Sequential Circuit Design

7

7.1 Introduction

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

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

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

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

The choice of sequencing strategy is intimately tied to the design flow that is being used by an organization. Thus, it is important before departing on a design direction to ensure that all phases of design capture, synthesis, and verification can be accommodated. This includes such aspects as cell libraries (Are the latch or flip-flop circuits and models available?); tools such as timing analyzers (Can timing closure be achieved easily?); and automatic test generation (Can self-test elements be inserted easily?).

7.2 Sequencing Static Circuits

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

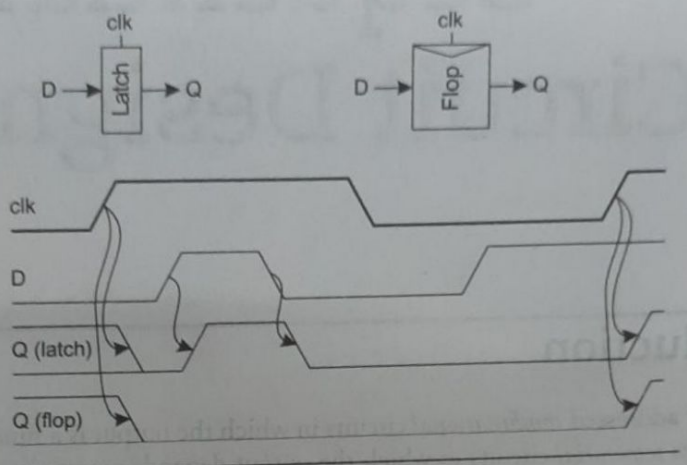

FIG 7.1 Latches and flip-flops

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

7.2.1 Sequencing Methods

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

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

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

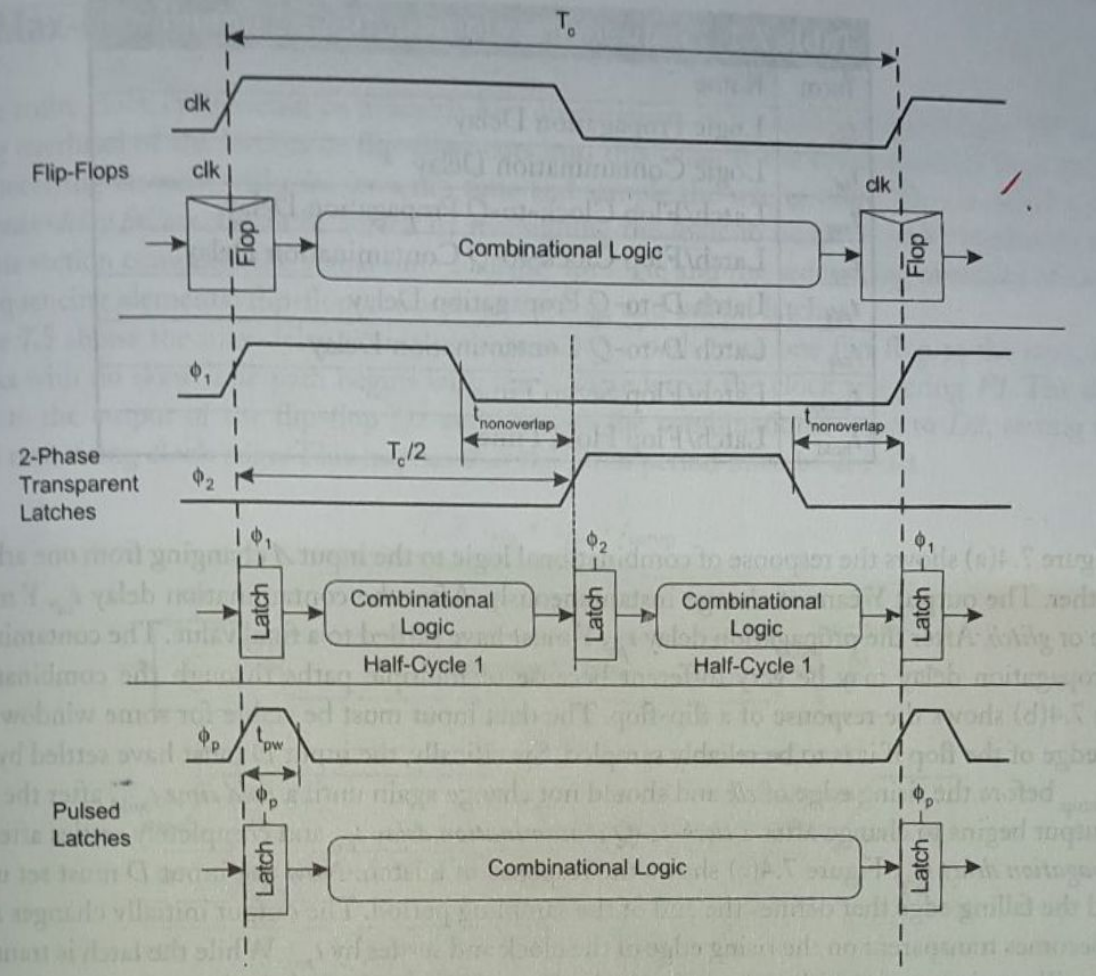

FIG 7.2 Static sequencing methods

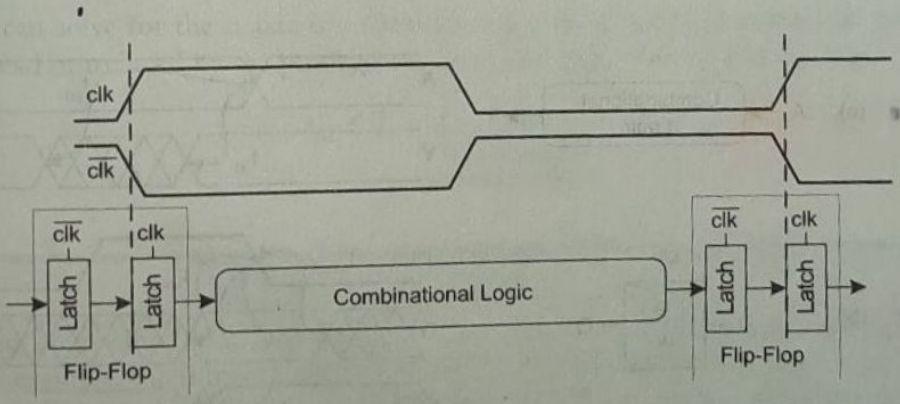

FIG 7.3 Flip-flop viewed as back-to-back latch pair

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

Table 7.1 Sequencing element timing notation

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

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

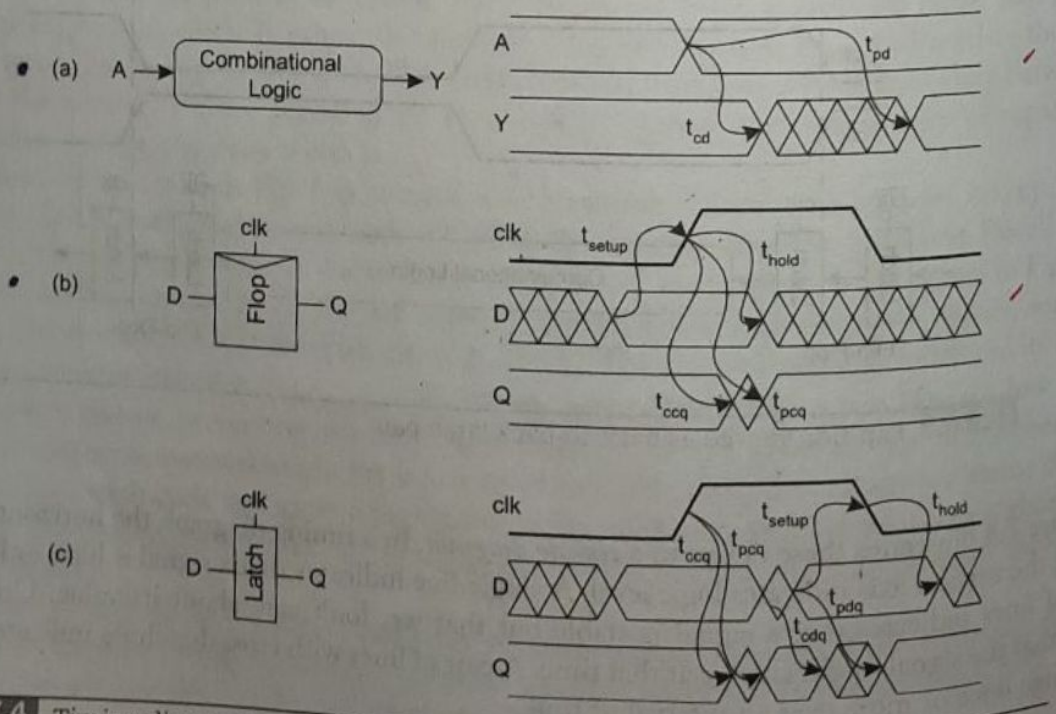

FIG 7.4 Timing diagrams

7.2.2 Max-Delay Constraints

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

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

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

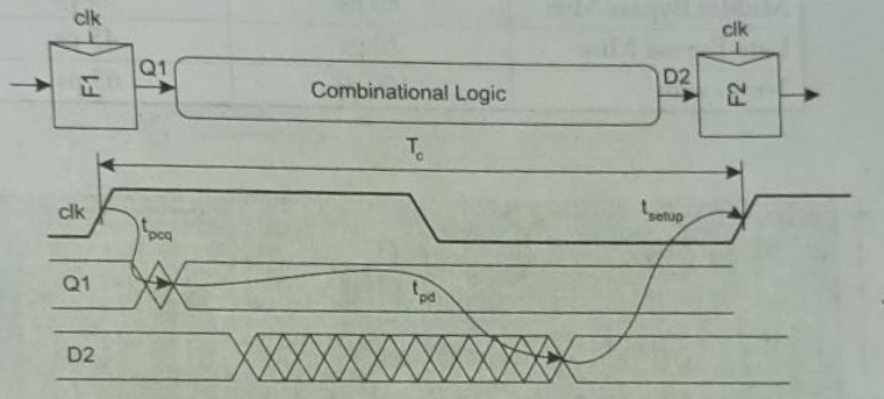

FIG 7.5 Flip-flop max-delay constraint

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

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

7.2.3 Min-delay Constraints

Ideally, sequencing elements can be placed back to back without intervening combinational logic and still function correctly. For example, a pipeline can use back-to-back registers to sequence along an instruction opcode without modifying it. However, if the hold time is large and the contamination delay is small, data can incorrectly propagate through two successive elements on one clock edge, corrupting the state of the system. This is called a *race condition*, *hold time failure*, or *min-delay failure*. It can only be fixed by redesigning the logic, not by slowing the clock. Therefore, designers should be very conservative in avoiding such failures because modifying and refabricating a chip is very expensive and time-consuming.

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

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

Example

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

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

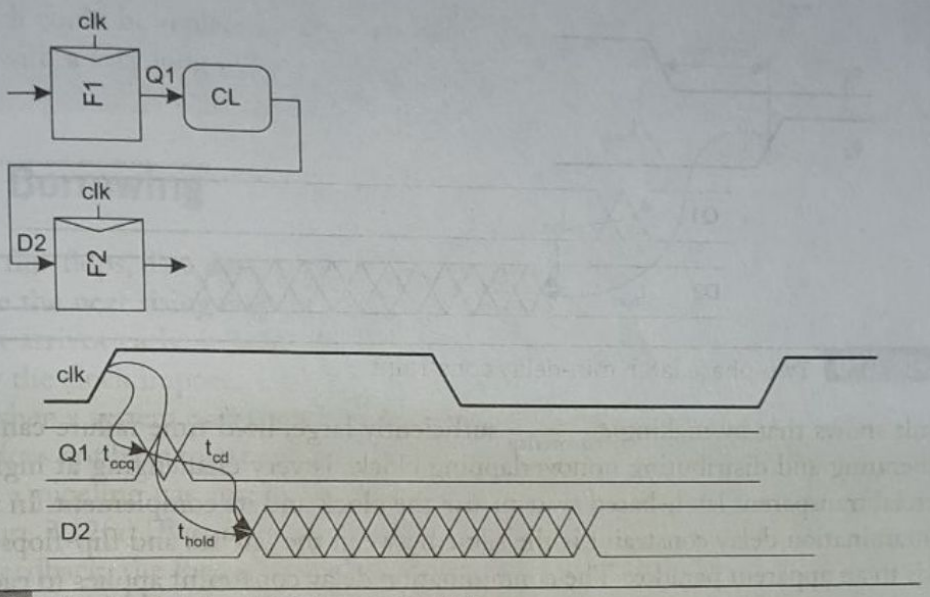

FIG 7.9 Flip-flop latch min-delay constraint

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

7.3.1 Conventional CMOS Latches

Figure 7.17(a) shows a very simple transparent latch built from a single transistor. It is compact and fast but suffers four limitations. The output does not swing from *rail-to-rail* (i.e., from GND to V_{DD}); it never rises above $V_{DD} - V_t$. The output is also *dynamic*; in other words, the output floats when the latch is opaque. If it floats long enough, it can be disturbed by leakage (see Section 4.7.8.3). *D* drives the *diffusion input* of a pass transistor directly, leading to potential noise issues (see Section 4.7.8.9) and making the delay harder to model with static timing analyzers. Finally, the state node is *exposed*, so noise on the output can corrupt the state. The remainder of the figures illustrate improved latches using more transistors to achieve more robust operation.

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

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

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

gate is too weak or far away. With the input inverter, the latch can be viewed as a cross between the designs shown in (g) and (i). Some latches add one more inverter to provide both true and complementary outputs.

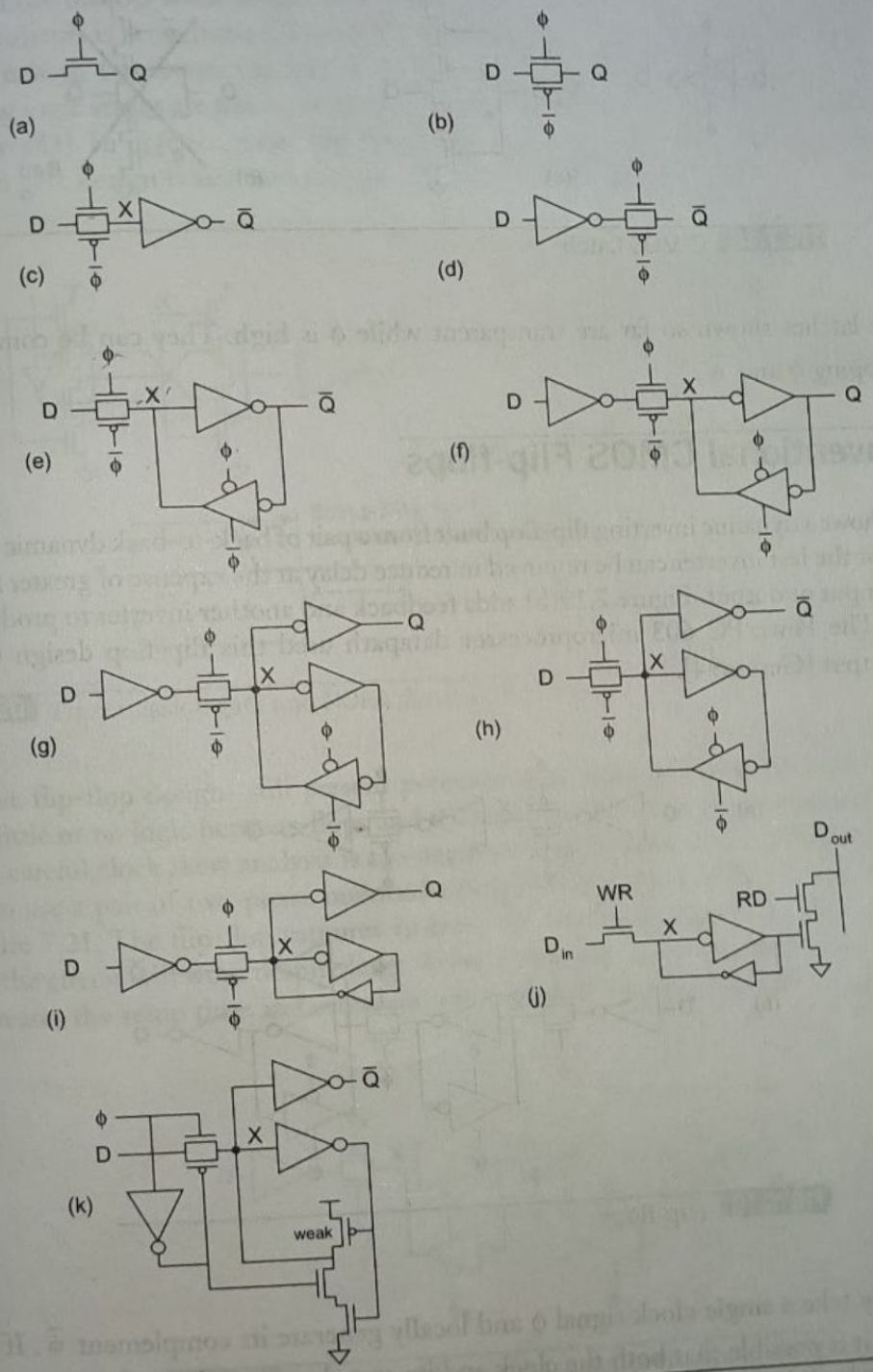

FIG 7.17 Transparent latches

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

7.3.2 Conventional CMOS Flip-flops

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

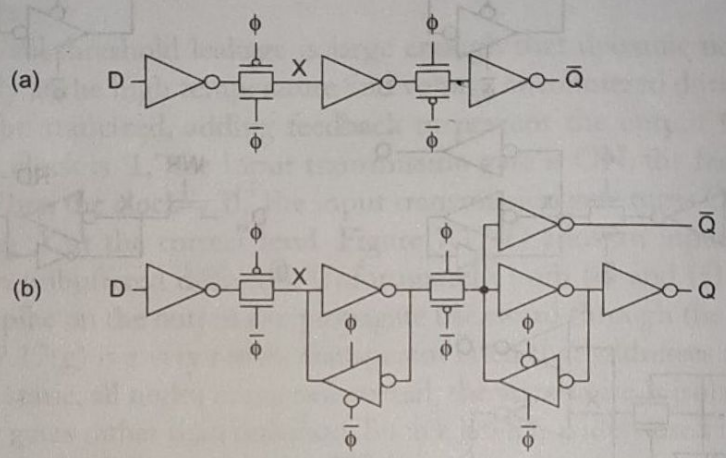

FIG 7.19 Flip-flops

Flip-flops usually take a single clock signal ϕ and locally generate its complement $\overline{\phi}$. If the clock rise/fall time is very slow, it is possible that both the clock and its complement will simultaneously be at intermediate voltages, making both latches transparent and increasing the flip-flop hold time. In ASIC standard cell libraries (such as the Artisan library), the clock is both complemented and buffered in the flip-flop cell to sharpen up the edge rates at the expense of more inverters and clock loading.

Recall that the flip-flop also has a potential internal race condition between the two latches. This race can be exacerbated by skew between the clock and its complement caused by the delay of the inverter. Figure 7.20(a) redraws Figure 7.19(a) with a built-in clock inverter. When ϕ falls, both the clock and its complement are momentarily low as shown in Figure 7.20(b), turning on the clocked pMOS transistors in both

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

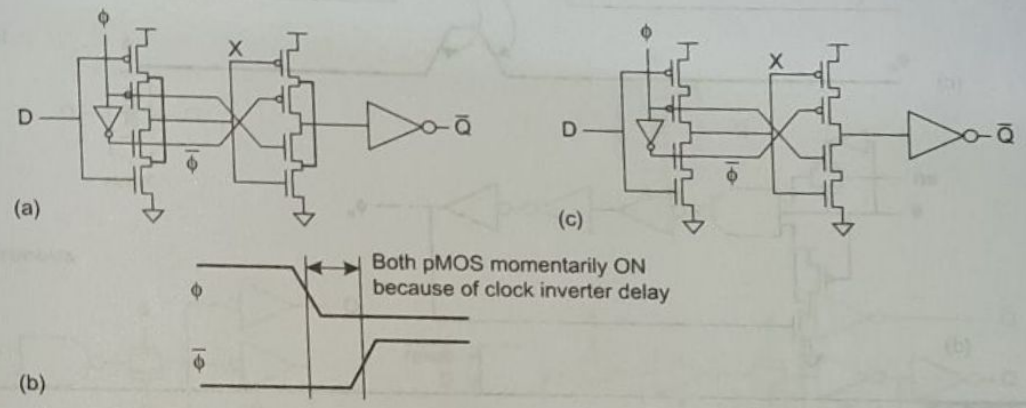

FIG 7.20 Transmission gate and NORA dynamic flip-flops

All of these flip-flop designs still present potential min-delay problems between flip-flops, especially when there is little or no logic between flops and the clock skew is large or poorly analyzed. For VLSI class projects where careful clock skew analysis is too much work and performance is less important, a reasonable alternative is to use a pair of two-phase nonoverlapping clocks instead of the clock and its complement, as shown in Figure 7.21. The flip-flop captures its input on the rising edge of ϕ_1. By making the nonoverlap large enough, the circuit will work despite large skews. However, the nonoverlap time is not used by logic, so it directly increases the setup time and sequencing overhead of the flip-flop (see Exercise 7.8).

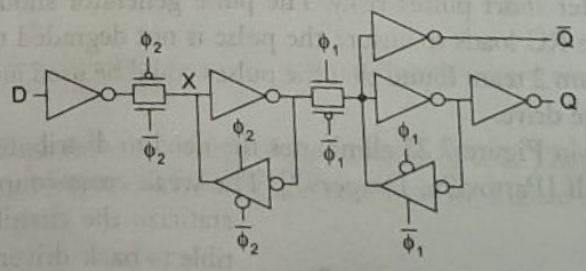

FIG 7.21 Flip-flop with two-phase nonoverlapping clocks

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

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

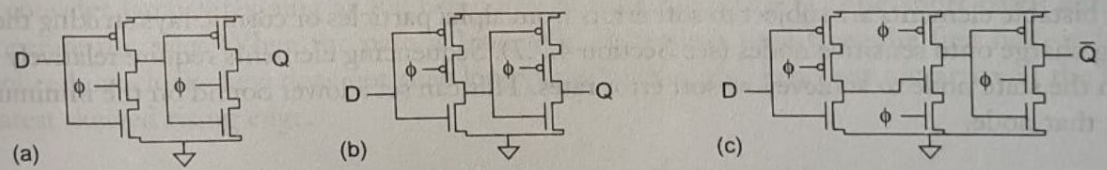

FIG 7.30 TSPC latches and flip-flops

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

11.5.2 Clock System Architechture

Figure 11.23 shows an overview of a typical clock subsystem. The chip receives an external clock signal through the I/O pads. The clock generation unit may include a *phase-locked loop* (PLL) or *delay-locked loop* (DLL) to adjust the frequency or phase of the global clock, as shall be discussed in Section 11.5.3. This global clock is then distributed across the chip to points near all of the clocked elements. The clock distribution network must be carefully designed to minimize clock skew. Local clock gaters receive this global clock and drive the physical clock signals along short wires to small groups of clocked elements.

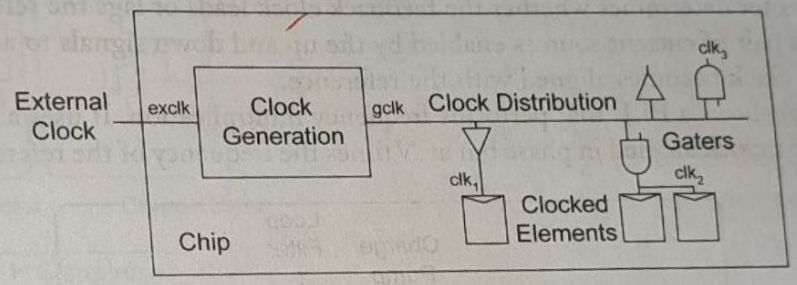

FIG 11.23 Clock subsystem

11.5.3 Global Clock Generation

The global clock generator receives an external clock signal and produces the global clock that will be distributed across the die. In this simplest case, the clock generator is simply a buffer to drive the large capacitance of the clock distribution network. However, the input pad, buffering, distribution network, and gaters have significant delay that leads to a large skew between the external clock and the physical clocks received at the clocked elements. Moreover, this delay varies with processing and environment. Because of this skew, clocked elements on the chip are no longer synchronized with the external I/O signals. Guaranteeing setup and hold times becomes problematic, particularly at high frequencies where the skew exceeds half of the clock period. More sophisticated clock generators use *phase-locked loops* (PLLs) to compensate for this delay. Moreover, phase-locked loops can perform frequency multiplication to provide an on-chip clock at a higher frequency than the external clock.

Figure 11.24 shows a typical synchronous chip interface using a phase-locked loop to compensate for on-chip clock delays [Chandrakasan01]. In this example, Chip 1 sends a clock and D_{out} to Chip 2 and

receives D_{in} back from Chip 2. The data should be synchronized to the clock so each chip can sample it on the positive clock edge. However, the internal clock on Chip 2 is delayed through the clock distribution network. The PLL adjusts the internal clock in Chip 2 to correct for this delay and keep the clock synchronized with the data.

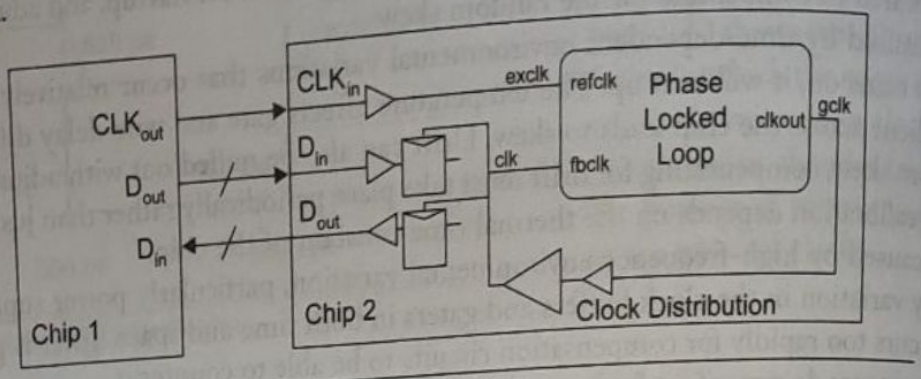

FIG 11.24 Synchronous chip interface with PLL

A phase-locked loop receives a *reference clock* and a *feedback clock* and produces an *output clock*. The phase and frequency of the output clock is automatically adjusted until the feedback clock is exactly aligned with the reference clock. In our application, the reference clock comes from Chip 1. The feedback clock is tapped off one of the physical clock wires that also drives the registers. The output clock is the *gclk* signal sent into the clock distribution network. Thus, the PLL adjusts *gclk* until the clock *clk* driving the data registers is aligned with the external clock *exclk*. This eliminates the systematic skew caused by the clock distribution delay.

Figure 11.25(a) shows a block diagram of a typical PLL. The heart of the PLL is a *voltage-controlled oscillator* (VCO). The control voltage is adjusted until the oscillator produces an output clock of the proper phase at the same frequency as the reference clock. This control is performed with a charge pump and RC filter. A phase detector determines whether the feedback clock leads or lags the reference clock. The charge pump consists of a pair of current sources enabled by the up and down signals to adjust the voltage on V_{ctrl} until the feedback clock becomes aligned with the reference.

Figure 11.25(b) shows a PLL that performs frequency multiplication. It uses a divide-by-N counter on the *fbclk* to generate a clock aligned in phase but at N times the frequency of the reference clock. For example,

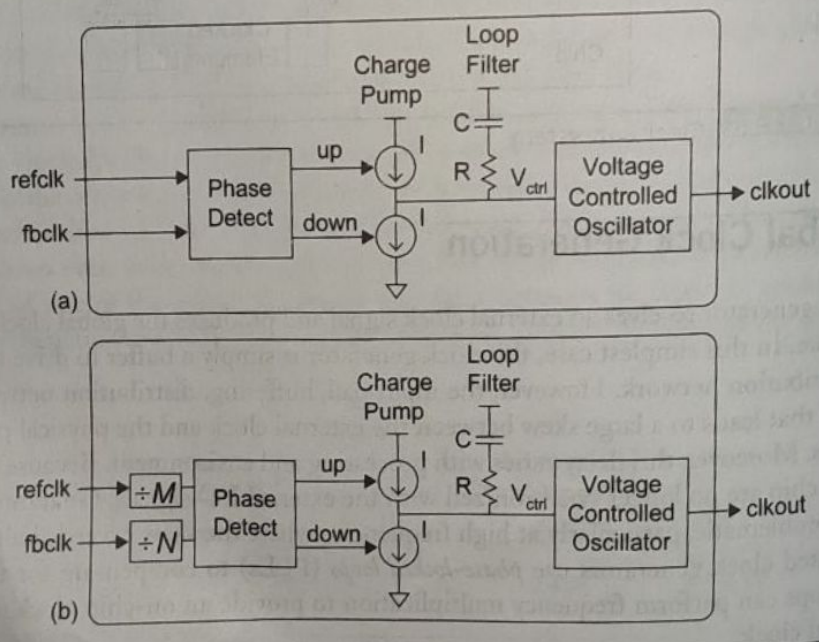

FIG 11.25 Phase-locked loop block diagram

the Pentium 4 uses a 100 MHz external system clock that can be multiplied up to a 3 GHz core clock. With another divide-by-M counter on the *refclk* terminal, the PLL can produce any rational N/M multiple of the input frequency.

A PLL is a feedback system and requires careful analysis to ensure stability over all process and environmental corners. The RC loop filter behaves as a second-order system, although stray capacitance on V_{ctrl} often leads to third-order effects. Noise on V_{ctrl} causes the VCO to change frequency, which leads to a phase error that increases over time; this problem is called *phase-error accumulation* and appears as jitter in the clock skew budget. Therefore, the loop filter should have high enough bandwidth to rapidly correct for noise. Phase-error accumulation impacts the timing budget for one chip communicating with another. However, only cycle-cycle jitter is important for on-chip paths between flip-flops. RC loop filters are difficult to build because of uncertainties in the passive component values caused by process variation. Thus, many PLLs use an active filter.

Figure 11.26 shows a circuit-level implementation of a simple PLL. The phase detector is a *phase-frequency detector* consisting of a pair of flip-flops with asynchronous reset that produces pulses to drive the frequency up or down depending on which clock arrived first. The charge pump uses a pair of current sources switched on by the *up* and *down* signals. Current sources are discussed further in Section 11.6.4. V_{ctrl} is buffered with an amplifier such as that from Figure 11.56 hooked up as a unity-gain follower. The VCO consists of an ordinary ring oscillator running off V_{ctrl} rather than V_{DD} so that its period increases as V_{ctrl} decreases. The number of stages determines the range over which the frequency can be adjusted. A level converter and some buffers amplify the signal to drive *clkout* across the chip.

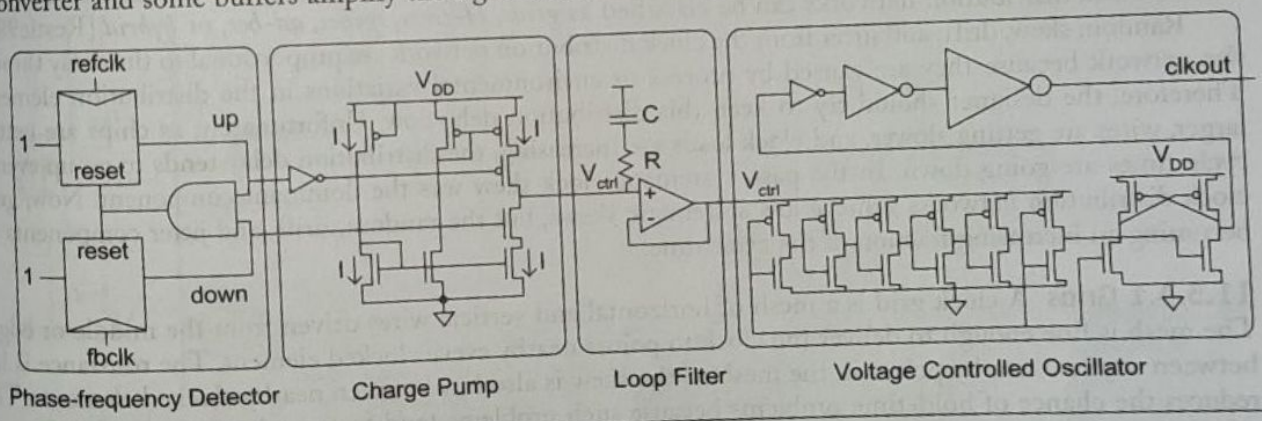

FIG 11.26 Simple PLL implementation

Power supply and substrate noise are the primary sources of PLL jitter, so the PLL should use a filtered power supply and generous guard rings to limit jitter. [Maneatis96, Maneatis03] describe self-biased delay elements that minimize sensitivity to supply noise and process variation. [Ingino01] describes a PLL with a filtered power supply operating up to 4 GHz with a ±25 ps peak-to-peak jitter.

A delay-locked loop (DLL) is a variant of a PLL that uses a voltage-controlled delay line rather than a voltage-controlled oscillator, as shown in Figure 11.27. It can be viewed as a control system that adjusts phase

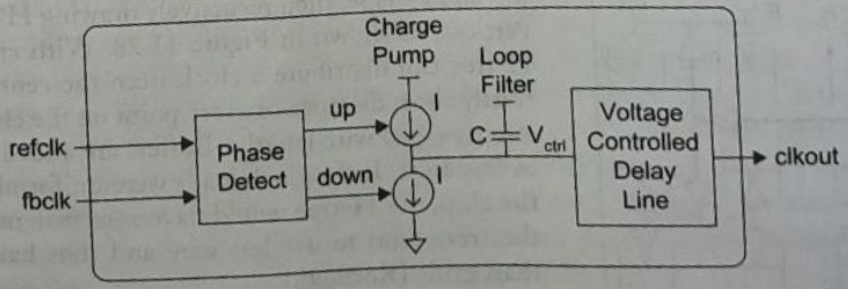

FIG 11.27 Delay-locked loop block diagram

rather than frequency on *clkout*. DLLs are not capable of frequency multiplication. However, noise on V_{ctl} creates only a steady phase error, so DLLs avoid phase-error accumulation. DLLs use a first-order loop filter, so ensuring stability is also easier.

PLLs and DLLs are notoriously difficult to build correctly. They require expertise in both feedback control systems and analog design. They must be carefully designed to acquire lock successfully and operate correctly with low jitter across process and environmental variations. A number of texts including [Chandrakasan01, Baker98, Dally98, Best03] offer introductions to loop design. For many applications, loops are best obtained as predesigned cells from a third party that specializes in loop design and offers a guarantee. True Circuits is a well-regarded PLL/DLL supplier.

11.5.4 Global Clock Distribution

The global clock must be distributed across the chip in a way that reaches all of the clocked elements at nearly the same time. In antiquated processes with slow transistors and fast wires, the clock wire had negligible delay and any convenient routing plan could be used to distribute the clock. In modern processes, the RC delay of the resistive clock wire driving its own capacitance and the clock load capacitance tends to be close to 1 ns for a well-designed distribution network covering a 15 mm square die. If the clock were routed randomly, this would lead to a clock skew of about 1 ns between physical clocks near and far from the clock generator. This could be several times the cycle time of the system. Thus, the clock distribution system must be carefully designed to equalize the *flight time* between the clock generator and the clocked receivers. Global clock distribution networks can be classified as *grids*, *H-trees*, *spines*, *ad-hoc*, or *hybrid* [Restle98].

Random skew, drift, and jitter from the clock distribution network are proportional to the delay through the network because they are caused by process or environmental variations in the distribution elements. Therefore, the designer should try to keep this distribution delay low. Unfortunately, as chips are getting larger, wires are getting slower, and clock loads are increasing, the distribution delay tends to go up even as cycle times are going down. In the past, systematic clock skew was the dominant component. Now, good clock distribution networks achieve low systematic skews, but the random, drift, and jitter components are becoming an increasing fraction of the cycle time.

11.5.4.1 Grids A clock grid is a mesh of horizontal and vertical wires driven from the middle or edges. The mesh is fine enough to deliver the clock to points nearby every clocked element. The resistance is low between any two nearby points in the mesh so the skew is also low between nearby clocked elements. This reduces the chance of hold-time problems because such problems tend to occur between nearby elements where the propagation delay between elements is also small. Grids also compensate for much of the random skew because shorting the clock together makes variations in delays irrelevant. They can be routed early in the design without detailed knowledge of latch placement. However, grids do have significant systematic skew between the points closest to the drivers and the points furthest away. They also consume a large amount of metal resources and hence have a high switching capacitance and power consumption. Section 11.8 traces the evolution of clock grids in the Alpha series of microprocessors.

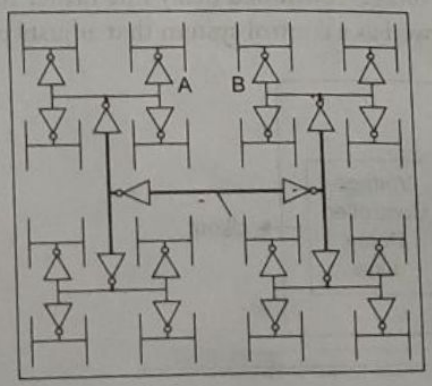

FIG 11.28 H-tree

11.5.4.2 H-Trees An H-tree is a fractal structure built by drawing an H shape, then recursively drawing H shapes on each of the vertices, as shown in Figure 11.28. With enough recursions, the H-tree can distribute a clock from the center to within an arbitrarily short distance of every point on the chip while maintaining exactly equal wire lengths. Buffers are added as necessary to serve as repeaters. If the clock loads were uniformly distributed around the chip, the H-tree would have zero systematic skew. Moreover, the trees tend to use less wire and thus have lower capacitance than grids [Restle98].

In practice, the H-tree still shows some skew because the clock loads are not uniform, loading some leaves of the tree more than others. Moreover, the tree often must be routed around obstructions such as memory arrays. The leaves of the H do not reach every point on the chip, so some short physical clock wires are required after the local clock gater. Nevertheless, with careful tapering of the wires and sizing of the clock gaters, H-trees can deliver nearly zero systematic skew. A drawback of H-trees is that they may have high random skew, drift, and jitter between two nearby points that are leaves of different legs of the tree. For example, the points *A* and *B* in Figure 11.28 might experience large skews. As the points are close, this is a particular problem for hold times.

Figure 11.29 shows a modified H-tree used on the Itanium 2. The primary clock driver in the center of the chip sends a differential output to four differential repeaters on the leaves of the H. These repeaters drive a somewhat irregular pattern of wiring to second-level clock buffers (SLCBs) serving units all across the chip. The wiring and SLCB placement is determined by the nonuniform clock loads and obstructions on the chip. A custom clock router automatically generated the tree based on the actual clock loads so that the tree could be easily rerouted when loads change late in the design process. The SLCBs drive local clock gaters, producing the multitude of clock waveforms used on the microprocessor.

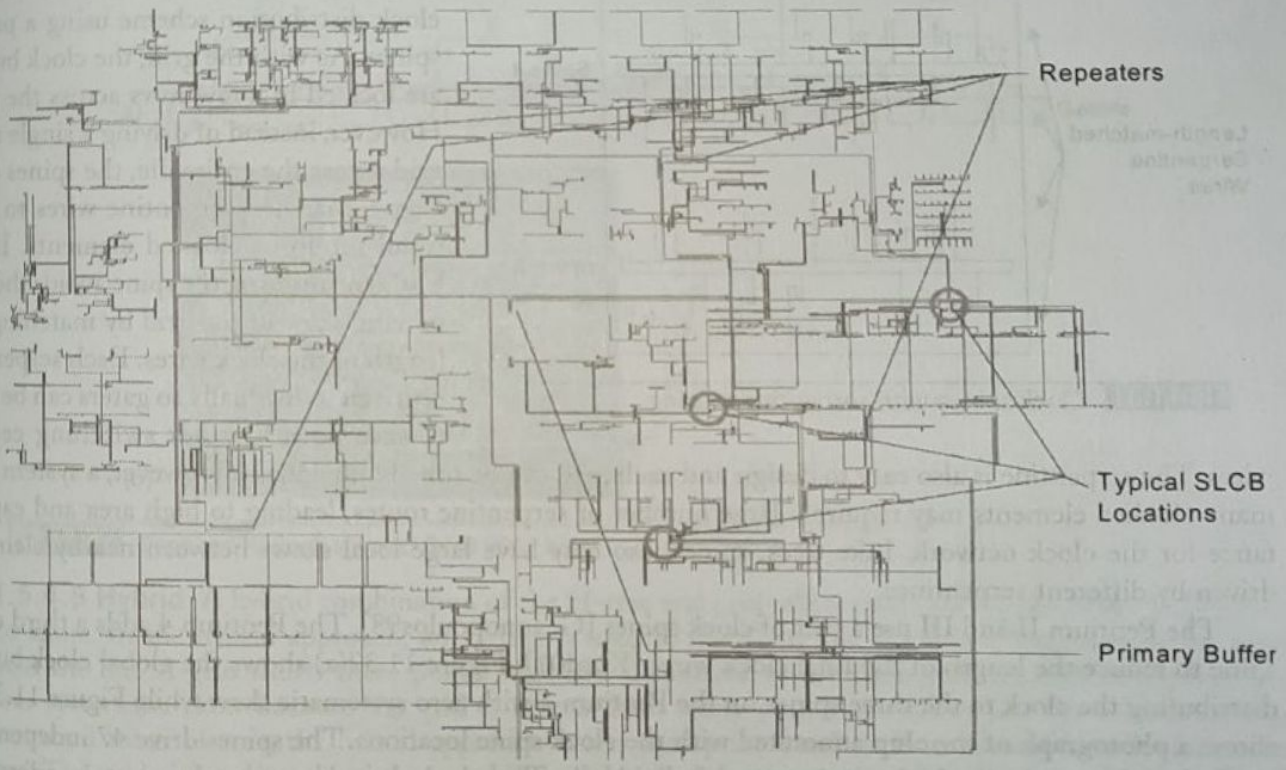

FIG 11.29 Itanium 2 modified H-tree

Figure 11.30 shows the differential driver used as a primary clock buffer and repeater on the Itanium 2 [Anderson02]. The input stage is a differential amplifier sensitive to the point where the differential inputs cross over. The repeater pulses either p_1 or n_1 and p_2 or n_2 to switch the internal nodes y and $\bar{y}$. The small tristate keeper prevents these nodes from floating after the pulse terminates. The SLCB uses the same structure, but produces only a single-ended output. It also provides a current-starved adjustable delay line to compensate for systematic skew and to help locate critical paths during debug. The repeater provides a high drive capability with a low input capacitance. Thus, few stages of clock buffering are needed in the network. With so few repeaters, the area overhead of providing a filtered power supply is modest. Although the repeaters are relatively slow, their jitter is controlled with supply filtering.

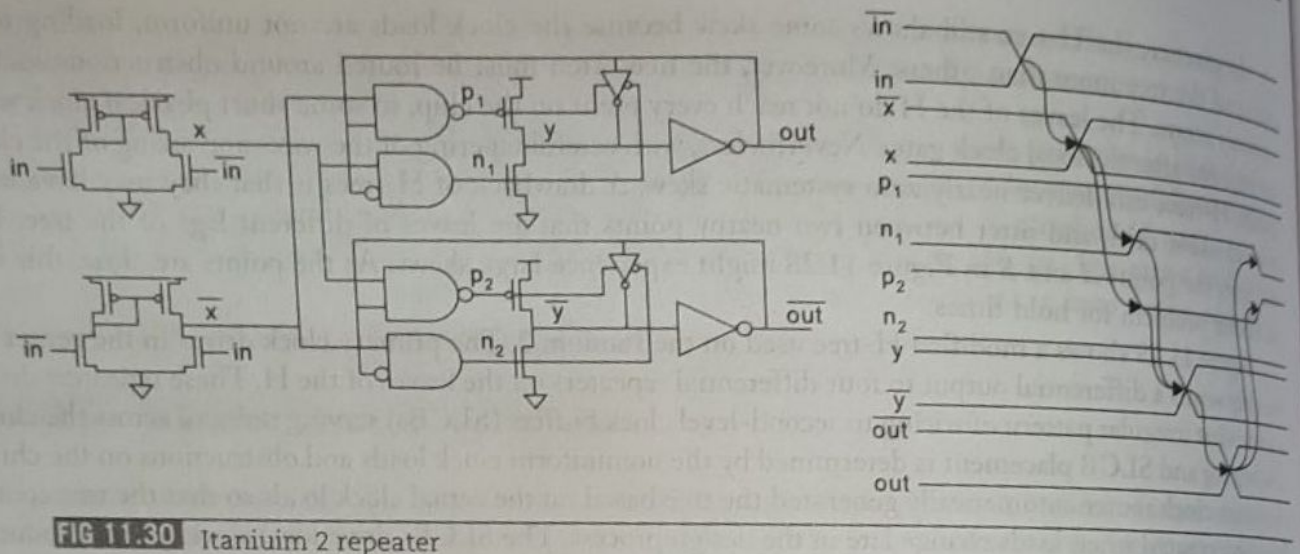

FIG 11.30 Itaniuim 2 repeater

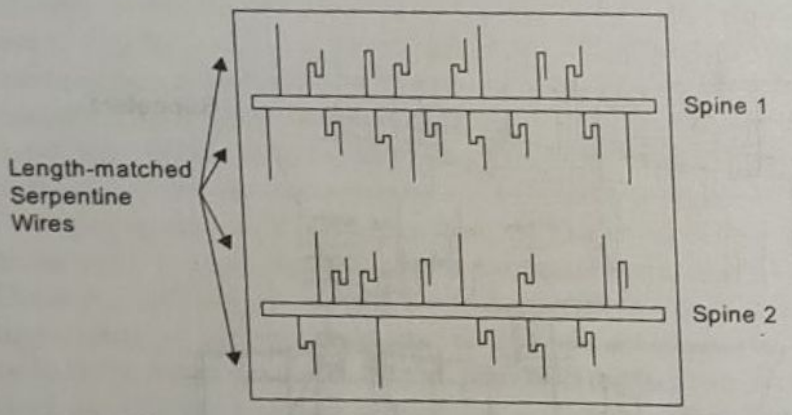

FIG 11.31 Clock spines with serpentine routing

11.5.4.3 Spines Figure 11.31 shows a clock distribution scheme using a pair of spines. As with the grid, the clock buffers are located in a few rows across the chip. However, instead of driving a single clock grid across the entire die, the spines drive length-matched serpentine wires to each small group of clocked elements. If the loads are uniform, the spine avoids the systematic skew of the grid by matching the length of the clock wires. Each serpentine is driven individually so gaters can be used to save power by not switching certain wires. The serpentine is also easy to design and each load can be tuned individually. However, a system with many clocked elements may require a large number of serpentine routes, leading to high area and capacitance for the clock network. Like trees, spines also may have large local skews between nearby elements driven by different serpentines.

The Pentium II and III use a pair of clock spines [Geannopoulos98]. The Pentium 4 adds a third clock spine to reduce the length of the final clock wires [Kurd01]. Figure 11.32(a) shows the global clock buffers distributing the clock to the three spines on the Pentium 4 with zero systematic skew while Figure 11.32(b) shows a photograph of the chip annotated with the clock spine locations. The spines drive 47 independent clock domains, each of which can be gated individually. The clock domain gaters also contain adjustable delay buffers used to null out systematic and random skew and even to force deliberate clock delay to improve performance.

11.5.4.4 Ad-hoc Many ASICs running at relatively low frequencies (100's of MHz) still get away with an ad-hoc clock distribution network in which the clock is routed haphazardly with some attempt to equalize wire lengths or add buffers to equalize delay. Such ad-hoc networks can have reasonably low systematic skews because the buffer sizes can be adjusted until the nominal delays are nearly equal. However, they are subject to severe random skew when process variations affect wire and gate delays differently. This is the level that most commonly available tools support. Most design teams using ad-hoc clock networks also lack the resources to do a careful analysis of random skew, jitter, and drift. Therefore, they should be conservative in defining a skew budget and must be careful about hold time violations.

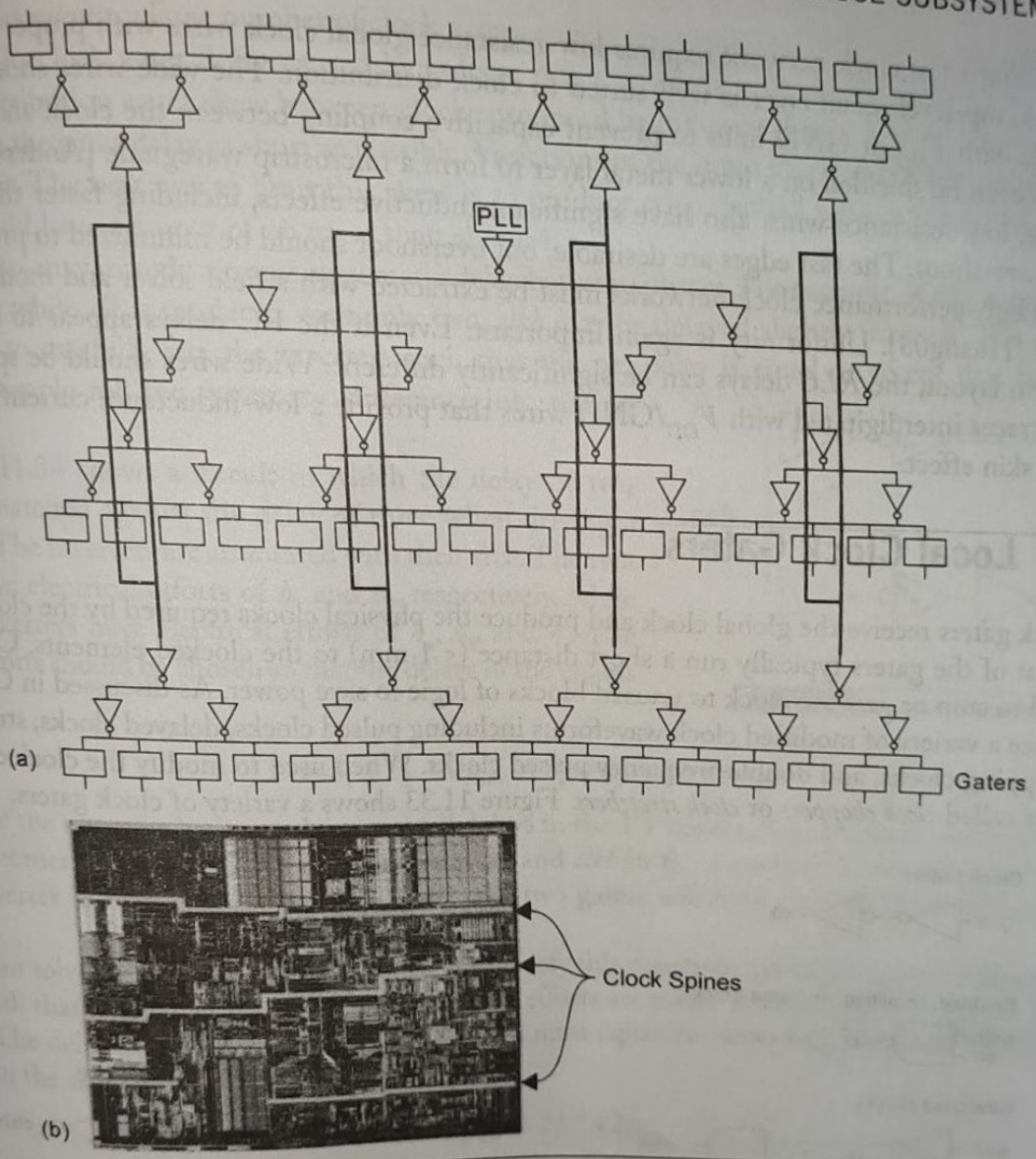

FIG 11.32 Pentium 4 clock spines. (b) © 2001 IEEE.

11.5.4.5 Hybrid A hybrid combination of the H-tree and grid offers lower skew than either an H-tree or grid alone. In the hybrid approach, an H-tree is used to distribute the clock to a large number of points across the die. A grid shorts these points together. Compared to a simple grid, the hybrid approach has lower systematic skew because the grid is driven from many points instead of just the middle or edge. Compared to an H-tree, the hybrid approach is less susceptible to skew from nonuniform load distributions. The grid also reduces local skew and brings the clock near every location where it is needed. Finally, the hybrid approach is regular, making layout of well-controlled transmission line structures easier.

IBM has used such a hybrid distribution network on a variety of microprocessors including the Power4, PowerPC, and S/390 [Restle01]. A primary buffered H-tree drives 16–64 sector buffers arranged across the chip. Each sector buffer drives a smaller tree network. Each tree can be tuned to accommodate nonuniform load capacitance by adjusting the wire widths. Together, the tunable trees drive the global clock grid at up to 1024 points. IBM uses a specialized tool to perform the tuning.

11.5.4.6 Layout Issues High-speed clock distribution networks require careful layout to minimize skew. The two guiding principles are that the network should be as uniform and as fast as possible. In a uniform network, chip-wide process or environmental variations should affect all clock paths identically. In a fast network, localized variations that cause a fractional difference between two clock path delays lead only to modest amounts of skew. For example, voltage noise that causes a 10% delay variation between two paths through an H-tree will lead to 80 ps of jitter if the tree delay is 800 ps, but 160 ps of jitter if the tree delay is 1600 ps.