azuro®

Clock Concurrent Optimization

Rethinking Timing Optimization to Target Clocks and Logic at the Same Time

Paul Cunningham, Marc Swinnen, Steev Wilcox

February 2009

Introduction

Ten years ago, the EDA industry faced a crippling divergence in timing between RTL synthesis and placement caused by rapidly rising wire capacitances relative to gate capacitances. Without some reasonable level of placement knowledge to give credible estimates of wire length it was becoming impossible to measure design timing with any accuracy during RTL synthesis. At this time, placement tools were not directly aware of timing and focused instead on metrics indirectly related to timing such as total wire length. As chip designs scaled to "deep sub-micron" geometries (180nm and 130nm), the change in timing around placement became so significant and unpredictable that even manual iterations between synthesis and placement no longer converged. The solution was to re-invent placement, making it both directly aware of timing and also weaving in many of the logic optimization techniques exploited during RTL synthesis, for example gate sizing and net buffering. This process was not easy, and ultimately saw a major turnover in the backend design tool landscape as a new generation of "physical optimization" tools were developed, released and proliferated throughout the chip design community.

Today timing is diverging once again, but for a different set of reasons (on-chip variation, low power, and design complexity) and at a different point in the design flow (CTS). While this divergence has so far received little media attention, this paper shows that the divergence is severe – so much so that we believe it is having a critical impact on the economic viability of migration to the 32nm process node. Clock concurrent optimization is a revolutionary new approach to timing optimization which comprehensively addresses this divergence by merging physical optimization into CTS and simultaneously optimizing both clock delays and logic delays using a single unified cost metric.

This paper begins with a brief overview of some basic concepts in clock based design, and a brief overview of the traditional role of CTS within digital design flows. It then explains why and by how much design timing is diverging around CTS. The concept of clock concurrent optimization is then introduced and its key defining features outlined. The paper concludes with a summary of the key benefits of clock concurrent optimization and an explanation of why it comprehensively addresses the divergence in design timing around CTS.

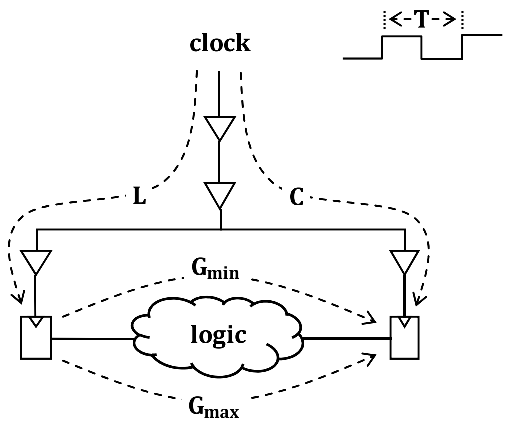
A Brief Overview of Clock-Based Design

Setup and Hold Constraints

Clocking was one of the great innovations which enabled the semiconductor industry to progress to where it is today. Clocking elegantly quantizes time, enabling transistors to be abstracted to sequential state machines and from state machines into a simple and intuitive programming paradigm for chip design, the register transfer language (RTL).

The fundamental assumption made by sequential state machines, and hence by any RTL specification, is the assumption of sequential execution, i.e. that all parts of a state machine stay "in-step" with respect to each other. This assumption translates to a set of constraints on delay which must be met by any clock-based design if it is to function correctly. These constraints split into two classes:

- Constraints which ensure that every flip-flop always makes a forward step from state n to state n+1 whenever the clock ticks. These constraints are typically referred to as **setup** constraints.
- Constraints which ensure that no flip-flop ever makes more than one forward step from state n to state n+2 on a single clock tick. These constraints are typically referred to as **hold** constraints.



Setup Constraint: $L + G_{max} < T + C$

Hold Constraint: $L + G_{min} > C$

Figure 1: Setup and hold constraints in clock based design.

One setup and one hold constraint are required for every pair of flip-flops in a design which have at least one functional logic path between them. Figure 1 summarizes the setup and hold constraints for a pair of flip-flops, A and B, triggered by a clock with period T. The clock delay to A is denoted by L for "launching" clock and the clock delay to B is denoted by C for "capturing" clock. G_{min}, G_{max} denote the minimum and maximum logic path delays between the two flip-flops. For simplicity, and because it makes no difference to the arguments we make in this paper, we have assumed the setup time, hold time, and clock-to-Q delay for the flip-flops are all zero.

The setup constraint is read as follows: the worst-case time taken for a clock tick to reach A, and propagate a new value to the input of B, must be less than the time taken for the next clock tick to reach B. If this isn't true then it is possible that B be clocked when its input does not yet hold the correct next-state value.

The hold constraint is read as follows: the best-case time taken for a clock tick to reach A, and propagate a new value to the input of B, must be greater than the time taken for that same clock tick to reach B. If this isn't true then it is possible that a next state value on the input to A may propagate all the way through to the output of B in one clock tick, in essence causing B to skip erroneously from state n to state n+2 in one clock cycle.

Ideal and Propagated Clocks Timing

In the context of modern digital chip design flows, the setup and hold constraints outlined above are referred to as a **propagated clocks** model of timing since the constraints start from the root of the clock and include the time taken for the clock edge to propagate through the clock tree to each flip-flop. The propagated clocks model of timing is the definitive criteria for correct chip function, and is the one used by timing sign-off tools in design flows.

An **ideal clocks** model of timing simplifies the propagated clocks model of timing by assuming that the launch and capture clock paths have the same delay, i.e. that L=C. In this case the setup and hold constraints simplify significantly:

	Propagated Clocks		**Ideal Clocks**
Setup:	$L + G_{max} < T + C$	(*assume L = C*)	$G_{max} < T$
Hold:	$L + G_{min} > C$	(*assume L = C*)	$G_{min} > 0$

Since $G_{min}>0$ is to a first approximation always true, assuming that L=C simplifies the entire problem of ensuring that a clock based design will function correctly to $G_{max} < T$. In this universe there is no need to worry about clock delays or about minimum logic delays. All that matters is making sure that the maximum logic path delay in the design, typically referred to as the "critical path", is faster than the clock period. In essence, clocks have been canceled out of the timing optimization problem.

The concept of ideal clocking is so dramatic and powerful as to have enabled an entire ecosystem of "frontend" engineers and design tools living in a world of ideal clocks, and the origins of ideal clocking are so deep rooted in the history books of the semiconductor industry that clock based design is itself often referred to as "synchronous design" even though there is nothing fundamentally synchronous about clock based design itself!

Clock Skew and Clock Tree Synthesis

If chip design begins in a world where clocks are ideal but ends in a world where clocks are propagated it follows that at some point in the design flow a transition must be made between these two worlds. This transition happens at the clock tree synthesis (CTS) step in the flow where clocks are physically built and inserted into a design: see Figure 2.

Since ideal clocks assumes L=C for all setup and hold constraints it follows that the traditional purpose of CTS is to build clocks such that L=C. If this can be achieved then propagated clocks timing will match ideal clocks timing and the design flow will be "convergent".

If a clock tree has n sinks and a set of paths P[1] to P[n] from its source to each sink then the "skew" of that clock is defined as the difference between the shortest and longest of these paths: see Figure 3.

Mainstream CTS tools are architected primarily to build highly efficient "balanced" buffer trees to a very large number of sinks with a small skew. Twenty years ago, the motivation and benefits of building balanced clocks was clear: clock skew was an upper bound on the worst difference between L and C for any pair of flip-flops, i.e. an upper bound on |L-C| for any possible setup or hold constraint which could apply to a design. If a small skew could be achieved relative to the clock period then a high degree of similarity between ideal and propagated clocks timing was guaranteed. But it is important to remember that clock skew and the worst difference between L and C are not the same thing and that for a modern SoC design at nanometer process nodes it is entirely possible (in fact very common) for |L-C| to be significantly greater than the clock skew.

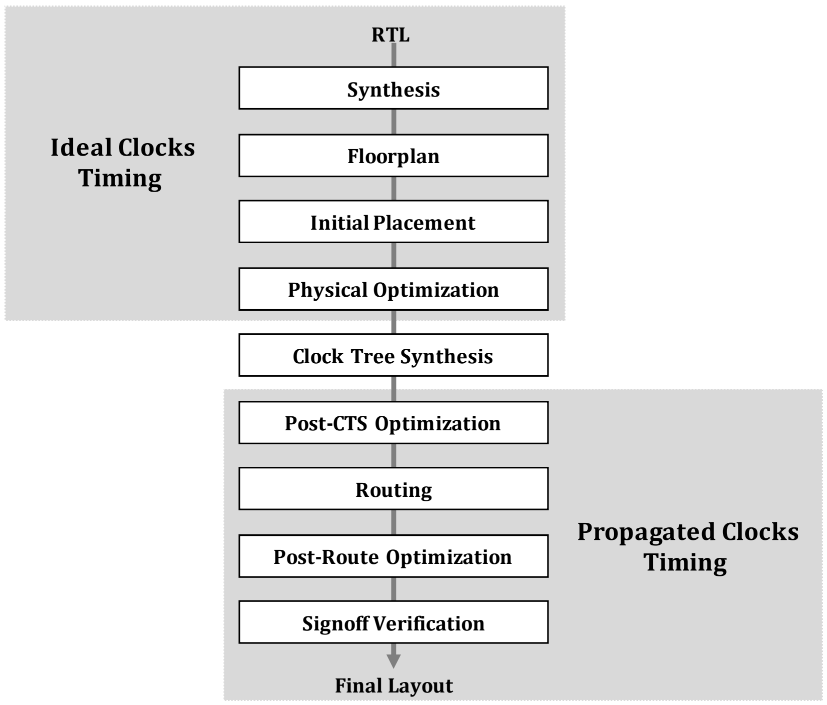
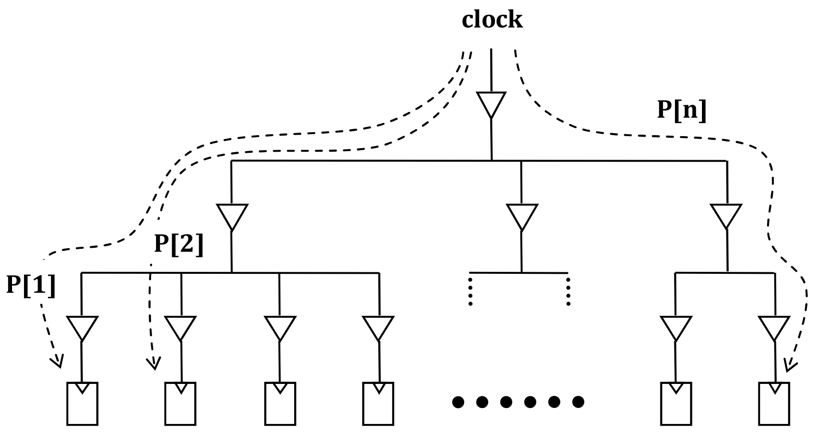


Figure 2: Traditional balanced clocks design flow



Skew = max(P[1],P[2],…,P[n]) – **min**(P[1],P[2],…,P[n])

Figure 3: Skew of a Clock Tree

The distinction between clock skew and |L-C| is a critical foundation stone for this paper. Clock skew is a concept defined in terms of worst differences in delay between source-to-sink paths in buffer trees. L and C are delay variables in the setup and hold constraints of a propagated clocks model of timing and are not really different in this context from the other delay variables, G_{min} and G_{max}. The somewhat slippery nature of this distinction and the ease with which a discussion can begin in the context of timing and then migrate mistakenly into a context of skew is one of the primary reasons why we believe the divergence in design timing around CTS has for so long gone relatively unnoticed by the chip design and EDA communities.

The purpose of this paper is not to argue that tight skews cannot be achieved for modern nanometer designs. Nor is it to argue that the skew minimization techniques used by mainstream CTS tools no longer work for modern nanometer designs. The purpose of this paper is to argue that the ability of tight clock skews to bind ideal clock timing to propagated clocks timing is broken – we estimate it broke in a commercial sense around the 65nm node. No tweak or refinement to the definition of skew can fix this. The only solution is to give up entirely on the concept of skew and focus CTS instead on the fundamental propagated clocks timing constraints that will matter post-CTS in the flow. But in this context there is no longer any material distinction between clock paths (L and C) and logic paths (G_{min} and G_{max}). Constructively exploiting this observation is the inspiration behind the clock concurrent approach optimization.

The Clock Timing Gap

There is no question that clock based design, ideal clocks timing, the use of RTL to specify a chip, and the concepts of frontend vs. backend design are all vital foundation stones for the continued success of the semiconductor industry – their collective ability to enable design automation and streamline engineering productivity would almost certainly be impossible to achieve by any other means.

However, the traditional role of CTS to build buffer trees with tight skew only makes sense if achieving these tight skews reasonably binds ideal clocks timing to propagated clocks timing. If this is not the case then timing decisions made using ideal clocks have only limited value. Accommodating a change in timing landscape after CTS requires either accepting degradation in chip speed or accepting delay in time to market due to increased iterations back to RTL synthesis and physical optimization. If it can be argued that the divergence between ideal clocks and propagated clocks is both significant and fundamental, i.e. one where there could never be any formula or metric which could bind them, then the only solution becomes to rethink CTS as a timing optimization step in the flow which directly targets propagated clocks timing as its objective.

In this section we attempt to define and measure the magnitude of the gap between ideal and propagated clocks timing. For a particular timing constraint i with launch clock delay L[i], capture clock delay C[i], minimum and maximum logic path delays $G[i]_{min}$ and $G[i]_{max}$, the difference between ideal and propagated clocks timing for that constraint can easily be seen to be the magnitude of C[i]-L[i]. For example if i were a setup constraint then we have:

$$\begin{array}{rl}
\textbf{Propagated clocks timing:} & L[i] + G[i]_{max} < T + C[i] \\
= & G[i]_{max} < T - (L[i]-C[i]) \\
\textbf{Ideal clocks timing:} & \underline{G[i]_{max} < T \qquad\qquad\qquad} \\
\textbf{Difference =} & L[i]-C[i]
\end{array}$$

A similar reasoning gives the opposite, L[i]-C[i], for hold constraints. We define the clock timing gap for a particular set of timing constraints (either setup or hold or a mixture of both) as:

$$\textbf{Clock Timing Gap} = \boldsymbol{\sigma}\left(\frac{L[i] - C[i]}{T}\right)$$

Our choice of standard deviation, σ, on L[i] – C[i] rather than average or worst |L[i] – C[i]| is important: we do not want to measure a large clock timing gap if the delta between ideal and propagated clocks timing is systematic or applies only to a very small number of timing constraints. If this were the case then the gap would not be a fundamental gap and could easily be worked around by applying a global safety margin (aka global uncertainty) to the ideal clocks timing model or by manually applying a few individual sink pin offsets to CTS. What we want to measure are true *unsystematic* divergences between ideal and propagated clocks timing which apply to a *significant proportion* of the timing constraints. These divergences will never be resolvable with a small amount of manual effort or with any generalizations to the concept of skew.

We divide L[i]-C[i] by the clock period T to normalize our metric so that it is expressed as a percentage of the clock period. This enables us to meaningfully compare the average clock timing gap across a large number of designs across a range of clock frequencies and process nodes.

Figure 4 below summarizes the average clock timing gap for the top 10% worst violated setup constraints across a portfolio of over 60 real world commercial chip designs from 180nm to 40/45nm and from 200k to 1.26M placeable instances. It shows that while at 180nm the clock timing gap is small at around 7% of the clock period, at 40/45nm the gap has widened to around 50% of the clock period. A gap of this magnitude is sufficient to completely transform the timing landscape of a design beyond recognition between before and after CTS. Since our measure is one of standard deviation and not average or worst difference this gap truly is a fundamental divergence which can only be addressed by a fundamental rethink of the role of CTS in the design flow. Building clocks to meet a tight skew target no longer achieves its purpose nor will any other indirect metric ever bind ideal clocks timing to propagated clocks timing since the divergence is unsystematic and large for a significant number of worst violating timing endpoints. The only solution is to directly target the propagated clocks timing constraints and treat the launch and capture clock paths (L and C) as optimization variables with the same significance and similar degrees of freedom to logic path variables (G_{min} and G_{max}). This is what clock concurrent optimization is all about.

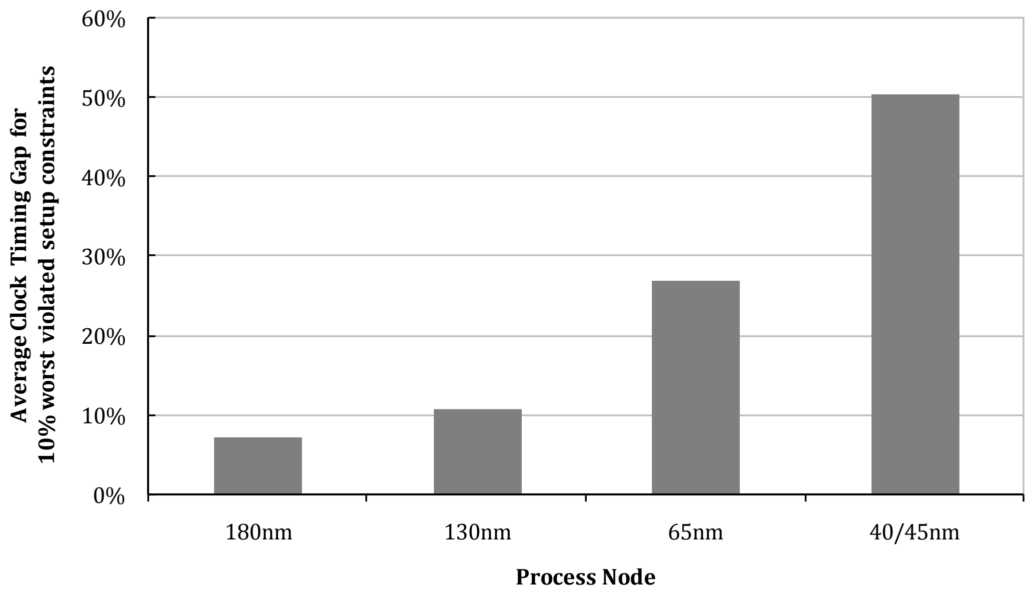


Figure 4: Clock Timing Gap across a portfolio of over 60 commercial designs

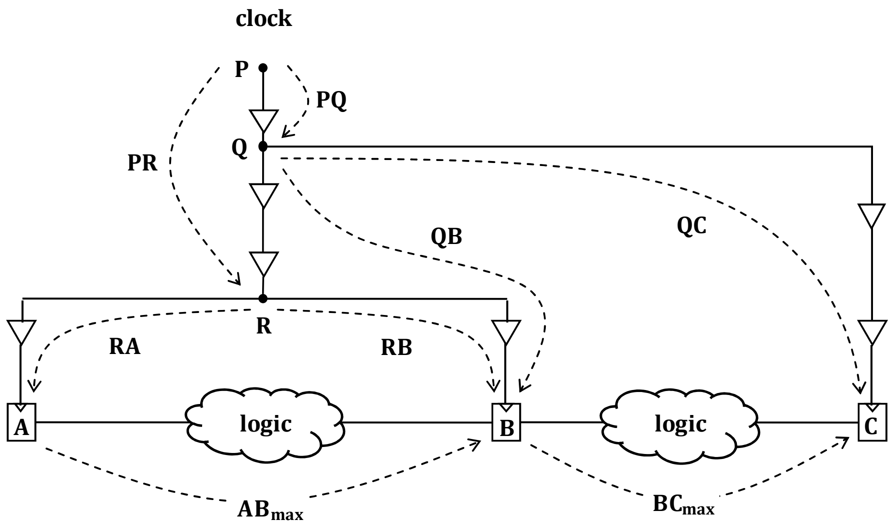
Explaining the Clock Timing Gap

There are three key underlying industry trends which are causing ideal and propagated clocks timing to diverge, and it is the relatively simultaneous onset of all three trends that has caused the clock timing gap to open up so dramatically at the 65nm node and below. These three trends are on-chip variation, clock gating, and clock complexity.

On Chip Variation

On chip variation (OCV) is a manufacturing driven phenomenon. Two wires or two transistors which are designed to be identical almost certainly won't be once printed in silicon due to the lithographic challenges of printing features smaller than the wavelength of light used to draw them. As a result the performance of two supposedly identical transistors can differ by an unpredictable amount. This problem is a significant and growing one, and at 45nm these random manufacturing variations can impact logic path delays by up to 20%.

OCV is particularly relevant for clock paths since the length of clock paths (i.e. the insertion delay of clock trees) is rising exponentially with respect to clock periods. This is in part because the number of flip-flops in a design continues to rise exponentially but also because resistances are rising so fast with successive process shrinks that buffering across long distances, as is typically necessary in the clock, requires more and more buffering per unit length. At 45nm it is not uncommon to see 3-5 times the clock period worth of delay in launch and capture clock paths. Even if the impact of OCV is only 10% of path delay this still amounts to a potential change in timing pictures of 30-50% of the clock period between ideal and propagated clock models. The only reason why OCV has not already ground chip design completely to a halt is the fact that it can be ignored on the common portion of the launch and capture clock paths using a technique known as common path pessimism removal (CPPR) or clock reconvergence pessimism removal (CRPR): see Figure 5.



Setup: $PR+1.1(RA+AB_{max}) < PR+0.9(RB)$ **Setup:** $PQ+1.1(QB+BC_{max}) < PQ+0.9(QC)$

Hold: $PR+0.9(RA+AB_{max}) > PR+1.1(RB)$ **Hold:** $PQ+0.9(QB+BC_{max}) > PQ+1.1(QC)$

Figure 5: Propagated clocks timing with ±10% OCV derates and CPPR

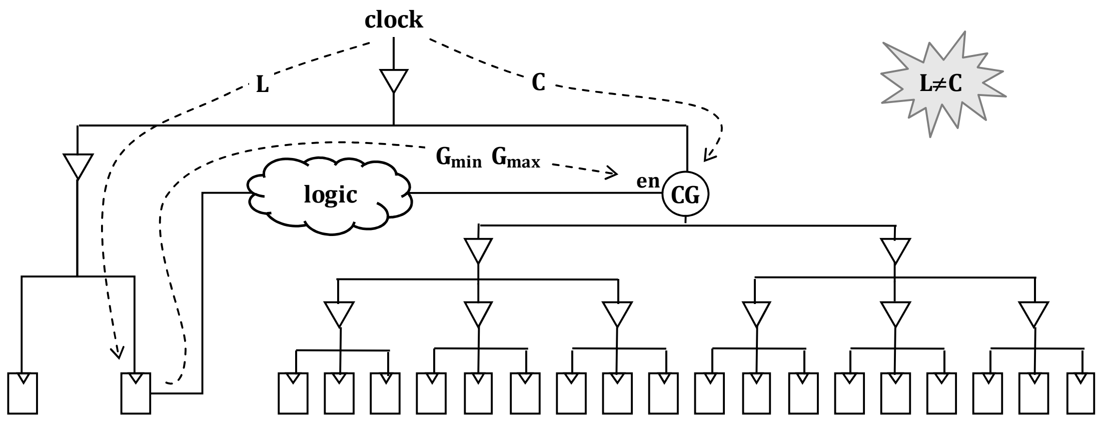
CPPR is highly constraint dependent, impacting one pair of flip-flops completely differently from another since it depends crucially on where in the clock tree the launch and capture clock paths converge for a particular pair of flip-flops.

A traditional measure of clock skew ignores OCV, so even if clock skew is zero, once OCV derates and CPPR are applied to a design the magnitude of L – C can be large for a significant number of logic paths. Also, since CPPR makes the impact of OCV on launch and capture clock paths constraint dependent, there is no meaningful way to predict or model this impact prior to CTS. In this sense OCV modeling, most specifically the use of CPPR, is a direct and fundamental contributor to the clock timing gap.

Clock Gating

Power has for many modern chip design teams become as significant an economic driver as chip speed and area, and the clock network is often the biggest single source of dynamic power dissipation in a design. It has become standard practice for almost all modern designs to manage clock power aggressively through the extensive use of clock gating to shut down the clock to portions of a design which do need to be clocked in a particular clock cycle. Clock gating can be at a very high level, for example shutting down the entire applications processor on a mobile phone when it is not needed, or at a very fine grained level, for example shutting down the top 8-bits of a 16-bit counter when they are not counting. Modern systems on a chip can contain tens of thousands of clock gating elements.

From a timing perspective, a clock gate is just like a flip-flop, bringing with it its own setup and hold constraint. But, unlike a flip-flop, clock gates exist inside a clock tree and not at its sinks: see Figure 6. In an ideal clocks timing model a clock gate typically looks exactly the same as a flip-flop since the clock arrives instantaneously everywhere (i.e. L=C=0), but in a propagated clocks timing model there can be a massive difference between the launch and capture clock path delays for a clock gate, especially for architectural clock gates high up in the clock tree.



Setup Constraint: $L + G_{max} < T + C$

Hold Constraint: $L + G_{min} > C$

Figure 6: clock gate enable timing using a propagated clocks model of timing

Every clock gate added to a design adds to the clock timing gap on that design, and the more aggressive the design team is in managing power, the more this gap is felt. This is because the most common workaround for the pain caused by clock gating timing diverging post-CTS is to force all clock gates to the bottom of the clock tree (by "cloning" them) so that the capture clock path delay is as close as possible to the launching clock delay thereby restoring convergence between ideal and propagated clocks timing on clock gates. This is however, the worst possible strategy for saving power.

Without insisting that all clock gates lie at the very bottom of the clock tree there is no meaningful way to predict the relationship between L and C for clock gates, and therefore clock gating is also a direct and fundamental contributor to the clock timing gap.

Clock Complexity

Modern system-on-a-chip (SoC) designs are big – tens of millions of gates. They also exploit extensive IP reuse, where pseudo-generic modules such as ARM cores, USB interfaces, PCI interfaces, memory controllers, DSPs, baseband modems, and graphics processors are instantiated on a single die, configured to run in particular modes, and stitched together to deliver some form of integrated system capability.

The clock network in these SoCs is not simple. In fact it has become phenomenally complex – often well over a hundred interlinked clock signals that can be branched and merged thousands of times. Part of the complexity is simply a result of stitching together the many IP blocks, but much of the complexity is also inherent to each of these IP blocks: the same low-power imperatives which drive clock gating are also driving the deployment of a wide variety of clocks and clocking schemes at a variety of frequencies and voltages to further control and manage clock activity. Power consumption during built-in system test and time on the tester during production further impact clock complexity as scan chains continue to be sliced and diced and ever more intricate clocking schemes are used to capture and shift quickly and power efficiently.

The end result is a dense spaghetti network of clock muxes, clock xors, and clock generators, entwined with clock gating elements from the highest levels in the clock tree where they shut down entire sub-chips, to the lowest levels in the clock tree, where they may shut down only a handful of flip-flops, see Figure 7.

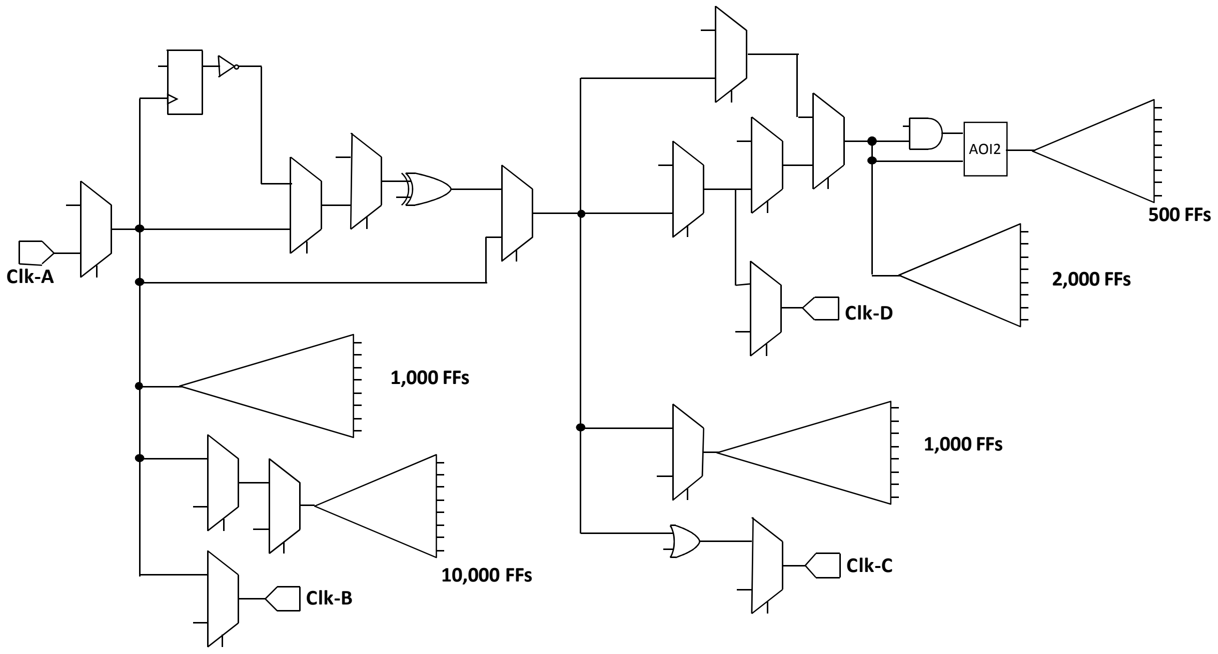


Figure 7: Clock network on a modern nanometer SoC

In this world of vast clock complexity, the definition of clock skew is itself non-obvious: rather than a tree or set of trees we have a network with hundreds of sources and hundreds of thousands of sinks. In such a world it is easy, and indeed even common, to find oneself constructing scenarios where making L=C for all flip-flops is mathematically impossible. And even in the cases where it is theoretically possible to achieve this objective the sheer size of the clock delays that would be required to achieve this would be so large, e.g. 10+ times the clock period, as to make timing impossible to meet even with the tiniest of OCV margins in place. The resulting dynamic power consumption and IR-drop would also almost certainly render the entire design useless.

The only way to implement designs of this complexity using a traditional design flow is to spend months of manual effort carefully crafting a complex set of overlapping balancing constraints, typically referred to as "skew groups", which require that CTS balance certain sets of clock paths with other sets of clock paths. It can take hundreds of such skew groups to be carefully crafted before any reasonable propagated clocks timing can be achieved and such timing is never in practice achieved by ensuring that L=C for all pairs of flip-flops.

In essence, the impact of clock complexity on the clock timing gap has already broken the traditional role of CTS in design flows, and design teams are already manually working around it at great cost by carefully crafting highly complex sets of balancing constraints which are designed to achieve acceptable propagated clocks timing and not L=C for all flip-flops.

Detailed Breakdown of the Clock Timing Gap

Figure 7 below shows the same clock timing gap graph as was shown in Figure 3 but further breaks out the relative contributions of OCV, clock gating and inter-clock timing to this gap. The figure makes it clear that each of the three trends contributes materially to the clock timing gap, and also highlights that clock complexity has the most significant impact on this gap. The graph also highlights our observation that clock skew, as traditionally defined only between registers on a single clock tree, is not broken. The clock timing gap restricted only to setup and hold constraints between pairs of flip-flops in the same clock tree is at most 10% of the clock period, even at 40/45nm. But there are other factors ignored by the concept of clock skew which are progressively eroding its ability to bind ideal to propagated clocks timing in the design flow.

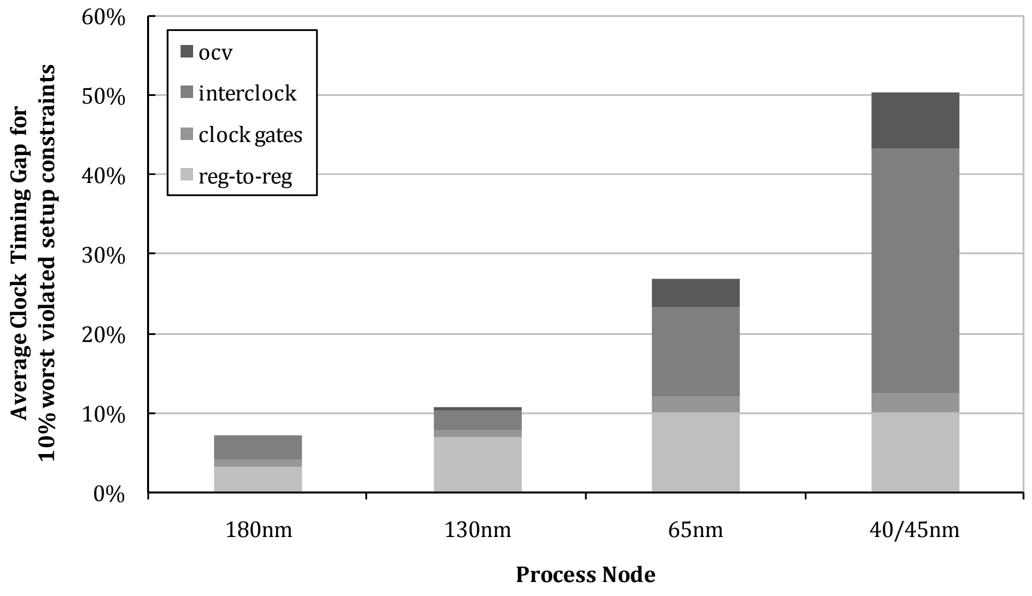


Figure 7: Breakdown of trends contributing to the Clock Timing Gap

The message is simple: the role of CTS in the design flow must change. It can no longer be about minimizing clock skews; it must somehow be about directly transitioning a design from ideal to propagated clocks timing and using every possible trick there is to counteract the surprises that occur as the true propagated clocks timing picture emerges, including clock gates, inter-clock paths and OCV margins.

Clock Concurrent Optimization

Before running CTS in the design flow there are no real launch and capture clock paths and timing optimization focuses exclusively on the slowest logic path, G_{max}, using an ideal clocks model of timing. Provided clocks can be implemented such that the launch and capture clock paths are almost the same for all setup and hold constraints, L=C, this focus on G_{max} makes sense. But, as this paper has shown, OCV, clock gating, and clock complexity have made balancing L and C, or indeed of determining any systematic relationship between L and C, an impossible goal.

The only meaningful goal for clock construction must therefore be to directly target the propagated clocks timing constraints, selecting L and C specifically for the purpose of delivering the best possible propagated clocks timing picture post-CTS. But the “best possible” L and C depend on the values of G_{max} and G_{min} so we have a chicken-and-egg problem: which comes first? Do we pick G_{max} and G_{min} then set L and C, or vice-versa?

Clock concurrent optimization, or CC-Opt for short, is the term we use to describe a new class of timing optimization tools that merge physical optimization with CTS and that directly control all four variables in the propagated clocks timing constraint equations (L, C, G_{min}, and G_{max}) at the same time.

Clock concurrent optimization (CC-Opt) merges physical optimization with CTS and directly controls all four variables in the propagated clocks timing constraint equations (L, C, G_{min}, and G_{max}) at the same time.

Figure 8 visualizes the conceptual distinction between a traditional approach to timing optimization and a clock concurrent approach to timing optimization. We use the term “clock concurrent design” or “clock concurrent flow” to refer to any design methodology or design flow employing the use of clock concurrent optimization.

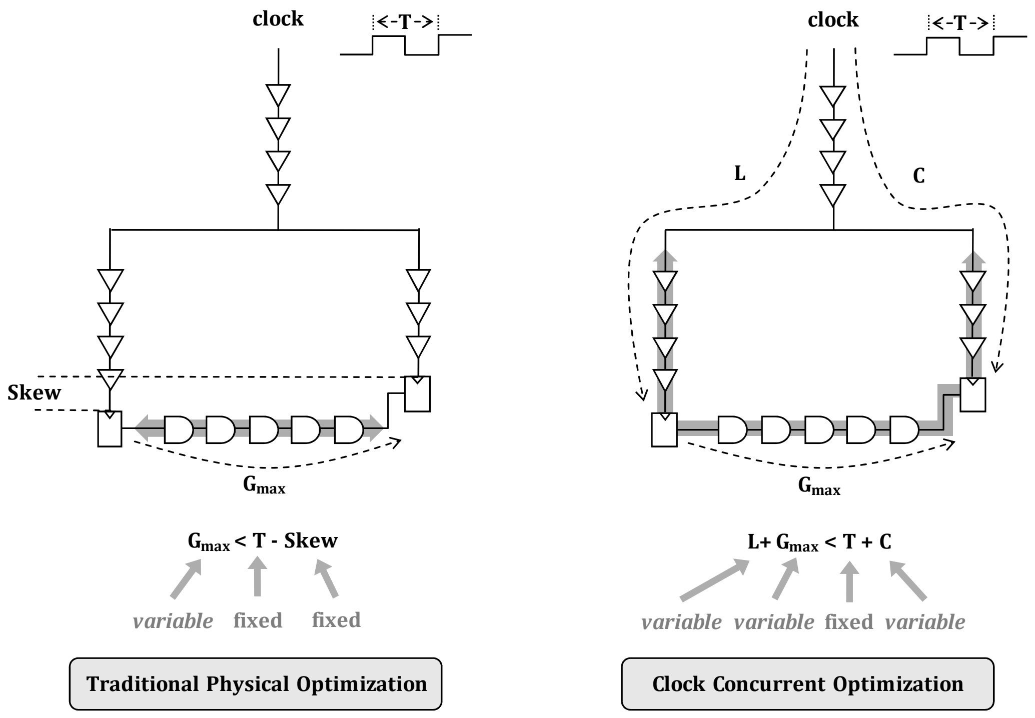


Figure 8: Illustration of the difference between Physical Optimization and CC-Opt

Logic Chains

Since CC-Opt treats both clock delays and logic delays as flexible parameters, the maximum possible speed that a chip can be clocked at is no longer limited by the slowest logic path in a design. CC-Opt allows the capture clock path to be longer than the launch clock path, in which case the logic path may have more than the clock period to compute its result. But this extra time is not a free lunch: if C is bigger than L then time has been borrowed either from either the preceding or subsequent pipeline stages: see Figure 9.

Such time borrowing is iterative across multiple logic stages: if time can be borrowed from logic stage n+1 to logic stage n, then time can also be borrowed from logic stage n+2 to logic stage n+1 and then again from logic stage n+1 to logic stage n, and so on both forwards and backwards from logic stage n, see Figure 10. However, the time borrowing is not unlimited, and must stop either when the chain of logic stages loops back on itself or when it reaches an IO to the chip, see Figure 11.

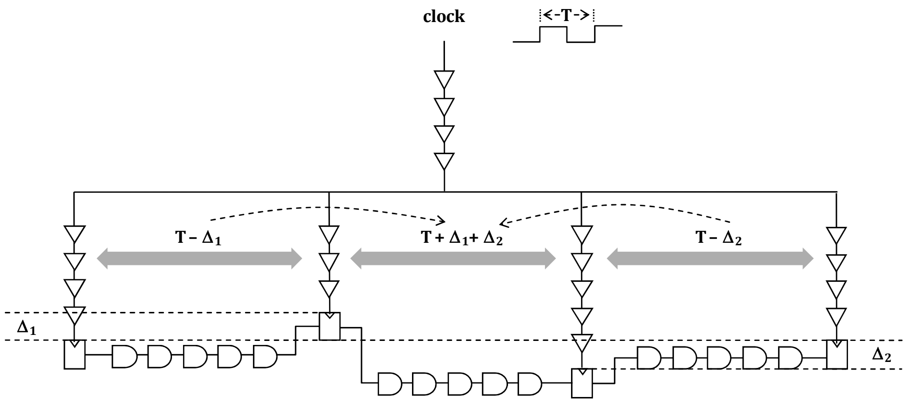


Figure 9: Time borrowing

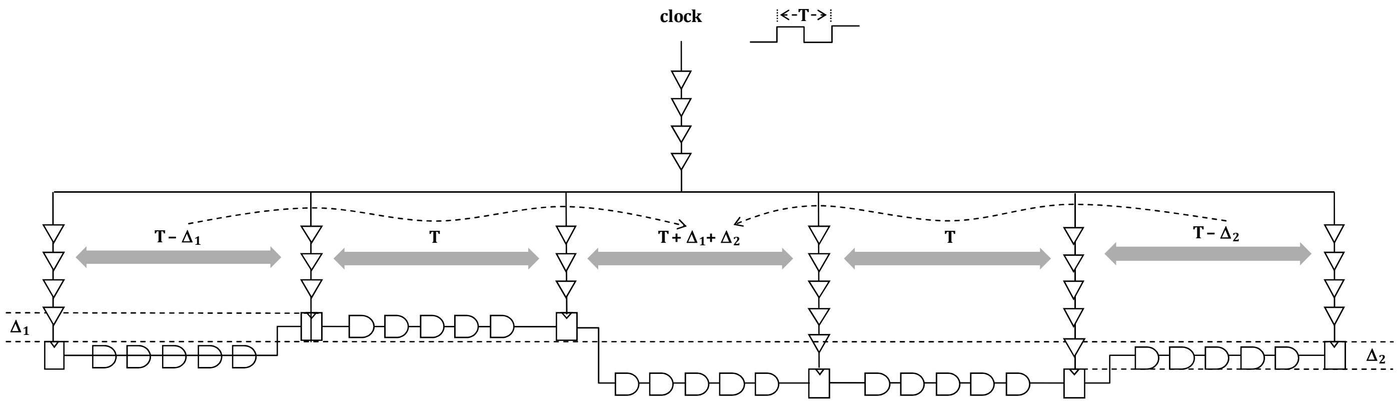


Figure 10: Multi-stage time borrowing

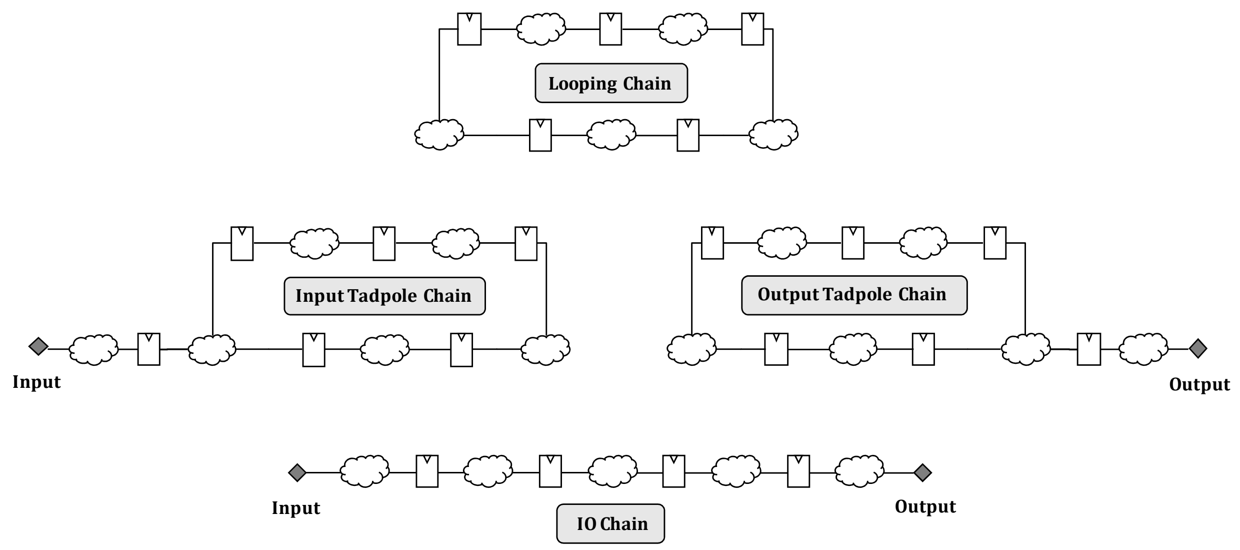


Figure 11: Different types of logic chain

In a world where launch and capture clock paths are flexible optimization parameters, it is these chains of logic functions which most influence the maximum possible clock speed: a chain with n logic stages has at most n clock periods of total time available irrespective of the clock delays to each register in the chain. Provided the worst total logic delay through the entire chain, $\Sigma_i(G[i]_{max})$, is less than nT, it will be possible to come up with a set of clock arrival times for each register on the chain that meets propagated clocks setup constraints. We refer to this relationship as a setup chain constraint:

Setup chain constraint: $\sum_{i=1}^{n} \mathrm{G}[i]_{\max} < \mathrm{nT}$

Only in the highly unusual situation where the most efficient distribution of delay along the chain is one where each stage has exactly the same delay will the optimum clock network be a balanced clock network. The traditional assumption of forcing each stage on the loop to have exactly the same amount of time by balancing clocks is fundamentally unnecessary – and furthermore, as this paper has shown, it is also impossible to achieve on modern chips due to clock complexity, on-chip-variation, and clock gating.

Setup Slack and Sequential Slack

The slowest logic function in a design is typically determined by computing the "setup slack" for each gate in a design and finding the logic path which comprises those gates with the lowest slack value. If we use the term Paths[g] to mean the set of all logic paths which pass through a logic gate g, and for each path p in Paths[g] we use the terms L[p], C[p], and G[p] to refer to the launch clock delay, capture clock delay, and logic delay for path p respectively, then:

Setup Constraint for a path p: $\mathrm{L[p]} + \mathrm{G[p]} < \mathrm{T} + \mathrm{C[p]}$

Setup Slack for gate g: $\mathbf{min}_{p\text{ in Paths[g]}}\big((\mathrm{T} + \mathrm{C}[p]) - (\mathrm{L}[p] + \mathrm{G}[p])\big)$

Setup slack is in essence the worst case margin by which all setup constraints which pass through g have been met. If the setup slack at a gate is negative then a setup constraint must have been violated, and the magnitude of the negativity denotes the amount by which that setup constraint has been violated. The sequence of gates with the smallest setup slacks denotes the logic path which is most limiting chip speed and is typically referred to as the worst negative path, worst violated path, or critical path. The slack value of the gates on the critical path is typically referred to as the Worst Negative Slack (WNS).

It is possible to generalize the concept setup slack to setup chain constraints, giving the concept of sequential slack [Pan98,Cong00], where the term "sequential" is used to emphasize the notion that these slacks can cross register boundaries. If we use the term Chains[g] to mean the set of all logic chains passing through a gate g, and for each chain c we use the term n[c] to refer to the number of logic stages in chain c and G[c,i] to refer to the worst logic delay at stage i in chain c, then:

Setup Chain Constraint for a chain c: $\sum_{i=1}^{\mathrm{n}[\mathrm{c}]} \mathrm{G}[\mathrm{c}, i]_{\max} < \mathrm{n}[\mathrm{c}]\mathrm{T}$

Sequential slack for gate g: $\mathbf{min}_{c\ in\ Chains[g]}\left(\mathrm{n}[c]\mathrm{T} - \sum_{i=1}^{\mathrm{n}[c]} \mathrm{G}[c, i]_{\max}\right)$

It is also helpful to normalize the sequential slack relative to the length of the chain by dividing by n[*c*] so that the margin can be thought of as an average margin per logic stage which is therefore independent of chain length. It also means that sequential slacks are reported on the same scale as traditional setup slacks – if the normalized sequential slack is 100ps it means that the average setup slack along that chain will also be 100ps, irrespective of the clock arrival times at each register on the chain.

Normalized Sequential slack for gate g: $\mathbf{min}_{c\ in\ Chains[g]}\left(\mathrm{T} - \sum_{i=1}^{\mathrm{n}[c]} \mathrm{G}[c, i]_{\max}/\mathrm{n}[c]\right)$

If the smallest sequential slack in a design is negative, then the amount by which the sequential slack is negative, referred to as the Worst Negative Sequential Slack (WNSS), denotes how far off the circuit is from achieving its desired clock speed. We use the term critical chain to describe the logic chain with the WNSS, although the term critical cycle is also used in the literature [Hurst04] to describe the critical chain. We prefer chain to cycle since it emphasizes that the sequence of gates with the WNSS need not form a loop and may equally, and indeed often does, terminate in a design IO.

If sequential slack is negative then traditional setup slacks will never be positive irrelevant of how the clocks are implemented. If sequential slack is positive then it will be possible to build a clock network which delivers clocks to each flip-flop such that traditional setup slacks will also be positive – although this network almost certainly will not be a balanced network. In this sense, using CC-Opt, the maximum speed at which a chip can be clocked is limited by the critical chain and not the critical path. This denotes a fundamental new degree of freedom which can be exploited by CC-Opt above and beyond physical optimization to make chips faster or smaller or lower power.

Using CC-Opt the maximum possible clock speed is limited by the critical chain not the critical path.

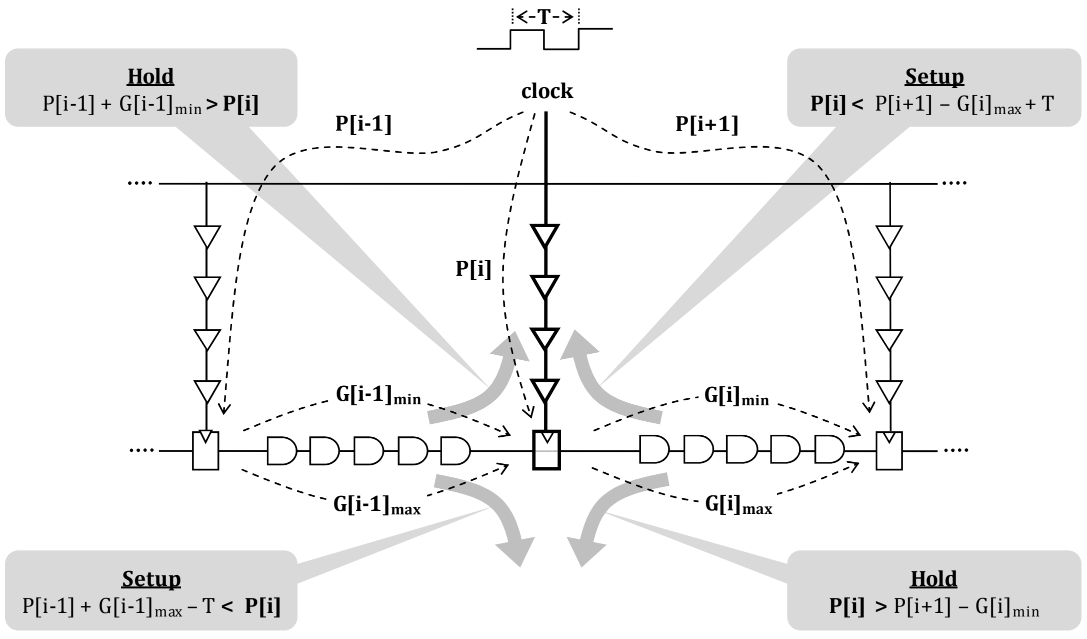
It is however important to note that if sequential slacks are positive this does not imply that setup slacks can also be made positive: firstly, once clocks are built the sequential slacks themselves will change due to OCV, clock gating, and inter-clock timing, and it is entirely possible that this change will make the sequential slacks negative again. Secondly, the clock arrival times necessary to achieve positive setup slacks may not be achievable with a feasibly sized clock tree in terms of area and power.

Sequential Optimization and Useful Skew

Design optimization methods which exploit sequential slacks are usually termed *sequential optimization* methods. These broadly fall into two camps: *retiming* approaches, which physically move logic across register boundaries, and *clock scheduling* approaches, which intelligently apply delays to the clock tree to improve setup slacks. Retiming was introduced over twenty years ago [Leiserson84, Leiserson91], but automatic retiming approaches are flow-invasive because of their impact on formal verification and testability, and have not gained widespread acceptance. Scheduling approaches have also been around for almost twenty years [Fishburn90], and are more applicable in today's flows.

Academic papers on clock scheduling tend to split the problem into schedule calculation [e.g. Kourtev99b, Ravindran03] and schedule implementation [e.g. Kourtev99, Xi97], although some papers [such as Held03] tackle both halves of the problem separately in the same paper. Commercial EDA tools exploiting clock scheduling typically favor more robust and direct algorithms which incrementally add buffers to an already balanced clock tree to borrow time from positive-slack stages to adjacent negative-slack logic stages without ever pre-computing a desired schedule. Although the term *useful skew* was originally applied to the two part calculate-then-implement approach [Xi97, Xi99], it is generally used today to mean any CTS approach that results in an unbalanced tree for timing reasons, even if the approach used doesn't ever explicitly calculate a desired clock schedule. In any case, the key feature in common is that timing ultimately drives the clock arrival times at registers and not a set of CTS balancing constraints.

The best way to think about how CTS can be generalized to be driven by timing and not by a set of balancing constraints is to think of each flip-flop as having four constraints on the arrival time of its clock which are a simple re-arrangement of the propagated clocks timing constraints on its D-pin and on its Q-pin: see Figure 12. These four constraints constrain the permissible arrival time to be within a window which depends on the arrival times of flops in the logical fan-in and fan-out.



$$\max((P[i-1] + G[i-1]_{max} - T), (P[i+1] - G[i]_{min})) < P[i] < \min((P[i+1] - G[i]_{max} + T), (P[i-1] + G[i-1]_{min}))$$

Figure 12: Timing driven clock arrival time windows

The dependency on the neighboring flops makes it easy to see that these clock arrival time windows are globally intertwined. If a clock network can be built which delivers the clock to all flip-flops within their permissible arrival time windows then setup and hold time constraints will be met. This concept of windows can also easily be generalized to include OCV derates and CPPR, and also can be extended to apply to other timing endpoints such as clock gates, clock muxes, and clock generator blocks. It can also be applied to internal nodes in the clock tree by a simple intersection of the windows of sub-nodes.

Although the concept of windowing sounds very powerful, it has one key limitation which we alluded to in the previous section: it is isolated from the physical optimization of logic paths. Separating the steps of optimizing the logic and building the clock tree causes two key problems on real world commercial designs.

The first is the clock timing gap, which as we have shown requires that the desired schedule be based on a true propagated clocks model of timing in order to properly account for OCV, inter-clock paths and clock gate enable timings. If schedule calculation happens before clocks are built then it cannot be based on a propagated clocks model of timing!

The second problem is that clocks are not free. They cost area and power, and any increase in insertion delay causes more setup and hold timing degradation due to OCV on clock paths. While positive sequential slacks imply that a clock network can theoretically be built to make traditional setup slacks positive, this does not mean that such a network would in practice have acceptable area and power, and nor does it mean that setup slacks could be met if OCV derates are being applied to clock paths as well as logic paths. In fact, when OCV derates are considered, it is entirely possible to get into a vicious spiral of increasing insertion delays causing increasingly tight windows, which cause further increases in insertion delay in order to meet the windows, culminating in the situation where it is impossible to make setup slacks positive.

CC-Opt differentiates itself from traditional approaches to useful skew by bringing both clock scheduling and physical optimization together under a unified architecture and basing all decisions on a true propagated clocks model of timing, including inter-clock paths, OCV and clock gate timing. CC-Opt treats both clocks and logic as equally important classes of citizen and understands that in practice either can be the limiting factor on achievable chip speed. CC-Opt must somehow enter the propagated clocks world as soon as possible and then globally optimize both the clock and logic delays according to some coherent optimization objective which can be bounded by either logic considerations or clock considerations.

The close marriage of physical optimization and clock construction, together with knowledge of the side-effects of various decisions in each domain, is the most difficult component of the CC-Opt problem to solve well. But it is also the key enabler for mainstream commercial adoption of CC-Opt. The relaxation of the requirement to balance clocks unleashes significant freedom, but this freedom is commercially useless if it is not exploited wisely and in the context of a full propagated clocks model of timing. Key signs of the failure to exploit this freedom properly are clock trees that are too large, insertion delays that are too long, and significant hold timing problems which result in an unreasonable increase in design area once hold fix buffers have been inserted.

CC-Opt's decisions are always based on a true propagated clocks measure of timing including clock gates, inter-clock paths, OCV derates, and CPPR.

CC-Opt will never close timing at the expense of creating an unreasonably large clock network or an unreasonable number of hold time violations.

Clock Concurrent Design Flow

Inserting CC-Opt into the design flow does not require any changes other than to replace physical optimization and CTS with CC-Opt, and skipping the traditional post-CTS optimization step which becomes redundant: see Figure 13. What is done before clock concurrent optimization and what is done after remain the same. There is no change in timing sign-off or in formal verification or in gate level simulation. No special data structures or file formats are needed, and complex CTS configuration scripts become redundant as there is no longer any need to specify any balancing constraints. There is a potential impact on the magnitude of scan chain hold violations but this can easily be managed by enhancing scan chain stitching algorithms to directly consider hold slacks and not just scan chain wire length. For example, Azuro's Rubix™ CC-Opt tool already includes such a scan chain re-stitching capability.

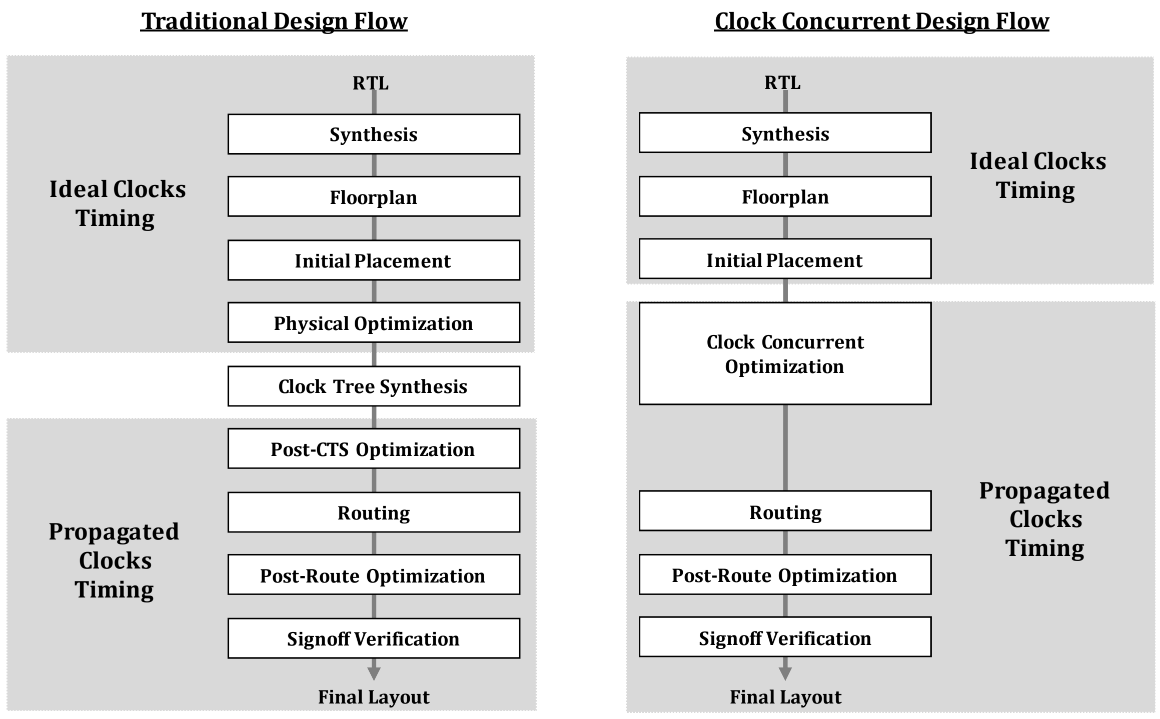


Figure 13: Clock Concurrent Design Flow

Key Benefits of Clock Concurrent Optimization

This section overviews the key benefits that CC-Opt can bring to the digital chip design community.

1. Increased chip speed or reduced chip area and power

Using CC-Opt the maximum possible clock speed is limited by the critical chain and not the critical path in a design. This is a fundamental new degree of freedom to help close timing above and beyond traditional design flows. If the desired chip speed is already achievable without CC-Opt then this same additional degree of freedom can be exploited to reduce chip area or power. At 65nm and below the achievable increases in clock speed can be as much as 20%.

2. Reduced IR-drop

Since CC-Opt does not balance clocks the peak current drawn by the clock network is significantly reduced. In fact it is entirely feasible to extend CC-Opt to directly consider peak current (or some reasonable estimate of peak current) as an optimization parameter and specifically skew clocks and adjust logic path delays to ensure that peak current is controlled to be within a pre-specified limit. At advanced process nodes IR-drop can have a critical impact on timing sign-off and chip packaging cost. CC-Opt unshackles chip designers from the traditional conflict of interest between tight skew being good for timing but terrible for IR-drop.

3. Increased productivity and accelerated time to market

There are two distinct ways in which CC-Opt increases designer productivity and accelerates time to market. The first is due to a lack of any requirement to configure clock tree balancing constraints or manually set insertion delay offsets for timing critical sink pins. For complex SoC designs composing a complete set of balancing constraints can take more than a month, and much of this effort often needs repeating every time a new netlist is provided by the frontend design team.

The second way in which CC-Opt increases designer productivity and accelerates time to market is due to a significant reduction in the number of iterations between the frontend and backend design teams. Many of these iterations are for the sole purpose of asking the frontend design team to manually move logic across register boundaries by changing the RTL. This manual moving of logic is in essence a form of sequential optimization being performed manually and very inefficiently. Since the entire flow must be re-run to incorporate the RTL changes, there is no guarantee that the satisfactory aspects of the post-placement timing picture will persist. Therefore the requested changes may not have the intended benefit. Most of the need to do this manual logic moving is completely eliminated using CC-Opt since it can simply skew the clocks instead. The time saving from these reduced iterations can be many months.

4. Accelerated migration to 45nm and below

The ability of CC-Opt to perform timing optimization is not degraded by the growing clock timing gap. This is because all decisions it makes are directly based on a propagated clocks model of timing. If architected correctly, the motto for CC-Opt is “if I can time it then I can optimize it”. Clock gates, complex clock muxing configurations, OCV derates, CPPR, multi-corner, and multi-mode should all fall out in the wash so long as the timing analysis engine is able to consider them. Without the use of a clock concurrent flow the clock timing gap increasingly cripples timing closure pre to post-CTS, and timing optimization steps downstream from CTS just don’t have the horsepower to recover from the damage. Using CC-Opt, migration to advanced process nodes can happen faster and with significantly less pain.

Conclusions

Clocking lies at the heart of commercial chip design flows and is almost as central to the digital chip design community as the transistor itself. But the traditional assumption that if clocks are balanced then propagated clocks timing will mirror ideal clocks timing is fundamentally broken. Clock gating, clock complexity and on-chip variation are the key industry trends causing this divergence, which we refer to as the clock timing gap. At 40/45nm the clock timing gap can be as much as 50% of the clock period resulting in an almost complete rewrite of the timing landscape between ideal and propagated clocks timing.

Clock concurrent optimization gives up on the idea of clock balancing as both restrictive and unhelpful at advanced process nodes. It merges CTS with physical optimization building both clocks and optimizing logic delays at the same time based directly on a propagated clocks model of timing. This unleashes a fundamental new degree of freedom to borrow time across register boundaries resulting in chip speed becoming limited by critical chains not critical paths.

Under the hood, the key challenge which CC-Opt must tackle is the potential for clock networks to become unreasonably large, and addressing this challenge requires the clock construction algorithms to become very tightly bound with the logic optimization algorithms. The intimate relationship between clock construction and logic optimization is what differentiates CC-Opt from the traditional techniques of sequential optimization and useful skew.

CC-Opt delivers four key types of benefit to the digital chip design community: increased chip speed or reduced chip area and power; reduced IR-drop; increased productivity and accelerated time-to-market; and accelerated migration to 45nm and below.

References

[Cong00] J. Cong and S. K. Lim, “Physical planning with retiming,” in Digest of Technical Papers of the IEEE/ACM International Conference on Computer-Aided Design, (San Jose, CA), pp. 1–7, November 2000

[Fishburn90] J. P. Fishburn, “Clock Skew Optimization”, IEEE Trans. on Computers, vol 39 pp 945–951, 1990

[Held03] S. Held, B. Korte, J. Maßberg, M. Ringe and J. Vygen, “Clock scheduling and clocktree construction for high performance ASICs”, Proceedings of the 2003 IEEE/ACM international conference on Computer-aided design

[Hurst04] A. P. Hurst, P. Chong, A. Kuehlmann, “Physical placement driven by sequential timing analysis.” Proc. ICCAD '04, pp. 379-386.

[Kourtev99] I. S. Kourtev and E. G. Friedman, “Synthesis of clock tree topologies to implement nonzero clock skew schedule,” in IEE Proceedings on Circuits, Devices, Systems, vol. 146, pp. 321–326, December 1999.

[Kourtev99b] I. S. Kourtev and E. G. Friedman, “Clock Skew Scheduling for Improved Reliability via Quadratic Programming”, Proc. ICCAD 1999.

[Leiserson83] C. Leiserson and J. Saxe, “Optimizing synchronous systems,” Journal of VLSI and Computer Systems, vol. 1, pp. 41–67, January 1983.

[Leiserson91] C. Leiserson and J. Saxe, “Retiming synchronous circuitry,” Algorithmica, vol. 6, pp. 5–35, 1991

[Pan98] P. Pan, A. K. Karandikar, and C. L. Liu, “Optimal clock period clustering for sequential circuits with retiming”, in IEEE Trans. on CAD, pp 489-498, 1998

[Ravindran03] A. K. K. Ravindran and E. Sentovich, “Multi-domain clock skew scheduling,” in Proceedings of the 21th International Conference on Computer Aided Design, ACM, 2003

[Xi97] J. G. Xi and W. W.-M. Dai, “Useful-skew clock routing with gate sizing for low power design,” J. VLSI Signal Process. Syst., vol. 16, no. 2-3, pp. 163–179, 1997.

[Xi99] J. G. Xi and D. Staepelaere, "Using Clock Skew as a Tool to Achieve Optimal Timing," Integrated System Design, April 1999.