AN ABSTRACT OF THE THESIS OF

Parag Shantu Shah for the degree of Master of Science in Electrical and Computer Engineering presented on July 8, 1998. Title: Low-Power High-Performance 32-Bit 0.5μm CMOS Adder.

Redacted for Privacy

Abstract approved: _____

Shih-Lien Lu

Currently, the two most critical factors of microprocessor design are performance and power. The optimum balance of these two factors is reflected in the speed-power product (SPP). 32-bit CMOS adders are used as representative circuits to investigate a method of reducing the SPP. The purpose of this thesis is to show that sizing gates according to fan-out and removing buffer drivers can reduce the SPP. This thesis presents a method for sizing gates in large fan-out parallel prefix circuits to reduce the SPP and compares it to other methods. Three different parallel prefix adders are used to compare propagation delay and SPP. The first adder uses the depth-optimal prefix circuit. The second adder is based on Wei, Thompson, and Chen's time-optimal adder. The third adder uses a recursive doubling formation where all cells have minimum transistor width dimensions. The component cells in the adders are static CMOS as described by Brent and Kung. For all circuits, the smallest propagation delay occurs when the highest voltage supply is applied. The smallest SPP occurs when the lowest voltage supply is applied, but with the lowest performance. The Recursive Doubling Adder always has the lowest propagation delay for a particular set of parameters. However, its SPP is nearly equal to the Brent-Kung Adders and lower than Wei's Adder. The power-frequency analysis reveals that a decrease in V_t causes higher power consumption due to leakage.

Low-Power High-Performance 32-Bit 0.5μm CMOS Adder

by

Parag Shantu Shah

A THESIS

submitted to

Oregon State University

in partial fulfillment of
the requirements for the
degree of

Master of Science

Presented July 8, 1998
Commencement June 1999

Master of Science thesis of Parag Shantu Shah presented on July 8, 1998

APPROVED:

Redacted for Privacy

Major Professor, representing Electrical and Computer Engineering

Redacted for Privacy

Head of Department of Electrical and Computer Engineering

Redacted for Privacy

Dean of Graduate School

I understand that my thesis will become part of the permanent collection of Oregon State University libraries. My signature below authorizes release of my thesis to any reader upon request.

Redacted for Privacy

Parag Shantu Shah, Author

ACKNOWLEDGEMENT

This thesis acknowledges the following people for their support and assistance:

Shih-Lien Lu

Çetin Kaya Koç

Roger Traylor

John F. Wager

David J. Griffiths

Isadore Netto

Vaishali Phatak

Deepika Sharma

Ravi Puri

Kysse Siala

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

LIST OF FIGURES

LIST OF TABLES

DEDICATION

This thesis is dedicated to my loving Father, Mother, Bhai, and Bhabi.

Low-Power High-Performance 32-Bit 0.5μm CMOS Adder

1. INTRODUCTION

In microprocessors, the adder is one of the most frequently used hardware structures for arithmetic operations. The adder is used extensively by multiplication and division algorithms. Besides arithmetic instructions, it is also used for address calculations. If the adder is not designed correctly, it can be a major bottleneck to a microprocessor. A frequently used, improperly optimized, adder will consume high amounts of power in comparison with other logic.

1.1 Problem Definition

There are many different ways of designing an adder both at the gate (or logic) level and the transistor (or circuit) level. At the gate level, different implementations for an adder include the carry-look-ahead adder, conditional sum adder, carry-skip adder, and the carry-save adder [4]. Some implementations at the circuit level include the static CMOS full adder, mirror adder, dynamic adder, and Manchester carry-chain adder [5].

Previously, the speed or area of an adder was the primary concern, with little attention being paid to power consumption. To create a high performance adder, the speed-power product (SPP) must be minimized, while reducing area. The SPP is equal to

the product of the adder's propagation delay and the average power consumption. This thesis presents a method to produce a static CMOS circuit that has an optimal SPP.

1.2 Statement of Purpose

The purpose of this thesis is to show that by sizing gates according to fan-out and removing buffer drivers then the SPP can be reduced. Adder circuits are used to test the transistor sizing method. The adder scheme used is the carry look-ahead adder. The main research began from Wei, Thompson, and Chen's "Time-Optimal Design of a CMOS Adder." [3] Their paper focused only on finding the fastest adder. This thesis examines different designs, and finds the SPP-optimal design. This thesis presents circuit simulations done in HSPICE. The adders were stimulated with several hundred random numbers. A method of sizing the transistors similar to [3] was used. This thesis intends to show that this new method is more energy efficient.

2. REVIEW OF LITERATURE

Before looking at the simulations and results, base concepts related to addition in computer arithmetic must be reviewed. To meet today's high performance requirements, the primary adder used for many microprocessors is the carry look-ahead adder (CLA). In an n-bit adder, the sum value for the i^{th} bit is a function of the i^{th} bits of the two operands and the carry out from the previous bit position. In a CLA, the core of the logic calculates the carry values for each bit position in a parallel fashion. The parallel prefix computation uses an associative operation to calculate the carry values. We begin with a discussion of parallel prefix computation. Following the description of parallel prefix computation is the description of associative operation used in adders. The chapter concludes with a brief discussion of power consumption in CMOS digital design.

2.1 Parallel Prefix Computation

The prefix problem occurs when given a particular associative operation ($x_1 \bullet x_2 \bullet x_3 \bullet \ldots \bullet x_k$) the product of k different numbers must be calculated. Ladner and Fisher presented a solution to the prefix problem in [1]. A recursive solution can solve the problem. The recursive solution from [1] allows multiple solutions that have different depths and sizes.

The parallel prefix computation circuit is represented by a directed acyclic graph where each node is a separate operation. An example of a graph where the depth is three

and the size is four is shown in Figure 2.1. It computes the end product $x_1 \bullet x_2 \bullet x_2 \bullet x_3 \bullet x_4$ along with some other sub-products such as $x_1 \bullet x_2$ and $x_2 \bullet x_3$.

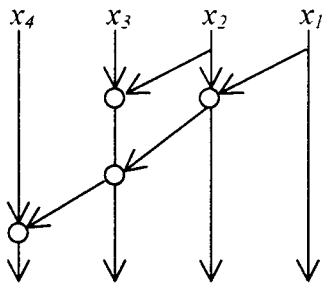


Figure 2.1 Sample Product Circuit

The depth is the number of nodes on the graph in the longest path (from top to bottom). Depth affects the propagation delay of the circuit. The size is the total number of nodes in the graph. Size determines the area of the circuit.

$P_k(n)$ denotes the family of circuits that solves the prefix problem for n inputs. The parameter k ranges from 0 to $\lceil \log_2 n \rceil$. The depth is less than $k - \lceil \log_2 n \rceil$ and the size is bounded below $2(1 - 2^{-k})n - 4$. For k=0, the reverse construction of $P_0(n)$ is shown in Figure 2.2. For k>0, the reverse construction of $P_k(n)$ is shown in Figure 2.3. For the recursive solutions shown, exact solutions exist when n is a power of two.

$$S_0(n) = 4n - F(5 + \log_2 n) + 1,$$

$$S_1(n) = 3n - F(4 + \log_2 n), \text{ and}$$

$$S_k = 2(1 + 2^{-k})n - F(5 + \log_2 n - k) + 1 - k.$$

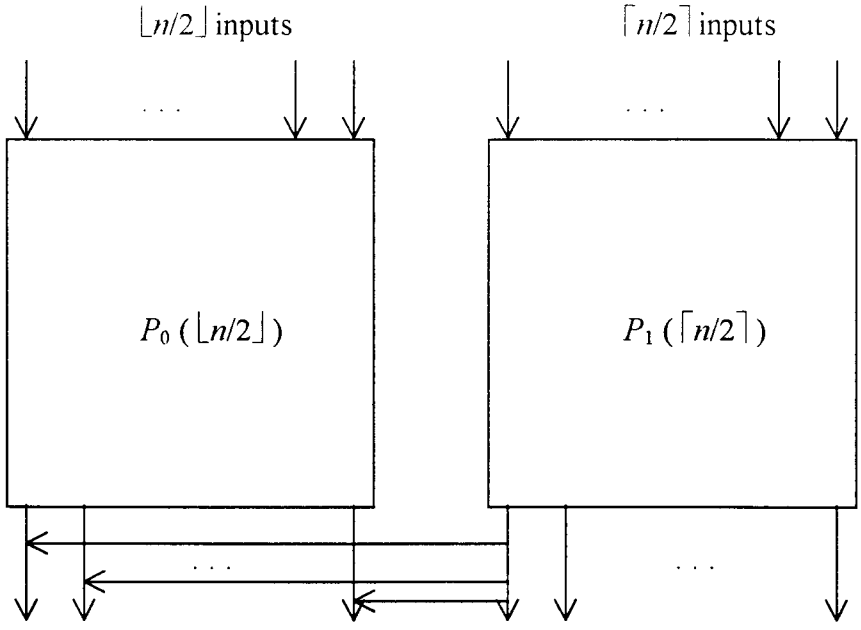


Figure 2.2 Recursive Construction of $P_0(n)$

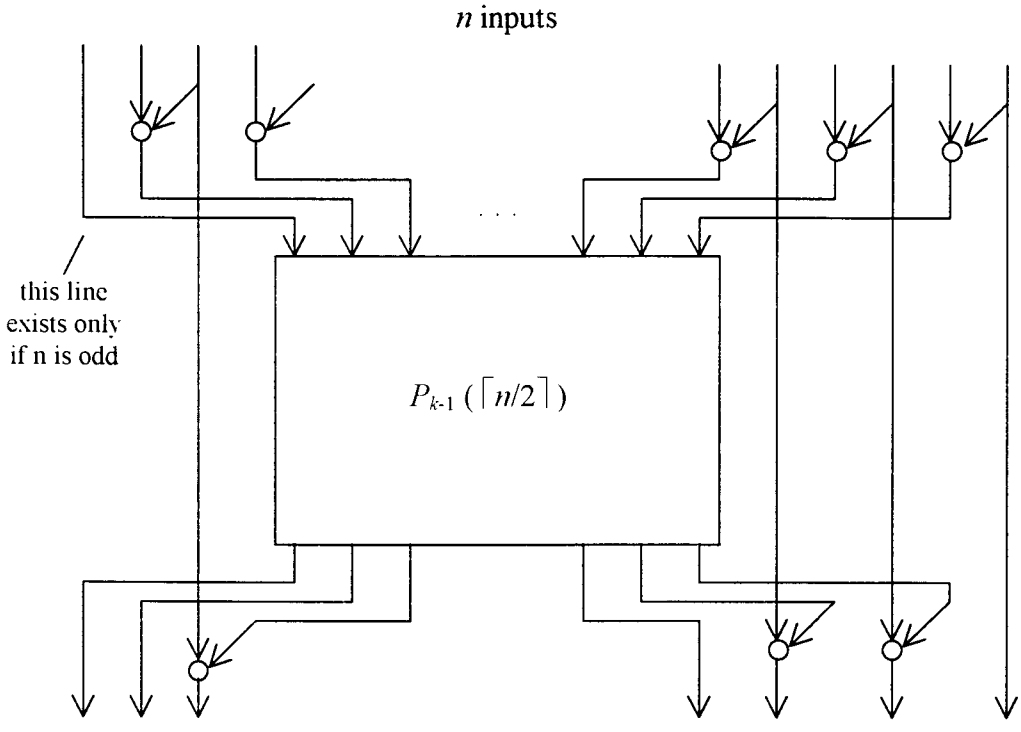


Figure 2.3 Recursive Construction of $P_k(n)$, $k \geq 1$

The function $F(m)$ is the m^{th} Fibonacci number. $F(m) = (\phi^m - \phi'^m) / \sqrt{5}$, where $\phi = (1 + \sqrt{5})/2$ and $\phi' = (1 - \sqrt{5})/2$.

Wei, Thompson, and Chen introduced a new family of circuits, $R(n)$, where the blocks have an arbitrary size (Figure 2.4) instead of a reduction of half. [3] Therefore, $R(n)$ is a super set of $P_0(n)$. The most significant bit (MSB) of the right block (bit m) has a large fan-out. When the depth of $R(n - m)$ is larger than the depth of $R(m)$, a

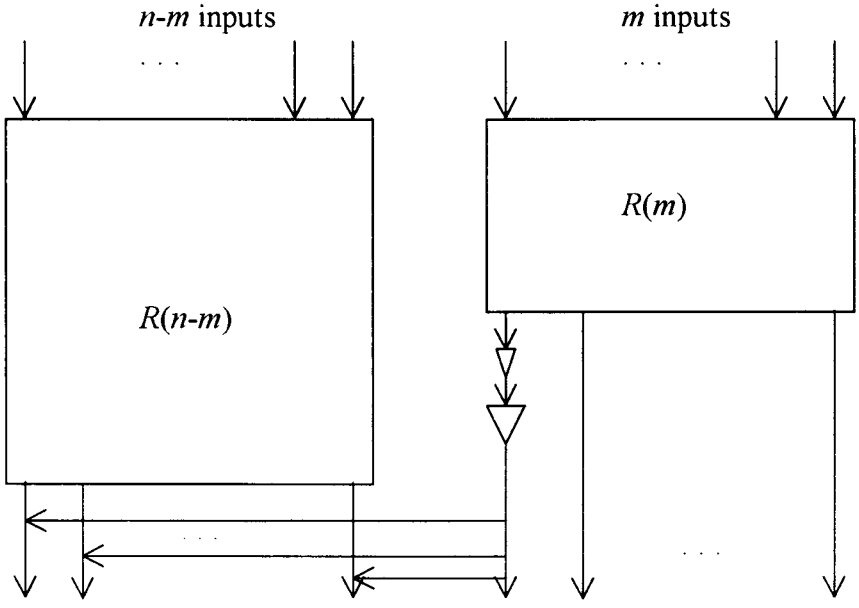


Figure 2.4 Drivers Used in Recursive Construction of $R(n)$

multi-stage driver is placed on the MSB of $R(m)$ as shown in Figure 2.4. The main objective for the new construction was to find the time-optimal adder. The multi-stage driver was used to solve the problem of delays associated with certain nodes having large fan-out.

In Figure 2.5, the construction of $P_k(n)$ for the basic graphs are shown, where $1 \leq n \leq 5$. The basic graphs shown are used to construct more complicated graphs for larger values of n.

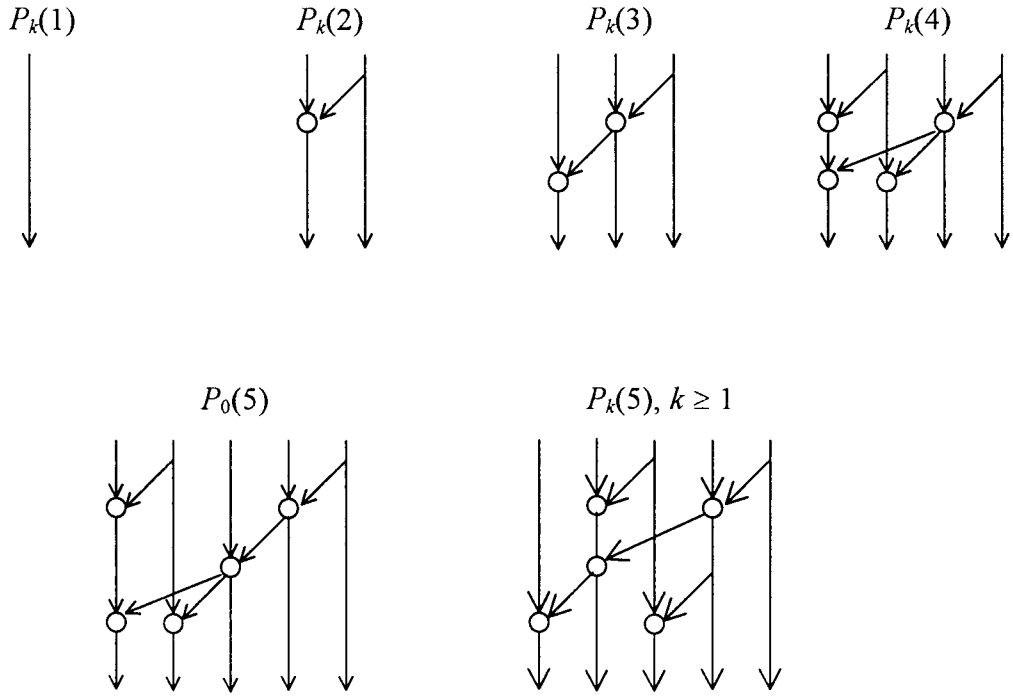


Figure 2.5 Base $P_k(n)$ Circuits for $1 \leq n \leq 5$

2.2 Circuit Formulations

Now that the concept of parallel prefix computation has been reviewed, the formulations of the nodes in the graphs for an adder can be introduced. The typical method used to compute the sum ($s_n s_{n-1} \ldots s_0$) of two n-bit operands ($a_n a_{n-1} \ldots a_0$ and $b_n b_{n-1} \ldots b_0$) is given by the equations

$$c_0 = 0,$$

$$c_i = a_i b_i + a_i c_{i-1} + b_i c_{i-1}, \text{ and}$$

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

The symbol $\oplus$ is the exclusive-or operation, and c_i is the carry generated by the i^{th} bit position. The generate and propagate values (g_i and p_i) are calculated for each bit position to offer an alternate way to determine c_i. The equations are

$$g_i = a_i b_i,$$

$$p_i = a_i + b_i,$$

$$c_0 = 0, \text{ and}$$

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

The latter equation is based upon the fact that a carry is generated if both input bits are 1 (high), or if at least one of the bits is high and there is an incoming carry from the previous bit. The equations above can be grouped into an associative operation, $\bullet$, such that

$$(g, p) \bullet (g', p') = (g + pg', pp'),$$

$$(G_i, P_i) = (g_i, p_i) \text{ if } i = 1,$$

$$(G_i, P_i) = (g_i, p_i) \bullet (G_{i-1}, P_{i-1}) \text{ if } 2 \le i \le n, \text{ and}$$

$$c_i = G_i,$$

where g, p, g', and p' are binary variables. The first equation represents the logic that each node in the prefix array evaluates. The left portion of the output is the G subcell and the right portion of the output is the P subcell. To make the nodes more efficient there are two types of circuits for each subcell, one with positive weighted inputs and one with negative weighted inputs. The static CMOS implementations of the various cells are shown in Figure 2.6 and Figure 2.7.

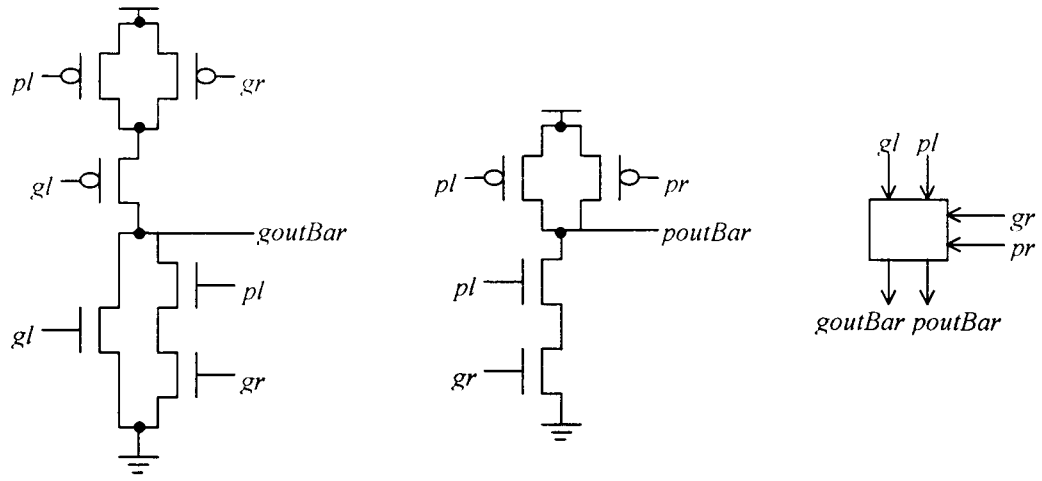


Figure 2.6 Static CMOS Implementations of G and P Subcells (positive weighted inputs)

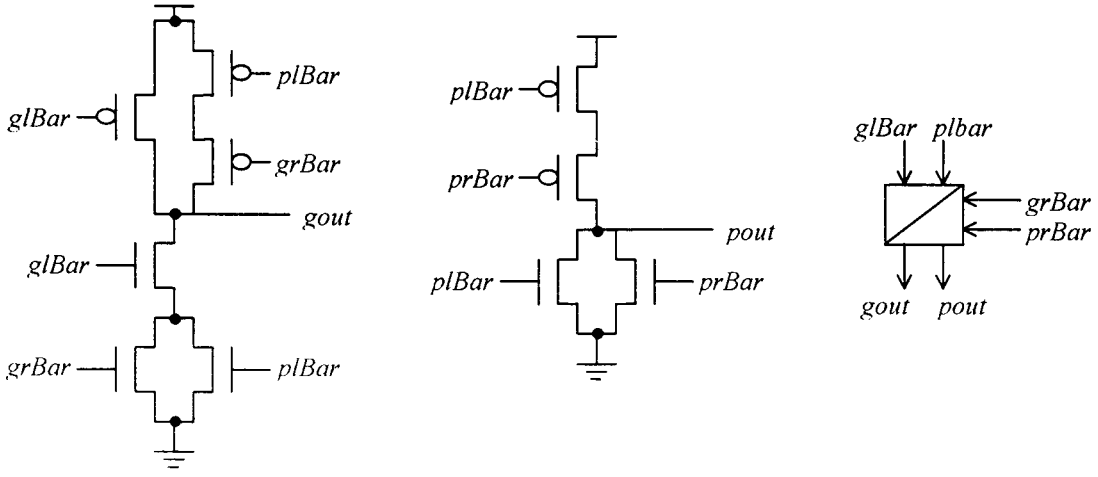


Figure 2.7 Static CMOS Implementations of G and P Subcells (negative weighted inputs)

The subcells in Figure 2.6 take in positive signals and output complemented signals. The subcells in Figure 2.7 take in complemented signals and output positive signals. The P and G subcells produce the output signals *pout* and *gout*, respectively. For

the inputs, the first letter stands for propagate or generate. The second letter stands for the direction the signal is coming from, either left or right. [3]

Typically, one uses minimum-length and minimum-width transistors for the pull-down transistors (NMOS). To produce a similar circuit response for the pull-up transistors (PMOS), minimum-length is used; however, the width needs to be increased by a factor of two. The minimum width for PMOS transistors is double the minimum width of NMOS transistors because the effective resistivity for PMOS is double the effective resistivity for NMOS. The higher resistivity slows response time, so the width must be increased to compensate. [5,8]

2.3 Propagation Delay

In the adder circuits, propagation delay is a function of fan-in, fan-out, and the depth. Here the main focus is propagation delay of the transistors with large loads. The propagation delay of a transistor is often calculated by using a simple first-order RC network. The resistance is dependent upon the width and length of the transistor's n-channel (NMOS) or p-channel (PMOS). Increased widths decrease the resistivity of the device, while decreasing the width causes a larger resistance. The capacitance is dependent upon the output capacitance of the driving transistor and the input capacitance of the load transistors. Typically, the gate capacitance, C_g, dominates all other capacitances by a factor of five to ten. The total resistance multiplied by the total load capacitance is the time constant, τ, of the network. The goal of minimum propagation

delay is to minimize τ by sizing transistors to provide minimum resistance and minimum capacitance. However, decreasing the width of a transistor, increases resistance and decreases capacitance, while increasing its width reduces resistance and increases capacitance. [8]

2.4 Power Consumption

The power consumption in a circuit is just as critical as the propagation delay. To optimize SPP, both must be reduced. In a CMOS circuit, power consumption is affected by several factors which include dynamic switching, short-circuit current, and static leakage current.

Dynamic switching is the dominant factor of power consumption. A capacitor is either charged or discharged any time an input changes and an output node voltage changes. The power dissipated in doing so is given by $P_{dyn} = C_L V_{dd}^2 f$. The power is dependent upon the size of the capacitor, the voltage supply, and the frequency. Larger capacitors store more energy. If the output switches from 0 V to V_{dd}, then one or more PMOS transistors will dissipate energy while the output is charged up to V_{dd}. If the output switches from V_{dd} to 0 V, then a capacitor is being discharged and the energy is dissipated through one or more NMOS transistors. Frequency is also a factor since increased activity of a circuit will require more power.

While dynamic switching contributes to most of the power usage, other factors are more apparent with lower supply and threshold voltages. Power consumption due to a short-circuit current occurs due to non-zero rise and fall times. There exists a direct path from V_{dd} to ground for a brief time when inputs are changing. This short circuit current occurs when both PMOS and NMOS transistors are on at the same time. The power due to direct-path currents is given by $P_{sc} = \frac{1}{2}(t_f + t_r)\, V_{dd}\ I_{peak}\ f$.

The third factor, static leakage, is partially caused by leakage current through the reversed-biased diode junctions in the transistors. The junction is either the source-substrate junction or the drain substrate junction. Another cause of static leakage is the subthreshold current. Smaller threshold voltages cause higher leakage currents. The static power, P_{stat}, is equal to the product $I_{leakage}\, V_{dd}$.

2.5 Summary of Review

At the beginning of this chapter, parallel prefix computation was described. There are two main families of parallel prefix circuits. The first is known as the $P_k(n)$ family and the second is the $R(n)$ family. Furthermore, the $R(n)$ family is a superset of the $P_0(n)$ family. Typical CMOS circuits used to evaluate logic in the nodes of a prefix circuit were also discussed. The third concept discussed was propagation delay. It is dependent upon a transistor's effective resistance and the total capacitive load on the output. The final topic of this chapter was power consumption. We described the major contributing factors including dynamic switching, short-circuit current, and static leakage current.

3. DESIGNS

Having reviewed the concepts of parallel prefix computation, propagation delay, and power consumption, different designs may now be examined in detail. The circuit simulator HSPICE was used to simulate three different parallel prefix designs. The first adder is based on the $P_k(n)$ family[1][2]. The second design is from the $R(n)$ family[3]. The final and third design is based on the basic concept of recursive doubling. All of the designs simulated are 32-bit adders. Besides having two 32-bit operands and an output sum, they each have a C_{in} and a C_{out}. The carry propagation with look-ahead portion of the circuit is the main difference among the three designs. Some XOR and XNOR gates are interchanged in the summation portion to maintain polarity correctness.

3.1 Brent-Kung Adder

The first design is based on the $P_k(n)$ family from Ladner and Fisher's work [1]. It is labeled as Brent-Kung because of the formulation of each node in the carry look-ahead array. The CMOS circuits for the node formulations are in Section 2.2. Brent and Kung used these CMOS circuits for their adder in [2]. They used a variation of recursive doubling for their carry look-ahead circuit. Unlike the Brent-Kung(BK) Adder, the other adders are named according to the carry look-ahead implementation. The particular design examined here is $P_0(32)$. This design has the smallest depth in the family and it has the largest number of nodes in its graph. This would imply that this circuit has the smallest delay and will use the most power.

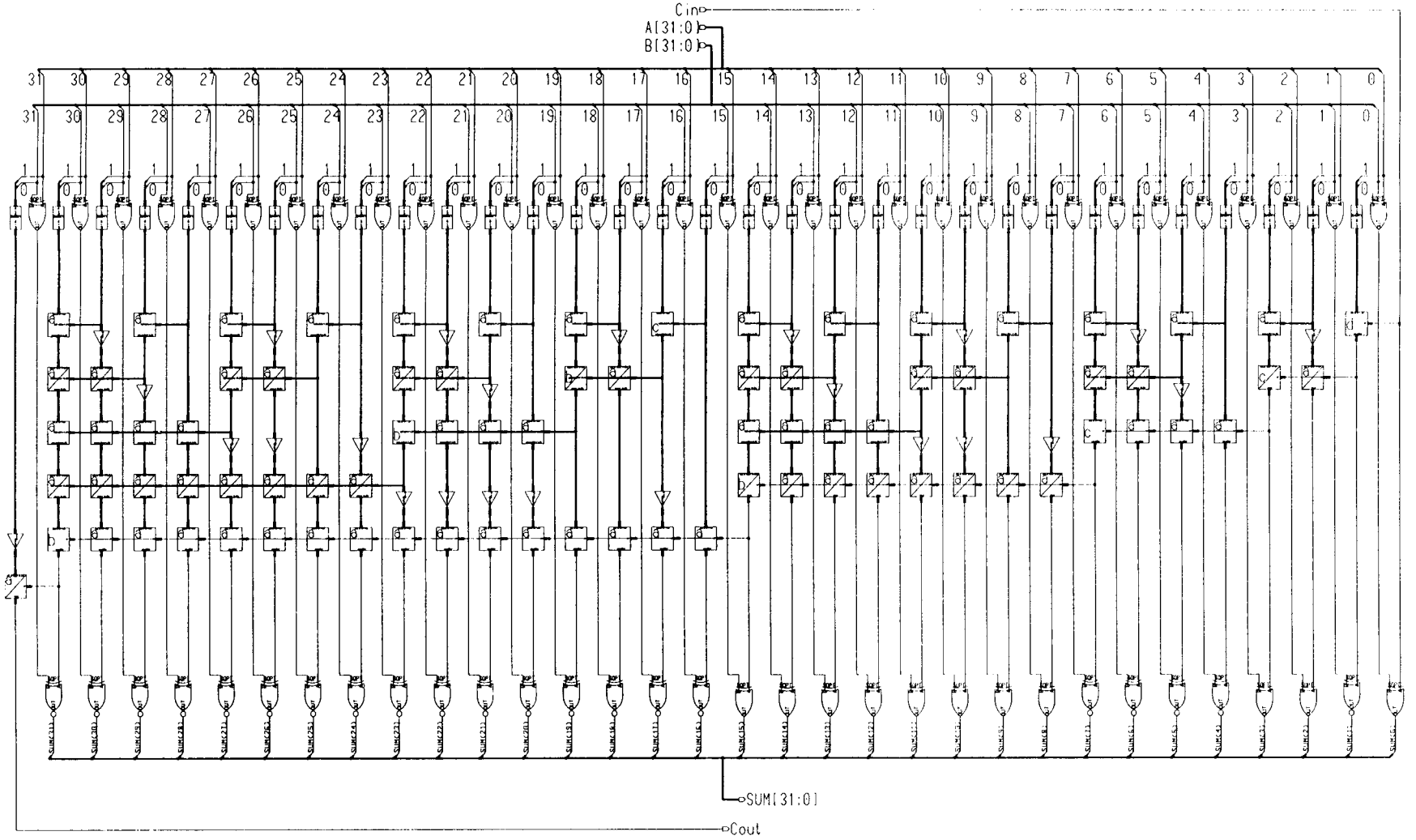


Figure 3.1 Brent-Kung Adder

The $P_0(32)$ circuit used in the simulations is shown in Figure 3.1. For symmetry and simplicity, $P_0(n)$ is recursively constructed from two sets of $P_0(n/2)$ instead of $P_0(n/2)$ and $P_1(n/2)$ (Figure 2.2). The top row of XOR gates and other combinational logic are for the pre-calculations of generate and propagate terms. Inverters are included in selective places because the signal changes polarity between positive weight and negative weight after each level of nodes. The bottom row of XOR and XNOR enable the final summation of the operands and the carries that were calculated in the parallel prefix computation. The bottom node for each column in the carry look-ahead (CLA) is different from the regular nodes since they only need to evaluate the G subcell.

There is a relationship between the capacitive load of a node and the delay that a node will experience when evaluating a new value. The nodes that act as loads to other nodes increase the capacitive load. Since some nodes are driving multiple nodes at the same time, there must be an increase in drive capability to offset the increased delay.

Many of the nodes near the bottom of the CLA have large fan-out. The bottom node in sum bit 15 has a load of 16 other nodes (to the left) plus an XOR gate. This large fan-out creates a large delay, which hinders fast propagation of the signals. A common solution is to put in a multi-stage driver between the node and its load. However, in this type of circuit it would not be feasible since it would actually increase the overall logic depth. The minimum depth of this design was the reason it was chosen. There are other CLA designs where multi-stage drivers are more appropriate. The multi-stage driver will be presented in the next section where Wei[3] found the time-optimal design.

A progressive sizing approach to find a power and time optimal design is used for this adder. Instead of putting in a multi-stage driver, the transistors in the node will have widths that are a fraction of the load. For instance, if the load is 16 nodes and the ratio is two, then the loaded node is given a relative size of eight. A loaded node is a node that has a fan-out of two or more. A fan-out of one is equivalent to the load of one inverter. For the circuit in Figure 3.1, the bottom loaded nodes are not minimum size (sum bits 15, 7, 3, and 1). Also, the second-to-last loaded nodes in the left half of the circuit are not minimum size (sum bits 23, 19, and 17). The intention is to show that circuits are more effective with re-sized nodes than those with larger depth and multi-stage drivers.

The node size ratio equations begin with the load of the C_{out} node and they go back along the bottom row of nodes, among other nodes mentioned previously. The equations are included in Appendix C and are repeated here in a more readable form.

WCa=Wmin

WCbara=Wmin

*WCb='(3*Wmin)*r'*

*WCbarb='(17*Wmin+WCb)*r'*

*WCc='(9*Wmin+WCbarb)*r'*

*WCbarc='(5*Wmin+WCc)*r'*

*WCd='(3*Wmin+WCbarc)*r'*

WGPa=Wmin

WGPbara=Wmin

*WGPb='(9*Wmin)*r'*

*WGPbarb='(4*Wmin+WGPb)*r'*

*WGPc='(2*Wmin+WGPbarb)*r'*

The "*W*" stands for transistor width. A "*C*" following the "*W*" stands for a partial node that is in the bottom generating the final carry for its respective bit. When a *"GP"* follows the name, then that is a full node with both outputs. The small letter at the end is used to differentiate the different loaded nodes. An "*a*" is a minimum sized node. A "*b*" is the first loaded node at the bottom-left portion of the carry generator, and so on until the last loaded node. The "bar" in some of the names refers to a node that inputs and outputs signals opposite in polarity to that of the regular node (refer to Section 2.2). The nodes whose sizes are being increased are on the critical delay paths.

3.2 Wei, Thompson, and Chen Adder

The second design, part of the *R*(*n*) family [3], is henceforth referred to as Wei's Adder. The adder has the same type of pre-computation circuit and sum circuit, while the carry generator is different. Since the block size can vary greatly (refer to Figure 2.4), Wei created a model to find out which combination of block sizes works the best. He included the use of multi-stage drivers for loaded nodes. The time-optimal circuit is shown in Figure 3.2.

There are no published results on the power usage optimization for this design. Wei's goal was to find a circuit that was optimized only for time delay. Once again,

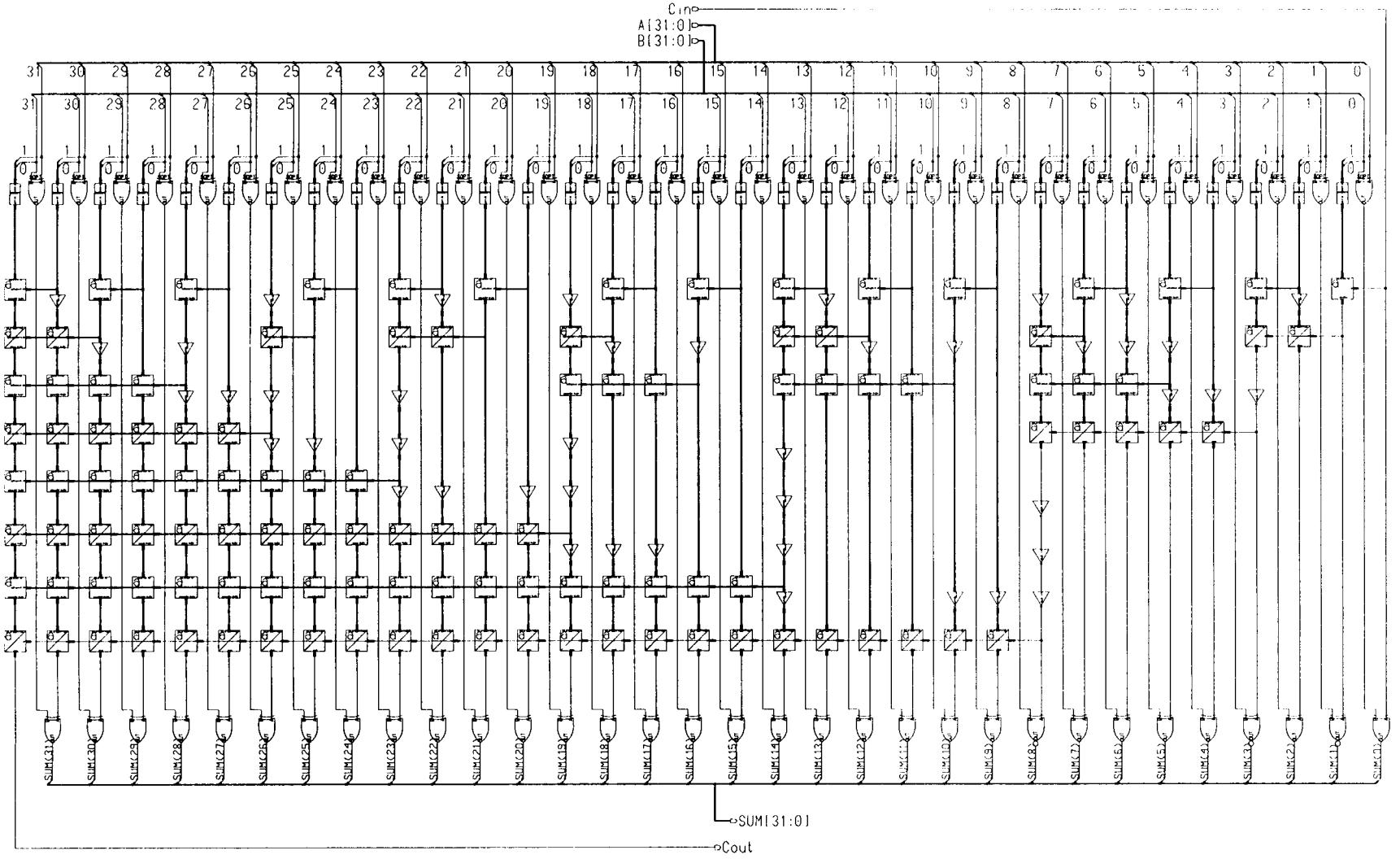


Figure 3.2 Wei's Adder

minimum-sized inverters are placed to keep the signal polarity correct. Here, the depth of the carry generator graph is eight, two levels more than the BK adder. The extra multi-stage drivers also affect the size of the adder. There are 115 nodes in Wei's Adder versus 81 nodes in the BK adder.

A simple ratio was not used to find the proper ratio of the multi-stage driver. For the cascaded driver, propagation time is a function of the fan-out, f, the ratio between stages, r, and the number of stages, s. The driver ratio is $r = f^{1/(s+1)}$ for minimum propagation delay [3]. As one saw earlier in Section 2.1 and Figure 2.3, the drivers are placed between each of the recursive blocks. From Figure 3.2, it can be seen that the number of stages in a driver ranges from one to three.

The intent of the design is that it be the fastest with the multi-stage drivers to compensate for the large fan-out. However, the BK Adder will be faster if specific nodes are increased in size as described in Section 3.1. The power consumption for Wei's Adder will be higher than the BK Adder because Wei's Adder requires more transistors.

3.3 Recursive Doubling Adder

The third design is the Recursive Doubling Adder (Figure 3.3). The maximum fan-out in this adder is two. For the top row, signals go to a node on the current bit and to the node on the next bit over to the left. For the second row, signals go to the current

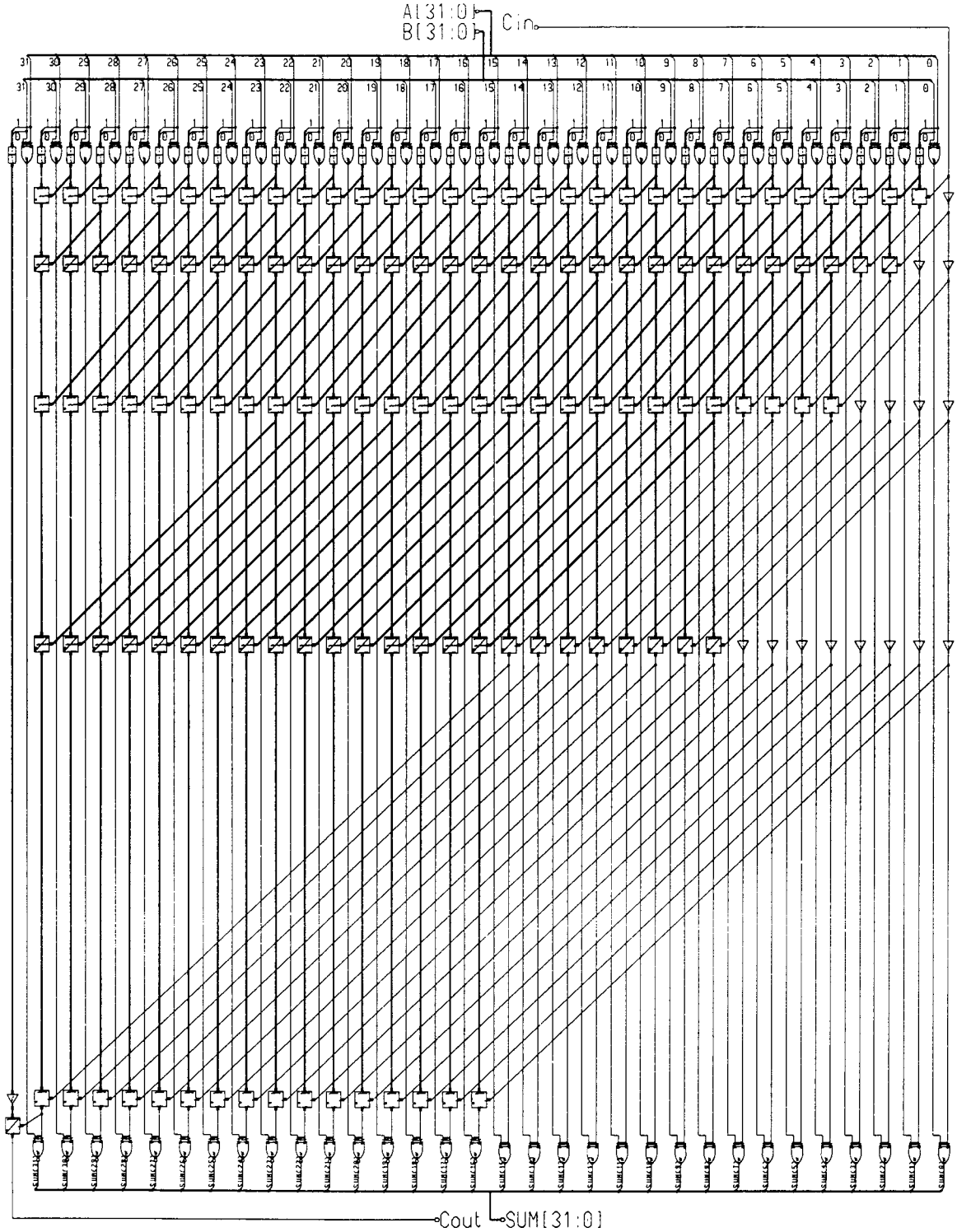


Figure 3.3 Recursive Doubling Adder

node and two nodes over to the left. The number of nodes that the signals skip doubles each time. For the last row (not including the C_{out} node), signals skip over 16 bits.

The Recursive Doubling Adder has a total of 130 nodes and its depth is six. Since there are no nodes with large fan-out, all of the nodes are minimum size. The power usage for this adder will be greater than or equal to the power usage of Wei's Adder (Section 3.2) because it requires more nodes than Wei's Adder. The Recursive Doubling Adder is the fastest of the group because it has the minimum depth possible for a 32-bit adder. Signals propagate quickly through all parts of the circuit due to the low fan-out.

3.4 Summary of Designs

The three different 32-bit adder designs simulated in HSPICE were presented. The first design was the BK Adder, part of the $P_k(n)$ family (Figure 3.1). The BK Adder uses the least amount of transistor devices and its carry look-ahead circuit has a minimum logical depth of six levels. Wei's Adder was the second adder, and it is part of the $R(n)$ family (Figure 3.2). This adder's carry look-ahead portion has a logical depth of eight levels and has a bigger size. The third adder was the Recursive Doubling Adder. Its carry look-ahead is six levels, and it uses the highest number of transistors.

Table 3.1 summarizes all three designs with respect to size and depth without accounting for the re-sized nodes. The Depth column shows the total number of logic levels for each adder's carry look-ahead circuit. The Size column shows the total number of nodes in each adder's carry look-ahead circuit. Since depth and size roughly

approximate delay and power, respectively, the Depth*Size column estimates the speed-power product for each adder. From Table 3.1, one should expect the BK Adder to have the best overall performance.

	Depth (levels)	Size (nodes)	Depth*Size
1. BK	6	81	486
2. Wei	8	115	920
3. Recursive Doubling	6	130	780

Table 3.1 Depth and Size Variations Between the Three Designs

4. EXPERIMENTAL RESULTS

There are four separate sets of propagation delay and power data from the designs described in Chapter 3. The product of the propagation delay(tp) and power was computed to present the speed-power product (SPP). Wei's Adder and the Recursive Doubling Adder each has one set of data, while the Brent-Kung (BK) adder has two sets of data. The difference between the two sets of data for the BK adders is that one used a fan-out size ratio of two ($r=2$), and the other one used a fan-out size ratio of three ($r=3$).

Section 4.1 describes the variations for the simulations that were run. Section 4.2 presents the propagation delays from each simulation, and Section 4.3 presents the SPP results. Section 4.4 describes simulations run to compare power leakage for various values of V_t. Interconnect is briefly discussed in Section 4.5. The HSPICE simulation files are in Appendix A, Appendix B, and Appendix C. Please note that the parameter *r* (driver size ratio) has a multiplicative inverse meaning in the netlist file in Appendix C.

4.1 Simulation Variations

For each of the four different adders, simulations were run with two different variations using a constant input frequency. The two variations were voltage supply, V_{dd}, and change in threshold voltage, dV_t. V_{dd} was varied from 1.5 V to 3.0 V in steps of 0.5 V, and dV_t was varied from -0.2 V to 0.6 V in steps of 0.1 V. A positive value for dV_t means a decrease in the threshold voltage, V_t, and a negative value means an increase in

V_t. A simple C-program created a file for random input vectors to the adders (Appendix D). There were 500 vectors simulated for each design. Since HSPICE integrates over the entire time interval of the 500 random vectors, the power measurement calculation represents a good average.

4.2 Propagation Delay

The propagation delay measurements taken were a measurement of the 50% point on the input rise (or fall) to the 50% point on C_{out}'s rise (or fall). Slightly over half of the 500 random input vectors caused a voltage swing in the C_{out} bit line. C_{out} has the largest theoretical delay since it is the bit line with the largest depth for each design. However, the actual worst case delay may differ due to transistors in series for each of the gates of a particular path. So, the delay data for each particular set of voltage parameters is the worst case of about 250 raw measurements from SPICE.

The following four pages contain the propagation delay data in graphical and in table format. The four different tables and the four different figures account for the four different V_{dd} values beginning with 3.0 V and ending with 1.5 V. The tables contain delay, power, and SPP values, so the data will not be repeated in the Section 4.3.

In the top graph of each figure, the delays are graphed with respect to V_t. In the bottom graph of each figure, the delays are graphed with respect to V_{dd}/V_t. In every

Type of Adder			Brent-Kung r=2			Brent-Kung r=3			Wei			Recursive Doubling		
dVt	Vt	Vdd/Vt	tp	Power	SPP	tp	Power	SPP	tp	Power	SPP	tp	Power	SPP
-0.2	0.9	3.5	3.6	5.5	19.7	3.7	5.4	19.8	3.8	6.1	23.3	3.3	5.8	19.1
-0.1	0.8	3.9	3.3	5.6	18.6	3.4	5.5	18.5	3.5	6.2	22.1	3.0	5.9	18.0
0.0	0.7	4.5	3.1	5.7	17.7	3.2	5.6	17.6	3.3	6.4	21.1	2.8	6.1	17.1
0.1	0.6	5.3	2.9	5.9	17.0	2.9	5.7	16.8	3.1	6.6	20.2	2.6	6.2	16.4
0.2	0.5	6.4	2.7	6.1	16.4	2.8	5.9	16.3	2.9	6.8	19.9	2.5	6.4	15.9
0.3	0.4	8.2	2.5	6.3	16.2	2.6	6.1	15.9	2.8	7.1	19.7	2.4	6.7	15.7
0.4	0.3	11.2	2.4	6.9	16.8	2.5	6.6	16.3	2.6	7.9	20.9	2.2	7.2	16.2
0.5	0.2	17.9	2.3	9.4	21.9	2.4	8.9	21.0	2.6	11.7	30.0	2.1	10.3	21.9
0.6	0.1	44.5	2.2	14.8	33.2	2.3	13.7	31.6	2.5	19.8	49.5	2.0	17.0	34.6
* dVt (Volts), Vt (Volts), Vdd/Vt (Volts/Volts), tp (ns), Power (mW), SPP (pJ)														

Table 4.1 Propagation Delay (V_{dd} = 3.0 Volts)

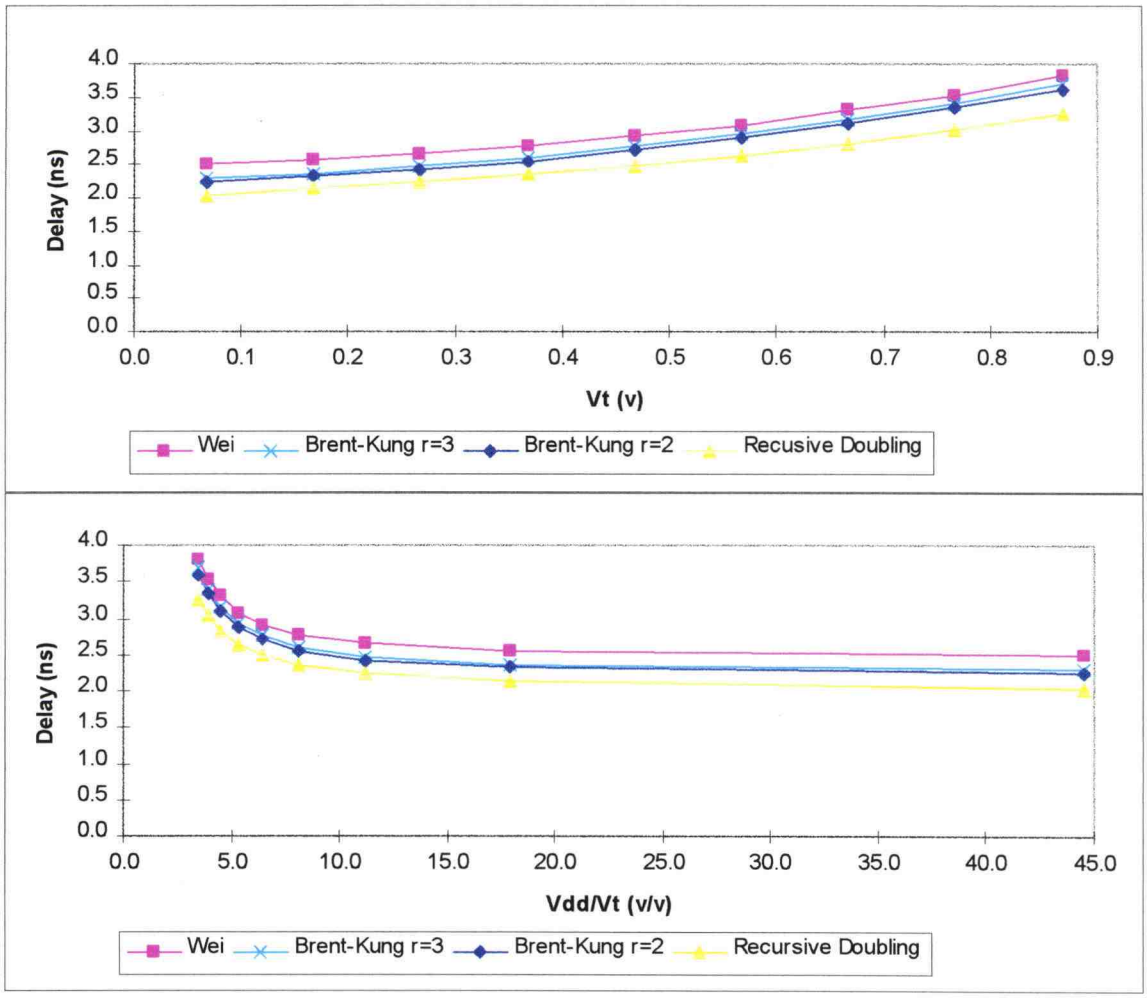


Figure 4.1 Propagation Delay (V_{dd} = 3.0 Volts)

Type of Adder			Brent-Kung r=2			Brent-Kung r=3			Wei			Recursive Doubling		
dVt	Vt	Vdd/Vt	tp	Power	SPP	tp	Power	SPP	tp	Power	SPP	tp	Power	SPP
-0.2	0.9	2.9	5.0	3.8	19.0	5.1	3.7	19.0	5.3	4.2	22.3	4.5	4.0	18.2
-0.1	0.8	3.3	4.5	3.8	17.3	4.6	3.7	17.3	4.7	4.3	20.3	4.1	4.1	16.7
0.0	0.7	3.7	4.1	3.9	16.1	4.2	3.8	16.2	4.3	4.4	18.9	3.7	4.2	15.5
0.1	0.6	4.4	3.7	4.0	15.0	3.8	3.9	14.9	4.0	4.5	17.8	3.4	4.3	14.5
0.2	0.5	5.3	3.4	4.1	14.2	3.5	4.0	14.1	3.6	4.6	16.9	3.1	4.4	13.8
0.3	0.4	6.8	3.2	4.3	13.7	3.2	4.2	13.5	3.4	4.8	16.4	2.9	4.6	13.2
0.4	0.3	9.3	2.9	4.6	13.5	3.0	4.4	13.3	3.2	5.2	16.6	2.7	4.8	13.1
0.5	0.2	14.9	2.8	5.8	16.2	2.8	5.5	15.5	3.0	7.0	21.3	2.5	6.2	15.8
0.6	0.1	37.1	2.7	9.1	24.1	2.7	8.5	22.9	3.0	11.9	35.1	2.4	10.2	24.8
* dVt (Volts), Vt (Volts), Vdd/Vt (Volts/Volts), tp (ns), Power (mW), SPP (pJ)														

Table 4.2 Propagation Delay (V_{dd} = 2.5 Volts)

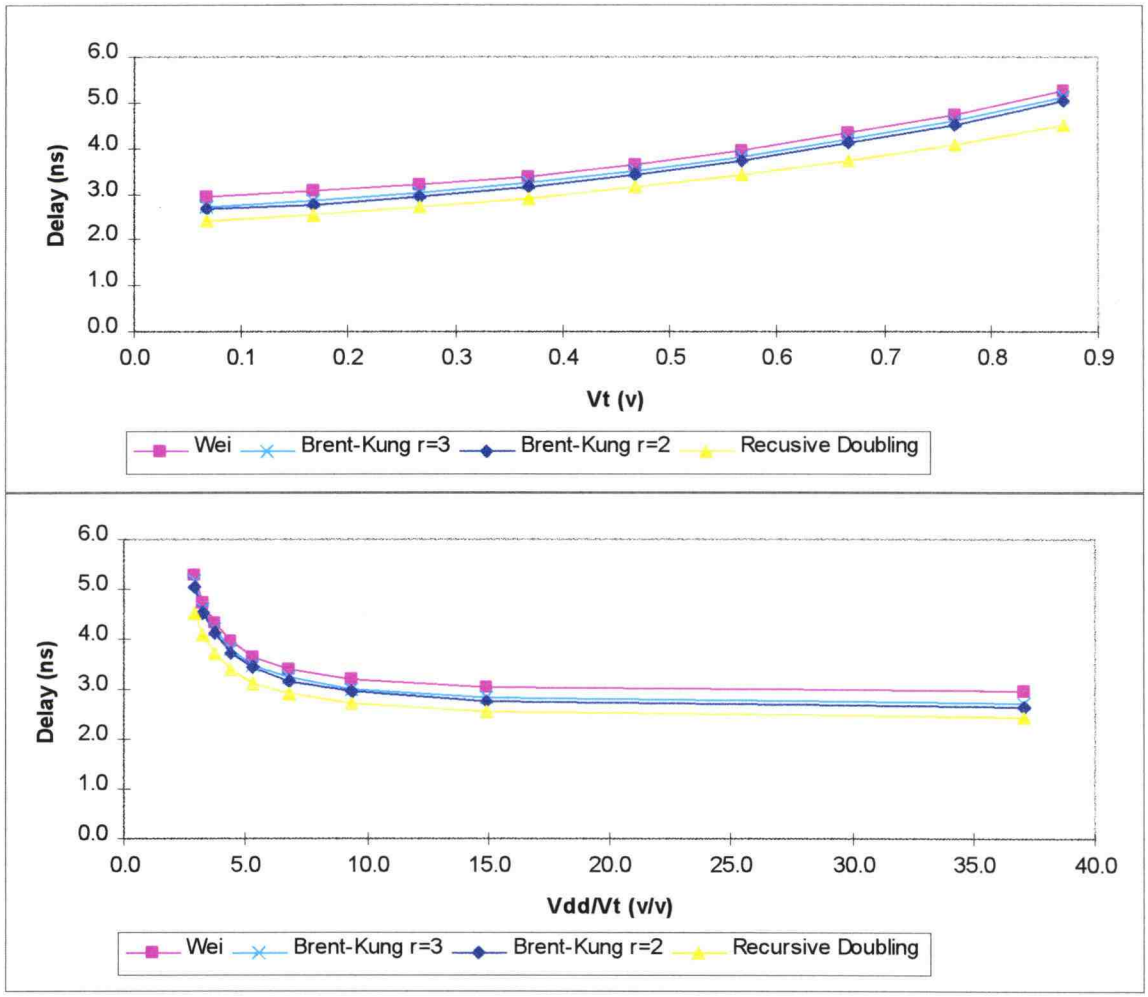


Figure 4.2 Propagation Delay (V_{dd} = 2.5 Volts)

Type of Adder			Brent-Kung r=2			Brent-Kung r=3			Wei			Recursive Doubling		
dVt	Vt	Vdd/Vt	tp	Power	SPP	tp	Power	SPP	tp	Power	SPP	tp	Power	SPP
-0.2	0.9	2.3	8.4	2.4	20.0	8.6	2.3	20.0	8.7	2.7	23.3	7.4	2.6	19.0
-0.1	0.8	2.6	7.1	2.4	17.2	7.3	2.4	17.2	7.4	2.7	20.2	6.3	2.6	16.4
0.0	0.7	3.0	6.2	2.5	15.2	6.3	2.5	15.5	6.4	2.8	17.8	5.5	2.6	14.5
0.1	0.6	3.5	5.4	2.5	13.6	5.5	2.5	13.5	5.7	2.8	16.0	4.8	2.7	13.1
0.2	0.5	4.3	4.8	2.6	12.4	4.9	2.5	12.3	5.0	2.9	14.5	4.3	2.8	11.9
0.3	0.4	5.4	4.3	2.7	11.5	4.3	2.6	11.4	4.5	3.0	13.6	3.9	2.9	11.1
0.4	0.3	7.5	3.8	2.9	11.0	3.9	2.8	10.8	4.1	3.2	13.2	3.5	3.0	10.6
0.5	0.2	11.9	3.5	3.3	11.6	3.6	3.2	11.3	3.8	3.9	14.7	3.2	3.5	11.3
0.6	0.1	29.7	3.3	5.2	17.0	3.3	4.8	16.2	3.6	6.6	23.9	3.0	5.7	17.2
* dVt (Volts), Vt (Volts), Vdd/Vt (Volts/Volts), tp (ns), Power (mW), SPP (pJ)														

Table 4.3 Propagation Delay (V_{dd} = 2.0 Volts)

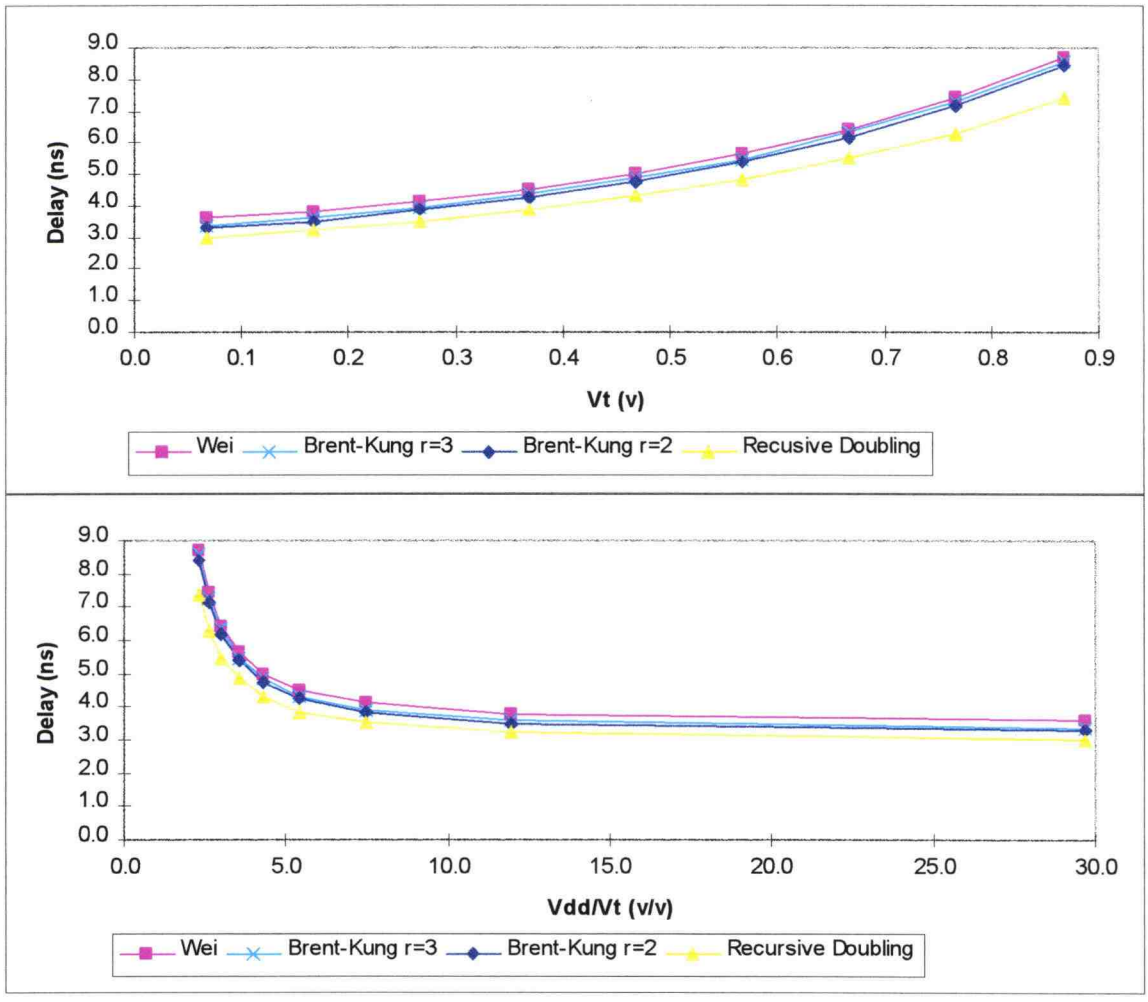


Figure 4.3 Propagation Delay (V_{dd} = 2.0 Volts)

Type of Adder			Brent-Kung r=2			Brent-Kung r=3			Wei			Recursive Doubling		
dVt	Vt	Vdd/Vt	tp	Power	SPP	tp	Power	SPP	tp	Power	SPP	tp	Power	SPP
0.0	0.7	2.2	12.5	1.3	16.6	12.7	1.3	17.1	13.0	1.5	19.6	10.9	1.5	15.8
0.1	0.6	2.6	9.8	1.4	13.5	10.0	1.3	13.5	10.3	1.6	16.0	8.7	1.5	12.9
0.2	0.5	3.2	8.0	1.4	11.3	8.1	1.4	11.3	8.3	1.6	13.3	7.1	1.5	10.8
0.3	0.4	4.1	6.7	1.5	9.9	6.8	1.4	9.8	6.9	1.6	11.5	6.0	1.6	9.5
0.4	0.3	5.6	5.7	1.6	8.9	5.8	1.5	8.8	6.0	1.7	10.4	5.1	1.7	8.5
0.5	0.2	9.0	4.9	1.7	8.6	5.0	1.7	8.4	5.2	2.0	10.3	4.5	1.8	8.2
0.6	0.1	22.3	4.4	2.6	11.5	4.5	2.4	10.9	4.8	3.2	15.6	4.0	2.8	11.4
* dVt (Volts), Vt (Volts), Vdd/Vt (Volts/Volts), tp (ns), Power (mW), SPP (pJ)														

Table 4.4 Propagation Delay (V_{dd} = 1.5 Volts)

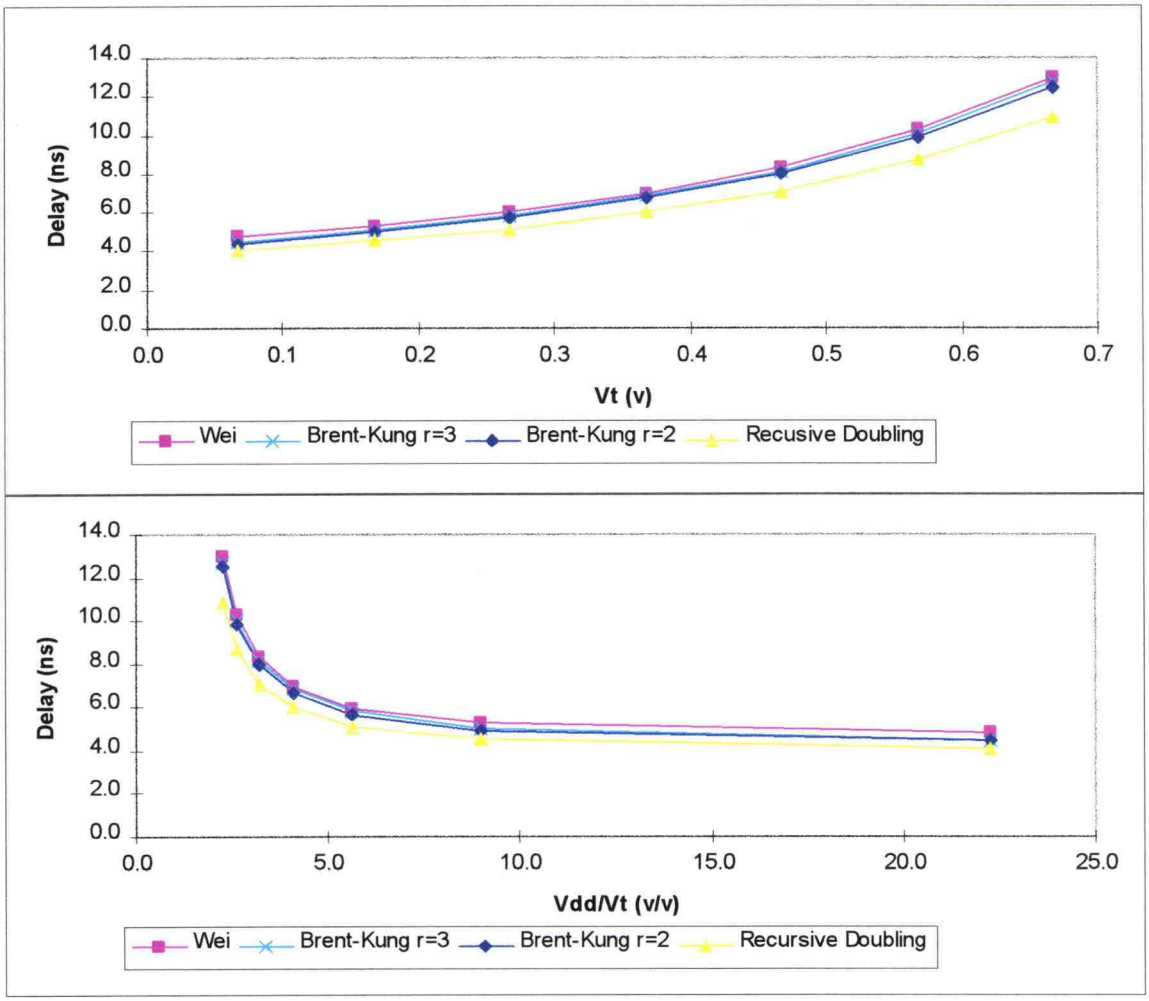


Figure 4.4 Propagation Delay (V_{dd} = 1.5 Volts)

case, the Recursive Doubling adder outperforms the other designs with respect to propagation delay. Wei's Adder, designed to be time-optimal, is actually the slowest. The two BK Adders have delays very close to each other. As expected, the BK Adder with r=2 has a slightly better propagation delay than the adder with r=3.

Each adder follows an interesting trend on every graph. One will notice that as V_t decreases, the slope of the delay line approaches zero. In other words, there is a critical point where the performance stops improving with the reduction of V_t. This is mainly because V_t is only one of the many factors that affect propagation delay such as V_{dd}, line resistance, line capacitance, load capacitance, and circuit design. Propagation delay is also squarely dependent upon the difference between V_{dd} and V_t. Increasing V_t causes an increase in the propagation delay. On the lower half of each figure, one will notice that higher values of V_t cause time delay to increase squarely. The lower values of V_{dd} are more susceptible to a change in V_t. For example, when V_{dd}=3.0 V and V_t is lowered to 0.1 V (Figure 4.1), tp decreases by 1/3. Yet, when V_{dd}=1.5 V and V_t is lowered to 0.1 V, tp decreases by 2/3. A higher voltage supply causes a faster design. For two different figures where the V_{dd}/V_t ratio is equivalent, the one with a higher V_{dd} is faster.

4.3 Speed-Power Product

The speed-power product (SPP) is equal to the propagation delay (in ns) times the power consumption (in mW). SPP is a measure of a circuit's energy efficiency. In microprocessors, energy consumption is important in circuit design.

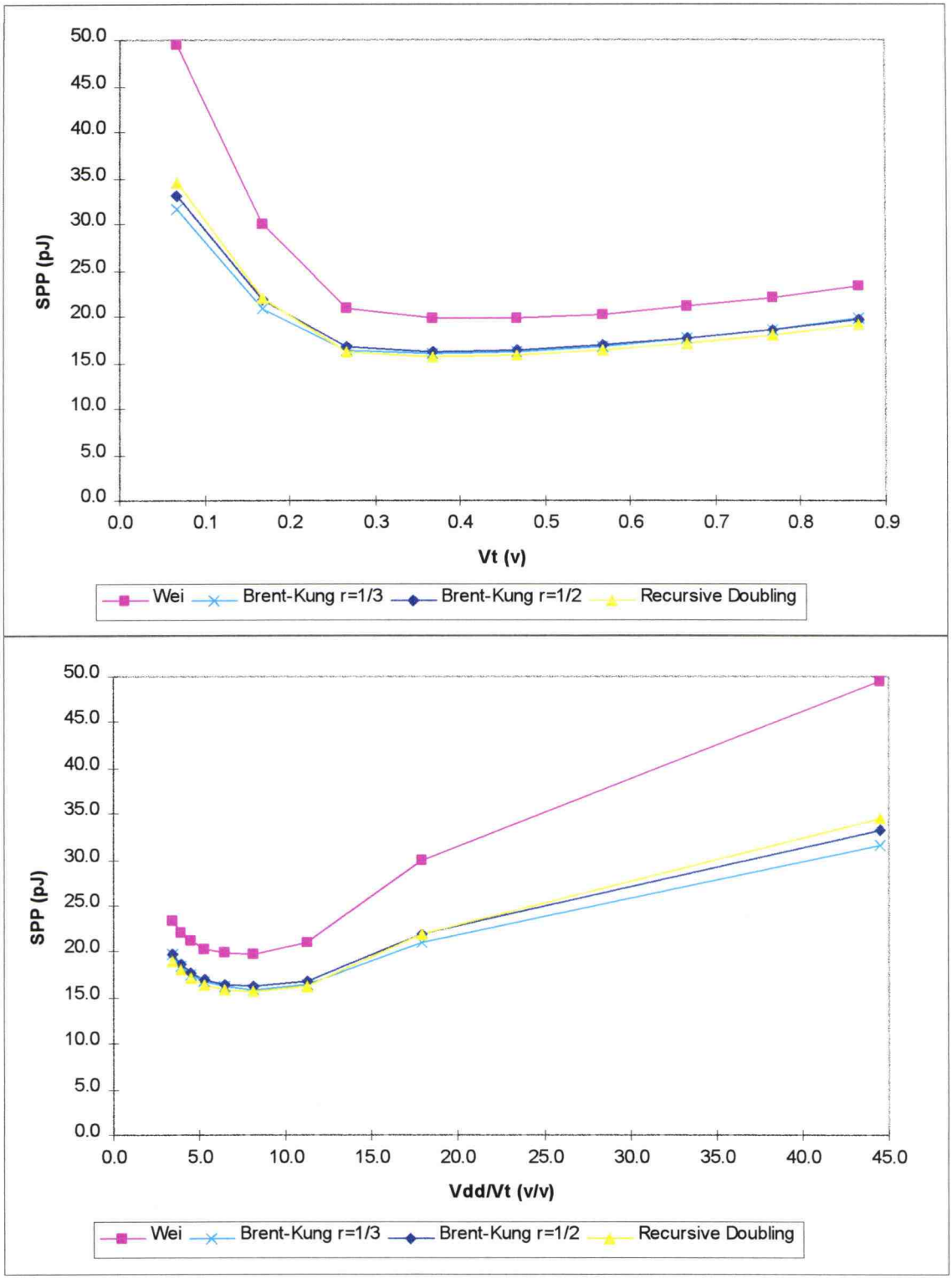


Figure 4.5 Speed-Power Product (V_{dd} = 3.0 Volts)

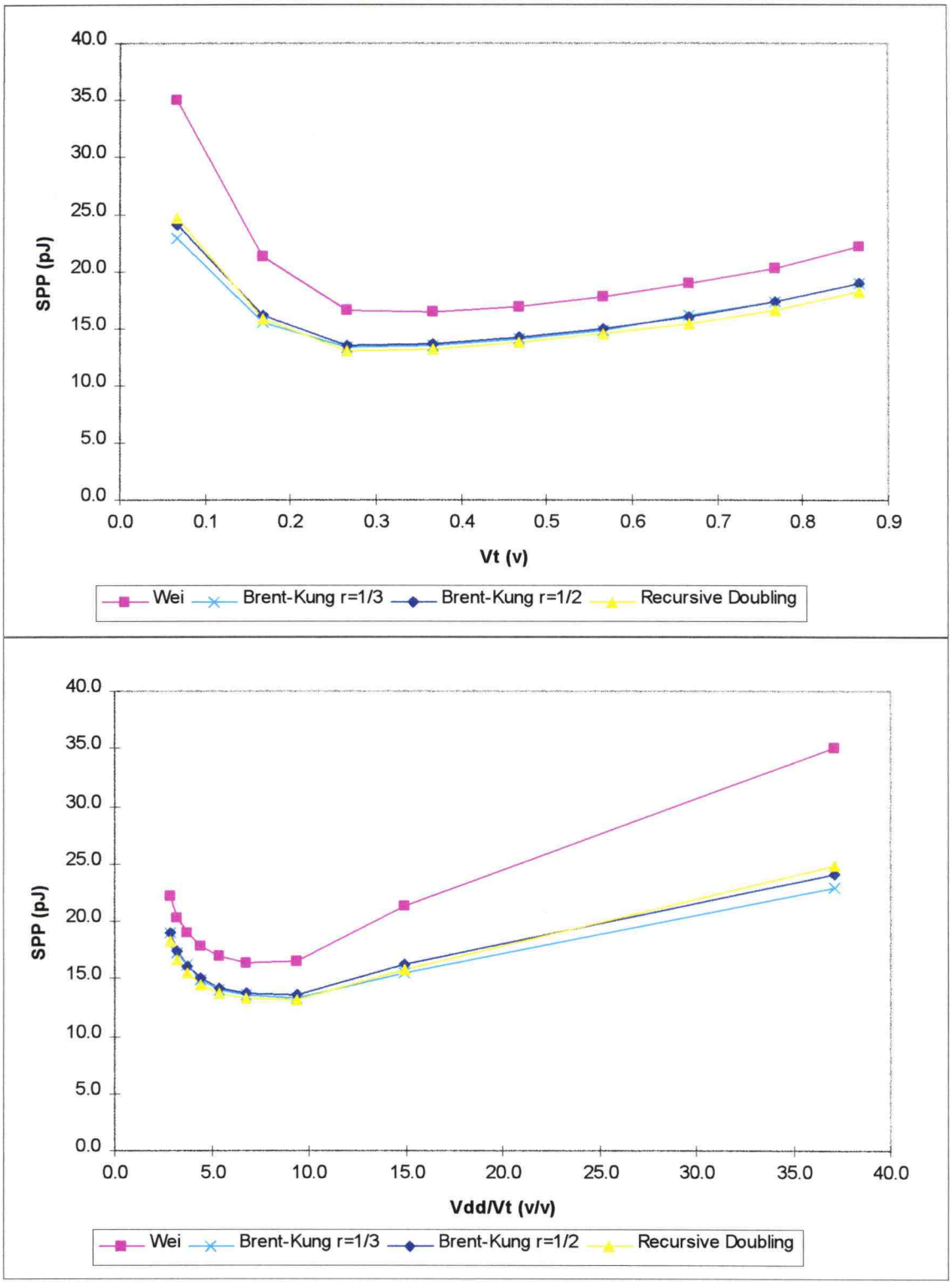


Figure 4.6 Speed-Power Product (V_{dd} = 2.5 Volts)

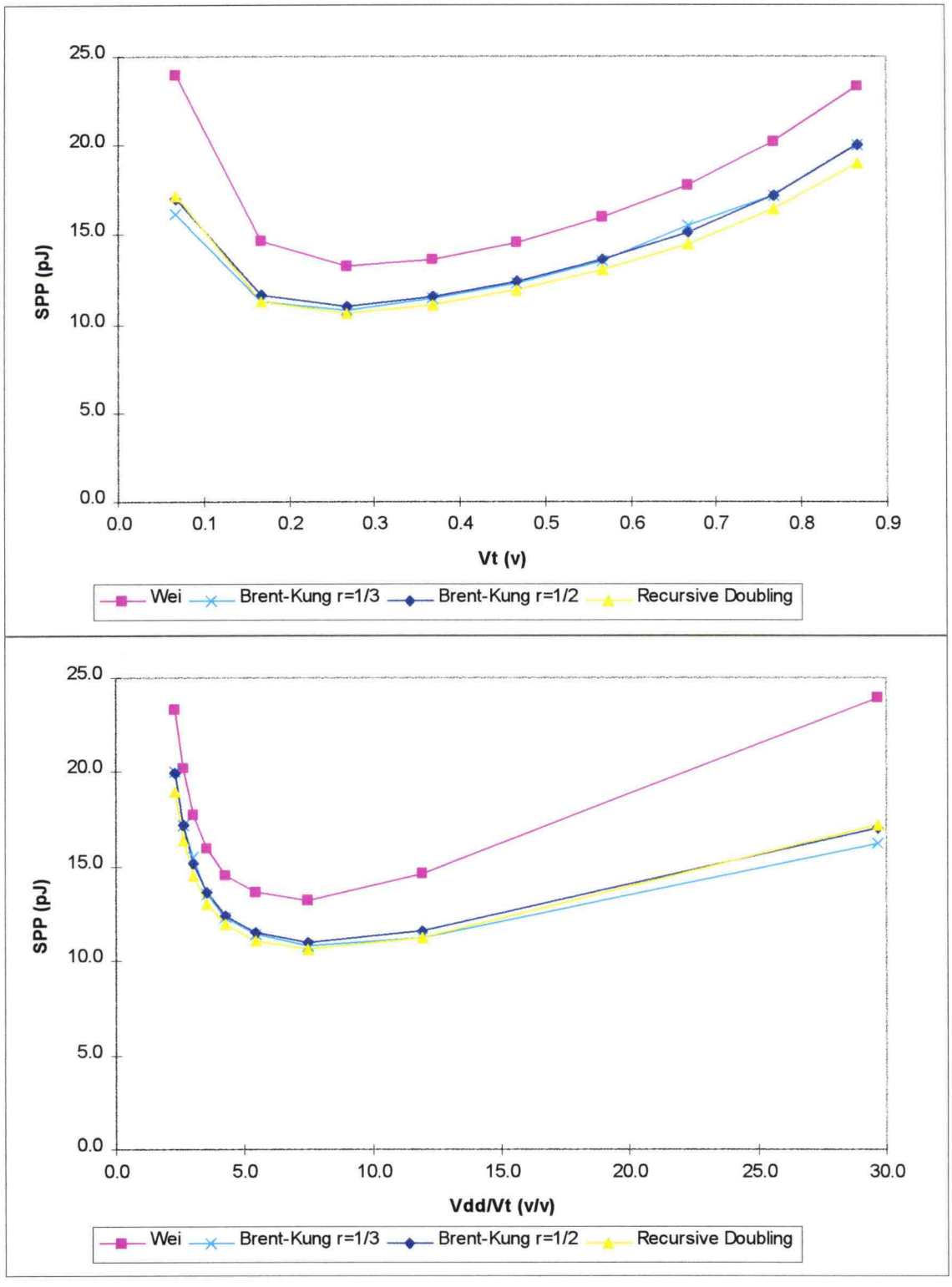


Figure 4.7 Speed-Power Product (V_{dd} = 2.0 Volts)

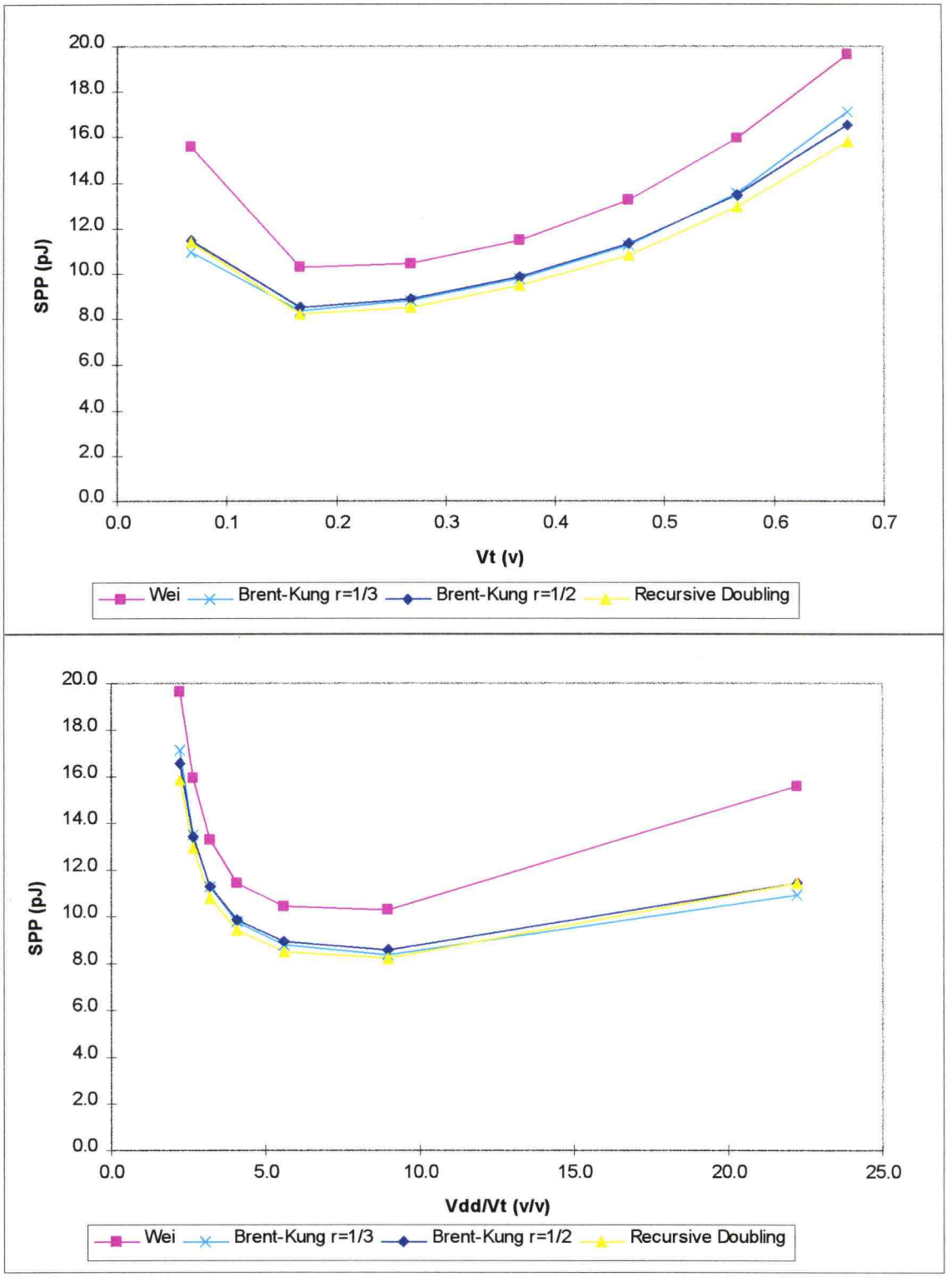


Figure 4.8 Speed-Power Product (V_{dd} = 1.5 Volts)

Similar to the propagation delay figures, the SPP graphs are also split up into four separate figures. The SPP data from Tables 4.1 through 4.4 are graphed in Figures 4.5 through 4.8. Also, the top half of each figure shows SPP versus V_t, and the bottom half of each figure shows SPP versus V_{dd}/V_t.

The first observation to make from the SPP graphs is that there exists a positive curvature for each of the sets of data. In other words, there exists a minimum SPP at a particular threshold value in between the highest and lowest values of V_t. The SPP data does not plateau as the delay did. The trend of the data of different designs for the SPP graphs is similar for each particular voltage. The minimum SPP occurs at a lower V_t for lower voltage supplies. At V_{dd}=3.0 V (Figure 4.5), the minimum SPP is around V_t=0.4 V, and at V_{dd}=1.5 V (Figure 4.8), the minimum SPP is around V_t=0.2 V.

The rate at which the SPP increases as V_t increases or decreases is different for different voltage supplies. For rising values of V_t, SPP rises slowly at higher voltages and quicker at lower voltages. For decreasing values of V_t, SPP rises slower at lower voltages than at the higher voltages. Both propagation delay and power consumption are dependent upon the voltage supply, V_{dd}, and the transistor currents. An increase in V_{dd} decreases delay and increases power. A decrease in V_{dd} increases delay and decreases power. As mentioned previously, the current in a transistor, which affects delay and power, is directly proportional to the difference between V_{dd} and V_t.

In the previous section, one saw that all of the designs ran faster at higher voltages. However, when the power is included, the best design is not the fastest. Power

consumption is much greater at higher voltages. The smallest values of SPP occur at lower supply voltages. For example, the lowest SPP at V_{dd}=3.0 V is 15.7 pJ (Table 4.1) while the lowest SPP at V_{dd}=1.5 V is 8.3 pJ (Table 4.4).

The device count is an important factor that affects the power consumption of a circuit since it contributes directly toward the overall capacitive load. Table 4.5 contains the total MOSFET count for each of the designs. Also, Table 4.6 lists the effective transistor count by taking into account that some designs have various transistor sizes. Designs that use more transistors consume more power. The BK Adder has fewer MOSFET devices than Wei's Adder and the Recursive Doubling adder. Remember from Chapter 3 that the BK Adder uses the least amount of nodes in the prefix carry

Adder Design	Total Number of Transistors
BK	1930
Wei	2342
Recursive Doubling	2358

Table 4.5 Actual MOSFET Count

Adder Design	Effective Number of Transistors
BK (r=2)	2181
BK (r=3)	2043
Wei	2389
Recursive Doubling	2358

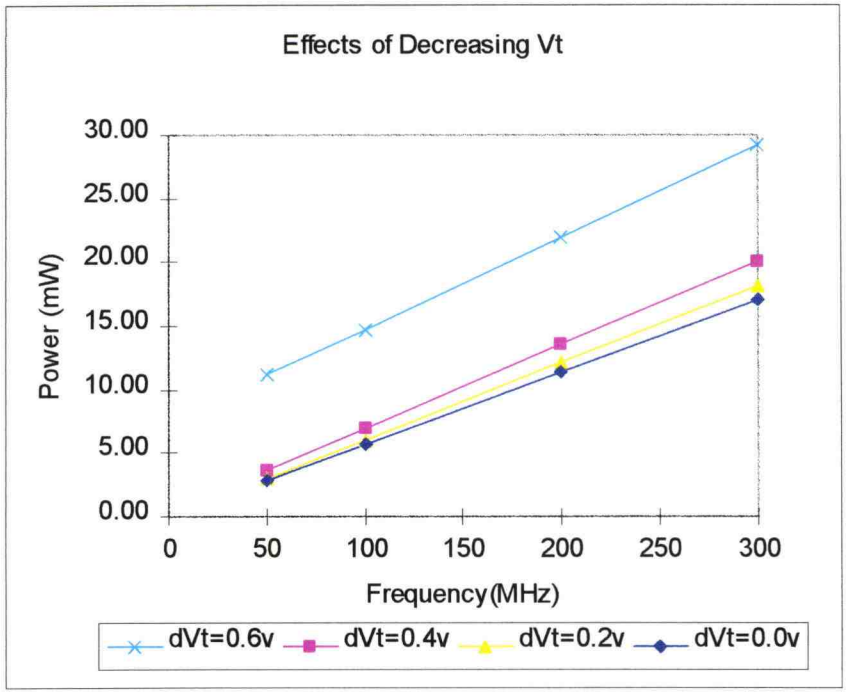
Table 4.6 Effective MOSFET Count

computation. Although the Recursive Doubling Adder has more nodes in its design, the multi-stage drivers make the effective MOSFET count in Wei's Adder greater than the effective MOSFET count for the Recursive Doubling Adder.

The total number of transistors were counted for 32-bit adders. The size of n-bit adders can be predicted by deriving equations for the total number of nodes, $S(n)$, in the carry look-ahead portion of the adder. The sizes of 4-bit, 8-bit, 16-bit and 32-bit adders were used to derive approximate equations. $S_{BK}(n) \approx (n/2)\log_2 n$ for BK Adders, and $S_{RD}(n) \approx n\log_2(n/2)$ for Recursive Doubling Adders. It is difficult to predict the size of an n-bit Wei's Adder, $S_{Wei}(n)$, because it is designed from dynamic programming[3]. $S_{Wei}(n)$ is close to $S_{BK}(n)$ for small n, and it increases towards $S_{RD}(n)$ for larger n.

4.4 Power Leakage

For the BK Adder, where the ratio between the load and driving node is two, simulations with different input frequencies were run. Here, V_{dd} was constant at 3 V and dV_t had values of 0.0 V, 0.2 V, 0.4 V and 0.6 V. The power leakage data is shown in Figure 4.9 and Table 4.7. The two far right columns in Table 4.11 contain power consumption values at f=0 MHz that were extrapolated from the data at the other frequencies. The calculated values are estimated power consumption due to current leakage (see Section 2.3).



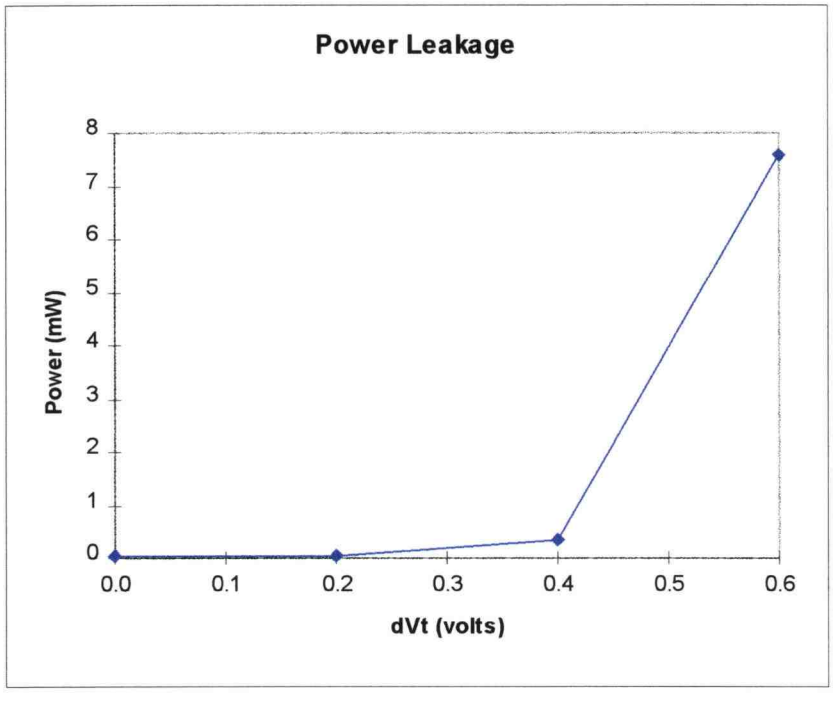


Figure 4.9 Power Leakage Effects of Decreasing V_t

	dV_t = 0.0		dV_t = 0.2		dV_t = 0.4		dV_t = 0.6		Power Leakage	
Freq.	I_{vdd}	Power	I_{vdd}	Power	I_{vdd}	Power	I_{vdd}	Power	dV_t	Power
50	1.0	2.9	1.0	3.1	1.2	3.6	3.7	11.2	0.0	0.04
100	1.9	5.7	2.0	6.1	2.3	6.9	4.9	14.8	0.2	0.04
200	3.8	11.4	4.1	12.1	4.5	13.5	7.3	22.0	0.4	0.33
300	5.7	17.1	6.1	18.1	6.7	20.1	9.7	29.2	0.6	7.58
* Freq. (MHz), I_{vdd} (mA), Power (mW), dV_t (Volts)										

Table 4.7 Power Leakage Effects of Decreasing V_t

The top graph in Figure 4.9 shows what happens when one decreases the threshold voltage. There is little change in power consumption when V_t is decreased by 0.4 V. Yet, when dV_t=0.6 V (a decrease of 0.6 V), power consumption increases significantly by about 9 mW for each particular frequency. By extending the lines to the 0 MHz axis, one can get the excess power consumption that is not caused by transistor switching. The lower part of Figure 4.9 graphically shows the extrapolated values. There is very little power leakage until V_t is close to zero.

4.5 Interconnect Wires

The simulations did not take interconnect delays into account. However, the total number of interconnects between CLA nodes are totaled in Table 4.8. Also, a spreadsheet program was used to determine the trend for *n*-bit adders based on 4-bit, 8-bit, 16-bit, and 32-bit adders. The BK Adder requires fewer interconnecting wires than Wei's Adder and the Recursive Doubling Adder. Also note that interconnect affects the Recursive

Doubling Adder more since the average wire length in the Recursive Doubling Adder is greater than in the other two adders

	BK	Wei	Recursive Doubling
32-bit	243	416	363
n-bit	$0.34n^{4.02}$	$0.24n^{4.57}$	$0.27n^{4.45}$

Table 4.8 Total Number of Interconnects Between CLA Nodes

4.6 Experimental Results Summary

In this chapter, the variations in the HSPICE simulations run on the designs described in Chapter 3 have been explained. Section 4.2 presented the experimental data for propagation delay. For most variations, the Recursive Doubling Adder was the fastest. Section 4.3 presented the experimental data for power consumption as the speed-power product (SPP). Except for Wei's Adder, all of the adders had similar SPP values when the parameters were varied. The SPP of Wei's Adder was higher than the other adders. Also in Section 4.3, we found that the BK Adder (r=2) had the lowest effective MOSFET transistor count and Wei's Adder had the highest effective MOSFET transistor count. In Section 4.4, it was seen that the power leakage is not significant until V_t is close to 0 V (within 0.1 V). Finally, the trend of wires interconnecting CLA nodes for the designs were compared. The BK Adder was found to have the fewest number of interconnects for an *n*-bit adder.

5. CONCLUSIONS

Conclusions reached are presented below. Section 5.1 provides a summary of the thesis. It includes the simulated designs and results after simulations, where different parameters were adjusted. Following the summary, Section 5.2 presents ideas for further research on this subject.

5.1 Summary

In the preceding chapters, three different adder designs were presented as well as the results of simulating the designs under several different voltage supplies, threshold voltages, and frequencies.

For all of the adder designs, the smallest propagation delay occurred with V_{dd}=3.0 V, while the lowest SPP occurred with V_{dd}=1.5 V. The Recursive Doubling Adder was always the fastest for any specific set of parameters.

Considering the speed-power product, the Recursive Doubling Adder and the BK Adders were nearly equal, and they were also all lower than Wei's Adder. Each set of data had a minimum SPP. The threshold voltage at which the minimum SPP occurred increased as V_{dd} was increased ranging from 0.2 Volts to 0.4 Volts. Also, the Brent-Kung Adders were the smallest in terms of the total number of MOSFET devices.

When V_t is decreased, power consumption increases. The power is linearly dependent on frequency. The increase in power is not significant until dV_t=0.6 V (V_t is close to zero). The power-frequency data provided the power leakage in each circuit. The power leakage difference between dV_t=0.0 V and dV_t=0.4 V is insignificant relative to the 7.6 mW of leakage that occurred at dV_t=0.6 V.

Looking at each of the results for the various parameters, an adder suitable to particular criteria may be chosen. If one is looking for just a fast adder, the Recursive Doubling Adder is best. However, considering size and area, the best alternative would be the BK Adder (r=2). The optimum speed-power product occurs at a supply voltage of 1.5 V and a threshold voltage of 0.2 V.

5.2 Future Research

The design approach for reducing the effects of fan-out can be applied to other circuits besides an adder. There are portions of a microprocessor where it may also be more suitable to size the driving gates or cells accordingly instead of placing multi-stage drivers. The need to use greater depth and include multi-stage drivers is not necessary when the design approach is used with the BK Adder. The threshold value must also be varied to get the optimum efficiency of a circuit if V_{dd} changes. Further studies should include the RC delays caused by interconnects.

REFERENCES

1. R. E. Ladner and M. J. Fischer, "Parallel Prefix Computation," *Journal of the Association for Computing Machinery*, vol. 27, pp. 831-838, October, 1980.

2. R. P. Brent and H. T. Kung, "A Regular Layout for Parallel Adders," *IEEE Transactions on Computers*, vol. C-31, pp. 260-264, March, 1982.

3. B. W. Y. Wei, C. Thompson, and Y. Chen, "Time-Optimal Design of a CMOS Adder," U. C. Berkeley, Dept. of EE and CS, CS Division, August, 1985.

4. I. Koren, *Computer Arithmetic Algorithms*: Englewood Cliffs: Prentice Hall, 1993.

5. J. M. Rabaey, *Digital Integrated Circuits: A Design Perspective*: Upper Saddle River: Prentice Hall, 1996.

6. *Star-HSPICE User's Manual Volume I Simulation and Analysis*: Meta-Software, Inc., February, 1996.

7. O. McSorley, "High Speed Arithmetic in Binary Computers," *IRE Proceedings*, vol. 49, pp. 67-91, 1961.

8. R. F. Pierret, *Field Effect Devices*, Addison-Wesley Publishing Company, 1990.

APPENDICES

APPENDIX A HSPICE Command File

```
* 32-bit Adder using Brent-Kung *
* Parag Shah *

.include "/nfs/stak/u1/s/shahp/mentor/class/ECE503/include/level3.param"
.include "/nfs/stak/u1/s/shahp/mentor/class/ECE503/brentKung/adder5.nl"
.include "/nfs/stak/u1/s/shahp/mentor/class/ECE503/include/200mhz2500ns30v.wav"

* increase Vp to get better speed
* decrease Vn to get better speed
Vp pwell 0 0
Vn nwell 0 Vd
VDD VDD 0 Vd DC
.GLOBAL VDD pwell nwell

.param Vd=3
.param dVt=0.0

.tran 1n 2500n
.meas tran avg_power   avg power
.meas tran avg_ivdd    avg i(vdd)
.meas tran avg_ipwell  avg i(vp)
.meas tran avg_inwell  avg i(vn)

.probe v(cout)
.plot v(cout)
.option post ingold=2 probe

xadder A0 A1 A10 A11 A12 A13 A14 A15 A16 A17 A18 A19 A2
+ A20 A21 A22 A23 A24 A25 A26 A27 A28 A29 A3 A30 A31
+ A4 A5 A6 A7 A8 A9 B0 B1 B10 B11 B12 B13 B14 B15
+ B16 B17 B18 B19 B2 B20 B21 B22 B23 B24 B25 B26 B27
+ B28 B29 B3 B30 B31 B4 B5 B6 B7 B8 B9 CIN COUT SUM0
+ SUM1 SUM10 SUM11 SUM12 SUM13 SUM14 SUM15 SUM16 SUM17
+ SUM18 SUM19 SUM2 SUM20 SUM21 SUM22 SUM23 SUM24 SUM25
+ SUM26 SUM27 SUM28 SUM29 SUM3 SUM30 SUM31 SUM4 SUM5 SUM6
+ SUM7 SUM8 SUM9 adder1a

.END
```

APPENDIX B Level 3 HSPICE Parameters

```
.MODEL N NMOS LEVEL=3 PHI=0.700000 TOX=9.5000E-09 XJ=0.200000U TPG=1
+ VTO=0.6674 DELTA=1.4270E+00 LD=6.3300E-08 KP=1.7146E-04
+ UO=471.7 THETA=1.6690E-01 RSH=3.3470E+01 GAMMA=0.5219
+ NSUB=1.0840E+17 NFS=5.9080E+11 VMAX=2.2650E+05 ETA=2.0550E-02
+ KAPPA=2.1270E-01 CGDO=9.0000E-11 CGSO=9.0000E-11
+ CGBO=3.6007E-10 CJ=5.69E-04 MJ=0.661 CJSW=2.00E-11
+ MJSW=0.609 PB=0.99
* Weff = Wdrawn - Delta_W
* The suggested Delta_W is 3.3260E-07

.MODEL P PMOS LEVEL=3 PHI=0.700000 TOX=9.5000E-09 XJ=0.200000U TPG=-1
+ VTO=-0.9188 DELTA=3.5350E-01 LD=7.8860E-08 KP=3.8312E-05
+ UO=105.4 THETA=3.3670E-02 RSH=1.6950E+01 GAMMA=0.7396
+ NSUB=2.1770E+17 NFS=5.9080E+11 VMAX=1.5650E+05 ETA=1.7260E-02
+ KAPPA=8.8780E+00 CGDO=9.0000E-11 CGSO=9.0000E-11
+ CGBO=3.6237E-10 CJ=9.19E-04 MJ=0.321 CJSW=4.60E-10
+ MJSW=0.100 PB=0.42
* Weff = Wdrawn - Delta_W
* The suggested Delta_W is 3.3680E-07
```

APPENDIX C HSPICE Adder Netlist

```
* Netlist for Brent-Kung 32-bit adder *
* For transistor size variations r *
* Parag Shah
* March 17, 1998

.param L=0.5u
.param Wmin=1u
.param r=0.5
.param k=0
.param Cw='Wmin*k'

.param Winv=Wmin
.param Wnand=Wmin
.param Wnor=Wmin
.param Wxor=Wmin
.param Wxnor=Wmin

.param WGPa=Wmin
.param WGPbara=Wmin
.param WGPb='(9*Wmin+9*Cw)*r'
.param WGPbarb='(4*Wmin+4*Cw+WGPb)*r'
.param WGPc='(2*Wmin+3*Cw+WGPbarb)*r'

.param WCa=Wmin
.param WCbara=Wmin
.param WCb='(3*Wmin+2*Cw)*r'
.param WCbarb='(17*Wmin+17*Cw+WCb)*r'
.param WCc='(9*Wmin+9*Cw+WCbarb)*r'
.param WCbarc='(5*Wmin+5*Cw+WCc)*r'
.param WCd='(3*Wmin+3*Cw+WCbarc)*r'

.SUBCKT inverter
+ IN OUT width=1u length=Wmin deltaVt=dVt
Vbn  Nin in  deltaVt
Vbp  in  Pin deltaVt
MXI_2 OUT Nin 0     pwell N w=width     l=length AS=16.0P AD=16.0P PS=12.0U
+ PD=12.0U
MXI_1 OUT Pin VDD nwell P w='2*width' l=length AS=16.0P AD=16.0P PS=12.0U
+ PD=12.0U
.ENDS * inverter *
```

```
.SUBCKT nor
+ PLBAR POUT PRBAR width=Wnor length=L deltaVt=dVt
VbAn  PLBARn PLBAR  deltaVt
VbAp  PLBAR  PLBARp deltaVt
VbBn  PRBARn PRBAR  deltaVt
VbBp  PRBAR  PRBARp deltaVt
MXI_4 POUT   PRBARp N_6 nwell P w='2*width' l=length m=1 AS=16.0P AD=16.0P
+ PS=12.0U PD=12.0U
MXI_3 N_6    PLBARp VDD nwell P w='2*width' l=length m=1 AS=16.0P AD=16.0P
+ PS=12.0U PD=12.0U
MXI_2 POUT   PRBARn GND pwell N w=width     l=length m=1 AS=16.0P AD=16.0P
+ PS=12.0U PD=12.0U
MXI_1 POUT   PLBARn GND pwell N w=width     l=length m=1 AS=16.0P AD=16.0P
+ PS=12.0U PD=12.0U
.ENDS * nor *

.SUBCKT nand
+ PL POUTBAR PR width=Wnand length=L deltaVt=dVt
VbAn  PLn    PL   deltaVt
VbAp  PL     PLp  deltaVt
VbBn  PRn    PR   deltaVt
VbBp  PR     PRp  deltaVt
MXI_4 POUTBAR PLp  VDD  nwell P w='2*width' l=length m=1 AS=16.0P AD=16.0P
+ PS=12.0U PD=12.0U
MXI_3 POUTBAR PRp  VDD  nwell P w='2*width' l=length m=1 AS=16.0P AD=16.0P
+ PS=12.0U PD=12.0U
MXI_2 POUTBAR PLn  N_10 pwell N w=width     l=length m=1 AS=16.0P AD=16.0P
+ PS=12.0U PD=12.0U
MXI_1 N_10    PRn  GND  pwell N w=width     l=length m=1 AS=16.0P AD=16.0P
+ PS=12.0U PD=12.0U
.ENDS * nand *

.SUBCKT Ga
+ GL GOUTBAR GR PL width=Wmin length=L deltaVt=dVt
VbGLn  GLn    GL   deltaVt
VbGLp  GL     GLp  deltaVt
VbGRn  GRn    GR   deltaVt
VbGRp  GR     GRp  deltaVt
VbPLn  PLn    PL   deltaVt
VbPLp  PL     PLp  deltaVt
MXI_18 GOUTBAR GLn  GND pwell N w=width     l=length m=1 AS=16.0P AD=16.0P
+ PS=12.0U PD=12.0U
MXI_17 GOUTBAR GLp  N_6 nwell P w='2*width' l=length m=1 AS=16.0P AD=16.0P
+ PS=12.0U PD=12.0U
MXI_4  N_6     GRp  VDD nwell P w='2*width' l=length m=1 AS=16.0P AD=16.0P
```

```
+ PS=12.0U PD=12.0U
MXI_3  N_6     PLp  VDD nwell P w='2*width' l=length m=1 AS=16.0P AD=16.0P
+ PS=12.0U PD=12.0U
MXI_2  GOUTBAR PLn  N_5 pwell N w=width     l=length m=1 AS=16.0P AD=16.0P
+ PS=12.0U PD=12.0U
MXI_1  N_5     GRn  GND pwell N w=width     l=length m=1 AS=16.0P AD=16.0P
+ PS=12.0U PD=12.0U
.ENDS * Ga *

.SUBCKT Gbara
+ GLBAR GOUT GRBAR PLBAR width=Wmin length=L deltaVt=dVt
VbGLn GLBARn GLBAR  deltaVt
VbGLp GLBAR  GLBARp deltaVt
VbGRn GRBARn GRBAR  deltaVt
VbGRp GRBAR  GRBARp deltaVt
VbPLn PLBARn PLBAR  deltaVt
VbPLp PLBAR  PLBARp deltaVt
MXI_6 N_10   GRBARn GND  pwell N w=width     l=length m=1 AS=16.0P AD=16.0P
+ PS=12.0U PD=12.0U
MXI_5 GOUT   GRBARp N_14 nwell P w='2*width' l=length m=1 AS=16.0P AD=16.0P
+ PS=12.0U PD=12.0U
MXI_4 N_14   PLBARp VDD  nwell P w='2*width' l=length m=1 AS=16.0P AD=16.0P
+ PS=12.0U PD=12.0U
MXI_3 GOUT   GLBARp VDD  nwell P w='2*width' l=length m=1 AS=16.0P
+ AD=16.0P PS=12.0U PD=12.0U
MXI_2 GOUT   GLBARn N_10 pwell N w=width     l=length m=1 AS=16.0P AD=16.0P
+ PS=12.0U PD=12.0U
MXI_1 N_10   PLBARn GND  pwell N w=width     l=length m=1 AS=16.0P AD=16.0P
+ PS=12.0U PD=12.0U
.ENDS * Gbara *

.SUBCKT gpgen
+ AB[0] AB[1] GP[0] GP[1]
XI_621  N_620 GP[0] inverter
XI_620  AB[1] N_620 AB[0] nor
XI_215  N_213 GP[1] inverter
XI_4  AB[1] N_213 AB[0] nand
.ENDS * gpgen *

.SUBCKT xor
+ A B OUT width=Wxor length=L deltaVt=dVt
VbAn   An   A    deltaVt
VbAp   A    Ap   deltaVt
VbBn   Bn   B    deltaVt
VbBp   B    Bp   deltaVt
```

```
MXI_8 N_11 N_203 GND pwell N w=width l=length m=1 AS=16.0P AD=16.0P
+ PS=12.0U PD=12.0U
MXI_7 N_9 Bn GND pwell N w=width l=length m=1 AS=16.0P AD=16.0P
+ PS=12.0U PD=12.0U
MXI_6 OUT N_18 N_11 pwell N w=width l=length m=1 AS=16.0P AD=16.0P
+ PS=12.0U PD=12.0U
MXI_5 OUT An N_9 pwell N w=width l=length m=1 AS=16.0P AD=16.0P
+ PS=12.0U PD=12.0U
MXI_4 OUT N_203 N_4 nwell P w='2*width' l=length m=1 AS=16.0P AD=16.0P
+ PS=12.0U PD=12.0U
MXI_3 OUT N_18 N_4 nwell P w='2*width' l=length m=1 AS=16.0P AD=16.0P
+ PS=12.0U PD=12.0U
MXI_2 N_4 Bp VDD nwell P w='2*width' l=length m=1 AS=16.0P AD=16.0P
+ PS=12.0U PD=12.0U
MXI_1 N_4 Ap VDD nwell P w='2*width' l=length m=1 AS=16.0P AD=16.0P
+ PS=12.0U PD=12.0U
XI_211 B N_203 inverter width=Wxor
XI_210 A N_18 inverter width=Wxor
.ENDS * xor *

.SUBCKT inverter2
+ IN[0] IN[1] OUT[0] OUT[1]
XI_2 IN[1] OUT[1] inverter
XI_1 IN[0] OUT[0] inverter
.ENDS * inverter2 *

.SUBCKT Ca
+ COUTBAR GPLEFT[0] GPLEFT[1] GR
XI_465 GPLEFT[1] COUTBAR GR GPLEFT[0] Ga width=WCa
.ENDS * Ca *

.SUBCKT GPbara
+ GPLEFTBAR[0] GPLEFTBAR[1] GPOUT[0] GPOUT[1] GPRIGHTBAR[0]
GPRIGHTBAR[1]
XI_213 GPLEFTBAR[1] GPOUT[1] GPRIGHTBAR[1] GPLEFTBAR[0] Gbara
+ width=WGPbara
XI_5 GPLEFTBAR[0] GPOUT[0] GPRIGHTBAR[0] nor width=WGPbara
.ENDS * GPbara *

.SUBCKT GPa
+ GPLEFT[0] GPLEFT[1] GPOUTBAR[0] GPOUTBAR[1] GPRIGHT[0] GPRIGHT[1]
XI_207 GPLEFT[0] GPOUTBAR[0] GPRIGHT[0] nand width=WGPa
XI_415 GPLEFT[1] GPOUTBAR[1] GPRIGHT[1] GPLEFT[0] Ga width=WGPa
.ENDS * GPa *
```

```
.SUBCKT GPb
+ GPLEFT[0] GPLEFT[1] GPOUTBAR[0] GPOUTBAR[1] GPRIGHT[0] GPRIGHT[1]
XI_207  GPLEFT[0] GPOUTBAR[0] GPRIGHT[0] nand width=WGPb
XI_415  GPLEFT[1] GPOUTBAR[1] GPRIGHT[1] GPLEFT[0] Ga width=WGPb
.ENDS * GPb *

.SUBCKT Cbara
+ COUT GPLEFTBAR[0] GPLEFTBAR[1] GR
XI_213  GPLEFTBAR[1] COUT GR GPLEFTBAR[0] Gbara width=WCbara
.ENDS * Cbara *

.SUBCKT Cbarb
+ COUT GPLEFTBAR[0] GPLEFTBAR[1] GR
XI_213  GPLEFTBAR[1] COUT GR GPLEFTBAR[0] Gbara width=WCbarb
.ENDS * Cbarb *

.SUBCKT GPc
+ GPLEFT[0] GPLEFT[1] GPOUTBAR[0] GPOUTBAR[1] GPRIGHT[0] GPRIGHT[1]
XI_207  GPLEFT[0] GPOUTBAR[0] GPRIGHT[0] nand width=WGPc
XI_415  GPLEFT[1] GPOUTBAR[1] GPRIGHT[1] GPLEFT[0] Ga width=WGPc
.ENDS * GPc *

.SUBCKT Cd
+ COUTBAR GPLEFT[0] GPLEFT[1] GR
XI_465  GPLEFT[1] COUTBAR GR GPLEFT[0] Ga width=WCd
.ENDS * Cd *

.SUBCKT xnor
+ A B OUT width=Wxnor length=L deltaVt=dVt
VbAn   An   A    deltaVt
VbAp   A    Ap   deltaVt
VbBn   Bn   B    deltaVt
VbBp   B    Bp   deltaVt
XI_26  B    N_19  inverter width=Wxnor
MXI_20 N_8  N_19  GND  pwell N w=width     l=length m=1 AS=16.0P AD=16.0P
+ PS=12.0U PD=12.0U
MXI_18 OUT  N_204 N_10 pwell N w=width     l=length m=1 AS=16.0P AD=16.0P
+ PS=12.0U PD=12.0U
MXI_17 OUT  An    N_8  pwell N w=width     l=length m=1 AS=16.0P AD=16.0P
+ PS=12.0U PD=12.0U
MXI_21 N_10 Bn    GND  pwell N w=width     l=length m=1 AS=16.0P AD=16.0P
+ PS=12.0U PD=12.0U
MXI_15 OUT  Bp    N_2  nwell P w='2*width' l=length m=1 AS=16.0P AD=16.0P
+ PS=12.0U PD=12.0U
MXI_14 OUT  N_204 N_2  nwell P w='2*width' l=length m=1 AS=16.0P AD=16.0P
```

```
+ PS=12.0U PD=12.0U
MXI_12 N_2  N_19  VDD  nwell P w='2*width' l=length m=1 AS=16.0P AD=16.0P
+ PS=12.0U PD=12.0U
MXI_11 N_2  Ap    VDD  nwell P w='2*width' l=length m=1 AS=16.0P AD=16.0P
+ PS=12.0U PD=12.0U
XI_9   A    N_204 inverter
.ENDS * xnor *

.SUBCKT Cb
+ COUTBAR GPLEFT[0] GPLEFT[1] GR
XI_465  GPLEFT[1] COUTBAR GR GPLEFT[0] Ga width=WCb
.ENDS * Cb *

.SUBCKT Cc
+ COUTBAR GPLEFT[0] GPLEFT[1] GR
XI_465  GPLEFT[1] COUTBAR GR GPLEFT[0] Ga width=WCc
.ENDS * Cc *

.SUBCKT GPbarb
+ GPLEFTBAR[0] GPLEFTBAR[1] GPOUT[0] GPOUT[1] GPRIGHTBAR[0]
+ GPRIGHTBAR[1]
XI_213  GPLEFTBAR[1] GPOUT[1] GPRIGHTBAR[1] GPLEFTBAR[0] Gbara
+ width=WGPbarb
XI_5  GPLEFTBAR[0] GPOUT[0] GPRIGHTBAR[0] nor width=WGPbarb
.ENDS * GPbarb *

.SUBCKT Cbarc
+ COUT GPLEFTBAR[0] GPLEFTBAR[1] GR
XI_213  GPLEFTBAR[1] COUT GR GPLEFTBAR[0] Gbara width=WCbarc
.ENDS * Cbarc *

.SUBCKT adder1a
+ A[0] A[1] A[10] A[11] A[12] A[13] A[14] A[15] A[16] A[17] A[18] A[19] A[2]
+ A[20] A[21] A[22] A[23] A[24] A[25] A[26] A[27] A[28] A[29] A[3] A[30] A[31]
+ A[4] A[5] A[6] A[7] A[8] A[9] B[0] B[1] B[10] B[11] B[12] B[13] B[14] B[15]
+ B[16] B[17] B[18] B[19] B[2] B[20] B[21] B[22] B[23] B[24] B[25] B[26] B[27]
+ B[28] B[29] B[3] B[30] B[31] B[4] B[5] B[6] B[7] B[8] B[9] CIN COUT SUM[0]
+ SUM[1] SUM[10] SUM[11] SUM[12] SUM[13] SUM[14] SUM[15] SUM[16]
+ SUM[17] SUM[18] SUM[19] SUM[2] SUM[20] SUM[21] SUM[22] SUM[23]
+ SUM[24] SUM[25] SUM[26] SUM[27] SUM[28] SUM[29] SUM[3] SUM[30]
+ SUM[31] SUM[4] SUM[5] SUM[6] SUM[7] SUM[8] SUM[9]
XI_4895  B[3] A[3] N_4191[0] N_4191[1] gpgen
XI_4898  A[3] B[3] N_4240 xor
XI_4894  B[5] A[5] N_3610[0] N_3610[1] gpgen
XI_4892  B[7] A[7] N_4095[0] N_4095[1] gpgen
```

XI_4215 N_3633[0] N_3633[1] N_4098[0] N_4098[1] inverter2
XI_3885 N_4154 N_4191[0] N_4191[1] N_4170 Ca
XI_3872 N_3640[0] N_3640[1] N_3665[0] N_3665[1] N_3641[0] N_3641[1] GPbara
XI_3858 N_4199[0] N_4199[1] N_3681[0] N_3681[1] N_4095[0] N_4095[1] GPa
XI_3870 N_3637[0] N_3637[1] N_3666[0] N_3666[1] N_3681[0] N_3681[1] GPbara
XI_3895 N_3673[0] N_3673[1] N_3696[0] N_3696[1] N_3746[0] N_3746[1] GPb
XI_3894 N_3671[0] N_3671[1] N_4079[0] N_4079[1] N_3746[0] N_3746[1] GPa
XI_3906 N_3699[0] N_3699[1] N_3764[0] N_3764[1] N_3696[0] N_3696[1] GPbara
XI_4896 B[26] A[26] N_4215[0] N_4215[1] gpgen
XI_4878 B[24] A[24] N_4213[0] N_4213[1] gpgen
XI_4877 B[22] A[22] N_4211[0] N_4211[1] gpgen
XI_4876 B[20] A[20] N_4209[0] N_4209[1] gpgen
XI_4167 A[30] B[30] N_3979 xor
XI_4176 A[21] B[21] N_4006 xor
XI_4180 A[17] B[17] N_4018 xor
XI_4174 A[23] B[23] N_4000 xor
XI_4181 A[16] B[16] N_4021 xor
XI_4182 A[15] B[15] N_4024 xor
XI_4179 A[18] B[18] N_4015 xor
XI_4875 B[18] A[18] N_4207[0] N_4207[1] gpgen
XI_4874 B[14] A[14] N_4205[0] N_4205[1] gpgen
XI_4873 B[12] A[12] N_4203[0] N_4203[1] gpgen
XI_4881 B[29] A[29] N_4222[0] N_4222[1] gpgen
XI_4879 B[31] A[31] N_4221[0] N_4221[1] gpgen
XI_4880 B[30] A[30] N_4219[0] N_4219[1] gpgen
XI_4897 B[28] A[28] N_4217[0] N_4217[1] gpgen
XI_3953 N_4116 N_4078[0] N_4078[1] N_4186 Ca
XI_3952 N_4118 N_4080[0] N_4080[1] N_4186 Ca
XI_3951 N_4120 N_4082[0] N_4082[1] N_4186 Ca
XI_3950 N_4122 N_4084[0] N_4084[1] N_4186 Ca
XI_3900 N_4178 N_4091[0] N_4091[1] N_4148 Cbara
XI_3898 N_4174 N_3681[0] N_3681[1] N_4148 Cbara
XI_3897 N_4172 N_4096[0] N_4096[1] N_4148 Cbara
XI_4872 B[10] A[10] N_4201[0] N_4201[1] gpgen
XI_4871 B[8] A[8] N_4199[0] N_4199[1] gpgen
XI_4870 B[6] A[6] N_4197[0] N_4197[1] gpgen
XI_4867 B[0] A[0] N_4188[0] N_4188[1] gpgen
XI_3960 N_4102 N_3768[0] N_3768[1] N_4186 Ca
XI_4194 A[2] B[2] N_4063 xor
XI_4901 B[16] A[16] N_4247[0] N_4247[1] gpgen
XI_4900 A[11] B[11] N_4245 xor
XI_4899 B[1] A[1] N_3606[0] N_3606[1] gpgen
XI_4213 N_4095[0] N_4095[1] N_4096[0] N_4096[1] inverter2
XI_4212 N_3614[0] N_3614[1] N_4094[0] N_4094[1] inverter2
XI_4211 N_4092[0] N_4092[1] N_4093[0] N_4093[1] inverter2

XI_4210 N_3666[0] N_3666[1] N_4091[0] N_4091[1] inverter2
XI_3935 COUT N_4165[0] N_4165[1] N_4100 Cbara
XI_3889 N_4090[0] N_4090[1] N_3689[0] N_3689[1] N_3666[0] N_3666[1] GPa
XI_3869 N_4097[0] N_4097[1] N_3660[0] N_3660[1] N_3633[0] N_3633[1] GPbara
XI_3949 N_4124 N_3746[0] N_3746[1] N_4186 Ca
XI_3888 N_4234[0] N_4234[1] N_3687[0] N_3687[1] N_3666[0] N_3666[1] GPa
XI_3873 N_4089[0] N_4089[1] N_3664[0] N_3664[1] N_3641[0] N_3641[1] GPbara
XI_3860 N_4203[0] N_4203[1] N_3641[0] N_3641[1] N_4234[0] N_4234[1] GPa
XI_3859 N_4201[0] N_4201[1] N_3637[0] N_3637[1] N_3614[0] N_3614[1] GPa
XI_3905 N_3698[0] N_3698[1] N_3766[0] N_3766[1] N_3696[0] N_3696[1] GPbara
XI_3904 N_3697[0] N_3697[1] N_3768[0] N_3768[1] N_3696[0] N_3696[1] GPbara
XI_3903 N_3695[0] N_3695[1] N_3770[0] N_3770[1] N_3696[0] N_3696[1] GPbara
XI_3865 N_4211[0] N_4211[1] N_3646[0] N_3646[1] N_3624[0] N_3624[1] GPa
XI_4891 B[9] A[9] N_3614[0] N_3614[1] gpgen
XI_4890 B[11] A[11] N_4234[0] N_4234[1] gpgen
XI_4889 B[13] A[13] N_3618[0] N_3618[1] gpgen
XI_4888 B[15] A[15] N_4235[0] N_4235[1] gpgen
XI_4887 B[17] A[17] N_3621[0] N_3621[1] gpgen
XI_4886 B[19] A[19] N_4236[0] N_4236[1] gpgen
XI_4885 B[21] A[21] N_3624[0] N_3624[1] gpgen
XI_4884 B[23] A[23] N_4076[0] N_4076[1] gpgen
XI_4883 B[25] A[25] N_3628[0] N_3628[1] gpgen
XI_4882 B[27] A[27] N_4190[0] N_4190[1] gpgen
XI_3957 N_4108 N_3762[0] N_3762[1] N_4186 Ca
XI_3956 N_4110 N_3760[0] N_3760[1] N_4186 Ca
XI_3955 N_4112 N_3758[0] N_3758[1] N_4186 Ca
XI_3954 N_4114 N_3756[0] N_3756[1] N_4186 Ca
XI_3887 N_4150 N_3660[0] N_3660[1] N_4170 Ca
XI_4170 A[27] B[27] N_3988 xor
XI_4175 A[22] B[22] N_4003 xor
XI_4171 A[26] B[26] N_3991 xor
XI_4178 A[19] B[19] N_4012 xor
XI_4172 A[25] B[25] N_3994 xor
XI_4177 A[20] B[20] N_4009 xor
XI_4173 A[24] B[24] N_3997 xor
XI_4869 B[4] A[4] N_4195[0] N_4195[1] gpgen
XI_4205 N_3624[0] N_3624[1] N_4085[0] N_4085[1] inverter2
XI_4204 N_4083[0] N_4083[1] N_4084[0] N_4084[1] inverter2
XI_4203 N_4081[0] N_4081[1] N_4082[0] N_4082[1] inverter2
XI_4241 A[1] B[1] N_4066 xor
XI_4240 A[0] B[0] N_4069 xor
XI_4214 N_3610[0] N_3610[1] N_4097[0] N_4097[1] inverter2
XI_5105 N_4170 N_4240 SUM[3] xor
XI_3948 N_4126 N_3744[0] N_3744[1] N_4186 Ca
XI_3947 N_4128 N_4088[0] N_4088[1] N_4186 Ca

XI_3946 N_4130 N_4235[0] N_4235[1] N_4186 Ca
XI_3909 N_4184 N_3691[0] N_3691[1] N_4148 Cbara
XI_3902 N_4182 N_3689[0] N_3689[1] N_4148 Cbara
XI_3907 N_4072[0] N_4072[1] N_3762[0] N_3762[1] N_3696[0] N_3696[1] GPbara
XI_3901 N_4180 N_3687[0] N_3687[1] N_4148 Cbara
XI_3942 N_4213[0] N_4213[1] N_3702[0] N_3702[1] N_4076[0] N_4076[1] GPa
XI_3941 N_4217[0] N_4217[1] N_3653[0] N_3653[1] N_4190[0] N_4190[1] GPa
XI_4163 N_3621[0] N_3621[1] N_3973[0] N_3973[1] inverter2
XI_3880 N_4168 N_4099[0] N_4099[1] N_4160 Cbara
XI_3864 N_4209[0] N_4209[1] N_3647[0] N_3647[1] N_4236[0] N_4236[1] GPa
XI_3910 N_4186 N_3693[0] N_3693[1] N_4148 Cbarb
XI_3908 N_4074[0] N_4074[1] N_3760[0] N_3760[1] N_3696[0] N_3696[1] GPbara
XI_3959 N_4104 N_3766[0] N_3766[1] N_4186 Ca
XI_3958 N_4106 N_3764[0] N_3764[1] N_4186 Ca
XI_5104 N_4168 N_4063 SUM[2] xor
XI_3892 N_4236[0] N_4236[1] N_4083[0] N_4083[1] N_3746[0] N_3746[1] GPa
XI_3891 N_3665[0] N_3665[1] N_3693[0] N_3693[1] N_3666[0] N_3666[1] GPa
XI_3857 N_4197[0] N_4197[1] N_3632[0] N_3632[1] N_3610[0] N_3610[1] GPa
XI_4199 N_3628[0] N_3628[1] N_4075[0] N_4075[1] inverter2
XI_4198 N_4073[0] N_4073[1] N_4074[0] N_4074[1] inverter2
XI_4197 N_3678[0] N_3678[1] N_4072[0] N_4072[1] inverter2
XI_4196 N_3653[0] N_3653[1] N_4071[0] N_4071[1] inverter2
XI_4195 N_4222[0] N_4222[1] N_4070[0] N_4070[1] inverter2
XI_3878 N_3649[0] N_3649[1] N_3678[0] N_3678[1] N_3702[0] N_3702[1] GPbara
XI_3875 N_3973[0] N_3973[1] N_3744[0] N_3744[1] N_4087[0] N_4087[1] GPbara
XI_4251 N_3606[0] N_3606[1] N_4099[0] N_4099[1] inverter2
XI_3861 N_4205[0] N_4205[1] N_3640[0] N_3640[1] N_3618[0] N_3618[1] GPa
XI_3862 N_4247[0] N_4247[1] N_4087[0] N_4087[1] N_4235[0] N_4235[1] GPc
XI_3937 N_3702[0] N_3702[1] N_3758[0] N_3758[1] N_3696[0] N_3696[1] GPbara
XI_3868 N_3632[0] N_3632[1] N_3661[0] N_3661[1] N_3633[0] N_3633[1] GPbara
XI_4192 A[5] B[5] N_4054 xor
XI_4190 A[7] B[7] N_4048 xor
XI_3856 N_4195[0] N_4195[1] N_3633[0] N_3633[1] N_4191[0] N_4191[1] GPa
XI_3884 N_4160 N_4188[0] N_4188[1] CIN Cd
XI_3863 N_4207[0] N_4207[1] N_3643[0] N_3643[1] N_3621[0] N_3621[1] GPa
XI_3945 N_4219[0] N_4219[1] N_3652[0] N_3652[1] N_4222[0] N_4222[1] GPa
XI_3879 N_4075[0] N_4075[1] N_4073[0] N_4073[1] N_3702[0] N_3702[1] GPbara
XI_3944 N_3676[0] N_3676[1] N_3697[0] N_3697[1] N_3678[0] N_3678[1] GPa
XI_4209 N_3641[0] N_3641[1] N_4090[0] N_4090[1] inverter2
XI_4208 N_3618[0] N_3618[1] N_4089[0] N_4089[1] inverter2
XI_4207 N_4087[0] N_4087[1] N_4088[0] N_4088[1] inverter2
XI_4206 N_3647[0] N_3647[1] N_4086[0] N_4086[1] inverter2
XI_4193 A[4] B[4] N_4057 xor
XI_4230 N_4128 N_4018 SUM[17] xnor
XI_3871 N_4094[0] N_4094[1] N_4092[0] N_4092[1] N_3681[0] N_3681[1] GPbara

XI_4453 N_4221[0] N_4221[1] N_4165[0] N_4165[1] inverter2
XI_4229 N_4126 N_4015 SUM[18] xnor
XI_4228 N_4124 N_4012 SUM[19] xnor
XI_4227 N_4122 N_4009 SUM[20] xnor
XI_4226 N_4120 N_4006 SUM[21] xnor
XI_4225 N_4118 N_4003 SUM[22] xnor
XI_4224 N_4116 N_4000 SUM[23] xnor
XI_4223 N_4114 N_3997 SUM[24] xnor
XI_4222 N_4112 N_3994 SUM[25] xnor
XI_4221 N_4110 N_3991 SUM[26] xnor
XI_4220 N_4108 N_3988 SUM[27] xnor
XI_4219 N_4106 N_3985 SUM[28] xnor
XI_4218 N_4104 N_3982 SUM[29] xnor
XI_5113 N_4186 N_4024 SUM[15] xor
XI_5112 N_4184 N_4027 SUM[14] xor
XI_5111 N_4182 N_4030 SUM[13] xor
XI_4185 A[12] B[12] N_4033 xor
XI_4187 A[10] B[10] N_4039 xor
XI_4184 A[13] B[13] N_4030 xor
XI_4868 B[2] A[2] N_4193[0] N_4193[1] gpgen
XI_3899 N_4176 N_4093[0] N_4093[1] N_4148 Cbara
XI_3886 N_4152 N_4098[0] N_4098[1] N_4170 Ca
XI_3936 N_4190[0] N_4190[1] N_3699[0] N_3699[1] N_3678[0] N_3678[1] GPa
XI_3893 N_4086[0] N_4086[1] N_4081[0] N_4081[1] N_3746[0] N_3746[1] GPa
XI_3877 N_4085[0] N_4085[1] N_3671[0] N_3671[1] N_3647[0] N_3647[1] GPbara
XI_5106 N_4172 N_4045 SUM[8] xor
XI_4188 A[9] B[9] N_4042 xor
XI_4189 A[8] B[8] N_4045 xor
XI_3940 N_4215[0] N_4215[1] N_3649[0] N_3649[1] N_3628[0] N_3628[1] GPa
XI_3939 N_4071[0] N_4071[1] N_3698[0] N_3698[1] N_3678[0] N_3678[1] GPa
XI_3882 N_3652[0] N_3652[1] N_3677[0] N_3677[1] N_3653[0] N_3653[1] GPbara
XI_3883 N_4070[0] N_4070[1] N_3676[0] N_3676[1] N_3653[0] N_3653[1] GPbara
XI_5110 N_4180 N_4033 SUM[12] xor
XI_5109 N_4178 N_4245 SUM[11] xor
XI_5108 N_4176 N_4039 SUM[10] xor
XI_3943 N_3677[0] N_3677[1] N_3695[0] N_3695[1] N_3678[0] N_3678[1] GPa
XI_4217 N_4102 N_3979 SUM[30] xnor
XI_4216 N_4100 N_3976 SUM[31] xnor
XI_4169 A[28] B[28] N_3985 xor
XI_4168 A[29] B[29] N_3982 xor
XI_5107 N_4174 N_4042 SUM[9] xor
XI_3855 N_4193[0] N_4193[1] N_3607[0] N_3607[1] N_3606[0] N_3606[1] GPa
XI_3876 N_3646[0] N_3646[1] N_3673[0] N_3673[1] N_3647[0] N_3647[1] GPbara
XI_3961 N_4100 N_3770[0] N_3770[1] N_4186 Cb
XI_4183 A[14] B[14] N_4027 xor

```
XI_4166  A[31] B[31] N_3976 xor
XI_3896  N_4148 N_3661[0] N_3661[1] N_4170 Cc
XI_4191  A[6] B[6] N_4051 xor
XI_4249  N_4160 N_4066 SUM[1] xnor
XI_5103  CIN N_4069 SUM[0] xor
XI_4246  N_4154 N_4057 SUM[4] xnor
XI_4245  N_4152 N_4054 SUM[5] xnor
XI_4244  N_4150 N_4051 SUM[6] xnor
XI_4243  N_4148 N_4048 SUM[7] xnor
XI_3874  N_3643[0] N_3643[1] N_3746[0] N_3746[1] N_4087[0] N_4087[1] GPbarb
XI_4231  N_4130 N_4021 SUM[16] xnor
XI_4202  N_4079[0] N_4079[1] N_4080[0] N_4080[1] inverter2
XI_4201  N_3696[0] N_3696[1] N_4078[0] N_4078[1] inverter2
XI_4200  N_4076[0] N_4076[1] N_4077[0] N_4077[1] inverter2
XI_3890  N_3664[0] N_3664[1] N_3691[0] N_3691[1] N_3666[0] N_3666[1] GPa
XI_3881  N_4170 N_3607[0] N_3607[1] N_4160 Cbarc
XI_3938  N_4077[0] N_4077[1] N_3756[0] N_3756[1] N_3696[0] N_3696[1] GPbara
.ENDS * adder1a *
```

APPENDIX D Random Input Generator

```
#include <stdio.h>

/* genclk.c generates clock waves for simulations
   Usuage:  genclk freq(Mhz) edgerate(ns) maxsimtime(ns)
outputfile_seed_name
            VCC(volts) deadtime(ns) start(1/0) maxwave(#)
*/

main(argc,argv)
int   argc;
char  *argv[];
{
  float t1,tmax,v1,v2,freq,edge,vcc,deadtime,startv,maxwav;
  char *outfileseed;
  int i,j,k;
  FILE *fp;
  char filename[100];

  if (argc != 9)
  {
      printf(" number of arguments incorrect\n");
      printf(" Usuage:  genclk frequency(Mhz) edgerate(ns)
          maxsimtime(ns)");
      printf(" outputfile_seed_name vcc(volts) deadtime(ns)
          start(1/0)");
      printf(" NumberOfWaves(n)\n");
      exit(-1);
  }

  sscanf(argv[1], "%f", &freq);
  sscanf(argv[2], "%f", &edge);
  sscanf(argv[3], "%f", &tmax);
  sscanf(argv[5], "%f", &vcc);
  sscanf(argv[6], "%f", &deadtime);
  sscanf(argv[7], "%f", &startv);
  sscanf(argv[8], "%f", &maxwav);

  outfileseed = argv[4];

  sprintf(filename,"%s.wav",outfileseed);
  fp = fopen(filename,"w");

  for (i=0; i<=maxwav; i++)
  {
     if (startv == 1)
          v2 = vcc;
```

```
        else
            v2 = 0;

        srand(time(0));

        if (i < maxwav/2)
           fprintf(fp,"Va%d  a%d  0  pwl 0nS %gV ",i,i,v2);
        else if (i < maxwav)
        {
            j=i-maxwav/2;
            fprintf(fp,"Vb%d  b%d  0  pwl 0nS %gV ",j,j,v2);
        }
        else
            fprintf(fp,"Vcin  cin  0  pwl 0nS %gV ",v2);

        for(t1=deadtime;t1<tmax;t1=t1+(1/freq)*1000)
        {
            v1 = (rand()%2) * vcc;
            if ((v2 != v1) && (t1 >= deadtime))
            {
                 k++;
                 if ( k==4 )
                 {
                      fprintf(fp, "\n + ");
                      k=0;
                 }
                 fprintf(fp,"%gnS %gV ",t1, v2);
                 fprintf(fp,"%gnS %gV ",t1+edge, v1);
                 v2 = v1;
            }
       }
       fprintf(fp, "\n");
       sleep(1);
  }
  fclose(fp);
  return 0;
}
```