EAI Endorsed Transactions
on Cloud Systems
Research Article EAI.EU

Automated RTL Generator for Optimized Flop Repeater Network

Rahul S Bhatt[1,*], Shweta Sharma[1] and Shyam A[1]

[1]Intel Technologies India Pvt Ltd, Bangalore, India

Abstract

In all Hard IP (HIP) designs, there is a need for signals to travel from its source to destination module. To meet timing, they are flop repeated based on the floorplan of the design. Different signals have unique repetition requirements based on their functionality/timing criticality. Also signals may have single destination or multiple destinations.
The flop repeater structure for each signal should be optimized so that it has minimum number of flops and timing is also met, to achieve desired targets for timing, area and power. This paper will show how we are creating optimized tree structure based on mathematical techniques and generating automated RTL for such repeater flop module. This submission presents a new tool **ART** (Auto Repeater Tool) that completely eliminates the manual effort/time required to generate functional, ready to plug-in RTL.

Keywords: Optimized repeater flop structure, Repeater, Flop, Automated RTL, SREP/TREE, Logic design.

Received on 15 November 2019, accepted on 03 January 2020, published on 10 January 2020

doi: 10.4108/eai.13-7-2018.162634

*Corresponding author. Email: rahul.s.bhatt@intel.com

1. Introduction

The Hard IP (HIP) designs, nowadays, are getting very competitive in terms of power and area. The basic element consuming power is the flip flop. The more the number of flops, the greater the power consumption. Modern designs are having thousands of signals which are flop repeated multiple times from source to destination. Most of the IPs have a dedicated module, let's call it "Repeater Channel", which flop repeats these signals and feeds them to their destination module. The functionality of such Repeater Channel module is very critical as it has to carefully flop repeat each signal based on multiple parameters like its functionality, timing criticality, number of endpoints (destination module) etc.

The important question is how to decide number of flop stages required for these signals to travel from source 'Misc Logic' partition to 'Data, Command, Control' partitions. There are multiple factors which are taken into account to calculate this.

- **Floorplan** of our IP (Dimensions of the different partitions, Type/Name of each partitions) [Figure 1]
- **Distance** that the signal has to travel from source to destination partition
- **Single flop drive strength**

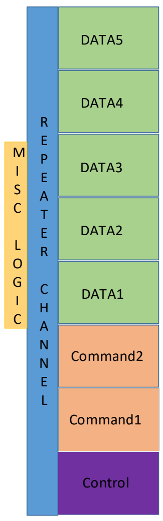


Figure 1. Reference HIP Floorplan

Problem 1: All signals cannot be flop repeated in same manner.

We need to come up with the custom structure for each signal to have the optimized number of flops. This structure can be a flop tree for multiple endpoints or a serial repeater chain (SREP) for single endpoint.

Problem 2: Finding a tree structure with least number of flops is very cumbersome.

We won't be talking about the serial chains here as they don't need any optimization. Number of flop stages in SREP, which is calculated based on geometry and flop drive strength, is fixed for that geometry and it doesn't have any room for optimization. For example, if a signal requires 5 stages to travel from partition A to partition B, then in a SREP we cannot have 4 stages as geometry itself requires 5 stages. However, if a signal is less timing critical and we want to add more than 5 stages, we have an option for that also.

There is a lot of scope for optimizing number of flops in the tree structure. This is critical as there are multiple mathematical ways to create the tree. Finding a tree structure with least number of flops is very cumbersome hence here we have developed an algorithm to solve this.

Traditionally we manually calculate number of repeater flop stages needed & then hand code the RTL for it, which is error prone, time consuming and not really scalable. The process involves putting a series of flops from source module till destination module with each flop branching into maximum of 2 flops. This binary flop generation is repeated till all the endpoints are connected.

The disadvantages of this approach are

- Increased Area & Power: More number of flops are used as the generated tree structure may not be optimal. This increased the area of the HIP.
- Manual & error prone: Time taken for development is more as each signal has to be hand coded.

We have developed an automated system which takes all the necessary inputs and does all the calculation and analysis. Our system comes up with an optimized tree structure and automatically generates a functional RTL module so the RTL we are getting is completely automated and also most optimized. On top of that, user also has flexibility to add additional flops for the signals that is not timing critical.

2. ART Framework

The system we have developed consists of 3 different utilities working together.

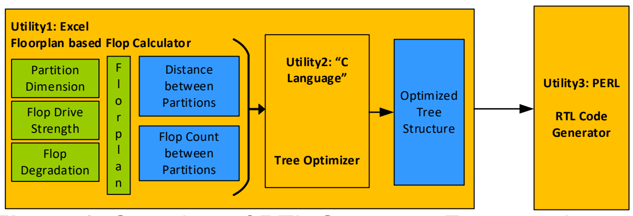


Figure 2. Overview of RTL Generator Framework

Utility 1 Floorplan based Flop Calculator: It is the master utility which controls the whole flow. This calculates the number of flops needed between source module and destination module based on the floorplan. It passes all these inputs to Tree Optimizer in '.csv' format.

Utility 2 Tree Optimizer: This is C based algorithm. Inputs are the number of flop stages and the degradation factors for various fanouts. Flop calculator then takes the optimum tree structure dumped by Tree Optimizer and then excel file is passed to the 'RTL Code Generator'.

Utility 3 RTL Code Generator: The RTL Code Generator generates a functional RTL code in .sv format and can be plugged in to create signal repetition between modules.

2.1. Floorplan based Flop Calculator

Below is the snapshot of the excel sheet in which we feed the required inputs and information to do the mathematical calculation.

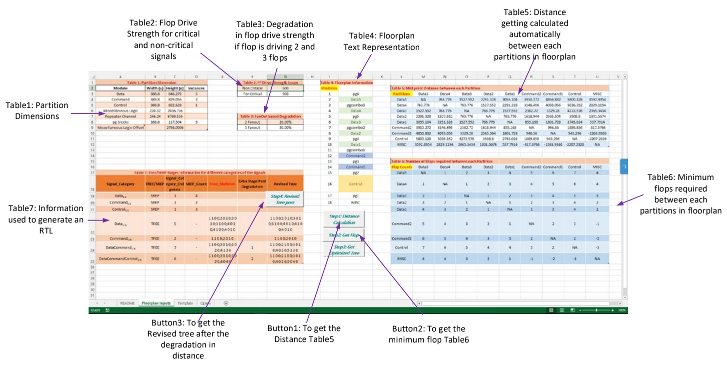


Table2: Flop Drive Strength for critical and non-critical signals
Table3: Degradation in flop drive strength if flop is driving 2 and 3 flops
Table4: Floorplan Text Representation
Table5: Distance getting calculated automatically between each partitions in floorplan
Table1: Partition Dimensions
Table6: Minimum flops required between each partitions in floorplan
Table7: Information used to generate an RTL
Button3: To get the Revised tree after the degradation in distance
Button1: To get the Distance Table5
Button2: To get the minimum flop Table6

Figure 3. Floorplan Inputs and Calculations

Calculations are done automatically with click of a button embedded in Excel Sheet. These buttons will run the VBA macro to generate the data.

Table1 consists of the dimensions of the partitions our IP is having.

Table2 is the drive strength of a flop when it is driving a single flop.

Table 3 indicates how much degradation the flop will have in its drive strength when it is driving 2 and 3 flops further down.

Table 4 is the textual representation of our floorplan which indicates which partition is at what position.

Button1 gives us the Table5 which is the midpoint distance between each partitions in floorplan.

We are using midpoint distance and not specific X/Y location of ports on the border as X/Y location of ports is very specific to Back-End and subject to change.

We want to have a little flexibility for the Back-End by not being too specific by having exact distance. Midpoint of partition gives us pessimistic distance which gives us extra margin in terms of distance. The ports interacting with North partitions are mostly located above the midpoint of the source partition boundary and those interacting with South partitions are mostly located below the midpoint of the source partition boundary.

There are 2 different cases for midpoint distance calculation.

<u>**Case 1**</u>: Distance between adjacent partitions (e.g. between Data5-Data4, Data5-Data3…Data5-Control).
To calculate center to center distance between partition x and partition y, following is the formula we used in the VBA macro.

$$Midpoint\ Distance = \left(\frac{x_{height}}{2}\right) + \left(\frac{y_{height}}{2}\right) + xy_{height}. \quad (1)$$

Where,
X_{height} = height of partition x
Y_{height} = height of partition y
XY_{height} = Sum of height of all partitions between x and y

<u>**Case 2**</u>: Distance between source partition (Misc Logic) to end partitions (data, command and control).
To calculate the center to center distance between Misc Logic and Partition Command1, we have used following formula.

$$Midpoint\ Distance = \left(rpt_{height} - \frac{cmd1_{height}}{2} - tillCmd1_{height}\right) - \left(\frac{MiscLogic_{height}}{2} + MiscLogic_{offset}\right). \quad (2)$$

Where,
rpt_{height} = Height of Repeater Channel partition
$cmd1_{height}$ = Height of Command1 partition
$tillCmd1_{height}$ = Sum of height of all partitions from top till Command1 partition
$MiscLogic_{height}$ = Height of Misc Logic partition
$MiscLogic_{Offset}$ = Distance between bottom edge of Repeater Channel and bottom edge of Misc Logic.

Button2 gives us the minimum number of flops required between each partition in Table6 which is based on Table5 and Table2. This will divide the distance we got in Table5 by the Table 2 numbers of 'For Critical' and 'Non Critical' Flop Drive Strength and round it up to the nearest ceiling value.

Button3 generates an optimized tree structure, which is an output of Tree Optimizer (.exe file), for all the predefined signal categories in Table 7. We are passing the .exe file to the excel macro running behind this button which runs the executable file within the macro and captures the output in Table 7.

Table7 consists of the initial optimized Tree structure given by our "C" based algorithm which takes .csv file as an input. We have divided all the input signals into some predefined categories based on the functionality of these signals. Tree structure for each of these categories will be different depending on the stages required and number of endpoints.

2.2. Tree Optimizer

Tree optimizer is a C based algorithm. It takes the distances between source and destination endpoints, flop drive strength and degradation factors as inputs and dumps the optimized tree structure for the given input. This is implemented based on the tree data structure. Here root of tree will be the source endpoint and all leaf nodes represent the destination endpoint. Each node in between root and tree is a flop. Tree Optimizer mainly has 3 phases.

Phase 1:

In phase 1 it finds the farthest endpoint (say x) from the source and creates a chain of flops to connect from the source module to x. In C, each flop is realized using a node of tree.

In Figure 4, MISC is the source module and we need to repeat the MISC signal to Data1, Data2, Data3, Data4, Data5 and Data6 modules. Farthest destination endpoint here is Data1 so in phase 1 we create a tree where the root is MISC and has only 1 leaf node - Data1. The number of levels of this tree will be equal to the flop drive length between MISC and Data1. The chain will look like below if the flop drive length is 6. In this phase each of the nodes (node1:6) are assigned its height position relative to the source endpoint.

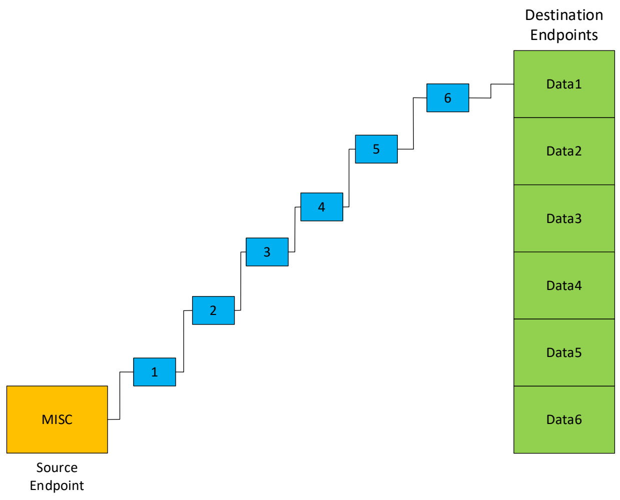


Figure 4. Tree Optimizer Phase 1

Phase 2:

In phase 2 the algorithm looks at the farthest endpoint from the leaf node of the main chain. In the above example the farthest is Data 6 (from Data1). It then calculate the number of flops required to connect to each of the nodes. i.e., the number of flops needed to branch from node 1 to Data6, node 2 to Data6 etc.

In the above example, number of flops needed to reach Data6

- From node 1 is 5 (To match the delay between Data1 and Data6).
- From node 2 is 4.
- From node 3 is 3 etc.

This calculation is done for all the destination endpoints. While doing the calculation there may be cases that from a node in main chain to Data6, it might take less flops than from that node to Data1. These nodes should not be considered as it will increase the delay for all the destination endpoints.

From the calculation, we find the node with least number of flops required to reach Data6. If this node has children less than the number of fanouts, then we add nodes till Data6, else next node, with least number of flops, is considered. If it takes least number of flops from node3, then the tree will look like below:

This process is repeated for all the destination endpoint till the full tree is found.

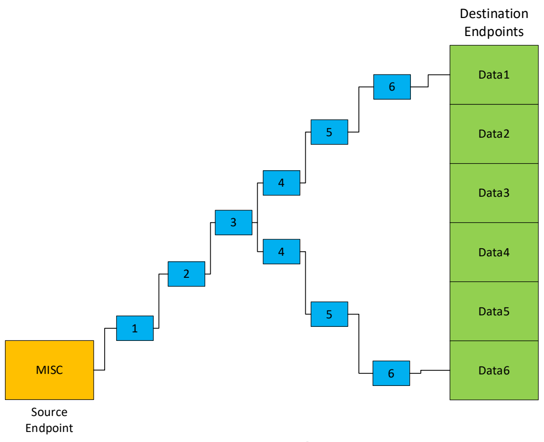
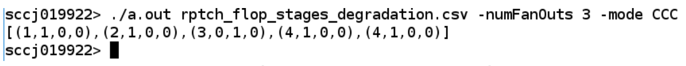


Figure 5. Tree Optimizer Phase 2

Phase 3

In this phase we have a tree structure with root at MISC and 6 leaf nodes – Data1:6. Now we parse through the tree to calculate the distance from the MISC to each node with degradation in consideration for each node because of one flop driving multiple endpoints. This distance is compared with the actual distance from the floorplan. If any of the node's calculated distance (by parsing the tree) is less than the actual distance (from floorplan), then the number of flops needed to reach from MISC to that flop is increased and rerun the program from Phase 1.



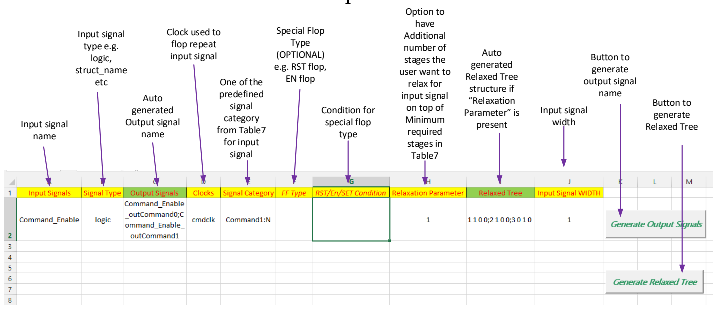
```
sccj019922> ./a.out rptch_flop_stages_degradation.csv -numFanOuts 3 -mode CCC
[(1,1,0,0),(2,1,0,0),(3,0,1,0),(4,1,0,0),(4,1,0,0)]
sccj019922> █
```

Figure 6. Tree Optimizer Phase 3 Output

So at the end of the Phase 3, we will get the most optimal tree structure which is passed to Table 7 in excel. Then the excel file is passed to RTL Code Generator. Refer Figure 6 for sample Tree Optimizer output.

2.3. RTL Code Generator

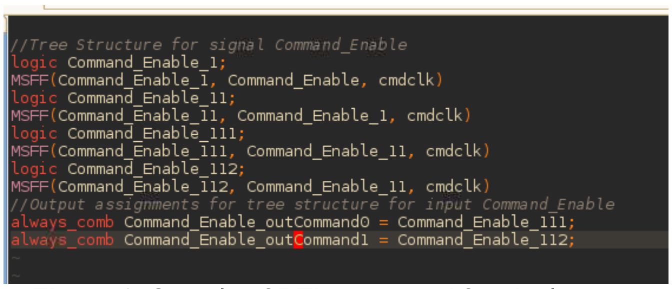
The PERL based script takes the tree structure and serial repeater count as input from Table 7 along with another tab in the excel sheet which consists of signal level information as shown in below snapshot.


Input signal name
Input signal type e.g. logic, struct_name etc
Auto generated Output signal name
Clock used to flop repeat input signal
One of the predefined signal category from Table7 for input signal
Special Flop Type (OPTIONAL) e.g. RST flop, EN flop
Condition for special flop type
Option to have Additional number of stages the user want to relax for input signal on top of Minimum required stages in Table7
Auto generated Relaxed Tree structure if "Relaxation Parameter" is present
Input signal width
Button to generate output signal name
Button to generate Relaxed Tree
Generate Output Signals
Generate Relaxed Tree

	A	B	C	D	E	F	G	H	I	J	K	L	M
1	Input Signals	Signal Type	Output Signals	Clocks	Signal Category	FF Type	RST/En/SET Condition	Relaxation Parameter	Relaxed Tree	Input Signal WIDTH			
2	Command_Enable	logic	Command_Enable_outCommand0;Command_Enable_outCommand1	cmdclk	Command1:N			1	1 1 0 0;2 1 0 0;3 0 1 0	1			
3													
4													
5													
6													
7													
8													

Figure 7. RTL Generator Utility

PERL script takes this excel sheet as input and generates an RTL module with optimized number of repeater flops. It will declare the input and output signals along with the intermediate logic declaration of the interconnect signals for tree. It will also assign the final stage tree outputs to the output ports. Below is the snapshot of RTL generated for signal (Command_Enable) mentioned in the excel sheet.



```
//Tree Structure for signal Command_Enable
logic Command_Enable_1;
MSFF(Command_Enable_1, Command_Enable, cmdclk)
logic Command_Enable_11;
MSFF(Command_Enable_11, Command_Enable_1, cmdclk)
logic Command_Enable_111;
MSFF(Command_Enable_111, Command_Enable_11, cmdclk)
logic Command_Enable_112;
MSFF(Command_Enable_112, Command_Enable_11, cmdclk)
//Output assignments for tree structure for input Command_Enable
always_comb Command_Enable_outCommand0 = Command_Enable_111;
always_comb Command_Enable_outCommand1 = Command_Enable_112;
~
~
```

Figure 8. Sample of RTL generated for single signal

The beauty of this is whenever there is a change in the tree structure (because of updated flop drive strength or change in the position of one of the partition), the updated RTL will be generated automatically with a click of a button. Designer won't have to go through the hassle of breaking the existing connections to include updated repeater structure. If new signals are added then integration of those will be required.

This will be applicable to all the signals going from one partition to another. If the signal is travelling from source partition to destination directly without any flops, it will be a feedthrough signal. Such signal can be included in respective signal category and will be taken care by simple assign statement.

3. Results

Following tree diagrams and tables show the comparison in number of flops seen using traditional approach vs the new automated approach mentioned in the paper.

3.1. Un-optimized Traditional Tree

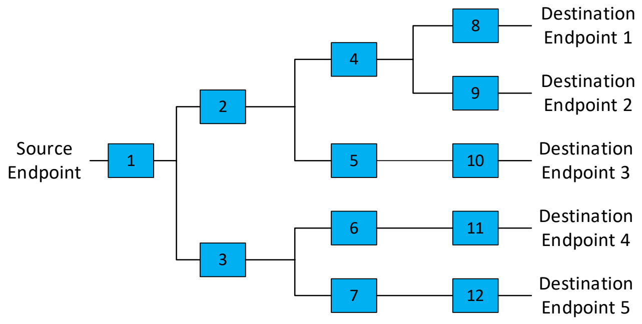


Figure 9. Un-optimized Tree Structure for a signal category Data1:N

Table 1. Number of flops without optimized tree

Signal Category	Number of Endpoints	Max Flop Stages	Flop Number for a Signal	Total number of flops (5 * Flop Number for a Signal)
Data1:N	5	4	12	60
CCC1:N	2	2	3	15
DataCommand1:N	7	4	28	140
DataCommandControl1:N	8	6	31	155
			Total Flops	**370**

3.2. Optimized Tree That ART has Generated

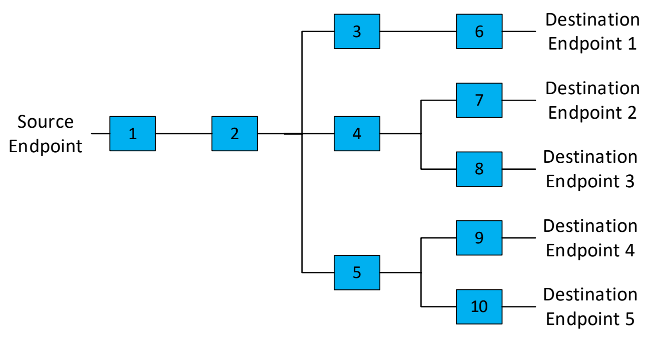


Figure 10. Optimized Tree Structure for a signal category Data1:N

Table 2. Number of flops with Optimized Tree

Signal Category	Number of Endpoints	Max Flop Stages	Flop Number for a Signal	Total number of flops (5 * Flop Number for a Signal)
Data1:N	5	4	10	50
CCC1:N	2	2	3	15
DataCommand1:N	7	4	14	70
DataCommandControl1:N	8	6	25	125
			Total Flops	**260**

3.3. Area Savings with New Automated Approach

Table 3. % Savings Calculation

Signal Category	Total number of flops for 5 signals (Un-Optimized)	Total number of flops for 5 signals (Optimized)	% Savings
Data1:N	60	50	16.66%
CCC1:N	15	15	0%
DataCommand1:N	140	70	50%
DataCommandControl1:N	155	125	19.35%
	370	**260**	**29.73%**

The Optimization & Area savings increase as the HIP designs

- Become more complex and hence floorplan/dimension increases.
- This causes the number of repeater stages to increase. (time-consuming & error prone)
- Hence increasing the optimization brought in by Tree optimizer utility.

4. Summary

This complete system, consisting of all 3 utilities, can be used for multiple projects in Intel in which this kind of repeater flops are required. With our solution we have designed a tool that

- Takes the floorplan information, distance from the source module to the destination and single flop drive strength as the input.
- Algorithmically finds out a tree structure with minimal number of flops.
- Generates the optimal functional RTL FUB that is ready to plug in.

The advantages of our solutions are:

- **Less Area**: This approach generates the least number of flops needed for a given source endpoint to destination endpoint. Thus area and flop delay are optimized.
- **Automated**: It needs minimal manual effort and it generates a functional RTL which is ready to plug-in. Thus the development is faster, easy to use and less prone to errors.
- **Scalable**: Adding new signals just requires adding the corresponding fields in the input to the tool and rerun the program again.
- **Fast development**: Using tool to generate automated RTL, aids quicker development & faster simulation readiness.

Optimized & automatic RTL, generated using the tool described in this document, is being intercepted in one of the Intel's IPs.

Acknowledgements.
We would like to thank Krishnamurthy B Venkataramana and Rajeev Gopalan for motivating and guiding us to create this system. We would like to thank Satheesh Pillai for letting us know the challenges back-end team is facing to close the timing on such module. We would like to thank Ryan Joseph who provided us more insights of the problems encountered in such module. We would like to thank Alex P Thomas, who is the first user of this tool, for providing us valuable feedback which made us improve our system. We would like to acknowledge the effort of Aman Aggarwal, who had helped us to develop few VBA macros for tool.

References

[1] Eng Keong Teh; Mohamad Adzhar Md Zawawi; Mohamed Fauzi Packeer Mohamed; Nor Ashidi Mat Isa, "Access-Practical System-on-a-Chip Flop Repeater Insertion with a Meta-heuristic Technique", 2018, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems