Clock Tree Power reduction by clock latency reduction

By Sunny Arora, Naveen Sampath, Shilpa Gupta, Sunit Bansal, Ateet Mishra

Abstract

The Current Clock Tree Synthesis strategy used in chips target to build all leaf cells of a clock at the same latency & skew targets. This causes addition of lots of extra clock buffers in the design. Clock tree power contributes nearly 40-45% of the total dynamic power in a chip. Reducing clock tree power will help in reducing the total power. Also, OCV impact, which is proportional to the clock latency, has become a big concern in high frequency design done on shrinking technology nodes. If bunches of leaf cells can be built at a reduced latency, after ensuring minimal impact on timing, a **reduction in average & Dynamic power** can be achieved. Also, **OCV derate impact** will be minimized; reduced latency will also improve OCV effect. We present a method to build leaf cells at lesser latencies.

Current Methodology

Fig 1 shows a typical implementation of clock tree being currently followed. All flops are built at the same latency number, which is decided by the maximum latency in the design, which is C in this case.

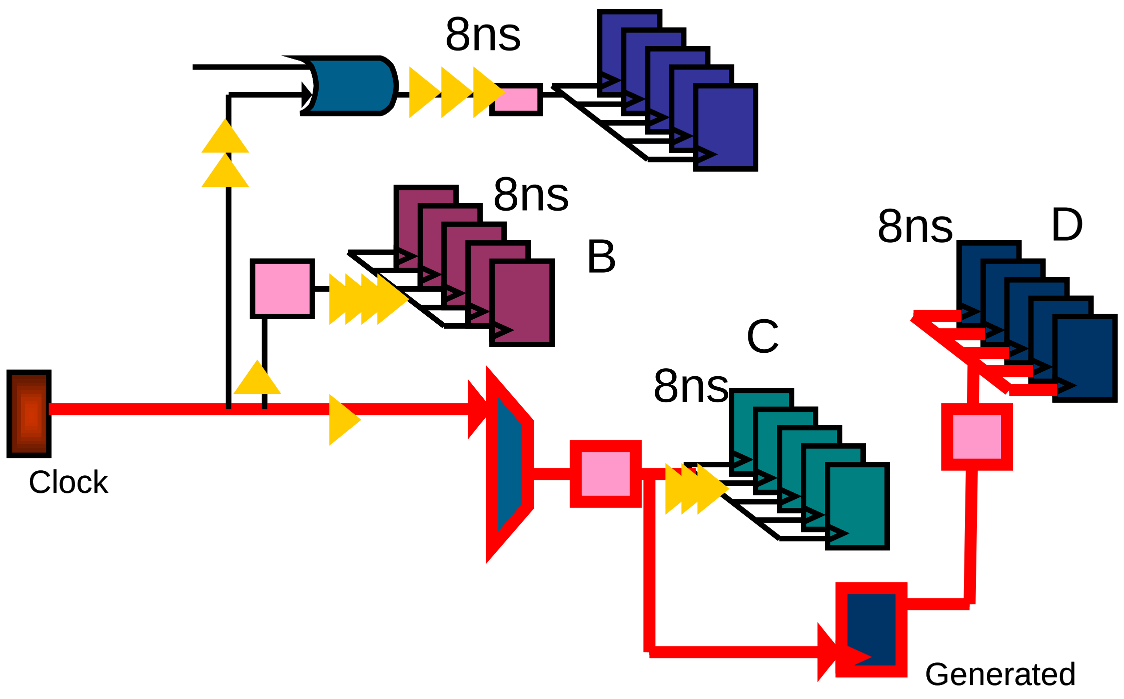


Fig 1 – Traditional Clock Tree Synthesis

This is done mainly for 2 reasons, to avoid hold violations in the design & for simpler implementation. Flops are built at the same latency without even considering whether flops are interacting or not. The design impact is obvious, by the addition of so many clock buffers design becomes power hungry, which is the most critical problem our digital world is currently facing. *(40 -50 % of the total power is constituted by clock tree).*Also, because of simultaneous switching of so many flops, peak power goes very high.

Our Idea

Fig 2 depicts the idea which we are proposing. Our idea is to build the flops at their minimum possible latency, with minimum or no timing impact.

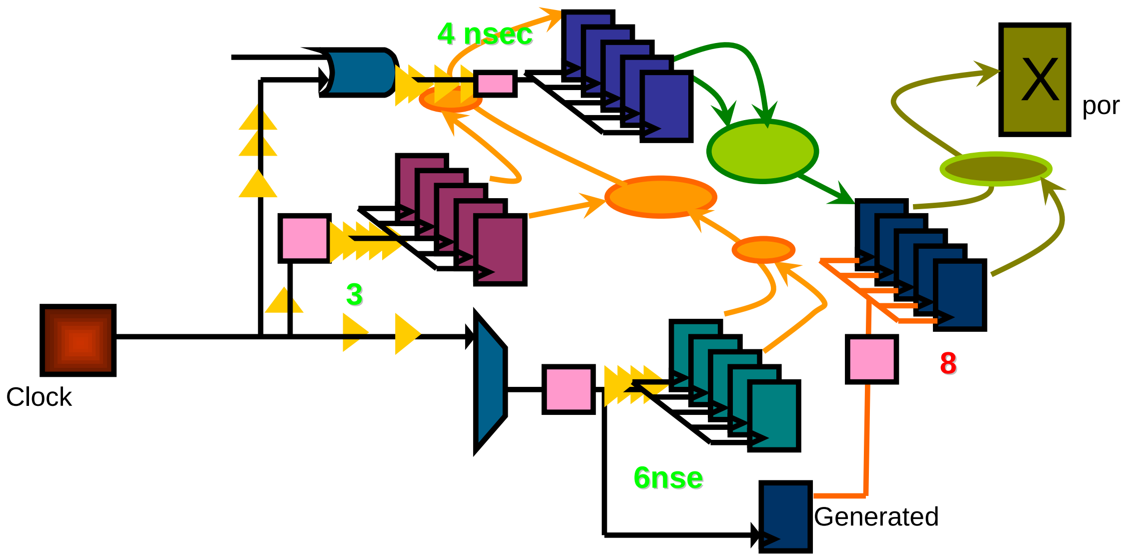
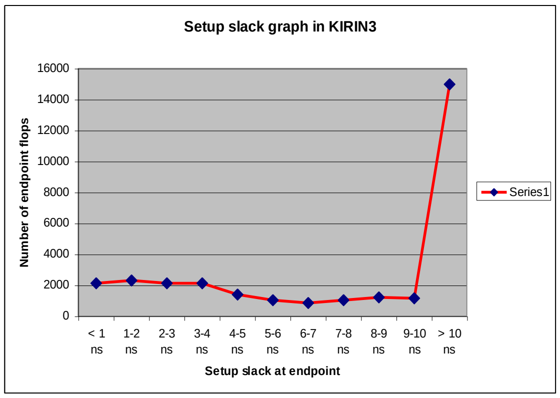


Fig 2 – Our Idea

As can be seen in Fig 2, flops have been built at different, but minimum possible latencies. Unlike the previous implementation, they are not built at the same latency of 8 ns.

Basis of Proposal

Fig 3 & 4 depicts two sets of observation which was made on two SoC designs.



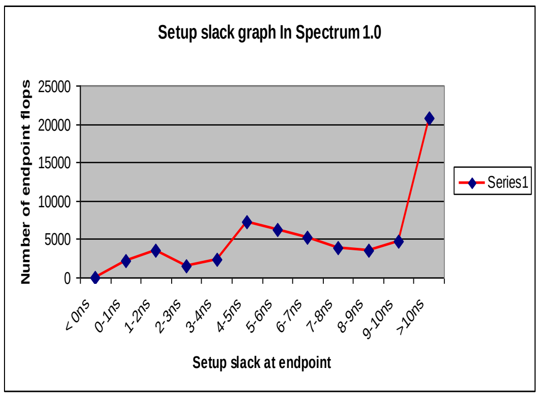


Fig 3

Fig 3 shows the Setup Slack distributions when flops are build at same latencies. It can be seen that most of the endpoints are non-timing critical wrt setup checks. This indicates that there won't be too many setup critical paths to deal with when the flops are built at different latencies.

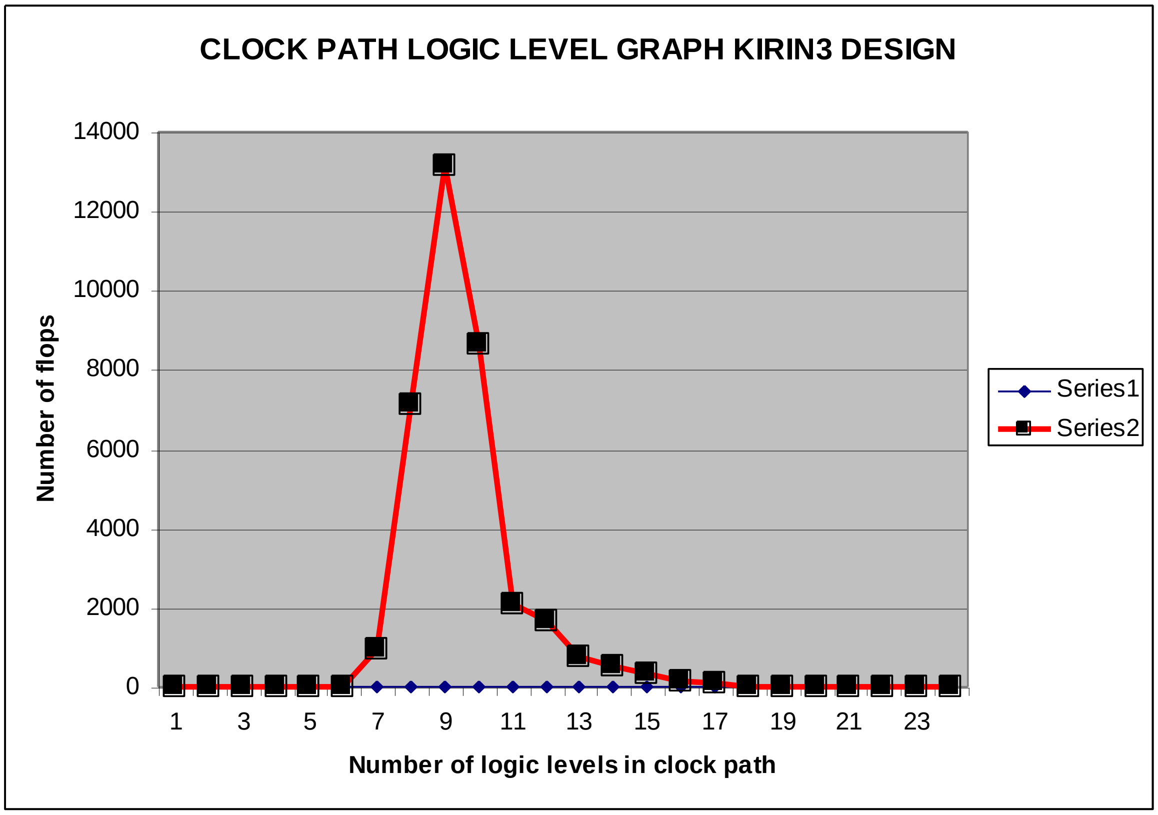


Fig 4

Fig 4 shows the Clock path distribution in terms of number of logic levels in the clock path from source to flop clock pins. As can be seen, there are a large number of flops which have the potential to be built at a much lesser latency. Current clock tree implementations aim to build all flops at the "maximum logic level clock path", which in this case is level 24. It is known fact that the minimum possible latency of a flop clock pin is limited by the number of logic levels which the clock path has. Here logic level means RTL instantiated cells like clock gating cells, muxes etc.

Another observation from Fig 4 is that the flops can be divided into 3 distinct "bins" on the basis of number of clock levels, LOW, MEDIUM & HIGH.

Complexity & Challenges

Fig 5 shows the latency distribution of flops with the conventional clock tree implementation, and our implementation

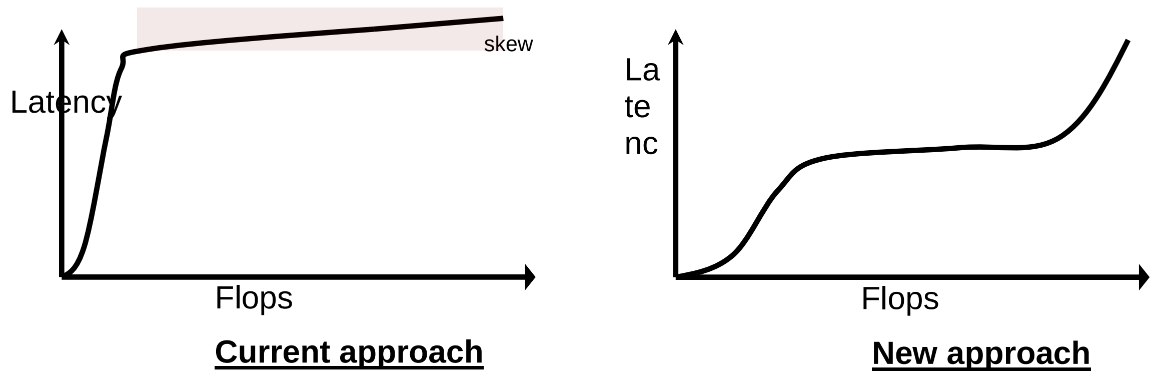


Fig 5 – Comparison b/w traditional & new approach

The advantages are obvious; there is a reduction in Dynamic power, peak power & OCV impact.
But, the challenge is to still meet design timing in terms of setup & hold violations at these flops. Also, we need to come with a criterion to decide the minimum possible latency for each flop in design; parameters being the setup/hold timing criticality of the flop, and the minimum latency at which it can actually be built.

Implementing this scheme/Idea can be very complex. Consider following 5 flops and their interaction. It is very complex to determine the setup & hold dependencies and deriving different latency numbers for each of them.
Also lot of calculations and iterations are required seeing the impact of new latency no. on other interacting flops. The possibility of interaction will increase in factorials (of no. of flops)
In general there are **thousands of flops** in the design. Looking at the complexity, amount of calculation (memory) required, writing an algorithm which will derive flop dependent latency number with its impact (setup & hold) on all the other interacting flops would be an impossible task. We really need an intelligent implementing algorithm which will give us the required benefit with ease in implementation.

Implementation Strategy
To reduce the computational complexity involved, we select a strategy shown in Fig 6.

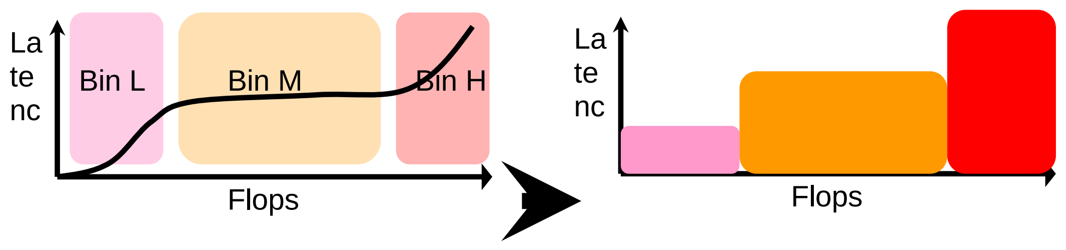


Fig 6 – Implementation idea

The flops are divided into "N" bins on the basis of the minimum latency at which they can be built. N needs to be an optimum number. Higher the number of bins more is the improvement. But, the percentage improvement as we move to higher number of bins reduces exponentially. Also, the implementation becomes tougher. A value of 3 is an optimum number.

Fig 7 shows the implementation flowchart of our idea. It can be implemented after design is timing clean in synthesis & placement, and before Clock tree synthesis. The flops are divided into bins, by looking at the logic in the clock path. In this figure, number of bins are 3. Script1.tcl looks at the setup critical paths, and recursively adjusts the latency numbers of flops to meet setup timing. Script2.tcl looks at the hold critical paths, and recursively adjusts the latency numbers of flops to meet hold timing.

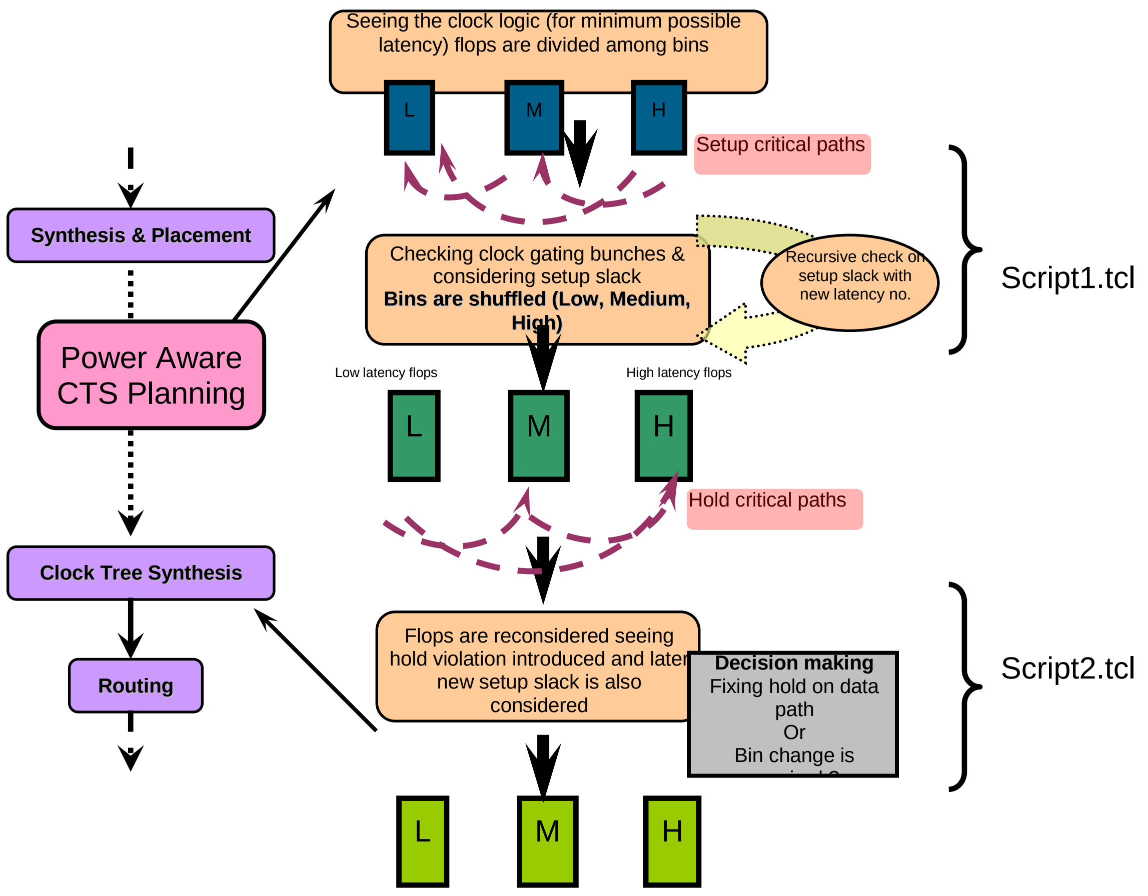

Seeing the clock logic (for minimum possible latency) flops are divided among bins
L
M
H
Setup critical paths
Synthesis & Placement
Power Aware CTS Planning
Checking clock gating bunches & considering setup slack
Bins are shuffled (Low, Medium, High)
Recursive check on setup slack with new latency no.
Script1.tcl
Low latency flops
High latency flops
L
M
H
Hold critical paths
Clock Tree Synthesis
Routing
Flops are reconsidered seeing hold violation introduced and later new setup slack is also considered
Decision making
Fixing hold on data path
Or
Bin change is
Script2.tcl
L
M
H

Fig 7 – Implementation flowchart

Fig 8 shows the algorithm of Script1.tcl. As all flops are built at their minimum possible latencies, the setup critical paths are from the highest latency bin to the lowest one. The 2nd highest bin (Medium bin here) is selected, and setup timing is checked at all the flops in this bin, *as endpoints*. If the timing does not meet, the flop is pushed by the appropriate amount. A recursive algorithm is applied to account for new setup violations which can

come up when a flop is pushed. After finishing a bin, the next lower bin is selected. The process is repeated till all bins are finished.
Whenever a flop is pushed, it can affect timing of 2 kinds of flops. First are flops present in a lower bin. As we are moving from higher to lower bin, this is automatically taken care of when the lower bin is considered. Second are flops in the same bin, which have not been pushed so far. A recursive algorithm is applied to address these flops; setup timing is checked on flops interacting with the pushed flop as startpoint. The flops for which setup does not meet are also pushed. This process is repeated till there all violations are resolved.

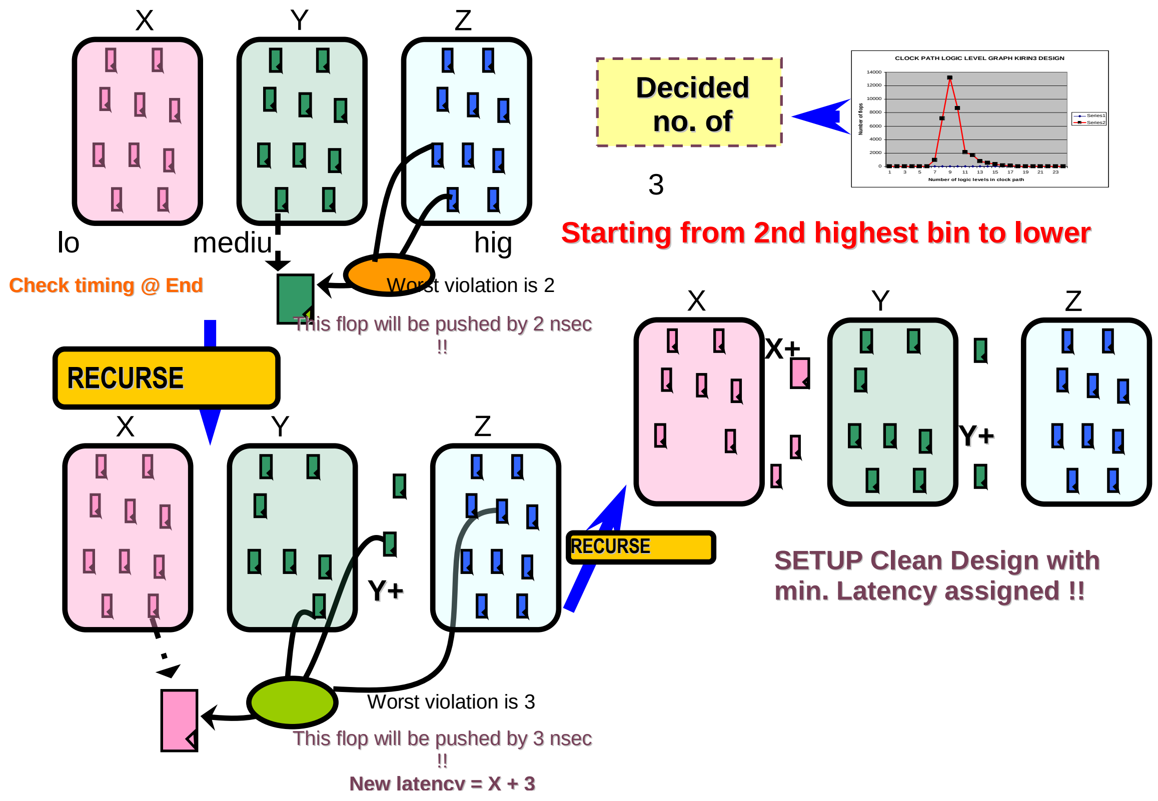

X
Y
Z
lo
mediu
hig
Decided no. of
3
CLOCK PATH LOGIC LEVEL GRAPH KIRIN3 DESIGN
Starting from 2nd highest bin to lower
Check timing @ End
Worst violation is 2
This flop will be pushed by 2 nsec !!
RECURSE
X+
Y+
Worst violation is 3
This flop will be pushed by 3 nsec !!
New latency = X + 3
SETUP Clean Design with min. Latency assigned !!

Fig 8 – Script1.tcl

As shown in Fig 8, in the end we will be left with a setup clean design, with some flops being assigned intermediate latencies.

Fig 9 shows the working of Script2.tcl, which looks at hold violations.

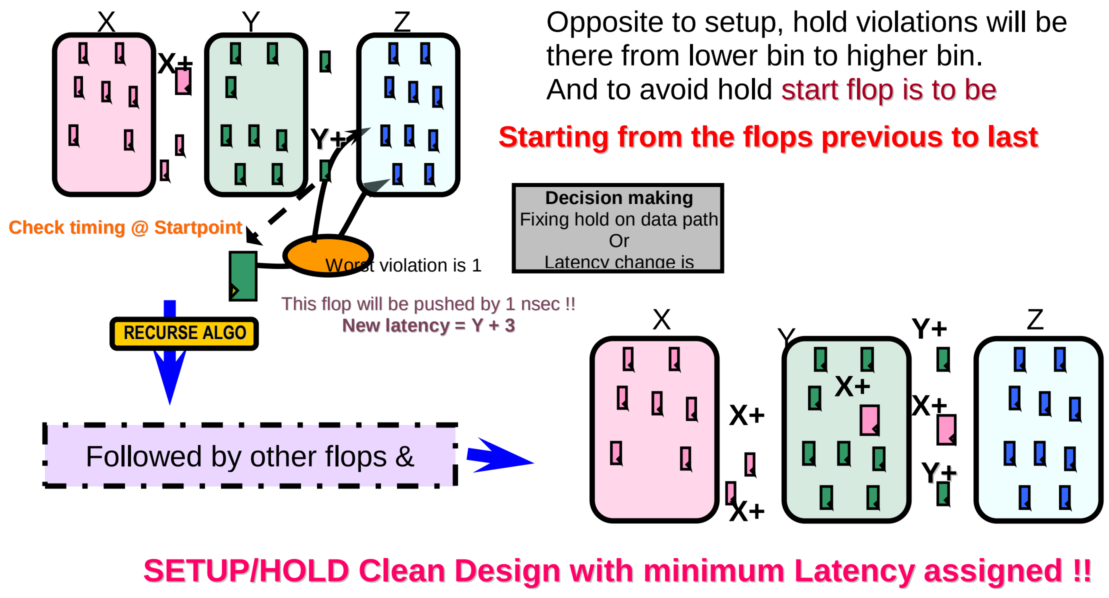

X
Y
Z
X+
Y+
Opposite to setup, hold violations will be there from lower bin to higher bin.
And to avoid hold start flop is to be
Starting from the flops previous to last
Check timing @ Startpoint
Decision making
Fixing hold on data path
Or
Latency change is
Worst violation is 1
This flop will be pushed by 1 nsec !!
New latency = Y + 3
RECURSE ALGO
Followed by other flops &
SETUP/HOLD Clean Design with minimum Latency assigned !!

Fig 9 – Script2.tcl

As opposed to setup, hold critical paths would be present from a lower bin or a lower latency flop to a flop at higher latency. Again, flops which are built at latency just less than that of the highest bin are considered first. Hold timing is checked as these flops as startpoints. If there is a hold violation, a decision is taken whether the hold violation can be fixed by adding buffers in data path, or by pushing the flop. Once all flops are exhausted at the present latency number, the next lower latency flops are selected. A recursive algorithm is also applied here to see if the pushing of flop causes a new hold (flop as endpoint) or setup violation (flop as startpoint).

Results
The idea was implemented on one our designs having around 64K flop.

	Present Approach	Proposed approach	% saving
No. of buffers Inserted	2343	1900	18.9%
Net switch power	31.72mW	28.6mW	9.8%
Inst internal power	8.9mW	6.8mW	23.6%
Total clock distribution power	40.6mW	35.4mW	12.8%

Fig 10 - Results

As can be seen in Fig 9, significant reduction in clock distribution power, and in number of buffers added have been achieved.

Summary

In this paper we have presented a new approach for clock tree synthesis, which is power aware. Unlike traditional strategies, we proposed to build flops at their minimum latencies, while keeping timing in mind. We proposed a method by which the computational complexity involved can be significantly reduced. We also presented the results from on of our designs where the idea was implemented.