Web Enhanced

This Web addendum to CMOS VLSI Design contains sections that have been moved out of the printed book to reduce the length.

9.4 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.

9.4.1 Differential Circuits

Several differential circuit families using nMOS pulldown networks are derived from the basic CVSL form, as shown in Figure W9.1.

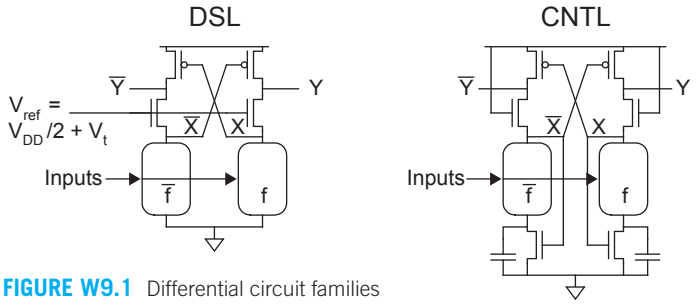


FIGURE W9.1 Differential circuit families

9.4.1.1 Differential Split-Level (DSL) *Differential Split-Level* (DSL) [Pfennings85] places nMOS transistors in series with the basic CVSL pulldown 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 pulldown stacks. The lower internal voltages also lead to lower electric fields across the pulldown 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 nanometer 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 pullup 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.

9.4.1.2 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 pulldown 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 W9.2, which is essentially a pseudo-nMOS gate with an extra transistor and shunting capacitor in series with the pulldown network. The shunting capacitor is built from the gate of an nMOS transistor. NTL consumes static power and is slower than pseudo-nMOS.

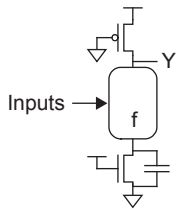


FIGURE W9.2 Nonthreshold Logic

9.4.2 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 12.2.3). 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 (9.1); 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 W9.3 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.

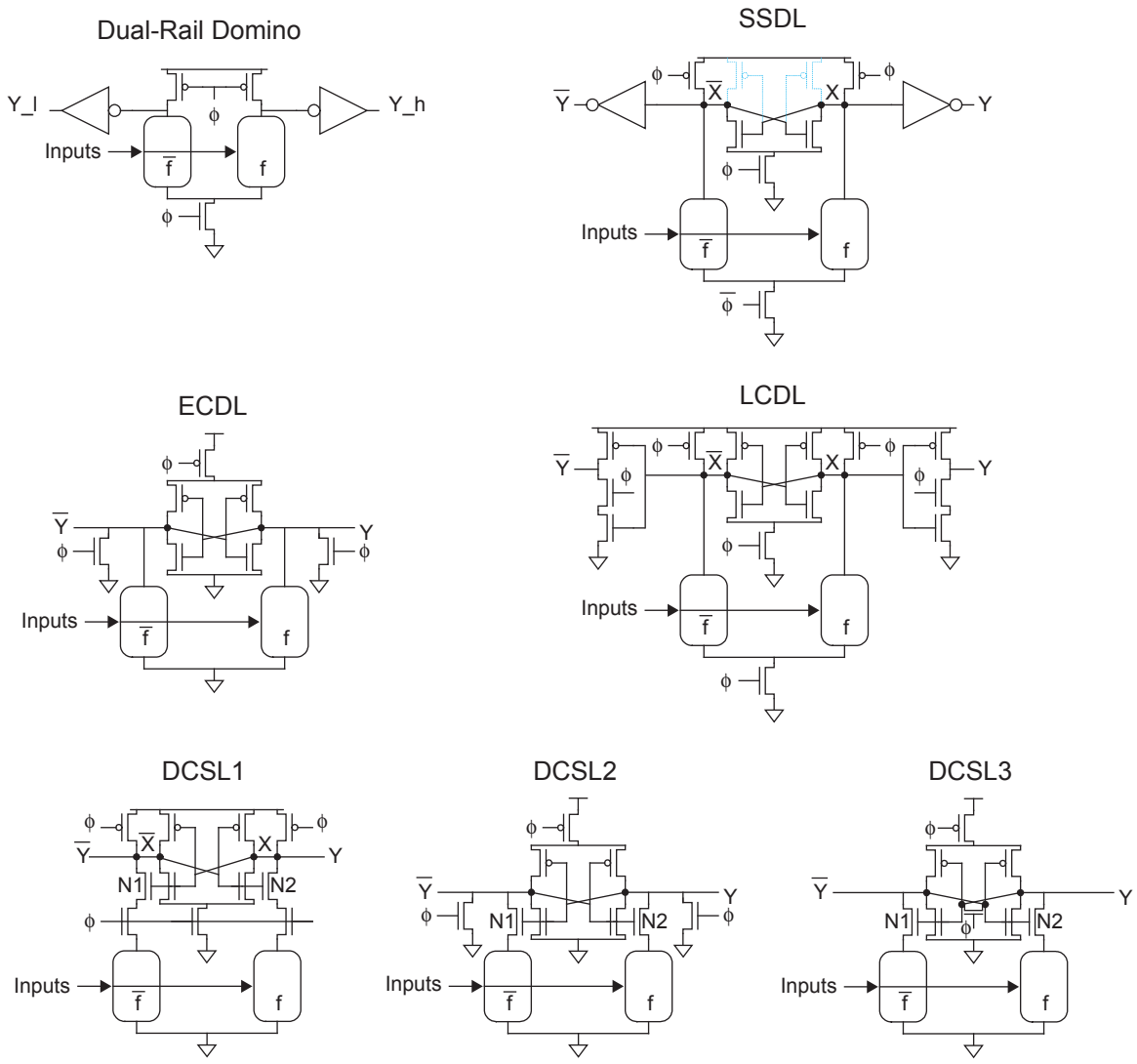


FIGURE W9.3 Sense-amplifier circuits

Figure W9.4 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 pulldown stage and the sense-amplifier stage may be better than the delay of a single complex stage.

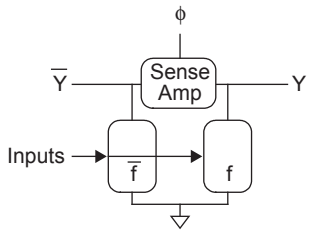


FIGURE W9.4 Generic sense-amplifier circuit

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 10.2.5).

As with clock-delayed domino (discussed in Section 10.5.4.2), 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 trade-off 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 W9.5(a) shows a pair of pulldown networks that are particularly sensitive to charge sharing noise. Figure W9.5(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 $\overline{Y}$. 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.

9.4.2.1 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 pulldown 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.

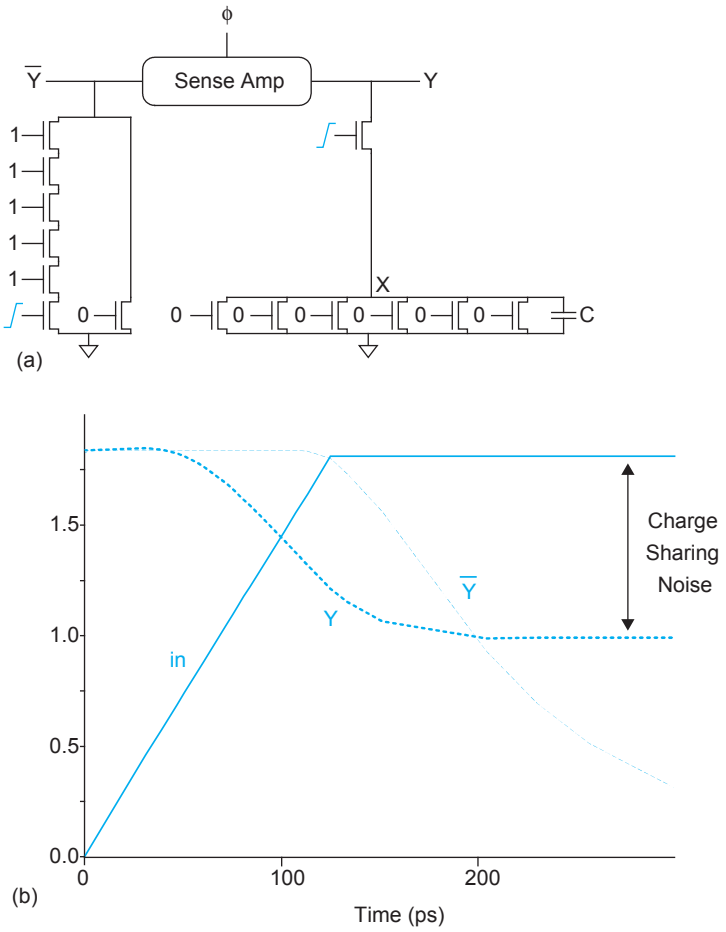


FIGURE W9.5 Charge sharing in sense-amplifier circuit

9.4.2.2 Enable/Disable CMOS Differential Logic (ECDL) *Enable/Disable CMOS Differential Logic* (ECDL) [Lu88, 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 W9.6(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 pullup 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 pulldown 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.

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 W9.6(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.

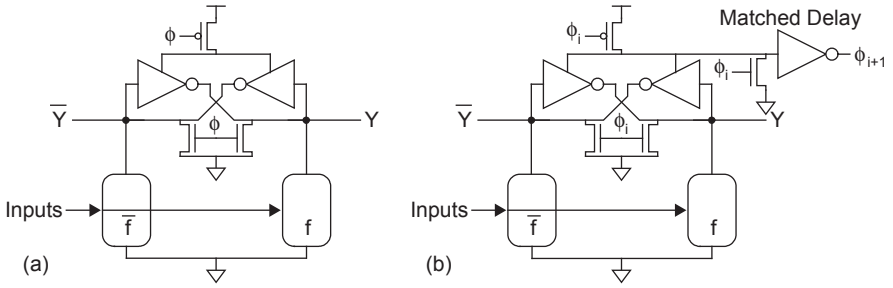


FIGURE W9.6
Enable/Disable CMOS Differential Logic

9.4.2.3 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.

9.4.2.4 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 pulldown 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 pulldown 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 pulldown 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.

9.4.2.5 Low-Voltage Swing Logic (LVS) *Low-Voltage Swing* (LVS) Logic [Deleganes04, Deleganes05] also uses a differential pair of series-connected nMOS networks connected to a sense amplifier. The networks are carefully balanced and equalized to minimize noise and allow long chains of transistors. LVS was extremely fast, but difficult to design and sensitive to process variation and noise. Intel used LVS extensively in the Integer unit of the 90 nm Pentium 4 processor, but later discarded the technique when it did not scale gracefully.

9.4.3 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 W9.7 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 *x* 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.'

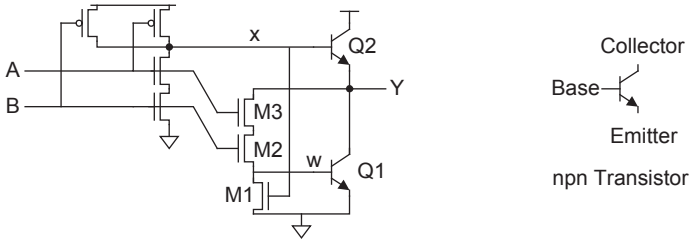


FIGURE W9.7 BiCMOS NAND gate

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.

9.4.4 Comparison

Table W9.1 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 fanin 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 fanin 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 pullup 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 10.5.1). 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.

10.3.11 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 C^2MOS stage with a pair of stages requiring only the clock, not its complement [Ji-ren87, Yuan89]. Figure W10.1 (a and b) shows active high and low TSPC dynamic latches. Figure W10.1(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.

TABLE 9.1 Comparison of circuit families

Family	nMOS	pMOS	Differ-ential	Static Power	Rail-to-Rail Output	Dynamic Nodes	Restor-ing	Ratioed	Cascade-able	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

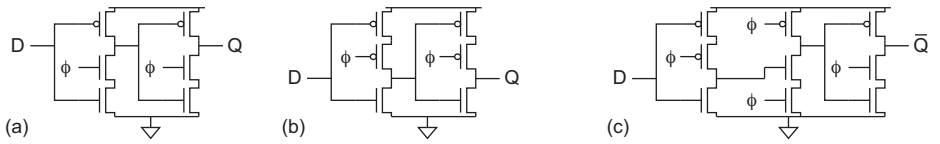


FIGURE W10.1 TSPC latches and flip-flops

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.

10.4.6 Two-Phase Timing Types

As discussed in Section 10.2, 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 W10.2 illustrates the timing of each of these types.

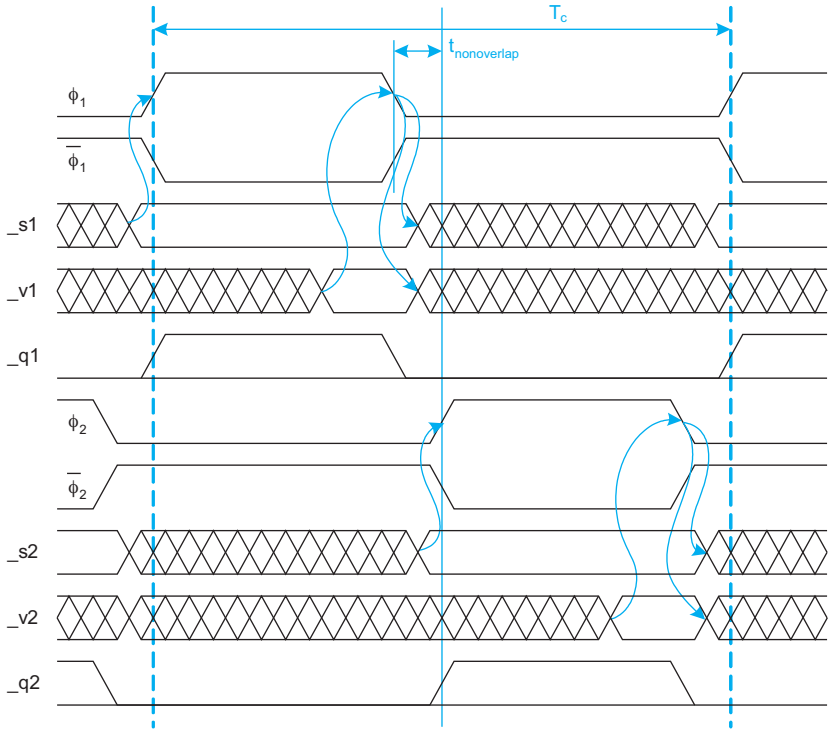


FIGURE W10.2 Timing types

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 W10.3. 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.

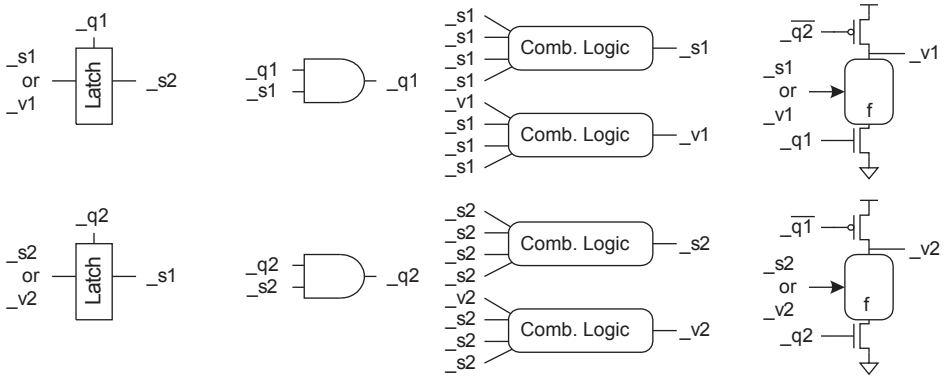


FIGURE W10.3 Rules for combining timing types

Figure W10.4(a) redraws the flip-flop of Figure 10.21 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 W10.4(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 sets up half a cycle earlier.

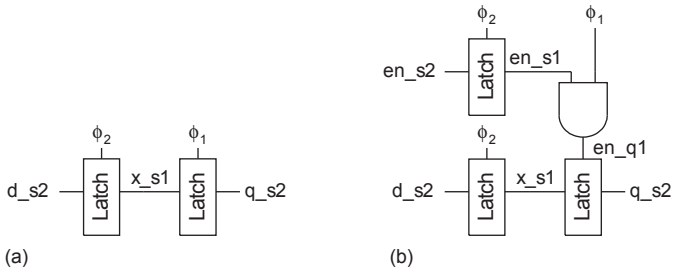


FIGURE W10.4 Flip-flops using two-phase nonoverlapping clocks

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.

Example W10.1

Annotate each of the signals in Figure W10.5 with its timing type. If the circuit contains any illegal connections, identify the problems and explain why the connections could cause malfunctions.

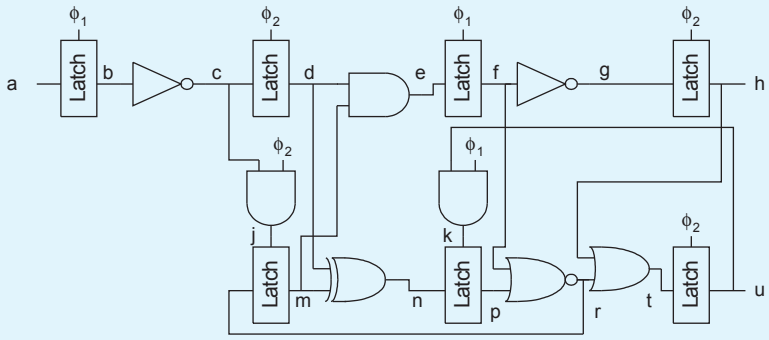


FIGURE W10.5 Example circuit for timing type checking

SOLUTION: Figure W10.6 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*.

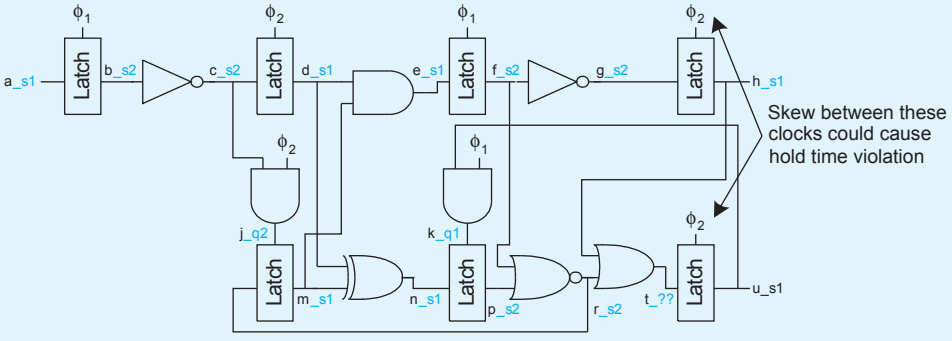


FIGURE W10.6 Annotated circuit showing timing types

10.5 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 10.5.1 divide the cycle into two half-cycles. One phase evaluates while the other precharges, and 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 10.5.2 use overlapping clocks to eliminate the latches and the sequencing overhead. Section 10.5.3 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 10.5.4 explores a selection of these *nonmonotonic* techniques.

10.5.1 Traditional Domino Circuits

Figure W10.7(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

$$t_{pd} = T_c - 2t_{pdq} \quad \textbf{(W10.1)}$$

Figure W10.7(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

$$t_{pd} = T_c - 2t_{\text{setup}} - 2t_{\text{skew}} \quad \textbf{(W10.2)}$$

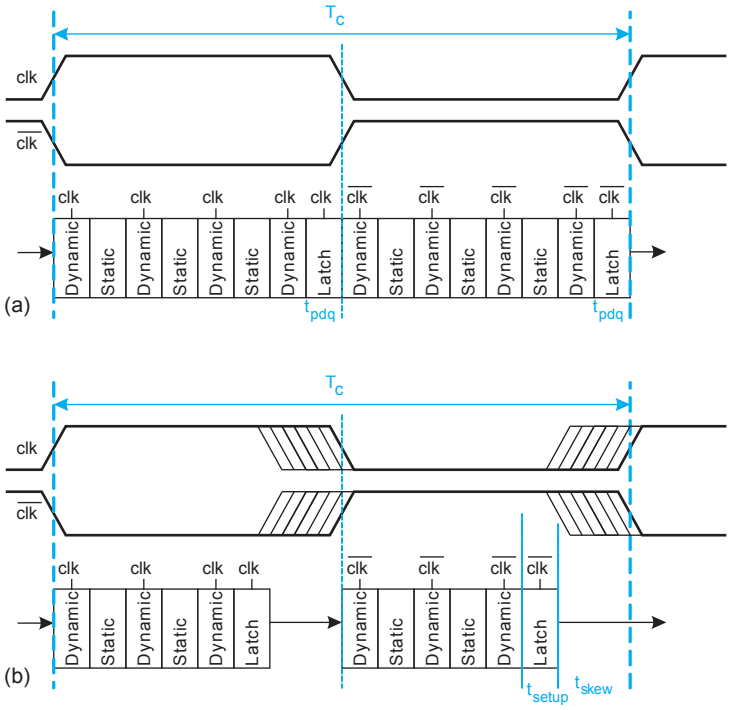


FIGURE W10.7 Traditional domino circuits

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.

10.5.2 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 W10.8 illustrates the implications of eliminating the latch. In general, let logic be divided into N phases rather than two half-cycles. Figure W10.8(a) shows the last domino gate in phase 1 driving the first gate in phase 2. Figure W10.8(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 W10.8(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 W10.9 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.

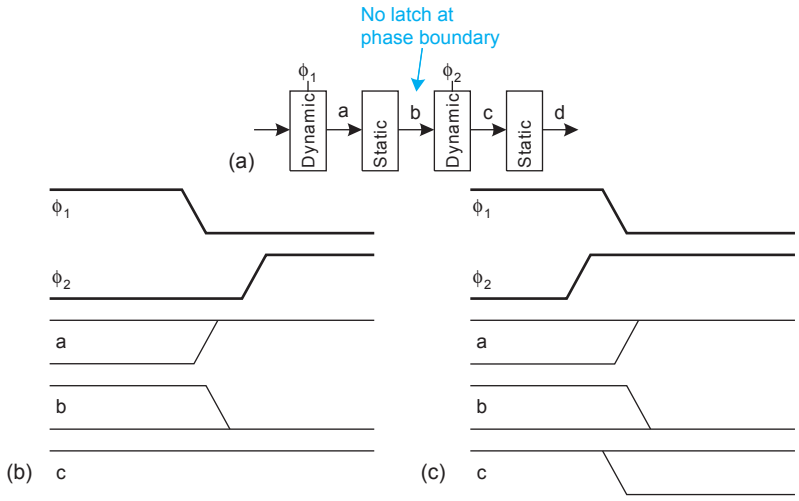


FIGURE W10.8 Eliminating latches in skew-tolerant domino circuits

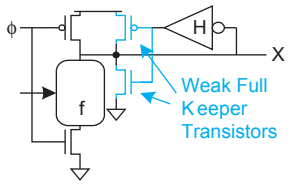


FIGURE W10.9 Full keeper

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

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* 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 W10.10. 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

$$t_{borrow} = t_{overlap} - t_{hold} - t_{skew} \quad \textbf{(W10.3)}$$

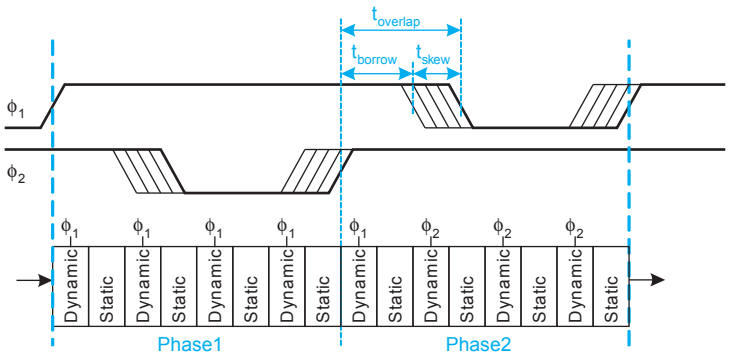


FIGURE W10.10 Time borrowing in skew-tolerant domino circuits

[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.

10.5.2.1 Two-Phase Skew-Tolerant Domino and OTB Domino Figure W10.11 shows a clock generator for the two-phase skew-tolerant domino system from Figure W10.10. 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

[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.

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

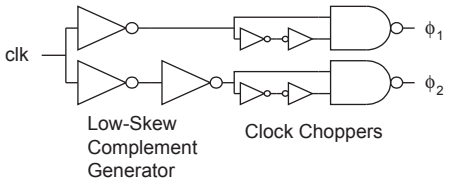


FIGURE W10.11 Two-phase skew-tolerant domino clock generator

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 W10.12. 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.

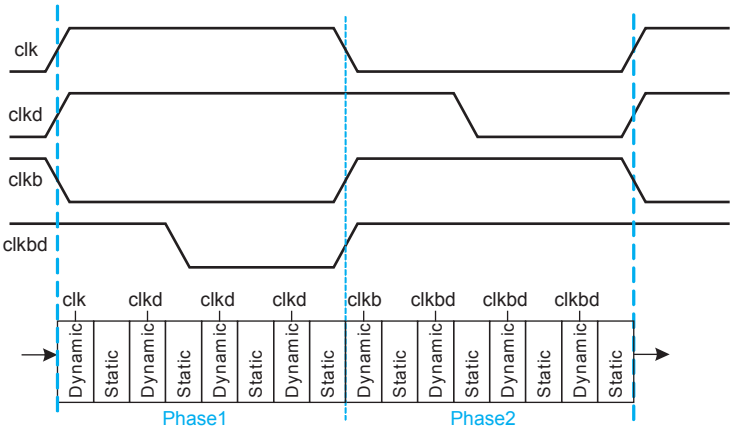


FIGURE W10.12 OTB domino

10.5.2.2 Four-Phase Skew-Tolerant Domino Figure W10.13 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 (W10.3), the maximum time available for borrowing from one phase to the next is

$$t_{\text{borrow}} = T_c / 4 - t_{\text{hold}} - t_{\text{skew}} \qquad \textbf{(W10.4)}$$

Figure W10.14(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

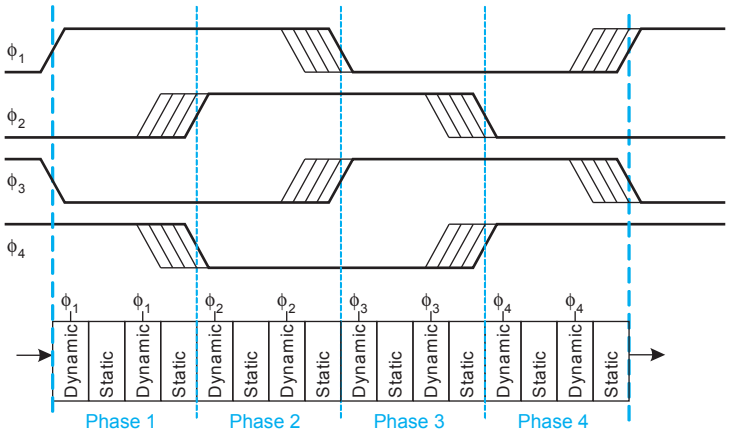


FIGURE W10.13 Four-phase skew-tolerant domino

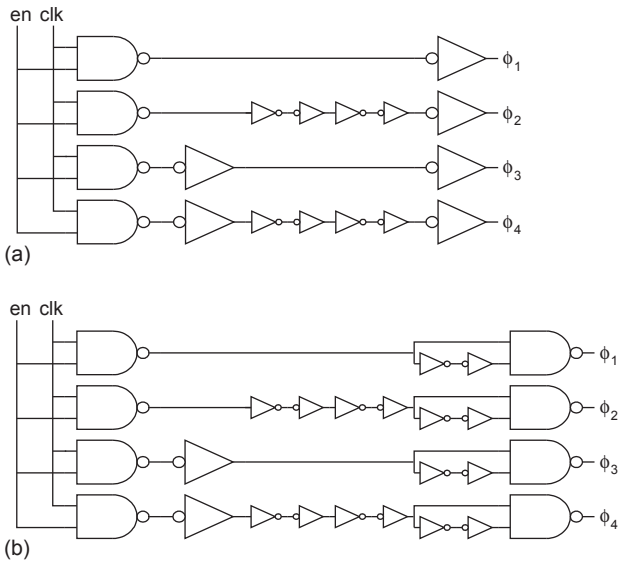


FIGURE W10.14 Clock generator for four-phase skew-tolerant domino

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 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 W10.14(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 10.4.6.

10.5.2.3 *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 W10.15 shows two ways of producing these phases. In Figure W10.15(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 W10.15(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

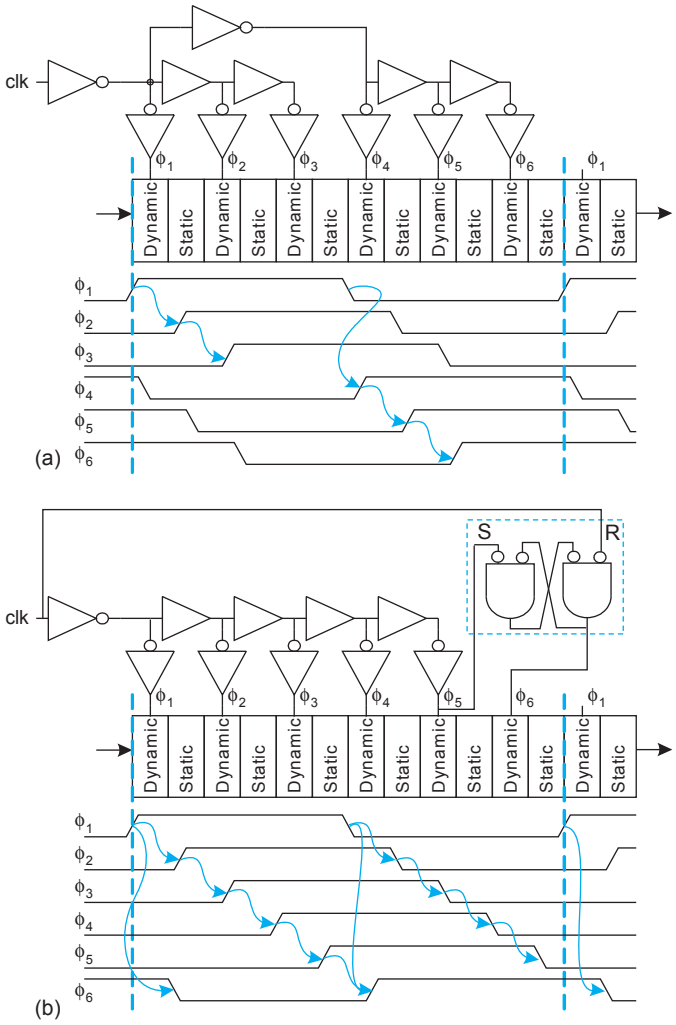


FIGURE W10.15 *N*-phase skew-tolerant domino

[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 10.5.3.

10.5.2.4 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 W10.16 shows a simple self-resetting domino gate. When the domino gate evaluates and the output rises, a timing chain produces a precharge signal *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.

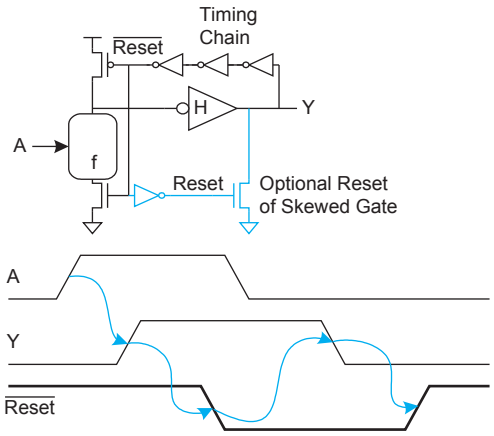


FIGURE W10.16 Self-resetting gate

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 12.2.2, we will see that this is particularly useful for RAM decoders. Only one of many wordlines 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 wordline drivers. For example, an IBM SRAM [Chappell91], the Intergraph Clipper cache [Heald93], and the Sun UltraSparc I cache [Heald98] use self-resetting gates.[2]

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 W10.17. The first inverter in the timing chain is replaced by a *generalized Muller C-element*, shown in blue, whose

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

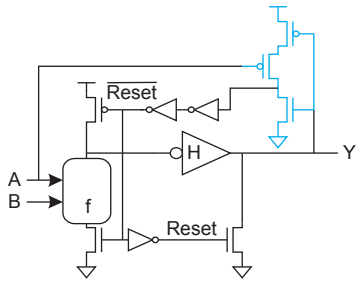


FIGURE W10.17 Predicated self-resetting gate

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.

[Proebsting91] applies self-resetting techniques to NORA gates for buffers and memory decoders. Figure W10.18 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

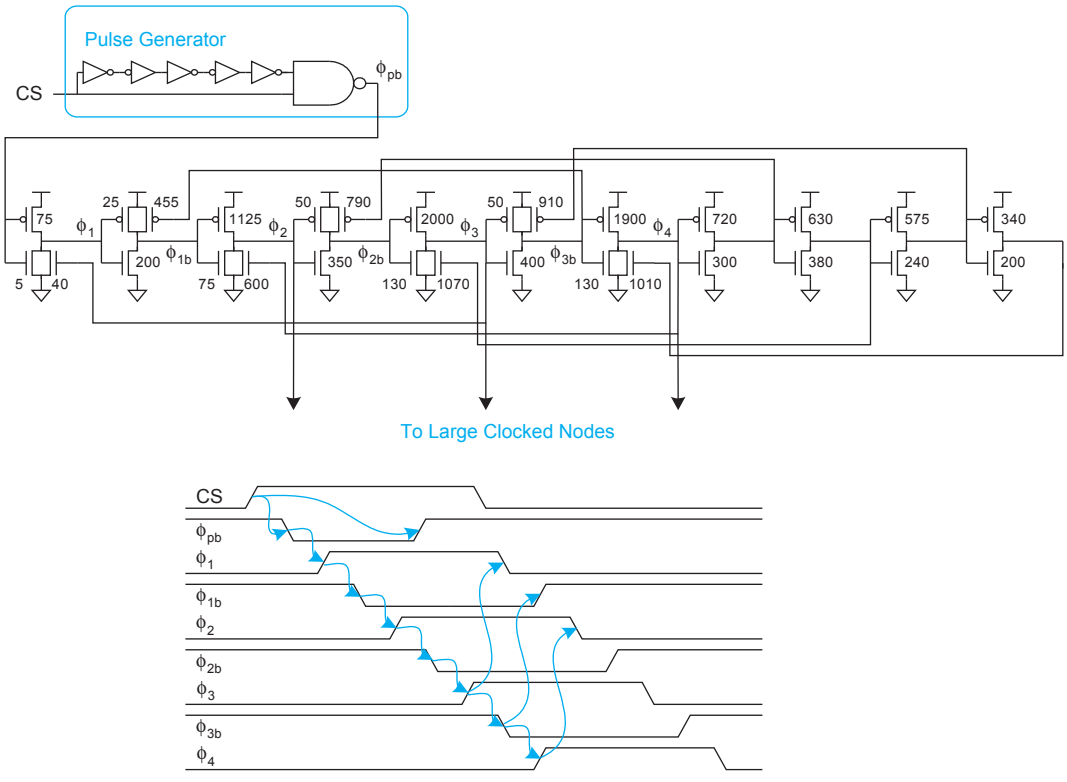


FIGURE W10.18 Postcharged buffer

HI- and LO-skew inverters with logical efforts of approximately two thirds and one third, 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. 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 W10.19. 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 $\overline{\text{se}}$ 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.

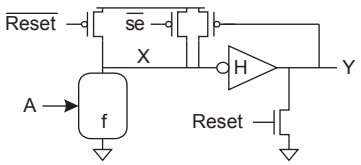


FIGURE W10.19 SRCMOS

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 W10.20 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 $\overline{\text{se}}$ is 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_h 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.

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.

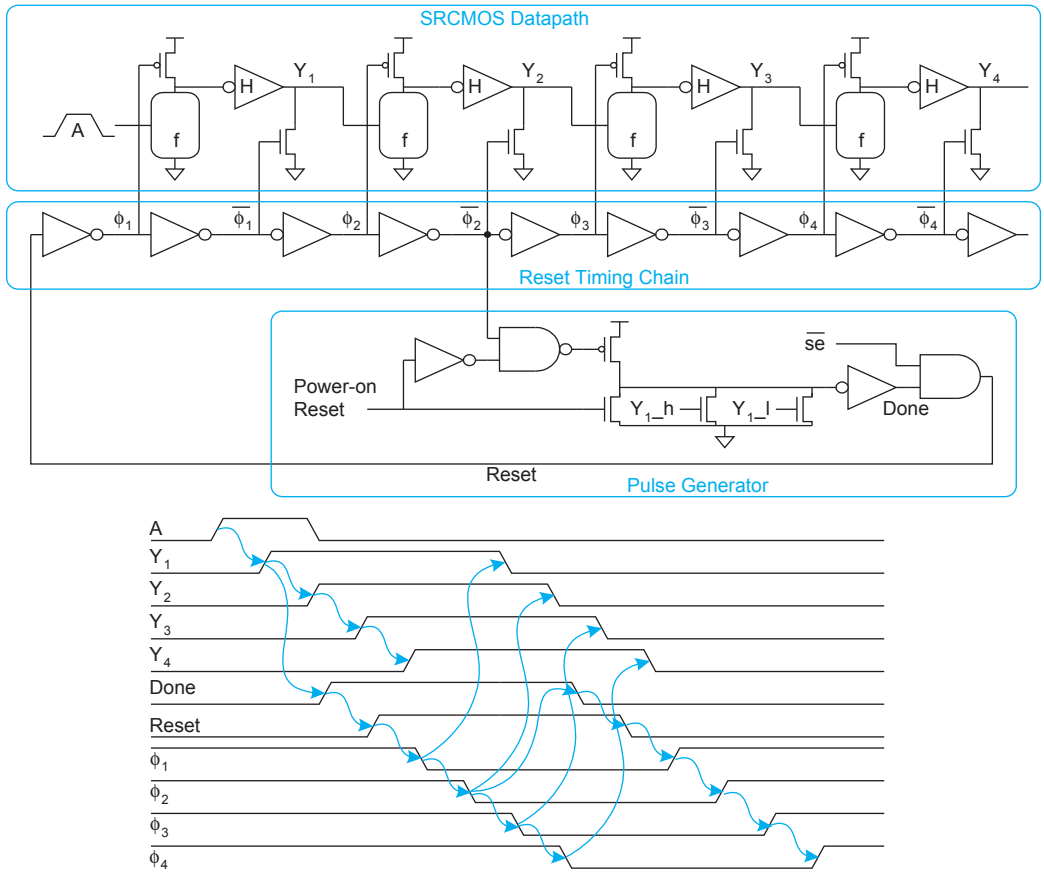


FIGURE W10.20 SRCMOS macro

- *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 W10.21 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

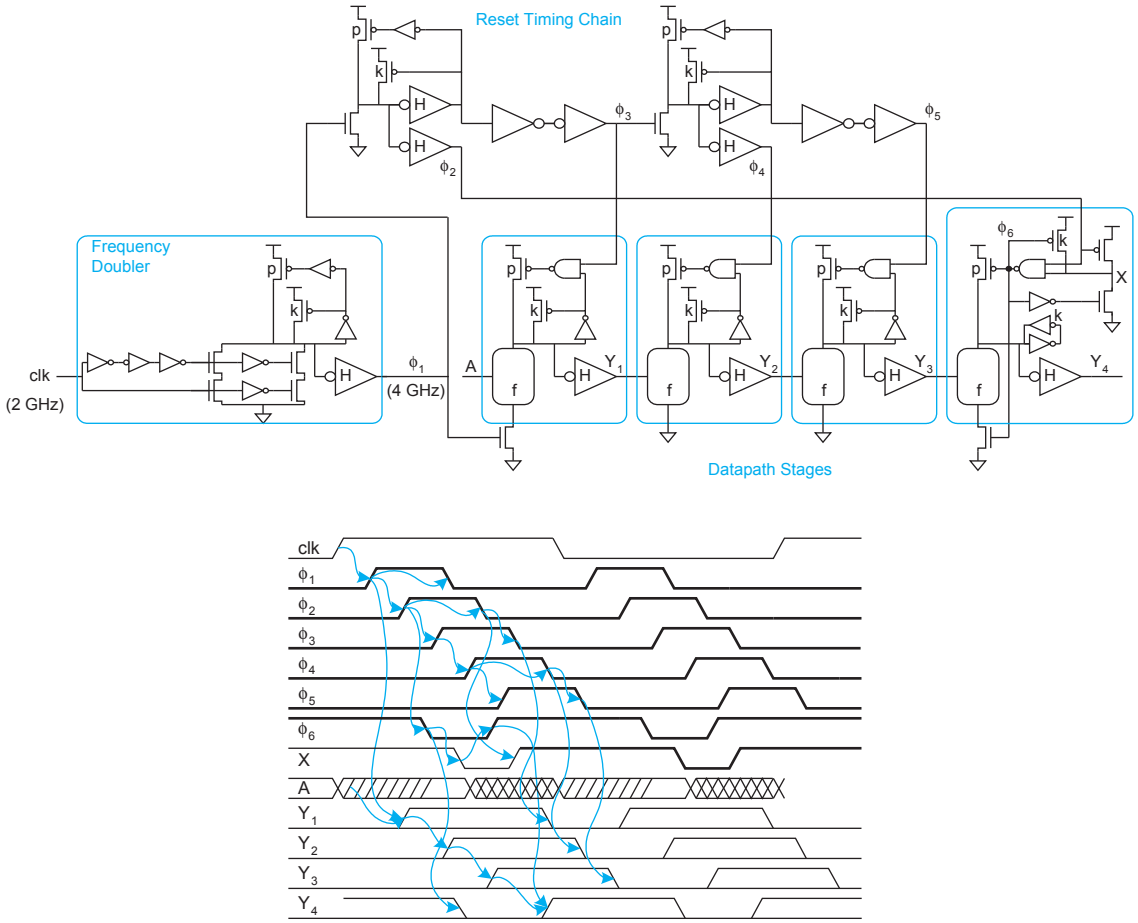


FIGURE W10.21 Global STP macro

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].

10.5.3 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. 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 W10.22(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 precharges 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 W10.22(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 shown in Figure W10.21.

The precharge time on each of the delayed phases in Figure W10.22(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 W10.23 shows a *delayed precharge* clock buffer [Colwell95] used on the Pentium II. The

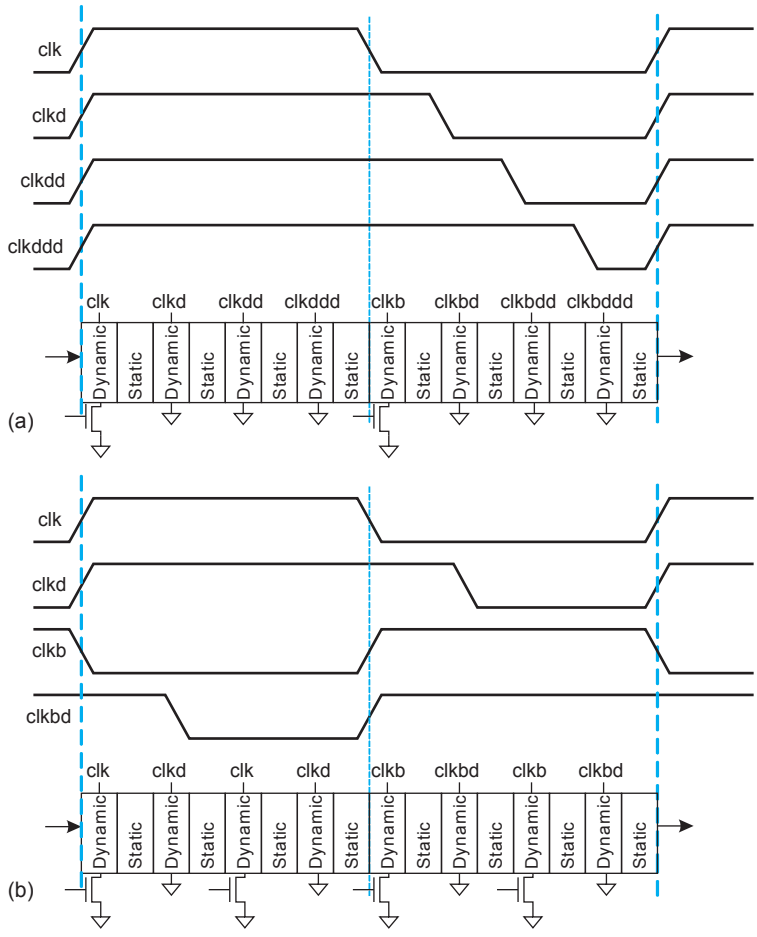


FIGURE W10.22 Clocking domino pipelines with unfooted gates

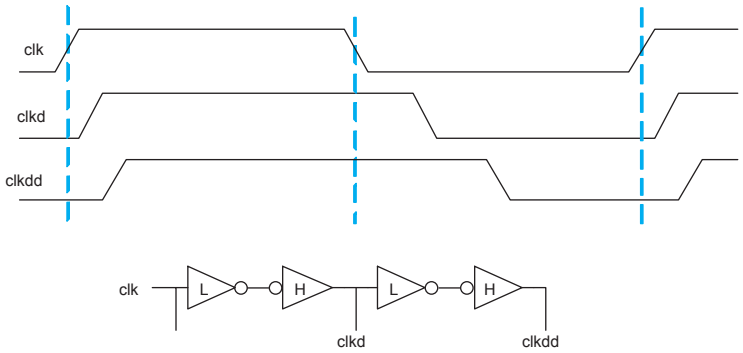


FIGURE W10.23 Delayed precharge clock buffer

delayed clocks are produced with skewed buffers that have fast rising edges but slower falling edges.

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, Figures W10.18, W10.20, and W10.21 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.

10.5.4 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 W10.24 compares a dual-rail domino 4-input OR/NOR gate to a 4-input dynamic NOR. The dual-rail design tends to be 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 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 shown in Figure 9.27. 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 in Figure W10.25. The primary trade-off 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.

Figure W10.25 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.

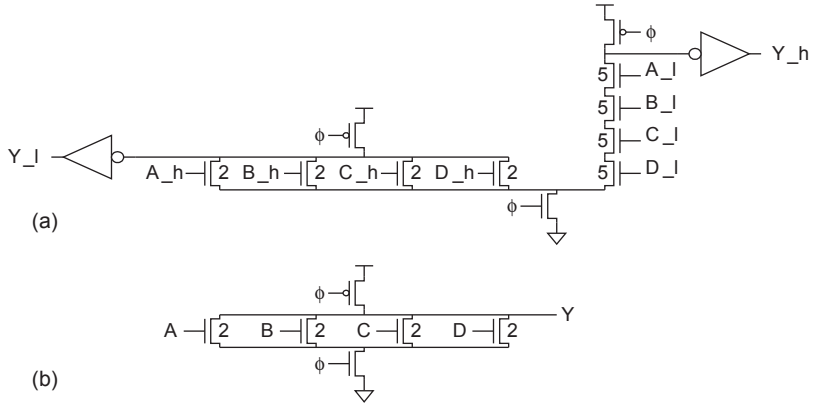


FIGURE W10.24 Comparison of NOR gates

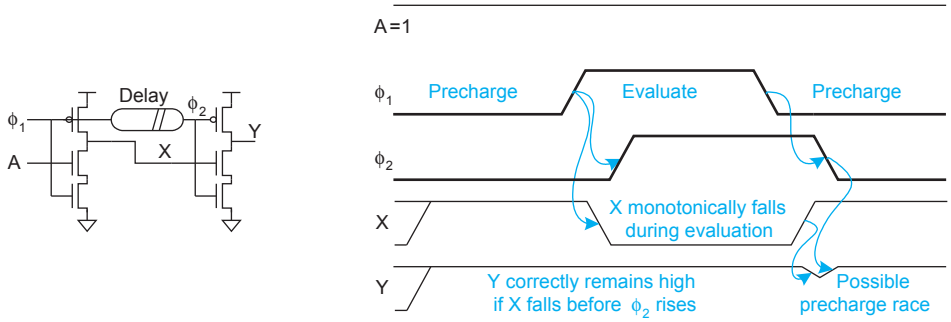


FIGURE W10.25 Cascading dynamic gates with a delayed clock

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

10.5.4.1 Delay Matching Figure W10.26 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.

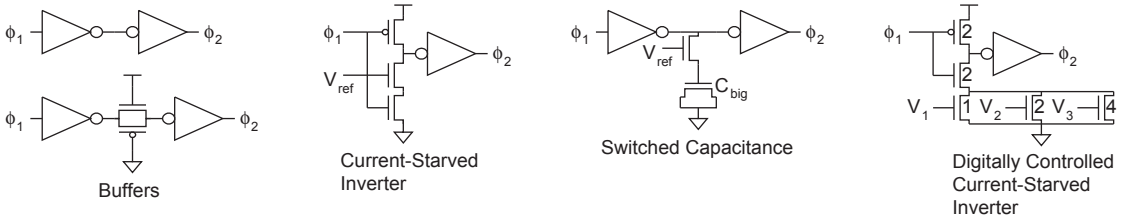


FIGURE W10.26 Delay elements

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 W10.27 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.

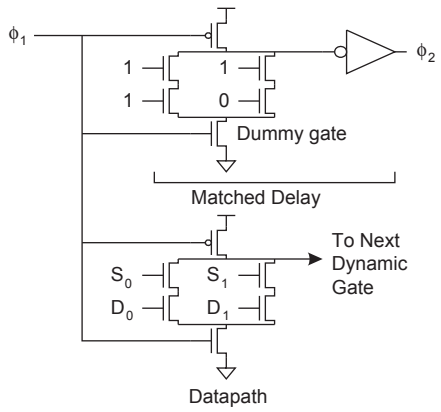


FIGURE W10.27 Delay matching with dummy gates

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 12.6 and 12.7). [Yee00] proposes wider use of delay matching in datapath applications and names the practice *Clock-Delayed* (CD) Domino.

10.5.4.2 Clock-Delayed Domino In the simplest CD Domino scheme, logic is *levelized* as shown in Figure W10.28(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 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 W10.28(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 fanin gates can be converted to wide dynamic NORs.

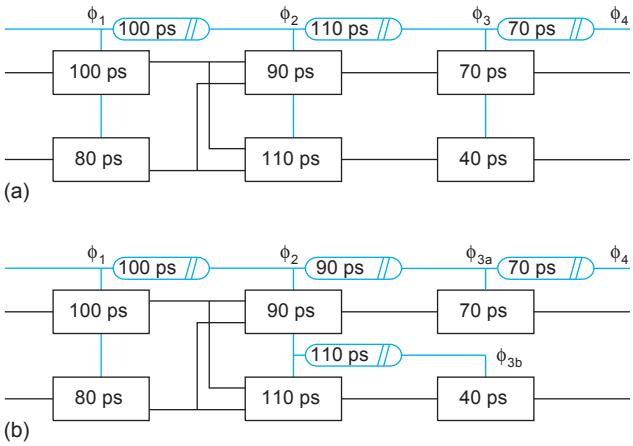


FIGURE W10.28 CD Domino timing

10.5.4.3 Race-Based Nonmonotonic Logic The Itanium II processor uses a specialized nonmonotonic structure called an *annihilation* gate for high fanin AND functions such as a 6-input decoder [Naffziger02]. An ordinary high fanin 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 W10.29 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.

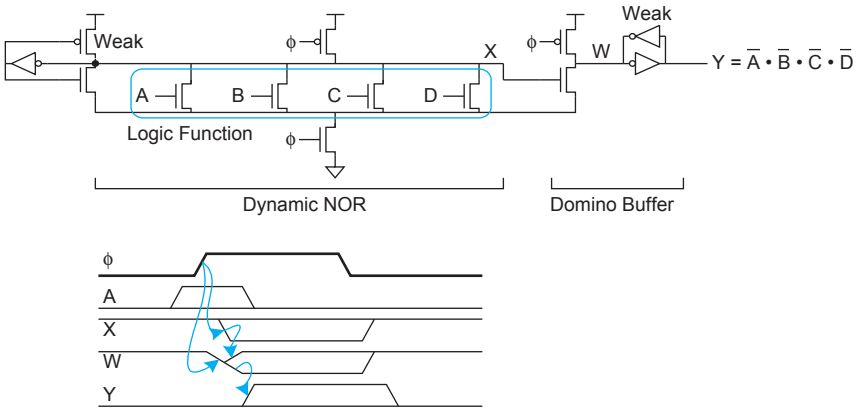


FIGURE W10.29 Annihilation gate

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 W10.30. 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.

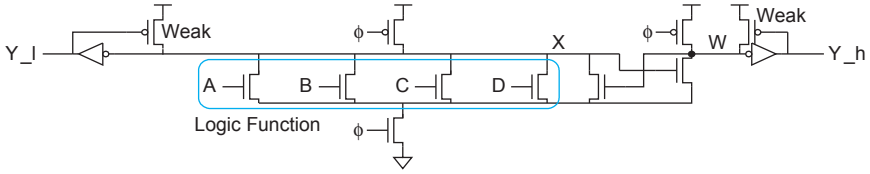


FIGURE W10.30 Latched domino gate

Intel uses a similar gate called a *Complementary Signal Generator* (CSG), shown in Figure W10.31, 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 pullups to restore the other node to full levels. These strong pullups also help fight leakage, permitting wide fanin 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.

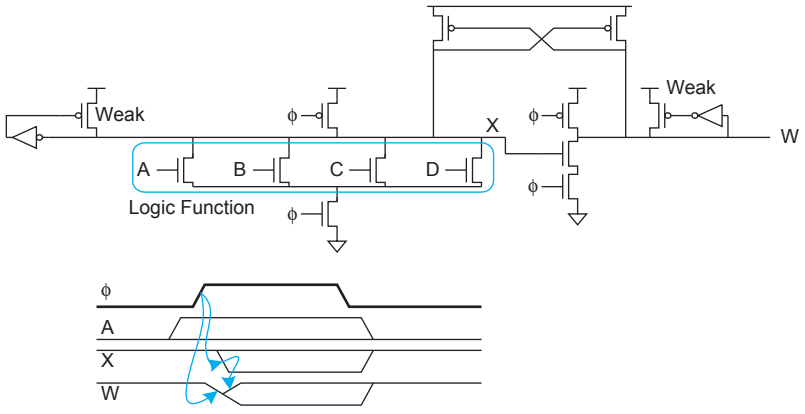


FIGURE W10.31 Complementary signal generator

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.

10.5.4.4 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 that 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 W10.32 shows a basic OPL gate consisting of a *Noise-Tolerant Precharge* dynamic stage (a dynamic gate with weak pMOS transistors to assist the keeper [Yamada95, Murabayashi96, Thorp99]). 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.

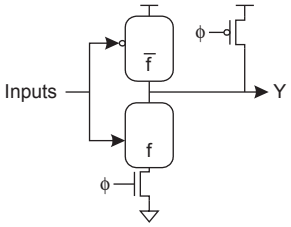


FIGURE W10.32 OPL gate

Figure W10.33 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 W10.34 shows

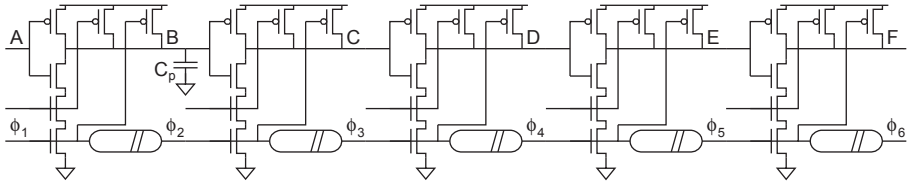


FIGURE W10.33 Chain of OPL gates

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*.

In Figure W10.34(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 W10.34(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 W10.34(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 W10.35 plots the delay from *A* to *F* against the spacing between clocks. The nMOS transistors are two units wide and the figure compares the performance for pMOS of one, three, or five 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.

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 fanin NOR structures such as pseudo-nMOS or dynamic gates, as illustrated in Figure W10.36(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

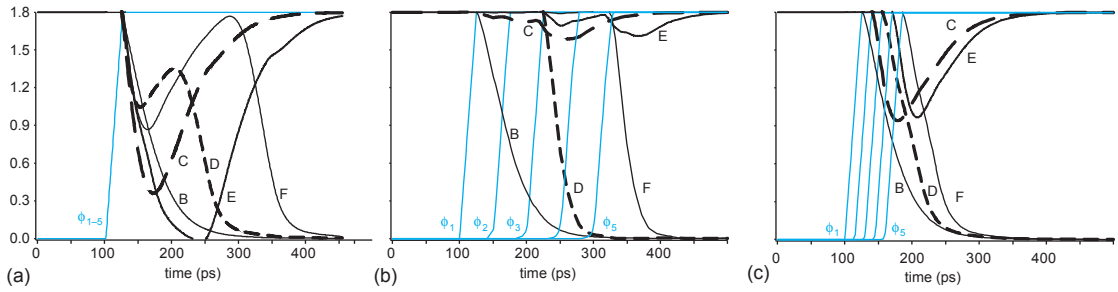


FIGURE W10.34 OPL waveforms for various clock delays

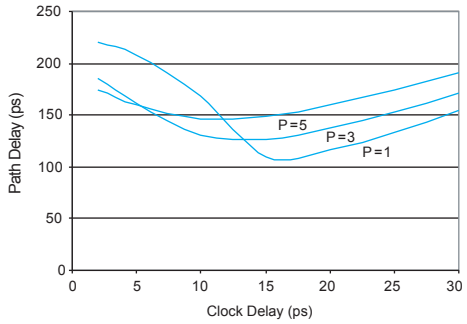


FIGURE W10.35 Path delay vs. clock delay

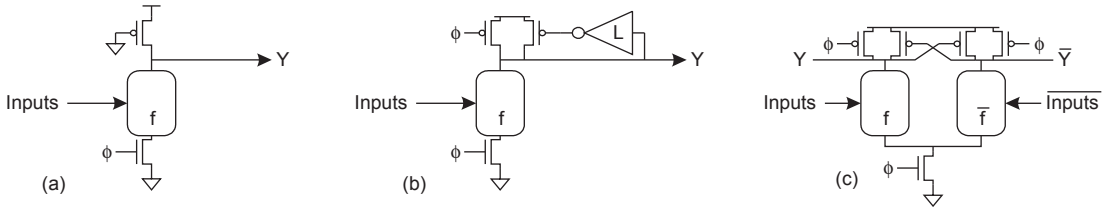


FIGURE W10.36 Alternative OPL circuit families

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 W10.36(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.

Other OPL implementations of functional units can be found in [Guo05, Chong06].

10.5.5 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, as well as circuitry at the domino-to-static interface to convert the pulses into levels.

10.5.5.1 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 W10.37(a). The latch is opaque while the domino gates evaluate. Figure

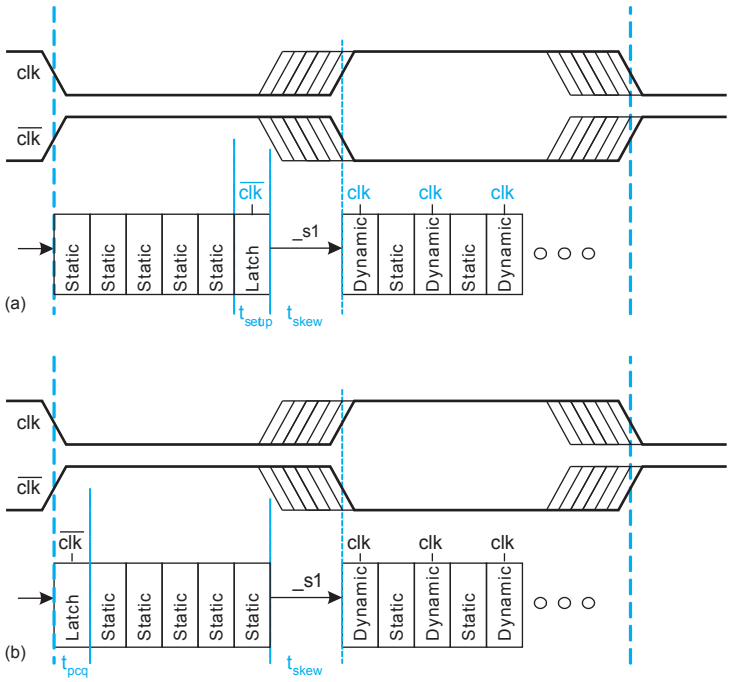


FIGURE W10.37 Latch at static to domino interface

W10.37(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.

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 W10.38. 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 W10.39 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].

10.5.5.2 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.

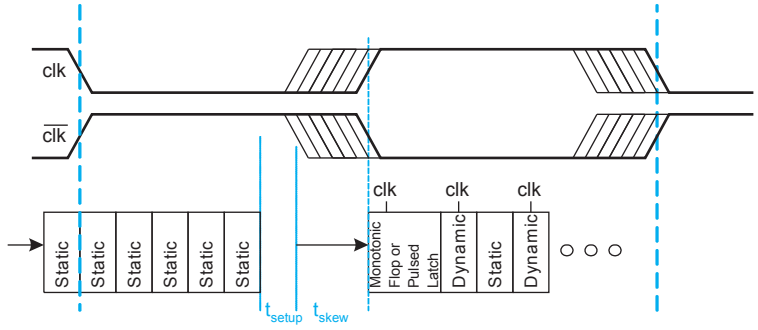


FIGURE W10.38 Monotonic flip-flop or pulsed latch at static to domino interface

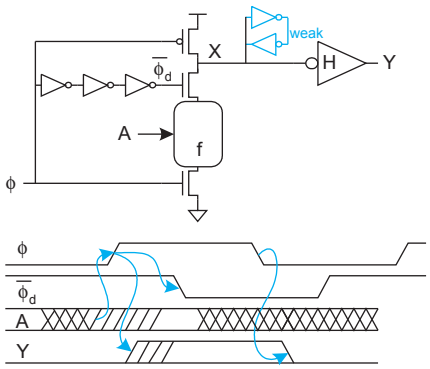


FIGURE W10.39 Pulsed domino flip-flop with integrated logic

Figure W10.40 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 a SR latch (see several designs in [Nikolic00]).

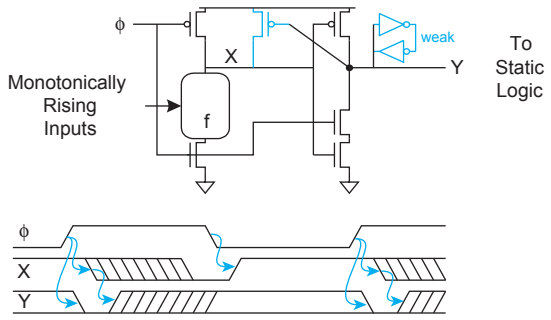


FIGURE W10.40 Domino gate with pulse-to-level output conversion

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 W10.41 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 that the inputs do not arrive until several gate delays after the clock edge. The DLC also may include scan circuitry illustrated in Section 15.6.2.3.

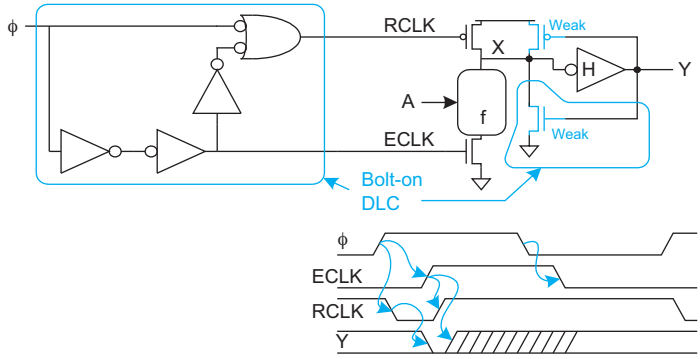


FIGURE W10.41 Dynamic latch converter

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 W10.41 showed such a pulse generator for Global STP domino.

10.5.6 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 9.2.4.3 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 W10.42(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 W10.42(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 W10.42(c). The inverting delay logic (IDL) can be an inverter, three inverters in series, or some other inverting structure with greater delay.

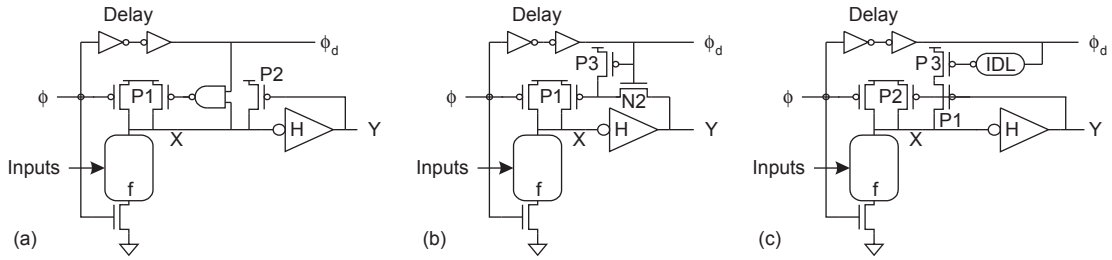


FIGURE W10.42 Delayed keepers

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].

10.9 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 W10.43 shows the date of introduction and the performance of several generations of these processors.

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 7.36 showed a die photo of the 42-million transistor Pentium 4. The chip consumes about 55 watts. A top-of-the-line Pentium 4 sold in 1000-unit quantities for around $400–$600

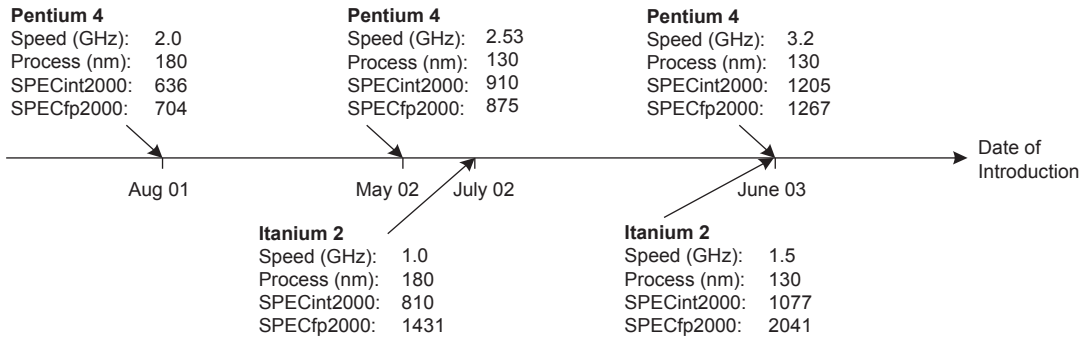


FIGURE W10.43 Microprocessor timeline

(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 W10.44 shows a die photo of the Itanium 2 with a 3 MB 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 6 MB 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.

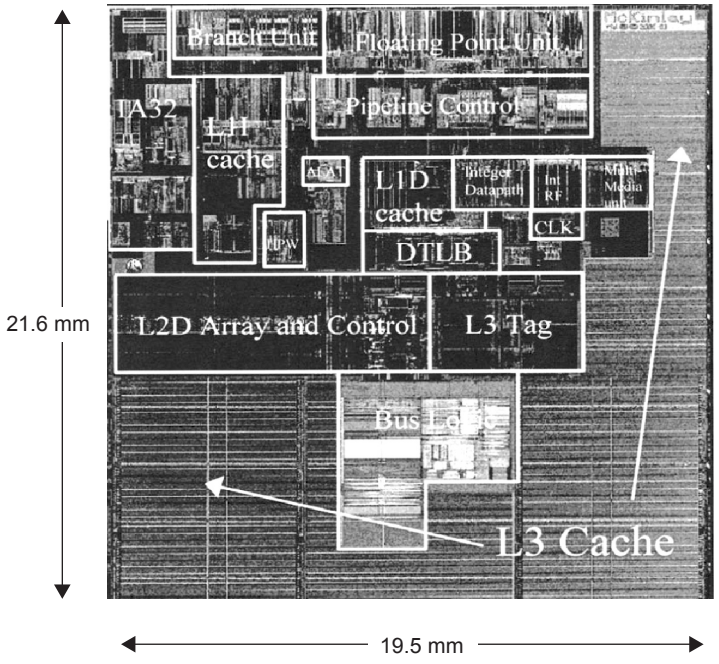


FIGURE W10.44 Itanium II die photo (© IEEE 2002.)

10.9.1 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 shown in Section 13.4.4.3. 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 Global STP domino. 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 W10.45 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

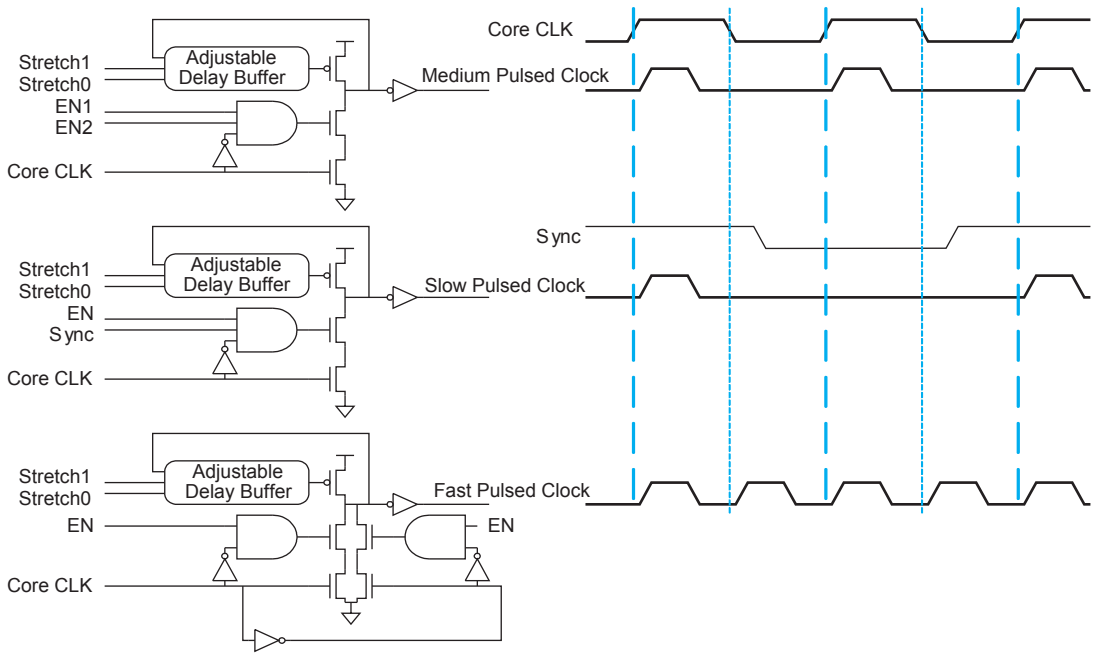


FIGURE W10.45 Pulse generators

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.

The 90 nm Pentium 4 adopted even more elaborate LVS circuits described in Section 9.4.2.4. However, design for extreme clock frequencies consumed too much power. Moreover, these circuits did not scale well as process variation increased and supply voltage decreased. Intel eventually abandoned these techniques and moved to the Core architecture, running at a lower frequency using mostly static logic and fewer pipeline stages.

10.9.2 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 shown in Section 13.4.4.2. 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 W10.46 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.

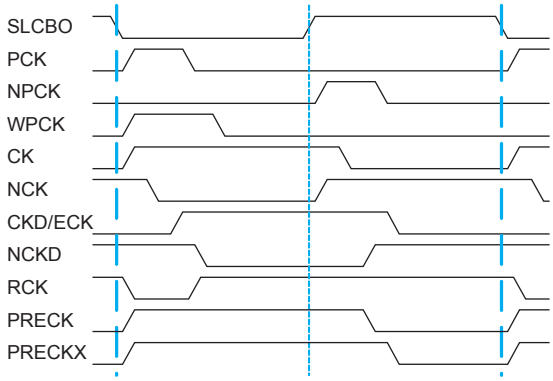


FIGURE W10.46 Clock gater waveforms

In the Itanium 2, 95% of the static logic blocks use Naffziger pulsed latches with 125-ps wide pulses called *PCK*. 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*. 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 W10.47 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).

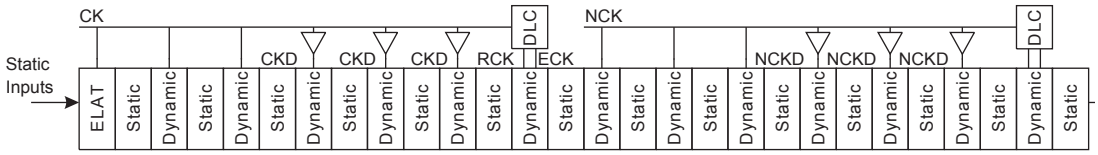


FIGURE W10.47 Four-phase skew-tolerant domino pipeline

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. 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 are shown in Figure W10.47 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 W10.48 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.

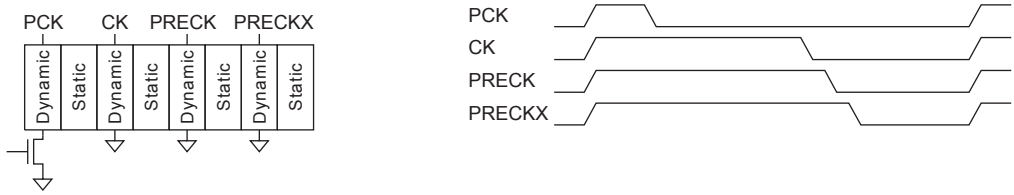


FIGURE W10.48 Clocking for footless domino

[Naffziger06, Fisher06, McGowen06, and Stackhouse09] describe the evolution of these techniques in subsequent Itanium processors and the introduction of dynamic voltage scaling.

11.2.2.4 Manchester Carry Chain Adder The carry chain can also be built from switch logic using propagate, generate, and kill signals. The majority gate of Figure W11.1(a) can be replaced with a switch network. Figure W11.1(b) shows a static implementation

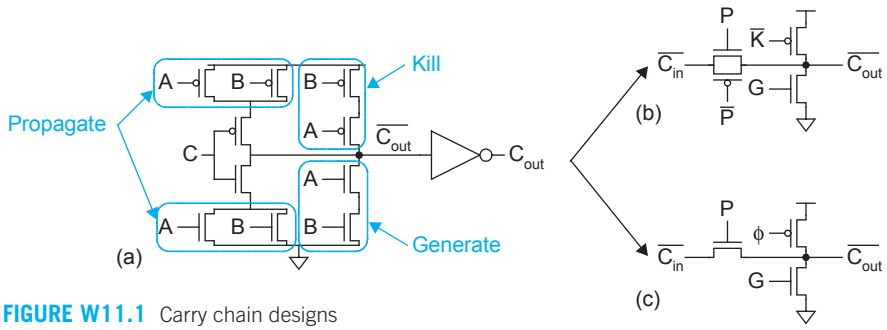


FIGURE W11.1 Carry chain designs

operating on a complementary carry. The complementary carry can be propagated through the transmission gate, generated with the nMOS transistor, or killed with the pMOS transistor. Figure W11.1(c) shows a dynamic version that is faster and requires less hardware.

Multiple stages are directly connected to build a *Manchester carry chain*, as shown in Figure W11.2(a) [Kilburn59]. The resistance and capacitance of the carry chain grow with the length, so the delay grows with the square of length. This is clearly not viable for long adders. As with long wires, the delay can be made linear with length by periodically breaking the chain and inserting an inverter to buffer the signals. The best chain length depends on the parasitic capacitance and can be determined through simulation or calculations for a particular technology (see Exercise 11.4), but is typically 3 or 4; Figure W11.2(b) shows a valency-4 carry chain. The widths of the transistors along the chain can be tapered to reduce parasitic delay.

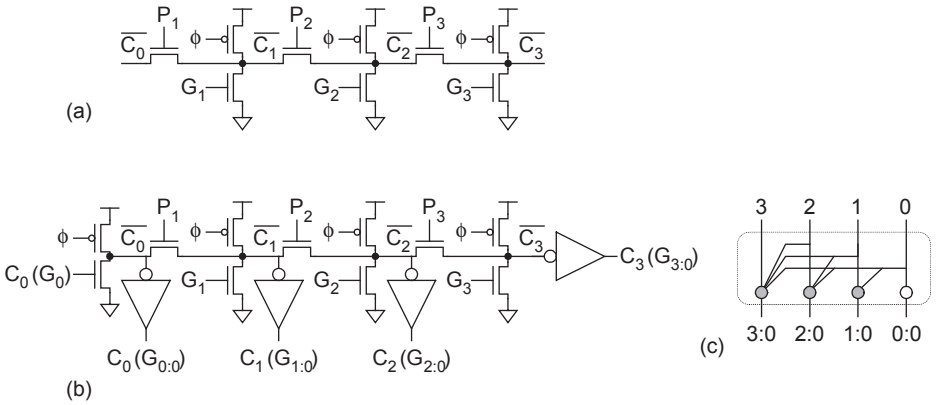


FIGURE W11.2 Manchester carry chains

Observe that the Manchester carry chain computes the functions

$$\begin{aligned} C_0 &= G_{0:0} = C_0 \\ C_1 &= G_{1:0} = G_1 + P_1 C_0 \\ C_2 &= G_{2:0} = G_2 + P_2\left(G_1 + P_1 C_0\right) \\ C_3 &= G_{3:0} = G_3 + P_3\left(G_2 + P_2\left(G_1 + P_1 C_0\right)\right) \end{aligned} \qquad \textbf{(W11.1)}$$

$G_{3:0}$ is analogous to the valency-4 group generate circuit of EQ_(11.8), while the other outputs are the generate signals for smaller groups (including a simple buffer of the input). In other words, the carry chain can be viewed as a buffer and three gray cells of increasing valency, as shown in Figure W11.2(c). If the carry chain of Figure W11.2(b) is redrawn in a more conventional form (Figure W11.3), it can also be seen to be another representation of a footless multiple-output domino gate, as discussed in Section 9.2.4.6.

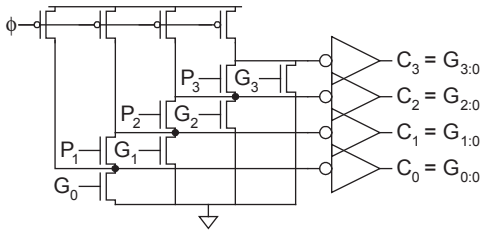


FIGURE W11.3 Equivalence of Manchester carry chain and multiple-output domino gate

Figure W11.4 shows a Manchester carry chain adder using valency-4 stages. It is similar to the carry-ripple adder, but uses $N/3$ stages.

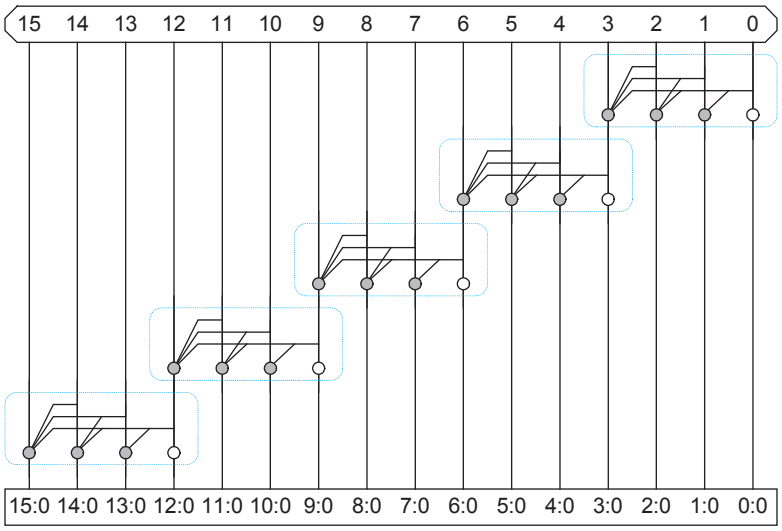


FIGURE W11.4 Manchester carry chain adder group PG network

11.2.2.12 An Aside on Domino Implementation Issues Using $\overline{K} = A + B$ in place of P, all the group generate signals $G_{i:0}$ are monotonic functions of the noninverted inputs and can be computed with single-rail domino gates. However, the final sum XOR is inherently nonmonotonic and cannot be computed this way. The two common choices for designers of domino adders are to build the final sum XOR with static logic or to construct the entire adder out of dual-rail domino.

Domino adders with a static sum XOR produce nonmonotonic outputs that must be stabilized on a clock edge before driving subsequent domino gates. As adders are often used in self-bypass loops where the output of the adder serves as one of the inputs on the next cycle, this introduces a hard edge and the associated costs of setup time and clock skew into the critical path.

The alternative is to build a dual-rail domino sum XOR accepting monotonic true and complementary (*sig_h* and *sig_l*) versions of the carries. Producing these carries in turn requires extra hardware all the way back to the adder inputs, which also must be provided in dual-rail form. If the sum is also computed in dual-rail form, the outputs can be directly bypassed to the inputs in a skew-tolerant fashion. The drawback of such adders is the extra hardware involved in the group PG network. Again, there are two common cell designs. One is to build dual-rail group propagate and generate signals, i.e., four signals per bundle. Another is to use monotonic 1-of-3 hot propagate-generate-kill (PGK) signals.

The first approach uses the following logic:

$$\begin{array}{lll} G_i\_h = A_i\_h \bullet B_i\_h & G_0\_h = C_{\text{in}}\_h & G_{i:j}\_h = G_{i:k}\_h + K_{i:k}\_l \bullet G_{k-1:j}\_h \\ K_i\_h = A_i\_l \bullet B_i\_l & K_0\_h = C_{\text{in}}\_l & K_{i:j}\_h = K_{i:k}\_h + G_{i:k}\_l \bullet K_{k-1:j}\_h \\ G_i\_l = A_i\_l + B_i\_l & G_0\_l = 0 & G_{i:j}\_l = G_{i:k}\_l \bullet G_{k-1:j}\_l \\ K_i\_l = A_i\_h + B_i\_h & K_0\_l = 0 & K_{i:j}\_l = K_{i:k}\_l \bullet K_{k-1:j}\_l \end{array} \quad \textbf{(W11.2)}$$

$$\begin{aligned} P_i\_h &= A_i \oplus B_i = A_i\_h \bullet B_i\_l + A_i\_l \bullet B_i\_h = G_i\_l \bullet K_i\_l \\ P_i\_l &= \overline{A_i \oplus B_i} = A_i\_h \bullet B_i\_h + A_i\_l \bullet B_i\_l = G_i\_h + K_i\_h \\ S_i\_h &= G_{i-1:0} \oplus P_i = G_{i-1:0}\_h \bullet P_i\_l + K_{i-1:0}\_h \bullet P_i\_h \\ S_i\_l &= \overline{G_{i-1:0} \oplus P_i} = G_{i-1:0}\_h \bullet P_i\_h + K_{i-1:0}\_h \bullet P_i\_l \end{aligned} \quad \textbf{(W11.3)}$$

Observe that the group generate *G_h* and *G_l* and kill *K_h* and *K_l* signals are not truly complementary; they are sometimes called *pseudo-complements* [Wang93]. They take advantage of the symmetry of the addition function so that the same type of gate can be reused. It is left to the reader to recursively verify that $G_{i-1:0}\_h$ and $K_{i-1:0}\_h$ are the true and complementary versions of the carries into bit *i*. Schematics of each gate are shown in Figure W11.5(a).

The 1-of-3 hot version uses less logic:

$$\begin{array}{lll} G_i = A_i\_h \bullet B_i\_h & G_0 = C_{\text{in}}\_h & G_{i:j} = G_{i:k} + P_{i:k} \bullet G_{k-1:j} \\ P_i = A_i\_h \bullet B_i\_l + A_i\_l \bullet B_i\_h = A \oplus B & P_0 = 0 & P_{i:j} = P_{i:k} \bullet P_{k-1:j} \\ K_i = A_i\_l \bullet B_i\_l & K_0 = C_{\text{in}}\_l & K_{i:j} = K_{i:k} + P_{i:k} \bullet K_{k-1:j} \end{array} \quad \textbf{(W11.4)}$$

$$\begin{aligned} P_i' &= G_i + K_i \\ S_i\_h &= G_{i-1:0} \bullet P_i' + K_{i-1:0} \bullet P_i \\ S_i\_l &= K_{i-1:0} \bullet P_i' + G_{i-1:0}\_h \bullet P_i \end{aligned} \quad \textbf{(W11.5)}$$

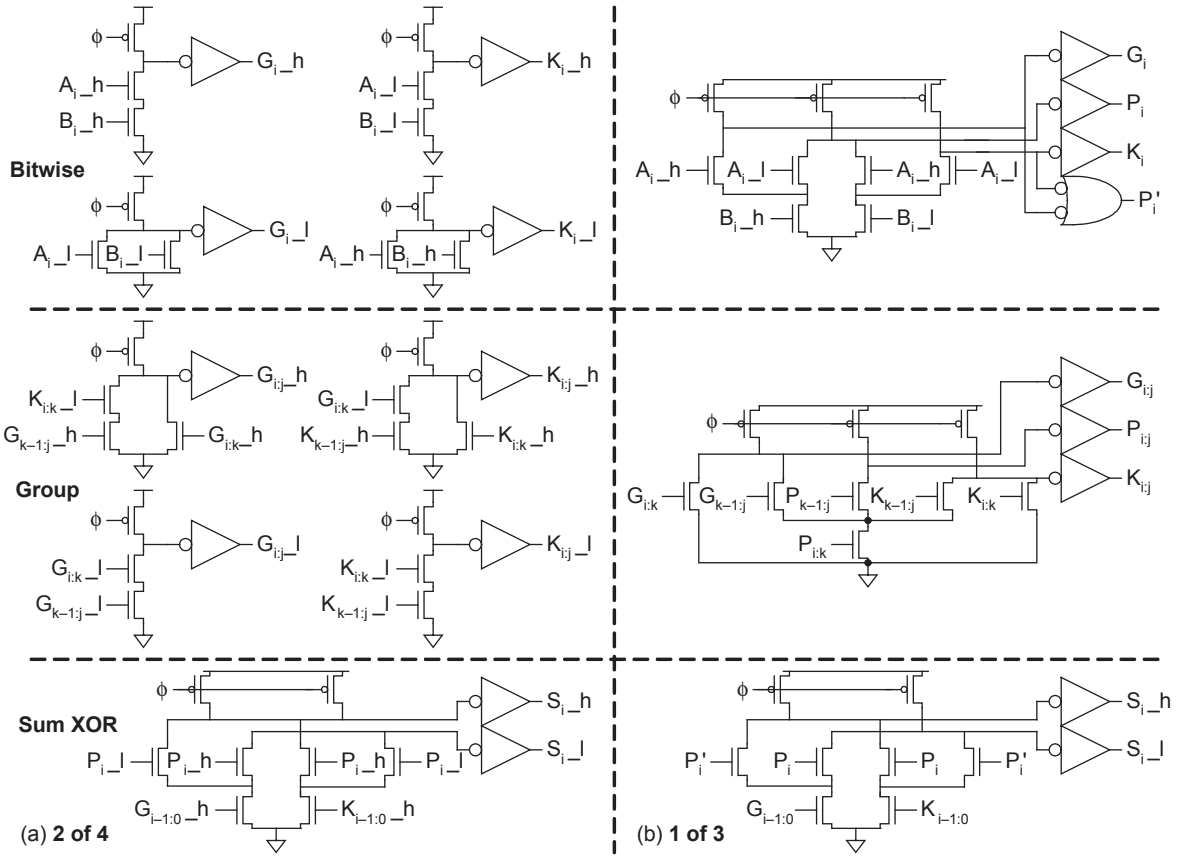


FIGURE W11.5 Domino adder circuit components

The approach gets its name because exactly one of the three signals *P*, *G*, or *K* is true for any group. The complementary propagate signal $P' = G + K$ is required for the final sum XOR, but nowhere earlier. Again, notice that the *G* and *K* functions are identical, simplifying design and layout. The group kill prefixes are the complements of the group generates ($G_{i-1:0} = \overline{K}_{i-1:0}$); this is used in EQ (W11.5) to reduce the loading on each signal in the sum XOR. The 1-of-*n* hot technique can be useful for other domino applications, such as multiplexer select signals and shifter control signals; it also reduces switching activity and power consumption. Figure W11.5(b) shows how transistors can be shared

between gates.

Manchester carry chains can also generate both polarities of carries. Figure W11.6 shows how the same type of carry chain can be used for both C_3_h and C_3_l carries as well as to find the group propagate signal using 1-of-3 hot encoding.

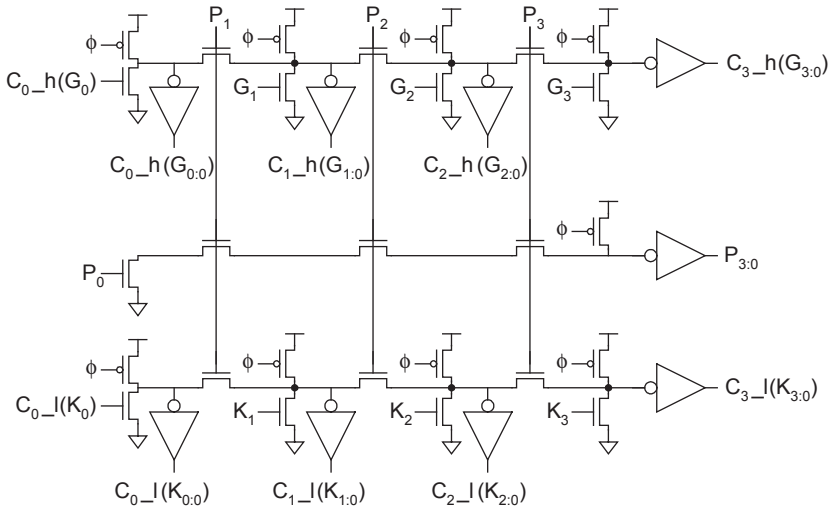


FIGURE W11.6 1-of-3 hot domino Manchester carry chain

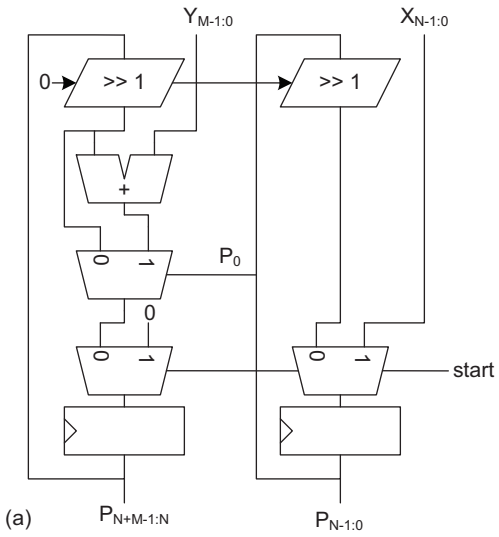
11.9.7 Serial Multiplication

Large parallel multipliers consume huge numbers of transistors. While transistor budgets have expanded to the point that this is often acceptable, designers of low-cost systems still may find serial multiplication attractive. Serial multiplication uses far less hardware, but requires multiple clock cycles to operate. Multiplication can be performed in a word-serial or bit-serial fashion.

Figure W11.7(a) shows a word-serial unsigned multiplication unit that only requires an M-bit adder and an $(M+N)$-bit loadable shift register [Patterson04]. On each step, it conditionally adds the multiplicand Y to the running product if the appropriate bit of the multiplier X is 1. It is based on the observation that on the kth step, the running product has a length of $M+k$ bits and that bits $0\ldots k-1$ of X have already been considered and are no longer necessary.

The multiplier is initialized by loading all of X into the lower portion of the shift register and a running product of zeroes into the upper portion. On step k, the running product shifts and Y is added to the most significant part if $x_k = 1$. Each shift doubles the weight at which the next partial product will be added to the running product. After N steps, the shift register will contain the final product. Figure W11.7(b) demonstrates multiplying $1100 \times 0101 = 00111100$. The vertical bar separates the running product from the remaining bits of X.

The cycle time of word-serial multiplication is set by the M-bit carry-propagate addition on each step. This CPA delay can be shortened to a CSA delay by maintaining the partial product in carry-save redundant form. The cost is doubling the number of registers to hold the redundant partial product and a final CPA to convert the redundant result into



Step	Shift Reg	Notes
	0000|0101	initialize
0a	1100|010	**1** add **1***Y
0b	01100|010	shift right
1a	01100|01	**0** add **0***Y
1b	001100|01	shift right
2a	111100|0	**1** add **1***Y
2b	0111100|0	shift right
3a	0111100|	**0** add **0***Y
3b	00111100|	shift right

(b)

FIGURE W11.7 Word-serial multiplier

a two's complement number at the end of the multiplication. Alternatively, only the upper *M* bits can be kept in redundant form, and a single full adder can convert each bit on the fly as it shifts into the lower *N* bits.

Serial multiplication can be accelerated by processing more bits of *X* on each step. For example, a radix-4 approach consumes 2 bits of *X* on each step, halving the cycle count. Booth encoding can be used to avoid having to compute 3*Y*. Booth-encoding also handles signed operands gracefully.

14.7 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].

14.7.1 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 when the following occurs:

- The vertices in the graph are the source/drain connections.
- The edges in the graph are gates of transistors that connect particular source/drain

vertices.

Two graphs, one for the pulldown network (n), and one for the pullup network (p), result. Figure W14.1(a) 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 pullup structure. Transistor *A* (*A* connected to gate) is an edge from the vertex *Y* to *I2*. The other transistors are similarly arranged in Figure W14.1(b). Note that the graphs are duals of each other because the pullup and pulldown networks are the dual of each other. The n-graph (dark lines and crosses) overlays the p-graph in Figure W14.1(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 as follows:

- Find all Euler paths that cover the graph.
- Find a p- and n-Euler path that have identical labeling (a labeling is an ordering of the gate labels on each vertex).
- 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.

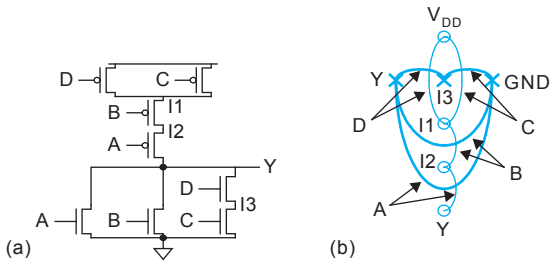


FIGURE W14.1 Circuit graphs

The original graph with a possible Euler path is shown in Figure W14.2(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 W14.2(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 W14.3(a). According to the style of layout that we have used

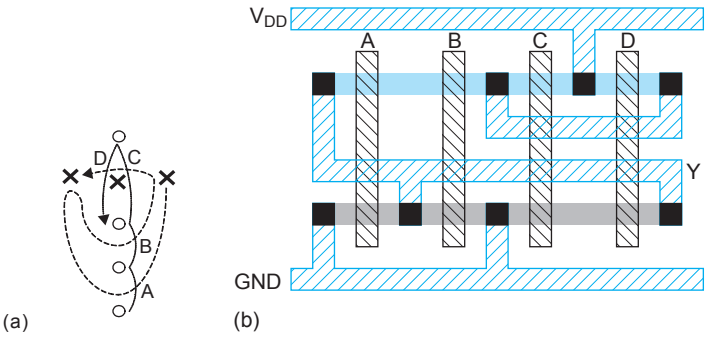


FIGURE W14.2 Stick diagram derived from Euler path

to date, two possible layouts are shown in Figure W14.3(b) and Figure W14.3(c). The layout in Figure W14.3(b) uses the single row of n- and p-diffusion with a break, while that of Figure W14.3(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.

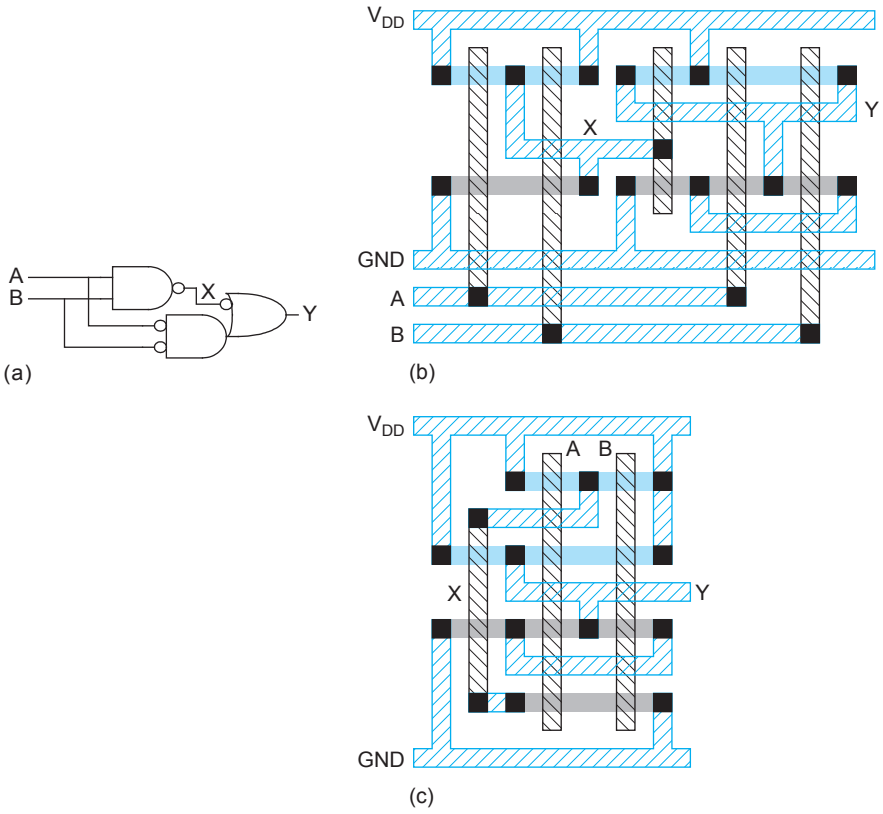


FIGURE W14.3 Broken line of diffusion and gate matrix cell layout styles

14.7.2 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:

- Complete the electrical gate design and verification before layout. Circuit changes after layout is started become schedule busters.
- 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.
- Run a vertical polysilicon line for each gate input.
- Order the polysilicon gate signals to allow the maximal connection between transistors via abutting source/drain connections. These form gate segments.
- Place n-diffusion segments close to GND and p-diffusion segments close to V_{DD}, as dictated by connectivity requirements.
- Make connections to complete the logic gate in polysilicon (for short connections between gates) or metal. Squeeze transistors together to minimize diffusion between transistors.
- Place well and substrate contacts under the supply lines in each cell.

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 W14.4 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.

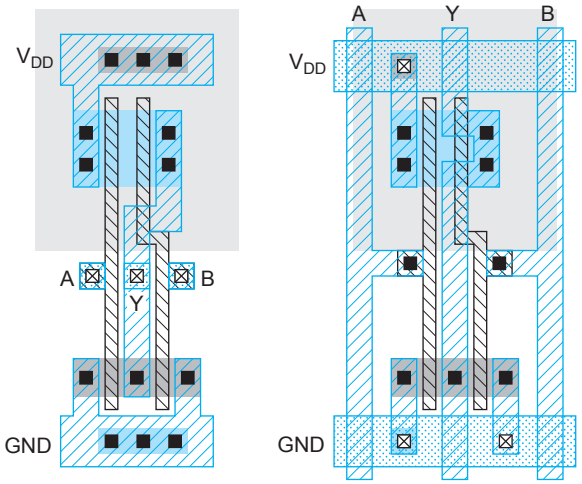


FIGURE W14.4 Standard cell metal usage

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 the following:

- 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.
- 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.
- 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.
- 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.
- Probe points should be placed on the top metal layer where they will be accessible during test (see Section 15.4).
- 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 W14.5(a) shows a transparent latch schematic. Figure W14.5(b) shows the latch layout built from simple standard cells, while Figure W14.5(c) shows an optimized layout with two thirds the area. This improvement in density is due to a number of factors, including the following:

- Better use of routing layers—routes can occur over cells
- More "merged" source/drain connections
- More use of "white space" (blank areas with no devices or connections) in sparse gates

Improvements gained by optimizing at this level over a poorly implemented standard-cell approach can be up to 100% or more in area. However, such an approach is quite labor-intensive. These days, it is only worth investing manual effort in highly repetitive and reused structures like datapaths and widely used standard cells. Implementing random control logic manually in this manner is clearly a mistake because this type of logic often changes and the manual effort has to be continually spent to keep up with the changes. With modern multilevel metallization processes and optimized standard cell libraries, the density difference between custom-designed cells and hand or algorithmically placed standard cells is minimal if the same circuits are used, because the transistor area fits under any routing. Density differences for custom circuits occur where the circuit is optimized to reduce the number of transistors (i.e., taking out buffer inverters in a latch). The point is

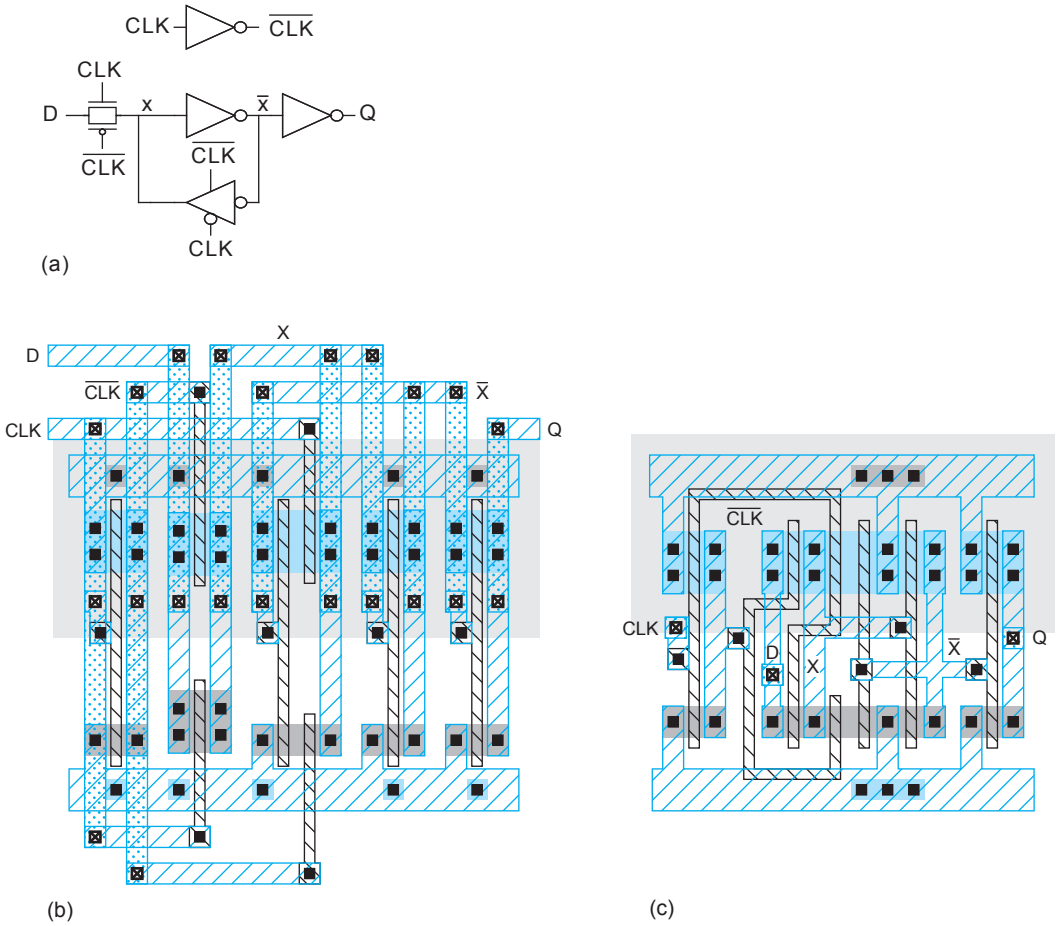


FIGURE W14.5 Transparent latch layouts

that "in the old days" there was a much greater difference between custom and even a well-implemented standard cell design than there is today (given the same circuits).

15.6.2.3 Other Scannable Elements During scan mode, the flip-flops are connected back-to-back. Clock skew can lead to hold time problems in the scan chain. These problems can be overcome by adding delay buffers on the *SI* input to flip-flops that might see large clock skews. Another approach is to use nonoverlapping clocks to ensure hold times. For example, the *Level Sensitive Scan Design* (LSSD) methodology developed at IBM uses flip-flops with two-phase nonoverlapping clocks like those shown in Figure 10.21. During scan mode, a scan clock ϕ_s is toggled in place of ϕ_2, as shown in Figure W15.1. The nonoverlapping clocks also prevent hold time problems in normal operation, but increase the sequencing overhead of the flip-flop. Alternatively, ϕ_1 and ϕ_2 can be complementary clocks, but ϕ_s can be nonoverlapping to prevent races. Figure W15.1(c) shows a conventional design using a weak feedback inverter on the master latch that can be overpowered when either the ϕ_2 or ϕ_s transmission gates are on. Figure W15.1(d) shows a design from the PowerPC 603 microprocessor using a generalized tristate feedback [Gerosa94]. Figure

W15.1(e) shows another gate-level LSSD flip-flop design [Eichelberger78]. Such a design is substantially larger and slower than a conventional pass-transistor circuit, so it is primarily of historical interest. In the IBM LSSD methodology, ϕ_s, ϕ_1, ϕ_2, and *SI* are often called *A*, *B*, *C*, and *I*, respectively.

Systems using latches can also be modified for scan. Typically, a scan input and an extra *slave scan latch* are added to convert the latch into a scannable flip-flop. Figure W15.2 shows a scannable transparent latch. During scan, the global clock is stopped low, so ϕ_1 is low and the latch is opaque. Then, a two-phase nonoverlapping scan clock ϕ_{1s} and ϕ_{2s} is toggled to march the data through the scan chain. The *SO* scan-out terminal of each latch connects to the *SI* scan-in terminal of the next latch. Figure W15.2(c) shows a faster and more compact but less robust version of the scannable latch suitable for custom datapaths [Harris01a]. Scanning one latch in each cycle is adequate to provide good observability and controllability in a system; there is no need to scan the ϕ_2 latch.

The same principle applies to pulsed latches. Figure W15.3 shows the scannable Naffziger pulsed latch used on the Itanium 2 [Naffziger02] (see also Section 10.3.3). It uses a single-phase scan clock. The global clock is stopped during scan so the pulsed latches remain opaque. The scan input overpowers the feedback node *Y* to avoid loading the critical path from *D* to *Q*. The transmission gate latch driving *SO* has a dynamic node *Z*, so ϕ_s has a limit on how long it can be high to properly retain data during scan. This is

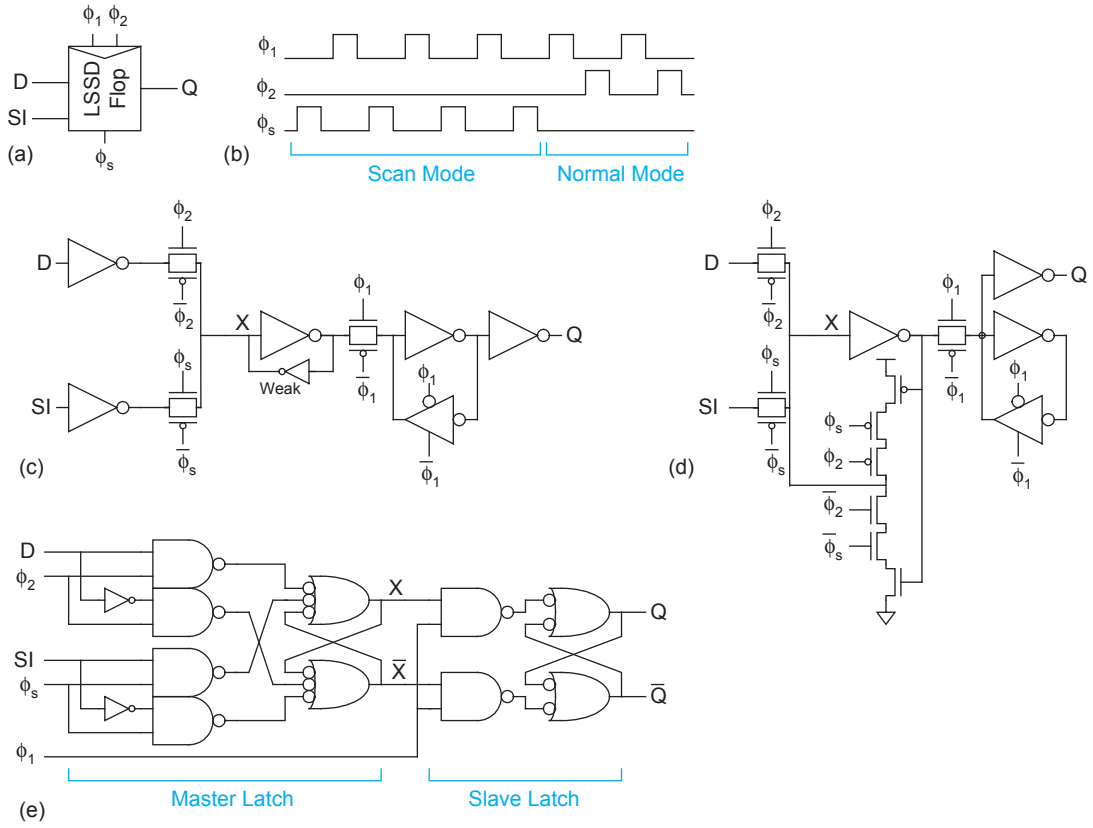


FIGURE W15.1 LSSD flip-flops

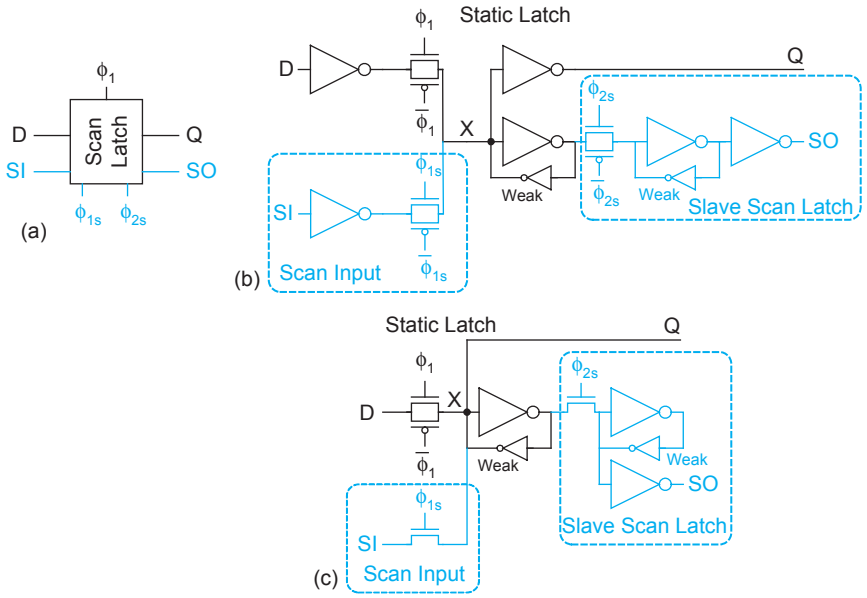


FIGURE W15.2 Scannable transparent latches

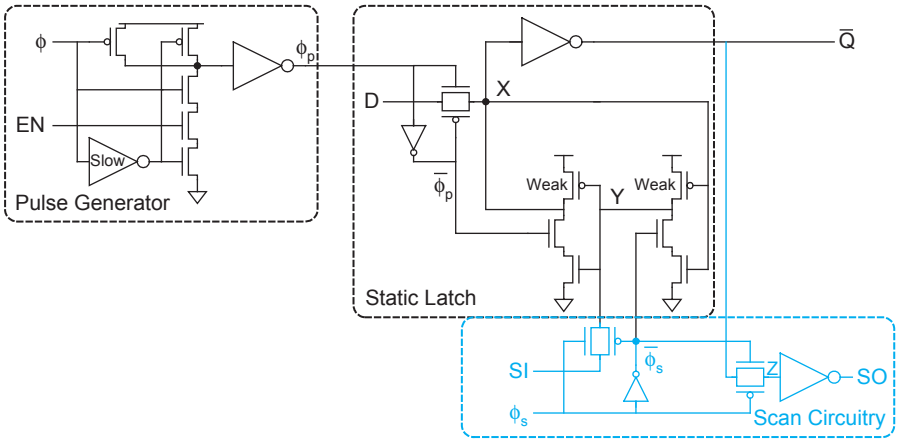


FIGURE W15.3 Scannable Naffziger pulsed latch

handled on-chip with a clock chopper that converts the external low-frequency scan clock into an on-chip ϕ_s with short pulses. The scan chain must also be checked for hold time races. Note that the *SO* transmission gate is ON during normal operation, loading the *Q* output and increasing power consumption through spurious transitions on *Z* and *SO*. Many designers would elect to use a second scan clock wire to avoid these problems.

Domino pipelines also can be scanned. Traditional domino pipelines incorporate scan into the two-phase transparent latches on the half-cycle boundaries. Skew-tolerant domino eliminates the latches and must include scan directly in the domino gates. One natural point to scan is the last gate of each cycle.

Figure W15.4(a) shows how to make the last ϕ_4 gate of each cycle in a skew-tolerant domino pipeline scannable [Harris01a]. The last dynamic gate has a full keeper and thus will retain its state when either high or low. The scan technique resembles that of a transparent latch from Figure W15.2(c). The key is to turn off both the precharge and the evaluation transistors so the output node floats and behaves like a master latch. Then a two-phase scan clock is toggled to shift data first onto the master node and then into a slave scan latch. These scan clocks are again called ϕ_{1s} and ϕ_{2s} and bear no relationship to the domino clocks ϕ_1 and ϕ_2. *gclk* is stopped low, so ϕ_4 is high and the precharge transistor is off. A special clock gater forces ϕ_{4s} low during scan to turn the evaluation transistor off. When scan is complete, *gclk* rises so the next ϕ_1 domino gate resumes normal operation. This scan approach adds a small amount of loading on the critical path through the dynamic gate. Figure W15.4(b) shows a clock gater that produces the domino phases. It uses an SR latch to stop and release ϕ_{4s} during scan, as illustrated in Figure W15.4(c). The gater also accepts an enable to stop the domino clocks when the pipeline is idle.

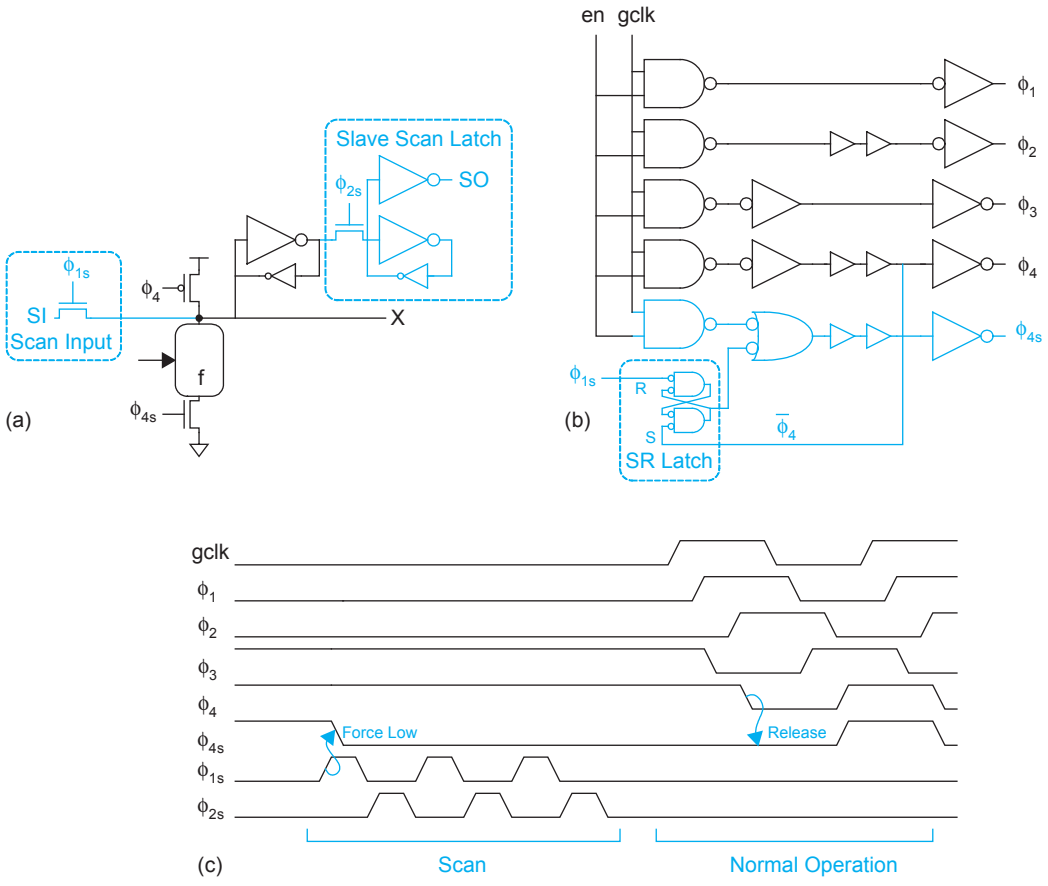


FIGURE W15.4 Itanium 2 scannable domino gate

The Itanium 2 provides domino scan in a similar fashion, but with a single-phase scan clock that is compatible with scan of the Naffziger pulsed latches [Naffziger02]. The last domino gate in each half-cycle uses a dynamic latch converter, as discussed in Section 10.5.5.2. Scan circuitry can be added to the DLC in much the same way as it is added to a latch, as shown in Figure W15.5.

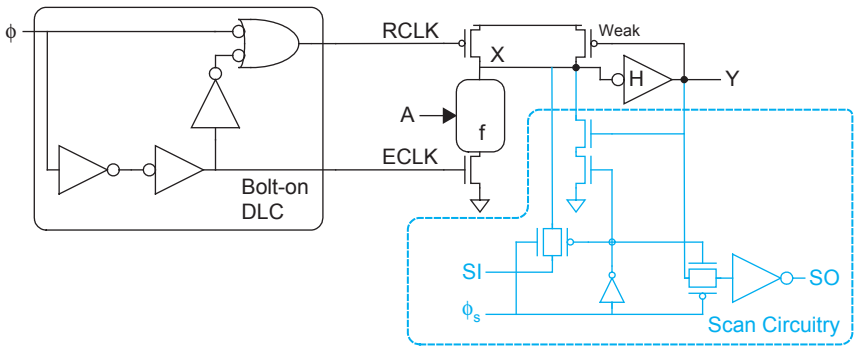


FIGURE W15.5 Scannable dynamic gate for four-phase skew-tolerant domino

Robust scan circuitry obeys a number of rules to avoid electrical failures. *SI* is locally buffered to prevent problems with directly driving diffusion inputs and overdriving feedback inside the latch. The output is also buffered so noise cannot back-drive the state node. Two-phase nonoverlapping scan clocks prevent hold-time problems, and static feedback on the state node allows low-frequency operation. All internal nodes should swing rail-to-rail. These rules can be bent to save area at the expense of greater electrical verification on the scan chain, as was done for the Itanium 2.

15.7.1 The Test Access Port (TAP)

The *Test Access Port* has four or five single-bit connections:

TCK	Test Clock	Input	Clocks tests into and out of the chip
TMS	Test Mode Select	Input	Controls test operations
TDI	Test Data In	Input	Test data into the chip
TDO	Test Data Out	Output	Test data out of the chip; driven only when TAP controller is shifting out test data
*TRST**	Test Reset Signal	Input	Optional active low signal to asynchronously reset the TAP controller if no power-up reset signal is automatically generated by the chip

When the chip is in normal mode, *TRST** and *TCK* are held low and *TMS* is held high to disable boundary scan. To prevent race conditions, inputs are sampled on the rising edge of *TCK* and outputs toggle on the falling edge.

15.7.2 The Test Logic Architecture and Test Access Port

The basic test architecture is shown in Figure W15.6. It consists of the following:

- The TAP interface pins
- A set of two or more test-data registers (DR) to collect data from the chip
- An instruction register (IR) specifying the type of test to perform
- A TAP controller, which controls the scan of bits through the instruction and test-data registers

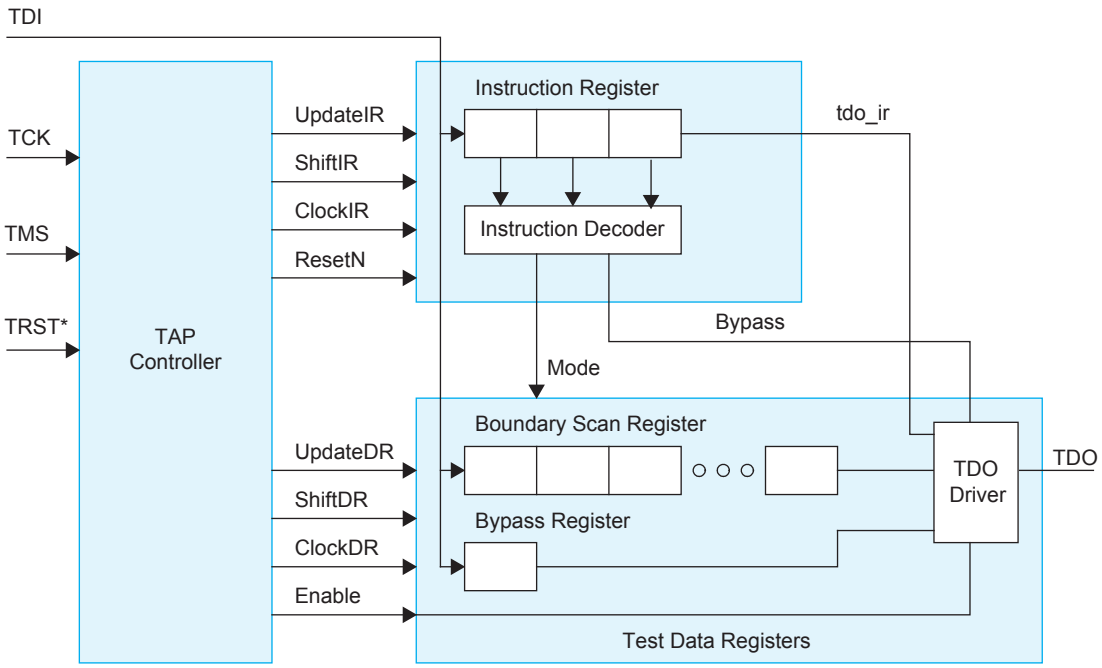


FIGURE W15.6 TAP architecture

The TAP controller is a small finite-state machine that configures the system. In one mode, it scans an instruction into the instruction register specifying what boundary scan should do. In another mode, it scans data in and out of the test-data registers. The specification requires at least two test-data registers: the boundary scan register and the bypass register. The boundary scan register is associated with all the inputs and outputs on the chip so that boundary scan can observe and control the chip I/Os. The bypass register is a single flip-flop used to accelerate testing by avoiding shifting data into the boundary scan registers of idle chips, when only a single chip on the board is being tested. Internal scan chain, BIST, or configuration registers can be treated as optional additional data registers controlled by boundary scan.

15.7.3 The TAP Controller

The TAP controller is a 16-state FSM that proceeds from state to state based on the *TCK* and *TMS* signals. It provides signals that control the test-data registers and the instruction register. These include serial shift clocks and update clocks.

The state transition diagram is shown in Figure W15.7. The TAP controller is initialized to Test-Logic-Reset on power-up by *TRST** or an internal power-up detection circuit. It moves from one state to the next on the rising edge of *TCK* based on the value of *TMS*.

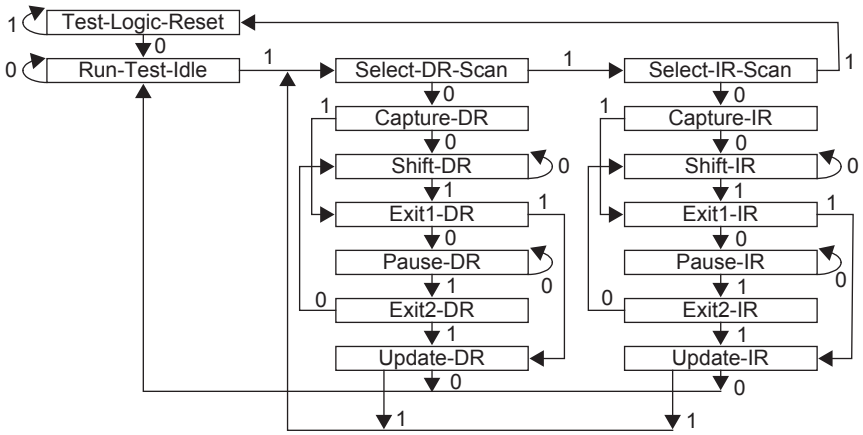


FIGURE W15.7 TAP controller state diagram

A typical test sequence will involve clocking *TCK* at some rate and setting *TRST** to 0 for a few cycles and then returning this signal to 1 to reset the TAP controller state machine. *TMS* is then toggled to traverse the state machine for whatever operation is required. These operations include serially loading an instruction register or serially loading or reading data registers that are used to test the chip. A variety of these operations will be described as this section unfolds.

The following Verilog code implements the TAP controller. The *TRST** is named `trstn`. Note that the controller produces gate clocks to control the data and instruction registers at the appropriate times.

```
// TAP Controller States
`define TEST_LOGIC_RESET 4'b1111
`define RUN_TEST_IDLE    4'b1100
`define SELECT_DR_SCAN   4'b0111
`define CAPTURE_DR       4'b0110
`define SHIFT_DR         4'b0010
`define EXIT1_DR         4'b0001
`define PAUSE_DR         4'b0011
`define EXIT2_DR         4'b0000
`define UPDATE_DR        4'b0101
`define SELECT_IR_SCAN   4'b0100
`define CAPTURE_IR       4'b1110
`define SHIFT_IR         4'b1010
`define EXIT1_IR         4'b1001
```

```
`define PAUSE_IR         4'b1011
`define EXIT2_IR         4'b1000
`define UPDATE_IR        4'b1101

module tapcontroller(input      tms, tck, trstn,
                     output reg ShiftIR, ShiftDR,
                     output     ClockIR, ClockDR,
                     output     UpdateIR, UpdateDR,
                     output reg Resetn, Enable);

   reg [3:0] state;

   // next state logic
   always @(posedge tck, negedge trstn)
      if (~trstn) state = `TEST_LOGIC_RESET;
      else case (state)
        `TEST_LOGIC_RESET:state = (tms) ? state : `RUN_TEST_IDLE;
        `RUN_TEST_IDLE:   state = (tms) ? `SELECT_DR_SCAN : state;
        `SELECT_DR_SCAN:  state = (tms) ? `SELECT_IR_SCAN : `CAPTURE_DR;
        `CAPTURE_DR:      state = (tms) ? `EXIT1_DR : `SHIFT_DR;
        `SHIFT_DR:        state = (tms) ? `EXIT1_DR : state;
        `EXIT1_DR:        state = (tms) ? `UPDATE_DR : `PAUSE_DR;
        `PAUSE_DR:        state = (tms) ? `EXIT2_DR : state;
        `EXIT2_DR:        state = (tms) ? `UPDATE_DR : `SHIFT_DR;
        `UPDATE_DR:       state = (tms) ? `SELECT_DR_SCAN : `RUN_TEST_IDLE;
        `SELECT_IR_SCAN:  state = (tms) ? `TEST_LOGIC_RESET : `CAPTURE_IR;
        `CAPTURE_IR:      state = (tms) ? `EXIT1_IR : `SHIFT_IR;
        `SHIFT_IR:        state = (tms) ? `EXIT1_IR : state;
        `EXIT1_IR:        state = (tms) ? `UPDATE_IR : `PAUSE_IR;
        `PAUSE_IR:        state = (tms) ? `EXIT2_IR : state;
        `EXIT2_IR:        state = (tms) ? `UPDATE_IR : `SHIFT_IR;
        `UPDATE_IR:       state = (tms) ? `SELECT_DR_SCAN : `RUN_TEST_IDLE;
      endcase

   // Clock registers on rising edge of tck at end of state
   // otherwise idle clock high
   assign ClockIR  = tck | ~((state == `CAPTURE_IR) | (state == `SHIFT_IR));
   assign ClockDR  = tck | ~((state == `CAPTURE_DR) | (state == `SHIFT_DR));

   // Update registers on falling edge of tck
   assign UpdateIR = ~tck & (state == `UPDATE_IR);
   assign UpdateDR = ~tck & (state == `UPDATE_DR);

   // Change control signals on falling edge of tck
   always @(negedge tck, negedge trstn)
     if (~trstn) begin
        ShiftIR <= 0;
        ShiftDR <= 0;
        Resetn  <= 0;
        Enable  <= 0;
     end else begin
        ShiftIR <= (state == `SHIFT_IR);
        ShiftDR <= (state == `SHIFT_DR);
        Resetn  <= ~(state == `TEST_LOGIC_RESET);
        Enable  <= (state == `SHIFT_IR) | (state == `SHIFT_DR);
     end
endmodule
```

15.7.4 The Instruction Register

The instruction register has to be at least 2 bits long. Recall that boundary scan requires at least two data registers. The instruction register specifies which data register will be placed in the scan chain when the DR is selected. It also determines from where the DR will load its value in the Capture-DR state, and whether the values will be driven to output pads or core logic. The following three instructions are required to be supported:

- BYPASS—This instruction places the bypass register in the DR chain so that the path from *TDI* to *TDO* involves only a single flip-flop. This allows specific chips to be tested in a serial scan chain without having to shift through the lengthy shift register stages in all the chips. This instruction is represented with all 1s in the IR.
- SAMPLE/PRELOAD—This instruction places the boundary scan registers (i.e., at the chip's I/O pins) in the DR chain. In the Capture-DR state, it copies the chip's I/O values into the DRs. They can then be scanned out in successive Shift-DR states. New values are shifted into the DRs, but not driven onto the I/O pins yet.
- EXTEST—This instruction allows for the testing of off-chip circuitry. It is similar to SAMPLE/PRELOAD, but also drives the values from the DRs onto the output pads. By driving a known pattern onto the outputs of some chips and checking for that pattern at the input of other chips, the integrity of connections between chips can be verified.

In addition to these instructions, the following are also recommended (others can be defined as needed):

- INTEST— This instruction allows for single-step testing of internal circuitry via the boundary scan registers. It is similar to EXTEST, but also drives the chip core with signals from the DRs rather than from the input pads.
- RUNBIST— This instruction is used to activate internal self-testing procedures within a chip.

Note that the instruction encodings are not part of the specification (except that BYPASS is all 1s). The component designer must document what encodings were selected for each instruction.

A typical IR bit is shown in Figure W15.8. Observe that it contains two flip-flops. The ClockIR flip-flops of each bit are connected to form a shift register. They are loaded with a constant value from the Data input in the Capture-IR state, and then are shifted out in the Shift-IR state while new values are shifted in. The constant value is user-defined, but must have a 01 pattern in the least significant two bits so that the integrity of the scan chain can be verified. In the Update-IR state, the contents of the shift register are copied in parallel to the IR output to load the entire instruction at once. This prevents the IR from momentarily having illegal values while new instructions are shifted in. On reset, the IR should be asynchronously loaded with an innocuous instruction such as BYPASS that does not interfere with the normal behavior of the core logic.

A minimal implementation of a 3-bit control register is shown below. Notice the instruction encoding definitions. This implements the six registers required for a 3-bit instruction. The instruction is decoded to produce `mode_in`, `mode_out`, and `bypass` signals to control the data registers, as will be discussed in the next sections.

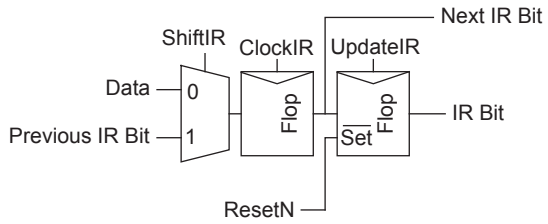


FIGURE W15.8 Instruction bit implementation

```
// Instructions
`define BYPASS         3'b111
`define SAMPLE_PRELOAD 3'b101
`define EXTEST         3'b110
`define NOP            3'b001
`define INTEST         3'b100

module inst_reg(input  tdi,
                input  Resetn, ClockIR, UpdateIR, ShiftIR,
                output tdo_ir, mode_in, mode_out, bypass);

   reg [2:0] shiftreg, instreg;

   always @(posedge ClockIR)
      shiftreg <= ShiftIR ? {tdi, shiftreg[2:1]} : `NOP;
   always @(posedge UpdateIR, negedge Resetn)
      if (~Resetn) instreg <= `BYPASS;
      else instreg <= shiftreg;

   assign tdo_ir = shiftreg[0];
   assign bypass = (instreg == `BYPASS);
   assign mode_in = (instreg == `INTEST);
   assign mode_out = (instreg == `INTEST) || (instreg == `EXTEST);
endmodule
```

15.7.5 Test Data Registers

The test data registers are used to set the inputs of modules to be tested and collect the results of running tests. The simplest data register configuration consists of a boundary scan register (passing through all I/O pads) and a bypass register (1-bit long). Figure W15.9 shows a generalized view of the data registers in which an internal data register has been added. This register might represent the scan chain within the chip or a BILBO signature register. Thus, boundary scan elegantly incorporates other built-in test structures. A multiplexer under the control of the TAP controller selects which data register is routed to the *TDO* pin. When internal data registers are added, the IR decoder must produce extra control signals to select which one is in the DR chain for a particular instruction.

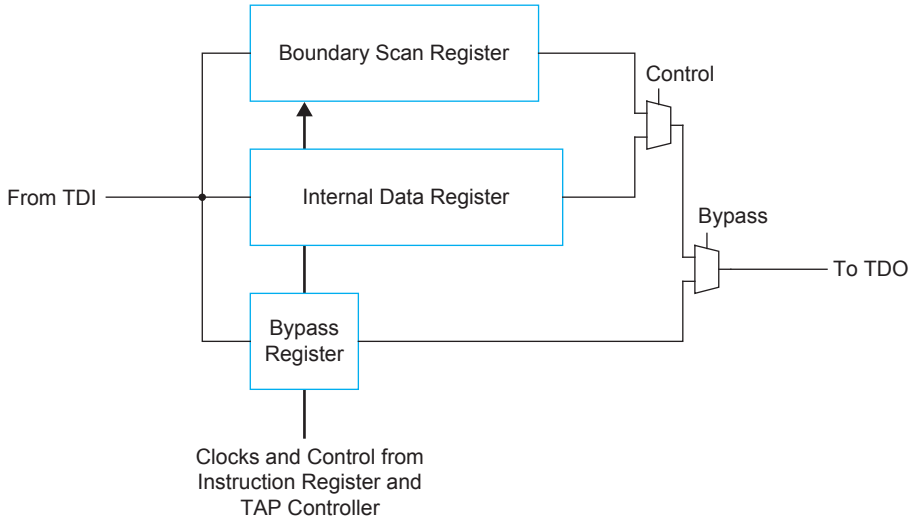


FIGURE W15.9 Test data registers

15.7.5.1 Boundary Scan Register The boundary scan register connects to all of the I/O circuitry. Like the instruction register, it internally consists of a shift register for the scan chain and an additional bank of flip-flops to update the outputs in parallel. An extra multiplexer on the output allows the boundary scan register to override the normal path through the I/O pad so it can observe and control inputs and outputs. The schematic and symbol for a single bit of the boundary scan register are shown in Figure W15.10.

The boundary scan register can be configured as an input pad or output pad, as shown in Figure W15.11(a and b). As an input, the register receives `DataIn` from the pad and sends `Qout` to the core logic in the chip. As an output, the register receives `DataIn` from the core logic and drives `Qout` to a pad. Tristate and bidirectional pads use two or three boundary scan register cells, as shown in Figure W15.11(c and d).

The `Mode` signal determines whether `Qout` should be taken from `DataIn` or the boundary scan register. Separate `mode_in` and `mode_out` signals are used for input and output pads so they can be controlled separately. In normal chip operation, both `mode` signals are 0, so the boundary scan registers are ignored. For the `EXTEST` instruction,

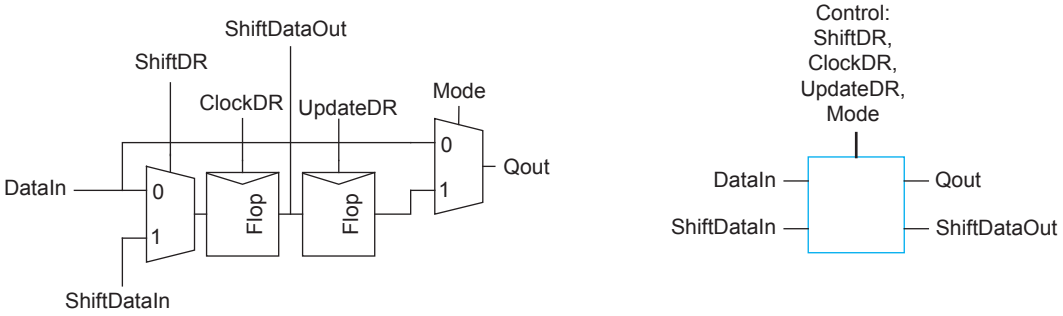


FIGURE W15.10 Boundary scan register bit

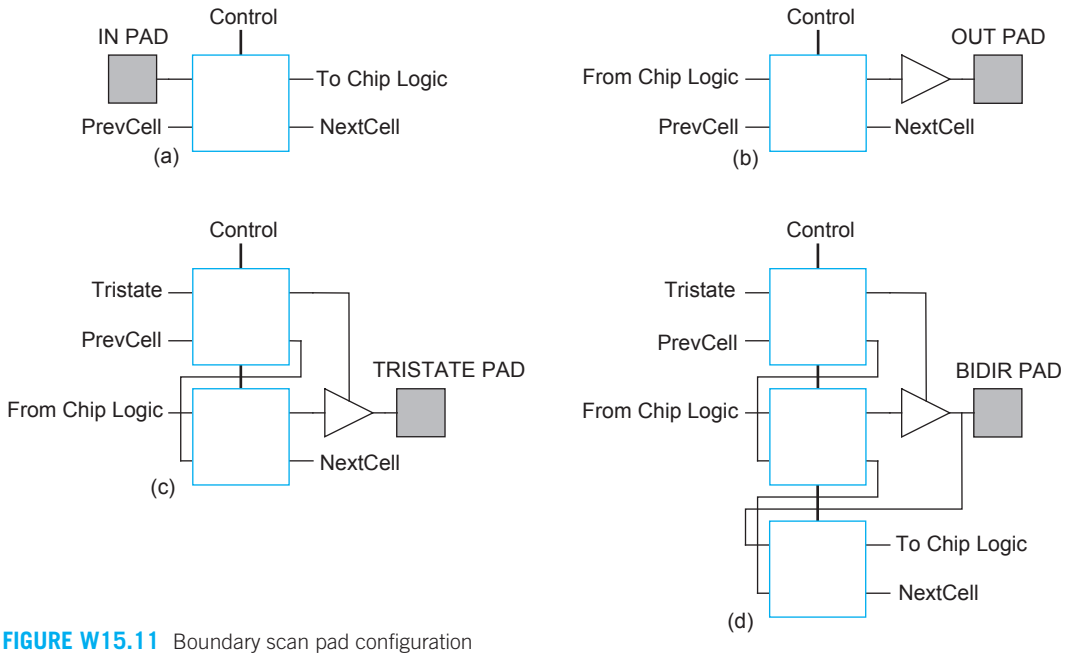


FIGURE W15.11 Boundary scan pad configuration

`mode_out` = 1, so the outputs can be controlled by the boundary scan registers. For `INTEST` or `RUNBIST` instructions, `mode_in` and `mode_out` are both 1, so the core logic receives its inputs from the boundary scan registers and the outputs are also driven to known safe values by the boundary scan registers.

15.7.5.2 Bypass Register When executing the `BYPASS` instruction, the single-bit Bypass register is connected between *TDI* and *TDO*. It consists of a single flip-flop that is cleared during Capture-DR, and then scanned during Shift-DR, as shown in Figure W15.12.

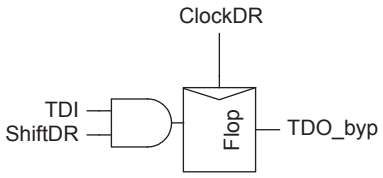


FIGURE W15.12 Bypass register

15.7.5.3 *TDO* Driver The *TDO* pin shifts out the least significant bit of the IR during Shift-IR, or the least significant bit of one of the data registers during Shift-DR, depending on which instruction is active. The IEEE boundary scan specification requires that *TDO* change on the falling edge of *TCK*, and be tristated except during the Shift states. This prevents race conditions when the value is clocked into the next chip in the rising edge of *TCK*, and allows multiple chips to be connected in parallel with their *TDO* pins tied together to reduce the length of the boundary scan chain.

Figure W15.13 shows a possible implementation of the *TDO* driver. The multiplexers choose among the possible shift registers including the instruction register, boundary scan register, and bypass register. Additional multiplexers would be used if more data registers were included. A flip-flop or latch delays the *TDO* signal until the falling edge of *TCK*. The tristate drives *TDO* during Shift-IR or Shift-DR.

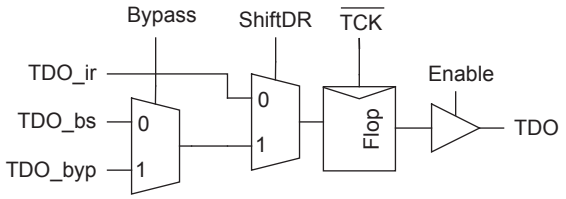


FIGURE W15.13 *TDO* driver

15.7.5.4 Complete Test Data Register Logic The Verilog code below describes the complete Test Data Register for a chip with four inputs `a`[3:0] and four outputs `y`[3:0]. The four input and four output boundary scan register bits are collected into a single 8-bit shift register. `mode_in` serves the four most significant bits connected to the inputs, while `mode_out` serves the four least significant bits connected to the outputs.

```
module data_reg(input  [3:0] a, fromlogic,
                input        tck, tdi, tdo_ir,
                input        ClockDR, UpdateDR, ShiftDR, Enable,
                input        mode_in, mode_out, bypass,
                output [3:0] y, tologic,
                output       tdo);

   reg  [7:0] shiftreg, datareg;
   wire       tdo_selected;
   reg        tdo_byp, tdo_delayed;

   // Boundary scan registers
   // four input registers and four output registers connected in 8-bit chain
   always @(posedge ClockDR)
      shiftreg <= ShiftDR ? {tdi, shiftreg[7:1]} : {a, fromlogic};
   always @(posedge UpdateDR)
      datareg <= shiftreg;
   assign tologic = mode_in ? datareg[7:4] : a;
   assign y = mode_out ? datareg[3:0] : fromlogic;

   // Bypass register
   always @(posedge ClockDR)
      tdo_byp <= tdi & ShiftDR;

   // tdo output driver
   // select appropriate register to shift out, delay to negative edge of tck
   assign tdo_selected = ShiftDR ? (bypass ? tdo_byp : shiftreg[0]) : tdo_ir;
   always @(negedge tck)
      tdo_delayed <= tdo_selected;
   assign tdo = Enable ? tdo_delayed : 1'bz;
endmodule
```

15.7.6 Summary

Figure W15.14 shows a complete implementation of boundary scan for a chip with four inputs and four outputs. It consists of the TAP controller state machine and state decoder, a 3-bit instruction register with instruction decode, the bypass register, four boundary scan input pads, and four boundary scan output pads. The other pads comprise the test access port. The boundary scan register control signals (UpdateDR, ClockDR, ShiftDR, `mode_in`, and `mode_out`) are shown as the Control bus.

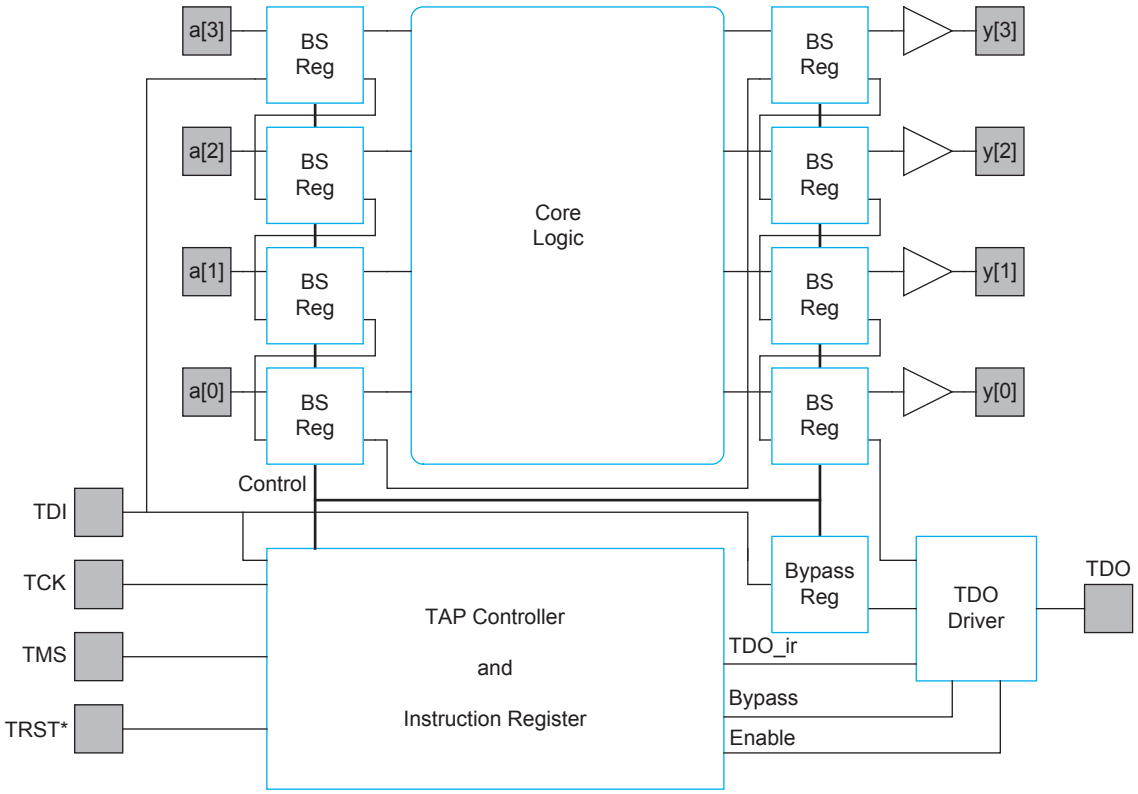


FIGURE W15.14 Complete boundary scan implementation

The Verilog for this design follows:

```
module core(input  [3:0] tologic,
            output [3:0] fromlogic);

   // a silly chip logic function
   assign fromlogic = {&tologic, |tologic, ^tologic, ~tologic[0]};
endmodule

module top(input        tck, tms, tdi, trstn,
           input  [3:0] a,
           output       tdo,
           output [3:0] y);
```

```
    wire [3:0] tologic, fromlogic;
    wire       UpdateIR, ShiftIR, ClockIR;
    wire       UpdateDR, ShiftDR, ClockDR;
    wire       Resetn, Enable;
    wire       mode_in, mode_out, bypass;
    wire       tdo_ir;

    // Core Logic
    core core(tologic, fromlogic);

    // TAP Controller
    tapcontroller tc(tms, tck, trstn, ShiftIR, ShiftDR, ClockIR, ClockDR,
                     UpdateIR, UpdateDR, Resetn, Enable);

    // Instruction register
    inst_reg ir(tdi, Resetn, ClockIR, UpdateIR, ShiftIR,
                tdo_ir, mode_in, mode_out, bypass);

    // Test data registers
    data_reg dr(a, fromlogic, tck, tdi, tdo_ir,
                ClockDR, UpdateDR, ShiftDR, Enable, mode_in, mode_out, bypass,
                y, tologic, tdo);
endmodule
```

Boundary scan testing typically begins with the SAMPLE/PRELOAD instruction. Then, a data value is preloaded into the boundary scan registers. Next, the EXTEST or INTEST instruction is applied to activate the loaded value. Subsequent data values are shifted into the boundary scan registers and the results of the tests are shifted out.

Figure W15.15 shows waveforms for this operation. The TAP controller is initially reset. At this point, the core logic operates normally with an input pattern of 0000 and an

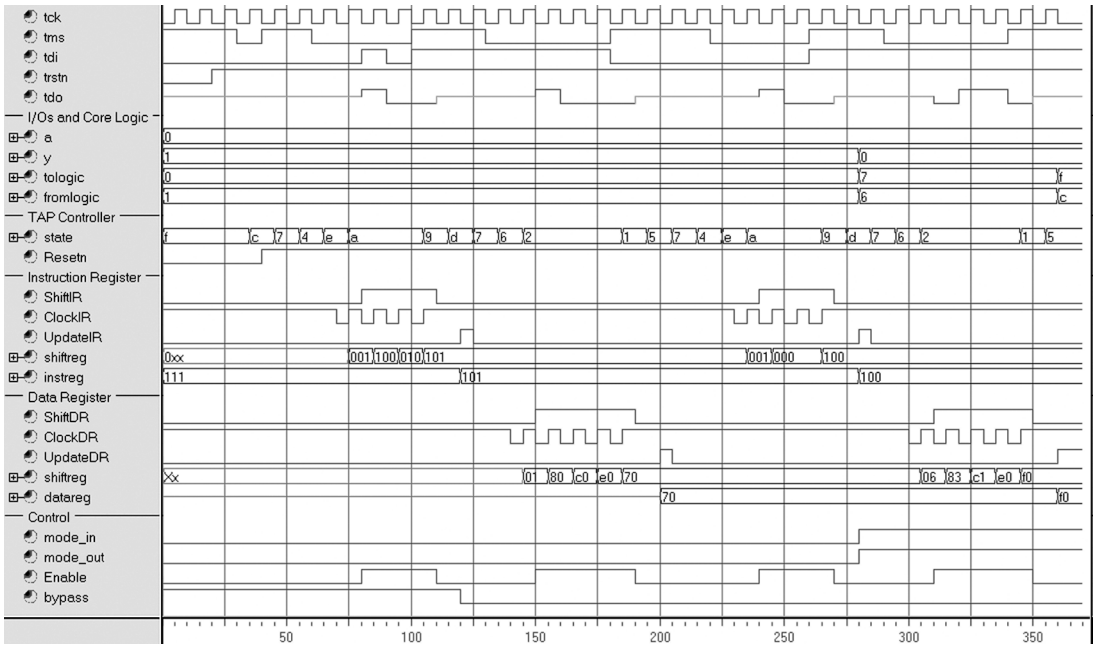


FIGURE W15.15 Boundary scan example waveforms

output pattern of 0001. Then the IR is loaded with 101 (SAMPLE/PRELOAD). The data pattern 0111 is shifted in. The IR is loaded with 1000 (INTEST). This sends the 0111 pattern to the core logic, producing an output pattern of 0110. Finally, the data pattern 1111 is shifted in and the old output 0110 is shifted out. Because the INTEST is still active, the 1111 is applied to the core, producing a new output of 1100.

Boundary scan is in widespread use in chips today. It provides a uniform interface to single- and multiple-chip testing and circuit-board testing.