4.29 Evaluate the benefits of the stack effect for subthreshold leakage by comparing I_1 and I_2 in Figure 4.29 (this problem is inspired by [Narendra01]). Assume all three transistors are identical and $\gamma = 0$, $n = 1$.

a) If the transistors suffer from no DIBL ($\eta = 0$), prove that $I_2/I_1 = 1/2$, just as you would expect if the transistors behaved as resistors.

b) Does increasing η increase or decrease I_1? By what fraction does I_1 change at room temperature if $\eta = 0.05$ and $V_{DD} = 1.8$ V?

c) Does increasing η increase or decrease I_2? By what fraction does I_2 change at room temperature if $\eta = 0.05$ and $V_{DD} = 1.8$ V?

d) Solve for I_2/I_1 and x as a function of $\Delta = \eta V_{DD}/v_T$ assuming $\Delta >> 1$.

e) Explain why the stack effect is most important for transistors with significant DIBL.

4.30 Consider a 5 mm long, 4 λ-wide metal2 wire in a 0.6 μm process. The sheet resistance is 0.08 $\Omega/\square$ and the capacitance is 0.2 fF/μm. Construct a 3-segment π-model for the wire.

4.31 A 10x unit-sized inverter drives a 2x inverter at the end of the 5 mm wire from Exercise 30. The gate capacitance is $C = 2$ fF/μm and the effective resistance is $R = 2.5$ k$\Omega \cdot \mu$m for nMOS transistors. Estimate the propagation delay using the Elmore delay model; neglect diffusion capacitance.

4.32 Find the best width and spacing to minimize the RC delay of a metal2 bus in the 180 nm process described in Table 4.8 if the pitch cannot exceed 1000 nm. Minimum width and spacing are 320 nm. First assume that neither adjacent bit is switching. How does your answer change if the adjacent bits may be switching?

4.33 Derive EQ(4.55)–(4.57). Assume the initial driver and final receiver are of the same size as the repeaters so the total delay is N times the delay of a segment. Neglect diffusion parasitics so each segment can be modeled as in Figure 4.82.

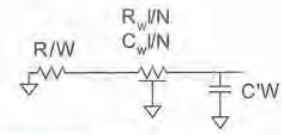


FIG 4.82 Model of repeater driving interconnect and next gate

TSMC-1041
U.S. Patent 8,587,076

4.34 Revisit Exercise 33 using pair of inverters (a noninverting buffer) instead of a single inverter. The first inverter in each pair is *W1* times unit width. The second is a factor of *k* larger than the first. Derive EQ (4.58)-(4.60).

4.35 Compute the characteristic velocity (delay per mm) of a repeated wire in the 180 nm process. A unit nMOS transistor has resistance of 2.5 KΩ and capacitance of 0.7 fF, and the pMOS has twice the resistance. Develop a table of results for metal1, metal2, and metal4 on minimum pitch and on double-pitch (twice minimum width and spacing). Use the data and pitches for the Intel 180 nm process. Assume solid metal above and below the wires and that the neighbors are not switching.

4.36 The Pentium 4 operated at 1.4 GHz in Intel's 180 nm process with an FO4 delay of 50 ps in 2001. If a new process is introduced every two years with a scaling factor of

$$S = \sqrt{2},$$

predict microprocessor clock frequencies in 2009, assuming all the improvement comes from ideal constant field scaling and that transistor speed sets clock rate. In practice, the frequency is likely to be higher. Why?

4.37 The path from the data cache to the register file of a microprocessor involves 500 ps of gate delay and 500 ps of wire delay along a repeated wire. The chip is scaled using constant field scaling and reduced height wires to a new generation with $S = 2$. Estimate the gate and wire delays of the path. By how much did the overall delay improve?



Circuit Simulation

5

5.1 Introduction

Fabricating chips is expensive and time-consuming, so designers need simulation tools to explore the design space and verify designs before they are fabricated. Simulators operate at many levels of abstraction, from process through architecture. *Process simulators* such as SUPREME predict how factors in the process recipe such as time and temperature affect device physical and electrical characteristics. *Circuit simulators* such as SPICE use device models and a circuit netlist to predict circuit voltages and currents, which indicate performance and power consumption. *Logic simulators* predict the function of digital circuits and are widely used to verify correct logical operation of designs specified in a hardware description language (HDL). *Architecture simulators* work at the level of instructions and registers to predict throughput and memory access patterns, which influence design decisions such as pipelining and cache memory organization. The various levels of abstraction offer tradeoffs between degree of detail and the size of the system that can be simulated. VLSI designers are primarily concerned with circuit and logic simulation. This chapter focuses on circuit simulation with SPICE. Section 9.3 discusses logic simulation.

Is it better to predict circuit behavior using paper-and-pencil analysis, as has been done in the previous chapters, or with simulation? VLSI circuits are complex and modern transistors have nonlinear, nonideal behavior, so simulation is necessary to accurately predict detailed circuit behavior. Even when closed-form solutions exist for delay or transfer characteristics, they are too time-consuming to apply by hand to large numbers of circuits. On the other hand, circuit simulation is notoriously prone to errors: *garbage in, garbage out* (GIGO). The simulator accepts the model of reality provided by the designer, but it is very easy to create a model that is inaccurate or incomplete. Moreover, the simulator only applies the stimulus provided by the designer and it is very easy to overlook the worst-case stimulus. In the same way that an experienced programmer doesn't expect a program to operate correctly before debugging, an experienced VLSI designer does not expect that the first run of a simulation will reflect reality. Therefore, the circuit designer needs to have a good intuitive understanding of circuit operation and should be able to predict the expected outcome before simulating. Only when expectation and simulation match can there be confidence in the results. In practice, circuit designers depend on both hand analysis and simulation, or as [Glasser85] puts it, "simulation guided through insight gained from analysis."

This chapter presents a brief SPICE tutorial by example. It then discusses models for transistors and diffusion capacitance. The remainder of the chapter is devoted to simulation techniques to characterize a process and to check performance, power, and correctness of circuits and interconnect.

5.2 A SPICE Tutorial

SPICE, a *Simulation Program with Integrated Circuit Emphasis*, was originally developed in the 1970s at Berkeley [Nagel75]. It solves the nonlinear differential equations describing components such as transistors, resistors, capacitors, and voltage sources. SPICE offers many ways to analyze circuits, but digital VLSI designers are primarily interested in *DC* and *transient* analysis that predicts the node voltages given inputs that are fixed or arbitrarily changing in time. SPICE was originally developed in FORTRAN and has some idiosyncrasies, particularly in file formats, related to its heritage. There are free versions of SPICE available on most platforms, but the commercial versions tend to offer more robust numerical convergence. In particular, HSPICE is widely used in industry because it converges well, supports the latest device and interconnect models, and has a large number of enhancements for measuring and optimizing circuits. PSPICE is another commercial version with a free limited student version. The examples throughout this section use HSPICE and generally will not run in ordinary SPICE.

While the details of using SPICE vary with version and platform, all versions of SPICE read an input file and generate a list file with results, warnings, and error messages. The input file is often called a *SPICE deck* and each line a *card* because it was once provided to a mainframe as a deck of punch cards. The input file contains a netlist consisting of components and nodes. It also contains simulation options, analysis commands, and device models. The netlist can be entered by hand or extracted from a circuit schematic or layout in a CAD program.

A good SPICE deck is like a good piece of software. It should be readable, maintainable, and reusable. Comments and white space help make the deck readable. Often the best way to write a SPICE deck is to start with a good deck that does nearly the right thing and then modify it.

The remainder of this section provides a sequence of examples illustrating the key syntax and capabilities of SPICE for digital VLSI circuits. For more detail, consult the Berkeley SPICE manual [Johnson91], the lengthy HSPICE manual, or any number of textbooks on SPICE (such as [Kielkowski95, Foty96]).

5.2.1 Sources and Passive Components

Suppose we would like to find the response of the RC circuit in Figure 5.1(a) given an input rising from 0 to 1.8 V over 50 ps. Because the RC time constant of 100 fF • 2 kΩ = 200 ps is much greater than the input rise time, we intuitively expect the output would

look like an exponential asymptotically approaching the final value of 1.8 V with a 200 ps time constant. Figure 5.2 gives a SPICE deck for this simulation and Figure 5.1(b) shows the input and output responses.

Cards beginning with * are comments. The first card of a SPICE deck must be a comment, typically indicating the title of the simulation. It is good practice to treat SPICE input files like computer programs and follow similar procedures for commenting the decks. In particular, giving the author, date, and objective of the simulation at the beginning is helpful when the deck must be revisited in the future (e.g., when a chip is in silicon debug and old simulations are being reviewed to track down potential reasons for failure).

Control cards begin with a dot (.). The `.option post` card instructs HSPICE to write the results to a file for use with a waveform viewer. The last card of a SPICE deck must be `.end`.

Each card in the netlist begins with a letter indicating the type of circuit element. Note that SPICE is case-insensitive. Common elements are given in Table 5.1. In this case the circuit consists of a voltage source named `Vin`, a resistor named `R1`, and a capacitor named `C1`. The nodes in the circuit are named `in`, `out`, and `gnd`. `gnd` is a special node name defined to be the 0 V reference. The units consist of one or two letters. The first character indicates the order of magnitude, as given in Table 5.2. The second letter indicates a unit for human convenience (such as F for farad or s for second) and is ignored by SPICE. For example, the hundred femtofarad capacitor can be expressed as `100fF`, `100f`, or simply `100e-15`.

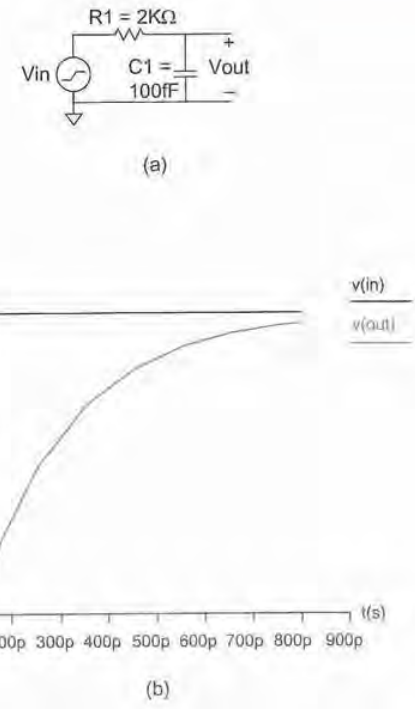


FIG 5.1 RC circuit response

The voltage source is defined as a piecewise linear (PWL) source. The waveform is specified with an arbitrary number of (time, voltage) pairs. Other common sources include DC sources and pulse sources. A DC voltage source named `Vdd` that sets node `vdd` to 2.5 V could be expressed as:

```
Vdd   vdd   gnd   2.5
```

Pulse sources are convenient for repetitive signals like clocks. The general form for a pulse source is illustrated in Figure 5.3. For example, a clock with a 1.8 V swing, 800 ps period, 100 ps rise and fall times, and 50% duty cycle (i.e., equal high and low times) would be expressed as

```
Vck   clk   gnd   PULSE 0 1.8 0ps 100ps 100ps 300ps 800ps
```

```
* rc.sp
* David_Harris@hmc.edu 2/2/03
* Find the response of RC circuit to rising input

*---------------------------------------------------------------------
* Parameters and models
*---------------------------------------------------------------------
.option post

*---------------------------------------------------------------------
* Simulation netlist
*---------------------------------------------------------------------
Vin    in     gnd    pwl    0ps 0 100ps 0 150ps 1.8 800ps 1.8
R1     in     out    2k
C1     out    gnd    100f

*---------------------------------------------------------------------
* Stimulus
*---------------------------------------------------------------------
.tran 20ps 800ps
.plot v(in) v(out)
.end
```

FIG 5.2 RC spice deck

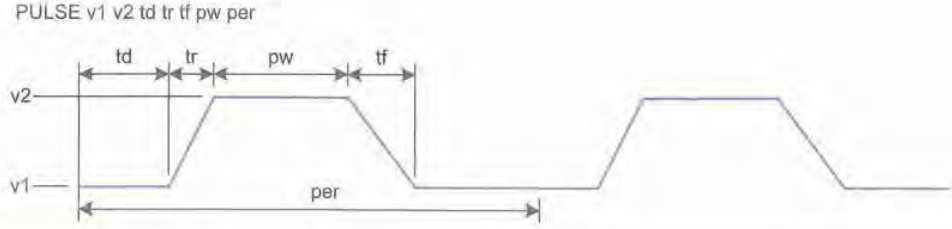


FIG 5.3 PULSE waveform

The stimulus specifies that a transient analysis (`.tran`) should be performed using a maximum step size of 20 ps for a duration of 800 ps. When plotting node voltages, the step size determines the spacing between points.

The `.plot` command generates a textual plot of the node variables specified (in this case the voltages at nodes `in` and `out`), as shown in Figure 5.4. Similarly, the `.print` statement prints the results in a multicolumn table. Both commands show the legacy of

Table 5.1 Common SPICE elements

Letter	Element
R	Resistor
C	Capacitor
L	Inductor
K	Mutual inductor
V	Independent voltage source
I	Independent current source
M	MOSFET
D	Diode
Q	Bipolar transistor
W	Lossy transmission line
X	Subcircuit
E	Voltage-controlled voltage source
G	Voltage-controlled current source
H	Current-controlled voltage source
F	Current-controlled current source

Table 5.2 SPICE units

Letter	Unit	Magnitude
a	atto	10^{-18}
f	femto	10^{-15}
p	pico	10^{-12}
n	nano	10^{-9}
u	micro	10^{-6}
m	mili	10^{-3}
k	kilo	10^{3}
x	mega	10^{6}
g	giga	10^{9}

FORTRAN and line printers. On modern computers with graphical user interfaces, the `.option post` command is usually preferred. It generates a file (in this case, `rc.tr0`) containing the results of the specified (transient) analysis. Then a separate graphical waveform viewer can be used to look at and manipulate the waveforms. AWAVES and COSMOSCOPE are waveform viewers compatible with HSPICE.

```
legend:
  a: v(in)
  b: v(out)

      time         v(in)
 (ab       )        0.          500.0000m         1.0000          1.5000          2.0000
                     +              +               +               +               +
   0.           0.     -2-------+-------+-------+-------+-------+-------+-------+-------+-
  20.0000p      0.      2       +       +       +       +       +       +       +       +
  40.0000p      0.      2       +       +       +       +       +       +       +       +
  60.0000p      0.      2       +       +       +       +       +       +       +       +
  80.0000p      0.      2       +       +       +       +       +       +       +       +
 100.0000p      0.      2       +       +       +       +       +       +       +       +
 120.0000p   720.000m  +b       +       +      a+       +       +       +       +       +
 140.0000p     1.440   +   b   +       +       +       +       +    a  +       +       +
 160.0000p     1.800   +       +b      +       +       +       +       +       +a      +
 180.0000p     1.800   +       +    b  +       +       +       +       +       +a      +
 200.0000p     1.800  -+-------+-------+b------+-------+-------+-------+-------+a------+-
 220.0000p     1.800   +       +       +    b  +       +       +       +       +a      +
 240.0000p     1.800   +       +       +      +b       +       +       +       +a      +
 260.0000p     1.800   +       +       +       +   b   +       +       +       +a      +
 280.0000p     1.800   +       +       +       +     b+       +       +       +a      +
 300.0000p     1.800   +       +       +       +      +b      +       +       +a      +
 320.0000p     1.800   +       +       +       +       +   b   +       +       +a      +
 340.0000p     1.800   +       +       +       +       +     b +       +       +a      +
 360.0000p     1.800   +       +       +       +       +       b       +       +a      +
 380.0000p     1.800   +       +       +       +       +       +b      +       +a      +
 400.0000p     1.800  -+-------+-------+-------+-------+-------+---b---+-------+a------+-
 420.0000p     1.800   +       +       +       +       +       +    b  +       +a      +
 440.0000p     1.800   +       +       +       +       +       +     b +       +a      +
 460.0000p     1.800   +       +       +       +       +       +      b+       +a      +
 480.0000p     1.800   +       +       +       +       +       +       b       +a      +
 500.0000p     1.800   +       +       +       +       +       +       +b      +a      +
 520.0000p     1.800   +       +       +       +       +       +       +b      +a      +
 540.0000p     1.800   +       +       +       +       +       +       + b     +a      +
 560.0000p     1.800   +       +       +       +       +       +       +  b    +a      +
 580.0000p     1.800   +       +       +       +       +       +       +  b    +a      +
 600.0000p     1.800  -+-------+-------+-------+-------+-------+-------+---b---+a------+-
 620.0000p     1.800   +       +       +       +       +       +       +    b  +a      +
 640.0000p     1.800   +       +       +       +       +       +       +     b +a      +
 660.0000p     1.800   +       +       +       +       +       +       +     b +a      +
 680.0000p     1.800   +       +       +       +       +       +       +     b +a      +
 700.0000p     1.800   +       +       +       +       +       +       +      b+a      +
 720.0000p     1.800   +       +       +       +       +       +       +      b+a      +
 740.0000p     1.800   +       +       +       +       +       +       +      b+a      +
 760.0000p     1.800   +       +       +       +       +       +       +      b+a      +
 780.0000p     1.800   +       +       +       +       +       +       +       ba      +
 800.0000p     1.800  -+-------+-------+-------+-------+-------+-------+-------ba------+-
                     +              +               +               +               +
```

FIG 5.4 Textual plot of RC circuit response

5.2.2 Transistor DC Analysis

One of the first steps in becoming familiar with a new CMOS process is to look at the I-V characteristics of the transistors. Figure 5.5(a) shows test circuits for a unit (4/2 λ) nMOS transistor in a 180 nm process at V_{DD} = 1.8 V. The I-V characteristics are plotted in Figure 5.5(b) using the SPICE deck in Figure 5.6.

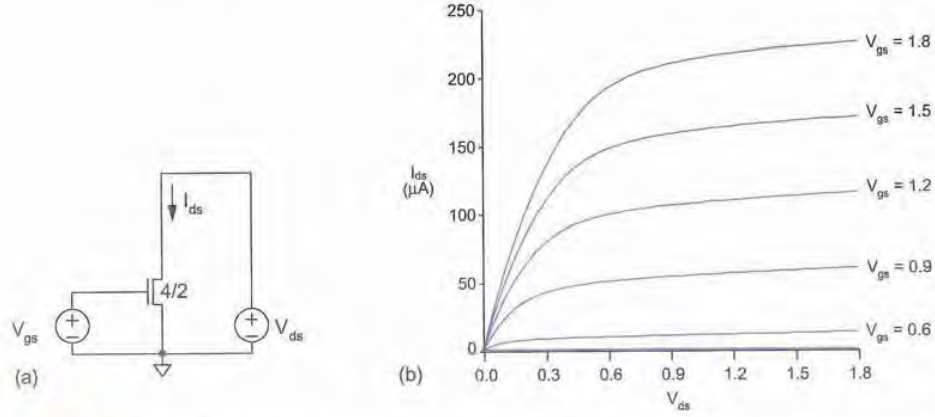


FIG 5.5 MOS I-V characteristics

```
* mosiv.sp

*---------------------------------------------------------------------
* Parameters and models
*---------------------------------------------------------------------
.include '../models/tsmc180/models.sp'
.temp 70
.option post

*---------------------------------------------------------------------
* Simulation netlist
*---------------------------------------------------------------------
*nmos
Vgs   g   gnd   0
Vds   d   gnd   0
M1    d   g     gnd   gnd   NMOS   W=0.36u  L=0.18u

*---------------------------------------------------------------------
* Stimulus
*---------------------------------------------------------------------
.dc Vds 0 1.8 0.05 SWEEP Vgs 0 1.8 0.3
.end
```

FIG 5.6 MOSIV SPICE deck

`.include` reads another SPICE file from disk. In this example, it loads device models that will be discussed further in Section 5.3. The circuit uses two independent voltage sources with default values of 0 V; these voltages will be varied by the `.dc` command. The nMOS transistor is defined with the MOSFET element `M` using the syntax

```
Mname   drain   gate   source   body   type   W=<width>   L=<length>
```

The `.dc` command varies the voltage source `Vds` DC voltage from 0 to 1.8 V in increments of 0.05 V. This is repeated multiple times as `Vgs` is swept from 0 to 1.8 V in 0.3 V increments to compute many I_{ds} vs. V_{ds} curves at different values of V_{gs}.

5.2.3 Inverter Transient Analysis

Figure 5.7 shows the step response of an unloaded unit inverter, annotated with propagation delay and 20%–80% rise and fall times. Observe that significant overshoot from bootstrapping occurs because there is no load. The SPICE deck for the simulation is shown in Figure 5.8.

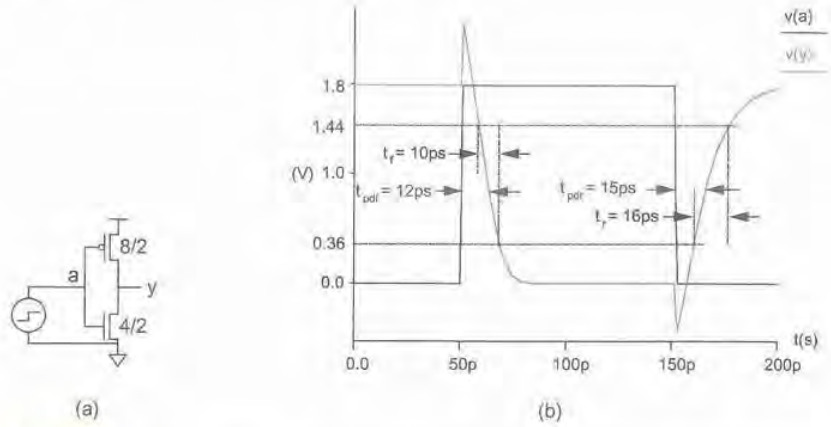


FIG 5.7 Unloaded inverter

This deck introduces the use of parameters and scaling. The `.param` statement defines a parameter named `SUPPLY` to have a value of 1.8. This is then used to set `Vdd` and the amplitude of the input pulse. If we wanted to evaluate the response at a different supply voltage, we would simply need to change the `.param` statement. The `.scale` sets a scale factor for all dimensions that would by default be measured in meters. In this case, it sets the scale to λ = 90 nm. Now the transistor widths and lengths in the inverter are specified in terms of lambda rather than in meters.

```
* inv.sp

*-----------------------------------------------------------------------
* Parameters and models
*-----------------------------------------------------------------------
.param SUPPLY=1.8
.option scale=90n
.include '../models/tsmc180/models.sp'
.temp 70
.option post

*-----------------------------------------------------------------------
* Simulation netlist
*-----------------------------------------------------------------------
Vdd   vdd   gnd   'SUPPLY'
Vin   a     gnd   PULSE  0 'SUPPLY' 50ps 0ps 0ps 100ps 200ps
M1    y     a     gnd    gnd   NMOS   W=4   L=2
+ AS=20 PS=18 AD=20 PD=18
M2    y     a     vdd    vdd   PMOS   W=8   L=2
+ AS=40 PS=26 AD=40 PD=26

*-----------------------------------------------------------------------
* Stimulus
*-----------------------------------------------------------------------
.tran 1ps 200ps
.end
```

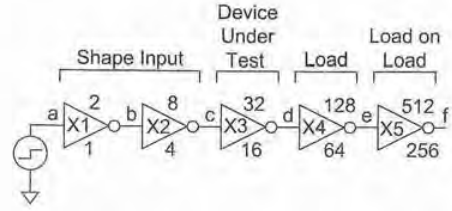
FIG 5.8 INV SPICE deck

Recall that parasitic delay is strongly dependent on diffusion capacitance, which in turn depends on the area and perimeter of the source and drain. As each diffusion region in an inverter must be contacted, the geometry resembles that of Figure 2.9(a). The diffusion width equals the transistor width and the diffusion length is 5 λ. Thus, the area of the source and drain are AS = AD = $5W\lambda^2$ and the perimeters are PS = PD = $(2W + 10)\ \lambda$. Note that the + sign in the first column of a card indicates that it is a continuation of the previous card. These dimensions are also affected by the scale factor.

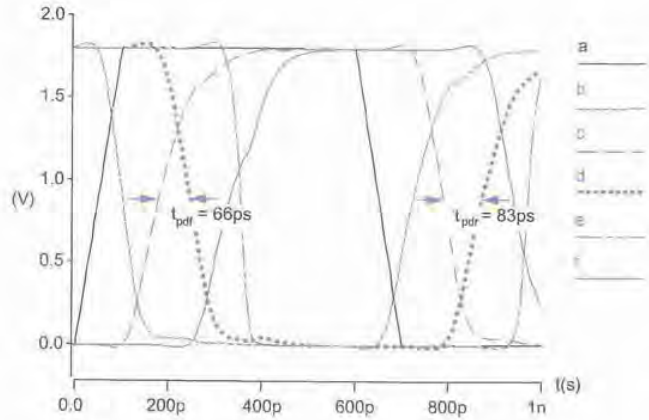
5.2.4 Subcircuits and Measurement

One of the simplest measures of a process's inherent speed is the fanout-of-4 inverter delay. Figure 5.9(a) shows a circuit to measure this delay. The nMOS and pMOS transistor sizes (in multiples of a unit 4/2 λ transistor) are listed below and above each gate, respectively. *X3* is the inverter under test and *X4* is its load, which is four times larger than *X3*. To first order, these two inverters would be sufficient. However, the delay of *X3* also depends on the input slope, as discussed in Section 4.2.5.1. One way to obtain a realistic input slope is to drive node c with a pair of FO4 inverters *X1* and *X2*. Also, as discussed in Section 4.2.5.4, the input capacitance of *X4* depends not just on its C_{gs} but also

on C_{gd}. C_{gd} is Miller-multiplied as node e switches and would be effectively doubled if e switched instantaneously. When e is loaded with *X5*, it switches at a slower, more realistic rate, slightly reducing the effective capacitance presented at node d by *X4*. The waveforms in Figure 5.9(b) are annotated with the rising and falling delays.



(a)



(b)

FIG 5.9 Fanout-of-4 inverters

SPICE decks are easier to read and maintain when common circuit elements are captured as subcircuits. For example, the deck in Figure 5.10 computes the FO4 inverter delay using an inverter subcircuit.

The `.global` statement defines vdd and gnd as global nodes that can be referenced from within subcircuits. The inverter is declared as a subcircuit with two terminals: a and y. It also accepts two parameters specifying the width of the nMOS and pMOS transis-

```
* fo4.sp
*-----------------------------------------------------------------------
* Parameters and models
*-----------------------------------------------------------------------
.param SUPPLY=1.8
.param H=4
.option scale=90n
.include '../models/tsmc180/models.sp'
.temp 70
.option post
*-----------------------------------------------------------------------
* Subcircuits
*-----------------------------------------------------------------------
.global vdd gnd
.subckt inv a y N=4 P=8
M1    y     a     gnd     gnd     NMOS    W='N'   L=2
+ AS='N*5' PS='2*N+10' AD='N*5' PD='2*N+10'
M2    y     a     vdd     vdd     PMOS    W='P'   L=2
+ AS='P*5' PS='2*P+10' AD='P*5' PD='2*P+10'
.ends
*-----------------------------------------------------------------------
* Simulation netlist
*-----------------------------------------------------------------------
Vdd   vdd   gnd   'SUPPLY'
Vin   a     gnd   PULSE   0 'SUPPLY' 0ps 100ps 100ps 500ps 1000ps
X1    a     b     inv                 * shape input waveform
X2    b     c     inv     M='H'       * reshape input waveform
X3    c     d     inv     M='H**2'    * device under test
X4    d     e     inv     M='H**3'    * load
X5    e     f     inv     M='H**4'    * load on load
*-----------------------------------------------------------------------
* Stimulus
*-----------------------------------------------------------------------
.tran 1ps 1000ps
.measure tpdr                                   * rising prop delay
+     TRIG v(c)    VAL='SUPPLY/2' FALL=1
+     TARG v(d)    VAL='SUPPLY/2' RISE=1
.measure tpdf                                   * falling prop delay
+     TRIG v(c)    VAL='SUPPLY/2' RISE=1
+     TARG v(d)    VAL='SUPPLY/2' FALL=1
.measure tpd param='(tpdr+tpdf)/2'              * average prop delay
.measure trise                                  * rise time
+     TRIG v(d)    VAL='0.2*SUPPLY' RISE=1
+     TARG v(d)    VAL='0.8*SUPPLY' RISE=1
.measure tfall                                  * fall time
+     TRIG v(d)    VAL='0.8*SUPPLY' FALL=1
+     TARG v(d)    VAL='0.2*SUPPLY' FALL=1
.end
```

FIG 5.10 FO4 SPICE deck

tors; these parameters have default values of 4 and 8, respectively. The source and drain area and perimeter are functions of the transistor widths. HSPICE evaluates functions given inside single quotation marks. The functions can include parameters, constants, parentheses, +, -, *, /, and ** (raised to a power).

The simulation netlist contains the power supply, input waveform, and four inverters. Each inverter is subcircuit (X) element. As N and P are not specified, each uses the default size. The M parameter multiplies all the currents in the subcircuit by the factor given, equivalent to M elements wired in parallel. In this case, the fanouts are expressed in terms of a parameter H. Thus, *X2* has the capacitance and output current of 4 unit inverters, while *X3* is equivalent to 16. Another way to model the inverters would have been to use the N and P parameters:

```
X1   a   b   inv   N=4      P=8      * shape input waveform
X2   b   c   inv   N=16     P=32     * reshape input waveform
X3   c   d   inv   N=64     P=128    * device under test
X4   d   e   inv   N=256    P=512    * load
X5   e   f   inv   N=1024   P=2048   * load on load
```

However, a transistor of four times unit width does not have exactly the same input capacitance or output current as four unit inverters tied in parallel, so the M parameter is usually preferred.

In this example, the subcircuit declaration and simulation netlist are part of the SPICE deck. When working with a standard cell library, it is common to keep subcircuit declarations in their own files and reference them with a `.include` statement instead. When the simulation netlist is extracted from a schematic or layout CAD system, it is common to put the netlist in a separate file and `.include` it as well.

The `.measure` statement measures simulation results and prints them in the listing file. The deck measures the rising propagation delay t_{pdr} as the difference between the time that the input c first falls through $V_{DD}/2$ and the time that the output d first rises through $V_{DD}/2$. TRIG and TARG indicate the trigger and target events between which delay is measured. The `.measure` statement can also be used to compute functions of other measurements. For example, the average FO4 inverter propagation delay t_{pd} is the mean of t_{pdr} and t_{pdf}, i.e., t_{pd} = 75 ps. The 20%–80% rise time is t_r = 94 ps and the fall time is t_f = 67 ps.

5.2.5 Optimization

In many examples, we have assumed that a *P/N* ratio of 2:1 gives approximately equal rise and fall delays. The FO4 inverter simulation showed that a ratio of 2:1 gives rising delays that are slower than the falling delays because the pMOS mobility is less than half that of the nMOS. You could repeatedly run simulations with different default values of P to find the ratio for equal delay. HSPICE has built-in optimization capabilities that will automatically tweak parameters to achieve some goal and report what parameter value gave the best results. Figure 5.11 shows a modified version of the FO4 inverter simulation using the optimizer.

```
* fo4opt.sp
*-----------------------------------------------------------------------
* Parameters and models
*-----------------------------------------------------------------------
.param SUPPLY=1.8
.option scale=90n
.include '../models/tsmc180/models.sp'
.temp 70
.option post
*-----------------------------------------------------------------------
* Subcircuits
*-----------------------------------------------------------------------
.global vdd gnd
.subckt inv a y N=4 P=8
M1   y   a   gnd   gnd   NMOS   W='N'   L=2
+ AS='N*5' PS='2*N+10' AD='N*5' PD='2*N+10'
M2   y   a   vdd   vdd   PMOS   W='P'   L=2
+ AS='P*5' PS='2*P+10' AD='P*5' PD='2*P+10'
.ends
*-----------------------------------------------------------------------
* Simulation netlist
*-----------------------------------------------------------------------
Vdd   vdd   gnd   'SUPPLY'
Vin   a     gnd   PULSE    0 'SUPPLY' 0ps 100ps 100ps 500ps 1000ps
X1    a     b     inv      P='P1'            * shape input waveform
X2    b     c     inv      P='P1'    M=4     * reshape input waveform
X3    c     d     inv      P='P1'    M=16    * device under test
X4    d     e     inv      P='P1'    M=64    * load
X5    e     f     inv      P='P1'    M=256   * load on load
*-----------------------------------------------------------------------
* Optimization setup
*-----------------------------------------------------------------------
.param P1=optrange(8,4,16)          * search from 4 to 16, guess 8
.model optmod opt itropt=30         * maximum of 30 iterations
.measure bestratio param='P1/4'     * compute best P/N ratio
*-----------------------------------------------------------------------
* Stimulus
*-----------------------------------------------------------------------
.tran 1ps 1000ps SWEEP OPTIMIZE=optrange RESULTS=diff MODEL=optmod
.measure tpdr                                 * rising propagation delay
+     TRIG v(c)   VAL='SUPPLY/2' FALL=1
+     TARG v(d)   VAL='SUPPLY/2' RISE=1
.measure tpdf                                 * falling propagation delay
+     TRIG v(c)   VAL='SUPPLY/2' RISE=1
+     TARG v(d)   VAL='SUPPLY/2' FALL=1
.measure tpd param='(tpdr+tpdf)/2' goal=0  * average prop delay
.measure diff param='tpdr-tpdf' goal = 0   * diff between delays
.end
```

FIG 5.11 FO4OPT SPICE deck

The subcircuits *X1–X4* override their default pMOS widths to use a width of `P1` instead. In the optimization setup, the difference of t_{pdr} and t_{pdf} is measured. The goal of the optimization will be to drive this difference to 0. To do this, `P1` may be varied from 4 to 16, with an initial guess of 8. The optimizer may use up to 30 iterations to find the best value of `P1`. Because the nMOS width is fixed at 4, the best *P/N* ratio is computed as `P1`/4. The transient analysis includes a `SWEEP` statement containing the parameter to vary, the desired result, and the number of iterations.

HSPICE determines that the *P/N* ratio for equal rise and fall delay is 3.6:1, giving a rising and falling delay of 84 ps. This is slower than the 2:1 ratio provides and requires large pMOS transistors that consume area and power, so such a high ratio is seldom used.

A similar scenario is to find the *P/N* ratio that gives lowest average delay. By changing the `.tran` card to use `RESULTS=tpd`, we find a best ratio of 1.4:1 with rising, falling, and average propagation delays of 87, 59, and 73 ps, respectively. Whenever you do an optimization, it is important to consider not only the optimum but also the sensitivity to deviations from this point. Further simulation finds that *P/N* ratios of anywhere from 1.2:1 to 1.7:1 all give an average propagation delay of 73 ps, there is no need to slavishly stick to the 1.4:1 "optimum." The best *P/N* ratio in practice is a compromise between using smaller pMOS devices to save area and power and using larger devices to achieve more nearly equal rise/fall times and avoid the hot electron reliability problems induced by very slow rising edges in circuits with weak pMOS transistors. *P/N* ratios are discussed further in Section 6.2.1.6.

5.2.6 Other HSPICE Commands

The full HSPICE manual fills over 2000 pages and includes many more capabilities than can be described here. A few of the most useful additional commands are covered in this section. Section 5.3 describes transistor models and library calls, while Section 5.6 discusses modeling interconnect with lossy transmission lines.

`.option accurate`

Tighten integration tolerances to obtain more accurate results. Useful for oscillators and high-gain analog circuits or when results seem fishy.

`.option autostop`

Conclude simulation when all `.measure` results are obtained rather than continuing for the full duration of the `.tran` card. This can substantially reduce simulation time.

```
.temp 0 70 125
```

Repeat the simulation three times at temperatures of 0°, 70°, and 125° C. Device models may contain information about how changing temperature changes device performance.

`.op`

Print the voltages, currents, and transistor bias conditions at the DC operating point.

5.3 Device Models

Most of the examples in Section 5.2 included a file containing transistor models. SPICE provides a wide variety of MOS transistor models with various tradeoffs between complexity and accuracy. Level 1 and Level 3 models were historically important, but they are no longer adequate to accurately model very small modern transistors. BSIM models are more accurate and are presently the most widely used. Some companies use their own proprietary models. This section briefly describes the main features of each of these models. It also describes how to model diffusion capacitance and how to run simulations in various process corners. The model descriptions are intended only as an overview of the capabilities and limitations of the models; refer to a SPICE manual for a much more detailed description if one is necessary.

5.3.1 Level 1 Models

The SPICE Level 1, or Shichman-Hodges Model [Shichman68] is closely related to the Shockley model described in EQ (2.10), enhanced with channel length modulation and the body effect. The basic current model is:

$$I_{ds} = \begin{cases} 0 & V_{gs} < V_t \quad \text{cutoff} \\ \text{KP}\dfrac{W_{\text{eff}}}{L_{\text{eff}}}(1+\text{LAMBDA}\cdot V_{ds})\left(V_{gs}-V_t-\dfrac{V_{ds}}{2}\right)V_{ds} & V_{ds} < V_{gs}-V_t \quad \text{linear} \\ \dfrac{\text{KP}}{2}\dfrac{W_{\text{eff}}}{L_{\text{eff}}}(1+\text{LAMBDA}\cdot V_{ds})(V_{gs}-V_t)^2 & V_{ds} > V_{gs}-V_t \quad \text{saturation} \end{cases} \tag{5.1}$$

The parameters from the SPICE model are given in ALL CAPS. Notice that β is written instead as KP($W_{\text{eff}}/L_{\text{eff}}$), where KP is a model parameter playing the role of k' from EQ (2.7). W_{eff} and L_{eff} are the effective width and length, as described in Section 2.4.8. The LAMBDA term models channel length modulation (see Section 2.4.2).

The threshold voltage is modulated by the source-to-body voltage V_{sb} through the body effect (see Section 2.4.3). For nonnegative V_{sb}, the threshold voltage is

$$V_t = \text{VTO} + \text{GAMMA}\left(\sqrt{\text{PHI}+V_{sb}} - \sqrt{\text{PHI}}\right) \tag{5.2}$$

Notice that this is identical to EQ (2.30), where VTO is the "zero-bias" threshold voltage V_{t0}, GAMMA is the body effect coefficient γ, and PHI is the surface potential ϕ_s.

The gate capacitance is calculated from the oxide thickness TOX. The default gate capacitance model in HSPICE is adequate for finding the transient response of digital circuits. More elaborate models exist that capture nonreciprocal effects that are important for analog design.

Level 1 models are useful for teaching because they are easy to correlate with hand analysis, but are too simplistic for modern design. Figure 5.12 gives an example of a Level 1 model card illustrating the card syntax. The card also includes terms to compute the diffusion capacitance, as described in Section 5.3.4.

```
.model NMOS NMOS (LEVEL=1 TOX=40e-10 KP=155E-6 LAMBDA=0.2
+                 VTO=0.4 PHI=0.93 GAMMA=0.6
+                 CJ=9.8E-5 PB=0.72 MJ=0.36
+                 CJSW=2.2E-10 PHP=7.5 MJSW=0.1)
```

FIG 5.12 Sample Level 1 MODEL card

5.3.2 Level 2 and 3 Models

The SPICE Level 2 and 3 models add effects of velocity saturation, mobility degradation, subthreshold conduction, and drain-induced barrier lowering. The Level 2 model is based on the Grove-Frohman equations [Frohman69], while the Level 3 model is based on empirical equations that provide similar accuracy and faster simulation times and better convergence. However, these models still do not provide good fits to the measured I-V characteristics of modern transistors.

5.3.3 BSIM Models

The Berkeley Short-Channel IGFET[1] Model (BSIM) is a very elaborate model that is now widely used in circuit simulation. The models are derived from the underlying device physics but use an enormous number of parameters to fit the behavior of modern transistors. BSIM versions 1, 2, 3v3, and 4 are implemented as SPICE levels 13, 39, 49, and 54, respectively.

BSIM version 3v3 requires an entire book [Cheng99] to describe the model. It includes over 100 parameters and the device equations span 27 pages. It is quite good for digital circuit simulation except that it does not model gate leakage. Features of the model include:

- Continuous and differentiable I-V characteristics across subthreshold, linear, and saturation regions for good convergence
- Sensitivity of parameters such as V_t to transistor length and width
- Detailed threshold voltage model including body effect and drain-induced barrier lowering
- Velocity saturation, mobility degradation, and other short-channel effects
- Multiple gate capacitance models
- Diffusion capacitance and resistance models

[1] IGFET in turn stands for Insulated-Gate Field Effect Transistor, a synonym for MOSFET.

BSIM version 4 was fairly new at the time this book was being written and adds support for gate leakage and other effects of very thin gates. As these effects are rapidly becoming more important and better characterized, the designer should check with the process engineers before blindly trusting gate leakage models.

Some device parameters such as threshold voltage change significantly with device dimensions. BSIM models can be *binned* with different models covering different ranges of length and width specified by LMIN, LMAX, WMIN, and WMAX parameters. For example, one model might cover transistors with channel lengths from 0.18–0.25 μm, another from 0.25–0.5 μm, and a third from 0.5–5 μm. SPICE will complain if a transistor does not fit in one of the bins.

As the BSIM models are so complicated, it is impractical to derive closed-form equations for propagation delay, switching threshold, noise margins, etc., from the underlying equations. However, it is not difficult to find these properties through circuit simulation. Section 5.4 will show simple simulations to plot the device characteristics over the regions of operation that are interesting to most digital designers and to extract effective capacitance and resistance averaged across the switching transition. The simple RC model continues to give the designer important insight about the characteristics of logic gates.

5.3.4 Diffusion Capacitance Models

The p–n junction between the source or drain diffusion and the body forms a reverse-biased diode. We have seen that the diffusion capacitance determines the parasitic delay of a gate and depends on the area and perimeter of the diffusion. HSPICE provides a number of methods to specify this geometry, controlled by the ACM (Area Calculation Method) parameter, which is part of the transistor model card. The model card must also have values for junction and sidewall diffusion capacitance, as described in Section 2.2.2.3. The diffusion capacitance model is common across most device models including Levels 1–3 and BSIM.

By default, HSPICE models use ACM=0. In this method, the designer must specify the area and perimeter of the source and drain of each transistor. For example, the dimensions of each diffusion region from Figure 2.9 is listed in Table 5.3 (in units of λ^2 for area or λ for perimeter). A SPICE description of the shared contacted diffusion case is shown in Figure 5.13, assuming `.option scale` is set to the value of λ.

Table 5.3 Diffusion area and perimeter

	AS1 / AD2	PS1 / PD2	AD1 / AS2	PD1 / PS2
(a) Isolated contacted diffusion	$W \cdot 5$	$2 \cdot W + 10$	$W \cdot 5$	$2 \cdot W + 10$
(b) Shared contacted diffusion	$W \cdot 5$	$2 \cdot W + 10$	$W \cdot 3$	$W + 6$
(c) Merged uncontacted diffusion	$W \cdot 5$	$2 \cdot W + 10$	$W \cdot 1.5$	$W + 3$

```
* (b): Shared contacted diffusion
M1    mid    b    bot    gnd    NMOS    W='w'    L=2
+ AS='w*5' PS='2*w+10' AD='w*3' PD='w+6'
M2    top    a    mid    gnd    NMOS    W='w'    L=2
+ AS='w*3' PS='w+6' AD='w*5' PD='2*w+10'
```

FIG 5.13 SPICE model of transistors with shared contacted diffusion

The SPICE models also should contain parameters CJ, CJSW, PB, PHP, MJ, and MJSW. Assuming the diffusion is reverse-biased and the area and perimeter are specified, the diffusion capacitance between source and body is computed as described in Section 2.3.3.

$$C_{sb} = \text{AS} \cdot \text{CJ} \cdot \left(1+\frac{V_{sb}}{\text{PB}}\right)^{-MJ} + \text{PS} \cdot \text{CJSW} \cdot \left(1+\frac{V_{sb}}{\text{PHP}}\right)^{-MJSW} \tag{5.3}$$

The drain equations are analogous, with S replaced by D in the model parameters.

The BSIM3 models offer a similar area calculation model (ACM = 10) that takes into account the different sidewall capacitance on the edge adjacent to the gate. Note that the PHP parameter is renamed to PBSW to be more consistent.

$$C_{sb} = \text{AS} \cdot \text{CJ} \cdot \left(1+\frac{V_{sb}}{\text{PB}}\right)^{-MJ} + (\text{PS}-W) \cdot \text{CJSW} \cdot \left(1+\frac{V_{sb}}{\text{PBSW}}\right)^{-MJSW} + W \cdot \text{CJSWG} \cdot \left(1+\frac{V_{sb}}{\text{PBSWG}}\right)^{-MJSWG} \tag{5.4}$$

If the area and perimeter are not specified, they default to 0 in ACM = 0 or 10, grossly underestimating the parasitic delay of the gate. HSPICE also supports ACM = 1, 2, 3, and 12 that provide nonzero default values when the area and perimeter are not specified. Check your models and read the HSPICE documentation carefully.

The diffusion area and perimeter is also used to compute the junction leakage current. However, this current is generally negligible compared to subthreshold leakage in modern devices.

5.3.5 Design Corners

Engineers often simulate circuits in multiple design corners to verify that operation across variations in device characteristics and environment. HSPICE includes the .lib card that makes changing libraries easy. For example, the deck in Figure 5.14 runs three simulations on the step response of an unloaded inverter in the TT, FF, and SS corners.

```
* corner.sp
* Step response of unloaded inverter across process corners

*-------------------------------------------------------------------------
* Parameters and models
*-------------------------------------------------------------------------
.option scale=90n
.param SUP=1.8 * Must set before calling .lib
.lib '../models/tsmc180/opconditions.lib' TT
.option post

*-------------------------------------------------------------------------
* Simulation netlist
*-------------------------------------------------------------------------
Vdd   vdd   gnd   'SUPPLY'
Vin   a     gnd   PULSE    0 'SUPPLY' 200ps 0ps 0ps 500ps 1000ps
M1    y     a     gnd      gnd     NMOS   W=4   L=2
+ AS=20 PS=18 AD=20 PD=18
M2    y     a     vdd      vdd     PMOS   W=8   L=2
+ AS=40 PS=26 AD=40 PD=26

*-------------------------------------------------------------------------
* Stimulus
*-------------------------------------------------------------------------
.tran 1ps 1000ps
.alter
.lib '../models/tsmc180/opconditions.lib' FF
.alter
.lib '../models/tsmc180/opconditions.lib' SS
.end
```

FIG 5.14 CORNER SPICE deck

The deck first sets SUP to the nominal supply voltage of 1.8 V. It then invokes the `.lib` card that reads in the library specifying the TT conditions. In the stimulus, the `.alter` statement is used to repeat the simulation with changes. In this case, the design corner is changed. Altogether, three simulations are performed and three sets of waveforms are generated for the three design corners.

The library file is given in Figure 5.15. Depending on what library was specified, the temperature is set (in degrees Celsius, with `.temp`) and the V_{DD} value SUPPLY is calculated from the nominal SUP. The library loads the appropriate nMOS and pMOS transistor models. A fast process file might have lower nominal threshold voltages V_{t0}, greater lateral diffusion L_D, and lower diffusion capacitance values.

```
* opconditions.lib
* For TSMC 180 nm process

* TT: Typical nMOS, pMOS, voltage, temperature
.lib TT
.temp 70
.param SUPPLY='SUP'
.include 'modelsTT.sp'
.endl TT

* SS: Slow nMOS, pMOS, low voltage, high temperature
.lib SS
.temp 125
.param SUPPLY='0.9 * SUP'
.include 'modelsSS.sp'
.endl SS

* FF: Fast nMOS, pMOS, high voltage, low temperature
.lib FF
.temp 0
.param SUPPLY='1.1 * SUP'
.include 'modelsFF.sp'
.endl FF

* FS: Fast nMOS, Slow pMOS, typical voltage and temperature
.lib FS
.temp 70
.param SUPPLY='SUP'
.include 'modelsFS.sp'
.endl FS

* SF: Slow nMOS, Fast pMOS, typical voltage and temperature
.lib SF
.temp 70
.param SUPPLY='SUP'
.include 'modelsSF.sp'
.endl SF
```

FIG 5.15 OPCONDITIONS library

5.4 Device Characterization

Modern SPICE models have so many parameters that the designer cannot easily read key performance characteristics from the model files. A more convenient approach is to run a set of simulations to extract the effective resistance and capacitance, the fanout-of-4 inverter delay, the I-V characteristics, and other interesting data. This section describes these simulations and compares the results across a variety of CMOS processes.

5.4.1 I-V Characteristics

When familiarizing yourself with a new process, a starting point is to plot the current-voltage (I-V) characteristics. Although digital designers seldom make calculations directly from these plots, it is helpful to know the ON current of nMOS and pMOS transistors, how severely velocity-saturated the process is, how the current rolls off below threshold, how the devices are affected by DIBL and body effect, and so forth. These plots are made with DC sweeps, as discussed in Section 5.2.2. Each transistor is 1 μm wide in a representative 180 nm process at 70° C with V_{DD} = 1.8 V. The left column shows nMOS behavior and the right column shows pMOS behavior.

Figure 5.16(a) plots I_{ds} vs. V_{ds} at various values of V_{gs}, as was done in Figure 5.5. The saturation current would ideally increase quadratically with $V_{gs} - V_t$, but in this plot shows closer to a linear dependence, indicating that the nMOS transistor is severely velocity-saturated (α closer to 1 than 2 in the α-power model). The increase in saturation current with V_{ds} is caused by channel length modulation. The saturation current for a pMOS transistor is lower than for the nMOS (note the different vertical scales), but the device is not as velocity-saturated. Figure 5.16(b) makes a similar plot for a device with a drawn channel length of 360 nm rather than 180 nm. The current is slightly flatter in saturation, indicating that channel length modulation has less impact at longer channel lengths.

Figure 5.16(c) plots I_{ds} vs. V_{gs} of an nMOS transistor on a semilogarithmic scale for V_{ds} = 0.1 V and 1.8 V. The straight line at low V_{gs} indicates that the current rolls off exponentially below threshold. The difference in subthreshold leakage at the varying drain voltage reflects the effects of drain-induced barrier lowering (DIBL) effectively reducing V_t at high V_{ds}. The saturation current I_{dsat} is measured at $V_{gs} = V_{ds} = V_{DD}$, while the OFF current I_{off} is measured at $V_{gs} = 0$ and $V_{ds} = V_{DD}$. The subthreshold slope is about 90 mV/decade and DIBL reduces the effective threshold voltage by about 40 mV over the range of V_{ds}.

Figure 5.16(d) makes a similar plot on a linear scale for $V_{bs} = -0.2$, 0, and 0.2 V. The curves shift horizontally, indicating that the body effect increases the threshold voltage as V_{bs} becomes more negative. V_{ds} is held constant at 0.1 V.

5.4.2 Threshold Voltage

In the Shockley model, the threshold voltage V_t is defined as the value of V_{gs} below which I_{ds} becomes 0. In the real transistor characteristics shown in Figure 5.16(c), subthreshold current continues to flow for $V_{gs} < V_t$, so measuring or even defining the threshold voltage becomes problematic. Moreover, the threshold voltage varies with L, W, V_{ds}, and V_{bs}. At least eleven different methods have been used in the literature to determine the threshold voltage from measured I_{ds}–V_{gs} data [Ortiz-Conde02]. This section will explore two common methods (constant current and linear extrapolation) and a hybrid that combines the advantages of each.

The *constant current* method defines threshold as the gate voltage at a given drain current I_{crit}. This method is easy to use, but depends on an arbitrary choice of critical drain current. A typical choice of I_{crit} is 0.1 μA • (W/L). Figure 5.17 shows how the extracted threshold voltage varies with the choice of I_{crit} = 0.1 or 1 μA at V_{ds} = 100 mV.

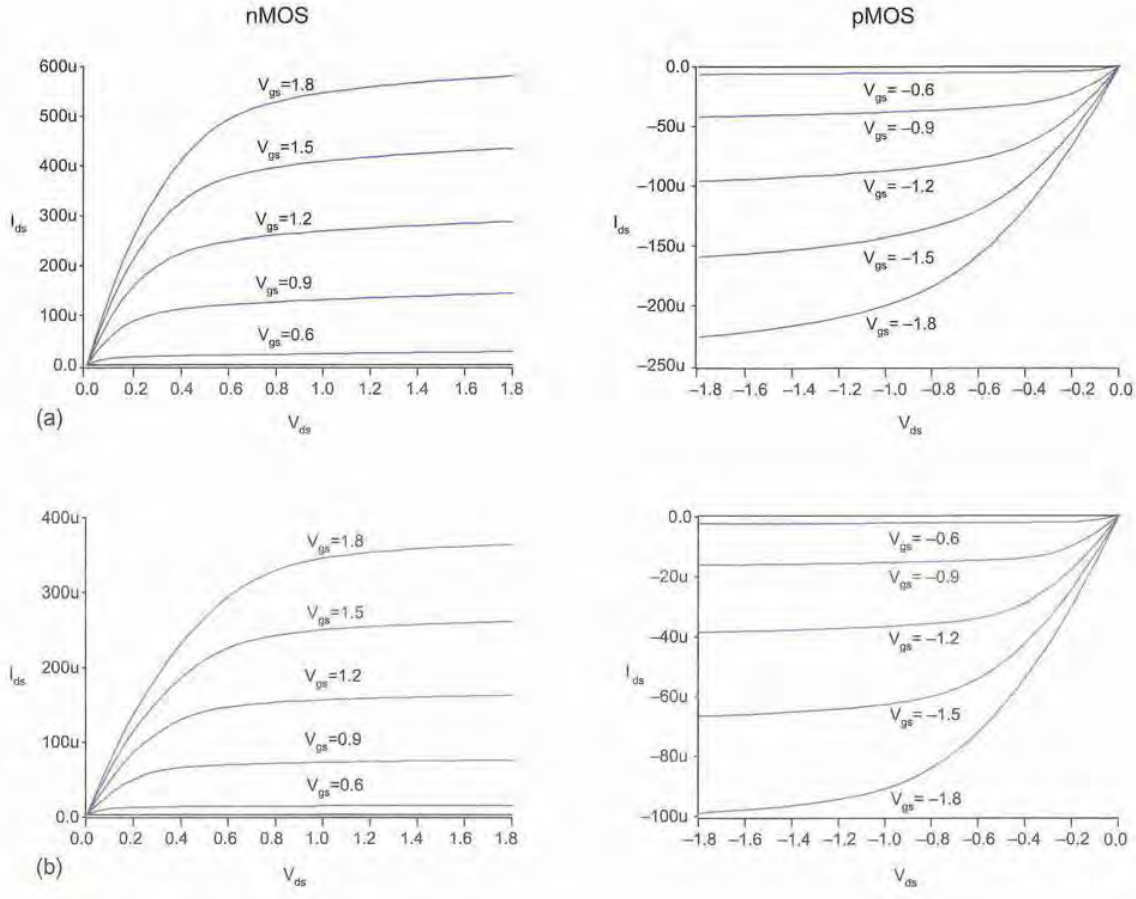


FIG 5.16 MOS I-V characteristics

The *linear extrapolation* (or *maximum-g_m*) method extrapolates the gate voltage from the point of maximum slope on the I_{ds}–V_{gs} characteristics. It is unambiguous but valid only for the linear region of operation (low V_{ds}) because of the series resistance of the source/drain diffusion and because drain-induced barrier lowering effectively reduces the threshold at high V_{ds}. Figure 5.18 shows how the threshold is extracted from measured data using the linear extrapolation method at V_{ds} = 100 mV. Observe that this method can give a significantly different threshold voltage and current at threshold, so it is important to check how the threshold voltage was measured when interpreting threshold voltage specifications. I_{crit} is defined to be the value of I_{ds} at $V_{gs} = V_t$.

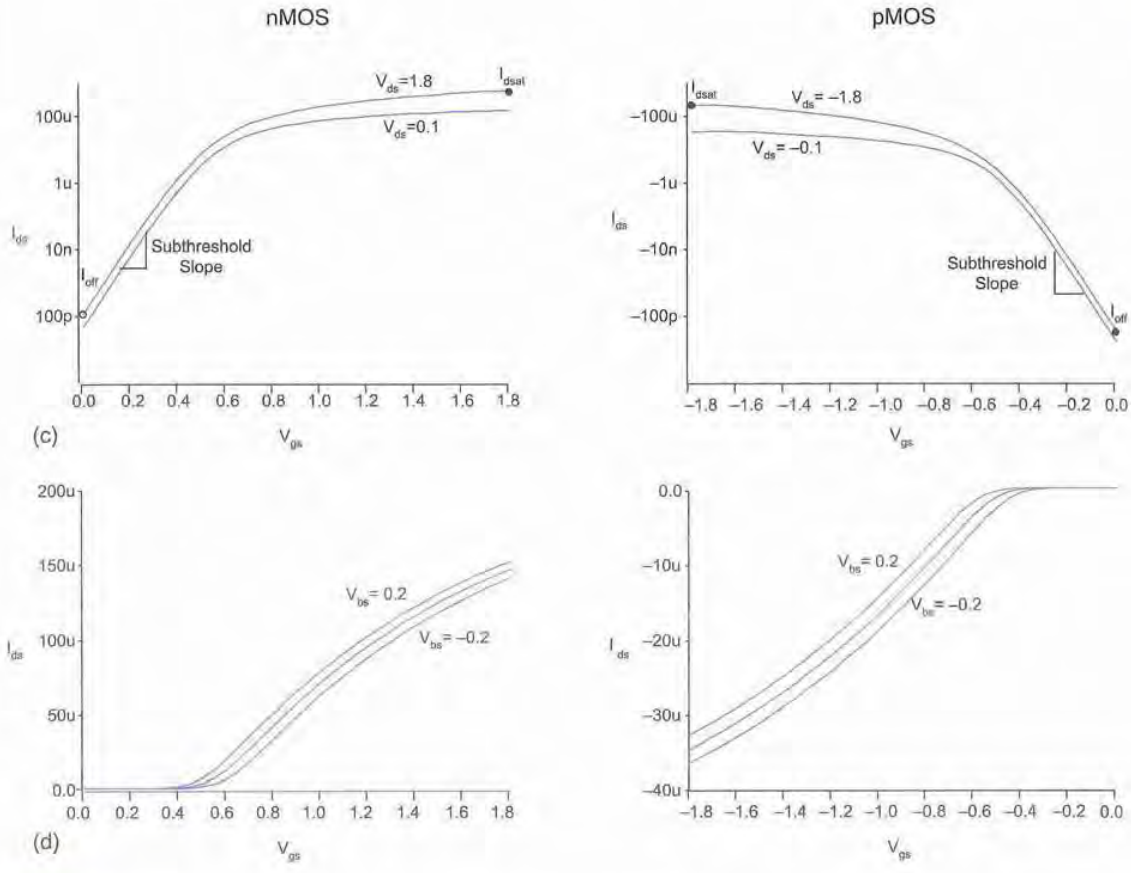


FIG 5.16 MOS I-V characteristics (continued)

[Zhou99] describes a hybrid method of extracting threshold voltage that is valid for all values of V_{ds} and does not depend on an arbitrary choice of critical current. V_t and I_{crit} are found at low V_{ds} (e.g., 100 mV) for a given value of L and W using the linear extrapolation method. For other values of V_{ds}, V_t is defined to be the gate voltage when $I_{ds} = I_{crit}$.

Figure 5.19(a) plots the threshold voltage V_t vs. length for a 16 λ-wide device over a variety of design corners and temperatures. The threshold is extracted using the linear extrapolation method and clearly is not constant. It decreases with temperature and is lower in the FF corner than in the SS corner. In an ideal long-channel transistor, the threshold is independent of width and length. In a real device, the geometry sensitivity

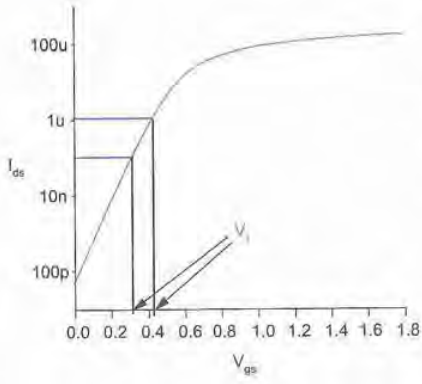


FIG 5.17 Constant current threshold voltage extraction method

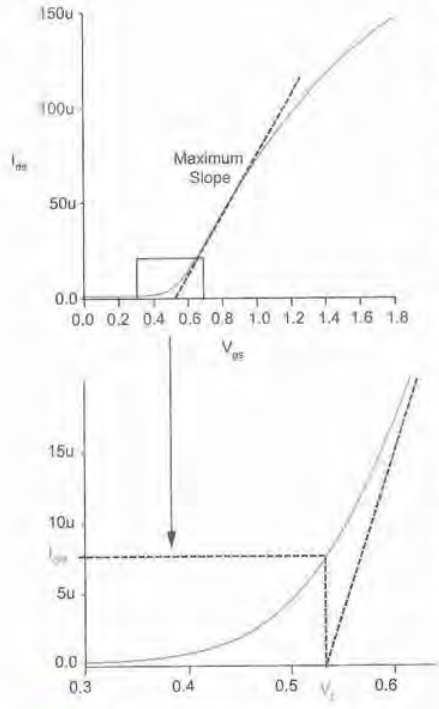


FIG 5.18 Linear extrapolation threshold voltage extraction method

depends on the particular doping profile of the process. Figure 5.19(b) plots V_t against V_{ds} for 16/2 λ transistors using Zhou's method. The threshold voltage decreases with V_{ds} because of DIBL.

The moral of this section is that V_t depends on length, width, temperature, processing, and how you define it. The current does not abruptly drop to zero at threshold and can be significant for leakage-sensitive circuits. When two devices need accurately matched thresholds (e.g., in a sense amplifier), the devices should be adjacent and identical in geometry.

5.4.3 Gate Capacitance

When using RC models to estimate gate delay, we need to know the effective gate capacitance for delay purposes. In Section 2.3.2, we saw that the gate capacitance is voltage-dependent. The gate-to-drain component may be effectively doubled when a gate switches because the gate and drain switch in opposite directions. Nevertheless, we can obtain an effective capacitance averaged across the switching time. We use fanout-of-4 inverters to represent gates with "typical" switching times because we know from logical effort that circuits perform well when the stage effort is approximately 4.

Figure 5.20 shows a circuit for determining the effective gate capacitance of inverter *X4*. The approach is to adjust the capacitance C_{delay} until the average delay from c to g equals the delay from c to d. Because *X6* and *X3* have the same input slope and are the same size, when they have the same delay, C_{delay} must equal the effective gate capacitance of *X4*. *X1* and *X2* are used to produce a reasonable input slope on node c. A single inverter would give reasonable results, but the inverter pair is even better because it provides a slope on c that is essentially independent of the rise time at a. *X5* is the load on *X4* to prevent node e from switching excessively fast, which would overpredict the significance of the gate-to-drain capacitance in *X4*.

Figure 5.21 lists a SPICE deck that uses the optimizer to automatically tune C_{delay} until the delays are equalized. This capacitance is divided by the total gate width (in μm) of *X4* to obtain the capacitance per micron of gate width $C_{permicron}$. This capacitance is listed as C_g (delay) in Table 5.5 for a variety of processes. Note that the deck sets diffusion area and perimeter to 0 to measure only the gate capacitance.

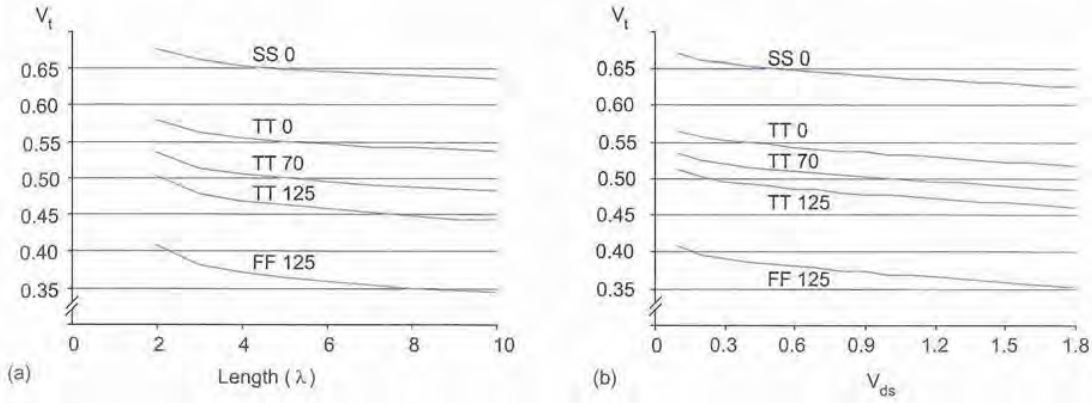


FIG 5.19 MOS threshold voltages

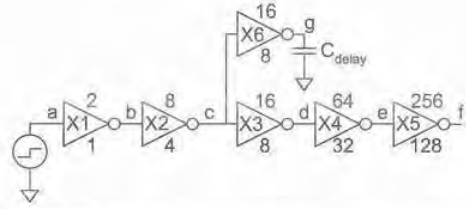


FIG 5.20 Circuit for extracting effective gate capacitance for delay estimation

Gate capacitance is also important for dynamic power consumption, as was given in EQ (4.29). The effective gate capacitance for power is typically somewhat higher than for delay because C_{gd} is effectively doubled from the Miller effect when we wait long enough for the drain to completely switch. Figure 5.22 shows a circuit for measuring gate capacitance for power purposes. A voltage step is applied to the input and the current out of the voltage source is integrated. The effective capacitance for dynamic power consumption is:

$$C_{\text{eff-power}} = \frac{\int i_{\text{in}}(t)dt}{V_{DD}} \tag{5.5}$$

Again, this capacitance can be divided by the total transistor width to find the effective capacitance per micron.

```
* capdelay.hsp
* Extract effective gate capacitance for delay estimation.
*-------------------------------------------------------------------------
* Parameters and models
*-------------------------------------------------------------------------
.option scale=90n
.param SUP=1.8 * Must set before calling .lib
.lib '../models/tsmc180/opconditions.lib' TT
.option post
*-------------------------------------------------------------------------
* Subcircuits
*-------------------------------------------------------------------------
.global  vdd   gnd
.subckt  inv   a     y
M1       y     a     gnd   gnd   NMOS   W=16 L=2 AD=0 AS=0 PD=0 PS=0
M2       y     a     vdd   vdd   PMOS   W=32 L=2 AD=0 AS=0 PD=0 PS=0
.ends
*-------------------------------------------------------------------------
* Simulation netlist
*-------------------------------------------------------------------------
Vdd      vdd   gnd   'SUPPLY'      * SUPPLY is set by .lib call
Vin      a     gnd   pulse 0 'SUPPLY' 1ns 0.5ns 0.5ns 4ns 10ns
X1       a     b     inv           * set appropriate slope
X2       b     c     inv   M=4     * set appropriate slope
X3       c     d     inv   M=8     * drive real load
X4       d     e     inv   M=32    * real load
X5       e     f     inv   M=128   * load on load (important!)
X6       c     g     inv   M=8     * drive linear capacitor
cdelay   g     gnd   'CperMicron*32*(16+32)*90n/1u' * linear capacitor
*-------------------------------------------------------------------------
* Optimization setup
*-------------------------------------------------------------------------
.measure errorR param='invR - capR' goal=0
.measure errorF param='invF - capF' goal=0
.param CperMicron=optrange(2f, 1f, 3.0f)
.model optmod opt itropt=30
.measure CperMic param = 'CperMicron'
*-------------------------------------------------------------------------
* Stimulus
*-------------------------------------------------------------------------
.tran .1ns 12ns SWEEP OPTIMIZE = optrange RESULTS=errorR,errorF MODEL=optmod
.measure invR
+       TRIG v(c)  VAL='SUPPLY/2' FALL=1
+       TARG v(d)  VAL='SUPPLY/2' RISE=1
.measure  capR
+       TRIG v(c)  VAL='SUPPLY/2' FALL=1
+       TARG v(g)  VAL='SUPPLY/2' RISE=1
.measure invF
+       TRIG v(c)  VAL='SUPPLY/2' RISE=1
+       TARG v(d)  VAL='SUPPLY/2' FALL=1
.measure  capF
+       TRIG v(c)  VAL='SUPPLY/2' RISE=1
+       TARG v(g)  VAL='SUPPLY/2' FALL=1
.end
```

FIG 5.21 CAPDELAY SPICE deck

5.4.4 Parasitic Capacitance

The parasitic capacitance associated with the source or drain of a transistor includes the gate-to-diffusion overlap capacitance, C_{gol}, and the diffusion area and perimeter capacitance C_{jb} and C_{jbsw}. As discussed in Section 5.3.4, some models assign a different capacitance C_{jbrwg} to the perimeter along the gate side. The diffusion capacitance is voltage-dependent, but as with gate capacitance, we can extract an effective capacitance averaged over the switching transition to use for delay estimation.

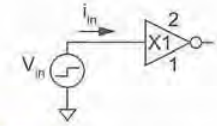


FIG 5.22 Circuit for extracting effective gate capacitance for power estimation

Figure 5.23 shows circuits for extracting these capacitances. They operate in much the same way as the gate capacitance extraction from Section 5.4.3. The first two fanout-of-4 inverters shape the input slope to match a typical gate. *X3* drives the drain of an OFF transistor *M1* with specified *W*, AD, and PD. *X4* drives a simple capacitor, whose value is optimized so that the delay of *X3* and *X4* are equal. This value is the effective capacitance of *M1*'s drain. Similar simulations must be run to find the parasitic capacitances of pMOS transistors.

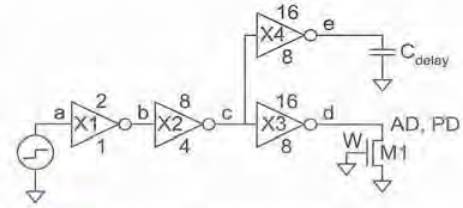


FIG 5.23 Circuit for extracting effective parasitic capacitance for delay estimation

Table 5.4 lists the appropriate values of *W*, AD, and PD to extract each of the capacitances. The sizes are chosen such that the gate delays and slope on node d are reasonable when a unit transistor is 16 λ wide (as in Figure 5.21). It also gives values to find the effective capacitance C_d of isolated contacted, shared contacted, and merged uncontacted diffusion regions. The capacitance is found, assuming the transistors are wide enough that the perimeter perpendicular to the polysilicon gate is a negligible fraction of the overall capacitance. The AD and PD dimensions are based on the layouts of Figure 2.9; you should substitute your own design rules. The total capacitance of shared and merged regions should be split between the two transistors sharing the diffusion node. The capacitance can be converted to units per micron (or per micron squared) by normalizing for the value of λ. For example, in a 180 nm process with λ = 0.09 μm, if C_{delay} is 53 fF for gate overlap, the capacitance per micron is

$$C_{gol} = \frac{53\ \text{fF}}{(1600\lambda)\left(\frac{0.09\mu\text{m}}{\lambda}\right)} = 0.37\frac{\text{fF}}{\mu\text{m}} \tag{5.6}$$

5.4.5 Effective Resistance

According to the RC delay model, if a unit transistor has gate capacitance *C*, parasitic capacitance C_d, and resistance R_n (for nMOS) or R_p (for pMOS), the rising and falling delays of a fanout-of-*h* inverter with a 2:1 *P/N* ratio can be found according to Figure 5.24. These delays can readily be measured from the fanout-of-4 inverter simulation in Figure 5.10 by changing *h*.

Table 5.4 Dimensions for diffusion capacitance extraction

	W (λ)	AD (λ²)	PD (λ)	To find effective C per micron
C_{gol}	1600	0	0	$C_{delay}/1600\lambda$ (per μm)
C_{jb}	0	8000	0	$C_{delay}/8000\lambda^2$ (per μm²)
C_{jbsw}	0	0	1600	$C_{delay}/1600\lambda$ (per μm)
C_{jbswg}	1600	0	1600	$C_{delay}/1600\lambda - C_{gol}$ (per μm)
C_d (isolated contacted)	1600	8000	3200	$C_{delay}/1600\lambda$ (per μm of gate width)
C_d (shared contacted)	3200	9600	3200	$C_{delay}/1600\lambda$ (per μm of gate width)
C_d (merged uncontacted)	3200	4800	3200	$C_{delay}/1600\lambda$ (per μm of gate width)

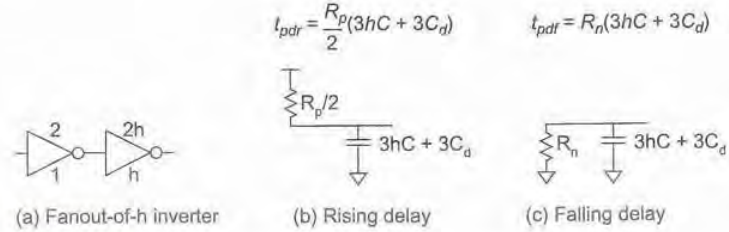


FIG 5.24 RC delay model for fanout-of-*h* inverter

The dependence on parasitics can be removed by calculating the difference between delays at different fanouts. For example, the difference between delays for $h = 3$ and $h = 4$ are

$$\Delta t_{pdr} = \frac{R_p}{2}(3 \cdot 4 \cdot C + 3C_d) - \frac{R_p}{2}(3 \cdot 3 \cdot C + 3C_d) = \tfrac{3}{2}R_pC$$
$$\Delta t_{pdf} = R_n(3 \cdot 4 \cdot C + 3C_d) - R_n(3 \cdot 3 \cdot C + 3C_d) = 3R_nC \quad (5.7)$$

As C is known from the effective gate capacitance extraction, R_n and R_p are readily calculated. These represent the effective resistance of single nMOS and pMOS transistors for delay estimation.

When two unit transistors are in series, each nominally would have the same effective resistance, giving twice the overall resistance. However, in modern processes where the transistors usually experience some velocity saturation, each transistor sees a smaller V_{ds} and hence less velocity saturation and a lower effective resistance. We can determine this resistance by simulating fanout-of-*h* tristates in place of inverters, as shown from *c* to *d* in Figure 5.25. By a similar reasoning, the difference between delays for $h = 2$ and $h = 3$ is

$$\Delta t_{pdr} = \tfrac{3}{2}(2R_{p\text{-series}})C$$
$$\Delta t_{pdf} = 3(2R_{n\text{-series}})C \quad (5.8)$$

As C is still known, we can extract the effective resistance of series nMOS and pMOS transistors for delay estimation and should expect this resistance to be slightly smaller than for single transistors.

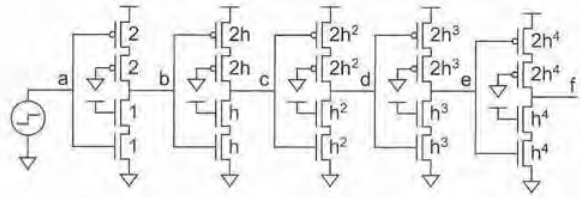


FIG 5.25 Circuit for extracting effective series resistance

It is important to use realistic input slopes when extracting effective resistance because the delay varies with input slope. *Realistic* means that the input and output edge rates should be comparable; if a step input is applied, the output will transition faster and the effective resistance will appear to decrease. h was chosen in this section to give stage efforts close to 4.

5.4.6 Comparison of Processes

Table 5.5 compares the characteristics of a variety of CMOS processes with feature sizes ranging from 2 μm down to 180 nm. Most are based on models published by MOSIS [Piña02] based on wafer test results, but one column uses models provided by TSMC (and indicates the test wafer results were slower than the TT vendor models). The MOSIS models use ACM = 0, so the diffusion sidewall capacitance is treated the same along the gate and the other walls. The 0.8 and 0.6 μm processes operate at either V_{DD} = 5 V (for higher speed) or V_{DD} = 3.3 V (for lower power). All characteristics are extracted for TTTT conditions (70° C).

The table indicates that the diffusion capacitance of an isolated contacted source or drain has been 1–2 fF/μm for both nMOS and pMOS transistors over many generations. The capacitance of a shared contacted diffusion region is slightly higher because it has more area and includes two gate overlaps. The capacitance of the merged diffusion reflects two gate overlaps but a smaller diffusion area.

The effective resistance of a 1 μm wide transistor has decreased with process scaling in proportion to the feature size f. However, the resistance of a unit (4/2 λ) nMOS transistor, $R/2f$, has remained roughly constant around 8 kΩ, as constant field scaling theory would predict. The effective resistance of pMOS transistors is 2–3 times that of nMOS transistors. A pair of nMOS transistors in series each have lower effective resistance than a single device because each has a smaller V_{ds} and thus experiences less velocity saturation. Series pMOS transistors show less pronounced improvement because they were not as velocity-saturated to begin with.

Table 5.5 Device characteristics for a variety of processes

Vendor		Orbit	AMI	HP	HP	AMI	AMI	TSMC	TSMC	TSMC	TSMC
Model		MOSIS	MOSIS	MOSIS	MOSIS	MOSIS	MOSIS	MOSIS	MOSIS	MOSIS	TSMC
Feature Size f	nm	2000	1600	800	800	600	600	350	250	180	180
V_{DD}	V	5	5	5	3.3	5	3.3	3.3	2.5	1.8	1.8
nMOS											
C_{gol}	fF/μm	0.37	0.18	0.30	0.30	0.20	0.20	0.27	0.57	0.73	0.37
C_{jb}	fF/μm²	0.05	0.15	0.06	0.09	0.25	0.28	0.73	1.23	0.78	0.77
C_{jbsw}	fF/μm	0.31	0.13	0.34	0.39	0.28	0.29	0.32	0.30	0.22	0.18
C_{jbswg}	fF/μm	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	0.24
C_d (isolated)	fF/μm	1.19	1.09	1.11	1.27	1.14	1.21	1.63	1.88	1.50	1.12
C_d (shared)	fF/μm	1.62	1.35	1.43	1.60	1.41	1.50	2.04	2.60	2.30	1.62
C_d (merged)	fF/μm	1.48	0.98	1.36	1.49	1.19	1.24	1.60	2.16	2.09	1.41
R_n (single)	kΩ•μm	30.3	25.2	10.1	14.2	9.19	11.9	5.73	4.02	2.96	2.69
R_n (series)	kΩ•μm	22.1	17.1	6.95	10.1	6.28	8.59	4.01	3.10	2.24	2.00
V_{tn} (const. I)	V	0.65	0.53	0.65	0.65	0.70	0.70	0.59	0.48	0.44	0.41
V_{tn} (linear ext.)	V	0.65	0.58	0.75	0.75	0.76	0.76	0.67	0.57	0.55	0.53
I_{dsat}	μA/μm	152	159	380	197	387	216	450	551	560	566
I_{off}	pA/μm	2.26	1.71	9.36	6.59	2.21	1.45	6.57	56.3	78.2	93.9
I_{crit} (linear ext.)	μA/μm	0.05	0.12	0.50	0.50	0.47	0.47	0.89	1.86	4.07	5.12
pMOS											
C_{gol}	fF/μm	0.36	0.23	0.32	0.32	0.29	0.29	0.30	0.66	0.66	0.33
C_{jb}	fF/μm²	0.13	0.16	0.32	0.37	0.42	0.48	0.91	1.39	0.94	0.85
C_{jbsw}	fF/μm	0.21	0.14	0.11	0.11	0.20	0.21	0.35	0.30	0.15	0.20
C_{jbswg}	fF/μm	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	0.33
C_d (isolated)	fF/μm	1.42	1.15	1.17	1.26	1.31	1.42	1.89	2.07	1.36	1.24
C_d (shared)	fF/μm	1.92	1.51	1.62	1.72	1.73	1.86	2.37	2.89	2.11	1.79
C_d (merged)	fF/μm	1.52	1.12	1.23	1.29	1.35	1.43	1.83	2.40	1.86	1.56
R_p (single)	kΩ•μm	67.1	62.8	26.7	40.8	19.9	29.6	16.1	8.93	6.55	6.51
R_p (series)	kΩ•μm	53.9	52.5	21.4	33.3	15.4	23.6	13.3	6.91	5.06	5.41
$\lvert V_{tp}\rvert$ (const. I)	V	0.72	0.77	0.91	0.91	0.90	0.90	0.83	0.46	0.45	0.43
$\lvert V_{tp}\rvert$ (linear ext.)	V	0.71	0.76	0.94	0.94	0.93	0.93	0.88	0.52	0.53	0.51
I_{dsat}	μA/μm	70.5	60.6	154	66.6	215.3	99.0	181	245	257	228
I_{off}	pA/μm	2.18	17.9	1.57	1.03	2.08	1.38	2.06	30.1	47.4	25.2
I_{crit} (linear ext.)	μA/μm	0.02	0.03	0.11	0.11	0.16	0.16	0.28	0.59	1.24	1.14
Gates											
C_g (delay)	fF/μm	1.77	1.67	1.67	1.60	1.55	1.48	1.90	2.30	2.13	1.67
C_g (power)	fF/μm	2.24	1.89	1.70	1.90	1.83	1.76	2.20	2.92	2.82	2.06
FO4 inv delay	ps	856	717	297	427	230	312	210	153	99.4	75.6

Threshold voltages are reported at V_{ds} = 100 mV for 16/2 λ devices using both the constant current (at I_{crit} = 0.1(W/L) μA for nMOS and 0.06(W/L) for pMOS) and linear extrapolation methods. Threshold voltages have generally decreased, but not as fast as channel length or supply voltage (because of subthreshold leakage). Therefore, the V_{DD}/V_t ratio is decreasing and pass transistor circuits with threshold drops do not perform well in modern processes.

Saturation current per micron has increased somewhat through aggressive device design as feature size decreases even though constant field scaling would suggest it should remain constant. OFF current was on the order of a few picoamperes per micron in the past, but is now exponentially increasing because of subthreshold conduction through devices with low threshold voltages. The current at threshold using the linear extrapolation method is somewhat higher than the constant current I_{crit}, corresponding to the higher threshold voltages found by the linear extrapolation method.

The gate capacitance for delay has held steady near 2 fF/μm for many generations, as scaling theory would predict. The gate capacitance for power is slightly higher than that for delay as discussed in Section 5.4.3.

The FO4 inverter delay has steadily improved with feature size as constant field scaling predicts. It fits our rule from Section 4.3.1 of 1/3 to 1/2 of the feature size, when delay is measured in picoseconds and feature size in nanometers.

5.4.7 Process and Environmental Sensitivity

Table 5.6 shows how the TSMC180 nm process characteristics vary with process corner, voltage, and temperature. The FO4 inverter delay varies by more than a factor of two between best and worst case. In the TT process, inverter delay varies by about 0.12% / °C and by about 1% for every percent of supply voltage change. These figures agree well with the Artisan library data from Section 4.7.4. Gate and diffusion capacitance change only slightly with process, but effective resistance is inversely proportional to supply voltage and highly sensitive to temperature and device corners. I_{off} subthreshold leakage rises dramatically at high temperature or in the fast corner where threshold voltages are lower.

5.5 Circuit Characterization

The device characterization techniques from the previous section are typically run once by engineers who are familiarizing themselves with a new process. SPICE is used more often to characterize entire circuits. This section gives some pointers on simulating paths and describes how to find the DC transfer characteristics, logical effort, and power consumption of logic gates.

Table 5.6 Process corners of TSMC 180 nm process

nMOS		T	F	S	F	S	T	T	T	T
pMOS		T	F	S	S	F	T	T	T	T
V_{DD}	V	1.8	1.98	1.62	1.8	1.8	1.98	1.62	1.8	1.8
T	°C	70	0	125	70	70	70	70	0	125
nMOS										
C_{gol}	fF/μm	0.37	0.39	0.35	0.37	0.37	0.37	0.37	0.37	0.37
C_{jb}	fF/μm^2	0.77	0.72	0.82	0.72	0.80	0.75	0.78	0.76	0.76
C_{jbsw}	fF/μm	0.18	0.17	0.19	0.17	0.19	0.18	0.18	0.18	0.18
C_{jbswg}	fF/μm	0.24	0.22	0.26	0.23	0.25	0.23	0.25	0.24	0.24
C_d (isolated)	fF/μm	1.12	1.09	1.16	1.08	1.16	1.10	1.13	1.12	1.12
C_d (shared)	fF/μm	1.62	1.58	1.65	1.58	1.66	1.60	1.64	1.62	1.62
C_d (merged)	fF/μm	1.41	1.40	1.43	1.38	1.45	1.40	1.43	1.42	1.41
R_n (single)	kΩ•μm	2.69	1.78	4.23	2.20	3.13	2.43	3.00	2.43	2.91
R_n (series)	kΩ•μm	2.00	1.23	3.42	1.51	2.42	1.78	2.32	1.76	2.17
V_{tn} (const. I)	V	0.41	0.35	0.47	0.30	0.51	0.41	0.41	0.45	0.37
V_{tn} (linear ext.)	V	0.53	0.48	0.60	0.43	0.63	0.53	0.53	0.58	0.50
I_{dsat}	μA/μm	566	834	354	624	508	656	476	623	520
I_{off}	pA/μm	93.9	61.4	84.9	1116	9.9	100	87.8	4.7	689
I_{crit} (linear ext.)	μA/μm	5.12	7.35	3.83	5.29	4.94	5.12	5.12	6.40	4.43
pMOS										
C_{gol}	fF/μm	0.33	0.34	0.31	0.33	0.33	0.33	0.33	0.33	0.33
C_{jb}	fF/μm^2	0.85	0.79	0.91	0.90	0.81	0.84	0.87	0.85	0.86
C_{jbsw}	fF/μm	0.20	0.19	0.21	0.21	0.19	0.20	0.20	0.20	0.20
C_{jbswg}	fF/μm	0.33	0.31	0.36	0.35	0.32	0.33	0.34	0.33	0.33
C_d (isolated)	fF/μm	1.24	1.20	1.28	1.28	1.20	1.23	1.26	1.24	1.24
C_d (shared)	fF/μm	1.79	1.75	1.83	1.84	1.73	1.76	1.80	1.78	1.78
C_d (merged)	fF/μm	1.56	1.53	1.58	1.60	1.51	1.54	1.57	1.55	1.56
R_p (single)	kΩ•μm	6.51	4.55	9.60	7.00	6.00	5.93	7.32	6.04	6.86
R_p (series)	kΩ•μm	5.41	3.65	8.03	5.98	4.73	4.84	6.08	4.98	5.68
V_{tn} (const. I)	V	0.43	0.43	0.45	0.50	0.36	0.43	0.43	0.50	0.38
V_{tp} (linear ext.)	V	0.51	0.51	0.52	0.57	0.44	0.51	0.51	0.57	0.46
I_{dsat}	μA/μm	228	345	144	209	248	273	184	238	222
I_{off}	pA/μm	25.2	5.21	748	7.66	127	26.7	23.7	2.76	938
I_{crit} (linear ext.)	μA/μm	1.14	1.40	0.95	1.11	1.16	1.14	1.14	1.20	1.09
Gates										
C_g (delay)	fF/μm	1.67	1.72	1.61	1.71	1.65	1.69	1.65	1.67	1.68
C_g (power)	fF/μm	2.06	2.12	1.98	2.06	2.07	2.09	2.03	2.06	2.06
FO4 inv delay	ps	75.6	53.3	112	74.4	77.6	69.7	83.9	69.4	80.6

5.5.1 Path Simulations

The delay of most static CMOS circuits today is computed with a static timing analyzer (see Section 8.4.1.5). As long as the noise sources (particularly coupling and power supply noise) are controlled, the circuits will operate correctly and will correlate reasonably well with static timing predictions. However, SPICE-level simulation is important for sensitive circuits such as the clock generator and distribution network, custom memory arrays, and novel circuit techniques.

Most experienced designers begin designing paths based on simple models in order to understand what aspects are most important, evaluate design tradeoffs, and obtain a qualitative prediction of the results. The ideal Shockley transistor models, RC delay models, and logical effort are all helpful here because they are simple enough to give insight. When a good first-pass design is ready, the designer simulates the circuit to verify that it operates correctly and meets delay and power specifications. Just as few new software programs run correctly before debugging, the simulation often will be incorrect at first. Unless the designer knows what results to expect, it is tempting to trust the false results that are nicely printed with beguilingly many significant figures. Once the circuit appears to be correct, it should be checked across design corners to verify that it operates in all cases. Section 4.7.4 gave examples of circuits sensitive to various corners.

Simulation is cheap, but silicon revisions are very expensive. Therefore, it is important to construct a circuit model that captures all of the relevant conditions, including real input waveforms, appropriate output loading, and adequate interconnect models. When matching is important, you must consider the effects of mismatches that are not given in the corner files (see Section 5.5.5). However, as SPICE decks get more complicated, they run more slowly, accumulate more mistakes, and are more difficult to debug. A good compromise is to start simple and gradually add complexity, ensuring after each step that the results still make sense.

5.5.2 DC Transfer Characteristics

The `.dc` card is useful for finding the transfer characteristics and noise margins of logic gates. Figure 5.27 shows an example of characterizing static and dynamic inverters (dynamic logic is covered in Section 6.2.4). Figure 5.26(a and b) show the circuit schematics of each gate. Figure 5.26(c) shows the simulation results. The static inverter characteristics are nearly symmetric around $V_{DD}/2$. The dynamic inverter has a lower switching threshold and its output drops abruptly beyond this threshold because positive feedback turns off the keeper.

Note that when the input `a` is '0' and the dynamic inverter is in evaluation (ϕ = '1'), the output would be stable at either '0' or '1.' To find the transfer characteristics, we initialize the gate with a '1' output using the `.ic` command.

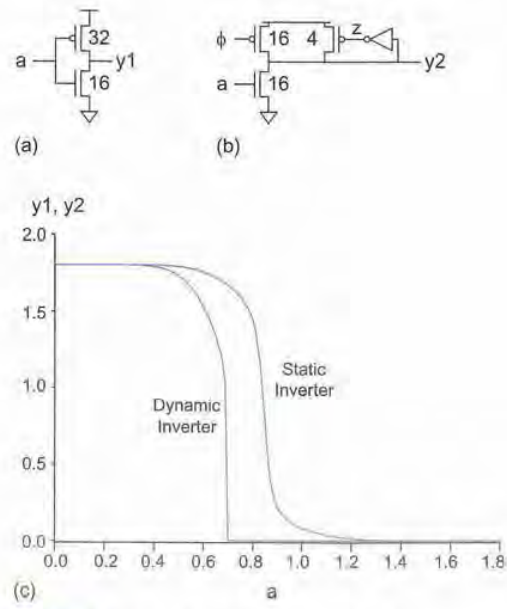


FIG 5.26 Circuits for DC transfer analysis

5.5.3 Logical Effort

The logical effort and parasitic delay of each input of a gate can be measured by fitting a straight line to delay vs. fanout simulation results. As with the FO4 inverter example, it is important to drive the gate with an appropriate input waveform and to provide two stages of loads. Figure 5.28(a) shows an example of a circuit for characterizing the delay of a 2-input NAND gate *X3* using the `M` parameter to simulate multiple gates in parallel. Figure 5.28(b) shows that the delay vs. fanout in a 180 nm process fit a straight line of $d_{abs} = 16.7h + 28.9$ ps within 0.5 ps even though the transistors experience all sorts of nonlinear and nonideal effects. This shows that the linear delay model is quite accurate as long as the input and output slopes are consistent.

The `SWEEP` command is convenient to vary the fanout and repeat the transient simulation multiple times. For example, the following card runs eight simulations varying `H` from 1 to 8 in steps of 1.

```
.tran 1ps 1000ps SWEEP H 1 8 1
```

To characterize an entire library, you can write a script in a language such as Perl that generates the appropriate SPICE decks, invokes the simulator, and postprocesses the list files to extract the data and do the curve fit.

Recall that τ is the coefficient of h (i.e., the slope) in a delay vs. fanout plot for an inverter. Given that τ = 15 ps in this process, we see the NAND gate has a logical effort of 16.7/15 = 1.11 and a parasitic delay of 28.9/15 = 1.93.

Table 5.7 compares the logical effort and parasitic delay of the different inputs of multi-input NAND gates for rising, falling, and average output transitions in the TSMC 180 nm process. For rising and falling transitions, we still normalize against the value of τ found from the average delay of an inverter. Input *A* is the outermost (closest to power or ground). As discussed in Section 6.2.1.3, the outer input has higher parasitic delay, but slightly lower logical effort. The rising and falling delays in this process are quite different because pMOS transistors have less than half the mobility of nMOS transistors and because the nMOS transistors are quite velocity-saturated so that series transistors have less resistance than expected.

```
* invdc.sp
* Static and dynamic inverter DC transfer characteristics

*-------------------------------------------------------------------------
* Parameters and models
*-------------------------------------------------------------------------
.param SUPPLY=1.8
.option scale=90n
.include '../models/tsmc180/models.sp'
.temp 70
.option post

*-------------------------------------------------------------------------
* Simulation netlist
*-------------------------------------------------------------------------
Vdd   vdd   gnd   'SUPPLY'
Va    a     gnd   0
Vclk  clk   gnd   'SUPPLY'
* Static Inverter
M1    y1    a     gnd   gnd   NMOS   W=16   L=2
M2    y1    a     vdd   vdd   PMOS   W=32   L=2
* Dynamic Inverter
M3    y2    a     gnd   gnd   NMOS   W=16   L=2
M4    y2    clk   vdd   vdd   PMOS   W=16   L=2
M5    y2    z     vdd   vdd   PMOS   W=4    L=2
M6    z     y2    gnd   gnd   NMOS   W=4    L=2
M7    z     y2    vdd   vdd   PMOS   W=8    L=2
.ic V(y2) = 'SUPPLY'

*-------------------------------------------------------------------------
* Stimulus
*-------------------------------------------------------------------------
.dc Va 0 1.8 0.01
.end
```

FIG 5.27 INVDC SPICE deck for DC transfer analysis

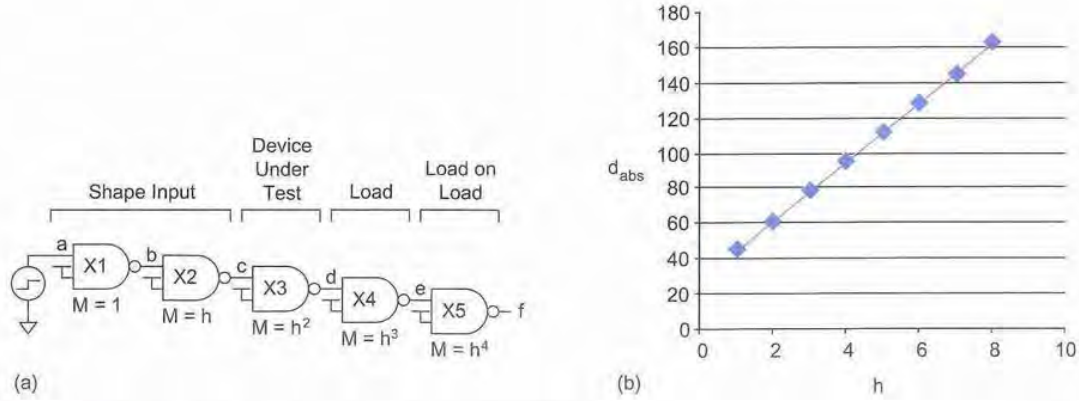


FIG 5.28 Logical effort characterization of 2-input NAND gate

Table 5.8 compares the average logical effort and parasitic delay of a variety of gates in many different processes. In each case, the simulations are performed in the TTTT corner for the outer input. For reference, the values of a FO4 inverter delay and τ are given for each process. The logical effort of NAND gates are substantially lower than we predicted in Section 4.2.3 because velocity saturation causes the resistance of N series nMOS transistors to be less than N times that of a single transistor. NOR gate efforts are slightly lower, but the effect is less pronounced because pMOS transistors have lower mobility and thus experience less velocity saturation. The efforts are fairly consistent across process and voltage. The parasitic delays show greater spread because of the variation in the relative capacitances of diffusion and gates.

Table 5.7 Logical effort and parasitic delay of different inputs of multi-input NAND gates

# of inputs	Input	Rising Logical Effort g_u	Falling Logical Effort g_u	Average Logical Effort g	Rising Parasitic Delay p_u	Falling Parasitic Delay p_d	Average Parasitic Delay p
2	*A*	1.45	0.82	**1.14**	2.37	1.42	**1.90**
	B	1.36	0.97	**1.17**	1.60	1.22	**1.41**
3	*A*	1.82	0.80	**1.31**	4.90	2.11	**3.51**
	B	1.73	0.95	**1.34**	3.94	1.86	**2.90**
	C	1.63	1.10	**1.36**	2.60	1.54	**2.07**
4	*A*	1.96	0.89	**1.42**	6.54	3.08	**4.81**
	B	1.86	1.02	**1.44**	5.71	2.84	**4.28**
	C	1.80	1.14	**1.47**	4.69	2.42	**3.56**
	D	1.71	1.28	**1.49**	3.26	1.97	**2.62**

This data includes more detail than the designer typically wants when doing design by hand; the coarse estimates of logical effort from Table 4.2 are generally sufficient for an initial design. However, the accurate delay vs. fanout information, often augmented with input slope dependence, is very important when characterizing a standard cell library to use with a static timing analyzer

Table 5.8 Logical effort and parasitic delay of gates in various processes

Vendor		Orbit	AMI	HP	HP	AMI	AMI	TSMC	TSMC	TSMC	TSMC
Model		MOSIS	MOSIS	MOSIS	MOSIS	MOSIS	MOSIS	MOSIS	MOSIS	MOSIS	TSMC
Feature Size f	nm	2000	1600	800	800	600	600	350	250	180	180
V_{DD}	V	5	5	5	3.3	5	3.3	3.3	2.5	1.8	1.8
FO4 inv delay	ps	856	717	297	427	230	312	210	153	99.4	75.6
τ	ps	170	143	59	84	45	60	40	30	20	15
Logical Effort											
Inverter		1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
NAND2		1.13	1.07	1.07	1.09	1.05	1.08	1.12	1.12	1.17	1.14
NAND3		1.32	1.22	1.21	1.25	1.19	1.24	1.29	1.29	1.36	1.31
NAND4		1.48	1.35	1.33	1.37	1.33	1.38	1.44	1.43	1.49	1.42
NOR2		1.57	1.62	1.59	1.58	1.58	1.60	1.52	1.50	1.46	1.50
NOR3		2.16	2.32	2.23	2.23	2.23	2.30	2.07	2.02	1.94	2.00
NOR4		2.55	2.74	2.61	2.64	2.57	2.68	2.46	2.37	2.27	2.38
Parasitic Delay											
Inverter		1.08	1.05	1.05	1.13	1.18	1.25	1.33	1.18	0.95	1.03
NAND2		1.87	1.79	1.85	2.05	1.92	2.10	2.28	2.07	1.74	1.90
NAND3		3.34	3.22	3.30	3.75	3.40	3.79	4.15	3.65	3.08	3.51
NAND4		4.64	4.42	4.54	5.12	4.70	5.23	5.75	5.01	4.26	4.81
NOR2		2.86	2.97	2.91	3.13	3.29	3.56	3.52	2.95	2.41	2.85
NOR3		5.65	6.22	6.05	6.47	7.02	7.70	6.89	5.61	4.49	5.57
NOR4		7.92	8.71	8.48	9.14	9.68	10.67	9.71	7.93	6.38	7.98

5.5.4 Power and Energy

Recall from Section 4.4 that energy and power are proportional to the supply current. They can be measured based on the current out of the power supply voltage source. For example, the following code uses the `INTEGRAL` command to measure charge and energy delivered to a circuit during the first 10 ns.

```
.measure charge INTEGRAL I(vdd) FROM=0ns TO=10ns
.measure energy param='charge*SUPPLY'
```

Alternatively, HSPICE allows you to directly measure the instantaneous and average power delivered by a voltage source.

```
.print P(vdd)
.measure pwr AVG P(vdd) FROM=0ns TO=10ns
```

Sometimes it is helpful to measure the power consumed by only one gate in a larger circuit. In that case, you can use a separate voltage source for that gate and measure power only from that source. Unfortunately, this means that `vdd` cannot be declared as `.global`.

When the input of a gate switches, it delivers power to the supply through the gate-to-source capacitances. Be careful to differentiate this input power from the power drawn by the gate discharging its internal and load capacitances.

5.5.5 Simulating Mismatches

Many circuits are sensitive to mismatches between nominally identical transistors. For example, the sense amplifiers in Fig 11.17 should respond to a small differential voltage between the inputs. Mismatches between nominally identical transistors add an offset that can significantly increase the required voltage. Merely simulating in different design corners is inadequate because the transistors will still match each other. As discussed in Section 4.7.5, the mismatch between currents in two nominally identical transistors can be primarily attributed to shifts in the threshold voltage and channel length. Figure 5.29 shows an example of simulating this mismatch. Each transistor is replaced by an equivalent circuit with a different channel length and a voltage source modeling the difference in threshold voltage. Note that many binned BSIM models do not allow setting the transistor length shorter than the minimum value supported by the process. Obtaining data on parameter variations can be difficult. Section 4.7.5 gives examples of channel length and threshold variation in a 180 nm process.

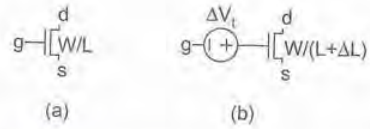


FIG 5.29 Modeling mismatch

In many cases, the transistors are not adjacent and may see substantial differences in voltage and temperature. For example, two clock buffers in different corners of the chip that see different environments will cause skew between the two clocks. The voltage difference can be modeled with two different voltage sources. The temperature difference is most easily handled through two separate simulations at different temperatures.

5.5.6 Monte Carlo Simulation

Monte Carlo simulation can be used to find the effects of random variations on a circuit. It consists of running a simulation repeatedly with different randomly chosen parameter offsets. To use Monte Carlo simulation, the transistor models must include the offset parameters. For example, Figure 5.30 shows the nMOS model card from Figure 5.12 modified to handle length, width, and threshold voltage changes. XL and XW specify offsets to the nominal length and width.

```
.model NMOS NMOS (LEVEL=1 TOX=40e-10 KP=155E-6 LAMBDA=0.2
+                 VTO='0.4+dvthn' PHI=0.93 GAMMA=0.6
+                 CJ=9.8E-5 PB=0.72 MJ=0.36
+                 CJSW=2.2E-10 PHP=7.5 MJSW=0.1
+                 XL='dxl' XW='dxw')
```

FIG 5.30 Sample Level 1 MODEL card with offsets

Consider modifying the FO4 inverter delay simulation from Figure 5.10 to handle the length and threshold voltage mismatches given in Section 4.7.5. The `.tran` card must be augmented to include the number of times to repeat the simulation with random values. The random distributions of each of the parameters must also be specified. Figure 5.31 shows these changes for 30 Monte Carlo repetitions. The channel length is assigned a uniform offset in the range of 0 +/− 12.5 nm. The nMOS threshold voltage offset has a Gaussian distribution centered around zero with a 1σ variation of 16.8 mV.

```
.param dxl=aunif(0,12.5nm) dvthn=agauss(0,16.8m,1) dvthp=agauss(0,14.6m,1)
.tran 1ps 1000ps SWEEP MONTE=30
```

FIG 5.31 Monte Carlo transient analysis

The `.measure` cards report average, minimum, maximum, and standard deviation computed from the repeated simulations. In the FO4 example, the average propagation delay is 74.9 ps with a standard deviation of 3.38 ps caused primarily by the channel length variation.

5.6 Interconnect Simulation

Interconnect parasitics can be a very important factor in overall delay. When an actual layout is available, the wire geometry can be directly extracted. If only the schematic is available, the designer may need to estimate wire lengths. For small gates, even the capacitances of the wires inside the gate are important. Therefore, some companies use *parasitic estimator* tools to guess wire parasitics in schematics based on the number and size of the transistors. In any case, the designer must explicitly model long wires based on their estimated lengths in the floorplan.

Once wire length and pitch are known or estimated, they can be converted to a wire resistance R and capacitance C using the methods discussed in Section 4.5. A short wire (where wire resistance is much less than gate resistance) can be modeled as a lumped



FIG 5.32 Four-segment π model for interconnect

capacitor. A longer wire can be modeled with a multisegment π-model. A four-segment model such as the one shown in Figure 5.32 is generally quite accurate. The model can be readily extended to include coupling between adjacent lines.

In general, interconnect consists of multiple interacting signal and power/ground lines [Young00]. For example, Figure 5.33(a) shows a pair of parallel signals running between a pair of ground wires. Although it is possible to model the ground lines with a resistance and inductance per unit length, it is usually more practical to treat the supply networks as ideal, then account for power supply noise separately in the noise budget. Figure 5.33(b) shows an equivalent circuit using a single π-segment model. Each line has a series resistance and inductance, a capacitance to ground, and mutual capacitance and inductance. The mutual elements describe how a changing voltage or current in one conductor induce a current or voltage in the other.

HSPICE also supports the `W` element that models lossy multiconductor transmission lines. This is more convenient than constructing an enormous π-model with resistance, capacitance, inductance, mutual capacitance, and mutual inductance. Moreover, HSPICE has a built-in two-dimensional field solver that can compute all of the terms from a cross-sectional description of the interconnect. Figure 5.34 gives a SPICE deck that uses the field solver to extract the element values and models the lines with the `W` element.

The deck describes a two-dimensional cross-section of the interconnect that the field solver uses to extract the electrical parameters. The interconnect consists of the two signal traces between two ground wires. Each wire is 2 μm wide and 0.7 μm thick. The copper wires are sandwiched with 0.9 μm of low-k ($\epsilon = 3.55\epsilon_0$) dielectric above and below. The $N = 2$ signal traces are spaced 6 μm from the ground lines and 2 μm from each other and have a length of 6 mm. The HSPICE field solver is quite flexible and is fully documented in the HSPICE manual. It generates the transmission line model and writes it to the

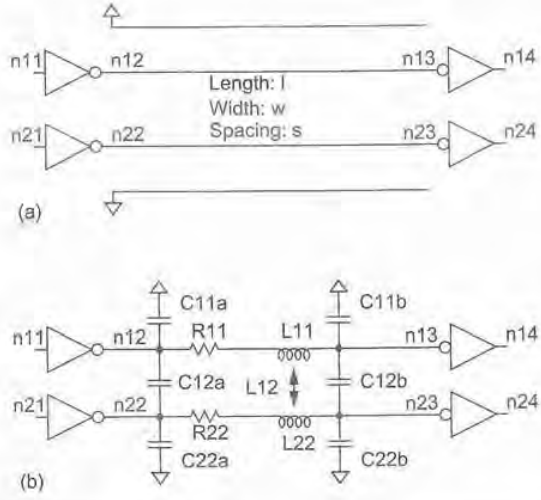


FIG 5.33 Lossy multiconductor transmission lines

coplanar.rlgc file. The file contains resistance, capacitance, and inductance matrices and is shown in Figure 5.35.

```
* interconnect.sp
*-----------------------------------------------------------------------
* Parameters and models
*-----------------------------------------------------------------------
.param SUPPLY=1.8
.include '../models/tsmc180/models.sp'
.temp 70
.option post
*-----------------------------------------------------------------------
* Subcircuits
*-----------------------------------------------------------------------
.global vdd gnd
.subckt inv a y N=360nm P=720nm
M1   y   a   gnd   gnd   NMOS   W='N'   L=180nm
+ AS='N*450nm' PS='2*N+900nm' AD='N*450nm' PD='2*N+900nm'
M2   y   a   vdd   vdd   PMOS   W='P'   L=180nm
+ AS='P*450nm' PS='2*P+900nm' AD='P*450nm' PD='2*P+900nm'
.ends
*-----------------------------------------------------------------------
* Compute transmission line parameters with field solver
*-----------------------------------------------------------------------
.material     oxide          DIELECTRIC           ER=3.55
.material     copper         METAL                CONDUCTIVITY=57.6meg
.layerstack   chipstack      LAYER=(oxide,2.5um)
.fsoptions    opt1           ACCURACY=MEDIUM      PRINTDATA=YES
.shape        widewire       RECTANGLE            WIDTH=2um    HEIGHT=0.7um
.model        coplanar       W                    MODELTYPE=FieldSolver
+ LAYERSTACK=chipstack       FSOPTIONS=opt1       RLGCFILE=coplanar.rlgc
+ CONDUCTOR=(SHAPE=widewire ORIGIN=(0,0.9um)     MATERIAL=copper TYPE=reference)
+ CONDUCTOR=(SHAPE=widewire ORIGIN=(8um,0.9um)   MATERIAL=copper)
+ CONDUCTOR=(SHAPE=widewire ORIGIN=(12um,0.9um)  MATERIAL=copper)
+ CONDUCTOR=(SHAPE=widewire ORIGIN=(20um,0.9um)  MATERIAL=copper TYPE=reference)
*-----------------------------------------------------------------------
* Simulation netlist
*-----------------------------------------------------------------------
Vdd  vdd   gnd   'SUPPLY'
Vin  n11   gnd   PULSE   0 'SUPPLY' 0ps 100ps 100ps 500ps 1000ps
W1   n12   n22   gnd    n13   n23   gnd   FSmodel=coplanar N=2 l=6mm
X1   n11   n12   inv    M=40
X2   n13   n14   inv    M=20
X3   gnd   n22   inv    M=40
X4   n23   n24   inv    M=20
*-----------------------------------------------------------------------
* Stimulus
*-----------------------------------------------------------------------
.tran 1ps 1000ps
.end
```

FIG 5.34 SPICE deck for lossy multiconductor transmission line

```
* L(H/m), C(F/m), Ro(Ohm/m), Go(S/m), Rs(Ohm/(m*sqrt(Hz)), Gd(S/(m*Hz))

.MODEL coplanar W MODELTYPE=RLGC, N=2

+ Lo =  6.68161e-007
+       3.67226e-007  6.68161e-007
+ Co =  2.53841e-011
+       -1.36778e-011  2.53841e-011
+ Ro =  12400.8
+       0  12400.8
+ Go =  0
+       0  0
```

FIG 5.35 coplanar.rlgc file

The matrices require a bit of effort to interpret. They are symmetric around the diagonal so only the lower half is printed. The resistances are $R_{11} = R_{22} = 12.4\ \Omega$/mm. The inductances are $L_{11} = L_{22} = 0.67$ nH/mm and $L_{12} = 0.37$ nH/mm. The capacitance matrix represents coupling capacitances with negative numbers and places the sum of all the capacitances for a trace on the diagonal. Therefore, $C_{11} = C_{22} = 0.117$ pF/mm and $C_{12} = 0.137$ pF/mm. In the π-model, half of each of these capacitances is lumped at each end.

Figure 5.36 shows the voltages along the wires. Observe the *ringing* (i.e., oscillation) caused by the wide wires with high inductance and low resistance driven with sharp edge rates. If the inductance were reduced by moving the ground lines closer to the conductors, the ringing would decrease. The switching line couples onto the quiet victim, causing noise at both ends as well.

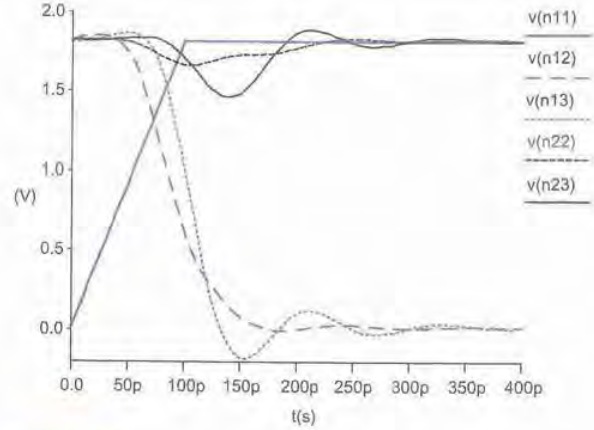


FIG 5.36 Transmission line response

5.7 Pitfalls and Fallacies

Failing to estimate diffusion and interconnect parasitics in simulations

The diffusion capacitance can account for 20% of the delay of an FO4 inverter and more than 50% of the delay of a high fan-in, low fanout gate. Be certain when simulating circuits that the area and perimeter of the source and drain are included in the simulations. Interconnect capacitance is also important, but difficult to estimate. For long wires, the capacitance and RC delay represent most of the path delay. A common error is to ignore wires while doing circuit design at the schematic level, and then discover after layout that the wire delay is important enough to demand major circuit changes and complete change of the layout.

Applying inappropriate input waveforms

Gate delay is strongly dependent on the rise/fall time of the input. For example, the propagation delay of an inverter is substantially shorter when a step input is applied than when an input with a realistic rise time is provided.

Applying inappropriate output loading

Gate delay is even more strongly dependent on the output loading. Some engineers, particularly those in the marketing department, measure gate delay as the delay of an unloaded inverter. This is about one-fifth of the delay of an FO4 inverter or other gate with "typical" loading. When simulating a critical path, it is important to include the estimated load that the final stage must drive.

Choosing inappropriate transistor sizes

Gate delay also depends on transistor widths. Some papers compare a novel design with carefully selected transistor sizes to a conventional design with poorly selected sizes, and arrive at the misleading conclusion that the novel design is superior.

Identifying the incorrect critical path

During preliminary design, it is much more efficient to compare circuits by modeling only the critical paths rather than the entire circuit. However, this requires that the designer correctly identify the path that will be most critical; sometimes this requires much consideration.

Failing to account for hidden scale factors

Many CAD systems introduce scaling factors. For example, a circuit can be drawn with one set of design rules and automatically scaled to the next process generation. The CAD tools may introduce a scaling factor to reflect this change. Specifying the proper transistor sizes reflecting this scaling is notoriously tricky. Simulation results will look good, but mean nothing if scaling is not accounted for properly.

Blindly trusting results from SPICE

Novice SPICE users often trust the results of simulation far too much. This is exacerbated by the fact that SPICE prints results to many significant figures and generates pretty waveforms. As we have seen, there are a multitude of reasons why simulation results may not reflect the behavior of the real circuit.

When first using a new process or tool set, always predict what the results should be for some simple circuits (e.g., an FO4 inverter) and verify that the simulation matches expectation. It doesn't hurt to be a bit paranoid at first. After proving that the flow is correct, lock down all the models and netlist generation scripts with version control if possible. That way, if any changes are made, a valid reason for the change must be evident and the simulations can be revalidated. The motto is: Check and recheck (but don't become compulsive!).

In general, assume SPICE decks are buggy until proven otherwise. If the simulation does not agree with your expectations, look closely for errors or inadequate modeling in the deck.

Using SPICE in place of thinking

A related error, common among perhaps the majority of circuit designers, is to use SPICE too much and one's brain too little. Circuit simulation should be guided by analysis.

Making common SPICE deck errors

Some of the common mistakes in SPICE decks include [Horowitz02]

- Omitting the comment on the first line
- Omitting the new line at the end of the deck
- Omitting the `.option` post command when using a waveform viewer
- Leaving out diffusion parasitics
- Forgetting to set initial values for dynamic logic or sequential circuits

Using incorrect dimensions when `.option scale` is not set

If `.option scale` is not used, a transistor with W = 4, L = 2 would be interpreted as 4 by 2 meters! This often is outside the legal range of sizes in a BSIM model file, causing SPICE to produce error messages. Similarly, a drain diffusion of 3 x 0.5 μm should be specified as PD = 7u AD = 1.5p as opposed to the common mistakes of PD = 7 AD = 1.5 or PD = 7u AD = 1.5u.

Summary

When used properly, SPICE is a powerful tool to characterize the behavior of CMOS circuits. This chapter began with a brief tutorial showing how to perform DC and transient analyses to characterize and optimize simple circuits. SPICE supports many different transistor models. At the time of writing, the BSIM3v3 model is most widely used and describes MOSFET behavior quite well for most digital applications. When specifying the MOSFET connection, you must include not only the terminal connections (drain, gate, source, and body) and width and length, but also the area and perimeter of the source and drain that are used to compute parasitic capacitance.

Modern SPICE models have so many parameters that they are intractable for hand calculations. However, the designer can perform some simple simulations to characterize a process. For example, it is helpful to know the effective gate capacitance and resistance, the diffusion capacitance, and the threshold voltage and leakage current You can also determine the delay of a fanout-of-4 inverter and the logical effort and parasitic delay of a library of gates to make quick estimates of circuit performance.

Most designers use SPICE to characterize real circuits. During preliminary design, you can model the critical path to quickly determine whether a circuit will meet performance requirements. A good model describes not only the circuit itself, but also the input edge rates, the output loading, and parasitics such as diffusion capacitance and interconnect. Most interconnect can be represented with a four-segment π model, although when inductance becomes important, the lossy multiconductor transmission line `W` element is convenient. Novel and "risky" circuits should be simulated in multiple design corners to ensure they will work correctly across variations in processing and environment. As

SPICE is prone to garbage-in, garbage-out, it is often best to begin with a simple model and debug until it matches expectations. Then more detail can be added and tested incrementally.

Exercises

Note: the book Web page (see preface) contains SPICE models and characterization scripts used to generate the data in this chapter. Unless otherwise stated, try the exercises using the mosistsmc180 model file (extracted by MOSIS from test structures manufactured on the TSMC 180 nm process) in TTTT conditions.

5.1 Find the average propagation delay of a fanout-of-5 inverter by modifying the SPICE deck in Figure 5.10.

5.2 By what percentage does the delay of Exercise 5.1 change if the input is driven by a voltage step rather than a pair of shaping inverters?

5.3 By what percentage does the delay of Exercise 5.1 change if *X*5, the load on the load, is omitted?

5.4 Find the input and output logic levels and high and low noise margins for an inverter with a 3:1 *P/N* ratio.

5.5 What *P/N* ratio maximizes the smaller of the two noise margins for an inverter?

5.6 Generate a set of eight I-V curves like those of Figure 5.16 for nMOS and pMOS transistors in your process.

5.7 The `char.pl` Perl script runs a number of simulations to characterize a process. Look for SPICE models for a newer process on the MOSIS Web site. Use the script to add another column to Table 5.5 for the new process.

5.8 The `charlib.pl` script runs a number of simulations to extract logical effort and parasitic delay of gates in a specified process. Add another column to Table 5.8 for a new process.

5.9 Use the `charlib.pl` script to find the logical effort and parasitic delay of a 5-input NAND gate for the outermost input.

5.10 Exercise 4.10 compares two designs of 2-input AND gates. Simulate each design and compare the average delays. What values of x and y give least delay? How much faster is the delay than that achieved using values of x and y suggested from logical effort calculations? How does the best delay compare to estimates using logical effort? Let $C = 10$ μm of gate capacitance.

5.11 Exercise 4.13 asks you to estimate the delay of a logic function. Simulate your design and compare your results to your estimate. Let one unit of capacitance be a minimum-sized transistor.



Combinational Circuit Design

6

6.1 Introduction

In Chapter 1, we introduced CMOS logic with the assumption that MOS transistors act as simple switches. *Static CMOS* gates used complementary nMOS and pMOS networks to drive '0' and '1' outputs, respectively. In Chapter 4, we used the RC delay model and logical effort to better understand the sources of delay in static CMOS logic.

In this chapter, we examine alternative CMOS logic configurations, called *circuit families*. While the vast majority of designs synthesize exclusively onto static CMOS libraries and even custom designs use static CMOS for 95% of the logic, high speed, low power, or density restrictions may force another solution. You should always use the circuit that satisfies the application and requires the least design and verification effort. The most commonly used alternative circuit families are ratioed circuits, dynamic circuits, and pass-transistor circuits.

As a grossly oversimplified yet still useful model, the delay of a logic gate depends on its output current I, load capacitance C, and output voltage swing ΔV, as given in EQ (6.1).

$$t \propto \frac{C}{I}\Delta V \tag{6.1}$$

Faster circuit families attempt to reduce one of these three terms. nMOS transistors provide more current than pMOS for the same size and capacitance, so nMOS networks are preferred. Observe that the logical effort is proportional to the C/I term because it is determined by the input capacitance of a gate that can deliver a specified output current.

One drawback of static CMOS is that it requires both nMOS and pMOS transistors on each input. During a falling output transition, the pMOS transistors add significant capacitance without helping the pull-down current; hence, static CMOS has a relatively large logical effort. Many faster circuit families seek to drive only nMOS transistors with the inputs, thus reducing capacitance and logical effort. An alternative mechanism must be provided to pull the output high. Determining when to pull outputs high involves monitoring the inputs, outputs, or some clock signal. Monitoring inputs and outputs inevitably loads the nodes, so clocked circuits are often fastest if the clock can be provided at the ideal time. Another drawback of static CMOS is that all the node voltages must transition between 0 and V_{DD}. Some circuit families use reduced voltage swings to improve

propagation delays (and power consumption). This advantage must be weighed against the delay and power of amplifying outputs back to full levels later or the costs of tolerating the reduced swings.

A host of other circuit families have been proposed, but most have never been used in commercial products and are doomed to reside on dusty library shelves. Every transistor contributes capacitance, so most fast structures are simple. Nevertheless, we will describe many of these circuits as a record of ideas that have been explored. A few hold promise for the future, particularly in specialized applications. Many texts simply catalog these circuit families without making judgments. This book attempts to evaluate the circuit families so that designers can concentrate their efforts on the most promising ones, rather than searching for the "gotchas" that were not mentioned in the original papers. Of course, any such evaluation runs the risk of overlooking advantages or becoming incorrect as technology changes, so you should use your own judgment.

Static CMOS logic is particularly popular because of its robustness. Given the correct inputs, it will eventually produce the correct output so long as there were no errors in logic design or manufacturing. Other circuit families are prone to numerous pathologies, including charge sharing, leakage, threshold drops, and ratioing constraints. When using alternative circuit families, it is vital to understand the pathologies and check that the circuits will work correctly in all design corners.

Because we are interested in building performance-optimized chips, both circuit and physical design must be considered. These two phases of design are intimately meshed. The density, behavior, and power dissipation of circuits can have a direct impact on any high-level architectural decision and can allow or preclude options based on the selection of a logic style. For this reason, it is important that the architect and microarchitect have some idea of low-level circuit options.

6.2 Circuit Families

Static CMOS circuits with complementary nMOS pull-down and pMOS pull-up networks are used for the vast majority of logic gates in integrated circuits. They have good noise margins, and are fast, low power, insensitive to device variations, easy to design, widely supported by CAD tools, and readily available in standard cell libraries. When noise does exceed the margins, the gate delay increases because of the glitch, but the gate eventually will settle to the correct answer. Indeed, many ASIC methodologies only allow static CMOS circuits. This section begins with a number of techniques for optimizing static CMOS circuits.

Nevertheless, performance or area constraints occasionally dictate the need for other circuit families. The remainder of this section describes the most popular circuit families, including ratioed and dynamic circuits and pass transistors.

6.2.1 Static CMOS

Designers accustomed to AND and OR functions must learn to think in terms of NAND and NOR to take advantage of static CMOS. In manual circuit design, this is often done through bubble pushing. Compound gates are particularly useful to perform complex functions with relatively low logical efforts. When a particular input is known to be latest, the gate can be optimized to favor that input. Similarly, when either the rising or falling edge is known to be more critical, the gate can be optimized to favor that edge. We have focused on building gates with equal rising and falling delays; however, using smaller pMOS transistors can reduce delay, power, and area. In processes with multiple threshold voltages, multiple flavors of gates can be constructed with different speed/leakage power tradeoffs.

6.2.1.1 Bubble Pushing CMOS stages are inherently inverting, so AND and OR functions must be built from NAND and NOR gates. DeMorgan's Law helps with this conversion:

$$\overline{A \cdot B} = \overline{A} + \overline{B}$$
$$\overline{A + B} = \overline{A} \cdot \overline{B} \qquad (6.2)$$

These relations are illustrated graphically in Figure 6.1. A NAND gate is equivalent to an OR of inverted inputs. A NOR gate is equivalent to an AND of inverted inputs. The same relationship applies to gates with more inputs. Switching between these representations is easy to do on a whiteboard and is often called *bubble pushing*.

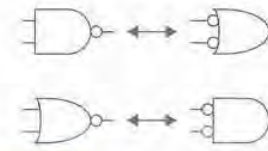


FIG 6.1 Bubble pushing with DeMorgan's law

6.2.1.2 Compound Gates As described in Section 1.4.5, static CMOS also efficiently handles compound gates computing various inverting combinations of AND/OR functions in a single stage. The function $F = AB + CD$ can be computed with an AND-OR-INVERT-22 (AOI22) gate and an inverter, as shown in Figure 6.2.

In general, logical effort of compound gates can be different for different inputs. Figure 6.4 shows how logical efforts can be estimated for the AOI21, AOI22, and a more complex compound AOI gate. The transistor widths are chosen to give the same drive as a unit inverter. The logical effort of each input is the ratio of the input capacitance of that input to the input capacitance of the inverter. For the AOI21 gate, this means the logical effort is slightly lower for the OR terminal (C) than for the two AND terminals (A, B). The parasitic delay is crudely estimated from the total diffusion capacitance on the output node by summing the sizes of the transistors attached to the output. The complex AOI will be used for comparison between logic families.

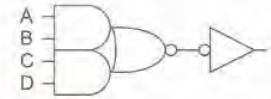


FIG 6.2 Logic using AOI22 gate

Example

Design a circuit to compute $F = AB + CD$ using NANDs and NORs.

Solution: By inspection, the circuit consists of two ANDs and an OR, shown in Figure 6.3(a). In Figure 6.3(b), the ANDs and ORs are converted to basic CMOS stages. In Figure 6.3(c and d), bubble pushing is used to simplify the logic to three NANDs.

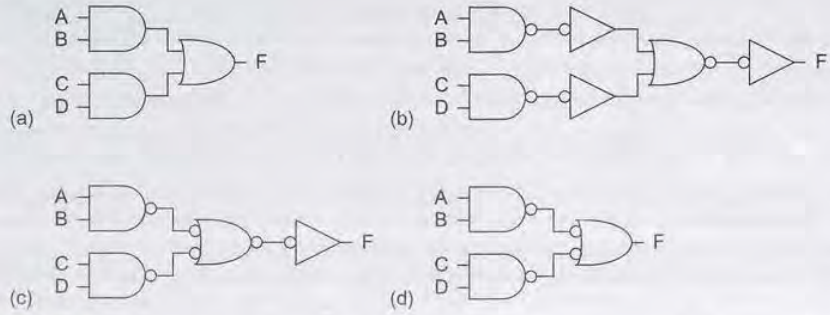


FIG 6.3 Bubble pushing to convert ANDs and ORs to NANDs and NORs

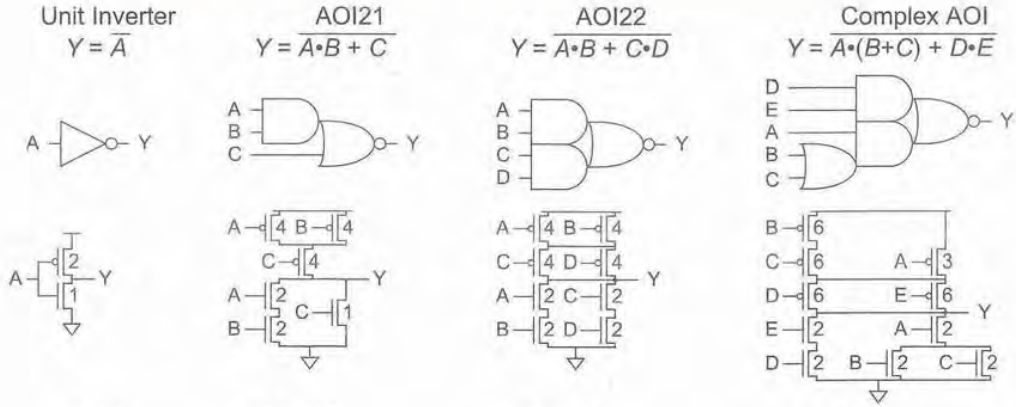


FIG 6.4 Logical efforts and parasitic delays of AOI gates

Example

Calculate the minimum delay, in τ, to compute $F = AB + CD$ using the circuits from Figure 6.2 and Figure 6.3(d). Each input can present a maximum of 20 λ of transistor width. The output must drive a load equivalent to 100 λ of transistor width. Choose transistor sizes to achieve this delay.

Solution: The path electrical effort is $H = 100/20 = 5$ and the branching effort is $B = 1$. The design using NAND gates has a path logical effort of $G = (4/3) \cdot (4/3) = 16/9$ and parasitic delay of $P = (2 + 2) = 4$. The design using the AOI22 and inverter has a path logical effort of $G = (6/3) \cdot 1 = 2$ and a parasitic delay of $P = (12/3 + 1) = 5$. Both designs have $N = 2$ stages. The path efforts $F = GBH$ are 80/9 and 10, respectively. The path delays are $NF^{1/N} + P$, or 10.0 τ and 11.3 τ, respectively. Using compound gates does not always result in faster circuits; simple 2-input NAND gates can be quite fast.

To compute the sizes, we determine the best stage efforts, $\hat{f} = F^{1/N}$, 3.0 and 3.2, respectively. These are in the range of 2.4–6 so we know the efforts are reasonable and the design would not improve too much by adding or removing stages. The input capacitance of the second gate is determined by the capacitance transformation

$$C_{\text{in}_i} = \frac{C_{\text{out}_i} \cdot g_i}{\hat{f}}.$$

For the NAND design,

$$C_{\text{in}} = \frac{100\,\lambda \cdot (4/3)}{3.0} = 44\,\lambda.$$

For the AOI22 design,

$$C_{\text{in}} = \frac{100\,\lambda \cdot (1)}{3.2} = 31\,\lambda.$$

The paths are shown in Figure 6.5 with transistor widths rounded to integer values.

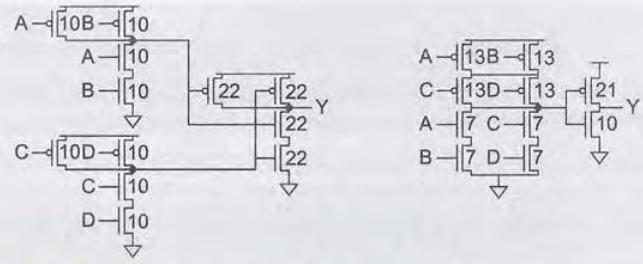


FIG 6.5 Transistor widths for example

6.2.1.3 Input Ordering Delay Effect The logical effort and parasitic delay of different gate inputs is often different. Some logic gates, like the AOI21 in the previous section, are inherently *asymmetric* in that one input sees less capacitance than another. Other gates, like NANDs and NORs, are nominally symmetric but actually have slightly different logical effort and parasitic delays for the different inputs.

Figure 6.6 shows a 2-input NAND gate annotated with diffusion parasitics. Consider the falling output transition occurring when one input held a stable '1' value and the other rises from '0' to '1.' If input B rises last, node x will initially be at $V_{DD} - V_t \approx V_{DD}$ because it was pulled up through the nMOS transistor on input A. As shown in Section 4.2.1, the Elmore delay is $(R/2)(2C) + R(6C) = 7RC = 2.33\ \tau$[1]. On the other hand, if input A rises last, node x will initially be at 0 V because it was discharged through the nMOS transistor on input B. No charge must be delivered to node x, so the Elmore delay is simply $R(6C) = 6RC = 2\ \tau$.

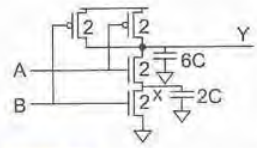


FIG 6.6 NAND gate delay estimation

In general, we define the *outer* input to be the input closer to the supply rail (e.g., B) and the *inner* input to be the input closer to the output (e.g., A). The parasitic delay is smallest when the inner input switches last because the intermediate nodes have already been discharged. Therefore, if one signal is known to arrive later than the others, the gate is fastest when that signal is connected to the inner input.

Table 5.7 listed the logical effort and parasitic delay for each input of various NAND gates, confirming that the inner input has a lower parasitic delay. The logical efforts are lower than initial estimates might predict because of velocity saturation. Interestingly, the inner input has a slightly higher logical effort because the intermediate node x tends to rise and cause negative feedback when the inner input turns ON (see Exercise 6.5) [Sutherland99]. This effect is seldom significant to the designer because the inner input remains faster over the range of fanouts used in reasonable circuits.

6.2.1.4 Asymmetric Gates When one input is far less critical than another, even nominally symmetric gates can be made asymmetric to favor the late input at the expense of the early one. For example, consider the path in Figure 6.7(a). Under ordinary conditions, the path acts as a buffer between A and Y. When reset is asserted, the path forces the output low. If reset only occurs under exceptional circumstances and can take place slowly, the circuit should be optimized for input-to-output delay at the expense of reset. This can be done with the *asymmetric* NAND gate in Figure 6.7(b). The pull-down resistance is $R/4 + R/(4/3) = R$, so the gate still offers the same driver as a unit inverter. However, the capacitance on input A is only 10/3, so the logical effort is 10/9. This is better than 4/3, which is normally associated with a NAND gate. In the limit of an infinitely large reset transistor and unit-sized nMOS transistor for input A, the logical effort approaches 1, just like an inverter. The improvement in logical effort of input A comes at the cost of much higher effort on the reset input. Note that the pMOS transistor on the reset input is also shrunk.

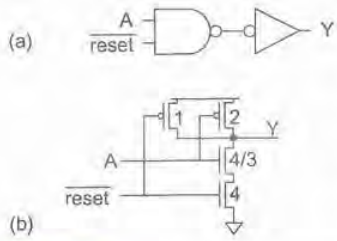


FIG 6.7 Resettable buffer optimized for data input

[1]Recall that $\tau = 3RC$ is the delay of an inverter driving the gate of an identical inverter.

This reduces its diffusion capacitance and parasitic delay at the expense of slower response to reset.

In other circuits such as arbiters, we may wish to build gates that are perfectly symmetric so neither input is favored. Figure 6.8 shows how to construct a symmetric NAND gate.

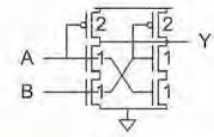


FIG 6.8 Perfectly symmetric 2-input NAND gate

6.2.1.5 Skewed Gates In other cases, one input transition is more important than the other. In Section 2.5.2 we defined *HI-skew* gates to favor the rising output transition and *LO-skew* gates to favor the falling output transition. This favoring can be done by decreasing the size of the noncritical transistor. The logical efforts for the rising (up) and falling (down) transitions are called g_u and g_d, respectively, and are the ratio of the input capacitance of the skewed gate to the input capacitance of an unskewed inverter with equal drive *for that transition*. Figure 6.9(a) shows how a HI-skew inverter is constructed by downsizing the nMOS transistor. This maintains the same effective resistance for the critical transition while reducing the input capacitance relative to the unskewed inverter of Figure 6.9(b), thus reducing the logical effort on that critical transition to $g_u = 2.5/3 = 5/6$. Of course, the improvement comes at the expense of the effort on the noncritical transition. The logical effort for the falling transition is estimated by comparing the inverter to a smaller unskewed inverter with equal pull-down current, shown in Figure 6.9(c), giving a logical effort of $g_d = 2.5/1.5 = 5/3$. The degree of skewing (e.g., the ratio of effective resistance for the fast transition relative to the slow transition) impacts the logical efforts and noise margins; a factor of two is common. Figure 6.10 catalogs HI-skew and LO-skew gates with a skew factor of two. Skewed gates are sometimes denoted with an *H* or an *L* on their symbol in a schematic.

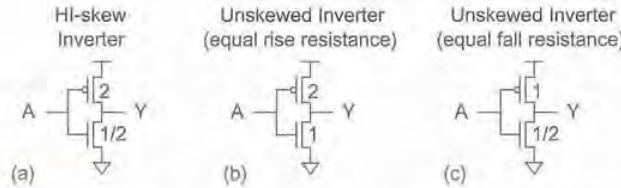


FIG 6.9 Logical effort calculation for HI-skew inverter

Alternating HI-skew and LO-skew gates can be used when only one transition is important [Solomatnikov00]. Skewed gates work particularly well with dynamic circuits, as we shall see in Section 6.2.4.

6.2.1.6 P/N Ratios Notice in Figure 6.10 that the average logical effort of the LO-skew NOR2 is actually better than that of the unskewed gate. The pMOS transistors in the unskewed gate are enormous in order to provide equal rise delay. They contribute input capacitance for both transitions, while only helping the rising delay. By accepting a slower

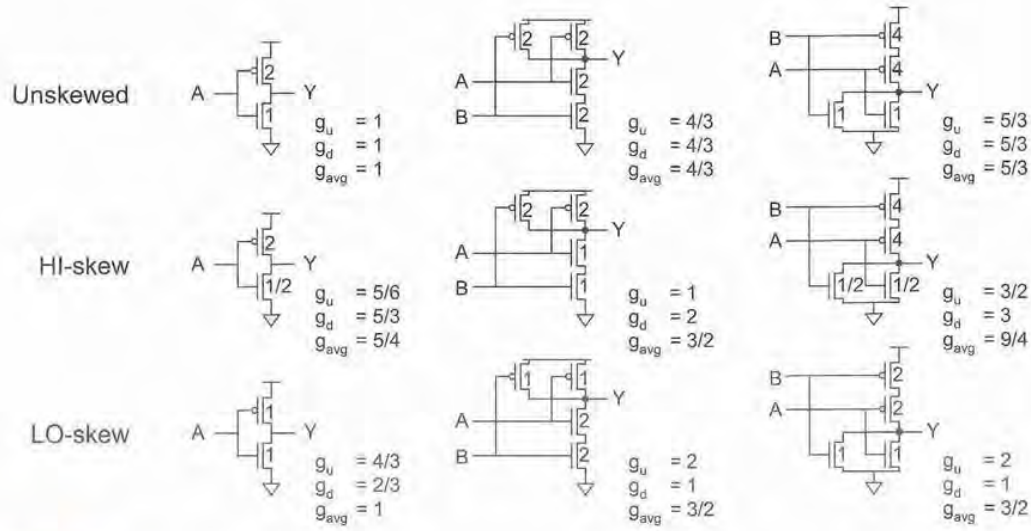


FIG 6.10 Catalog of skewed gates

rise delay, the pMOS transistors can be downsized to significantly reduce input capacitance and average delay.

In general, what is the best *P/N* ratio for logic gates (i.e., the ratio of pMOS to nMOS transistor width)? You will see in Exercise 6.13 that the ratio giving lowest average delay is the square root of the ratio that gives equal rise and fall delays. For processes with a mobility ratio of $\mu_n/\mu_p = 2$ as we have generally been assuming, the best ratios are shown in Figure 6.11.

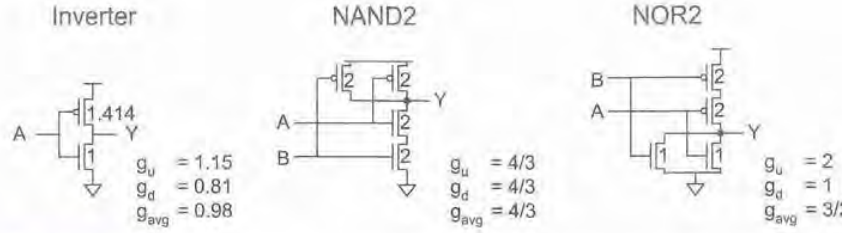


FIG 6.11 Gates with P/N ratios giving least delay

Reducing the pMOS size from 2 to $\sqrt{2} \approx 1.4$ for the inverter gives the theoretical fastest average delay, but this delay improvement is only 2%. However, this significantly reduces the pMOS transistor area. It also reduces input capacitance, which in turn reduces power consumption. Unfortunately, it leads to unequal delay between the outputs. Some paths can be slower than average if they trigger the worst edge of each gate. Excessively slow rising outputs can also cause hot electron degradation. And reducing the pMOS size also moves the switching point lower and reduces the noise margin.

In summary, the *P/N* ratio of a library of cells should be chosen on the basis of area, power, and reliability, not average delay. For NOR gates, reducing the size of the pMOS transistors significantly improves both delay and area. In most standard cell libraries, the pitch of the cell determines the *P/N* ratio that can be achieved in any particular gate. Ratios of 1.5–2 are commonly used for inverters.

6.2.1.7 Multiple Threshold Voltages Some CMOS processes offer two or more threshold voltages. Transistors with lower threshold voltages produce more ON current, but also leak exponentially more OFF current. Libraries can provide both high- and low-threshold versions of gates. The low-threshold gates can be used sparingly to reduce the delay of critical paths [Kumar94, Wei98]. Skewed gates can use low-threshold devices on only the critical network of transistors.

6.2.2 Ratioed Circuits

Ratioed circuits, introduced in Section 2.5.4, use weak pull-up devices and stronger pull-down devices. They reduce the input capacitance and hence improve logical effort by eliminating large pMOS transistors loading the inputs, but depend on the correct ratio of pull-up to pull-down strength. If the pull-up is too strong, $V_{OL\max}$ may be too high; $V_{OL\max}$ is best chosen to be less than V_{tn} so the low output does not turn ON the next stage. If the pull-up is too weak, the rising delay will be too slow. Ratioed circuits also dissipate static power while the output is low, so they must be used in a limited fashion where they provide significant benefits.

6.2.2.1 Pseudo-nMOS Figure 6.12 shows *pseudo-nMOS* logic gates, which are the most common form of CMOS ratioed logic. The pull-down network is like that of a static gate, but the pull-up network has been replaced with a single pMOS transistor that is grounded so it is always ON. The pMOS transistor width is selected to be about 1/4 the strength (i.e., 1/2 the effective width) of the nMOS pull-down network as a compromise between noise margin and speed; this best size is highly process-dependent, but is usually in the range of 1/3 to 1/6.

To calculate the logical effort of pseudo-nMOS gates, suppose a complementary CMOS unit inverter delivers current *I* in both rising and falling transitions. For the widths shown, the pMOS transistors produce *I*/3 and the nMOS networks produce 4*I*/3. The logical effort for each transition is computed as the ratio of the input capacitance to that of a complementary CMOS inverter with equal current for that transition. For the

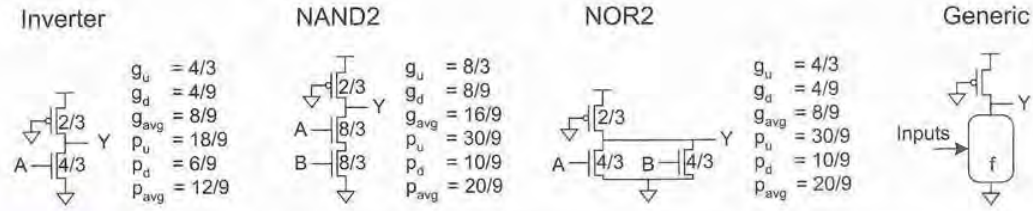


FIG 6.12 Pseudo-nMOS logic gates

falling transition, the pMOS transistor effectively fights the nMOS pull-down. The output current is estimated as the pull-down current minus the pull-up current, $(4I/3 - I/3) = I$. Therefore, we will compare each gate to a unit inverter to calculate g_u. For example, the logical effort for a falling transition of the pseudo-nMOS inverter is the ratio of its input capacitance (4/3) to that of a unit complementary CMOS inverter (3), i.e., 4/9. g_u is three times as great because the current is 1/3 as much.

The parasitic delay is also found by counting output capacitance and comparing it to an inverter with equal current. For example, the pseudo-nMOS NOR has 10/3 units of diffusion capacitance as compared to 3 for a unit-sized complementary CMOS inverter, so its parasitic delay pulling down is 10/9. The pull-up current is 1/3 as great, so the parasitic delay pulling up is 10/3.

As can be seen, pseudo-nMOS is slower on average than static CMOS for NAND structures. However, it works well for NOR structures. The logical effort is independent of the number of inputs in wide NORs, so pseudo-nMOS is useful for fast wide NOR gates or NOR-based structures like ROMs and PLAs when power permits.

Pseudo-nMOS gates will not operate correctly if $V_{OL} > V_{IL}$ of the receiving gate. This is most likely in the SF design corner where nMOS transistors are weak and pMOS transistors are strong. Designing for acceptable noise margin in the SF corner forces conservative choice of weak pMOS transistors in the normal corner. A biasing circuit can be used to reduce process sensitivity, as shown in Figure 6.15. The goal of the biasing circuit is to create a V_{bias} that causes *P2* to deliver 1/3 the current of *N2*, independent of the relative mobilities of the pMOS and nMOS transistors. Transistor *N2* has width of 3/2 and hence produces current $3I/2$ when ON. Transistor *N1* is tied ON to act as a current source with 1/3 the current of *N2*, i.e., $I/2$. *P1* acts as a current mirror using feedback to establish the bias voltage sufficient to provide equal current as *N1*, $I/2$. The size of *P1* is noncritical so long as it is large enough to produce sufficient current and is equal in size to *P2*. Now, *P2* ideally also provides $I/2$. In summary, when *A* is low, the pseudo-nMOS gate pulls up with a current of $I/2$. When *A* is high, the pseudo-nMOS gate pulls down with an effective current of $(3I/2 - I/2) = I$. To first order, this biasing technique sets the relative currents strictly by transistor widths, independent of relative pMOS and nMOS mobilities.

Example

Design a k-input AND gate with DeMorgan's Law using static CMOS inverters followed by a k-input pseudo-nMOS NOR, as shown in Figure 6.13. Let each inverter be unit-sized. If the output load is an inverter of size H, determine the best transistor sizes in the NOR gate and estimate the average delay of the path.

Solution: The path electrical effort is H and the branching effort is $B = 1$. The inverter has a logical effort of 1. The pseudo-nMOS NOR has an average logical effort of 8/9 according to Figure 6.12. The path logical effort is $G = 1 \cdot (8/9) = 8/9$, so the path effort is $8H/9$. Each stage should bear an effort of $\hat{f} = \sqrt{8H/9}$. Using the capacitance transformation gives NOR pull-down transistor widths of

$$C_{in} = \frac{gC_{out}}{\hat{f}} = \frac{(8/9)H}{\sqrt{8H/9}} = \frac{\sqrt{8H}}{3}$$

unit-sized inverters. As a unit inverter has three units of input capacitance, the NOR transistor nMOS widths should be $\sqrt{8H}$. According to Figure 6.12, the pull-up transistor should be half this width. The complete circuit marked with nMOS and pMOS widths is drawn in Figure 6.14.

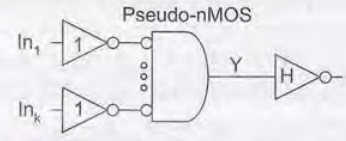


FIG 6.13 k-input AND gate driving load of H

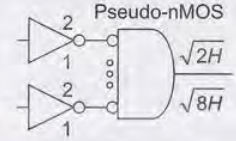


FIG 6.14 k-input AND marked with transistor widths

We estimate the average parasitic delay of a k-input pseudo-nMOS NOR to be $(8k + 4)/9$. The total delay in τ is

$$D = N\hat{f} + P = \frac{4\sqrt{2}}{3}\sqrt{H} + \frac{8k+13}{9}.$$

Increasing the number of inputs only impacts the parasitic delay, not the effort delay.

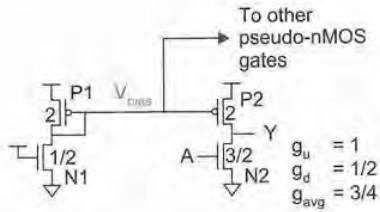


FIG 6.15 Replica biasing of pseudo-nMOS gates

Such replica biasing permits the 1/3 current ratio rather than the conservative 1/4 ratio in the previous circuits, resulting in lower logical effort. The bias voltage V_{bias} can be distributed to multiple pseudo-nMOS gates. Ideally, V_{bias} will adjust itself to keep V_{OL} constant across process corners. Unfortunately, the currents through the two pMOS transistors do not exactly match because their drain voltages are unequal, so this technique still has some process sensitivity. Also note that this bias is relative to V_{DD}, so any noise on either the bias voltage line or the V_{DD} supply rail will impact circuit performance.

Turning off the pMOS transistor can reduce power when the logic is idle or during IDDQ test mode (see Section 9.6.8), as shown in Figure 6.16.

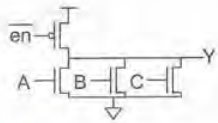


FIG 6.16 Pseudo-nMOS gate with enabled pull-up

An alternate way to represent the pseudo-nMOS gate is to draw the pull-up at the input and the pull-down transistors with open drains, as shown in Figure 6.17. Multiple gates are tied together in a "wired-OR" fashion. This representation is called *CMOS Multidrain Logic* by the inventors [Wu87], but offers no advantages over normal pseudo-nMOS circuits in modern CAD environments.

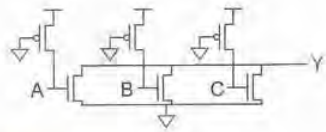


FIG 6.17 3-input NOR drawn as multidrain logic

6.2.2.2 Ganged CMOS Figure 6.18 illustrates pairs of CMOS inverters ganged together. The truth table is given in Table 6.1, showing that the pair compute the NOR function. Such a circuit is sometimes called a *symmetric*[2] *NOR* [Johnson88], or more generally, *ganged CMOS* [Schultz90]. When one input is '0' and the other '1,' the gate can be viewed as a pseudo-nMOS circuit with appropriate ratio constraints. When both inputs are '0,' both pMOS transistors turn on in parallel, pulling the output high faster than they would in an ordinary pseudo-nMOS gate. Moreover, when both inputs are '1,' both pMOS transistors turn OFF, saving static power dissipation. As in pseudo-nMOS, the transistors are sized so the pMOS are about 1/4 the strength of the nMOS and the pull-down current matches that of a unit inverter. Hence,

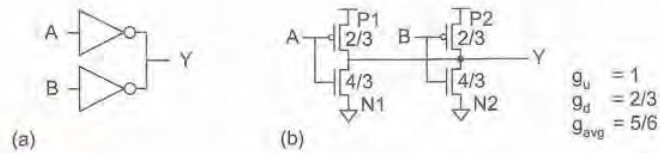


FIG 6.18 Symmetric 2-input NOR gate

[2]Do not confuse this use of *symmetric* with the concept of *symmetric* and *asymmetric* gates from Section 6.2.1.4.

Table 6.1	Operation of symmetric NOR					
A	*B*	*N1*	*P1*	*N2*	*P2*	*Y*
0	0	OFF	ON	OFF	ON	1
0	1	OFF	ON	ON	OFF	~ 0
1	0	ON	OFF	OFF	ON	~ 0
1	1	ON	OFF	ON	OFF	0

the symmetric NOR achieves both better performance and lower power dissipation than a 2-input pseudo-nMOS NOR.

Johnson also showed that symmetric structures can be used for NOR gates with more inputs and even for NAND gates (see Exercises 6.23–6.24). The 3-input symmetric NOR also works well, but the logical efforts of the other structures are unattractive.

6.2.2.3 Source Follower Pull-up Logic Figure 6.19 shows a *Source Follower Pull-up Logic* (SFPL) 4-input NOR gate [Simon92]. It is similar to a pseudo-nMOS gate except that the pull-up is controlled by the inputs. *N6–N9* and *P1* form a pseudo-nMOS NOR function. The gate of the pull-up *P1* is driven by a parallel *source follower* consisting of drive transistors *N1–N4* and load transistor N_{load}. When one input turns on, the source follower pulls node *x* to approximately $V_{DD}/2$. This tends to partially turn off *P1*, which allows smaller nMOS pulldowns *N6–N9* to be used. However, *N1–N4* also load the input, so the overall reduction in input capacitance is not clear. SFPL is primarily applicable to constructing wide NOR gates.

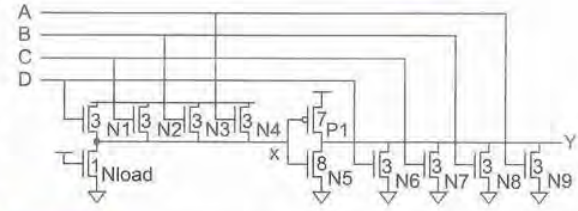


FIG 6.19 SFPL 4-input NOR gate

6.2.3 Cascode Voltage Switch Logic

Cascode Voltage Switch Logic (CVSL[3]) [Heller84] seeks the performance of ratioed circuits without the static power consumption. It uses both true and complementary input signals and computes both true and complementary outputs using a pair of nMOS pull-down networks, as shown in Figure 6.20(a). The pull-down network *f* implements the logic function as in a static CMOS gate, while $\bar{f}$ uses inverted inputs feeding transistors arranged in the conduction complement. For any given input pattern, one of the pull-down networks will be ON and the other OFF. The pull-down network that is ON will pull that output low. This low output turns ON the pMOS transistor to pull the opposite output high. When the opposite output rises, the other pMOS transistor turns OFF so no static power dissipation occurs. Figure 6.20(b) shows a CVSL AND/NAND gate. Observe how the

[3] Many authors call this circuit family *Differential Cascode Voltage Switch Logic* (DCVS [Chu86] or DCVSL [Ng96]). The term *cascode* comes from analog circuits where transistors are placed in series.

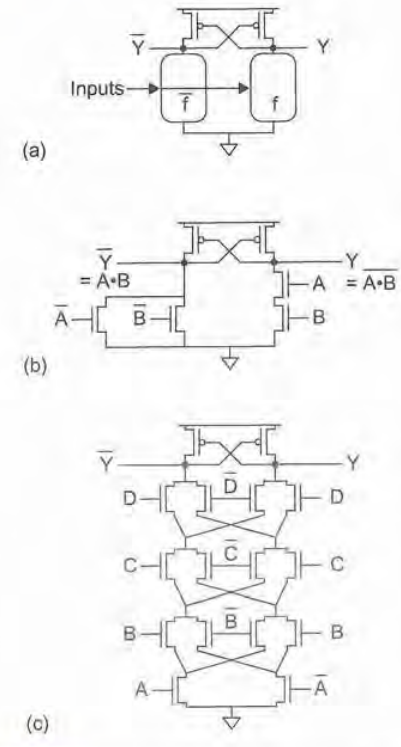


FIG 6.20 CVSL gates

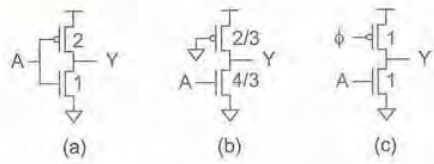


FIG 6.21 Comparison of (a) static CMOS, (b) pseudo-nMOS, and (c) dynamic inverters

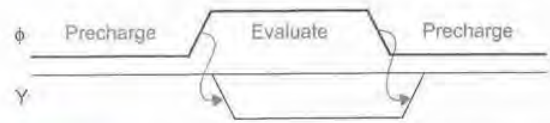


FIG 6.22 Precharge and evaluation of dynamic gates

pull-down networks are complementary, with parallel transistors in one and series in the other. Figure 6.20(c) shows a 4-input XOR gate. The pull-down networks share A and $\overline{A}$ transistors to reduce the transistor count by two. Sharing is often possible in complex functions, and systematic methods exist to design shared networks [Chu86].

CVSL has a potential speed advantage because all of the logic is performed with nMOS transistors, thus reducing the input capacitance. As in pseudo-nMOS, the size of the pMOS transistor is important. It fights the pull-down network, so a large pMOS transistor will slow the falling transition. Unlike pseudo-nMOS, the feedback tends to turn off the pMOS, so the outputs will eventually settle to a legal logic level. A small pMOS transistor is slow at pulling the complementary output high. In addition, the CVSL gate requires both the low- and high-going transitions, adding more delay. Contention current during the switching period also increases power consumption.

Pseudo-nMOS worked well for wide NOR structures. Unfortunately, CVSL also requires the complement, a slow tall NAND structure. Therefore, CVSL is poorly suited to general NAND and NOR logic. Even for symmetric structures like XORs, it tends to be slower than static CMOS, as well as more power-hungry [Chu87, Ng96]. However, the ideas behind CVSL help us understand dual-rail domino and complementary pass-transistor logic discussed in later sections.

6.2.4 Dynamic Circuits

Ratioed circuits reduce the input capacitance by replacing the pMOS transistors connected to the inputs with a single resistive pull-up. The drawbacks of ratioed circuits include slow rising transitions, contention on the falling transitions, static power dissipation, and a non-zero V_{OL}. Dynamic circuits circumvent these drawbacks by using a clocked pull-up transistor rather than a pMOS that is always ON. Figure 6.21 compares (a) static CMOS, (b) pseudo-nMOS, and (c) dynamic inverters. Dynamic circuit operation is divided into two modes, shown in Figure 6.22. During *precharge*, the clock ϕ is '0,' so the clocked pMOS is ON and initializes the output Y high. During *evaluation*, the clock is '1' and the clocked pMOS turns OFF. The output may remain high or may be discharged low through the pull-down network. Dynamic circuits are the fastest commonly used circuit family because they have lower input capacitance and no contention during switching. They also have zero static power dissipation.

However, they require careful clocking, consume significant dynamic power, and are sensitive to noise during evaluation. Clocking of dynamic circuits will be discussed in much more detail in Section 7.5.

In Figure 6.21(c), if the input *A* is '1' during precharge, contention will take place because both the pMOS and nMOS transistors will be ON. When the input cannot be guaranteed to be '0' during precharge, an extra clocked evaluation transistor can be added to the bottom of the nMOS stack to avoid contention as shown in Figure 6.23. The extra transistor is sometimes called a *foot*. Figure 6.24 shows generic *footed* and *unfooted* gates[4].

Figure 6.25 estimates the falling logical effort of both footed and unfooted dynamic gates. As usual, the pull-down transistors' widths are chosen to give unit resistance. Precharge occurs while the gate is idle and often may take place more slowly. Therefore, the precharge transistor width is chosen for twice unit resistance. This reduces the capacitive load on the clock and the parasitic capacitance at the expense of greater rising delays. We see that the logical efforts are very low. Footed gates have higher logical effort than their unfooted counterparts but are still an improvement over static logic. In practice, the logical effort of footed gates is better than predicted because velocity saturation means series nMOS transistors have less resistance than we have estimated. Moreover, logical efforts are also slightly better than predicted because there is

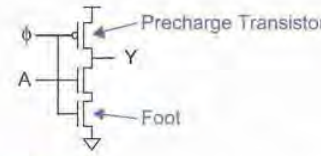


FIG 6.23 Footed dynamic inverter

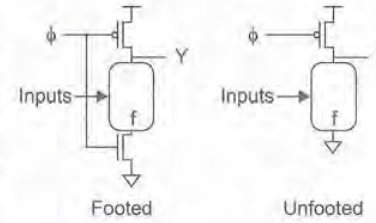


FIG 6.24 Generalized footed and unfooted dynamic gates

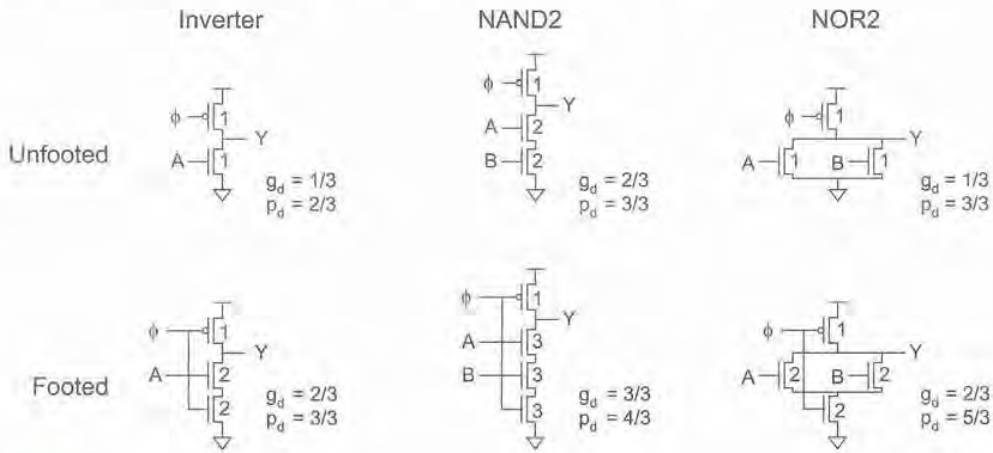


FIG 6.25 Catalog of dynamic gates

[4]The footed and unfooted terminology is from IBM [Nowka98]. Intel calls these styles D1 and D2, respectively.

no contention between nMOS and pMOS transistors during the input transition. The size of the foot can be increased relative to the other nMOS transistors to reduce logical effort of the other inputs at the expense of greater clock loading. Like pseudo-nMOS gates, dynamic gates are particularly well suited to wide NOR functions or multiplexers because the logical effort is independent of the number of inputs. Of course, the parasitic delay does increase with the number of inputs because there is more diffusion capacitance on the output node. Characterizing the logical effort and parasitic delay of dynamic gates is tricky because the output tends to fall much faster than the input rises, leading to potentially misleading dependence of propagation delay on fanout [Sutherland99].

A fundamental difficulty with dynamic circuits is the *monotonicity* requirement. While a dynamic gate is in evaluation, the inputs must be *monotonically rising*. That is, the input can start LOW and remain LOW, start LOW and rise HIGH, start HIGH and remain HIGH, but not start HIGH and fall LOW. Figure 6.26 shows waveforms for a footed dynamic inverter in which the input violates monotonicity. During precharge, the output is pulled HIGH. When the clock rises, the input is HIGH so the output is discharged LOW through the pull-down network, as you would want to have happen in an inverter. The input later falls LOW, turning off the pull-down network. However, the precharge transistor is also OFF so the output floats, staying LOW rather than rising as it would in a normal inverter. The output will remain low until the next precharge step. In summary, the inputs must be monotonically rising for the dynamic gate to compute the correct function.

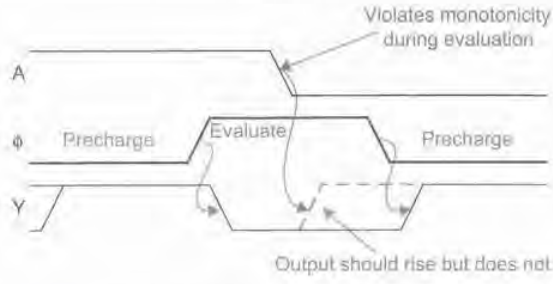


FIG 6.26 Monotonicity problem

Unfortunately, the output of a dynamic gate begins HIGH and monotonically falls LOW during evaluation. This monotonically falling output X is not a suitable input to a second dynamic gate expecting monotonically rising signals, as shown in Figure 6.27. Dynamic gates sharing the same clock cannot be directly connected. This problem is often overcome with domino logic, described in the next section.

6.2.4.1 Domino Logic The monotonicity problem can be solved by placing a static CMOS inverter between dynamic gates, as shown in Figure 6.28(a). This converts the monotonically falling output into a monotonically rising signal suitable for the next gate, as shown in Figure 6.28(b). The dynamic-static pair together is called a *domino* gate [Krambeck82] because precharge resembles setting up a chain of dominos and evaluation causes the gates to fire like dominos tipping over, each triggering the next. A single clock can be used to precharge and evaluate all the logic gates within the chain. The dynamic output is monotonically falling during evaluation, so the static inverter output is monotonically rising. Therefore, the static inverter is usually a HI-skew gate to favor this rising output. Observe that precharge occurs in parallel, but evaluation occurs sequentially. This explains why precharge is usually less critical. The symbols for the dynamic NAND, HI-skew inverter, and domino AND are shown in Figure 6.28(c).

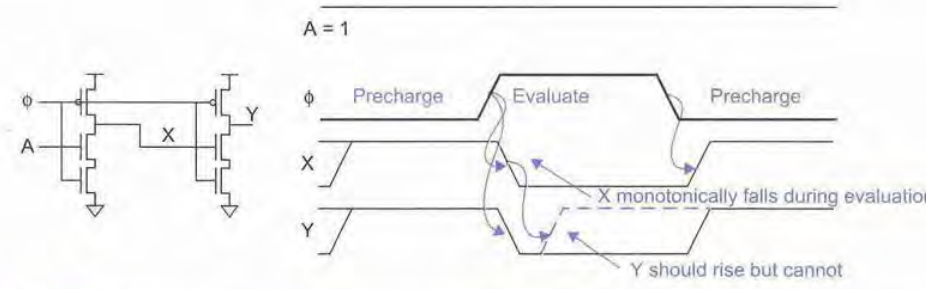


FIG 6.27 Incorrect connection of dynamic gates

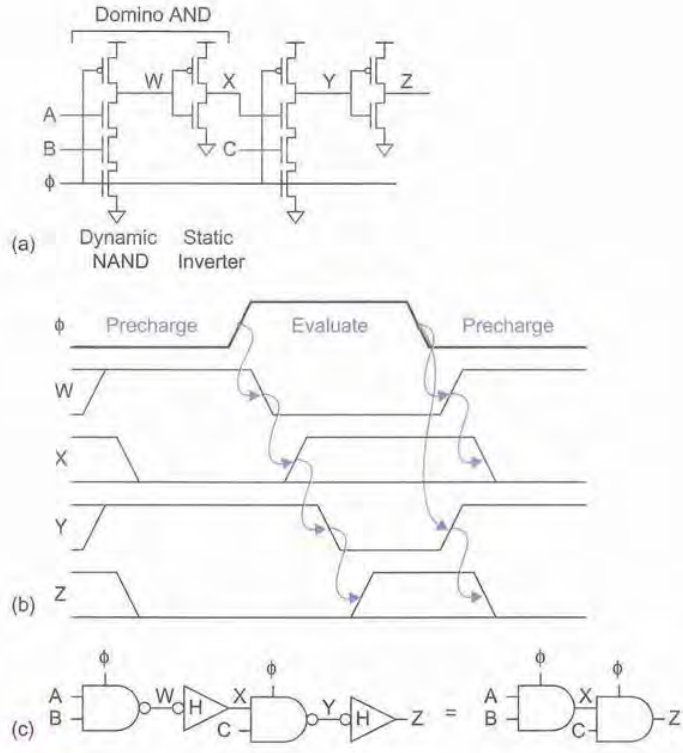


FIG 6.28 Domino gates

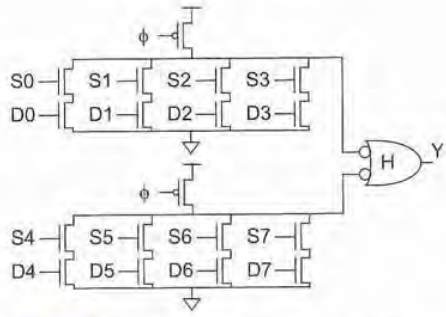


FIG 6.29 Domino gate using logic in static CMOS stage

In general, more complex inverting static CMOS gates such as NANDs or NORs can be used in place of the inverter [Sutherland99]. For example, Figure 6.29 shows an 8-input domino multiplexer built from two 4-input dynamic multiplexers and a HI-skew NAND gate. This is often faster than an 8-input dynamic mux and HI-skew inverter because the dynamic stage has less diffusion capacitance and parasitic delay.

Domino gates are inherently noninverting, while some functions like XOR gates necessarily require inversion. Three methods of addressing this problem include pushing inversions into static logic, delaying clocks, and using dual-rail domino logic. In many circuits such as arithmetic logic units (ALUs), the necessary XOR gate at the end of the path can be built with a conventional static CMOS XOR gate driven by the last domino circuit. However, the XOR output no longer is monotonically rising and thus cannot directly drive more domino logic. A second approach is to directly cascade dynamic gates without the static CMOS inverter, delaying the clock to the later gates to ensure the inputs are monotonic during evaluation. This is commonly done in content-addressable memories (CAMs) and NOR-NOR PLAs and will be discussed in Sections 7.5.4 and 11.7. The third approach, dual-rail domino logic, is discussed in the next section.



6.2.4.2 Dual-rail Domino Logic *Dual-rail domino* gates encode each signal with a pair of wires. The input and output signal pairs are denoted with *_h* and *_l*, respectively. Table 6.2 summarizes the encoding. The *_h* wire is asserted to indicate that the output of the gate is "high" or '1.' The *_l* wire is asserted to indicate that the output of the gate is "low" or '0.' When the gate is precharged, neither *_h* nor *_l* is asserted. The pair of lines should never be both asserted simultaneously during correct operation.

Table 6.2 Dual-rail domino signal encoding

sig_h	*sig_l*	Meaning
0	0	precharged
0	1	'0'
1	0	'1'
1	1	invalid

Dual-rail domino gates accept both true and complementary inputs and compute both true and complementary outputs, as shown in Figure 6.30(a). Observe that this is identical to static CVSL circuits from Figure 6.20 except that the cross-coupled pMOS transistors are instead connected to the precharge clock. Therefore, dual-rail domino can be viewed as a dynamic form of CVSL, sometimes called DCVS [Heller84]. Figure

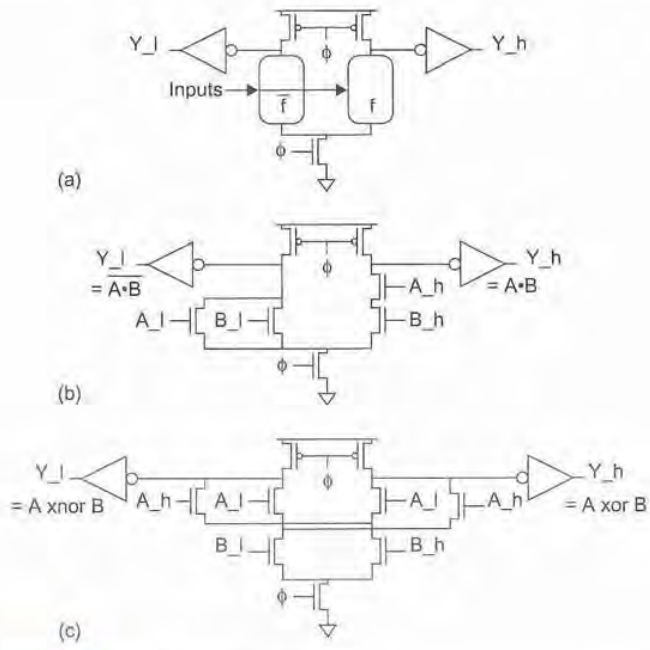


FIG 6.30 Dual-rail domino gates

6.30(b) shows a dual-rail AND/NAND gate and Figure 6.30(c) shows a dual-rail XOR/XNOR gate. The gates are shown with clocked evaluation transistors, but can also be unfooted. Dual-rail domino is a complete logic family in that it can compute all inverting and noninverting logic functions. However, it requires more area, wiring, and power. Dual-rail structures also lose the efficiency of wide dynamic NOR gates because they require complementary tall dynamic NAND stacks.

Dual-rail domino signals not only the result of a computation but also indicates when the computation is done. Before computation completes, both rails are precharged. When the computation completes, one rail will be asserted. A NAND gate can be used for completion detection as shown in Figure 6.31. This is particularly useful for asynchronous circuits [Williams91, Sparsø01].

Coupling can be reduced in dual-rail signal busses by interdigitating the bits of the bus, as shown in Figure 6.32. Each wire will never see more than one aggressor switching at a time because only one of the two rails switches in each cycle.

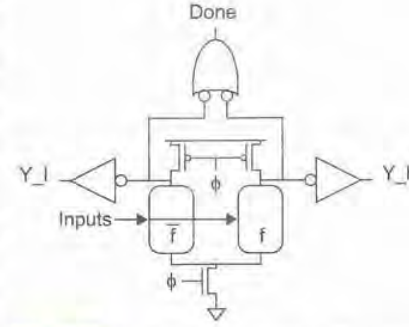


FIG 6.31 Dual-rail domino gate with completion detection



6.2.4.3 Keepers Dynamic circuits also suffer from charge leakage on the dynamic node. If a dynamic node is precharged high and then left floating, the voltage on the dynamic node will drift over time due to subthreshold, gate, and junction leakage. The time constants tend to be in the millisecond to nanosecond range, depending on process and temperature. This problem is analogous to leakage in dynamic RAMs. Moreover, dynamic circuits also have poor input noise margins. If the input rises above V_t while the gate is in evaluation, the input transistors will turn on weakly and can incorrectly discharge the output. Both leakage and noise margin problems can be addressed by adding a *keeper* circuit.

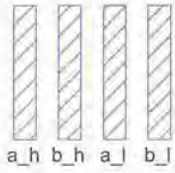


FIG 6.32 Reducing coupling noise on dual-rail busses

Figure 6.33 shows a conventional keeper on a domino buffer. The keeper is a weak transistor that holds, or *staticizes*, the output at the correct level when it would otherwise float. When the dynamic node X is high, the output Y is low and the keeper is ON to prevent X from floating. When X falls, the keeper initially opposes the transition so it must be much weaker than the pull-down network. Eventually Y rises, turning the keeper OFF and avoiding static power dissipation.

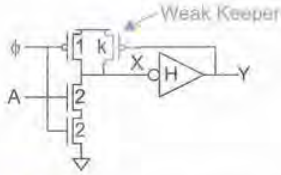


FIG 6.33 Conventional keeper

The keeper must be strong (i.e., wide) enough to compensate for any leakage current drawn when the output is floating and the pull-down stack is OFF. Strong keepers also improve the noise margin because when the inputs are slightly above V_t the keeper can supply enough current to hold the output high. Figure 5.26 showed the DC transfer characteristics of a dynamic inverter. As the keeper width k increases, the switching point shifts right. However, strong keepers also increase delay, typically by 5%–10%. Keeper transistors are usually on the order of 1/10 the strength of the pull-down stack, although they may need to be stronger on wide NOR gates or multiplexers in particularly leaky processes. For small dynamic gates, this ratio implies that the keeper must be weaker than a minimum-sized transistor. This is achieved by increasing the keeper length, as shown in Figure 6.34(a). Long keeper transistors increase the capacitive load on the output Y. This can be avoided by splitting the keeper, as shown in Figure 6.34(b).

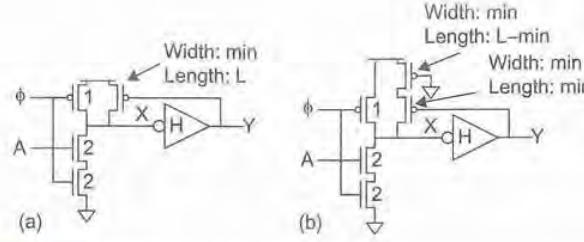


FIG 6.34 Weak keeper implementations

Figure 6.35 shows a *differential keeper* for a dual-rail domino buffer. When the gate is precharged, both keeper transistors are OFF and the dynamic outputs float. However, as soon as one of the rails evaluates low, the opposite keeper turns ON. The differential keeper is fast because it does not oppose the falling rail. As long as one of the rails is guaranteed to fall promptly, the keeper on the other rail will turn on before excessive leakage or noise causes failure. Of course, dual-rail domino can also use a pair of conventional keepers.

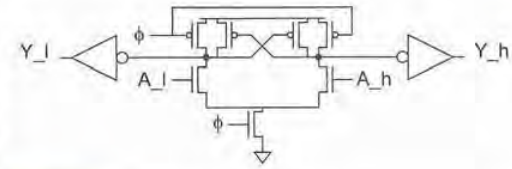


FIG 6.35 Differential keeper

An alternative approach is to build the keeper with a small complementary pull-up stack connected to the inputs rather than a weak pMOS transistor driven by the inverted output, as shown in Figure 6.36 for a 2-input NAND gate. In the terminology of Section 6.2.1.5, such a gate can be viewed as a very LO-skew (e.g., skew factor 5) static CMOS gate augmented with precharge and (possibly) evaluation transistors. Skew factors of 4-8 are reasonable. The approach has been called *noise tolerant precharge* (NTP) [Yamada95, Murabayashi96] or *monotonic static CMOS* [Thorp99]. Such noise-tolerant precharge gates have the advantage that the keeper will eventually recover from a noise event, in contrast to standard keepers that never recover if the input glitches enough to turn them off. Noise-tolerant precharge does not work well for wide NOR structures because it requires many series pMOS transistors. It is also most practical for gates with wide input transistors where it is feasible to construct a comparatively weak complementary network with minimum-sized pMOS transistors. Figure 6.37 plots the static noise margin and delay of dynamic NAND2 gates using conventional keepers and

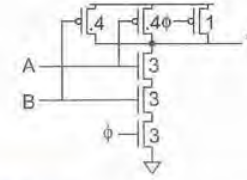


FIG 6.36 Noise-tolerant precharge

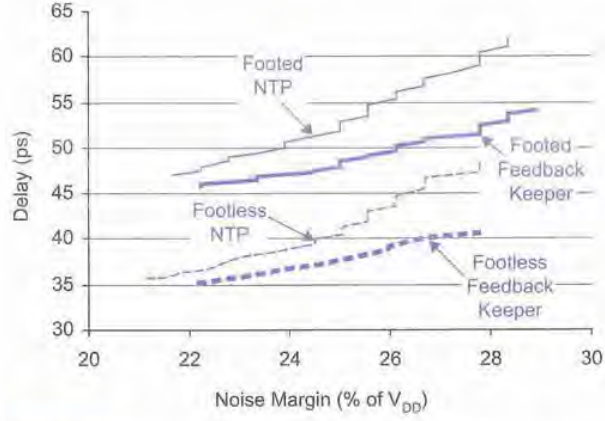


FIG 6.37 Delay vs. noise margin of conventional and noise-tolerant precharge dynamic NAND2s

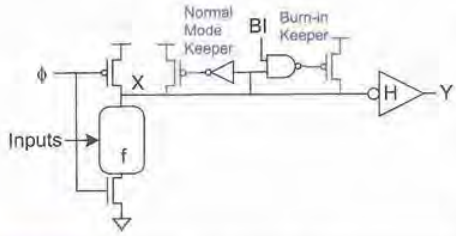


FIG 6.38 Burn-in conditional keeper

NTP [Harris03]. Increasing the keeper size improves noise margin at the expense of delay. For the same noise margin, the conventional keepers are consistently faster.

During *burn-in*, the chip operates at reduced frequency, but at very high temperature and voltage. This causes severe leakage that can overpower the keeper in wide dynamic NOR gates where many nMOS transistors leak in parallel. Figure 6.38 shows a domino gate with a *burn-in conditional keeper* [Alvandpour02]. The *BI* signal is asserted during burn-in to turn on a second keeper in parallel with the primary keeper. The second keeper slows the gate during burn-in, but provides extra current to fight leakage.

Domino circuits with delayed clocks can use full keepers consisting of cross-coupled inverters to hold the output either high or low, as discussed in Section 7.5.2. Many other keeper techniques have been proposed including a "leaker" that is always ON [Krambeck82], input-controlled refresh [Lakshmanan01], the inverter technique [Covino97], pMOS pull-up technique [D'Souza96], mirror technique [Wang001], and twin-transistor technique [Balamurugan01], but the conventional keeper is generally satisfactory because it is simple and has little cost in speed and power dissipation.

OPTIONAL

6.2.4.4 Secondary Precharge Devices Dynamic gates are subject to problems with *charge sharing* [Oklobdžija86]. For example, consider the 2-input dynamic NAND gate in Figure 6.39(a). Suppose the output Y is precharged to V_{DD} and inputs A and B are low. Also suppose that the intermediate node x had a low value from a previous cycle. During evaluation, input A rises, but input B remains low so the output Y should remain high. However, charge is shared between C_x and C_Y, shown in Figure 6.39(b). This behaves as a capacitive voltage divider and the voltages equalize at

$$V_x = V_Y = \frac{C_Y}{C_x + C_Y} V_{DD} \tag{6.3}$$

Charge sharing is most serious when the output is lightly loaded (small C_Y) and the internal capacitance is large. For example, 4-input dynamic NAND gates and complex AOI gates can share charge among multiple nodes. If the charge-sharing noise is small, the keeper will eventually restore the dynamic output to V_{DD}. However, if the charge-sharing noise is large, the output may flip and turn off the keeper, leading to incorrect results.

Charge sharing can be overcome by precharging some or all of the internal nodes with *secondary precharge transistors*, as shown in Figure 6.40. These transistors should be small because they only must charge the small internal capacitances and their diffusion capacitance slows the evaluation. It is often sufficient to precharge every other node in a tall stack. SOI processes are less susceptible to charge sharing in dynamic gates because the diffusion capacitance of the internal nodes is smaller. If some charge sharing is acceptable, a gate can be made faster by predischarging some internal nodes [Ye00].

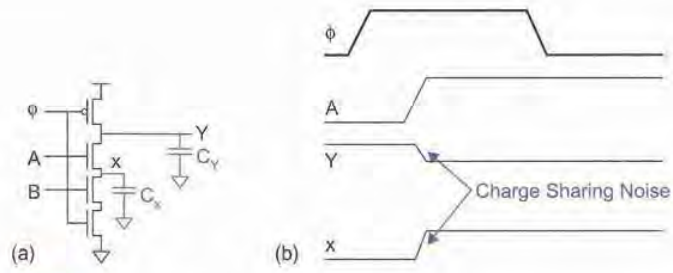


FIG 6.39 Charge-sharing noise

In summary, domino logic was originally proposed as a fast and compact circuit technique. In practice, domino is prized for its speed. However, by the time feet, keepers, and secondary precharge devices are added for robustness, domino is seldom much more compact than static CMOS and it demands a tremendous design effort to ensure robust circuits. When dual-rail domino is required, the area exceeds static CMOS.

6.2.4.5 Logical Effort of Dynamic Paths In Section 4.3.3, we found the best stage effort by hypothetically appending static CMOS inverters onto the end of the path. The best effort depended on the parasitic delay and was 3.59 for $p_{inv} = 1$. When we employ alternative circuit families, the best stage effort may change. For example, with domino circuits, we may consider appending domino buffers onto the end of the path. Figure 6.41 shows that the logical effort of a domino buffer is $G = 5/9$ for footed domino and 5/18 for unfooted domino. Therefore, each buffer appended to a path actually decreases the path effort. Hence, it is better to add more buffers, or equivalently, to target a lower stage effort than you would in a static CMOS design.

OPTIONAL

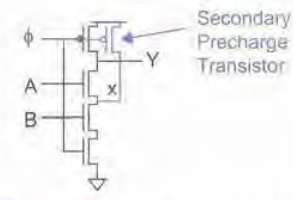


FIG 6.40 Secondary precharge transistor

[Sutherland99] showed that the best stage effort is $p = 2.76$ for paths with footed domino and 2.0 for paths with unfooted domino. In paths mixing footed and unfooted domino, the best effort is somewhere between these extremes. As a rule of thumb, just as you target a stage effort of 4 for static CMOS paths, you can target a stage effort of 2–3 for domino paths.

We have also seen that it is possible to push logic into the static CMOS stages between dynamic gates. The following example explores under what circumstances this is beneficial.

In summary, dynamic stages are fast because they build logic using nMOS transistors. Moreover, the low logical efforts suggest that using a relatively large number of stages is beneficial. Pushing logic into the static CMOS stages uses slower

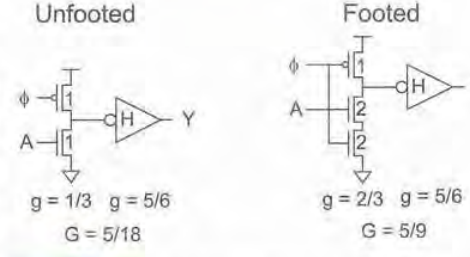


FIG 6.41 Logical efforts of domino buffers

Example

Figure 6.42 shows two designs for an 8-input domino AND gate using footed dynamic gates. One uses four stages of logic with static CMOS inverters. The other uses only two stages by employing a HI-skew NOR gate. For what range of path electrical efforts is the 2-stage design faster?

Solution: You might expect that the second design is superior because it scarcely increases the complexity of the static gate and uses half as many stages, but this is only true for low electrical efforts. Figure 6.43 shows the paths annotated with (a) logical effort, (b) parasitic delay, and (c) total delay. The parasitic delays only consider diffusion capacitance on the output node. The delay of each design is plotted against path electrical effort H^5. For $H > 2.9$, the 4-stage design becomes preferable because the domino gates are effective buffers.

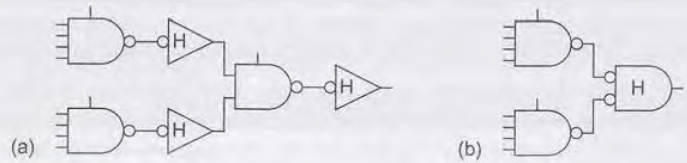


FIG 6.42 8-input domino AND gates

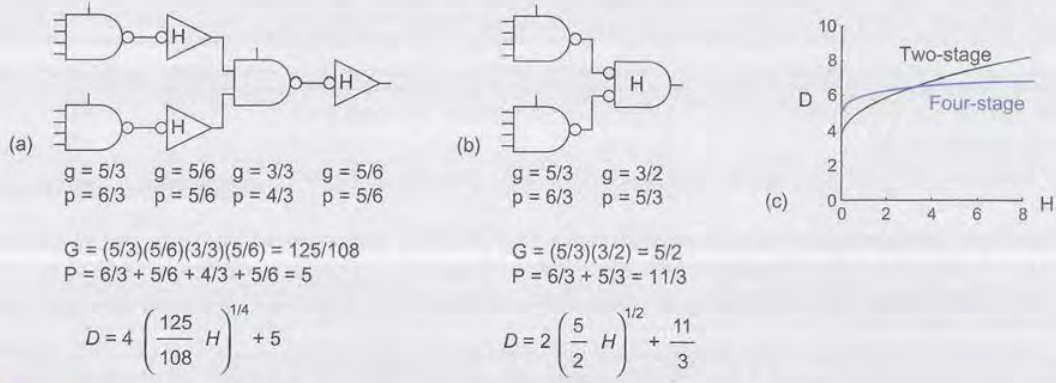


FIG 6.43 8-input domino AND delays

[5]Do not confuse the path electrical effort H with the letter H designating the HI-skew static CMOS gates in the schematic.

pMOS transistors and reduces the number of stages. Thus, it is usually good to use static CMOS gates only on paths with low electrical effort.

6.2.4.6 Multiple-output Domino Logic (MODL) It is often necessary to compute multiple functions where one is a subfunction of another or shares a subfunction. *Multiple-output domino logic* (MODL) [Hwang89, Wang97] saves area by combining all of the computations into a multiple-output gate.



A popular application is in addition, where the carry-out c_i of each bit of a 4-bit block must be computed, as will be discussed in Section 10.2.2.2. Each bit position i in the block can either propagate the carry (p_i) or generate a carry (g_i). The carry-out logic is

$$\begin{aligned} c_1 &= g_1 + p_1 c_0 \\ c_2 &= g_2 + p_2(g_1 + p_1 c_0) \\ c_3 &= g_3 + p_3(g_2 + p_2(g_1 + p_1 c_0)) \\ c_4 &= g_4 + p_4(g_3 + p_3(g_2 + p_2(g_1 + p_1 c_0))) \end{aligned} \tag{6.4}$$

This can be implemented in four compound AOI gates, as shown in Figure 6.44(a). Notice that each output is a function of the less significant outputs. The more compact MODL design is often called a *Manchester carry chain*, shown in Figure 6.44(b). Note that the intermediate outputs require secondary precharge transistors. Also note that care must be taken for certain inputs to be mutually exclusive in order to avoid *sneak paths*. For example, in the adder we must define

$$\begin{aligned} g_i &= a_i b_i \\ p_i &= a_i \oplus b_i \end{aligned} \tag{6.5}$$

If p_i were defined as $a_i + b_i$, a sneak path could exist when a_4 and b_4 are '1' and all other inputs are '0.' In that case, $g_4 = p_4 = 1$. c_4 would fire as desired, but c_3 would also fire incorrectly, as shown in Figure 6.45.

6.2.4.7 NP and Zipper Domino Another variation on domino is shown in Figure 6.46(a). The HI-skew inverting static gates are replaced with predischarged dynamic gates using pMOS logic. For example, a footed dynamic p-logic NOR gate is shown in Figure 6.46(b). When ϕ is 0, the first and third stages precharge high while the second stage predischarges low. When ϕ rises, all the stages evaluate. Domino connections are possible, as shown in Figure 6.46(c). The design style is called *NP Domino* or *NORA Domino* (NO RAce) [Gonclaves83, Friedman84].



NORA has two major drawbacks. The logical effort of footed p-logic gates is generally worse than that of HI-skew gates (e.g., 2 vs. 3/2 for NOR2 and 4/3 vs. 1 for NAND2). Secondly, NORA is extremely susceptible to noise. In an ordinary dynamic

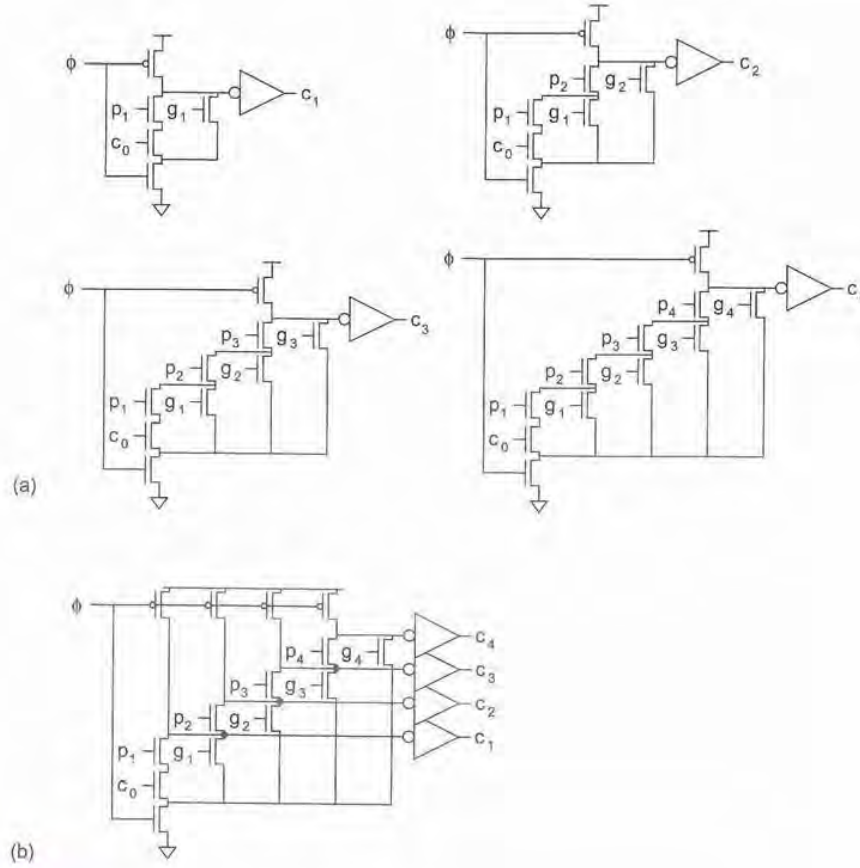


FIG 6.44 Conventional and MODL carry chains

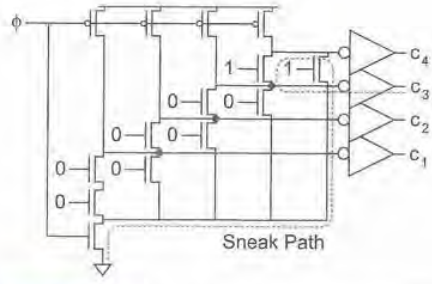


FIG 6.45 Sneak path

gate, the input has a low noise margin (about V_t), but is strongly driven by a static CMOS gate. The floating dynamic output is more prone to noise from coupling and charge sharing, but drives another static CMOS gate with a larger noise margin. In NORA, however, the sensitive dynamic inputs are driven by noise-prone dynamic outputs. Given these drawbacks and the extra clock phase required, there is little reason to use NORA.

Zipper domino [Lee86] is a closely related technique that leaves the precharge transistors slightly ON during evaluation by using precharge clocks that swing between 0 and $V_{DD} - |V_{tp}|$ for the pMOS precharge and V_{tn} and V_{DD} for the nMOS precharge. This

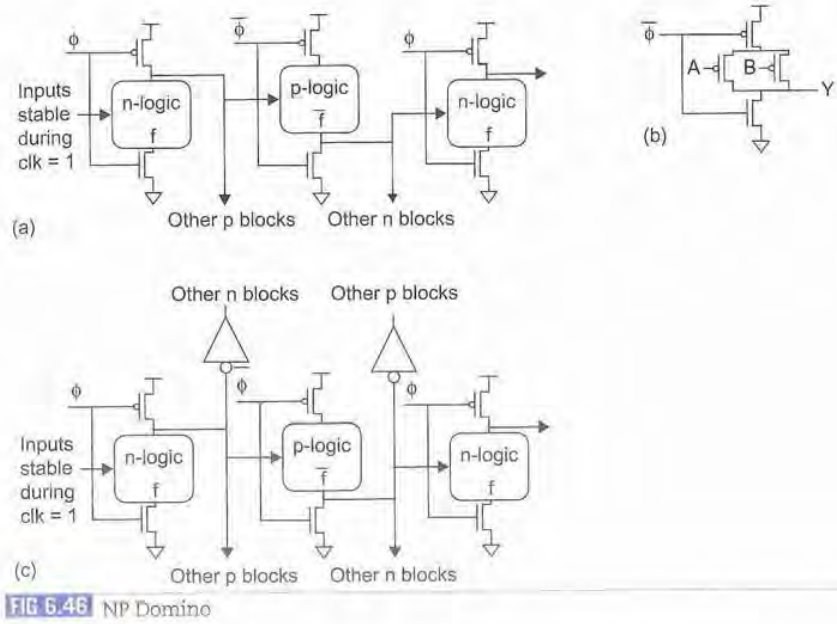


FIG 6.46 NP Domino

plays much the same role as a keeper. Zipper never saw widespread use in the industry [Bernstein99].

6.2.5 Pass-transistor Circuits

In the circuit families we have explored so far, inputs are applied only to the gate terminals of transistors. In pass-transistor circuits, inputs are also applied to the source/drain diffusion terminals. These circuits build switches using either nMOS pass transistors or parallel pairs of nMOS and pMOS transistors called *transmission gates*. Many authors have claimed substantial area, speed, and/or power improvements for pass transistors compared to static CMOS logic. In specialized circumstances this can be true; for example, pass transistors are essential to the design of efficient 6-transistor static RAM cells used in most modern systems (see Section 11.2). Full adders and other circuits rich in XORs also can be efficiently constructed with pass transistors. In certain other cases, we will see that pass-transistor circuits are essentially equivalent ways to draw the fundamental logic structures we have explored before. An independent evaluation finds that for most general-purpose logic, static CMOS is superior in speed, power, and area [Zimmermann97a].

For the purpose of comparison, Figure 6.47 shows a 2-input multiplexer constructed in a wide variety of pass-transistor circuit families along with static CMOS, pseudo-nMOS, CVSL, and single- and dual-rail domino. Some of the circuit families are dual-rail, producing both true and complementary outputs, while others are single-rail and may require an additional inversion if the other polarity of output is needed. U XOR V can be

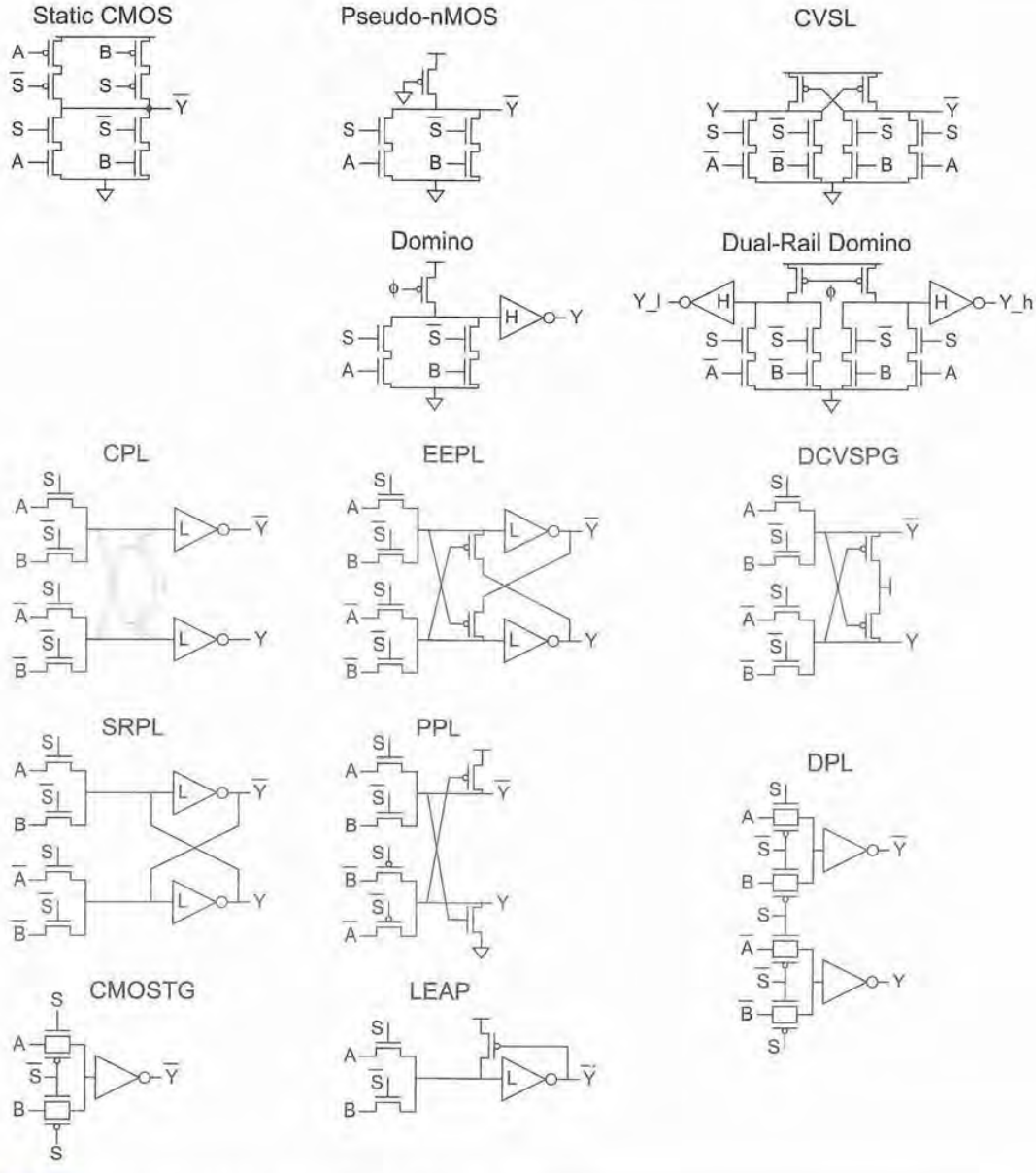


FIG 6.47 Comparison of circuit families for 2-input multiplexers

computed with exactly the same logic using $S = U$, $\overline{S} = \overline{U}$, $A = V$, $B = \overline{V}$. This shows that static CMOS is particularly poorly suited to XOR because the complex gate and two additional inverters are required; hence, pass-transistor circuits become attractive. In comparison, static CMOS NAND and NOR gates are relatively efficient and benefit less from pass transistors.

This section first examines mixing CMOS with transmission gates, as is common in multiplexers and latches. It next examines *Complementary Pass-transistor Logic* (CPL), which can work well for XOR-rich circuits like full adders and *LEAn integration with Pass transistors* (LEAP), which illustrates single-ended pass-transistor design. Finally, it catalogs and compares a wide variety of alternative pass-transistor families.

6.2.5.1 CMOS with Transmission Gates Structures such as tristates, latches, and multiplexers are often drawn as transmission gates in conjunction with simple static CMOS logic. For example, Figure 1.27 introduced the transmission gate multiplexer using two transmission gates. The circuit was nonrestoring; i.e., the logic levels on the output are no better than those on the input so a cascade of such circuits may accumulate noise. To buffer the output and restore levels, a static CMOS output inverter can be added, as in Figure 6.47 (CMOSTG).

At first, CMOS with transmission gates might appear to offer an entirely new range of circuit constructs. A careful examination shows that the topology is actually almost identical to static CMOS, as was seen in Section 2.5.6. If multiple stages of logic are cascaded, they can be viewed as alternating transmission gates and inverters. Figure 6.48(a) redraws the multiplexer to include the inverters from the previous stage that drive the diffusion inputs but to exclude the output inverter. Figure 6.48(b) shows this multiplexer drawn at the transistor level. Observe that this is identical to the static CMOS multiplexer of Figure 6.47 except that the intermediate nodes in the pull-up and pull-down networks are shorted together as *N1* and *N2*.

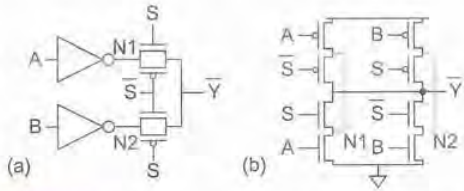


FIG 6.48 Alternate representations of CMOSTG in a 2-input inverting multiplexer

The shorting of the intermediate nodes has two effects on delay. The effective resistance decreases somewhat (especially for rising outputs) because the output is pulled up or down through the parallel combination of both pass transistors rather than through a single transistor. However, the effective capacitance increases slightly because of the extra diffusion and wire capacitance required for this shorting. This is apparent from layouts of the multiplexers; the transmission gate design in Figure 6.49(a) requires contacted diffusion on *N1* and *N2* while the static CMOS gate in Figure 6.49(b) does not. In most processes the improved resistance dominates for gates with moderate fanouts, making shorting generally beneficial.

There are several factors that favor the static CMOS representation over CMOS with transmission gates. If the inverter is on the output rather than the input, the delay of the gate depends on what is driving the input as well as the capacitance driven by the output. This input driver sensitivity makes characterizing the gate more difficult and is incompati-

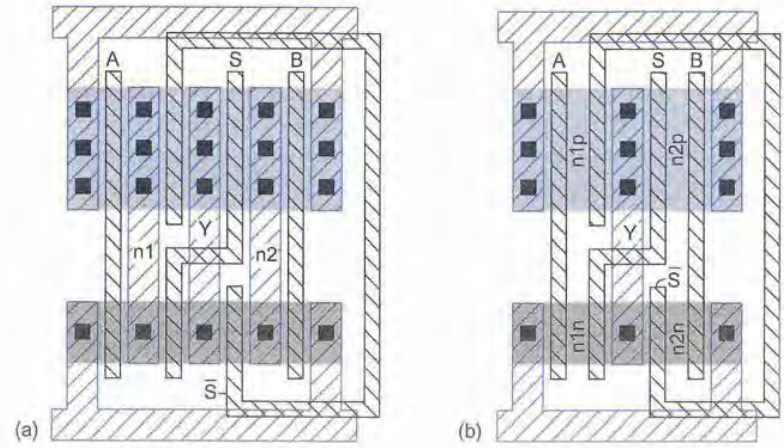


FIG 6.49 Multiplexer layout comparison

ble with most timing analysis tools. Novice designers often erroneously characterize transmission gate circuits by applying a voltage source directly to the diffusion input. This makes transmission gate multiplexers look very fast because they only involve one transistor in series rather than two. For accurate characterization, the driver must also be included. A second drawback is that diffusion inputs to tristate inverters are susceptible to noise that may incorrectly turn on the inverter; this is discussed further in Section 6.3. Finally, the contacts slightly increase area and their capacitance increases power consumption.

If transmission gates are used, it is common practice to make the pMOS and nMOS transistors roughly the same width rather than use a double-width pMOS. This is done because the two transistors pass a signal in parallel. The incremental resistance benefit of a wider pMOS transistor is not worth the extra gate and diffusion capacitance it presents. The effective resistance of a transmission gate built from unit transistors is approximately R in both rising and falling directions, as discussed in Section 2.6. The logical effort of circuits involving transmission gates is computed by drawing stages that begin at gate inputs rather than diffusion inputs, as in Figure 6.50 for a transmission gate multiplexer. The effect of the shorting can be ignored, so the logical effort from either the A or B terminals is 6/3, just as in a static CMOS multiplexer. Note that the parasitic delay of transmission gate circuits with multiple series transmission gates increases rapidly because of the internal diffusion capacitance, so it is seldom beneficial to use more than two transmission gates in series without buffering.

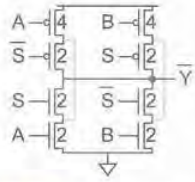


FIG 6.50 Logical effort of transmission gate circuit

6.2.5.2 Complementary Pass Transistor Logic (CPL) CPL [Yano90] can be understood as an improvement on CVSL. CVSL is slow because one side of the gate pulls down, and then the cross-coupled pMOS transistor pulls the other side up. The size of the

cross-coupled device is an inherent compromise between a large transistor that fights the pull-down excessively and a small transistor that is slow pulling up. CPL resolves this problem by making one half of the gate pull up while the other half pulls down.

Figure 6.51(a) shows the CPL multiplexer from Figure 6.47 rotated sideways. If a path consists of a cascade of CPL gates, the inverters can be viewed equally well as being on the output of one stage or the input of the next. Figure 6.51(b) redraws the mux to include the inverters from the previous stage that drives the diffusion input, but to exclude the output inverters. Figure 6.51(c) shows the mux drawn at the transistor level. Observe that this is identical to the CVSL gate from Figure 6.47 except that the internal node of the stack can be pulled up through the weak pMOS transistors in the inverters.

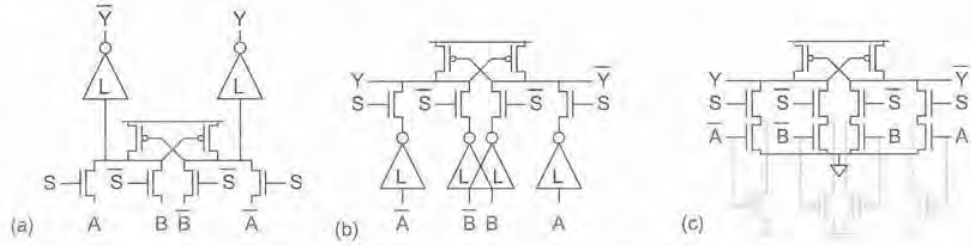


FIG 6.51 Alternate representations of CPL

When the gate switches, one side pulls down well through its nMOS transistors. The other side pulls up. CPL can be constructed without cross-coupled pMOS transistors, but the outputs would only rise to $V_{DD} - V_t$ (or slightly lower because the nMOS transistors experience the body effect). This costs static power because the output inverter will be turned slightly ON. Adding weak cross-coupled devices helps bring the rising output to the supply rail while only slightly slowing the falling output. The output inverters can be LO-skewed to reduce sensitivity to the slowly rising output.

6.2.5.3 Lean Integration with Pass Transistors (LEAP) Like CPL, LEAP[6] [Yano96] builds logic networks using only fast nMOS transistors, as shown in Figure 6.47. It is a single-ended logic family in that the complementary network is not required, thus saving area and power. The output is buffered with an inverter, which can be LO-skewed to favor the asymmetric response of an nMOS transistor. The nMOS network only pulls up to $V_{DD} - V_t$ so a pMOS feedback transistor is necessary to pull the internal node fully high, avoiding power consumption in the output inverter. The pMOS width is a tradeoff between fighting falling transitions and assisting the last part of a rising transition; it generally should be quite weak and the circuit will fail if it is too strong. LEAP can be a good way to build wide 1-of-N hot multiplexers with many of the advantages of pseudo-nMOS but

[6]The LEAP topology was reinvented under the name *Single Ended Swing Restoring Pass Transistor Logic* [Pihl98].

without the static power consumption. It was originally proposed for use in a pass transistor logic synthesis system because the cells are compact.

Unlike most circuit families that can operate down to $V_{DD} \geq \max(V_{tn}, |V_{tp}|)$, LEAP is limited to operating at $V_{DD} \geq 2V_t$ because the inverter must flip even when receiving an input degraded by a threshold voltage.



6.2.5.4 Other Pass Transistor Families There have been a host of pass transistor families proposed in the literature, including *Differential Pass Transistor Logic* (DPTL) [Pasternak87, Pasternak91], *Double Pass Transistor Logic* (DPL) [Suzuki93], *Energy Economized Pass Transistor Logic* (EEPL) [Song96], *Push-Pull Pass Transistor Logic* (PPL) [Paik96], *Swing-Restored Pass Transistor Logic* (SRPL) [Parameswar96], and *Differential Cascode Voltage Switch with Pass Gate Logic* (DCVSPG) [Lai97]. All of these are dual-rail families like CPL, as contrasted with the single-rail CMOSTG and LEAP.

DPL is a double-rail form of CMOSTG optimized to use single pass transistors where only a known '0' or '1' needs to be passed. It passes good high and low logic levels without the need for level-restoring devices. However, the pMOS transistors contribute substantial area and capacitance, but do not help the delay much, resulting in large and relatively slow gates.

The other dual-rail families can be viewed as modifications to CPL. EEPL drives the cross-coupled level restoring transistors from the opposite rail rather than V_{DD}. The inventors claimed this led to shorter delay and lower power dissipation than CPL, but the improvements could not be confirmed [Zimmermann97a]. SRPL cross-couples the inverters instead of using cross-coupled pMOS pull-ups. This leads to a ratio problem in which the nMOS transistors in the inverter must be weak enough to be overcome as the pass transistors try to pull up. This tends to require small inverters, which make poor buffers. DCVSPG eliminates the output inverters from CPL. Without these buffers, the output of a DCVSPG gate makes a poor input to the diffusion terminal of another DCVSPG gate because a long unrestored chain of nMOS transistors would be formed, leading to delay and noise problems. PPL also has unbuffered outputs and associated delay and noise issues. DPTL generalizes the output buffer structure to consider alternatives to the cross-coupled pMOS transistors and LO-skewed inverters of CPL. All of the alternatives are slower and larger than CPL.

6.3 Circuit Pitfalls

Circuit designers tend to use simple circuits because they are robust. Elaborate circuits, especially those with more transistors, tend to add more area, more capacitance, and more things that can go wrong. Static CMOS is the most robust circuit family and should be used whenever possible. This section catalogs a variety of circuit pitfalls that can cause chips to fail. They include:

- threshold drops
- ratio failures

- leakage
- charge sharing
- power supply noise
- coupling
- minority carrier injection
- back-gate coupling
- diffusion input noise sensitivity
- race conditions
- delay matching
- metastability
- hot spots
- soft errors
- process sensitivity

Capacitive and inductive coupling were discussed in Section 4.5. Sneak paths were discussed in Section 6.2.4.6. Reliability issues such as soft errors impacting circuit design were discussed in Section 4.8. Timing-related problems including race conditions, delay matching, and metastability will be examined in Sections 7.2.3, 7.5.4, and 7.6.1. The other pitfalls are described here.

6.3.1 Threshold Drops

Pass transistors are good at pulling in a preferred direction, but only swing to within V_t of the rail in the other direction; this is called a *threshold drop*. For example, Figure 6.52 shows a pass transistor driving a logic '1' into an inverter. The output of the pass transistor only rises to $V_{DD} - V_t$. Worse yet, the body effect increases this threshold voltage because $V_{sb} > 0$ for the pass transistor. The degraded level is insufficient to completely turn off the pMOS transistor in the inverter, resulting in static power dissipation. Indeed, for low V_{DD}, the degraded output can be so poor that the inverter no longer sees a valid input logic level V_{IH}. Finally, the transition becomes lethargic as the output approaches $V_{DD} - V_t$. Threshold drops were sometimes tolerable in older processes where $V_{DD} \approx 5V_t$, but are seldom acceptable in modern processes where the power supply has been scaled down faster than the threshold voltage to $V_{DD} \approx 3V_t$. As a result, pass transistors must be replaced by full transmission gates or may use weak pMOS feedback transistors to pull the output to V_{DD}, as was done in several pass transistor families.

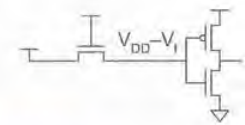


FIG 6.52 Pass transistor with threshold drop

6.3.2 Ratio Failures

Pseudo-nMOS circuits illustrated ratio constraints that occur when a node is simultaneously pulled up and down, typically by strong nMOS transistors and weak pMOS transistors. The weak transistors must be sufficiently small that the output level falls below V_{IL} of the next stage by some noise margin. Ideally, the output should fall below V_t so the next stage does not conduct static power. Ratioed circuits should be checked in the SF and FS corners.

Another example of ratio failures occurs in circuits with feedback. For example, dynamic keepers, level-restoring devices in SRPL and LEAP, and feedback inverters in static latches all have weak feedback transistors that must be ratioed properly.

Ratioing is especially sensitive for diffusion inputs. For example, Figure 6.53(a) shows a static latch with a weak feedback inverter. The feedback inverter must be weak enough to be overcome by the series combination of the pass transistor and the gate driving the D input, as shown in Figure 6.53(b). This cannot be verified by checking the latch alone; it requires a global check of the latch and driver. Worse yet, if the driver is far away, the series wire resistance must also be considered, as shown in Figure 6.53(c).

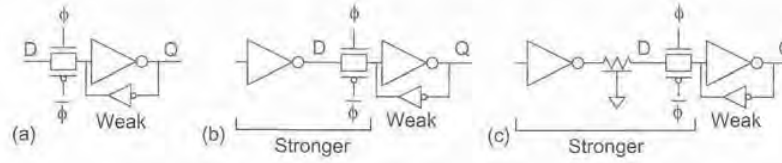


FIG 6.53 Ratio constraint on static latch with diffusion input

6.3.3 Leakage

Leakage current is a growing problem as technology scales, especially for dynamic nodes and wide NOR structures. Recall that leakage arises from subthreshold conduction, gate tunneling, and reverse-biased diode leakage. Subthreshold conduction is presently the most important component because V_t is low and getting lower, but gate tunneling will become profoundly important too as oxide thickness diminishes. Besides causing static power dissipation, leakage can result in incorrect values on dynamic or weakly driven nodes. The time required for leakage to disturb a dynamic node by some voltage ΔV is

$$t = \frac{C_{node}\Delta V}{I_{leak}} \tag{6.6}$$

Subthreshold leakage gradually discharges dynamic nodes through transistors that are nominally OFF. Fully dynamic gates and latches without keepers are not viable in most modern processes. DRAM refresh times are also set by leakage and DRAM processes must minimize leakage to have satisfactory retention times.

Even when a keeper is used, it must be wide enough. This seems trivial because the keeper is fully ON while leakage takes place through transistors that are supposed to be OFF. However, in wide dynamic NOR structures, many parallel nMOS transistors may be leaking simultaneously. Similar problems apply to wide pseudo-nMOS NOR gates and PLAs. Leakage increases exponentially with temperature, so the problem is especially bad at burn-in. For example, a preliminary version of the Sun UltraSparc V had difficulty with burn-in because of excess leakage.

Subthreshold leakage is much lower through two OFF transistors in series than through a single transistor because the outer transistor has a lower drain voltage and sees a much lower effect from DIBL. Multiple threshold voltages are also frequently used to achieve high performance in critical paths and lower leakage in other paths.

6.3.4 Charge Sharing

Charge sharing was introduced in Section 6.2.4.4 in the context of a dynamic gate. Charge sharing can also occur when dynamic gates drive pass transistors. For example, Figure 6.54 shows a dynamic inverter driving a transmission gate. Suppose the dynamic gate has been precharged and the output is floating high. Further suppose the transmission gate is OFF and $Y = 0$. If the transmission gate turns on, charge will be shared between X and Y, disturbing the dynamic output.

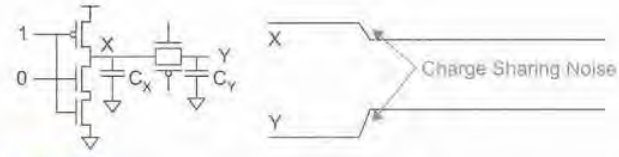


FIG 6.54 Charge sharing on dynamic gate driving pass transistor

6.3.5 Power Supply Noise

V_{DD} and GND are not constant across a large chip. Both are subject to *power supply noise* caused by IR drops and di/dt noise. IR drops occur across the resistance R of the power supply grid between the supply pins and a block drawing a current I, as shown in Figure 6.55. di/dt noise occurs across the power supply inductance L as the current rapidly changes. di/dt noise can be especially important for blocks that are idle for several cycles and then begin switching. Power supply noise hurts performance and can degrade noise margins. Typical targets are for power supply noise on the order of 5%–10% of V_{DD}. Power supply noise causes both noise margin problems and delay variations. The noise margin issues can be managed by placing sensitive circuits near each other and having them share a common low-resistance power wire.

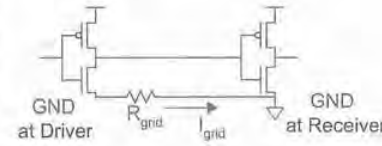


FIG 6.55 Power supply IR drops

Power supply noise can be estimated from simulations of the chip power grid, bypass capacitance, and packaging, as will be discussed in Section 12.3. Figure 6.56 shows a plot of the simulated power supply voltage (V_{DD} – GND) at points across the Itanium 2 [Harris01b].

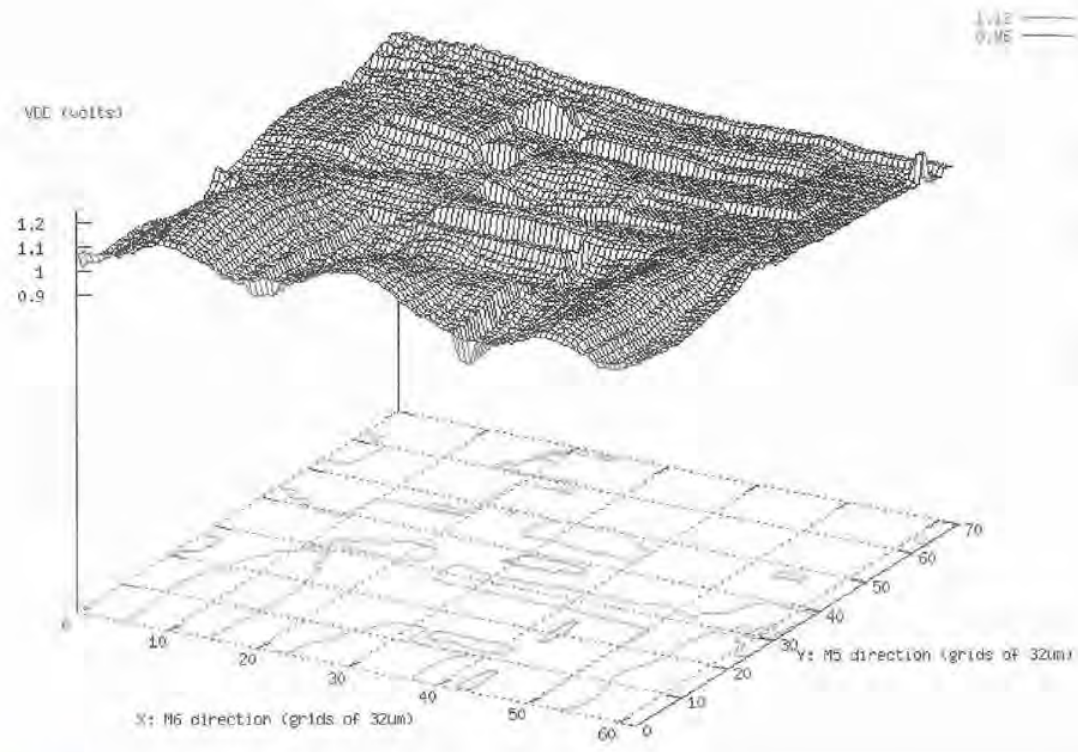


FIG 6.56 Itanium 2 supply voltage. © IEEE 2001.

6.3.6 Hot Spots

Transistor performance degrades with temperature, so care must be taken to avoid excessively *hot spots*. These can be caused by nonuniform power dissipation even when the overall power consumption is within budget. The nonuniform temperature distribution leads to variation in delay between gates across the chip. Full-chip temperature plots can be generated through electrothermal simulation [Petegem94, Cheng00y]; this can begin when the floorplan and preliminary power estimates for each unit are available. Figure 6.57 plots the simulated temperature distribution of the Itanium 2 processor. Observe the temperature gradient across the chip from the hot integer core near the center to the cool level2 caches around the periphery. A particularly localized form of hot spots is self-heating in resistive wires, described in Section 4.8.3.

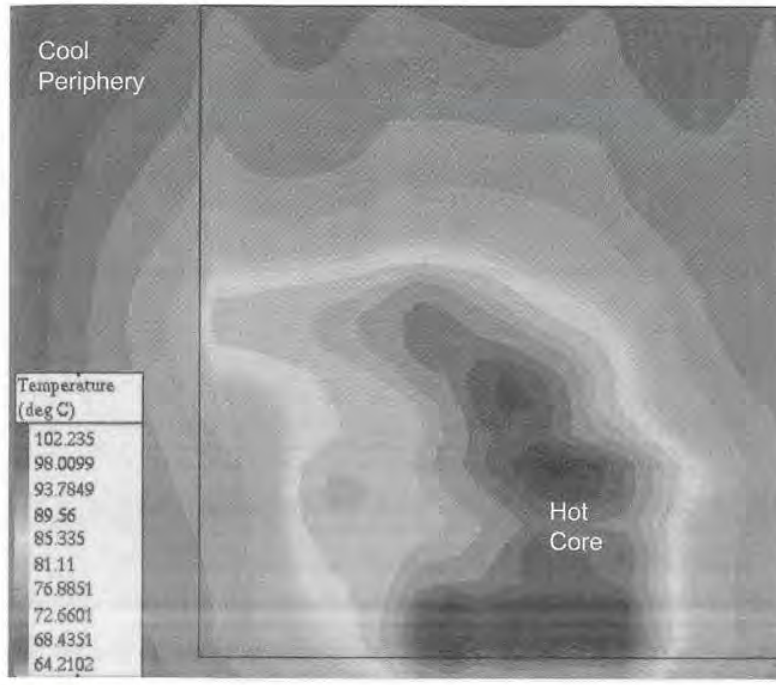


FIG 6.57 Itanium 2 die temperature. © IEEE 2001.

6.3.7 Minority Carrier Injection

It is sometimes possible to drive a signal momentarily outside the rails, either through capacitive coupling or through inductive ringing on I/O drivers. In such a case, the junctions between drain and body may momentarily become forward-biased, causing current to flow into the substrate. This effect is called *minority carrier injection* [Chandrakasan01]. For example, in Figure 6.58, the drain of an nMOS transistor is driven below GND, injecting electrons into the p-type substrate. These can be collected on a nearby transistor diffusion node (Figure 6.58(a)), disturbing a high voltage on the node. This is a particular problem for dynamic nodes and sensitive analog circuits.

Minority carrier injection problems are avoided by keeping injection sources away from sensitive nodes. In particular, I/O pads should not be located near sensitive nodes. Noise tools can identify potential coupling problems so the layout can be modified to reduce coupling. Alternatively, the sensitive node can be protected by an intermediate substrate or well contact. For example in Figure 6.58(b), most of the injected electrons will be collected into the substrate contact before reaching the dynamic node. In I/O pads, it is

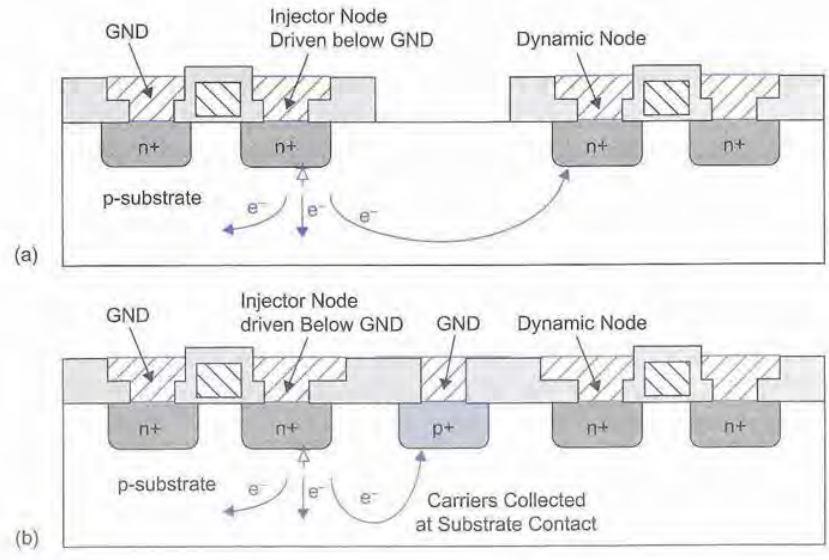


FIG 6.58 Minority carrier injection and collection

common to build *guard rings* of substrate/well contacts around the output transistors. Guard rings were illustrated in Figure 4.64.

6.3.8 Back-gate Coupling

Dynamic gates driving multiple-input static CMOS gates are susceptible to the *back-gate coupling* effect [Chandrakasan01] illustrated in Figure 6.59. In this example, a dynamic NAND gate drives a static NAND gate. The gate-to-source capacitance C_{gs1} of *N1* is shown explicitly. Suppose that the dynamic gate is in evaluation and its output X is floating high. The other input B to the static NAND gate is initially low. Therefore, the NAND output Y is high and the internal node W is charged up to $V_{DD} - V_t$. At some time B rises, discharging Y and W through transistor *N2*. The source of *N1* falls. This tends to bring the gate along for the ride because of the C_{gs1} capacitance, resulting in a droop on the dynamic node X. As with charge sharing, the magnitude of the droop depends on the ratio of C_{gs1} to the total capacitance on node X.

Back-gate coupling is eliminated by driving the input closer to the rail. For example, if X drove *N2* instead of *N1*, the problem would be avoided. Otherwise, the back-gate coupling noise must be included in the dynamic noise budget.

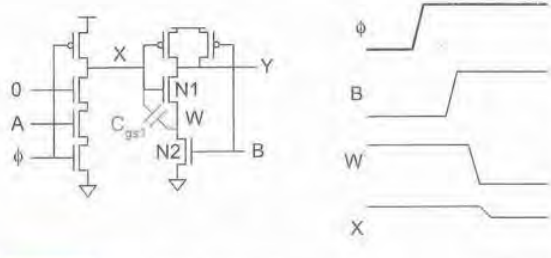


FIG 6.59 Back-gate coupling

6.3.9 Diffusion Input Noise Sensitivity

Figure 6.53(a) showed a static latch with an exposed diffusion input. Such an input is also particularly sensitive to noise. For example, imagine that power supply noise and/or coupling noise drove the input voltage below $-V_t$ relative to GND seen by the transmission gate, as shown in Figure 6.60. V_{gs} now exceeds V_t for the nMOS transistor in the transmission gate, so the transmission gate turns on. If the latch had contained a '1,' it could be incorrectly discharged to '0.' A similar effect can occur for voltage excursions above V_{DD}.

For this reason, along with the ratio issues discussed in Section 6.3.2, standard cell latches are usually built with buffered inputs rather than exposed diffusion nodes. Exposing the diffusion input results in a faster latch and can be used in datapaths where the inputs are carefully controlled and checked.

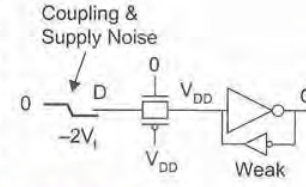


FIG 6.60 Noise on diffusion input of latch

6.3.10 Process Sensitivity

Marginal circuits can operate under nominal process conditions, but fail in certain process corners or when the circuit is migrated to another process. Novel circuits should be simulated in all process corners and carefully scrutinized for any process sensitivities. They should also be verified to work at all voltages and temperatures, including the elevated voltages and temperatures used during burn-in and the lower voltage that might be used for low-power versions of a part.

When a design is likely to be migrated to another process for cost-reduction, circuits should be designed to facilitate this migration. You can expect that leakage will increase, threshold drops will become a greater fraction of the supply voltage, wire delay will become a greater portion of the cycle time, and coupling may get worse as aspect ratios of wires increase. For example, the Pentium 4 processor was originally fabricated in a 180 nm process. Designers placed repeaters closer than was optimal for that process because they knew the best repeater spacing would become smaller as transistor dimensions were reduced later in the product's life [Kumar01].

6.3.11 Example: Domino Noise Budgets

Domino logic is important in many high-performance microprocessors, but requires careful verification because it is sensitive to noise. Noise in static CMOS gates usually results in greater delay, but noise in domino logic can produce incorrect results. This section reviews the various noise sources that can affect domino gates and presents a sample noise budget.

Dynamic outputs are especially susceptible to noise when they float high, held only by a weak keeper. Dynamic inputs have low noise margins (approximately V_t). Noise issues that should be considered include [Chandrakasan01]:

- **Charge leakage** Subthreshold leakage on the dynamic node is presently most important, but gate leakage will become important, too. Subthreshold leakage is worst for wide NOR structures at high temperature (especially during burn-in). Keepers must be sized appropriately to compensate for leakage.

- **Charge sharing** Charge sharing can take place between the dynamic output node and the nodes within the dynamic gate. Secondary precharge transistors should be added when the charge sharing could be excessive. Do not drive dynamic nodes directly into transmission gates because charge sharing can occur when the transmission gate turns ON.
- **Capacitive coupling** Capacitive coupling can occur on both the input and output. The inputs of dynamic gates have the lowest noise margin, but are actively driven by a static gate, which fights coupling noise. The dynamic outputs have more noise tolerance, but are weakly driven. Coupling is minimized by keeping wires short and increasing the spacing to neighbors or shielding the lines. Coupling can be extremely bad in processes below 250 nm because the wires have such high aspect ratios.
- **Back-gate coupling** Dynamic gates connected to multiple-input CMOS gates should drive the outer input when possible. This is not a factor for dynamic gates driving inverters.
- **Minority carrier injection** Dynamic nodes should be protected from nodes that can inject minority carriers. These include I/O circuits and nodes that can be coupled far outside the supply rails. Substrate/well contacts and guard rings can be added to protect dynamic nodes from potential injectors.
- **Power supply noise** Static gates should be located close to the dynamic gates they drive to minimize the amount of power supply noise seen.
- **Soft errors** Alpha particles and cosmic rays can disturb dynamic nodes. The probability of failure is reduced through large node capacitance and strong keepers.
- **Noise feedthrough** Noise that pushes the input of a previous stage to near its noise margin will cause the output to be slightly degraded, as was shown in Figure 2.28.
- **Process corner effects** Noise margins are degraded in certain process corners. Dynamic gates have the smallest noise margin in the FS corner where the nMOS transistors have a low threshold and the pMOS keepers are weak. HI-skew static gates have the smallest noise margins in the SF corner where the gates are most skewed.

In a domino gate, the noise-prone dynamic output drives a static gate with a reasonable noise margin. The noise-sensitive dynamic gate is strongly driven by a noise-resistant static gate. In an NP domino gate or clock-delayed domino gate, the noise-prone dynamic output directly drives a noise-sensitive dynamic input, making such circuits particularly risky.

Consider a noise budget for a 3.3 V process [Harris01a]. A HI-skew inverter in this process has V_{IH} = 2.08 V, resulting in NM_H = 37% of V_{DD} if $V_{OH} = V_{DD}$ A dynamic gate with a small keeper has V_{IL} = 0.63 V, resulting in NM_L = 19% of V_{DD}. Table 6.3 allocates

these margins to the primary noise sources. In a full design methodology, different margins can be used for different gates. For example, wide NOR structures have no charge sharing noise, but may see significant leakage instead. More coupling noise could be tolerated if other noise sources are known to be smaller. Noise analysis tools are discussed further in Section 8.4.2.7.

Table 6.3 Sample domino noise budget

Source	Dynamic Output	Dynamic Input
Charge sharing	10	n/a
Coupling	17	7
Supply noise	5	5
Feedthrough noise	5	7
Total	37%	19%

6.4 More Circuit Families



Static CMOS is satisfactory for the great majority of logic gates in modern integrated circuits and an assortment of domino, pass-transistor circuits, and pseudo-nMOS accounts for nearly all of the remaining gates. A large number of other circuit families have been proposed in the literature. This section describes some of these circuit families and their strengths and limitations.

6.4.1 Differential Circuits

Several differential circuit families using nMOS pull-down networks are derived from the basic CVSL form, as shown in Figure 6.61.

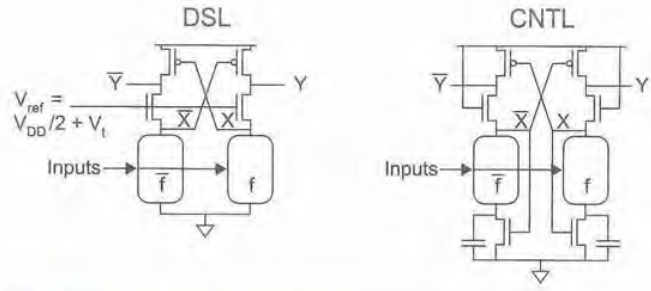


FIG 6.61 Differential circuit families

6.4.1.1 Differential Split-Level (DSL) *Differential Split-Level* (DSL) [Pfennings85] places nMOS transistors in series with the basic CVSL pull-down networks. By applying a reference voltage of $V_{DD}/2 + V_t$ to these transistors, the swing on the internal nodes (X and $\overline{X}$) are limited to $0 - V_{DD}/2$. This reduces the parasitic delay of the pull-down stacks. The lower internal voltages also lead to lower electric fields across the pull-down transistors. The inventors took advantage of this lower voltage to reduce the channel lengths of the transistors without compromising hot-electron reliability. They claimed a tenfold speedup over a static CMOS full adder; this was attributed to a factor of 2 for the CVSL structure, another factor of 2 from the low-swing signals, and a factor of 2.5 for using shorter transistors.

In a modern process, transistors are generally as short as can be reliably manufactured, so DSL cannot use even shorter transistors. The authors have been unable to reproduce any advantage over static CMOS in a submicron process. The resistance of the extra series transistor does not help. Another disadvantage of DSL is that the voltages on the pMOS gates only swing between 0 and $V_{DD}/2$. Therefore, the pull-up that should be OFF is actually partially ON, resulting in static power dissipation. Finally, generating and distributing the reference voltage requires some effort and the reference may be sensitive to power supply noise and threshold voltage variations.

6.4.1.2 Cascode Nonthreshold Logic (CNTL) *Cascode Nonthreshold Logic* (CNTL) [Wang89] is derived from DSL by adding a transistor and shunting capacitor to the bottom of each pull-down network and setting the reference voltage to V_{DD} rather than $V_{DD}/2 + V_t$. The series transistors are connected with negative feedback. The internal swing is limited to V_t to $V_{DD} - V_t$ and there is much less quiescent current draw than in DSL because the pMOS transistors are nearly turned OFF. CNTL requires more area than CVSL and the extra series transistors tend to slow it down, although large shunting capacitors partially alleviate this problem.

CNTL is a variant of Nonthreshold Logic (NTL), shown in Figure 6.62, which is essentially a pseudo-nMOS gate with an extra transistor and shunting capacitor in series with the pull-down network. The shunting capacitor is built from the gate of an nMOS transistor. NTL consumes static power and is slower than pseudo-nMOS.

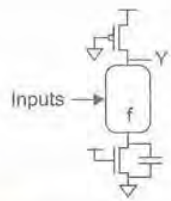


FIG 6.62 Nonthreshold Logic

6.4.2 Sense-amplifier Circuits

Sense amplifiers magnify small differential input voltages into larger output voltages. They are commonly used in memories in which differential bitlines have enormous capacitive loads (see Section 11.2.3). Because of the large load, the bitlines swing slowly. To reduce this delay, the bitline voltages are first equalized. Then, when they are driven apart, the sense amplifier can detect a small swing and bring it up to normal logic levels. This reduces the ΔV term in EQ (6.1); in other words, it reduces the delay by avoiding waiting for a full swing on the bitlines. Sense amplifiers offer potential for reducing delay in heavily loaded logic circuits as well.

Figure 6.63 shows more differential circuit families derived from CVSL. These families add sense amplifiers to dual-rail domino (also repeated in the figure) to detect a small differential voltage and amplify it to a full-rail output. They will be discussed in detail later in this section.

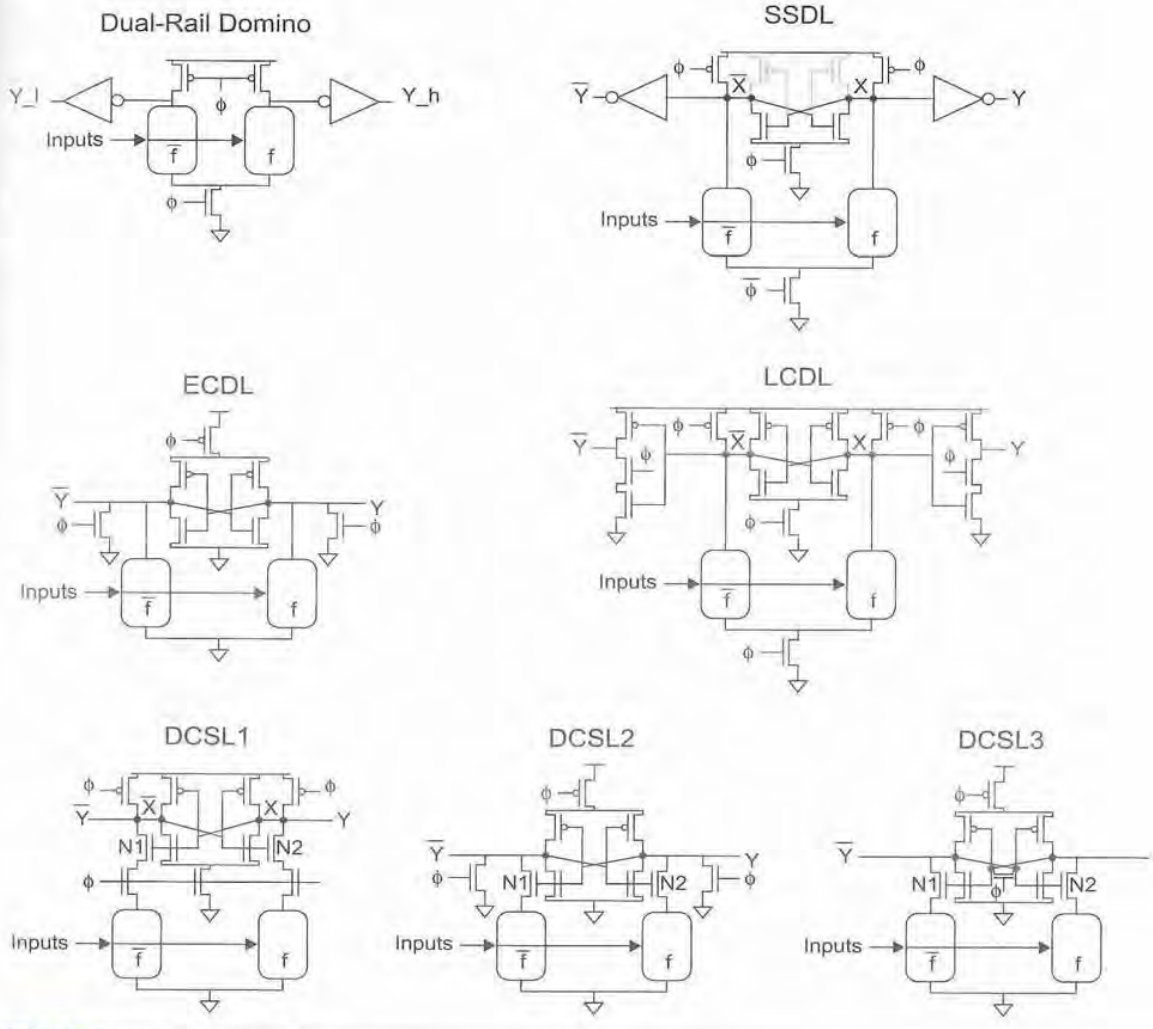


FIG 6.63 Sense-amplifier circuits

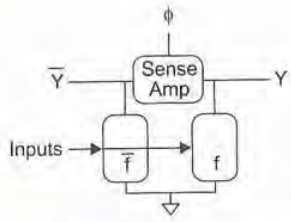


FIG 6.64 Generic sense-amplifier circuit

Figure 6.64 shows a generic sense-amplifier circuit. It works best for complex pull-down networks that would have a large RC delay. The sense amplifier fires after a small ΔV develops. Once fired, it turns on a driver with a low resistance to slew the outputs between rails. The combined delay of the pull-down stage and the sense-amplifier stage may be better than the delay of a single complex stage.

The sense amplifiers must be clocked after a sufficient differential voltage has developed. Therefore, the inputs must settle some setup time before the clock edge. The outputs become valid at some clock-to-out delay after the clock edge. The total delay of the sense-amplifier circuit is the sum of the setup time, clock-to-out delay, and any clock skew that must be budgeted (see Section 7.2.5).

As with clock-delayed domino (to be discussed in Section 7.5.4.2), it is tricky to cascade sense-amplifier circuits because the successive clocks must be delivered at the appropriate times. If only a single clock phase is used, only one sense-amplifier circuit can be placed in each cycle. If multiple clocks are generated using delay lines, sufficient timing margin must be allowed so that the delay line is always slow enough. If multiple clocks are generated through completion detection, time must be budgeted for the completion detection circuits.

An inherent tradeoff exists between the setup time and circuit reliability because a longer setup time allows a greater differential voltage ΔV to develop and overcome noise. One of the important sources of noise is charge sharing. For example, Figure 6.65(a) shows a pair of pull-down networks that are particularly sensitive to charge sharing noise. Figure 6.65(b) shows the response as the inputs arrive, assuming the outputs are precharged, node X carries a residual low voltage from a previous cycle's operation, and the sense amp is inactive. Observe that charge sharing from the large internal diffusion capacitance on node X initially causes Y to fall faster than its complement. Eventually, the resistive path pulls down the correct output Y. This charge sharing noise increases the setup time before the amplifier can safely fire. Yet another risk for unbuffered sense-amplifier circuits is that unequal output loading or coupling will cause one output to fall faster than the other, resulting in incorrect sensing. In summary, sense-amplifier circuits offer promise for special-purpose applications, but present many design risks to manage.

6.4.2.1 Sample Set Differential Logic (SSDL) *Sample Set Differential Logic* (SSDL) [Grotjohn86] modifies dual-rail domino logic by adding a clocked sense amplifier and modifying the clocking. Rather than using precharge and evaluation phases, SSDL uses *sample* and *set* phases. During sample, ϕ is low and both the precharge and evaluation transistors are ON. One of the internal nodes (X or $\overline{X}$) is precharged high while the other experiences contention between the precharge transistor and pull-down stack so its output settles somewhere below V_{DD}. Static power is consumed through the sample phase. During set when ϕ is high, precharge and evaluate transistors turn OFF and the clocked sense amplifier turns ON. The amplifier tends to pull the lower of the two internal nodes down to GND. At first, it tends to pull down the other side as well, so it is helpful to have a keeper (shown in blue) to restore the high level.

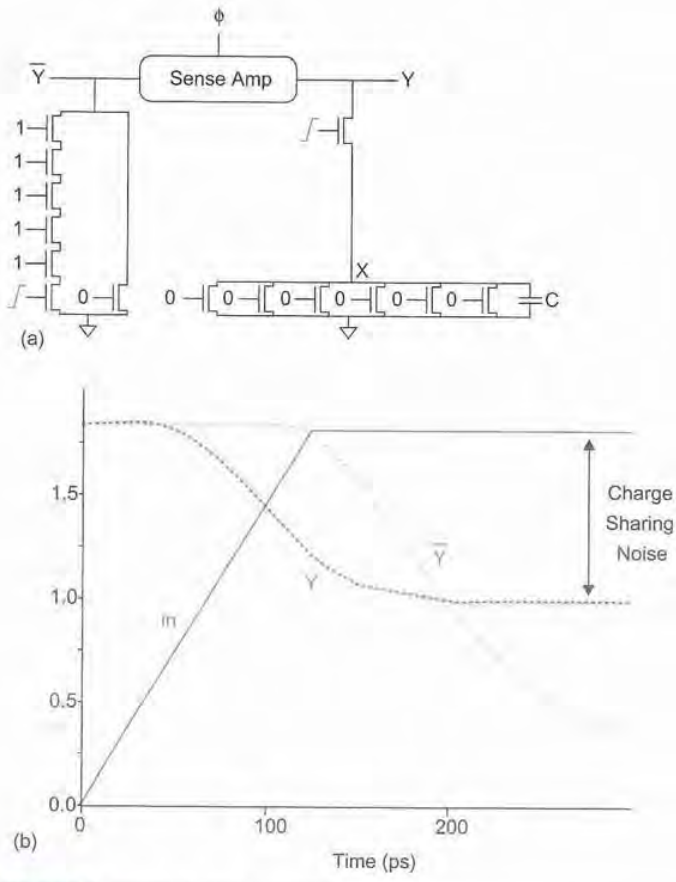


FIG 6.65 Charge sharing in sense-amplifier circuit

6.4.2.2 Enable/Disable CMOS Differential Logic (ECDL) *Enable/Disable CMOS Differential Logic* (ECDL) [Lu88b, Lu91] improves on SSDL by eliminating the static power consumption. The sense amplifier is made from a pair of cross-coupled clocked inverters, as redrawn in Figure 6.66(a) to emphasize the inverters. The cycle is again divided into two phases of operation: *enable* and *disable*. When ϕ is high, the gate is disabled. Both outputs are pulled low and the pull-up stack is turned OFF. When ϕ falls, the gate is enabled. The cross-coupled pMOS transistors are both initially ON and attempt to pull the outputs high. One output will be held down by its pull-down stack and will lag. Positive feedback will pull one output fully high and the other back fully low. The sense amplifier rising delay is somewhat longer than in SSDL because it pulls high through two series pMOS transistors.

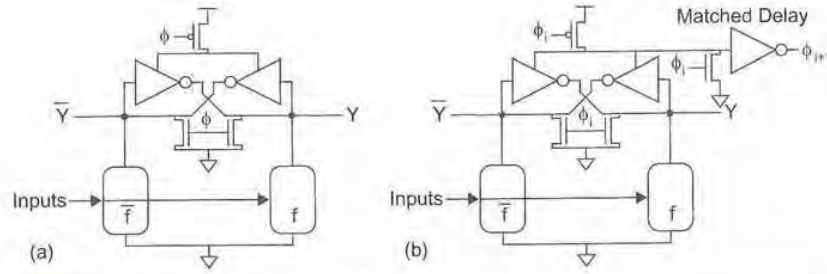


FIG 6.66 Enable/Disable CMOS Differential Logic

To avoid the difficulty of only having two clock edges in each cycle for gates, Lu proposes creating a local clock with matched delays, as shown in Figure 6.66(b). The delay from ϕ_i to ϕ_{i+1} must exceed the ECDL gate delay for correct operation. Another possibility would be to generate the next clock through *completion detection* as the OR of the two outputs.

6.4.2.3 Latched CMOS Differential Logic (LCDL) *Latched CMOS Differential Logic* (LCDL) [Wu91] adds a sense amplifier directly to the output nodes of a dual-rail domino gate and includes n-latches on the outputs. The topology is similar to SSDL, but the non-inverted clock is used for evaluation. The sense amplifier fires at exactly the same time as the dual-rail gate, so there is a serious risk of amplifying noise rather than signal. This can be overcome with a second clock to delay firing the amplifier.

6.4.2.4 Differential Current Switch Logic (DCSL) Differential circuits can consume significant power because one of the outputs transitions every cycle. *Differential Circuit Switch Logic* (DCSL) [Somasekhar96] seeks to reduce the power consumption of internal nodes and offer higher speed by swinging the pull-down networks through a small voltage. This is done by adding a pair of feedback transistors *N1* and *N2* to the SSDL and ECDL structures to cut off the pull-down networks before the internal nodes rise far above 0.

DCSL1 is a "precharge high" circuit related to SSDL and LCDL. When the clock is low, the outputs precharge high. When the clock rises, the circuit begins evaluation. As one side or the other pulls low, the sense amplifier accelerates the transition. *N1* or *N2* turns off to prevent the internal nodes of the pull-down stack on the other side from rising too much.

DCSL2 is a "precharge low" circuit related to ECDL. It again adds *N1* and *N2* to prevent the internal nodes from rising too much. DCSL3 improves on DCSL2 by replacing the two predischarge transistors with a single equalization transistor.

Because the sense amplifiers fire at the same time as the outputs begin to fall, DCSL is sensitive to amplifying noise instead of signal. It also performs poorly for $V_{DD} < 5V_t$. LVDCSL [Somasekhar98] operates better at low voltages, but uses a complex sense amplifier.

6.4.3 BiCMOS Circuits

Bipolar transistors can deliver a much higher output current than can CMOS transistors of equal input capacitance. Therefore, they can be used to build gates with low logical effort and are good for driving large capacitive loads. Gates mixing bipolar and CMOS transistors are called *BiCMOS*.

Figure 6.67 shows a BiCMOS NAND gate using two npn bipolar transistors. An npn transistor behaves as a switch between the collector and the emitter controlled by the base. The base voltage must be about 0.7 V above the emitter to turn the transistor ON. The BiCMOS gate contains an ordinary CMOS NAND gate to compute x. If A or B is '0,' x will be driven to '1.' This turns on $Q2$ and pulls the output Y up. When Y is high, $M1$ turns ON, pulling down w and turning off $Q1$. If A and B are both '1,' $M3$ and $M2$ are both 'ON.' If Y begins at '1,' w will rise to '1' and turn on $Q1$. $Q1$ in turn discharges Y to '0.'

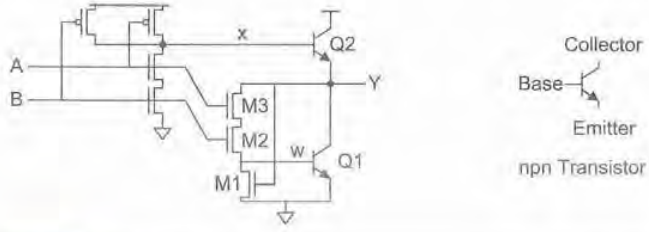


FIG 6.67 BiCMOS NAND gate

Unfortunately, bipolar transistors have an inherent V_{be} drop of about 0.7 V when ON. Hence, Y will never rise above $V_{DD} - V_{be}$. This was acceptable at V_{DD} = 5 V, tolerable at V_{DD} = 3.3 V, and perhaps manageable with elaborate circuit tricks at V_{DD} = 2.5 V. In modern processes with low supply voltages, $V_{DD} - V_{be}$ is too low to form a valid logic level, so BiCMOS circuits are no longer particularly useful for digital logic in processes below 0.35 μm. Moreover, CMOS circuits have been scaled much more aggressively than bipolar transistors, so the short-channel CMOS transistors are now competitive in performance with older, larger bipolar transistors.

6.4.4 Other Circuit Families

Other circuit families in the literature include *Regenerative Push-Pull Differential Logic* (RPPDL) [Partovi94], *Switched Output Differential Structure* (SODS) [Acosta95], *Current Sensing Differential Logic* (CSDL) [Park99], *Dynamic Current Mode Logic* (DyCML) [Allam01], *Subthreshold Logic* [Soeleman01], *Swing Limited Logic* (SLL) [Fahim02], and *Race Logic Architecture* (RALA) [Lee02]. It is left as an exercise to the reader to evaluate the advantages and pitfalls of these and future circuit families.

6.5 Low-power Logic Design

Given the logic families covered in this chapter, what are the particular strategies to achieve low-power dissipation? For the answers, we return to the basics introduced in Section 4.4. Commencing with dynamic power, recall that this component of power dissipation is proportional to α, the activity factor, C, the capacitance switched, f, the frequency of operation and V_{DD}^2, where V_{DD} is the supply voltage.

The first step is to choose a suitable process. Dynamic power is lowest in the most advanced (shortest channel length) manufacturing process available. Capacitance per micron of channel width is approximately 2 fF/μm in most processes, but more advanced processes use minimum-sized transistors with narrower channel widths. Moreover, the processes run at lower V_{DD}. Thus, constant field scaling provides a cubic improvement in dynamic power for a given function. Static power increases with more advanced processes because subthreshold leakage goes up as the threshold voltages decrease. In the past, dynamic power has dominated. In the present and future, low-power processes will be optimized to have higher threshold voltages than their high-performance counterparts. Advanced processes are also more expensive, so tradeoffs between power and cost may be necessary.

The most obvious parameter to adjust for low dynamic power is the supply voltage. Having made the process choice, the supply voltage can usually be further scaled, provided adequate speed and voltage margins are retained. For instance, in a process with a 3.3 V nominal V_{DD} and 0.6 V threshold voltages, V_{DD} can be scaled down to say 1.8 V ($V_{DDmin} = 3V_t$). This comes at a substantial reduction in speed, but if the critical path still meets the design target, then this approach can be taken. In particular, smaller processes are faster, offering more headroom to trade away speed for supply voltage. Depending on the workload, the voltage and clock frequency can both be adjusted to conserve power when the system does not need maximum performance. Chips may also sense their own die temperature and throttle the supply voltage or instruction issue rate when the temperature exceeds the operating envelope.

A variation on adjusting power supplies is to divide the logic into high-speed and low-power groups run from separate power supplies. Some designs have been proposed that have dual-supply rails embedded in each logic cell so the decision of what supply to use is left as a late binding routing option. Fast logic is connected to the high supply and slow logic is connected to the low supply. Level converters have to be inserted between the two styles of logic. Given that the low V_{DD} supply transistors can take the high V_{DD} on their gates, the interface between high V_{DD} gate and low V_{DD} gate can be a direct connection. For low V_{DD} to high V_{DD} the CVSL inverter/buffer circuit shown in Figure 6.68(a) can be used. The level converters add delay, power, and area, so they should be used sparingly. [Usami94] proposes *clustered voltage scaling* where the low V_{DD} cells are grouped at the end of each cycle of logic. Then, the level conversion back to high V_{DD} can be

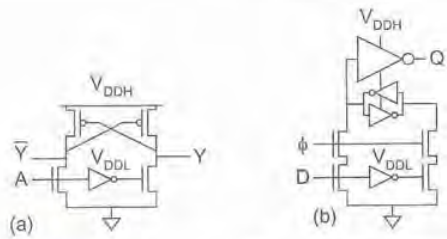


FIG 6.68 Level conversion circuits

built into the register at the end of the cycle. Figure 6.68(b) shows such a latch with integrated level conversion.

The next dynamic power parameter to attack for low-power operation is C, the switched capacitance. This consists of gate capacitance, diffusion capacitance, and wire capacitance. Good floorplanning reduces the number of long wires in a system. Good layout minimizes the size of the diffusion regions for a particular gate width. Thus, gate and diffusion capacitance are both reduced by using a small number of small transistors. Custom layout of datapaths and arrays usually provides shorter wires and lower capacitance than automatic placement and routing.

The activity factor offers more opportunity for power savings. Complementary CMOS logic has a typical activity factor of about 0.1, while dynamic logic has an activity factor of 0.5 and clocks have an activity factor of 1. Gating the clock to registers in unused units saves power not only in the registers but also in all the downstream logic because the activity factor is 0 if the inputs do not switch.

What circuit families are best for low power? Dynamic and pseudo-nMOS gates appear attractive because they eliminate the bulky pMOS transistors that account for 2/3 of the gate width in complementary CMOS logic. However, dynamic gates have a high activity factor that results in high dynamic power consumption. Pseudo-nMOS static power dissipation will dwarf the dynamic power in most applications. Pass transistor circuits have received much hype for their low-power merits. However, complementary CMOS benefits from efficient layout of simple gates, no swing restoration circuitry, single rail logic designs, and better scaling at low V_{DD}/V_t ratios; [Zimmermann97a] demonstrated that pass transistors have a higher power-delay product in most situations. Unless you are really motivated, minimum-sized complementary CMOS with a low P/N ratio is normally the best tradeoff of reliable circuit technology and low power.

Short circuit current is minimized by using sharp edges. Slow, heavily loaded nodes should be avoided. Logical effort teaches us that delay is minimized if all stages have a stage effort of about 4. If a path has slack (time to spare), the power and area can be improved by reducing the size of the largest gates. Efforts exceeding about 8 have diminishing returns and lead to excessive short circuit currents and hot electron reliability problems.

6.6 Comparison of Circuit Families

Table 6.4 summarizes the characteristics of the circuit families described in this chapter. The number of transistors required for k-input gates are listed. Differential circuit families are those that require true and complementary inputs and generate true and complementary outputs. Static power indicates that the gate may consume power while quiescent; this is often not acceptable for battery-operated devices. Circuits with rail-to-rail outputs swing between GND and V_{DD}. Dynamic nodes are those that have been precharged and may float or be only weakly held by a keeper; they are particularly sensitive to noise. Restoring logic families are those whose output logic levels are better than the input logic levels; if families are not restoring, buffers must be periodically placed between gates.

Ratioed circuits are those whose operation depends on the relative strength of nMOS and pMOS transistors; they must be sized properly for correct operation. Circuits are cascadeable if the output of a gate is a legal input to another gate of the same family without any special delayed clocking or self-timing. For example, domino gates sharing a common clock can be cascaded, but dynamic gates cannot be without violating monotonicity. Robustness characterizes the amount of care required to ensure a gate will work. Highly robust circuits like static CMOS will eventually get the right answer independent of sizing and noise, while less robust circuits are more sensitive. Undesirable characteristics are marked in blue.

A large number of circuit families have been presented in this section. A natural question is how to choose the appropriate circuit family for the application.

Static CMOS logic is the best option for the vast majority of CMOS circuits. It is noise-immune, dissipates no static power, and is fast. Highly automated tools and readily available libraries exist to synthesize, place, and route static logic. Don't overlook compound AOI and OAI gates. High fan-in static CMOS gates offer low power but have large logical effort and are best split into multiple stages of simpler gates when speed is essential.

Certain high fan-in functions are implemented much more efficiently with pseudo-nMOS or dynamic NOR gates because the logical effort is independent of the width. Examples include ROMs, PLAs, and CAMs. Pseudo-nMOS static power dissipation can be a problem for battery-operated systems, but sometimes the pMOS pull-up can be turned OFF during idle periods to save power.

Domino logic remains the technique of choice for high-speed applications, especially in high-performance microprocessors. However, it has poor noise margins and is susceptible to noise from charge sharing, coupling, leakage, and alpha particles. If you are not prepared to exhaustively simulate the gates at the circuit level with back-annotated capacitances from the layout, do not consider domino. Remember that the precharge time will rob the speed advantage over static designs in poorly designed clocking schemes (this will be discussed further in Section 7.5.1). Many novices (and pros too!) have been caught by not understanding all the problems that can arise when domino logic is used.

Pass transistors have their vocal advocates, but transmission gate logic can be viewed as an alternative way of drawing static CMOS gates with the driving stage at the output rather than the input. Of the multitude of pass-transistor circuit families that have been proposed, CPL is the most promising.

Other circuit families offer potential for niche applications (i.e., low noise generation in sensitive analog circuits), but one must be wary of pitfalls and consider carefully why so many circuit families have never seen commercial application.

Table 6.4 Comparison of circuit families

Family	nMOS	pMOS	Differential	Static Power	Rail-to-rail Output	Dynamic Nodes	Restoring	Ratioed	Cascadable	Robustness
Static CMOS	k	k	NO	NO	YES	NO	YES	NO	YES	HIGH
Pseudo-nMOS	k	1	NO	YES	NO	NO	YES	YES	YES	MEDIUM
SFPL	2k + 2	1	NO	YES	NO	NO	YES	YES	YES	MEDIUM
CVSL	2k	2	YES	NO	YES	NO	YES	NO	YES	HIGH
Dynamic	k + 1	1	NO	NO	YES	YES	YES	NO	NO	LOW
Domino	k + 2	2	NO	NO	YES	YES	YES	NO	YES	LOW
Dual-rail Domino	2k + 3	4	YES	NO	YES	YES	YES	NO	YES	LOW
CMOSTG	k	k	NO	NO	YES	NO	YES	NO	YES	HIGH
LEAP	k	2	NO	NO	YES	NO	YES	YES	YES	MEDIUM
DPL	2k	2k	YES	NO	YES	NO	YES	NO	YES	HIGH
CPL	2k	4	YES	NO	YES	NO	YES	NO	YES	MEDIUM
EEPL	2k	4	YES	NO	YES	NO	YES	NO	YES	MEDIUM
SRPL	2k	2	YES	NO	YES	NO	YES	YES	YES	LOW
DCVSPG	2k − 2	2	YES	NO	YES	NO	NO	NO	YES	MEDIUM
PPL	k	k	YES	NO	YES	NO	NO	NO	YES	LOW
DSL	2k + 2	2	YES	YES	NO	NO	YES	NO	YES	MEDIUM
CNTL	2k + 4	2	YES	YES	NO	NO	YES	NO	YES	MEDIUM
NTL	k + 1	1	NO	YES	NO	NO	YES	YES	YES	MEDIUM
SSDL	2k + 6	6	YES	YES	YES	NO	YES	NO	NO	VERY LOW
EDCL	2k + 4	3	YES	NO	YES	NO	YES	NO	NO	VERY LOW
LCDL	2k + 8	6	YES	NO	YES	NO	YES	NO	NO	VERY LOW
DCSL1	2k + 7	4	YES	NO	YES	NO	YES	NO	NO	VERY LOW
BiCMOS	2k + 1	k	NO	YES	NO	NO	YES	NO	YES	MEDIUM

6.7 Silicon-on-Insulator Circuit Design

Silicon-on-Insulator (SOI) technology has been a subject of research for decades, but has become commercially important since it was adopted by IBM for PowerPC microprocessors in 1998 [Shahidi02]. SOI is attractive because it offers potential for higher performance and lower power consumption, but also has a higher manufacturing cost and some unusual transistor behavior that complicates circuit design.

The fundamental difference between SOI and conventional bulk CMOS technology is that the transistor source, drain, and body are surrounded by insulating oxide rather than the conductive substrate or well (called the *bulk*). Using an insulator eliminates most of the

parasitic capacitance of the diffusion regions. However, it means that the body is no longer tied to GND or V_{DD} through the substrate or well. Any change in body voltage modulates V_t, leading to both advantages and complications in design.

Figure 6.69 shows a cross-section of an inverter in a SOI process. The process is similar to standard CMOS, but starts with a wafer containing a thin layer of SiO_2 buried beneath a thin single-crystal silicon layer. Section 3.4.1 discussed several ways to form this buried oxide. Shallow trench isolation is used to surround each transistor by an oxide insulator. Figure 6.70 shows a scanning electron micrograph of a 6-transistor static RAM cell in a 0.22 μm IBM SOI process.

SOI devices are categorized as partially depleted (PD) or fully depleted (FD). A depletion region empty of free carriers forms in the body beneath the gate. In FD SOI, the body is thinner than the channel depletion width, so the body charge is fixed and thus the body voltage does not change. In PD SOI, the body is thicker and its voltage can vary depending on how much charge is present. This varying body voltage in turn changes V_t through the body effect. FD SOI has been difficult to manufacture because of the thin body, so PD SOI appears to be the most promising technology.

Throughout this section we will concentrate on nMOS transistors. pMOS transistors have analogous behaviors.

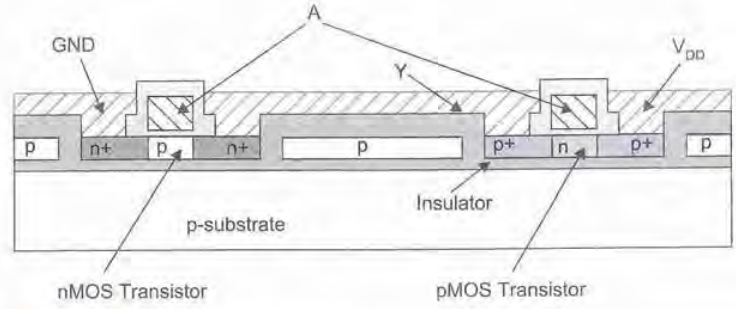


FIG 6.69 SOI inverter cross-sections

6.7.1 Floating Body Voltage

The key to understanding PD SOI is to follow the body voltage. If the body voltage were constant, the threshold voltage would be constant as well and the transistor would behave much like a conventional bulk device except that the diffusion capacitance is lower.

In PD SOI, the floating body voltage varies as it charges or discharges. Figure 6.71 illustrates the mechanisms by which charges enter into or exit from the body [Bernstein00]. There are two paths through which charge can slowly build up in the body:

- Reverse-biased drain-to-body D_{db} and possibly source-to-body D_{sb} junctions carry small diode leakage currents into the body.

- High-energy carriers cause impact ionization, creating electron-hole pairs. Some of these electrons are injected into the gate or gate oxide. (This is the mechanism for hot-electron wearout described in Section 4.8.4.) The corresponding holes accumulate in the body. This effect is most pronounced at V_{DS} above the intended operating point of devices and is relatively unimportant during normal operation. The impact ionization current into the body is modeled as a current source I_{ii}.

The charge can exit the body through two other paths:

- As the body voltage increases, the source-to-body D_{sb} junction becomes slightly forward-biased. Eventually, the charge exiting from this junction equals the charge leaking in from the drain-to-body D_{db} junction.
- A rising gate or drain capacitively couples the body upward, too. This may strongly forward-bias the source-to-body D_{sb} junction and rapidly spill charge out of the body.

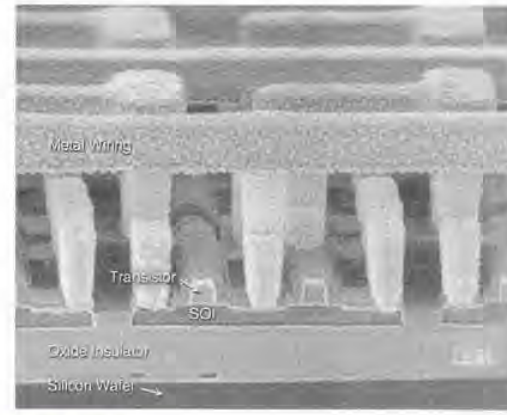


FIG 6.70 IBM SOI process electron micrograph. Courtesy of International Business Machines Corporation. Unauthorized use not permitted.

In summary, when a device is idle long enough (on the order of microseconds), the body voltage will reach equilibrium when based on the leakage currents through the source and drain junctions. When the device then begins switching, the charge may spill off the body, shifting the body voltage (and threshold voltage) significantly.

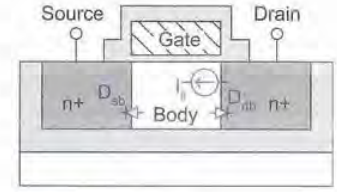


FIG 6.71 Charge paths to/from floating body

6.7.2 SOI Advantages

A major advantage of SOI is the lower diffusion capacitance. The source and drain abut against oxide on the bottom and sidewalls not facing the channel, essentially eliminating the parasitic capacitance of these sides. This results in a smaller parasitic delay and lower dynamic power consumption.

A more subtle advantage is the potential for lower threshold voltages. In bulk processes, threshold voltage varies with channel length. Hence, variations in polysilicon etching show up as variations in threshold voltage. The threshold voltage must be high enough in the worst (lowest) case to limit subthreshold leakage, so the nominal threshold voltage must be higher. In SOI processes, the threshold variations tend to be smaller. Hence, the nominal V_t can be closer to worst-case. Lower nominal V_t results in faster transistors, especially at low V_{DD}.

As discussed in Section 2.4.4, CMOS devices have a subthreshold slope of about nv_Tln10, where $v_T = kT/q$ is the thermal voltage (26 mV at room temperature) and n is process-dependent. Bulk CMOS processes typically have $n \approx 1.5$, corresponding to a subthreshold slope of 90 mV/decade. In other words, for each 90 mV decrease in V_g below V_t,

the subthreshold leakage current reduces by an order of magnitude. Misleading claims have been made suggesting SOI has $n = 1$ and thus an ideal subthreshold slope of only 60 mV/decade. IBM has found that real SOI devices actually subthreshold slopes of 75–85 mV/decade. This is better than bulk, but not as good as the hype would suggest. Double-gate MOSFETs and FINFETs discussed in Section 3.4.1 are variations on SOI transistors that offer lower subthreshold slopes because the gate surrounds the channel on more sides and thus turns the transistor off more abruptly.

Finally, SOI is immune to latchup because the insulating oxide eliminates the parasitic bipolar devices that could trigger latchup.

6.7.3 SOI Disadvantages

PD SOI suffers from the *history effect*. Changes in the body voltage modulate the threshold voltage and thus adjust gate delay. The body voltage depends on whether the device has been idle or switching, so gate delay is a function of the switching history. Overall, the elevated body voltage reduces the threshold and makes the gates faster, but the uncertainty makes circuit design more challenging. The history effect can be modeled in a simplified way by assigning different propagation and contamination delays to each gate. IBM found the history effect tends to result in about an 8% variation in gate delay, which is modest compared to the combined effects of manufacturing and environmental variations [Shahidi02].

Unfortunately, the history effect causes significant mismatches between nominally identical transistors. For example, if a sense amplifier has repeatedly read a particular input value, the threshold voltages of the differential pair will be different, introducing an offset voltage in the sense amplifier. This problem can be circumvented by adding a contact to tie the body to ground or to the source for sensitive analog circuits.

Another PD SOI problem is the presence of a parasitic bipolar transistor within each transistor. As shown in Figure 6.72, the source, body, and drain form an emitter, base, and collector of an npn bipolar transistor. In an ordinary transistor, the body is tied to a supply, but in SOI, the body/base floats. If the source and drain are both held high for an extended period of time while the gate is low, the base will float high as well through diode leakage. If the source should then be pulled low, the npn transistor will turn ON. A current I_B flows from body/base to source/emitter. This causes βI_B to flow from the drain/collector to source/emitter. The bipolar transistor gain β depends on the channel length and doping levels but can be greater than 1. Hence, a significant pulse of current can flow from drain to source when the source is pulled low even though the transistor should be OFF.

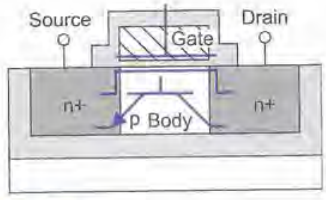


FIG 6.72 Parasitic bipolar transistor in PD SOI

This pulse of current is sometimes called *pass-gate leakage* because it commonly happens to OFF pass transistors where the source and drain are initially high and then pulled low. It is not a major problem for static circuits because the ON transistors oppose the glitch. However, it can cause malfunctions in dynamic latches and logic. Thus, dynamic nodes should use strong keepers to hold the node steady.

A third problem common to all SOI circuits is self-heating. The oxide is a good thermal insulator as well as an electrical insulator. Thus, heat dissipated in switching transistors tends to accumulate in the transistor rather than spreading rapidly into the substrate. Individual transistors dissipating large amounts of power may become substantially warmer than the die as a whole. At higher temperature they deliver less current and hence are slower. Self-heating can raise the temperature by 10°–15° C for clock buffer and I/O transistors, although the effects tend to be much smaller for logic transistors.

6.7.4 Implications for Circuit Styles

In summary, SOI is attractive for fast CMOS logic. The smaller diffusion capacitance offers a lower parasitic delay. Lower threshold voltages offer better drive current and lower gate delays. Moreover, SOI is also attractive for low-power design. The smaller diffusion capacitance reduces dynamic power consumption. The speed improvements can be traded for lower supply voltage to further reduce dynamic power. Sharper subthreshold slopes offer the opportunity for reduced static leakage current, especially in FINFETs.

Complementary static CMOS gates in PD SOI behave much like their bulk counterparts except for the delay improvement. The history effect also causes pattern-dependent variation in the gate delay.

Circuits with dynamic nodes must cope with a new noise source from pass gate leakage. In particular, dynamic latches and dynamic gates can lose the charge on the dynamic node. Figure 6.73 shows the pass gate leakage mechanism. In each case, the dynamic node X is initially high and the transistor connected to the node is OFF. The source of this transistor starts high and pulls low, turning on the parasitic bipolar transistor and partially discharging X. To overcome pass gate leakage, X should be staticized with a cross-coupled inverter pair for latches or a pMOS keeper for dynamic gates. The staticizing transistors must be relatively strong (e.g., 1/4 as strong as the normal path) to fight the leakage. The gates are slower because they must overcome the strong keepers. Dynamic gates may precharge the internal nodes to prevent pass gate leakage, but then must deal with charge sharing onto those internal nodes.

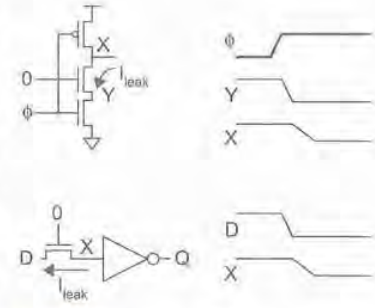


FIG 6.73 Pass gate leakage in dynamic latches and gates

Analog circuits, sense amplifiers, and other circuits that depend on matching between transistors suffer from major threshold voltage mismatches caused by the history of the floating body. They require body contacts to eliminate the mismatches by holding the body at a constant voltage. Gated clocks also have greater clock skew because the history effect makes the clock switch more slowly on the first active cycle after the clock has been disabled for an extended time.

6.7.5 Summary

In summary, Silicon-on-Insulator is attractive because it greatly reduces the source/drain diffusion capacitance, resulting in faster and power-efficient transistors. It also is immune to latchup. Partially depleted SOI is the most practical technology and also boosts drive current because the floating body leads to lower threshold voltages.

SOI design is more challenging because of the floating body effects. Gate delay becomes history-dependent because the voltage of the body depends on the previous state of the device. This complicates device modeling and delay estimation. It also contributes to mismatches between devices. In specialized applications like sense amplifiers, a body contact may be added to create a fully depleted device.

A second challenge with SOI design is pass-gate leakage. Dynamic nodes may be discharged from this leakage even when connected to OFF transistors. Strong keepers can fight the leakage to prevent errors.

Finally, the oxide surrounding SOI devices is a good thermal insulator. This leads to greater self-heating. Thus, the operating temperature of individual transistors may be up to 10°–15° C higher than that of the substrate. Self-heating reduces ON current and makes modeling more difficult.

This section only scratches the surface of a subject worthy of entire books. In particular, SOI static RAMs require special care because of pass gate leakage and floating bodies. [Bernstein00] offers a definitive treatment of partially depleted SOI circuit design and [Kuo01] surveys the literature of SOI circuits.

6.8 Pitfalls and Fallacies

Failing to plan for advances in technology

There are many advances in technology that change the relative merits of different circuit techniques. For example, interconnect delays are not improving as rapidly as gate delays, threshold drops are becoming a greater portion of the supply voltage, and leakage currents are increasing. Failing to anticipate these changes leads to inventions whose usefulness is short-lived.

A salient example is the rise and fall of BiCMOS circuits. Bipolar transistors have a higher current output per unit input capacitance (i.e., a lower logical effort) than CMOS circuits in the 0.8 μm generation, so they became popular, particularly for driving large loads. In the early 1990s, hundreds of papers were written on the subject. The Pentium and Pentium Pro processors were built using BiCMOS processes. Investors poured at least $40 million into a startup company called Exponential, which sought to build a fast PowerPC processor in a BiCMOS process.

Unfortunately, technology scaling works against BiCMOS because of the faster CMOS transistors, lower supply voltages, and larger numbers of transistors on a chip. The relative benefit of bipolar transistors over fine-geometry CMOS decreased. As discussed in Section 6.4.3, the V_{be} drop became an unacceptable fraction of the power supply. Finally, the static power consumption caused by bipolar base currents limits the number of bipolar transistors that can be used.

The Pentium II was based on the Pentium Pro design, but the bipolar transistors had to be removed because they no longer provided advantages in the 0.35 μm generation. Despite a talented engineering team, Exponential failed entirely, ultimately producing a processor that lacked compelling performance advantages and dissipated far more power than anything else on the market [Maier97].

Comparing a well-tuned new circuit to a poor example of existing practice

A time-honored way to make a new invention look good is to tune it as well as possible and compare it to an untuned strawman held up as an example of "existing practice." For example, [Zimmermann97a] points out

that most papers finding pass-transistor adders faster than static CMOS adders use 40-transistor static adder cells rather than the faster and smaller 28-transistor cells (Figure 10.4).

Ignoring driver resistance when characterizing pass-transistor circuits

Another way to make pass-transistor circuit families look about twice as fast as they really are is to drive diffusion inputs with a voltage source rather than with the output stage of the previous gate.

Reporting only part of the delay of a circuit

Clocked circuits all have a setup time and a clock-to-output delay. A good way to make clocked circuits look fast is to only report the clock-to-output delay. This is particularly common for the sense-amplifier logic families.

Making outrageous claims about performance

Many published papers have made outrageous performance claims. For example, while comparing full adder designs, some authors have found that DSL and dual-rail domino are 8–10x faster than static CMOS. Neither statement is anywhere close to what designers see in practice; for example, [Ng96] finds that an 8x8 multiplier built from DSL is 1.5x faster and one built from dual-rail domino is 2x faster than static CMOS.

In general, "there ain't no such thing as a free lunch" in circuit design. CMOS design is a fairly mature field and designers are not stupid (or at least not all designers are stupid all the time), so if some new invention seems too good to be true, it probably is. Beware of papers that push the advantages of a new invention without disclosing the inevitable tradeoffs. The tradeoffs may be acceptable, but they must be understood.

Building circuits without adequate verification tools

It is impractical to manually verify circuits on chips that have many millions (soon billions) of transistors. Automated verification tools should check for any pitfalls common to widely used circuit families. If you cannot afford to buy or write appropriate tools, stick with robust static CMOS logic.

6.9 Historical Perspective

Despite rumors of their demise, dynamic circuits remain essential to high-performance digital design. Dynamic circuits predate the widespread use of CMOS. In an nMOS process, pMOS transistors were not available to build complementary gates. One strategy was to build ratioed gates, as shown in Figure 6.74. Conceptually, the ratioed gate consists of an nMOS pull-down network and some pull-up device. A resistor would be simple, but large resistors consume a large layout area in typical MOS processes. Another technique is to use an nMOS transistor with the gate tied to V_{GG}. If $V_{GG} = V_{DD}$, the nMOS transistor will only pull up to $V_{DD} - V_t$. Worse yet, the threshold is increased by the body effect. Thus, using $V_{GG} > V_{DD}$ was attractive. To eliminate this extra supply voltage, some nMOS processes offered *depletion mode* transistors. These transistors, indicated with the thick bar, are identical to

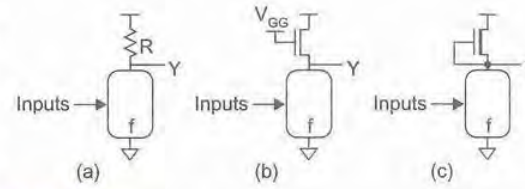


FIG 6.74 nMOS ratioed gates

ordinary *enhancement mode* transistors except that an extra ion implantation was performed to create a negative threshold voltage. The depletion mode pull-ups have their gate wired to the source so $V_{gs} = 0$ and the transistor is always weakly ON.

All the ratioed gates consume static power whenever the outputs are low. The speed is proportional to the RC product, so fast gates need low-resistance pull-ups, exacerbating the power problem. An alternative was to use dynamic gates. The classic MOS textbook of the early 1970s [Penney72] devotes 29 pages to describing a multitude of dynamic gate configurations. Unfortunately, dynamic gates suffer from the monotonicity problem, so each phase of logic may contain only one gate. Phases were separated using nMOS pass transistors that behaved as dynamic latches. Figure 6.75 shows an approach using two-phase nonoverlapping clocks. Each gate precharges in one phase while the subsequent latch is opaque. It then evaluates while making the latch transparent. This approach is prone to charge-sharing noise when the latch opens and precharge only rises to $V_{DD} - V_t$. Numerous four-phase clocking techniques were also developed.

With the advent of CMOS technology, dynamic logic lost its advantage of power consumption. However, chip space was at a premium and dynamic gates could eliminate most of the pMOS transistors to save area. Domino gates were developed at Bell Labs for a 32-bit adder in the BELLMAC-32A microprocessor to solve problems of both area and speed [Krambeck82, Shoji82]. Domino allows multiple noninverting gates to be cascaded in a single phase.

High-performance microprocessors have boosted clock speeds faster than simple process improvement would allow, so the number of gate delays per cycle has shrunk. The DEC Alpha microprocessors pioneered this trend through the 1990s [Gronowski98] and most other CPUs have followed. Domino circuits have become crucial to achieving these fast cycle times. IBM is a notable exception, relying almost exclusively on complementary CMOS logic with fast time to market in cutting-edge processes to deliver adequate mainframe processor performance [Curran02].

Some older domino designs leave out the keeper to save area and gain a slight performance advantage. This has become more difficult as leakage and coupling noise have increased with process scaling. The 0.35 μm Alpha 21164 was one of the last designs to have no keeper (and to use dynamic latches). Its fully dynamic operation gave advantages in both speed and area, but during test it had a minimum operating frequency of 20 MHz

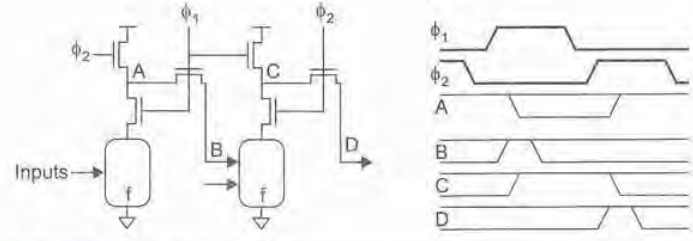


FIG 6.75 nMOS dynamic gates

to retain state. In the Alpha 21264, leakage current had increased to the point that keepers were essential. Modern designs always need keepers. As an interesting aside, the Alpha microprocessors also did not use scan latches because scan cost area and a small amount of performance. This proved unfortunate on the Alpha 21264, which was difficult to debug because of the limited observability into the processor state. Now virtually all design methodologies require scan capability in the latches or registers, as will be discussed in Section 9.6.

Contemporary domino design involves substantial overhead besides the keepers. Dual-rail domino is often necessary for nonmonotonic functions. Secondary precharge devices are used to prevent charge sharing. Multiple clock phases are used to tolerate clock skew and each phase requires a clock buffer and wire. Dynamic inputs and outputs must be shielded or spaced far from aggressors to avoid coupling noise problems. Considering all these factors, domino no longer has significant area advantages over complementary CMOS gates for general-purpose logic. However, static RAMs and register files discussed in Chapter 11 can be viewed as special applications of domino that do achieve high density.

Summary

Circuit delay is related to the $(C/I)\Delta V$ product of gates. This chapter explored alternative combinational circuit structures to improve the C/I ratio or respond to smaller voltage swings. Many of these techniques trade higher power consumption and/or lower noise margins for better delay. While complementary CMOS circuits are quite robust, the alternative circuit families have pitfalls that must be understood and managed. The chapter also examined the layout of cells and layout optimizations to reduce parasitic capacitance.

Three of the commonly used alternatives to complementary CMOS are domino, pseudo-nMOS, and pass transistor logic. Each attempts to reduce the input capacitance by performing logic mostly through nMOS transistors.

Pseudo-nMOS replaces the pMOS pull-up network with a single weak pMOS transistor that is always ON. The pMOS transistor dissipates static power when the output is low. If it is too weak, the rising transition is slow. If it is too strong, V_{OL} is too high and the power consumption increases. When the static power consumption is tolerable, pseudo-nMOS gates work well for wide NOR functions.

Dynamic gates resemble pseudo-nMOS, but use a clocked pMOS transistor in place of the weak pull-up. When the clock is low, the gates precharge high. When the clock rises, the gates evaluate, pulling the output low or leaving it floating high. The input of a dynamic gate must be monotonically rising while the gate is in evaluation, but the output monotonically falls. Domino gates consist of a dynamic gate followed by an inverting static gate and produce monotonically rising outputs. Therefore, domino gates can be cascaded, but only compute noninverting functions. Dual-rail domino accepts true and complementary inputs and produces true and complementary outputs to provide any logic function at the expense of larger gates and twice as many wires. Dynamic gates are also sensitive to noise because V_{IL} is close to the threshold voltage V_t and the output floats.

Major noise sources include charge sharing, leakage, and coupling. Therefore, domino circuits typically use secondary precharge transistors, keepers, and shielded or carefully routed interconnect. The high-activity factors of the clock and dynamic node make domino power hungry. Despite all of these challenges, domino offers a 1.5–2x speedup over static CMOS, giving it a compelling advantage for the critical paths of high-performance systems.

Pass-transistor circuits use inputs that drive the diffusion inputs as well as the gates of transistors. Many pass-transistor techniques have been explored and Complementary Pass Transistor logic has proven to be one of the most effective. This dual-rail technique uses networks of nMOS transistors to compute true and complementary logic functions. The nMOS transistors only pull up to $V_{DD} - V_t$, so cross-coupled pMOS transistors boost the output to full-rail levels. Some designers find that pass-transistor circuits are faster and smaller for functions such as XOR, full adders, and multiplexers that are clumsy to implement in static CMOS. Because of the threshold drop, the circuits do not scale well as V_{DD}/V_t decreases.

As designer productivity, measured in gates per designer per week, must increase exponentially, static CMOS circuits become even more attractive because they are so simple and robust.

Exercises

6.1 Design a fast 6-input OR gate in each of the following circuit families. Sketch an implementation using two stages of logic (e.g., NOR6 + INV, NOR3 + NAND2, etc.). Label each gate with the width of the pMOS and nMOS transistors. Each input can drive no more than 30 λ of transistor width. The output must drive a 60/30 inverter (i.e., an inverter with a 60 λ-wide pMOS and 30 λ-wide nMOS transistor). Use logical effort to choose the topology and size for least average delay. Estimate this delay using logical effort. When estimating parasitic delays, count only the diffusion capacitance on the output node.

a) static CMOS

b) pseudo-nMOS with pMOS transistors 1/4 the strength of the pull-down stack

c) domino (a footed dynamic gate followed by a HI-skew inverter); only optimize the delay from rising input to rising output

6.2 Simulate each gate you designed in Exercise 6.1. Determine the average delay (or rising delay for the domino design). Logical effort is only an approximation. Tweak the transistor sizes to improve the delay. How much improvement can you obtain?

6.3 Sketch a schematic for a 12-input OR gate built from NANDs and NORs of no more than 3 inputs each.

6.4 Design a static CMOS circuit to compute $F = (A + B)(C + D)$ with least delay. Each input can present a maximum of 30 λ of transistor width. The output must drive a load equivalent to 500 λ of transistor width. Choose transistor sizes to achieve least delay and estimate this delay in τ.

6.5 Figure 6.76 shows two series transistors modeling the pull-down network of 2-input NAND gate.

a) Plot I vs. A using ideal transistor models for $0 \le A \le 1$, $B = Y = 1$, $V_t = 0$, $\beta = 1$. On the same axes, plot I vs. B for $0 \le B \le 1$, $A = 1$. *Hint*: you will need to solve for x; this can be done best numerically.

b) Using your results from (a), explain why the inner input of a 2-input NAND gate has a slightly greater logical effort than the outer input.

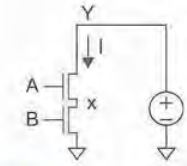


FIG 6.76 Current in series transistors

6.6 What is the logical effort of an OR-AND-INVERT gate at either of the OR terminals? At the AND terminal? What is the parasitic delay if only diffusion capacitance on the output is counted?

6.7 Simulate a 3-input NOR gate in your process. Determine the logical effort and parasitic delay from each input.

6.8 Using the data sheet from Figure 4.25, find the rising and falling logical effort and parasitic delay of the *X1* 2-input NAND gate from the A input.

6.9 Repeat Exercise 6.8 for the B input. Explain why the results are different for the different inputs.

6.10 Sketch HI-skew and LO-skew 3-input NAND and NOR gates. What are the logical efforts of each gate on its critical transition?

6.11 Derive a formula for g_u g_d and g_{avg} for HI-skew and LO-skew k-input NAND gates with a skew factor of $s < 1$ (i.e., the noncritical transistor is s times normal size) as a function of s and k.

6.12 Design an asymmetric 3-input NOR gate that favors a critical input over the other two. Choose transistor sizes so the logical effort on the critical input is 1.5. What is the logical effort of the noncritical inputs?

6.13 Prove that the P/N ratio that gives lowest average delay in a logic gate is the square root of the ratio that gives equal rise and fall delays.

6.14 Let $\rho(g, p)$ be the best stage effort of a path if one is free to add extra buffers with a parasitic delay p and logical effort g. For example, Section 4.3.3 showed that $\rho(1, 1) = 3.59$. It is easy to make a plot of $\rho(1, p)$ by solving EQ (4.19) numerically; this gives the best stage effort of static CMOS circuits where the inverter has a parasitic delay of p. Prove the following result, which is useful for determining the best stage effort of domino circuits where buffers have lower logical efforts:

$$\rho(g,p) = g\rho(1,\frac{p}{g})$$

6.15 Simulate a fanout-of-4 inverter. Use a unit-sized nMOS transistor. How wide must the pMOS transistor be to achieve equal rising and falling delays? What is the delay? How wide must the pMOS transistor be to achieve minimum average delay? What is the delay? How much faster is the average delay?

6.16 Many standard cell libraries choose a *P/N* ratio for an inverter in between that which would give equal rising and falling delays and that which would give minimum average delay. Why is this done?

6.17 A static CMOS NOR gate uses 4 transistors, while a pseudo-nMOS NOR gate uses only 3. Unfortunately, the pseudo-nMOS output does not swing rail to rail. If both the inputs and their complements are available, it is possible to build a 3-transistor NOR that swings rail to rail without using any dynamic nodes. Show how to do it. Explain any drawbacks of your circuit.

6.18 Sketch pseudo-nMOS 3-input NAND and NOR gates. Label the transistor widths. What are the rising, falling, and average logical efforts of each gate?

6.19 Sketch a pseudo-nMOS gate that implements the function

$$F = \overline{A(B+C+D)+E \cdot F \cdot G}.$$

6.20 Design an 8-input AND gate with an electrical effort of 6 using pseudo-nMOS logic. If the parasitic delay of an n-input pseudo-nMOS NOR gate is $(4n + 2)/9$, what is the path delay?

6.21 Simulate a pseudo-nMOS inverter in which the pMOS transistor is half the width of the nMOS transistor. What are the rising, falling, and average logical efforts? What is V_{OL}?

6.22 Repeat Exercise 6.21 in the FS and SF process corners.

6.23 Sketch a 3-input symmetric NOR gate. Size the inverters so that the pull-down is four times as strong as the net worst-case pull-up. Label the transistor widths. Estimate the rising, falling, and average logical efforts. How do they compare to a static CMOS 3-input NOR gate?

6.24 Sketch a 2-input symmetric NAND gate. Size the inverters so that the pull-down is four times as strong as the net worst-case pull-up. Label the transistor widths. Estimate the rising, falling, and average logical efforts. How do they compare to a static CMOS 3-input NAND gate?

6.25 Compare the average delays of a 2, 4, 8, and 16-input pseudo-NMOS and SFPL NOR gate driving a fanout of 4 identical gates.

6.26 Sketch a 3-input CVSL OR/NOR gate.

6.27 Sketch dynamic footed and unfooted 3-input NAND and NOR gates. Label the transistor widths. What is the logical effort of each gate?

6.28 Sketch a 3-input dual-rail domino OR/NOR gate.

6.29 Sketch a 3-input dual-rail domino majority/minority gate. This is often used in domino full adder cells. Recall that the majority function is true if more than half of the inputs are true.

6.30 Compare a standard keeper with the noise tolerant precharge device. Larger pMOS transistors result in a higher V_{IL} (and thus better noise margins) but more delay. Simulate a 2-input footed NAND gate and plot V_{IL} vs. delay for various sizes of keepers and noise tolerant precharge transistors.

6.31 Design a 4-input footed dynamic NAND gate driving an electrical effort of 1. Estimate the worst charge sharing noise as a fraction of V_{DD} assuming that diffusion capacitance on uncontacted nodes is about half of gate capacitance and on contacted nodes it equals gate capacitance.

6.32 Repeat Exercise 6.31, generating a graph of charge sharing noise vs. electrical effort for $h = 0, 1, 2, 4$, and 8.

6.33 Repeat Exercise 6.31 if a small secondary precharge transistor is added on one of the internal nodes.

6.34 Perform a simulation of your circuits from Exercise 6.31. Explain any discrepancies.

6.35 Design a domino circuit to compute $F = (A + B)(C + D)$ as fast as possible. Each input may present a maximum of 30 λ of transistor width. The output must drive a load equivalent to 500 λ of transistor width. Choose transistor sizes to achieve least delay and estimate this delay in τ.

6.36 Redesign the memory decoder from Section 4.3.4 using footed domino logic. You can assume you have both true and complementary monotonic inputs available, each capable of driving 5 unit transistors. Label gate sizes and estimate the delay.

6.37 Sketch an NP Domino 8-input AND circuit.

6.38 Sketch a 4:1 multiplexer. You are given four data signals *D0*, *D1*, *D2*, and *D3*, and two select signals, *S0* and *S1*. How many transistors does each design require?

a) Use only static CMOS logic gates.

b) Use a combination of logic gates and transmission gates.

6.39 Sketch 3-input XOR functions using each of the following circuit techniques:

a) static CMOS

b) pseudo-nMOS

c) dual-rail domino

d) CPL

e) EEPL

f) DCVSPG

g) SRPL
h) PPL
i) DPL
j) LEAP

6.40 Repeat Exercise 6.39 for a 2-input NAND gate.

6.41 Design sense-amplifier gates using each of the following circuit families to compute an 8-input XOR function in a single gate: SSDL, ECDL, LCDL, DCSL1, DCSL2, DCSL3. Each true or complementary input can drive no more than 24 λ of transistor width. Each output must drive a 32/16 λ inverter. Simulate each circuit to determine the setup time and clock-to-out delays.

6.42 Figure 6.77 shows a Switched Output Differential Structure (SODS) gate. Explain how the gate operates and sketch waveforms for the gate acting as an inverter/buffer. Comment on the strengths and weaknesses of the circuit family.

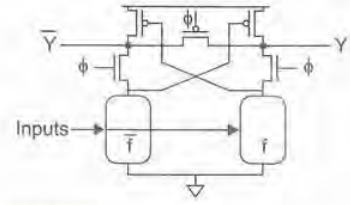


FIG 6.77 SODS

6.43 Choose one of the circuit families (besides SODS, Exercise 6.42) mentioned in Section 6.4.4 or published in a recent paper. Critically evaluate the original paper in which the circuit was proposed. Sketch an inverter or buffer and explain how it operates, including appropriate waveforms. What are the strengths of the circuit family? If you were the circuit manager choosing design styles for a large chip, what concerns might you have about the circuit family?



Sequential Circuit Design

7

7.1 Introduction

The previous chapter addressed *combinational* circuits in which the output is a function of the current inputs. This chapter discusses *sequential* circuits in which the output depends on previous as well as current inputs; such circuits are said to have *state*. Finite state machines and pipelines are two important examples of sequential circuits.

Sequential circuits are usually designed with flip-flops or latches, which are sometimes called *memory elements*, that hold data called *tokens*. The purpose of these elements is not really memory; instead, it is to enforce sequence, to distinguish the *current* token from the *previous* or *next* token. Therefore, we will call them *sequencing elements* [Harris01a]. Without sequencing elements, the next token might catch up with the previous token, garbling both. Sequencing elements delay tokens that arrive too early, preventing them from catching up with previous tokens. Unfortunately, they inevitably add some delay to tokens that are already critical, decreasing the performance of the system. This extra delay is called *sequencing overhead*.

This chapter considers sequencing for both static and dynamic circuits. *Static circuits* refer to gates that have no clock input, such as complementary CMOS, pseudo-nMOS, or pass transistor logic. *Dynamic circuits* refer to gates that have a clock input, especially domino logic. To complicate terminology, sequencing elements themselves can be either static or dynamic. A sequencing element with *static storage* employs some sort of feedback to retain its output value indefinitely. An element with *dynamic storage* generally maintains its value as charge on a capacitor that will leak away if not refreshed for a long period of time. The choices of static or dynamic for gates and for sequencing elements can be independent.

Sections 7.2–7.4 explore sequencing elements for static circuits, particularly flip-flops, 2-phase transparent latches, and pulsed latches. Section 7.5 delves into a variety of ways to sequence dynamic circuits. A periodic clock is commonly used to indicate the timing of a sequence. Section 7.6 describes how external signals can be synchronized to the clock and analyzes the risks of synchronizer failure. Wave pipelining is discussed in Section 7.7. Clock generation and distribution will be examined further in Section 12.5.

The choice of sequencing strategy is intimately tied to the design flow that is being used by an organization. Thus, it is important before departing on a design direction to ensure that all phases of design capture, synthesis, and verification can be accommodated.

This includes such aspects as cell libraries (Are the latch or flip-flop circuits and models available?); tools such as timing analyzers (Can timing closure be achieved easily?); and automatic test generation (Can self-test elements be inserted easily?).

7.2 Sequencing Static Circuits

Recall from Section 1.4.9 that *latches* and *flip-flops* are the two most commonly used sequencing elements. Both have three terminals: data input (D), clock (clk), and data output (Q). The latch is transparent when the clock is high and opaque when the clock is low; in other words, when the clock is high, D flows through to Q as if the latch were just a buffer, but when the clock is low, the latch holds its present Q output even if D changes. The flip-flop is an edge-triggered device that copies D to Q on the rising edge of the clock and ignores D at all other times. These are illustrated in Figure 7.1. The unknown state of Q before the first rising clock edge is indicated by the pair of lines at both low and high levels.

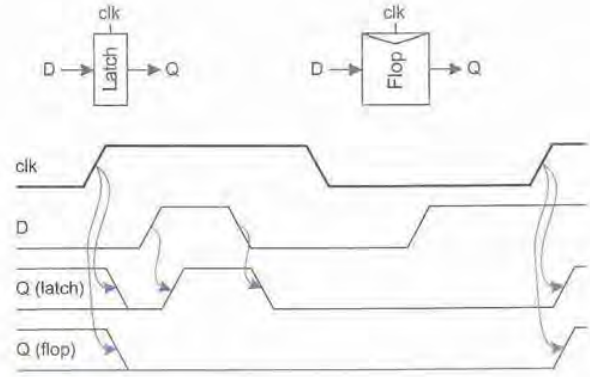


FIG 7.1 Latches and flip-flops

This section explores the three most widely used methods of sequencing static circuits with these elements: flip-flops, 2-phase transparent latches, and pulsed latches [Unger86]. An ideal sequencing methodology would introduce no sequencing overhead, allow sequencing elements back-to-back with no logic in between, grant the designer flexibility in balancing the amount of logic in each clock cycle, tolerate moderate amounts of clock skew without degrading performance, and consume zero area and power. We will compare these methods and explore the tradeoffs they offer. We will also examine a number of transistor-level circuit implementations of each element.

7.2.1 Sequencing Methods

Figure 7.2 illustrates three methods of sequencing blocks of combinational logic. In each case, the clock waveforms, sequencing elements, and combinational logic are shown. The horizontal axis corresponds to the time at which a token reaches a point in the circuit. For example, the token is captured in the first flip-flop on the first rising edge of the clock. It propagates through the combinational logic and reaches the second flip-flop on the second rising edge of the clock. The dashed vertical lines indicate the boundary between one clock cycle and the next. The clock period is T_c. In a 2-phase system, the phases may be separated by $t_{\text{nonoverlap}}$. In a pulsed system, the pulse width is t_{pw}.

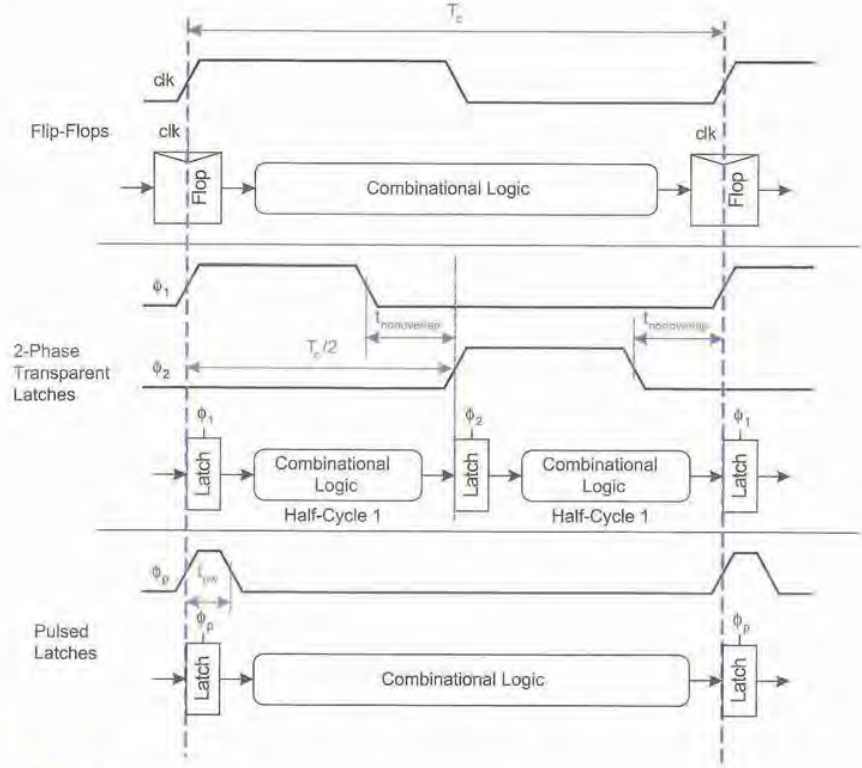


FIG 7.2 Static sequencing methods

Flip-flop-based systems use one flip-flop on each cycle boundary. Tokens advance from one cycle to the next on the rising edge. If a token arrives too early, it waits at the flip-flop until the next cycle. Recall that the flip-flop can be viewed as a pair of back-to-back

latches using *clk* and its complement, as shown in Figure 7.3. If we separate the latches, we can divide the full cycle of combinational logic into two phases, sometimes called *half-cycles*. The two latch clocks are often called ϕ_1 and ϕ_2. They may correspond to *clk* and its complement $\overline{clk}$ or may be nonoverlapping ($t_{nonoverlap} > 0$). At any given time, at least one clock is low and the corresponding latch is opaque, preventing one token from catching up with another. The two latches behave in much the same manner as two watertight gates in a canal lock [Mead80]. Pulsed latch systems eliminate one of the latches from each cycle and apply a brief pulse to the remaining latch. If the pulse is shorter than the delay through the combinational logic, we can still expect that a token will only advance through one clock cycle on each pulse.

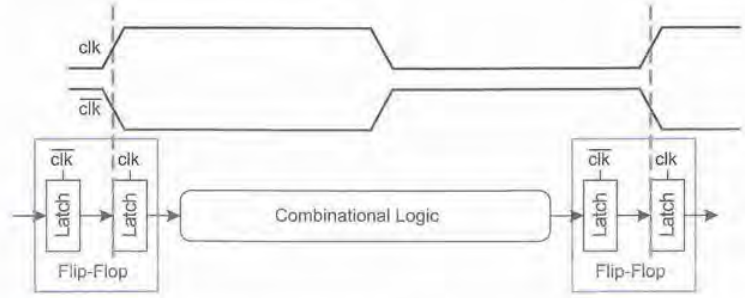


FIG 7.3 Flip-flop viewed as back-to-back latch pair

Table 7.1 defines the delays and timing constraints of the combinational logic and sequencing elements. These delays may differ significantly for rising and falling transitions and can be distinguished with an *r* or *f* suffix. For brevity, we will use the overall maximum and minimum.

Table 7.1 Sequencing element timing notation

Term	Name
t_{pd}	Logic Propagation Delay
t_{cd}	Logic Contamination Delay
t_{pcq}	Latch/Flop Clock-to-Q Propagation Delay
t_{ccq}	Latch/Flop Clock-to-Q Contamination Delay
t_{pdq}	Latch D-to-Q Propagation Delay
t_{cdq}	Latch D-to-Q Contamination Delay
t_{setup}	Latch/Flop Setup Time
t_{hold}	Latch/Flop Hold Time

Figure 7.4 illustrates these delays in a *timing diagram*. In a timing diagram, the horizontal axis indicates time and the vertical axis indicates logic level. A single line indicates that a signal is high or low at that time. A pair of lines indicates that a signal is stable but that we don't care about its value. Criss-crossed lines indicate that the signal might change at that time. A pair of lines with cross-hatching indicates that the signal may change once or more over an interval of time.

Figure 7.4(a) shows the response of combinational logic to the input A changing from one arbitrary value to another. The output Y cannot change instantaneously. After the contamination delay t_{cd}, Y may begin to change or *glitch*. After the propagation delay t_{pd}, Y must have settled to a final value. The contamination delay and propagation delay may be very different because of multiple paths through the combinational logic. Figure 7.4(b) shows the response of a flip-flop. The data input must be stable for some window around the rising edge of the flop if it is to be reliably sampled. Specifically, the input D must have settled by some *setup time* t_{setup} before the rising edge of *clk* and should not change again until a *hold time* t_{hold} after the clock edge. The output begins to change after a *clock-to-Q contamination delay* t_{ccq} and completely settles after a *clock-to-Q propagation delay* t_{pcq}. Figure 7.4(c) shows the response of a latch. Now the input D must set up and hold around the falling edge that defines the end of the sampling period. The output initially changes t_{ccq} after the latch becomes transparent on the rising edge of the clock and settles by t_{pcq}. While the latch is transparent, the output will continue to track the input after some D-to-Q delay t_{cdq} and t_{pdq}. Section 7.4.4 discusses how to measure the setup and hold times and propagation delays in simulation.

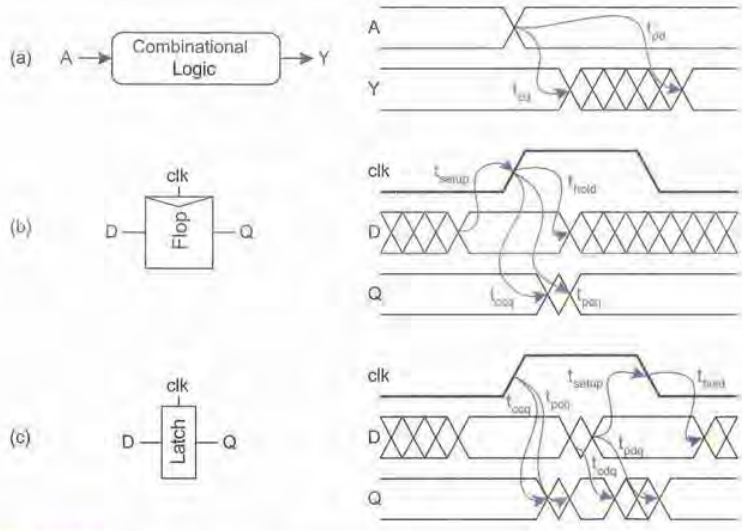


FIG 7.4 Timing diagrams

7.2.2 Max-Delay Constraints

Ideally, the entire clock cycle would be available for computations in the combinational logic. Of course, the sequencing overhead of the latches or flip-flops cuts into this time. If the combinational logic delay is too great, the receiving element will miss its setup time and sample the wrong value. This is called a *setup time failure* or *max-delay failure*. It can be solved by redesigning the logic to be faster or by increasing the clock period. This section computes the actual time available for logic and the sequencing overhead of each of our favorite sequencing elements: flip-flops, two-phase latches, and pulsed latches.

Figure 7.5 shows the max-delay timing constraints on a path from one flip-flop to the next, assuming ideal clocks with no skew. The path begins with the rising edge of the clock triggering *F1*. The data must propagate to the output of the flip-flop *Q1* and through the combinational logic to *D2*, setting up at *F2* before the next rising clock edge. This implies that the clock period must be at least

$$T_c \geq t_{pcq} + t_{pd} + t_{\text{setup}} \tag{7.1}$$

Alternatively, we can solve for the maximum allowable logic delay, which is simply the cycle time less the sequencing overhead introduced by the propagation delay and setup time of the flip-flop.

$$t_{pd} \leq T_c - \underbrace{\left(t_{\text{setup}} + t_{pcq}\right)}_{\text{sequencing overhead}} \tag{7.2}$$

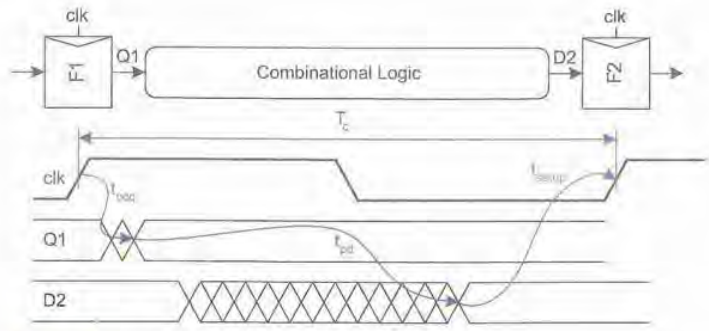


FIG 7.5 Flip-flop max-delay constraint

Example

The Arithmetic/Logic Unit (ALU) *self-bypass path* limits the clock frequency of some pipelined microprocessors. For example, the Integer Execution Unit (IEU) of the Itanium 2 contains self-bypass paths for six separate ALUs, as shown in Figure 7.6(a) [Fetzer02]. The path for one of the ALUs begins at registers containing the inputs to an adder, as shown in Figure 7.6(b). The adder must compute the sum (or difference, for subtraction). A *result multiplexer* chooses between this sum, the output of the logic unit, and the output of the shifter. Then a series of *bypass multiplexers* selects the inputs to the ALU for the next cycle. The early bypass multiplexer chooses among results of ALUs from previous cycles and is not on the critical path. The 8:1 middle bypass multiplexer chooses a result from any of the six ALUs, the early bypass mux, or the register file. The 4:1 late bypass multiplexer chooses a result from either of two results returning from the data cache, the middle bypass mux result, or the immediate operand specified by the next instruction. The late bypass mux output is driven back to the ALU to use on the next cycle. Because the six ALUs and the bypass multiplexers occupy a significant amount of area, the critical path also involves 2 mm wires from the result mux to middle bypass mux and from the middle bypass mux back to the late bypass mux. (Note: In the Itanium 2, the ALU self-bypass path is built from four-phase skew-tolerant domino circuits. For the purposes of these examples, we will hypothesize instead that it is built from static logic and flip-flops or latches.)

For our example, the propagation delays and contamination delays of the path are given in Table 7.2. Suppose the registers are built from flip-flops with a setup time of 62 ps, hold time of −10 ps, propagation delay of 90 ps, and contamination delay of 75 ps. Calculate the minimum cycle time T_c at which the ALU self-bypass path will operate correctly.

Table 7.2 Combinational logic delays

Element	Propagation Delay	Contamination Delay
Adder	590 ps	100 ps
Result Mux	60 ps	35 ps
Early Bypass Mux	110 ps	95 ps
Middle Bypass Mux	80 ps	55 ps
Late Bypass Mux	70 ps	45 ps
2-mm wire	100 ps	65 ps

continued

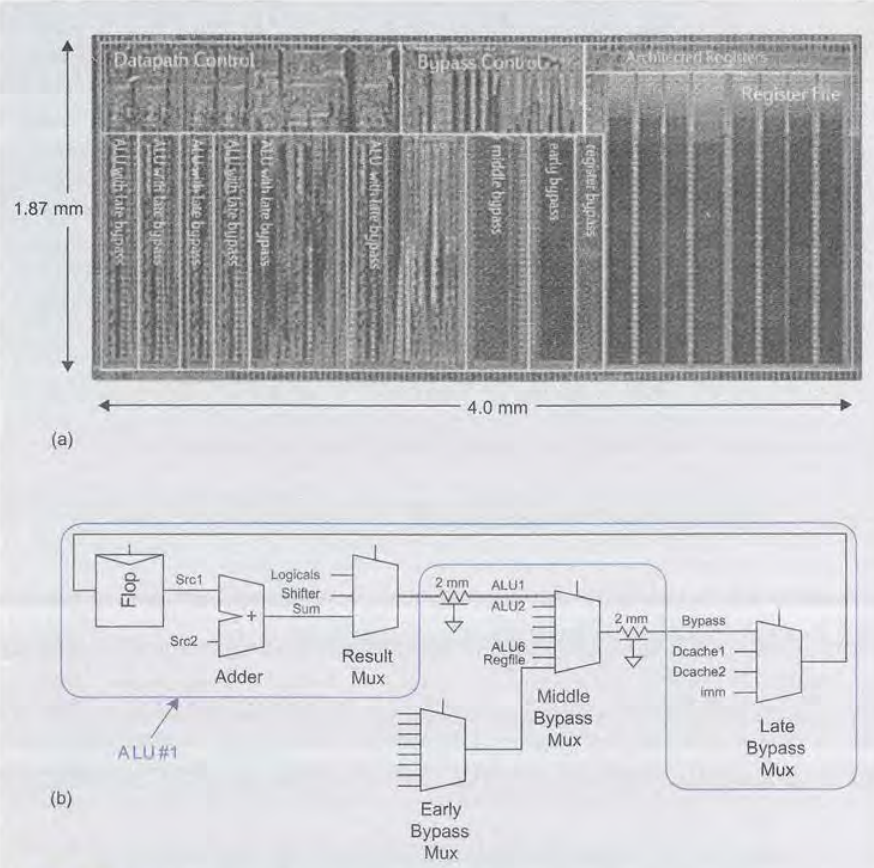


FIG 7.6 Itanium 2 ALU self-bypass path. (a) © IEEE 2002.

Solution: The critical path involves propagation delays through the adder (590 ps), result mux (60 ps), middle bypass mux (80 ps), late bypass mux (70 ps), and two 2-mm wires (100 ps each), for a total of t_{pd} = 1000 ps. According to EQ (7.1), the cycle time T_c must be at least 90 + 1000 + 62 = 1152 ps.

Figure 7.7 shows the analogous constraints on a path using two-phase transparent latches. Let us assume that data *D1* arrives at *L1* while the latch is transparent (ϕ_1 high). The data propagates through *L1*, the first block of combinational logic, *L2*, and the second block of combinational logic. Technically, *D3* could arrive as late as a setup time

before the falling edge of ϕ_1 and still be captured correctly by *L3*. To be fair, we will insist that *D3* nominally arrive no more than one clock period after *D1* because, in the long run, it is impossible for every single-cycle path in a design to consume more than a full clock period. Certain paths may take longer if other paths take less time; this technique is called *time borrowing* and will be addressed in Section 7.2.4. Assuming the path takes no more than a cycle, we see the cycle time must be

$$T_c \geq t_{pdq1} + t_{pd1} + t_{pdq2} + t_{pd2} \tag{7.3}$$

Once again, we can solve for the maximum logic delay, which is the sum of the logic delays through each of the two phases. The sequencing overhead is the two latch propagation delays. Notice that the nonoverlap between clocks does not degrade performance in the latch-based system because data continues to propagate through the combinational logic between latches even while both clocks are low. Realizing that a flip-flop can be made from two latches whose delays determine the flop propagation delay and setup time, we see EQ (7.4) is closely analogous to EQ (7.2).

$$t_{pd} = t_{pd1} + t_{pd2} \leq T_c - \underbrace{\left(2t_{pdq}\right)}_{\text{sequencing overhead}} \tag{7.4}$$

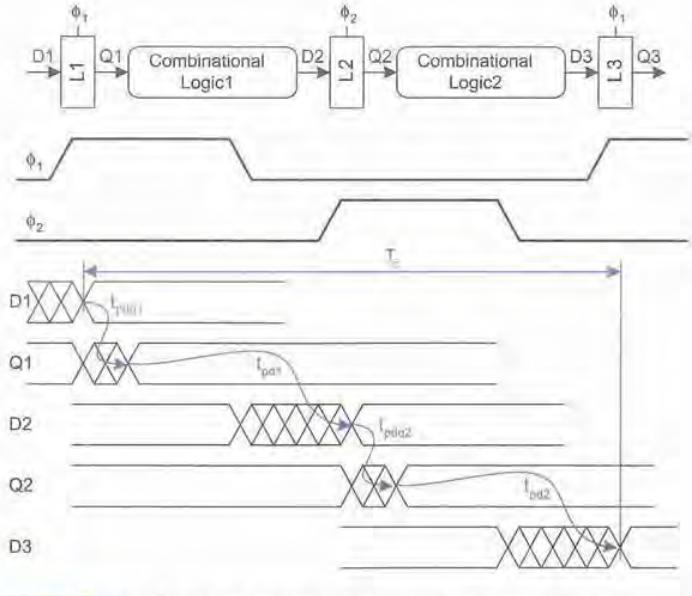


FIG 7.7 Two-phase latch max-delay constraint

If the pulse is wide enough, the max-delay constraint for pulsed latches is similar to that of two-phase latches except that only one latch is in the critical path, as shown in Figure 7.8(a). However, if the pulse is narrower than the setup time, the data must set up before the pulse rises, as shown in Figure 7.8(b). Combining these two cases gives

$$T_c \geq \max\left(t_{pdq} + t_{pd}, t_{pcq} + t_{pd} + t_{\text{setup}} - t_{pw}\right) \tag{7.5}$$

Solving for the maximum logic delay shows that the sequencing overhead is just one latch delay if the pulse is wide enough to hide the setup time

$$t_{pd} \leq T_c - \underbrace{\max\left(t_{pdq}, t_{pcq} + t_{\text{setup}} - t_{pw}\right)}_{\text{sequencing overhead}} \tag{7.6}$$

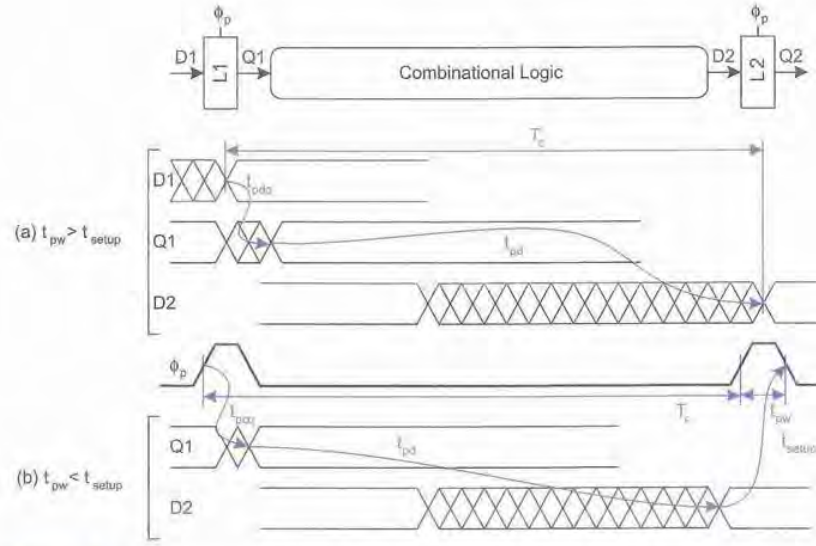


FIG 7.8 Pulsed latch max-delay constraint

7.2.3 Min-delay Constraints

Ideally, sequencing elements can be placed back to back without intervening combinational logic and still function correctly. For example, a pipeline can use back-to-back registers to sequence along an instruction opcode without modifying it. However, if the hold time is large and the contamination delay is small, data can incorrectly propagate through two successive elements on one clock edge, corrupting the state of the system. This is

Example

Recompute the ALU self-bypass path cycle time if the flip-flop is replaced with a pulsed latch. The pulsed latch has a pulse width of 150 ps, a setup time of 40 ps, a hold time of 5 ps, a *clk*-to-*Q* propagation delay of 82 ps and contamination delay of 52 ps, and a *D*-to-*Q* propagation delay of 92 ps.

Solution: t_{pd} is still 1000 ps. According to EQ (7.5), the cycle time must be at least 92 + 1000 = 1092 ps.

called a *race condition*, *hold time failure*, or *min-delay failure*. It can only be fixed by redesigning the logic, not by slowing the clock. Therefore, designers should be very conservative in avoiding such failures because modifying and refabricating a chip is very expensive and time-consuming.

Figure 7.9 shows the min-delay timing constraints on a path from one flip-flop to the next assuming ideal clocks with no skew. The path begins with the rising edge of the clock triggering *F1*. The data may begin to change at *Q1* after a *clk*-to-*Q* contamination delay, and at *D2* after another logic contamination delay. However, it must not reach *D2* until at least the hold time t_{hold} after the clock edge, lest it corrupt the contents of *F2*. Hence we solve for the minimum logic contamination delay:

$$t_{cd} \geq t_{\text{hold}} - t_{cq} \tag{7.7}$$

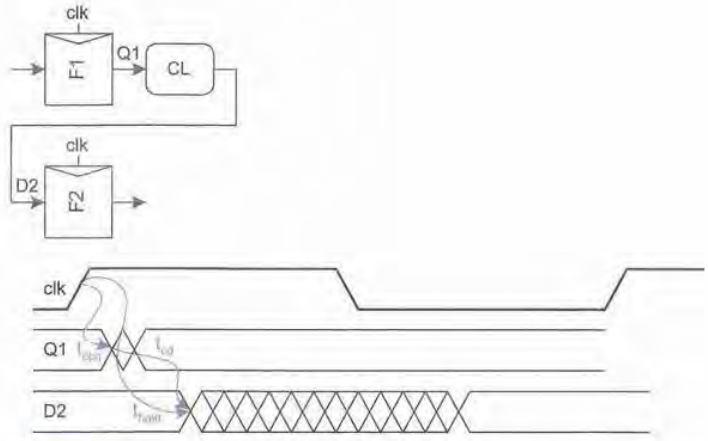


FIG 7.9 Flip-flop latch min-delay constraint

Example

In the ALU self-bypass example with flip-flops from Figure 7.6, the earliest input to the late bypass multiplexer is the *imm* value coming from another flip-flop. Will this path experience any hold time failures?

Solution: No. The late bypass mux has t_{cd} = 45 ps. The flip-flops have $t_{\text{hold}} = -10$ ps and t_{ccq} = 75 ps. Hence, EQ (7.7) is easily satisfied.

If the contamination delay through the flip-flop exceeds the hold time, you can safely use back-to-back flip-flops. If not, you must explicitly add delay between the flip-flops (e.g., with a buffer) or use special slow flip-flops with greater than normal contamination delay on paths that require back-to-back flops. Scan chains are a common example of paths with back-to-back flops.

Figure 7.10 shows the min-delay timing constraints on a path from one transparent latch to the next. The path begins with data passing through *L1* on the rising edge of ϕ_1. It must not reach *L2* until a hold time after the previous falling edge of ϕ_2 because *L2* should have become safely opaque before *L1* becomes transparent. As the edges are separated by $t_{\text{nonoverlap}}$, the minimum logic contamination delay through each phase of logic is

$$t_{cd1}, t_{cd2} \geq t_{\text{hold}} - t_{ccq} - t_{\text{nonoverlap}} \tag{7.8}$$

(Note that our derivation found the minimum delay through the first half-cycle, but that the second half-cycle has the same constraint.)

This result shows that by making $t_{\text{nonoverlap}}$ sufficiently large, hold time failure can be avoided entirely. However, generating and distributing nonoverlapping clocks is very challenging at high speeds. Therefore, most commercial transparent latch-based systems use the clock and its complement. In this case, $t_{\text{nonoverlap}} = 0$ and the contamination delay constraint is the same between the latches and flip-flops.

This leads to an apparent paradox: The contamination delay constraint applies to each phase of logic for latch-based systems, but to the entire cycle of logic for flip-flops. Therefore, latches seem to require twice the overall logic contamination delay as compared to flip-flops. Yet flip-flops can be built from a pair of latches! The paradox is resolved by observing that a flip-flop has an internal race condition between the two latches. The flip-flop must be carefully designed so that it always operates reliably.

Figure 7.11 shows the min-delay timing constraints on a path from one pulsed latch to the next. Now data departs on the rising edge of the pulse but must hold until after the falling edge of the pulse. Therefore, the pulse width effectively increases the hold time of the pulsed latch as compared to a flip-flop.

$$t_{cd} \geq t_{\text{hold}} - t_{ccq} + t_{pw} \tag{7.9}$$

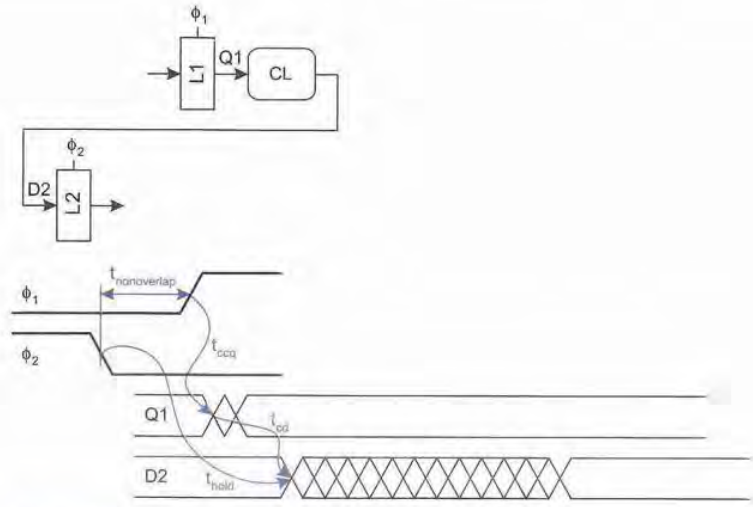


FIG 7.10 Two-phase latch min-delay constraint

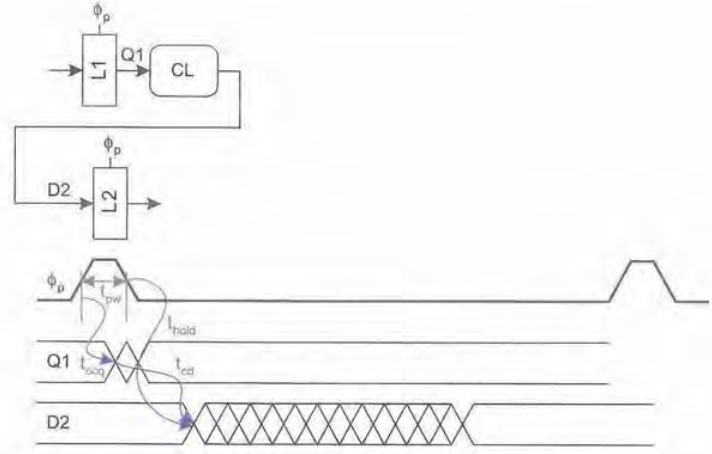


FIG 7.11 Pulsed latch min-delay constraint

Example

If the ALU self-bypass path uses pulsed latches in place of flip-flops, will it have any hold-time problems?

Solution: Yes. The late bypass mux has t_{cd} = 45 ps. The pulsed latches have t_{pw} = 150 ps, t_{hold} = 5 ps, and t_{ccq} = 52 ps. Hence, EQ (7.9) is badly violated. *Src1* may receive *imm* from the next instruction rather than the current instruction. The problem could be solved by adding buffers after the *imm* pulsed latch. The buffers would need to add a minimum delay of $t_{\text{hold}} - t_{ccq} + t_{pw} - t_{cd}$ = 58 ps. Alternatively, the *imm* pulsed latch could be replaced with a flip-flop without slowing the critical path. If the flip-flop were designed with a very long (>110 ps) contamination delay, the race would be avoided.

7.2.4 Time Borrowing

In a system using flip-flops, data departs the first flop on the rising edge of the clock and must set up at the second flop before the next rising edge of the clock. If the data arrives late, the circuit produces the wrong result. If the data arrives early, it is blocked until the clock edge, and the remaining time goes unused. Therefore, we say the clock imposes a *hard edge* because it sharply delineates the cycles.

In contrast, when a system uses transparent latches, the data can depart the first latch on the rising edge of the clock, but does not have to set up until the falling edge of the clock on the receiving latch. If one half-cycle or stage of a pipeline has too much logic, it can borrow time into the next half-cycle or stage, as illustrated in Figure 7.12(a) [Bernstein99]. *Time borrowing* can accumulate across multiple cycles. However, in systems with feedback, the long delays must be balanced by shorter delays so that the overall loop completes in the time available. For example, Figure 7.12(b) shows a single-cycle self-bypass loop in which time borrowing occurs across half-cycles, but the entire path must fit in one cycle. A typical example of a self-bypass loop is the execution stage of a pipelined processor in which an ALU must complete an operation and bypass the result back for use in the ALU on a dependent instruction. Most critical paths in digital systems occur in self-bypass loops because otherwise latency does not matter.

Figure 7.13 illustrates the maximum amount of time that a two-phase latch-based system can borrow (beyond the $T_c/2 - t_{pdq}$ nominally available to each half-cycle of logic). Because data does not have to set up until the falling edge of the receiving latch's clock, one phase can borrow up to half a cycle of time from the next (less setup time and nonoverlap):

$$t_{\text{borrow}} \leq \frac{T_c}{2} - \left(t_{\text{setup}} + t_{\text{nonoverlap}}\right) \tag{7.10}$$

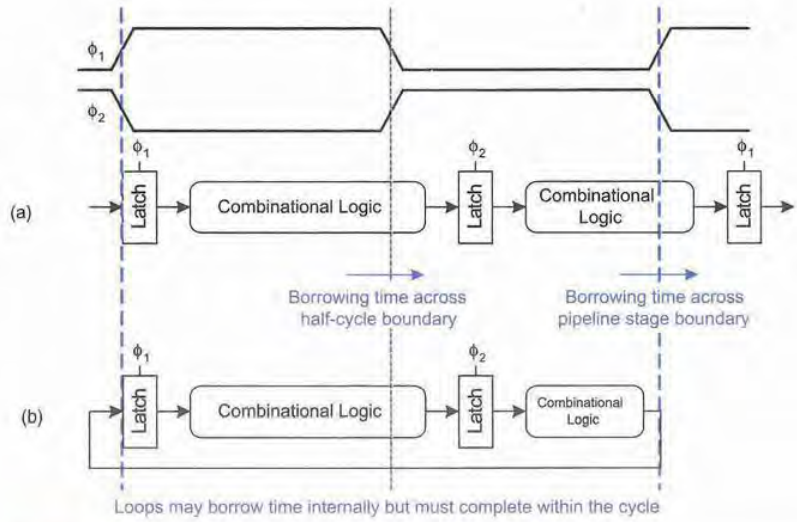


FIG 7.12 Time borrowing

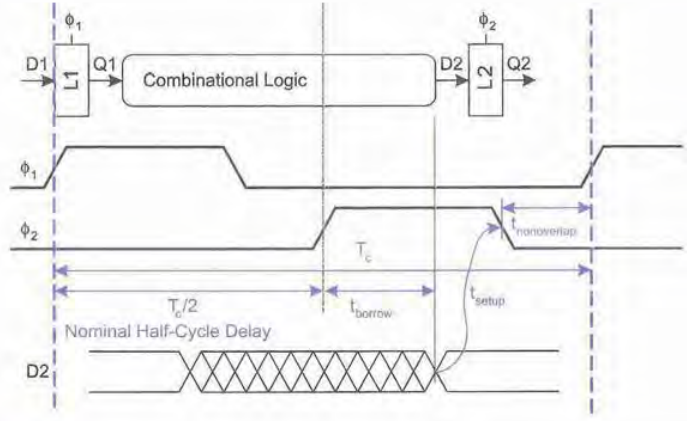


FIG 7.13 Maximum amount of time borrowing

Example

Suppose the ALU self-bypass path is modified to use two-phase transparent latches. A mid-cycle ϕ_2 latch is placed after the adder, as shown in Figure 7.14. The latches have a setup time of 40 ps, a hold time of 5 ps, a *clk*-to-*Q* propagation delay of 82 ps and contamination delay of 52 ps, and a *D*-to-*Q* propagation delay of 82 ps. Compute the minimum cycle time for the path. How much time is borrowed through the mid-cycle latch at this cycle time? If the cycle time is increased to 2000 ps, how much time is borrowed?

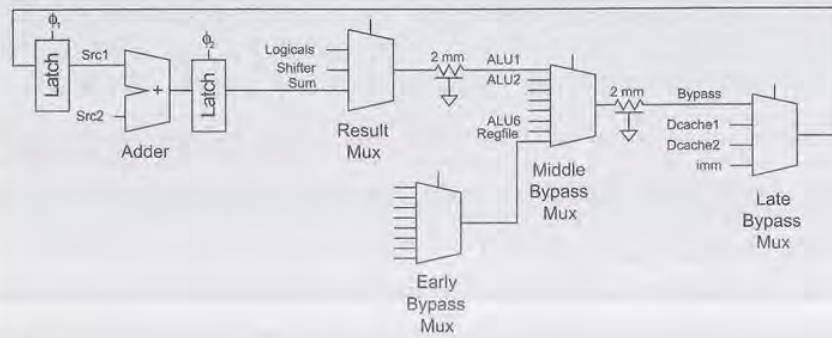


FIG 7.14 ALU self-bypass path with two-phase latches

Solution: According to EQ (7.3), the cycle time is $T_c = 82 + 590 + 82 + 410 = 1164$ ps. The first half of the cycle involves the latch and adder delays and consumes $82 + 590 = 672$ ps. The nominal half-cycle time is $T_c/2 = 582$ ps. Hence, the path borrows 90 ps from the second half-cycle. If the cycle time increases to 2000 ps and the nominal half-cycle time becomes 1000 ps, time borrowing no longer occurs.

Pulsed latches can be viewed as transparent latches with a narrow pulse. If the pulse is wider than the setup time, pulsed latches are also capable of a small amount of time borrowing from one cycle to the next.

$$t_{\text{borrow}} \leq t_{pw} - t_{\text{setup}} \tag{7.11}$$

Time borrowing has two benefits for the system designer. The most obvious is *intentional time borrowing*, in which the designer can more easily balance logic between half-cycles and pipeline stages. This leads to potentially shorter design time because the balancing can take place during circuit design rather than requiring changes to the microarchitecture to explicitly move functions from one stage to another. The other is

Example

If the ALU self-bypass path uses pulsed latches, how much time may it borrow from the next cycle?

Solution: None. Because the path is a feedback loop, if its outputs arrive late and borrow time, the path begins later on the next cycle. This in turn causes the outputs to arrive later. Time borrowing can only be used to balance logic within a pipeline but, despite the wishes of many designers, it does not increase the amount of time available in a clock cycle.

opportunistic time borrowing. Even if the designer carefully equalizes the delay in each stage at design time, the delays will differ from one stage to another in the fabricated chip because of process and environmental variations and inaccuracies in the timing model used by the CAD system. In a system with hard edges, the longest cycle sets the minimum clock period. In a system capable of time borrowing, the slow cycles can opportunistically borrow time from faster ones and average out some of the variation.

Some experienced design managers forbid the use of intentional time borrowing until the chip approaches tapeout. Otherwise designers are overly prone to assuming that their pipeline stage can borrow time from adjacent stages. When many designers make this same assumption, all of the paths become excessively long. Worse yet, the problem may be hidden until full chip timing analysis begins, at which time it is too late to redesign so many paths. Another solution is to do full-chip timing analysis starting early in the design process.

7.2.5 Clock Skew

The analysis so far has assumed ideal clocks with zero skew. In reality clocks have some uncertainty in their arrival times that can cut into the time available for useful computation, as shown in Figure 7.15(a). The bold *clk* line indicates the latest possible clock arrival time. The hashed lines show that the clock might arrive over a range of earlier times because of skew. The worst scenario for max delay in a flip-flop-based system is that the launching flop receives its clock late and the receiving flop receives its clock early. In this case, the clock skew is subtracted from the time available for useful computation and appears as sequencing overhead. The worst scenario for min delay is that the launching flop receives its clock early and the receiving clock receives its clock late, as shown in Figure 7.15(b). In this case, the clock skew effectively increases the hold time of the system.

$$t_{pd} \leq T_c - \underbrace{\left(t_{pcq} + t_{\text{setup}} + t_{\text{skew}}\right)}_{\text{sequencing overhead}} \tag{7.12}$$

$$t_{cd} \geq t_{\text{hold}} - t_{ccq} + t_{\text{skew}} \tag{7.13}$$

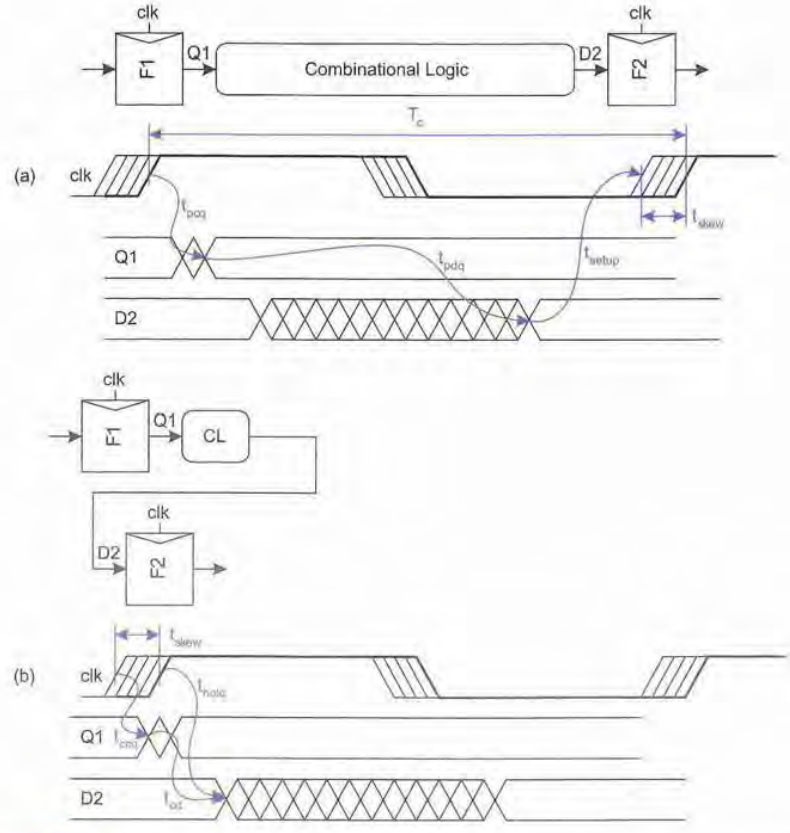


FIG 7.15 Clock skew and flip-flops

In the system using transparent latches, clock skew does not degrade performance. Figure 7.16 shows how the full cycle (less two latch delays) is available for computation even when the clocks are skewed because the data can still arrive at the latches while they are transparent. Therefore, we say that transparent latch-based systems are *skew-tolerant*. However, skew still effectively increases the hold time in each half-cycle. It also cuts into the window available for time borrowing.

$$t_{pd} \leq T_c - \underbrace{\left(2t_{pdq}\right)}_{\text{sequencing overhead}} \tag{7.14}$$

$$t_{cd1}, t_{cd2} \geq t_{\text{hold}} - t_{ccq} - t_{\text{nonoverlap}} + t_{\text{skew}} \tag{7.15}$$

Example

If the ALU self-bypass path from Figure 7.6 can experience 50 ps of skew from one cycle to the next between flip-flops in the various ALUs, what is the minimum cycle time of the system? How much clock skew can the system have before hold time failures occur?

Solution: According to EQ (7.12), the cycle time should increase by 50 ps to 1202 ps. The maximum skew for which the system can operate correctly at any cycle time is $t_{cd} - t_{\text{hold}} + t_{ccq} = 45 - (-10) + 75 = 130$ ps.

$$t_{\text{borrow}} \le \frac{T_c}{2} - \left(t_{\text{setup}} + t_{\text{nonoverlap}} + t_{\text{skew}}\right) \tag{7.16}$$

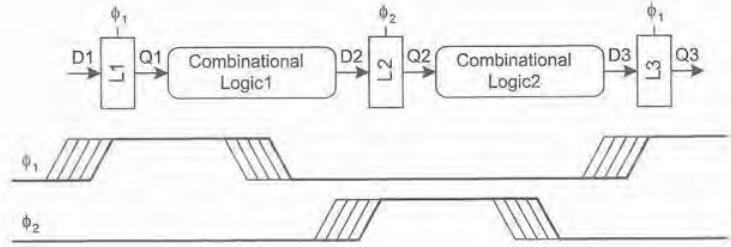


FIG 7.16 Clock skew and transparent latches

Pulsed latches can tolerate an amount of skew proportional to the pulse width. If the pulse is wide enough, the skew will not increase the sequencing overhead because the data can arrive while the latch is transparent. If the pulse is narrow, skew can degrade performance. Again skew effectively increases the hold time and reduces the amount of time available for borrowing (see Exercise 7.7).

$$t_{pd} \le T_c - \underbrace{\max\left(t_{pdq},\, t_{pcq} + t_{\text{setup}} - t_{pw} + t_{\text{skew}}\right)}_{\text{sequencing overhead}} \tag{7.17}$$

$$t_{cd} \ge t_{\text{hold}} + t_{pw} - t_{ccq} + t_{\text{skew}} \tag{7.18}$$

$$t_{\text{borrow}} \le t_{pw} - \left(t_{\text{setup}} + t_{\text{skew}}\right) \tag{7.19}$$

In summary, systems with hard edges (e.g., flip-flops) subtract clock skew from the time available for useful computation. Systems with softer edges (e.g., latches) take advan-

tage of the window of transparency to tolerate some clock skew without increasing the sequencing overhead. Clock skew will be addressed further in Section 12.5. In particular, different amounts of skew can be budgeted for min-delay and max-delay checks. Moreover, nearby sequential elements are likely to see less skew than elements on opposite corners of the chip. Current automated place & route tools spend considerable effort to model clock delays and insert buffer elements to minimize clock skew, but skew is a growing problem for systems with aggressive cycle times.

7.3 Circuit Design of Latches and Flip-flops

Conventional CMOS latches are built using pass transistors or tristate buffers to pass the data while the latch is transparent and feedback to hold the data while the latch is opaque. We begin by exploring circuit designs for basic latches, then build on them to produce flip-flops and pulsed latches. Many latches accept reset and/or enable inputs. It is also possible to build logic functions into the latches to reduce the sequencing overhead.

A number of alternative latch and flip-flop structures have been used in commercial designs. The True Single Phase Clocking (TSPC) technique uses a single clock with no inversions to simplify clock distribution. The Klass Semidynamic Flip-Flop (SDFF) is a fast flip-flop using a domino-style input stage. Differential flip-flops are good for certain applications. Each of these alternatives are described and compared.

7.3.1 Conventional CMOS Latches

Figure 7.17(a) shows a very simple transparent latch built from a single transistor. It is compact and fast but suffers four limitations. The output does not swing from *rail-to-rail* (i.e., from GND to V_{DD}); it never rises above $V_{DD} - V_t$. The output is also *dynamic*; in other words, the output floats when the latch is opaque. If it floats long enough, it can be disturbed by leakage (see Section 6.3.3). D drives the *diffusion input* of a pass transistor directly, leading to potential noise issues (see Section 6.3.9) and making the delay harder to model with static timing analyzers. Finally, the state node is *exposed*, so noise on the output can corrupt the state. The remainder of the figures illustrate improved latches using more transistors to achieve more robust operation.

Figure 7.17(b) uses a CMOS transmission gate in place of the single nMOS pass transistor to offer rail-to-rail output swings. It requires a complementary clock $\overline{\phi}$, which can be provided as an additional input or locally generated from ϕ through an inverter. Figure 7.17(c) adds an output inverter so that the state node X is isolated from noise on the output. Of course, this creates an inverting latch. Figure 7.17(d) also behaves as an inverting latch with a buffered input but unbuffered output. As discussed in Sections 2.5.6 and 6.2.5.1, the inverter followed by transmission gate is essentially equivalent to a tristate inverter but has a slightly lower logical effort because the output is driven by both transistors of the transmission gate in parallel. Both (c) and (d) are fast dynamic latches.

In modern processes, subthreshold leakage is large enough that dynamic nodes retain their values for only a short time, especially at the high temperature and voltage encountered during burn-in test. Therefore, practical latches need to be staticized, adding feedback to prevent the output from floating, as shown in Figure 7.17(e). When the clock is '1,' the input transmission gate is ON, the feedback tristate is OFF, and the latch is transparent. When the clock is '0,' the input transmission gate turns OFF. However, the feedback tristate turns ON, holding X at the correct level. Figure 7.17(f) adds an input inverter so the input is a transistor gate rather than unbuffered diffusion. Unfortunately, both (e) and (f) reintroduced output noise sensitivity: A large noise spike on the output can propagate backward through the feedback gates and corrupt the state node X. Figure 7.17(g) is a very robust transparent latch that addresses all of the deficiencies mentioned so far: The latch is static, all nodes swing rail-to-rail, the state noise is isolated from output noise, and the input drives transistor gates rather than diffusion. Such a latch is widely used in standard cell applications including the Artisan standard cell library [Artisan02]. It is recommended for all but the most performance- or area-critical designs.

In semicustom datapath applications where input noise can be better controlled, the inverting latch of Figure 7.17(h) may be preferable because it is faster and more compact; for example, Intel uses this as a standard datapath latch [Karnik01]. Figure 7.17(i) shows the *jamb latch*, a variation of (g) that reduces the clock load and saves two transistors by using a weak feedback inverter in place of the tristate. This requires careful circuit design to ensure that the tristate is strong enough to overpower the feedback inverter in all process corners. Figure 7.17(j) shows another jamb latch commonly used in register files and Field Programmable Gate Array (FPGA) cells. Many such latches read out onto a single D_{out} wire and only one is enabled at any given time with its *RD* signal. The Itanium 2 processor uses the latch shown in Figure 7.17(k) [Naffziger02]. In the static feedback, the pulldown stack is clocked, but the pullup is a weak pMOS transistor. Therefore, the gate driving the input must be strong enough to overcome the feedback. The Itanium 2 cell library also contains a similar latch with an additional input inverter to buffer the input when the previous gate is too weak or far away. With the input inverter, the latch can be viewed as a cross between the designs shown in (g) and (i). Some latches add one more inverter to provide both true and complementary outputs.

Figure 8.53 shows layouts for the latch of Figure 7.17(h) with a built-in clock inverter. The state node X can be shared between the transmission gate and tristate diffusions.

The dynamic latch of Figure 7.17(d) can also be drawn as a clocked tristate, as shown in Figure 7.18(a). Such a form is sometimes called *clocked CMOS* (C^2MOS) [Suzuki73]. The conventional form using the inverter and transmission gate is slightly faster because the output is driven through the nMOS and pMOS working in parallel. C^2MOS is slightly smaller because it eliminates two contacts. Figure 7.18(b) shows another form of the tristate that swaps the data and clock terminals. It is logically equivalent but electrically inferior because toggling D while the latch is opaque can cause charge sharing noise on the output node [Suzuki73].

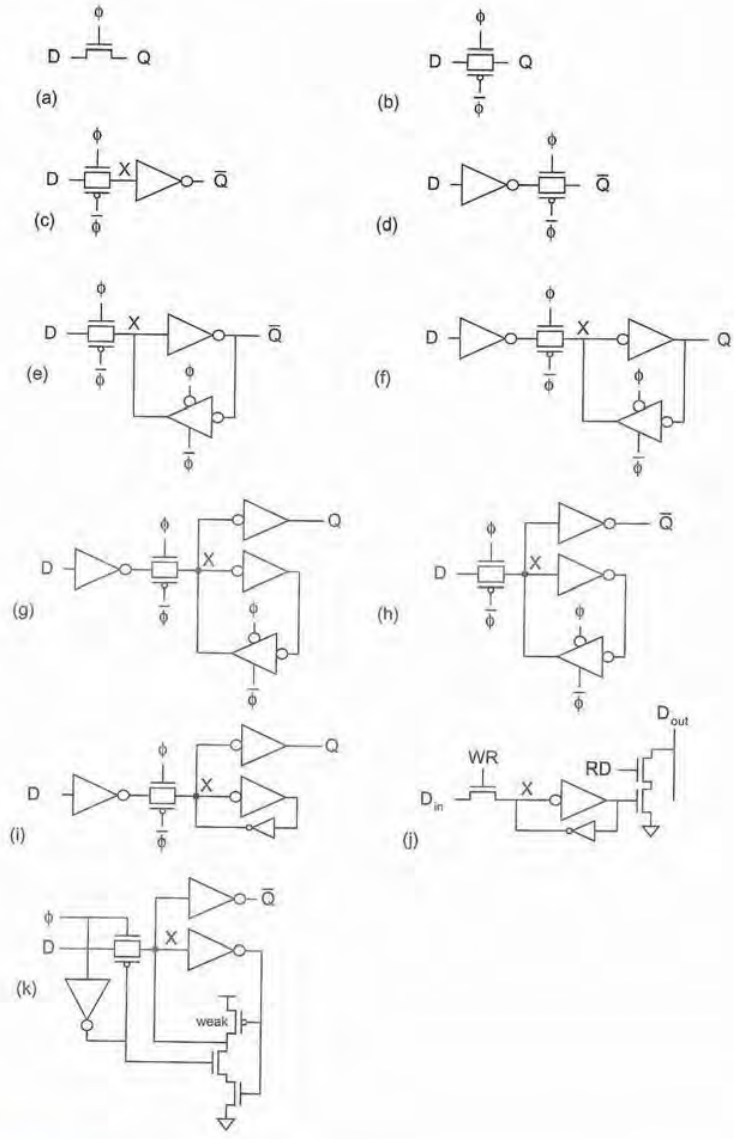


FIG 7.17 Transparent latches

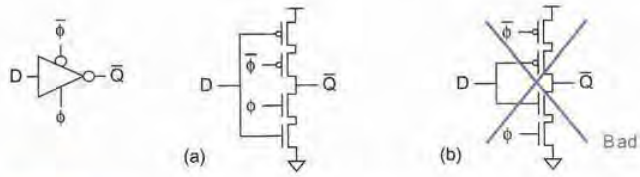


FIG 7.18 C²MOS Latch

All of the latches shown so far are transparent while ϕ is high. They can be converted to active-low latches by swapping ϕ and $\overline{\phi}$.

7.3.2 Conventional CMOS Flip-flops

Figure 7.19(a) shows a dynamic inverting flip-flop built from a pair of back-to-back dynamic latches [Suzuki73]. Either the first or the last inverter can be removed to reduce delay at the expense of greater noise sensitivity on the unbuffered input or output. Figure 7.19(b) adds feedback and another inverter to produce a noninverting static flip-flop. The PowerPC 603 microprocessor datapath used this flip-flop design without the input inverter or $\overline{Q}$ output [Gerosa94].

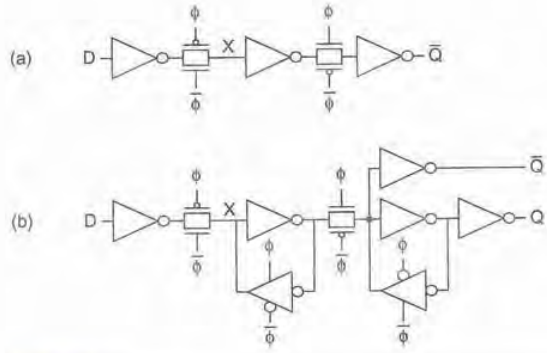


FIG 7.19 Flip-flops

Flip-flops usually take a single clock signal ϕ and locally generate its complement $\overline{\phi}$. If the clock rise/fall time is very slow, it is possible that both the clock and its complement will simultaneously be at intermediate voltages, making both latches transparent and increasing the flip-flop hold time. In ASIC standard cell libraries (such as the Artisan

library), the clock is both complemented and buffered in the flip-flop cell to sharpen up the edge rates at the expense of more inverters and clock loading.

Recall that the flip-flop also has a potential internal race condition between the two latches. This race can be exacerbated by skew between the clock and its complement caused by the delay of the inverter. Figure 7.20(a) redraws Figure 7.19(a) with a built-in clock inverter. When ϕ falls, both the clock and its complement are momentarily low as shown in Figure 7.20(b), turning on the clocked pMOS transistors in both transmission gates. If the skew (i.e., inverter delay) is too large, the data can sneak through both latches on the falling clock edge, leading to incorrect operation. Figure 7.20(c) shows a C²MOS dynamic flip-flop built using C²MOS latches rather than inverters and transmission gates [Suzuki73]. Because each stage inverts, data passes through the nMOS stack of one latch and the pMOS of the other, so skew that turns on both clocked pMOS transistors is not a hazard. However, the flip-flop is still susceptible to failure from very slow edge rates that turn both transistors partially ON. The same skew advantages apply even when an even number of inverting logic stages are placed between the latches; this technique is sometimes called *NO RAce* (NORA) [Gonclaves83]. In practice, most flip-flop designs carefully control the delay of the clock inverter so the transmission gate design is safe and slightly faster than C²MOS [Chao89].

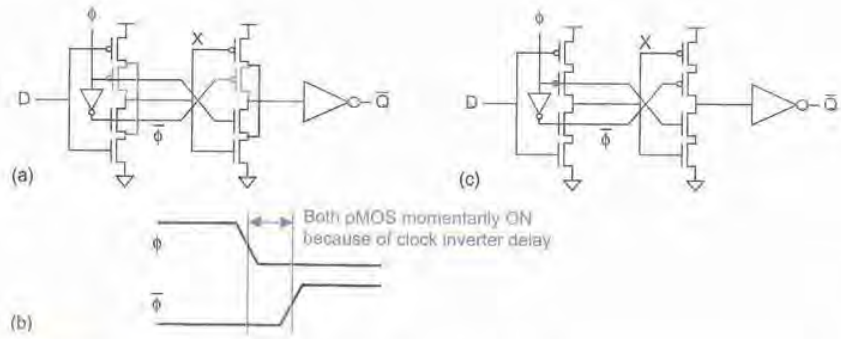


FIG 7.20 Transmission gate and NORA dynamic flip-flops

All of these flip-flop designs still present potential min-delay problems between flip-flops, especially when there is little or no logic between flops and the clock skew is large or poorly analyzed. For VLSI class projects where careful clock skew analysis is too much work and performance is less important, a reasonable alternative is to use a pair of two-phase nonoverlapping clocks instead of the clock and its complement, as shown in Figure 7.21. The flip-flop captures its input on the rising edge of ϕ_1. By making the nonoverlap large enough, the circuit will work despite large skews. However, the nonoverlap time is

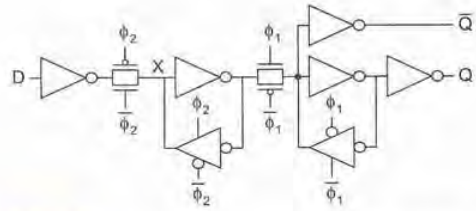


FIG 7.21 Flip-flop with two-phase nonoverlapping clocks

not used by logic, so it directly increases the setup time and sequencing overhead of the flip-flop (see Exercise 7.8).

7.3.3 Pulsed Latches

A pulsed latch can be built from a conventional CMOS transparent latch driven by a brief clock pulse. Figure 7.22(a) shows a simple pulse generator, sometimes called a *clock chopper* or *one-shot* [Harris01a].

The *Naffziger pulsed latch* used on the Itanium 2 processor consists of the latch from Figure 7.17(j) driven by even shorter pulses produced by the generator of Figure 7.22(b) [Naffziger02]. This pulse generator uses a fairly slow (weak) inverter to produce a pulse with a nominal width of about one-sixth of the cycle (125 ps for 1.2 GHz operation). When disabled, the internal node of the pulse generator floats high momentarily, but no keeper is required because the duration is short. Of course, the enable signal has setup and hold requirements around the rising edge of the clock, as shown in Figure 7.22(c).

Figure 7.22(d) shows yet another pulse generator used on an NEC RISC processor [Kozu96] to produce substantially longer pulses. It includes a built-in dynamic transmission-gate latch to prevent the enable from glitching during the pulse.

Many designers consider short pulses risky. The pulse generator should be carefully simulated across process corners and possible RC loads to ensure the pulse is not degraded too badly by process variation or routing. However, the Itanium 2 team found that the pulses could be used just as regular clocks as long as the pulse generator had adequate drive.

The *Partovi pulsed latch* in Figure 7.23 eliminates the need to distribute the pulse by building the pulse generator into the latch itself [Partovi96, Draper97]. The weak cross-coupled inverters in the dashed box staticize the circuit, although the latch is susceptible to back-driven output noise on Q or $\overline{Q}$ unless an extra inverter is used to buffer the output. The Partovi pulsed latch was used on the AMD K6 and Athlon [Golden99], but is slightly slower than a simple latch [Naffziger02]. It was originally called an *Edge Triggered Latch* (ETL), but strictly speaking is a pulsed latch because it has a brief window of transparency.

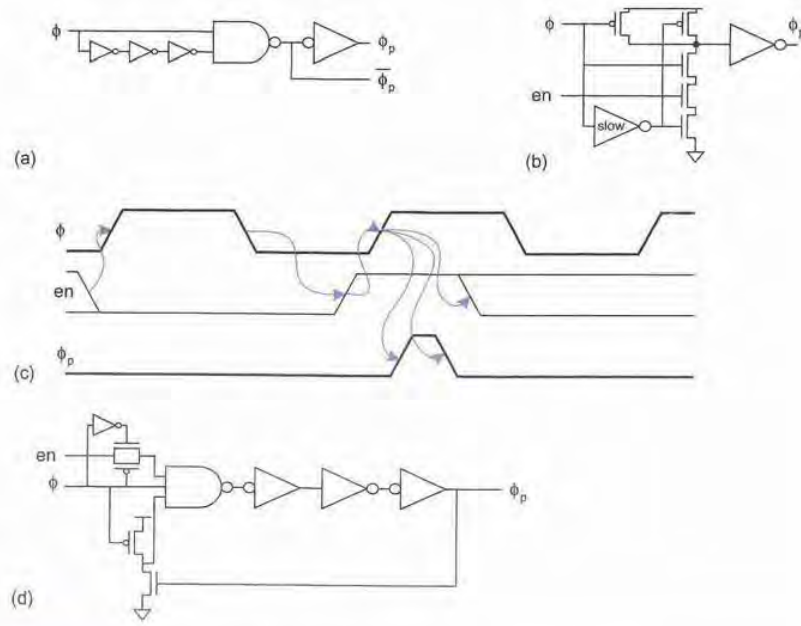


FIG 7.22 Pulse generators

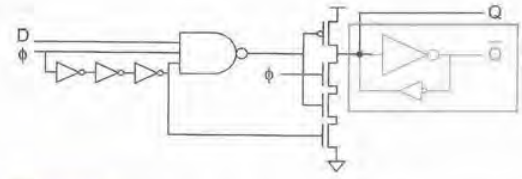


FIG 7.23 Partovi pulsed latch

7.3.4 Resettable Latches and Flip-flops

Most practical sequencing elements require a reset signal to enter a known initial state on startup. Figure 7.24 shows latches and flip-flops with reset inputs. There are two types of reset: *synchronous* and *asynchronous*. Asynchronous reset forces Q low immediately, while synchronous reset waits for the clock. Synchronous reset signals must be stable for a setup and hold time around the clock edge while asynchronous reset is characterized by a propagation delay from reset to output. Synchronous reset simply requires ANDing the input D with $\overline{reset}$. Asynchronous reset requires gating both the data and the feedback to force the

reset independent of the clock. The tristate NAND gate can be constructed from a NAND gate in series with a clocked transmission gate.

Settable latches and flip-flops force the output high instead of low. They are similar to resettable elements of Figure 7.24 but replace NAND with NOR and $\overline{reset}$ with *set*. Figure 7.25 shows a flip-flop combining both asynchronous set and reset.

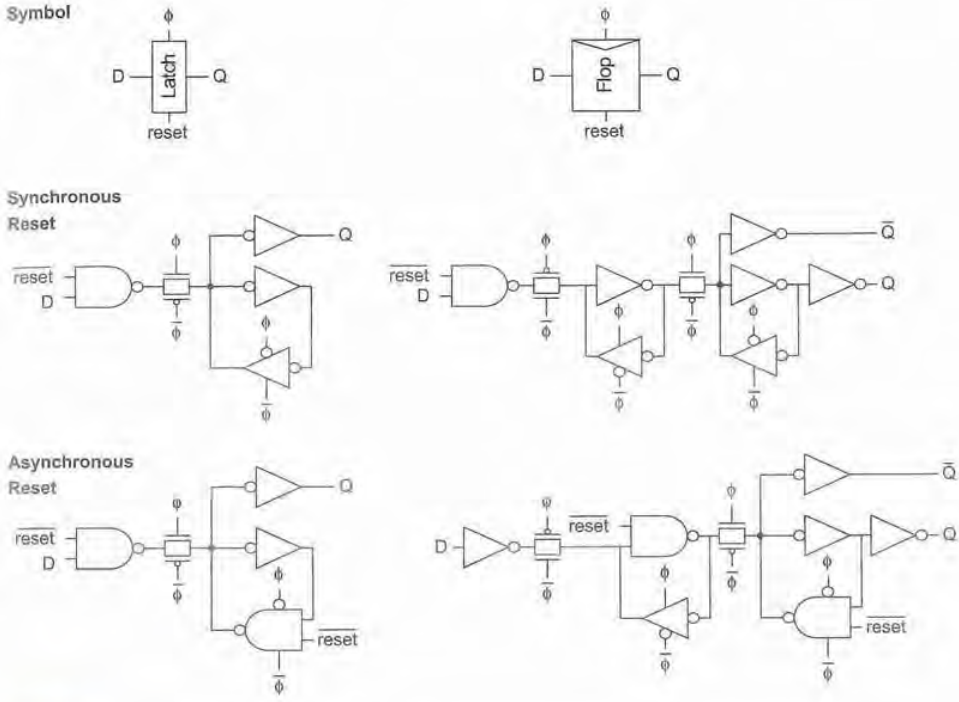


FIG 7.24 Resettable latches and flip-flops

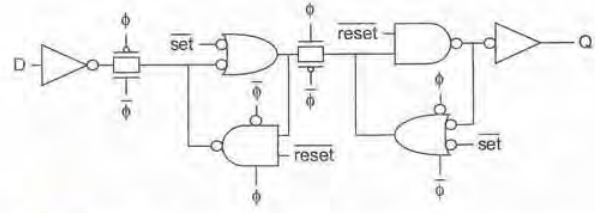


FIG 7.25 Flip-flop with asynchronous set and reset

7.3.5 Enabled Latches and Flip-flops

Sequencing elements also often accept an enable input. When enable *en* is low, the element retains its state independently of the clock. The enable can be performed with an input multiplexer or clock *gating*, as shown in Figure 7.26. The input multiplexer feeds back the old state when the element is disabled. The multiplexer adds area and delay. Clock gating does not affect delay from the data input and the AND gate can be shared among multiple clocked elements. Moreover, it significantly reduces power consumption because the clock on the disabled element does not toggle. However, the AND gate delays the clock, potentially introducing clock skew. Section 12.5.5 addresses techniques to minimize the skew by building the AND gate into the final buffer of the clock distribution network. *en* must be stable while the clock is high to prevent glitches on the clock, as will be discussed further in Section 7.4.3.

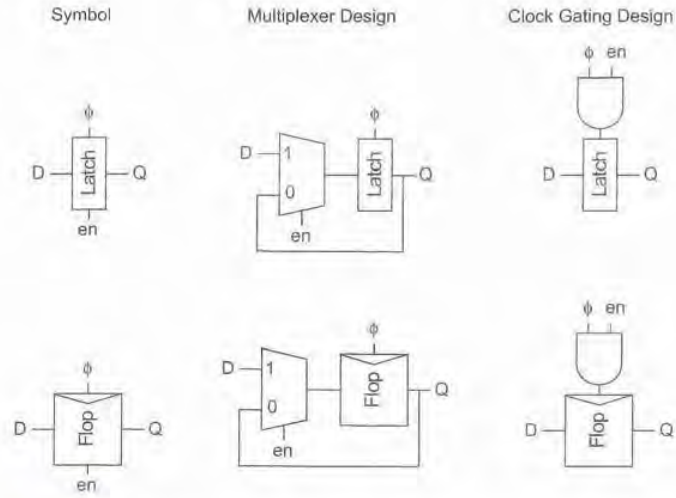


FIG 7.26 Enabled latches and flip-flops

7.3.6 Incorporating Logic into Latches

Another way to reduce the sequencing overhead of latches is to replace some of the inverters in the latch with gates that perform useful computation. Figure 7.27 shows two ways to do this in dynamic latches. The DEC Alpha 21164 used an assortment of latches built from a clocked transmission gate preceded and followed by inverting static CMOS gates

such as NANDs, NORs, or inverters [Bowhill95]. This provides the low overhead of the transmission gate latch while preserving the buffered inputs and outputs. The *mux-latch* consists of two transmission gates in parallel controlled by clocks gated with the corresponding select signals. It integrates the multiplexer function with no extra delay from the D inputs to the Q outputs except the small amount of extra diffusion capacitance on the state node. Note that the setup time on the select inputs is relatively high. The clock gating will introduce skew unless the clocking methodology systematically plans to gate all clocks. The same principles extend to static latches and flip-flops.

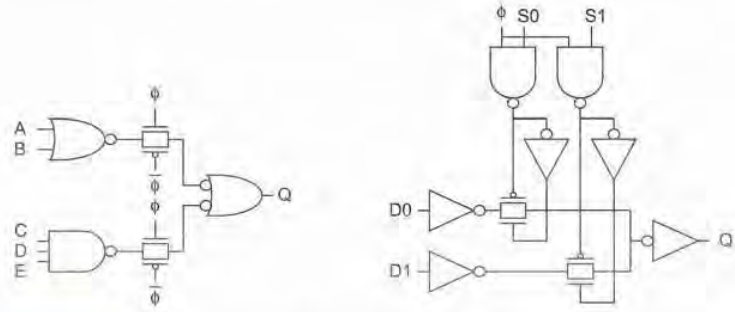


FIG 7.27 Combining logic and latches

7.3.7 Klass Semidynamic Flip-flop (SDFF)

The *Klass semidynamic flip-flop* (SDFF) [Klass99] shown in Figure 7.28 is a cross between a pulsed latch and a flip-flop. Like the Partovi pulsed latch, it operates on the principle of intersecting pulses. However, it uses a dynamic NAND gate in place of the static NAND. While the clock is low, X precharges high and Q holds its old state. When the clock rises, the dynamic NAND evaluates. If D is '0,' X remains high and the top nMOS transistor turns OFF. If D is '1' and X starts to fall low, the transistor remains ON to finish the transition. This allows for a very short pulse and short hold time. The weak cross-coupled inverters staticize the flip-flop and the final inverter buffers the output node.

Like a pulsed latch, the SDFF accepts rising inputs slightly after the rising clock edge. Like a flip-flop, falling inputs must set up before the rising clock edge. It is called *semidynamic* because it combines the dynamic input stage with static operation. The SDFF is slightly faster than the Partovi pulsed latch but loses the skew tolerance and time borrowing capability. The Sun UltraSparc III built logic into the SDFF very efficiently by replacing the single transistor connected to D with a collection of transistors performing the OR or multiplexer functions [Heald00].

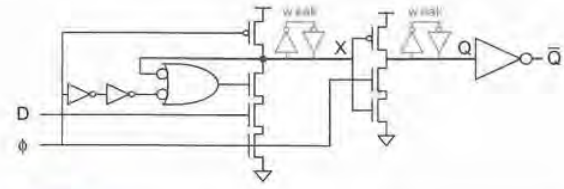


FIG 7.28 Klass semidynamic flip-flop

7.3.8 Differential Flip-flops

Differential flip-flops accept true and complementary inputs and produce true and complementary outputs. They are built from a clocked sense amplifier so they can rapidly respond to small differential input voltages. While they are larger than an ordinary single-ended flip-flop—having an extra inverter to produce the complementary output—they work well with low-swing inputs such as register file bitlines (Section 11.2.3) and low-swing busses (Section 4.6.7).

Figure 7.29(a) shows a differential *sense-amplifier flip-flop* (SA-F/F) receiving differential inputs and producing a differential output [Matsui94]. When the clock is low, the internal nodes X and $\overline{X}$ precharge. When the clock rises, one of the two nodes is pulled down, while the cross-coupled pMOS transistors act as a keeper for the other node. The SR latch formed by the cross-coupled NAND gates behaves as a slave stage, capturing the output and holding it through precharge. The flip-flop can amplify and respond to small differential input voltages, or it can use an inverter to derive the complementary input from D. This flip-flop was used in the Alpha 21264 [Gronowski98]. It has a small clock load and avoids the need for an inverted clock. If the two input transistors are replaced by true and complementary nMOS logic networks, the SA-F/F can also perform logic functions at the expense of greater setup time [Klass99].

The original SA-F/F suffers from the possibility that one of the internal nodes will float low if the inputs switch while the clock is high. The StrongArm 110 processor [Montanaro96] adds the weak nMOS transistor shown in Figure 7.29(a) to fully staticize the flip-flop at the expense of a small amount more internal loading and delay.

Although the sense amplifier stage is fast, the propagation delay through the two cross-coupled NAND gates hurts performance. The NAND gates serve as a slave SR latch and are only necessary to convert the monotonically falling pulsed X signals to static Q outputs; they can be replaced by HI-skew inverters when Q drives domino gates. Alternatively, Figure 7.29(b) shows how to build a faster slave latch to replace the cross-coupled NANDs at the expense of eight more transistors [Nikolić00]. When the X signal falls, it turns on the pMOS transistor to immediately pull the Q output high. It also drives one of the inverters high, which pulls down the $\overline{Q}$ output through the opposite nMOS transistor.

The four blue transistors serve as small cross-coupled tristate keepers that hold the outputs after the master stage precharges, but turn off to avoid contention when the outputs need to switch. This slave latch still involves two gate delays for the falling output, but the delays are faster because the gates avoid crowbar current, have lower logical effort, and are skewed to favor the critical edges.

The AMD K6 used another differential flip-flop shown in Figure 7.29(c) at the interface from static to self-resetting domino logic [Draper97]. The master stage consists of a self-resetting dual-rail domino gate. Assume the internal nodes are initially precharged. On the rising edge of the clock, one of the two will pull down and drive the corresponding output high. The OR gate detects this and produces a *done* signal that precharges the internal nodes and resets the outputs. Therefore, the flip-flop produces pulsed outputs primarily suitable for use in subsequent self-resetting domino gates (see Section 7.5.2.4). The cross-coupled pMOS transistors improve the noise immunity while the cross-coupled inverters staticize the internal nodes.

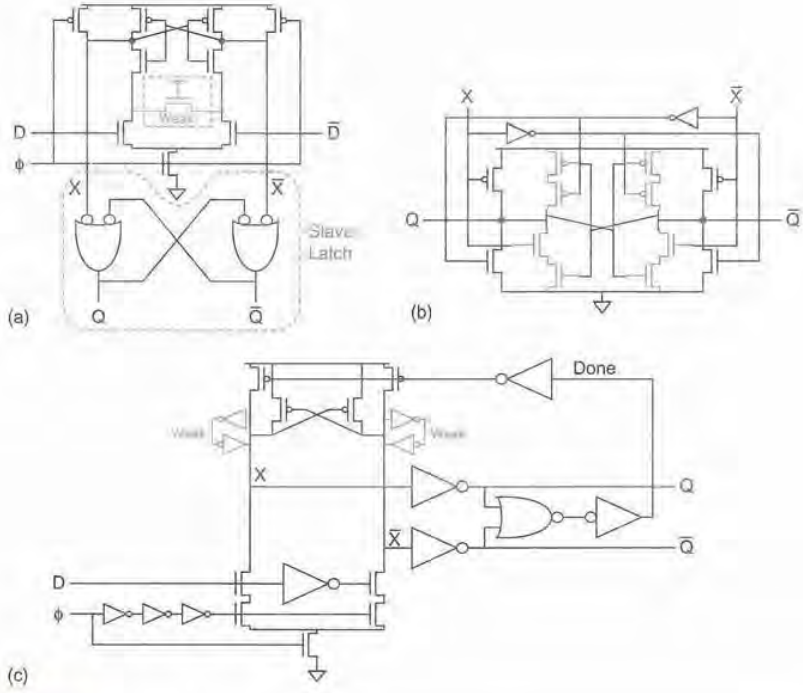


FIG 7.29 Differential flip-flops



7.3.9 True Single-phase-clock (TSPC) Latches and Flip-flops

Conventional latches require both true and complementary clock signals. In modern CMOS systems, the complement is normally generated locally with an inverter in the latch cell. In the late 1980s, some researchers worked to avoid the complementary signal. The *True Single-Phase-Clock* (TSPC) latches and flip-flops replace the inverter-transmission gate or C²MOS stage with a pair of stages requiring only the clock, not its complement [Ji-ren87, Yuan89]. Figure 7.30(a and b) show active high and low TSPC dynamic latches. Figure 7.30(c) shows a TSPC dynamic flip-flop. Note that this flip-flop produces a momentary glitch on $\overline{Q}$ after the rising clock edge when D is low for multiple cycles; this increases the activity factor of downstream circuits and costs power. [Afghahi90] extends the TSPC principle to handle domino, RAMs, and other precharged circuits.

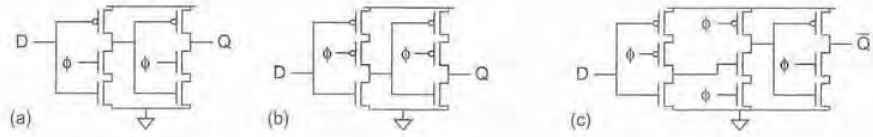


FIG 7.30 TSPC latches and flip-flops

The dynamic TSPC latches were used on the groundbreaking Alpha 21064 microprocessor [Dobberpuhl92]. Logic can be built into the first stage of each latch. The latch is not easy to staticize [Afghahi90]. In any case, the clock must also be reasonably sharp to prevent races when both transistors are partially ON [Larsson94]. The Alpha 21164 reverted to conventional dynamic latches for an estimated 10% speed improvement [Bowhill95]. In summary, TSPC is primarily of historic interest.

7.4 Static Sequencing Element Methodology

This section examines a number of issues designers must address when selecting a sequencing element methodology. We begin with general issues, and then proceed to techniques specific to flip-flops, pulsed latches, and transparent latches.

Until the 0.5 or 0.35 μm generation, leakage was relatively low and thus dynamic latches held their state for acceptably long times. The DEC Alpha 21164 was one of the last major microprocessors to use a dynamic latching methodology in a 0.35 μm process in the mid-1990s. It required a minimum operating frequency of 1/10th full speed to retain state, even during testing. Modern systems generally require static sequencing elements to hold state when clocks are gated or the system is tested at a moderate frequency. Leakage

is usually worst during burn-in testing at elevated temperature and voltage, where the chip must still function correctly to ensure good toggle coverage. Static elements are larger and somewhat slower than their dynamic counterparts.

Similarly, the growing difficulty and cost of debugging and testing has forced engineers to build design-for-test (DFT) features into the sequencing elements. The most important feature is *scan*, a special mode in which the latches or flip-flops can be chained together into a large shift register so that they can be read and written under external control during testing. This technique is discussed further in Section 9. Scan has become particularly important because chips have so many metal layers that most internal signals cannot be directly reached with probes. Moreover, some *flip-chips* are mounted upside down, making physical access even more difficult. Scan can dramatically decrease the time required to debug a chip and reduce the cost of testing, so most design methodologies dictate that all sequencing elements must be scannable despite the extra area this entails. The Alpha 21264 did not support full scan and was very difficult to debug, leading to a later-than-desired release.

Clock distribution is another key challenge. As we will see in Section 12.5, it is very difficult to distribute a single clock across a large die in a fashion that gets it to all sequencing elements at nearly the same time. Controlling the clock skew on more than one clock is even more difficult, so almost all modern designs distribute a single high-speed clock. Other signals such as complementary clocks, pulses, and delayed clocks are generated locally where they are needed. The clock edge rates must be relatively sharp to avoid races in which both the master and slave latches are partially on simultaneously. The global clock may have slow edge rates after propagating along long wires, so it is typically buffered locally (either in each sequencing element or in a buffer cell serving a bank of elements) to sharpen the edge rates. Clock power, from the clock distribution network and the clocked loads, typically accounts for one third to one half of the total chip power consumption. Therefore, clocks are often gated with an AND gate in the local clock buffer to turn off the sequencing elements for inactive units of the chip.

All bistable elements are subject to soft errors from alpha particles or cosmic rays striking the circuits and injecting charge onto sensitive nodes (see Section 4.8.7). Sequencing elements require relatively high capacitance on the state node to achieve low soft error rates. This can set a lower bound on the minimum transistor sizes on that node.

7.4.1 Choice of Elements

Flip-flops, pulsed latches, and transparent latches offer tradeoffs in sequencing overhead, skew tolerance, and simplicity.

7.4.1.1 Flip-flops As we have seen, flip-flops have fairly high sequencing overhead but are popular because they are so simple. Nearly all engineers understand how flip-flops work. Some synthesis tools and timing analyzers handle flip-flops much more gracefully than transparent latches. Many ASIC methodologies use flip-flops exclusively for pipelines and state machines. If performance requirements are not near the cutting edge of a process, flip-flops are clearly the right choice in today's CAD flows.

7.4.1.2 Pulsed Latches Pulsed latches are faster than flip-flops and offer some time-borrowing capability at the expense of greater hold times. They have fewer clocked transistors and hence lower power consumption. If intentional time borrowing is not necessary, you can model a pulsed latch as a flip-flop triggered on the rising edge of the pulse with a lower delay but a lengthy hold time. This makes pulsed latches relatively easy to integrate into flip-flop-based CAD flows. Moreover, the pulsed latches still offer opportunistic time borrowing to compensate for modeling inaccuracies even if the intentional time borrowing is not used.

The long hold times make pulsed latches unsuitable for use in pipelines with no logic between pipeline stages. One solution is to use ordinary flip-flops in place of the pulsed latches in these circumstances where speed is not important. Unfortunately, some pulsed latches fan out to multiple paths, some of which are short and others long. The Itanium 2 processor used the *clocked deracer* in conjunction with Naffziger pulsed latches, as shown in Figure 7.31 [Naffziger02]. These were placed before the receiving latches on short paths and block incoming paths while the receiving latch is transparent. They automatically adapt to pulse with variation and hence have a shorter nominal propagation delay than buffers, but also consume more power than buffers because of the clock loading [Rusu03].

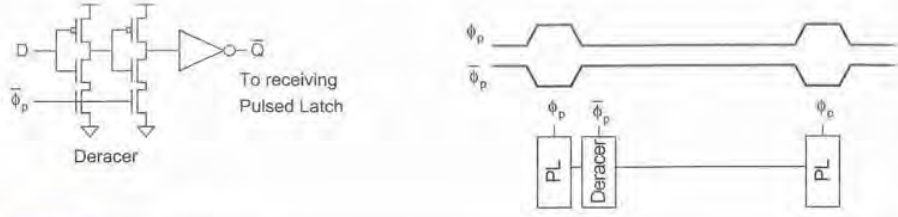


FIG 7.31 Clocked deracer

7.4.1.3 Transparent Latches Transparent latches also have lower sequencing overhead than flip-flops and are attractive because they permit nearly half a cycle of time borrowing. One latch must be placed in each half-cycle. Data can arrive at the latch any time the latch is transparent. A convenient design approach is to nominally place the latch at the beginning of each half-cycle. Then time borrowing occurs when the logic in one half-cycle is longer than nominal and data does not arrive at the next latch until some time into the next half-cycle.

Figure 7.32 illustrates pipeline timing for short and long logic paths between latches. When the path is short (a), the data arrives at the second latch early and is delayed until the rising edge of ϕ_2. Therefore, it is natural to consider latches residing at the beginning of their half-cycle because short paths automatically adjust to operate this way. When the path is longer (b), it borrows time from the first half-cycle into the second. Notice how

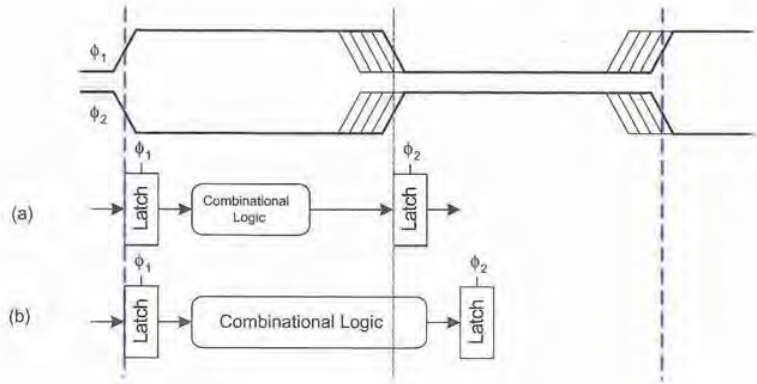


FIG 7.32 Latch placement and time borrowing

clock skew does not slow long paths because the data does not arrive at the latch until after the latest skewed rising edge.

Logic blocks involving multiple signals must ensure that each signal path passes through two latches in each cycle. Signals can be classified as phase 1 or phase 2 and logic gates must receive all their inputs from the same phase. Section 7.4.3 develops a formal notation of timing types to track when signals are safe to use.

7.4.2 Low-power Sequential Design

All of the techniques mentioned in Section 6.5 can be used in common to create low-power libraries for sequential elements. However, because flip-flops and latches are normally clocked every cycle, both the core latch design and the clock distribution network must be carefully scrutinized to achieve low dynamic power. Keep device sizes small inside the core latch and minimize the number of clocked transistors. Pulsed latches are attractive because they have fewer clocked transistors than two-phase latches or flip-flops. The conventional flip-flop of Figure 7.19(b) is also power-efficient because it is simple. [Stojanovic99] presents an extensive study of power and delay in sequential elements.

Scan latches and flip-flops (see Chapter 9) used for testing increase the internal sequential element switched load. Unfortunately, there is a tradeoff between testability and power consumption. The decision is normally toward testability.

Clock gating can be effectively used to turn off sections of circuitry that are not required during certain time intervals. When clock gaters are inserted, the relative delay between blocks must be carefully monitored to ensure that no clock races occur (see Section 12.5.5).

Example

Figure 7.33 shows a simplified block diagram of an IEEE 802.11a Wireless Local Area Network receiver [Ryan01] . Being a packet-based system, the clocking takes advantage of the fact that the clocks only have to be applied to modules when a packet arrives. The packet activity is monitored by the packet detector, which is carefully designed to dissipate the least dynamic power as it is "listening" constantly. The rest of the circuitry is usually idle and all clocks to the remaining modules are turned off. When a packet is detected, a control block issues a start of packet signal to the clock gater that controls the synchronization module (Synch). The control block also figures the length of the packet and thus, in conjunction with the pipeline delay through modules, knows how long to turn the local clock to each module in the data pipe. The Fast Fourier Transform (FFT) and Viterbi Decoder modules are similarly controlled. The Viterbi Decoder uses a register exchange technique, which has the ramification that it has a large number of registers (128 • 64 bits). During clocking each register clocks constantly and so the decoder dissipates a significant portion of the dynamic power in the signal processing chain. Fortunately, the decoder contains very little logic between registers. Thus, it can be operated on a lower supply voltage to reduce dynamic power.

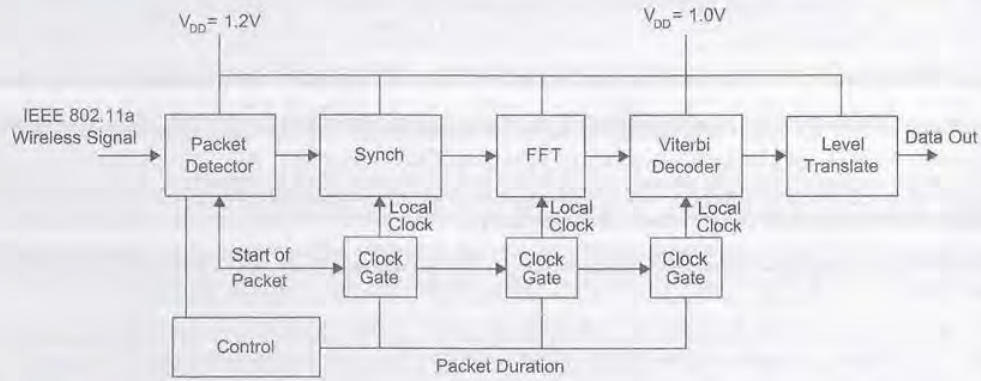


FIG 7.33 Wireless system with clock gating and variable power supply

7.4.3 Two-phase Timing Types

As discussed in Section 7.2.3, latches with two-phase nonoverlapping clocks (ϕ_1 and ϕ_2) are attractive for class projects because with an adequately long clock period and sufficiently great nonoverlap, they are guaranteed to be safe from both setup and hold problems as long as they are used correctly. Logic must be divided into phases 1 and 2. Signals can only interact with other signals in the same phase. Passing through a latch changes the phase of the signal. The situation becomes slightly more complicated when gated clocks

and domino circuits are mixed with the latches. [Noice83] describes a method of timing types that can be appended to signal names to keep track of which signals can be safely combined at inputs to gates and latches.

In the two-phase timing discipline, a signal can belong to either phase 1 or phase 2 and be of one of three classes: *stable*, *valid*, or *qualified clock*. A signal is said to be stable during phase 1 (*_s1*) if it settles to a value before ϕ_1 rises and remains constant until after ϕ_1 falls. It is said to be valid during phase 1 (*_v1*) if it settles to a value before ϕ_1 falls and remains at that value until after ϕ_1 falls. It is said to be a phase 1 gated or *qualified clock* (*_q1*) if it either rises and falls like ϕ_1 or remains low for the entire cycle. By definition, ϕ_1 is a *_q1* signal. Phase 2 signals are analogous. Figure 7.34 illustrates the timing of each of these types.

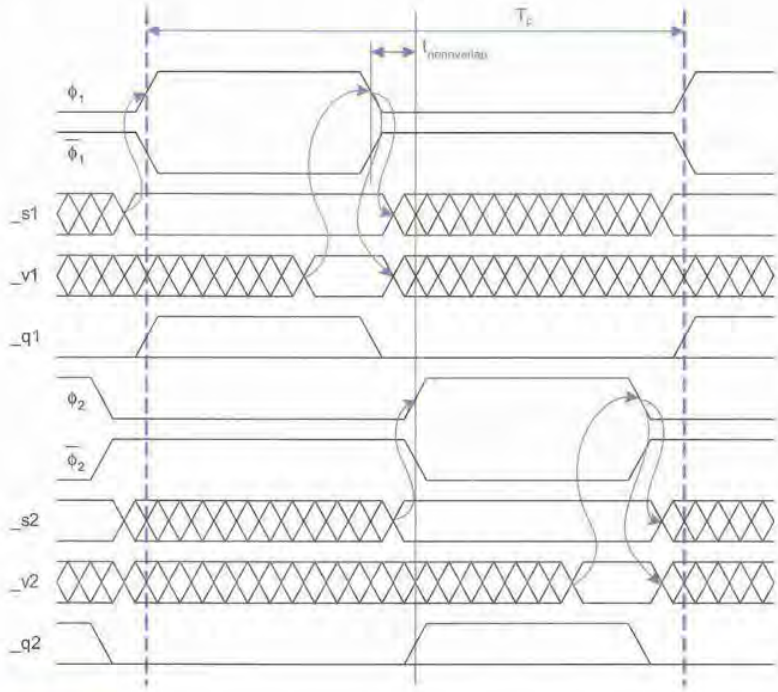


FIG 7.34 Timing types

Latches must take qualified clocks (either *_q1* or *_q2* signals) to their clock inputs. A phase 1 latch requires a *_s1* or *_v1* input (so that the input satisfies setup and hold times around the falling edge of ϕ_1) and produces a *_s2* output because the output settles while

ϕ_1 is high (before ϕ_2 rises) and does not change again until the next time ϕ_1 is high (after ϕ_2 falls). A phase 2 latch requires a *_s2* or *_v2* input and produces a *_s1* output. Qualified clocks are formed as the AND of a clock phase or another qualified clock with a stable signal belonging to the same phase. The qualifying signal must be stable to ensure there are no glitches in the clock. Qualified clocks are only used at the clock terminals of latches or dynamic logic. A block of static CMOS combinational logic requires that all inputs belong to the same phase. If all inputs are stable, the output is also stable. If any are valid, the output is valid. The phase of a domino gate is defined by the clock or qualified clock driving its evaluation transistor. The precharge transistor accepts the complement of the other phase. The inputs must be stable or valid during the evaluation phase and the output is valid during that phase because it settles before the end of the phase and does not change until precharge at the beginning of the next phase. All of these rules are illustrated in Figure 7.35. The definitions are based on the assumption that the propagation delays are short compared to the cycle time so that no time borrowing takes place; however, the connections continue to be safe even if time borrowing does occur.

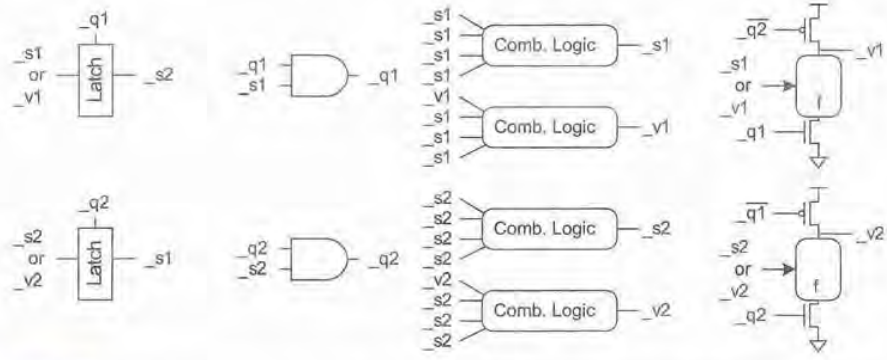


FIG 7.35 Rules for combining timing types

Figure 7.38(a) redraws the flip-flop of Figure 7.21 built from master and slave latches using two-phase nonoverlapping clocking. The flip-flop changes its output on the rising edge of ϕ_1. Both input and output are *_s2* signals. Figure 7.38(b) shows an enabled version of the flip-flop using clock gating. The enable signal to the slave must be *_s1* to prevent glitches on the qualified clock; in other words, the enable must not change while ϕ_1 is high. If the system is built primarily from flip-flops with *_s2* outputs, the enable must be delayed through a phase 2 latch to become *_s1*. Alternatively, the master (ϕ_2) latch could be enabled, but this requires that the enable set up half a cycle earlier.

Example

Annotate each of the signals in Figure 7.36 with its timing type. If the circuit contains any illegal connections, identify the problems and explain why the connections could cause malfunctions.

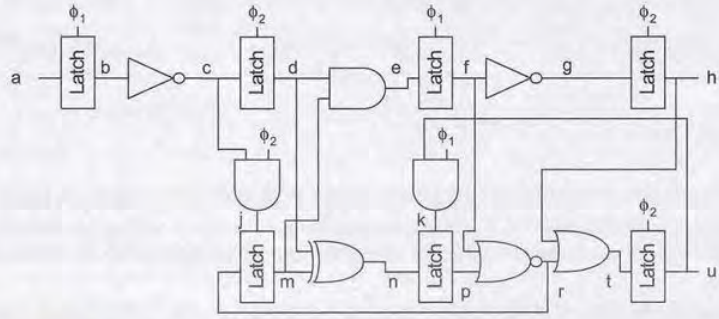


FIG 7.36 Example circuit for timing type checking

Solution: Figure 7.37 shows the timing types of each signal. *t_??* is the OR of *h_s1* and *r_s2*. Hence, it might change after the rising edge of ϕ_2 or ϕ_1. Excessive clock skew on ϕ_2 could cause a hold time violation, affecting the result seen at *u_s1*.

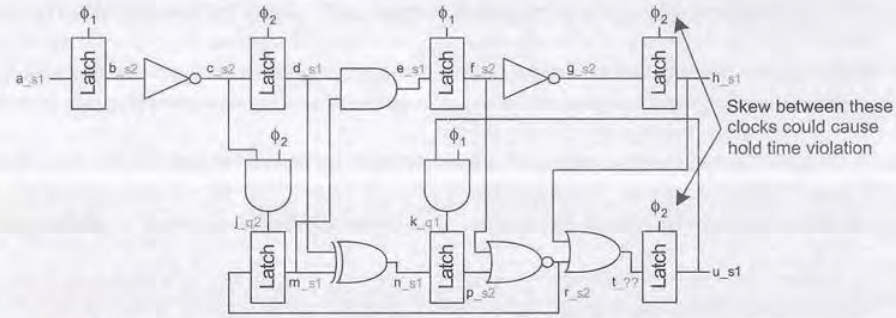


FIG 7.37 Annotated circuit showing timing types

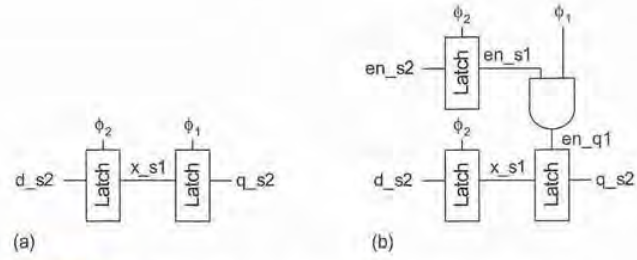


FIG 7.38 Flip-flops using two-phase nonoverlapping clocks

Even when conventional two-phase latches with 50% duty cycles are used, the timing types are still convenient to track which signals can interact. Typically, one distributes a single 50% duty cycle clock *clk* and locally generates its complement $\overline{clk}$. In such a case, *clk* plays the role of ϕ_1 and $\overline{\phi}_2$ while $\overline{clk}$ plays the role of $\overline{\phi}_1$ and ϕ_2. This means that both the precharge and evaluate transistors of dynamic gates receive the same signal. Because there is no nonoverlap, you must analyze each path to ensure no hold problems exist. In particular, be careful to guarantee a stable enable signal for gated clocks.

7.4.4 Characterizing Sequencing Element Delays

Previous sections have derived sequencing element performance in terms of the setup and hold times and propagation and contamination delays. These delays are interrelated and are used for budgeting purposes. For example, a flip-flop might still capture its input properly if the data changes slightly less than a setup time before the clock edge. However, the clock-to-Q delay might be quite long in this situation. If we call t_{DC} the time that the data actually sets up before the clock edge and t_{CQ} the actual delay from clock to Q, we could define t_{setup} as the smallest value of t_{DC} such that $t_{CQ} \leq t_{pcq}$. Moreover, we could choose t_{pcq} to minimize the sequencing overhead $t_{setup} + t_{pcq}$. In this section we will explore how to characterize these delays through simulation.

Figure 7.39 shows the timing of a conventional static edge-triggered flip-flop from Figure 7.19(b). Delays are normalized to a 75 ps FO4 inverter. The actual *clk*-to-Q (t_{CQ}) and D-to-Q (t_{DQ}) delays for a rising input are plotted against the D-to-*clk* (t_{DC}) delay, i.e., how long the data arrived before the clock rises. If the data arrives long before the clock, t_{CQ} is short and essentially independent of t_{DC} delay. $t_{DQ} = t_{DC} + t_{CQ}$, so it increases linearly as data arrives earlier because the data is blocked and waits for the clock before proceeding. As the data arrives closer to the clock, t_{CQ} begins to rise. However, t_{DQ} initially decreases and reaches a minimum when t_{CQ} has a slope of −1 (note the axes are not to scale). Therefore, let us define the setup time t_{setup} as t_{DC} at which this minimum t_{DQ} occurs and the propagation delay t_{pcq} as t_{CQ} at this time. The contamination delay t_{ccq} is the minimum t_{CQ} that occurs when the input arrives early.

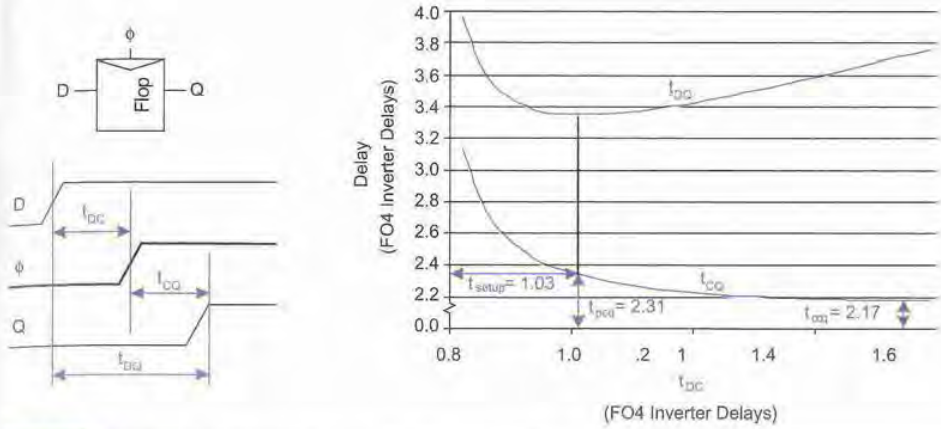


FIG 7.39 Flip-flop delay vs. data arrival time

In general, the delays can differ for inputs of '0' and '1.' Figure 7.40 plots t_{CQ} vs. t_{DC} for the four combinations of rising and falling D and Q. The setup times Δ_{DC0} and Δ_{DC1} are the times that D must fall or rise, respectively, before the clock so that the data is properly captured with the least possible t_{DQ}. Observe that this flip-flop has a longer setup time but shorter propagation delay for low inputs than high inputs. The hold times t_{hold0} and t_{hold1} are the times that D must rise or fall, respectively, after the clock so that the old value of '0' or '1' is captured instead of the new value. Observe that the hold times are typically negative. The contamination delay $t_{cq0/1}$ again is the lowest possible t_{CQ} and occurs when the input changes well before the clock edge. When only one delay is quoted for a flip-flop timing parameter, it is customarily the worst of the '0' and '1' delays.

The *aperture width* t_a is the width of the window around the clock edge during which the data must not transition if the flip-flop is to produce the correct output with a propagation delay less than t_{pcq}. The aperture times for rising and falling inputs are

$$\begin{aligned} t_{ar} &= t_{setup1} + t_{hold0} \\ t_{af} &= t_{setup0} + t_{hold1} \end{aligned} \tag{7.20}$$

If the data transitions within the aperture, Q can become metastable and take an unbounded amount of time to settle. Metastability is discussed further in Section 7.6.1.

If D is a very short pulse, the flip-flop may fail to capture it even if D is stable during the setup and hold times around the rising clock edge. Similarly, if the clock pulse is too short, the flip-flop may fail to capture stable data. Well-characterized libraries sometimes specify minimum pulse widths for the clock and/or data as well as setup and hold times.

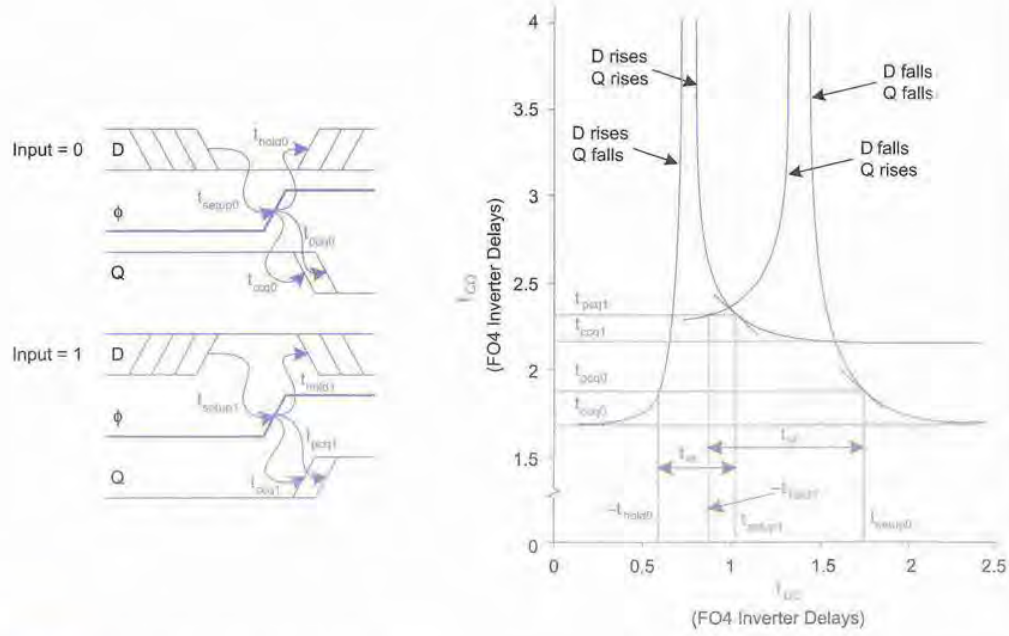


FIG 7.40 Flip-flop setup and hold times

Level-sensitive latches have somewhat different timing constraints because of their transparency, as shown in Figure 7.41 for a conventional static latch from Figure 7.17(g) using a pulse width of 4 FO4 inverter delays. As with an edge-triggered flip-flop, if the data arrives before the clock rises ($t_{DCr} > 0$), it must wait for the clock. In this region the clock-to-Q t_{CrQ} delay is nearly constant and t_{DQ} increases as the data arrives earlier. If the data arrives after the clock rises while the latch is transparent, t_{DQ} is essentially independent of the arrival time. The data must set up before the falling edge of the clock. The second set of labels on the X-axis indicates the D-to-clk fall time t_{DCf}. As the data arrives too close to the falling edge, t_{DQ} increases. Now to achieve low t_{DQ}, we choose the setup time before the knee of the curve, e.g., 5% greater than its minimum value. The setup time is measured relative to the falling edge of the clock. If the data changes less than a hold time after the falling edge of the clock, Q may momentarily glitch. Thus, the hold time t_{hold} for a latch is defined to be $-t_{DCf}$ for which Q displays a negligible glitch.

Pulsed latches have setup and hold times measured around the falling edge of the clock. However, designers often wish to treat pulsed latches as edge-triggered flip-flops from the perspective of timing analysis. Therefore, we can define "virtual" setup and hold times relative to the rising clock edge [Stojanovic99]. For example, the pulsed latch in

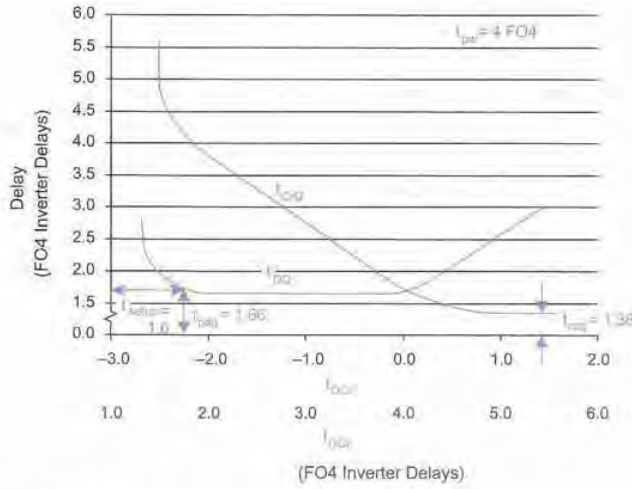


FIG 7.41 Latch delay vs. data arrival time

Figure 7.41 has $t_{\text{setup–virtual}} = t_{\text{setup}} - t_{pw} = -2.4$ FO4 but $t_{pcq\text{–virtual}} = t_{pdq} + (t_{pw} - t_{\text{setup}}) = 4.06$ FO4 so the total sequencing overhead of $t_{pdq} = t_{\text{setup–virtual}} + t_{pcq\text{–virtual}}$ is unaffected by the change of reference or pulse width. The virtual hold time is now $t_{\text{hold–virtual}} = t_{\text{hold}} + t_{pw} = 2.6$ FO4, which is positive as one should expect because the input must hold long after the rising edge of the clock.

The delays vary with input slope, voltage, and temperature. The contamination delay should be measured in the environment where it is shortest while the setup and hold times and propagation delay should be measured in the environment where it is longest.

The designer can trade off setup time, hold time, and propagation delay. Figure 7.42 shows the effects of adding delay t_{buf} to the clock, D, or Q terminals of a flip-flop. Recall that the sequencing overhead depends on the sum of the setup time and propagation delay while the minimum delay between flip-flops depends on the hold time less the contamination delay. Adding delay on either the input or output eases min-delay at the expense of sequencing overhead. Many standard cell libraries intentionally use slow flip-flops so that logic designers do not have to worry about hold time violations. Adding delay on the clock simply shifts when the flop activates. The sequencing overhead does not change, but the system can accommodate more logic in the previous cycle and less in the next cycle. This is similar to time borrowing in latch-based systems, but must be done intentionally by adjusting the clock rather than opportunistically by taking advantage of transparency. Some authors refer to delaying the clock as *intentional clock skew*. This book reserves the term *clock skew* for uncertainty in the clock arrival times.

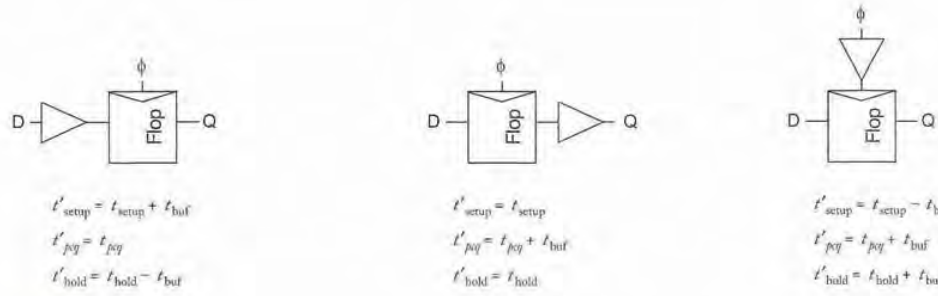


FIG 7.42 Delay tradeoffs



7.5 Sequencing Dynamic Circuits

Dynamic and domino circuits operate in two steps: precharge and evaluation. Ideally, the delay of a path should be the sum of the evaluation delays of each gate along the path. This requires some careful sequencing to hide the precharge time. Traditional domino circuits discussed in Section 7.5.1 divide the cycle into two half-cycles. One phase evaluates while the other precharges, then the other evaluates while the first precharges. Transparent latches hold the result of each phase while it precharges. This scheme hides the precharge time but introduces substantial sequencing overhead because of the latch delays and setup time. A variety of skew-tolerant domino circuit schemes described in Section 7.5.2 use overlapping clocks to eliminate the latches and the sequencing overhead. Section 7.5.3 expands on skew-tolerant domino clocking for unfooted dynamic gates.

Recall that dynamic gates require that inputs be monotonically rising during evaluation. They produce monotonically falling outputs. Domino gates consist of dynamic gates followed by inverting static gates to produce monotonically rising outputs. Because of these two levels of inversion, domino gates can only compute noninverting logic functions. We have seen that dual-rail domino gets around this problem by accepting both true and complementary inputs and producing both true and complementary outputs. Dual-rail domino is not always practical. For example, dynamic logic is very efficient for building wide NOR structures because the logical effort is independent of the number of inputs. However, the complementary structure is a tall NAND, which is quite inefficient. When inverting functions are required, an alternative is to use a dynamic gate that produces monotonically falling outputs, but delays the clock to the subsequent dynamic gate so that the inputs are stable by the time the gate enters evaluation. Section 7.5.4 explores a selection of these *nonmonotonic* techniques.

7.5.1 Traditional Domino Circuits

Figure 7.43(a) shows a traditional domino clocking scheme. While the clock is high, the first half-cycle evaluates and the second precharges. While the clock is low, the second evaluates and the first precharges. With this ping-pong approach, the precharge time does not appear in the critical path. The inverting latches hold the result of one half-cycle while that half-cycle precharges and the next evaluates. The data must arrive at the first half-cycle latch a setup time before the clock falls. It propagates through the latch, so the overhead of each latch is the maximum of its setup time and D-to-Q propagation delay [Harris97]. Assuming the propagation delay is longer, the time available for computation in each cycle is

$$t_{pd} = T_c - 2t_{pdq} \tag{7.21}$$

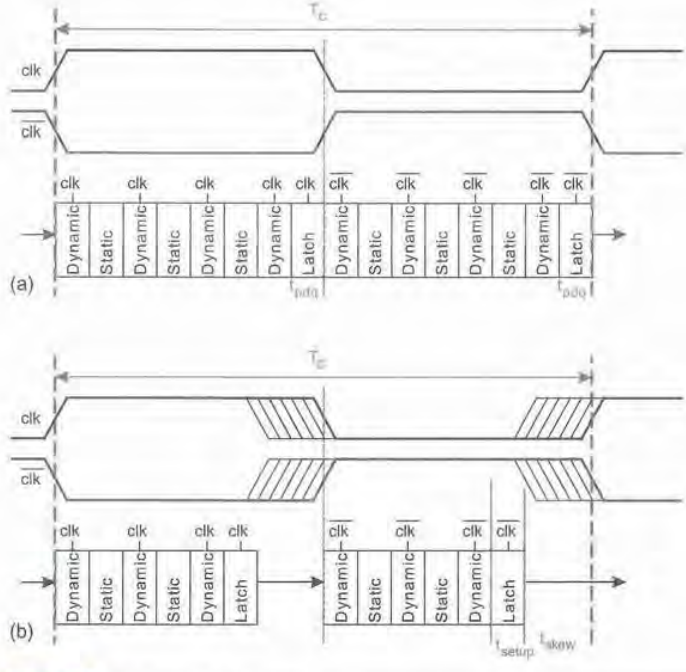


FIG 7.43 Traditional domino circuits

Figure 7.43(b) shows the pipeline with clock skew. Data is launched into the first dynamic gate of each cycle on the rising edge of the clock and must set up before the falling edge. Hence, clock skew cuts into the time available for computation in each half-cycle. This is even worse than flip-flops, which pay clock skew once per cycle. Assuming the skew and setup time are greater than the propagation delay, the time for computation becomes

$$t_{pd} = T_c - 2t_{\text{setup}} - 2t_{\text{skew}} \tag{7.22}$$

Moreover, like flip-flops, traditional domino circuits suffer from imbalanced logic. Gates cannot borrow time into the next half-cycle, so a fraction of a gate delay at the end of each half-cycle may be wasted. This penalty is hard to quantify, but clearly the ability to borrow time intentionally or opportunistically would help performance.

In summary, traditional domino circuits have high sequencing overhead from latch delay, clock skew, and imbalanced logic. For heavily pipelined systems with short cycle times, this overhead can be such a large fraction of the cycle time that it wipes out the performance advantage that domino was intended to bring. Therefore, many system designers have developed skew-tolerant domino sequencing techniques with lower overhead. The next section is devoted to these techniques.

7.5.2 Skew-tolerant Domino Circuits

Traditional domino circuits have such high sequencing overhead because they have a hard edge in each half-cycle: The first domino gate does not begin evaluating until the rising edge of the clock, but the result must set up at the latch before the falling edge of the clock. If we could remove the latch, we could soften the falling edge and cut the overhead. The latch serves two functions: (1) to prevent nonmonotonic signals from entering the next domino gate while it evaluates and (2) to hold the results of the half-cycle while it precharges and the next half-cycle evaluates. Within domino pipelines, all the signals are monotonic, so the first function is unnecessary. Moreover, after the next half-cycle has had sufficient time to evaluate using the results of the first half-cycle, the first half-cycle can precharge without impacting the output of the next.

Figure 7.44 illustrates the implications of eliminating the latch. In general, let logic be divided into N phases rather than two half-cycles. Figure 7.44(a) shows the last domino gate in phase 1 driving the first gate in phase 2. Figure 7.44(b) shows that the circuit fails if the clocks are nonoverlapping. When ϕ_1 falls, nodes a and b precharge high and low, respectively. When ϕ_2 rises, the input to the first domino gate in this phase has already fallen, so c will never discharge and the circuit loses information. Figure 7.44(c) shows that the second dynamic gate receives the correct information if the clocks overlap. Now ϕ_2 rises while b still holds its correct value. Therefore, the first phase 2 domino gate can evaluate using the results of phase 1. When ϕ_1 falls and b precharges low, c holds its value. Without a keeper, c can float either high or low. Figure 7.45 shows a *full keeper* consisting of weak cross-coupled inverters to hold the output either high or low. In summary, the

latches can be eliminated at phase boundaries as long as the clocks overlap and the first dynamic gate of each phase uses a full keeper.

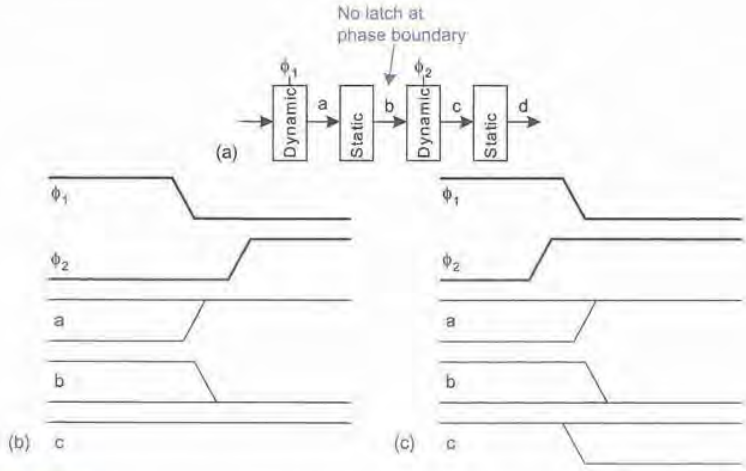


FIG 7.44 Eliminating latches in skew-tolerant domino circuits

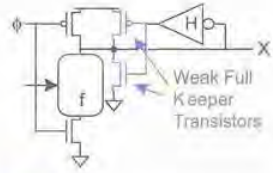


FIG 7.45 Full keeper

In general, as long as the clock overlap is long enough that the second phase can evaluate before the first precharges, the latch between phases is unnecessary. Let us define t_{hold} as the required overlap so that the second phase can evaluate before the first precharges. It is typically a small negative number because the dynamic gate evaluation is fast, but precharge is slow and must ripple through the static stage. The clocks must overlap enough such that they still overlap by t_{hold} even under worst-case clock skew[1]. The sequencing overhead is zero because data propagates from one domino gate to the next without waiting at any sequencing elements. Therefore, we use the generic name *skew-tolerant domino*

[1]Do not confuse this t_{hold}, the amount of time that the clocks must overlap in a skew-tolerant domino pipeline, with t_{hold} on a sequencing element, the time that the data must remain stable after the clock edge.

for domino circuits with overlapping clocks that eliminate the latches between phases [Harris01a]. Using more clock phases also helps spread the power consumption across the cycle rather than drawing large noisy current spikes on the two clock edges.

Skew-tolerant domino circuits can also borrow time from one phase into the next, as illustrated in Figure 7.46. Nominally each phase in this example occupies half the cycle. However, a ϕ_1 dynamic gate can borrow time into phase 2 if that is convenient because both clocks are simultaneously high. If one phase overlaps the next by $t_{overlap}$ less any clock skew, the maximum time that gates in one phase can borrow into time nominally allocated for the next is

$$t_{borrow} = t_{overlap} - t_{hold} - t_{skew} \tag{7.23}$$

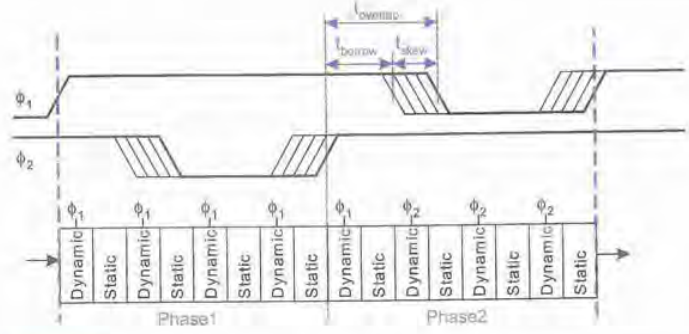


FIG 7.46 Time borrowing in skew-tolerant domino circuits

[Williams91] observed that self-timed pipelines could use overlapping clocks to eliminate latches, but such asynchronous design has not been widely adopted. The Alpha 21164 overlapped clocks in the ALU to eliminate the mid-cycle latch and improve performance [Bowhill95]. Since then, most high-performance synchronous systems using domino have employed some form of skew-tolerant domino to avoid the high sequencing overhead of traditional domino.

There are many ways to produce overlapping clocks. In general, you can use N separate clock phases. Each phase can use 50% duty-cycle waveforms or can stretch the falling edge for even greater overlap. Generating multiple overlapping clocks with low skew is a challenge. The remainder of this section describes a number of methods that have been used successfully.

7.5.2.1 Two-phase Skew-tolerant Domino and OTB Domino Figure 7.47 shows a clock generator for the two-phase skew-tolerant domino system from Figure 7.46. The generator uses *clock choppers* (also called *clock stretchers*) that delay the falling edge to provide the overlap. A potential problem with two-phase systems is that if a phase of logic

has short contamination delay, the data can race through while both clocks are high.

Opportunistic Time Borrowing (OTB) Domino addresses the race problem by introducing two more clocks (*clk* and *clkb*) with 50% duty cycles that are used on the first gate of each half-cycle, as shown in Figure 7.48. These first gates block data that arrives too early so that it will not race ahead. The delayed clocks *clkd* and *clkbd* play the role of ϕ_1 and ϕ_2. OTB domino was used on the Itanium processor [Rusu00]. However, OTB domino has relatively short overlap and time borrowing capability set by the delay of the clock chopper. The next section describes how to achieve better performance with four phases.

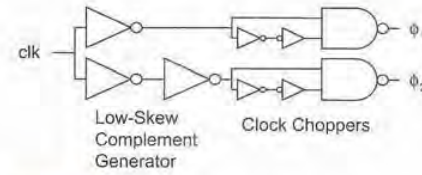


FIG 7.47 Two-phase skew-tolerant domino clock generator

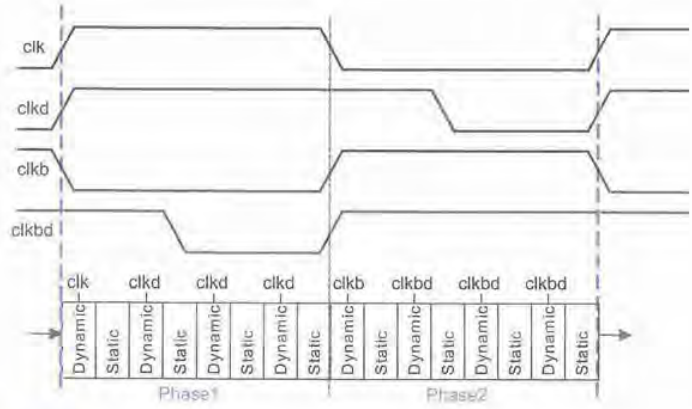


FIG 7.48 OTB domino

7.5.2.2 Four-phase Skew-tolerant Domino Figure 7.49 shows a four-phase skew-tolerant domino system. Each of the phases has a 50% duty cycle and is spaced a quarter cycle after the previous one, so the nominal overlap is a quarter cycle. The clocks are never all simultaneously high so race problems are solved unless skew approaches a quarter cycle. According to EQ (7.23), the maximum time available for borrowing from one phase to the next is

$$t_{\text{borrow}} = T_c / 4 - t_{\text{hold}} - t_{\text{skew}} \tag{7.24}$$

Figure 7.50(a) shows a local clock generator producing the four phases. ϕ_1 and ϕ_3 are produced directly from the global clock and its complement. ϕ_2 and ϕ_4 are delayed by buffers with nominal quarter cycle latency. By using both clock edges, each phase is guaranteed to overlap the next phase independent of clock frequency. Variations in these buffer delays with

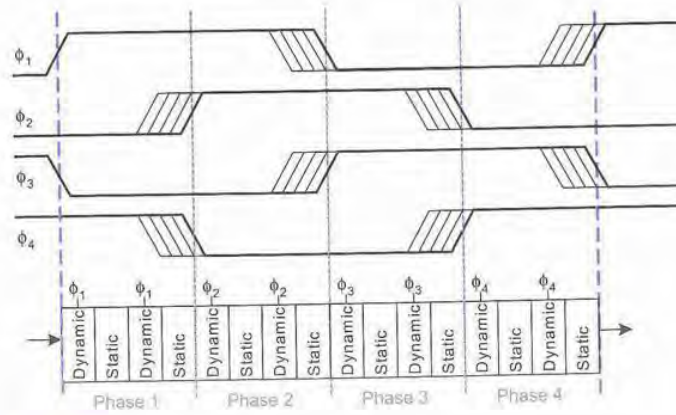


FIG 7.49 Four-phase skew-tolerant domino

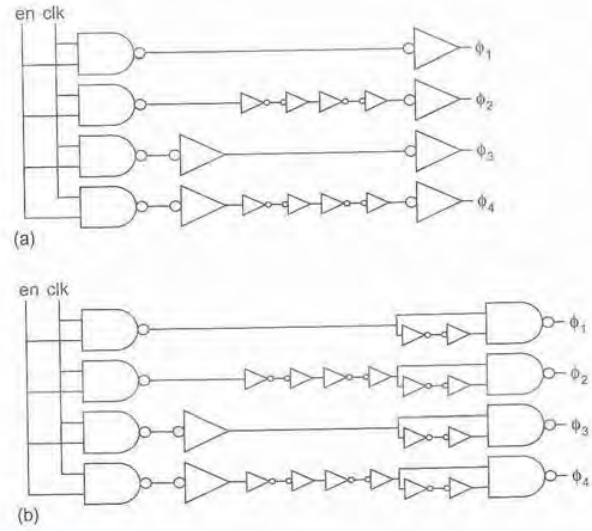


FIG 7.50 Clock generator for four-phase skew-tolerant domino

process, voltage, and temperature can reduce the overlap and available time for borrowing. To avoid excessive pessimism, remember that in the fast corner where overlaps are short, the rest of the gates are also faster. The clock generator also includes a built-in enable.

In general, clock choppers can be used to produce even greater overlap at the expense of greater race concerns. The Itanium II uses four-phase skew-tolerant domino with duty cycles exceeding 50% [Naffziger02]. Figure 7.50(b) shows a four-phase clock generator with clock choppers to provide longer duty cycles. [Harris01a] describes four-phase circuit methodology in much more detail, including testability and a generalization of timing types from Section 7.4.3.

7.5.2.3 *N*-phase Skew-tolerant Domino Another approach to domino clocking is to use a chain of buffers to produce a unique phase for each level of logic in a cycle. Figure 7.51 shows two ways of producing these phases. In Figure 7.51(a), half the phases are generated off the rising edge of the clock and half off the falling edge. In this way, each phase is guaranteed to overlap the next independent of cycle time. In Figure 7.51(b), all of the phases are generated off the rising edge. If the clock period is long, the final phase must delay its falling edge to guarantee it will still overlap the first phase of the next cycle. The SR latch ensures that the last phase, ϕ_6, will not rise until after *clk* falls (to avoid min-delay problems) and will not fall until after *clk* rises (to ensure overlap of ϕ_1).

A number of design teams have independently developed these techniques. The approach of one phase for each level of logic has been called *Delayed Reset* (IBM [Nowka98]), *Cascaded Reset* (IBM [Silberman98]), and *Delayed Clocking* (Sun [Heald00]). The phase generator for Cascaded Reset domino is well suited to driving footless dynamic gates and will be discussed further in Section 7.5.3.

7.5.2.4 Self-resetting (Postcharge) Domino In the methods examined so far, the timing of the precharge operation has been controlled by the clock generator. An alternative approach, called *Self-Resetting* or *Postcharge Domino*, is to control the precharge based on the output of the domino gate. Figure 7.52 shows a simple self-resetting domino gate. When the domino gate evaluates and the output rises, a timing chain produces a precharge signal $\overline{reset}$ to precharge the dynamic stage (and possibly assist pulling the HI-skew inverter low, particularly if the inverter is highly skewed). Once the output has fallen, the precharge signal turns off the precharge transistors and the gate is ready to evaluate again. The input must have fallen before the gate reenters evaluation so the gate does not repeatedly pulse on a steady input. Therefore, self-resetting gates accept input pulses and produce output pulses whose duration of five gate delays is determined by the delay of the timing chain. As long as the first inverter in the timing chain is small compared to the rest of the load on node *Y*, its extra loading has negligible impact on performance.

Self-resetting gates save power because they reduce the loading on the clock. Moreover, they only toggle the precharge signal when the gate evaluates low. In Section 11.2.2, we will see that this is particularly useful for RAM decoders. Only one of many word lines in a RAM will rise on each cycle, so a self-resetting decoder saves power by resetting only that line without applying precharge to the other word line drivers. For example, an IBM SRAM [Chappell91], the Intergraph Clipper cache [Heald93], and the Sun UltraSparc I cache [Heald98] use self-resetting gates[2].

[2]Confusingly referred to as "delayed reset" by Sun in [Lev95, Heald98].

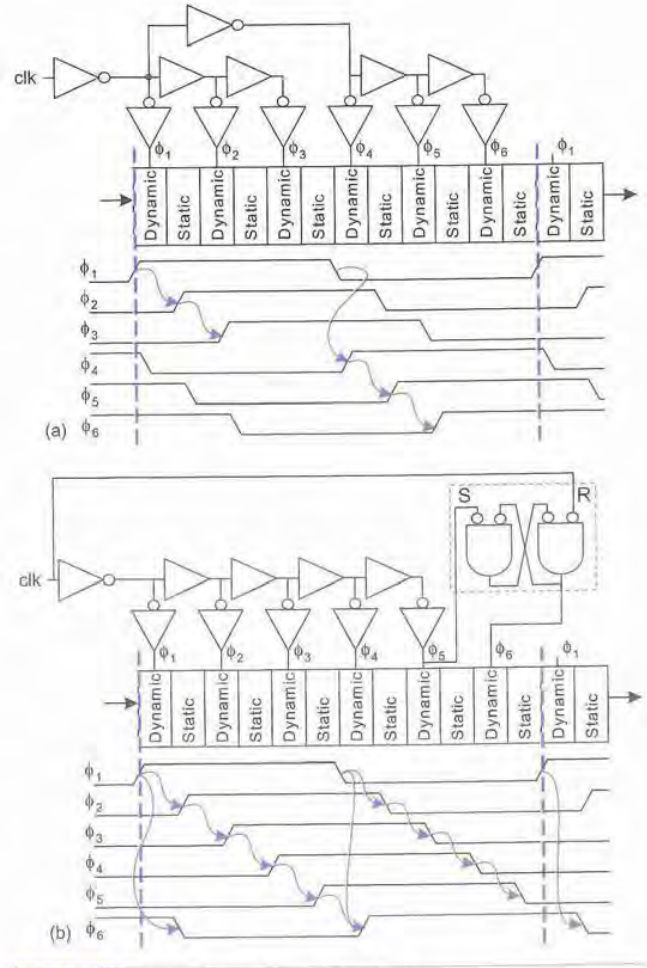


FIG 7.51 N-phase skew-tolerant domino

Self-resetting AND gates in these decoders often receive the address inputs as static levels rather than pulses. *Predicated self-resetting* AND gates [Amrutur01] wait for the input to fall before precharging the output to stretch the pulse width and prevent multiple output pulses when the input is held high, as shown in Figure 7.53. The first inverter in the timing chain is replaced by a *generalized Muller C-element*, shown in blue, whose output does not rise until both Y and one of the inputs have fallen. This only works for functions such as AND or OR-AND where one of the inputs is in series with all of the others.

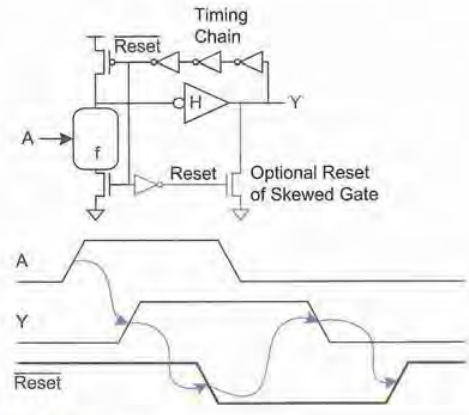


FIG 7.52 Self-resetting gate

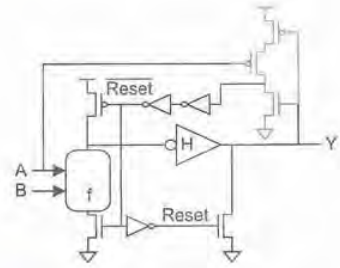


FIG 7.53 Predicated self-resetting gate

[Proebsting91] applies self-resetting techniques to NORA gates for buffers and memory decoders. Figure 7.54 shows an example of a postcharged buffer for a memory chip. It rapidly amplifies the chip select signal CS and provides a series of pulses that serve as clocks for large (multi-pF) loads across the chip. The clock chopper produces a pulse to trigger the first stage of the buffer. The buffer consists of alternating extremely HI- and LO-skew inverters with logical efforts of approximately 2/3 and 1/3, respectively. Each inverter also receives a postcharge signal from a subsequent stage to assist the weak device in resetting the gate. The very small transistor serves as a keeper, so the gates can be viewed as unfooted NTP dynamic nMOS and pMOS inverters. Forward moving pulses trigger each gate. Signals from four stages ahead feed back to postcharge the gate. The buffer is roughly twice as fast as an ordinary chain of inverters because of the lower logical efforts. It also avoids the need for an external clock to precharge the dynamic gates.

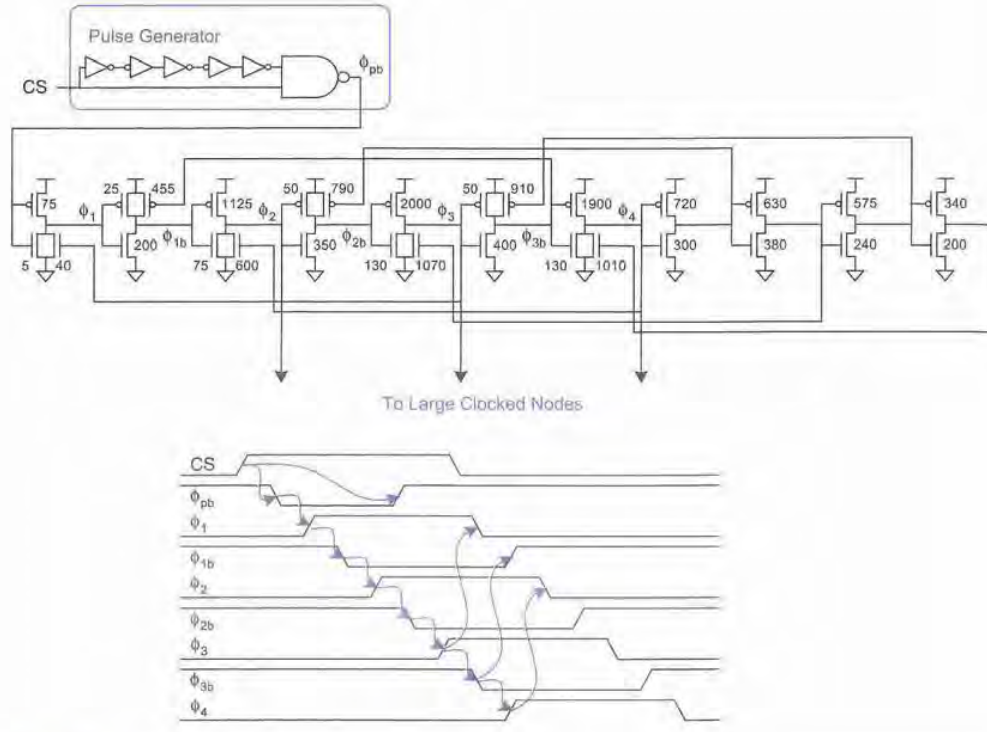


FIG 7.54 Postcharged buffer

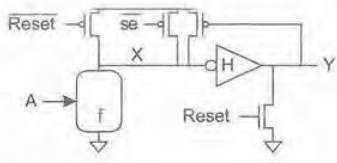


FIG 7.55 SRCMOS

IBM has developed an extensive methodology for self-resetting domino gates called *SRCMOS* [Haring96] that has been applied to circuits including a register file [Hwang99a], 64-bit adder [Hwang99b], and the S/390 G4 CPU cache [Webb97]. SRCMOS gates are typically unfooted dynamic gates followed by highly skewed static inverters, as shown in Figure 7.55. True and complementary reset signals precharge the dynamic stage and help pull the output low. An additional weak *static evaluation* transistor converts the gate into pseudo-nMOS when the global $\overline{se}$ signal is asserted to assist with testing and low-frequency debug. The inputs and outputs are pulses. The reset signals are generated from the gate outputs or from a global reset.

To avoid the overhead and timing constraints of reset circuitry on every gate, the reset signals can be derived from the output of the first gate in a pipeline and delayed through buffers to reset subsequent gates. Figure 7.56 shows an example of an SRCMOS macro adapted from [Hwang99b]. The upper portion represents an abstract

datapath. None of the keepers or static evaluation devices are shown. The center is a timing chain that provides reset pulses to each gate. These pulses may be viewed as N-phase skew-tolerant domino clocks. The bottom shows a pulse generator. In normal operation, the power-on reset signal is low and the static evaluation signal $\overline{se}$ high. Assume that all of the gates have been precharged. When the input pulse arrives at A, the datapath will begin evaluating. The first stage must use dual-rail (or in general, 1-of-N hot) encoding so that Y_1_b or Y_1_l will rise when the stage has completed. This triggers the pulse generator, which raises the done signal and initiates a reset. A wave of low-going reset pulses propagates along the timing chain to precharge each gate. One of the reset pulses also precharges the pulse generator, terminating the reset operation. At this point, the datapath can accept a new input pulse. If the data idles low, none of the nodes toggle and the circuit consumes no dynamic power.

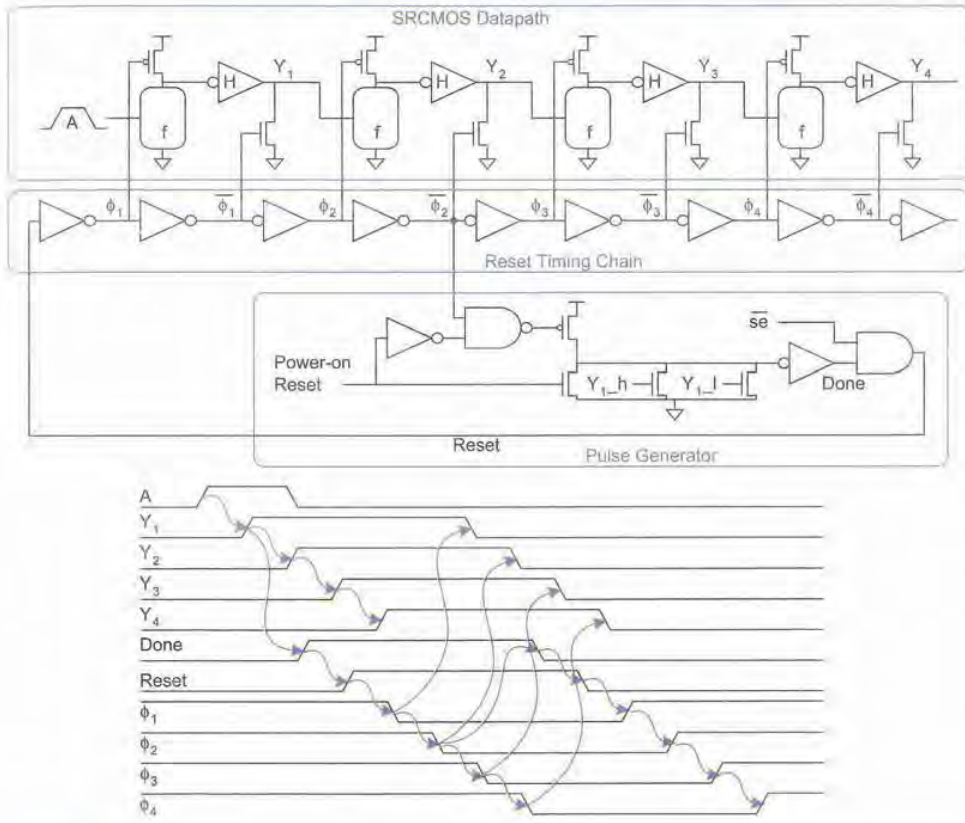


FIG 7.56 SRCMOS macro

The power-on reset forces *done* and *reset* high to initialize the pipeline at startup. When the static evaluation signal is asserted, the reset pulses are inhibited. In this mode, the datapath gates behave as pseudo-nMOS rather than dynamic, permitting low-frequency test and debug.

Self-resetting gates require very careful design because they act on pulses rather than static levels. Some of the timing checks include [Narayanan96]:

- Pulse overlap constraints
  - Pulses arriving at series transistors must overlap so the dynamic gate can pull down through all the transistors
- Pulse width constraints
  - Pulses must be wide enough for a gate to evaluate
- Collision avoidance constraints
  - Pulses must not arrive at dynamic gates while the gates are being precharged

The Pentium 4 uses yet another form of self-resetting domino called *Globally-Reset Domino with Self-Terminating Precharge* (Global STP) to achieve very fast cycle times [Hinton01]. The first design operated at 2 GHz in a 180 nm process (< 16 FO4 inverter delays / cycle). More remarkably, the integer execution was double-pumped to 4 GHz using Global STP domino. Each cycle has time for only eight gate delays: four dynamic gates and four static gates.

Figure 7.57 illustrates the Global STP circuits. A frequency doubler generates pulses off both edges of the clock to drive the datapath. Each stage of the datapath is a domino gate with a keeper (k) and precharge transistor (p). The gates are shown using HI-skew inverters but could use any HI-skew inverting static gate. The small NAND gates save power by only turning on the precharge transistor if the dynamic gate had evaluated low. The first stage requires a foot to only sample the input while ϕ_1 is high. The last stage also uses a foot, a full keeper, and more complex reset circuitry to stretch the width of the output pulse so that it is compatible with static logic. The reset timing chain must be carefully designed to produce precharge clocks properly aligned to the data. For example, ϕ_3 should be timed to rise close to the time Y_1 evaluates high to prevent contention between the precharge transistor and the pulldown network. Global STP circuit design can be a very labor-intensive process. IBM used a similar timing chain without the frequency doubler on an experimental 1 GHz PowerPC chip and called the method *cascaded reset* [Silberman98].

7.5.3 Unfooted Domino Gate Timing

Unfooted domino gates have a lower logical effort than footed gates because they eliminate the clocked evaluation transistor. They also reduce clock loading, which can save power. However, at least one input in each series stack must be OFF during precharge to prevent crowbar current flowing from V_{DD} to GND through the precharge device and ON stack.

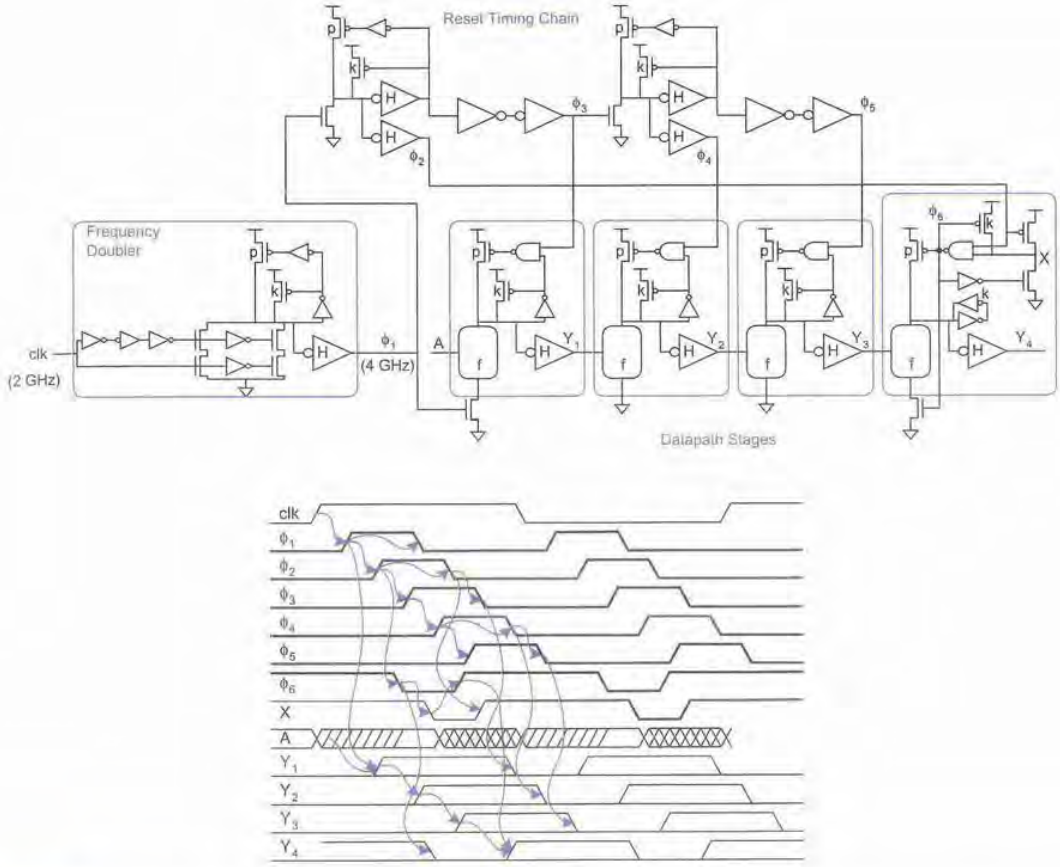


FIG 7.57 Global STP macro

The easiest way to ensure this is to require that the input come from a previous domino gate that has completed precharge before the footless gate begins precharge. Moreover, the previous gate must not output a '1' again until the unfooted gate is in evaluation.

One way to ensure these constraints is to delay the falling edge of clocks to footless gates as shown in Figure 7.58(a). The first domino gate is footed to accept static inputs that might be high during precharge. The subsequent unfooted gates begin evaluating at the same time but have their precharge delayed until the previous gate has precharged. Multiple delayed clocks can be used to allow multiple stages of unfooted gates. For example, the Itanium II processor uses one footed gate followed by four unfooted gates in the first half-cycle of the execution stage for the 64-bit adder [Fetzer02]. If the falling edge is

delayed too much in a system with a short clock period, the clock may not be low long enough to fully precharge the gate. Figure 7.58(b) shows an OTB domino system that uses only one delayed clock but allows every other domino gate to be footless. The delayed clocks can be produced with clock choppers as was shown in Figure 7.47.

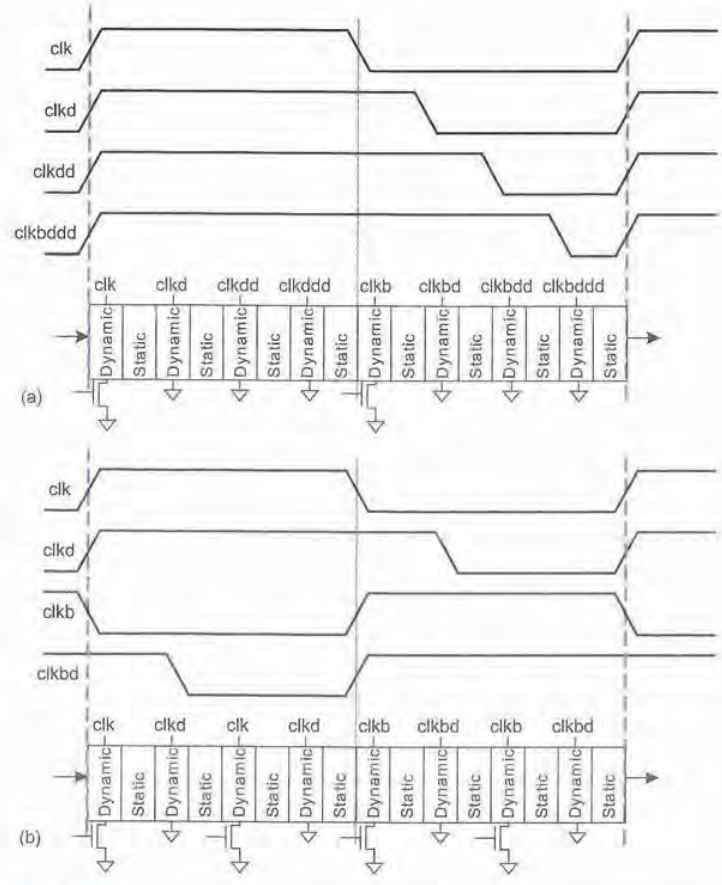


FIG 7.58 Clocking domino pipelines with unfooted gates

The precharge time on each of the delayed phases in Figure 7.58(a) becomes shorter because the falling edge is delayed but the rising edge is not. It is not strictly necessary for all the rising edges to coincide; some delay can be accepted so long as the delayed clock is in evaluation by the time the input arrives at its unfooted gate. Figure 7.59 shows a *delayed precharge* clock buffer [Colwell95] used on the Pentium II. The delayed clocks are produced with skewed buffers that have fast rising edges but slower falling edges.

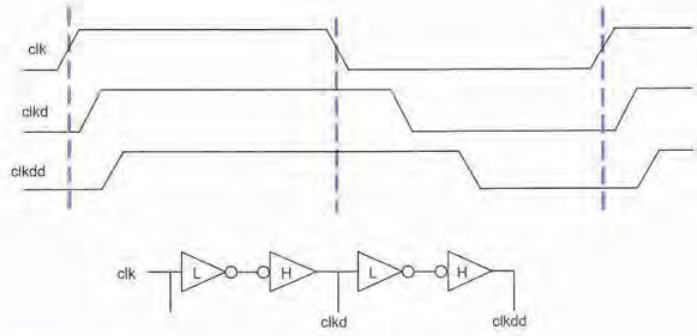


FIG 7.59 Delayed precharge clock buffer

Self-resetting domino also works well with unfooted gates. The inputs are pulses rather than levels. As long as the pulses are only high while the gate is in evaluation, no precharge contention will occur. For example, Figure 7.54, Figure 7.56, and Figure 7.57 illustrate self-resetting circuits with unfooted gates in some or all of the stages.

The consequence of precharging an unfooted gate before its input has fully fallen low is excess power consumption rather than outright circuit failure. Therefore, delays can be set to nominally avoid precharge contention, yet accept that, under worst-case clock skew, contention may occur in a few places.

7.5.4 Nonmonotonic Techniques

The monotonicity requirement forces domino gates to perform only noninverting functions. Dual-rail domino accepts true and complementary inputs and produces true and complementary outputs. This works reasonably well for circuits such as XORs at the expense of twice the hardware. However, domino is particularly poorly suited to wide NOR functions. Figure 7.60 compares a dual-rail domino 4-input OR/NOR gate to a 4-input dynamic NOR. The dual-rail design tends to be very slow because the complementary gate is a tall NAND with a logical effort of 5/3. On the other hand, a dynamic wide NOR is very compact and has a logical effort of only 2/3. The problem is exacerbated for wider gates.

The output of a dynamic gate is monotonically falling so it cannot directly drive another dynamic gate controlled by the same clock, as was shown in Figure 6.27. However, if the rising edge of the clock for the second gate is delayed until the first gate has fully evaluated, the second gate sees a stable input and will work correctly, as shown by Figure 7.61. The primary tradeoff in such *clock-blocked* circuits is the amount of delay: If the delay is too short, the circuit will fail, but as the delay becomes longer, the circuit sacrifices the performance advantages that dynamic logic was supposed to provide. This challenge is exacerbated by process and environmental variations that require margins on the delay in the nominal case so that the circuit continues to operate correctly in the worst case.

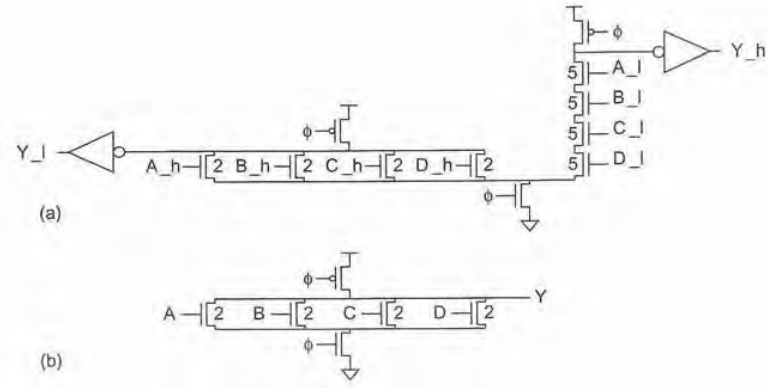


FIG 7.60 Comparison of NOR gates

Figure 7.61 also illustrates the *precharge race* problem. When X precharges while Y is still in evaluation, Y will start to fall. If ϕ_2 falls too late, Y will incorrectly glitch low. We can alleviate this problem by latching Y before X precharges or by delaying the falling edge of ϕ_1.

This section addresses a number of nonmonotonic techniques using delayed clocks to directly cascade dynamic gates and examines the margins required for matched delays.

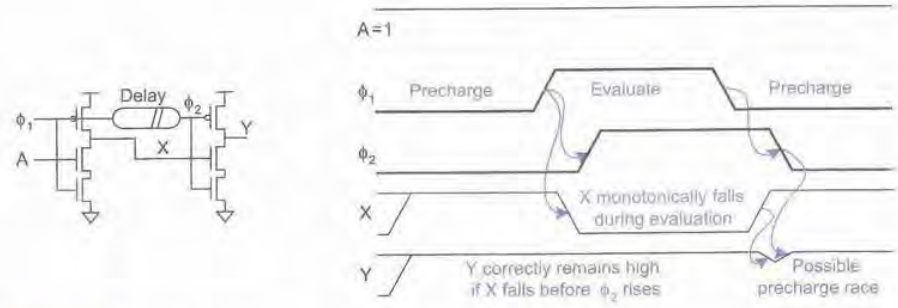


FIG 7.61 Cascading dynamic gates with a delayed clock

7.5.4.1 Delay Matching Figure 7.62 shows a number of simple delay elements. The buffer delay can be set by adjusting gate widths. The buffer with transmission gates provides flexibility for longer delays. The current-starved inverter and switched capacitance designs use a reference voltage to adjust the delay externally. The digitally controlled current-starved inverter uses several digital signals rather than an analog voltage to adjust delay.

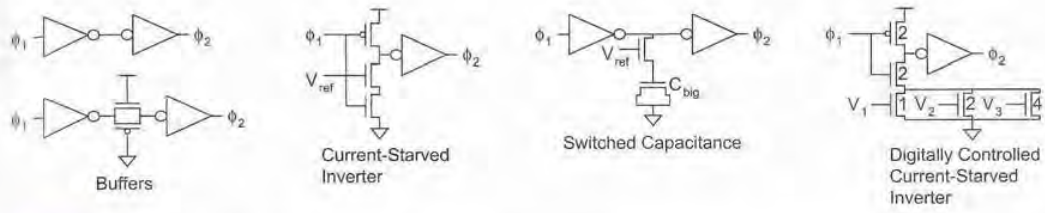


FIG 7.62 Delay elements

Section 4.7.6 showed that the delay of gates can vary by as much as 30% relative to an FO4 inverter across process, voltage, and temperature variations. Therefore, the delay line should provide some margin to guarantee it always is slower than the gate it must match. For example, [Yee00] uses a 20% margin. Many industrial designs use even more margin to ensure the circuit will have good yield in high-volume production. (Who wants to explain to the big boss why he or she wasted millions of dollars for the sake of saving a few picoseconds?) You should always make sure that the circuit works correctly in all process and environmental corners because it is not obvious which corner will cause the worst-case mismatches. Moreover, random device variations and inaccuracies in the parasitic extraction and device models cause further mismatch that cannot be captured through the design corner files. Yet another problem is that matching differs from one process to another, potentially requiring expensive redesign of circuits with matched delays when they are ported to the next process generation. Adjustable delay lines are attractive because the margin can be set more aggressively and increased after fabrication (as was done in [Vangal02]); however, generating and distributing a low-noise reference voltage can be challenging.

The key to good matching is to make the delay circuit behave like the gate it should match as much as possible. A good technique is to use a *dummy gate* in the delay line, as shown in Figure 7.63 for a 2:1 dynamic multiplexer. The dummy gate replicates the gate being matched so that to first order, process and environmental variations will affect both identically. The input pattern is selected for worst-case delay.

You might be tempted to use longer-than-minimum length transistors to create long delays, but this is not good because transistor length variations will affect the delay circuit much differently than the gate it matches.

Despite all of these difficulties, delay matching has been used for decades in specialized circumstances that require wide NOR operation such as CAMs and PLAs (see Sections 11.6 and 11.7). [Yee00] proposes wider use of delay matching in datapath applications and names the practice *Clock-Delayed* (CD) Domino.

7.5.4.2 Clock-delayed Domino In the simplest CD Domino scheme, logic is *levelized* as shown in Figure 7.64(a). The boxes represent domino gates annotated with their worst-case delay. Delay elements produce clocks tuned to the slowest gate in each level. The overall path delay is the sum of the delays of each element, which may be longer than the

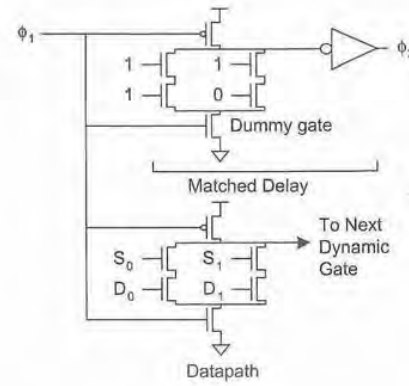


FIG 7.63 Delay matching with dummy gates

actual critical path through logic. An alternative scheme is to clock each gate at a time matched to its latest input, as shown in Figure 7.64(b). This better matches the critical path at the expense of more delay elements and design effort. CD Domino is most effective for functions where high fan-in gates can be converted to wide dynamic NORs.

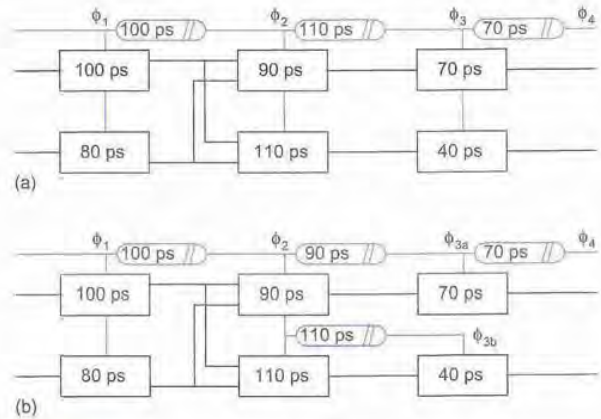


FIG 7.64 CD Domino timing

7.5.4.3 Race-based Nonmonotonic Logic The Itanium II processor uses a specialized nonmonotonic structure called an *annihilation* gate for high fan-in AND functions such as a 6-input decoder [Naffziger02]. An ordinary high fan-in AND gate requires many series transistors. Using DeMorgan's Law, it can be converted to a wide NOR with complementary inputs. The annihilation gate in Figure 7.65 performs this NOR function very rapidly

while generating a monotonically rising output suitable as an input to subsequent domino gates. It can be viewed as a dynamic NOR followed by a domino buffer with no clock delay. This introduces a race condition, but the two stages are carefully sized so the NOR will always win the race.

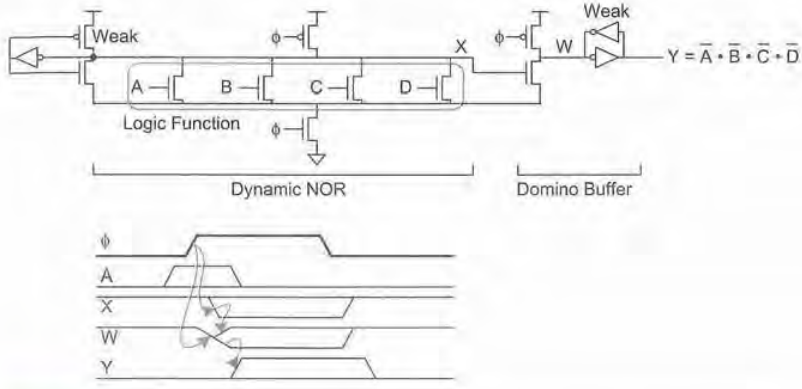


FIG 7.65 Annihilation gate

Initially, both X and W are precharged. The inputs must set up and hold around the rising edge of ϕ. When ϕ rises and the gate evaluates, W begins pulling down. If one or more of the inputs are asserted, X will also pull down, cutting off the transistor that was discharging W. The keeper will restore W back to a high level and the output Y will remain low. If all of the inputs are low, X will remain high, W will discharge, and Y will monotonically rise. The full keepers hold both X and W after evaluation. The gate has a built-in race: X must fall quickly so that W does not droop too much and cause a glitch on Y. The annihilation gate requires very careful design and attention to noise sources, but is fast and compact.

The annihilation gate is a new incarnation of a long-lost circuit called *Latched Domino* [Pretorius86] shown in Figure 7.66. The Latched Domino gate adds a cross-coupled nMOS transistor to help pull down node X. It also replaces the full keepers with ordinary keepers. As long as the glitches on X and W are small enough, Y_h and Y_l are good monotonic dual-rail outputs.

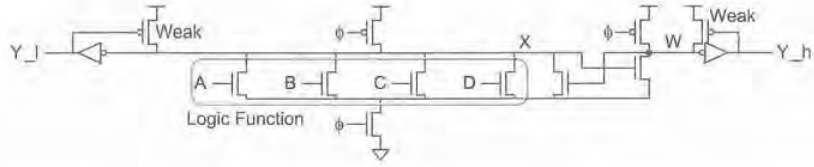


FIG 7.66 Latched domino gate

Intel uses a similar gate called a *Complementary Signal Generator* (CSG) shown in Figure 7.67 to produce dual-rail outputs from single rail inputs in a 5 GHz ALU [Vangal02]. Again, nodes X and W precharge and the inputs must set up before the rising edge of ϕ. When ϕ rises, W begins to discharge. If any of the inputs are true, X also begins to discharge. The pulldown and keeper strengths must be chosen so that X falls much faster than W. Once one of these nodes falls, it turns on the cross-coupled pMOS pull-ups to restore the other node to full levels. These strong pull-ups also help fight leakage, permitting wide fan-in logic functions. The CSG was designed so the glitch on W would not exceed 10% of V_{DD}. In a dual-V_t process, low V_t transistors were used on all but the noise-sensitive input transistors.

The CSG is very effective in circuits that can use single-rail signals through most of the path but that require dual-rail monotonic inputs to the last stage for functions such as XOR. They can be much faster and more compact than dual-rail domino but suffer from the very delicate race. The clock does impose a hard edge before which the inputs must set up so that skew and delay mismatches on this clock appear as sequencing overhead.

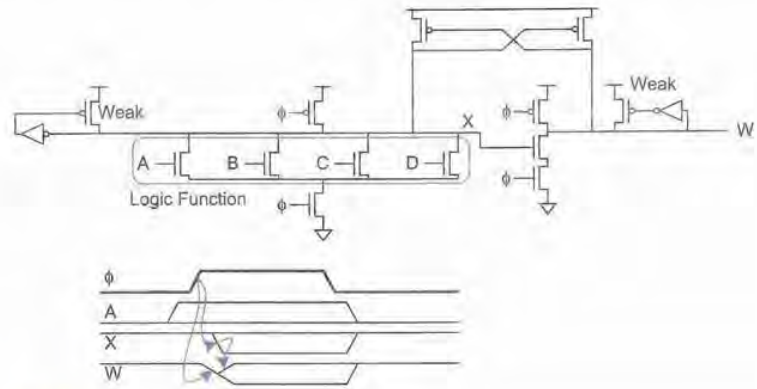


FIG 7.67 Complementary signal generator

7.5.4.4 Output Prediction Logic Clock-delayed and race-based dynamic logic represent two extremes in nonmonotonic logic. Both consist of two cascaded dynamic gates. CD Domino delays the clock to the second gate until the first has had time to fully discharge so the second gate will not glitch. Race-based logic such as annihilation gates and CSGs do not delay the clock but use transistor and keeper sizing to ensure the glitch on the second gate remains acceptably small. *Output Prediction Logic* (OPL) fits between these two extremes, delaying the clock by a moderate amount and accepting modest glitches [McMurchie00]. The delay is chosen as a compromise between performance and glitch size.

Figure 7.68 shows a basic OPL gate consisting of a Noise-Tolerant Precharge dynamic stage that was discussed in Section 6.2.4.3. You can view it either as a complementary CMOS structure with clocked evaluation and precharge transistors or as a dynamic gate plus a complementary pMOS pullup network. Like an ordinary dynamic gate, the output precharges high while the clock is low, then evaluates low when the clock rises and the appropriate inputs are asserted. However, like a static CMOS gate, the output can pull back high through the pMOS network to recover from output glitches.

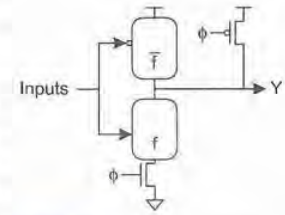


FIG 7.68 OPL gate

Figure 7.69 shows a chain of OPL 2-input NAND gates. Each receives a clock delayed from the previous stage. As the stages are inverting, it resembles a chain of CD Domino gates. The amount of delay is critical to the circuit operation. Suppose A is '1' and all the unnamed outer inputs are also '1' so B, D, and F should pull low and C and E stay high. OPL precharges all the outputs to predict each output will remain high. The gates can be very fast because only half of the outputs have to transition. Figure 7.70 shows three cases of short (a), long (b), and medium (c) clock delays between a pair of OPL inverters. Simulating OPL is tricky because if all the gates are identical, the outputs will tend to settle at a metastable point momentarily, then diverge as the previous gate transitions. To break this misleading symmetry, a small parasitic capacitance C_p was added to node B.

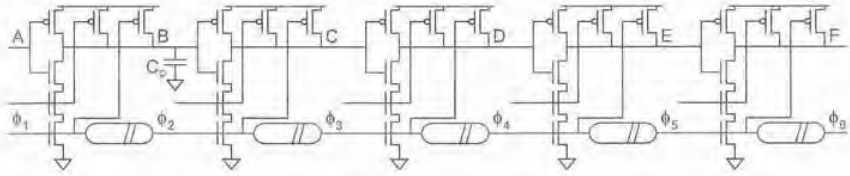


FIG 7.69 Chain of OPL gates

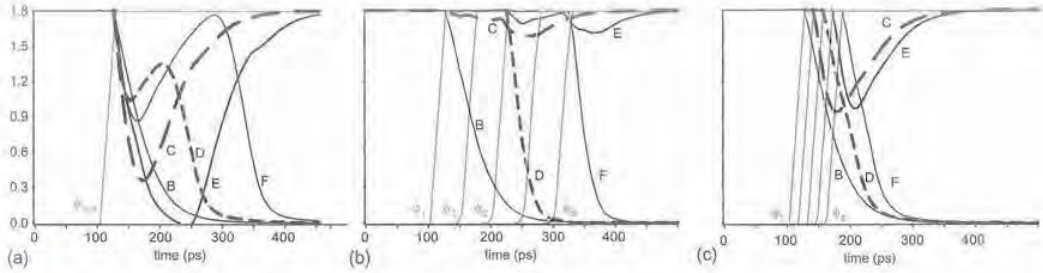


FIG 7.70 OPL waveforms for various clock delays

In Figure 7.70(a), all the clocks rise simultaneously. ϕ_2 arrives at the second stage while the input B is still high so C pulls most of the way low. When B falls, C rises back up. This causes D to fall, E to rise, and F to fall. In this mode of operation, the data ripples through the gates much as in static CMOS and the path delay is rather slow.

In Figure 7.70(b), the clock spacing is 50 ps. ϕ_2 arrives at the second stage after the input B has pulled most of the way low so C remains high. After another delay, ϕ_3 rises, D falls, and so forth. In this mode of operation, the OPL chain behaves in clock-blocked mode just like clock-delayed domino. The path delay is the sum of the clock delays plus the propagation delay of the final stage, which again is rather slow because the clock delay is lengthy.

In Figure 7.70(c), the clock spacing is 15 ps. ϕ_2 arrives at the second stage as the input B is falling so C glitches slightly, then returns to a good high value. After another delay, D falls. Again, the path delay is essentially the sum of the clock delays and final stage delay, but it is now faster because the clock delay is shorter than required for CD domino. The extra speed comes at the expense of some glitching.

A challenge in designing OPL gates is to choose just the right clock spacing. It should be as short as possible but not too short. Figure 7.71 plots the delay from A to F against the spacing between clocks. The nMOS transistors are 2 units wide and the figure compares the performance for pMOS of 1, 3, or 5 units. Wider pMOS transistors have slower evaluation delays but recover better from glitches. The lowest path delay occurs with a clock spacing of 10–15 ps. The path slows significantly if the clock spacing is too short, so the designer should nominally provide some margin in clock delay to ensure the worst case is still long enough. In comparison, a chain of complementary CMOS NAND gates has a delay of 213 ps.

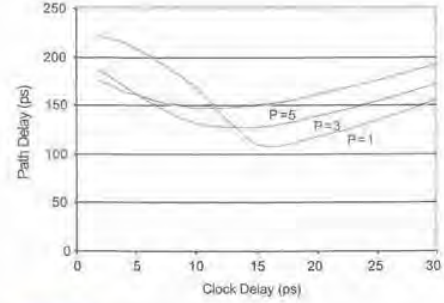


FIG 7.71 Path delay vs. clock delay

The basic OPL technique was illustrated for modified complementary CMOS gates that are relatively slow but recover quickly from large glitches. It also applies to other circuit families that have faster evaluation delays for high fan-in NOR structures such as

pseudo-nMOS or dynamic gates, as illustrated in Figure 7.72(a and b). Pseudo-nMOS OPL is faster at evaluating because of the lower logical effort, but slower at recovery if the glitch is large. Dynamic OPL gates evaluate even faster but cannot recover at all if the glitch is large enough to flip the keeper. Using a low-skew feedback inverter improves the glitch tolerance for the keeper. As the best delay between clocks is a function of both evaluation delay and glitch tolerance, pseudo-nMOS and dynamic OPL are comparable in performance. Dynamic gates dissipate less power than pseudo-nMOS but may fail entirely if the clock delay is too short. Figure 7.72(c) shows a differential OPL gate using cross-coupled pMOS keepers that do not fight the initial transition and that can recover from arbitrarily large glitches [Kio01]. The inventors found that this was the fastest family of all, nearly five times faster than static CMOS.

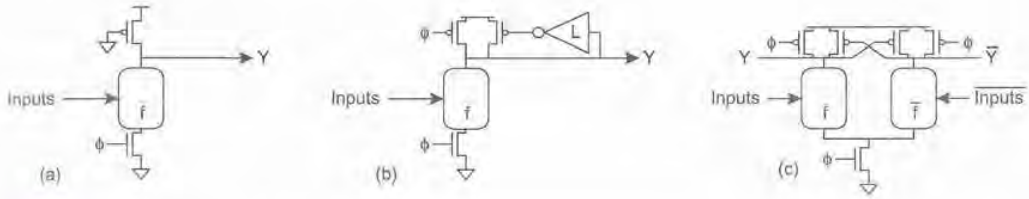


FIG 7.72 Alternative OPL circuit families

7.5.5 Static-to-domino Interface

Static CMOS gates require inputs that are levels and may produce nonmonotonic glitches on the outputs. Domino gates require inputs that are monotonic during evaluation and produce pulses on the outputs. Therefore, interface circuitry is necessary at the static-to-domino interface to avoid glitches and circuitry at the domino-to-static interface to convert the pulses into levels.

7.5.5.1 Static-to-domino Interface Falling static inputs to domino gates must set up by the time the gate begins evaluation and should not change until evaluation is complete. This imposes a hard edge and the associated clock skew penalties, so the static-to-domino interface is relatively expensive. High-performance skew-tolerant domino pipelines build entire loops out of domino to avoid paying the skew at the static-to-domino interface.

A simple solution to avoiding glitches at the interface is to latch the static signals as shown in Figure 7.73(a). The latch is opaque while the domino gates evaluate. Figure 7.73(b) shows that the latch does not need to be placed at the end of the previous half-cycle. The static logic must be designed to set up before domino gates enter evaluation. The latch prevents the next token from arriving too early and upsetting the domino input.

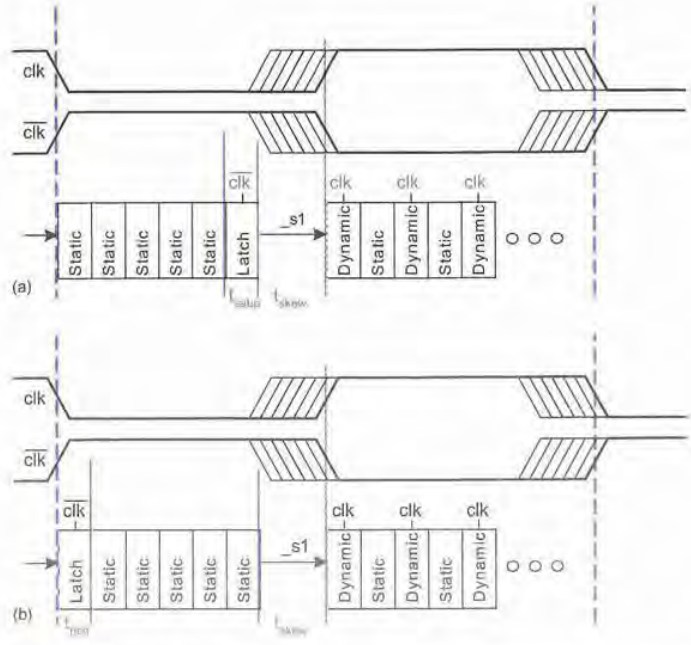


FIG 7.73 Latch at static to domino interface

In systems using flip-flops or pulsed latches, another approach is to capture the input on the clock edge with a flop or latch that produces monotonically rising outputs, as shown in Figure 7.74. The SA/F-F produces dual-rail monotonic outputs if the SR latch is replaced by HI-skew inverters. The K6 differential flip-flop also produces dual-rail monotonic pulsed outputs suitable for self-resetting logic that requires pulsed inputs. In any of these cases, you can build logic into the latch or flip-flop. For example, Figure 7.75 shows a single-rail *pulsed domino flip-flop* or *entry latch* (ELAT) with integrated logic used on UltraSparc and Itanium 2 [Klass99, Naffziger02]. It can be viewed as a fully dynamic version of the Klass SDFF. Falling inputs must set up before the clock edge, but rising inputs can borrow a small amount of time after the edge. The output is a monotonically rising signal suitable as an input to subsequent domino gates. The pulsed domino flip-flop can also use a single pulsed nMOS transistor in place of the two clocked devices [Mehta99].

7.5.5.2 Domino-to-static Interface Domino outputs are pulses that terminate when the gates precharge. Static logic requires levels that remain stable until they are sampled, independent of the clock period. At the domino-to-static interface, another latch is required as a pulse-to-level converter. The output of this latch can borrow time into subsequent static logic, so the latch does not impose a hard edge.

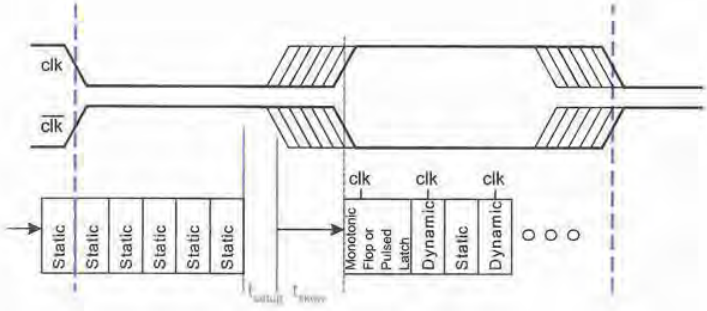


FIG 7.74 Monotonic flip-flop or pulsed latch at static to domino interface

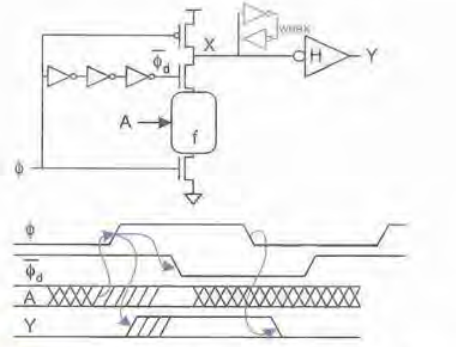


FIG 7.75 Pulsed domino flip-flop with integrated logic

Figure 7.76 shows a domino gate with a simple built-in output latch. The HI-skew inverter is replaced with a clocked inverter. The critical path still passes through only the pMOS transistor, so the latch is nearly as fast as a simple inverter. On the falling edge of the clock, the latch locks out the precharge, holding the result of the domino gate until the next rising edge of the clock. A weak inverter staticizes the *Y* output. *Y* should typically be buffered before driving long wires to prevent noise from backdriving the latch. Note that *Y* does glitch low shortly after the rising edge of the clock. The glitch can cause excess power dissipation in the static logic. Dual-rail domino outputs can avoid the glitch at the cost of greater delay by using the SR latch shown in Figure 7.29(a or b).

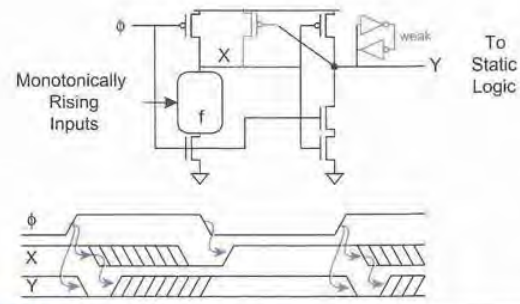


FIG 7.76 Domino gate with pulse-to-level output conversion

The Itanium 2 uses a *dynamic latch converter* (DLC) on the last domino gate in each half-cycle to hold the output by delaying the precharge until the next rising clock edge. This provides greater skew tolerance in domino paths and allows the output to drive static logic. An ordinary dynamic gate receives the same clock for the precharge (*RCLK*) and evaluation (*ECLK*) transistors and has a weak pMOS keeper. Figure 7.77 shows a DLC that is a "bolt-on" block consisting of a delayed clock generator and an extra nMOS keeper to make a full keeper The *RCLK* generator produces a brief low-going precharge pulse on the rising edge of the clock. Although the precharge and evaluate transistors may be on momentarily, this is not a large concern because the DLC operates the last gate of the half-cycle so the inputs do not arrive until several gate delays after the clock edge. The DLC also may include scan circuitry illustrated in Section 9.6.5.

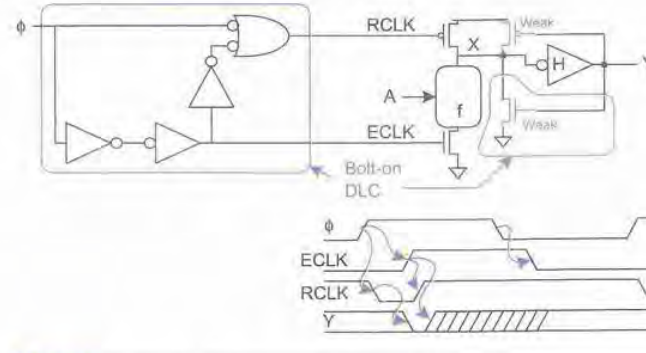


FIG 7.77 Dynamic latch converter

In self-resetting domino, the reset pulse for the last gate can also be delayed so that the domino output is compatible with static logic. For example, Figure 7.57 showed such a pulse generator for Global STP domino.

7.5.6 Delayed Keepers

Dynamic gates with high leakage current will eventually discharge to an invalid logic level unless they have strong keepers. The problem is especially severe when the inputs use many parallel low-V_t transistors. Unfortunately, the strong keeper slows the dynamic gate, reducing the performance advantage it was supposed to provide. As discussed in Section 6.2.4.3 for the burn-in keeper, this problem can be addressed by breaking the keeper into two parts. One part operates in the typical fashion. The second part turns on after some delay when the gate has had adequate time to evaluate. This combines the advantage of fast initial evaluation from the smaller keeper with better long-term leakage immunity from the two keepers in parallel.

Figure 7.78(a) shows such a *conditional keeper* [Alvandpour02]. *P2* is the conventional feedback keeper. *P1* turns on three gate delays after ϕ rises to help fight leakage. Figure 7.78(b) shows *High-Speed Domino* that leaves *X* floating momentarily until *P1* turns ON [Allam00]. *Skew-Tolerant High-Speed Domino* uses two transistors in series as the second keeper [Jung01], as shown in Figure 7.78(c). The inverting delay logic (IDL) can be an inverter, three inverters in series, or some other inverting structure with greater delay.

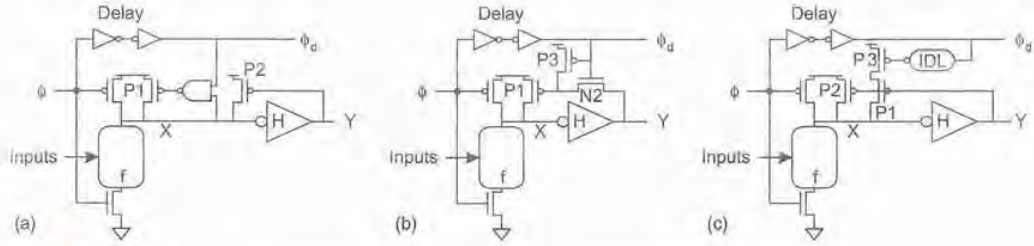


FIG 7.78 Delayed keepers

A challenge with any of these delayed keeper techniques is to ensure that the second part of the keeper turns on at a suitable time after the input arrives, but before too much leakage occurs. They work best for the first gate after a phase boundary, where the inputs are known to set up by the time the clock rises [Alvandpour02].

7.6 Synchronizers



Sequencing elements are characterized by a setup and hold time. If the data input changes before the setup time, the output reflects the new value after a bounded propagation delay. If the data changes after the hold time, the output reflects the old value after a bounded propagation delay. If the data changes during the *aperture* between the setup and hold

times, the output may be unpredictable and the time for the output to settle to a good logic level may be unbounded. Properly designed synchronous circuits guarantee the data is stable during the aperture. However, many interesting systems must interface with data coming from sources that are not synchronized to the same clock. For example, the user can press a key at any time and data coming over a network can be aligned with a clock of differing phase or frequency.

A *synchronizer* is a circuit that accepts an input that can change at arbitrary times and produces an output aligned to the synchronizer's clock. Because the input can change during the synchronizer's aperture, the synchronizer has a nonzero probability of producing a *metastable* output [Chaney73]. This section first examines the response of a latch to an analog voltage that can change near the sampling clock edge. The latch can enter a metastable state for some amount of time that is unbounded, although the probability of remaining metastable drops off exponentially with time. Therefore, you can build a simple synchronizer by sampling a signal, waiting until the probability of metastability is acceptably low, then sampling again. In certain circumstances, the relationship of the data and clock timing is more predictable, permitting more reliable synchronizers.

7.6.1 Metastability

A latch is a bistable device; i.e., it has two stable states (0 and 1). Under the right conditions, that latch can enter a metastable state in which the output is at an indeterminate level between 0 and 1. For example, Figure 7.79 shows a simple model for a static latch consisting of two switches (probably transmission gates in practice) and two inverters. While the latch is transparent, the sample switch is closed and the hold switch open (Figure 7.79(a)). When the latch goes opaque, the sample switch opens and the hold switch closes (Figure 7.79(b)). Figure 7.79(c) shows the DC transfer characteristics of the two inverters. Because $A = B$ when the latch is opaque, the stable states are $A = B = 0$ and $A = B = V_{DD}$. The metastable state is $A = B = V_m$, where V_m is not a legal logic level. This point is called *metastable* because the voltages are self-consistent and can remain there indefinitely. However, any noise or other disturbance will cause A and B to switch to one of the two stable states. Figure 7.79(d) shows an analogy of a ball on a hill. The top of the hill is a metastable state. Any disturbance will cause the ball to roll down to one of the two stable states on the left or right side of the hill.

Figure 7.80(a) plots the output of the latch from Figure 7.17(g) as the data transitions near the falling clock edge. If the data changes at just the wrong time t_m within the aperture, the output can remain at the metastable point for some time before settling to a valid logic level. Figure 7.80(b) plots t_{DQ} vs. $t_{DC} - t_m$ on a semilogarithmic scale for a rising input and output. The delay is less than or equal to t_{pdq} for inputs that meet the setup time and increases for inputs that arrive too close to t_m. The points marked on the graph will be used in the example at the end of this section.

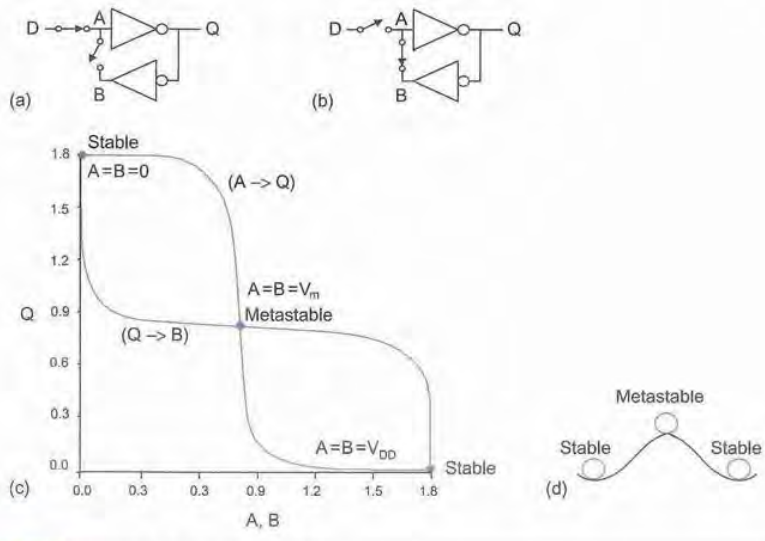


FIG 7.79 Metastable state in static latch

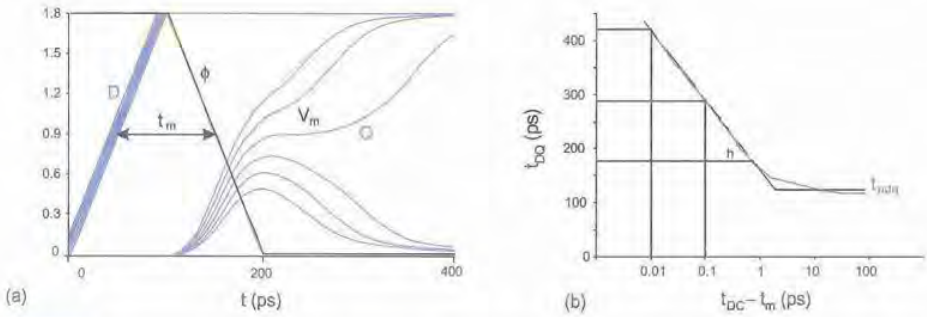


FIG 7.80 Metastable transients and propagation delay

The cross-coupled inverters behave like a linear amplifier with gain G when A is near the metastable voltage V_m. The inverter delay can be modeled with an output resistance R and load capacitance C. We can predict the behavior in metastability by assuming that the initial voltage on node A when the latch becomes opaque at time $t = 0$ is

$$A(0) = V_m + a(0) \tag{7.25}$$

where $a(0)$ is a small signal offset from the metastable point. Figure 7.81 shows a small-signal model for $a(t)$. The behavior after time 0 is given by the first-order differential equation

$$\frac{Ga(t)-a(t)}{R}=C\frac{da(t)}{dt} \tag{7.26}$$

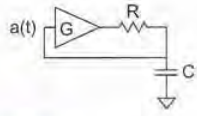


FIG 7.81 Small signal model of bistable element in metastability

Solving this equation shows that the positive feedback drives $a(t)$ exponentially away from the metastable point with a time constant determined by the gain and RC delay of the cross-coupled inverter loop.

$$a(t)=a(0)e^{\frac{t}{\tau_s}};\tau_s=\frac{RC}{G-1} \tag{7.27}$$

Suppose the node is defined to reach a legal logic level when $|a(t)|$ exceeds some deviation ΔV. The time to reach this level is

$$t_{DQ}=\tau_s\left[\ln\Delta V-\ln a(0)\right] \tag{7.28}$$

This shows that the latch propagation delay increases as $A(0)$ approaches the metastable point and $a(0)$ approaches 0. The delay approaches infinity if $a(0)$ is precisely 0, but this can never physically happen because of noise. However, there is no upper bound on the possible waiting time t required for the signal to become valid. If the input $A(t)$ is a ramp that passes through V_m at time t_m, $a(0)$ is proportional to $t_{DC}-t_m$. Observe that EQ (7.28) is a good fit to the log-linear portion of Figure 7.80(b). The time constant τ_s is essentially the reciprocal of the gain-bandwidth product [Flannagan85]. Therefore, the feedback loop in a latch should have a high gain-bandwidth product to resolve from metastability quickly.

Designers need to know the probability that latch propagation delay exceeds some time t'. Longer propagation delays are less likely because they require $a(0)$ to be closer to 0. This probability should decrease with the clock period T_c because a uniformly distributed input change is less likely to occur near the critical time. Projecting through EQ (7.28) shows that it should also decrease exponentially with waiting time t'. Theoretical and experimental studies [Chaney83, Veendrick80, Horstmann89] find that the probability can be expressed as

$$P\left(t_{DQ}>t'\right)=\frac{T_0}{T_c}e^{\frac{t'}{\tau_s}}\text{ for }t'>h \tag{7.29}$$

where T_0 and τ_s can be extracted through simulation [Baghini02] or measurement. Intuitively, T_0/T_c describes the probability that the input would change during the aperture,

Example

Find τ_s, T_0, and h for the latch using the data in Figure 7.80.

Solution: h is the propagation delay above which the data fits a good straight line on a log-linear scale. In Figure 7.80, this appears to be approximately 175 ps. The probability that the delay exceeds some t' is the chance that the input changing at a random time falls within the small aperture that leads to the high delay. We can choose two points on the linear portion of the plot and solve for the two unknowns. For example, choosing (0.1 ps, 290 ps) and (0.01 ps, 415 ps), we solve

$$P(t_{DQ} > 290 \text{ ps}) = \frac{0.1 ps}{T_c} = \frac{T_0}{T_c} e^{-\frac{290 \text{ ps}}{\tau_s}}$$
$$P(t_{DQ} > 415 \text{ ps}) = \frac{0.01 ps}{T_c} = \frac{T_0}{T_c} e^{-\frac{415 \text{ ps}}{\tau_s}} \quad (7.30)$$

T_c drops out of the equations and we find τ_s = 54 ps and T_0 = 21 ps. Recall that this data was taken for a rising input. A conservative design should also consider the falling input and take data in the slow rather than typical environment.

causing metastability, and the exponential term describes the probability that the output has not resolved after t' if it did enter metastability. The model is only valid for sufficiently long propagation delays (h significantly greater than t_{pdq}).

We have seen that a good synchronizer latch should have a feedback loop with a high-gain-bandwidth product. Conventional latches have data and clock transistors in series, increasing the delay (i.e., reducing the bandwidth). Figure 7.82 shows a synchronizer flip-flop in which the feedback loops simplify to cross-coupled inverter pairs [Dike99]. Furthermore, the flip-flop is reset to 0, and then is only set to 1 if D = 1 to minimize loading on the feedback loop.

The flip-flop consists of master and slave jamb latches. Each latch is reset to 0 while D = 0. When D rises before ϕ, the master output X is driven high. This in turn drives the slave output Q high when ϕ rises. The pulldown transistors are just large enough to overpower the cross-coupled inverters, but should add as little stray capacitance to the feedback loops as possible. X and Q are buffered with small inverters so they do not load the feedback loops.

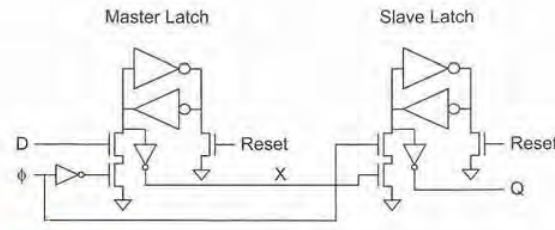


FIG 7.82 Fast synchronizer flip-flop

7.6.2 A Simple Synchronizer

A synchronizer accepts an input D and a clock ϕ. It produces an output Q that ought to be valid some bounded delay after the clock. The synchronizer has an aperture defined by a setup and hold time around the rising edge of the clock. If the data is stable during the aperture, Q should equal D. If the data changes during the aperture, Q can be chosen arbitrarily. Unfortunately, it is impossible to build a perfect synchronizer because the duration of metastability can be unbounded. We define synchronizer failure as occurring if the output has not settled to a valid logic level after some time t'.

Figure 7.83 shows a simple synchronizer built from a pair of flip-flops. *F1* samples the asynchronous input D. The output X may be metastable for some time, but will settle to a good level with high probability if we wait long enough. *F2* samples X and produces an output Q that should be a valid logic level and be aligned with the clock. The synchronizer has a latency of one clock cycle, T_c. It can fail if X has not settled to a valid level by a setup time before the second clock edge.

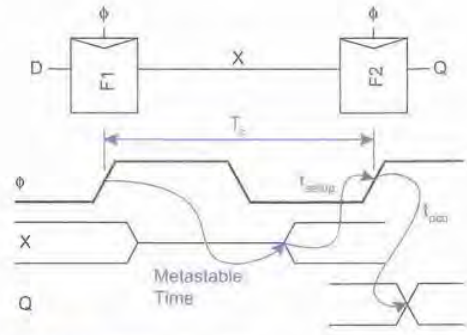


FIG 7.83 Simple synchronizer

Each flip-flop samples on the rising clock edge when the master latch becomes opaque. The slave latch merely passes along the contents of the master and does not sig-

Example

A particular synchronizer flip-flop in a 0.25 μm process has τ_s = 20 ps and T_0 = 15 ps [Dike99]. Assuming the input toggles at N = 50 MHz and the setup time is negligible, what is the minimum clock period T_c for which the MTBF exceeds one year?

Solution: 1 year ≈ π • 10^7 seconds. Thus, we must solve

$$\pi \cdot 10^7 = \frac{T_c e^{\frac{T_c}{20 \cdot 10^{-12}}}}{\left(5 \cdot 10^7\right)\left(15 \cdot 10^{-12}\right)} \tag{7.31}$$

numerically for a minimum clock period of 625 ps (1.6 GHz).

Example

How much longer must we wait for a 1000-year MTBF?

Solution: Solving a similar equation gives 760 ps. Increasing the waiting time by 135 ps improved MTBF by a factor of 1000.

nificantly affect the probability of metastability. If the synchronizer receives an average of N asynchronous input changes at D each second, the probability of synchronizer failure in any given second is

$$P(\text{failure}) = N \frac{T_0}{T_c} e^{\frac{-\left(T_c - t_{\text{setup}}\right)}{\tau_s}} \tag{7.32}$$

and the mean time between failures increases exponentially with cycle time

$$MTBF = \frac{1}{P(\text{failure})} = \frac{T_c e^{\frac{T_c - t_{\text{setup}}}{\tau_s}}}{NT_0} \tag{7.33}$$

The acceptable MTBF depends on the application. For medical equipment where synchronizer reliability is crucial and latency is relatively unimportant, the MTBF can be chosen to be longer than the life of the universe (~10^{19} seconds) by waiting more than one clock cycle before using the data. For noncritical applications, the MTBF can be chosen to be merely longer than the designer's expected duration of employment at the company!

7.6.3 Communicating Between Asynchronous Clock Domains

A common application of synchronizers is in communication between asynchronous clock domains, i.e., blocks of circuits that do not share a common clock. Suppose System A is controlled by *clkA* that needs to transmit *N*-bit data words to System B, which is controlled by *clkB*, as shown in Figure 7.84. The systems can represent separate chips or separate units within a chip using unrelated clocks. Each word should be received by system B exactly once. System A must guarantee that the data is stable while the flip-flops in System B sample the word. It indicates when new data is valid by using a request signal (*Req*), so System B receives the word exactly once rather than zero or multiple times. System B replies with an acknowledge signal (*Ack*) when it has sampled the data so System A knows when the data can safely be changed. If the relationship between *clkA* and *clkB* is completely unknown, a synchronizer is required at the interface.

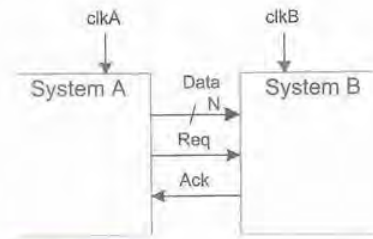


FIG 7.84 Communication between asynchronous systems

The request and acknowledge signals are called *handshaking* lines. Figure 7.85 illustrates two-phase and four-phase handshaking protocols. The four-phase handshake is level-sensitive while the two-phase handshake is edge-triggered. In the four-phase handshake, system A places data on the bus. It then raises *Req* to indicate that the data is valid. System B samples the data when it sees a high value on *Req* and raises *Ack* to indicate that the data has been captured. System A lowers *Req*, then system B lowers *Ack*. This protocol requires four transitions of the handshake lines. In the two-phase handshake, system A places data on the bus. Then it changes *Req* (low to high or high to low) to indicate that the data is valid. System B samples the data when it detects a change in the level of *Req* and toggles *Ack* to indicate that the data has been captured. This protocol uses fewer transitions (and thus possibly less time and energy), but requires circuitry that responds to edges rather than levels.

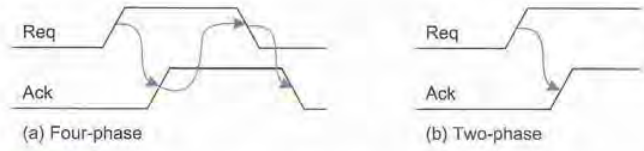


FIG 7.85 Four-phase and two-phase handshake protocols

Req is not synchronized to *clkB*. If it changes at the same time *clkB* rises, System *B* may receive a metastable value. Thus, System *B* needs a synchronizer on the *Req* input. If the synchronizer waits long enough, the request will resolve to a valid logic level with very high probability. The synchronizer may resolve high or low. If it resolves high, the rising request was detected and System *B* can sample the data. If it resolves low, the rising request was just missed. However, it will be detected on the next cycle of *clkB*, just as it would have been if the rising request occurred just slightly later. *Ack* is not synchronized to *clkA*, so it also requires a synchronizer.

Figure 7.86 shows a typical two-phase handshaking system [Crews03]. *clkA* and *clkB* operate at unrelated frequencies and each system may not know the frequency of its counterpart. Each system contains a synchronizer, a level-to-pulse converter, and a pulse-to-level converter. System A asserts *ReqA* for one cycle when *DataA* is ready. We will refer to this as a *pulse*. The XOR and flip-flop form a pulse-to-level converter that toggles the level of *Req*. This level is synchronized to *clkB*. When an edge is detected, the level-to-pulse converter produces a pulse on *ReqB*. This pulse in turn toggles *Ack*. The acknowledge level is synchronized to *clkA* and converted back to a pulse on *AckA*. The synchronizers add significant latency so the throughput of asynchronous communication can be much lower than that of synchronous communication.

7.6.4 Common Synchronizer Mistakes

Although a synchronizer is a simple circuit, it is notoriously easy to misuse. For example, the AMD 9513 system timing controller, AMD 9519 interrupt controller, Zilog Z-80 Serial I/O interface, Intel 8048 microprocessor, and AMD 29000 microprocessor are all said to have suffered from metastability problems [Wakerly00].

One way to build a bad synchronizer is to use a bad latch or flip-flop. The synchronizer depends on positive feedback to drive the output to a good logic level. Therefore, dynamic latches without feedback such as Figure 7.17(a–d) do not work. The probability of failure grows exponentially with the time constant of the feedback loop. Therefore, the loop should be lightly loaded. The latch from Figure 7.17(f) is a poor choice because a large capacitive load on the output will increase the time constant; Figure 7.17(g) is a much better choice.

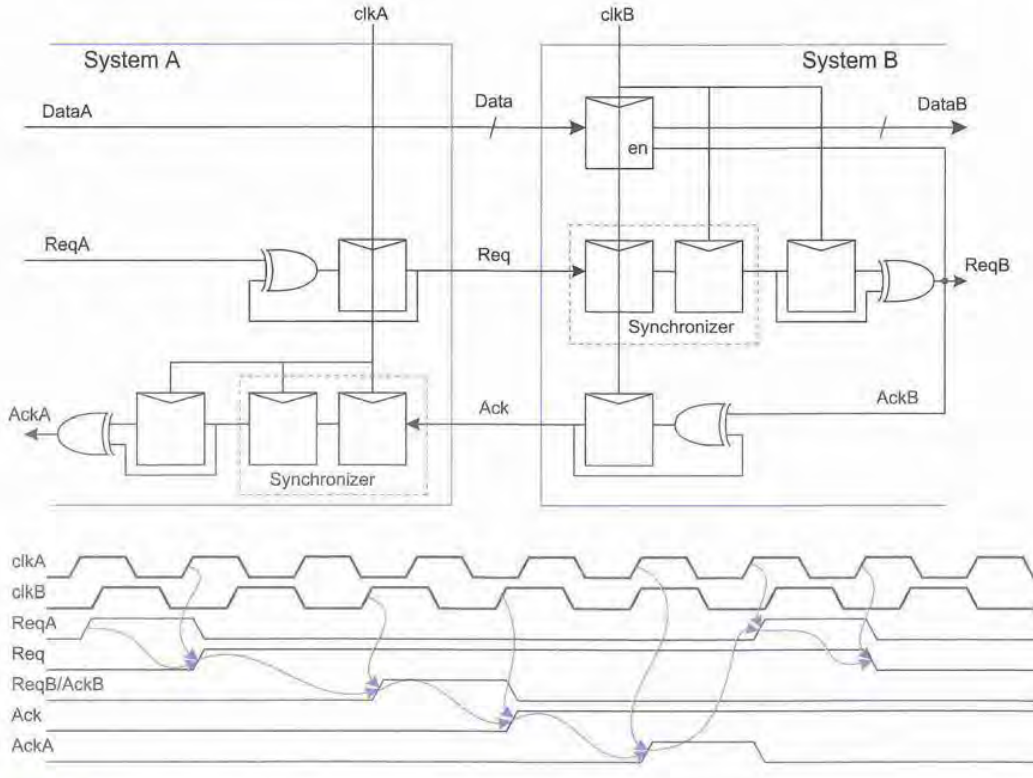


FIG 7.86 Two-phase handshake circuitry with synchronizers

Another error is to capture inconsistent data. For example, Figure 7.87(a) shows a single signal driving two synchronizers (each consisting of a pair of back-to-back flip-flops). If the signal is stable through the aperture, *Q1* and *Q2* will be the same. However, if the signal changes during the aperture, *Q1* and *Q2* might resolve to different values. If the system requires that *Q1* and *Q2* be identical representations of the data input, they must come from a single synchronizer.

Another example is to synchronize a multi-bit word where more than one bit might be changing at a time. For example, if the word in Figure 7.87(b) is transitioning from 0000 to 1111, the synchronizer might produce a value such as 0101 that is neither the old nor the new data word. For this reason, the system in Figure 7.86 synchronized only the *Req*/*Ack* signals and used them to indicate that data was stable to sample or finished being sampled. *Gray codes* (see Section 10.7.3) are also useful for counters whose outputs must be synchronized because exactly one bit changes on each count so the synchronizer is guaranteed to find either the old or the new data value.

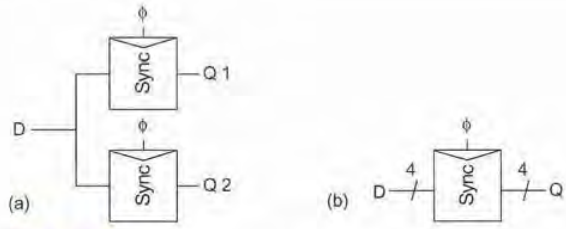


FIG 7.87 Bad synchronizer designs

In general, synchronizer bugs are intermittent and very difficult to locate and diagnose. For this reason, the number of synchronizers in a system should be strictly limited.

7.6.5 Arbiters

The *arbiter* of Figure 7.88(a) is closely related to the synchronizer. It determines which of two inputs arrived first. If the spacing between the inputs exceeds some aperture time, the first input should be acknowledged. If the spacing is smaller, exactly one of the two inputs should be acknowledged, but the choice is arbitrary. For example, in a television game show, two contestants may pound buttons to answer a question. If one presses the button first, she should be acknowledged. If both press the button at times too close to distinguish, the host may choose one of the two contestants arbitrarily.

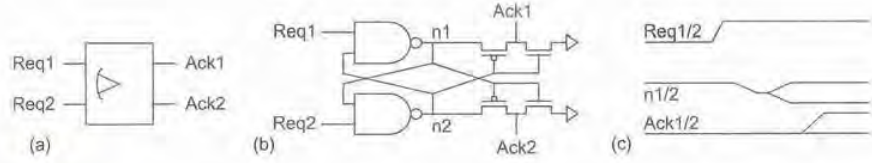


FIG 7.88 Arbiter

Figure 7.88(b) shows an arbiter built from an SR latch and a four-transistor metastability filter. If one of the request inputs arrives well before the other, the latch will respond appropriately. However, if they arrive at nearly the same time, the latch may be driven into metastability, as shown in Figure 7.88(c). The filter keeps both acknowledge signals low until the voltage difference between the internal nodes $n1$ and $n2$ exceeds V_t, indicating that a decision has been made. Such an asynchronous arbiter will never produce metastable outputs. However, the time required to make the decision can be unbounded, so the acknowledge signals must be synchronized before they are used in a clocked system.

Arbiters can be generalized to select 1-of-N or M-of-N inputs. However, such arbiters have multiple metastable states and require careful design [van Berkel99].



7.6.6 Degrees of Synchrony

The simple synchronizer from Section 7.6.2 accepts inputs that can change at any time, but has a nonzero probability of failure. In practice, many inputs may not be aligned to a single system clock, but they may still be predictable. Table 7.3 provides a classification of degrees of synchrony between input signals and the receiver system clock [Messerschmitt90] based on the difference in phase $\Delta\phi$ and frequency Δf.

[Dally98] describes a number of synchronizers that have zero failure probability and possibly lower latency when the input is predictable. They are based on the observation that either the signal or a copy of the signal delayed by t_a will be stable throughout the aperture. Hence, a synchronizer that can predict the input arrival time can choose the signal or its delayed counterpart to safely sample. Mesochronous signals are synchronized by measuring the phase difference and delaying the input enough to ensure it falls outside the aperture. Plesiochronous signals can be synchronized in a similar fashion, but the phase difference slowly varies so the delay must be occasionally adjusted. Because the frequencies differ, the synchronizer requires some control flow to handle the missing or extra data items. Periodic signals also require control flow and use a clock predictor to calculate where the next clock edge will occur and whether the signal must be delayed to avoid falling in the aperture.



7.7 Wave Pipelining

Recall that sequencing elements are used in pipelined systems to prevent the current token from overtaking the next token or from being overtaken by the previous token in the pipeline. If the elements propagate through the pipeline at a fairly constant rate, explicit sequencing elements may not be necessary to maintain sequence. As an analogy, fiber optic cables carry data as a series of light pulses. Many pulses enter the cable before the first one reaches the end, yet the cable does not need internal latches to keep the pulses separated because they propagate along the cable at a well-controlled velocity. The maximum data rate is limited by the dispersion along the line that causes pulses to smear over time and blur into one another if they become too short.

Figure 7.89 compares traditional pipelining with wave pipelining. In both cases, the pipeline contains combinational logic separated by registers (Figure 7.89(a)). The registers *F1* and *F2* receive clocks *clk1* and *clk2* that are nominally identical, but might experience skew. Figure 7.89(b) shows traditional pipelining. The data is launched on the rising edge of *clk1*. Its propagation is indicated by the hashed cone. *D2* becomes stable somewhere between the contamination and propagation delays after the clock edge (neglecting the flip-flop *clk*-to-*Q* delay). *D2* must not change during the setup and hold aperture around *clk2*, marked with the gray box. The figure shows two successive cycles in which tokens *i*

Table 7.3 Degrees of synchrony

Classification	Periodic	$\Delta\phi$	Δf	Description
Synchronous	Yes	0	0	Signal has same frequency and phase as clock. Safe to sample signal directly with the clock. **Example:** Flip-flop to flip-flop on chip.
Mesochronous	Yes	constant	0	Signal has same frequency, but is out of phase with the clock. Safe to sample signal if it is delayed by a constant amount to fall outside aperture. **Example:** Chip-to-chip where chips use same clock signal, but might have arbitrarily large skews.
Plesiochronous	Yes	varies slowly	small	Signal has nearly the same frequency. Phase drifts slowly over time. Safe to sample signal if it is delayed by a variable but predictable amount. Difference in frequency can lead to dropped or duplicated data. **Example:** Board-to-board where boards use clock crystals with small mismatches in nominally identical rates.
Periodic	Yes	varies rapidly	large	Signal is periodic at an arbitrary frequency. Periodic nature can be exploited to predict and delay accordingly when data will change during aperture. **Example:** Board-to-board where boards use different frequency clocks.
Asynchronous	No	unknown	unknown	Signal may change at arbitrary times. Full synchronizer is required. **Example:** Input from pushbutton switch.

and $i + 1$ move through the pipeline. Each token passes through the combinational logic in a single cycle. Figure 7.89(c) shows wave pipelining with a clock of twice the frequency. Token i enters the combinational logic, but takes two cycles to reach *F2*. Meanwhile, token $i + 1$ enters the logic a cycle later. As long as each token is stable to sample at *F2* and the cones do not overlap, the pipeline will operate correctly with the same latency but twice the throughput.

[Burleson98] gives a tutorial on wave pipelining and derives the timing constraints. In general, a wave pipeline can contain N tokens between each pair of registers. The maximum value of N is limited by the ratio of propagation delay to dispersion of the logic cones:

$$N < \frac{t_{pd}}{t_{pd} - t_{cd}} \tag{7.34}$$

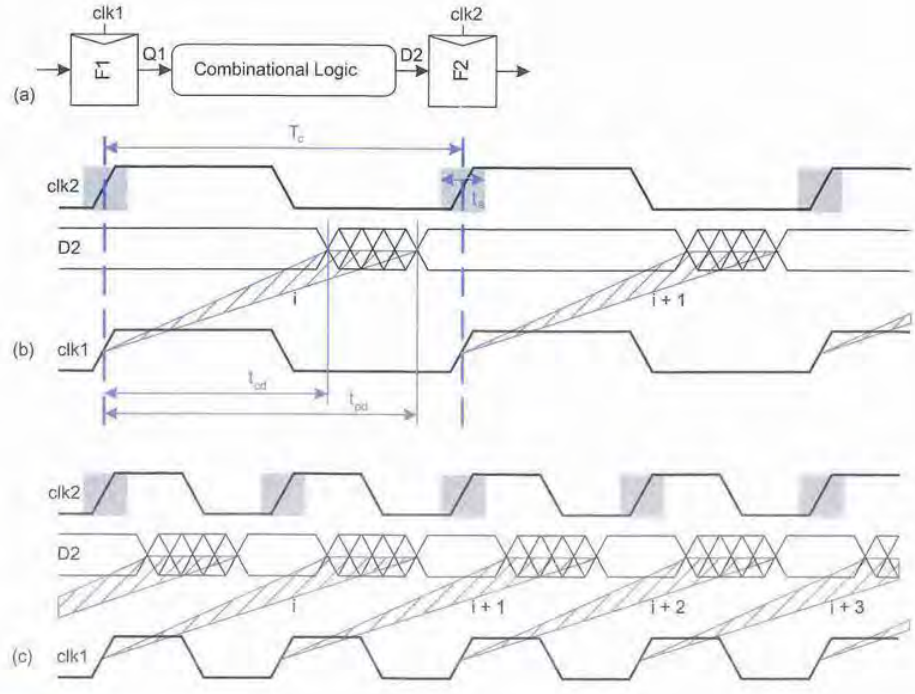


FIG 7.89 Wave pipelining

If the contamination and propagation delays are nearly equal, the combinational logic can contain many tokens simultaneously. In practice, the delays tend to be widely variable because of voltage, temperature, and processing as well as differences in path lengths through the logic. Clock skew and sequencing overhead also eat into the timing budgets. In practice, even achieving $N = 2$ simultaneous tokens can be difficult and wave pipelining has not achieved widespread popularity for general-purpose logic.

7.8 Pitfalls and Fallacies

Incompletely reporting flip-flop delay

The effective delay of a flip-flop is its minimum D-to-Q time. This is the sum of the setup time t_{setup} and the *clk*-to-Q delay t_{pdq} if these delays are defined to minimize the sum. Some engineers focus on only the *clk*-to-Q delay or define setup and *clk*-to-Q delays in a way that does not minimize the sum.

Failing to check hold times

One of the leading reasons that chips fail to operate even though they appear to simulate correctly is hold time violations, especially violations caused by unexpected clock skew. Unless a design uses two-phase nonoverlapping clocks, the clock skew should be carefully modeled and the hold times should be checked with a static timing analyzer. These checks should happen as soon as a block is designed so that errors can be corrected immediately. For example, a large microprocessor used a wide assortment of delayed clocks to solve setup time problems on long paths. Hold times were not checked until shortly before tapeout, leading to a significant schedule slip when many violations were found.

Choosing a sequencing methodology too late in the design cycle

Designers may choose from many sequencing methodologies, each of which has tradeoffs. The best methodology for a particular application is very debatable, and engineers love a good debate. If the sequencing methodology is not settled at the beginning of the project, experience shows that engineers will waste tremendous amounts of time redoing work as the method changes, or supporting and verifying multiple methodologies. Projects need a strong technical manager to demand that a team choose one method at the beginning and stick with it.

Failing to synchronize asynchronous inputs

Unsynchronized inputs can cause strange and wonderful sporadic system failures that are very difficult to locate. For example, a finite state machine running off one clock received a *READY* input from a *UART* running on another clock when the *UART* had data available, as shown in Figure 7.90. The designer reasoned that synchronizing the *READY* signal was unimportant because if it changed near the clock edge of the FSM, she did not care whether it was detected in one cycle or the next. Moreover, the clock was so slow that metastability would have time to resolve. However, the FSM occasionally failed by jumping to seemingly random states that could never legally occur. After two months of debugging, she realized that the problem was triggered if the asynchronous *READY* signal was asserted a few gate delays before the FSM clock edge. The propagation delay through the combinational logic was different for various bits of the next state logic. Some bits had changed to their new values while others were still at their old values, so the FSM could jump to an undefined state. Registering the *READY* signal with the FSM clock before it drove the combinational logic solved the problem.

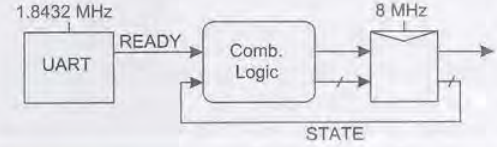


FIG 7.90 Unsynchronized input

Building faulty synchronizers

Designers have found many ways to build faulty synchronizers. For example, if an asynchronous input drives more than one synchronizer, the two synchroniz-

continued

ers can resolve to different values. If they must produce consistent outputs, only one synchronizer should be used. In another example, synchronizers must not accept multi-bit inputs where more than one of the bits can change simultaneously. This would pose the risk that some of the bits resolve as changed while others resolve in their old state, resulting in an invalid pattern that is neither the old nor the new input word. In yet another example, synchronizers with poorly designed feedback loops can be much slower than expected and can have exponentially worse mean time between failures.

7.9 Case Study: Pentium 4 and Itanium 2 Sequencing Methodologies

The Pentium 4 and Itanium 2 represent two philosophies of high-performance microprocessor design sometimes called *Speed Demon* and *Braniac*, respectively. The Pentium 4 was designed by Intel for server and desktop applications and has migrated into laptop computers as well. The Itanium 2 was jointly designed by Hewlett-Packard and Intel for high-end server applications. Figure 7.91 shows the date of introduction and the performance of several generations of these processors.

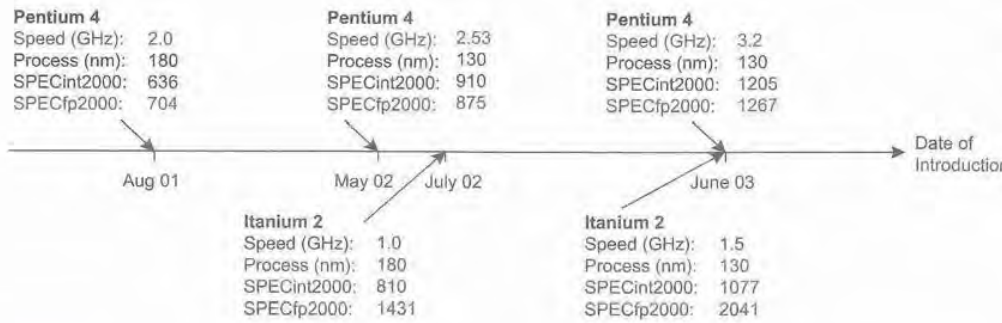


FIG 7.91 Microprocessor timeline

The Pentium 4 uses a very long (20+ stage) pipeline with few stages of logic per cycle to achieve extremely high frequencies. It issues up to three instructions per cycle, but the long pipeline causes severe penalties for branch mispredictions and cache misses, so the overall average number of instructions executed per cycle is relatively low. Figure 4.76 showed a die photo of the 42-million transistor Pentium 4. The chip consumes around 55 watts. A top-of-the-line Pentium 4 sold in 1000-unit quantities for around $400–$600 (depending on price pressure from competitor AMD). The chip has aggressively migrated into Intel's most advanced processes both to achieve high performance and to reduce the

die size and manufacturing cost. The Speed Demon approach also gives Intel bragging rights to the highest clock frequency microprocessors, which is important because many consumers compare processors on clock frequency rather than benchmark performance. [Hrishikesh02] argues that the best logic depth is only 6 to 8 FO4 inverter delays per cycle.

In contrast, the Itanium 2 focuses on executing many instructions per cycle at a lower clock rate. It uses an 8-stage integer pipeline clocked at about half the rate of the Pentium 4 in the same process, so each cycle accommodates about twice as many gate delays (roughly 20–24 FO4 inverter delays, compared to roughly 10–12 for the Pentium 4). However, it issues up to six instructions per cycle and has a very high-bandwidth memory and I/O system to deliver these instructions and their data. As a result, it achieves nearly the same integer performance and much better floating-point benchmark results than the Pentium 4. Moreover, it also performs well on multiprocessor and transaction processing tasks typical of high-end servers. Figure 7.92 shows a die photo of the Itanium 2 with a 3MB level 3 (L3) cache; notice that the three levels of cache occupy most of the die area and most of the 221 million transistors. The 1.5 GHz model with 6MB cache bumps the transistor count to 410 million and further dwarfs the processor core. The chip consumes about 130 watts, limited by the cost of cooling multiprocessor server boxes. A high-end Itanium 2 sold for more than $4000 because the server market is much less price-sensitive. The chip has lagged a year behind the Pentium 4 in process technology.

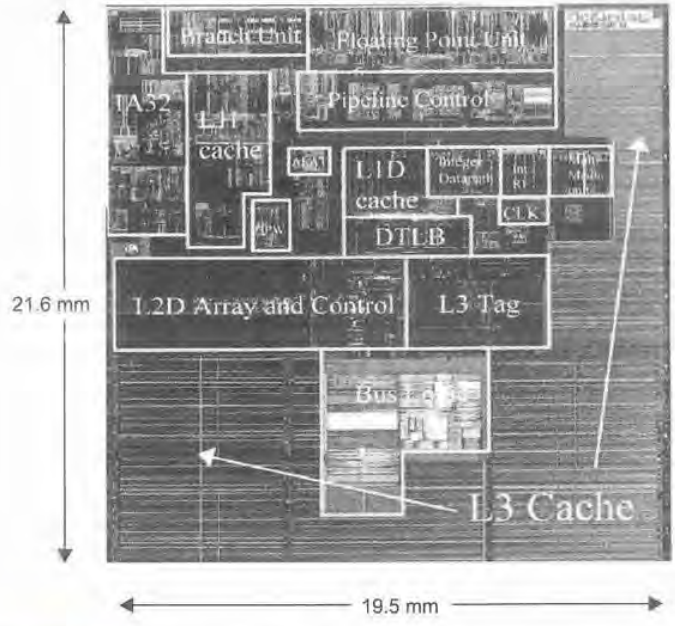


FIG 7.92 Itanium II die photo. © IEEE 2002.

7.9.1 Pentium 4 Sequencing

The Pentium 4 actually operates at three different internal clock rates [Hinton01, Kurd01]. In addition to the *core clock* that drives most of the logic, it has a double-speed *fast clock* for the ALU core and a half-speed *slow clock* for noncritical portions of the chip. The core clock is distributed across the chip using a triple spine, as will be shown in Section 12.5.4.3. These clocks drive pulsed latches, flip-flops, and self-resetting domino gates.

The ALU runs at a remarkable rate of twice the core clock frequency (about 6 FO4 inverter delays). To achieve this speed, it is stripped down to just the essential functions of the bypass multiplexer and the 16-bit add/subtract unit. Other less commonly used blocks such as the shifter and multiplier operate at core frequency. The ALU uses unfooted domino gates. The gates produce pulsed outputs and precharge in a self-timed fashion using the Global STP approach described in Section 7.5.2.4. These circuits demanded extensive verification by expert circuit designers to ensure the domino gates function reliably.

The Pentium 4 uses pulsed latches operating at all three clock speeds. Figure 7.93 shows pulse generators that receive the core clock and produce the appropriate output pulses. The medium-speed pulse generator produces a pulse on the rising edge of the core clock. The pulse width can be shaped by the adjustable delay buffer to provide both long pulses (offering more time borrowing) and short pulses (to prevent hold-time problems). The buffer is built from a digitally controlled current-starved inverter with four discrete settings. The pulse generator also accepts enable signals to gate the clock or save power on unused blocks. The slow pulse generator produces a pulse on every other rising edge of the core clock. To do this, it receives a sync signal that is asserted every other cycle. While the sync signal must be distributed globally, it is more convenient than distributing a half-speed clock because it can accept substantial skew while still being stable around the clock edge. The fast pulse generator produces pulses on both the rising and falling edges of the core clock. Therefore, the core clock should have nearly equal high and low times, i.e., 50% duty cycle, so the pulses are equally spaced.

7.9.2 Itanium 2 Sequencing

The Itanium 2 operates at a single primary clock speed, but also makes use of extensive domino logic and pulsed latches [Naffziger02, Fetzer02, Rusu03]. The clock is distributed across the chip using an H-tree, as will be shown in Section 12.5.4.2. The H-tree drives 33 second-level clock buffers distributed across the chip. These buffer outputs, called SLCBOs, in turn drive local clock gaters that serve banks of sequencing elements within functional blocks. There are 24 different types of clock gaters producing inverted, stretched, delayed, and pulsed clocks. Figure 7.94 shows some of these clocks. Each gater comes in many sizes and is tuned to drive different clock loads with low skew over regions of up to about 1000 μm. Section 12.5.6.3 analyzes the clock skew from this distribution network.

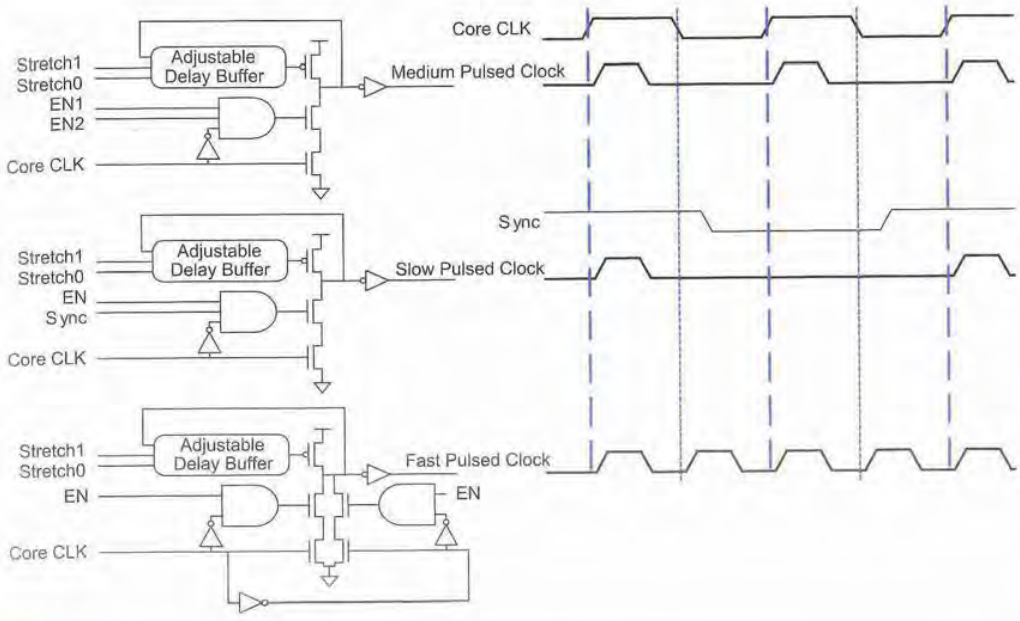


FIG 7.93 Pulse generators

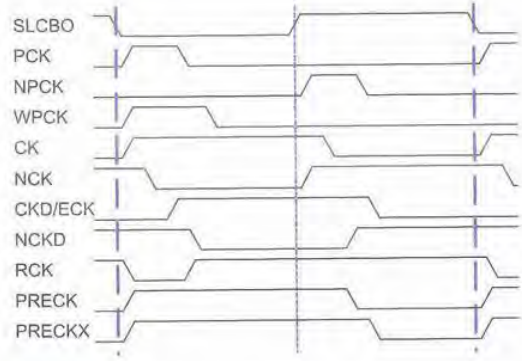


FIG 7.94 Clock gater waveforms

In the Itanium 2, 95% of the static logic blocks use Naffziger-pulsed latches with 125-ps wide pulses called *PCK*, as were described in Section 7.3.3. The pulsed latches are fast, permit a small amount of time borrowing, and present a small load to the clock. In situations where more time borrowing is needed, the gater may produce a wider pulsed clock *WPCK*. As discussed in Section 7.4.1.2, *clocked deracers* using *NPCK* can be inserted between back-to-back pulsed latches to prevent hold time violations.

The Itanium 2 uses extensive amounts of domino logic to achieve high performance at the expense of power consumption and careful design. Figure 7.95 shows a typical four-phase skew-tolerant domino pipeline from the Itanium 2. *CK* and *NCK* are clocks with a duty cycle slightly higher than 50% that are playing the roles of ϕ_1 and ϕ_3. They are delayed with buffers to produce *CKD* and *NCKD* (ϕ_2 and ϕ_4).

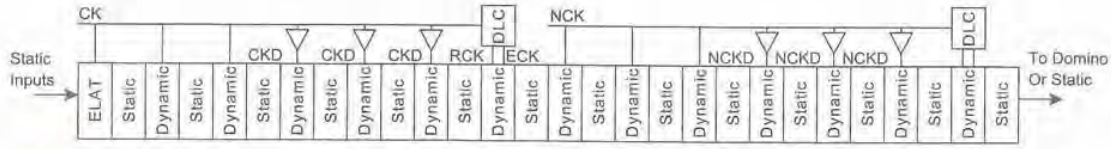


FIG 7.95 Four-phase skew-tolerant domino pipeline

The last gate in each phase uses a dynamic latch converter (DLC) to hold the output so that it can drive static logic and retain its state when the clock stops, as was discussed in Section 7.5.5.2. The DLC also provides scan capability at each half-cycle boundary to help with test.

At static-to-dynamic interfaces, inputs pass through pulsed entry latches (ELATs) that capture the static signal and convert it into a single-rail monotonic dynamic signal. These ELATs were shown in Figure 7.75 and can perform logic as well as latching. Some ELATs use *PCK*, while others derive the pulse internally from *CK*.

In some especially critical paths, alternating stages use unfooted domino gates. The falling edge of the clocks for these stages is delayed further to avoid contention during precharge. Figure 7.96 shows an extreme example in which a footed gate is followed by three stages of unfooted domino with successively delayed precharge edges, as was done in the 64-bit Naffziger adder used in the integer execution units.



FIG 7.96 Clocking for footless domino

Summary

This chapter has examined the tradeoffs of sequencing with flip-flops, two-phase transparent latches, and pulsed latches. The ITRS forecasts cycle times dropping well below 10 FO4 delays (see Table 4.17). Minimizing sequencing overhead will be very important in these high-performance systems. Flip-flops are the simplest, but have the greatest sequencing overhead. Transparent latches are most tolerant of skew and allow the most time borrowing, but require greater design effort to partition logic into half-cycles instead of cycles. Pulsed latches have the lowest sequencing overhead, but are most susceptible to min-delay problems. Table 7.4 compares the sequencing overhead, minimum delay constraint, and time borrowing capability of each technique. All of the techniques are used in commercial products, and the designer's choice depends on the design constraints and CAD tools.

Table 7.4 Comparison of sequencing elements

	Sequencing overhead ($T_c - t_{pd}$)	Minimum logic delay t_{cd}	Time borrowing t_{borrow}
Flip-Flops	$t_{pcq} + t_{setup} + t_{skew}$	$t_{hold} - t_{ccq} + t_{skew}$	0
Two-Phase Transparent Latches	$2t_{pdq}$	$t_{hold} - t_{ccq} - t_{nonoverlap} + t_{skew}$ in each half-cycle	$\frac{T_c}{2} - \left(t_{setup} + t_{nonoverlap} + t_{skew}\right)$
Pulsed Latches	$\max\left(t_{pdq}, t_{pcq} + t_{setup} - t_{pw} + t_{skew}\right)$	$t_{hold} - t_{ccq} + t_{pw} + t_{skew}$	

In class projects for introductory VLSI classes, timing analysis is often rudimentary or nonexistent. Using two-phase nonoverlapping clocks generated off chip is attractive because you can guarantee the chip will have no max-delay or min-delay failures if the clock period and nonoverlap are sufficiently large. However, it is not practical to generate and distribute two nonoverlapping phases on a large, high-performance commercial chip.

The great majority of low- and mid-performance designs and some high-speed designs use flip-flops. Flip-flops are very easy to use and are well understood by most designers. Even more importantly, they are handled well by synthesis tools and timing analyzers. Unfortunately, in systems with few gate delays per cycle, the sequencing overhead can consume a large fraction of the cycle. Moreover, many standard cell flip-flops are intentionally rather slow to prevent hold time violations at the expense of greater sequencing overhead.

Most two-phase latch systems distribute a single clock and locally invert it to drive the second latch. These systems tolerate significant amounts of clock skew without loss of performance and can borrow time to balance delay intentionally or opportunistically. However, the systems require more effort to understand because time borrowing distrib-

utes the timing constraints across many stages of a pipeline rather than isolating them at each stage. Not all timing analyzers handle latches gracefully, especially when there are different amounts of clock skew between different clocks [Harris99]. Two-phase latches have been used in the Alpha 21064 and 21164 [Gronowski98], PowerPC 603 [Gerosa94], and many other IBM designs.

Pulsed latches have low sequencing overhead. They present a tradeoff when choosing pulse width: A wide pulse permits more time borrowing and skew tolerance, but makes min-delay constraints harder to meet. Pulsed latches are also popular because they can be modeled as fast flip-flops with a lousy hold time from the point of view of a timing analyzer (or novice designer) if intentional time borrowing is not permitted. The min-delay problems can be largely overcome by mixing pulsed latches for long paths and flip-flops for short paths. Unfortunately, many real designs have paths in which the propagation delay is very long but the contamination delay is very short, making robust design more challenging. Pulsed latches have been used on Itanium 2 [Naffziger02], Pentium 4 [Kord01], Athlon [Draper97], and CRAY 1 [Unger86]. However, they can wreak havoc with conventional commercially available design flows and are best avoided unless the performance requirements are extreme.

Domino circuits are widely used in high-performance systems because they are 1.5-2x faster than static CMOS. Traditional domino circuits with latches have high sequencing overhead that wastes much of the potential speedup, so most designers have moved to skew-tolerant techniques. Static-to-domino interfaces impose hard edges and the associated sequencing overhead, motivating the use of domino throughout critical loops. Single-rail domino only computes noninverting functions, so most loops require dual-rail domino that consumes more area, wiring, and power and is ill-suited to wide NORs. An alternative is to push the inverting functions to the end of the pipeline, using single-rail domino through most of the pipeline and nonmonotonic static logic at the end. The area savings comes at the cost of one hard edge in the cycle.

Four-phase or delayed reset skew-tolerant domino circuits work well in datapaths because the clock generation is relatively simple. Self-resetting domino is ideally suited to memories where the decoder power consumption is greatly reduced by only precharging the output that switched and where the number of unique circuits to design is relatively small. It was also used on the Pentium 4, but was costly in terms of designer effort because so many pulse constraints must be satisfied.

Clock-delayed domino is used in wide dynamic NOR functions where the power consumption of pseudo-nMOS is unacceptable. For example, it is an important technique for CAMs and PLAs. The delay matching raises an unpleasant tradeoff between speed and correct operation, requiring significant margin for safe operation. The risk of race conditions deters many designers from using it more widely. Annihilation gates and complementary signal generators are interesting special cases in which no clock gate delay at all is required. Output prediction logic is also interesting, but has yet to be proven in a large application.

When inputs to a system arrive asynchronously, they cannot be guaranteed to meet setup or hold times at clocked elements. Even if we do not care whether an input arrived in one cycle or the next, we must ensure that the clocked element produces a valid logic

level. Unfortunately, if the element samples a changing input at just the wrong time, it may produce a metastable output that remains invalid for an unbounded amount of time. The probability of metastability drops off exponentially with time. Systems use synchronizers to sample the asynchronous input and hold it long enough to resolve to a valid logic level with very high probability before passing it onward.

Most synchronous VLSI systems use opaque sequencing elements to separate one token from the next. In contrast, many optical systems transmit data as pulses separated in time. As long as the propagation medium does not disperse the pulses too badly, they can be recovered at a receiver. Similarly, if a VLSI system has low dispersion, i.e., nearly equal contamination and propagation delays, it can send more than one wave of data without explicit latching. Such wave pipelining offers the potential of high throughput and low sequencing overhead. However, it is difficult to perform in practice because of the variability of data delay.

Exercises

Use the following timing parameters for the questions in this section.

Table 7.5 Sequencing element parameters

	Setup Time	*clk*-to-*Q* Delay	*D*-to-*Q* Delay	Contamination Delay	Hold Time
Flip-flops	65 ps	50 ps	n/a	35 ps	30 ps
Latches	25 ps	50 ps	40 ps	35 ps	30 ps

7.1 For each of the following sequencing styles, determine the maximum logic propagation delay available within a 500 ps clock cycle. Assume there is zero clock skew and no time borrowing takes place.

a) Flip-flops

b) Two-phase transparent latches

c) Pulsed latches with 80 ps pulse width

7.2 Repeat Exercise 7.1 if the clock skew between any two elements can be up to 50 ps.

7.3 For each of the following sequencing styles, determine the minimum logic contamination delay in each clock cycle (or half-cycle, for two-phase latches). Assume there is zero clock skew.

a) Flip-flops

b) Two-phase transparent latches with 50% duty cycle clocks

c) Two-phase transparent latches with 60 ps of nonoverlap between phases

d) Pulsed latches with 80 ps pulse width

7.4 Repeat Exercise 7.3 if the clock skew between any two elements can be up to 50 ps.

7.5 Suppose one cycle of logic is particularly critical and the next cycle is nearly empty. Determine the maximum amount of time the first cycle can borrow into the second for each of the following sequencing styles. Assume there is zero clock skew.

a) Flip-flops

b) Two-phase transparent latches with 50% duty cycle clocks

c) Two-phase transparent latches with 60 ps of nonoverlap between phases

d) Pulsed latches with 80 ps pulse width

7.6 Repeat Exercise 7.5 if the clock skew between any two elements can be up to 50 ps.

7.7 Prove EQ (7.17).

7.8 Consider a flip-flop built from a pair of transparent latches using nonoverlapping clocks. Express the setup time, hold time, and clock-to-Q delay of the flip-flop in terms of the latch timing parameters and $t_{nonoverlap}$.

7.9 For the path in Figure 7.97, determine which latches borrow time and if any setup time violations occur. Repeat for cycle times of 1200, 1000, and 800 ps. Assume there is zero clock skew and that the latch delays are accounted for in the propagation delay Δ's.

a) $\Delta 1 = 550$ ps; $\Delta 2 = 580$ ps; $\Delta 3 = 450$ ps; $\Delta 4 = 200$ ps

b) $\Delta 1 = 300$ ps; $\Delta 2 = 600$ ps; $\Delta 3 = 400$ ps; $\Delta 4 = 550$ ps

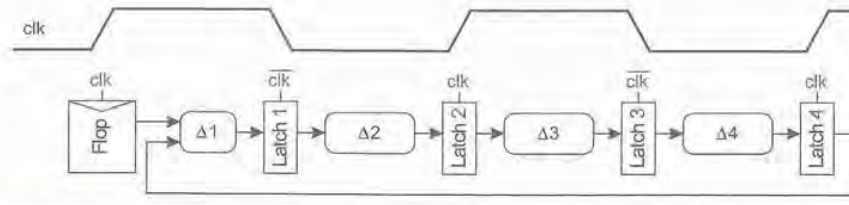


FIG 7.97 Example path

7.10 Determine the minimum clock period at which the circuit in Figure 7.98 will operate correctly for each of the following logic delays. Assume there is zero clock skew and that the latch delays are accounted for in the propagation delay Δ's.

a) $\Delta 1 = 300$ ps; $\Delta 2 = 400$ ps; $\Delta 3 = 200$ ps; $\Delta 4 = 350$ ps

b) Δ1 = 300 ps; Δ2 = 400 ps; Δ3 = 400 ps; Δ4 = 550 ps

c) Δ1 = 300 ps; Δ2 = 900 ps; Δ3 = 200 ps; Δ4 = 350 ps

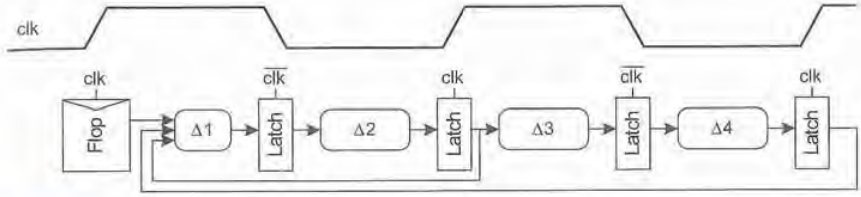


FIG 7.98 Another example path

7.11 Repeat Exercise 7.10 if the clock skew is 100 ps.

7.12 Label the timing types of each signal in the circuit from Figure 7.97. The flip-flop is constructed with back-to-back transparent latches—the first controlled by *clk_b* and the second by *clk*.

7.13 Using a simulator, compare the *D*-to-*Q* propagation delays of a conventional dynamic latch from Figure 7.17(d) and a TSPC latch from Figure 7.30(a). Assume each latch is loaded with a fanout of 4. Use 4 λ-wide clocked transistors and tune the other transistor sizes for least propagation delay.

7.14 Using a simulator, find the setup and hold times of a TSPC latch under the assumptions of Exercise 7.13.

7.15 Determine the maximum logic propagation delay available in a cycle for a traditional domino pipeline using a 500 ps clock cycle. Assume there is zero clock skew.

7.16 Repeat Exercise 7.15 if the clock skew between any two elements can be up to 50 ps.

7.17 Determine the maximum logic propagation delay available in a cycle for a four-phase skew-tolerant domino pipeline using a 500 ps clock cycle. Assume there is zero clock skew.

7.18 Repeat Exercise 7.17 if the clock skew between any two elements can be up to 50 ps.

7.19 How much time can one phase borrow into the next in Exercise 7.18 if the clocks each have a 50% duty cycle?

7.20 Repeat Exercise 7.18 if the clocks have a 65% duty cycle.

7.21 Design a fast-pulsed latch. Make the gate capacitance on the clock and data inputs equal. Let the latch drive an output load of four identical latches. Simulate your latch and find the setup and hold times and clock-to-*Q* propagation and contamination delays. Express your results in FO4 inverter delays.

7.22 Simulate the worst-case propagation delay of an 8-input dynamic NOR gate driving a fanout of 4. Report the delay in all 16 design corners (voltage, temperature, nMOS, pMOS). Also determine the delay of a fanout-of-4 inverter in each of these corners. By what percentage does the absolute propagation delay of the NOR gate vary across corners? By what percentage does its normalized delay vary (in terms of FO4 inverters)? Comment on the implications for circuits using matched delays.

7.23 A synchronizer uses a flip-flop with τ_c = 54 ps and T_0 = 21 ps. Assuming the input toggles at 10 MHz and the setup time is negligible, what is the minimum clock period for which the mean time between failures exceeds 100 years?

7.24 Simulate the synchronizer flip-flop of Figure 7.82 and make a plot analogous to Figure 7.80. From your plot, find Δ_{DQ}, b, τ, and T_0.

7.25 InferiorCircuits, Inc., wants to sell you a perfect synchronizer that they claim never produces a metastable output. The synchronizer consists of a regular flip-flop followed by a high-gain comparator that produces a high output for inputs above $0.25 \cdot V_{DD}$ and a low output for inputs below that point. The VP of marketing argues that even if the flip-flop enters metastability, its output will hover near $V_{DD}/2$ so the synchronizer will produce a good high output after the comparator. Why wouldn't you buy this synchronizer?



Design Methodology and Tools

8

8.1 Introduction

The manner in which you go about designing a particular system, chip, or circuit can have a profound impact on both the effort expended and the outcome of the design. IC designers have developed and adapted strategies from allied disciplines such as software engineering to form a cohesive set of principles to increase the likelihood of timely, successful designs. We will explore these principles in this chapter. While the broad principles of design have not changed in decades, the details of design styles and tools have evolved along with advances in technology and increasing levels of productivity. This chapter represents current CMOS design methods and provides an overview of a complex subject that could fill many books on its own. We encourage you to actively monitor the companies discussed and literature cited in the chapter to track the latest developments in this rapidly changing field.

As introduced in Section 1.6, an integrated circuit can be described in terms of three *domains*: (1) the *behavioral* domain, (2) the *structural* domain, and (3) the *physical* domain. The behavioral domain specifies what we wish to accomplish with a system. For instance, at the highest level, we might want to build an ultra-low-power radio for a distributed sensor network. The structural domain specifies the interconnection of components required to achieve the behavior we desire. Again, by way of example, our sensor radio might require a sensor, a radio transceiver, a processor and memory (with software), and a power source connected in a particular manner. Finally, the physical domain specifies how to arrange the components in order to connect them, which in turn allows the required behavior. Our example might start with the specification for an enclosure to hold the device, followed by a succession of physical drawings or specifications that may culminate in descriptions of geometry to be used to define a chip. Design flows from behavior to structure and ultimately to a physical implementation via a set of manual or automated transformations. At each transformation, the correctness of the transformation is tested by comparing the pre- and post-transformation design. For instance, if a power level is specified in the original behavioral description of the sensor radio, a test is run on the design in the structural domain with feedback from the physical domain to ensure this design goal is met.

In each of these domains there are a number of design options that can be selected to solve a particular problem. For instance, at the behavioral level, we can choose the wireless standard and the format in which data is transmitted by the sensor radio. In the structural

domain, we can select which particular circuit style, logic family, or clocking strategy to use. At the physical level, we have many options about how the circuit is implemented in terms of chips, boards, and enclosures. These domains can further be hierarchically divided into different levels of *design abstraction*. Classically, these have included the following for digital chips:

- Architectural or functional level
- Logic or Register Transfer level (RTL)
- Circuit level

For analog and RF circuits, the block diagram level replaces the logic level.

The relationship between description domains and levels of abstraction is elegantly shown by the *Gajski-Kuhn Y chart* in Figure 8.1. In this diagram, the three radial lines represent the behavioral, structural, and physical domains. Along each line are enumerated types of objects in that domain. In the behavioral domain, we have represented conventional software and hardware description language categories. As we move out along any of the radial axes, the increasing level of design abstraction is able to represent greater



FIG 8.1 Gajski-Kuhn Y chart

complexity. Thus, in the behavioral domain, the lowest level of abstraction is an instruction or a statement in software or HDL descriptions, respectively. Circles represent levels of similar design abstraction: the architectural, RTL, logic, and circuit levels. The particular abstraction levels and design objects may differ slightly depending on the design method.

In this chapter, we will examine how to transform a description from one domain into another while maintaining the integrity of the design. It is only in this way that we can start with a behavior and successfully build a product.

We begin by discussing some of the guiding principles that apply to most engineering projects. Then we survey the various design strategies available to the CMOS IC designer; these range from rapid prototyping or small-volume approaches to those suitable for high-volume digital, analog, or RF design. We then examine the economics of design, which can guide us to the right selection of an implementation strategy, and review documentation requirements. In conclusion, we look at the reasons for the gap in performance between custom and ASIC design methods.

8.2 Structured Design Strategies

The viability of an IC is in large part affected by the productivity that can be brought to bear on the design. This in turn depends on the efficiency with which the design can be converted from concept to architecture, to logic and memory, to circuit, and ultimately to physical layout. A good VLSI design system should provide for consistent descriptions in all three description domains (behavioral, structural, and physical) and at all relevant levels of abstraction (e.g., architecture, RTL/block, logic, circuit). The means by which this is accomplished can be measured in various terms that differ in importance based on the application. These parameters can be summarized in terms of:

- Performance—speed, power, function, flexibility
- Size of die (hence, cost of die)
- Time to design (hence, cost of engineering and schedule)
- Ease of verification, test generation, and testability (hence, cost of engineering and schedule)

Design is a continuous tradeoff to achieve adequate results for all of the above parameters. As such, the tools and methodologies used for a particular chip will be a function of these parameters. Certain end results have to be met (i.e., the chip must conform to certain performance specifications), but other constraints may depend on economics (i.e., size of die affecting yield) or even subjectivity (i.e., what one designer finds easy, another might find incomprehensible).

Given that the process of designing a system on silicon is complicated, the role of good VLSI-design aids is to reduce this complexity, increase productivity, and assure the

designer of a working product. A good method of simplifying the approach to a design is by the use of constraints and abstractions. By using constraints, the tool designer has some hope of automating procedures and taking a lot of the "legwork" (effort) out of a design. By using abstractions, the designer can collapse details and arrive at a simpler object to handle.

In this chapter, we will examine design methodologies that allow a variation in the freedom available in the design strategy. The choice, assuming all styles are equally available, should be entirely economic. According to function, suitable design methods are selected. Following these steps, the required chip cost is estimated and the quickest means of achieving that chip should be chosen. We will focus on structured approaches to design since they offer the most appropriate method of dealing with design complexity.

The successful implementation of almost any integrated circuit requires attention to the details of the engineering design process. Over the years, a number of structured design techniques have been developed to deal with complex hardware and software projects. Not surprisingly, the techniques have a great deal of commonality. Rigorous application of these techniques can drastically alter the amount of effort that has to be expended on a given project and also, in all likelihood, the chances of successful conclusion.

8.2.1 A Software Radio—A System Example

To guide you through the process of structured design, we will use as an example a hypothetical "software radio," as illustrated in Figure 8.2. This device is used to transmit and receive radio frequency (RF) signals. Information is modulated onto an RF *carrier* to transmit data, voice, or video. The RF carrier is demodulated to receive information. An ideal software radio could receive any frequency and decode or encode any type of information at any data rate. Some day, this might be possible, but given the limitations of current processes there are some bounds. To understand the impact of design methods on system solutions, we will examine the software radio in more detail. This system will then form the basis for discussion about structured approaches to design.

Figure 8.3 illustrates a typical transmit path for a generic radio transmitter, which is called an *IQ modulator*. An input data stream is encoded into *inphase* (*I*) and *quadrature* (*Q*) signals. The *I* and *Q* represent signal amplitudes of a (voltage) vector that vary instantaneously in time as shown in the bottom of Figure 8.3. For appropriate *I* and *Q* values, any form of modulated carrier can be synthesized. *I* is multiplied by an oscillator (sine)

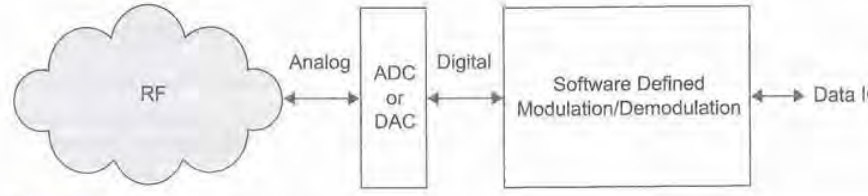


FIG 8.2 Software radio block diagram

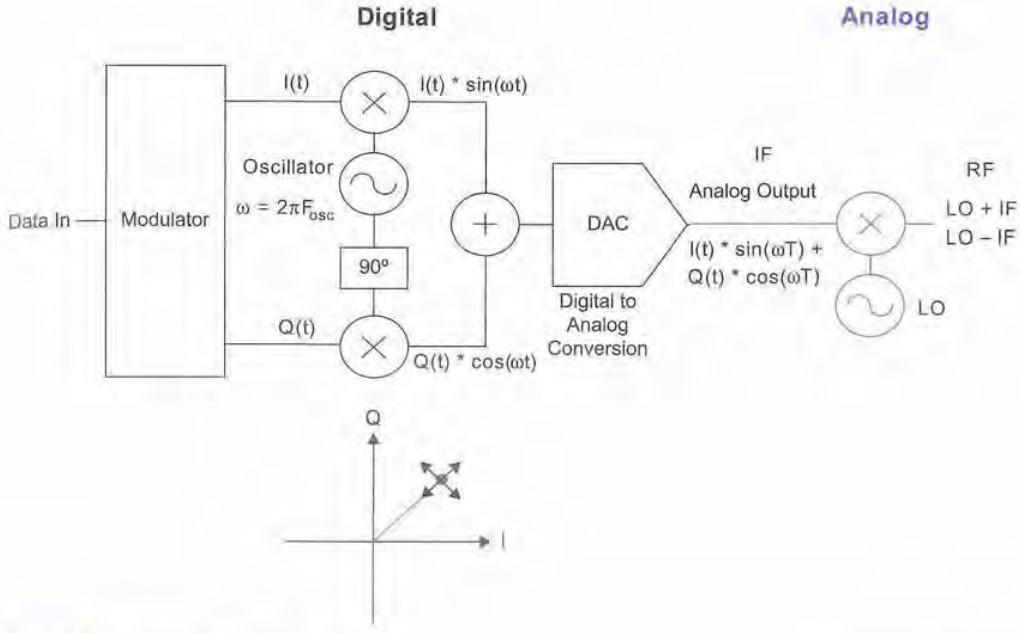


FIG 8.3 Software radio transmit path

operating at a frequency of F_{osc}. The quadrature (Q) signal is multiplied by the cosine of this frequency. The resultant signals are summed and passed to a digital-to-analog converter (DAC). In the design shown, this generates what we term an *Intermediate Frequency* or IF.

Typical IQ constellations are shown in Figure 8.4. *Amplitude Modulation* (AM), depicted in Figure 8.4(a), varies only in the magnitude of the carrier that varies in accordance with the amplitude of the modulation waveform. This is shown as a signal with an arbitrary phase angle (which we don't care about) and a vector that travels from the origin to a point on a circle that represents the maximum value of the carrier. In the case of an AM radio, the carrier frequency might be 800 KHz (in the AM band) and the modulation frequencies range from roughly 300 Hz to 6 KHz (voice and music frequencies). *Phase Modulation* is shown in Figure 8.4(b). Here, the vector travels around the maximum carrier amplitude circle varying the phase angle (δ) as the modulation changes. This is a constant amplitude modulation, which might be used with a carrier frequency of 100 MHz (in the FM broadcast band—we are loosely associating phase modulation with *frequency modulation* (FM) as they are closely related) and could have modulation frequencies of 200 Hz to 20 KHz (hi-fi audio). Finally, Figure 8.4(c) shows *Quadrature Phase Shift Keying* (QPSK) modulation, which is typical of data transmission systems. Two bits of data are encoded onto four phase points as shown in the diagram. A typical carrier frequency

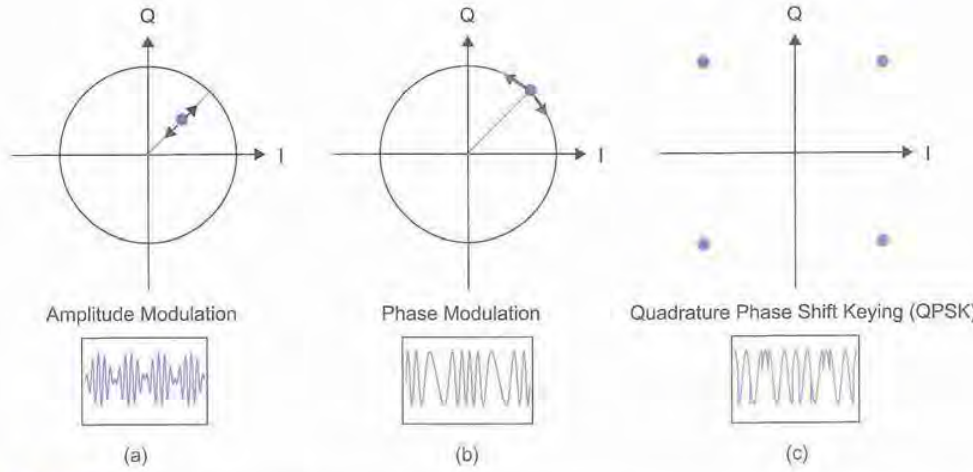


FIG 8.4 Examples of IQ modulation

might be 2.4 GHz in the Industrial Scientific and Medical (ISM) band and the modulation data rate might be 10 Megabits/second.

Clearly, the ranges of carrier and modulation frequencies vary considerably. Generally, for high carrier frequencies, the modulation can be performed at a moderate frequency and then "mixed" up to a higher frequency by analog multiplication. This is completed in the analog domain and is illustrated by the blue components on the right side of Figure 8.3. An analog multiplier (called a *mixer* in RF terminology) takes an analog Local Oscillator (LO) and the Intermediate Frequency (IF) signal that we have generated and produces sum and difference frequencies. (It is also possible to generate the desired RF frequency directly, but in this design we will use an intermediate frequency approach.) Analog bandpass filtering or a slightly more sophisticated mixer can be used to select the mixing component (LO+IF or LO-IF) that we desire. For instance, if we generate a data signal on a 20 MHz IF and mix it with a 2.4 GHz LO, we can generate a 2.402 or 2.398 GHz data signal. This is called *upconversion*.

To complete the software radio, the receive path is shown in Figure 8.5. It is roughly the reverse of the transmit path. As in the transmit case, higher frequencies can be *downconverted* to lower IF frequencies that are suitable for processing by practical ADCs. The RF signal is mixed with the LO and low pass filtered to produce the difference frequency. For example, if a 2.4 GHz LO is mixed with the 2.402 GHz RF signal, the 20 MHz IF signal is restored. An analog-to-digital converter (ADC) converts the modulated IF carrier into a digital stream of data. This data is mixed (multiplied) in the digital domain by an oscillator operating at the IF frequency. After digital low pass filtering (LPF), the original I and Q signals can be reconstructed and passed to a demodulator. For further details on digital radio, consult a communications theory text such as [Haykin00].

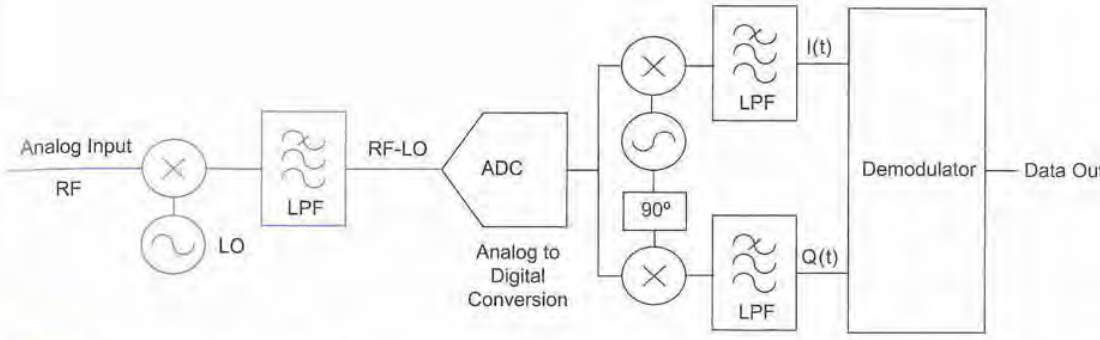


FIG 8.5 Software radio receive path

In summary, we see that multiplication, sine wave generation, and filtering are important for a software radio. While the modulation and demodulation have not been described in detail, operations can include equalization (multiplication), time to frequency conversion (fast Fourier transform), correlation, and other specialized coding operations. In the subsequent sections we will explore the design principles of hierarchy, regularity, modularity, and locality with concrete examples applied to the software radio.

8.2.2 Hierarchy

The use of hierarchy, or "divide and conquer," involves dividing a system into modules, then repeating this process on each module until the complexity of the submodules is at an appropriately comprehensible level of detail. This may entail stopping at a level where a prebuilt component is available for the particular function. The process parallels the software strategy in which large programs are split into smaller and smaller sections until simple subroutines with well-defined behavior and interfaces can be written. In the case of predefined modules, the design task involves using library code intended for the required function. The notion of "parallel hierarchy" can be used to aggregate descriptions in each of the behavioral, structural, and physical domains that represent a design (parallel hierarchy means a hierarchy—not necessarily identical—is used in each domain). Furthermore, equivalency tools can ensure the consistency of each domain. Because these tools can be applied hierarchically, you can progress in verification from the bottom to the top of a design, checking each level of hierarchy where domains are intended to correspond. For instance, a RISC processor core can have an HDL model that describes the behavior of the processor; a gate netlist that describes the type and interconnection of gates required to produce the processor; and a placement and routing description that describes how to physically build the processor in a given process. Later in the chapter, we will see how domain-to-domain comparisons are used to ensure consistency between domains.

Hierarchy allows the use of *virtual components*, soft versions of the more conventional packaged IC. Virtual components are placed into a chip design as pieces of code and come with support documentation such as verification scripts. They can be supplied by an independent *intellectual property* (IP) provider or can be reused from a previous product developed in your organization. Virtual components are discussed further in Section 8.5.7.

Example

The digital operations in the transmit path of the software radio (Figure 8.3) can be performed in software. Hence, a microprocessor can form the basis for the design. In this case, the design might have the hierarchy of a typical microprocessor, as shown in Figure 8.6. At the top level, the microprocessor contains an arithmetic logic unit (ALU), program counter (PC), register file, instruction decoder, and memory. The ALU can be further decomposed into an adder, a Boolean logic unit, and a shifter. The shifter and adder can together perform multiplication. The diagram illustrates how a relatively complex component can be rapidly decomposed into simple components within a few levels of hierarchy. Each level only has a few modules, which aids in the understanding of that level of the hierarchy.

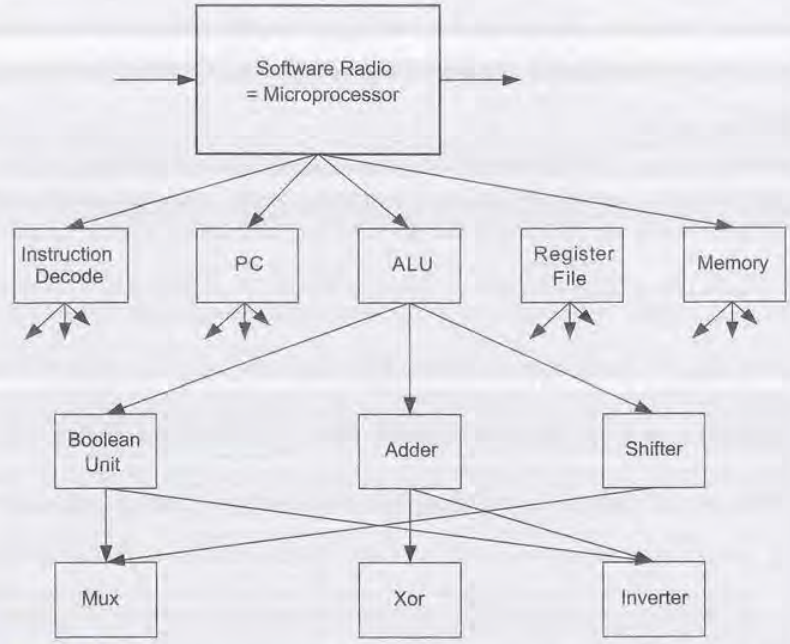


FIG 8.6 Possible hierarchy of software radio using a single microprocessor

Example

We can roughly estimate the performance required in the transmit path by noting that we require at least two multiplications, an addition, and two table lookups (sine and cosine). Another addition would be required to maintain a loop counter. An iterative multiply takes N cycles for an N-bit word, so for a 16-bit word width, the total number of cycles for the steps described would be approximately 16 + 16 + 1 + 2 + 2 + 1 (if table lookups take two clock cycles). This yields a total of roughly 40 clock cycles. For a 100 MHz processor, the fastest we could perform the IQ conversion would be approximately 400 ns, which, according to Nyquist's criteria ($F_{analog\_max} = F_{sample}/2$), would be capable of generating a 1.25 MHz IF signal. This is, of course, without any extra processing for modulating the carrier. While we could add another processor, this may be wasteful of area and power, given the operation that has to be performed.

A better approach is to use dedicated hardware for the computationally intensive fixed-function blocks. The trick is to notice that the IQ modulator portion of the software radio transmit and receive path for a given DAC and ADC resolution has a relatively fixed architecture. For the transmit path, the hierarchy shown in Figure 8.7 can be used where the blue sections have been converted to fixed function blocks. This is a relatively safe bet because the IQ upconversion is a generic communications building block. In addition to the multipliers, a device called a *Numerically Controlled Oscillator* (NCO) has been introduced [Lu93, Lu93b, Hwang02]. The NCO, described in detail in the next section, generates sine or cosine waveforms at a speed determined by the delay through an N-bit adder where N is in the range of 16 to 32 for typical NCOs. The

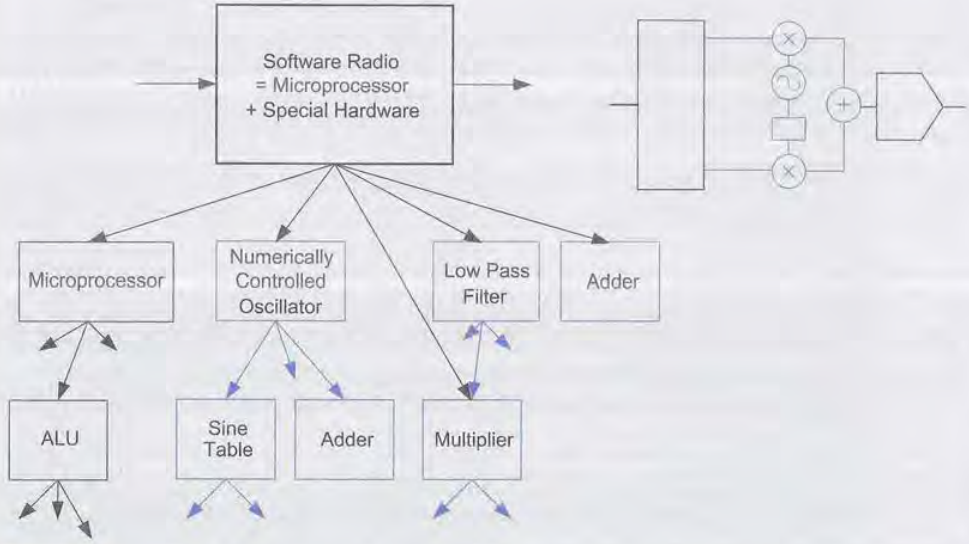


FIG 8.7 Transmit chain with dedicated IQ upconverter

move to dedicated hardware for the IQ upconversion allows the circuit to produce a new value once every clock cycle. If we conservatively say that the arithmetic blocks operate at the same speed that the microprocessor ALU does, then the circuit will now operate at 100 MHz. Taking into account sampling theory, this means that we can generate analog frequencies up to almost 50 MHz with a suitable DAC. The microprocessor now only has to respond at the modulation data rate, providing IQ values to the IQ upconverter.

8.2.3 Regularity

Hierarchy involves dividing a system into a set of submodules. However, hierarchy alone does not solve the complexity problem. For instance, we could repeatedly divide the hierarchy of a design into different submodules but still end up with a large number of different submodules. With regularity as a guide, the designer attempts to divide the hierarchy into a set of similar building blocks. Regularity can exist at all levels of the design hierarchy. At the circuit level, uniformly sized transistors can be used, while at the gate level, a finite library of fixed-height, variable-length logic gates can be used (see later in the chapter for a description of standard cells). At the logic level, parameterized RAMs and ROMs could be used in multiple places. At the architectural level, multiple identical processors can be used to boost performance.

Regularity aids in verification efforts by reducing the number of subcomponents to validate and by allowing formal verification programs (see Section 8.4.1.4) to operate more efficiently. Design reuse depends on the principle of regularity to use the same virtual component in multiple places or products.

Example

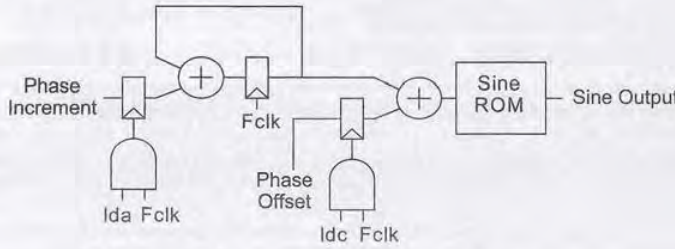
In an example of regularity applied to the software radio, we first look inside two of the blocks used in the designs shown in Figure 8.3 and Figure 8.5 to assess what kinds of functions are required.

The NCO is shown in Figure 8.8(a). It is composed of a registered adder that is incremented every clock cycle by a phase increment register. This implements a phase counter, which is used to step through a ROM lookup table that provides phase-to-amplitude conversion. A phase offset can be added to the phase incrementer to perform phase modulation. With this structure we are able to generate a digital sine wave.

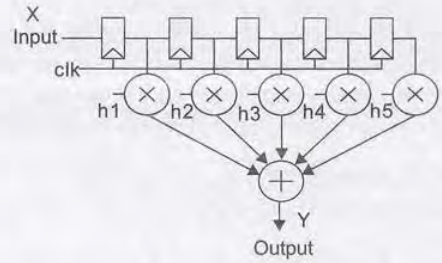
Turning to the low-pass filter shown in Figure 8.5, Figure 8.8(b) shows the structure for a commonly used low-pass filter implementation that is called a *Finite Impulse Response* (FIR) filter [Edwards93, Choi97]. The structure computes the function:

$$Y[n] = \sum X[n-k]b[k] \tag{8.1}$$

where $X[n]$ is the sampled input, $b[k]$ are the filter coefficients that characterize the particular filter, and $Y[n]$ is the output. As the structure indicates, the filter is composed of registers, multipliers, and an adder. Filters are characterized by the number of *taps* (coefficients). More taps yield better filters approaching an ideal "brick wall" filter with steeper cutoff and low ripple. This in turn requires more registers and more multipliers.



(a) Numerically Controlled Oscillator (NCO) Structure



(b) Finite Impulse Response (FIR) Filter Structure

FIG 8.8 Structure of numerically controlled oscillator and low-pass filter (implemented as a finite impulse response (FIR) filter)

Having examined the detail of these blocks, we notice that the common functions are registers, adders, and multipliers with precisions as yet undefined. Parallel N-bit adders can be composed of N single-bit full adders. Multipliers are also built from full adders. N-bit registers are built from 1-bit flip-flops. Thus, one form of

regularity might be to use the same full adder for all parallel adders and multipliers. Similarly, the same flip-flop would be used in all locations.

Typically, the phase counter adder in the NCO would be of the order of 16–32 bits wide. The phase increment adder might be 8–16 bits wide. The sizes of the multipliers and adders in the FIR filter vary widely, but depend on the input data width. This typically varies from 1–12 bits.

Example

As illustrated in the previous section, IQ upconversion and downconversion can be converted to fixed hardware, as highlighted in blue in Figure 8.9. Whether the hardware is shared (i.e., the NCO and the multipliers) is a determination that can be made at the time of design. Once this is decided, the IQ modulation and demodulation is still undefined. These blocks tend to be highly variable depending on the particular system. Software radios have been proposed in areas where the standards are likely to evolve as time progresses. Rather than have any product fixed to an old standard, a software radio allows the product to be updated in the field via a firmware update. Thus, in our quest for a software radio architecture, we still want programmability.

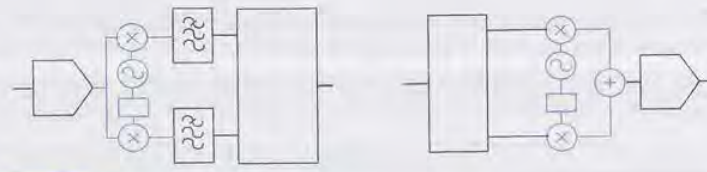


FIG 8.9 Common IQ blocks

A solution to maintaining programmability while increasing processing power might be to use a multiprocessor as shown in Figure 8.10. Here, the IQ up-and-down conversion has been retained and the IQ modulation/demodulation is performed by the four processors. The number of processors is arbitrary and would be ascertained by a detailed analysis of the required computational power.

Imagine that the computational power required slightly exceeds that provided by the four processors shown in Figure 8.10. Because multiplication is a frequently required operation in signal processing operations, it makes sense to build a multiplier into each microprocessor, as shown in Figure 8.11. Hence, we maintain regularity and improve processing power.

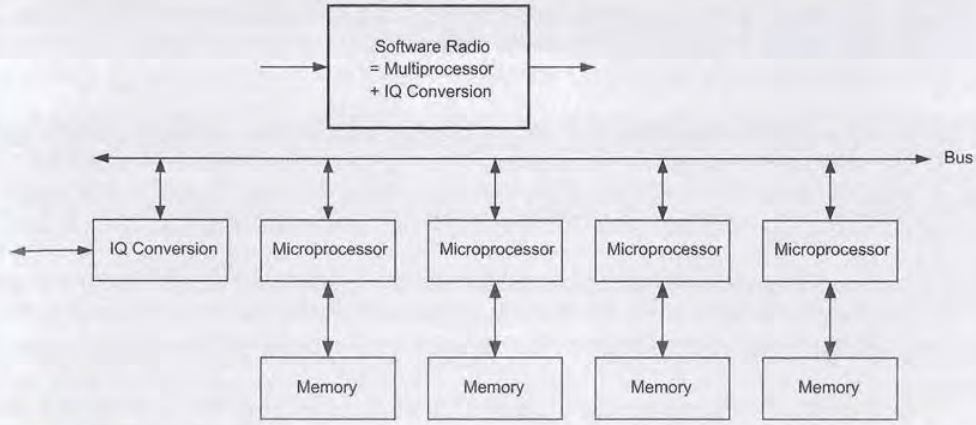


FIG 8.10 Software radio as a multiprocessor

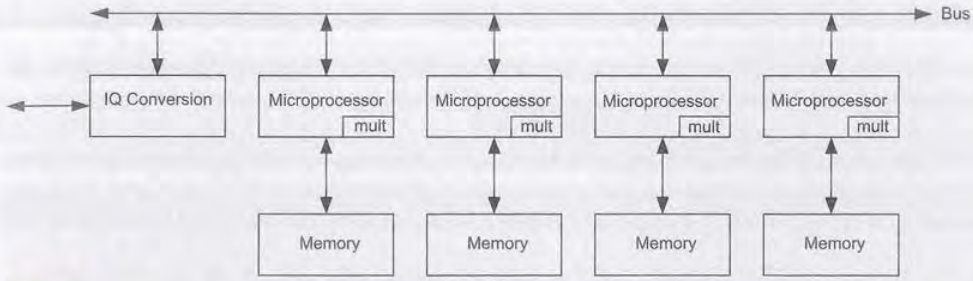


FIG 8.11 Enhanced multiprocessor for software radio

If the multiplication is a one-cycle operation, the throughput for multiplication-intensive operations can improve by a factor of up to M as compared to an M-bit processor with an iterative multiplication operation. This style of acceleration can be repeated for any operation that is computationally intensive. The application code is profiled, timing bottlenecks are identified, and custom hardware is added with appropriate instructions to access the hardware. In this manner the overall solution remains programmable while the speed of processing increases markedly. Tensilica sells extensible processors using such an approach. However, adding functional units increases die size and power dissipation, so tradeoffs are necessary.

8.2.4 Modularity

The tenet of modularity states that modules have well-defined functions and interfaces. If modules are "well-formed," the interaction with other modules can be well characterized. The notion of "well-formed" may differ from situation to situation, but a good starting point is the criteria placed on a "well-formed" software subroutine. First of all, a clearly defined interface is required. In the case of software, this is an argument list with typed variables. In the IC case, this corresponds to a clearly defined behavioral, structural, and physical interface that indicates the function as well as the name, signal type, and electrical and timing constraints of the ports on the design. Reasonable load capacitance and drive capability should be required for I/O ports. Too large a fan-in or too small a drive capability can lead to unexpected timing problems that take effort to solve, where we are trying to minimize effort. For noise immunity and predictable timing, inputs should only drive transistor gates, not diffusion terminals. The physical interface specification includes such attributes as position, connection layer, and wire width. In common with HDL descriptions, we usually classify ports as inputs, outputs, bidirectional, power, or ground. In addition, we would note whether a port is analog or digital. Modularity helps the designer clarify and document an approach to a problem, and also allows a design system to more easily check the attributes of a module as it is constructed (i.e., that outputs are not shorted to each other). The ability to divide the task into a set of well-defined modules also aids in System-On-Chip (SOC) designs where a number of IP sources have to be interfaced to complete a design.

Example

The low-pass filter in our software radio needs a large number of registers. Assuming that we wanted to implement a long filter (say 128 taps) with a 12-bit word, 3072 flip-flops are required for the I and Q filters. These can contribute significantly to the power of a portable communications IC, so it is tempting to prune transistors from the flip-flops. Figure 8.12 illustrates static and dynamic latches. The dynamic latch is much smaller but is sensitive to noise on the input and V_{DD}, suffers from the effects of leakage, and must be continuously clocked to preserve its state. The choice between the two options would depend to some extent on the design style that was being used. If the filter was being designed as a custom layout with known placement and hence wiring loads, the dynamic latch could be used. However, if the design was being automatically placed and routed, the static latch is the only prudent choice. For reasons discussed in Section 6.3, module inputs should only drive the gate terminals of MOS transistors used in a standard-cell place and route scenario. Usually, "discretion is the better part of valor" in these situations.

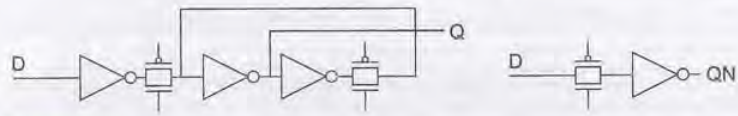


FIG 8.12 Cell level modularity example

Example

Another form of modularity, which aids in debugging and testability, is to ensure that all registers in a design are writable and readable. Figure 8.13 shows this principle applied to the NCO circuit. All registers have been made loadable via enables (lda, ldb, etc.) rather than the gated clocks shown in the original figure. Tristate buffers have been added so that the registers can also be read. Input and output registers have been added to the sine ROM to provide a temporally modular design. All modules should be registered on both their inputs and outputs. In this manner we know the input arrival time is a *D*-to-*clk* setup before the clock edge and the output timing is a *clk*-to-*Q* delay after the clock edge. The test task is identical for each register save for the number of bits that have to be written or read and the address of the register. In fact, by taking this approach, a table of register name, address, and bit width can be compiled and the HDL code to control the registers, documentation, and test programs automatically generated. With a more *ad hoc* approach, these productivity aids are not available.

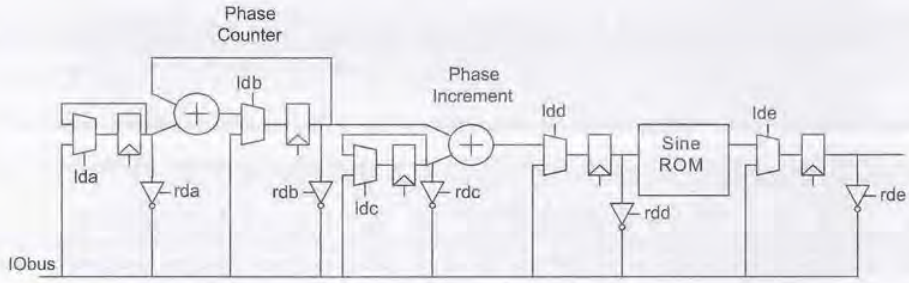


FIG 8.13 Modular test added to NCO

Example

In Figure 8.13, the phase-to-amplitude conversion is achieved using a sine ROM. The number of phase points determines the number of words in the ROM, and the desired precision determines the width of each word. A 7-bit ROM with an 8 bit-phase address will be 256 words long. Improving the phase precision involves increasing the length of the ROM. One trick to reduce the ROM length by a factor of four is to use the scheme shown in Figure 8.14. Only the first quadrant of the sine wave needs to be stored in the ROM. The sign bit (MSB) is used to conditionally invert the sine wave and the next bit in significance (*Phase*[*n*-1]) is used to count up or down through the ROM. Thus, a 256-word phase-to-amplitude table only needs 64 entries in the ROM.

While the sine lookup table is adequate for small ROMs, large ROMs can be slow and sometimes difficult to work into a standard cell design flow. An alternate structure that can

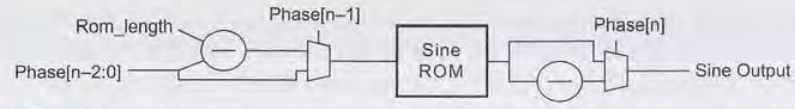


FIG 8.14 Sine ROM optimization

be used to generate a sine is called a *CORDIC processor*. The structure of the CORDIC processor is shown in Figure 8.15 [Volder59, Timmermann94, Gielis91, Grayver98, Sharma00]. It can be used to iteratively compute the functions:

$$x = x_0 \cos z_0 - y_0 \sin z_0$$
$$y = y_0 \cos z_0 + x_0 \sin z_0 \qquad (8.2)$$

Figure 8.15(a) shows a single CORDIC stage, which is composed of a parallel m-bit adder and two parallel m-bit adder/subtracters (an m-bit adder/subtracter is made up of an m-bit adder and m XOR gates, as described in Section 10.2.3.1). In addition, each stage contains two fixed shifters and a constant. The output of each stage is registered with three m-bit registers.

k stages are cascaded to perform the computation, as shown in Figure 8.15(b). If $x_0 = 1$, $y_0 = 0$, and $z_0 = \phi$, the structure will iteratively calculate $x_k \approx \cos \phi$, $y_k \approx \sin \phi$.[1] The right shift amount is equal to the iteration number (n), as shown in the figure. The constant is an approximation to the arctangent of an angle that decreases as the iteration index increases. The sign of the constant and the choice of addition or subtraction at stage n depends on the sign of z_{n-1}. See [Timmermann94] for more details. The amplitude precision is determined by the size of the adders and the phase precision is determined by the number of stages. The CORDIC processor can also compute square root, sinh, and cosh, and can perform IQ upconversion.

The objective of showing the CORDIC structure is to illustrate a possible improvement in regularity and modularity over the ROM implementation. The CORDIC only uses adders/subtracters and registers composed of regular logic gates and flip-flops, as opposed to a ROM structure. Moreover, the CORDIC is modular in three ways. First, it scales by simply extending the pipeline length or the bit width of the adders and registers. Second, it can provide a general class of polar functions, which might be of more utility in a situation where the maximum flexibility is required (i.e., as a coprocessor to a general-purpose microprocessor [Timmermann94]). If the cost is deemed satisfactory, the CORDIC provides a degree of future-proofing. Finally, it may be possible to eliminate the multipliers in the IQ

[1]Strictly speaking, CORDIC introduces a gain of approximately 1.647. To compensate, we can use $x_0 = 1/1.647$.

downconversion block by using an appropriately dimensioned CORDIC. Possible disadvantages include the pipeline delay through the CORDIC and perhaps power dissipation.

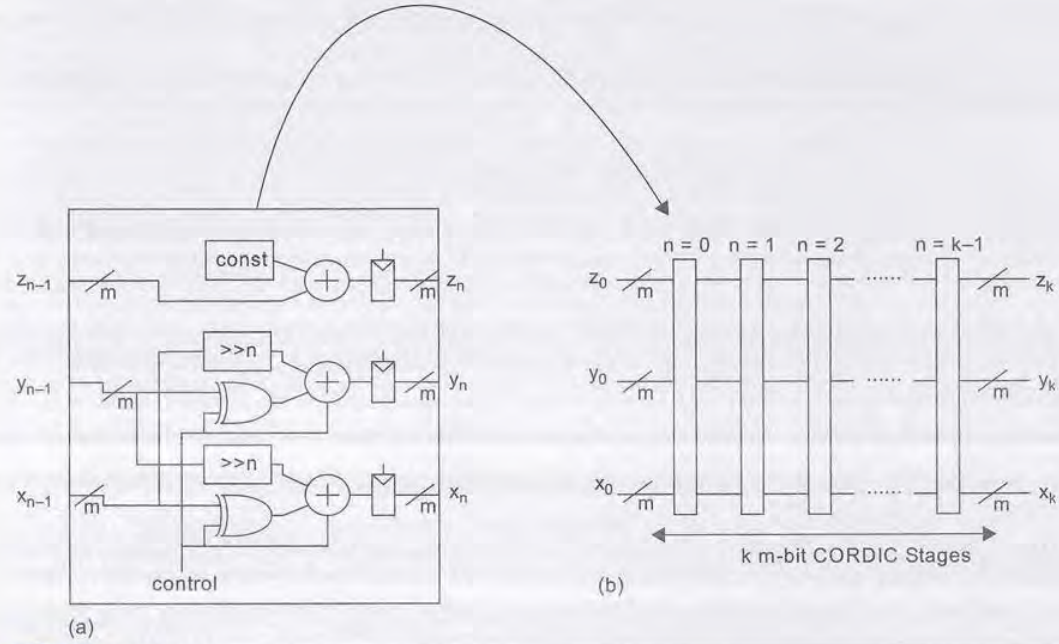


FIG 8.15 CORDIC processor as a sine generator

8.2.5 Locality

By defining well-characterized interfaces for a module, we are effectively stating that other than the specified external interfaces, the internals of the module are unimportant to other modules. In this way we are performing a form of "information hiding" that reduces the apparent complexity of the module. In the software and HDL world, this is paralleled by a reduction of global variables to a minimum (hopefully to zero). Increasingly, locality often means temporal locality or adherence to a clock or timing protocol. This is addressed in Chapter 7, where different clocking strategies are examined. One of the central themes of temporal locality is to reference all signals to a clock. Thus, input signals are specified with required setup and hold times relative to the clock, and outputs have delays related to the edges of the clock.

Example

In the example of the software radio, locality would probably be most evident in the floorplan of the chip. One example floorplan is shown in Figure 8.16. The analog blocks (ADC and DAC) are placed adjacent to the I/O pads. This is an example of physical locality because the analog blocks draw significant DC current and therefore the power busses have to be short and exhibit low resistance. Furthermore, the analog input and analog output signals can be routed to the pads without interference from digital signals. If necessary, the left edge of the chip can be guard-ringed and placed in a deep n-well if this process option is available. The digital IQ upconversion module is placed near the DAC and ADC, and the four programmable processor/memory composites are arrayed across the chip.

An alternative floorplan is shown in Figure 8.17. Here, the analog blocks and IQ conversion module are placed at the top of the chip. The four processor/memory blocks are then arrayed around a centrally located bus. The area for both array possibilities is roughly the same, but the second floorplan is better because the bus connecting the processors is shorter and hence faster and potentially dissipates less power. This is an example of physical locality used to obtain good temporal performance.

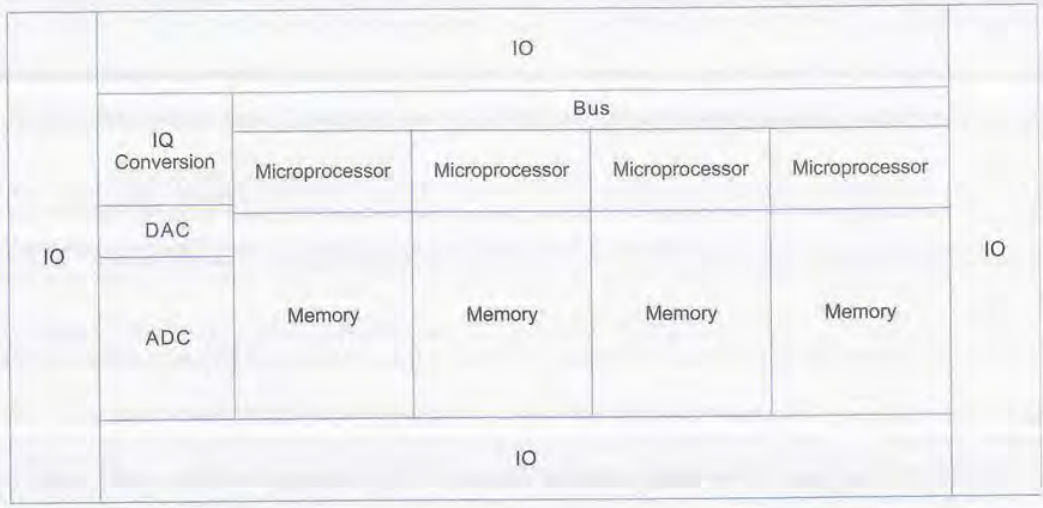


FIG 8.16 One possible floorplan for the software radio

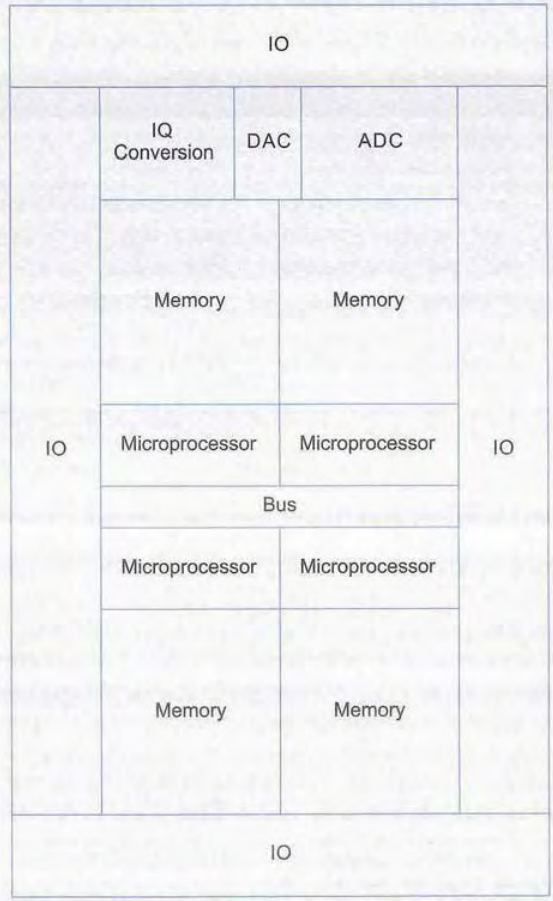


FIG 8.17 Alternate floorplan for software radio

8.2.6 Summary

There are strong parallels between the methods of design for software and hardware systems. Table 8.1 summarizes some of these parallels for the principles outlined above.

Table 8.1 Structured software and VLSI hardware design

Design Principle	Software	Hardware
Hierarchy	Subroutines, libraries	Modules
Regularity	Iteration, code sharing, object-oriented procedures	Datapaths, module reuse, regular arrays, gate arrays, standard cells
Modularity	Well-defined subroutine interfaces	Well-defined module interfaces, timing and loading data for modules, registered inputs and outputs
Locality	Local scoping, no global variables	Local connections through floorplanning

8.3 Design Methods

In this section, we will examine a range of design methods that can be used to implement a CMOS system. This section will concentrate on the target of the design method, in contrast to the design flow used to build a chip. Design flows, which deal with how a design progresses through a set of tools, will be dealt with in the subsequent section. The base design methods are arranged roughly in order of "increased investment," which loosely relates to the time and cost it takes to design and implement the system. It is important to understand the costs, capabilities, and limitations of a given implementation technology to select the right solution. For instance, it is futile to design a custom chip when an off-the-shelf solution that meets the system criteria is available for the same or lower cost.

8.3.1 Microprocessor/DSP

Many times, the most practical method to solve a system design problem is to use a standard microprocessor or digital signal processor (DSP). There are many single-chip microprocessors with built-in RAM and EEROM/EPROM available in the market. For example, the PIC family of processors from Microchip offers a wide range of clock speeds, memory sizes, and analog I/O capability (ADCs) in a small package. For more signal-intensive problems, classical DSPs from vendors such as Analog Devices and Texas Instruments can be employed. Microprocessors provide great flexibility because systems can be upgraded in the field through software patches. Do not underestimate the cost of software development for microprocessor-based systems.

Even when you decide to build a system with an off-the-shelf microprocessor, you should consider the possibility of eventual integration. For example, if your product becomes very successful and you want to reduce costs by integrating it into a single system-on-chip rather than building it as a board with a microprocessor and various support chips, you will need a microprocessor that is available in embedded form so that you can keep your software. Examples of embedded commercial processor cores include ARM, MIPS, and IBM's PowerPC. The (Sun Microsystems) SPARC V8 architecture is also available in VHDL for research or commercial use from Gaisler Research.

8.3.2 Programmable Logic

Often, the cost, speed, or power dissipation of a microprocessor may not meet system goals and an alternative solution is required. A variety of programmable chips are available that can be more efficient than general purpose microprocessors yet faster to develop than dedicated chips:

- Chips with programmable logic arrays
- Chips with programmable interconnect
- Chips with reprogrammable logic and interconnect

The system designer should be familiar with these options for two reasons:

- First, it allows the designer to competently assess a particular system requirement for an IC and recommend a solution, given the system complexity, the speed of operation, cost goals, time-to-market goals, and any other top-level concerns.
- Second, it familiarizes the IC designer with methods of making any chip reprogrammable at the hardware level and hence both more useful and of wider spread use.

8.3.2.1 Programmable Logic Devices The devices covered in this section are descended from chips that implement two-level sum-of-product programmable logic arrays (PLAs) discussed in Section 11.7. They differ from the field-programmable gate arrays described in the next section in that they have limited routing capability. Historically, process densities did not allow the transistor count and routing resources found in modern field-programmable gate arrays. Programmable logic devices based on PLAs allowed a useful product to be fielded and well-established techniques allowed logic optimization to target PLA structures, so the associated CAD tools were relatively simple. They are still occasionally used because the regular array and interconnect make timing very predictable.

A PLA consists of an AND plane and an OR plane to compute any function expressed as a sum of products. Each transistor in the AND and OR plane must be capable of being programmed to be present or not. This can be achieved by fully populating the AND and OR plane with a NOR structure at each PLA location. Each node is pro-

grammed with a floating-gate transistor (see Section 3.4.3), a fusible link, or a RAM-controlled transistor, as illustrated in Figure 8.18. The first two versions were the way these types of devices were programmed when device densities were low. These devices are generally used for low-complexity problems (a few hundred gates), which require fairly high speed (100's MHz).

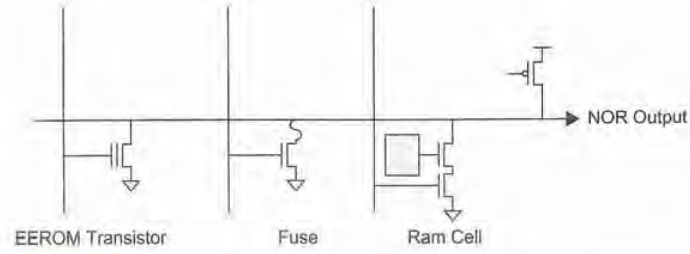


FIG 8.18 PLA NOR structure (one plane shown)

8.3.2.2 Field-programmable Gate Arrays (FPGAs) Field-Programmable Gate Arrays (FPGAs) use the high circuit densities in modern processes to construct ICs that, as their name suggests, are completely programmable even after a product is shipped or "in the field." Two basic versions exist. The first uses a special process option such as a fuse or antifuse to permanently program interconnect and personalize logic. These are one-time programmable. The second type uses small static RAM cells to customize routing and logic functions. In general, an FPGA chip consists of an array of logic cells surrounded by programmable routing resources.

As an example of the first type of FPGA, devices manufactured by Actel embed an array of logic modules within an interconnect matrix that is formed on the top metal layers. Successive routing channels run vertically or horizontally, as shown in Figure 8.19. At the intersection of routing traces is placed a special one-time programmable contact called an *antifuse*. These normally have high resistance (effectively an open circuit). Upon application of a special programming voltage across the contact, the resistance permanently drops to a few ohms. CMOS switches allow the programming voltage to be directed to any antifuse in the chip. The advantage of this type of routing is that the size of the programmable interconnect is tiny—the intersection area of two metal traces. Moreover, the on-resistance is low compared to a CMOS switch, so the circuit speed is not compromised. The disadvantage is that the interconnect is not reprogrammable, so once a chip is programmed, its function is fixed to the extent that the interconnect has been personalized.

The floorplan of a typical Actel chip is shown in Figure 8.20. An array of logic modules (LM) is placed on the chip with the routing passing adjacent to the cells. Routes extend into and out of the logic modules and intersect vertical or horizontal routing channels. The cell is personalized in terms of functionality and connectivity (to other cells) by

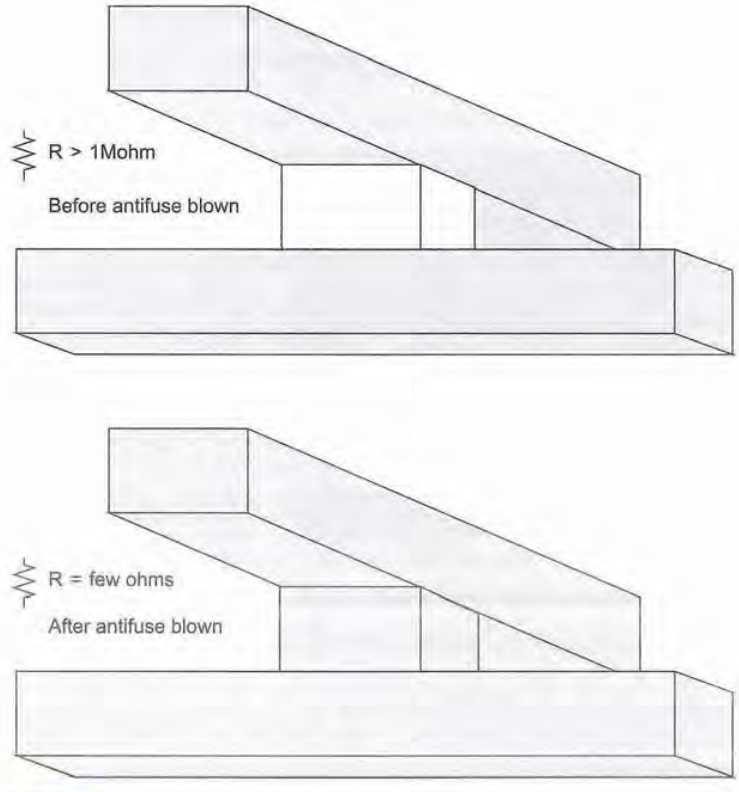


FIG 8.19 Actel programmable interconnect

programming these crossing interconnects. The logic module consists of a combinational block, which can compute more than 4000 combinational functions of five primary inputs. A flip-flop cell provides for asynchronous set and reset, active low enable, and programmable clock polarity. With typical utilization, effective gate count (equivalent two input gates) can range from 82K to over 1 million gates.

This type of array would be suitable for reasonably complex, high-speed (100's MHz) applications where cost and power constraints were moderate. Antifuses are only available in specialized CMOS processes.

Figure 8.21 shows the floorplan of a simplified RAM-based FPGA. The chip is composed of an array of *configurable logic blocks* (CLBs). Similar to the one-time programmable devices, metal routing tracks run vertically and horizontally between the array of CLBs. These terminate at the gray blocks, which are routing switches that can be implemented using CMOS transmission gates or tristate buffers. The routing resources can also be con-

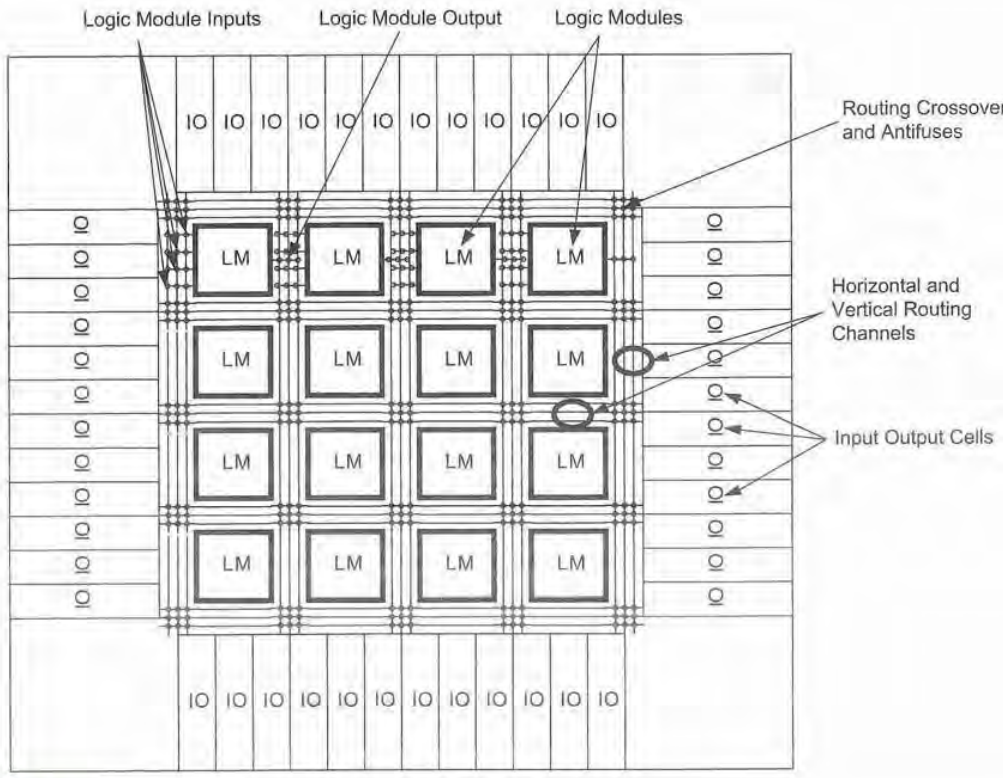


FIG 8.20 Representative Actel FPGA floorplan

nected to the inputs and outputs of the adjacent CLBs. CLBs use lookup tables to compute any function of several variables. The contents of the lookup table are stored in static RAM so the FPGA is programmed by reading the SRAM at startup from an external data source such as a serial EEPROM. Similarly, static RAM cells program the routing switches. Configurable I/O cells that can be used as input, output, or bidirectional pads surround the core array of CLBs.

A simple FPGA logic cell is shown in Figure 8.22. It is composed of a 16 × 1 static RAM as the logic element. This provides for any logic function of four variables merely by loading the RAM with the appropriate contents. Table 8.2 illustrates how the table should be loaded to perform various logic functions. A full adder can be implemented in two CLBs (one for carry and one for sum). The CLB shown also provides an optional output

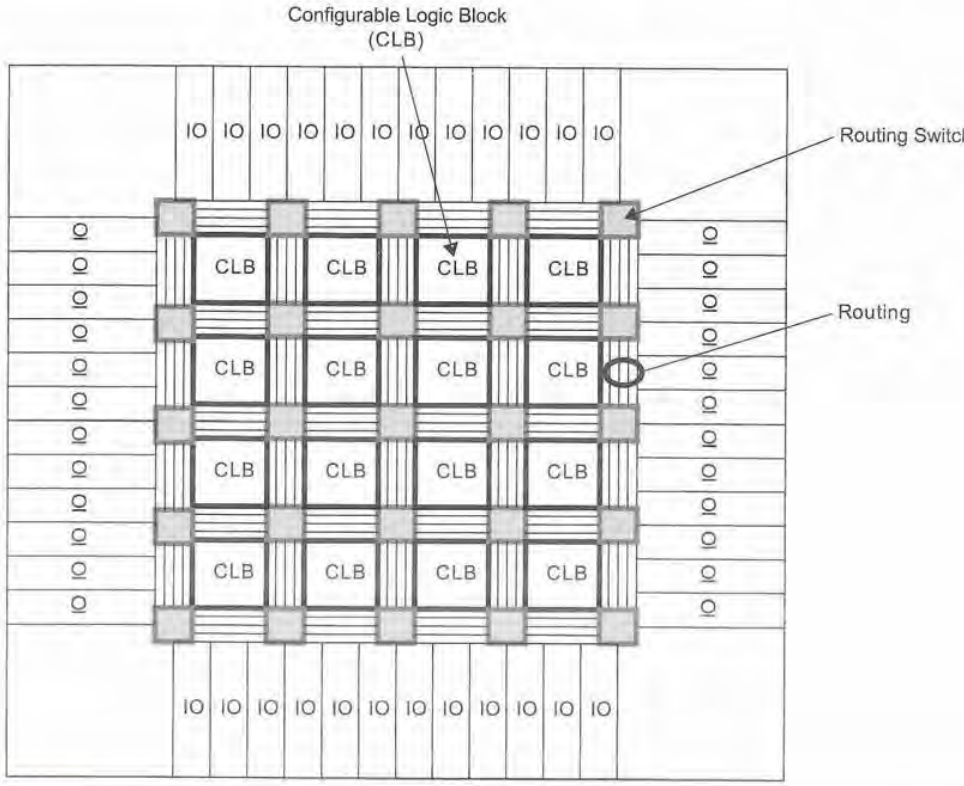


FIG 8.21 Simplified FPGA floorplan

register. While it may seem inefficient or slow to use a RAM to perform logic, specially designed single-data line RAMs are small and fast in current processes, and resources such as the routing tend to dominate modern designs from a density and speed viewpoint. The cell shown in Figure 8.22 is representative of first-generation FPGAs, and many optimizations have been made to improve performance. These include multiple dedicated clock lines, CLBs optimized for fast arithmetic, dedicated blocks of RAM, high-speed I/O interfaces, and even embedded RISC microprocessors. The Xilinx XCVP125 FPGA, state of the art in 2003, combines over 125,000 logic cells (approximately seven million two input gates), four RISC processors, multi-gigabit I/O and over 1000 user-available I/O pads (i.e., XILINX XC2VP125).

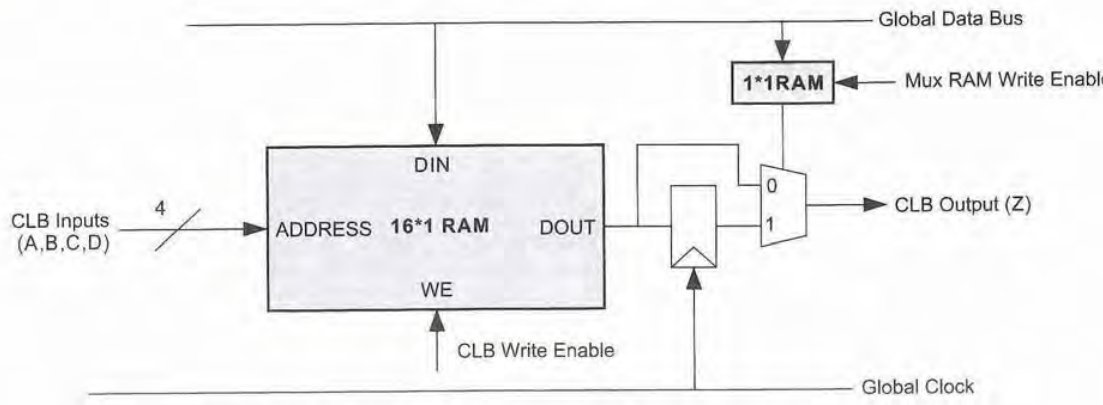
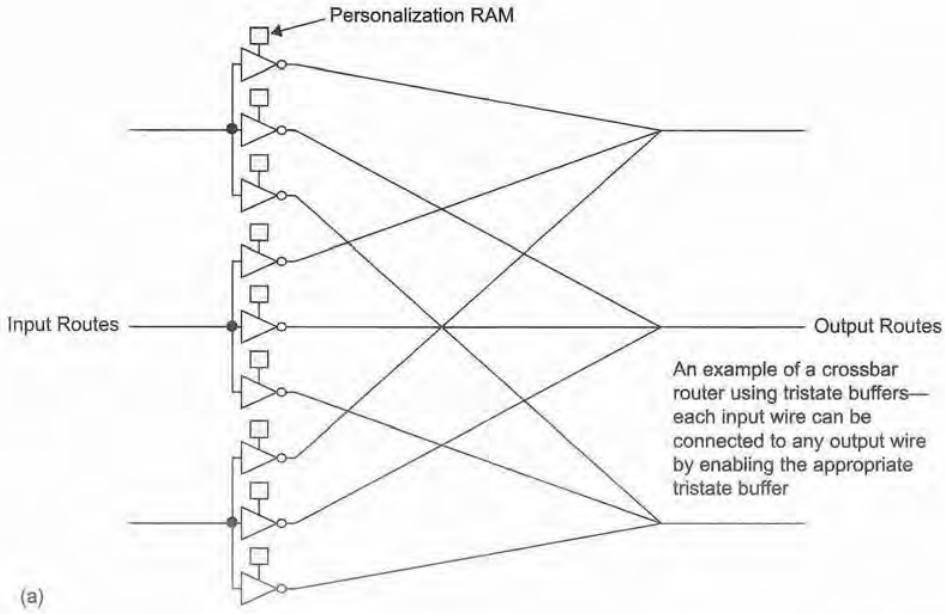


FIG 8.22 Simple FPGA logic cell

Table 8.2 RAM CLB functions

Address	*ABCD*	*A • B • C • D*	~A	SUM(A,B,C)
0	0000	0	1	0
1	1000	0	0	1
2	0100	0	1	1
3	1100	0	0	0
4	0010	0	1	1
5	1010	0	0	0
6	0110	0	1	0
7	1110	0	0	1
8	0001	0	1	0
9	1001	0	0	1
10	0101	0	1	1
11	1101	0	0	0
12	0011	0	1	1
13	1011	0	0	0
14	0111	0	1	0
15	1111	1	0	1



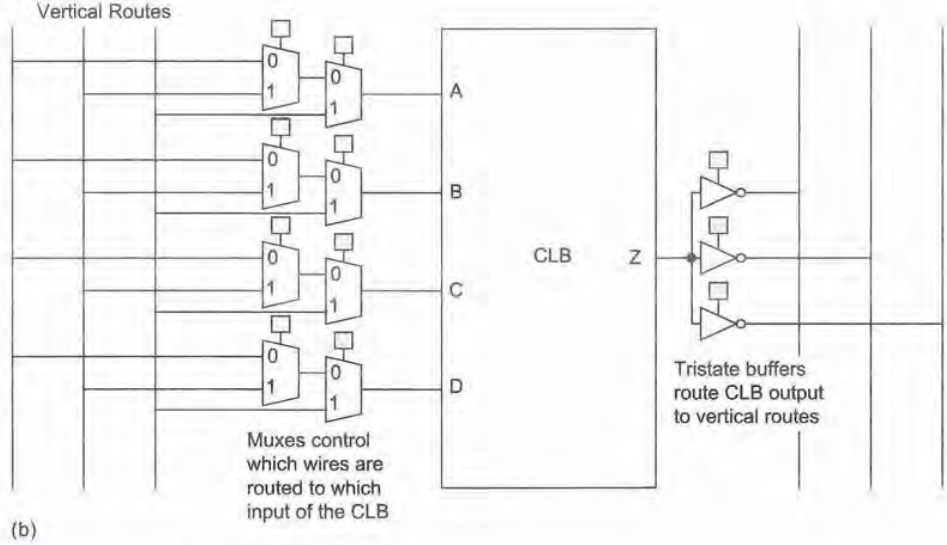


FIG 8.23 Simplified FPGA routing cell

One approach to routing cells is shown in Figure 8.23. The upper diagram shows a horizontal routing switch. Tristate buffers controlled by RAM bits determine which output routes are driven. The lower diagram shows the routing to the CLB of Figure 8.22. Each input can be connected to any of the three vertical routes to the left of the cell, while the tristate buffers can connect the output of the CLB to any route along the right of the cell.

Putting it all together, Figure 8.24 shows an example of a personalized array with the associated routing. Three inverting input pads enter on the left and are routed to two XOR gates, the output of which is routed out of the chip on the right.

FPGAs have matured to the point where they are the best choice for many low- to medium-volume custom logic applications. The economy of scale of modern processes allows extremely high densities of reprogrammable logic gates to be delivered at low cost to the designer, and hence to the consumer. Compared to conventional ICs, FPGAs are slower and are less efficient in area and power. However, the cost to manufacture is virtually zero (programming cost). The cost per part is moderate to high. As initial manufacturing costs for custom integrated circuits increase and FPGA costs decrease, FPGAs become attractive for more and more low- to mid-volume, moderate power applications.

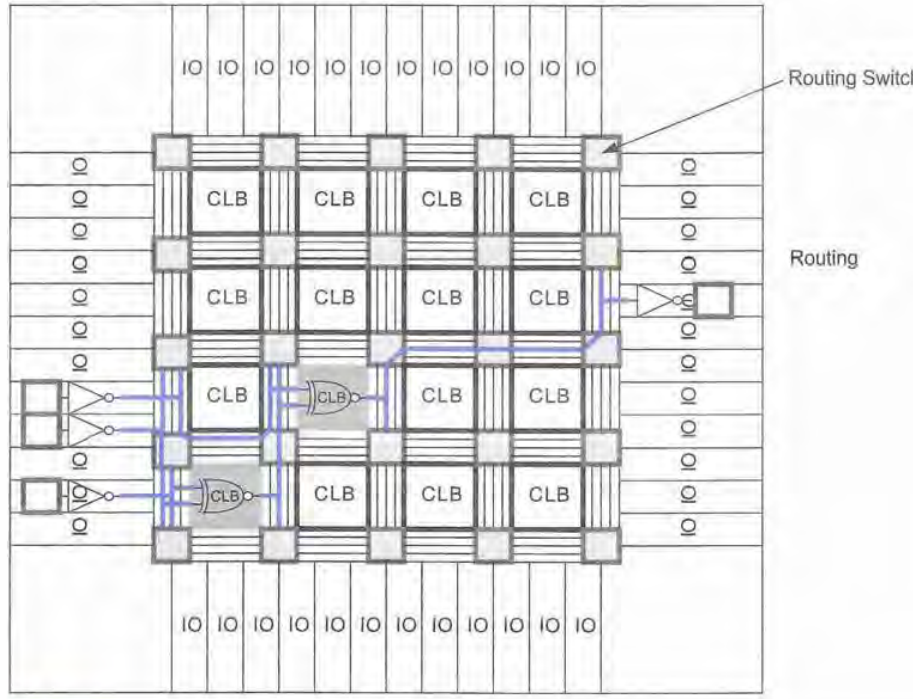


FIG 8.24 Personalized and routed FPGA

Note that (after sorting out the intellectual property rights with the appropriate patent holders) it is possible to implement FPGA blocks on any CMOS chip to provide some degree of programmability at the gate level.

8.3.3 Gate Array and Sea of Gates Design

The chips described in the previous section do not require a fabrication run. Designers typically strive to keep the nonrecurring engineering cost (NRE, see Section 8.5) as low as possible. One method of doing this is to construct a common base array of transistors and personalize the chip by altering the metallization (metal and via masks) that is placed on top of the transistors. This style of chip is called a Gate Array (GA). A particular subclass of a gate array is known as a Sea-of-Gates (SOG) chip. A vendor stocks *master* or *base* wafers that have been processed up to the polysilicon gate layer (i.e., the transistors have been formed). Contacts and metallization are then specified on a per-design basis to complete the chip. The cost can be kept low due to the following factors:

- The wafer cost can be reduced by producing many base wafers for a variety of different chips.
- Only the metallization masks are required to personalize the design; cost and processing time can be kept low for a small number of metal layers.
- Packaging cost is reduced by using standard packages and pinouts.
- Test and production costs are reduced by reusing common test fixtures.

Companies such as LSI Logic have pushed gate array technology to an advanced point by combining gate array structures with dedicated high-speed I/O circuitry. Their RapidChip technology boasts integrated RISC cores combined with nearly 20 million gates, 10 Mbits of RAM memory, and I/O speeds up to 4.25 Gbps. Speeds in the range of 300 MHz are possible with specially designed logic blocks.

Aside from their use as a System on Chip technology, it is worthwhile reviewing GA/SOG techniques because they can also be used on custom chips to provide an area of reprogrammable logic on an otherwise fixed function chip. The basic GA/SOG approach is shown in Figure 8.25. Rows of nMOS and pMOS transistors are arrayed across a chip area. Each logic row consists of an n row and p row. Gate arrays differ from SOG in that the array of transistors is not continuous and transistors can be grouped and perhaps individually sized for economic implementation of specific applications such as memories. Figure 8.26(a) shows an SOG structure, which features continuous rows of transistors. Figure 8.26(b) shows a gate array structure that uses groups of three transistor pairs. Because the SOG structure is a continuous array of transistors, it has to allow for the isolation of a group of transistors. Grounding the gate of the nMOS transistor or connecting the gate of the pMOS transistor to the V_{DD} rail provides this isolation.

Figure 8.27 shows a portion of an SOG structure programmed to be a 3-input NAND gate. Note that the nMOS and pMOS transistors at each end isolate the gate, as described previously. Personalization of this SOG structure commences at contact and

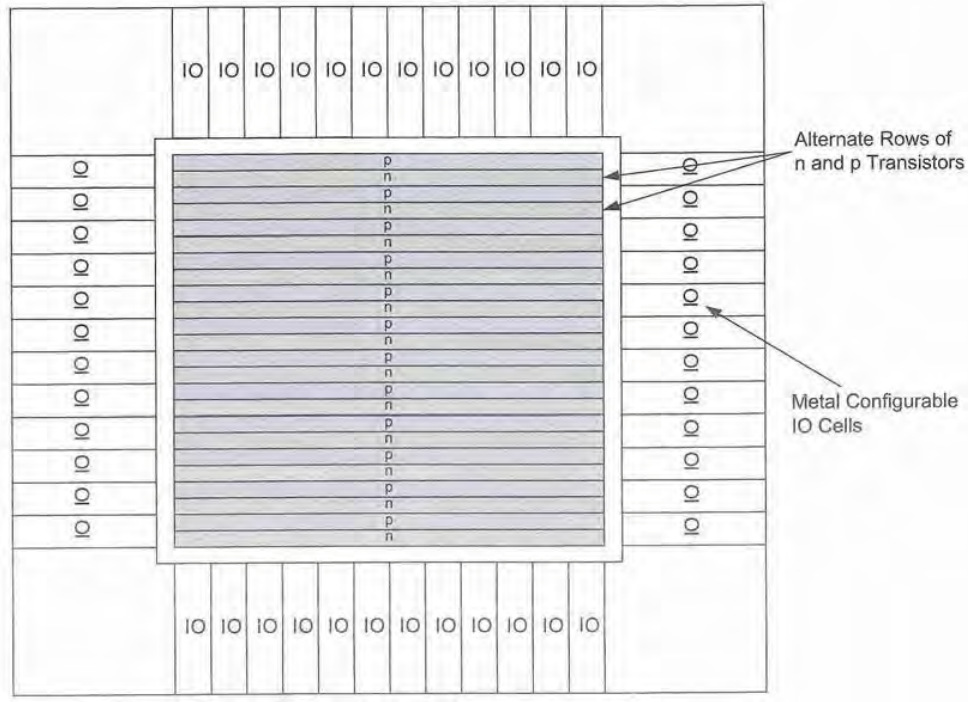


FIG 8.25 Sea of Gates floorplan and transistor layout

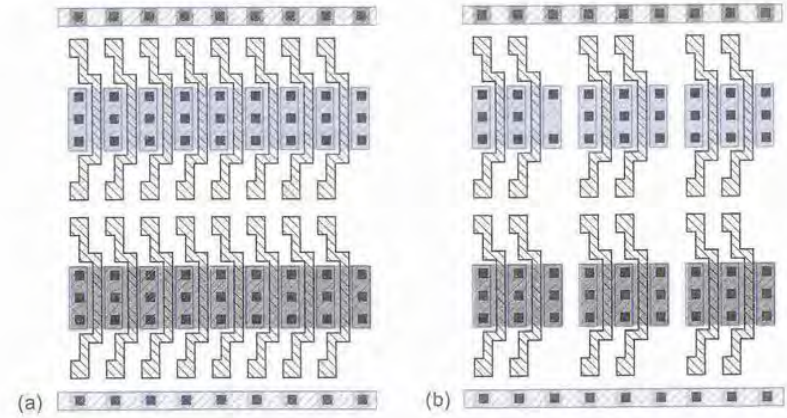


FIG 8.26 SOG and Gate Array cell layouts

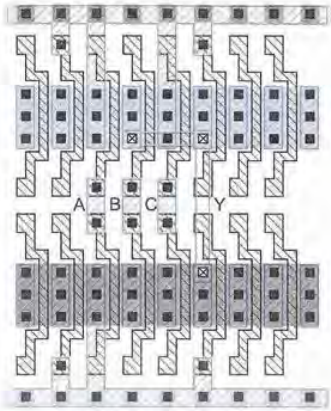


FIG 8.27 SOG programming example for 3-input NAND gate

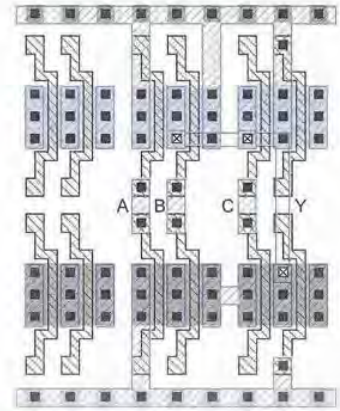


FIG 8.28 Gate Array programming example for 3-input NAND gate

metal1 masks, and can continue up for all metal layers available in the process. Figure 8.28 shows the 3-input NAND on a gate array.

Gate and SOG arrays have lower up-front costs because they require fewer masks. They are also faster to manufacture because only the metallization steps are required. Some vendors offer automatic conversions of field-programmable gate arrays into fixed gate arrays to reduce costs as a product moves from low- to mid-volume manufacturing.

Compared to an FPGA given the same process, a gate array dissipates less power and costs less in volume production. The NRE cost (i.e., cost to fabricate) can be a fraction of the cost of a custom mask set.

8.3.4 Cell-based Design

Cell-based design uses a standard cell library as the basic building blocks of a chip. The cells are placed in appropriate positions, then their interconnections are routed. Cell-based design can deliver smaller, faster, and lower-power chips than gate arrays or programmable logic but has high NRE costs to produce the custom mask set. Therefore, it is only economical for high volume parts or when the performance commands a lucrative sales price. As compared to full-custom design, cell-based design offers much higher productivity because it uses predesigned cells with layouts. Foundries and library vendors supply cells with a wide range of functionality. These include:

- Small-scale integration (SSI) logic (NAND, NOR, XOR, AOI, OAI, inverters, buffers, registers)

- Memories (RAM, ROM, CAM, register files)
- System level modules such as processors, protocol processors, serial interfaces, and bus interfaces
- Possibility of mixed-signal and RF modules

Whereas Medium Scale Integration (MSI) functions such as adders, multipliers, and parity blocks used to be supplied as cells, synthesis engines commonly construct these from base-level Small Scale Integration (SSI) gates in current design systems.

A typical standard cell library is shown in Table 8.3. A 1X (normal power) cell commonly is defined to use the widest transistors that fit within the vertical pitch of the standard cell. 2X and larger (high power) cells use wider transistors to deliver more current. They must fold the transistors (see Section 8.8.3) to fit within the cell; this comes at the expense of increased cell width. Gates are often available in low power versions as well. These cells use minimum-width transistors to reduce capacitance. Low-power cells tend to be slow because of the wire capacitance they must drive. Although they do not save area, they do reduce power consumption on noncritical paths.

Sophisticated libraries also generate memories of assorted sizes from a graphical user interface. The generators yield not only the physical layout but also a complete data sheet indicating access times, cycle times, and power dissipation.

In the event that a standard cell library may not be available for a process, it is worthwhile to review some of the approaches to standard cell design. Usually, standard cells are a fixed height with power and ground routed respectively at the top and bottom of the cells, as was shown in Figure 1.62. This allows the cells to be abutted end to end and to have the supply rails connect. A single row of nMOS transistors adjacent to GND (ground) and a single row of pMOS transistors adjacent to V_{DD} (power) are normally used. The polysilicon gate is connected from nMOS transistor to pMOS transistor and, in the case of multiplexers and registers, the polysilicon connection has to be crossed between vertically coincident nMOS and pMOS transistors. Decisions about the sizes of transistors have to be made. Following this decision, the cells are almost completely defined by the process design rules. Figure 8.29 illustrates this point. The height of the cell is defined by the sum of the nMOS and pMOS transistor widths, the separation on n and p regions, the spacing to V_{DD} and GND busses, and the width of these busses. The horizontal pitch is defined by the poly-to-metal2 contacted pitch, as shown in the figure. It is relatively easy to construct a software program to automatically generate cells like those shown in Figure 8.29. Cell delay is characterized through simulation to good agreement with silicon, as was discussed in Section 5.5.3. Fabrication of such cells to prove performance is rarely required. Options to standard cells include routing the clock with the power and ground busses and routing multiple supply voltages to each cell. The latter technique is sometimes used to reduce power by connecting gates that are not in the critical path to a lower than normal supply voltage. Recall that the power drops with the square of the supply voltage.

Table 8.3 Typical standard cell library

Gate Type	Variations	Options
Inverter / buffer / tristate buffers		Wide range of power options, 1X, 2X, 4X, 8X, 16X, 32X, 64X minimum size inverter
NAND / AND	2–8 inputs	High, normal, low power
NOR / OR	2–8 inputs	High, normal, low power
XOR / XNOR		High, normal, low power
AOI / OAI		High, normal, low power
Multiplexers	Inverting/noninverting	High, normal, low power
Schmitt trigger		High, normal, low power
Adder / half adder		High, normal, low power
Latches		High, normal, low power
Flip-flops	D, with and without synch/asych set and reset, scan	High, normal, low power
I/O pads	Input, output, tristate, bidirectional, boundary scan, slew rate limited, crystal oscillator	Various drive levels (1–16 mA) and logic levels

8.3.5 Full Custom Design

A number of techniques can be used to design standard cells or larger circuit blocks at the mask level. The oldest and most traditional technique is termed *custom mask layout*, in which a designer sits in front of a graphics display running an interactive editor and pieces designs together at the geometry level one rectangle at a time. This work is sometimes called polygon pushing. Historically, when there were no interactive graphics displays or editors (yes, there was a time when this was true!!), designers manually cut masks from a material called Rubylith. Sheets of the material (a red strippable layer on an otherwise clear Mylar plastic background) were used directly to produce photomasks for chips. Mask design under such conditions was laborious and prone to error because there were no tools to check the designs other than human eyes.

A variation of custom mask design is called *symbolic layout*. Rather than dealing with rectangles and polygons on various mask levels, the primitives are transistors, contacts, wires, and ports (points of connection). These primitives can also be manipulated by a graphics editor. Some systems allow for a "design rule free" placement of symbolic entities. The actual placement occurs after a spacing process that compacts each primitive as close to its neighbor as possible according to the design rules of the process in use. By using a symbolic layout system, layout topologies can be transported from process to process without a huge amount of effort.

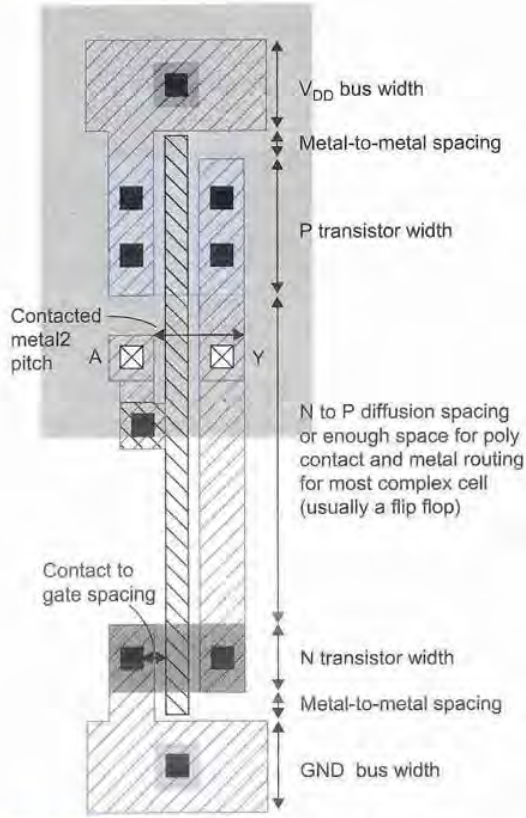


FIG 8.29 Typical standard cell layout with some of the constraints

In these times of cell-based design, digital CMOS ICs use custom mask design only for the highest of volume parts such as microprocessor datapaths. However, analog and RF designs, cell libraries, memories, and I/O cells still frequently use custom design. There are a variety of custom MOS layout hints in Section 8.8. Custom design is also worthwhile pedagogically because it completes the link from transistors to systems.

From time to time, we have mentioned software generators as a method of generating physical layout. This kind of idea has been around for a long time and was often referred to as *silicon compilation*. Complete microprocessors were typical of layouts that were generated. A "correct by construction" method was used to build the layouts hierarchically. In other words, only the mask description was generated, with perhaps a high-level instruction level simulator being the behavioral model. Generators are the most common method used today for library generation.

With modern design flows, many different "views" of a design are required to integrate with the regular path through the design system. For instance, in addition to the behavioral model, a timing view would be needed for timing verification, a logic view might be required for simulation, and a circuit view for layout versus schematic or netlist comparisons would be needed. Software generators can be used to provide all of these views automatically.

Modern versions of the venerable "silicon compiler" can be built in a structured hierarchical manner to generate memories, register files, and other special-purpose structures that can benefit from a customized layout. One of the most straightforward approaches is to write custom placement routines that in essence "hand place" certain standard cells within the row structure of a standard cell design. For instance, you may prefer a certain adder design and have a datapath layout for the adder. An algorithm can be written to place the cells on the standard cell grid. In addition, a linked algorithm can be written to generate a gate netlist in an HDL. In this way, both the physical and structural design are captured. The behavior can be represented by an HDL function or module call. Such custom placement can shorten wire lengths and thus improve speed and power.

To illustrate a memory generator, consider the sine ROM required for the NCO in the software radio. The ROM contents are available as a file with a list of numbers and we want to generate a layout and the Verilog. Generating the Verilog is straightforward and would proceed according to the following pseudo-code (loosely based on the C language):

```
function generate_ROM_verilog(ROM_length, ROM_width, ROM_filename);
   verilog_file_pointer = open("sinerom.v");
   generate_verilog_head(verilog_file_pointer, ROM_length, ROM_width);
   ROM_file_pointer = open(ROM_filename);
   index = 0;
   loop for index below 2<<ROM_length /* rom_length=6 -> 64 */
      ROM_entry = get_line_from_file(file_pointer);
      fprintf(verilog_file_pointer, " %d: q = %d'h%h",
              Index, ROM_width, ROM_entry);
      index++;
      file_pointer++;
   end_loop
   generate_verilog_tail(verilog_file_pointer);
end_function;

function generate_verilog_head(file_pointer, length, width);
   fprintf(file_pointer, "module quarter_wave(");
   fprintf(file_pointer, " input [%d:0] addr,", length-1);
   fprintf(file_pointer, " output reg [%d:0] q;)", width-1);
   fprintf(file_pointer, " always @(addr)");
   fprintf(file_pointer, " begin");
   fprintf(file_pointer, " case(addr)");
end_function;

function generate_verilog_tail(file_pointer);
   fprintf(file_pointer, " endcase");
   fprintf(file_pointer, " end");
   fprintf(file_pointer, "endmodule");
end_function;
```

The function is quite easy to follow. It first writes out the Verilog file header structure. This includes the module name, input/output ports, and the bit widths of the ports. These are specified to the function from the calling function. The program then sequences through the ROM file contents and formats each line of the case statement. Finally, the tail of the Verilog file is generated. The following code segment illustrates the output.

```
module quarter_wave(input      [5:0] addr,
                    output reg [6:0] q);
   always @(addr)
   begin
      case(addr)
         0: q = 7'h00;
         1: q = 7'h03;
         2: q = 7'h06;
         ...
         62: q = 7'h7f;
         63: q = 7'h7f;
      endcase
   end
endmodule
```

While this function is basic, the sophistication can be built up over time. There are a number of academic and commercial projects that have been aimed at HDL generation such as Verilog++ and Verilog2C++.

For the generation of the physical layout, we will concentrate on the generation of the ROM array. Details on ROM design may be found in Section 11.4. The main program is shown below. The program assumes that layout cells or tiles connect by abutment, i.e., the routing is implicit in the tile.

```
function generate_rom_layout(ROM_length, ROM_width, ROM_filename);
   lfp = open("sinerom.lay"); /*layout file pointer*/
   generate_rom(lfp, 0, 0, ROM_length, ROM_width, ROM_filename);
   x = length_of_rom_cell * ROM_length;
   generate_rom_right(lfp, x, ROM_length, ROM_width, ROM_filename);
endfunction
```

The notion of *tiling* is introduced in this code. The function `generate_rom` tiles two rectangular blocks horizontally. The first is the ROM array and row decoder. Then the `rom_right` is placed to the right of the array. This contains the ROM sense amplifier, which in this case is shown as a pMOS pull-up and buffer inverter for the pseudo-nMOS NOR structure used in the ROM. The bottom right of the `rom_right` structure can contain a column decoder, but it is not necessary in this design. The address buffers for the row decoder are placed in this corner. This physical structure is shown in Figure 8.30.

```
function generate_rom(lfp, xorigin, yorigin, length, width);
   generate_row_decoder(lfp, xorigin, yorigin, length, width);
   y = yorigin + row_decoder_height;
   generate_rom_array(lfp, xorigin, y, length, width);
end_function
```

The function `generate_rom` tiles two layout blocks vertically. The lower block is the ROM row decoder and the upper block is the ROM array proper. The row decoder gate is shown to the right of the floorplan. In this design, a pseudo-nMOS NOR gate is used for compactness and simplicity. This is shown in Figure 8.31.

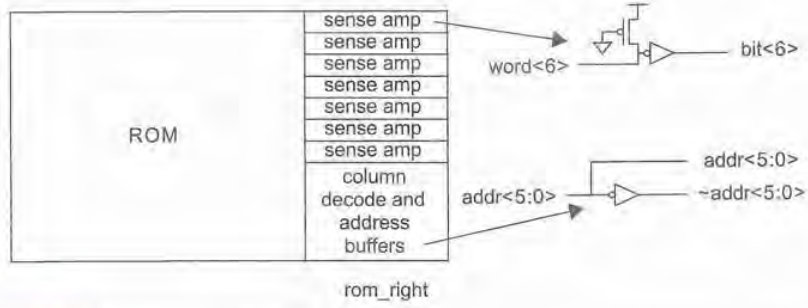


FIG 8.30 ROM layout—phase 1

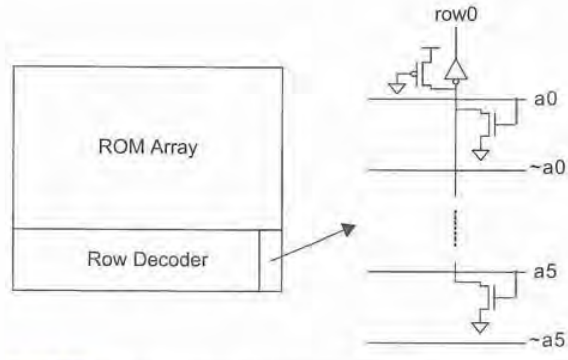


FIG 8.31 ROM layout—phase 2

Finally, the function `generate_rom_array` is invoked to build the ROM array itself.

```
function generate_rom_array(lfp, xorigin, yorigin, ROM_length, ROM_width, ROM_filename);
   ROM_file_pointer = open(ROM_filename);
   index = 0;
   x = xorigin;
   loop for index below 2<<ROM_length /* rom_length=6 -> 64 */
      ROM_entry = get_line_from_file(file_pointer);
      bit_index = 0;
      y = yorigin;
      loop for bit_index below ROM_width
         bit = get_bit_from_entry(bit_index, ROM_entry);
         if (bit) ROM_one(lfp, x, y);
         else ROM_zero(lfp, x, y);
         y = y + yorigin;
      end_loop
      x = x + rom_bit_width;
      index++;
      file_pointer++;
   end_loop
end_function;
```

This function tiles in both *x* and *y* directions and builds a ROM array as shown in Figure 8.32.

This builds a long, skinny ROM. For the sine ROM example, the array would be 64 cells long by 7 cells high. The loop variables can be changed in the function to generate, for example, four ROM words vertically, which would result in a structure that was 16 cells long by 28 cells high. A column decoder would be added to the ROM sense amplifier in the `rom_right` block to select one bit out of each group of four.

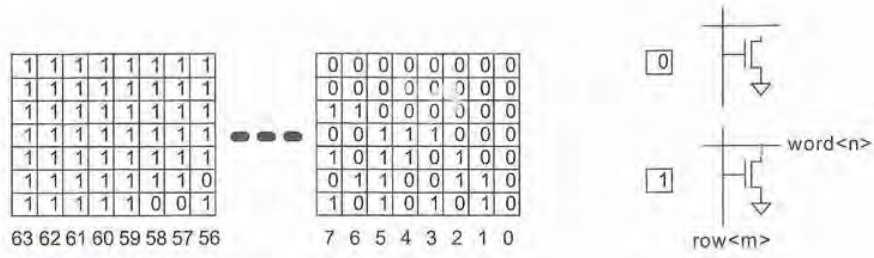


FIG 8.32 ROM array layout

Datapaths are another class of structure that can benefit from a software generator. The following code is representative of what might be used to place the cells in a CORDIC processor. Routing may be able to be incorporated into the tiled cells, or specific routing cells may have to be generated in addition to the pseudo-code shown below.

```
function generate_cordic_layout(bit_width, length);
   lfp = open("cordic.lay"); /*layout file pointer*/
   index = 0;
   x = 0;
   loop for index below length;
      generate_cordic_slice(lfp, x, width, index);
      index++;
      x = x + cordic_slice_width;
   end_loop
end_function
```

The function above arrays the CORDIC slices horizontally. This is shown in Figure 8.33.

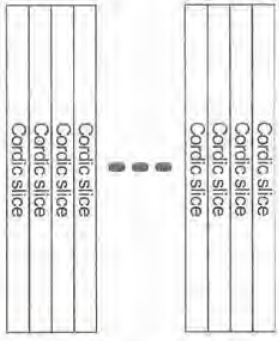


FIG 8.33 CORDIC processor

```
function generate_cordic_slice(lfp, x, width, index);
   y = 0;
   xleft = x;
   generate_cordic_shift(lfp, x, y, width, index);
   generate_cordic_constant(lfp, x, y + 2 * width * adder_height,
      width,index);
   x = x + shift_width;
   generate_cordic_adder(lfp, x, y, width,0); /* x slice */
   y = y + adder_height * width;
   generate_cordic_adder(lfp, x, y, width,0); /* y slice */
   y = y + adder_height * width;
   generate_cordic_adder(lfp, x, y, width,1); /* z slice */
   y = y + adder_height * width;
   generate_cordic_control(lfp, xleft, y); /* control on top */
end_function
```

The `generate_cordic_slice` function adds a shifter for the X and Y paths (`generate_cordic_shift`) and a constant for the Z path (`generate_cordic_constant`) and then builds the X and Y adder/subtracter and the Z adder vertically (`generate_cordic_adder`). Each is `width` bits wide. The control block resides on top of the three datapaths. This is shown in Figure 8.34. The shifter block would consist of wiring that would perform an n-bit shift and XY swizzle according to the horizontal index. Similarly, the constant block would generate a constant dependent on the horizontal index.

The next function generates the stage adders. This is shown in Figure 8.35.

```
function generate_cordic_adder(lfp, x, y, width, flag);
   index = 0;
   y = 0;
   loop for yindex below width
      generate_xyz_bit(lfp, x, y, flag);
      yindex++;
      y = y + adder_bit_height;
   end_loop
end_function
```

The X and Y adder bits have a multiplexer, XOR, ripple carry adder, and flip-flop, while the Z bits have a blank (null) cell, an adder, and a flip-flop. These bits are shown in Figure 8.36(a) and 8.36(b) respectively.

```
function generate_xyz_bit(lfp, x, y, flag);
   if (flag) place_null_cell(lfp, x, y);
   else place_xor(lfp, x, y);
   x = x + xor_width;
   place_adder(lfp, x, y);
   x = x + adder_width;
   place_flipflop(lfp, x, y);
end_function
```

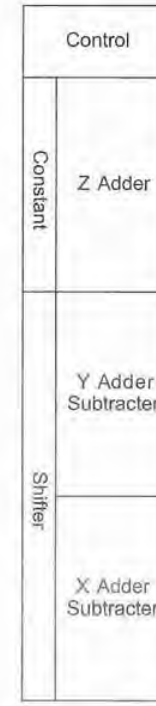


FIG 8.34 CORDIC processor slice

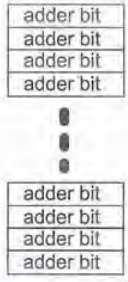
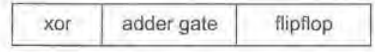
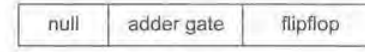


FIG 8.35 CORDIC XYZ adders



(a)



(b)

FIG 8.36 CORDIC XYZ adder bits

In the previous examples, the x and y locations for cell placement have been calculated using "magic" global dimensions (i.e., `adder_width` in the last code example). It is quite possible to build an automated tiling engine that just takes alternating ordered lists of cells and abuts them in the x or y direction, incrementing the origin of the cell to be placed by the dimensions of the previous cell. In this way, no global numbers need to be used in the code generation and the code is generic for a wide range of processes. The other detail omitted from the preceding examples is the generation of signal pins. These are usually needed eventually for layout routers to connect to the generated modules and for comparison with structural views.

8.3.6 Platform-based Design—System on a Chip

As systems have become more complex, the use of predefined intellectual property (IP) blocks has become commonplace. Designs frequently use a number of common blocks such as RISC processors, memory, and I/O functions attached to common busses. A *platform* can be used to implement a design by using common structures such as busses and common high-level languages (such as C) to program the processors. To a large extent, the RISC processor and memories can be interchanged and the number and type of peripherals can be changed while maintaining good design and verification times because the modules have been predesigned and the test and verification scripts come with the IP blocks. The design task is to put the blocks together, design any application-specific blocks, and place and route a correctly operational chip. Note that the last step, while automated, still takes considerable engineering effort.

As many current chips feature one or more embedded microprocessors, the task of writing software is added to the task of designing logic. Moreover, platform-based design poses the problem of partitioning the complete solution between hardware (HDL, gates) and software (programmed on the processor/s). This tends to remain a somewhat manual task, but is increasingly automated by CAD tools.

Platform-based systems typically consist of a basic RISC processor, which can be extended with multipliers, floating point units, or specialized DSP units. In addition (e.g., in Tensilica's Xtensa system), by profiling the executable code, special hardware can be added that corresponds to hardware-assisted instructions, which are introduced into the instruction set. In theory, additional hardware or extra processors can deal with a wide range of computational loads.

Manual techniques for hardware-software codesign mirror this approach. That is, the design begins with a software simulation (ideally on the embedded processor). Timing estimates are gathered, and manual decisions about what to commit to hardware are made. Special simulators to deal with embedded processors and logic have been developed.

With platform-based design, we have in essence come full circle from the first design method suggested: programming a microprocessor. This is the reason processor selection is important when starting out on a product design that may eventually be integrated. As the software effort will often exceed the hardware effort, you don't want to repeat that effort.

8.3.7 Summary

In this section, we have summarized a wide range of CMOS design options ranging from a software-based microprocessor to full custom design. The following table summarizes these options in terms of a variety of criteria. Each category is ranked in relation to each design method from low to high.

Table 8.4 Comparison of CMOS design methods

Design Method	Non-recurring Engineering	Unit Cost	Power Dissipation	Complexity of Implementation	Time to Market	Performance	Flexibility
Microprocessor/ DSP	low	medium	high	low	low	low	high
PLA	low	medium	medium	low	low	medium	low
FPGA	low	high	medium	medium	low	medium	high
Gate Array/SOG	medium	medium	low	medium	medium	medium	medium
Cell Based	high	low	low	high	high	high	low
Custom Design	high	low	low	high	high	very high	low
Platform Based	high	low	low	high	high	high	medium

In the next section, we review the design flows necessary to build a complete chip. While decisions can be difficult, the following diagram (Figure 8.37) should offer some assistance.

The diagram (Figure 8.37) indicates that the most cost-effective approach should be taken to hardware (or software) design given speed, power, and cost targets (occasionally, size will count as well). You should always use an off-the-shelf solution if system constraints are met, because the non-recurring engineering (NRE) costs are amortized over many units. The next most likely prospect is an FPGA design, especially for low-volume (100,000's) applications. Power and cost are the most likely attributes to be challenged in medium- to high-volume applications, and this is where standard cell designs will be used. Mixed-signal, RF, and high-speed digital designs require a cell-based or custom approach.

Increasingly, the NRE cost (predominantly mask cost) is approaching a level where even industry prototypes must be done using multi-project chips, amortizing the mask cost over multiple designs on the same reticle. Designs must be as re-programmable or adaptable as possible.



FIG 8.37 Design decision tradeoffs

8.4 Design Flows

A design flow is a set of procedures that allows designers to progress from a specification for a chip to the final chip implementation in an error-free way. In the previous section, we discussed the basic CMOS design methods without mentioning how we actually design an FPGA, gate array, or cell-based system. In this section, we will summarize the main design flows in use today.

A general design flow is shown in Figure 8.38. Design starts at the behavioral level and then proceeds to the structural level (gates and registers). This step is called behavioral or *Register Transfer Level* (RTL) synthesis because the designs are captured at the RTL (memory elements and logic) level in an HDL. The description is then transformed to a physical description suitable for chip fabrication. This step is called *physical synthesis* (or *layout generation*). Normally, the synthesis steps are automated, albeit guided by human judgment. The verification steps are also shown.

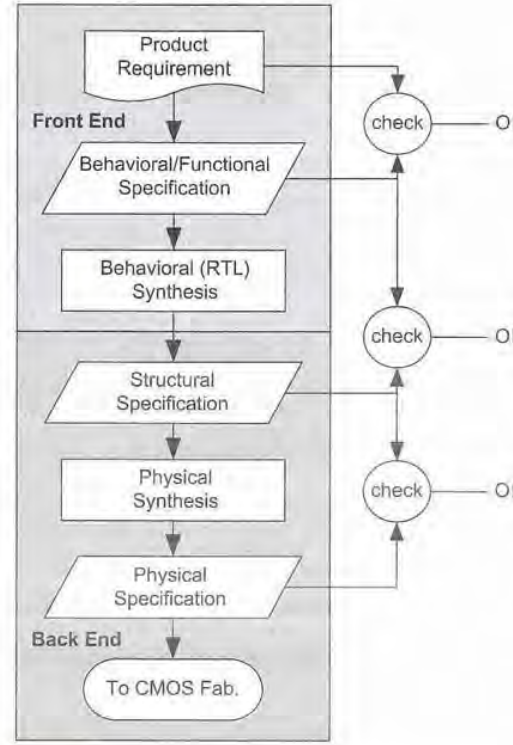


FIG 8.38 Generalized design flow

In Figure 8.38, the design has been partitioned into the *front end* stage at the behavioral level and the *back end* at the structural and physical levels. This is important because it illustrates a partitioning that is used to build *Application Specific Integrated Circuits* (ASICs). In an ASIC, the design can be developed at the HDL level and then passed to a company that completes the transition to an actual chip. In this way, the original design company does not have to invest the personnel or tools required to translate an HDL specification into a physical chip. Theoretically, in an ASIC flow, only a behavioral HDL needs to be designed and simulated (at the behavioral level). All subsequent operations can be completed by a third-party design service with only the final timing having to be verified by the back-end process. This is sometimes referred to as a "throw it over the wall" approach. While it works for moderately complex designs, the interaction between logic and layout is so important in more demanding circuits that such a flow becomes a schedule risk. Primarily, this occurs because the iteration time between logic design and physical placement takes too long when spread over two organizations. Multiple iterations are necessary because the prelayout timing estimates available to the HDL designer correlate poorly with the true postlayout timing because wire lengths are unpredictable before layout. Consider the case where the design cycle from logic to layout takes two hours when completed as an integrated task or one week if split into front-end and back-end tasks, as shown in the figure. If there are 100 iterations for the design, the integrated approach takes roughly 25 working days or five weeks, while the split approach takes two years (without vacations!). Having said this, companies are in business to make this approach work. If there are only ten iterations, the times are much more reasonable.

The next two sections summarize each of the tools required to perform the automatic transformation. We also will examine the verification tools required to guarantee the correctness of the transformation and look at specific design flows. Then, we will describe a

manual flow that is typical of a mixed-signal or RF design. Finally, we will outline a method of transforming directly from the behavioral to the physical level.

8.4.1 Behavioral Synthesis Design Flow (ASIC Design Flow)

At the behavioral level, the operation of the system is captured without having to specify the implementation. This level provides the most independence from implementation details and is the most dependent on the tool flow for a good design.

The most popular style of tools for behavioral synthesis are those that directly transform a behavioral RTL description to a structural gate-level netlist. A typical behavioral flow for an ASIC is shown in Figure 8.39. Tool suppliers include Synopsys, Cadence Design Systems, Mentor Graphics, and Synplicity.

8.4.1.1 Logic Design and Verification The design starts with a specification, which might be a text description or a description in a system specification language. The designer(s) convert this to an RTL behavioral description in an HDL such as Verilog, or VHDL. A set of test benches are then constructed and the HDL is simulated to verify the correct behavior as defined by the specification and product requirements. Typical interactive design environments and simulators include NC-Verilog/SystemC/VHDL or Desk-

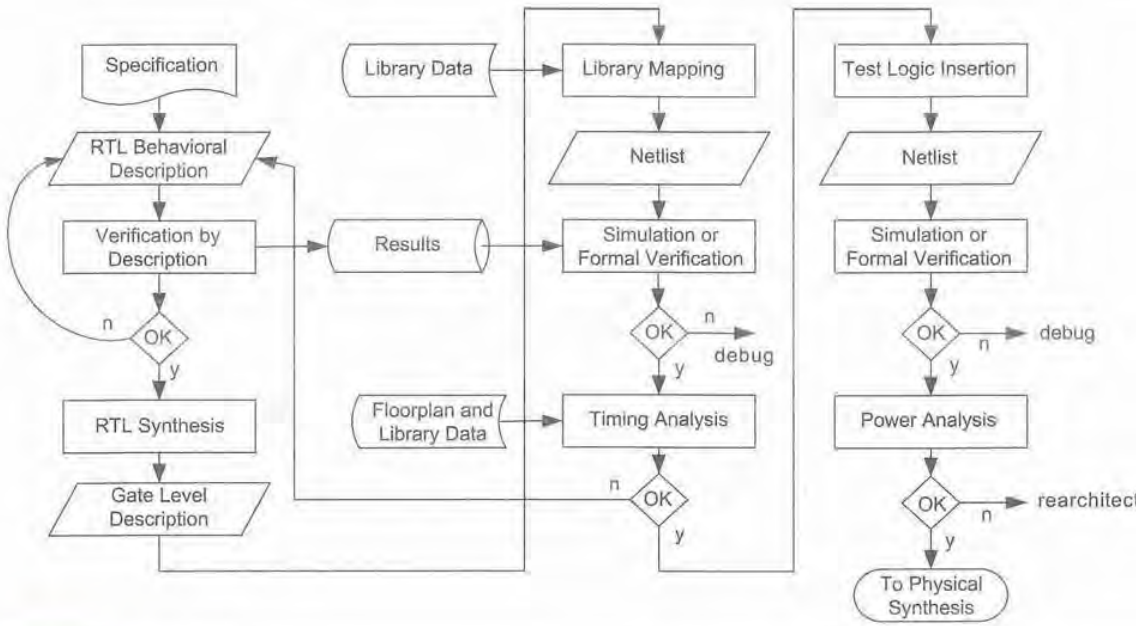


FIG 8.39 RTL synthesis flow

top Verilog/VHDL from Cadence Design Systems, VCS from Synopsys, ModelSim from Mentor Graphics and ActiveHDL from Aldec. Bear in mind that functional verification via simulation is usually carried out hierarchically. That is, after the overall architecture is defined, modules are successively built from the bottom up, verifying at each step. The design is iterated at this level until the correct behavior is evident. Test benches are covered further in Section 9.3.

Behavioral Verilog for an 8-bit implementation of the NCO previously introduced is presented below.

```
module nco #(parameter size = 8,
             counter_size = 16,
             table_size = 64)
            (input fclock, reset,
             input [counter_size-1:0] initial_phase, phase_increment,
             output [size-1:0] q);

   reg  [counter_size-1:0] phase;
   wire [size-3:0]         phase_part, inverted_adr, ROM_adr;
   wire [size-2:0]         ROM_data;
   wire [size-1:0]         wave_out;

   // numerically controlled oscillator
   // note that some constants are hardwired in the code below

   // phase counter
   always @(posedge fclock)
      if (reset) phase <= initial_phase;
      else phase <= phase + phase_increment;

   // add offset and determine ROM address
   assign phase_part = phase[counter_size-3:counter_size-8];
   assign inverted_adr = 7'3f - phase_part;
   assign ROM_adr = phase[counter_size-2] ? inverted_adr : phase_part;

   // look up data in ROM and negate if appropriate
   quarter_wave sine_table(ROM_adr, ROM_data);
   assign wave_out = phase[counter_size-1] ? -ROM_data : ROM_data;
   assign q = wave_out + 8'h80 + phase[counter_size-1];
endmodule
```

8.4.1.2 RTL Synthesis The next step is to *synthesize* the behavioral description. This involves converting the RTL to generic gates and registers, then optimizing the logic to improve speed and area. Other steps involved at this stage are state machine decomposition, datapath optimization, and power optimization. Typical products include Design Compiler from Synopsys, BuildGates from Cadence, and Synplify Pro from Synplicity.

The behavioral code is synthesized into the following structural code using a generic gate library.

```
module nco_struct(input          fclock, reset,
                  input  [15:0]  initial_phase, phase_increment,
                  output [7:0] q);

   wire [6:0] ROM_data;
   wire [15:0] phase_0;
   wire [5:0] nbus_5;
   wire [7:0] wave_out;
   wire [5:0] nbus_2;
   wire [15:0] nbus_1;
   wire [15:0] phase;

   xor i_7(wave_out[7], phase[15], 1'b0);
   xor i_6(wave_out[6], phase[15], ROM_data[6]);
   .
   .
   .
   xor i_2(wave_out[2], phase[15], ROM_data[2]);
   .
   .
   not i_10(nbus_5[2], phase[10]);
   not i_9(nbus_5[1], phase[9]);
   not i_8(nbus_5[0], phase[8]);
   generic_flip_flop phase_reg_15(.q(phase[15]), .d(phase_0[15]),
      .clk(fclock));
   .
   .
   .
   .
   quarter_wave sine_table(.addr(the_address), .q(ROM_Table));
   .
endmodule
```

The mapping to simple Verilog logic statements can be seen at the start of the program. Flip-flops are mapped to generic flip-flops that would implement a variety of flip-flop styles. The hierarchy is maintained, as illustrated by the call to the sine ROM.

8.4.1.3 Library Mapping *Library mapping* takes a generic HDL gate-level description and translates it to a netlist that specifies particular gates in the target library. This stage also maps predefined blocks such as memories to their appropriate descriptions. The following description is a portion of the mapped generic Verilog for the NCO shown above.

```
module nco_struct_mapped(input          fclock, reset,
                         input  [15:0] initial_phase, phase_increment,
                         output [7:0]  q);
   .
   .
   BUFX4 i_506(.A(n_355), .Y(q[7]));
```

```
    .
    MX2X1 i_00(.S0(reset), .B(initial_phase[15]), .A(nbus_1[15]),
       .Y(phase_0[15]));
    NAND2BX1 i_8(.AN(n_102), .B(n_101), .Y(n_104));
    XOR2X1 i_6(.A(phase[15]), .B(ROM_Table[6] ), .Y(n_103));
    .
    .
    DFFHQX1 phase_reg_0(.D(phase_0[15]), .CK(fclock), .Q(phase[15]));
    .
    .
    .
endmodule
```

Now the specific library elements are called out. XOR2X1 is a 2-input normal-power XOR gate and DFFHQX1 is a high-speed normal-power D flip-flop in the Artisan library used in the mapping.

8.4.1.4 Functional or Formal Verification We must now prove that the structural netlist performs the same function as the original behavioral HDL. Ideally, the netlist would be correct-by-construction, but ambiguities in HDLs sometimes cause the synthesizer to produce incorrect netlists from poorly written behavioral code. One verification strategy is to rerun the logic test benches and check that they produce exactly the same output for the behavioral and structural descriptions.

Another strategy is to use a *formal verification* program that compares the logical equivalence of the two descriptions. Formal verification tools are still maturing, but offer the advantage that they mathematically prove both descriptions have exactly the same Boolean functions [Anastasakis02]. In contrast, simulation only is as good as the choice of test vectors. Formality from Synopsys and Incisive Conformal from Cadence are examples of formal verifiers.

Other types of verification that can be run are semantic and structural checks on the HDL. An example of a semantic check would be ensuring that all bus assignments match in bit width, while an example of a structural check would include making sure all outputs are connected.

8.4.1.5 Static Timing Analysis At this point, the functional equivalence of the gate-level description and the original behavioral description has been established. Now the temporal requirements of the design have to be checked. For example, the adder may add, but does it add fast enough? At the behavioral level, clock cycle time is an abstract notion, but at the structural level, an actual cycle time has to be met by a particular set of gates. A *timing analyzer* is used to verify the timing.

The timing analyzer is a critical analytical tool in the arsenal of the modern CMOS digital designer. Timing can be verified in a cursory manner using a timing simulator, i.e., a simulator in which the actual gate timings are used rather than a cycle-based or unit delay simulator. While useful, this approach is usually neither complete nor rigorous and can take an extraordinary amount of time to run.

Static timing analysis, on the other hand, runs quickly and exhaustively evaluates *all* timing paths. The inputs to the timing analyzer at this point are derived from the basic timing of the library gates due to intrinsic gate delays and routing loads that can be either estimated statistically or derived from floorplanning data. (See Section 8.4.2.2 for a description of floorplanning.) Timing analyzers check for both *max-delay* (will all flip-flops meet their setup time at the required cycle time?) and *min-delay* (will any flip-flop violate its hold time?).

Static timing analysis can suffer from *false path* problems. Typical of this problem might be a reset line in a circuit that has many clock cycles to operate. The timing analyzer might report that it cannot complete in one cycle. The designer must manually flag such multicycle paths for the timing analyzer.

Typical timing analyzers include Pearl from Cadence and Pathmill and PrimeTime from Synopsys. The listing on the following page is a portion of a typical report for the NCO from a timing analyzer (this listing is from the Cadence BuildGates tool suite). The initial information specifies the required cycle time (11 ns) and then displays the calculated arrival time (5.89 ns). This results in what is termed a *slack time* of 5.11 ns. A table is then produced that lists the mapped Verilog instance, the signals that are changing and which direction the signals are toggling (by ^ and v characters for rising and falling), the cell type, the delay through that instance, and the arrival time. The required arrival time is also listed. Each line is examined for excessive timing and then normally the original Verilog is changed to improve delays through better pipelining or logic design. The synthesizer can also be directed to improve the timing. The sine table has been synthesized onto gates in this implementation.

8.4.1.6 Test Insertion Logic and registers are then inserted/modified to aid in manufacturing tests (see Section 9.6). Two basic techniques are used. One involves inserting scannable registers so that the state of a circuit can be set and monitored. Accompanying this option is a technique called *Automatic Test Pattern Generation* (ATPG), which is used to generate tests for a scannable design. The other technique, called *Built-In Self-Test* (BIST), modifies registers to allow *in situ* testing within the chip. Figure 8.40 shows the NCO after a test insertion program has run. The difference between the original design shown in Figure 8.13 and this design is the fact that blue *scan registers* have been inserted.

Typical commercially available test programs include DFT Compiler from Synopsys for scan insertion and Tetramax for ATPG. LogicVision markets Logic BIST and Memory BIST for built-in self-test.

8.4.1.7 Power Analysis The power consumption of the circuit is then estimated. Power consumption depends on the activity factors of the gates, which in turn depends on the inputs the chip receives. *Power analysis* can be performed for a particular set of test vectors by running a simulator and evaluating the total capacitance switched at each clock transition at each node. At this stage, if the power is too high, the design must return to the architectural level to rethink the solution. Commercial power analysis tools include PrimePower and Powermill from Synopsys.

```
Beginpoint: phase_reg_14/Q (^) triggered by leading edge of 'clock'
Required Time 11.00
- Arrival Time 5.89
= Slack Time 5.11
   Clock Rise Edge 0.00
   + Source Insertion Delay 0.20
   = Beginpoint Arrival Time 0.20
```

Instance	Arc	Cell	Delay	Arrival Time	Required Time
	fclock ^			0.20	5.31
phase_reg_14	CK ^ -> Q ^	DFFHQX1	0.58	0.78	5.89
i_34	A ^ -> Y ^	XOR2X1	0.78	1.57	6.68
sine_table	addr[3] ^	quarter_wave		1.57	6.68
sine_table/i_371	A ^ -> Y v	INVX1	0.45	2.02	7.13
sine_table/i_42	B v -> Y ^	NOR2X1	0.19	2.21	7.32
sine_table/i_44	AN ^ -> Y ^	NAND2BX1	0.16	2.37	7.48
sine_table/i_145	A0 ^ -> Y v	AOI211X1	0.10	2.47	7.58
sine_table/i_6	C0 v -> Y ^	OAI211X1	0.13	2.61	7.72
sine_table/i_484	A ^ -> Y ^	BUFX4	0.96	3.57	8.68
sine_table	q[0] ^	quarter_wave		3.57	8.68
i_120	B ^ -> Y v	NOR2BX1	0.22	3.79	8.90
i_7	AN v -> Y v	NOR2BX1	0.20	3.99	9.10
i_9	AN v -> Y v	NOR2BX1	0.21	4.20	9.31
i_20	B v -> Y ^	NAND2BX1	0.12	4.32	9.43
i_3	AN ^ -> Y ^	NAND2BX1	0.17	4.49	9.60
i_8	AN ^ -> Y ^	NAND2BX1	0.19	4.67	9.78
i_31	A ^ -> Y ^	XNOR2X1	0.28	4.96	10.07
i_504	A ^ -> Y ^	BUFX4	0.93	5.89	11.00
	q[6] ^		0.00	5.89	11.00

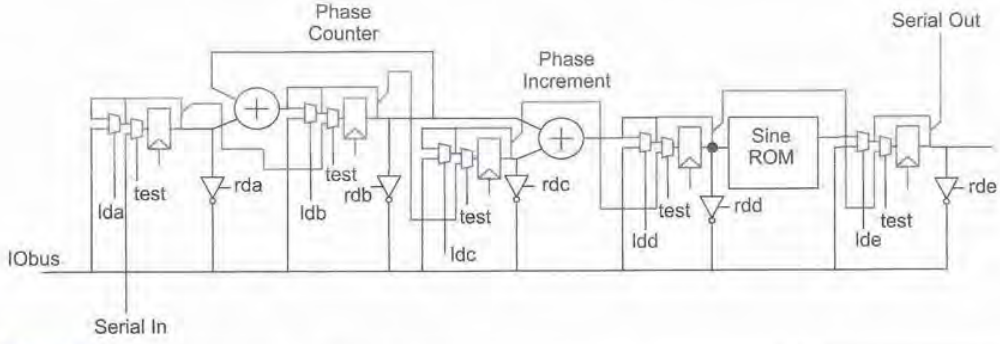


FIG 8.40 Scan register insertion for testing

Example

While automated tools are the solution to estimating power consumption in large circuits, sometimes you need a rapid, "back of the envelope" method of estimating power. How can we achieve this?

Solution: The following technique will give you a rough idea of the power dissipated in a circuit.

Step 1: Estimate the power in switched flip-flops and registers. Sum the clock input capacitance of all registers and, if possible, the internal switched capacitance. The latter is often hard to ascertain unless you have access to the internals of the cell library, but as a rough approximation, double the specified clock input capacitance. To this, add the capacitance of the clock routing. Again, conservatively, double the total clock load that you have previously calculated. The clock has an activity factor of 1. Now, given V_{DD} and the clock frequency, you can estimate the clock power using EQ (4.30). Thus, the power will be $2 \cdot 2 \cdot C_{clk} \cdot V_{DD}^2 \cdot f = 4C_{clk}V_{DD}^2 f$.

Step 2: Estimate the power in the combinational circuits. Sum all input and output capacitances for all gates. This approximates the internal capacitance of the logic gates. To this we must add the routing capacitance. This is best done after some experience with the process and design tools you are using, but as a guide, use a number between 2 and 10. The activity factor has to be estimated. Again experience helps here, but 0.5 is very conservative. Thus, the total logic power will be roughly $5 \cdot 0.5 \cdot C_{logic} \cdot V_{DD}^2 \cdot f = 2.5 C_{logic} V_{DD}^2 f$.

These two steps are very approximate. After a bit of experience with a technology and design flow, you will be able to put your own numbers on the approximations.

8.4.1.8 Summary Apart from increasing design productivity, logic synthesis systems are useful for transforming between technologies. For instance, you might synthesize behavioral HDL onto multiple FPGAs and construct a prototype used to verify the operation of the circuit under real-world conditions. Then you can compile a single-chip version from the same HDL using a gate-array library.

8.4.2 Automated Layout Generation

Layout generation is the last step in the process of turning a design into a manufacturable database. It transforms a design from the structural to the physical domain. This step is sometimes called *physical synthesis* when the structural netlist is manipulated as the physical layout is generated. There are two main methods of automated layout generation, namely, standard cell place and route and the use of a software generator. Generators were discussed in Section 8.3.5, while this section emphasizes place and route tools used on most ASICs.

Figure 8.41 shows a standard place and route layout generation design flow. It begins with the structural netlist describing gates, flip-flops, and their interconnections. The

netlist might be provided in the *Design Exchange Format* (DEF) described in Section 8.9.4 or as a Verilog netlist like the one in Section 8.4.1.3. The placement tool also takes a standard cell library definition describing cell dimensions and port locations. Typically, this is in the *Library Exchange Format* (LEF), described in Section 8.11.3.

8.4.2.1 Placement The first step in Figure 8.41 is to place the standard cells. The key to automation of standard cell layouts is the use of constant-height, variable-width standard cells that are arrayed in rows across a chip, as shown in Figure 8.42. In contrast to SOG and gate array chips, standard cell chips can add application-specific custom blocks such as memories and analog blocks by allowing the standard cell rows to "flow" around the fixed-shape custom blocks. No separation has been shown between standard cell rows because routing takes place over the cells using multiple layers of metal. In older processes with two or three metal layers, a space between rows would be needed to allow routing. LEF summarizes the salient physical details of cells.

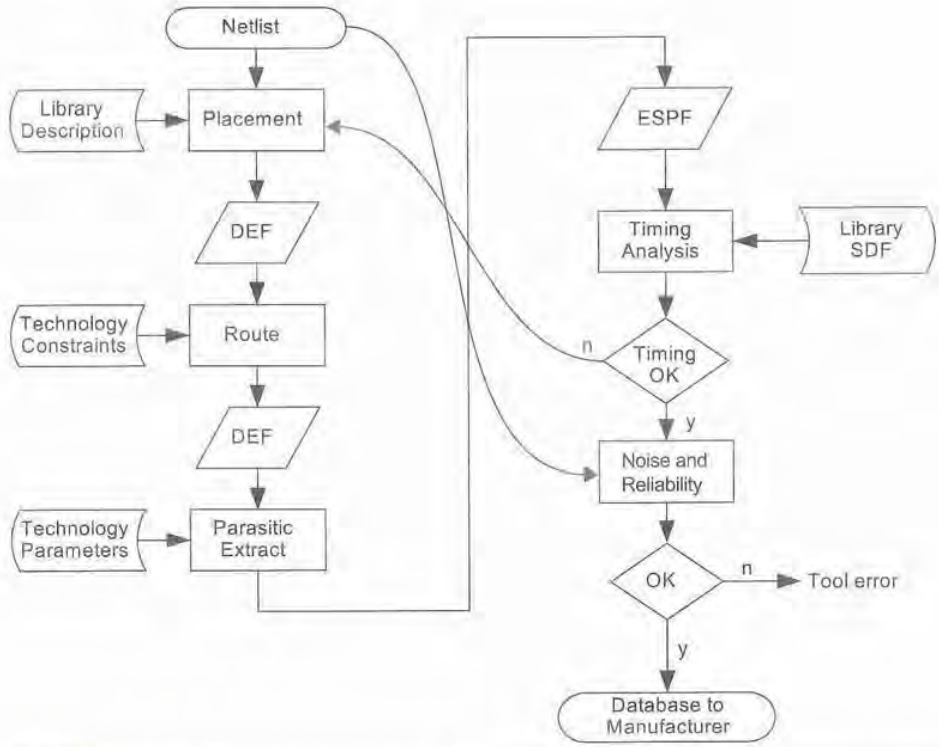


FIG 8.41 Standard cell place and route design flow

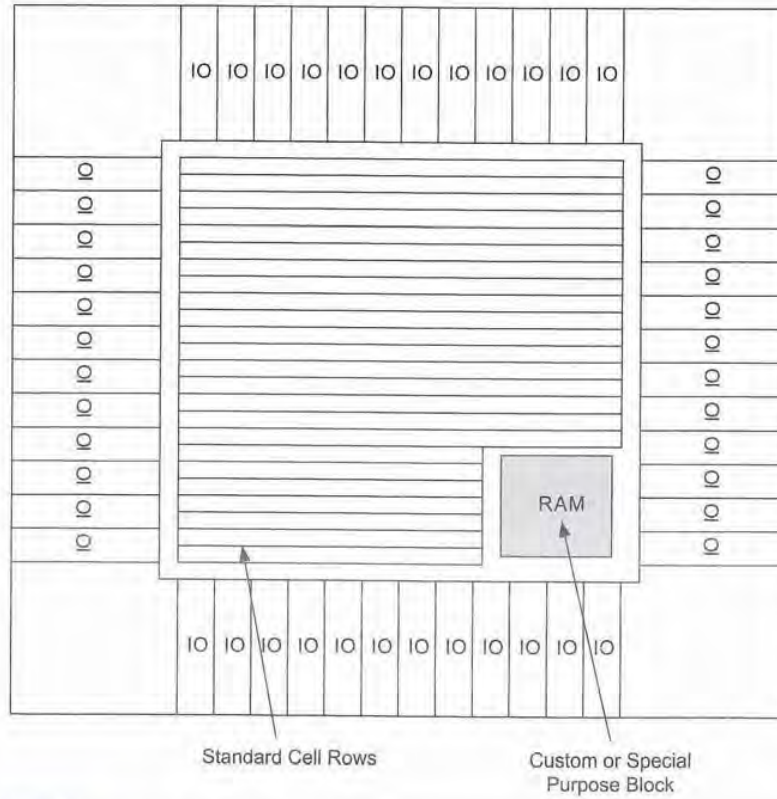


FIG 8.42 Standard cell chip layout

The objective of a simple placement algorithm is to minimize the length of wires. In *timing-driven placement*, the cost of wires is weighted to minimize delay on the critical paths. At the end of the placement phase, the cells have been fixed in position in the overall array. The placed design is saved in a standard format (e.g., DEF) for routing.

8.4.2.2 Floorplanning Increasingly a manual floorplanning step is required in the placement process. Rather than place a design "flat" (i.e., all cells at the same level of the hierarchy), modules are clustered in areas that are dictated by the need to communicate with other modules. Section 8.2.5 illustrated some floorplans for the software radio. This style of floorplanning might be completed prior to automatic placement.

8.4.2.3 Routing After placement of cells, the signal nets in the circuit need to be routed. Routing is normally divided into two steps: *global* routing and *detailed* routing.

A global router abstracts the routing problem to a notional set of abutting channels that cover the chip surface through which wires are routed. Routes are added to channels according to a cost function. Wires can be changed from channel to channel if the density of wires in a channel becomes too high. The detailed router places the actual geometry required to complete signal connections. Over time, a selection of detailed routers have been developed to automatically route signals. Older routers constrained signals to a grid of tracks, but newer *gridless* routers are more flexible for variable pitch wires. Moreover, they allow easy interface to foreign cells that may have I/O pin locations that are not on any specific routing grid. Routers also can route over the top of cells. LEF definitions are used to indicate obstructions on various layers in cell definitions. Advanced routers take into account manufacturability concerns such as redundant vias (more that one via inserted when space is available) and adjustable spacing (to separate wires and reduce coupling when there is room).

In the example of the flow shown in Figure 8.41, the router uses a technology file to specify routing layers and pitches for the process technology. It writes the results to another DEF file.

8.4.2.4 Parasitic Extraction The placed and routed design is then passed to the circuit *parasitic extractor*. In the example shown in Figure 8.42, the placed and routed design is provided to the extractor in DEF format and the output is an *Extended Standard Parasitic Format* (ESPF), *Reduced Standard Parasitic Format* (RSPF), or *Standard Parasitic Exchange Format* (SPEF) that describes the R's and C's associated with all nets in the layout. The extractor uses another technology file defining the interlayer capacitances and layer resistances. The formats are covered in more detail in Section 8.11.6.

The capacitance extractor can be a 2D, 2.5D, or 3D extractor. Two-dimensional (2D) extractors look at a cross-section assuming wires extend uniformly outside the section. A 2.5D extractor uses lookup tables to more accurately estimate capacitance near nonuniformities. A 3D extractor solves Maxwell's equations in three dimensions to precisely determine capacitance of complex geometries. 3D extraction used to be prohibitively time-consuming, but new statistical algorithms, such as those in QuickCap from Magma Design, deliver good accuracy with faster runtimes.

8.4.2.5 Timing Analysis Static timing analysis is now rerun with the actual routing loads placed on the gates. This is usually the bottleneck in the design process as the full reality of a physical realization is apparent. Multiple iterations of synthesis and placement & routing are usually necessary to converge on timing requirements.

Additionally, if possible (especially where dynamic circuits are used), a transistor-level timing simulation should be run. While this cannot usually be achieved using a SPICE-based simulator, a variety of transistor level simulators with "almost SPICE accuracy" have been in use since the late 1970s. These currently have the capacity to do whole-chip simulations at the transistor level, but at somewhat reduced transistor modeling accuracy. Path-mill and Nanosim from Synopsys and UltraSim from Cadence are examples of current simulators of this type.

8.4.2.6 Noise, V_{DD} Drop, and Electromigration Analysis Analyses are now run to check noise, IR drop in supply lines, and electromigration limits. Noise analysis is run to evaluate crosstalk due to interlayer routing capacitance. For static circuits, aggressor nodes can alter the timing of victim nodes but not their function, as described in Section 4.5.4. For poorly designed dynamic circuits, crosstalk can cause irrecoverable errors. SignalStorm, ElectronStorm, and VoltageStorm from Cadence are examples of such tools.

8.4.2.7 Timing-driven Placement The trouble with a place-then-route strategy is that after the layout is completed, the parasitic routing capacitance is extracted and the timing analysis is done to estimate timing. The timing is not known until the physical layout is complete. If timing problems are found, the cycle has to be repeated with some kind of constraint placed on the problematic paths. With complex designs this quickly gets out of control, to the point where changing something on one iteration could undo something fixed on a previous iteration. There are stories of designs that never were completed because of this problem.

The solution is to use a technique called *timing-driven placement*, which takes into account the timing (speed) of the circuit as cells are placed. Cells on critical paths are given priority to minimize wire delay. This approach, illustrated in Figure 8.43, has been successful and often results in a one-pass approach for many designs.

8.4.2.8 Clock-tree Routing Central to modern high-speed designs is the clock distribution strategy. In Section 12.5.4, a number of these approaches are explained. To minimize skew, it is often best to route the clock and its buffers before the main logic placement and routing is completed. This task is performed with a *clock tree router*. Any of the clocking approaches described in Chapter 7 can be implemented in this manner.

8.4.2.9 Power Analysis Power estimation can be repeated for the extracted design now that real wire capacitances are available. Similar techniques to those used during RTL synthesis are used.

8.4.3 Mixed-signal or Custom-design Flow

In the previous section, we described a flow that would be used for a purely digital chip in which the procedure for converting from HDL to layout is highly automated. This flow offers high productivity for most large digital chips with moderate performance requirements. But what of smaller analog, RF, and high-speed digital sections of a chip? For these sections we use a custom-design flow, which is shown in Figure 8.44.

The designer begins by drawing a schematic (or possibly writing a netlist). An electrical rule check (ERC) verifies port connectivity and checks for unconnected inputs or outputs—the kind of simple connectivity errors that can occur easily in a manually drawn schematic. When the schematic is deemed correct, circuit simulation is then carried out using a SPICE-type simulator to verify DC, AC, transient, noise, and/or RF performance.

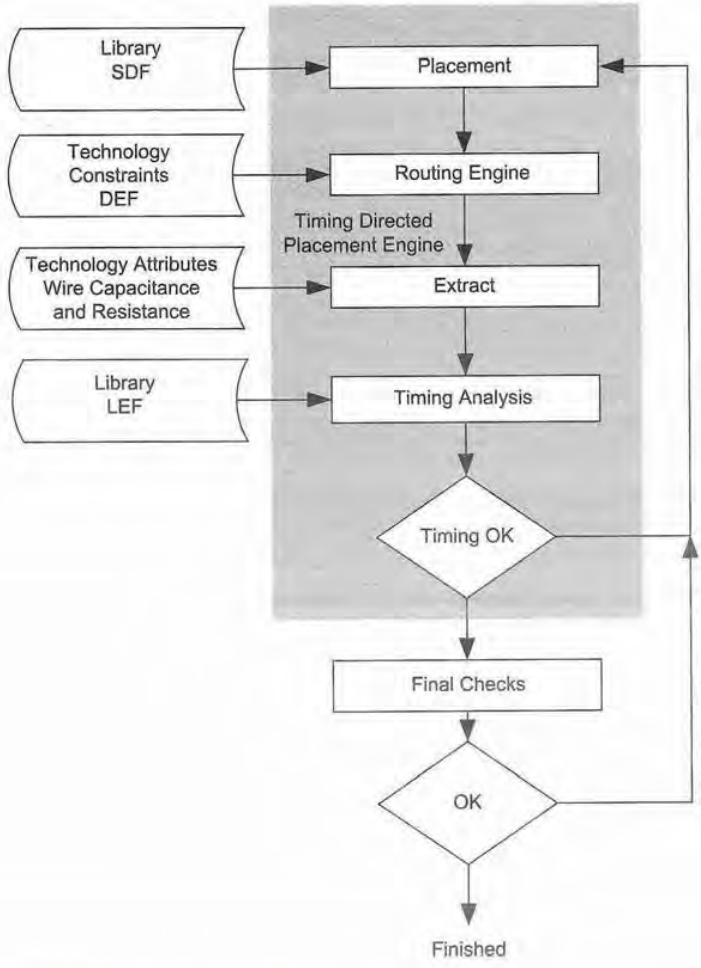


FIG 8.43 Timing directed placement design flow

Once the circuit behavior of the module has been verified, the layout can commence, starting with the floorplan. Floorplanning can be an iterative process that is refined as actual module sizes and critical paths become known. Custom layout is a very time-consuming task; for example, a large microprocessor can keep a hundred mask design technicians busy for two years. Automating noncritical parts of the layout is essential for productivity. When the layout for the module is complete, a layout circuit

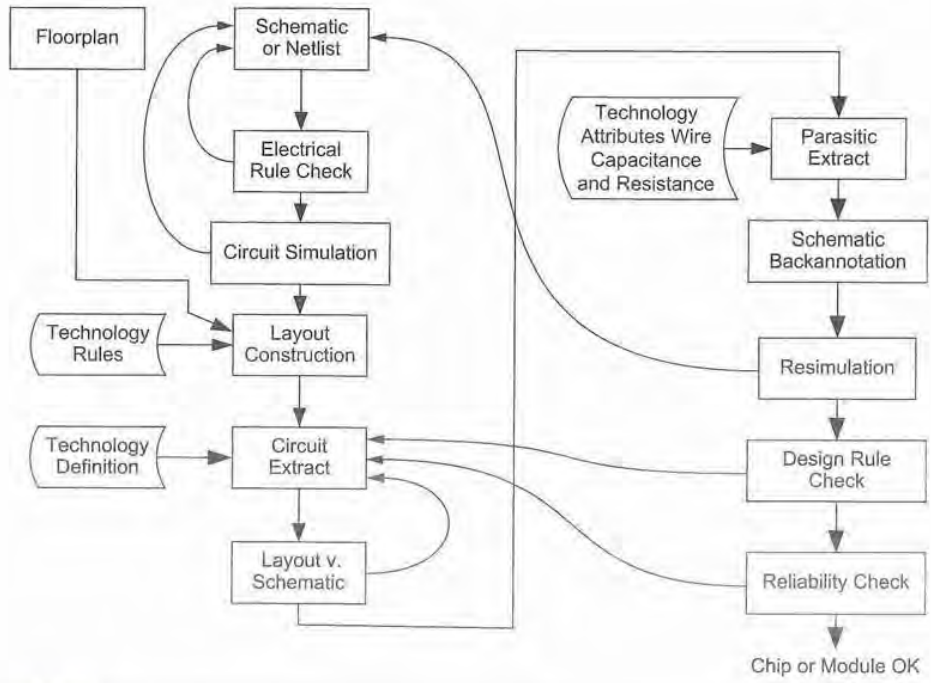


FIG 8.44 Mixed-signal or custom-design flow

extractor is invoked to determine the connectivity of primitives (MOS and bipolar transistors, diodes, resistors, capacitors, inductors) in the layout using rules like those illustrated in Section 3.5.2.

In the next step, the extracted netlist is compared to the schematic using a graph isomorphism program to determine whether the two netlists are identical in connectivity. This proceeds by assigning primitives to the nodes of a graph and the connections to the arcs in the graph. Graph coloring based on the connectivity and circuit parameters (i.e., transistor type, width, and length) determines the extent of the match. Once connectivity equivalence has been determined, each primitive attribute is checked for equivalence (i.e., capacitor or resistor value, transistor W/L). Discrepancies are reported to the user. Graphical feedback may be provided to help the designer find the source of any mismatch. This step is commonly called *layout versus schematic* (LVS).

Once the structural-to-physical equivalence has been established, the parasitic extract is completed. This adds the parasitic routing capacitance and resistance to the original primitive elements. In general, inductors are not extracted, but are dealt with by cookie cutting the inductor out of the layout and substituting a previously generated physical model. This is sometimes called *macro substitution*. The parasitic capacitance and resis-

tance can be *back annotated* onto the schematic and the complete circuit resimulated. It must be pointed out that this step is extremely important. Matching simulated behavior to real device behavior is of critical importance in being able to accurately predict performance. It is too late when the circuit has been built!

The module layout can then be design-rule checked (DRC). Alternatively, this step can be completed just as the layout is completed. Normally, the AC performance is more important than tweaking the last design-rule error because running DRC on a circuit that does not meet performance goals is a waste of time.

Following this, a set of manufacturability verification steps needs to be completed. These can be manual or automated. In common with the standard cell design flow, power bus widths should be checked to ensure that they comply with metal migration and IR drop constraints. Power consumption can be found directly from circuit simulation. Adequate substrate and well contacts should be present in a bulk CMOS design, and all external I/O must be guard-ringed. At this stage, a check can also be made for substrate noise injection from digital to analog circuits. SubstrateStorm from Cadence performs this task.

This process can be completed hierarchically to build up large modules. Usually, the ultimate limitation comes from trying to simulate vast numbers of transistors accurately in SPICE. A variety of fast transistor-level simulators have been developed to deal with this problem, although there is always some upper limit to what can be simulated at the desired accuracy.

8.4.4 Programmed Behavioral Synthesis

An alternate class of behavioral synthesizer directly synthesizes blocks in a fashion reminiscent of silicon compilers. This type of synthesizer takes a fairly targeted application space (such as digital signal processing) and constructs a logic, circuit, and physical description based on a high-level input description. An example of such a system is the design system used at the Berkeley Wireless Research Center, which takes high-level descriptions defined in Simulink, and automatically translates them through a number of steps to a mask layout. The basis of this system is a set of predefined modules available in the Simulink library that have targets in the physical domain. This type of flow is illustrated in Figure 8.45. The translation between behavioral and physical layers of abstraction occurs directly. RTL and structural descriptions can also be generated as a side product to aid in verification. For example, a multiplier for complex numbers could be specified as a Simulink block. An instance of this multiplier would trigger the automatic construction of the appropriate RTL, gate, and mask-level description following a preprogrammed algorithm.

8.5 Design Economics

It is important for the IC designer to be able to predict the cost and the time to design a particular IC or sets of ICs. This can guide the choice of an implementation strategy. This section will summarize a simplified approach to estimate these values.

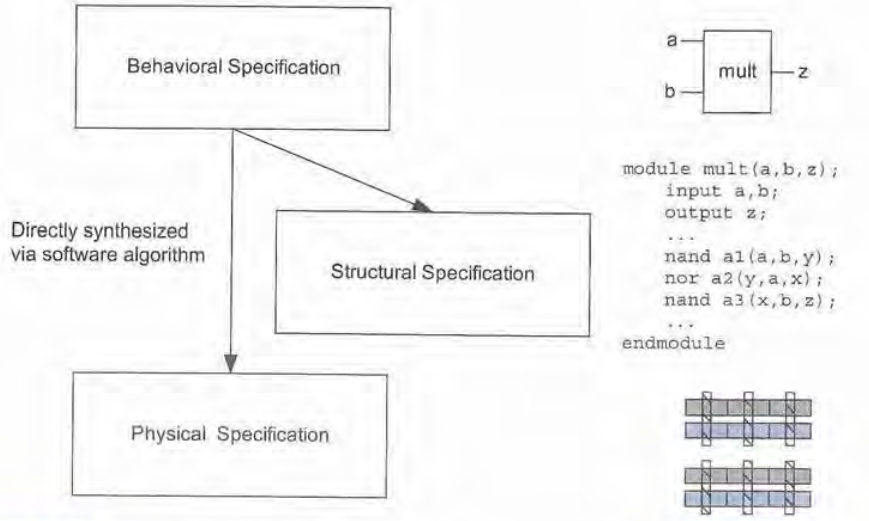


FIG 8.45 Direct behavioral-to-physical translation

In this section, we will concentrate on the cost of a single IC, although you should consider the overall system when making such decisions. System-level issues such as packaging and power dissipation can affect the cost of an IC.

The selling price S_{total} of an integrated circuit may be given by

$$S_{total} = C_{total} / (1-m) \tag{8.3}$$

where

C_{total} is the manufacturing cost of a single IC to the vendor
m is the desired profit margin

The margin has to be selected to ensure a profit after overhead (G&A) and the cost of sales (marketing and sales costs) have been considered.

The costs to produce an integrated circuit are generally divided into the following elements:

- Non-recurring engineering costs (NREs)
- Recurring costs
- Fixed costs

8.5.1 Non-recurring Engineering Costs (NREs)

Non-recurring engineering costs are those that are spent once during the design of an integrated circuit. They include

- engineering design cost E_{total}
- prototype manufacturing cost P_{total}

These costs are amortized over the total number of ICs sold. F_{total}, the total non-recurring cost, is given by

$$F_{\text{total}} = E_{\text{total}} + P_{\text{total}} \tag{8.4}$$

The NRE costs can be amortized over the lifetime volume of the chips. Alternatively, the non-recurring costs can be viewed as an investment for which there is a required rate of return. For instance, if \$1M is invested in NRE for a chip, then \$10M has to be generated for a rate of return of 10.

8.5.1.1 Engineering Costs The cost of designing the IC E_{total} hopefully will happen once during the chip design process. The costs include:

- personnel cost
- support costs

The personnel costs might include the labor for

- architectural design
- logic capture
- simulation for functionality
- layout of modules and chip
- timing verification
- DRC and tapeout procedures
- test generation

The support costs amortized over the life of the equipment and the length of the design project include

- computer costs
- CAD software costs
- education or re-education costs

Costs can be drastically reduced by reusing modules or acquiring fully completed modules from an intellectual property vendor. As a guide the per annum costs might break down as follows (these figures are in US dollars for engineers in the USA circa 2004):

Salary	\$50–\$100K
Overhead	\$10–\$30K
Computer	\$10K
CAD Tools (digital front end)	\$10K
CAD Tools (analog)	\$100K
CAD Tools (digital back end)	\$1M

The cost of the back-end tools clearly must be shared over the group designing the chips.

8.5.1.2 Prototype Manufacturing Costs These costs (P_{total}) are the fixed costs to get the first ICs from the vendor. They include

- the mask cost
- test fixture costs
- package tooling

The photo-mask cost is proportional to the number of steps used in the process. Mask costs increase as the process dimensions are reduced, so although newer, smaller processes generally have increased mask costs, masks on the metallization layers can be less expensive than on the lower layers. A mask currently costs between \$500 and \$30,000 or more. A complete mask set for 130 nm CMOS costs in the vicinity of \$500K to \$1M. These prices will increase as process linewidths are reduced.

A test fixture consists of a printed wiring board probe assembly to probe individual die at the wafer level and interface to a tester. Costs range from \$1000 to \$50,000, depending on the complexity of the interface electronics.

If a custom package is required, it may have to be designed and manufactured (tooled). The time and expense of tooling a package depends on the sophistication of the package. Where possible, standard packages should be used.

An economical way of prototyping chips is to use a multi-project reticle that combines a number of different chip designs onto one mask set. Thus, if there were 200 sites available on a mask set and 20 projects were implemented, each project would get 10 die per wafer and the mask cost per project would be $1/20^{th}$ of the cost of a complete mask set. This kind of service is provided by many of the silicon vendors and also MOSIS. For modest technology this can be quite cheap (~ \$1000 per mm^2 for 0.6 μm). Some commercial users worry about protection of intellectual property when they share a mask set.

Example

You are starting a company to commercialize your brilliant research idea. Estimate the cost to prototype a mixed-signal chip. Assume you have seven digital designers, three analog designers, and five support personnel and that the prototype takes two fabrication runs and two years.

Solution: The seven digital designers will cost 7 • ($70K + $30K + $10K + $10K) = $840K. The three analog designers will cost 3 • ($100K + $30K + $10K + $100K) = $720K. The five support personnel cost 5 • ($40K + $20K + $10K) = $350K. One fabrication run with the back-end tools will cost $2M. Thus, the cost is $3.91M per year with one fab run. The total predicted cost here is nearly $8M.

You may see ways to improve this. Clearly, you can reduce the number of people and the labor cost. You might reduce the CAD tool cost and the fabrication cost by doing multiproject chips. However, the latter approach will not get you to a pre-production version, because issues such as yield and behavior across process variations will not be proved. Figure 8.46 shows the breakdown of the overall cost.

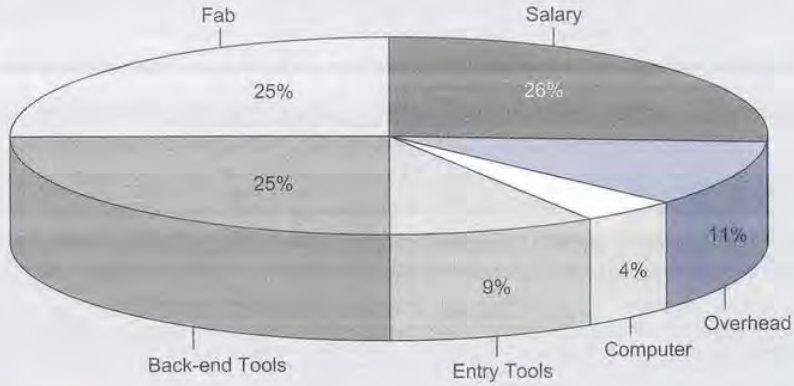


FIG 8.46 Pie chart showing prototyping costs for a mixed-signal IC

8.5.2 Recurring Costs

Once the development cost of an IC has been determined, the IC manufacturer will arrive at a price for the specific IC. A few large companies such as Intel, TI, STMicroelectronics, Toshiba, and IBM have in-house manufacturing divisions. Many fabless semiconductor companies outsource their manufacturing to a silicon foundry such as TSMC, Hitachi/

UMC, IBM, LSI Logic, or ST. This is a recurring cost; that is, it recurs every time an IC is sold. Another component of the recurring cost is the continuing cost to support the part from a technical viewpoint. Finally, there is what is called "the cost of sales," which is the marketing, sales force, and overhead costs associated with selling each IC. In a captive situation such as the IBM microelectronics division selling CPUs to the mainframe division, this might be zero.

The IC manufacturer will determine a part price for an IC based on the cost to produce that IC and a profit margin. The margin generally falls as the volume increases. An expression for the cost to fabricate an IC is as follows:

$$R_{total} = R_{process} + R_{package} + R_{test} \tag{8.5}$$

where

$R_{package}$ = package cost

R_{test} = test cost—the cost to test an IC is usually proportional to the number of vectors and the time to test.

$$R_{process} = W / (N * Y_w * Y_{pa}) \tag{8.6}$$

where

W = wafer cost ($500–$3000 depending on process and wafer size)

N = gross die per wafer (the number of complete die on a wafer)

Y_w = die yield per wafer (should be ~70%–90+% for moderate sized dice in a mature process)

Y_{pa} = packaging yield (should be ~95%–99%)

If a die has area A and is fabricated on a wafer with radius r, the gross number of dice per wafer is

$$N = \pi\left[\frac{r^2}{A} - \frac{2r}{\sqrt{2A}}\right] \tag{8.7}$$

where the second term accounts for wasted area around the edges of a circular wafer.

The packaging yield is the number of chips that pass testing after the wafer has been diced and the parts packaged. The die yield is affected by defects randomly distributed around the wafer. The probability of a random defect causing a particular die to fail depends on the size of the die A and average number of defects per unit area D. If defects obey a Poisson distribution, Y_w may be given by [Seeds67]

$$Y_w = e^{-AD} \tag{8.8}$$

For small dice ($AD << 1$), Y_w is nearly 1 and $R_{process}$ grows linearly with A. For large dice ($AD >> 1$), Y_w drops off rapidly because most chips will have defects and $R_{process}$ grows exponentially with A.

8.5.3 Fixed Costs

Once a chip has been designed and put into manufacture, the cost to support that chip from an engineering viewpoint may have a few sources. *Data sheets* describing the characteristics of the IC have to be written, even for application-specific ICs that are not sold outside the company that developed them. From time to time, *application notes* describing how to use the IC may be needed. In addition, specific application support may have to be provided to help particular users. This is especially true for ASICs, where the designer usually becomes the walking, talking, data sheet and application note. Another ongoing task may be failure or yield analysis if the part is in high volume and you want to increase the yield.

As a side comment, every chip or test chip designed should have accompanying documentation that explains what it is and how to use it. This even applies to chips designed in the academic environment because the time between design submission and fabricated chip can be quite large and can tax even the best memory.

Example

Suppose your startup seeks a return on investment of 5. The wafers cost \$2000 and hold 400 gross die with a yield of 70%. If packaging, test, and fixed costs are negligible, how much do you need to charge per chip to have a 60% profit margin? How many chips do you need to sell to obtain a 5-fold return on your \$8M investment?

Solution: $R_{total} = R_{process} = \$2000/(400 \cdot 0.7) = \$7.14$. For a 60% margin, the chips are sold at $\$7.14 / (1 - 0.6) = \$17.86$ with a profit of \$10.72 per unit. The desired ROI implies a profit of $\$8M \cdot 5 = \$40M$. Thus, $\$40M / \$10.72 = 3.73M$ chips must be sold. Clearly, a large market is necessary to justify the investment in custom chip design.

8.5.4 Schedule

At the outset of a system design project involving newly designed ICs, it is important to estimate the design cost and design time for that system. Estimating the cost can help you determine the method by which the ICs will be designed. Estimating the schedule is essential to be able to select a strategy by which the ICs will be available in the right time and at the right price. This second task is usually the least well specified and requires some experience.

If we assume that fixed costs are kept reasonable and that for a given IC size, $R_{process}$ is constant, the variables left in determining the cost of an IC are E_{total}, the engineering design cost, and P_{total}, the prototype manufacturing cost. P_{total} depends on the way in which the IC is implemented. We examined a variety of strategies for the design of CMOS systems earlier in the chapter. The fixed costs of prototyping P_{total} are relatively constant, given an implementation technology. The engineering costs depend on the complexity of the chip, the design strategy, and the amount of sustaining engineering needed. Usually, the design and verification engineering costs dominate. For this reason, it is important to be able to estimate a schedule for the design of an IC and then manage the available resources to bring the project to a successful conclusion.

Increased engineering effort can reduce the size of the die, which reduces $R_{process}$. Hence, it is important to be able to trade off the reduction in die cost with the increase in engineering effort. Opinions vary, but it is usually best to get a product first to market and then shrink the die when the product becomes successful. Optimizing without market feedback is usually a recipe for loss of market share or even failure to gain any market share at all.

[Paraskevopoulos87] suggests a number of fairly obvious methods for increasing productivity, thereby improving schedules:

- using a high-productivity design method
- improving the productivity of a given technique
- decreasing the complexity of the design task by partitioning

A final caution: Adding people to a project that is already late tends to make it later [Brooks95].

8.5.5 Personpower

To estimate the schedule, you must have some idea of the amount of effort required to complete the design. As we have seen, typical IC projects will involve the following tasks

1. architectural design
2. HDL capture
3. functional verification
4. place and route
5. timing verification, signal integrity, reliability verification
6. DRC and tapeout procedures (ERC, LVS, mask generation)
7. test generation

While some researchers have attempted to derive analytical formulae for productivity, the best predictor of design schedule for a team is previous performance. Design time for a

Example

While it is hard to predict the design and test time for a chip, we can at least identify the main tasks and corresponding fixed periods in a chip design project. A representative Gantt chart is shown in Figure 8.47 for a project running over one year. The logic design time is shown as 12 weeks, which would be appropriate for an extremely simple chip. Double this time would be representative of moderately complex digital chips. The fixed times tend to be the fabrication time and packaging time, which are shown to be 10 weeks in the example. The design, debug, and test times will expand or contract to fit the complexity of the chip. And, if you are meticulous and lucky, you will not have to respin the chip.

ID	Task Name	Start	Finish	Duration
1	Specification	1/1/2004	1/28/2004	4w
2	Digital Design	1/29/2004	4/21/2004	12w
3	Place and Route	4/22/2004	6/16/2004	8w
4	Fabrication	6/17/2004	8/11/2004	8w
5	Packaging	8/12/2004	8/25/2004	2w
6	Lab Test	8/26/2004	10/20/2004	8w
7	Respin	10/21/2004	11/17/2004	4w
8	Lab Test	11/18/2004	12/29/2004	6w

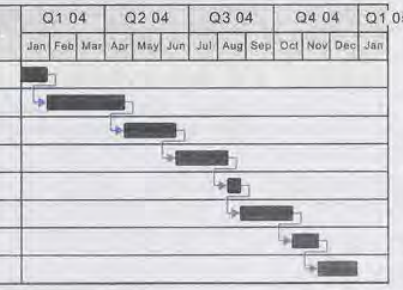


FIG 8.47 Gantt chart for simple chip

given team can be improved by design reuse or component-based design. It would seem that the time to design is proportional to the number of "modules" that are in the design raised to some power. That is, a four-module design is more than four times as complex as a single-module design. A module in this instance refers to a significant section of a chip such as a microprocessor, serial interface, or special functional unit.

Normally, projects are schedule-driven. In this case, it is important to make maximal use of design aids to meet the required schedule. Of importance is the cycle time of the so-called "edit-compile-debug" loop: i.e., the time it takes to make a change to the HDL; synthesize, place, and route it; and have a timing-verified final design. This can depend strongly on the efficiency of the design tools used, but if it is more than a day, design productivity can suffer. Ideally, the cycle is a few hours so that multiple bugs can be fixed each day.

Broadly speaking, schedules on the order of 18–24 months for a completely new chip seem to fit current average-complexity chips and state-of-the-art tools. For respins to slightly differentiate products, this can be reduced to six months or less, but there are certain fixed times such as IC fabrication and packaging that set hard limits on the complete design cycle time. Of course, for technologies such as FPGAs, design turnaround can be minutes (which is why FPGA verification is so important to ASIC or custom IC designs).

New microprocessors seem to take three to five years, and most experience one or more schedule slips.

8.5.6 Project Management

Project management is the overall supervision of the project. Tasks include making certain sufficient resources are available at the appropriate time, ensuring communication between different groups assigned to the project, and summarizing progress and risks to management. The development of processes for the conception, design, and ultimate manufacture of products is also the purview of the project manager.

There are two main ways to manage a chip design. The first is what might be called the rapid prototyping approach that is typical of startup companies, where a full-time project manager may be a luxury (and probably is more aptly named "seat-of-the-pants project management"). In this approach, a time goal is set and the workload is set to fit the time available. It is vital to rapidly get to the point where a prototype of the design is working—in essence, the skeleton—and the meat (detail) is gradually added. This can be risky.

The more conventional approach, which is appropriate for large companies and the military, is to preplan everything, estimating task times and putting these into a project planning tool. This approach, while necessary for large groups, tends to be feature-driven and rarely delivers products in shorter time scales than the rapid prototyping approaches. It is suitable when the tasks are well-defined and have been done before (then you know what the task times should be). The approach is stable and, depending on the team, often delivers products within budget and on time.

8.5.7 Design Reuse

Rarely is an IC designed as a single event. Rather, companies wish to amortize the development effort of a particular IC over several generations of products. This normally means that the design has to be transferred between several different processes. When design was mainly manual and at the mask level, a great deal of effort was expended on techniques to allow porting of designs between processes with the minimum of human intervention. Techniques used here include the use of symbolic layout methods and mask resizing software.

With the emergence of cell-based design, design migration falls into two steps:

- Acquiring or building a standard cell library in the new technology
- Retargeting the HDL description to the new cell library

The design and test generation does not have to be redone, although timing analysis and regression test bench simulation should definitely be completed.

In design flows where these steps cannot be followed, strict use of structured design techniques and software generator technologies can markedly improve porting times. Maintaining accurate and clear documentation will alleviate many problems downstream.

With the maturation of cell-based design, especially standard cell libraries and the use of hardware description languages, the notion of virtual components has become important as a method of transferring and reusing designs. Virtual components on an IC are notionally the same as discrete ICs used on a printed wiring board design. Each component has precisely defined behavior and a well-defined interface represented by a set of I/O pins and corresponding specifications for loading, setup and hold times, and delays. Components can be relatively simple or as complex as a RISC processor, MPEG decoder, or Wireless LAN modem. Virtual components can be classified as *hard*, *firm*, or *soft*. A hard module is normally defined at the mask level in a particular process. Thus, it will have a fixed floorplan, size, and a well-known set of timing parameters. A firm block will normally have a specific or generic netlist that describes each gate or register that must be used in the design (i.e., a 3-input NAND gate of normal power). This allows the design to be ported to multiple processes purely by netlist translation. The timing is dictated by the process and the final physical placement, however. A soft block is normally defined at the RTL level in the HDL. This captures the function of the block, but the detailed implementation is left to automated tools. Again, timing is dependent on the specific implementation. The Virtual Socket Interface Alliance monitors and encourages standards governing the implementation and use of virtual components.

8.6 Data Sheets and Documentation

A data sheet for an IC describes what it does and outlines the specifications for making the IC work in a system, such as power supply voltages, currents, input setup times, output delay times, and clock cycle times. The data sheet also includes package and pinout details.

A good habit to acquire is that of compiling a data sheet for any chip you might design. Not only is it the interface between the chip designer and the board-level designer, but also it is the interface to other members of the design team. In particular, it is good practice and is mandatory in industry to compile the data sheet for the chip and give it to the ultimate customer before the chip is fabricated. This prevents many undesirable scenarios that can arise when a perfectly designed chip meets a perfectly designed system. In this section, an outline of a typical data sheet will be reviewed by way of example.

8.6.1 The Summary

A summary of the chip includes the following details to orient the user:

- the designation and descriptive name of the chip
- a concise description of what the chip does
- a features list (optional for an internal product—but good for your ego!)
- a high-level block diagram of the chip function

8.6.2 Pinout

The pinout section should contain a description of the following pin attributes to document the external interface of the chip:

- name of the pin
- type of pin (i.e., whether input, output, tristate, digital, analog, etc.)
- a brief description of the pin function
- the package pin number

8.6.3 Description of Operation

This section should outline the operation of the chip as far as the user of the chip is interested. Programming options, data formats, and control options should be summarized.

8.6.4 DC Specifications

This section communicates the power dissipation and required voltages for the chip to correctly operate. The absolute maximum ratings should be stated for:

- supply voltage
- pin voltages
- junction temperature

The style of each I/O (i.e., TTL, CMOS, LVDS, ECL) should be summarized and the following DC specifications should be given over the operating range (temperature and voltage—i.e., mins and maxes):

- V_{IL} and V_{IH} for each input
- V_{OL} and V_{IH} for each output (at a given maximum drive current level)
- the input loading for each input
- quiescent current
- leakage current
- power-down current (if applicable)
- any other relevant voltages and currents

8.6.5 AC Specifications

The following timing specifications should be presented:

- setup and hold times on all inputs
- clock (and all other relevant inputs) to output delay times
- other critical timing such as minimum pulse widths

This data should be tabulated in table form and supported by a timing diagram where necessary. This is probably the most important section and an area where data provided ahead of the chip fabrication will aid the board designer. Designs are frequently snagged—for instance, when chip designers assume infinitely fast external memories and do not allow enough time between outputs changing and the next rising edge of the clock.

8.6.6 Package Diagram

A diagram of the package with the pin names attached should be supplied.

8.6.7 Principles of Operation Manual

Although the data sheet provides enough data to familiarize a user of a particular chip with the device, it is good practice to provide a Principles of Operation manual for internal users that have to test the chip or build support systems.

8.6.8 User Manual

A User Manual should also be provided. This is designed for use outside the group that designed the chip and can be a "cut down" version of the Principles of Operation manual.

8.7 Closing the Gap between ASIC and Custom



High-performance microprocessors and digital signal processing chips are typically designed in a custom methodology in which designers use a wide variety of circuit techniques and carefully tune all of the critical parts of the chip to achieve high performance at the expense of great design effort. Many other logic chips are built in a high-productivity ASIC methodology in which HDL code is synthesized onto a standard cell library and automatically placed and routed, as was described in Sections 8.4.1-8.4.2.

In a 180 nm process, custom microprocessors commonly operate at 1–2 GHz. In contrast, high-performance ASICs in comparable processes tend to operate at 200–350 MHz. Overall, custom design has consistently offered a 3–8x frequency advantage over ASIC designs. This is a remarkable difference, comparable to nearly 4–5 process generations of transistor improvement. [Chinnery02] offers a fascinating comparison of the two methodologies, explaining why custom flows offer so much higher performance and how ASIC designs can approach custom while maintaining high productivity. The remainder of this section summarizes the study's conclusions.

Measuring delays in terms of an FO4 inverter eliminates the process dependency and helps predict performance in the future. Overall, ASICs typically have cycle times of 50–100 FO4 inverter delays. High-performance ASICs such as the 520 MHz iCORE from STMicroelectronics can approach 25 FO4 delays per cycle. In comparison, the Pentium II and III operate at 20–24 FO4 delays, Alpha microprocessors at 13–16 FO4 delays, and the Pentium 4 at about 10 FO4 delays per cycle.

Chinnery identifies seven primary factors contributing to the custom performance advantage. Table 8.5 compares the effect of each of these factors. Custom design offers a large advantage over poor ASIC design in which performance is a secondary concern. Best practice ASIC methodologies can close much of the gap while still delivering far higher productivity than custom design. Poor is a relative term; often, economics demands fastest time to market rather than highest clock rate. Note that some factors affect only parts of the design so they cannot be simply multiplied to give the performance difference. The remainder of this section explores each of the factors.

Table 8.5 Relative performance of custom vs. ASIC design methodologies

Factor	vs. Poor ASIC	vs. Best Practice ASIC
Microarchitecture (e.g., pipelining)	1.8x	1.3x
Sequencing overhead: elements, skew, time borrowing	1.45x	1.1x
Circuit families (e.g., domino)	1.4x	1.2x
Logic design	1.3x	1.0x
Cell design, cell sizing, and wire sizing	1.45x	1.1x
Layout: floorplanning, placement, wire management	1.4x	1.0x
Exploiting process variation and accessibility	2x	1.2x

8.7.1 Microarchitecture

Many problems have inherent parallelism. Pipelining or performing multiple operations simultaneously can speed them up. The benefits depend on the degree of parallelism in the problem and in the sequencing overhead of the pipeline registers. Increasing pipelining or issuing more operations at once also increases the area, power consumption, and design effort. In the next section, we will see that ASICs have greater sequencing overhead and thus cannot benefit from pipelining as much as custom designs. However, many ASICs still have plenty of room to benefit from aggressive microarchitecture.

8.7.2 Sequencing Overhead

ASIC designers have little control of layout so they need sequencing elements that avoid races and are tolerant of noise. Most ASICs use very conservative flip-flops with a sequencing overhead (setup time + clock-to-Q delay) of 4–6 FO4 inverter delays.

In comparison, custom designs tend to use more aggressive sequential circuits. Some use pulsed latches or transparent latches. Some integrate logic into the latches to reduce sequencing overhead or use faster, but less robust elements when noise on the input or output can be better controlled. These improvements drive the sequencing overhead down to 2–3 FO4 delays.

Moreover, most systems are designed with a clock skew budget of about 10% of the cycle time. This corresponds to about 5 FO4 delays for a typical ASIC as compared to 1-2 for a custom design. The ASICs use clock distribution networks that are automatically generated by CAD tools. These networks are not as well balanced as in a good custom design and are much more susceptible to process variation. Moreover, they are not analyzed as carefully as in a custom design, so the clock skew budget must be conservative to account for uncertainties.

Finally, flip-flops impose hard edges, so clock skew directly impacts performance. They also are incapable of borrowing time to balance logic between pipeline stages or to opportunistically adjust for unpredicted delay variations. Custom latches have some skew tolerance. Custom designs also sometimes intentionally delay clocks to balance logic.

In summary, ASICs tend to have a sequencing overhead of about 10 FO4 delays, while custom designs have overhead as small as 3 FO4 delays. ASICs can reduce sequencing overhead by substituting pulsed latches or transparent latches for flip-flops on critical paths whenever tools determine that no races will occur. Better clock distribution networks and filtered power supplies on the PLL reduce skew.

8.7.3 Circuit Families

Most custom designs use skew-tolerant domino circuits on critical paths. Domino gates are 1.5–2x faster than complementary CMOS. Skew-tolerant domino pipelines use overlapping clocks to eliminate latches and their sequencing overhead. Custom designs occasionally use other circuit families such as CPL or pseudo-nMOS for specialized applications.

Domino is sensitive to noise and must be shielded from crosstalk. The monotonicity requirement tends to force the use of dual-rail gates. Domino has a high activity factor and thus high power consumption. It is not well understood by ASIC timing analyzers and test tools. While some groups are developing domino synthesis methodologies, many observers doubt that domino will gain widespread use in ASIC flows.

8.7.4 Logic Design

Custom designs tend to carefully optimize the logic design. Many functions have clever implementations that can be significantly faster. For example, lookahead adders are much faster than ripple carry adders, and Booth-encoded multipliers can be faster than those using simple arrays.

ASICs are synthesized from HDL code. Older synthesis tools took the code literally, so a Booth-encoded multiplier had to be specified manually. Modern tools are better at

choosing an appropriate implementation of common blocks like adders or multipliers (as long as you've purchased the vendor's intellectual property libraries!), but still need careful coding to produce good results for more exotic functions.

8.7.5 Cell and Wire Design

ASICs are synthesized onto a standard cell library. Some libraries have a small selection of gates. A library with only two drive strengths for NANDs, NORs, and inverters may be 25% slower than a library with a rich selection of drive strengths and buffer sizes and a full selection of compound and noninverting gates. The richer library can also reduce circuit area. Good libraries also offer asymmetric and skewed gates. While custom designs may use an unlimited set of cell sizes, this only offers a small percentage improvement over a good ASIC library with discrete sizes.

Wire capacitance and RC delay also are important. Custom designers choose the metal layer, wire width, and wire spacing and may even shield critical wires. ASIC designers have little control over the wires. Better ASIC tools are needed to automatically optimize wires on critical nets.

8.7.6 Layout

Global wires can account for a large fraction of total path delay and are getting worse with each process generation. Custom designers plan for these wires in advance and pipeline where necessary to tolerate the latency. They also add buffers and repeaters as needed. ASIC designers often do not think about the physical implementation and thus are surprised when long wires make a path much longer than simply counting gates would predict. Poor wire models in synthesis may mean these slow paths are not detected until after place and route has completed.

The key to managing wire delay is to carefully floorplan the chip. Units should be partitioned so that most communication happens inside the unit with short wires. Units that must communicate rapidly should be placed near each other. To reduce congestion, units communicating over wide busses should also be near each other. ASIC floorplanning tools are still maturing, but should eventually achieve parity with custom tools.

8.7.7 Process Variation

Custom designs are often fabricated on aggressive processes. For example, the Intel 130 nm process uses 60 nm effective channel lengths, while the TSMC process uses 80 nm effective lengths. Thus, an identical design will deliver a higher frequency on the Intel process. Intel also sometimes shrinks the effective channel length by another 5% as the process matures, improving performance even further. Foundries with higher performance processes tend to charge correspondingly higher prices.

Design corners also impact performance. Synchronous chips must be designed to operate at the worst temperature and voltage for which they are rated. Many ASICs are designed for the SS process corner so all of the chips fabricated without defects will meet timing. Thus, they can be tested only for functionality, not performance. Unfortunately,

the distribution of chips may have a long tail at the slow end, as shown in Figure 8.48.

Custom chips are often designed at the TT process corner to give a 17%–28% speedup at the expense of rejecting a few chips that fall in the tail. Moreover, the fastest 10% of the chips (the FF corner) may be up to 30% faster than TT. Microprocessors are commonly sorted by speed, and the fastest ones are sold at a premium. This is sometimes called *binning*. Over time, process improvements may yield even faster chips.

ASIC performance is being crippled by the lack of testing at speed. High-performance ASICs can be designed for the TT corner of a fast process rather than the SS corner of a slower process. In some markets, they may be binned to deliver some very fast chips.

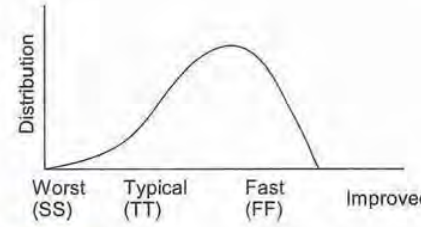


FIG 8.48 Distribution of chip behavior

8.7.8 Summary

Floorplanning and circuit families are commonly cited as the major advantages of custom over ASIC methodologies. Overall, [Chinnery02] concludes that these factors, while significant, are relatively overstated in their significance. Pipelining and process variation are of equal or greater significance. With attention to all of these details, ASICs may reduce their gap to within 2–3x the performance of custom designs.

8.8 CMOS Physical Design Styles

Basic gate layout was introduced in Section 1.5.4. In this section, we will examine the physical layout of CMOS gates in a general sense to understand the impact of the physical structure on the behavior and performance of circuits. For more extensive treatment by one of IBM's mask design instructors, see [Saint02].

8.8.1 Static CMOS Gate Layout

Complementary static CMOS gates can be designed using a single row of nMOS transistors below (or above) a single row of pMOS transistors, aligned at common gate connections. Most "simple" gates can be designed using an unbroken row of transistors in which abutting source/drain connections are made. This is sometimes called the "*line of diffusion*" rule, referring to the fact that the transistors form a line of diffusion intersected by polysilicon gate connections.

If we adopt this layout style, we can use automated techniques for designing such gates [Uehara81]. The CMOS circuit is converted to a graph where

1. the vertices in the graph are the source/drain connections
2. the edges in the graph are transistors gates that connect particular source/drain vertices

Two graphs, one for the pull-down network (n), and one for the pull-up network (p), result. Figure 8.49 shows an example of the graph transformation. The connection of edges in the graphs mirrors the series-parallel connection of the transistors in the circuits. Each edge is named with the gate signal name for that particular transistor. For example, the p-graph (light lines and circles) has four vertices: *Y*, *I1*, *I2*, and V_{DD}. It has four edges, representing the four transistors in the pull-up structure. Transistor *A* (*A* connected to gate) is an edge from the vertex *Y* to *I2*. The other transistors are similarly arranged in Figure 8.49(b). Note that the graphs are duals of each other because the pull-up and pull-down networks are the dual of each other. The n-graph (dark lines and crosses) overlays the p-graph in Figure 8.49(b) to illustrate this point. If two edges are adjacent in the p- or n-graph, then they can share a common source/drain connection and can be connected by abutment. Furthermore, if there exists a sequence of edges (containing all edges) in both graphs that have identical labeling, then the gate can be designed with no breaks in the line of diffusion. This path is known as a *Euler path*. The main points of the algorithm are:

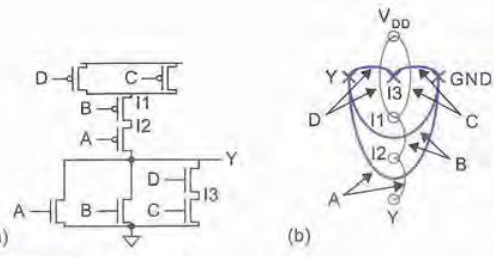


FIG 8.49 Circuit graphs

1. Find all Euler paths that cover the graph.
2. Find a p- and n-Euler path that have identical labeling (a labeling is an ordering of the gate labels on each vertex).
3. If the paths in step 2 are not found, then break the gate in the minimum number of places to achieve step 2 by separate Euler paths.

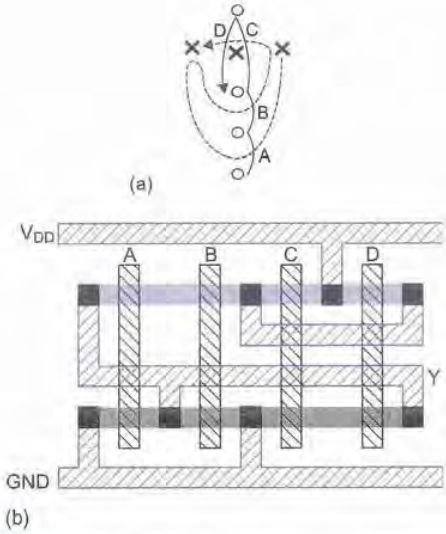


FIG 8.50 Stick diagram derived from Euler path

The original graph with a possible Euler path is shown in Figure 8.50(a). The sequence of gate signal labels in the Euler path is (*A*, *B*, *C*, *D*). To complete a layout, the transistors are arranged in the order of the labeling in parallel rows, as shown in stick diagram form in Figure 8.50(b). Vertical polysilicon lines form the gate connections. Metal routing wires complete the layout. This procedure can be followed when manually designing a gate, although good layouts usually become possible by inspection with a bit of practice.

A variation of the "line of diffusion" style occurs in circuits where a signal is applied to the gates of multiple transistors. In this case, transistors can be stacked on the appropriate gate signal using multiple rows of diffusion in a style called *gate matrix layout* [Wing82, Hu90]. This also

occurs in cascaded gates that cannot be constructed from a single row of transistors. A good example of this is the complementary XNOR gate. A schematic for this gate is shown in Figure 8.51(a). According to the style of layout that we have used to date, two possible layouts are shown in Figure 8.51(b) and Figure 8.51(c). The layout in Figure 8.51(b) uses the single row of n- and p-diffusion with a break, while that of Figure 8.51(c) uses a gate matrix layout. The selection of styles would depend on the overall layout—whether a short fat or long thin cell were needed. Note that the gate segments that are maximally connected to the power and ground rails should be placed adjacent to these signals.

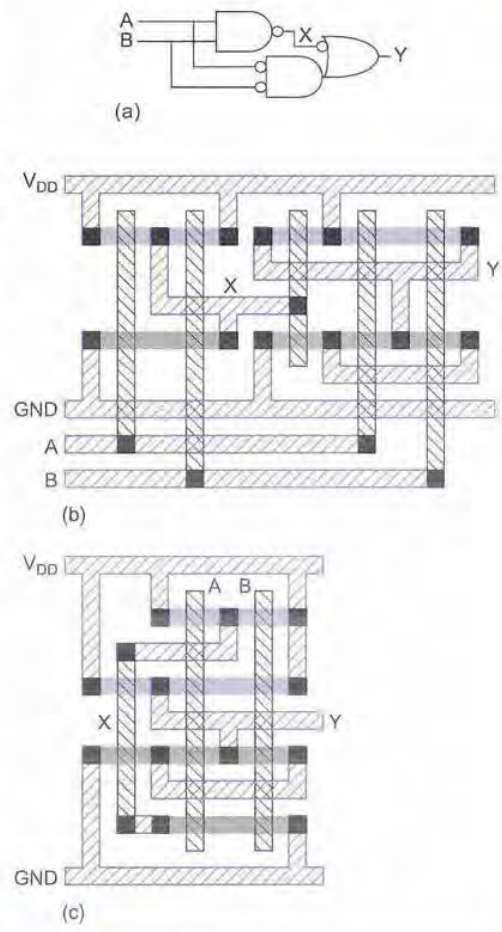


FIG 8.51 Broken line of diffusion and gate matrix cell layout styles

8.8.2 General CMOS Layout Guidelines

Layout can consume an unlimited amount of time because there are so many degrees of freedom and there is so much opportunity to squeeze a lambda here or there. In general, time to market is much more important than reducing chip area by a few percent, so it is important to settle on a simple and consistent layout design methodology. The following general layout guidelines can be stated:

1. Complete the electrical gate design and verification before layout. Circuit changes after layout is started become schedule busters.
2. Run V_{DD} and GND horizontally in metal at the top and bottom of the cell. Often these wires are wider than minimum to carry large DC currents without electromigration problems.
3. Run a vertical polysilicon line for each gate input.
4. Order the polysilicon gate signals to allow the maximal connection between transistors via abutting source/drain connections. These form gate segments.
5. Place n-diffusion segments close to GND and p-diffusion segments close to V_{DD}, as dictated by connectivity requirements.
6. Make connections to complete the logic gate in polysilicon (for short connections between gates) or metal. Squeeze transistors together to minimize diffusion between transistors.
7. Place well and substrate contacts under the supply lines in each cell.

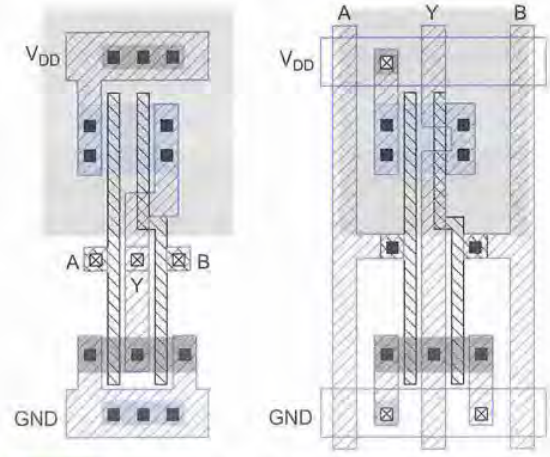


FIG 8.52 Standard cell metal usage

In general, metal layers should run perpendicular to each other to avoid "routing oneself into a corner." Exceptions are sometimes made to allow limited use of metal1 in the "wrong direction" to shorten connections or avoid the need for metal2 within a cell. Figure 8.52 shows two styles of standard cell layout for 2-input NOR gates. The first uses metal1 horizontally. The second uses metal1 vertically. Observe that the polysilicon gates are bent to minimize the diffusion between series transistors. The layouts assume that metal1–metal2 vias can be stacked on top of poly-metal1 contacts, as is common in modern planarized processes. If this is not allowed, the contacts must be placed adjacent to each other, sometimes increasing cell area.

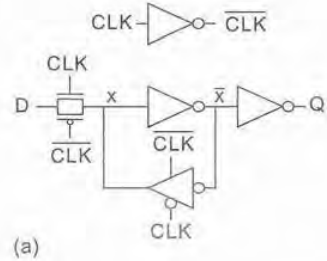
For standard cells, inputs and outputs must usually be routed to contacts near the center in current processes or the top or bottom of the cell where they can connect to the routing channels for older processes with few metal layers. In the vertical metal1 style, this often increases the cell area because the metal1 cannot run over the top of other contacts within the cell. In datapath cells, however, inputs and outputs can contact bitlines running over the top of cells parallel with V_{DD} and GND. In this case, the vertical metal1 style may be preferred because metal2 bitlines are free to run horizontally over the cells.

Other layout guidelines include:

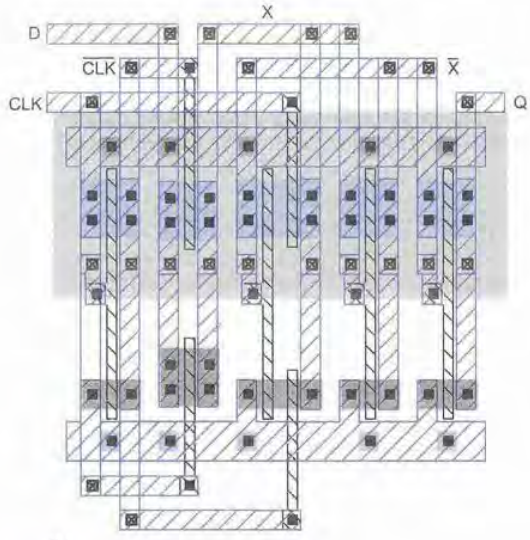
1. Diffusion has high resistance and capacitance. Never wire in diffusion. Minimize the area of diffusion regions. Fully contact large transistors to avoid series resistance through the diffusion between the contact and the edge of the transistor.
2. Polysilicon has high resistance, so use it only for short connections within cells. When long polysilicon lines are required (e.g., in the word line of a memory), strap the poly periodically with metal.
3. Lower levels of metal are thin and on a tight pitch. They are best for shorter connections (e.g., within a functional block) where density is important.
4. Upper levels of metal are thicker and on a wider pitch. They are faster and well-suited to global interconnections, the clock, and the global power/ground network. However, they are a scarce resource and must be carefully allocated.
5. Probe points should be placed on the top metal layer where they will be accessible during test (see Section 9.4).
6. Consider adding an assortment of unused gate array "happy gates" scattered through random logic. This facilitates making metal-only changes to fix logic bugs during silicon debug.

Note that the style of layout discussed involves optimizing the interconnection at the transistor level rather than the gate level. As a rule, smaller and perhaps faster layouts result by taking logic blocks with 10- to 100-transistor complexities rather than designing individual gates and trying to piece them together. For example, Figure 8.53(a) shows a transparent latch schematic. Figure 8.53(b) shows the latch layout built from simple standard cells, while Figure 8.53(c) shows an optimized layout with 2/3 the area. This improvement in density is due to a number of factors, including:

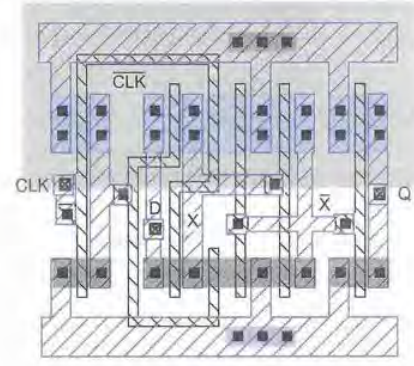
1. Better use of routing layers—routes can occur over cells
2. More "merged" source/drain connections
3. More use of "white space" (blank areas with no devices or connections) in sparse gates



(a)



(b)



(c)

FIG 8.53 Transparent latch layouts