

Synchronization in Digital System Design

DAVID G. MESSERSCHMITT, FELLOW, IEEE

(Invited Paper)

Abstract—In digital system design, synchronization ensures that operations occur in the logically correct order, and is a critical factor in ensuring the correct and reliable system operation. As the physical size of a system increases, or as the speed of operation increases, synchronization plays an increasingly dominant role in the system design. Digital communication has developed a number of techniques to deal with synchronization on a global and even cosmic scale; and as the clock speeds of chip, board, and room-sized digital systems increase, they may benefit from similar techniques. Yet, the digital system and digital communication communities have evolved synchronization techniques independently, choosing different techniques and different terminology. In this paper, we attempt to present a unified framework and terminology for synchronization design in digital systems, borrowing techniques and terminologies from both digital system and digital communication design disciplines. We then compare the throughput of synchronous and asynchronous interconnect, emphasizing how it is impacted by interconnect delay. Finally, we discuss opportunities to apply principles long employed in digital communications to the design of digital systems, with the goal of reducing this dependence on interconnect delay.

I. INTRODUCTION

OPERATIONS in digital systems can either proceed concurrently, or they must obey a precedence relationship. If two operations obey a precedence, then the role of synchronization is to ensure that the operations follow in the correct order. Synchronization is thus a critical part of digital system design.

The most common approach to synchronization is to distribute a clock signal to all modules of the system. With the scaling of feature-sizes in VLSI design, clock speeds are increasing rapidly, but increases in complexity tend to prevent significant reductions in chip size. As a consequence of this scaling, clock speeds in digital system designs are increasing in relation to propagation delays. This is causing increasing problems with the traditional synchronous design methodologies, certainly at the system and board levels, and increasingly even within high performance chips [1]. This problem will be accentuated with the more common application of optics to system interconnection.

Problems such as large propagation delay have been faced since the earliest days of digital communication sys-

tem design, and hence there are a number of opportunities to apply digital communications principles to VLSI and digital system design. However, because of the wide difference between system physical sizes in relation to clock speeds, the design styles of the communities have developed almost independently. This is in spite of the fact that digital design is a necessary element of digital communication design (for example, in modems, switches, etc.).

In this paper, we place the synchronization problem and design approaches in digital communication and digital system design in a common framework, and then examine opportunities for cross-fertilization between the two fields. In attaining the unification of design methodologies that we attempt in this paper, the first difficulty we face is the inconsistencies, and even contradictions, between terms as used in the two fields. For example, the term "self-timed" generally means "no independent clock," but as used in digital communication it means no clock at all (in the sense that if a clock is needed it can be derived from the data), and in digital system design it indicates that there is a clock signal that is slaved to the data rather than being independent. Therefore, in Section II we attempt to define a taxonomy of terminology that can apply to both fields, while retaining as much of the terminology as presently used as possible. We discuss this terminology in terms of the necessary levels of abstraction in the design process. In Section III we discuss the synchronization techniques commonly used in digital systems and digital communications, relate the two, and compare the fundamental limitations they place on throughput. In Section IV we discuss specifically how some synchronization techniques from digital communication might be beneficial in digital system design, particularly in reducing dependence of throughput on interconnect delay. Finally, in Section V we discuss briefly the interrelationship between architectural design and synchronization in digital systems. This discussion expands on an earlier conference paper [2].

II. ABSTRACTIONS IN SYNCHRONIZATION

A basic approach in system design is to define *abstractions* that enable the designer to ignore unnecessary details and focus on the essential features of the design. While every system is ultimately dependent on underlying physical laws, it is clear that if we relied on the solution of Maxwell's equations at every phase of the design, systems could never get very complex. Abstractions are often applied in a hierarchical fashion, where each layer of ab-

Manuscript received January 19, 1990. This work was supported by the National Science Foundation, the California MICRO Program, Rockwell Semiconductor, Level One, and Texas Instruments. This paper was presented at the 6th IEEE International Workshop on Microelectronics and Photonics in Communications, New Seabury, Cape Cod, MA, June 6-9, 1989.

The author is with the Department of Electrical Engineering and Computer Science, University of California, Berkeley, CA 94720.

IEEE Log Number 9036205.

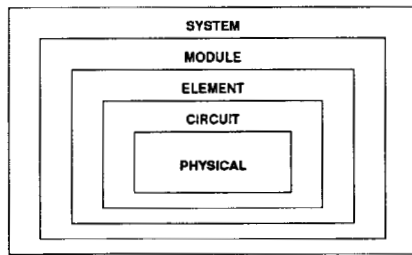


Fig. 1. An example of some abstractions applied to digital system design.

straction relies on the essential features of the abstraction level below, and hides unessential details from the higher level. This as illustrated for digital system design in Fig. 1. At the base of the design, we have the physical representation, where we have semiconductor materials, interconnect metalization, etc., and we are very concerned about the underlying physical laws that govern their properties. Above this, we have the circuit abstractions, where we deal with circuit entities such as transistors, interconnections, etc. Our descriptions of the circuit entities attempt to ignore the details of the physical laws underlying them, but rather characterize them in terms that emphasize their operational properties, such as current-voltage curves, transfer functions, etc. Above the circuit abstraction is the element, which groups circuit entities to yield slightly higher functions, such as flip-flops and gates. We describe elements in terms that deal with their operational characteristics (timing diagrams, Boolean functions, etc.) but ignore the details of how they are implemented. Above the element abstraction, we have the module (sometimes called "macrocell") abstraction, where elements are grouped together to form more complex entities (such as memories, register files, arithmetic-logic units, etc.). Finally, we group these modules together to form systems (like microcomputers).

Many other systems of abstractions are possible, depending on circumstances. For example, in digital communication system protocols, we extend the abstractions in Fig. 1, which carry us to the extreme of hardware design, to add various abstractions that hierarchically model the logical (usually software-defined) operation of the system (including many layers of protocols). Similarly, the computer industry defines other system abstractions such as instruction sets, operating system layers, etc.

In the present paper, we are concerned specifically with synchronization in the design of digital systems and digital communications. In the context of digital systems, by *synchronization* we mean the set of techniques used to ensure that operations are performed in the proper order. The following subsections define some appropriate abstractions for the synchronization design. While these abstractions are by no means new, perhaps this is the first time that a systematic treatment of them has been attempted. This systematic treatment gives us a common base of terminology for synchronization design, and is utilized in the following subsections.

While abstractions are very useful, and in fact abso-

lutely necessary, they should be applied with care. The essence of an abstraction is that we are ignoring some details of the underlying behavior, which we hope are irrelevant to the operation, while emphasizing others that are most critical to the operation. It should always be verified that in fact the characteristics being ignored are in fact irrelevant, and with considerable margin, or else the final system may turn out to be inoperative or unreliable. This is especially true of synchronization, which is one of the most frequent causes of unreliable operation of a system. In the following, we therefore highlight behaviors that are ignored or hidden by the abstraction.

A. Some Basic Synchronization Abstractions

1) *Boolean Signals*: A *Boolean signal* (voltage or current) is assumed to represent, at each time, one of two possible levels. At the physical level, this signal is generated by saturating circuits and bistable memory elements. There are a couple of underlying behaviors that are deliberately ignored in this abstraction: *finite rise-time* and the *metastable* behavior of memory elements.

Finite rise-time behavior is illustrated in Fig. 2 for a simple *RC* time constant. The deleterious effects of rise-time can often be bypassed by the *sampling* of the signal. In digital systems this is often accomplished using edge-triggered memory elements. In digital communications, rise-time effects are often much more severe because of the long distances traversed, and manifest themselves in the more complex phenomenon of *intersymbol interference* [3]. In this case, one of several forms of equalization can precede the sampling operation.

Metastability is an anomalous behavior of all bistable devices, in which the device gets stuck in an unstable equilibrium midway between the two states for an indeterminate period of time [4]. Metastability is usually associated with the sampling (using an edge-triggered bistable device) of a signal whose Boolean state can change at any time, with the result that sampling at some point very near the transition will occasionally occur. Metastability is less severe a problem than rise-time in the sense that it happens only occasionally, but more severe in that the condition will persist for an indeterminate time (like a rise-time which is random in duration).

The Boolean abstraction is most valid when the rise-time is very much shorter than the interval between transitions, and metastability is avoided or carefully controlled. But the designer must be sure that these effects are negligible, or unreliable system operation will result.

2) *Signal Transitions*: For purposes of synchronization we are often less concerned with the signal level than with the times at which the signal changes state. In digital systems this time of change would be called an *edge* or transition of the signal. The notion of the transition time ignores rise-time effects. In fact, the transition time is subject to interpretation. For example, if we define the transition time as the instant that the waveform crosses some slicer level (using terminology from digital communication), then it will depend on the slicer level as illustrated in Fig. 3. Even this definition will fail if the



Fig. 2. Illustration of a digital signal with a finite rise-time as generated by an RC time-constant, where the Boolean signal abstraction is shown below.

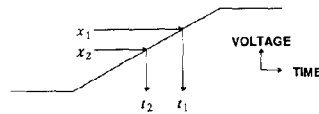


Fig. 3. The transition time t_r depends on slicer level x_s .

waveform is not monotonic in the region of the slicer level.

Transition times are a useful abstraction for the case where the rise times are very short in relation to the interval between transitions, with the result that the variation in the transition time is negligibly small over the set of all possible definitions of the transition time. Rise-time is governed by underlying physical phenomena, such as transmission line dispersion, and can be reduced by using wider bandwidth drivers or intermediate repeaters. As system clock rates increase, however, for a given interconnect style the behavior ignored by this abstraction inevitably becomes important.

The transition abstraction can be extended to the notion of *uniformly spaced transitions*. For example, a clock signal can be modeled as a square wave, in which the transitions alternate in sign, and each adjacent pair of transitions (called a *cycle*) represents a time equal to the reciprocal of the clock *frequency*. For a data signal, transitions may or may not be present depending on the Boolean data (see Fig. 2), so we have to introduce the notion of the times where transitions *might* occur, called a *transition opportunity*, whether they actually occur or not. A data signal whose transitions are slaved to a clock with uniformly spaced transitions then has uniformly spaced transition opportunities (an example is shown in Fig. 4). We can think of these transitions as being associated with a clock that has positive transitions at identical times, which we call the *associated clock*, whether or not such a clock signal exists physically.

Uniformly spaced transitions ignore possible jitter effects in the generation or transmission of the Boolean signals, which often result in small variations in the times between transitions. Hence, there is the need to define the concepts of instantaneous phase and frequency.

3) *Phase and Frequency*: For a Boolean signal, we can define a phase and frequency of the signal as the phase and frequency of the associated clock. It is convenient to describe mathematically a clock signal with uniformly spaced transitions as

$$x(t) = p((ft + \phi) \text{ modulo } 1), \quad (1)$$

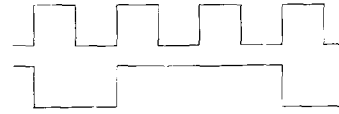


Fig. 4. A Boolean signal (below) and associated clock (above). The transitions of the Boolean signal are slaved to the positive transitions of the associated clock.

where $p(t)$ is a 50% duty cycle pulse

$$p(t) = \begin{cases} 1, & 0 \leq t < 0.5 \\ 0, & 0.5 \leq t \leq 1, \end{cases} \quad (2)$$

f is the *nominal frequency*, and ϕ is the *phase*. As ϕ varies over the range $0 \leq \phi < 1$, the transitions are shifted in time over one cycle. The phase is thus expressed as the fraction of a cycle. When we have two Boolean signals, the *relative phase* can be expressed as the phase difference $(\phi_1 - \phi_2)$ between their respective associated clocks.

A more general model that includes more possible effects replaces (1) by

$$x(t) = p(((f + \Delta f)t + \phi(t)) \text{ modulo } 1), \quad (3)$$

where f is the *nominal frequency* of the associated clock, Δf is a possible *offset* in the nominal frequency, and $\phi(t)$ is the *instantaneous phase variation* versus time. The intention here is that $\phi(t)$ does not embody a frequency offset, but rather any offset from the nominal frequency is summarized by Δf . The precise mathematical conditions for this are complicated by the modulo operation, and also depend on the model for $\phi(t)$ (deterministic signal, random process, etc.). For example, if $\phi(t)$ is assumed to be a deterministic differentiable and continuous function (with no phase jumps), then it suffices for $\phi(t)$ to be bounded,

$$\phi(t) \leq \phi_{\max}, \quad (4)$$

and for such a function the derivative (instantaneous frequency) must average to zero,

$$\frac{d\phi(t)}{dt} = 0 \quad (5)$$

(where the average is interpreted as a time average).

The model of (3) makes the assumption that the average frequency is a constant, although that average frequency may not be known *a priori* (for example, when it depends on the free-running frequency of an oscillator). Such a signal is said to be *isochronous* (from "iso," the Greek root for "equal"), whereas if the frequency is not constant (Δf is actually a function of time), the signal is said to be *anisochronous* (or "not equal"). An anisochronous signal can be modeled using (3), but the resulting phase will not be bounded. Thus, the essential difference between isochronous and anisochronous signals is the bounded phase condition of (4).

The time-varying phase in (3) is crucial where we cannot ignore small variations in the intervals between tran-

sitions, known as *phase jitter*. This jitter is usually ignored in digital system design, but becomes quite significant in digital communications, especially where the Boolean signal is passed through a chain of regenerative repeaters [3].

Directly following from the concept of phase is the *instantaneous frequency*, defined as the derivative of the instantaneous phase,

$$f(t) = f + \Delta f + \frac{d\phi(t)}{dt}. \quad (6)$$

From (5), the *instantaneous frequency deviation* $d\phi(t)/dt$ has mean value zero, so that $(f + \Delta f)$ is the *average frequency*.

Given two signals $x_i(t)$, $i = 1, 2$ with the same nominal frequency, and frequency and phase offsets Δf_i and $\phi_i(t)$, the *instantaneous phase difference* between the two signals is

$$\Delta\phi(t) = (\Delta f_1 - \Delta f_2)t + (\phi_1(t) - \phi_2(t)). \quad (7)$$

4) *Synchronism*: Having carefully defined some terms, we can now define some terminology related to the *synchronization* of two signals. A taxonomy of synchronization possibilities is indicated in Fig. 5 [5], [3]. As previously mentioned, a Boolean signal can be either isochronous or anisochronous. Given two data signals, if both are isochronous, the frequency offsets are the same, and the instantaneous phase difference is zero,

$$\Delta\phi(t) = 0 \quad (8)$$

then they are said to be *synchronous* (from the Greek “syn” for “together”). Common examples would be a Boolean signal that is synchronous with its associated clock (by definition), or two signals slaved to the same clock at their point of generation. Any two signals that are not synchronous are *asynchronous* (or “not together”). Some people would relax the definition of synchronous signals to allow a nonzero phase difference that is *constant* and *known*.

In practice, we have to deal with several distinct forms of asynchrony. Any signal that is anisochronous will be asynchronous to any other signals, except in the special and unusual case that two anisochronous signals have identical transitions (this can happen if they are co-generated by the same circuit). Thus, anisochrony is a form of asynchrony.

If two isochronous signals have exactly the same average frequency $f + \Delta f$, then they are called *mesochronous* (from the Greek “meso” for “middle”). For mesochronous signals, the fact that each of their phases is bounded per (4) ensures also that the phase difference is also bounded,

$$\Delta\phi(t) \leq 2\phi_{\max}, \quad (9)$$

a fact of considerable practical significance. Two signals generated from the same clock (even different phases of the same clock), but suffering indeterminate interconnect delays relative to one another, are mesochronous.

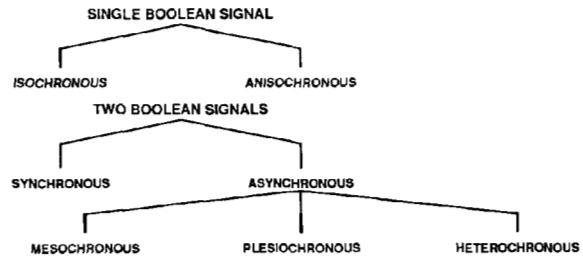


Fig. 5. A taxonomy of synchronization approaches.

Two signals that have average frequencies that are nominally the same, but not exactly the same (usually because they are derived from the independent oscillators), are *plesiochronous* (from the Greek “plesio” for “near”). Suppose the nominal frequencies are both f , but the actual frequencies are $f + \Delta f_1$ and $f + \Delta f_2$, then the instantaneous phase difference is

$$\Delta\phi(t) = (\Delta f_1 - \Delta f_2)t + (\phi_1(t) - \phi_2(t)), \quad (10)$$

where the first term increases (or decreases) linearly with time. For two plesiochronous signals, it cannot be predicted which has the higher frequency.

Finally, if two signals have *nominally different* average frequencies, they are called *heterochronous* (from the Greek “hetero” for “different”). Usually the tolerances on frequency are chosen to guarantee that one signal will have an actual frequency guaranteed higher than the other (naturally the one with the higher nominal rate). For example, if they have nominal frequencies f_1 and f_2 , where $f_1 < f_2$, and the worst-case frequency offsets are $|\Delta f_1| \leq \eta_1$ and $|\Delta f_2| \leq \eta_2$, then we would guarantee this condition if f_1 and f_2 were chosen such that

$$f_1 + \eta_1 < f_2 - \eta_2. \quad (11)$$

B. Additional Timing Abstractions in Digital Systems

This section has covered some general considerations in the modeling of Boolean signals from a synchronization perspective. In this subsection we describe a couple of additional abstractions that are sometimes applied in digital system design, and represent simplifications due to the relatively small physical size of such systems.

1) *Equipotential Region*: Returning to Fig. 1, it is often assumed at the level of the element abstraction that the signal is identical at all points along a given wire. The largest region for which this is true is called (by Seitz [6]) the *equipotential region*. Like our other models, this is never strictly valid, but is useful if the actual time it takes to equalize the potential along a wire is small in relation to other aspects of the signals, such as the associated clock period or rise-time. The equipotential region is a useful concept in digital system design because of the relatively small size of such systems. However, the element dimensions for which it is valid is decreasing because of increases in clock frequency with scaling, and a single chip

can generally no longer be considered an equipotential region.

2) *Ordering of Signals:* In the design of digital systems, it is often true that one Boolean signal is *slaved* to another, so that at the point of generation the one signal transitions can always be guaranteed to precede the other. Conversely, the correct operation of circuits is often dependent on the correct ordering of signal transitions, and quantitative measures such as the minimum time between transitions. One of the main reasons for defining the equipotential region is that if a given pair of signals obey an ordering condition at one point in a system, then that ordering will be guaranteed anywhere within the equipotential region.

III. SYNCHRONIZATION

The role of synchronization is to coordinate the operation of a digital system. In Section III-A and B, we review two traditional approaches to synchronization in digital system design: synchronous and anisochronous interconnection. In Section III-C, we briefly describe how synchronization is accomplished in digital communication systems. This will suggest, as discussed further in Section IV, opportunities to use digital communication techniques in digital system design.

A. Synchronous Interconnection

As shown in Fig. 6, each element (or perhaps module) is provided a clock, as well as one or more signals that were generated with transitions slaved to the clock. The common clock controls the order of operations, ensuring correct and reliable operation of the system.

We will first, in Section III-A-1), examine some fundamental limitations in the operation of synchronous interconnection, making idealistic assumptions about the ability to control the clock phases in the system and neglecting interconnect delays. In Section III-A-2), we will show how pipeline registers can be used to extend the performance of synchronous interconnect; and in Section III-A-3), we will make more realistic estimates of performance considering the effects of the inevitable variations in clock phase and interconnect delays.

1) *Principle of Synchronous Interconnection:* The fundamental principle of synchronous interconnection is illustrated in Fig. 7. In Fig. 7(a) a *computational block* C1 is connected to a *synchronizing register* R1 at its input. This register is *clocked* using the positive transitions of a periodic clock signal, where the assumption is that the output signal of the register changes synchronously with the positive transition of the clock. The computational block performs the same computation repeatedly on new input signals applied at each clock transition. The purpose of R1 is to control the time at which the computational block starts to perform its work, in order to synchronize it to other computational blocks in the system. This is an *edge-triggered* logic model, which we employ for its relative simplicity. There is also a more complicated *level-*

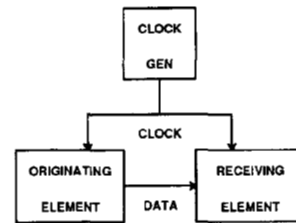


Fig. 6. Synchronous interconnection, in which a common clock is used to synchronize computational elements.

sensitive model that leads to virtually identical conclusions.

The performance measures of interest in Fig. 7 are the *throughput* (rate at which the computation is repeated) and *computational latency* (delay from the time a new input is applied until the result is available). Focusing on the latter, inevitably the computational latency is not entirely predictable. It is likely that the output signal will change more than once before it finally settles to its final correct value. For example, if the output signal actually consists of $M > 1$ Boolean signals in parallel, as is often the case, some of those Boolean signals may transition before others, or some may transition more than once before reaching a steady-state value. This behavior is an inevitable consequence of real circuit implementation of the computational block, and presents a considerable problem in the synchronization to other computational blocks. Assume the computational block has a minimum time before *any* outputs transition, called the *propagation time* t_p , and a maximum time before all the outputs reach their final and correct values, called the *settling time* t_s . Since settling time is the maximum time before the result is guaranteed to be available, it is also the computational latency. It is assumed that the propagation and settling times can be characterized and ensured over expected processing variations in the circuit fabrication.

The synchronous interconnect isolates the system behavior from these realities of circuit implementation by setting the clock period T so that there is a *certainty period* during which the output signal is guaranteed to be correct, and then samples the output signal again (using another register R2) during this certainty period as shown in Fig. 7(b). Of course, the phase of the clock for R2 must be adjusted to fall in this certainty period. This interconnect is called synchronous because the proper phase of clock to use at R2 must be known in advance—there cannot be any substantial uncertainty or time variation in this phase. With synchronous interconnect, the undesired behavior (multiple signal transitions and uncertain completion time) is hidden at the output of R2. We can then abstract the operation of the computational block, as viewed from the output of R2, as an element that completes its computation precisely at the positive transitions of the second clock, and this makes it easy to synchronize this block with others since the uncertainty period has been eliminated.

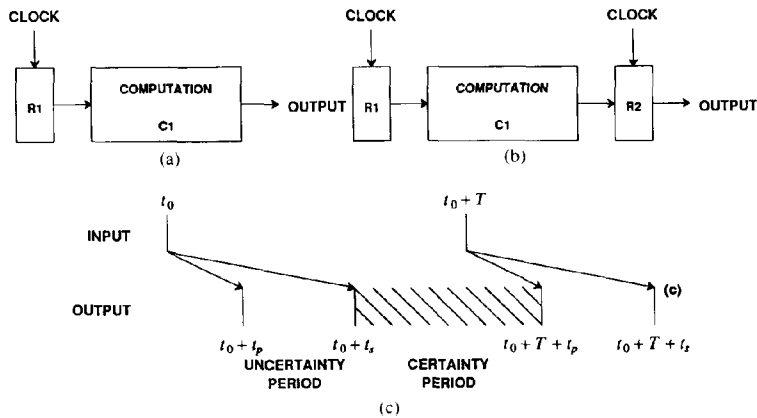


Fig. 7. Synchronizing a computation C1 by latching its input. (a) A register R1 at the input controls the starting time for the computation. (b) A second register R2 at the output samples the output signal during the certainty period. (c) Timing diagram showing the uncertainty and certainty period (crosshatched) as a function of the setting time t_s , the propagation time t_p , and the clock period T .

Given the propagation and settling times, the certainty period is shown as the crosshatched period in Fig. 7(c). The region of time not crosshatched, during which the signal may possibly be changing, is known as the *uncertainty period*. If two successive clock transitions come at times t_0 and $(t_0 + T)$, the certainty period starts at time $(t_0 + t_s)$, the time when the steady-state output due to the computation starting at time t_0 is guaranteed, and ends at time $(t_0 + T + t_p)$, which is the earliest that the output can transition due to the new computation starting at time $(t_0 + T)$. The length of the certainty period is $(T + t_p - t_s)$. An acceptable register clocking phase for R2 requires that this period must be positive in length, or

$$T > t_s - t_p. \tag{12}$$

Alternatively, we can define the *throughput* of computational block C1 as the reciprocal of the clock period, and note that this throughput is upper-bounded by

$$\frac{1}{T} < \frac{1}{t_s - t_p}. \tag{13}$$

The length of the uncertainty period $(t_s - t_p)$ is a fundamental property of the computational block, and the maximum throughput is the reciprocal of this length. In contrast, the length of the certainty period depends on the clock period T , and the goal is generally to make this length as small as practical by choosing T small.

The maximum throughput is dependent only on the length of uncertainty period, $(t_s - t_p)$, and not directly on the settling time t_s . In Fig. 8, an example is given for throughput much higher than the reciprocal of the settling time (because the uncertainty time is a small fraction of the settling time). In Fig. 8, before each computation is completed, two more computations are initiated. At any

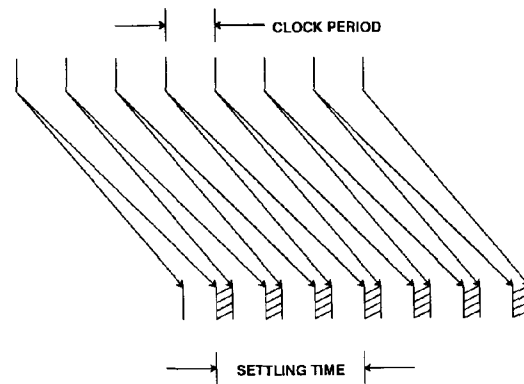


Fig. 8. An example of a synchronous interconnect with a clock period much smaller than the reciprocal of the settling time, due to a small uncertainty period.

point in time, there are three concurrent computations in progress.

The number of concurrent computations is limited only by the inevitable uncertainty in the computational latency. We can give three examples that illustrate a range of possibilities.

Example 1: Consider a fiber optic digital communication system, where the "computational block" is not a computation at all but rather a propagation through a guided medium. For this case, due to propagation dispersion effects, t_p and t_s are not identical, but close enough that throughputs in the range of 10^{10} bits/s are possible. Assuming a group velocity of 10^8 m/s, and a fiber length of 50 km, the settling time is 0.5 ms. At a conservative bit rate of 100 mb/s, there are 50 000 bits propagating through the fiber at any time ("concurrent computations" in the language above). At this velocity and bit rate, the maximum distance for which there is no concurrency in the communication medium is 1 m. \square

Example 2: For typical practical Boolean logic circuits, designed to minimize the settling time rather than maximize the propagation time, t_p is typically very small, and concurrent computations within the computational block are not possible. The maximum throughput is the reciprocal of the settling time. \square

Example 3: Consider a hypothetical (and perhaps impractical) circuit technology and logic design strategy which is designed to achieve $t_p \approx t_s$. In this case, the throughput can be much higher than the reciprocal of the settling time, and many concurrent computations within the computational block are possible. \square

While Example 3 is not likely to be achieved, Examples 2 and 3 suggest the possibility of designing circuits and logic to minimize the uncertainty period ($t_s - t_p$) (even at the expense of increasing t_s) rather than minimizing t_s as is conventional. For example, one could ensure that every path from input to output had the same number of gates, and carefully match the gate settling times. In such a design style, the throughput could be increased to exceed the reciprocal of the settling time. This has recently been considered in the literature, and is called *wave pipelining* [7].

2) *Pipelining:* The form of concurrency associated with Fig. 8 is known as *pipelining*. A useful definition of pipelining is the ability to *initiate a new* computation at the input to a computational block prior to the *completion* of the *last* computation at the output of that block. Since this results in more than a single computation in process within the block at any given time, pipelining is a form of concurrency, always available when $t_p > 0$. The number of *pipeline stages* is defined as the number of concurrent computations in process at one time. For example, if we take the liberty of calling the fiber propagation in Example 1 a "computation," then the fiber has 50 000 pipeline stages.

In conventional digital system design, t_p for computational blocks is typically small, and pipelining requires the addition of *pipeline registers*. To see this potential, make the idealistic assumption that the computational block of Fig. 7 can be split into N subblocks, the output of each connected to the input of the next, where each subblock has a propagation time of t_p/N and a settling time of t_s/N . If this is possible, then the block can be pipelined by inserting $(N - 1)$ pipeline registers between each pair of these subblocks as shown in Fig. 9. Each of these registers, according to the analysis, can use a clock frequency of $N/(t_s - t_p)$ because the uncertainty period is correspondingly smaller, assuming that each clock phase is adjusted to fall within the certainty period relative to the last clock phase.

To see that the uncertainty period is reduced for Fig. 9, in Fig. 7 the middle of the certainty period is delayed relative to the clock time t_0 by $(t_p + t_s + T)/2$. Assuming that the clock phase for each pipeline register in Fig. 9 is chosen in the middle of this certainty period, then relative to the previous clock phase the delay is N times smaller, or $(t_p + t_s + T)/2N$. The total propagation and

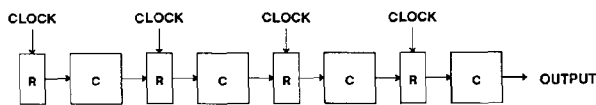


Fig. 9. Pipelining of Fig. 7 for $N = 4$ subblocks and $(N - 1 = 3)$ intermediate pipeline registers.

settling times for the pipeline are then

$$t_{p, \text{pipeline}} = (N - 1) \cdot \frac{t_p + T + t_s}{2N} + \frac{t_p}{N} \quad (14)$$

$$t_{s, \text{pipeline}} = (N - 1) \cdot \frac{t_p + T + t_s}{2N} + \frac{t_s}{N} \quad (15)$$

and the length of the uncertainty period is now

$$t_{s, \text{pipeline}} - t_{p, \text{pipeline}} = \frac{t_s - t_p}{N}, \quad (16)$$

a factor of N smaller. Thus, the theoretical maximum throughput is a factor of N higher. Again, the reason for this increase is that the intermediate pipeline registers have reduced the length of the uncertainty period, since the pipeline registers have rendered the uncertainty period zero for all but the last block (by controlling the computational latency with the clock).

This interpretation of the role of pipeline registers as reducing the uncertainty period is unconventional. A more common approach is to *start* with the assumption that there are pipeline registers (and, hence, pipeline stages) and a given fixed clock frequency, and then place as much of the total computation within each stage as possible, with the constraint that the settling time has to be less than the clock period. In this common viewpoint, pipelining and pipeline registers are synonymous. Within their domain of common applicability, the uncertainty period and the common viewpoints are simply different ways of expressing the same design approach. However, the uncertainty period approach we have presented here is more general, in that it includes pipelining without pipeline registers, as often occurs in communications or interconnect. More importantly, this approach serves as a unified framework under which computational blocks, communication or interconnect, and combinations of the two can be characterized—where the computational blocks often utilize pipeline registers to reduce the uncertainty period, and communications links often do not (because the uncertainty period is inherently very small).

In some situations, the *total* settling time of the computation is just as important as the throughput, for example, when the computation sits in a feedback loop. Thus, the effect of pipeline registers on this computational latency is also of interest. If we use the minimum clock period $T = t_s - t_p$ in (15), the total settling time through the pipeline is

$$t_{s, \text{pipeline}} = t_s, \quad (17)$$

and the total settling time is the *same* before and after the insertion of the pipeline registers. Thus, we have not paid

a penalty in total settling time in return for the increase in throughput by a factor of N , since only the variability in settling time has been reduced.

In practice, depending on the system constraints, there are two interpretations of computational latency, as illustrated in the following examples.

Example 4: In a computer or signal processing system, the pipeline registers introduce a *logical delay*, analogous to the z^{-1} operator in Z-transforms. Expressed in terms of these logical delays, the N pipeline registers increase computational latency by N (equivalent to a z^{-N} operator). This introduces difficulties, such as unused pipeline stages immediately following a jump instruction, or additional logical delays in a feedback loop. \square

Example 5: In some circumstances, the computational latency as measured in time is the critical factor. For example, in the media access controller for a local area network, the time to respond to an external stimulus is critical. For this case, as we have seen, the addition of pipeline registers need not increase the computational latency at all. With or without pipeline registers, the computational latency is bounded below by the inherent precedences in the computation as implemented by a particular technology. \square

In practice, it is usually not possible to precisely divide a computational block into "equal-sized" pieces. In that case, the throughput has to be adjusted to match the *largest* uncertainty period for a block in the pipeline, resulting in a lowered throughput. There are a number of other factors, such as register setup times, which reduce the throughput and increase the overall settling time relative to the fundamental bounds that have been discussed here. One of the most important of these is the effect of interconnect delay and clock skew, which we will address next.

3) *Clock Skew in Synchronous Interconnect:* The effects of clock phase and *interconnect delay* (delay of signals passing between computational blocks) will now be considered. Clearly, any uncertainty in clock phase will reduce the throughput relative to (13), since earlier results required precise control of clock phase within a vanishing certainty period. Conversely, any fixed delay in the interconnect will not necessarily affect the achievable throughput, because it will increase the propagation and settling times equally and thus not affect the length of the uncertainty period. In practice, for common digital system design approaches, the effect of any uncertainty in clock phase is magnified by any interconnect delays.

In this subsection, we relax the previous assumptions, and assume that the clock phase can be controlled only within some known range (similar to the uncertainty period for the computational block). We then determine the best throughput that can be obtained following an approach similar to [8] and [9].

We can analyze clock skew with the aid of Fig. 10, in which we modify Fig. 7(b) to introduce some new effects. In particular, we model the following.

- *Propagation and Settling Times:* As before, the prop-

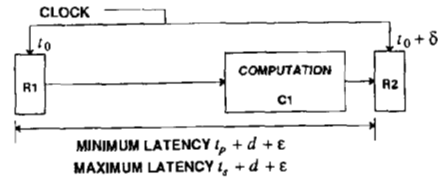


Fig. 10. Illustration of the effects of clock skew δ , where two computations C1 and C2 are synchronized using this clock.

agation and settling times of the computation are t_p and t_s , except that we now include in these times any latencies relative to the clock transition imposed by the implementation of R1.

- *Interconnect Delay:* We assume interconnect delay d due to the diffusion of the signal through the interconnect wires between R1 and C1 and between C1 and R2.

- *Artificially Added Delay:* We assume that another delay ϵ is artificially added to the interconnect delay. The ϵ delay could be introduced, for example, by making R1 a double register with two clock phases, thereby introducing an artificial delay. We will find that ϵ is helpful in controlling the effects of clock skew, by effectively making the minimum interconnect delay larger than zero.

- *Clock Skew:* We assume that the clock transition at R1 occurs at time t_0 at R1, and the clock phase at R2 is $t_0 + \delta$, where δ is the clock skew. This clock skew can be either inadvertent, due, for example, to processing variations or interconnect delays in the clock distribution, or it can be deliberately controlled, for example, to adjust the R2 clock phase to fall within the certainty period. Further, it is possible for δ to be either positive or negative.

There is a certainty region of parameters $\{t_p, t_s, d, \epsilon, \delta\}$ where reliable operation is assured in Fig. 10. This is analyzed in Appendix A for the following three cases.

- *Idealistic Case:* If there is no uncertainty in d or δ and we set the delay ϵ most advantageously, then the bound of (13) is achieved. As expected, the interconnect delay need not necessarily slow the throughput.

- *Pessimistic Case:* Assume that ϵ can be precisely controlled (since it is controlled by relative clock phases) but that δ will inevitably only be controllable within a range, say, $|\delta| < \delta_{\max}$. Thus, we are not attempting to set δ most advantageously. Further, we assume that all that is known about the interconnect delay is that it is bounded, $|d| < d_{\max}$. The throughput is then bounded by

$$\frac{1}{T} < \frac{1}{t_s - t_p + d_{\max} + 2\delta_{\max}} \quad (18)$$

For a system with a complex interconnect pattern, it would be very difficult to control the relationship of δ and d . In this case, we should expect $\delta_{\max} \approx d_{\max}$, and the throughput would be bounded by

$$\frac{1}{T} < \frac{1}{(t_s - t_p) + 3d_{\max}} \quad (19)$$

• *Optimistic Case:* For simple topologies like a one-dimensional pipeline, much higher throughput can be obtained by routing the signals and clocks in such a way that d and δ can be coordinated with one another [8]. Assume that the interconnect delay is known to be d_0 with variation Δd , and the skew is chosen to be δ_0 with variation $\Delta\delta$. Further assume that ϵ and δ_0 are chosen most advantageously to maximize the throughput. Then the throughput is bounded by

$$\frac{1}{T} < \frac{1}{(t_s - t_p) + 2(\Delta\delta + \Delta d)}, \quad (20)$$

which is a considerable improvement over (19) if the delay and skew variations are small. This analysis shows that the reliable operation of the idealized synchronous interconnection of Section III-A-1) can be extended to accommodate interconnect delays and clock skew, even with variations of these parameters, albeit with some necessary reduction in throughput.

To get a feeling for the numbers, consider a couple of numerical examples.

Example 6: Consider the pessimistic case, which would be typical of a digital system with an irregular interconnection topology that prevents easy coordination of interconnect delay and clock skew. For a given clock speed or throughput, we can determine from (19) the largest interconnect delay d_{\max} , that can be tolerated, namely, $(T - t_s)/3$, assuming that the interconnect delay and clock skew are not coordinated and assuming the worst-case propagation delay, $t_p = 0$. For a 100 MHz clock frequency, a clock period of 10 ns and, assuming the settling time is 80% of the clock period, the maximum interconnect delay is 667 ps. The delay of a data or clock signal on a printed circuit board is on the order of 5–10 ps/mm (as compared to a free-space speed of light of 3.3 ps/mm). The maximum interconnect distance is then 6.7–13.3 cm. Clearly, synchronous interconnect is not viable on PC boards at this clock frequency under these pessimistic assumptions. This also does not take into account the delay in passing through the pins of a package, roughly 1–3 ns (for ECL or CMOS, respectively) due to capacitive and inductive loading effects. Thus, we can see that interconnect delays become a very serious limitation in the board-level interconnection with 100 MHz clocks. □

Example 7: On a chip, the interconnect delays are much greater (about 90 ps/mm for Al-SiO₂-Si interconnect), and are also somewhat variable due to dielectric and capacitive processing variations. Given the same 667 ps interconnect delay, the maximum interconnect distance is now about 8 mm. (This is optimistic since it neglects the delay due to source resistance and line capacitance—which will be dominant effects for relatively short interconnects.) Thus, we see difficulties in using synchronous interconnect on a single chip for a complex and global interconnect topology. □

Again, it should be emphasized that greater interconnect distance is possible if the clock skew and interconnect delay can be coordinated, which may be possible if the interconnect topology is simple as in one-dimensional pipeline. This statement applies at both the chip and board levels.

4) *Parallel Signal Paths:* An important practical importance of pipeline registers is in synchronizing the signals on parallel paths. The transition phase offset between these parallel paths tends to increase through computational blocks and interconnect, and can be reduced by a pipeline register to the order of the clock skew across the bits of this multibit register. Again, the register can be viewed as reducing the size of the uncertainty region, in this case spatially as well as temporally.

In Section III-A-1), we defined the total uncertainty region for a collection of parallel signals as the union of the uncertainty regions for the individual signals. From the preceding, the total throughput is then bounded by the reciprocal of the length of this aggregate uncertainty period. In contrast, if each signal path from among the parallel paths were treated independently (say using the mesochronous techniques to be described later), the resulting throughput could in principle be increased to the reciprocal of the maximum of the individual uncertainty periods. For many practical cases, we would expect the longest uncertainty period to include the other uncertainty periods as subsets, in which case these two bounds on throughput would be equal; that is, there is no advantage in treating the signals independently. The exception to this rule would be where the uncertainty periods were largely nonoverlapping due to a relative skew between the paths that is larger than the uncertainty period for each path, in which case there would be considerable advantage to dealing with the signals independently.

B. Anisochronous Interconnect

The synchronous interconnect approach uses isochronous signals throughout the system, since all signals are slaved to an isochronous clock. A popular alternative to synchronous interconnect has been to abandon the isochronous assumption, and further abandon the use of a global clock signal altogether. Rather, the elements of the system are chosen to fall within an equipotential region, and the interconnection between elements is designed to operate in a delay-insensitive manner; that is, operate reliably regardless of what the delays are. This is accomplished by having each element of the system generate a *completion signal*, which has a transition coincident with the settling time of that element. The completion signal is a sort of locally generated clock, and is used to synchronize the different elements. Since the completion signal depends on the settling time, which can be data-dependent, the resulting signals are anisochronous. For example, if an ALU has two instructions with different settling times, then the signal frequencies will depend on the mix of instructions, and will thus be anisochronous.

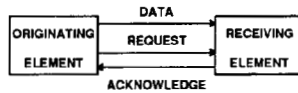


Fig. 11. Anisochronous interconnection, where coordination is performed by handshake signals.

In VLSI design, the term *self-timed interconnect* is used to describe this design approach [6]. Since this terminology conflicts with that of digital communication, we will use the term *anisochronous* instead. It is also common to refer to this method of synchronization as *asynchronous interconnect*, which is not precise, because this is only one of a number of possible asynchronous approaches (we will see some others later).

The idea behind the anisochronous interconnection is shown in Fig. 11. The originating element generates a handshake signal called a *request*, indicating that the settling time has been reached and the data are available. In addition, another handshaking signal called the *acknowledge* is generated at the destination, and indicates to the originator that the signal has been received and the originator is free to proceed with the generation of a new signal. The correct operation of the anisochronous interconnection does not depend on assumptions about the interconnect delays, other than that they be essentially identical for the data and handshake wires (which is reasonable if they are routed together). This is because the roundtrip feedback ensures that the two operations are synchronized, independent of the interconnect delay. Further details on anisochronous interconnect can be found in [6], with recent results summarized in [10]–[12].

A more detailed diagram of the anisochronous interconnect is shown in Fig. 12. The clock in Fig. 10 has been replaced by a pair of handshake blocks H1 and H2, which generate the request signal R_{out} for the next block and accept the acknowledge signal A_{in} from that same block. In addition, the calculation block now generates a completion signal C , indicating that the setting time has been completed, and accepts an input signal R_{out} which initiates its computation.

A *state transition diagram* for H1 is shown in Fig. 13 for a four-phase handshaking circuit appropriate in Fig. 12. This diagram models the order in which transitions in the H1 occur, and also the precedences that must be maintained by the circuitry in H1. For example, an arc from A_{out}^+ to R_{out}^+ indicates that the positive transition in A_{out} must precede the positive transition R_{out} .

The maximum throughput with which the system can operate is determined by the largest latency in any loop in Fig. 13. That maximum latency is the loop on the right, $R_{out}^+ \rightarrow A_{in}^+ \rightarrow R_{out}^- \rightarrow A_{in}^-$. If the settling time between the registers is the computational settling time t_s plus the interconnect delay d as shown, then the latency of the $R_{out}^+ \rightarrow A_{in}^+$ transition must be $t_s + 2d$ because this loop through H2 includes two interconnect delays plus the settling time, and the latency in the transition $R_{out}^- \rightarrow A_{in}^-$ must similarly be $2d$ because there is no computation in

this loop. Additional latencies due to the handshake circuitry have been neglected. The total throughput is thus bounded by

$$\frac{1}{T} \leq \frac{1}{t_s + 4d} \quad (22)$$

There are two-phase handshake approaches that are less reliable but reduce the $4d$ in the denominator to $2d$. Comparing (22) to (19), we see that the anisochronous interconnect throughput depends on the *actual* delay, whereas the pessimistic bound on throughput for synchronous interconnect depends on the *maximum* delay. (In the case of a pipeline, the total throughput will be dominated by the block in the pipeline with the maximum delay, so the two will be essentially the same.) On the other hand, the synchronous interconnect can take advantage of a nonzero propagation time t_p to pipeline without pipeline registers, whereas the anisochronous interconnect does not. At its best, synchronous interconnect can operate with a throughput that is independent of interconnect delay, and only limited by delay *variation*, as demonstrated in (20). Thus, depending on the circumstances, either the synchronous or anisochronous interconnect can achieve the higher throughput. However, in the presence of large interconnect delays, the synchronous interconnect clearly has a potentially higher throughput.

An important point is that the throughput of both the synchronous and anisochronous interconnect are generally adversely affected by the interconnect delay, and especially so for complicated interconnect topologies. As technologies scale, this restriction will become a more and more severe limitation on the performance of digital systems. However, as demonstrated by digital communication, which has experienced large interconnect delays from its inception, this delay-imposed throughput limitation is not fundamental, but is imposed in the synchronous case by the open-loop nature of the setting of clock phase.

An advantage of both synchronous and anisochronous interconnect is that they are free of metastability. This is avoided by ensuring through the design methodology that the clock and data are never lined up precisely, making it possible for clocked memory elements to reliably sample the signal.

C. Synchronization in Digital Communication

In digital communication, the interconnect delays are very large, so that alternative synchronization techniques are required [3]. These approaches are all isochronous, implying that the signals are all slaved to clocks, but differ as to whether a common clock distributed to each node of the network is used (mesochronous) or independent clocks are used at the nodes (plesiochronous and heterochronous). They also share a common disadvantage relative to synchronous and anisochronous interconnect—the inevitability of metastable behavior. Thus, they all have to be designed carefully with metastability in mind, keeping the probability of that condition at an acceptable level.

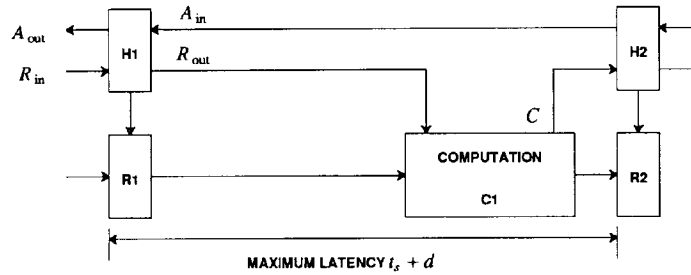


Fig. 12. An anisochronous interconnect circuit, in which relative to Fig. 10 the clock has been removed and replaced by two handshake circuits H1 and H2. H1 and H2 generate the clocks for R1 and R2, respectively. The signals R and A are, respectively, the request and acknowledge handshaking signals, essentially a locally generated clock, and C is the completion signal for the logic block.

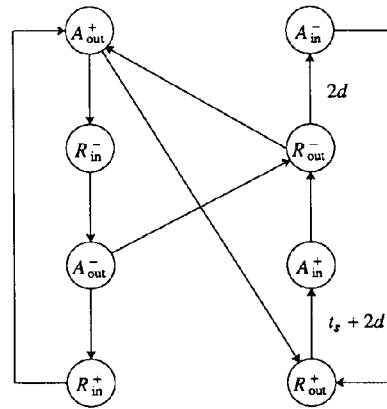


Fig. 13. A state transition diagram for a four-phase pipeline handshaking circuit H1 from [10]. The superscript “+” or “-” indicates a positive or negative transition latency in the corresponding signal. Arcs are labeled with the corresponding latency.

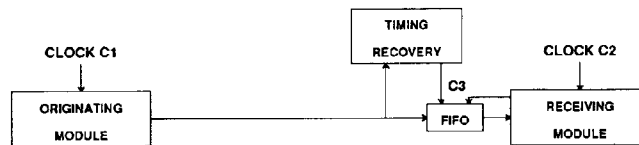


Fig. 14. An illustration of how throughput is made independent of delay in digital communication.

Limitations to throughput due to propagation delay are avoided in digital communications as shown in Fig. 14. First, two communicating modules are each provided a clock—C1 and C2. A clock is also derived from the incoming signal in the *timing recovery circuit*, and is denoted C3. These clocks have the following relationships.

- C3 is synchronous with the signal at the input to the FIFO, since it is derived from that signal.
- C3 is mesochronous to C1, since it has the same average frequency as dictated by the common signal but has an indeterminate phase due to the interconnect delay. It can also have significant phase jitter due to a number of effects in the long-distance transmission [3].

- C1 and C2 are either mesochronous, if they originated from a common source, or they are independent. In the latter case, they are either plesiochronous or heterochronous.

The purpose of the FIFO (also called an *elastic store*) is to adjust for the differences in phase, likely to be time varying, between C3 and C2. For mesochronous C1 and C2, this phase difference is guaranteed to be bounded, so that the FIFO can be chosen with sufficient storage capacity to never overflow. For heterochronous C1 and C2, where the average frequency of C1 is guaranteed to be lower than the average frequency of C2, the FIFO occupancy will decrease with time, and hence no data will ever

be lost. For plesiochronous C1 and C2, the FIFO could overflow, with the loss of data, if the average frequency of C1 happens to be higher than C2. This loss of data is acceptable on an occasional basis, but may not be permissible in a digital system design.

IV. ISOCHRONOUS INTERCONNECT IN DIGITAL SYSTEMS

We found previously that the performance of both synchronous and anisochronous interconnects in digital systems are limited as a practical matter by the interconnect delays in the system. With the anisochronous approach, this limitation was fundamental to the use of roundtrip handshaking to control synchronization. In the synchronous (but not anisochronous) case, we showed that this limitation is not fundamental, but rather comes from the inability to tightly control clock phases at synchronization points in the system. The reason is the "open-loop" nature of the clock distribution, making us susceptible to processing variations in delay. If we can more precisely control clock phase using a "closed-loop" approach, the throughput of the synchronous approach can more nearly approach the fundamental limit of (13), and considerably exceed that of anisochronous interconnect in the presence of significant interconnect delays. In this section, we explore some possibilities in that direction, borrowing techniques long used in digital communication.

A. Mesochronous Interconnect

Consider the case where a signal has passed over an interconnect and experienced interconnect delay. The interconnect delay does not increase the uncertainty period, and thus does not place a fundamental limitation on throughput. If this signal has a small uncertainty period, as for example it has been resynchronized by a register, then the certainty period is likely to be a significant portion of the clock cycle, and the phase with which this signal is resampled by another register is not even very critical. The key is to avoid a sampling phase within the small uncertainty period, which in synchronous interconnect can be ensured only by reducing the throughput. But if the sampling phase can be controlled in closed-loop fashion, the interconnect delay should not be a factor, as demonstrated in digital communication systems.

Another perspective on clock skew is that it results in an indeterminate phase relationship between local clock and signal; in other words, the clock and signal are actually mesochronous. In *mesochronous interconnect*, we live with this indeterminate phase, rather than attempting to circumvent it by careful control of interconnect delays for clock and signal. This style of interconnect is illustrated in Fig. 15. Variations on this method were proposed some years ago [13] and pursued into actual chip realizations by a group at M.I.T. and BBN [14], [15] (although they did not use the term "mesochronous" to describe their technique). We have adapted our version of this approach from the mesochronous approach used worldwide in digital communication, except that in this case we can make the simplifying assumption that the

phase variation of any signal or clock arriving at a node can be ignored. The primary cause of the residual phase modulation will be variations in temperature of the wires, and this should occur at very slow rates and at most requires infrequent phase adjustments. This simplification implies that clocks need not be derived from incoming signals, as in the timing recovery of Fig. 14, but rather a distributed clock can be used as a reference with which to sample the received signal. However, we must be prepared to adjust the *phase* of the clock used for sampling the incoming signal to account for indeterminate interconnect delays of both clock and signal. We thus arrive at Fig. 15.

First, we divide the digital system into functional entities called "synchronous islands." The granularity of partitioning into synchronous islands is guided by the principle that within an island the interconnect delays are small relative to logic speed, and traditional synchronous design can be used with *minimal impact* from interconnect delays. The maximum size of the synchronous island depends to a large extent on the interconnect topology, as we have seen previously. In near-term technology, a synchronous island might encompass a single chip within a digital system and, for the longer term, a single chip may be partitioned into two or more synchronous islands. The interconnection of a pair of synchronous islands is shown in Fig. 15(a); externally, the connection is identical to the synchronous interconnection in Fig. 6. The difference is in the relaxed assumptions on interconnect delays.

The mesochronous case requires a more complicated internal interface circuit, as illustrated in Fig. 15(b). This circuit performs the function of *mesochronous-to-synchronous conversion*, similar in function but simpler than the FIFO in Fig. 14. This conversion requires the following.

- A *clock phase generator* to create a set of discrete phases of the clock. This does not require any circuitry running at speeds greater than the clock speed, but rather can be accomplished using a circuit such as a ring oscillator phase-locked to the external clock. A single phase generator can be shared among synchronous islands (such as one per chip), although to reduce the routing overhead it can be duplicated on a per-island or even on a multiple generator per-island basis.
- A *phase detector* determines the certainty period of the signal, for example, by estimating the phase difference between a particular phase of the clock and the transitions of the signal. Generally, one phase detector is required for each signal line or group of signal lines with a common source and destination and similarly routed.
- A *sampling register* R1 with sampling time chosen well within the certainty period of the signal, as controlled by the phase detector. This register reduces the uncertainty period of the signal, for example, eliminating any finite rise-time effects due to the dispersion of the interconnect, and also controls the phase of the uncertainty period relative to the local clock. Depending on the effect of temperature variations in the system, this appropriate

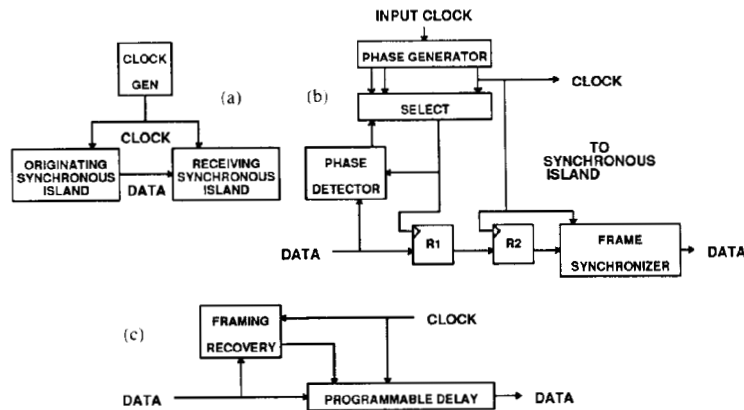


Fig. 15. A mesochronous interconnection for a digital system. (a) Clock and signal distribution are identical to the synchronous case of Fig. 6(a). Each functional unit corresponds to a synchronous island. (b) Interface circuit required for mesochronous-to-synchronous conversion. (c) Optional frame synchronizer circuit (within the synchronous island).

phase may be chosen once as power-up, or may be adjusted infrequently if the phase variations are a significant portion of a clock cycle.

- A second register R2 resamples the signal using the clock phase used internally to the synchronous island. At this point, the signal and this internal clock are synchronous. By inference, all the signals at the input to the synchronous island are also synchronous.

- If the system interconnect delays are larger than one clock cycle, it may be convenient at the architectural level to add an optional *frame synchronizer*, which is detailed in Fig. 15(c). The purpose of this block is to synchronize all incoming signal lines at the word-level (for example, at byte boundaries) by adding a programmable delay. This framing also generally requires additional framing information added to the signal.

The practicality of the mesochronous interconnection depends to a large extent on the complexity of the interconnection circuit and the speed at which it can be implemented. All the elements of this circuit are simple and pretty standard in digital systems with the possible exception of the phase detector. Thus, the practicality of mesochronous interconnection hinges largely on innovative approaches to implementing a phase detector. A phase detector can be shared over a group of signals with the same origin and destination and same path length, and it can even be time-shared over multiple such functions since we do not expect the phase to change rapidly. Nevertheless, the phase detector should be realized in a small area, at least competitive with anisochronous handshaking circuits. Further, it should be fast, not restricting the clock rate of the interconnect, since the whole point is to obtain the maximum speed. This should not be a problem on slower out-of-package interconnections, but may be difficult to achieve on-chip. Any phase detector design is also likely to display metastable properties, which must be minimized. The design of the phase detector may also

place constraints on the signal, such as requiring a minimum number of transitions. In view of all these factors, the phase detector is a challenging analog circuit design problem.

B. Heterochronous and Plesiochronous Interconnect

Like synchronous interconnect, and unlike anisochronous interconnect, mesochronous interconnect requires distribution of a clock to all synchronous islands, although any interconnect delays in this distribution are not critical. If the power consumption of this clock is of concern, in principle it would be possible to distribute a sub-multiple of the clock frequency, and phase-lock to it at each synchronous island. But if the interconnect wires for clock distribution are of concern, they can be eliminated as in the digital communication system of Fig. 14, where the timing recovery is now required. There are two useful cases.

- If clocks C1 and C2 are *heterochronous*, such that C1 is guaranteed to have a lower frequency than C2, then the FIFO can be very small and overflow can never occur. However, the signal from the FIFO to the receiving module would have to contain "dummy bits," inserted whenever necessary to prevent FIFO underflow, and a protocol to signal where those bits occur.

- If clocks C1 and C2 are *plesiochronous*, then overflow of the FIFO can occur in the absence of flow control. Flow control would insert the "dummy bits" at the originating module whenever necessary to prevent FIFO overflow. Flow control could be implemented with a reverse handshaking signal that signaled the originating module, or if another interconnect link existed in the opposite direction, that could be used for this purpose.

V. ARCHITECTURAL ISSUES

While synchronous interconnect is still the most common, both anisochronous and mesochronous intercon-

nects are more successful in abstracting the effects of interconnect delay and clock skew. Specifically, both design styles ensure reliable operation independent of interconnect delay, without the global constraints of sensitivity to clock distribution phase. Each style of interconnect has disadvantages. Mesochronous interconnect will have metastability problems and requires phase detectors, while anisochronous interconnect requires extra handshake wires, handshake circuits, and a significant silicon area for completion-signal generation.

The style of interconnection will substantially influence the associated processor architecture. The converse is also true—the architecture can be tailored to the interconnect style. As an example, if at the architectural level we can make a significant delay one stage of a pipeline, then the effects of this delay are substantially mitigated. (Consider, for example, (22) with $t_s = 0$.)

Many of these issues have been discussed with respect to anisochronous interconnection in [16]. On the surface one might presume that mesochronous interconnection architectural issues are similar to synchronous interconnection which, if true, would be a considerable advantage because of the long history and experience with synchronous design. However, the indeterminate delay in the interconnection (measured in bit intervals at the synchronous output of the mesochronous-to-synchronous conversion circuit) must be dealt with at the architectural level. For synchronous interconnection, normally the delay between modules is guaranteed to be less than one clock period (that does not have to be the case as illustrated in Fig. 8), but with mesochronous interconnection the delay can be multiples of a bit period. This has fundamental implications to the architecture. In particular, it implies a higher level of synchronization (usually called “framing” in digital communications) which line up signal word boundaries at computational and arithmetic units.

In the course of addressing this issue, the following key question must probably be answered:

Is there a maximum propagation delay that can be assumed between synchronous islands? If so, is this delay modest?

The answer to this question is most certainly “yes” in a given chip design, but difficult to answer for a chip designed to be incorporated into a mesochronous board- or larger-level system. As an example of an architectural approach suitable for worst-case propagation delay assumptions, we can include frame synchronizers like Fig. 15(c) which “build-out” each interconnection to some worst-case delay. Every interconnection thus becomes worst-case and, more importantly, predictable in the design of the remainder of the architecture. However, this approach is probably undesirable for reasons that will soon be elaborated. Another approach is to build into the architecture adjustment for the actual propagation delays, which can easily be detected at power-up. Yet another approach is to use techniques similar to packet switching in which each interconnection carries associated synchronization

information (beginning and end of message, etc.), and design the architecture to use this information. We have studied this problem in some detail in the context of interprocessor communication [17].

As previously mentioned, each data transfer in an anisochronous interconnect requires two to four propagation delays (four in the case of the most reliable four-phase handshake). In contrast, the mesochronous interconnection does not have any feedback signals and is thus able to achieve whatever throughput can be achieved by the circuitry, independent of propagation delay. This logic applies to feedforward-only communications. The more interesting case is the command-response situation or, more generally, systems with feedback, since delay will have a considerable impact on the performance of such systems. To some extent, the effect of delay is fundamental and independent of interconnect style: the command-response cycle time cannot be smaller than the roundtrip propagation delay. However, using the “delay build-out” frame synchronizer approach described earlier would have the undesirable effect of unnecessarily increasing the command-response time of many interconnections in the system. Since this issue is an interesting point of contrast between the anisochronous and mesochronous approaches, we will now discuss it in more detail.

A. Command-Response Processing

Suppose two synchronous islands are interconnected in a bilateral fashion, where one requests data and the other responds. An example of this situation would be a processor requesting data from a memory—the request is the address and the response is the data residing at that address. Assuming for the moment that there is no delay generating the response within the responding synchronous island, the command-response cycle can be modeled simply as a delay by N clock cycles, corresponding to roughly a delay of $2t_p$, as illustrated in Fig. 16.

It is interesting that this delay is precisely analogous to the delay (measured in clock cycles) introduced by pipelining—we can consider this command-response delay as being generated by N pipeline registers. As in pipelining, this delay can be deleterious in reducing the throughput of the processing, since the result of an action is not available for N clock cycles. One way of handling this delay is for the requesting synchronous island to go into a wait state for N clock cycles after each request. The analogous approach in pipelining is to launch a new data sample each N cycles, which is known as *N-slow* [18], and the throughput will be inversely proportional to N . This approach is analogous to the anisochronous interconnection (which may require considerably more delay, such as four propagation delays for the transfer in each direction).

For the mesochronous case, there are some architectural alternatives that can result in considerably improved performance under some circumstances, but only if this issue is addressed at the architectural level. Some examples of these include the following.

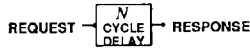


Fig. 16. A model for command-response in a mesochronous interconnection. The delay is with respect to cycles of the common clock.

- If we have some forward-only communications coincident with the command-response communications, we can *interleave* these feedforward communications on the same lines.

- If we have a set of N independent command-response communications, we can interleave them on the interconnection. This is analogous to *pipeline interleaving* [19] (which can make full use of the throughput capabilities of a pipelined processing element provided that we can decompose the processing into N independent streams). If the responding island cannot accommodate this high a throughput, then it can be duplicated as many times as necessary (in pipelining, an analogous example would be memory interleaving).

- If we cannot fully utilize the interconnection because of the propagation delay, then at least we should be able to allow each processing element to do useful processing (at its full throughput) while awaiting the response. This is analogous to what is sometimes done when a synchronous processor interacts asynchronously with a slower memory.

The last of these options would be available to an anisochronous system, since the communication portion could be made a separate pipeline stage. However, the first two options—two forms of interleaving of communications—are not available in anisochronous systems because the *total* throughput is bounded by the propagation delay. If the communication bottlenecks are taken into account at the architectural level, it appears that mesochronous interconnection offers considerable opportunity for improved performance.

VI. CONCLUSIONS

In this paper we have attempted to place the comparison of digital system synchronization on a firm theoretical foundation, and compare the fundamental limitations of the synchronous, anisochronous, and mesochronous approaches. A firm conclusion is that interconnect delays place a fundamental limitation on the communication throughput for anisochronous interconnect (equal to the reciprocal of two or four delays), this limitation does not exist for mesochronous interconnect. Further, mesochronous interconnect can actually achieve pipelining in the interconnect (as illustrated in Example 1) without additional pipeline registers, whereas anisochronous cannot. Further, anisochronous requires extra interconnect wires and completion signal generation. Thus, as clock speeds increase and interconnect delays become more important, mesochronous interconnect shows a great deal of promise. However, the advantages of any synchronization technique cannot be fully exploited without modifications at the architectural level.

Synchronization is the most difficult issue faced in digital system interconnect viewed as a digital communication problem, but there are some other techniques that can be considered. One of the more interesting is line code design [3]. Electrical interconnect displays dispersive properties which cause intersymbol interference at high speeds, through both bandwidth constraints and microreflections. Multilevel line codes would be an interesting possibility from the perspective of increasing the data rate within a bandwidth constraint. Also, the line code can constrain the number of transitions per unit time, easing, for example, the design of phase detectors [20].

Equalization of the intersymbol interference is a well-proven technology, but it is likely not practical at the high speeds of digital system interconnect.

APPENDIX A

In this Appendix, we determine the certainty region for the parameters in Fig. 10. Consider a computation initiated by the clock at R1 at time t_0 . Extending our earlier results, the conditions for reliable operation are now as follows.

- The earliest the signal can change at R2 is after the clock transition $t_0 + \delta$, or

$$t_0 + \delta < t_0 + t_p + d + \epsilon. \quad (23)$$

- The latest time the signal has settled at R2 must be before the next clock transition $t_0 + \delta + T$, where T is the clock period,

$$t_0 + t_s + d + \epsilon < t_0 + \delta + T. \quad (24)$$

Simplified, these equations become

$$\delta < t_p + d + \epsilon \quad (25)$$

$$T + \delta > t_s + d + \epsilon. \quad (26)$$

Together, (25) and (26) specify a *certainty region* for $\{\delta, T\}$ where reliable operation is guaranteed.

With the aid of Fig. 17, we gain some interesting insights. First, if the skew is sufficiently positive, reliable operation cannot be guaranteed because the signal at R2 might start to change before the clock transition. If the skew is negative, reliable operation is always guaranteed if the clock period is large enough.

Idealistic Case: The most advantageous choice for the skew is $\delta = t_p + d + \epsilon$ at which point we get reliable operation for $T > t_s - t_p$. Choosing this skew, the fundamental limit of (13) would be achieved. This requires precise knowledge of the interconnect delay d as well.

Pessimistic Case: In Fig. 17 we see the beneficial effect of ϵ , because if it is large enough, reliable operation is guaranteed for any δ_{\max} . In particular, the condition is $t_p + d + \epsilon > \delta_{\max}$, or

$$t_p + d > \delta_{\max} - \epsilon. \quad (27)$$

Since the interconnect delay d is always positive, (27) is guaranteed for any t_p and d so long as $\epsilon > \delta_{\max} - t_p$. Since ϵ also has the effect of increasing T , it is advantageous to ϵ as small as possible; namely, $\epsilon = (\delta_{\max} - t_p)$. Referring

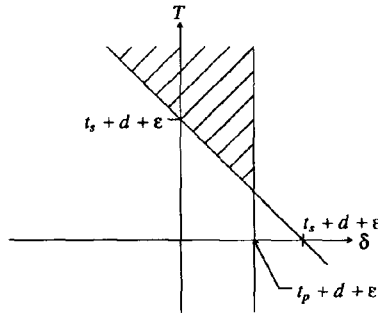


Fig. 17. The crosshatched area is the certainty region, now in terms of $\{\delta, T\}$. Note that it is not always possible to choose the clock period T sufficiently large to fall in the certainty region, but it is always possible to choose T and ϵ together large enough to accommodate any bounded clock skew.

back to Fig. 17, reliable operation is guaranteed if

$$T > (t_s - t_p) + d + 2\delta_{\max}. \quad (28)$$

Relative to the fundamental bound on clock period ($t_s - t_p$), the clock period must be increased by $(d + 2\delta_{\max})$. Taking account of the worst case $d = d_{\max}$, we get (18).

Optimistic Case: The throughput of (20) follows easily by the same method.

ACKNOWLEDGMENT

The author is pleased to acknowledge helpful conversations on these issues with friends and colleagues, including T. Meng of Stanford; C.-S. Li, H.-D. Lin, S.-F. Chang, and P. Gray of U.C. Berkeley; K. Parhi of the University of Minnesota; M. Hatamian of AT&T Bell Laboratories; and H. Stone of IBM Research.

REFERENCES

- [1] M. Hatamian, L. A. Hornak, T. E. Little, S. K. Tewksbury, and P. Franzon, "Fundamental interconnection issues," *AT&T Tech. J.*, vol. 66, p. 134, July 1987.
- [2] D. G. Messerschmitt, "Digital communication in VLSI design," in *Proc. 23rd Asilomar Conf. Signals, Syst. Comput.*, Oct. 1989.
- [3] E. A. Lee and D. G. Messerschmitt, *Digital Communication*. Norwell, MA: Kluwer Academic, 1988.
- [4] T. J. Chaney and C. E. Molnar, "Anomalous behavior of synchronizer and arbiter circuits," *IEEE Trans. Comput.*, vol. C-22, p. 421, Apr. 1973.
- [5] G. H. Bennett, "Pulse code modulation and digital transmission," *Murconi Instruments*, Apr. 1978.
- [6] C. Mead and L. Conway, *Introduction to VLSI Systems*. Reading, MA: Addison-Wesley, 1980.
- [7] D. Wong, G. De Micheli, and M. Flynn, "Designing high-performance digital circuits using wave pipelining," *IEEE ICCAD-89 Dig. Tech. Papers*, Nov. 1989.

- [8] M. Hatamian and G. L. Cash, "Parallel bit-level pipelined VLSI designs for high-speed signal processing," *Proc. IEEE*, vol. 75, p. 1192, Sept. 1987.
- [9] M. Hatamian, "Understanding clock skew in synchronous systems," in *Concurrent Computations*, S. C. Schwartz, Ed. New York: Plenum, 1988.
- [10] T. Meng, R. W. Brodersen, and D. G. Messerschmitt, "Asynchronous logic synthesis for signal processing from high-level specifications," *IEEE ICCAD 87 Dig. Tech. Papers*, Nov. 1987.
- [11] T. Meng, G. Jacobs, R. Brodersen, and D. Messerschmitt, "Implementation of high sampling rate adaptive filters using asynchronous design techniques," in *Proc. IEEE Workshop VLSI Signal Processing*, Nov. 1988.
- [12] T. Meng and D. G. Messerschmitt, "Automatic synthesis of asynchronous circuits from high-level specifications," *IEEE Trans. Comput. Aided Design*, accepted for publication.
- [13] W. Nix, "A system for synchronous data transmission over a line of arbitrary delay," M.S. Project Report, U.C. Berkeley, 1981.
- [14] P. Bassett, L. Glasser, and R. Rettberg, *Dynamic Delay Adjustment: A Technique for High-Speed Asynchronous Communication*. Cambridge, MA: M.I.T. Press, 1986.
- [15] P. Bassett, "A high-speed asynchronous communication technique for MOS VLSI systems," *Mass. Inst. Technol.*, Dec. 1985.
- [16] T. Meng, G. Jacobs, R. Brodersen, and D. Messerschmitt, "Asynchronous processor design for digital signal processing," in *Proc. IEEE Int. Conf. ASSP*, Apr. 1988.
- [17] M. Ilovich, "High performance programmable DSP architectures," Ph.D. dissertation, Univ. Calif., Apr. 1988.
- [18] K. Parhi and D. G. Messerschmitt, "Pipeline interleaving and parallelism in recursive digital filters, Part I: Pipelining using scattered look-ahead and decomposition," *IEEE Trans. Acoust., Speech, Signal Processing*, July 1989.
- [19] E. A. Lee and D. G. Messerschmitt, "A coupled hardware and software architecture for programmable digital signal processors, Part I: Hardware," *IEEE Trans. Acoust., Speech, Signal Processing*, Sept. 1987.
- [20] D. G. Messerschmitt and M.-K. Liu, "A fixed transition coding for high speed timing recovery in fiber optics networks," in *Proc. IEEE Int. Conf. Commun.*, June 1987.



David G. Messerschmitt (S'65-M'68-SM'78-F'83) received the B.S. degree from the University of Colorado, Boulder, in 1967, and the M.S. and Ph.D. degrees from the University of Michigan, Ann Arbor, in 1968 and 1971, respectively.

He is a Professor of Electrical Engineering and Computer Sciences at the University of California, Berkeley. Prior to 1977, he was at Bell Laboratories in Holmdel, NJ. Current research interests include applications of digital signal processing, digital communications (subscriber loop, fiber optics, and in VLSI and digital systems), architectural approaches to dedicated-hardware digital signal processing (especially video compression applications), video applications of broad-band packet networks, and computer aided design of communications and signal processing systems. He has published over 110 papers, is coauthor of two books, and has 10 patents. He also serves as a consultant to a number of companies.

Dr. Messerschmitt is a member of Eta Kappa Nu, Tau Beta Pi, Sigma Xi, and has several best paper awards. He has served as a Senior Editor of the IEEE COMMUNICATIONS MAGAZINE, as Editor for Transmission of the IEEE TRANSACTIONS ON COMMUNICATIONS, and as a member of the Board of Governors of the Communications Society. He has also organized and participated in a number of short courses and seminars devoted to continuing engineering education.