

Practical GALS for Multicore SoCs

December 19, 2007
Steve Hamilton, Application Architect
Sonics, Inc.

1 Introduction

As multi-core SoCs continue to evolve, their interconnect architectures have become the major design challenge. Implementing Quality of Service (QoS) in order to share access to tightly-coupled memories; coordinating error detection and handle protocols; debugging data flow for concurrent processors; as well as fundamental challenges such as SoC timing closure are all effected by the level of interoperability the interconnect provides between IP cores and external memory..

New generations of multi-core SoCs are now adding asynchronicity to their design as operating frequencies soar and power consumption emerges as a top design priority. The rise in power and voltage domain isolation requirements increases the need to extend the functionality of advanced interconnects, such as Sonics SMART Interconnect solutions that currently decouple IP core interoperability through packet-based communications techniques, to expand their decoupling and include clock and voltage domain isolation to IP core boundaries as well. Expanding the levels of isolation is another step that Sonics has taken in the practical application of Globally Asynchronous Locally Synchronous design – so called GALS - to SoC designs.

A practical approach to the application of GALS is an important design consideration because the insertion of asynchronous components into an SoC design must also come without the disruption of design flows and without breaking modeling tools. SonicsExpress, Sonics' high performance asynchronous bridge, enables SoC developers to extend the interconnect architecture, while keeping a balance with the fundamental design flows required to complete the design.

This report looks at the practical extension of interconnect architectures to include more of the GALS system style. It first examines the various technological approaches that might be used to implement a GALS system and weighs the costs and benefits of each. The requirements for high performance asynchronous bridging to serve GALS are next examined. Finally, a brief survey of the overall requirements for GALS systems is taken, which is then matched with the features contained in the Sonics SMART Interconnect solutions.

Motivations for GALS designs

Based upon the recent volume of discussion of GALS in academic and conference materials as well as in trade press, some may think GALS is a new idea. However, GALS implementations have been present since the beginning of the SoC era. There are many examples, such as the NSC8000 chip released in 1997 by National Semiconductor. While it is clear that GALS is not a new design concept, it was never broadly adopted. So why the recent renaissance? There are two primary forces driving the renewed interest in GALS design: timing closure and power management.

Timing Closure

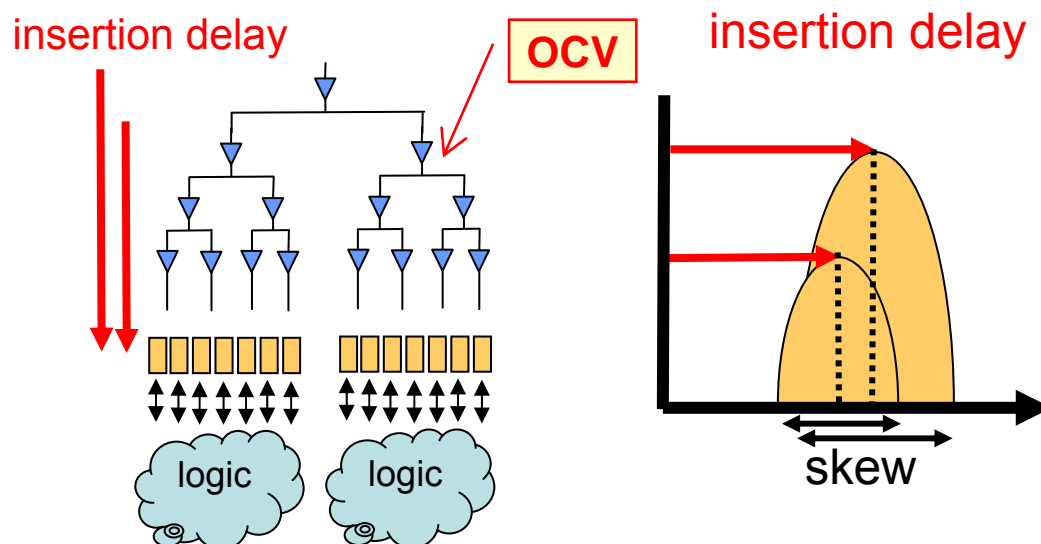
Clock distribution costs are scaling up, not down. Consider a design that exists today in a given process node. It has a number of logic gates and a number of flip flops. The flip flops are clocked from a single clock via a clock distribution tree. This tree is created by a back-end process as a balanced tree, so that the insertion delay from the clock input pin to any single flip flop is the same. The process, of course, is not perfect, so we have some distribution on insertion delay across the population of flip flops. The width of this distribution is commonly known as *clock skew*.

As we move to the next process node, we historically get 70% scaling in both linear dimensions. This means we get a doubling of logic density - twice as many gates in the same area. If we have twice as many gates, we also have twice as many flip flops.

The same clock tree generation process can be run, but the tree it produces needs roughly the same rate of fanout. The tree will be deeper in order to span twice as many leaves. Therefore, the insertion delay increases. Worse than this, the spread on the larger distribution cannot be controlled to the same absolute value. This absolute value is, to a certain extent, dependent upon the depth of the tree. Thus, clock skew will be greater in the denser process node.

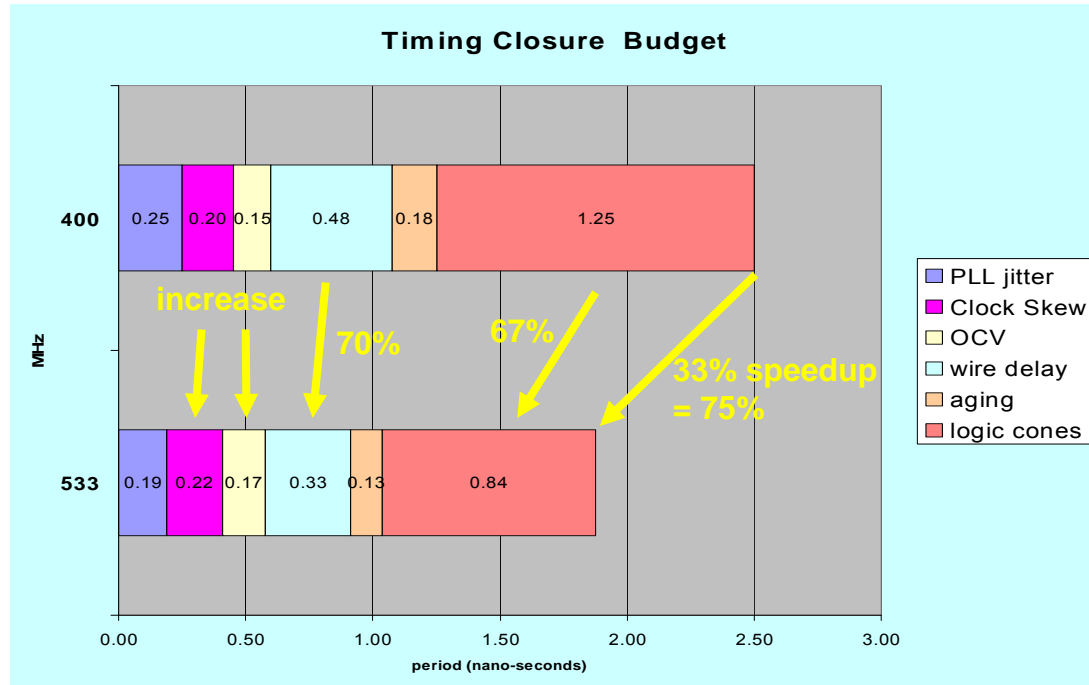
There is a second effect called *On Chip Variation*. This is a local and not very predictable variation in the manufacturing process. It manifests as variation in signal propagation rates. This includes the clock signal. The percentage variation gets bigger with each process generation and as the clock tree is deeper, these variations accumulate to a greater value.

Figure 1 - Clock Skew Increasing



Increasing skew and OCV lead to the squeeze depicted in Figure 2. Basically, increasing clock distribution budgets don't blend very well with targets that are increasing in frequency

Figure 2 - Timing Budget Squeeze



Suppose we have a 400 Mhz design in the current process node, and in the next process node we want to speed that up by 33% to run at 533 Mhz. Is it reasonable to expect a 33% speedup? That means the clock period must scale to 75% of its current value. We can examine the constituent parts of our clock period budget to see how they scale.

PLL jitter should scale nicely as a fixed percentage as long as an analog PLL is used.

However, the clock skew and OCV are actually increasing rather than scaling. It turns out that the total amount (in ps) of clock uncertainty (PLL jitter plus clock distribution) is roughly unchanged.

Wiring lengths should reduce with the 70% scaling factor. Unfortunately, capacitance per unit length is actually increasing, as is resistance because of the thinner wires. So the RC delay per unit length is increasing. This will make it challenging to scale the wiring delays by the 70% factor. If the wires are not too dominant, we might squeak by at least close to the overall 75% scaling target.

Scaling the logic delays (and the aging budget associated with them) is even more difficult. Here we need a 50% speedup to make our budget – which calls for a 67% scaling factor. Unfortunately, the setup and hold times for flip flops are hardly scaling at all now. Also, as oxide thicknesses scale, VDD must be lowered to avoid punch through. The result is that drive current only increases slightly. A 50% speed up in gate delay is not likely and a greater speedup than that may be needed.

We use an aging budget because over time ions implanted in the silicon during the manufacturing process will migrate. This will lower transistor currents and slow down the logic. As dimensions shrink, the same amount of migration has a bigger percentage increase. Therefore, it is likely that a greater percentage of logic delay should be reserved for aging margin, which would squeeze the

synthesis timing allocations even more. So, unless next year's synthesis tool is able to turn the same RTL into shorter path lengths than last year's tool did, we will be hard pressed to make our budget.

One promise of GALS is to allow the clock skew and OCV budgets to scale with everything else. By breaking the logic into smaller local blocks, the size of the clock trees stay reasonable, and scale with process. This then makes a greater portion of the budget available for logic. All we need to do in order to have this benefit is to deal with the absence of phase lock where signals cross from one block to another.

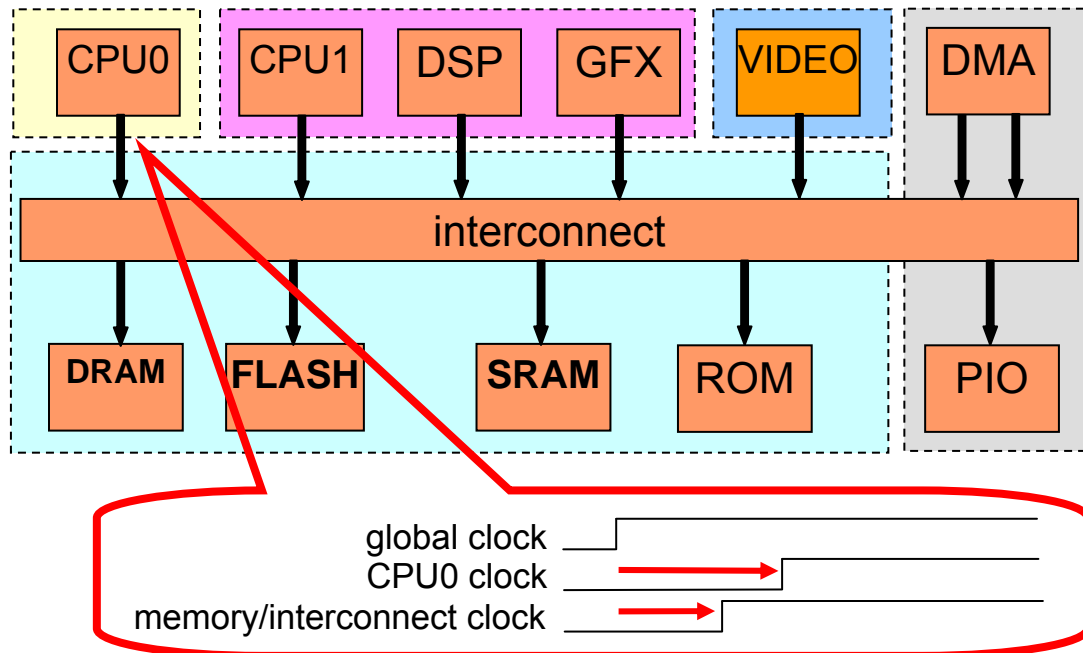
Power Management

Active power in CMOS devices are largely determined by switching power, which is modeled as $P = CV^2f$. This equation shows active power consumption which grows linearly with frequency increases. If we hold voltage constant (because voltage scaling has at least slowed, if not temporarily stopped), as logic densities double, the power density is increasing at twice the rate of frequency increase. In general then, the increased density is exploited by putting more logic and function on the SoC, rather than by reducing the die size. Generally the die sizes are remaining the same from process node to process node. This means that the total active power of the SoCs is increasing – at roughly twice the rate of frequency increase.

Meanwhile, the ability to dissipate that power is actually decreasing. This is driven by mechanical design. Typical form factors are becoming smaller, and in particular lower in profile, while products are becoming more mobile. The end results are less surface area through which to dissipate power, less room for heat sinks, and lesser reliance upon fans.

The balance between heat generated and heat dissipated must be maintained. The only solution is to explicitly control how much of the chip produces heat. This requires aggressive activity management within the SoC. Because leakage power is increasing so much, activity management can no longer be implemented primarily through clock gating. Increasingly, it must be implemented via voltage management. This means multiple voltage domains within the SoC with dynamic management of the voltages on those domains. The dynamic management extends to include the 0 volts point allowing whole sections of the SoC to be powered off, while other sections operate simultaneously at reduced voltage or full voltage levels.

Figure 3 - Power Managed SoC



An SoC with aggressive power management may look something like that shown in Figure 3 with a number of separate voltage domains.

Of course logic speed is a function of supply voltage. As voltage is reduced in one of these regions, the logic in the region will slow down. We already looked at how difficult timing closure is becoming. When voltage is reduced, it becomes necessary to lower the operating frequency on the region. This can be done in either “open loop” fashion, where certain pre-determined voltage-frequency steps are designed in, or in “closed loop” fashion, where either voltage or frequency is set and a control loop optimizes the other variable. Open loop control is typically less optimal, since designs must allow for worst case variations.

If either open loop control or closed loop control is used with frequency as the dominant parameter, the frequency steps can be restricted to whole divides of a single root frequency. This allows the interfaces – where the data paths cross voltage domain boundaries - to run in divided synchronous fashion. It is cost effective to operate the whole SoC with a single PLL, as divided synchronous clocking still preserves the fundamental globally synchronous nature.

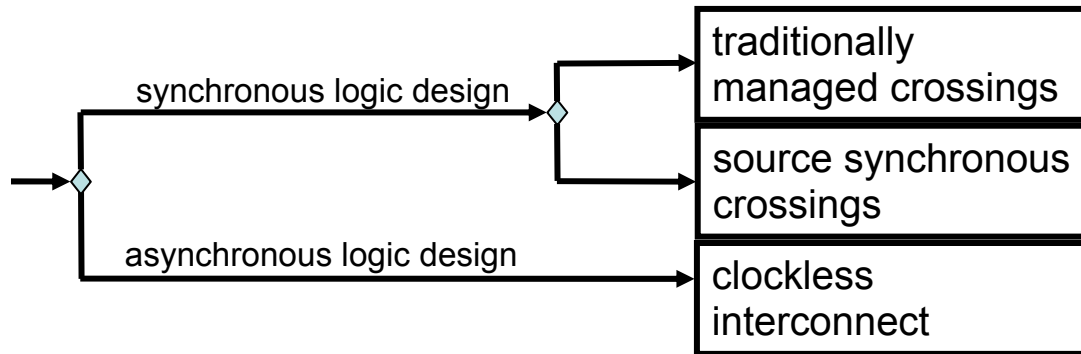
However, this doesn't work very well. Even if the frequency is lowered, there is still a phase lock requirement on any synchronous interface. In this case, in order to maintain that phase lock, the clock tree synthesis process must balance the insertion delay across two separate voltage domains. This is very hard. In addition, if the domain voltages vary – rather than just switch on and off – the level shifters are required at the boundary. The propagation delay through those level shifters will vary as a function of the voltage – typically to a much greater extent than delay varies for other logic. – This will occur in addition to delay variation in all logic. This means the clock insertion delay will vary as a function of voltage, resulting in a phase lock that is just too hard to maintain.

This shows the point of asynchronicity naturally wants to occur at the voltage domain boundary. Even if we maintain frequency lock between these domains, we must abandon phase lock. If we do, we can vary the domain voltages at will, and must only manage clock skews within the domains.

GALS Implementation Approaches

We should now understand how the technology is driving us to organize our designs more as islands of synchronicity in an asynchronous world, and less as globally synchronous. In addition, we see the bridges between these islands should also span voltage to facilitate power management. By keeping the islands a reasonable size, margins are managed to facilitate timing closure.

Figure 4 - GALS Approaches



As shown in Figure 4, there are currently 3 basic approaches to implement GALS designs.

Traditional synchronous logic design style using asynchronous crossing techniques is the most well proven approach. However, it has some shortcomings. Signals that cross the boundary must be re-synchronized to the receiving clock domain using double or triple rank synchronizers. This adds considerable latency. It is especially costly for flow control loops that span the boundary since signals in both directions are delayed. In order to maintain data rates in the presence of these long flow control loops, asynchronous FIFO crossing buffers are used for payloads. The synchronizers and crossing buffers add significant gate count. Because the flow control loops are long, the crossing buffers can fill up and block the path. To manage blocking among parallel activities, each activity needs a parallel path. This increases the number of lines that cross the boundary, requiring isolation buffers, level shifters and crossing buffers. The added wires and gate count consumes more area; thus, there is some penalty to be paid when using this approach for the asynchronicity.

Source synchronous clocking is another approach to GALS. While this still uses traditional synchronous logic design style, it attempts to minimize the latency cost of the boundary crossings. It is most attractive if global frequency lock is maintained while allowing un-restricted clock phasing. Since this doesn't fit our GALS environment well, this approach is only discussed briefly.

The third approach is clockless design. Typically it is the interconnect only that is clockless. The claim is that by applying an asynchronous logic design style to the interconnect, latencies associated with crossings can be eliminated or minimized without resorting to crossing buffers. This of course offers lower gate counts. In addition, since the interconnect is asynchronous, it idles automatically when no traffic is present and therefore saves power. Let's look at these claims in greater depth.

Clockless Interconnect

Gate Count

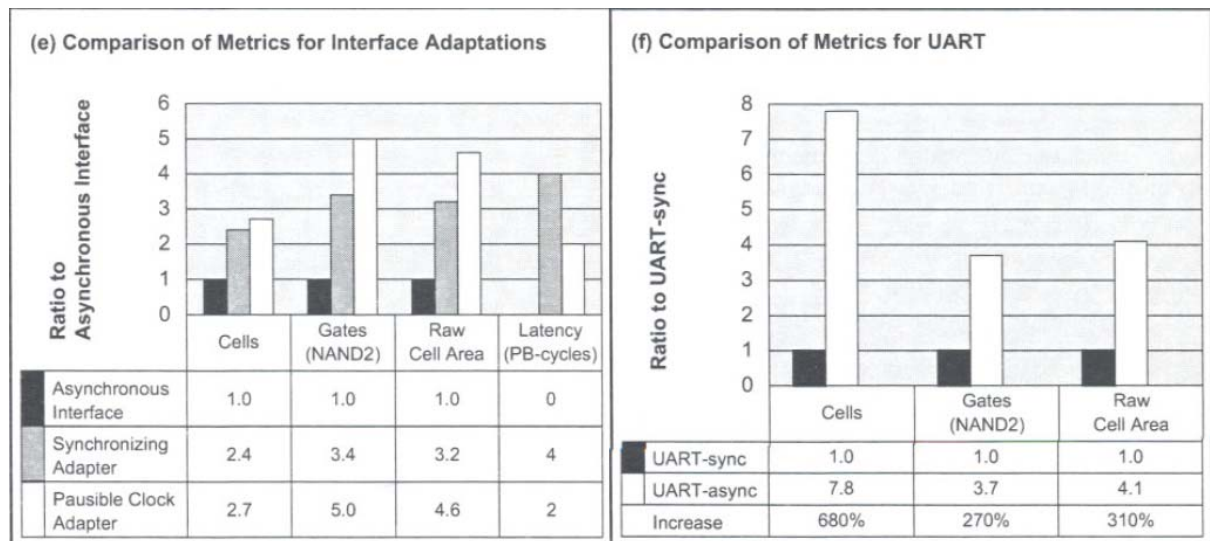
There are a lot of cores that exist today, and it is desirable that they be reusable in the future. Thus, it is not realistic to expect a wholesale shift to asynchronous logic design style. Instead, it is generally

expected that just the interconnect will be clockless. Unfortunately, an adapter is then required for each core to bridge the synchronous interfaces into the world of asynchronous handshakes and synchronize the asynchronous world back into the synchronous.

These adapters represent additional logic. They also bring with them latency associated with the re-synchronization. Figure 5 is taken from a paper co-authored by Intel on the subject. Note that in the raw cell area for (e) the synchronizing adapter is 3.2 times bigger than if the core also had a clockless interface. It also adds 4 latency cycles to each transfer. The latency can be reduced by using special clock pausing techniques, but these come at additional gate cost. Note in (f) the clockless adapted UART is 310% bigger than the un-adapted UART.

The conclusion drawn is that these adapters represent significant area overhead and introduce non-trivial latency. The paper also describes the additional engineering needed to migrate the core testbenches, and that the very likely occurrence that tightly coupled testbenches will report failures in the face of this added latency and require testbench modifications.

Figure 5 - Cost of Adapters for Clockless Interconnect

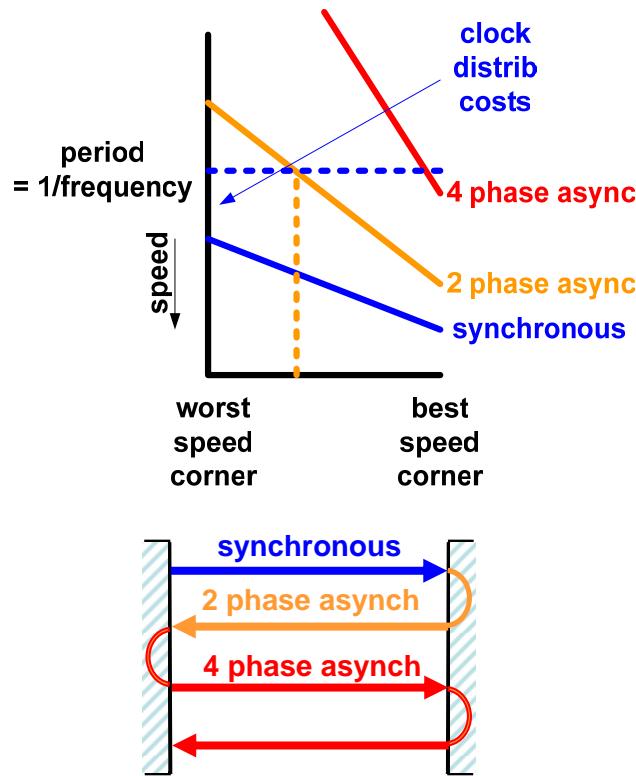


However, this just scratches the surface regarding the other problems associated with intermixing synchronous and clockless design styles.

Predictability

Automatic adaptability, which is the great strength of clockless logic, is also one of its greatest weaknesses. Fundamentally, clockless design lacks predictability. This shows up both at the architectural planning level and at the implementation level associated with manufacturing test.

Figure 6 - Logic Speed and Predictability



You may occasionally hear the claim that asynchronous design style logic is faster than synchronous logic. In fact the opposite is true. The two phase or four phase handshakes used in clockless logic make it inherently slower – by a factor of two or more. This becomes more evident as distance increases because the handshake signals take longer to propagate. As shown in Figure 6, the handshake must only cross once (blue) from sender to receiver in synchronous design. With clockless design, if a two phase handshake is used (orange), then the handshake requires a round trip from sender to receiver and back. If a four phase handshake is used, then two round trips are required. Signal propagation is increasingly dominating logic delay in advanced processes, this means this handshake distance is a reasonable indicator of the time needed for a transfer.

What is true is that clockless logic runs as fast as the particular silicon permits – so-called best case operation – while synchronous logic is typically run as fast as the worst silicon permits – so called worst case operation. This worst case must include a budget for clock distribution, which is not needed in clockless design. The figure shows the handshake paths mapped onto the normal CMOS process distribution (typically six sigma yields roughly a 3:1 speed difference from best to worst case). The figure shows the product speed for the synchronous design is set by the worst case corner, after adding the clock distribution delays. Thus, all copies of a synchronous design operate the same, while each copy of an asynchronous design operates differently. This can result in a portion of the manufacturing distribution where the clockless parts are faster than synchronous parts - shown by the orange dotted line for the two phase handshake approach. However, this variability in the population is a design and manufacturing problem.

The asynchronous handshakes need not be time bound in order to be properly functional. Constraining them is difficult. However, it may not be until AFTER place and route that system level performance problems surface. Such late discoveries can be very expensive! The Intel-Silistix paper reported such an event. A core connection was serialized over a narrow link because “asynchronous logic is so fast,” and it was discovered in post-route analysis that this path was limiting system level performance.

This same problem also significantly complicates ATPG; in fact, it complicates any scan based testing. Obviously, the state elements associated with asynchronous logic design – such as Mueller elements and latches – can be bypassed or mode switched to reduce the clockless interconnect to a combinatorial function. It can be surrounded with a scan ring to control and observe I/Os, respectively. But at what frequency can it be tested? Are there timing faults in such logic? If not, then how can system performance be assured? If so, then how does testing expose them?

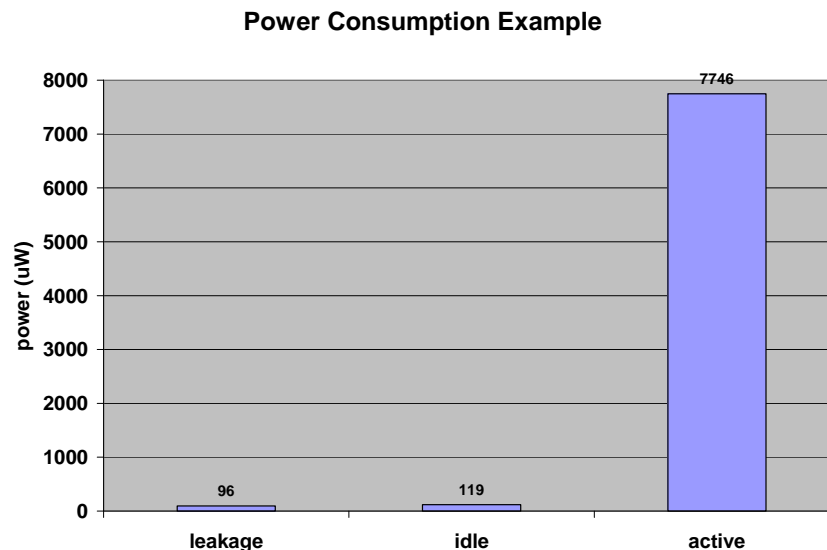
Power Consumption

One of the most often claimed advantages of clockless interconnect is reduction of power consumption. While this is true, it's a fair question to ask how big an advantage this is. Clock gating in synchronous design has been used for years. Very aggressive use of clock gating is possible. How much difference is there between a clockless interconnect and an aggressively clock gated interconnect?

Sonics SonicsMX is such an interconnect. It explicitly codes clock gating, rather than relying upon the synthesis process to find and exploit clock gating opportunities. It creates very fine grained gated domains so that active power is limited to only that needed by the switching activity. In addition, it includes a second level of gating higher up the clock tree to absolutely minimize the number of ungated flip-flops. The course level gating is auto-enabled including wakeup logic that monitors the inputs for newly arriving activity.

The graph below shows the power consumption of one configured instance. This is a substantial interconnect with roughly a dozen high performance initiators and approximately 10 high performance targets. It is a real customer design example, and the numbers reported are from the customer. The constituent parts of the power consumption are separated in the graph. Leakage power is consumed whenever supply voltage is present. Idle power is consumed whenever clocks are running. Active power is consumed whenever traffic is flowing through the interconnect. The customer supplied the realistic set of stimulus to model a video decode and display activity.

Figure 7 - Aggressive Clock Gating



You'll note that idle power is quite low compared to active power. As a matter of fact, idle power is roughly the same size as the leakage power. If the use profile for the device is dominated by idle mode, as it is for most handheld devices these days, then the leakage number is the one that matters. If that number is too high for the requirements, then voltage switching must be used. This is the common case today, where the leakage number alone is too high. And, in this case, there is little

advantage to a clockless interconnect. Both clockless and clock gated interconnects have good active numbers to minimize heat, and leakage numbers requiring voltage switching in order to preserve battery life. A small multiple of leakage when clocked but idle is acceptable.

Conclusion on Clockless

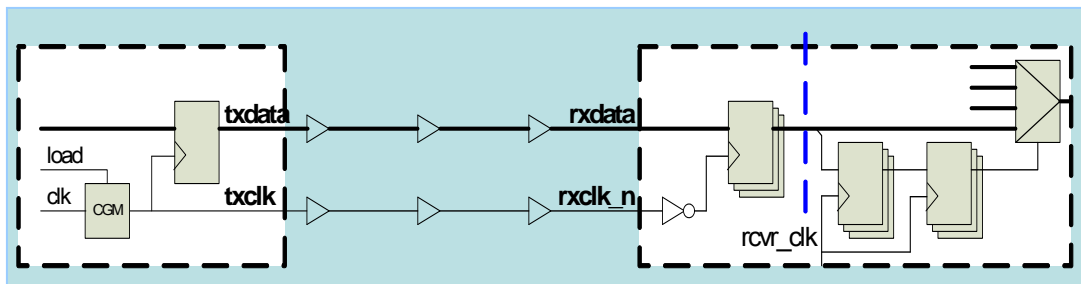
Currently, none of the problems with the use of clockless interconnect are insurmountable. In time, techniques will be devised and tools will become available, and the mainstream design culture will be expanded to comfortably include this option. However, the Sonics position is that clockless interconnect is not yet ready for prime time. Any area reductions resulting from elimination of synchronous registers in the design are offset by the need for adapters and scan based test structures. The latency of the adapters is roughly the same as the latency of traditional management of asynchronous handshakes. Speed is actually reduced. In addition, unless clever techniques are invented for characterization based binning, this is across the board on all parts. Power consumption is likely only marginally better than achievable with aggressive clock gating. Collectively all of this is not a strong enough case to justify the extra risks and problems created in the design flow.

It should also be noted that at present the costs associated with mixing design styles is too high for this to be an attractive approach. Clockless interconnect is not the best choice for GALS design.

Source Synchronous

The second GALS approach is Source Synchronous clocking. It will be discussed only briefly.

Figure 8 - Source Synchronous Clocking



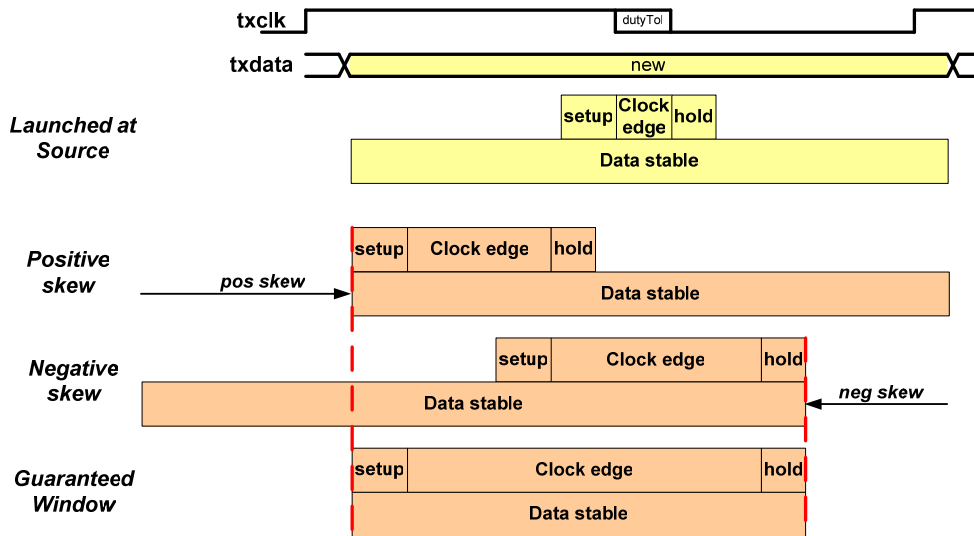
Source Synchronous clocking still requires a copy of the sender's clock to exist in the receiver's domain. But it minimizes the use of that clock, and it permits those uses to be excluded from skew management for the clock. As the little logic diagram shows, the sender uses the clock to produce the data, then sends the clock in parallel with the data. The receiver inverts the clock, then uses it to capture the incoming data into a register or small FIFO. Today, if the sender and receiver are frequency locked, then this register can be cheaply phase adjusted for consumption in the receiver domain by something like a latch. If they are not frequency locked, then the captured data still needs to be re-synchronized into the receiver's domain (as shown above).

A big problem with source synchronous clocking is that the clock and data can skew relative to one another in either direction. This means the design window must be large enough to work in either case. This tends to put limitations on the frequencies obtainable with this technique. Unfortunately, the ASIC tool chain has relatively few control knobs available to force skew in either direction. .

Remember also that it will often occur that the sender and receiver are in different voltage domains. The transmission path is not as simple as the one shown here; it will include isolation buffers and level shifters on both the clock and the data paths. Level shifters in particular are sensitive to input

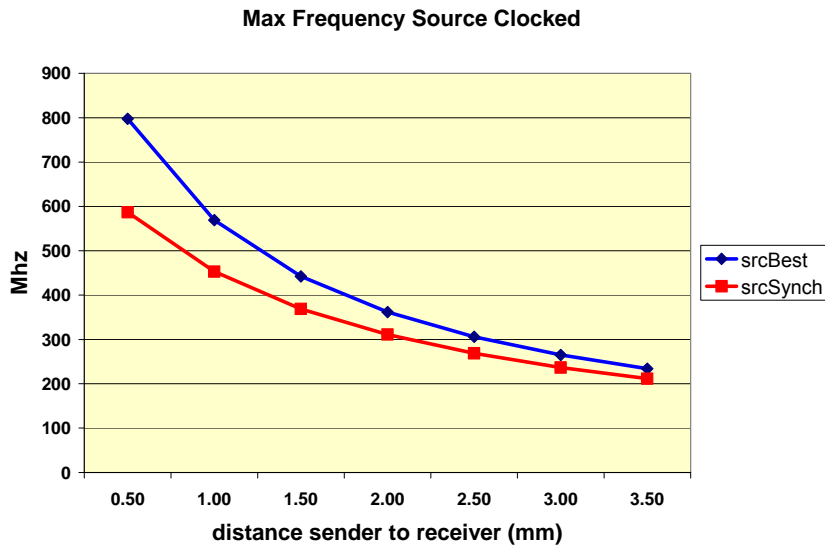
slew rates, which could easily differ from line to line. The result is level shifters are a dynamically variable source of skew.

Figure 9 - Source Synchronous skew window



Another problem to overcome is jitter. The skew windows must be large enough to cover not only systematic skew, but cycle to cycle jitter. We often forget that jitter is inherent to CMOS ASIC design flow. Globally synchronous clocking is immune to clock jitter not because the jitter coupling is absent, but because the jitter sources are guaranteed to be quiescent near the global clock edge. As soon as we create numerous local clocks, we need to worry about what actively coupled jitter sources do to those clock edges. A model was created for the standard ASIC design flow in order to predict what kind of jitter and skew would result. Inputs such as routing efficiency, placement dispersion, likelihood of aggressor nets were included.

Figure 10 - Source Synchronous Model



The model, depicted in Figure 10, shows source synchronous clocking is not able to achieve high frequencies even over moderate distances. Even the srcBest curve, which assumes an ideal duty cycle and ideal clock launch timing, can only achieve about 400 Mhz at 2 mm distances. It is hard to see this as anything but marginally better than globally synchronous.

Source Synchronous Conclusion

As just shown, frequency is limited. Latency is actually higher than for traditional synchronous design, since the traversal takes about the same time, but then a normal double rank synchronization is still needed into the receiver domain. And, gate count can be quite high. There will be a crossing link associated with each source to destination pair. Each link will have a source synchronous clock, and a crossing buffer will be needed between that clock domain and the receiver domain. The result is the number of crossing buffers scales with the number of connections, rather than with the number of senders and/or receivers.

Although there may be specific cases where source synchronous clocking is a good fit, it does not appear that this is a good generalized approach to implementing GALS designs.

Synchronous with Traditional Crossings

At this point in time, Sonics believes the traditional method for managing asynchronous domains within a synchronous logic design style is the best choice for implementing GALS designs. It is the most effective, the most efficient, the most conservative, the most compatible, and the most familiar solution. This does not mean, however, that it is without its drawbacks. These were discussed in the introduction to this section. What is needed for an optimal solution is to precisely understand the use environment, then to design a solution that serves the particular needs of that environment while minimizing the traditional costs of asynchronous crossings.

Practical GALS Designs

The extreme case of a GALS SoC would have every single core in its own voltage and clock domain. But, nobody is trying to do that. It requires too many voltage regulators; too many clock generators; too much area for voltage switches, isolation buffers, level shifters, and crossing buffers; and too much complexity for the power manager to control it all. Plus, there is little to be gained by being that extreme. Since synchronization incurs a latency penalty, there is an advantage to operating groups of cores at one frequency. In addition, most devices have well defined operating modes, where a particular activity is present, and that activity requires a specific set of cores. So let's consider for a moment what a practical GALS SoC will look like.

In almost every SoC there will be an "always on" power domain. It will contain the oscillator driver, maybe the primary PLL, and a few essential peripherals. Things like the low frequency heartbeat that you might find in a GSM application and a real time clock are usually found in this set. And, of course, this is where at least part of the power manager lives. For vast periods of time when the rest of the SoC is asleep, this is the only domain that is powered. Software access to the domain is needed, but such access is rare and has extremely low performance needs. All that is required is an access port that can be voltage isolated. Allowing the access port to operate asynchronously simplifies the design, since otherwise the regulators for the always on domain and the accessing domain would have to lock to the same voltage, so they would operate at approximately the same clock speed. Since this is a very low performance port, it is desirable to be low cost.

Currently, if any processing is happening anywhere on the device, the primary memories need to be powered. For the vast majority of SoCs today, this means the path to external DRAM must be active. Since the interconnect will be powered to serve the access to memory, and the memory controller itself will be powered, it's not so certain that a voltage boundary will exist between the DRAM controller and the rest of the system. However, it is very likely that a clock boundary will exist. Often different product cost points are addressed with different speed DRAMS. DRAM interface speeds are getting quite high, requiring special mixed signal circuitry. It is likely undesirable to attempt to run the entire system at these speeds. Since DRAM utilizations are usually less than 100%, the system interface can be run at a lower clock rate and a higher utilization in order to match the DRAM bandwidth. Furthermore, the memory controller generally has a reasonably sized buffer to queue memory transfers. This buffer is a natural place for an asynchronous crossing without incurring additional gate costs. The crossing latencies can often be hidden behind normal queuing delays.

Most SoCs have some peripherals, or even initiator cores, that are bound to external interfaces such as USB, I2C, SSI, etc. In many cases, the specification for the external interface protocol will dictate a frequency. This requires a clock boundary, but not necessarily a voltage boundary. For most cases, the performance demand fits easily within the capabilities of typical peripheral interconnect.

Beyond the cases already mentioned, the major need for local clocks is driven by the power management needs of the device. Most devices have distinct operating modes where certain activity is present such as graphics, audio processing, or video processing. The set of cores involved in that activity are grouped into a sub-system, which lives within a voltage domain. If the activity is not present, the domain is switched off. If the activity is present, parameters associated with the activity such as screen resolution, audio sampling rate, etc. may be used to control the voltage and frequency. The number of such sub-systems is usually relatively small – 4 to 8. In most cases, the performance needs for these sub-systems are substantial, and there are usually multiple independent threads of activity within the sub-system, which act as sources for accesses to memory that is outside the sub-system. These sub-systems require an asynchronous bridge that is high performance and voltage isolating.

Although the gate count needed to support maximum performance across an asynchronous boundary can be large, not every sub-system needs the maximum performance. On the other hand, limiting the gate count could lower the achievable performance. Some sub-systems cannot tolerate this. What is really needed is a family of bridges, or a configurable architecture, so that an optimum cost vs performance trade-off point can be achieved for each sub-system.

Sonics Solutions

Sonics interconnect products include the features necessary to address each of the needs described above.

Table 1 - Sonics GALs Building Blocks

<i>need</i>	<i>solution</i>
Always on power domain <ul style="list-style-type: none"> — 32Khz heart beat — RTC — power manager 	OCP Async Bridge <ul style="list-style-type: none"> — voltage isolating — low performance — low cost
DRAM interface <ul style="list-style-type: none"> — different frequencies/cost points utilization discontinuity 	MemMax™ <ul style="list-style-type: none"> — scheduler storage buffer — isolates system from DRAM freq — MT non-blocking, QOS
External I/F peripherals <ul style="list-style-type: none"> — specified i/f frequency 	Sonics3220™ target Async <ul style="list-style-type: none"> — async without isolation
Initiator sub-systems <ul style="list-style-type: none"> — device operating modes — handful (4 to 8) 	SonicsExpress™ <ul style="list-style-type: none"> — high performance, async, isolating, — power handshaking

The OCP async bridge is an OCP to OCP bridge that crosses clock domains. It places no restrictions on clock frequency ratio. It is optimized for extremely low gate count to support low performance sockets. It is designed to support an optional voltage domain boundary that is coincident with the clock domain boundary. It is an ideal solution for the connectivity needs of the always on voltage domain, as well as for connecting to the control port of sub-systems.

MemMax is a memory scheduler that acts as an intelligent front end to a DRAM controller. It provides QoS control of DRAM access scheduling that is “DRAM aware,” considering costs of things like read-write turn around, chip switch, and page miss. It implements a multi-threaded storage buffer for request queuing. This storage buffer has asynchronous crossing capability. The scheduler storage buffers are a natural place for such a crossing, so that the system frequency can be decoupled from the DRAM frequency without incurring additional gate costs.

The Sonics3220 interconnect is optimized to service peripherals. It supports a large physical span and connectivity of a large number of peripheral targets. The gate count needed to connect each

peripheral is on the order of 1K gates. The performance achievable at each target interface is moderate. An option is available at each target to include a low performance asynchronous bridge. This bridge is similar to the OCP async bridge, except it does not support voltage domain crossing. This makes it possible to directly connect targets whose operation is tied to an external clock frequency.

SonicsExpress is a high performance asynchronous OCP to OCP bridge that crosses clock domains. It is designed to support an optional voltage domain boundary that is coincident with the clock domain boundary. It places no restrictions on clock frequency ratio. It is highly configurable with respect to the gate cost vs performance trade-off including support for non-blocking multi-threaded behavior. It is an ideal solution for connecting sub-system initiator ports to the interconnect that services shared primary memories and peripherals.

Figure 11- Mobile Handset Example

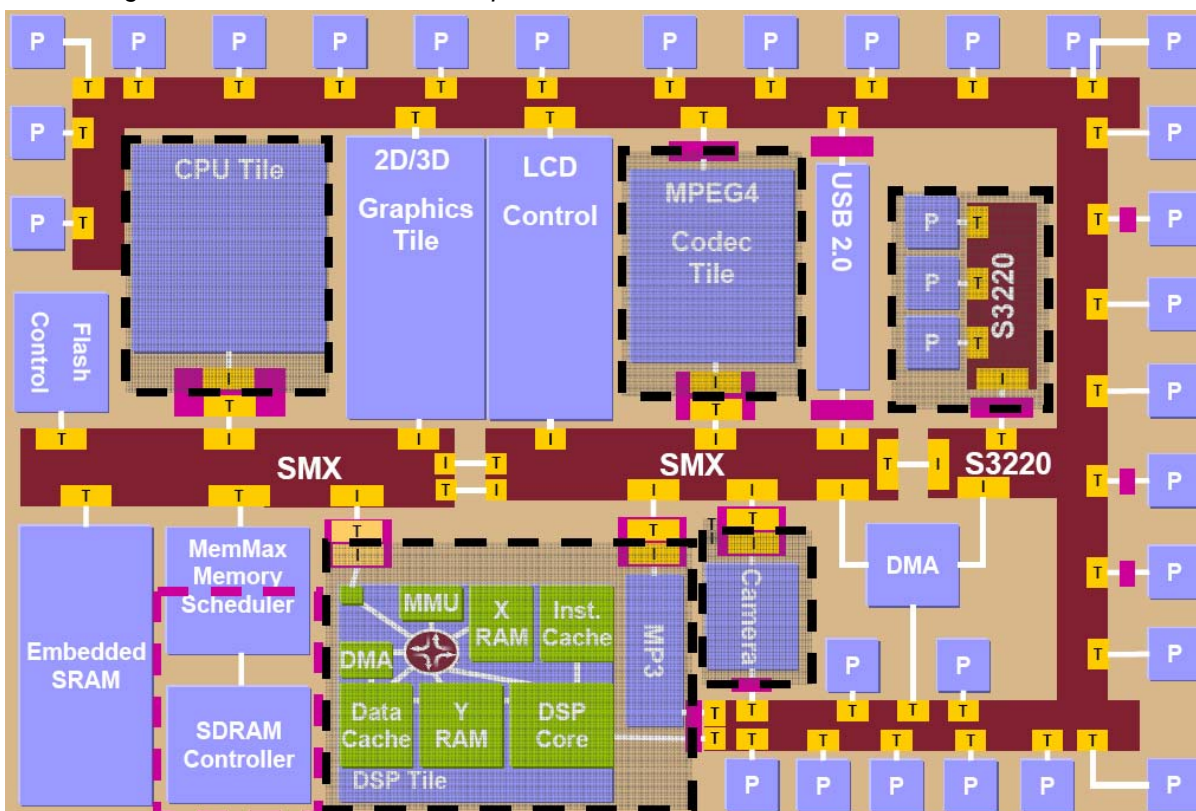


Figure 11 shows an example SoC in a handset design. It has all the elements just discussed. There is a small power domain that is always on. It contains three peripherals connected by a private instance of Sonics3220. This instance acts as a sub-system target for the main peripheral interconnect. The domain is isolated via a low performance isolating OCP async bridge.

The design also calls for a set of peripherals that are tied to an external clock frequency. No isolation function is needed here. Each of these targets is served by the low performance asynchronous bridging available in the Sonics3220.

The external DRAM runs at a separate frequency from the rest of the SoC. The asynchronous crossing in MemMax is used for this, creating a small high frequency clock domain.

Finally, four separate voltage islands are separated from the rest of the system, this enables them to be powered off independently, and/or voltage scaled independently. These use a SonicsExpress for the high performance crossing, including isolation, and the low performance OCP async bridge for the programming port, also including isolation.

As you can see the gate count burden imposed by the asynchronous bridging is reasonable, yet significant chunks of power consumption are partitioned into separate manageable domains. The gate costs of the clock crossings within the OCP async bridge and the bridge within the Sonics3220 target agent, as well as within MemMax are extremely low. This means the overall cost of the GALS solution is dominated by the SonicsExpress uses. The performance implications of the GALS solution are dominated by the SonicsExpress and MemMax uses. For the most part, the crossing delays in MemMax are hidden as queuing delay. Because the DRAM service rate is typically less than the demand rate of the initiators, requests typically see contention and pacing. The SonicsExpress instances will have the most visible impact on performance.

Conclusions

In this paper we have examined GALS style SOC design in light of today's practical design realities. There are a set of notions we should take away from this examination.

1. First, it is important to understand that challenges with respect to clock budgets and power dissipation are what motivate the GALS approach today. Splitting the SoC into islands with switchable and/or scalable voltages lets power consumption be minimized. Keeping the islands reasonably sized minimizes timing closure problems caused by clock distribution costs.
2. Second, clockless and source synchronous approaches pose more problems and risks than benefits. The tried and true traditional synchronous logic design style is still the best choice for the domain crossing components needed for GALS, so long as those crossing components also integrate the voltage crossing.
3. Third, these crossing components do represent measurable costs, both in terms of area and latency. In order to amortize these costs, it is the best trade-off to structure domain crossings mostly at sub-system boundaries. This exposes blocking challenges that must be effectively handled by the crossing components.
4. Fourth, the most significant component needed to support GALS implementations is a high performance, voltage isolating, multi-threaded and non-blocking bridge. There are a number of key characteristics for this component that are required in order for it to be practical to use.
5. Finally, with the addition of SonicsExpress, Sonics offers a complete solution for practical GALS architectures today. The solution includes all the optimized crossing elements needed to achieve a GALS implementation that is optimized for high performance, low area, and low power consumption.



1098 Alta Ave, Suite 101
Mountain View, CA 94043 USA

Phone: +1 650 938 2500
Toll Free: +1 877 938 2800
Fax: +1 650-938 2577

Customer Support and Feedback
support@sonicsinc.com

www.sonicsinc.com