algorithm, based on the stochastic model of logical signals, has been given to propagate transition densities of the primary inputs to the internal nodes and the output nodes. Ghosh et al. [1992] have included the impact of gate delays on switching activity. A general delay model is used for computing the Boolean conditions that cause glitching in the circuit. Then the probability of each gate switching at any particular time is computed from input switching rates. Then the sum of these probabilities over all the gates gives the switching activity in the entire circuit over all the time points in a clock cycle. Burch et al. [1993] have used a statistical

technique, Monte-Carlo approach, for power estimation. It requires monitoring the total power directly during the random simulation. This is continued until a value of power is obtained with a desired accuracy. This technique requires less time than that required to compute individual gate powers.

For portable applications, low power adder circuits along with sufficiently high performance are needed. Callaway et al. [1993, 2000] have investigated the worst-case delay and average power dissipation of adders and multipliers. They have analyzed the worst-case delay and average number of gate-output transitions per addition for 16-bit adders designed in fully static logic from three sources: (1) from gate level simulation, (2) detailed circuit simulation, and (3) actual measurement from a test chip. The circuits are subjected to 10,000 pseudo-random inputs. Based on unit-delay, unit-power gate model, average number of gate output transitions is obtained. Results show that number of transitions for each adder, except the conditional sum adder, increase linearly with the word size and that the power dissipation is normally distributed. These results are than compared with circuit simulation results and actually measured results. It is found that the simple unit gate delay model is inaccurate for carry-look ahead and conditional sum adders due to either large fan-in or large fan out gates in the worst-case path. Similarly, the distribution in time of the average power dissipation is found to be quite similar to those from the unit power model though it underestimates the power dissipation for adders with large fan-in and fan-out. Hence, unit-delay unit-power gate model can be used to generate only a first estimate of the power dissipation and worst-case delay of adders.

Nagendra et al. [1994] have compared the power-delay product of 32-bit ripple carry, blocked carry look ahead (BCLA), and signed digit adders (SDA). Results show that SDA with large bases consumes less power due to saving in the logic as fewer sign bits and corrections have to be handled. At the same time, the delay of the adder increases by a large factor due to longer ripple carry chains. Thus power-delay product increases with the base of SDA, when digit addition is done using ripple carry adder. A transistor-sizing methodology is also presented using which the speed benefits can be derived without increasing the power dissipation by a large factor. The transistor-sizing algorithm aims at optimizing the individual modules separately and then connecting them by hand for a compact layout. Apart from these detailed studies, efforts have also been made to

optimize a particular adder architecture- like the work of Hwang et al. [1989], Abu-Khater et al. [1994], and Lim et al. [1999].

Besides considering different adder architectures, another approach is to employ different CMOS circuit design styles [Chandrakasan et al.1992] to design energy efficient, high performance adder architectures for a given architecture. Chu et al. [1987] have compared differential cascode voltage switch logic (DCVS) and fully static CMOS logic using the full-adder circuit as a test vehicle. Fully static CMOS logic has been found to be superior to DCVS in regards to power dissipation and inferior in regards to input capacitance and transistor count. The speeds of the two technologies have been found to be similar. Ko et al. [1995 (a)] have compared different circuit techniques by implementing a 32-bit carry look-ahead adder in different circuit design styles. Results show that static styles are more energy efficient. Among the static circuit styles- fully static CMOS, complementary pass transistor logic (CPL), and dual pass transistor logic (DPL); DPL adder has been found to be more energy efficient than the other two at the cost of increased worst-case delay and silicon area. They have shown that with the use of low power design techniques, the DPL logic is most suited for energy efficient, high performance adder designs. Also with simultaneous scaling of power supply and threshold voltage, energy efficiency has been found to improve significantly.

The work of Ko et al. [1995 (b)] analyzed the effect of the input slew on the short- circuit power dissipation in conjunction with the variations in the output load. Based on the results, a low power design technique for non-speed critical net in the circuit has been suggested which involves judicious selection of gate strengths by considering input slew and output load conditions. Zimmermann et al. [1997] have compared fully static CMOS logic with CPL for a range of simple and complex logic gates. The results show that for all logic gates except full adder fully static CMOS performs much better than CPL and other pass transistor logic styles for low-power applications.

Also, efforts have been made to optimize 1-bit full adder cell. Zhuang et al [1992] have presented simple CMOS full adder circuit using the transmission function theory. Wang et al. [1994] have proposed better implementations of the exclusive-OR and exclusive – NOR functions for adder circuits. The proposed designs have non-complementary inputs and are shown to have good signal level outputs and driving capability. Shams et al.

[2002] have developed a library of full adder cells. An extensive performance analysis of these 1-bit full adder cells has been presented to help the designers in making a choice. Fahim et al. [2002] have recently proposed a new dynamic differential logic 'swing-limited logic' for low-power high performance applications. An 8-bit ripple carry adder designed using swing-limited logic has shown a small power delay product compared to other logic styles.

CHAPTER 3

ADDER ARCHITECTURES

There are different kinds of adder architectures available for conventional number systems. Some of these which are frequently used in designs [Hennesy et al. 1996], and are analyzed in the present work, are

- Ripple Carry Adder
- Carry Skip Adder
- Carry Select Adder
- Conditional Sum Adder
- Carry Look-ahead adder

3.1 RIPPLE CARRY ADDER

This adder is implemented by cascading n 1-bit full adders as shown in Fig. 3.1. In this adder, the carry ripples through the n-stages of the adder. The sum of n^{th} bits of the operands cannot be computed until the carry c_{n-1} is evaluated. Hence, the carry propagation path contributes to the worst-case delay.

For each i^{th} full adder block, the sum 's_i' and carry-out 'c_i' are evaluated using equations

$$s_i = a_i \oplus b_i \oplus c_{i-1} \qquad ...3.1$$

$$c_i = a_i.b_i + (a_i + b_i).c_{i-1} \qquad ...3.2$$

where a_i and b_i are the i^{th} bits of the operands to be added and c_{i-1} is the carry input.

The ripple carry adder is the slowest adder, but also the cheapest as it is built with only n simple cells, connected in a simple regular way.

3.2 CARRY SKIP ADDER

A carry skip adder is midway between a ripple carry adder and a carry look-ahead adder, both in terms of speed and cost. This adder improves the performance of a ripple carry adder by using a special speedup carry chain (skip chain). This adder has a good

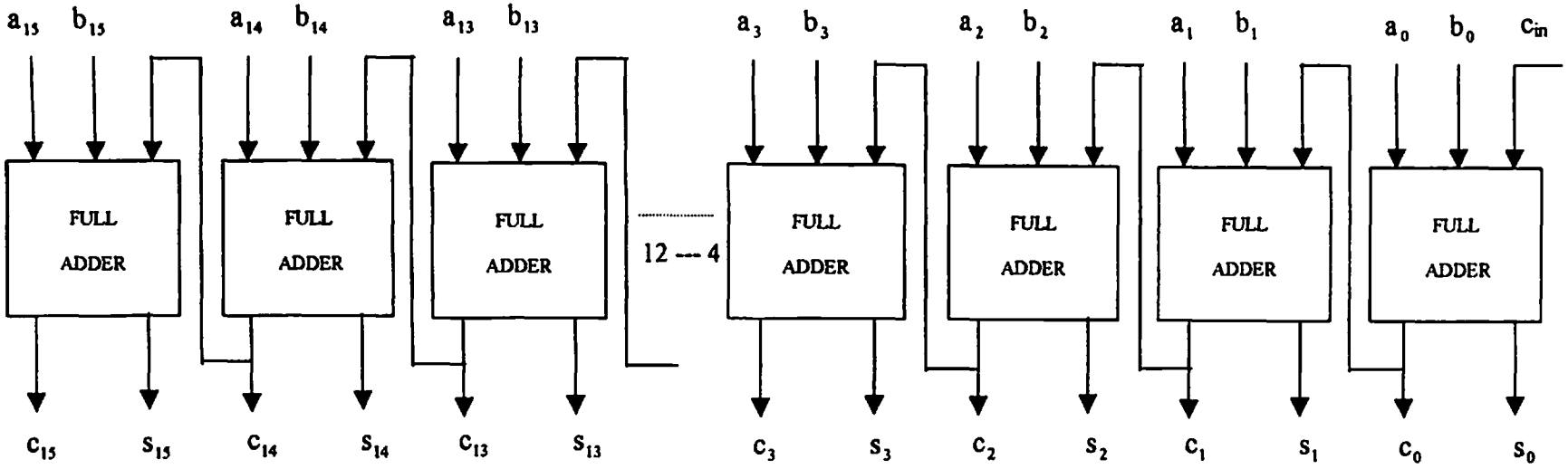


Fig. 3. 1 16-bit Ripple Carry Adder

topological regularity (due to ripple carry adder part), modest area increase (over the ripple carry adder), good modularity, and design simplicity.

The implementation of a 16-bit carry skip adder is shown in Fig. 3.2. A 4-bit carry skip adder, which is a 4-bit ripple carry adder with a carry skip path, is the basic block of this design.

At the start of the operation, carries begin rippling simultaneously through each block. If any block generates a carry, the carry-out of the block will be true, even though the carry-in to the block may not be ready yet. Once the carry out c_3 of the first block is generated, not only it goes to the second block directly, it is also fed to the third block through the AND gate of the skip path. The c_3 is allowed to pass through the skip path if the group propagate signal (p_{47}) from the second block is true.

The p_{47} and $p_{propagate}$ signal of the second block are generated using equation

$$p_{47} = p_4 \cdot p_5 \cdot p_6 \cdot p_7 \qquad ...3.3$$

$$p_{propagate} = c_3 \cdot p_{47} \qquad ...3.4$$

$$c_7 = p_{propagate} + c_{7a} \qquad ...3.5$$

For each i^{th} full adder in the 4-bit block, the sum 's_i' and carry-out 'c_i' signals are evaluated using equations

$$p_i = a_i \oplus b_i \qquad ...3.6$$

$$s_i = a_i \oplus b_i \oplus c_{i-1} \qquad ...3.7$$

$$c_i = a_i.b_i + (a_i \oplus b_i).c_{i-1} \qquad ...3.8$$

The critical path in the 16-bit carry skip adder starts with a carry generated at the 0^{th} position, which then ripples through the first block, skips second and third blocks and is used by the fourth block to generate its last sum signal s_{15}.

$$(c_0) \rightarrow (c_1) \rightarrow (c_2) \rightarrow (c_3) \rightarrow (c_7) \rightarrow (c_{11}) \rightarrow (s_{12}) \rightarrow (s_{13}) \rightarrow (s_{14}) \rightarrow s(15)$$

The speed of the carry skip adder can be improved by making interior blocks larger.

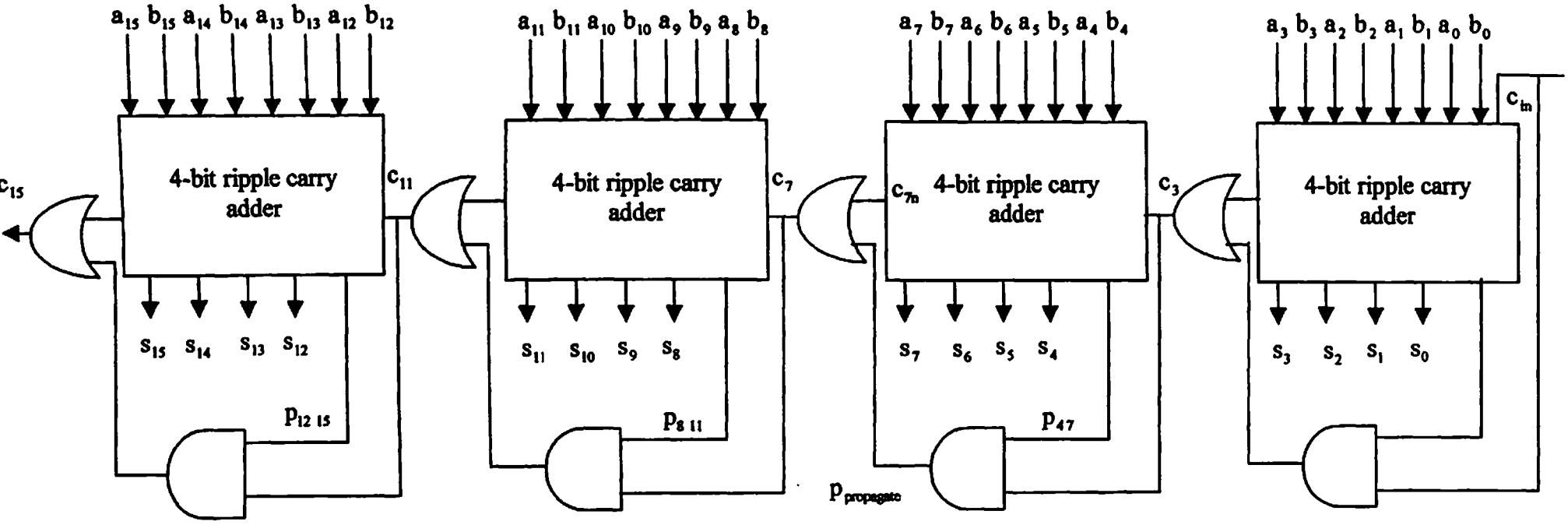


Fig. 3. 2 16-bit Carry Skip Adder

3.3 CARRY SELECT ADDER

A carry select adder is a modification of the ripple carry adder to improve its speed. Because of ripple carry adder part, it has a regular and simple layout. Fig. 3.3 shows the implementation of a 16-bit carry select adder with a 4-bit carry select adder as the basic block. The first stage is a 4-bit ripple carry adder to save silicon area because input carry is known. A 4-bit carry select adder consists of two 4-bit ripple carry adder blocks. It works on the principle of two additions in parallel: one block assuming the carry-in is '0' and the other assuming the carry-in is '1'. The real carry-in is computed from the previous block and is used to select one of the two sum outputs. This implementation though, regular and easy to layout, has a larger area as compared to ripple carry adder. For 4-bit ripple carry adder block in the second stage with carry-in '0' or '1', the sum signals 's_i^0', s_i^1 and carry-out signals 'c_i^0', c_i^1 signals are evaluated using equations (3.6), (3.7), and (3.8).

The correct sum for the i^{th} bits of the operands in second stage is selected through a multiplexer using the input carry 'c_{in}' to the 4-bit block (or carry-out of the previous block) as the select signal using equation

$$s_i = s_i^1 . c_{in} + s_i^0 . \bar{c}_{in} \qquad \text{...3. 9}$$

Since c_{in} drives as many multiplexers as bits in an adder block, its large fan-out can have an effect on speed.

The correct carry-out 'c_7' of the second stage is obtained as

$$a = c_{71} . c_3$$

$$c_7 = c_{70} + a \qquad \text{...3. 10}$$

where 'a' is an intermediate signal in the carry select path.

Worst-case delay path for the 16-bit carry select adder corresponds to the case of carry being generated at the 0^{th} position, rippling through the first 4-bit ripple carry adder block, skipping second and third 4-bit carry select block through carry select path, and being used by the fourth block to generate its carry-out 'c_{15}'.

$$(c_0) \rightarrow (c_1) \rightarrow (c_2) \rightarrow (c_3) \rightarrow (c_7) \rightarrow (c_{11}) \rightarrow (c_{15})$$

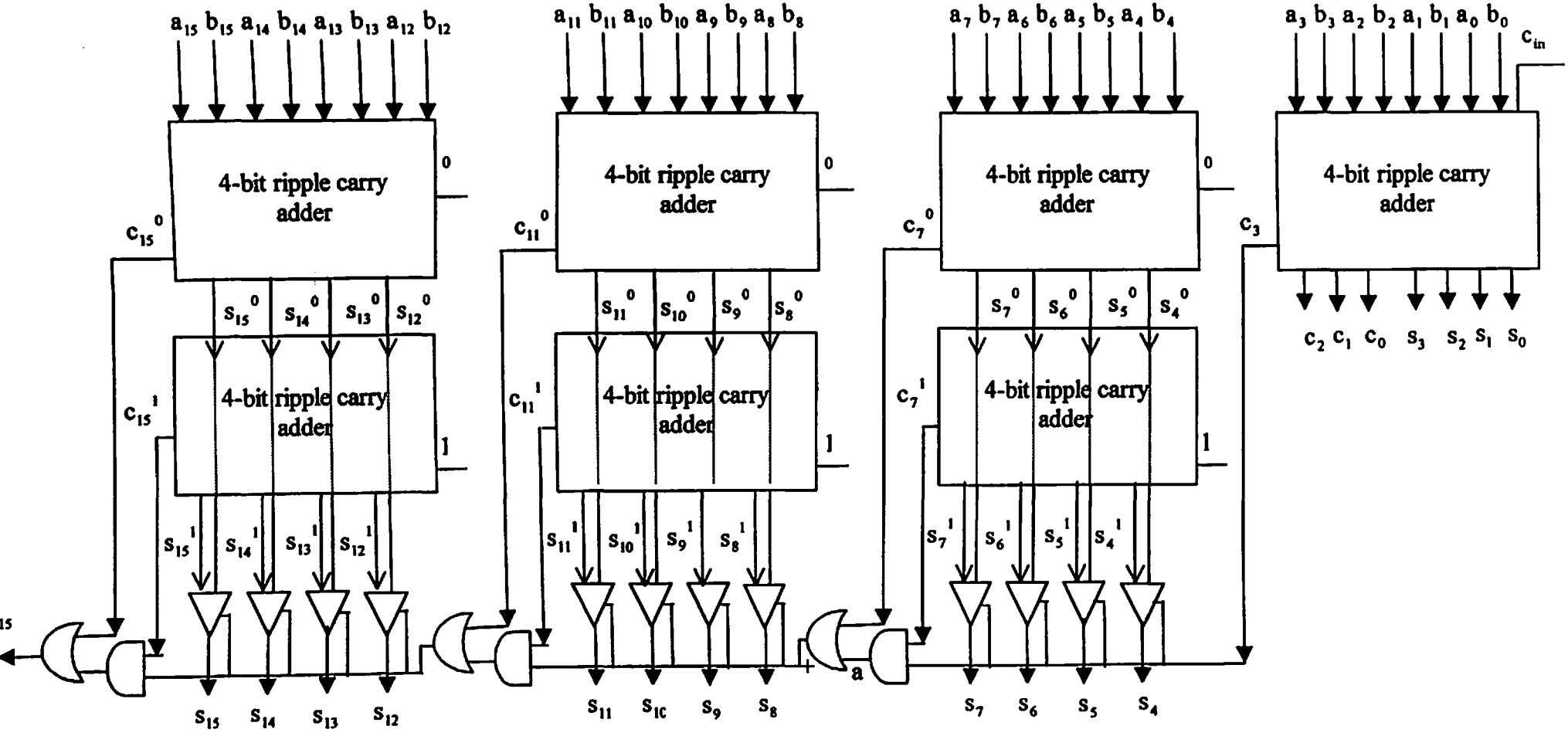


Fig. 3. 3 16-bit Carry Select Adder

3.4 CONDITIONAL SUM ADDER

The implementation of a 16-bit conditional sum adder is shown in Fig. 3.4 (a). A 4-bit conditional sum adder shown in Fig.3.4 (b) is the basic unit of this design. It uses two types of the cells:

i. the conditional cell

ii. the multiplexer.

For each bit of the adder, there is one conditional cell circuit. It computes two sums and two carries: s^0 and c^0 are calculated for a carry in of 'zero', and s^1 and c^1 are calculated for a carry in of 'one'. These signals are constructed using equations

$$s_i^0 = a_i \oplus b_i \qquad ...3.11$$

$$s_i^1 = \overline{a_i \oplus b_i} \qquad ...3.12$$

$$c_i^0 = a_i . b_i \qquad ...3.13$$

$$c_i^1 = a_i + b_i \qquad ...3.14$$

To design an n-bit adder, blocks of constant width or variable width can be cascaded together. At the start of each operation, all conditional cells compute their respective double sums and double carries in parallel. The true sums and carry-outs of each block are then selected by the carry-out of the previous block. Hence the carry-out signal has large fan-out.

For the 16-bit implementation, the critical path starts with generation of carry-signals (c_0^0, c_0^1) at the first conditional cell, which are used to select correct values of (c_1^0, c_1^1) of next conditional cell and then selection of other carry signals in the following sequence.

$$(c_0^0, c_0^1) \rightarrow (c_1^0, c_1^1) \rightarrow (c_2^0, c_2^1) \rightarrow (c_3^0, c_3^1) \rightarrow (c_3) \rightarrow (c_7) \rightarrow (c_{11}) \rightarrow (c_{15})$$

3.5 CARRY LOOK-AHEAD ADDER

Carry look-ahead adder avoids the linear growth of carry delay by generating carries in parallel. Fig. 3.5 shows the tree structure of an 8-bit carry look-ahead adder. The adder performs addition in two parts. In the first part, propagate and generate signals are derived from operands applied to each of the 1-bit adder blocks (A-cell) at the first level of the tree structure, for the i^{th} 1-bit adder, using equations

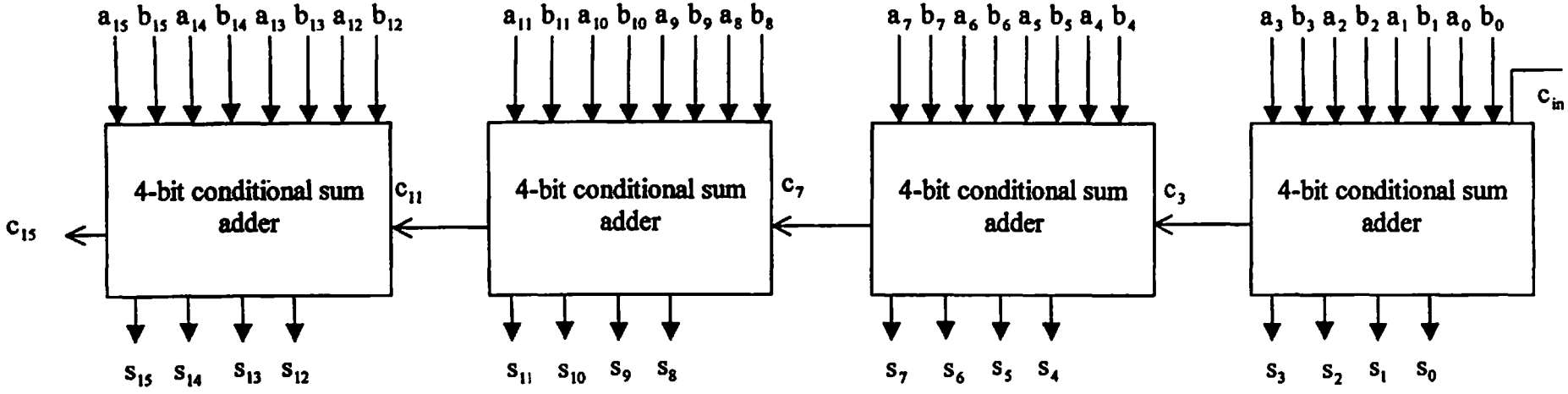


Fig. 3.4 a 16-bit Conditional sum adder

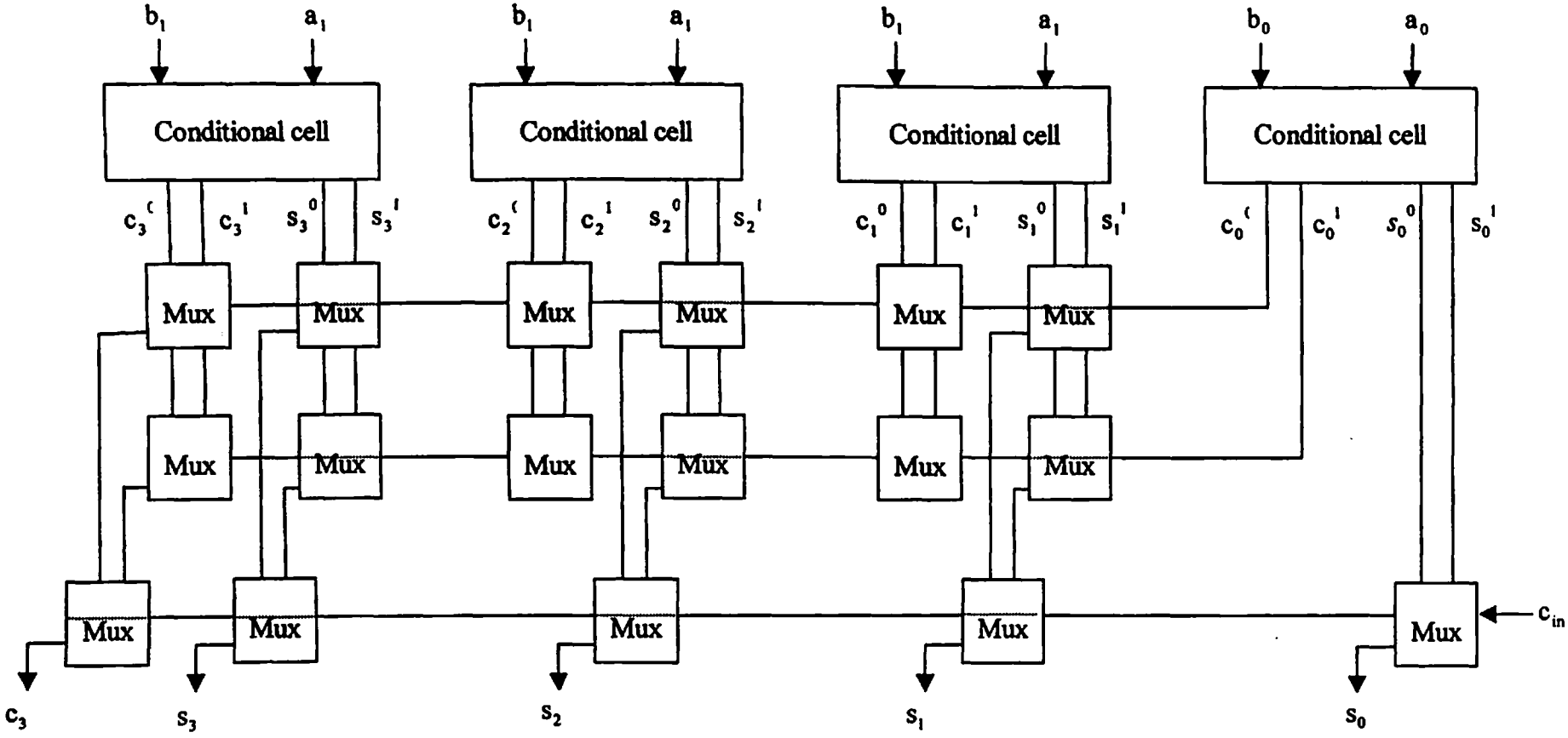


Fig. 3.4 b 4-bit Conditional sum adder block

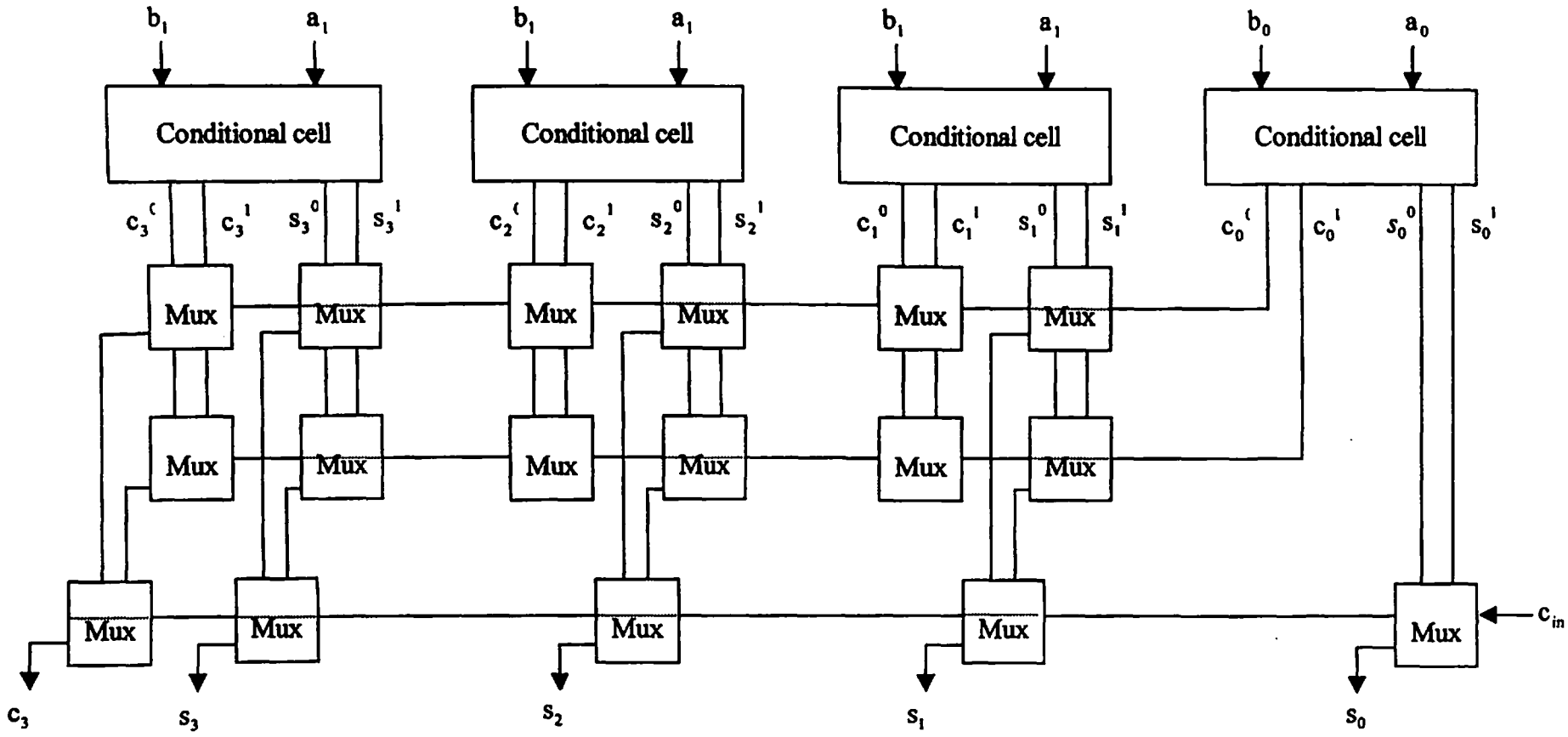


Fig. 3.4 b 4-bit Conditional sum adder block

$$g_i = a_i.b_i \quad \text{...3.15}$$

$$p_i = a_i \oplus b_i \quad \text{...3.16}$$

These values are then used to find the group propagate and generate (p_{01} and g_{01}) values for a group of two successive A-cells at the second level of the tree (comprising of B-cells) using, for the first group, equations

$$g_{01} = g_1 + p_1.g_0 \quad \text{...3.17}$$

$$p_{01} = p_0.p_1 \quad \text{...3.18}$$

p_{01} and g_{01} contain the information regarding the propagation through or generation of carry in the group comprised of the first and second A-cells.
The values of (g_{01}, p_{01}) and (g_{23}, p_{23}) are then used by another B-cell at the third level to find the values of group generate and propagate functions (g_{03}, p_{03}) for the group of four successive A-cells. Similarly, values of g_{07} and p_{07} are found by combining (g_{03}, (p_{03}) and (g_{47}, p_{47}) at the fourth level of the tree structure.

In the second part, the input carry to the first A-cell, 'c_0', is fed to B-cell at the bottom of the tree structure. B-cells, at the fourth, third level, and second level generate c_1, c_2, and c_4 signals simultaneously with c_0 as one of the inputs using equations

$$c_1 = g_0 + p_0.c_0 \quad \text{...3.19}$$

$$c_2 = g_{01} + p_{01}.c_0 \quad \text{...3.20}$$

$$c_4 = g_{03} + p_{03}.c_0 \quad \text{...3.21}$$

Similarly, carries c_6, c_5 and c_3 are generated at the same time using a similar logic. Carry c_7 is obtained using c_6, g_6, and p_6 as the inputs.
An extra logic is used to compute the carry-out 'c_8' of the 8-bit adder using equation

$$c_8 = g_7 + p_7.c_7 \quad \text{...3.22}$$

The sum signals are generated by individual A-Cell from operands a_i, b_i and c_i (not c_{i-1} for this adder as c_0 is the input carry to the adder) and their complements using equations

$$p_i = a_i \oplus b_i \quad \text{...3.23}$$

$$s_i = p_i \oplus c_i \quad \text{...3.24}$$

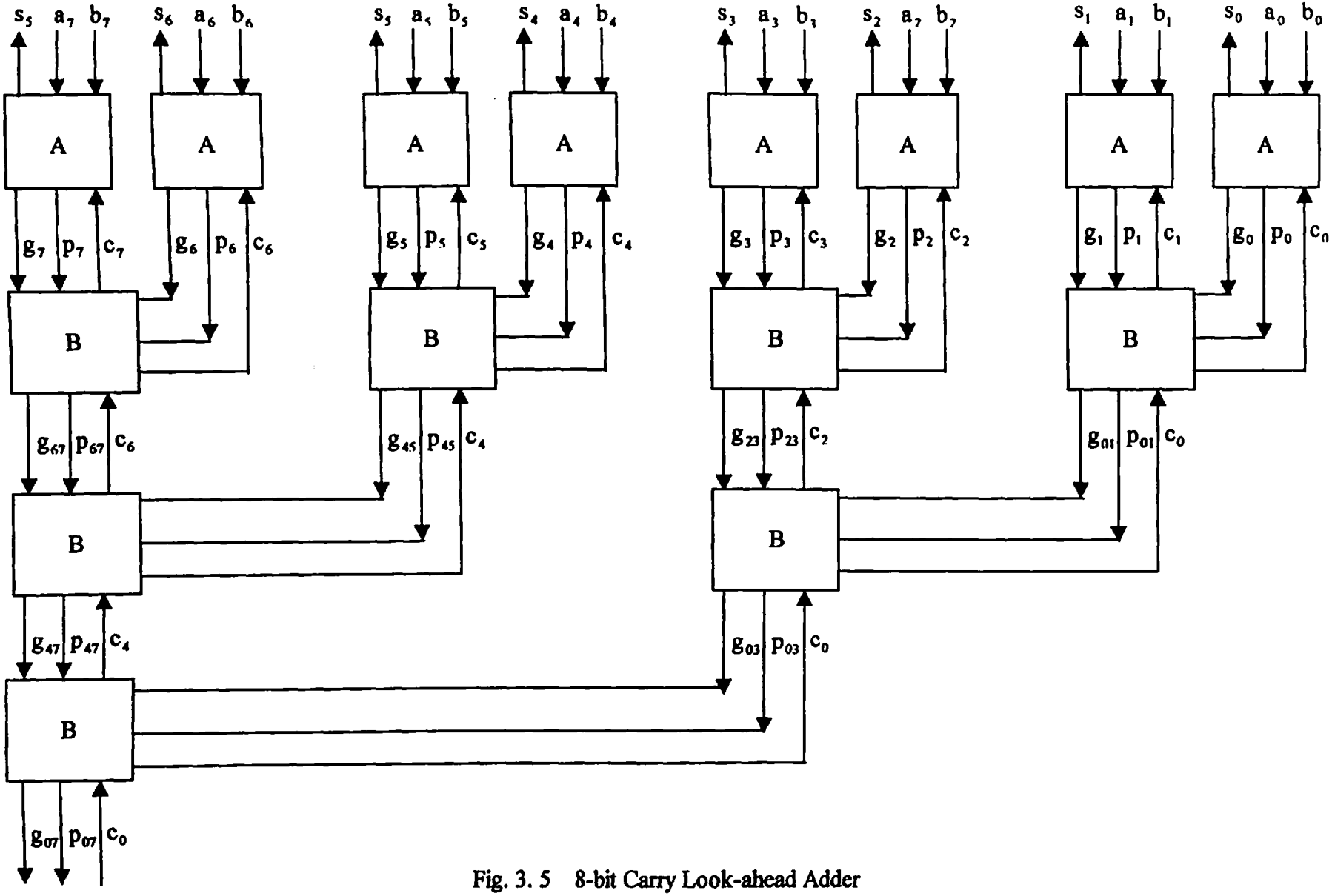


Fig. 3. 5 8-bit Carry Look-ahead Adder

Since the carries are generated in parallel, the maximum delay occurs in generation of c_8 The critical path for an 8-bit carry look-ahead adder starts with the generation of signals (g_0, p_0, g_1, p_1, g_2, p_2, g_3, p_3) in A-cells simultaneously and ends at determination of c_8 in the following sequence.

$(g_0, p_0), (g_1, p_1) \rightarrow (g_{01}, p_{01}), (g_{23}, p_{23}) \rightarrow (g_{03}, p_{03}) \rightarrow c_4 \rightarrow c_6 \rightarrow c_7 \rightarrow c_8$

Same structure can be extended to implement a larger sized carry look-ahead adder, like 16-bit, 32-bit, and 64-bit.

CHAPTER 4

DIFFERENT CMOS LOGIC DESIGN STYLES

The CMOS technology has the advantage of low power consumption, and large noise margins. [Shoji 1988] Besides considering different adder architectures, one can also use different CMOS logic styles to design energy efficient, high performance adder circuits for a given architecture. There are a number of options available in the choice of basic circuit approach and topology for implementing various logic functions. [Glasser 1985] For the present study, four different logic design styles have been used for designing the adders, based on available information in the literature regarding their suitability for high speed and for low power logic circuit design.

- Conventional fully static CMOS logic
- Domino CMOS logic
- Complementary pass-transistor logic
- Dual pass transistor logic

4.1 CONVENTIONAL FULLY STATIC CMOS LOGIC

Conventional fully static CMOS logic is a common CMOS logic design style choice since it involves ruggedness of design, large noise margins, low power consumption, and validity of logic design style at scaled down technologies. [Weste et al. 1993, Kang et al. 1999, Martin 2000] A basic circuit is shown in Fig.4.1. Two logic blocks, N and P, form a CMOS gate. The topology of N block is the dual of that of the P block. The two blocks have equal number of transistors. This style gives negligible DC power dissipation, as there is no direct path between power supply and ground for any of the logic input combinations. The channel widths of series connected n-channel MOS transistors (NMOS) or p-channel MOS transistors (PMOS) have to be increased to obtain a reasonable conducting current to drive capacitive loads. This results in a significant area overhead, as also an increased gate input capacitance, and therefore, high dynamic power dissipation.

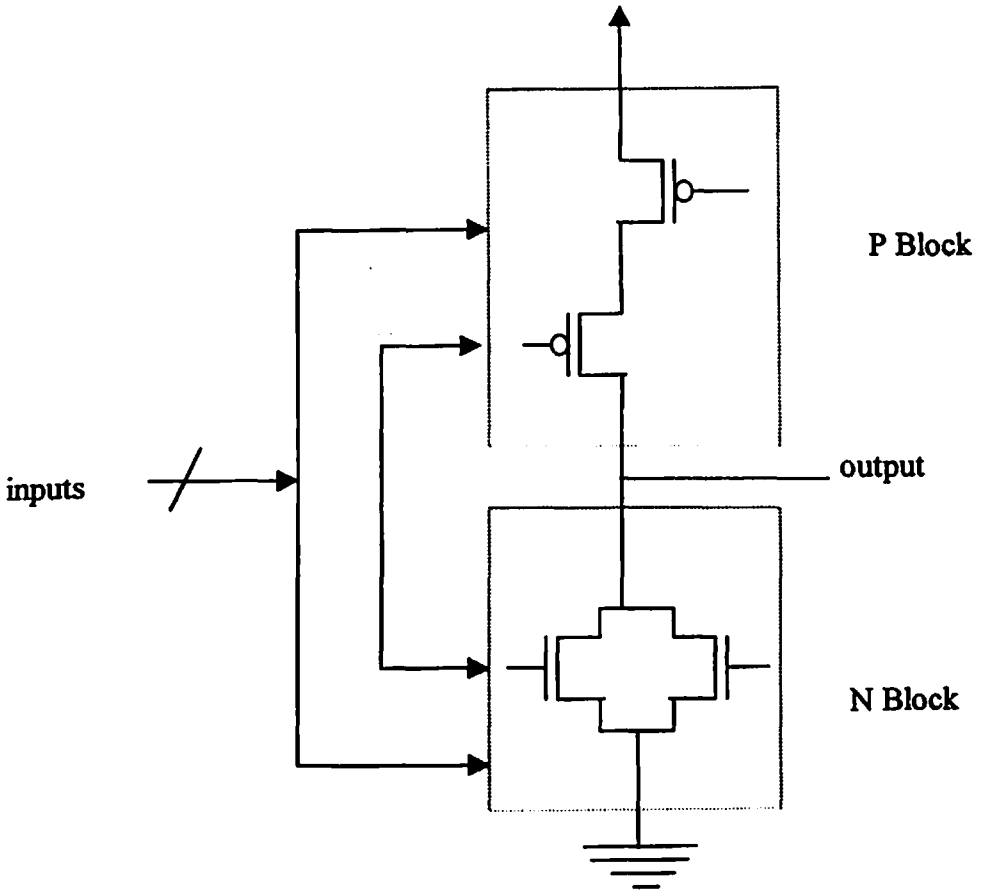


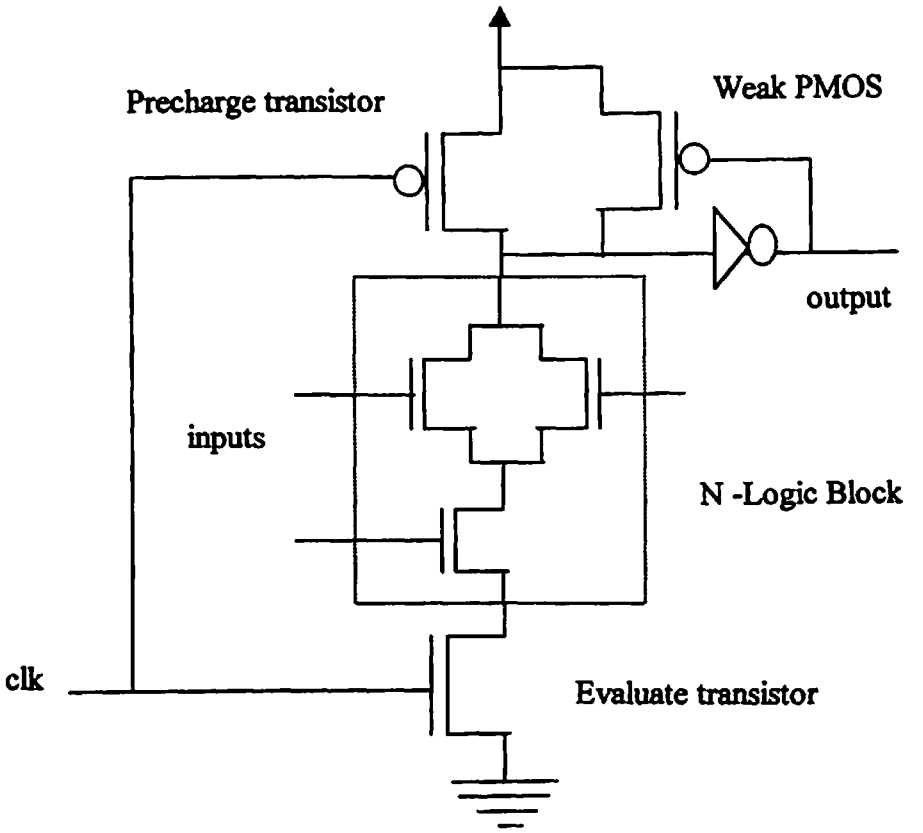
Fig. 4.1 Conventional fully static CMOS gate

Furthermore, the high input capacitance also loads the previous stage, increasing its delay. In a static logic gate, the output, or, the internal nodes can spuriously switch before their correct logical value stabilizes due to finite propagation delay from one logic block / sub-block to the next. Such transitions increase the dynamic power consumption of the circuit by adding unwanted switching activity in the circuits. Buffers must, therefore, be inserted to equalize the fast paths. Static CMOS gates also exhibit short-circuit currents. However, by sizing transistors for equal rise and fall times, the short-circuit power component can be minimized.

4.2 DOMINO CMOS LOGIC

A domino CMOS implementation is shown in Fig. 4.2. It consists of a dynamic CMOS circuit followed by static CMOS buffer. The dynamic circuit consists of a PMOS precharge transistor, an evaluation transistor, and N-logic block, which, in general, is a series-parallel combination of NMOS transistors activated by the inputs and implementing the required logic. [Weste et al. 1993, Kang et al. 1999, Martin 2000] This circuit style uses a single clock phase *clk*. During the low phase of the clock, output of the buffer is precharged to ground. During the evaluation phase, the output node is either charged or stays discharged. The number of stages in a cascaded set of domino logic gates is limited by the duration of the evaluation clock phase.

In comparison to the conventional fully static CMOS logic, domino logic has a smaller input capacitance because of fewer (only NMOS) transistors in complex designs. It also has a comparatively improved fall time. However, the rise time is larger, since there is an additional series transistor (the evaluate transistor) in pull-down path. The domino gate suffers from the charge-sharing problem in which the parasitic capacitances at the internal nodes get coupled to the load capacitance. This can degrade the 'high' logic voltage at the input of the buffer and provoke the buffer to dissipate high static power.



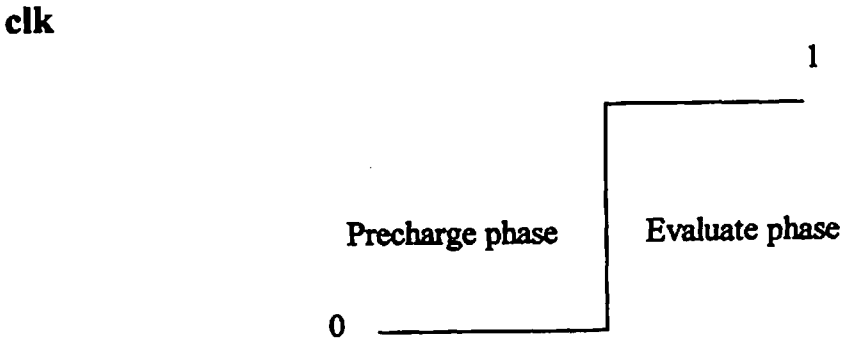


Fig. 4.2 Domino CMOS gate

A weak PMOS transistor driven from the output of the inverting CMOS buffer can be used to remove this problem. An alternative technique is the use of additional precharge PMOS transistors to precharge intermediate nodes of complex gates. Another limitation of the domino logic gate is that it requires a large clock distribution network to drive the clocked transistors. This highly loaded network consumes a significant amount of dynamic power. Also domino circuits implement only non-inverting logic functions.

The domino logic circuits can be mixed with conventional fully static CMOS circuits to optimize the overall performance. These gates do not experience short-circuit power dissipation and glitching problems as any node can undergo at most only one transition per clock cycle.

4.3 COMPLEMENTARY PASS TRANSISTOR LOGIC

Complementary pass transistor logic (CPL) belongs to the pass transistor logic family. It has the advantage of improving the speed of CMOS circuits. It is different from the conventional CMOS transmission gate logic, which uses the transmission gate as a primitive element. [Weste et al. 1993, Kang et al. 1999, Martin 2000] The basic circuit idea of CPL is shown in Fig. 4.3. The circuit consists of NMOS pass transistor logic network driven by complementary inputs and producing complementary outputs driven by two CMOS inverters used as buffers. The NMOS pass-transistor logic network performs the pull-up and pull-down functions. The transistors in the network can be sized for fast operation. The transistors having larger sizes should be farther from the output.

CPL uses fewer transistors to implement logic functions - especially XOR. This particularly efficient implementation is important as it is widely used in arithmetic functions. However, a CPL circuit suffers from the threshold drop across the NMOS transistor, which results in reduced current drive and hence slower operation at reduced supply voltages. The buffers are used to obtain the full logic levels at the output. These also help in driving large capacitive loads efficiently. But, since the 'high' input voltage level at the regenerative inverters is not V_{DD}, the PMOS device in the inverter may not be fully turned off.

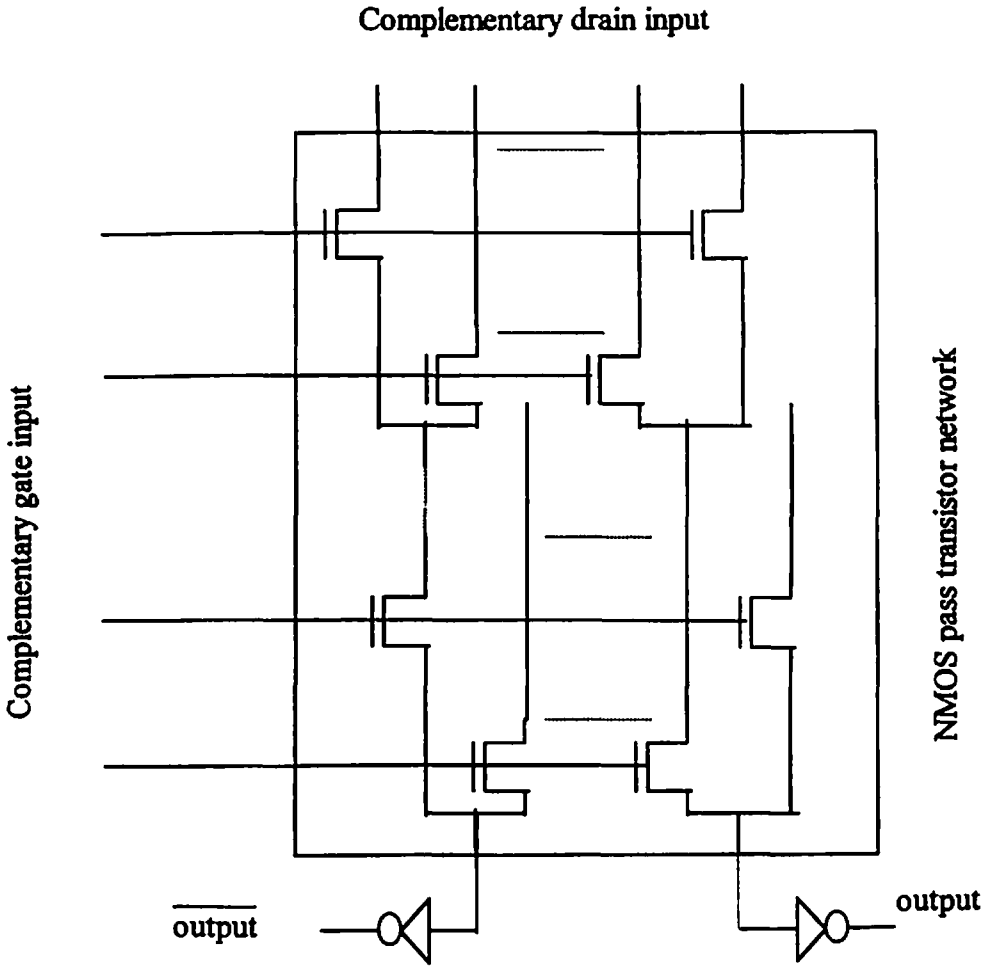


Fig. 4.3 Complementary pass transistor logic gate

This may leads to significant direct path static power dissipation if the threshold voltages of N and P devices are not designed appropriately. Alternatively, a PMOS latch driven by the complementary output signal can be used.

In comparison to the conventional fully static CMOS logic, this logic is faster and dissipates lesser power due to lower internal swings (excluding power dissipated by buffers). Also, the circuit schematic is typically structured and results in a simplified compact layout.

4.4 DUAL PASS TRANSISTOR LOGIC

The dual pass transistor logic (DPL) is a modified version of CPL. [Weste et al. 1993, Kang et al. 1999, Martin 2000] The basic circuit idea of CPL is shown in Fig. 4.4. It alleviates the problems of noise margin and speed degradation at reduced 'high level' associated with CPL. A DPL gate consists of NMOS and PMOS transistors in contrast to CPL gate, where only NMOS devices are used. The NMOS transistors pass the low level while PMOS transistors are used to pass the high level. The output of DPL gate is full rail-to-rail swing. Additional PMOS transistors result in increased input capacitance. For any input combination, there are always two current paths driving the output. This may compensates for any reduction in speed due to additional PMOS devices.

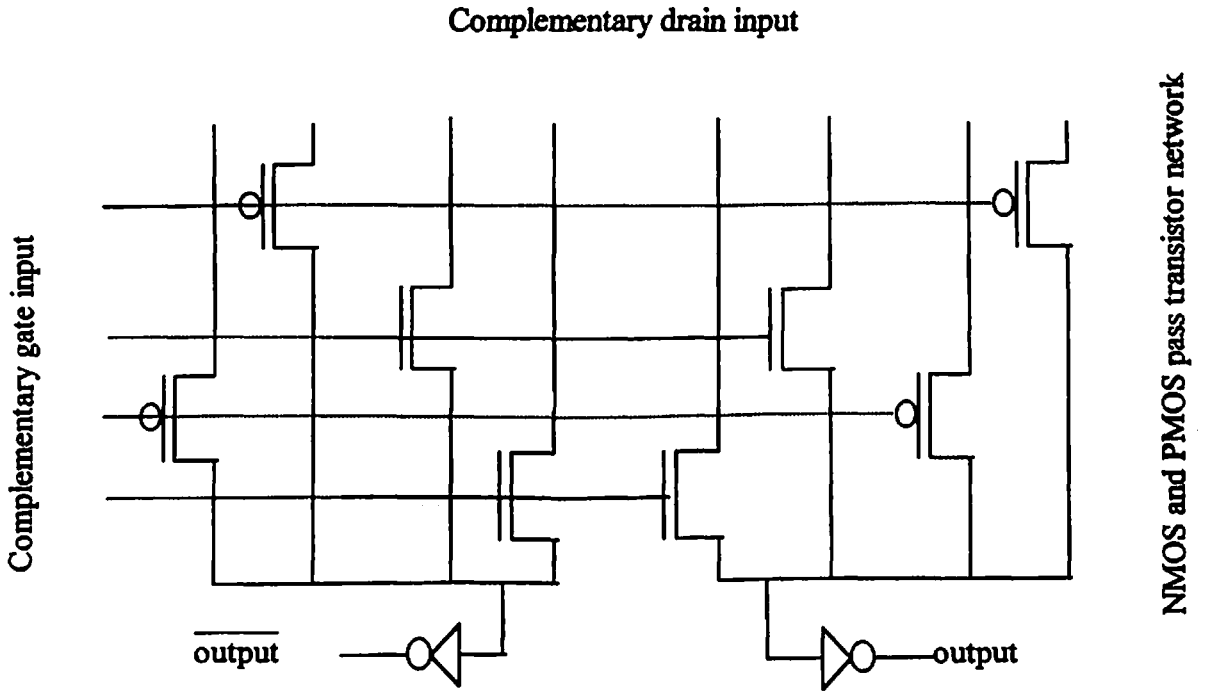


Fig. 4.4 Dual pass transistor logic gate

CHAPTER 5

CELL DESIGN FOR DIFFERENT ADDERS

There are many layout design methodologies to accomplish the physical design of a complex circuit. In the full-custom design methodology, the layout of each transistor is optimized. The layout of a complex chip using full-custom design is carried out for reasons of speed enhancement and area reduction. However, this style has low design productivity and a longer turn around time. But, in the case of low-power circuits, the full-custom design can be used to minimize the power of the circuit. In the standard-cell approach, layouts for different gates, latches, and flip-flops (cells) are created using a 'standard' layout format which imposes a standard height for all the cells, together with standard positions along the vertical edges of the cell at which supply and ground lines interface with it, and standard layers on which inputs and outputs of the cell are available along the horizontal edges of the cell. This approach, which is very popular for ASICs, provides a lower design cost and a higher productivity- at the expense of chip area, speed and power to different extents.

We have chosen the standard cell approach to create the designs of different adders. Standard cells of basic logic blocks are designed in fully static, domino CMOS, complementary pass transistor and dual pass transistor logic design styles. The technology used is a 1.2μm N-well CMOS technology. Also, three versions of each cell have been created for three different transistor sizes keeping channel length the same and varying only the channel widths. The adders circuits have been designed using Tanner Tools and the library of schematic symbols corresponding to the standard cells are generated using schematic editor S-Edit. The layouts have been generated using automatic placement and routing tool 'SPR' of Tanner Tool's layout editor L-Edit. In the following sections, transistor level circuits that were used to implement different adders for different design styles are shown. There are repetitions of circuits since some cells were used in other architectures also. The standard cell layouts have been drawn only once at the end of this chapter under logic design styles categories. These cells are stored in a library and used by the designer of the chip

5.1 CELL DESIGN FOR RIPPLE CARRY ADDER

The ripple carry adder topology is shown in Figure 3.1

Fully static CMOS logic

The standard cells (gates) used in the designs are –

FSXOR, FSCARRYOUT, INV.

The standard cell schematics and circuits used for implementing the sum and carry equations using above cells in fully static CMOS logic are shown in Fig. 5.1 (a), and 5.1(b).

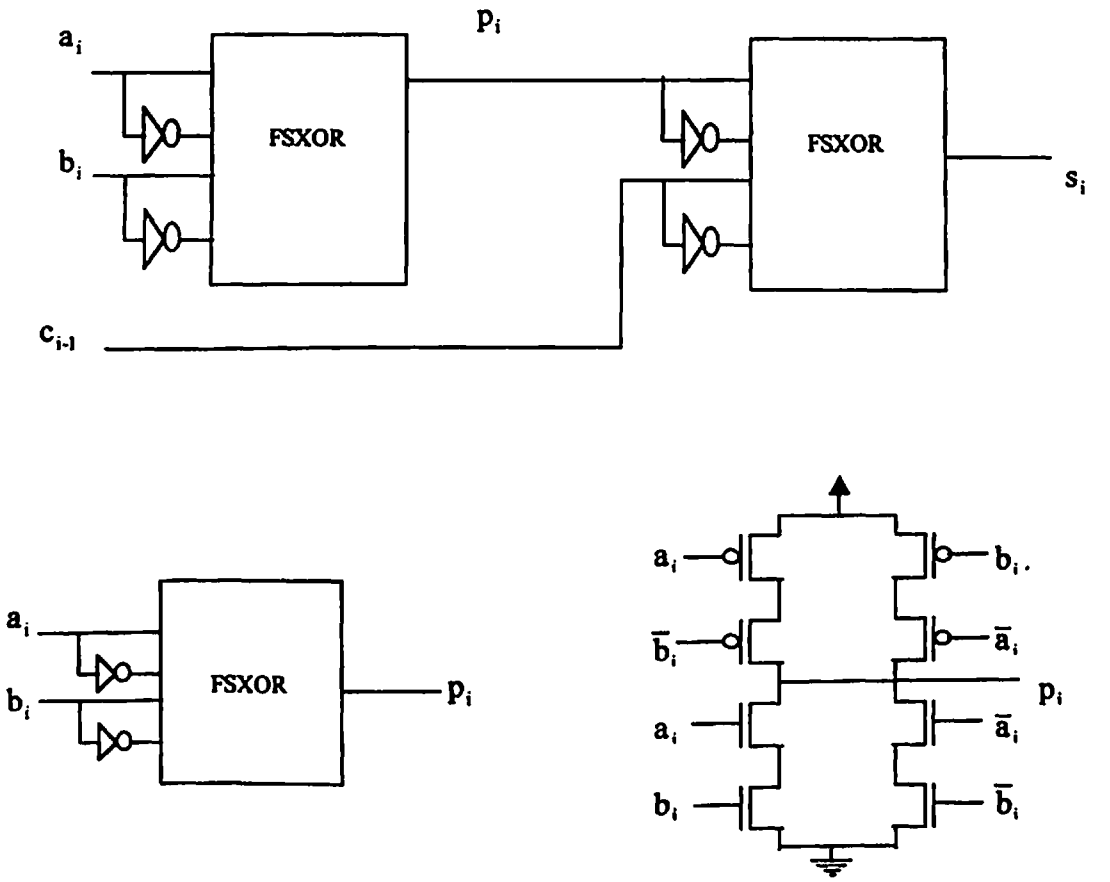


Fig. 5.1. a Sum signal generation circuit and cell-schematic using fully static CMOS logic

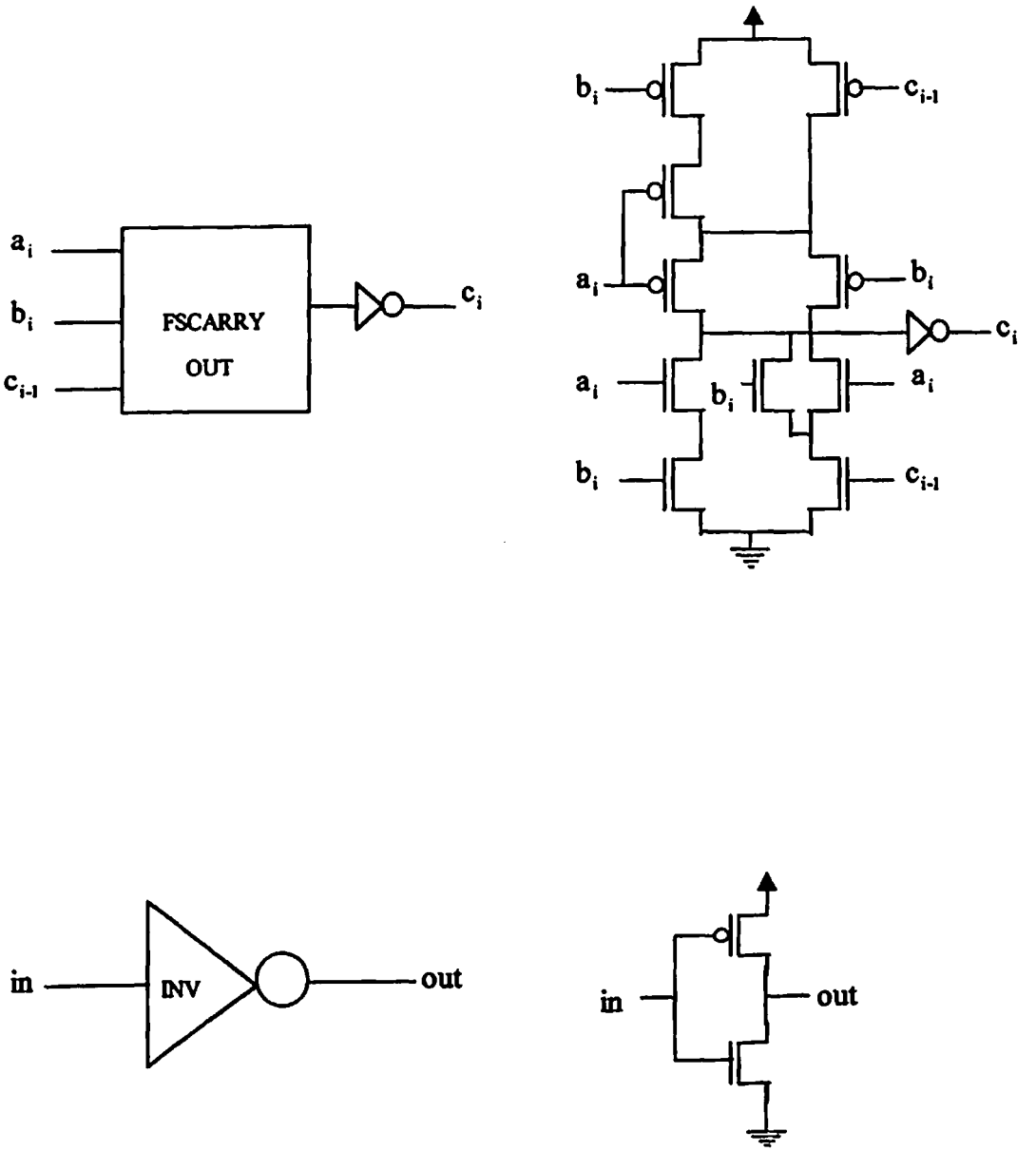


Fig. 5.1. b Carry signal generation circuit and cell-schematic using fully static CMOS logic

Domino CMOS logic

The standard cells used in designs are-

DMXOR, DMXNOR, DMNAND, DMNOR, DMCARRYCOMP, INV

The cell schematics and circuits used for implementing sum and carry equations of ripple carry adder in domino CMOS logic are shown in Fig. 5.2 (a), and 5.2 (b).

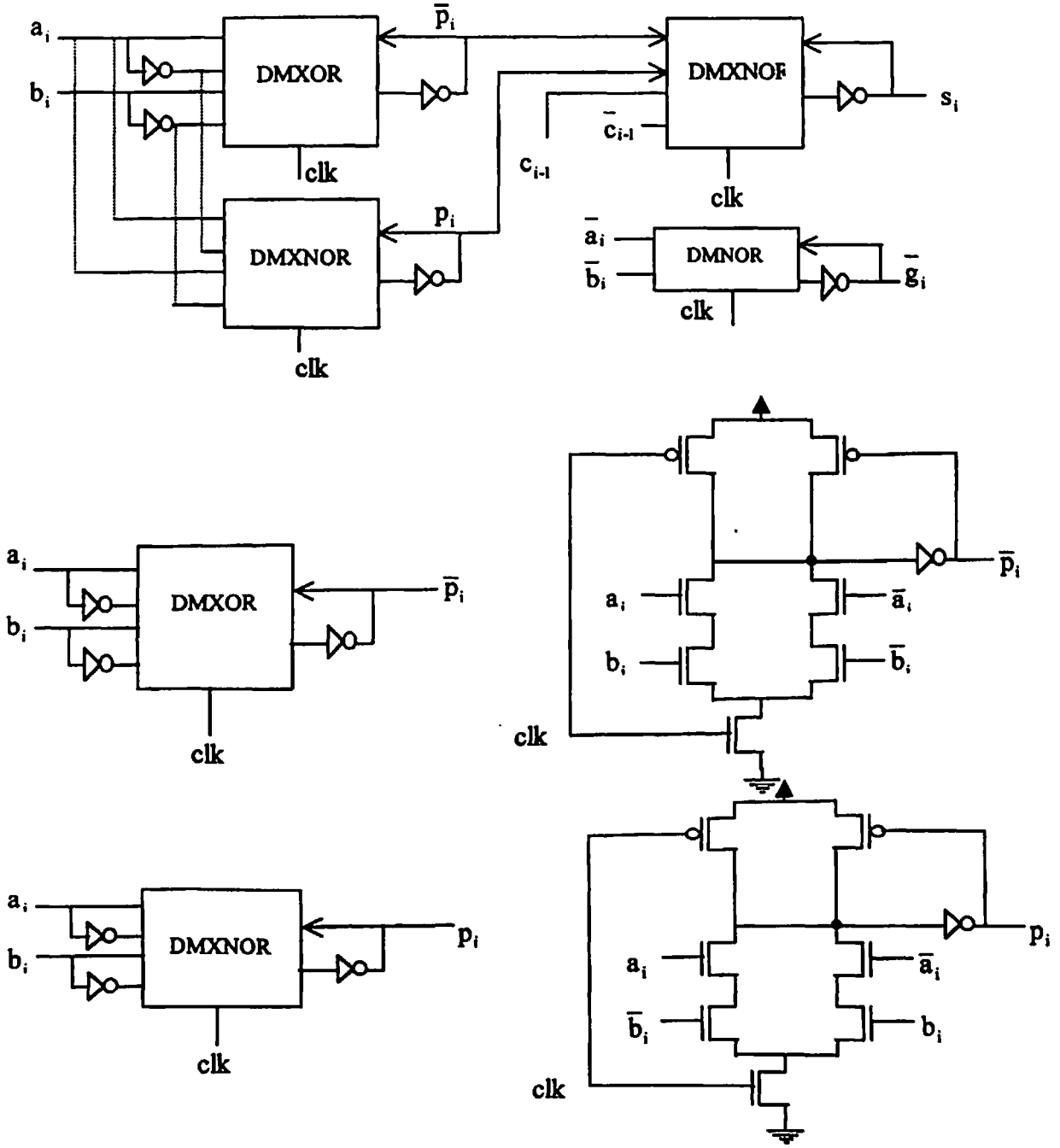


Fig. 5.2. a Sum signal generation circuit and cell-schematic using domino CMOS logic

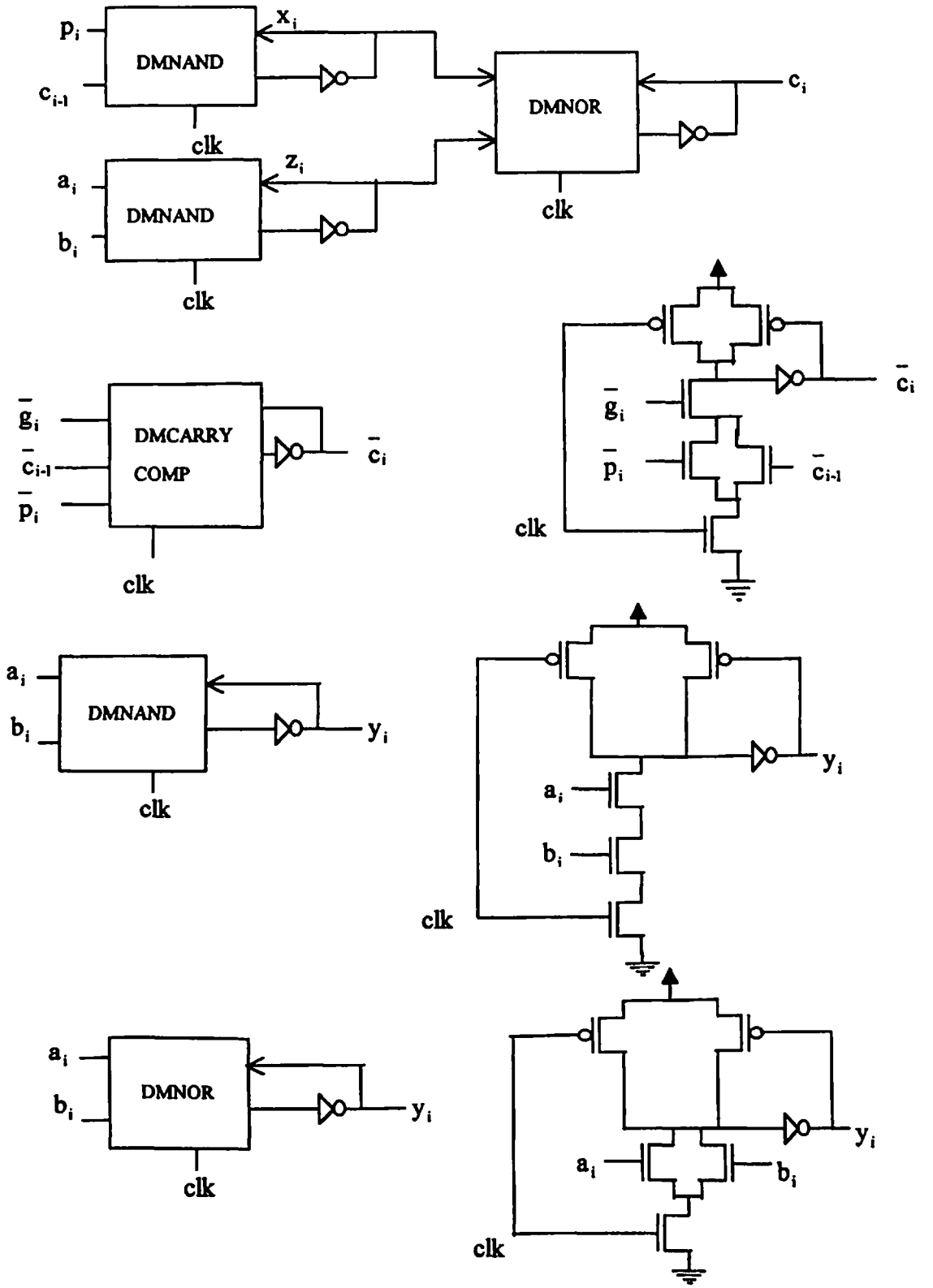


Fig. 5.2. b Carry signal generation circuit and cell-schematic using domino CMOS logic

Complementary pass transistor logic

The standard cells used in the designs are-

CPXOR/XNOR, CPAND/NAND, CPOR/NOR, INV

The cell schematics and circuits used for implementing the sum and carry equations using complementary pass-transistor logic are shown in Fig. 5.3 (a), and 5.3 (b).

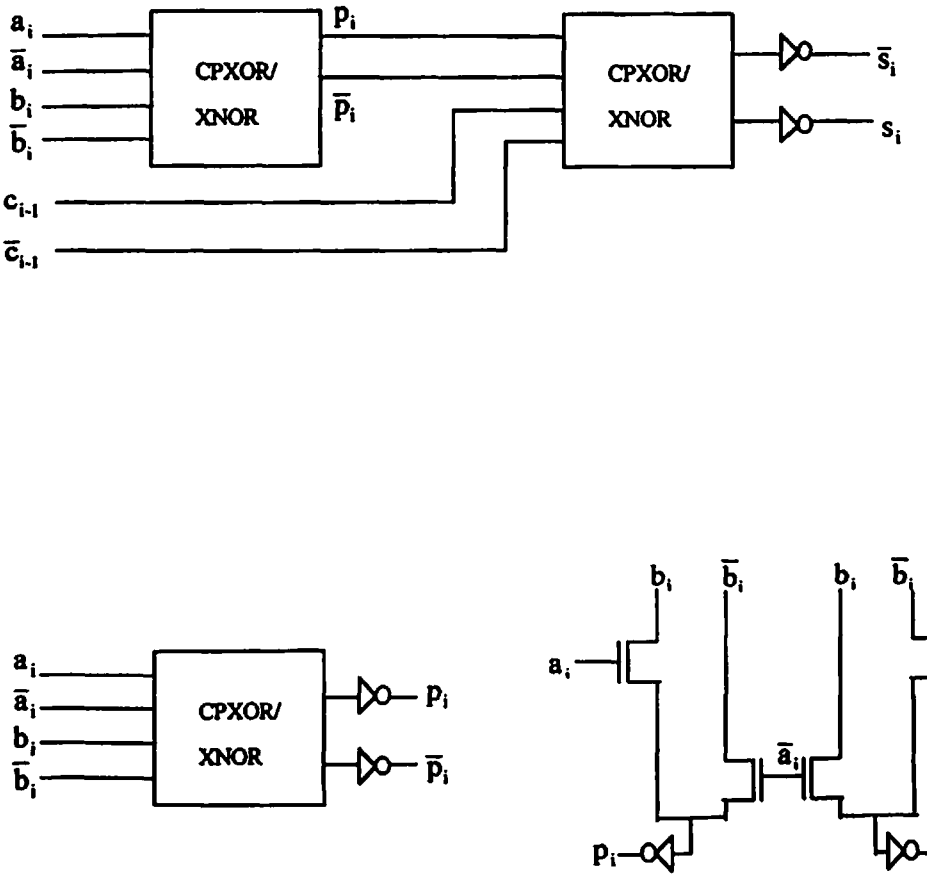


Fig. 5.3. a Sum signal generation circuit and cell-schematic using complementary pass-transistor logic

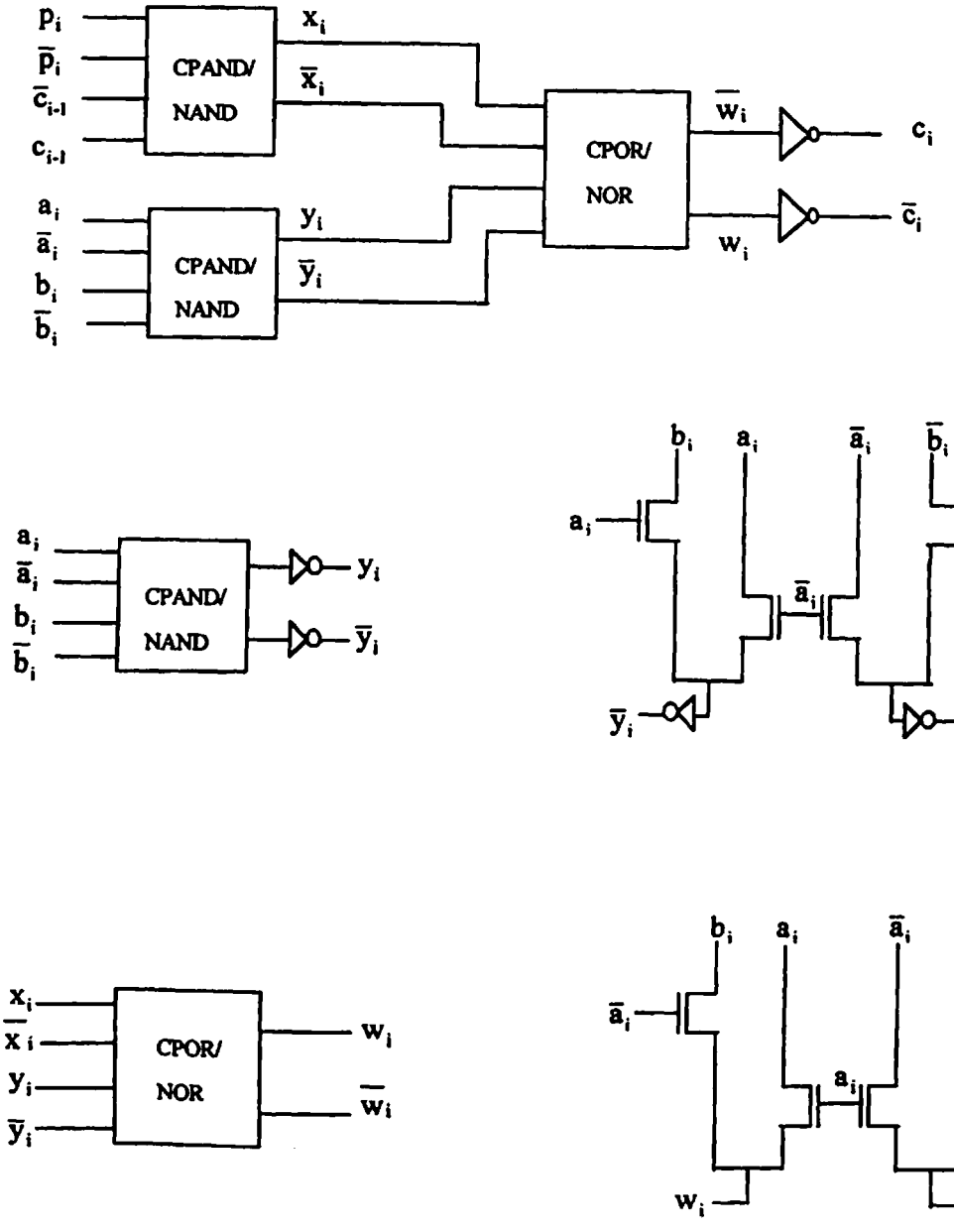


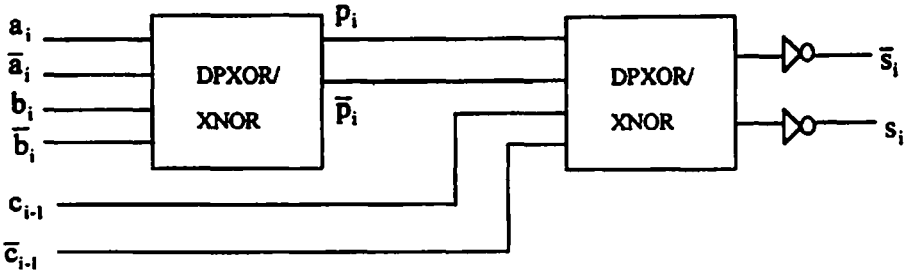
Fig. 5.3. b Carry signal generation circuit and cell-schematic using complementary pass-transistor logic

Dual pass transistor logic

The standard cells used in the designs are-

DPXOR/XNOR, DPAND/NAND, DPOR/NOR, INV

The cell schematics and circuits used for implementing the sum and carry equations using dual pass transistor logic are shown in Fig. 5.4 (a), and 5.4 (b).



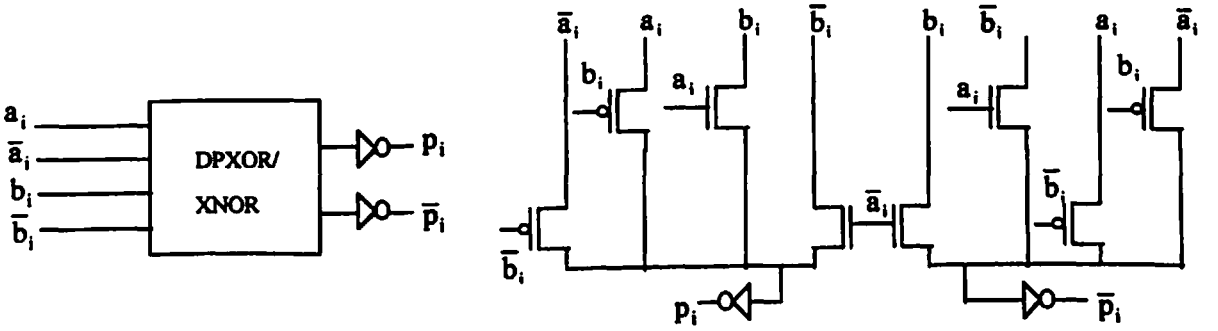


Fig. 5.4. a Sum signal generation circuit and cell-schematic using dual pass transistor logic

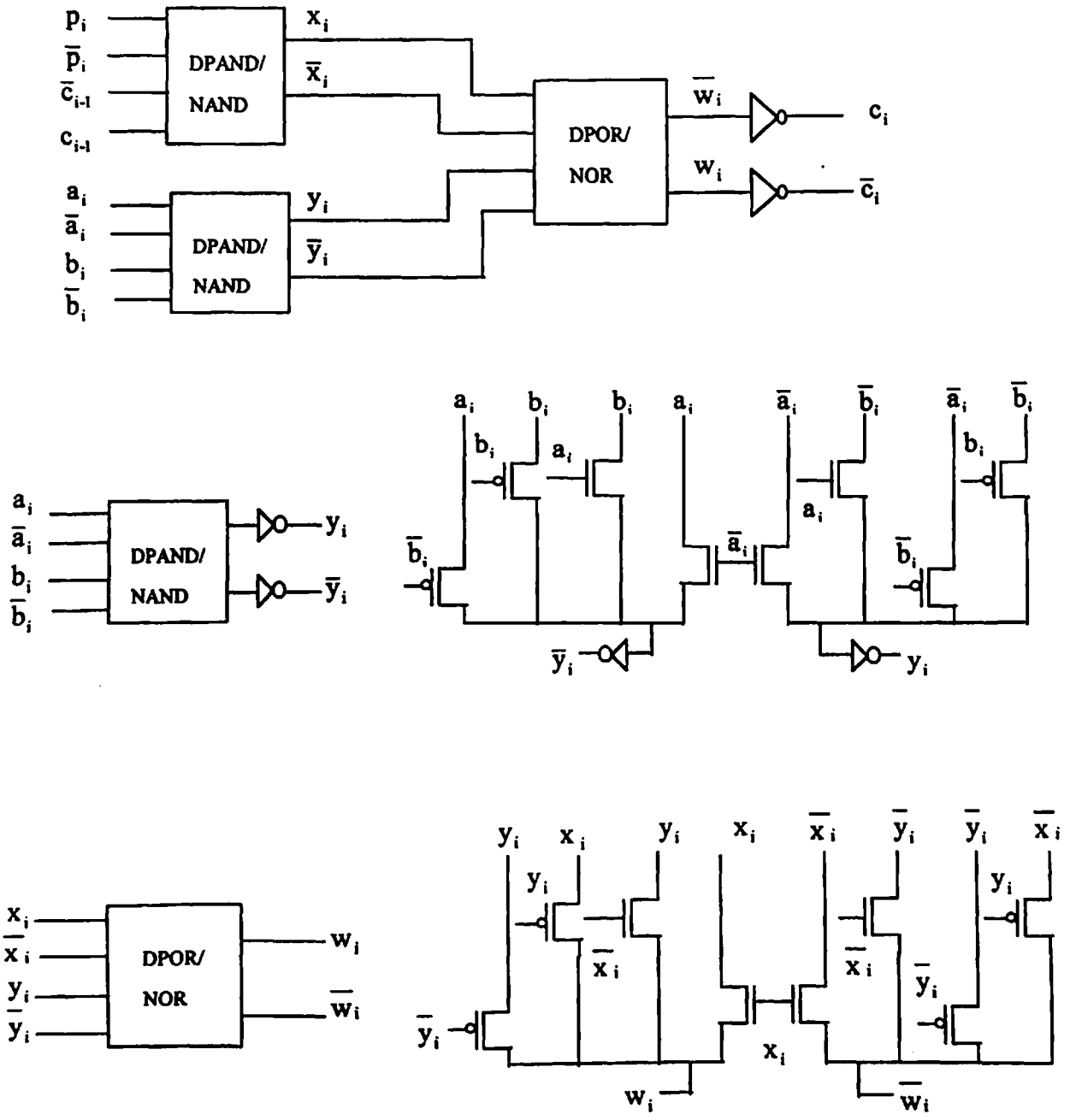


Fig. 5.4. b Carry signal generation circuit and cell-schematic using dual pass transistor logic

5.2 CELL DESIGN FOR CARRY SKIP ADDER

The carry skip adder architecture is shown in Figure 3.3.

Fully static CMOS logic

The standard cells used in the designs are-

FSXOR, FSCARRYOUT, FSNAND4, FSNAND, FSNOR, INV.

The standard cell schematics and circuits used for implementing the sum and carry equations in fully static CMOS logic are the same as shown in Fig. 5.1 (a), and 5.1 (b). The schematics of cells FSNAND4, FSNAND, FSNOR which are used in the carry-skip path for propagating carry-input (c_{in}) to the 4-bit carry skip block are shown in Fig.5.5 (a), and 5.5 (b).

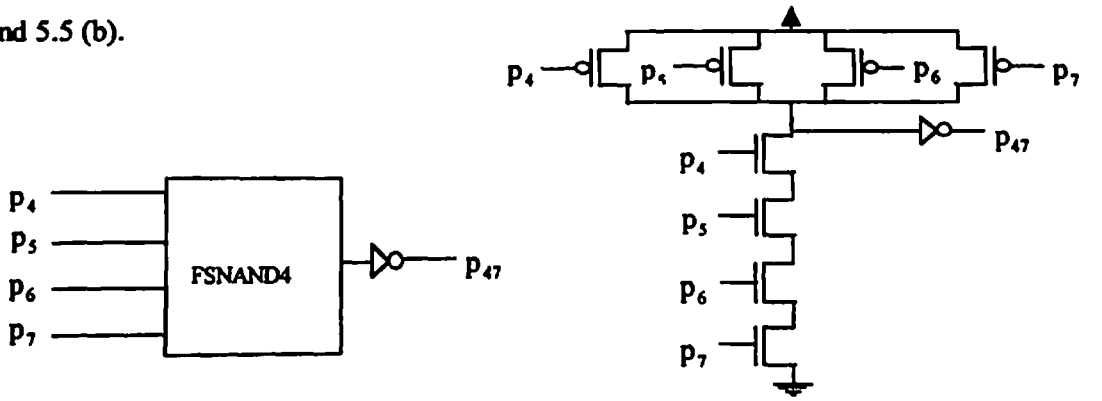


Fig. 5.5. a Group propagate (p_{47}) signal generation circuit using fully static CMOS logic.

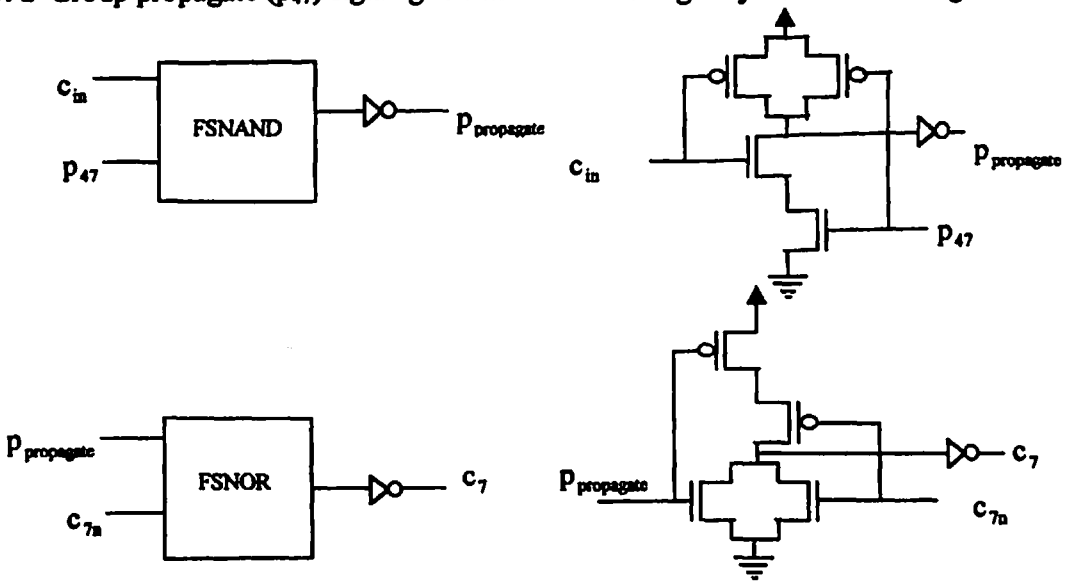


Fig. 5.5. b Carry propagation cells of carry-skip path using fully static CMOS logic.

Domino CMOS logic

The standard cells used in the designs are-

DMXOR, DMNAND4, DMNAND, DMNOR, DMCARRYCOMP, INV.

The standard cell schematics and circuits used for implementing the sum and carry equations in domino CMOS logic are the same as shown in Fig. 5.2 (a), and 5.2 (b). The schematics of cells DMNAND4, DMNAND, DMNOR, which are used in the carry-skip path for propagating carry input (c_{in}) are shown in Fig. 5.6 (a), and 5.6 (b).

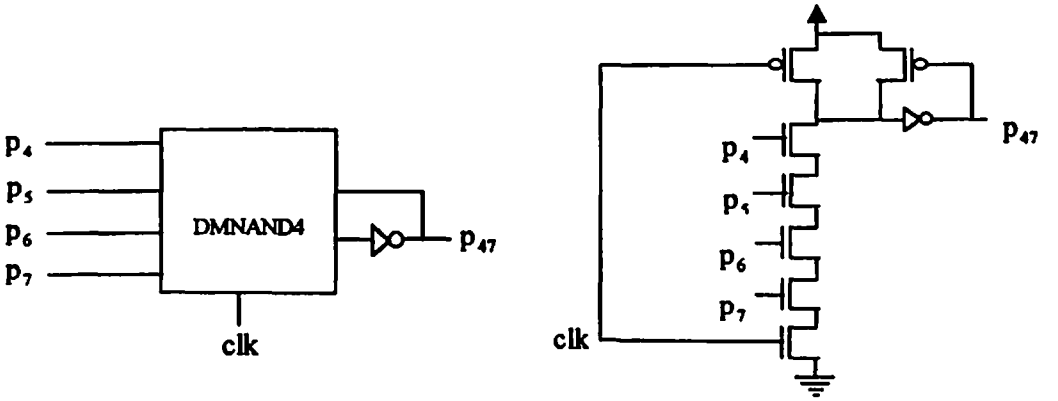


Fig. 5.6. a Group propagate (p_{47}) generation circuit using domino CMOS logic.

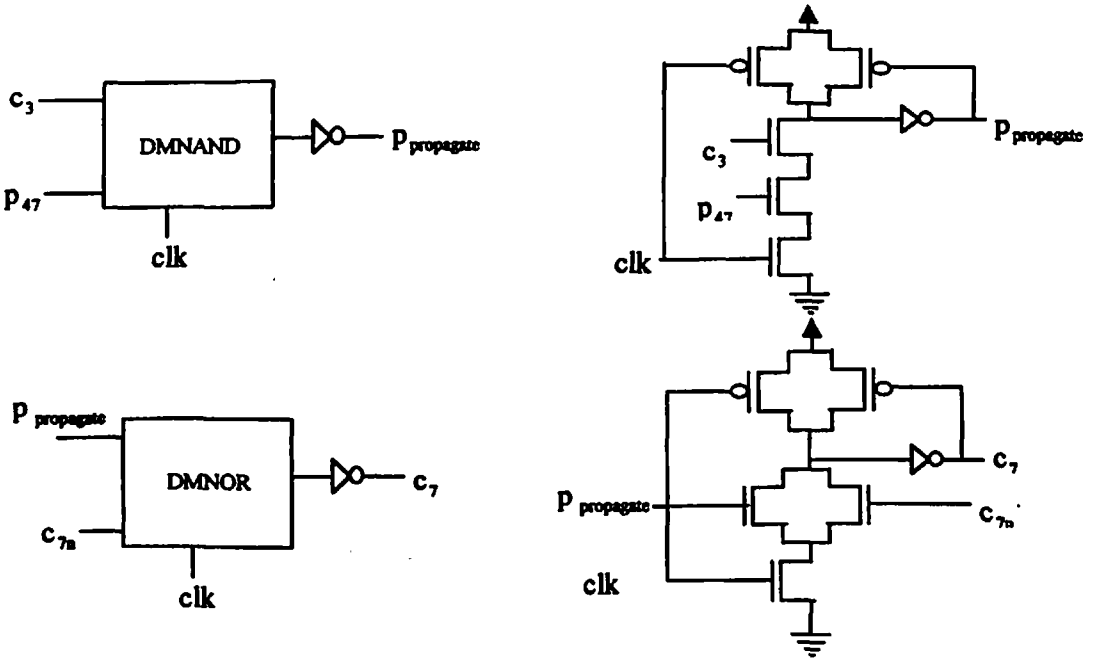


Fig. 5.6. b Carry propagation cells of carry-skip path using domino CMOS logic.

Complementary pass transistor logic

The standard cells used in the designs are-

CPXOR/XNOR, CPAND4/NAND4, CPAND/NAND, CPOR/NOR, INV.

The standard cell schematics and circuits used for implementing the sum and carry equations in domino CMOS logic are the same as shown in Fig. 5.3 (a), and 5.3 (b). The schematics of cells CPAND4/NAND4, CPAND/NAND, CPOR/NOR, which are used in the carry-skip path for propagating carry-input (c_{in}) are shown in Fig. 5.7 (a), and 5.7 (b).

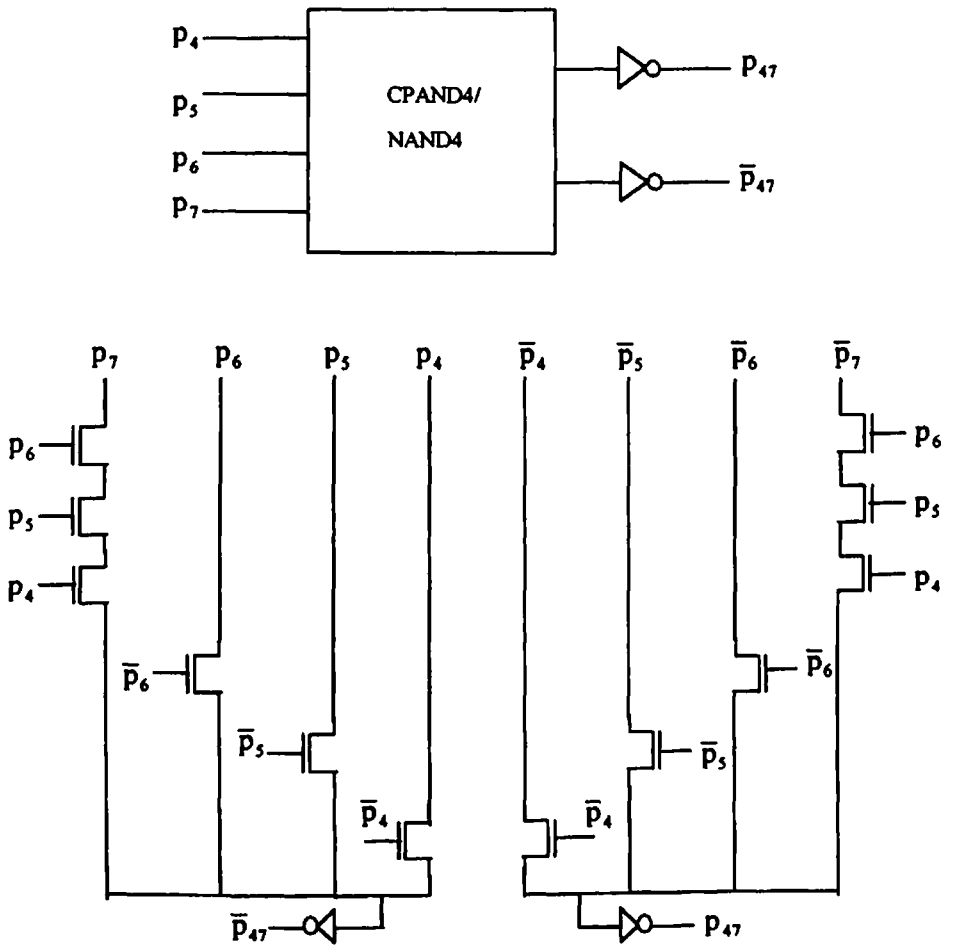


Fig. 5.7. a Group propagate (p_{47}) signal generation circuit using complementary pass transistor logic.

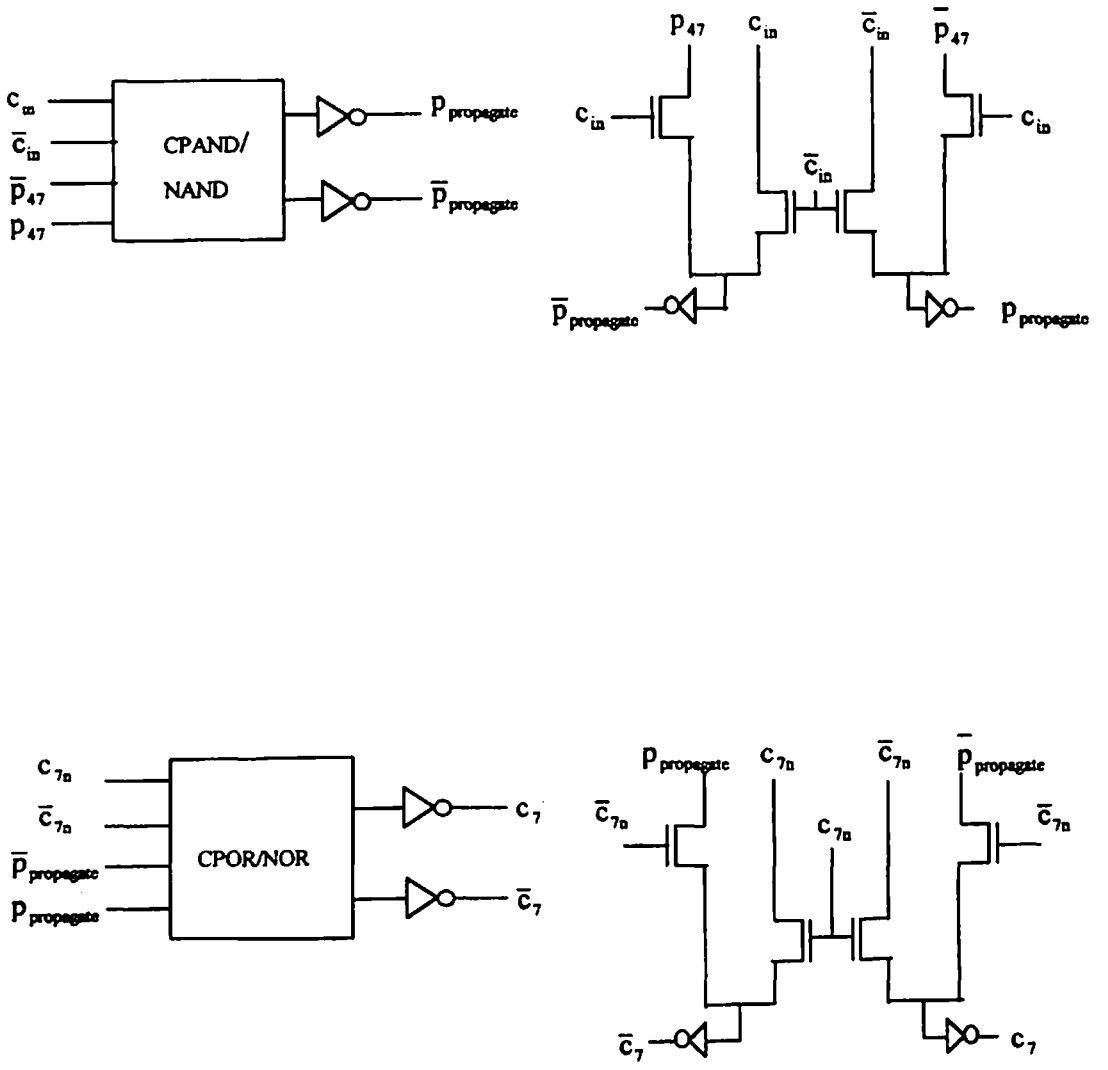


Fig. 5.7. b Carry propagation cells of carry-skip path using complementary pass transistor logic.

Dual pass transistor logic

The standard cells used in the designs are-

DPXOR/XNOR, DPAND4/NAND4, DPAND/NAND, DPOR/NOR, INV.

The standard cell schematics and circuits used for implementing the sum and carry equations in domino CMOS logic are the same as shown in Fig. 5.4 (a), and 5.4 (b). The schematics of cells DPAND4/NAND4, DPAND/NAND, DPOR/NOR, which are used in the carry-skip path for propagating carry-input (c_{in}) are shown in Fig. 5.8 (a), and 5.8 (b).

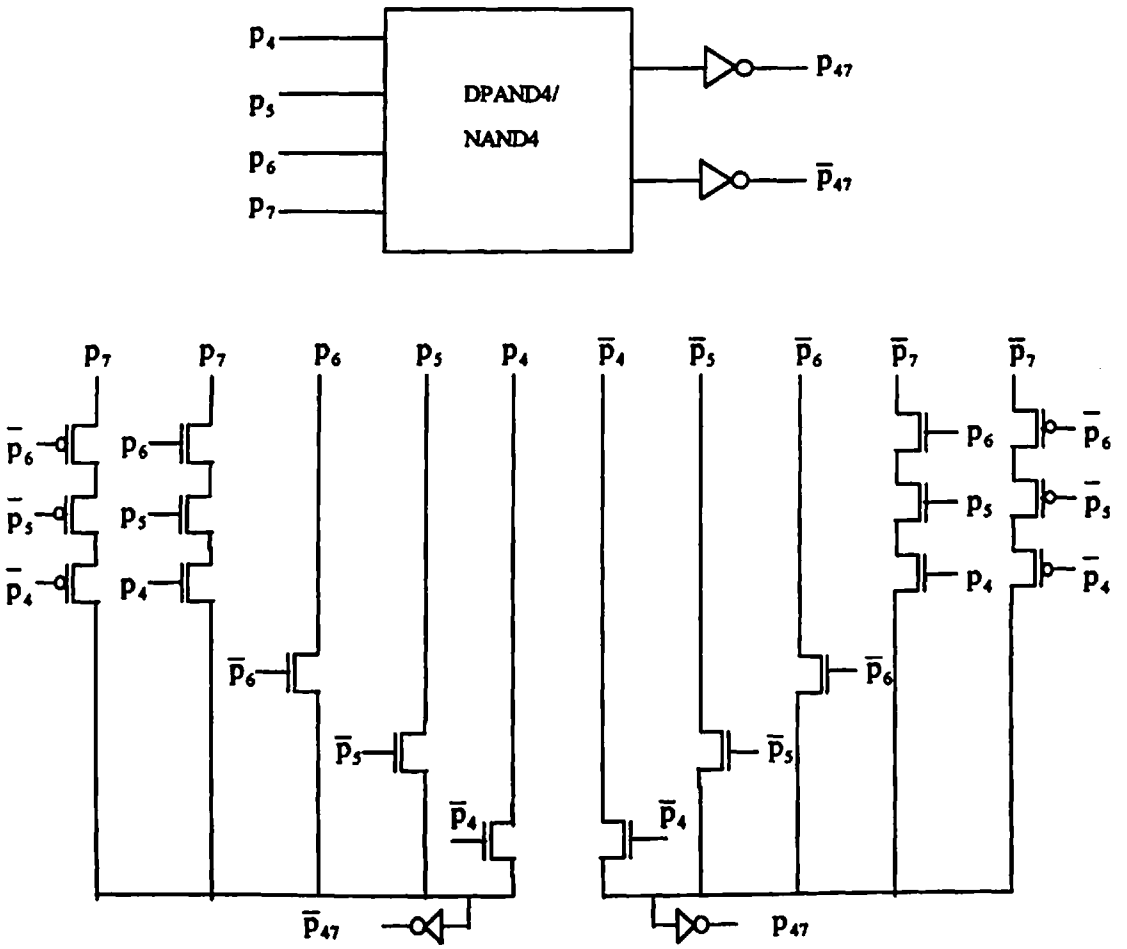


Fig. 5.8. a Group propagate (p_{47}) signal generation circuit using dual pass transistor logic.

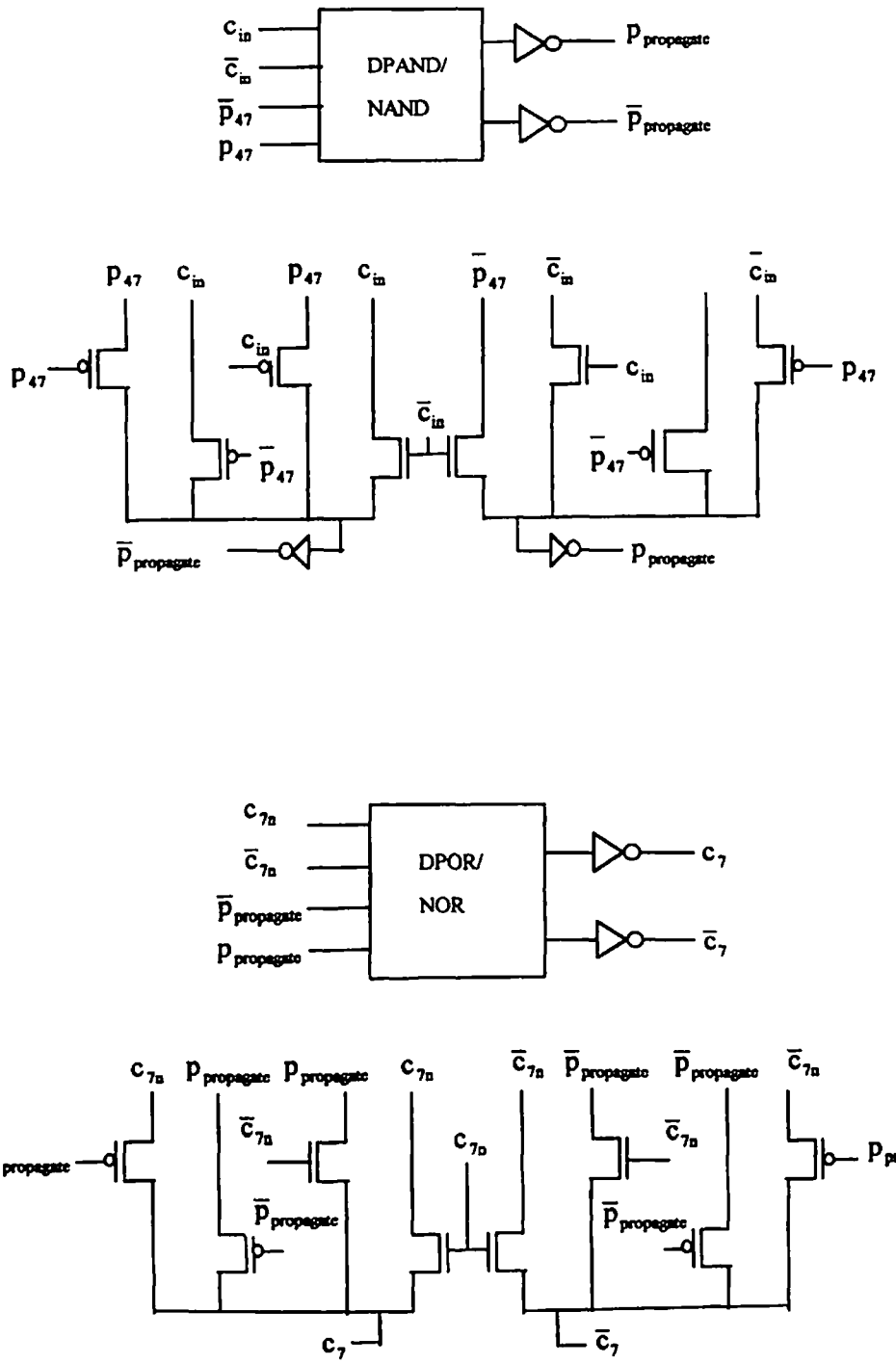


Fig. 5.8. b Carry propagation cells of carry-skip path using dual pass transistor logic.

5.3 CELL DESIGN FOR CARRY SELECT ADDER

The carry select adder is shown in Fig. 3.4

Fully static CMOS logic

The standard cells used in the designs are-

FSXOR, FSCARRYOUT, FSNAND4, FSNOR, FSMUX, INV.

The standard cell schematics and circuits used for implementing the sum and carry equations in fully static CMOS logic are the same as shown in Fig. 5.1 (a), and 5.1 (b). The schematics of cells FSNAND, and FSNOR, which are used to select carry, are same as shown in Fig. 5.5 (b). The schematic of cell FSMUX used to select correct sum signal using 'c_{in}' (carry input to the 4-bit carry select block) as the select signal is shown in Fig. 5.9.

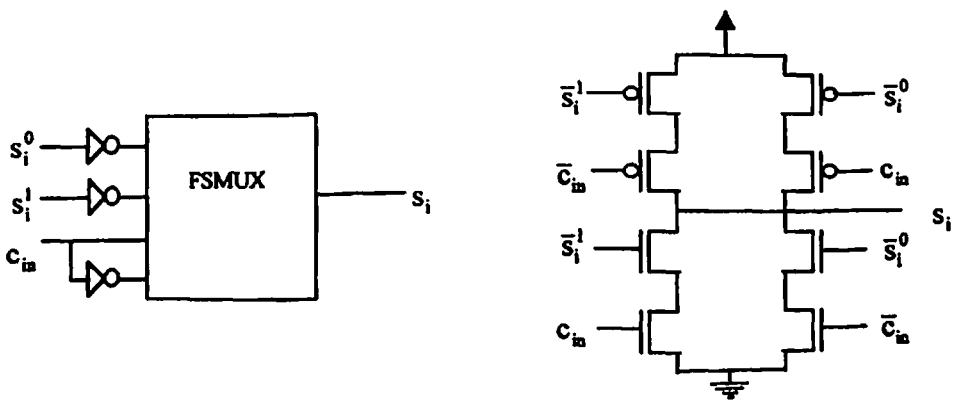


Fig. 5.9 Sum signal selection circuit using fully static CMOS logic

Domino CMOS Logic

The standard cells used in the designs are-

DMXOR, DMNAND4, DMNOR, DMMUX, DMCARRYCOMP, INV.

The standard cell schematics and circuits used for implementing the sum and carry equations in domino CMOS logic are the same as shown in Fig. 5.2 (a), and 5.2 (b). The schematics of cells DMNAND, and DMNOR, which are used to select carry, are same as shown in Fig. 5.6 (b). The schematic of cell DMMUX used to select correct sum signal using c_{in} (carry input to the 4-bit block) as the select signal is shown in Fig. 5.10.

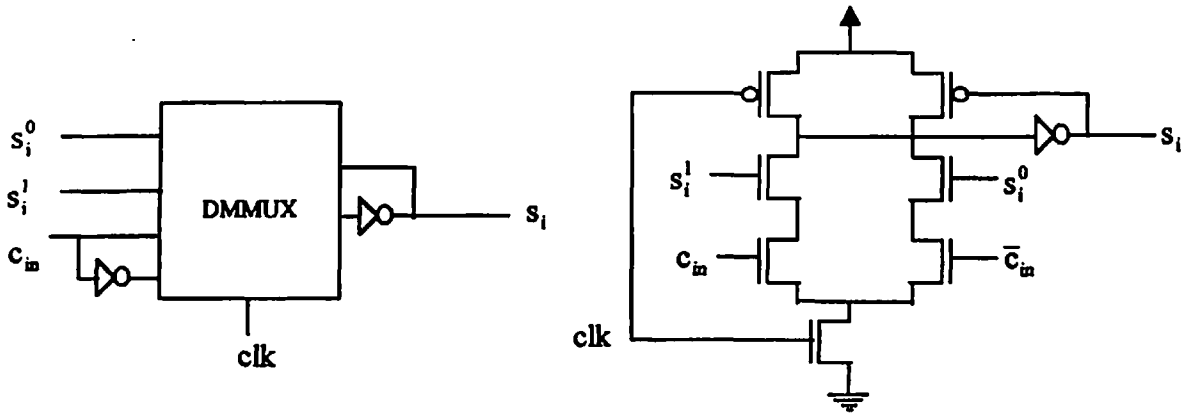


Fig. 5.10 Sum signal selection circuit using domino CMOS logic

Complementary pass transistor logic

The standard cells used in the designs are-

CPXOR/XNOR, CPAND/NAND, CPOR/NOR, CPMUX, INV.

The standard cell schematics and circuits used for implementing the sum and carry equations in complementary pass transistor logic are the same as shown in Fig. 5.3 (a), and 5.3 (b). The schematics of cells CPAND/NAND, and CPOR/NOR, which are used to select carry, are the same as shown in Fig. 5.3 (b). The schematic of cell CPMUX used to select correct sum signal using c_{in} (carry input to the 4-bit block or carry-out of the previous block) as the select signal is shown in Fig. 5.11.

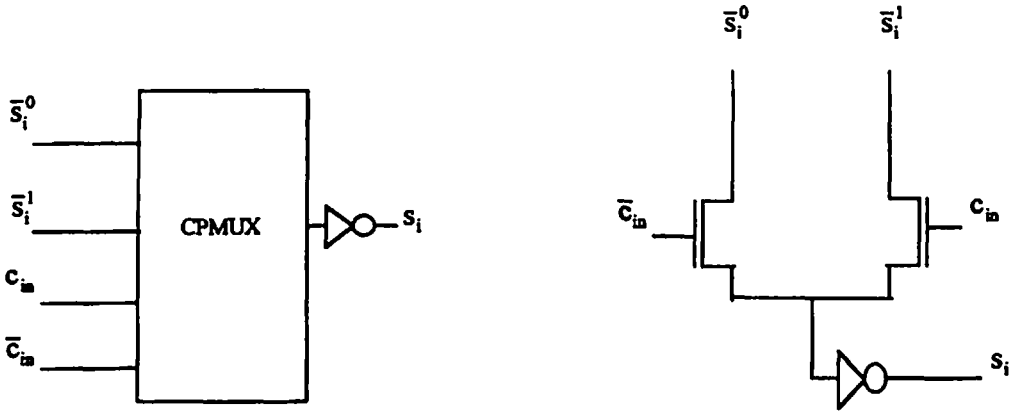


Fig. 5.11 Sum signal selection circuit using complementary pass transistor logic

Dual pass transistor logic

The standard cells used in the designs are-

DPXOR/XNOR, DPAND/NAND, DPOR/NOR, DPMUX, INV.

The standard cell schematics and circuits used for implementing the sum and carry equations in dual pass transistor logic are the same as shown in Fig. 5.4 (a), and 5.4 (b). The schematics of cells DPAND/NAND, and DPOR/NOR, which are used to select carry, are the same as shown in Fig. 5.4 (b). The schematic of cell DPMUX used to select correct sum signal using 'c_{in}' (carry input to the 4-bit block or carry-out of the previous block) as the select signal is shown in Fig. 5.12.

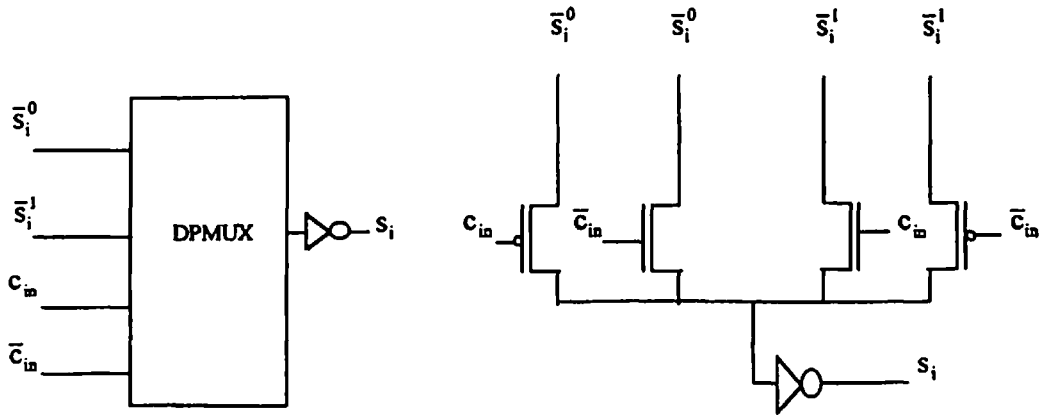


Fig. 5.12 Sum signal selection circuit using dual pass transistor logic

5.4 CELL DESIGN FOR CONDITIONAL SUM ADDER

The conditional sum adder is shown in Fig. 3.4.

Fully static CMOS Logic

The standard cells used in the designs are-

FSXNOR, FSXOR, FSNAND, FSNOR, FSMUX, INV.

The standard cell schematics and circuits used for generating sum and carry signals are given in Fig. 5.13 (a), 5.13 (b), 5.13 (c), 5.13 (d), and 5.13 (e).

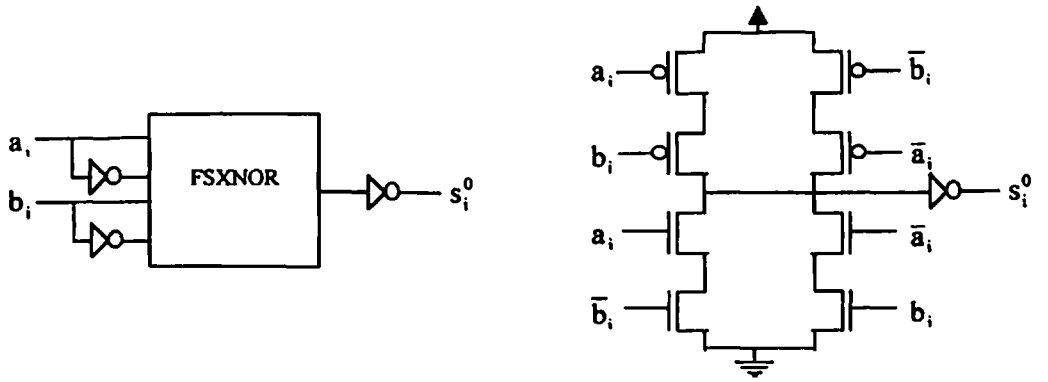


Fig. 5.13. a Sum signal generation circuit with input carry '0' using fully static CMOS logic.

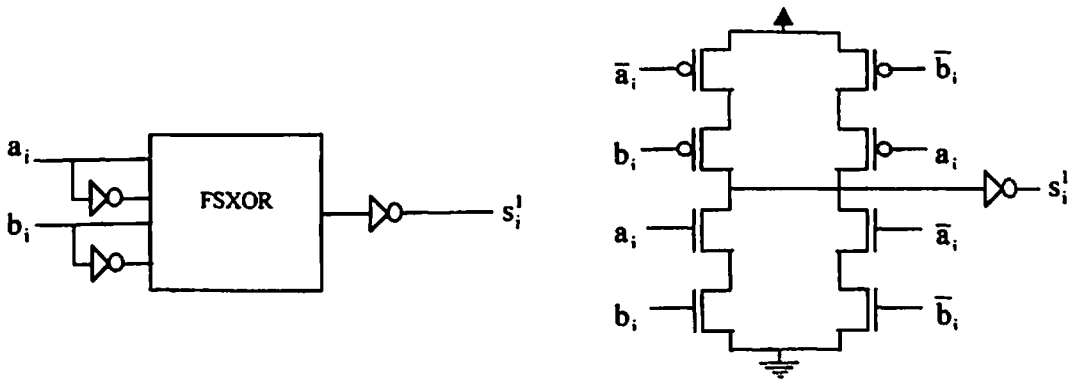


Fig. 5.13. a Sum signal generation circuit with input carry '1' using fully static CMOS logic.

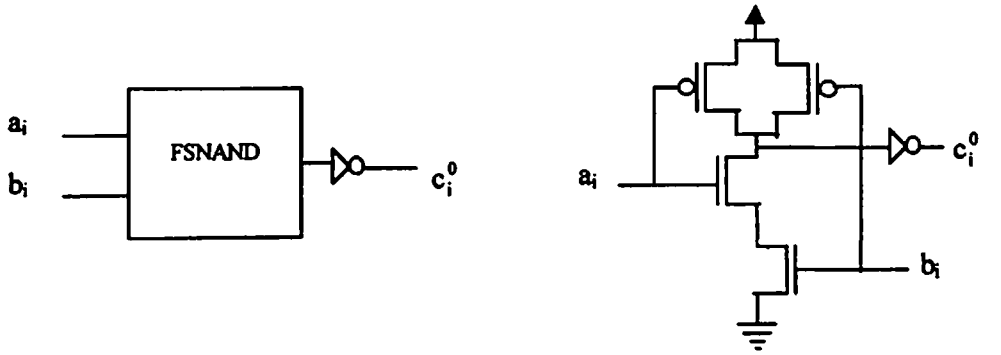


Fig. 5.13. c Carry signal generation circuit with input carry '0' using fully static CMOS logic.

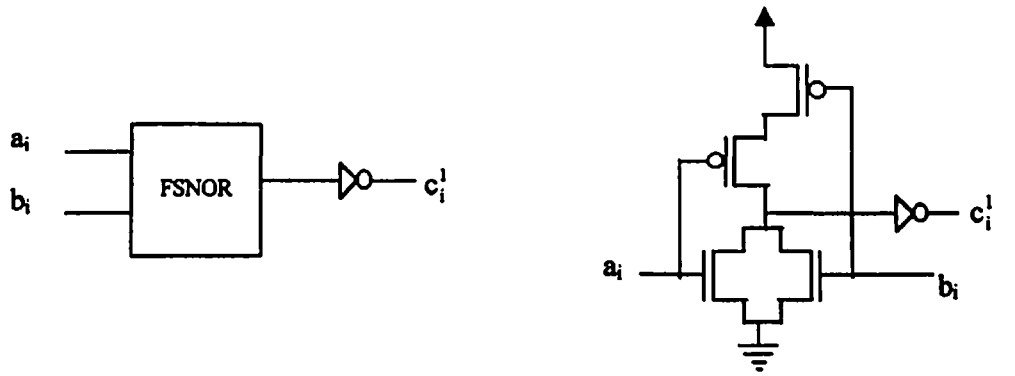


Fig. 5.13. d Carry signal generation circuit with input carry '1' using fully static CMOS logic.

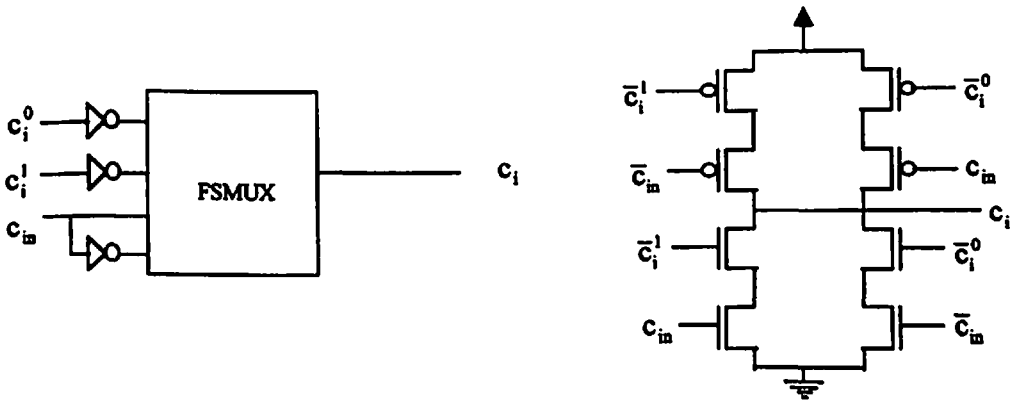


Fig. 5.13. e Sum (and carry) signal selection circuit (multiplexer) using fully static CMOS logic

Domino CMOS Logic

The standard cells used in the designs are-

DMXNOR, DMXOR, DMNAND, DMNOR, DMMUX, INV.

The standard cell schematics and circuits used for generating sum and carry signals are given in Fig. 5.14 (a), 5.14 (b), 5.14 (c), 5.14 (d), and 5.14 (e).

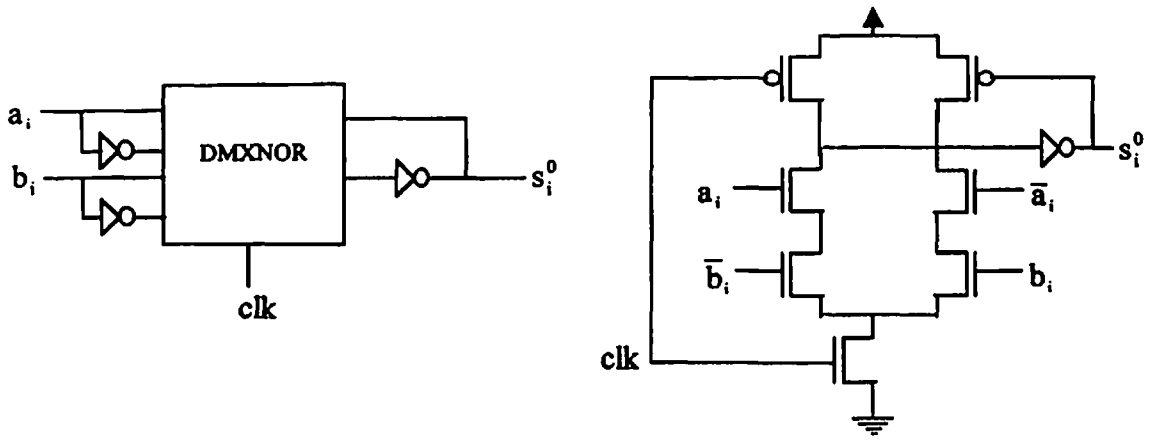


Fig. 5.14. a Sum signal generation circuit with input carry '0' using domino CMOS logic.

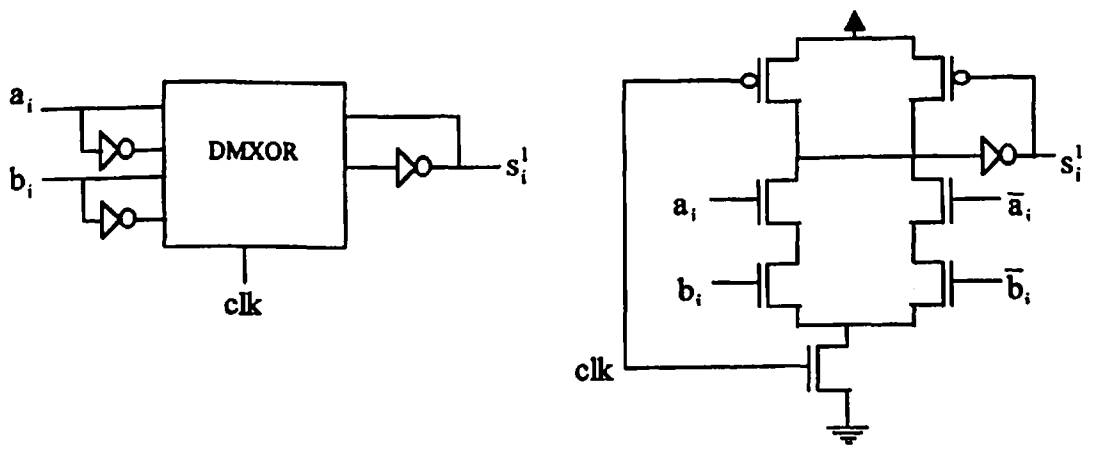


Fig. 5.14. b Sum signal generation circuit with input carry '1' using domino CMOS logic.

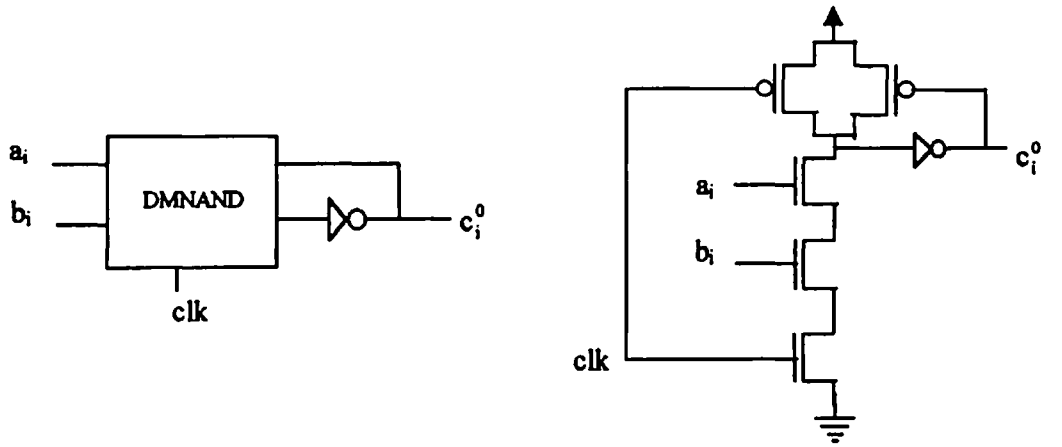


Fig. 5.14. c Carry signal generation circuit with input carry '0' using domino CMOS logic.

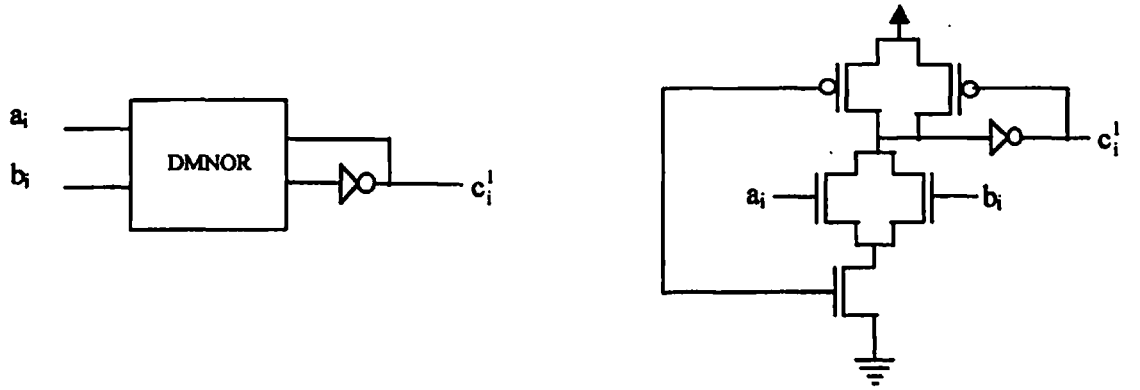


Fig. 5.14. d Carry signal generation circuit with input carry '1' using domino CMOS logic.

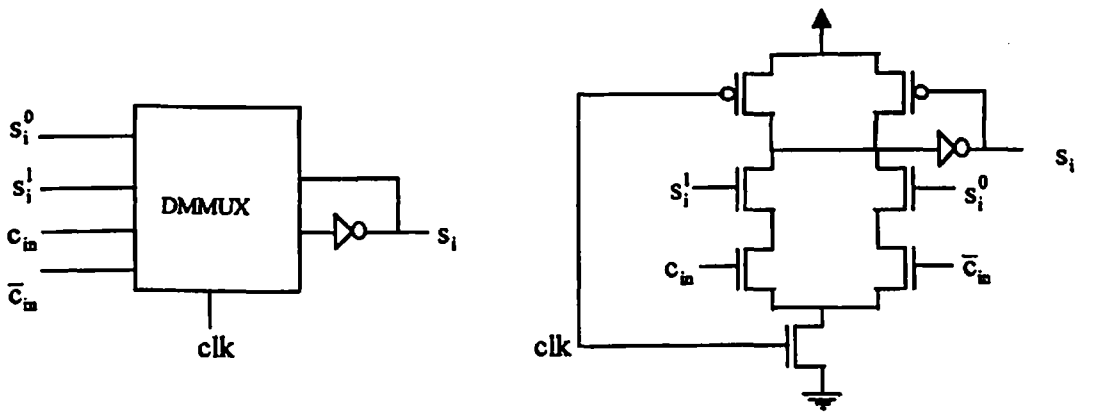


Fig. 5.14. e Sum (and carry) signal selection circuit (multiplexer) using domino CMOS logic

Complementary pass transistor logic

The standard cells used in the designs are-

CPCONDITIONAL-CELL, CPMUX, INV.

The standard cell schematics and circuits used for generating sum and carry signals are given in Fig. 5.15 (a), and 5.15 (b).

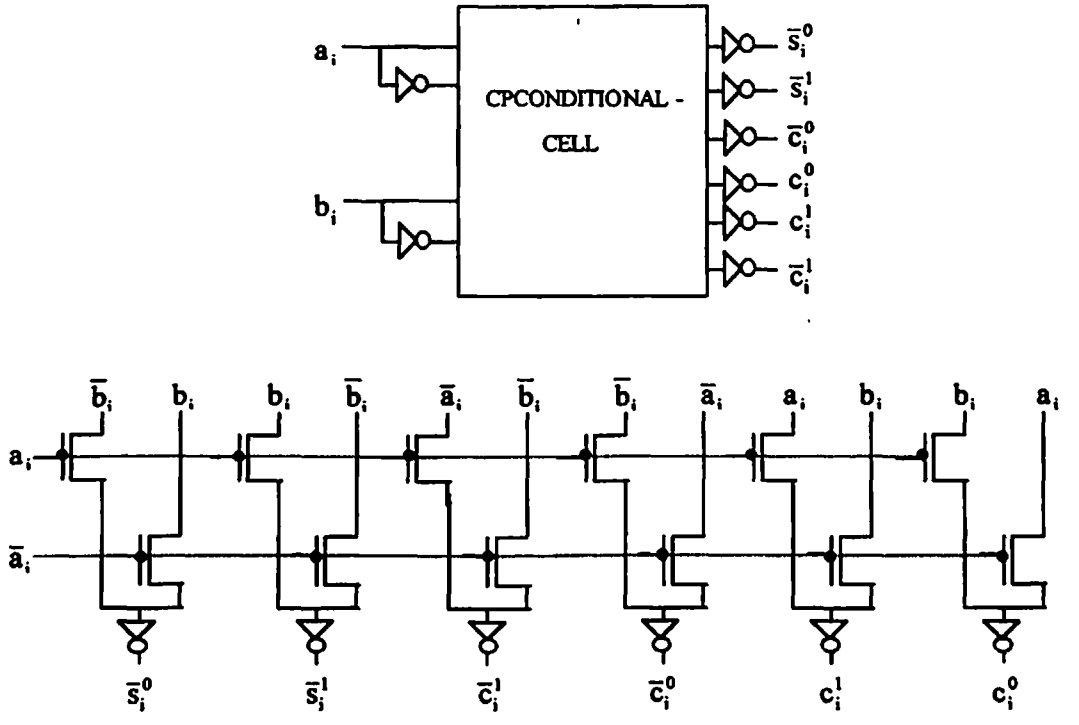


Fig. 5.15. a Conditional cell circuit for generation of conditional sum and carry signals using complementary pass transistor logic

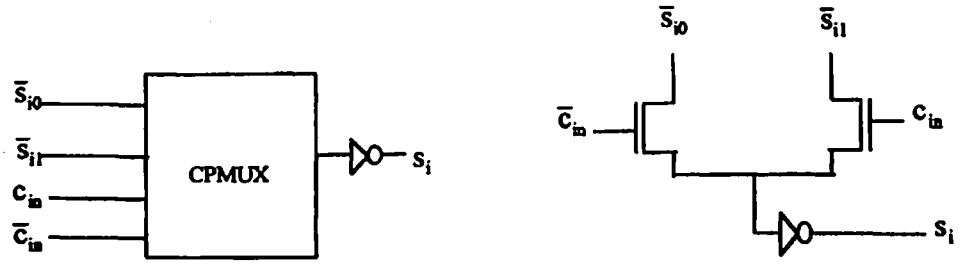


Fig. 5.15. b Sum (and carry) signal selection circuit (multiplexer) using complementary pass transistor logic

Dual pass transistor logic

The standard cells used in the designs are-

DPCONDITIONAL-CELL, DPMUX, INV.

The standard cell schematics and circuits used for generating sum and carry signals are given in Fig. 5.16 (a), and 5.16 (b).

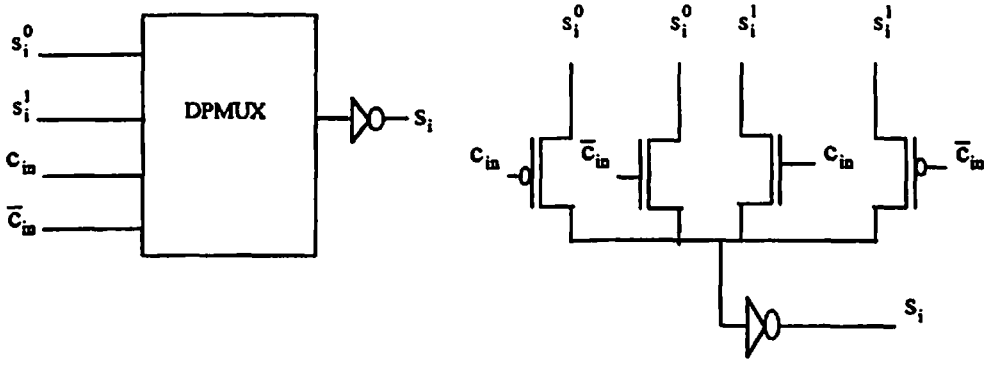


Fig. 5.16. a Sum (and carry) signal selection circuit (multiplexer) using complementary pass transistor logic

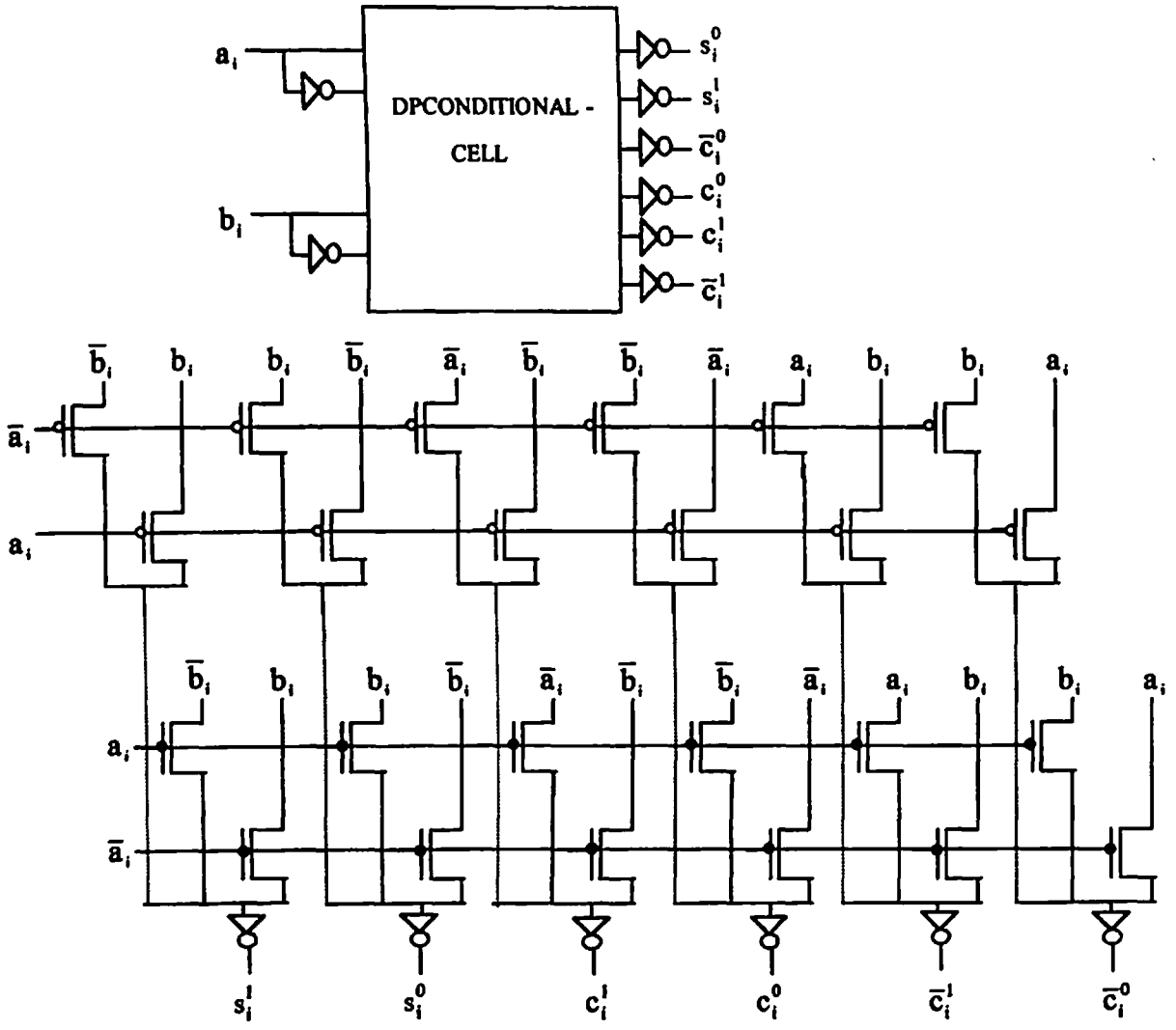


Fig. 5.16. b Conditional cell circuit for generation of conditional sum and carry signals using dual pass transistor logic

5.5 CELL DESIGN FOR CARRY LOOK-AHEAD ADDER

The carry look-ahead adder architecture is shown in Fig. 3.5.

Fully static CMOS logic

The standard cells used in the designs are-

FSXNOR, FSNAND, FSPROP, FSGENCARRY.

The circuits for implementing propagate function 'p_i' and sum signal 's_i' are the same as shown in Fig. 5.1 (a). The cell-schematic of 'FSNAND' used in implementing generate function is the same as shown in 5.5 (b). The circuits implementing A-cell, B-cell, group propagate, and group generate (and carry) functions are shown in Fig. 5.17 (a), 5.17 (b), 5.17 (c), 5.17 (d).

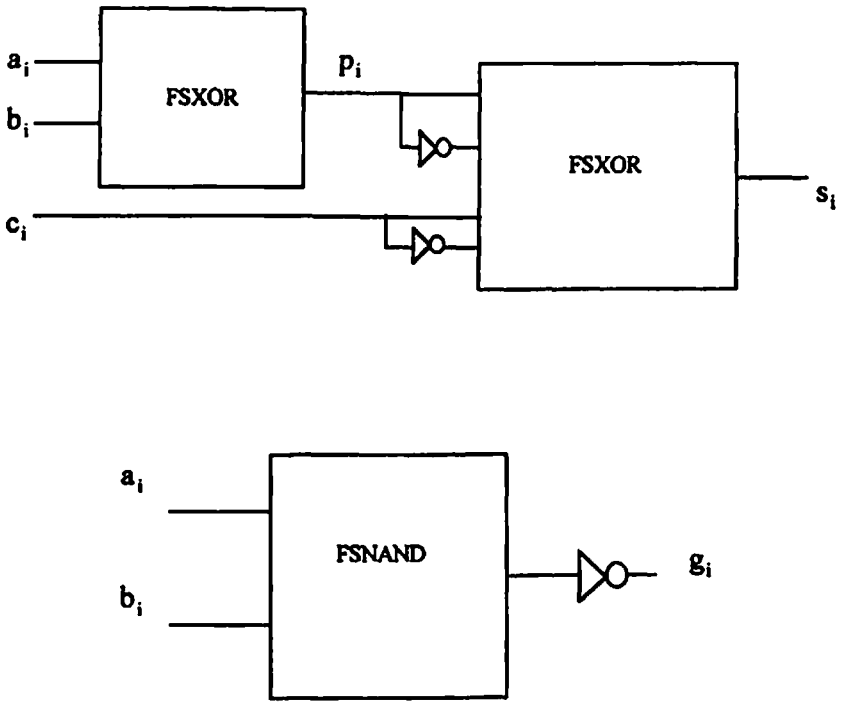


Fig. 5.17. a Block diagram of A-cell and generate signal generation circuit using fully static CMOS logic

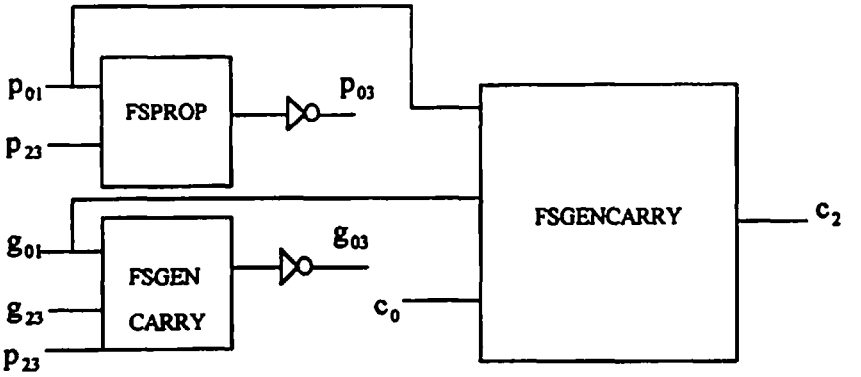


Fig. 5.17 b Block diagram of B-cell

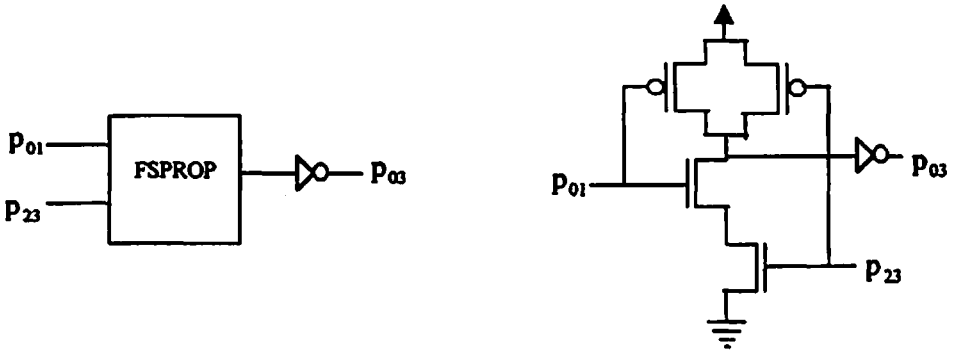


Fig. 5.17 c Group propagate signal generation circuit using fully static CMOS logic.

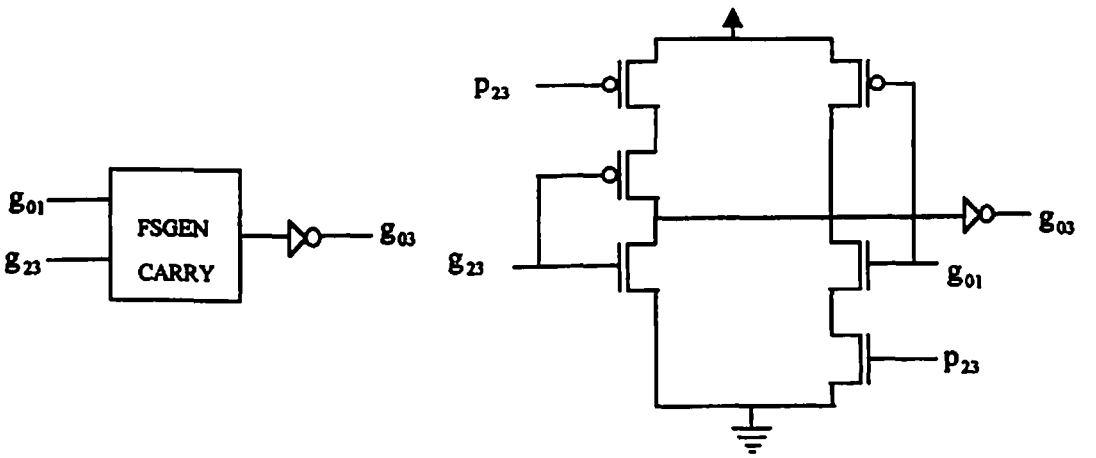


Fig. 5.17 d Group generate (and carry) signal generation circuit using fully static CMOS logic.

Domino CMOS logic

The standard cells used in the designs are-

DMXOR, DMXNOR, DMNAND, DMPROP, DMGENCARRY, DMCARRYCOMP, INV.

The circuits for generating propagate 'p_i' and sum 's_i' signals are the same as shown in Fig. 5.2 (a). The block diagrams of A-cell and B-cell in domino CMOS logic are shown in Fig. 5.18 (a), and 5.18 (b). The schematic of standard cell 'DMNAND' used for implementing generate 'g_i' functions is the same as shown in Fig.5.2 (b). The schematics of standard cell used for implementing group generate (and carry), carry complement, and group propagate are shown in Fig.5.18 (c), 5.18 (d), and 5.18 (e).

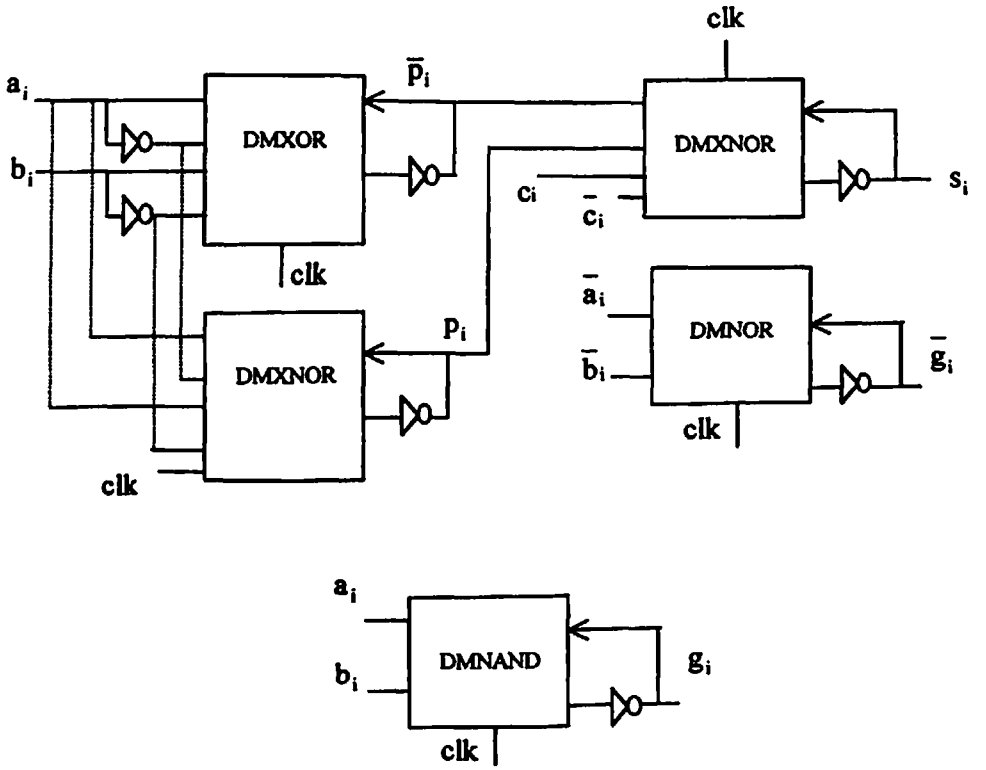


Fig. 5.18. a Block diagram of A-cell

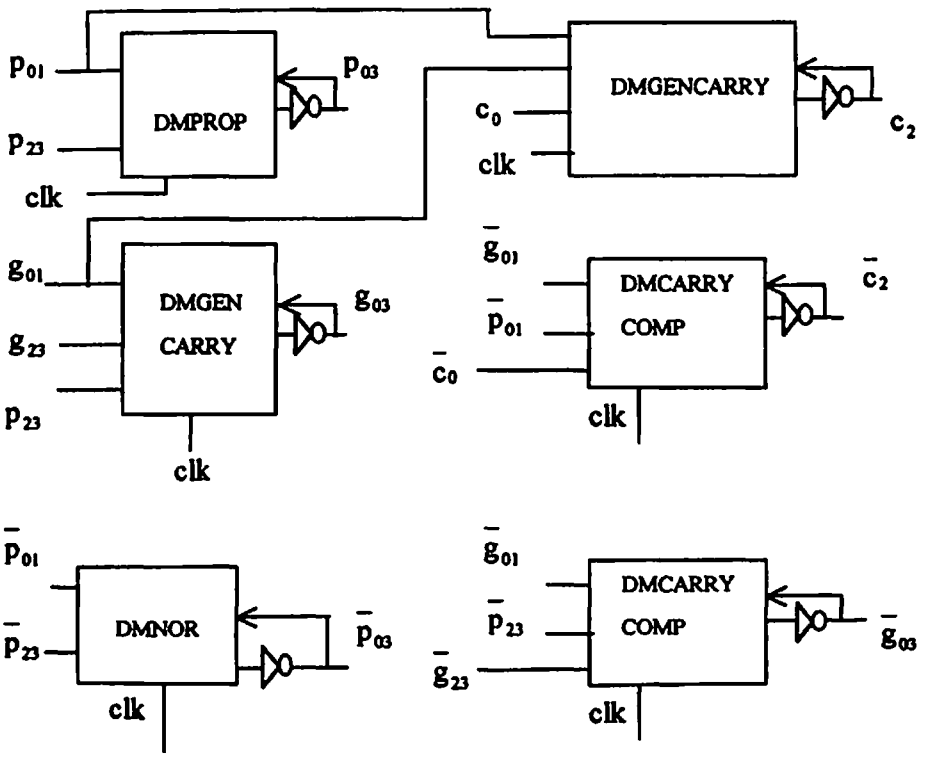


Fig. 5.18. b Block diagram of B-cell

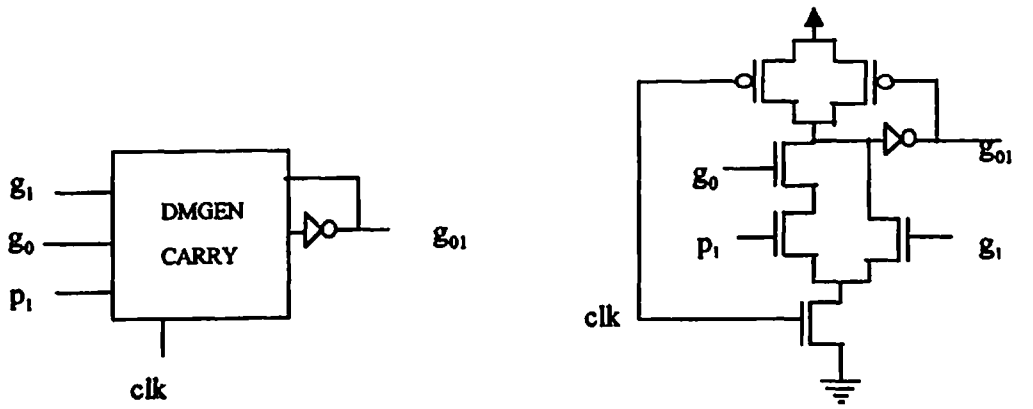


Fig. 5.18. c. Group generate (and carry) signal generation circuit using domino CMOS logic.

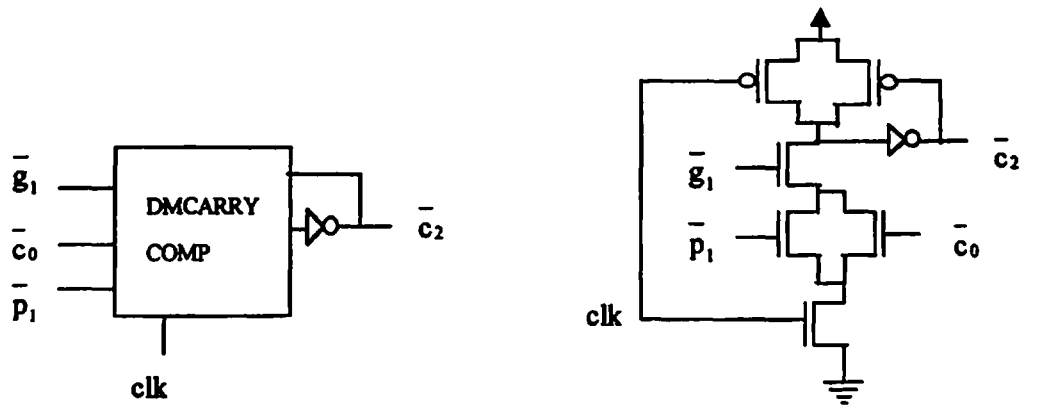


Fig. 5.18. d. Complement carry signal generation circuit using domino CMOS logic

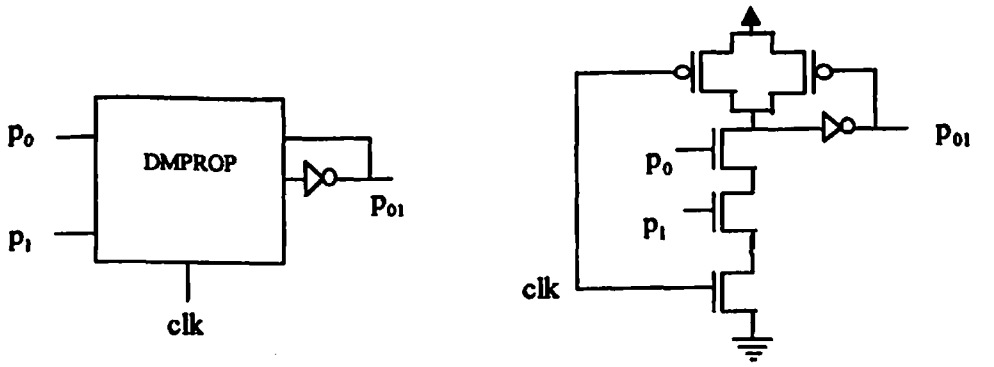


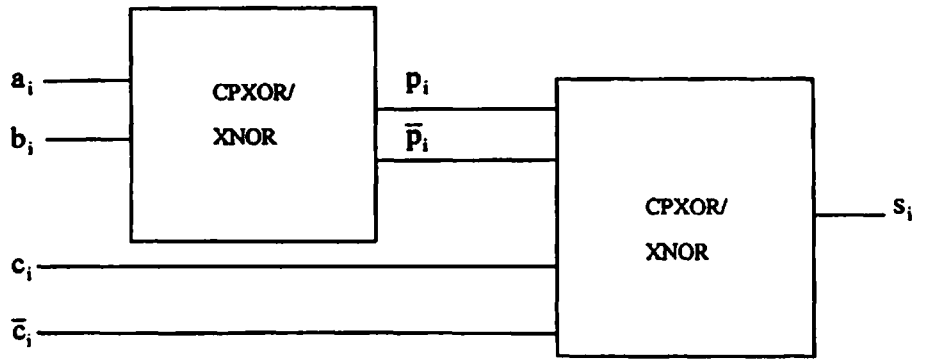
Fig. 5.18. e. Group propagate signal generation circuit using domino CMOS logic

Complementary pass transistor logic

The standard cells used in the designs are-

CPXOR/XNOR, CPAND/NAND, CPPROP, CPGENCARRY, INV.

The circuits for generating propagate 'p_i' and sum 's_i' signals are the same as shown in Fig. 5.3 (a). The block diagrams of A-cell and B-cell in domino CMOS logic are shown in Fig. 5.19 (a), and 5.19 (b). The schematic of standard cell 'CPAND/NAND' used for implementing generate 'g_i' function is the same as shown in Fig.5.3 (b). The circuits for generating group propagate and group generate (and carry) signals are shown in Fig. 5.19 (c), and 5.19 (d).



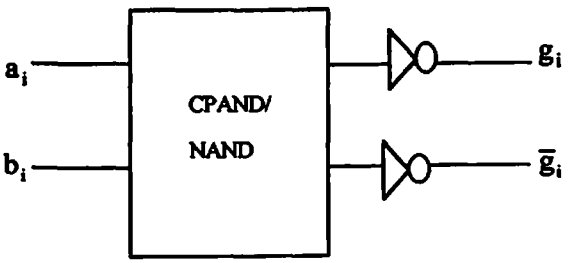


Fig. 5.19. a Block diagram of A-cell

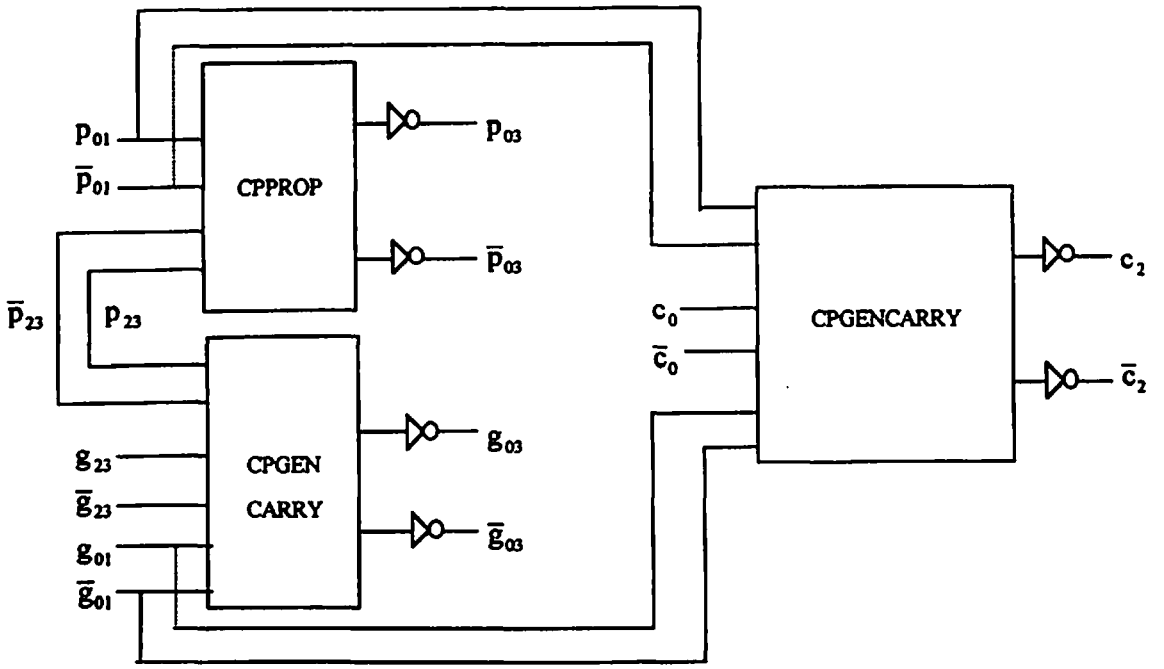


Fig. 5.19. b Block diagram of B-cell

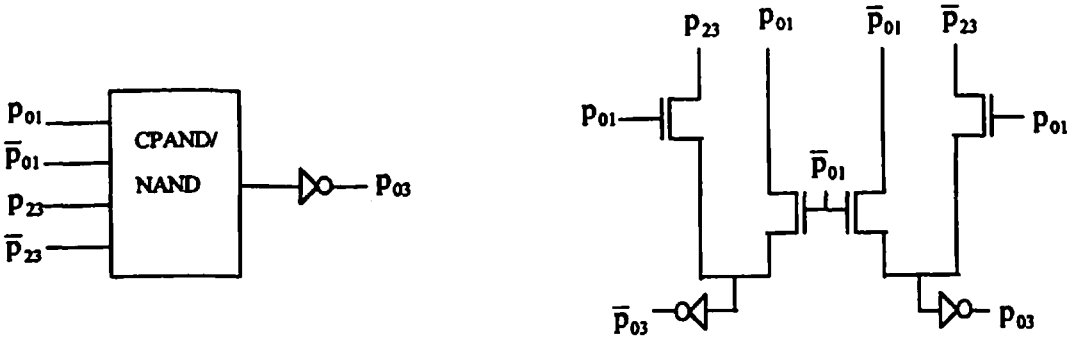


Fig. 5.19. c Group propagate signal generation circuit using complementary pass transistor logic

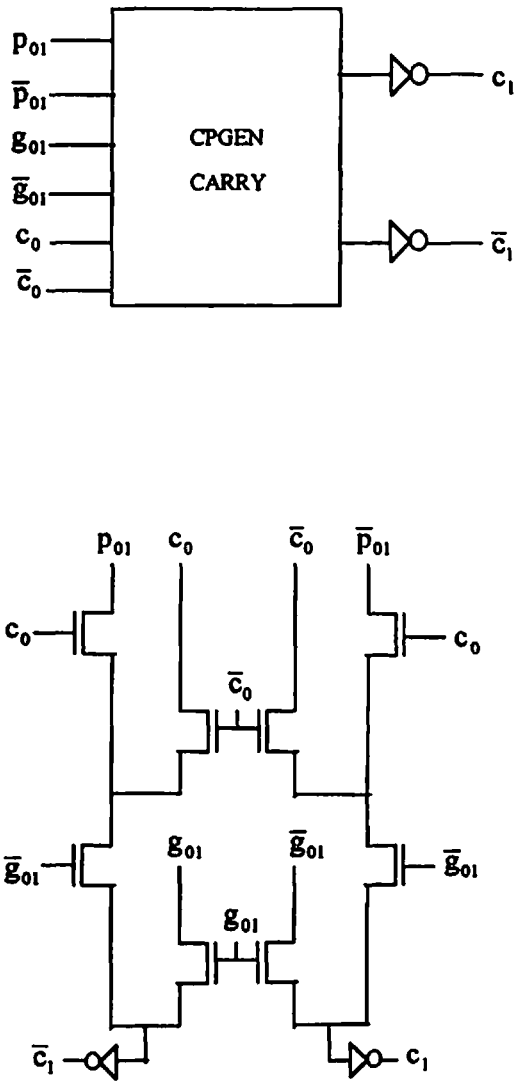


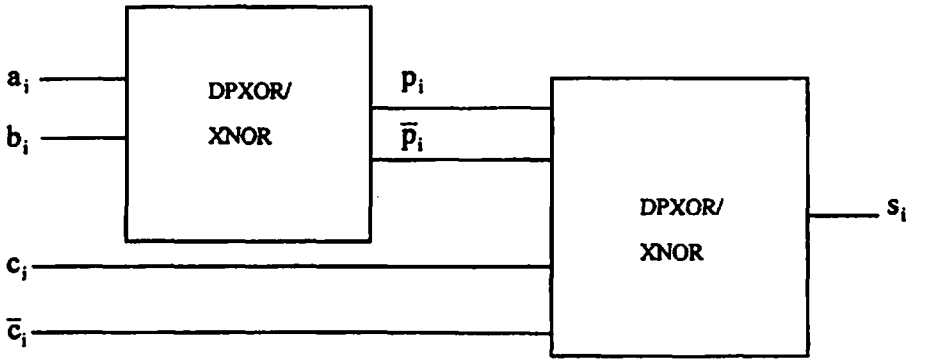
Fig. 5.19. d Carry (and group generate) signal generation circuit using complementary pass transistor logic

Dual pass transistor logic

The standard cells used in the designs are-

DPXOR/XNOR, DPAND/NAND, DPPROP, DPGENCARRY, INV.

The circuits for generating propagate 'pi' and sum 's_i' signals are the same as shown in Fig. 5.4 (a). The block diagrams of A-cell and B-cell in domino CMOS logic are shown in Fig. 5.20 (a), and 5.20 (b). The schematic of standard cell 'DPAND/NAND' used for implementing generate 'g_i' function is the same as shown in Fig.5.4 (b). The circuits for generating group propagate, group generate (and carry) signals are shown in Fig. 5.20 (c), and 5.20 (d).



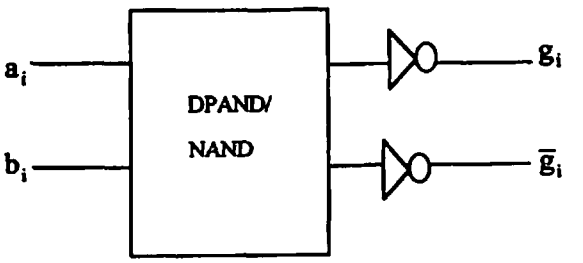


Fig. 5.20. a Block diagram of A-cell

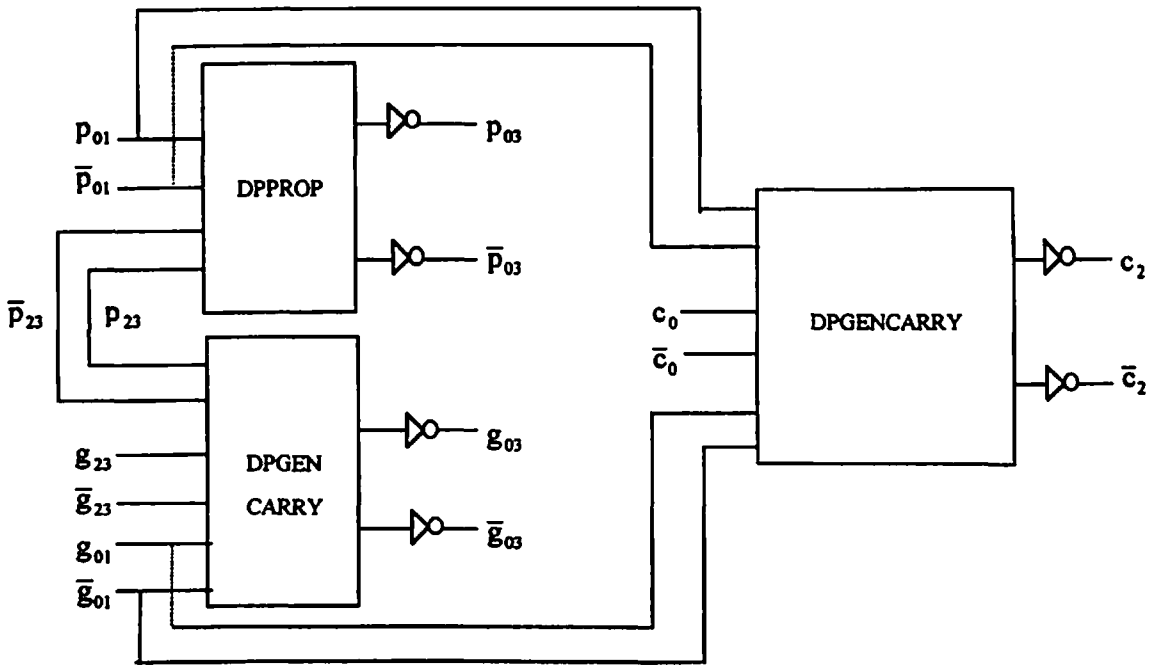


Fig. 5.20. b Block diagram of B-cell

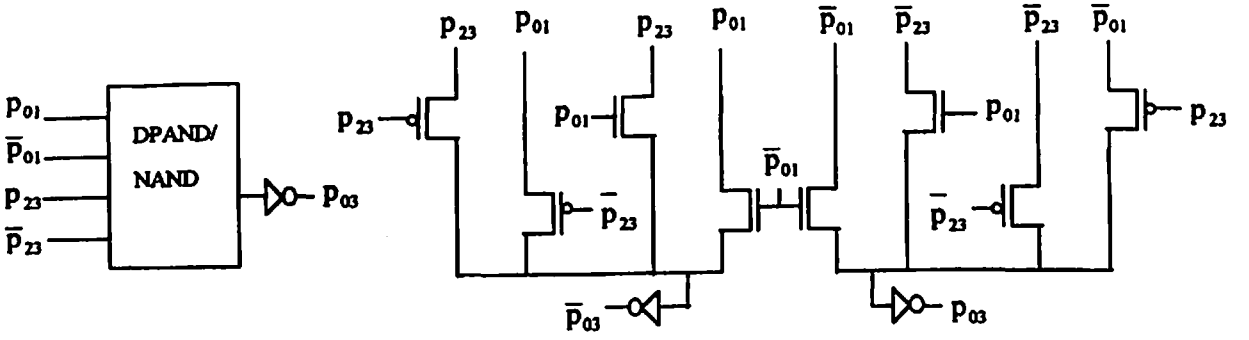
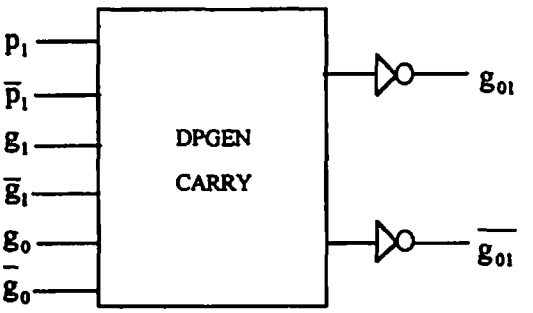


Fig. 5.20. c Group propagate signal generation circuit using dual pass transistor logic



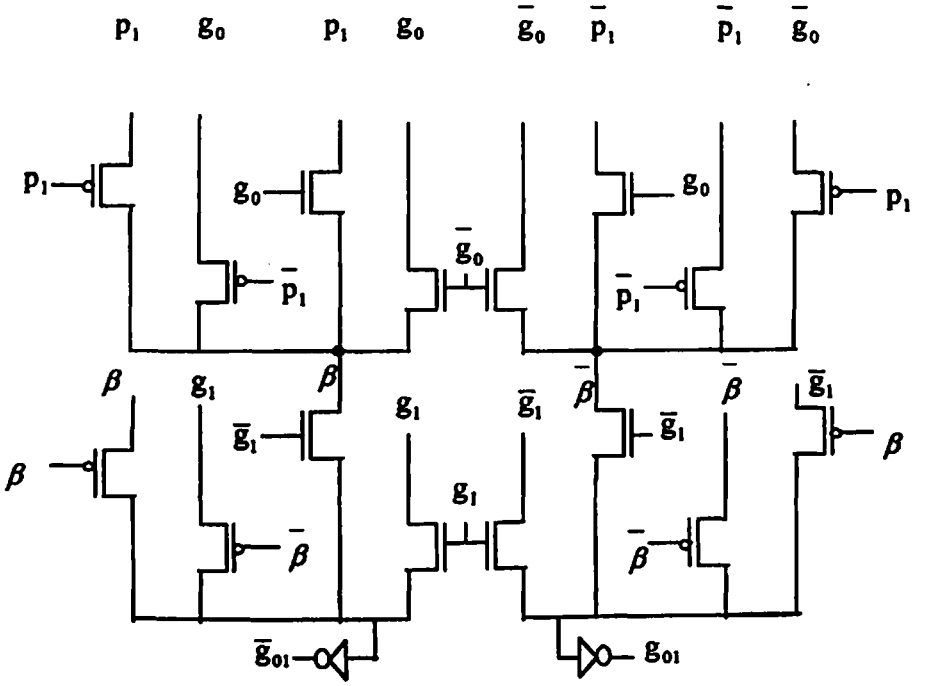


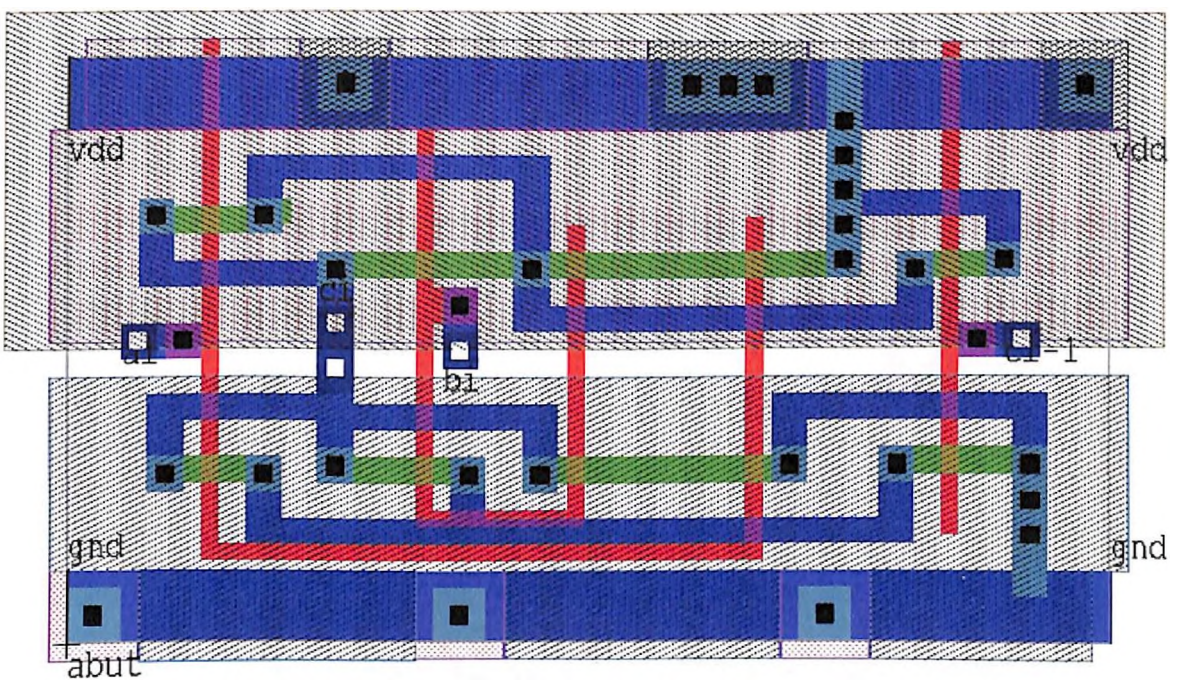
Fig. 5.20. d Group generate signal generation circuit using complementary pass transistor logic

5.6 STANDARD CELL LAYOUTS

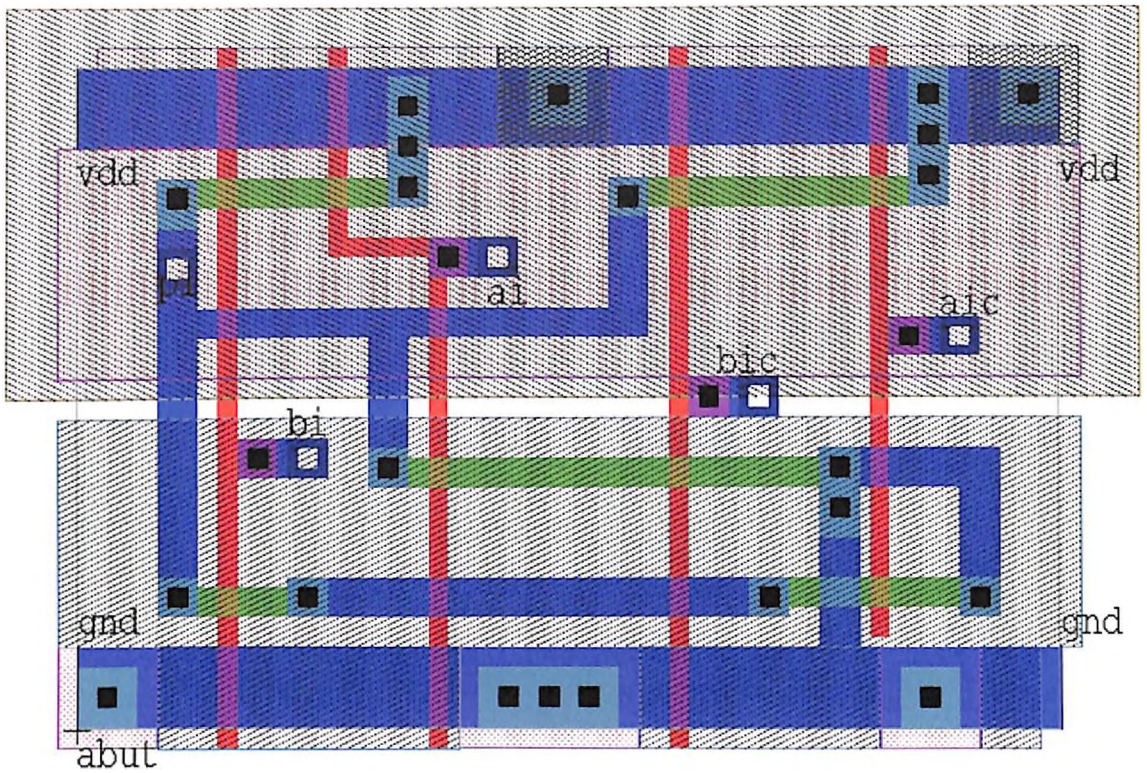
Standard cell layouts of cells of different adders with transistor size w=1.8μm, l=1.2μm and designed in different logic design styles are shown in this section. Layouts of cells with widths w=3.6μm and w=6μm can be obtained by increasing the width of active layer (green in colour).

FULLY STATIC CMOS LOGIC

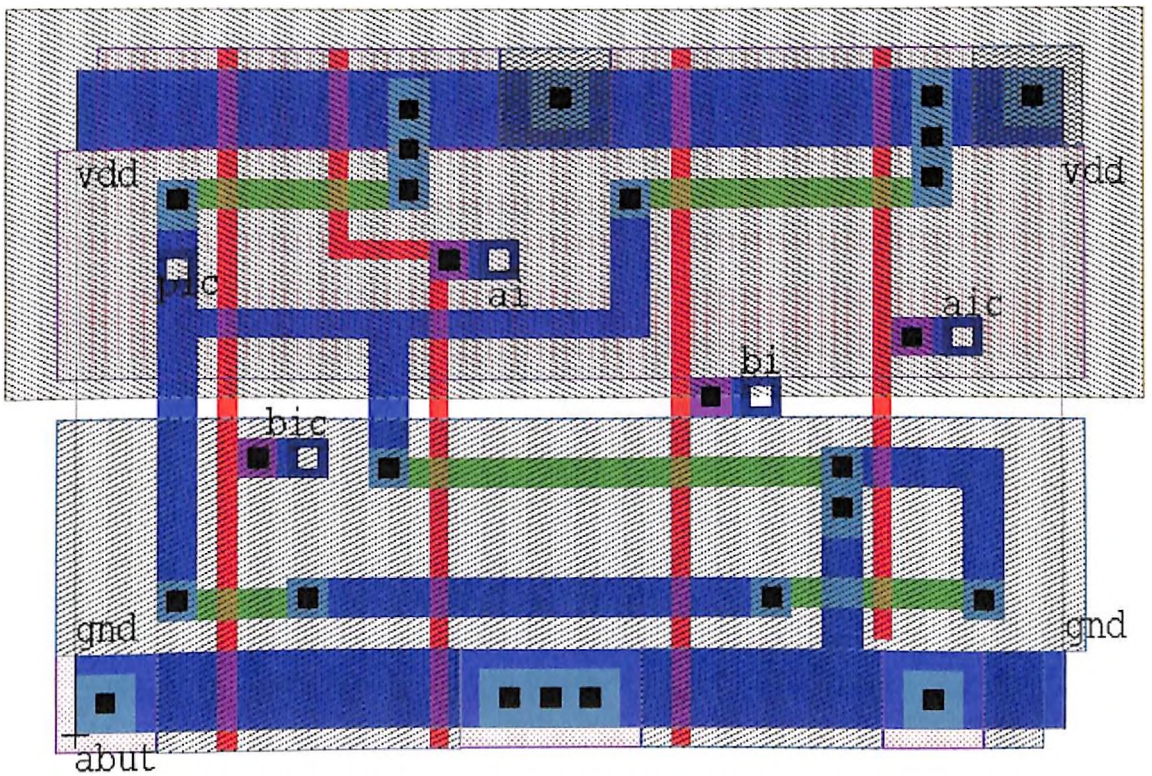
1) Standard Cell FSNAND4



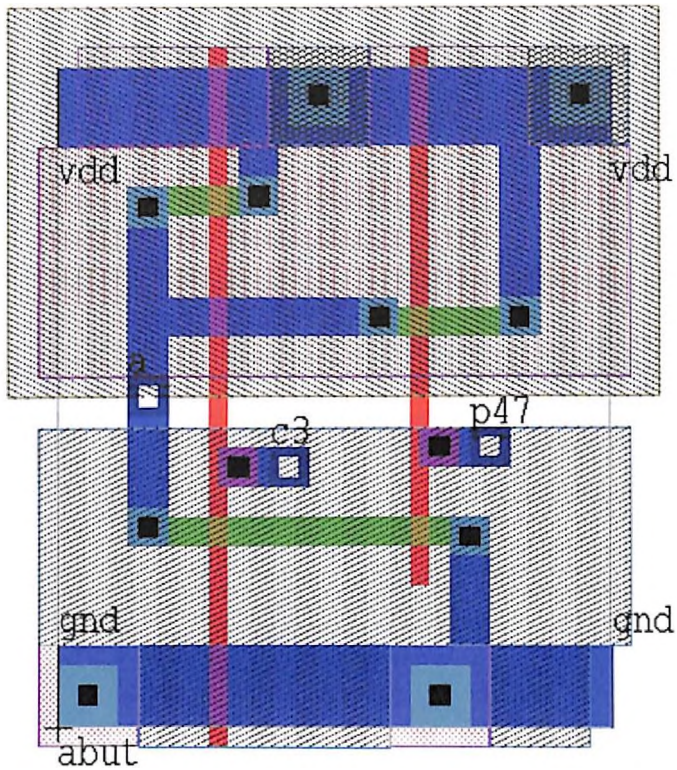
2) Standard Cell FSXOR



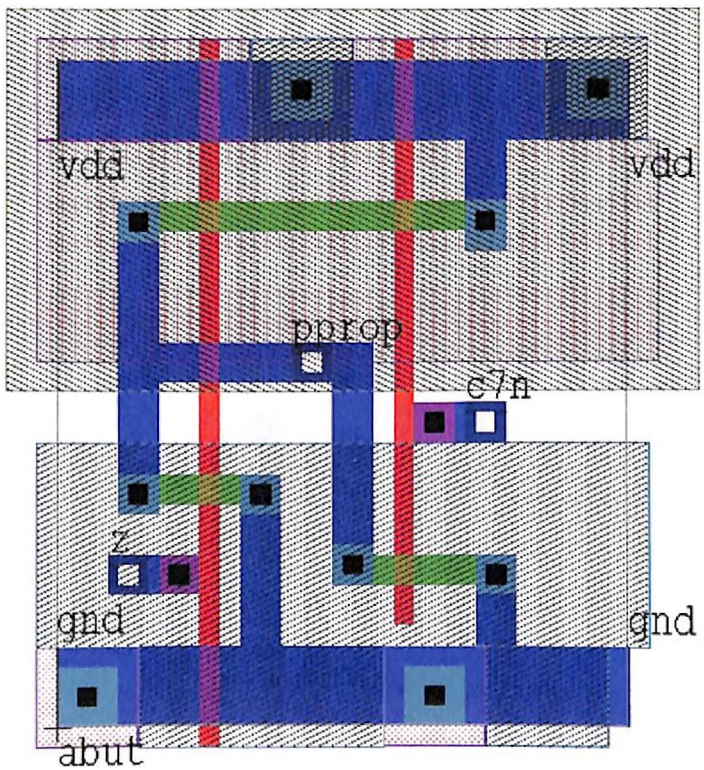
3) Standard Cell FSXNOR



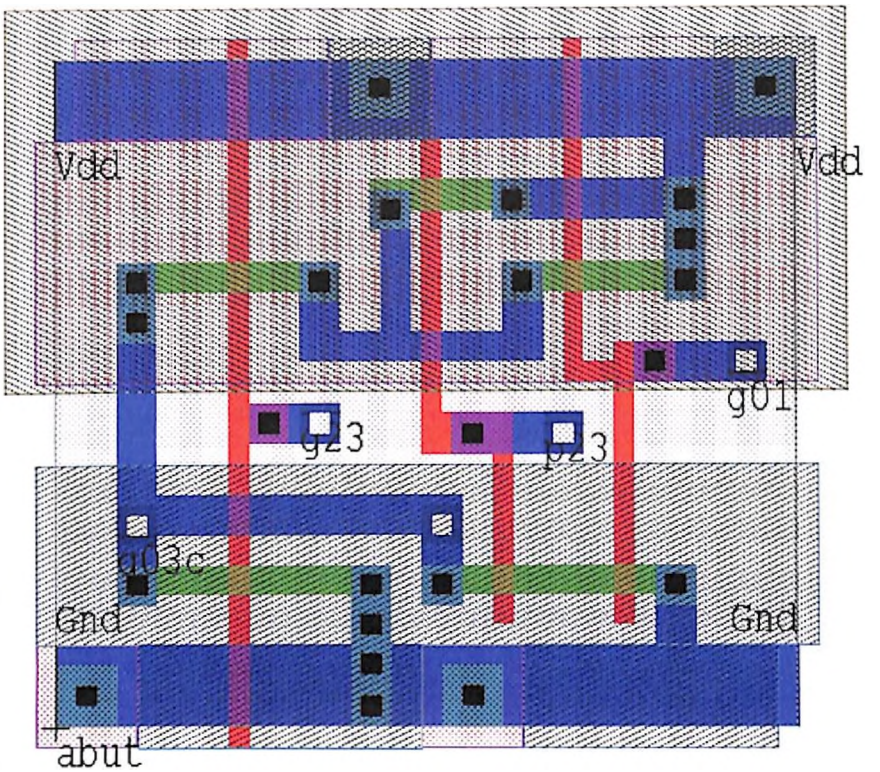
4) Standard Cell FSNAND



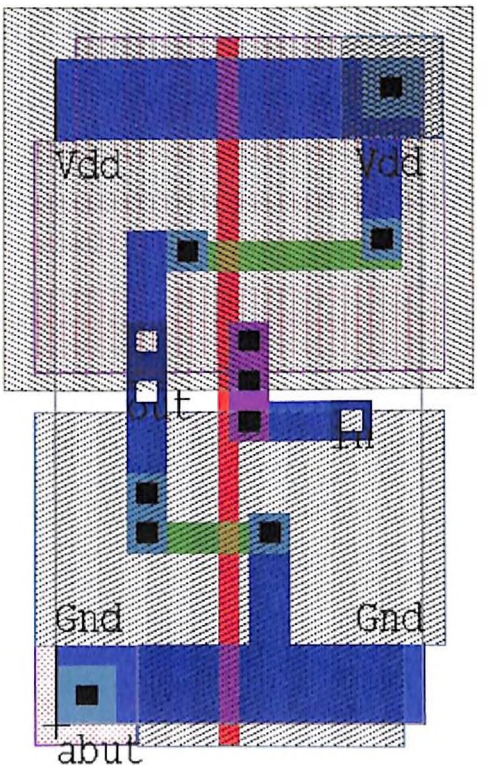
5) Standard Cell FSNOR



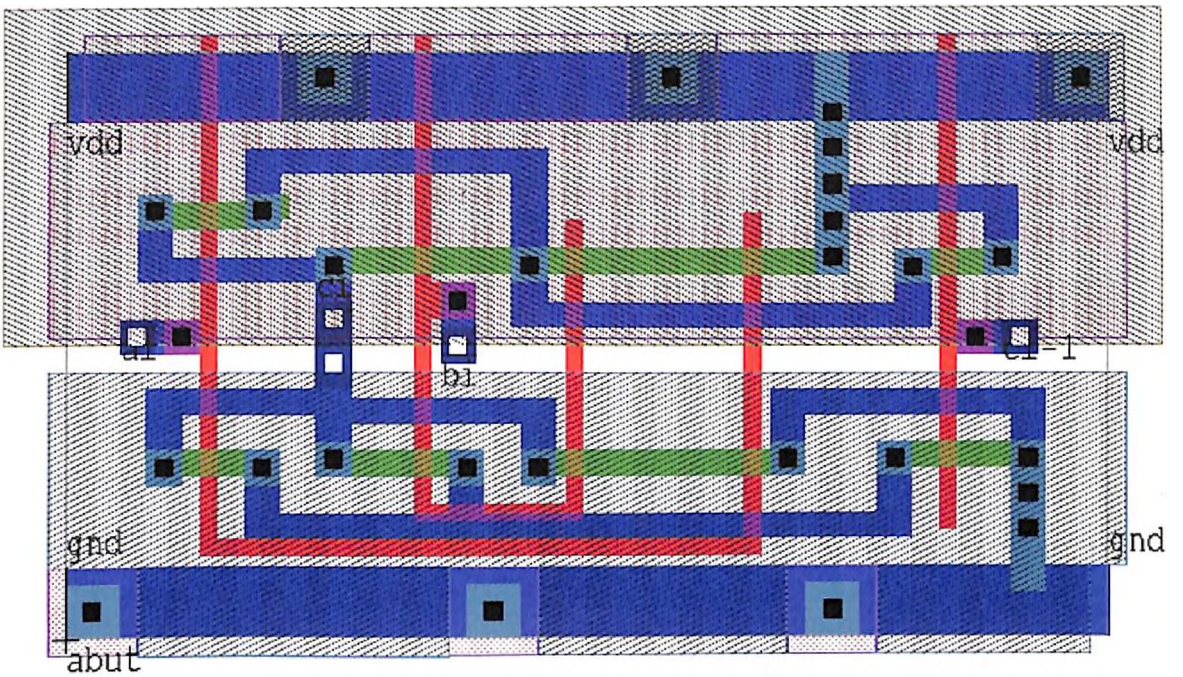
6) Standard Cell FSCARRYGEN



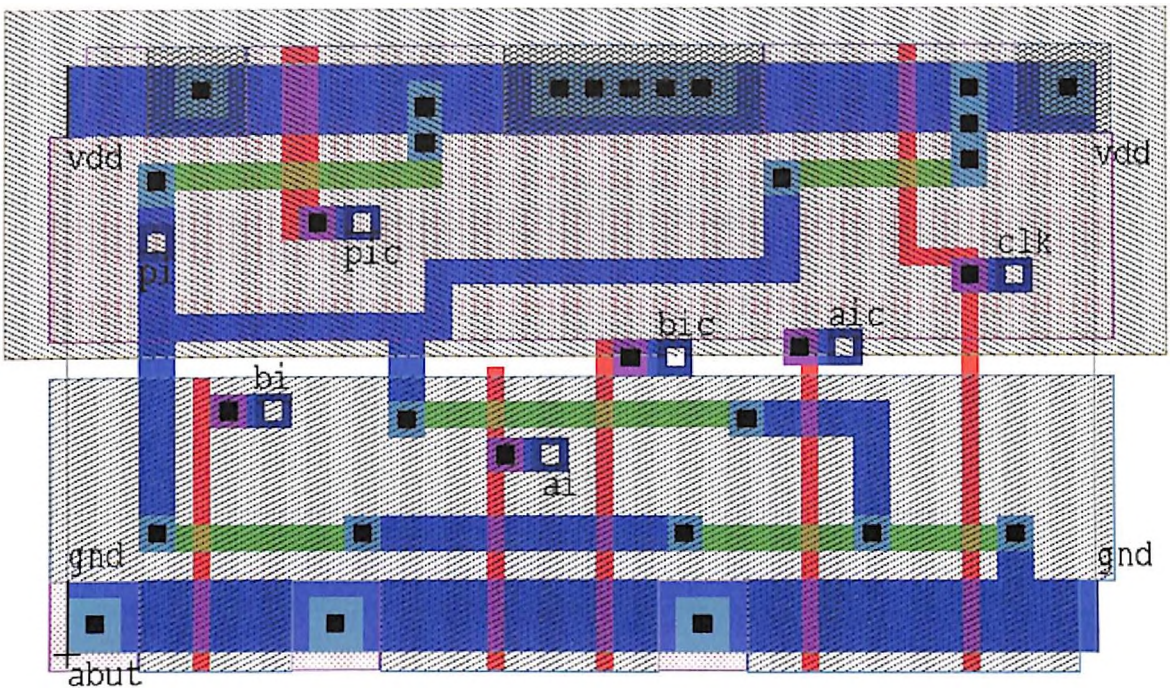
7) Standard Cell INV



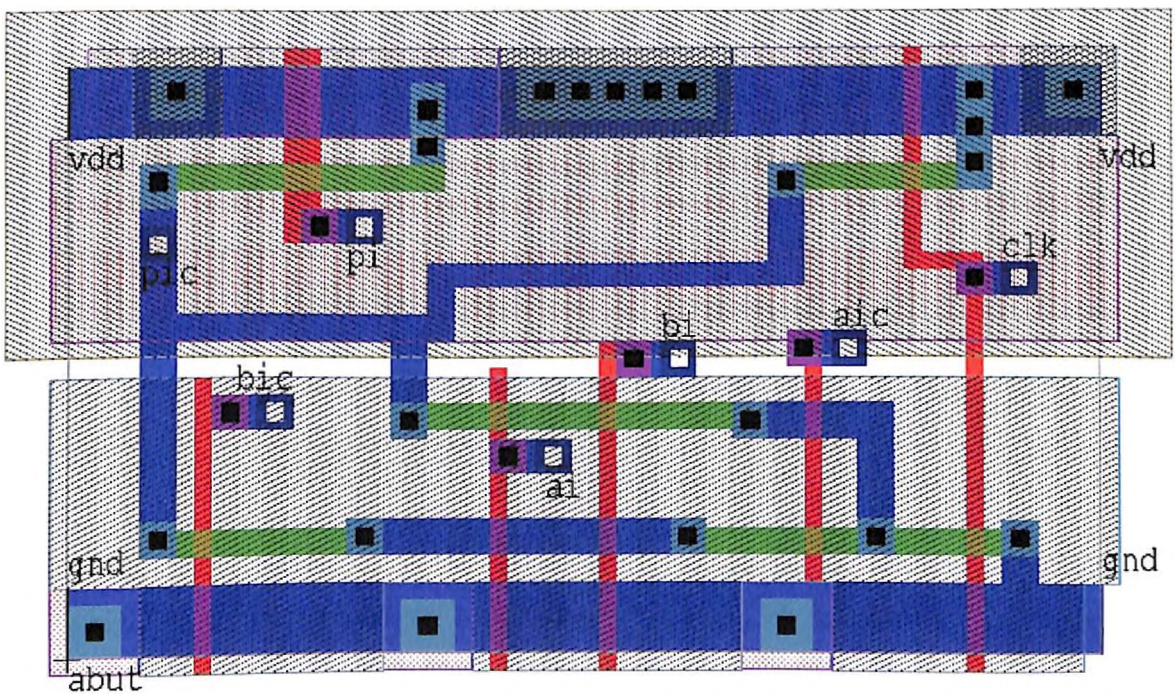
8) Standard Cell FSCARRYOUT



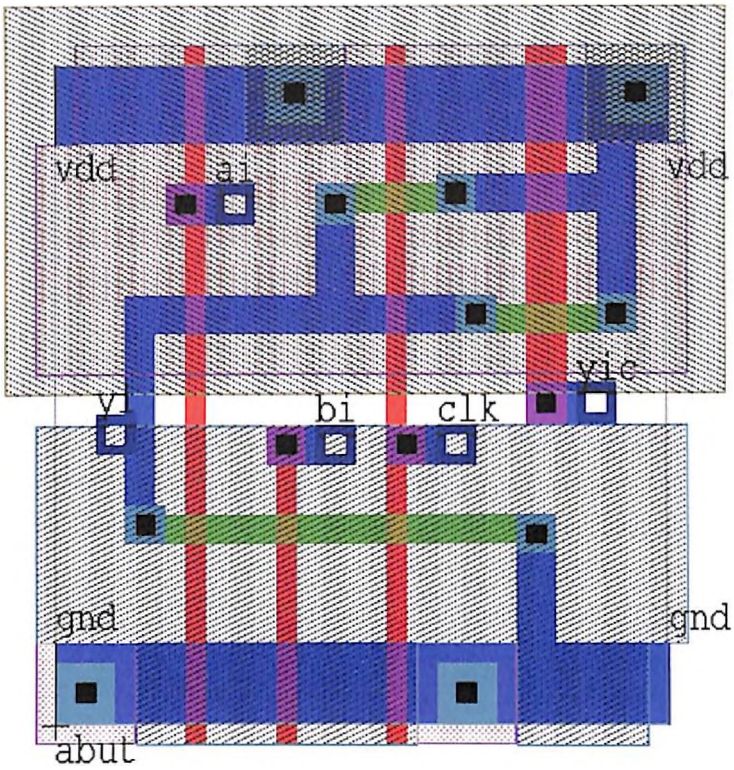
9) Standard Cell DMXOR



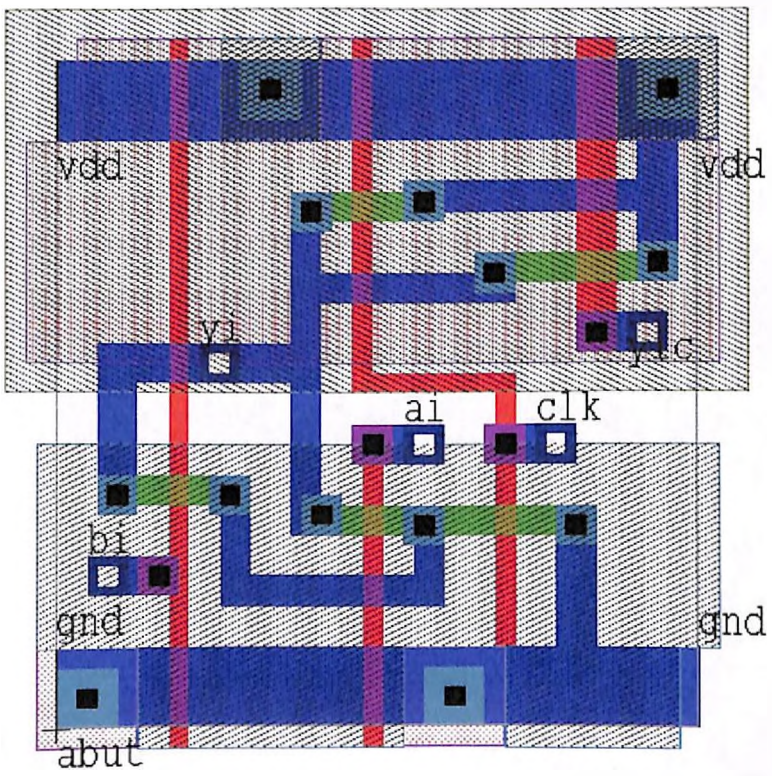
10) Standard Cell DMXNOR



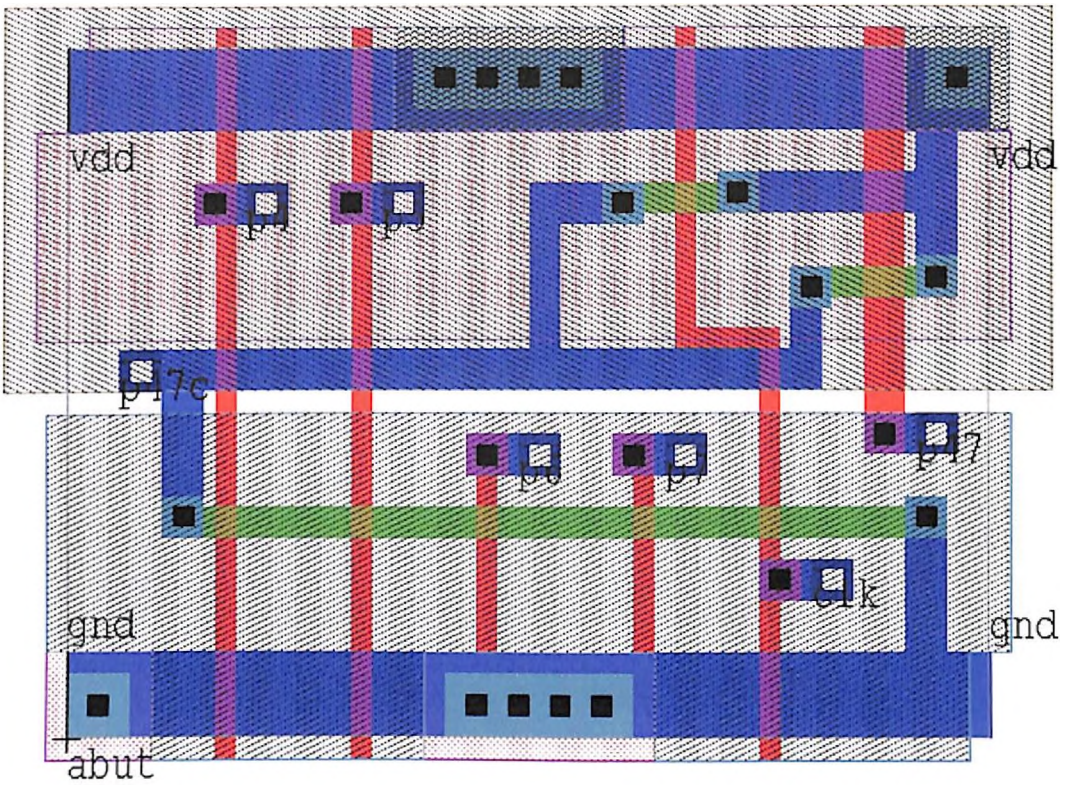
11) Standard Cell DMNAND



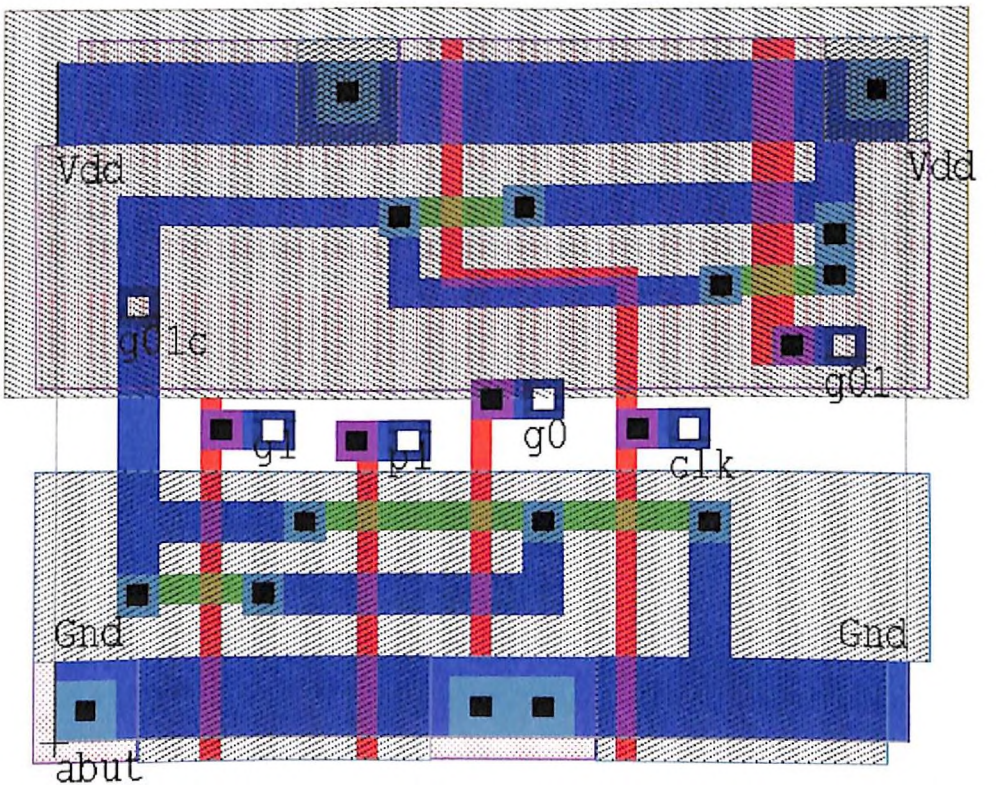
12) Standard Cell DMNOR



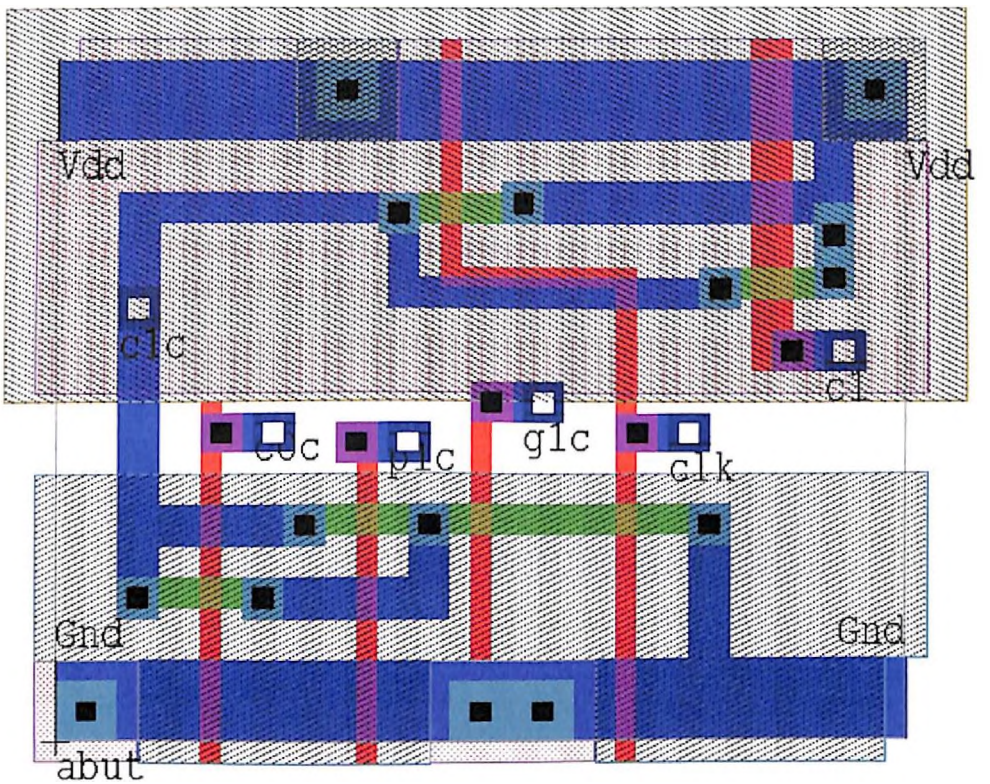
13) Standard Cell DMNAND4



14) Standard Cell DMGENCARRY

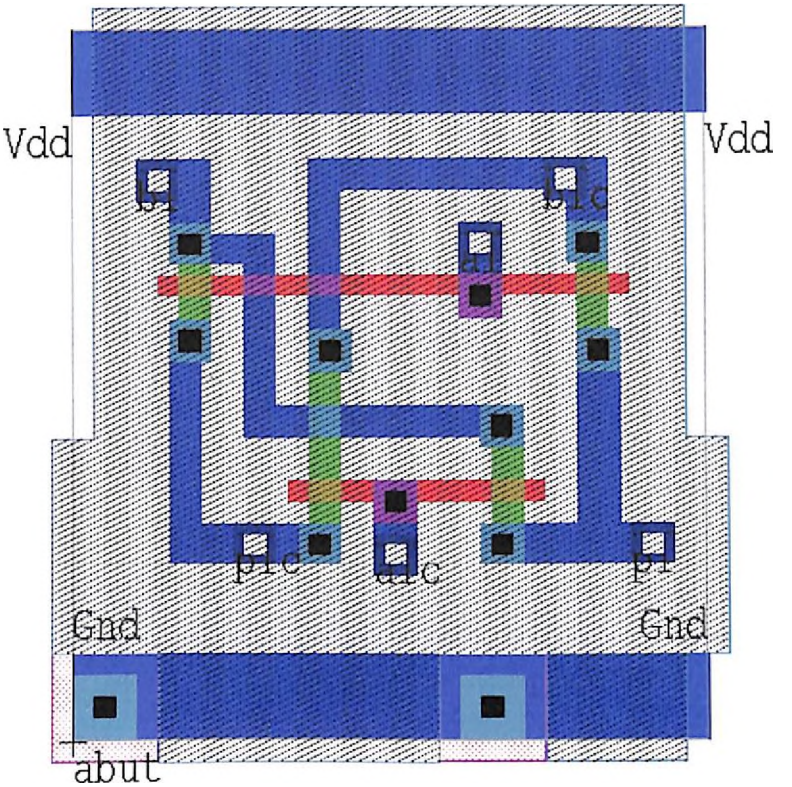


15) Standard Cell DMCARRYCOMP

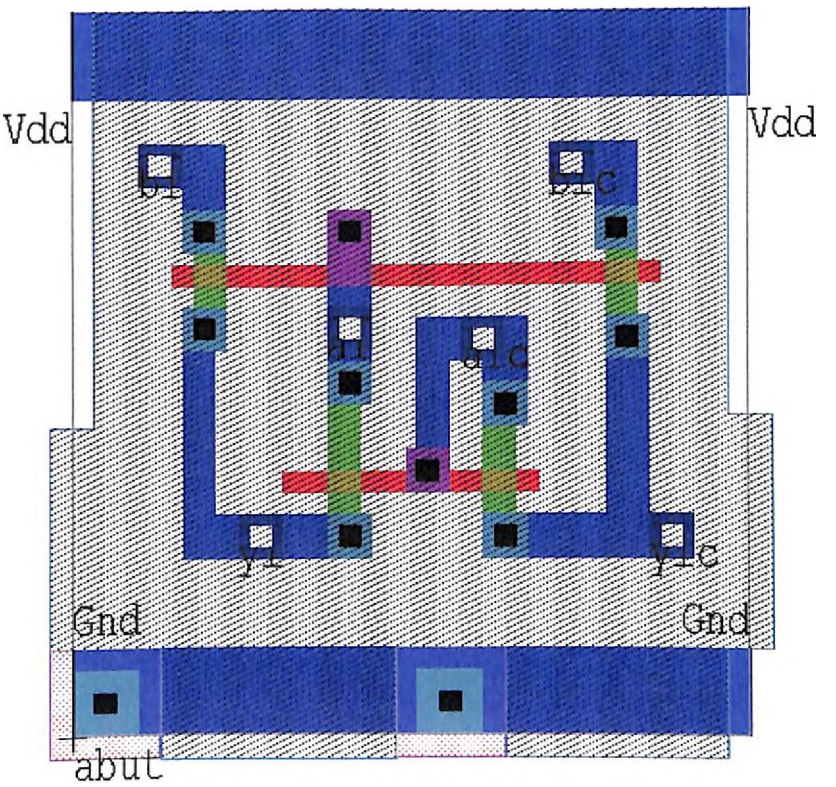


COMPLEMENTARY PASS TRANSISTOR LOGIC

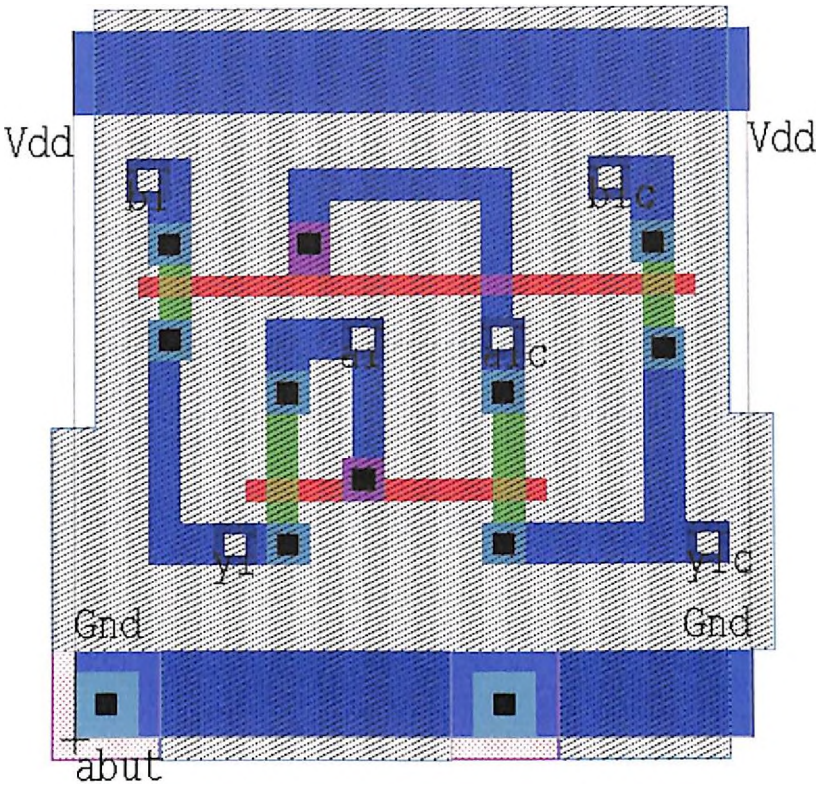
16) Standard Cell CPXOR/XNOR



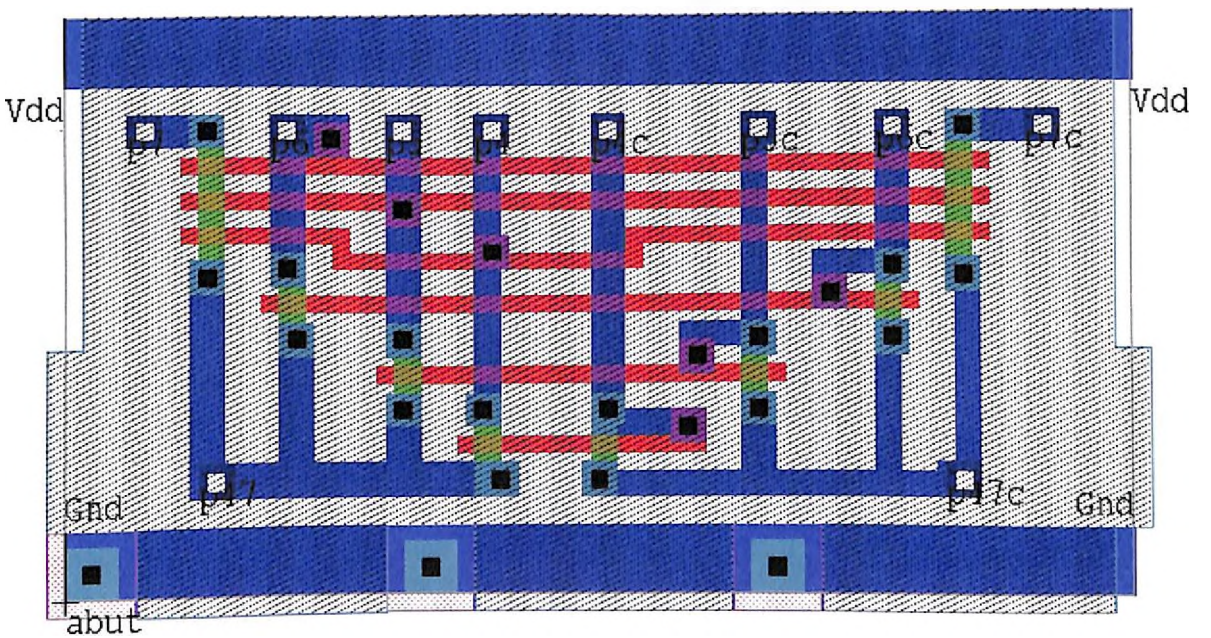
17) Standard Cell CPAND/NAND



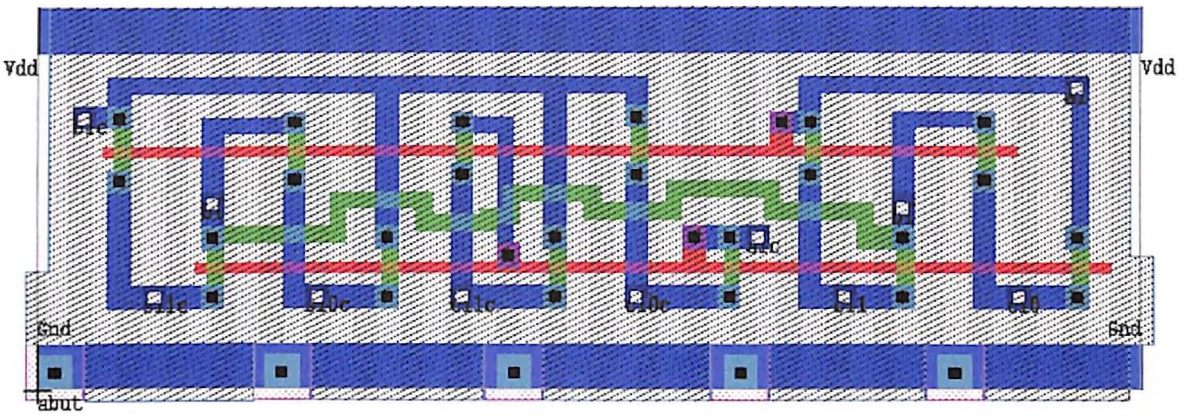
18) Standard Cell CPOR/NOR



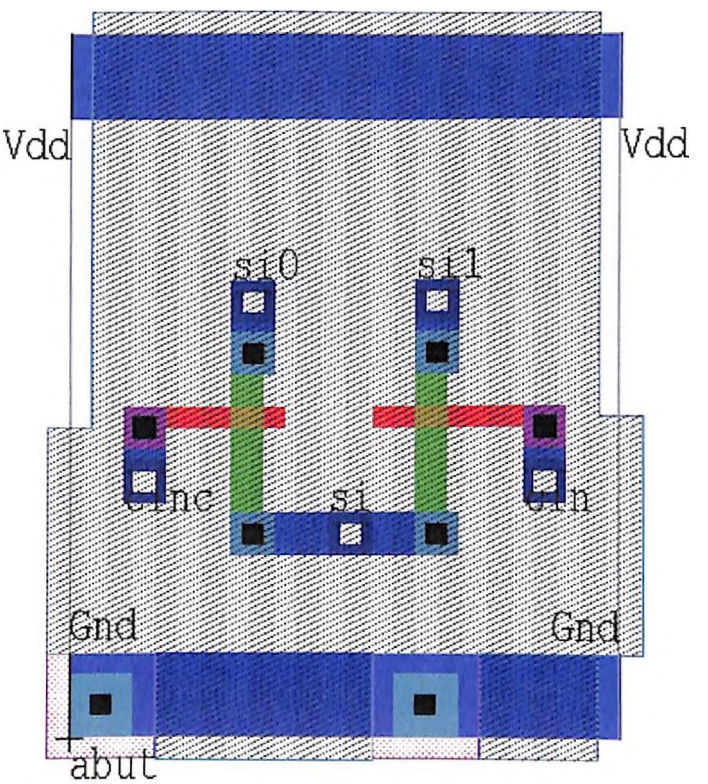
19) Standard Cell CPAND4/NAND4



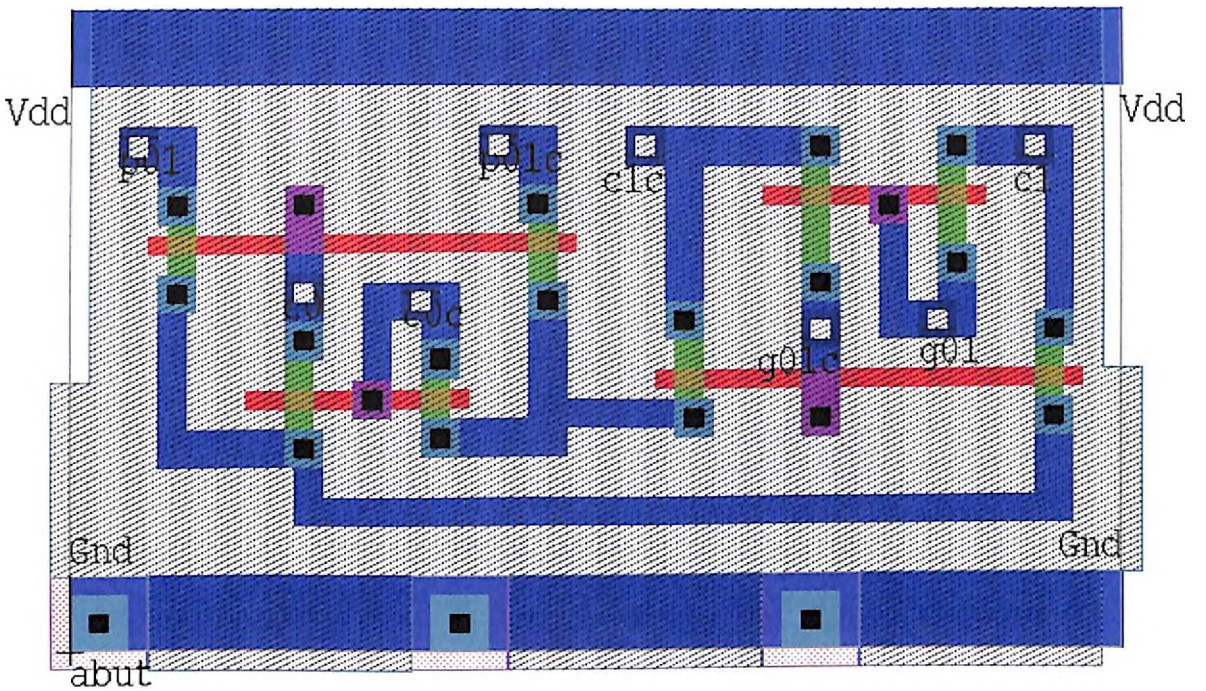
20) Standard Cell CPCONDITIONAL-CELL



21) Standard Cell CPMUX

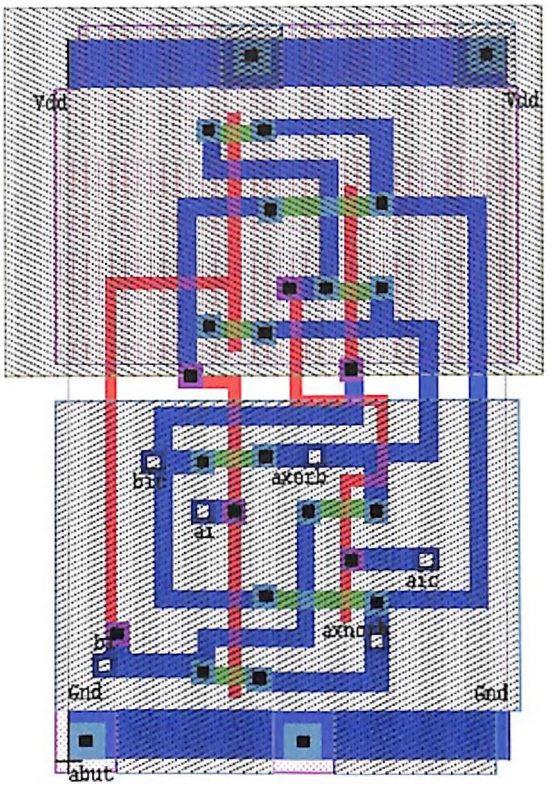


22) Standard Cell CPGENCARRY

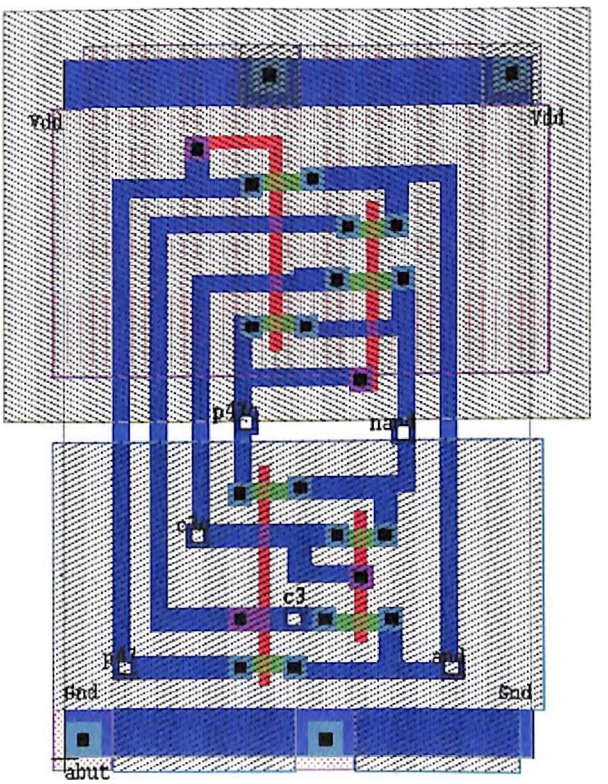


DUAL PASS TRANSISTOR LOGIC

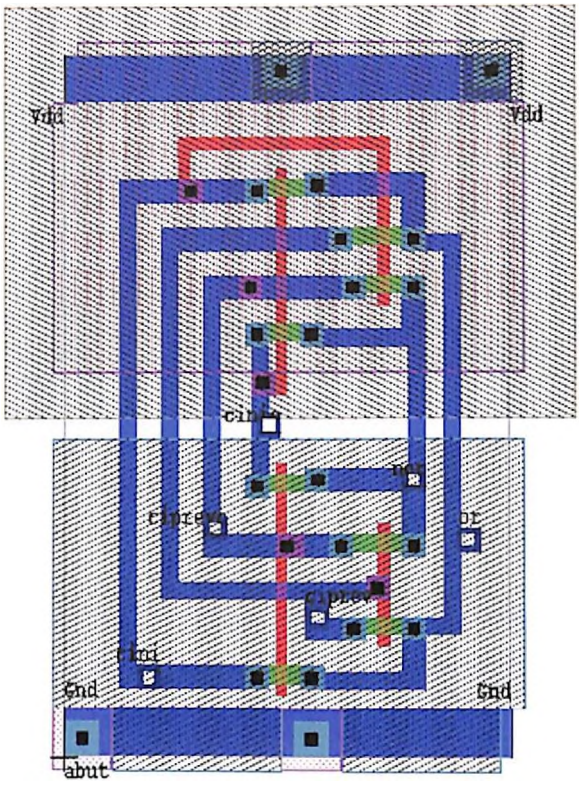
23) Standard Cell DPXOR/XNOR



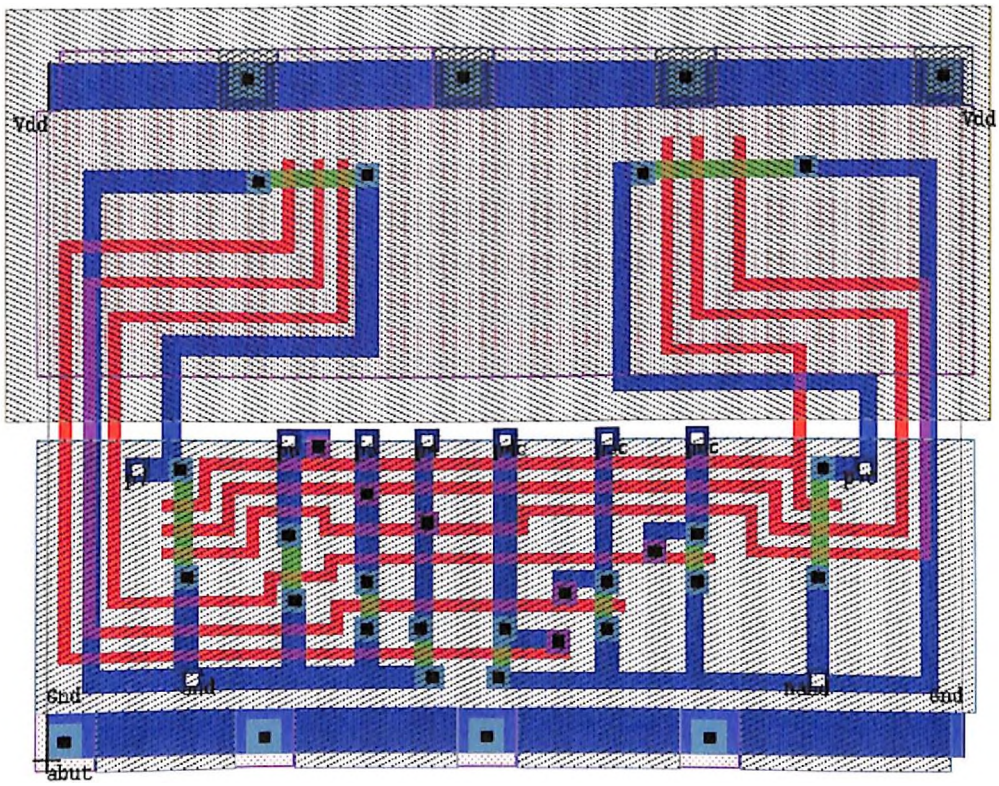
24) Standard Cell DPAND/NAND



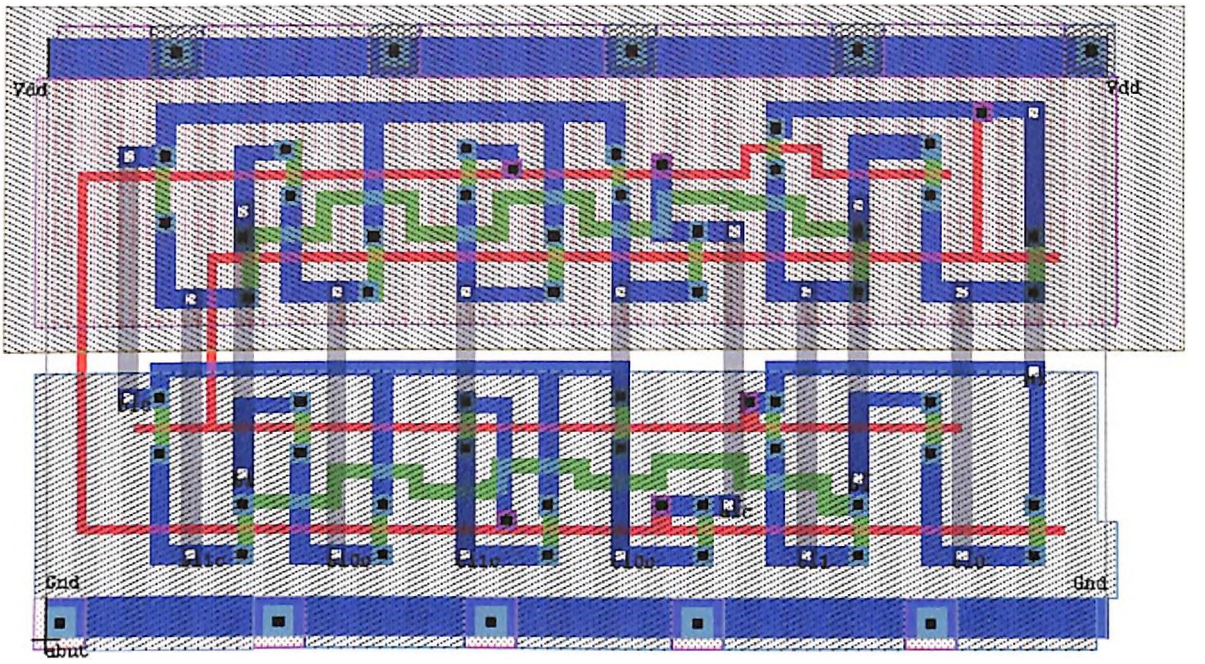
25) Standard Cell DPOR/NOR



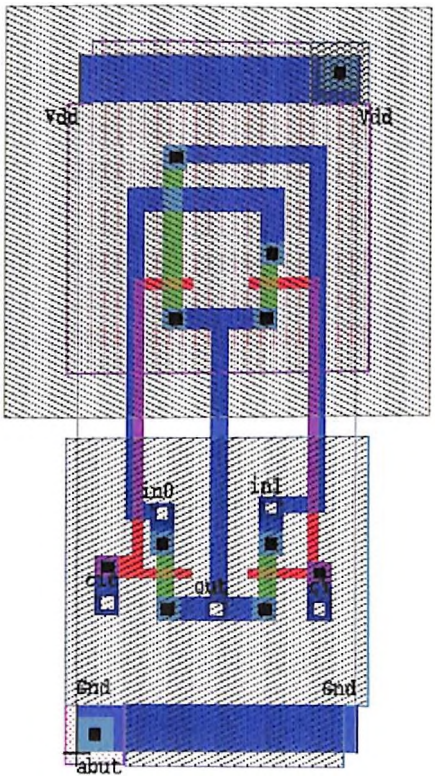
26) Standard Cell DPAND4/NAND4



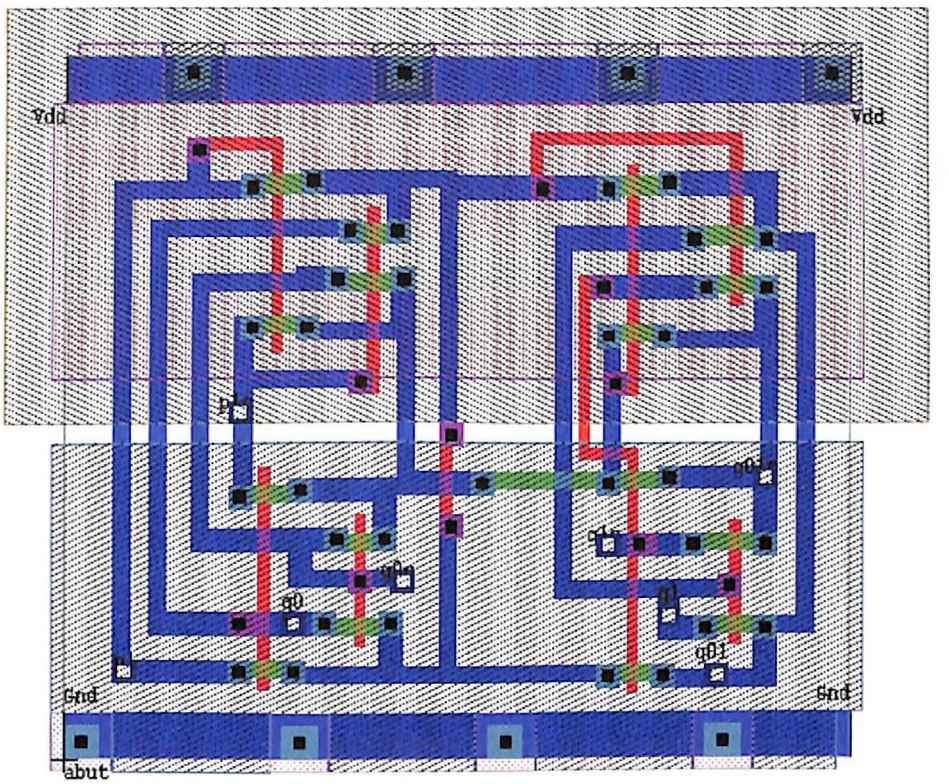
27) Standard Cell DPCONDITIONAL-CELL



28) Standard Cell DPMUX



29) Standard Cell DPGENCARRY



CHAPTER 6

SIMULATION RESULTS OF DIFFERENT ADDERS

Different architectures used for designing adders have been discussed in chapter 3. Each adder architecture has been designed using four different logic design styles that have been discussed in chapter 4. Each design has been implemented with three-transistor sizes - *(w/l)* = 1.5, 3, and 5 to take into account the effect of increased *(w/l)* on the adder performance. The minimum size (*w/l*)=1.5 corresponds to an area efficient implementation, the moderate size (*w/l*)=3 corresponds to a trade off between area and speed, and, still larger size (*w/l*)=5 corresponds to the highest speed realizable - being least encumbered by the circuit parasitics. The standard cells used in the designs are described in chapter 5. The circuit extraction tool 'EXTRACT' along with the appropriate technology file containing necessary information for extracting circuit parasitics was used to generate circuit netlists for simulation. Tanner's SPICE tool 'T-SPICE Pro' has been used for simulation to obtain the power consumption and propagation delay measures of the adder designs. The transistor parameters used are from the 1.2μm MOSIS fabrication run and are listed in ***Appendix B1.***

Transient simulations have been carried out using SPICE. Nominal values of device parameters, temperature of 27 degree Celsius, and level 2 model of MOSFET [Antognetti 1988] have been used. Transient simulations have been done with carry-in to the adder set to zero. Input pulses applied have rise and fall times of 1 nanosecond, pulse width of 150 nanoseconds, and pulse repetition of 200 nano-seconds. Propagation-delay has been measured from the input signal to the most delayed bit of the sum or carry output generated by the adder in the worst-case. Worst-case propagation delay has been measured directly from the waveforms as the time delay between the 50% transition point of the input voltage and the 50% transition of the most delayed output voltage.

Simulation tool 'T-SPICE' can measure the power consumed by a circuit for a given set of inputs. The instantaneous power P(τ) drawn from a voltage source at time τ is given by the product of current through the source and the voltage drop across the source. T-

SPICE computes the average power $\bar{P}$ over a time interval (t_1, t_2) by using the trapezoidal rule to evaluate the integral

$$\bar{P}(t_1, t_2) = \frac{1}{t_1 - t_2} \int_{t1}^{t2} P(\tau)\, d\tau \qquad \ldots 6.1$$

Average power dissipated by the input sources is also added to obtain total average power dissipation. We have calculated the energy consumption per addition for the adders. Energy-consumption is calculated by taking the product of average power dissipation and length of simulation time. The numerical accuracy and discretization error in T-spice have been optimized through a number of user-adjustable parameters. The values set for the absolute error, absolute charge, relative charge, and relative error tolerances on the norm of change in voltage for simulations were 5 nanoamperes, 0.01 picocoulombs, 0.001, and 0.0001 respectively.

Simulation results for worst-case propagation delay and energy consumption per addition for different adder designs with circuit parasitics are listed in tables T1 to T20 in ***Appendix B3***. Results listed in ***Appendix B2*** give the propagation delay values of different adder designs without including circuit parasitics for fully static CMOS logic design style. ***Appendix B6*** lists the transistor count of different adders and the core area of different 64-bit adder designs. The simulation results showing the variation of worst-case propagation delay with operand size and the variation of energy consumption with operand size (Figs. 6.1 to 6.5) are given in this chapter. A second order polynomial is fitted to the simulation results. The results show that worst-case propagation delay of adder designs increases with the increase in operand size for a constant transistor size. Also, it decreases with increase in transistor size for a given operand size. Energy consumption per addition increases with the increase in operand size and the transistor size.

6.1 RESULTS OF RIPPLE CARRY ADDER

Ripple carry adder designs for five operand sizes - 4, 8, 16, 32, and 64-bit are simulated. Results are obtained for four logic design styles and three transistor sizes and are listed graphically in Figs. 6.1

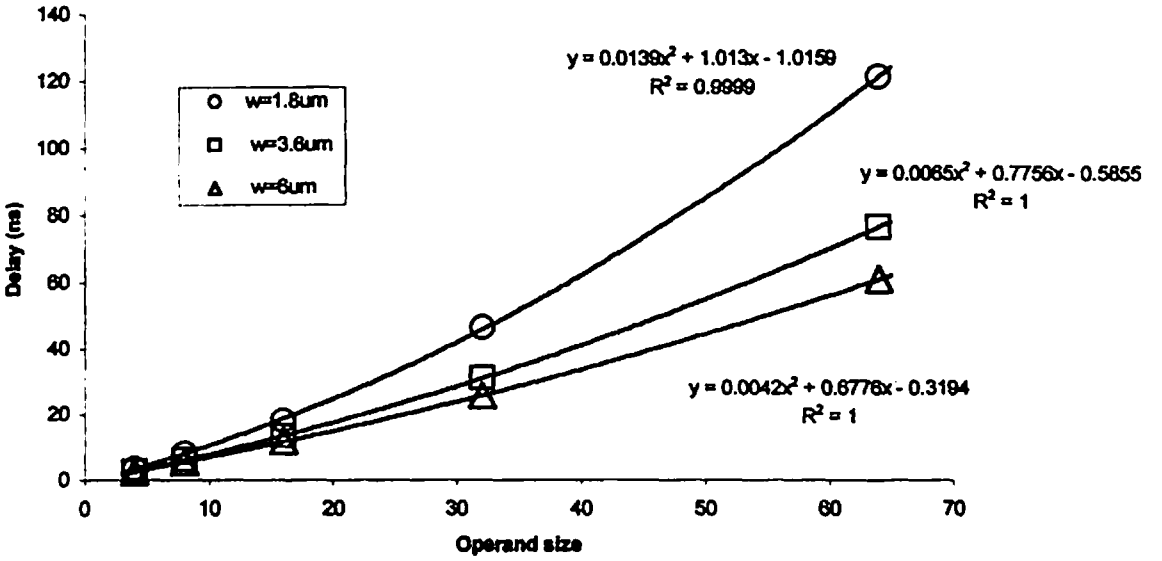


Fig. 6.1.a Propagation delay versus operand size for Ripple carry adder Fully static CMOS logic

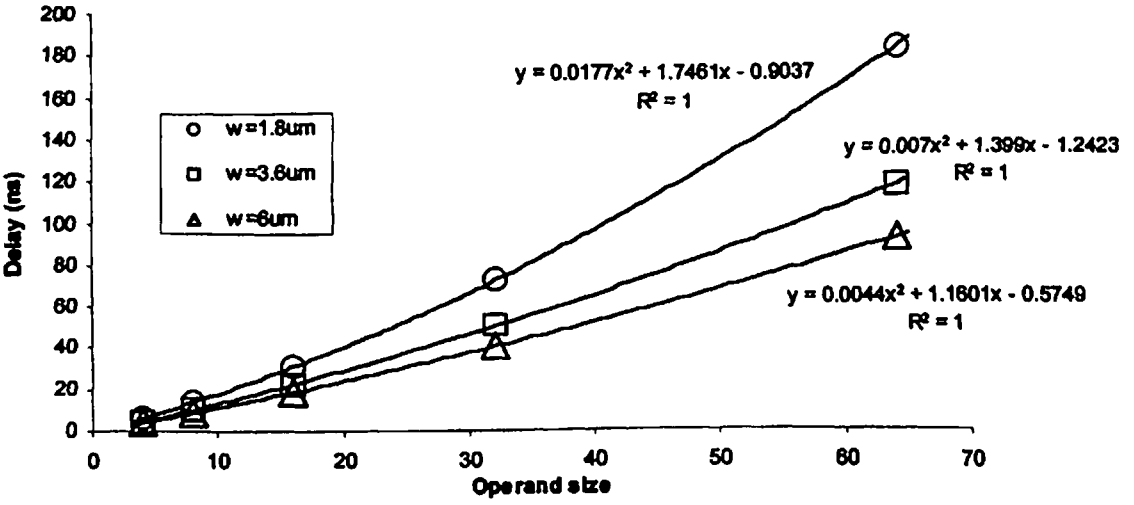


Fig. 6.1.b Propagation delay versus operand size for Ripple carry adder Domino CMOS logic

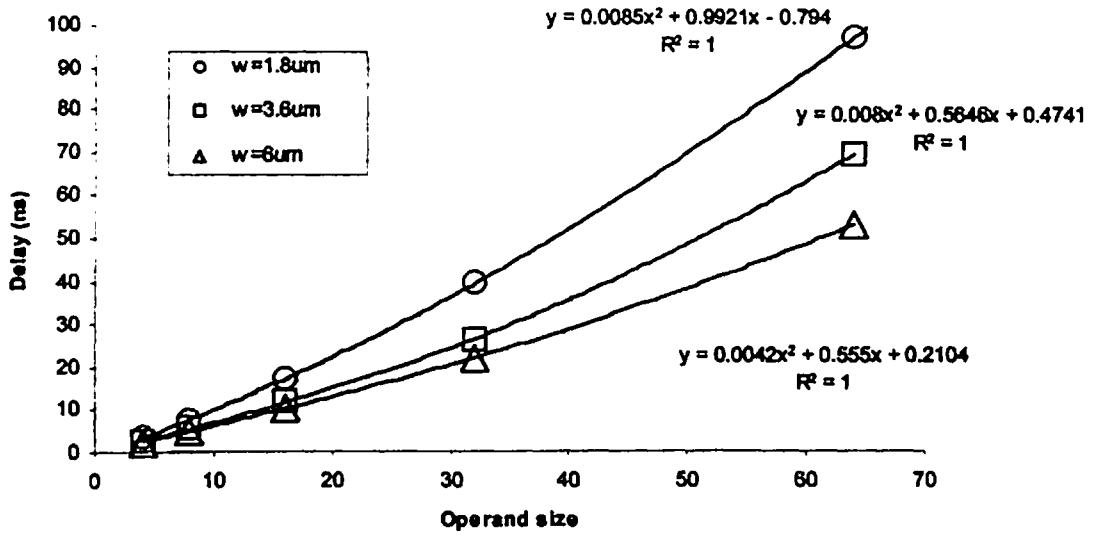


Fig. 6.1.c Propagation delay versus operand size for Ripple carry adder Complementary pass transistor logic

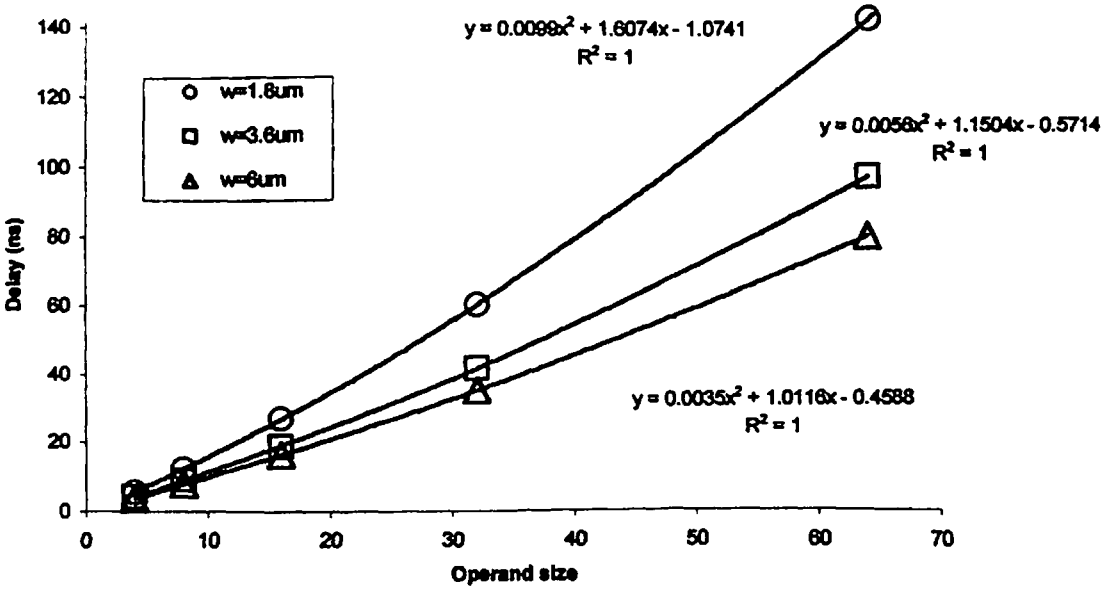


Fig. 6.1.d Propagation delay versus operand size for Ripple carry adder Dual pass transistor logic

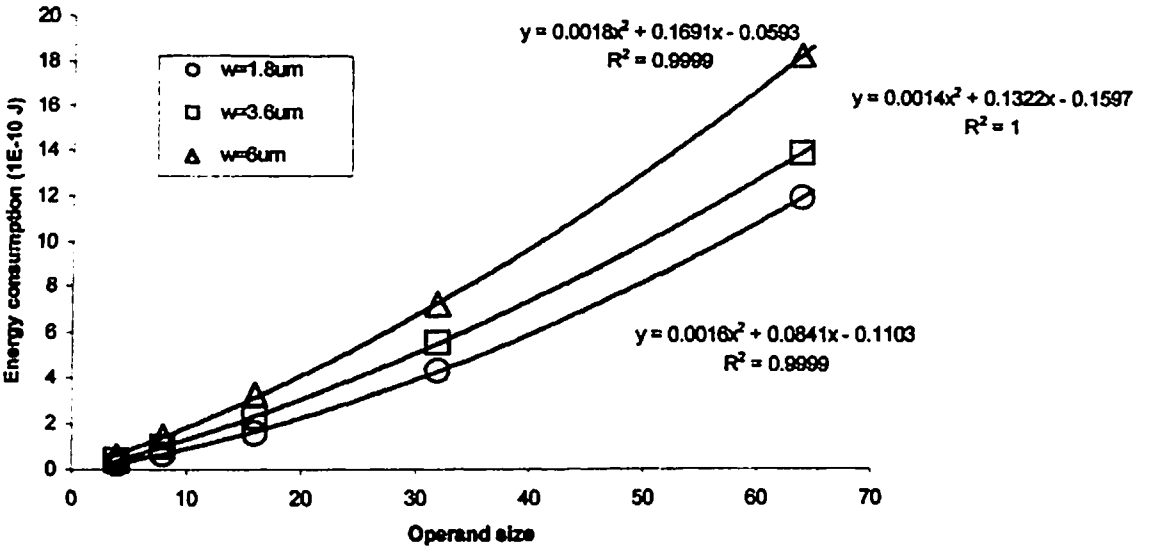


Fig.6.1. e Energy consumption versus operand size for Ripple carry adder Fully static CMOS logic

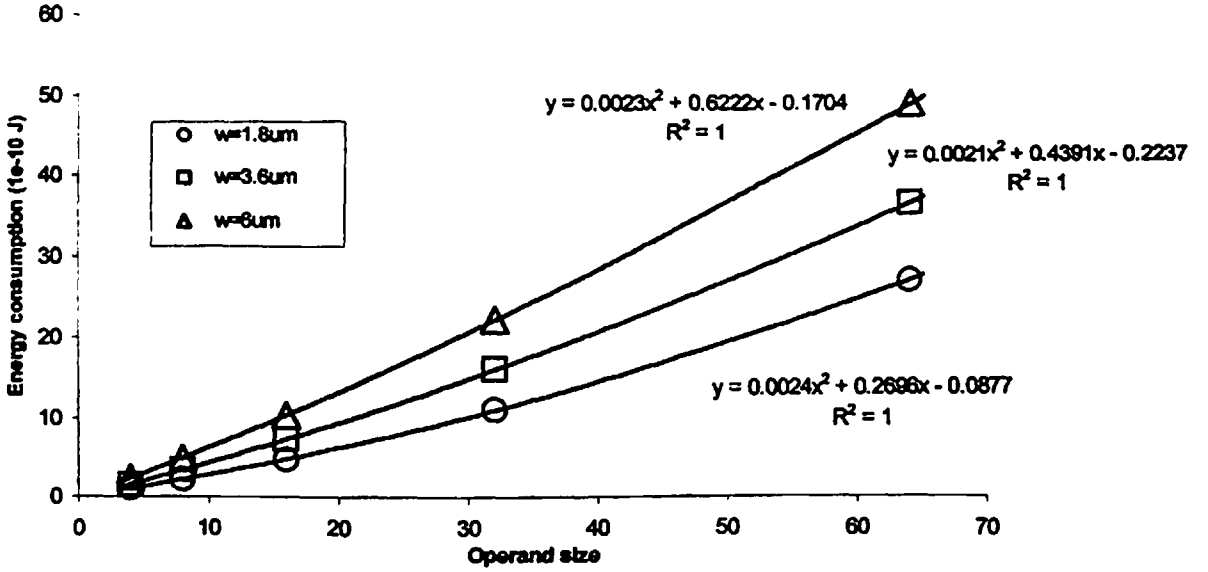


Fig. 6.1.f Energy consumption versus operand size for Ripple carry adder Domino CMOS logic

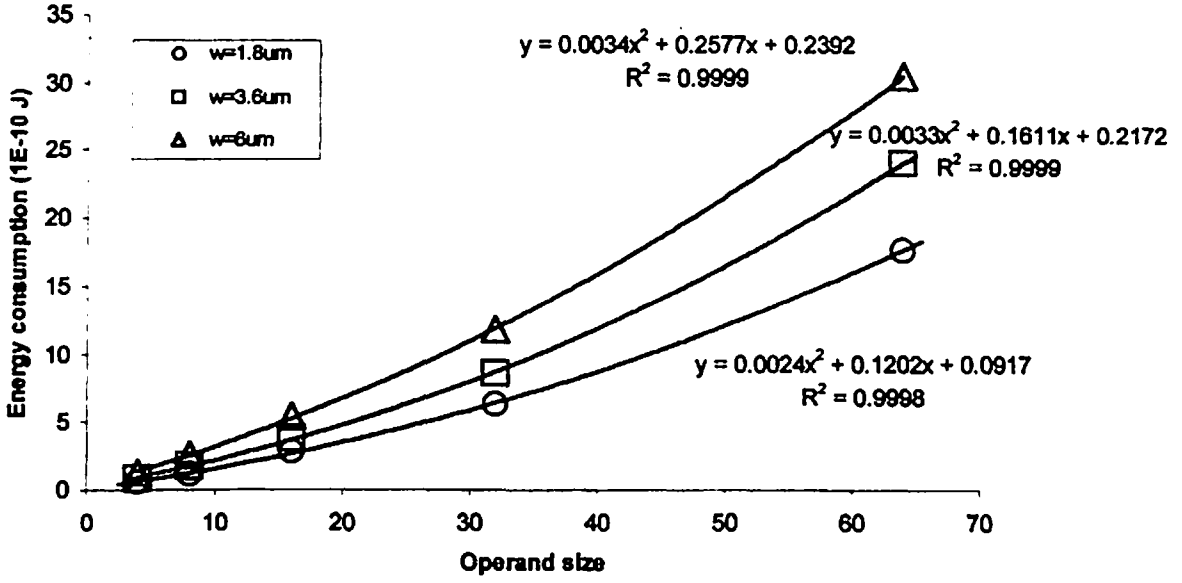


Fig. 6.1. g Energy consumption versus operand size for Ripple carry adder Complementary pass transistor logic

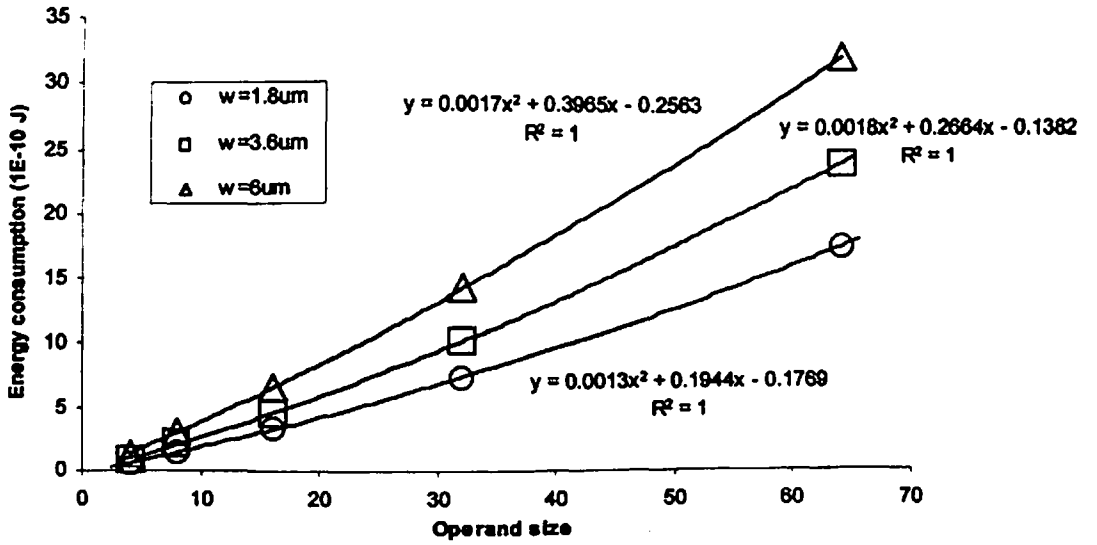


Fig. 6.1.h Energy consumption versus operand size for Ripple carry adder Dual pass transistor logic

6.2 RESULTS OF CARRY SKIP ADDER

The simulation results of carry skip adder designs are shown graphically in Figs. 6.2.

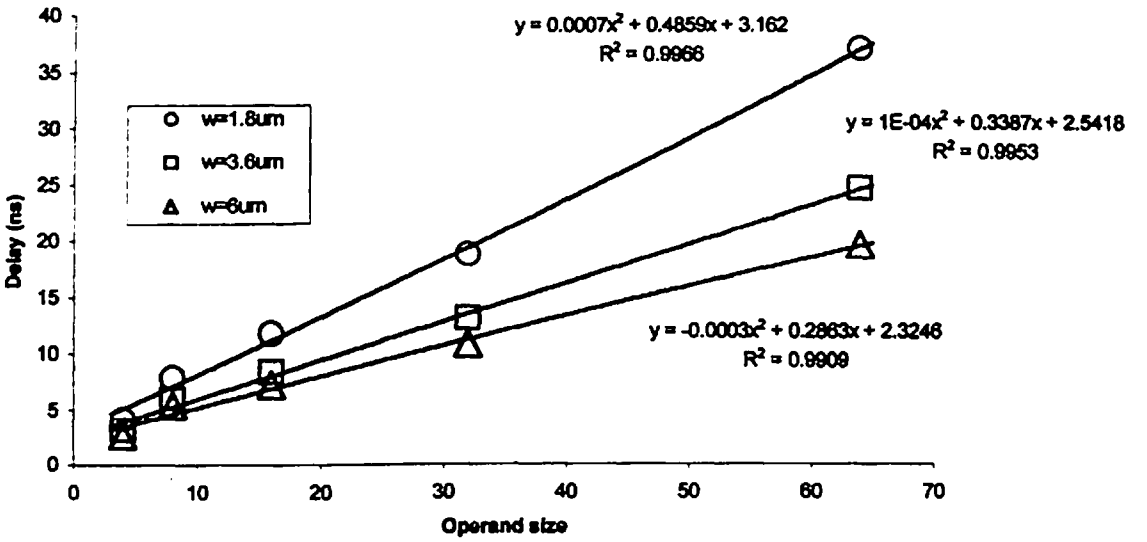


Fig. 6.2. a Propagation delay versus operand size of Carry skip adder
Fully static CMOS logic

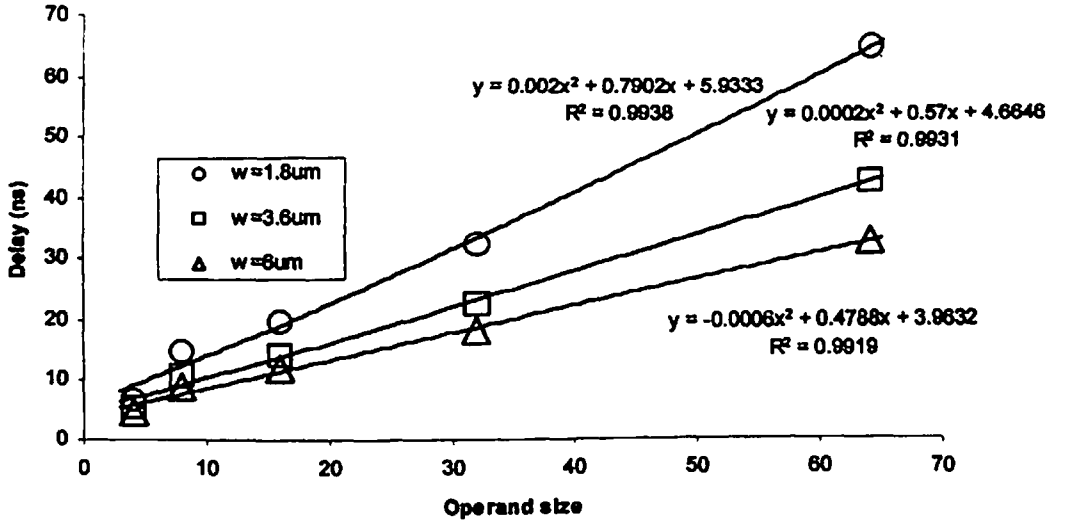


Fig. 6.2. b Propagation delay versus operand size of Carry skip adder
Domino CMOS logic

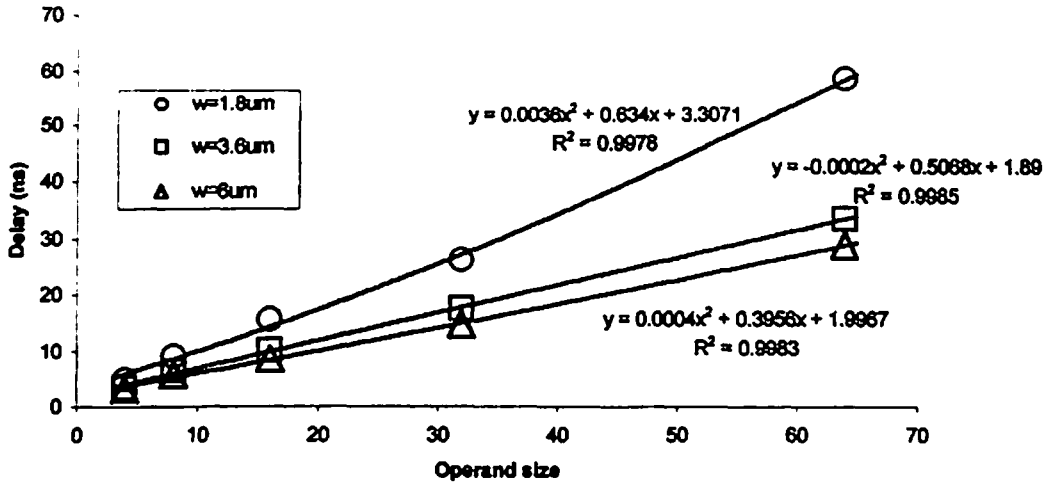


Fig. 6.2. c Propagation delay versus operand size of Carry skip adder Complementary pass transistor logic

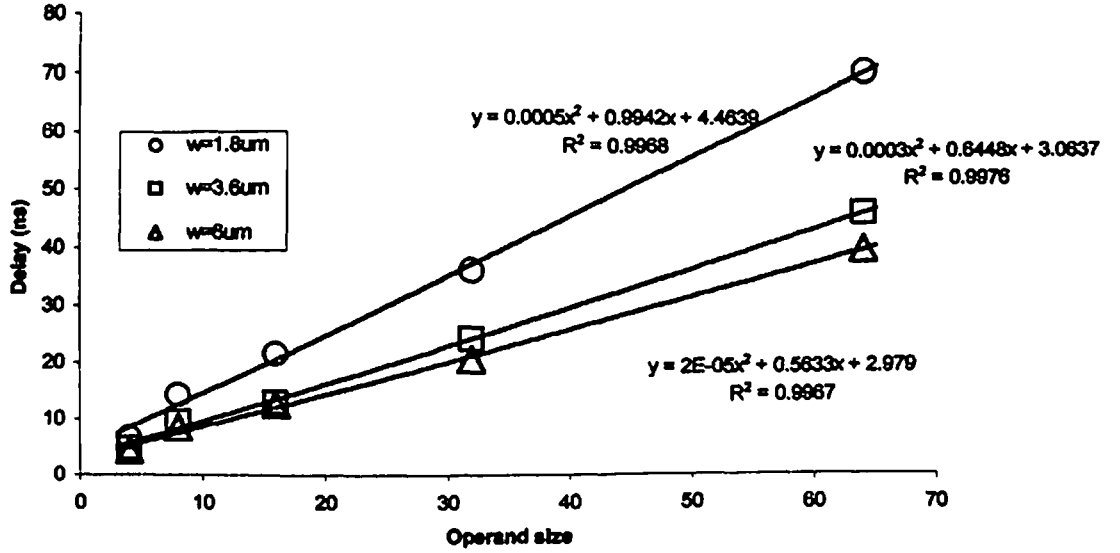


Fig. 6.2.d Propagation delay versus operand size of Carry skip adder Dual pass transistor logic

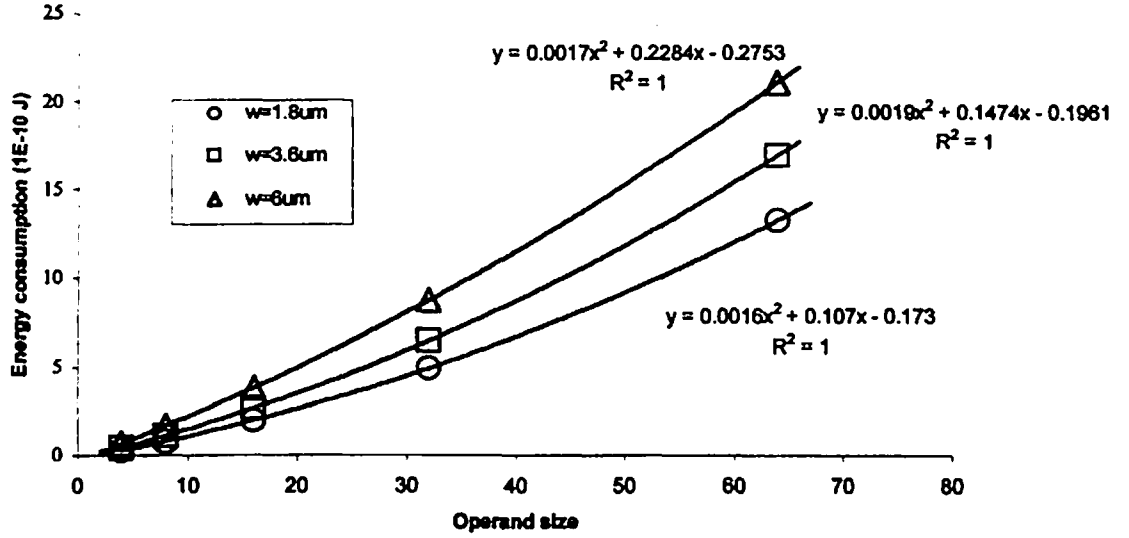


Fig. 6.2. e Energy consumption versus operand size for Carry skip adder Fully static CMOS logic

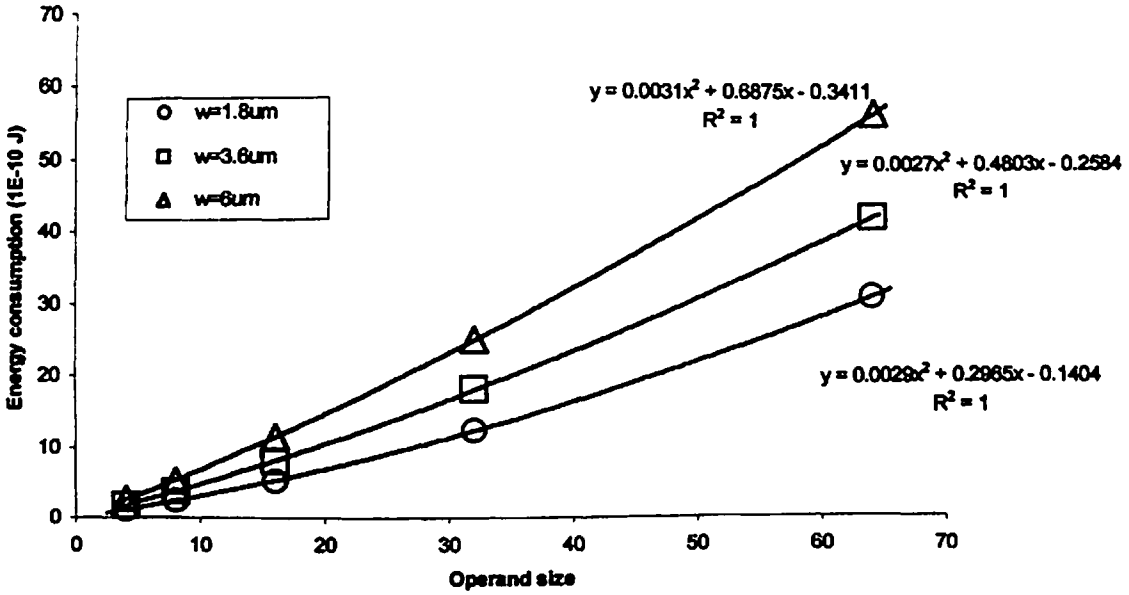


Fig. 6.2. f Energy consumption versus operand size of Carry skip adder Domino CMOS logic

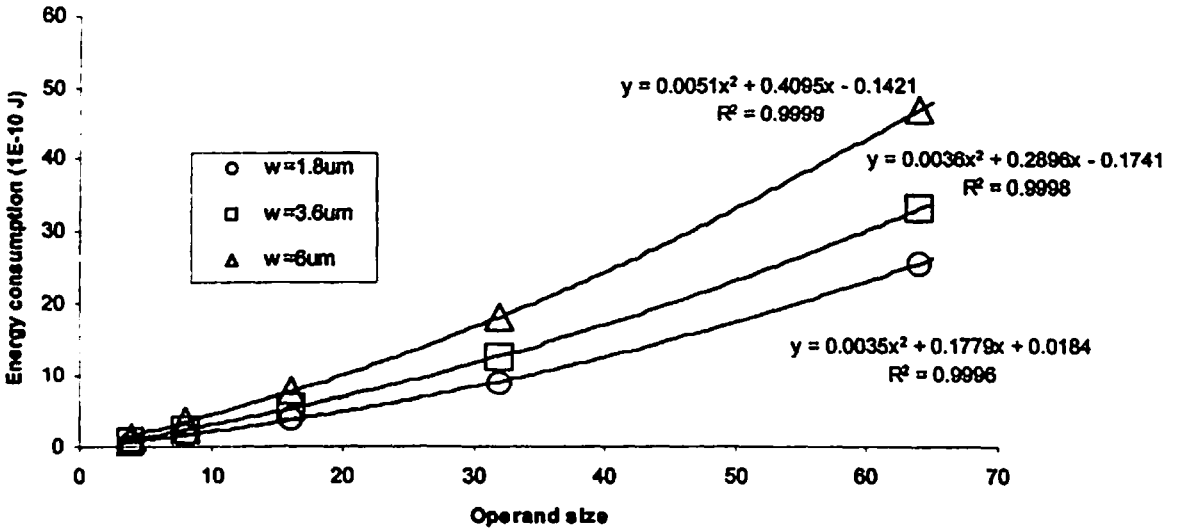


Fig. 6.2. g Energy consumption versus operand size of Carry skip adder Complementary pass transistor logic

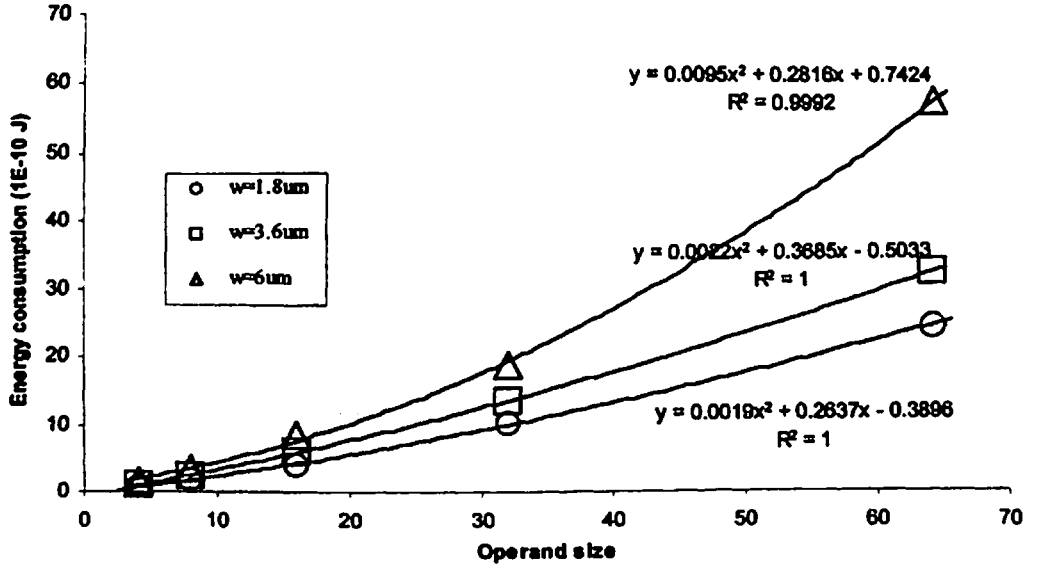


Fig. 6.2. h Energy consumption versus operand size of Carry skip adder Dual pass transistor logic

6.3 RESULTS OF CARRY SELECT ADDER

The simulation results of carry select adder designs are shown in Figs. 6.3.

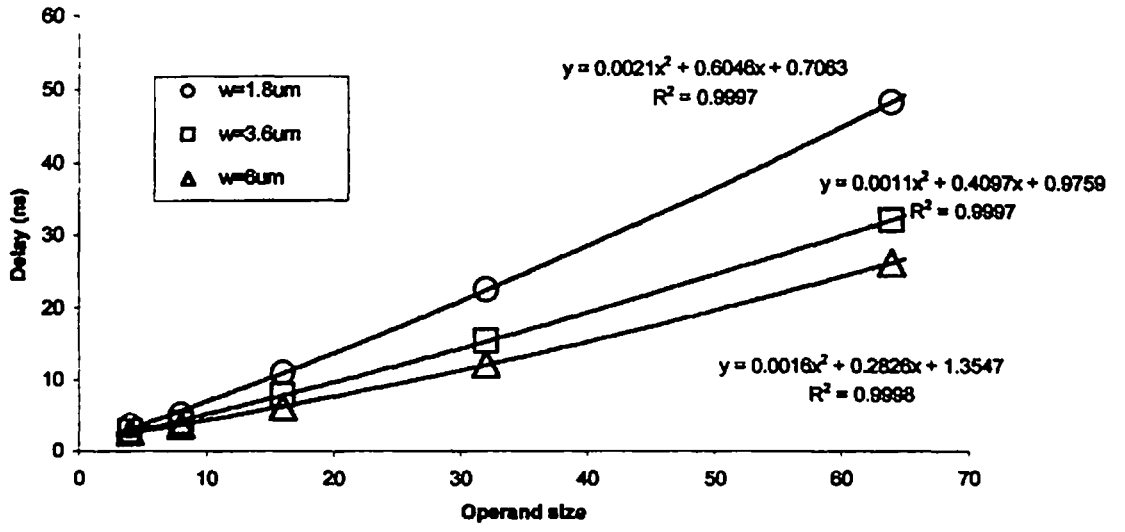


Fig. 6.3. a Propagation delay versus operand size for Carry select adder Fully static CMOS logic

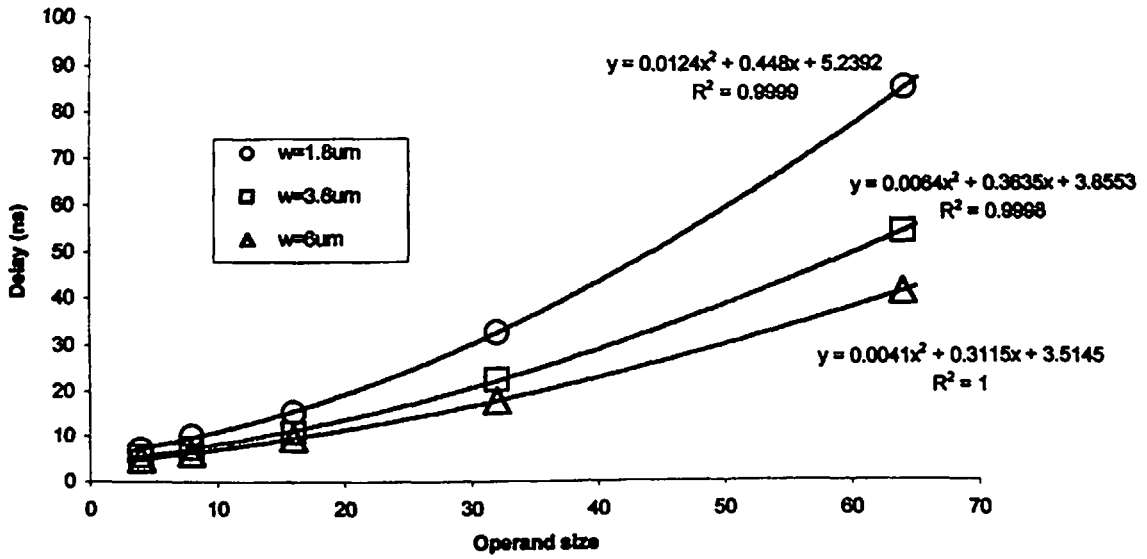


Fig. 6.3. b Propagation delay versus operand size for Carry select adder Domino CMOS logic

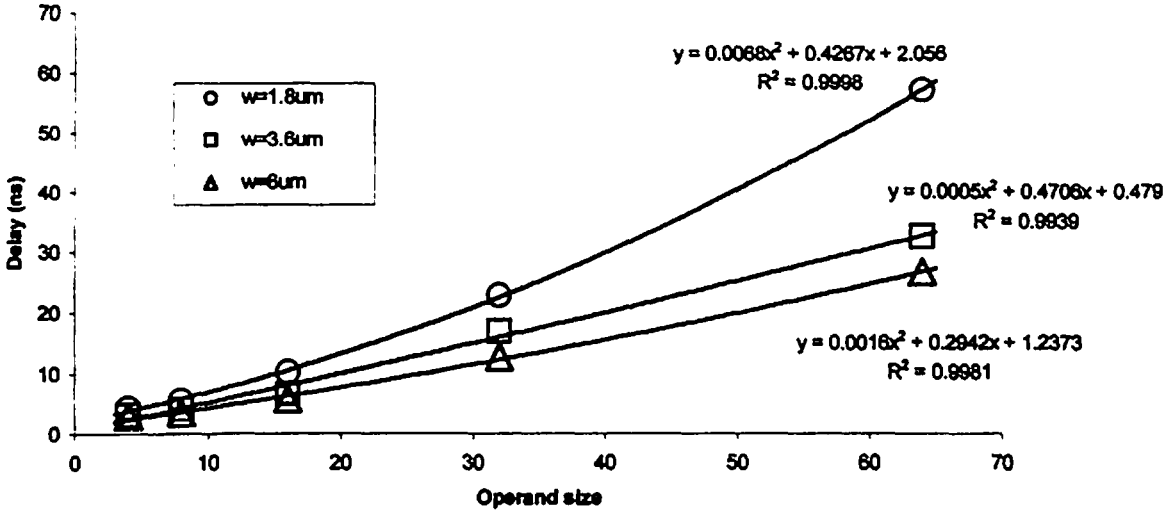


Fig. 6.3. c Propagation delay versus operand size for Carry select adder Complementary pass transistor logic

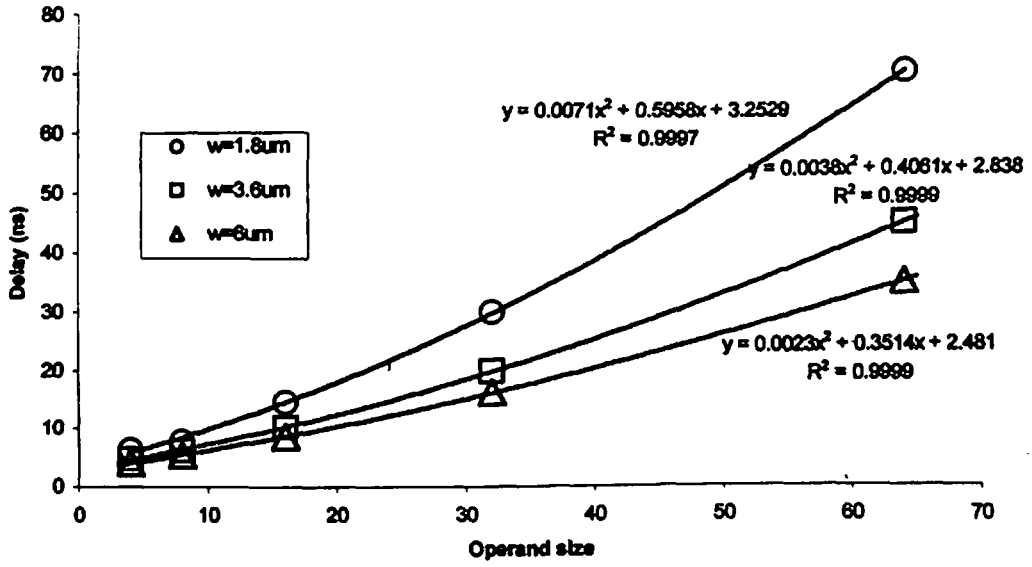


Fig. 6.3. d Propagation delay versus operand size for Carry select adder Dual pass transistor logic

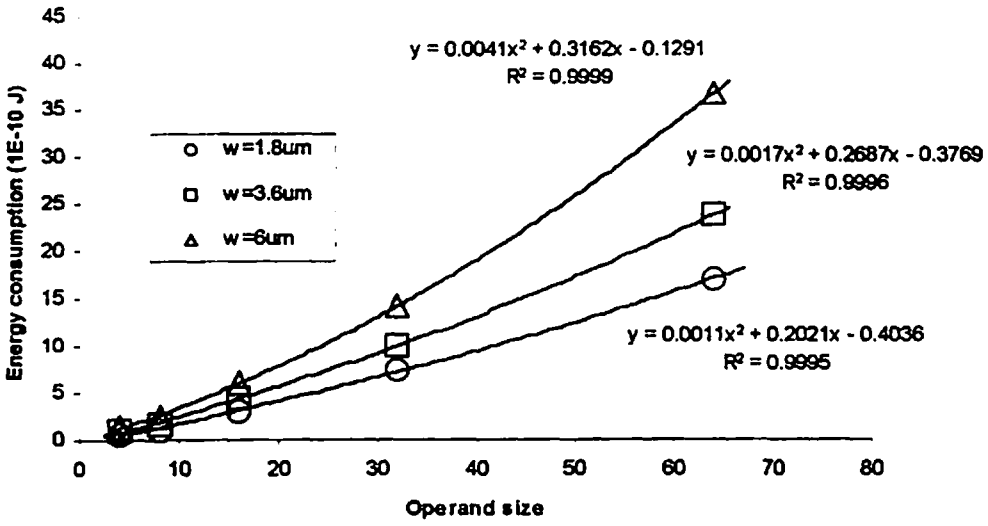


Fig. 6.3. e Energy consumption versus operand size for Carry select adder Fully static CMOS logic

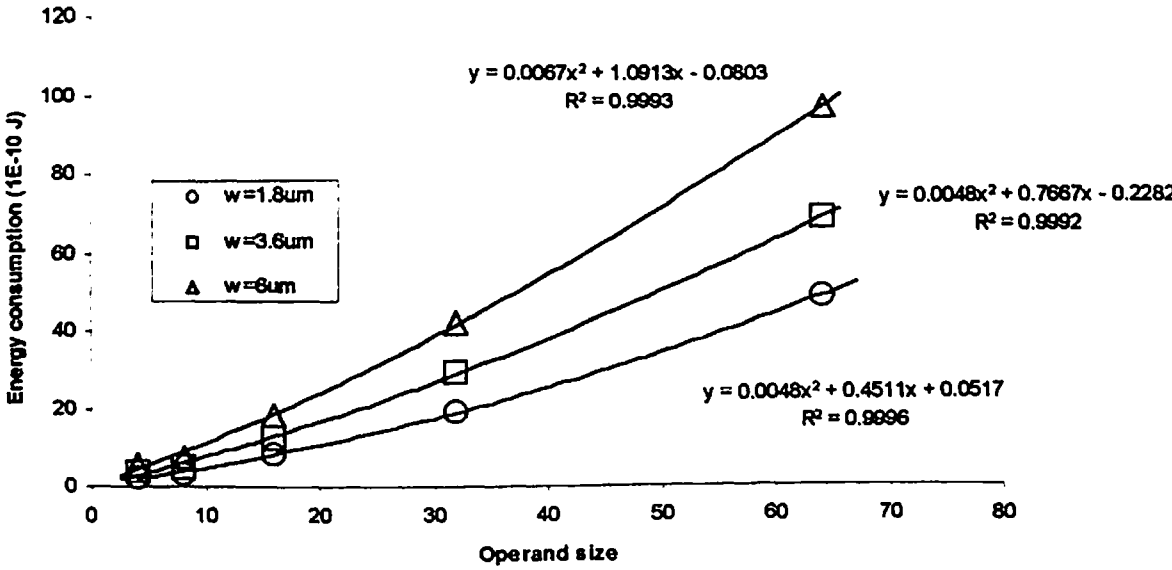


Fig. 6.3. f Energy consumption versus operand size for Carry select adder Domino CMOS logic

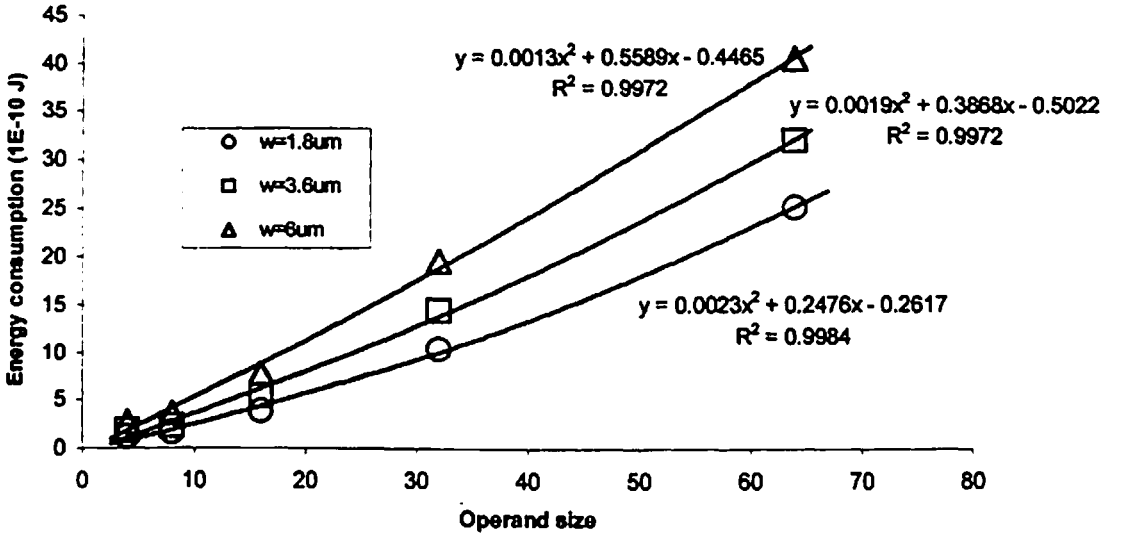


Fig. 6.3. g Energy consumption versus operand size for Carry select adder Complementary pass transistor logic

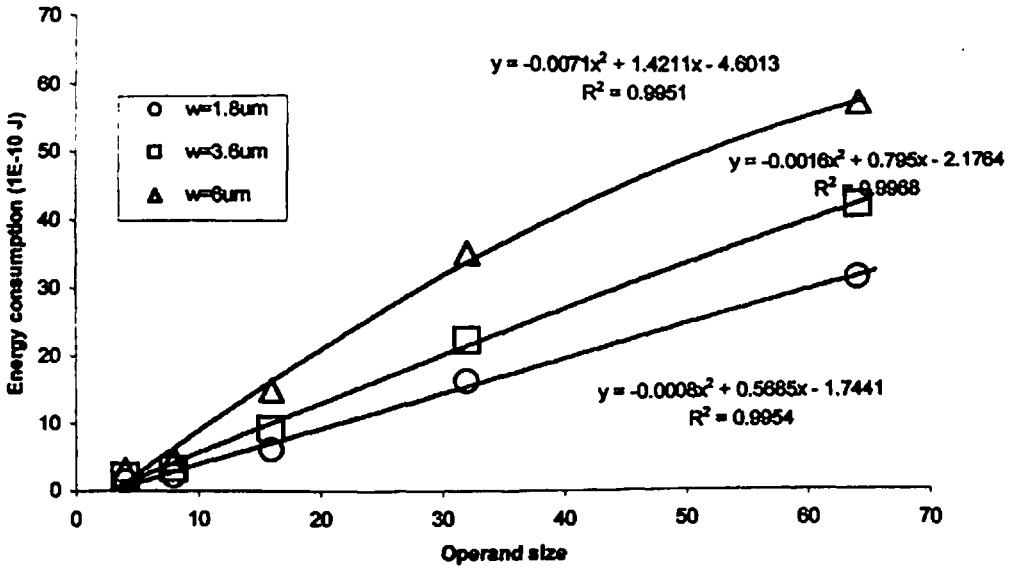


Fig. 6.3. h Energy consumption versus operand size for Carry select adder Dual pass transistor logic

6.4 RESULTS OF CONDITIONAL SUM ADDER

The simulation results of conditional sum adder are shown in Figs. 6.4

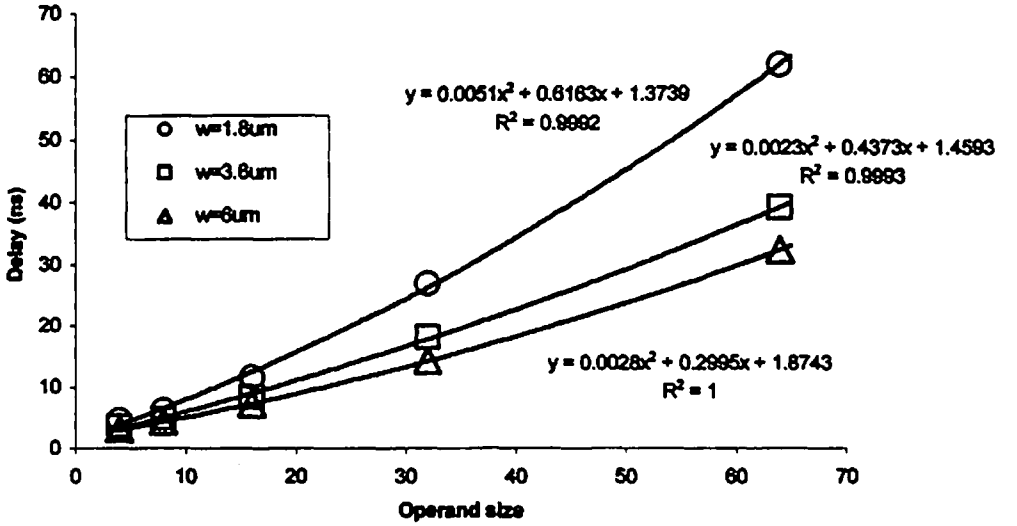


Fig. 6.4. a Propagation delay versus operand size for Conditional sum adder Fully static CMOS logic

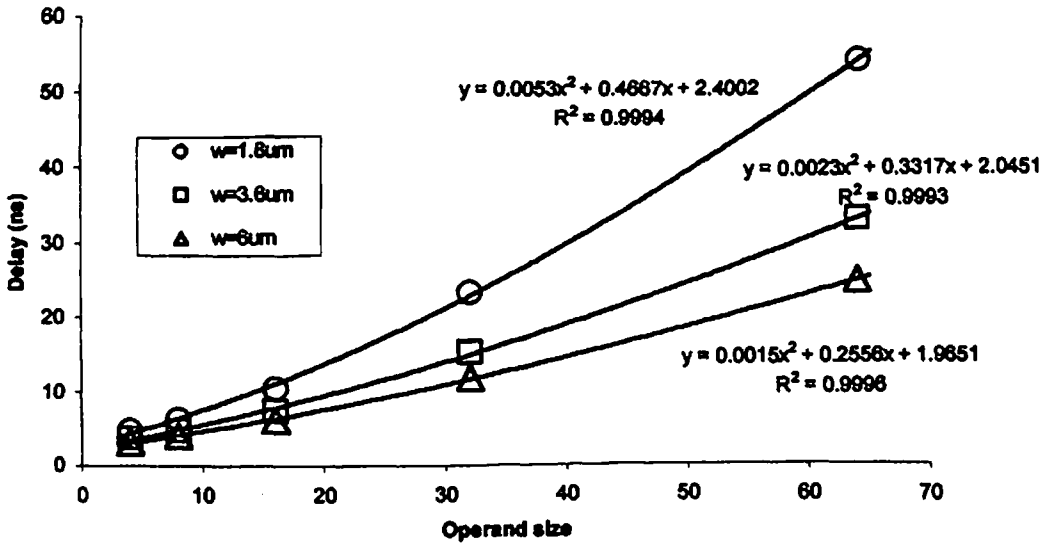


Fig. 6.4. b Propagation delay versus operand size of Conditional sum adder Domino CMOS logic

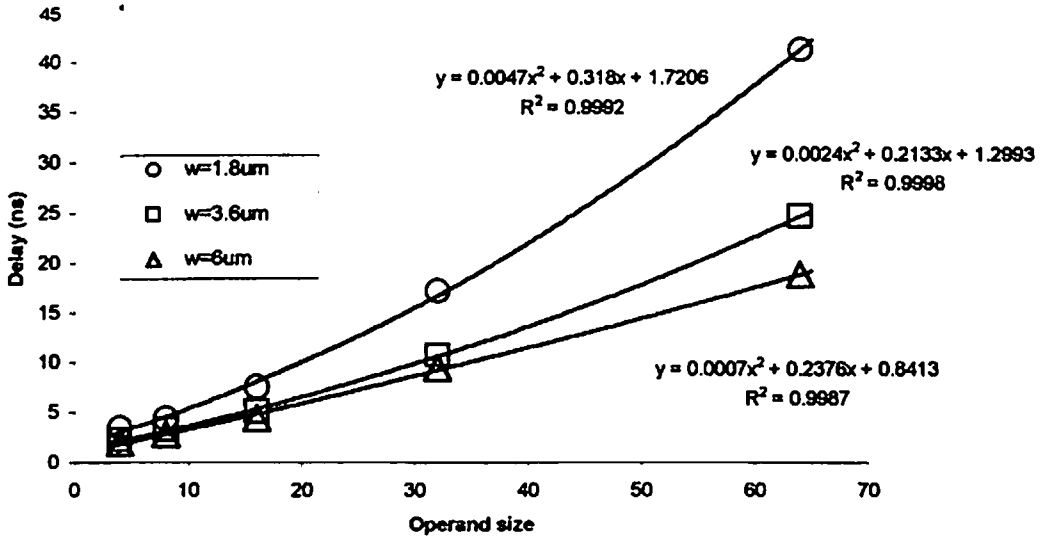


Fig.6.4. c Propagation delay versus operand size for Conditional sum adder Complementary pass transistor logic

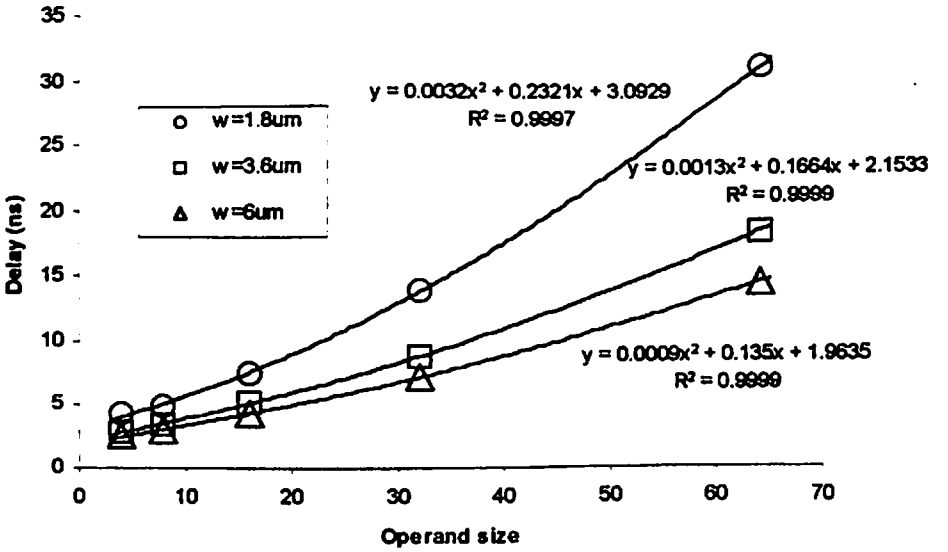


Fig. 6.4. d Propagation delay versus operand size for Conditional sum adder Dual pass transistor logic

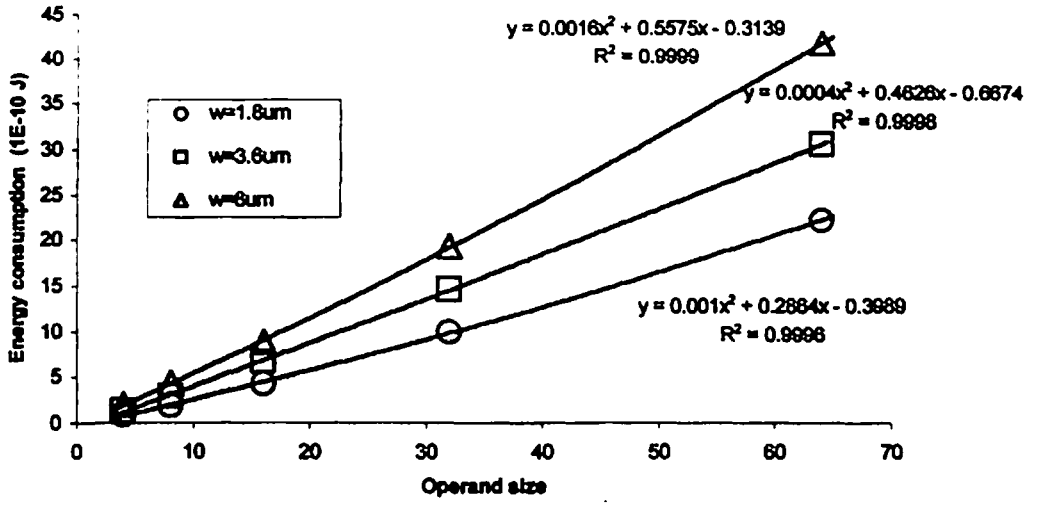


Fig. 6.4. e Energy consumption versus operand size for Conditional sum adder Fully static CMOS logic

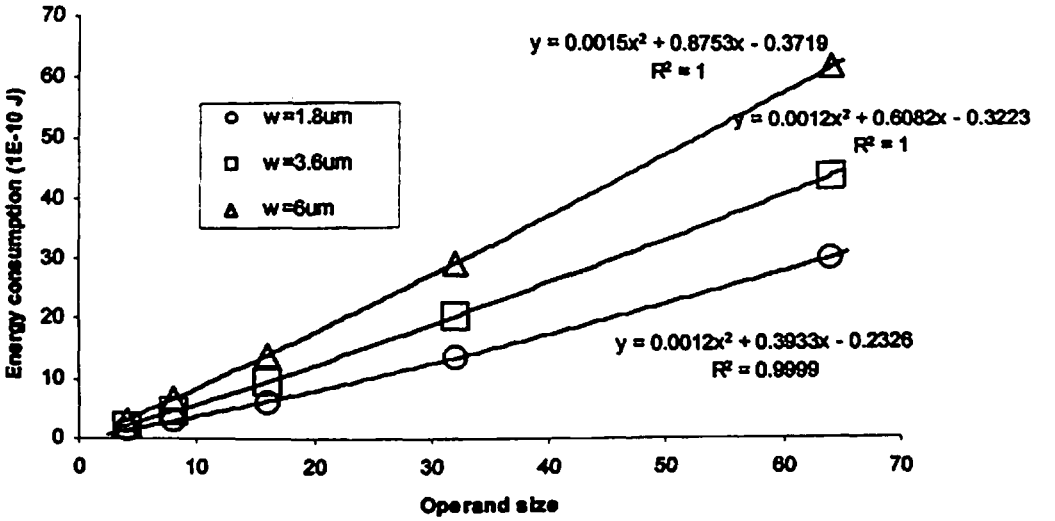


Fig. 6.4. f Energy consumption versus operand size for Conditional sum adder Domino CMOS logic

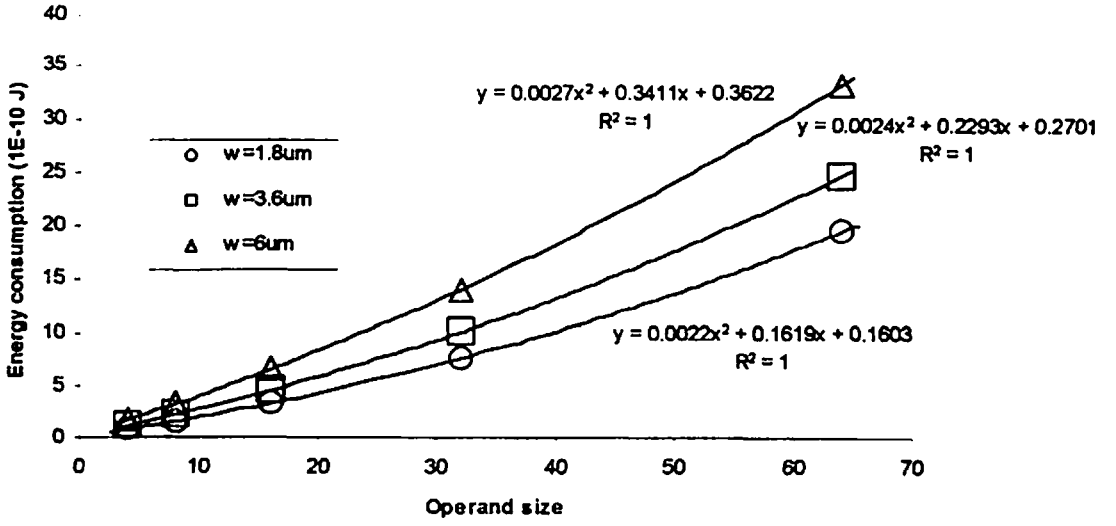


Fig. 6.4. g Energy consumption versus operand size for Conditional sum adder Complementary pass transistor logic

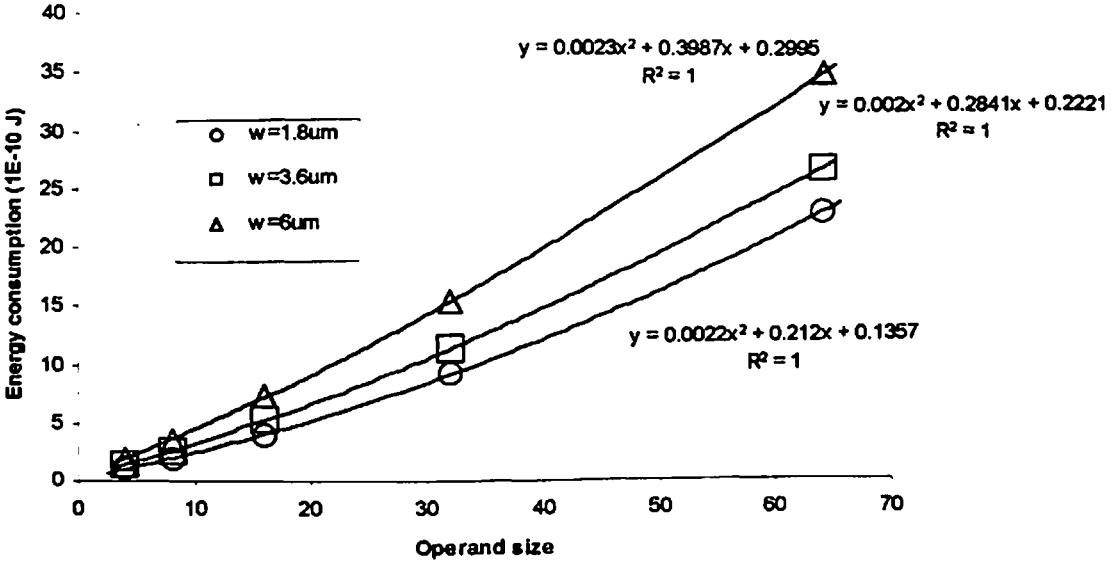


Fig. 6.4. h Energy consumption versus operand size for Conditional sum adder Dual pass transistor logic

6.5 RESULTS OF CARRY LOOK-AHEAD ADDER

The simulation results of carry look-ahead adder designs are shown in Figs. 6.5.

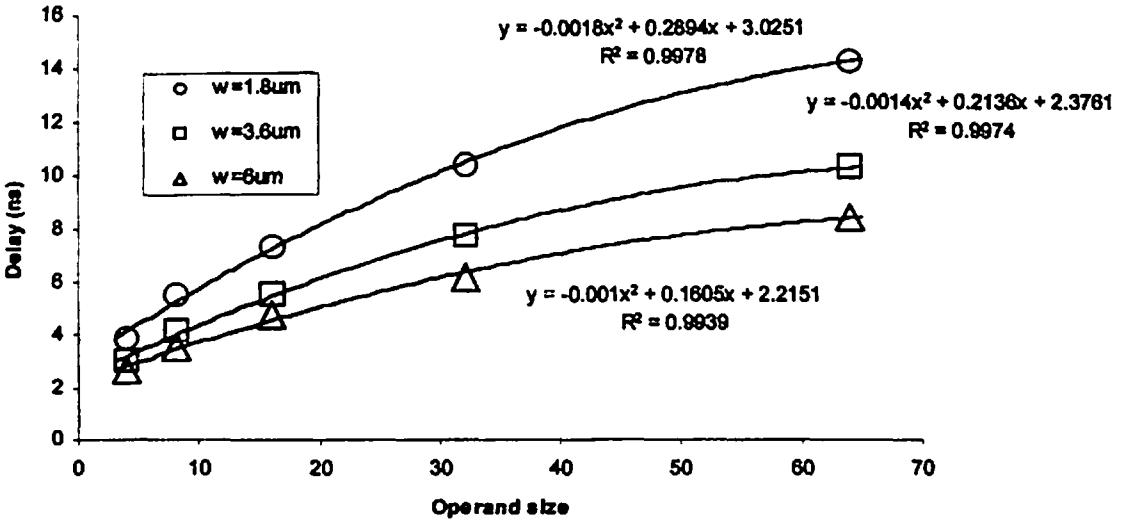


Fig. 6.5. a Propagation delay versus operand size for Carry look-ahead adder Fully static CMOS logic

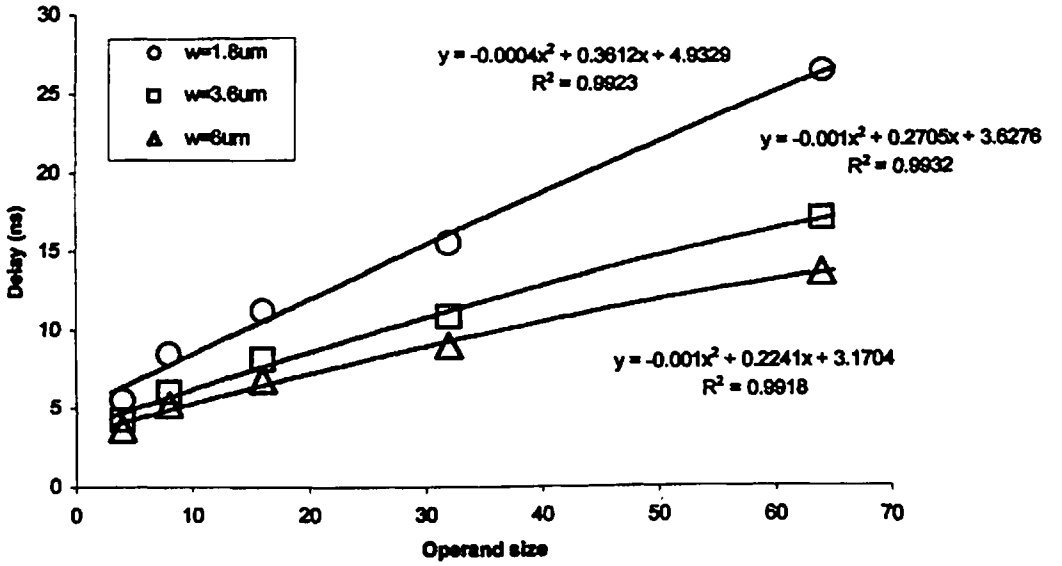


Fig. 6.5. b Propagation delay versus operand size for Carry look-ahead adder Domino CMOS logic

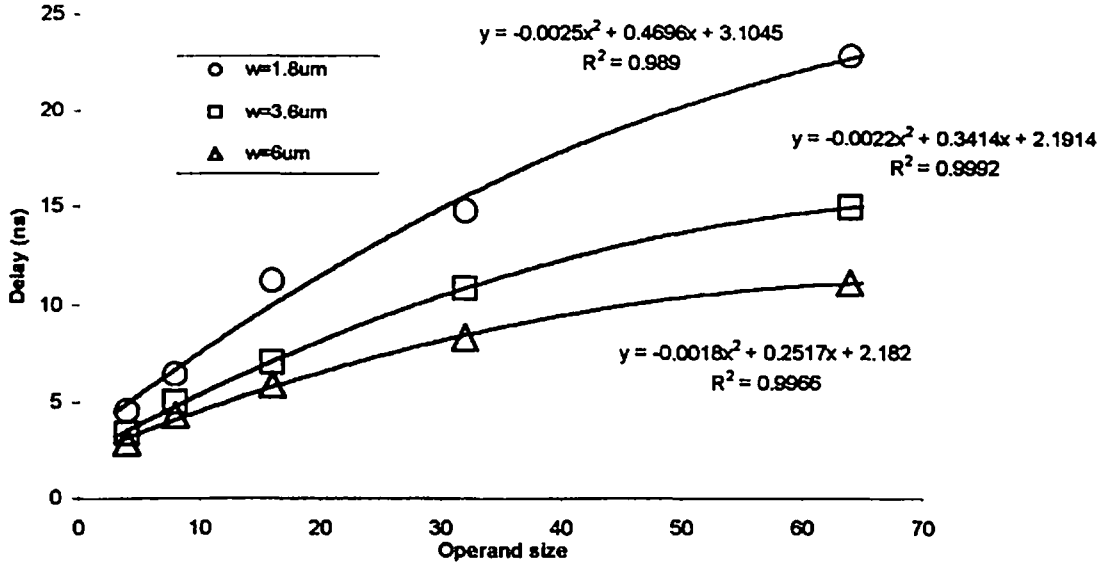


Fig. 6.5. c Propagation delay versus operand size for Carry look-ahead adder Complementary pass transistor logic

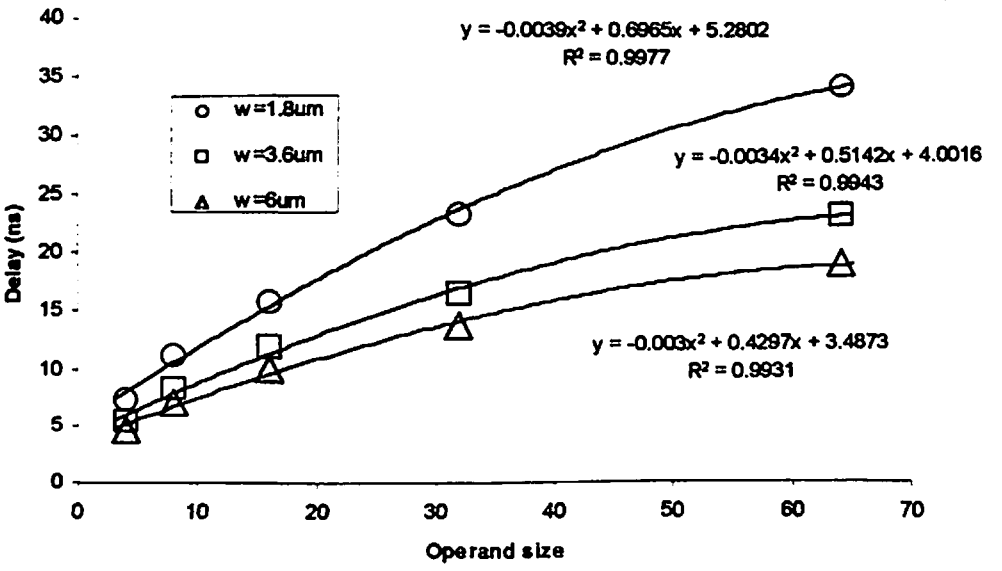


Fig. 6.5. d Propagation delay versus operand size for Carry look-ahead adder Dual pass transistor logic

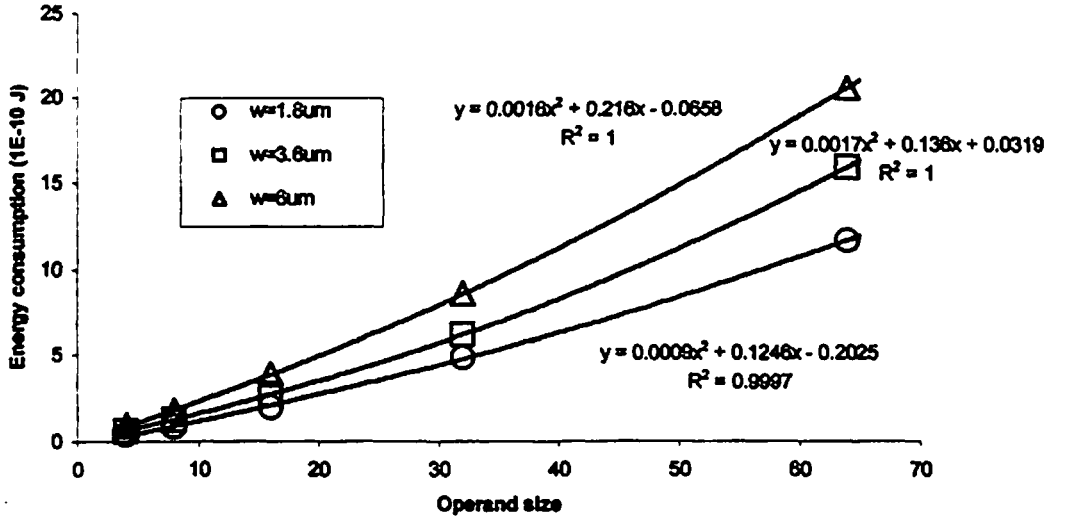


Fig. 6.5. e Energy consumption versus operand size for Carry look-ahead adder Fully static CMOS logic

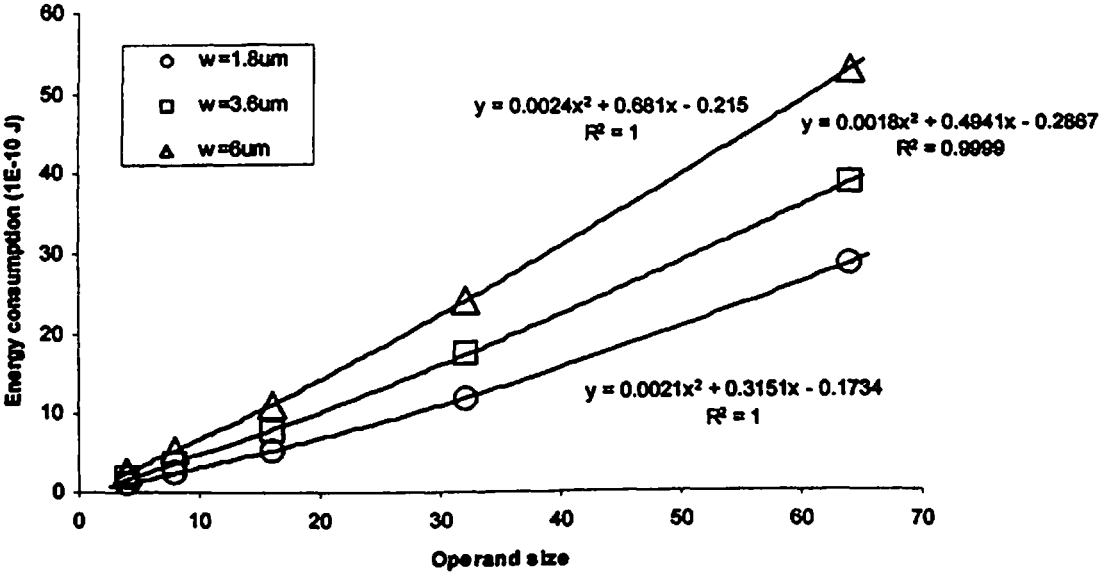


Fig. 6.5. f Energy consumption versus operand size for Carry look-ahead adder Domino CMOS logic

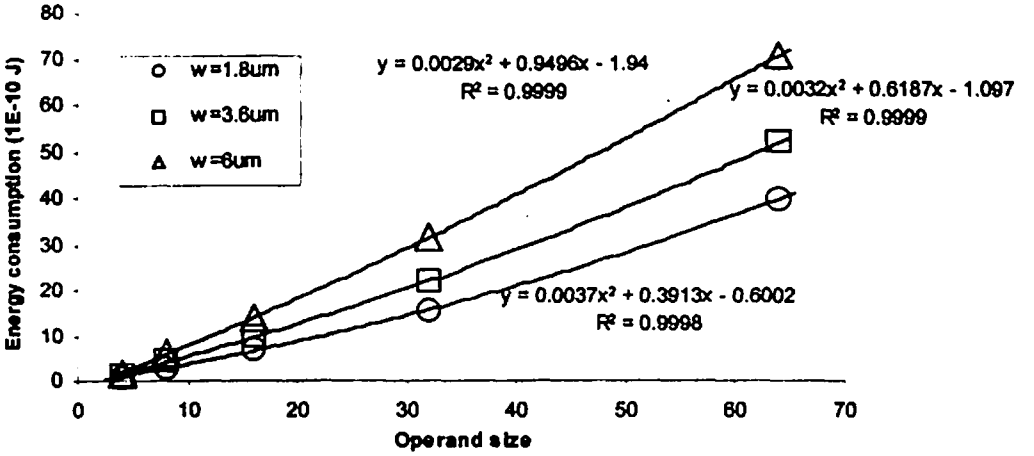


Fig. 6.5. g Energy consumption versus operand size for Carry look-ahead adder Complementary pass transistor logic

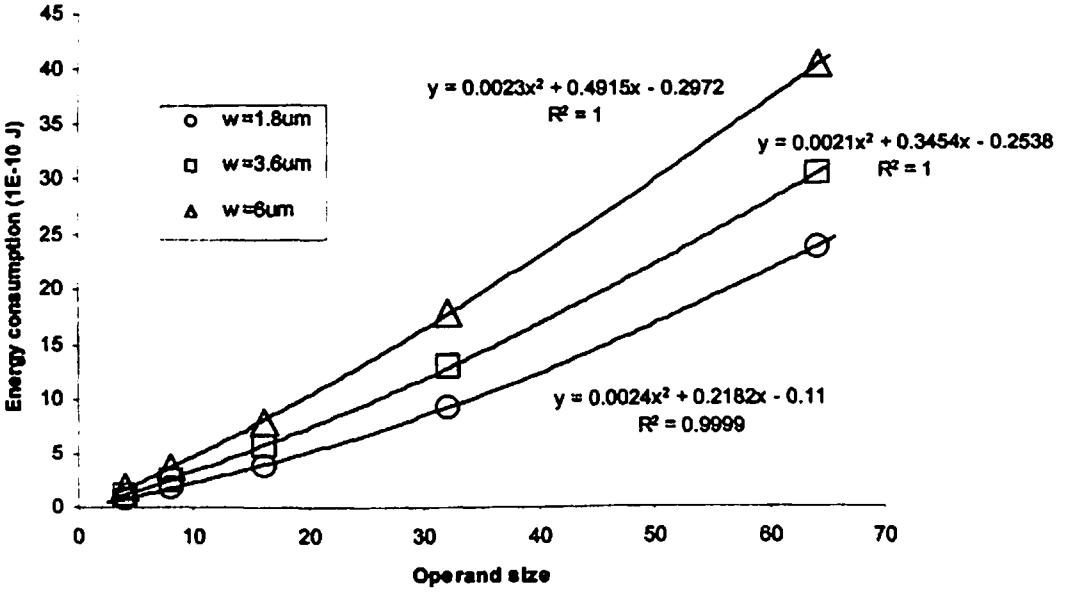


Fig. 6.6. h Energy consumption versus operand size for Carry look-ahead adder Dual pass transistor logic

CHAPTER 7

MODELING OF DIFFERENT ADDERS

The primary objective of our modeling efforts has been to develop a model that can estimate the worst-case propagation delay and average energy consumption per addition for each adder design. The results obtained from that model can be used for selecting an adder design that satisfies the area, energy consumption and throughput requirements.

The effect of circuit parasitics have been included in the model as they have a significant impact on the speed and energy consumption due to increased capacitive loading and R-C delay through the wires. Thus for modeling purposes, the parasitic degradation factor of each design due to circuit parasitics has been obtained using simulation data. The variation of parasitic degradation factor with operand size of an adder is then used to calculate the worst-case propagation delay of any adder with operand size between 4-bits to 64-bits in multiples of four as the basic unit is 4-bit adder block in each design.

7.1 MODELING OF WORST-CASE PROPAGATION DELAY (DELAY MODEL)

The modeling of worst-case propagation delay of each adder is done by taking the product of total number of gates in the critical delay path, time constant of the technology, average fan-out per gate in the critical delay path, and parasitic degradation factor.

$$\text{Propagation delay} = n \times \tau \times f \times \alpha \qquad \ldots 7.1$$

where
- n = number of gates in critical delay path
- τ = time constant of the technology
- f = average fan-out per gate in critical delay path
- α = parasitic degradation factor

The time constant of the technology (τ) has been taken as the time delay in driving one gate capacitance (C_g) through the resistance of one transistor (R_{nmos} or R_{pmos}) without

including the circuit parasitics using SPICE. Here the value of (τ) is taken as the average of ($R_{pmos} \times C_g$) and ($R_{nmos} \times C_g$). Table T21 lists the average fan-out per gate in the critical delay path obtained for different adder architectures. Table T22 lists the time constant of technology obtained for different transistor sizes.

TABLE T21: AVERAGE FAN-OUT PER GATE (f) IN CRITICAL DELAY PATH FOR DIFFERENT ARCHITECTURES FOR DIFFERENT LOGIC DESIGN STYLES

Adder architecture	Average fan-out per gate (f)			
	Fully static CMOS	Domino CMOS	CPL	DPL
Ripple carry	3.11	2.44	2.66	2.42
Carry skip	3.27	2.42	2.36	2.50
Carry select	4.00	3.21	2.66	2.63
Carry look-ahead	4.66	3.30	3.25	3.89
Conditional sum	5.82	2.77	3.11	5.20

TABLE T22: TIME CONSTANT (τ) FOR DIFFERENT TRANSISTOR SIZES

(w/l)	Time constant (τ) in seconds $\times 10^{-9}$
1.5	0.0885
3.0	0.0700
5.0	0.0600

Results show that the average fan-out per gate varies with the type of architecture and logic design style. The time constant (τ) decreases with increase in transistor width. The parasitic degradation factor (α) is obtained by equating the propagation delay value obtained from simulation to the equation 7.1. The value of parasitic degradation factor obtained for each adder design is given in tables T30 to T49 listed in ***Appendix B4.*** Graphs showing variation of (α) with operand size for different adders are shown in Figs. 7.1, 7.2, 7.3, 7.4, and 7.5. A second order polynomial is fitted to the data.

Results show that the value of (α) increases with increase in operand size. This is because interconnection lengths and hence node capacitances in an adder design are

dependent on the shape of layout and the number of rows in the layout. Here, we have used the optimal area settings for generation of square core layouts in the layout editor with all inputs applied from the top and all outputs collected at the bottom of the layout. It has been observed from the extracted capacitance values for 4-bit and 16-bit ripple carry adders shown in ***Appendix A7*** that the capacitances at similar circuit nodes are greater for 16-bit operand size. This is due to longer interconnection lengths as the wires have to be routed through many rows in standard cell based layouts for larger operand sizes for square layout settings. For most of the cases, the change in the value of (α) from 4-bit to 64-bit operand size is small which further decreases as transistor size increases. Adder designs with minimum transistor size are most affected by circuit parasitics (value of (α) is higher) due to relatively higher contribution of parasitics. The value of (α) for an adder design decreases as transistor size increases because of the interconnection parasitics becoming less significant.

7.2 MODELING OF ENERGY CONSUMPTION PER ADDITION (ENERGY MODEL)

The dynamic power dissipation of a circuit depends primarily on the number of logic transitions per unit time. As a result, the average number of logic transitions per operation has been used as the basis for determining the power dissipation of adders. The average energy consumption per addition for an adder is then calculated as

$$\text{Average energy consumption} = \hat{E} \times n_s \times \alpha \times \chi \times g \times \Psi \qquad \ldots 7.2$$

where $\hat{E}$ = energy consumption in driving one gate capacitance (C_g) through a transistor

n_s = average number of gate output transitions in a design

α = parasitic degradation factor

χ = average load capacitance at the output of a gate in units of C_g in an adder design

g = glitch factor

Ψ = average weight-factor per gate in an adder design for energy consumption due to switching of internal nodes of the gate

The average number of gate output transitions (n_s) in a 4-bit adder design has been obtained by taking average of the number of gate output nodes switching for all the

different transitions from an input pattern to another input pattern. For 4-bit adders with two 4-bit operands and 1 carry input, the total number transitions are $2^9 \times 2^9$. ***Appendix A8*** lists the programs used for calculating (n_s) for different 4-bit adders for fully static logic. Similar programs for other logic design styles have been used for estimating (n_s). The value of (n_s) cannot be obtained for other operand sizes because of computation limitations of the software used.

Energy consumption ($\bar{E}$) in driving one gate capacitance ($1C_g$ load capacitance) through a transistor has been determined for different transistor sizes using SPICE without including circuit parasitics. The energy consumption has also been estimated for the case when high-level input is applied through a pass transistor to CMOS inverter driving one gate capacitance to account for the static power dissipation due to threshold voltage degradation. ***Appendix A9*** shows the current flow through the CMOS inverter with a full logic swing at the input and degraded high-level logic at the input. Table T23 lists the values of ($\bar{E}$) in driving one gate capacitance through an inverter with full high-level logic, and through an inverter with degraded high-level logic at the input. The value of energy consumption for degraded high-level logic ($E_{degraded}$) has been found to be high as static current flows continuously as long as input remains high. The energy consumption for adders designed using CPL design style has been estimated using ($E_{degraded}$) instead of ($\bar{E}$) for one-fourth nodes of the total switching nodes and using ($\bar{E}$) for three-fourth nodes of total switching nodes. The two values have been then added to calculate total energy consumption per addition.

Average load capacitance at the output of a gate (γ) has been estimated for different 4-bit adder architectures from circuit schematic. Tables T24, and T25 give the values of (n_s), and (γ).

The glitch factor (g) has been included to account for the spurious transitions before nodes acquire stable logic values. For static logic, (g) is taken as 1.5 due to extra transitions made by the nodes of 'sum' circuit. For domino CMOS logic, nodes can make only one transition at a time. So, value of (g) is taken as 1. The average weight-factor (Ψ) accounts for the energy consumption due to switching of circuit nodes internal to the gate. The value of (Ψ) has been estimated by taking average of weight-factors of different gates used in the design for a particular logic design style. The value of the weight-factor for an individual gate has been determined by taking the ratio of the energy consumed by

the gate to the energy ($\tilde{E}$) in driving one gate capacitance for same output conditions. Table T26 lists the average weight-factor (Ψ) per gate for different logic design styles.

ABLE T23: ENERGY CONSUMPTION ($\tilde{E}$) IN DRIVING 1 GATE CAPACITANCE FOR DIFFERENT TRANSISTOR SIZES

(w/l)	Energy consumption in driving one gate capacitance ($J\times10^{-13}$)	
	Through a CMOS inverter with full high-level logic at the input ($\tilde{E}$)	Through a CMOS inverter with degra -ded high-level logic at the input ($E_{degraded}$)
1.5	1.40	32.18
3.0	3.06	65.40
5.0	4.99	109.29

TABLE T24: AVERAGE NUMBER OF GATE OUPUT TRANSITIONS (n_g) FOR DIFFERENT 4-BIT ADDERS AND DIFFERENT LOGIC DESIGN STYLES

Logic design style	Ripple carry adder n_g	Carry skip adder n_g	Carry select adder n_g	Conditi- onal sum adder n_g	Carry look- ahead adder n_g
Fully static CMOS	16.00	21.00	38.05	48.45	22.00
Domino CMOS	28.00	32.80	69.58	42.99	23.61
CPL	26.00	28.70	53.13	35.50	34.24
DPL	30.00	32.70	63.00	41.00	34.24

TABLE T25: AVERAGE LOAD CAPACITANCE (χ) PER GATE FOR DIFFERENT 4-BIT ARCHITECTURES FOR DIFFERENT LOGIC DESIGN STYLES

Adder architecture	Average load capacitance (χ) in units of C_g at the output of a gate			
	Fully static CMOS	Domino CMOS	CPL	DPL
Ripple carry	2.80	2.28	1.60	2.30
Carry skip	3.00	2.29	1.40	2.30
Carry select	3.50	3.70	1.60	2.46
Conditional sum	4.00	4.00	2.00	5.70
Carry look-ahead	5.50	4.40	1.60	3.00

TABLE T26: AVERAGE WEIGHT FACTOR (Ψ) PER GATE FOR DIFFERENT LOGIC DESIGN STYLES

Logic design style	Average weight-factor (Ψ) per gate (w/l)		
	1.5	3.0	5.0
Fully static CMOS	1.767	1.278	1.255
Domino CMOS	3.547	3.333	3.219
CPL	0.634	0.643	0.848
DPL	1.675	1.890	2.027

7.3 RESULTS OF RIPPLE CARRY ADDER

The parasitic degradation factors of ripple carry adder designs for different logic design style were obtained using model described in 7.1. The variation of α with operand size is shown in Figs. 7.1.

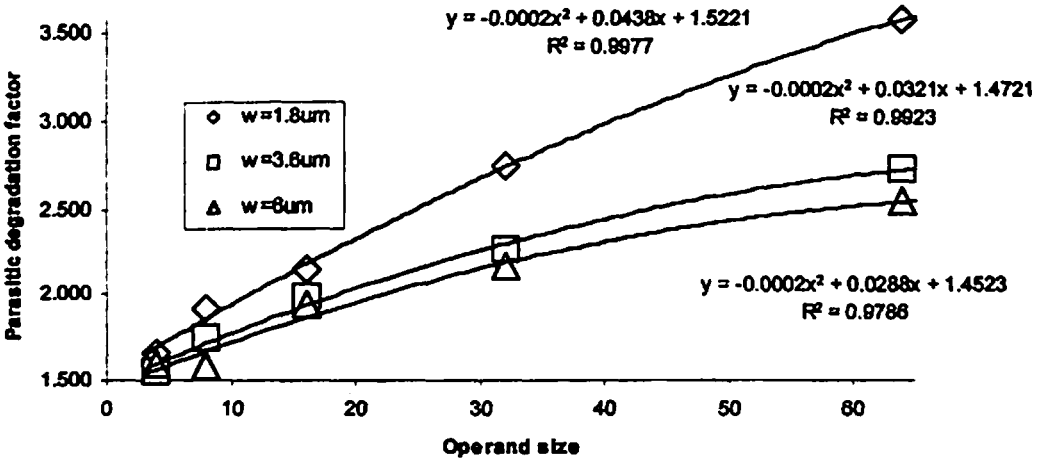


Fig. 7.1. a Parasitic degradation versus operand size for Ripple carry adder Fully static CMOS logic

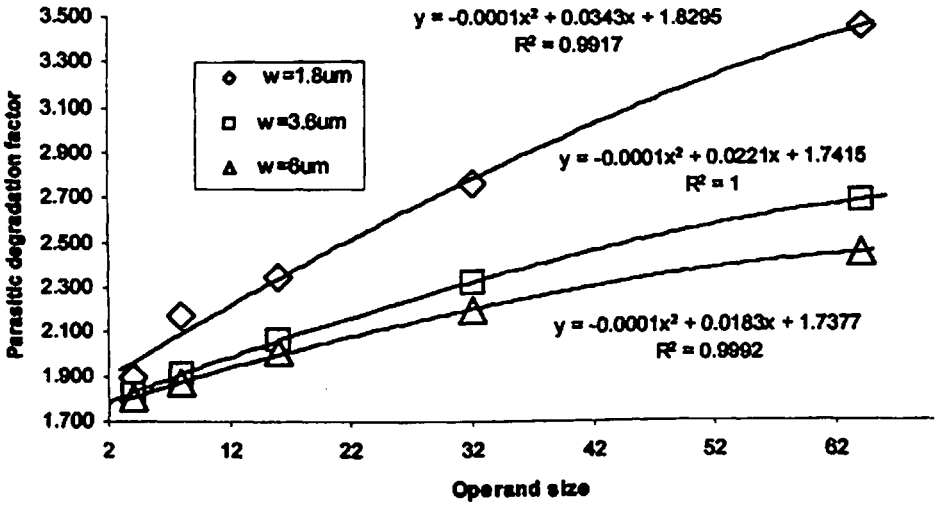


Fig. 7.1. b Parasitic degradation versus operand size for Ripple carry adder Domino CMOS logic

7.3 RESULTS OF RIPPLE CARRY ADDER

The parasitic degradation factors of ripple carry adder designs for different logic design style were obtained using model described in 7.1. The variation of α with operand size is shown in Figs. 7.1.

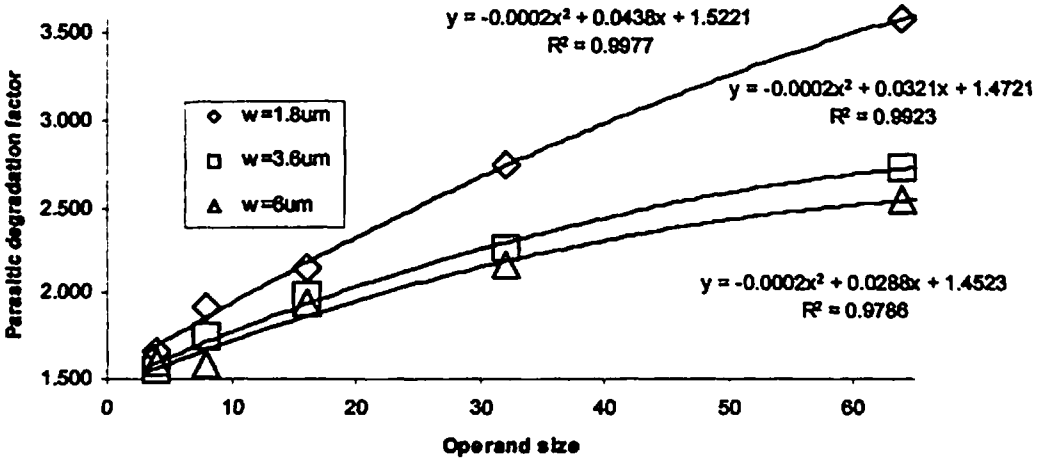


Fig. 7.1. a Parasitic degradation versus operand size for Ripple carry adder Fully static CMOS logic

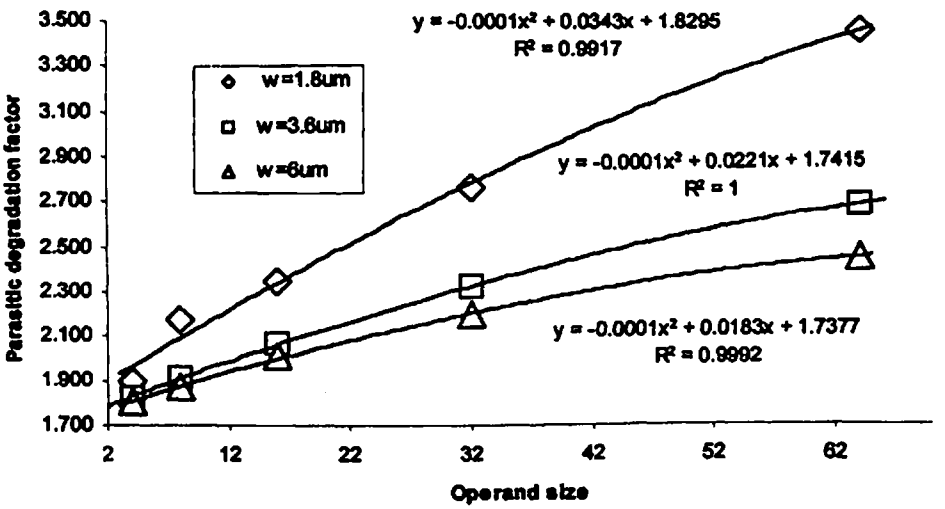


Fig. 7.1. b Parasitic degradation versus operand size for Ripple carry adder Domino CMOS logic

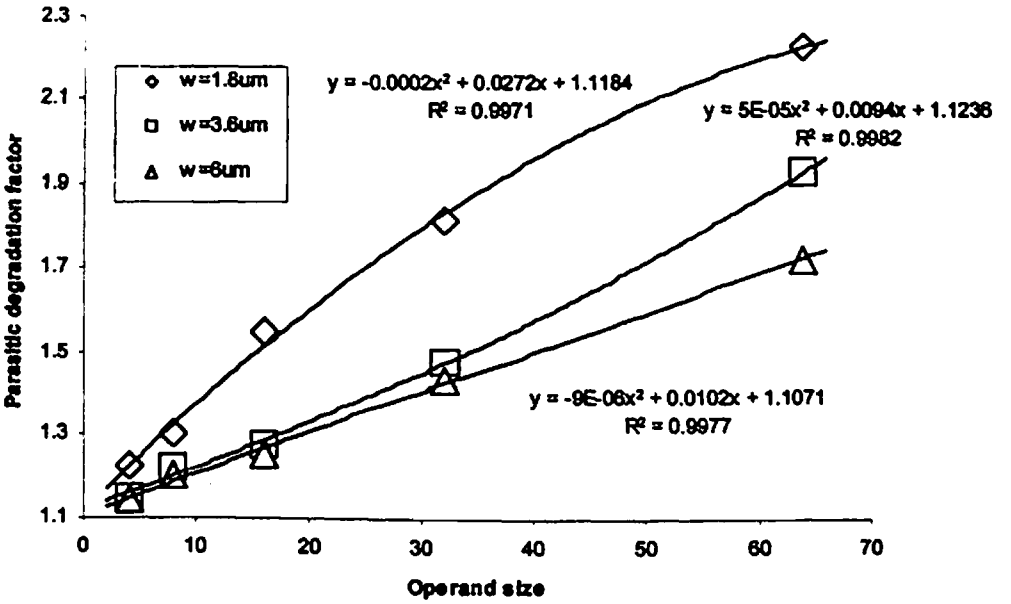


Fig. 7.1. c Parasitic degradation versus operand size for Ripple carry adder Complementary pass transistor logic

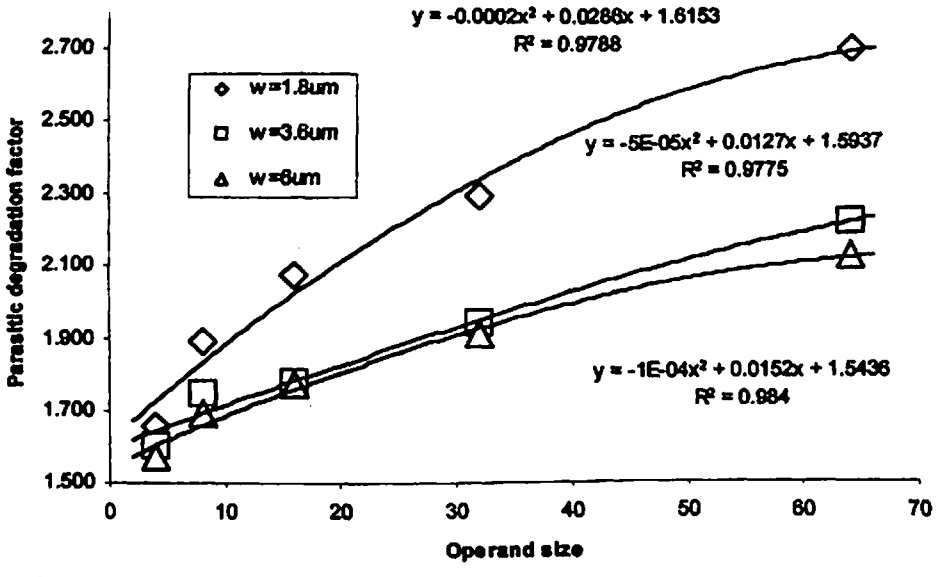


Fig. 7.1. d Parasitic degradation versus operand size for Ripple Carry Adder Dual pass transistor logic

7.4 RESULTS OF CARRY SKIP ADDER

The variation of parasitic degradation factor (α) with operand size of carry skip adder designs is shown in Figs. 7.2.

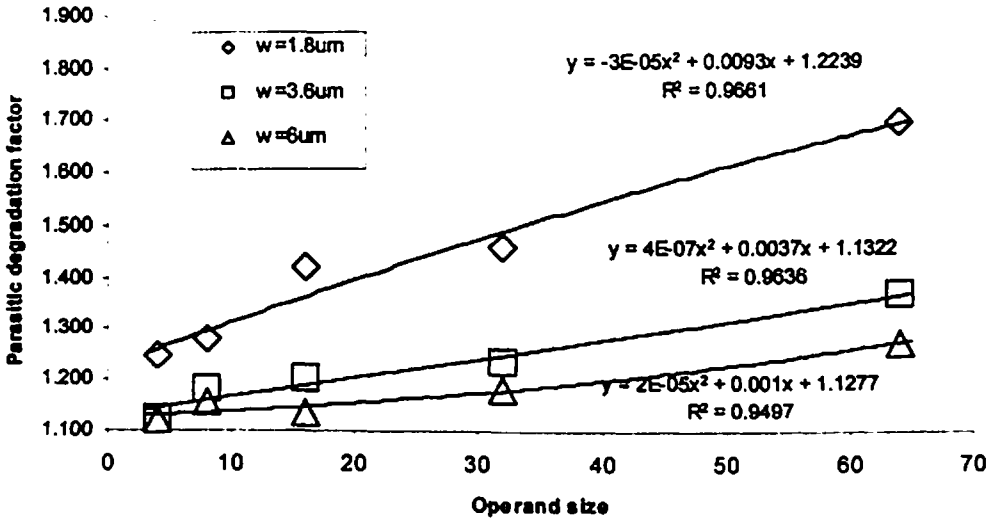


Fig. 7.2. a Parasitic degradation versus operand size of Carry skip adder Fully static CMOS logic

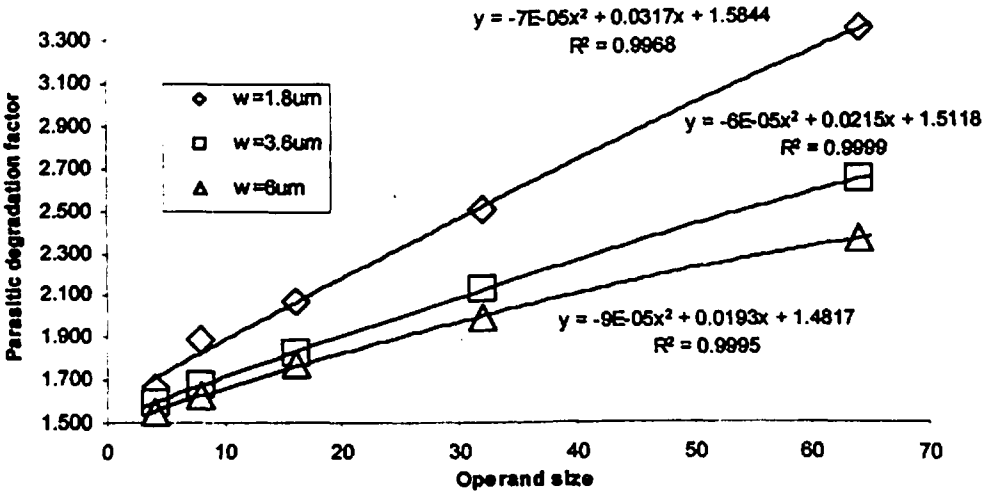


Fig. 7.2. b Parasitic degradation versus operand size of Carry skip adder Domino CMOS logic

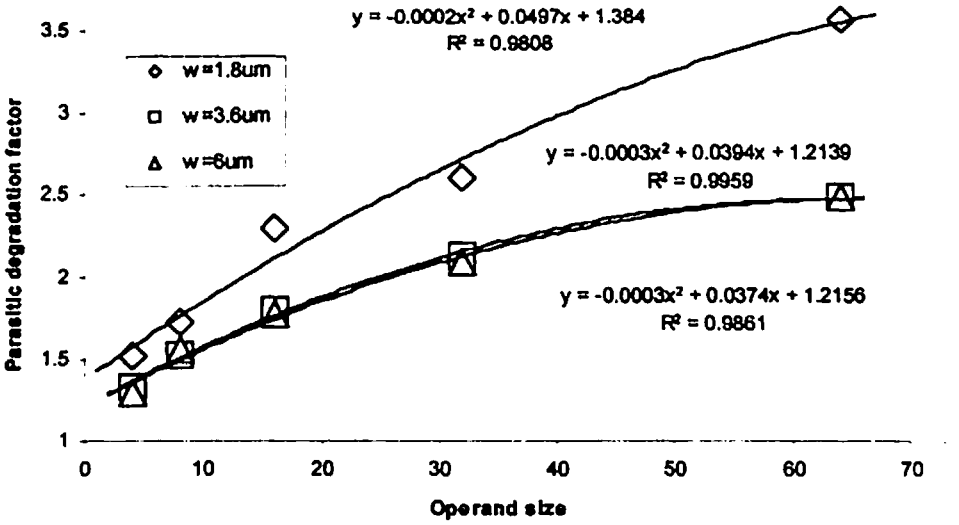


Fig. 7.2. c Parasitic degradation versus operand size of Carry skip adder Complementary pass transistor logic

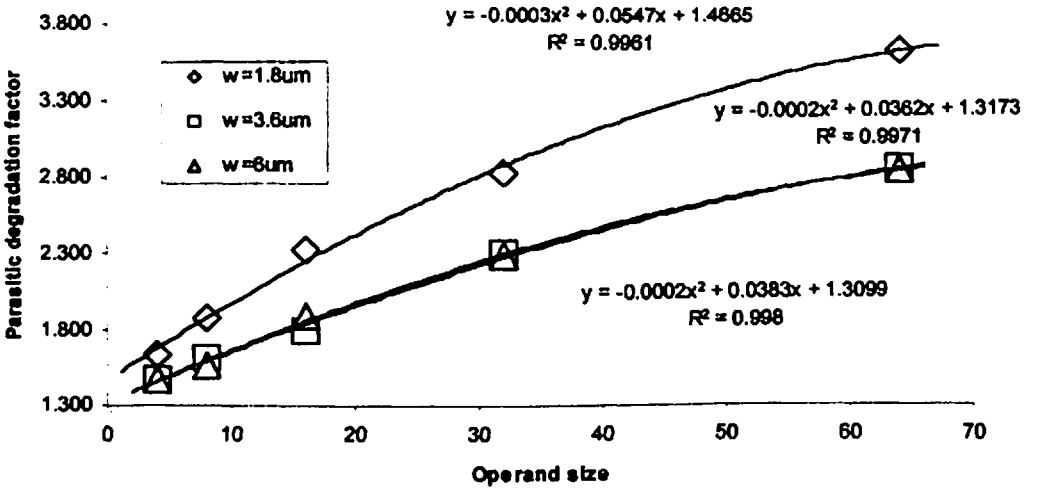


Fig. 7.2. d Parasitic degradation versus operand size of Carry skip adder Dual pass transistor logic

7.5 RESULTS OF CARRY SELECT ADDER

The variation of parasitic degradation factor (α) with operand size of carry select adder designs is shown in Figs. 7.3.

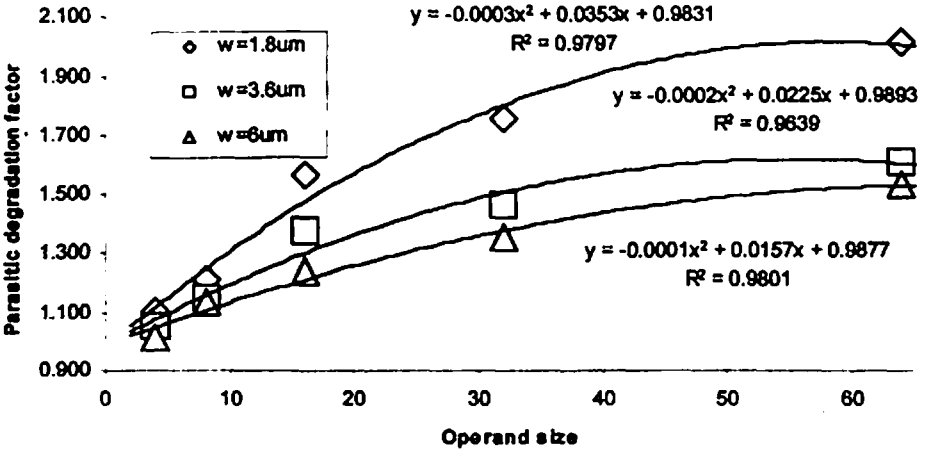


Fig. 7.3 a Parasitic degradation versus operand size of Carry select adder Fully static CMOS logic

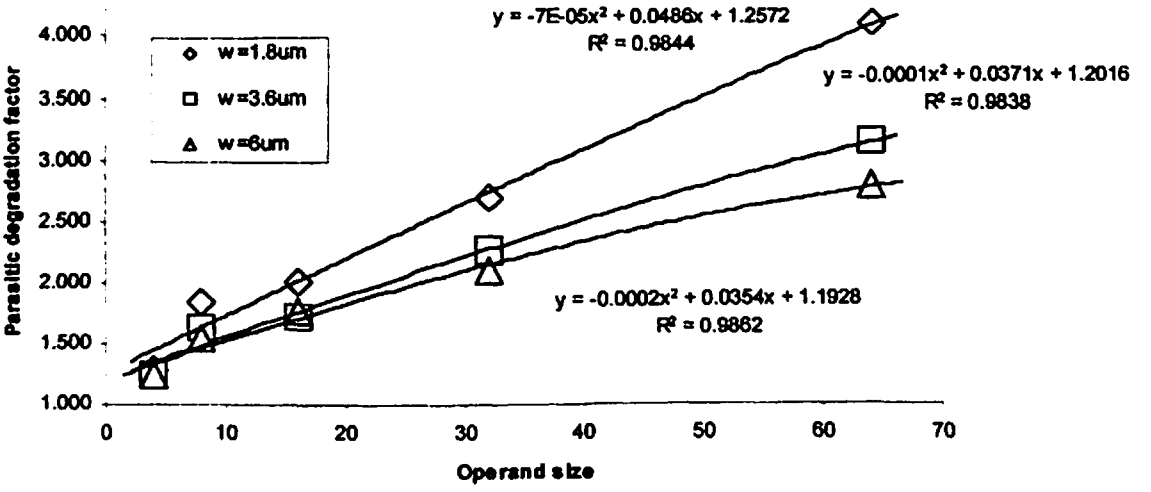


Fig. 7.3. b Parasitic degradation versus operand size of Carry select adder Domino CMOS logic

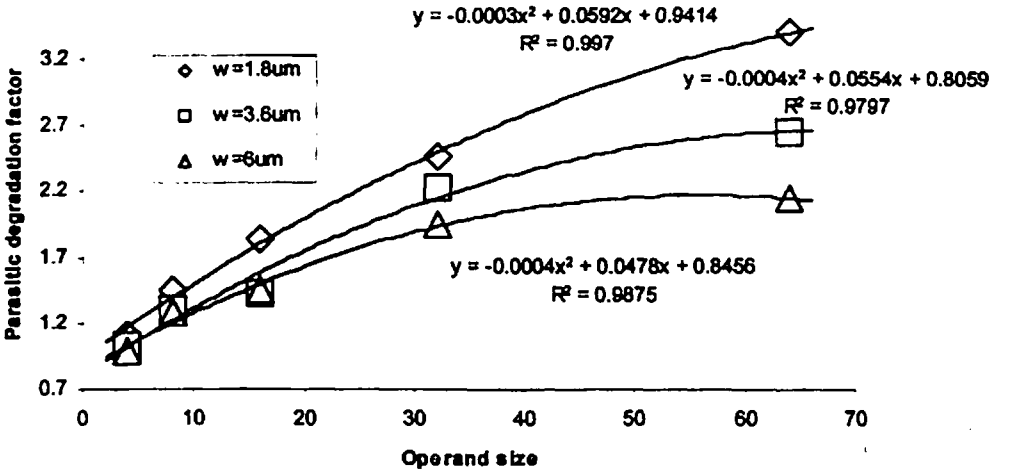


Fig. 7.3. c Parasitic degradation versus operand size of Carry select adder Complementary pass transistor logic

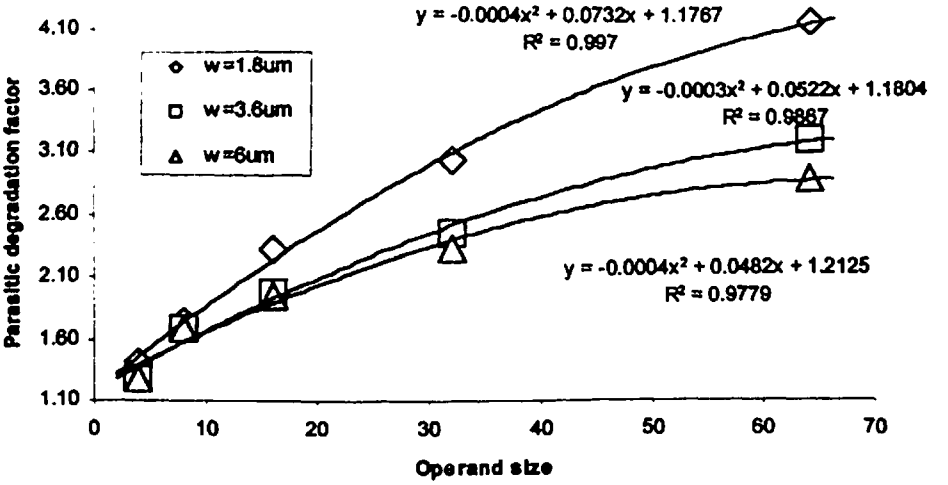


Fig. 7.3. d Parasitic degradation versus operand size of Carry select adder Dual pass transistor logic

7.6 RESULTS OF CONDITIONAL SUM ADDER

The variation of parasitic degradation factor (α) with operand size of conditional sum adder designs is shown in Figs. 7.4.

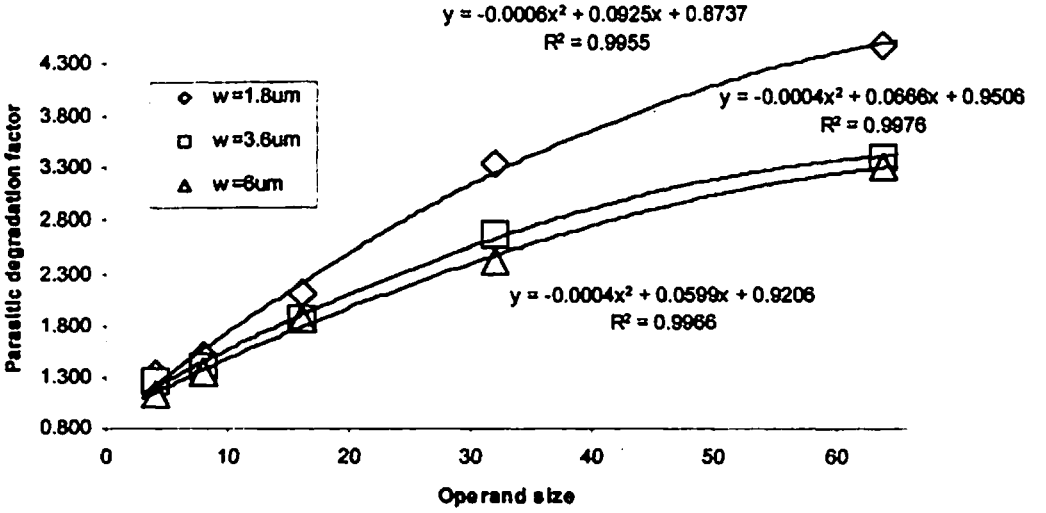


Fig. 7.4. a Parasitic degradation versus operand size for Conditional sum adder Fully Static CMOS logic

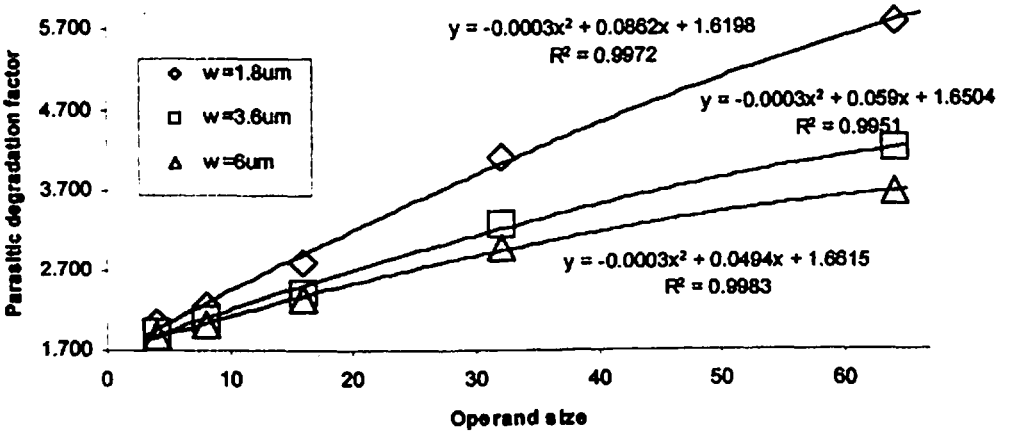


Fig. 7.4. b Parasitic degradation versus operand size for Conditional sum adder Domino CMOS logic

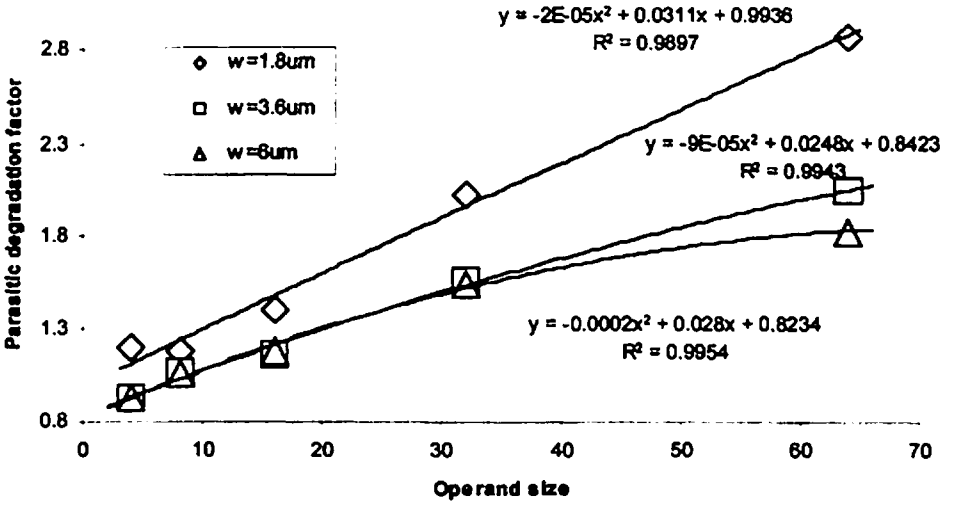


Fig. 7.4. c Parasitic degradation versus operand size for Conditional sum adder Complementary pass transistor logic

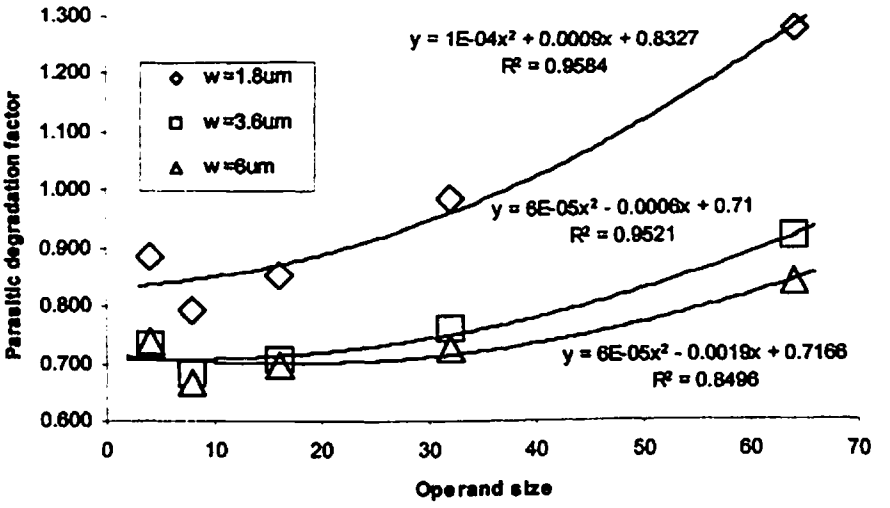


Fig. 7.4. d Parasitic degradation versus operand size for Conditional sum adder Dual pass transistor logic

7.7 RESULTS OF CARRY LOOK-AHEAD ADDER

The variation of parasitic degradation factor (α) with operand size of carry look-ahead adder designs is shown in Figs. 7.5.

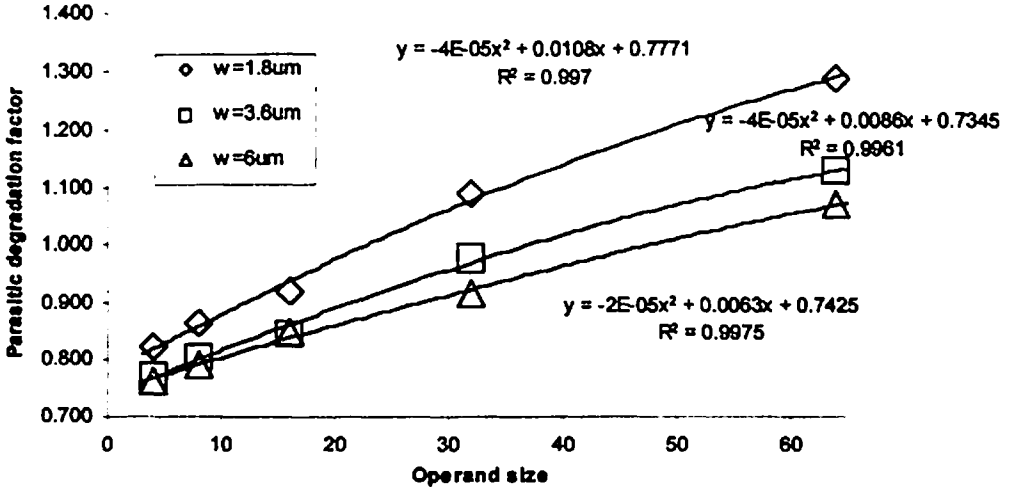


Fig. 7.5. a Parasitic degradation versus operand size for Carry look-ahead adder Fully static CMOS logic

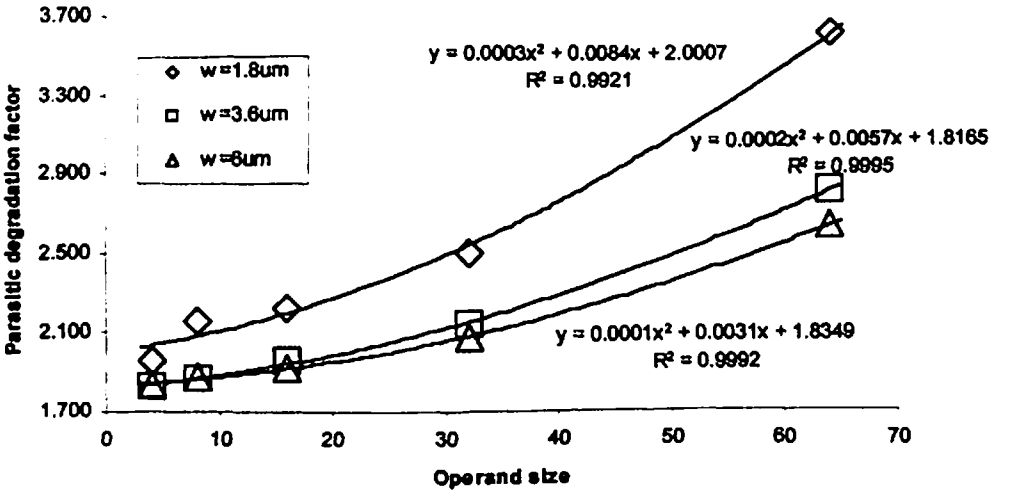


Fig. 7.5. b Parasitic degradation versus operand size for Carry look-ahead adder Domino CMOS logic

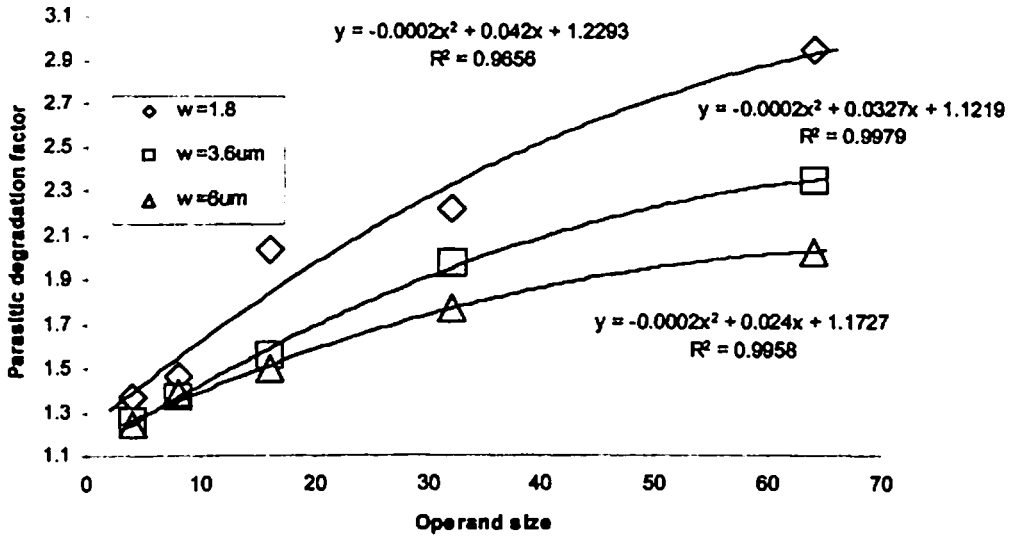


Fig. 7.5. c Parasitic degradation versus operand size for Carry look-ahead adder Complementary pass transistor logic

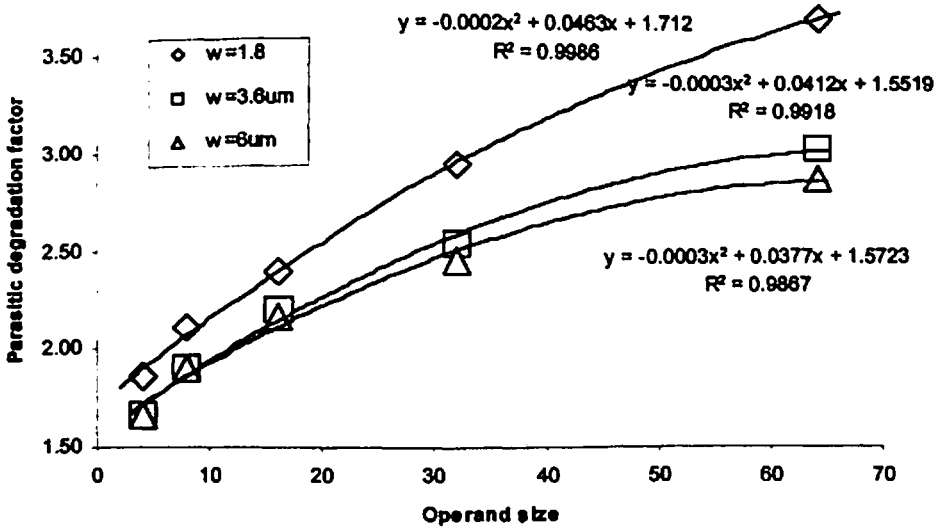


Fig. 7.5. d Parasitic degradation versus operand size for Carry look-ahead adder Dual pass transistor logic

CHAPTER 8

SYNTHESIS OF OPTIMAL ADDERS

Designing adders for high-speed, small silicon area and low energy consumption is a significant goal. This is not a straightforward task as the performance of an adder varies with the choice of architecture, logic design style, transistor size and operand size. Different combinations of these choices lead to many different adder designs. This makes the selection of an optimum adder that satisfies the requirements of a given application situation very difficult and time consuming- as this would involve extensive performance analysis of all the designs. Here, a design-advisor tool can help a designer to simplify the task. Therefore models are needed which can estimate the worst-case propagation delay and energy consumption per addition for different adder designs. A design-advisor tool can use these models to predict an optimum adder design for a given requirement.

With this aim, we have developed models for estimating worst-case propagation delay and energy consumption per addition for different adder designs. The models- the delay model and the energy model have been discussed in chapter 7.

In the present work we have generated the propagation delay and energy consumption values for different adder designs for different operand sizes 4, 8, 16, 32, and 64-bits using SPICE. The worst-case propagation delay and energy consumption per addition for other operand sizes have been predicted using the developed models. The predicted values have been compared with the values obtained by actually creating those designs and simulating them using SPICE. Besides, the propagation delay and energy consumption values for a specified adder design have also been obtained from the quadratic fits of propagation delay versus operand size and energy consumption versus operand size data shown in chapter 6 for the purpose of comparison.

The programs listed in ***Appendices (A1, A2, A3)*** estimate the worst-case propagation delay and energy consumption per addition using equations obtained from quadratic fits. A sample output of this program is listed in ***Appendix A6***. ***Appendices (A4, A5)*** list the programs that calculate the worst-case propagation delay and energy consumption per

addition using the models developed. Tables T27 (a, and b) list the worst-case propagation delay values obtained from: (1) delay model, (2) through quadratic fits to available data points (3) actual values obtained by creating the adder designs and simulating their netlists using SPICE. Results show a close match between the values.

TABLE T27 a: WORST-CASE PROPAGATION DELAY OF DIFFERENT ADDERS OBTAINED THROUGH ACTUAL SIMULATION, MODELING, AND QUADRATIC FITS TO AVAILABLE DATA POINTS

Logic design style	Operand size	Adder architecture	(w/l)	Propagation delay (ns) (Simulation)	Propagation delay (ns) (Modeling)	Propagation delay (ns) (Curve fitting)
Fully static logic design style	12	Conditional sum	1.5	9.94	9.21	9.50
			3	6.77	6.71	7.04
			5	5.78	5.58	5.87
	20	Ripple carry	1.5	23.90	24.51	24.80
			3	17.10	17.71	17.50
			5	14.10	14.54	14.90
	24	Carry select	1.5	15.10	15.77	16.40
			3	10.70	11.08	11.40
			5	8.28	8.78	90.6
	28	Carry skip	1.5	16.60	17.05	17.30
			3	11.90	11.88	12.10
			5	9.96	9.65	10.60
	40	Ripple carry	1.5	58.45	62.47	62.44
			3	40.11	42.42	41.46
			5	32.56	34.10	34.54
Domino CMOS logic design style	20	Ripple carry	1.5	40.70	41.07	41.10
			3	28.40	29.28	29.50
			5	23.70	24.17	24.40
	20	Carry select	1.5	20.69	19.21	19.15
			3	13.91	13.68	13.68
			5	12.09	11.22	11.38
	48	Carry skip	1.5	47.80	47.24	48.50
			3	32.00	38.90	32.50
			5	25.10	24.92	25.60
	56	Carry select	1.5	70.30	69.74	69.20
			3	44.60	45.31	44.30
			5	34.40	33.37	33.80

TABLE T27 b: WORST-CASE PROPAGATION DELAY OF DIFFERENT ADDERS OBTAINED THROUGH ACTUAL SIMULATION, MODELING, AND QUADRATIC FITS TO AVAILABLE DATA POINTS

Logic design style	Operand size	Adder architecture	(w/l)	Propagation delay (ns) (Simulation)	Propagation delay (ns) (Modeling)	Propagation delay (ns) (Curve fitting)
CPL	12	Ripple carry	1.5	12.20	11.85	12.30
			3	8.58	8.57	8.40
			5	7.49	7.25	7.48
	20	Conditional sum	1.5	9.86	9.77	9.96
			3	6.53	6.52	6.53
			5	5.57	5.59	5.87
	24	Carry skip	1.5	20.10	20.74	20.60
			3	14.10	13.79	13.90
			5	12.00	11.54	11.70
	52	Carry select	1.5	42.20	44.25	42.60
			3	26.90	29.59	26.30
			5	21.30	21.90	20.90
DPL	12	Ripple carry	1.5	20.14	19.05	19.64
			3	14.11	14.14	14.04
			5	12.26	11.93	12.18
	20	Conditional sum	1.5	8.80	9.05	9.01
			3	5.85	6.04	6.00
			5	5.00	5.04	5.02
	28	Carry skip	1.5	30.45	32.56	32.69
			3	21.35	21.11	21.35
			5	18.47	18.52	18.76
	40	Carry select	1.5	39.90	40.28	38.44
			3	25.84	26.69	25.16
			5	20.74	20.52	20.21

Tables T28 (a and b), list the energy consumption per addition for the worst-case propagation delay input combination for different adders obtained through (1) SPICE simulation, and (2) through quadratic fits to available data points. The results show a close match between the two energy consumption values.

TABLE T28 a : ENERGY CONSUMPTION PER ADDITION OF DIFFERENT ADDERS OBTAINED THROUGH SPICE SIMULATION, AND QUADRATIC FITS TO AVAILABLE DATA POINTS

Logic design style	Operand size	Adder architecture	*(w/l)*	Energy consumption $J \times 10^{-10}$ (Simulation)	Energy consumption $J \times 10^{-10}$ (Curve fitting)
Fully static logic design style	12	Condit-ional sum	1.5	3.11	3.18
			3	4.57	4.94
			5	6.26	6.70
	20	Ripple carry	1.5	2.20	2.21
			3	2.98	3.04
			5	4.06	4.04
	24	Carry select	1.5	5.01	5.08
			3	6.97	7.05
			5	8.91	9.82
	28	Carry skip	1.5	3.86	4.08
			3	5.38	5.42
			5	7.33	7.45
	40	Ripple carry	1.5	5.67	5.81
			3	7.36	7.37
			5	9.59	9.58
Domino CMOS logic design style	20	Ripple carry	1.5	6.24	6.26
			3	9.14	9.40
			5	12.94	13.19
	20	Carry select	1.5	11.13	10.99
			3	16.77	17.02
			5	24.68	24.42
	48	Carry skip	1.5	20.90	20.77
			3	28.89	29.02
			5	39.82	39.80
	56	Carry select	1.5	40.59	40.37
			3	58.01	57.76
			5	83.70	82.04

TABLE T28 b: ENERGY CONSUMPTION PER ADDITION OF DIFFERENT ADDERS OBTAINED THROUGH SPICE SIMULATION, QUADRATIC FITS TO AVAILABLE DATA POINTS

Logic design style	Operand size	Adder architecture	*(w/l)*	Energy consumption $J\times10^{-10}$ (Simulation)	Energy consumption $J\times10^{-10}$ (Curve fitting)
CPL	12	Ripple carry	1.5	1.94	1.88
			3	2.72	2.63
			5	3.91	3.82
	20	Condit-ional sum	1.5	4.24	4.28
			3	5.83	5.83
			5	8.48	8.26
	24	Carry skip	1.5	6.51	6.30
			3	9.07	8.85
			5	12.82	12.62
	52	Carry select	1.5	16.68	18.83
			3	21.36	24.75
			5	34.83	32.13
DPL	12	Ripple carry	1.5	2.36	2.34
			3	3.33	3.63
			5	4.78	4.75
	20	Condit-ional sum	1.5	5.14	5.26
			3	6.71	6.70
			5	9.29	9.19
	28	Carry skip	1.5	8.36	8.48
			3	11.61	11.54
			5	16.35	16.07
	40	Carry select	1.5	16.79	19.71
			3	23.40	27.06
			5	32.33	40.88

Table T29 lists energy consumption per addition obtained using energy model for different 4-bit adders.

TABLE T29: ENERGY CONSUMPTION PER ADDITION OF DIFFERENT 4-BIT ADDERS OBTAINED USING ENERGY MODEL AND SPICE SIMULATION

Adder architecture	(w/l)	Energy consumption of 4-bit Adder $J\times10^{-10}$							
		Fully static CMOS logic		Domino CMOS logic		CPL		DPL	
		energy model	simulation	energy model	simulation	energy model	simulation	energy model	simulation
Ripple carry	1.5	0.282	0.276	0.687	1.042	0.547	0.563	0.620	0.633
	3.0	0.419	0.408	1.307	1.598	0.861	0.826	0.982	0.946
	5.0	0.655	0.595	2.032	2.407	1.265	1.239	1.670	1.363
Carry skip	1.5	0.295	0.317	0.703	1.121	0.682	0.651	0.449	0.737
	3.0	0.423	0.452	1.351	1.796	0.978	0.903	0.958	1.068
	5.0	0.667	0.679	1.979	2.582	1.448	1.381	1.672	1.518
Carry select	1.5	0.554	0.612	2.044	2.407	1.071	1.200	0.799	1.427
	3.0	0.839	0.944	3.887	3.844	1.546	1.805	1.858	2.111
	5.0	1.307	1.385	6.029	5.642	2.321	2.711	3.274	2.959
Conditional sum	1.5	0.889	0.902	1.844	1.423	0.852	0.803	0.731	1.082
	3.0	1.329	1.356	3.623	2.197	1.189	1.166	1.498	1.399
	5.0	2.073	1.987	5.609	3.232	1.753	2.711	2.594	1.943
Carry look-ahead	1.5	0.369	0.376	1.173	1.146	0.820	0.781	0.673	0.853
	3.0	0.545	0.608	2.280	1.812	1.219	1.182	1.472	1.242
	5.0	0.869	0.869	3.379	2.607	1.837	1.710	2.570	1.786

The table clearly shows that the energy model correctly grades the designs for their energy consumptions - even though the estimates of energy consumption it provides are slightly different than those provided by simulation. The difference in values of energy consumption by the model is due to the fact that in our modeling we have considered an average value of weight factor for all the gates used in the design using a particular logic design style that accounts for energy consumption in switching of nodes internal to a gate.

CHAPTER 9

CONCLUSION

The objective of this thesis is to explore the design space of VLSI adders of different bit-widths in terms of their architectures, logic design styles, transistor sizes, and layout design styles with a view to understand the contributions to speed, energy consumption and area that come from each of these factors so that a design-advisor tool can be built to automatically select an optimal adder configuration based on the constraints of speed and energy consumption per addition. First, adder designs are created, and laid out. The pre and post layout netlists extracted from these designs have been simulated using SPICE. We have chosen five adder architectures that are frequently used in arithmetic circuits i.e. ripple carry adder, carry skip adder, carry select adder, conditional sum adder, and carry look-ahead adder for operand sizes of 4, 8, 16, 32, and 64 bits. We have used four circuit design styles: fully static CMOS, domino CMOS, dual pass transistor logic and complementary pass transistor logic and gates with 2 to 4 inputs, for designing adders. Also, each adder architecture has been designed using three different transistor sizes- *(w/l)* =1.5, 3, and 5 for a particular logic design style.

Simulation results show that -

- Energy consumption per addition increases and worst-case propagation delay decreases with increase in transistor size.
- Fully static CMOS logic design style is the most energy efficient design style for all architectures except for conditional sum adder architecture, which consumes less energy per addition for CPL design style.
- Ripple carry adder architecture consumes least energy per addition in comparison to other architectures for all logic design styles.
- For 4-bit, 8-bit and 16-bit operand sizes, conditional sum adder architecture designed in CPL design style results in highest-speed addition. For 32-bit and 64-bit operand sizes, carry look-ahead adder architecture designed in fully static CMOS logic gives the fastest addition.
- Carry look-ahead adder architecture is faster than other architectures for fully static CMOS, and domino CMOS logic design styles.

- Performance of adder architectures varies with operand size. For 4-bit operand size, ripple carry adder architecture has the least energy-delay product (EDP) irrespective of logic design style and transistor size except in domino CMOS logic where carry look-ahead adder shows nearly the same value but with higher transistor count. For 16-bit and larger operand sizes - carry look-ahead adder architecture has the least EDP for fully static CMOS and domino CMOS logic design styles whereas conditional sum adder architecture has the smallest EDP for pass transistor logic design styles. For 8-bit operand size, carry look-ahead adder architecture for fully static CMOS and conditional sum adder architecture for other logic design styles have the least EDP.
- Ripple carry adder architecture has the smallest transistor count and core area for all logic design styles. CPL design style based adders have the smallest transistor count for all architectures. Domino CMOS logic design style has a higher transistor count than fully static CMOS logic design style because standard cells having only 2 to 4 inputs have been used in designs.

From the above results, it is observed that ripple carry adder architecture designed using fully static logic design style is most suitable for low-energy applications. Carry look-ahead architecture is most suitable for high-speed operation for larger operand sizes. In pass transistor logic; CPL is better than DPL, both, in terms of energy consumption and worst-case propagation delay due to less parasitic capacitances.

Energy-delay product (EDP) has been used as the basis for choosing an optimal adder design. The EDP has been found to vary with logic design style, and adder architecture. For fully static CMOS, and domino CMOS logic design styles, carry look-ahead adder architecture has the lowest EDP, and for pass transistor logic design styles, conditional sum architecture has the lowest EDP followed by carry look-ahead architecture. Tables T50 to T54 of ***Appendix B5*** list the EDPs of different adders. The EDPs obtained from simulation results of adders exhibit a smaller value at transistor width equal to 3.6μm (0.6 to 0.99 times the EDP of an adder obtained at transistor width equal to 1.8μm). This is because a moderate increase in (w/l) of the transistors used causes a more significant reduction in delay than the increase in energy consumed. This trend is observed in all adder designs for operand size of 64-bit (in some cases for smaller operand sizes also).

Further increase in channel-width leads to an increase in EDP, as the reduction in delay decreases whereas the energy consumption keeps on increasing. Thus, the EDP of an adder shows a minimum at the channel width that is twice the minimum allowed channel width value.

We have studied the effect of layout methodology on EDP by comparing standard-cell based design and full-custom design of 64-bit ripple carry adder and carry look-ahead adder in fully static CMOS logic design style. Full custom layouts of the full-adder cell, A-cell and B-cell have been drawn which were then used in creating designs of 64-bit ripple carry and carry look-ahead adders. The EDP (shown in ***Appendix B7)*** of full custom designs is found to be much less than the EDP of standard cell based designs due to significant decrease in parasitic capacitances. But, in full custom designs too, the EDP has been found to be minimum for $(w/l)=3$ and increases with further increase in (w/l). Hence, the adders designed with full-custom layout methodology also give an optimum value for channel width (w) twice the minimum allowed value.

In the present work, an analysis of adders has been done using energy consumption per addition for an input combination that corresponds to the worst-case propagation delay. Since the power dissipation in combinational circuits is pattern dependent, we have also, obtained the average energy consumption per addition for a set of randomly generated input vectors (E_{random}) for 4-bit adder designs. Results show that the energy consumption per addition for a set of randomly applied input vectors is nearly a constant multiple of the energy consumption per addition for the worst-case propagation delay input pattern ($E_{worst\text{-}case}$) for a given logic design style. The ratio M defined as

$$M = \frac{E_{random}}{E_{worst-case}}$$

has been found to be nearly constant for different adder architectures designed using a particular logic design style for different channel-widths (shown in ***Appendix B7)***. Hence, the grading of different adder architectures in terms of their energy consumption obtained for worst-case delay input pattern applies to the case of randomly applied input vectors also.

For modeling the worst-case propagation delay, the approach of identifying the critical delay paths of adders is used. The product of number of gates in critical delay path, time constant of the technology (τ), average fan-out per gate in critical delay path (f), and the parasitic degradation factor (α) has been used to estimate the worst-case propagation delay. The time constant (τ) has been obtained for different transistor sizes without including the circuit parasitics using SPICE. The parasitic degradation factor (α) of all adder designs is obtained empirically by equating the delay value obtained from simulation to this product. The variation of (α) with operand size is obtained for all adders designed using different circuit design styles and a second order polynomial is fitted to the data. Thus equations linking (α) to operand size are obtained which have been used for determining (α) of an adder of any size between 4 to 64-bit. This value is then used in the product for estimating the worst-case propagation delay.

For modeling the energy consumption per addition, product of average energy consumption ($\bar{E}$) in driving $1C_g$ load, average number of nodes (n_s) undergoing transitions for all possible transitions from an input pattern to another input pattern, glitch factor (g), average weight factor (Ψ) per gate, average load capacitance at the output of a gate (χ), and the parasitic degradation factor (α) is taken. Energy consumption ($\bar{E}$) in driving one gate capacitance has been obtained through SPICE simulation without including circuit parasitics and (χ) has been calculated from adder schematics. The value of (g) is taken as 1.5 for static logic and 1 for dynamic logic to take care of spurious transitions. Average weight factor (Ψ) has been estimated by taking average of weight factors of different gates used in the design for a particular logic design style.

Worst-case propagation delay values have been obtained from: (1) delay model, (2) through quadratic fits to available data points and, (3) actual SPICE simulation. The results listed in tables T27 (a), and (b) show a close match between the values.

Energy consumption per addition for worst-case propagation delay input combination has been obtained through SPICE simulation, and (2) through quadratic fits to available data points. The results listed in tables T28 (a), and (b) show a close match between the

values. Table T29 shows energy consumption per addition obtained using energy model for 4-bit adders. Energy model correctly grades the adder designs for their energy consumption.

Towards the end, we have developed tools, which are listed in ***Appendix A1, A2, A3, A4*** and ***A5.*** A sample output of these programs is listed in ***Appendix A6***. These tools can be used to select an optimum adder design for a given application.

Future directions

In the present work an effort has been made to analyze different adder designs with a view to select the optimum adder for a given application situation. The study can be further augmented by including the techniques of transistor sizing in the critical delay path and transistor-reordering to increase the speed of the circuit. Different adder architectures can be combined to develop hybrid adder designs. The selection of different adders can be based on the performance analysis done in the present work. Energy model developed can be improved further by considering weighted average factor for different gates used in the design. The developed models can be tested for adder designs of different channel length technologies with a view to validate their wide applicability.

REFERENCES

[1]. Gilchrist, B., Pomerene, J. J., and Wong, S. Y., "Fast Carry Logic for Digital Computers," IRE Trans. on Electronic Computers, vol. EC-4, no. 4, pp.133-136, Dec. 1955.

[2]. Weinberger, A., and J. L Smith, J. L., "A One Microsecond Adder Using One Megacycle Circuitry," IRE Trans. on Electronic Computers, vol. EC-5, no. 2, pp.67-73, Jun. 1956.

[3]. Morgan, C. P., and Jarvis, D. B., "Transistor logic using current switching and routing techniques and its application to a fast-carry propagation adder," Proc. IEE, vol. 106, pp. 467-468, Sep. 1959.

[4]. Lehman, M., and Burla, N., "A note on the simultaneous carry generation system for High Speed adders," IRE Trans. on Electronic Computers, EC-9, no. 4, pp.510, Dec. 1960.

[5]. Sklansky, J., "An Evaluation of Several Two-Summand Binary Adders," IRE Trans. on Electronic Computers, vol. EC-9, no. 2, pp.213-226, Jun. 1960. (a)

[6]. Sklansky, J., "Conditional-Sum Addition Logic," IRE Transactions on Electronic Computers, vol. EC-9, no. 2, pp.226-231, Jun. 1960. (b)

[7]. Lehman, M., and Burla, N., "Skip Techniques for High-Speed Carry-Propagation in Binary Arithmetic Units," IRE Trans. on Electronic Computers, vol. EC-10, no. 4, pp.691-698, Dec. 1961.

[8]. Bedrij, O. J., "Carry select adder," IRE Transactions on Electronic Computers, vol. EC-11, no. 3, pp. 340-346, Jun. 1962.

[9]. Majerski, S., "On Determination of Optimal Distributions of Carry-Skips in Adders," IEEE Trans. on Electronic Computers, vol. EC-16, no. 1, pp. 45-58, Feb. 1967.

[10]. Gosling, J. B., "Review of High Speed Addition Techniques," Proc. IEE, vol.118, pp. 29-35, Jan. 1971.

[11]. Lee, C. M., and Soukup, H., "An algorithm for CMOS timing and area optimization," IEEE J. Solid- State Circuits, vol. SC-19, no. 5, pp. 781-787, Oct. 1984.

[12]. Veendrick, H. J. M., "Short-circuit dissipation of static CMOS circuitry and its impact on the Design of Buffer circuits," IEEE J. Solid-State Circuits, vol. SC-19, no. 4, pp. 468-473, Aug. 1984.

[13]. Barnes, E. R., and Oklobdzija, V. G., "New Schemes for VLSI Implementation of Fast ALU," IBM Tech. Discl. Bull, vol.28, no.3, pp. 1277-1282, Aug.1985.

[14]. Glasser, L. A., and Dobberphul, D. W., "The Design and Analysis of VLSI circuits," Addison-Wesley, 1985.

[15]. Auvergne, D., "Delay-time evaluation in ED MOS logic LSI," IEEE J. Solid-State Circuits, vol. SC-21, no.2, pp.337-343, Apr.1986.

[16]. Tsao, D., and Chen, C. F., "A fast Timing Simulator for digital CMOS circuits," IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems, vol. CAD-5, no.4, pp. 536-540, Oct. 1986.

[17]. Auvergne, D., and Deschacht, D., and Roberts, M., "Explicit Formulation of delays in CMOS VLSI," Electron. Lett., vol. 23, no.14, p.71, Jul. 1987.

[18]. Guyot, A., Hochet, B., and Muller, J. M., "A Way to Build Efficient Carry-Skip Adders," IEEE Trans. On Computers, vol. C-36, no.10, pp.1144-1151, Oct. 1987.

[19]. Hedenstierna, N., and Jeppson, K. O., "CMOS circuit speed and buffer optimization," IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems. vol. CAD-6, no. 2, pp. 270-281, Mar.1987.

[20]. Chu, K. M, and Pulfrey, D. L., "A Comparison of CMOS Circuit Techniques: Differential Cascode Voltage Switch Logic Versus Conventional Logic," IEEE J. Solid- State Circuits, vol. SC-22, no.4, pp.528-532, Aug.1987.

[21]. Antognetti, P., and Massobrio, G., "Semiconductor Device Modeling with SPICE," McGraw Hill International Editions, 1988.

[22]. Deschacht, D., Robert, M., and Auvergne, D., "Explicit Formulation of Delays in CMOS data Paths," IEEE J. Solid-State Circuits, vol.23, no. 5, pp. 1257-1264, Oct. 1988.

[23]. Shoji, M., "CMOS Digital Circuit Technology," Prentice Hall, 1988.

[24]. Hwang, I., and Fisher, A., "Ultrafast compact 32-b CMOS adder in Multiple-output Domino logic," IEEE.J. Solid-State Circuits, vol.24, no. 2, pp.358-369, Jun. 1989.

[25]. Auvergne, D., Azemard, N., Deschacht, D., and Robert, M., "Input waveform slope effects in CMOS delays," IEEE J. Solid-State Circuits, vol.25, no. 6, pp.1588-1590, Dec. 1990.

[26]. Chan, P. K., and Schlag, M. D. F., "Analysis and Design of CMOS Manchester Adders with Variable Carry-Skip," IRE Trans. on Computers, vol.39, no.8, pp. 983-992, Aug. 1990.

[27]. Chowdhury, S., and Barkatullah, J. S., "Estimation of maximum currents in MOS IC logic circuits," IEEE Trans. on Computer Aided Design of Integrated Circuits and Systems, vol. 9, no. 6, pp. 642-654, Jun.1990.

[28]. Chandrakasan, A., Sheng, S., and Broderson, R., "Low-power CMOS digital design," IEEE.J. Solid-State Circuits, vol.27, no.4, pp.473-484, Apr. 1992.

[29]. Ghosh, A., Devdas, S., Keutzer, K., and White, J., "Estimation of Average Switching Activity in Combinational and Sequential Circuits," IEEE/ACM Design Automation Conference, pp.253-259, Jun. 1992.

[30]. Zhuang, N., and Wu, H., "A new design of the CMOS full adder", IEEE J. Solid-State Circuits, vol.27, no. 5, pp.840-844, May. 1992.

[31]. Burch, R., and Najm, F. N., "A Monte Carlo Approach for Power Estimation," IEEE Trans. on Very Large Scale Integration Systems, vol.1, no.1, pp.63-71, March 1993.

[32]. Callaway, T. K., and Swartzlander, E. E., "Estimating the Power consumption of CMOS adders," Proc. 11th Symp. On Comp. Arithmetic, pp. 210-219, Jun. 1993.

[33]. Najm, F. N., "Transition density: A New Measure of Activity in Digital Circuits," IEEE Trans. On Computer Aided Design of Integrated Circuits and Systems, vol.12, no. 2, pp.310-323, Feb.1993.

[34]. Weste, N., and Eshraghian, K., "Principles of CMOS VLSI Design," Addison-Wesley, 1993.

[35]. Abu-Khater, I. S., Yan, R. H., Bellaouar, A., and Elmasry, M.I., "A 1-v Low Power high-performance 32-b Conditional Sum Adder," IEEE Symposium on Low-Power Electronics, Tech. Dig., pp. 66-67, Oct. 1994.

[36]. Nagendra, C., Owens, R. M., and Irwin, M. J., "Power-delay characteristics of CMOS adders," IEEE. Trans. Very Large Scale Integration Systems, vol. 2, no. 3, pp. 377-381,Sep. 1994.

[37]. Najm F. N., "A Survey of Power Estimation Techniques in VLSI Circuits," IEEE Trans. Very Large Scale Integration Systems, vol.2, no.4, pp. 446-454, Dec. 1994.

[38]. Wang, J. M., Fang, S. C., and Feng, W. S., "New efficient designs for XOR and XNOR functions on the transistor level," IEEE J. Solid-State Circuits, vol.29, no. 7, pp.780-786, Jul. 1994.

[39]. Bellaouar, A., and Elmasry, M. I., "Low-Power Digital VLSI Design Circuit and Systems," Kluwer Academic Publisher, 1995.

[40]. Ko, U., and Balsara, P.T., "Low power Design Techniques for High performance CMOS Adders," IEEE Trans. Very Large Scale Integration Systems, vol.3, no.2, pp. 327-333, Jun. 1995. (a)

[41]. Ko, U., Balsara, P.T., "Short-Circuit Power Driven Gate Sizing Technique for Reducing Power Dissipation," IEEE Trans. Very Large Scale Integration Systems, vol.3, no.3, pp. 450-454, Sept. 1995. (b)

[42]. Powers, R. A., "Batteries for Low Power Electronics," Proc. Of IEEE, vol.83, no.4, pp. 687-693, Apr. 1995.

[43]. Hennessy, J. L., and Patterson, D. A., "Computer Architecture- A Quantitative Approach," Morgan Kaufmann Publishers, Inc., 1996.

[44]. Zimmermann. R., and Fichtner, F., "Low-Power logic styles: CMOS versus pass-transistor logic," IEEE J. Solid-State Circuits, vol.32, no. 7, pp.1079-1090, Jul. 1997.

[45]. Kang, S. M., and Leblebici, Y., "CMOS Digital Integrated Circuits Analysis and Design," McGraw Hill International Editions, 1999.

[46]. Lim, J., Kim, D., and Chae, S., "A 16-bit carry look-ahead adder using Reversible energy recovery logic for ultra low-energy systems," IEEE J. Solid-State Circuits, vol.34, no. 6, pp.898-903, Jun. 1999.

[47]. Sinencio, E. E., and Andreou, A. G., "Low-Voltage/Low-Power Integrated Circuits and Systems, Low-voltage mixed signal circuits," IEEE Press, 1999.

[48]. Callaway, T. K., and Swartzlander, E. E., "The Power Consumption of CMOS Adders and Multipliers," in 'Low-Power CMOS Design', edited by Chandrakasan, A., and Brodersen, R., pp. 218-224, IEEE Press, Standard Publishers Distributors, 2000.

[49]. Martin, K., "Digital Integrated Circuit Design', Oxford University Press, 2000.

[50]. Roy, K., and Prasad, S., "Low Power CMOS VLSI Circuit Design," John Wiley and Sons, Inc., 2000.

[51]. Hafed, M., Oulmane, M., and Rumin, N. C., "Delay and current estimation in a CMOS inverter with an RC load," IEEE Trans. on Computer Aided Design of Integrated Circuits and Systems, vol. 20, no.1, pp. 80-89, Jan.2001.

[52]. Wu, Q., Qiu, Q., and Pedram, M., "Estimation of Peak Power Dissipation in VLSI Circuits using the Limiting Distributions of Extreme Order Statistics," IEEE Trans. on Computer Aided Design of Integrated Circuits and Systems, vol. 20, no. 8, pp. 942-955, Aug.2001.

[53]. Evmorfopoulos, N. E., Stamoulis, G. I., Avaritsiotis, J. N., "A Monte Carlo Approach for Maximum Power Estimation Based on Extreme Value Theory," IEEE Trans. on Computer Aided Design of Integrated Circuits and Systems, vol. 21, no. 4, pp. 415-431, April.2002.

[54]. Shams, A. M., Darwish, T. K., and Bayoumi, M. A., "Performance analysis of low power 1-bit CMOS full adder cells," IEEE Trans. Very Large Scale Integration Systems, vol.10, no.1, pp. 20-29, Feb. 2002.

[55]. Fahim, A. M., Elmasry, M. I., "Low power high-performance arithmetic circuits and architectures," IEEE J. Solid-State Circuits, vol.37, no. 1, pp.90-94, Jan. 2002.

BIBLIOGRAPHY

i. Burks, H., Goldstein, H., and Von-Neumann. J., "Preliminary Discussion of the Logic Design of Electronic Computing Instrument, Instit. Adv. Study, 1946.

ii. Muller, D. E., "Complexity in Electronic Switching Circuits," IRE Trans. on Electronic Computers, vol. EC-5, no. 1, pp.15-19, 1956.

iii. Bell, C. G., "Computer Structures: Reading and Examples," McGraw Hill, 1971.

iv. Parker, K. P., "Probabilistic treatment of general combinational networks," IEEE Trans. on Computers, vol. C-24, no. 6, pp. 668-670, 1975.

v. Blaauw, G. A., "Digital System Implementation," Prentice-Hall, 1976.

vi. Mead, C., and Conway, L., "Introduction to VLSI Systems," Addison-Wesley, 1980.

vii. Brent, R.P., and Kung, H. T., "A Regular layout for parallel adders," IEEE Trans. on Comp., vol. C-31, no. 3, pp. 260-264, 1982.

viii. Kramback, R. H., Lee, C. M., and Law, H., "High Speed Compact Circuits with CMOS," IEEE.J. Solid-State Circuits, vol. SC-17, no. 6, pp. 614-619, 1982.

ix. Pomper, M., Beifuss, W., Horninger, K., and Kadchte, W., "A 32-bit execution unit in an advanced NMOS technology," IEEE.J. Solid-State Circuits, vol. SC-17, no-6 pp. 533-538, 1982.

x. Trivedi, K. S., "Probanility & Statistics with Reliability, Queuing, and Computer Science Applications," Prentice-Hall, Inc., 1982.

xi. Hodges, D. A., and Jackson, H. G., "Analysis and Design of Digital Integrated Circuits," McGraw Hill, 1983.

xii. Cavanagh, J. J. F., "Digital Computer Arithmetic, Design, and Implementation," McGraw Hill, 1984.

xiii. Scott, N. R., "Computer Systems and Arithmetic," Cliffs, Prentice Hall, 1985.

xiv. Annaratone, M., "Digital CMOS Circuit Design", Kluwer, 1986

xv. Kang, S. M., "Accurate simulation of Power Dissipation in VLSI Circuits," IEEE.J. Solid-State Circuits, vol. SC-21, no. 10, pp. 889-891, 1986.

xvi. Tsividis, Y. P., "Operation and Modeling of the MOS Transistor," McGraw Hill, 1987.

xvii. Hwang, K., "Computer Arithmetic: Principles, Architecture, and Design," John Wiley and Sons, 1979.

xviii. Bokaglu, H. B., "Circuits, Interconnections, and Packaging for VLSI," Addison-Wesley, 1990.

xix. Huizer, C. M., "Power dissipation analysis of CMOS VLSI circuits by means switch-level simulation," IEEE Europ. Solid-State Circuits Conf. pp. 61-64, France, 1990

xx. Eager, "Advances in rechargeable batteries pace portable computer growth," in Proc. Silicon Valley Person. Comput. Conf., pp. 693-697, 1991

xxi. Uyemura, J. P., "Circuit design for CMOS VLSI," Kluwer Academic Publishers, 1992.

xxii. Liu, D., and Svensson, C., "Trading Speed for Low Power by Choice of Supply and Threshold Voltages," IEEE J. of Solid-State Circuits, vol. SC-28, no. 1, pp. 10-17, 1993.

xxiii. Stone, H. S., "High-Performance Computer Architecture," Addison-Wesley Publishing Company, 1993.

xxiv. Suzuki, M., "A 1.5-ns 32-b CMOS ALU in Double Pass-transistor logic," IEEE J. Solid- State Circuits, vol. SC-28, no. 11, pp. 1145-1151, 1993.

xxv. Abu-Shama, E., and Bayoumi, M., "A new cell for low power adders," Proc. Int. Midwest Symp. Circuits and Syst., 1995.

xxvi. Tsui, C., Monterio, J., Pedram, M., Devdas, S., Despain, A. M., Lin, B., "Power Estimation Methods For Sequential Logic Circuits," IEEE Trans. of Very Large Scale Integration Systems, vol. 3, no. 3, pp. 406-416, 1995.

xxvii. Ding, C. S., Hsieh, C. T., Wu, Q., and Pedram, M., "Stratified Random Sampling for Power Estimation," IEEE/ACM International Conference on Computer Aided Design, pp. 576-582, 1996.

xxviii. Rabaey, J. M., "Digital Integrated Circuits," Prentice-Hall, 1996.

xxix. Rabaey, J. M., and Pedram, M., "Low Power Design Methodologies," Kluwer Academic, 1996.

xxx. Tanner Tools, L-Edit™ User's Manual version 6., Tanner Research, Inc., 1996.

xxxi. Tanner Tools, T-Spice Pro™ User's Manual version 4., Tanner Research, Inc., 1996.

xxxii. Smith, M. J. S., "Application Specific Integrated Circuits," Addison-Wesley, 1997.

xxxiii. Baker, R. J., Li, H., and Boyce, D. E., "CMOS Circuit Design, Layout, and simulation," IEEE Press, 1998.

xxxiv. Bernstein, K., "High-Speed CMOS Design Styles," Kluwer Academic Publishers, 1998.

xxxv. Walpole, R. E., Myres, R. H., and Myres, S. L., "Probability and statistics for Engineers and Scientists," Prentice-Hall International, Inc., 1998.

xxxvi. Chandrakasan, A., and Brodersen, R., "Low-Power CMOS Design," IEEE Press, Standard Publishers Distributors, First Indian edition, 2000.

xxxvii. Gupta, A., and Shekhra, C., "Optimal Adder Architectures for Fully Static and Complementary Pass Logic Designs," Proc. National Seminar on VLSI: Systems, Designs and Technology, pp. 140-145, IIT, Bombay, Dec. 2000.

xxxviii. Johnson, R. A., "Miller and Freund's Probability and Statistics for Engineers," Prentice-Hall of India Private limited, 2000.

xxxix. Uyemura, J. P., "Introduction to VLSI circuits and systems," John Wiley and Sons, Inc. 2002.

APPENDIX A1

This program calculates the interpolated values of worst-case propagation delay and energy consumption for a given adder design. Also, this program gives the adder designs for a given value of worst-case propagation delay and energy consumption.

```
clc;
clear all
diary prgresult.m
diary on
disp('**********************************************
**************************************** ')
disp(' ')
disp('Adder analysis program')
disp(' ')
disp('**********************************************
**************************************** ')
%Take input from user
disp(' ')
disp('Adder analysis options')
disp(' ')
disp('Analysis option values are:')
disp(' ')
disp('(1)Delay and energy consumption value for a given
adder')
disp(' architecture, logic design style, and channel width = 1')
disp(' ')
disp('(2)Name of architecture/s for a given delay and energy
value= 2')
disp(' ')
disp('**********************************************
**************************************** ')
op=input('Please input analysis option= ');
while op>2
disp('ERROR: Option value is out of the range. Please give
option value between 1 and 2')
disp(' ')
 d=input('Please input analysis option again= ');
disp(' ')
end;
clc;
disp(' ')
if op==1;
disp('**********************************************
**************************************** ')
disp(' ')
disp('Program for obtaining value of energy consumption and
propagation delay ')
disp(' delay for a given arhitecture,design style and channel
width')
disp(' ')
disp('**********************************************
**************************************** ')
%Take input from user
disp(' ')
disp('Please input size of adder from 1 to 64 bits in multiples of
four')
disp(' ')
x=input('Please input adder size (number of bits)= ');
disp(' ')
while x==0;
disp(' ')
 x=input('Please input adder size i.e(number of bits) from 1 to
64 again in multiples of 4 = ');
disp(' ')
end;
while x>64;
disp('ERROR: Adder size is out of the range. Please give adder
size from 1 to 64 in multiples of 4')
disp(' ')
x=input('Please input adder size i.e(number of bits) from 1 to
64 again in multiples of 4= ');
disp(' ')
while x==0;
disp('ERROR: Adder size is out of the range. Please give adder
size from 1 to 64 in multiples of 4')
disp(' ')
x=input('Please input adder size i.e(number of bits) from 1 to
64 again in multiples of 4= ');
end;
end;
clc;
disp('**********************************************
**************************************** ')
disp('Architecture options and their values are:')
disp(' ')
disp('(1)No specific adder architecture.Print result for all
architectures= 0')
disp('(2)Ripple carry adder architecture= 1')
disp('(3)Carry select adder architecture= 2')
disp('(4)Carry skip adder architecture= 3')
disp('(5)Carry look ahead adder architecture= 4')
disp('(6)Conditional sum adder architecture= 5')
disp(' ')
b=input('Please input architecture option= ');
while b>5
disp('ERROR: Option value is out of the range. Please give
option value between 0 and 5')
disp(' ')
b=input('Please input architecture option again= ');
disp(' ')
end
clc
disp('**********************************************
**************************************** ')
disp('Logic design style options and their values are:')
disp(' ')
disp('(1)No specific logic design style.Print result for all design
styles.= 0')
disp('(2)Fully static CMOS logic= 1')
disp('(3)Domino CMOS logic= 2')
disp('(4)Complementary pass transistor logic= 3')
disp(' ')
c=input('Please input logic design style option value= ');
while c>3
disp('ERROR: Option value is out of the range. Please give
option value between 0 and 3')
disp(' ')
 c=input('Please input logic design style option again= ');
disp(' ')
end
clc
disp('**********************************************
**************************************** ')
disp(' ')
disp('MOSFET Channel length is =1.2um')
disp(' ')
```

```
disp('**************************************************
**************************************** ')
disp('Channel width options and their values are:')
disp(' ')
disp('(1)No specific channel width choice.Print result for all
(w/l) values= 0')
disp('(2)Channel width (1.8um)= 1')
disp('(3)Channel width (3.6um)= 2')
disp('(4)Channel width (6.0um)= 3')
disp(' ')
d=input('Please input channel width option= ');
while d>3
disp('ERROR: Option value is out of the range. Please give
option value between 0 and 3')
disp(' ')
d=input('Please input channel width option again= ');
disp(' ')
end
disp('**************************************************
**************************************************
****************************** ')
 x,b,c,d%display adder size,architecture,transistor width,logic
design style
[delay]=p(x,b,c,d);%calling function p
disp('**************************************************
**************************************************
****************************** ')
disp('**************************************************
**************************************************
****************************** ')
disp('Energy consumption results')
disp(' ')
[energy]=ep(x,b,c,d);%calling function ep
elseif op==2
disp('**************************************************
**************************************** ')
disp(' ')
disp('Programme for obtaining best architecture for a required')
disp('energy consumption or propagation delay constraint or
both')
disp(' ')
disp('**************************************************
**************************************** ')
disp('Options for constraint/s are: ')
disp(' ')
disp('Propagation delay constraint only= 1 ')
disp('Energy consumption constraint only= 2 ')
disp('Both energy consumption and propagation delay
constraints =0')
disp(' ')
constraint=input('Please input constraint option= ');
disp('**************************************************
**************************************** ')
clc
while constraint>3;
disp('ERROR: Option value is out of the range. Please give
option value from 0 to 3')
disp(' ')
d=input('Please input constraint option again= ');
disp(' ')
end;
disp(' ')
clc;
x=input('Please input adder size i.e(number of bits) from 1 to
64 = ');
disp(' ')
while x==0;
disp(' ')
x=input('Please input adder size i.e(number of bits) from 1 to
64 again= ');
disp(' ')
end;
while x>64;
disp('ERROR: Adder size is out of the range. Please give adder
size from 1 to 64')
disp(' ')
 x=input('Please input adder size i.e(number of bits) from 1 to
64 again= ');
disp(' ')
while x==0;
disp('ERROR: Adder size is out of the range. Please give adder
size from 1 to 64')
disp(' ')
x=input('Please input adder size i.e(number of bits) from 1 to
64 again= ');
end;
end;
clc;
if constraint==1;
[delay]=delsim(x); %calling function delsim
 elseif constraint==2
 [energy]=enrgsim(x);%calling function enrgsim
 elseif constraint==0;
[delay]=delsim(x);
disp('----------------------------------------------------------
---------')
 [energy]=enrgsim(x);
end;
end;
diary off;
```

APPENDIX A2

Functions called in program listed in *Appendix A1*

FUNCTION 'DELSIM'

This function is called in main program listed in ***Appendix A1***. It takes the propagation delay as input and gives names of the adder designs having the specified value of critical delay.

```
function[delay]=delsim(x)
.
pdelay=input('Please
input propagationdelay
value,in seconds,= ');
disp(' ')
clc
disp('Tolerance --
acceptable variation in
propagationdelay and
energy consumption
value')
disp(' ')
ptol=input('Please input
tolerance value in
propagationdelay(seconds
) ');
clc
disp('*****************
**********************
**********************
**********************
**********')
countdel=0;
%function vrca1fs
[delay]=vrca1fs(x);
ifdelay<(pdelay+ptol);
ifdelay>(pdelay-ptol);
disp(' ')
disp(' ')
delay
disp('Architecture=
Ripple carry adder ')
disp('Channel width=
1.8um')
disp('Logic design style=
Fully static logic ')
countdel=countdel+1;
end;
end;
%function vrca2fs
[delay]=vrca2fs(x);
ifdelay<(pdelay+ptol);
ifdelay>(pdelay-ptol);
disp(' ')
disp(' ')
delay
disp('Architecture=
Ripple carry adder ')
disp('Channel width=
3.6um')
disp('Logic design style=
Fully static logic ')
countdel=countdel+1;
end;
end;
%function vrca3fs
[delay]=vrca3fs(x);
ifdelay<(pdelay+ptol);
ifdelay>(pdelay-ptol);
disp(' ')
disp(' ')
delay
disp('Architecture=
Ripple carry adder ')
disp('Channel width=
6um')
disp('Logic design style=
Fully static logic ')
countdel=countdel+1;
end;
end;
%function vrca1dm
[delay]=vrca1dm(x);
ifdelay<(pdelay+ptol);
ifdelay>(pdelay-ptol);
disp(' ')
disp(' ')
delay
disp('Architecture=
Ripple carry adder ')
disp('Channel width=
1.8um')
disp('Logic design style=
Domino CMOS logic ')
countdel=countdel+1;
end;
end;
%function vrca2dm
[delay]=vrca2dm(x);
ifdelay<(pdelay+ptol);
ifdelay>(pdelay-ptol);
disp(' ')
disp(' ')
delay
disp('Architecture=
Ripple carry adder ')
disp('Channel width=
3.6um')
disp('Logic design style=
Domino CMOS logic ')
countdel=countdel+1;
end;
end;
%function vrca3dm
[delay]=vrca3dm(x);
ifdelay<(pdelay+ptol);
ifdelay>(pdelay-ptol);
disp(' ')
disp(' ')
delay
disp('Architecture=
Ripple carry adder ')
disp('Channel width=
6um')
disp('Logic design style=
Domino CMOS logic ')
countdel=countdel+1;
end;
end;
%function vrca1cp
[delay]=vrca1cp(x);
ifdelay<(pdelay+ptol);
ifdelay>(pdelay-ptol);
disp(' ')
disp(' ')
delay
disp('Architecture=
Ripple carry adder ')
disp('Channel width=
1.8um')
disp('Logic design style=
Complementary pass-
transistor logic ')
countdel=countdel+1;
end;
end;
%function vrca2cp
[delay]=vrca2cp(x);
ifdelay<(pdelay+ptol);
ifdelay>(pdelay-ptol);
disp(' ')
disp(' ')
delay
disp('Architecture=
Ripple carry adder ')
disp('Channel width=
3.6um')
disp('Logic design style=
Complementary pass-
transistor logic ')
countdel=countdel+1;
end;
end;
%function vrca3cp
[delay]=vrca3cp(x);
ifdelay<(pdelay+ptol);
ifdelay>(pdelay-ptol);
disp(' ')
disp(' ')
delay
disp('Architecture=
Ripple carry adder ')
disp('Channel width=
6um')
disp('Logic design style=
Complementary pass-
transistor logic ')
countdel=countdel+1;
end;
end;
disp(' ')
%function vrca1dp
[delay]=vrca1dp(x);
ifdelay<(pdelay+ptol);
ifdelay>(pdelay-ptol);
disp(' ')
disp(' ')
delay
disp('Architecture=
Ripple carry adder ')
disp('Channel width=
1.8um')
disp('Logic design style=
Dual pass-transistor logic
')
countdel=countdel+1;
end;
end;
%function vrca2dp
[delay]=vrca2dp(x);
ifdelay<(pdelay+ptol);
ifdelay>(pdelay-ptol);
disp(' ')
disp(' ')
delay
disp('Architecture=
Ripple carry adder ')
disp('Channel width=
3.6um')
disp('Logic design style=
Dual pass-transistor logic
')
countdel=countdel+1;
end;
end;
%function vrca3dp
[delay]=vrca3dp(x);
ifdelay<(pdelay+ptol);
ifdelay>(pdelay-ptol);
disp(' ')
disp(' ')
delay
disp('Architecture=
Ripple carry adder ')
disp('Channel width=
6um')
disp('Logic design style=
Dual pass-transistor logic
')
countdel=countdel+1;
end;
end;
disp(' ')
%function vcrl1fs
[delay]=vcrl1fs(x);
ifdelay<(pdelay+ptol);
```

```
ifdelay>(pdelay-ptol);
disp(' ')
disp(' ')
delay
disp('Architecture= Carry
select adder ')
disp('Channel width=
1.8um')
disp('Logic design style=
Fully static logic ')
countdel=countdel+1;
end;
end;
%function vcrl2fs
[delay]=vcrl2fs(x);
ifdelay<(pdelay+ptol);
ifdelay>(pdelay-ptol);
disp(' ')
disp(' ')
delay
disp('Architecture= Carry
select adder ')
disp('Channel width=
3.6um')
disp('Logic design style=
Fully static logic ')
countdel=countdel+1;
end;
end;
%function vcrl3fs
[delay]=vcrl3fs(x);
ifdelay<(pdelay+ptol);
ifdelay>(pdelay-ptol);
disp(' ')
disp(' ')
delay
disp('Architecture= Carry
select adder ')
disp('Channel width=
6um')
disp('Logic design style=
Fully static logic ')
countdel=countdel+1;
end;
end;
%function vcrl1dm
[delay]=vcrl1dm(x);
ifdelay<(pdelay+ptol);
ifdelay>(pdelay-ptol);
disp(' ')
disp(' ')
delay
disp('Architecture= Carry
select adder ')
disp('Channel width=
1.8um')
disp('Logic design style=
Domino CMOS logic ')
countdel=countdel+1;
end;
end;
%function vcrl2dm
[delay]=vcrl2dm(x);
ifdelay<(pdelay+ptol);
ifdelay>(pdelay-ptol);
disp(' ')
disp(' ')
delay
disp('Architecture= Carry
select adder ')
disp('Channel width=
3.6um')
disp('Logic design style=
Domino CMOS logic ')
countdel=countdel+1;
end;
end;
%function vcrl3dm
[delay]=vcrl3dm(x);
ifdelay<(pdelay+ptol);
ifdelay>(pdelay-ptol);
disp(' ')
disp(' ')
delay
disp('Architecture= Carry
select adder ')
disp('Channel width=
6um')
disp('Logic design style=
Domino CMOS logic ')
countdel=countdel+1;
end;
end;
%function vcrl1cp
[delay]=vcrl1cp(x);
ifdelay<(pdelay+ptol);
ifdelay>(pdelay-ptol);
disp(' ')
disp(' ')
delay
disp('Architecture= Carry
select adder ')
disp('Channel width=
1.8um')
disp('Logic design style=
Complementary pass-
transistor logic ')
countdel=countdel+1;
end;
end;
%function vcrl2cp
[delay]=vcrl2cp(x);
ifdelay<(pdelay+ptol);
ifdelay>(pdelay-ptol);
disp(' ')
disp(' ')
delay
disp('Architecture= Carry
select adder ')
disp('Channel width=
3.6um')
disp('Logic design style=
Complementary pass-
transistor logic ')
countdel=countdel+1;
end;
end;
%function vcrl3cp
[delay]=vcrl3cp(x);
ifdelay<(pdelay+ptol);
ifdelay>(pdelay-ptol);
disp(' ')
disp(' ')
delay
disp('Architecture= Carry
select adder ')
disp('Channel width=
6um')
disp('Logic design style=
Complementary pass-
transistor logic ')
countdel=countdel+1;
end;
end;
%function vcrl1dp
[delay]=vcrl1dp(x);
ifdelay<(pdelay+ptol);
ifdelay>(pdelay-ptol);
disp(' ')
disp(' ')
delay
disp('Architecture= Carry
select adder ')
disp('Channel width=
1.8um')
disp('Logic design style=
Dual pass-transistor logic
')
countdel=countdel+1;
end;
end;
%function vcrl2dp
[delay]=vcrl2dp(x);
ifdelay<(pdelay+ptol);
ifdelay>(pdelay-ptol);
disp(' ')
disp(' ')
delay
disp('Architecture= Carry
select adder ')
disp('Channel width=
3.6um')
disp('Logic design style=
Dual pass-transistor logic
')
countdel=countdel+1;
end;
end;
%function vcrl3dp
[delay]=vcrl3dp(x);
ifdelay<(pdelay+ptol);
ifdelay>(pdelay-ptol);
disp(' ')
disp(' ')
delay
disp('Architecture= Carry
select adder ')
disp('Channel width=
6um')
disp('Logic design style=
Dual pass-transistor logic
')
countdel=countdel+1;
end;
end;
%function vcla1fs
[delay]=vcla1fs(x);
ifdelay<(pdelay+ptol);
ifdelay>(pdelay-ptol);
disp(' ')
disp(' ')
delay
disp('Architecture= Carry
look ahead adder ')
disp('Channel width=
1.8um')
disp('Logic design style=
Fully static logic ')
countdel=countdel+1;
end;
end;
%function vcla2fs
[delay]=vcla2fs(x);
ifdelay<(pdelay+ptol);
ifdelay>(pdelay-ptol);
disp(' ')
disp(' ')
delay
disp('Architecture= Carry
look ahead adder ')
disp('Channel width=
3.6um')
disp('Logic design style=
Fully static logic ')
countdel=countdel+1;
end;
end;
%function vcla3fs
[delay]=vcla3fs(x);
ifdelay<(pdelay+ptol);
ifdelay>(pdelay-ptol);
disp(' ')
disp(' ')
delay
disp('Architecture= Carry
look ahead adder ')
disp('Channel width=
6um')
disp('Logic design style=
Fully static logic ')
countdel=countdel+1;
end;
end;
%function vcla1dm
[delay]=vcla1dm(x);
ifdelay<(pdelay+ptol);
ifdelay>(pdelay-ptol);
disp(' ')
disp(' ')
delay
disp('Architecture= Carry
look ahead adder ')
disp('Channel width=
1.8um')
disp('Logic design style=
Domino CMOS logic ')
countdel=countdel+1;
end;
end;
%function vcla2dm
[delay]=vcla2dm(x);
ifdelay<(pdelay+ptol);
ifdelay>(pdelay-ptol);
disp(' ')
disp(' ')
delay
disp('Architecture= Carry
look ahead adder ')
disp('Channel width=
3.6um')
disp('Logic design style=
Domino CMOS logic ')
countdel=countdel+1;
end;
end;
%function vcla3dm
[delay]=vcla3dm(x);
ifdelay<(pdelay+ptol);
```

```
ifdelay>(pdelay-ptol);
disp(' ')
disp(' ')
delay
disp('Architecture= Carry
look ahead adder ')
disp('Channel width=
6um')
disp('Logic design style=
Domino CMOS logic ')
countdel=countdel+1;
end;
end;
%function vcla1cp
[delay]=vcla1cp(x);
ifdelay<(pdelay+ptol);
ifdelay>(pdelay-ptol);
disp(' ')
disp(' ')
delay
disp('Architecture= Carry
look ahead adder ')
disp('Channel width=
1.8um')
disp('Logic design style=
Complementary pass-
tansistor logic ')
countdel=countdel+1;
end;
end;
%function vcla2cp
[delay]=vcla2cp(x);
ifdelay<(pdelay+ptol);
ifdelay>(pdelay-ptol);
disp(' ')
disp(' ')
delay
disp('Architecture= Carry
look ahead adder ')
disp('Channel width=
3.6um')
disp('Logic design style=
Complementary pass-
tansistor logic ')
countdel=countdel+1;
end;
end;
%function vcla3cp
[delay]=vcla3cp(x);
ifdelay<(pdelay+ptol);
ifdelay>(pdelay-ptol);
disp(' ')
disp(' ')
delay
disp('Architecture= Carry
look ahead adder ')
disp('Channel width=
6um')
disp('Logic design style=
Complementary pass-
tansistor logic ')
countdel=countdel+1;
end;
end;
%function vcla1dp
[delay]=vcla1dp(x);
ifdelay<(pdelay+ptol);
ifdelay>(pdelay-ptol);
disp(' ')
disp(' ')
delay
disp('Architecture= Carry
look ahead adder ')
disp('Channel width=
1.8um')
disp('Logic design style=
Dual pass-tansistor logic
')
countdel=countdel+1;
end;
end;
%function vcla2dp
[delay]=vcla2dp(x);
ifdelay<(pdelay+ptol);
ifdelay>(pdelay-ptol);
disp(' ')
disp(' ')
delay
disp('Architecture= Carry
look ahead adder ')
disp('Channel width=
3.6um')
disp('Logic design style=
Dual pass-tansistor logic
')
countdel=countdel+1;
end;
end;
%function vcla3dp
[delay]=vcla3dp(x);
ifdelay<(pdelay+ptol);
ifdelay>(pdelay-ptol);
disp(' ')
disp(' ')
delay
disp('Architecture= Carry
look ahead adder ')
disp('Channel width=
6um')
disp('Logic design style=
Dual pass-tansistor logic
')
countdel=countdel+1;
end;
end;
%function vcsk1fs
[delay]=vcsk1fs(x);
ifdelay<(pdelay+ptol);
ifdelay>(pdelay-ptol);
disp(' ')
disp(' ')
delay
disp('Architecture= Carry
skip adder ')
disp('Channel width=
1.8um')
disp('Logic design style=
Fully static logic ')
countdel=countdel+1;
end;
end;
%function vcsk2fs
ifdelay<(pdelay+ptol);
ifdelay>(pdelay-ptol);
disp(' ')
disp(' ')
delay
disp('Architecture= Carry
skip adder ')
disp('Channel width=
3.6um')
disp('Logic design style=
Fully static logic ')
countdel=countdel+1;
end;
end;
%function vcsk1dm
[delay]=vcsk1dm(x);
ifdelay<(pdelay+ptol);
ifdelay>(pdelay-ptol);
disp(' ')
disp(' ')
delay
disp('Architecture= Carry
skip adder ')
disp('Channel width=
1.8um')
disp('Logic design style=
Domino CMOS logic ')
countdel=countdel+1;
end;
end;
%function vcsk2dm
[delay]=vcsk2dm(x);
ifdelay<(pdelay+ptol);
ifdelay>(pdelay-ptol);
disp(' ')
disp(' ')
delay
disp('Architecture= Carry
skip adder ')
disp('Channel width=
3.6um')
disp('Logic design style=
Domino CMOS logic ')
countdel=countdel+1;
end;
end;
%function vcsk3dm
[delay]=vcsk3dm(x);
ifdelay<(pdelay+ptol);
ifdelay>(pdelay-ptol);
disp(' ')
disp(' ')
delay
disp('Architecture= Carry
skip adder ')
disp('Channel width=
6um')
disp('Logic design style=
Domino CMOS logic ')
countdel=countdel+1;
end;
%function vcsk1cp
[delay]=vcsk1cp(x);
ifdelay<(pdelay+ptol);
ifdelay>(pdelay-ptol);
disp(' ')
disp(' ')
delay
disp('Architecture= Carry
skip adder ')
disp('Channel width=
1.8um')
disp('Logic design style=
Complementary pass-
transistor logic ')
countdel=countdel+1;
end;
end;
%function vcsk2cp
[delay]=vcsk2cp(x);
ifdelay<(pdelay+ptol);
ifdelay>(pdelay-ptol);
disp(' ')
disp(' ')
delay
disp('Architecture= Carry
skip adder ')
disp('Channel width=
3.6um')
disp('Logic design style=
Complementary pass-
transistor logic ')
countdel=countdel+1;
end;
end;
%function vcsk3cp
[delay]=vcsk3cp(x);
ifdelay<(pdelay+ptol);
ifdelay>(pdelay-ptol);
disp(' ')
disp(' ')
delay
disp('Architecture= Carry
skip adder ')
disp('Channel width=
6um')
disp('Logic design style=
Complementary pass-
transistor logic ')
countdel=countdel+1;
end;
end;
%function vcsk1dp
[delay]=vcsk1dp(x);
ifdelay<(pdelay+ptol);
ifdelay>(pdelay-ptol);
disp(' ')
disp(' ')
delay
disp('Architecture= Carry
skip adder ')
disp('Channel width=
1.8um')
disp('Logic design style=
Dual pass-transistor logic
')
countdel=countdel+1;
end;
end;
%function vcsk2dp
```

```
[delay]=vcsk2dp(x);
ifdelay<(pdelay+ptol);
ifdelay>(pdelay-ptol);
disp(' ')
disp(' ')
delay
disp('Architecture= Carry
skip adder ')
disp('Channel width=
3.6um')
disp('Logic design style=
Dual pass-transistor logic
')
countdel=countdel+1;
end;
end;
%function vcsk3dp
[delay]=vcsk3dp(x);
ifdelay<(pdelay+ptol);
ifdelay>(pdelay-ptol);
disp(' ')
disp(' ')
delay
disp('Architecture= Carry
skip adder ')
disp('Channel width=
6um')
disp('Logic design style=
Dual pass-transistor logic
')
countdel=countdel+1;
end;
end;
%function vcnd1fs
[delay]=vcnd1fs(x);
ifdelay<(pdelay+ptol);
ifdelay>(pdelay-ptol);
disp(' ')
disp(' ')
delay
disp('Architecture=
Conditional sum adder ')
disp('Channel width=
1.8um')
disp('Logic design style=
Fully static logic ')
countdel=countdel+1;
end;
end;
%function vcnd2fs
[delay]=vcnd2fs(x);
ifdelay<(pdelay+ptol);
ifdelay>(pdelay-ptol);
disp(' ')
disp(' ')
delay
disp('Architecture=
Conditional sum adder ')
disp('Channel width=
3.6um')
disp('Logic design style=
Fully static logic ')
disp(' ')
countdel=countdel+1;
end;
end;
%function vcnd3fs
[delay]=vcnd3fs(x);
ifdelay<(pdelay+ptol);
ifdelay>(pdelay-ptol);
disp(' ')
disp(' ')
delay
disp('Architecture=
Conditional sum adder ')
disp('Channel width=
6um')
disp('Logic design style=
Fully static logic ')
countdel=countdel+1;
end;
end;
%function vcnd1dm
[delay]=vcnd1dm(x);
ifdelay<(pdelay+ptol);
ifdelay>(pdelay-ptol);
disp(' ')
disp(' ')
delay
disp('Architecture=
Conditional sum adder ')
disp('Channel width=
1.8um')
disp('Logic design style=
Domino CMOS logic ')
countdel=countdel+1;
end;
end;
%function vcnd2dm
[delay]=vcnd2dm(x);
ifdelay<(pdelay+ptol);
ifdelay>(pdelay-ptol);
disp(' ')
disp(' ')
delay
disp('Architecture=
Conditional sum adder ')
disp('Channel width=
3.6um')
disp('Logic design style=
Domino CMOS logic ')
countdel=countdel+1;
end;
end;
%function vcnd3dm
[delay]=vcnd3dm(x);
ifdelay<(pdelay+ptol);
ifdelay>(pdelay-ptol);
disp(' ')
disp(' ')
delay
disp('Architecture=
Conditional sum adder ')
disp('Channel width=
6um')
disp('Logic design style=
Domino CMOS logic ')
countdel=countdel+1;
end;
end;
%function vcnd1cp
[delay]=vcnd1cp(x);
ifdelay<(pdelay+ptol);
ifdelay>(pdelay-ptol);
disp(' ')
disp(' ')
delay
disp('Architecture=
Conditional sum adder ')
disp('Channel width=
1.8um')
disp('Logic design style=
Complementary pass-
transistor logic ')
countdel=countdel+1;
end;
end;
%function vcnd2cp
[delay]=vcnd2cp(x);
ifdelay<(pdelay+ptol);
ifdelay>(pdelay-ptol);
disp(' ')
disp(' ')
delay
disp('Architecture=
Conditional sum adder ')
disp('Channel width=
3.6um')
disp('Logic design style=
Complementary pass-
transistor logic ')
countdel=countdel+1;
end;
end;
%function vcnd3cp
[delay]=vcnd3cp(x);
ifdelay<(pdelay+ptol);
ifdelay>(pdelay-ptol);
disp(' ')
disp(' ')
delay
disp('Architecture=
Conditional sum adder ')
disp('Channel width=
6um')
disp('Logic design style=
Complementary pass-
transistor logic ')
countdel=countdel+1;
end;
end;
%function vcnd1dp
[delay]=vcnd1dp(x);
ifdelay<(pdelay+ptol);
ifdelay>(pdelay-ptol);
disp(' ')
disp(' ')
delay
disp('Architecture=
Conditional sum adder ')
disp('Channel width=
1.8um')
disp('Logic design style=
Dual pass-transistor logic
')
countdel=countdel+1;
end;
end;
%function vcnd2dp
[delay]=vcnd2dp(x);
ifdelay<(pdelay+ptol);
ifdelay>(pdelay-ptol);
disp(' ')
disp(' ')
delay
disp('Architecture=
Conditional sum adder ')
disp('Channel width=
3.6um')
disp('Logic design style=
Dual pass-transistor logic
')
countdel=countdel+1;
end;
end;
%function vcnd3dp
[delay]=vcnd3dp(x);
ifdelay<(pdelay+ptol);
ifdelay>(pdelay-ptol);
disp(' ')
disp(' ')
delay
disp('Architecture=
Conditional sum adder ')
disp('Channel width=
6um')
disp('Logic design style=
Dual pass-transistor logic
')
countdel=countdel+1;
end;
end;
countdel;
if countdel==0;
disp('Adder architecture
with required worst-case
propagationdelay is not
found. Please relax
tolerance')
 end;
```

FUNCTION 'ENRGSIM'

This function is called in ***Appendix A1***. It takes the energy consumption per addition as input and gives names of the adder designs having energy consumption near the specified value.

```
function[energy]=enrgsim
(x);
%function call
%sim
%width=1.8um
cenergy=input('Please
input requiredenergy
consumption value,in
Joules,= ');
disp(' ')
clc
disp('Tolerance -->
acceptable
variationenergy
consumption value (in
Joules)')
disp(' ')
etol=input('Please input
tolerance value inenergy
consumption value
(Joules) = ');
clc
disp('*****************
**********************
**********************
**********************
**********')
count=0;
%function vrcae1fs
[energy]=vrcae1fs(x);
if energy<(cenergy+etol);
if energy>(cenergy-etol);
disp(' ')
disp(' ')
energy
disp('Architecture=
Ripple carry adder ')
disp('Channel width=
1.8um')
disp('Logic design style=
Fully static logic ')
count=count+1;
end;
end;
%function vrcae2fs
[energy]=vrcae2fs(x);
if energy<(cenergy+etol);
if energy>(cenergy-etol);
disp(' ')
disp(' ')
energy
disp('Architecture=
Ripple carry adder ')
disp('Channel width=
3.6um')
disp('Logic design style=
Fully static logic ')
count=count+1;
end;
end;
%function vrcae3fs
[energy]=vrcae3fs(x);
if energy<(cenergy+etol);
if energy>(cenergy-etol);
disp(' ')
disp(' ')
energy
disp('Architecture=
Ripple carry adder ')
disp('Channel width=
6um')
disp('Logic design style=
Fully static logic ')
count=count+1;
end;
end;
%function vrcae1dm
[energy]=vrcae1dm(x);
if energy<(cenergy+etol);
if energy>(cenergy-etol);
disp(' ')
disp(' ')
energy
disp('Architecture=
Ripple carry adder ')
disp('Logic design style=
Domino CMOS logic ')
count=count+1;
end;
end;
%function vrcae2dm
[energy]=vrcae2dm(x);
if energy<(cenergy+etol);
if energy>(cenergy-etol);
disp(' ')
disp(' ')
energy
disp('Architecture=
Ripple carry adder ')
disp('Channel width=
3.6um')
disp('Logic design style=
Domino CMOS logic ')
count=count+1;
end;
end;
%function vrcae3dm
[energy]=vrcae3dm(x);
if energy<(cenergy+etol);
if energy>(cenergy-etol);
disp(' ')
disp(' ')
energy
disp('Architecture=
Ripple carry adder ')
disp('Channel width=
6um')
disp('Logic design style=
Domino CMOS logic ')
count=count+1;
end;
end;
%function vrcae1cp
[energy]=vrcae1cp(x);
if energy<(cenergy+etol);
if energy>(cenergy-etol);
disp(' ')
disp(' ')
energy
disp('Architecture=
Ripple carry adder ')
disp('Channel width=
1.8um')
disp('Logic design style=
Complementary pass-
transistor logic ')
count=count+1;
end;
end;
%function vrcae2cp
[energy]=vrcae2cp(x);
if energy<(cenergy+etol);
if energy>(cenergy-etol);
disp(' ')
disp(' ')
energy
disp('Architecture=
Ripple carry adder ')
disp('Channel width=
3.6um')
disp('Logic design style=
Complementary pass-
transistor logic ')
count=count+1;
end;
end;
%function vrcae3cp
[energy]=vrcae3cp(x);
if energy<(cenergy+etol);
if energy>(cenergy-etol);
disp(' ')
disp(' ')
energy
disp('Architecture=
Ripple carry adder ')
disp('Channel width=
6um')
disp('Logic design style=
Complementary pass-
transistor logic ')
count=count+1;
end;
end;
%function vrcae1dp
[energy]=vrcae1dp(x);
if energy<(cenergy+etol);
if energy>(cenergy-etol);
disp(' ')
disp(' ')
energy
disp('Architecture=
Ripple carry adder ')
disp('Channel width=
1.8um')
disp('Logic design style=
Dual pass-transistor logic
')
count=count+1;
end;
end;
%function vrcae2dp
[energy]=vrcae2dp(x);
if energy<(cenergy+etol);
if energy>(cenergy-etol);
disp(' ')
disp(' ')
energy
disp('Architecture=
Ripple carry adder ')
disp('Channel width=
3.6um')
disp('Logic design style=
Dual pass-transistor logic
')
count=count+1;
end;
end;
%function vrcae3dp
[energy]=vrcae3dp(x);
if energy<(cenergy+etol);
if energy>(cenergy-etol);
disp(' ')
disp(' ')
energy
disp('Architecture=
Ripple carry adder ')
disp('Channel width=
6um')
disp('Logic design style=
Dual pass-transistor logic
')
count=count+1;
end;
end;
%function vcrle1fs
[energy]=vcrle1fs(x);
if energy<(cenergy+etol);
if energy>(cenergy-etol);
disp(' ')
disp(' ')
energy
disp('Architecture= Carry
select adder ')
disp('Channel width=
1.8um')
disp('Logic design style=
Fully static logic ')
count=count+1;
end;
end;
%function vcrle2fs
[energy]=vcrle2fs(x);
if energy<(cenergy+etol);
if energy>(cenergy-etol);
disp(' ')
disp(' ')
energy
disp('Architecture= Carry
select adder ')
disp('Channel width=
3.6um')
disp('Logic design style=
Fully static logic ')
count=count+1;
end;
end;
%function vcrle3fs
[energy]=vcrle3fs(x);
if energy<(cenergy+etol);
if energy>(cenergy-etol);
disp(' ')
disp(' ')
energy
disp('Architecture= Carry
select adder ')
disp('Channel width=
6um')
disp('Logic design style=
Fully static logic ')
count=count+1;
end;
end;
%function vcrle1dm
[energy]=vcrle1dm(x);
if energy<(cenergy+etol);
if energy>(cenergy-etol);
disp(' ')
disp(' ')
energy
```

```
disp('Architecture= Carry
select adder ')
disp('Channel width=
1.8um')
disp('Logic design style=
Domino CMOS logic ')
count=count+1;
end;
end;
%function vcrle2dm
[energy]=vcrle2dm(x);
if energy<(cenergy+etol);
if energy>(cenergy-etol);
disp(' ')
disp(' ')
energy
disp('Architecture= Carry
select adder ')
disp('Channel width=
3.6um')
disp('Logic design style=
Domino CMOS logic ')
count=count+1;
end;
end;
%function vcrle3dm
[energy]=vcrle3dm(x);
if energy<(cenergy+etol);
if energy>(cenergy-etol);
disp(' ')
disp(' ')
energy
disp('Architecture= Carry
select adder ')
disp('Channel width=
6um')
disp('Logic design style=
Domino CMOS logic ')
count=count+1;
end;
end;
%function vcrle1cp
[energy]=vcrle1cp(x);
if energy<(cenergy+etol);
if energy>(cenergy-etol);
disp(' ')
disp(' ')
energy
disp('Architecture= Carry
select adder ')
disp('Channel width=
1.8um')
disp('Logic design style=
Complementary pass-
transistor logic ')
count=count+1;
end;
end;
%function vcrle2cp
[energy]=vcrle2cp(x);
if energy<(cenergy+etol);
if energy>(cenergy-etol);
disp(' ')
disp(' ')
energy
disp('Architecture= Carry
select adder ')
disp('Channel width=
3.6um')
disp('Logic design style=
Complementary pass-
transistor logic ')
count=count+1;
end;
end;
%function vcrle3cp
[energy]=vcrle3cp(x);
if energy<(cenergy+etol);
if energy>(cenergy-etol);
disp(' ')
disp(' ')
energy
disp('Architecture= Carry
select adder ')
disp('Channel width=
6um')
disp('Logic design style=
Complementary pass-
transistor logic ')
count=count+1;
end;
end;
%function vcrle1dp
[energy]=vcrle1dp(x);
if energy<(cenergy+etol);
if energy>(cenergy-etol);
disp(' ')
disp(' ')
energy
disp('Architecture= Carry
select adder ')
disp('Channel width=
1.8um')
disp('Logic design style=
Dual pass-transistor logic
')
count=count+1;
end;
end;
%function vcrle2dp
[energy]=vcrle2dp(x);
if energy<(cenergy+etol);
if energy>(cenergy-etol);
disp(' ')
disp(' ')
energy
disp('Architecture= Carry
select adder ')
disp('Channel width=
3.6um')
disp('Logic design style=
Dual pass-transistor logic
')
count=count+1;
end;
end;
%function vcrle3dp
[energy]=vcrle3dp(x);
if energy<(cenergy+etol);
if energy>(cenergy-etol);
disp(' ')
disp(' ')
energy
disp('Architecture= Carry
select adder ')
disp('Channel width=
6um')
disp('Logic design style=
Dual pass-transistor logic
')
count=count+1;
end;
end;
%function vclae1fs
[energy]=vclae1fs(x);
if energy<(cenergy+etol);
if energy>(cenergy-etol);
disp(' ')
disp(' ')
energy
disp('Architecture= Carry
look ahead adder ')
disp('Channel width=
1.8um')
disp('Logic design style=
Fully static logic ')
count=count+1;
end;
end;
%function vclae2fs
[energy]=vclae2fs(x);
if energy<(cenergy+etol);
if energy>(cenergy-etol);
disp(' ')
disp(' ')
energy
disp('Architecture= Carry
look ahead adder ')
disp('Channel width=
3.6um')
disp('Logic design style=
Fully static logic ')
count=count+1;
end;
end;
%function vclae3fs
[energy]=vclae3fs(x);
if energy<(cenergy+etol);
if energy>(cenergy-etol);
disp(' ')
disp(' ')
energy
disp('Architecture= Carry
look ahead adder ')
disp('Channel width=
6um')
disp('Logic design style=
Fully static logic ')
count=count+1;
end;
end;
%function vclae1dm
[energy]=vclae1dm(x);
if energy<(cenergy+etol);
if energy>(cenergy-etol);
disp(' ')
disp(' ')
energy
disp('Architecture= Carry
look ahead adder ')
disp('Channel width=
1.8um')
disp('Logic design style=
Domino CMOS logic ')
count=count+1;
end;
end;
%function vclae2dm
[energy]=vclae2dm(x);
if energy<(cenergy+etol);
if energy>(cenergy-etol);
disp(' ')
disp(' ')
energy
disp('Architecture= Carry
look ahead adder ')
disp('Channel width=
3.6um')
disp('Logic design style=
Domino CMOS logic ')
count=count+1;
 end;
end;
%function vclae3dm
[energy]=vclae3dm(x);
if energy<(cenergy+etol);
if energy>(cenergy-etol);
disp(' ')
disp(' ')
energy
disp('Architecture= Carry
look ahead adder ')
disp('Channel width=
6um')
disp('Logic design style=
Domino CMOS logic ')
count=count+1;
end;
end;
%function vclae1cp
[energy]=vclae1cp(x);
if energy<(cenergy+etol);
if energy>(cenergy-etol);
disp(' ')
disp(' ')
energy
disp('Architecture= Carry
look ahead adder ')
disp('Channel width=
1.8um')
disp('Logic design style=
Complementary pass-
tansistor logic ')
count=count+1;
end;
end;
%function vclae2cp
[energy]=vclae2cp(x);
if energy<(cenergy+etol);
if energy>(cenergy-etol);
disp(' ')
disp(' ')
energy
disp('Architecture= Carry
look ahead adder ')
disp('Channel width=
3.6um')
disp('Logic design style=
Complementary pass-
tansistor logic ')
count=count+1;
end;
end;
%function vclae3cp
[energy]=vclae3cp(x);
if energy<(cenergy+etol);
if energy>(cenergy-etol);
```

```
disp(' ')
disp(' ')
energy
disp('Architecture= Carry
look ahead adder ')
disp('Channel width=
6um')
disp('Logic design style=
Complementary pass-
tansistor logic ')
count=count+1;
end;
end;
%function vclae1dp
[energy]=vclae1dp(x);
if energy<(cenergy+etol);
if energy>(cenergy-etol);
disp(' ')
disp(' ')
energy
disp('Architecture= Carry
look ahead adder ')
disp('Channel width=
1.8um')
disp('Logic design style=
Dual pass-tansistor logic
')
count=count+1;
end;
end;
%function vclae2dp
[energy]=vclae2dp(x);
if energy<(cenergy+etol);
if energy>(cenergy-etol);
disp(' ')
disp(' ')
energy
disp('Architecture= Carry
look ahead adder ')
disp('Channel width=
3.6um')
disp('Logic design style=
Dual pass-tansistor logic
')
count=count+1;
end;
end;
%function vclae3dp
[energy]=vclae3dp(x);
if energy<(cenergy+etol);
if energy>(cenergy-etol);
disp(' ')
disp(' ')
energy
disp('Architecture= Carry
look ahead adder ')
disp('Channel width=
6um')
disp('Logic design style=
Dual pass-tansistor logic
')
count=count+1;
end;
end;
%function vcske1fs
[energy]=vcske1fs(x);
if energy<(cenergy+etol);
if energy>(cenergy-etol);
disp(' ')
disp(' ')
energy
disp('Architecture= Carry
skip adder ')
disp('Channel width=
1.8um')
disp('Logic design style=
Fully static logic ')
count=count+1;
end;
end;
%function vcske2fs
[energy]=vcske2fs(x);
if energy<(cenergy+etol);
if energy>(cenergy-etol);
disp(' ')
disp(' ')
energy
disp('Architecture= Carry
skip adder ')
disp('Channel width=
3.6um')
disp('Logic design style=
Fully static logic ')
count=count+1;
end;
end;
%function vcske3fs
[energy]=vcske3fs(x);
if energy<(cenergy+etol);
if energy>(cenergy-etol);
disp(' ')
disp(' ')
energy
disp('Architecture= Carry
skip adder ')
disp('Channel width=
6um')
disp('Logic design style=
Fully static logic ')
count=count+1;
end;
end;
%function vcske1dm
[energy]=vcske1dm(x);
if energy<(cenergy+etol);
if energy>(cenergy-etol);
disp(' ')
disp(' ')
energy
disp('Architecture= Carry
skip adder ')
disp('Channel width=
1.8um')
disp('Logic design style=
Domino CMOS logic ')
count=count+1;
end;
end;
%function vcske2dm
[energy]=vcske2dm(x);
if energy<(cenergy+etol);
if energy>(cenergy-etol);
disp(' ')
disp(' ')
energy
disp('Architecture= Carry
skip adder ')
disp('Channel width=
3.6um')
disp('Logic design style=
Domino CMOS logic ')
count=count+1;
end;
end;
%function vcske1cp
[energy]=vcske1cp(x);
if energy<(cenergy+etol);
if energy>(cenergy-etol);
disp(' ')
disp(' ')
energy
disp('Architecture= Carry
skip adder ')
disp('Channel width=
1.8um')
disp('Logic design style=
Complementary pass-
transistor logic ')
count=count+1;
end;
end;
%function vcske2cp
[energy]=vcske2cp(x);
if energy<(cenergy+etol);
if energy>(cenergy-etol);
disp(' ')
disp(' ')
energy
disp('Architecture= Carry
skip adder ')
disp('Channel width=
3.6um')
disp('Logic design style=
Complementary pass-
transistor logic ')
count=count+1;
end;
end;
%function vcske3cp
[energy]=vcske3cp(x);
if energy<(cenergy+etol);
if energy>(cenergy-etol);
disp(' ')
disp(' ')
energy
disp('Architecture= Carry
skip adder ')
disp('Channel width=
6um')
disp('Logic design style=
Complementary pass-
transistor logic ')
count=count+1;
end;
end;
%function vcske1dp
[energy]=vcske1dp(x);
if energy<(cenergy+etol);
if energy>(cenergy-etol);
disp(' ')
disp(' ')
energy
disp('Architecture= Carry
skip adder ')
disp('Channel width=
1.8um')
disp('Logic design style=
Dual pass-transistor logic
')
count=count+1;
end;
end;
%function vcske2dp
[energy]=vcske2dp(x);
if energy<(cenergy+etol);
if energy>(cenergy-etol);
disp(' ')
disp(' ')
energy
disp('Architecture= Carry
skip adder ')
disp('Channel width=
3.6um')
disp('Logic design style=
Dual pass-transistor logic
')
count=count+1;
end;
end;
%function vcske3dp
[energy]=vcske3dp(x);
if energy<(cenergy+etol);
if energy>(cenergy-etol);
disp(' ')
disp(' ')
energy
disp('Architecture= Carry
skip adder ')
disp('Channel width=
6um')
disp('Logic design style=
Dual pass-transistor logic
')
count=count+1;
end;
end;
%function vcnde1fs
[energy]=vcnde1fs(x);
if energy<(cenergy+etol);
if energy>(cenergy-etol);
disp(' ')
disp(' ')
energy
disp('Architecture=
Conditional sum adder ')
disp('Channel width=
1.8um')
disp('Logic design style=
Fully static logic ')
count=count+1;
end;
end;
%function vcnde2fs
```

```
[energy] = vcnde2fs(x);
if energy<(cenergy-etol);
if energy>(cenergy-etol);
disp(' ')
disp(' ')
energy
disp('Architecture=
Conditional sum adder ')
disp('Channel width=
3.6um')
disp('Logic design style=
Fully static logic ')
disp(' ')
count=count+1;
end;
end;
%function vcnde3fs
[energy]=vcnde3fs(x);
if energy<(cenergy+etol);
if energy>(cenergy-etol);
disp(' ')
disp(' ')
energy
disp('Architecture=
Conditional sum adder ')
disp('Channel width=
6um')
disp('Logic design style=
Fully static logic ')
count=count+1;
end;
end;
%function vcnde1dm
[energy]=vcnde1dm(x);
if energy<(cenergy+etol);
if energy>(cenergy-etol);
disp(' ')
disp(' ')
energy
disp('Architecture=
Conditional sum adder ')
disp('Channel width=
1.8um')
disp('Logic design style=
Domino CMOS logic ')
count=count+1;
end;
end;
%function vcnde2dm
[energy]=vcnde2dm(x);
if energy<(cenergy+etol);
if energy>(cenergy-etol);
disp(' ')
disp(' ')
disp('Architecture=
Conditional sum adder ')
disp('Channel width=
3.6um')
disp('Logic design style=
Domino CMOS logic ')
count=count+1;
end;
end;
%function vcnde3dm
[energy]=vcnde3dm(x);
if energy<(cenergy+etol);
if energy>(cenergy-etol);
disp(' ')
disp(' ')
energy
disp('Architecture
Conditional sum adder ')
disp('Channel width-
6um')
disp('Logic design style=
Domino CMOS logic ')
count=count+1;
end;
end;
%function vcnde1cp
[energy]=vcnde1cp(x);
if energy<(cenergy+etol);
if energy>(cenergy-etol);
disp(' ')
disp(' ')
energy
disp('Architecture=
Conditional sum adder ')
disp('Channel width=
1.8um')
disp('Logic design style=
Complementary pass-
transistor logic ')
count=count+1;
end;
end;
%function vcnde2cp
[energy]=vcnde2cp(x);
if energy<(cenergy+etol);
if energy>(cenergy-etol);
disp(' ')
disp(' ')
energy
disp('Architecture=
Conditional sum adder ')
disp('Channel width=
3.6um')
disp('Logic design style=
Complementary pass-
transistor logic ')
count=count+1;
end;
end;
%function vcnde3cp
[energy]=vcnde3cp(x);
if energy<(cenergy+etol);
if energy>(cenergy-etol);
disp(' ')
disp(' ')
energy
disp('Architecture=
Conditional sum adder ')
disp('Channel width=
6um')
disp('Logic design style=
Complementary pass-
transistor logic ')
count=count+1;
end;
end;
%function vcnde1dp
[energy]=vcnde1dp(x);
if energy<(cenergy+etol);
if energy>(cenergy-etol);
disp(' ')
disp(' ')
energy
disp('Architecture=
Conditional sum adder ')
disp('Channel width
1.8um')
disp('Logic design style=
Dual pass-transistor logic
')
count=count+1;
end;
end;
%function vcnde2dp
[energy]=vcnde2dp(x);
if energy<(cenergy+etol);
if energy>(cenergy-etol);
disp(' ')
disp(' ')
energy
disp('Architecture=
Conditional sum adder ')
disp('Channel width=
3.6um')
disp('Logic design style=
Dual pass-transistor logic
')
count=count+1;
end;
end;
%function vcnde3dp
[energy]=vcnde3dp(x);
if energy<(cenergy+etol);
if energy>(cenergy-etol);
disp(' ')
disp(' ')
energy
disp('Architecture=
Conditional sum adder ')
disp('Channel width=
6um')
disp('Logic design style=
Dual pass-transistor logic
')
count=count+1;
end;
end;
count;
ifcount==0;
disp('Adder architecture
with requiredenergy
consumption is not found.
Please relax tolerance')
end;
```

FUNCTION ' P'

This function is called in ***Appendix A1***. It takes the adder design as input and calculates the worst-case propagation delays of the given adder designs

```
function[delay]=p(x,b,c,
d);
if b==1;
disp('Ripple carry adder
architecture')
disp('----------
---------- ')
disp(' ')
if c==1;
disp('Fully static CMOS
logic design style')
disp('----------
---------- ')
disp(' ')
if d==1;
disp(' ')
[delay]=rca1fs(x)
elseif d==2;
[delay]=rca2fs(x)
elseif d==3;
[delay]=rca3fs(x)
else;
[delay]=rca1fs(x)
[delay]=rca2fs(x)
[delay]=rca3fs(x)
end;
elseif c==2;
disp('Domino CMOS
logic design style')
disp('----------
---------- ')
disp(' ')
if d==1;
disp(' ')
[delay]=rca1dm(x)
elseif d==2;
[delay]=rca2dm(x)
elseif d==3;
[delay]=rca3dm(x)
else;
[delay]=rca1dm(x)
[delay]=rca2dm(x)
[delay]=rca3dm(x)
end;
elseif c==3;
disp('Complementary
pass-transistor logic
design style')
disp('----------
---------- ')
disp(' ')
if d==1;
disp(' ')
[delay]=rca1cp(x)
elseif d==2;
[delay]=rca2cp(x)
elseif d==3;
[delay]=rca3cp(x)
else;
[delay]=rca1cp(x)
[delay]=rca2cp(x)
[delay]=rca3cp(x)
end;
elseif c==4;
disp('Dual pass-
transistor logic design
style')
disp('----------
---------- ')
disp(' ')
if d==1;
disp(' ')
[delay]=rca1dp(x)
elseif d==2;
[delay]=rca2dp(x)
elseif d==3;
[delay]=rca3dp(x)
else;
[delay]=rca1dp(x)
[delay]=rca2dp(x)
[delay]=rca3dp(x)
end;
elseif c==0;
disp('Fully static CMOS
logic design style')
disp('----------
---------- ')
disp(' ')
if d==1;
disp(' ')
[delay]=rca1fs(x)
elseif d==2;
[delay]=rca2fs(x)
elseif d==3;
[delay]=rca3fs(x)
 else;
[delay]=rca1fs(x)
[delay]=rca2fs(x)
[delay]=rca3fs(x)
end;
disp('Domino CMOS
logic design style')
disp('----------
---------- ')
disp(' ')
if d==1;
disp(' ')
[delay]=rca1dm(x)
elseif d==2;
[delay]=rca2dm(x)
elseif d==3;
[delay]=rca3dm(x)
else;
[delay]=rca1dm(x)
[delay]=rca2dm(x)
[delay]=rca3dm(x)
end;
disp('Complementary
pass-transistor logic
design style')
disp('----------
---------- ')
disp(' ')
if d==1;
disp(' ')
[delay]=rca1cp(x)
elseif d==2;
[delay]=rca2cp(x)
elseif d==3;
[delay]=rca3cp(x)
else;
[delay]=rca1cp(x)
[delay]=rca2cp(x)
[delay]=rca3cp(x)
end;
disp('Dual pass-
transistor logic design
style')
disp('----------
---------- ')
disp(' ')
if d==1;
disp(' ')
[delay]=rca1dp(x)
elseif d==2;
[delay]=rca2dp(x)
elseif d==3;
[delay]=rca3dp(x)
else;
[delay]=rca1dp(x)
[delay]=rca2dp(x)
[delay]=rca3dp(x)
end;
end;
elseif b==2;
disp('Carry select adder
architecture')
disp('----------
---------- ')
disp(' ')
if c==1;
disp('Fully static CMOS
logic design style')
disp('----------
---------- ')
disp(' ')
if d==1;
disp(' ')
disp(' ')
[delay]=crl1fs(x)
elseif d==2;
[delay]=crl2fs(x)
elseif d==3;
[delay]=crl3fs(x)
 else;
[delay]=crl1fs(x)
[delay]=crl2fs(x)
[delay]=crl3fs(x)
end;
elseif c==2;
disp('Domino CMOS
logic design style')
disp('----------
---------- ')
disp(' ')
if d==1;
disp(' ')
[delay]=crl1dm(x)
elseif d==2;
[delay]=crl2dm(x)
elseif d==3;
[delay]=crl3dm(x)
else;
[delay]=crl1dm(x)
[delay]=crl2dm(x)
[delay]=crl3dm(x)
end;
elseif c==3;
disp('Complementary
pass-transistor logic
design style')
disp('----------
---------- ')
disp(' ')
if d==1;
disp(' ')
[delay]=crl1cp(x)
elseif d==2;
[delay]=crl2cp(x)
elseif d==3;
[delay]=crl3cp(x)
else;
[delay]=crl1cp(x)
[delay]=crl2cp(x)
[delay]=crl3cp(x)
end;
elseif c==4;
disp('Dual pass-
transistor logic design
style')
disp('----------
---------- ')
disp(' ')
if d==1;
disp(' ')
[delay]=crl1dp(x)
elseif d==2;
[delay]=crl2dp(x)
elseif d==3;
[delay]=crl3dp(x)
else;
[delay]=crl1dp(x)
[delay]=crl2dp(x)
[delay]=crl3dp(x)
end;
elseif c==0;
disp('Fully static CMOS
logic design style')
disp('----------
---------- ')
disp(' ')
if d==1;
disp(' ')
[delay]=crl1fs(x)
elseif d==2;
[delay]=crl2fs(x)
elseif d==3;
[delay]=crl3fs(x)
 else;
[delay]=crl1fs(x)
[delay]=crl2fs(x)
[delay]=crl3fs(x)
end;
disp('Domino CMOS
logic design style')
disp('----------
---------- ')
disp(' ')
if d==1;
```

```
disp(' ')
[delay]=crldm(x)
elseif d==2;
[delay]=crl2dm(x)
elseif d==3;
[delay]=crl3dm(x)
 else;
[delay]=crl1dm(x)
 [delay]=crl2dm(x)
[delay]=crl3dm(x)
end;
disp('Complementary
pass-transistor logic
design style')
disp('————————————
————————————')
disp(' ')
if d==1;
disp(' ')
[delay]=crl1cp(x)
elseif d==2;
[delay]=crl2cp(x)
elseif d==3;
[delay]=crl3cp(x)
else;
[delay]=crl1cp(x)
[delay]=crl2cp(x)
[delay]=crl3cp(x)
end;
disp('Dual pass-
transistor logic design
style')
disp('————————————
————————————')
disp(' ')
if d==1;
disp(' ')
[delay]=crl1dp(x)
elseif d==2;
[delay]=crl2dp(x)
elseif d==3;
[delay]=crl3dp(x)
 else;
[delay]=crl1dp(x)

[delay]=crl2dp(x)
[delay]=crl3dp(x)
end;
end;
elseif b==3;
disp('Carry skip adder
architecture')
disp('————————————
————')
disp(' ')
if c==1;
disp('Fully static CMOS
logic design style')
disp('————————————
————')
disp(' ')
if d==1;
disp(' ')
disp(' ')
[delay]=csk1fs(x)
elseif d==2;
[delay]=csk2fs(x)
elseif d==3;
[delay]=csk3fs(x)
 else;
[delay]=csk1fs(x)
[delay]=csk2fs(x)
[delay]=csk3fs(x)
end;
elseif c==2;
disp('Domino CMOS
logic design style')
disp('————————————
————')
disp(' ')
if d==1;
disp(' ')
[delay]=csk1dm(x)
elseif d==2;
[delay]=csk2dm(x)
elseif d==3;
[delay]=csk3dm(x)
else;
[delay]=csk1dm(x)
[delay]=csk2dm(x)
[delay]=csk3dm(x)
end;
elseif c==3;
disp('Complementary
pass-transistor logic
design style')
disp('————————————
————————————')
disp(' ')
if d==1;
disp(' ')
[delay]=csk1cp(x)
elseif d==2;
[delay]=csk2cp(x)
elseif d==3;
[delay]=csk3cp(x)
else;
[delay]=csk1cp(x)
[delay]=csk2cp(x)
[delay]=csk3cp(x)
end;
elseif c==4;
disp('Dual pass-
transistor logic design
style')
disp('————————————
————————————')
disp(' ')
if d==1;
disp(' ')
[delay]=csk1dp(x)
elseif d==2;
[delay]=csk2dp(x)
elseif d==3;
[delay]=csk3dp(x)
else;
[delay]=csk1dp(x)
[delay]=csk2dp(x)
[delay]=csk3dp(x)
end;
elseif c==0;
disp('Fully static CMOS
logic design style')
disp('————————————
————')
disp(' ')
if d==1;
disp(' ')
[delay]=csk1fs(x)
elseif d==2;
[delay]=csk2fs(x)
elseif d==3;
[delay]=csk3fs(x)
 else;
[delay]=csk1fs(x)
[delay]=csk2fs(x)
[delay]=csk3fs(x)
end;
disp('Domino CMOS
logic design style')
disp('————————————
————')
disp(' ')
if d==1;
disp(' ')
[delay]=csk1dm(x)
elseif d==2;
[delay]=csk2dm(x)
elseif d==3;
[delay]=csk3dm(x)
 else;
[delay]=csk1dm(x)

[delay]=csk2dm(x)
[delay]=csk3dm(x)
end;
disp('Complementary
pass-transistor logic
design style')
disp('————————————
————————————')
disp(' ')
if d==1;
disp(' ')
[delay]=csk1cp(x)

elseif d==2;
[delay]=csk2cp(x)

elseif d==3;
[delay]=csk3cp(x)
 else;
[delay]=csk1cp(x)

[delay]=csk2cp(x)
[delay]=csk3cp(x)
end;
disp('Dual pass-
transistor logic design
style')
disp('————————————
————————————')
disp(' ')
if d==1;
disp(' ')
[delay]=csk1dp(x)
elseif d==2;
[delay]=csk2dp(x)
elseif d==3;
[delay]=csk3dp(x)
 else;
[delay]=csk1dp(x)

[delay]=csk2dp(x)
[delay]=csk3dp(x)
end;
end;
elseif b==4;
disp('Carry look ahead
adder architecture')
disp('————————————
————————————')
disp(' ')
if c==1;
disp('Fully static CMOS
logic design style')
disp('————————————
————————————')
disp(' ')
if d==1;
disp(' ')
disp(' ')
[delay]=cla1fs(x)
elseif d==2;
[delay]=cla2fs(x)
elseif d==3;
[delay]=cla3fs(x)
 else;
[delay]=cla1fs(x)
[delay]=cla2fs(x)
[delay]=cla3fs(x)
end;
elseif c==2;
disp('Domino CMOS
logic design style')
disp('————————————
————')
disp(' ')
if d==1;
disp(' ')
[delay]=cla1dm(x)
elseif d==2;
[delay]=cla2dm(x)
elseif d==3;
[delay]=cla3dm(x)
 else;
[delay]=cla1dm(x)
[delay]=cla2dm(x)
[delay]=cla3dm(x)
end;
elseif c==3;
disp('Complementary
pass-transistor logic
design style')
disp('————————————
————————————')
disp(' ')
if d==1;
disp(' ')
[delay]=cla1cp(x)
elseif d==2;
[delay]=cla2cp(x)
elseif d==3;
[delay]=cla3cp(x)
 else;
[delay]=cla1cp(x)
[delay]=cla2cp(x)
[delay]=cla3cp(x)
end;
elseif c==4;
disp('Dual pass-
transistor logic design
style')
disp('————————————
————————————')
disp(' ')
if d==1;
disp(' ')
[delay]=cla1dp(x)
 elseif d==2;
[delay]=cla2dp(x)
 elseif d==3;
[delay]=cla3dp(x)
 else;
[delay]=cla1dp(x)
 [delay]=cla2dp(x)
[delay]=cla3dp(x)
end;
elseif c==0;
disp('Fully static CMOS
logic design style')
```

```
disp('------------
--------')
disp(' ')
if d==1;
disp(' ')
[delay]=cla1fs(x)
elseif d==2;
[delay]=cla2fs(x)
elseif d==3;
[delay]=cla3fs(x)
 else;
[delay]=cla1fs(x)
[delay]=cla2fs(x)
[delay]=cla3fs(x)
end;
disp('Domino CMOS
logic design style')
disp('------------
--------')
disp(' ')
if d==1;
disp(' ')
[delay]=cla1dm(x)
elseif d==2;
[delay]=cla2dm(x)
elseif d==3;
[delay]=cla3dm(x)
  else;
[delay]=cla1dm(x)
 [delay]=cla2dm(x)
[delay]=cla3dm(x)
end;
disp('Complementary
pass-transistor logic
design style')
disp('------------
----------')
disp(' ')
if d==1;
disp(' ')
[delay]=cla1cp(x)
elseif d==2;
[delay]=cla2cp(x)
elseif d==3;
[delay]=cla3cp(x)
 else;
[delay]=cla1cp(x)
[delay]=cla2cp(x)
[delay]=cla3cp(x)
end;
disp('Dual pass-
transistor logic design
style')
disp('------------
----------')
disp(' ')
if d==1;
disp(' ')
[delay]=cla1dp(x)
elseif d==2;
[delay]=cla2dp(x)
elseif d==3;
[delay]=cla3dp(x)
 else;
[delay]=cla1dp(x)
[delay]=cla2dp(x)
[delay]=cla3dp(x)
end;
end;
elseif b==5;
disp('Conditional sum
adder architecture')
disp('------------
--------')
disp(' ')
if c==1;
disp('Fully static CMOS
logic design style')
disp('------------
--------')
disp(' ')
if d==1;
disp(' ')
disp(' ')
[delay]=cnd1fs(x)
elseif d==2;
[delay]=cnd2fs(x)
elseif d==3;
[delay]=cnd3fs(x)
 else;
[delay]=cnd1fs(x)
[delay]=cnd2fs(x)
[delay]=cnd3fs(x)
end;
elseif c==2;
disp('Domino CMOS
logic design style')
disp('------------
--------')
disp(' ')
if d==1;
disp(' ')
[delay]=cnd1dm(x)

elseif d==2;
[delay]=cnd2dm(x)

elseif d==3;
[delay]=cnd3dm(x)
  else;
[delay]=cnd1dm(x)
[delay]=cnd2dm(x)
[delay]=cnd3dm(x)
end;
elseif c==3;
disp('Complementary
pass-transistor logic
design style')
disp('------------
----------')
disp(' ')
if d==1;
disp(' ')
[delay]=cnd1cp(x)
elseif d==2;
[delay]=cnd2cp(x)
elseif d==3;
[delay]=cnd3cp(x)
 else;
[delay]=cnd1cp(x)

[delay]=cnd2cp(x)
[delay]=cnd3cp(x)
        end;
elseif c==4;
disp('Dual pass-
transistor logic design
style')
disp('------------
----------')
disp(' ')
if d==1;
disp(' ')
[delay]=cnd1dp(x)
elseif d==2;
[delay]=cnd2dp(x)

elseif d==3;
[delay]=cnd3dp(x)
 else;
[delay]=cnd1dp(x)
[delay]=cnd2dp(x)
[delay]=cnd3dp(x)
end;
elseif c==0;
disp('Fully static CMOS
logic design style')
disp('------------
--------')
disp(' ')
if d==1;
disp(' ')
[delay]=cnd1fs(x)
elseif d==2;
[delay]=cnd2fs(x)
elseif d==3;
[delay]=cnd3fs(x)
 else;
[delay]=cnd1fs(x)
[delay]=cnd2fs(x)
[delay]=cnd3fs(x)
end;
disp('Domino CMOS
logic design style')
disp('------------
--------')
disp(' ')
if d==1;
disp(' ')
[delay]=cnd1dm(x)

elseif d==2;
[delay]=cnd2dm(x)
elseif d==3;
[delay]=cnd3dm(x)
  else;
[delay]=cnd1dm(x)
[delay]=cnd2dm(x)
[delay]=cnd3dm(x)
end;
disp('Complementary
pass-transistor logic
design style')
disp('------------
----------')
disp(' ')
if d==1;
disp(' ')
[delay]=cnd1cp(x)
elseif d==2;
[delay]=cnd2cp(x)
elseif d==3;
[delay]=cnd3cp(x)
 else;
[delay]=cnd1cp(x)
[delay]=cnd2cp(x)
[delay]=cnd3cp(x)
end;
disp('Dual pass-
transistor logic design
style')
disp('------------
----------')
disp(' ')
if d==1;
disp(' ')
[delay]=cnd1dp(x)
elseif d==2;
[delay]=cnd2dp(x)
elseif d==3;
[delay]=cnd3dp(x)
 else;
[delay]=cnd1dp(x)
[delay]=cnd2dp(x)
[delay]=cnd3dp(x)
end;
end;
elseif b==0;
disp('Ripple carry adder
architecture')
disp('------------
--------')
disp(' ')
if c==1;
disp('Fully static CMOS
logic design style')
disp('------------
--------')
disp(' ')
if d==1;
disp(' ')
disp(' ')
[delay]=rca1fs(x)
elseif d==2;
[delay]=rca2fs(x)
elseif d==3;
[delay]=rca3fs(x)
 else;
[delay]=rca1fs(x)
[delay]=rca2fs(x)
[delay]=rca3fs(x)
end;
elseif c==2;
disp('Domino CMOS
logic design style')
disp('------------
--------')
disp(' ')
if d==1;
disp(' ')
[delay]=rca1dm(x)
elseif d==2;
[delay]=rca2dm(x)
elseif d==3;
[delay]=rca3dm(x)
  else;
[delay]=rca1dm(x)
[delay]=rca2dm(x)
[delay]=rca3dm(x)
end;
elseif c==3;
disp('Complementary
pass-transistor logic
design style')
disp('------------
----------')
disp(' ')
if d==1;
disp(' ')
[delay]=rca1cp(x)
elseif d==2;
[delay]=rca2cp(x)
elseif d==3;
[delay]=rca3cp(x)
 else;
[delay]=rca1cp(x)
[delay]=rca2cp(x)
[delay]=rca3cp(x)
end;
elseif c==4
```

```
disp('Dual pass-
transistor logic design
style')
disp('-----------
-----------')
disp(' ')
if d==1;
disp(' ')
[delay]=rca1dp(x)
elseif d==2;
[delay]=rca2dp(x)
elseif d==3;
[delay]=rca3dp(x)
else;
[delay]=rca1dp(x)
[delay]=rca2dp(x)
[delay]=rca3dp(x)
end;
elseif c==0;
disp('Fully static CMOS
logic design style')
disp('-----------
---------')
disp(' ')
if d==1;
disp(' ')
[delay]=rca1fs(x)
elseif d==2;
[delay]=rca2fs(x)
elseif d==3;
[delay]=rca3fs(x)
 else;
[delay]=rca1fs(x)
[delay]=rca2fs(x)
[delay]=rca3fs(x)
end;
disp('Domino CMOS
logic design style')
disp('-----------
-------')
disp(' ')
if d==1;
disp(' ')
[delay]=rca1dm(x)
elseif d==2;
[delay]=rca2dm(x)
elseif d==3;
[delay]=rca3dm(x)
 else;
[delay]=rca1dm(x)
[delay]=rca2dm(x)
[delay]=rca3dm(x)
end;
disp('Complementary
pass-transistor logic
design style')
disp('-----------
-----------')
disp(' ')
if d==1;
disp(' ')
[delay]=rca1cp(x)
elseif d==2;
[delay]=rca2cp(x)
elseif d==3;
[delay]=rca3cp(x)
else;
[delay]=rca1cp(x)
[delay]=rca2cp(x)
[delay]=rca3cp(x)
end;
```

```
disp('Dual pass-
transistor logic design
style')
disp('-----------
-----------')
disp(' ')
if d==1;
disp(' ')
[delay]=rca1dp(x)
elseif d==2;
[delay]=rca2dp(x)
elseif d==3;
[delay]=rca3dp(x)
else;
[delay]=rca1dp(x)
[delay]=rca2dp(x)
[delay]=rca3dp(x)
end;
end;
disp('Carry select adder
architecture')
disp('-----------
-------')
disp(' ')
if c==1;
disp('Fully static CMOS
logic design style')
disp('-----------
---------')
disp(' ')
if d==1;
disp(' ')
disp(' ')
[delay]=crl1fs(x)
elseif d==2;
[delay]=crl2fs(x)
elseif d==3;
[delay]=crl3fs(x)
 else;
[delay]=crl1fs(x)
[delay]=crl2fs(x)
[delay]=crl3fs(x)
end;
elseif c==2;
disp('Domino CMOS
logic design style')
disp('-----------
-------')
disp(' ')
if d==1;
disp(' ')
[delay]=crl1dm(x)
elseif d==2;
[delay]=crl2dm(x)
elseif d==3;
[delay]=crl3dm(x)
else;
[delay]=crl1dm(x)
[delay]=crl2dm(x)
[delay]=crl3dm(x)
end;
elseif c==3;
disp('Complementary
pass-transistor logic
design style')
disp('-----------
-----------')
disp(' ')
if d==1;
disp(' ')
[delay]=crl1cp(x)
elseif d==2;
[delay]=crl2cp(x)
```

```
elseif d==3;
[delay]=crl3cp(x)
else;
[delay]=crl1cp(x)
[delay]=crl2cp(x)
[delay]=crl3cp(x)
end;
elseif c==4
disp('Dual pass-
transistor logic design
style')
disp('-----------
-----------')
disp(' ')
if d==1;
disp(' ')
[delay]=crl1dp(x)
elseif d==2;
[delay]=crl2dp(x)
elseif d==3;
[delay]=crl3dp(x)
else;
[delay]=crl1dp(x)
[delay]=crl2dp(x)
[delay]=crl3dp(x)
end;
elseif c==0;
disp('Fully static CMOS
logic design style')
disp('-----------
---------')
disp(' ')
if d==1;
disp(' ')
[delay]=crl1fs(x)
elseif d==2;
[delay]=crl2fs(x)
elseif d==3;
[delay]=crl3fs(x)
 else;
[delay]=crl1fs(x)
[delay]=crl2fs(x)
[delay]=crl3fs(x)
end;
disp('Domino CMOS
logic design style')
disp('-----------
-------')
disp(' ')
if d==1;
disp(' ')
[delay]=crl1dm(x)
elseif d==2;
[delay]=crl2dm(x)
elseif d==3;
[delay]=crl3dm(x)
else;
[delay]=crl1dm(x)
[delay]=crl2dm(x)
[delay]=crl3dm(x)
end;
disp('Complementary
pass-transistor logic
design style')
disp('-----------
-----------')
disp(' ')
if d==1;
disp(' ')
[delay]=crl1cp(x)
elseif d==2;
[delay]=crl2cp(x)
elseif d==3;
[delay]=crl3cp(x)
else;
[delay]=crl1cp(x)
[delay]=crl2cp(x)
[delay]=crl3cp(x)
end;
disp('Dual pass-
transistor logic design
style')
disp('-----------
-----------')
disp(' ')
if d==1;
disp(' ')
[delay]=crl1dp(x)
elseif d==2;
[delay]=crl2dp(x)
elseif d==3;
[delay]=crl3dp(x)
else;
[delay]=crl1dp(x)

[delay]=crl2dp(x)
[delay]=crl3dp(x)
end;
end;
disp('Carry skip adder
architecture')
disp('-----------
-------')
disp(' ')
if c==1;
disp('Fully static CMOS
logic design style')
disp('-----------
---------')
disp(' ')
if d==1;
disp(' ')
disp(' ')
[delay]=csk1fs(x)
elseif d==2;
[delay]=csk2fs(x)
elseif d==3;
[delay]=csk3fs(x)
 else;
[delay]=csk1fs(x)
[delay]=csk2fs(x)
[delay]=csk3fs(x)
end;
elseif c==2;
disp('Domino CMOS
logic design style')
disp('-----------
-------')
disp(' ')
if d==1;
disp(' ')
[delay]=csk1dm(x)
elseif d==2;
[delay]=csk2dm(x)
elseif d==3;
[delay]=csk3dm(x)
else;
[delay]=csk1dm(x)
[delay]=csk2dm(x)
[delay]=csk3dm(x)
end;
elseif c==3;
disp('Complementary
pass-transistor logic
design style')
```

```
disp('------------------------------------------------------------')
disp(' ')
if d==1;
disp(' ')
[delay]=csk1cp(x)
elseif d==2;
[delay]=csk2cp(x)
elseif d==3;
[delay]=csk3cp(x)
else;
[delay]=csk1cp(x)
[delay]=csk2cp(x)
[delay]=csk3cp(x)
end;
elseif c==4;
disp('Dual pass-transistor logic design style')
disp('------------------------------------------------------------')
disp(' ')
if d==1;
disp(' ')
[delay]=csk1dp(x)
elseif d==2;
[delay]=csk2dp(x)
elseif d==3;
[delay]=csk3dp(x)

else;
[delay]=csk1dp(x)
[delay]=csk2dp(x)
[delay]=csk3dp(x)
end;
elseif c==0;
disp('Fully static CMOS logic design style')
disp('------------------------------------------------------------')
disp(' ')
if d==1;
disp(' ')
[delay]=csk1fs(x)
elseif d==2;
[delay]=csk2fs(x)
elseif d==3;
[delay]=csk3fs(x)
 else;
[delay]=csk1fs(x)
[delay]=csk2fs(x)
[delay]=csk3fs(x)
end;
disp('Domino CMOS logic design style')
disp('------------------------------------------------------------')
disp(' ')
if d==1;
disp(' ')
[delay]=csk1dm(x)
elseif d==2;
[delay]=csk2dm(x)
elseif d==3;
[delay]=csk3dm(x)
else;
[delay]=csk1dm(x)
[delay]=csk2dm(x)
[delay]=csk3dm(x)
end;
disp('Complementary pass-transistor logic design style')
disp('------------------------------------------------------------')
disp(' ')
if d==1;
disp(' ')
[delay]=csk1cp(x)
elseif d==2;
[delay]=csk2cp(x)

elseif d==3;
[delay]=csk3cp(x)
else;
[delay]=csk1cp(x)
[delay]=csk2cp(x)
[delay]=csk3cp(x)
end;
disp('Dual pass-transistor logic design style')
disp('------------------------------------------------------------')
disp(' ')
if d==1;
disp(' ')
[delay]=csk1dp(x)
elseif d==2;
[delay]=csk2dp(x)
elseif d==3;
[delay]=csk3dp(x)
else;
[delay]=csk1dp(x)
[delay]=csk2dp(x)
[delay]=csk3dp(x)
end;
end;
disp('Carry look ahead adder architecture')
disp('------------------------------------------------------------')
disp(' ')
if c==1;
disp('Fully static CMOS logic design style')
disp('------------------------------------------------------------')
disp(' ')
if d==1;
disp(' ')
disp(' ')
[delay]=cla1fs(x)
elseif d==2;
[delay]=cla2fs(x)
elseif d==3;
[delay]=cla3fs(x)
 else;
[delay]=cla1fs(x)
[delay]=cla2fs(x)
[delay]=cla3fs(x)
end;
elseif c==2;
disp('Domino CMOS logic design style')
disp('------------------------------------------------------------')
disp(' ')
if d==1;
disp(' ')
[delay]=cla1dm(x)
elseif d==2;
[delay]=cla2dm(x)
elseif d==3;
[delay]=cla3dm(x)
else;
[delay]=cla1dm(x)

[delay]=cla2dm(x)
[delay]=cla3dm(x)
end;
elseif c==3;
disp('Complementary pass-transistor logic design style')
disp('------------------------------------------------------------')
disp(' ')
if d==1;
disp(' ')
[delay]=cla1cp(x)
elseif d==2;
[delay]=cla2cp(x)
elseif d==3;
[delay]=cla3cp(x)
else;
[delay]=cla1cp(x)
[delay]=cla2cp(x)
[delay]=cla3cp(x)
end;
elseif c==4;
disp('Dual pass-transistor logic design style')
disp('------------------------------------------------------------')
disp(' ')
if d==1;
disp(' ')
[delay]=cla1dp(x)
elseif d==2;
[delay]=cla2dp(x)
elseif d==3;
[delay]=cla3dp(x)
else;
[delay]=cla1dp(x)
[delay]=cla2dp(x)
[delay]=cla3dp(x)
end;
elseif c==0;
disp('Fully static CMOS logic design style')
disp('------------------------------------------------------------')
disp(' ')
if d==1;
disp(' ')
[delay]=cla1fs(x)
elseif d==2;
[delay]=cla2fs(x)
elseif d==3;
[delay]=cla3fs(x)
 else;
[delay]=cla1fs(x)
[delay]=cla2fs(x)
[delay]=cla3fs(x)
end;
disp('Domino CMOS logic design style')
disp('------------------------------------------------------------')
disp(' ')
if d==1;
disp(' ')
[delay]=cla1dm(x)
elseif d==2;
[delay]=cla2dm(x)
elseif d==3;
[delay]=cla3dm(x)
else;
[delay]=cla1dm(x)
[delay]=cla2dm(x)
[delay]=cla3dm(x)
end;
disp('Complementary pass-transistor logic design style')
disp('------------------------------------------------------------')
disp(' ')
if d==1;
disp(' ')
[delay]=cla1cp(x)
elseif d==2;
[delay]=cla2cp(x)
elseif d==3;
[delay]=cla3cp(x)
else;
[delay]=cla1cp(x)
[delay]=cla2cp(x)
[delay]=cla3cp(x)
end;
disp('Dual pass-transistor logic design style')
disp('------------------------------------------------------------')
disp(' ')
if d==1;
disp(' ')
[delay]=cla1dp(x)
elseif d==2;
[delay]=cla2dp(x)
elseif d==3;
[delay]=cla3dp(x)
else;
[delay]=cla1dp(x)
[delay]=cla2dp(x)
[delay]=cla3dp(x)
end;
end;
disp('Conditional sum adder architecture')
disp('------------------------------------------------------------')
disp(' ')
if c==1;
disp('Fully static CMOS logic design style')
disp('------------------------------------------------------------')
disp(' ')
if d==1;
disp(' ')
disp(' ')
[delay]=cnd1fs(x)
elseif d==2;
[delay]=cnd2fs(x)
elseif d==3;
[delay]=cnd3fs(x)
 else;
[delay]=cnd1fs(x)
[delay]=cnd2fs(x)
[delay]=cnd3fs(x)
end;
elseif c==2;
disp('Domino CMOS logic design style')
disp('------------------------------------------------------------')
disp(' ')
if d==1;
```

```
disp(' ')
[delay]=cnd1dm(x)
elseif d==2;
[delay]=cnd2dm(x)
elseif d==3;
[delay]=cnd3dm(x)
else;
[delay]=cnd1dm(x)
[delay]=cnd2dm(x)
[delay]=cnd3dm(x)
end;
elseif c==3;
disp('Complementary
pass-transistor logic
design style')
disp('------------
------------ ')
disp(' ')
if d==1;
disp(' ')
[delay]=cnd1cp(x)
elseif d==2;
[delay]=cnd2cp(x)
elseif d==3;
[delay]=cnd3cp(x)
 else;
[delay]=cnd1cp(x)
[delay]=cnd2cp(x)
[delay]=cnd3cp(x)
        end;
elseif c==4;
disp('Dual pass-
transistor logic design
style')
disp('------------
------------ ')
disp(' ')
if d==1;
disp(' ')
[delay]=cnd1dp(x)
elseif d==2;
[delay]=cnd2dp(x)
elseif d==3;
[delay]=cnd3dp(x)
else;
[delay]=cnd1dp(x)
[delay]=cnd2dp(x)
[delay]=cnd3dp(x)
end;
elseif c==0;
disp('Fully static CMOS
logic design style')
disp('------------
------------ ')
disp(' ')
if d==1;
disp(' ')
[delay]=cnd1fs(x)
elseif d==2;
[delay]=cnd2fs(x)
elseif d==3;
[delay]=cnd3fs(x)
 else;
[delay]=cnd1fs(x)
[delay]=cnd2fs(x)
[delay]=cnd3fs(x)
end;
disp('Domino CMOS
logic design style')
disp('------------
------------ ')
disp(' ')
if d==1;
disp(' ')
[delay]=cnd1dm(x)
elseif d==2;
[delay]=cnd2dm(x)
elseif d==3;
[delay]=cnd3dm(x)
 else;
[delay]=cnd1dm(x)
[delay]=cnd2dm(x)
[delay]=cnd3dm(x)
end;
disp('Complementary
pass-transistor logic
design style')
disp('------------
------------ ')
disp(' ')
if d==1;
disp(' ')
[delay]=cnd1cp(x)
elseif d==2;
[delay]=cnd2cp(x)
elseif d==3;
[delay]=cnd3cp(x)
 else;
[delay]=cnd1cp(x)
[delay]=cnd2cp(x)
[delay]=cnd3cp(x)
end;
disp('Dual pass-
transistor logic design
style')
disp('------------
------------ ')
disp(' ')
if d==1;
disp(' ')
[delay]=cnd1dp(x)
elseif d==2;
[delay]=cnd2dp(x)
elseif d==3;
[delay]=cnd3dp(x)
else;
[delay]=cnd1dp(x)
[delay]=cnd2dp(x)
[delay]=cnd3dp(x)
end;
end;
end;
```

FUNCTION 'EP'

This function is called in ***Appendix A1***. It takes the adder design as input and calculates the energy consumption per addition for the given adder

```
function[energy]=ep(x,b
,c,d);
if b==1;
disp('Ripple carry adder
architecture')
disp('------------
------------ ')
disp(' ')
if c==1;
disp('Fully static CMOS
logic design style')
disp('------------ ')
disp(' ')
if d==1;
disp(' ')
disp(' ')
[energy]=rcae1fs(x)
elseif d==2;
[energy]=rcae2fs(x)
elseif d==3;
[energy]=rcae3fs(x)
else;
[energy]=rcae1fs(x)
[energy]=rcae2fs(x)
[energy]=rcae3fs(x)
end;
elseif c==2;
disp('Domino CMOS
logic design style')
disp('------------ ')
disp(' ')
if d==1;
disp(' ')
[energy]=rcae1dm(x)
elseif d==2;
[energy]=rcae2dm(x)
elseif d==3;
[energy]=rcae3dm(x)
else;
[energy]=rcae1dm(x)
 [energy]=rcae2dm(x)
[energy]=rcae3dm(x)
end;
elseif c==3;
disp('Complementary
pass-transistor logic
design style')
disp('------------ ')
disp(' ')
if d==1;
disp(' ')
[energy]=rcae1cp(x)
elseif d==2;
[energy]=rcae2cp(x)
elseif d==3;
[energy]=rcae3cp(x)
else;
[energy]=rcae1cp(x)
 [energy]=rcae2cp(x)
[energy]=rcae3cp(x)
end;
elseif c==4;
disp('Dual pass-
transistor logic design
style')
disp('------------ ')
disp(' ')
if d==1;
disp(' ')
[energy]=rcae1dp(x)
elseif d==2;
[energy]=rcae2dp(x)
elseif d==3;
[energy]=rcae3dp(x)
else;
[energy]=rcae1dp(x)
 [energy]=rcae2dp(x)
[energy]=rcae3dp(x)
end;
elseif c==0;
disp('Fully static CMOS
logic design style')
disp('------------ ')
disp(' ')
if d==1;
disp(' ')
[energy]=rcae1fs(x)
elseif d==2;
[energy]=rcae2fs(x)
elseif d==3;
[energy]=rcae3fs(x)
else;
[energy]=rcae1fs(x)
[energy]=rcae2fs(x)
[energy]=rcae3fs(x)
end;
disp('Domino CMOS
logic design style')
disp('------------ ')
disp(' ')
if d==1;
disp(' ')
[energy]=rcae1dm(x)
elseif d==2;
[energy]=rcae2dm(x)
elseif d==3;
[energy]=rcae3dm(x)
else;
[energy]=rcae1dm(x)
 [energy]=rcae2dm(x)
[energy]=rcae3dm(x)
end;
disp('Complementary
pass-transistor logic
design style')
disp('------------ ')
disp(' ')
if d==1;
disp(' ')
[energy]=rcae1cp(x)
elseif d==2;
[energy]=rcae2cp(x)
```

```
elseif d==3;
[energy]=rcae3cp(x)
else;
[energy]=rcae1cp(x)
 [energy]=rcae2cp(x)
[energy]=rcae3cp(x)
end;
disp('Dual pass-
transistor logic design
style')
disp('————————— ')
disp(' ')
if d==1;
disp(' ')
[energy]=rcae1dp(x)
elseif d==2;
[energy]=rcae2dp(x)
elseif d==3;
[energy]=rcae3dp(x)
else;
[energy]=rcae1dp(x)
 [energy]=rcae2dp(x)
[energy]=rcae3dp(x)
end;
end;
elseif b==2;
disp('Carry select adder
architecture')
disp('————————————
—— ')
disp(' ')
if c==1;
disp('Fully static CMOS
logic design style')
disp('————————— ')
disp(' ')
if d==1;
disp(' ')
disp(' ')
[energy]=crle1fs(x)
elseif d==2;
[energy]=crle2fs(x)
elseif d==3;
[energy]=crle3fs(x)
else;
[energy]=crle1fs(x)
[energy]=crle2fs(x)
[energy]=crle3fs(x)
end;
elseif c==2;
disp('Domino CMOS
logic design style')
disp('————————— ')
disp(' ')
if d==1;
disp(' ')
[energy]=crle1dm(x)
elseif d==2;
[energy]=crle2dm(x)
elseif d==3;
[energy]=crle3dm(x)
else;
[energy]=crle1dm(x)
 [energy]=crle2dm(x)
[energy]=crle3dm(x)
end;
elseif c==3;
disp('Complementary
pass-transistor logic
design style')
disp('————————— ')
disp(' ')
if d==1;
disp(' ')
[energy]=crle1cp(x)
elseif d==2;
[energy]=crle2cp(x)
elseif d==3;
[energy]=crle3cp(x)
else;
[energy]=crle1cp(x)
 [energy]=crle2cp(x)
[energy]=crle3cp(x)
end;
elseif c==4;
disp('Dual pass-
transistor logic design
style')
disp('————————— ')
disp(' ')
if d==1;
disp(' ')
[energy]=crle1dp(x)
elseif d==2;
[energy]=crle2dp(x)
elseif d==3;
[energy]=crle3dp(x)
else;
[energy]=crle1dp(x)
 [energy]=crle2dp(x)
[energy]=crle3dp(x)
end;
elseif c==0;
disp('Fully static CMOS
logic design style')
disp('————————— ')
disp(' ')
if d==1;
disp(' ')
[energy]=crle1fs(x)
elseif d==2;
[energy]=crle2fs(x)
elseif d==3;
[energy]=crle3fs(x)
else;
[energy]=crle1fs(x)
[energy]=crle2fs(x)
[energy]=crle3fs(x)
end;
disp('Domino CMOS
logic design style')
disp('————————— ')
disp(' ')
if d==1;
disp(' ')
[energy]=crledm(x)
elseif d==2;
[energy]=crle2dm(x)
elseif d==3;
[energy]=crle3dm(x)
else;
[energy]=crle1dm(x)
 [energy]=crle2dm(x)
[energy]=crle3dm(x)
end;
disp('Complementary
pass-transistor logic
design style')
disp('————————— ')
disp(' ')
if d==1;
disp(' ')
[energy]=crle1cp(x)
elseif d==2;
[energy]=crle2cp(x)
elseif d==3;
[energy]=crle3cp(x)
else;
[energy]=crle1cp(x)
 [energy]=crle2cp(x)
[energy]=crle3cp(x)
end;
disp('Dual pass-
transistor logic design
style')
disp('————————— ')
disp(' ')
if d==1;
disp(' ')
[energy]=crle1dp(x)
elseif d==2;
[energy]=crle2dp(x)
elseif d==3;
[energy]=crle3dp(x)
else;
[energy]=crle1dp(x)
 [energy]=crle2dp(x)
[energy]=crle3dp(x)
end;
end;
elseif b==3;
disp('Carry skip adder
architecture')
disp('————————— ')
disp(' ')
if c==1;
disp('Fully static CMOS
logic design style')
disp('————————— ')
disp(' ')
if d==1;
disp(' ')
disp(' ')
[energy]=cske1fs(x)
elseif d==2;
[energy]=cske2fs(x)
elseif d==3;
[energy]=cske3fs(x)
else;
[energy]=csk1fs(x)
[energy]=csk2fs(x)
[energy]=csk3fs(x)
end;
elseif c==2;
disp('Domino CMOS
logic design style')
disp('————————— ')
disp(' ')
if d==1;
disp(' ')
[energy]=cske1dm(x)
elseif d==2;
[energy]=cske2dm(x)
elseif d==3;
[energy]=cske3dm(x)
else;
[energy]=cske1dm(x)
 [energy]=cske2dm(x)
[energy]=cske3dm(x)
end;
elseif c==3;
disp('Complementary
pass-transistor logic
design style')
disp('————————— ')
disp(' ')
if d==1;
disp(' ')
[energy]=cske1cp(x)
elseif d==2;
[energy]=cske2cp(x)
elseif d==3;
[energy]=cske3cp(x)
else;
[energy]=cske1cp(x)
 [energy]=cske2cp(x)
[energy]=cske3cp(x)
end;
elseif c==4;
disp('Dual pass-
transistor logic design
style')
disp('————————— ')
disp(' ')
if d==1;
disp(' ')
[energy]=cske1dp(x)
elseif d==2;
[energy]=cske2dp(x)
elseif d==3;
[energy]=cske3dp(x)
else;
[energy]=cske1dp(x)
 [energy]=cske2dp(x)
[energy]=cske3dp(x)
end;
elseif c==0;
disp('Fully static CMOS
logic design style')
disp('————————————
—— ')
disp(' ')
if d==1;
disp(' ')
[energy]=cske1fs(x)
elseif d==2;
[energy]=cske2fs(x)
elseif d==3;
[energy]=cske3fs(x)
else;
[energy]=cske1fs(x)
[energy]=cske2fs(x)
[energy]=cske3fs(x)
end;
disp('Domino CMOS
logic design style')
disp('————————— ')
disp(' ')
if d==1;
disp(' ')
[energy]=cske1dm(x)
elseif d==2;
[energy]=cske2dm(x)
elseif d==3;
[energy]=cske3dm(x)
else;
[energy]=cske1dm(x)
[energy]=cske2dm(x)
[energy]=cske3dm(x)
end;
disp('Complementary
pass-transistor logic
design style')
disp('————————— ')
disp(' ')
if d==1;
disp(' ')
[energy]=cske1cp(x)
elseif d==2;
[energy]=cske2cp(x)
elseif d==3;
[energy]=cske3cp(x)
```

```
else;
[energy]=cske1cp(x)
 [energy]=cske2cp(x)
[energy]=cske3cp(x)
end;
disp('Dual pass-
transistor logic design
style')
disp('---------- ')
disp(' ')
if d==1;
disp(' ')
[energy]=cske1dp(x)
elseif d==2;
[energy]=cske2dp(x)
elseif d==3;
[energy]=cske3dp(x)
else;
[energy]=cske1dp(x)
 [energy]=cske2dp(x)
[energy]=cske3dp(x)
end;
end;
elseif b==4;
disp('Carry look ahead
adder architecture')
disp('----------
---------- ')
disp(' ')
if c==1;
disp('Fully static CMOS
logic design style')
disp('---------- ')
disp(' ')
if d==1;
disp(' ')
disp(' ')
[energy]=clae1fs(x)
elseif d==2;
[energy]=clae2fs(x)
elseif d==3;
[energy]=clae3fs(x)
else;
[energy]=clae1fs(x)
[energy]=clae2fs(x)
[energy]=clae3fs(x)
end;
elseif c==2;
disp('Domino CMOS
logic design style')
disp('---------- ')
disp(' ')
if d==1;
disp(' ')
[energy]=clae1dm(x)
elseif d==2;
[energy]=clae2dm(x)
elseif d==3;
[energy]=clae3dm(x)
else;
[energy]=clae1dm(x)
 [energy]=clae2dm(x)
[energy]=clae3dm(x)
end;
elseif c==3;
disp('Complementary
pass-transistor logic
design style')
disp('---------- ')
disp(' ')
if d==1;
disp(' ')
[energy]=clae1cp(x)
elseif d==2;
[energy]=clae2cp(x)
elseif d==3;
[energy]=clae3cp(x)
else;
[energy]=clae1cp(x)
 [energy]=clae2cp(x)
[energy]=clae3cp(x)
end;
elseif c==4;
disp('Dual pass-
transistor logic design
style')
disp('---------- ')
disp(' ')
if d==1;
disp(' ')
[energy]=clae1dp(x)
elseif d==2;
[energy]=clae2dp(x)
elseif d==3;
[energy]=clae3dp(x)
 else;
[energy]=clae1dp(x)
[energy]=clae2dp(x)
[energy]=clae3dp(x)
end;
elseif c==0;
disp('Fully static CMOS
logic design style')
disp('---------- ')
disp(' ')
if d==1;
disp(' ')
[energy]=clae1fs(x)
elseif d==2;
[energy]=clae2fs(x)
elseif d==3;
[energy]=clae3fs(x)
else;
[energy]=clae1fs(x)
[energy]=clae2fs(x)
[energy]=clae3fs(x)
end;
disp('Domino CMOS
logic design style')
disp('---------- ')
disp(' ')
if d==1;
disp(' ')
[energy]=clae1dm(x)

elseif d==2;
[energy]=clae2dm(x)
elseif d==3;
[energy]=clae3dm(x)
else;
[energy]=clae1dm(x)
 [energy]=clae2dm(x)
[energy]=clae3dm(x)
end;
disp('Complementary
pass-transistor logic
design style')
disp('---------- ')
disp(' ')
if d==1;
disp(' ')
[energy]=clae1cp(x)
elseif d==2;
[energy]=clae2cp(x)
elseif d==3;
[energy]=clae3cp(x)
else;
[energy]=clae1cp(x)
 [energy]=clae2cp(x)
[energy]=clae3cp(x)
end;
disp('Dual pass-
transistor logic design
style')
disp('---------- ')
disp(' ')
if d==1;
disp(' ')
[energy]=clae1dp(x)
elseif d==2;
[energy]=clae2dp(x)
elseif d==3;
[energy]=clae3dp(x)
else;
[energy]=clae1dp(x)
 [energy]=clae2dp(x)
[energy]=clae3dp(x)
end;
end;
elseif b==5;
disp('Conditional sum
adder architecture')
disp('---------- ')
disp(' ')
if c==1;
disp('Fully static CMOS
logic design style')
disp('---------- ')
disp(' ')
if d==1;
disp(' ')
disp(' ')
[energy]=cnde1fs(x)
elseif d==2;
[energy]=cnde2fs(x)
elseif d==3;
[energy]=cnde3fs(x)
else;
[energy]=cnde1fs(x)
[energy]=cnde2fs(x)
[energy]=cnde3fs(x)
end;
elseif c==2;
disp('Domino CMOS
logic design style')
disp('---------- ')
disp(' ')
if d==1;
disp(' ')
[energy]=cnde1dm(x)
elseif d==2;
[energy]=cnde2dm(x)
elseif d==3;
[energy]=cnde3dm(x)
else;
[energy]=cnde1dm(x)
 [energy]=cnde2dm(x)
[energy]=cnde3dm(x)
end;
elseif c==3;
disp('Complementary
pass-transistor logic
design style')
disp('---------- ')
disp(' ')
if d==1;
disp(' ')
[energy]=cnde1cp(x)
elseif d==2;
[energy]=cnde2cp(x)
elseif d==3;
[energy]=cnde3cp(x)
else;
[energy]=cnde1cp(x)
 [energy]=cnde2cp(x)
[energy]=cnde3cp(x)
end;
elseif c==4;
disp('Dual pass-
transistor logic design
style')
disp('---------- ')
disp(' ')
if d==1;
disp(' ')
[energy]=cnde1dp(x)
elseif d==2;
[energy]=cnde2dp(x)
elseif d==3;
[energy]=cnde3dp(x)
else;
[energy]=cnde1dp(x)
 [energy]=cnde2dp(x)
[energy]=cnde3dp(x)
end;
elseif c==0;
disp('Fully static CMOS
logic design style')
disp('---------- ')
disp(' ')
if d==1;
disp(' ')
[energy]=cnde1fs(x)
elseif d==2;
[energy]=cnde2fs(x)
elseif d==3;
[energy]=cnde3fs(x)
else;
[energy]=cnde1fs(x)
[energy]=cnde2fs(x)
[energy]=cnde3fs(x)
end;
disp('Domino CMOS
logic design style')
disp('---------- ')
disp(' ')
if d==1;
disp(' ')
[energy]=cnde1dm(x)
elseif d==2;
[energy]=cnde2dm(x)
elseif d==3;
[energy]=cnde3dm(x)
else;
[energy]=cnde1dm(x)
 [energy]=cnde2dm(x)
[energy]=cnde3dm(x)
end;
disp('Complementary
pass-transistor logic
design style')
disp('----------
---------- ')
disp(' ')
if d==1;
disp(' ')
[energy]=cnde1cp(x)
elseif d==2;
[energy]=cnde2cp(x)
elseif d==3;
[energy]=cnde3cp(x)
else;
```

```
[energy]=cnde1cp(x)
[energy]=cnde2cp(x)
[energy]=cnde3cp(x)
end;
disp('Dual pass-
transistor logic design
style')
disp('————————')
disp(' ')
if d==1;
disp(' ')
[energy]=cnde1dp(x)
elseif d==2;
[energy]=cnde2dp(x)
elseif d==3;
[energy]=cnde3dp(x)
else;
[energy]=cnde1dp(x)
[energy]=cnde2dp(x)
[energy]=cnde3dp(x)
end;
end;
elseif b==0;
disp('Ripple carry adder
architecture')
disp('————————')
disp(' ')
if c==1;
disp('Fully static CMOS
logic design style')
disp('————————')
disp(' ')
if d==1;
disp(' ')
disp(' ')
[energy]=rcae1fs(x)
elseif d==2;
[energy]=rcae2fs(x)
elseif d==3;
[energy]=rcae3fs(x)
else;
[energy]=rcae1fs(x)
[energy]=rcae2fs(x)
[energy]=rcae3fs(x)
end;
elseif c==2;
disp('Domino CMOS
logic design style')
disp('————————')
disp(' ')
if d==1;
disp(' ')
[energy]=rcae1dom(x)
elseif d==2;
[energy]=rcae2dm(x)
elseif d==3;
[energy]=rcae3dm(x)
else;
[energy]=rcae1dm(x)
[energy]=rcae2dm(x)
[energy]=rcae3dm(x)
end;
elseif c==3;
disp('Complementary
pass-transistor logic
design style')
disp('————————')
disp(' ')
if d==1;
disp(' ')
[energy]=rcae1cp(x)
elseif d==2;
[energy]=rcae2cp(x)
elseif d==3;
[energy]=rcae3cp(x)
else;
[energy]=rcae1cp(x)
[energy]=rcae2cp(x)
[energy]=rcae3cp(x)
end;
disp('Dual pass-
transistor logic design
style')
disp('————————')
disp(' ')
if d==1;
disp(' ')
[energy]=rcae1dp(x)
elseif d==2;
[energy]=rcae2dp(x)

elseif d==3;
[energy]=rcae3dp(x)
else;
[energy]=rcae1dp(x)
[energy]=rcae2dp(x)
[energy]=rcae3dp(x)
end;
end;
disp('Carry select adder
architecture')
disp('————————')
disp(' ')
if c==1;
disp('Fully static CMOS
logic design style')
disp('————————')
disp(' ')
if d==1;
disp(' ')
disp(' ')
[energy]=crle1fs(x)
elseif d==2;
[energy]=crle2fs(x)
elseif d==3;
[energy]=crle3fs(x)
else;
[energy]=crle1fs(x)
[energy]=crle2fs(x)
[energy]=crle3fs(x)
end;
elseif c==2;
disp('Domino CMOS
logic design style')
disp('————————
————')
disp(' ')
if d==1;
disp(' ')
[energy]=crle1dm(x)
elseif d==2;
[energy]=crle2dm(x)
elseif d==3;
[energy]=crle3dm(x)
else;
[energy]=crle1dm(x)
[energy]=crle2dm(x)
[energy]=crle3dm(x)
end;
elseif c==3;
disp('Complementary
pass-transistor logic
design style')
disp('————————')
disp(' ')
if d==1;
disp(' ')
[energy]=crle1cp(x)
elseif d==2;
[energy]=crle2cp(x)
elseif d==3;
[energy]=crle3cp(x)
else;
[energy]=crle1cp(x)
[energy]=crle2cp(x)
[energy]=crle3cp(x)
end;
elseif c==4;
disp('Dual pass-
transistor logic design
style')
disp('————————')
disp(' ')
if d==1;
disp(' ')
[energy]=crle1dp(x)
elseif d==2;
[energy]=crle2dp(x)
elseif d==3;
[energy]=crle3dp(x)
else;
[energy]=crle1dp(x)
[energy]=crle2dp(x)
[energy]=crle3dp(x)
end;
disp('Fully static CMOS
logic design style')
disp('————————')
disp(' ')
if d==1;
disp(' ')
[energy]=crle1fs(x)
elseif d==2;
[energy]=crle2fs(x)
elseif d==3;
[energy]=crle3fs(x)
else;
[energy]=crle1fs(x)
[energy]=crle2fs(x)
[energy]=crle3fs(x)
end;
disp('Domino CMOS
logic design style')
disp('————————')
disp(' ')
if d==1;
disp(' ')
[energy]=crle1dm(x)
elseif d==2;
[energy]=crle2dm(x)
elseif d==3;
[energy]=crle3dm(x)
else;
[energy]=crle1dm(x)
[energy]=crle2dm(x)
[energy]=crle3dm(x)
end;
disp('Complementary
pass-transistor logic
design style')
disp('————————')
disp(' ')
if d==1;
disp(' ')
[energy]=crle1cp(x)
elseif d==2;
[energy]=crle2cp(x)

elseif d==3;
[energy]=crle3cp(x)
else;
[energy]=crle1cp(x)
[energy]=crle2cp(x)
```

```
[energy]=crlc3cp(x)
end;
disp('Dual pass-
transistor logic design
style')
disp('————————— ')
disp(' ')
if d==1;
disp(' ')
[energy]=crlc1dp(x)
elseif d==2;
[energy]=crlc2dp(x)
elseif d==3;
[energy]=crlc3dp(x)
 else;
[energy]=crlc1dp(x)
 [energy]=crlc2dp(x)
[energy]=crlc3dp(x)
end;
 end;
disp('Carry skip adder
architecture')
disp('————————— ')
disp(' ')
if c==1;
disp('Fully static CMOS
logic design style')
disp('————————— ')
disp(' ')
if d==1;
disp(' ')
disp(' ')
[energy]=cskc1fs(x)
elseif d==2;
[energy]=cskc2fs(x)
elseif d==3;
[energy]=cskc3fs(x)
else;
[energy]=cskc1fs(x)
[energy]=cskc2fs(x)
[energy]=cskc3fs(x)
end;
elseif c==2;
disp('Domino CMOS
logic design style')
disp('————————— ')
disp(' ')
if d==1;
disp(' ')
[energy]=cskc1dm(x)
elseif d==2;
[energy]=cskc2dm(x)
elseif d==3;
[energy]=cskc3dm(x)
else;
[energy]=cskc1dm(x)
 [energy]=cskc2dm(x)
[energy]=cskc3dm(x)
end;
elseif c==3;
disp('Complementary
pass-transistor logic
design style')
disp('————————— ')
disp(' ')
if d==1;
disp(' ')
[energy]=cskc1cp(x)
elseif d==2;
[energy]=cskc2cp(x)
elseif d==3;
[energy]=cskc3cp(x)
else;
[energy]=cskc1cp(x)
 [energy]=cskc2cp(x)
[energy]=cskc3cp(x)
end;
[energy]=cskc1cp(x)
 [energy]=cskc2cp(x)
[energy]=cskc3cp(x)
end;
disp('Dual pass-
transistor logic design
style')
disp('————————— ')
disp(' ')
if d==1;
disp(' ')
[energy]=cskc1dp(x)
elseif d==2;
[energy]=cskc2dp(x)
elseif d==3;
[energy]=cskc3dp(x)
 else;
[energy]=cskc1dp(x)
[energy]=cskc2dp(x)
[energy]=cskc3dp(x)
end;
end;
disp('Carry look ahead
adder architecture')
disp('————————— ')
disp(' ')
if c==1;
disp('Fully static CMOS
logic design style')
disp('————————— ')
disp(' ')
if d==1;
disp(' ')
disp(' ')
[energy]=clac1fs(x)
elseif d==2;
[energy]=clac2fs(x)
elseif d==3;
[energy]=clac3fs(x)
else;
[energy]=clac1fs(x)
[energy]=clac2fs(x)
[energy]=clac3fs(x)
end;
elseif c==2;
disp('Domino CMOS
logic design style')
disp('————————— ')
disp(' ')
if d==1;
disp(' ')
[energy]=clac1dm(x)
elseif d==2;
[energy]=clac2dm(x)
elseif d==3;
[energy]=clac3dm(x)
else;
[energy]=clac1dm(x)
 [energy]=clac2dm(x)
[energy]=clac3dm(x)
end;
elseif c==3;
disp('Complementary
pass-transistor logic
design style')
disp('————————— ')
disp(' ')
if d==1;
disp(' ')
[energy]=clac1cp(x)
elseif d==2;
[energy]=clac2cp(x)
elseif d==3;
[energy]=clac3cp(x)
 else;
[energy]=clac1cp(x)
[energy]=clac2cp(x)
[energy]=clac3cp(x)
end;
elseif c==4;
disp('Dual pass-
transistor logic design
style')
disp('————————— ')
disp(' ')
if d==1;
disp(' ')
[energy]=clac1dp(x)
elseif d==2;
[energy]=clac2dp(x)
elseif d==3;
[energy]=clac3dp(x)
else;
[energy]=clac1dp(x)
 [energy]=clac2dp(x)
[energy]=clac3dp(x)
end;
elseif c==0;
disp('Fully static CMOS
logic design style')
disp('————————— ')
disp(' ')
if d==1;
disp(' ')
[energy]=clac1fs(x)
elseif d==2;
[energy]=clac2fs(x)
elseif d==3;
[energy]=clac3fs(x)
else;
[energy]=clac1fs(x)
[energy]=clac2fs(x)
[energy]=clac3fs(x)
end;
disp('Domino CMOS
logic design style')
disp('————————— ')
disp(' ')
if d==1;
disp(' ')
[energy]=clac1dm(x)
elseif d==2;
[energy]=clac2dm(x)
elseif d==3;
[energy]=clac3dm(x)
else;
[energy]=clac1dm(x)
 [energy]=clac2dm(x)
[energy]=clac3dm(x)
end;
disp('Complementary
pass-transistor logic
design style')
disp('————————— ')
disp(' ')
if d==1;
disp(' ')
[energy]=clac1cp(x)
elseif d==2;
[energy]=clac2cp(x)
elseif d==3;
[energy]=clac3cp(x)
else;
[energy]=clac1cp(x)
 [energy]=clac2cp(x)
[energy]=clac3cp(x)
end;
disp('Dual pass-
transistor logic design
style')
```

```
disp('————————— ')
disp(' ')
if d==1;
disp(' ')
[energy]=clac1dp(x)
elseif d==2;
[energy]=clac2dp(x)
elseif d==3;
[energy]=clac3dp(x)
else;
[energy]=clac1dp(x)
 [energy]=clac2dp(x)
[energy]=clac3dp(x)
end;
end;
disp('Conditional sum
adder architecture')
disp('————————— ')
disp(' ')
if c==1;
disp('Fully static CMOS
logic design style')
disp('————————— ')
disp(' ')
if d==1;
disp(' ')
disp(' ')
[energy]=cnde1fs(x)
elseif d==2;
[energy]=cnde2fs(x)
elseif d==3;
[energy]=cnde3fs(x)
else;
[energy]=cnde1fs(x)
[energy]=cnde2fs(x)
[energy]=cnde3fs(x)
end;
elseif c==2;
disp('Domino CMOS
logic design style')
disp('————————— ')
disp(' ')
if d==1;
disp(' ')
[energy]=cnde1dm(x)
elseif d==2;
[energy]=cnde2dm(x)
elseif d==3;
[energy]=cnde3dm(x)
else;
[energy]=cnde1dm(x)
 [energy]=cnde2dm(x)
[energy]=cnde3dm(x)
end;
elseif c==3;
disp('Complementary
pass-transistor logic
design style')
disp('————————— ')
disp(' ')
if d==1;
disp(' ')
[energy]=cnde1cp(x)
elseif d==2;
[energy]=cnde2cp(x)
elseif d==3;
[energy]=cnde3cp(x)
else;
[energy]=cnde1cp(x)
 [energy]=cnde2cp(x)
[energy]=cnde3cp(x)
end;
elseif c==4;
disp('Dual pass-
transistor logic design
style')
disp('————————— ')
disp(' ')
if d==1;
disp(' ')
[energy]=cnde1dp(x)
elseif d==2;
[energy]=cnde2dp(x)
elseif d==3;
[energy]=cnde3dp(x)

else;
[energy]=cnde1dp(x)

[energy]=cnde2dp(x)
[energy]=cnde3dp(x)
end;
elseif c==0;
disp('Fully static CMOS
logic design style')
disp('————————— ')
disp(' ')
if d==1;
disp(' ')
[energy]=cnde1fs(x)
elseif d==2;
[energy]=cnde2fs(x)
elseif d==3;
[energy]=cnde3fs(x)
else;
[energy]=cnde1fs(x)
[energy]=cnde2fs(x)
[energy]=cnde3fs(x)
end;
disp('Domino CMOS
logic design style')
disp('————————————
——— ')
disp(' ')
if d==1;
disp(' ')
[energy]=cnde1dm(x)
elseif d==2;
[energy]=cnde2dm(x)
elseif d==3;
[energy]=cnde3dm(x)
else;
[energy]=cnde1dm(x)
 [energy]=cnde2dm(x)
[energy]=cnde3dm(x)
end;
disp('Complementary
pass-transistor logic
design style')
disp('————————— ')
disp(' ')
if d==1;
disp(' ')
[energy]=cnde1cp(x)
elseif d==2;
[energy]=cnde2cp(x)
elseif d==3;
[energy]=cnde3cp(x)
else;
[energy]=cnde1cp(x)
 [energy]=cnde2cp(x)
[energy]=cnde3cp(x)
end;
disp('Dual pass-
transistor logic design
style')
disp('————————— ')
disp(' ')
if d==1;
disp(' ')
[energy]=cnde1dp(x)
elseif d==2;
[energy]=cnde2dp(x)
elseif d==3;
[energy]=cnde3dp(x)
 else;
[energy]=cnde1dp(x)
[energy]=cnde2dp(x)
[energy]=cnde3dp(x)
end;
end;
end;
```

APPENDIX A3

Functions for worst-case propagation-delay calculations

These functions are called in main functions 'P' and 'DELSIM' listed in ***Appendix A2*** and are used to compute the values of worst-case propagation-delay using the quadratic fits to the available data points.

```
function[delay]=cla1dp(x)
format;
delay=(1E-9)*(-
0.0039*x^2 + 0.6965*x +
5.2802);
disp(' ')
disp('Propagation Delay
for carry look ahead adder
(W=1.8um) Dual pass-
transistor logic ,in
seconds, is ')

function[delay]=cla1cp(x)
format;
delay=(1E-9)*( -
0.0025*x^2 + 0.4696*x +
3.1045);
disp(' ')
disp('Propagation Delay
for carry look ahead adder
(W=1.8um)
Complementary pass-
transistor logic ,in
seconds, is ')

function[delay]=cla1dm(x
)
format;
delay= (1e-9)*(-
0.0004*x^2 + 0.3612*x +
4.9329);
disp(' ')
disp('Propagation Delay
for carry look ahead adder
(W=1.8um) Domino
CMOS logic ,in seconds,
is ')

function[delay]=cla1fs(x)
format;
delay=(1e-9)*( -
0.0018*x^2 + 0.2894*x +
3.0251);
disp(' ')
disp('Propagation Delay
for carry look ahead adder
(W=1.8um) fully static
CMOS logic ,in seconds,
is ')

function[delay]=cla2dp(x)
format;
delay=(1E-9)*( -
0.0034*x^2 + 0.5142*x +
4.0016);
disp(' ')
disp('Propagation Delay
for carry look ahead adder
(W=3.6um) Dual pass-
transistor logic ,in
seconds, is ')

function[delay]=cla2cp(x)
format;
delay=(1e-9)*(-
0.0022*x^2 + 0.3414*x +
2.1914);
disp(' ')
disp('Propagation Delay
for carry look ahead adder
(W=3.6um)
Complementary pass-
transistor logic ,in
seconds, is ')

function[delay]=cla2dm(x
)
format;
delay=(1e-9)*( -
0.001*x^2 + 0.2705*x +
3.6276);
disp(' ')
disp('Propagation Delay
for carry look ahead adder
(W=3.6um) Domino
CMOS logic ,in seconds,
is ')

function[delay]=cla2fs(x)
format;
delay=(1e-9)*( -
0.0014*x^2 + 0.2136*x +
2.3761);
disp(' ')
disp('Propagation Delay
for carry look ahead adder
(W=3.6um) fully static
CMOS logic ,in seconds,
is ')

function[delay]=cla3dp(x)
format;
delay=(1E-9)*( -
0.003*x^2 + 0.4297*x +
3.4873);
disp(' ')
disp('Propagation Delay
for carry look ahead adder
(W=6um) Dual pass-
transistor logic ,in
seconds, is ')

function[delay]=cla3cp(x)
format;
delay=(1e-9)*( -
0.0018*x^2 + 0.2517*x +
2.182);
disp(' ')
disp('Propagation Delay
for carry look ahead adder
(W=6um)
Complementary pass-
transistor logic ,in
seconds, is ')

function[delay]=cla3dm(x
)
format;
delay=(1e-9)*( -
0.001*x^2 + 0.2241*x +
3.1704);
disp(' ')
disp('Propagation Delay
for carry look ahead adder
(W=6um) Domino CMOS
logic ,in seconds, is ')

function[delay]=cla3fs(x)
format;
delay=(1e-9)*( -
0.001*x^2 + 0.1605*x +
2.2151);
disp(' ')
disp('Propagation Delay
for carry look ahead adder
(W=6um) fully static
CMOS logic ,in seconds,
is ')

function[delay]=cnd1dp(x
)
format;
delay= (1e-9)*(
0.0032*x^2 + 0.2321*x +
3.0929);
disp(' ')
disp('Propagation Delay
for conditional sum adder
(W=1.8um) Dual pass-
transistor logic ,in
seconds, is ')
function[delay]=cnd1cp(x
)
format;
delay= (1e-
9)*(0.0047*x^2 +
0.318*x + 1.7206);
disp(' ')
disp('Propagation Delay
for conditional sum adder
(W=1.8um)
Complementary pass-
transistor logic ,in
seconds, is ')

function[delay]=cnd1dm(
x)
format;
delay=(1e-9)*(
0.0053*x^2 + 0.4667*x +
2.4002);
disp(' ')
disp('Propagation Delay
for conditional sum adder
(W=1.8um) Domino
CMOS logic ,in seconds,
is ')

function[delay]=cnd1fs(x)
format;
delay=(1e-9)*(
0.0051*x^2 + 0.6163*x +
1.3739);
disp(' ')
disp('Propagation Delay
for conditional sum adder
(W=1.8um) fully static
CMOS logic ,in seconds,
is ')

function[delay]=cnd2dp(x
)
format;
delay= (1e-9)*(
0.0013*x^2 + 0.1664*x +
2.1533);
disp(' ')
disp('Propagation Delay
for conditional sum adder
(W=3.6um) Dual pass-
transistor logic ,in
seconds, is ')

function[delay]=cnd2cp(x
)
format;
delay=(1e-9)*(
0.0024*x^2 + 0.2133*x +
1.2993);
```

```
disp(' ')
disp('Propagation Delay
for conditional sum adder
(W=3.6um)
Complementary pass-
transistor logic ,in
seconds, is ')

function[delay]=cnd2dm(
x)
format;
delay=(1e-9)*(
0.0023*x^2 + 0.3317*x +
2.0451);
disp(' ')
disp('Propagation Delay
for conditional sum adder
(W=3.6um) Domino
CMOS logic ,in seconds,
is ')

function[delay]=cnd2fs(x)
format;
delay=(1e-9)*(
0.0023*x^2 + 0.4373*x +
1.4593);
disp(' ')
disp('Propagation Delay
for conditional sum adder
(W=3.6um) fully static
CMOS logic ,in seconds,
is ')

function[delay]=cnd3dp(x
)
format;
delay= (1e-9)*(
0.0009*x^2 + 0.135*x +
1.9635);
disp(' ')
disp('Propagation Delay
for conditional sum adder
(W=6um) Dual pass-
transistor logic ,in
seconds, is ')

function[delay]=cnd3cp(x
)
format;
delay=(1e-9)*(
0.0007*x^2 + 0.2376*x +
0.8413);
disp(' ')
disp('Propagation Delay
for conditional sum adder
(W=6um)
Complementary pass-
transistor logic ,in
seconds, is ')

function[delay]=cnd3dm(
x)
format;
delay=(1e-9)*(
0.0015*x^2 + 0.2556*x +
1.9651);
disp(' ')
disp('Propagation Delay
for conditional sum adder
(W=6um) Domino CMOS
logic ,in seconds, is ')

function[delay]=cnd3fs(x)
format;
delay=(1e-9)*(
0.0028*x^2 + 0.2995*x +
1.8743);
disp(' ')
disp('Propagation Delay
for conditional sum adder
(W=6um) fully static
CMOS logic ,in seconds,
is ')

function[delay]=crl1dp(x)
format;
delay=(1e-9)*(
0.0071*x^2 + 0.5958*x +
3.2529);
disp(' ')
disp('Propagation Delay
for carry select adder
(W=1.8um) Dual pass-
transistor CMOS logic ,in
seconds, is ')

function[delay]=crl1cp(x)
format;
delay=(1e-9)*(
0.0068*x^2 + 0.4267*x +
2.056);
disp(' ')
disp('Propagation Delay
for carry select adder
(W=1.8um)
Complementary pass-
transistor CMOS logic ,in
seconds, is ')

function[delay]=crl1dm(x
)
format;
delay=(1e-9)*(
0.0124*x^2 + 0.448*x +
5.2392);
disp(' ')
disp('Propagation Delay
for carry select adder
(W=1.8um) Domino
CMOS logic ,in seconds,
is ')

function[delay]=crl1fs(x)
format;
delay=(1e-9)*(
0.0021*x^2 + 0.6046*x +
0.7063);
disp(' ')
disp('Propagation Delay
for carry select adder
(W=1.8um) fully static
CMOS logic ,in seconds,
is ')

function[delay]=crl2dp(x)
format;
delay=(1e-9)*(
0.0038*x^2 + 0.4061*x +
2.838);
disp(' ')
disp('Propagation Delay
for carry select adder
(W=3.6um) Dual pass-
transistor CMOS logic ,in
seconds, is ')

function[delay]=crl2cp(x)
format;
delay=(1e-
9)*(0.0005*x^2 +
0.4706*x + 0.479);
disp(' ')
disp('Propagation Delay
for carry select adder
(W=3.6um)
Complementary pass-
transistor CMOS logic ,in
seconds, is ')

function[delay]=crl2dm(x
)
format;
delay=(1e-9)*(
0.0064*x^2 + 0.3635*x +
3.8553);
disp(' ')
disp('Propagation Delay
for carry select adder
(W=3.6um) Domino
CMOS logic ,in seconds,
is ')

function[delay]=crl2fs(x)
format;
delay=(1e-9)*(
0.0011*x^2 + 0.4097*x +
0.9759);
disp(' ')
disp('Propagation Delay
for carry select adder
(W=3.6um) fully static
CMOS logic ,in seconds,
is ')

function[delay]=crl3dp(x)
format;
delay=(1e-9)*(
0.0023*x^2 + 0.3514*x +
2.481);
disp(' ')
disp('Propagation Delay
for carry select adder
(W=6um) Dual pass-
transistor CMOS logic ,in
seconds, is ')

function[delay]=crl3cp(x)
format;
delay=(1e-
9)*(0.0016*x^2 +
0.2942*x + 1.2373);
disp(' ')
disp('Propagation Delay
for carry select adder
(W=6um)
Complementary pass-
transistor CMOS logic ,in
seconds, is ')

function[delay]=crl3dm(x
)
format;
delay=(1e-
9)*(0.0041*x^2 +
0.3115*x + 3.5145);
disp(' ')
disp('Propagation Delay
for carry select adder
(W=6um) Domino CMOS
logic ,in seconds, is ')

function[delay]=crl3fs(x)
format;
delay=(1e-9)*
(0.0016*x^2 + 0.2826*x
+ 1.3547);
disp(' ')
disp('Propagation Delay
for carry select adder
(W=6um) fully static
CMOS logic ,in seconds,
is ')

function[delay]=csk1dp(x
)
format;
delay=(1e-9)*(
0.0005*x^2 + 0.9942*x +
4.4639);
disp(' ')
disp('Propagation Delay
for carry skip adder
(W=1.8um) Dual pass-
transistor logic ,in
seconds, is ')

function[delay]=csk1cp(x
)
format;
delay=(1e-9)*(
0.0036*x^2 + 0.634*x +
3.3071);
disp(' ')
disp('Propagation Delay
for carry skip adder
(W=1.8um)
Complementary pass-
transistor logic ,in
seconds, is ')

function[delay]=csk1dm(
x)
format;
delay=(1e-9)*( 0.002*x^2
+ 0.7902*x + 5.9333);
disp(' ')
disp('Propagation Delay
for carry skip adder
(W=1.8um) Domino
CMOS logic ,in seconds,
is ')

function[delay]=csk1fs(x)
format;
delay=(1e-9)*(
0.0007*x^2 + 0.4859*x +
3.162);
disp(' ')
```

```
disp('Propagation Delay
for carry skip
adder(W=1.8um) fully
static CMOS logic ,in
seconds, is ')

function[delay]=csk2dp(x
)
format;
delay=(1e-
9)*(0.0003*x^2 +
0.6448*x - 3.0637);
disp(' ')
disp('Propagation Delay
for carry skip adder
(W=3.6um) Dual pass-
transistor logic ,in
seconds, is ')

function[delay]=csk2cp(x
)
format;
delay=(1e-9)*( -
0.0002*x^2 + 0.5068*x +
1.89);
disp(' ')
disp('Propagation Delay
for carry skip adder
(W=3.6um)
Complementary pass-
transistor logic,in
seconds, is ')
function[delay]=csk2dm(
x)
format;
delay=(1e-9)*(
0.0002*x^2 + 0.57*x +
4.6646);
disp(' ')
disp('Propagation Delay
for carry skip adder
(W=3.6um) Domino
CMOS logic ,in seconds,
is ')
function[delay]=csk2fs(x)
format;
delay=(1e-9)*(
0.0001*x^2 + 0.3387*x +
2.5418);
disp(' ')
disp('Propagation Delay
for carry skip
adder(W=3.6um) fully
static CMOS logic ,in
seconds, is ')

function[delay]=csk3dp(x
)
format;
delay=(1e-9)*(2E-05*x^2
+ 0.5633*x + 2.979);
disp(' ')
disp('Propagation Delay
for carry skip adder
(W=6um) Dual pass-
transistor logic ,in
seconds, is ')

function[delay]=csk3cp(x
)
format;
delay=(1e-9)*(
0.0004*x^2 + 0.3956*x +
1.9967);
disp(' ')
disp('Propagation Delay
for carry skip adder
(W=6um)
Complementary pass-
transistor logic ,in
seconds, is ')

function[delay]=csk3dm(
x)
format;
delay=(1e-9)*(-
0.0006*x^2 + 0.4788*x +
3.9632);
disp(' ')
disp('Propagation Delay
for carry skip adder
(W=6um) Domino CMOS
logic ,in seconds, is ')
function[delay]=csk3fs(x)
format;
delay=(1e-
9)*(0.0003*x^2 +
0.2863*x + 2.3246);
disp(' ')
disp('Propagation Delay
for carry skip adder
(W=6um) fully static
CMOS logic ,in seconds,
is ')
function[delay]=rca1dp(x
)
format;
delay=(1e-
9)*(0.0099*x^2 +
1.6074*x - 1.0741);
disp(' ')
disp('Propagation Delay
for ripple carry adder
(W=1.8um) Dual pass-
transistor CMOS logic ,in
seconds, is ')
function[delay]=rca1cp(x)
format;
delay=(1e-9)*(
0.0085*x^2 + 0.9921*x -
0.794);
disp(' ')
disp('Propagation Delay
for ripple carry adder
(W=1.8um)
Complementary pass-
transistor CMOS logic ,in
seconds, is ')

function[delay]=rca1dm(x
)
format;
delay=(1e-
9)*(0.0177*x^2 +
1.7461*x - 0.9037);
disp(' ')
disp('Propagation Delay
for ripple carry adde
(W=1.8um) Domino
CMOS logic ,in seconds,
is ')

function[delay]=rca1fs(x)
format;
delay=(1e-9)*(
0.0139*x^2 + 1.013*x -
1.0159);
disp(' ')
disp('Propagation delay
for ripple carry adder
(W=1.8um) fully static
CMOS logic ,in seconds,
is ')

function[delay]=rca2dp(x
)
format;
delay=(1e-
9)*(0.0056*x^2 +
1.1504*x - 0.5714);
disp(' ')
disp('Propagation Delay
for ripple carry adder
(W=3.6um) Dual pass-
transistor CMOS logic ,in
seconds, is ')
function[delay]=rca2cp(x)
format;
delay=(1e-9)*( 0.008*x^2
+ 0.5646*x + 0.4741);
disp(' ')
disp('Propagation Delay
for ripple carry adder
(W=3.6um)
Complementary pass-
transistor CMOS logic ,in
seconds, is ')
function[delay]=rca2dm(x
)
format;
delay=(1e-9)*(0.007*x^2
+ 1.399*x - 1.2423);
disp(' ')
disp('Propagation Delay
for ripple carry adder
(W=3.6um) Domino
CMOS logic ,in seconds,
is ')

function[delay]=rca2fs(x)
format;
delay=(1e-9)*(
0.0065*x^2 + 0.7756*x -
0.5855);
disp(' ')
disp('Propagation delay
for ripple carry adder
(W=3.6um) fully static
CMOS logic ,in seconds,
is ')

function[delay]=rca3dp(x
)
format;
delay=(1e-9)*(
0.0035*x^2 + 1.0116*x -
0.4588);
disp(' ')
disp('Propagation Delay
for ripple carry adder
(W=6um) Dual pass-
transistor CMOS logic ,in
seconds, is ')

function[delay]=rca3cp(x)
format;
delay=(1e-9)*(
0.0042*x^2 + 0.555*x +
0.2104);
disp(' ')
disp('Propagation Delay
for ripple carry adder
(W=6um)
Complementary pass-
transistor CMOS logic ,in
seconds, is ')
function[delay]=rca3dm(x
)
format;
delay=(1e-9)*(
0.0044*x^2 + 1.1601*x -
0.5749);
disp(' ')
disp('Propagation Delay
for ripple carry adder
(W=6um) Domino CMOS
logic ,in seconds, is ')
function[delay]=rca3fs(x)
format;
delay=(1e-
9)*(0.0042*x^2 +
0.6776*x - 0.3194);
disp(' ')
disp('Propagation delay
for ripple carry adder
(W=6um) fully static
CMOS logic ,in seconds,
is ')
```

Functions for energy consumption calculations

These functions are called in main function 'EP' and 'ENRGSIM' listed in ***Appendix A2*** and are used to compute the values of energy consumption per operation using the quadratic fits to the available data points.

```
function[energy]=clae1dp
(x)
format;
energy=(1e-
10)*(0.0024*x^2 +
0.2182*x - 0.11);
disp(' ')
disp('Energy consumption
for carry look ahead adder
(W=1.8um) Dual pass-
transistor logic ,in Joules,
is ')

function[energy]=clae1cp
(x)
format;
energy=(1e-10)*(
0.004*x^2 + 0.3664*x -
0.2074);
disp(' ')
disp('Energy consumption
for carry look ahead adder
(W=1.8um)
Complementary pass-
transistor logic ,in Joules,
is ')

function[energy]=clae1d
m(x)
format;
energy=(1e-10)*(
0.0021*x^2 + 0.3151*x -
0.1734);
disp(' ')
disp('Energy consumption
for carry look ahead adder
(W=1.8um) Domino
CMOS logic ,in Joules, is
')

function[energy]=clae1fs(
x)
format;
energy=(1e-10)*(
0.0009*x^2 + 0.1246*x -
0.2025);
disp(' ')
disp('Energy consumption
for carry look ahead adder
(W=1.8um) fully static
CMOS logic ,in Joules, is
')

function[energy]=clae2dp
(x)
format;
energy=(1e-
10)*(0.0021*x^2 +
0.3454*x - 0.2538);
disp(' ')
disp('Energy consumption
for carry look ahead adder
(W=3.6um) Dual pass-
transistor logic ,in Joules,
is ')

function[energy]=clae2cp
(x)
format;
energy=(1e-10)*(
0.0036*x^2 + 0.5845*x -
0.5567);
disp(' ')
disp('Energy consumption
for carry look ahead adder
(W=3.6um)
Complementary pass-
transistor logic ,in Joules,
is ')

function[energy]=clae2d
m(x)
format;
energy=(1e-10)*(
0.0018*x^2 + 0.4941*x -
0.2867);
disp(' ')
disp('Energy consumption
for carry look ahead adder
(W=3.6um) Domino
CMOS logic ,in Joules, is
')

function[energy]=clae2fs(
x)
format;
energy=(1e-10)*(
0.0017*x^2 + 0.136*x +
0.0319);
disp(' ')
disp('Energy consumption
for carry look ahead adder
(W=3.6um) fully static
CMOS logic ,in Joules, is
')

function[energy]=clae3dp
(x)
format;
energy=(1e-10)*(
0.0023*x^2 + 0.4915*x -
0.2972);
disp(' ')
disp('Energy consumption
for carry look ahead adder
(W=6um) Dual pass-
transistor logic ,in Joules,
is ')

function[energy]=clae3cp
(x)
format;
energy=(1e-10)*(
0.0026*x^2 + 0.9725*x -
2.3015);
disp(' ')
disp('Energy consumption
for carry look ahead adder
(W=6um)
Complementary pass-
transistor logic ,in Joules,
is ')

function[energy]=clae3d
m(x)
format;
energy=(1e-10)*(
0.0024*x^2 + 0.681*x -
0.215);
disp(' ')
disp('Energy consumption
for carry look ahead adder
(W=6um) Domino CMOS
logic ,in Joules, is ')

function[energy]=clae3fs(
x)
format;
energy=(1e-10)*(
0.0016*x^2 + 0.216*x -
0.0658);
disp(' ')
disp('Energy consumption
for carry look ahead adder
(W=6um) fully static
CMOS logic ,in Joules, is
')

function[energy]=cnde1d
p(x)
format;
energy=(1e-10)*(
0.0022*x^2 + 0.212*x +
0.1357);
disp(' ')
disp('Energy consumption
for conditional sum adder
(W=1.8um) Dual pass-
transistor logic ,in Joules,
is ')

function[energy]=cnde1c
p(x)
format;
energy=(1e-10)*(
0.0022*x^2 + 0.1619*x +
0.16031);
disp(' ')
disp('Energy consumption
for conditional sum adder
(W=1.8um)
Complementary pass-
transistor logic ,in Joules,
is ')

function[energy]=cnde1d
m(x)
format;
energy=(1e-
10)*(0.0012*x^2 +
0.3933*x - 0.2326);
disp(' ')
disp('Energy consumption
for conditional sum adder
(W=1.8um) Domino
CMOS logic ,in Joules, is
')

function[energy]=cnde1fs
(x)
format;
energy=(1e-10)*(
0.001*x^2 + 0.2864*x -
0.3989);
disp(' ')
disp('Energy consumption
for conditional sum adder
(W=1.8um) fully static
CMOS logic ,in Joules, is
')

function[energy]=cnde2d
p(x)
format;
energy=(1e-
10)*(0.002*x^2 +
0.2841*x + 0.2221);
disp(' ')
disp('Energy consumption
for conditional sum adder
(W=3.6um) Dual pass-
transistor logic ,in Joules,
is ')

function[energy]=cnde2c
p(x)
format;
energy=(1e-10)*(
0.0024*x^2 + 0.2293*x +
0.2701);
disp(' ')
disp('Energy consumption
for conditional sum adder
(W=3.6um)
Complementary pass-
transistor logic ,in Joules,
is ')
```

```
function[energy]=cnde2dm(x)
format;
energy=(1e-10)*(0.0012*x^2 + 0.6082*x - 0.3223);
disp(' ')
disp('Energy consumption for conditional sum adder (W=3.6um) Domino CMOS logic ,in Joules, is ')

function[energy]=cnde2fs(x)
format;
energy=(1e-10)*(0.0004*x^2 + 0.4626*x - 0.6674);
disp(' ')
disp('Energy consumption for conditional sum adder (W=3.6um) fully static CMOS logic ,in Joules, is ')

function[delay]=cnd3dp(x)
format;
delay= (1e-9)*(0.0009*x^2 + 0.135*x + 1.9635);
disp(' ')
disp('Propagation Delay for conditional sum adder (W=6um) Dual pass-transistor logic ,in seconds, is ')

function[energy]=cnde3cp(x)
format;
energy=(1e-10)*(0.0027*x^2 + 0.3411*x + 0.3622);
disp(' ')
disp('Energy consumption for conditional sum adder (W=6um) Complementary pass-transistor logic ,in Joules, is ')

function[energy]=cnde3dm(x)
format;
energy=(1e-10)*(0.0015*x^2 + 0.8753*x - 0.3719);
disp(' ')
disp('Energy consumption for conditional sum adder (W=6um) Domino CMOS logic ,in Joules, is ')

function[energy]=cnde3fs(x)
format;
energy=(1e-10)*(0.0016*x^2 + 0.5575*x - 0.3139);
disp(' ')
disp('Energy consumption for conditional sum adder (W=6um) fully static CMOS logic ,in Joules, is ')

function[energy]=crle1dp(x)
format;
energy=(1e-10)*( - 0.0008*x^2 + 0.5685*x - 1.7441);
disp(' ')
disp('Energy consumption for carry select adder (W=1.8um) Dual pass-transistor logic ,in Joules, is ')

function[energy]=crle1cp(x)
format;
energy=(1e-10)*(0.0023*x^2 + 0.2476*x - 0.2617);
disp(' ')
disp('Energy consumption for carry select adder (W=1.8um) Complementary pass-transistor logic ,in Joules, is ')

function[energy]=crle1dm(x)
format;
energy=(1e-10)*(0.0048*x^2 + 0.4511*x + 0.0517);
disp(' ')
disp('Energy consumption for carry select adder (W=1.8um) Domino CMOS logic ,in Joules, is ')

function[energy]=crle1fs(x)
format;
energy=(1e-10)*(0.0011*x^2 + 0.2021*x - 0.4036);
disp(' ')
disp('Energy consumption for carry select adder (W=1.8um) fully static CMOS logic ,in Joules, is ')

function[energy]=crle2dp(x)
format;
energy=(1e-10)*(- 0.0016*x^2 + 0.795*x - 2.1764);
disp(' ')
disp('Energy consumption for carry select adder (W=3.6um) Dual pass-transistor logic ,in Joules, is ')

function[energy]=crle2cp(x)
format;
energy=(1e-10)*(0.0019*x^2 + 0.3868*x - 0.5022);
disp(' ')
disp('Energy consumption for carry select adder (W=3.6um) Complementary pass-transistor logic ,in Joules, is ')

function[energy]=crle2dm(x)
format;
energy=(1e-10)*(0.0048*x^2 + 0.7667*x - 0.2282);
disp(' ')
disp('Energy consumption for carry select adder (W=3.6um) Domino CMOS logic ,in Joules, is ')

function[energy]=crle2fs(x)
format;
energy=(1e-10)*(0.0017*x^2 + 0.2687*x - 0.3769);
disp(' ')
disp('Energy consumption for carry select adder (W=3.6um) fully static CMOS logic ,in Joules, is ')

function[energy]=crle3dp(x)
format;
energy=(1e-10)*( - 0.0071*x^2 + 1.4211*x - 4.6013);
disp(' ')
disp('Energy consumption for carry select adder (W=6um) Dual pass-transistor logic ,in Joules, is ')

function[energy]=crle3cp(x)
format;
energy=(1e-10)*(0.0013*x^2 + 0.5589*x - 0.4465);
disp(' ')
disp('Energy consumption for carry select adder (W=6um) Complementary pass-transistor logic ,in Joules, is ')

function[energy]=crle3dm(x)
format;
energy=(1e-10)*(0.0067*x^2 + 1.0913*x - 0.0803);
disp(' ')
disp('Energy consumption for carry select adder (W=6um) Domino CMOS logic ,in Joules, is ')

function[energy]=crle3fs(x)
format;
energy=(1e-10)*(0.0041*x^2 + 0.3162*x - 0.1291);
disp(' ')
disp('Energy consumption for carry select adder (W=6um) fully static CMOS logic ,in Joules, is ')

function[energy]=cske1dp(x)
format;
energy=(1e-10)*(0.0019*x^2 + 0.2637*x - 0.3896);
disp(' ')
disp('Energy consumption for carry skip adder (W=1.8um) Dual pass-transistor logic ,in Joules, is ')

function[energy]=cske1cp(x)
format;
energy=(1e-10)*(0.0035*x^2 + 0.1779*x + 0.0184);
disp(' ')
disp('Energy consumption for carry skip adder (W=1.8um) Complementary pass-transistor logic ,in Joules, is ')

function[energy]=cske1dm(x)
format;
energy= (1e-10)*(0.0029*x^2 + 0.2965*x - 0.1404);
disp(' ')
disp('Energy consumption for carry skip adder (W=1.8um) Domino CMOS logic ,in Joules, is ')

function[energy]=cske1fs(x)
```

```
format.
energy=(1e-10)*(
0.0016*x^2 + 0.107*x -
0.173);
disp(' ')
disp('Energy consumption
for carry skip adder
(W=1.8um) fully static
CMOS logic ,in Joules, is
')

function[energy]=cske2dp
(x)
format;
energy=(1e-10)*(
0.0022*x^2 + 0.3685*x -
0.5033);
disp(' ')
disp('Energy consumption
for carry skip adder
(W=3.6um) Dual pass-
transistor logic ,in Joules,
is ')

function[energy]=cske2cp
(x)
format;
energy=(1e-10)*(
0.0036*x^2 + 0.2896*x -
0.1741);
disp(' ')
disp('Energy consumption
for carry skip adder
(W=3.6um)
Complementary pass-
transistor logic ,in Joules,
is ')
function[energy]=cske2d
m(x)
format;
energy=(1e-10)*(
0.0027*x^2 + 0.4803*x -
0.2584);
disp(' ')
disp('Energy consumption
for carry skip adder
(W=3.6um) Domino
CMOS logic ,in Joules, is
')
function[energy]=cske2fs
(x)
format;
energy=(1e-10)*(
0.0019*x^2 + 0.1474*x -
0.1961);
disp(' ')
disp('Energy consumption
for carry skip adder
(W=3.6um) fully static
CMOS logic ,in Joules, is
')
function[energy]=cske3dp
(x)
format;
energy=(1e-10)*(
0.0095*x^2 + 0.2816*x +
0.7424);
disp(' ')
disp('Energy consumption
for carry skip adder
(W=6um) Dual pass-
transistor logic ,in Joules,
is ')

function[energy]=cske3cp
(x)
format;
energy=(1e-10)*(
0.0051*x^2 + 0.4095*x -
0.1421);
disp(' ')
disp('Energy consumption
for carry skip adder
(W=6um)
Complementary pass-
transistor logic ,in Joules,
is ')

function[energy]=cske3d
m(x)
format;
energy=(1e-10)*(
0.0031*x^2 + 0.6875*x -
0.3411);
disp(' ')
disp('Energy consumption
for carry skip adder
(W=6um) Domino CMOS
logic ,in Joules, is ')

function[energy]=cske3fs
(x)
format;
energy= (1e-10)*(
0.0017*x^2 + 0.2284*x -
0.2753);
disp(' ')
disp('Energy consumption
for carry skip adder
(W=6um) fully static
CMOS logic ,in Joules, is
')
function[energy]=rcae1dp
(x)
format;
energy=(1e-10)*(
0.0013*x^2 + 0.1944*x -
0.1769);
disp(' ')
disp('Energy consumption
for ripple carry adder
(W=1.8um) Dual pass-
transistor logic ,in Joules,
is ')
function[energy]=rcae2dp
(x)
format;
energy=(1e-10)*(
0.0013*x^2 + 0.2939*x -
0.0864);
disp(' ')
disp('Energy consumption
for ripple carry adder
(W=3.6um) Dual pass-
transistor logic ,in Joules,
is ')
function[energy]=rcae1cp
(x)
format;
energy=(1e-10)*(
0.0024*x^2 + 0.1202*x +
0.0917);
disp(' ')
disp('Energy consumption
for ripple carry adder
(W=1.8um)
Complementary pass-
transistor logic ,in Joules,
is ')

function[energy]=rcae1d
m(x)
format;
energy=(1e-10)*(
0.0024*x^2 + 0.2696*x -
0.0877);
disp(' ')
disp('Energy consumption
for ripple carry adder
(W=1.8um) Domino
CMOS logic ,in Joules, is
')

function[energy]=rcae1fs(
x)
format;
energy= (1e-
10)*(0.0016*x^2 +
0.0841*x - 0.1103);
disp(' ')
disp('Energy consumption
for ripple carry adder
(W=1.8um) fully static
CMOS logic ,in Joules, is
')
function[energy]=rcae2dp
(x)
format;
energy=(1e-10)*(
0.0013*x^2 + 0.2939*x -
0.0864);
disp(' ')
disp('Energy consumption
for ripple carry adder
(W=3.6um) Dual pass-
transistor logic ,in Joules,
is ')
function[energy]=rcae2cp
(x)
format;
energy=(1e-
10)*(0.0033*x^2 +
0.1611*x + 0.2172);
disp(' ')
disp('Energy consumption
for ripple carry adder
(W=3.6um)
Complementary pass-
transistor logic ,in Joules,
is ')
function[energy]=rcae2d
m(x)
format;
energy=(1e-10)*(
0.0021*x^2 + 0.4391*x -
0.2237);
disp(' ')
disp('Energy consumption
for ripple carry adder
(W=3.6um) Domino
CMOS logic ,in Joules, is
')

function[energy]=rcae2fs(
x)
format;
energy=(1e-10)*(
0.0014*x^2 + 0.1322*x -
0.1597);
disp(' ')
disp('Energy consumption
for ripple carry adder
(W=3.6um) fully static
CMOS logic ,in Joules, is
')

function[energy]=rcae3dp
(x)
format;
energy=(1e-10)*(
0.0017*x^2 + 0.3965*x -
0.2563);
disp(' ')
disp('Energy consumption
for ripple carry adder
(W=6um) Dual pass-
transistor logic ,in Joules,
is ')
function[energy]=rcae3cp
(x)
format;
energy=(1e-10)*(
0.0034*x^2 + 0.2577*x +
0.2392);
disp(' ')
disp('Energy consumption
for ripple carry adder
(W=6um)
Complementary pass-
transistor logic ,in Joules,
is ')
function[energy]=rcae3d
m(x)
format;
energy=(1e-10)*(
0.0023*x^2 + 0.6222*x -
0.1704);
disp(' ')
disp('Energy consumption
for ripple carry adder
(W=6um) Domino CMOS
logic ,in Joules, is ')
function[energy]=rcae3fs(
x)
format;
energy=(1e-
10)*(0.0018*x^2 +
0.1691*x - 0.0593);
disp(' ')
disp('Energy consumption
for ripple carry adder
(W=6um) fully static
CMOS logic ,in Joules, is
')
```

APPENDIX A4

This program calculates the worst-case propagation delay using Delay Model

```
clear
diary prgresult.m
diary on
disp('*****************
**********************
**********************
**********************
****** ')
disp(' ')
disp('Programme for
choosing best architecture
for a required propagation
delay constraint')
disp(' ')
disp('Modelling results
using Delay Models ')
disp(' ')
disp('*****************
**********************
**********************
**********************
****** ')
%Take input from user
disp('*****************
**********************
**********************
** ')
disp(' ')
disp('Please input size of
adder from 1 to 64 bits')
disp(' ')
x=input('Please input
adder size (number of
bits)= ');
disp(' ')
while x>64
disp('ERROR: Adder size
is out of the range.
Please give size from 1 to
64')
disp(' ')
if b==1;
disp('Ripple carry adder
architecture')
disp('-----------------
------ ')
disp(' ')
if c==1;
disp('Fully static CMOS
logic design style')
disp('-----------------
-------- ')
disp(' ')
if d==1;
disp(' ')
[delay]=rcam1fs(x)
elseif d==2;
[delay]=rcam2fs(x)
elseif d==3;
[delay]=rcam3fs(x)
else;
[delay]=rcam1fs(x)
x=input('Please input
adder size (number of
bits) again= ');
disp(' ')
end
clc
disp('*****************
**********************
**********************
** ')
disp('Architecture options
and their values are:')
disp(' ')
disp('(1)No specific adder
architecture.Print result
for all architectures= 0')
disp('(2)Ripple carry
adder architecture= 1')
disp('(3)Carry select
adder architecture= 2')
disp('(4)Carry skip adder
architecture= 3')
disp('(5)Carry look ahead
adder architecture= 4')
disp('(6)Conditional sum
adder architecture= 5')
disp(' ')
b=input('Please input
architecture option= ');
while b>5
disp('ERROR: Option
value is out of the range.
Please give option value
between 0 and 5')
disp(' ')
b=input('Please input
architecture option
again= ');
disp(' ')
end
clc;

[delay]=rcam2fs(x)
[delay]=rcam3fs(x)
end;
elseif c==2;
disp('Domino CMOS
logic design style')
disp('-----------------
------ ')
disp(' ')
if d==1;
disp(' ')
[delay]=rcam1dm(x)
elseif d==2;
[delay]=rcam2dm(x)
elseif d==3;
[delay]=rcam3dm(x)
 else;
[delay]=rcam1dm(x)
 [delay]=rcam2dm(x)
[delay]=rcam3dm(x)
end;
disp('*****************
**********************
**********************
** ')
disp('Logic design style
options and their values
are:')
disp(' ')
disp('(1)No specific logic
design style.Print result
for all design styles.= 0')
disp('(2)Fully static
CMOS logic= 1')
disp('(3)Domino CMOS
logic= 2')
disp('(4)Complementary
pass transistor logic= 3')
disp(' ')
c=input('Please input
logic design style option
value= ');
while c>3
disp('ERROR: Option
value is out of the range.
Please give option value
between 0 and 3')
disp(' ')
c=input('Please input
logic design style option
again= ');
disp(' ')
end
clc
disp('*****************
**********************
**********************
** ')
disp(' ')
disp('MOSFET Channel
length is =1.2um')
disp(' ')

elseif c==3;
disp('Complementary
pass-transistor logic
design style')
disp('-----------------
-------- ')
disp(' ')
if d==1;
disp(' ')
[delay]=rcam1cp(x)
elseif d==2;
[delay]=rcam2cp(x)
elseif d==3;
[delay]=rcam3cp(x)
else;
[delay]=rcam1cp(x)
[delay]=rcam2cp(x)
[delay]=rcam3cp(x)
end;
elseif c==4;
disp('*****************
**********************
**********************
** ')
disp('Channel width
options and their values
are:')
disp(' ')
disp('(1)No specific
channel width
choice.Print result for all
(w/l) values= 0')
disp('(2)Channel width
(1.8um)= 1')
disp('(3)Channel width
(3.6um)= 2')
disp('(4)Channel width
(6.0um)= 3')
disp(' ')
d=input('Please input
channel width option= ');
while d>3
disp('ERROR: Option
value is out of the range.
Please give option value
between 0 and 3')
disp(' ')
d=input('Please input
channel width option
again= ');
disp(' ')
end
disp('*****************
**********************
**********************
** ')
x,b,c,d %
size,arch,width,design
style
%%%SIMULATION
RESULTS

disp('Dual pass-transistor
logic design style')
disp('-----------------
-------- ')
disp(' ')
if d==1;
disp(' ')
[delay]=rcam1dp(x)
elseif d==2;
[delay]=rcam2dp(x)
elseif d==3;
[delay]=rcam3dp(x)
else;
[delay]=rcam1dp(x)
[delay]=rcam2dp(x)
[delay]=rcam3dp(x)
end;
 elseif c==0;
disp('Fully static CMOS
logic design style')
```

```
disp('------------------
--------')
disp(' ')
if d ~=1.
disp(' ')
[delay]=rcam1fs(x)
elseif d==2;
[delay]=rcam2fs(x)
elseif d==3;
[delay]=rcam3fs(x)
else;
[delay]=rcam1fs(x)
[delay]=rcam2fs(x)
[delay]=rcam3fs(x)
end;
disp('Domino CMOS
logic design style')
disp('------------------
-------')
disp(' ')
if d==1;
disp(' ')
[delay]=rcam1dm(x)
elseif d==2;
[delay]=rcam2dm(x)
elseif d==3;
[delay]=rcam3dm(x)
 else;
[delay]=rcam1dm(x)
 [delay]=rcam2dm(x)
[delay]=rcam3dm(x)
end;
disp('Complementary
pass-transistor logic
design style')
disp('------------------
-----------------')
disp(' ')
if d==1;
disp(' ')
[delay]=rcam1cp(x)
elseif d==2;
[delay]=rcam2cp(x)
elseif d==3;
[delay]=rcam3cp(x)
else;
[delay]=rcam1cp(x)
 [delay]=rcam2cp(x)
[delay]=rcam3cp(x)
end;
disp('Dual pass-transistor
logic design style')
disp('------------------
-----------------')
disp(' ')
if d==1;
disp(' ')
[delay]=rcam1dp(x)
elseif d==2;
[delay]=rcam2dp(x)
elseif d==3;
[delay]=rcam3dp(x)
else;
[delay]=rcam1dp(x)
[delay]=rcam2dp(x)
[delay]=rcam3dp(x)
end;
end;
elseif b==2;
disp('Carry select adder
architecture')
disp('------------------
-------')
disp(' ')
if c==1;
disp('Fully static CMOS
logic design style')
disp('------------------
--------')
disp(' ')
if d==1;
disp(' ')
disp(' ')
[delay]=crlm1fs(x)
elseif d==2;
[delay]=crlm2fs(x)
elseif d==3;
[delay]=crlm3fs(x)
else;
[delay]=crlm1fs(x)
[delay]=crlm2fs(x)
[delay]=crlm3fs(x)
end;
elseif c==2;
disp('Domino CMOS
logic design style')
disp('------------------
-------')
disp(' ')
if d==1;
disp(' ')
[delay]=crlm1dm(x)
 elseif d==2;
[delay]=crlm2dm(x)
 elseif d==3;
[delay]=crlm3dm(x)
 else;
[delay]=crlm1dm(x)
 [delay]=crlm2dm(x)
[delay]=crlm3dm(x)
end;
elseif c==3;
disp('Complementary
pass-transistor logic
design style')
disp('------------------
-----------------')
disp(' ')
if d==1;
disp(' ')
[delay]=crlm1cp(x)
 elseif d==2;
[delay]=crlm2cp(x)
 elseif d==3;
[delay]=crlm3cp(x)
 else;
[delay]=crlm1cp(x)
 [delay]=crlm2cp(x)
[delay]=crlm3cp(x)
end;
elseif c==4;
disp('Dual pass-transistor
logic design style')
disp('------------------
-----------------')
disp(' ')
if d==1;
disp(' ')
[delay]=crlm1dp(x)
 elseif d==2;
[delay]=crlm2dp(x)
 elseif d==3;
[delay]=crlm3dp(x)
 else;
[delay]=crlm1dp(x)
 [delay]=crlm2dp(x)
[delay]=crlm3dp(x)
end;
 elseif c==0;
disp('Fully static CMOS
logic design style')
disp('------------------
--------')
disp(' ')
if d==1;
disp(' ')
[delay]=crlm1fs(x)
elseif d==2;
[delay]=crlm2fs(x)
elseif d==3;
[delay]=crlm3fs(x)
else;
[delay]=crlm1fs(x)
[delay]=crlm2fs(x)
[delay]=crlm3fs(x)
end;
disp('Domino CMOS
logic design style')
disp('------------------
-------')
disp(' ')
if d==1;
disp(' ')
[delay]=crlmdm(x)

elseif d==2;
[delay]=crlm2dm(x)
 elseif d==3;
[delay]=crlm3dm(x)
 else;
[delay]=crlm1dm(x)

[delay]=crlm2dm(x)
[delay]=crlm3dm(x)
end;
disp('Complementary
pass-transistor logic
design style')
disp('------------------
-----------------')
disp(' ')
if d==1;
disp(' ')
[delay]=crlm1cp(x)
 elseif d==2;
[delay]=crlm2cp(x)
 elseif d==3;
[delay]=crlm3cp(x)
 else;
[delay]=crlm1cp(x)
 [delay]=crlm2cp(x)
[delay]=crlm3cp(x)
end;
disp('Dual pass-transistor
logic design style')
disp('------------------
-----------------')
disp(' ')
if d==1;
disp(' ')
[delay]=crlm1dp(x)
 elseif d==2;
[delay]=crlm2dp(x)
 elseif d==3;
[delay]=crlm3dp(x)
 else;
[delay]=crlm1dp(x)
 [delay]=crlm2dp(x)
[delay]=crlm3dp(x)
end;
 end;
elseif b==3;
disp('Carry skip adder
architecture')
disp('------------------
-------')
disp(' ')
if c==1;
disp('Fully static CMOS
logic design style')
disp('------------------
--------')
disp(' ')
if d==1;
disp(' ')
disp(' ')
[delay]=cskm1fs(x)
elseif d==2;
[delay]=cskm2fs(x)
elseif d==3;
[delay]=cskm3fs(x)
else;
[delay]=cskm1fs(x)
[delay]=cskm2fs(x)
[delay]=cskm3fs(x)
end;
elseif c==2;
disp('Domino CMOS
logic design style')
disp('------------------
-------')
disp(' ')
if d==1;
disp(' ')
[delay]=cskm1dm(x)
elseif d==2;
[delay]=cskm2dm(x)
 elseif d==3;
[delay]=cskm3dm(x)
 else;
[delay]=cskm1dm(x)
 [delay]=cskm2dm(x)
[delay]=cskm3dm(x)
end;
elseif c==3;
disp('Complementary
pass-transistor logic
design style')
disp('------------------
-----------------')
disp(' ')
if d==1;
disp(' ')
[delay]=cskm1cp(x)
 elseif d==2;
[delay]=cskm2cp(x)
 elseif d==3;
[delay]=cskm3cp(x)
 else;
```

```
[delay]=cskm1cp(x)
[delay]=cskm2cp(x)
[delay]=cskm3cp(x)
end;
elseif c==4;
disp('Dual pass-transistor
logic design style')
disp('------------')
disp(' ')
if d==1;
disp(' ')
[delay]=cskm1dp(x)
 elseif d==2;
[delay]=cskm2dp(x)
 elseif d==3;
[delay]=cskm3dp(x)
 else;
[delay]=cskm1dp(x)
[delay]=cskm2dp(x)
[delay]=cskm3dp(x)
end;
 elseif c==0;
disp('Fully static CMOS
logic design style')
disp('------------')
disp(' ')
if d==1;
disp(' ')
[delay]=cskm1fs(x)
elseif d==2;
[delay]=cskm2fs(x)
elseif d==3;
[delay]=cskm3fs(x)
else;
[delay]=cskm1fs(x)
[delay]=cskm2fs(x)
[delay]=cskm3fs(x)
end;
disp('Domino CMOS
logic design style')
disp('------------')
disp(' ')
if d==1;
disp(' ')
[delay]=cskm1dm(x)
elseif d==2;
[delay]=cskm2dm(x)
 elseif d==3;
[delay]=cskm3dm(x)
 else;
[delay]=cskm1dm(x)
[delay]=cskm2dm(x)
[delay]=cskm3dm(x)
end;
disp('Complementary
pass-transistor logic
design style')
disp('------------')
disp(' ')
if d==1;
disp(' ')
[delay]=cskm1cp(x)
elseif d==2;
[delay]=cskm2cp(x)
 elseif d==3;
[delay]=cskm3cp(x)
else;
[delay]=cskm1cp(x)
 [delay]=cskm2cp(x)
[delay]=cskm3cp(x)
end;
disp('Dual pass-transistor
logic design style')
disp('------------')
disp(' ')
if d==1;
disp(' ')
[delay]=cskm1dp(x)
 elseif d==2;
[delay]=cskm2dp(x)
 elseif d==3;
[delay]=cskm3dp(x)
 else;
[delay]=cskm1dp(x)
[delay]=cskm2dp(x)
[delay]=cskm3dp(x)
end;
 end;
 elseif b==4;
disp('Carry look ahead
adder architecture')
disp('------------')
disp(' ')
if c==1;
disp('Fully static CMOS
logic design style')
disp('------------')
disp(' ')
if d==1;
disp(' ')
disp(' ')
[delay]=clam1fs(x)
elseif d==2;
[delay]=clam2fs(x)
elseif d==3;
[delay]=clam3fs(x)
else;
[delay]=clam1fs(x)
[delay]=clam2fs(x)
[delay]=clam3fs(x)
end;
elseif c==2;
disp('Domino CMOS
logic design style')
disp('------------')
disp(' ')
if d==1;
disp(' ')
[delay]=clam1dm(x)
elseif d==2;
[delay]=clam2dm(x)
 elseif d==3;
[delay]=clam3dm(x)
 else;
[delay]=clam1dm(x)
 [delay]=clam2dm(x)
[delay]=clam3dm(x)
end;
elseif c==3;
disp('Complementary
pass-transistor logic
design style')
disp('------------')
disp(' ')
if d==1;
disp(' ')
[delay]=clam1cp(x)
 elseif d==2;
[delay]=clam2cp(x)
 elseif d==3;
[delay]=clam3cp(x)
 else;
[delay]=clam1cp(x)
[delay]=clam2cp(x)
[delay]=clam3cp(x)
end;
elseif c==4;
disp('Dual pass-transistor
logic design style')
disp('------------')
disp(' ')
if d==1;
disp(' ')
[delay]=clam1dp(x)
 elseif d==2;
[delay]=clam2dp(x)
 elseif d==3;
[delay]=clam3dp(x)
 else;
[delay]=clam1dp(x)
[delay]=clam2dp(x)
[delay]=clam3dp(x)
end;
 elseif c==0;
disp('Fully static CMOS
logic design style')
disp('------------')
disp(' ')
if d==1;
disp(' ')
[delay]=clam1fs(x)
elseif d==2;
[delay]=clam2fs(x)
elseif d==3;
[delay]=clam3fs(x)
else;
[delay]=clam1fs(x)
[delay]=clam2fs(x)
[delay]=clam3fs(x)
end;
disp('Domino CMOS
logic design style')
disp('------------')
disp(' ')
if d==1;
disp(' ')
[delay]=clam1dm(x)
elseif d==2;
[delay]=clam2dm(x)
 elseif d==3;
[delay]=clam3dm(x)
 else;
[delay]=clam1dm(x)
[delay]=clam2dm(x)
[delay]=clam3dm(x)
end;
disp('Complementary
pass-transistor logic
design style')
disp('------------')
disp(' ')
if d==1;
disp(' ')
[delay]=clam1cp(x)
elseif d==2;
[delay]=clam2cp(x)
 elseif d==3;
[delay]=clam3cp(x)
else;
[delay]=clam1cp(x)
 [delay]=clam2cp(x)
[delay]=clam3cp(x)
end;
disp('Dual pass-transistor
logic design style')
disp('------------')
disp(' ')
if d==1;
disp(' ')
[delay]=clam1dp(x)
 elseif d==2;
[delay]=clam2dp(x)
 elseif d==3;
[delay]=clam3dp(x)
 else;
[delay]=clam1dp(x)
[delay]=clam2dp(x)
[delay]=clam3dp(x)
end;
 end;
elseif b==5;
disp('Conditional sum
adder architecture')
disp('------------')
disp(' ')
if c==1;
disp('Fully static CMOS
logic design style')
disp('------------')
disp(' ')
if d==1;
disp(' ')
disp(' ')
[delay]=cndm1fs(x)
elseif d==2;
[delay]=cndm2fs(x)
elseif d==3;
[delay]=cndm3fs(x)
else;
[delay]=cndm1fs(x)
[delay]=cndm2fs(x)
[delay]=cndm3fs(x)
end;
elseif c==2;
disp('Domino CMOS
logic design style')
disp('------------')
disp(' ')
if d==1;
disp(' ')
[delay]=cndm1dm(x)
elseif d==2;
[delay]=cndm2dm(x)
 elseif d==3;
[delay]=cndm3dm(x)
 else;
[delay]=cndm1dm(x)
 [delay]=cndm2dm(x)
[delay]=cndm3dm(x)
end;
elseif c==3;
disp('Complementary
pass-transistor logic
design style')
disp('------------')
```

```
disp(' ')
if d==1;
disp(' ')
[delay]=cndm1cp(x)
elseif d==2;
[delay]=cndm2cp(x)
elseif d==3;
[delay]=cndm3cp(x)
else;
[delay]=cndm1cp(x)
[delay]=cndm2cp(x)
[delay]=cndm3cp(x)
end;
elseif c==4;
disp('Dual pass-transistor
logic design style')
disp('-------------------
-------------------')
disp(' ')
if d==1;
disp(' ')
[delay]=cndm1dp(x)
elseif d==2;
[delay]=cndm2dp(x)
elseif d==3;
[delay]=cndm3dp(x)
else;
[delay]=cndm1dp(x)
[delay]=cndm2dp(x)
[delay]=cndm3dp(x)
end;
elseif c==0;
disp('Fully static CMOS
logic design style')
disp('-------------------')
disp(' ')
if d==1;
disp(' ')
[delay]=cndm1fs(x)
elseif d==2;
[delay]=cndm2fs(x)
elseif d==3;
[delay]=cndm3fs(x)
else;
[delay]=cndm1fs(x)
[delay]=cndm2fs(x)
[delay]=cndm3fs(x)
end;
disp('Domino CMOS
logic design style')
disp('-------------------')
disp(' ')
if d==1;
disp(' ')
[delay]=cndm1dm(x)

elseif d==2;
[delay]=cndm2dm(x)
elseif d==3;
[delay]=cndm3dm(x)
else;
[delay]=cndm1dm(x)
[delay]=cndm2dm(x)
[delay]=cndm3dm(x)
end;
disp('Complementary
pass-transistor logic
design style')
disp('-------------------')
disp(' ')
if d==1;
disp(' ')
[delay]=cndm1cp(x)
elseif d==2;
[delay]=cndm2cp(x)
elseif d==3;
[delay]=cndm3cp(x)
else;
[delay]=cndm1cp(x)
[delay]=cndm2cp(x)
[delay]=cndm3cp(x)
end;
disp('Dual pass-transistor
logic design style')
disp('-------------------')
disp(' ')
if d==1;
disp(' ')
[delay]=cndm1dp(x)
elseif d==2;
[delay]=cndm2dp(x)
elseif d==3;
[delay]=cndm3dp(x)
else;
[delay]=cndm1dp(x)
[delay]=cndm2dp(x)
[delay]=cndm3dp(x)
end;
end;
elseif b==0;
disp('Ripple carry adder
architecture')
disp('-------------------')
disp(' ')
if c==1;
disp('Fully static CMOS
logic design style')
disp('-------------------')
disp(' ')
if d==1;
disp(' ')
disp(' ')
[delay]=rcam1fs(x)
elseif d==2;
[delay]=rcam2fs(x)
elseif d==3;
[delay]=rcam3fs(x)
else;
[delay]=rcam1fs(x)
[delay]=rcam2fs(x)
[delay]=rcam3fs(x)
end;
elseif c==2;
disp('Domino CMOS
logic design style')
disp('-------------------')
disp(' ')
if d==1;
disp(' ')
[delay]=rcam1dm(x)
elseif d==2;
[delay]=rcam2dm(x)
elseif d==3;
[delay]=rcam3dm(x)
else;
[delay]=rcam1dm(x)
[delay]=rcam2dm(x)
[delay]=rcam3dm(x)
end;
elseif c==3;
disp('Complementary
pass-transistor logic
design style')
disp('-------------------')
disp(' ')
if d==1;
disp(' ')
[delay]=rcam1cp(x)
elseif d==2;
[delay]=rcam2cp(x)
elseif d==3;
[delay]=rcam3cp(x)
else;
[delay]=rcam1cp(x)
[delay]=rcam2cp(x)
[delay]=rcam3cp(x)
end;
elseif c==4
disp('Dual pass-transistor
logic design style')
disp('-------------------')
disp(' ')
if d==1;
disp(' ')
[delay]=rcam1dp(x)
elseif d==2;
[delay]=rcam2dp(x)
elseif d==3;
[delay]=rcam3dp(x)
else;
[delay]=rcam1dp(x)
[delay]=rcam2dp(x)
[delay]=rcam3dp(x)
end;
elseif c==0;
disp('Fully static CMOS
logic design style')
disp('-------------------')
disp(' ')
if d==1;
disp(' ')
[delay]=rcam1fs(x)
elseif d==2;
[delay]=rcam2fs(x)
elseif d==3;
[delay]=rcam3fs(x)
else;
[delay]=rcam1fs(x)
[delay]=rcam2fs(x)
[delay]=rcam3fs(x)
end;
disp('Domino CMOS
logic design style')
disp('-------------------
-------')
disp(' ')
if d==1;
disp(' ')
[delay]=rcam1dm(x)
elseif d==2;
[delay]=rcam2dm(x)
elseif d==3;
[delay]=rcam3dm(x)
else;
[delay]=rcam1dm(x)
[delay]=rcam2dm(x)
[delay]=rcam3dm(x)
end;
disp('Complementary
pass-transistor logic
design style')
disp('-------------------')
disp(' ')
if d==1;
disp(' ')
[delay]=rcam1cp(x)
elseif d==2;
[delay]=rcam2cp(x)
elseif d==3;
[delay]=rcam3cp(x)
else;
[delay]=rcam1cp(x)
[delay]=rcam2cp(x)
[delay]=rcam3cp(x)
end;
disp('Dual pass-transistor
logic design style')
disp('-------------------')
disp(' ')
if d==1;
disp(' ')
[delay]=rcam1dp(x)
elseif d==2;
[delay]=rcam2dp(x)
elseif d==3;
[delay]=rcam3dp(x)
else;
[delay]=rcam1dp(x)
[delay]=rcam2dp(x)
[delay]=rcam3dp(x)
end;
end;
disp('Carry select adder
architecture')
disp('-------------------')
disp(' ')
if c==1;
disp('Fully static CMOS
logic design style')
disp('-------------------')
disp(' ')
if d==1;
disp(' ')
disp(' ')
[delay]=crlm1fs(x)
elseif d==2;
[delay]=crlm2fs(x)
elseif d==3;
[delay]=crlm3fs(x)
else;
[delay]=crlm1fs(x)
[delay]=crlm2fs(x)
[delay]=crlm3fs(x)
end;
elseif c==2;
disp('Domino CMOS
logic design style')
disp('-------------------')
disp(' ')
if d==1;
disp(' ')
[delay]=crlm1dm(x)
elseif d==2;
[delay]=crlm2dm(x)
elseif d==3;
[delay]=crlm3dm(x)
else;
[delay]=crlm1dm(x)
```

```
[delay]=crlm2dm(x)
[delay]=crlm3dm(x)
end;
elseif c==3;
disp('Complementary
pass-transistor logic
design style')
disp('----------')
disp(' ')
if d==1;
disp(' ')
[delay]=crlm1cp(x)
 elseif d==2;
[delay]=crlm2cp(x)
 elseif d==3;
[delay]=crlm3cp(x)
 else;
[delay]=crlm1cp(x)
 [delay]=crlm2cp(x)
[delay]=crlm3cp(x)
end;
elseif c==4
disp('Dual pass-transistor
logic design style')
disp('----------')
disp(' ')
if d==1;
disp(' ')
[delay]=crlm1dp(x)
 elseif d==2;
[delay]=crlm2dp(x)
 elseif d==3;
[delay]=crlm3dp(x)
 else;
[delay]=crlm1dp(x)
 [delay]=crlm2dp(x)
[delay]=crlm3dp(x)
end;
 elseif c==0;
disp('Fully static CMOS
logic design style')
disp('----------')
disp(' ')
if d==1;
disp(' ')
[delay]=crlm1fs(x)
elseif d==2;
[delay]=crlm2fs(x)
elseif d==3;
[delay]=crlm3fs(x)
else;
[delay]=crlm1fs(x)
[delay]=crlm2fs(x)
[delay]=crlm3fs(x)
end;
disp('Domino CMOS
logic design style')
disp('----------')
disp(' ')
if d==1;
disp(' ')
[delay]=crlm1dm(x)
 elseif d==2;
[delay]=crlm2dm(x)
 elseif d==3;
[delay]=crlm3dm(x)
 else;
[delay]=crlm1dm(x)

[delay]=crlm2dm(x)
[delay]=crlm3dm(x)
end;
disp('Complementary
pass-transistor logic
design style')
disp('----------')
disp(' ')
if d==1;
disp(' ')
[delay]=crlm1cp(x)
 elseif d==2;
[delay]=crlm2cp(x)
 elseif d==3;
[delay]=crlm3cp(x)
 else;
[delay]=crlm1cp(x)
 [delay]=crlm2cp(x)
[delay]=crlm3cp(x)
end;
 disp('Dual pass-transistor
logic design style')
disp('----------')
disp(' ')
if d==1;
disp(' ')
[delay]=crlm1dp(x)
 elseif d==2;
[delay]=crlm2dp(x)
 elseif d==3;
[delay]=crlm3dp(x)
 else;
[delay]=crlm1dp(x)
 [delay]=crlm2dp(x)
[delay]=crlm3dp(x)
end;
 end;
 disp('Carry skip adder
architecture')
disp('----------')
disp(' ')
if c==1;
disp('Fully static CMOS
logic design style')
disp('----------')
disp(' ')
if d==1;
disp(' ')
disp(' ')
[delay]=cskm1fs(x)
elseif d==2;
[delay]=cskm2fs(x)
elseif d==3;
[delay]=cskm3fs(x)
else;
[delay]=cskm1fs(x)
[delay]=cskm2fs(x)
[delay]=cskm3fs(x)
end;
elseif c==2;
disp('Domino CMOS
logic design style')
disp('----------')
disp(' ')
if d==1;
disp(' ')
[delay]=cskm1dm(x)

elseif d==2;
[delay]=cskm2dm(x)
elseif d==3;
[delay]=cskm3dm(x)
 else;
[delay]=cskm1dm(x)
 [delay]=cskm2dm(x)
[delay]=cskm3dm(x)
end;
elseif c==3;
disp('Complementary
pass-transistor logic
design style')
disp('----------')
disp(' ')
if d==1;
disp(' ')
[delay]=cskm1cp(x)
 elseif d==2;
[delay]=cskm2cp(x)
 elseif d==3;
[delay]=cskm3cp(x)
 else;
[delay]=cskm1cp(x)
[delay]=cskm2cp(x)
[delay]=cskm3cp(x)
end;
elseif c==4;
disp('Dual pass-transistor
logic design style')
disp('----------
----------')
disp(' ')
if d==1;
disp(' ')
[delay]=cskm1dp(x)
 elseif d==2;
[delay]=cskm2dp(x)
 elseif d==3;
[delay]=cskm3dp(x)
 else;
[delay]=cskm1dp(x)
[delay]=cskm2dp(x)
[delay]=cskm3dp(x)
end;
 elseif c==0;
disp('Fully static CMOS
logic design style')
disp('----------')
disp(' ')
if d==1;
disp(' ')
[delay]=cskm1fs(x)
elseif d==2;
[delay]=cskm2fs(x)
elseif d==3;
[delay]=cskm3fs(x)
else;
[delay]=cskm1fs(x)
[delay]=cskm2fs(x)
[delay]=cskm3fs(x)
end;
disp('Domino CMOS
logic design style')
disp('----------')
disp(' ')
if d==1;
disp(' ')
[delay]=cskm1dm(x)
elseif d==2;
[delay]=cskm2dm(x)
 elseif d==3;
[delay]=cskm3dm(x)
 else;
[delay]=cskm1dm(x)

[delay]=cskm2dm(x)
[delay]=cskm3dm(x)
end;
disp('Complementary
pass-transistor logic
design style')
disp('----------')
disp(' ')
if d==1;
disp(' ')
[delay]=cskm1cp(x)
elseif d==2;
[delay]=cskm2cp(x)
 elseif d==3;
[delay]=cskm3cp(x)
else;
[delay]=cskm1cp(x)

[delay]=cskm2cp(x)
[delay]=cskm3cp(x)
end;
disp('Dual pass-transistor
logic design style')
disp('----------')
disp(' ')
if d==1;
disp(' ')
[delay]=cskm1dp(x)
 elseif d==2;
[delay]=cskm2dp(x)
 elseif d==3;
[delay]=cskm3dp(x)
 else;
[delay]=cskm1dp(x)
[delay]=cskm2dp(x)
[delay]=cskm3dp(x)
end;
end;
 disp('Carry look ahead
adder architecture')
disp('----------')
disp(' ')
if c==1;
disp('Fully static CMOS
logic design style')
disp('----------')
disp(' ')
if d==1;
disp(' ')
disp(' ')
[delay]=clam1fs(x)
elseif d==2;
[delay]=clam2fs(x)
elseif d==3;
[delay]=clam3fs(x)
else;
[delay]=clam1fs(x)
[delay]=clam2fs(x)
[delay]=clam3fs(x)
end;
elseif c==2;
disp('Domino CMOS
logic design style')
disp('----------')
disp(' ')
if d==1;
```

```
disp(' ')
[delay]=clam1dm(x)
elseif d==2;
[delay]=clam2dm(x)
elseif d==3;
[delay]=clam3dm(x)
else;
[delay]=clam1dm(x)
[delay]=clam2dm(x)
[delay]=clam3dm(x)
end;
elseif c==3;
disp('Complementary
pass-transistor logic
design style')
disp('----------')
disp(' ')
if d==1;
disp(' ')
[delay]=clam1cp(x)
elseif d==2;
[delay]=clam2cp(x)
elseif d==3;
[delay]=clam3cp(x)
else;
[delay]=clam1cp(x)
[delay]=clam2cp(x)
[delay]=clam3cp(x)
end;
elseif c==4;
disp('Dual pass-transistor
logic design style')
disp('----------')
disp(' ')
if d==1;
disp(' ')
[delay]=clam1dp(x)
elseif d==2;
[delay]=clam2dp(x)

elseif d==3;
[delay]=clam3dp(x)
else;
[delay]=clam1dp(x)
[delay]=clam2dp(x)
[delay]=clam3dp(x)
end;
elseif c==0;
disp('Fully static CMOS
logic design style')
disp('----------')
disp(' ')
if d==1;
disp(' ')
[delay]=clam1fs(x)
elseif d==2;
[delay]=clam2fs(x)
elseif d==3;
[delay]=clam3fs(x)
else;
[delay]=clam1fs(x)
[delay]=clam2fs(x)
[delay]=clam3fs(x)
end;
disp('Domino CMOS
logic design style')
disp('----------')
disp(' ')
if d==1;
disp(' ')
[delay]=cndm1dm(x)
elseif d==2;
[delay]=cndm2dm(x)
elseif d==3;
[delay]=cndm3dm(x)
else;
[delay]=cndm1dm(x)

[delay]=cndm2dm(x)
[delay]=cndm3dm(x)
end;
disp('Complementary
pass-transistor logic
design style')
disp('----------')
disp(' ')
if d==1;
disp(' ')
[delay]=cndm1cp(x)
elseif d==2;
[delay]=cndm2cp(x)
elseif d==3;
[delay]=cndm3cp(x)
else;
[delay]=cndm1cp(x)

[delay]=cndm2cp(x)
[delay]=cndm3cp(x)
end;
disp('Dual pass-transistor
logic design style')
disp('----------')
disp(' ')
if d==1;
disp(' ')
[delay]=cndm1dp(x)

elseif d==2;
[delay]=cndm2dp(x)

elseif d==3;
[delay]=cndm3dp(x)

else;
[delay]=cndm1dp(x)

[delay]=cndm2dp(x)
[delay]=cndm3dp(x)
end;
end;
disp('Conditional sum
adder architecture')
disp('----------')
disp(' ')
if c==1;
disp('Fully static CMOS
logic design style')
disp('----------')
disp(' ')
if d==1;
disp(' ')
disp(' ')
[delay]=cndm1fs(x)
elseif d==2;
[delay]=cndm2fs(x)
elseif d==3;
[delay]=cndm3fs(x)
else;
[delay]=cndm1fs(x)
[delay]=cndm2fs(x)
[delay]=cndm3fs(x)
end;
elseif c==2;
disp('Domino CMOS
logic design style')
disp('----------')
disp(' ')
if d==1;
disp(' ')
[delay]=cndm1dm(x)
elseif d==2;
[delay]=cndm2dm(x)
elseif d==3;
[delay]=cndm3dm(x)
else;
[delay]=cndm1dm(x)
[delay]=cndm2dm(x)
[delay]=cndm3dm(x)
end;
elseif c==3;
disp('Complementary
pass-transistor logic
design style')
disp('----------')
disp(' ')
if d==1;
disp(' ')
[delay]=cndm1cp(x)
elseif d==2;
[delay]=cndm2cp(x)
elseif d==3;
[delay]=cndm3cp(x)
else;
[delay]=cndm1cp(x)
[delay]=cndm2cp(x)
[delay]=cndm3cp(x)
end;
elseif c==4;
disp('Dual pass-transistor
logic design style')
disp('----------')
disp(' ')
if d==1;
disp(' ')
[delay]=cndm1dp(x)
elseif d==2;
[delay]=cndm2dp(x)

elseif d==3;
[delay]=cndm3dp(x)

else;
[delay]=cndm1dp(x)
[delay]=cndm2dp(x)
[delay]=cndm3dp(x)
end;
elseif c==0;
disp('Fully static CMOS
logic design style')
disp('----------')
disp(' ')
if d==1;
disp(' ')
[delay]=cndm1fs(x)
elseif d==2;
[delay]=cndm2fs(x)
elseif d==3;
[delay]=cndm3fs(x)
else;
[delay]=cndm1fs(x)
[delay]=cndm2fs(x)
[delay]=cndm3fs(x)
end;
disp('Domino CMOS
logic design style')
disp('----------')
disp(' ')
if d==1;
disp(' ')
[delay]=cndm1dm(x)

elseif d==2;
[delay]=cndm2dm(x)

elseif d==3;
[delay]=cndm3dm(x)
else;
[delay]=cndm1dm(x)
[delay]=cndm2dm(x)
[delay]=cndm3dm(x)
end;
disp('Complementary
pass-transistor logic
design style')
disp('----------')
disp(' ')
if d==1;
disp(' ')
[delay]=cndm1cp(x)
elseif d==2;
[delay]=cndm2cp(x)
elseif d==3;
[delay]=cndm3cp(x)
else;
[delay]=cndm1cp(x)
[delay]=cndm2cp(x)
[delay]=cndm3cp(x)
end;
disp('Dual pass-transistor
logic design style')
disp('----------')
disp(' ')
if d==1;
disp(' ')
[delay]=cndm1dp(x)
elseif d==2;
[delay]=cndm2dp(x)
elseif d==3;
[delay]=cndm3dp(x)
else;
[delay]=cndm1dp(x)
[delay]=cndm2dp(x)
[delay]=cndm3dp(x)
end;
end;
end;
```

APPENDIX A5

Functions called in program listed in *Appendix A4*

These functions are called in program listed in ***Appendix A4*** and are used to compute the propagation delay using Delay Model

```
function[
delay]=clam1dp(x)
format;
p= -0.0002*x^2 +
0.0463*x + 1.712;
p
d=8.50E-11;
if x==4
n=12
elseif x==8;
n=16;
elseif x==16;
n=20;
elseif x==32;
n=24;
elseif x==64;
n=28;
else
disp('Adder size not
appropriate')
end
f=3.89;
delay=p*d*n*f;
disp(' ')
disp('Propagation delay
for carry look ahead
adder (W=1.8um)dual
pass-transistor logic ,in
seconds, is ')
```

```
function[
delay]=clam1dm(x)
format;
p=0.0003*x^2 +
0.0084*x + 2.0007;
p
d=8.5E-11;
if x==4;
n=10
elseif x==8;
n=14;
elseif x==16;
n=18;
elseif x==32;
n=22;
elseif x==64;
n=26;
else
disp('Adder size not
appropriate')
end
f=3.3;
delay=p*d*n*f;
disp(' ')
disp('Propagation delay
for carry look ahead
adder
(W=1.8um)domino
CMOS logic ,in seconds,
is ')
```

```
function[
delay]=clam2dp(x)
format;
p= -0.0003*x^2 +
0.0412*x + 1.5519;
p
d=7.00E-11;
if x==4
n=12
elseif x==8;
n=16;
elseif x==16;
n=20;
elseif x==32;
n=24;
elseif x==64;
n=28;
else
disp('Adder size not
appropriate')
end
f=3.89;
delay=p*d*n*f;
disp(' ')
disp('Propagation delay
for carry look ahead
adder (W=3.6um)dual
pass-transistor logic ,in
seconds, is ')
```

```
function[
delay]=clam2dm(x)
format;
p= 0.0002*x^2 +
0.0057*x + 1.8165;
p
d=7.0E-11;
if x==4;
n=10
elseif x==8;
n=14;
elseif x==16;
n=18;
elseif x==32;
n=22;
elseif x==64;
n=26;
else
disp('Adder size not
appropriate')
end
f=3.3;
delay=p*d*n*f;
disp(' ')
disp('Propagation delay
for carry look ahead
adder
(W=3.6um)domino
CMOS logic ,in seconds,
is ')
```

```
function[
delay]=clam1cp(x)
format;
p=-0.0002*x^2 +
0.042*x + 1.2293;
p
d=8.5e-11;
if x==4
n=12
elseif x==8;
n=16;
elseif x==16;
n=20;
elseif x==32;
n=24;
elseif x==64;
n=28;
else
disp('Adder size not
appropriate')
end
f=3.25
delay=p*d*n*f;
disp(' ')
disp('Propagation delay
for carry look ahead
adder (W=1.8um)
complementary pass-
transistor logic ,in
seconds, is ')
```

```
function[
delay]=clam1fs(x)
format;
p= -4E-05*x^2 +
0.0108*x + 0.7771;
p
d=8.5e-11;
if x==4
n=12
elseif x==8;
n=16;
elseif x==16;
n=20;
elseif x==32;
n=24;
elseif x==64;
n=28;
else
disp('Adder size not
appropriate')
end
f=4.66;
delay=p*d*n*f;
disp(' ')
disp('Propagation delay
for carry look ahead
adder (W=1.8um) fully
static CMOS logic ,in
seconds, is ')
```

```
function[
delay]=clam2cp(x)
format;
p= -0.0002*x^2 +
0.0327*x + 1.1219;
p
d=7.0e-11;
if x==4
n=12
elseif x==8;
n=16;
elseif x==16;
n=20;
elseif x==32;
n=24;
elseif x==64;
n=28;
else
disp('Adder size not
appropriate')
end
f=3.25
delay=p*d*n*f;
disp(' ')
disp('Propagation delay
for carry look ahead
adder (W=3.6um)
complementary pass-
transistor logic ,in
seconds, is ')
```

```
function[
delay]=clam2fs(x)
format;
p=-4E-05*x^2 +
0.0086*x + 0.7345;
p
d=7e-11;
if x==4
n=12
elseif x==8;
n=16;
elseif x==16;
n=20;
elseif x==32;
n=24;
elseif x==64;
n=28;
else
disp('Adder size not
appropriate')
end
f=4.66;
delay=p*d*n*f;
disp(' ')
disp('Propagation delay
for carry look ahead
adder (W=3.6um) fully
static CMOS logic ,in
seconds, is ')
```

```
function[
delay]=clam3dp(x)
format;
p= -0.0003*x^2 +
0.0377*x + 1.5723;
p
d=6.00E-11;
if x==4
n=12
elseif x==8;
n=16;
elseif x==16;
n=20;
elseif x==32;
n=24;
elseif x==64;
n=28;
else
disp('Adder size not
appropriate')
end
f=3.89;
delay=p*d*n*f;
disp(' ')
disp('Propagation delay
for carry look ahead
adder (W=6um)dual
pass-transistor logic ,in
seconds, is ')

function[
delay]=clam3cp(x)
format;
p= -0.0002*x^2 +
0.024*x + 1.1727;
p
d=6e-11;
if x==4
n=12
elseif x==8;
n=16;
elseif x==16;
n=20;
elseif x==32;
n=24;
elseif x==64;
n=28;
else
disp('Adder size not
appropriate')
end
f=3.25
delay=p*d*n*f;
disp(' ')
disp('Propagation delay
for carry look ahead
adder (W=6um)
complementary pass-
transistor logic ,in
seconds, is ')

function[
delay]=clam3dm(x)
format;
p=0.0001*x^2 +
0.0031*x + 1.8349;
p
d=6.0E-11;
if x==4;
n=10
elseif x==8;
n=14;
elseif x==16;
n=18;
elseif x==32;
n=22;
elseif x==64;
n=26;
else
disp('Adder size not
appropriate for carry
look ahead adder')
end
f=3.3;
delay=p*d*n*f;
disp(' ')
disp('Propagation delay
for carry look ahead
adder (W=6um)domino
CMOS logic ,in seconds,
is ')

function[
delay]=cndm1dp(x)
format;
p= 1E-04*x^2 +
0.0009*x + 0.8327;
p
d=8.5E-11;
if x==4;
n=11;
elseif x>4;
n=11+(((x/4)-1)*3)
 end;
f=5.2;
delay=p*d*n*f;
disp(' ')
disp('Propagation delay
for conditional sum
adder (W=1.8um)dual
pass-transistor logic ,in
seconds, is ')

function[
delay]=cndm1cp(x)
format;
p=-2E-05*x^2 +
0.0311*x + 0.9936;
p
d=8.5e-11;
if x==4;
n=11;
elseif x>4;
n=11+(((x/4)-1)*3)
 end;
f=3.11;
delay=p*d*n*f;
disp(' ')
disp('Propagation delay
for conditional sum
adder (W=1.8um)
complementary pass-
transistor logic ,in
seconds, is ')

function[
delay]=cndm1dm(x)
format;
p= -0.0003*x^2 +
0.0862*x + 1.6198;
p
d=8.5e-11;
if x==4;
n=10;
elseif x>4;
n=10+(((x/4)-1)*2);
 end;
f=2.77;
delay=p*d*n*f;
disp(' ')
disp('Propagation delay
for conditional sum
adder
(W=1.8um)domino
CMOS logic ,in seconds,
is ')

function[
delay]=cndm1fs(x)
format;
p=-3E-05*x^2 +
0.0093*x + 1.2239;
p
d=8.5e-11;
if x==4
n=10
elseif x>4;
n=10+((x/4)-1)*2;
end
f=5.82;
delay=p*d*n*f;
disp(' ')
disp('Propagation delay
for conditional sum
adder (W=1.8um) fully
static CMOS logic ,in
seconds, is ')

function[
delay]=cndm2dp(x)
format;
p=6E-05*x^2 -
0.0006*x + 0.71;
p
d=7E-11;
if x==4;
n=11;
elseif x>4;
n=11+(((x/4)-1)*3)
 end;
f=5.2;
delay=p*d*n*f;
disp(' ')
disp('Propagation delay
for conditional sum
adder (W=3.6um)dual
pass-transistor logic ,in
seconds, is ')

function[
delay]=cndm2cp(x)
format;
p=-9E-05*x^2 +
0.0248*x + 0.8423;
p
d=7e-11;
if x==4;
n=11;
elseif x>4;
n=11+(((x/4)-1)*3)
 end;
f=3.11;
delay=p*d*n*f;
disp(' ')
disp('Propagation delay
for conditional sum
adder (W=3.6um)
complementary pass-
transistor logic ,in
seconds, is ')

function[
delay]=cndm2dm(x)
format;
p= -0.0003*x^2 +
0.059*x + 1.6504;
p
d=7e-11;
if x==4;
n=10;
elseif x>4;
n=10+(((x/4)-1)*2);
 end;
f=2.77;
delay=p*d*n*f;
disp(' ')
disp('Propagation delay
for conditional sum
adder
(W=3.6um)domino
CMOS logic ,in seconds,
is ')

function[
delay]=cndm2fs(x)
format;
p= 4E-07*x^2 +
0.0037*x + 1.1322;
p
d=7e-11;
if x==4
n=10
elseif x>4;
n=10+((x/4)-1)*2;
end
f=5.82;
delay=p*d*n*f;
disp(' ')
disp('Propagation delay
for conditional sum
adder (W=3.6um) fully
```

```
static CMOS logic ,in
seconds, is ')

function[
delay]=cndm3dp(x)
format;
p=6E-05*x^2 -
0.0019*x + 0.7166;
p
d=6.00E-11;
if x==4;
n=11;
elseif x>4;
n=11+(((x/4)-1)*3)
 end;
f=5.2;
delay=p*d*n*f;
disp(' ')
disp('Propagation delay
for conditional sum
adder (W=6um)dual
pass-transistor logic ,in
seconds, is ')

function[
delay]=cndm3cp(x)
format;
p= -0.0002*x^2 +
0.028*x + 0.8234;
p
d=6e-11;
if x==4;
n=11;
elseif x>4;
n=11+(((x/4)-1)*3)
 end;
f=3.11;
delay=p*d*n*f;
disp(' ')
disp('Propagation delay
for conditional sum
adder (W=6um)
complementary pass-
transistor logic ,in
seconds, is ')

function[
delay]=cndm3dm(x)
format;
p= -0.0003*x^2 +
0.0494*x + 1.6615;
p
d=6e-11;
if x==4;
n=10;
elseif x>4;
n=10+(((x/4)-1)*2);
 end;
f=2.77;
delay=p*d*n*f;
disp(' ')
disp('Propagation delay
for conditional sum
adder (W=6um)domino
CMOS logic ,in seconds,
is ')

function[
delay]=cndm3fs(x)
format;
p= 2E-05*x^2 + 0.001*x
+ 1.1277;
p
d=6e-11;
if x==4
n=10
elseif x>4;
n=10+((x/4)-1)*2;
end
f=5.82;
delay=p*d*n*f;
disp(' ')
disp('Propagation delay
for conditional sum
adder (W=6um) fully
static CMOS logic ,in
seconds, is ')

function[
delay]=crlm1dp(x)
format;
p= -0.0004*x^2 +
0.0732*x + 1.1767;
p
d=8.5E-11;
if x==4;
n=20;
elseif x>4;
n=16+(((x/4)-1)*4)
 end;
f=2.63;
delay=p*d*n*f;
disp(' ')
disp('Propagation delay
for carry select adder
(W=1.8um)dual pass-
transistor logic ,in
seconds, is ')

function[
delay]=crlm1cp(x)
format;
p= -0.0003*x^2 +
0.0592*x + 0.9414;
p
d=8.5E-11;
if x==4;
n=17;
elseif x>4;
n=13+(((x/4)-1)*4)
 end;
f=2.66;
delay=p*d*n*f;
disp(' ')
disp('Propagation delay
for carry select adder
(W=1.8um)
complementary pass-
transistor logic ,in
seconds, is ')

function[
delay]=crlm1dm(x)
format;
p=-7E-05*x^2 +
0.0486*x + 1.2572;
p
d=8.5E-11;
if x==4;
n=20;
elseif x>4;
n=16+(((x/4)-1)*4);
 end;
f=3.21;
delay=p*d*n*f;
disp(' ')
disp('Propagation delay
for carry select adder
(W=1.8um)domino
CMOS logic ,in seconds,
is ')

function[
delay]=crlm1fs(x)
format;
p= -0.0003*x^2 +
0.0353*x + 0.9831;
p
d=8.5e-11;
if x==4
n=10
elseif x>4;
n=8+((x/4)-1)*4;
end
f=4;
delay=p*d*n*f;
disp(' ')
disp('Propagation delay
for carry select adder
(W=1.8um) fully static
CMOS logic ,in seconds,
is ')
function[
delay]=crlm2dp(x)
format;
p= -0.0003*x^2 +
0.0522*x + 1.1804;
p
d=7E-11;
if x==4;
n=20;
elseif x>4;
n=16+(((x/4)-1)*4)
 end;
f=2.63;
delay=p*d*n*f;
disp(' ')
disp('Propagation delay
for carry select adder
(W=3.6um)dual pass-
transistor logic ,in
seconds, is ')

function[
delay]=crlm2cp(x)
format;
p= -0.0004*x^2 +
0.0554*x + 0.8059;
p
d=7E-11;
if x==4;
n=17;
elseif x>4;
n=13+(((x/4)-1)*4)
 end;
f=2.66;
delay=p*d*n*f;
disp(' ')
disp('Propagation delay
for carry select adder
(W=3.6um)
complementary pass-
transistor logic ,in
seconds, is ')

function[
delay]=crlm2dm(x)
format;
p=-0.0001*x^2 +
0.0371*x + 1.2016;
p
d=7E-11;
if x==4;
n=20;
elseif x>4;
n=16+(((x/4)-1)*4);
 end;
f=3.21;
delay=p*d*n*f;
disp(' ')
disp('Propagation delay
for carry select adder
(W=3.6um)domino
CMOS logic ,in seconds,
is ')

function[
delay]=crlm2fs(x)
format;
p= -0.0002*x^2 +
0.0225*x + 0.9893;
p
d=7e-11;
if x==4
n=10
elseif x>4;
n=8+((x/4)-1)*4;
end
f=4;
delay=p*d*n*f;
disp(' ')
disp('Propagation delay
for carry select adder
(W=3.6um) fully static
CMOS logic ,in seconds,
is ')

function[
delay]=crlm3dp(x)
format;
p=-0.0004*x^2 +
0.0482*x + 1.2125;
p
d=6.00E-11;
if x==4;
n=20;
elseif x>4;
n=16+(((x/4)-1)*4)
 end;
f=2.63;
delay=p*d*n*f;
disp(' ')
disp('Propagation delay
for carry select adder
(W=6um)dual pass-
transistor logic ,in
seconds, is ')

function[
delay]=crlm3cp(x)
format;
p= -0.0004*x^2 +
0.0478*x + 0.8456;
p
d=6E-11;
if x==4;
n=17;
elseif x>4;
n=13+(((x/4)-1)*4)
 end;
f=2.66;
```

```
delay=p*d*n*f;
disp(' ')
disp('Propagation delay
for carry select adder
(W=6um)
complementary pass-
transistor logic ,in
seconds, is ')

function[
delay]=crlm3dm(x)
format;
p= -0.0002*x^2 +
0.0354*x + 1.1928;
p
d=6E-11;
if x==4;
n=20;
elseif x>4;
n=16+(((x/4)-1)*4);
 end;
f=3.21;
delay=p*d*n*f;
disp(' ')
disp('Propagation delay
for carry select adder
(W=6um)domino
CMOS logic ,in seconds,
is ')

function[
delay]=crlm3fs(x)
format;
p= -0.0001*x^2 +
0.0157*x + 0.9877;
p
d=6e-11;
if x==4
n=10
elseif x>4;
n=8+((x/4)-1)*4;
end
f=4;
delay=p*d*n*f;
disp(' ')
disp('Propagation delay
for carry select adder
(W=6um) fully static
CMOS logic ,in seconds,
is ')

function[
delay]=cslm1dp(x)
format;
p= -0.0003*x^2 +
0.0551*x + 1.4782;
p
d=8.5E-11;
if x==4;
n=19;
elseif x>4;
n=19+(((x/4)-2)*4)+16
 end;
f=2.5;
delay=p*d*n*f;
disp(' ')
disp('Propagation delay
for carry skip adder
(W=1.8um)dual pass-
transistor logic ,in
seconds, is ')

function[
delay]=cslm1cp(x)
format;
p= -0.0002*x^2 +
0.0497*x + 1.384;
p
d=8.5E-11;
if x==4;
n=16;
elseif x>4;
n=16+(((x/4)-2)*4)+10
 end;
f=2.36;
delay=p*d*n*f;
disp(' ')
disp('Propagation delay
for carry skip adder
(W=1.8um)
complementary pass-
transistor logic ,in
seconds, is ')

function[
delay]=cslm1dm(x)
format;
p= -7E-05*x^2 +
0.0317*x + 1.5844;
p
d=8.5E-11;
if x==4;
n=20;
elseif x>4;
n=20+(((x/4)-2)*4)+18;
 end;
f=2.42;
delay=p*d*n*f;
disp(' ')
disp('Propagation delay
for carry skip adder
(W=1.8um)domino
CMOS logic ,in seconds,
is ')

function[
delay]=cslm1fs(x)
format;
p=-3E-05*x^2 +
0.0093*x + 1.2239;
p
d=8.5e-11;
if x>4
n=(12)+((((x-4)/4)-
1)*4)+10;
else
n=12;
end
f=3.27;
delay=p*d*n*f;
disp(' ')
n
disp('Propagation delay
for carry skip adder
(W=1.8um) fully static
CMOS logic ,in seconds,
is ')

function[
delay]=cslm2dp(x)
format;
p= -0.0002*x^2 +
0.0365*x + 1.3278;
p
d=7E-11;
if x==4;
n=19;
elseif x>4;
n=19+(((x/4)-2)*4)+16
 end;
f=2.5;
delay=p*d*n*f;
disp(' ')
disp('Propagation delay
for carry skip adder
(W=3.6um)dual pass-
transistor logic ,in
seconds, is ')

function[
delay]=cslm2cp(x)
format;
p= -0.0003*x^2 +
0.0394*x + 1.2139;
p
d=7E-11;
if x==4;
n=16;
elseif x>4;
n=16+(((x/4)-2)*4)+10
 end;
f=2.36;
delay=p*d*n*f;
disp(' ')
disp('Propagation delay
for carry skip adder
(W=3.6um)
complementary pass-
transistor logic ,in
seconds, is ')

function[
delay]=cslm2dm(x)
format;
p= -7E-05*x^2 +
0.0317*x + 1.5844;
p
d=7E-11;
if x==4;
n=20;
elseif x>4;
n=20+(((x/4)-2)*4)+18;
 end;
f=2.42;
delay=p*d*n*f;
disp(' ')
disp('Propagation delay
for carry skip adder
(W=3.6um)domino
CMOS logic ,in seconds,
is ')

function[
delay]=cslm2fs(x)
format;
p= 4E-07*x^2 +
0.0037*x + 1.1322;
p
d=7e-11;
if x>4
n=(12)+((((x-4)/4)-
1)*4)+10;
else
n=12
end
f=3.27;
delay=p*d*n*f;
disp(' ')
n
disp('Propagation delay
for carry skip adder
(W=3.6um) fully static
CMOS logic ,in seconds,
is ')

function[
delay]=cslm3dp(x)
format;
p= -0.0002*x^2 +
0.0386*x + 1.3203;
p
d=6.00E-11;
if x==4;
n=19;
elseif x>4;
n=19+(((x/4)-2)*4)+16
 end;
f=2.5;
delay=p*d*n*f;
disp(' ')
disp('Propagation delay
for carry skip adder
(W=6um)dual pass-
transistor logic ,in
seconds, is ')

function[
delay]=cslm3cp(x)
format;
p= -0.0003*x^2 +
0.0374*x + 1.2156;
p
d=6E-11;
if x==4;
n=16;
elseif x>4;
n=16+(((x/4)-2)*4)+10
 end;
f=2.36;
delay=p*d*n*f;
disp(' ')
disp('Propagation delay
for carry skip adder
(W=6um)
complementary pass-
transistor logic ,in
seconds, is ')

function[
delay]=cslm3dm(x)
format;
p= -9E-05*x^2 +
0.0193*x + 1.4817;
p
d=6E-11;
if x==4;
n=20;
elseif x>4;
n=20+(((x/4)-2)*4)+18;
 end;
f=2.42;
delay=p*d*n*f;
disp(' ')
disp('Propagation delay
for carry skip adder
(W=6um)domino
CMOS logic ,in seconds,
is ')

function[
delay]=cslm3fs(x)
```

```
format;
p = 2E-05*x^2 + 0.001*x
+ 1.1277;
p
d = 6e-11;
if x>4
n=(12)+((((x-4)/4)-
1)*4)+10;
else
n=12
end
f= 3.27;
delay=p*d*n*f;
disp(' ')
n
disp('Propagation delay
for carry skip adder
(W'=6um) fully static
CMOS logic ,in seconds,
is ')

function[
delay]=rcam1dp(x)
format;
p= -0.0002*x^2 +
0.0286*x + 1.6153;
p
d=8.5E-11;
n=(x*4);
f=2.42;
delay=p*d*n*f;
disp(' ')
disp('Propagation delay
for ripple carry adder
(W=1.8um)dual pass-
transistor logic ,in
seconds, is ')

function[
delay]=rcam1cp(x)
format;
p= -0.0002*x^2 +
0.0272*x + 1.1184;
p
d=8.5E-11;
n=(x*3)+1
f=2.66;
delay=p*d*n*f;
disp(' ')
disp('Propagation delay
for ripple carry adder
(W=1.8um)
complementary pass-
transistor logic ,in
seconds, is ')

function[
delay]=rcam1dm(x)
format;
p= -0.0001*x^2 +
0.0343*x + 1.8295;
p
d=8.5E-11;
n=x*4;
f=2.44;
delay=p*d*n*f;
disp(' ')
disp('Propagation delay
for ripple carry adder
(W=1.8um)domino
CMOS logic ,in seconds,
is ')

function[
delay]=rcam1fs(x);
format;
p=-0.0002*x^2 +
0.0438*x + 1.5221;
p
d=8.5E-11;
n=2*x;
f=3.11;
delay=p*d*n*f;
disp(' ')
disp('Propagation delay
for ripple carry adder
(W=1.8um) fully static
CMOS logic ,in seconds,
is ')

function[
delay]=rcam2dp(x)
format;
p= -5E-05*x^2 +
0.0127*x + 1.5937;
p
d=7E-11;
n=(x*4);
f=2.42;
delay=p*d*n*f;
disp(' ')
disp('Propagation delay
for ripple carry adder
(W=3.6um)dual pass-
transistor logic ,in
seconds, is ')

function[
delay]=rcam2cp(x)
format;
p= 5E-05*x^2 +
0.0094*x + 1.1236;
p
d=7E-11;
n=(x*3)+1
f=2.66;
delay=p*d*n*f;
disp(' ')
disp('Propagation delay
for ripple carry adder
(W=3.6um)
complementary pass-
transistor logic ,in
seconds, is ')

function[
delay]=rcam2dm(x)
format;
p= -0.0001*x^2 +
0.0221*x + 1.7415;
p
d=7E-11;
n=x*4;
f=2.44;
delay=p*d*n*f;
disp(' ')
disp('Propagation delay
for ripple carry adder
(W=3.6um)domino
CMOS logic ,in seconds,
is ')

function[
delay]=rcam2fs(x);
format;
p= -0.0002*x^2 +
0.0321*x + 1.4721;
p
d=7e-11 ;
n=2*x;
f=3.11;
delay=p*d*n*f;
disp(' ')
disp('Propagation delay
for ripple carry adder
(W=3.6um) fully static
CMOS logic ,in seconds,
is ')

function[
delay]=rcam3dp(x)
format;
p= -1E-04*x^2 +
0.0152*x + 1.5436;
p
d=6.00E-11;
n=(x*4);
f=2.42;
delay=p*d*n*f;
disp(' ')
disp('Propagation delay
for ripple carry adder
(W=6um)dual pass-
transistor logic ,in
seconds, is ')

function[
delay]=rcam3cp(x)
format;
p= -9E-06*x^2 +
0.0102*x + 1.1071;
p
d=6E-11;
n=(x*3)+1
f=2.66;
delay=p*d*n*f;
disp(' ')
disp('Propagation delay
for ripple carry adder
(W=6um)
complementary pass-
transistor logic ,in
seconds, is ')

function[
delay]=rcam3dm(x)
format;
p= -0.0001*x^2 +
0.0183*x + 1.7377;
p
d=6E-11;
n=x*4;
f=2.44;
delay=p*d*n*f;
disp(' ')
disp('Propagation delay
for ripple carry adder
(W=6um)domino
CMOS logic ,in seconds,
is ')

function[
delay]=rcam3fs(x);
format;
p= -0.0002*x^2 +
0.0288*x + 1.4523;
p
d=6e-11 ;
n=2*x;
f=3.11;
delay=p*d*n*f;
disp(' ')
disp('Propagation delay
for ripple carry adder
(W=6um) fully static
CMOS logic ,in seconds,
is ')
```

APPENDIX A6

Output of the program listed in *Appendix A1 with* the input as given below:

Input→

```
**************************************************************************
*
(Adder size) ---------- x =52     (52-bit)
(Adder architecture)--- b = 0     (for all five adder architectures)
(Logic design style)--- c =0      (for all four logic design styles)
(Transistor size)-------- d =0    (for all three transistor sizes)
**************************************************************************
*
```

Propagation delay results

Ripple carry adder architecture

Fully static CMOS logic design style

Propagation delay for ripple carry adder (W=1.8um) fully static CMOS logic ,in seconds, is

delay =

8.9246e-008

Propagation delay for ripple carry adder (W=3.6um) fully static CMOS logic ,in seconds, is

delay =

5.7322e-008

Propagation delay for ripple carry adder (W=6um) fully static CMOS logic ,in seconds, is

delay =

4.6273e-008

Domino CMOS logic design style

Propagation Delay for ripple carry adde (W=1.8um) Domino CMOS logic ,in seconds, is

delay =

1.3775e-007

Propagation Delay for ripple carry adder (W=3.6um) Domino CMOS logic ,in seconds, is

delay =

9.0434e-008

Propagation Delay for ripple carry adder (W=6um) Domino CMOS logic ,in seconds, is

delay =

7.1648e-008

Complementary pass-transistor logic design style

Propagation Delay for ripple carry adder (W=1.8um) Complementary pass-transistor CMOS logic ,in seconds, is

delay =

7.3779e-008

Propagation Delay for ripple carry adder (W=3.6um) Complementary pass-transistor CMOS logic ,in seconds, is

delay =

5.1465e-008

Propagation Delay for ripple carry adder (W=6um) Complementary pass-transistor CMOS logic ,in seconds, is

delay =

4.0427e-008

Dual pass-transistor logic design style

Propagation Delay for ripple carry adder (W=1.8um) Dual pass-transistor CMOS logic ,in seconds, is

delay =

1.0928e-007

Propagation Delay for ripple carry adder (W=3.6um) Dual pass-transistor CMOS logic ,in seconds, is

delay =

7.4392e-008

Propagation Delay for ripple carry adder (W=6um) Dual pass-transistor CMOS logic ,in seconds, is

delay =

6.1608e-008

Carry select adder architecture

Fully static CMOS logic design style

Propagation Delay for carry select adder (W=1.8um) fully static CMOS logic ,in seconds, is

delay =

3.7824e-008

Propagation Delay for carry select adder (W=3.6um) fully static CMOS logic ,in seconds, is

delay =

2.5255e-008

Propagation Delay for carry select adder (W=6um) fully static CMOS logic ,in seconds, is

delay =

2.0376e-008

Domino CMOS logic design style

Propagation Delay for carry select adder (W=1.8um) Domino CMOS logic ,in seconds, is

delay =

6.2065e-008

Propagation Delay for carry select adder (W=3.6um) Domino CMOS logic ,in seconds, is

delay =

4.0063e-008

Propagation Delay for carry select adder (W=6um) Domino CMOS logic ,in seconds, is

delay =

3.0799e-008

Complementary pass-transistor logic design style

Propagation Delay for carry select adder (W=1.8um) Complementary pass-transistor CMOS logic ,in seconds, is

delay =

4.2632e-008

Propagation Delay for carry select adder (W=3.6um) Complementary pass-transistor CMOS logic ,in seconds, is

delay =

2.6302e-008

Propagation Delay for carry select adder (W=6um) Complementary pass-transistor CMOS logic ,in seconds, is

delay =

2.0862e-008

Dual pass-transistor logic design style

Propagation Delay for carry select adder (W=1.8um) Dual pass-transistor CMOS logic ,in seconds, is

delay =

5.3433e-008

Propagation Delay for carry select adder (W=3.6um) Dual pass-transistor CMOS logic ,in seconds, is

delay =

3.4230e-008

Propagation Delay for carry select adder (W=6um) Dual pass-transistor CMOS logic ,in seconds, is

delay =

2.6973e-008

Carry skip adder architecture

Fully static CMOS logic design style

Propagation Delay for carry skip adder(W=1.8um) fully static CMOS logic ,in seconds, is

delay =

3.0322e-008

Propagation Delay for carry skip adder(W=3.6um) fully static CMOS logic ,in seconds, is

delay =

2.0425e-008

Propagation Delay for carry skip adder (W=6um) fully static CMOS logic ,in seconds, is

delay =

1.8023e-008

Domino CMOS logic design style

-

Propagation Delay for carry skip adder (W=1.8um) Domino CMOS logic ,in seconds, is

delay =

5.2432e-008

Propagation Delay for carry skip adder (W=3.6um) Domino CMOS logic ,in seconds, is

delay =

3.4845e-008

Propagation Delay for carry skip adder (W=6um) Domino CMOS logic ,in seconds, is

delay =

2.7238e-008

Complementary pass-transistor logic design style

Propagation Delay for carry skip adder (W=1.8um) Complementary pass-transistor logic ,in seconds, is

delay =

4.6010e-008

Propagation Delay for carry skip adder (W=3.6um) Complementary pass-transistor logic,in seconds, is

delay =

2.7703e-008

Propagation Delay for carry skip adder (W=6um) Complementary pass-transistor logic ,in seconds, is

delay =

2.3650e-008

Dual pass-transistor logic design style

-

Propagation Delay for carry skip adder (W=1.8um) Dual pass-transistor logic ,in seconds, is

delay =

5.7514e-008

Propagation Delay for carry skip adder (W=3.6um) Dual pass-transistor logic ,in seconds, is

delay =

3.7404e-008

Propagation Delay for carry skip adder (W=6um) Dual pass-transistor logic ,in seconds, is

delay =

3.2325e-008

Carry look ahead adder architecture

Fully static CMOS logic design style

Propagation Delay for carry look ahead adder (W=1.8um) fully static CMOS logic ,in seconds, is

delay =

1.3207e-008

Propagation Delay for carry look ahead adder (W=3.6um) fully static CMOS logic ,in seconds, is

delay =

9.6977e-009

Propagation Delay for carry look ahead adder (W=6um) fully static CMOS logic ,in seconds, is

delay =

7.8571e-009

Domino CMOS logic design style

-

Propagation Delay for carry look ahead adder (W=1.8um) Domino CMOS logic ,in seconds, is

delay =

2.2634e-008

Propagation Delay for carry look ahead adder (W=3.6um) Domino CMOS logic ,in seconds, is

delay =

1.4990e-008

Propagation Delay for carry look ahead adder (W=6um) Domino CMOS logic ,in seconds, is

delay =

1.2120e-008

Complementary pass-transistor logic design style

Propagation Delay for carry look ahead adder (W=1.8um) Complementary pass-transistor logic ,in seconds, is

delay =

2.0764e-008

Propagation Delay for carry look ahead adder (W=3.6um) Complementary pass-transistor logic ,in seconds, is

delay =

1.3995e-008

Propagation Delay for carry look ahead adder (W=6um) Complementary pass-transistor logic ,in seconds, is

delay =

1.0403e-008

Dual pass-transistor logic design style

Propagation Delay for carry look ahead adder (W=1.8um) Dual pass-transistor logic ,in seconds, is

delay =

3.0953e-008

Propagation Delay for carry look ahead adder (W=3.6um) Dual pass-transistor logic ,in seconds, is

delay =

2.1546e-008

Propagation Delay for carry look ahead adder (W=6um) Dual pass-transistor logic ,in seconds, is

delay =

1.7720e-008

Conditional sum adder architecture

Fully static CMOS logic design style

Propagation Delay for conditional sum adder (W=1.8um) fully static CMOS logic ,in seconds, is

delay =

4.7212e-008

Propagation Delay for conditional sum adder (W=3.6um) fully static CMOS logic ,in seconds, is

delay =

3.0418e-008

Propagation Delay for conditional sum adder (W=6um) fully static CMOS logic ,in seconds, is

delay =

2.5020e-008

Domino CMOS logic design style

-

Propagation Delay for conditional sum adder (W=1.8um) Domino CMOS logic ,in seconds, is

delay =

4.1000e-008

Propagation Delay for conditional sum adder (W=3.6um) Domino CMOS logic ,in seconds, is

delay =

2.5513e-008

Propagation Delay for conditional sum adder (W=6um) Domino CMOS logic ,in seconds, is

delay =

1.9312e-008

Complementary pass-transistor logic design style

Propagation Delay for conditional sum adder (W=1.8um) Complementary pass-transistor logic ,in seconds, is

delay =

3.0965e-008

Propagation Delay for conditional sum adder (W=3.6um) Complementary pass-transistor logic ,in seconds, is

delay =

1.8880e-008

Propagation Delay for conditional sum adder (W=6um) Complementary pass-transistor logic ,in seconds, is

delay =

1.5089e-008

Dual pass-transistor logic design style

Propagation Delay for conditional sum adder (W=1.8um) Dual pass-transistor logic ,in seconds, is

delay =

2.3815e-008

Propagation Delay for conditional sum adder (W=3.6um) Dual pass-transistor logic ,in seconds, is

delay =

1.4321e-008

Propagation Delay for conditional sum adder (W=6um) Dual pass-transistor logic ,in seconds, is

delay =

1.1417e-008

Energy consumption results

Ripple carry adder architecture

-

Fully static CMOS logic design style

Energy consumption for ripple carry adder (W=1.8um) fully static CMOS logic ,in Joules, is

energy =

8.5893e-010

Energy consumption for ripple carry adder (W=3.6um) fully static CMOS logic ,in Joules, is

energy =

1.0500e-009

Energy consumption for ripple carry adder (W=6um) fully static CMOS logic ,in Joules, is

energy =

1.3601e-009

Domino CMOS logic design style

-

Energy consumption for ripple carry adder (W=1.8um) Domino CMOS logic ,in Joules, is

energy =

2.0421e-009

Energy consumption for ripple carry adder (W=3.6um) Domino CMOS logic ,in Joules, is

energy =

2.8288e-009

Energy consumption for ripple carry adder (W=6um) Domino CMOS logic ,in Joules, is

energy =

3.8403e-009

Complementary pass-transistor logic design style

Energy consumption for ripple carry adder (W=1.8um) Complementary pass-transistor logic ,in Joules, is

energy =

1.2832e-009

Energy consumption for ripple carry adder (W=3.6um) Complementary pass-transistor logic ,in Joules, is

energy =

1.7518e-009

Energy consumption for ripple carry adder (W=6um) Complementary pass-transistor logic ,in Joules, is

energy =

2.2833e-009

Dual pass-transistor logic design style

Energy consumption for ripple carry adder (W=1.8um) Dual pass-transistor logic ,in Joules, is

energy =

1.3447e-009

Energy consumption for ripple carry adder (W=3.6um) Dual pass-transistor logic ,in Joules, is

energy =

1.8712e-009

Energy consumption for ripple carry adder (W=6um) Dual pass-transistor logic ,in Joules, is

energy =

2.4959e-009

Carry select adder architecture

-

Fully static CMOS logic design style

Energy consumption for carry select adder (W=1.8um) fully static CMOS logic ,in Joules, is

energy =

1.3080e-009

Energy consumption for carry select adder (W=3.6um) fully static CMOS logic ,in Joules, is

energy =

1.8192e-009

Energy consumption for carry select adder (W=6um) fully static CMOS logic ,in Joules, is

energy =

2.7400e-009

Domino CMOS logic design style

-

Energy consumption for carry select adder (W=1.8um) Domino CMOS logic ,in Joules, is

energy =

3.6488e-009

Energy consumption for carry select adder (W=3.6um) Domino CMOS logic ,in Joules, is

energy =

5.2619e-009

Energy consumption for carry select adder (W=6um) Domino CMOS logic ,in Joules, is

energy =

7.4784e-009

Complementary pass-transistor logic design style

Energy consumption for carry select adder (W=1.8um) Complementary pass-transistor logic ,in Joules, is

energy =

1.8833e-009

Energy consumption for carry select adder (W=3.6um) Complementary pass-transistor logic ,in Joules, is

energy =

2.4749e-009

Energy consumption for carry select adder

(W=6um)
Complementary pass-transistor logic ,in Joules, is

energy =

3.2131e-009

Dual pass-transistor logic design style

Energy consumption for carry select adder (W=1.8um) Dual pass-transistor logic ,in Joules, is

energy =

2.5655e-009

Energy consumption for carry select adder (W=3.6um) Dual pass-transistor logic ,in Joules, is

energy =

3.4837e-009

Energy consumption for carry select adder (W=6um) Dual pass-transistor logic ,in Joules, is

energy =

5.0097e-009

Carry skip adder architecture

Fully static CMOS logic design style

Energy consumption for carry skip adder (W=1.8um) fully static CMOS logic ,in Joules, is

energy =

9.7174e-010

Energy consumption for carry skip adder (W=3.6um) fully static CMOS logic ,in Joules, is

energy =

1.2606e-009

Energy consumption for carry skip adder (W=6um) fully static CMOS logic ,in Joules, is

energy =

1.6198e-009

Domino CMOS logic design style

Energy consumption for carry skip adder (W=1.8um) Domino CMOS logic ,in Joules, is

energy =

2.3119e-009

Energy consumption for carry skip adder (W=3.6um) Domino CMOS logic ,in Joules, is

energy =

3.2018e-009

Energy consumption for carry skip adder (W=6um) Domino CMOS logic ,in Joules, is

energy =

4.3791e-009

Complementary pass-transistor logic design style

Energy consumption for carry skip adder (W=1.8um) Complementary pass-transistor logic ,in Joules, is

energy =

1.8733e-009

Energy consumption for carry skip adder (W=3.6um) Complementary pass-transistor logic ,in Joules, is

energy =

2.4619e-009

Energy consumption for carry skip adder (W=6um) Complementary pass-transistor logic ,in Joules, is

energy =

3.4942e-009

Dual pass-transistor logic design style

Energy consumption for carry skip adder (W=1.8um) Dual pass-transistor logic ,in Joules, is

energy =

1.8460e-009

Energy consumption for carry skip adder (W=3.6um) Dual pass-transistor logic ,in Joules, is

energy =

2.4607e-009

Energy consumption for carry skip adder (W=6um) Dual pass-transistor logic ,in Joules, is

energy =

4.1074e-009

Carry look ahead adder architecture

Fully static CMOS logic design style

Energy consumption for carry look ahead adder (W=1.8um) fully static CMOS logic ,in Joules, is

energy =

8.7103e-010

Energy consumption for carry look ahead adder (W=3.6um) fully static CMOS logic ,in Joules, is

energy =

1.1701e-009

Energy consumption for carry look ahead adder (W=6um) fully static CMOS logic ,in Joules, is

energy =

1.5493e-009

Domino CMOS logic design style

Energy consumption for carry look ahead adder (W=1.8um) Domino CMOS logic ,in Joules, is

energy =

2.1890e-009

Energy consumption for carry look ahead adder (W=3.6um) Domino CMOS logic ,in Joules, is

energy =

3.0274e-009

Energy consumption for carry look ahead adder (W=6um) Domino CMOS logic ,in Joules, is

energy =

4.1687e-009

Complementary pass-transistor logic design style

Energy consumption for carry look ahead adder (W=1.8um) Complementary pass-transistor logic ,in Joules, is

energy =

2.9752e-009

Energy consumption for carry look ahead adder (W=3.6um) Complementary pass-transistor logic ,in Joules, is

energy =

3.9728e-009

Energy consumption for carry look ahead adder (W=6um) Complementary pass-transistor logic ,in Joules, is

energy =

5.5281e-009

Dual pass-transistor logic design style

Energy consumption for carry look ahead adder (W=1.8um) Dual pass-transistor logic ,in Joules, is

energy =

1.7726e-009

Energy consumption for carry look ahead adder (W=3.6um) Dual pass-transistor logic ,in Joules, is

energy =

2.3385e-009

Energy consumption for carry look ahead adder (W=6um) Dual pass-transistor logic ,in Joules, is

energy =

3.1480e-009

Conditional sum adder architecture

Fully static CMOS logic design style

Energy consumption for conditional sum adder (W=1.8um) fully static CMOS logic ,in Joules, is

energy =

1.7198e-009

Energy consumption for conditional sum adder (W=3.6um) fully static CMOS logic ,in Joules, is

energy =

2.4469e-009

Energy consumption for conditional sum adder (W=6um) fully static CMOS logic ,in Joules, is

energy =

3.3003e-009

Domino CMOS logic design style.

-

Energy consumption for conditional sum adder (W=1.8um) Domino CMOS logic ,in Joules, is

energy =

2.3464e-009

Energy consumption for conditional sum adder (W=3.6um) Domino CMOS logic ,in Joules, is

energy =

3.4549e-009

Energy consumption for conditional sum adder (W=6um) Domino CMOS logic ,in Joules, is

energy =

4.9200e-009

Complementary pass-transistor logic design style

Energy consumption for conditional sum adder (W=1.8um) Complementary pass-transistor logic ,in Joules, is

energy =

1.4528e-009

Energy consumption for conditional sum adder (W=3.6um) Complementary pass-transistor logic ,in Joules, is

energy =

1.8683e-009

Energy consumption for conditional sum adder (W=6um) Complementary pass-transistor logic ,in Joules, is

energy =

2.5400e-009

Dual pass-transistor logic design style

Energy consumption for conditional sum adder (W=1.8um) Dual pass-transistor logic ,in Joules, is

energy =

1.7108e-009

Energy consumption for conditional sum adder (W=3.6um) Dual pass-transistor logic ,in Joules, is

energy =

2.0403e-009

Energy consumption for Conditional sum adder (W=6um) Dual pass-transistor logic ,in Joules, is

energy =

2.7251e-009

Output of the program listed in *Appendix A1* for listing of adder designs satisfying the delay and energy constraint.

Input→

****
*

Please input adder size i.e (number of bits) from 1 to 64 in multiples of four = 40

Please input propagation delay value, in seconds, = 20e-09

Tolerance → acceptable variation in propagation delay and energy consumption value

Please input tolerance value in propagation delay (seconds) = 5e-09

****
*

delay =

1.9124e-008

Architecture=
Carry select adder
Channel width=
3.6um
Logic design style= Fully static logic

delay =

1.5219e-008

Architecture=
Carry select adder
Channel width=
6um
Logic design style= Fully static logic

delay =

2.2535e-008

Architecture=
Carry select adder
Channel width=
6um
Logic design style= Domino CMOS logic

delay =

2.0103e-008

Architecture=
Carry select adder

Channel width=
3.6um
Logic design style=
Complementary pass-transistor logic

delay =

1.5565e-008

Architecture=
Carry select adder
Channel width=
6um
Logic design style=
Complementary pass-transistor logic

delay =

1.8741e-008

Architecture=
Carry look ahead adder
Channel width=
1.8um
Logic design style= Domino CMOS logic

delay =

1.7888e-008

Architecture=
Carry look ahead adder
Channel width=
1.8um

Logic design style=
Complementary pass-tansistor logic

delay =

2.3718e-008

Architecture=
Carry skip adder
Channel width=
1.8um
Logic design style= Fully static logic

delay =

2.3718e-008

Architecture=
Carry skip adder
Channel width=
3.6um
Logic design style= Fully static logic

delay =

2.2155e-008

Architecture=
Carry skip adder
Channel width=
6um
Logic design style= Domino CMOS logic

delay =

3.4427e-008

Architecture=
Carry skip adder
Channel width=
1.8um
Logic design style=
Complementary pass-transistor logic

delay =

2.1842e-008

Architecture=
Carry skip adder
Channel width=
3.6um
Logic design style=
Complementary pass-transistor logic

delay =

2.1842e-008

Architecture=
Carry skip adder
Channel width=
6um
Logic design style=
Complementary pass-transistor logic

delay =

2.2631e-008

Architecture=
Conditional sum adder

Channel width=
3.6um
Logic design style= Fully static logic

delay =

1.8334e-008

Architecture=
Conditional sum adder
Channel width=
6um
Logic design style= Fully static logic

delay =

1.8993e-008

Architecture=
Conditional sum adder
Channel width=
3.6um
Logic design style= Domino CMOS logic

delay =

2.1961e-008

Architecture=
Conditional sum adder
Channel width=
1.8um
Logic design style=
Complementary pass-transistor logic

Please input energy consumption value, (in Joules),= 30e-10

Tolerance --> acceptable variation in energy consumption value (in Joules)

Please input tolerance value in energy consumption value (in Joules) = 4e-10

****

*

energy =

2.8398e-009

Architecture= Ripple carry adder
Channel width= 6um
Logic design style= Domino CMOS logic

energy =

3.3209e-009

Architecture= Carry select adder
Channel width= 6um
Logic design style= Complementary pass-transistor logic

energy =

3.0865e-009

Architecture= Carry look ahead adder
Channel width= 6um
Logic design style= Domino CMOS logic

energy =

2.8583e-009

Architecture= Carry look ahead adder
Channel width= 3.6um
Logic design style= Complementary pass-transistor logic

energy =

3.2119e-009

Architecture= Carry skip adder
Channel width= 6um
Logic design style= Domino CMOS logic

APPENDIX A7

Interconnection capacitances of 4–bit and 16-bit ripple carry adder design for *(w/l)*=1.8μm and fully static logic design style.

4-bit Ripple carry adder

C1 b0 0 26.7041FF
C2 1125 0 6.82344FF
C3 1112 0 6.12144FF
C4 U5/cmosinv1_2/out 0 17.4247FF
C5 U5/rcacarryc_1/coc 0 35.2415FF
C6 1078 0 18.8658FF
C7 1068 0 22.0446FF
C8 1050 0 9.67752FF
C9 1035 0 14.6966FF
C10 1027 0 2.82528FF
C11 1021 0 5.65056FF
C12 a2 0 18.1876FF
C13 968 0 22.4266FF
C14 958 0 9.97272FF
C15 951 0 10.8302FF
C16 947 0 10.3594FF
C17 cin 0 15.9473FF
C18 898 0 22.4266FF
C19 893 0 9.97272FF
C20 881 0 10.8302FF
C21 878 0 10.3594FF
C22 a0 0 25.1806FF
C23 U5/rca_si_1/pic 0 19.224FF
C24 U5/rca_pi_1/pi 0 37.35FF
C25 799 0 6.82344FF
C26 788 0 6.12144FF
C27 U5/rca_pi_1/aic 0 22.1227FF
C28 767 0 22.0446FF
C29 765 0 18.8658FF
C30 U5/rca_pi_1/bic 0 21.667FF
C31 720 0 22.4266FF
C32 709 0 9.97272FF
C33 702 0 10.8302FF
C34 698 0 10.3594FF
C35 b2 0 20.1195FF
C36 632 0 6.12144FF
C37 U6/cmosinv1_4/out 0 18.0436FF
C38 U6/cmosinv1_3/out_1 0 23.9501FF
C39 559 0 6.12144FF
C40 U2/rcafulladder_3/coc 0 38.2752FF
C41 490 0 10.8302FF
C42 487 0 10.3594FF
C43 U6/cmosinv1_4/out_1 0 19.0998FF
C44 U6/cmosinv1_3/out 0 23.9101FF
C45 448 0 22.0446FF
C46 428 0 6.82344FF
C47 b3 0 14.9384FF
C48 U7/rcacarryc_1/coc 0 34.1852FF
C49 a3 0 28.4816FF
C50 406 0 18.8658FF
C51 b1 0 21.7447FF
C52 U7/rcacarryc_1/coc_1 0 34.3278FF
C53 a1 0 24.1391FF
C54 361 0 22.0446FF
C55 360 0 18.8658FF
C56 Gnd 0 1145.77FF
C57 339 0 6.82344FF
C58 278 0 22.4266FF
C59 264 0 9.97272FF
C60 U7/cmosinv1_3/out 0 22.8334FF
C61 U7/cmosinv1_4/out 0 18.3719FF
C62 U8/cmosinv1_1/in_2 0 39.2488FF
C63 Vdd 0 1287.79FF
C64 U8/rca_si_1/cic_2 0 20.884FF
C65 U8/rca_si_1/pic 0 20.2241FF
C66 190 0 9.67752FF
C67 179 0 14.6966FF
C68 171 0 2.82528FF
C69 165 0 5.65056FF
C70 c2 0 45.7569FF
C71 U8/rca_si_1/pi_2 0 37.4605FF
C72 c3 0 20.1321FF
C73 U8/rca_si_1/cic_1 0 16.6655FF
C74 124 0 2.82528FF
C75 s0 0 43.8034FF
C76 c0 0 51.3619FF
C77 s1 0 27.005FF
C78 U8/cmosinv1_1/out_2 0 18.5058FF
C79 97 0 9.67752FF
C80 86 0 14.6966FF
C81 74 0 5.65056FF
C82 U8/rca_si_1/pi 0 43.1206FF
C83 s2 0 25.4304FF
C84 s3 0 39.1928FF
C85 56 0 9.67752FF
C86 43 0 14.6966FF
C87 35 0 2.82528FF
C88 29 0 5.65056FF
C89 U8/cmosinv1_2/out_1 0 21.0384FF
C90 U8/cmosinv1_1/out 0 19.9444FF
C91 c1 0 50.0899FF

16-bit Ripple carry adder

C1 4590 0 2.82528FF
C2 U8/cmosinv1_1/out_1 0 17.9204FF
C3 U8/cmosinv1_2/out 0 22.5482FF
C4 4563 0 9.67752FF
C5 4545 0 14.6966FF
C6 4530 0 5.65056FF
C7 cin 0 17.2195FF
C8 b0 0 26.1868FF
C9 a0 0 49.9975FF
C10 4507 0 22.0446FF
C11 4505 0 18.8658FF
C12 4482 0 6.82344FF
C13 4467 0 6.12144FF
C14 4440 0 22.4266FF
C15 4427 0 9.97272FF
C16 4415 0 10.8302FF
C17 4412 0 10.3594FF
C18 U8/cmosinv1_3/out_2 0 22.9252FF
C19 U8/cmosinv1_4/out_2 0 18.8428FF
C20 4325 0 22.4266FF
C21 4310 0 9.97272FF
C22 4298 0 10.8302FF
C23 4294 0 10.3594FF
C24 U8/cmosinv1_3/out 0 24.6888FF
C25 4267 0 22.4266FF
C26 4256 0 9.97272FF
C27 4243 0 10.8302FF
C28 4241 0 10.3594FF
C29 4199 0 6.82344FF
C30 4187 0 6.12144FF
C31 b8 0 39.8639FF
C32 a8 0 20.3612FF
C33 4160 0 22.0446FF
C34 4158 0 18.8658FF
C35 4113 0 22.0446FF
C36 4111 0 18.8658FF
C37 Gnd 0 5594.33FF
C38 4098 0 6.82344FF
C39 4089 0 6.12144FF
C40 4048 0 9.67752FF
C41 4032 0 14.6966FF
C42 4024 0 2.82528FF
C43 4018 0 5.65056FF
C44 U9/cmosinv1_1/out 0 18.5674FF
C45 3957 0 10.8302FF
C46 U9/cmosinv1_3/out 0 21.4661FF
C47 U8/rcacarryc_1/coc 0

34.79FF
C48 3921 0 22.4266FF
C49 3905 0 9.97272FF
C50 3890 0 10.3594FF
C51 b1 0 31.4789FF
C52 U9/cmosinv1_1/in_3 0 35.2415FF
C53 U8/rca_pi_1/pi_1 0 47.4253FF
C54 3863 0 22.0446FF
C55 3861 0 18.8658FF
C56 3838 0 6.82344FF
C57 3823 0 6.12144FF
C58 U8/cmosinv1_4/out_1 0 18.7002FF
C59 3774 0 6.82344FF
C60 3761 0 6.12144FF
C61 U9/cmosinv1_1/in_2 0 34.9132FF
C62 a2 0 33.3536FF
C63 b2 0 17.4751FF
C64 3729 0 22.0446FF
C65 3727 0 18.8658FF
C66 U9/cmosinv1_3/out_2 0 21.7231FF
C67 U9/rca_pi_1/bic 0 22.2664FF
C68 3657 0 22.4266FF
C69 3644 0 9.97272FF
C70 3629 0 10.8302FF
C71 3626 0 10.3594FF
C72 U8/cmosinv1_3/out_1 0 24.4415FF
C73 U8/cmosinv1_4/out 0 21.2663FF
C74 b10 0 42.8902FF
C75 a10 0 30.1271FF
C76 3598 0 22.0446FF
C77 3596 0 18.8658FF
C78 3575 0 6.82344FF
C79 3563 0 6.12144FF
C80 U9/cmosinv1_1/in 0 41.9234FF
C81 U8/rcacarryc_1/coc_1 0 34.79FF
C82 a1 0 54.4477FF
C83 3528 0 22.4266FF
C84 3520 0 9.97272FF
C85 3506 0 10.8302FF
C86 3504 0 10.3594FF
C87 b9 0 55.0051FF
C88 U9/cmosinv1_4/out_2 0 18.1872FF
C89 a9 0 32.7748FF

C90 U9/cmosinv1_3/out_1 0 23.233FF
C91 3410 0 10.8302FF
C92 U10/cmosinv1_4/out 0 19.8288FF
C93 U9/cmosinv1_2/out 0 22.8766FF
C94 3378 0 22.4266FF
C95 3360 0 9.97272FF
C96 3345 0 10.3594FF
C97 U9/rca_pi_1/bic_2 0 21.0092FF
C98 U9/rca_pi_1/pi_2 0 40.7048FF
C99 b7 0 35.6026FF
C100 a7 0 53.334FF
C101 3320 0 22.0446FF
C102 3318 0 18.8658FF
C103 3293 0 6.82344FF
C104 3289 0 2.82528FF
C105 3277 0 6.12144FF
C106 c1 0 71.2213FF
C107 U10/cmosinv1_1/out 0 20.0772FF
C108 U10/cmosinv1_2/out 0 20.3818FF
C109 3246 0 9.67752FF
C110 3227 0 14.6966FF
C111 3210 0 5.65056FF
C112 U10/cmosinv1_1/in_2 0 47.3006FF
C113 3121 0 22.4266FF
C114 3107 0 9.97272FF
C115 3092 0 10.8302FF
C116 3089 0 10.3594FF
C117 b12 0 57.0524FF
C118 a12 0 39.3116FF
C119 3060 0 22.0446FF
C120 3058 0 18.8658FF
C121 3039 0 6.82344FF
C122 3027 0 6.12144FF
C123 3003 0 22.0446FF
C124 3001 0 18.8658FF
C125 2979 0 6.82344FF
C126 2967 0 6.12144FF
C127 U10/cmosinv1_1/in 0 34.7576FF
C128 2941 0 9.67752FF
C129 2928 0 14.6966FF
C130 2916 0 2.82528FF
C131 2911 0 5.65056FF
C132 U10/cmosinv1_1/in_4 0 31.0946FF
C133 U10/cmosinv1_3/out 0 24.2384FF
C134 2798 0 9.67752FF
C135 2779 0 14.6966FF
C136 2768 0 2.82528FF
C137 2762 0 5.65056FF
C138 2732 0 22.4266FF

C139 2721 0 9.97272FF
C140 2707 0 10.8302FF
C141 2704 0 10.3594FF
C142 U11/cmosinv1_3/out_1 0 21.3948FF
C143 2664 0 6.82344FF
C144 2652 0 6.12144FF
C145 b3 0 45.5175FF
C146 U10/cmosinv1_1/in_1 0 43.2562FF
C147 a3 0 56.3103FF
C148 2610 0 22.0446FF
C149 2608 0 18.8658FF
C150 U10/rca_pi_1/aic_1 0 22.4003FF
C151 U10/rca_pi_1/bic_1 0 24.9721FF
C152 2548 0 22.4266FF
C153 2534 0 9.97272FF
C154 2519 0 10.8302FF
C155 2516 0 10.3594FF
C156 b4 0 59.51FF
C157 2472 0 6.82344FF
C158 2458 0 6.12144FF
C159 a4 0 40.1765FF
C160 U11/cmosinv1_1/in_2 0 39.055FF
C161 2398 0 18.8658FF
C162 2388 0 22.0446FF
C163 U10/cmosinv1_1/in_3 0 36.5569FF
C164 a11 0 41.8655FF
C165 2350 0 22.4266FF
C166 2337 0 9.97272FF
C167 2322 0 10.8302FF
C168 2319 0 10.3594FF
C169 U10/rca_si_1/pic 0 19.859FF
C170 b11 0 81.5303FF
C171 U10/cmosinv1_2/out_1 0 24.4469FF
C172 2230 0 2.82528FF
C173 U11/rca_si_1/cic 0 18.5209FF
C174 U11/cmosinv1_1/out_2 0 17.5208FF
C175 2186 0 9.67752FF
C176 2167 0 14.6966FF
C177 2152 0 5.65056FF
C178 U4/8rca_1/4rca_2/rcafulladder_4/rcasum_1/N1_1 0 0.23976FF
C179 2117 0 22.4266FF
C180 2109 0 9.97272FF
C181 2097 0 10.8302FF
C182 2094 0 10.3594FF
C183 U11/cmosinv1_4/out_2 0 21.3656FF
C184 2063 0 22.0446FF
C185 2061 0 18.8658FF
C186 2049 0 6.82344FF
C187 2034 0 6.12144FF
C188 U11/rcacarryc_1/coc_1 0 35.1184FF
C189 U12/cmosinv1_1/in_3 0 34.9132FF
C190 1979 0 9.67752FF
C191 1966 0 14.6966FF
C192 1956 0 2.82528FF
C193 1951 0 5.65056FF
C194 U11/rca_pi_1/pi_1 0 53.1806FF
C195 U11/cmosinv1_3/out 0 22.2145FF
C196 U11/cmosinv1_4/out 0 22.9079FF
C197 1859 0 22.0446FF
C198 1857 0 18.8658FF
C199 c4 0 53.4575FF
C200 1840 0 6.82344FF
C201 1824 0 6.12144FF
C202 U11/cmosinv1_1/in_3 0 34.79FF
C203 1780 0 22.4266FF
C204 1770 0 9.97272FF
C205 1756 0 10.8302FF
C206 1753 0 10.3594FF
C207 U11/rca_pi_1/aic 0 22.4003FF
C208 1698 0 10.8302FF
C209 U11/cmosinv1_4/out_1 0 21.2663FF
C210 1672 0 22.4266FF
C211 1664 0 9.97272FF
C212 1652 0 10.3594FF
C213 U12/rca_si_1/pi 0 44.2091FF
C214 U12/cmosinv1_2/out 0 21.141FF
C215 1596 0 10.8302FF
C216 U12/rca_si_1/pic 0 20.0999FF
C217 c2 0 102.051FF
C218 b13 0 52.3494FF
C219 1559 0 22.4266FF
C220 1537 0 9.97272FF
C221 1522 0 10.3594FF
C222 U12/rca_pi_1/bic_1 0 23.7222FF
C223 U12/cmosinv1_3/out 0 21.1378FF
C224 a13 0 52.177FF
C225 1491 0 22.0446FF
C226 1489 0 18.8658FF
C227 1471 0 6.82344FF
C228 1455 0 6.12144FF
C229 U12/rca_si_1/pic_1 0 18.3103FF
C230 c7 0 103.468FF
C231 U12/cmosinv1_1/in_1 0 56.0972FF
C232 U12/cmosinv1_2/out_1 0 21.9942FF
C233 1350 0 22.4266FF
C234 1335 0 9.97272FF
C235 1320 0 10.8302FF
C236 1317 0 10.3594FF
C237 U5/8rca_2/4rca_1/rcafulladder_1/rcasum_1/cic 0 0.23976FF
C238 1284 0 22.0446FF
C239 1282 0 18.8658FF
C240 1260 0 6.82344FF
C241 1244 0 6.12144FF
C242 c5 0 92.6525FF
C243 a5 0 38.8501FF
C244 b5 0 79.9438FF
C245 U12/cmosinv1_1/in_2 0 34.2565FF
C246 1206 0 9.67752FF
C247 1184 0 14.6966FF
C248 1172 0 2.82528FF
C249 1166 0 5.65056FF
C250 c10 0 67.0666FF
C251 U12/cmosinv1_4/out 0 22.2091FF
C252 U12/cmosinv1_3/out_1 0 21.4661FF
C253 1134 0 9.67752FF
C254 1114 0 14.6966FF
C255 1103 0 2.82528FF
C256 1097 0 5.65056FF
C257 U12/rca_pi_1/aic_1 0 22.3182FF
C258 U12/cmosinv1_4/out_2 0 19.1722FF
C259 1020 0 22.0446FF
C260 1018 0 18.8658FF
C261 1010 0 6.82344FF
C262 1000 0 6.12144FF
C263 U13/cmosinv1_3/out 0 23.8896FF
C264 b6 0 55.0048FF
C265 U13/cmosinv1_4/out 0 18.2585FF
C266 a6 0 75.1131FF
C267 918 0 2.82528FF
C268 U13/cmosinv1_1/in_1 0 34.3278FF
C269 857 0 2.82528FF
C270 852 0 5.65056FF

C271 845 0 5.65056FF
C272 b14 0 65.9068FF
C273 775 0 2.82528FF
C274 U13/rca_pi_1/bic 0 25.9078FF
C275 U13/cmosinv1_3/out_1 0 21.0665FF
C276 a14 0 43.715FF
C277 711 0 2.82528FF
C278 702 0 5.65056FF
C279 U13/cmosinv1_2/out_1 0 24.2773FF
C280 639 0 5.65056FF
C281 U13/cmosinv1_1/out 0 19.6949FF
C282 c11 0 81.1397FF
C283 U13/cmosinv1_1/in_2 0 54.1152FF
C284 581 0 2.82528FF
C285 575 0 5.65056FF
C286 U13/cmosinv1_1/out_2 0 22.261FF
C287 U13/cmosinv1_1/in 0 41.4113FF
C288 U13/cmosinv1_2/out_2 0 22.2685FF
C289 517 0 2.82528FF
C290 511 0 5.65056FF
C291 U13/cmosinv1_1/in_3 0 50.6941FF
C292 456 0 2.82528FF
C293 451 0 5.65056FF
C294 a15 0 72.4423FF
C295 b15 0 88.2979FF
C296 s0 0 64.1311FF
C297 s1 0 57.4785FF
C298 s3 0 48.8034FF
C299 s4 0 52.6869FF
C300 373 0 14.6966FF
C301 U14/rca_si_1/pi_4 0 32.9288FF
C302 U14/rca_si_1/pic_2 0 23.0123FF
C303 s6 0 29.326FF
C304 U14/cmosinv1_2/out_5 0 22.68FF
C305 s7 0 28.327FF
C306 U7/c5 0 0.23976FF
C307 341 0 9.67752FF
C308 338 0 9.67752FF
C309 U14/cmosinv1_1/in_6 0 59.3618FF
C310 328 0 14.6966FF
C311 U14/rca_si_1/pic 0 19.562FF
C312 s9 0 99.2794FF
C313 s2 0 67.8618FF
C314 U14/cmosinv1_2/out 0 23.2049FF
C315 s11 0 69.3769FF
C316 s12 0 62.7035FF
C317 U14/rca_si_1/cic_3 0 18.1796FF
C318 U14/cmosinv1_1/in_3 0 43.7798FF
C319 s13 0 28.6067FF
C320 U14/rca_si_1/pic_5 0 19.562FF
C321 s8 0 55.7129FF
C322 240 0 9.67752FF
C323 U14/cmosinv1_1/in_4 0 35.6094FF
C324 224 0 14.6966FF
C325 s15 0 49.3859FF
C326 c0 0 84.3901FF
C327 s14 0 29.8454FF
C328 U14/rca_si_1/pic_1 0 18.9054FF
C329 c3 0 83.5947FF
C330 U14/cmosinv1_2/out_3 0 24.1898FF
C331 184 0 9.67752FF
C332 169 0 14.6966FF
C333 s5 0 51.9833FF
C334 U14/rca_si_1/pic_4 0 18.1775FF
C335 c6 0 97.391FF
C336 U14/cmosinv1_2/out_4 0 23.4619FF
C337 141 0 9.67752FF
C338 U14/cmosinv1_1/in_7 0 48.9537FF
C339 122 0 14.6966FF
C340 c8 0 74.6593FF
C341 s10 0 50.7445FF
C342 U14/rca_si_1/pic_3 0 19.2337FF
C343 91 0 9.67752FF
C344 U14/cmosinv1_1/in_5 0 62.6087FF
C345 76 0 14.6966FF
C346 c9 0 55.0552FF
C347 U14/cmosinv1_2/out_2 0 24.1898FF
C348 c12 0 98.7104FF
C349 c13 0 117FF
C350 47 0 9.67752FF
C351 U14/cmosinv1_1/in_1 0 42.6217FF
C352 33 0 14.6966FF
C353 c14 0 30.5892FF
C354 U14/cmosinv1_2/out_1 0 22.3517FF
C355 U14/cmosinv1_1/out_1 0 21.2576FF
C356 c15 0 16.5776FF
C357 Vdd 0 6260.12FF

APPENDIX A8

Program to calculate all possible input patterns for 4-bit adders

```
clear all
n=input('input no. of bits = ')
for i=0:(2^n)-1
quot=i;
for j=1:n
c(j)=rem(quot,2);
quot=fix(quot/2);
end
a(i+1,:)=c;
end
a
save inpgen4.mat a
```

Programs to calculate average number of gate output transitions for different 4-bit adders for fully static CMOS logic design style.

Fully static CMOS logic design style

1) Program for Ripple carry adder

```
clear all
load inpgen4.mat
a
n=input('input no. of bits
= ')
p=(2^n)+1;
for i=1:512;
n1(i)=not(a(i,1));
n2(i)=not(a(i,2));
n3(i)=xor(a(i,1),a(i,2));
n4(i)=not(a(i,p));
n5(i)=xor(n3(i), a(i,p));
n6(i)=not(n3(i));
y1=and(a(i,1),a(i,2));
x1=or(a(i,1),a(i,2));
z1=and(x1,a(i,p));
n7(i)=not(or(z1,y1));
n8(i)=not(n7(i));

k=3;
n9(i)=not(a(i,k));
n10(i)=not(a(i,(k+1)));
n11(i)=xor(a(i,k),a(i,(k+
1)));
n12(i)=not(n8(i));
n13(i)=xor(n11(i),
n8(i));
n14(i)=not(n11(i));
y2=and(a(i,k),a(i,(k+1)))
;
x2=or(a(i,k),a(i,(k+1)));
z2=and(x2,n8(i));
n15(i)=not(or(z2,y2));
n16(i)=not(n15(i));

k=5;
n17(i)=not(a(i,k));
n18(i)=not(a(i,(k+1)));
n19(i)=xor(a(i,k),a(i,(k+
1)));
n20(i)=not(n16(i));
n21(i)=xor(n19(i),
n16(i));
n22(i)=not(n19(i));
y3=and(a(i,k),a(i,(k+1)))
;
x3=or(a(i,k),a(i,(k+1)));
z3=and(x3,n16(i));
n23(i)=not(or(z3,y3));
n24(i)=not(n23(i));
k=7;
n25(i)=not(a(i,k));
n26(i)=not(a(i,(k+1)));
n27(i)=xor(a(i,k),a(i,(k+
1)));
n28(i)=not(n24(i));
n29(i)=xor(n27(i),
n24(i));
n30(i)=not(n27(i));
y4=and(a(i,k),a(i,(k+1)))
;
x4=or(a(i,k),a(i,(k+1)));
z4=and(x4,n16(i));
n31(i)=not(or(z4,y4));
n32(i)=not(n31(i));
end;
 c1=0;
c2=0;c3=0;c4=0;c5=0;c
6=0;c7=0;c8=0;c9=0;c1
0=0;c11=0;c12=0;c13=0
;c14=0;c15=0;c16=0;c1
7=0;
c18=0;c19=0;c20=0;c21
=0;c22=0;c23=0;c24=0;
c25=0;c26=0;c27=0;c28
=0;c29=0;c30=0;c31=0;
c32=0;
for i=1:512;
 for j=1:512;
if n1(i)==n1(j);
c1=c1;
else;
c1=c1+1;
end;
if n2(i)==n2(j);
c2=c2;
else;
c2=c2+1;
end;
if n3(i)==n3(j);
c3=c3;
else;
c3=c3+1;
end;
if n4(i)==n4(j)
c4=c4;
else;
c4=c4+1;
end;
if n5(i)==n5(j);
c5=c5;
else
c5=c5+1;
end;
if n6(i)==n6(j)
c6=c6;
else;
c6=c6+1;
end;
if n7(i)==n7(j)
c7=c7;
else;
c7=c7+1;
end;
if n8(i)==n8(j);
c8=c8;
else;
c8=c8+1;
end;
if n9(i)==n9(j);
c9=c9;
else;
c9=c9+1;
end;
if n10(i)==n10(j);
c10=c10;
else;
c10=c10+1;
end;
if n11(i)==n11(j);
c11=c11;
else;
c11=c11+1;
end;
if n12(i)==n12(j);
c12=c12;
else;
c12=c12+1;
end;
if n13(i)==n13(j);
c13=c13;
else;
c13=c13+1;
end;
if n14(i)==n14(j);
c14=c14;
else;
c14=c14+1;
end;
if n15(i)==n15(j);
c15=c15;
else;
c15=c15+1;
end;
 if n16(i)==n16(j);
c16=c16;
else;
c16=c16+1;
end;
 if n17(i)==n17(j);
c17=c17;
else;
c17=c17+1;
end;
```

```
if n18(i)==n18(j);
c18=c18;
else;
c18=c18+1;
end;
if n19(i)==n19(j);
c19=c19;
else;
c19=c19+1;
end;
if n20(i)==n20(j);
c20=c20;
else;
c20=c20+1;
end;
 if n21(i)==n21(j);
c21=c21;
else;
c21=c21+1;
end;
 if n22(i)==n22(j);
c22=c22;
else;
c22=c22+1;
end;
if n23(i)==n23(j);
c23=c23;
else;
c23=c23+1;
end;
 if n24(i)==n24(j);
c24=c24;
else;
c24=c24+1;
end;
 if n25(i)==n25(j);
c25=c25;
else;
c25=c25+1;
end;
if n26(i)==n26(j);
c26=c26;
else;
c26=c26+1;
end;
if n27(i)==n27(j);
c27=c27;
else;
c27=c27+1;
end;
 if n28(i)==n28(j);
c28=c28;
else;
c28=c28+1;
end;
 if n29(i)==n29(j);
c29=c29;
else;
c29=c29+1;
end;
if n30(i)==n30(j);
c30=c30;
else;
c30=c30+1;
end;
 if n31(i)==n31(j);
c31=c31;
else;
c31=c31+1;
end;
if n32(i)==n32(j);
c32=c32;
else;
c32=c32+1;
end;
end;
end;
count=c1+c2+c3+c4+c5
+c6+c7+c8+c9+c10+c11
+c12+c13+c14+c15+c16
+c17+c18+c19+c20+c21
+c22+c23+c24+c25+c26
+c27+c28+c29+c30+c31
+c32;
 aver=count/(512*512);
save resrca.mat aver
```

2) Program for Carry skip adder

```
clearall;
load inpgen4.mat;
a;
n=input('input
no.ofbits=');
p=(2*n)+1;
for i=1:512;
n1(i)=not(a(i,1));
n2(i)=not(a(i,2));
n3(i)=xor(a(i,1),a(i,2));
n4(i)=not(a(i,p));
n5(i)=xor(n3(i),a(i,p));
n6(i)=not(n3(i));
y1=and(a(i,1),a(i,2));
x1=or(a(i,1),a(i,2));
z1=and(x1,a(i,5));
n7(i)=not(or(z1,y1));
n8(i)=not(n7(i));

k=3;
n9(i)=not(a(i,k));
n10(i)=not(a(i,(k+1)));
n11(i)=xor(a(i,k),a(i,(k+
1)));
n12(i)=not(n8(i));
n13(i)=xor(n11(i),n8(i));
n14(i)=not(n11(i));
y2=and(a(i,k),a(i,(k+1)))
;
x2=or(a(i,k),a(i,(k+1)));
z2=and(x2,n8(i));
n15(i)=not(or(z2,y2));
n16(i)=not(n15(i));

k=5;
n17(i)=not(a(i,k));
n18(i)=not(a(i,(k+1)));
n19(i)=xor(a(i,k),a(i,(k+
1)));
n20(i)=not(n16(i));
n21(i)=xor(n19(i),n16(i)
);
n22(i)=not(n19(i));
y3=and(a(i,k),a(i,(k+1)))
;
x3=or(a(i,k),a(i,(k+1)));
z3=and(x3,n16(i));
n23(i)=not(or(z3,y3));
n24(i)=not(n23(i));

k=7;
n25(i)=not(a(i,k));
n26(i)=not(a(i,(k+1)));
n27(i)=xor(a(i,k),a(i,(k+
1)));
n28(i)=not(n24(i));
n29(i)=xor(n27(i),n24(i)
);
n30(i)=not(n27(i));
y4=and(a(i,k),a(i,(k+1)))
;
x4=or(a(i,k),a(i,(k+1)));
z4=and(x4,n24(i));
n31(i)=not(or(z4,y4));
n32(i)=not(n31(i));
s=and(n3(i),n11(i));
g=and(n19(i),n27(i));
n33(i)=not(and(s,g));
n34(i)=not(n33(i));
n35(i)=not(and(n34(i),a(
i,p)));
n36(i)=not(n35(i));
n37(i)=not(or(n36(i),n32
(i)));
n38(i)=not(n37(i));
end;
c1=0;
c2=0;c3=0;c4=0;c5=0;c
6=0;c7=0;c8=0;c9=0;c1
0=0;c11=0;c12=0;c13=0
;c14=0;c15=0;c16=0;c1
7=0;
c18=0;c19=0;c20=0;c21
=0;c22=0;c23=0;c24=0;
c25=0;c26=0;c27=0;c28
=0;c29=0;c30=0;c31=0;
c32=0;c33=0;c34=0;c35
=0;c36=0;c37=0;c38=0;
for i=1:512;
for j=1:512;
if n1(i)==n1(j);
c1=c1;
else;
c1=c1+1;
end;
If n2(i)==n2(j);
c2=c2;
else;
c2=c2+1;
end;
If n3(i)==n3(j);
c3=c3;
else;
c3=c3+1;
end;
If n4(i)==n4(j)
c4=c4;
else;
c4=c4+1;
end;
if n5(i)==n5(j);
c5=c5;
else
c5=c5+1;
end;
if n6(i)==n6(j)
c6=c6;
else;
c6=c6+1;
end;
if n7(i)==n7(j)
c7=c7;
else;
c7=c7+1;
end;
if n8(i)==n8(j);
c8=c8;
else;
c8=c8+1;
end;
if n9(i)==n9(j);
c9=c9;
else;
c9=c9+1;
end;
if n10(i)==n10(j);
c10=c10;
else;
c10=c10+1;
end;
if n11(i)==n11(j);
c11=c11;
else;
c11=c11+1;
end;
if n12(i)==n12(j);
c12=c12;
else;
c12=c12+1;
end;
if n13(i)==n13(j);
c13=c13;
else;
c13=c13+1;
end;
if n14(i)==n14(j);
c14=c14;
else;
c14=c14+1;
end;
if n15(i)==n15(j);
c15=c15;
else;
c15=c15+1;
end;
if n16(i)==n16(j);
c16=c16;
else;
c16=c16+1;
end;
if n17(i)==n17(j);
c17=c17;
else;
c17=c17+1;
end;
if n18(i)==n18(j);
c18=c18;
else;
c18=c18+1;
end;
if n19(i)==n19(j);
c19=c19;
else;
c19=c19+1;
end;
if n20(i)==n20(j);
c20=c20;
else;
```

```
c20=c20+1;
end;
if n21(i)==n21(j);
c21=c21;
else;
c21=c21+1;
end;
if n22(i)==n22(j);
c22=c22;
else;
c22=c22+1;
end;
if n23(i)==n23(j);
c23=c23;
else;
c23=c23+1;
end;
if n24(i)==n24(j);
c24=c24;
else;
c24=c24+1;
end;
if n25(i)==n25(j);
c25=c25;
else;
c25=c25+1;
end;
if n26(i)==n26(j);
c26=c26;
else;
c26=c26+1;
end;
if n27(i)==n27(j);
c27=c27;
else;
c27=c27+1;
end;
if n28(i)==n28(j);
c28=c28;
else;
c28=c28+1;
end;
if n29(i)==n29(j);
c29=c29;
else;
c29=c29+1;
end;
if n30(i)==n30(j);
c30=c30;
else;
c30=c30+1;
end;
if n31(i)==n31(j);
c31=c31;
else;
c31=c31+1;
end;
if n32(i)==n32(j);
c32=c32;
else;
c32=c32+1;
end;
if n33(i)==n33(j);
c33=c33;
else;
c33=c33+1;
end;
if n34(i)==n34(j);
c34=c34;
else;
c34=c34+1;
end;
if n35(i)==n35(j);
c35=c35;
else;
c35=c35+1;
end;
if n36(i)==n36(j);
c36=c36;
else;
c36=c36+1;
end;
if n37(i)==n37(j);
c37=c37;
else;
c37=c37+1;
end;
if n38(i)==n38(j);
c38=c38;
else;
c38=c38+1;
end;
end;
end;
count=c1+c2+c3+c4+c5
+c6+c7+c8+c9+c10+c11
+c12+c13+c14+c15+c16
+c17+c18+c19+c20+c21
+c22+c23+c24+c25+c26
+c27+c28+c29+c30+c31
+c32+c33+c34+c35+c36
+c37+c38;
aver=count/(512*512);
save rcsck.matsver
```

3) Program for Carry select adder

```
clear all
load inpgen4.mat
a
n=input('inputno. of bits
= ')
p=(2*n)+1
for i=1:512;
cin1=0;
n1(i)=not(a(i,1));
n2(i)=not(a(i,2));
n3(i)=xor(a(i,1),a(i,2));
n4(i)=not(cin1);
n5(i)=xor(n3(i), cin1);
n6(i)=not(n3(i));
y1=and(a(i,1),a(i,2));
x1=or(a(i,1),a(i,2));
z1=and(x1,cin1);
n7(i)=not(or(z1,y1));
n8(i)=not(n7(i));

k=3;
n9(i)=not(a(i,k));
n10(i)=not(a(i,(k+1)));
n11(i)=xor(a(i,k),a(i,(k+
1)));
n12(i)=not(n8(i));

n13(i)=xor(n11(i),n8(i));
n14(i)=not(n11(i));

y2=and(a(i,k),a(i,(k+1)))
;
x2=or(a(i,k),a(i,(k+1)));
z2=and(x2,n8(i));
n15(i)=not(or(z2,y2));
n16(i)=not(n15(i));

k=5;
n17(i)=not(a(i,k));
n18(i)=not(a(i,(k+1)));
n19(i)=xor(a(i,k),a(i,(k+
1)));
n20(i)=not(n16(i));
n21(i)=xor(n19(i),n16(i)
);
n22(i)=not(n19(i));

y3=and(a(i,k),a(i,(k+1)))
;
x3=or(a(i,k),a(i,(k+1)));
z3=and(x3,n16(i));
n23(i)=not(or(z3,y3));
n24(i)=not(n23(i));

k=7;
n25(i)=not(a(i,k));
n26(i)=not(a(i,(k+1)));
n27(i)=xor(a(i,k),a(i,(k+
1)));
n28(i)=not(n24(i));
n29(i)=xor(n27(i),n24(i)
);
n30(i)=not(n27(i));

y4=and(a(i,k),a(i,(k+1)))
;
x4=or(a(i,k),a(i,(k+1)));
z4=and(x4,n24(i));
n31(i)=not(or(z4,y4));
n32(i)=not(n31(i));
cin2=1;
n31(i)=not(a(i,1));
n32(i)=not(a(i,2));

n33(i)=xor(a(i,1),a(i,2));
n34(i)=not(cin2);
n35(i)=xor(n33(i),
cin2);
n36(i)=not(n33(i));
y5=and(a(i,1),a(i,2));
x5=or(a(i,1),a(i,2));
z5=and(x5,cin2);
n37(i)=not(or(z5,y5));
n38(i)=not(n37(i));

k=3;
n39(i)=not(a(i,k));
n40(i)=not(a(i,(k+1)));
n41(i)=xor(a(i,k),a(i,(k+
1)));
n42(i)=not(n38(i));

43(i)=xor(n11(i),n38(i));
n44(i)=not(n41(i));

y6=and(a(i,k),a(i,(k+1)))
;
x6=or(a(i,k),a(i,(k+1)));
z6=and(x6,n38(i));
n45(i)=not(or(z6,y6));
n46(i)=not(n45(i));

k=5;
n47(i)=not(a(i,k));
n48(i)=not(a(i,(k+1)));
n49(i)=xor(a(i,k),a(i,(k+
1)));
n50(i)=not(n46(i));

n51(i)=xor(n49(i),n46(i)
);
n52(i)=not(n49(i));

y7=and(a(i,k),a(i,(k+1)))
;
x7=or(a(i,k),a(i,(k+1)));
z7=and(x7,n46(i));
n53(i)=not(or(z7,y7));
n54(i)=not(n53(i));

k=7;
n55(i)=not(a(i,k));
n56(i)=not(a(i,(k+1)));
n57(i)=xor(a(i,k),a(i,(k+
1)));
n58(i)=not(n54(i));
n59(i)=xor(n57(i),n54(i)
);
n60(i)=not(n57(i));

y8=and(a(i,k),a(i,(k+1)))
;
x8=or(a(i,k),a(i,(k+1)));
z8=and(x8,n54(i));
n61(i)=not(or(z8,y8));
n62(i)=not(n61(i));

n63(i)=not(and(n62(i),a(
i,p)));
n64(i)=not(n63(i));

n65(i)=not(or(n64(i),n32
(i)));
n66(i)=not(n65(i));
n67(i)=not(n5(i));
```

```
n68(i)=not(n35(i));
n69(i)=not(a(i,p));
q1=and(n35(i),a(i,p));
p1=and(n5(i),n69(i));
n70(i)=or(q1,p1);
n71(i)=not(n12(i));
n72(i)=not(n43(i));
n73(i)=not(a(i,p));
q2=and(n43(i),a(i,p));
p2=and(n12(i),n73(i));
n74(i)=or(q2,p2);
n75(i)=not(n21(i));
n76(i)=not(n51(i));
n77(i)=not(a(i,p));
q3=and(n51(i),a(i,p));
p3=and(n21(i),n77(i));
n78(i)=or(q3,p3);
n79(i)=not(n29(i));
n80(i)=not(n59(i));
n81(i)=not(a(i,p));
q4=and(n59(i),a(i,p));
p4=and(n29(i),n81(i));
n82(i)=or(q4,p4);
end;

c1=0;c2=0;c3=0;c4=0;c
5=0;c6=0;c7=0;c8=0;c9
=0;c10=0;c11=0;c12=0;
c13=0;c14=0;c15=0;c16
=0;c17=0;c18=0;c19=0;
c20=0;c21=0;c22=0;c23
=0;c24=0;c25=0;c26=0;
c27=0;c28=0;c29=0;c30
=0;c31=0;c32=0;c33=0;
c34=0;c35=0;c36=0;c37
=0;c38=0;c39=0;c40=0;
c41=0;c42=0;c43=0;c44
=0;c45=0;c46=0;c47=0;
c48=0;
c49=0;c50=0;c51=0;c52
=0;c53=0;c54=0;c55=0;
c56=0;c57=0;c58=0;c59
=0;c60=0;c61=0;c62=0;
c63=0;c64=0;c65=0;c66
=0;c67=0;c68=0;c69=0;
c70=0;c71=0;c72=0;c73
=0;c74=0;c75=0;c76=0;
c77=0;c78=0;
c79=0;c80=0;c81=0;c82
=0;
for i=1:512;
 for j=1:512;
 if n1(i)==n1(j);
 c1=c1;
 else ;
 c1=c1+1;
 end;
if n2(i)==n2(j);
 c2=c2;
 else ;
 c2=c2+1;
 end;
if n3(i)==n3(j);
 c3=c3;
 else ;
 c3=c3+1;
 end;
if n4(i)==n4(j)
 c4=c4;
 else ;
 c4=c4+1;
 end;
if n5(i)==n5(j);
 c5=c5;
 else
 c5=c5+1;
 end;
if n6(i)==n6(j)
 c6=c6;
 else ;
 c6=c6+1;
 end;
if n7(i)==n7(j)
 c7=c7;
 else ;
 c7=c7+1;
 end;
if n8(i)==n8(j);
 c8=c8;
 else ;
 c8=c8+1;
 end;
if n9(i)==n9(j);
 c9=c9;
 else ;
 c9=c9+1;
 end;
if n10(i)==n10(j);
 c10=c10;
 else ;
 c10=c10+1;
 end;
if n11(i)==n11(j);
 c11=c11;
 else ;
 c11=c11+1;
 end;
if n12(i)==n12(j);
 c12=c12;
 else ;
 c12=c12+1;
 end;
if n13(i)==n13(j);
 c13=c13;
 else ;
 c13=c13+1;
 end;
if n14(i)==n14(j);
 c14=c14;
 else ;
 c14=c14+1;
 end;
if n15(i)==n15(j);
 c15=c15;
 else ;
 c15=c15+1;
 end;
 if n16(i)==n16(j);
 c16=c16;
 else ;
 c16=c16+1;
 end;
 if n17(i)==n17(j);
 c17=c17;
 else ;
 c17=c17+1;
 end;
if n18(i)==n18(j);
 c18=c18;
 else ;
 c18=c18+1;
 end;
if n19(i)==n19(j);
 c19=c19;
 else ;
 c19=c19+1;
 end;
if n20(i)==n20(j);
 c20=c20;
 else ;
 c20=c20+1;
 end;
 if n21(i)==n21(j);
 c21=c21;
 else ;
 c21=c21+1;
 end;
 if n22(i)==n22(j);
 c22=c22;
 else ;
 c22=c22+1;
 end;
if n23(i)==n23(j);
 c23=c23;
 else ;
 c23=c23+1;
 end;
 if n24(i)==n24(j);
 c24=c24;
 else ;
 c24=c24+1;
 end;
 if n25(i)==n25(j);
 c25=c25;
 else ;
 c25=c25+1;
 end;
if n26(i)==n26(j);
 c26=c26;
 else ;
 c26=c26+1;
 end;
if n27(i)==n27(j);
 c27=c27;
 else ;
 c27=c27+1;
 end;
 if n28(i)==n28(j);
 c28=c28;
 else ;
 c28=c28+1;
 end;
 if n29(i)==n29(j);
 c29=c29;
 else ;
 c29=c29+1;
 end;
if n30(i)==n30(j);
 c30=c30;
 else ;
 c30=c30+1;
 end;
 if n31(i)==n31(j);
 c31=c31;
 else ;
 c31=c31+1;
 end;
 if n32(i)==n32(j);
 c32=c32;
 else ;
 c32=c32+1;
 end;
if n33(i)==n33(j);
 c33=c33;
 else ;
 c33=c33+1;
 end;
 if n34(i)==n34(j);
 c34=c34;
 else ;
 c34=c34+1;
 end;
if n35(i)==n35(j);
 c35=c35;
 else ;
 c35=c35+1;
 end;
if n36(i)==n36(j);
 c36=c36;
 else ;
 c36=c36+1;
 end;
if n37(i)==n37(j);
 c37=c37;
 else ;
 c37=c37+1;
 end;
 if n38(i)==n38(j);
 c38=c38;
 else ;
 c38=c38+1;
 end;

 if n39(i)==n39(j);
 c39=c39;
 else ;
 c39=c39+1;
 end;
 if n40(i)==n40(j);
 c40=c40;
 else ;
 c40=c40+1;
 end;
 if n41(i)==n41(j);
 c41=c41;
 else ;
 c41=c41+1;
 end;
 if n42(i)==n42(j);
 c42=c42;
 else ;
 c42=c42+1;
 end;

 if n43(i)==n43(j);
 c43=c43;
 else ;
 c43=c43+1;
 end;

 if n44(i)==n44(j);
 c44=c44;
 else ;
 c44=c44+1;
 end;
 if n45(i)==n45(j);
 c45=c45;
 else ;
 c45=c45+1;
 end;
 if n46(i)==n46(j);
 c46=c46;
 else ;
 c46=c46+1;
 end;
 if n47(i)==n47(j)
 c47=c47;
 else ;
 c47=c47+1;
 end;
 if n48(i)==n48(j);
 c48=c48;
```

```
else
c48=c48+1;
end;
if n49(i)==n49(j)
c49=c49;
else ;
c49=c49+1;
end;
if n50(i)==n50(j)
c50=c50;
else ;
c50=c50+1;
end;
if n51(i)==n51(j);
c51=c51;
else ;
c51=c51+1;
end;
if n52(i)==n52(j);
c52=c52;
else ;
c52=c52+1;
end;
if n53(i)==n53(j);
c53=c53;
else ;
c53=c53+1;
end;
if n54(i)==n54(j);
c54=c54;
else ;
c54=c54+1;
end;
if n55(i)==n55(j);
c55=c55;
else ;
c55=c55+1;
end;
if n56(i)==n56(j);
c56=c56;
else ;
c56=c56+1;
end;
if n57(i)==n57(j);
c57=c57;
else ;
c57=c57+1;
end;
if n58(i)==n58(j);
c58=c58;
```

```
else ;
c58=c58+1;
end;
if n59(i)==n59(j);
c59=c59;
else ;
c59=c59+1;
end;
if n60(i)==n60(j);
c60=c60;
else ;
c60=c60+1;
end;
if n61(i)==n61(j);
c61=c61;
else ;
c61=c61+1;
end;
if n62(i)==n62(j);
c62=c62;
else ;
c62=c62+1;
end;
if n63(i)==n63(j);
c63=c63;
else ;
c63=c63+1;
end;
if n64(i)==n64(j);
c64=c64;
else ;
c64=c64+1;
end;
if n65(i)==n65(j);
c65=c65;
else ;
c65=c65+1;
end;
if n66(i)==n66(j);
c66=c66;
else ;
c66=c66+1;
end;
if n67(i)==n67(j);
c67=c67;
else ;
c67=c67+1;
end;
if n68(i)==n68(j);
c68=c68;
```

```
else ;
c68=c68+1;
end;
if n69(i)==n69(j);
c69=c69;
else ;
c69=c69+1;
end;
if n70(i)==n70(j);
c70=c70;
else ;
c70=c70+1;
end;
if n71(i)==n71(j);
c71=c71;
else ;
c71=c71+1;
end;
if n72(i)==n72(j);
c72=c72;
else ;
c72=c72+1;
end;
if n73(i)==n73(j);
c73=c73;
else ;
c73=c73+1;
end;
if n74(i)==n74(j);
c74=c74;
else ;
c74=c74+1;
end;
if n75(i)==n75(j);
c75=c75;
else ;
c75=c75+1;
end;
if n76(i)==n76(j);
c76=c76;
else ;
c76=c76+1;
end;
if n77(i)==n77(j);
c77=c77;
else ;
c77=c77+1;
end;
if n78(i)==n78(j);
c78=c78;
```

```
else ;
c78=c78+1;
end;
if n79(i)==n79(j);
c79=c79;
else ;
c79=c79+1;
end;
if n80(i)==n80(j);
c80=c80;
else ;
c80=c80+1;
end;
if n81(i)==n81(j);
c81=c81;
else ;
c81=c81+1;
end;
if n82(i)==n82(j);
c82=c82;
else ;
c82=c82+1;
end;
end;
end;
count=c1+c2+c3+c4+c5
+c6+c7+c8+c9+c10+c11
+c12+c13+c14+c15+c16
+c17+c18+c19+c20+c21
+c22+c23+c24+c25+c26
+c27+c28+c29+c30+c31
+c32+c33+c34+c35+c36
+c37+c38+c39+c40+c41
+c42+c43+c44+c45+c46
+c47+c48+c49+c50+c51
+c52+c53+c54+c55+c56
+c57+c58+c59+c60+c61
+c62+c63+c64+c65+c66
+c67+c68+c69+c70+c71
+c72+c73+c74+c75+c76
+c77+c78+c79+80+c81
+c82;
aver=count/(512*512);
save rescrl.mat aver
```

4) Program for Conditional sum adder

```
clear all
load inpgen4.mat
a;
n=input('inputno. of bits
= ')
p=(2^n)+1
for i=1:512;
n1(i)=not(a(i,1));
n2(i)=not((a(i,2)));
n3(i)=xor(a(i,1),a(i,2));
n4(i)=not((n3(i)));
n5(i)=not(xor(a(i,1),a(i,2
)));
n6(i)=not(n5(i));
n7(i)=not(and(a(i,1),a(i,
2)));
n8(i)=not(n7(i));
n9(i)=not(or(a(i,1),a(i,2)
));
```

```
n10(i)=not(n9(i));
n41(i)=not(n6(i));
n42(i)=not(n4(i));
n43(i)=not(a(i,p));
n44(i)=or((and(a(i,p),n6(
i))),(and(n43(i),n4(i))));

k=3;
n11(i)=not(a(i,k));
n12(i)=not((a(i,(k+1))));
n13(i)=xor(a(i,k),a(i,(k+
1)));
n14(i)=not((n13(i)));
n15(i)=not(xor(a(i,k),a(i,
(k+1))));
n16(i)=not(n15(i));
n17(i)=not(and(a(i,k),a(i
,(k+1))));
n18(i)=not(n17(i));
```

```
n19(i)=not(or(a(i,k),a(i,(
k+1))));
n20(i)=not(n19(i));
n45(i)=not(n16(i));
n46(i)=not(n14(i));
n47(i)=not(n8(i));
n48(i)=or((and(n8(i),n16
(i))),(and(n47(i),n14(i)))
);
n49(i)=not(n16(i));
n50(i)=not(n14(i));
n51(i)=not(n10(i));
n52(i)=or((and(n10(i),n1
6(i))),(and(n47(i),n14(i))
));
n53(i)=not(n48(i));
n54(i)=not(n52(i));
n55(i)=not(a(i,p));
```

```
n56(i)=or((and(a(i,p),n4
8(i))),(and(n55(i),n52(i))
));
n57(i)=not(n18(i));
n58(i)=not(n20(i));
n59(i)=not(n8(i));
n60(i)=or((and(n59(i),n2
0(i))),(and(n8(i),n18(i)))
);
n61(i)=not(n18(i));
n62(i)=not(n20(i));
n63(i)=not(n10(i));
n64(i)=or((and(n63(i),n2
0(i))),(and(n10(i),n18(i))
));

k=5;
n21(i)=not(a(i,k));
n22(i)=not((a(i,(k+1))));
```

```
n23(i)=xor(a(i,k),a(i,(k+
1)));
n24(i)=not((n11(i)));
n25(i)=not(xor(a(i,k),a(i,
(k+1))));
n26(i)=not(n25(i));
n27(i)=not(and(a(i,k),a(i
,(k+1))));
n28(i)=not(n27(i));
n29(i)=not(or(a(i,k),a(i,(
k+1))));
n30(i)=not(n29(i));
n65(i)=not(n24(i));
n66(i)=not(n26(i));
n67(i)=not(n60(i));
n68(i)=or((and(n60(i),n2
4(i))),(and(n67(i),n26(i))
));
n69(i)=not(n24(i));
n70(i)=not(n26(i));
n71(i)=not(n64(i));
n72(i)=or((and(n64(i),n2
4(i))),(and(n71(i),n26(i))
));
n73(i)=not(n68(i));
n74(i)=not(n72(i));
n75(i)=not(a(i,p));
n76(i)=or((and(a(i,p),n7
2(i))),(and(n75(i),n68(i))
));
n77(i)=not(n28(i));
n78(i)=not(n30(i));
n79(i)=not(n60(i));
n80(i)=or((and(n30(i),n6
0(i))),(and(n28(i),n79(i))
));
n81(i)=not(n28(i));
n82(i)=not(n30(i));
n83(i)=not(n64(i));
n84(i)=or((and(n83(i),n2
8(i))),(and(n64(i),n30(i))
));

k=7;
n31(i)=not(a(i,k));
n32(i)=not((a(i,(k+1))));
n33(i)=xor(a(i,k),a(i,(k+
1)));
n34(i)=not((n33(i)));
n35(i)=not(xor(a(i,k),a(i,
(k+1))));
n36(i)=not(n35(i));
n37(i)=not(and(a(i,k),a(i
,(k+1))));
n38(i)=not(n37(i));
n39(i)=not(or(a(i,k),a(i,(
k+1))));
n40(i)=not(n39(i));
n85(i)=not(n34(i));
n86(i)=not(n36(i));
n87(i)=not(n80(i));
n88(i)=or((and(n80(i),n3
4(i))),(and(n87(i),n36(i))
));
n89(i)=not(n34(i));
n90(i)=not(n36(i));
n91(i)=not(n84(i));
n92(i)=or((and(n84(i),n3
4(i))),(and(n91(i),n36(i))
));
n93(i)=not(n88(i));
n94(i)=not(n92(i));
n95(i)=not(a(i,p));
n96(i)=or((and(a(i,p),n9
2(i))),(and(n95(i),n88(i))
));
n97(i)=not(n38(i));
n98(i)=not(n40(i));
n99(i)=not(n80(i));
n100(i)=or((and(n80(i),n
40(i))),(and(n99(i),n38(i
))));
n101(i)=not(n38(i));
n102(i)=not(n40(i));
n103(i)=not(n84(i));
n104(i)=or((and(n84(i),n
40(i))),(and(n103(i),n38(
i))));
n105(i)=not(n100(i));
n106(i)=not(n104(i));
n107(i)=not(a(i,p));
n108(i)=or((and(n104(i),
a(i,p))),(and(n100(i),n10
7(i))));
end;
c1=0;
c2=0;c3=0;c4=0;c5=0;c
6=0;c7=0;c8=0;c9=0;c1
0=0;c11=0;c12=0;c13=0
;c14=0;c15=0;c16=0;c1
7=0;
c18=0;c19=0;c20=0;c21
=0;c22=0;c23=0;c24=0;
c25=0;c26=0;c27=0;c28
=0;c29=0;c30=0;c31=0;
c32=0;
c33=0;c34=0;c35=0;c36
=0;c37=0;c38=0;c39=0;
c40=0;c41=0;c42=0;c43
=0;c44=0;c45=0;c46=0;
c47=0;c48=0;
c49=0;c50=0;c51=0;c52
=0;c53=0;c54=0;c55=0;
c56=0;c57=0;c58=0;c59
=0;c60=0;c61=0;c62=0;
c63=0;
c64=0;c65=0;c66=0;c67
=0;c68=0;c69=0;c70=0;
c71=0;c72=0;c73=0;c74
=0;c75=0;c76=0;c77=0;
c78=0;
c79=0;c80=0;c81=0;c82
=0;c83=0;c84=0;c85=0;
c86=0;c87=0;c88=0;c89
=0;c90=0;c91=0;c92=0;
c93=0;
c94=0;c95=0;c96=0;c97
=0;c98=0;c99=0;c100=0
;c101=0;c102=0;c103=0
;c104=0;c105=0;c106=0
;c107=0;c108=0;
for i=1:512;
for j=1:512;
 if n1(i)==n1(j);
c1=c1;
else;
c1=c1+1;
end;
if n2(i)==n2(j);
c2=c2;
else;
c2=c2+1;
end;
if n3(i)==n3(j);
c3=c3;
else;
c3=c3+1;
end;
if n4(i)==n4(j)
c4=c4;
else;
c4=c4+1;
end;
if n5(i)==n5(j);
c5=c5;
else
c5=c5+1;
end;
if n6(i)==n6(j)
c6=c6;
else;
c6=c6+1;
end;
if n7(i)==n7(j)
c7=c7;
else;
c7=c7+1;
end;
if n8(i)==n8(j);
c8=c8;
else;
c8=c8+1;
end;
if n9(i)==n9(j);
c9=c9;
else;
c9=c9+1;
end;
if n10(i)==n10(j);
c10=c10;
else;
c10=c10+1;
end;
if n11(i)==n11(j);
c11=c11;
else;
c11=c11+1;
end;
if n12(i)==n12(j);
c12=c12;
else;
c12=c12+1;
end;
if n13(i)==n13(j);
c13=c13;
else;
c13=c13+1;
end;
if n14(i)==n14(j);
c14=c14;
else;
c14=c14+1;
end;
if n15(i)==n15(j);
c15=c15;
else;
c15=c15+1;
end;
 if n16(i)==n16(j);
c16=c16;
else;
c16=c16+1;
end;
 if n17(i)==n17(j);
c17=c17;
else;
c17=c17+1;
end;
if n18(i)==n18(j);
c18=c18;
else;
c18=c18+1;
end;
if n19(i)==n19(j);
c19=c19;
else;
c19=c19+1;
end;
if n20(i)==n20(j);
c20=c20;
else;
c20=c20+1;
end;
 if n21(i)==n21(j);
c21=c21;
else;
c21=c21+1;
end;
 if n22(i)==n22(j);
c22=c22;
else;
c22=c22+1;
end;
if n23(i)==n23(j);
c23=c23;
else;
c23=c23+1;
end;
 if n24(i)==n24(j);
c24=c24;
else;
c24=c24+1;
end;
 if n25(i)==n25(j);
c25=c25;
else;
c25=c25+1;
end;
if n26(i)==n26(j);
c26=c26;
else;
c26=c26+1;
end;
if n27(i)==n27(j);
c27=c27;
else;
c27=c27+1;
end;
 if n28(i)==n28(j);
c28=c28;
else;
c28=c28+1;
end;
 if n29(i)==n29(j);
c29=c29;
else;
c29=c29+1;
end;
if n30(i)==n30(j);
c30=c30;
else;
c30=c30+1;
end;
 if n31(i)==n31(j);
c31=c31;
else;
c31=c31+1;
end;
if n32(i)==n32(j);
c32=c32;
else;
c32=c32+1;
```

```
end;
if n33(i)==n33(j);
c33=c33;
else;
c33=c33+1;
end;
if n34(i)==n34(j);
c34=c34;
else;
c34=c34+1;
end;
if n35(i)==n35(j);
c35=c35;
else;
c35=c35+1;
end;
if n36(i)==n36(j);
c36=c36;
else;
c36=c36+1;
end;
if n37(i)==n37(j);
c37=c37;
else;
c37=c37+1;
end;
if n38(i)==n38(j);
c38=c38;
else;
c38=c38+1;
end;
if n39(i)==n39(j);
c39=c39;
else;
c39=c39+1;
end;
if n40(i)==n40(j);
c40=c40;
else;
c40=c40+1;
end;
if n41(i)==n41(j);
c41=c41;
else;
c41=c41+1;
end;
 if n42(i)==n42(j);
c42=c42;
else;
c42=c42+1;
end;
if n43(i)==n43(j);
c43=c43;
else;
c43=c43+1;
end;
 if n44(i)==n44(j);
c44=c44;
else;
c44=c44+1;
end;
if n45(i)==n45(j);
c45=c45;
else;
c45=c45+1;
end;
if n46(i)==n46(j);
c46=c46;
else;
c46=c46+1;
end;
if n47(i)==n47(j)
c47=c47;
else;
c47=c47+1;
end;
if n48(i)==n48(j);
c48=c48;
else
c48=c48+1;
end;
if n49(i)==n49(j)
c49=c49;
else;
c49=c49+1;
end;
if n50(i)==n50(j)
c50=c50;
else;
c50=c50+1;
end;
if n51(i)==n51(j);
c51=c51;
else;
c51=c51+1;
end;
if n52(i)==n52(j);
c52=c52;
else;
c52=c52+1;
end;
if n53(i)==n53(j);
c53=c53;
else;
c53=c53+1;
end;
if n54(i)==n54(j);
c54=c54;
else;
c54=c54+1;
end;
if n55(i)==n55(j);
c55=c55;
else;
c55=c55+1;
end;
if n56(i)==n56(j);
c56=c56;
else;
c56=c56+1;
end;
if n57(i)==n57(j);
c57=c57;
else;
c57=c57+1;
end;
if n58(i)==n58(j);
c58=c58;
else;
c58=c58+1;
end;
 if n59(i)==n59(j);
c59=c59;
else;
c59=c59+1;
end;
 if n60(i)==n60(j);
c60=c60;
else;
c60=c60+1;
end;
if n61(i)==n61(j);
c61=c61;
else;
c61=c61+1;
end;
if n62(i)==n62(j);
c62=c62;
else;
c62=c62+1;
end;
if n63(i)==n63(j);
c63=c63;
else;
c63=c63+1;
end;
 if n64(i)==n64(j);
c64=c64;
else;
c64=c64+1;
end;
 if n65(i)==n65(j);
c65=c65;
else;
c65=c65+1;
end;
if n66(i)==n66(j);
c66=c66;
else;
c66=c66+1;
end;
 if n67(i)==n67(j);
c67=c67;
else;
c67=c67+1;
end;
 if n68(i)==n68(j);
c68=c68;
else;
c68=c68+1;
end;
if n69(i)==n69(j);
c69=c69;
else;
c69=c69+1;
end;
if n70(i)==n70(j);
c70=c70;
else;
c70=c70+1;
end;
 if n71(i)==n71(j);
c71=c71;
else;
c71=c71+1;
end;
 if n72(i)==n72(j);
c72=c72;
else;
c72=c72+1;
end;
if n73(i)==n73(j);
c73=c73;
else;
c73=c73+1;
end;
 if n74(i)==n74(j);
c74=c74;
else;
c74=c74+1;
end;
if n75(i)==n75(j);
c75=c75;
else;
c75=c75+1;
end;
if n76(i)==n76(j);
c76=c76;
else;
c76=c76+1;
end;
if n77(i)==n77(j);
c77=c77;
else;
c77=c77+1;
end;
if n78(i)==n78(j);
c78=c78;
else;
c78=c78+1;
end;
if n79(i)==n79(j);
c79=c79;
else;
c79=c79+1;
end;
if n80(i)==n80(j);
c80=c80;
else;
c80=c80+1;
end;
 if n81(i)==n81(j);
c81=c81;
else;
c81=c81+1;
end;

 if n82(i)==n82(j);
c82=c82;
else;
c82=c82+1;
end;
 if n83(i)==n83(j);
c83=c83;
else;
c83=c83+1;
end;
 if n84(i)==n84(j);
c84=c84;
else;
c84=c84+1;
end;
 if n85(i)==n85(j);
c85=c85;
else;
c85=c85+1;
end;
 if n86(i)==n86(j);
c86=c86;
else;
c86=c86+1;
end;
 if n87(i)==n87(j);
c87=c87;
else;
c87=c87+1;
end;
 if n88(i)==n88(j);
c88=c88;
else;
c88=c88+1;
end;
 if n89(i)==n89(j);
c89=c89;
else;
c89=c89+1;
end;
 if n90(i)==n90(j);
c90=c90;
else;
c90=c90+1;
end;
```

```
if n91(i)==n91(j);
c91=c91;
else;
c91=c91+1;
end;
if n92(i)==n92(j);
c92=c92;
else;
c92=c92+1;
end;
if n93(i)==n93(j);
c93=c93;
else;
c93=c93+1;
end;
if n94(i)==n94(j);
c94=c94;
else;
c94=c94+1;
end;
if n95(i)==n95(j);
c95=c95;
else;
c95=c95+1;
end;
if n96(i)==n96(j);
c96=c96;
else;
c96=c96+1;
end;
if n97(i)==n97(j);
c97=c97;
else;
c97=c97+1;
end;
if n98(i)==n98(j);
c98=c98;
else;
c98=c98+1;
end;
if n99(i)==n99(j);
c99=c99;
else;
c99=c99+1;
end;
if n100(i)==n100(j);
c100=c100;
else;
c100=c100+1;
end;
if n101(i)==n101(j);
c101=c101;
else;
c101=c101+1;
end;
if n102(i)==n102(j);
c102=c102;
else;
c102=c102+1;
end;
if n103(i)==n103(j);
c103=c103;
else;
c103=c103+1;
end;
if n104(i)==n104(j);
c104=c104;
else;
c104=c104+1;
end;
if n105(i)==n105(j);
c105=c105;
else;
c105=c105+1;
end;
if n106(i)==n106(j);
c106=c106;
else;
c106=c106+1;
end;
if n107(i)==n107(j);
c107=c107;
else;
c107=c107+1;
end;
if n108(i)==n108(j);
c108=c108;
else;
c108=c108+1;
end;
end;
end;
count1=c1+c2+c3+c4+c5+c6+c7+c8+c9+c10+c11+c12+c13+c14+c15+c16+c17+c18+c19+c20+c21+c22+c23+c24+c25+c26+c27+c28+c29+c30+c31+c32+c33+c34+c35+c36+c37+c38+c39+c40+c41+c42+c43+c44+c45+c46+c47+c48+c49+c50;
count2=count1+c51+c52+c53+c54+c55+c56+c57+c58+c59+c60+c61+c62+c63+c64+c65+c66+c67+c68+c69+c70+c71+c72+c73+c74+c75+c76+c77+c78+c79+80+c81+c82+c83+c84+c85;
count=count2+c86+c87+c88+c89+c90+c91+c92+c93+c94+c95+c96+c97+c98+c99+c100+c102+c103+c104+c105+c106+c107+c108;

aver=count/(512*512);
save reacnda.mat aver
save reacnda.mat count
```

5) Program for Carry look-ahead adder

```
clear all
load inpgen4.mat
a
n=input('inputno. of bits = ')
p=(2^n)+1
for i=1:512;
n1(i)=not(and(a(i,1),a(i,2)));
n2(i)=not(n1(i));
n3(i)=not(n2(i));
n4(i)=not(or(a(i,1),a(i,2)));
n5(i)=not(n4(i));
n6(i)=not(n5(i));
n7(i)=not(a(i,p));

k=3;
n9(i)=not(and(a(i,k),a(i,(k+1))));
n10(i)=not(n9(i));
n11(i)=not(n10(i));
n12(i)=not(or(a(i,k),a(i,(k+1))));
n13(i)=not(n12(i));
n14(i)=not(n13(i));
n17(i)=not(or(and(n5(i),n10(i)),n13(i)));
n18(i)=not(n17(i));
n19(i)=not(and(n2(i),n10(i)));
n20(i)=not(n19(i));
n21(i)=not(or(n5(i),(and(n2(i),a(i,p)))));
n22(i)=not(n21(i));
n15(i)=not(n22(i));
x2=and(n10(i),and(n14(i),n22(i)));
y2=and(n15(i),(or(n13(i),n11(i))));
n16(i)=or(x2,y2);
x1=and(n2(i),and(n6(i),a(i,p)));
y1=and((a(i,p)),or(n5(i),n3(i)));
n8(i)=or(x1,y1);

k=5;
n23(i)=not(and(a(i,k),a(i,(k+1))));
n24(i)=not(n23(i));
n25(i)=not(n24(i));
n26(i)=not(or(a(i,k),a(i,(k+1))));
n27(i)=not(n26(i));
n28(i)=not(n27(i));

k=7;
n31(i)=not(and(a(i,k),a(i,(k+1))));
n32(i)=not(n31(i));
n33(i)=not(n32(i));
n34(i)=not(or(a(i,k),a(i,(k+1))));
n35(i)=not(n34(i));
n36(i)=not(n35(i));
n39(i)=not(or(and(n27(i),n32(i)),n35(i)));
n40(i)=not(n39(i));
n41(i)=not(and(n24(i),n32(i)));
n42(i)=not(n41(i));
n45(i)=not(or(and(n18(i),n42(i)),n40(i)));
n46(i)=not(n45(i));
n47(i)=not(and(n20(i),n42(i)));
n48(i)=not(n47(i));
n49(i)=not(or(n18(i),(and(n20(i),a(i,p)))));
n50(i)=not(n49(i));
n43(i)=not(or(n27(i),(and(n24(i),n50(i)))));
n44(i)=not(n43(i));
n37(i)=not(n43(i));
x4=and(n32(i),and(n34(i),n43(i)));
y4=and(n37(i),or(n35(i),n33(i)));
n38(i)=or(x4,y4);
n29(i)=not(n50(i));
x3=and(n24(i),and(n28(i),n50(i)));
y3=and(n29(i),or(n27(i),n25(i)));
n30(i)=or(x3,y3);
end;
c1=0;
c2=0;c3=0;c4=0;c5=0;c6=0;c7=0;c8=0;c9=0;c10=0;c11=0;c12=0;c13=0;c14=0;c15=0;c16=0;c17=0;
c18=0;c19=0;c20=0;c21=0;c22=0;c23=0;c24=0;c25=0;c26=0;c27=0;c28=0;c29=0;c30=0;c31=0;c32=0;
c33=0;c34=0;c35=0;c36=0;c37=0;c38=0;c39=0;c40=0;c41=0;c42=0;c43=0;c44=0;c45=0;c46=0;c47=0;c48=0;
c49=0;c50=0;
for i=1:512;
 for j=1:512;
 if n1(i)==n1(j);
c1=c1;
else;
c1=c1+1;
end;
if n2(i)==n2(j);
c2=c2;
else;
c2=c2+1;
end;
if n3(i)==n3(j);
c3=c3;
else;
c3=c3+1;
end;
if n4(i)==n4(j)
c4=c4;
else;
c4=c4+1;
end;
if n5(i)==n5(j);
c5=c5;
else
c5=c5+1;
end;
if n6(i)==n6(j)
c6=c6;
else;
c6=c6+1;
end;
if n7(i)==n7(j)
c7=c7;
else;
c7=c7+1;
end;
```

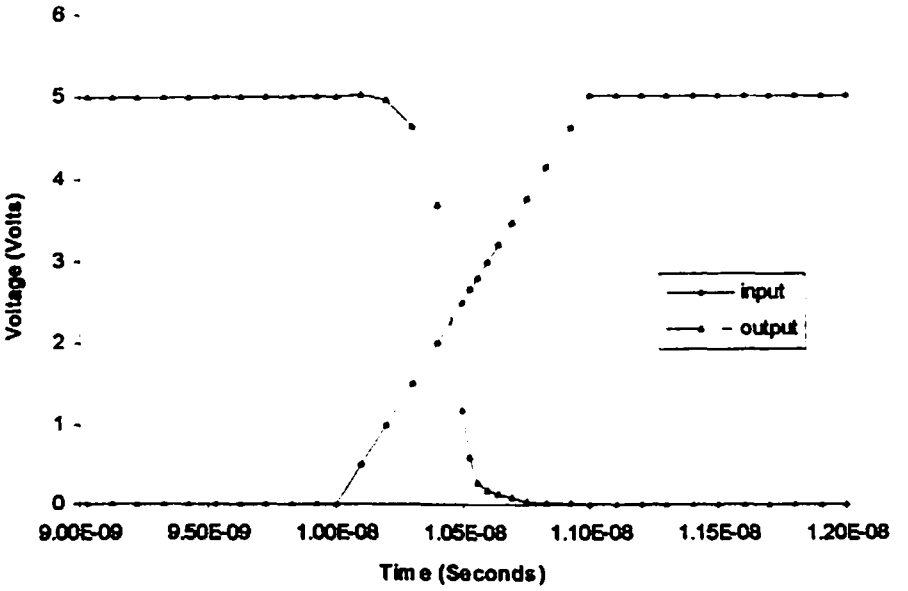
```
if n8(i)==n8(j);
c8=c8;
else;
c8=c8+1;
end;
if n9(i)==n9(j);
c9=c9;
else;
c9=c9+1;
end;
if n10(i)==n10(j);
c10=c10;
else;
c10=c10+1;
end;
if n11(i)==n11(j);
c11=c11;
else;
c11=c11+1;
end;
if n12(i)==n12(j);
c12=c12;
else;
c12=c12+1;
end;
if n13(i)==n13(j);
c13=c13;
else;
c13=c13+1;
end;
if n14(i)==n14(j);
c14=c14;
else;
c14=c14+1;
end;
if n15(i)==n15(j);
c15=c15;
else;
c15=c15+1;
end;
 if n16(i)==n16(j);
c16=c16;
else;
c16=c16+1;
end;
 if n17(i)==n17(j);
c17=c17;
else;
c17=c17+1;
end;
if n18(i)==n18(j);
c18=c18;
else;
c18=c18+1;
end;
if n19(i)==n19(j);
c19=c19;
else;
c19=c19+1;
end;
if n20(i)==n20(j);
c20=c20;
else;
c20=c20+1;
end;
 if n21(i)==n21(j);
c21=c21;
else;
c21=c21+1;
end;
 if n22(i)==n22(j);
c22=c22;
else;
c22=c22+1;
end;
if n23(i)==n23(j);
c23=c23;
else;
c23=c23+1;
end;
 if n24(i)==n24(j);
c24=c24;
else;
c24=c24+1;
end;
 if n25(i)==n25(j);
c25=c25;
else;
c25=c25+1;
end;
if n26(i)==n26(j);
c26=c26;
else;
c26=c26+1;
end;
if n27(i)==n27(j);
c27=c27;
else;
c27=c27+1;
end;
 if n28(i)==n28(j);
c28=c28;
else;
c28=c28+1;
end;
 if n29(i)==n29(j);
c29=c29;
else;
c29=c29+1;
end;
if n30(i)==n30(j);
c30=c30;
else;
c30=c30+1;
end;
 if n31(i)==n31(j);
c31=c31;
else;
c31=c31+1;
end;
if n32(i)==n32(j);
c32=c32;
else;
c32=c32+1;
end;
if n33(i)==n33(j);
c33=c33;
else;
c33=c33+1;
end;
 if n34(i)==n34(j);
c34=c34;
else;
c34=c34+1;
end;
if n35(i)==n35(j);
c35=c35;
else;
c35=c35+1;
end;
if n36(i)==n36(j);
c36=c36;
else;
c36=c36+1;
end;
 if n37(i)==n37(j);
c37=c37;
else;
c37=c37+1;
end;
 if n38(i)==n38(j);
c38=c38;
else;
c38=c38+1;
end;

 if n39(i)==n39(j);
c39=c39;
else;
c39=c39+1;
end;
if n40(i)==n40(j);
c40=c40;
else;
c40=c40+1;
end;
if n41(i)==n41(j);
c41=c41;
else;
c41=c41+1;
end;
 if n42(i)==n42(j);
c42=c42;
else;
c42=c42+1;
end;

 if n43(i)==n43(j);
c43=c43;
else;
c43=c43+1;
end;

 if n44(i)==n44(j);
c44=c44;
else;
c44=c44+1;
end;
if n45(i)==n45(j);
c45=c45;
else;
c45=c45+1;
end;
if n46(i)==n46(j);
c46=c46;
else;
c46=c46+1;
end;
if n47(i)==n47(j)
c47=c47;
else;
c47=c47+1;
end;
if n48(i)==n48(j);
c48=c48;
else
c48=c48+1;
end;
if n49(i)==n49(j)
c49=c49;
else;
c49=c49+1;
end;
if n50(i)==n50(j)
c50=c50;
else;
c50=c50+1;
end;
end;
```

```
end;
count=c1+c2+c3+c4+c5
+c6+c7+c8+c9+c10+c11
+c12+c13+c14+c15+c16
+c17+c18+c19+c20+c21
+c22+c23+c24+c25+c26
+c27+c28+c29+c30+c31
+c32+c33+c34+c35+c36
+c37+c38+c39+c40+c41
+c42+c43+c44+c45+c46
+c47+c48+c49+c50;
aver=count/(512*512);
save resfscla.mat aver
```

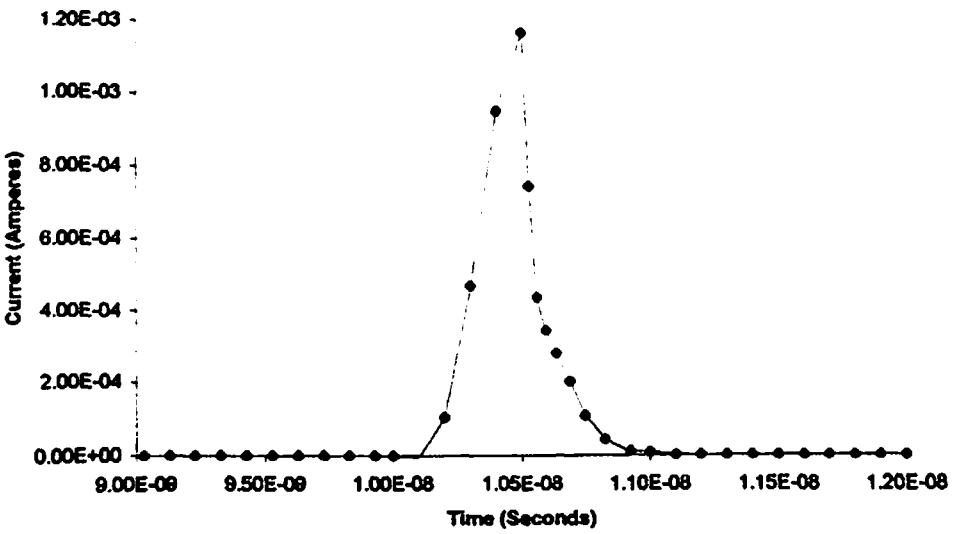
APPENDIX A9

CMOS inverter with full high-level logic at the input

Variation of output voltage with time for (*w/l*) =5

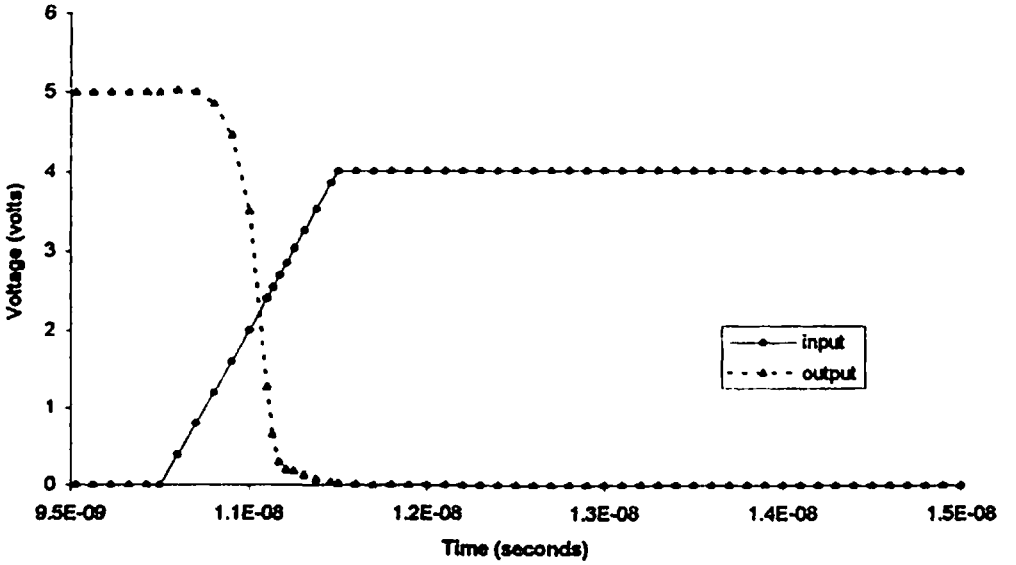


Variation of static current through inverter with time

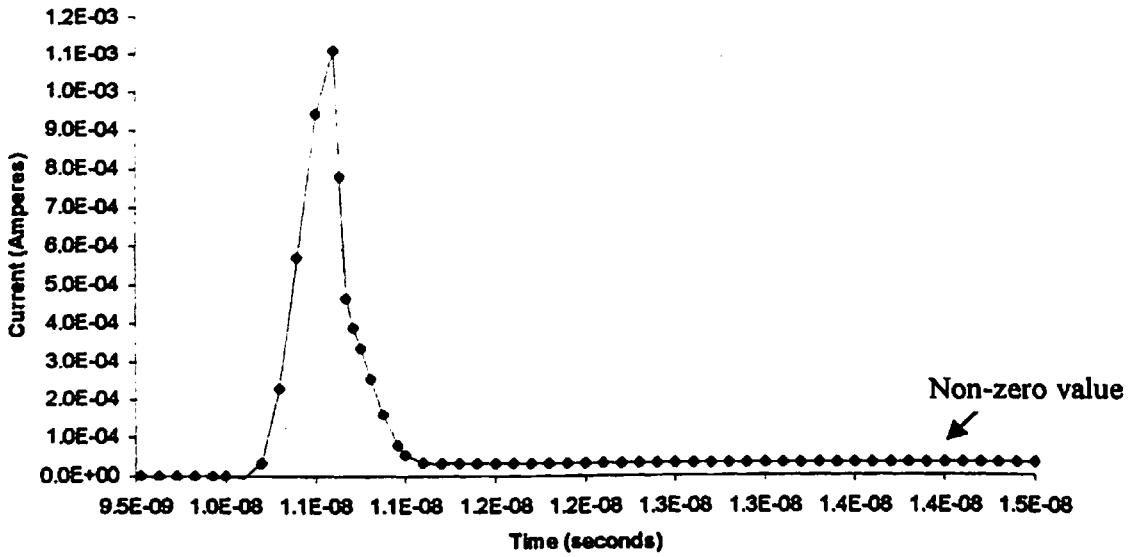


CMOS inverter with degraded high-level logic at the input

Variation of output voltage with time for *(w/l)* =5



Variation of static current through inverter with time



APPENDIX B1

Nominal Level 2 MOSFET Parameter of MCNC 1.2um CMOS Process

.model NMOS nmos

```
+  Level=2              Ld=0.0u          Tox=225.00E-10
+  Nsub=1.066E+16       Vto=0.622490     Uo=1215.74
+  Uexp=4.612355E-2     Ucrit=174667     Delta=0.0
+  Vmax=177269          Xj=0.9u
+  Nfs=4.55168E+12      Neff=4.68830     Nss=3.00E+10
+  Tpg=1.000            Rsh=60           Cgso=2.89E-10
+  Cgdo=2.89E-10        Cj=3.27E-04      Mj=1.067
+  Cjsw=1.74E-10        Mjsw=0.195
```

.model PMOS pmos

```
+  Level=2              Ld=0.03000u      Tox=225.000E-10
+  Nsub=6.575441E+16    Vto=-0.63025     Uo=361.941
+  Uexp=8.886957E-02    Ucrit=637449     Delta=0.0
+  Vmax=63253.3         Xj=0.112799u
+  Nfs=1.668437E+11     Neff=0.64354     Nss=3.00E+10
+  Tpg=-1.00            Rsh=150          Cgso=3.35E-10
+  Cgdo=3.35E-10        Cj=4.75E-04      Mj=0.341
+  Cjsw=2.23E-10        Mjsw=0.307
```

APPENDIX B2

Results of worst-case propagation delay without taking circuit parasitics into account for fully static logic design style

Transistor size *(w/l)*	Operand Size	Worst-case propagation delay (ns)				
		Carry look-ahead	Condition-al sum	Carry select	Carry skip	Ripple carry
1.5	4	1.4	1.78	1.41	1.44	1.24
	8	1.97	2.64	1.93	2.67	2.61
	16	2.58	4.30	3.28	3.38	5.28
	32	3.19	7.65	5.9	4.93	10.65
	64	3.89	14.38	11.25	7.88	21.44
3	4	1.32	1.73	1.35	1.37	1.20
	8	1.89	2.58	1.87	2.58	2.51
	16	2.48	4.23	3.16	3.28	5.14
	32	3.15	7.53	5.74	4.73	10.41
	64	3.80	14.16	11.01	7.72	20.90
5	4	1.31	1.69	1.34	1.36	1.19
	8	1.86	2.55	1.84	2.52	2.47
	16	2.44	4.18	3.11	3.21	5.03
	32	3.09	7.49	5.71	4.66	10.30
	64	3.75	14.07	10.95	7.53	20.70

APPENDIX B3

Simulation results of different adders

Results of ripple carry adder

TABLE T1: WORST-CASE PROPAGATION-DELAY AND ENERGY-CONSUMPTION PER ADDITION OF RIPPLE CARRY ADDER FOR DIFFERENT OPERAND SIZES AND TRANSISTOR SIZES

FULLY STATIC CMOS LOGIC (l=1.2μm)

(w/l)	Performance metrics	4-bit	8-bit	16-bit	32-bit	64-bit
1.5	Worst-case propagation delay in (Seconds×10^{-9})	3.51	8.09	18.01	46.09	120.70
	Energy consumption per additon in (Joules ×10^{-10})	0.27	0.67	1.58	4.30	11.96
3	Worst-case propagation delay in (Seconds×10^{-9})	2.68	6.07	13.30	31.01	75.69
	Energy consumption per additon in (Joules ×10^{-10})	0.41	0.98	2.26	5.51	13.95
5	Worst-case propagation delay in (Seconds×10^{-9})	2.39	5.30	11.87	25.52	60.33
	Energy consumption per additon in (Joules ×10^{-10})	0.59	1.41	3.23	7.16	18.34

TABLE T2: WORST-CASE PROPAGATION DELAY AND ENERGY CONSUMPTION OF RIPPLE CARRY ADDER FOR DIFFERENT OPERAND SIZES AND TRANSISTOR SIZES

DOMINO CMOS LOGIC (l=1.2μm)

(w/l)	Performance metrics	4-bit	8-bit	16-bit	32-bit	64-bit
1.5	Worst-case propagation delay in (Seconds×10^{-9})	6.29	14.6	31.03	73.36	183.38
	Energy consumption per additon in (Joules ×10^{-10})	1.04	2.22	4.78	10.98	26.82
3	Worst-case propagation delay in (Seconds×10^{-9})	4.58	10.44	22.6	50.86	116.82
	Energy consumption per additon in (Joules ×10^{-10})	1.59	3.42	7.26	15.99	36.33
5	Worst-case propagation delay in (Seconds×10^{-9})	4.24	8.99	18.88	41.20	91.66
	Energy consumption per additon in (Joules ×10^{-10})	2.41	4.93	10.28	22.14	49.03

TABLE T3: WORST-CASE PROPAGATION DELAY AND ENERGY CONSUMPTION OF RIPPLE CARRY ADDER FOR DIFFERENT OPERAND SIZES AND TRASISTOR SIZES

COMPLEMENTARY PASS TRANSISTOR LOGIC (l=1.2μm)

(w/l)	Performance metrics	4-bit	8-bit	16-bit	32-bit	64-bit
1.5	Worst-case propagation delay in (Seconds×10^{-9})	3.62	7.37	17.11	39.78	97.32
	Energy consumption per additon in (Joules ×10^{-10})	0.56	1.16	2.82	6.32	17.76
3	Worst-case propagation delay in (Seconds×10^{-9})	2.73	5.55	11.77	26.62	69.52
	Energy consumption per additon in (Joules ×10^{-10})	0.82	1.73	3.80	8.65	24.11
5	Worst-case propagation delay in (Seconds×10^{-9})	2.40	4.93	10.37	22.18	53.10
	Energy consumption per additon in (Joules ×10^{-10})	1.23	2.52	5.41	11.84	30.65

TABLE T4: WORST-CASE PROPAGATION DELAY AND ENERGY CONSUMPTION OF RIPPLE CARRY ADDER FOR DIFFERENT OPERAND SIZES AND TRASISTOR SIZES

DUAL PASS TRANSISTOR LOGIC (l=1.2μm)

(w/l)	Performance metrics	4-bit	8-bit	16-bit	32-bit	64-bit
1.5	Worst-case propagation delay in (Seconds×10^{-9})	5.43	12.42	27.34	60.34	142.18
	Energy consumption per additon in (Joules ×10^{-10})	0.63	1.43	3.29	7.36	17.56
3	Worst-case propagation delay in (Seconds×10^{-9})	4.16	9.07	19.05	42.12	96.05
	Energy consumption per additon in (Joules ×10^{-10})	0.95	2.14	4.54	10.20	24.10
5	Worst-case propagation delay in (Seconds×10^{-9})	3.64	7.93	16.51	35.57	78.68
	Energy consumption per additon in (Joules ×10^{-10})	1.36	3.04	6.50	14.23	32.22

Results of carry skip adder

TABLE T5: WORST-CASE PROPAGATION-DELAY AND ENERGY-CONSUMPTION OF CARRY SKIP ADDER FOR DIFFERENT OPERAND SIZES AND TRANSISTOR SIZES

FULLY STATIC CMOS LOGIC (l=1.2μm)

(w/l)	Performance metrics	4-bit	8-bit	16-bit	32-bit	64-bit
1.5	Worst-case propagation delay in (Seconds×10^{-9})	4.18	7.90	11.74	18.79	37.12
	Energy consumption per additon in (Joules ×10^{-10})	0.31	0.75	1.93	4.91	13.24
3	Worst-case propagation delay in (Seconds×10^{-9})	3.16	6.00	8.30	13.09	24.68
	Energy consumption per additon in (Joules ×10^{-10})	0.45	1.08	2.61	6.45	16.89
5	Worst-case propagation delay in (Seconds×10^{-9})	2.67	5.38	7.21	10.78	19.66
	Energy consumption per additon in (Joules ×10^{-10})	0.67	1.64	3.79	8.73	21.11

TABLE T6: WORST-CASE PROPAGATION DELAY AND ENERGY CONSUMPTION OF CARRY SKIP ADDER FOR DIFFERENT OPERAND SIZES AND TRANSISTOR SIZES

DOMINO CMOS LOGIC (l=1.2μm)

(w/l)	Performance metrics	4-bit	8-bit	16-bit	32-bit	64-bit
1.5	Worst-case propagation delay in (Seconds×10^{-9})	6.86	14.84	19.74	32.22	64.84
	Energy consumption per additon in (Joules ×10^{-10})	1.12	2.39	5.30	12.28	30.50
3	Worst-case propagation delay in (Seconds×10^{-9})	5.44	10.88	14.25	22.43	42.08
	Energy consumption per additon in (Joules ×10^{-10})	1.79	3.70	8.02	17.98	41.65
5	Worst-case propagation delay in (Seconds×10^{-9})	4.64	9.02	11.92	18.05	32.23
	Energy consumption per additon in (Joules ×10^{-10})	2.58	5.22	11.41	24.87	56.31

TABLE T7: WORST-CASE PROPAGATION DELAY AND ENERGY CONSUMPTION OF CARRY SKIP ADDER FOR DIFFERENT OPERAND SIZES AND TRASISTOR SIZES

COMPLEMENTARY PASS TRANSISTOR LOGIC (l=1.2μm)

(w/l)	Performance metrics	4-bit	8-bit	16-bit	32-bit	64-bit
1.5	Worst-case propagation delay in (Seconds×10^{-9})	4.90	9.08	15.73	26.27	57.12
	Energy consumption per additon in (Joules ×10^{-10})	0.65	1.65	4.05	9.06	25.60
3	Worst-case propagation delay in (Seconds×10^{-9})	3.33	6.54	10.16	17.61	33.54
	Energy consumption per additon in (Joules ×10^{-10})	0.90	2.41	5.62	12.59	33.05
5	Worst-case propagation delay in (Seconds×10^{-9})	3.05	5.77	8.57	14.8	28.9
	Energy consumption per additon in (Joules ×10^{-10})	1.38	3.58	7.94	18.04	47.12

TABLE T8: WORST-CASE PROPAGATION DELAY AND ENERGY CONSUMPTION OF CARRY SKIP ADDER FOR DIFFERENT OPERAND SIZES AND TRASISTOR SIZES

DUAL PASS TRANSISTOR LOGIC (l=1.2μm)

(w/l)	Performance metrics	4-bit	8-bit	16-bit	32-bit	64-bit
1.5	Worst-case propagation delay in (Seconds×10^{-9})	6.66	14.15	21.44	35.77	70.35
	Energy consumption per additon in (Joules ×10^{-10})	0.74	1.82	4.27	10.06	24.36
3	Worst-case propagation delay in (Seconds×10^{-9})	4.84	9.52	12.98	23.97	45.50
	Energy consumption per additon in (Joules ×10^{-10})	1.07	2.46	6.05	13.54	32.19
5	Worst-case propagation delay in (Seconds×10^{-9})	4.23	8.43	12.54	20.44	39.21
	Energy consumption per additon in (Joules ×10^{-10})	1.52	3.65	8.68	18.78	57.60

TABLE T7: WORST-CASE PROPAGATION DELAY AND ENERGY CONSUMPTION OF CARRY SKIP ADDER FOR DIFFERENT OPERAND SIZES AND TRASISTOR SIZES

COMPLEMENTARY PASS TRANSISTOR LOGIC (l=1.2μm)

(w/l)	Performance metrics	4-bit	8-bit	16-bit	32-bit	64-bit
1.5	Worst-case propagation delay in (Seconds×10^{-9})	4.90	9.08	15.73	26.27	57.12
	Energy consumption per additon in (Joules ×10^{-10})	0.65	1.65	4.05	9.06	25.60
3	Worst-case propagation delay in (Seconds×10^{-9})	3.33	6.54	10.16	17.61	33.54
	Energy consumption per additon in (Joules ×10^{-10})	0.90	2.41	5.62	12.59	33.05
5	Worst-case propagation delay in (Seconds×10^{-9})	3.05	5.77	8.57	14.8	28.9
	Energy consumption per additon in (Joules ×10^{-10})	1.38	3.58	7.94	18.04	47.12

TABLE T8: WORST-CASE PROPAGATION DELAY AND ENERGY CONSUMPTION OF CARRY SKIP ADDER FOR DIFFERENT OPERAND SIZES AND TRASISTOR SIZES

DUAL PASS TRANSISTOR LOGIC (l=1.2μm)

(w/l)	Performance metrics	4-bit	8-bit	16-bit	32-bit	64-bit
1.5	Worst-case propagation delay in (Seconds×10^{-9})	6.66	14.15	21.44	35.77	70.35
	Energy consumption per additon in (Joules ×10^{-10})	0.74	1.82	4.27	10.06	24.36
3	Worst-case propagation delay in (Seconds×10^{-9})	4.84	9.52	12.98	23.97	45.50
	Energy consumption per additon in (Joules ×10^{-10})	1.07	2.46	6.05	13.54	32.19
5	Worst-case propagation delay in (Seconds×10^{-9})	4.23	8.43	12.54	20.44	39.21
	Energy consumption per additon in (Joules ×10^{-10})	1.52	3.65	8.68	18.78	57.60

Results of carry select adder

TABLE T9: WORST-CASE PROPAGATION-DELAY AND ENERGY-CONSUMPTION OF CARRY SELECT ADDER FOR DIFFERENT OPERAND SIZES AND TRANSISTOR SIZES

FULLY STATIC CMOS LOGIC (l=1.2μm)

(w/l)	Performance metrics	4-bit	8-bit	16-bit	32-bit	64-bit
1.5	Worst-case propagation delay in (Seconds×10^{-9})	3.55	5.14	11.00	22.32	48.10
	Energy consumption per additon in (Joules ×10^{-10})	0.61	1.08	3.04	7.30	17.09
3	Worst-case propagation delay in (Seconds×10^{-9})	2.85	4.01	7.90	15.28	31.85
	Energy consumption per additon in (Joules ×10^{-10})	0.94	1.59	4.38	10.02	23.77
5	Worst-case propagation delay in (Seconds×10^{-9})	2.65	3.61	6.16	12.13	25.95
	Energy consumption per additon in (Joules ×10^{-10})	1.38	2.37	6.07	14.18	36.84

TABLE T10: WORST-CASE PROPAGATION DELAY AND ENERGY CONSUMPTION OF CARRY SELECT ADDER FOR DIFFERENT OPERAND SIZES AND TRANSISTOR SIZES

DOMINO CMOS LOGIC (l=1.2μm)

(w/l)	Performance metrics	4-bit	8-bit	16-bit	32-bit	64-bit
1.5	Worst-case propagation delay in (Seconds×10^{-9})	7.02	10.05	15.3	32.4	84.88
	Energy consumption per additon in (Joules ×10^{-10})	2.40	3.44	8.37	19.58	48.44
3	Worst-case propagation delay in (Seconds×10^{-9})	5.49	7.34	10.83	22.31	53.30
	Energy consumption per additon in (Joules ×10^{-10})	3.84	5.32	12.78	29.73	68.32
5	Worst-case propagation delay in (Seconds×10^{-9})	4.86	6.23	9.56	17.74	40.44
	Energy consumption per additon in (Joules ×10^{-10})	5.64	7.71	18.75	42.26	97.12

TABLE T11: WORST-CASE PROPAGATION DELAY AND ENERGY CONSUMPTION OF CARRY SELECT ADDER FOR DIFFERENT OPERAND SIZES AND TRASISTOR SIZES

COMPLEMENTARY PASS TRANSISTOR LOGIC (l=1.2μm)

(w/l)	Performance metrics	4-bit	8-bit	16-bit	32-bit	64-bit
1.5	Worst-case propagation delay in (Seconds×10^{-9})	4.26	5.60	10.30	22.94	57.11
	Energy consumption per additon in (Joules ×10^{-10})	1.19	1.59	3.81	10.39	24.96
3	Worst-case propagation delay in (Seconds×10^{-9})	3.15	4.14	6.67	17.04	32.54
	Energy consumption per additon in (Joules ×10^{-10})	1.80	2.26	5.35	14.41	31.77
5	Worst-case propagation delay in (Seconds×10^{-9})	2.86	3.50	5.78	12.73	26.63
	Energy consumption per additon in (Joules ×10^{-10})	2.71	3.57	7.78	19.54	40.42

TABLE T12: WORST-CASE PROPAGATION DELAY AND ENERGY CONSUMPTION OF CARRY SELECT ADDER FOR DIFFERENT OPERAND SIZES AND TRASISTOR SIZES

DUAL PASS TRANSISTOR LOGIC (l=1.2μm)

(w/l)	Performance metrics	4-bit	8-bit	16-bit	32-bit	64-bit
1.5	Worst-case propagation delay in (Seconds×10^{-9})	6.31	7.81	14.55	29.82	70.53
	Energy consumption per additon in (Joules ×10^{-10})	1.43	2.20	6.11	16.40	31.11
3	Worst-case propagation delay in (Seconds×10^{-9})	4.69	6.19	10.19	19.84	44.39
	Energy consumption per additon in (Joules ×10^{-10})	2.11	3.18	9.21	22.39	41.82
5	Worst-case propagation delay in (Seconds×10^{-9})	4.06	5.33	8.59	16.19	34.43
	Energy consumption per additon in (Joules ×10^{-10})	2.96	4.58	15.01	34.92	57.18

Results of conditional sum adder

TABLE T13: WORST-CASE PROPAGATION-DELAY AND ENERGY-CONSUMPTION OF CONDITIONAL SUM ADDER FOR DIFFERENT OPERAND SIZES AND TRANSISTOR SIZES

FULLY STATIC CMOS LOGIC (l=1.2μm)

(w/l)	Performance metrics	4-bit	8-bit	16-bit	32-bit	64-bit
1.5	Worst-case propagation delay in (Seconds×10^{-9})	4.62	6.26	11.67	27.00	61.70
	Energy consumption per additon in (Joules ×10^{-10})	0.90	1.81	3.76	7.90	22.14
3	Worst-case propagation delay in (Seconds×10^{-9})	3.62	4.95	8.51	18.24	38.98
	Energy consumption per additon in (Joules ×10^{-10})	1.35	2.71	5.60	11.31	30.49
5	Worst-case propagation delay in (Seconds×10^{-9})	3.17	4.39	7.35	14.3	32.3
	Energy consumption per additon in (Joules ×10^{-10})	1.98	4.04	8.28	16.76	41.80

TABLE T14: WORST-CASE PROPAGATION DELAY AND ENERGY CONSUMPTION OF CONDITIONAL SUM ADDER FOR DIFFERENT OPERAND SIZES AND TRANSISTOR SIZES

DOMINO CMOS LOGIC (l=1.2μm)

(w/l)	Performance metrics	4-bit	8-bit	16-bit	32-bit	64-bit
1.5	Worst-case propagation delay in (Seconds×10^{-9})	4.80	6.32	10.53	23.30	54.08
	Energy consumption per additon in (Joules ×10^{-10})	1.42	2.99	6.22	13.65	29.77
3	Worst-case propagation delay in (Seconds×10^{-9})	3.69	4.74	7.48	15.30	32.52
	Energy consumption per additon in (Joules ×10^{-10})	2.19	4.63	9.54	20.46	43.47
5	Worst-case propagation delay in (Seconds×10^{-9})	3.19	4.02	6.19	11.85	24.40
	Energy consumption per additon in (Joules ×10^{-10})	3.23	6.65	13.97	29.25	61.92

TABLE T15: WORST-CASE PROPAGATION DELAY AND ENERGY CONSUMPTION OF CONDITIONAL SUM ADDER FOR DIFFERENT OPERAND SIZES AND TRASISTOR SIZES

COMPLEMENTARY PASS TRANSISTOR LOGIC (λ=1.2μm)

(w/l)	Performance metrics	4-bit	8-bit	16-bit	32-bit	64-bit
1.5	Worst-case propagation delay in (Seconds×10^{-9})	3.50	4.36	7.43	17.15	41.31
	Energy consumption per additon in (Joules ×10^{-10})	0.80	1.65	3.29	7.55	19.39
3	Worst-case propagation delay in (Seconds×10^{-9})	2.22	3.24	5.09	10.66	24.59
	Energy consumption per additon in (Joules ×10^{-10})	1.16	2.33	4.53	10.00	24.58
5	Worst-case propagation delay in (Seconds×10^{-9})	1.98	2.78	4.42	9.40	18.8
	Energy consumption per additon in (Joules ×10^{-10})	1.69	3.36	6.47	13.99	33.09

TABLE T16: WORST-CASE PROPAGATION DELAY AND ENERGY CONSUMPTION OF CONDITIONAL SUM ADDER FOR DIFFERENT OPERAND SIZES AND TRASISTOR SIZES

DUAL PASS TRANSISTOR LOGIC (λ=1.2μm)

(w/l)	Performance metrics	4-bit	8-bit	16-bit	32-bit	64-bit
1.5	Worst-case propagation delay in (Seconds×10^{-9})	4.31	4.91	7.54	13.93	31.09
	Energy consumption per additon in (Joules ×10^{-10})	1.08	1.92	4.05	9.21	22.70
3	Worst-case propagation delay in (Seconds×10^{-9})	2.93	3.47	5.14	8.89	18.29
	Energy consumption per additon in (Joules ×10^{-10})	1.40	2.58	5.34	11.34	26.63
5	Worst-case propagation delay in (Seconds×10^{-9})	2.59	3.02	4.35	7.26	14.39
	Energy consumption per additon in (Joules ×10^{-10})	1.94	3.59	7.30	15.35	35.04

Results of carry look-ahead adder

TABLE T17: WORST-CASE PROPAGATION-DELAY AND ENERGY-CONSUMPTION OF CARRY LOOK-AHEAD ADDER FOR DIFFERENT OPERAND SIZES AND TRANSISTOR SIZES

FULLY STATIC CMOS LOGIC (l=1.2μm)

(w/l)	Performance metrics	4-bit	8-bit	16-bit	32-bit	64-bit
1.5	Worst-case propagation delay in (Seconds×10^{-9})	3.91	5.48	7.29	10.34	14.27
	Energy consumption per additon in (Joules ×10^{-10})	0.37	0.83	1.92	4.82	11.61
3	Worst-case propagation delay in (Seconds×10^{-9})	3.02	4.19	5.50	7.67	10.27
	Energy consumption per additon in (Joules ×10^{-10})	0.61	1.22	2.65	6.16	15.85
5	Worst-case propagation delay in (Seconds×10^{-9})	2.65	3.55	4.74	6.15	8.38
	Energy consumption per additon in (Joules ×10^{-10})	0.86	1.71	3.82	8.53	20.49

TABLE T18: WORST-CASE PROPAGATION DELAY AND ENERGY CONSUMPTION OF CARRY LOOK-AHEAD ADDER FOR DIFFERENT OPERAND SIZES AND TRANSISTOR SIZES

DOMINO CMOS LOGIC (l=1.2μm)

(w/l)	Performance metrics	4-bit	8-bit	16-bit	32-bit	64-bit
1.5	Worst-case propagation delay in (Seconds×10^{-9})	5.51	8.50	11.25	15.46	26.40
	Energy consumption per additon in (Joules ×10^{-10})	1.14	2.48	5.32	12.06	28.41
3	Worst-case propagation delay in (Seconds×10^{-9})	4.23	6.05	8.14	10.87	16.82
	Energy consumption per additon in (Joules ×10^{-10})	1.81	3.76	7.90	17.47	38.65
5	Worst-case propagation delay in (Seconds×10^{-9})	3.64	5.21	6.84	9.01	13.44
	Energy consumption per additon in (Joules ×10^{-10})	2.60	5.40	11.15	24.16	53.31

TABLE T19: WORST-CASE PROPAGATION DELAY AND ENERGY CONSUMPTION OF CARRY LOOK-AHEAD ADDER FOR DIFFERENT OPERAND SIZES AND TRASISTOR SIZES

COMPLEMENTARY PASS TRANSISTOR LOGIC (l=1.2μm)

(w/l)	Performance metrics	4-bit	8-bit	16-bit	32-bit	64-bit
1.5	Worst-case propagation delay in (Seconds×10^{-9})	4.56	6.46	11.22	14.78	22.85
	Energy consumption per additon in (Joules ×10^{-10})	0.78	2.92	6.85	15.49	39.59
3	Worst-case propagation delay in (Seconds×10^{-9})	3.37	4.99	7.05	10.81	14.94
	Energy consumption per additon in (Joules ×10^{-10})	1.18	4.31	9.73	21.87	51.73
5	Worst-case propagation delay in (Seconds×10^{-9})	2.92	4.32	5.87	8.30	11.09
	Energy consumption per additon in (Joules ×10^{-10})	1.71	6.24	13.70	31.48	70.65

TABLE T20: WORST-CASE PROPAGATION DELAY AND ENERGY CONSUMPTION OF CARRY LOOK - AHEAD ADDER FOR DIFFERENT OPERAND SIZES AND TRASISTOR SIZES

DUAL PASS TRANSISTOR LOGIC (l=1.2μm)

(w/l)	Performance metrics	4-bit	8-bit	16-bit	32-bit	64-bit
1.5	Worst-case propagation delay in (Seconds×10^{-9})	7.38	11.15	15.85	23.22	34.14
	Energy consumption per additon in (Joules ×10^{-10})	0.85	1.80	3.86	9.40	23.63
3	Worst-case propagation delay in (Seconds×10^{-9})	5.42	8.28	11.95	16.45	22.90
	Energy consumption per additon in (Joules ×10^{-10})	1.24	2.60	5.69	13.00	30.33
5	Worst-case propagation delay in (Seconds×10^{-9})	4.64	7.08	10.11	13.72	18.68
	Energy consumption per additon in (Joules ×10^{-10})	1.79	3.74	8.05	17.85	40.51

APPENDIX B4

Parasitic degradation of different adder

Results of ripple carry adder

TABLE T30: PARASTIC DEGRADATION FACTOR (α) AND GATE COUNT (n) OF RIPPLE CARRY ADDER FOR DIFFERENT OPERAND SIZES AND TRANSISTOR SIZES

FULLY STATIC CMOS LOGIC (l=1.2μm)

(w/l)	4-bit n=8	8-bit n=16	16-bit n=32	32-bit n=64	64-bit n=128
	α	α	α	α	α
1.5	1.661	1.919	2.145	2.742	3.581
3	1.554	1.743	1.983	2.258	2.719
5	1.600	1.587	1.935	2.160	2.535

TABLE T31: PARASTIC DEGRADATION FACTOR (α) AND GATE COUNT (n) OF RIPPLE CARRY ADDER FOR DIFFERENT OPERAND SIZES AND TRANSISTOR SIZES

DOMINO CMOS LOGIC (l=1.2μm)

(w/l)	4-bit n=16	8-bit n=32	16-bit n=64	32-bit n=128	64-bit n=256
	α	α	α	α	α
1.5	1.896	2.178	2.347	2.760	3.454
3	1.827	1.912	2.066	2.327	2.671
5	1.807	1.873	2.013	2.200	2.443

TABLE T32: PARASTIC DEGRADATION FACTOR (α) AND GATE COUNT (n) OF RIPPLE CARRY ADDER FOR DIFFERENT OPERAND SIZES AND TRANSISTOR SIZES

COMPLEMENTARY PASS TRANSISTOR LOGIC (l=1.2μm)

(w/l)	4-bit n=13	8-bit n=25	16-bit n=49	32-bit n=97	64-bit n=193
	α	α	α	α	α
1.5	1.228	1.303	1.548	1.815	2.229
3	1.150	1.223	1.276	1.476	1.926
5	1.144	1.202	1.252	1.431	1.722

TABLE T33: PARASTIC DEGRADATION FACTOR (α) AND GATE COUNT (n) OF RIPPLE CARRY ADDER FOR DIFFERENT OPERAND SIZES AND TRANSISTOR SIZES

DUAL PASS TRANSISTOR LOGIC (l=1.2μm)

(w/l)	4-bit n=16	8-bit n=32	16-bit n=64	32-bit n=128	64-bit n=256
	α	α	α	α	α
1.5	1.656	1.890	2.078	2.293	2.699
3	1.602	1.747	1.780	1.942	2.214
5	1.569	1.692	1.778	1.911	2.116

RESULTS OF CARRY SKIP ADDER

TABLE T34: PARASTIC DEGRADATION FACTOR (α) AND GATE COUNT (n) OF CARRY SKIP ADDER FOR DIFFERENT OPERAND SIZES AND TRANSISTOR SIZES

FULLY STATIC CMOS LOGIC (λ=1.2μm)

(w/l)	4-bit n=12	8-bit n=22	16-bit n=30	32-bit n=46	64-bit n=78
	α	α	α	α	α
1.5	1.248	1.280	1.419	1.459	1.699
3	1.125	1.182	1.203	1.236	1.370
5	1.121	1.158	1.136	1.184	1.274

TABLE T35: PARASTIC DEGRADATION FACTOR (α) AND GATE COUNT (n) OF CARRY SKIP ADDER FOR DIFFERENT OPERAND SIZES AND TRANSISTOR SIZES

DOMINO CMOS LOGIC (λ=1.2μm)

(w/l)	4-bit n=20	8-bit n=38	16-bit n=46	32-bit n=62	64-bit n=94
	α	α	α	α	α
1.5	1.663	1.890	2.079	2.513	3.345
3	1.593	1.686	1.836	2.136	2.634
5	1.552	1.631	1.779	2.002	2.357

TABLE T36: PARASTIC DEGRADATION FACTOR (α) AND GATE COUNT (n) OF CARRY SKIP ADDER FOR DIFFERENT OPERAND SIZES AND TRANSISTOR SIZES

COMPLEMENTARY PASS TRANSISTOR LOGIC (λ=1.2μm)

(w/l)	4-bit n=16	8-bit n=26	16-bit n=34	32-bit n=50	64-bit n=82
	α	α	α	α	α
1.5	1.524	1.730	2.299	2.611	3.566
3	1.335	1.528	1.802	2.132	2.479
5	1.292	1.564	1.778	2.091	2.482

TABLE T37: PARASTIC DEGRADATION FACTOR (α) AND GATE COUNT (n) OF CARRY SKIP ADDER FOR DIFFERENT OPERAND SIZES AND TRANSISTOR SIZES

DUAL PASS TRANSISTOR LOGIC (λ=1.2μm)

(w/l)	4-bit n=19	8-bit n=35	16-bit n=43	32-bit n=59	64-bit n=91
	α	α	α	α	α
1.5	1.656	1.901	2.351	2.849	3.645
3	1.478	1.627	1.811	2.322	2.856
5	1.485	1.580	1.909	2.308	2.846

RESULTS OF CARRY SELECT ADDER

TABLE T38: PARASTIC DEGRADATION FACTOR (α) AND GATE COUNT (n) OF CARRY SELECT ADDER FOR DIFFERENT OPERAND SIZES AND TRANSISTOR SIZES

FULLY STATIC CMOS LOGIC (λ=1.2μm)

(w/l)	4-bit n=10	8-bit n=12	16-bit n=20	32-bit n=36	64-bit n=68
	α	α	α	α	α
1.5	1.106	1.213	1.558	1.754	2.008
3	1.050	1.149	1.367	1.460	1.606
5	1.015	1.132	1.236	1.349	1.528

TABLE T39: PARASTIC DEGRADATION FACTOR (α) AND GATE COUNT (n) OF CARRY SELECT ADDER FOR DIFFERENT OPERAND SIZES AND TRANSISTOR SIZES

DOMINO CMOS LOGIC (λ=1.2μm)

(w/l)	4-bit n=20	8-bit n=20	16-bit n=28	32-bit n=44	64-bit n=76
	α	α	α	α	α
1.5	1.286	1.849	2.013	2.692	4.095
3	1.246	1.640	1.730	2.263	3.127
5	1.248	1.538	1.772	2.096	2.765

TABLE T40: PARASTIC DEGRADATION FACTOR (α) AND GATE COUNT (n) OF CARRY SELECT ADDER FOR DIFFERENT OPERAND SIZES AND TRANSISTOR SIZES

COMPLEMENTARY PASS TRANSISTOR LOGIC (λ=1.2μm)

(w/l)	4-bit n=17	8-bit n=17	16-bit n=25	32-bit n=41	64-bit n=73
	α	α	α	α	α
1.5	1.110	1.457	1.834	2.470	3.412
3	1.023	1.308	1.431	2.233	2.647
5	0.989	1.282	1.447	1.949	2.146

TABLE T41: PARASTIC DEGRADATION FACTOR (α) AND GATE COUNT (n) OF CARRY SELECT ADDER FOR DIFFERENT OPERAND SIZES AND TRANSISTOR SIZES

DUAL PASS TRANSISTORLOGIC (λ=1.2μm)

(w/l)	4-bit n=20	8-bit n=20	16-bit n=28	32-bit n=44	64-bit n=76
	α	α	α	α	α
1.5	1.412	1.749	2.328	3.029	4.151
3	1.277	1.681	1.981	2.453	3.176
5	1.285	1.688	1.947	2.332	2.870

RESULTS OF CONDITIONAL SUM ADDER

TABLE T42: PARASTIC DEGRADATION FACTOR (α) AND GATE COUNT (n) OF CONDITIONAL SUM ADDER FOR DIFFERENT OPERAND SIZES AND TRANSISTOR SIZES

FULLY STATIC CMOS LOGIC (λ=1.2μm)

(w/l)	4-bit n=10	8-bit n=12	16-bit n=16	32-bit n=24	64-bit n=40
	α	α	α	α	α
1.5	1.334	1.511	2.107	3.335	4.459
3	1.264	1.407	1.870	2.665	3.401
5	1.132	1.358	1.873	2.424	3.305

TABLE T43: PARASTIC DEGRADATION FACTOR (α) AND GATE COUNT (n) OF CONDITIONAL SUM ADDER FOR DIFFERENT OPERAND SIZES AND TRANSISTOR SIZES

DOMINO CMOS LOGIC (λ=1.2μm)

(w/l)	4-bit n=10	8-bit n=12	16-bit n=16	32-bit n=24	64-bit n=40
	α	α	α	α	α
1.5	2.041	2.247	2.805	4.114	5.749
3	1.939	2.086	2.422	3.294	4.176
5	1.883	2.028	2.336	2.983	3.660

TABLE T44: PARASTIC DEGRADATION FACTOR (α) AND GATE COUNT (n) OF CONDITIONAL SUM ADDER FOR DIFFERENT OPERAND SIZES AND TRANSISTOR SIZES

COMPLEMENTARY PASS TRANSISTOR LOGIC (λ=1.2μm)

(w/l)	4-bit n=11	8-bit n=14	16-bit n=20	32-bit n=32	64-bit n=55
	α	α	α	α	α
1.5	1.202	1.179	1.407	2.028	2.872
3	0.934	1.072	1.164	1.565	2.034
5	0.926	1.063	1.184	1.540	1.812

TABLE T45: PARASTIC DEGRADATION FACTOR (α) AND GATE COUNT (n) OF CONDITIONAL SUM ADDER FOR DIFFERENT OPERAND SIZES AND TRANSISTOR SIZES

DUAL PASS TRANSISTOR LOGIC (λ=1.2μm)

(w/l)	4-bit n=11	8-bit n=14	16-bit n=20	32-bit n=32	64-bit n=55
	α	α	α	α	α
1.5	0.888	0.795	0.949	1.217	1.675
3	0.740	0.682	0.788	0.941	1.202
5	0.735	0.668	0.760	0.895	1.098

RESULTS OF CARRY LOOK-AHEAD ADDER

TABLE T46: PARASTIC DEGRADATION FACTOR (α) AND GATE COUNT (n) OF CARRY-LOOK AHEAD ADDER FOR DIFFERENT OPERAND SIZES AND TRANSISTOR SIZES

FULLY STATIC CMOS LOGIC (l=1.2μm)

(w/l)	4-bit n=12	8-bit n=16	16-bit n=20	32-bit n=24	64-bit n=28
	α	α	α	α	α
1.5	0.823	0.866	0.921	1.087	1.287
3	0.772	0.805	0.847	0.979	1.127
5	0.762	0.792	0.848	0.919	1.070

TABLE T47: PARASTIC DEGRADATION FACTOR (α) AND GATE COUNT (n) OF CARRY-LOOK AHEAD ADDER FOR DIFFERENT OPERAND SIZES AND TRANSISTOR SIZES

DOMINO CMOS LOGIC (l=1.2μm)

(w/l)	4-bit n=10	8-bit n=14	16-bit n=18	32-bit n=22	64-bit n=26
	α	α	α	α	α
1.5	1.965	2.164	2.225	2.503	3.615
3	1.835	1.872	1.959	2.143	2.797
5	1.839	1.879	1.929	2.074	2.620

TABLE T48: PARASTIC DEGRADATION FACTOR (α) AND GATE COUNT (n) CARRY-LOOK AHEAD ADDER FOR DIFFERENT OPERAND SIZES AND TRANSISTOR SIZES

COMPLEMENTARY PASSTRANSISTOR LOGIC (l=1.2μm)

(w/l)	4-bit n=12	8-bit n=16	16-bit n=20	32-bit n=24	64-bit n=28
	α	α	α	α	α
1.5	1.375	1.463	2.034	2.226	2.952
3	1.262	1.373	1.558	1.974	2.351
5	1.248	1.386	1.498	1.774	2.030

TABLE T49: PARASTIC DEGRADATION FACTOR (α) AND GATE COUNT (n) CARRY LOOK-AHEAD ADDER FOR DIFFERENT OPERAND SIZES AND TRANSISTOR SIZES

DUAL PASS TRANSISTOR LOGIC (l=1.2μm)

(w/l)	4-bit n=12	8-bit n=16	16-bit n=20	32-bit n=24	64-bit n=28
	α	α	α	α	α
1.5	1.860	2.108	2.394	2.935	3.688
3	1.659	1.897	2.195	2.530	2.999
5	1.657	1.900	2.164	2.446	2.857

APPENDIX B5

Energy-delay product (EDP) of different adders

Energy-delay product of Ripple carry adder

TABLE T50: ENERGY DELAY PRODUCT OF RIPPLE CARRY ADDER FOR DIFFERENT LOGIC DESIGN STYLES AND TRANSISTOR SIZES

(l=1.2μm)

Logic design style	*(w/l)*	EDP in (Joules-sec $\times 10^{-19}$)				
		4-bit	8-bit	16-bit	32-bit	64-bit
Fully static CMOS	1.5	0.970	5.488	28.506	198.233	1444.764
	3	1.093	6.001	30.173	170.870	1055.965
	5	1.423	7.466	38.429	182.964	1106.776
Domino CMOS	1.5	6.552	32.536	148.463	805.576	4919.352
	3	7.319	35.727	164.077	813.338	4244.932
	5	10.204	44.386	194.242	912.382	4494.149
CPL	1.5	2.039	8.586	48.283	251.517	1728.942
	3	2.254	9.640	44.777	230.417	1676.437
	5	2.977	12.423	56.155	262.693	1627.743
DPL	1.5	3.435	17.766	90.018	443.861	2496.525
	3	3.935	19.392	86.436	429.607	2314.917
	5	4.960	24.096	107.302	505.997	2535.070

Energy-delay product of Carry skip adder

TABLE T51: ENERGY DELAY PRODUCT OF CARRY SKIP ADDER FOR DIFFERENT LOGIC DESIGN STYLES AND TRANSISTOR SIZES

(l=1.2μm)

Logic design style	*(w/l)*	EDP in (Joules-sec $\times 10^{-19}$)				
		4-bit	8-bit	16-bit	32-bit	64-bit
Fully static CMOS	1.5	1.326	5.919	22.722	92.345	491.752
	3	1.428	6.487	21.723	84.529	417.009
	5	1.813	8.834	27.390	94.192	415.120
Domino CMOS	1.5	7.692	35.508	104.705	395.979	1978.020
	3	9.772	40.308	114.396	403.410	1752.878
	5	11.987	47.135	136.135	449.169	1815.228
CPL	1.5	3.190	15.039	63.808	237.984	1503.516
	3	3.006	15.739	57.100	221.783	1108.498
	5	4.213	20.670	68.074	267.097	1361.797
DPL	1.5	4.911	25.809	91.461	359.840	1713.574
	3	5.168	23.441	78.485	324.627	1464.615
	5	6.422	30.785	108.808	383.792	2258.606

Energy-delay product of Carry select adder

TABLE T52: ENERGY DELAY PRODUCT OF CARRY SELECT ADDER FOR DIFFERENT LOGIC DESIGN STYLES AND TRANSISTOR SIZES

(l=1.2μm)

Logic design style	*(w/l)*	EDP in (Joules-sec $\times 10^{-19}$)				
		4-bit	8-bit	16-bit	32-bit	64-bit
Fully static CMOS	1.5	2.178	5.583	33.541	162.953	822.065
	3	2.691	6.397	34.602	153.125	757.286
	5	3.671	8.577	37.441	172.089	956.188
Domino CMOS	1.5	16.901	34.587	128.084	634.556	4111.623
	3	21.132	39.097	138.423	663.347	3641.943
	5	27.420	48.060	179.252	749.869	3927.642
CPL	1.5	5.110	8.951	39.255	238.516	1425.785
	3	5.685	9.372	35.688	245.644	1033.827
	5	7.752	12.504	45.007	248.834	1076.593
DPL	1.5	9.004	17.209	88.831	489.048	2194.421
	3	9.903	19.686	93.863	444.134	1284.121
	5	12.012	24.431	128.897	565.323	1968.777

Energy-delay product of Conditional sum adder

TABLE T53: ENERGY DELAY PRODUCT OF CONDITIONAL SUM ADDER FOR DIFFERENT LOGIC DESIGN STYLES AND TRANSISTOR SIZES

(l=1.2μm)

Logic design style	*(w/l)*	EDP in (Joules-sec $\times 10^{-19}$)				
		4-bit	8-bit	16-bit	32-bit	64-bit
Fully static CMOS	1.5	4.253	11.834	49.277	269.644	1366.246
	3	5.076	14.294	56.758	267.806	1188.771
	5	6.359	18.909	64.403	275.764	1350.191
Domino CMOS	1.5	6.830	18.916	65.482	318.223	1610.230
	3	8.126	21.952	71.408	313.168	1413.880
	5	10.310	26.791	86.514	346.705	1510.901
CPL	1.5	2.811	7.195	24.463	129.518	801.102
	3	2.589	7.568	23.101	106.601	604.505
	5	3.368	9.379	28.607	131.524	622.131
DPL	1.5	4.664	9.416	30.520	128.307	705.598
	3	4.098	8.953	27.440	100.823	487.002
	5	5.033	10.853	31.744	111.429	504.290

Energy-delay product of Carry look-ahead adder

TABLE T54: ENERGY DELAY PRODUCT OF CARRY LOOK-AHEAD ADDER FOR DIFFERENT LOGIC DESIGN STYLES AND TRANSISTOR SIZES

(l=1.2μm)

Logic design style	*(w/l)*	EDP in (Joules-sec $\times 10^{-19}$)				
		4-bit	8-bit	16-bit	32-bit	64-bit
Fully static CMOS	1.5	1.471	4.569	14.033	49.859	165.743
	3	1.835	5.130	14.621	47.279	162.872
	5	2.302	6.056	18.123	52.514	171.736
Domino CMOS	1.5	6.315	21.145	59.910	186.480	750.227
	3	7.664	22.777	64.327	189.974	650.209
	5	9.497	28.137	76.366	217.684	716.533
CPL	1.5	3.563	18.888	76.960	229.037	904.637
	3	3.983	21.494	68.632	236.469	772.992
	5	4.994	26.998	80.443	261.355	783.581
DPL ·	1.5	6.299	20.081	61.133	218.347	806.610
	3	6.730	21.562	68.045	213.896	694.610
	5	8.286	26.468	81.389	244.872	756.764

Core area of different 64-bit adders (standard-cell based designs)

TABLE T55: CORE AREA OF DIFFERENT 64-BIT ADDERS FOR DIFFERENT LOGIC DESIGN STYLES WITH $(w/l) = 1.5$ (t=1.2μm)

Logic design style	Core Area in $(\mu m)^2$				
	Ripple carry adder	Carry skip adder	Carry select adder	Conditional sum adder	Carry look ahead adder
Fully static CMOS	6,022,972.08	6,792,722.64	9,399,124.80	10,959,092.64	7,276,699.80
Domino CMOS	9,355,792.32	10,364,664.96	18,029,017.44	13,215,294.00	10,930,532.40
CPL	6,839,835.84	8,356,225.68	11,920,346.28	9,935,391.96	10,025,341.92
DPL	7,699,924.80	10,026,925.20	16,032,816.36	12,388,320.00	12,949,881.12

Transistor count of different adders

TABLE T56: TRANSISTOR COUNT OF DIFFERENT ADDERS DESIGNED IN DIFFERENT LOGIC DESIGN STYLES

Logic design style	Operand size	Adder				
		Ripple carry	Carry skip	Carry select	Conditional sum	Carry look-ahead
Fully static CMOS	4	144	172	356	382	198
	8	288	316	500	764	406
	16	576	660	1212	1528	822
	32	1152	1348	2636	3056	1654
	64	2304	2724	5484	6112	3318
Domino CMOS	4	224	254	530	416	258
	8	448	478	754	832	518
	16	896	986	1814	1664	1038
	32	1792	2002	3934	3328	2078
	64	3584	4034	8174	6656	4158
CPL	4	120	156	280	170	204
	8	240	320	400	340	420
	16	480	640	960	680	852
	32	960	1280	2080	1360	1716
	64	1920	2560	4320	2720	3444
DPL	4	216	274	488	306	324
	8	432	548	704	612	668
	16	864	1096	1680	1224	1356
	32	1728	2192	3632	2448	2732
	64	3456	4384	7536	4896	5484

APPENDIX B7

Energy consumption per addition of different 4-bit adders for random inputs

TABLE T57: AVERAGE ENERGY CONSUMPTION PER ADDITION OF DIFFERENT 4-BIT ADDERS WITH RANDOMLY APPLIED TEN SETS OF INPUTS

$(w/l) = 1.5$, (l=1.2μm)

4-bit Adder	Energy Consumption per addition (Joules × 10^{-10})							
	Fully static	M	Domino CMOS	M	CPL	M	DPL	M
Ripple carry	0.343	1.24	0.890	0.85	0.514	0.91	0.686	0.73
Carry skip	0.383	1.21	0.943	0.84	0.626	0.96	0.855	0.80
Carry select	0.853	1.39	2.115	0.88	1.173	0.98	1.673	0.79
Conditional select	0.955	1.04	1.255	0.88	0.878	1.09	1.224	0.88
Carry look-ahead	0.459	1.22	0.978	0.85	0.696	0.89	1.024	0.82

* M: $\frac{E_{random}}{E_{worst-case}}$

TABLE T58: AVERAGE ENERGY CONSUMPTION PER ADDITION OF DIFFERENT 4-BIT ADDERS WITH RANDOMLY APPLIED TEN SETS OF INPUTS

$(w/l) = 3$, (l=1.2μm)

4-bit Adder	Energy Consumption per addition (Joules× 10^{-10})							
	Fully static	M	Domino CMOS	M	CPL	M	DPL	M
Ripple carry	0.549	1.35	1.342	0.84	0.794	0.96	1.060	1.12
Carry skip	0.587	1.30	1.518	0.85	0.902	1.00	1.272	1.19
Carry select	1.371	1.45	3.408	0.89	1.795	0.99	2.484	1.18
Conditional select	1.532	1.09	1.972	0.90	1.331	1.14	1.720	1.23
Carry look-ahead	0.734	1.21	1.566	0.86	1.041	0.88	1.534	1.24

TABLE T59: AVERAGE ENERGY CONSUMPTION PER ADDITION OF DIFFERENT 4-BIT ADDERS WITH RANDOMLY APPLIED TEN SETS OF INPUTS

$(w/l) = 5$, (l=1.2μm)

4-bit Adder	Energy Consumption per addition (Joules× 10^{-10})							
	Fully static		Domino CMOS		CPL		DPL	
		M		M		M		M
Ripple carry	0.819	1.38	2.095	0.87	1.192	0.96	1.544	1.13
Carry skip	0.907	1.34	2.237	0.87	1.400	1.01	1.900	1.25
Carry select	2.067	1.49	5.086	0.90	2.706	1.00	3.454	1.17
Conditional select	2.262	1.14	2.997	0.93	1.986	1.17	2.502	1.29
Carry look-ahead	1.076	1.24	2.335	0.90	1.536	0.90	2.225	1.25

Energy-Delay Product for full custom adder designs

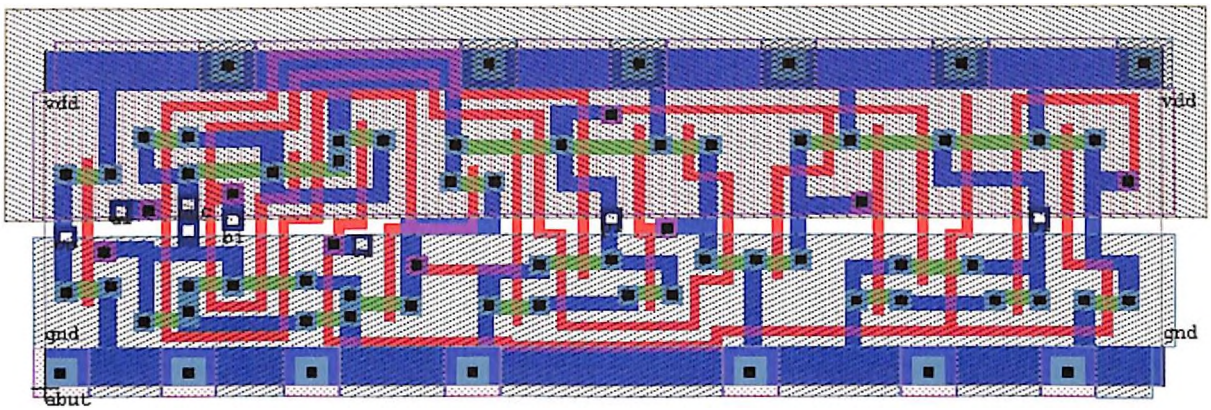
TABLE T60: WORST-CASE PROPAGATION DELAY AND ENERGY CONSUMPTION PER ADDITION OF FULL CUSTOM 64-BIT RIPPLE CARRY (RCA) AND CARRY LOOK-AHEAD (CLA) ADDER DESIGNS

FULLY STATIC CMOS LOGIC (l=1.2μm)

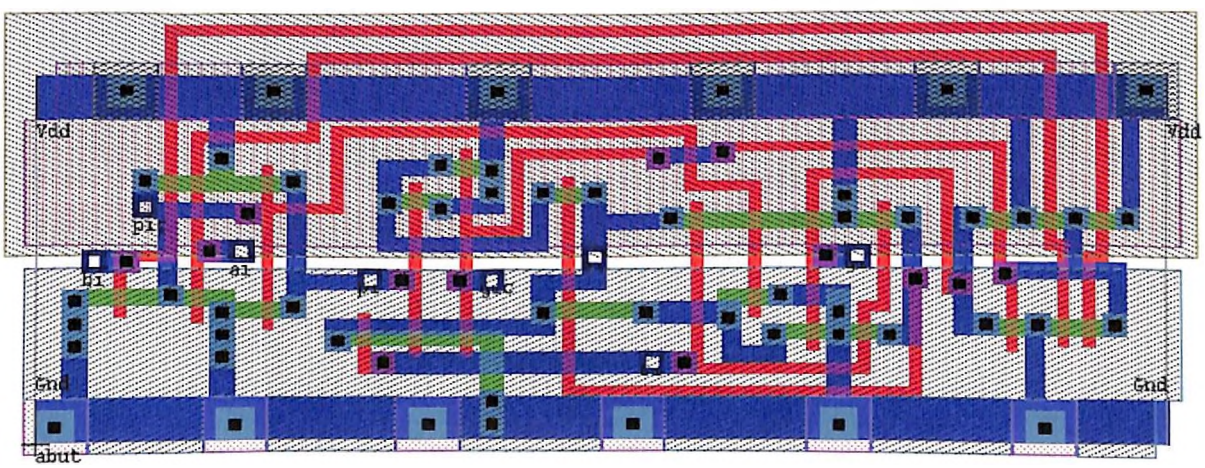
(w/l)	Performance metrics	64-bit Adder		EDP (J-Sec×10^{-19})	
		RCA	CLA	RCA	CLA
1.5	Worst-case propagation delay (Seconds×10^{-9})	71.4	9.70	397.768	86.968
	Energy consumption per additon (Joules ×10^{-10})	5.57	8.97		
3	Worst-case propagation delay (Seconds×10^{-9})	49.74	7.55	394.494	86.558
	Energy consumption per additon (Joules ×10^{-10})	7.93	11.46		
5	Worst-case propagation delay in (Seconds×10^{-9})	41.26	6.22	569.299	94.919
	Energy consumption per additon (Joules ×10^{-10})	13.80	15.26		

FULL CUSTOM LAYOUT

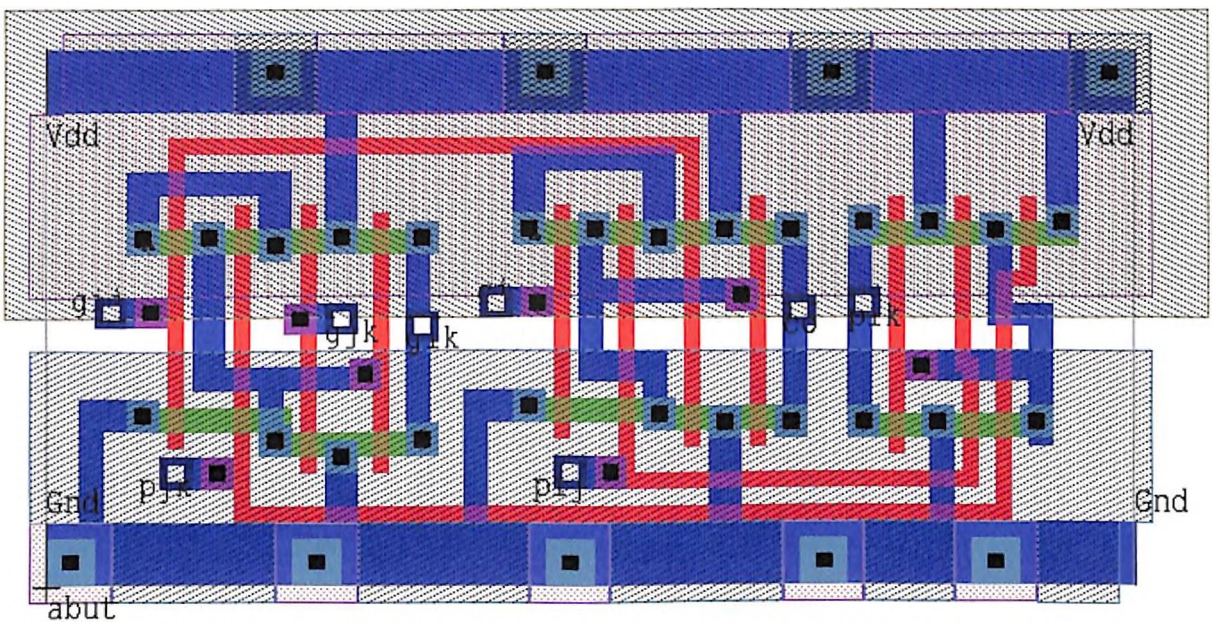
STANDARD CELL FULLADDER



STANDARD CELL A-CELL



STANDARD CELL B-CELL



LIST OF PUBLICATION

1) Gupta, A., and Shekhra, C., "Optimal Adder Architectures for Fully Static and Complementary Pass Logic Designs," Proc. National Seminar on VLSI: Systems, Designs and Technology, pp. 140-145, IIT, Bombay, Dec. 2000.