CHAPTER 1

INTRODUCTION

Clocking is one of the single most important decisions facing the designer of a digital system. Unfortunately much too often it has been taken lightly at the beginning of a design and that viewpoint has proven to be very costly in the long run (Wagner 1988). Thus, it is not pretentious to dedicate an entire book to this subject. However, this book is limited to the even narrower issue of clocked storage elements (CSE), widely known as flip-flops and latches. The issues dealing with clock generation, frequency stability and control, and clock distribution are too numerous to be discussed in depth in this book and so they are covered only briefly. The interested reader is referred to the other books dealing with those issues, such as the one by Friedman (1995).

The importance of clocking has become even more emphasized, as the clock speed is rising rapidly, doubling every three years, as seen in Fig. 1.1. However, the clock uncertainties have not been scaling proportionally with the frequency increase, and an increasingly large portion of the clock cycle has been spent on the clocking overhead. The ability to absorb clock skew or to make the clocked storage element faster is reflected directly in the enhanced performance, since the performance is directly proportional to the clock frequency of a given system. Such performance improvements are very difficult to obtain using traditional techniques on the architecture or microarchitecture levels. The difficulties are caused by the overhead imposed by the CSE delay, and the clock uncertainties. Thus, setting the clock to the right frequency, and utilizing every available picosecond of the critical path, is increasingly important. It is our opinion that traditional clocking techniques will reach their limit when the clock frequency reaches the 5 to 10 GHz range. Thus, new ideas and new ways of designing digital systems are needed. We do not pretend to know what the future trend in clocking should

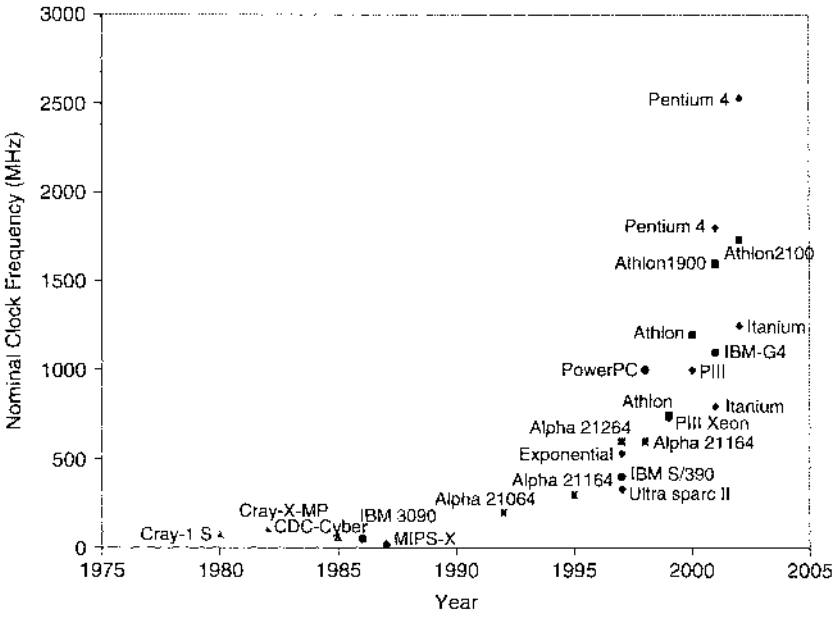

Figure 1.1. Clock frequency versus year for various representative machines.

be, but we feel that some of the ideas discussed in this book can provide a good path to follow.

Computers built in the past were large and filled several electronic cabinets in large air-conditioned rooms that occupied entire floors. They were built from discrete components or used a few large-scale integration (LSI) chips in the later models. Those systems were clocked at frequencies of about one or a few tens of megahertz, as shown in Table 1.1. The first electronic computer, ENIAC (Electronic Numerical Integrator and Calculator), for example, operated at the maximal clock frequency of 18 kHz. Given the low scale of integration, it was possible to "tune" the clock. This was achieved by either adjusting the length of the wires that distributed the clock signals, or by tuning the various delay elements on the cabinets or the circuit boards, so that the clock signal arrived at every circuit board at approximately the same time. With the advent of very large-scale integration (VLSI) technology, and increased integration levels, the ability to tune the clock has been greatly diminished. The clock signals are generated and distributed internally within the VLSI chip. Therefore, much of the burden of absorbing clock signal variations at various points on the VLSI chip has fallen on the clocked storage element.

1.1. CLOCKING IN SYNCHRONOUS SYSTEMS

The notion of clock and clocking is essential for the concept of synchronous design of digital systems. The synchronous system assumes the presence of the

Table 1.1 Clock Frequency of Selected Historic Computers and Supercomputers

System	Date Introduced	Technology	Class	Nominal Clock Period (ns)	Nominal Clock Frequency (MHz)
Cray-X-MP	1982	MSI ECL	Vector processor	9.5	105.3
Cray-1S,-1M	1980	MSI ECL	Vector processor	12.5	80.0
CDC Cyber 180/990	1985	ECL	Mainframe	16.0	62.5
IBM 3090	1986	ECL	Mainframe	18.5	54.1
Amdahl 58	1982	LSI ECL	Mainframe	23.0	43.5
IBM 308X	1981	LSI TTL	Mainframe	24.5, 26.0	40.8,38.5
Univac 1100/90	1984	LSI ECL	Mainframe	30.0	33.3
MIPS-X	1987	VLSI CMOS	Microprocessor	50.0	20.0
HP-900	1982	VLSI CMOS	Micromainframe	55.6	18.0
Motorola 68020	1985	VLSI CMOS	Microprocessor	60.0	16.7
Bellmac-32A	1982	VLSI CMOS	Microprocessor	125.0	8.0

Source: Wagner 1988.

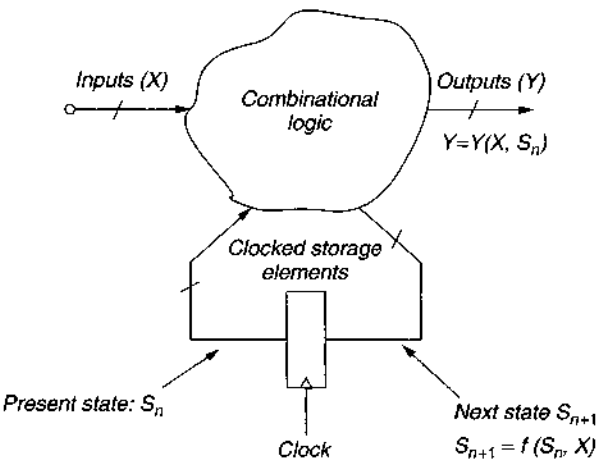

Figure 1.2. The concept of finite-state machine.

storage elements and combinational logic, which together make up a finite-state machine (FSM). The changes in the FSM are in general the result of two events: clock and input signal changes, as illustrated in Fig. 1.2.

The next state, S_{n+1}, is a function of the present state, S_n, and the logic value of the input signals: $S_{n+1} = S_{n+1}(S_n, X_n)$. The remaining question is: When in time will FSM change to the next state, S_{n+1}. This change is determined by the

type of clocked storage elements used and the clock signal. The function of the clock signal is to provide a reference point in time when the FSM changes from the present, S_n, to the next state, S_{n+1}. This process is illustrated in Fig. 1.3.

In Fig. 1.3, we have implicitly assumed that the moment when the state changes from S_n to S_{n+1} is determined by the change in the clock signal from logic "0" to logic "1." In fact, this change is determined by the type of clocked storage element and its functionality. We will be discussing this point in detail later in this book. For the purposes of this discussion, we observe that without the clock signal, the change from S_n to S_{n+1} could not be precisely determined. There are digital systems where this change is not caused by the presence, or more precisely, by a change in the clock signal, but by a change of the data signal, for example. Such systems are known as *asynchronous systems*, because they do not require the presence of the clock signal in order to effect an orderly transition from S_n to S_{n+1}. A great deal of research in defining a workable asynchronous system has been done in the last several decades. Recently a microprocessor was designed to operate in an asynchronous manner, and it has been claimed that some small advantages in power consumption were obtained (Woods et al. 1997). In spite of that, the practicality and advantage of the asynchronous design has yet to be proven (Furber et al. 2001). In this book, we limit our discussion to synchronous systems.

If we extend the FSM state diagram in time, we obtain an illustration of the pipeline design (Fig. 1.3). In many cases, when dealing with the synchronous design, the delay throughout the logic block is excessive and the signal change cannot propagate to the inputs of the clocked storage elements in time to effect the change to the next state. In that case, the machine has not met the "critical-path requirement." Such an FSM will fail in its functionality, because the changes

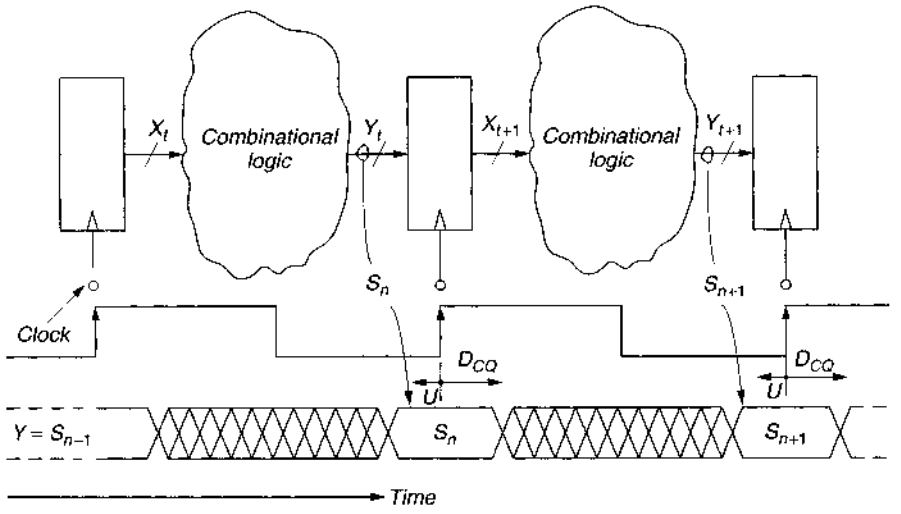

Figure 1.3. State changes in the finite-state machine.

initiated by the input signals will have no effect. This is because the time allowed to change to the next state, S_{n+1}, is too short and the input signal change does not have sufficient time to propagate. In technical jargon this is known as *critical-path violation*. Critical path is defined as the chain of gates in the *longest* (slowest) path through the logic, which causes a signal to take a certain length of time to propagate from the input to the output. Often times, an additional state (or states) is inserted to assure that every transition proceeds in an orderly and timely fashion. This is known as pipelining. A diagram of a pipelined system is shown in Fig. 1.4.

Several clock cycles may be needed in order for the signal to propagate through various stages of a computer system. In general, execution of an instruction may require several *machine cycles*, where machine cycle is defined as the time interval necessary for one atomic operation to execute an instruction. One machine cycle normally takes several clock cycles. The machine cycle is often designated by a waveform defining its own cycle. This is especially true if *microcode* is used to control the machine. In the past, microcoding was a popular concept and it was used extensively in Complex Instruction Set Computers (CISC). In those cases, a process of executing an instruction required several machine cycles. During each machine cycle one *microinstruction* was executed. It normally took several microinstructions to execute an instruction. Each machine cycle required one or several register transfers or passes through several pipeline stages. That in turn required one or more clock cycles, or multiple phases of the clock. Thus, the clocking was quite complex and encompassed several levels of hierarchy. This

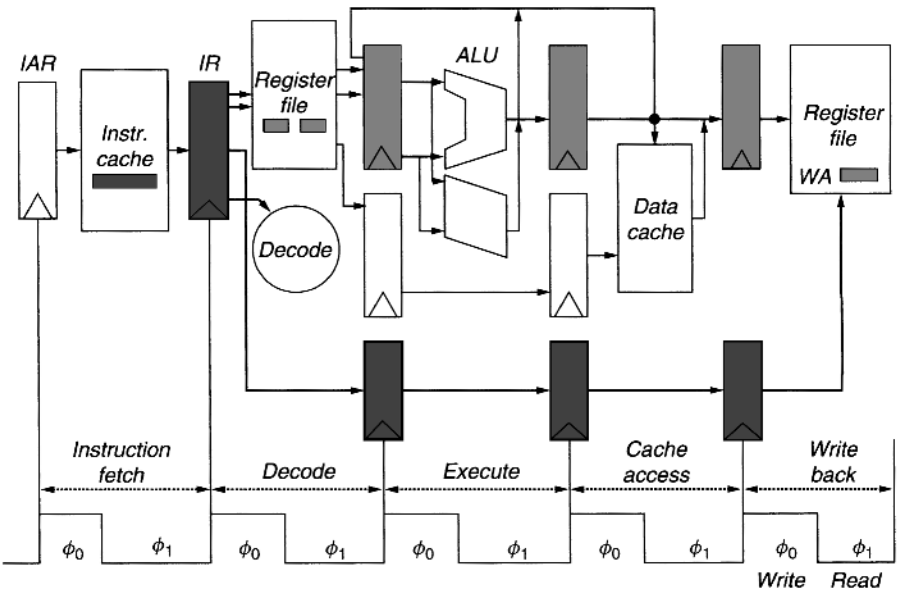

Figure 1.4. Diagram of a pipelined system.

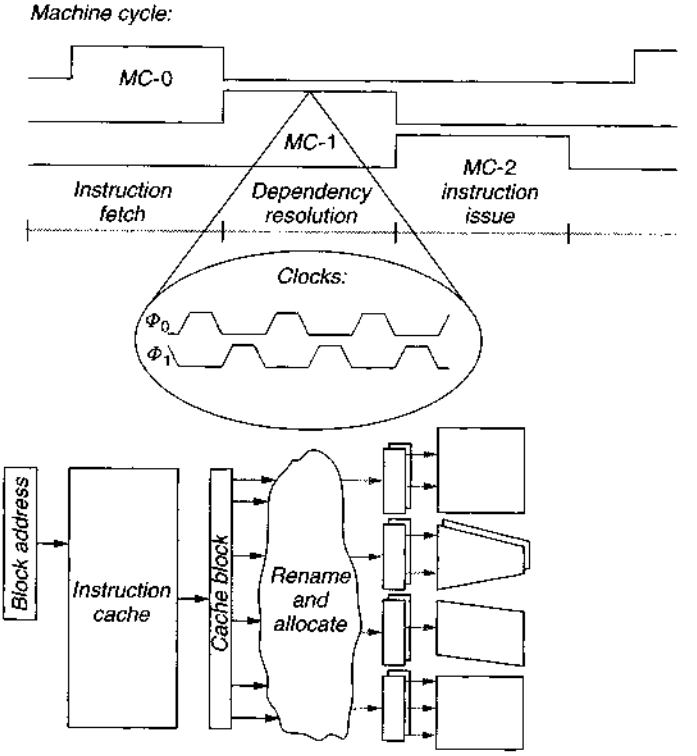

Figure 1.5. Machine execution phases with respect to the clock cycles.

is illustrated in Fig. 1.5, where three distinct machine cycles, *Instruction Fetch*, *Dependency Resolution*, and *Instruction Issue*, are shown. Dependency resolution can be quite a complex operation, requiring several *register transfers*, which means several clock cycles are necessary to complete this operation (as shown in Fig. 1.5). The machine would normally scan the cache block for several instructions and try to resolve any data dependencies. At the end of this cycle, operands will be fetched and placed in the corresponding registers (reservation stations) of the execution units.

In microcoded machines a large disparity existed between the speed of the clock and the speed of logic. It could take several clock cycles or even several tens or hundreds of clock cycles to execute one instruction. A more complex instruction required many more clock cycles. There could be tens of *logic levels* in the critical path, and 40 to 50 were not uncommon. Thus, the time associated with the clock and clocking was not as critical as it is today.

As the level of integration increased, combined with the increased speed of today's machines, the number of logic levels in the critical path started to diminish rapidly. Today's high-speed processors are either implementing Reduced Instruction Set Computer (RISC) architecture, or are running CISC code. However, to

be able to efficiently implement superscalar execution cores, even CISC computers are translating their instructions into simple RISC-type operations called ROPs (RISC operations). Their microarchitecture can execute one or several ROPs in place of one CISC instruction. Therefore, the concept of microcoding has disappeared, as did the concept of machine cycle when implementing a particular machine architecture. The instructions (or ROPs) are executed in one cycle, which is usually driven by a single-phase clock. In other words, one instruction (or one ROP) is executed in every clock cycle. The levels of hierarchy that existed between the clock cycle and instruction execution no longer exist. In addition, the number and depth of pipeline stages keeps increasing in order to accommodate the trend toward increasing speed. As a result, the number of logic stages between the two CSEs keeps decreasing. Today 10 levels of logic in the critical path are more common. This number is still decreasing, as illustrated in Fig. 1.6. Any overhead associated with the clock system and clocking mechanism directly and adversely affects machine performance and is therefore critically important.

With this introduction we should be able to understand the function of the clock signal before we proceed with other definitions. The function of the clock signal is comparable to the function of the metronome in music. Similarly, in the digital system the clock designates the exact moment when the state is changing, as well as when the next state is to be captured. Also, all of the logic operations have to finish before the tick of the clock, because their final values are being captured by that clock event. Therefore, the clock provides the time reference point, which determines the flow of the data in the digital system.

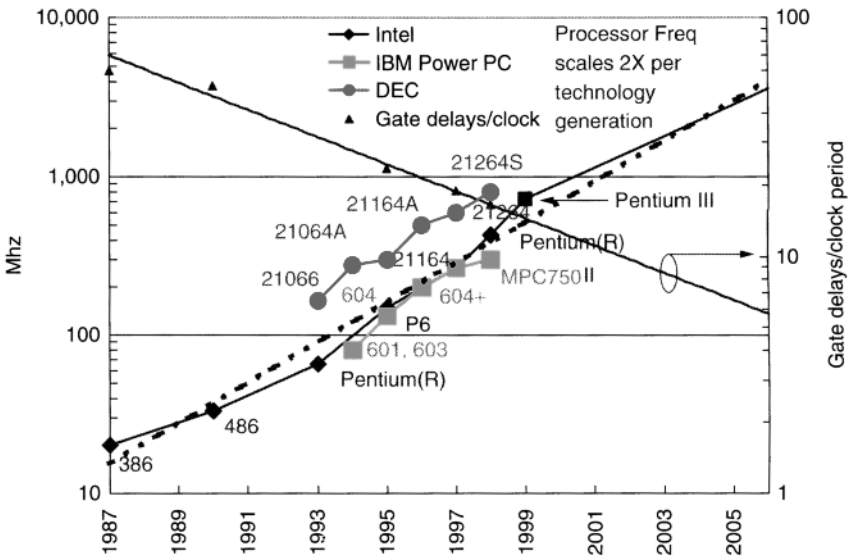

Figure 1.6. Increase in the clock frequency and decrease in the number of logic levels in the pipeline. (Borkar 1999), Copyright © 1999 IEEE.

1.2. SYSTEM CLOCK DESIGN

The clock system is usually divided into two distinct categories: *clock generation* and *clock distribution*. However, this classification should be extended by adding CSEs as an additional category, because the nature of the clocked storage elements is intimately connected to the clock system generation and distribution, and it is the nature of clocked storage elements that dictates the requirements imposed on the clock system. This relationship is best illustrated by the choice of clocking scheme, as shown in Fig. 1.7. The clock system can consist of a single-phase, a two-phase, or a multiple-phase clock. Transfer of data between CSEs in the system is usually accomplished by using an active phase of the clock. Thus, the clock phase controls the transfer of the information among the CSEs in the system. To prevent data from moving further then desired (achieving *nontransparency*), the clock phases are separated in time. This is referred to as *nonoverlapped* clock phases. In high-performance systems various phases of the clock can be overlapped in order to increase total system performance.

In the older systems it was more common to use multiple-phase clocks (Siewiorek et al. 1982). Transparent latches or flip-flops triggered by short pulses were used as storage elements. As the frequency of the operation kept increasing,

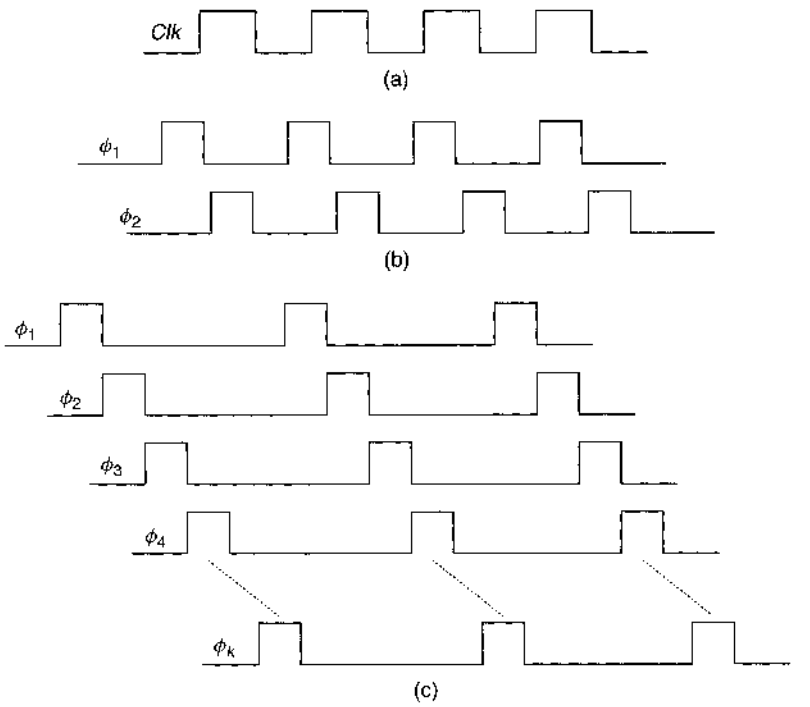

Figure 1.7. System clocking schemes: (a) single-phase clock; (b) two-phase clock; (c) multiple-phase clock.

it became exceedingly difficult to control various phases of the clock and their relationship to each other.

The two-phase clock is a robust scheme and is compatible with the design for testability, a desired feature of a complex computer system. Such a scheme, which incorporates a test mode, has been used in generations of IBM mainframe computers as a part of level-sensitive scan design (LSSD) methodology (LSSD 1985). The two nonoverlapping phases of the clock assure a robust clocking system that can tolerate manufacturing and process-parameter changes.

Given the continuing search for more speed and increased level of integration, even the two phases of the clock became difficult to control on the VLSI chip. This led to the widespread adoption of the single-phase clock in use today. Although two-phase clocking is still used, it is a single-phase clock that is distributed throughout the system, allowing the two necessary phases to be generated locally. This technique achieves two goals: (1) necessary amplification of the clock signals and ability to drive a large row of storage elements (register, for example), and (2) generation of two clock phases and compatibility with scan methodology. A scheme used for local two-phase clock generation from a single-phase clock distributed on the chip is shown in Fig. 1.8. Such a scheme is also capable of supporting the test and debug mode. The two phases of the clock, C_1 and C_2, are generated from the global clock CLKG. Specialized circuitry was added to allow for edge shifting at the cycle boundary (Sigal et al. 1997). Enabling and disabling of the clock phases is used to switch from normal operation to the *scan* mode that is used for testing.

1.2.1. Global System Clock Generation

Clock generation begins on a system board, where the global system clock reference is generated from a "crystal" oscillator. This is a circuit that uses

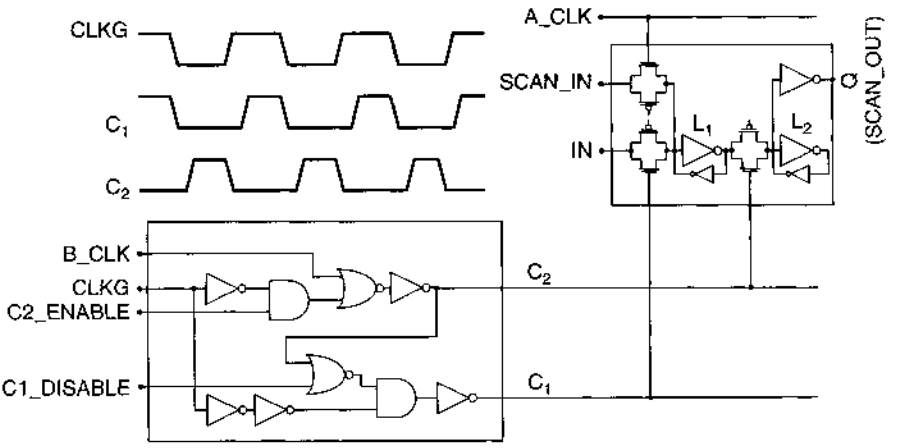

Figure 1.8. Local generation of two-phase clocks as used in IBM S/390 G4. (Sigal et al. 1997), reproduced by permission.

a piezoelectric quartz crystal or some other ceramic material as a mechanical representation of an electrical inductance–capacitance–resistance (LRC) series resonant circuit. Piezoelectric effect in a material occurs with the exchange of energy between the mechanical forces and applied electric field. In quartz crystal, the physical dimensions of the lattice can very precisely determine the oscillation frequency. One excellent property of such resonators is their extremely high Q-factor, typically 1000–10,000. By attaching a nonlinear element (such as an NFET) to the resonator, the series resistance of the resonator is canceled by the negative resistance of the nonlinear element and "lossless" oscillations are maintained. Due to the high-quality Q-factor, the variation of the resonant frequency of the oscillator is only a few parts per million (ppm). Two realizations of the clock oscillator are shown in Fig. 1.9a and 1.9b.

System clock is set to directly correspond to the speed of data busses on the system board, that is, from 66 MHz, 100 MHz, 133 MHz, 266 MHz, and higher, in PC boards, to a few hundred MHz in specialized systems. However, the on-chip clocks operate at frequencies that are in the GHz range. Even if the on-board clock signal of the same frequency as the on-chip clock could be generated, it would be very hard to bring it on-chip because of large parasitic capacitances and inductances in the package and bond-wires/balls that connect to the die. For these reasons, the low-frequency system clock is first brought on-chip and then frequency multiplication is performed to achieve the desired on-chip clock rate.

The time difference between the external clock and the internal clock, called *insertion delay* (shown in Fig. 1.10), increases relative to the clock period with the increase in the clock frequency. Input data are synchronized with the external clock, but can be stored directly in the storage elements clocked by the internal clock. Any insertion delay between the external and internal clocks directly

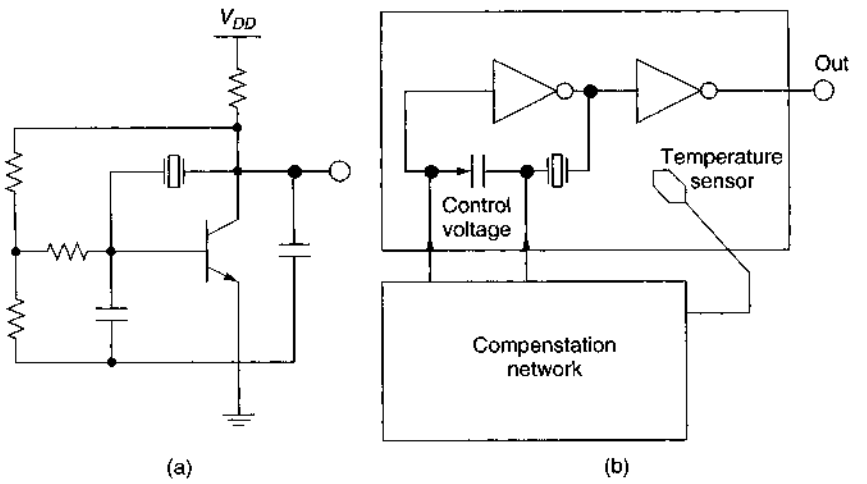

Figure 1.9. (a) Crystal oscillator. (b) Temperature-compensated crystal oscillator.

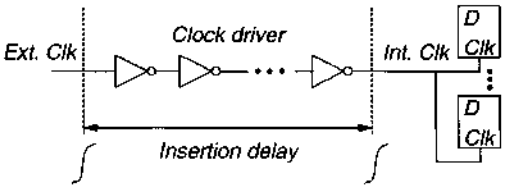

Figure 1.10. On-chip clock insertion delay.

impacts the cycle time of the processor. The insertion delay is caused by the on-chip clock-driver delay, with the inverter chain representing the equivalent of the clock-driver tree, and clocked storage elements representing the total clock load. Several nF of the clock load are routinely encountered in modern microprocessor designs (Young et al. 1992). The clock-driver tree requires five or more fan-out of 4 (FO4) delays, which easily accounts for over 50% of the processor cycle time. Moreover, due to process and environmental variations, the delay of the clock driver may vary, causing an unknown phase relationship of the external and internal clocks.

The problem of external and internal clock alignment can be solved by using the phase-locked loop (PLL). The main task of the PLL is to align the external reference clock with the on-chip internal clock at the end of the clock driver, thus effectively removing the driver delay.

1.2.2. On-Chip Clock Generation

There are two main types of PLLs. In the first type, the PLL has its own voltage-controlled oscillator (VCO) that generates the internal clock, which is then aligned to the external reference clock by the virtue of negative feedback, as shown in Fig. 1.11. The phase difference between the external reference clock and the internal distributed clock is detected with the phase detector (PD), and low-pass filtered (LP), to create the control voltage for the VCO, steering the oscillation frequency in order to align the external and internal clocks, ideally achieving a zero phase difference. At this point, a so-called *phase lock* is achieved (Gardner 1979). This type of PLL was introduced first, and so historically it kept the name PLL. One example of the PLL operation is shown in Fig. 1.11, where the output of the phase detector is the XOR of the external clock reference and the internal clock, producing pulses, p, that are then low-pass filtered to produce the slowly changing control voltage, cv, which changes the frequency of the VCO, and hence the internal clock. At first, the external and internal clocks have a phase difference of 135°, but after the phase difference is detected and the frequency of the internal clock changes, the phase difference is decreased to 45°.

The other type of PLL is delay-line based or delay-locked loop (DLL). As shown in Fig. 1.12, the VCO in the PLL is replaced by the voltage-controlled delay line (VCDL), which delays the external clock, feeding the clock driver,

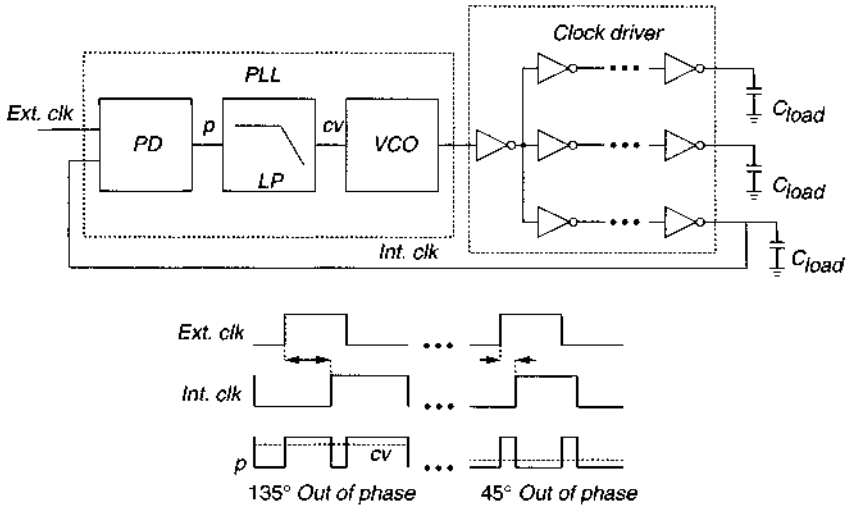

Figure 1.11. Phase-locked loop block diagram and operation.

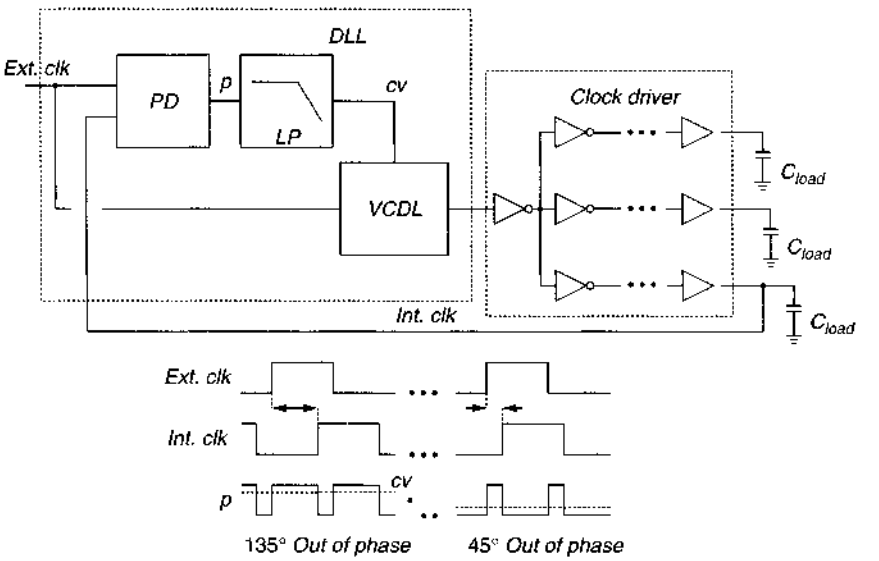

Figure 1.12. Delay-locked loop block diagram and operation.

until the internal clock becomes aligned with the external clock, at which point the control voltage of the VCDL become steady and the loop stays in lock. An example similar to that in Fig. 1.11 is given in Fig. 1.12. The main difference between the examples is that, unlike in Fig. 1.11, the internal clock in Fig. 1.12 does not change frequency over time, but is delayed in order to achieve phase

alignment. The key point to understand is that alignment is possible in both PLL and DLL, because both the external and internal clocks are periodic, which delays them by an integer number of cycles with respect to each other, resulting in cancellation of the phase difference. Otherwise, it would not be physically possible to eliminate this delay. It is only possible to add more delay until the total delay becomes an integer number of clock cycles.

In addition to clock alignment, PLLs can perform frequency multiplication. Figure 1.13 shows a general block diagram where the VCO operates at $f_{vco} = f_{ext} \times B \times C/A$, and the frequency of the internal clock is $f_{int} = f_{vco}/B$. Typically, the value of B is two, to guarantee a 50% duty cycle of the internal clock, and the value of A is one. The value of C is set to the ratio between the desired internal-clock frequency and the external (system) -clock frequency (Young et al. 1992), which is always conveniently set to be an integer value, preferably base two. There are, however, cases where multiple values of A, B, and C are used in the power-up sequence to avoid excessive supply noise on large chips, like Alpha 21264 (von Kaenel et al. 1998).

From the standpoint of noise performance, the VCO (VCDL) is the most critical part of the PLL (DLL). It is therefore illustrative to compare most common design styles and discuss the possible trade-offs. VCO is built either as a ring oscillator topology, Fig. 1.14, or an inductance–capacitance (LC) tank oscillator, Fig. 1.15. Ring-oscillator-based VCOs are relatively easy to implement, and require much less area than LC tank oscillators. By regulating the supply,

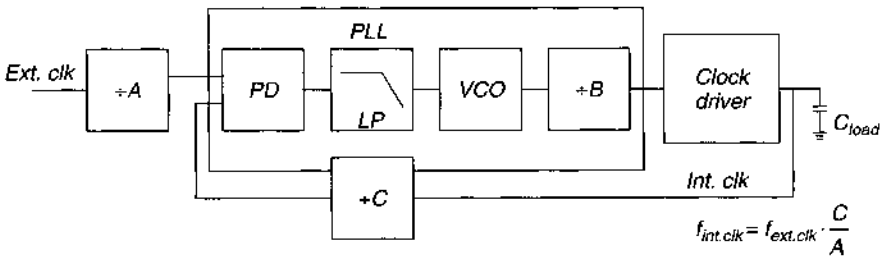

Figure 1.13. PLL frequency multiplication.

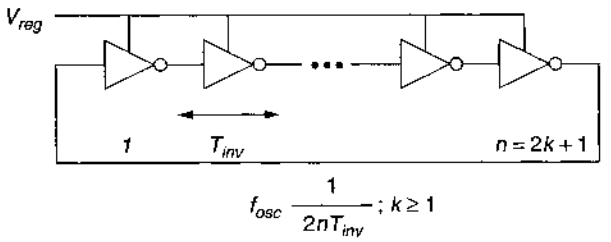

Figure 1.14. Ring-oscillator-based VCO, with CMOS inverters as delay elements.

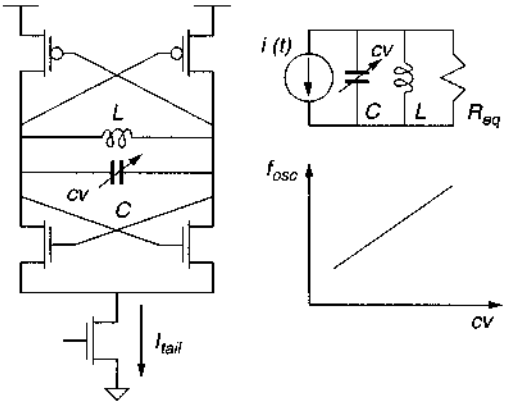

Figure 1.15. *LC* tank-based VCO, equivalent ac circuit model and current waveform.

inverter delay is controlled, and so is the oscillation frequency. The minimum number of stages needed to sustain oscillations is three, since it provides sufficient delay, while typical numbers range from three to seven or more stages, (Hajimiri 1998).

With the increase in clock frequency and the use of on-chip spiral inductors, both feasible with today's technology, *LC* tank-based VCOs are becoming increasingly popular due to superior phase-noise performance. However, *LC* tank oscillators do not always perform better than ring oscillators. This largely depends on the dominant source of noise and the number of stages in the output buffer and ring oscillator (Hajimiri 1998). A typical *LC* tank VCO is shown in Fig. 1.15, with an equivalent small-signal model and frequency characteristic as a function of applied *cv*.

A VCDL can be built from the same delay elements as the ring-oscillator VCO. The delay elements most often used are differential pairs, which provide good power-supply rejection, and the recently popular inverters with a power-supply regulator that performs power-supply filtering and effectively shields the inverters from any power-supply noise (von Kaenel et al. 1998; Sidiropoulos et al. 2000). For details on other PLL and DLL building blocks, see, among others, Gardner (1979), Kim et al. (1994), and Razavi (1996). The following section briefly describes some of the most important noise sources and trade-offs involved in PLL and DLL design, and gives a comparative analysis of PLL versus DLL performance.

1.2.3. Noise Sources and Loop Bandwidth

For the purposes of high-level analysis, we divide the noise sources into three main categories: (1) noise of the reference clock, (2) noise induced in the VCO (or VCDL), and (3) noise induced on the clock during distribution from the PLL (DLL) to the CSE, here defined as *clock driver noise*. Since these noise

sources are introduced into the loop at different locations, the transfer functions to the output are different for each of them. For example, input reference noise is low-pass filtered at the output of the PLL, with the filter bandwidth set by the bandwidth of the PLL. On the other hand, input reference noise passes directly through the VCDL to the output of the DLL, without any filtering. Noise induced in the VCO is fed back to the VCO input (in ring-oscillator implementation) and "accumulated" over time (Kim et al. 1994). Any noise induced in the VCO or VCDL is tracked and rejected by the loop, up to the loop bandwidth. Therefore, the transfer function of noise from the VCO (VCDL) to the output is high-pass, contrary to the one from the input reference to the output, which is low-pass. This immediately points to the possible trade-off between the amount of input reference noise and VCO noise at the output of the PLL. Indeed, the optimal bandwidth at which these two noise sources are balanced exists and minimum total noise is achieved (Lim et al. 2000; Mansuri and Yang 2002). In summary, DLLs perform better in cases where the reference clock is not the main source of clock uncertainty and most major noise comes from the noise induced in the VCDL line. PLLs are, however, better in cases where the input reference noise is dominant, and typically worse in cases where the major noise is induced in the VCO, due to the noise accumulation effect, given that compared VCOs and VCDLs are implemented using the same type of delay element.

The preceding analysis is somewhat blurred in modern systems, due to the noise induced in the clock driver. While VCOs and VCDLs are typically implemented using three to seven delay stages, because of the increasing amount of clock load, clock driver depth has increased from generation to generation, and is now over five stages in modern processors. Given that sensitivity of the delay elements in VCO or VCDL is typically an order of magnitude better than that of inverter, which has a 1% delay variation for a 1% power supply variation, it can be easily seen that the overall noise of the distributed on-chip clock is usually dominated by the noise induced in the clock driver tree.

1.2.4. Design Considerations

Regarding the design of the PLLs and DLLs, PLLs are typically harder to design, due to stability issues (PLL is a second-order system due to the integrating function of the VCO), but offer more flexibility than DLLs, that is, wider locking range and, frequency multiplication. DLLs are simpler to design, given that they are first-order systems (unconditionally stable), but offer limited lock range. However, it is true that more complicated DLLs that offer similar flexibility to PLLs are also very complex systems (Sidiropoulos and Horowitz 1997).

PLLs are mostly used in modern processors to multiply the frequency of the external system clock and reject any existing high-frequency reference clock noise. DLLs have recently found application as deskewing elements in high-performance processors, synchronizing different clock domains on a die to the global clock reference from the PLL (Rusu and Tam 2000; Xanthopoulos et al. 2001). It should be noted, however, that these approaches only deal with the DC

portion of the noise on the clock (skew), while AC portion of the noise (jitter) is not eliminated. The jitter induced in the clock driver by power supply variations still presents the dominant source of noise in the on-chip clock distribution and needs to be budgeted for in any clocking methodology.

1.3. TIMING PARAMETERS

It is appropriate at this point to consider the clock distribution system and define the clock parameters that will be used throughout this text. For the purposes of definition we should start with the Fig. 1.16, which shows the timing parameters for a single-phase clock.

The clock signal is characterized by its *period*, T, which is inversely proportional to the *clock frequency*, f. The time during which the clock is active (assuming logic 1 value) is defined as *clock width*, W. The ratio of W/T is defined as *clock duty cycle* (w). Usually, the clock signal has a symmetric shape, which implies a 50% duty cycle. This is also the best we can expect, especially when distributing a high-frequency clock. Another important point is the ability to precisely control the duty cycle. This point is of special importance when each phase of the clock is used for logic evaluation, or when we trigger the clock storage elements on each edge of the clock (as we will see later in the book). Some recently reported work demonstrates the ability to control the duty cycle to within ±0.5% (Bailey and Benschneider 1998).

There are two other important timing parameters that we need to define: *clock skew* and *clock jitter*.

1.3.1. Clock Skew

Clock skew is defined as a *spatial variation* of the clock signal as distributed through the system. The clock skew is measured from some reference point in the system; the clock entry point to the board or VLSI chip, or the central point from where the clock distribution starts. Because of the various delay characteristics of the clock paths to the various points in the system, as well as different loading

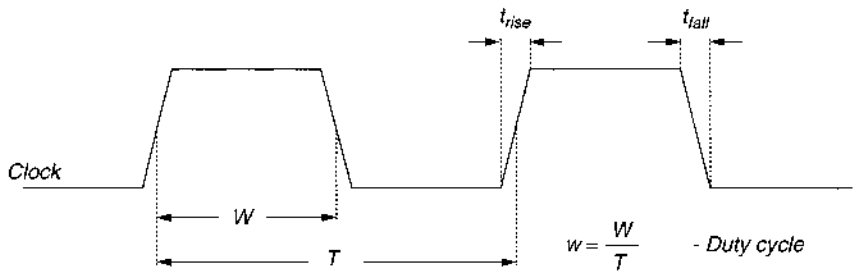

Figure 1.16. Clock parameters: period, width, rise, and fall times.

of the clock signal at different points, the clock signal arrives at different points at different times. This clock skew is defined as the difference between the reference point and the particular destination CSE. Further, we can distinguish *global clock skew* and *local clock skew*. We define global clock skew as the maximal difference between two clock signals reaching any of the two storage elements on the chip, or in the system, that exchange data under the control of the same clock. Our definition of the clock skew describes global clock skew. Clock skew occurring between two adjacent CSEs represents local clock skew. If the two adjacent clock storage elements are connected with no logic in-between, the problem of data race-through can occur. Characterizing a maximum local clock skew is therefore important. These clock skew definitions are equally important in high-performance system design.

1.3.2. Clock Jitter

Clock jitter is defined as *temporal variation* of the clock signal with regard to the reference transition (reference edge) of the clock signal, as illustrated in Fig. 1.17. Clock jitter represents edge-to-edge variation of the clock signal in time. As such, clock jitter can also be classified as *long-term jitter* and *cycle-to-cycle* (or *edge-to-edge*) *jitter*. Edge-to-edge clock jitter is the clock signal

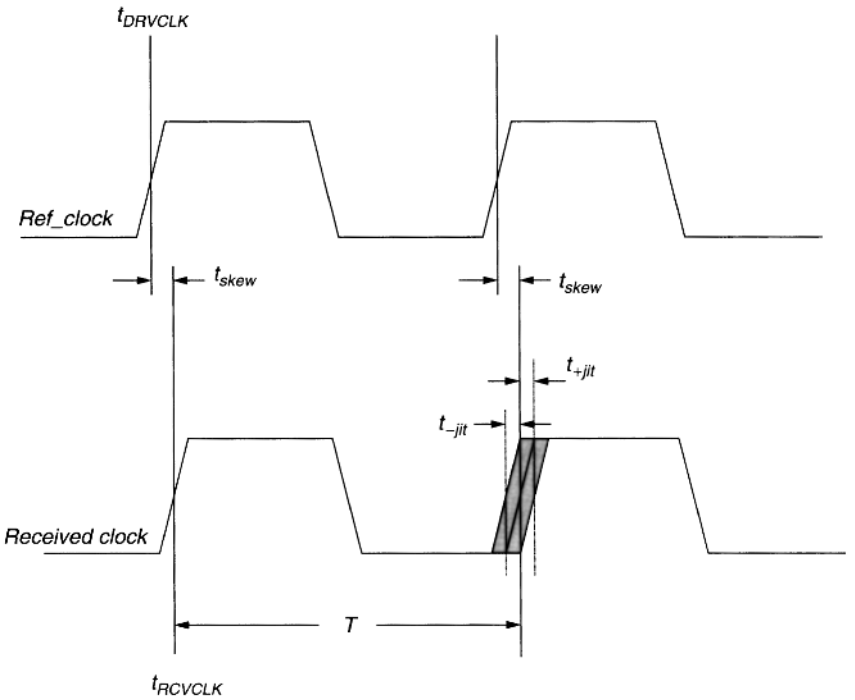

Figure 1.17. Clock parameters: period, width, clock skew, and clock jitter.

variation between two consecutive clock edges. In the course of high-speed logic design, we are more concerned about cycle-to-cycle clock jitter, because it is this phenomena that affects the time available to the logic. Long-term jitter represents clock-edge variation over a large number of clock cycles (long-term). While short-term jitter is dependent on the type and quality of the clock generator, long-term jitter is a result of the accumulated effects. Long-term jitter usually affects communication and synchronization between various blocks within a system that are same distance apart and need to operate in synchrony.

1.4. CLOCK SIGNAL DISTRIBUTION

1.4.1. Historical Overview

Usually a clock signal was generated using a quartz-crystal-controlled oscillator that provides an accurate and stable frequency. Given the size limitation of the quartz crystal, the frequency of such a generated clock signal cannot be very high, and frequencies in excess of 30–50 MHz are rarely generated using a quartz crystal. The clock signal is then conditioned and amplified to reach desirable driving strength before it is applied to the outside pins of a VLSI chip, from which it drives an internal PLL or DLL. Before reaching the boundaries of the VLSI chip, adjustments to its shape and form are possible. In contrast, in older computer systems, which consisted of several electronic cabinets distributed over the computer floor, and which contained a number of printed circuit boards, adjustments to the clock signal were made at each level. Thus, the clock signals were distributed over longer distances and over several levels, including the cabinet, printed circuit boards, and internal modules. Those separate entities entered by the clock signal were referred to as "logic islands," a term introduced by Amdahl (Flynn and Amdahl 1965; Kogge 1981). The concept of logic islands is illustrated in Fig. 1.18.

Figure 1.19 shows that further tuning and delay adjustment of the clock signal is possible at the point where the clock enters the board or cabinet (called an island). Those elements are usually called *tuning points*. The positioning of tuning points in the system is illustrated in Fig. 1.19. Various clock shaping, forming, and tunable delay elements are employed, and some of them are illustrated in Fig. 1.20. These elements make it possible to control the timing of the leading as well as the trailing edge of the clock signal, and to produce an early as well as late clock signal with reference to the nominal clock.

By adjusting the clock delay and subsequently shaping the edges of the clock signal, it is possible to create early, nominal, and late clocks, as shown in Fig. 1.21c. Those clocks then can be routed to various points on the board. Older systems had much greater control of the clock signal than what is possible today, because once the clock reaches the boundary of the LSI chip, tuning and shaping the clock is not possible. This is because it is much more difficult to tune on the chip due to the lack of external control and greater parameter variations

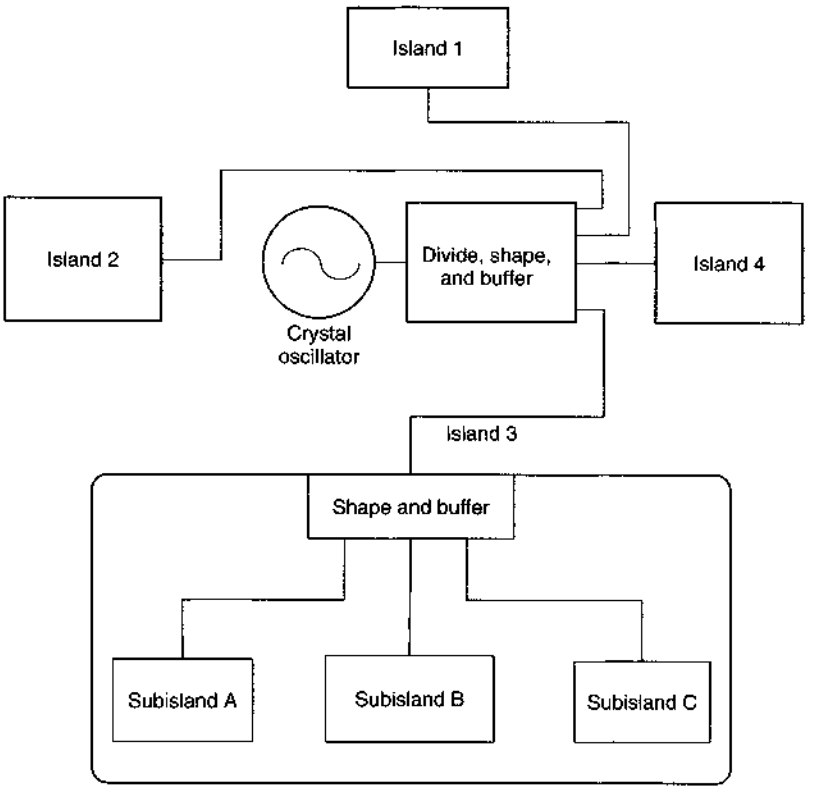

Figure 1.18. The concept of logic islands. (Wagner 1988), Copyright © 1988, IEEE.

on the chip. It is also difficult to build tuning elements such as inductors on the chip and to make adjustments from outside.

With the advent of integration, the systems have shrunk dramatically in size. Today, it is quite common for a processor to have several levels of cache memory contained entirely on a VLSI chip. The chip's capacity for hundreds of millions of transistors makes it possible to integrate not only one processor but also a multiprocessor system onto a single chip. The inability to introduce tuning elements on the chip further aggravates the problem of distributing the clock signals precisely in time, since it is not possible to make further manual adjustment to the clock signal once it has crossed the boundaries of the VLSI chip. Therefore, careful planning and design of the on-chip clock distribution network is one of the most critical tasks in high-performance processor design.

1.4.2. Clock Distribution in Modern Microprocessors

Typically, the clock signal has to be distributed to several hundreds of thousands of the clocked storage elements (flip-flops and latches) on a complex processor

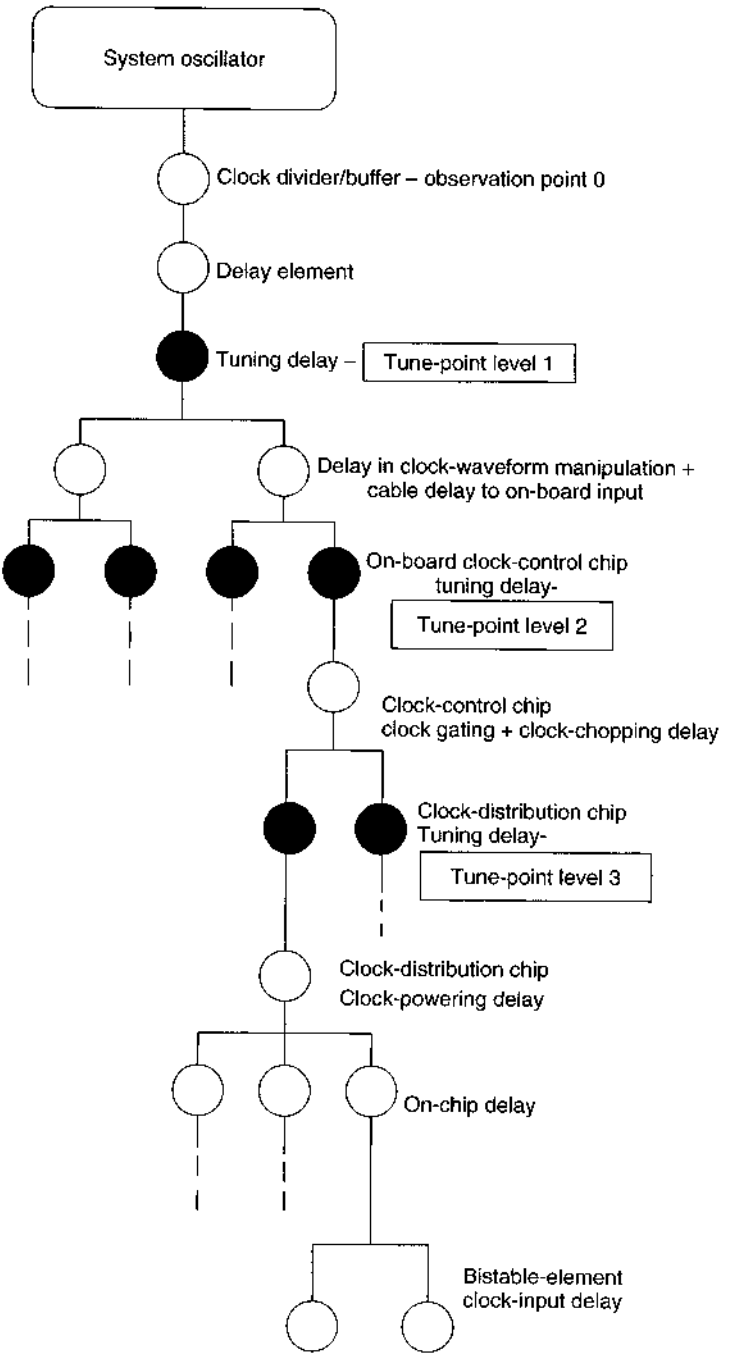

Figure 1.19. Clock tuning points. (Wagner 1988), Copyright © 1988 IEEE.

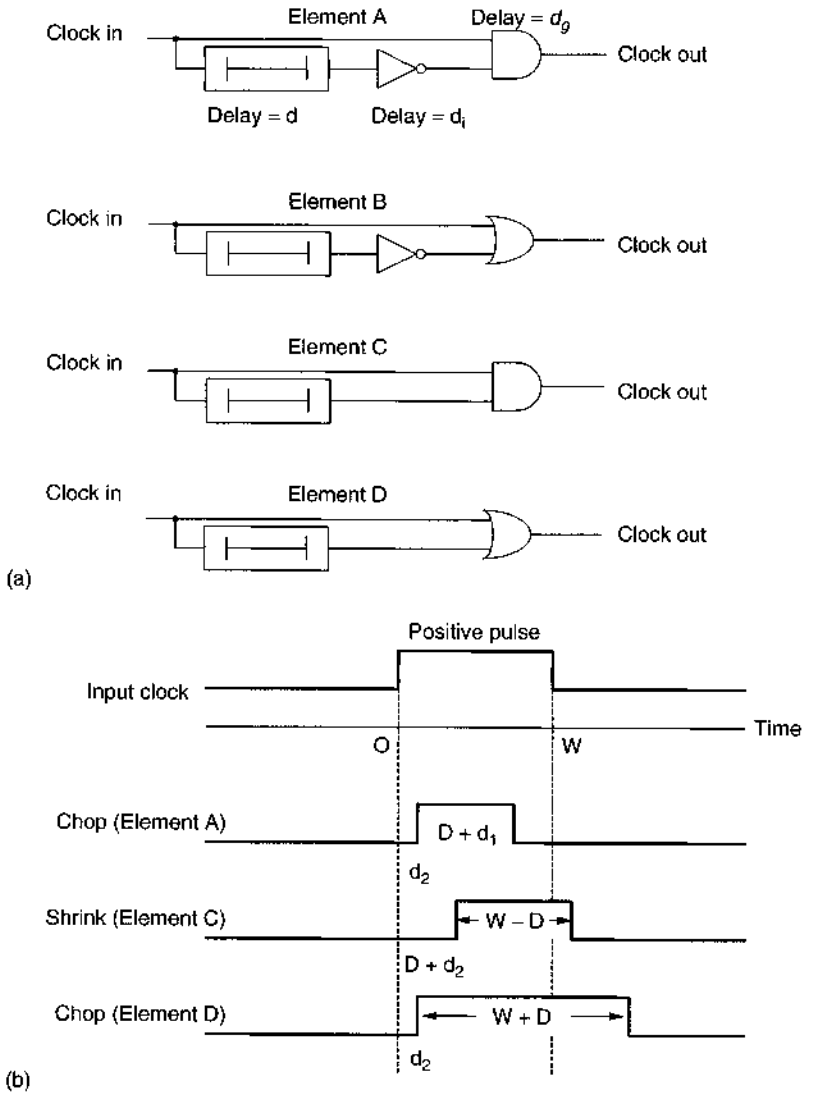

Figure 1.20. Various clock shaping elements and obtained clock signals. (Wagner 1988), Copyright © 1988 IEEE.

chip. Therefore, the clock signal has the largest fan-out of any node in the design, which requires several levels of amplification (buffering). One consequence of imposing such a load on the clock signal is that the clock system by itself can use up to 40–50% of the power of the entire VLSI chip (Gronowsky et al. 1998). However, power is not the only problem associated with the distribution of the clock signals. Since we are dealing with synchronous systems, we must assure that every clocked storage element receives the clock signal at precisely

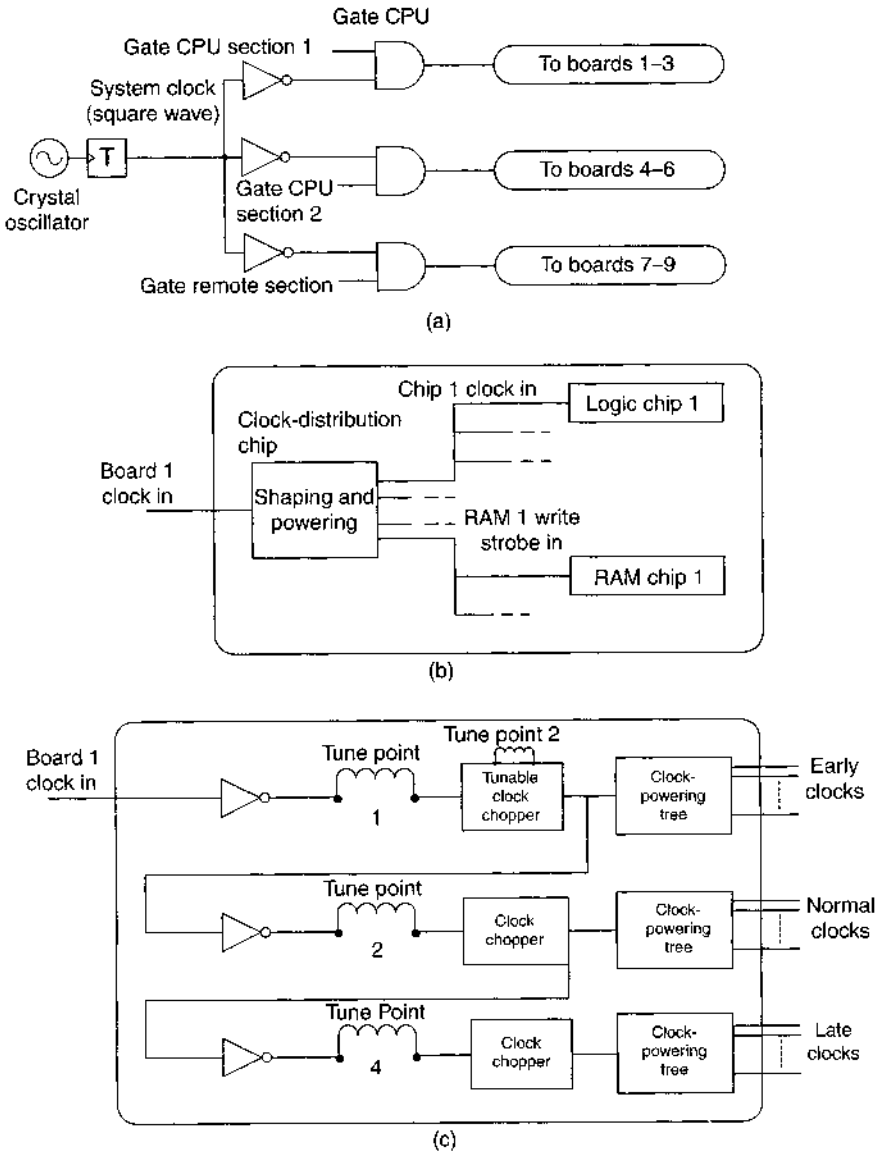

Figure 1.21. (a) Clock distribution network within a system, (b) on the board, and (c) tuning of the clock. (Wagner 1988), Copyright © 1988 IEEE.

the same moment. The clock signal traverses different paths on the VLSI chip, while tracing its path from its origin, the entry point to the VLSI chip, to different clocked storage elements receiving it. Those paths can differ in several attributes, such as the length of the path (wire), the physical properties of the material along different paths, the differences in clock buffers on the chip as a consequence of

the process variations. The negative effect of these variations on the synchronous design is that different points on the chip will receive the clock signal at different moments, which results in a further increase in both local and global clock uncertainties.

There are several methods for the on-chip clock signal distribution that attempt to minimize the clock skew and to contain the power dissipated by the clock system. The clock can be distributed in several ways, two of which are worth considering here: (1) resistance–capacitance (RC) matched tree shown in Fig. 1.22a, and (2) the grid shown in Fig. 1.22b.

An RC matched tree is a method of assuring (to the best of our abilities) that all the paths in the clock distribution tree have the same delay, which includes the same RC of the wire, as well as the same number of equal-size buffers on the clock signal path to the storage element. There are several different topologies used to implement an RC matched tree. The common objective is to do the best possible in balancing various clock signal paths across the various points on the VLSI chip. An example of four different topologies, as taken from Bailey (Chandrakasan et al. 2001), is shown in Fig. 1.23.

If we had superior computer-aided design (CAD) tools, a perfect and uniform process, and the ability to route wires and balance loads with a high degree of flexibility, a matched RC delay clock distribution (Fig. 1.23) would be preferable to grid (b) as shown in Fig. 1.22b and Fig. 1.24. However, none of that is true. Therefore the grid is used when clock distribution on the chip has to be very precisely controlled, which results in higher clock power, as is the case in high-performance systems. This is not difficult to understand given that in a grid arrangement a high-capacitance plate is driven by buffers connected at various points.

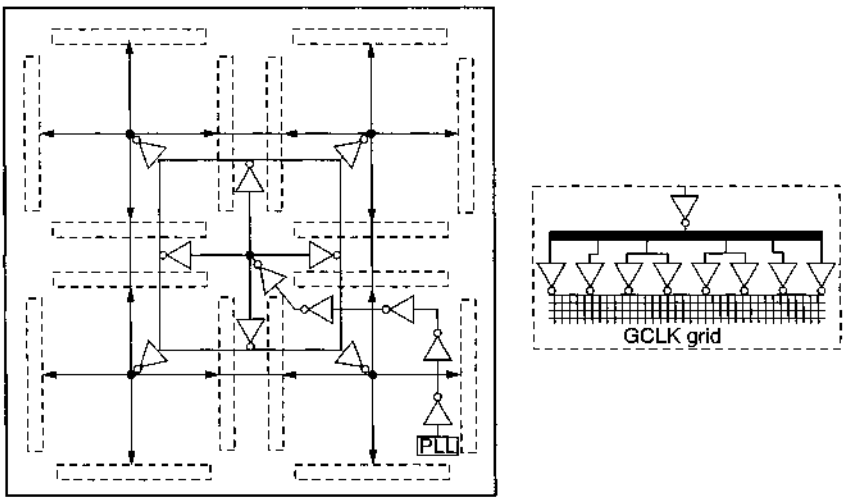

Figure 1.22. Clock distribution methods: (a) an RC matched tree, and (b) a grid. (Bailey and Benschneider 1998), Copyright © 1998 IEEE.

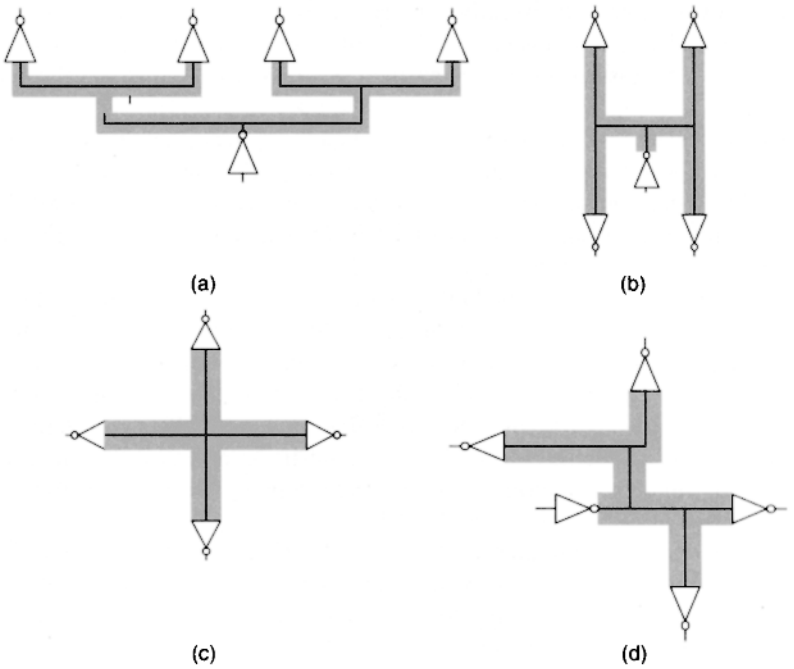

Figure 1.23. *RC* delay matched clock distribution topologies: (a) a binary tree (b); an H tree; (c) an X tree; (d) an arbitrary matched *RC* matched tree. (From Bailey in Chandrakasan et al. 2001), Copyright © 2001 IEEE.

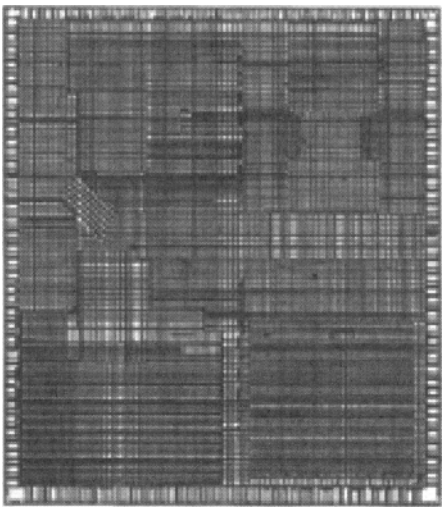

Figure 1.24. Clock distribution grid used in a DEC Alpha 600-MHz processor. (Bailey and Benschneider 1998), Copyright © 1998 IEEE.

One such example is the DEC Alpha processor, which was the fastest processor for several generations of microprocessors starting with the first 200-MHz design introduced in 1992 and ending with the 600-MHz design in 1998 (Dobberpuhl et al. 1992; Benschneider et al. 1995; Gieske et al. 1997). A picture of the clock distribution grid is shown in Fig. 1.24.

With an increased number of transistors, local variations in device geometry and supply voltage become a more important component of the clock uncertainty, which cannot be compensated for by layout (Schutz and Wallace 1998). A more sophisticated clock distribution than simple *RC*-matched or grid-based schemes is therefore necessary. One such example will be described in Chapter 9 of this book. The active schemes with adaptive digital deskewing typically reduce clock skew of the simple passive clock networks by an order of magnitude, allowing for more tightly controlled clock period and higher clock rates. The digital deskewing circuit for clock distribution evens out the *static components of skew* (load, interconnect, and device mismatches). Additionally, it compensates for the *dynamic* variations in temperature and voltage gradients during all phases of active microprocessor operation.