Design of a Fast Adder Accumulator

Hala Tabl

A Thesis

in

The Department

of

Electrical and Computer Engineering

Presented in Partial Fulfillment of the Requirements
for the Degree of Master of Engineering at
Concordia University
Montréal, Québec, Canada

July 1990

ISBN 0-315-80086-8

Canada

ABSTRACT

DESIGN OF A FAST ADDER ACCUMULATOR

The design of a pipelined adder accumulator is given. Pipelining is achieved both at the macro and micro level. The method is applied in the design of a 24-bit adder accumulator capable of adding 64K 8 bit words. The circuit is used as a threshold generator in a multi-processor image processing system. The adder is implemented in a new non standard domino logic, namely, *MULTIPLE OUTPUT DOMINO LOGIC*, with mixed dynamic and static logic.

Furthermore the overall circuit is modified to achieve a degree of fault tolerance. Fault tolerance is achieved through hardware redundancy. In our fault tolerance approach, new designs are introduced through duplication to achieve error correction, without resorting to traditional Triple Module Redundancy (TMR).

ACKNOWLEDGMENTS

My deepest thanks and gratitude goes to my supervisor Dr. A. J. AlKhalili, for his support, encouragement, and guidance, throughout the whole work. His helpful suggestions and discussions made the realization of this work possible.

My thanks also goes to Mr. D. Hargreaves and Mr. G. Patel, of the VLSI Design Lab., for providing the environment that helped in the completion of this work.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1: INTRODUCTION

Addition circuitry is an important part of any computer architecture, since it is the major component of an ALU, floating point processors, or special purpose chips[1].

Various types of adders were introduced in the literature such as carry-look-ahead, ripple carry, hierarchical, transmission gate, manchester carry, combinational, ...etc.[1]. The implementation of the circuit could be in either static logic such as CMOS complementary or Pseudo-nMOS, or dynamic (complementary or non-complementary) logic such as dynamic CMOS, clocked CMOS, domino, cascade voltage switch, ...etc.[2].

In VLSI, design area complexity and time/delay performance are important factors that affect the performance of a design. The decision to implement a certain type of logic to a certain type of adder is based on speed and area considerations needed for the required application. In our design, the carry-look-ahead adder implemented with the new Multiple Output Domino Logic, was used for its advantages of speed, and smaller area.

Adder Accumulator circuits find their uses in a variety of digital signal processing systems. In our work, the addition circuitry is used as a threshold generator, which is part of a multi-processor image processing system. The adder accumulator (threshold generator) is used to calculate the threshold of a frame as soon as the frame pixels are generated from the A/D converter. The speed of the adder accumulator has to match the speed at which the pixels are generated. In our application, this is 18MHz.

The adder accumulator is built upon two types of carry-look-ahead adders. **A full 4-bit** adder which is used to add 8 bit pixels, and a **4-bit partial** adder

which is used to accumulate the generated carry's. In addition to implementing the adders in the adder accumulator, they can also be used in the convolver which is used in convolution calculation (one of the most frequent operations in image processing). They can also be implemented in the edge detector, and any other module that requires fast addition circuitry.

With the increasing demand for reliable components, and with the advent of VLSI, many fault tolerance techniques were made available. As the designed adders could be implemented in different modules, and the reliability of these modules is of importance, thus the need to redesign the adders as fault tolerant ones.

To design a fault tolerant component, first a fault model is built covering the various faults that the fault tolerant module would be able to tolerate. In our fault analysis, classical stuck-at-faults, non-classical stuck-open faults, and physical shorts (short between gate and source, gate and drain, drain and source) are studied.

The second step in designing a fault tolerant module is to decide between various fault tolerance techniques, i.e. to select the most suitable technique for the application, in terms of time, area, and fault coverage.

In this thesis :

Chapter 2 covers the theory and design of the adder accumulator.

Chapter 3 involves the design of the 4-bit partial adder and full 4-bit adder, a comparison between our adders and other adders involving either different types of logic (classical domino, static) or different types of adders (hierarchical); then an implementation and simulation of the adder accumulator follows.

Chapter 4 covers the fault analysis for the 4-bits and full 4-bits adders.

Chapter 5 presents the analysis and design of the fault tolerant 4-bits and full 4-bits adders.

Finally Chapter 6 concludes the work.

CHAPTER 2: DESIGN OF A 24 bit ADDER ACCUMULATOR

2.1 INTRODUCTION

In this chapter we present the design of an adder accumulator using pipelining techniques. Although the method is general to any length of bits, we will consider the design of a 24 bit adder accumulator that will be used in a multiprocessor image processing system. The adder accumulator is part of the threshold generator that is used as a slave accelerator for the overall image processing multiprocessor being developed[3]. The images produced by the camera are digitised[4] to provide a digital image with a resolution of 256 by 256 pixels. Each pixel is converted to an eight-bit word, thus, it will represent a grey level[5] of 256 grey levels. In our system, pixels are produced by the A/D converter at a rate of 18 MHz. During each cycle the accumulated value of the previous pixels is retrieved from the storage flip flops, the value of the newly generated pixel is added to that previous sum, and finally the new sum is stored again.

2.2. THRESHOLD GENERATOR

The threshold generator is used to calculate the threshold value of all the pixels in a given frame, as that frame is sent by the camera. Thus we need to monitor the data bus of the system and capture the value of each of the 64K pixels, in order to calculate their threshold value, and have it available to the system.

The threshold value is defined as the mean value of the pixels within a frame, and is calculated using:

$$T = \frac{\sum_{i=1}^{256} \sum_{j=1}^{256} P_{ij}}{256 \times 256}$$

Where P_{ij} is the pixel value represented by 8 bits.

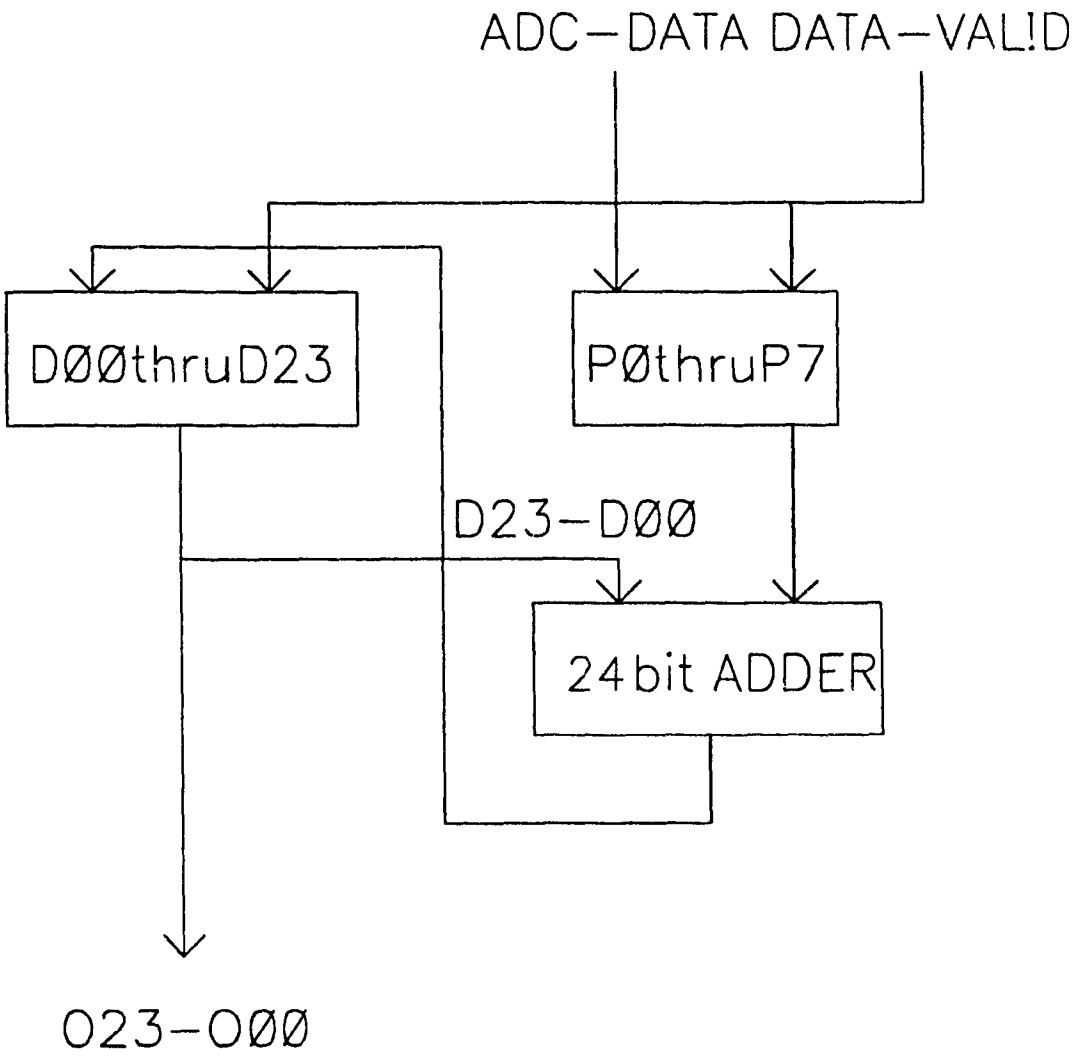
2.3 ADDER ACCUMULATOR DESIGN

As was seen in the previous section that calculating the threshold value for a frame involves the addition of 64K words of eight bits. This addition will produce a maximum sum of 24 bits. A simple solution would have been adding a 24 bit accumulator to 8 bits from A/D converter using a 24-bit adder. Fig.2.1 shows that simple configuration. However that structure would be slow.

However, introducing pipelining, makes that structure plausible, since the 24-bit adder required will be divided into multiple stages. Fig.2.2 shows the pipeline structure using 8-bit partial adders, and full 8-bit adder.

Implementing the same principle but with using 4-bit partial adders and full 4-bit adders will result in our adder accumulator shown in Fig.2.3.

In deciding on the adders size, we had the choice of either having 8-bit, 4-bit, 2-bit, or 1-bit (partial and full) adders. The 8-bit size was discarded as it would have been slow, since in the carry generation unit (of the partial or full adder) a path of 10 serial nmos transistors will have a slowing impact on the speed by which the carry's are generated, thus slowing down the adder, in addition to having a low noise margin due to the length of the nmos chain. Implementing the adders in 2-bit (or 1-bit) size would have resulted in an increased number of transistors. Considering the case of the full adder (and the same argument holds true for the partial adder), with 2-bit (or 1-bit) we would have 4 (or 8) discharge nmos for carry generation unit as opposed to 2 in the case of using 4-bit, in addition to 4 (or 8) nmos which are to be used to add carrys generated by other carry generation units, as opposed to 2 in the case of 4-bit. As to the impact of having 2-bit (or 1-bit) adders on the adder accumulator, an increase in the number of stages will result, which in turn, would increase the flushing time



ADC–DATA is from A/D converter

D23–D00 is data accumulated

O23–O00 is final result

Fig.2.1 SINGLE STAGE ADDER ACCUMULATOR

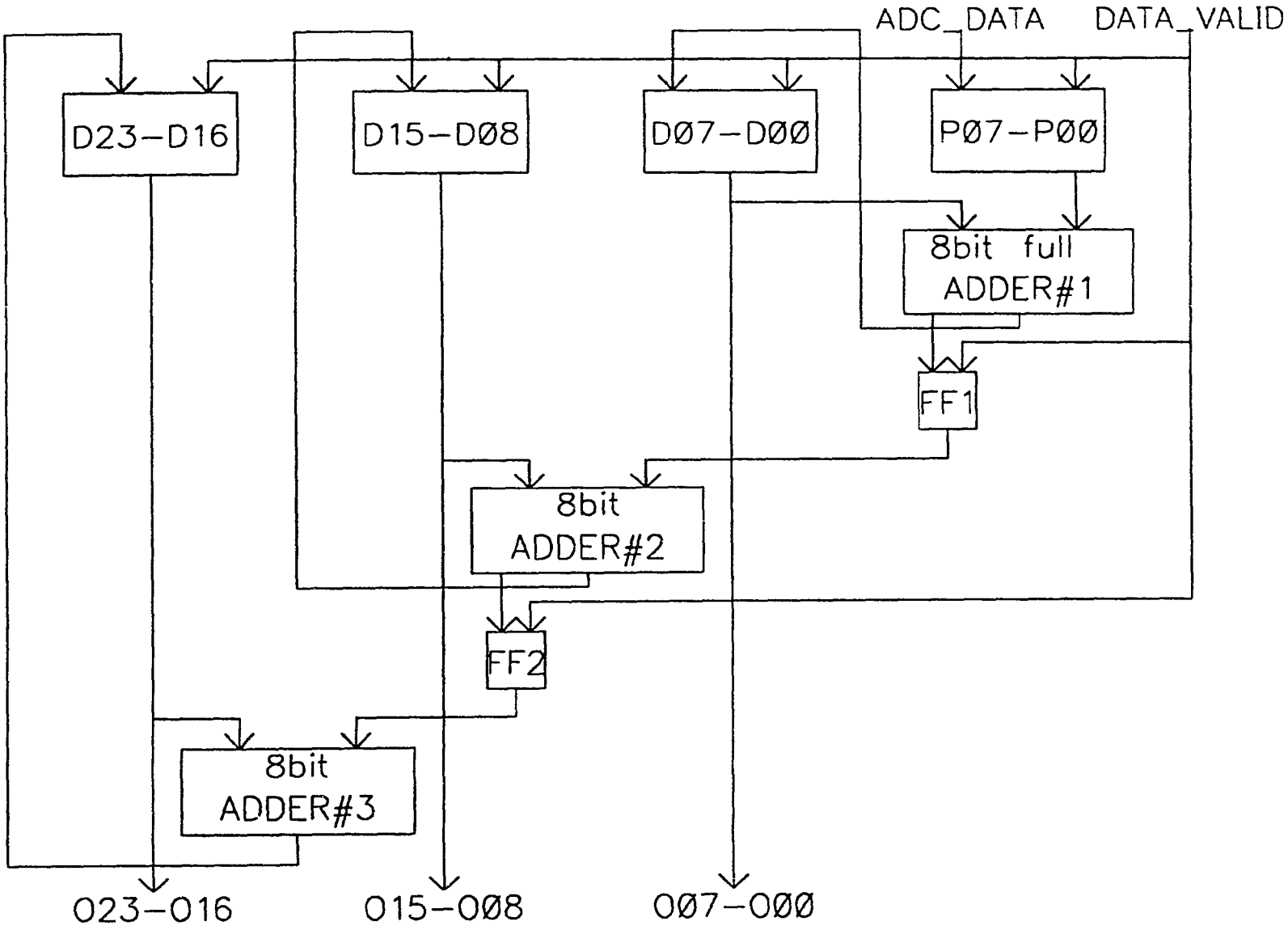
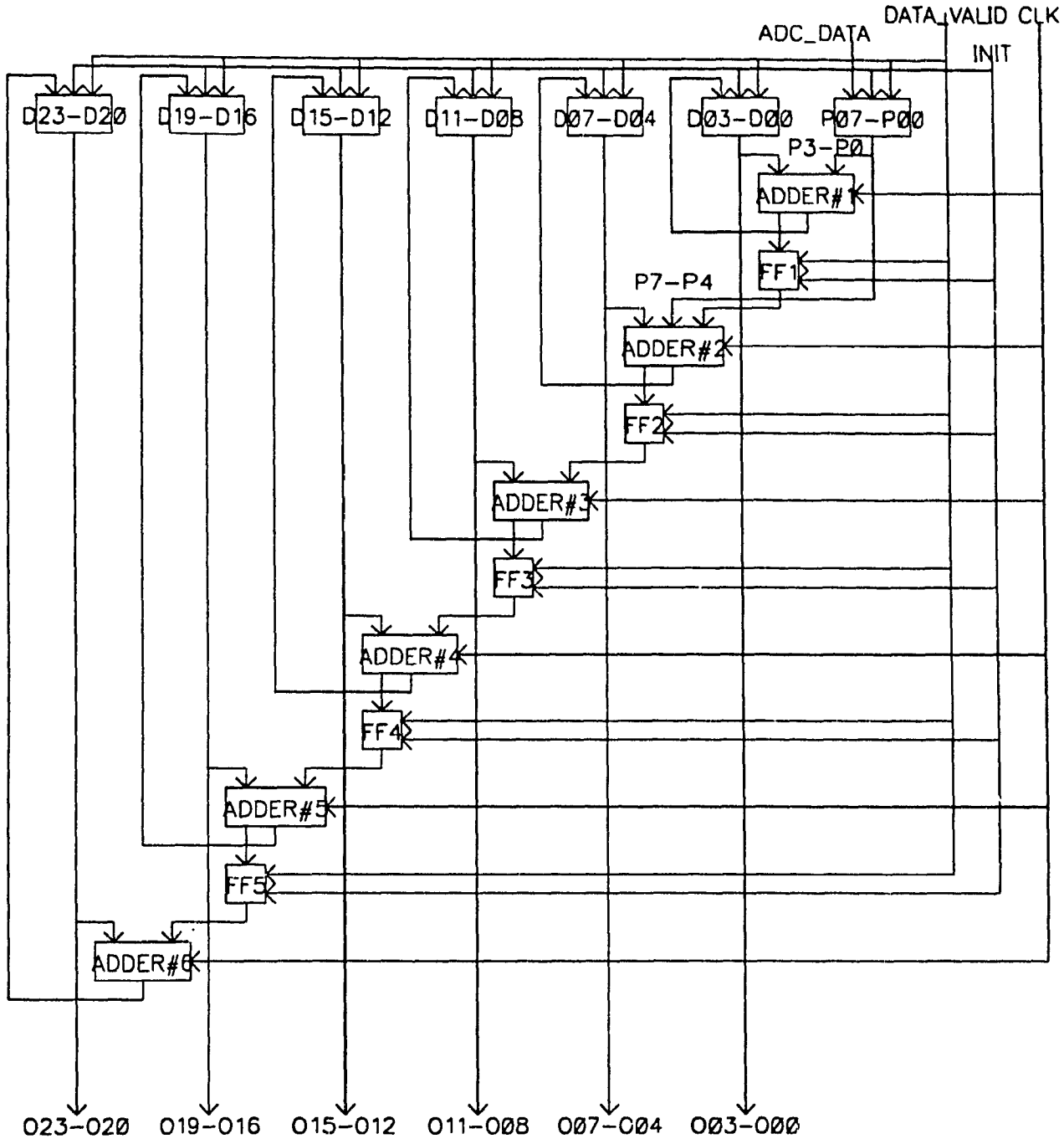


Fig.2.2 3 STAGE PIPELINE ADDER ACCUMULATOR



ADDERS 1,2 full 4-bit adders

ADDERS 3,4,5,6 4-bit partial adders

Fig.2.3 6 STAGE PIPELINE ADDER ACCUMULATOR

of the pipeline, in addition to having extra flip flops to store the resulting carry of each stage.

2.4. ADDER ACCUMULATOR FUNCTIONALITY

Each 8 bit word that comes in, is first latched by the valid data signal. The first adder sees a stable value from the input latch (4 bits from the input word), and another input from D03 - D00 of the accumulator, thus the adder produces a stable sum that is passed back to the D03 - D00 of the accumulator, while the generated carry is passed to a flip-flop, to be used with the second stage of the pipeline during the next data valid signal. At the same time the second adder sees the remaining input 4 bits, another 4 bits from the accumulator (D07 - D04) and a carry bit from the flip-flop (initially 0). Again the second adder produces a sum that is passed back to the accumulator (D07 - D04), while the generated carry is again stored in another flip-flop, to be used with the third stage of the pipeline during the next data valid signal. When the next data valid signal arrives, the first carry will be added to the sum stored in bits 4 through 7 of the accumulator, in addition to the 4 bits of the input word. Also, the second carry will be added to the sum in bits 8 through 11 of the accumulator using a 4-bit adder.

The structure shown in Fig.2.3 implements the following functions:

$$D03_i \rightarrow D00_i = D03_{i-1} \rightarrow D00_{i-1} + P03_i \rightarrow P00_i$$

$$FF1_i = carry\,(\, D03_{i-1} \rightarrow D00_{i-1} + P03_i \rightarrow P00_i \,)$$

$$D07_i \rightarrow D04_i = D07_{i-1} \rightarrow D04_{i-1} + P07_i \rightarrow P04_i + FF1_{i-1}$$

$$FF2_i = carry\,(\, D07_{i-1} \rightarrow D04_{i-1} + P07_i \rightarrow P04_i + FF1_{i-1} \,)$$

$$D11_i \rightarrow D08_i = D11_{i-1} \rightarrow D08_{i-1} + FF2_{i-1}$$

$$FF3_i = carry(D11_{i-1} \rightarrow D08_{i-1} + FF2_{i-1})$$

$$D15_i \rightarrow D12_i = D15_{i-1} \rightarrow D12_{i-1} + FF3_{i-1}$$

$$FF4_i = carry(D15_{i-1} \rightarrow D11_{i-1} + FF3_{i-1})$$

$$D19_i \rightarrow D16_i = D19_{i-1} \rightarrow D16_{i-1} + FF4_{i-1}$$

$$FF5_i = carry(D19_{i-1} \rightarrow D16_{i-1} + FF4_{i-1})$$

$$D23_i \rightarrow D20_i = D23_{i-1} \rightarrow D20_{i-1} + FF5_{i-1}$$

$$O03_i \rightarrow O00_i = D03_i \rightarrow D00_i$$

$$O07_{i+1} \rightarrow O04_{i+1} = D07_i \rightarrow D04_i + P07_{i+1} \rightarrow P04_{i+1} + FF1_i$$

$$O11_{i+2} \rightarrow O08_{i+2} = D11_{i+1} \rightarrow D08_{i+1} + FF2_{i+1}$$

$$O15_{i+3} \rightarrow O12_{i+3} = D15_{i+2} \rightarrow D12_{i+2} + FF3_{i+2}$$

$$O19_{i+4} \rightarrow O16_{i+4} = D19_{i+3} \rightarrow D16_{i+3} + FF4_{i+3}$$

$$O23_{i+5} \rightarrow O20_{i+5} = D23_{i+4} \rightarrow D16_{i+4} + FF5_{i+4}$$

Where Dxx is data in accumulator, Pxx is input pixel bits, and Oxx is resulting output, i stands for clock instant.

Fig.2.4 shows how temporary results propagate through the pipeline. It can be seen that a carry bit generated by a data word only propagates to the last bit of the accumulator five clock cycles later, i.e. with this design we have a flushing time of 5 cycles.

To explain this we give an example, and assume that at step 500 hex the

STEP	P7-PØ	DØ3-DØØ	FF1	DØ7-DØ4	FF2	D11-DØ8	FF3	D15-D12	.....
INIT.	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	
1	FF	F	Ø	F	Ø	Ø	Ø	Ø	
2	FF	E	1	E	1	Ø	Ø	Ø	
3	FF	D	1	E	1	1	Ø	Ø	
4	FF	C	1	E	1	2	Ø	Ø	
5	FF	B	1	E	1	3	Ø	Ø	
STEPS IN HEX.									
11	FF	F	Ø	E	1	F	Ø	Ø	
12	FF	E	1	D	1	Ø	1	Ø	
.....	.....	.....	.....	.....	.....	.....	.....	.....	.....

Fig.2.4 Results Propagation through Pipeline

accumulator contained 7FFFFF hex. That is the lower part contains F hex, and so does the following four parts, and the upper part contains 7 hex. Suppose also that all the carry flip-flops contained zero's. Also for the sake of clarity we'll assume that after this step the input word is 00 hex.

Now, referring to Fig.2.5, if our input word was 0F hex, say at step 500 hex, the lower part of the accumulator will be added to an F hex thus producing E hex and a 1 carry, meanwhile the second part of the accumulator wil be added to 0 hex and a 0 carry (from the previous step), producing an F hex and a zero carry. In the next step (501 hex) the carry in FF1 will be added to the second part of the accumulator thus producing 0 hex and a 1 carry. In step 502 hex, the carry in FF2 will be added to the third part of the accumulator producing a 0 hex and a 1 carry. In step 503 hex, the carry in FF3 will be added to the fourth part of the accumulator producing a 0 hex and a 1 carry. In step 504 hex, the carry in FF4 will be added to the fifth part of the accumulator producing a 0 hex and a 1 carry. Finally in step 505 hex, the carry in FF5 will be added to the upper part of the accumulator producing 8 hex and a 0 carry, and at that point the resulting value of the addition can be read from the accumulator which is 80000E hex. Fig.2.5 shows the computation steps.

Thus it can be seen that it takes five clock cycles after the last data word has arrived to get the final result of the threshold computation.

In the following chapter, the design of the adders used, and the simulation of the adder accumulator will be given.

STEP	P7-PØ	DØ3-DØØ	FF1	DØ7-DØ4	FF2	D11-DØ8	FF3	D15-D12
INIT.	X	F	Ø	F	Ø	F	Ø	F
5ØØ	ØF	E	1	F	Ø	F	Ø	F
5Ø1	ØØ	E	Ø	Ø	1	F	Ø	F
5Ø2	ØØ	E	Ø	Ø	Ø	Ø	1	F
5Ø3	ØØ	E	Ø	Ø	Ø	Ø	Ø	Ø
5Ø4	ØØ	E	Ø	Ø	Ø	Ø	Ø	Ø
→ 5Ø5	ØØ	E	Ø	Ø	Ø	Ø	Ø	Ø

O3-OØØ O7-Ø4 O11-Ø8 O15-O12

STEP	FF4	D19-D16	FF5	D23-D20
INIT.	Ø	F	Ø	7
5ØØ	Ø	F	Ø	7
5Ø1	Ø	F	Ø	7
5Ø2	Ø	F	Ø	7
5Ø3	1	F	Ø	7
5Ø4	Ø	Ø	1	7
→ 5Ø5	Ø	Ø	Ø	8

019-016 023-020

7FFFFF + ØF = 8ØØØØE

Fig.2.5 Computation Steps

CHAPTER 3: ADDERS DESIGN

3.1. INTRODUCTION

With the increasing demand for higher performance of CMOS VLSI processors with increased sophistication, there is a greater need to improve the performance, area efficiency, and functionality of the arithmetic unit within them.

As mentioned in chapter 1, various adders were introduced in the literature. Ripple carry adders provided circuits with small area but with large propagation delays[2]. Carry skip adders, implementing Manchester carry chain[3] or basic ripple adders[7], improved on the performance of ripple carry adders by making early signals more available by trading available time for extra hardware[6]. Carry select adders, on the other hand, expended area in favor of speed by duplicating two ripple carry adders, then selecting the appropriate sum[2]. Carry-look-ahead provided circuits with small propagation delay, however the complexity involved in the circuits implementing this principle remained.

Various CMOS logic styles were introduced exploiting noncomplementary structure with dynamic operations, in order to improve area efficiency, and speed of CMOS logic. Among these logic styles, NORA [8] provided high speed circuitry, high logic flexibility, and compact chip area, however, charge redistribution and leakage of storage nodes problems were disadvantages to that CMOS logic style. Domino logic [9] provided speed and area advantages, in addition to being valuable for circuits involving complex gates with high fan-in and fan-out, but still shared with NORA the charge redistribution problems. Zipper CMOS [10] incorporated the advantages of NORA and Domino in terms of structural simplicity and performance, and at the same time had a better immunity to instability and charge redistribution problems.

In the following section we will discuss the new dynamic CMOS logic style, namely Multiple Output Domino Logic (MODL), that is going to be used for the implementation of our circuits.

3.2. MULTIPLE OUTPUT DOMINO LOGIC

MODL is a new dynamic CMOS logic style introduced by Hwang and Fisher[11], that allows a single gate to produce multiple outputs. In conventional domino logic and other noncomplementary MOS logic, only one output is available from a certain gate. Thus if a certain subfunction is needed, as a separate output, it will have to be implemented with additional gates, hence, resulting in added circuitry.

The basic principle behind MODL is the use of intermediate functions that are available within the logic tree, thereby, saving the need to replicate circuitry, and hence, reducing the complexity of the design. These intermediate functions are obtained by precharging the corresponding intermediate nodes and adding inverters to obtain required outputs.

This logic design is particularly favorable as it reduces device number in a given circuit, in addition to inherent higher circuit stability as compared to standard domino and other dynamic circuits. Also MODL circuits are less susceptible to charge sharing problems, due to the precharging of the internal nodes. Due to the reduced device number, an improved performance results since load capacitance is reduced correspondingly.

The decision to implement the carry generation part of our circuits in MODL with carry-look-ahead principle stems from the overall advantage of using MODL in reducing the area and increasing the circuit performance plus allowing speed efficient designs.

As discussed previously two types of adders are used in our Adder Accumulator design. A partial bit adder which adds only a single bit to a full number, and a full adder. This distinction is based on the principle of saving area and enhancing speed.

3.3. A 4-BIT MODL ADDER

A 4-bit partial adder with a carry-look-ahead organization is shown in Fig.3.1. The adder consists of two main units. A carry generation unit which generates the intermediate carry terms and the output carry, applying the formula:

$$C_i = A_i \bullet C_{i-1}$$

As it can be seen from Fig.3.1, the series connection of N1, N2, ..., N5 gives the ANDing function at each output node.

The second unit calculates the sum product, using a static XOR, using:

$$S_i = A_i \oplus C_{i-1}$$

The circuit consisted of 20 pmos and 22 nmos, with an overall gate area of 175.68 pm^2. The carry-look-ahead devices' widths were sized in multiples of the minimum design feature (L) following the "pyramid" FET scaling technique[12]. By scaling the nmos's using this technique, better speed was achieved as opposed to having all the serial nmos's in one size. It could also be seen here that had we implemented the adder in 8-bits, the increase in the size of nmos's, specially the bottom ones, would have an increasing effect on the area, in addition to a slowing effect on speed. Also with a 2-bit or a 1-bit implementation, that technique would not have an obvious effect on speed, thus we would have only achieved an overhead in the number of extra devices, and therefore area, without achieving

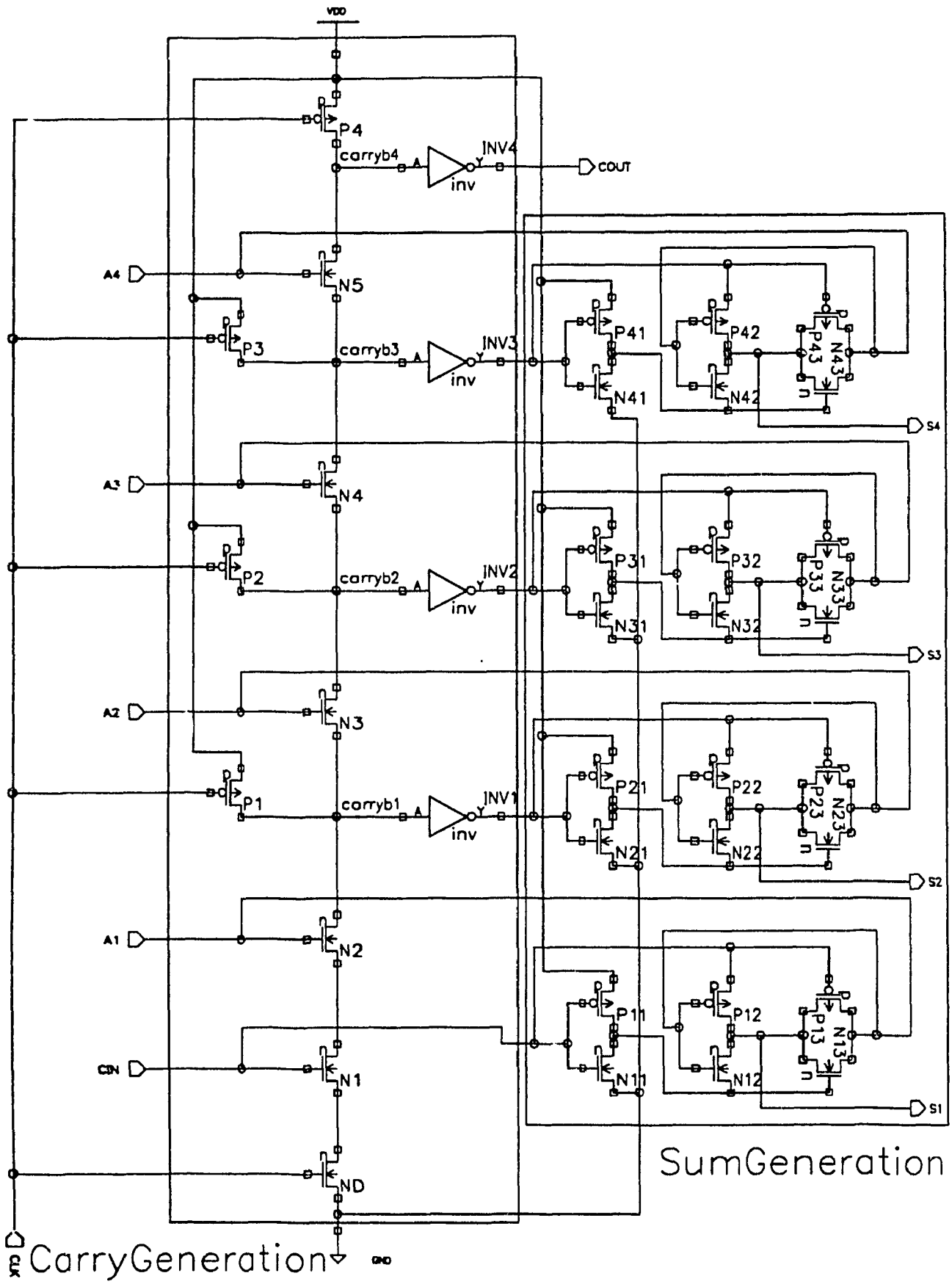


Fig.3.1 4–Bit Partial Adder using MODL

any gain in speed.

3.3.1. SIMULATION

Simulation was carried out using HSPICE, with CMOS 1.2u, at worst case conditions, i.e. at input test vector A_i 1111 and carry in $C_{in}=1$, with Vdd= 5v, and at 25° C. The worst case condition resulted in a delay, from the clock to S4, of 0.78nsec. This comes from a worst case delay of 0.57ns in the carry generation block from clock to Carry3, in addition to 0.21ns delay in the XOR in order to get S4. Fig.3.2(a, b, and c) is the input test vector and the simulation results for the 4-bit partial adder circuit.

3.4. A FULL 4-BIT MODL ADDER

The full 4-bit adder is designed using the carry-look-ahead principle in addition to MODL logic style. Fig.3.3 shows the overall design of the adder. The adder is split up to three units. The first unit generates the generate and propagate terms. The second unit generates the carry signals, and the third unit generates the sum products.

3.4.1. GENERATE AND PROPAGATE GENERATION UNIT (GEN:P&G)

This unit mainly calculates whether a carry should be generated, or simply the previous carry should be propagated. A carry is generated whenever both Ai and Bi (the bits to be added) are 1's, while the carry from the previous addition will be propagated whenever either Ai or Bi is 1. Thus the generate signal is calculated as:

$$G_i = A_i \bullet B_i$$

While the propagate signal is calculated as:

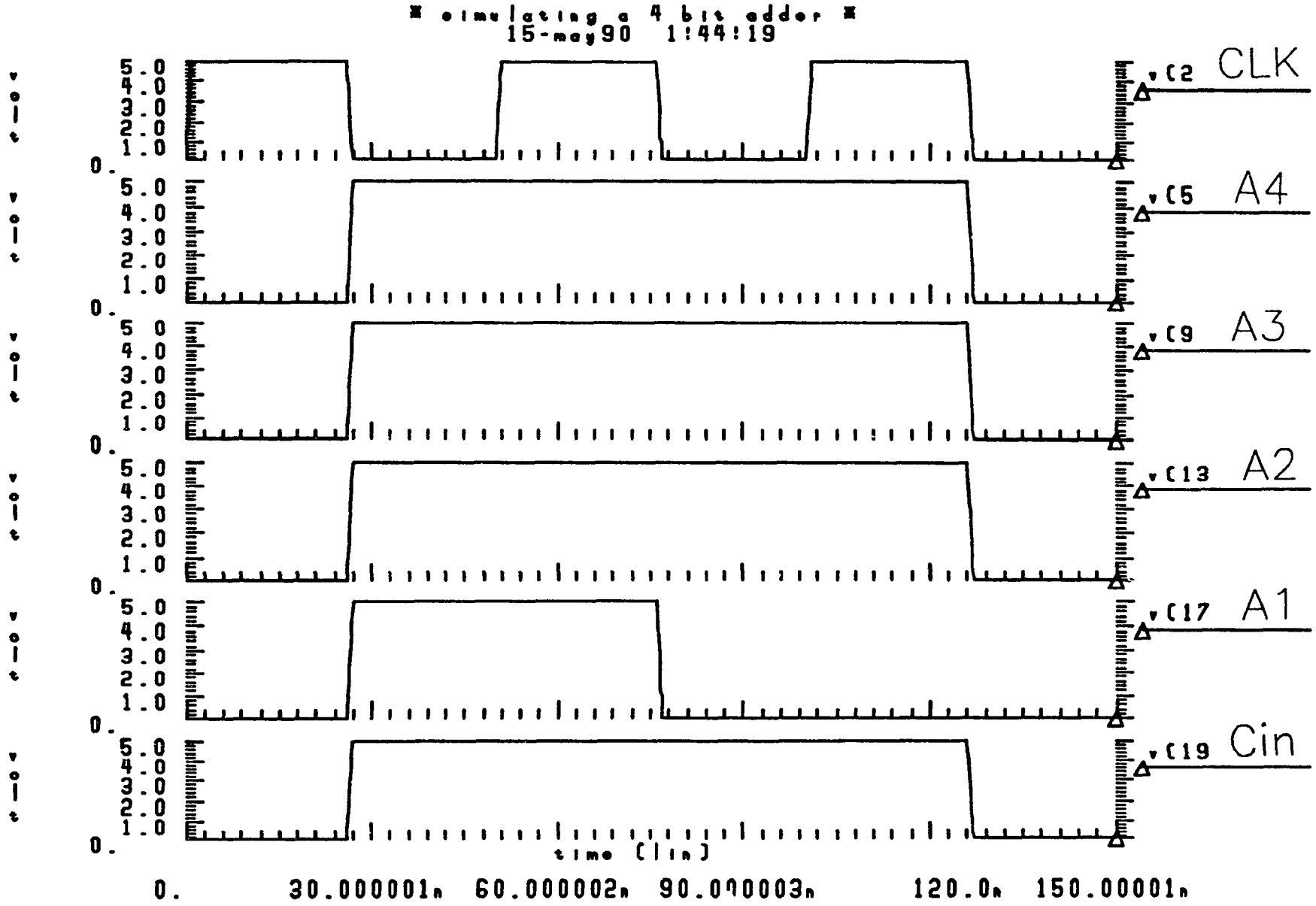


Fig.3.2.a Input Test Vectors to4-Bit Partial Adder

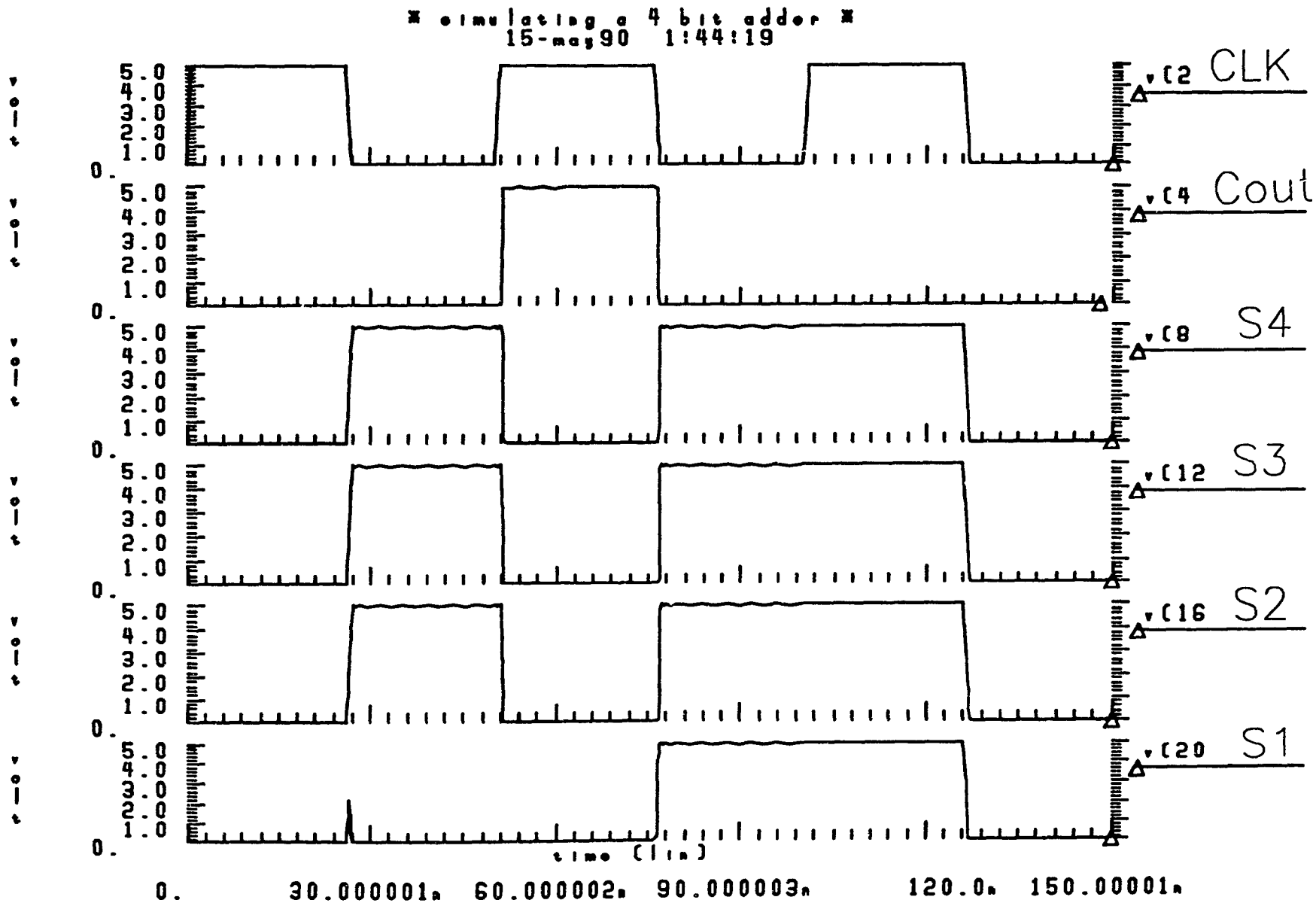


Fig.3.2.b Output Results from 4-Bit Partial Adder in Response to Fig.3.2.a

```
*PROPAGATION DELAY BETWEEN CLK AND COUT*
delay1          =  6.1315E-10  targ=  5.1113E-08  trig=  5.0500E-08

*PROPAGATION DELAY BETWEEN CLK AND SUM4*
delay2          =  7.8332E-10  targ=  5.1283E-08  trig=  5.0500E-08

*PROPAGATION DELAY BETWEEN CLK AND SUM3*
delay3          =  6.7383E-10  targ=  5.1174E-08  trig=  5.0500E-08

*PROPAGATION DELAY BETWEEN CLK AND SUM2*
delay4          =  6.2051E-10  targ=  5.1121E-08  trig=  5.0500E-08
```

*Propagation delay between CLK and SUM1 is not measured as both Cin and A1 are calculated during low CLK, thereby SUM1 has an already steady state value by the time the CLK becomes high.

Fig.3.2.c Summary of Simulation Result

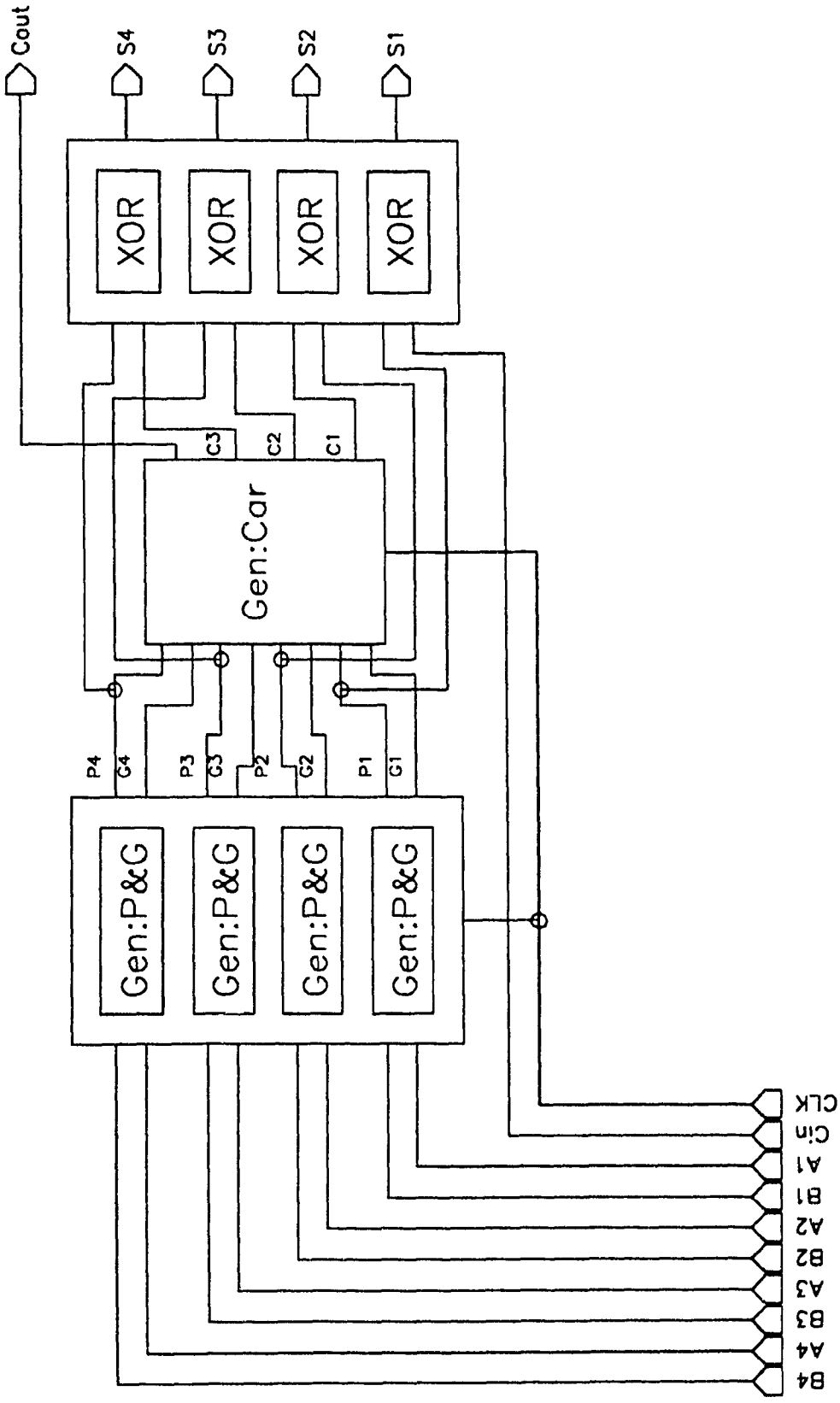


Fig.3.3 Full 4-Bit Adder using MODL

$$P_i = A_i \oplus B_i$$

The propagate signal is calculated using the exclusive OR function instead of an OR function (which is commonly found in literature) in order to avoid the case of having both the P signal and the G signal to be simultaneously equal to 1 (case when both Ai and Bi are equal to 1), as this would cause a charge redistribution problem. As the P signal is calculated during low clock, while the G signal is calculated during high clock, with P equal to 1 the nmos it controls will be turned on, now, if during high clock G also goes high a charge redistribution will occur, which might cause erroneous results.

Fig.3.4 shows the design of the Gen:P&G unit.

3.4.2. CARRY GENERATION UNIT (GEN:CAR)

The Gen:Car unit is 4-bit MODL carry generator that collects the resulting 8 generate and propagate signals, from the previous unit, together with the input carry (Cin). It then calculates the intermediate carrys (C1, C2, and C3), together with the output carry (Cout). The carry calculation is carried as follows:

$$C_i = G_i + (P_i \bullet C_{i-1})$$

Fig.3.5 shows the design of the Gen:Car unit.

3.4.3. SUM CALCULATION UNIT (XOR)

Now with both the carrys and the propagate signals available from the previous stages, the sum products are calculated through a static XOR using the formula:

$$S_i = P_i \oplus C_{i-1}$$

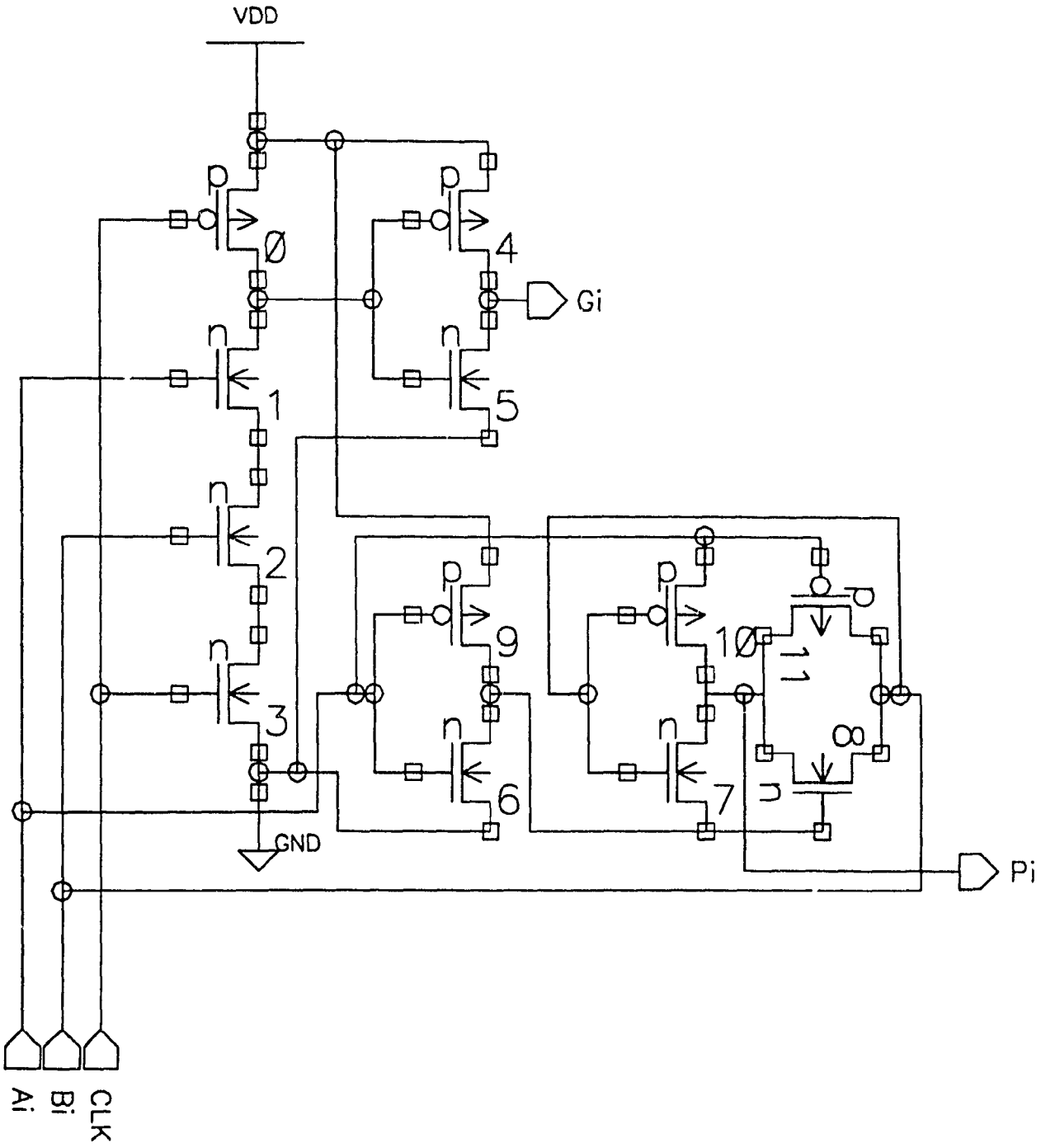


Fig.3.4 Generate & Propagate Circuit (GEN:P&G)

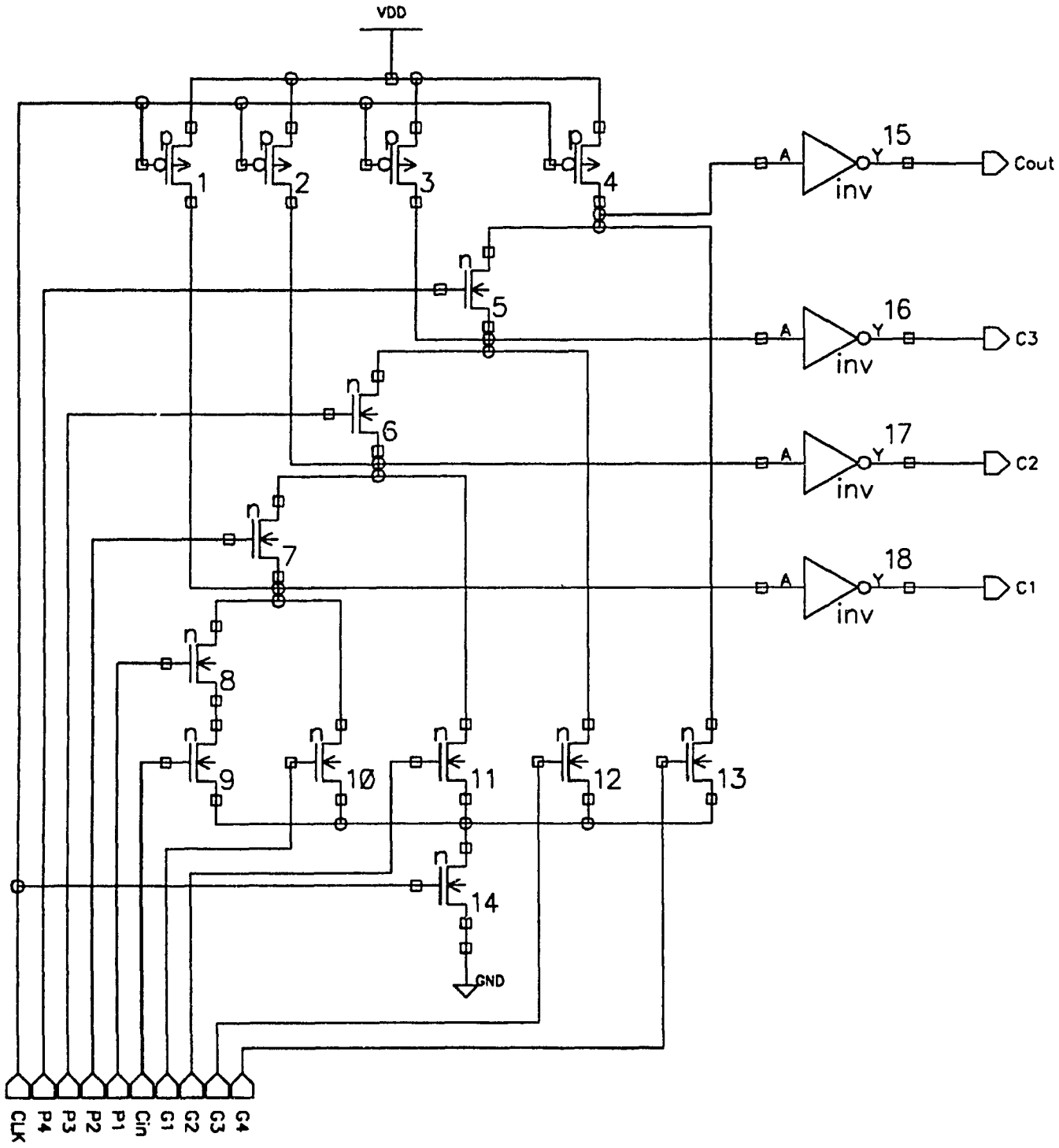


Fig.3.5 Carry Generation Circuit(GEN:CAR)

Fig.3.6 shows the static XOR used. While Fig.3.7 shows the full 4-bit adder in transistor level.

As in the previous adder design, the devices' widths were sized in multiples of the minimum design feature (L), with "pyramid" FET scaling technique [12]. The circuit contains 40 pmos and 54 nmos with overall gate area of 368.64 pm^2.

3.4.4. SIMULATION

Simulation was carried out again using HSPICE, with CMOS 1.2u, with a set of test vectors, Vdd=5v, and at 25° C. The worst case condition resulted in a delay, from clock to S4, of 0.92nsec. In the Gen:P&G block, with the worst case condition simulation, all G's were set to 0. The resulting P's from the XOR were calculated during low clock, therefore, they represented no delay during the following evaluation clock. In the Gen:Car unit, the worst case delay between clock and Carry3 was 0.7ns, that in addition to 0.22ns delay in XOR generating the sum, resulted in the 1ns overall worst case delay of the complete adder. Fig.3.8(a, b, c, and d) shows the simulation results for the full 4-bit adder circuit.

3.5. COMPARISON

In this section we present the results of various simulations carried on three other full 4-bit adders so as to compare between these adders' performance in term of area and gate count, and speed to our full 4-bit adder. Fig.3.9.a shows a static carry-look-ahead adder[2], while Fig.3.9.b shows the simulation results (the same test vectors were used as in the case of the full 4-bit adder). From the simulation the adder gave a worst case propagation delay of 2.29ns (input A4 to S4), and the overall transistor area was 410.112 pm^2, with a total transistor count of 190 mosfets.

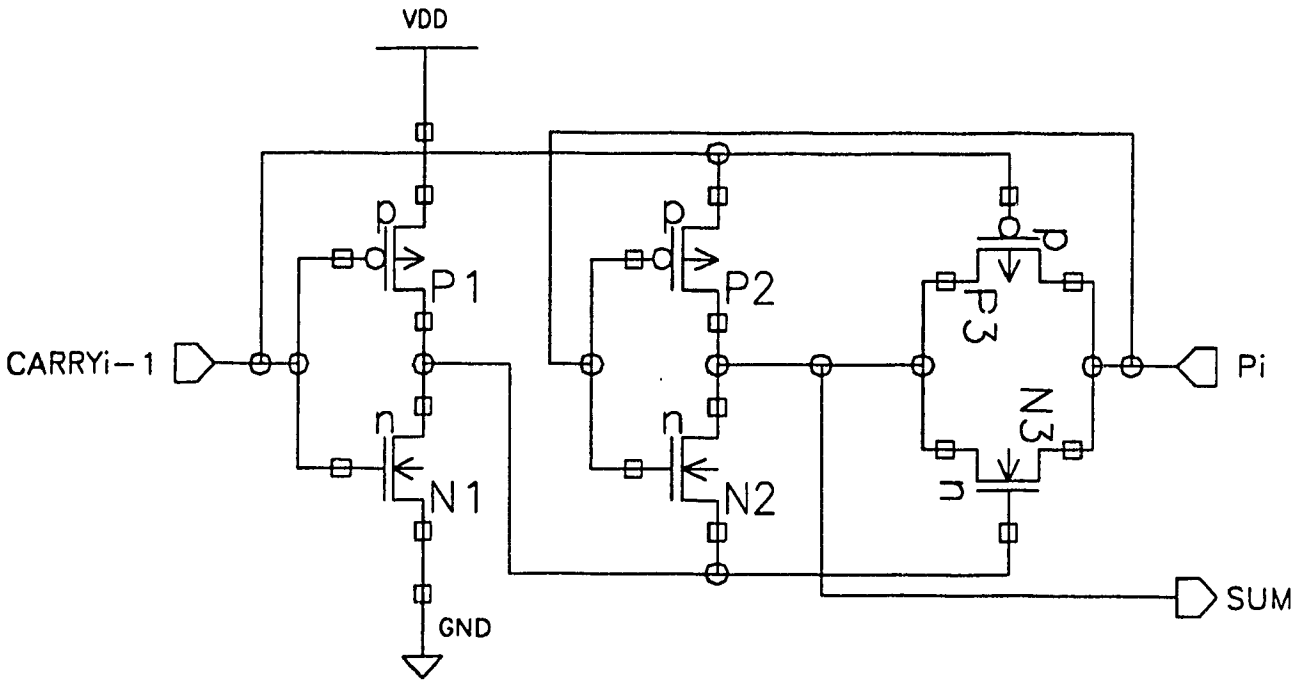


Fig.3.6 Exclusive OR Circuit

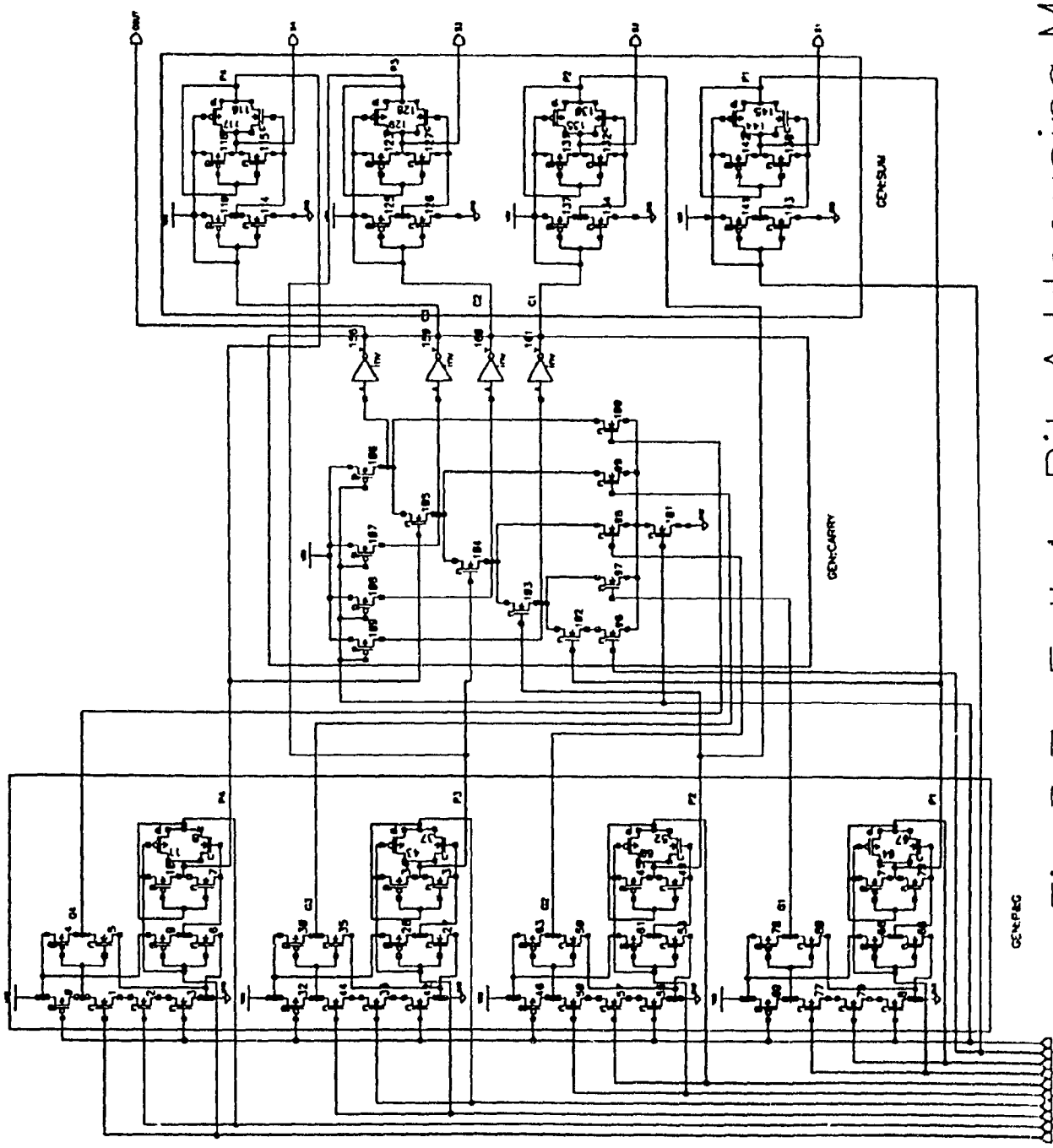


Fig.3.7 Full 4-Bit Adder using MODL

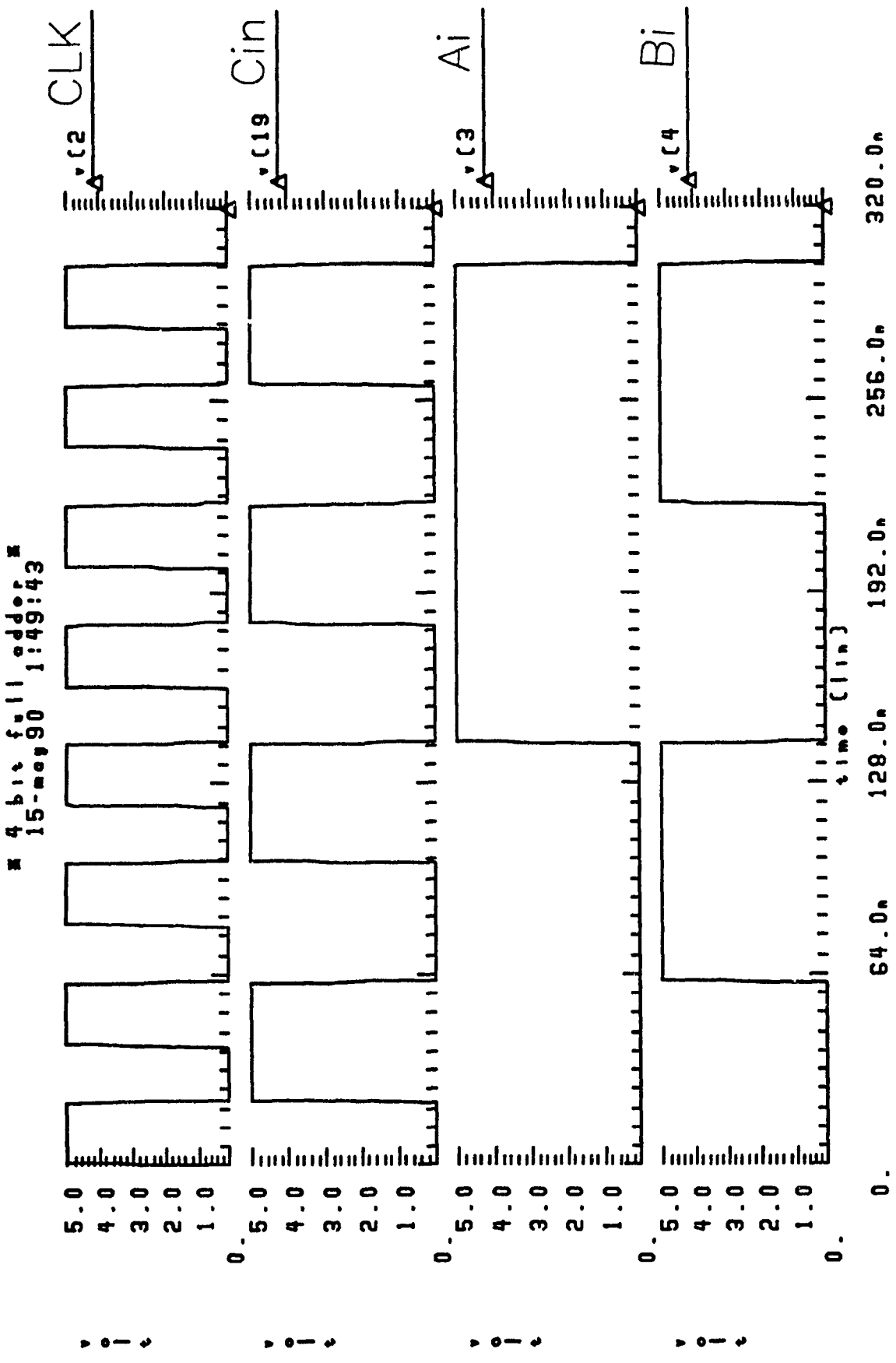


Fig.3.8.a Input Test Vectors to Full 4—bit Adder

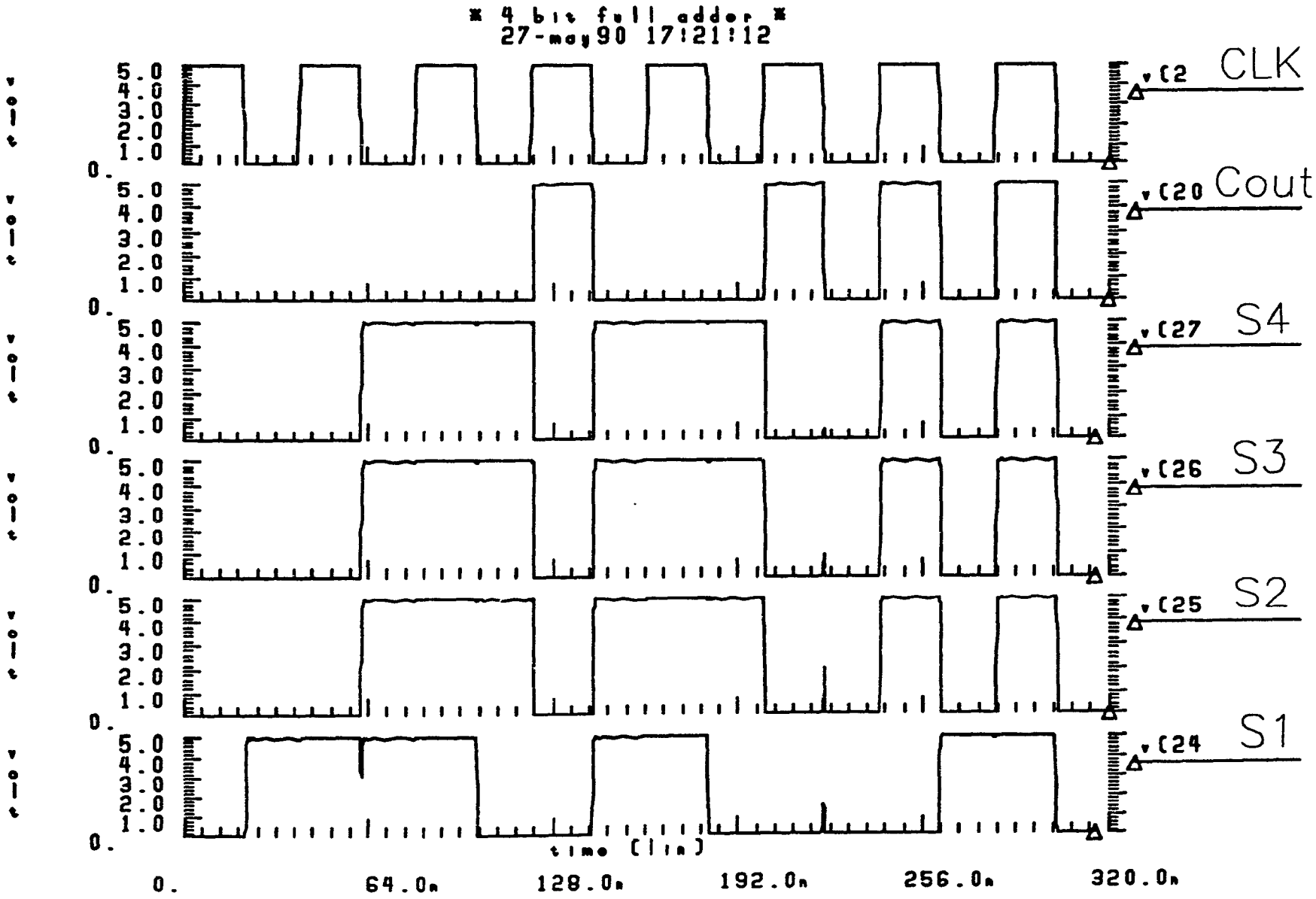


Fig.3.8.b Output Results from Full 4-bit Adder in Response to Fig.3.8.a

```
*PROPAGATION DELAY BETWEEN CLK AND COUT*
delay1          =  7.0488E-10  targ=  1.2120E-07  trig=  1.2050E-07
delay2          =  7.0252E-10  targ=  2.0120E-07  trig=  2.0050E-07
delay3          =  5.2661E-10  targ=  2.4103E-07  trig=  2.4050E-07
delay4          =  5.1726E-10  targ=  2.8102E-07  trig=  2.8050E-07

*PROPAGATION DELAY BETWEEN CLK AND SUM4*
delay5          =  9.1651E-10  targ=  1.2142E-07  trig=  1.2050E-07
delay6          =  9.1828E-10  targ=  2.0142E-07  trig=  2.0050E-07
delay7          =  7.9400E-10  targ=  2.4129E-07  trig=  2.4050E-07
delay8          =  7.7795E-10  targ=  2.8128E-07  trig=  2.8050E-07

*PROPAGATION DELAY BETWEEN CLK AND SUM3*
delay9          =  7.1115E-10  targ=  1.2121E-07  trig=  1.2050E-07
delay10         =  7.1331E-10  targ=  2.0121E-07  trig=  2.0050E-07
delay11         =  7.9210E-10  targ=  2.4129E-07  trig=  2.4050E-07
delay12         =  7.7383E-10  targ=  2.8127E-07  trig=  2.8050E-07

*PROPAGATION DELAY BETWEEN CLK AND SUM2*
delay13         =  6.5226E-10  targ=  1.2115E-07  trig=  1.2050E-07
delay14         =  6.4978E-10  targ=  2.0115E-07  trig=  2.0050E-07
delay15         =  7.9230E-10  targ=  2.4129E-07  trig=  2.4050E-07
delay16         =  7.7252E-10  targ=  2.8127E-07  trig=  2.8050E-07

*Propagation delay between CLK and SUM1 is not measured as both Cin
and P1 are calculated during low CLK, thereby SUM1 has an already
steady state value by the time the CLK becomes high.
```

Fig.3.8.c Summary of Simulation Result

```
* PROPAGATION DELAY is 9.1828E-10 sec. *
* RISE TIME is 5.3262E-10 sec. *
* FALL TIME is 3.1559E-10 sec. *
* AVERAGE POWER DISSIPATION is 2.6450E-04 watt *
* MINIMUM POWER DISSIPATION is 6.0400E-09 watt *
* MAXIMUM POWER DISSIPATION is 1.9132E-02 watt *
* MAXIMUM Idd = 2.3869E-4 amp*
* TEMP = 25 C *
* Vdd = 5 volts *
* CAPCITIVE LOAD ON EACH OUTPUT(assuming a buffer) is 5.5801E-14 farad *
```

Fig.3.8.d Specification of Full 4-Bit Adder

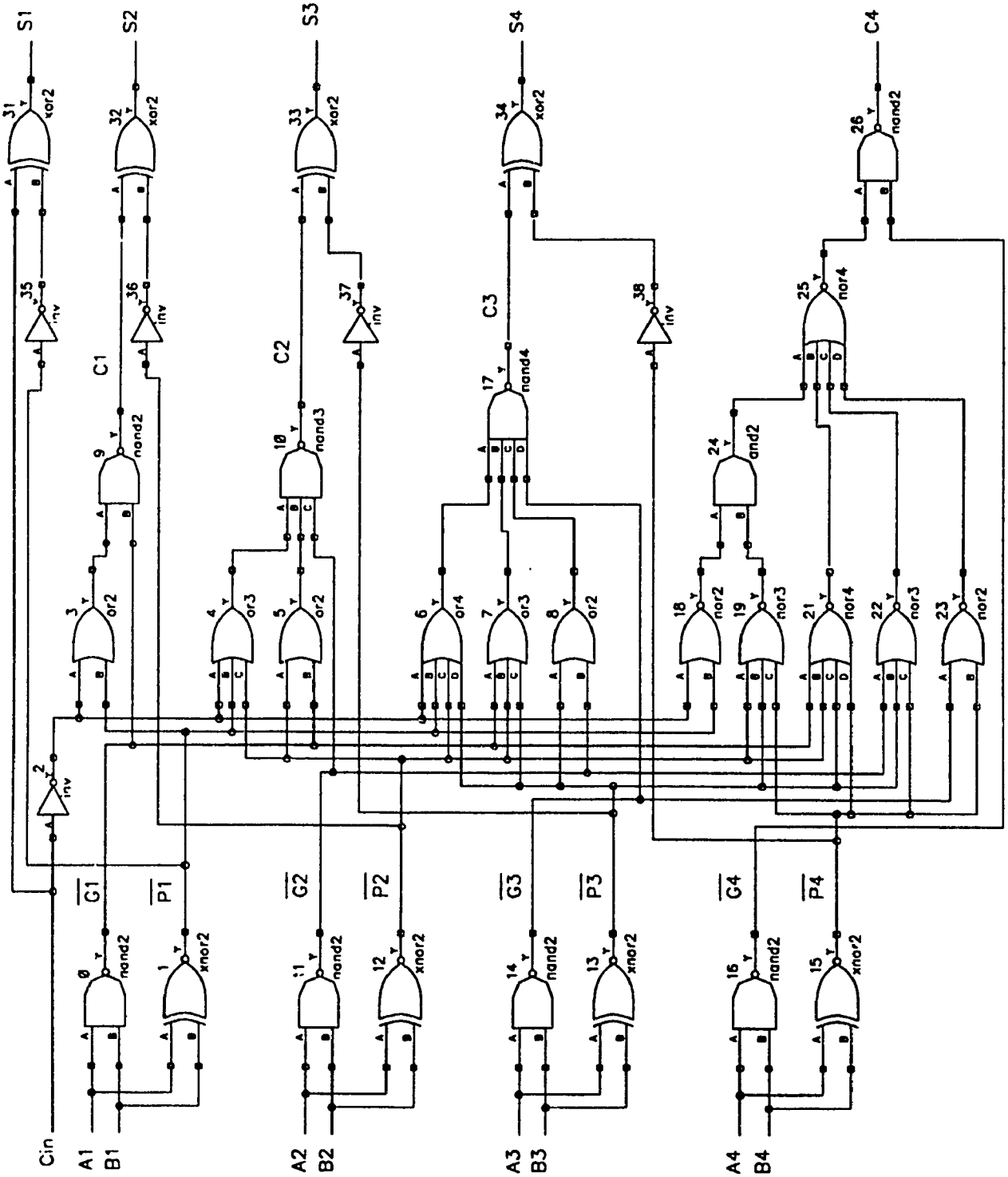


Fig.3.9.a STATIC CLA ADDER

```
*PROPAGATION DELAY BETWEEN I/P AND COUT*
delay9          =  1.7790E-09  targ=  1.0283E-07  trig=  1.0105E-07
delay14         =  2.1115E-09  targ=  1.4316E-07  trig=  1.4105E-07
delay19         =  1.7652E-09  targ=  1.8282E-07  trig=  1.8105E-07
delay28         =  2.0686E-10  targ=  3.0136E-07  trig=  3.0115E-07

*PROPAGATION DELAY BETWEEN I/P AND SUM4*
delay4          =  7.4525E-10  targ=  6.1795E-08  trig=  6.1050E-08
delay8          =  1.7139E-09  targ=  1.0276E-07  trig=  1.0105E-07
delay13         =  2.2905E-09  targ=  1.4334E-07  trig=  1.4105E-07
delay18         =  1.7160E-09  targ=  1.8277E-07  trig=  1.8105E-07
delay22         =  9.6583E-10  targ=  2.2202E-07  trig=  2.2105E-07
delay27         =  6.0324E-10  targ=  3.0175E-07  trig=  3.0115E-07

*PROPAGATION DELAY BETWEEN I/P AND SUM3*
delay3          =  8.4915E-10  targ=  6.1899E-08  trig=  6.1050E-08
delay7          =  1.4280E-09  targ=  1.0248E-07  trig=  1.0105E-07
delay12         =  1.9063E-09  targ=  1.4296E-07  trig=  1.4105E-07
delay17         =  1.4191E-09  targ=  1.8247E-07  trig=  1.8105E-07
delay21         =  1.1141E-09  targ=  2.2216E-07  trig=  2.2105E-07
delay26         =  5.5220E-10  targ=  3.0170E-07  trig=  3.0115E-07

*PROPAGATION DELAY BETWEEN I/P AND SUM2*
delay2          =  8.9622E-10  targ=  6.1946E-08  trig=  6.1050E-08
delay6          =  1.1643E-09  targ=  1.0221E-07  trig=  1.0105E-07
delay11         =  1.5297E-09  targ=  1.4258E-07  trig=  1.4105E-07
delay16         =  1.1644E-09  targ=  1.8221E-07  trig=  1.8105E-07
delay20         =  1.2782E-09  targ=  2.2233E-07  trig=  2.2105E-07
delay25         =  5.6864E-10  targ=  3.0172E-07  trig=  3.0115E-07

*PROPAGATION DELAY BETWEEN I/P AND SUM1*
delay1          =  1.4397E-10  targ=  2.1194E-08  trig=  2.1050E-08
delay5          =  2.1000E-10  targ=  1.0126E-07  trig=  1.0105E-07
delay10         =  2.6148E-10  targ=  1.4131E-07  trig=  1.4105E-07
delay15         =  2.1000E-10  targ=  1.8126E-07  trig=  1.8105E-07
delay23         =  1.4184E-10  targ=  2.6119E-07  trig=  2.6105E-07
delay24         =  1.0709E-10  targ=  3.0126E-07  trig=  3.0115E-07
```

Fig.3.9.b Summary of Simulation Result

The second adder we used for comparison was a standard domino carry-look-ahead adder[2] (shown in Fig.3.10.a). From the simulation (Fig.3.10.b) the adder gave a worst case propagation delay of 1.54ns (CLK to S4), and the overall transistor area was 665.136 pm^2, with a total transistor count of 116 mosfets.

Our last simulation was carried on a static hierarchical adder (shown in Fig.3.11.a) so as to compare our adder with an adder with a different type. Simulation results (Fig.3.11.b) showed a propagation delay of 3.02ns (input B4 to S4), and with a transistor area of 327.6 pm^2, and a total transistor count of 136 mosfets.

The following table summarizes the simulation results and the percentage of speed, area, and gate count between the MODL adder and the other adders.

Adder	Prop.del. (ns)	% speed	Area (pm)	% Inc. area	no.of gates	% Inc. gate
MODL	0.92	100	368.64	0	94	0
Static CLA	2.29	40.17	410.112	111.25	190	202
Std. Domino CLA	1.54	59.7	665.136	180.4	116	123.4
Hierarchical	3.02	30.46	327.6	88.9	136	144.7

Table 3.1 Comparison Between Various Adders

From the simulation results it could be seen that in terms of speed the MODL adder is the fastest, followed by the standard Domino CLA adder (with only 60 % of the MODL speed). In terms of area MODL compares favorably with standard Domino, and static CLA adder, and even though it is slightly larger than the hierarchical adder (due to the pyramid sizing in the carry-look-ahead block, which is not required in the hierarchical), it still compares favorably to all the adders in term of gate count.

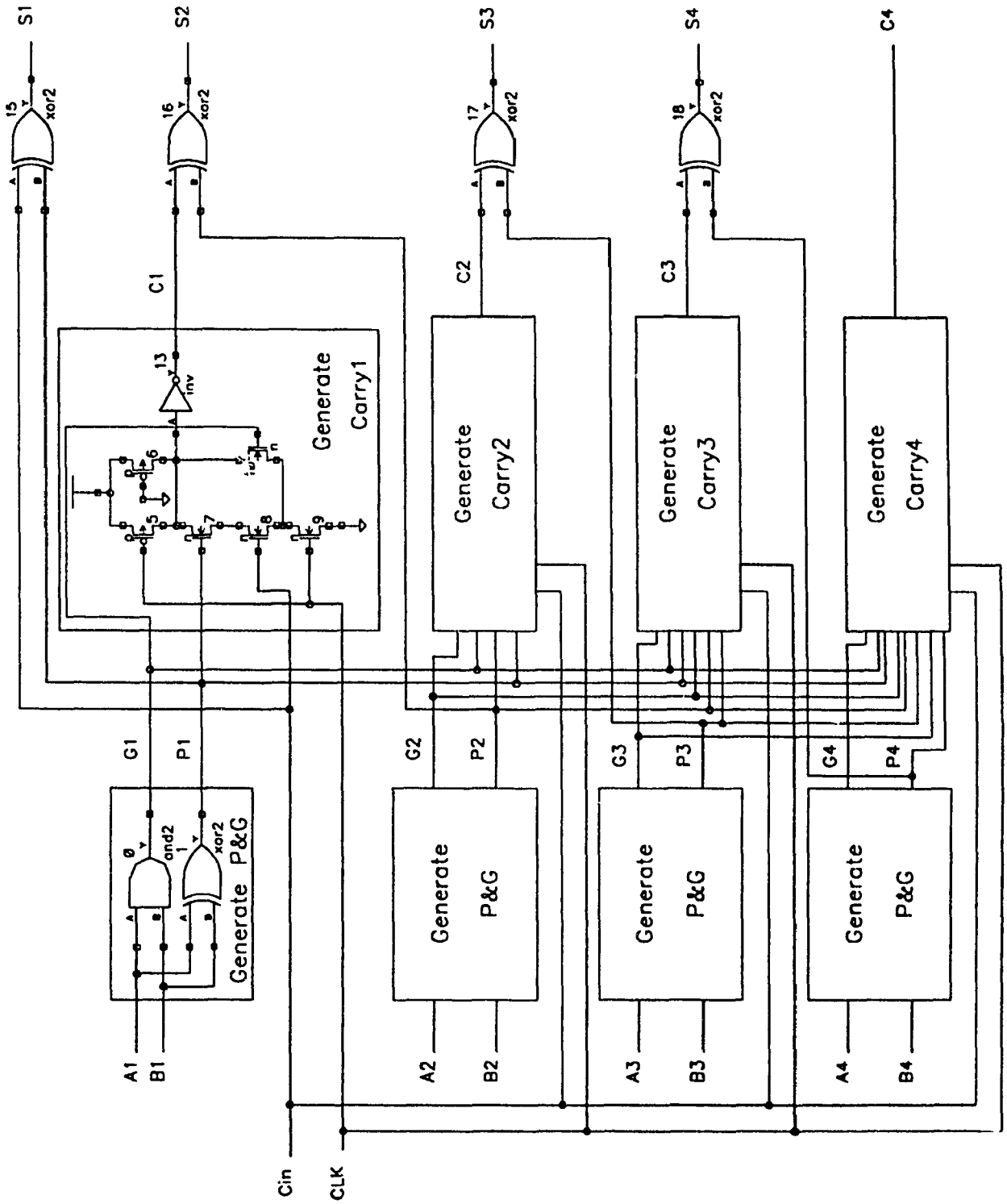


Fig.3.10.a Standard DOMINO CLA ADDER

```
*PROPAGATION DELAY BETWEEN CLK AND COUT*
delay4          =  1.2071E-09  targ=  1.2171E-07  trig=  1.2050E-07
delay8          =  1.2071E-09  targ=  2.0171E-07  trig=  2.0050E-07
delay12         =  3.0608E-10  targ=  2.4081E-07  trig=  2.4050E-07
delay16         =  3.2386E-10  targ=  2.8082E-07  trig=  2.8050E-07

*PROPAGATION DELAY BETWEEN CLK AND SUM4*
delay3          =  1.5360E-09  targ=  1.2204E-07  trig=  1.2050E-07
delay7          =  1.5360E-09  targ=  2.0204E-07  trig=  2.0050E-07
delay11         =  7.8898E-10  targ=  2.4129E-07  trig=  2.4050E-07
delay15         =  8.1950E-10  targ=  2.8132E-07  trig=  2.8050E-07

*PROPAGATION DELAY BETWEEN CLK AND SUM3*
delay2          =  1.1681E-09  targ=  1.2167E-07  trig=  1.2050E-07
delay6          =  1.1683E-09  targ=  2.0167E-07  trig=  2.0050E-07
delay10         =  7.8934E-10  targ=  2.4129E-07  trig=  2.4050E-07
delay14         =  8.1857E-10  targ=  2.8132E-07  trig=  2.8050E-07

*PROPAGATION DELAY BETWEEN CLK AND SUM2*
delay1          =  9.3269E-10  targ=  1.2143E-07  trig=  1.2050E-07
delay5          =  9.3272E-10  targ=  2.0143E-07  trig=  2.0050E-07
delay9          =  7.9733E-10  targ=  2.4130E-07  trig=  2.4050E-07
delay13         =  8.2783E-10  targ=  2.8133E-07  trig=  2.8050E-07
```

Fig.3.10.b Summary of Simulation Result

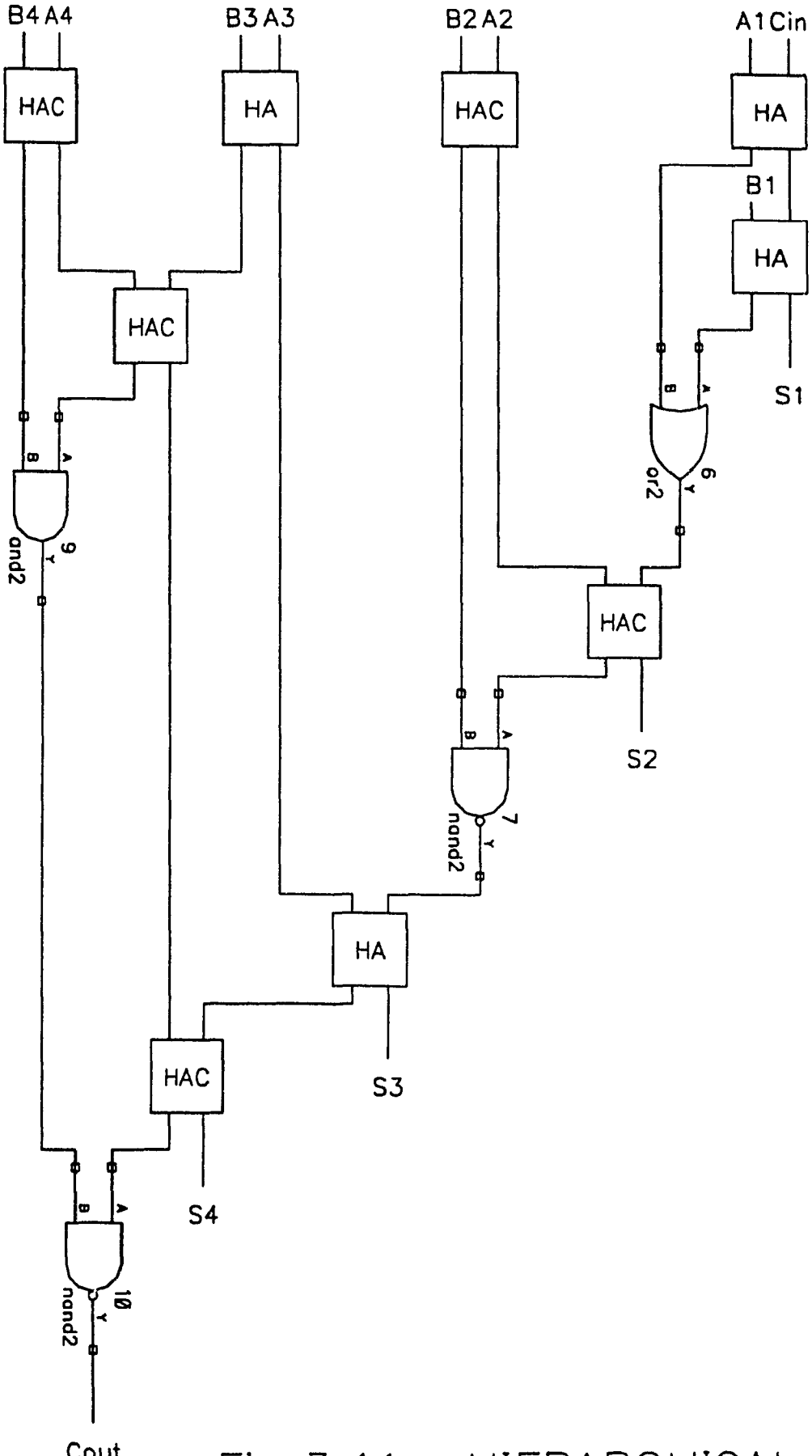


Fig.3.11.a HIERARCHICAL ADDER

```
*PROPAGATION DELAY BETWEEN I/P AND COUT*
delay9          =  2.9736E-09  targ=  1.0402E-07  trig=  1.0105E-07
delay14         =  2.7272E-09  targ=  1.4378E-07  trig=  1.4105E-07
delay19         =  2.4102E-09  targ=  1.8346E-07  trig=  1.8105E-07
delay28         =  5.6742E-10  targ=  3.0172E-07  trig=  3.0115E-07

*PROPAGATION DELAY BETWEEN I/P AND SUM4*
delay4          =  1.3697E-09  targ=  6.2420E-08  trig=  6.1050E-08
delay8          =  3.0177E-09  targ=  1.0407E-07  trig=  1.0105E-07
delay13         =  2.8698E-09  targ=  1.4392E-07  trig=  1.4105E-07
delay18         =  2.4424E-09  targ=  1.8349E-07  trig=  1.8105E-07
delay22         =  1.5263E-09  targ=  2.2258E-07  trig=  2.2105E-07
delay27         =  1.3780E-09  targ=  3.0253E-07  trig=  3.0115E-07

*PROPAGATION DELAY BETWEEN I/P AND SUM3*
delay3          =  8.2208E-10  targ=  6.1872E-08  trig=  6.1050E-08
delay7          =  2.4870E-09  targ=  1.0354E-07  trig=  1.0105E-07
delay12         =  2.3677E-09  targ=  1.4342E-07  trig=  1.4105E-07
delay17         =  1.8859E-09  targ=  1.8294E-07  trig=  1.8105E-07
delay21         =  1.0305E-09  targ=  2.2208E-07  trig=  2.2105E-07
delay26         =  7.2975E-10  targ=  3.0188E-07  trig=  3.0115E-07

*PROPAGATION DELAY BETWEEN I/P AND SUM2*
delay2          =  7.5472E-10  targ=  6.1805E-08  trig=  6.1050E-08
delay6          =  1.9635E-09  targ=  1.0301E-07  trig=  1.0105E-07
delay11         =  1.8164E-09  targ=  1.4287E-07  trig=  1.4105E-07
delay16         =   .4206E-09  targ=  1.8247E-07  trig=  1.8105E-07
delay20         =   .9427E-09  targ=  2.2299E-07  trig=  2.2105E-07
delay25         =  1.1525E-09  targ=  3.0230E-07  trig=  3.0115E-07

*PROPAGATION DELAY BETWEEN I/P AND SUM1*
delay1          =  9.5003E-10  targ=  2.2000E-08  trig=  2.1050E-08
delay5          =  1.0607E-09  targ=  1.0211E-07  trig=  1.0105E-07
delay10         =  1.1313E-09  targ=  1.4218E-07  trig=  1.4105E-07
delay15         =  1.0082E-09  targ=  1.8206E-07  trig=  1.8105E-07
delay23         =  1.0481E-09  targ=  2.6210E-07  trig=  2.6105E-07
delay24         =  8.7432E-10  targ=  3.0202E-07  trig=  3.0115E-07
```

Fig.3.11.b Summary of Simulation Result

3.6. SIMULATING THE ADDER ACCUMULATOR

With both the 4-bit partial adder and the full 4-bit adders, designed, the following step is the simulation of the adder accumulator, discussed in the previous chapter.

Due to the dynamic nature of the adders, the output of the adders will be stored during high clock after the outputs have reached a steady state by using a result valid signal(result valid signal goes high 6ns after the clock is high, and it remains high for 12ns, then goes low 1ns before the clock goes low and the results are no longer valid). On the other hand, data from the registers are to be presented to the adders during the low clock using a data valid signal(data valid signal goes high during low clock 6.5ns before the clock goes high, remains high for 5ns, then goes low 2.3ns before clock goes high). Fig.3.12 shows a segment of the adder accumulator with various control signals. As D flip-flops are used for storage, it was taken in consideration that the duration of the control signals be long enough to ensure stable results. Fig.3.13 shows the timing diagram of the control signals. The INIT signal is used to clear the registers at the beginning of every new frame(INIT signal, again, goes high during low clock, remains high for 1 ns and meanwhile a 0 input is presented to the flip flops to initialize them, then signal goes low before the data valid signal goes high by 0.3ns).

The adder accumulator is simulated with 0 initial conditions, followed by FF words for 4 clock cycles, followed by 00 words to clear the pipeline and propagate the generated carry's down the pipe. Fig.3.14 shows the simulation results.

The delay in every stage of the adder accumulator is proportional to the adder's delay in that stage, and as explained in the previous chapter, the final result of the addition, will be available after 5 clock cycles from the last word arrival.

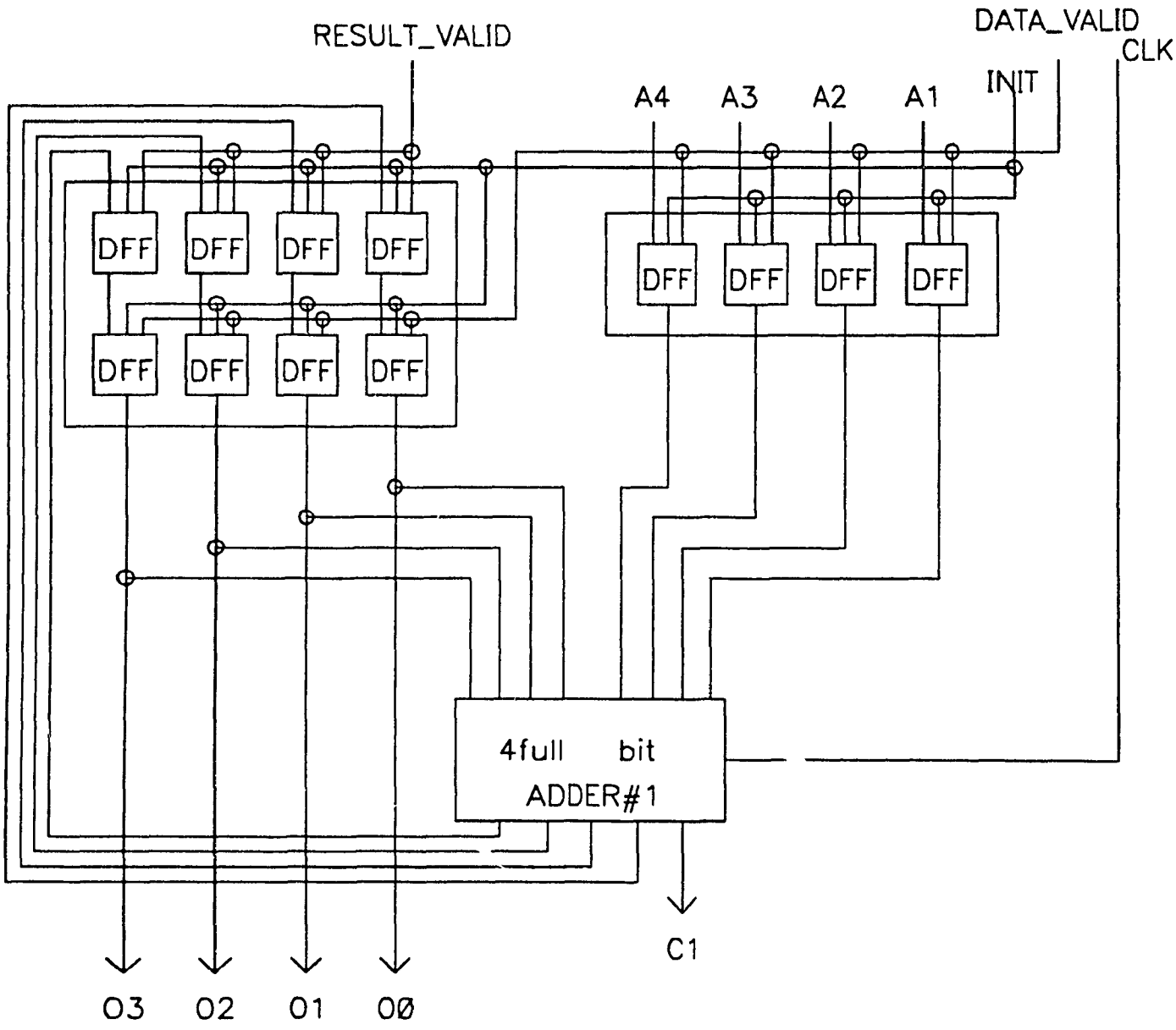


Fig.3.12 ADDER ACCUMULATOR

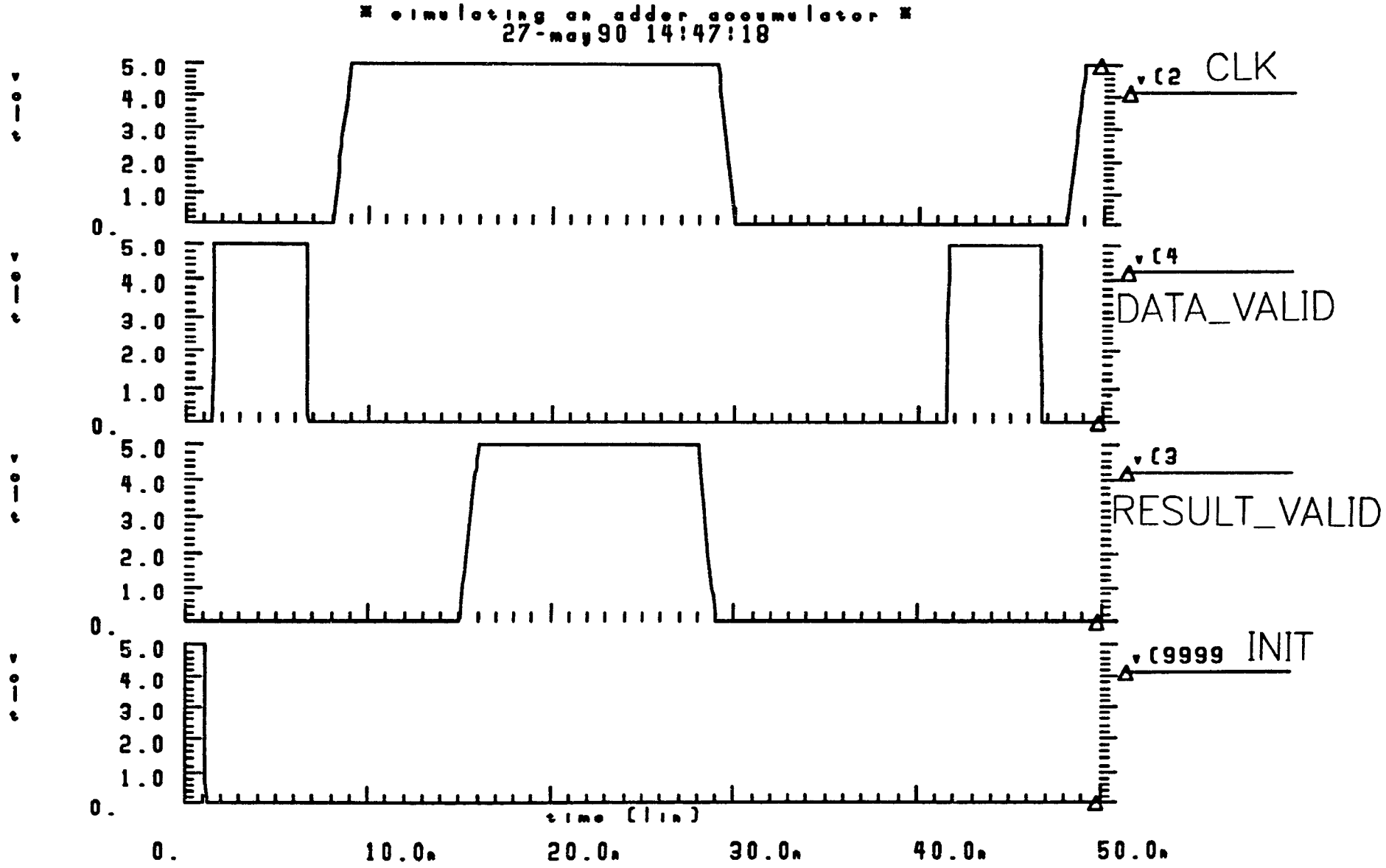


Fig.3.13 Timing Diagram of Control Signals

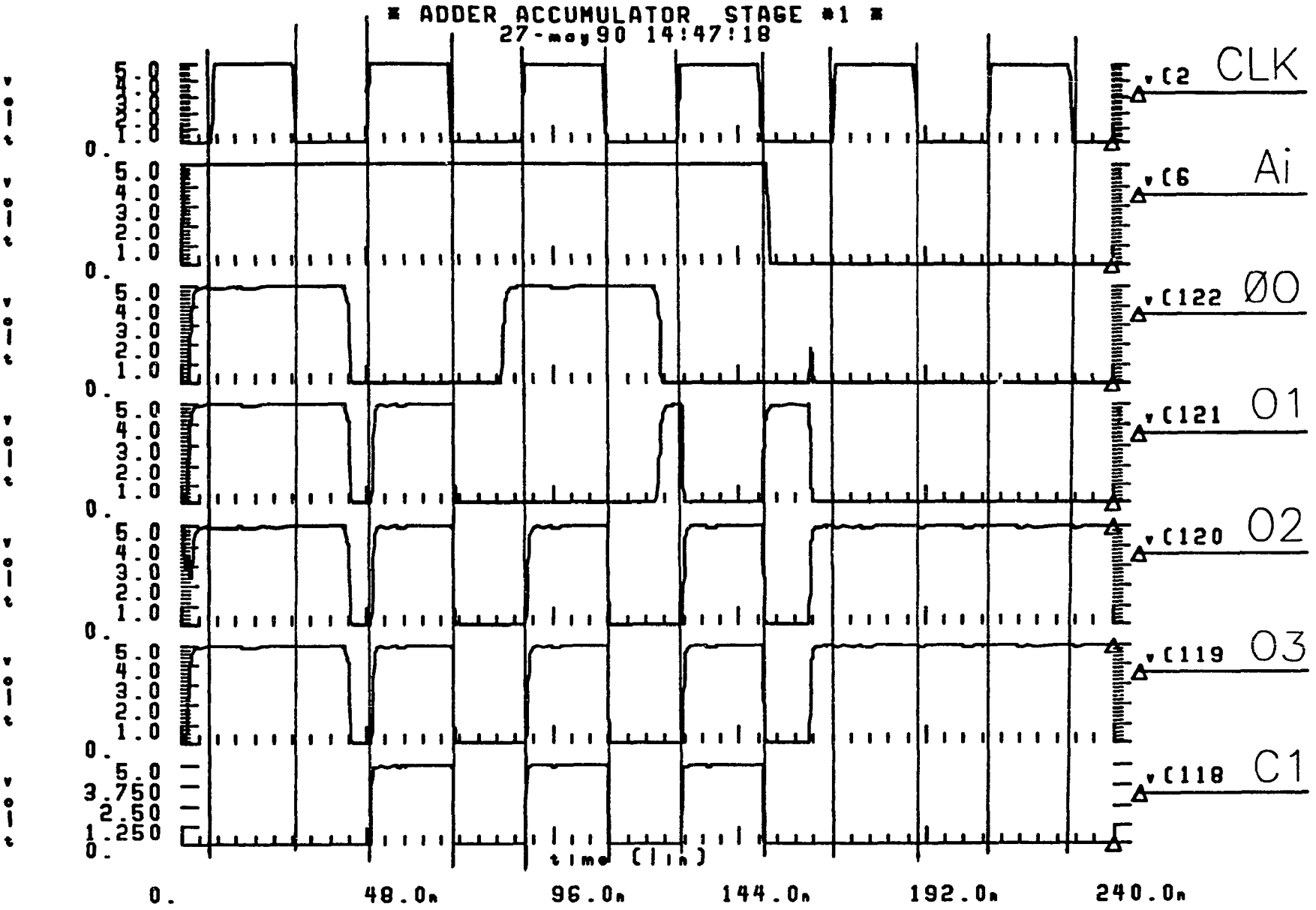


Fig.3.14.a Simulation Result of Stage#1

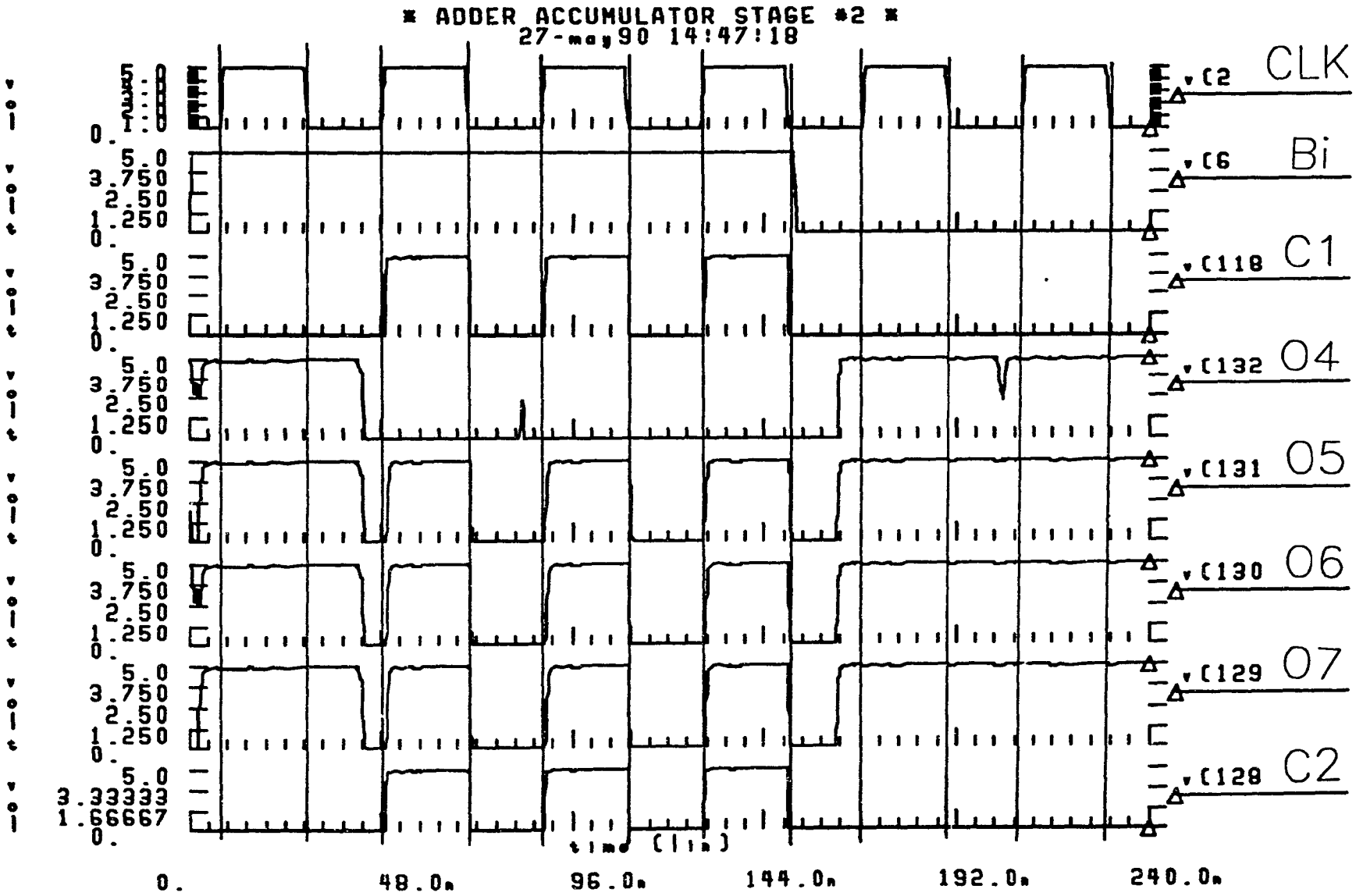


Fig.3.14.b Simulation Result of Stage#2

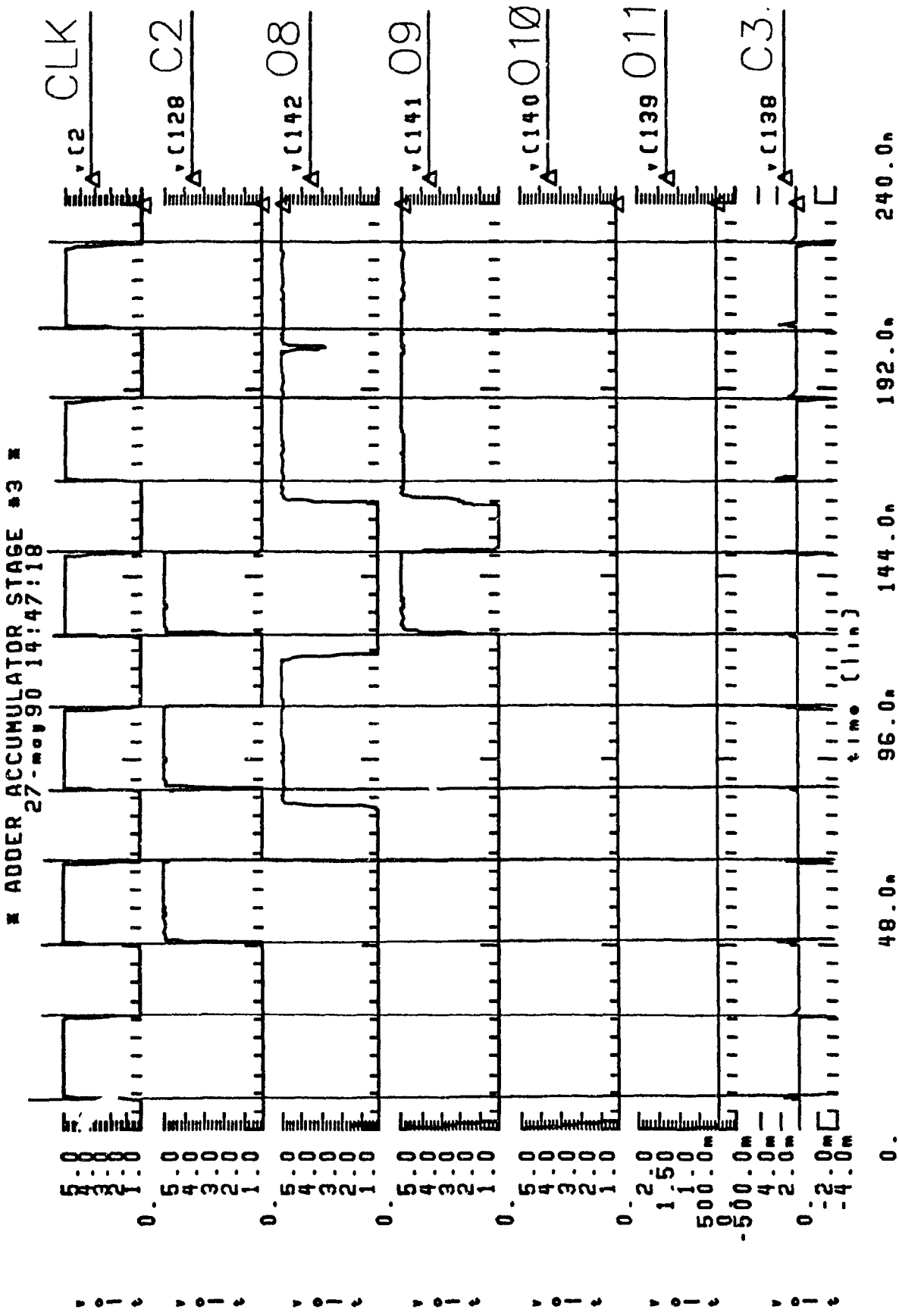


Fig.3.14.c Simulation Result of Stage#3

It is interesting to note two points in favor of this design. One is that: as the number of bits increase, because of the pipeline nature of the design, the clock rate remains constant depending solely on the speed of the full 4-bit adder, therefore, the performance is independent of the bit size apart from the flushing requirements at the end of the calculation. The second point is in regard to the area: apart from the first word size adder, which is a full adder, the rest of the adders are all partial bit adders, thus saving on area tremendously as the bit size increases.

Appendix A gives the spice decks for the 4-bit partial adder, the full 4-bit adder, and the adder accumulator.

CHAPTER 4: FAULT ANALYSIS

4.1. INTRODUCTION

In this chapter results of a complete analysis is given for the faults that might occur in both the 4-bit partial adder, and the full 4-bit adder, and which will be used as the base for designing the fault tolerant adders. Due to the bulk of the work, results of simulation runs are presented in Appendix C, with only the overall conclusion of the analysis presented in this chapter. We begin by discussing the various types of fault models, this will be followed by the analysis of the faults and the effect of these faults on the circuits' behaviour.

4.2. FAULT CHARACTERIZATION

Appendix B gives why and where physical faults occur. These physical faults are translated to stuck-at faults. Stuck-at fault model had been widely accepted in the past since it was adequate for the earlier fabrication technologies[13]. However, Wadsack[14] discovered that digital circuits implemented in CMOS technology could display fault models other that stuck-at faults. Wadsack[14] introduced the non-classical stuck-open fault. This condition is defined as the open or high state impedance. Stuck-open faults are the result of missing connections to gate, source, or drain of FET. Non-classical faults cause combinational circuits to behave in a sequential way. However, Wunderlich and Rosenstiel[15] proved that for dynamic circuits these type of faults will not cause sequential behaviour.

Relation of physical faults to fault models and their classification is given in Appendix B. In our analysis the following faults will be studied:

a. Short between gate and drain.

b. Short between gate and source.

c. Short between drain and source.

d. Open gate.

e. Open drain.

f. Open source.

g. Inputs stuck-at.

h. Outputs stuck-at.

Appendix C gives detailed analysis of the above mentioned faults, their manifestation in the adders' circuits, and the test vectors to detect them. The work is comprehensive and have been put in the appendix to focus this chapter only on the conclusion of the analysis work.

4.3. FAULT MODEL

As mentioned in the previous section, the faults that we are going to cover include shorts (between gate and drain, gate and source, and drain and source), opens (in gate, drain, and source), and stuck at faults. Based on the fault analysis supplied in Appendix C, faults manifest one of three states. The first manifestation gives the effect of the unit's output being stuck-at-1, the second gives a stuck-at-0 effect, and the third gives intermediate values (such as soft 1's, or soft 0's). A summary of these manifestations follows for various units of the 4-bit partial adder and the full 4-bit adder. For more understanding of how these conclusions were drawn, please refer to Appendix C.

4.4. FAULT MODELS FOR 4-BIT PARTIAL ADDER

In the following sections the fault model of each module of the adder is given.

4.4.a. CARRYB GENERATION

The effect of various faults is grouped into:

A. Faults that give the effect of output being stuck-at-1:

1. Inputs s-a-0.

2. Outputs s-a-1.

3. Short between gate and source in any of the nmos (this would cause the carryb's above the faulty nmos to become s-a-1, while those below will behave normally, meanwhile the carryb connected to the source of the faulty nmos, will vary between 1 and 0 depending on the value of the input).

4. Short between drain and source in the charging pmos would cause the carryb connected to the faulty pmos drain to be s-a-1, while those below it might have intermediate values depending on the inputs.

5. Open in ground connection of the discharge nmos.

6. Open in gate connection of the discharge nmos.

7. Open in input connection to any of the nmos's gate.

8. Open in source connection of any nmos.

B. Faults that give the effect of output being stuck-at-0:

1. Inputs s-a-1.

2. Outputs s-a-0.

3. Short between drain and source in any of the nmos (again here the carryb's above the faulty nmos will be affected depending on the input values, while those below it will be unaffected).

4. Open in drain connection of any pmos.

C. Faults that give outputs with intermediate values:

1. Short between drain and gate of any nmos.

2. Short between gate and source of input nmos (this would affect the carryb that is connected to the source of the faulty nmos and those below it whenever the input to that faulty nmos is 1).

3. During low clock with a short between source and drain of the discharge nmos. This, however, will have no effect on the circuit behaviour during evaluation phase, but it can be seen that the domino concept is no longer applicable.

4. Short between drain and source in charging pmos (this will affect the carryb's below the faulty pmos).

5. Open in drain connection of any nmos would cause the carryb connected to the faulty nmos to be floating, while the carryb's above it would be kept at 1.

6. Open in gate connection of any the charging pmos.

4.4.b. INVERTER

In the inverter the manifestation of faults is grouped into:

A. Faults that give the effect of outputs stuck-at-1:

1. Input s-a-0.
2. Output s-a-1.
3. Short between gate and source of nmos.
4. Short between drain and source of pmos.
5. Open in ground connection of nmos.
6. Open in nmos gate connection.

B. Faults that give the effect of output being stuck-at-0:

1. Input s-a-1.
2. Output s-a-0.

3. Short between gate and source of pmos.

4. Short between drain and source of nmos.

5. Open in Vdd connection of pmos.

C. Faults that give intermediate values for the output. This basically includes an open in the pmos gate connection.

In addition to these three groups, the inverter manifests a fourth type of error, which is the its failing to invert the input. This happens whenever a short between gate and drain occurs in either the pmos or the nmos of the inverter.

4.4.c. XOR

Due to the nature of the static XOR, even though the faults manifest the same three values of output, they cannot be grouped since they depend on the different combination of inputs and the same fault could give different manifestations with different combinations of input.

4.5. FAULT MODELS FOR FULL 4-BIT ADDER

In the following sections a summary of the manifestations of various faults in the adder's different units, and the units' fault models are presented. The effect of faults is grouped into: output stuck-at-1 effect, output-stuck-at-0 effect, and output having intermediate values.

4.5.a. GEN:P&G

The effect of various faults on the Gen:P&G block (this does not include the XOR faults' effect as they are the same as the previous XOR given in the fault tolerant 4-bit partial adder) is grouped into:

A. Faults that give the effect of output being stuck-at-1:

1. Inputs s-a-1.

2. Output s-a-1.

3. Output of small totem pole (or input to inverter) s-a-0.

4. Short between gate and source in inverter nmos.

5. Short between drain and source in any of the totem pole nmos's.

6. Short between drain and source of inverter pmos.

7. Short between gate and drain in any of the totem pole nmos's (whenever the input to that faulty nmos is 0).

8. Open drain connection of charging pmos.

9. Open in GND connection of inverter nmos.

10. Open in the gate connection of the inverter nmos

B. Faults that give the effect of output being stuck-at-0:

1. Inputs s-a-0.

2. Output s-a-1.

3. Output of small totem pole (or input to inverter) s-a-1.

4. Short between gate and source in any of the totem pole nmos's.

5. Short between gate and source of inverter pmos.

6. Short between drain and source of charging pmos.

7. Short between drain and source of inverter nmos.

8. Short between gate and drain in any of the totem pole nmos's (whenever the input is 11).

9. Open in GND connection of discharge nmos.

10. Open in the gate connection of any of the totem pole nmos's (including the discharge nmos).

11. Open in the source connection of any of the totem pole nmos's.

12. Open in the VDD connection of the inverter pmos.

C. Faults that give outputs with intermediate values:

1. Short between drain and source in discharge nmos (seen during low clock only).
2. Short between gate and drain in inverter nmos or pmos.
3. Open in gate connection of inverter pmos.
4. Open in gate connection of charging pmos.

4.5.b. GEN:CAR

The effect of various faults is grouped into:

A. Faults that give the effect of output being stuck-at-1:

1. Inputs s-a-1.
2. Outputs s-a-1.
3. Outputs of totem pole (or inputs to inverters) s-a-0.
4. Short between gate and source in inverter nmos.
5. Short between drain and source in any of the totem pole nmos's.
6. Short between drain and source of inverter pmos.
7. Open in drain connection of any of the charging pmos.
8. Open in GND connection of inverter nmos.
9. Open in gate connection of inverter nmos.

B. Faults that give the effect of output being stuck-at-0:

1. Inputs s-a-0.
2. Outputs s-a-1.
3. Outputs of totem pole (or inputs to inverters) s-a-1.
4. Short between gate and source in any of the totem pole nmos's.
5. Short between gate and source of inverter pmos.
6. Short between drain and source of any charging pmos (for the

carry's above the faulty pmos, and the carry connected to the faulty pmos drain).

7. Short between drain and source of inverter nmos.

8. Open in GND connection of discharge nmos.

9. Open in gate connection of discharge nmos.

10. Open in source connection of any of the totem pole nmos's.

11. Open in VDD connection of inverter pmos.

12. Open in gate connection of any of the totem pole nmos's.

C. Faults that give outputs with intermediate values:

1. Short between drain and source in discharge nmos (seen during low clock only).

2. Short between gate and drain in inverter nmos or pmos.

3. Short between gate and drain of any of the totem pole nmos's.

4. Short between drain and source of any of the charging pmos (for the carry's below the faulty pmos).

5. Open in gate connection of inverter pmos.

6. Open in gate connection of charging pmos.

7. Open in drain connection of any of the totem pole nmos's.

4.5.c XOR

The faults manifestation for the XOR was already given before, in the fault analysis of the 4-bit partial adder.

In the following chapter, the design of the fault tolerant adders will be given, based on the fault models that were given in this chapter.

CHAPTER 5: FAULT TOLERANT ADDERS DESIGN

5.1 INTRODUCTION

In this chapter the design and analysis of two fault tolerant adders (based on the previous design of the 4-bit partial adder and full 4-bit adder and the previous fault models) is presented. A study of the available fault tolerance techniques is first given, followed by the design and the simulation results of the two adders.

5.2. FAULT TOLERANCE TECHNIQUES

Achieving increased reliability could be done through one of two approaches: fault avoidance (fault intolerance) and fault tolerance[21]. In this section, fault tolerance techniques will be presented, as our goal is to design a fault tolerant system.

Ever since the idea of introducing redundancy to improve the reliability of systems originated in the 1950's, redundancy has been recognized as a realistic means for constructing reliable systems[22]. In fault tolerant designs redundancy is used to provide information necessary to negate the effect of the failures.

Redundancy is manifested in two ways: extra time, or extra components[21]. Time redundancy involves the repetition of computations in ways that allow errors to be detected, thus it attempts to reduce the amount of extra hardware at the expense of using extra time[23]. Component redundancy, on the other hand, involves the use of extra gates, functional modules, and the like to supply the extra information needed to guard against the effect of failures[21].

The fundamental concept of time redundancy is to perform the same computation two or more times and compare the results to determine if a discrepancy

exists. If an error is detected, the computations are repeated to see if the error still exists. This approach is seen to be useful for detecting errors resulting from transient faults.

On the other hand, alternating logic[24], recomputing with shifted operands (RESO)[25], recomputing with swapped operands (RESWO), and recomputing using duplication with comparison (REDWC)[23], have the ability to detect permanent faults. However, most of the time redundant techniques require a considerable amount of extra circuitry (100% increase in alternating logic technique to make a circuit self dual, added circuitry such as shifters and registers in order to implement RESO, RESWO), besides, being able to only detect faults and not correct them.

The time redundant approach, however, can be used for error correction, if the computation is repeated three or more times, then the results are compared, and the error is corrected using a majority voter[26]. Another method for error correction was given in[27], where the computation is carried twice to detect any discrepancy (based on RESO method), and once an error was detected, a third rotation and a third computation is carried out.

Hardware redundancy has become more practical as the semiconductor components have become smaller and less expensive. There are three basic forms of hardware redundancy[21,26].

Passive techniques, which involves masking the faults in order to hide the occurrence of faults and prevent the faults from giving erroneous results, include N modular redundancy and voting (NMR), error correcting codes, and masking logic.

Active techniques, or dynamic redundancy, achieves fault tolerance through detecting the existence of faults, and performing some action to remove the faulty hardware from the system. These techniques include reconfigurable

duplication, reconfigurable NMR, backup sparing, and graceful degradation.

Hybrid techniques combines both the passive and active approaches. Fault masking is used to prevent the faults from generating erroneous results, while fault detection, location, and recovery, are used to improve the fault tolerance by removing faulty hardware and replacing it with spares.

5.3. DESIGN OF A FAULT TOLERANT 4-BIT ADDER

In the following sections the design of a fault tolerant 4-bit partial adder is given, based on the previous design of our 4-bit partial adder. We begin by introducing the fault tolerance approach that will be implemented. The design of the various adder circuitry (used to achieve fault tolerance) is given, together with the complete design of the fault tolerant adder. Finally, simulation is carried out to see the adder's behaviour in the presence of various faults, followed by a comparison with Triple Modular Redundancy.

5.3.1. FAULT TOLERANCE APPROACH

In order to achieve fault tolerance, hardware redundancy, as opposed to time redundancy, was chosen to correct faults effects,as the area of the circuit is fairly small, besides, opting for speed.

With duplication, we can only detect the occurrence of a fault in either modules. Our approach was to add to the error detection module additional circuitry that would enable the choice of the correct value from the two available results of the two modules whenever possible.

In the design, each part of the circuit (carryb generation, inverters, and XOR's) is duplicated, then the results are compared and corrected if necessary. This approach does not allow a fault that happens in one module to propagate to the following modules thus making the correction circuits more simple, and the

percentage of fault tolerance even higher.

To decide which of the two available values is correct, we are going to use two different approaches. The first one is to make use of the fault analysis that was done, in order to determine on a statistical basis, which are the most likely faults to occur and their manifestation in the circuit, so that to choose the right value. It can be seen that this approach will not achieve a 100% fault tolerance, but, it will achieve a percentage of fault tolerance proportional to the number of most likely faults to occur, beside, being simple and easy to implement. The second one, is based on the fact that knowing the correct value, we can simply pick the correct value. This approach offers a very high percentage of fault tolerance, as the correcting circuit will be able to select the correct value of the two available results, as long as the two results do not have the same faulty value, regardless of the type of fault that caused the erroneous result, but, it can be implemented only with small circuits, as otherwise with large circuits, the involved circuitry will be too complicated.

In the design of our fault tolerant adder, the first approach was used to achieve fault tolerance in the carryb generation unit, while the second approach will be used for both the inverters set, and the XOR's set.

We assume that no faults will occur that will affect the clock signal, i.e. faults such as short between gate and source or between gate and drain of any of the charging pmos's or the discharge nmos will be considered not to occur. The second assumption is that, for the carryb generation unit, inverters and the XOR's, no two modules fail at the same time such that the values of the two outputs are erroneously identical. This assumption does not imply single fault assumption (as will be seen later, the fault tolerant addrers will be able to tolerate multiple faults), it simply assumes that if faults occur in any two modules the manifestation of the faults will not be identical (for example one

module might manifest a stuck-at-1 value, while the other might give an intermediate value, as opposed to both modules simultaneously giving a stuck-at-1, or stuck-at-0 ... etc.). The third assumption is that correcting circuitry are assumed to be fault free. One last thing is that all global input signals are buffered, so that a fault in one module (e.g. short or open to that input signal line) will not affect another module that uses the same signal.

5.3.2. FAULT CORRECTING CIRCUITS

In this section, the design for each fault correcting circuit will be given together with the various faults that it will cover.

5.3.2.a. CARRYB GENERATION CORRECTION CIRCUIT

The circuit shown in Fig.5.1, shows a simple AND circuit that would choose a 0 whenever one or both outputs from the two carryb generation circuit is 0, and 1 with both outputs equal to 1.

In this circuit we introduced an additional pmos at the output, for the sake of controlling it in such a way that it will always be charged to 1 during low CLK signal, regardless of what the values of the carryb's being compared are, so that the output does not affect the following circuits or cause discharge problems, also this will cover the problem of having a short between drain and source of discharge nmos which invalidates the domino principle. Also, an extra nmos was introduced in series with the inverter nmos, and is controlled with CLK signal, in order to avoid a direct dc path to gnd that might occur in case either carryb or carrybdup or both of them become 0's during low CLK.

The faults that will be covered by this circuit include any fault that might cause the carryb to be erroneously stuck-at-1, and faults that result in an intermediate value where the correct value is 0.

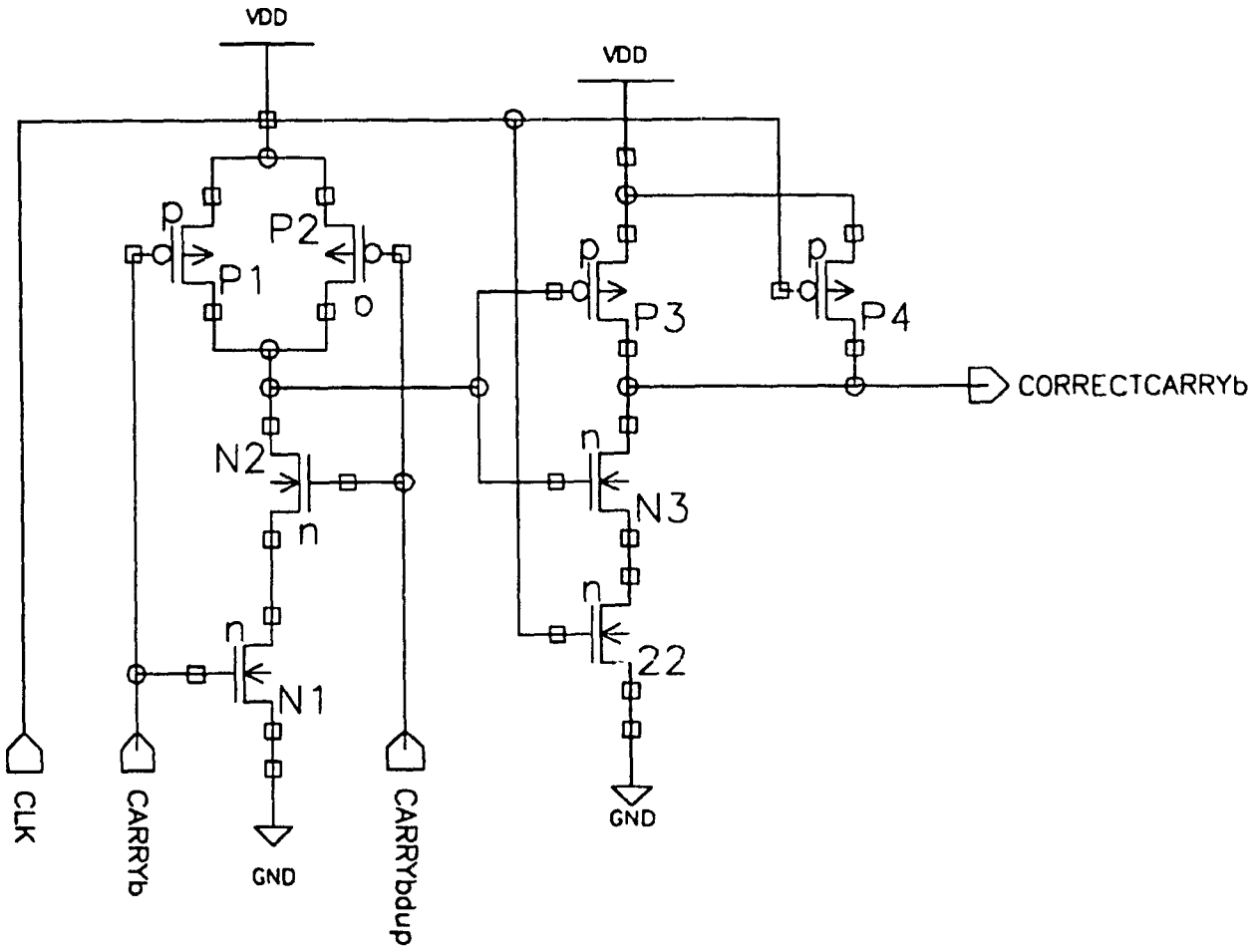


Fig.5.1 Carryb Correction Circuit (CARCORb)

Fig.5.2 shows the result of the circuit simulation.

5.3.2.b. INVERTER CORRECTION CIRCUIT

The second circuit to be discussed is the circuit used to correct the output of the inverter. As mentioned before, the design depends on the fact that knowing what the right value of the output should be, the circuit should be able to pick the correct value from the available outputs. The circuit implements the following function:

$$Correctcarry = \overline{Correctcarryb}\ (Carry1 + Carry2)$$

Where correctcarryb is the correct carryb generated by the totem pole and corrected by the carryb correct circuit. Carry1, and carry2 are two outputs from two inverters inverting simultaneously correctcarryb.

Fig.5.3 shows the inverter correction circuit.

The circuit would be able to correct any fault occurring in either inverter, so long as both inverters do not fail simultaneously with a failure that makes them both result in the same erroneous value. However, this circuit is weak due to the presence of an inverter in its serial path, which in a way defeats what we are trying to do.

Fig.5.4 shows the simulation results of the circuit. It should be noted that the test vectors given are for the sake of fully testing the circuit under all possible combinations, but in actuality, Correctcarryb will always be 1 during low CLK (it was taken care of that issue in the carryb correction circuit), thus avoiding a direct dc path to gnd when both carry and carydup are equal to 1 during low CLK.

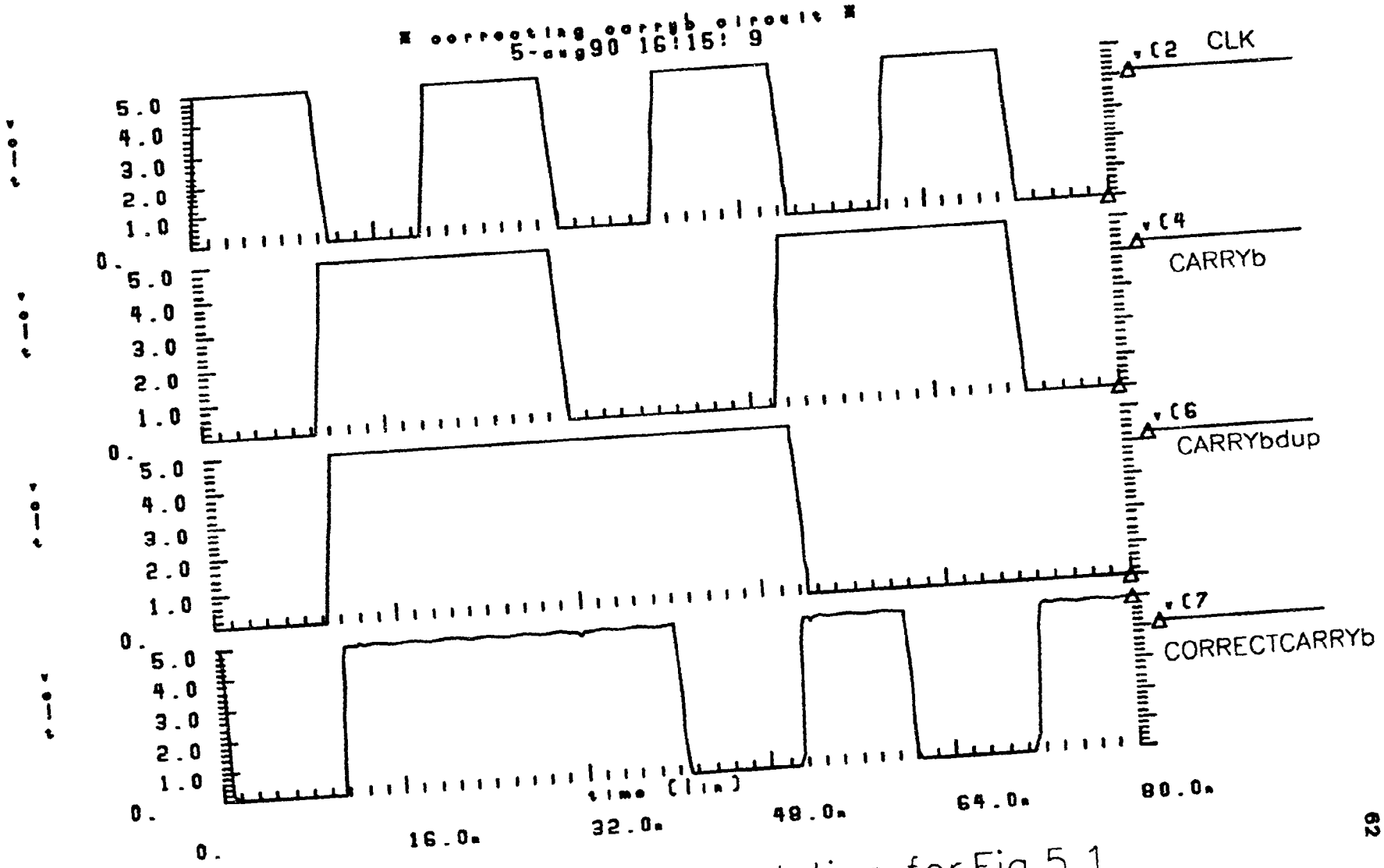


Fig.5.2 Circuit Simulation for Fig.5.1

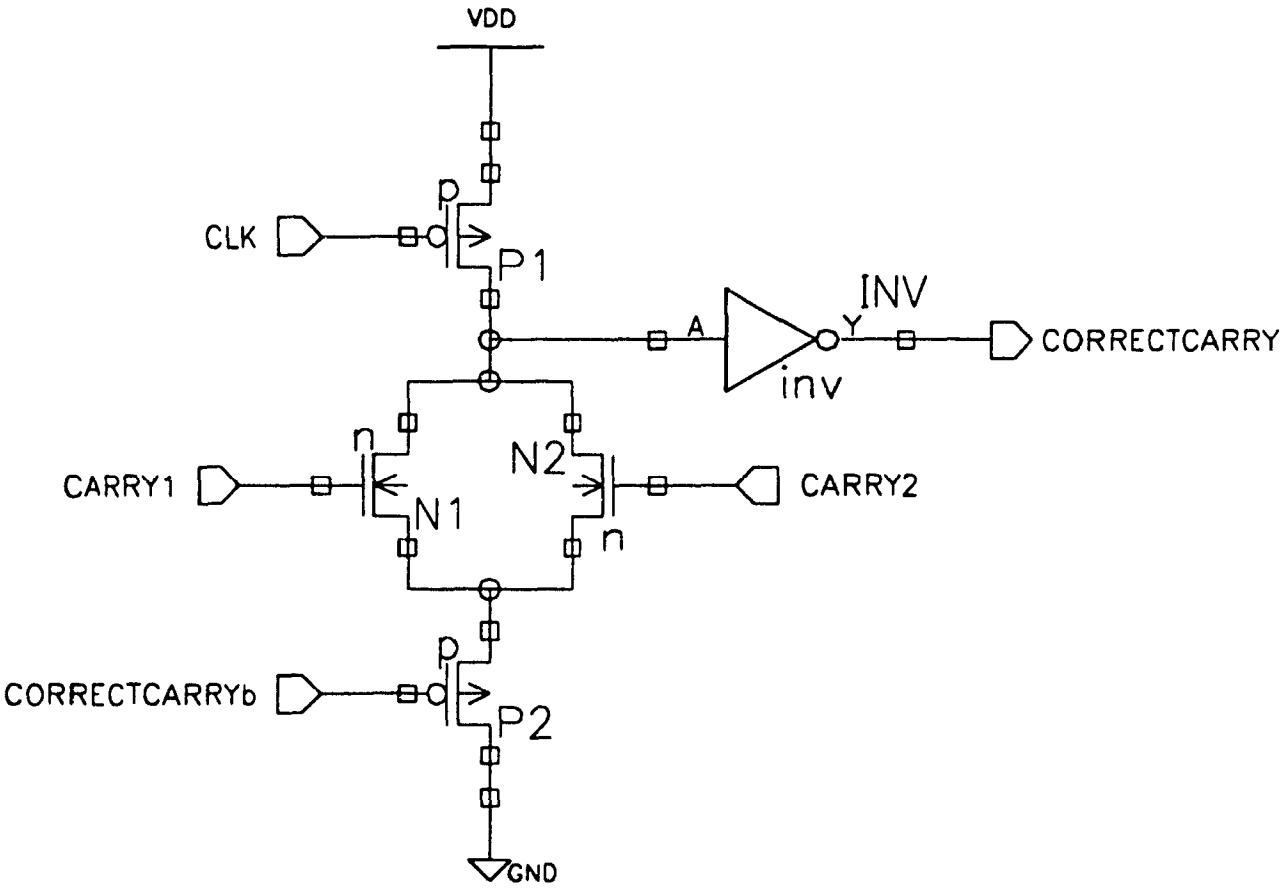


Fig.5.3 Inverter Correcting Circuit (INVCOR)

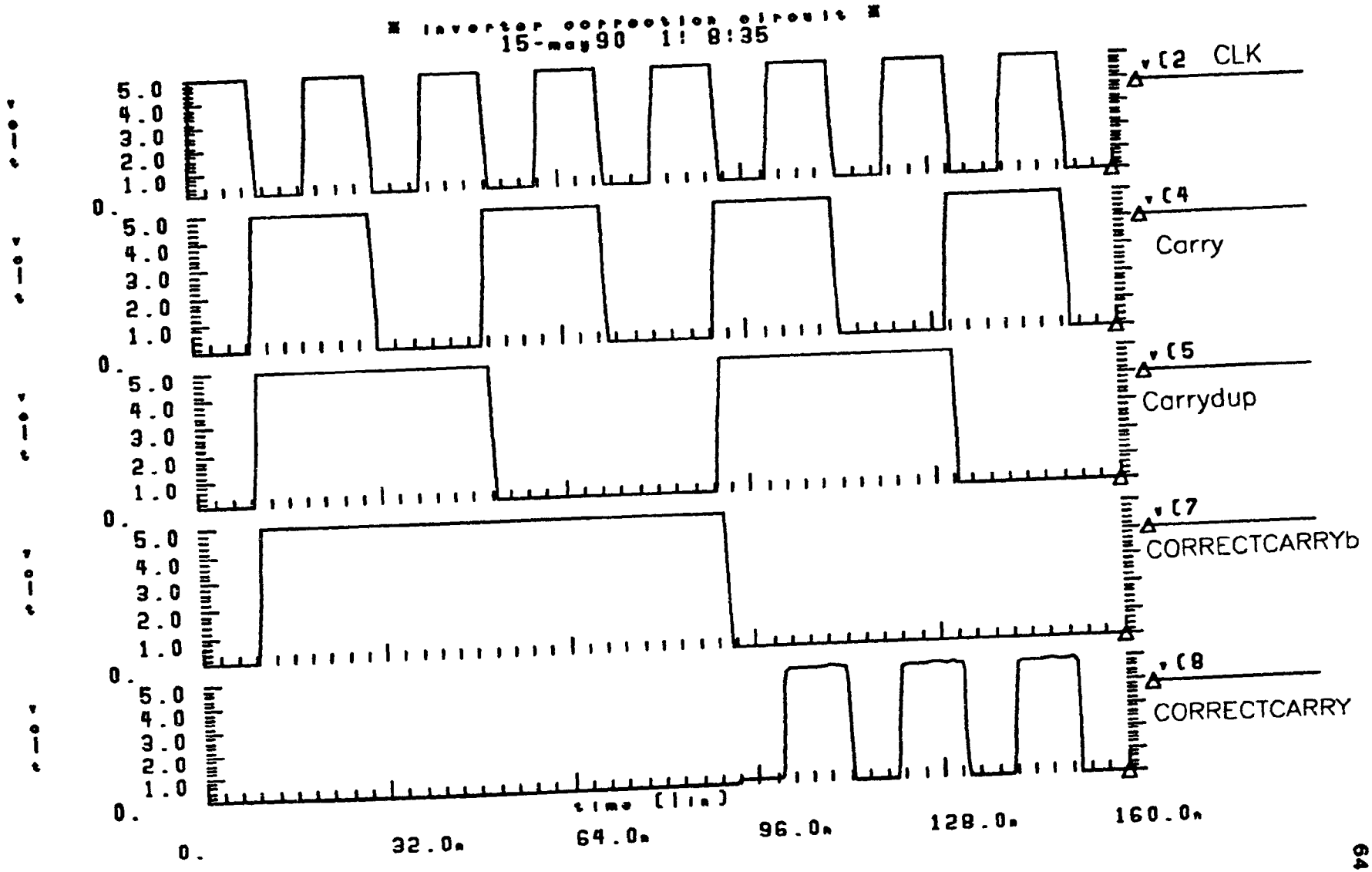


Fig.5.4 Circuit Simulation for Fig.5.3

5.3.2.c. XOR CORRECTION CIRCUIT

Fig.5.5 shows a slightly modified circuit for the static XOR that was used in the original adder circuit. An extra nmos (that is controlled by CLKb) is added at the output. The reason for that modification is to maintain the XOR output at 0 during low CLK so that no discharging problem would occur in the following XOR correction circuit.

As in the case of the inverter correction circuit, we again use the fact that knowing both inputs to the XOR, the correct resulting value could be known. With the outputs from two identical XOR's available, the XOR correction circuit implements the function:

$$Correctsum_i = (A_i \oplus C_{i-1})\,(Sum\,1 + Sum\,2)$$

Where C is the correctcarry resulting from the previous stage, sum1 and sum2 are the two outputs of two duplicate XOR's that are to be compared.

Fig.5.6 shows the XOR correction circuit.

It could be seen that N3 and N4 implement the exclusive or part of the function, while N1 and N2 implement the oring part of it. Again the XOR correction circuit is able to cover any type of faults occurring in either XOR, even if different faults occur simultaneously in both XOR, so long as both XOR's do not result in identical erroneous results.

Fig.5.7 shows the simulation results for the circuit. Again, it should be noted here that the test vectors given are for the sake of fully testing the circuit under all possible combinations, but in actuality, Sum1 and Sum2 will always be 0 during low CLK (it was taken care of that by adding extra nmos in XOR circuit as mentioned before), thus avoiding any discharge during low CLK.

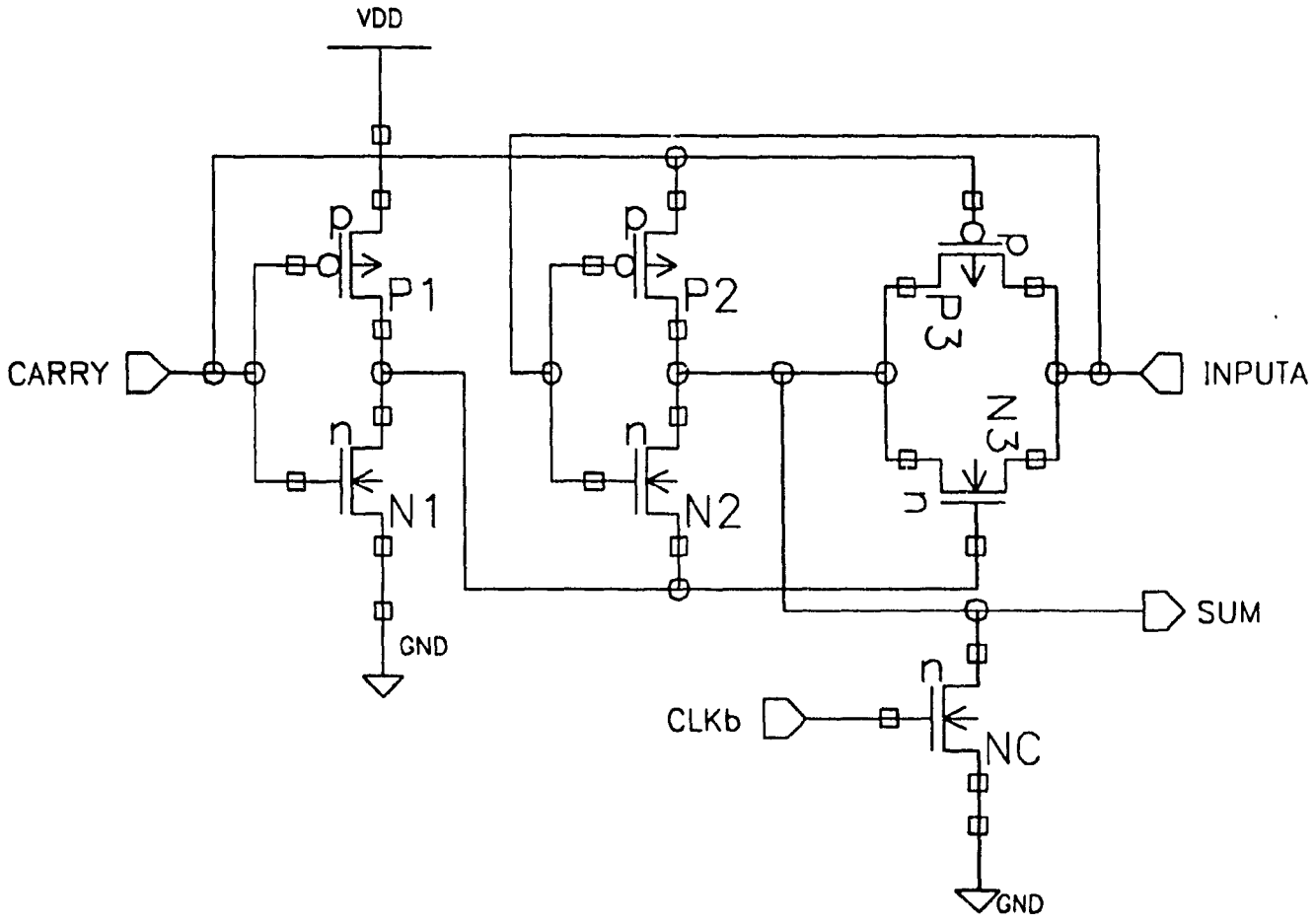


Fig.5.5 Modified Exclusive Or Circuit

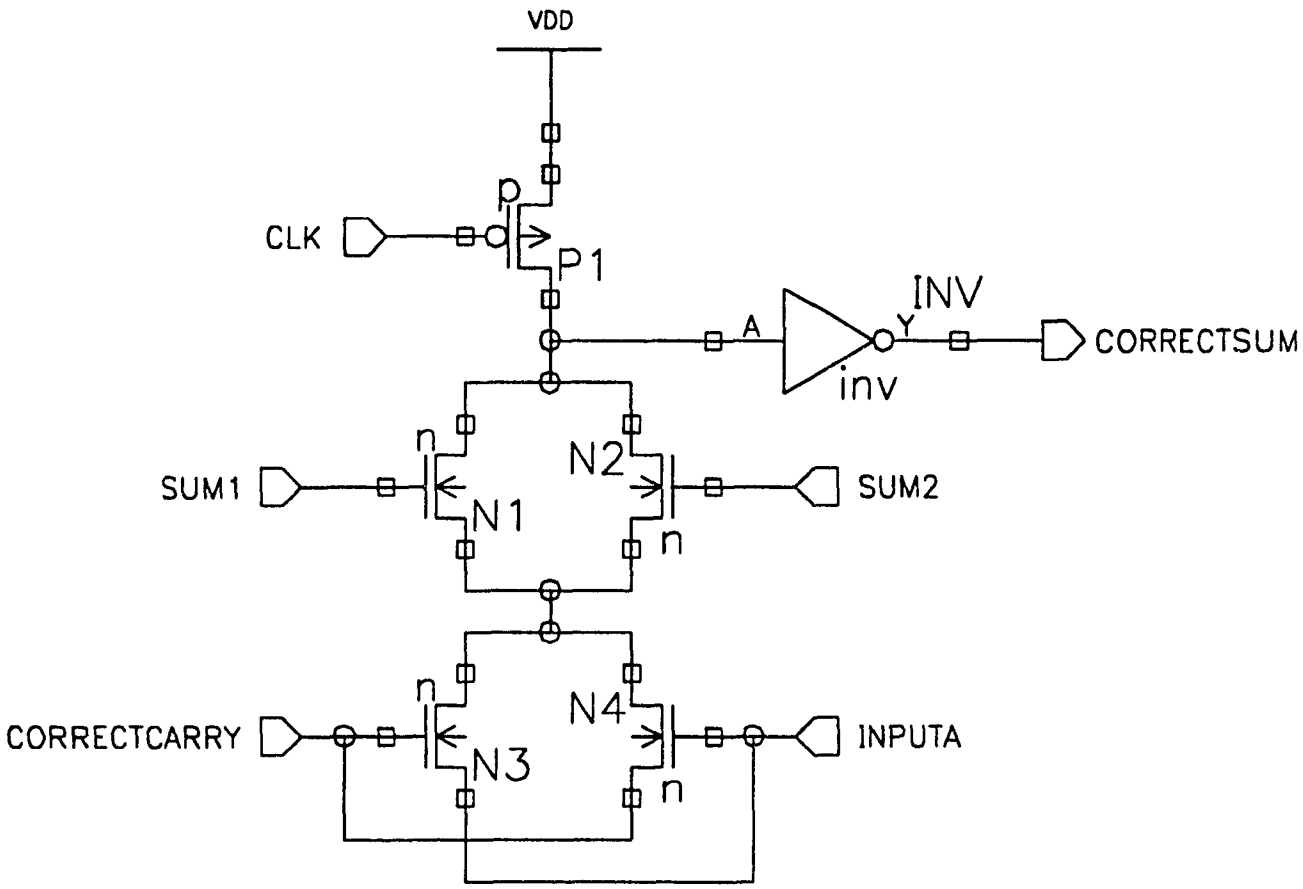


Fig.5.6 XOR Correcting Circuit(XORCOR)

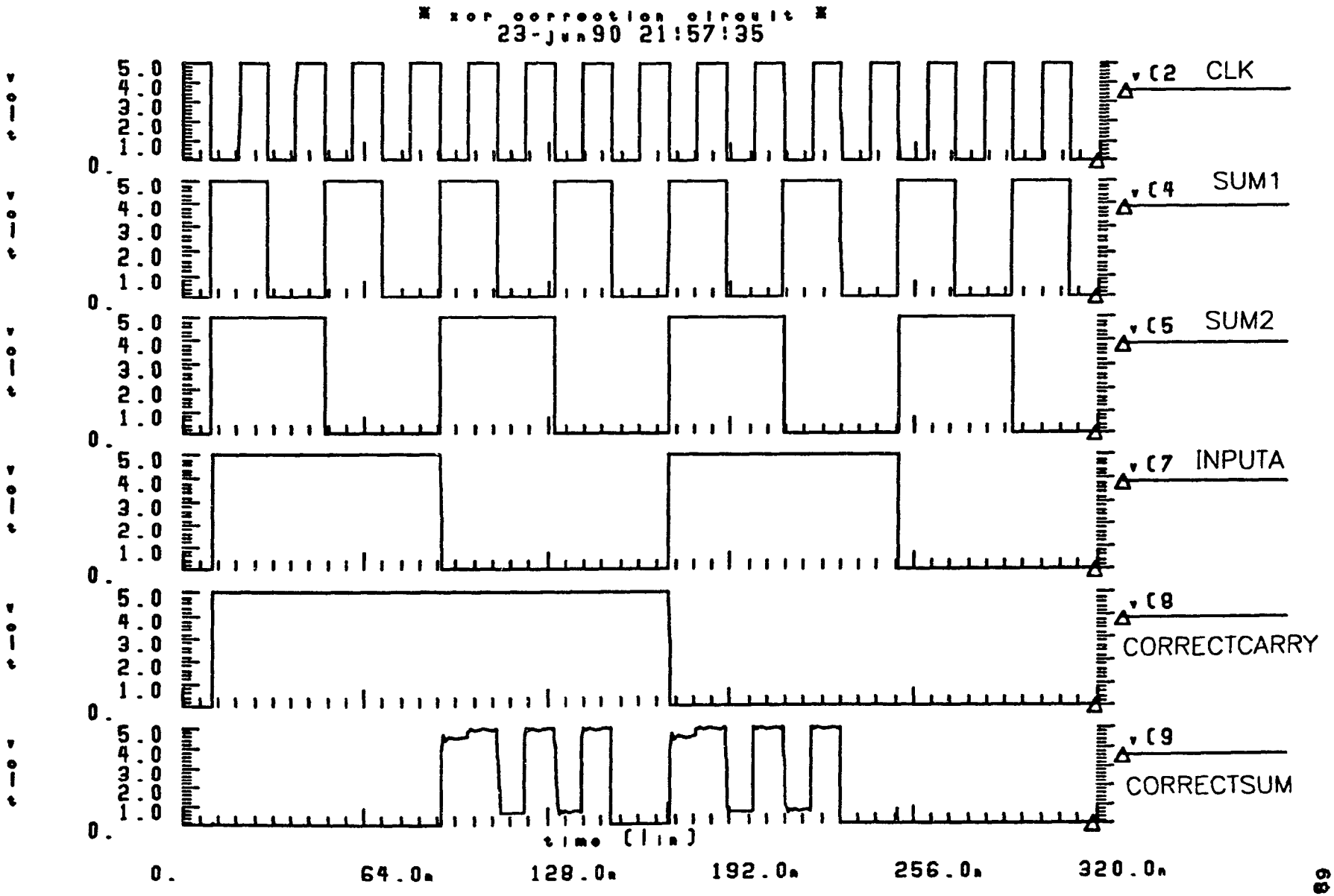


Fig.5.7 Circuit Simulation for Fig.5.6

5.3.2.d. FAULT TOLERANT 4-BIT PARTIAL ADDER

Fig.5.8 shows the complete adder, while Fig.5.9.a shows a set of worst case test vectors. Results of simulation are shown in Fig.5.9.b. From the simulation results, the complete adder gives a worst case propagation delay of 1.8ns. Worst case delay is considered to be for the carry signal to be produced correctly (that is from CLK to Carry4). This comes from 1.2ns delay to get the Carry4b correctly, plus 0.6ns delay to get the correct result from the inverter correcting circuit. The reason we did not take the CLK to Sum4 propagation delay as the worst case delay, is that, of the added circuitry, the output of XOR is kept low during low CLK, also, the carry signal being fed to the XOR is kept at 0 during low clock. So even though input A4 might reach the XOR during low clock, the output will be kept at 0 during that duration, therefore, abolishing the need to discharge the XOR output during evaluation period which was the case with our non-fault tolerant adder.

5.3.3. FAULT SIMULATION

Next, a set of faults were introduced to the circuit. These faults are:

1. An open fault in the drain connection of nmos N4, in carryb generation circuit (Fig.5.10).
2. A short between drain and source in discharge nmos NDd, in duplicate carryb generation circuit (Fig.5.10).
3. A short between gate and source of nmos Ni1,in inverter circuit(Fig.5.11).
4. A short between drain and source of nmos Ni1d, in duplicate inverter circuit (Fig.5.11).
5. A short between drain and source of nmos Nx1 in xor circuit (Fig.5.12).
6. A short between gate and source of nmos Nx2d in duplicate xor circuit (Fig.5.12).

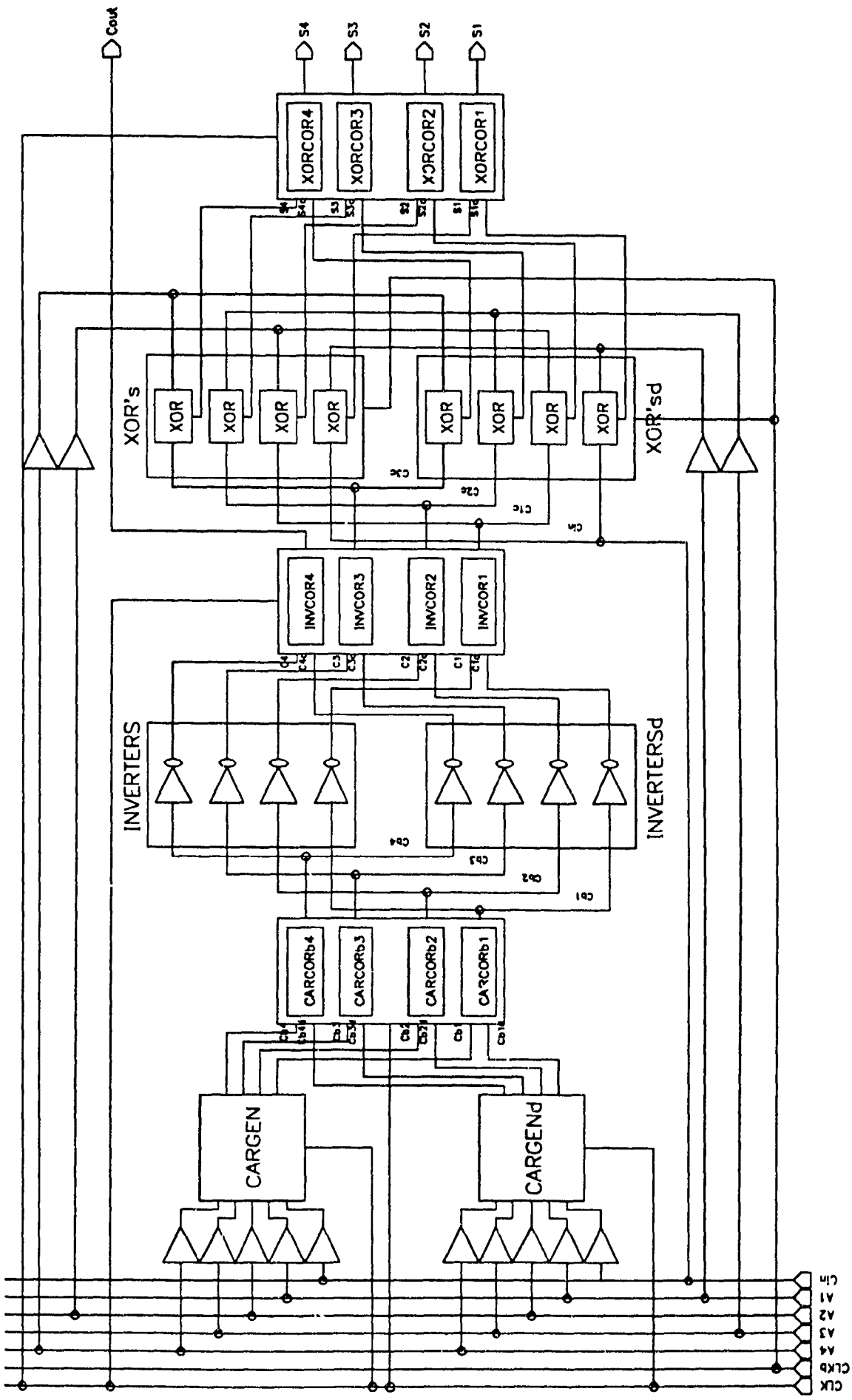


Fig.5.8 Fault Tolerant 4-bit Partial Adder(FT4BADDER)

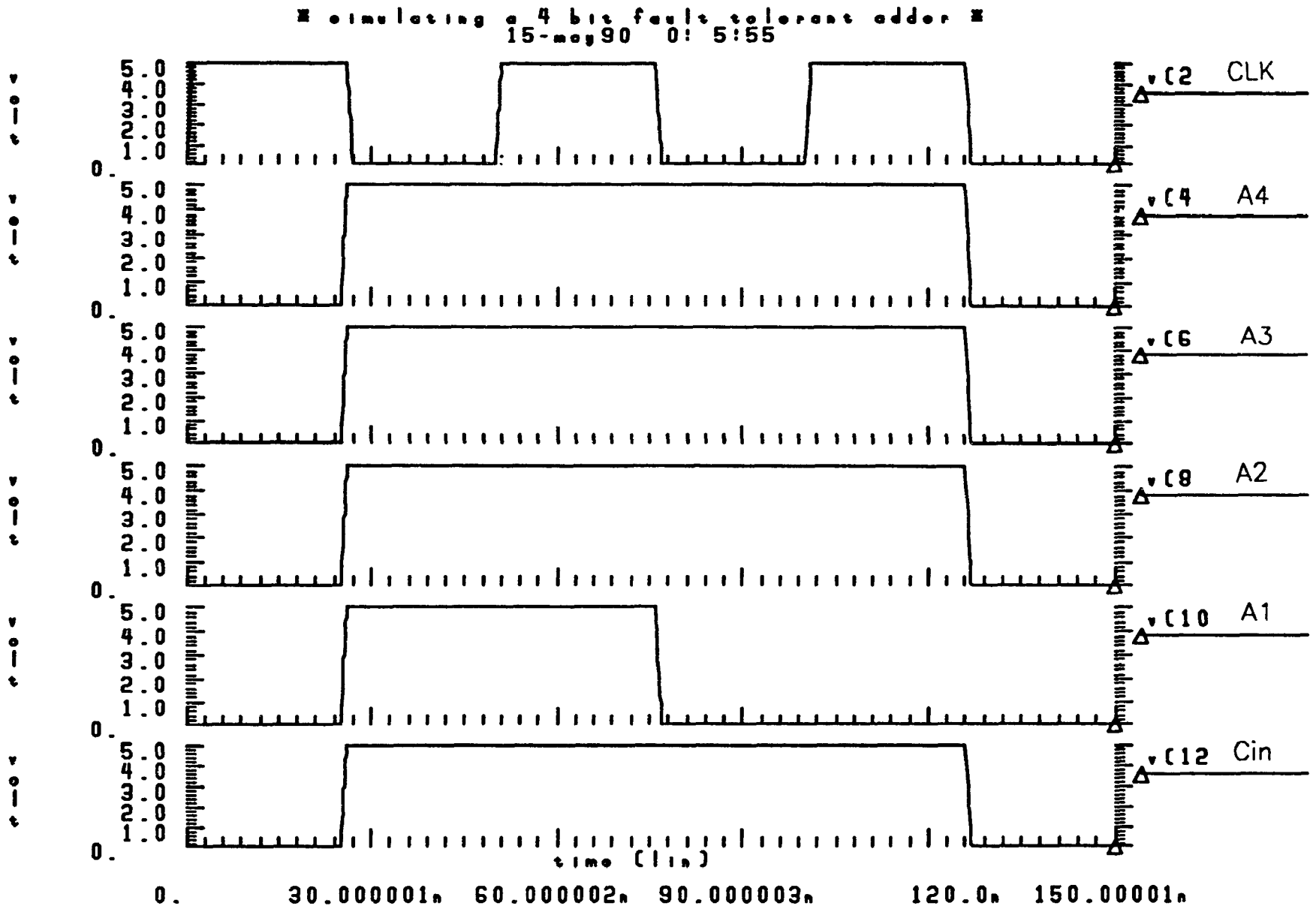


Fig.5.9.a Input Test Vectors for FT4BADDER

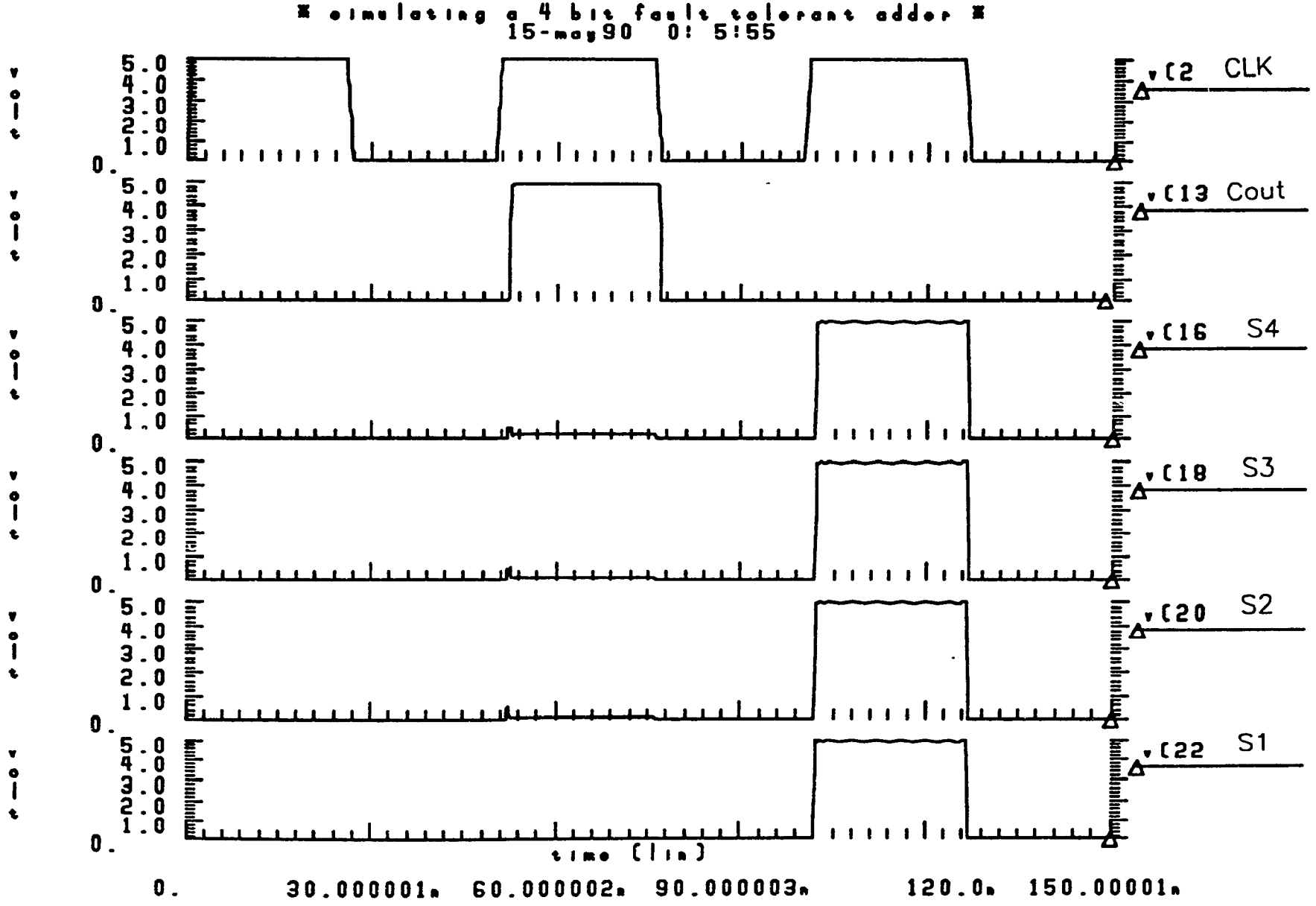
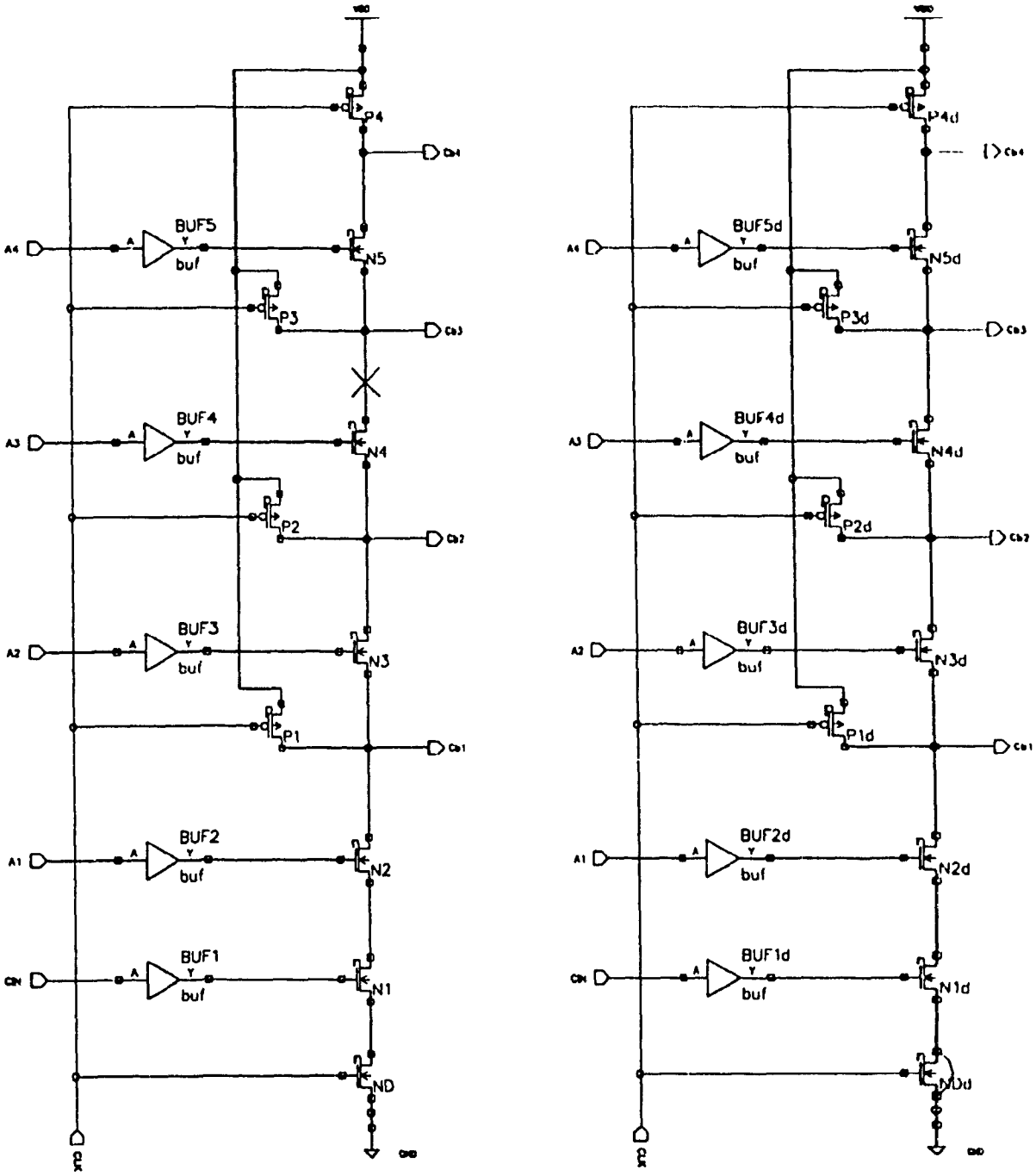


Fig.5.9.b Output Results of FT4BADDER in Response to Fig.5.9.a

```
* PROPAGATION DELAY BETWEEN CLK AND COUT*
delay1          =  1.8013E-09  targ=  5.2301E-08  trig=  5.0500E-08

*PROPAGATION DELAY BETWEEN CLK AND SUM4*
delay2          =  1.5870E-09  targ=  1.0209E-07  trig=  1.0050E-07

*PROPAGATION DELAY BETWEEN CLK AND SUM3*
delay3          =  1.5870E-09  targ=  1.0209E-07  trig=  1.0050E-07

*PROPAGATION DELAY BETWEEN CLK AND SUM2*
delay4          =  1.5870E-09  targ=  1.0209E-07  trig=  1.0050E-07

* PROPAGATION DELAY BETWEEN CLK AND SUM1*
delay5          =  1.5625E-09  targ=  1.0206E-07  trig=  1.0050E-07
```

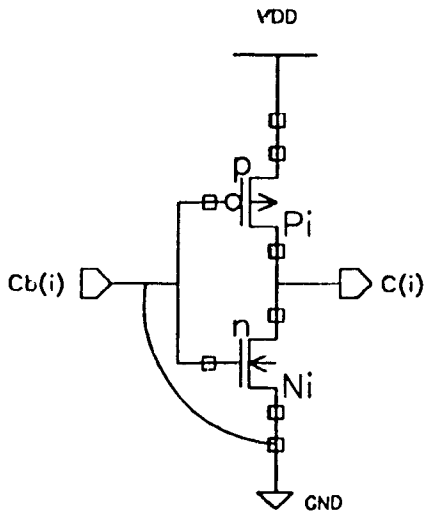
Fig.5.9.c Summary of Simulation Result



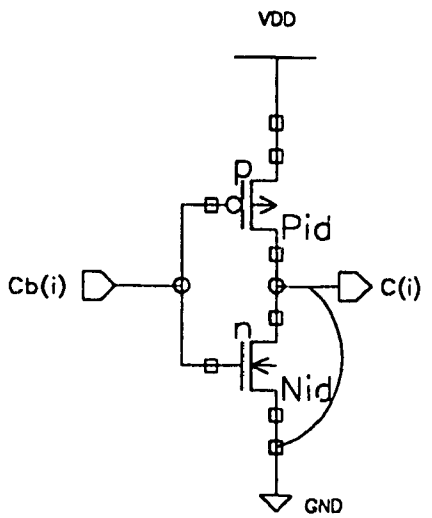
Carryb Generation

CarrybDup. Generation

Fig.5.10 Simulated Faults in Carryb Generation Block

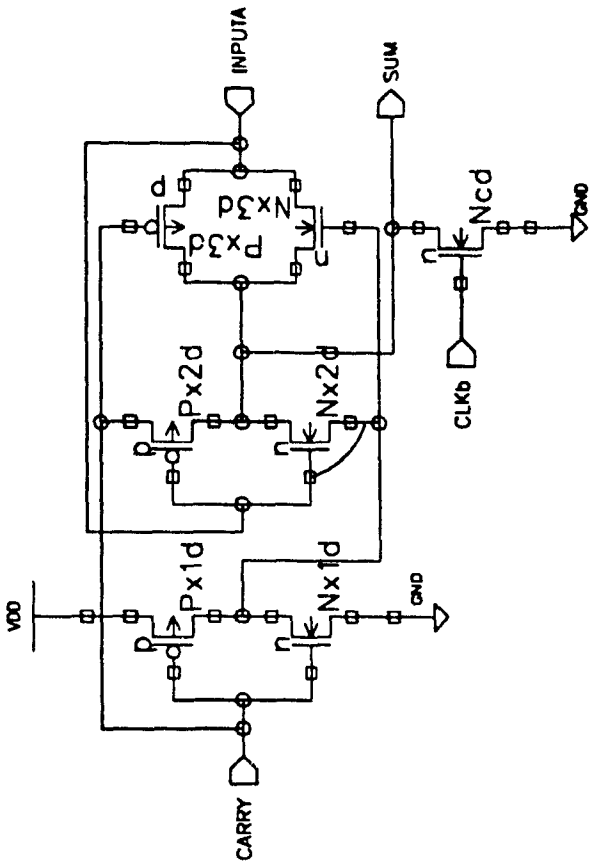


Inverter

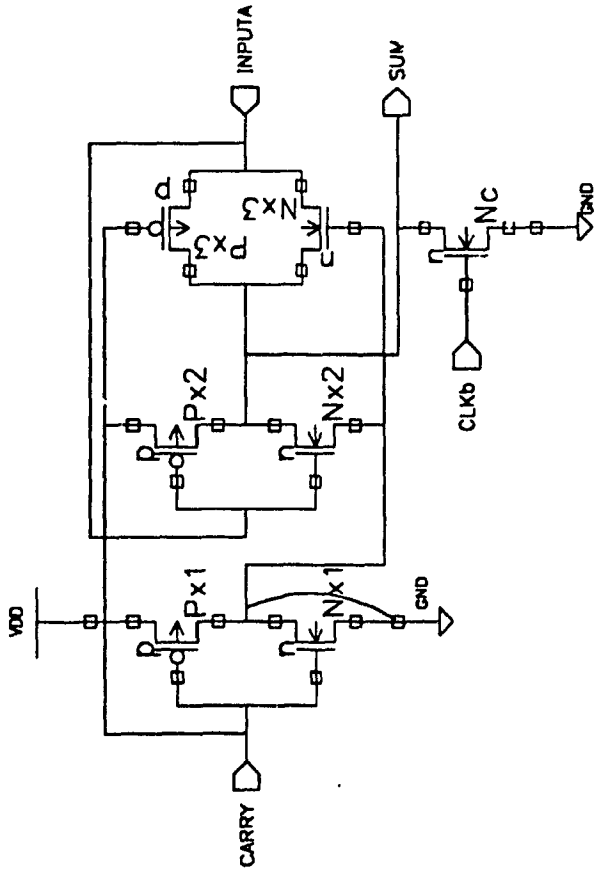


Inverterdup.

Fig.5.11 Simulated Faults in Inverter Block



XORdup.



XOR

Fig.5.12 Simulated Faults in XOR Block

The effect of these faults could be seen as following:

A. For carryb generation circuit:

1. Due to the open drain connection in the first circuit carryb4 and carryb3, will always have a value of 1, which should have been 0 with test vector 11111. However, carryb2 and carryb1 will not be affected by that fault.

2. Due to the short between drain and source of discharge nmos in the duplicate circuit, the carryb's will never charge up during low clock.

B. For inverter circuit:

1. Due to short between gate and source of nmos in first inverter, the output will be always stuck-at-1.

2. Due to short between drain and source of nmos in duplicate inverter, the output will be always stuck-at-0.

It should be noted that for the sake of simulation it is assumed that the faults in both inverter and duplicate inverter are repeated for the set of 4 inverters generating the carry's.

C. For exclusive or circuit:

1. Due to short between drain and source of nmos in the first inverter of the xor, the output of this inverter is always kept at 0, thus disabling the transmission gate nmos and causing the output to be a soft 0 (1v) to result with a 10 input to the xor.

2. Due to the short between gate and source of nmos in the second inverter of duplicate xor, with a 11 input to the xor, the transmission gate nmos will be turned on thus passing a faulty 1 to the output.

From the simulation results it could be seen that the carrryb correction circuit was able to correct the faults that occurred in the carryb generation circuit and the duplicate one, thus passing a fault free set of carryb's to the inverter

sets. Fig.5.13.a, 5.13.b, 5.13.c, and 5.13.d show the simulation results.

In the inverter stage, even though both inverters were giving stuck-at output values, however, the inverter correction circuit was able to correct these faults, and passed a set of correct carry's to the exclusive or stage. Fig.5.14.a, 5.14.b, 5.14.c, and 5.14.d show the simulation results.

In the third and final stage of the adder, the xor correction circuit corrected the faults occurring in both xor and duplicate xor, by depending on the fact that we have both inputs to the xor fault free and comparing the two available outputs to determine the correct sum values. Fig.5.15.a, 5.15.b, 5.15.c, and 5.15.d show the simulation results.

Fig.5.16, shows the final results of the simulation of the complete circuit (with faults), and it could be seen that all outputs are correct and match the fault free circuit results, in term of values and propagation delays.

5.3.4. COMPARISON

It could be seen that in dividing the original adder into three parts, the design of fault correction circuits became less complicated, and the fault coverage inadvertently increased.

In the circuit 176 transistors were used, which would compare, in terms of area, to 258 transistors that would have been used had we implemented TMR for each part of the adder as we did in our fault tolerance approach, and which would compare, in terms of area and fault coverage, to 188 transistors that would have been used had TMR been implemented for the adder as one part.

In terms of speed our circuit compares favorably with TMR implemented on the original circuit as one part. In our circuit the worst case propagation delay is 1.8ns which compares to 1.6ns for TMR. Even though our circuit is 0.2ns slower than TMR, our implementation is better if area, speed, and fault coverage are

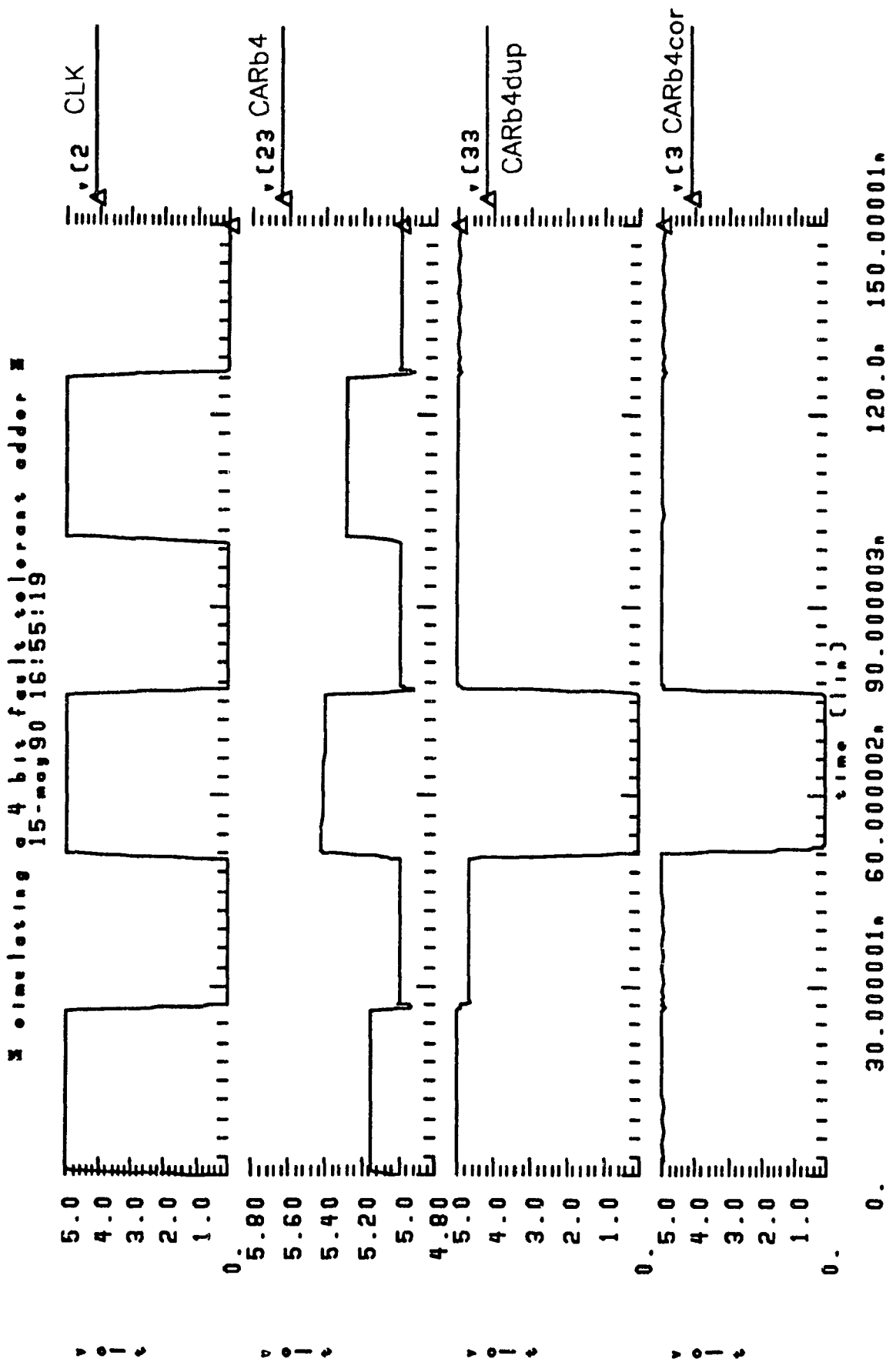


Fig.5.13.a Correcting Carry4b

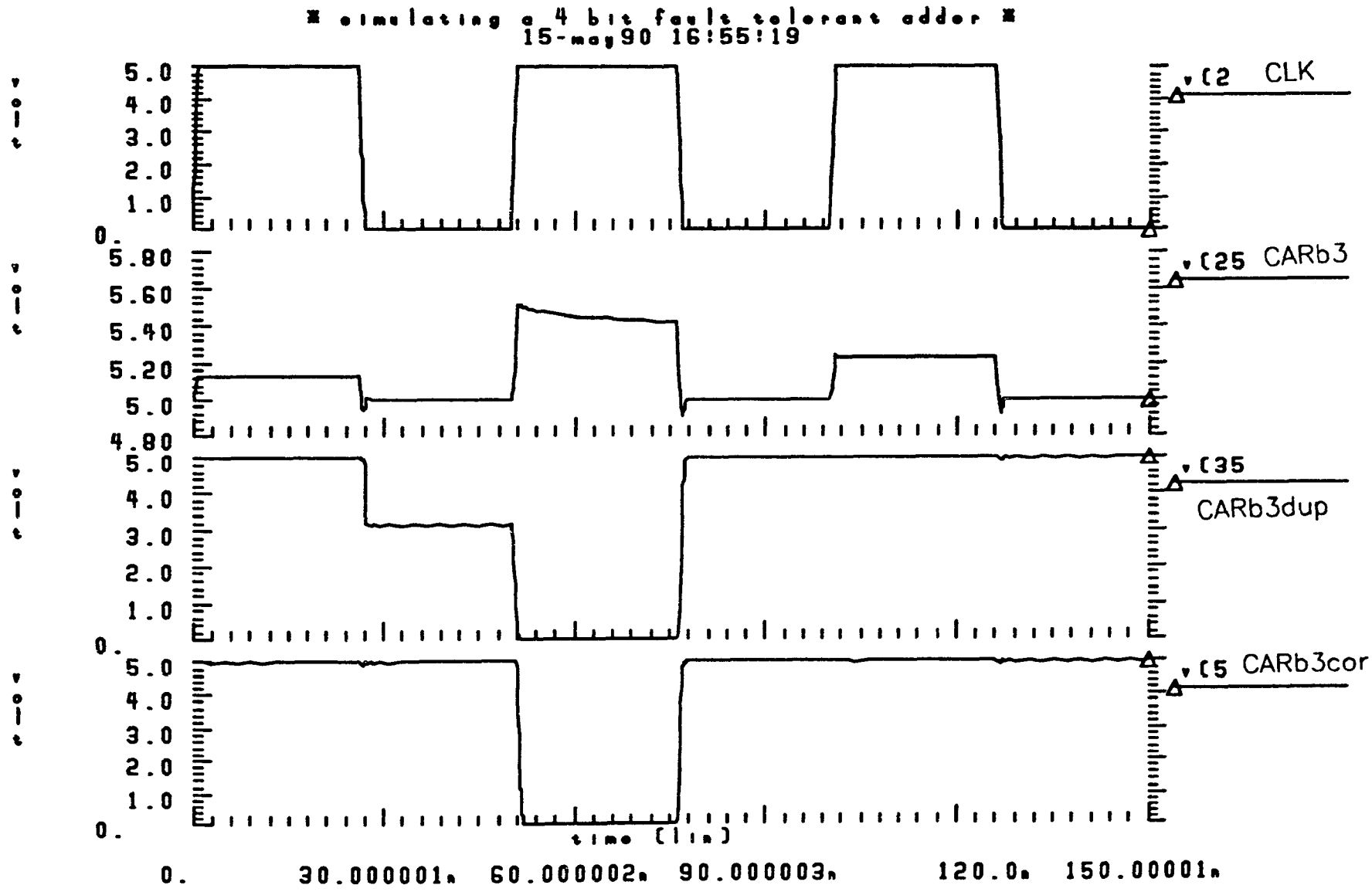


Fig.5.13.b Correcting Carry3b

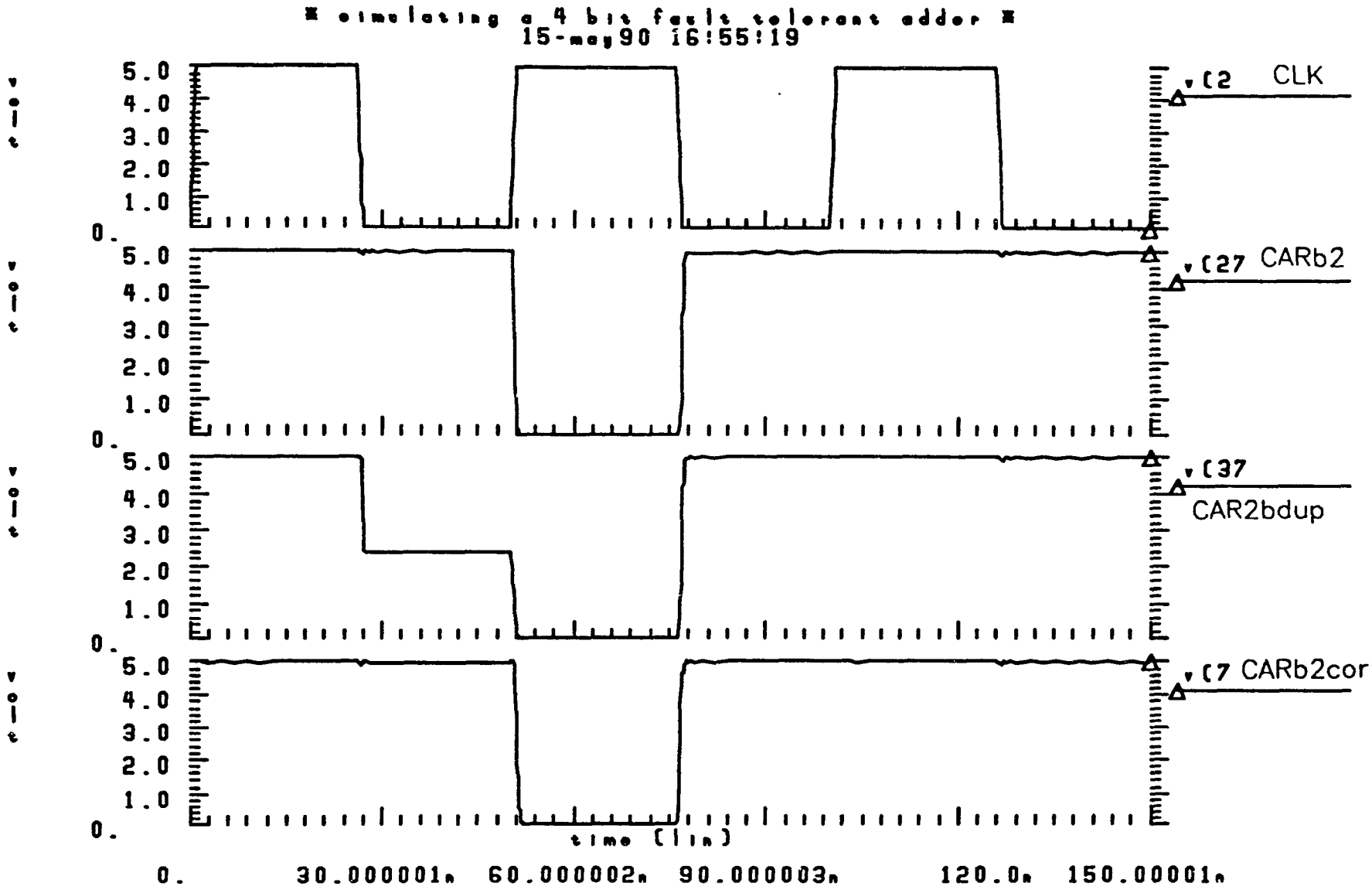


Fig.5.13.c Correcting Carry2b

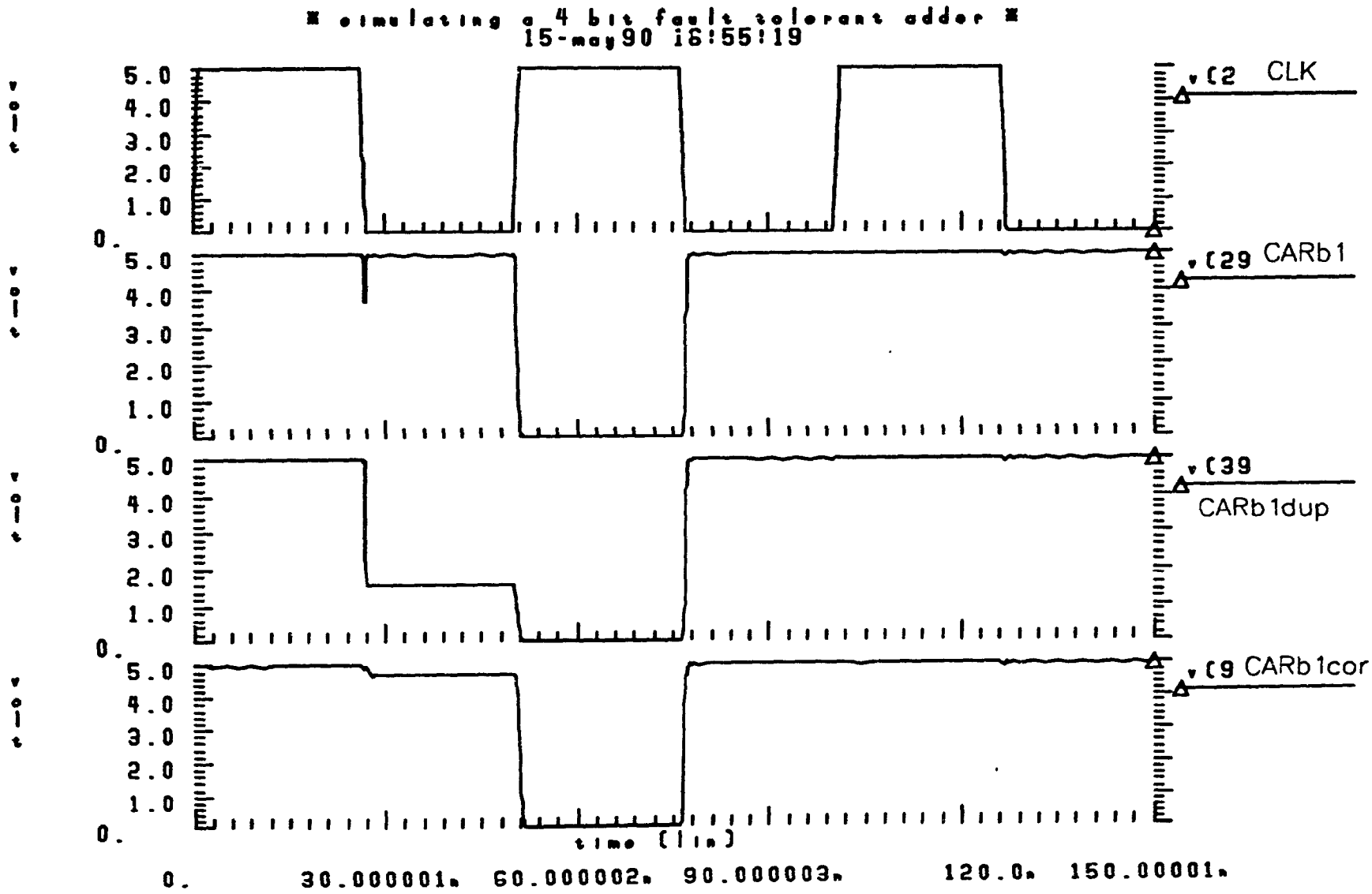


Fig.5.13.d Correcting Carry1b

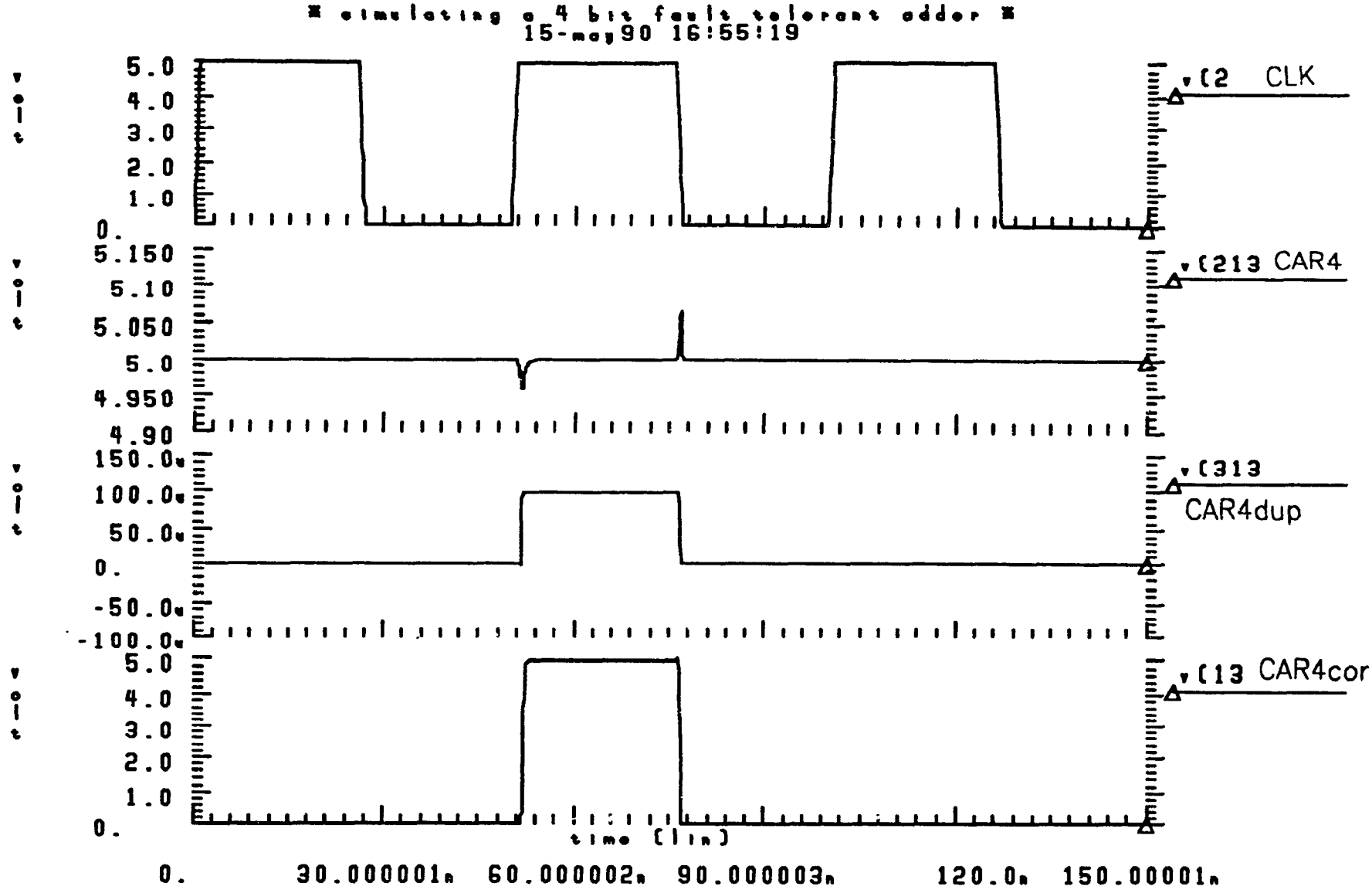


Fig.5.14.a Correcting Carry4

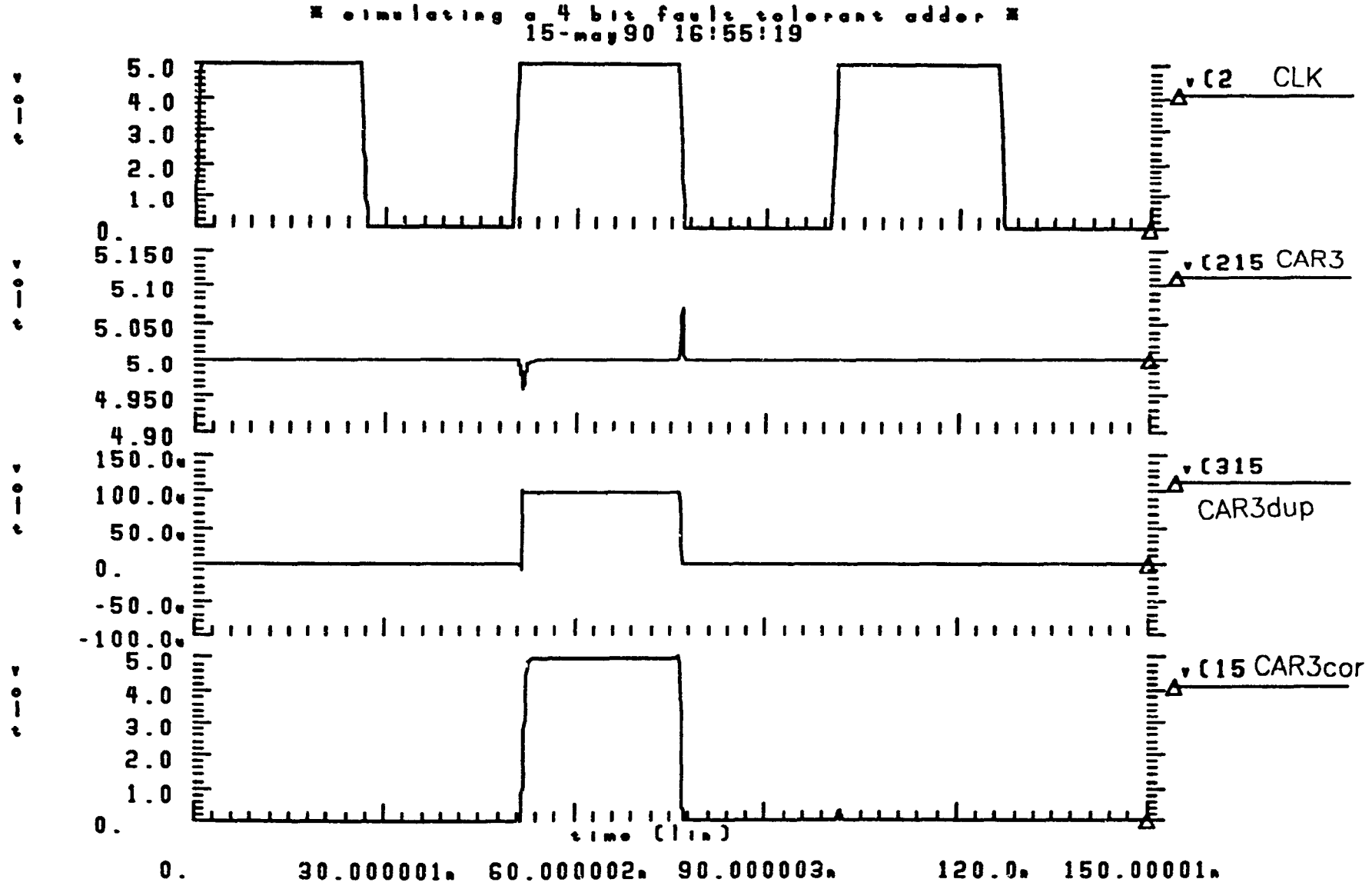


Fig.5.14.b Correcting Carry3

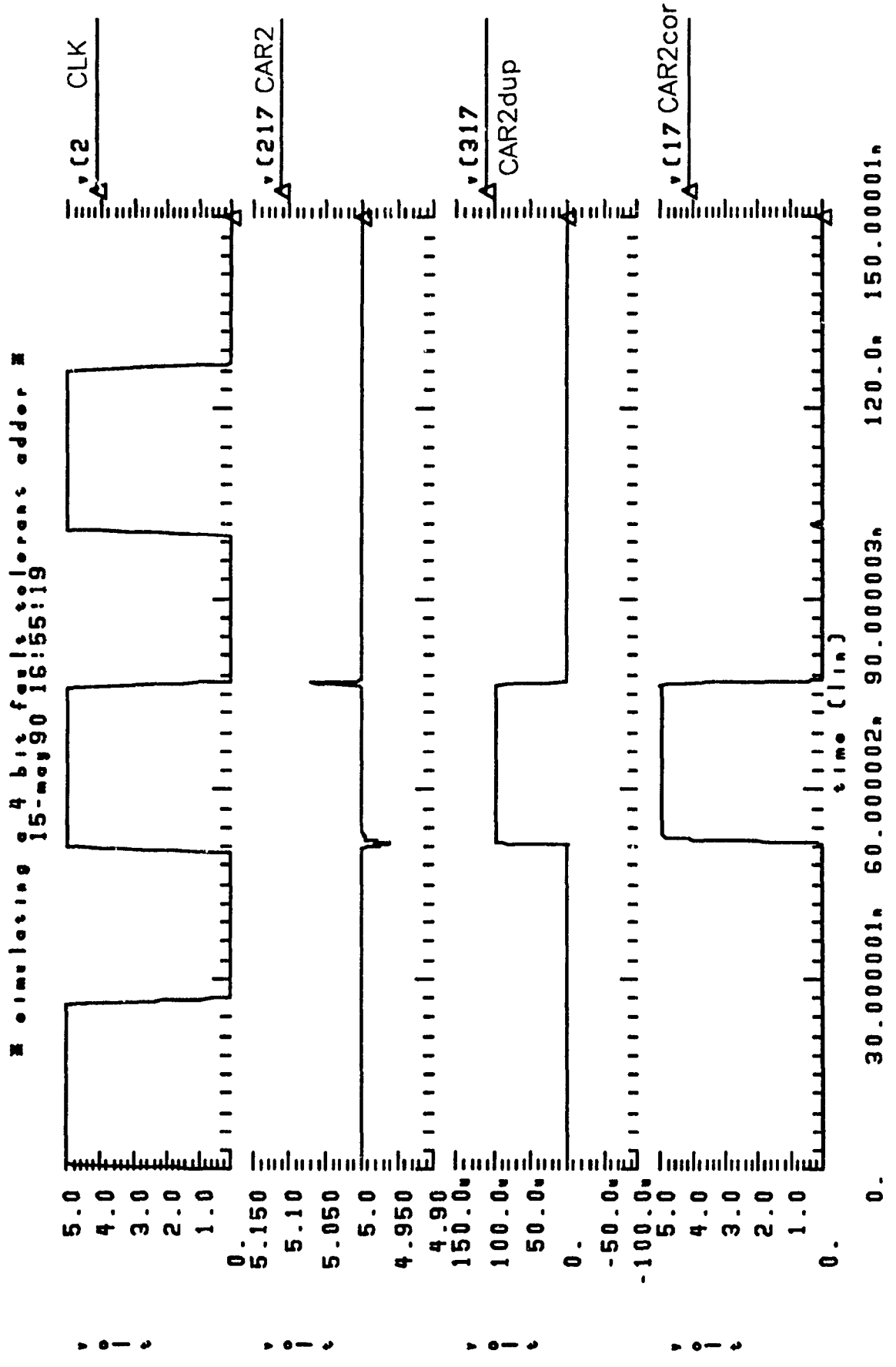


Fig.5.14.c Correcting Carry2

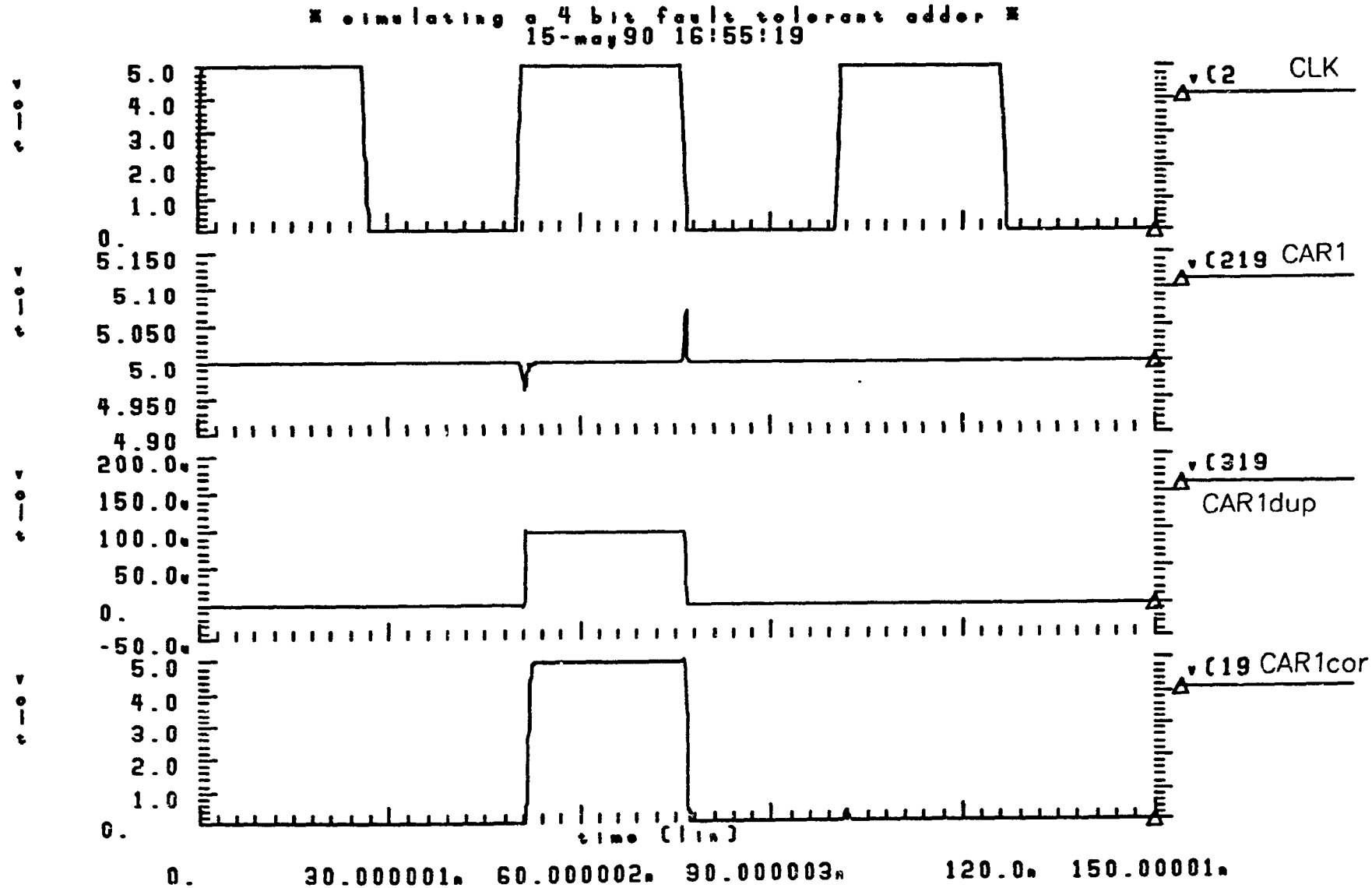


Fig.5.14.d Correcting Carry1

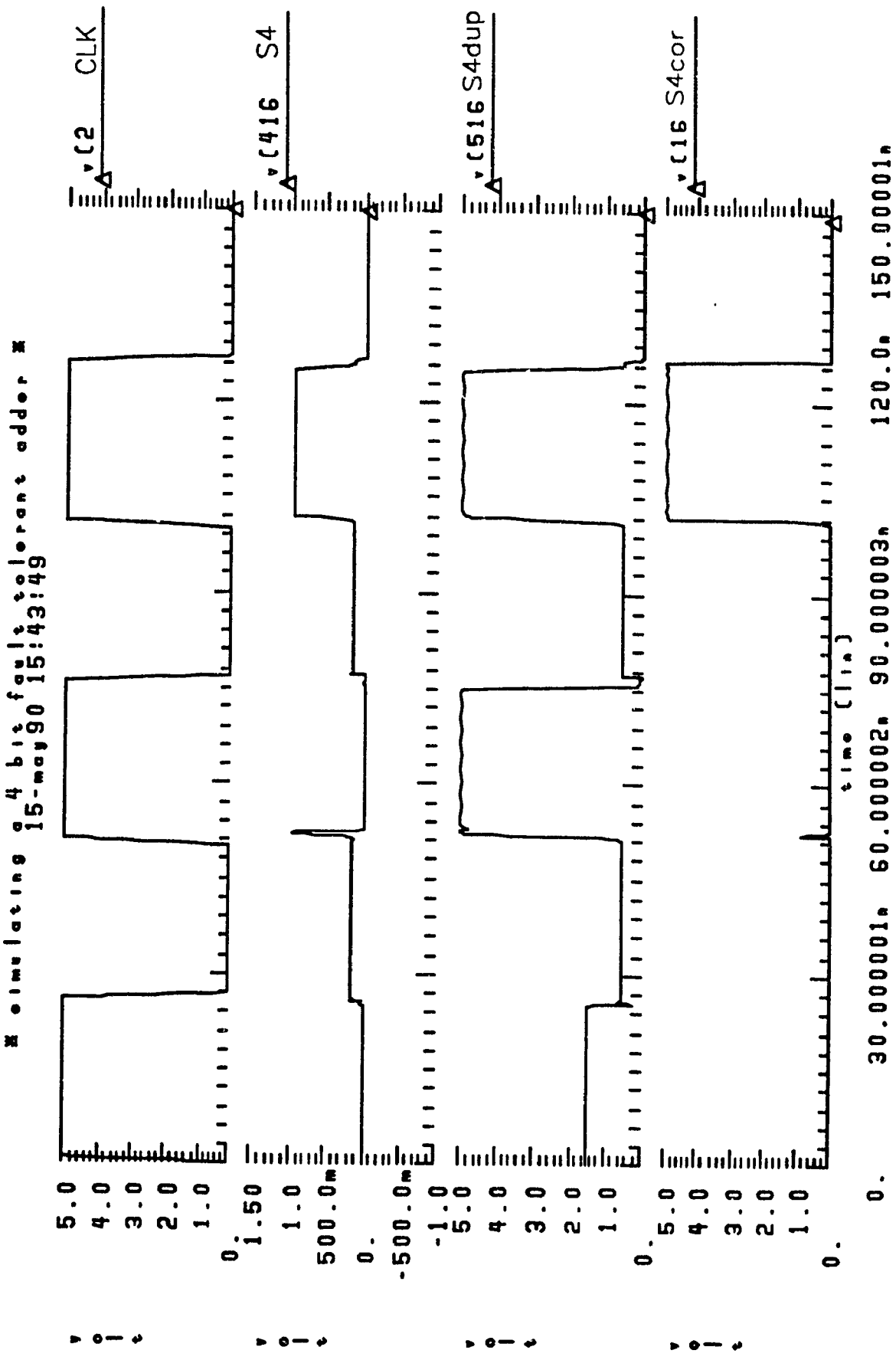


Fig.5.15.a Correcting Sum4

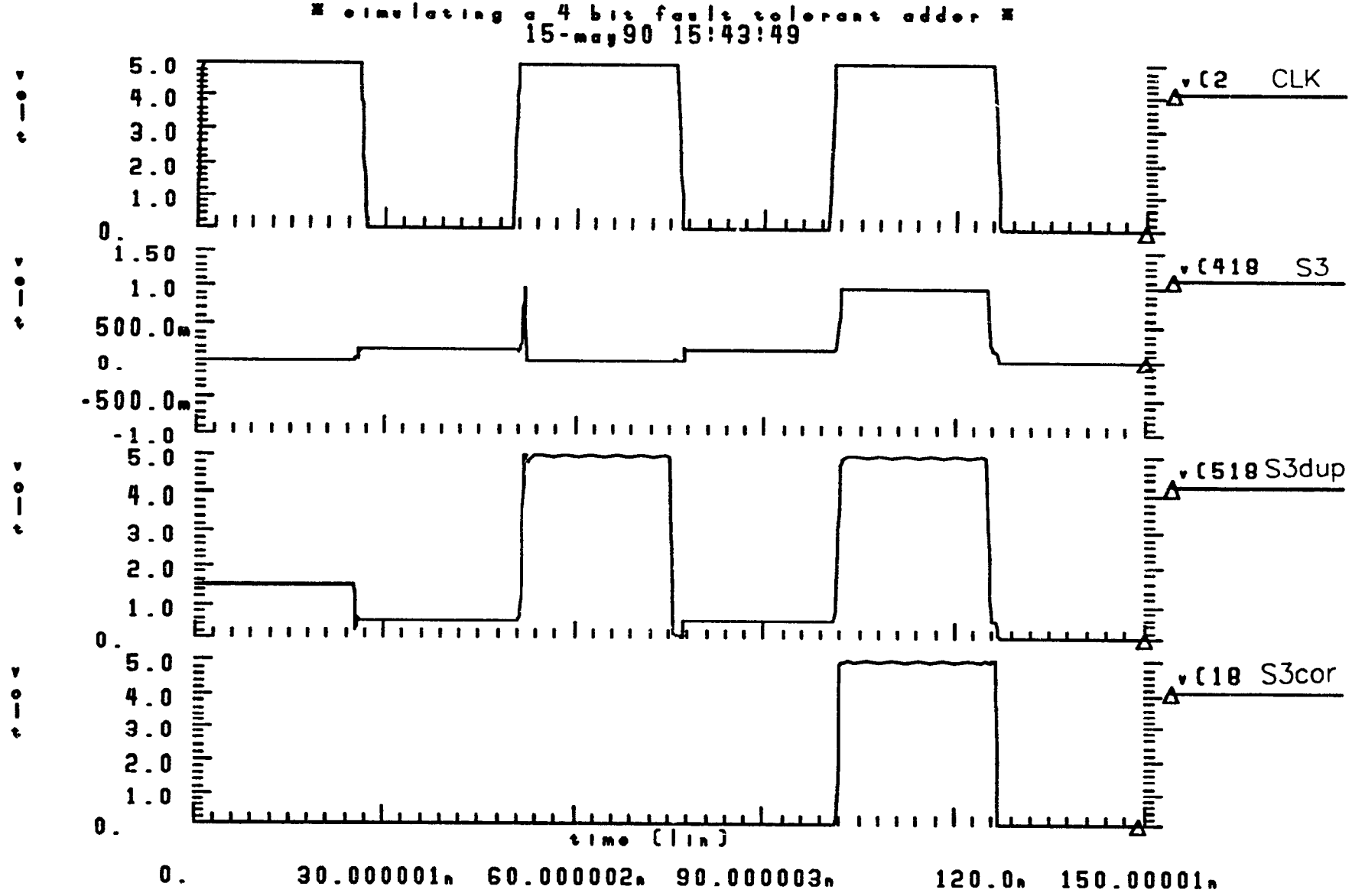


Fig.5.15.b Correcting Sum3

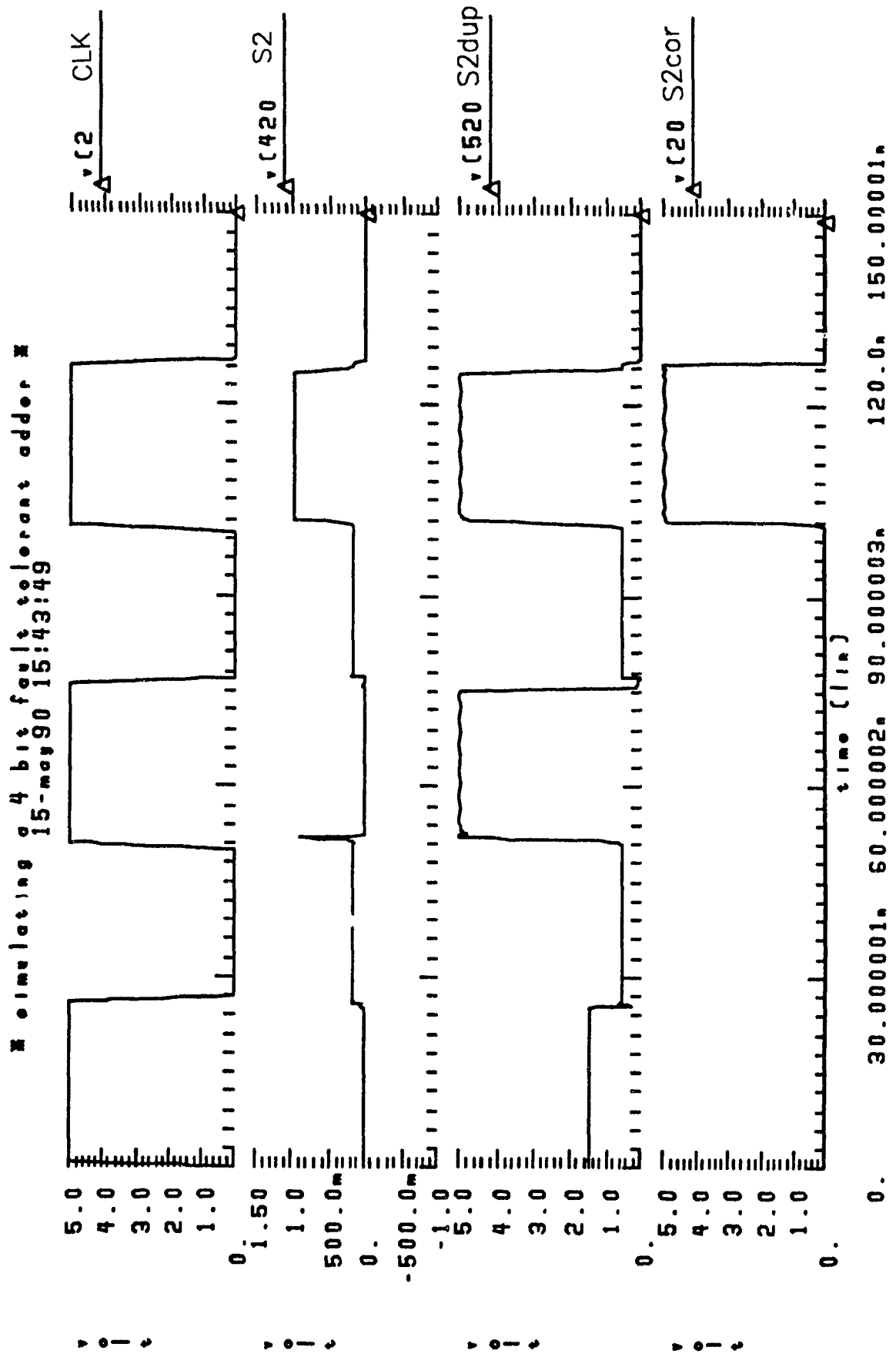


Fig.5.15.c Correcting Sum2

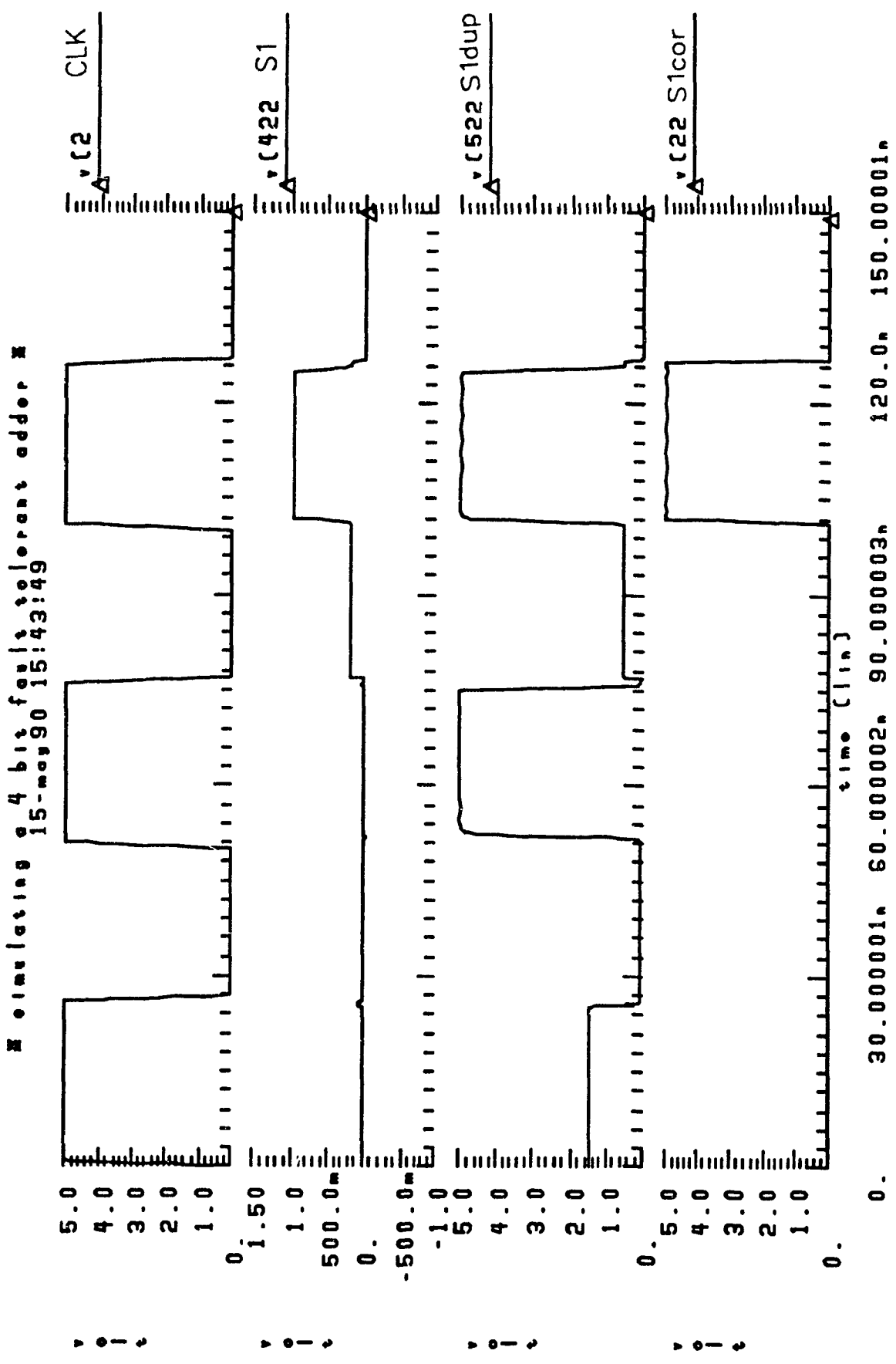


Fig.5.15.d Correcting Sum1

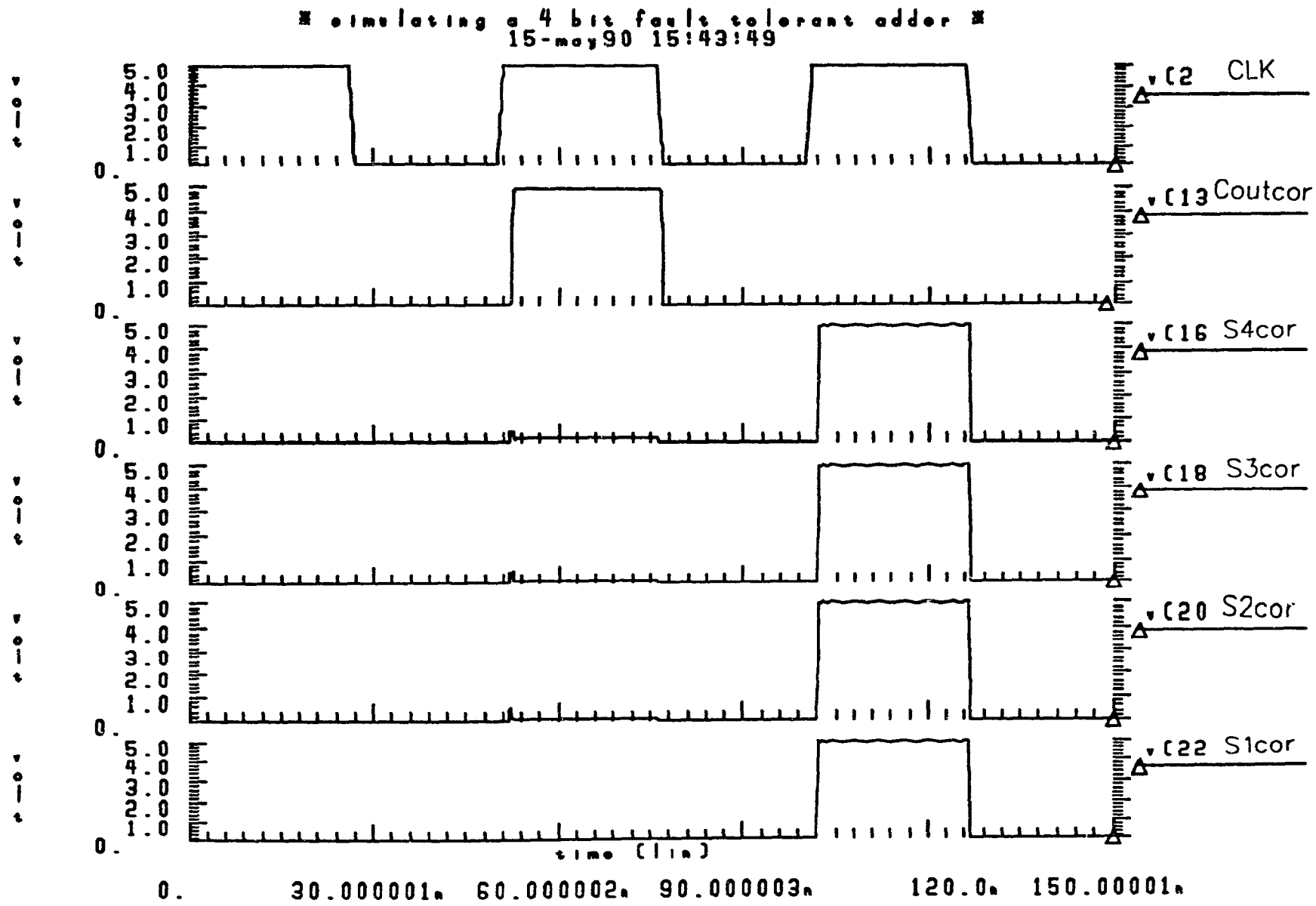


Fig.5.16 Overall Results from FT4BADDER after faults correction

combined. In terms of reliability, if we assume the reliability of each unit in the adder is R_m, then the reliability of the adder (R_{sys}) before introducing redundancy is:

$$R_{sys} = \prod_{1}^{3} R_m$$

If we assume the reliability of each of the correction circuit and the voter (for TMR) is R_c, then for TMR (implemented on the adder as one unit) the reliability of the resulting system would be[21]:

$$R_{TMR} = R_c \, (3R_{sys}^{\,2} - 2R_{sys}^{\,3})$$

While the reliability of our fault tolerant adder (R_{dup}) is[21]:

$$R_{dup} = \prod_{1}^{3} R_c \, (2R_m - R_m^{\,2})$$

It could be seen that the reliability of our fault tolerant adder is better than that of TMR, and also of the original adder under the assumption that the correcting circuits and the voter have better reliabilities than the adder's modules. However, if we assume that the correcting circuits and the voter have the same reliability as the adder's modules, we still have better reliability with our circuit in comparison to TMR or the original adder.

5.4. DESIGN OF A FAULT TOLERANT FULL 4-BIT ADDER

In the following sections, the design of a fault tolerant full 4-bit adder will be presented based on the given design of the full 4-bit adder, and the fault tolerance approach used in the previous fault tolerant 4-bit partial adder. The design of the various added circuitry (again based on the fault models of the various units of the adder introduced in the previous chapter) will be given, together with the complete design of the adder. And finally, the simulation results of the adder

with various faults simulated will be presented, followed by a comparison with Triple Modular Redundancy.

5.4.1 FAULT TOLERANCE APPROACH

The fault tolerance approach that we are going to use is the same approach used in the fault tolerant 4-bit partial adder. In the design, the first approach (to choose between two outputs based on the most likely fault occurrence and effect) was used to achieve fault tolerance in the Gen:Car unit, while the second approach (to choose between two outputs knowing the correct value) will be used for both the Gen:P&G units, and the XOR's set.

5.4.2. FAULT CORRECTING CIRCUITS

In this section, the design for each fault correction circuit will be given together with the various faults that it will cover.

5.4.2.a. GEN:P&G CORRECTION CIRCUIT

The circuit shown in Fig.5.17 is used to correct the generate and propagate signals (G&P). It is designed based on the fact that knowing the inputs to the Gen:P&G block, and thus the correct output, the circuit should be able to pick the correct value from the available outputs. The circuit implements the following functions:

$$Pcorrect = (A_i \oplus B_i)(P1 + P2)$$

Where P1 and P2 are the two P outputs from duplicate P&G generation blocks, Ai and Bi are the inputs to the two duplicate blocks, and Pcorrect is the correct P value. The G value is corrected through the function:

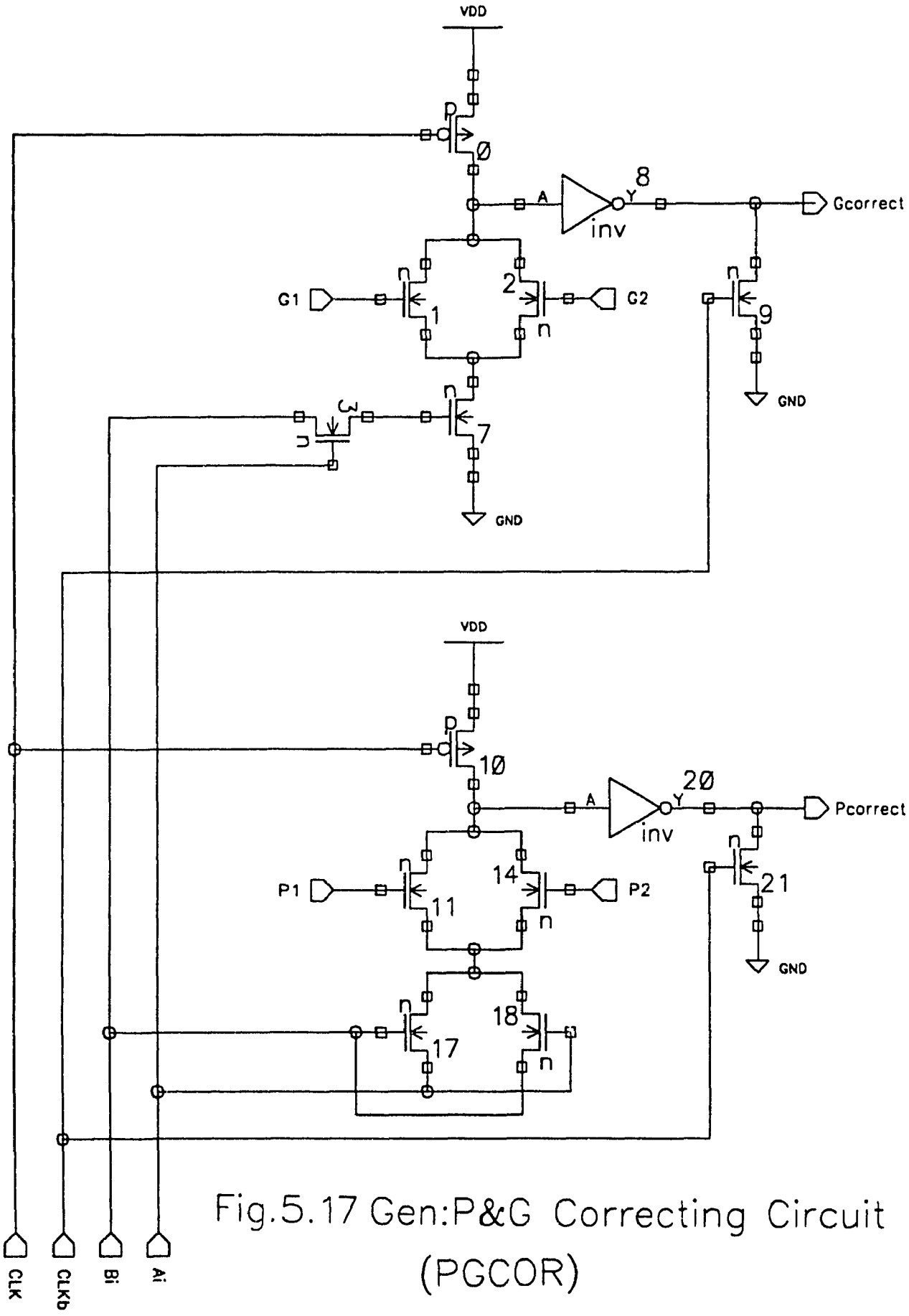


Fig.5.17 Gen:P&G Correcting Circuit (PGCOR)

$$Gcorrect = (A_i \bullet B_i)(G1 + G2)$$

Where again, G1 and G2 are the two G outputs from duplicate P&G generation block, and Gcorrect is the correct G value. The circuit would be able to cover all shorts, opens, and stuck at faults, so long as the duplicate blocks do not fail simultaneously with the same failure.

Fig.5.18 show the simulation results for the correction circuit.

5.4.2.b. GEN:CAR CORRECTION CIRCUIT

The circuit shown in Fig.5.19, shows a simple OR circuit that would choose a 1 whenever one or both outputs from the two carry generation circuit is 1 or Gcorrect is 1, and 0 with all outputs equal to 0.

The reason for adding G is that we know that we have to generate a carry whenever G is 1. Therefore, no matter what values the two duplicate carrys have, as long as G is 1 the correct carry will be produced. On the other hand, with P' equal 1, we have to propagate whatever value the previous carry had. Thus when ever G is equal 0, we will choose the value of the carry that is equal to 1.

The faults that will be covered by this circuit include any fault that might cause the carry to be erroneously stuck-at-0, and faults that result in an intermediate value where the correct value is 1. plus faults that might occur in any nmos that is controlled by G input.

Fig.5.20 shows the result of the circuit simulation.

5.4.2.c XOR CORRECTION CIRCUIT

The same modified XOR and XOR correction circuit will be used for the correction of the sum product.

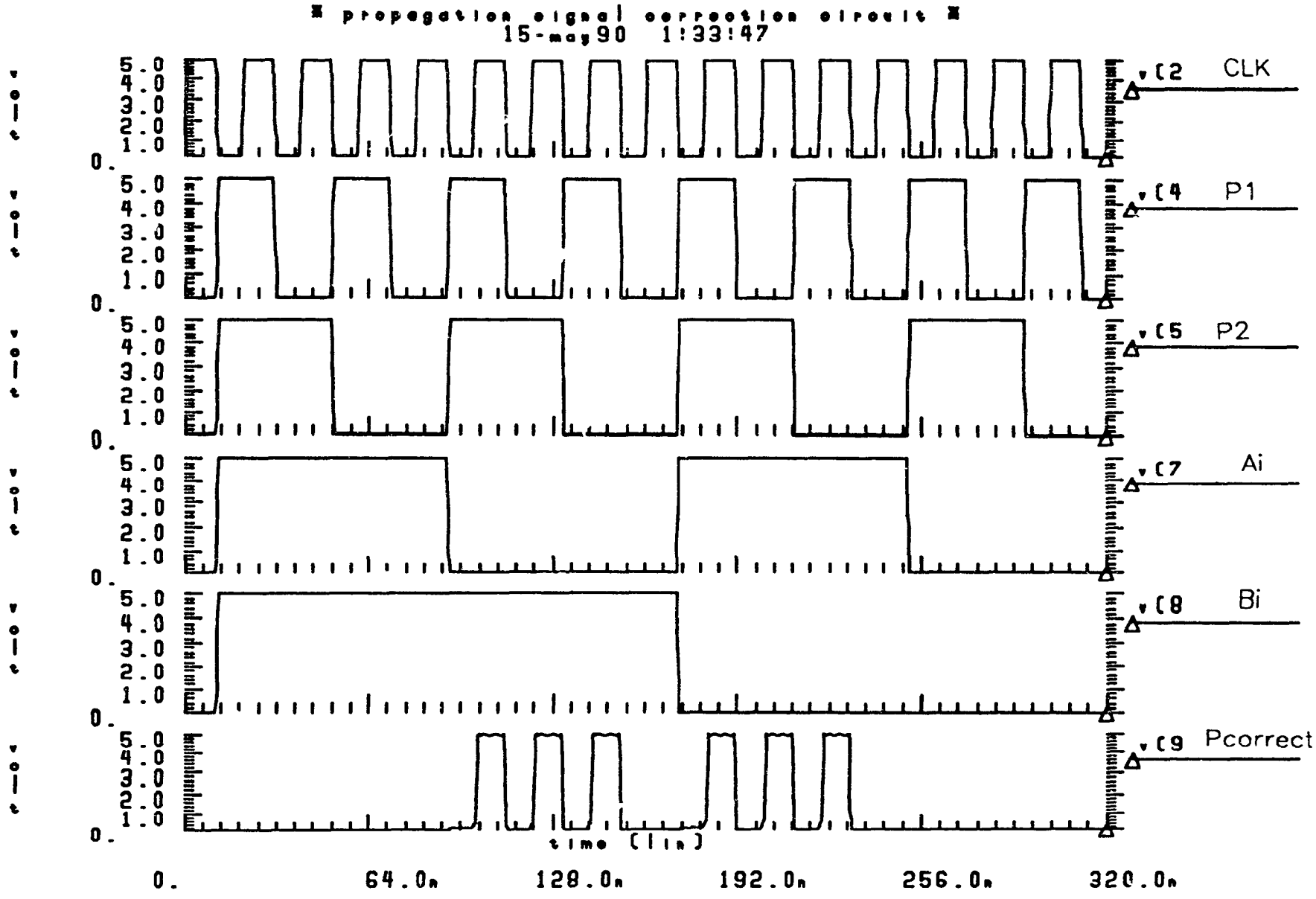


Fig.5.18.a Circuit Simulation for Fig.5.17 (P signal)

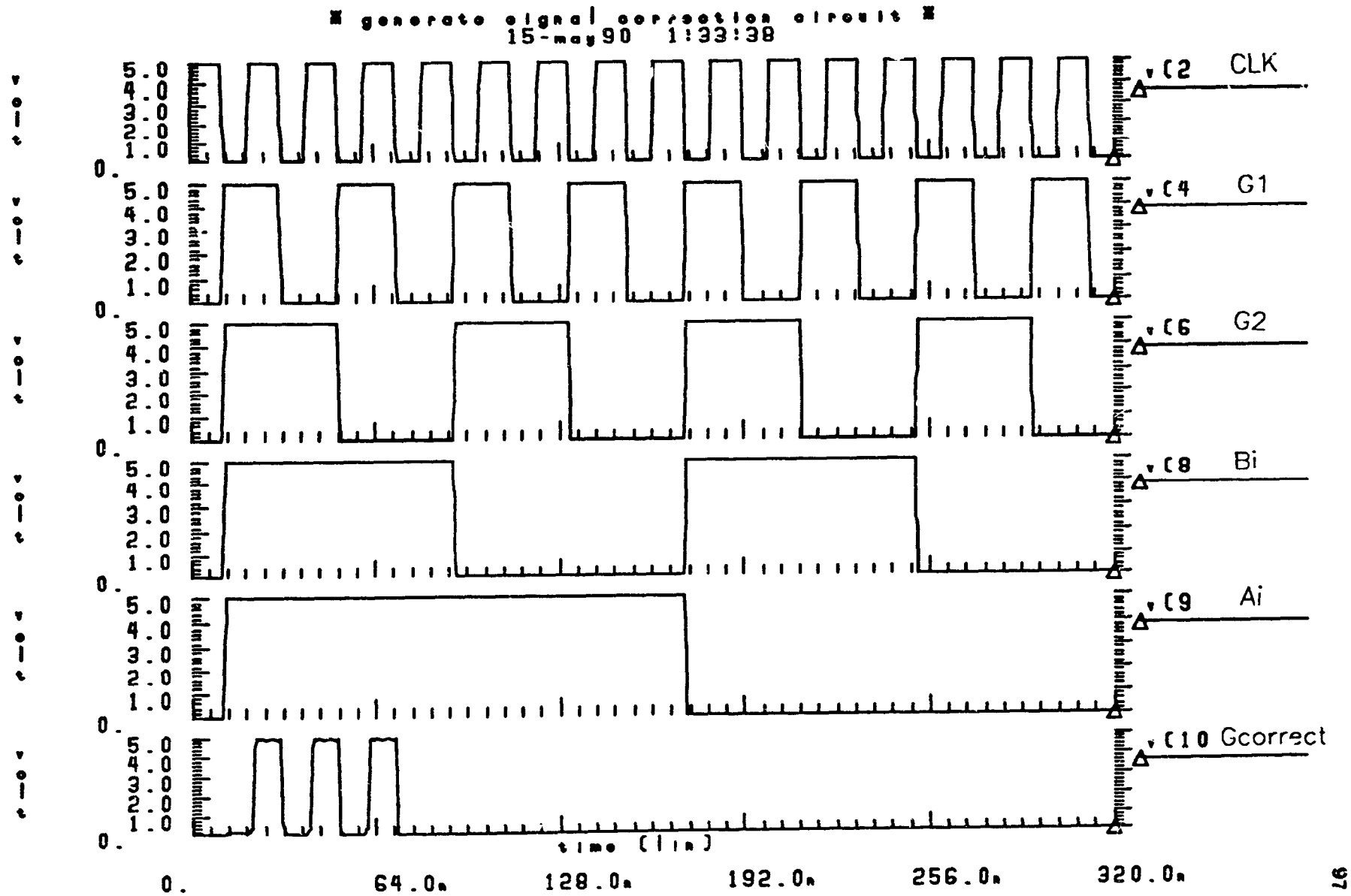


Fig.5.18.b Circuit Simulation for Fig.5.17 (G signal)

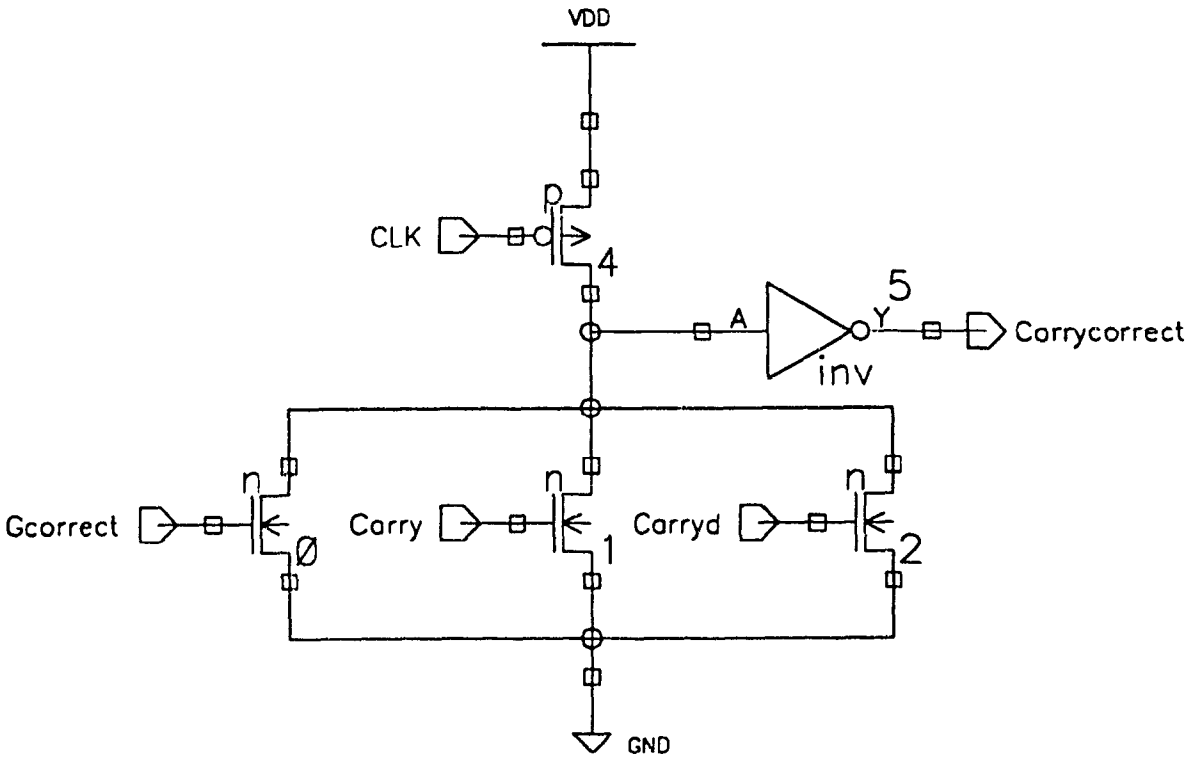


Fig.5.19 Gen:CAR Correcting Circuit (CARCOR)

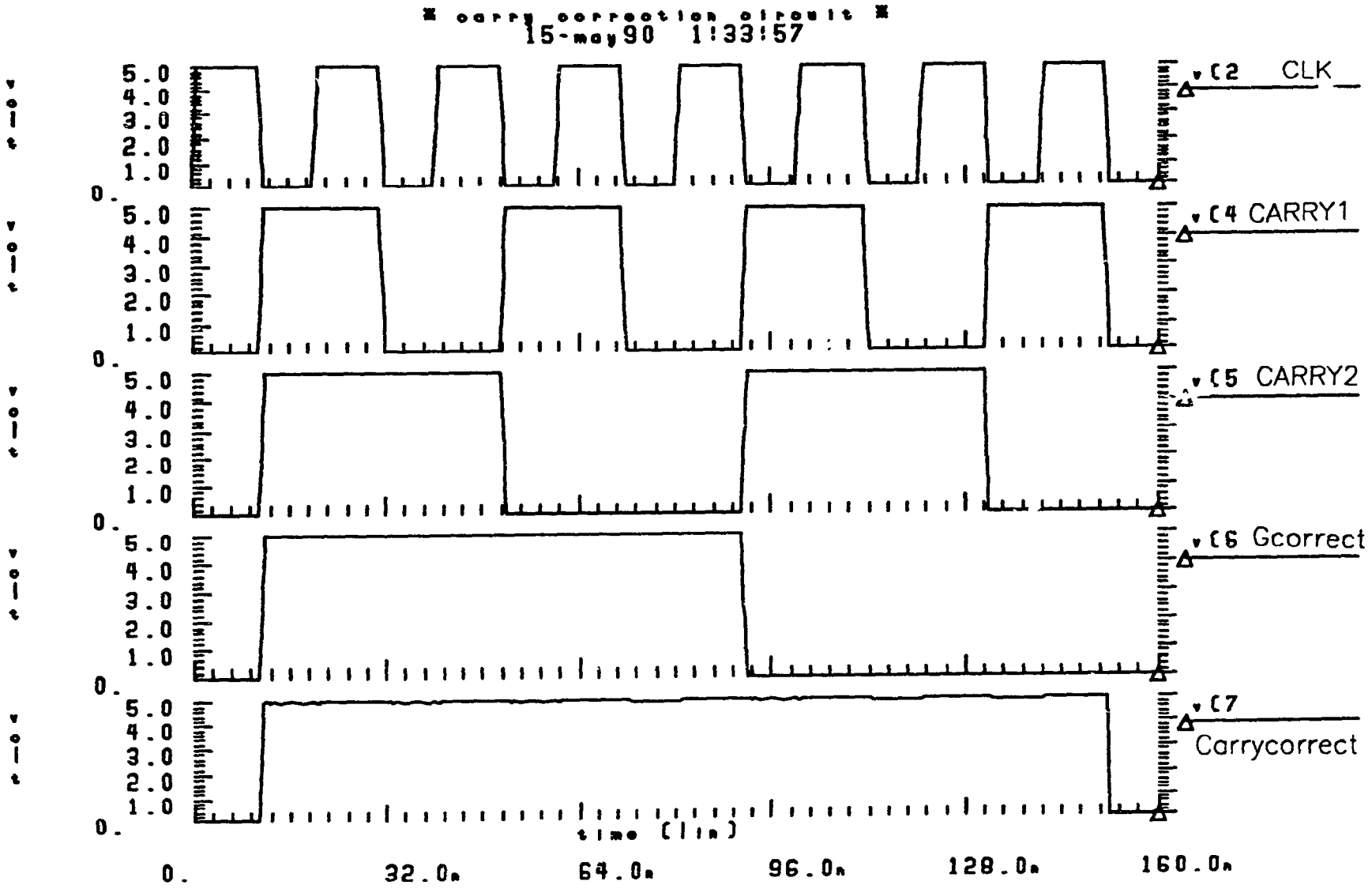


Fig.5.20 Circuit Simulation for Fig.5.19

5.4.2.d. FAULT TOLERANT FULL 4-BIT ADDER

Fig.5.21 shows the complete adder, while Fig.5.22.a shows a set of test vectors (the same used with the full 4-bit adder). Results of simulation are shown in Fig.5.22.b. From the simulation results, the complete adder gives a worst case propagation delay of 2.6ns. Worst case delay is considered to be for the sum signal to be produced correctly (that is from CLK to Sum4). This comes from 1ns delay to get the generate signals correctly, plus 0.8ns delay to get the correct carry signals, plus 0.8ns to get the correct sum signal.

5.4.3. FAULT SIMULATION

Next, a set of faults were introduced to the circuit. These faults are:

1. A short between gate and source in nmos of input A1 in Gen:P&G (Fig.5.23).
2. A short between gate and drain in nmos and pmos of the inverter in Gen:P&G (Fig.5.23).
3. A short between drain and source in nmos1 of xor in Gen:P&G(Fig.5.23).
4. A short between drain and source in discharge nmos of Gen:P&G duplicate (Fig.5.23).
5. A short between drain and source in pmos1 of xor in Gen:P&G duplicate (Fig.5.23).
6. An open in GND connection of discharge nmos in Gen:Car (Fig.5.24).
7. A short between gate and source in nmos of input G3 in Gen:Car (Fig.5.24).
8. A short between drain and source in nmos1 of xor in sum calculation (Fig.5.25).
9. A short between drain and source in pmos1 of xor in sum calculation (Fig.5.25).

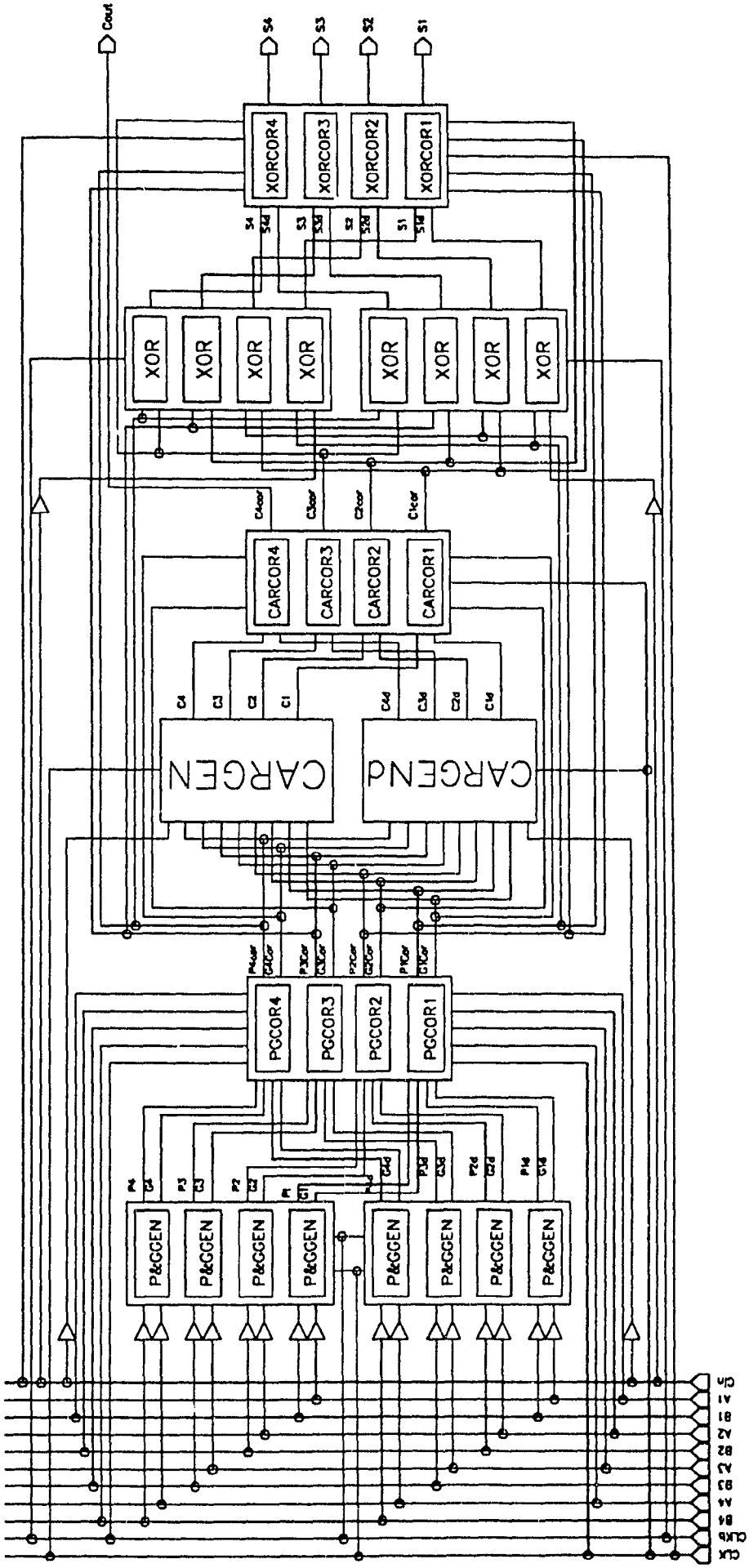


Fig.5.21 Fault Tolerant Full 4–Bit Adder (FT4FBADDER)

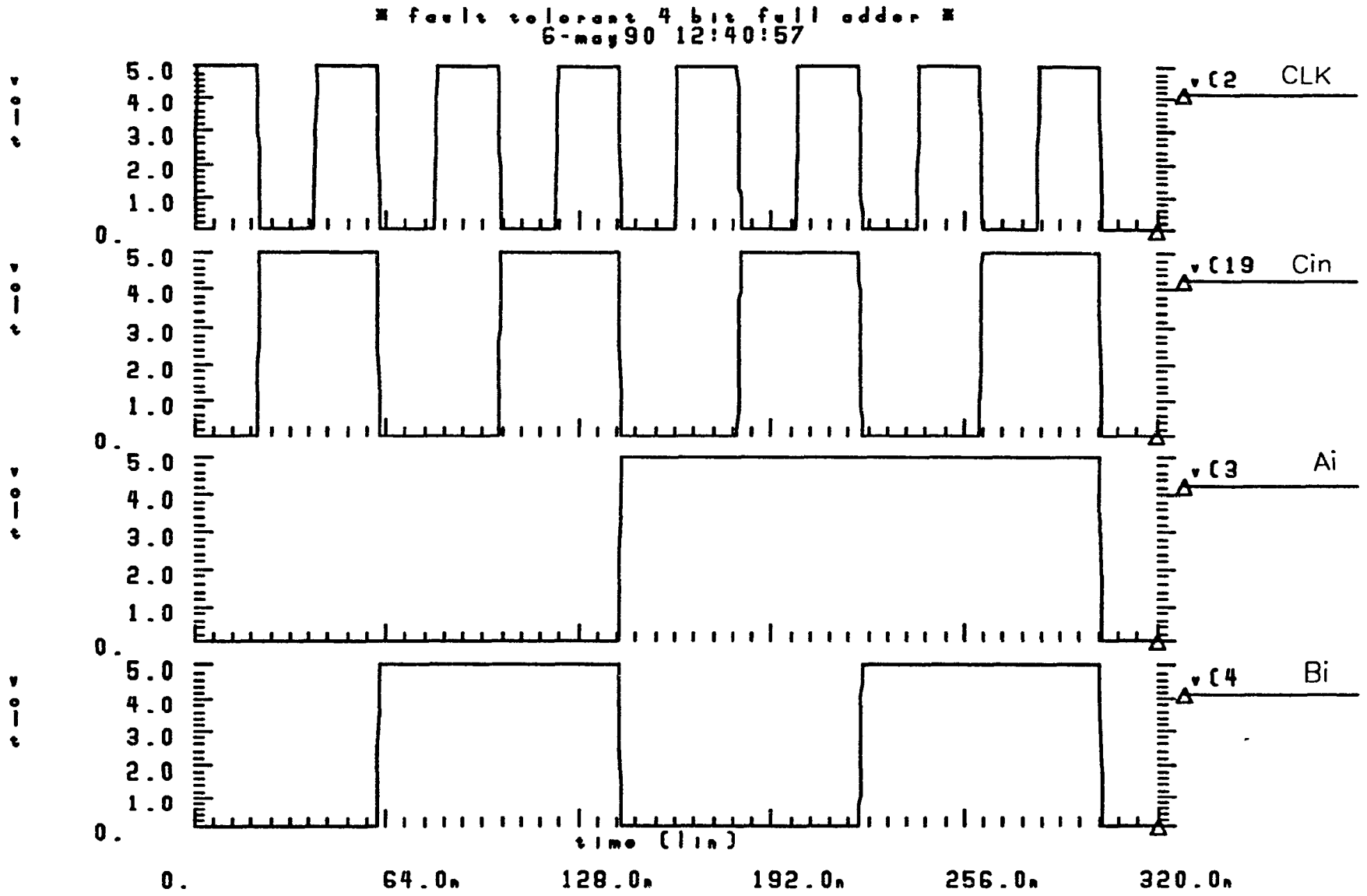


Fig.5.22.a Input Test Vectors for FT4FBADDER

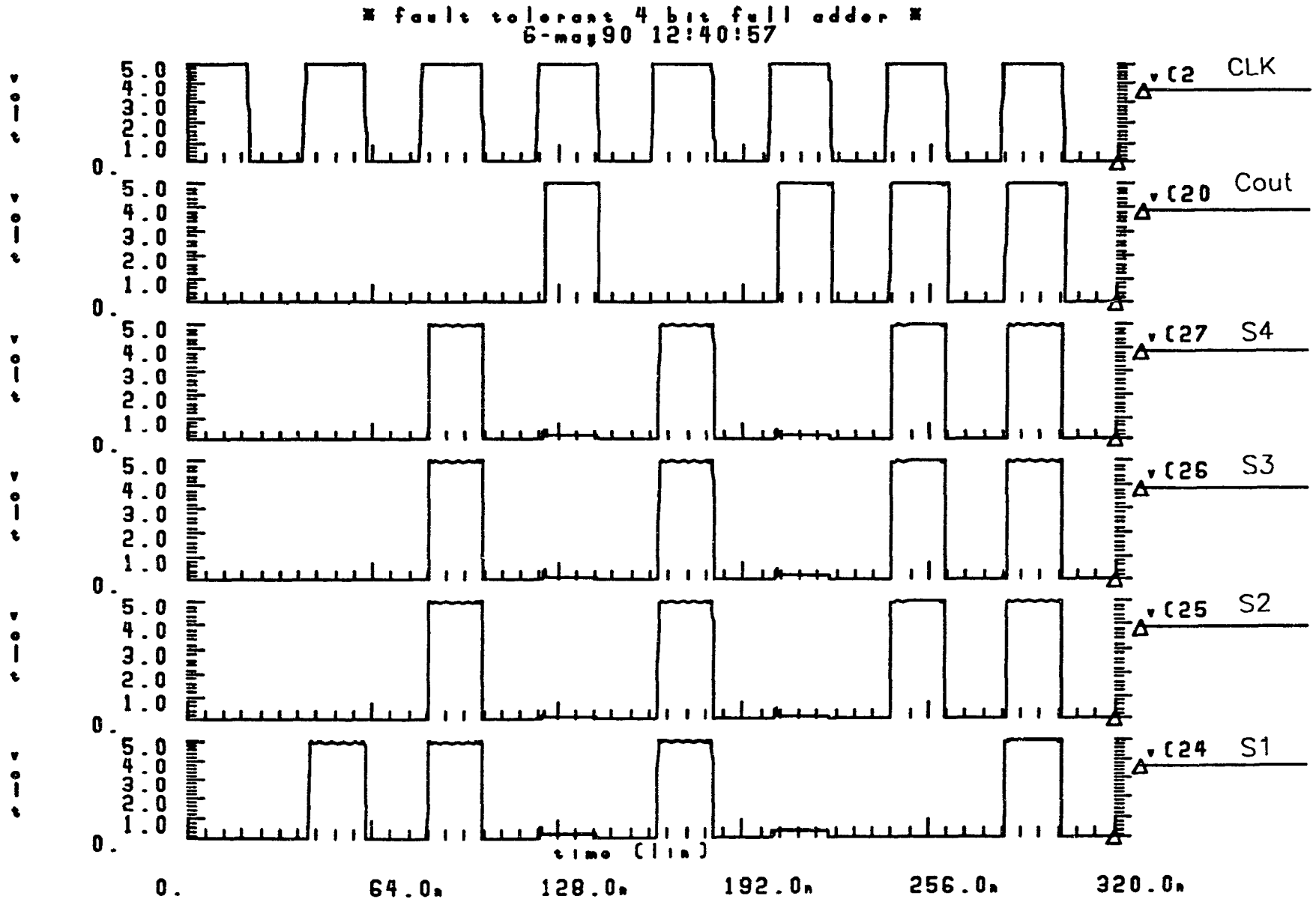
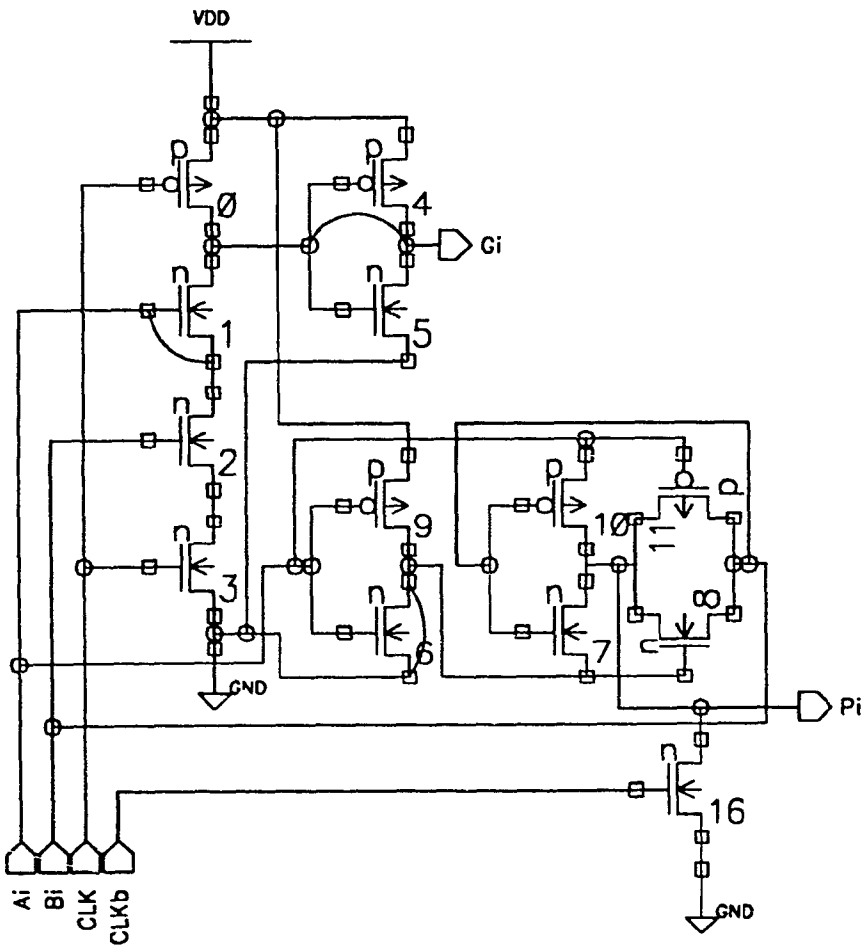


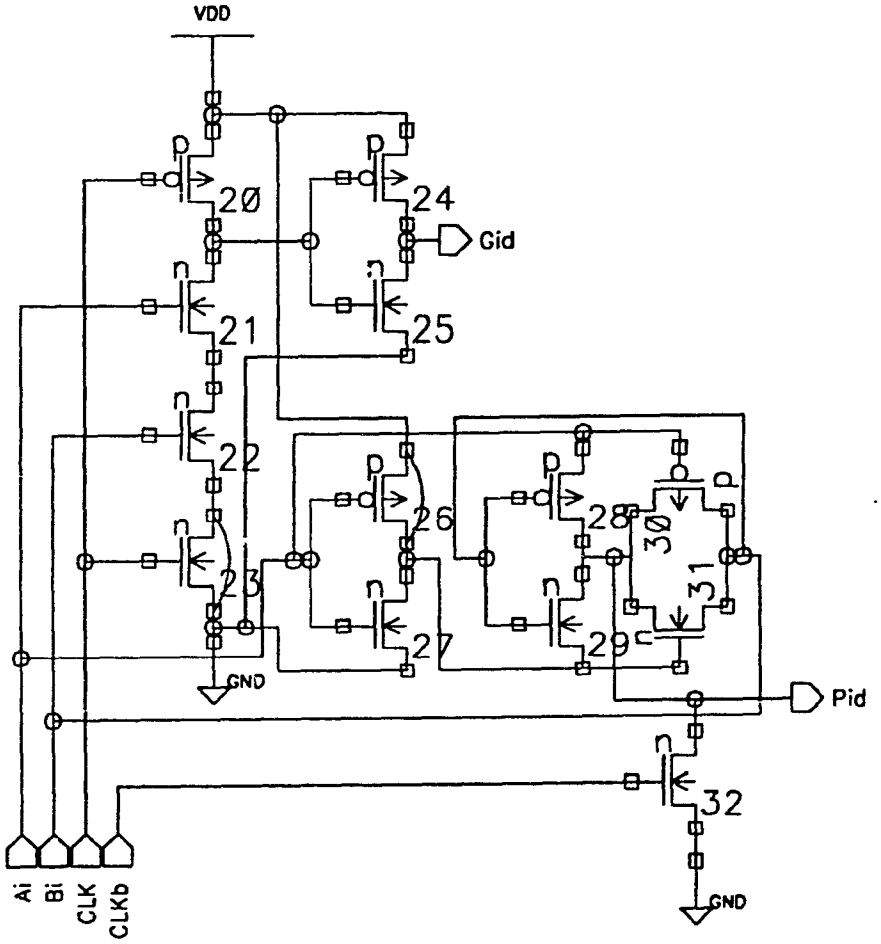
Fig.5.22.b Output Results of FTF4BADDER in Response to Fig.5.22.a

```
* PROPAGATION DELAY BETWEEN CLK AND COUT*
delay1          =  2.5570E-09  targ=  1.2306E-07  trig=  1.2050E-07
delay2          =  2.6001E-09  targ=  2.0310E-07  trig=  2.0050E-07
delay3          =  1.4643E-09  targ=  2.4196E-07  trig=  2.4050E-07
delay4          =  1.5002E-09  targ=  2.8200E-07  trig=  2.8050E-07

*PROPAGATION DELAY BETWEEN CLK AND SUM4*
delay5          =  2.5880E-09  targ=  8.3088E-08  trig=  8.0500E-08
delay6          =  2.6004E-09  targ=  1.6310E-07  trig=  1.6050E-07
delay7          =  2.4697E-09  targ=  2.4297E-07  trig=  2.4050E-07
delay8          =  2.5194E-09  targ=  2.8302E-07  trig=  2.8050E-07

*PROPAGATION DELAY BETWEEN CLK AND SUM3*
delay9          =  2.5880E-09  targ=  8.3088E-08  trig=  8.0500E-08
delay10         =  2.5704E-09  targ=  1.6307E-07  trig=  1.6050E-07
delay11         =  2.4697E-09  targ=  2.4297E-07  trig=  2.4050E-07
delay12         =  2.5194E-09  targ=  2.8302E-07  trig=  2.8050E-07

*PROPAGATION DELAY BETWEEN CLK AND SUM2*
delay13         =  2.5880E-09  targ=  8.3088E-08  trig=  8.0500E-08
delay14         =  2.5505E-09  targ=  1.6305E-07  trig=  1.6050E-07
delay15         =  2.4696E-09  targ=  2.4297E-07  trig=  2.4050E-07
delay16         =  2.5194E-09  targ=  2.8302E-07  trig=  2.8050E-07

*PROPAGATION DELAY BETWEEN CLK AND SUM1*
delay17         =  1.6352E-09  targ=  4.2135E-08  trig=  4.0500E-08
delay18         =  2.5354E-09  targ=  8.3035E-08  trig=  8.0500E-08
delay19         =  2.5398E-09  targ=  1.6304E-07  trig=  1.6050E-07
delay20         =  1.6646E-09  targ=  2.8216E-07  trig=  2.8050E-07
```

Fig.5.22.c Summary of Simulation Result

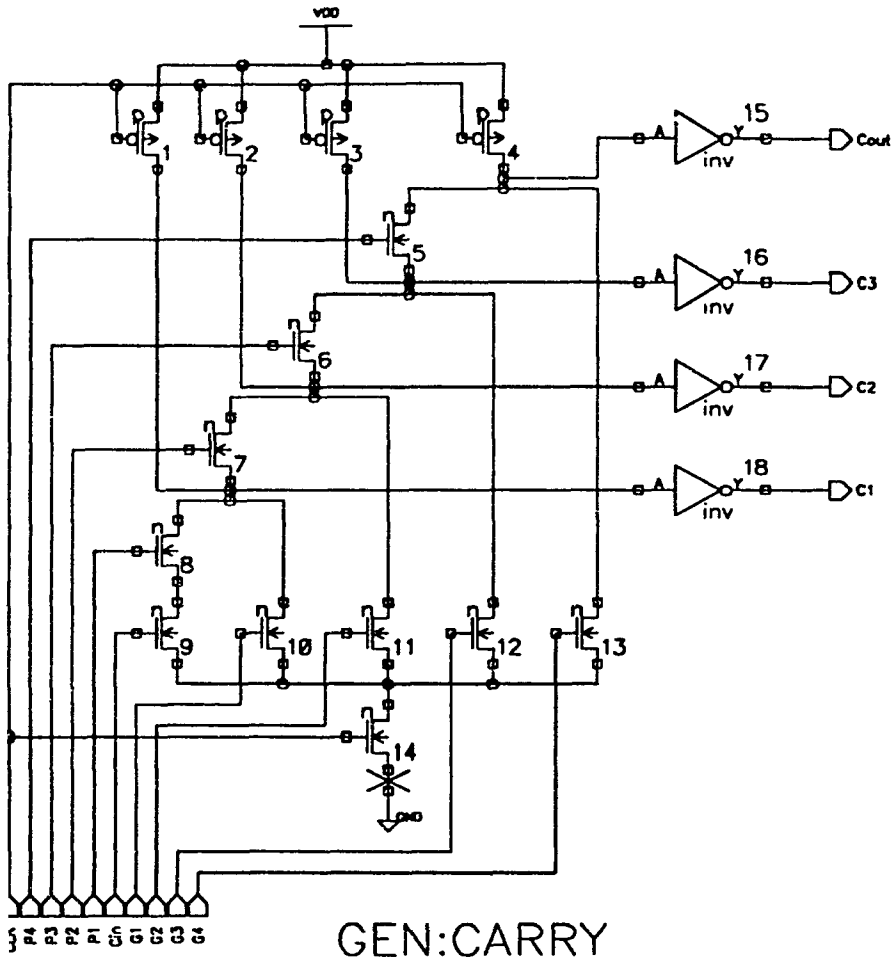


GEN:P&G

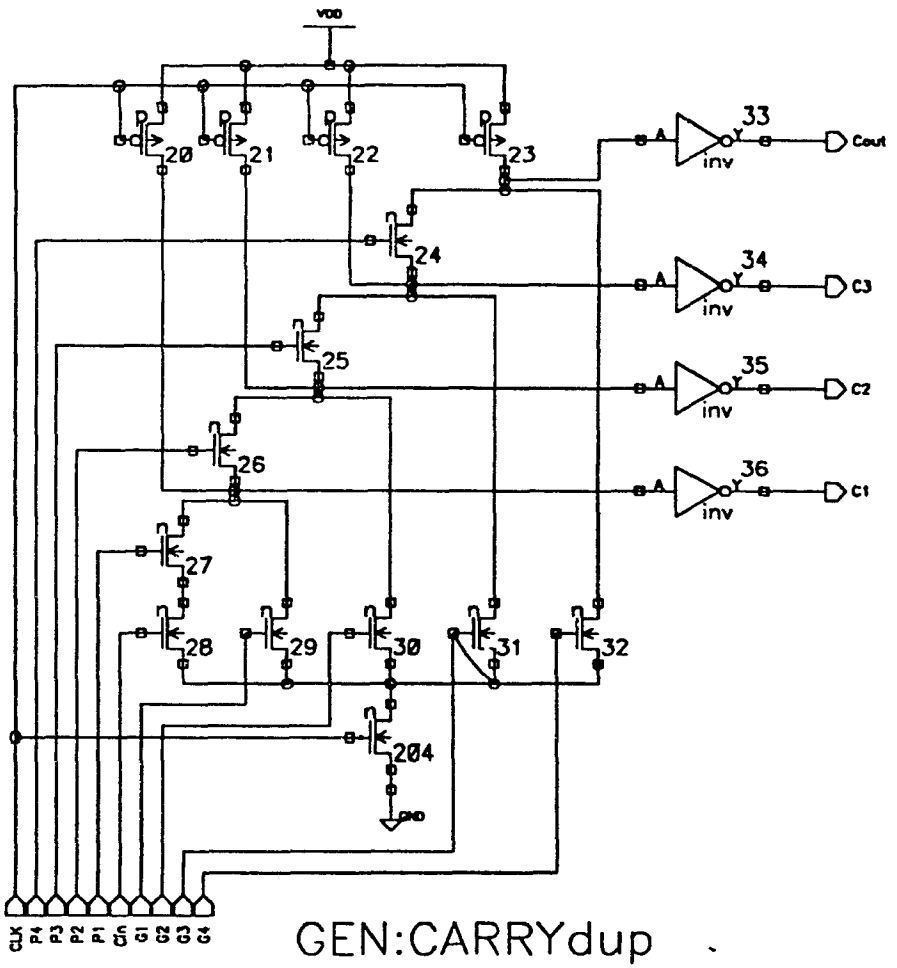


GEN:P&Gdup

Fig.5.23 Simulated Faults in Gen:P&G Block



GEN:CARRY



GEN:CARRYdup

Fig.5.24 Simulated Faults in Gen:CAR Block

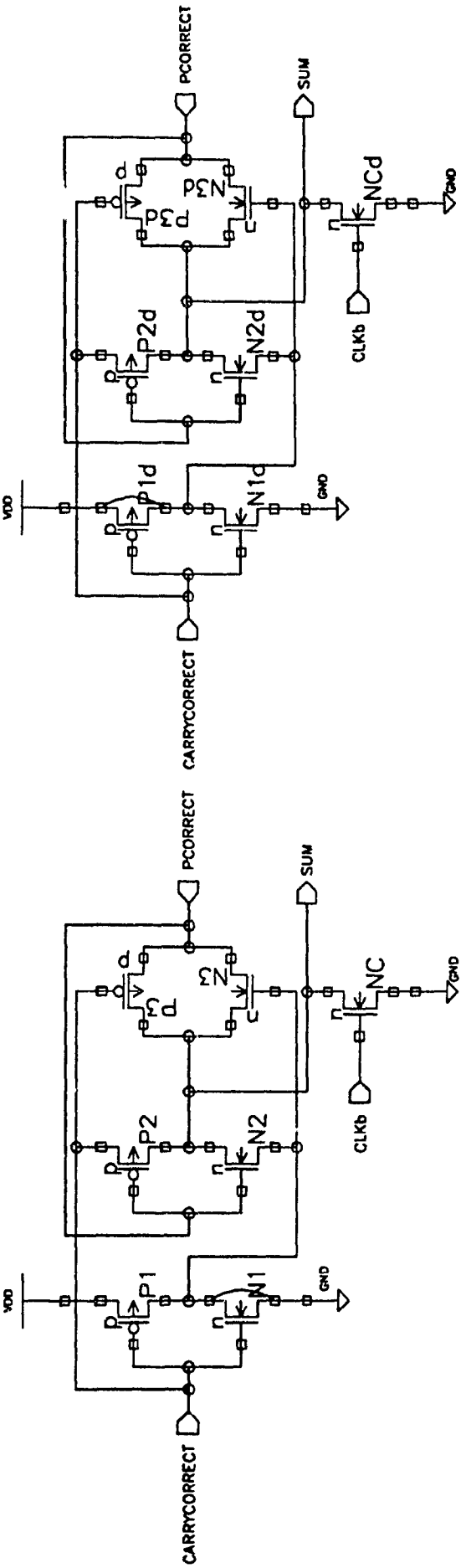


Fig.5.25 Simulated Faults in XOR Block

The effect of these faults could be seen as following:

A. For Gen:P&G block. Due to short between gate and source of nmos of Al input, the output is kept from discharging, however due to the added short between gate and drain in nmos and pmos of the inverter the output Gl fluctuates between 2.5v and 3.6v.

Also due to the short between drain and source of nmos1 in xor of Gen:P&G, nmos3 in the xor is always turned off, thus with Al equal 0 and Bl equal 1, nmos2 is turned on, thereby, discharging the 1 that is passed through pmos3.

The short between drain and source in discharge nmos in duplicate Gen:P&G causes Gb (G bar) to charge to an intermediate value during low clock, specially whenever both Al and Bl are 1's.

Finally a short between drain and source of pmos1 in xor of duplicate Gen:P&G causes nmos2 of xor to be turned off while nmos3 will be always turned on, thus with both A and B equal 1, we get a faulty 1, while with Al equal 1 and Bl equal 0 we get a faulty soft 0 (1v).

B. For Gen:Car block. With an open source connection to ground in discharge nmos, all nodes will be unable to discharge no matter what input values of P's and G's we have, thus resulting in a fixed value of 0 for all carry's.

Also with a short between gate and source of nmos of G3 input, will cause the carry to have a 0 faulty value whenever G3 is 1.

C. For XOR (sum generation block), a short between drain and source of nmos1 in xor block, and a short between drain and source of pmos1 in xor duplicate block was simulated. The effect of these faults were described in the xor behaviour of the Gen:P&G block.

From the simulation results it could be seen that the P&G correction circuit

was able to correct the faults that occurred in the Gen:P&G blocks and the duplicate ones, thus passing a fault free set of P's & G's to both the Gen:Car, and Gen:Car duplicate blocks.

Fig.5.26.a, 5.26.b, 5.26.c, and 5.26.d show the simulation results.

In the carry generation stage, even though at one point both Gen:Car blocks were giving a faulty result (with one of the blocks giving a constant 0 for the carry's and the other giving a 0 for Carry3 (C3) when G3 equal 1), however, the carry correction circuit was able to correct these faults, and passed a set of correct carry's to the exclusive or stage. Fig.5.27.a, 5.27.b, 5.27.c, and 5.27.d show the simulation results.

In the third and final stage of the adder, the xor correction circuit corrected the faults occurring in both xor and duplicate xor, depending on both inputs to the xor that are fault free and comparing the two available outputs to determine the correct sum values.

Fig.5.28.a, 5.28.b, 5.28.c, and 5.28.d show the simulation results.

Fig.5.29 shows the final results of the simulation of the complete circuit, and it could be seen that all outputs are correct and match the fault free circuit results, in term of values and propagation delays.

5.4.4. COMPARISON

As was seen in the fault tolerant 4-bit partial adder, by dividing the original adder into three parts, the design of fault correction circuits became less complicated, and the fault coverage inadvertently increased.

In the circuit 320 transistors were used, which would compare, in terms of area, to 426 transistors that would have been used had we implemented TMR for each part of the adder as we did in our fault tolerance approach, and which

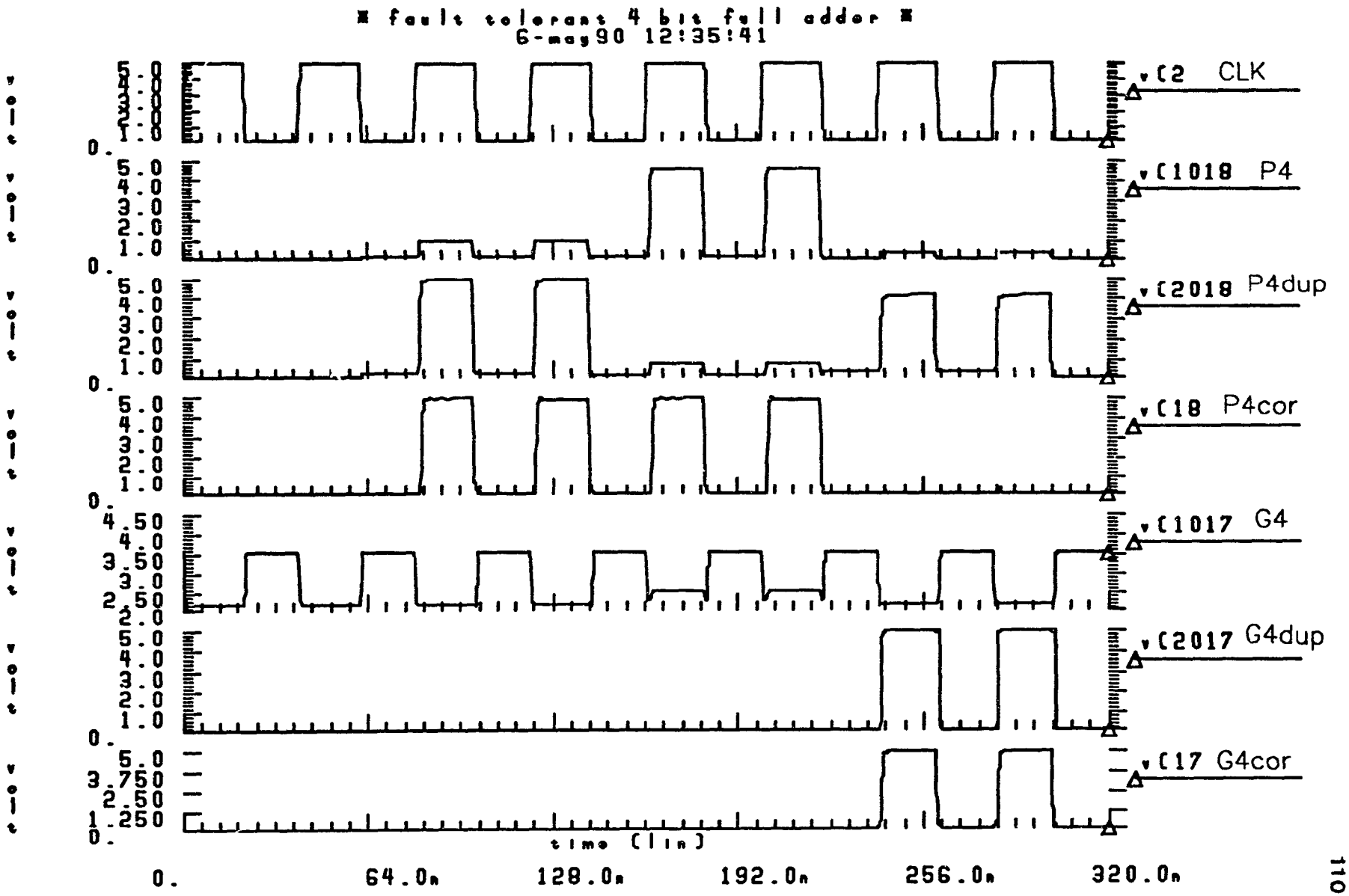


Fig.5.26.a Correcting P4 & G4

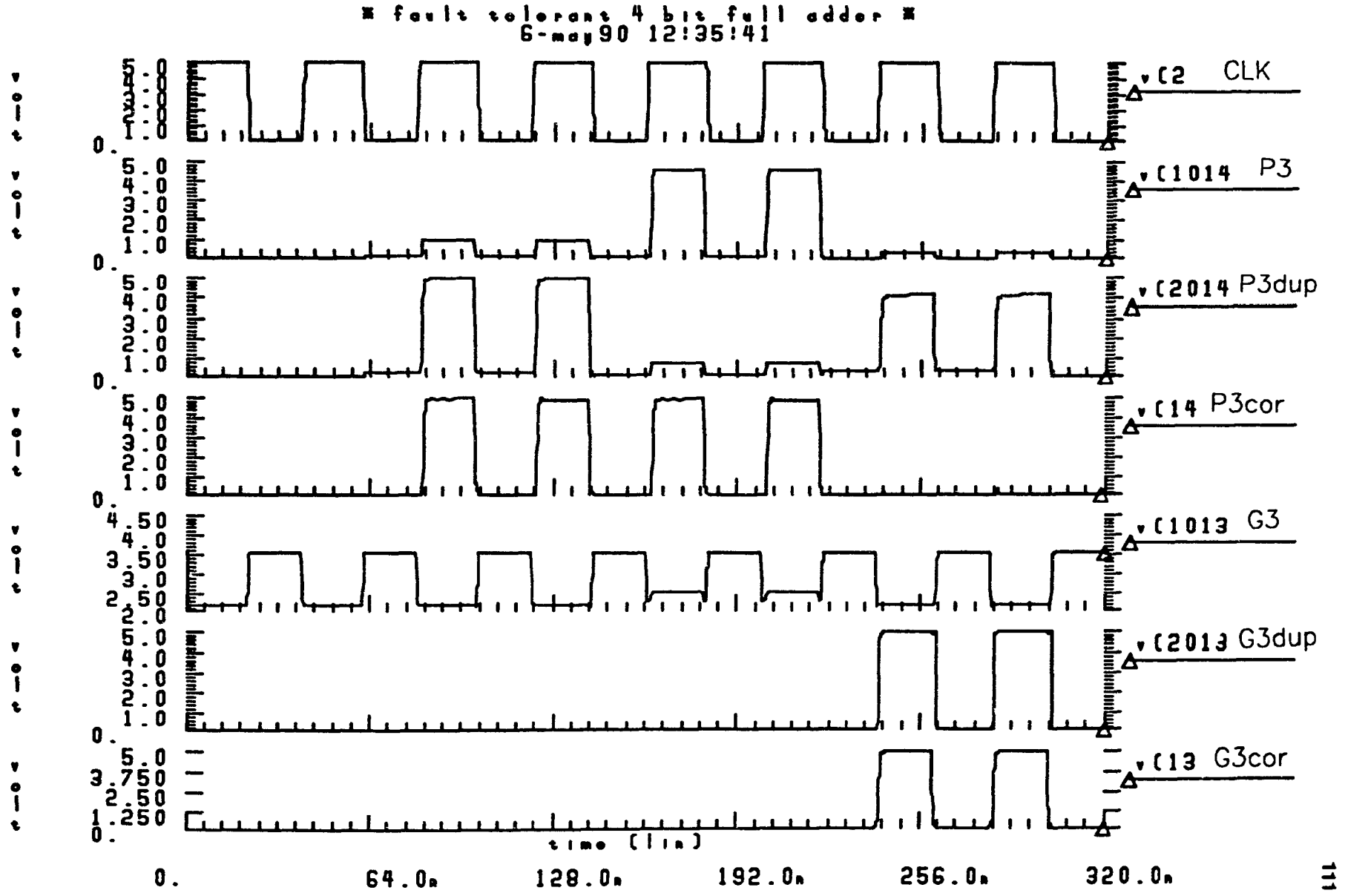


Fig.5.26.b Correcting P3 & G3

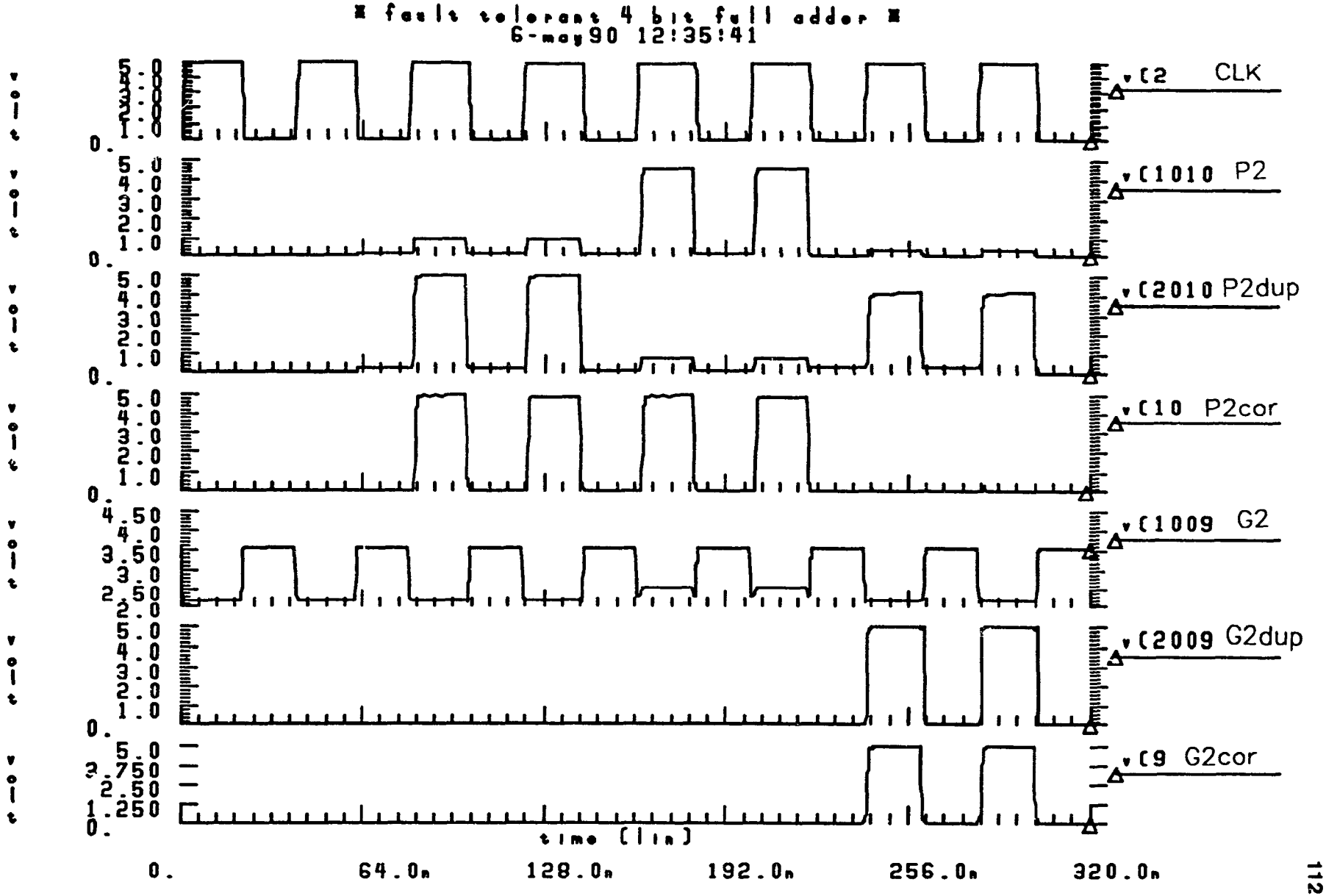


Fig.5.26.c Correcting P2 & G2

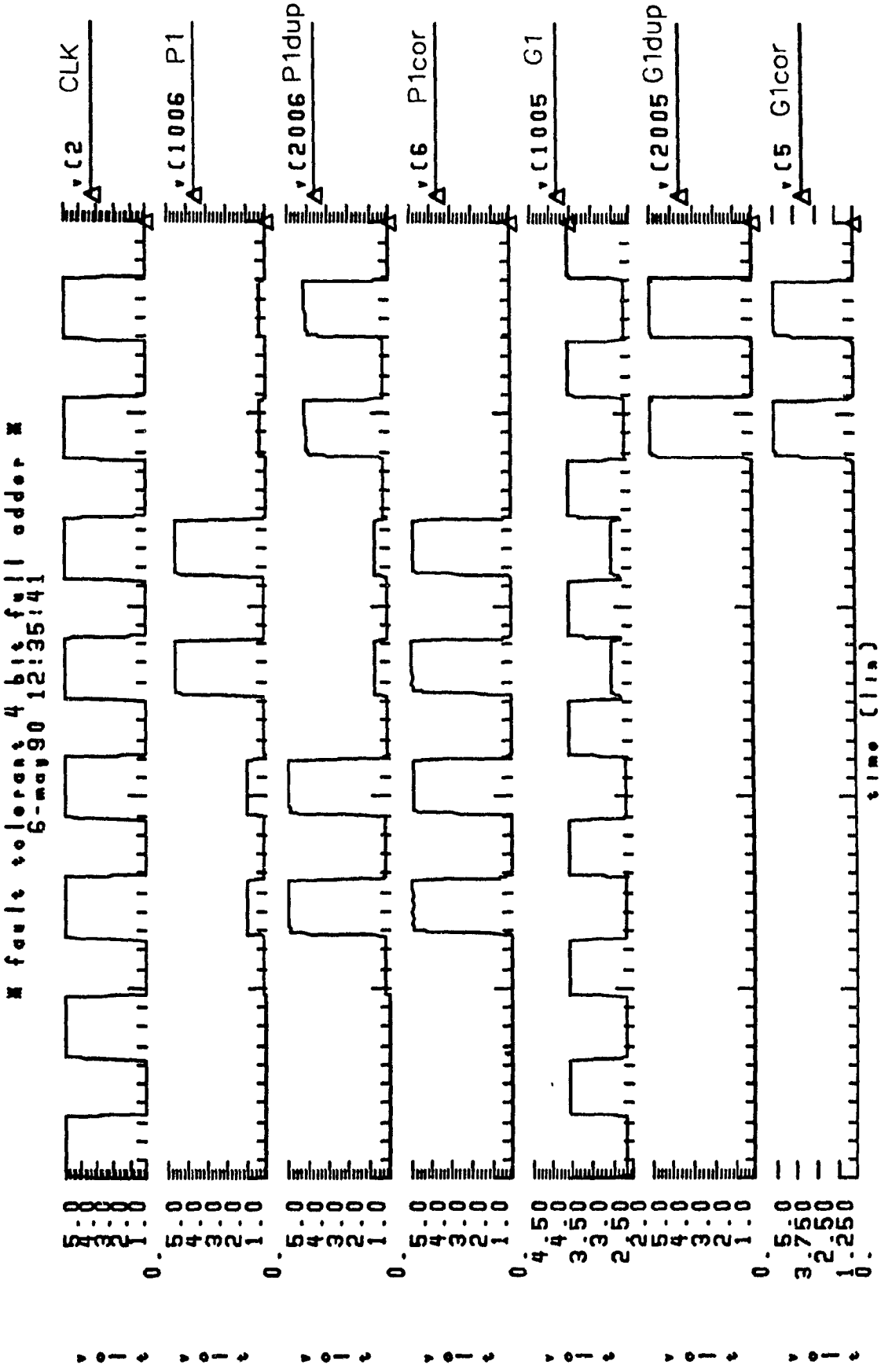


Fig.5.26.d Correcting P1 & G1

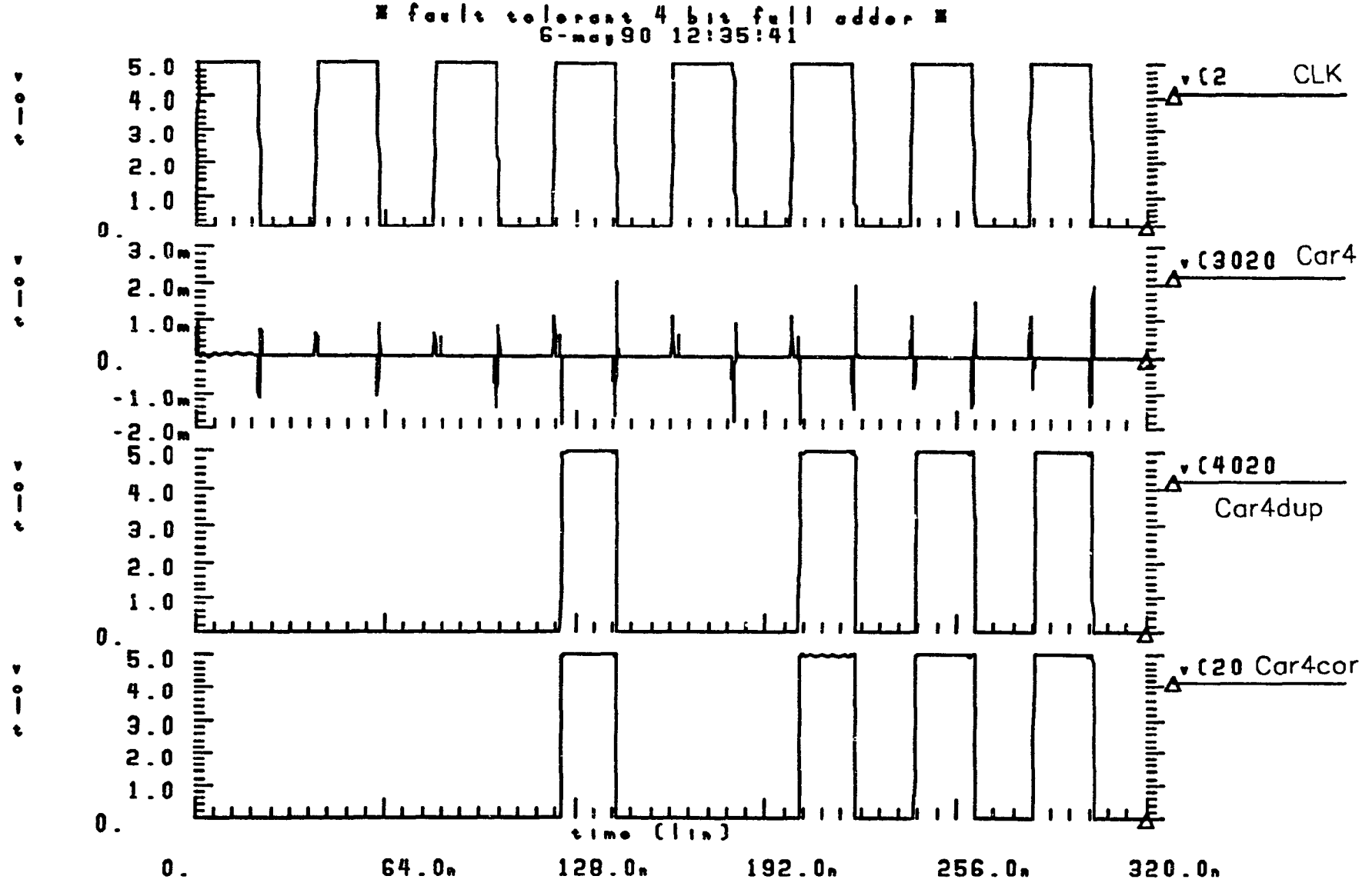


Fig.5.27.a Correcting Carry4

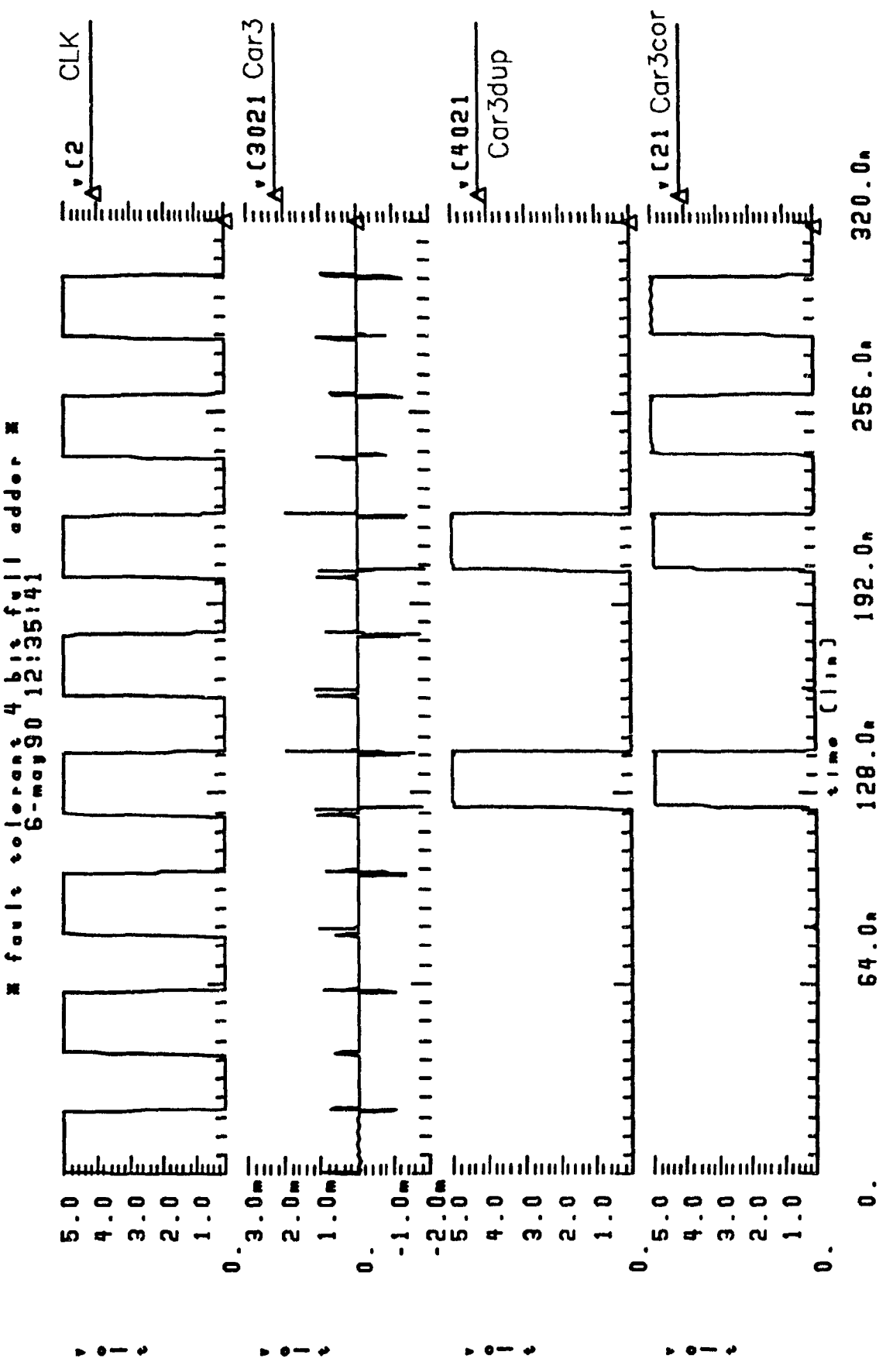


Fig.5.27.b Correcting Carry3

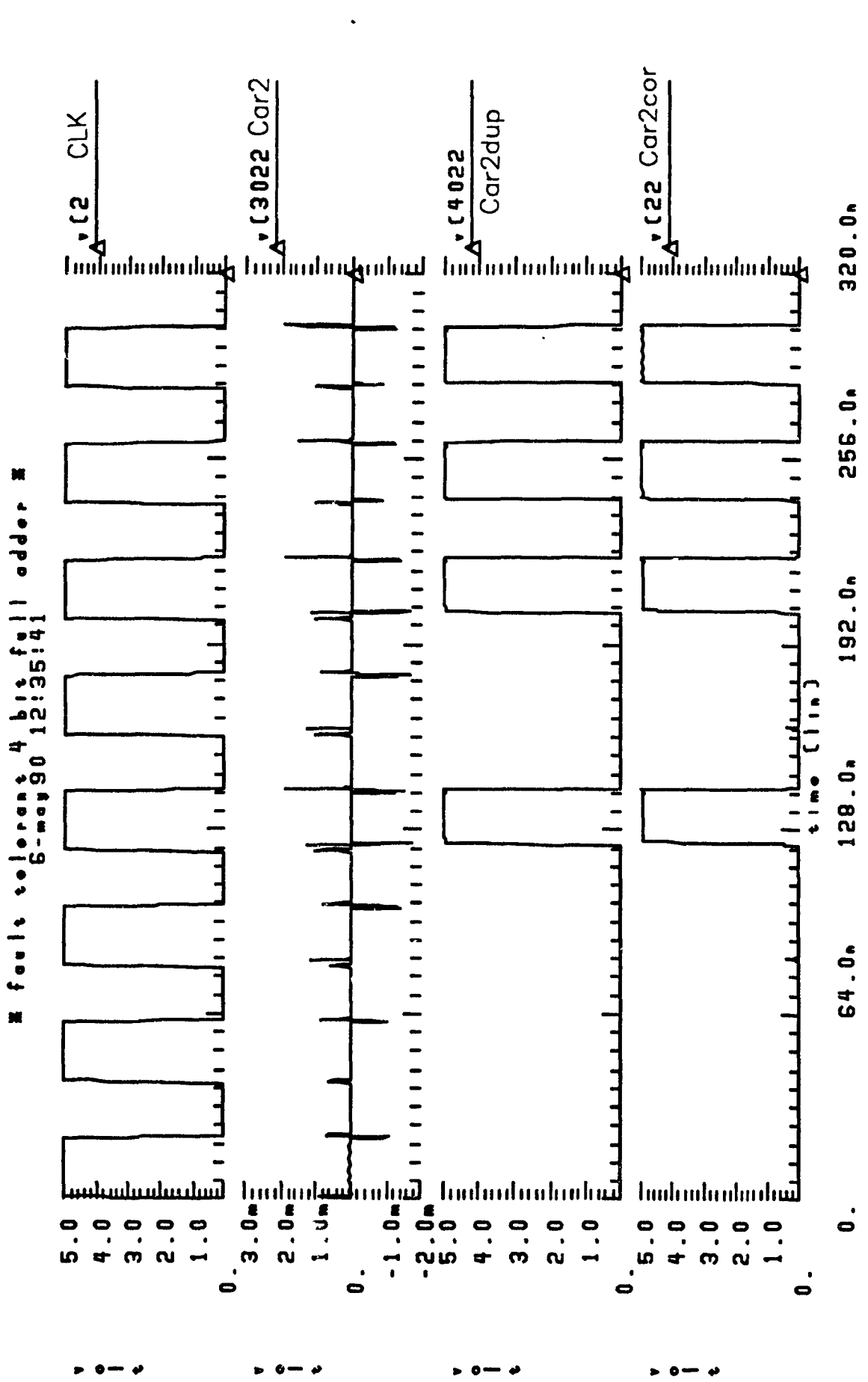


Fig.5.27.c Correcting Carry2

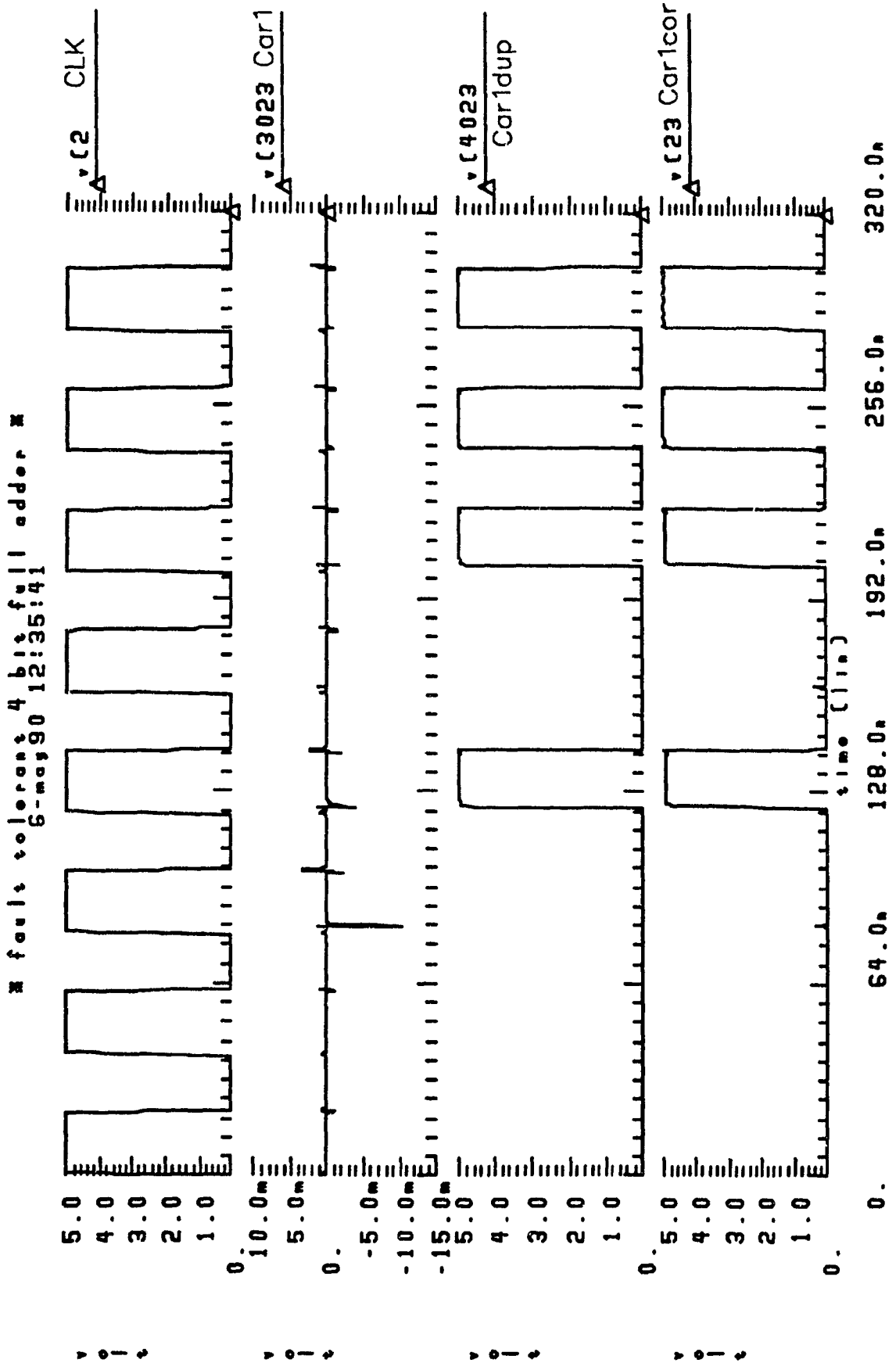


Fig.5.27.d Correcting Carry1

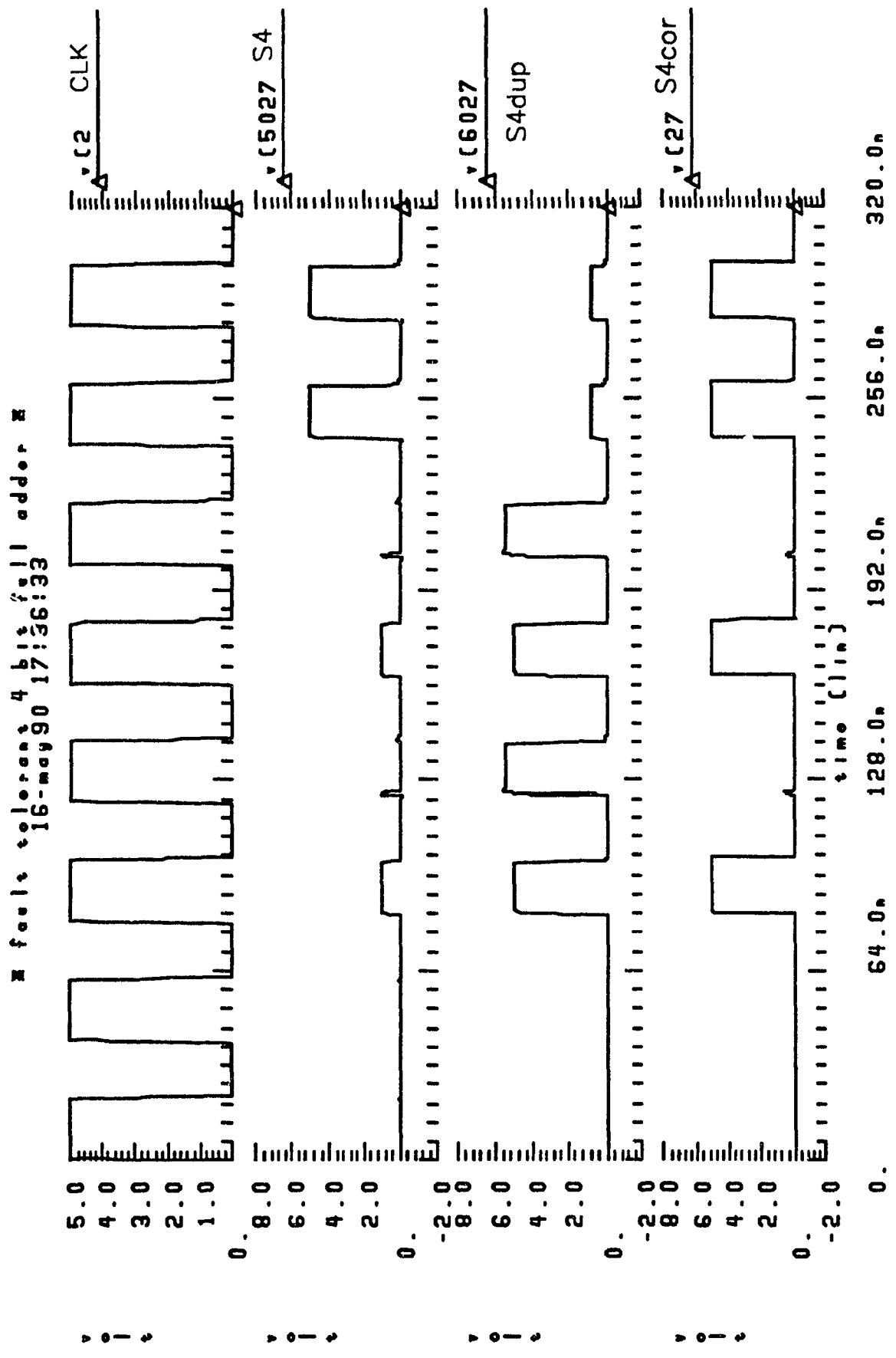


Fig.5.28.a Correcting Sum4

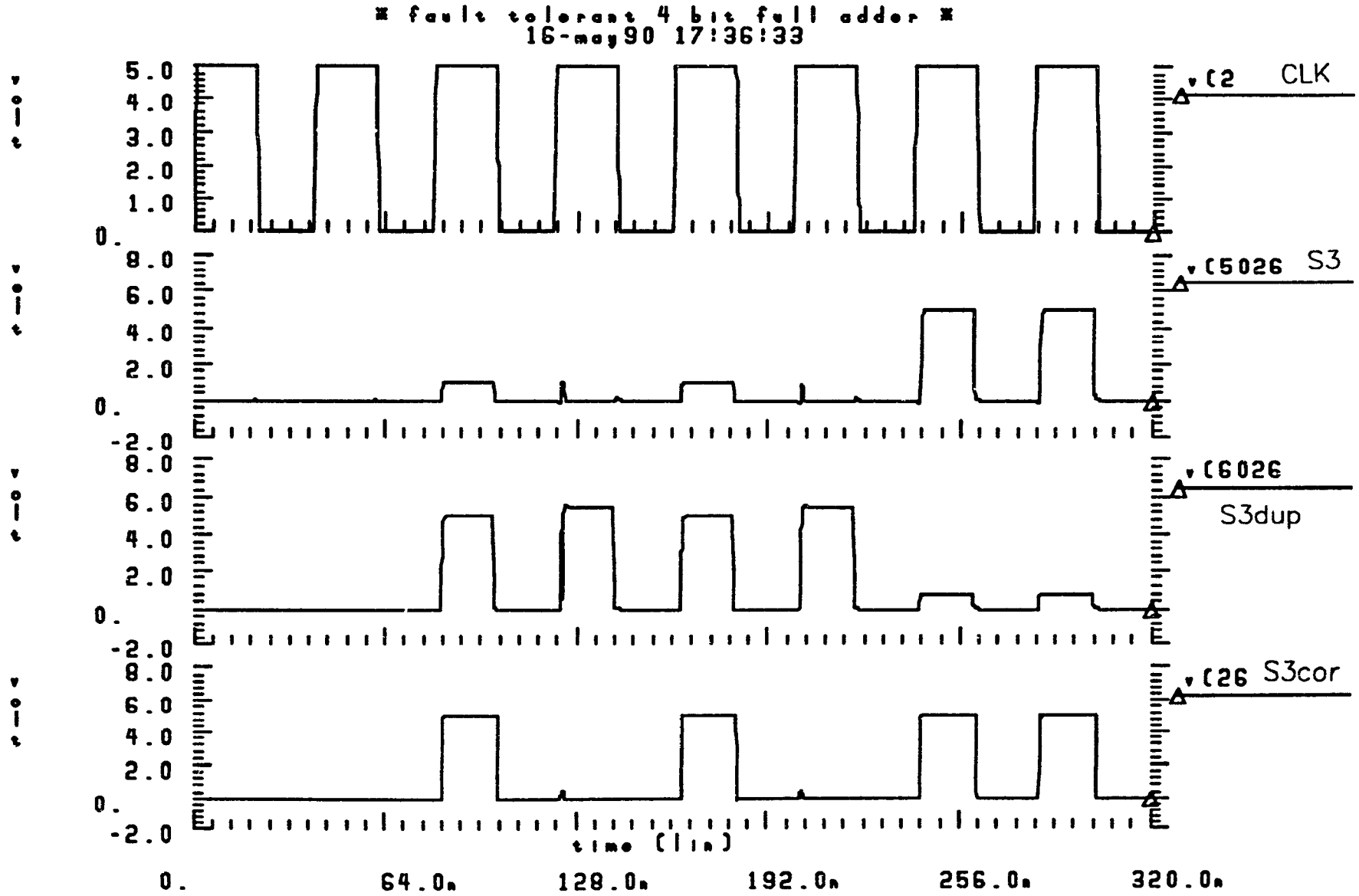


Fig.5.28.b Correcting Sum3

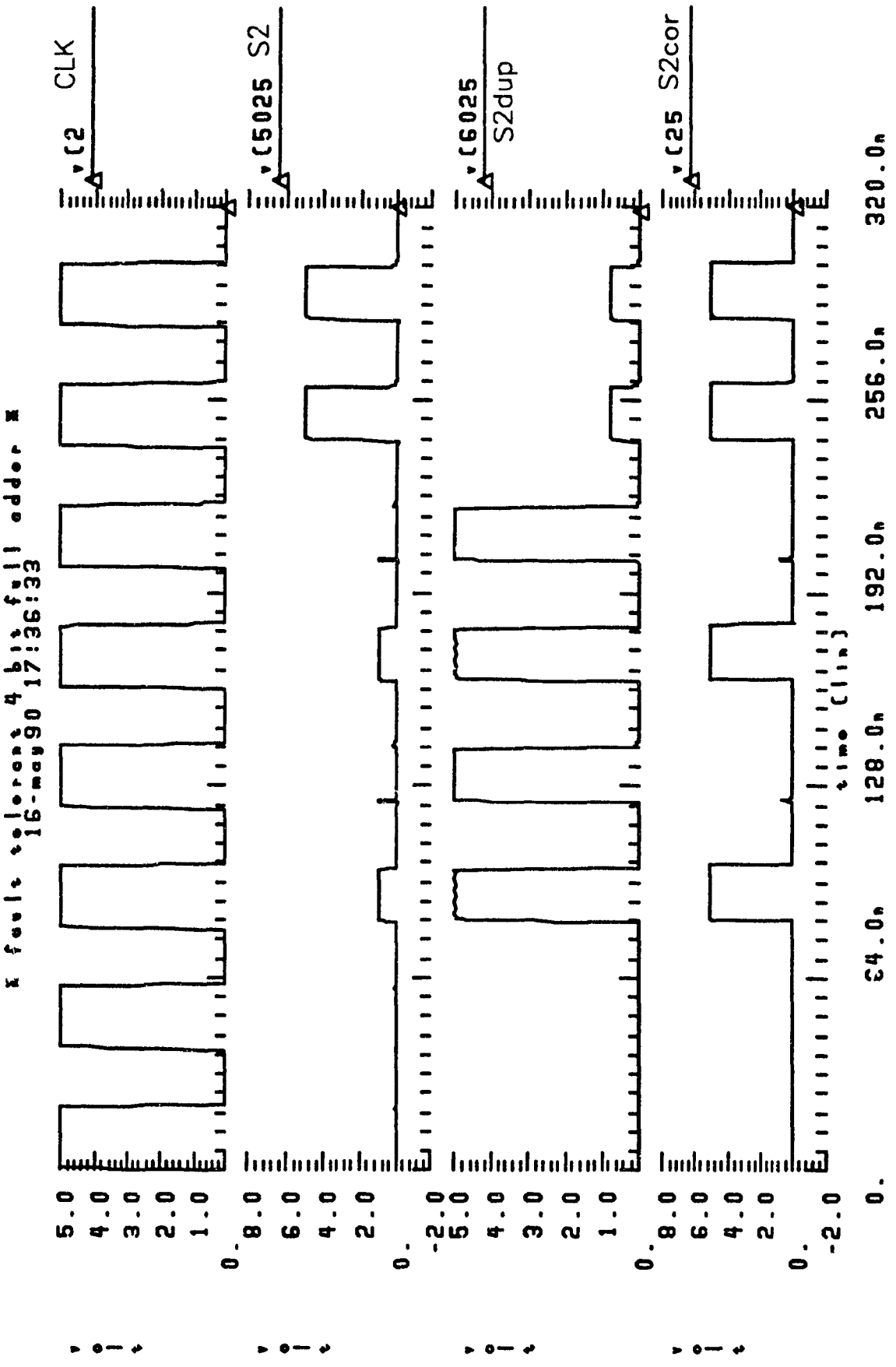


Fig.5.28.c Correcting Sum2

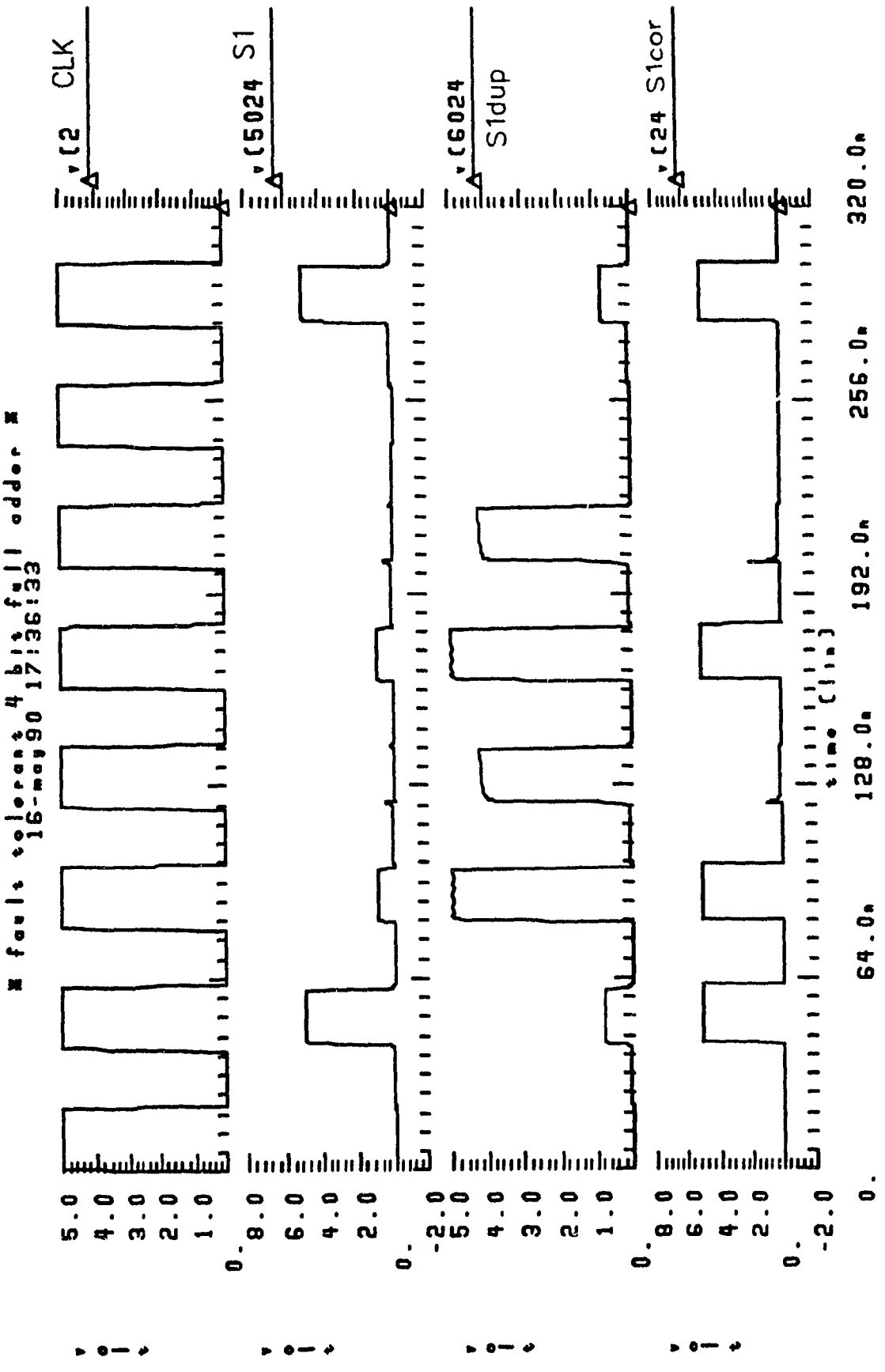


Fig.5.28.d Correcting Sum1

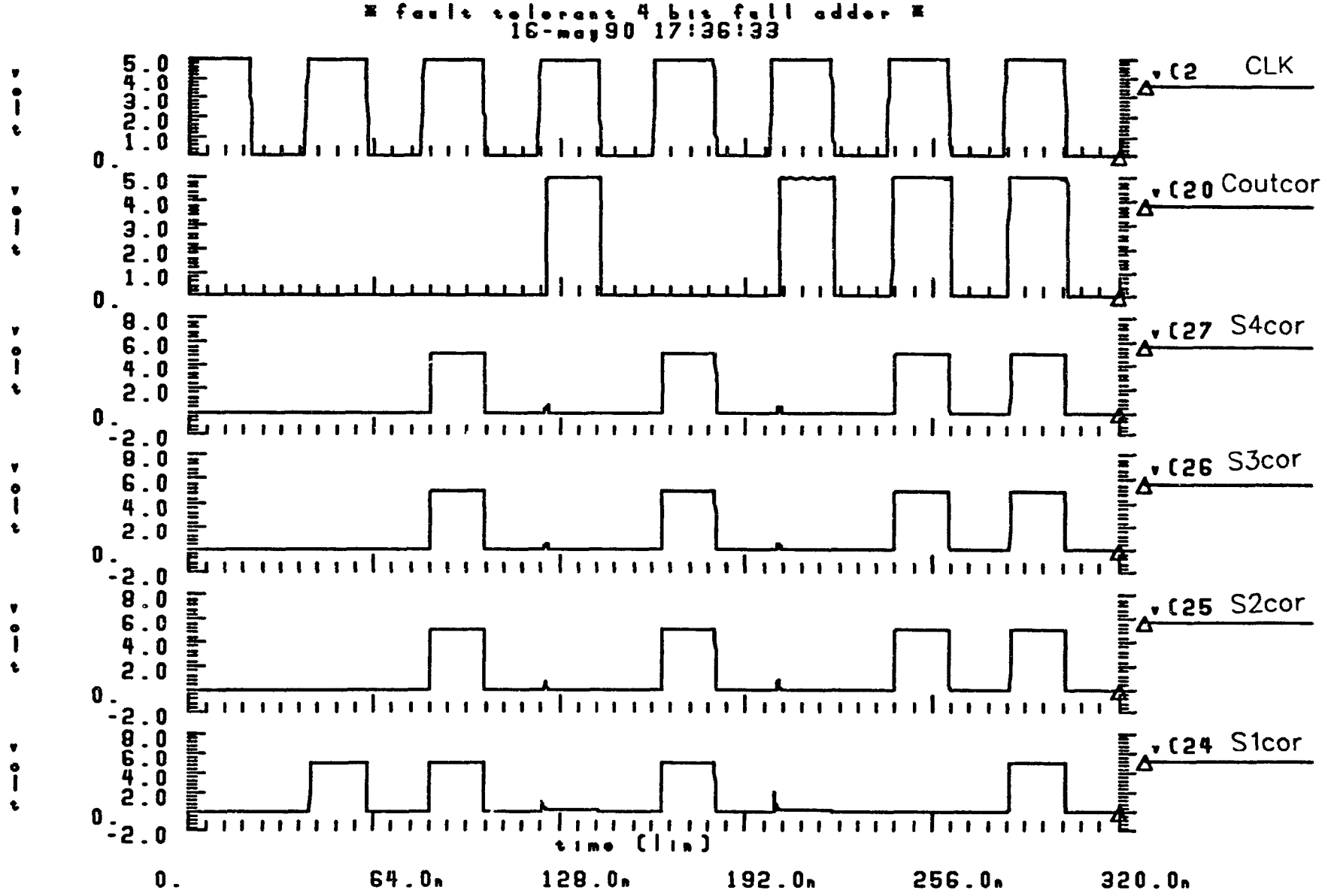


Fig.5.29 Overall Results from FTF4BADDER after faults Correction

would compare, in terms of area and fault coverage, to 356 transistors that would have been used had TMR been implemented for the adder as one part.

In terms of speed our circuit compares favorably with TMR implemented on the original circuit as one part. In our circuit the worst case propagation delay is 2.6ns which compares to 2.1ns for TMR. Even though our circuit is 0.5ns slower than TMR, our implementation is better if area, speed, and fault coverage are combined. Finally, the same reliability equations, developed for the 4-bit fault tolerant partial adder, hold true for the full 4-bit fault tolerant adder, which again indicates that our adder has better reliability.

CHAPTER 6: CONCLUSION

In this thesis the design of a pipelined adder accumulator is given, with a 24-bit implementation. The adder accumulator is used as a threshold generator. In order to achieve the design, a 4-bit partial adder and a full 4-bit adder were designed. The adders are designed in MODL which gave the advantage of small propagation delays, small area, in addition to eliminating the charge redistribution problems which normally occurs in standard domino logic. The full 4-bit adder is compared to other types of adders (hierarchical), and to adders of the same type (carry-look-ahead) but implemented in other types of logic (standard domino, and static CMOS), and was proven to be both faster, and more area efficient. The design of the adder accumulator was then presented, based on the adders, implemented in a pipeline design. The delay of the adder accumulator is proportional to the delay of the full 4-bit adder, plus it requires five extra cycles (after the last pixel of the frame arrives) to clear the pipeline. This is acceptable as the delay in the adder is 0.92ns while the pixels are generated from the A/D converter with a rate of 18MHz.

In the second part of the thesis, fault analysis of the two adders is presented, plus the design of the fault tolerant adders. Fault tolerance was achieved by hardware duplication for detection, and the addition of extra circuitry for correction. The fault tolerant adders are proven to be able to handle most of the studied faults, in addition to the ability to handle more than one failure in any of the adder's modules. Comparison with Triple Modular Redundancy showed that our fault tolerant design compared favorably in terms of area, speed, and fault coverage.

REFERENCES

[1]. Han, T., and D.A. Carlson, "Fast area-efficient VLSI adders", IEEE 1987 8th Symposium on Computer Arithmatic, pp. 49-56, 1987.

[2]. Weste, N., and K. Eshraglan, Principles of CMOS VLSI Design: A System Perspective. Reading, MA: Addison-Wesley Publishing Company, 1985.

[3]. Al-Khalili, A.J., D. Glenn, and D. Al-Khalili, "A smoother chip for Image Processing", Proceedings of the IEEE Conference on Industrial Electronics, November 1989.

[4]. Gonzalez, R.C., and P. Wintz, Digital Image Processing. Reading, MA: Addison-Wesley Publishing Company, 1977.

[5]. Boyle, R.D., and R.C. Thomas, Computer Vision: a First Course. Oxford, U.K.: Blackwell Scientific Publications, 1988.

[6]. Chan, P.K., and M.D.F. Schlag, "Analysis and design of CMOS Manchester adders with variable Carry-Skip", IEEE trans. on Computers, pp. 983-992, August 1990.

[7]. Guyot, A., B. Hochet, and J.M. Muller, "A way to build efficient Carry-Skip adders", IEEE trans. on Computers, pp. 1144-1152, October 1987.

[8]. Goncalves, N.F., and H.J. DeMan, "NORA: A racefree dynamic CMOS technique for pipelined logic structures", IEEE J. Solid State Circuits, Vol. SC-18, pp. 261-266, June 1983.

[9]. Krambeck, R.H., C.M. Lee, and H.F.S. Law, "High-speed compact circuits with CMOS", IEEE J. Solid State Circuits, Vol. SC-17, pp. 614-619, June 1982.

[10]. Lee, C.M., and E.W. Szeto, "Zipper CMOS", IEEE Circuits and Devices, pp. 10-16, May 1986.

[11]. Hwang, I.S., and A.L. Fisher, "Ultrafast compact 32-bit CMOS adders in multiple output domino logic", IEEE J. Solid State Circuits, Vol. SC-24, pp. 358-369, April 1989.

[12]. Shoji, M., "FET scaling in domino CMOS gates", IEEE J. Solid State Circuits, Vol. SC-20, pp. 1067-1071, October 1985.

[13]. Al-Arrian, S.A., and D.P. Agrawal, "Physical failures and fault models of CMOS circuits", IEEE trans. Circuits and Systems, Vol. CAS-34, pp. 269-277, March 1987.

[14]. Wadsack, R.L., "Fault modeling and logic simulators of CMOS and MOS integerated circuits", the Bell System Technical Journal, pp. 1449-1473, May/June 1978.

[15]. Wunderlich, H.J., and W. Rosenstiel, "On fault modeling for dynamic MOS circuits", IEEE Design Automation Conference, pp. 540-546, 1986.

[16]. Banerjee, P., and J.A. Abraham, "Characterization and testing of physical failures in MOS logic circuits", IEEE Design and Test Computing, pp. 76-86, August 1984.

[17]. Mangir, T.E., "Sources of failures and yield improvement for VLSI", Proceedings of IEEE, pp. 690-708, June 1984.

[18]. Timoc, C., et al., "Logic model of physical failures", Int'l Test Conference, pp. 546-553, 1983.

[19]. Gallay, J., Y. Crouzet, and M. Vergniault, "Physical vs. logical fault models MOS LSI circuits", IEEE trans. on Computers, pp. 527-531, June, 1980.

[20]. Banerjee, P., and J.A. Abraham, "Fault characterization of VLSI MOS circuits", Int'l Conference on Circuits and Computers, pp. 564-568, 1982.

[21]. Siewiorek, D.P., and R.S. Swarz, The Theory and Practice of Reliable System Design. Bedford, MA:Digital Press, 1982.

[22]. Pradhan, D.K. Fault Tolerant Computing: Theory and Techniques. Englewood Cliffs, NJ: Prentice Hall, 1986.

[23]. Johnson, B.W., J.H. Aylor, and H.H. Hana, "Efficient use of time and hardware redundancy for concurrent error detection in a 32-bit adder", IEEE J. Solid State Circuits, Vol. Sc-23, pp. 208-215, February 1988.

[24]. Reynolds, D.A., and G. Metze, "Fault detection capabilities of alternating logic", IEEE trans. on Computers, pp. 1093-1098, December 1978.

[25]. Patel, J.H., and L.Y. Fung, "Concurrent error detection in ALU's by recomputing using shifted operands", IEEE trans. on Computers, pp. 589-595, July 1982.

[26]. Johnson, B.W., Design and Analysis of Fault Tolerant Digital Systems. Reading, MA: Addison-Wesley Publishing Company, 1989.

[27]. Wu, C.C., L. Tzeng, and T. Wu, "Time redundant fault masking in ALUs", IEEE 1987 8th Symposium on Computer Arithmatic, pp. 239-243, 1987.

APPENDIX A

SPICE DECK FOR 4BIT ADDER

```
* simulating a 4 bit adder *
.OPTIONS post=2 ACCT OPTS probe
.options nomod
**********************************************************************
* model for the PMOS                                                 *
**********************************************************************
.MODEL p1 PMOS (LEVEL=8 VTO=-0.9665 KP=0.000 GAMMA=0.001 PHI=0.000
+LAMBDA=0.000 RD=91.75 RS=91.75 CBD=0.000 CBS=0.000 IS=100E-18 PB=0.8
+CGSO=93.84E-12 CGDO=93.84E-12 CGBO=940E-12 RSH=0.000 CJ=410E-6 MJ=0.540
+CJSW=3.4E-10 MJSW=0.300 JS=100E-6 TOX=25E-9 NSUB=1.983E+16 NSS=0.000
+NFS=2.989E+12 TPG=1.000 XJ=58.41E-9 LD=147.5E-9 UO=378.4 UCRIT=1.0E+4
+UEXP=0.2550 UTRA=0.000 VMAX=3.642E+4 NEFF=1.815 XQC=1.000 FC=0.5
+DELTA=1.101 THETA=0.000 ETA=0.000 KAPPA=0.2 )
**********************************************************************
* model for the NMOS                                                 *
**********************************************************************
.MODEL n1 NMOS (LEVEL=8 VTO=0.8199 KP=0.000 GAMMA=0.001 PHI=0.000
+LAMBDA=0.000 RD=33.26 RS=33.26 CBD=0.000 CBS=0.000 IS=100E-18 PB=0.8
+CGSO=17.94E-12 CGDO=17.94E-12 CGBO=940E-12 RSH=0.000 CJ=290E-6 MJ=0.486
+CJSW=3.3E-10 MJSW=0.330 JS=100E-6 TOX=25E-9 NSUB=1.500E+16 NSS=0.000
+NFS=1.000E+12 TPG=1.000 XJ=100E-9 LD=107.3E-9 UO=813.1 UCRIT=1.0E+4
+UEXP=0.1429 UTRA=0.000 VMAX=4.925E+4 NEFF=5.147 XQC=1.000 FC=0.5
+DELTA=0.6954 THETA=0.000 ETA=0.000 KAPPA=0.2 )
**********************************************************************
* clock and power signals                                            *
**********************************************************************
vdd 1 0 dc 5v
vclk 2 0 pulse(0 5 0 1ns 1ns 25ns 50ns)
vb4 5 0 pulse(0 5 26ns 1ns 1ns 99ns 150ns)
vb3 9 0 pulse(0 5 26ns 1ns 1ns 99ns 150ns)
vb2 13 0 pulse(0 5 26ns 1ns 1ns 99ns 150ns)
vb1 17 0 pulse(0 5 26ns 1ns 1ns 49ns 150ns)
vcin 19 0 pulse(0 5 26ns 1ns 1ns 99ns 150ns)
**********************************************************************
* transmission gate for Xor                                          *
**********************************  ***********************************
.subckt trg 1 102 104 105
*1 is VDD *
*102 is C input , 104 is B input, 105 is output *
mb11 103 102 1 1 p1 l=1.2u w=3.6u ad=10.8p pd=13.2u
mb12 103 102 0 0 n1 l=1.2u w=2.4u ad=7.2p pd=10.8u
mb13 105 104 102 1 p1 l=1.2u w=3.6u ad=10.8p pd=13.2u
mb14 105 104 103 0 n1 l=1.2u w=2.4u ad=7.2p pd=10.8u
mb15 104 102 105 1 p1 l=1.2u w=3.6u ad=10.8p pd=13.2u
mb16 104 103 105 0 n1 l=1.2u w=2.4u ad=7.2p pd=10.8u
.ends trg
**********************************************************************
* inv definitions                                                    *
**********************************************************************
.subckt inv 1 201 202
* 1 is VDD *
* 201 is input , 202 is output *
mb21 202 201 1 1 p1 l=1.2u w=3.6u ad=10.8p pd=13.2u
mb22 202 201 0 0 n1 l=1.2u w=2.4u ad=7.2p pd=10.8u
.ends inv
```

```
*******************************************************************************
* adder circuit definitions                                                   *
*******************************************************************************

mct1 3 2 1 1 p1 l=1.2u w=3.6u ad=10.8p pd=13.2u
x1 1 3 4 inv
*                                   *4 == COUT
mct2 3 5 6 0 n1 l=1.2u w=3.6u ad=10.8p pd=13.2u
mct3 6 2 1 1 p1 l=1.2u w=3.6u ad=10.8p pd=13.2u
x2 1 6 7 inv
*                                   *6 == C3 b    7 == C3
x3 1 7 5 8 trg
*                                   *5 == B4      7 == C3      8 == S4
mct4 6 9 10 0 n1 l=1.2u w=4.8u ad=14.4p pd=15.6u
mct5 10 2 1 1 p1 l=1.2u w=3.6u ad=10.8p pd=13.2u
x4 1 10 11 inv
*                                   *10 == C2 b   11 == C2
x5 1 11 9 12 trg
*                                   *9 == B3      11 == C2     12 == S3
mct6 10 13 14 0 n1 l=1.2u w=6u ad=18p pd=18u
mct7 14 2 1 1 p1 l=1.2u w=3.6u ad=10.8p pd=13.2u
x6 1 14 15 inv
*                                   *14 == C1 b   15 == C1
x7 1 15 13 16 trg
*                                   *13 == B2     15 == C1     16 == S2
mct8 14 17 18 0 n1 l=1.2u w=8.4u ad=25.2p pd=22.8u
mct9 18 19 21 0 n1 l=1.2u w=8.4u ad=25.2p pd=22.8u
x8 1 19 17 20 trg
*                                   *17 == B1     19 == Cin    20 == S1
mc_10 21 2 0 0 n1 l=1.2u w=9.6u ad=28.8p pd=25.2u
.tran 2ns 150ns
.print tran v(2) v(4) v(8) v(12) v(16) v(20)
.print tran v(5) v(9) v(13) v(17) v(19)
.meas delay1 trig v(2) val=2.5 rise=2 targ v(4) val=2.5 rise=1
.meas delay2 trig v(2) val=2.5 rise=2 targ v(8) val=2.5 fall=1
.meas delay3 trig v(2) val=2.5 rise=2 targ v(12) val=2.5 fall=1
.meas delay4 trig v(2) val=2.5 rise=2 targ v(16) val=2.5 fall=1
.end
```

SPICE DECK FOR 4FULL BIT ADDER

```
* 4 bit full adder *
.OPTIONS post=2 ACCT OPTS probe
.options nomod limpts=5000
*************************************************************************
* model for the PMOS                                                    *
*************************************************************************
.MODEL p1 PMOS (LEVEL=8 VTO=-0.9665 KP=0.000 GAMMA=0.001 PHI=0.000
+LAMBDA=0.000 RD=91.75 RS=91.75 CBD=0.000 CBS=0.000 IS=100E-18 PB=0.8
+CGSO=93.84E-12 CGDO=93.84E-12 CGBO=940E-12 RSH=0.000 CJ=410E-6 MJ=0.540
+CJSW=3.4E-10 MJSW=0.300 JS=100E-6 TOX=25E-9 NSUB=1.983E+16 NSS=0.000
+NFS=2.989E+12 TPG=1.000 XJ=58.41E-9 LD=147.5E-9 UO=378.4 UCRIT=1.0E+4
+UEXP=0.2550 UTRA=0.000 VMAX=3.642E+4 NEFF=1.815 XQC=1.000 FC=0.5
+DELTA=1.101 THETA=0.000 ETA=0.000 KAPPA=0.2 )
*************************************************************************
* model for the NMOS                                                    *
*************************************************************************
.MODEL n1 NMOS (LEVEL=8 VTO=0.8199 KP=0.000 GAMMA=0.001 PHI=0.000
+LAMBDA=0.000 RD=33.26 RS=33.26 CBD=0.000 CBS=0.000 IS=100E-18 PB=0.8
+CGSO=17.94E-12 CGDO=17.94E-12 CGBO=940E-12 RSH=0.000 CJ=290E-6 MJ=0.486
+CJSW=3.3E-10 MJSW=0.330 JS=100E-6 TOX=25E-9 NSUB=1.500E+16 NSS=0.000
+NFS=1.000E+12 TPG=1.000 XJ=100E-9 LD=107.3E-9 UO=813.1 UCRIT=1.0E+4
+UEXP=0.1429 UTRA=0.000 VMAX=4.925E+4 NEFF=5.147 XQC=1.000 FC=0.5
+DELTA=0.6954 THETA=0.000 ETA=0.000 KAPPA=0.2 )
*************************************************************************
* clock and power signals                                               *
*************************************************************************
vdd 1 0 dc 5v
vclk 2 0 pulse(0 5 0 1ns 1ns 20ns 40ns)
vc0 19 0 pulse(0 5 21ns 1ns 1ns 39ns 80ns)
va1 3 0 pulse(0 5 141ns 1ns 1ns 159ns 320ns)
vb1 4 0 pulse(0 5 61ns 1ns 1ns 79ns 160ns)
va2 7 0 pulse(0 5 141ns 1ns 1ns 159ns 320ns)
vb2 8 0 pulse(0 5 61ns 1ns 1ns 79ns 160ns)
va3 11 0 pulse(0 5 141ns 1ns 1ns 159ns 320ns)
vb3 12 0 pulse(0 5 61ns 1ns 1ns 79ns 160ns)
va4 15 0 pulse(0 5 141ns 1ns 1ns 159ns 320ns)
vb4 16 0 pulse(0 5 61ns 1ns 1ns 79ns 160ns)
*************************************************************************
* transmission gate for Xor                                             *
*************************************************************************
.subckt trg 1 102 104 105
*1 is VDD *
*102 is C input , 104 is B input, 105 is output *
mb11 103 102 1 1 p1 l=1.2u w=3.6u ad=10.8p pd=13.2u
mb12 103 102 0 0 n1 l=1.2u w=2.4u ad=7.2p pd=10.8u
mb13 105 104 102 1 p1 l=1.2u w=3.6u ad=10.8p pd=13.2u
mb14 105 104 103 0 n1 l=1.2u w=2.4u ad=7.2p pd=10.8u
mb15 104 102 105 1 p1 l=1.2u w=3.6u ad=10.8p pd=13.2u
mb16 104 103 105 0 n1 l=1.2u w=2.4u ad=7.2p pd=10.8u
.ends trg
*************************************************************************
* inv1 definitions                                                      *
*************************************************************************
.subckt inv1 1 201 202
* 1 is VDD *
* 201 is input , 202 is output *
mb21 202 201 1 1 p1 l=1.2u w=3.6u ad=10.8p pd=13.2u
mb22 202 201 0 0 n1 l=1.2u w=2.4u ad=7.2p pd=10.8u
.ends inv1
```

```
*********************************************************************************
* genpg (block1) definitions                                                    *
*********************************************************************************
.subckt genpg 1 2 305 307 304  309
* 1 is VDD , 2 is CLK *
* 305 is Bi , 307 is Ai *
* 304 is Gi , 309 is Pi *
mct31 303 2 1 1 p1 l=1.2u w=3.6u ad=10.8p pd=13.2u
x31 1 303 304 inv1
*                                    *304 == G
mct32 303 305 306 0 n1 l=1.2u w=2.4u ad=7.2p pd=10.8u
mct33 306 307 308 0 n1 l=1.2u w=3.6u ad=10.8p pd=13.2u
*
x32 1 305 307 309 trg
*                                    *309 == P
mct34 308 2 0 0 n1 l=1.2u w=4.8u ad=14.4p pd=15.6u
.ends genpg
*********************************************************************************
* cgen (block2) definitions                                                     *
*********************************************************************************
.subckt cgen 1 2 406 409 410 413 414 417 418 421 420 405 408 412 416
* 1 is VDD , 2 is CLK *
* 406 is p4 , 409 is g4 *
* 410 is p3 , 413 is g3 *
* 414 is p2 , 417 is g2 *
* 418 is p1 , 421 is g1 , 420 is C0 **
* 405 is C4 (cout) , 408 is C3 , 412 is C2 , 416 is C1 *
mct41 404 2 1 1 p1 l=1.2u w=3.6u ad=10.8p pd=13.2u
x41 1 404 405 inv1
*                                    *405 == COUT (C4)
mct42 404 406 407 0 n1 l=1.2u w=3.6u ad=10.8p pd=13.2u
mct43 404 409 422 0 n1 l=1.2u w=3.6u ad=10.8p pd=13.2u
mct44 407 2 1 1 p1 l=1.2u w=3.6u ad=10.8p pd=13.2u
x42 1 407 408 inv1
*                                    *408 == C3
mct45 407 410 411 0 n1 l=1.2u w=4.8u ad=14.4p pd=15.6u
mct46 407 413 422 0 n1 l=1.2u w=4.8u ad=14.4p pd=15.6u
mct47 411 2 1 1 p1 l=1.2u w=3.6u ad=10.8p pd=13.2u
x43 1 411 412 inv1
*                                    *412 == C2
mct48 411 414 415 0 n1 l=1.2u w=6u ad=18p pd=18u
mct49 411 417 422 0 n1 l=1.2u w=6u ad=18p pd=18u
mct50 415 2 1 1 p1 l=1.2u w=3.6u ad=10.8p pd=13.2u
x44 1 415 416 inv1
*                                    *416 == C1
mct51 415 418 419 0 n1 l=1.2u w=7.2u ad=21.6p pd=20.4u
mct52 419 420 422 0 n1 l=1.2u w=8.4u ad=25.2p pd=22.8u
mct53 415 421 422 0 n1 l=1.2u w=7.2u ad=21.6p pd=20.4u
mct54 422 2 0 0 n1 l=1.2u w=9.6u ad=28.8p pd=25.2u
.ends cgen
*********************************************************************************
* sgen (block3) definitions                                                     *
*********************************************************************************
.subckt sgen 1 503 504 505
* 1 is VDD *
* 503 is Ci , 504 is Pi , 505 is Si *
X51 1 503 504 505 trg
.ends sgen
```

```
*********************************************************************
* adder circuit                                                     *
*********************************************************************
x1 1 2 3 4 5 6 genpg
*                          *  3 = A1   4 = B1   5 = G1   6 = P1
x2 1 2 7 8 9 10 genpg
*                          *  7 = A2   8 = B2   9 = G2  10 = P2
x3 1 2 11 12 13 14 genpg
*                          * 11 = A3  12 = B3  13 = G3  14 = P3
x4 1 2 15 16 17 18 genpg
*                          * 15 = A4  16 = B4  17 = G4  18 = P4
x5 1 2 18 17 14 13 10 9 6 5 19 20 21 22 23 cgen
*                          * 18 = P4  14 = P3  10 = P2   6 = P1
*                          * 17 = G4  13 = G3   9 = G2   5 = G1
*                          * 19 = C0
*                          * 20 = C4  21 = C3  22 = C2  23 = C1
x6 1 19 6 24 sgen
*                          * 19 = C0   6 = P1  24 = S1
x7 1 23 10 25 sgen
*                          * 23 = C1  10 = P2  25 = S2
x8 1 22 14 26 sgen
*                          * 22 = C2  14 = P3  26 = S3
x9 1 21 18 27 sgen
*                          * 21 = C3  18 = P4  27 = S4
.tran 2ns 320ns

.print tran v(2) v(20) v(27) v(26) v(25) v(24)
.print tran v(19) v(3) v(4)
.meas delay1 trig v(2) val=2.5 rise=4 targ v(20) val=2.5 rise=1
.meas delay2 trig v(2) val=2.5 rise=6 targ v(20) val=2.5 rise=2
.meas delay3 trig v(2) val=2.5 rise=7 targ v(20) val=2.5 rise=3
.meas delay4 trig v(2) val=2.5 rise=8 targ v(20) val=2.5 rise=4
.meas delay5 trig v(2) val=2.5 rise=4 targ v(27) val=2.5 fall=1
.meas delay6 trig v(2) val=2.5 rise=6 targ v(27) val=2.5 fall=2
.meas delay7 trig v(2) val=2.5 rise=7 targ v(27) val=2.5 rise=3
.meas delay8 trig v(2) val=2.5 rise=8 targ v(27) val=2.5 rise=4
.meas delay9 trig v(2) val=2.5 rise=4 targ v(26) val=2.5 fall=1
.meas delay10 trig v(2) val=2.5 rise=6 targ v(26) val=2.5 fall=2
.meas delay11 trig v(2) val=2.5 rise=7 targ v(26) val=2.5 rise=3
.meas delay12 trig v(2) val=2.5 rise=8 targ v(26) val=2.5 rise=4
.meas delay13 trig v(2) val=2.5 rise=4 targ v(25) val=2.5 fall=1
.meas delay14 trig v(2) val=2.5 rise=6 targ v(25) val=2.5 fall=2
.meas delay15 trig v(2) val=2.5 rise=7 targ v(25) val=2.5 rise=3
.meas delay16 trig v(2) val=2.5 rise=8 targ v(25) val=2.5 rise=4
.end
```

SPICE DECK FOR 24BIT ADDER ACCUMULATOR

```
* simulating an adder accumulator *
.OPTIONS post=2 ACCT OPTS probe
.cptions nomod
*********************************************************************
* model for the PMOS                                                *
*********************************************************************
.MODEL p1 PMOS (LEVEL=8 VTO=-0.9665 KP=0.000 GAMMA=0.001 PHI=0.000
+LAMBDA=0.000 RD=91.75 RS=91.75 CBD=0.000 CBS=0.000 IS=100E-18 PB=0.8
+CGSO=93.84E-12 CGDO=93.84E-12 CGBO=940E-12 RSH=0.000 CJ=410E-6 MJ=0.540
+CJSW=3.4E-10 MJSW=0.300 JS=100E-6 TOX=25E-9 NSUB=1.983E+16 NSS=0.000
+NFS=2.989E+12 TPG=1.000 XJ=58.41E-9 LD=147.5E-9 UO=378.4 UCRIT=1.0E+4
+UEXP=0.2550 UTRA=0.000 VMAX=3.642E+4 NEFF=1.815 XQC=1.000 FC=0.5
+DELTA=1.101 THETA=0.000 ETA=0.000 KAPPA=0.2 )
*********************************************************************
* model for the NMOS                                                *
*********************************************************************
.MODEL n1 NMOS (LEVEL=8 VTO=0.8199 KP=0.000 GAMMA=0.001 PHI=0.000
+LAMBDA=0.000 RD=33.26 RS=33.26 CBD=0.000 CBS=0.000 IS=100E-18 PB=0.8
+CGSO=17.94E-12 CGDO=17.94E-12 CGBO=940E-12 RSH=0.000 CJ=290E-6 MJ=0.486
+CJSW=3.3E-10 MJSW=0.330 JS=100E-6 TOX=25E-9 NSUB=1.500E+16 NSS=0.000
+NFS=1.000E+12 TPG=1.000 XJ=100E-9 LD=107.3E-9 UO=813.1 UCRIT=1.0E+4
+UEXP=0.1429 UTRA=0.000 VMAX=4.925E+4 NEFF=5.147 XQC=1.000 FC=0.5
+DELTA=0.6954 THETA=0.000 ETA=0.000 KAPPA=0.2 )
*********************************************************************
* clock and power signals                                           *
*********************************************************************
vdd 1 0 dc 5v
vclk 2 0 pulse(0 5 8ns 1ns 1ns 20ns 40ns)
vdv 3 0 pulse(0 5 15ns 1ns 1ns 12ns 40ns)
vinda 4 0 pulse(0 5 1.5ns 0.1ns 0.1ns 5ns 40ns)
vc0 5 0 dc 0v
va1 6 0 pulse(0 5 0 1ns 1ns 150ns 1s)
vb1 7 0 pulse(0 5 0 1ns 1ns 150ns 1s)
va2 8 0 pulse(0 5 0 1ns 1ns 150ns 1s)
vb2 9 0 pulse(0 5 0 1ns 1ns 150ns 1s)
va3 10 0 pulse(0 5 0 1ns 1ns 150ns 1s)
vb3 11 0 pulse(0 5 0 1ns 1ns 150ns 1s)

va4 12 0 pulse(0 5 0 1ns 1ns 150ns 1s)
vb4 13 0 pulse(0 5 0 1ns 1ns 150ns 1s)
vinits 9999 0 pulse(0 5 0 .1ns .1ns 1ns 1s)
vinit 9998 0 dc 0v
*********************************************************************
* EXCLUSIVE OR CIRCUIT                                              *
*********************************************************************
.subckt trg 1 102 104 105
*1 is VDD *
*102 is C input , 104 is B input, 105 is output *
mb11 103 102 1 1 p1 l=1.2u w=3.6u ad=10.8p pd=13.2u
mb12 103 102 0 0 n1 l=1.2u w=2.4u ad=7.2p pd=10.8u
mb13 105 104 102 1 p1 l=1.2u w=3.6u ad=10.8p pd=13.2u
mb14 105 104 103 0 n1 l=1.2u w=2.4u ad=7.2p pd=10.8u
mb15 104 102 105 1 p1 l=1.2u w=3.6u ad=10.8p pd=13.2u
mb16 104 103 105 0 n1 l=1.2u w=2.4u ad=7.2p pd=10.8u

.ends trg
```

```
*******************************************************************************
* INVERTER CIRCUIT                                                            *
*******************************************************************************
.subckt inv 1 201 202
* 1 is VDD *
* 201 is input , 202 is output *
mb21 202 201 1 1 pl l=1.2u w=3.6u ad=10.8p pd=13.2u
mb22 202 201 0 0 nl l=1.2u w=2.4u ad=7.2p pd=10.8u
.ends inv
*******************************************************************************
* genpg (block1) definitions                                                  *
*******************************************************************************
.subckt genpg 1 2 305 307 304  309
* 1 is VDD , 2 is CLK *
* 305 is Bi , 307 is Ai *
* 304 is Gi , 309 is Pi *
mct31 303 2 1 1 pl l=1.2u w=3.6u ad=10.8p pd=13.2u
x31 1 303 304 inv
*                                    *304 == G
mct32 303 305 306 0 nl l=1.2u w=2.4u ad=7.2p pd=10.8u
mct33 306 307 308 0 nl l=1.2u w=3.6u ad=10.8p pd=13.2u
*
x32 1 305 307 309 trg
*                                    *309 == P
mct34 308 2 0 0 nl l=1.2u w=4.8u ad=14.4p pd=15.6u
.ends genpg
*******************************************************************************
* cgen (block2) definitions                                                   *
*******************************************************************************
.subckt cgen 1 2 406 409 410 413 414 417 418 421 420 405 408 412 416
* 1 is VDD , 2 is CLK *
* 406 is p4 , 409 is g4 *
* 410 is p3 , 413 is g3 *
* 414 is p2 , 417 is g2 *
* 418 is p1 , 421 is g1 , 420 is C0 *
* 405 is C4 (cout) , 408 is C3 , 412 is C2 , 416 is C1 *
mct41 404 2 1 1 pl l=1.2u w=3.6u ad=10.8p pd=13.2u
x41 1 404 405 inv
*                                    *405 == COUT (C4)
mct42 404 406 407 0 nl l=1.2u w=3.6u ad=10.8p pd=13.2u
mct43 404 409 422 0 nl l=1.2u w=3.6u ad=10.8p pd=13.2u
mct44 407 2 1 1 pl l=1.2u w=3.6u ad=10.8p pd=13.2u
x42 1 407 408 inv
*                                    *408 == C3
mct45 407 410 411 0 nl l=1.2u w=4.8u ad=14.4p pd=15.6u
mct46 407 413 422 0 nl l=1.2u w=4.8u ad=14.4p pd=15.6u
mct47 411 2 1 1 pl l=1.2u w=3.6u ad=10.8p pd=13.2u
x43 1 411 412 inv
*                                    *412 == C2
mct48 411 414 415 0 nl l=1.2u w=6u ad=18p pd=18u
mct49 411 417 422 0 nl l=1.2u w=6u ad=18p pd=18u
mct50 415 2 1 1 pl l=1.2u w=3.6u ad=10.8p pd=13.2u
x44 1 415 416 inv
*                                    *416 == C1
mct51 415 418 419 0 nl l=1.2u w=7.2u ad=21.6p pd=20.4u
mct52 419 420 422 0 nl l=1.2u w=8.4u ad=25.2p pd=22.8u
mct53 415 421 422 0 nl l=1.2u w=7.2u ad=21.6p pd=20.4u
mct54 422 2 0 0 nl l=1.2u w=9.6u ad=28.8p pd=25.2u
.ends cgen
```

```
*********************************************************************
* sgen (block3) definitions                                         *
*********************************************************************
.subckt sgen 1 503 504 505
* 1 is VDD *
* 503 is Ci , 504 is Pi , 505 is Si *
x51 1 503 504 505 trg
.ends sgen
*********************************************************************
* 4 full bit adder circuit                                          *
*********************************************************************
.subckt adder1 1 2 603 604 607 608 611 612 615 616 619 620 627 626
+625 624
* 1 Vdd, 2 CLK *
* 603, 607, 611, 615 are A1, A2, A3, and A4 *
* 604, 608, 612, 616 are B1, B2, B3, and B4 *
* 619 is Cin *
* 620,627, 626, 625, 624 are Cou, S4, S3, S2, and S1 *
x1 1 2 603 604 605 606 genpg
*                          * 603 = A1   604 = B1   605 = G1   606 = P1
x2 1 2 607 608 609 610 genpg
*                          * 607 = A2   608 = B2   609 = G2  610 = P2
x3 1 2 611 612 613 614 genpg
*                          * 611 = A3  612 = B3  613 = G3  614 = P3
x4 1 2 615 616 617 618 genpg
*                          * 615 = A4  616 = B4  617 = G4  618 = P4
x5 1 2 618 617 614 613 610 609 606 605 619 620 621 622 623 cgen
*                          * 618 = P4  614 = P3  610 = P2   606 = P1
*                          * 617 = G4  613 = G3   609 = G2   605 = G1
*                          * 619 = C0
*                          * 620 = C4  621 = C3  622 = C2  623 = C1
x6 1 619 606 624 sgen
*                          * 619 = C0   606 = P1  624 = S1
x7 1 623 610 625 sgen
*                          * 623 = C1  610 = P2  625 = S2
x8 1 622 614 626 sgen
*                          * 622 = C2  614 = P3  626 = S3
x9 1 621 618 627 sgen
*                          * 621 = C3  618 = P4  627 = S4
.ends adder1
*********************************************************************
* 4 bit adder circuit                                               *
*********************************************************************

.subckt adder2 1 2 705 709 713 717 719 704 708 712 716 720
* 1 is VDD, 2 is CLK *
* 705, 709, 713, 717, 719 are B4, B3, B2, B1, and Cin *
* 704, 708, 712, 716, 720 are Cout, S4, S3, S2, and S1 *
mb71 703 2 1 1 p1 l=1.2u w=3.6u ad=10.8p pd=13.2u
x701 1 703 704 inv
*                                *704 == COUT
mb72 703 705 706 0 n1 l=1.2u w=3.6u ad=10.8p pd=13.2u
mb73 706 2 1 1 p1 l=1.2u w=3.6u ad=10.8p pd=13.2u
x702 1 706 707 inv
*                                *706 == C3 b   707 == C3
x703 1 707 705 708 trg
*                                *705 == B4     707 == C3     708 == S4
mb74 706 709 710 0 n1 l=1.2u w=4.8u ad=14.4p pd=15.6u
mb75 710 2 1 1 p1 l=1.2u w=3.6u ad=10.8p pd=13.2u
x704 1 710 711 inv
*                                *710 == C2 b  711 == C2
x705 1 711 709 712 trg
*                                *709 == B3     711 == C2    712 == S3
```

```
mb76 710 713 714 0 nl l=1.2u w=6u ad=18p pd=18u
mb77 714 2 1 1 pl l=1.2u w=3.6u ad=10.8p pd=13.2u
x706 1 714 715 inv
*                                   *714 == C1 b  715 == C1
x707 1 715 713 716 trg
*                                   *713 == B2    715 == C1    716 == S2
mb78 714 717 718 0 nl l=1.2u w=8.4u ad=25.2p pd=22.8u
mb79 718 719 721 0 nl l=1.2u w=8.4u ad=25.2p pd=22.8u
x708 1 719 717 720 trg
*                                   *717 == B1    719 == Cin   720 == S1
mb80 721 2 0 0 nl l=1.2u w=9.6u ad=28.8p pd=25.2u
.ends adder2
*******************************************************************************
* nand gate                                                                    *
*******************************************************************************
.subckt nand 1 902 903 904
* 1 is VDD *
* 902 is data_available, 903 is i/p *
* 904 is o/p *
mb90 904 902 1 1 pl l=1.2u w=3.6u ad=10.8p pd=13.2u
mb91 904 903 1 1 pl l=1.2u w=3.6u ad=10.8p pd=13.2u
mb92 904 902 905 0 nl l=1.2u w=2.4u ad=9p pd=11u
mb93 905 903 0 0 nl l=1.2u w=3.6u ad=10.8p pd=13.2u
.ends nand
*******************************************************************************
* D flip flop                                                                  *
*******************************************************************************
.subckt dff 1 1002 1003 1007
* 1 is VDD *
* 1002 is data_available, 1003 is i/p *
* 1007 is o/p *
x1001 1 1003 1005 inv
x1002 1 1002 1003 1004 nand
x1003 1 1002 1005 1006 nand
x1004 1 1004 1008 1007 nand
x1005 1 1006 1007 1008 nand
.ends dff
*******************************************************************************
* STORE INPUT DATA                                                             *
*******************************************************************************
.subckt stdata 1 4 1103 1104 1105 1106 1107 1108 1109 1110 1111 1112
+1113 1114 1115 1116 1117 1118 1119 1120
* 1 is VDD, 4 is input data valid signal *
* 1103, 1104, 1105, 1106, 1107, 1108, 1109, 1110, 1111  are data to be stored *
* 1112, 1113, 1114, 1115, 1116, 1117, 1118, 1119, 1120  are data to be retrieved
x1101 1 4 1103 1112 dff
x1102 1 4 1104 1113 dff
x1103 1 4 1105 1114 dff
x1104 1 4 1106 1115 dff
x1105 1 4 1107 1116 dff
x1106 1 4 1108 1117 dff
x1107 1 4 1109 1118 dff
x1108 1 4 1110 1119 dff
x1109 1 4 1111 1120 dff
.ends stdata
*******************************************************************************
* STORE PARTIAL SUMS                                                           *
*******************************************************************************
.subckt stpts 1 3 1203 1204 1205 1206 1207 1208 1209 1210
* 1 is VDD, 3 is generated results valid *
```

```
* 1203, 1204, 1205, 1206 results to be stored *
* 1207, 1208, 1209, 1210 results to be retrieved to be passed to second
*                                                          set of D ff *
x1201 1 3 1203 1207 dff
x1202 1 3 1204 1208 dff
x1203 1 3 1205 1209 dff
x1204 1 3 1206 1210 dff
.ends stpts
********************************************************************************
* RETRIEVE PARTIAL SUMS                                                        *
********************************************************************************
.subckt rtpts 1 4 1303 1304 1305 1306 1307 1308 1309 1310
* 1 is VDD, 4 is input data valid *
* 1303, 1304, 1305, 1306 results to be stored *
* 1307, 1308, 1309, 1310 results to be retrieved to be passed to adders
x1301 1 4 1303 1307 dff
x1302 1 4 1304 1308 dff
x1303 1 4 1305 1309 dff
x1304 1 4 1306 1310 dff
.ends rtpts
********************************************************************************
* STORE CARRY                                                                  *
********************************************************************************
.subckt stcar 1 3 1403 1404
* 1 is VDD, 3 generated result is ready *
* 1403 carry to be stored *
* 1404 carry to be passed to second D ff *
x1401 1 3 1403 1404 dff
.ends stcar
********************************************************************************
* RETRIEVE CARRY                                                               *
********************************************************************************
.subckt rtcar 1 4 1503 1504
* 1 is VDD, 4 input data valid *
* 1503 carry to be stored *
* 1504 carry to be passed to adder *
x1501 1 4 1503 1504 dff
.ends rtcar
********************************************************************************
* INITIALIZE SUMS                                                              *
********************************************************************************
.subckt inpts 1 9999 1603 1604 1605 1606 1607 1608 1609 1610
x1601 1 9999 1603 1607 dff
x1602 1 9999 1604 1608 dff
x1603 1 9999 1605 1609 dff
x1604 1 9999 1606 1610 dff
.ends inpts
********************************************************************************
* INITIALIZE CARRY                                                             *
********************************************************************************
.subckt incar 1 9999 1703 1704
x1701 1 9999 1703 1704 dff
.ends incar
********************************************************************************
* ADAC CIRCUIT                                                                 *
********************************************************************************
xxi1 1 9999 9998 9998 9998 9998 114 115 116 117 inpts
xxi2 1 9999 9998 9998 9998 9998 800 801 802 803 inpts
xxi3 1 9999 9998 9998 9998 9998 124 125 126 127 inpts
xxi4 1 9999 9998 9998 9998 9998 804 805 806 807 inpts
xxi5 1 9999 9998 9998 9998 9998 134 135 136 137 inpts
```

```
xxi6 1 9999 9998 9998 9998 9998 808 809 810 811 inpts
xxi7 1 9999 9998 9998 9998 9998 144 145 146 147 inpts
xxi8 1 9999 9998 9998 9998 9998 812 813 814 815 inpts
xxi9 1 9999 9998 9998 9998 9998 154 155 156 157 inpts
xxi10 1 9999 9998 9998 9998 9998 816 817 818 819 inpts
xxi11 1 9999 9998 9998 9998 9998 164 165 166 167 inpts
xxi12 1 9999 9998 9998 9998 9998 820 821 822 823 inpts
xxi13 1 9999 9998 123 incar
xxi14 1 9999 9998 900 incar
xxi15 1 9999 9998 133 incar
xxi16 1 9999 9998 901 incar
xxi17 1 9999 9998 143 incar
xxi18 1 9999 9998 902 incar
xxi19 1 9999 9998 153 incar
xxi20 1 9999 9998 903 incar
xxi21 1 9999 9998 163 incar
xxi22 1 9999 9998 904 incar
xx1 1 4 5 6 7 8 9 10 11 12 13 105 106 107 108 109 110 111 112 113 stdata
xx2 1 4 800 801 802 803 114 115 116 117 rtpts
xx3 1 2 106 114 108 115 110 116 112 117 105 118 119 120 121 122 adder1
xx4 1 3 122 121 120 119 800 801 802 803 stpts
xx5 1 3 118 900 stcar
xx6 1 4 804 805 806 807 124 125 126 127 rtpts
xx7 1 4 900 123 rtcar
xx8 1 2 107 124 109 125 111 126 113 127 123 128 129 130 131 132 adder1
xx9 1 3 132 131 130 129 804 805 806 807 stpts
xx10 1 3 128 901 stcar
xx11 1 4 808 809 810 811 134 135 136 137 rtpts
xx12 1 4 901 133 rtcar
xx13 1 2 137 136 135 134 133 138 139 140 141 142 adder2
xx14 1 3 142 141 140 139 808 809 810 811 stpts
xx15 1 3 138 902 stcar
xx16 1 4 812 813 814 815 144 145 146 147 rtpts
xx17 1 4 902 143 rtcar
xx18 1 2 147 146 145 144 143 148 149 150 151 152 adder2
xx19 1 3 152 151 150 149 812 813 814 815 stpts
xx20 1 3 148 903 stcar
xx21 1 4 816 817 818 819 154 155 156 157 rtpts
xx22 1 4 903 153 rtcar
xx23 1 2 157 156 155 154 153 158 159 160 161 162 adder2
xx24 1 3 162 161 160 159 816 817 818 819 stpts
xx25 1 3 158 904 stcar
xx26 1 4 820 821 822 823 164 165 166 167 rtpts
xx27 1 4 904 163 rtcar
xx28 1 2 167 166 165 164 163 168 169 170 171 172 adder2
xx29 1 3 172 171 170 169 820 821 822 823 stpts
.tran 2ns 240ns
.print tran v(9999) v(4) v(2) v(3) v(5) v(6)
.print tran v(118) v(119) v(120) v(121) v(122)
.print tran v(128) v(129) v(130) v(131) v(132)
.print tran v(138) v(139) v(140) v(141) v(142)
.print tran v(148) v(149) v(150) v(151) v(152)
.print tran v(158) v(159) v(160) v(161) v(162)
.print tran v(168) v(169) v(170) v(171) v(172)
.end
```

APPENDIX B

As mentioned in chapter 4, digital circuits implemented in CMOS technology could display fault models other than stuck-at faults. Thus considering the stuck-at or stuck-open fault model only, is not sufficient to cover the effects of all physical errors. Understanding the effect of physical failures is essential to design tests in order to be able to detect them and tolerate them [16]. According to Mangir[17], process related failures could be due to either photolithography or process quality. Photolithography errors include oxide defects, implant defects, extra or missing contact cuts, unintended or missing connections between different parts of the device, alignment errors, ...etc. Process quality errors, on the other hand, include metallization defects, interlayer defects, contact defects, contamination, interconnect defects, ...etc.

Faults could be divided into shorts, opens, and circuit degradations[13]. Shorts are due to oxide breakdown, hot electrons, hot holes, punchthrough, or avalanche breakdown. While opens are the results of several manufacturing steps such as over-etching, poor contacts, and sharp step coverage[18]. Metallization problems, caused by electromigration or electromechanical corrosion, can produce short and open faults. Circuit degradations include threshold voltage shifts, which are caused by ionic contamination, surface charge spreading, and the trapping of hot electrons in the gate oxide. These degradations, if permanent, are translated into stuck-at faults[13].

According to Gallay et al.[19] the faults occurring were observed to be divided with the following percentages:

FAULT	%
* Short between metallizations	39
* Open metallizations	14
* Short between diffusions	14
* Open diffusion	6
* Short between metallization & substrate	2
* Inobservable	10
* Insignificant	15

Where the inobservable faults are cases were logical faults were established, without observable physical faults. On the other hand the insignificant faults were faults that presented a large imperfection (e.g. a scratch from one side to the other of the chip) and which could be easily detected by any test sequence.

On the other hand, Banerjee and Abraham[20] classified physical faults into most likely, less likely, and least likely. The following table shows the classification:

Class	Device Failure	Interconnect Failure
Most likely	* Gate to drain short * Gate to source short	* Short between diffusion lines
Less likely	* Drain contact open * Source contact open	* Aluminium polysilicon cross over broken
Least likely	* Gate to substrate short * Floating gate	* Short between aluminium lines

APPENDIX C

C.1. INTRODUCTION

In the following sections a complete analysis is given for the faults that might occur in both the 4-bit partial adder, and the full 4-bit adder. We analyze the various faults that might occur and the input patterns that will expose them, and the effect of these faults on the circuits' behaviour.

C.2. FAULT ANALYSIS IN 4-BIT PARTIAL ADDER

In the following sections a coverage of possible physical faults is presented, along with the results of simulations done to verify the output from the circuit due to the occurrence of different faults. In our fault analysis faults are considered to be single faults, and that each part of the 4-bit partial adder will be considered separately in terms of analyzing the faults and detecting them. The 4-bit partial adder will be split up into three separate parts, the carryb generation part, the inverter, and the XOR.

A study of the physical faults (shorts and opens), their effect on each part's behaviour, and their detection, is first presented. This is followed by the classical stuck-at faults.

N.B. In the simulation results, the number in parenthesis represents the voltage value in fault free case. Also, a (b) next to any variable, indicates the negation of this value (e.g. Cb1 is C1 bar).

C.3. CARRYB GENERATION BLOCK

In the carryb generation block, we'll start by studying the effect of different types of short faults in the circuit (short between gate and drain, short between

gate and source, and short between drain and source), this will be followed by the study of open faults, and finally we present the effect of classical stuck-at-faults. Fig.C.1 shows the carryb generation block with various faults.

C.3.1. SHORTS BETWEEN GATE AND DRAIN

The effect of this short and its detection is studied in a- the discharge nmos, b- any of the input nmos's, and c- the charging pmos's.

C.3.1.a. THE DISCHARGE NMOS

A short between the gate and drain would cause the clock to have intermediate values ranging from 1.18v for logical 0, to 2.18v for logical 1. When a test vector of 11111 is applied (during a low clock), we notice that respective carryb's (carry bar) are not pulled up to 1 during the charging period. Now during the evaluation period, instead of the carryb's going down to 0, a weak 1 ranging from 3.1v to 4.9v could be seen (this could be attributed to the fact that during the evaluation period the pmos's are still turned on, thus charging up carryb's at the very time they are being discharged).

clk	cin	a1	a2	a3	a4	cb1	cb2	cb3	cb4
1.17(0)	5	5	5	5	5	2.3(5)	2.9(5)	3.5(5)	4.8(5)
2.4(5)	5	5	5	5	5	3.1(0)	3.4(0)	3.8(0)	4.9(0)

C.3.1.b. ANY OF THE INPUT NMOS's

A short between gate and drain would affect the behaviour of the block depending on the input value to the gate of the faulty nmos. Testing for that fault could be seen by applying a test vector of 00000, the nmos that has a short would have a carryb equal to 0 (carryb is seen at the drain of the nmos and

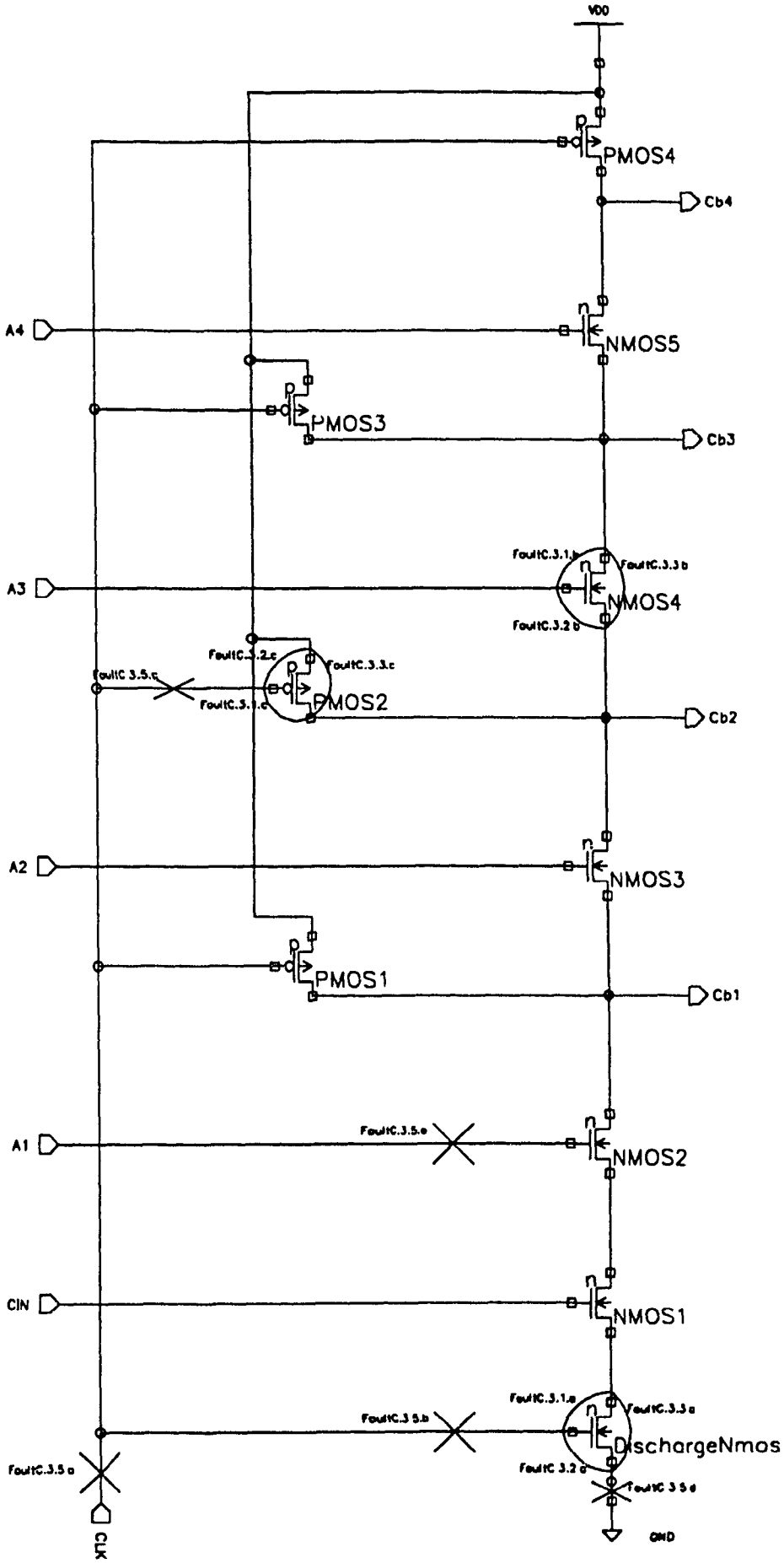


Fig.C.1 FAULTS IN CARRYb GEN. BLOCK

pmos). It could also be seen when a test vector of 11111 is applied that an intermediate value of 1 (3.27v) is seen for carryb, and all carryb's above the faulty nmos, while a soft 0 (1.08) is seen for the carryb directly below the faulty nmos (this is instead of a 0 for all carryb's). Simulation is for shorted nmos n4.

clk	cin	a1	a2	a3	a4	cb1	cb2	cb3	cb4
5	0	0	0	0	0	5	5	0(5)	5
0	5	5	5	5	5	5	5	5	5
5	5	5	5	5	5	0	1.08(0)	3.27(0)	3.27(0)

C.3.1.c. THE CHARGING PMOS's

Again as with the case of discharge nmos, a short between gate and drain would result in the clock being pulled to a changed level rather than 5v and 0v. This would result in intermediate to 0 charge of carryb during charging time, and intermediate carryb values during evaluation time. Simulation is for shorted pmos p2.

clk	cin	a1	a2	a3	a4	cb1	cb2	cb3	cb4
5	0	0	0	0	0	5	5	5	5
1.5(0)	5	5	5	5	5	1.54(5)	1.56(5)	2.27(5)	3.4(5)
2.9(5)	5	5	5	5	5	1.99(0)	2.89(0)	3.26(0)	4.07(0)
1.76(0)	5	0	5	5	5	1.89(5)	1.76(5)	2.4(5)	3.5(5)
5	5	0	5	5	5	4.2(5)	5	4.2(5)	4.2(5)
0	0	0	0	0	0	5	0(5)	5	5

It could be seen that applying a test vector of 11111 would detect all the

faults of shorts between gate and drain in discharge nmos, charging pmos's, and any of the input nmos's (even though a test of 00000 would detect which of the input nmos's is faulty). On the other hand due to the intermediate values being seen care should be taken in observing them.

C.3.2. SHORTS BETWEEN GATE AND SOURCE

A short between gate and source would cause the transistor to be in cutoff. However due to the fact that the nmos are connected serially, when the gate of the shorted nmos is subjected to a high voltage, it would charge the drain of the following nmos causing it to discharge slowly or to an intermediate value, as will be seen later.

C.3.2.a. THE DISCHARGE NMOS

A short between the gate and source in the discharge nmos would cause the transistor to be turned off no matter what phase of the clock we have. Thus causing the carryb's to be always at a value of 1. This fault could be tested by applying a test vector of 11111, which would result in a faulty 1 for all carryb's. Thus this fault could be modeled as all inputs stuck at 0.

clk	cin	a1	a2	a3	a4	cb1	cb2	cb3	cb4
0(5)	0	0	0	0	0	5	5	5	5
0	5	5	5	5	5	5	5	5	5
0(5)	5	5	5	5	5	5(0)	5(0)	5(0)	5(0)

C.3.2.b. ANY OF THE INPUT NMOS's

This fault could be tested easily by applying a 00000 vector, which would result in a faulty carryb, with a 0 value, which is connected to the source of the

faulty nmos. On the other hand, with a test vector of 11111, the carryb's above the faulty nmos will be faulty with a value of 1, while the lower carryb will discharge to a soft 0 (this is due to the gate of the faulty nmos charging the drain of the following nmos). Simulation is for nmos n4.

clk	cin	a1	a2	a3	a4	cb1	cb2	cb3	cb4
5	0	0	0	0	0	5	0(5)	5	5
0	5	5	5	5	5	5	5	5	5
5	5	5	5	5	5	1.16(0)	1.68(0)	5(0)	5(0)

C.3.2.c. THE CHARGING PMOS's

With a short between gate and source in any pmos, it would cause all the pmos's to be turned off (as the clock is always pulled to 1), while the discharge nmos will be turned on always. This fault is a very serious one as the domino principle will be non applicable, since no charging or discharging occurs. Thus a 0 output is always resulting no matter what combination of input values we're having. Simulation is for pmos p2.

clk	cin	a1	a2	a3	a4	cb1	cb2	cb3	cb4
5(0)	5	5	5	5	5	0(5)	0(5)	0(5)	0(5)
5	5	5	5	5	5	0	0	0	0
5(0)	5	0	5	5	5	0(5)	0(5)	0(5)	0(5)
5	5	0	5	5	5	0(5)	0(5)	0(5)	0(5)

C.3.3. SHORTS BETWEEN DRAIN AND SOURCE

A short between drain and source causes the transistor to be turned on

always. The effect of that fault could be serious as in the case of discharge nmos and the charging pmos's.

C.3.3.a. THE DISCHARGE NMOS

With the discharge nmos turned on always, again here the domino principle is invalid. It could be detected by monitoring the output carryb's during low clock. The resulting output would be intermediate values.

clk	cin	a1	a2	a3	a4	cb1	cb2	cb3	cb4
5	0	0	0	0	0	5	5	5	5
0	5	5	5	5	5	1.57(5)	2.39(5)	3.14(5)	4.67(5)
5	5	5	5	5	5	0	0	0	0

C.3.3.b. ANY OF THE INPUT NMOS's

Detecting a short between drain and source in any of the input nmos is feasible by applying a 0 to the faulty nmos and 1's to the rest, resulting in all 0's carryb's instead of some 1's and some 0's. Simulation is for nmos n4.

clk	cin	a1	a2	a3	a4	cb1	cb2	cb3	cb4
5	0	0	0	0	0	5	5	5	5
0	5	5	5	0	5	5	5	5	5
5	5	5	5	0	5	0	0	0(5)	0(5)

C.3.3.c. THE CHARGING PMOS's

With the faulty pmos being always turned on, it will cause the corresponding carryb and those above it to retain a high value, even though with a test

vector of 11111 they should be all 0's, while those below it will discharge to a very soft 0 (2.12v). Simulation is for pmos p2

clk	cin	a1	a2	a3	a4	cb1	cb2	cb3	cb4
5	0	0	0	0	0	5	5	5	5
0	5	5	5	5	5	5	5	5	5
5	5	5	5	5	5	2.12(0)	5(0)	5(0)	5(0)

As could be seen for most short faults, a single test vector could usually detect the occurrence of the fault, and some of these faults could even be modeled as classical stuck at faults.

C.3.4. OPEN FAULTS

Open faults, which are considered to be less or least likely to occur faults, on the other hand require two consecutive test vectors to be detected. Their effect will be studied when they occur in the clock line, input to gates, and ground connection.

C.3.4.a. OPEN IN THE CLOCK LINE

An open fault in the clock line (controlling the gates of the discharge nmos, and the charging pmos), would cause the discharge nmos to be turned off, while the charging pmos will be turned on always, resulting in a constant output of 1.

clk	cin	a1	a2	a3	a4	cb1	cb2	cb3	cb4
5	0	0	0	0	0	5	5	5	5
0	5	5	5	5	5	5	5	5	5
5	5	5	5	5	5	5(0)	5(0)	5(0)	5(0)

C.3.4.b. OPEN IN DISCHARGE NMOS GATE

With a floating gate of the discharge nmos, the transistor will act as if it is in the cutoff, thus resulting in constant outputs of 1.

clk	cin	a1	a2	a3	a4	cb1	cb2	cb3	cb4
5	0	0	0	0	0	5	5	5	5
0	5	5	5	5	5	5	5	5	5
5	5	5	5	5	5	5(0)	5(0)	5(0)	5(0)

C.3.4.c. OPEN IN ANY CHARGING PMOS GATE

Any pmos having a floating gate would act as if the gate is subjected to a 0 input (results of simulation showed a gate voltage ranging from 0v to 2.2v, but this is however enough to keep the pmos turned on). The following will show the result of applying two test vectors, a 00000 vector, and a 11111 vector. It will be noticed that the output is somehow correct, but it does have changed levels (i.e. instead of getting a 0 result with test vector 11111, we get a changed level 0).

clk	cln	a1	a2	a3	a4	cb1	cb2	cb3	cb4
5	0	0	0	0	0	5	5	5	5
0	5	5	5	5	5	5	5	5	5
5	5	5	5	5	5	0	1.3(0)	1.3(0)	1.3(0)

C.3.4.d. OPEN IN GND CONNECTION

An open in the ground connection would cause the output not to discharge, whenever there is a path to ground.

clk	cln	a1	a2	a3	a4	cb1	cb2	cb3	cb4
5	0	0	0	0	0	5	5	5	5
0	5	5	5	5	5	5	5	5	5
5	5	5	5	5	5	5(0)	5(0)	5(0)	5(0)

C.3.4.e. OPEN IN THE GATE CONNECTION OF ANY INPUT NMOS

A floating gate in one of the input nmos would again make the transistor behave as if it were turned off. The effect of this fault could be observed on the carryb's above the faulty nmos, since they never discharge. However, for carryb's below the faulty nmos behave normally. The following table shows the effect of a floating gate to the nmos connected to input a2(n3). It is seen that c2b, c3b, and c4b do not discharge even with a 11111 test vector, however c1b behaves normally.

clk	cln	a1	a2	a3	a4	cb1	cb2	cb3	cb4
5	0	0	0	0	0	5	5	5	5
0	5	5	5	5	5	5	5	5	5
5	5	5	5	5	5	0	5(0)	5(0)	5(0)

It could be seen that some of the open faults could be modeled as stuck at faults, however, it is sometimes difficult to model the rest of them due to the changed voltage levels they display.

C.3.5. STUCK AT FAULTS

For classical stuck at faults, these are detected easily for inputs stuck at 1 (0) by applying test vectors 1's and 0 (0's and 1) , 0(1) to the faulty nmos, to check for the fault.

C.4. INVERTER BLOCK

The same sequence of analysis will be carried for the inverter as for the carry generation block. Fig.C.2 shows the inverter block with various faults.

C.4.1. SHORTS BETWEEN GATE AND DRAIN

A short between gate and drain in either the nmos or the pmos would cause the output either to a have a changed value but correct output (i.e. soft 1 and soft 0) or to follow the input (i.e. no inversion), depending on whether the short impedance is large or very small respectively. Simulation results shown are for small impedance value.

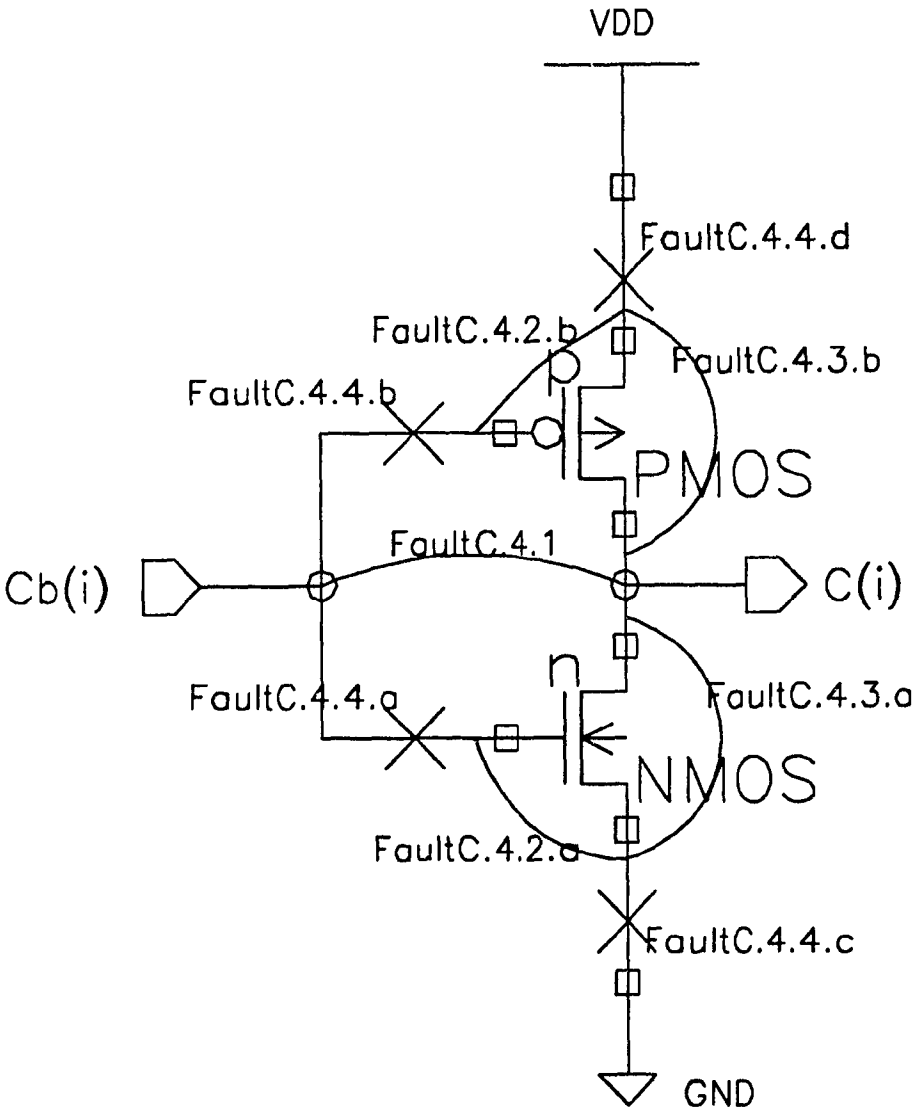


Fig.C.2 FAULTS IN INVERTER BLOCK

cb	c
5	5(0)
0	0(5)

C.4.2. SHORTS BETWEEN GATE AND SOURCE

A short occurring between gate and source causes the transistor to be in the cutoff region.

C.4.2.a. FOR NMOS

With the gate of the nmos shorted to the source, the nmos is turned off, while the input to the inverter will also be pulled down to 0 -regardless of the actual input- (simulation was carried with voltages driven from two fault free inverters), causing the pmos to be always turned on and the output remaining at 1.

cb	c
0(5)	5(0)
0	5

C.4.2.b. FOR PMOS

Again the short would cause the pmos to be turned off, but here the input would be always pulled up to 5v, thus turning on the nmos and keeping the output always at 0.

cb	c
5	0
5(0)	0(5)

It could be seen that the short between gate and source could be modeled as either the output stuck at 1 or the input stuck at 0 for faulty nmos, while it could be modeled as either the output stuck at 0 or the input stuck at 1 for faulty pmos.

C.4.3. SHORTS BETWEEN DRAIN AND SOURCE

Short between drain and source causes the transistor to be always turned on, thus keeping the output constantly either at 0 or at 1 depending on whether the short was in the nmos or the pmos respectively.

C.4.3.a. FOR NMOS

cb	c
5	0
0	0(5)

C.4.3.b. FOR PMOS

cb	c
5	5(0)
0	5

Again this fault could be modeled as either the output stuck at 0 or the input stuck at 1 for faulty nmos, and modeled as either the output stuck at 1 or the input stuck at 0 for faulty pmos.

C.4.4. OPEN FAULTS

Open faults are studied for opens in input to either the nmos or pmos gates, open in the vdd connection, and open in the gnd connection.

C.4.4.a. OPEN IN NMOS GATE CONNECTION

Due to the open in gate connection, nmos behaves as if input gate voltage is 0, thus keeping the output to be always charged to 1.

cb	c
5	5(0)
0	5

C.4.4.b. OPEN IN PMOS GATE CONNECTION

With a floating gate of the pmos, when a 5v is applied to the inverter, instead of having a 0, a soft 0 results, due to the fault which keeps the pmos turned on, thus charging the output when it is being discharged.

cb	c
5	1.6(0)
0	5
5	1.6(0)

C.4.4.c. OPEN IN GND CONNECTION

An open in the ground connection would keep the output from the inverter from discharging, thus keeping it fixed at 1.

cb	c
5	5(0)
0	5

C.4.4.d. OPEN IN VDD CONNECTION

An open in the VDD connection would cause the output never to be charged up, thus keeping it always at 0.

cb	c
5	0
0	0(5)

C.4.5. STUCK AT FAULTS

It could be seen that stuck at faults could be detected directly by testing for them or by being implied through other faults which give the same erroneous output.

C.5. STATIC XOR

The analysis of faults in the XOR is rather long, since we have to see the effect of each fault and how it could be detected in each transistor of six transistors composing the XOR. Fig.C.3 shows the static XOR with faults.

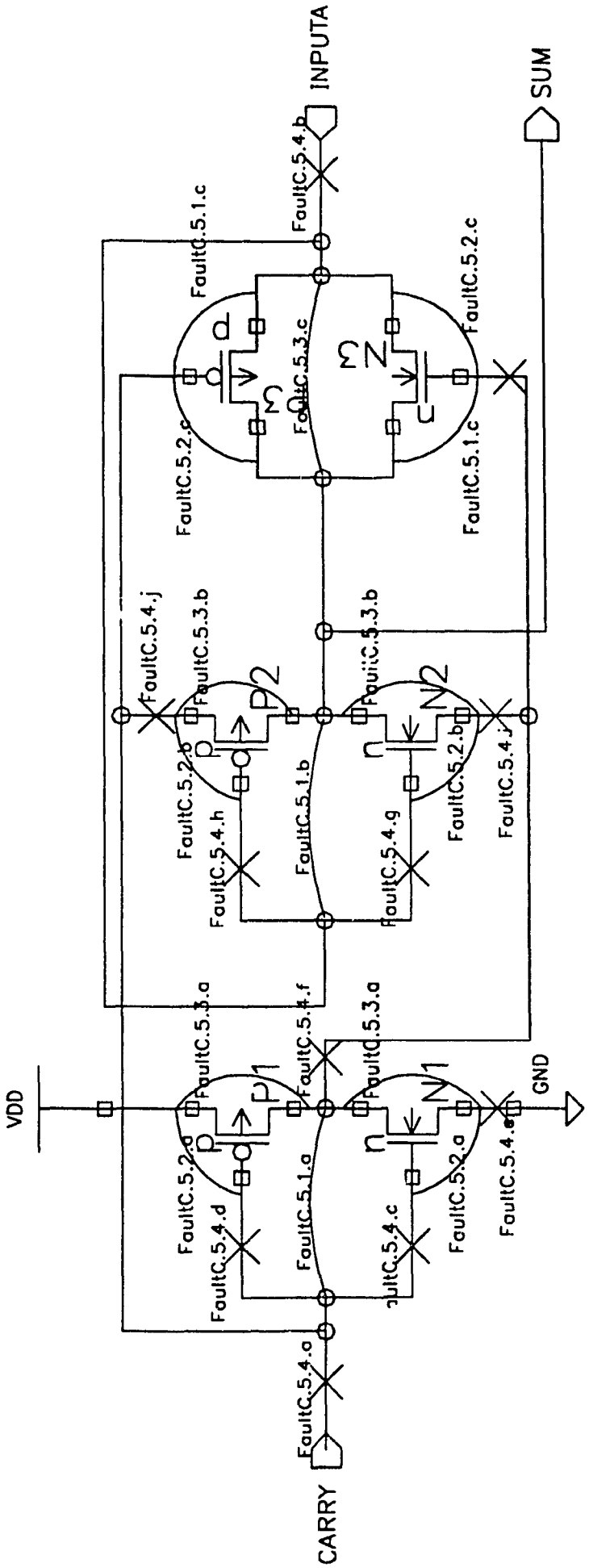


Fig.C.3 FAULTS IN XOR BLOCK

C.5.1. SHORTS BETWEEN GATE AND DRAIN

C.5.1.a. FOR NMOS & PMOS PAIR no. 1

We know from previous analysis that a short between gate and drain causes the inverter to fail from inverting its input, thus to test for this fault applying test vector 11 would result in an output of a changed value 1 (4.29v). Due to the fault, n2 is turned off, while n3 is turned on. With a equal to 1, p2 is also turned off, thus leaving n3 to pass 1, resulting in a changed value 1. The fault could be further seen by a 01 test vector resulting in a soft 0 (1.5v).

c	a	o/p	comments
5	5	4.2(0)	faulty o/p with changed level
0	5	1.5(5)	faulty o/p with changed level
5	0	0(5)	faulty o/p
0	0	0	fault free

C.5.1.b. FOR NMOS & PMOS PAIR no. 2

Again with the knowledge that due to that short the inverter would fail to invert, we apply a test vector that would inhibit the function of the transmission gate. Applying c equal to 1 and any value to a, the output would follow a, i.e. with a 10 vector we get 0, and with 11 vector we get 1.

c	a	o/p	comments
5	5	5(0)	faulty o/p
0	5	5	fault free
5	0	0(5)	faulty o/p
0	0	0	fault free

C.5.1.c. FOR NMOS & PMOS PAIR no. 3

The effect of the short and detecting it would be studied for each transistor separately.

C.5.1.c.i. FOR NMOS

Due to the short between the gate and drain, changed levels of the applied a will result. With c equal to 1, the transmission gate would be disabled, however due to the short, a will be pulled to a changed level (instead of the applied 5v, a becomes 1.24v since the gate voltage of n3 is 0), thus turning p2 on and n2 off, resulting in a 1 instead of a 0.

c	a	o/p	comments
5	1.2(5)	5(0)	faulty o/p & changed (a) level
0	5	5	fault free
5	0	5	fault free
0	1.5(0)	1.5(0)	faulty o/p & changed (a) level

C.5.1.c.ii. FOR PMOS

With that short occurring in p3, a will be pulled to the value of c. Thus,

with c equal to 0 and a equal to 1, a will be pulled to 0, resulting in a 0 passing through the transmission gate.

c	a	o/p	comments
5	5	0	fault free
0	0(5)	0(5)	faulty o/p & changed (a) value
5	5(0)	0(5)	faulty o/p & changed (a) value
0	0	0	fault free

C.5.2. SHORTS BETWEEN GATE AND SOURCE

C.5.2.a. FOR NMOS & PMOS PAIR no. 1

With short occurring in nmos, we know that c will be always pulled to 0, thus resulting in an output of 1 from first inverter, and disabling the second inverter. That makes our output to be always equal to the values of a.

c	a	o/p	comments
0(5)	5	5(0)	faulty o/p & changed (c) value
0	5	5	fault free
0(5)	5	0(5)	faulty o/p & changed (c) value
0	0	0	fault free

On the other hand, a short occurring in the pmos would cause c to be always 1, and the output ,from the first inverter, to be a 0. Thus the transmission gate will always be disabled, and the output from the XOR will be equal to ab (a bar).

c	a	o/p	comments
5	5	0	fault free
5(0)	5	0(5)	faulty o/p & changed (c) value
5	0	5	fault free
5(0)	0	5(0)	faulty o/p & changed (c) value

We can see that a short between the gate and source in the first inverter transistors could be modeled as c stuck at 0 for faulty nmos, and as c stuck at 1 for faulty pmos.

C.5.2.b. FOR NMOS & PMOS PAIR no. 2

The short between gate and source in nmos would cause n2 to be turned off. To see the effect of the fault we disable transmission gate by applying c equal to 1. Applying a equal to 1, we notice that a is pulled down to 1.24v due to the short in n2, thus enabling p2 and resulting in an output of 1.

c	a	o/p	comments
5	1.24(5)	5(0)	faulty o/p & changed (a) level
0	5	5	fault free
5	0	5	fault free
0	1.5(0)	1.5(0)	faulty o/p & changed (a) level

On the other hand, if the short was in p2, p2 will be turned off, while both c and a will have the same value. Thus with a equal to 1, and c equal to 0, c will be pulled to 1 thus disabling the transmission gate, and resulting in a 0 instead of 1.

c	a	o/p	comments
5	5	0	fault free
5(0)	5	0(5)	faulty o/p & changed (c) value
0(5)	0	0(5)	faulty o/p & changed (c) value
0	0	0	fault free

C.5.2.c. FOR NMOS & PMOS PAIR no. 3

The effect of this short in n3 can be seen by applying c equal to 1, which in fault free case disable the transmission gate. Applying a equal to 0 would enable the second inverter, and again in fault free case result in a 1. However, due to the short, the output is pulled to 0.

c	a	o/p	comments
5	5	0	fault free
0	5	5	fault free
5	0	0(5)	faulty o/p
0	0	2.7(0)	faulty o/p with changed level

While the short in p3 could be seen by applying c equal to 0, in order to enable transmission gate, and applying a equal to 1. The output will be a 0 instead of a 1 (passing through p3) due to the short.

c	a	o/p	comments
5	5	5(0)	faulty o/p
0	5	0(5)	faulty o/p
5	0	5	fault free
0	0	0	fault free

C.5.3. SHORTS BETWEEN DRAIN AND SOURCE

C.5.3.a. FOR NMOS & PMOS PAIR no. 1

A short between drain and source would cause transistor to be turned on regardless of what gate voltage is applied. To test for this fault in n1, applying c equal to 0 should result in a 1 from the inverter, however as 0 results, p2 and n3 will be turned off, while n2 could turn on depending on gate voltage. With a equal to 1, we should get a 1 (passing through p3), however as n2 is turned on, it will discharge the XOR output, resulting in a soft 0 (1.51v).

c	a	o/p	comments
5	5	0	fault free
0	5	1.5(5)	faulty o/p with changed level
5	0	5	fault free
0	0	0	fault free

If the short was in p1, we know that the first inverter will always give an output of 1, thus disabling n2. Applying c equal to 1, would only disable p3, but n3 is turned on due to fault. With a equal to 1, p2 will be disabled, leaving n3 to pass a changed value 1.

c	a	o/p	comments
5	5	4.2(0)	faulty o/p with changed level
0	5	5	fault free
5	0	0(5)	faulty o/p
0	0	0	fault free

C.5.3.b. FOR NMOS & PMOS PAIR no. 2

Testing for this fault in either n2 or p2, will be by disabling the transmission gate, i.e. by applying 1 to c.

To test for the fault in n2, we apply a equal to 0. Instead of getting a 1, we only get 0.

c	a	o/p	comments
5	5	0	fault free
0	5	5	fault free
5	0	0(5)	faulty o/p
0	0	2.7(0)	faulty o/p with changed level

For the pmos, on the other hand, we apply a equal to 1. Again instead of getting a 0, we get 1.

c	a	o/p	comments
5	5	5(0)	faulty o/p
0	5	0(5)	faulty o/p
5	0	5	fault free
0	0	0	fault free

C.5.3.c. FOR NMOS & PMOS PAIR no. 3

In fault free case, transmission gate should be turned off when c is equal to 1. However, due to fault, it is always turned on. Thus with a equal to 1, instead of getting a 0 as a result of the second inverter inverting a, our output is charged up to 1 due to the fault.

c	a	o/p	comments
5	5	5(0)	faulty o/p
0	5	5	fault free
5	0	0(5)	faulty o/p
0	0	0	fault free

C.5.4. OPEN FAULTS

In the following section the effect of open faults are studied, together with simulations to detect them.

C.5.4.a. OPEN FAULT IN LINE C

With an open fault in the carry line, the output follows the value of input a, i.e. fault could be seen as a stuck at 0 fault of c.

c	a	o/p	comments
5	5	5(0)	faulty o/p
0	5	5	fault free
5	0	0(5)	faulty o/p
0	0	0	fault free

C.5.4.b. OPEN FAULT IN LINE A

An open fault in the a line, results in a voltage of 1.8 to 2.1v supplied to both the transmission gate and the second inverter. This results in a faulty 5v output when a 11 test vector is applied, otherwise the voltage ranges from 1.06v to 1.86v for other test vectors.

c	a	o/p	comments
5	5	5(0)	faulty o/p
0	5	1.86(5)	faulty o/p with changed level
5	0	1.07(5)	faulty o/p with changed level
0	0	1.45(0)	faulty o/p with changed level

C.5.4.c. OPEN IN N1 GATE CONNECTION

A floating gate of n1 could be seen whenever an input of 5v is applied to c. With the output of the first inverter kept at 5v, the output from the xor will therefore be faulty.

c	a	o/p	comments
5	5	4.5(0)	faulty o/p with changed level
0	5	5	fault free
5	0	0(5)	faulty o/p
0	0	0	fault free

C.5.4.d. OPEN IN P1 GATE CONNECTION

With a floating gate of p1, the output of the first inverter is kept floating between 2v and 5v. Hence again affecting the output of xor whenever c is 5v.

a	c	o/p	comments
5	5	2.1(0)	o/p with changed level
0	5	5	fault free
5	0	4.2(5)	o/p with changed level
0	0	0	fault free

C.5.4.e. OPEN IN GND CONNECTION

Open in gnd connection would disable first inverter from discharging, thus turning on n3 all the time. Again the effect of this fault could be seen whenever c is 5v.

c	a	o/p	comments
5	5	4.5(0)	faulty o/p with changed level
0	5	5	fault free
5	0	0(5)	faulty o/p
0	0	0	fault free

C.5.4.f. OPEN IN OUTPUT OF 1st INVERTER

With an open fault in output line of first inverter, the output is noticed to follow the input a.

c	a	o/p	comments
5	5	5(0)	faulty o/p
0	5	5	fault free
5	0	0(5)	faulty o/p
0	0	0	fault free

C.5.4.g. OPEN IN N2 GATE CONNECTION

Due to floating gate, n2 acts as if it is turned off. It's effect could be seen by disabling transmission gate, by applying c equal to 5v.

c	a	o/p	comments
5	0	5	fault free
5	5	5(0)	faulty o/p
5	0	5	fault free

C.5.4.h. OPEN IN P2 GATE CONNECTION

Again by disabling transmission gate, we can observe the fault.

c	a	o/p	comments
5	0	5	fault free
5	5	2(0)	o/p with changed level
5	0	5	fault free

C.5.4.i. OPEN IN N2 SOURCE CONNECTION

With the output of the second inverter kept constant at 5v, the fault could be seen whenever c is 5v.

c	a	o/p	comments
5	0	5	fault free
5	5	5(0)	faulty o/p
5	0	5	fault free

C.5.4.j. OPEN IN P2 SOURCE CONNECTION

Again disabling the transmission gate the fault could be seen.

c	a	o/p	comments
5	5	0	fault free
5	0	0(5)	faulty o/p

C.5.4.k. OPEN IN N3 GATE CONNECTION

Due to floating gate of n3, charge is trapped on this gate, causing the nmos to be always turned on. Applying 5v to c should in fault free case disable transmission gate, however as n3 is turned on, and with the application of 0v to a, the output is pulled down to 0 instead of being 5v. On the other hand with a being 5v, and c being 5v, the output is charged to a soft 1, instead of its being pulled down to 0v.

c	a	o/p	comments
5	0	0(5)	faulty o/p
0	5	5	fault free
5	5	3.6(0)	faulty o/p with changed level

C.5.5. STUCK AT FAULTS

For classical stuck at faults in inputs, with a stuck at 1 the output would be equal to cb (carry bar), while a stuck at 0 would result in c. Moreover with c stuck at 1 we get an output equal to ab (a bar), while a stuck at 0 c would result in a.

C.6. FAULT ANALYSIS OF FULL 4-BIT ADDER

In the following sections we'll continue with the analysis of the full 4-bit adder. In these sections we'll only analyze faults in the carry generation block (Gen:Car), and the propagate & generate signals generation block (Gen:P&G). As the static XOR used for sum calculation and in the Gen:P&G block is the same static XOR used in the 4-bit partial adder, we will not repeat the analysis. Also, for some of the transistors, like the discharge nmos and the charging pmos in both the Gen:P&G and the Gen:Car, the analysis for some of the faults will

not be repeated as it is the same analysis as in the carry generation block of the 4bit partial adder.

C.7. GEN:P&G BLOCK

In the following section the analysis of the faults that might occur in the Gen:P&G block is given. We'll be looking at faults that might occur in the small totem pole and the inverter. Fig.C.4 shows the Gen:P&G block with various faults.

C.7.1. SHORTS BETWEEN GATE AND DRAIN

C.7.1.a. NMOS1

A short between gate and drain in nmos1 causes the evaluated node to have intermediate value with a 11 vector (3.5v) instead of 0, thus causing the output to be a faulty 0. On the other hand, with a 00 or 01 vector, the evaluated node is pulled down to 0, thus causing the output to be a faulty 1(also during low clock, the node is not fully charged up..4.3v). Finally, with a 10 vector, no effect of the fault is seen.

clk	al	bl	gl
5	0	0	5(0)
0	5	5	0
5	5	5	0(5)
0	1.9(0)	5	0
5	0	5	5(0)

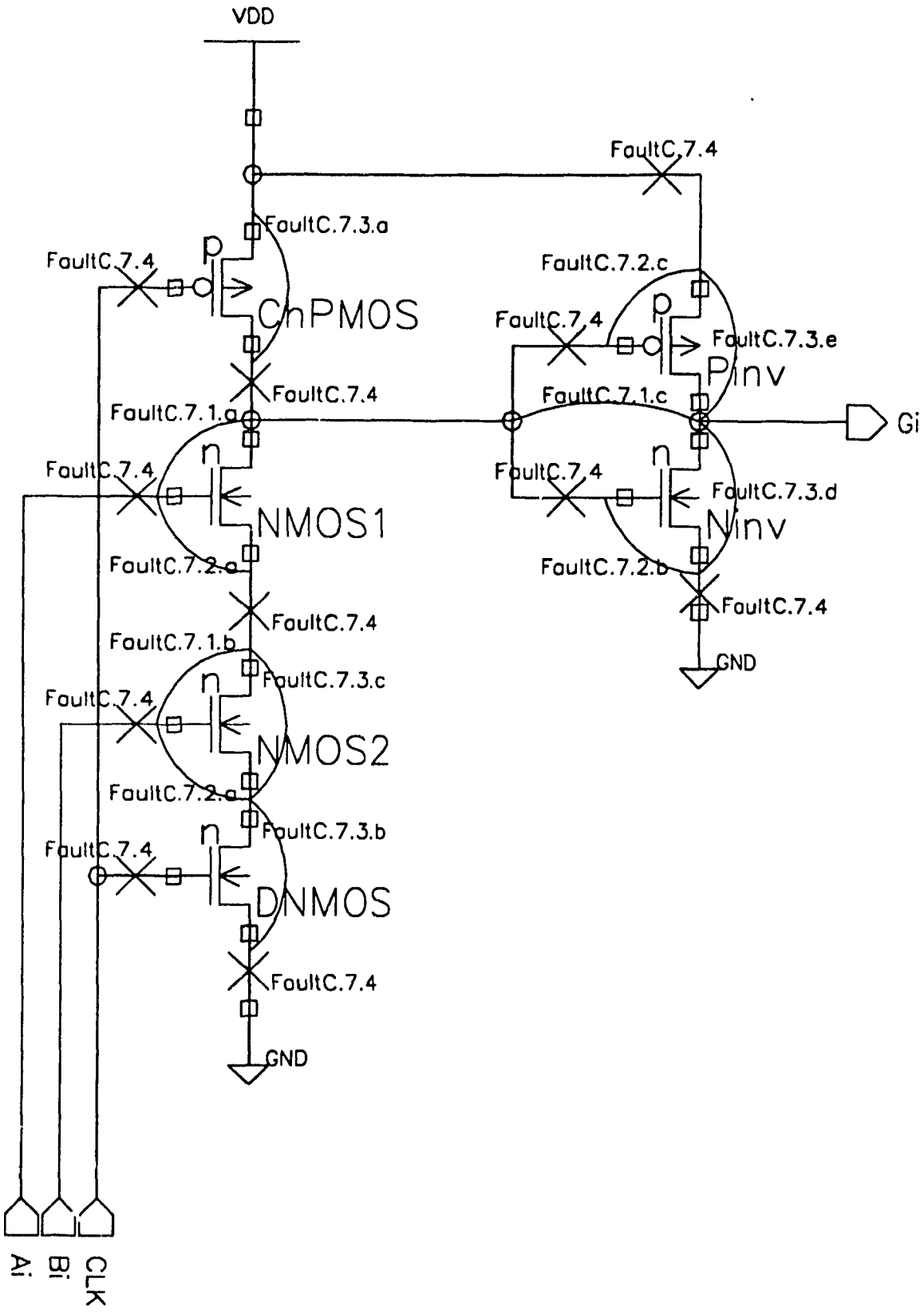


Fig.C.4 FAULTS IN GEN:P&G BLOCK

C.7.1.b. NMOS2

A short between gate and drain in nmos2 causes again intermediate value to be seen at the evaluated node with a 11 vector (2.8v) instead of 0, thus causing the output to be a faulty 0. With a 10 vector, the evaluated node is pulled down to a 0 thus causing the output to be a faulty 1(also during low clock the node is not fully charged up..3.3v). Finally, both 00 and 01 vectors do not cause the fault to show.

clk	a1	b1	g1
5	5	5	0(5)
0	5	1.2(0)	0
5	5	0	5(0)

C.7.1.c. INVERTER NMOS & PMOS

A short between gate and drain in either the inverter pmos or nmos, will cause the output of the inverter and thus the output of the Gen:P&G block to have intermediate values ranging from 1.1v to 3.6v. With a 11 vector the output becomes 1.1v as opposed to a 5v that should have resulted. With 00, 01, or 10 the output is a 2.2v which should have been a 0. Finally during low clock the output is never fully charged up (only 3.5v).

clk	a1	b1	g1
5	0	0	2.2(0)
0	5	5	3.5(0)
5	5	5	1.1(5)
0	0	5	3.5(0)
5	0	5	2.2(0)
0	5	0	3.5(0)
5	5	0	2.2(0)

C.7.2. SHORTS BETWEEN GATE AND SOURCE

C.7.2.a. NMOS1 OR NMOS2

A short between gate and source in either nmos1 or nmos2 will cause either nmos to be always turned off, thus disabling the totem pole from discharging, and thus keeping the output of the Gen:P&G block to be always kept at 0.

clk	a1	b1	g1
5	0	0	0
0	5	5	0
5	5	5	0(5)

C.7.2.b. INVERTER NMOS

A short between gate and source in inverter nmos will keep the nmos turned off, thus keeping the output always at 1.

clk	a1	b1	g1
5	0	0	5(0)
0	0	5	5(0)
5	0	5	5(0)

C.7.2.c. INVERTER PMOS

A short between gate and source of Inverter pmos will cause the output of the Gen:P&G block to be always at 0.

clk	a1	b1	g1
0	5	5	0
5	5	5	0(5)

C.7.3. SHORTS BETWEEN DRAIN AND SOURCE

C.7.3.a. CHARGING PMOS

A short between drain and source will cause the charging pmos of the totem pole to be always turned on, thus keeping the node charged up and causing the output of the Gen:P&G to be always kept at 0.

clk	a1	b1	g1
0	5	5	0
5	5	5	0(5)

C.7.3.b. DISCHARGE NMOS

A short between drain and source will cause the discharge nmos to be always turned on. This fault could be seen during low clock, and with both Al and Bl equal to 1, where the node does not charge up full (only to 2.8v). This fault, however, has no effect during evaluation period, but it fails the domino principle.

clk	al	bl	glb
0	5	5	2.8(5)

C.7.3.c. NMOS1 OR NMOS2

A short in any of these nmos's will keep the faulty transistor always turned on. This fault could be seen by applying a 1 to the non faulty nmos and a 0 to the faulty one (i.e. test vector 10, or 01). The output of Gen:P&G goes to 1 instead of 0. Simulation is for faulty nmos2.

clk	al	bl	gl
5	5	0	5(0)

C.7.3.d. INVERTER NMOS

With the inverter nmos always turned on, the output of Gen:P&G will always be 0.

clk	al	bl	gl
5	5	5	5(0)

C.7.3.e. INVERTER PMOS

Finally, with the inverter pmos turned on always because of the fault, the output of Gen:P&G is kept at 1.

clk	a1	b1	g1
5	0	0	0(5)

C.7.4. OPEN FAULTS

In this section we'll discuss the effect of open faults. This will include open connections to transistor's gate, source, and drain. An open drain connection of the charging pmos, or an open connection to the ground in the inverter nmos, or an open connection in the inverter nmos gate, will cause the circuit to behave as if the output was stuck-at-1.

Meanwhile, an open connection to the ground of the discharge nmos, or an open connection in the gate of the either the discharge nmos or any of the totem pole nmos's, or an open in the source of any of the totem pole nmos's, or an open in the drain of nmos2, or an open in the VDD connection in the inverter pmos, will cause the output to behave as if it is stuck-at-0.

Finally, an open in the inverter pmos gate connection, or an open in the drain of nmos1, or an open in the gate connection of the charging pmos, will cause the output to have an intermediate value.

C.8. GEN:CAR BLOCK

We'll start the fault analysis by covering first the different short faults, followed by the open faults. Fig.C.5 shows the Gen:Car block with various faults.

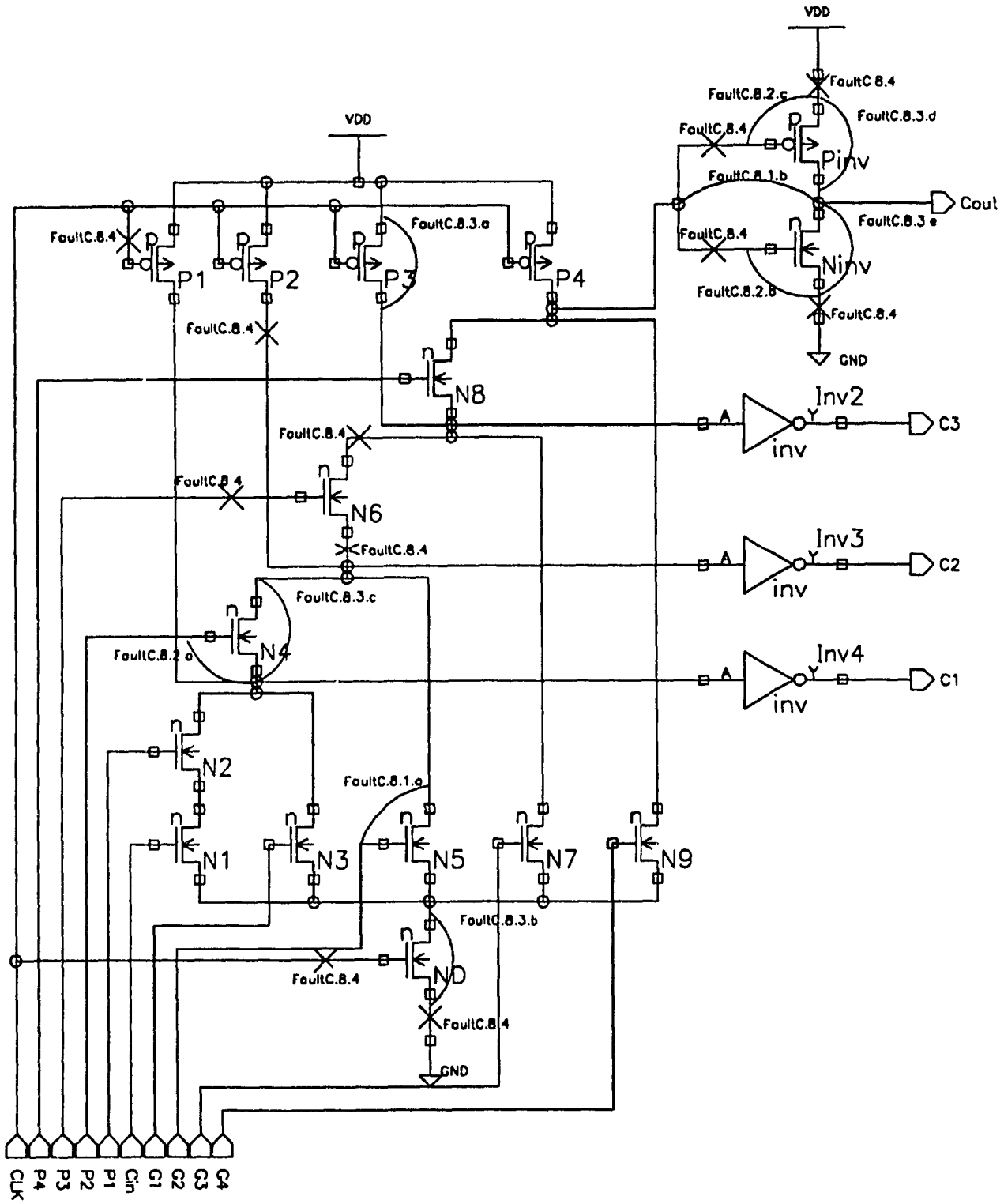


Fig.C.5 FAULTS IN GEN:CAR BLOCK

C.8.1. SHORTS BETWEEN GATE AND DRAIN

C.8.1.a. ANY OF THE NMOS's

A short between gate and drain in any of the totem pole nmos will cause the nodes above the faulty nmos to discharge to an intermediate value, thus causing the corresponding carry's to have a faulty intermediate value. Also the node directly above the faulty nmos will not charge up (actually it charges up to 1.1v) during low clock when the gate of that nmos is subjected to a 0 input. The nodes below the faulty nmos (and hence the corresponding carry's) will not be affected by this fault. Simulation is for fault in nmos with g2 as input.

clk	cin	p1,3,4	g2	g1,3,4&p2	c1	c2	c3	c4
5	0	0	0	0	0	5(0)	0	0
0	5	5	5	0	0	0	0	0
5	5	5	5	0	5	3.6(5)	3.7(5)	3.8(5)
0	0	0	0	0	0	5(0)	0	0

C.8.1.b. INVERTER NMOS & PMOS

A short between gate and drain of either nmos or pmos of inverter will cause the resulting carry to have an intermediate value ranging from 1.1v to 2.2v (instead of 5v and 0 respectively) during high clock, and 3.2v during low clock.

clk	cin	pi	gi	c1	c2	c3	c4
5	0	0	0	2.2(0)	2.2(0)	2.2(0)	2.2(0)
0	5	5	0	3.2(0)	3.2(0)	3.2(0)	3.2(0)
5	5	5	0	1.1(5)	1.4(5)	1.6(5)	1.9(5)

C.8.2. SHORTS BETWEEN GATE AND SOURCE

C.8.2.a. ANY OF THE NMOS's

A short between gate and source in any nmos in the totem pole (excluding the discharge nmos) will cause this nmos to be turned off, thus keeping the carry's above this faulty nmos to be always kept at 0, while those below it will be unaffected by this fault. Simulation is for faulty nmos with p2 input.

clk	cin	p1	g1	c1	c2	c3	c4
0	5	5	0	0	0	0	0
5	5	5	0	5	0(5)	0(5)	0(5)

C.8.2.b. INVERTER NMOS

A short between gate and source in the inverter nmos will cause the nmos to be turned off, thus keeping the output always at 1.

clk	cin	p1	g1	c1	c2	c3	c4
5	0	0	0	5(0)	5(0)	5(0)	5(0)
0	5	5	0	5(0)	5(0)	5(0)	5(0)

C.8.2.c. INVERTER PMOS

A short between gate and source in the inverter pmos will cause the pmos to be turned off, thus preventing the output from being charged and keeping the output always at 0.

clk	cln	pl	gl	c1	c2	c3	c4
0	5	5	0	0	0	0	0
5	5	5	0	0(5)	0(5)	0(5)	0(5)

C.8.3. SHORTS BETWEEN DRAIN AND SOURCE

C.8.3.a. ANY CHARGING PMOS

A short between drain and source in any of the charging pmos of the totem pole will keep the pmos always turned on, thus keeping the nodes above the faulty pmos, and the node connected to it from discharging, therefore, keeping the carry's of these nodes always at 0. However, the nodes below the faulty pmos will discharge to an intermediate value (2.15v), thus causing the corresponding carry's to have an intermediate value of 3.5v instead of 5v. Simulation for faulty pmos3.

clk	cln	pl	gl	c1	c2	c3	c4
0	5	5	0	0	0	0	0
5	5	5	0	3.5(5)	0(5)	0(5)	0(5)

C.8.3.b. THE DISCHARGE NMOS

The effect of this fault as before could only be seen during low clock. However, despite the fact that this fault will not affect the circuit behavior during evaluation period, the domino effect will not be valid.

clk	cin	pl	gl	c1b	c2b	c3b	c4b
0	5	5	0	1.5(5)	2.3(5)	3(5)	3.6(5)

C.8.3.c. ANY OF THE NMOS's

With a 0 applied to the gate to the faulty nmos, this transistor should be turned off, however, due to the short between drain and source the nmos will be turned on, thus discharging the nodes above the faulty nmos and the carry's corresponding to these nodes will give a faulty 1. Meanwhile, this fault will not affect the carry's below the faulty nmos. Simulation is for faulty nmos of p2 input.

clk	cin	p1,3,4	gl&p2	c1	c2	c3	c4
0	5	5	0	0	0	0	0
5	5	5	0	5	5(0)	5(0)	5(0)

C.8.3.d. INVERTER PMOS

A short between drain and source in the inverter pmos will cause the output of the carry resulting from that inverter to be always 1.

C.8.3.e. INVERTER NMOS

On the other hand, the short between drain and source in the inverter nmos will cause the carry resulting from that inverter to be always 0.

C.8.4. OPEN FAULTS

Open faults that will be covered here include open connections to transistors'

gate, source, and drain. An open in the drain connection of any of the charging pmos in the totem pole, or an open connection to the ground in the inverter nmos, or an open connection to the inverter nmos gate will cause the output of the carry generation circuit to behave as if the output is stuck-at-1.

While an open connection to the ground of the discharge nmos, or an open connection in the gate of the discharge nmos, or an open connection in any of the totem pole nmos's gate, or an open in the source of any of the nmos in the totem pole (this regards the carryb's above the faulty connection), or an open to the VDD connection in the inverter pmos, will cause the output of the block to behave as if the output is stuck-at-0.

Finally, an open in the inverter pmos gate connection, or an open in the drain (or source) connection of the totem pole nmos (this regards the carry directly connected to the faulty nmos), or an open in the gate connection of any of the charging pmos, will cause the output to have an intermediate value.