Copyright © 1985, by the author(s).
All rights reserved.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission.

SOME DESIGN TECHNIQUES FOR HIGH-PERFORMANCE MOS CIRCUITS

by

Shing Ip Kong

Memorandum No. UCB/ERL M85/10

22 February 1985

(cover)

SOME DESIGN TECHNIQUES FOR HIGH-PERFORMANCE MOS CIRCUITS

by

Shing Ip Kong

Memorandum No. UCB/ERL M85/10

22 February 1985

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

Research sponsored by DARPA - NESC under Contract N00039-85-R-0269.

SOME DESIGN TECHNIQUES FOR HIGH-PERFORMANCE MOS CIRCUITS

Shing Ip Kong

Department of Electrical Engineering & Computer Sciences
University of California
Berkeley, California

ABSTRACT

Several design techniques for metal oxide semi-conductor MOS circuit are described in this report. Some of these techniques are bootstrap drivers in NMOS, CMOS dynamic circuits, and control logic using a finite state machine. The detailed design of a 32-bit ALU is presented in this report as a design example.

January 23, 1985

1. INTRODUCTION

In recent years, economies of scale have made very large scale integration, VLSI, one of the most popular way to implement large digital systems. Among all the semiconductor technologies available, metal oxide semiconductor, MOS, is one of the technologies most suitable for VLSI. Because of its relatively simple layout, MOS circuits can have very high circuit density. Furthermore, MOS circuits are also easier to scale down and can gain more in performance than bipolar circuits as feature size is scaled down. This report describes several design techniques for metal oxide semiconductor, MOS, circuit with emphasis on VLSI system that uses a 5V power supply.

This report is organized into six chapters. The design of a bootstrap driver is described in Chapter 2. NMOS is the technology considered here because bootstrap drivers are mostly used in NMOS to minimize static power consumption. Furthermore most of the author's experience in bootstrap drivers was learned from fine tuning the bootstrap drivers used in NMOS SOAR [Ung84]. In Chapter 3, Domino and NORA logic, two of the most promising CMOS dynamic circuit design styles are described. The charge sharing problem, which is common to all dynamic circuits, is also described in this chapter. Chapter 4 is a CMOS design example in which the details design of a 32-bit ALU is shown. Chapter 5 shows how the control logic in a VLSI system can be implemented using a finite state machine. The first part of this chapter is devoted to the basic principles of the finite state machine. The goal here is to introduce all the basic principles one needs to understand the rest of the chapter such that even a circuit designer has no previous experience with finite state machines can understand the materials presented in the rest of chapter 6.

It must be pointed out that this report is not intended to be a complete design manual for MOS circuit. The author's intention is to summarize some of his experience in MOS circuit design such that circuit designers can use this report as a guide in certain aspects of MOS circuit design. The information covered in this report are gathered by the author during his first year of research in the EECS department of the University of California, Berkeley.

2. NMOS BOOTSTRAP DRIVER

2.1 Introduction

In NMOS circuits, a bootstrap driver is usually used when a large capacitive load has to be driven with minimum delay and minimum static power consumption. Besides using bootstrap driver, whose schematic is shown in Figure_2-2b, a large capacitive load can also be driven by a super buffer [M&C80], or a low power push-pull driver. The schematic of a super buffer is shown in Figure_2-1a, and Figure_2-1b shows the schematic of a low power push-pull driver.

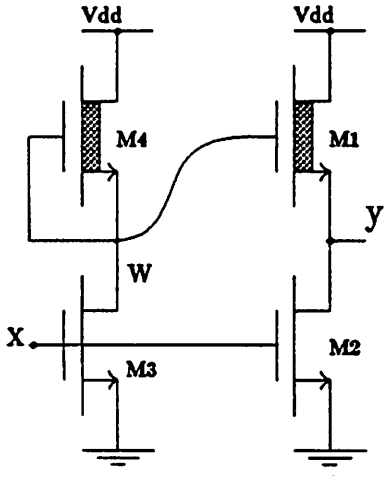


Figure 2-1a Super Buffer

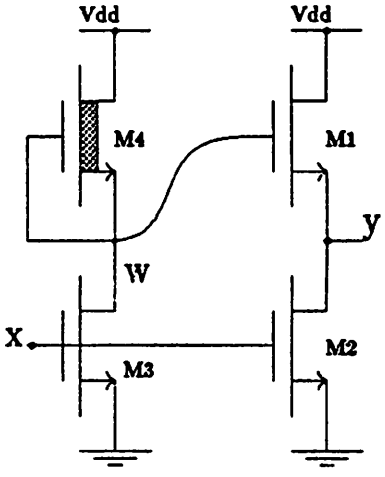


Figure 2-1b Push-Pull Driver

In order for the super buffer shown in Figure_2-1a to drive the output node (node y) to a high voltage level (V_{dd}) rapidly, (W/L) of M1 has to be large. Furthermore, to keep the low voltage level within the noise margin requirement, (W/L) of M2 has to be k times bigger than M1. A general rule is to use k=4 if node x can be driven to V_{dd} and k=8 if node x is driven high through a pass transistor and therefore cannot rise to V_{dd} due to threshold loss [M&C80]. The large (W/L)'s of M1 and M2 implies that the effective resistance between V_{dd} and GND is relatively small when both of these devices are on (the output node is at the low voltage level). This small effective resistance results in large static current and a high power consumption results.

In Figure_2-1b, the push-pull driver minimized its static power consumption by using an enhancement mode transistor (M1) as the pull up device. The static power consumption of this driver is small because only one of the two big output transistors (M1 and M2) can be on at any time. Furthermore, no ratio is required for the (W/L)'s of M1 and M2. However using an enhancement mode pull up does introduce the penalty of a high voltage level lower than V_{dd} at the output node. The voltage at the output node V(y) is given by:

$$V(y) = V_{dd} - V_{t1}\Big[V(y)\Big]$$

where

$V_{t1}\Big[V(y)\Big]$ is the threshold voltage of transistor M1 when its source voltage equals to V(y)

The voltage degradation problem mentioned above can be eliminated by using an extra power supply V_{gg} as shown in Figure_2-2a. The desired value of V_{gg} is:

$$V_{gg} \geq V_{dd} + V_{t1}(V_{dd})$$

where

$V_{t1}(V_{dd})$ is the threshold voltage of M1 when its source voltage equals V_{dd}

When input (node x) is low, the gate of M1 (node w) will rise to V_{gg}. If V_{gg} mets the requirement stated above, then the output node (node y) can rise to V_{dd}.

Figure_2-2b is a bootstrap driver which operates on the same idea as Figure_2-2a except that the extra power supply V_{gg} is eliminated. To achieve a voltage level higher than V_{dd} at the gate of M1 (node w), node p is bootstraped. When input (node x) is high, output node (node y) is at GND level and through proper (W/L) ratios of M3, M4, and M5, voltage at node p should be approximately 3V. The voltage across the bootstrap capacitor C_{boot} is therefore approximately 3V. When input switches to low, the voltage across the bootstrap capacitor cannot change instantaneously. As the output node (node y) rises towards V_{dd}, node p is bootstraped towards a voltage higher than V_{dd}. Ideally node p is bootstraped to $V_{dd} + 3V$ but due to charge sharing with

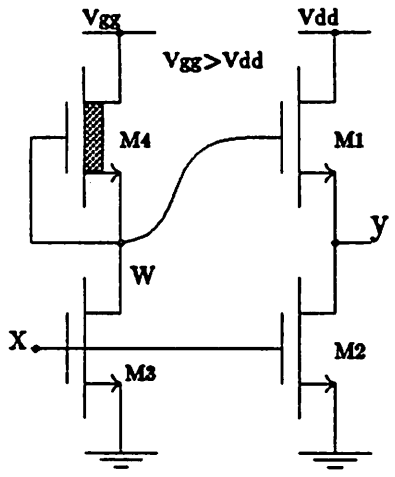


Figure 2-2a Extra Power Supply

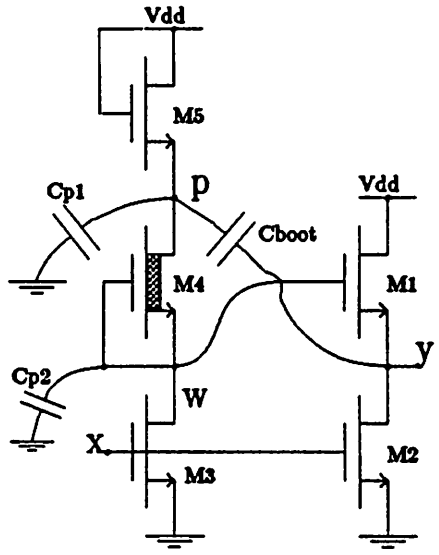


Figure 2-2b Bootstrap Driver

the gate capacitor of M1 C_{g1}, the parasitic capacitors C_{p1}, and C_{p2}, node p can only rise to a lower than ideal voltage. The charge sharing problem can be controlled by proper sizing of the bootstrap capacitor C_{boot}.

2.2 Size Of The Bootstrap Capacitor

2.2.1 Factor Determines The Size Of Bootstrap Capacitor

The bootstrap capacitor C_{boot} must be big enough such that the bootstrap node (node p in Figure_2-2b) can be bootstraped to V_x despite of charge sharing.

$$V_x > V_{dd} + V_{t1}(V_{dd}) \qquad 2.1$$

Inequality 2.1 can be satisfied if the following rule is followed:

$$C_{boot} >> \left[C_{g1} + C_{p1} + C_{p2}\right]$$

where

$V_{t1}(V_{dd}) =$ *threshold voltage of* $M1$ *when its source voltage equals* V_{dd}

$$= V_{tn0} + \gamma\left\{\left[V_{dd}+2\left|\phi_f\right|\right]^{1/2} - \left[2\left|\phi_f\right|\right]^{1/2}\right\} \qquad 2.2$$

$C_{g1} =$ *gate capacitance of* $M1$

$$= W_1 \times L_1 \times \left[\frac{\epsilon_{ox}}{t_{ox}}\right] \qquad 2.3$$

$C_{p1} =$ *parasitic capacitance of node p (see Figure_2–2b)*

$C_{p2} =$ *parasitic capacitance of node w (see Figure_2–2b)*

In Equation 2.2, the first term V_{tn0} is the threshold voltage when the body-source junction of the transistor is zero biased while the second term takes into account the increase in threshold voltage due to body effect [H&J83]. In Equation 2.3, W_1 and L_1 are the width and length of the transistor respectively. Furthermore, ϵ_{ox} is the dielectric constant of silicon dioxide and and t_{ox} is the silicon dioxide thickness [H&J83].

Parasitic capacitors C_{p1} and C_{p2} depend strongly on the layout and are therefore very hard to estimate without the final layout. Unfortunately, the final layout won't be available until design is done. However, in a good design, the parasitic capacitors should be much smaller than the gate capacitor C_{g1}. Using the assumption $C_{g1} >> C_{p1}, C_{p2}$, one possible strategy is to pick

$$C_{boot} = N \times C_{g1} \qquad 2.4$$

where N can be found using the following simple charge sharing analysis.

2.2.2 Simple Charge Sharing Analysis

This charge sharing analysis is based on the simplified model shown in Figure_2-3. Let the charge inside the control surface in Figure_2-3a be Q_{t0}, and the charge inside the control surface in Figure_2-3b be Q_{t2}. Q_{t0} and Q_{t2} are thus the charge inside the control surface before and after the input has switched low.

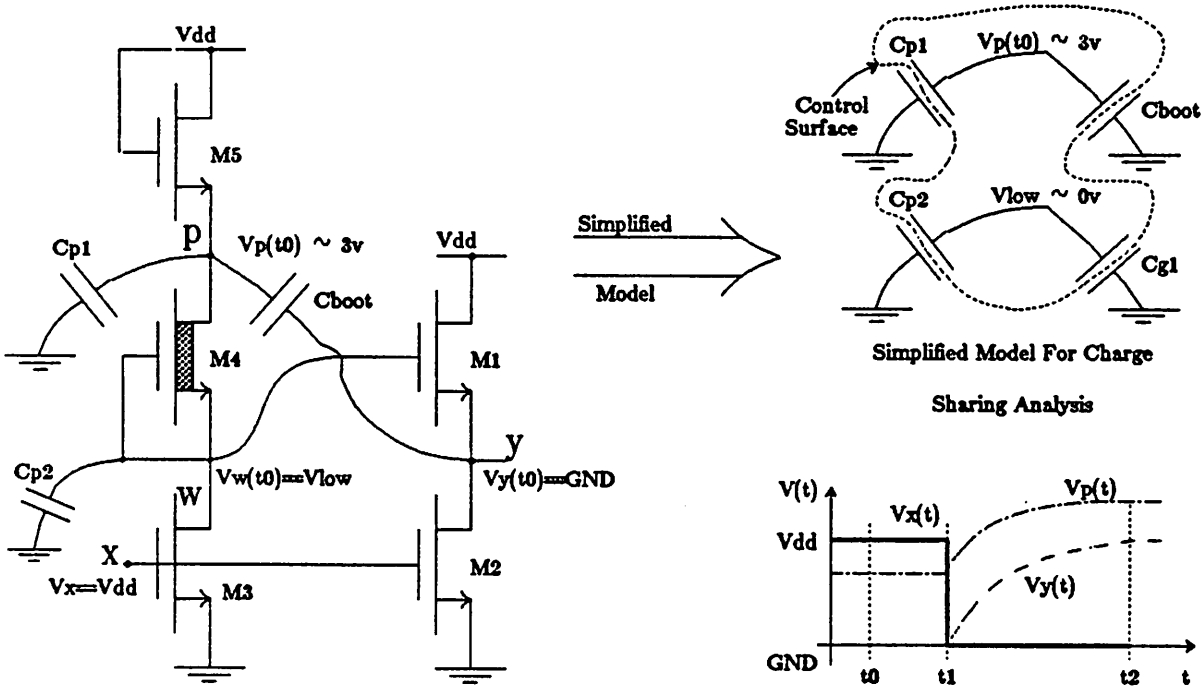


Figure 2-3a Bootstrap Driver With Input Vx(t0)=Vdd

$$Q_{t0} = V_p(t0) \times \left[C_{boot}+C_{p1}\right] + V_{low} \times \left[C_{p2}+C_{g1}\right]$$

$$Q_{t2} = \left[V_p(t2)-V_{dd}\right] \times C_{boot} + V_p(t2) \times \left[C_{p1}+C_{p2}+C_{g1}\right]$$

Since both node p and w are isolated from V_{dd} and GND after input switched from high to low, the initial charge Q_{t0} should equal to the final charge Q_{t2} (assuming leakage current can be neglected).

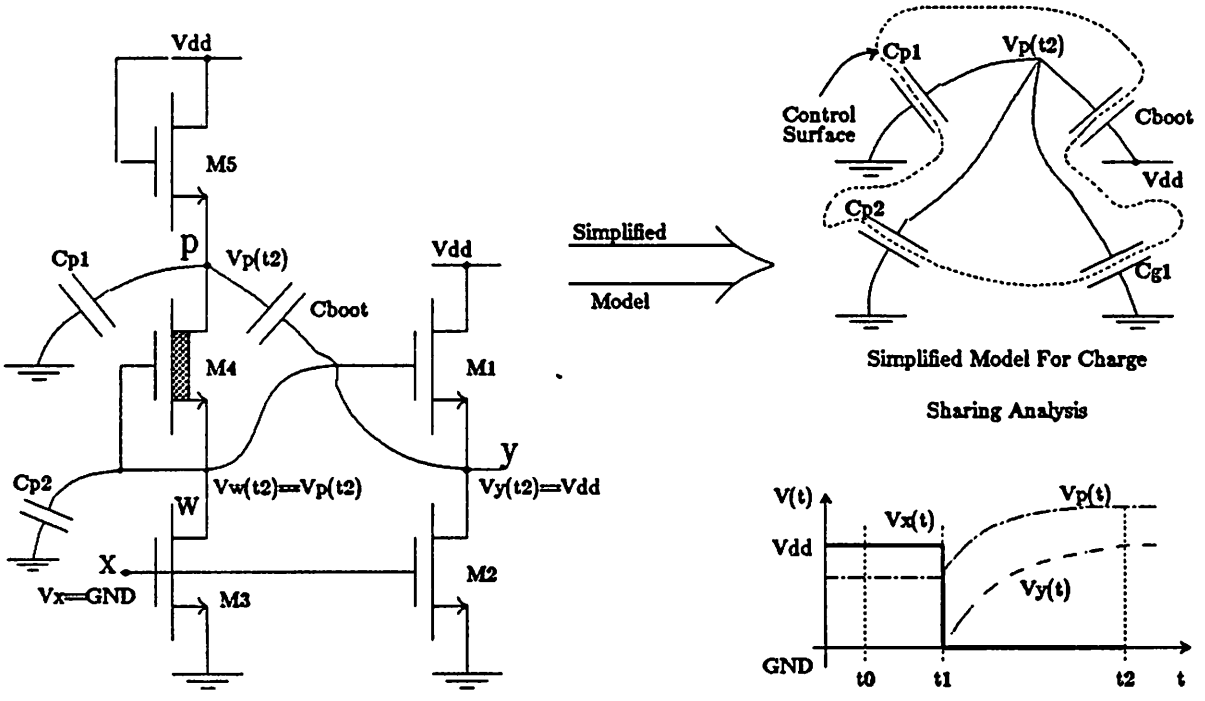


Figure 2-3b Bootstrap Driver With Input Vx(t2)=GND

$$Q_{t0} = Q_{t2}$$

implies

$$V_p(t0) \times \left[C_{boot}+C_{p1}\right] + V_{low} \times \left[C_{p2}+C_{g1}\right]$$

$$= \left[V_p(t2)-V_{dd}\right] \times C_{boot} + V_p(t2) \times \left[C_{p1}+C_{p2}+C_{g1}\right]$$

Solve for $V_p(t2)$

$$V_p(t2) = \frac{\left[V_p(t0) \times \left[C_{boot}+C_{p1}\right] + V_{low} \times \left[C_{p2}+C_{g1}\right] + V_{dd} \times C_{boot}\right]}{\left[C_{p1} + C_{p2} + C_{g1} + C_{boot}\right]} \qquad 2.5$$

V_{low} has to be smaller than the threshold voltage of transistor M1 and therefore can be assumed to be zero in Equation 2.5. This is a conservative assumption because it makes $V_p(t2)$ appear to have a smaller value. In other words, it makes charge sharing look more severe.

Ignoring the term $V_{low} \times \left[C_{p2}+C_{g1}\right]$ Equation 2.5 becomes:

$$V_p(t2) = \frac{V_p(t0) \times \left[C_{boot}+C_{p1}\right] + V_{dd} \times C_{boot}}{C_{p1} + C_{p2} + C_{g1} + C_{boot}} \qquad 2.6$$

Substitute Equation 2.4 into Equation 2.6:

$$V_p(t2) = \frac{V_p(t0) \times \left[N \times C_{g1}+C_{p1}\right] + V_{dd} \times N \times C_{g1}}{C_{p1} + C_{p2} + (N+1) \times C_{g1}} \qquad 2.7$$

As stated at the end of Section 2.2.1, both C_{p1} and C_{p2} are hard to estimate but likely to be much smaller than the gate capacitance of M1 (C_{g1}). Based on this, it is safe to assume

$$(N+1) \times C_{g1} >> \left[C_{p1}+C_{p2}\right]$$

and

$$N \times C_{g1} >> C_{p1}$$

Using these two assumptions, Equation 2.7 is reduced to:

$$V_p(t2) = \frac{N \times \left[V_p(t0) + V_{dd}\right]}{(N+1)} \qquad 2.8$$

As discussed in Section 2.1:

$$V_p(t0) \approx 3V$$

In digital circuit,

$$V_{dd} = 5V$$

To ensure that the output reaches V_{dd}=5V, it is necessary to have:

$$V_p(t2) \geq 6.5V$$

Using these values, Equation 2.8 can be solved for N:

$$6.5V \times (N+1) \leq N \times (5V+3V)$$

$$N \geq \frac{6.5}{1.5}$$

$$N > 4.33$$

To account for the optimistic assumptions made when going from Equation 2.7 to 2.8 and have some safety margin (this safety margin will be used when an extra depletion mode pull up is added as shown in Section 2-5), it is recommended that:

$$C_{boot} = N \times C_{g1}$$

where

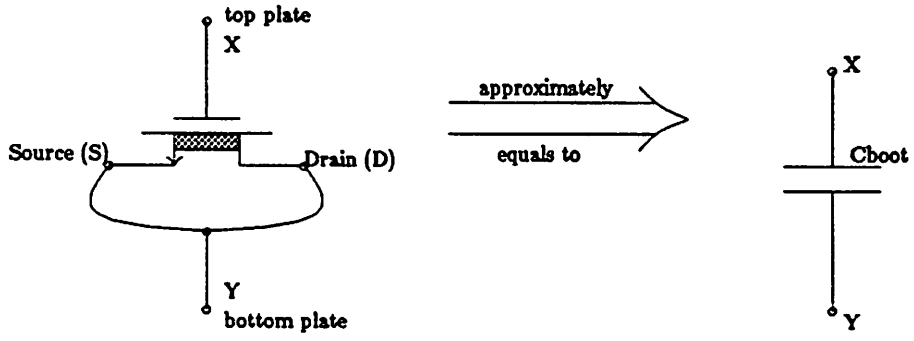
$$N \geq 6$$

It is shown in Section 2.3 that C_{boot} is the gate capacitor of a special transistor whose drain and source are connected. N=6 then simply implies that the gate area of this special transistor must have a gate area 6 times as big as the gate area of transistor M1.

2.3 Construction Of The Bootstrap Capacitor

2.3.1 Basic Structure Of The Bootstrap Capacitor

In a single layer polysilicon, single layer metal NMOS process, the gate capacitance of a depletion mode transistor structure is used as the bootstrap capacitor C_{boot}. This is shown in Figure_2-4. Notice that the source (S) and drain (D) of the depletion mode transistor are connected together.

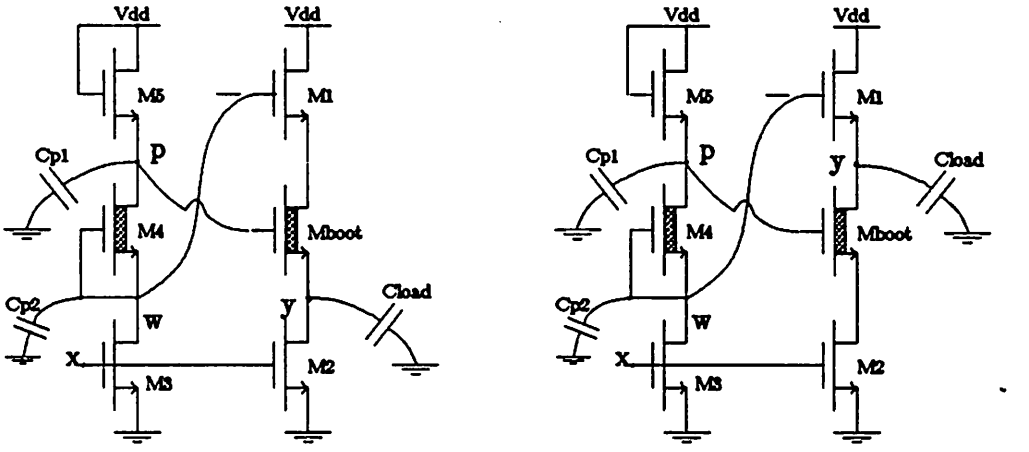


(a) Actual Capacitor (b) Theoretical Capacitor

Figure 2-4 Basic Structure Of The Bootstrap Capacitor

The reason for such a connection is illustrated in Figure_2-5. As a result of not connecting the source and drain of the depletion mode transistor M_{boot} together, pull up transistor M1 is in series with transistor M_{boot} in Figure_2-5a. The effective resistance of a depletion mode transistor is high and since M1 has to pull up the load through this transistor, the rise time of the bootstrap driver is severely degraded.

The above problem cannot be solved by connecting the load to the other end of the depletion mode transistor M_{boot} as shown in Figure_2-5b. Although this configuration does improve the rise time of the bootstrap driver, it also has an obvious side effect. The pull down transistor M2 is now in series with transistor M_{boot} and the fall time of the driver is now severely degraded.



(a) Rise Time is Degraded (b) Fall Time is Degraded

Figure 2-5 Bootstrap Drivers With Degraded Rise and Fall Times

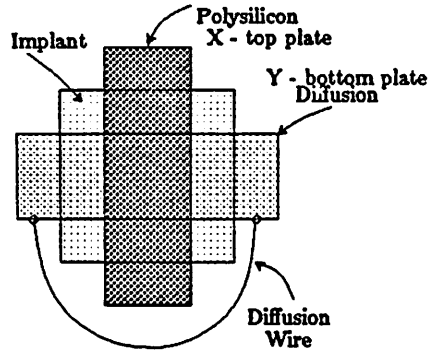
2.3.2 Deviation From Ideal Capacitor

One major difference between the real bootstrap capacitor (Figure_2-4a) and the ideal bootstrap capacitor (Figure_2-4b) is that the two terminals X and Y are NOT interchangeable for the real bootstrap capacitor. For this reason, the two terminals should be identified as the "top plate" and the "bottom plate" as shown in Figure_2-4a. This difference is a direct result of how the bootstrap capacitor is constructed and can be understood easily by looking at the layout.

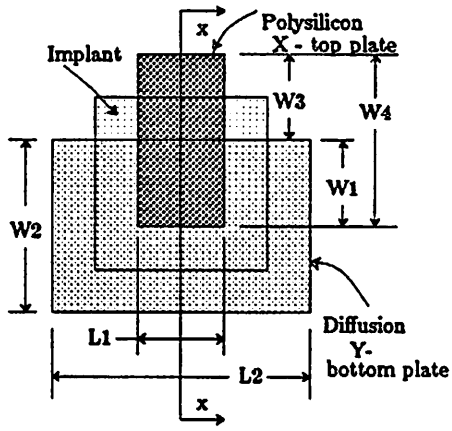
The layout of a bootstrap capacitor is shown in Figure_2-6b and Figure_2-6c is its cross sectional view. Notice that a diffusion wire is used here to connect the source and drain of the depletion mode transistor. A metal wire connection will be more ideal but it will use up much more area because two diffusion-metal contacts are needed. This area penalty is severe for a small driver. Furthermore for small driver, the connection is short and the resistance and capacitance of the diffusion wire is acceptable.

The equivalent circuit of this bootstrap capacitor is shown Figure_2-6d. In this Figure:

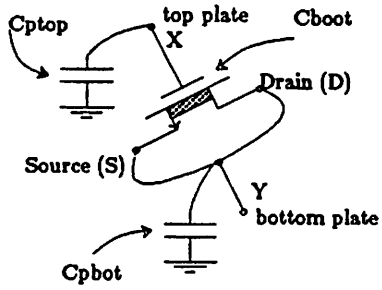
$$C_{boot} = W_1 \times L_1 \times \left[\frac{\epsilon_{ox}}{t_{ox}}\right]$$



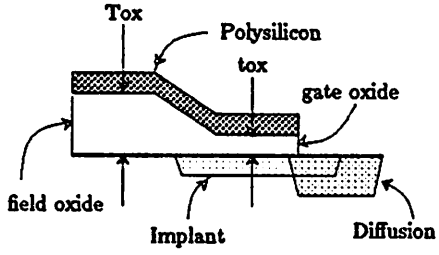
(a) Conceptual View



(b) Layout - Top View



(d) Equivalent Circuit



(c) Layout - Side View x-x

Figure 2-6 Layout Of The Bootstrap Capacitor

C_{ptop} = *parasitic capacitance between the top plate* and *the substrate*

$$= W_s \times L_1 \times \left[\frac{\epsilon_{ox}}{T_{ox}}\right]$$

C_{pbot} = *parasitic capacitance between the bottom plate* and *the substrate*

$$= W_2 \times L_2 \times C_j + \left[2\times(W_2+L_2) - L_1\right] \times C_{jsw}$$

where

C_j = *diffusion junction capacitance per unit area*

C_{jsw} = *diffusion sidewall capacitance per unit perimeter*

Using parameters from any typical NMOS process and a reasonable set of W's and L's, the above equations imply $C_{boot} >> C_{pbot} >> C_{ptop}$. Since $C_{pbot} >> C_{ptop}$, the bottom plate, which is in diffusion, should always be connected to the output node. On the other hand, the top plate, which is in polysilicon, should always be connected to the bootstrap node (node p in Figure_2-2b). There are two reasons for such a connection:

(1) The large bottom plate parasitic capacitor C_{pbot} is driven by large output transistors M1 and M2.

(2) Charge sharing problem at node p (see Figure_2-3) is minimized because only the small top plate parasitic capacitor C_{ptop} is making contribution to the total parasitic capacitance at node p C_{p1}.

2.3.3 Problems With Design Tools

The layout of this capacitor has to obey a different set of design rules from those for an ordinary depletion mode transistor. A special layer, dcap, is introduced in the layout system Magic [Ous84] to specify the area where polysilicon overlaps diffusion and is used as a capacitor instead of a transistor. Physically, this layer is the same as the dfet layer which specifies a depletion mode transistor. A different layer name is introduced because it enables the design rule checker within Magic to check the design rules differently.

Circuit extractor Mextra [M&O&S83] does not extract the polysilicon to substrate capacitance correctly. From Figure_2-6b and c, it is obvious that the poly to substrate capacitance

should be:

$$C_{poly/sub} = (W_4 - W_1) \times L_1 \times \left[\frac{\epsilon_{ox}}{T_{ox}}\right]$$

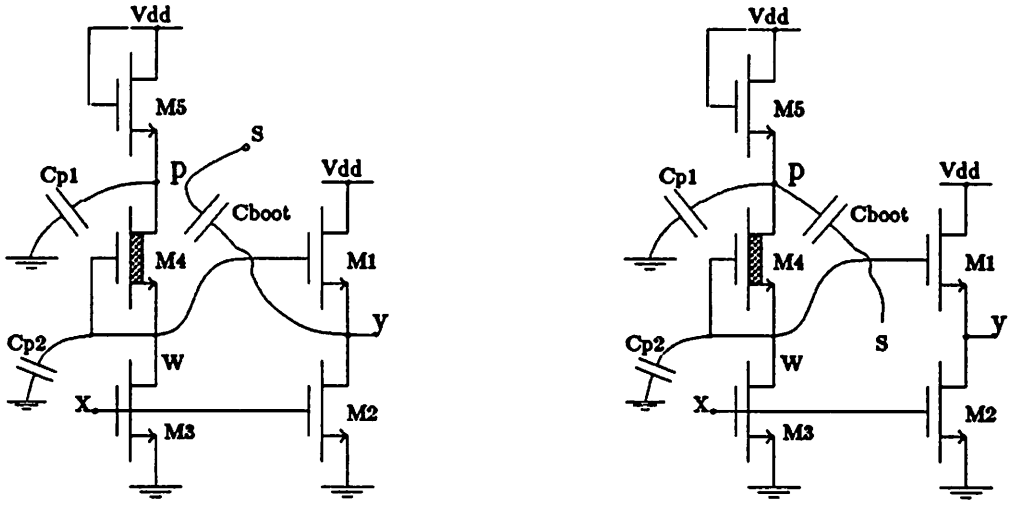
Instead of what Mextra gives:

$$C_{poly/sub} = W_4 \times L_1 \times \left[\frac{\epsilon_{ox}}{T_{ox}}\right]$$

In other words, Mextra does not subtract the gate area when it calculates the poly to substrate capacitance. This over estimation only has a small effect on ordinary circuit because the gate area is usually small and the field oxide thickness T_{ox} is much thicker than the gate oxide thickness t_{ox}.

This is not the case in a bootstrap driver because the "gate area" (poly overlap diffusion area) of the bootstrap capacitor is big. As a result, Mextra grossly over-estimates the parasitic capacitance at node p (C_{p1} in Figure_2-3) and thus make charge sharing look much worse.

Sim2spice [M&O&S83] assumes all capacitors are connected between a circuit node and substrate. As a result, even though the .sim file (output file of Mextra, circuit description in ESIM [M&O&S83] format) has the correct connectivity, when Sim2spice converts the .sim file to .spice file (output of Sim2spice, circuit description in spice format), one end of the bootstrap capacitor is always connected to the substrate. The resulting circuit either looks like Figure_2-7a or Figure_2-7b. Fortunately when Sim2spice is doing the conversion, it creates a .name file which maps all the node names in the .sim file to all the node names in the .spice file. Using this information and knowing the fact that all connection is correct in the .sim file, one can easily edit the .spice file to obtain a correct circuit description in Spice format.



(a) Top plate connects to substrate (b) Bottom plate connects to substrate

Figure 2-7 Erroneous Circuits Created By Design Tool Sim2spice

2.4 Design Decisions Concerning Transistor Size

The basic bootstrap driver shown in Figure 2-8a consists of five transistors. In Sections 2.4.1 and 2.4.2, simple methods are introduced to estimate their sizes by hand calculation. However the sizes of all these transistors, especially the two output transistors M1 and M2, should be refined further by Spice simulation.

2.4.1 Sizes of The Supporting Transistors M3 M4 and M5

The sizes of transistors M3, M4, and M5 (Figure_2-8a) can be determined by the voltage level requirement at node p and w when input is high (see Section 2.1 and 2.1). This is illustrated in Figure 2-8b.

In Figure 2-8b, voltage at node p is approximately 3V. The voltage at node w has to be smaller than V_{te}, the threshold voltage of an enhancement transistor, to ensure transistor M1 is

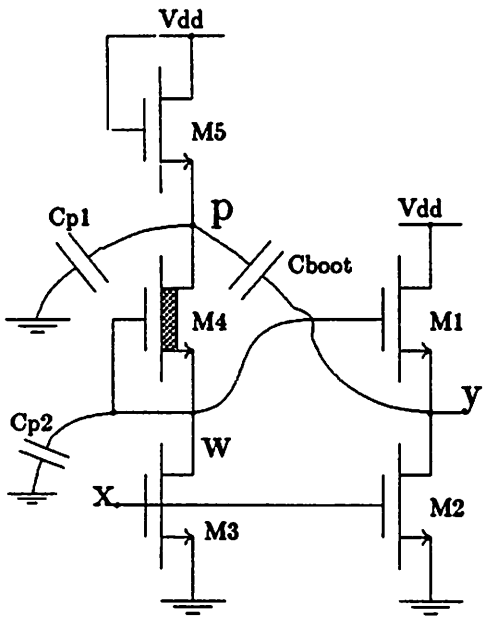


Figure 2-8a Bootstrap Driver

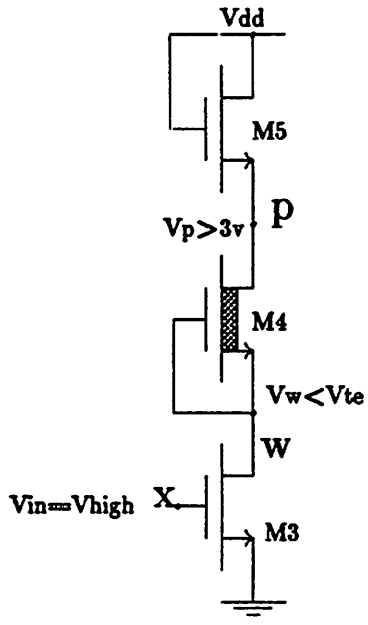


Figure 2-8b Simple Circuit Determines Sizes Of M1 M2 M3

off. Under these conditions, transistors M3 is resistive. M4 are likely to be resistive because $V_{gd4} = V_{te} - 3v$ is likely to be greater than V_{td} (threshold voltage of a depletion mode transistor).

Using Kirchoff's current law:

$$I_{ds3}(res) = I_{d4}(res) \qquad 2.9$$

The drain current of a transistor in the resistive region is given by [H&J83]:

$$I_d(res) = \frac{k'}{2} \times \frac{W}{L} \times \left[2(V_{gs} - V_t)V_{ds} - V_{ds}^2\right] \qquad 2.10$$

where

k' = *transconductance parameter*

V_t = *threshold voltage of the transistor*

V_{gs} = *gate source voltage*

V_{ds} = *drain source voltage*

Using Equation 2.10 and the voltages shown in Figure_2-8b, Equation 2.9 becomes:

$$\frac{W_3}{L_3} \times \left[2(V_{high}-V_{te})V_{te}-V_{te}^2\right] > \frac{W_4}{L_4} \times \left[2(-V_{td})(3v-V_{te})-(3v-V_{te})^2\right]$$

Solve for $\frac{W_4}{L_4}$

$$\frac{W_4}{L_4} < \frac{\left[2(V_{high}-V_{te})V_{te}-V_{te}^2\right]}{\left[2(-V_{td})(3v-V_{te})-(3v-V_{te})^2\right]} \times \frac{W_3}{L_3} \qquad 2.11$$

Assume $V_{te} = 1v$, $V_{td} = -2.5v$, and $V_{high} = 5v$ Equation 2.11 gives:

$$\frac{W_4}{L_4} < 1.17 \times \frac{W_3}{L_3}$$

The parasitic capacitance at node p and w, C_{p1} and C_{p2} in Figure_2-8a, can be minimized if both transistors M3 and M4 are minimum size device.

Transistor M5 is in saturation because $V_{gd5} = 0v < V_{te}$. The size of this transistor can be found by equating the drain current of M5 and M3:

$$I_{d3}(res) = I_{d5}(sat) \qquad 2.12$$

The drain current of a transistor in the saturation region is given by [H&J83]:

$$I_d(sat) = \frac{k'}{2} \times \frac{W}{L} \times (V_{gs}-V_t)^2 \qquad 2.13$$

Using Equation 2.13 and the voltages shown in Figure_2-8b, Equation 2.12 becomes:

$$\frac{W_3}{L_3} \times \left[2(V_{high}-V_{te})V_{te}-V_{te}^2\right] > \frac{W_5}{L_5} \times (V_{dd}-3v-V_{te})^2$$

Solve for $\frac{W_5}{L_5}$

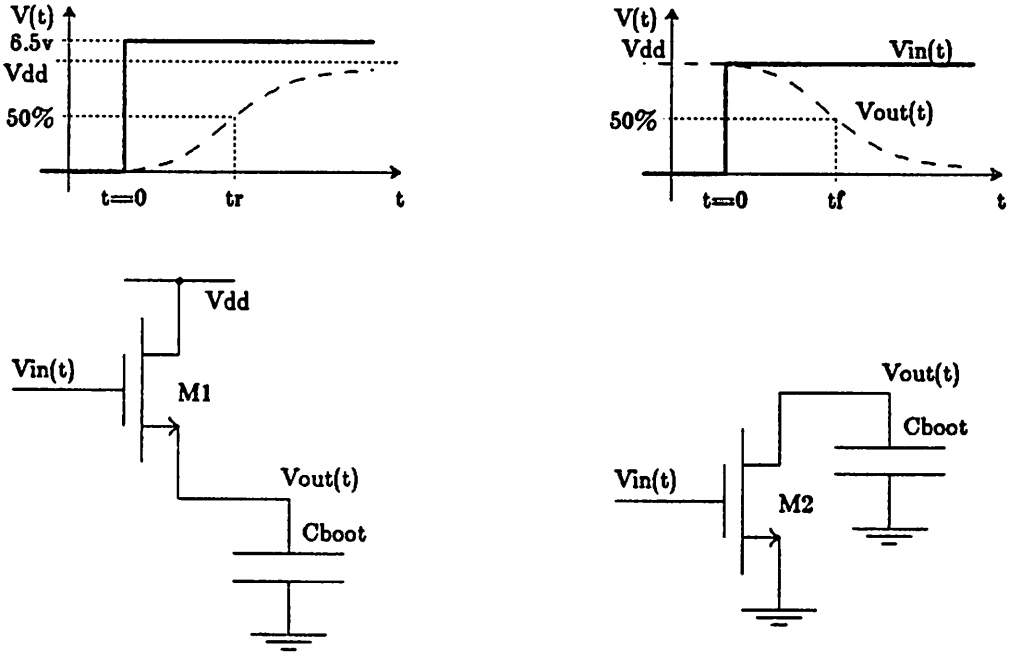
$$\frac{W_5}{L_5} > \frac{\left[2(V_{high}-V_{te})V_{te}-V_{te}^2\right]}{(V_{dd}-3v-V_{te})^2} \times \frac{W_3}{L_3} \qquad 2.14$$

Using the same assumption as above namely $V_{high} = 5v$, $V_{te} = 1v$, and $V_{dd} = 5v$ Equation 2.14 gives:

$$\frac{W_5}{L_5} > 7 \times \frac{W_3}{L_3}$$

2.4.2 Sizes Of The Two Output Transistors M1 and M2

The sizes of the two output transistors M1 and M2 (see Figure_2-8a) can be approximated by the two simple models shown in Figure_2-9 and a specification of the 50% rise time t_r and fall time t_f .



(a) Size Of Pull-Up (b) Size Of Pull-Down

Figure 2-9 Simple Circuits For Calculation Of Output Transistors' Size

Figure_2-9a is a simple model for estimating the size of the output pull up transistor M1.

At $time = 0$, $V_{gs} = 6.5v$, $V_{ds} = 5v$ transistor is resistive. Using equation 2.10:

$$I_d(0) = \frac{k'}{2} \times \frac{W_1}{L_1} \times \left[2(6.5v - V_{t0})(5v) - (5v)^2\right] \qquad 2.15$$

At $time = t_r$, $V_{gs} = 6.5v$, $V_{ds} = 2.5v$ transistor is resistive. Using equation 2.10:

$$I_d(t_r) = \frac{k'}{2} \times \frac{W_1}{L_1} \times \left[2(5v - V_{t2.5})(2.5v) - (2.5v)^2\right] \qquad 2.16$$

In equation 2.16, $V_{t2.5}$ is the threshold voltage of transistor M1 when its source to body voltage V_{sb} equals to 2.5v. From [H&J83] :

$$V_{t2.5} = V_{t0} + \gamma\left\{\left[2.5v + 2\left|\phi_f\right|\right]^{1/2} - \left[2\left|\phi_f\right|\right]^{1/2}\right\}$$

$$V_{t0} = \textit{threshold voltage when } V_{sb} = GND$$

Assume the average current during transition equals to the arithmetic mean of equation 2.15 and 2.16, then:

$$t_r = \frac{2.5v \times C_{load} \times 2}{I_d(0) + I_d(t_r)} \qquad 2.17$$

Substitute equation 2.15 and 2.16 into equation 2.17 and solve for for W_1/L_1 :

$$\frac{W_1}{L_1} = \frac{10v \times C_{load}}{k' \times t_r \times \left[2(6.5v - V_{t0})(5v) + 2(5v - V_{t2.5})(2.5v) - 31.25v^2\right]} \qquad 2.18$$

Figure_2-9b is a simple model for estimating the size of the output pull down transistor M2.

At $time = 0$, $V_{gs} = 5v$, $V_{ds} = 5v$ transistor is in saturation. Using equation 2.13:

$$I_d(0) = \frac{k'}{2} \times \frac{W_2}{L_2} \times (5v - V_{t0})^2 \qquad 2.19$$

At $time = t_f$, $V_{gs} = 5v$, $V_{ds} = 2.5v$ transistor is resistive. Using equation 2.10:

$$I_d(t_f) = \frac{k'}{2} \times \frac{W_2}{L_2} \times \left[2(5v-V_{t0})(2.5v)-(2.5v)^2\right] \qquad 2.20$$

Assume the average current during transition equals to the arithmetic mean of equation 2.19 and 2.20, then:

$$t_f = \frac{2.5v \times C_{load} \times 2}{I_d(0) + I_d(t_f)} \qquad 2.21$$

Substitute equation 2.19 and 2.20 into equation 2.21 and solve for W_2/L_2 :

$$\frac{W_2}{L_2} = \frac{10v \times C_{load}}{k' \times t_f \times \left[(5v-V_{t0})^2+2(5v-V_{t0})(2.5v)-(2.5v)^2\right]} \qquad 2.22$$

The 50% rise and fall time t_r and t_f are usually given as design goals of the bootstrap driver. Using these and some process parameters, Equation 2.18 and Equation 2.22 can be solved for the approximate sizes of M1 and M2. These approximate sizes can then be used as the starting point for further design iteration. Computer aided design tools such as circuit simulator Spice must be used for any further design iteration.

2.5 Extra Depletion Mode Pull Up

The bootstrap driver shown in Figure_2.10 is identical to the basic bootstrap driver shown in Figure_2.8a except a small (relative to the pull down transistor M2) depletion mode pull up transistor M6 is added. This depletion mode transistor ensures the high voltage level at the output node (node y) won't drop below V_{dd} even if input remains low for a long period of time.

There are several side effects when this transistor (M6) is added:

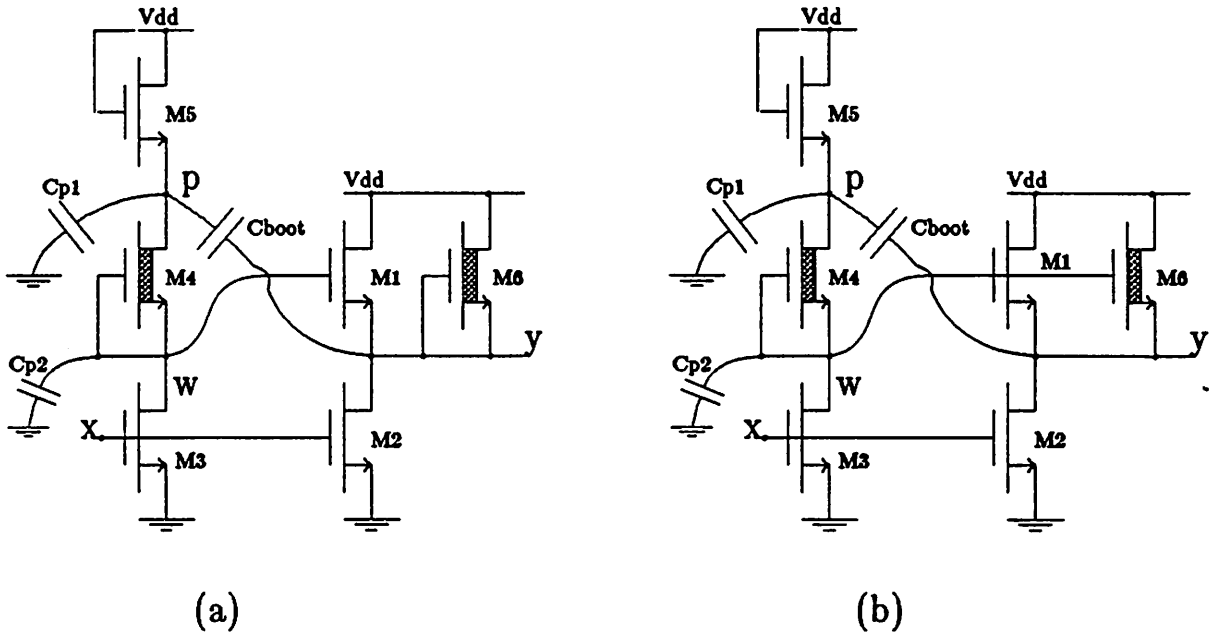


Figure 2-10 Addition Of An Extra Depletion Mode Transistor

(1) The rise time of the bootstrap driver is improved especially when the gate of the depletion mode transistor is connected to node w as shown in Figure_2-10b.

(2) An extra capacitor, namely the gate capacitance of transistor M6 (C_{g6}) , is now added to node w in parallel with C_{g1} and C_{p2}. As a result, charge sharing problem described in Section 2.2.2. gets worse.

(3) The low level voltage at node y is no longer GND but a little bit higher. This will reduce the voltage (hopefully by a small amount) across the bootstrap capacitor C_{boot} when input is low.

Side effect 2 becomes important only if the gate area of the depletion mode transistor M6 (see Figure_2.10) become big relative to transistor M1. Furthermore this side effect can be taken into account by the charge sharing analysis in Section 2.2.2 if C_{g1} in all equations are replaced by an effective C_{g1} :

$$C_{g1}eff = C_{g1} + C_{g6}$$

Side effect 3 is small unless W_6/L_6 of transistor M6 becomes too big relative to W_2/L_2 of M2. In any case, W_6/L_6 should be smaller than $(1/k)\times(W_2/L_2)$ where k=4 if node x can be driven to V_{dd} and k=8 if node x is driven high through a pass transistor [M&C80].

The rise time of the bootstrap driver can be improved by making M6 larger and connecting its gate to node w (see Figure_2.10b). However in doing so, we are taking the risk of making side effects 2 and 3 bigger. It is obvious that there is a compromise between the size of M1, M6 and the size of the bootstrap capacitor. Unfortunately the best combination of these three is not obvious. One approach to this problem is to run Spice simulation on different combinations until a reasonable compromise is achieved. This was done when the bootstrap drivers for SOAR [Ung84] were designed. These drivers are described in Section 2.6.

2.6 Examples Of Bootstrap Drivers

There are six different types of bootstrap drivers in SOAR [Ung84] which are ctrdriver, lowctrdriver, ungatedctrdriver, granddaddy, addrdriver, and 3statedriver. Addrdriver and 3statedriver are pad drivers while all others are control line drivers. Their circuit diagrams are shown in Figure_2-11 and Figure_2-12 is the layouts of these cells. Notice the following:

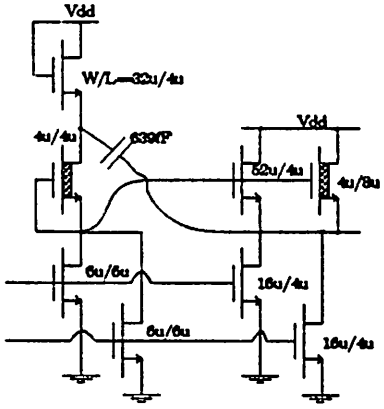


Figure 2-11a ctrdriver

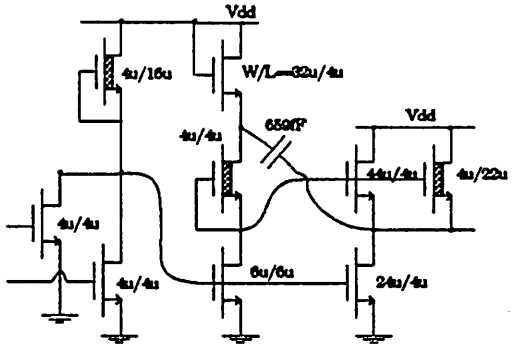


Figure 2-1b lowctrdriver

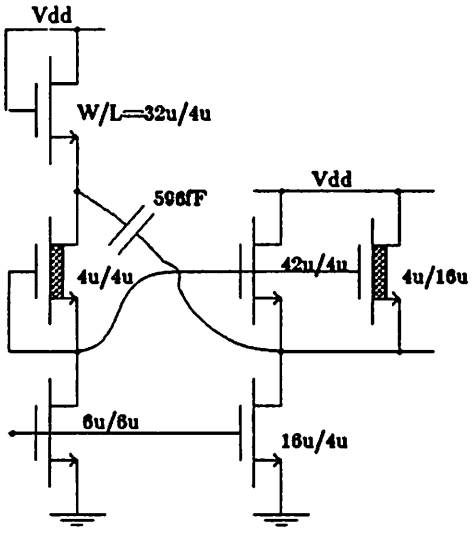


Figure 2-11c ungatedctrdriver

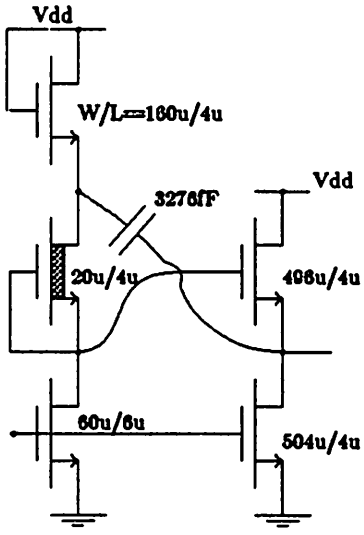


Figure 2-11e addrdriver

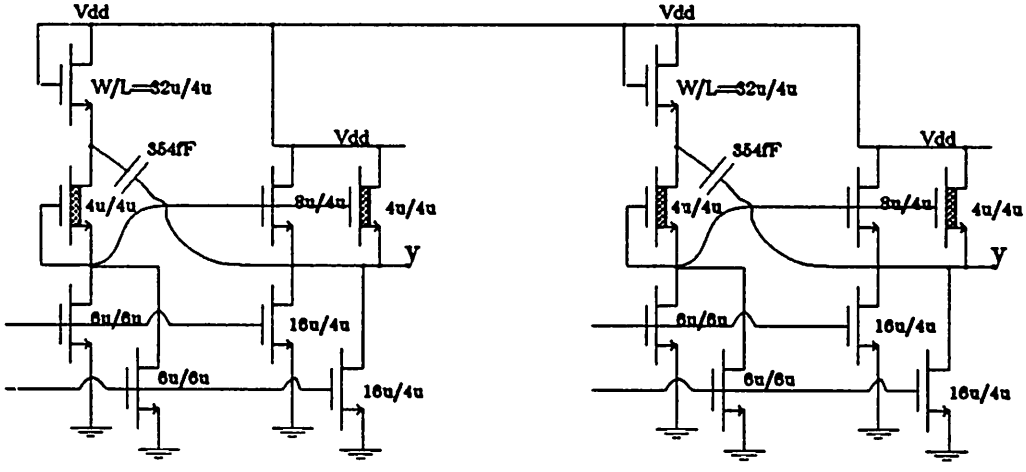


Figure 2-11d Bootstrap Driver - granddaddy

(1) All input transistors have a L=6u instead of the minimum requirement of L=4u. The reason is that the drain of this transistor is bootstraped to a voltage higher than V_{dd} and its source is at GND. Under a source-drain voltage higher than *Vdd*, punch through may occur if the minimum length is used.

(2) The two pad drivers addrdriver and 3statedriver do not have the extra depletion mode pull up described in Section 2.5. The reason is that these two drivers are used to drive the address and data lines which are expected to change every cycle.

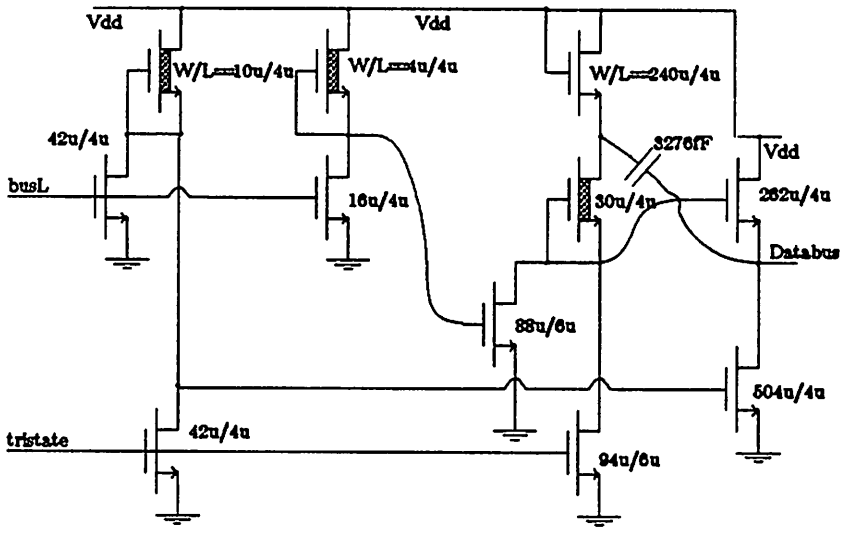


Figure 2-11f Output Driver - 3statedriver

A test chip, which contains all these drivers, was built to test their performance. The output of the two pad drivers addrdriver and 3statedriver can connect to the output pads of the test chip directly.

The outputs of the other drivers, which are not designed to drive I/O pads, cannot connect directly to the output pads of the test chip. Instead a source follower is used as shown in Figure_2-13. To simulate these drivers' working environment inside SOAR, long poly lines are used to connect the output of these drivers to the source followers. The length of these poly lines are approximately the same as the poly control lines in SOAR. Furthermore the size of the source follower is sixteen times bigger than minimum size because sixteen minimum size gates are attached to each control line in SOAR.

The rise and fall time measurements of addrdriver and 3statedriver showed that these drivers are able to drive the pads to 5V or GND within 50ns after input has crossed the 50% point. The input and output waveforms recorded during the rise time measurement of 3statedriver is shown in Figure_2-14.

The rise time measurements of other drivers are done with the help of the variable resistor. The variable resistor, which is connected between the pad and GND (see Figure_2-13), are adjust-

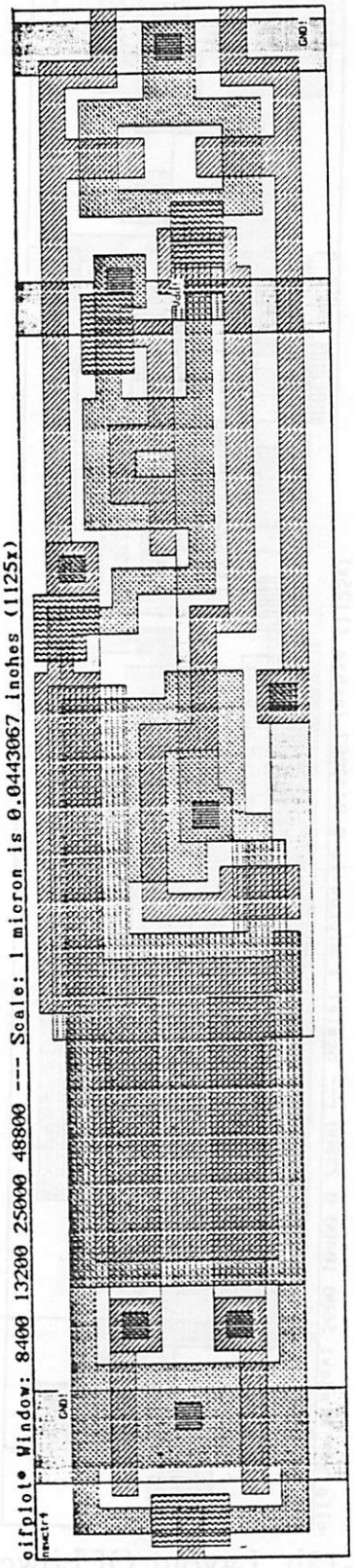


Figure 2-12a Layout Of Ctrdriver

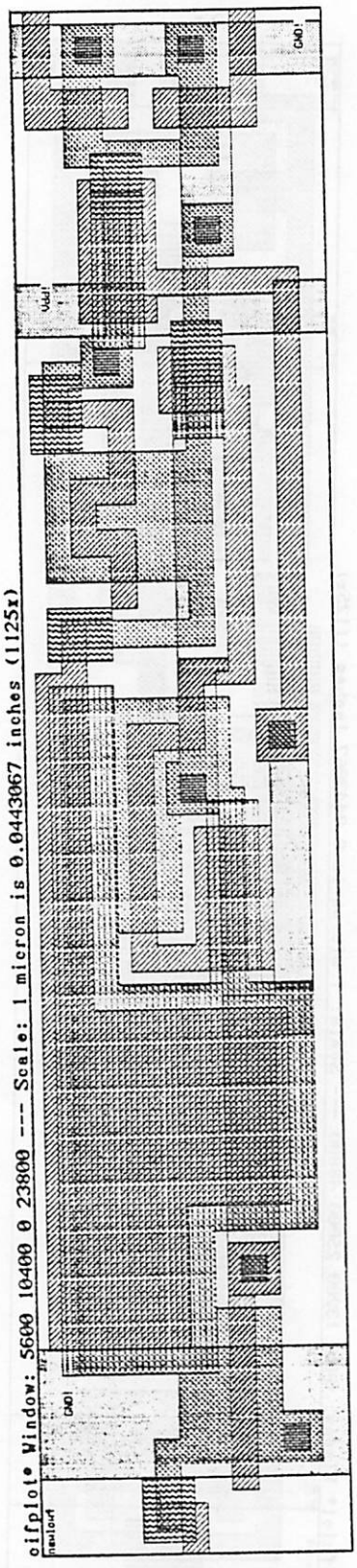


Figure 2-12b Layout Of Lowctrdriver

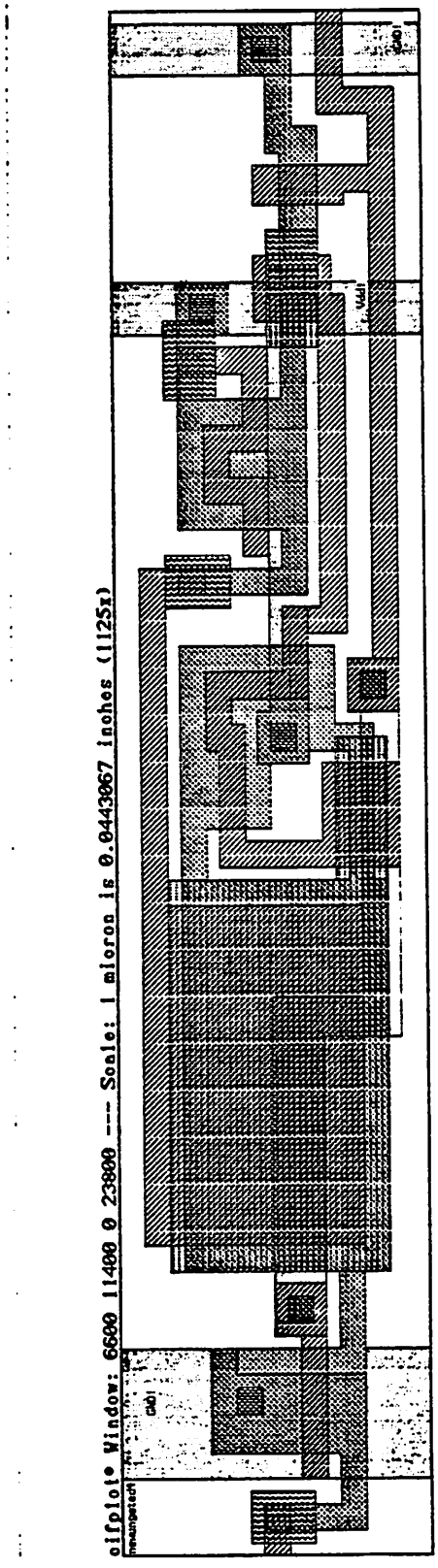


Figure 2-12c Layout Of Ungatedctrdriver

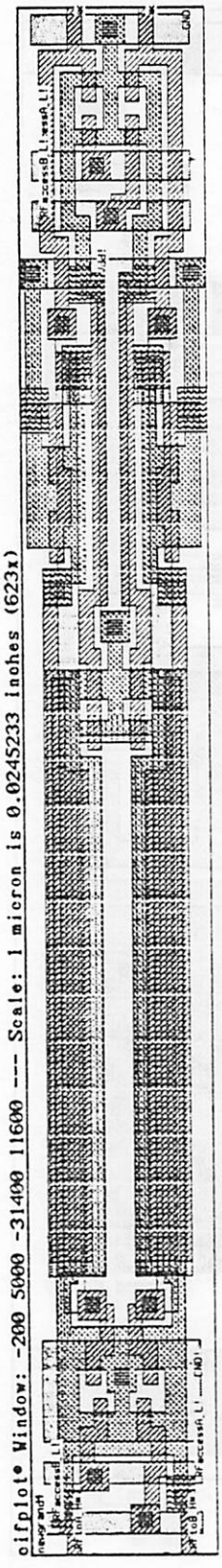


Figure 2-12d Layout Of Granddaddy

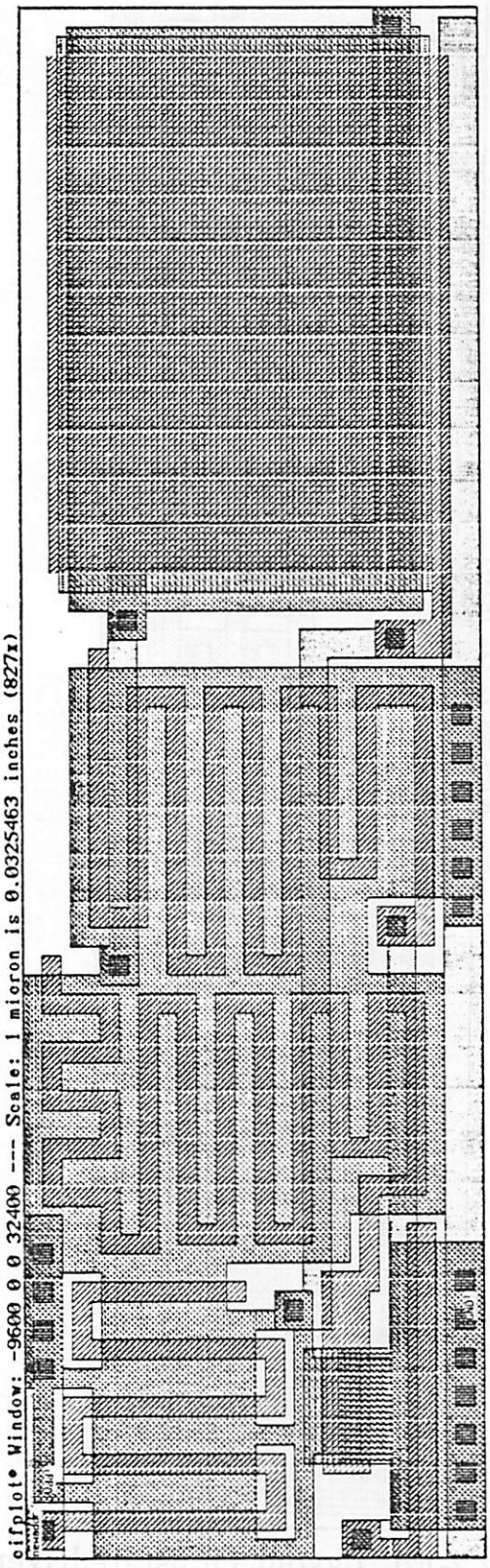


Figure 2-12e Layout Of Addrdriver

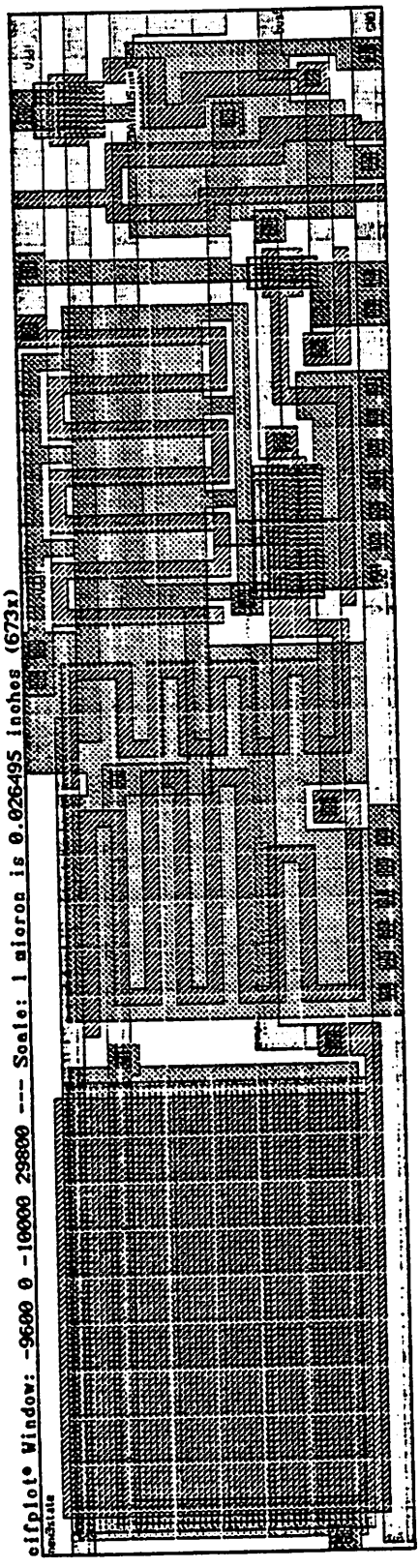


Figure 2-12f Layout Of 3statedriver

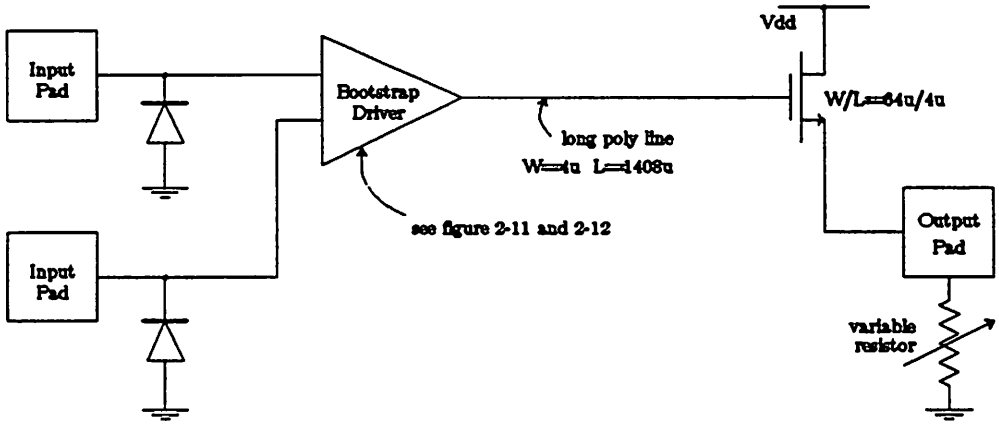


Figure 2-13 Test Setup For Testing SOAR's Bootstrap Drivers

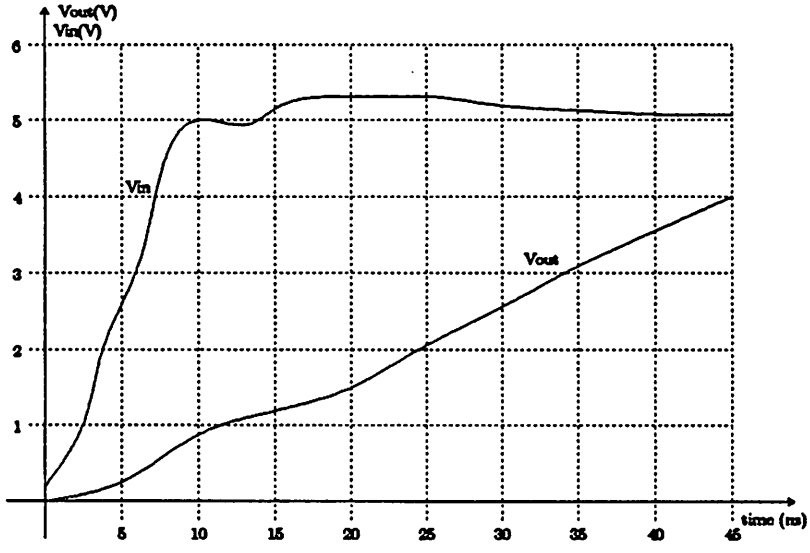


Figure 2-14 3statedriver's Rise Time Measurement

ed until the voltage swing is 0.5V at the probe. This voltage swing is kept small such that the delay from the gate of the source follower to the probe is minimized. Figure_2-15 shows that ctrdriver can drive the probe to 0.5V within 30ns after input has crossed the 50% point. If the delay from the gate of the source follower to the probe is negligible, then it implies the ctrdriver is able to drive the load to 5V within 30ns. Similar measurements, which are not shown here, indicated all drivers are able to drive the probes to 0.5V within 40ns. Since source followers are used, the fall time cannot be measured. However the fall time was found to be much shorter than the

rise time in Spice simulation.

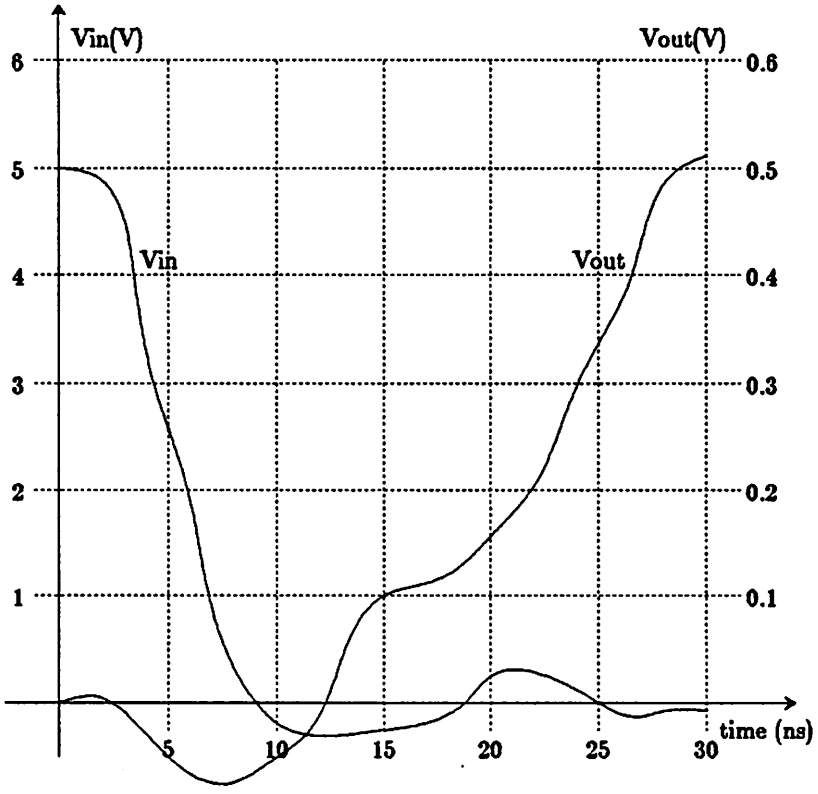


Figure 2-15 Ctrdriver's Rise Time Measurement

3. CMOS DYNAMIC CIRCUIT

There are many ways to implement dynamic logic in CMOS. Most of these approaches require some complicated clocking scheme. Domino [Kra82] and NORA [Gon83] are two exceptions which only require a simple clocking scheme. In section 3.1 and 3.2, CMOS Domino and NORA logic are discussed. In section 3.3, charge sharing, which is a common problem in all dynamic circuits, is described together with methods to prevent or at least control it.

3.1 Domino Circuit

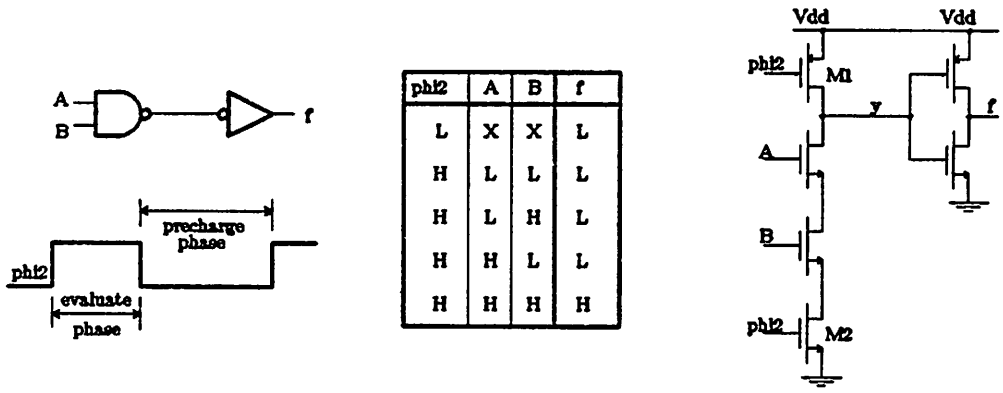
3.1.1 Basic Structure

The two basic logic gates, the AND gate and the OR gate, implemented in Domino logic are shown in Figure_3-1. These two Domino gates operate as follows:

$\phi_2 = GND$ – This is the precharged phase of the gate. The precharge transistor M1 is on and the evaluation transistor M2 is off. As a result, node y is precharged to high (V_{dd}) and the output node (node f) is precharged to low (GND).

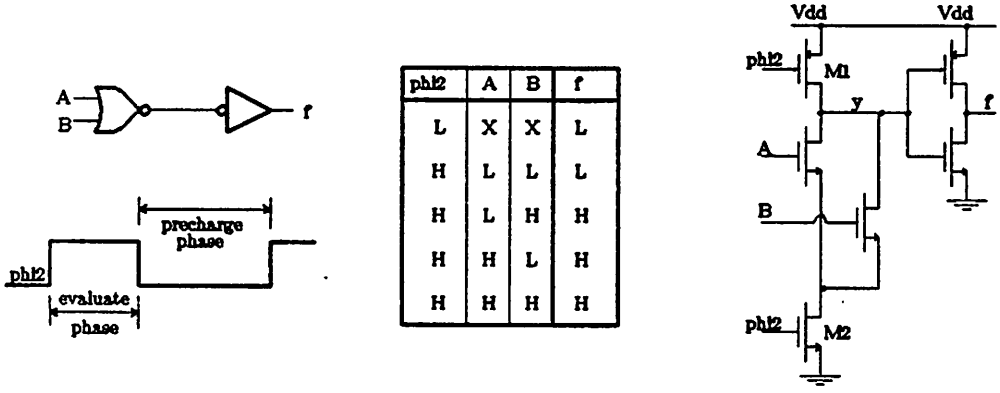
$\phi_2 = V_{dd}$ – This is the evaluation phase of the gate. M1 is off and M2 is on, enabling node y to be discharged to GND conditionally. Assuming inputs A and B are stable during this time, then node y of the AND gate will be discharged to GND only if both A and B are high. On the other hand, node y of the OR gate will be discharged to GND if either one of the inputs is high.

In the above discussion, ϕ_2 can either be the system clock in a single phase system or one phase of the system clock in a multi phase system. In either case, especially the multi phase case, the time during which $\phi_2 = GND$ is likely to be longer than the time during which $\phi_2 = V_{dd}$. Furthermore, the precharge transistor M1 connects V_{dd} to the precharge node (node y) directly (not in series with anything). Because of these two reasons, the precharged transistor M1 can be



ph12	A	B	f
L	X	X	L
H	L	L	L
H	L	H	L
H	H	L	L
H	H	H	H

Figure 3-1a AND Gate Implemented In Domino Logic



ph12	A	B	f
L	X	X	L
H	L	L	L
H	L	H	H
H	H	L	H
H	H	H	H

Figure 3-1b OR Gate Implemented In NORA Logic

small.

The assumption of both inputs A and B being stable during the evaluation phase is an overly conservative requirement. The only requirement one needs to impose on the inputs is to ensure that node y won't be discharged accidentally during the evaluation phase. Once node y is discharged, no active device exists to drive it back to V_{dd} until the next precharge phase. Instead of requiring all inputs be stable (no transition) during the evaluation phase, accidental discharge can still be avoided even if inputs are allowed to make one transition. However this single transition MUST be a low to high transition which implies all inputs must start out low at the beginning of the evaluation phase. This is illustrated in Figure_3-2 and is summarized below:

The input must either be stable or makes at most one low to high transition (this implies the input must start out low) during the evaluation phase.

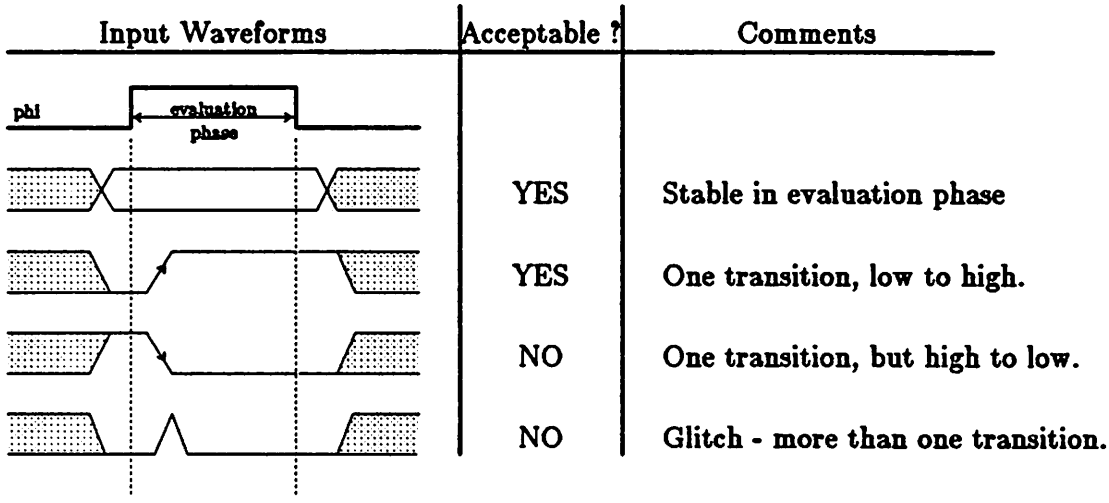


Figure 3-2 Accepatable Input Waveforms For Domino Gates

This simplification in input requirements makes the connection of Domino gates much easier (see Section 3.1.2) but it also introduces a charge sharing problem to some Domino gates. The charge sharing problem will be discussed in Section 3.3.

3.1.2 Connection Of Domino Gates

The only requirement on the Domino gates' inputs during the evaluation phase, as explained at the end of Section 3.1.1, is that they must either be stable or start out low and make one low to high transition. This simple requirement together with the property of the Domino gate's output make connecting Domino gates together very easy.

The property of the Domino gate's output can be understood by examining Figure_3-1. In either the AND gate (Figure_3-1a) or the OR gate (Figure_3-1b), node y is precharged to V_{dd} in the precharged phase and then it is isolated from V_{dd} during the evaluation phase when M1 is off. Consequently node y, which is precharged to high, can make at most one high to low transition

during the evaluation phase. The output node (node f), which is driven by node y through an inverter, is therefore precharged to low and can make at most one low to high transition.

Notice that the property of the Domino gate's output fits exactly the input requirement of the Domino gate. This implies the output of a Domino gate can be connected to the input of another Domino gate directly. This is illustrated in Figure_3-3 which shows the implementation of the logic function $f = (A \cdot B) + (C \cdot D)$. As indicated in the timing diagram, as soon as ϕ_2 goes high, $(A \cdot B)$ and $(C \cdot D)$ are evaluated by the two AND gates. The low to high transition (if any) of these two signals then cause the OR gate to evaluate the function f. This is similar to the behavior of a row dominos toppling into one another and this is the reason why it is called Domino logic.

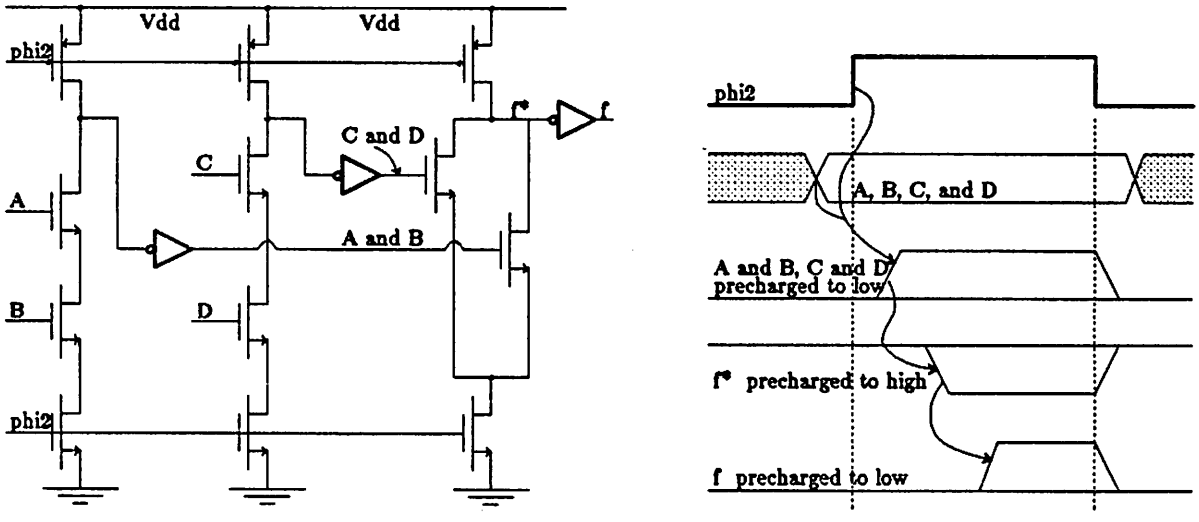


Figure 3-3 Implementation Of (A and B) or (C and D) In Domino Logic

3.1.3 Timing Considerations At The Circuit Boundary

In the example shown in Figure_3-3, input signals A, B, C, and D of the two AND Domino gates must follow the same rule that governs all Domino gates' inputs. That is, they must either be stable or make at most one low to high transition during the evaluation phase.

The easiest way to ensure these signals make at most one low to high transition during the evaluation phase is to generate these signals from other Domino gates whose evaluation phase is also ϕ_2. Unfortunately this is sometimes very hard or even impossible to accomplish. For example, these signals can be outputs of some static registers or can be inputs from the external world. In these cases, it will be much easier to ensure these signals be stable (no transition) during the evaluation phase by latching them into a dynamic latch prior to the evaluation phase.

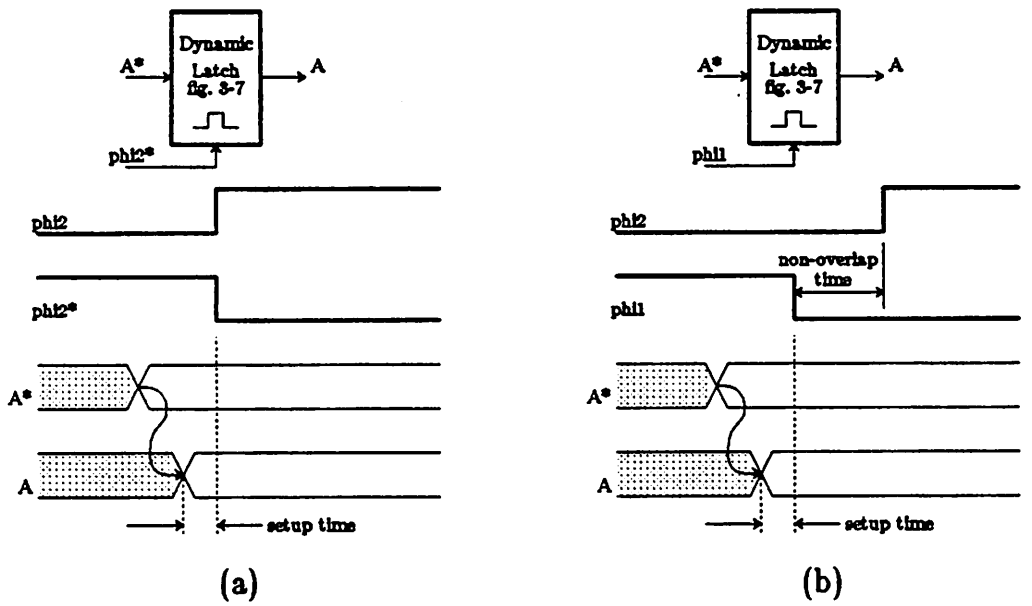


Figure 3-4 Two Ways Of Using A Dynamic Latch

There are two ways of using a dynamic latch. In Figure_3-4a, the inverse of the evaluation clock ($\overline{\phi_2}$) is used for latching and in Figure_3-4b, an extra phase ϕ_1, which does not overlap ϕ_2, is used. The approach in Figure_3-4a has the advantage of using a single phase clock while the approach in Figure_3-4b has much higher tolerance of clock skew. This is illustrated in Figure_3-

5. Figure_3-5a shows that a clock skew larger than the set up time of the latch is fatal for the single phase approach while the two non-overlap phase approach (Figure_3-5b) can tolerate a clock skew up to the sum of the set up and the nominal non-overlap time between the two phases. Since the nominal non-overlap time is specified by the system designer, the designer has much more control over the clock skew problem in the two-phase approach.

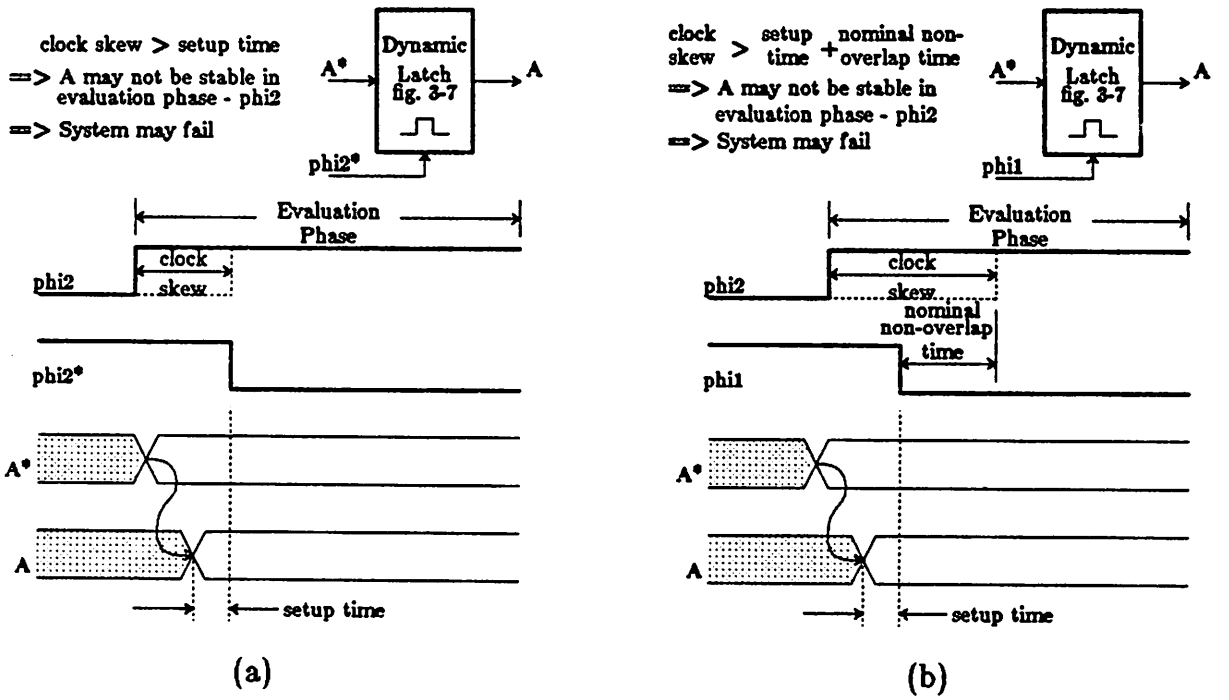
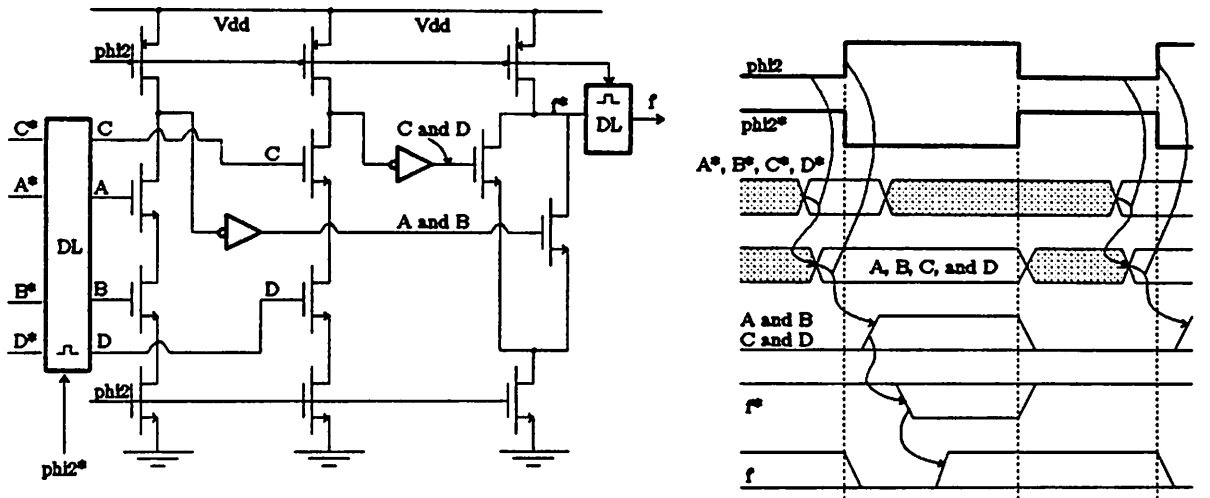


Figure 3-5 Comparison Of Clock Skew Tolerance

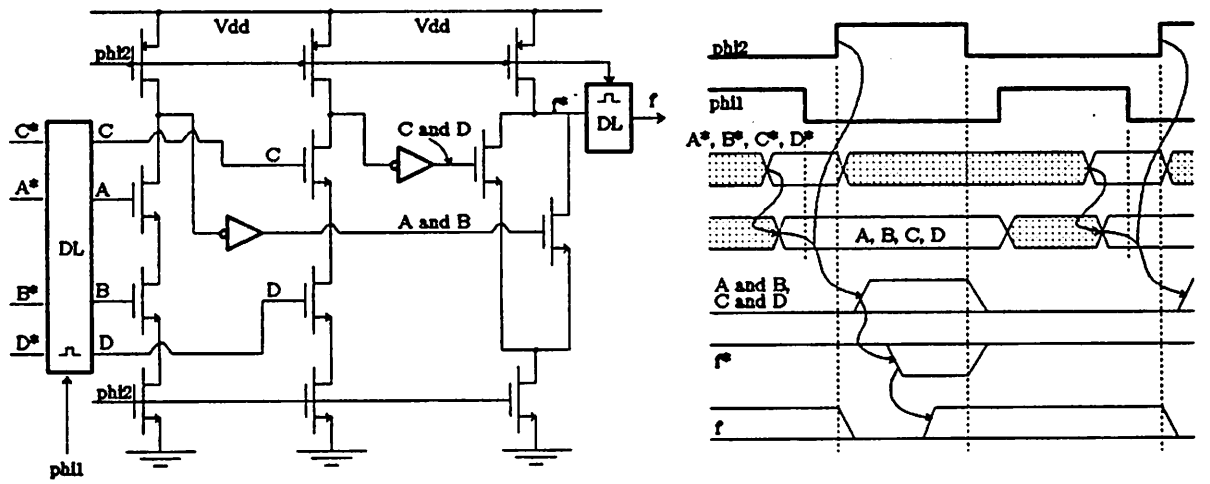
The output of the circuit (node f) in Figure_3-3 will be valid at the later part of the evaluation phase (ϕ_2=V_{dd}). After ϕ_2 goes low, node f* is precharged to V_{dd} and node f is precharged to GND. In other words, the output is valid only during part of the evaluation phase. The time at which output is valid can be prolonged if the inverter between node f* and node f is replaced by a dynamic latch as shown in Figure_3-6. Figure_3-6a and b show the two different approaches in latching the inputs that are discussed in the last paragraph. In either case, the value at node f* must be latched in during ϕ_2, not during $\overline{\phi_2}$. During $\overline{\phi_2}$, node f* is precharged to V_{dd}.

The dynamic latches used here can be implemented in two different ways as shown in Figure_3-7. Figure_3-7a uses a composite pass gate followed by an inverter and Figure_3-7b is a



DL - Dynamic latch (see figure 3-7)

Figure 3-6a Implementation Of (A and B) or (C and D) - Single Phase Approach



DL - Dynamic latch (see figure 3-7)

Figure 3-6b Implementation Of (A and B) or (C and D) - Two Phase Approach

C^2MOS latch [Suz73]. Although these two implement identical logic, they have very different electrical behavior. The details of their electrical behavior will be discussed in Section 3.3.2. For the discussion here, it is sufficient to state that the following rules must be followed:

(1) If the input node (node x) is a dynamic node (node is precharged to either V_{dd} or GND), C^2MOS latch MUST be used.

(2) If the input node (node x) is NOT a dynamic node; approach 1, which uses a composite pass gate followed by an inverter, is preferred but the C^2MOS latch can also be used.

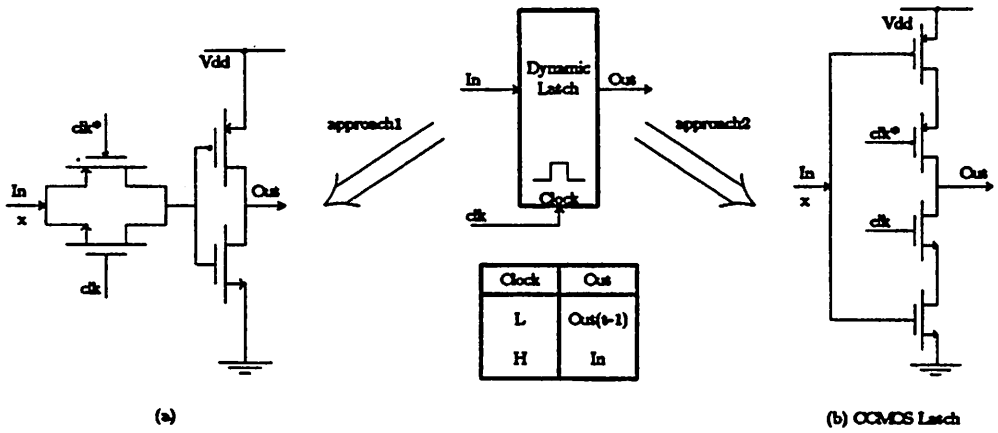


Figure 3-7 Two Ways To Implement A Dynamic Latch

As a result of these two rules; the dynamic latch between node f* which is precharge node, and node f (see Figure_3-6) MUST be a C^2MOS latch. On the other hand, the type of dynamic latch to be used at the input depends on whether signals A, B, C, and D are actively driven or are precharged.

3.1.4 How To Use Non-Overlapping Multi-Phase Clocks

After reading Section 3.1.3, some readers may think any complicated combinational circuit (for example a 32 bit ALU) can be implemented in Domino logic using a single phase clock (with its complement) as shown in Figure_3-8. This is true in theory and it also has been done. However it has some practical problems.

One problem is that Domino logic does not provide logic inversion. When both a signal s and its complement s* is needed, extra logic gates are needed. Consider the implementation of the following example:

$$f = x \; XOR \; y \qquad 3.1$$

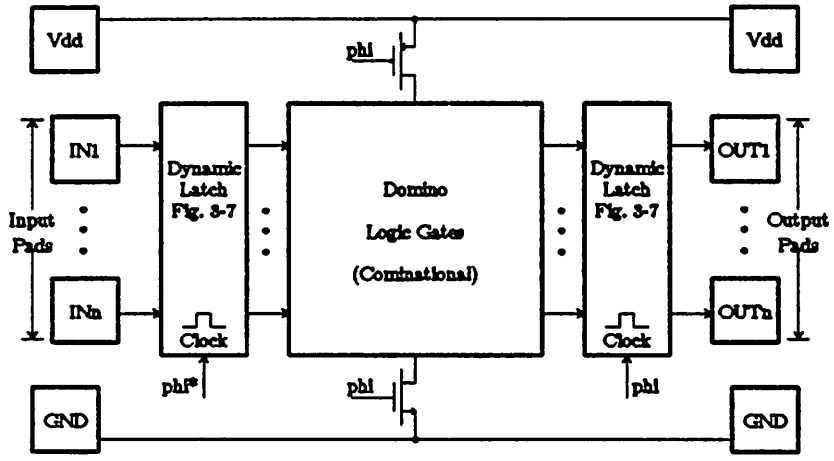


Figure 3-8 Implementation Of Domino Logic Using A One Phase Clock

where

$$x = a \; OR \; b$$

$$y = c \; OR \; d$$

One simple (but wrong) way to implement this in a one phase system is shown in Figure_3-9a. Unfortunately this is also the WRONG way to do it because both signals x* and y* violate the input requirement of the Domino gate (see Section 3.1.1 and Figure_3.2). Both these signals are high at the beginning of the evaluation phase and therefore has the potential of discharging node f* by accident.

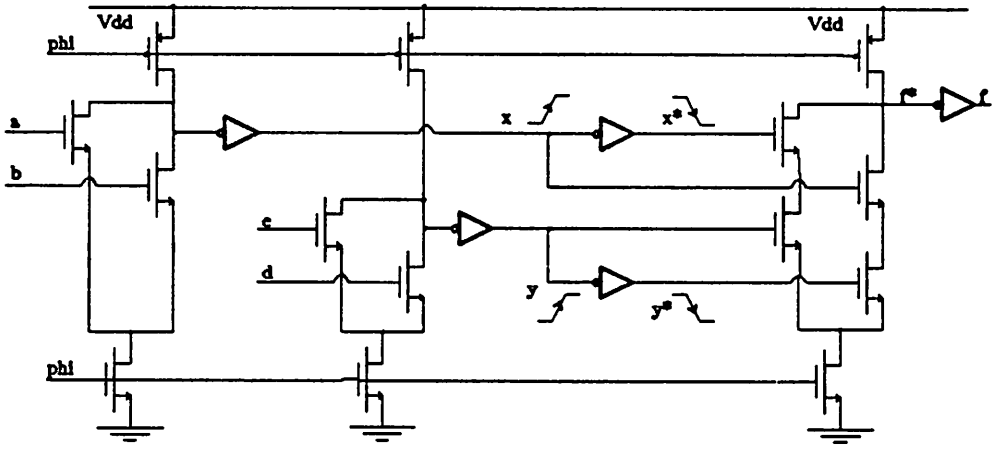


Figure 3-9a Erroneous Implementation Of (a or b) xor (c or d)

One possible race condition that can lead to accidental discharge is shown in Figure_3-9a. In this case both x and y are supposed to be high during the evaluation phase (both x* and y* are therefore low during the evaluation phase). During the precharged phase ($\phi = GND$), both x and y are precharged to low (which is desirable) and as a result both x* and y* are precharged to high (which is extremely undesirable). As soon as the evaluation begins ($\phi = V_{dd}$), both x and y will start rising and x* and y* will start falling. If for some unfortunate reason (one very reasonable reason is that the delay of the inverter is NOT zero) x rises faster than y* falls or y rises faster than x* falls then, kaboom, node f* is discharged by accident and there is no way to recover this accident.

The correct way to implement this example in a one phase system is shown in Figure_3-9b. Notice that signals a*, b*, c*, and d* are assumed to be available and the dual of the logic has to be implemented by two extra gates.

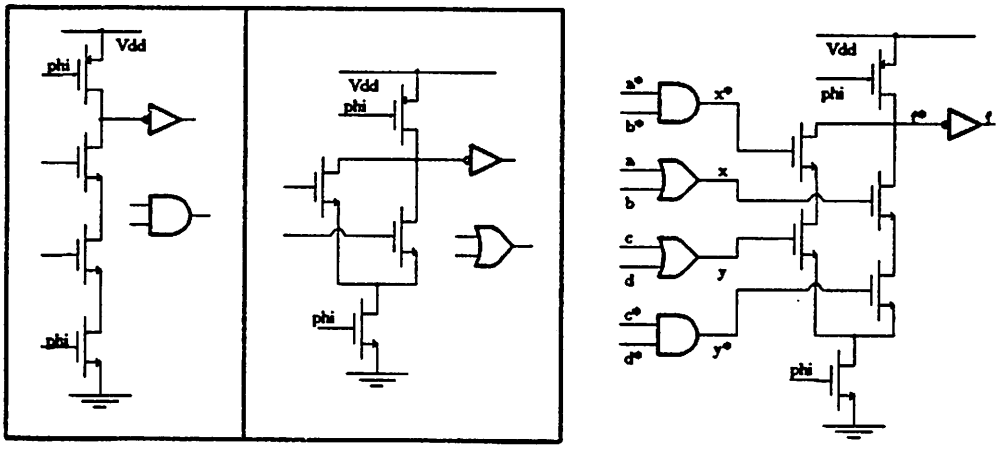


Figure 3-9b Correct Implementation Of (a or b) xor (c or d)

Figure_3-10a is another WRONG approach to the problem. From the timing diagram, it is obvious that this approach may not work because the delay of the inverters are not zero. Since the delay of the inverters are not zero, signal x* and y* may not be stable at the beginning of $\bar{\phi}=V_{dd}$. Notice that x* and y*, which may not be stable at the beginning of $\bar{\phi}=V_{dd}$, are inputs to

the second stage and $\bar{\phi}=V_{dd}$ is the evaluation phase of this second stage. This is another race condition that may cause node f* to discharge incorrectly.

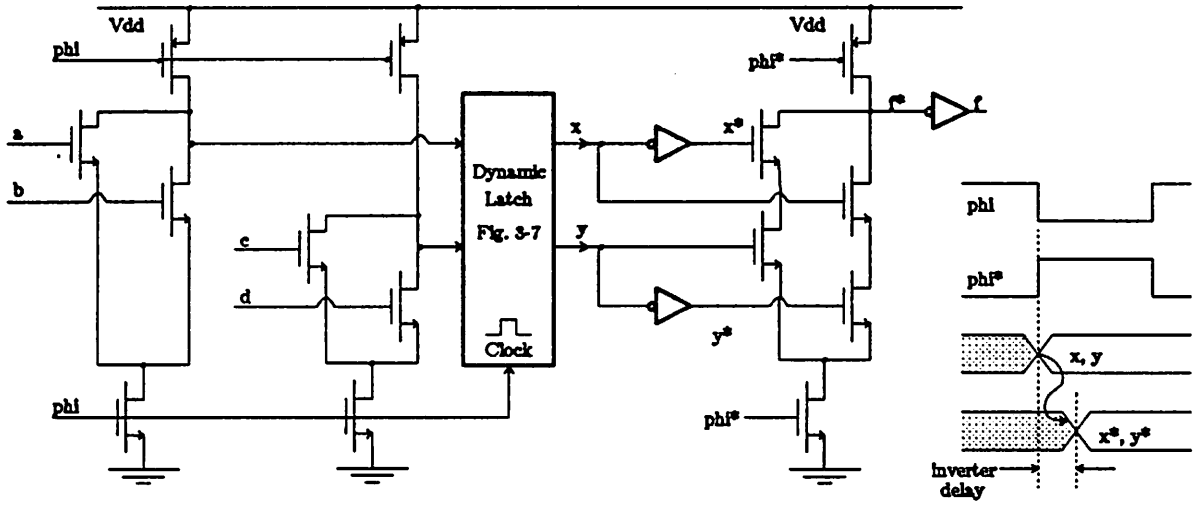


Figure 3-10a Erroneous Implementation Of (a or b) xor (a or d) Using Dynamic Latch

The implementation of both the logic and its dual as shown in Figure_3-9b can be avoided if a non-overlapping multi-phase clock is used as shown in Figure_3-10b. From the timing diagram, it can be seen that if the non-overlap time between ϕ_1 and ϕ_2 is greater than the delay of the inverter, then all inputs to the second stage (x, x* and y, y*) will be stable when the second stage enters its evaluation phase (ϕ_2 goes high). This fulfills the input requirement of Domino logic (see Section 3.1.1 and Figure_3.2) and therefore has no potential danger of discharging the precharge node (node f*) by accident.

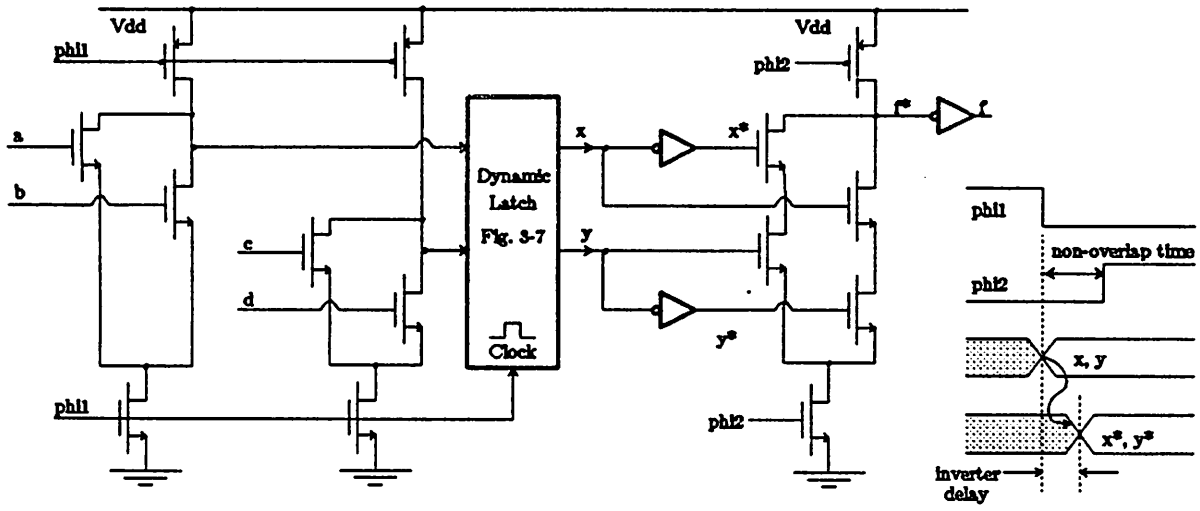


Figure 3-10b Correct Implementation Of (a or b) xor (c or d) Using Dynamic Latch

3.2 NORA Logic

3.2.1 Basic Principle

NORA logic [Gon83] is based on the pipelining concept of which Figure_3-10b is a good example. In this Figure, when $\phi_1=V_{dd}$, the first stage does its evaluation and the second stage is precharged. The output of the first stage is piped to the second stage which starts its evaluation as soon as $\phi_2=V_{dd}$. When the second stage is evaluating, the first stage is being precharged. The advantage here is that no time is wasted for precharging only. Whenever one stage is evaluating, the other stage is being precharged.

The pipeline in Figure_3-10b is not yet ideal because during the non-overlap time, both stages are being precharged (not evaluating) and this time is thus wasted.[1] The pipeline proposed in NORA logic [Gon83], which is shown in Figure_3-11, has no such dead time and only a single phase clock ϕ together with its complement $\bar{\phi}$ is required.

[1] Actually in Figure_3-10b, the non-overlap time is not quite wasted because x^* and y^* are evaluated by the static inverters during this time.

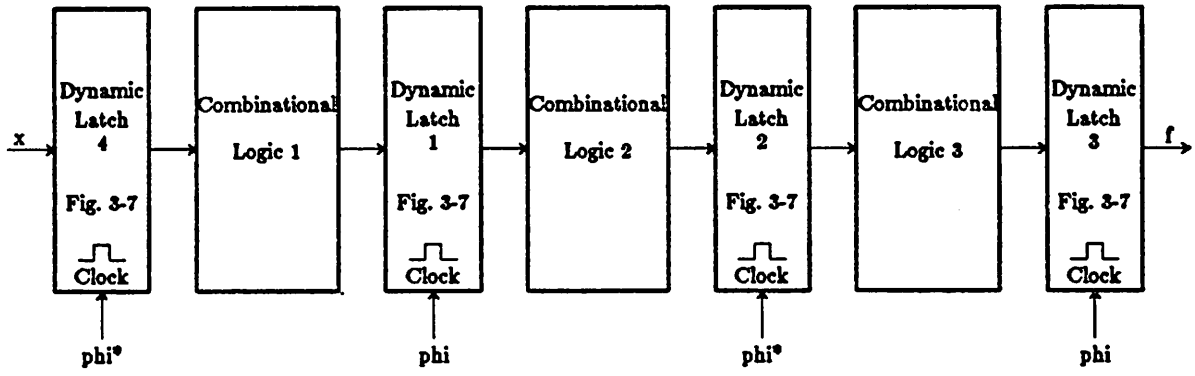


Figure 3-11 A Generic NORA Pipeline

The combinational logic within each stage of the NORA pipeline is implemented by dynamic gates that are similar to Domino logic gates. Consequently, the NORA pipeline is very similar to the ill-fated pipeline shown in Figure_3-10a. Due to the similarities between these two pipelines, static inverters cannot be used between the NORA pipeline stages (after each dynamic latch) either. Otherwise the same race condition that causes the circuit of Figure_3-10a (see Section 3.1.4) to fail will also kill the NORA pipeline. Since a static inverter is not allowed between pipeline stages, the method showed in Figure_3-10b cannot be used to implement complementary logic. The concept of N- and P-logic blocks is introduced in NORA logic to facilitates the implementation of complementary logic. This concept will be discussed in detail in Section 3.2.2.

3.2.2 The Concept Of N-logic Block And P-logic Block

The implementations of the NAND gate and NOR gate in N-logic blocks are shown in Figure_3-12a and b and Figure_3-13a and b show how the same logic is implemented in P-logic blocks. Figure_3-12c (Figure_3-13c) is a generic N-logic (P-logic) block for a more complicated logic gate. An example of a more complicated gate, which implements the AND-OR function in N-logic (P-logic) block, is shown in Figure_3-12d (Figure_3-13d).

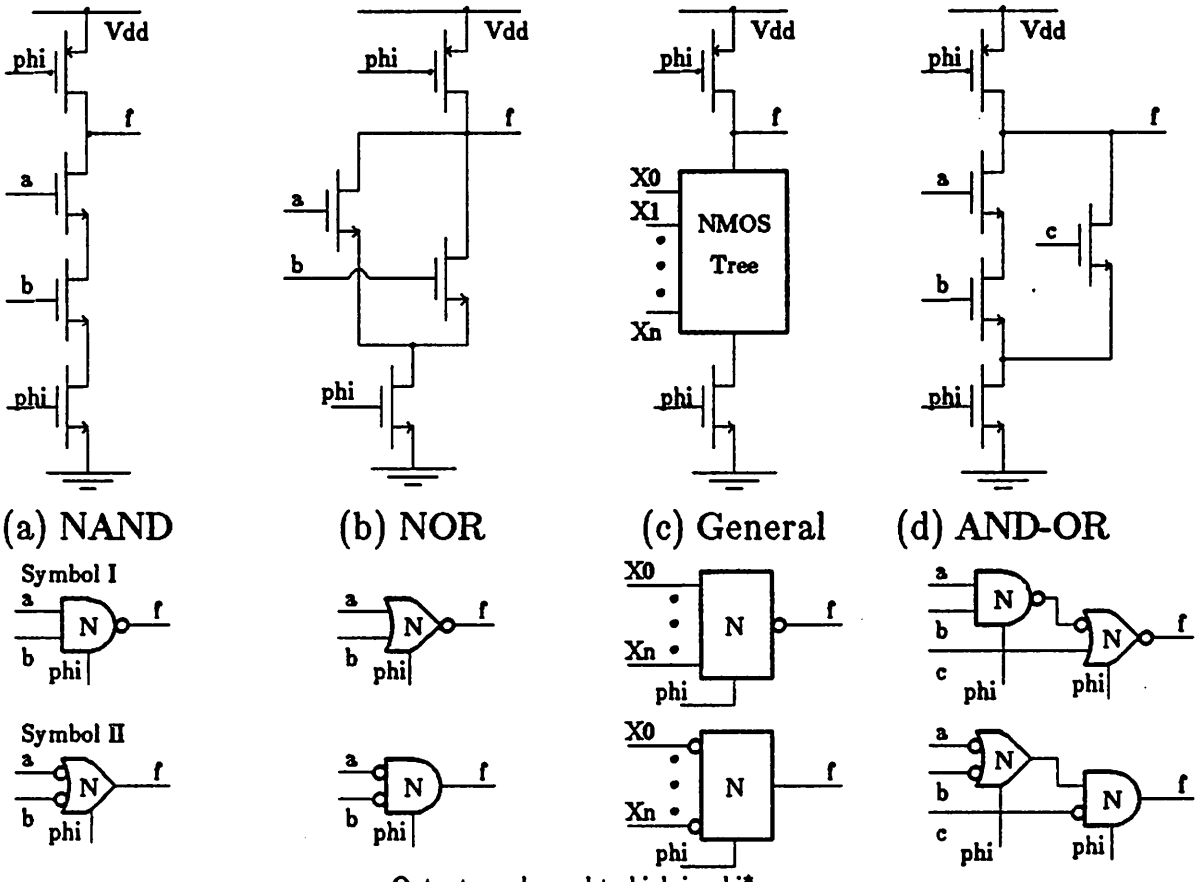


Figure 3-12 Examples Of NORA N-logic Blocks

In Figure_3-12 and 3-13, two sets of symbols (Symbol I and Symbol II) are proposed. These symbols are designed to capture both the electrical and logical behavior of the N-logic and P-logic gates. If one set of symbols is used everywhere consistently, (either use Symbol I or Symbol II exclusively) then the gate connection rules, which are direct results of the input requirement and output behavior of the N- and P-logic blocks, can be checked symbolically. This will be discussed later in this section.

N-logic gate's output (node f in Figure_3-12) is precharged to V_{dd} during $\bar{\phi}$ and is evaluated during ϕ. This is the same as a Domino gate without its inverter. The rule that governs the N-logic block's input is therefore the same as Domino logic (see Figure_3-2):

> The input must either be stable or makes at most one low to high transition (this implies the input must start out low) during the evaluation phase.

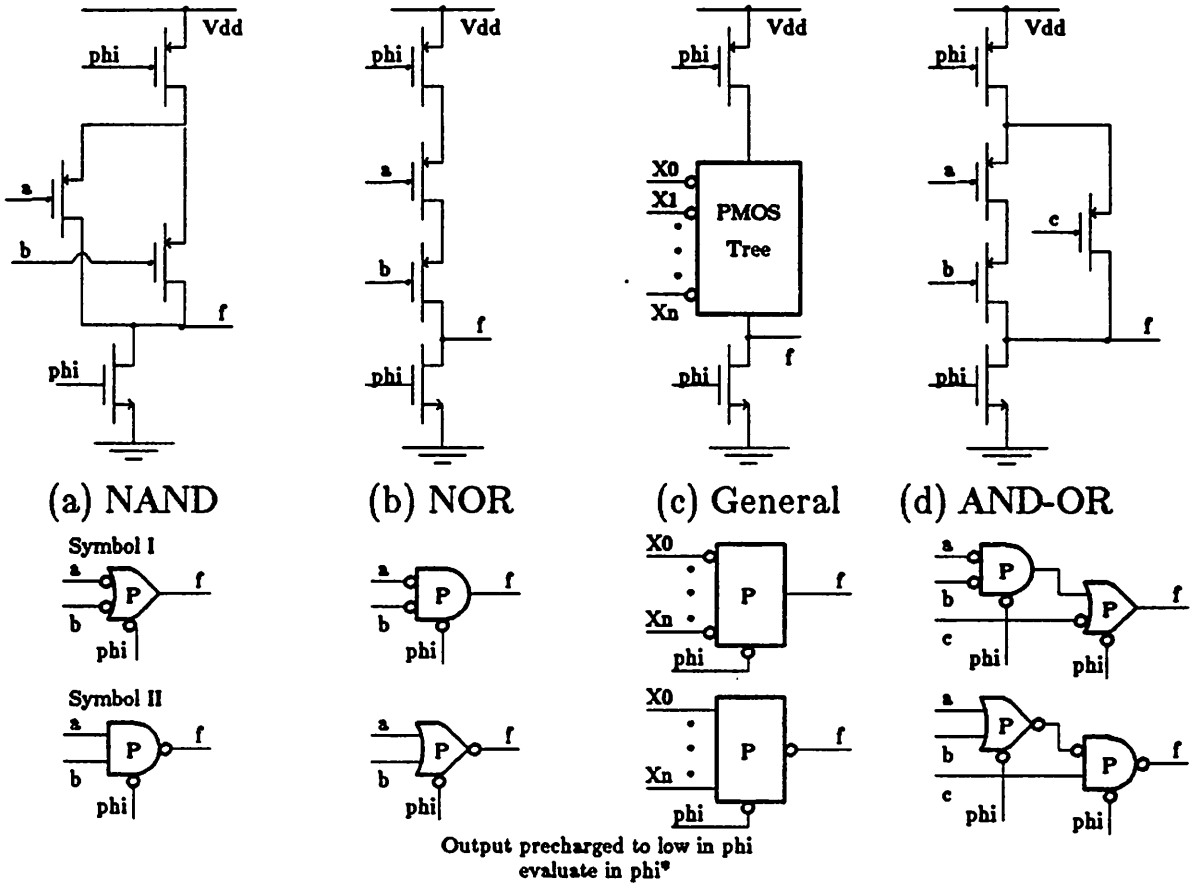


Figure 3-13 Examples Of NORA P-logic Blocks

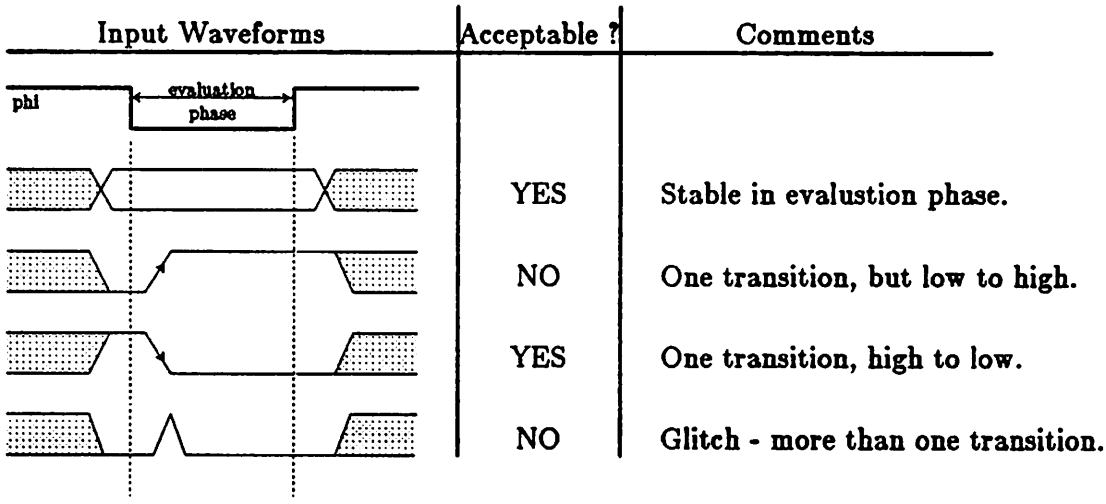


Figure 3-14 Acceptable Input Waveforms For P-logic Blocks

P-logic gate's output (node f in Figure_3-13) is precharged to GND during ϕ and is evaluated during $\bar{\phi}$. Using reasons similar to those used in Section 3.1.1, the following

requirement, which is illustrated in Figure_3-14, is derived for the P-logic block's input:

> The input must either be stable or makes at most one high to low transition (this implies the input must start out high) during the evaluation phase.

The rules concerning how these gates can be connected are direct results of the input requirement and output behavior of the N- and P-logic blocks. From the above discussions, the output of the N-logic block can make at most one high to low transition during the evaluation phase [2] and therefore can be connected to the inputs of the P-logic block directly. For similar reasons, the output of the P-logic block can be connected to the inputs of the N-logic block directly. When N-logic block is connected to N-logic block, it becomes Domino logic and an inverter must place between them for the same reasons as in Domino logic. Similarly, an inverter must also be used whenever a P-logic block is connected to other P-logic block. The rules can be summarized as:

(1) No inverter is needed when N-logic is connected to P-logic block or vice-versa.

(2) An inverter is needed when N-logic block is connected to N-logic block or when P-logic block is connected to P-logic block.

There is an easy way to check for violation of these two rules. If all gates in a circuit are represented by one set of symbols show in Figure_3-12 and Figure_3-13 (either Symbol I or Symbol II but not both), then there is no violation of either rule stated above if bubbles are matched for every signal line. The term bubble refers to the small circle that represents voltage inversion. Bubbles are matched for a signal line if either:

(a) There is no bubble on either end of the signal line, or

(b) There are two bubbles on both ends of the signal line.

This is illustrated in Figure_3-15. In Figure_3-15a and d, bubbles are matched for signal line aORb.H because there is no bubble on either side of the signal line. On the other hand, in

[2] The output node of the N-logic block is precharged to high and is isolated from V_{dd} in the evaluation phase. As a result, it can only be discharged to GND once in the evaluation phase.

Figure_3-15b and f, bubbles are matched for signal line aORb.L because there are bubbles on both ends. Figure_3-15c, d and Figure_3-15e, f show how bubble mismatch can be corrected by inserting inverters. This requires the inverter to have two different logic symbols. One with the bubble at the input and one at the output. This bubble matching technique is a special case of the mixed logic notation suggested in [Win80].

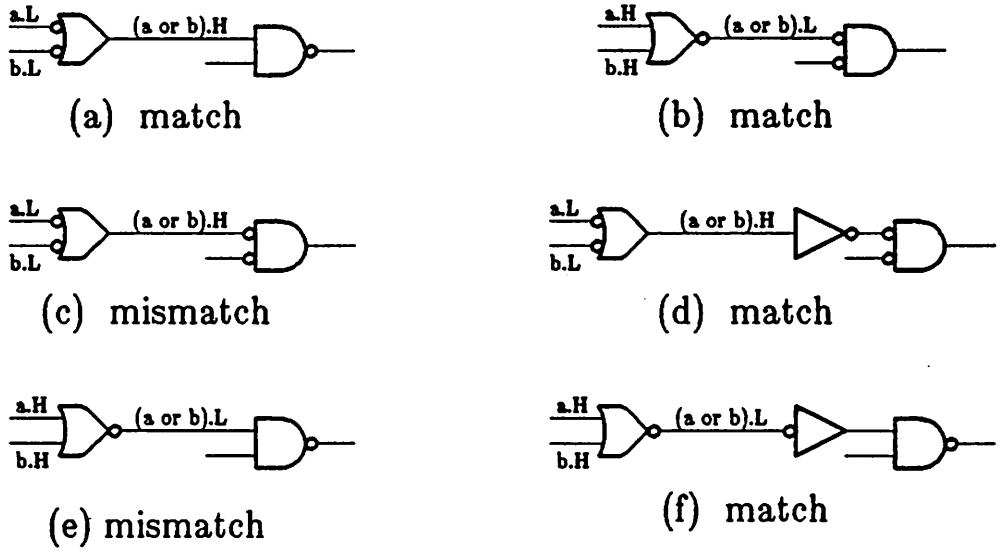


Figure 3-15 Illustration Of "Bubble Matching"

Using both N- and P-logic blocks, complementary logic can be implemented more easily than in Domino. Figure_3-16 shows how the example in Section 3.1.4 (Equation 3.1), which Domino logic has problem implementing, is implemented in one NORA stage. This function is repeated here:

$$f = x \; XOR \; y$$

where

$$x = a \; OR \; b$$

$$y = c \; OR \; d$$

In Figure_3-16, s.L means signal s is asserted whenever it is low and s.H means signal s is asserted whenever it is high. This is an improvement over the implementation in Figure_3-9b (Section 3.1.4) because the complements of a, b, c, and d are NOT required here. Since every

control line has its bubbles matched in Figure_3-16, neither rule (1) nor (2) stated earlier in this section are violated.

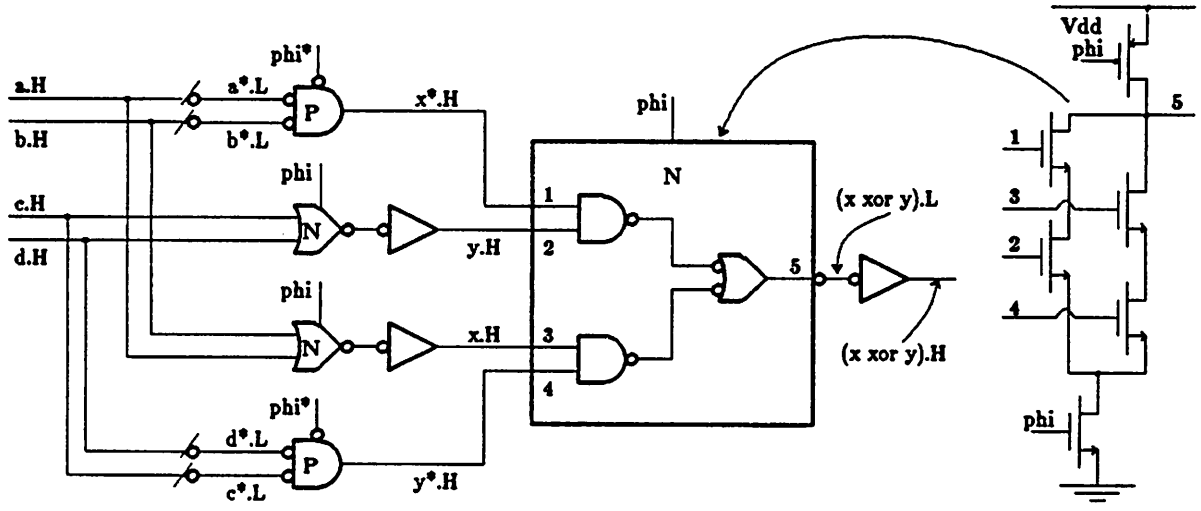


Figure 3-16 Implementation Of (a or b) xor (c or d) Using N- And P-logic Blocks

3.2.3 Alternative Clocking In NORA Logic

The NORA pipeline show in Figure_3-11 uses a single phase clock (ϕ) together with its complement ($\bar{\phi}$). However in some digital system, it may be desirable to have a multi-phase clock. For example, some microprocessors have a three-phase clock such that instruction fetch, decoding, and execution can perform in each of the three phases separately.

As far as NORA logic is concerned, the multi-phase clock does not have to be non-overlapping. Figure_3-17 shows how the original NORA pipeline (Figure_3-11) is implemented by a three-phase clock (ϕ_1, ϕ_2, ϕ_3) with non-overlap time equals to zero. This pipeline performs the function in one cycle while the NORA pipeline in Figure_3-11 performs the same function in three, but probably much shorter cycles. In general, a multi-phase clock system performs more function in one longer cycle than a single phase clock system does in one shorter cycle. One good reason for

using a longer cycle is that the NORA pipeline may be a subsystem of a bigger system and the cycle time may be limited by other subsystems. Consequently the cycle time in the single phase approach cannot be $1/m$ of the m-phase approach. Therefore the m-phase clock approach has better performance because more things are done in the same amount of time.

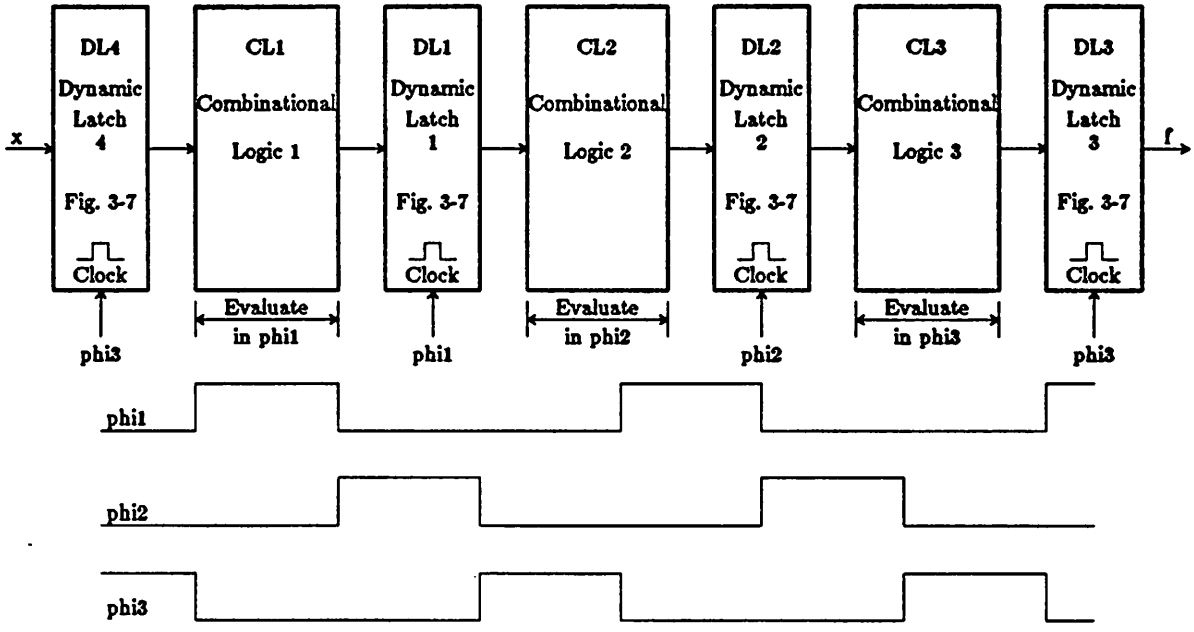


Figure 3-17 A Generic Pipeline Using A 3-phase Clock

It was stated in last paragraph that the multi-phase clock can have a non-overlap time equals to zero. As a matter of fact, different phases can even overlap. This is shown in Figure_3-18. In this figure combinational logic blocks CL1, CL2, and CL3 are assumed to have their evaluation completed before ϕ_1, ϕ_2, ϕ_3 end. Consequently, their outputs can be latched into dynamic latches DL1, DL2, and DL3 at the falling edge of $\overline{\phi_2}$, $\overline{\phi_3}$, and $\overline{\phi_1}$ respectively. One possible reason why ϕ_1, ϕ_2, ϕ_3 are not shorten here is that it may be impossible to generate phase as short as desired.

The clocking scheme used in Figure_3-11, 3-17, and 3-18 all assumed complementary logic can be implemented within each pipeline stage either by method showed in Figure_3-9b or more likely by combinations of N- and P-logic blocks. However in practice, the N-logic block is not the same as the P-logic block. For high performance, sometimes it is desirable to use N-logic blocks exclusively. Furthermore the method showed in Figure_3-9b, which implements complementary

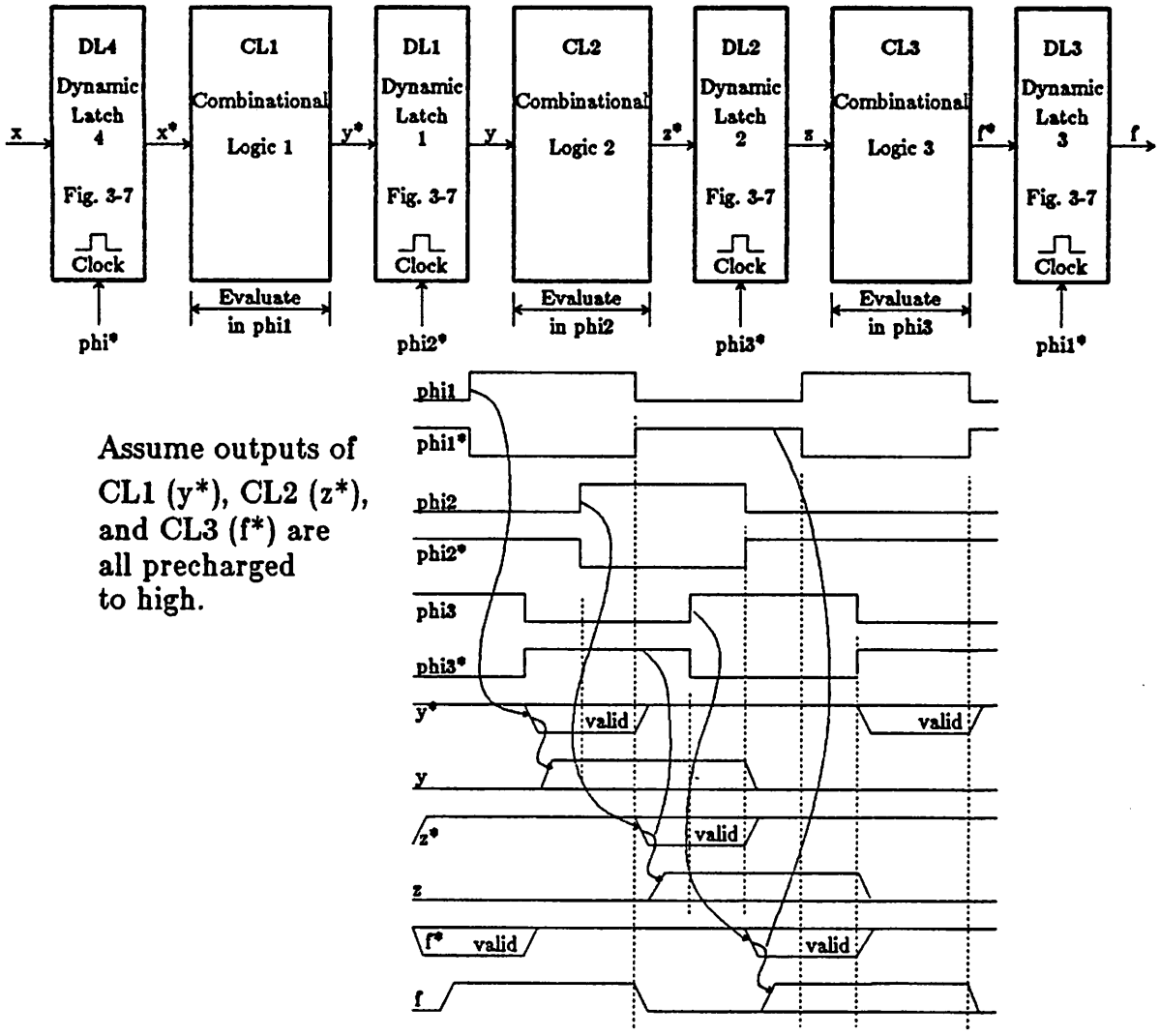
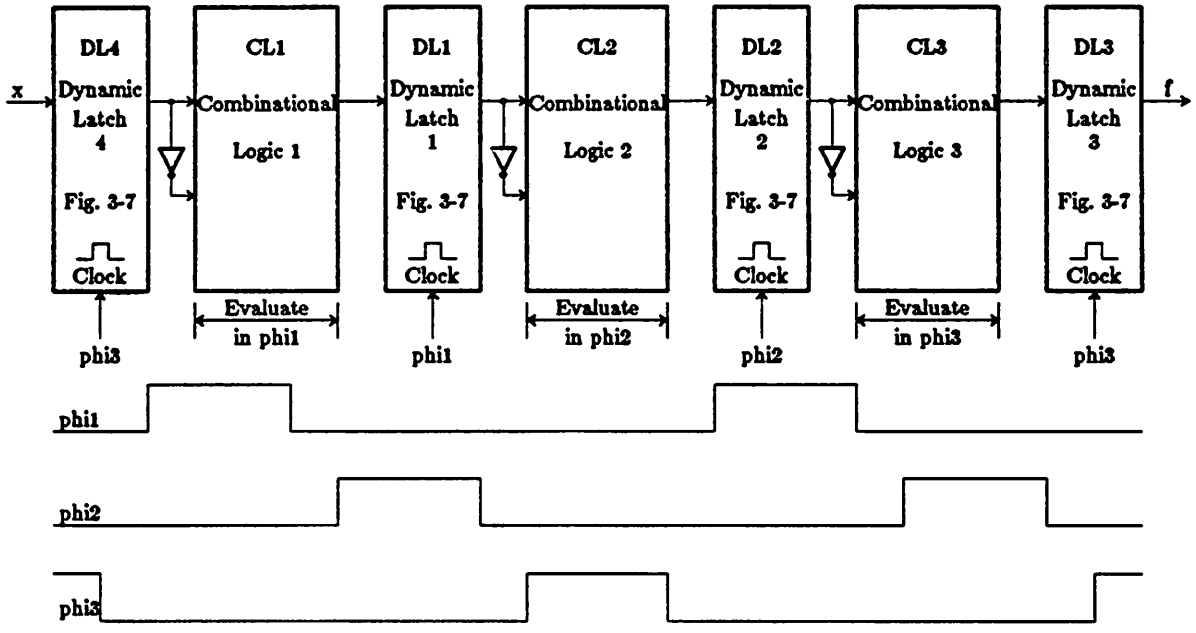


Figure 3-18 A Generic Pipeline Using A 3-phase Overlap Clock

logic by duplicating the logic, has severe area penalty. Consequently complementary logic may have to be implemented by the method shown in Figure_3-10b. The resulting pipeline thus has inverters after each dynamic latch and is shown in Figure_3-19. As illustrated in Figure_3-10b, the non-overlap time between different phases of the multi-phase clock used in Figure_3-19 must be at least bigger than the static inverter delay.

Even if no static inverter is placed between stages as in Figure_3-19, it may still be desirable to use an nominal non-overlap clock as shown in Figure_3-20. The reason is the difference in clock skew tolerance which is illustrated in Figure_3-21. The clocking scheme used in Figure_3-11, 3-17, and 3-18 all have a maximum clock skew tolerance equal to the dynamic latch set up time. On



(Please also refer to figure 3-10b)

Figure 3-19 A Generic Pipeline Using Static Inverters

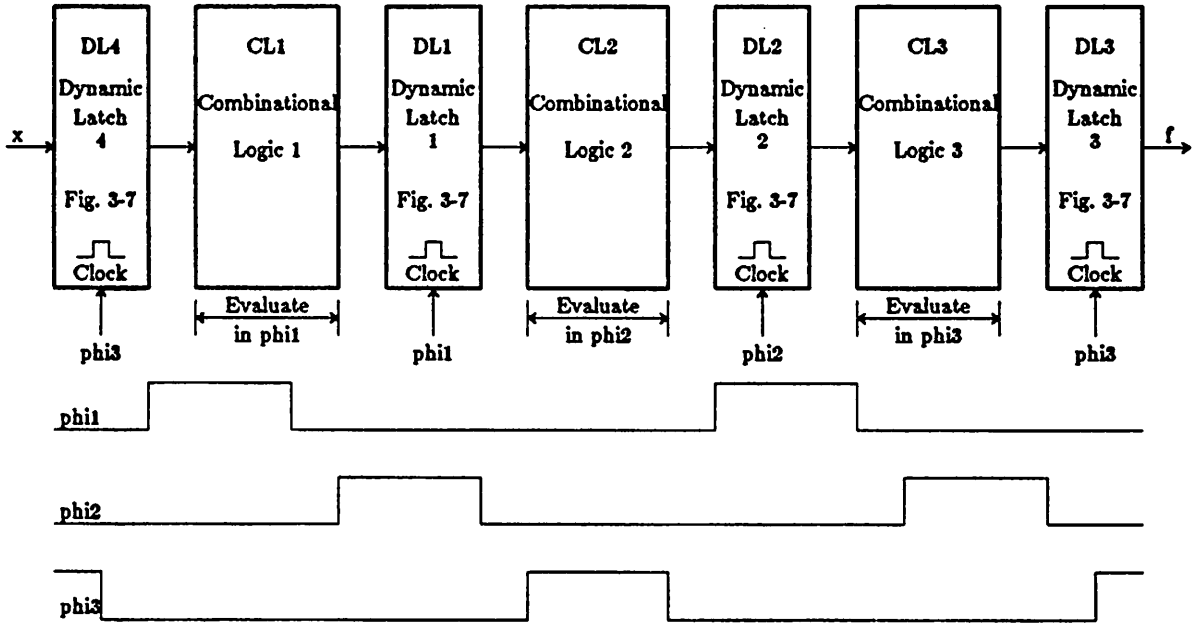
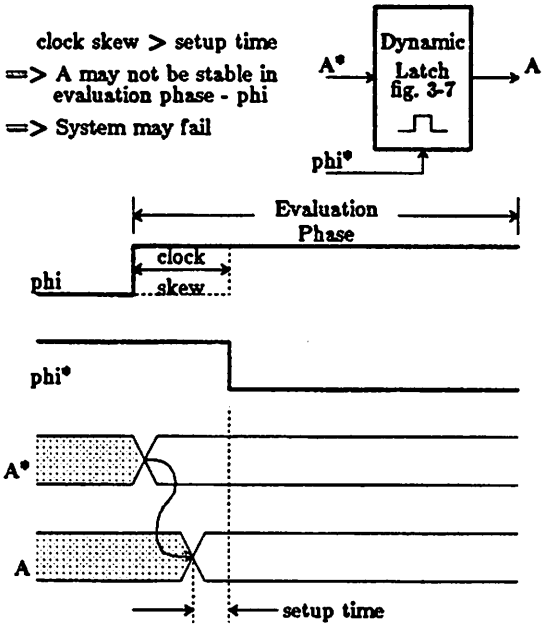
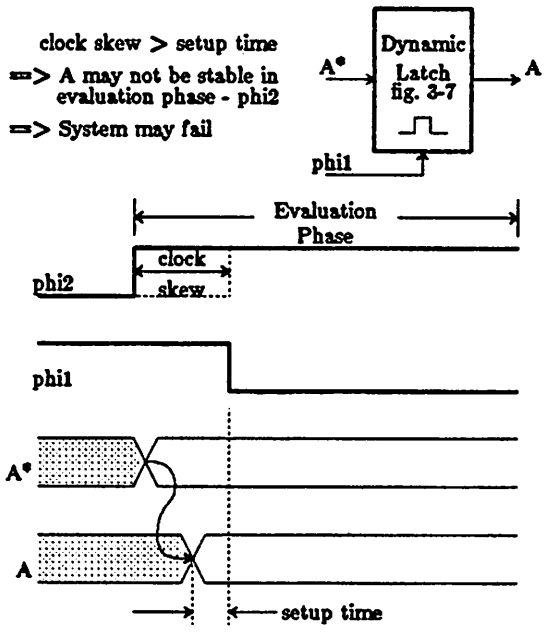


Figure 3-20 A Generic Pipeline Using A 3-phase Non-Overlap Clock

the other hand, the clocking scheme used in Figure_3-20 has a maximum clock skew tolerance equals to the dynamic latch set up time plus the nominal non-overlap time.

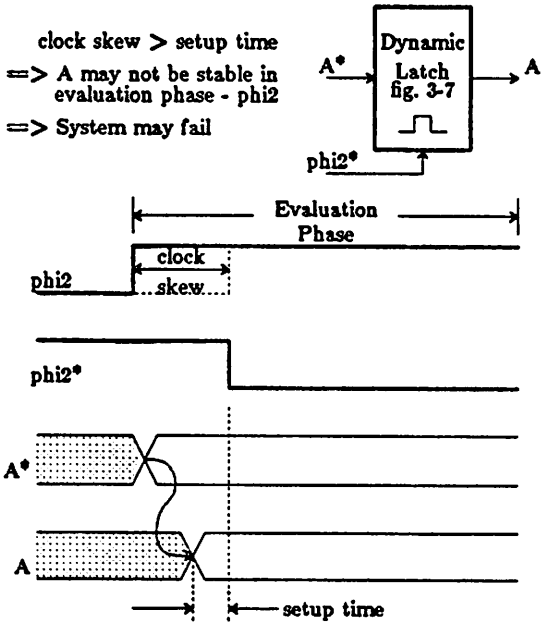


(a) Clock Skew Tolerance Of Figure 3-11

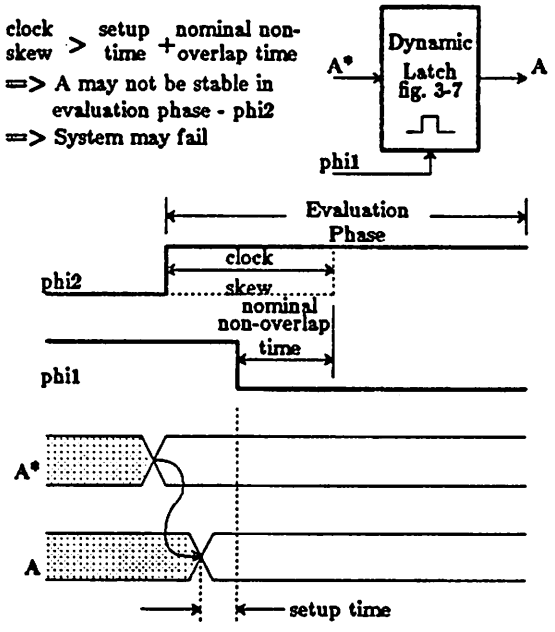


(b) Clock Skew Tolerance Of Figure 3-17

Figure 3-21ab Comparison Of Clock Skew Tolerance



(c) Clock Skew Tolerance Of Figure 3-18



(d) Clock Skew Tolerance Of Figure 3-20

Figure 3-21cd Comparison Of Clock Skew Tolerance

3.3 Charge Sharing Problem

3.3.1 The Essence Of The Problem

In dynamic logic, there is a potential of charge sharing whenever there are more than one transistors in series between the precharged node and the virtual power supply node. The virtual power supply node is the node which connects to the power supply (V_{dd} or GND) during every evaluation phase. This is shown in Figure_3-22.

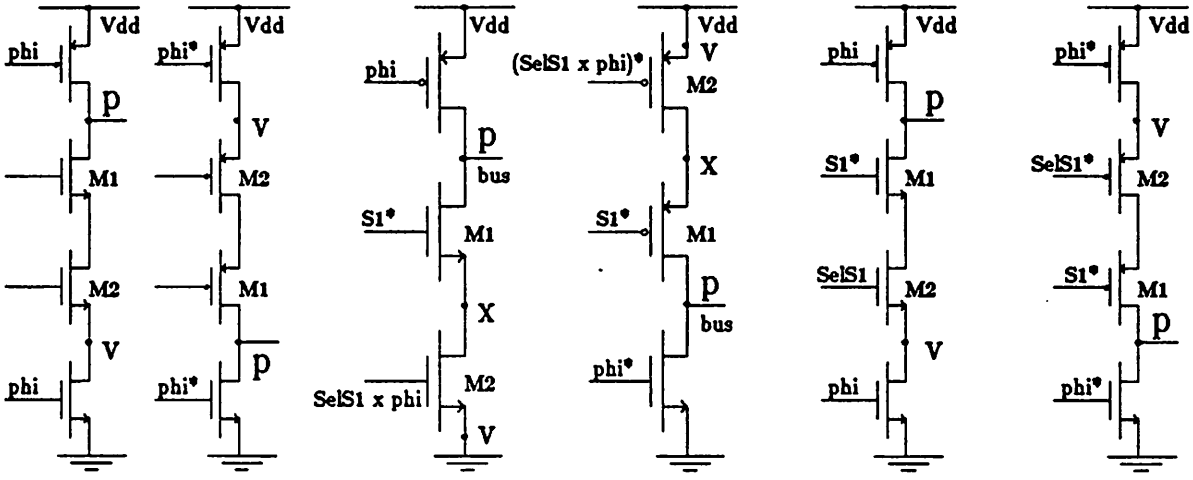
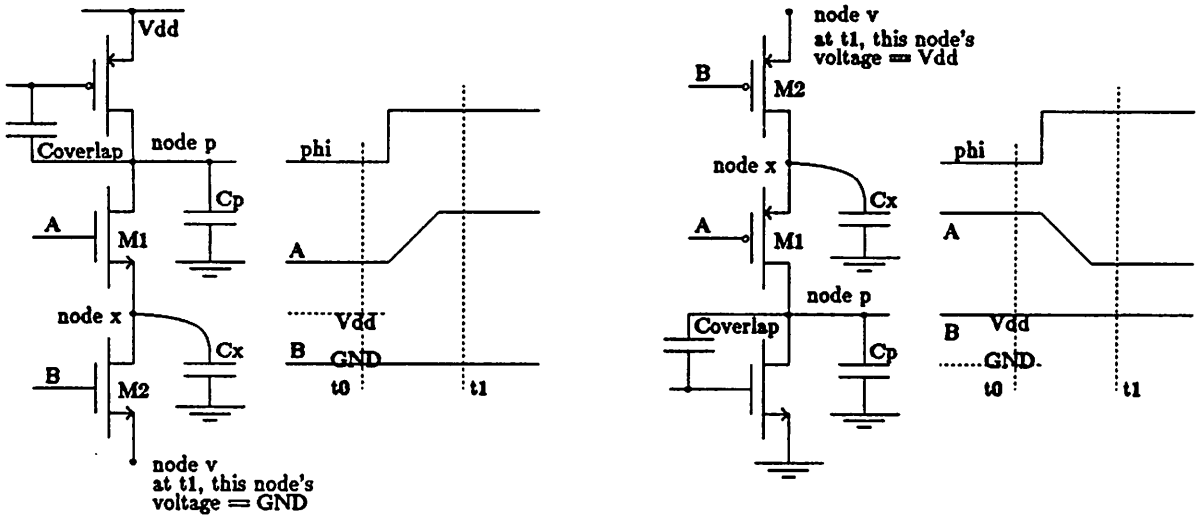


Figure 3-22 Precharged Node (p) And Virtual Supply Node (v)

In Figure_3-22, node p's are the precharge nodes and node v's are the virtual power supply nodes. In Figure_3-22c and 3-22d, the virtual power supply nodes are the same as the actual power supply because node x does not necessary connect to the power supply in every evaluation phase in either figure. Every circuit in this figure has potential charge sharing problem because there is more than one transistor (M1 and M2) in series between the precharged node (node p) and the virtual power supply node (node v).

The charge sharing problems of circuits in Figure_3-22a, 3-22c, and 3-22e are illustrated in Figure_3-23a. For the sake of generality, the generic names A and B are given to the two inputs.



(a) (b)

Figure 3-23 Charge Sharing Problems

As one can see from the timing diagram, if input B remains low but input A makes a low to high transition during the evaluation phase (after ϕ has gone to V_{dd}) then there is charge sharing between the precharged capacitor C_p and the parasitic capacitor at node x C_x. The worst case situation can be defined as follows:

$$V_x(t_0) = GND$$

and

$$C_{overlap} \ll maximum(C_p, C_x)$$

then

$$V_p(t_1) = \frac{V_{dd} \times C_p}{C_p + C_x} < V_{dd}$$

The charge sharing problem of circuits in Figure_3-22b, 3-22d, and 3-22f are illustrated in Figure_3-33b. Once again, the generic names A and B are given to the two inputs. As one can see from the timing diagram, if input B remains high but input A makes a high to low transition during the evaluation phase (after $\bar{\phi}$ has gone to GND) then there is charge sharing between the precharged capacitor C_p and the parasitic capacitor at node x C_x. The worst case situation can be

defined as follows:

$$V_z(t_0) = V_{dd}$$

and

$$C_{overlap} << maximum(C_p, C_z)$$

then

$$V_p(t_1) = \frac{V_{dd} \times C_z}{C_p + C_z} > GND$$

3.3.2 How Charge Sharing Can Be Avoided And/OR Controlled

Charge sharing problem can be prevented or at least controlled by careful layout, and/or placement of the inputs.

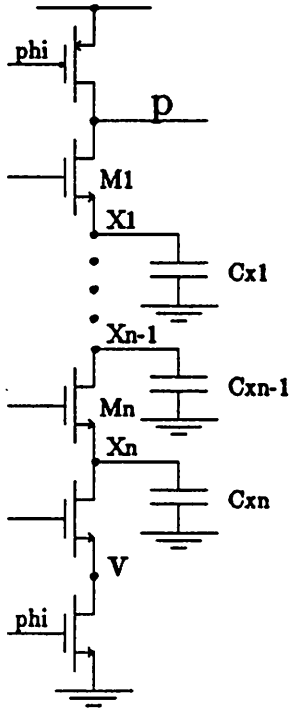
The charge sharing problem illustrated in Figure_3-23 can be avoided if the input closest to the precharge node in a series combination (in Figure_3-23, it is the input to M1's gate) is required to settle down before the end of the precharge phase (C_z can then be precharged to the same voltage as C_p) and stable throughout the evaluation phase.

The above requirement can be fulfilled easily in bus structures such as Figure_3-22c, 3-22d, 3-22e, and 3-22f. Due to the way Domino gates are connected together (see Section 3.1.2) it is impossible for Domino (and thus NORA) logic to fulfill the above requirement and the charge sharing problem has to be controlled by the following techniques:

(1) Do not use any series combination of transistors. This implies NOR (OR) gates are used exclusively, or

(2) If series combination of transistors are used, keep the number of transistors in series small such that the effective C_z is small (see Figure_3-24), and

(3) Keep the parasitic capacitor C_s small by careful layout.



Cx eff = Cx1 + Cx2 + ... + Cxn-1 + Cxn

Figure 3-24 AND Gate With Large Fan In

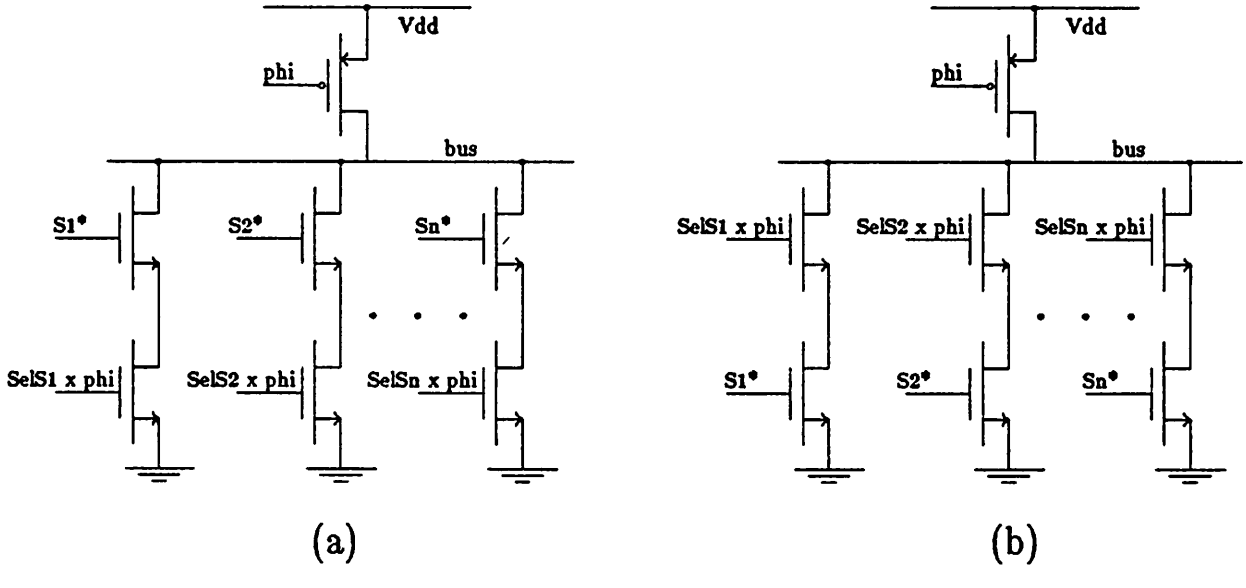


Figure 3-25 Possible Charge Sharing Problem In Bus Structure

The charge sharing problem can also be avoided by careful placement of the inputs. For example, the bus structure shown in Figure_3-25a can be free of charge sharing if signals S_1^*, S_2^*,

... S_m^* are required to settle down before the end of the precharge phase (ϕ=GND). On the other hand, there is no way the charge sharing problem can be avoided for the bus structure shown in Figure_3-25b. One of the $SelS_m$ signals will go high when ϕ=V_{dd} and if its corresponding S_m^* is low, then busD will end up with a voltage lower than V_{dd} due to charge sharing. Figure_3-26 is another example. In Figure_3-26a, charge sharing will degrade the output voltage at node y if input x changes value after ϕ has gone low (ϕ=GND and $\bar{\phi}$=V_{dd}). On the other hand, the conventional C^2MOS latch, which is shown in Figure_3-26b, does not have this problem because as soon as ϕ goes low, the M2 and M3 isolate the output node (node y) from the input.

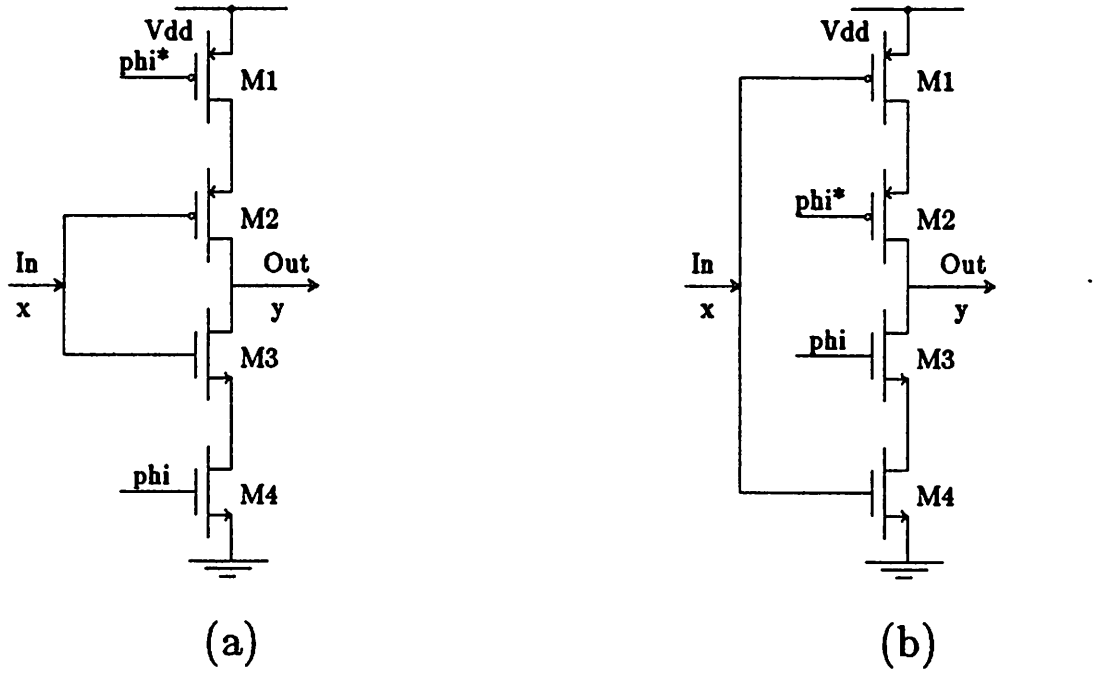


Figure 3-26 Possible Charge Sharing Problem In CCMOS Latch

3.3.3 Dynamic Latch

The two ways to implement a dynamic latch are shown in Figure_3-7 and are discussed briefly in Section 3.1.3. In this section, the difference in their electrical behavior will be discussed.

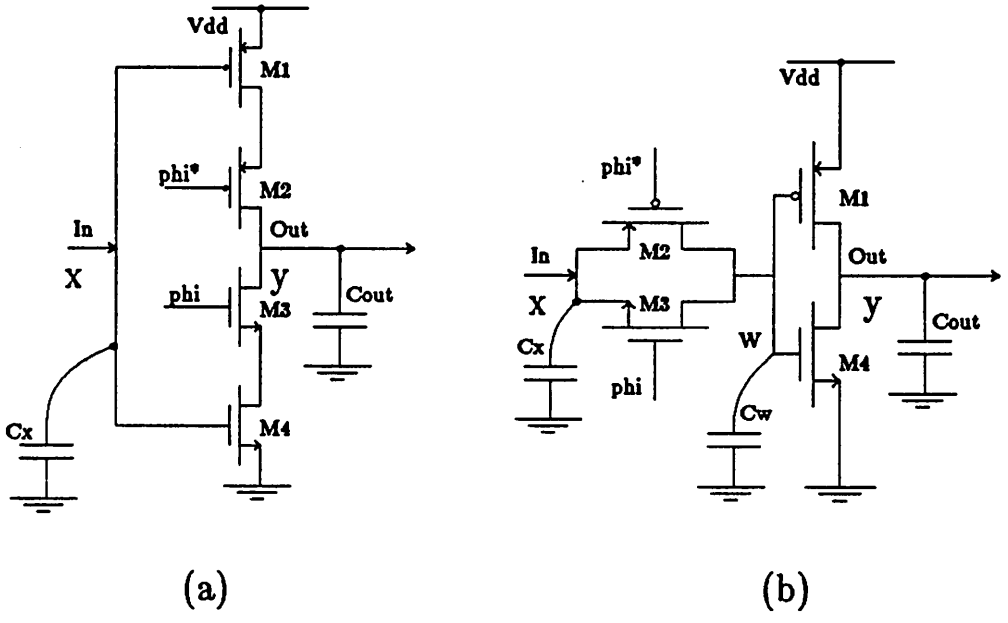


Figure 3-27 Charge Sharing Problem In Dynamic Latch

The C^2MOS latch is shown in Figure_3-27a and the other approach, which uses a composite pass gate followed by an inverter, is shown in Figure_3-27b. Notice that if node x is a precharged node, then as soon as $\phi=V_{dd}$, charge sharing will occur between C_x and the capacitor at node w C_w. Therefore if node x is a precharged node, C^2MOS latch must be used.

However the C^2MOS latch does have its disadvantage. The clock $\phi=V_{dd}$ must be long enough to do two things:

(1) Charge or discharge the capacitance C_x at node x, which consists mainly of the gate capacitance of M1 and M4; then

(2) Charge or discharge the output capacitance C_{out} at node y .

Capacitor C_{out} can be big and its charge and discharge paths consist of two transistors in series (M1, M2 and M3, M4) which make the charge and discharge time even longer unless M1, M2, M3, and M4 are big. However making M1 and M4 bigger will increase C_x and therefore worsen the first part of the problem (see (1) above).

On the other hand, the pass gate followed by an inverter approach only requires $\phi=V_{dd}$ be long enough to charge or discharge the capacitor at node w C_w. C_w is not likely to be big because it consists mainly of the gate capacitance of M1 and M4. M1 and M4 here are likely to be smaller than the M1 and M4 of the C^2MOS latch because they are connected to the output node directly here while in the C^2MOS they are connected in series with M2 and M3. Therefore if node x is not a precharged node, it is preferable to use a pass gate followed by an inverter to implement a dynamic latch.

4. A CMOS 32-BIT ALU - A DESIGN EXAMPLE

This 32-bit arithmetic and logic unit (ALU) is intended to be used in a 32-bit data-path which operates in a four-phase non-overlap clock. The ALU itself operates on the last three phases (ϕ_2, ϕ_3, and ϕ_4) of this four-phase clock. As shown in Figure_4-1, this 32-bit ALU is formed by cascading four 8-bit ALU. The ALU can perform five different operations which are (1) add (A + B + Cin), (2) subtract (A + B* + Cin), (3) and, (4) or, and (5) xor.

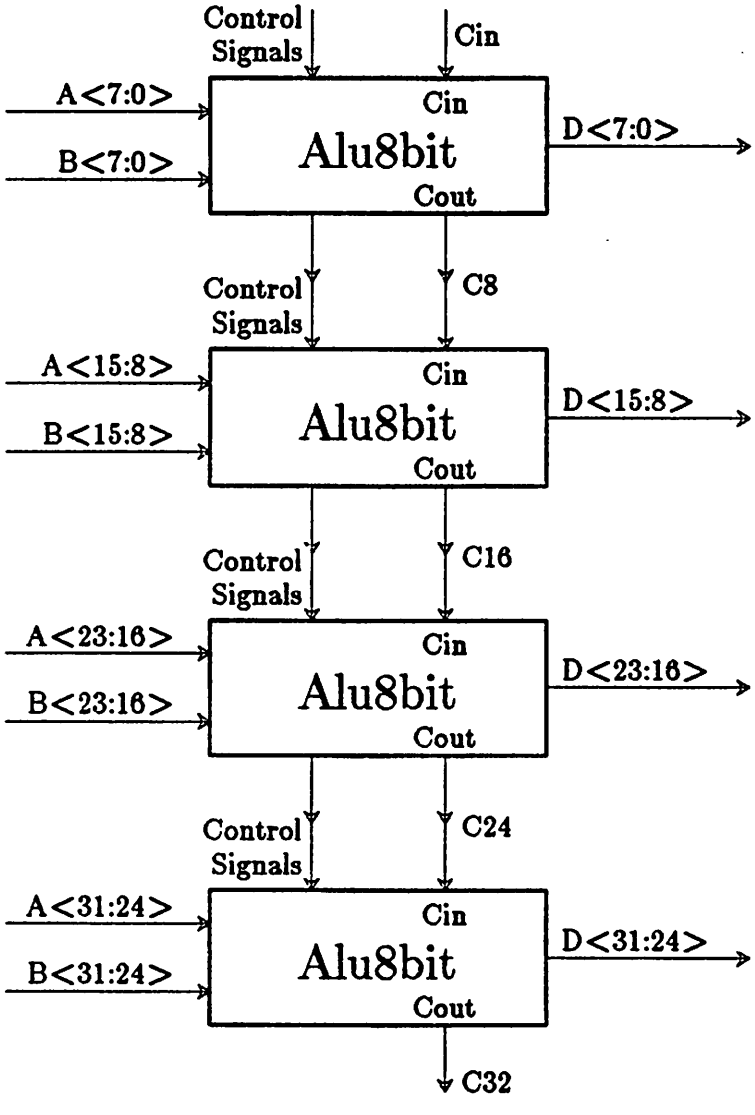


Figure 4-1 32-bit Arithmetic And Logic Unit

The process to be used is a 3 micron CMOS with two layers of metal, metal1 and metal2. However, only the first layer of metal, metal1, can make contacts to both polysilicon and

diffusion. The second layer of metal, metal2, can only make contact to metal1. Furthermore, buried contact is not supported by this process.

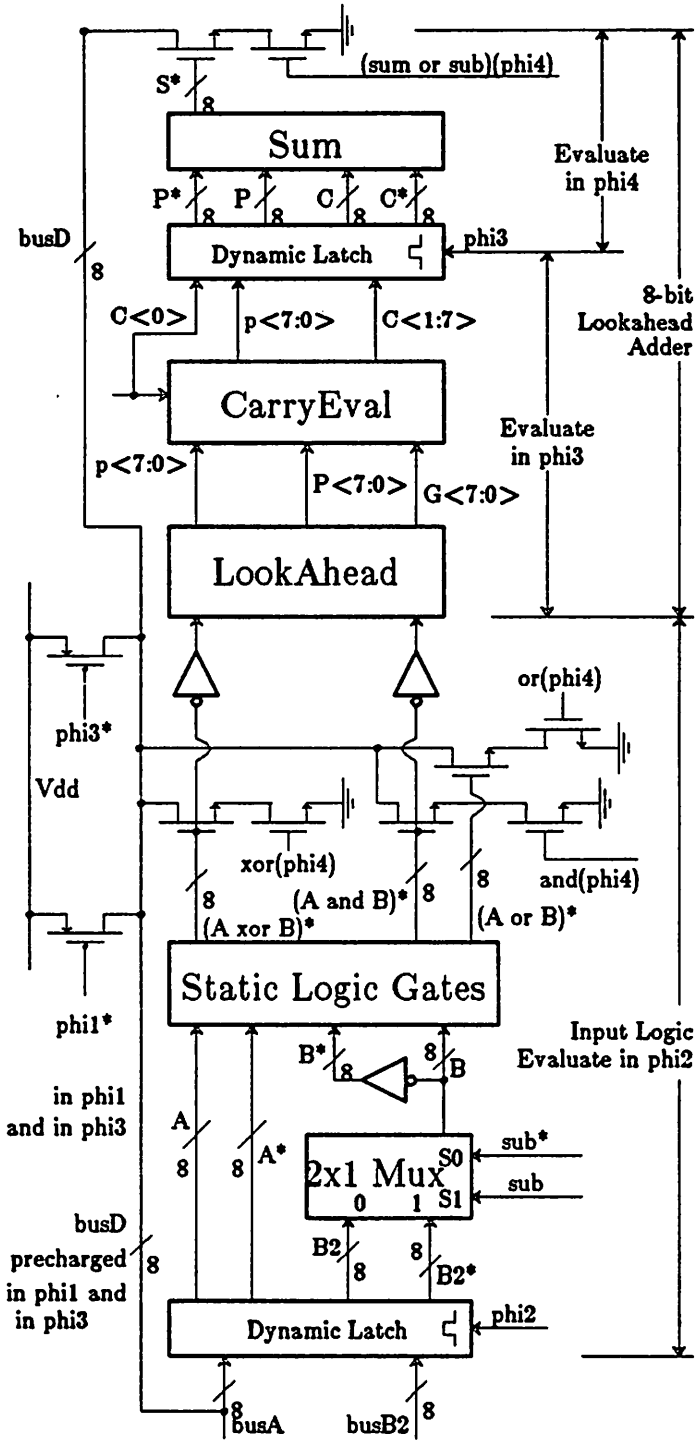


Figure 4-2 8-bit Arithmetic And Logic Unit

The internal structure of the 8-bit ALU is shown in Figure_4-2. Each 8-bit ALU consists of a Input Logic section, followed by an 8-bit look-ahead adder. There are two reasons why full 32-bit look-ahead adder is not used:

(1) A 32-bit look-ahead adder will require much more area, especially in the horizontal dimension, which is very critical in the overall data-path. From [SW84], it is clear that it will take much more than 500λ (1λ=1.5μ => 750μ) to implement a 32-bit look-ahead circuit. The ALU implemented here only has a horizontal dimension of 450λ.

(2) Due to the distribution of tasks over different clock phases along the data-path (see Section 4.1), an 8-bit look-ahead adder is fast enough. This means in each of the three phases the ALU operates, the ALU can complete what it is supposed to do in that phase faster than other parts in the data-path can complete their tasks. This clearly illustrates the difference between local optimization and global optimization.

4.1 Distribution of Task

The distribution of task is illustrated in Figure_4-2.

ϕ_2 Inputs from busA and busB2 are latched. Logic functions A xor B (same as p, the carry propagate signal), A and B (same as g, the carry generate signal), and A or B are evaluated for each bit.

ϕ_3 The p's and g's evaluated in ϕ_2 should be settled by ϕ_3 and they are used by the LookAhead block to evaluated the composite P's and G's (see Section 4.2 for details). The CarryEval block then uses the P's and G's to calculated the carry output for each bit.

ϕ_4 The p's together with the carry out of each bit is used by the Sum block to evaluate the sum. The result is put onto busD if it is an ADD or SUB operation. Otherwise the

results from ϕ_2 (A xor B, A and B, and A or B) is put onto the busD depending on what ALU operation is selected.

The most critical and complicated part of the ALU is the 8-bit look-ahead adder. It is described in detail in the next section.

4.2 8-bit Look-ahead Adder

The 8-bit look-ahead adder consists of three parts. The LookAhead, the CarryEval and Sum (see Figure 4-2).

4.2.1 LookAhead

The LookAhead is based on the signal flow graph shown in Figure_4-3a. The inputs of this graph are the carry propagate (p) and generate (g) signals for each bit and the outputs are the composite propagate (P) and generate (G) signals over i bits.

$$p_i = A_i \; xor \; B_i$$

$$g_i = A_i \cdot B_i$$

$$P_i = p_0 \cdot p_1 \cdot \cdots \cdot p_i$$

$$G_i = g_i + g_{i-1} \cdot p_i + g_{i-2} \cdot p_i \cdot p_{i-1} + \cdots + g_0 \cdot p_1 \cdot p_2 \cdots \cdot p_7$$

where

$$i = 0, 1, 2, 3, ..., 7$$

Each circle in the graph shown in Figure_4-3a represents a functional node. To keep layout more uniform and avoid long routing wire, buffers (triangles) are added whenever functional node is not necessary. Each functional node performs the logic function shown in Figure_4-3b. Its implementation is shown Figure_4-3c and Figure_4-3d. Figure_4-3d is the transistor diagram but

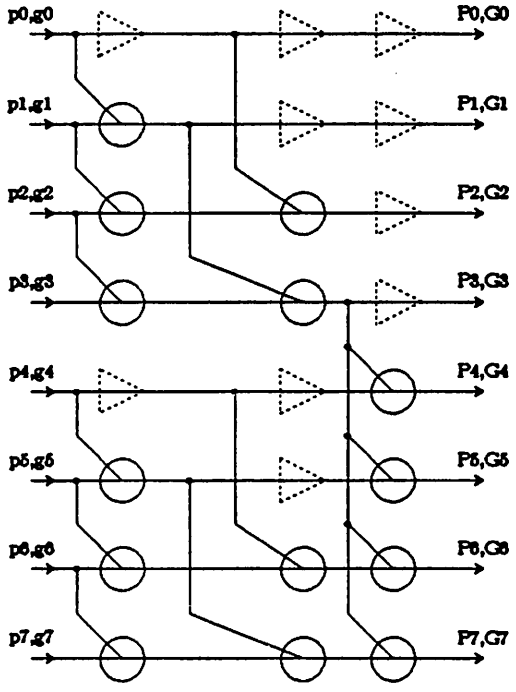
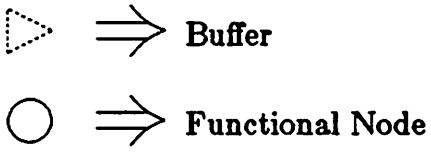


Figure 4-3a Signal Flow Graph

pi = Ai xor Bi gi = Ai and Bi

Pi = p0 and p1 and ... pi

Gi = gi or (gi-1 and pi)
or (gi-2 and pi and pi-1)
or ... (g0 and p1 and p2 and ... p7)



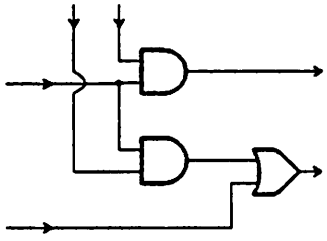


Figure 4-3b Functional Node

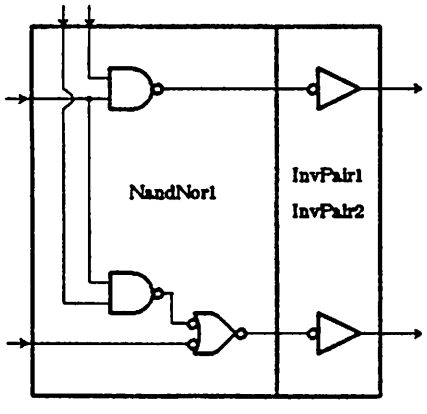


Figure 4-3c Implementation In Domino Logic

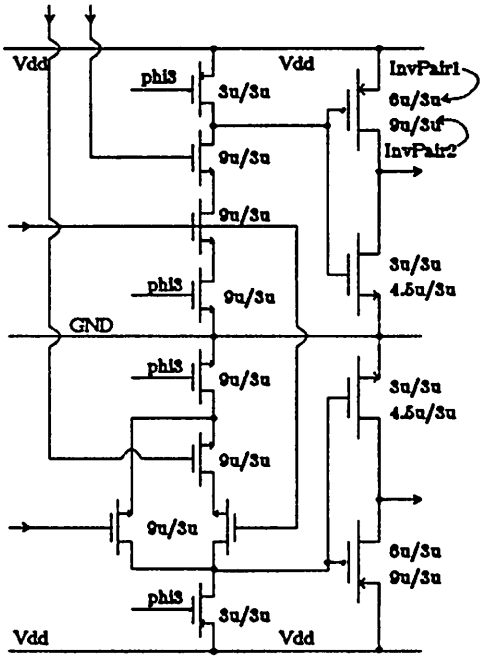


Figure 4-3d Transistor Diagram

it also represents the topology of the circuit. There are three things worth noticing:

(1) The basic topology is the same as the register cell of the register file to be used in the data-path. Namely V_{dd} on top, GND in the middle, and V_{dd} on the bottom. As a reference, the circuit diagram of the register cell is shown in Figure_4-3e. Using this structure keep the ALU pitch match with the register file. Furthermore, using this structure also enable vertical space to be traded in for horizontal space such that the complete ALU has a horizontal dimension less than 500λ (750 micron).

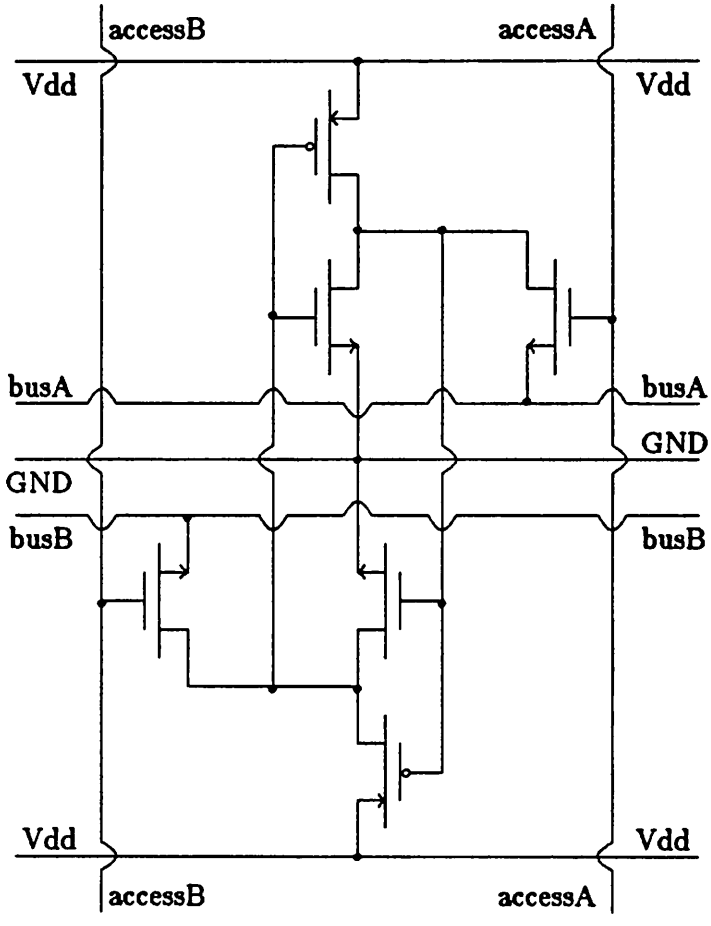


Figure 4-3e Register Cell

(2) Only N-logic blocks are used because NMOS transistors are much faster than PMOS transistors, especially when they have to be connected in series. Furthermore, using N-logic gates exclusively require inverting buffers between every gate. By changing the size of these buffers, the critical path of the circuit can be fine tuned relatively easily (see paragraph below).

(3) V_{dd} and GND, which run horizontally, and all other horizontal connections will be routed in metal2. Control lines, which run vertically, and all other vertical connections will be routed in metal1. Using metal1 to route the control lines makes them much easier to drive because of the lower resistance and capacitance of metal1 relative to polysilicon.

The overall structure of the LookAhead block, which is derived from the signal flow graph Figure_4-3a, is shown in Figure_4-4a. From this, the floor plan of the LookAhead block is derived in Figure_4-4b. This Figure shows that the LookAhead circuit is built from three basic cells, which are NandNor1, InvPair1, and InvPair2, only. InvPair1 and InvPair2 have the same function except InvPair2 has bigger transistor size to drive higher fan out. The logic and transistor diagrams are in Figure_4-3c and 4-3d respectively. Layouts of these basic cells are plotted in Figure_4-5, 4-6, and 4-7 respectively.

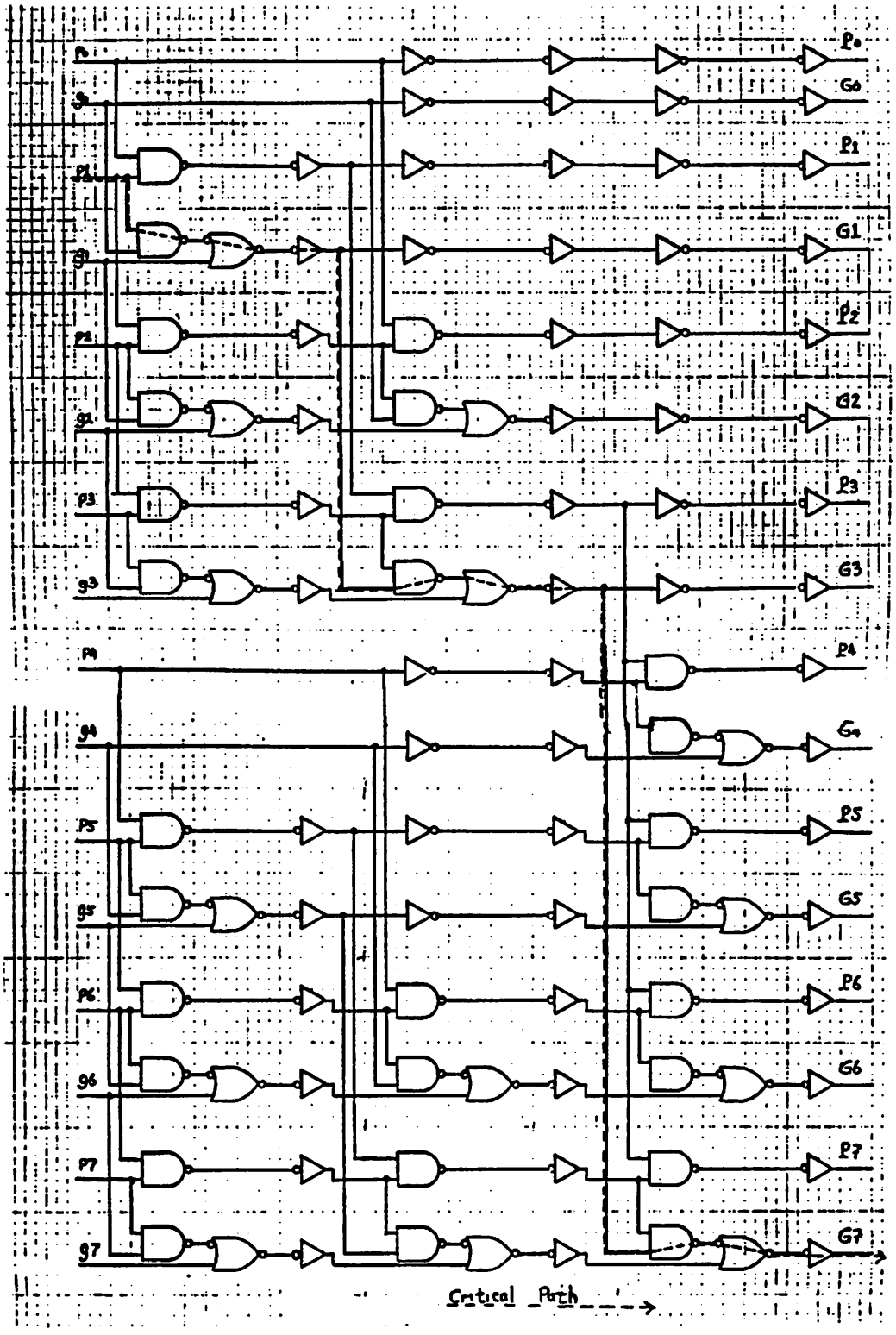
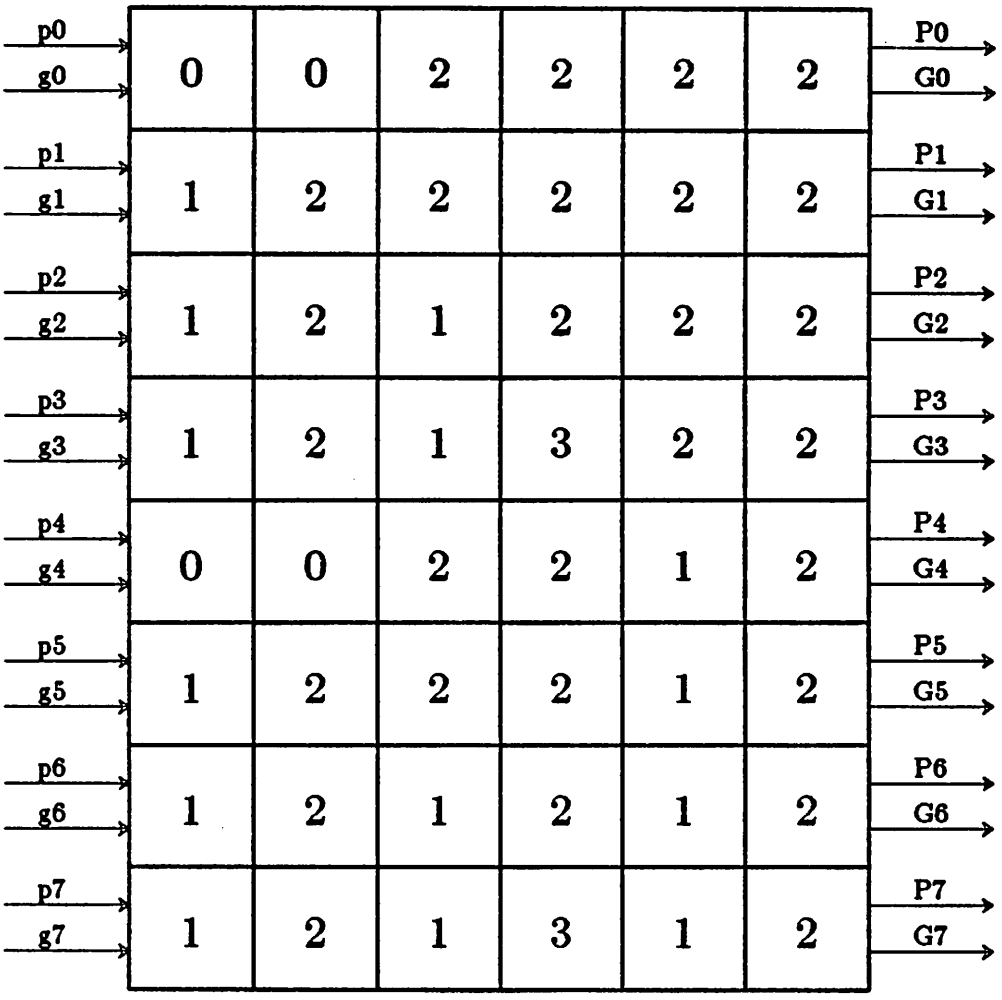


Figure 4-4a Logic Diagram Of LookAhead Block



0 - Space Not Occupied 1 - NandNor1
2 - InvPair1 3 - InvPair2

Figure 4-4b LookAhead's Floor Plan

4.2.2 CarryEval

The CarryEval block is much simplier compared to the LookAhead block. It only consists of two basic building blocks, NandNor2 and NandNor3. Both are Domino gates which evaluate carry out C_{i+1} from P_i, G_i and Cin. Their circuit diagrams are shown in Figure_4-8a and Figure_4-8b. The layouts are plotted in Figure_4-9a and Figure_4-9b respectively.

NandNor2 is used for every bit except bit7, 15, 23, and 31 (ie. the MSB of the four 8-bit ALU's). Notice that NandNOR3 is very similar to NandNor2 except it has an extra driver to

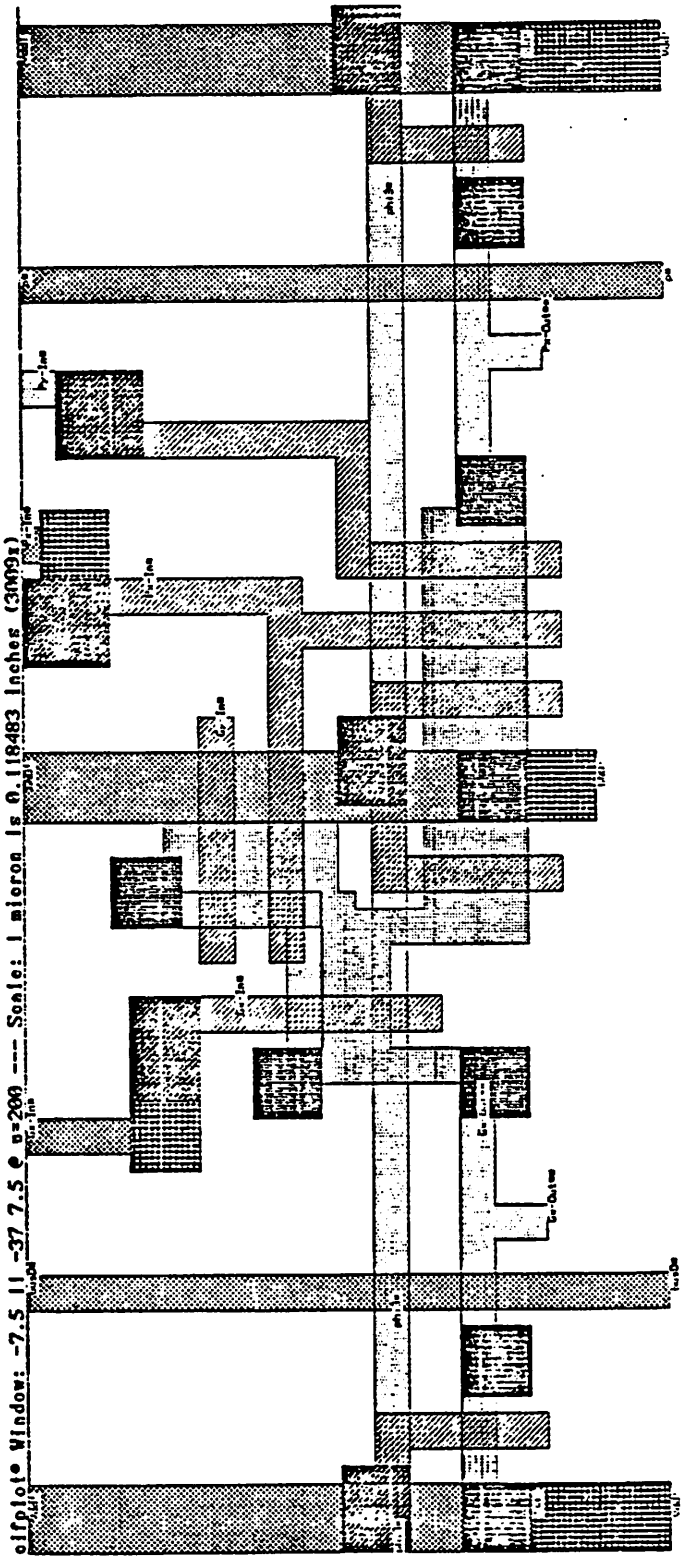


Figure 4-5 Layout Of NandNor1

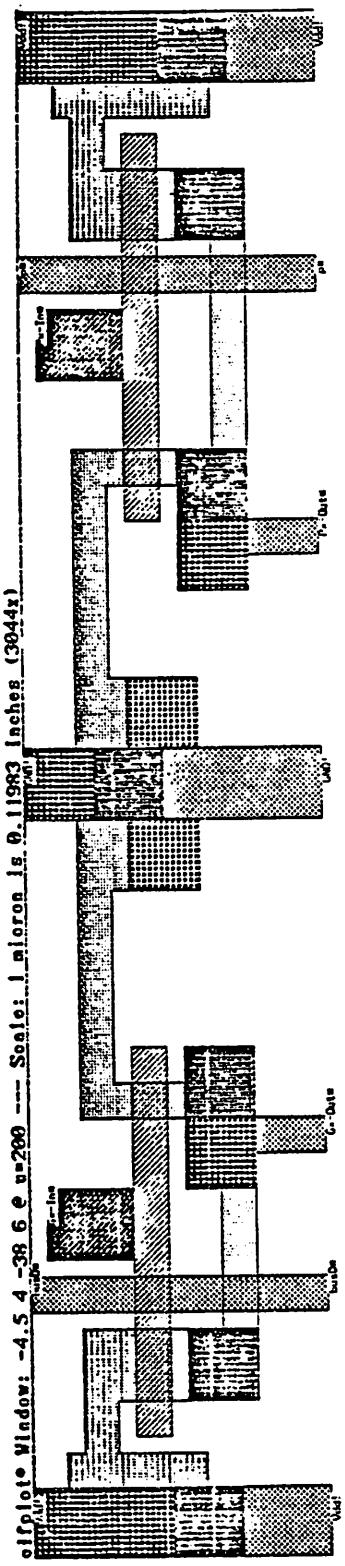


Figure 4-6 Layout Of InvPair1

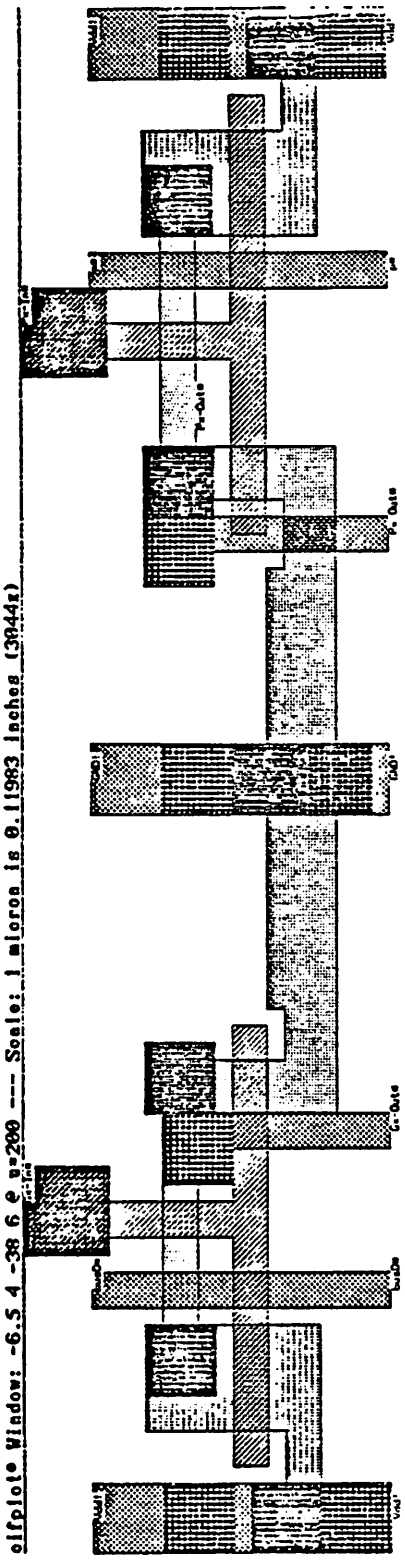


Figure 4-7 Layout Of InvPair2

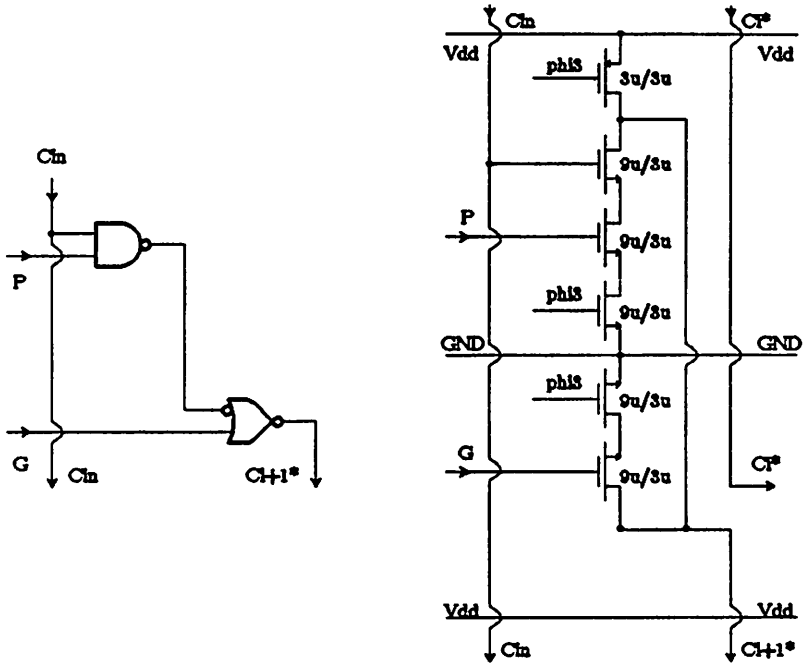


Figure 4-8a NandNor2's Circuit Diagram

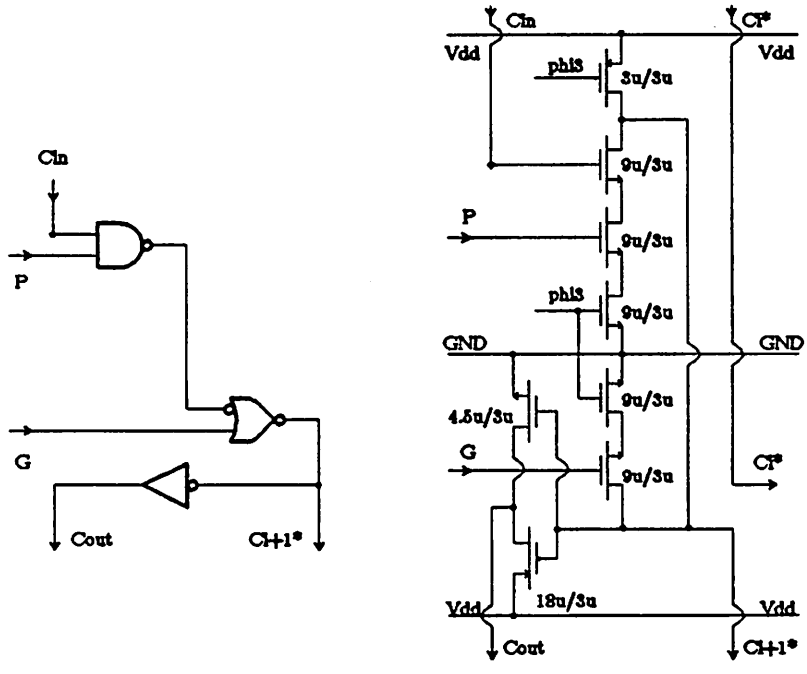


Figure 4-8b NandNor3's Circuit Diagram

drive the Cin input of the next 8 bits. This driver is big because it has a large fan out (fan out = 8) and the Cin line is long because it has to route through eight bit. Fortunately this vertical routing can be done in metal1 thus parasitic capacitance and resistance is minimized.

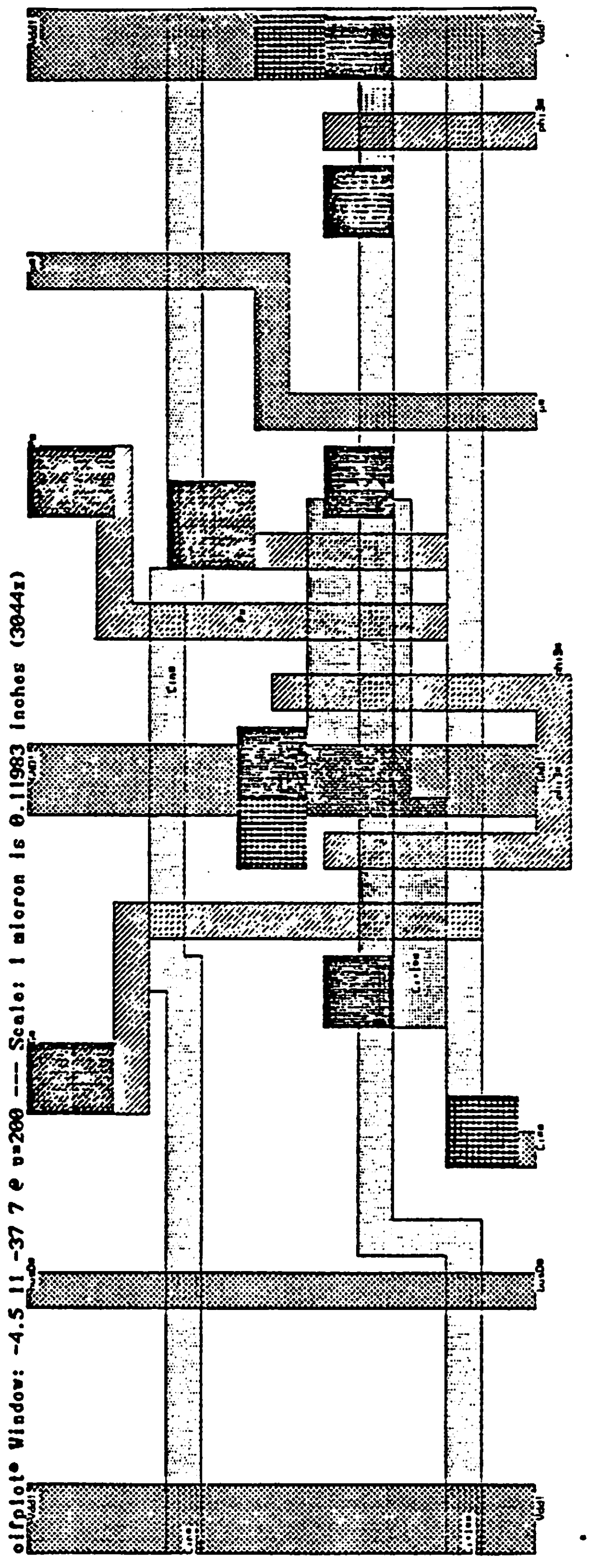


Figure 4-9a Layout Of NandNor2

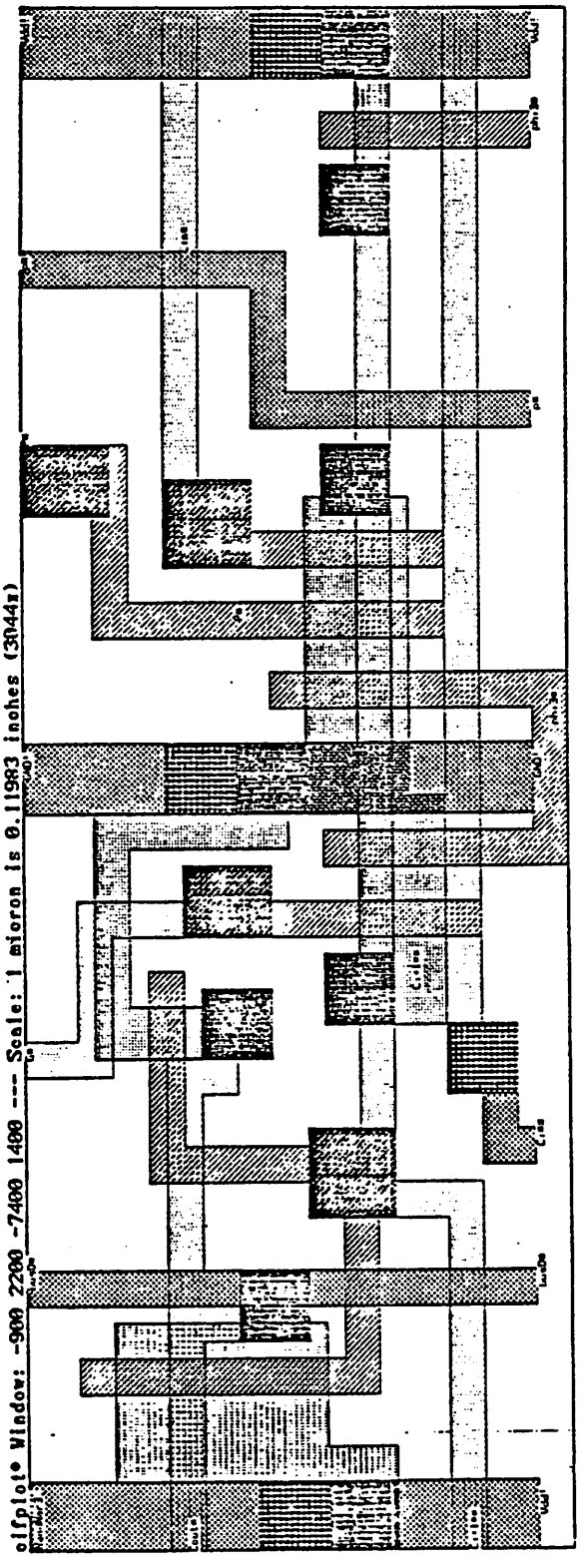


Figure 4-9b Layout Of NandNor3

In actual layout, dynamic latches and inverters are integrated into the CarryEval block. The circuit diagrams for one CarryEval bit are shown in Figure_4-10a and Figure_4-10b. Figure_4-10b is the MSB of the 8-bit adder and therefore it contains NandNor3 instead of NandNor2. The layouts of these two bit slices are plotted in Figure_4-11a and Figure_4-11b respectively.

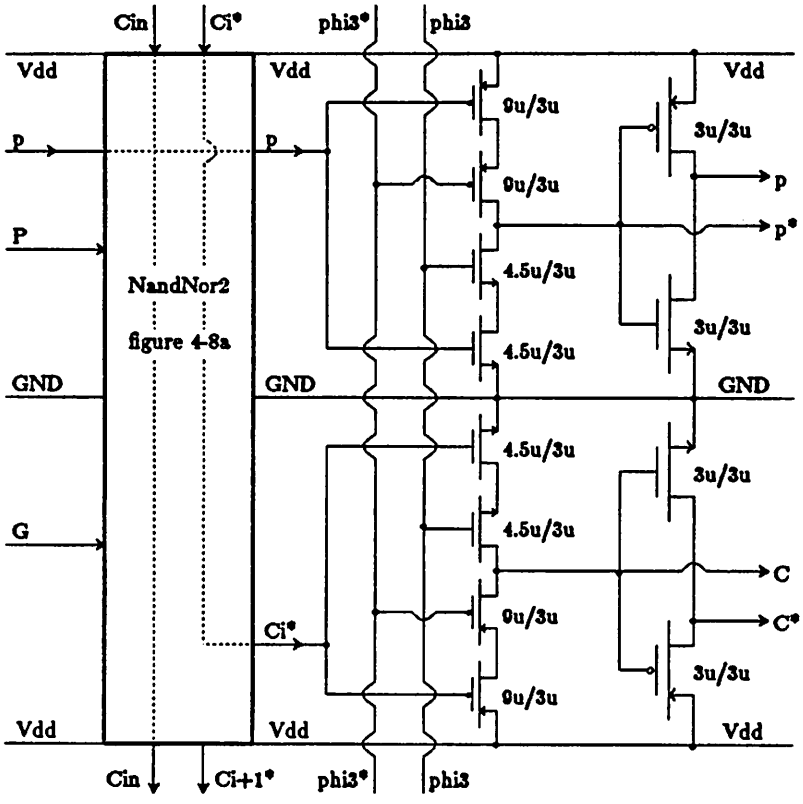


Figure 4-10a C_EvalCell1's Circuit Diagram

The function of the latches are to latch in $A_i xor B_i = p_i$ and C_i (C_{i+1} from the previous bit) at the end of ϕ_3. These latches and inverters then provide stable p_i C_i and their compliments for the Sum block during ϕ_4. Notice that C_i is latched instead of C_{i+1} because the Sum block evaluates $S_i = C_i \; xor \; p_i$ instead of $S_i = C_{i+1} \; xor \; p_i$. By doing the routing of C_i's here, which is more natural, no routing is needed in the CarryEval and Sum block interface.

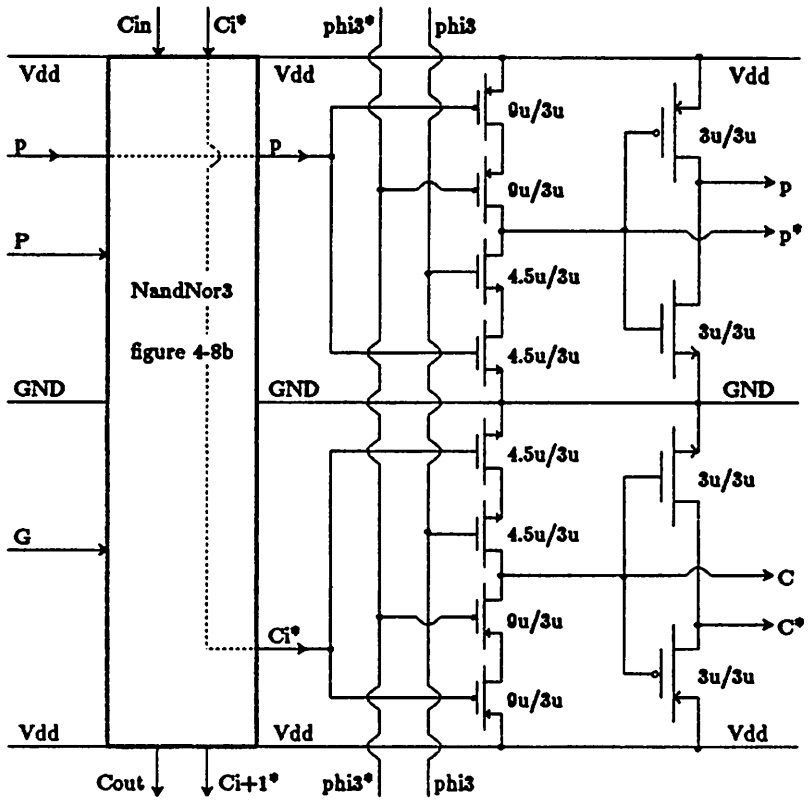


Figure 4-10b C_EvalCell2's Circuit Diagram

4.2.3 Sum

The Sum block is also relatively simple. It consists of an exclusive-or (xor) gate and some interface logic to busD. The circuit diagram for one bit of Sum is shown in Figure_4-12. The layout is plotted in Figure_4-13.

This circuit simply implements a Domino xor gate with busD as the output node. However the value cannot put onto busD unless it is either a add or subtract operation. Therefore to ensure functional correctness, control signal addORsub_ϕ_4 must be stable in ϕ_4. Furthermore busD has to be precharged in ϕ_1 (for ALU input) and in $\phi 3$ (for ALU output). This is done by the PMOS transistor with signal ϕ_1*ANDϕ_3* at its gate. Precharged has to be done in both ϕ_1 and ϕ_3, therefore if a NMOS transistor is used, the control signal is ϕ_1ORϕ_3. But a PMOS transistor is used here, therefore the signal has to be inverted: $(\phi_1 \text{OR} \phi_3)^* = \phi_1{}^* \text{AND} \phi_3{}^*$

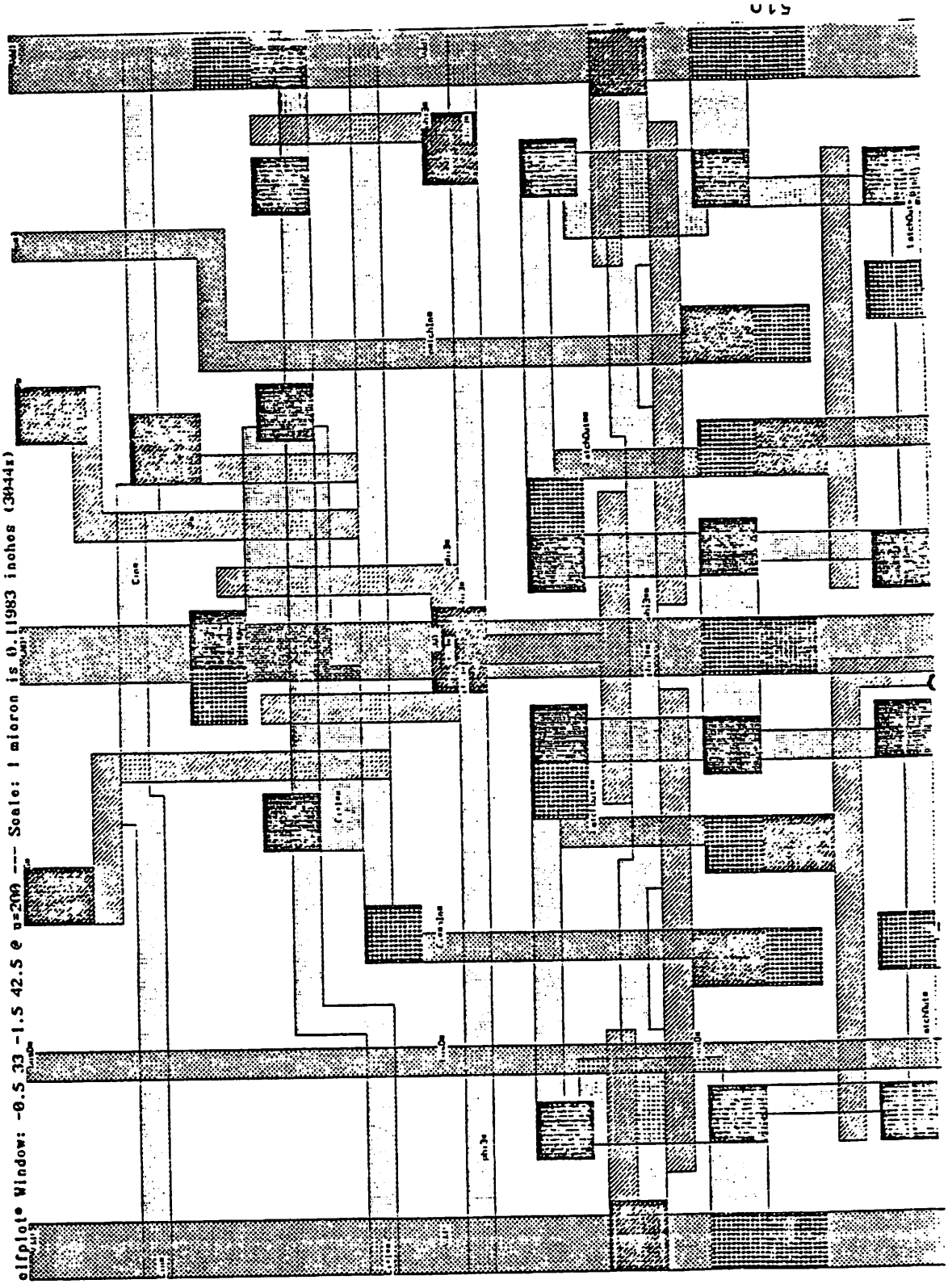


Figure 4-11a Layout Of C_EvalCell1

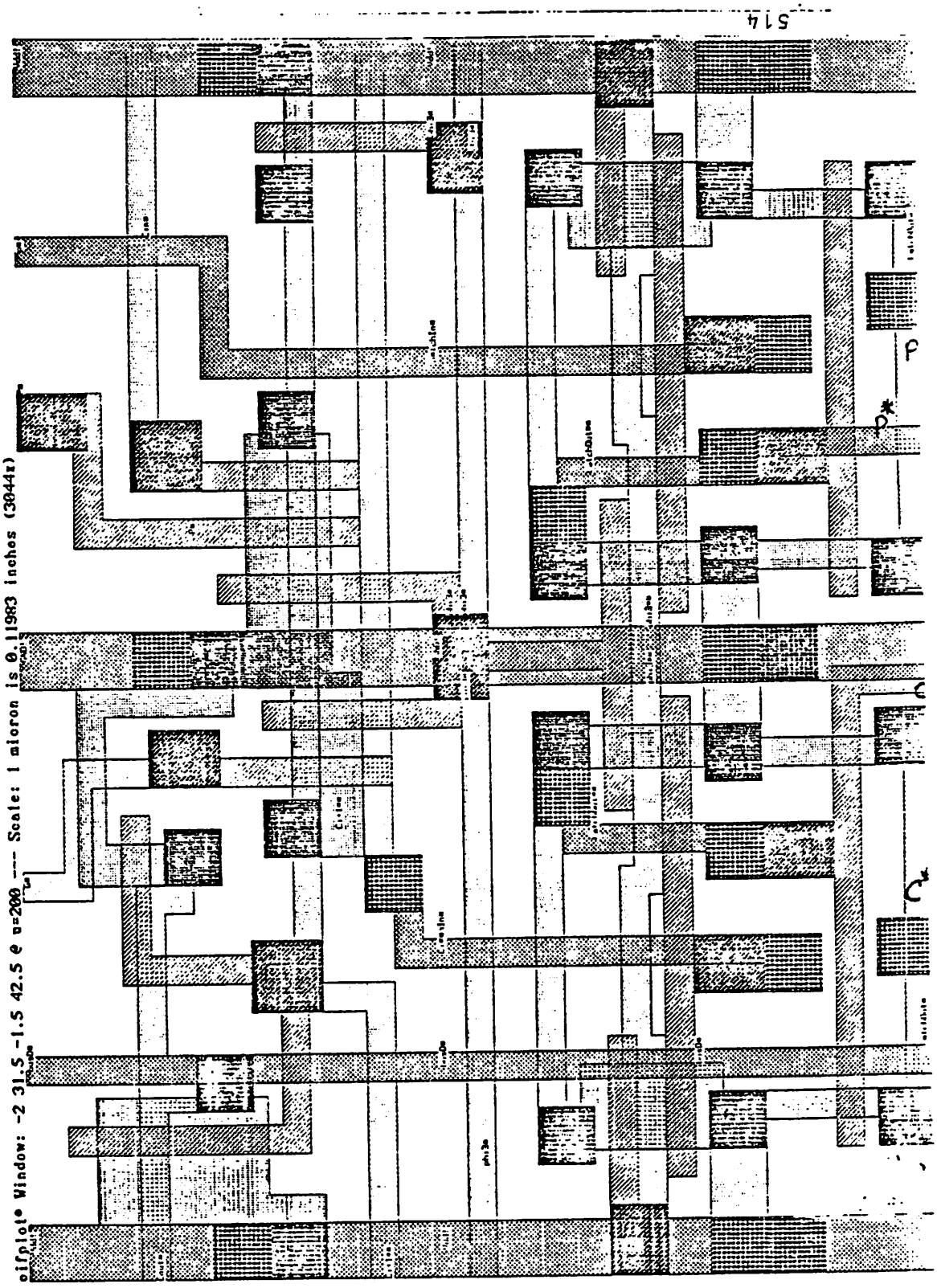


Figure 4-11b Layout Of C_EvalCell2

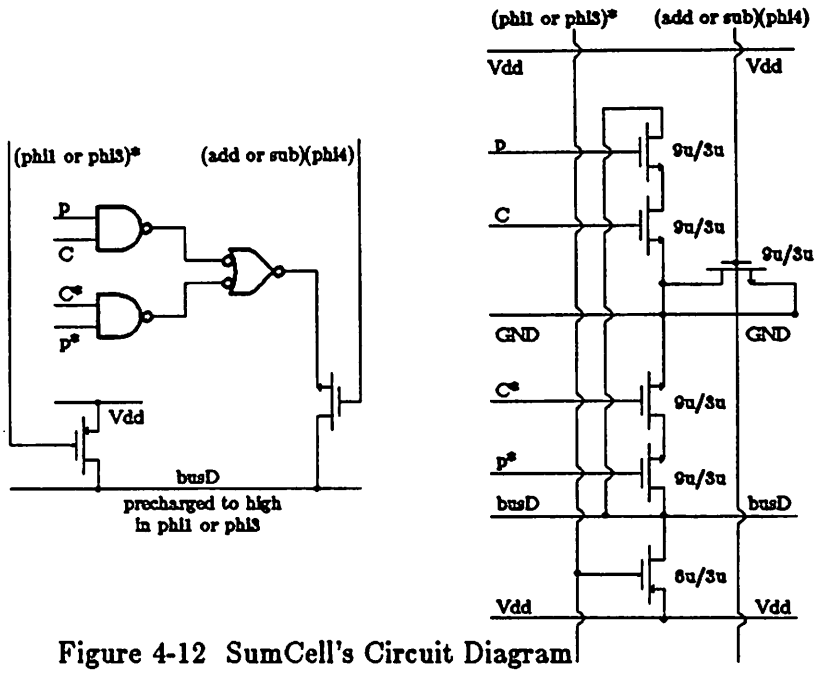


Figure 4-12 SumCell's Circuit Diagram

4.3 Input Logic

The Input Logic section consists dynamic latches, inverters, a 2x1 Mux, three static NAND gates and some busD interface circuitry. Figure_4-14 is the circuit diagram of a bit slice which shows how these components are connected together. Figure_4-15 is the layout for one bit.

The 2x1 Mux is used to select the complement of B whenever the subtract (A + B* + Cin) operation is desired. From the circuit diagram, it is observed that sub and sub* must settle before ϕ_2 ends, otherwise node B* won't be charged or discharged to a known value. This requirement was discovered when ESIM [M&O&S] was run on this section.

Static logic gates are used because busses' values are latched in during ϕ_2 and logic is also evaluated in ϕ_2. If dynamic logic is used, race conditions will discharge precharged nodes mistakenly.

The control signals or_ϕ_4, and_ϕ_4, and xor_ϕ_4 together with addORsub_ϕ_4 in the Sum block determine which output will get onto the busD in ϕ_4. To avoid bus conflict, only one can be asserted. Furthermore they can be asserted ONLY in ϕ_4.

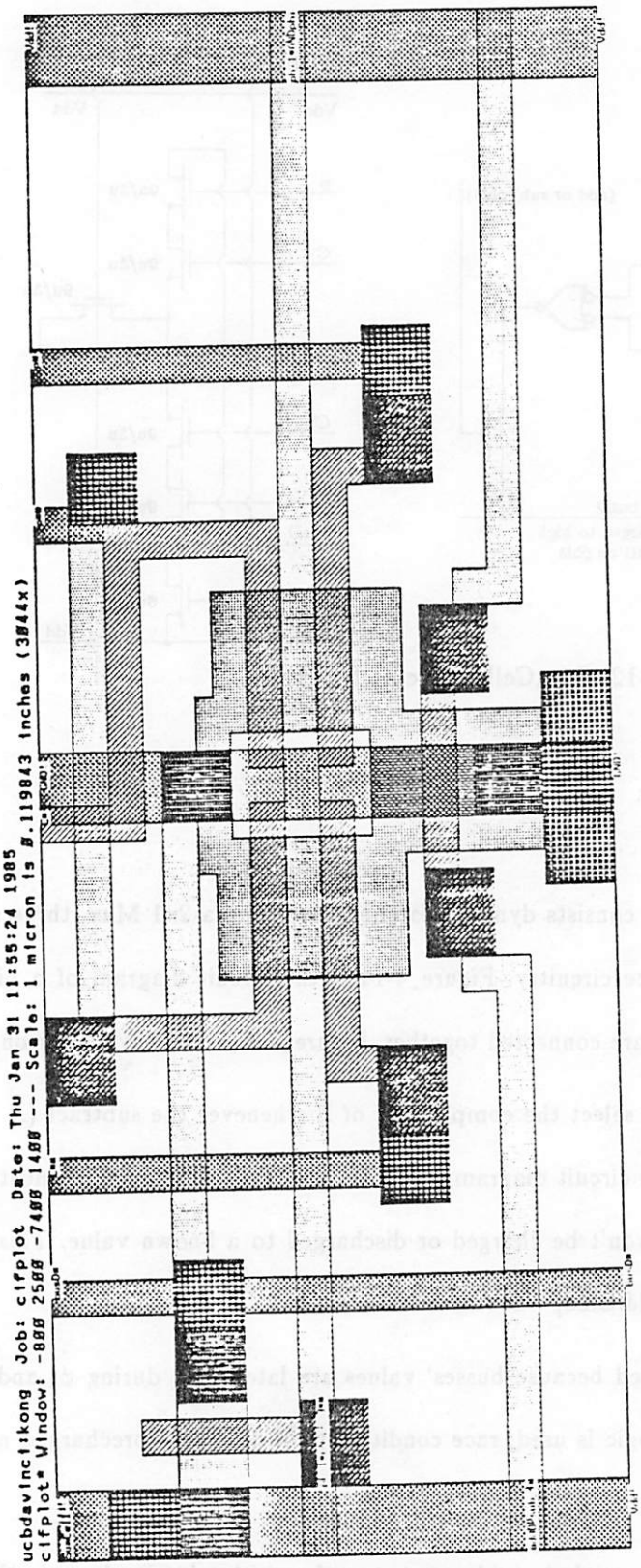


Figure 4-13 Layout Of SumCell

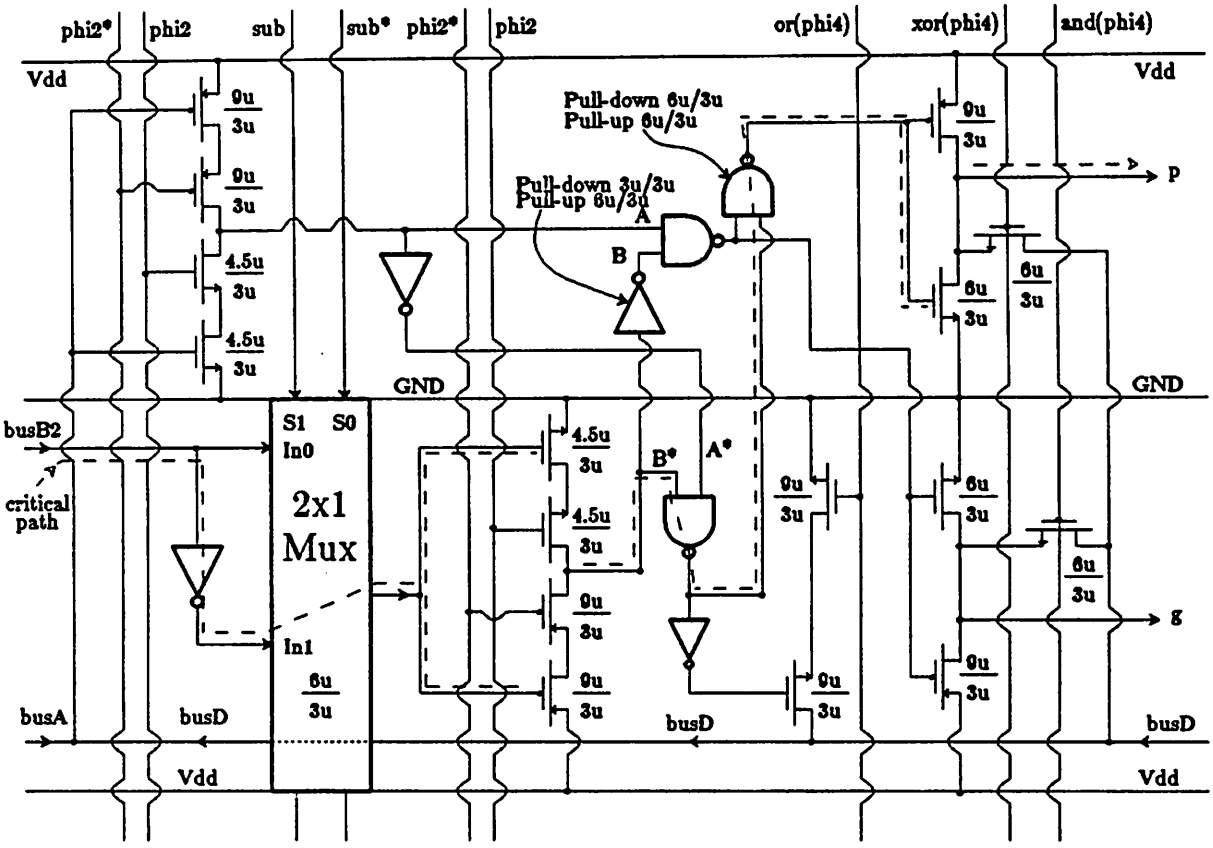


Figure 4-14 Input Logic (AluInCell)

4.4 ALU Summary

The floor plan of the ALU together with all the interface signals is shown in Figure_4-16. Figure_4-17 is the layout of a 8-bit ALU. The 32-bit ALU has been proved to be functionally correct by ESIM [M&O&S]. The patch file for ESIM and ESIM's output are all included in appendix A.

The critical path for ϕ_3, as shown in Figure_4-18, is simulated before and after the layout (first to get an idea of how big transistors have to be and then after the layout, parasitic capacitance can be approximated more accurately) in Spice. The final simulation, which is included in appendix A, shows this critical path has a delay of 31.6ns.

The critical path for ϕ_2 is shown in Figure_4-19. It is also simulated by by Spice and the result is 18.4ns. The print out of this simulation is also in appendix A.

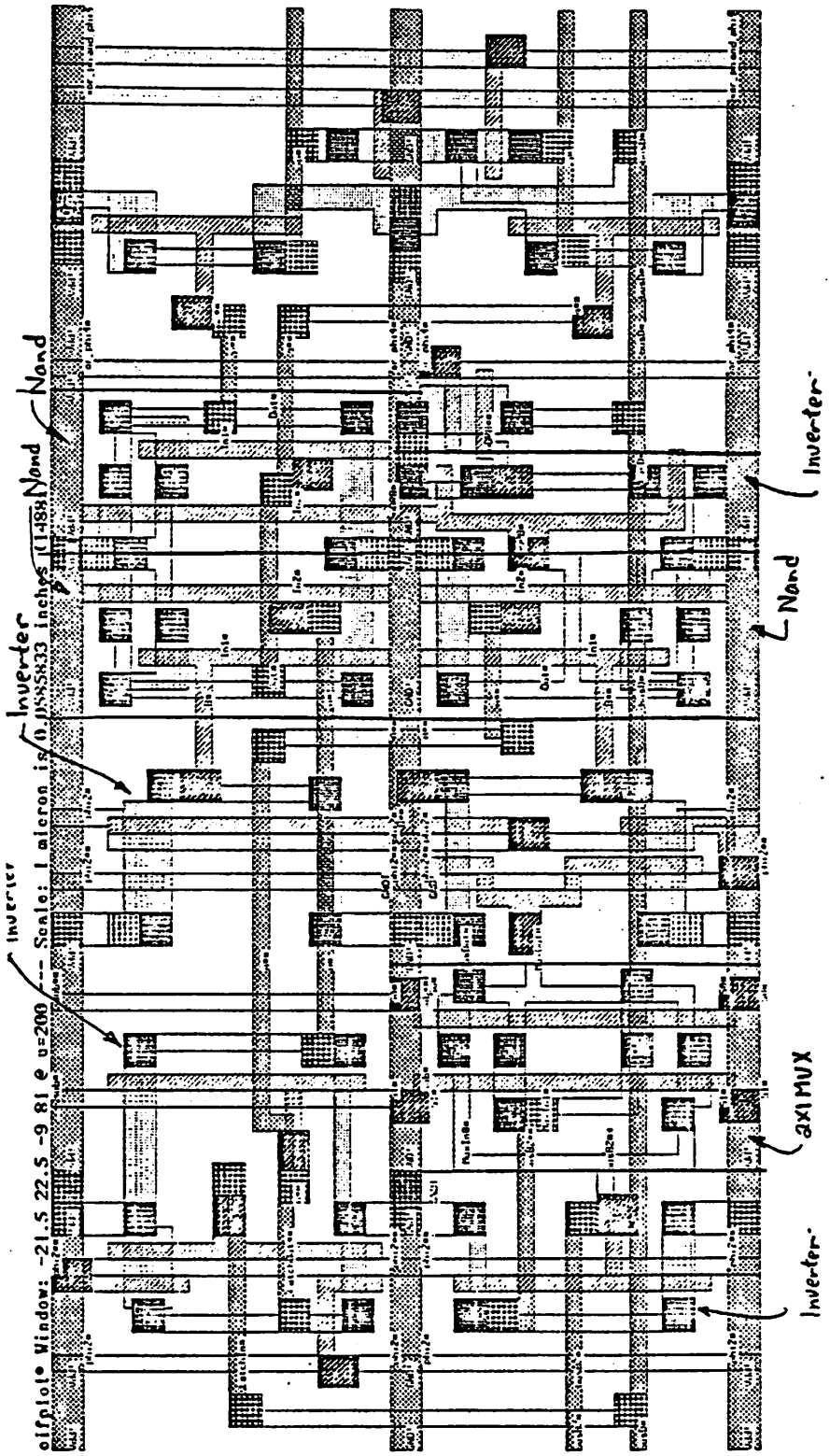


Figure 4-15 Layout Of Input Logic (AluInCell)

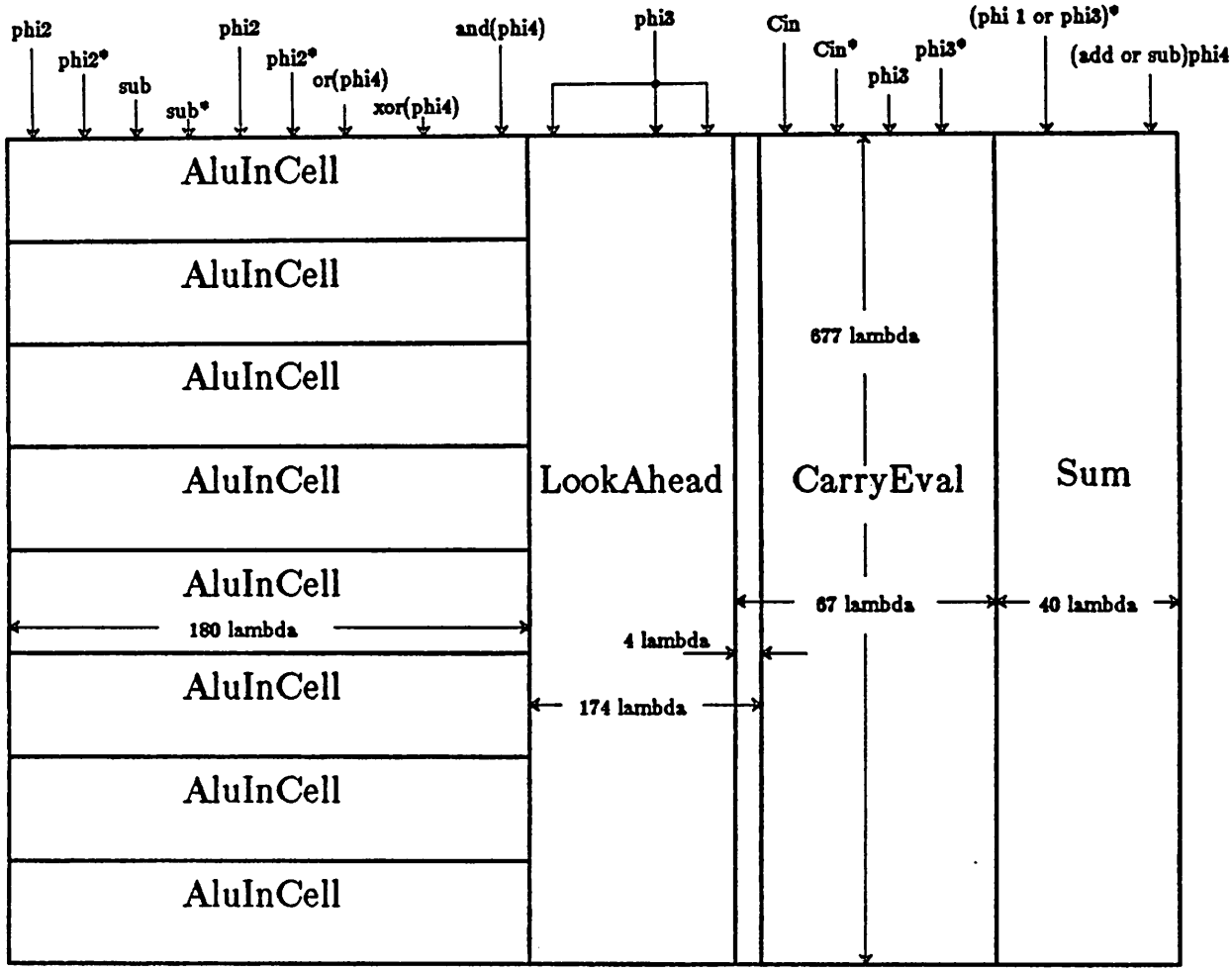
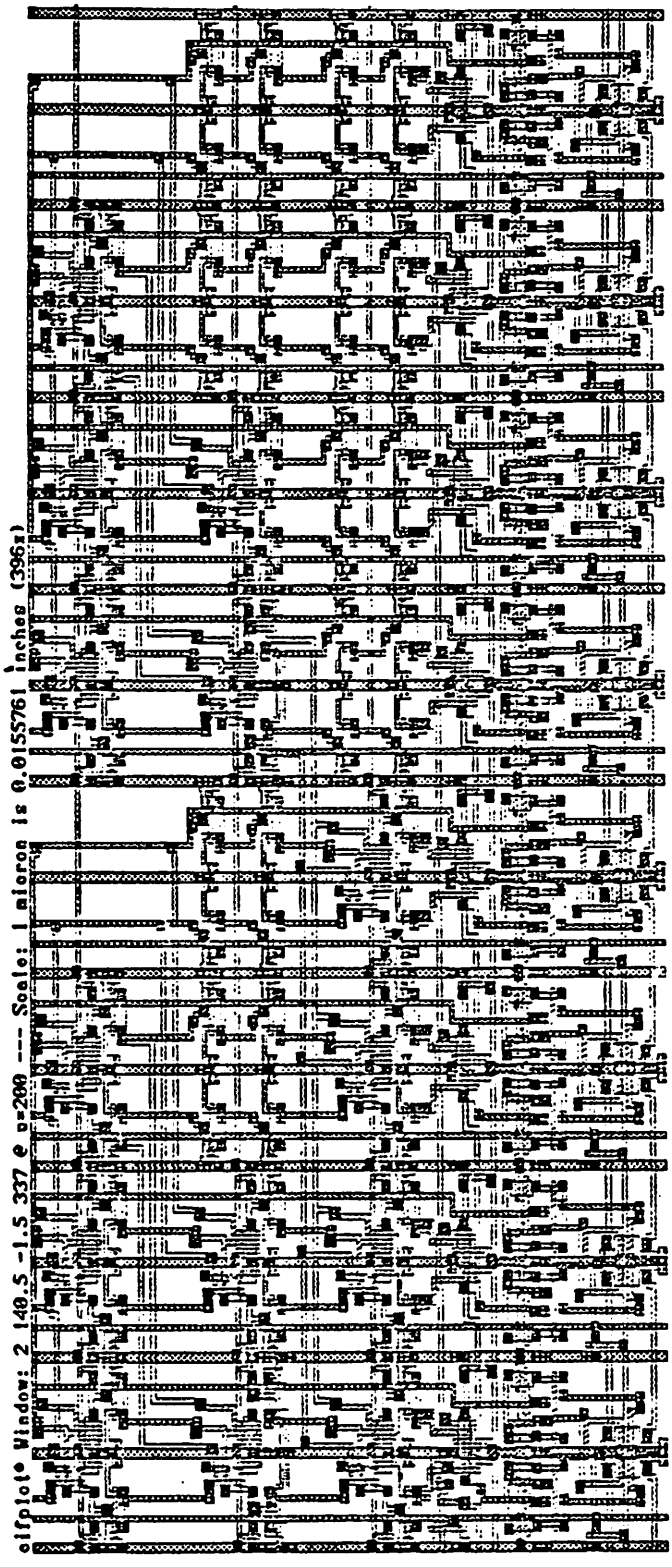


Figure 4-16 8-bit ALU's Floor Plan (Not in scale)

Figure 4-17a Layout Of 8-bit Adder



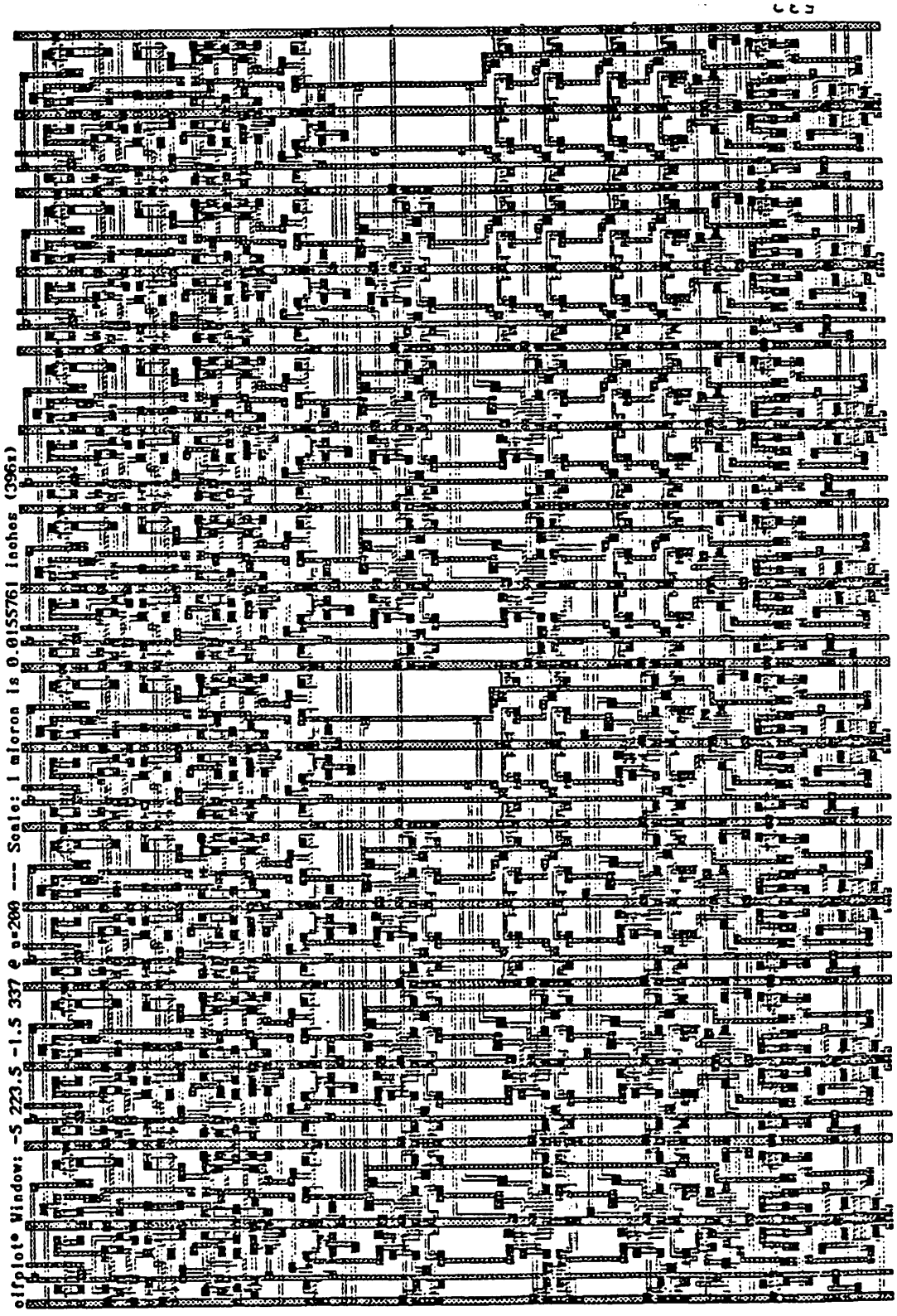


Figure 4-17b Layout Of 8-bit Arithmetic Logic Unit

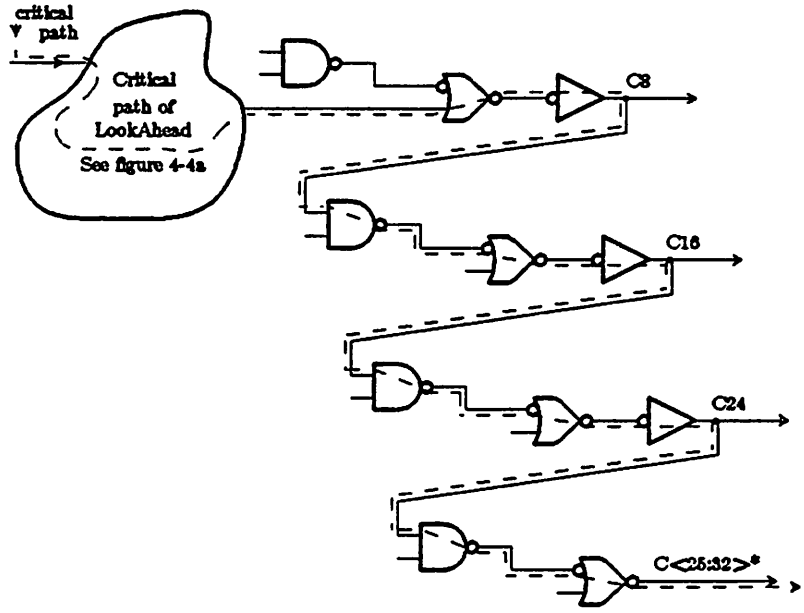
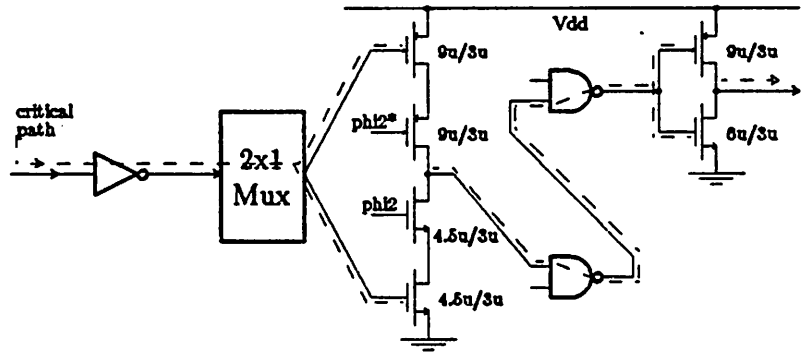


Figure 4-18 Critical Path In Phi3



This critical path is also shown in figure 4-14

Figure 4-19 Critical Path In Phi2

5. IMPLEMENTING CONTROL FINITE STATE MACHINE IN VLSI

5.1 Finite State Machine

5.1.1 Definition Of a Finite State Machine

A finite state machine is defined in [Koh78] to have its next state $S(n+1)$ uniquely defined by the present state $S(n)$ and the present input $x(n)$:

$$S(n+1) = \delta\Big[S(n), x(n)\Big] \tag{5.1}$$

where δ is the state transition function.

The output of the finite state machine in state $S(n)$ is $z(n)$, which in the most general case, is a function of both the present state $S(n)$ and the present input $x(n)$:

$$z(n) = \lambda\Big[S(n), x(n)\Big] \tag{5.2}$$

where λ is the output function. A special case of this general case (Equation 5.2) is that the output $z(n)$ is a function of the present state $S(n)$ only and is independent of the external input:

$$z(n) = \lambda\Big[S(n)\Big] \tag{5.3}$$

The above definition of the finite state machine is sufficient for most applications. For a more formal definition of the finite state machine, please refer to [Koh78]. Before going any further, it must be pointed out that the finite state machine defined above is a sequential machine and clocking is NOT used anywhere in the definition. This means that the behavior of the machine is determined by a sequence of events but NOT by a system clock (a high low high low square wave). Sequences of events are events with respect to each other, which is NOT the same as events with respect to the system clock. The clock is introduced later for implementation but not as part of the finite state machine definition.

The finite state machine defined above (where clocking is not used as part of the definition) can be completely specified by a state diagram similar to the one shown in Figure_5-1. One of the

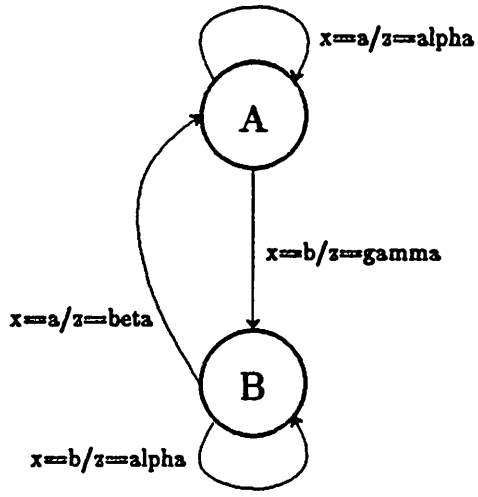


Figure 5-1 Example Of A State Diagram

things showed in Figure_5-1 is that if the present state is A then B will become the present state and the output will become γ as soon as input x changes from a to b. After this, even if input remains b, the output z will still change to α because the present state is now B. Theoretically any finite state machine can be implemented by a combinational circuit block and a delay block as shown in Figure_5-2. This approach has many problems and will be discussed in details in Section_5-2.

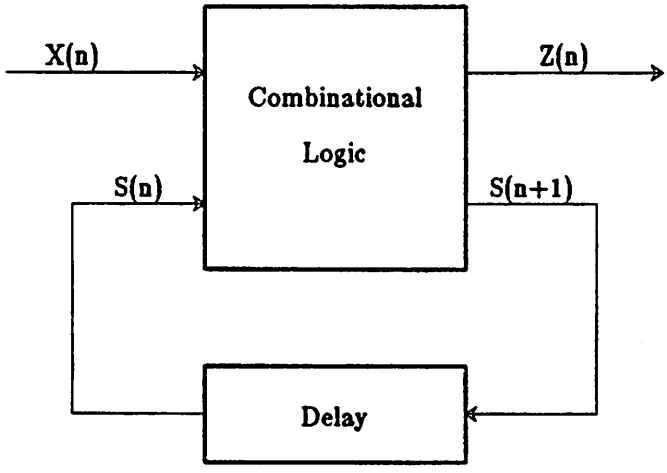


Figure 5-2 Theoretical Implementation Of Finite State Machine

5.1.2 Mealy vs Moore Machine

The general finite state machine defined by Equation 5.1 and Equation 5.2 (see Section 5.1.1)

$$S(n+1) = \delta\Big[S(n), x(n)\Big] \tag{5.1}$$

$$z(n) = \lambda\Big[S(n), x(n)\Big] \tag{5.2}$$

is known as the Mealy machine [Koh78] and the machine defined by Equation 5.1 and Equation 5.3 (see Section 5.1.1)

$$z(n) = \lambda\Big[S(n)\Big] \tag{5.3}$$

is known as the Moore machine [Koh78]. Moore machine's output $z(n)$ depends only on the present state $S(n)$ and is a special case of the Mealy machine whose output depends on both the present state $S(n)$ and the present input $x(n)$.

The above definitions of the Mealy and Moore machines do not use the concept of a system clock (a square wave) either. However in some literature, the difference between the Mealy and Moore machines were considered as finite state machines that either has asynchronous or synchronous inputs and outputs. This definition may be useful in certain applications, but in general it causes much confusion.

The confusion arises from the fact that finite state machine itself is not defined in terms of a system clock and one cannot talk about synchronization unless a clock is introduced. Trying to define the Mealy and Moore machines in terms of synchronization is like trying to define the AND and OR logic functions in terms of series and parallel combinations of transistors. Although logic can be implemented by combining transistors, logic and transistors combinations are independent things and logic was invented long before transistor was invented. Similarly although a clock (on which synchronization is based) is usually used in the implementation of finite state machine, finite state machine and clocking are independent entries and finite state machine was invented long before the concept of clocking was invented.

An example of a state diagram for a Moore machine is shown in Figure_5-3. Since the output of a Moore machine is a function of the present state only, for a given node (representing a state) in the state diagram, all arrows leaving it has the same output (z) values on them. Instead of duplicating this value over all arrows that leave a state node, the value is put inside the state node as shown in Figure_5-3b. Furthermore to make the diagram more compact, the input(s) that determines the next state is (are) put inside the state node with square bracket [ ] around it (them) as shown in Figure_5-3c. This notation is suggested by [Des83].

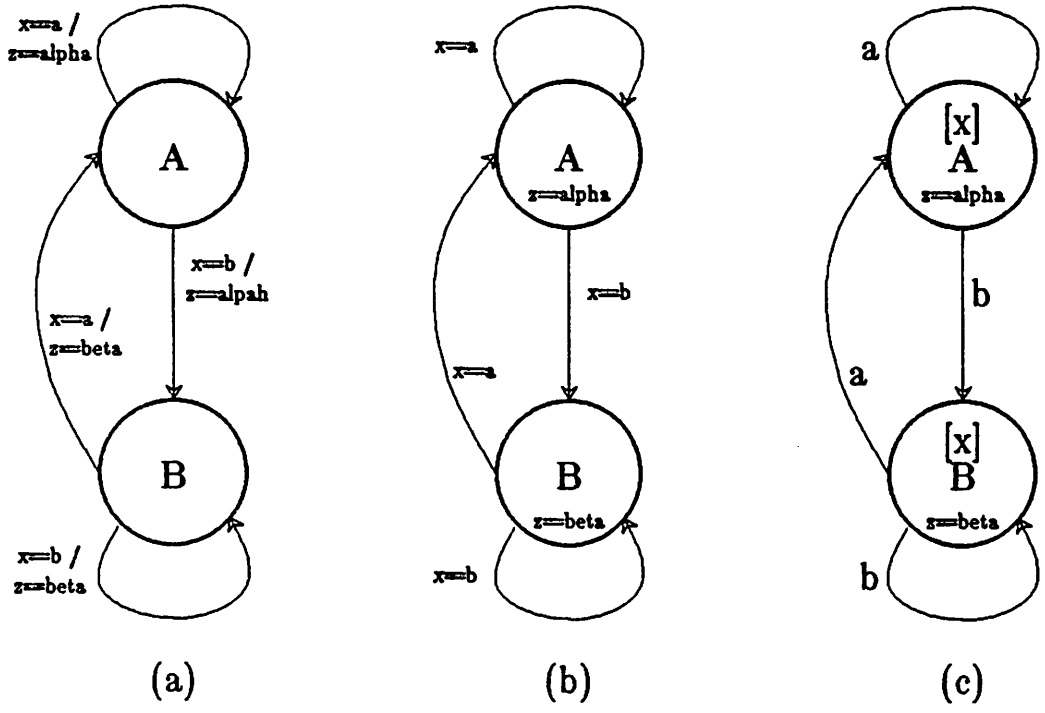
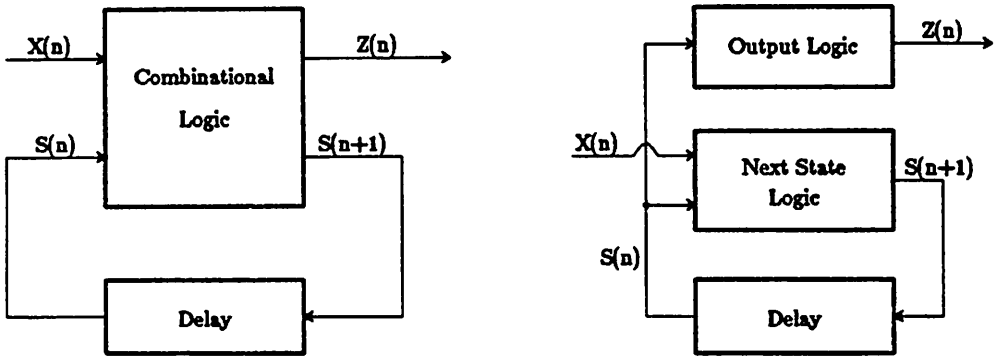


Figure 5-3 State Diagram Of A Moore Machine

The less general output function of the Moore machine given by Equation 5.3 (less general as compared to the output function of Mealy machine in Equation 5-2) also makes the Moore machine easier to implement. Instead of using a large combinational block as shown in Figure_5-2 and Figure_5-4a to implement both the output and the next state function, the combinational circuit block can be decoupled into two smaller blocks as shown in Figure_5-4b. One of this block will implement the next state function and the other will implement the relatively simple output function. Figure_5-4 is only a theoretical approach to the problem. The actual implementation is

more complicated and is discussed in Section 5-2.



(a) Mealy Machine (b) Moore Machine

Figure 5-4 Theoretical Implementations Of Mealy And Moore Machine

5.2 Implementation Of The Finite State Machine Using Clock

Theoretically, a finite state machine can be implemented by a combinational logic block and a delay block as shown in Figure_5-2. In practice, this approach has many problems. One of the problems is illustrated in Figure_5-5. Assume the finite state machine is currently in state A, then a glitch on the signal line x will causes a catastrophe. First, the low to high transition of the glitch is interpreted as a x=1 input and cause the machine to change its state to B and its output z to 1. The falling edge of the glitch is then interpreted as a x=0 input and cause the the machine to change its state to C and its output to 0.

The only solution to this problem (and all other race conditions that are not shown) is to use an edge trigger register (D type flip-flop) and drive it by a periodic square wave as shown in Figure_5-6. This periodic square wave is the clock of the system. Assume the edge trigger flip-flop in Figure_5-6 is triggered by the rising edge (low to high transition) of the clock, then at each rising edge of the clock, S and x are sampled. The sample values, which are denoted as $S(t)$ and $x(t)$, are then used by the combinational circuit to evaluate the output z and the next state S.

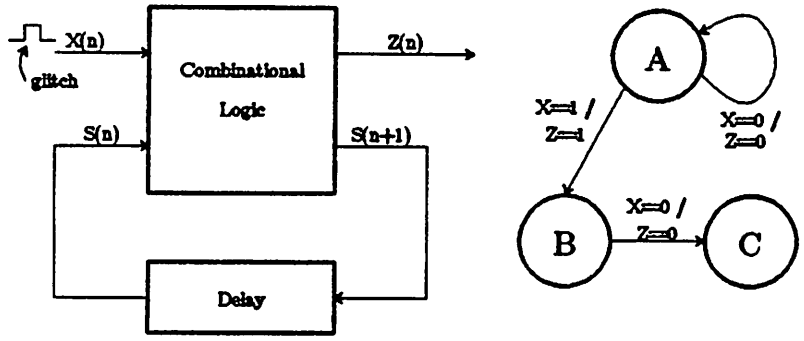
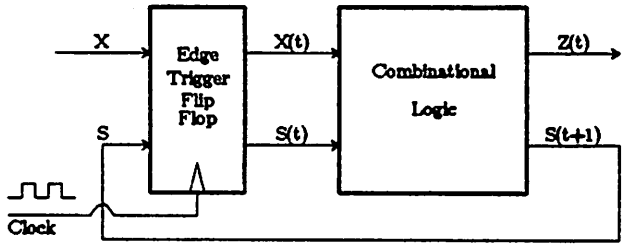


Figure 5-5 One Problem Of The Theoretical Implementation

The output z is the output of the current state $S(t)$ and is thus denoted as $z(t)$. On the other hand, the S just evaluated won't have any effect until the next rising edge of the clock when it is sampled by the register. This S therefore represents the next state and is denoted as $S(t+1)$.

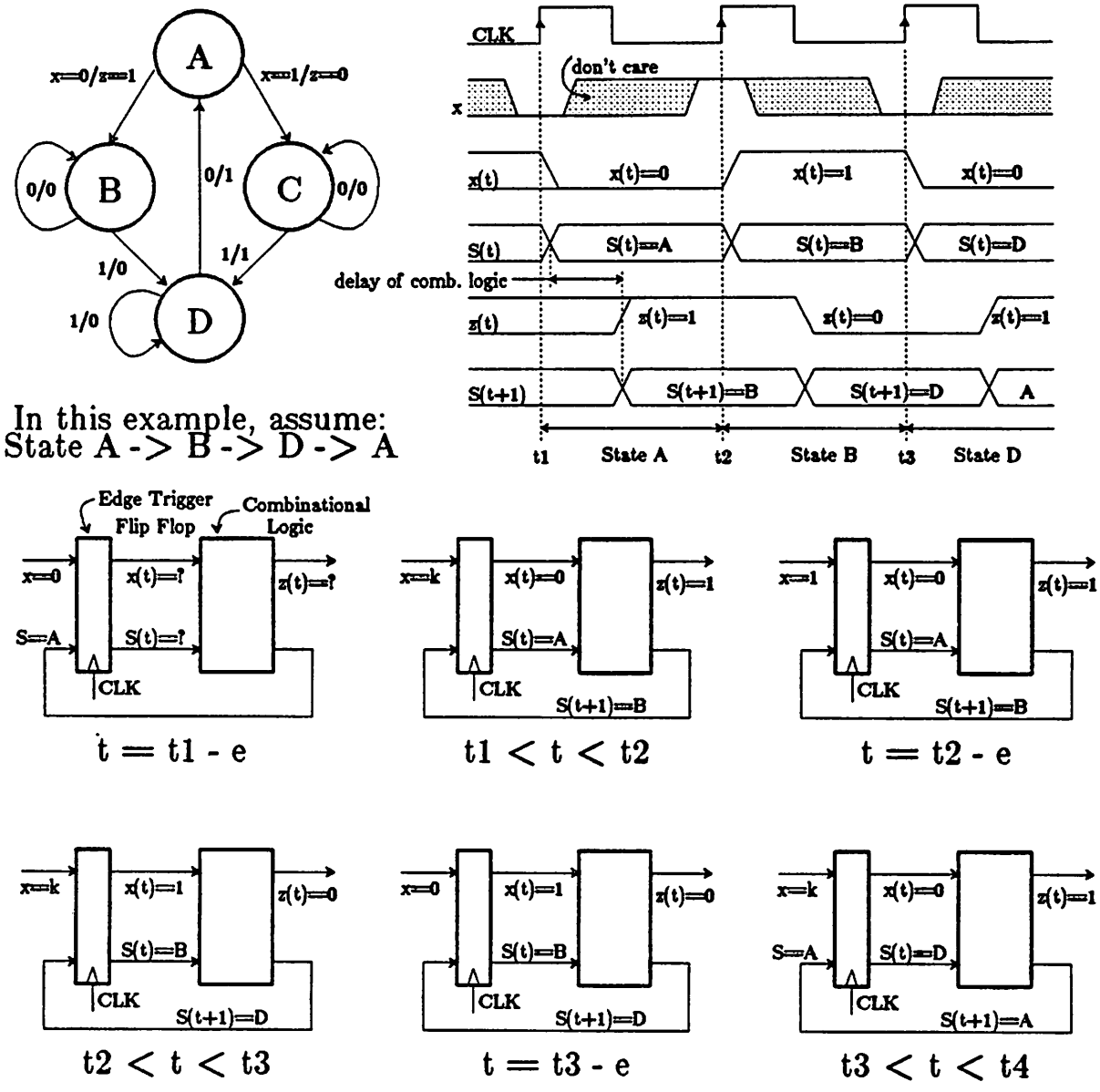


Clock	X(t)	S(t)
L	X(t-1)	S(t-1)
H	X(t-1)	S(t-1)
↗	X	S

Figure 5-6 Implementation Of Finite State Machine Using A Clock Signal

Each rising edge of the clock marks the beginning of a state because a new S, namely $S(t+1)$, is sampled and becomes $S(t)$. The $x(t)$, which determines this new state's successor and its output z (if this is a Mealy machine), is also updated at this same rising edge. Since x and S are sampled simultaneously, the input x that are intended for a given state must stabilize to its proper value before the beginning of that state (marked by the rising edge of the clock). This is illustrated in Figure_5-7 where state A is assumed to start at $t > t_1$ and state B is the desirable next state (which starts at $t > t_2$). The input x then must settle to x=0 one flip-flop set up time before t_1. This is illustrated clearly by the timing diagram in Figure_5-7.

Now that clocking is introduced to the implementation of the finite state machine, finite state machines can be classified into synchronous and asynchronous according to the way they are



In the above diagrams: ? = unknown, k = don't care, and e = setup time of the flip flop

Figure 5-7 Illustrations Of The Timing Of A Finite State Machine

implemented:

Any finite state machine that are implemented by the approach shown in Figure_5-2 is classified as an asynchronous finite state machine.

Any finite state machine that is implemented by the approach shown in Figure_5-6 (using a clock signal) is classified as a synchronous finite state machine.

This classification together with the logical behavior classification are summarized in Figure_5-8. From this figure, it can be seen that finite state machines can be classified into four groups, which are: (1) Asynchronous Mealy , (2) Asynchronous Moore, (3) Synchronous Mealy, and (4) Synchronous Moore.

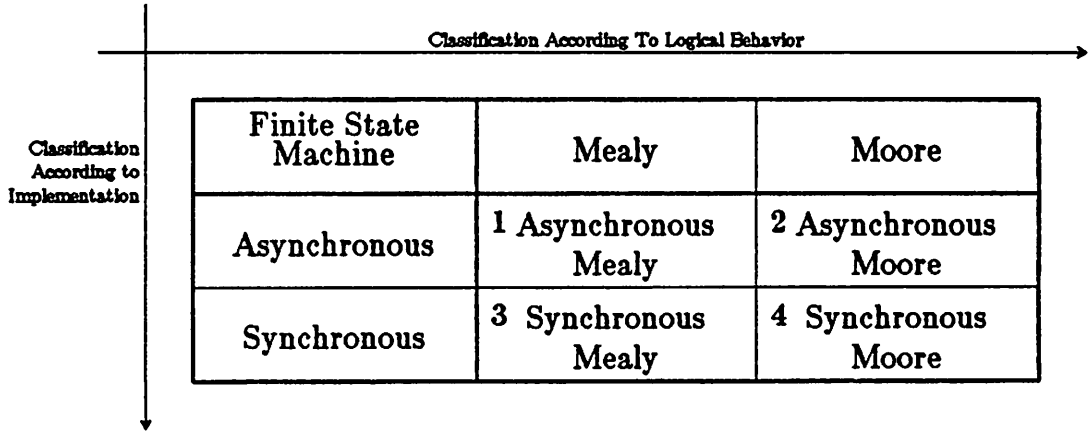


Figure 5-8 Classification Of Finite State Machine

As far as implementing the control logic in a digital system is concerned, the synchronous Moore machine is the most popular. There are two reasons for its popularity:

(1) Its output is a function of the present state only.

(2) High tolerance to glitches at the inputs.

The first reason is very important because it makes the design of the state diagram much easier by enabling the design effort to be split into two steps. First, the designer will consider all the possible states his system can arrive at as a result of all the conditions (inputs to the finite state machine), then he can determine all the control signals (output of his finite state machine) it needs at each state.

A generic Synchronous Moore machine is shown in Figure_5-9a. When it is used to control a digital system, different names can be given to the different components according to their functions as shown in Figure_5-9b. The first step in designing the state diagram mentioned in the previous paragraph will result in the specifications for the "Next State Logic" block. The second step will then produce the specifications for the "Control Logic" block.

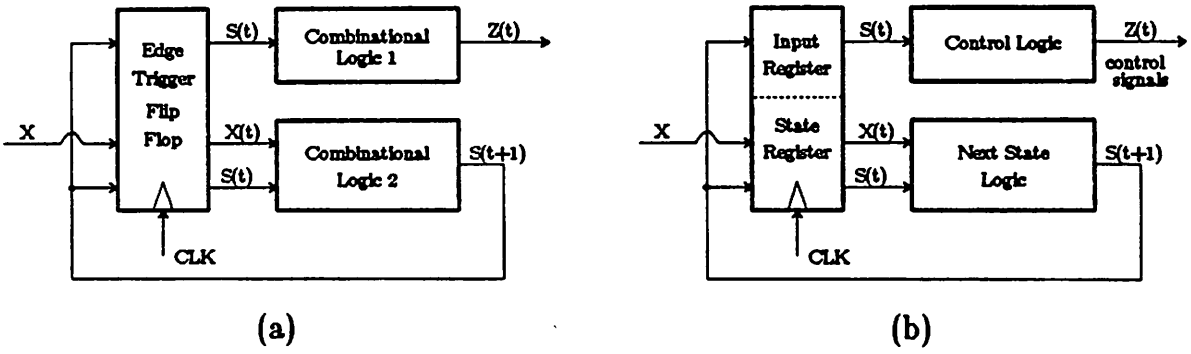


Figure 5-9 Implementation Of Synchronous Moore Machine

5.3 Implementing Finite State Machine In MOS VLSI

5.3.1 How The Non-Overlapping Two-Phase Clock Is Used

In MOS circuits, an edge triggered flip-flop is hard to build. However solution similar to Figure_5-6 can be achieved if dynamic latches are placed at both the input and the output side of the combinational circuit. These two dynamic latches are driven by ϕ_1 and ϕ_2 of a two-phase non-overlap clock. This is shown in Figure_5-10.

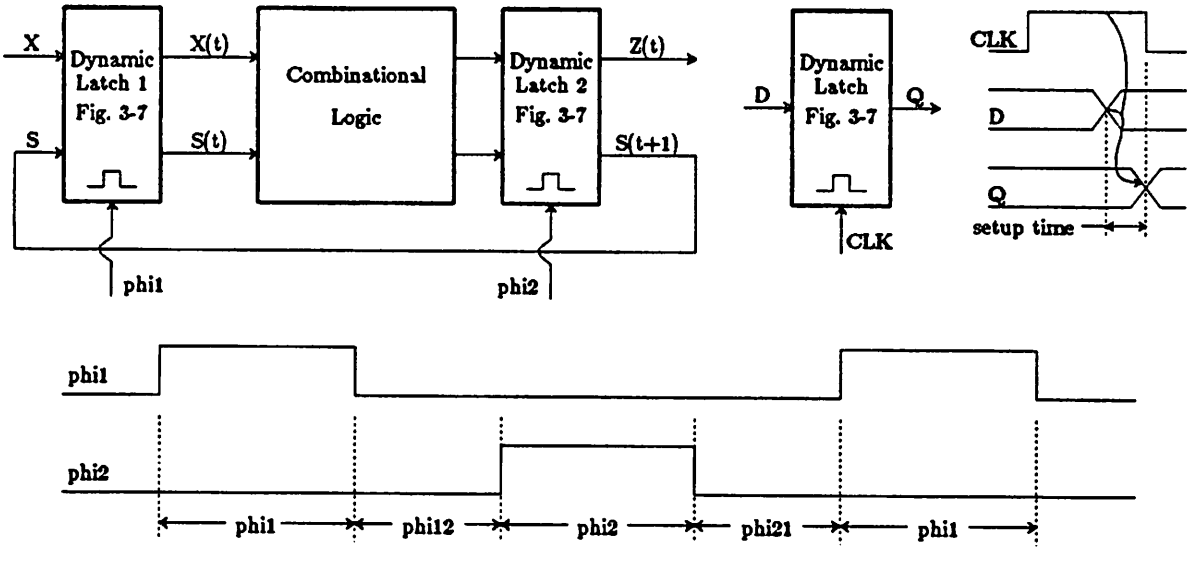


Figure 5-10 Finite State Machine Using A 2-phase Non-overlap Clock

S and x are sampled whenever $\phi_1=V_{dd}$. The sample values, which are denoted as $S(t)$ and $x(t)$ are then used by the combinational circuit to evaluate the output z and the next state S. The output z, which won't be available from dynamic latch#2 until $\phi_2=V_{dd}$, is the output of the current state $S(t)$ and is thus denoted as $z(t)$. The S, which is available at the same time as $z(t)$, won't be sampled by dynamic latch#1 and thus won't have any effect on the finite state machine until the next $\phi_1=V_{dd}$. This S therefore represents the next state and is denoted as $S(t+1)$.

Each $\phi_1=V_{dd}$ can be considered as the beginning of a new state because $S(t)$ is updated. The $x(t)$, which determines this new state's successor and its output z (if it is a Mealy machine), is also updated when $\phi_1=V_{dd}$. Since x and S are sampled simultaneously, the input x that are intended for a given state must stabilize to its proper value before that state begins (marked by $\phi_1=V_{dd}$). Since dynamic latch#1 won't stop latching until one set up time before ϕ_1 goes low, x does not have to settle to the proper value until one set up time before the falling edge of ϕ_1. This, however, will require the combinational logic to have a delay t_{delay} smaller than:

$$t_{delay} < \phi_{12} + \phi_2 - t_{setup2}$$

$$t_{setup2} = \textit{setup time of dynamic latch\#2}$$

On the other hand, if x is required to settle down before the rising edge of ϕ_1, then the combinational logic delay t_{delay} only has to be:

$$t_{delay} < \phi_1 + \phi_{12} + \phi_2 - t_{setup2}$$

The timing of the finite state machine can be best illustrated by an example similar to Figure_5-7. This is done is Figure_5-11 where state A is assumed to start at $t > t_1$ and state B is the next desirable state (which starts at $t > t_2$). Consequently the input x must settle to x=0 one set up time before ϕ_1 goes low.

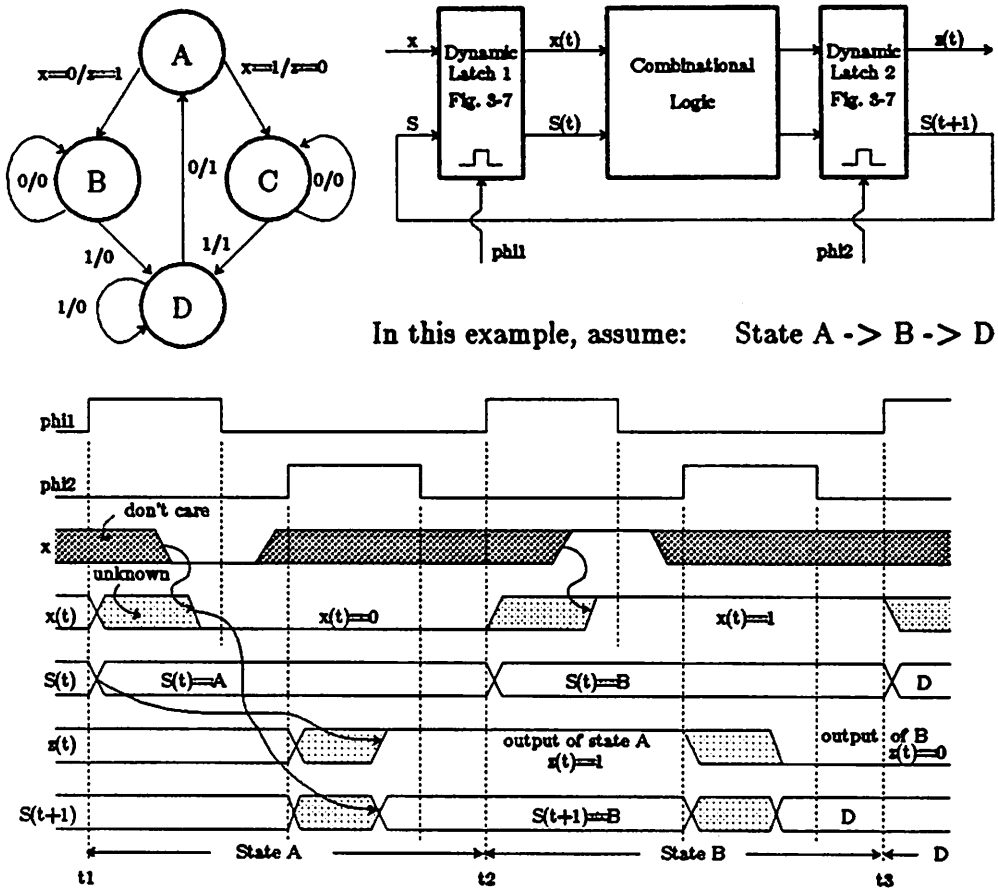


Figure 5-11 Timing Of A Finite State Machine Using A 2-phase Non-overlap Clock

5.3.2 Two Set Of Signals In Each State

In MOS system, it is common that some control signals can only be asserted in certain phases of the clock. For example, in a non-overlap two phases system, some control signals $Z_1(\phi_1)$ can only be asserted in ϕ_1 while some other control signals $Z_2(\phi_2)$ can only be asserted in ϕ_2

The above requirement can be handled by a more general finite state machine shown in Figure_5-12. A Moore machine is used here (compare this to Figure_5-9) because Moore machine is highly recommended for implementing control logic as discussed at the end of Section 5.2. The timing diagram in Figure_5-12, however, also applies to the more general Mealy machine. $Z_1(\phi_1)$ is glitch free and the extra dynamic latch#3 is used here to prevent glitches on the control line $Z_2(\phi_2)$. Since both $Z_1(\phi_1)$ and $Z_2(\phi_2)$ are glitch free, they can be used to control precharged nodes as shown in Figure_5-13.

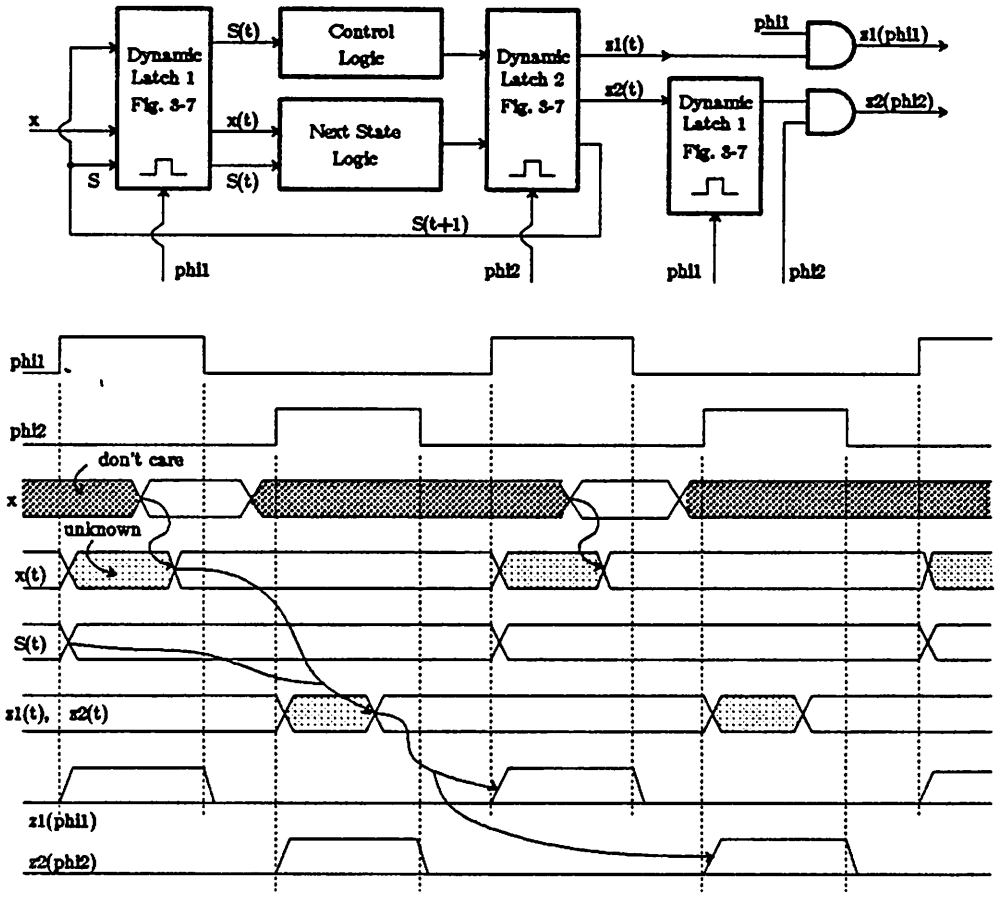
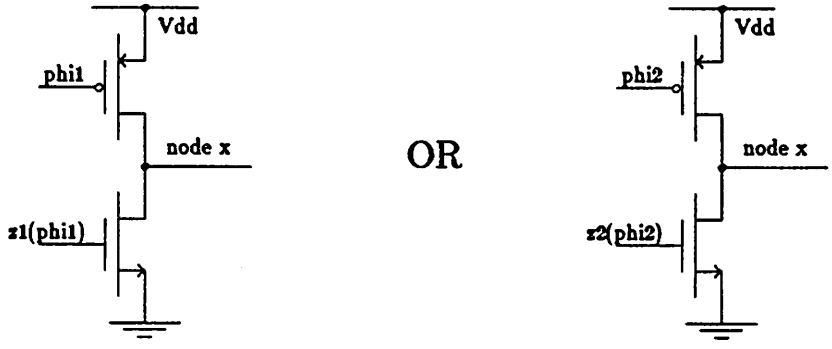


Figure 5-12 Timing Of A Finite State Machine With Two Sets Of Output Signals



Node x precharged to Vdd in phi1*
Discharged conditionally in phi1

Node x precharged to Vdd in phi2*
Discharged conditionally in phi2

Figure 5-13 Control Signals For Precharged Nodes

An example similar to Figure_5-11 is shown in Figure_5-14. From the timing diagram in this Figure (and Figure_5-12), it can be seen that control signal $Z_1(\phi_1)$ and $Z_2(\phi_2)$ of the current state

won't be updated until $\phi 1$ and $\phi 2$ of the next cycle. This does not pose any serious problem because this delay happens in every cycle and can be taken into account easily when designing the state diagram.

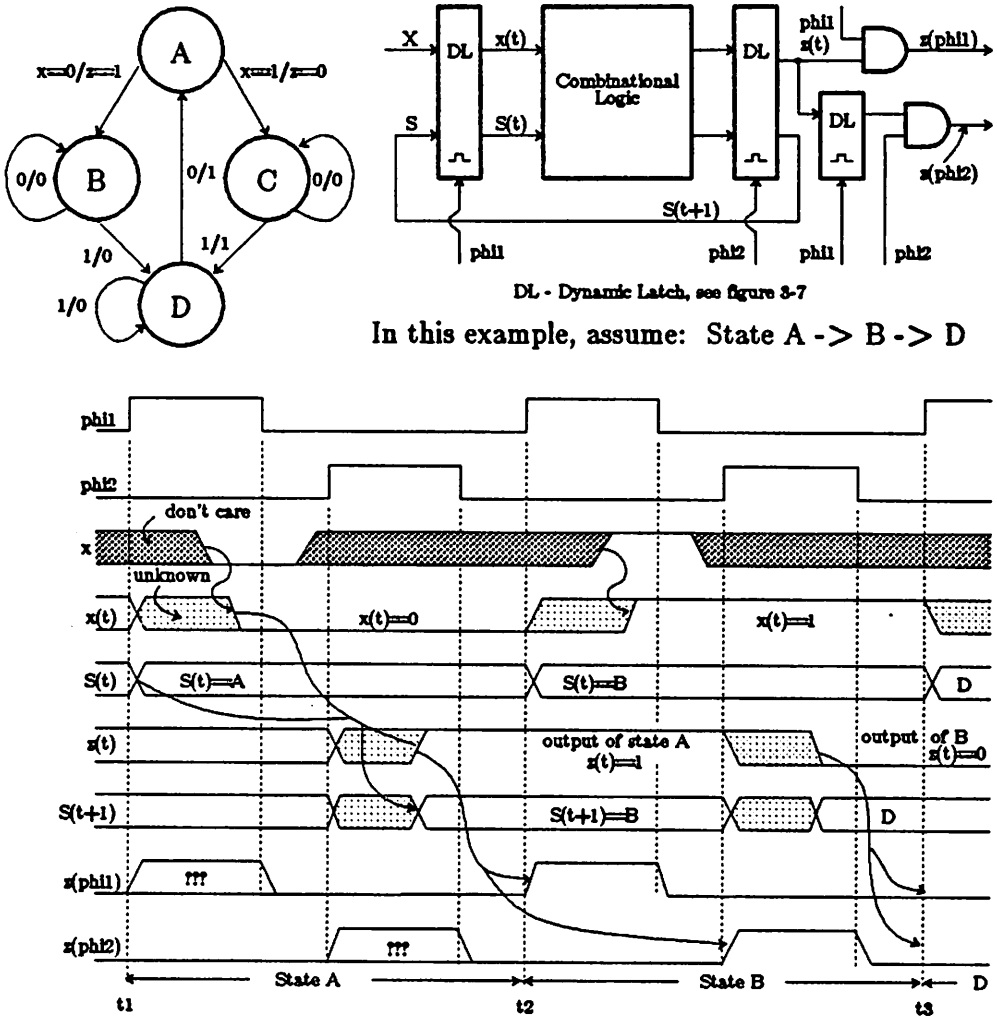
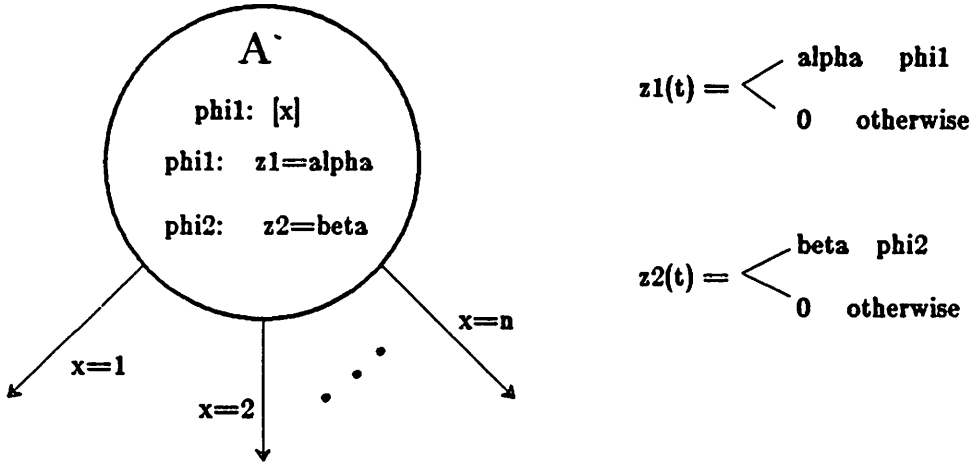


Figure 5-14 Example Of A Finite State Machine With Two Sets Of Outputs

5.3.3 Suggested State Diagram

The finite state machine shown in Figure_5-12 has two sets of outputs. One can only be asserted in $\phi 1$ and the other can only be asserted in ϕ_2. This is hard to represent in a general state diagram but can be done easily for the state diagram of a Moore machine.



Current state is A, next state can be one of n possible states determined by the value of input X.

Figure 5-15 A General State Of The Proposed State Diagram

Figure_5-15 shows the general state of a state diagram that can be used to represent the Moore machine shown in Figure_5-12. Figure_5-15 is a simple extension of the state diagram suggested in Figure_5-3c (Section 5.1.2). Besides showing explicitly that control signals Z_1 and Z_2 can be valid only in ϕ_1 and ϕ_2 respectively, Figure_5-15 also shows that the input x, which determines the next state, is latched in during ϕ_1.

6. SUGGESTIONS FOR FUTURE RESEARCH

This report is by no means a complete coverage of MOS circuit design techniques. As a matter of fact, this report asks more questions that can be only be answered by further research.

First of all, the bootstrap driver discussed in chapter 2 is in NMOS. Bootstrap drivers may also be useful in CMOS just to achieve faster rise and/or fall time through a gate drive higher (lower) than the supply voltage(s) (V_{dd} and GND) . However in CMOS, certain nodes may be desirable to be bootstraped to a voltage lower than GND while some other nodes are desirable to be bootstraped to a voltage higher than V_{dd}. Furthermore, no depletion mode transistor is available in CMOS and how a PMOS transistor can be used as a substitute will be an interesting question to be answered. All these matters can only be understood by further research.

In the discussion of Domino and NORA logic discussion, a tradeoff exists between using all N-logic block and using a mixed of N- and P-logic blocks. Using a mixed of N- and P-logic blocks eliminates the needs of inverters between gates and thus eliminates the inverter delay. However, P-logic block is inherently slower than N-logic block. The inverter delay saved by using P-logic block is therefore partially offset.

Another challenging question to be answered in Domino and NORA logic is the complexity of a gate versus number of gate levels in implementing a given logic function. Figure_3-12d and 3-13d show how the composite function AND-OR is implemented by one single gate. This clearly is an improvement over the implementation using two gate levels but how far this can be extended is not known. As a gate becomes more complex, it becomes slower because of higher parasitic capacitance and also because of more transistors are combined in series. Charge sharing problem also intensifies as the gate becomes more complicated due to the increase in parasitic capacitance.

The bubble matching technique, which is described in Section 3.2.2 and is used to check for combination rule violation when N- and P-logic blocks are connected, can be automated. Another candidate for automation is fine tuning the critical path in Domino logic. It is shown in Section 4.2.2 that critical path in Domino logic can be fine tuned by varying the size of the buffter

between gates. This process can probably be automated, but the 2nd order effects of varying the buffer size must also examined.

Finally in chapter 5, only basic ideas of implementing control logic are introduced. It will be extremely interesting to find out how these basic ideas can be extended to a multi-phase system that has more than two phases in its system clock. Do the phases have to be non-overlapped? Which two phases should be used to drive the input and output dynamic latches of the combinational logic? The decision on this question will determine the maximum allowed propagation delay of the combinational circuit.

7. REFERENCES

[Des83] A. Despain, Personal Communication, Berkeley 1983.

[Gon83] N. Goncalves, H. De Man: "NORA: A Racefree Dynamic CMOS Technique for Pipelined Logic Structure", IEEE Journal of Solid-State Circuits, vol. sc-18, no.3, June, 1983.

[H&J83] D.A. Hodges and H.G. Jackson: "Analysis and Design of Digital Integrated Circuits", McGraw-Hill, New York 1983.

[Koh78] Z. Kohavi: "Switching and Finite Automata Theory", McGraw-Hill, New York 1978.

[Kra82] R. Krambeck, C. Lee, and S. Law: "High-Speed Compact Circuits with CMOS", IEEE Journal of Solid-State Circuits, vol. sc-17, no.3, June, 1982.

[M&C80] C. Mead and L. Conway: "Introduction to VLSI Systems", Addison- Wesley Publishing Company, Reading Massachusetts, 1980.

[M&O&S] R. Mayo, J. Ousterhout, and W. Scott, editors: "1983 VLSI Tools: Selected Works by the Original Artists", Computer Science Division, University of California, Berkeley 1983.

[Ous84] J. Ousterhout: "Magic Tutorial #1-#6", Computer Science Division, University of California, Berkeley 1984.

[Suz73] Y. Suzuki, K. Odagawa, and T. Abe: "Clocked CMOS Calculator Circuitry", IEEE Journal of Solid-State Circuits, vol. sc-8, Dec., 1973.

[SW84] S. Whalen: "CMOS Adder Designs for High Performance Microprocessors", Masters' Report, Computer Science Division, University of California, Berkeley 1984.

[Ung84] D. Ungar, R. Blau, P. Foley, D. Samples, and D. Patterson: "Architecture of SOAR: Smalltalk on a RISC", 11th Annual Symposium on Computer Architecture, Ann Arbor, Michigan 1984.

[Win80] D. Winkel and F. Prosser: "The Art Of Digital Design: An Introduction To Top-Down Design", Prentice-Hall, New Jersey 1980.

A. SIMULATION OF THE ARITHMETIC LOGIC UNIT

A.1 Logic Simulation (ESIM) Of The Arithmetic Logic Unit

Before simulating the whole 32-bit ALU, the subsystems CarryEval, LookAhead, and the input logic section AluInCell were simulated separately. First a bit slice was simulated, then a byte of each subsystem was simulated. The subsystems were then put together to form an 8-bit ALU and its function was simulated. Finally, after all these were done, the whole 32-bit ALU, which is formed by cascading four 8-bit ALU together, is simulated. This appendix, only includes the final simulation of the 32-bit ALU.

The following test cases were used in the 32-bit ALU simulation (see the patch file included in this report):

Case 1:

Test (A - B) with $C_{in} = 0$ (borrow = 1) and A = B

$busA\langle 31{:}0\rangle$ = 01 001 010 01 001 010 01 001 010 01 001 010

$busB\langle 31{:}0\rangle$ = 01 001 010 01 001 010 01 001 010 01 001 010

Expected result (see ESIM's output file included in this report):

$busD\langle 31{:}0\rangle$ = 11 111 111 11 111 111 11 111 111 11 111 111

$$C_{out} = 0$$

Case 2:

Test (A - B) with $C_{in} = 1$ (borrow = 0) and A = B

$busA\langle 31{:}0\rangle$ = 01 001 010 01 001 010 01 001 010 01 001 010

$busB\langle 31{:}0\rangle$ = 01 001 010 01 001 010 01 001 010 01 001 010

Expected result:

$busD\langle 31{:}0\rangle$ = 00 000 000 00 000 000 00 000 000 00 000 000

$$C_{out} = 1$$

Case 3:

Test (A + B) with $C_{in} = 1$ and A = B

$busA\langle 31{:}0\rangle$ = 10 110 101 10 110 101 10 110 101 10 110 101

$busB\langle 31{:}0\rangle$ = 01 001 010 01 001 010 01 001 010 01 001 010

Expected result:

$busD\langle 31{:}0\rangle$ = 00 000 000 00 000 000 00 000 000 00 000 000

$$C_{out} = 1$$

Case 4:

Test (A + B) with $C_{in} = 0$ and A = B

$busA\langle 31{:}0\rangle$ = 10 110 101 10 110 101 10 110 101 10 110 101

$busB\langle 31{:}0\rangle$ = 01 001 010 01 001 010 01 001 010 01 001 010

Expected result:

$busD\langle 31{:}0\rangle$ = 11 111 111 11 111 111 11 111 111 11 111 111

$$C_{out} = 0$$

Case 5:

Test (A or B) with $C_{in} = don't\ care$

$busA\langle 31{:}0\rangle$ = 1110 1110 1110 1110 1110 1110 1110 1110

$busB\langle 31{:}0\rangle$ = 1010 1010 1010 1010 1010 1010 1010 1010

Expected result:

$busD\langle 31{:}0\rangle$ = 1110 1110 1110 1110 1110 1110 1110 1110

Case 6:

Test (A and B) with C_{in} = *don't care*

$busA\langle 31{:}0\rangle$ = 1110 1110 1110 1110 1110 1110 1110 1110

$busB\langle 31{:}0\rangle$ = 1010 1010 1010 1010 1010 1010 1010 1010

Expected result:

$busD\langle 31{:}0\rangle$ = 1000 1000 1000 1000 1000 1000 1000 1000

Case 7:

Test (A xor B) with C_{in} = *don't care*

$busA\langle 31{:}0\rangle$ = 1110 1110 1110 1110 1110 1110 1110 1110

$busB\langle 31{:}0\rangle$ = 1010 1010 1010 1010 1010 1010 1010 1010

Expected result:

$busD\langle 31{:}0\rangle$ = 0110 0110 0110 0110 0110 0110 0110 0110

All these expected values are observed when ESIM was run (see ESIM's output included in this report). Furthermore, notice:

(1) busD has the expected value only during ϕ_4. During ϕ_1 and ϕ_3, busD is precharged to 1.

(2) C_{out} has the expected value only during ϕ_3. During all other times, C_{out} is precharged to 0.

(3) During ϕ_2, busD=busA because busD is driven by busA as input to the ALU.

(4) In the patch file, it can be seen that each clock cycle is simulated by four ESIM cycles. This is a direct result of the four-phase clock.

(5) In the patch file, it can be seen that busA and busB have don't care values except during ϕ_2 because the ALU latches in these two busses only during ϕ_2.

A.2 Timing Simulation (Spice) Of The Arithmetic Logic Unit

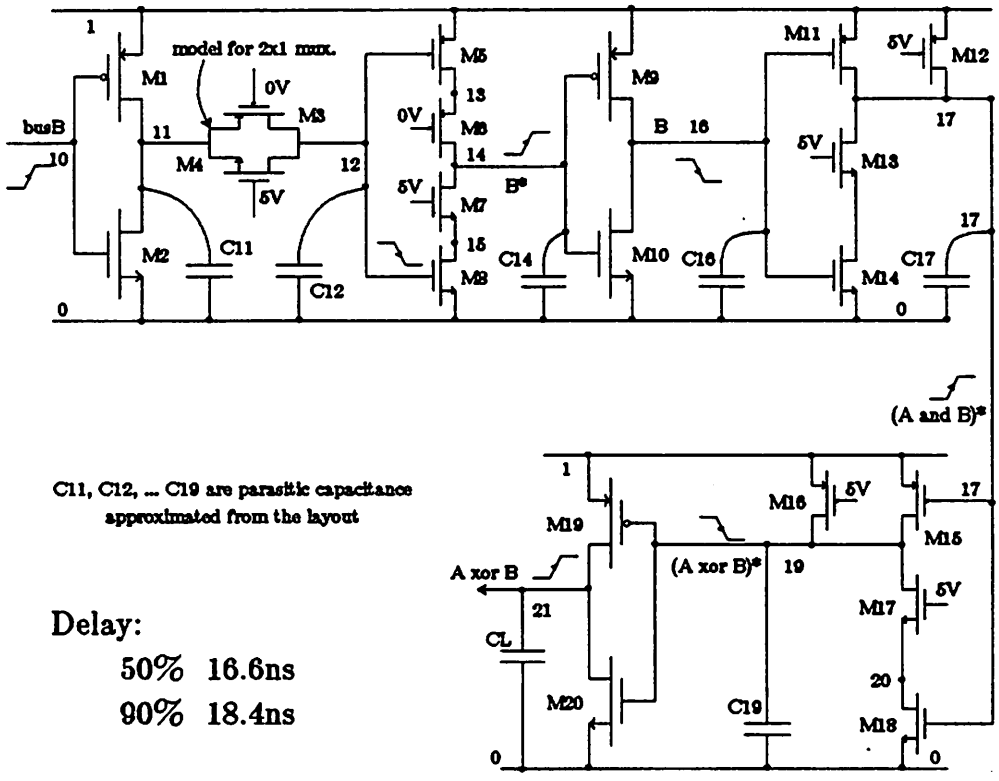
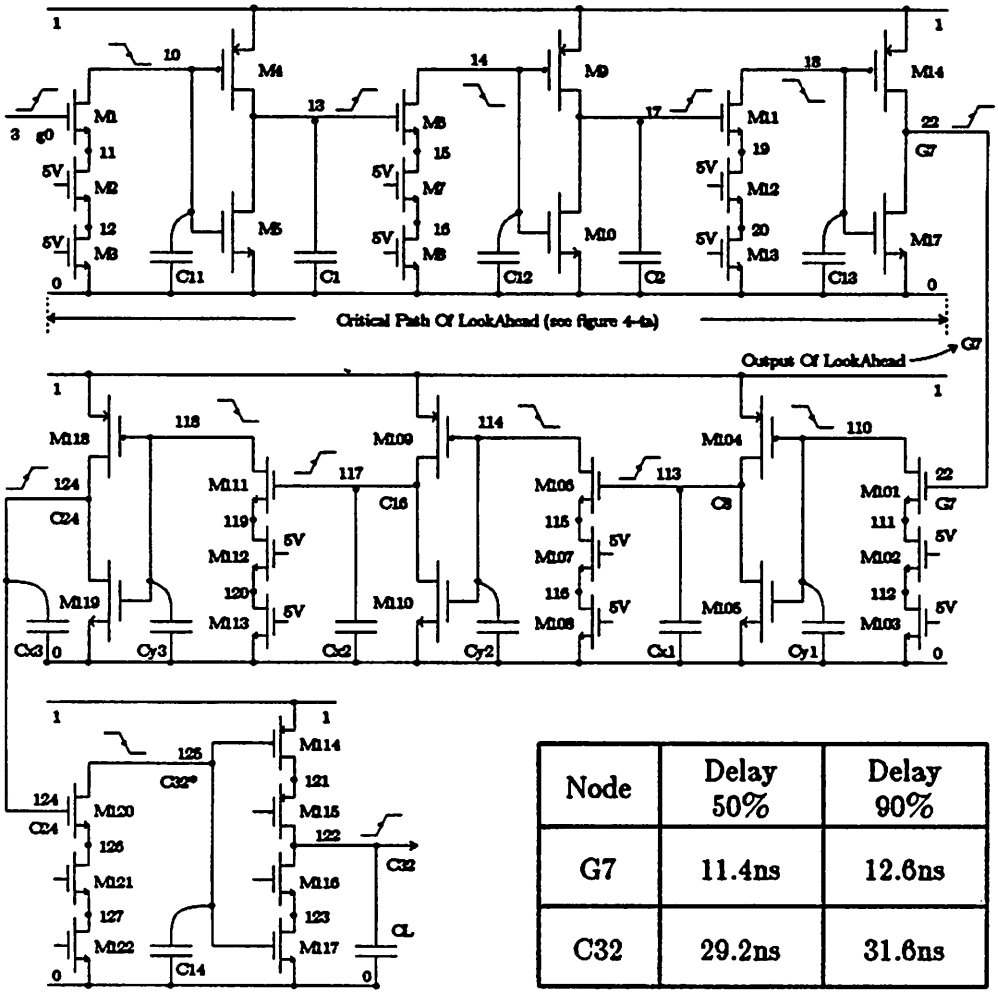


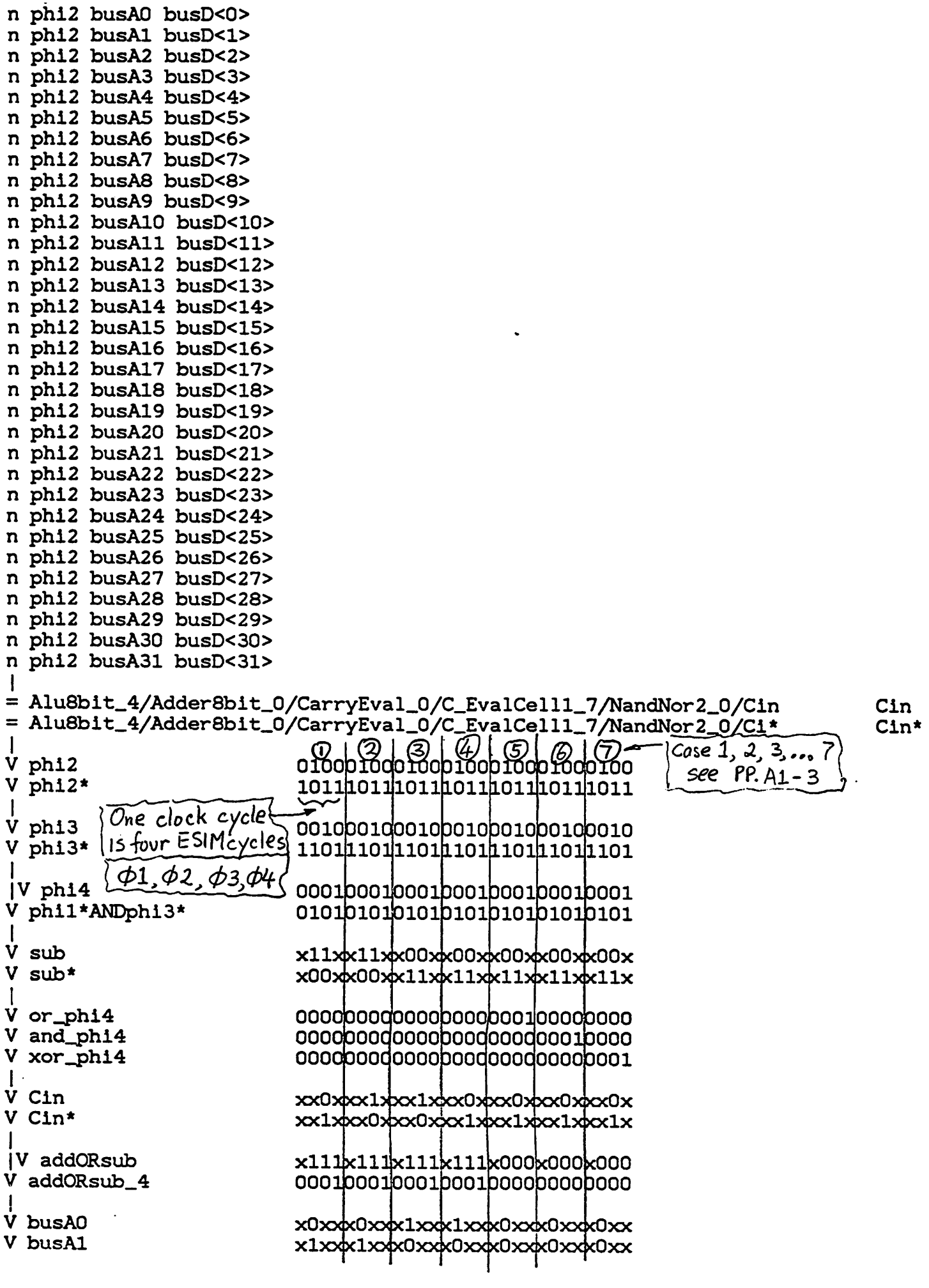
Figure A-1 Spice Simulation Of Critical Path In Phi2

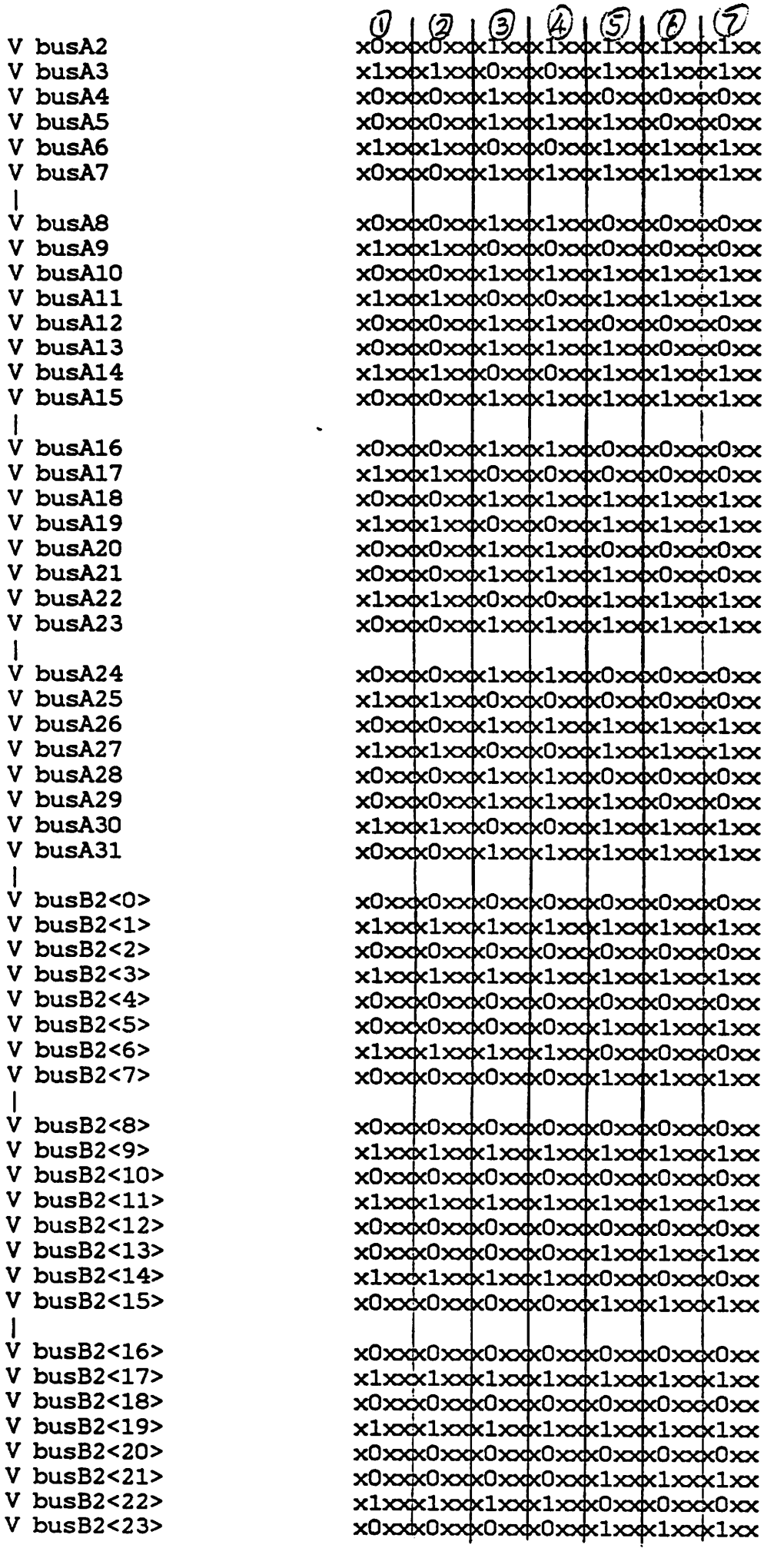


Node	Delay 50%	Delay 90%
G7	11.4ns	12.6ns
C32	29.2ns	31.6ns

Figure A-2 Spice Simulation Of Critical Path In Phi3

ESIM's PATCH FILE


n phi2 busA0 busD<0>
n phi2 busA1 busD<1>
n phi2 busA2 busD<2>
n phi2 busA3 busD<3>
n phi2 busA4 busD<4>
n phi2 busA5 busD<5>
n phi2 busA6 busD<6>
n phi2 busA7 busD<7>
n phi2 busA8 busD<8>
n phi2 busA9 busD<9>
n phi2 busA10 busD<10>
n phi2 busA11 busD<11>
n phi2 busA12 busD<12>
n phi2 busA13 busD<13>
n phi2 busA14 busD<14>
n phi2 busA15 busD<15>
n phi2 busA16 busD<16>
n phi2 busA17 busD<17>
n phi2 busA18 busD<18>
n phi2 busA19 busD<19>
n phi2 busA20 busD<20>
n phi2 busA21 busD<21>
n phi2 busA22 busD<22>
n phi2 busA23 busD<23>
n phi2 busA24 busD<24>
n phi2 busA25 busD<25>
n phi2 busA26 busD<26>
n phi2 busA27 busD<27>
n phi2 busA28 busD<28>
n phi2 busA29 busD<29>
n phi2 busA30 busD<30>
n phi2 busA31 busD<31>
= Alu8bit_4/Adder8bit_0/CarryEval_0/C_EvalCell1_7/NandNor2_0/Cin
Cin
= Alu8bit_4/Adder8bit_0/CarryEval_0/C_EvalCell1_7/NandNor2_0/Ci*
Cin*
① ② ③ ④ ⑤ ⑥ ⑦
Case 1, 2, 3, ... 7
see PP. A1-3
V phi2
V phi2*
One clock cycle
is four ESIMcycles
φ1, φ2, φ3, φ4
V phi3
V phi3*
V phi4
V phi1*ANDphi3*
V sub
V sub*
V or_phi4
V and_phi4
V xor_phi4
V Cin
V Cin*
V addORsub
V addORsub_4
V busA0
V busA1


① ② ③ ④ ⑤ ⑥ ⑦
V busA2 x0xxx0xxx1xxx1xxx1xxx1xxx1xx
V busA3 x1xxx1xxx0xxx0xxx1xxx1xxx1xx
V busA4 x0xxx0xxx1xxx1xxx0xxx0xxx0xx
V busA5 x0xxx0xxx1xxx1xxx1xxx0xxx0xx
V busA6 x1xxx1xxx0xxx0xxx1xxx1xxx1xx
V busA7 x0xxx0xxx1xxx1xxx1xxx1xxx1xx
V busA8 x0xxx0xxx1xxx1xxx0xxx0xxx0xx
V busA9 x1xxx1xxx0xxx0xxx0xxx0xxx0xx
V busA10 x0xxx0xxx1xxx1xxx1xxx1xxx1xx
V busA11 x1xxx1xxx0xxx0xxx1xxx1xxx1xx
V busA12 x0xxx0xxx1xxx1xxx0xxx0xxx0xx
V busA13 x0xxx0xxx1xxx1xxx1xxx0xxx0xx
V busA14 x1xxx1xxx0xxx0xxx1xxx1xxx1xx
V busA15 x0xxx0xxx1xxx1xxx1xxx1xxx1xx
V busA16 x0xxx0xxx1xxx1xxx0xxx0xxx0xx
V busA17 x1xxx1xxx0xxx0xxx0xxx0xxx0xx
V busA18 x0xxx0xxx1xxx1xxx1xxx1xxx1xx
V busA19 x1xxx1xxx0xxx0xxx1xxx1xxx1xx
V busA20 x0xxx0xxx1xxx1xxx0xxx0xxx0xx
V busA21 x0xxx0xxx1xxx1xxx1xxx0xxx0xx
V busA22 x1xxx1xxx0xxx0xxx1xxx1xxx1xx
V busA23 x0xxx0xxx1xxx1xxx1xxx1xxx1xx
V busA24 x0xxx0xxx1xxx1xxx0xxx0xxx0xx
V busA25 x1xxx1xxx0xxx0xxx0xxx0xxx0xx
V busA26 x0xxx0xxx1xxx1xxx1xxx1xxx1xx
V busA27 x1xxx1xxx0xxx0xxx1xxx1xxx1xx
V busA28 x0xxx0xxx1xxx1xxx0xxx0xxx0xx
V busA29 x0xxx0xxx1xxx1xxx1xxx0xxx0xx
V busA30 x1xxx1xxx0xxx0xxx1xxx1xxx1xx
V busA31 x0xxx0xxx1xxx1xxx1xxx1xxx1xx
V busB2<0> x0xxx0xxx0xxx0xxx0xxx0xxx0xx
V busB2<1> x1xxx1xxx1xxx1xxx1xxx1xxx1xx
V busB2<2> x0xxx0xxx0xxx0xxx0xxx0xxx0xx
V busB2<3> x1xxx1xxx1xxx1xxx1xxx1xxx1xx
V busB2<4> x0xxx0xxx0xxx0xxx0xxx0xxx0xx
V busB2<5> x0xxx0xxx0xxx0xxx1xxx1xxx1xx
V busB2<6> x1xxx1xxx1xxx1xxx0xxx0xxx0xx
V busB2<7> x0xxx0xxx0xxx0xxx1xxx1xxx1xx
V busB2<8> x0xxx0xxx0xxx0xxx0xxx0xxx0xx
V busB2<9> x1xxx1xxx1xxx1xxx1xxx1xxx1xx
V busB2<10> x0xxx0xxx0xxx0xxx0xxx0xxx0xx
V busB2<11> x1xxx1xxx1xxx1xxx1xxx1xxx1xx
V busB2<12> x0xxx0xxx0xxx0xxx0xxx0xxx0xx
V busB2<13> x0xxx0xxx0xxx0xxx1xxx1xxx1xx
V busB2<14> x1xxx1xxx1xxx1xxx0xxx0xxx0xx
V busB2<15> x0xxx0xxx0xxx0xxx1xxx1xxx1xx
V busB2<16> x0xxx0xxx0xxx0xxx0xxx0xxx0xx
V busB2<17> x1xxx1xxx1xxx1xxx1xxx1xxx1xx
V busB2<18> x0xxx0xxx0xxx0xxx0xxx0xxx0xx
V busB2<19> x1xxx1xxx1xxx1xxx1xxx1xxx1xx
V busB2<20> x0xxx0xxx0xxx0xxx0xxx0xxx0xx
V busB2<21> x0xxx0xxx0xxx0xxx1xxx1xxx1xx
V busB2<22> x1xxx1xxx1xxx1xxx0xxx0xxx0xx
V busB2<23> x0xxx0xxx0xxx0xxx1xxx1xxx1xx


|
V busB2<24>
V busB2<25>
V busB2<26>
V busB2<27>
V busB2<28>
V busB2<29>
V busB2<30>
V busB2<31>
|
w busD<0>
w busD<1>
w busD<2>
w busD<3>
w busD<4>
w busD<5>
w busD<6>
w busD<7>
w busD<8>
w busD<9>
w busD<10>
w busD<11>
w busD<12>
w busD<13>
w busD<14>
w busD<15>
w busD<16>
w busD<17>
w busD<18>
w busD<19>
w busD<20>
w busD<21>
w busD<22>
w busD<23>
w busD<24>
w busD<25>
w busD<26>
w busD<27>
w busD<28>
w busD<29>
w busD<30>
w busD<31>
|
w Cout Cout*
1
2
3
4
5
6
7
x0xxx0xxx0xxx0xxx0xxx0xxx0xx
x1xxx1xxx1xxx1xxx1xxx1xxx1xx
x0xxx0xxx0xxx0xxx0xxx0xxx0xx
x1xxx1xxx1xxx1xxx1xxx1xxx1xx
x0xxx0xxx0xxx0xxx0xxx0xxx0xx
x0xxx0xxx0xxx0xxx1xxx1xxx1xx
x1xxx1xxx1xxx1xxx0xxx0xxx0xx
x0xxx0xxx0xxx0xxx1xxx1xxx1xx

ESIM's OUTPUT FILE

```
Script started on Mon Dec 17 15:29:56 1984
renoir1> kesim
sim> @ ALU.sim
sim> @ ALU.al
sim> @ ALU.patch
sim> I
initialization took 2733 steps
sim> I
initialization took 5 steps
sim> I
initialization took 0 steps
sim> R
>1011101010111011111010101010:busD<0>
>1111111010101011101110101011:busD<1>
>1011101010111011111111111111:busD<2>
>1111111010101011111111111110:busD<3>
>1011101010111011111010101010:busD<4>
>1011101010111011111111101011:busD<5>
>1111111010101011111111101111:busD<6>
>1011101010111011111111111110:busD<7>
>1011101010111011111010101010:busD<8>
>1111111010101011101110101011:busD<9>
>1011101010111011111111111111:busD<10>
>1111111010101011111111111110:busD<11>
>1011101010111011111010101010:busD<12>
>1011101010111011111111101011:busD<13>
>1111111010101011111111101111:busD<14>
>1011101010111011111111111110:busD<15>
>1011101010111011111010101010:busD<16>
>1111111010101011101110101011:busD<17>
>1011101010111011111111101111:busD<18>
>1111111010101011111111111110:busD<19>
>1011101010111011111010101010:busD<20>
>1011101010111011111111101011:busD<21>
>1111111010101011111111101111:busD<22>
>1011101010111011111111111110:busD<23>
>1011101010111011111010101010:busD<24>
>1111111010101011101110101011:busD<25>
>1011101010111011111111101111:busD<26>
>1111111010101011111111111110:busD<27>
>1011101010111011111010101010:busD<28>
>1011101010111011111111101011:busD<29>
>1111111010101011111111101111:busD<30>
>1011101010111011111111111110:busD<31>
>0000001000100000001000100010:Cout
>1111110111011111110111011101:Cout*
sim> q
renoir2> exit
renoir3>
script done on Mon Dec 17 15:34:14 1984
```

Case 1, 2, 3 ... 7
see PP. A1–A3

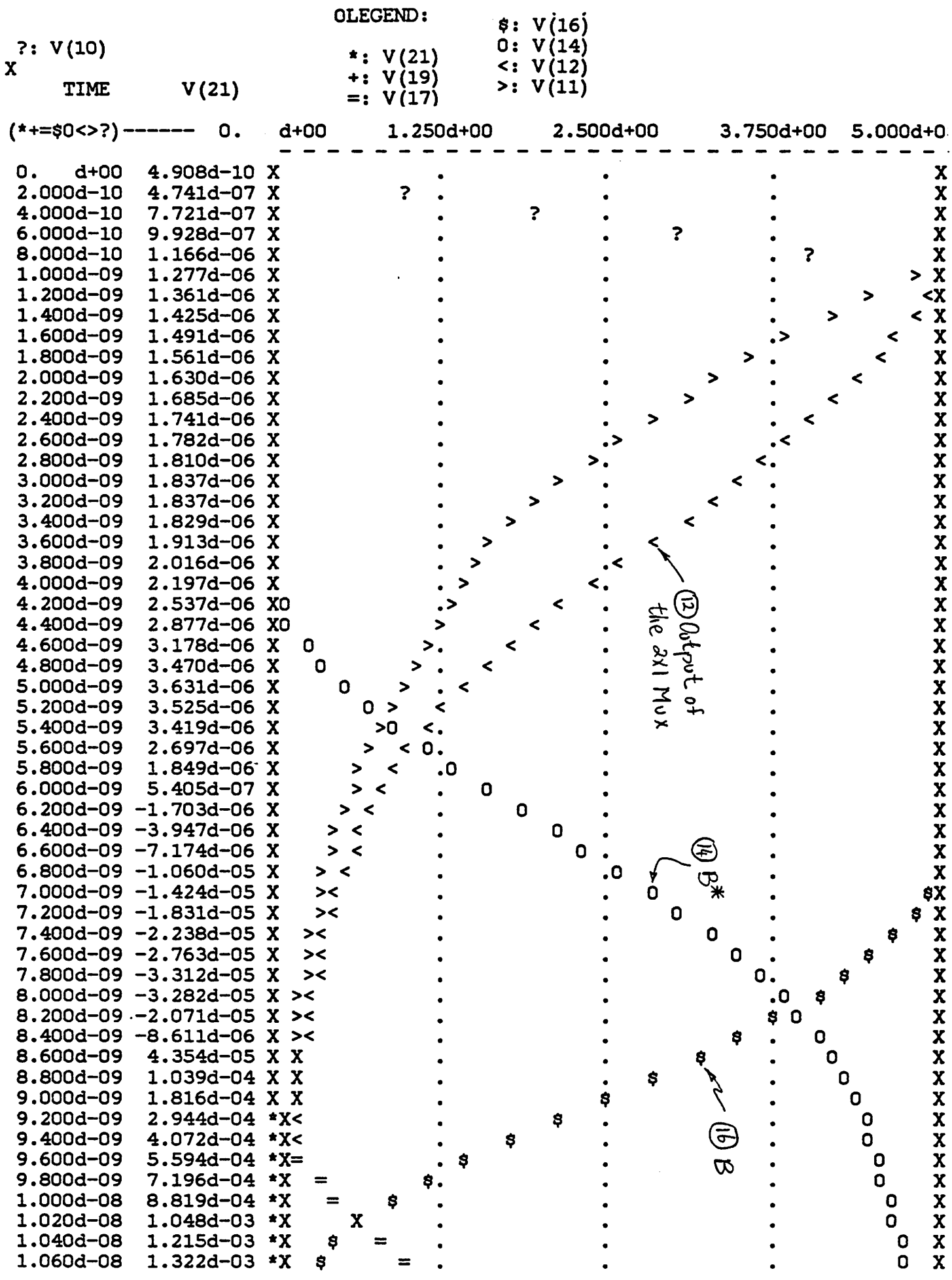
① ② ③ ④ ⑤ ⑥ ⑦

$\phi 1, \phi 2, \phi 3, \phi 4$

SPICE'S OUTPUT - CRITICAL PATH IN PHI2

```
1*******12/07/84 ********  SPICE 2G.6     3/15/83 ********04:12:21*****

0CRITICAL PATH FOR ALUINCELL CIRCUIT

0****      INPUT LISTING                          TEMPERATURE =    27.000 DEG C

0******************************************************************************


 M1 11 10 1 1 CMOSP W=6U L=3U AD=54P AS=54P PD=24U PS=24U
 M2 11 10 0 0 CMOSN W=3U L=3U AD=45P AS=45P PD=24U PS=24U
 *
 C11 11 0 0.002P
 * PASS TRANSISTORS OF 2X1 MUX
 M3 11 0 12 1 CMOSP W=6U L=3U AD=54P AS=54P PD=24U PS=24U
 M4 11 1 12 0 CMOSN W=6U L=3U AD=54P AS=54P PD=24U PS=24U
 *
 C12 12 0 0.002P
 *
 M5 13 12 1 1 CMOSP W=9U L=3U AD=13.5P AS=81P PD=3U PS=27U
 M6 14 0 13 1 CMOSP W=9U L=3U AD=81P AS=13.5P PD=27U PS=3U
 M7 14 1 15 0 CMOSN W=4.5U L=3U AD=49.5P AS=6.75P PD=24U PS=3U
 M8 15 12 0 0 CMOSN W=4.5U L=3U AD=6.75P AS=49.5P PD=3U PS=24U
 *
 C14 14 0 0.002P
 *
 M9 16 14 1 1 CMOSP W=6U L=3U AD=54P AS=54P PD=24U PS=24U
 M10 16 14 0 0 CMOSN W=3U L=3U AD=45P AS=45P PD=24U PS=24U
 *
 C16 16 0 0.002P
 *
 M11 17 16 1 1 CMOSP W=6U L=3U AD=54P AS=54P PD=24U PS=24U
 M12 17 1 1 1 CMOSP W=6U L=3U AD=54P AS=54P PD=24U PS=24U
 M13 17 1 18 0 CMOSN W=6U L=3U AD=54P AS=54P PD=24U PS=24U
 M14 18 16 0 0 CMOSN W=6U L=3U AD=54P AS=54P PD=24U PS=24U
 *
 C17 17 0 0.002P
 *
 M15 19 17 1 1 CMOSP W=6U L=3U AD=54P AS=54P PD=24U PS=24U
 M16 19 1 1 1 CMOSP W=6U L=3U AD=54P AS=54P PD=24U PS=24U
 M17 19 1 20 0 CMOSN W=6U L=3U AD=54P AS=54P PD=24U PS=24U
 M18 20 17 0 0 CMOSN W=6U L=3U AD=54P AS=54P PD=24U PS=24U
 *
 C19 19 0 0.002P
 *
 M19 21 19 1 1 CMOSP W=9U L=3U AD=81P AS=81P PD=27U PS=27U
 M20 21 19 0 0 CMOSN W=6U L=3U AD=54P AS=54P PD=24U PS=24U
 *
 CL 21 0 0.05P
 *
 .MODEL CMOSN NMOS LEVEL=2.00000 LD=0.245423U TOX=500.000E-10
 +NSUB=1.0000000E+16 VTO=0.932797 KP=2.696667E-05 GAMMA=1.280470
 +PHI=0.600000 UO=381.905 UEXP=1.001000E-03 UCRIT=999000.
 +DELTA=1.47242 VMAX=55346.1 XJ=0.145596U LAMBDA=2.491255E-02
 +NFS=3.727796E+12 NEFF=1.001000E-02 NSS=0.000000E+00 TPG=1.00000
 +RSH=25 CGSO=5.2E-10 CGDO=5.2E-10 CJ=3.2E-4 MJ=0.5 CJSW=9E-10 MJSW=0.33
 *
 .MODEL CMOSP PMOS LEVEL=2.00000 LD=0.512860U TOX=500.000E-10
 +NSUB=2.971614E+14 VTO=-0.844293 KP=1.048805E-05 GAMMA=0.723071
```

```
 +PHI=0.600000 UO=100.000 UEXP=0.145531 UCRIT=18543.6
 +DELTA=2.19030 VMAX=100000. XJ=2.583588E-08 LAMBDA=5.274834E-02
 +NFS=1.615644E+12 NEFF=1.001000E-02 NSS=0.000000E+00 TPG=-1.00000
 +RSH=95 CGSO=4E-10 CGDO=4E-10 CJ=2E-4 MJ=0.5 CJSW=4.5E-10 MJSW=0.33
 *
 VDD 1 0 5
 VIN 10 0 PULSE (0 5 0N 1N 1N)
 .WIDTH OUT=80
 .IC V(11)=5 V(12)=5 V(13)=1 V(14)=0 V(15)=0 V(16)=5 V(17)=0 V(18)=0
 + V(19)=5 V(20)=4 V(21)=0
 .TRAN 0.2N 25N
 .PRINT TRAN V(21) V(19) V(17) V(16) V(14) V(12) V(11) (0,5)
 .PLOT TRAN V(21) V(19) V(17) V(16) V(14) V(12) V(11) V(10) (0,5)
 .END
1*******12/07/84 ********  SPICE 2G.6    3/15/83 ********04:12:21*****

0CRITICAL PATH FOR ALUINCELL CIRCUIT

0****     MOSFET MODEL PARAMETERS          TEMPERATURE =   27.000 DEG C

0******************************************************************************



              CMOSN      CMOSP
0TYPE         NMOS       PMOS
0LEVEL          2.000      2.000
0VTO            0.933     -0.844
0KP          2.70d-05   1.05d-05
0GAMMA          1.280      0.723
0PHI            0.600      0.600
0LAMBDA      2.49d-02   5.27d-02
0CGSO        5.20d-10   4.00d-10
0CGDO        5.20d-10   4.00d-10
0RSH           25.000     95.000
0CJ          3.20d-04   2.00d-04
0MJ             0.500      0.500
0CJSW        9.00d-10   4.50d-10
0MJSW           0.330      0.330
0TOX         5.00d-08   5.00d-08
0NSUB        1.00d+16   2.97d+14
0NSS         0.  d+00   0.  d+00
0NFS         3.73d+12   1.62d+12
0TPG            1.000     -1.000
0XJ          1.46d-07   2.58d-08
0LD          2.45d-07   5.13d-07
0UO           381.905    100.000
0UCRIT       9.99d+05   1.85d+04
0UEXP           0.001      0.146
0VMAX        5.53d+04   1.00d+05
0NEFF           0.010      0.010
0DELTA          1.472      2.190
1*******12/07/84 ********  SPICE 2G.6    3/15/83 ********04:12:21*****

0CRITICAL PATH FOR ALUINCELL CIRCUIT

0****     INITIAL TRANSIENT SOLUTION       TEMPERATURE =   27.000 DEG C

0******************************************************************************
```

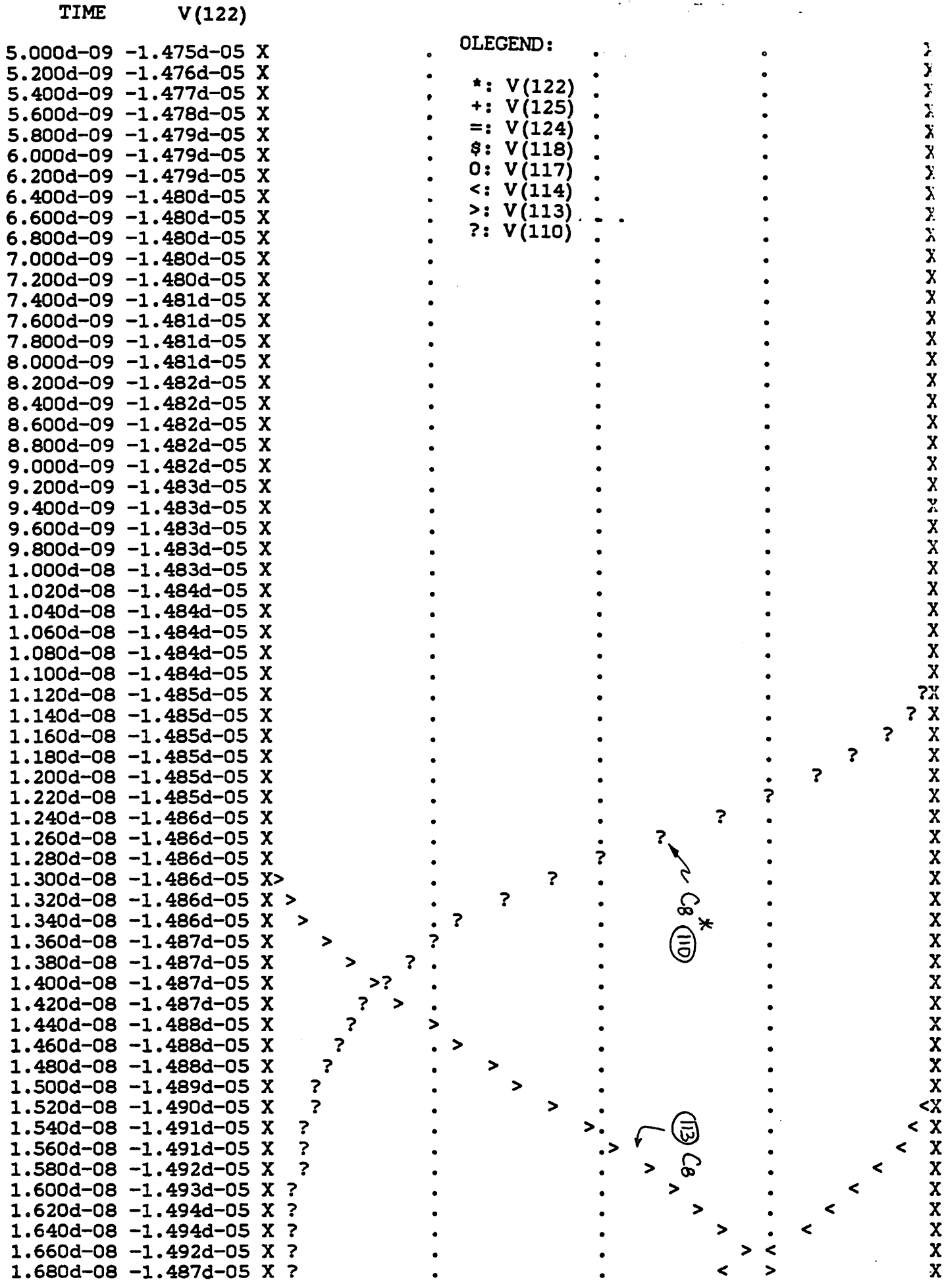

OLEGEND:
?: V(10)
*: V(21)
+: V(19)
=: V(17)
$: V(16)
0: V(14)
<: V(12)
>: V(11)
TIME
V(21)
(*+=$0<>?)------ 0. d+00 1.250d+00 2.500d+00 3.750d+00 5.000d+0
⑫ Output of the 2x1 Mux
⑭ B*
⑯ B

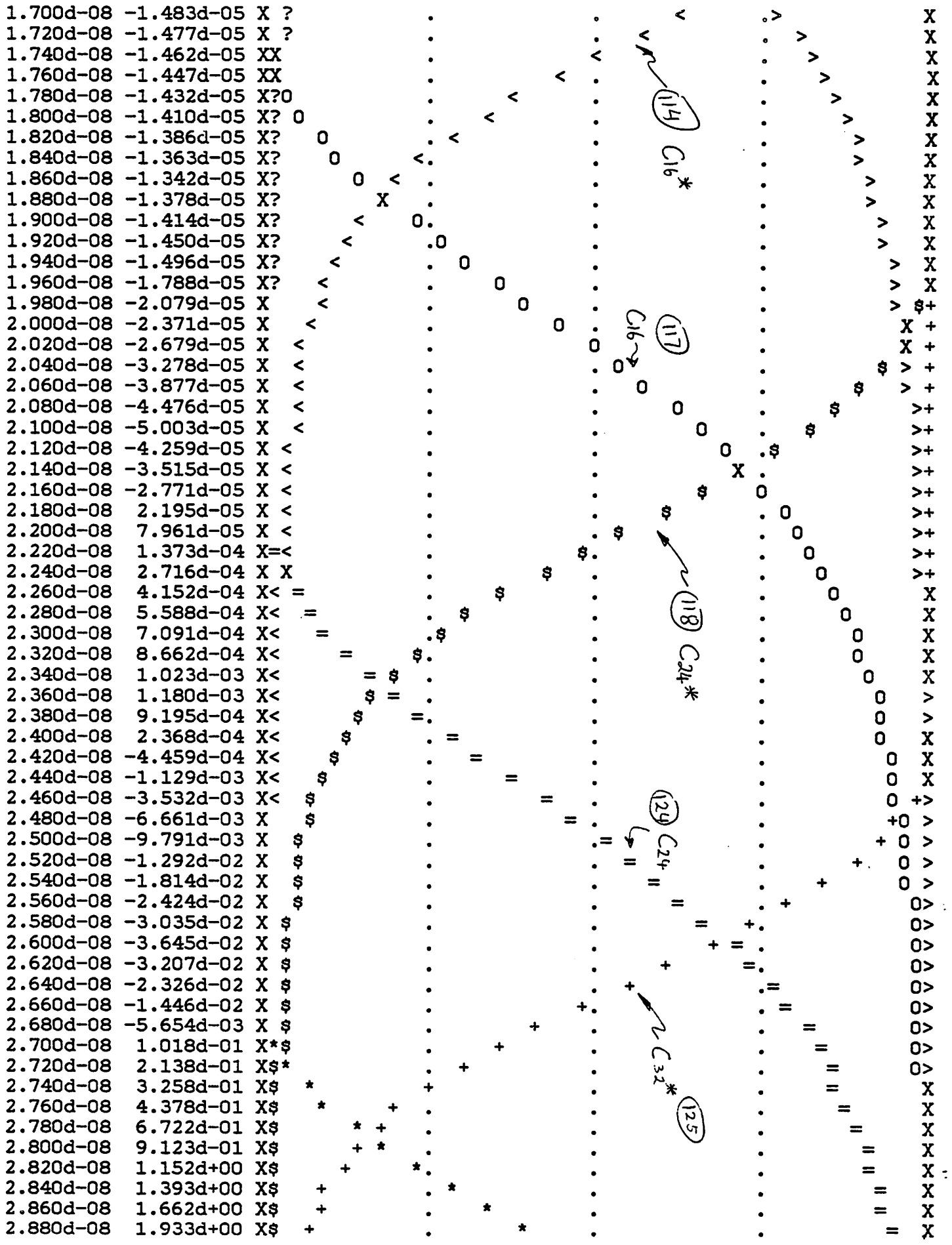
1.080d-08 1.418d-03 *X $
1.100d-08 1.492d-03 *X$
1.120d-08 1.524d-03 X<$
1.140d-08 1.557d-03 X<$
1.160d-08 1.573d-03 XX
1.180d-08 1.587d-03 X$
1.200d-08 1.570d-03 X$
1.220d-08 1.493d-03 X$
1.240d-08 1.415d-03 X$
1.260d-08 -8.408d-04 X$
1.280d-08 -3.545d-03 X$
1.300d-08 -8.220d-03 X$
1.320d-08 -1.691d-02 X$
1.340d-08 -2.559d-02 X$
1.360d-08 -3.685d-02 X$
1.380d-08 -4.863d-02 X
1.400d-08 -5.518d-02 X
1.420d-08 -5.106d-02 X
1.440d-08 -4.693d-02 X
1.460d-08 -3.752d-03 X
1.480d-08 4.746d-02 X*
1.500d-08 1.320d-01 X*
1.520d-08 2.843d-01 X
1.540d-08 4.366d-01 X
1.560d-08 6.944d-01 X
1.580d-08 9.739d-01 X
1.600d-08 1.282d+00 X
1.620d-08 1.650d+00 X
1.640d-08 2.017d+00 X
1.660d-08 2.393d+00 X
1.680d-08 2.770d+00 X
1.700d-08 3.123d+00 X
1.720d-08 3.427d+00 X
1.740d-08 3.731d+00 X
1.760d-08 3.952d+00 X
1.780d-08 4.156d+00 X
1.800d-08 4.334d+00 X
1.820d-08 4.457d+00 X
1.840d-08 4.580d+00 X
1.860d-08 4.660d+00 X
1.880d-08 4.731d+00 X
1.900d-08 4.791d+00 X+
1.920d-08 4.831d+00 X+
1.940d-08 4.870d+00 X+
1.960d-08 4.895d+00 X+
1.980d-08 4.917d+00 X+
2.000d-08 4.935d+00 X+
2.020d-08 4.947d+00 X+
2.040d-08 4.960d+00 X
2.060d-08 4.967d+00 X
2.080d-08 4.974d+00 X
2.100d-08 4.980d+00 X
2.120d-08 4.983d+00 X
2.140d-08 4.987d+00 X
2.160d-08 4.989d+00 X
2.180d-08 4.991d+00 X
2.200d-08 4.993d+00 X
2.220d-08 4.994d+00 X
2.240d-08 4.996d+00 X
2.260d-08 4.996d+00 X
(17) (A·B)
(19) A xor B
(21) A xor B
(A xor B)'s delay measured at 50% point: 16.6ns
(A xor B)'s delay measured at 90% point: 18.4ns

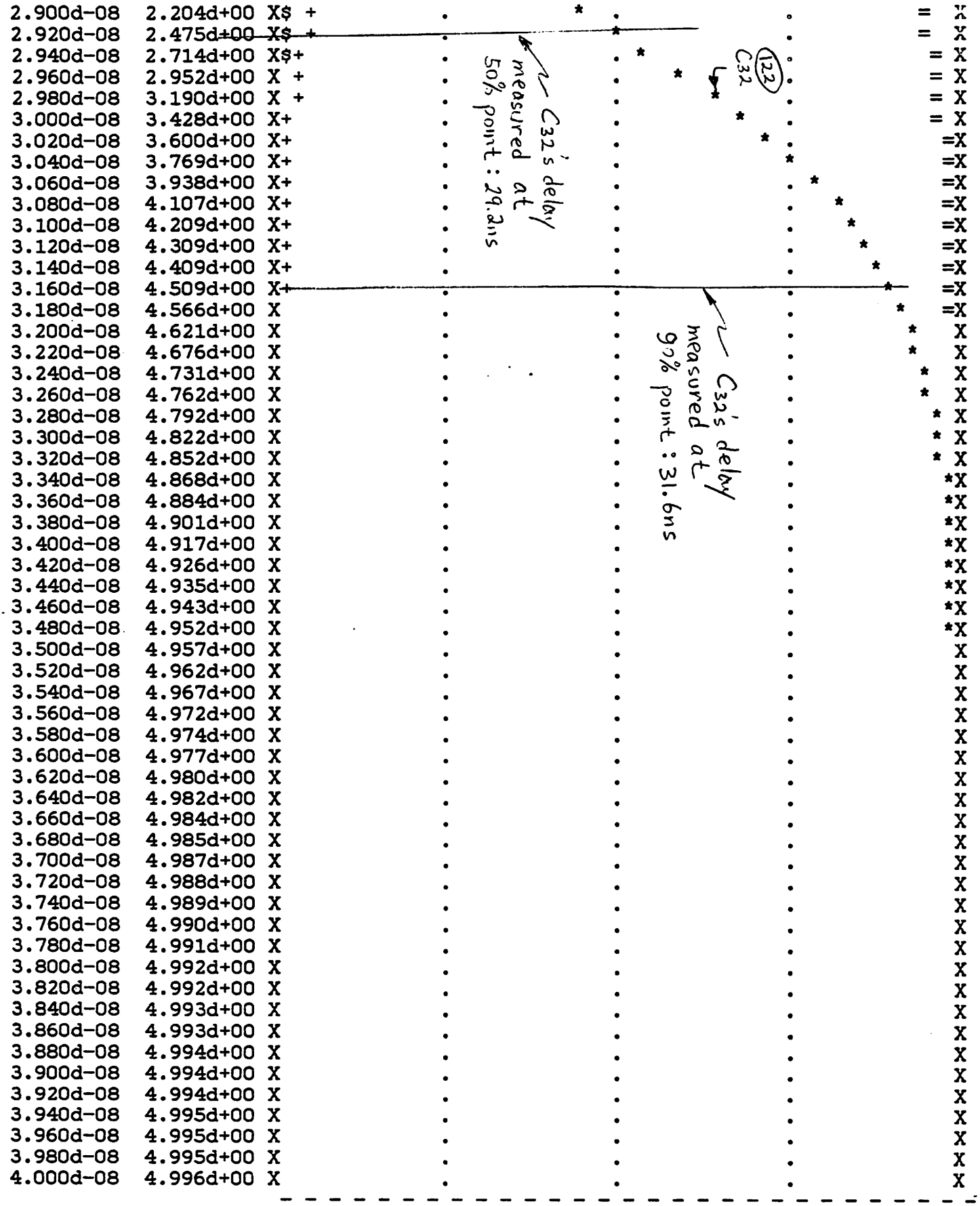
SPICE'S OUTPUT - CRITICAL PATH IN PHI3

```
1*******11/08/84 ********  SPICE 2G.6    3/15/83 ********12:38:58*****

0OVERALL CRITICAL PATH FOR 8 BIT LOOKAHEAD

0****     INPUT LISTING                          TEMPERATURE =   27.000 DEG C

0******************************************************************************


 *
 * CRITICAL PATH FOR LOOKAHEAD CIRCUIT
 M1 10 3 11 0 CMOSN W=9U L=3U AD=81P AS=13.5P PD=27U PS=3U
 M2 11 1 12 0 CMOSN W=9U L=3U AD=13.5P AS=13.5P PD=3U PS=3U
 M3 12 1 0 0 CMOSN W=9U L=3U AD=13.5P AS=81P PD=3U PS=27U
 *
 C11 10 0 0.00297P
 *
 M4 13 10 1 1 CMOSP W=6U L=3U AD=54P AS=54P PD=24U PS=24U
 M5 13 10 0 0 CMOSN W=3U L=3U AD=45P AS=45P PD=24U PS=24U
 *
 C1 13 0 0.0396P
 *
 M6 14 13 15 0 CMOSN W=9U L=3U AD=81P AS=13.5P PD=27U PS=3U
 M7 15 1 16 0 CMOSN W=9U L=3U AD=13.5P AS=13.5P PD=3U PS=3U
 M8 16 1 0 0 CMOSN W=9U L=3U AD=13.5P AS=81P PD=3U PS=27U
 *
 C12 14 0 0.00297P
 *
 M9 17 14 1 1 CMOSP W=9U L=3U AD=81P AS=81P PD=27U PS=27U
 M10 17 14 0 0 CMOSN W=4.5U L=3U AD=49.5P AS=49.5P PD=24U PS=24U
 *
 C2 17 0 0.13514P
 *
 M11 18 17 19 0 CMOSN W=9U L=3U AD=81P AS=13.5P PD=27U PS=3U
 M12 19 1 20 0 CMOSN W=9U L=3U AD=13.5P AS=13.5P PD=3U PS=3U
 M13 20 1 0 0 CMOSN W=9U L=3U AD=13.5P AS=81P PD=3U PS=27U
 *
 C13 18 0 0.00297P
 *
 M14 22 18 1 1 CMOSP W=9U L=3U AD=81P AS=81P PD=27U PS=27U
 * M15 22 0 21 1 CMOSP W=9U L=3U AD=81P AS=13.5P PD=27U PS=3U
 * M16 22 1 23 0 CMOSN W=4.5U L=3U AD=49.5P AS=6.75P PD=24U PS=3U
 M17 22 18 0 0 CMOSN W=4.5U L=3U AD=49.5P AS=49.5P PD=24U PS=24U
 *
 * CRITICAL PATH FOR CARRY EVALUATION
 M101 110 22 111 0 CMOSN W=9U L=3U AD=81P AS=13.5P PD=27U PS=3U
 M102 111 1 112 0 CMOSN W=9U L=3U AD=13.5P AS=13.5P PD=3U PS=3U
 M103 112 1 0 0 CMOSN W=9U L=3U AD=13.5P AS=81P PD=3U PS=27U
 *
 CY1 110 0 0.00297P
 *
 M104 113 110 1 1 CMOSP W=18U L=3U AD=135P AS=135P PD=33U PS=33U
 M105 113 110 0 0 CMOSN W=3U L=3U AD=45P AS=45P PD=24U PS=24U
 *
 CX1 113 0 0.28893P
 *
 M106 114 113 115 0 CMOSN W=9U L=3U AD=81P AS=13.5P PD=27U PS=3U
 M107 115 1 116 0 CMOSN W=9U L=3U AD=13.5P AS=13.5P PD=3U PS=3U
 M108 116 1 0 0 CMOSN W=9U L=3U AD=13.5P AS=81P PD=3U PS=27U
```

```
*
CY2 114 0 0.00297P
*
M109 117 114 1 1 CMOSP W=18U L=3U AD=135P AS=135P PD=33U PS=33U
M110 117 114 0 0 CMOSN W=4.5U L=3U AD=49.5P AS=49.5P PD=24U PS=24U
*
CX2 117 0 0.28893P
*
M111 118 117 119 0 CMOSN W=9U L=3U AD=81P AS=13.5P PD=27U PS=3U
M112 119 1 120 0 CMOSN W=9U L=3U AD=13.5P AS=13.5P PD=3U PS=3U
M113 120 1 0 0 CMOSN W=9U L=3U AD=13.5P AS=81P PD=3U PS=27U
*
CY3 118 0 0.00297P
*
M118 124 118 1 1 CMOSP W=18U L=3U AD=135P AS=135P PD=33U PS=33U
M119 124 118 0 0 CMOSN W=4.5U L=3U AD=49.5P AS=49.5P PD=24U PS=24U
*
CX3 124 0 0.28893P
*
M120 125 124 126 0 CMOSN W=9U L=3U AD=81P AS=13.5P PD=27U PS=3U
M121 126 1 127 0 CMOSN W=9U L=3U AD=13.5P AS=13.5P PD=3U PS=3U
M122 127 1 0 0 CMOSN W=9U L=3U AD=13.5P AS=81P PD=3U PS=27U
*
C14 125 0 0.00297P
*
M114 121 125 1 1 CMOSP W=9U L=3U AD=13.5P AS=81P PD=3U PS=27U
M115 122 0 121 1 CMOSP W=9U L=3U AD=81P AS=13.5P PD=27U PS=3U
M116 122 1 123 0 CMOSN W=4.5U L=3U AD=49.5P AS=6.75P PD=24U PS=3U
M117 123 125 0 0 CMOSN W=4.5U L=3U AD=6.75P AS=49.5P PD=3U PS=24U
*
CL 122 0 0.022P
*
.MODEL CMOSN NMOS LEVEL=2.00000 LD=0.245423U TOX=500.000E-10
+NSUB=1.0000000E+16 VTO=0.932797 KP=2.696667E-05 GAMMA=1.280470
+PHI=0.600000 UO=381.905 UEXP=1.001000E-03 UCRIT=999000.
+DELTA=1.47242 VMAX=55346.1 XJ=0.145596U LAMBDA=2.491255E-02
+NFS=3.727796E+12 NEFF=1.001000E-02 NSS=0.000000E+00 TPG=1.00000
+RSH=25 CGSO=5.2E-10 CGDO=5.2E-10 CJ=3.2E-4 MJ=0.5 CJSW=9E-10 MJSW=0.33
*
.MODEL CMOSP PMOS LEVEL=2.00000 LD=0.512860U TOX=500.000E-10
+NSUB=2.971614E+14 VTO=-0.844293 KP=1.048805E-05 GAMMA=0.723071
+PHI=0.600000 UO=100.000 UEXP=0.145531 UCRIT=18543.6
+DELTA=2.19030 VMAX=100000. XJ=2.583588E-08 LAMBDA=5.274834E-02
+NFS=1.615644E+12 NEFF=1.001000E-02 NSS=0.000000E+00 TPG=-1.00000
+RSH=95 CGSO=4E-10 CGDO=4E-10 CJ=2E-4 MJ=0.5 CJSW=4.5E-10 MJSW=0.33
*
VDD 1 0 5
VIN 3 0 PULSE (0 5 0N 1N 1N)
.WIDTH OUT=80
.IC V(110)=5 V(111)=0 V(112)=0 V(113)=0 V(115)=0 V(116)=0 V(114)=5 V(117)=0
+ V(118)=5 V(119)=0 V(120)=0  V(122)=0 V(124)=0 V(125)=5
+ V(126)=0 V(127)=0
+ V(10)=5 V(11)=0 V(12)=0 V(13)=0 V(15)=0 V(16)=0 V(14)=5 V(17)=0 V(18)=5
+ V(19)=0 V(20)=0 V(21)=0.7 V(22)=0 V(23)=0
.TRAN 0.2N 40N
.PRINT TRAN V(3) V(10) V(13) V(14) V(17)
.PRINT TRAN V(18) V(22)  V(110) V(113) V(114)
.PRINT TRAN V(117) V(118) V(124) V(125) V(122)
* .PLOT TRAN V(22) V(18) V(17) V(14) V(13) V(10) V(3) (0,5)
.PLOT TRAN V(122) V(125) V(124) V(118) V(117) V(114) V(113) V(110) (0,5)
```


TIME V(122)
LEGEND:
*: V(122)
+: V(125)
=: V(124)
$: V(118)
O: V(117)
<: V(114)
>: V(113)
?: V(110)
C8
(110)
(113)
C8


1.700d-08 -1.483d-05 X ?
1.720d-08 -1.477d-05 X ?
1.740d-08 -1.462d-05 XX
1.760d-08 -1.447d-05 XX
1.780d-08 -1.432d-05 X?O
1.800d-08 -1.410d-05 X? O
1.820d-08 -1.386d-05 X? O
1.840d-08 -1.363d-05 X? O
1.860d-08 -1.342d-05 X? O
1.880d-08 -1.378d-05 X? X
1.900d-08 -1.414d-05 X? O
1.920d-08 -1.450d-05 X? O
1.940d-08 -1.496d-05 X? O
1.960d-08 -1.788d-05 X? O
1.980d-08 -2.079d-05 X O
2.000d-08 -2.371d-05 X O
2.020d-08 -2.679d-05 X O
2.040d-08 -3.278d-05 X O
2.060d-08 -3.877d-05 X O
2.080d-08 -4.476d-05 X O
2.100d-08 -5.003d-05 X O
2.120d-08 -4.259d-05 X O
2.140d-08 -3.515d-05 X O
2.160d-08 -2.771d-05 X O
2.180d-08 2.195d-05 X O
2.200d-08 7.961d-05 X O
2.220d-08 1.373d-04 X=O
2.240d-08 2.716d-04 X X O
2.260d-08 4.152d-04 X= O
2.280d-08 5.588d-04 X= O
2.300d-08 7.091d-04 X= O
2.320d-08 8.662d-04 X= O
2.340d-08 1.023d-03 X= O
2.360d-08 1.180d-03 X= O
2.380d-08 9.195d-04 X= O
2.400d-08 2.368d-04 X= O
2.420d-08 -4.459d-04 X= O
2.440d-08 -1.129d-03 X= O
2.460d-08 -3.532d-03 X= O
2.480d-08 -6.661d-03 X= O
2.500d-08 -9.791d-03 X= O
2.520d-08 -1.292d-02 X= O
2.540d-08 -1.814d-02 X= O
2.560d-08 -2.424d-02 X= O
2.580d-08 -3.035d-02 X= O
2.600d-08 -3.645d-02 X= O
2.620d-08 -3.207d-02 X= O
2.640d-08 -2.326d-02 X= O
2.660d-08 -1.446d-02 X= O
2.680d-08 -5.654d-03 X= O
2.700d-08 1.018d-01 X= O
2.720d-08 2.138d-01 X= O
2.740d-08 3.258d-01 X= X
2.760d-08 4.378d-01 X= X
2.780d-08 6.722d-01 X= X
2.800d-08 9.123d-01 X= X
2.820d-08 1.152d+00 X= X
2.840d-08 1.393d+00 X= X
2.860d-08 1.662d+00 X= X
2.880d-08 1.933d+00 X= X
(114) C16*
(117) C16
(118) C24*
(124) C24
(125) C32*


2.900d-08 2.204d+00 X$ +
2.920d-08 2.475d+00 X$ +
2.940d-08 2.714d+00 X$+
2.960d-08 2.952d+00 X +
2.980d-08 3.190d+00 X +
3.000d-08 3.428d+00 X+
3.020d-08 3.600d+00 X+
3.040d-08 3.769d+00 X+
3.060d-08 3.938d+00 X+
3.080d-08 4.107d+00 X+
3.100d-08 4.209d+00 X+
3.120d-08 4.309d+00 X+
3.140d-08 4.409d+00 X+
3.160d-08 4.509d+00 X+
3.180d-08 4.566d+00 X
3.200d-08 4.621d+00 X
3.220d-08 4.676d+00 X
3.240d-08 4.731d+00 X
3.260d-08 4.762d+00 X
3.280d-08 4.792d+00 X
3.300d-08 4.822d+00 X
3.320d-08 4.852d+00 X
3.340d-08 4.868d+00 X
3.360d-08 4.884d+00 X
3.380d-08 4.901d+00 X
3.400d-08 4.917d+00 X
3.420d-08 4.926d+00 X
3.440d-08 4.935d+00 X
3.460d-08 4.943d+00 X
3.480d-08 4.952d+00 X
3.500d-08 4.957d+00 X
3.520d-08 4.962d+00 X
3.540d-08 4.967d+00 X
3.560d-08 4.972d+00 X
3.580d-08 4.974d+00 X
3.600d-08 4.977d+00 X
3.620d-08 4.980d+00 X
3.640d-08 4.982d+00 X
3.660d-08 4.984d+00 X
3.680d-08 4.985d+00 X
3.700d-08 4.987d+00 X
3.720d-08 4.988d+00 X
3.740d-08 4.989d+00 X
3.760d-08 4.990d+00 X
3.780d-08 4.991d+00 X
3.800d-08 4.992d+00 X
3.820d-08 4.992d+00 X
3.840d-08 4.993d+00 X
3.860d-08 4.993d+00 X
3.880d-08 4.994d+00 X
3.900d-08 4.994d+00 X
3.920d-08 4.994d+00 X
3.940d-08 4.995d+00 X
3.960d-08 4.995d+00 X
3.980d-08 4.995d+00 X
4.000d-08 4.996d+00 X
C32's delay measured at 50% point: 29.2ns
C32's delay measured at 90% point: 31.6ns
(122)
C32
Y

B. A TYPICAL NMOS PROCESS

Included below are the Spice models for the enhancement mode transistor (enmos) and the depletion mode transistor (dnmos) in a typical NMOS four micron process.

```
.model enmos nmos vto=1.0 kp=17.2u gamma=.40 lambda=.01 cgdo=350p
+ cgso=350p cgbo=200p cj=1.3e-8 cjsw=350p tox=85n ld=.5u uo=350
+ ucrit=2.6e5 uexp=.23 vmax=4e4 level=2

.model dnmos nmos vto=-2.5 kp=18.0u gamma=.51 lambda=.015 cgdo=350p
+ cgso=350p cgbo=200p cj=1.6e-8 cjsw=350p tox=85n ld=.5u uo=366
+ ucrit=2.6e5 uexp=.23 vmax=3e4 level=2
```