PREDICTIVE CLOCK SKEW REDISTRIBUTION METHODOLOGY FOR IMPROVED TIMING QOR

M.Tech. Thesis

By

SANJAY SHARMA

DISCIPLINE OF ELECTRICAL ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY INDORE

JUNE, 2019

PREDICTIVE CLOCK SKEW REDISTRIBUTION METHODOLOGY FOR IMPROVED TIMING QOR

A THESIS

Submitted in partial fulfillment of the requirements for the award of the degree
of
Master of Technology

by
SANJAY SHARMA

DISCIPLINE OF ELECTRICAL ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY INDORE
JUNE, 2019

INDIAN INSTITUTE OF TECHNOLOGY INDORE

CANDIDATE'S DECLARATION

I hereby certify that the work which is being presented in the thesis entitled **PREDICTIVE CLOCK SKEW REDISTRIBUTION METHODOLOGY FOR IMPROVED TIMING QOR** in the partial fulfillment of the requirements for the award of the degree of **MASTER OF TECHNOLOGY** and submitted in the **DISCIPLINE OF ELECTRICAL ENGINEERING, Indian Institute of Technology Indore**, is an authentic record of my own work carried out during the time period from July, 2017 to June, 2019 under the supervision of Dr. Santosh Kumar Vishvakarma, Associate Professor, IIT Indore and Shrikrishna Nana Mehetre, Sr. Engg. Manager, Seagate Technology HDD (India) Pvt Ltd, Pune.

The matter presented in this thesis has not been submitted by me for the award of any other degree of this or any other institute.

Sanjay Sharma
Signature of the student with date
(SANJAY SHARMA)

This is to certify that the above statement made by the candidate is correct to the best of my/our knowledge.

[illegible]
Signature of the Supervisor of
M.Tech. thesis #1 (with date)
(Dr. SANTOSH KUMAR VISHVAKARMA)

Mehetre
Signature of the Supervisor of
M.Tech. thesis #2 (with date)
(SHRIKRISHNA NANA MEHETRE)

SANJAY SHARMA has successfully given his/her M.Tech. Oral Examination held on **24 June, 2019**.

[illegible] 24/06/2019 Mehetre
Signature(s) of Supervisor(s) of M.Tech. thesis
Date:

for [illegible] 02/07/19
Convener, DPGC
Date:

[illegible] 24/6/19
Signature of PSPC Member #1
Date:

[illegible] 24.6.2019
Signature of PSPC Member #1
Date:

ACKNOWLEDGEMENTS

I would like to take this opportunity to express my extreme gratitude to my research supervisor **Dr. Santosh Kumar Vishvakarma,** Associate Professor IIT Indore and **ShriKrishna Nana Mehetre** Sr. Engineering Manager at Seagate. At many stages in the course of this research project I benefited from their advice, particularly so when exploring new ideas. Their positive outlook and confidence in my research inspired me and gave me confidence.

The many hours that I spent at the college have been very stimulating and enriching, and thanks to the wonderful people that I have been privileged to interact with. Particularly, I would like to thank **Mr. Mahesh Kumawat ,** Phd. Scholar IIT Indore, **Apoorv Srivastav** and **Himanshu Soni** for the great discussions that we had in the lab and for useful comments that improved my work significantly.

I would like to thank the great people of the **Seagate** community. The divine inspiration from the Tech Events and study session gave me the confidence and power to stand up against the difficulties that I faced in the course of my Project. At last, I would like to thank **IIT Indore** for providing the essential research facilities for my research work. I gratefully acknowledge the contribution of all the faculty members and staff of Electrical Engineering for their help and support. I would also like to express my sincere gratitude to Ministry of Human Resource and Development (MHRD), Government of India for providing the Teaching Assistantship during the course of M.Tech.

DEDICATION

“This thesis work is dedicated to my uncle Rajesh Kumar”

Abstract

The functionality of a digital IC depends upon whether its design meets the required timing specifications or not. Though meeting these timing requirements remains a priority throughout the physical design flow but the final design step which specifically deals with meeting these requirements is timing closure.The conventional way of meeting these timing specifications is by adjusting the data path delay. Data path is made faster to fix the setup timing violations. Though this methodology is able to fix setup violations but it has a disadvantage as it increases the area and power consumption of the design. This technique doesn't uses the positive slack available in the design , so the available positive slack becomes a wastage during timing closure and this can also limit the speed of the design. During timing closure, when such available slacks are recovered and used, then use of standard cells having low threshold voltage is also reduced. Hence distribution of such positive slack improves overall system performance. The work presented in this thesis talks about an algorithm which predicts the positive slack available across the design during synthesis and then redistribute these slack values across timing critical paths to remove their timing violations. These skew values are later used during physical implementation of design so that placement of cells is done in accordance with these skew values. This also helps the design to operate at high frequency. The results shows better timing QoR as we see reduction in total negative slack, worst negative slack and also in the number of violated timing paths. Runtime increases compared to the conventional methodology but not by a significant amount.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACRONYMS

WNS – worst negative slack

TNS – total negative slack

NVP – number of violated timing paths

DC – Design compiler

ICC2 – IC Compiler 2

PT – Primetime

PnR flow – placement and route flow

QoR – quality of result

RTL – register transfer logic

CTS – clock tree synthesis

ECO – Engineering change order

VT – Threshold voltage

G.tech – generic technology

SPEF – standard parasitic exchange format

R,C – resistance, capacitance

R2R – register to register timing path group

HVT – high threshold voltage

SVT – standard threshold voltage

LVT – low threshold voltage

Chapter 1

Introduction

Most of the digital electronic systems in today's time employ digital IC's as one of their key components. An IC can implement a simple logic gate to a complex design like a CPU processor. Depending upon the application of digital system, its circuit formulation takes place. Digital system design in today's time has three things in focus:-

- First one is the area occupied by the design should be as low as possible i.e. the circuit design should be compact.
- Second one is the performance of the design in terms of speed should be as optimum as possible i.e. circuit should be able to operate at higher frequencies.
- Third one is the power consumption, high speed circuit always has issue in terms of leakage power, so with speed power should also be kept in mind

Digital IC design has a standard flow of design steps which is called digital design flow or physical design flow. This flow comprises of basically two individual standard flows one is logic design which is also called as frontend design and other one is physical design which is also called backend design. The logic design includes specifying the basic functionality of the design in terms of a HDL (hardware description language) whether its VHDL or Verilog or System Verilog , converting that specification into an RTL (register transfer logic) code and then doing synthesis of that RTL code to convert it into a gate level netlist that contains standard cells, macros, input- output pads and their interconnection to each other. In physical design, the netlist along with some other inputs (like timing constraints, standard cell libraries, tech file etc.) are given as an input to the physical implementation tool and after going through a standard flow of design steps like floorplanning, partitioning (generally done at the top level itself), power planning, placement, CTS and routing. Different sanitary checks are done before and after each step to verify the design implementation. After routing, if needed ECO is done, design testing is done and then finally the design is taped out in terms of GDSII file. Doing timing analysis at the end of certain design steps is also a very important part of physical design flow.

The performance of a digital system such as "system on chip" depends upon the clock frequency at which it operates. Systems operating at higher frequencies are better in terms of overall performance. Continuous advancements has been made in design technology for the fabrication of such high performance systems. The design technologies are scaled down rapidly over the past few decades so as to fabricate IC's that can operate over the frequency range of hundred's of MHz to GHz. But one of the critical issue faced by the physical design engineers while designing the layout of such high performance systems is meeting the required timing specifications. These specifications are necessary to met otherwise the digital design will not work properly. Meeting these timing requirements becomes more and more difficult as the system clock frequency increases. So improving the timing quality of design remains a key objective across entire design flow. So timing closure becomes a very important part of VLSI design flow. The timing optimization is done both during the synthesis as well as during physical implementation of design.

One of the conventional way of doing timing optimization during synthesis is adjusting the delay of the data path so as to remove timing violations. To remove setup violations, the delay of the data path is reduced by replacing the standard cells having high VT with the one having low VT . This technique successfully removes the setup timing violations but the issue is increase power consumption because of the low VT cells. Also in this case, the positive slack available in design is not utilized so that becomes a kind of wastage at the time of timing closure during physical implementation of design. This also limits the capability of that design to operate at higher frequency. Also this increases the area of the overall design.

The other method to deal with this issue is the use of skew available in design. Generally, clock signal is considered ideal during RTL synthesis, this consideration works for low speed designs but this can cause issues for high speed designs. Some skew should be considered while doing synthesis as it brings synthesis closer to actual physical implementation of clock during CTS. Skew values are calculated during physical implementation and are fed back to the DC flow and hence entire design flow runs again. This increases the overall runtime. Skew numbers can be calculated after synthesis as well and again running the DC flow. But as the data path is already optimized during first synthesis , this limits the overall effect of skew adjustment.

This thesis work describes a methodology that redistributes the useful skew available across the design during synthesis itself to improve timing of the design. So there is no feedback loop from implementation stage and hence no need of re-synthesis in this case. The skew numbers calculated during synthesis are fed to the placement and CTS design steps as part of timing constraints to effect the standard cell placement and timing optimization during CTS to improve timing of the overall design. This also reduces area required and also improves power consumption of design. Upcoming chapters will talk about physical design flow, the build up to the problem statement, proposed skew adjustment algorithm in detail and also a detailed analysis and discussion is on results and final conclusion along with possible improvements that can be made to the algorithm in the coming future.

Appendix contains some tcl proc’s that are used in useful_skew_search script. These proc’s helps the useful_skew_proc to perform smaller functions like finding number of violated paths, getting information like startpoint, endpoint and violated slack value and arranging them in decreasing order of violated slack value in a list, updating the list with new values e calculating updated slack value available for any timing path etc.

In last manuals for various implementation and analysis tool, books and articles that are referred during this thesis work are mentioned.

Chapter 2

Review of Past Work and Problem Formulation

2.1 Standard Synthesis Flow:-

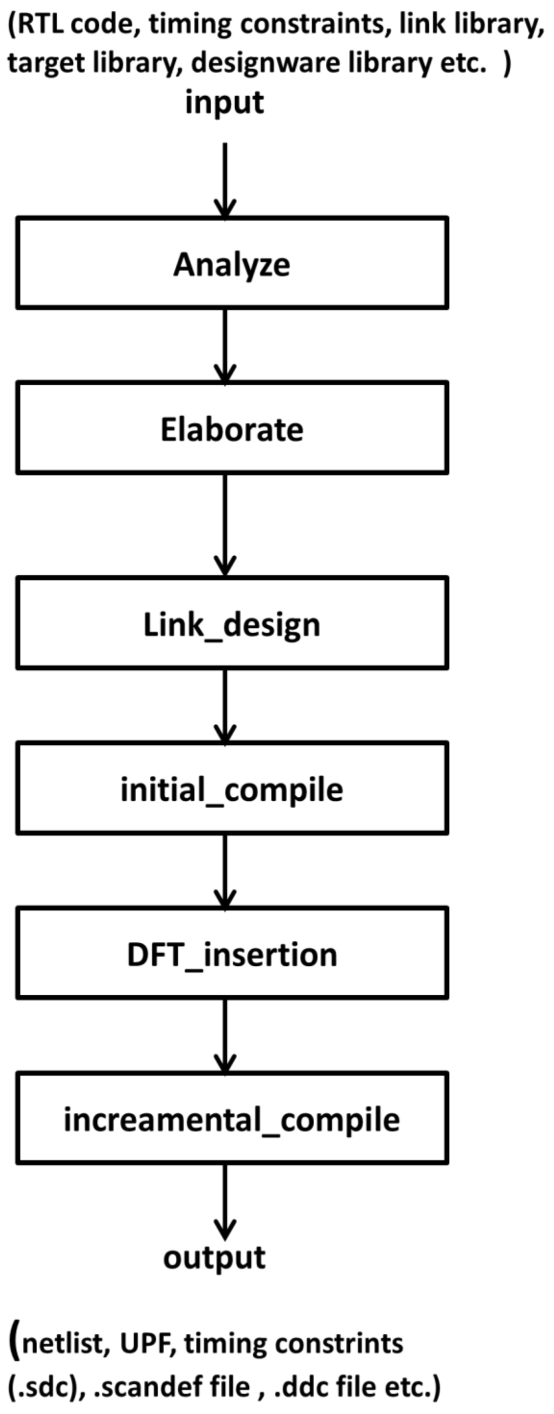
(RTL code, timing constraints, link library, target library, designware library etc.)
input
Analyze
Elaborate
Link_design
initial_compile
DFT_insertion
increamental_compile
output
(netlist, UPF, timing constrints (.sdc), .scandef file , .ddc file etc.)

Fig. 2(1) Standard Synthesis Flow

Synthesis is done to convert the RTL code to gate level netlist. The netlist contains logical functionality of design. It contains standard cells that may be a combinational gate or a combinational logic like adder, subtractor or a sequential logic like latch, flip-flop, registers etc along with their interconnection.

Brief description of the steps mentioned in above figure is:-

1. **Analyze:-**

   Design compiler reads the RTL code and checks if all the syntaxes written are correct or not. Also checks whether the operands used in a syntax belongs to correct datatype or not.
2. **Elaborate:-**

   Designs to be synthesized are hierarchical in nature. There hierarchies are explored in this step. In elaboration, analyzed RTL files are mapped into technology independent G.TECH cells. Arithmetic operations are converted into adders, subtractors etc. with the help of designware libraries.
3. **Link design:-**

   Here all parts of design are linked and all references are resolved.
4. **Compilation:-**

   In this step the design is compiled to perform a number of optimizations like timing, area and power optimizations. These optimizations are done based on design constraints feed to the compiler.
5. **DFT insertion:-**

   Since after physical implementation of the design, it needs to be tested, for that reason, scanchains, clock gating etc. are inserted into the design. During this step, timing , area, power constraints are also fed to compiler.
6. **Increamental Compile:-**

   Design can be compiled any number of times depending upon whether the timing, area and power requirements are met or not. In this step, we can use a number of optimization techniques to improve timing. If only area or power optimization is to be done, that is also possible.
7. **Write out:-**

   A number of files are written out during this step as output of synthesis:- Netlist, .upf file, scandef file, .v file, .ddc file and also a number of timing constraints files.

2.2 Objective of synthesis:-

The main objective of synthesis is to provide the best possible seed i.e. netlist for the PnR flow. This can only be done if degree of correlation is high between the synthesis and PnR tool. Providing aspect ratio, flooplan etc. physical design constraints and skew values helps in achieving this correlation.

2.3 Conventional timing optimization in synthesis:-

Timing optimization in case of synthesis simply means removing the timing violations present in the design. The conventional synthesis methodology improves by replacing the high VT cells with low VT cells in the data path. So whatever optimization is done that is done only in data path. This generates two issues, first one is , using low VT cells or high speed cells in place of slow or high VT cells increases leakage power. Second issue with this approach is the useful or positive skew present in different timing paths in a design remains unused which can limit the ASIC design CPU's to operate from higher frequencies. Also in conventional DC flow, the clock skew is ideal that is zero. So all the flip flops, latches present in the design gets triggering clock edges at the same time. This is not a good assumption as during actual physical implementation, there is always some clock skew in the design. This approach hence contributes to increase in total number of timing violations. Hence to increase the correlation between the synthesis and PnR tool, it is very important to consider some skew during synthesis, that can increase the correlation between two tools. Hence timing closure becomes a bit easier compared to the conventional approach of design flow.

2.4 Conventional Timing Closure:-

This step is done after completion of the PnR flow i.e. after routing. After routing, the netlist generated from routing along with timing constraints are given to the Primetime. SPEF files (Standard Parasitic Exchange Format) are generated using Star RC tool. These files contains parasitic information i.e. R, C extraction information which is used by Primetime or PT for calculating net delays etc. PT generates both QoR and max/min timing reports. QoR tells about overall timing quality of the design. Max timing reports helps in analysing the data and clock path to fix setup violations and same is done using the min timing report for fixing the hold violation. Now, to fix setup violations, the data path has to be fast, to fix hold violations, data path has to be slower.

Fixing of setup violation can be done in two ways:-

1. Faster the data path
2. Delay the clock path of endpoint or faster the clock path of startpoint

Conventional timing closure talks about first approach, where sizing of cells is done to make a path faster or slower. But its consequences are in terms of area and power penalty as explained earlier.

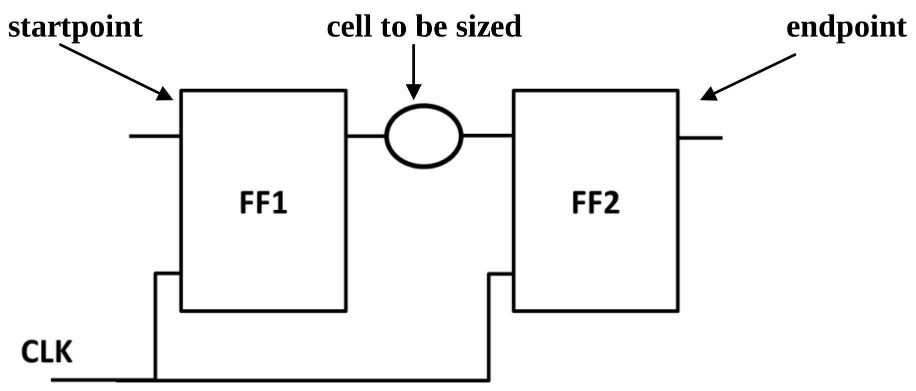

Fig. 2 (2) typical R2R timing path

In above figure, FF stands for flip-flop. The positive slack available in this case is wasted. And even if this positive slack is used here, since synthesis is already done so the effectiveness of using the second approach of clock path optimization will be very less. Hence utilizing useful skew available during the synthesis itself becomes important. Skew value generally which are given during synthesis are calculated after running the complete design flow right from synthesis to PnR flow and then calculating the skew numbers from the CTS stage or the routing stage. This approach helps in improving the timing QoR of the design but

again this approach also has issues that affects the overall performance and runtime of entire design flow.

2.5 Useful skew:-

Useful skew is that skew which is intentionally introduced in the design to improve its overall timing quality. It may completely remove a setup or hold violation or can reduce the value of violation. Both way, we are improving the timing QoR.

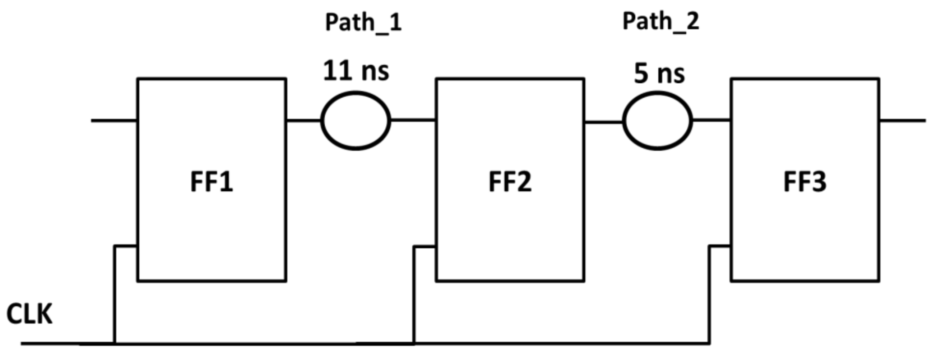

Fig. 2 (3) timing path without useful skew

Suppose the clock period is of 10ns . So in case of fig. 2(3) the data should be available before or at 10ns for register FF2. But we can clearly see that at the first data path path_1, delay is of 11ns. Hence there is a timing violation of 1ns. But in case of path_2, delay is of just 5ns. So data become available at input of FF3 register at 5ns. So there is a positive slack of 5ns. This positive slack can be used to remove timing violation at path_1 as shown below:-

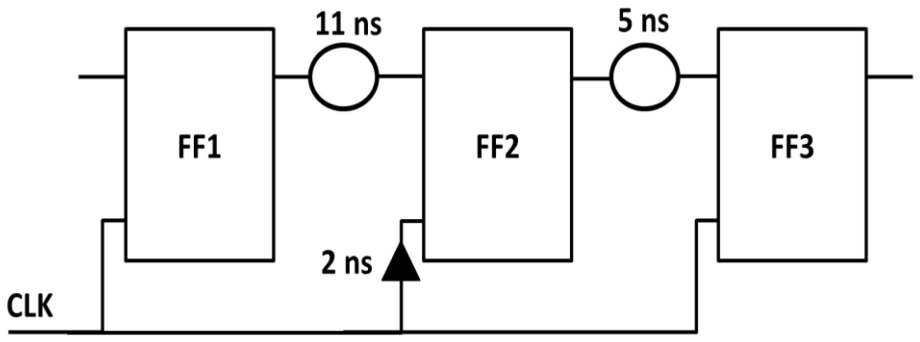

Fig. 2 (4) timing path with useful skew

In this case of above figure 2(4) , now along with clock delay of 10ns , an additional delay of 2ns is added with the help of a buffer. Now net clock path delay becomes 12ns and while the data path path_1 delay remains 11ns. Hence data at the same FF2 register is now available 1ns earlier to the clock edge which will now come at 12ns. So this path now has a positive slack of 1ns. Now if we look at data path path_2, the buffer delay gets added to the data path delay of path_2, so the net path delay becomes 7ns for path_2 but still as the clock edge will come at 10ns. Hence still there is some positive slack of 3ns.

So, it is now clear that we can remove a timing violation by utilizing the useful skew available in the design. As there is no cell sizing, so leakage power can also be reduced.

2.6 Conventional design flow with useful skew:-

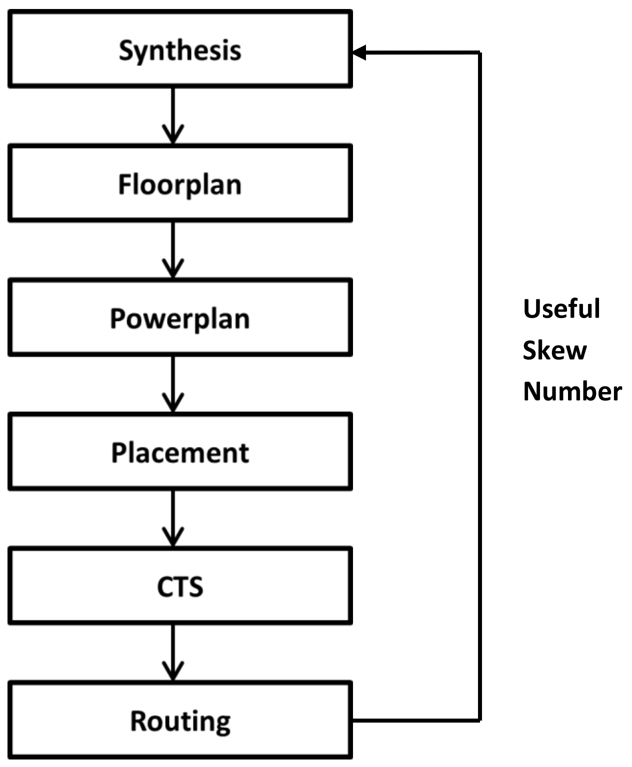

Fig. 2(5) Conventional PD flow with useful skew

In conventional design flow, the skew calculation is done after CTs or routing. Then those skew numbers are fed back to the synthesis tool and again the entire flow needs to run till routing to see the effect of applied skew on timing. But the issue with this approach is that synthesis is already done one time before the skew calculation, so the data paths during timing optimization are already optimized by the design compiler. So the net effect of using the useful skew methodology reduces. For already optimized data paths , cell sizing is already done so leakage power will increase for such timing paths and also this will effect area utilization after completion of timing closure during signoff. Another disadvantage of this approach is design flow will run two times hence runtime increases. But the methodology which is being proposed in this thesis work, doesn’t need any feedback of skew numbers during synthesis, improves runtime of the design flow, final design will take less area and will also consume less power. Next chapter will talk about the proposed methodology in detail.

Chapter 3

Proposed Methodology

3.1 Proposed Design Flow:-

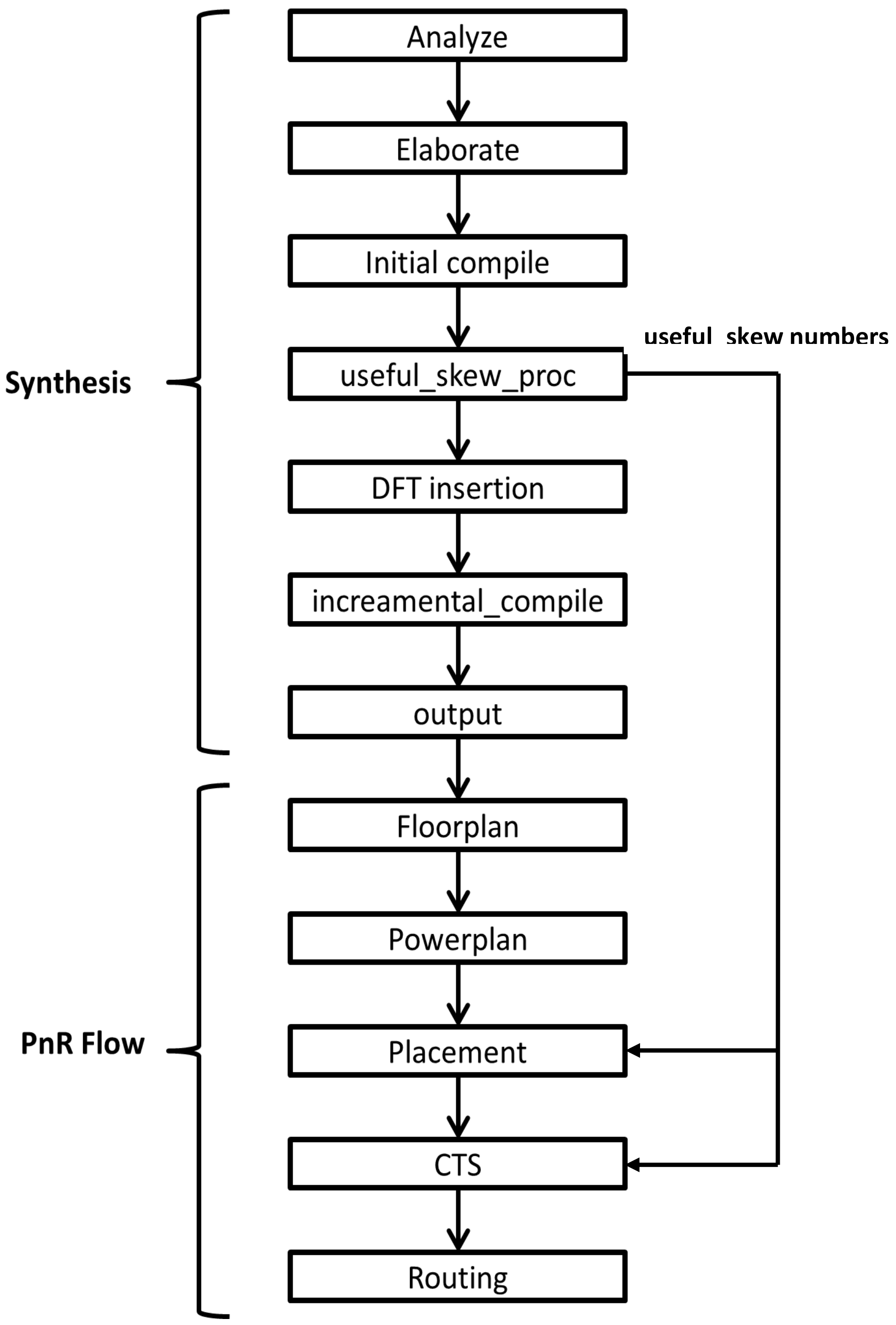

Fig. 3 (1) Proposed Methodology

The design flow shown in above figure is very similar to the conventional design flow except the introduction of useful skew calculating proc after initial compile during synthesis. What this tcl procedure doing is that after initial compile is completed by synthesis tool, it calculates the positive slack available for the violated timing paths (where setup violation has occurred), and uses that calculated useful skew to remove timing violations. Hence the synthesis tool has now skew numbers to play with for timing optimization. So instead of datapath optimization , now the tool will look to optimize clock path based on the given skew. Hence there will be no cell sizing i.e. no replacing of low speed cells with high speed cells. So, the issues such as increase in leakage power that contributes to increase in overall power consumption will get eradicated. The overall area requirement of the design also reduces. Since the positive slack available is now being utilized, hence the issue caused by this positive slack when if not utilized can limit the frequency of operation of a CPU of a ASIC design get resolved. Apart from this, one can see that there is no feedback from CTS OR routing stage to synthesis, so it is a unidirectional flow, no need of re-spinning the synthesis in this case. Hence runtime of the overall design flow improves which is another added advantage with this methodology. So we are getting better timing QoR, improvement in power consumption and area requirement along with the improved runtime. What else can be asked from a design flow. The challenge that comes with this methodology is the prediction of useful skew during synthesis, because from the diagram we can see that the calculated skew numbers are fed as timing constraints during placement and CTS. Since placement and CTS are physical implementation stages, so whether actually that much skew will be available there or not, so we have to be very careful while predicting the useful skew. Now why skew calculation is done after initial compile only and not before because during initial compile only the mapping of generic technology cells to a target technology happens. Skew calculation can be done after DFT insertion as well but then improvement in timing will be less because then violation due to DFT insertion will also comes into the picture and our motive is to improve timing for “functional” mode and not for “mbist” mode. Apart from this, calculated useful skew is fed to the “placement” stage because we want the placement of standard cell to be in accordance with the skew numbers. Also these numbers are fed during CTS as well because during CTS, clock tree is formed, and hence for affecting the clock tree formulation, we have to provide this skew information to the PnR tool. That’s how, correlation is achieved between the synthesis and PnR tool which makes the digitl design efficient.

3.2 Details of proposed work:-

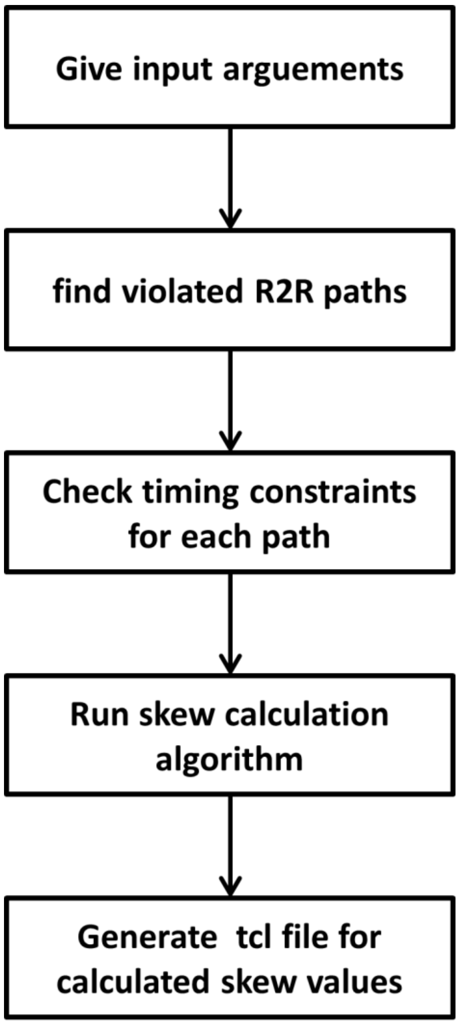
Give input arguements
find violated R2R paths
Check timing constraints for each path
Run skew calculation algorithm
Generate tcl file for calculated skew values

Fig. 3(2) Block diagram of useful_skew_proc

The above diagram explains the basic flow of skew calculating procedure. The first step is to provide input arguements. The runtime of the proc script and timing quality of the design depends upon the values of these arguements. Second stage is to find out the information about violated R2R timing paths as this script focuses on improving the timing QoR for Register to Register paths only. The next step is to check whether the startpoints and endpoints for each violated R2R paths are constrained or not because tool can only optimize constrained timing paths so this is a very important step. Next step is to search for useful skew based on value of violation for each path. So, first the proc will look for required skew in first order and if that skew not enough, then skew will move towards second order search. Finally, a tcl file is generated that contains the all calculated skew values, that skew is sourced by synthesis tool and also during placement and CTS by physical implementation tool.

3.2.1 Input arguements for useful_skew_proc:-

These input arguements are neede so that the user can control the timing quality and runtime of the design according to his requirements. So these arguements makes the procedure much more flexible. Following are the input arguements :-

- **target_slack** – violated slack value should become less than or equal to this value after skew adjustment.
- **target_slack_for_borrow** – lowest value of violated slack for which slack can be borrowed.
- **slack_to_borrow_from_max_path** – from the available positive max-path slack value, how much percentage of slack can be borrowed.
- **slack_to_borrow_from_min_path** – from the available positive min-path slack value, how much percentage of slack can be borrowed.
- **maximum_iteration** – maximum number of times that proc can run, this is done to avoid infinite loop.
- **max_path_margin** – minimum positive slack that must be available at the sequential cell after skew adjustment in case of max-path.
- **min_path_margin** – minimum positive slack that must be available at the sequential cell after skew adjustment in case of min-path.

3.2.2 Calculating the updated_timing_path_slack:-

Whenever a skew value is retrieved, if that skew value is applied at the that time only when it is obtained, then we don't need to do anything to calculate updated slack as the synthesis tool will automatically update the timing and will calculate next skew value with updated timing. But since timing update after applying a skew value takes its own time, so this increases the runtime of skew calculating proc drastically. To avoid this problem, we can write a tcl procedure which will calculate the updated skew value based on previously calculated skew. So the tool doesn't need to calculate the updated slack after updating the timing hence, runtime will reduce drastically for the useful_skew_proc. Algorithm given below explains the updated skew calculating procedure:-

- Using report_timing calculate the **slack_value**.
- Check if startpoint is already delayed or early or no slack applied till now:-

```
if { startpoint delayed } {
    slack_value == slack_value – applied_skew
} elseif {startpoint early} {
    slack_value == slack_value + applied_skew
} else {
    slack_value == slack_value
}
```

- Check if endpoint is already delayed or early or no slack applied till now:-

```
if { endpoint delayed } {
    slack_value == slack_value + applied_skew
} elseif {endpoint early} {
    slack_value == slack_value - applied_skew
} else {
    slack_value == slack_value
}
```

Note:- The above written algorithm also helps in finding the useful for startpoints and endpoints.

3.2.3 Detailed skew adjustment flow:-

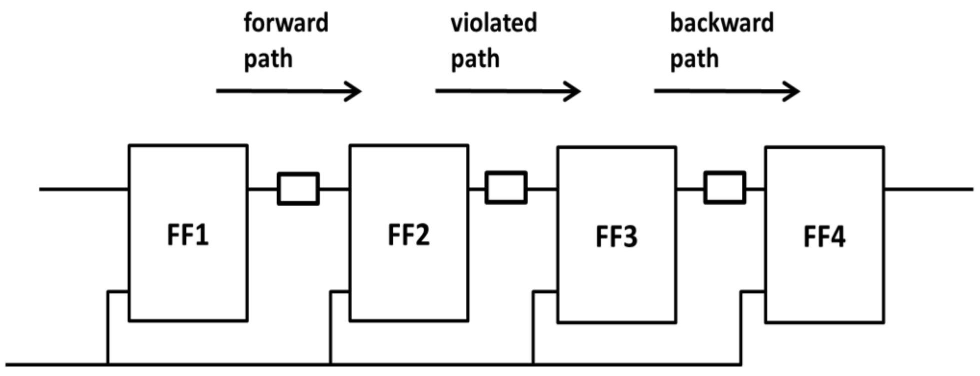

Fig. 3 (3) Path type

In above figure :-

- **Forward Path:- Register to startpoint of violated path**

- **Backward Path:- Endpoint of violated path to register**

Here is the complete flow sequence of skew calculation:-

Step_1: Assign values to the input arguements of useful_skew_proc as per required timing QoR.

Step_2: Filter out all violated reg_to_reg timing paths:-

- Create a R2R path group with "group_path" command whose source should be clock pin of a register and should end at data pin of a register.
- Use "report_qor" to find out total no. of violated R2R timing paths
- Use "report_timing" to find out startpoint, endpoint and violated slack value for each such timing path.
- Prepare a list whose each row has startpoint, endpoint and violated slack value of a particular timing path. List should be prepared in decreasing order of slack value.
- Startpoint and endpoint should have valid timing constraints, this is must as unconstrained timing paths can't be optimized by the tool.

Step_3: check for unconstrained forward and backward paths and categorize them in one of the four possible scenarios mentioned below:-

- Both forward and backward paths doen't have valid timing constraints
- Only forward path has valid timing constraints.
- Only backward path has valid timing constraints.
- Both forward and backward path has valid timing constraints.

Step_4: Based on one of the timing constraint scenario found in step 3, the proc decides which kind of skew adjustment can be done as given below:-

- If both startpoint and endpoint doesn't have valid timing constraints then there is no meaning of skew adjustment as tool can't optimize unconstrained paths.
- If only the startpoint has valid timing constraints then skew calculation can be done only on the forward path not backward path.
- If only the endpoint has valid timing constraints then skew calculation can be done only on the backward path not forward path.

- If both startpoint and endpoint has valid timing constraints then skew calculation can be done on both forward and backward paths.

Step_5: Start the skew search algorithm. This algorithm searches for useful skew depending upon the required skew value in two ways:-

- First_Order_slack_redistribution
- Second_Order_slack_redistribution

The overview of this skew search algorithm is given with the help of if_else condition below:-

```
If  { required_slack <= startpoint_slack , endpoint_slack
                                  and sum_of_both_slacks } {
      Run  first_order_slack_redistribution
} else {
      Run second_order_slack_redistribution
}
```

3.3 First_order_slack_redistribution:-

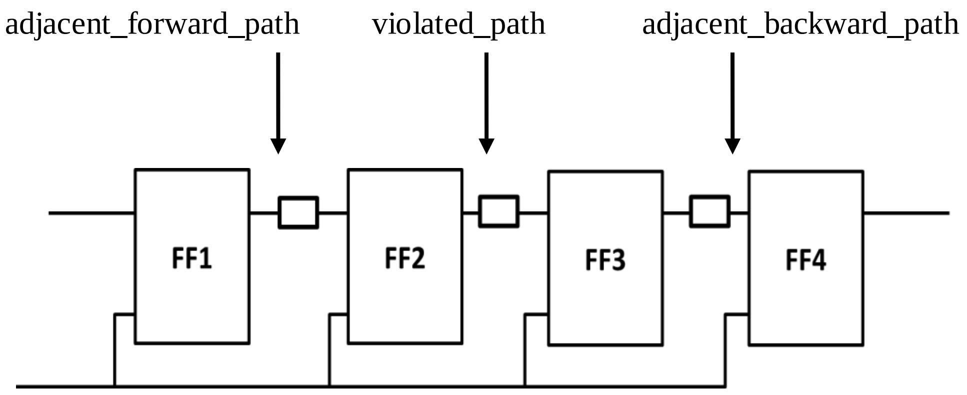

Fig. 3(4) first_order_slack_redistribution

First order slack redistribution means, the algorithm will look only on the adjacent_forward_path and adjacent_backward_path to the violated timing path. If positive_slack of any of the two adjacent_path or sum of their individual slack value mets the required slack value, then there is no need of going towards second order skew adjustment. Hence the skew search for such a timing path will stop at the first order itself.

Following are the cases covered by first_order_skew_redistribution:-

Case_1: required_slack <= endpoint_positive_slack

- **Delay the endpoint by required_slack**

Case_2: required_slack <= startpoint_slack

- **Early the startpoint by required_slack**

Note: Case_2 also covers one more scenario that if no slack is available at the endpoint side.

Case_3: required_slack <= sum of startpoint and endpoint positive slack

- **Delay the endpoint by positive slack available**
- **remainng_slack_required = required_slack – endpoint_slack**
- **Early the startpoint by remaining_required_slack**

If we can see, first we try to delay endpoint then if neede early the endpoint.

This is to avoid worsening of hold violations as useful skew effects hold violations exactly opposite of setup violations.

3.4 Second_order_slack_redistribution:-

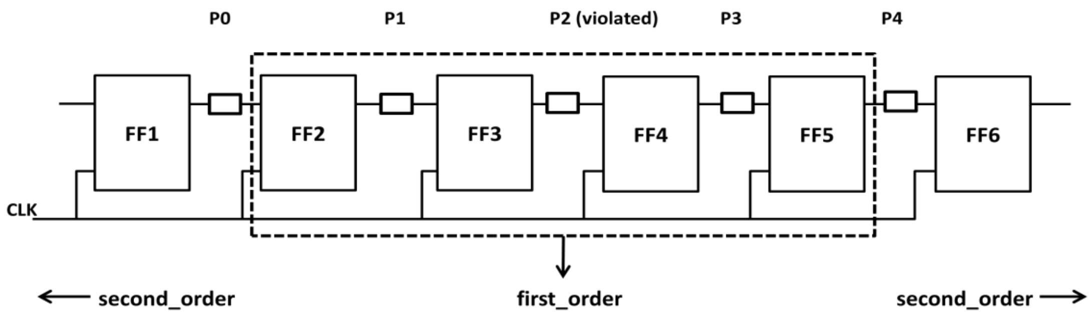

Fig. 3(5) second_order_slack_redistribution

Skew search algorithm uses second_order_slack_redistribution when no skew is available in case of first order search or the retrieved useful skew is not enough to meet the required slack value. Second order means the algorithm will go deep into the endpoint side or startpoint side along the chain of registers to met the required slack. It will adjust skew values not only for the adjacent forward and backward paths but also the other forward backward paths. In above figure, P2 is violated timing path, P1 and P3 timing paths belongs to the first order search while P0 and P4 belongs to the second order search.

E.q. Suppose in fig. 3 (5), P2 path is violated by 0.5ns,

P1 has positive slack of 0.1ns

P3 has positive slack of 0.2ns

P0 has positive slack of 0.05ns

P4 has positive slack of 0.15ns

Since total positive slack of first order paths is 0.3ns, still there is violation of 0.2 ns while the second order paths have total skew of 0.2ns , hence by including useful skew values from second order paths, total positive slack becomes 0.5ns and hence required slack value is met.

3.5 Issue with Second_Order_Slack_Redistribution:-

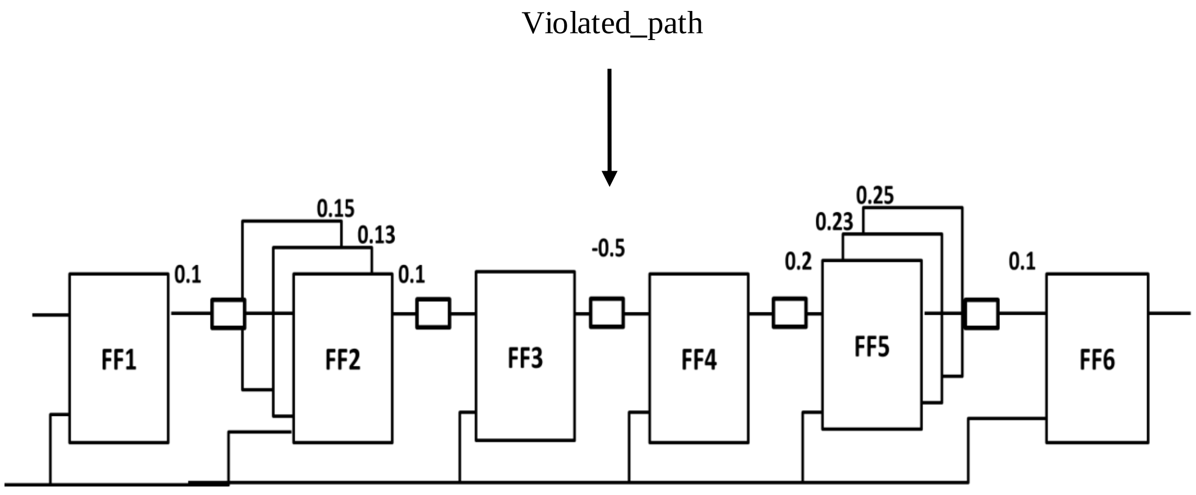

Fig. 3(6) issue with second order

Consider the example shown in above fig. 3(6). In this case, to get useful skew of -0.5ns delay FF6 by 0.1ns, then delay FF4 by 0.3ns, early FF2 by 0.1 and then early FF3 by 0.2 ns But since FF4 has other endpoints as well, second lowest skew for FF4 is 0.23ns and third lowest slack is 0.25 ns. For FF3 the second lowest slack is 0.13 ns and third lowest slack is 0.15ns. So when you delay FF4 by 0.3ns , the second and third lowest skew paths gets violated. Same happens in case of FF3 registers. So by removing the ti ming violations for one path we are creating four more violations which is not at all a good trade - off.hence this things needs to be taken into account while searching for useful skew. Apart from this, there might be case where when algorithm moves across timing paths in search of skew, due to feedback loop, it may encounter the same register, this can create an infinite loop. Apart from this , there are self loops as well..So we have to consider case of such timing loops as well. Another possible issue is what if a register is delayed already and current requirement to met required slack is to early the register or vice – versa of the above situation. In this case if we try to early or delay the register by required value , we will see that earlier removed violations again comes into picture and will come with more worst value. The reason for this is that, we early or delay the register using "set_clock_latency" command The proposed work takes into account all such situations and hence is able to dealt with all such issue highlighted above.

3.6 Proposed Second_Order_Slack_Redistribution:-

In order to solve above problem in second order, whenever skew value is adjusted for any register, the maximum skew value that can be taken for any timing path is the second lowest useful slack value of that register whether in case of startpoint or in case of end point. By doing this, we can avoid extra timing path violations that occurs in parallel timing paths because of the skew adjustment. E.q. in the previous example of figure 3 (6), the FF4 register will be delayed by 0.25ns instead of 0.35ns even if the timing violation is not completely removed. But by doing this , we can reduce the value of WNS which is also can be considered as a improvement in timing QoR. There are few cases that we have to consider in case of second order so that skew calculation is optimum. There are mainly three scenarios that we have to consider with this optimized second order approach. Timing improvement not only done by removing the violation completely but can also be done by reducing the WNS as well as TNS. Different cases in optimized second_order_search are explained below:-

Case_1: required_slack <= endpoint_second_lowest_slack

In this case, second_lowest_slack for endpoint is greater or atleast equal to the required slack value. In this case, second_order_slack_redistribution follows the steps given below:-

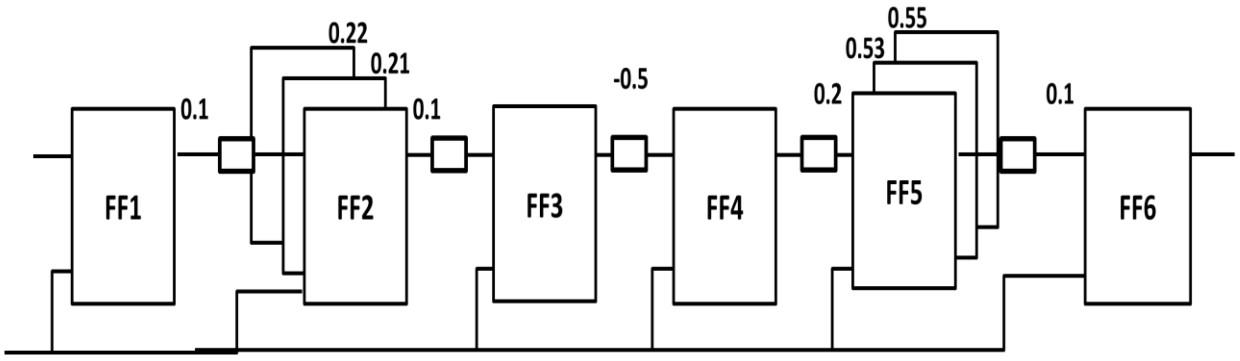

Fig. 3(7) Case_1 for proposed algorithm

- Delay the endpoint by required_slack value.
- Repeat step_5 of the skew_adjustment_flow for newly violated path but only at endpoint side.
- Continue with adjustment until required slack achieved or no positive slack value left for skew redistribution.

Fig. 3(7) explains such a case, here 0.5ns is needed, lowest positive slack is 0.2ns so the skew search algorithm will go for second order search. Now the second lowest skew available for FF4 endpoint is 0.53ns. Hence the algorithm will delay FF4 by 0.5ns.

Case_2: required_slack > endpoint_second_lowest_slack

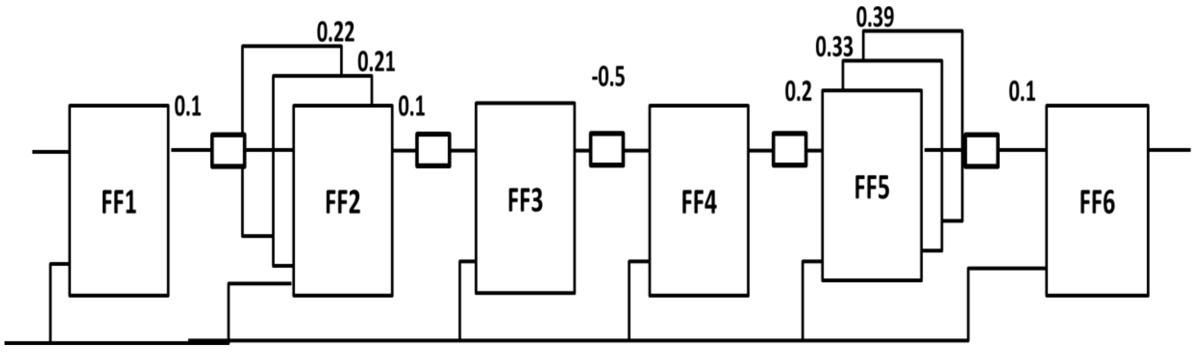

Fig. 3(8) Case_2 for proposed algorithm

In this case, the second_lowest_slack can’t meet the skew requirement , hence we have to look towards startpoint side as well. Steps followed by proc in this case are given below:-

- Delay the endpoint by required_slack value.
- Repeat step_5 of the skew_adjustment_flow for newly violated path but only at endpoint side.

- Continue with adjustment until required slack achieved or no positive slack value left for skew redistribution.
- After completing search from endpoint side, move towards startpoint side to search for remaining required slack value.
- First see whether required value can be met with first order slack of startpoint. If yes, stop there only, otherwise start second order search towards startpoint side.
- The remaining steps are same as for the first case discussed above, the only difference is that endpoint gets replaced by startpoint.

Fig. 3(7) explains such a case, where 0.5ns is needed but the available second lowest slack is 0.33ns hence the FF4 will be delayed by 0.33ns and for remaining the tool will look towards startpoint side.

Case_3: required_slack < startpoint_second_lowest_slack

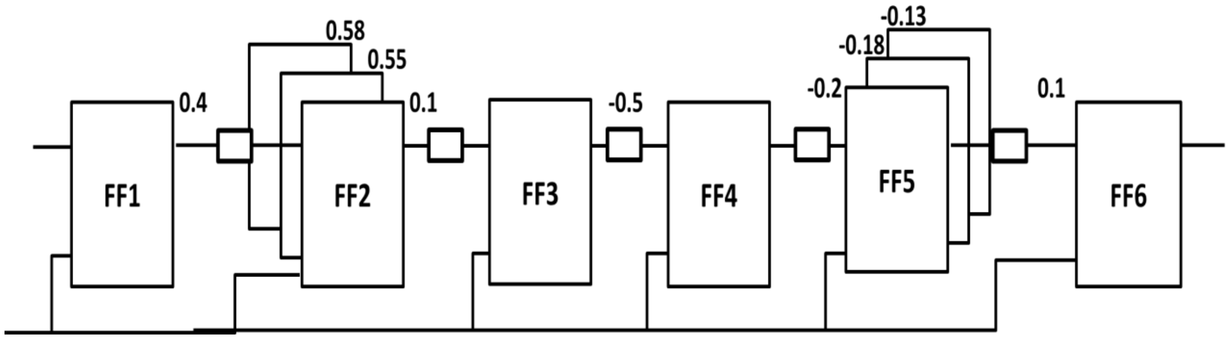

Fig. 3(9) Case_3 for proposed algorithm

This case generally arises when either no positive slack is available at endpoint or the positive slack retrieved from endpoint is not enough to meet the required timing violation. Again the steps followed by the proc in this case are same as in first case, just the endpoint gets replaced by startpoint.

- Look for first order skew towards startpoint side or adjacent forward path.
- If first order skew not sufficient then look for second order skew.

In fig. 3 (9), the frist order slack is 0.1ns which is much lower than required 0.5ns but the second order slack for FF3 is 0.55ns. Hence the Ff3 can be made early by 0.5ns to meet timing violations.

Case_4: required_slack > startpoint_second_lowest_slack

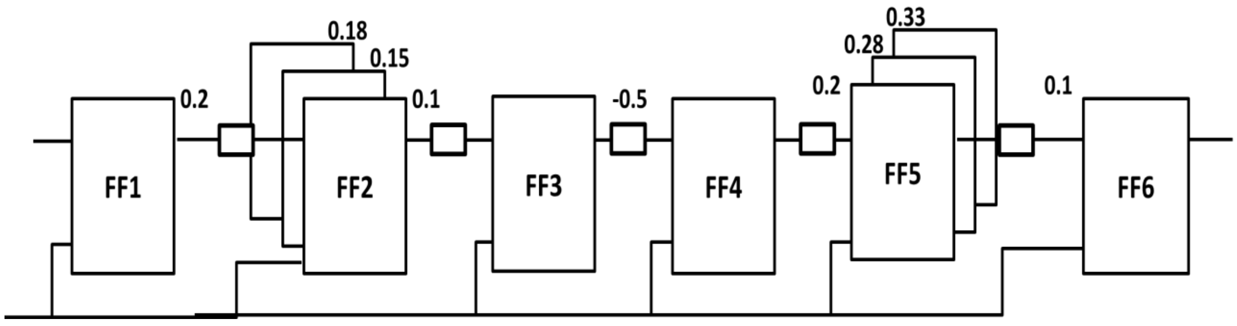

Fig. 3(10) Case_4 for proposed algorithm

Steps followed are:-

- First lok for any useful skew available at endpoint side
- Then for the remaining slack move towards startpoint side and whatever skew is available, take that skew value.

In fig. 3(10), at most 0.28ns can be taken from endpoint side in second order, for the remaining 0.22ns, when skew search algorithm moves towards second order, only 0.15ns is available.

Case_5: both the adjacent forward and backward path have negative slacks

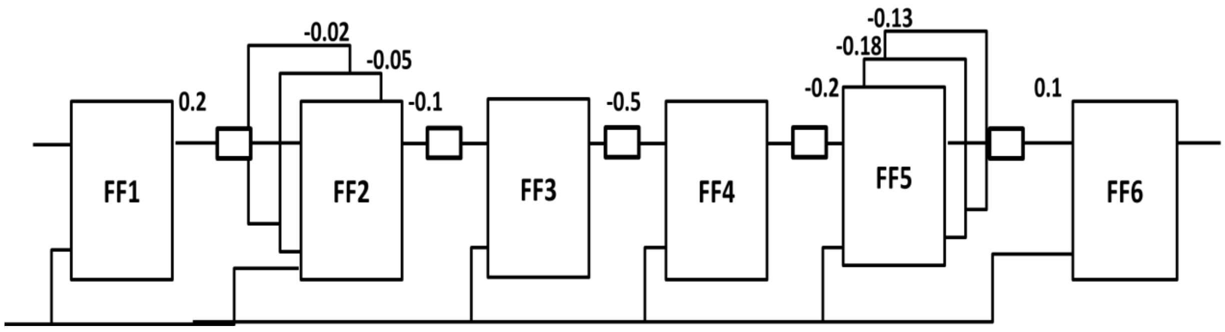

Fig. 3(11) Case_5 for proposed algorithm

In this case, tool will move across timing paths both across forward and backward paths only if the negative slack path has just one such path across all parallel paths otherwise when the skew adjustment is done for such a path, other negative parallel path becomes worse by the same value. This is done for each such negative slack path.

So in this chapter, we have discussed about the proposed methodology in detail, learned about various issues and also different possible scenarios. This methodology is relatively a new approach compared to other skew adjustment techniques discussed in previous chapters. In the upcoming chapter, we will discuss about the results that we have got in detail. This approach can also be modified to make it more effective, that will be discussed in the end during conclusion.

Case_6: Handling a loop in skew search path

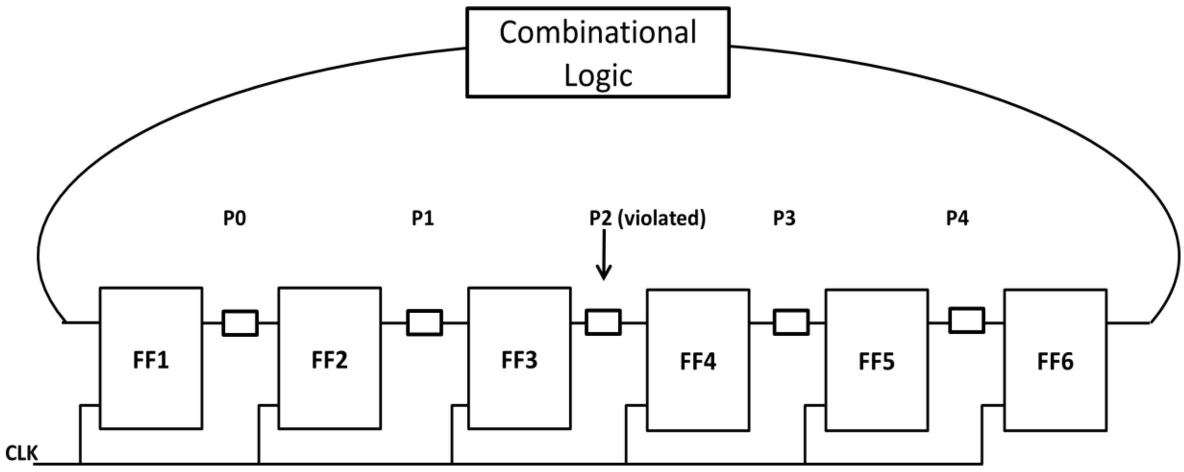

Fig. 3(12) loop in skew search path

If a is loop encountered while searching for the second order slack, then in that case, the algorithm will stop there only for such a case hence avoiding an infinite loop situation. Apart from this there might be a self loop possibility, such a situation is also avoided by this algorithm as it stops searching skew for such cases.

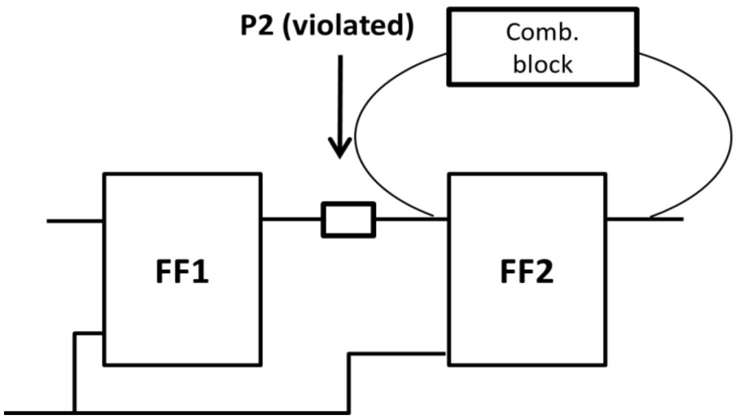

Fig. 3(13) self loop in skew search path

Chapter 4

Results and Discussion

Complete design flow is run for the conventional and proposed design methodology right from synthesis till routing stage of PnR flow. In this chapter, results are shown for different stages of digital design flow such as after synthesis, clock tree synthesis and routing. Timing QoR during synthesis for conventional synthesis flow and synthesis with skew search proc are compared. For CTS and routing , not only timing QoR but also the area and power comparision along with VT distribution is done for conventional design flow and proposed design flow. This skew predicting proc is tested on a design which has approximately 2.5 million standard cells, 85 macros or memory instances and is operating at a frequency of 725MHz.

4.1 Timing QoR comparision after initial compile with and without skew search algorithm:-

Timing_parameter	Without_skew	With_skew
WNS (ns)	-0.4	-0.39
TNS (ns)	-11466.04	-4430.3
NVP	46908	28890

Table 4(1) Timing QoR comparision after initial coimpile

From the above table, it is clear that there is significant improvement in timing quality of design when skew search algorithm is used. If we compare the TNS of the two flows, the total negative slack (TNS) has reduced from whopping -11466.04 ns to -4430.3 ns. Also the number of violating paths i.e. NVP has also reduced from 46908 to 28890. There is not a big improvement in WNS i.e. worst negative slack but still some improvement is there as well. Hence if we look at overall timing QoR, there is significant improvement in timing quality of design. The above results are observed at design compiler itself. But since design compiler is synthesis tool, so for more accurate timing analysis primetime is used.

4.2 Primetime Results after complete synthesis flow:-

Here the timing Qor of the two flows i.e. with skew search proc and conventional flow are compared after full synthesis flow.

As the results observed at primetime which is basically a signoff tool and also gives more accurate timing QoR as its specifically designed for performing timing analysis. These results are for C_worst scenario at 0 degree temperature. This is the worst case scenario as due low voltage and temperature inversion effect , the delaya get worsen. Also this scenario indicates high capacitance for smaller nets. We don't look at best case scenarios during synthesis as there is no clock information during synthesis and hence clock is ideal in this case. The table shown below gives timing QoR comparision after synthesis flow for C_worst_0 scenario:-

Timing_parameter	**Without_skew**	**With_skew**
WNS (ns)	-0.17	-0.13
TNS (ns)	-40.3	-17.66
NVP	2173	1264

Table 4(2) Timing QoR comparision after synthesis

From above table we can clearly say that useful skew has been able to improve timing quality of the design significantly as WNS is reduced from -0.17 ns to -0.13 ns and TNS is reduced from -40.3 to -17.66 which is around 58% improvement and number of violated timing paths are reduced from 2173 to 1264 which shows 40% improvement.

4.3 Results comparision after CTS:-

CTS is a physical implementation stage where the clock tree is built and optimized. At this stage, timing QoR is compared for a number of different scenarios, C_worst is basically used for shorter nets as capacitance is dominant figure in the net delay while RC_worst is used for longer nets as resistance also becomes significant in the net delay for longer nets. Temprature values are 0 and 125 degree celcius. For max path violations, C_worst_0 is the worst case and for min path violations RC_best_125 is the worst case. Reason we are checking for hold violation is that skew values used to improve setup violations worsens the hold violation. As improvement in timing QoR also includes improvement in hold violations so these violations should have acceptable value . Hence hold violations should also be kept in check.

Results are shown in the table below:-

4.3.1 Timing QoR for max_path_violations:-

Timing Parameter	C_worst_0		RC_worst_0		RC_worst_125	
	Without skew	With skew	Without skew	With skew	Without skew	With skew
WNS(ns)	-0.56	-0.45	-0.53	-0.44	-0.34	-0.26
TNS(ns)	-5289.2	-4122.1	-3994.7	-3019.9	-384.9	-278.9
NVP	40885	35182	35235	29716	5518	3158

Table 4(3) Timing QoR comparision for max_path after CTS

4.3.2 Timing QoR for min_path_violations:-

Note:- Scenario used for hold – RC_best_125

Timing_parameter	Without_skew	With_skew
WNS(ns)	-0.01	-0.3
TNS(ns)	-57.6	-765.6
NVP	4954	22337

Table 4(4) Timing QoR comparision for min_path after CTS

4.3.3 VT distribution comparision after CTS:-

VT_group	Without_skew	With_skew
HVT_30	13101	14528
HVT_35	81143	116818
HVT_40	546242	525264
SVT_30	373659	308365
SVT_35	351816	352407
SVT_40	1075130	1114297
LVT_35	20882	26353
LV T_40	2026	2049
Total	2464084	2460166

Table 4(5) Table for VT distribution comparision after CTS

4.3.4 Area and Power comparision after CTS:-

parameter	Without_skew	With_skew
Area	3514972.9	3478596.1
Leakage Power(nW)	4.46e+08	4.23e+08

Table 4(6) Table for area and power comparision after CTS

From table 4(3), we can clearly see that there is a improvement in the setup timing QoR with proposed methodology as compared to conventional methodology.All the timing QoR shown for CTS are obtained from primetime hence are very accurate compared to physical implementation tool like ICC2 or encounter etc.We are seeing improvement in all three scenarios for WNS, TNS and NVP. The table 4 (4) shows possible disadvantage with this method as hold violations increases. But the purposed of this proposed work is to improve setup violations while keeping the hold violations in check. Table 4(5) shows the VT distribution. Here , the count for different channel length and different VT standard cells is given. From the above table , we can clearly see that LVT count increases in CTS as tool tries to incorporate the skew values while building and optimizing the clock tree. Also HVT count increases for proposed work as HVT doesn't needed to be sized to LVT cells to improve setup timing. As discussed in previous chapter that, there should be a improvement in area and power consumption, table 4 (6) validates this result.

4.4 Results Comparision after Routing:-

This is the final stage of physical implementation and is the best stage to analyze the overall effect of proposed methodology on timing quality of design. The reason is that during CTS only clock tree is routed while the delay for data path is still approximated with global routing. But after this stage datapath also gets routed and hence actual delays comes into the picture. Just as in case of CTS, at this stage also, timing for both setup and hold violations for different scenarios will be compared. VT distribution is also compared along with area and power comparision for conventional and proposed methodologies. Tables given below shows all such comparisions and we can see the same trend followed here as was seen in case of CTS. In every corner , we can see that for worst_corners, the setup timing has improved in case of flow with proposed useful_skew_proc as compared to conventional flow. When it comes to best case scenarios, again as in case of CTS, the hold scenario has got worse. This was expected that hold will degrade as for now the focus is to improve setup violations. Table 4(7) gives timing QoR for the setup violations in case of conventional flow while the Table 4(8) talks of the timing QoR for setup violations in case of design flow with proposed useful_skew_proc. While table 4(9) and 4(10) will put light on timing QoR for conventional flow and design flow with useful_skew_proc and table 4(11), 4(12) and 4(13) talk about the comparision of VT distribution, area and leakage power consumption and runtime for the conventional flow and flow with useful_skew_proc.

4.4.1 Timing QoR for setup_violations after routing:-

Parameters	C_worst_0	RC_worst_0	C_worst_125	RC_worst_125
WNS	-0.46	-0.37	-0.26	-0.21
TNS	-4224.9	-3980.4	-278	-274.7
NVP	37947	35782	4701	4719

Table 4(7) Timing QoR for setup after routing for conventional flow

Parameters	C_worst_0	RC_worst_0	C_worst_125	RC_worst_125
WNS	-0.39	-0.37	-0.26	-0.18
TNS	-3672	-3980.4	-189	-219
NVP	32733	35782	3835	3230

Table 4(8) Timing QoR comparision for setup after routing for design flow with proposed useful_skew_proc

4.4.2 Timing QoR for hold_violations after routing:-

Parameters	C_best_0	RC_best_0	C_best_125	RC_best_125
WNS	-0.19	-0.19	-0.21	-0.21
TNS	-133.9	-145.1	-170.9	-183.9
NVP	3755	3903	4654	4864

Table 4(9) Timing Qor for hold after Routing with conventional flow

Parameters	C_best_0	RC_best_0	C_best_125	RC_best_125
WNS	-0.25	-0.26	-0.28	-0.29
TNS	-1257	-1354	-1576	-1682
NVP	20270	21605	23198	24489

Table 4(10) Timing QoR for hold after Routing for design flow with useful_skew_proc

4.4.3 VT_distribution comparision after routing:-

VT_group	**without_skew**	**with_skew**
HVT_30	**26569**	**30238**
HVT_35	**94224**	**116184**
HVT_40	**708924**	**739751**
SVT_30	**390474**	**306239**
SVT_35	**379861**	**335249**
SVT_40	**971975**	**986520**
LVT_35	**20882**	**26353**
Total	**2592994**	**2540619**

Table 4(11) Table for VT distribution comparision after Routing

4.4.4 Area and power comparision after routing:-

parameter	**Without_skew**	**With_skew**
area	3648045.15	3622606
Leakage_power(nW)	4.51e+08	4.44e+08

Table 4(12) Table for area and power comparision after Routing

4.4.5 Runtime comparision of complete design flow:-

Runtime in case of conventional design flow generally depends the core area, no. of standard cells as they directly impact the timing QoR and hence the tool will take its own time to improve their timing. In case of proposed work, runtime contains extra segment which is the time taken by the useful_skew_proc to calculate the skew values.

Flow type	**Conventional_flow**	**Proposed_flow**
Synthesis flow	39 hrs 46 minutes	44 hrs 46 minures
PnR flow	151 hrs 44 minutes	135 hrs
Total	191 hrs 30 minutes	179 hrs 46 minutes

Table 4(13) Table for runtime comparision

If we take a look from from table 4(7) to table 4(13), the same trend is followed as in case of CTS, we are seeing improvement in setup violations for each worst case scenario whether it is C-worst at a temperature 0 or 125 OR RC_worst at the same temperature values and we can see degradation in timing QoR for hold violations in case of every best case scenario whether it is C_best OR RC_best. This was expected that setup will improve and hold will degrade. Apart from this, we are also seeing improvement in area and leakage power. Apart from this there is improvement in run time as well approximately of 10 hr, specially in case of PnR flow, because the skew values has improved the timing QoR, so implementation tool needs less effort in case of proposed design methodology. But there was increase in synthesis runtime as skew search proc takes his own time to search for useful skew values. So we are able to obtain required results using the proposed methodology. The skew search algorithm can be further modified to improve the hold violations as then we can achieve complete timing QoR improvement. How we can hold along with setup will be discussed in next chapter.

Chapter 5
Conclusions and Future Scope

In this thesis work, first we talked about the limitations associated with the conventional or standard design flow such as less optimized timing QoR, increased area requirement, increase in leakage power, feedback loop from physical implementation stage back to synthesis and increased runtime. Then we talked about the proposed methodology that deals with all the problems mentioned above. This method uses useful_skew_proc that calculates useful skew available in design during synthesis itself not only at the adjacent forward and backward timing paths but also searches for skew in second order and distributes that skew value across the design to optimize setup timing violations while keeping the hold violations in check. The results discussed earlier shows that there is improvement in setup violations, area improves, leakage power reduces and since there is no feedback from implementation stages to synthesis has no re-spinning of synthesis and hence flow becomes unidirectional. As optimized netlist is obtained from synthesis, so runtime of overall flow reduces as shown in results earlier.

One of the issue with skew calculation algorithm is that hold time worsens, as we have seen in case of both CTS and routing results for best case scenarios. The algorithm can be modified in future to improve hold violations as well. One of the reason for hold violation to occur is thode endpoints where skew value is applied, then timing paths whose startpoints are such registers, hold violation has occurred in such paths. As the skew values are calculated with worst corner active, so skew calculation might not be right for such paths and hence such path needs greater margin for a given hold slack value. Apart from this, the reverse of the current algorithm to improve violated hold paths can also be done.

Also this algorithm distributes useful skew for those paths that belongs to same clock domain. If there is a path whose startpoint and endpoint belongs to different clock domains, no skew adjustment for such paths. Hence cross clock domain skew distribution is not possible with current skew search algorithm. This is also something that needs to be improved. Hence the proposed methodology has solved some issues of conventional approach but still there is scope for improvement as well in terms of improvement in hold violation and also in cross clock domain adjustment.

APPENDIX

1. proc to find no. of violated R2R paths:-

```
proc number_of_R2R_paths_violated {} {

set experiment_dir_path "./Final_Script_2"

report_qor > $experiment_dir_path/report_qor.rpt

set aa [sh grep "Scenario " $experiment_dir_path/report_qor.rpt]

set bb [sh grep "Timing Path Group" $experiment_dir_path/report_qor.rpt]

set cc [sh grep "No. of Violating Paths:" $experiment_dir_path/report_qor.rpt]

set max_paths 0

foreach a [split $aa \n] b [split $bb \n] c [split $cc \n] {

if {[regexp func [lindex $a 1]] && [regexp R2R [lindex $b 3]]} {

set max_paths [lindex $c 4]

break

}

}

set max_paths [expr {int ($max_paths)}]

puts "max_paths are $max_paths"

return $max_paths

}
```

2. proc to generate list containing violated paths

```
proc S_E_S { memory_list } {

set experiment_dir_path "./Final_Script_2"
report_timing -group R2R -scenarios func -delay_type max -max_paths
[number_of_R2R_paths_violated] -significant_digits 6 >
$experiment_dir_path/report_timing_post_compile.rpt

set aa [sh grep "Startpoint:" "$experiment_dir_path/report_timing_post_compile.rpt"]

set bb [sh grep "Endpoint:" "$experiment_dir_path/report_timing_post_compile.rpt"]

set cc [sh grep "slack" "$experiment_dir_path/report_timing_post_compile.rpt"]

set start_end_slack ""

foreach a [split $aa \n] b [split $bb \n] c [split $cc \n] {

if { [mem_reg_info [lindex $a 1] $memory_list] == 0 && [mem_reg_info [lindex $b 1]
$memory_list] == 0} {

set violated_slack "[lindex $c 2]"

if {$violated_slack < 0} {

lappend start_end_slack "[lindex $a 1] [lindex $b 1] $violated_slack"

}

}

}

set start_end_slack_length [llength $start_end_slack]

puts "start_end_slack length is $start_end_slack_length"

return $start_end_slack

}
```

3. proc to check timing path with same clocks for startpoint and endpoint:-

```
proc matching_clocks_of_SP_EP {reg_name_1 reg_name_2} {

set experiment_dir_path "./Final_Script_2"
report_timing -group R2R -scenarios func -delay_type max -significant_digits 6 -from
$reg_name_1/CK -to $reg_name_2/D > $experiment_dir_path/report_timing.rpt

set clock_line [sh grep "clock" $experiment_dir_path/report_timing.rpt]

set i 0

foreach ss [split $clock_line \n] {

incr i

if {$i == 3} {

split $ss \n

set sp_clock [lindex $ss 1]

}

if {$i == 5} {

split $ss \n

set ep_clock [lindex $ss 1]

}

}

if {$sp_clock == $ep_clock} {

return 1

} else {

return 0

}}
```

4. proc with all violated path info can cure self_loop:-

```
proc vio_path_info_FROM_TO {reg_name_1 reg_name_2 list_macro} {

set experiment_dir_path "./Final_Script_2"
```

```
if { [mem_reg_info $reg_name_1 $list_macro] == 0 && [mem_reg_info $reg_name_2
$list_macro] == 0 && $reg_name_1 != $reg_name_2} {

report_timing -group R2R -scenarios func -delay_type max -significant_digits 6 -from
$reg_name_1/CK -to $reg_name_2/D > $experiment_dir_path/report_timing_slack.rpt

set non_R2R_check  "not_empty"

set a [open $experiment_dir_path/report_timing_slack.rpt r]

while {[gets $a line]!= -1} {

if {[regexp "No" $line]} {

set non_R2R_check ""

}

}

if {$non_R2R_check != ""} {

close $a

set slack_line [sh grep "slack" $experiment_dir_path/report_timing_slack.rpt]

split $slack_line \n

set vio_slack [lindex $slack_line 2]

if {[matching_clocks_of_SP_EP $reg_name_1 $reg_name_2]} {

return "$vio_slack"

} else {

echo "start_point and endpoint has different clocks"

return ""

}

} else {
```

```
close $a
return ""
}
} else {
return ""
}
}
```

5. proc for all low skew information with self loop taken care of in endpoint side:-

```
proc  low_skew_information {reg_s list_macro already_done_startpoints
already_done_endpoints} {
if { [mem_reg_info $reg_s $list_macro] == 0 } {
set experiment_dir_path "./Final_Script_2"
report_timing -group R2R -scenarios func -delay_type max -max_paths 1000 -
significant_digits 6 -from $reg_s/CK > $experiment_dir_path /  report_timing1.rpt
set non_R2R_check   "not_empty"
set a [open $experiment_dir_path/report_timing1.rpt r]
while {[gets $a line]!= -1} {
if {[regexp "No" $line]} {
set non_R2R_check ""
}
}
if { $non_R2R_check != "" } {
 close $a
```

```
}

 set start_end_pos_slack_list ""

 set ff [sh grep "Endpoint:" $experiment_dir_path/report_timing1.rpt]

set gg [sh grep "slack" $experiment_dir_path/report_timing1.rpt]

foreach f [split $ff \n] g [split $gg \n] {

lappend start_end_pos_slack_list "$reg_s  [lindex $f 1]  [lindex $g 2]"

}

if {[reg_match_to_list_with_2_elements $reg_s $already_done_startpoints]==0 &&
[reg_match_to_list_with_2_elements $reg_s $already_done_endpoints]==0} {

set i 0

foreach start_end_pos_slack_line $start_end_pos_slack_list {

set reg_e   [lindex $start_end_pos_slack_line 1]

set slack_v [lindex $start_end_pos_slack_line 2]

if { [reg_match_to_list_with_2_elements $reg_e $already_done_endpoints] == 1} {

set latency_set_1 [lindex [get_list_elements $reg_e $already_done_endpoints] 1]

set slack_v       [expr {$slack_v + $latency_set_1}]

lset start_end_pos_slack_list $i 2 $slack_v

}

if { [reg_match_to_list_with_2_elements $reg_e $already_done_startpoints] == 1 } {

set latency_set_1 [lindex [get_list_elements $reg_e $already_done_startpoints] 1]

set slack_v       [expr {$slack_v - $latency_set_1}]

lset start_end_pos_slack_list $i 2 $slack_v

}

incr i
}
} elseif {[reg_match_to_list_with_2_elements $reg_s $already_done_startpoints]==1} {
set latency_set   [lindex [get_list_elements $reg_s $already_done_startpoints] 1]
set i 0
foreach start_end_pos_slack_line $start_end_pos_slack_list {
set reg_e   [lindex $start_end_pos_slack_line 1]
set slack_v [expr { [lindex $start_end_pos_slack_line 2] + $latency_set }]
lset start_end_pos_slack_list $i 2 $slack_v
if { [reg_match_to_list_with_2_elements $reg_e $already_done_endpoints] == 1} {
set latency_set_1 [lindex [get_list_elements $reg_e $already_done_endpoints] 1]
set slack_v       [expr {$slack_v + $latency_set_1}]
 lset start_end_pos_slack_list $i 2 $slack_v
}
if { [reg_match_to_list_with_2_elements $reg_e $already_done_startpoints] == 1 } {
set latency_set_1 [lindex [get_list_elements $reg_e $already_done_startpoints] 1]
set slack_v       [expr {$slack_v - $latency_set_1}]
lset start_end_pos_slack_list $i 2 $slack_v
}
incr i
}
} else {
set latency_set   [lindex [get_list_elements $reg_s $already_done_endpoints] 1]
set i 0
```

```
foreach start_end_pos_slack_line $start_end_pos_slack_list {

set reg_e   [lindex $start_end_pos_slack_line 1]

set slack_v [expr { [lindex $start_end_pos_slack_line 2] - $latency_set }]

lset start_end_pos_slack_list $i 2 $slack_v

if { [reg_match_to_list_with_2_elements $reg_e $already_done_endpoints] == 1} {

set latency_set_1 [lindex [get_list_elements $reg_e $already_done_endpoints] 1]

set slack_v       [expr {$slack_v + $latency_set_1}]

lset start_end_pos_slack_list $i 2 $slack_v

}

if { [reg_match_to_list_with_2_elements $reg_e $already_done_startpoints] == 1 } {

set latency_set_1 [lindex [get_list_elements $reg_e $already_done_startpoints] 1]

set slack_v       [expr {$slack_v - $latency_set_1}]

lset start_end_pos_slack_list $i 2 $slack_v

}

 incr i

}

}

set start_end_pos_slack_list [lsort -real -index 2 $start_end_pos_slack_list]

set lowest_slack     ""

set lowest_slack_2nd ""

set lowest_slack_reg ""

if {[mem_reg_info [lindex $start_end_pos_slack_list 0 1] $list_macro] == 0} {

if {$reg_s != [lindex $start_end_pos_slack_list 0 1]} {
```

```
if {[matching_clocks_of_SP_EP $reg_s [lindex $start_end_pos_slack_list 0 1]]} {

set lowest_slack     [lindex $start_end_pos_slack_list 0 2]

set lowest_slack_reg [lindex $start_end_pos_slack_list 0 1]

if {[llength $start_end_pos_slack_list]>=2 && [mem_reg_info [lindex
$start_end_pos_slack_list 1 1] $list_macro] == 0 } {

if {$reg_s != [lindex $start_end_pos_slack_list 1 1]} {

 if {[matching_clocks_of_SP_EP $reg_s [lindex $start_end_pos_slack_list 1 1]]} {

set lowest_slack_2nd [lindex $start_end_pos_slack_list 1 2]

} else {

echo "start_point and endpoint has different clocks"

}

} else {

if {[llength $start_end_pos_slack_list]>=3 && [mem_reg_info [lindex
$start_end_pos_slack_list 2 1] $list_macro] == 0 } {

if {[matching_clocks_of_SP_EP $reg_s [lindex $start_end_pos_slack_list 2 1]]} {

set lowest_slack_2nd [lindex $start_end_pos_slack_list 2 2]

} else {

echo "start_point and endpoint has different clocks"

}

}

}

}

} else {
```

```
echo "start_point and endpoint has different clocks"

}

} else {

if {[llength $start_end_pos_slack_list]>=2 && [mem_reg_info [lindex
$start_end_pos_slack_list 1 1] $list_macro] == 0 } {

if {[matching_clocks_of_SP_EP $reg_s [lindex $start_end_pos_slack_list 1 1]]} {

set lowest_slack [lindex $start_end_pos_slack_list 1 2]

set lowest_slack_reg [lindex $start_end_pos_slack_list 1 1]

if {[llength $start_end_pos_slack_list]>=3 && [mem_reg_info [lindex
$start_end_pos_slack_list 2 1] $list_macro] == 0 } {

if {[matching_clocks_of_SP_EP $reg_s [lindex $start_end_pos_slack_list 2 1]]} {

set lowest_slack_2nd [lindex $start_end_pos_slack_list 2 2]

} else {

echo "start_point and endpoint has different clocks"

}

}

} else {

echo "start_point and endpoint has different clocks"

}

}

}

return "$lowest_slack_reg $lowest_slack $lowest_slack_2nd"

} else {
```

```
return ""
}
} else {
close $a
 return ""
}
} else {
return ""
}
}
```

6. proc to calculate lowest skew for startpoint:-

```
proc start_low_skew_information {reg_e list_macro already_done_startpoints
already_done_endpoints} {
if { [mem_reg_info $reg_e $list_macro] == 0 } {
set experiment_dir_path "./final_test"
 report_timing -group R2R -scenarios func -delay_type max -max_paths 1000 -nworst
1000 -significant_digits 6 -to $reg_e/D > $experiment_dir_path/report_timing1.rpt
set non_R2R_check   "not_empty"
set a [open $experiment_dir_path/report_timing1.rpt r]
while {[gets $a line]!= -1} {
if {[regexp "No" $line]} {
set non_R2R_check ""
}
```

```
}

if { $non_R2R_check != "" } {

close $a

set start_end_pos_slack_list ""

set ff [sh grep "Startpoint:" $experiment_dir_path/report_timing1.rpt]

set gg [sh grep "slack" $experiment_dir_path/report_timing1.rpt]

foreach f [split $ff \n] g [split $gg \n] {

lappend start_end_pos_slack_list "[lindex $f 1] $reg_e [lindex $g 2]"

}

if {[reg_match_to_list_with_2_elements $reg_e $already_done_startpoints]==0 &&
[reg_match_to_list_with_2_elements $reg_e $already_done_endpoints]==0} {

set i 0

foreach start_end_pos_slack_line $start_end_pos_slack_list {

set reg_s   [lindex $start_end_pos_slack_line 0]

set slack_v [lindex $start_end_pos_slack_line 2]

 if { [reg_match_to_list_with_2_elements $reg_s $already_done_endpoints] == 1} {

set latency_set_1 [lindex [get_list_elements $reg_s $already_done_endpoints] 1]

set slack_v       [expr {$slack_v - $latency_set_1}]

lset start_end_pos_slack_list $i 2 $slack_v

}

if { [reg_match_to_list_with_2_elements $reg_s $already_done_startpoints] == 1 } {

set latency_set_1 [lindex [get_list_elements $reg_s $already_done_startpoints] 1]

set slack_v       [expr {$slack_v + $latency_set_1}]
lset start_end_pos_slack_list $i 2 $slack_v

}

incr i

}

} elseif {[reg_match_to_list_with_2_elements $reg_e $already_done_startpoints]==1} {

set latency_set   [lindex [get_list_elements $reg_e $already_done_startpoints] 1]

set i 0

foreach start_end_pos_slack_line $start_end_pos_slack_list {

set reg_s   [lindex $start_end_pos_slack_line 0]

set slack_v [expr { [lindex $start_end_pos_slack_line 2] - $latency_set }]

lset start_end_pos_slack_list $i 2 $slack_v

if { [reg_match_to_list_with_2_elements $reg_s $already_done_endpoints] == 1} {

set latency_set_1 [lindex [get_list_elements $reg_s $already_done_endpoints] 1]

set slack_v       [expr {$slack_v - $latency_set_1}]

lset start_end_pos_slack_list $i 2 $slack_v

}

if { [reg_match_to_list_with_2_elements $reg_s $already_done_startpoints] == 1 } {

set latency_set_1 [lindex [get_list_elements $reg_s $already_done_startpoints] 1]

 set slack_v       [expr {$slack_v + $latency_set_1}]

 lset start_end_pos_slack_list $i 2 $slack_v

}

incr i

}
```

```
} else {
set latency_set   [lindex [get_list_elements $reg_e $already_done_endpoints] 1]
set i 0
foreach start_end_pos_slack_line $start_end_pos_slack_list {
set reg_s   [lindex $start_end_pos_slack_line 0]
set slack_v [expr { [lindex $start_end_pos_slack_line 2] + $latency_set }]
lset start_end_pos_slack_list $i 2 $slack_v
if { [reg_match_to_list_with_2_elements $reg_s $already_done_endpoints] == 1} {
set latency_set_1 [lindex [get_list_elements $reg_s $already_done_endpoints] 1]
set slack_v       [expr {$slack_v - $latency_set_1}]
lset start_end_pos_slack_list $i 2 $slack_v
}
if { [reg_match_to_list_with_2_elements $reg_s $already_done_startpoints] == 1 } {
set latency_set_1 [lindex [get_list_elements $reg_s $already_done_startpoints] 1]
set slack_v       [expr {$slack_v + $latency_set_1}]
lset start_end_pos_slack_list $i 2 $slack_v
}
incr i
}
}
 set start_end_pos_slack_list [lsort -real -index 2 $start_end_pos_slack_list]
 set lowest_slack     ""
 set lowest_slack_2nd ""
 set lowest_slack_reg ""

 if {[mem_reg_info [lindex $start_end_pos_slack_list 0 0] $list_macro] == 0} {

 if {$reg_e != [lindex $start_end_pos_slack_list 0 0]} {

if {[matching_clocks_of_SP_EP [lindex $start_end_pos_slack_list 0 0] $reg_e]} {

 set lowest_slack     [lindex $start_end_pos_slack_list 0 2]

 set lowest_slack_reg [lindex $start_end_pos_slack_list 0 0]

 if {[llength $start_end_pos_slack_list]>=2 && [mem_reg_info [lindex
$start_end_pos_slack_list 1 0] $list_macro] == 0 } {

if {$reg_e != [lindex $start_end_pos_slack_list 1 0]} {

if {[matching_clocks_of_SP_EP [lindex $start_end_pos_slack_list 1 0] $reg_e ]} {

set lowest_slack_2nd [lindex $start_end_pos_slack_list 1 2]

} else {

echo "start_point and endpoint has different clocks"

}

} else {

if {[llength $start_end_pos_slack_list]>=3 && [mem_reg_info [lindex
$start_end_pos_slack_list 2 0] $list_macro] == 0 } {

if {[matching_clocks_of_SP_EP [lindex $start_end_pos_slack_list 2 0] $reg_e]} {

set lowest_slack_2nd [lindex $start_end_pos_slack_list 2 2]

} else {

echo "start_point and endpoint has different clocks"

}

}
}
}
} else {
echo "start_point and endpoint has different clocks"
}
} else {
if {[llength $start_end_pos_slack_list]>=2 && [mem_reg_info [lindex
$start_end_pos_slack_list 1 0] $list_macro] == 0 } {
if {[matching_clocks_of_SP_EP [lindex $start_end_pos_slack_list 1 0] $reg_e]} {
set lowest_slack [lindex $start_end_pos_slack_list 1 2]
set lowest_slack_reg [lindex $start_end_pos_slack_list 1 0]
if {[llength $start_end_pos_slack_list]>=3 && [mem_reg_info [lindex
$start_end_pos_slack_list 2 0] $list_macro] == 0 } {
if {[matching_clocks_of_SP_EP [lindex $start_end_pos_slack_list 2 0] $reg_e]} {
set lowest_slack_2nd [lindex $start_end_pos_slack_list 2 2]
} else {
echo "start_point and endpoint has different clocks"
}
}
} else {
echo "start_point and endpoint has different clocks"
}
}
```

```
}
return "$lowest_slack_reg $lowest_slack $lowest_slack_2nd"
} else {
return ""
}
} else {
close $a
return ""
}
} else {
return ""
}
}
```

7. proc to calculate avg. value of elements in a list:-

```
proc slack_avg { input_list macro_list already_done_SP already_done_EP} {
set x [llength $input_list]
set y 0
set i 0
foreach line $input_list {
set z [calculate_updated_slack [lindex $line 1] [lindex $line 0] $already_done_EP
$already_done_SP $macro_list]
if { $z != ""} {
set y [expr { $y + $z }]
}

}

return "[expr { $y / $x}]"

}
```

8. proc to calculating timing path slack:-

```
proc calculate_updated_slack { reg_e reg_s already_done_endpoints
already_done_startpoints macro_list} {

set slack_v       [vio_path_info_FROM_TO $reg_s $reg_e $macro_list]

if { $slack_v == "" } {

return ""

} else {

if {  [reg_match_to_list_with_2_elements $reg_e $already_done_endpoints]== 0 &&
[reg_match_to_list_with_2_elements $reg_e $already_done_startpoints]==0} {

if {[reg_match_to_list_with_2_elements $reg_s $already_done_startpoints]==0 &&
[reg_match_to_list_with_2_elements $reg_s $already_done_endpoints]==0} {

 return $slack_v

} elseif {[reg_match_to_list_with_2_elements $reg_s $already_done_startpoints]==1} {

set latency_set   [lindex [get_list_elements $reg_s $already_done_startpoints] 1]

set slack_v       [expr { $slack_v + $latency_set }]

 return $slack_v

} else {

set latency_set   [lindex [get_list_elements $reg_s $already_done_endpoints] 1]

set slack_v       [expr { $slack_v - $latency_set }]

 return $slack_v
```

```
}

} elseif { [reg_match_to_list_with_2_elements $reg_e $already_done_endpoints] == 1} {

 set latency_set_1 [lindex [get_list_elements $reg_e $already_done_endpoints] 1]

set slack_v       [expr {$slack_v + $latency_set_1}]

if {[reg_match_to_list_with_2_elements $reg_s $already_done_startpoints]==0 &&
[reg_match_to_list_with_2_elements $reg_s $already_done_endpoints]==0} {

return $slack_v

} elseif {[reg_match_to_list_with_2_elements $reg_s $already_done_startpoints]==1} {

 set latency_set   [lindex [get_list_elements $reg_s $already_done_startpoints] 1]

 set slack_v       [expr { $slack_v + $latency_set }]

 return $slack_v

} else {

set latency_set   [lindex [get_list_elements $reg_s $already_done_endpoints] 1]

set slack_v       [expr { $slack_v - $latency_set }]

return $slack_v

 }

} else {

set latency_set_1 [lindex [get_list_elements $reg_e $already_done_startpoints] 1]

set slack_v       [expr {$slack_v - $latency_set_1}]

 if {[reg_match_to_list_with_2_elements $reg_s $already_done_startpoints]==0 &&
[reg_match_to_list_with_2_elements $reg_s $already_done_endpoints]==0} {

return $slack_v

} elseif {[reg_match_to_list_with_2_elements $reg_s $already_done_startpoints]==1} {
```

```
set latency_set   [lindex [get_list_elements $reg_s $already_done_startpoints] 1]

set slack_v       [expr { $slack_v + $latency_set }]

return $slack_v

} else {

set latency_set   [lindex [get_list_elements $reg_s $already_done_endpoints] 1]

set slack_v       [expr { $slack_v - $latency_set }]

return $slack_v }}}}
```

9. proc for deleting a line in a file:-

```
set file_1 or 2 or 3 [open ./sasharma/project_work/post_comple/file_to_modify w]

proc  delete_line {file_with_path pattern} {

set file_name [open $file _with_path r]

set duplicate_file [open ./Final_Script_2/tempfile w]

while {[gets $file_name line] >= 0} {

if {![regexp "$pattern" $line]} {

puts $duplicate_file $line

}

}

close $duplicate_file

close $file_name

sh mv ./Final_Script_2/tempfile $file_with_path }
```

10. proc to put a dummy variable "empty" in a empty list:-

```
proc  check_for_empty_list { list_name } {

if {[llength $list_name] == 0 } {
```

```
set list_name "empty"

}

return $list_name

}
```

11. proc to make a list empty if it contains dummy variable "empty" is present:-

```
proc  set_empty_list { list_name } {

set element_1 [lindex $list_name 0]

if { [regexp "empty" $element_1] } {

set list_name ""

}

 return $list_name

}
```

12. proc for matching a reg to a reg:-

```
proc reg_match { pattern_1 pattern_2 } {

if {$pattern_1 == $pattern_2} {

 return 1

} else {

return 0

}}
```

13. proc for matching register to a list of registers:-

```
proc reg_match_to_list { pattern_1 list_input } {

set list_input [set_empty_list $list_input]

set ret 0
if { [llength $list_input] > 0} {

foreach pattern_2 $list_input {

if { $pattern_1 == $pattern_2} {

set ret 1

break

}

}

}

return $ret

}
```

14. proc to match with a list with each of it:-

```
proc reg_match_to_list_with_2_elements { pattern_1 list_input } {

set list_input [set_empty_list $list_input]

 set list_input_length [llength $list_input]

 set ret 0

 if { [llength $list_input] > 0} {

 foreach line_list_input $list_input {

 set pattern_2 [lindex $line_list_input 0]

if { $pattern_1 == $pattern_2} {

 set ret 1

 break

 }

 }
 }
 return $ret
}
```

15. **proc for indentifying memory_registers:-**

```
proc mem_reg_info { pattern mem_list} {
if {[reg_match_to_list $pattern $mem_list]} {
return 1
} else {
return 0
}
}
```

16. proc for getting already applied value by matching register to a list of registers:-

```
proc get_list_elements { pattern_1 list_input } {
set list_input [set_empty_list $list_input]
foreach list_line $list_input {
set pattern_2 [lindex $list_line 0]
if { $pattern_1 == $pattern_2} {
return "[lindex $list_line 0] [lindex $list_line 1]"
 break
 } else {
 continue
}
}}
```

17. proc for updating a value in a list whose contains sublist and pattern matched with always firstst element of sublist:-

```
proc update_value_in_list { pattern_1 value list_input } {

set i 0

set list_input_1 $list_input

foreach list_line $list_input_1 {

set pattern_2     [lindex $list_line 0]

if {$pattern_1 == $pattern_2} {

lset list_input_1 $i 1 $value

break

}

incr i

}

return $list_input_1

}
```

18. **proc for updating a value in a output_list:-**

```
proc update_value_in_output_list { pattern_1 value list_input } {

set j 0

set list_input_2 $list_input

foreach list_line $list_input_2 {

set pattern_2     [lindex $list_line 2]

if {[regexp $pattern_1 $pattern_2]} {

 lset list_input_2 $j 1 $value
 break
}
incr j
}
return $list_input_2
}
```

19. proc for modifying a list by matching to a another list:-

```
if {0} {
proc match_list_to_list { list_to_modify list_with_match } {
 set i 0
 set list_1 $list_to_modify
 set list_1_index ""
 set list_2 $list_with_match
 foreach line_list_1 $list_1 {
 set pattern_1 [lindex $line_list_1 1]
 set j 0
 foreach line_list_2 $list_2 {
set pattern_2 [lindex $line_list_2 2]
if { $pattern_1 == $pattern_2 } {
set list_2 [lreplace $list_2 0 0]
lappend list_1_index $i
break
} else {
 incr j
 }
 }
 incr i
 if { [llength $list_1] == 0 || [llength $list_2] == 0 } {
 break
 }
 }
 set list_1_index [lreverse $list_1_index]
 foreach hihi $list_1_index {
 set list_1 [lreplace $list_1 $hihi $hihi]
 }
 return $list_1
}
}
```

20. proc for matching a file to a file:-

```
proc match_files { file_1_with_path file_2_with_path } {
set list_1 [open $file_1_with_path r]
set list_2 [open $file_2_with_path r]
set list_1_mo [split [read $list_1] \n]
set list_1_up [lreplace $list_1_mo end end]
set list_2_mo [split [read $list_2] \n]
set list_2_up [lreplace $list_2_mo end end]
set j 1

set list_1_length [llength $list_1_mo]

set list_2_length [llength $list_2_up]

puts $list_1_mo

puts "$list_1_length $list_2_length"

while {1} {

foreach xx $list_1_up {

split $xx \n

set string_1 [lindex $xx 1]/CK

foreach yy $list_2_up {

split $yy \n

set string_2 "[lindex $yy 2]"

if {$string_1 == $string_2} {

puts $j

}

}

if {$j > $list_1_length} {

break

}

incr j

}

break

}
```

```
close $list_1

close $list_2

}
```

REFRENCES

1. Design Compiler User Guide
2. IC Compiler 2 User Guide
3. Primetime User Guide
4. Solvnet – Synopsys User Group
5. Chee Jau Soon, Wang Bo, Predictive clock skew redistribution methodology for improved timing QoR, Intel Technology, Asia Pte. Ltd. , Singapore
6. J. Bhasker, Rakesh Chadha, Static Timing Analysis for Nanometer Designs, Springer, 978-0-387-93819-6 (ISBN)