Adaptive Synchronization

Ran Ginosar and Rakefet Kol

VLSI Systems Research Center, Technion--Israel Institute of Technology, Haifa 32000, Israel

ran@ee.technion.ac.il

Abstract

Delay variations are typically accounted for by increasing cycle time margins. Adaptive Synchronization eliminates this on inter-modular interfaces in very large, high performance chips. The chip is divided into multiple smaller synchronous modules or clock domains. Multi-synchronous hierarchical clocking provides the same frequency to all modules, but does not maintain any particular phase. Adaptive synchronizers compensate for the time-varying inter-modular clock and data phases, and out-perform conventional synchronizers. A novel modeling approach to delay variations is introduced to support Adaptive Synchronization.

Introduction

On-chip clocks in high performance processors are typically distributed over a balanced tree network, where the delay from the root to each and every leaf is the same [1], [2], [3]. This provides for reliable synchronization of on-chip interconnects. However, achieving balanced clock trees in future high performance chips integrating large systems (of four billion transistors and operating at close to 20 GHz [4]) is becoming more difficult and less effective as delay variations grow in proportion to the clock cycle. This motivates a search for architectures that do not rely on a single, zero skew clock available for the entire chip. A fully synchronous clock can be maintained within smaller modules, but inter-module timing may be controlled less tightly.

To accommodate faster clocks and larger dies, contemporary synchronous chips are divided into multiple clock domains, where each domain maintains its own single synchronous clock. How many clock domains are needed? As clocks become faster, the distance traveled by a data signal during a single clock cycle becomes progressively shorter. We define *effective wavelength* as the distance propagated by the electric signal on metal interconnect during one clock cycle, where by 'propagated' we mean the ability to drive a gate rather than the mere electromagnetic wavefront propagation. At typical such driving-capability propagation of c/40 (c=speed of light) and with a 10 GHz clock, the effective wavelength is only 0.75mm. Assuming a clock domain no larger than 0.5x0.5 effective wavelength, a 36x36 mm^2 die would need about 10,000 clock domains to maintain proper synchronous operation. This is obviously a difficult challenge (a similar analysis is given in [5]).

Treating on-chip data interconnects between separate modules or clock domains as either asynchronous or source-synchronous inputs and synchronizing them at the receivers has been proposed for such systems. Methods similar to inter-chip serial link synchronizers [6], [7], [8] could be used but the related latency overhead may be too high for on-chip interconnects. Synchronization is statistically subject to failures and metastability; when the data and clock switch approximately at the same time, synchronization typically takes longer to resolve and, in rare cases, fails completely [6]. To reduce failure probability, multiple synchronizer stages are employed in series, incurring an added latency of at least one and often more than one clock cycles (Fig. 1). Typically, synchronization resolution requires a certain number of gate delays [7]; while clock frequencies increase with the advent of technology, gate switching speeds do not scale as fast, and the number of gate delays per clock cycle diminishes. Thus, as technology progresses, more clock cycles are needed to obtain the same level of synchronization. In addition, when successive delays are correlated (as explained below), normal synchronization schemes are subject to repeated failure patterns: Once data and clock coincide, they are highly likely to conflict again in the following cycle.

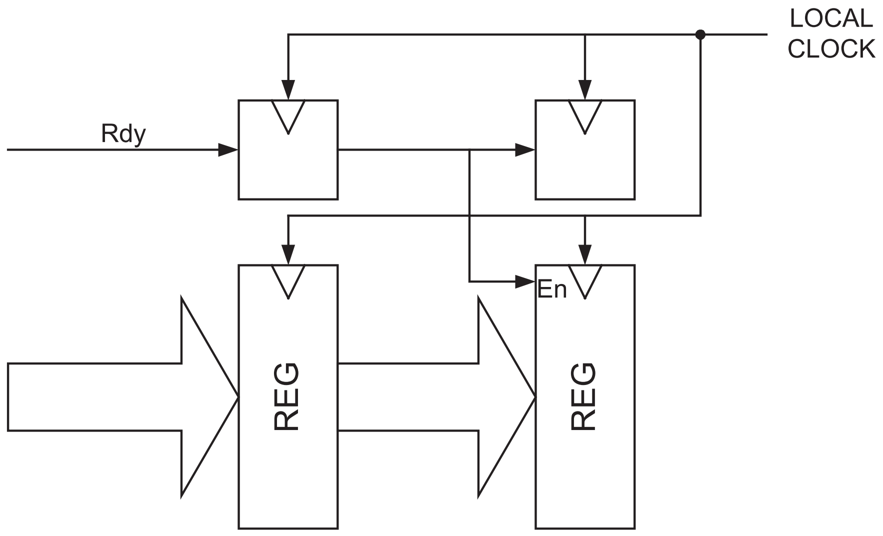

Fig. 1: Bundled data synchronization. The Ready signal is assumed stable no later than one clock cycle after it is first latched. It is used to enable the second data latch. If this is too short, additional synchronizer stages would be required.

Pipeline synchronizers [9], [10], a proposed alternative to regular synchronizers, incur substantial latency. In such a scheme, the multiple stages of synchronization are distributed over a number of pipeline stages. However, pipeline synchronization is not a practical method for large chips: Mixing computational pipelines and synchronization is incompatible with structured design methodologies and with IP core-based SOC (System on Chip) designs.

Others propose to tune the clocks [11], [12], [13], [14]. According to that scheme, the clock is locally generated by each module, and an arbiter occasionally defers the clock if the incoming data switch simultaneously with the clock. There are two problems associated with stoppable clocks. First, the frequency of a locally generated, non-crystal based clock varies with temperature and supply voltage, and thus the chip cannot be specified at one dependable frequency. This drawback may or may not matter, depending on the application.

The other problem is more severe: In large systems, many modules intercommunicate with a large number of other modules, say 10. When one incoming bus faces a data-clock conflict, the clock is shifted a bit. This shift may cause a new conflict with another data bus incident upon the same module, and the clock is shifted yet again. This new shift may now cause yet another conflict, and so on. Depending on the number of incident buses and the clock shift resolution, this problem may evolve into a cyclical never-ending dance of the clock.

It turns out that for many on-chip and MCM applications, even when tight clock phase control over the entire chip becomes impractical, the relative timing of nearby modules or clock domains is not asynchronous but is highly correlated. We propose that tuning of data interconnects (rather than the clock) addresses these issues. A hierarchical *multi-sync* clock abandons phase control at the high level, and *adaptive synchronization* provides for efficient on-chip communications.

Delay Variations and Correlation

We propose to model delay variations as follows. The delay of any specific gate or wire is sampled repeatedly over time. The resulting function of time is Fourier-transformed, yielding a frequency characterization of the variation of the delay. Such a presentation is shown, on a *log*F scale, in Fig. 2. Typical clock and data delay variations are separated into four distinct types in Fig. 2:

- *Skew* comprises all intra- and inter-chip delay non-uniformities that are constant over time. They result from variations in V_{th} and geometric dimensions. Due to such statistical variations, two different gates which are drawn the same in two different places on the chip may end up manufactured with slightly different dimensions, and thus may incur different delays. The same fact is true for one specific gate implemented in two different samples of the same chip. Once fabricated, these delay differences remain unchanged for the life of the chips. Since the skew never changes, it is represented by the δ function at DC (the origin of the graph in Fig. 2). Note that we restrict the meaning of the term 'skew' relative to its common use.

- *Jitter* relates to fast (cycle-to-cycle) delay variations. Delays vary between cycles mostly due to power supply coupling: As some gates switch, they draw switching current from the supply lines, leading to momentary decrease in voltage near the gates. This voltage reduction, in turn, results in slower switching speeds of nearby gates which are fed by the same supply lines. Since not all gates switch every cycle, the exact voltage functions vary from cycle to cycle and from place to place. Another closely related source of jitter is instantaneous temperature: Excessive switching of some gates causes a brief increase in substrate temperature around them, and this in turn may slow down nearby gates. While these phenomena are data-dependent and could, in principle, be predicted, they are so complicated that it makes much more sense to treat these variations as random processes, namely jitter. Additionally, some jitter is attributed to capacitive and inductive cross-talk. To address the jitter problem, it is first estimated and then a suitable safety margin is added to all switching times. In other words, although a full clock cycle could theoretically allow a certain number of gate delays*,* we usually design synchronous circuits with only a smaller number of gates in series to allow for the jitter uncertainty. Since it is impractical to construct circuits with margins beyond, say, 30% of the total clock cycle, the designer always insures that the jitter is kept well under such limits. This is ensured by expensive power buffering, by geometric design rules, and by limiting the clock frequency of the circuit; all measures unfortunately limit circuit performance. The constrained jitter appears as a constant level over all frequencies in Fig. 2.

- *Drift* is similar in principle to jitter, but it relates to much slower variations of delay. As computations vary, e.g. when a processor starts to execute a floating point computation that exercises the floating point unit intensively, the average temperature around that unit may rise a bit and the average supply voltage may drop a bit. These changes could, in turn, affect the delay of other nearby units. Once the processor changes its computing pattern, e.g. completes the floating point subroutine, local temperature and voltage may resume their previous values, resulting in another delay shift. Such slow variations tend to be cumulative over a large number of cycles, because the voltage and temperature factors affect unidirectional changes. This is in contrast to jitter, which is not correlated from cycle to cycle. Thus, over a large number of cycles, the drift may accumulate into significant changes of delay. This fact is reflected in the triangular shape of the drift in Fig. 2.
- Another source of delay variation consists of very fast switching and clock harmonics. These variations are typically much harder to characterize and are beyond the scope of this treatment. They appear as the dashed line beyond the clock frequency in Fig. 2.

While jitter is constrained to much less than a clock cycle, skew and drift may present delay variations of dangerous dimensions, and their relative threat increases as the clock cycle decreases. Adaptive Synchronization is designed to counter those two sources of delay variation on intra-chip communication links. Since skew is fixed in time and drift is band-limited to F_D (Fig. 2), it is sufficient to apply adaptation at a rate that is slightly higher than F_D. In applications in which drift is insignificant, one-time power-up adaptation may be all that is required to account for skew.

Multi-Synchronous Clocking

In the proposed method, the system is decomposed into *modules* which are sufficiently small (say, one million transistors) so that they may comprise individual clock domains (Fig. 3). A global network distributes a single *multi-synchronous* global clock to all modules. While the exact same frequency is provided to every module, the relative phase is undetermined. Since the global clock network needs not guarantee the phase, it may be designed to conserve power and area. Each module regenerates its local clock, by reshaping and (if necessary) frequency-multiplying the global multi-sync clock. While the frequency of the local clock is the same as in other modules, its relative phase is *a-priori* unknown, and may change over time. The relative phases are affected by delay variations, as discussed above. Since the skew is constant and the drift changes very slowly, we say that (up to jitter) the relative phases are *stationary*. In other words, we can identify relatively long periods of time over which the relative phases may be considered fixed. We take advantage of this fact with *Adaptive Synchronization*.

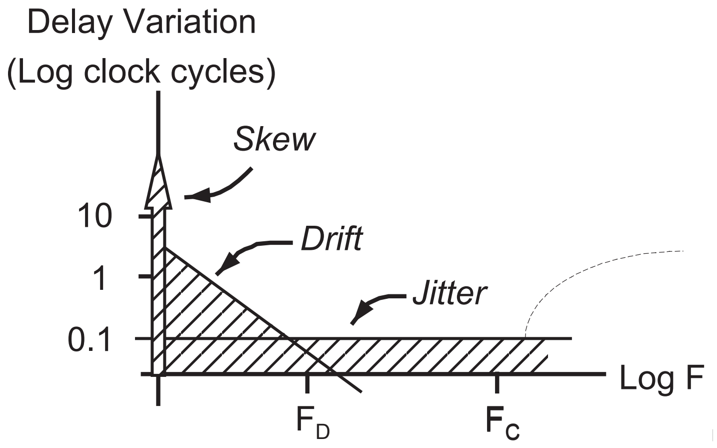

Fig. 2: Frequency-domain representation of characteristic delay variations, measured in units of the clock cycle. *Skew* is constant over time, *jitter* is amplitude-limited to a small portion of the clock cycle, and *drift* represents band-limited cumulative variations.

Adaptive Synchronization

Modules communicate over data channels, which provide Data Ready (*Rdy*) signals. The clock delays D_A , D_B and the data delay D_{AB} in Fig. 3 are all stationary, and all clocks are of the same frequency. Thus, the arrival time of the *Rdy* signal at module B is typically correlated with the receiver's clock, CB.

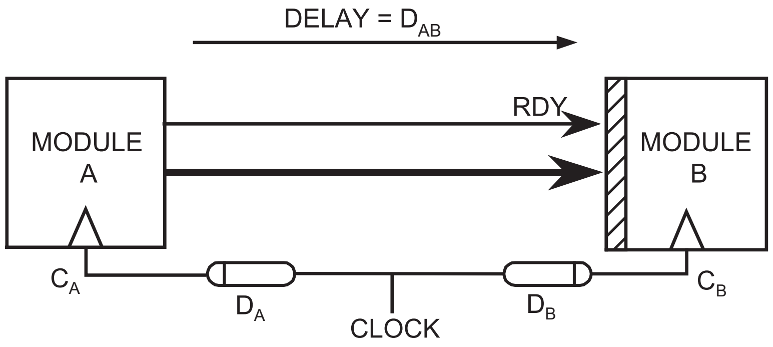

Fig. 3: Multi-sync clocking

The shaded interface of Module B is detailed in Fig. 4. Each data input bus D_i is equipped with an *Adaptive Synchronization* (A/S) circuit that detects the particular phase difference between *Rdy*$_i$ and the local clock. A number of phase detection and locking circuits may be used ([6], [15]), and they typically employ low pass filtering over multiple cycles to smooth out metastability. A simple A/S circuit is shown in Fig. 5a. Data / clock collisions are detected by the four ME elements in Fig. 5b. Given that the delay lines are *d* sec. long, the circuit detects when $|\ t(\text{Data}) - t(\text{Clock})\ | \leq d$ for both rising and falling data edges. The ME elements are shown in Fig. 5c. The circuit also contains a digital delay line (Fig. 5d) controlled by the counter. To start an adaptation cycle, the controller resets the counter. As long as there is a collision, the counter increases the programmable data delay. If we design the conflict threshold *d* to be close to, but slightly less than, half a clock cycle, we can guarantee that all data transitions are positioned as far away from clock transitions as possible, and typically at the center of the clock cycle, so as to minimize jitter sensitivity.

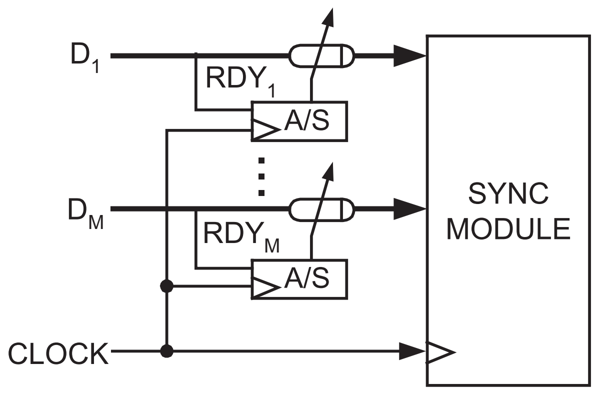

Fig. 4: Adaptive Synchronization

Adaptation is achieved by means of a *training session* as follows. Normal operation of the chip is suspended. The sender side of each data bus generates a sequence of dummy transmissions on the RDY lines. The receiver side performs adaptation until the receiving delay is adjusted as above. Normal operation resumes after a predefined period of time, assuming all adaptations are complete.

Adaptation can be performed according to one of the following five modes.

- One time adaptation at test / burn-in right after manufacture of each chip. This mode requires some form of permanent setting of the delays. It provides adjustments of skew only.
- Power-up adaptation is performed once power is applied. Only skew is accounted for, and no permanent setting is required.

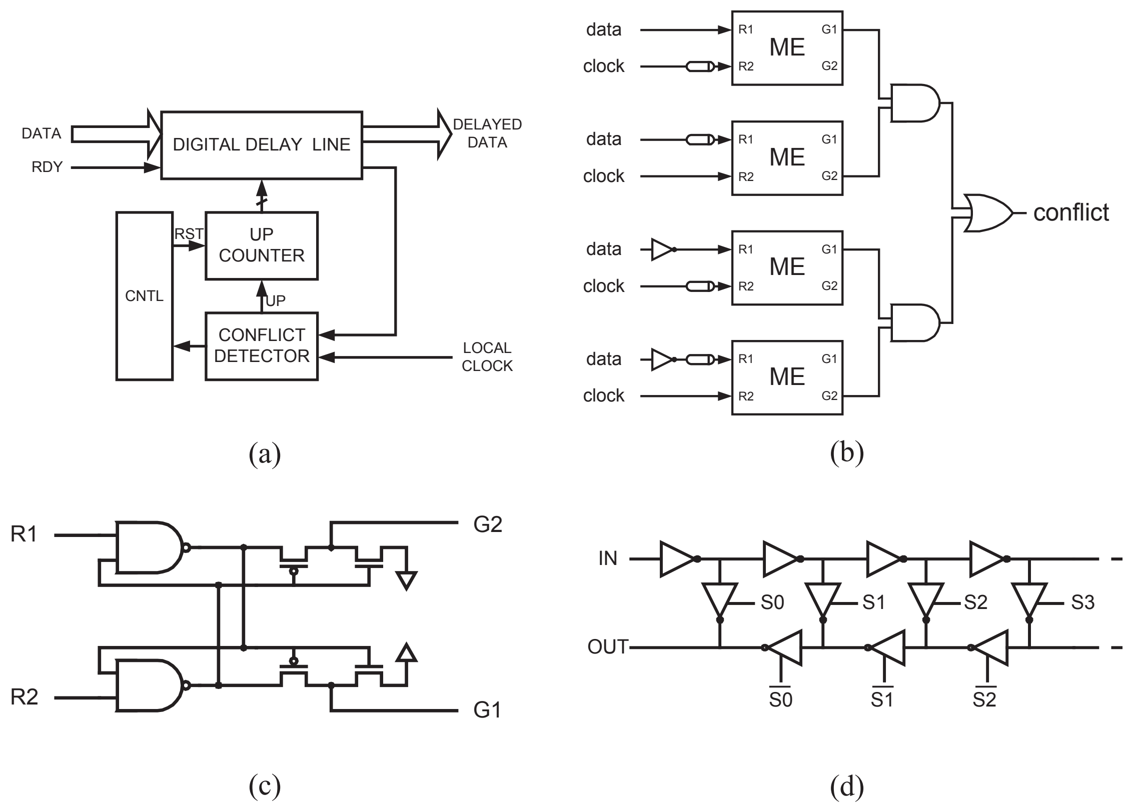

Fig. 5: Adaptive Sensitivity implementation (a), the Conflict Detector (b), a Mutual Exclusion element (c), and the Digital Delay Line (d).

- Periodic training sessions. In this mode, the normal operation of the chip is suspended and an adaptation session is called for at a rate slightly higher than F_D (the drift cut-off frequency, Fig. 2). This mode compensates for both skew and drift variations.
- Triggered training sessions. Once adjusted, the various phase detectors can also serve to detect and flag suspected variations as they develop. Since drift is cumulative, the phase detectors show a growing phase difference. A threshold can be employed to trigger a training session once any one of the phase detectors has detected a phase variation beyond the preset threshold. This scheme optimizes the use of Adaptive Synchronization in an adaptive manner.
- Continuous tracking -- the A/S mechanism may be allowed to continually adjust the delays, avoiding accumulation of any drift, simultaneously with normal chip operation. An up/down counter is used in Fig. 5a instead of the up-counter. In this mode, no training sessions are needed, but it may require higher power dissipation than the other ones.

Stochastic analysis shows that failure probability of A/S is substantially lower than standard synchronizers [16]. An asynchronous signal has non-zero probability of coinciding with the clock, leading to non-zero failure probability. Adaptive Synchronization accounts for skew and drift variations, but is still subject to jitter. Since A/S tunes the data transition to the middle of the clock cycle (about one half of a clock cycle from each sampling clock transition), and since jitter is restricted to well below one half a clock cycle, the probability of clock / data conflict is substantially lower than in the asynchronous case.

Discussion

A/S adapts the circuit to the delay variations, rather than accounting for them as margins added to the clock cycle. On average, it incurs lower latency than standard synchronizers (one half cycle, as opposed to one or more cycles), thus resulting in faster chips. It increases the yield with less compromise of performance. A/S helps contain the clocking problem: Most parts of the chip are designed with a simple clocking scheme (constrained into clock domains), and only limited portions (at module / domain interfaces) need to be more complex. A/S is expected to scale towards high frequencies. It is applicable to both single chips and some MCM systems. The phase detection circuits may also serve as temperature and voltage shift alarms.

A/S implies globally asynchronous, locally synchronous architecture (GALS [14], [16], [17]): An input cannot be expected on any particular cycle. Rather, the cycle of any particular arrival may change between chips and over time. The architecture must provide for this by tagging the data [16], [18].

The main cost of A/S lies in the adaptation circuits. About 1000 transistors are required per Adaptive Synchronizer, resulting in 1% overhead per 1M transistor modules with ten input busses. Time (for training sessions) and power overheads are negligible: A typical training session should converge on all interfaces well within 1,000 clock cycles. Assuming $F_D <$ 1MHz for a 10 GHz locally-clocked chip, the total time overhead is 0.1%. Upper bound on power overhead can be estimated by (Transistor count overhead) x (Time overhead)=10^{-5}.

Acknowledgement

Discussions with Charles Dike and others at Intel are gratefully acknowledged.

References

[1] D.W. Bailey and B.J. Benschneider, "Clocking Design and Analysis for a 600-MHz Alpha Microprocessor," *IEEE J. Solid-State Circuits*, 33(11), pp. 1627-1633, 1998.

[2] E. Friedman, *Clock Distribution Networks in VLSI Circuits and Systems*, New York: IEEE Press, 1995.

[3] P.J. Restle and A. Deutsch, "Designing the Best Clock Distribution Network," *Proc. 1998 Symp. on VLSI Circuits,* pp. 1-5, 1998.

[4] Semiconductor Industry Association. International Technology Roadmap for Semiconductors: 1999 edition. Austin, TX: International SEMATECH, 1999 (http://www.itrs.net/1999_SIA_Roadmap).

[5] D. Sylvester, K. Keutzer, "A Global Wiring Paradigm for Deep Submicron Design," *IEEE Trans. CAD IC&S*, 19(2), pp. 242-252, 2000.

[6] J. Jex and C. Dike, "A fast resolving BiNMOS synchronizer for parallel processor interconnect," *IEEE JSSC* 30(2), pp. 133-139, 1995.

[7] C. Dike and E. Burton, "Miller and Noise Effects in a Synchronizing Flip-Flop," *IEEE Journal of Solid-State Circuits,* **34**(6), pp. 849-855, 1999.

[8] K. K.-Y. Chang et al., "A 2Gb/s asymmetric serial link for high bandwidth packet switches," *Hot Interconnects*, pp. 171-179, August 1997.

[9] J. N. Seizovic, "Pipeline synchronization," *Symp. Adv. Res. Asynchronous Circuits and Systems*, pp. 87-96, 1994.

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

[11] T. Kehl, "Hardware self-tuning and circuit performance monitoring," *ICCD'93*, pp. 188-192, 1993.

[12] G. A. Pratt and J. Nguyen, "Distributed synchronous clocking," *IEEE Trans. Parallel and Distributed Systems*, 6(3), pp. 314-328, 1995.

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

[14] J. Muttersbach, T. Villiger, W. Fichtner, "Practical Design of Globally-Asynchronous Locally-Synchronous Systems," *Proc. 6th IEEE Symp. Adv. Res. Asynchronous Circuits and Systems*, Israel, pp. 52-59, Apr. 2000.

[15] S. Sidiropoulos and M. Horowitz, "A semi-digital DLL with unlimited phase shift capability and 0.8-400MHz operating range," *ISSCC'97*, pp. 332-3, 1997.

[16] R. Kol, Self-timed asynchronous architecture of an advanced general purpose microprocessor, PhD thesis, Elect. Eng., Technion, Israel, 1997.

[17] D. M. Chapiro, Globally asynchronous locally synchronous systems, PhD thesis, Comp. Sci., Stanford Univ., 1984.

[18] Rakefet Kol and Ran Ginosar, "Avid Execution in the Asynchronous Processor Kin," *3rd Euromicro Conference on Massively Parallel Computing Systems (MPCS'98)*, Apr. 1998.