DESIGNLINES | PROGRAMMABLE LOGIC DESIGNLINE < https://www.eetimes.com/designline/programmable-logic-designline/>

System synchronization styles and trends

By Sandeep Srinivasan and Eby G. Friedman 03.06.2006 0

This article describes emerging trends in synchronizing digital ICs and shows how process scaling, rapid increases in clock frequencies, and demand for lower power dissipation will affect the choice of synchronization styles going forward.

System synchronization controls the flow of events in a system. In the same manner that all signals are, in reality, analog in nature, all timing is in reality asynchronous. Despite this characteristic, local timing constraints can be placed on a system to permit the system to behave as if the system is completely synchronous.

This strategy uses a central clock signal to control the relative timing of events and is called *synchronous clocking.* Fully synchronous clocking makes it easier to understand the temporal behavior of events in a hardware system with reference to a clock edge.

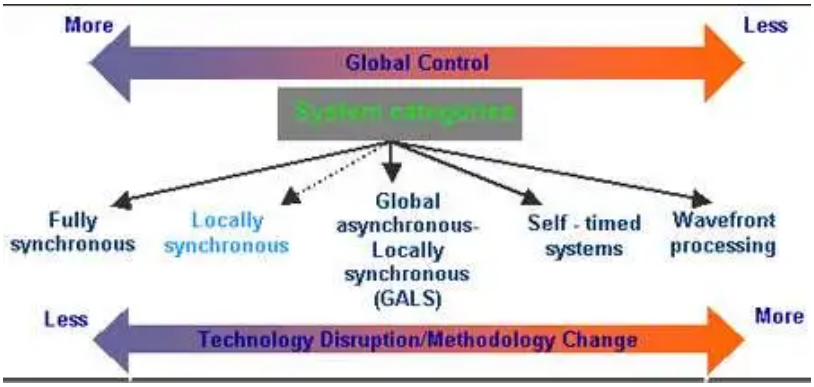


Figure 1 System synchronization styles

Moving one level down from the system into a digital IC, the different synchronization methodologies can be categorized in terms of the level of global control.

Fully synchronous

Globally asynchronous locally synchronous (GALS)

GALS based systems provide local synchronization by introducing an asynchronous handshaking protocol among elements at the system level, where these individual elements often operate with different clock domains, possibly at different frequencies. An interesting example of GALS at the system level is the communication between an interrupt controller and the central processing unit (CPU) on a PC motherboard. The CPU and the interrupt controller both operate at different clock frequencies and communicate through a predetermined protocol.

Extending this concept to different functional units on a system on chip (SoC) gives rise to a GALS based system, where the local clocks operate on functional units and synchronize with other local clock domains using various locking protocols. There has been a great deal of progress in the development of GALS architectures, but EDA tools and design practices remain at a very nascent state. Significant research is currently underway to develop methodologies for integrating GALS into mainstream microprocessors and SoC systems.

Self-timed systems (asynchronous)

Self-timed systems utilize handshaking protocols to communicate between separate functional elements. Each element ensures that a data packet has been successfully received, transmitting back to the sender the successful receipt of the data. This dual communication requires significant power and area overhead but is highly amenable to system scaling.

The issue of single-rail communication versus dual-rail communication has a significant effect on overall system performance characteristics. Testability also remains a primary issue in the successful implementation of self-timed systems; however, the local use of these protocols within GALS systems has attracted some interest from industry.

Wavefront processing

Consider a self-timed system without area and power penalties. Conceptually, toss a pebble into a lake and imagine the waves propagating from the place where the stone hits the water. As long as no communication exists between any of the wave fronts, the signals will propagate smoothly to the end source.

This synchronization architecture is the essence of wavefront processing[2] . The advantage of this simple synchronization methodology is significantly less hardware overhead. The disadvantage of this approach is a highly constrained data flow, which can only be applied to highly specific, forward flowing architectures. This synchronization architecture is therefore most frequently applied to specialized topologies such as arrays.

Overhead on synchronous systems

As clock cycle times decrease (and frequency increases), the timing overhead required for a fully synchronous system increases as a percentage of the overall clock cycle time. The amount of useful work that can be accomplished within one clock cycle (T_{logic}) in a synchronous system can be described by the following relationship:

$$T_{logic} = T_{cycle} - (T_{Cq} + T_{setup} \pm T_{skew})$$ ← *Useful work per cycle*

T_{cycle} = Clock Cycle Time
T_{cq} = Clock to Q (Output) delay of a Flip-Flop
T_{setup} = Setup time of a Flip-Flop
T_{skew} = Clock Skew between Launching and Receiving Register

The timing penalty incurred in a synchronous system due to the sequential elements (the registers and latches) is termed the “sequential overhead.” The sequential overhead can be described by the following relationship:

$$T_{seq} = T_{Cq} + T_{setup} \pm T_{skew}$$

Delay uncertainty can be treated as a portion (or budget) of the total clock period. Clock distribution networks typically have the largest signal delay, and are therefore prone to the greatest amount of delay uncertainty. The delay of the clock network (or clock insertion delay T_{clock}) on large systems-on-chip (SOC) is typically a multiple of several clock cycle times. Delay uncertainty on combinational logic (T_{logic}) is typically much smaller as compared to delay uncertainty within the clock network, since the delay of the combinational logic is required to be less than a clock cycle time.

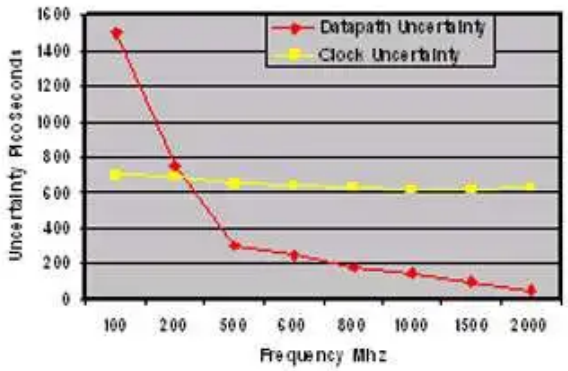


Figure 2 Delay uncertainty on logic paths and clock distribution network

$T_{clock} > T_{cycle}$: Typically $T_{clock} = n^* T_{cycle}$: where $n>1$
$T_{logic} < T_{cycle}$
$T_{clock} >> T_{logic}$
$T_{uncertainty}$ **(clock)** $= n^* T_{clock}$; $0<n<1$
$T_{uncertainty}$ **(logic)** $= n^* T_{logic}$; $0<n<1$

In Figure 2, the delay uncertainty of a logic path ($T_{uncertainty}$ ***(logic)***) is shown to decrease as frequency increases, while the delay uncertainty within the clock network ($T_{uncertainty}$ ***(clock)***) is independent of frequency and is a function of the clock network delay (T_{clock}).

Minimizing the delay uncertainty of the clock distribution network can dramatically reduced on-chip delay variance.

$$T_{uncertainty}(clock) >> T_{uncertainty}(logic)$$

Including delay uncertainty, the sequential overhead can be re-written as:

$$T_{seq} = T_{Cq} + T_{setup} \pm T_{skew} \pm T_{uncertainty}(clock)$$

And the amount of time for useful work within a single clock cycle can be rewritten as:

$$T_{logic} = T_{cycle} - T_{seq}$$

Decreasing the cycle time reduces the amount of available time to accomplish useful work (T_{logic}) per clock cycle, since the sequential overhead becomes a greater portion of the overall clock cycle period.

The growth in sequential overhead with increasing clock frequency in synchronous systems for a 130 nm process is illustrated in the figure below.

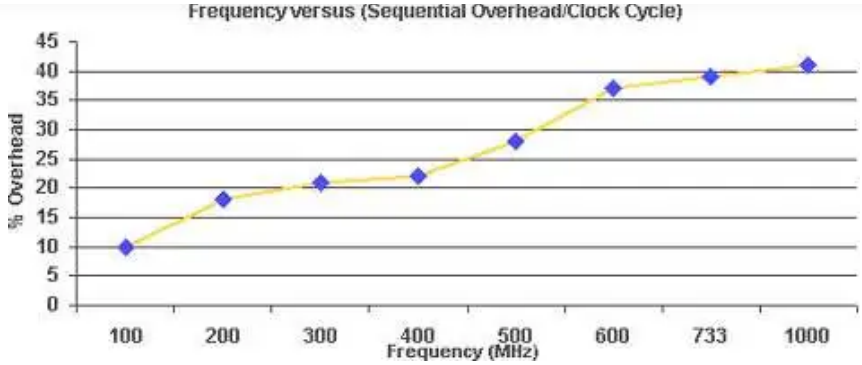


Figure 3 Sequential overhead of ASIC designs, 130 nm CMOS process

Note that the sequential overhead is becoming a larger portion of the total clock cycle time, leaving less time within each clock cycle to accomplish functional work. For example, at 600 Mhz, the sequential overhead can be as large as 33% on application-specific ICs (ASIC) using state-of-the-art EDA tools.

Large variations in the arrival time of the clock signal require increased design margins or guard bands. These design margins have become unattainable with rapid increases in clock frequencies and power dissipation. Managing skew and delay uncertainty globally across large SOC's has therefore become a fundamental challenge.

An important solution for mitigating and managing the problem of sequential overhead is the proper application of locally synchronous techniques.

Locally synchronous

Local synchronization[1] is a method for exploiting the natural data flow and imbalance of logic stages in a synchronous system by using zero skew full synchronization only where needed and appropriate.

Local synchronization can be realized on a fully synchronous system, by spatial and temporal grouping of *sequentially-adjacent* (those state elements that are directly connected through combinational logic) state elements that have similar or the same "local clock skew" requirements. This system of grouping is analogous to creating individual clock domains for different on-chip functional units.

Local synchronization implies that clock skew is only relevant between sequentially-adjacent state elements. An important concept is that since clock skew is only relevant in sequentially-adjacent registers, clock skew between non-sequentially adjacent registers has no physical meaning.

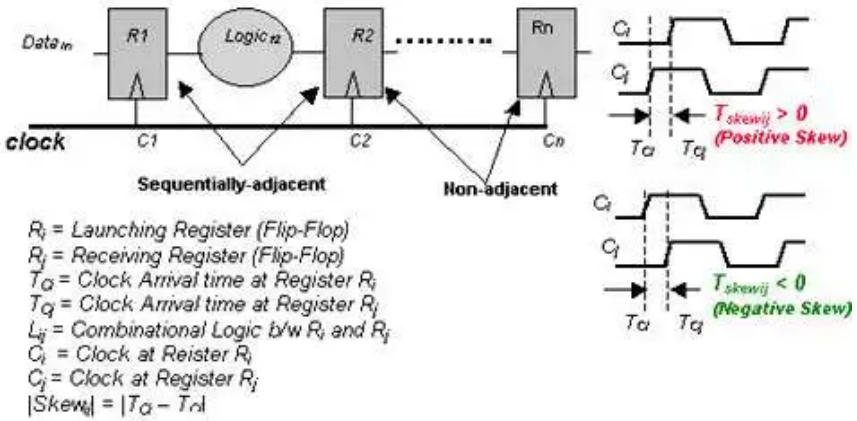


Figure 4 Sequential adjacency and definition of clock skew

The objective of a locally synchronous system is to create locally synchronous regions. These regions contain a grouping of sequential elements that may have similar local skew requirements. These regions are determined, permitting the skew requirements among these regions to be relaxed.

Aggressive global skew constraints can lead to over design and place unrealistic design objectives on logic synthesis, placement, optimization, and clock synthesis tools. Relaxing the global skew target for an integrated circuit has significant advantages and implications on the power, performance, and tolerance to process variations of a high performance integrated circuit.

Note that local synchronization is a natural extension of fully synchronous systems.

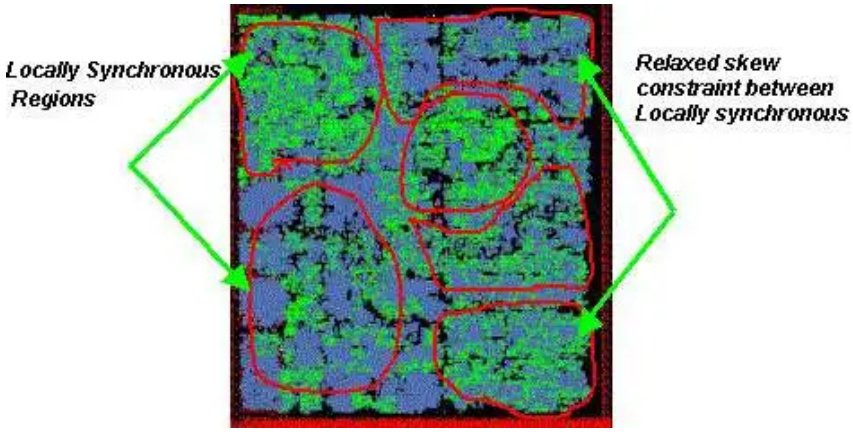


Figure 5 Locally synchronous regions on an IC

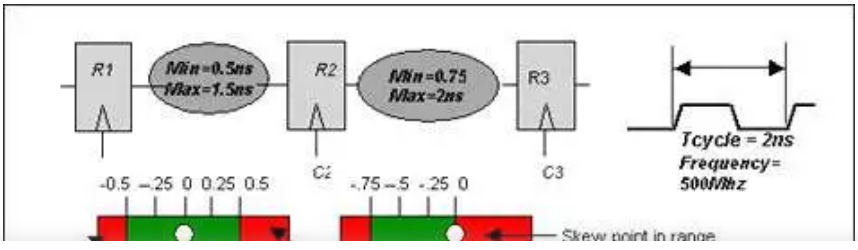
Implementing local synchronization

Local synchronization can be implemented in the architectural planning stage, as well as in the physical implementation step of the IC design process. Local synchronization is realized on a fully synchronous system by creating locally synchronous regions based on a *clock skew schedule.*

Clock skews are local in nature and specific to each local logic path. The set of “local clock skews” for every individual local logic path is called a “clock skew schedule[3] .” The clock skew schedule provides the range of time that each state element (such as register, flip-flop) within a system can receive a clock signal.

Every synchronous system has an optimal range of clock skews between each sequentially-adjacent pair of registers. This range of skew between sequentially-adjacent registers is called a “permissible range of clock skew[3] .” If the skew is maintained within the permissible range for all sequentially-adjacent registers, no setup or hold violations will occur (no frequency limitations or race conditions will be produced).

The permissible skew ranges for two sequentially-adjacent register pairs and the zero-skew point within the range is shown in Figure 6. Local synchronization can be exploited on large SoCs by spatially minimizing local skew and allowing for greater tolerance to global skew.



the local on-chip skew regions, while maximizing the global skew requirements among those regions.

All synchronous circuits exhibit some locally synchronous regions due to efficient architectural partitioning and the natural data flow of the system. Circuits that are properly architected will exhibit a natural tendency for locally synchronous regions. For example, a typical microprocessor has a well developed floorplan and synchronization methodology. These circuits tend to have locally synchronous regions that overlap both spatially and temporally.

By exploiting clock skew scheduling, enhanced insight as to how best to plan the temporal characteristics of an SoC is achieved. This insight provides the means for determining locally synchronous on-chip regions.

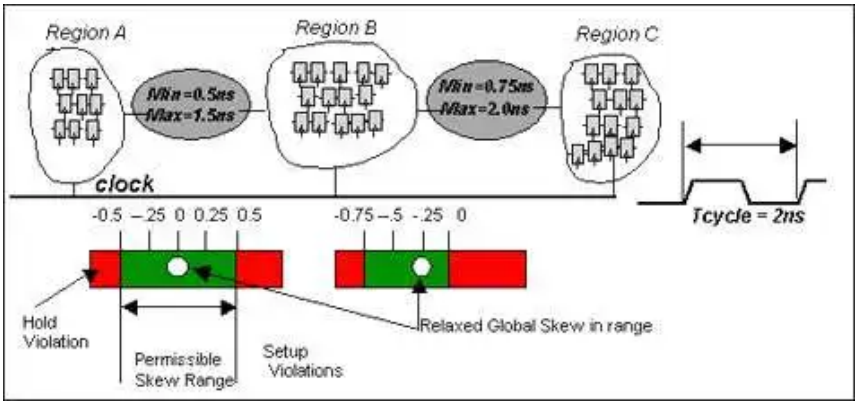


Figure 7 Relaxing global skew constraints among locally synchronous regions

Locally synchronous regions can be implemented by combining placement, partitioning, and clock skew scheduling during the physical implementation of a circuit.

Locally synchronous regions can also be realized and planned for at the architectural planning stage. For instance, in the design of a microprocessor, an issue is whether all of the registers in the Bus Interface Unit (BIU) need to switch or remain active at the same time as the registers in the Arithmetic Unit (ALU).

These units may need to communicate with each other within the same clock cycle, but perhaps not simultaneously within the current cycle. This characteristic leads to a notion of the BIU receiving the clock at a slightly different time than the ALU.

Benefits of local synchronization

Locally synchronous regions can reduce power, improve performance, and enhance the fidelity (or robustness) of the circuit design. Specifically, local synchronization accomplishes the following:

- Lowers simultaneous switching noise and peak currents. The clocks driving different regions on the IC switch at different times, reducing the switching noise.
- Requires fewer number of buffers within the clock network. Due to relaxed global skew requirements, significantly fewer buffers are required to implement an optimal clock tree.
- Reduces dynamic power consumption due to reduced capacitance in locally synchronous regions.

- Reduces leakage current. Local synchronization uncovers opportunities for downsizing or replacing non-critical combinational cells with low-leakage cells.

Extending the life of synchronous design styles

Distributing the clock signal with precise timing to millions of on-chip elements has become one of the most challenging problems in digital IC's on process nodes below 130 nm. It is now imperative that an effective clocking strategy be properly applied before physical design and implementation.

With geometries below 130 nm, managing sequential overhead on a high performance circuit design has become a primary design issue. Locally synchronous design methodologies can be used to manage this overhead while enhancing performance and reliability, greatly extending the longevity of fully synchronous design methodologies.

Summary

The future of fully synchronous design styles is in jeopardy due to increasing chip size, rising clock frequencies, and greater sequential overhead. While GALS and asynchronous systems are gaining momentum within the research community, their use in industrial applications remains rare and it may be a long time before these methods become a viable alternative for mainstream digital ICs. A primary reason for this trend is the lack of EDA tools and methodologies to support these design styles.

Asynchronous design methodologies appear attractive since these methodologies promise to remove challenging issues such as the effective design of the clock distribution network. However, asynchronous methodologies also produce additional challenges such as functional verification, increased area and power, complex testability, and managing global enable signals. GALS and asynchronous design styles may become more accepted in the future, but over the next decade there is significant economic interest in protecting and extending the life of synchronous design methodologies.

The application of locally synchronous techniques is an important design technology that will significantly extend the life of synchronous design methodologies. Clock skew scheduling provides a systematic and optimal approach to synchronization for fully synchronous designs.

Merging clock skew scheduling techniques with placement and floor-planning is necessary to realize local synchronization. This merging of technologies leads to a new class of tools clock planning or clock-centric floor-planning.

It is no longer sufficient to insert or retrofit a clock tree after logic synthesis and placement into a back-end design on advanced process nodes.
Clock planning and clock-centric floor-planning will be required on all large SoC's designed in advanced process technologies to converge on timing, power, and reliability.

References

[1] E. G. Friedman, "Clock Distribution Networks in Synchronous Digital Integrated Circuits" Proceedings of the IEEE, Vol. 89, No. 5, pp. 665-692, May 2001.

[2] S. Y. Kung, VLSI Array Processors, Prentice Hall, 1988.

[3] I. S. Kourtev and E. G. Friedman, Timing Optimization Through Clock Skew Scheduling, Kluwer Academic Publishers, 2000.

Sandeep Srinivasan is Founder & CEO of ***Synchronous Design Automation. < http://www.synchronous-da.com>***

on high performance circuits and clock distribution networks. His current research and teaching interests are in high performance synchronous digital and mixed-signal microelectronic design and analysis with application to high-speed portable processors and low power wireless communications.

RELATED ARTICLES

EDN **System synchronization styles and trends < https://www.edn.com/system-synchronization-styles-and-trends/?**

<

https://www.edn.com/system-

By EDN 06.03.06

synchronization-

styles-

EDN **On-chip nets look to rewire next-gen ICs < https://www.edn.com/on-chip-nets-look-to-rewire-next-gen-ics/?**

<

https://www.edn.com/on-

By Richard Goering 26.12.05

chip-nets-

look-to-