Adaptive Synchronization for Multi-Synchronous Systems

Rakefet Kol and Ran Ginosar[*]
VLSI Systems Research Center, Electrical Engineering Department
Technion - Israel Institute of Technology, Haifa 32000, Israel
rakefet@techunix.technion.ac.il

[*] On Sabbatical leave at Intel Corp., Hillsboro, OR

Abstract

Synchronizers and other methods may become ineffective for high performance systems implemented at future technologies, operating at clock frequencies above 1GHz . As a transition from fully synchronous to fully asynchronous implementations, such a system can be implemented as a *multi-synchronous* system, wherein a common clock is distributed over thin wires, avoiding the massive power investment needed for phase matching and skew minimization in clock distribution networks. *Adaptive synchronization* reduces the probability of synchronization failures. In contrast with methods like clock stretching, adaptive synchronization adjusts data delays. The stationarity of wire and logic delays is exploited to contain asynchrony. We show that adaptive synchronization is more widely applicable to high performance systems than other synchronization methods. Training sessions are devised to minimize adaptation overhead.

1. Introduction

This paper applies some asynchronous concepts to future clocked high performance chips, in order to overcome certain limitations of synchronous design.

With the advent of VLSI technology, chips grow larger and run faster. Over one half billion transistors on a die and clock rates well above 1GHz are predicted for the year 2010 [KG97, SIA94 / SIA97]. But this dramatic progress also poses a new challenge: Signal propagation delays over clock and data lines increase substantially, both in absolute terms and relative to the clock cycle, and relative delay variations increase as well. Thus it will become increasingly difficult to employ present synchronous design methodology in future high performance chips, and a little asynchrony may help meet this challenge.

The issue of increased delays is manifested in both clock and data lines. While the electromagnetic field travels in vacuum at the speed of light (c = 30 mm / 100 pSec, in VLSI terms), the electric signals inside chips progress about 10-100× slower, depending on drive strength and on the capacitive load of the bus. Assuming c/20 signal (clock and data) propagation speeds and chip size of 25-35mm in 2010 technology [SIA94], typical signals will require 2.5-3.5 nSec to cross the chip end-to-end. In a chip clocked at 2GHz, about 5-7 clock cycles may be required for signal propagation alone. The electrically lumped circuit model on which synchronous design is based does not hold any more, and the chip should be treated as a distributed system.

At present, clocks are typically distributed over high power, balanced, skew-free distribution networks, which mask the clock propagation delays. This is very costly, as a growing portion of total power is dissipated by the clock distribution network, including the phase lock loops, buffers, and tuning circuits [Fri95]. For instance, in 1995 it was reported that over 40% of the power budget of the Alpha chip were consumed by the clock distribution network in order to reduce clock skew problems [Bow95]. However, if data transmitted from one module cannot be instantly received at another module, then investing substantial area and power resources in minimizing skew is questionable. In addition, increasing delay variations make it less feasible to produce skew-free clocks.

Delay variations on data and clock lines can be sorted into three categories: Skew, jitter and drift. Skew refers to spatial variation, while jitter and drift are temporal. Skew represents in-die delay variations which are due to variations in physical attributes such as threshold voltage, oxide thickness, and geometric dimensions and fabrication parameters. Skew may easily reach one or more clock cycle times in large, fast chips. Jitter represents cycle-to-cycle variations of the same delay path, mainly due to fast changes in supply voltage and in temperature and to cross talk. Fortunately, jitter in CMOS circuits is typically limited to less than 10% of the clock cycle time, and this stability is expected to sustain for some time in the future. Drift is similar to jitter, but it relates to changes in supply voltage and in temperature that accumulate very slowly, and can be noticed only after millions of cycles or more. Similar to skew, drift can grow into delay variations many times larger than the clock cycle time. The 'asynchronous' clocking methodology discussed in this paper is designed to accommodate jitter and overcome skew and drift. Chips are designed such that skew does not matter, jitter can be safely ignored, and drifts are compensated for by adaptive circuits. We also say that the delays are *stationary*, in the sense that they can be considered fixed over long periods of time, and when they do vary (drift) the circuit adapts to the changes.

Asynchronous design is often proposed as a viable solution for high performance processors, removing the clock altogether [DGY93, Hau95, Kol97, Pav94, SSM94, Mar97]. The other approach is to divide a large clocked chip into multiple smaller clock domains, each with its own skew-free clock. The modules communicate over asynchronous channels, and each channel must be synchronized at every input to a clocked module [Keh93, Sei94, Gre95, PN95]. In this paper we introduce the *multi-sync* clocking methodology and the *adaptive synchronization*. Together they provide superior operation over other known methods, and for substantially less power.

1.1. Previous work

Synchronizers have traditionally been employed for communications among modules with uncorrelated clocks. Alternative methods that have been proposed include stretchable clocks and clock tuning. Self-clocked data transmission is used for both high and low bandwidth communications, such as Manchester coding on Ethernet [MB76], source synchronous transfer in cache-CPU buses [CM97], and start/stop bits on RS232 serial communications.

The synchronization problem has received a lot of attention [Cha87, CM73, CW75, Gre95, Keh93, Mar81, Pec76, PN95, RMC+88, Sei80, Sei94, Sto82, Vee80, YD96]. Solutions have been developed for a wide range of applications, from intra-chip communications to wide area networks. As technology progresses the integration levels and computational speeds increase, and systems which used to require multi-board implementations are expected to fit inside single chips. Likewise, the synchronization methods that were once applicable to backplanes and multiple boards should now be considered for the inner circles of chips.

Synchronizers are principally suitable for low bandwidth communications, and a number of issues render them less effective in high performance chips. First, synchronizers may occasionally fail due to metastability [CM73, CW75, Mar81, Pec76, Sto82, Vee80]: A synchronizer might enter a metastable state, or take abnormally long time to settle. While the probability of failure has been kept very low, this is exponentially more difficult to achieve when the cycle time becomes aggressively shorter (as described below in Sect. 5). Second, in high performance systems, modules may receive many data inputs concurrently from many other modules and at high rates; consequently, the probability of at least one input switching at the same time as the clock may grow beyond negligible levels. Third, synchronizers incur at least one clock cycle delay; this may lead to unacceptable long latencies accumulating over multi-module paths, and be especially limiting on cyclic paths such as between a reservation station and the execution units of a high performance processor.

Stretchable (or stoppable) clocks [Cha84, Cha87, Pec76, RMC+88, Sei80, YD96] have been proposed as

an alternative to synchronizers. A ring-oscillator based clock generator is attached to each synchronous module. An arbiter detects clock/data conflicts and stretches the 'off' phase of the clock (thus trading failure for a long delay). Stretchable clocks are subject to two drawbacks. First, the multiple clock generators typically develop frequency variations, due to temperature and supply voltage in-die variations. As a result, relative inter-module phase shifts drift continuously, causing frequent recurrences of conflicts. Second, as with synchronizers, high bandwidth communications received over many channels increase the probability of clock/data conflicts. This fact leads to a high rate of clock stretching events, severely impeding performance.

Many other variations have also been proposed. [Keh93] suggests clock (phase and frequency) tuning for performance enhancement. [Sei94] hides some synchronization latency by inter-module FIFO buffers; the main drawback is the latency required. The STARI protocol also employs asynchronous FIFOs to achieve synchronization at the cost of large latency [Gre95]. Synchronization is achieved on the first data transfer, and is automatically maintained thereafter. The FIFO must be kept about half full, and each insertion and removal operation must complete within one cycle. If these requirements are violated (e.g., on FIFO underflow), synchronization is lost, and the system has to be restarted. [PN95] employs analog adjustable clock generators, achieving local self-alignment of all clocks. Unlike clock adjusting methods ([Keh93, PN95, and stretchable clocks), our proposed method adjusts data delays rather than the clocks.

1.2. Multi-Synchronous Systems

A multi-synchronous system is presented in Fig. 1. The common clock is distributed over thin wires (saving area and power, compared to minimal skew clock distribution networks). While clock frequency is the same for all modules, the actual phase shifts are considered unknown. As explained above, these phase differences are considered stationary over large time windows. Each module contains a local clock generator, which amplifies the clock and distributes it internally with minimal skew. Thus, each module operates internally as a traditional synchronous circuit, but asynchronously to all other modules.

Consider modules *A* and *B* in Fig. 1, which are tightly coupled over an asynchronous channel without a FIFO, for low latency high bandwidth communication. Similar to clock delays, the data delay δ_{AB} is also stationary and is considered unknown. Module *A* generates output transitions on D_A at a fixed phase difference relative to its own clock C_A. The data propagate to module *B*, which samples *DataRdy* on the rising edge of its own clock C_B. New data are sent over from *A* to *B* at a high rate, e.g., on almost every clock cycle. Since the relative clock phase difference Δ_A-Δ_B of modules *A* and *B* is presumed unknown, the data may arrive at *B* simultaneously with the rising edge of clock C_B, creating a clock/data conflict and possibly resulting in a metastable state at the input of *B*, and in loss of data. If the relative clock phases and data delays remain fixed (stationary), and since both modules operate at the same clock frequency, this unfortunate situation is most likely to recur. Clearly, the incoming data must be synchronized with C_B at the input to *B*. A regular synchronizer is unsuitable, because the synchronizer may enter a metastable state on every repeated conflict.

Most synchronization methods assume that the arrival (switching) time of data at any module is uniformly distributed over the clock cycle, as in Fig. 2(a). However, in multisync systems, the arrival time of certain data channels incident upon a certain module may be distributed unevenly, e.g., as in Fig. 2(b). When one synchronous module outputs data to another, data output is synchronized with the local clock of the sender. Since the phase difference between the receiver and sender clocks, as well as the data interconnect delay, are stationary, data arrival time at the receiver is correlated with the receiver clock. However, in systems with a high degree of connectivity the combined distribution of all channels incident upon a specific module looks more like Fig. 2(c), and the danger of clock/data conflicts cannot be ignored.

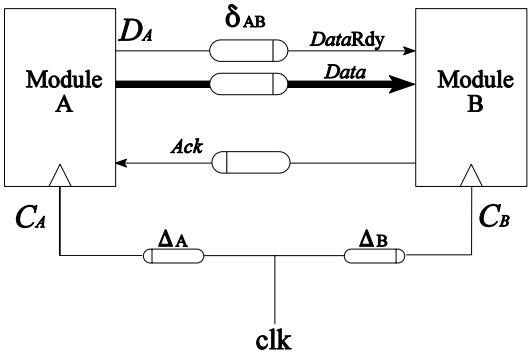

Figure 1: A multi-synchronous system.

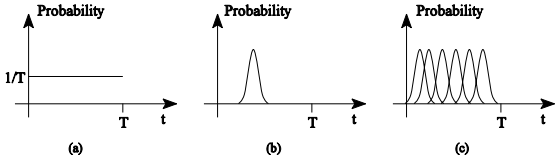

Figure 2: Arrival time distribution of inputs (over a clock cycle *T*): (a) uniform distribution (asynchronous input); (b) clustered distribution when the sender and receiver clocks are correlated (single synchronous input); (c) combined distribution of many independently correlated inputs is similar to uniform distribution (multiple synchronous inputs).

The novel method we propose adjusts data timing (rather than the clock), thus converting data arrival time distributions into forms like Fig. 2(b) and substantially reducing the synchronization problem. Section 2 further defines the method, and a sample adaptive synchronization circuit is described in Section 3. In Section 4 we propose that the adaptive synchronization be performed semi-statically, adapting various data delays from time to time. The method is statistically analyzed in Section 5.

2. Data Adaptive Synchronization

Data adaptive synchronization adjusts the delays on the data lines instead of adjusting the local clock phase. Since the communication channels are connected point to point, the delays on them can be changed so that they do not conflict with the local clock, without affecting the other channels (this approach also applies to bus taps). We add a data coordination circuit for each communication channel, as in Fig. 3(a). When a conflict is detected, the data delay is adjusted to prevent conflicts in future communications.

Note that three different phases are assumed stationary in the multisync model (Fig. 1): The clock phase difference Δ_A-Δ_B, the sender data phase D_A-C_A, and the data delay δ_{AB}. Consequently, the phase of the arriving *DataRdy* at module *B* relative to C_B is also stationary. In other words, the arrival time distribution is represented in this model by Fig. 2(b), and is highly non-uniform. We take advantage of this fact and control data delays so as to assure that the center of this distribution is safely remote from the clock transition for every data line in the system. This is achieved by tuning the data delay δ_{AB}.

The adaptive mechanism architecture for a specific module is shown in Fig. 3(a). Data input channels D_i are subject each to given data delays δ_i (ref. δ_{AB} in Fig. 1). Adaptive synchronization circuits A_i, clocked by the local clock CK_{in}, monitor the *DataRdy*$_i$ lines, and control adjustable data delays λ_i, whose value is in the range $0 \leq \lambda_i < T$ (T is the clock cycle). The function of the A_i circuits is to separate the clock and data transitions. The multiple input delays can be adjusted independently of each other, so the combined data arrival time distribution at the entry to the module looks like Fig. 3(b). Adaptive synchronization applies equally well to single sender, multiple receivers buses (Fig. 4).

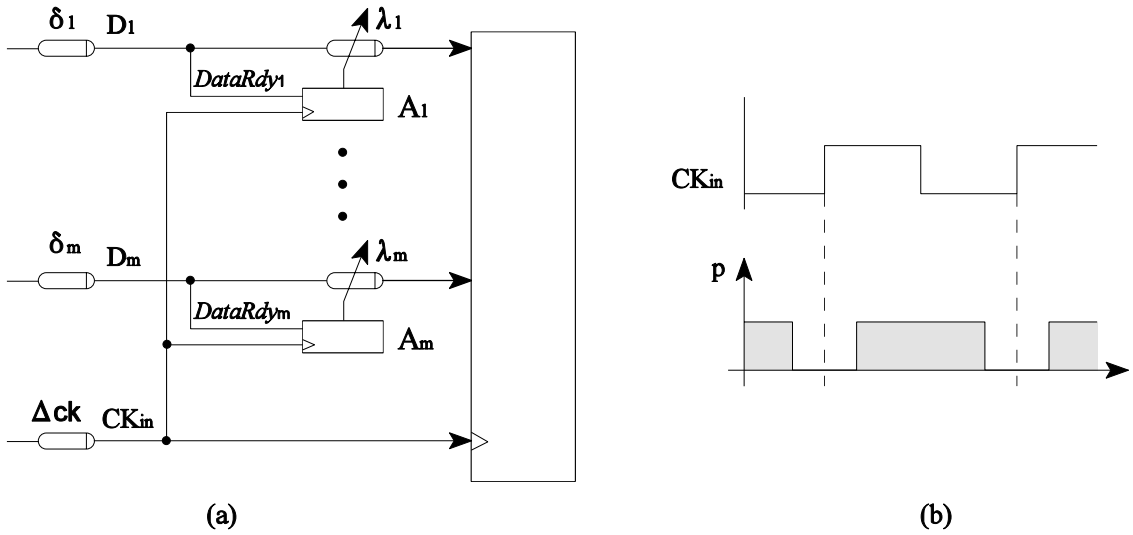

Figure 3: (a) Adaptive Synchronization; (b) Combined data arrival time distribution — data delays are adjusted to avoid conflicts.

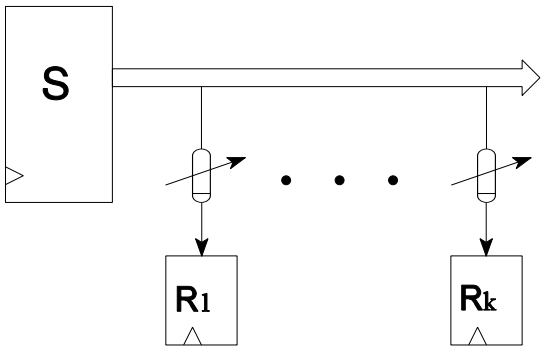

Figure 4: Adaptive Synchronization for single sender, multiple receivers buses; each receiver adjusts its own data input.

3. Data Adaptive Synchronization Circuit

The principles of adaptive synchronization resemble self-clocking communication mechanisms, such as in UARTs. The challenge is to obtain proper operation even at the presence of metastability. Consider the adaptive synchronization circuit in Fig. 5, with an adjustable delay (Fig. 6). A four-phase data signaling discipline is assumed, wherein *DataRdy* rises to '1' after the new data are available (the circuit may be readily extended to two-phase operation as well). The receiving module latches the inputs upon the positive edge of its local clock, and only if *DataRdy* is '1'. Thus, the purpose of the adaptive synchronization circuit is to detect the phase of *DataRdy* relative to the local clock, and to adapt the λ_i delay if that phase is dangerously close to 0 or T (2π).

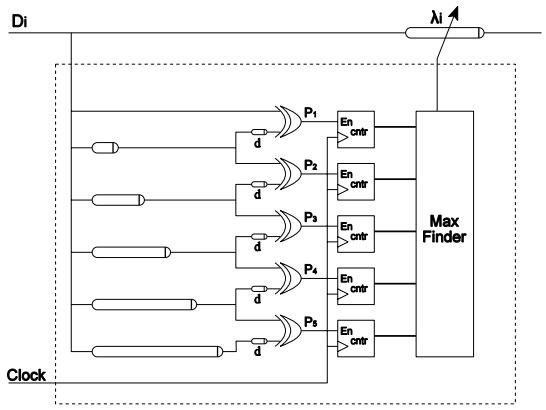

Figure 5: Adaptive Synchronization Circuit.

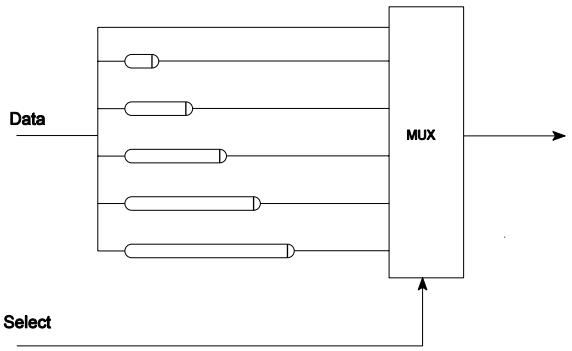

Figure 6: Adjustable delay circuit, consisting of multiple delay lines and a selector.

First, the *DataRdy* line is fed into a statistical phase detector. Let's assume that the *DataRdy* lines switch (up or down) on every cycle. In [Keh93], several delayed phases of the clock are used to detect data transition time. In Fig. 5, several delayed versions of the data are employed instead. The XOR gates generate a sequence of pulses, as in Fig. 7. The delays marked 'd' assure a small pulse overlap. The outputs of the XOR gates are the enable signals of counters which are triggered by the local clock edge. On the rising edge of the clock, one or two of the counters increment their count. This is repeated for a large number of cycles, e.g., 1000 times. At the end of that time, the counters are expected to show a distribution similar to either Fig. 8(a) or Fig. 8(b). In either case, two or three counters show large counts, and the remaining ones are close to zero. The spread is caused by pulse overlap, by clock and delay jitter, and by pulse/clock conflicts which may result in metastable states, in long settling times, and in indeterminate counting. In spite of such physical difficulties, the statistical phase detector is robust thanks to many repeat counts, and it produces a very clear indication of the relative phase of the *DataRdy* line. The circuit in Fig. 5 is similar to delay lock loop (DLL) circuits, except that the proposed circuit is digital rather than analog, and its operation is algorithmically controlled. Other types of phase detectors may also be employed.

Next, the *MaxFinder* circuit determines, according to which counter has won, if the λ_i delay of the data lines should be changed, and by how much. For example, the count depicted in Fig. 8(a) indicates no change, while that of Fig. 8(b) calls for adding a delay of at least T/5. The adjustable delay consists of multiple parallel delay lines and a selector (Fig. 6). Notice that although the phase detector examines only the *DataRdy* line, λ_i is applied to all data lines of the *i*'th channel.

Although the examples present circuits for the case where the clock cycle is divided into five periods, at high frequencies it might be simpler to implement using only three such periods (since the clock cycle time may be only a few gate delays long). The circuit complexity of a proper adaptive synchronization circuit, for a 32-bits data path, is approximately 5,000 transistors (comprising the delays, XOR gates, counters, comparators and switches in the *MaxFinder* circuit, and the adjustable delay circuits). Thus, the total overhead per a 5M transistor module with 10 input channels is about 50,000 / 5M = 1% (recall that it replaces a massive clock distribution network). The extra power consumption is similarly marginal.

Adaptive synchronization is suitable for a wide range of applications. Typical data delay range is $0.1T < \delta_i < 1.5T$ for a 0.5B transistors, 1GHz chip, but the delay may be much larger than T for multi-chip and MCM configurations. In such cases, new asynchronous data signaling methods could be used, such as multiple message windows (wherein multiple messages are sent before an acknowledge is expected). As long as relative delays are stationary, adaptive synchronization remains applicable.

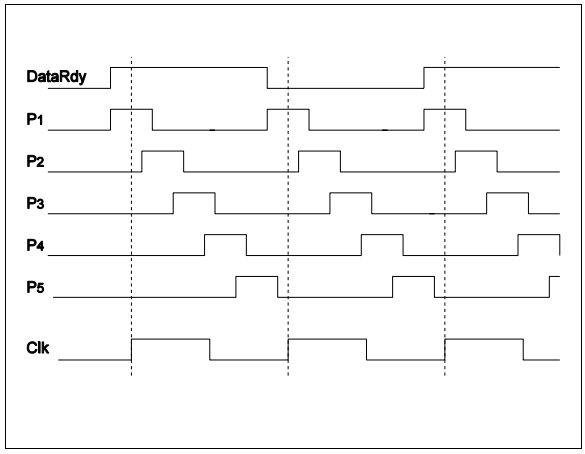

Figure 7: Phase detection waveforms.

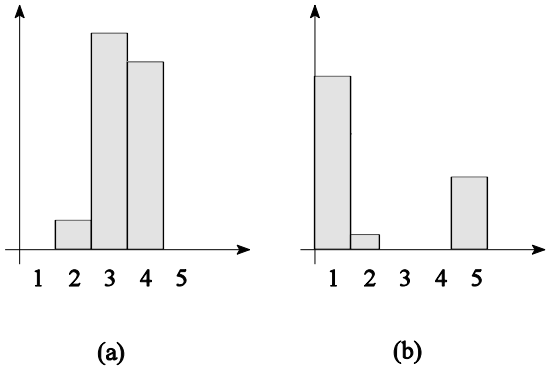

Figure 8: Typical phase detection counter outputs: (a) data transition is safely within the cycle; (b) data delay should be increased to avoid clock/data conflicts.

4. Training Sessions

Adaptive synchronization may be performed continuously, in parallel with normal circuit operation. However, modifying the data delays may cause timing problems at the time of change, so this is best carried out while the system is not performing any real task. In addition, during normal operation it cannot be guaranteed that all *DataRdy* lines switch frequently enough. And continuous adaptation may be unnecessary if all delays are highly stationary and stable.

Consequently, special training sessions are proposed for adaptive synchronization. During a training session the system stops performing all real computations. Instead, all *DataRdy* lines are toggled every cycle, and all adaptive synchronization circuits operate and adjust the δ_i delays. Any synchronization failures during a training session can obviously be ignored. The training session requires a relatively small number of counting cycles. Since all adaptive synchronization circuits operate in parallel, 100,000 clock cycles (0.1mS at 1GHz) seems a safe bound on the required session duration.

A training session is always employed after reset, for initial adjustment of all delays. Thereafter, training sessions can be invoked either periodically or as required. Periodical training frequency depends on process parameters (especially delay stability) and operational parameters (such as clock frequency and dynamic temperature and voltage variations), but it is estimated that at 2010 technology much less than one training per second will be required. The expected performance overhead is thus much less than 10^5 cycles / 10^9 Hz = 0.01%.

Training sessions are also proposed in [SCI92], wherein a point-to-point communication ring architecture is defined. Training sessions are utilized to send sync packets at ringlet initialization, and once every time interval appropriate for normal operation of the particular implementation. Clock skew in [SCI92] is handled (using Phase Lock Loop circuits) by observing incoming clock and local clock phases.

Significant temperature and voltage variations may be sensed on-chip by special sensors in order to invoke a training session when a problem seems imminent. Alternatively, the adaptive synchronization circuits themselves may be modified to act as the sensors. If any such circuit detects that any switching phase

approaches 0 (or 2π) closer than some safety threshold, a hardware interrupt is invoked to start a training session. In addition, a training session can also be triggered when a higher level logic (or software) detects a synchronization or communication failure. A similar tuning idea is used in [Keh93].

5. Probability of Synchronization Failure

In this section we analyze the failure probability of the adaptive synchronization (A/S), and compare it to synchronizers.

Synchronization failure might happen at a training session, failing the delay adaptation process and causing the system to fail, or during regular operation, after a successful training. Failures during training sessions do not affect system operation, and might only cause the training itself to fail. These synchronization failures can happen in the phase detection circuit (Fig. 5), when one of the counters enters a metastable state while incrementing its count, due to marginal triggering. Since a training session takes many cycles, the counters are allowed sufficient time to resolve any metastability before their outputs are read. Thus, the probability of failure of the training session is practically zero. After a successful training session, all delays are adapted properly so that data are expected to arrive at a module around the middle of the local clock cycle, and avoid synchronization failures. However, due to possible jitters in clock phase and line delays, the data arrival time might randomally change from cycle to cycle, and become dangerously close to a clock edge.

The system model for the failure analysis is described in Fig. 9. The phase of the clock at module *B* is affected by the delay along the clock distribution network from the clock source to module *B*. Data sent from module *A* will arrive at module *B* with a phase affected by the delay of the clock signal to module *A*, the internal logic delay from clock edge to data output, and data propagation delay to the input of *B* (Fig. 9(a)). We assume normally distributed jitters, and define two random variables with normal (Gaussian) distribution, X_c and X_d, representing the phases of the clock and data at module *B*, respectively. $X_c = \boldsymbol{N}(\mu_c,\sigma_c)$ is normally distributed with mean μ_c (equal to the clock cycle time T) and standard deviation σ_c (caused by jitter effects). Without loss of generality, we take the phase of the clock to be 0 (i.e., μ_c=T), since we are only interested in the relative phase of data to clock, and cyclically the phase is $2\pi k$ (k an integer). $X_d = \boldsymbol{N}(\mu_d,\sigma_d)$, wherein μ_d is the expected arrival time within a clock cycle. After a training session, μ_d is expected to be at the middle of the clock cycle, i.e., μ_d= T/2, assuming X_c is centered on $0+2\pi k$ (see Fig. 9(b)). Note that X_d is actually a sum of three normally distributed variables, so its variance (σ^2_d) is calculated as the sum of three variances. Assume ϵ is the time window within a clock cycle (Fig. 9(c)) in which data must be stable (generally considered to be the setup-and-hold period) to avoid metastability.

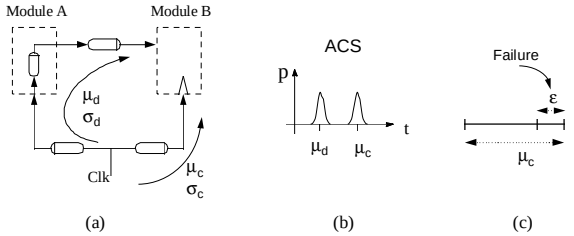

Figure 9: Model for analyzing synchronization failure.

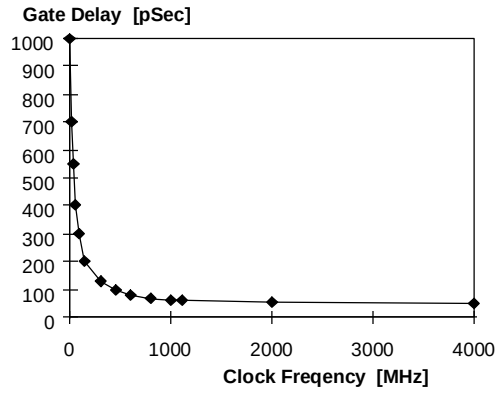

Figure 10: Gate delay vs. clock frequency.

When using a synchronizer, there is no knowledge of the data arrival time, so uniform arrival time distribution is assumed. Once a synchronizer has entered the metastable state, the probability that it will still be metastable some time later has been shown to be an exponentially decreasing function [Cha83, RC82]. The probability of synchronization failure of a synchronizer is given by

(1) $$P_{failure}(\textit{Synchronizer}) = P[meta\ |t=0] \times P[meta\ |t=(\mu_c-\epsilon)] = \frac{\epsilon}{\mu_c} \times e^{-\frac{(\mu_c-\epsilon)}{\tau}}$$

It equals to the probability that a synchronizer which enters a metastable state (at time *t=0*), still remains in the metastable state at the time its output should be stable for sampling in the next clock cycle. The parameter τ is the exponential time constant of the decay rate of the metastability (discussed below).

The failure probability of adaptive synchronization is the probability that the values of the two random variables X_c and X_d are too close (within ϵ) to each other, i.e., the data switch too close to the clock edge. This probability can be calculated as the probability that a random variable, equals to the difference of the two random variables, has a value in the forbidden range:

(2) $$P_{failure}(A/S) = P[-\epsilon \le (X_c - X_d) \le \epsilon] + P[-\epsilon \le (X_d - X_c) \le \epsilon]$$

Note that the normal distribution of the difference random variable spans beyond [0,T], and because of the 2π cycling, X_c should be considered at both 0 and T. Since we assume normal distributions, each of the probabilities in Eq. 2, can be calculated by the Gaussian function, with the proper parameters [Pap91], e.g.,

(3) $$P[-\epsilon \le (X_c - X_d) \le \epsilon] = \mathbf{G}\left(\frac{\epsilon - (\mu_c - \mu_d)}{\sqrt{\sigma_c^2 + \sigma_d^2}}\right) - \mathbf{G}\left(\frac{-\epsilon - (\mu_c - \mu_d)}{\sqrt{\sigma_c^2 + \sigma_d^2}}\right)$$

The value of the Gaussian function is determined by the error function, *erf(x)*, whose value can be obtained by the *ERF(x)* function with parameter transformation:

(4) $$\mathbf{G}(x) = \frac{1}{2} + \mathrm{erf}(x); \qquad \mathrm{erf}(x) = \frac{1}{2}\,\mathrm{ERF}\left(x/\sqrt{2}\right)$$

Technology is defined by the gate delay, which also limits the highest clock frequency that can be used. However, the clock frequency increases faster than the gate delay decreases, as can be observed from Fig. 10 (based on data from [KG97, SIA94, Wei96]). Since gate delay does not scale linearly with frequency, less gate delays are available in a clock cycle time, as frequency rises. The probability of failure goes up because the clock cycle time T(=μ_c) shrinks faster than ϵ (the settling window). To compare the failure probabilities, we assume the following model: The metastability window ϵ width is assumed to be equal to a gate delay, the parameter τ is taken as 1/3 of a gate delay, and the jitter (which equals 6σ) is half a gate delay (and no more than 15% of the clock cycle). Figure 11 presents a logarithmic graph comparing the synchronization failure probabilities of a synchronizer relative to the A/S scheme.

For high communication bandwidth (e.g., almost every cycle), the mean time between failures (MTBF) is given by

(5) $$MTBF = \frac{1}{P_{failure} * f_c}$$

The failure probabilities required to achieve an MTBF of once a year and once a minute at the various technologies are also presented in the graph. As can be observed from the graph, using a synchronizer can be practical for lower frequencies, but as clock frequency increases, the synchronizer has less time to resolve and the probability of failure rises rapidly. Using a sequence of synchronizers decreases the failure probability, but increases the latency and affects performance. Note also that the failure probability presented is of a single synchronizer, and since many synchronizers are required (for every bit in every bus between modules), the failure probability is worse than drawn on the graph. When synchronizers fail to deliver a flawless operation at higher frequencies, A/S still applies. The zero values of A/S failure probability cannot be plotted on the logarithmic graph. The inter-module clock jitter will be the limiting factor on maximum clock frequency in A/S scheme. At even higher frequencies, when A/S fails, it can be used together with a synchronizer, to decrease the probability of the synchronizer entering a metastable state. Beyond a certain technology (e.g., when the jitter is more than 15% of the clock cycle), all synchronization methods fail, and the only solution is to use a complete asynchronous design, with asynchronous communication.

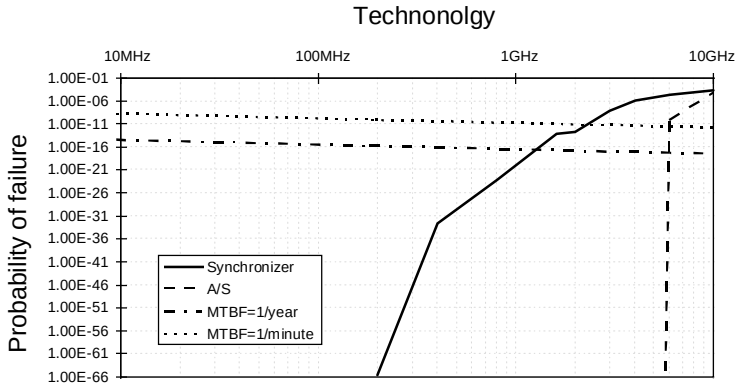

Figure 11: Probability of synchronization failure.

6. Conclusions

The technological constraints applicable to future large and complex chips imply at least partially asynchronous operation. A single clock is either impractical or impossible for such very high performance chips, e.g., as predicted by the SIA technology roadmap for the year 2010 (over 0.5B transistors operating at over 1GHz clock) [SIA94 / SIA97]. We have presented an adaptive synchronization solution for multi-synchronous systems. Multi-synchronous architectures (locally synchronous, globally asynchronous) could be a viable alternative to fully asynchronous design. We focus on common clock multi-synchronous systems, wherein a single crystal clock is distributed over minimal area and minimal power networks, so that all modules operate on the same clock frequency but at unknown phase differences.

We have presented a novel adaptive synchronization method, addressing multisync systems. While most previously proposed methods manipulate the clock, adaptive synchronization adjusts data delays. The method

exploits the high stability of delays and the stationarity of most relative phases. Data timing is dynamically adjusted to avoid clock/data conflicts. The probability of synchronization failure is reduced substantially. Timing adaptation can be limited to special training sessions (as commonly practiced in data communication networks). Thus, the synchronization monitoring circuits are kept off the critical paths. The adaptation circuits incur only marginal overhead in area, power and performance. A study of alternative methods (such as synchronizers and stretchable clocks) shows that they may not be as usable as adaptive synchronization. In contrast with fully asynchronous architecture, multisync design with adaptive synchronization allows the continued use of existing know-how in synchronous design. Asynchronous aspects are limited to system architecture and some circuit design, while the majority of the chip can be designed in synchronous subsystems.

References

[Bow95] W. J. Bowhill, et. al., "Circuit Implementation of a 300-MHz 64-bit Second-generation CMOS Alpha CPU," *Digital Technical Journal*, **7**(1), pp.100-115, 1995.

[Cha83] T. J. Chaney, "Measured Flip-Flop Responses to Marginal Triggering," *IEEE Trans. on Computers*, **32**(12), pp. 1207-1209, Dec. 1983.

[Cha84] D. M. Chapiro, *Globally-Asynchronous Locally-Synchronous Systems*, PhD thesis, Dept. of Computer Science, Stanford Univ., 1984.

[Cha87] D. M. Chapiro, "Reliable High-Speed Arbitration and Synchronization," *IEEE Trans. on Computers*, **36**(10), pp. 1251-1255, Oct. 1987.

[CM73] T. J. Chaney and C. E. Molnar, "Anomalous Behavior of Synchronizer and Arbiter Circuits," *IEEE Trans. on Computers*, **22**(4), pp. 421-422, Apr. 1973.

[CM97] M. Choudhury and J. Miller, "A 300MHz CMOS Microprocessor with Multi-Media Extensions," *Proc. ISSCC'97*.

[CW75] G. R. Couranz and D. F. Wann, "Theoretical and Experimental Behavior of Synchronizers Operating in the Metastable Region," *IEEE Trans. on Computers*, **24**(6), pp. 604-616, Jun. 1975.

[DGY93] I. David, R. Ginosar, and M. Yoeli, "Self-Timed Architecture of a Reduced Instruction Set Computer," in *Asynchronous Design Methodologies*, S. Furber and M. Edwards editors, IFIP Transactions Vol. A-28, Elsevier Science Publishers, pp. 29-43, 1993.

[Fri95] E. G. Friedman, editor, *Clock Distribution Networks in VLSI Circuits and Systems*, IEEE Press, 1995.

[Gre95] M. R. Greenstreet, "Implementing a STARI chip,", *ICCD'95*, pp. 38-43, 1995.

[Hau95] S. Hauck, "Asynchronous Design Methodologies: An Overview," *Proc. IEEE*, **83**(1), pp. 69-93, Jan. 1995.

[Keh93] T. Kehl, "Hardware Self-Tuning and Circuit Performance Monitoring," *ICCD'93*, pp. 188-192, 1993.

[KG97] R. Kol and R. Ginosar, "Future Processors will be Asynchronous (sub-title: *Kin*: A High Performance Asynchronous Processor Architecture)," Technical Report CC PUB#202 (EE PUB#1099), Department of Electrical Engineering, Technion, Israel, Jul. 1997.

[Kol97] R. Kol, *Self-Timed Asynchronous Architecture of an Advanced General Purpose Microprocessor*, PhD thesis, Dept. of Electrical Engineering, Technion, Israel, 1997.

[Mar81] L. R. Marino, "General Theory of Metastable Operation," *IEEE Trans. on Computers*, **30**(2), pp. 107-115, Feb. 1981.

[Mar97] A. Martin, *et al.*, "The Design of an Asynchronous MIPS R3000 Microprocessor," *Proc. Advanced Research in VLSI*, Sept. 1997.

[MB76] R. M. Metcalfe and D. R. Boggs, "Ethernet: Distributed packet switching for local computer networks," *Comm. ACM.*, **19**, pp. 395-404, Jul. 1976.

[Pap91] A. Papoulis, *Probability, Random Variables, and Stochastic Processes*, 3rd edition,

McGraw-Hill, 1991.

[Pav94] N. C. Paver, *The Design and Implementation of an Asynchronous Microprocessor*, PhD thesis, Dept. of Computer Science, Univ. of Manchester, 1994.

[Pec76] M. Pechoucek, "Anomalous Response Times of Input Synchronizers," *IEEE Trans. on Computers*, **25**(2), pp. 133-139, Feb. 1976.

[PN95] G. A. Pratt and J. Nguyen, "Distributed Synchronous Clocking," *IEEE Trans. on Parallel and Distributed Systems*, **6**(3), pp. 314-328, Mar. 1995.

[RC82] F. U. Rosenberger and T. J. Chaney, "Flip-Flop Resolving Time Test Circuit," *IEEE J. of Solid-State Circuits*, **SC-17**(4), pp. 731-738, Aug. 1982.

[RMC+88] F. U. Rosenberger, C. E. Molnar, T. J. Chaney, and T.-P. Fang, "Q-modules: Internally clocked delay-insensitive modules," *IEEE Trans. on Computers*, **37**(9), pp. 1005-1018, Sep. 1988.

[SCI92] IEEE std 1596-1992, *IEEE standard for Scalable Coherent Interface (SCI)*, 1992.

[Sei80] C. L. Seitz, System timing, in C. A. Mead and L. A. Conway, *Introduction to VLSI Systems*, Ch. 7, Addison-Wesley, 1980.

[Sei94] J. N. Seizovic, "Pipeline Synchronization," *Proc. Int. Symp. on Advanced Research in Asynchronous Circuits and Systems*, pp. 87-96, 1994.

[SIA94] SIA, *The National Technology Roadmap for Semiconductors*, 1994 (See also: *http://www.sematech.orh/public/roadmap/index.htm*).

[SIA97] The 1997 SIA Technology Roadmap will be published in Dec. 1997, and will predict up to 10GHz clock by 2012. We will cite it in the final paper instead of the 1994 edition.

[SSM94] R. F. Sproull, I. E. Sutherland, and C. E. Molnar, "The Counterflow Pipeline Processor Architecture," *IEEE Design & Test of Computers*, **11**(3), pp. 48-59, Fall 1994.

[Sto82] P. A. Stoll, "How to Avoid Synchronization Problems," *VLSI Design*, pp. 56-59, Nov./Dec. 1982.

[Vee80] H. J. M. Veendrick, "The Behavior of Flip-Flops Used as Synchronizers and Prediction of Their Failure Rate," *IEEE J. of Solid-State Circuits*, **15**(2), pp. 169-176, Apr. 1980.

[Wei96] U. Weiser, "Future Directions in Microprocessor Design," Invited lecture, presented at *2nd Int. Symp. on Advanced Research in Asynchronous Circuits and Systems (Async'96)*, Mar. 1996.

[YD96] K. Y. Yun and R. P. Donohue, "Pausible Clocking: A First Step Toward Heterogeneous Systems," *ICCD'96*, pp. 118-123, 1996.