Minimizing Clock Domain Crossing in Network on Chip Interconnect

Parag Kulkarni[1], Puneet Gupta[2], Rudy Beraha[3]
[1]Synopsys, 1101 Slater Road, Durham, NC USA
[2]Department of Electrical Engineering, University of California, Los Angeles
[3]Qualcomm, 5775 Morehouse Drive, San Diego, CA USA
E-mail: paragk@ucla.edu

Abstract— Network-on-Chip (NoC) architectures have been widely adopted as the preferred solution to the communication challenges of System-on-Chip (SoC) design in the nanoscale regime. SoC designs often incorporate custom NoC architectures that do not conform to regular topologies. This requires the generation of a power and resource efficient interconnection architecture that can support the communication requirements for the SoC with the desired performance. Hence automated topology generation tools optimize for mutiple objectives like power and area to synthesize a NoC topology that meets these communication constraints. Clock-Domain-Crossings (CDCs) is an important consideration in fabric design that gets ignored in existing topology tools. CDCs add to latency and area, while also increasing verification and implementation effort. In this paper we propose a method to model and optimize the clock-domain-crossings (CDC) during NoC topology generation and prove that the underlying problem is NP-hard. We present and compare multiple approaches to model the CDC cost, including a novel fast heuristic. When applied within an existing topology generation tool our approach results in a $5\% - 39\%$ reduction in flop count of the resulting interconnect while still satisfying the original communication/performance constraints.

I. Introduction

Network-on-Chip (NoC) architectures have been proposed as promising alternatives to traditional bus-based and point-to-point communication systems in the era of shrinking feature sizes [1, 2, 3]. NoCs achieve communication among various cores through on-chip micro-network components (such as routers and network interfaces) instead of the tradional non-scalable buses. The modularity of NoCs allow for better design predictability besides offering lower power consumption and greater scalability. NoCs can be designed as regular or application-specific/irregular network topologies. Regular topologies have been successfully employed in a number of tile based multiprocessor projects [4, 5]; this works well due to processor homogenity and application traffic variability. On the other hand, for custom SoC applications, the design challenges are different and usually result in varied core sizes, irregularly spread core locations, and different communication bandwidth requirements, making an application-specific irregular network architecture customized to the needs of the application is more appropriate.

The topology generation problem can be defined as follows : ***given*** *a set of n cores* $C = \{c_1, c_2, ..., c_n\}$*, a constraint on the number of routers m, a core communication graph (CCG) and a power/area model for the network components,* ***find*** *a NoC topology that satisfies the communication constraints while minimizing area and power consumption of the resulting network.* The application specific network topology generation problem is a variation of the generalized steiner forest problem [6], which is known to be NP hard. Researchers have usually utilized linear-programming (LP) [7, 8], simulated-annealing (SA) [9,10] or heuristic optimization [11,12,13,14,15] based techniques to solve the topology generation problem. Early approaches [13, 8] focused on minimizing the dual objectives of area and power by using cost functions expressed as weighted linear combinations of area and power. More recent approaches [7,12,9,15] have begun incorporating increasing levels of physical information such as estimates of wire-length and floorplan area, with [9] integrating physical floorplanning and NoC topology generation to make use of more realistic interconnect information.

While cost and optimization functions have become more complex by adding multiple objectives, no existing work explicitly looks to optimize for CDC during topology generation. CDCs are needed in NoC fabrics as a result of various subsystems of an SoC operating on different clock domains. These CDCs are a major design consideration and have to be accounted for at each step of the ASIC design flow, all the way from RTL design to physical implementation. Reduction in CDCs in the NoC fabric have multiple benefits including -

- Improving Latency : the lack of a domain crossing can result in better end to end latency through that link by avoiding any delays needed for synchronization of signals.
- Reducing Area : as shown by our results in Section **6** the optimization of CDCs in NoC fabrics leads to a significant reduction in the number of flops used, leading to significant savings in silicon real-estate.
- Reduction in Verification Effort : each CDC has to be verified, effort is needed on both simualtion and formal verification fronts [16].
- Reduction in Physical Design Effort : the skew of bits crossing CDC boundaries has to be carefully managed, often requiring manual intervention and checks [17].

Hence *the optimization of CDC during the synthesis of application-specific NoCs is the focus of this paper.* Section **2** introduces the background behind CDCs in NoCs, in Section **3** we define CDC optimization as a *Router Coloring Problem* and prove it to be NP-hard. Three different ap-

proaches are discussed in Section **4** and their relative merits compared in **5**. The benefits of integrating CDC optimization into a NoC topology generation tool are reported in Section **6**, and we conclude in Section **7**.

II. CDC in Network on Chip Interconnect

A clock domain is defined as a part of the design that is driven by a single clock or clocks that have constant phase relationships. Domains that have clocks with variable phase and time relationships are considered different clock domains. Today's systems have a multitude of components working with different clock domains running at varying speeds. Signals that cross these clock-domain boundaries commonly referred to as clock domain crossings, or CDCs, have to be synchronized before they can be safely used in the receiving domain. Figure 1 shows an example of clock domain crossings occuring in a NoC fabric. When the asynchronous clock domains A (in red) and B (in blue) interact at the highlighted edge, data crosses a clock domain boundary and a CDC is incurred. The data crossing the domain boundary needs to be synchronized (to the new clock domain) before usage in the receiving domain. This is done usually by either dedicated Synchronizer library cells (typically for single bit or small set of signals) or using an ASYNC FIFO (for multiple bits and wide data-buses) [18, 16].

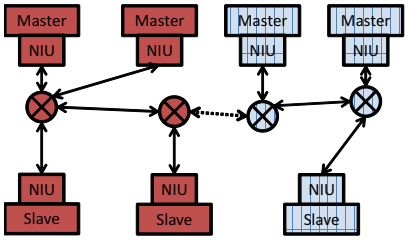


Fig. 1: Clock Domain Crossing in a NoC fabric. The figure shows Master, Slave and Network Interface Units. The highlighted edge represents a CDC.

Existing NoC topology aproaches such as [7,10,9,12] generate a network topology but do not assign clock-domains (or frequencies) to their constituent routers. This job is left to the designer to do by visual inspection and taking into consideration the clock domains of the various cores (Masters and Slaves) which are known beforehand. While it is feasible for the designer to assign clock-domains (or frequencies) to each router in small designs, this will be increasingly harder as SoC sizes increase. For example the 4G Modem SoC presented in [10] consists of 16 Masters and 18 Slaves with between 4 and 10 routers, even the presence of 3 or 4 clock domains would leave the designer with a large set of possible router frequency assignments. More importantly by optimizing for CDC during topology generation, the tools can generate a topology that still satisfies the performance/latency requirements, optimizes for area/power but also has a lower number of CDCs. Fewer number of CDCs in the fabric is highly desireable for a number of reasons highlighted in Section **1**.

III. The Router Coloring Problem

The optimization of CDC in a NoC requires for the modeling of a *CDC cost* and its inclusion in the optimization function of the SA [10,9] or GA [12] algorithms of the topology generation tools. This cost can be measured as the number of CDCs itself or as a secondary objective such as number of flops needed in the fabric. To compute such a cost requires an algorithm to assign frequency to every router in a given topology, such that the number of CDCs (or a secondary metric) is minimized. Given that this algorithm sits in the inner-most loop of the SA/GA algorithms, it is imperative that we have a fast yet accurate way to come up with a frequency assignment for each router that minimizes the CDCs.

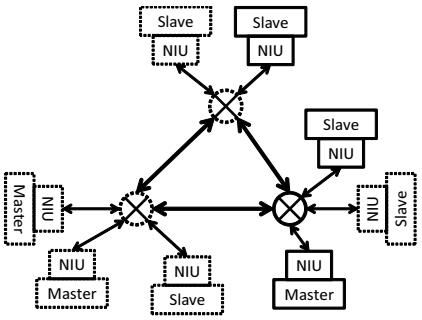
More formally, an algorithm is needed such that : ***Given*** *a set of cores (both masters and slaves)* $C = \{c_1, c_2, ..., c_n\}$, *each with a frequency* $f_i \in F = \{f_1, f_2, ..., c_k\}$ *and a set of routers* $R = \{r_1, r_2, ..., r_m\}$ *that connect them in a topology* T. ***Assign*** *a frequency* $f_j \in F$ *to each router* $r_j \in R$ *such that the number of CDCs in* T *are minimized.*

By representing each clock domain by a unique color the problem can also be reformulated as the *Router Coloring Problem : **Given** a graph of interconnected nodes, some colored (cores) and some uncolored (routers). **Color** the uncolored nodes such that the number of edges that connect two nodes of different colors is minimized* . Figure 2 demonstrates the conversion of NoC fabric with CDCs into the forumation specified by the Router Coloring Problem. CDCs are now represented by any edge that connects vertices of different color. As the Figure 2(b) highlights, CDCs can be of two types : on connections between cores and routers or connections between two routers.

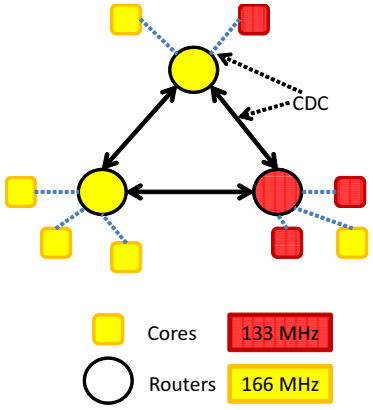
Router Coloring is a variation of the Multiterminal Cut problem, which is defined as follows : ***Given*** *a Graph* $G = (V, E)$, *a set* $S = \{s_1, s_2, ..., s_k\}$ *of* k *specified vertices or* terminals *and a positive weight* $w(e)$ *for each edge* $e \in E$, ***find*** *a minimum set of edges* $E' \subseteq E$ *such that the removal of* E' *from* E *disconnects each terminal from all the others.* Multiterminal cut is NP-hard for arbitary graphs for any $k > 2$ [19]. A *solver* for the router coloring problem can be used to solve the multiterminal-cut problem using the following mapping :

- Each terminal becomes a core with a unique color in the NoC.
- All other vertices are routers and all edges remain the same.
- Solving router-coloring on the thus created network assigns a color to each router.
- Let the set of edges in the colored graph, between any two vertices of different colors be E''. Since router coloring minimizes the number of edges between different colors E'' is the multiterminal cut E'.

Hence **multiterminal cut is polynomial time Turing reducible to router coloring**, making router coloring NP-hard for arbitary graphs. Figure 3 shows an



(a)Original Network. The dotted lines represent elements running at 166MHz and the solid lines those at 133MHz.



(b)Network represented in the Router Coloring Formulation. Colors represent the two frequencies.

Fig. 2: Router Coloring Example. Showing CDCs on router to core connections as well as between two routers.

example of this transformation. While multiterminal cut can be solved in polynomial time for planar graphs and a fixed k [19], but the solution which is exponential in k is prohibitively slow and can only be solved in $O((4k)^k n^{2k-1} logn)$, making it unsuitable for router coloring.

Note that while there is no such constraint in our formulation of the router-coloring problem above, in real world applications direct connections between cores (nodes of fixed/pre-assigned color) are unlikely (or possibly prohibited) in the NoC. The router coloring problem is NP-hard even with such a constraint since a *solver* for router-coloring with such a restriction (no direct connections between terminals/cores allowed) can still be used to solve multiterminal cut once all edges that connect any two terminals are removed (these are by definition part of the final cut). Any edges connecting terminals can be removed in $O(n^2)$ ensuring that multiterminal cut is still polynomial time Turing reducible to router coloring even in the presence of such a constraint.

Exploring all possible configurations of routers with all possible colors (frequencies) available in the network, would require $O(numClocks^{numRouters})$. Where $numClocks$ is the number of unique clock domains used in the SoC and $numRouters$ is the number of routers present in the NoC fabric. To benchmark the other algorithms and approaches we present in the following sections, we also implemented a brute force technique (that explores all possible configurations) for comparison purposes (Section **5** has further details). We found that this approach is slow, taking multiple hours to evaulate a single configuration once the number of routers was greater than ten. Given that we intend to run this in the inner-most loop of a SA/GA algorithm, such large runtimes would not be acceptable. Hence we focus on approximate or near-optimal solutions to the router coloring problem that can then be used effectively to guide NoC topology synthesis.

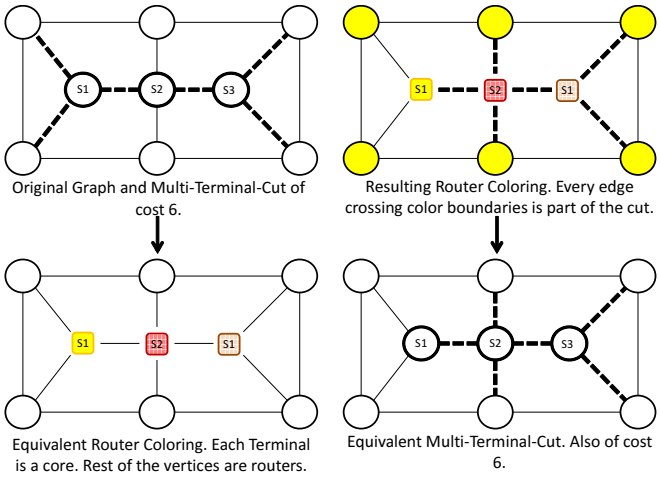


Fig. 3: Transformation of a multi-terminal cut problem to an equivalent router coloring setup and vice-versa.

IV. Approaches to the Router Coloring Problem

In this section we introduce three different approaches to the router coloring problem introduced in Section **3**, each with varying optimality and runtime tradeoffs (that we cover in Section **5**). The first and second approaches model the router coloring problem as integer linear programming and partitioning problems respectively, and leverage existing solutions to those problems to assign color to each router. The ILP approach is optimal and always results in a *coloring* that is optimal, the partitioning approach on the other hand is occasionally sub-optimal. The last approach is a fast heuristic that colors the network one router at a time.

A. Integer Linear Programing

The NoC Router Coloring problem can also be setup and solved as an integer linear programming (ILP) problem. There are $N \times M$ variables in the system, where M is the number of unique colors (frequencies) in the network and N is the number of routers. Given unique colors $C = \{c_1, c_2, ..., c_m\}$ and routers $R = \{r_1, r_2, ..., r_n\}$ in the network, then $X = \{x_{11}, x_{12}, ..., x_{nm}\}$ are the variables in

the corresponding ILP system of equations. Each of these variables x_{ij} represents whether or not the router r_i is assigned the color c_j in the final networking coloring. That is if x_{ij} is equal to one, then the router r_i is painted with the color c_j. Of course each router can only be painted with exactly one color, leading to the following constraints on the variables :

- Each of the variables x_{ij} must be either zero or one, that is each $x_{ij} \in [0, 1]$.
- Exactly one of the variables associated with any given router r_i, $X_i = \{x_{i1}, x_{i2}, ..., x_{im}\}$, must be one and all the others must be zero. That is $\sum_{j=1}^{M} x_{ij} = 1$ for every router r_i.

The objective function of the ILP is modeled by CDC_{NOC} and is the focus of the rest of this section. Every router is represented by a single equation, which reflects the colors of its connected cores. In addition every connection between two routers is modeled with its own unique equation, which is a function of the color of both the connected routers. The sum of each of these equations represents the objective function to be minimized by the solver. We explain this in detail with the help of a simple example network shown in Figure 4.

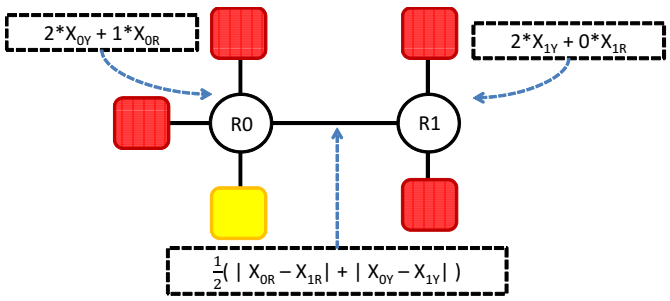


Fig. 4: Example of Linear Equations for NoC Router Coloring.

The network in Figure 4 has cores of two colors Red and Yellow, resulting in two variables associated with each router R0 and R1. The variables X_{0R} and X_{0Y} represent the router R0 being colored Red and Yellow respectively. Similarly X_{1R} and X_{1Y} represent the router R1 being colored Red and Yellow respectively. Equation 1 shows the set of equations for our sample network in Figure 4. The equations that model the routers themselves are simple, in our example if R0 is colored Yellow there are two CDCs between R0 and its connected (Red) cores. Coloring R0 Red leads to a single CDC between R0 and its connected (Yellow) core. The equation for R1 is derived in a similar manner. The equation modeling the connection between the two routers aims to account for the fact that if both are identically colored (either both Red or both Yellow) the CDC cost is zero, in all other situations the CDC cost is one.

$$\begin{aligned} CDC_{R0} &= 2 \times X_{0Y} + 1 \times X_{0R} \\ CDC_{R1} &= 2 \times X_{1Y} + 0 \times X_{1R} \\ CDC_{R0R1} &= \frac{1}{2} \times (|X_{0R} - X_{1R}| + |X_{0Y} - X_{1Y}|) \\ CDC_{NOC} &= CDC_{R0} + CDC_{R1} + CDC_{R0R1} \end{aligned} \quad (1)$$

Given these equations modeling the cost and objective function, the linear program for optimizing the CDC in the network in Figure 4 can be setup as :

$$\begin{aligned} &\text{Minimize } CDC_{NOC} \\ &\text{s.t.} \quad \{X_{0Y}, X_{0R}, X_{1Y}, X_{1R}\} \in [0, 1] \\ &\qquad X_{0Y} + X_{0R} = 1 \\ &\qquad X_{1Y} + X_{1R} = 1 \end{aligned}$$

To solve the ILP we use the solver from the *lp_solve* programming library [20] which uses the branch-and-bound method for ILP. Note that the absolute values in Equation 1 can be converted to traditional ILP format using the common technique explained in [21].

B. Partitioning

With little modification the *Router Coloring problem* can also be modeled as a traditional partitioning problem :

- Each clock-domain/color in the NoC corresponds to exactly one unique partition.
- As a result all cores have pre-assigned and fixed partitions (which correspond to their clock domain or color).
- Only the routers in the NoC can *move* and be assigned partitions (which will correspond to their assigned clock domain or color).

We use the approach and implementation provided by [22] to solve the partitioning problem we set up for Router Coloring. The complexity of this approach is $O(n + m + klog(k))$, where n is the number of routers, m the number of edges and k the number of partitions/colors. The results from partitioning are compared with the other two approaches (and a brute force solution where possible) in Section **5**.

C. Approximate Heuristic

The last approach we demonstrate uses a simple heuristic to select which router to color (assign frequency) next, with the objective of minimizing the number of CDCs in the resulting networking coloring. It builds on the following principles -

- For a single router connected only to other cores (and not to any other routers), selecting the most common color amongst its connections minimizes the number of CDCs.
- It is *easier* to make a choice of color on a router that has a larger percentage of its connections already colored.

The Algorithm 1 descibes the heuristic in detail. At each step or iteration the approach seeks to color those routers for which we have the *most known information*. That is

those routers for which we know the color of the higest percentage of its connections. Ties at any step are broken by selecting the color that is more common amongst the cores in the graph. It then colors each such router so as to minimize the CDCs on the basis of the connections that have already been colored. Upon coloring the current router, all its connected routers are updated with this new information before preceeding to the next router/iteration. The algorithm effectively maintains a priority queue of which routers to color next, prioritized on the basis of the *most known information*. While the algorithm described here assumes every connection/link has the same weight, that might not always be the case, for example the data widths of some connections maybe 64 bit others 32 bit. For such situations we adapt our approach to choose a color that minizmes the total *cost* of CDCs instead of the *number* of CDCs.

The while loop in Algorithm 1 executes once for each router, finding the router with the highest $knownInfo$ is $O(n)$, painting the selected router is $O(n+k)$ and the for loop to update each connected router is $O(n)$. This algorithm has a runtime complexity of $O(n^2+nk)$, where n is the number of routers in the NoC and k the number of colors. The heuristic runs significantly faster than the other approaches we present, a detailed comparison is provided in Section **5**.

Figure 5 shows how the algorithm would work on two simple examples, showing the step by step coloring of each individual router. As described in Algorithm 1 at each step the router we know most about is colored and ties are broken by selecting the more common of the two colors (in the network). Figure 5(a) displays the advantage of breaking ties with the more common color in the network. In the first step router R1 could have been colored either Red or Yellow since R1 has two known colored connections that that point, one Red and one Yellow. The tie is broken with the selection of the more common color in the network which is Red. Resulting in the final coloring with CDC cost of one, painting R1 Yellow would have led to at least two CDCs in the final colored network. Figure 5(b) demonstrates the motivation on coloring routers in the order of *most known information* (the percentage of colored connections). Router R2 is colored last, when we know the colors of both routers R1 and R3 leading to a final networking coloring with just two CDCs. Coloring R2 first would have led to it being colored Yellow, which would result in at least three CDCs.

Algorithm 1 Approximate Heurisitic for Router Coloring

Require: Network Topology, including router to router and router to core connections. Color of each core.
  //Compute the colors/clocks connected to each Router
  for all Routers $r \in R$ **do**
    Compute connected colors for r
    $r.knownInfo \leftarrow \frac{r.numConnectedCores}{r.numTotalConnections}$
    //Initially we know the colors of only the cores
  end for
  $L \leftarrow List\ of\ Routers$
  //Color each router iteratively.
  while Uncolored routers remain in L **do**
    $r \leftarrow Router\ with\ highest\ knownInfo\ in\ L$
    //Paint with most common connected color
    //Break ties with more common color in network
    $r.color \leftarrow mostCommonConnectedColor$
    //Now update routers connected to r
    for all Routers c *connected to* r **do**
      Update connected colors for c with $r.color$
      //We now know one more color connected to c
      Update $c.knownInfo$
    end for
    $delete\ r\ from\ L$
  end while

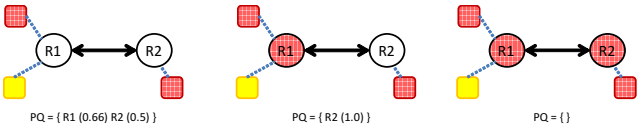
V. Runtime and Optimailty Comparison

In this section we compare the runtime and optimality of the approaches to Router Coloring introduced in Section **4** with each other and a brute force approach (that finds the optimum by enumerating each possible configuration) where possible. For optimality comparison we use a suite of topologies which are a mixture of randomly generated and common NoC topologies. We use the ring, star, mesh and torus topologies shown in [23] and introduce a random number of colors into the network. Table I shows the result of the various approaches on these regular topolgies while Figure 6 shows the comparison between the optimality of the different approaches on the entire suite of topologies sorted by network size (number of routers).

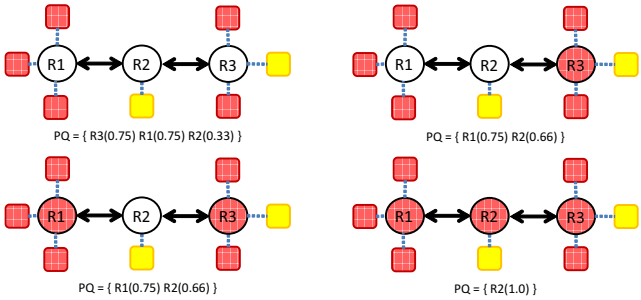
The graph in Figure 7 tracks the sub-optimality of our heuristic compared to the optimal solution generated by ILP. For this purpose we generate a suite of completely random topologies with varying number of routers (fixing the number of colors to four), we generate five unique topologies for each network size (number of routers) and track the average sub-optimality with increasing network size. On the basis of these experiments the following observations can be made :

- The partitioning approach is optimal for all but a couple of the topologies. The few sub optimalities can be attributed to the partition balancing in [22].
- **The ILP approach is optimal**, the ILP is solved optimally by the branch and bound solver. It leads to a coloring of mimimum cost that matches the brute force result on every tested topology both random and manual.
- The heuristic approach is the least optimal, lying between $5\% - 10\%$ for networks with less than fifteen routers. Its average sub-optimality increases to almost 13% for the largest networks of close to twenty routers.
- Table I suggests that all the approaches are nearly optimal on regular common NoC topologies.

Runtime is the other metric of interest for comparison between the various approaches from Section **4**. We use the random topologies from the sub-optimality experiment to also track runtime of the approaches, highlighted in Fig-



(a)Step by step router coloring and priority queue at each stage.



(b)Step by step router coloring and priority queue at each stage.

Fig. 5: Example operation of the coloring heuristic.

ure 8. The partitioning aproach is the slowest, taking twice as much time as the ILP solution. The heuristic solution is the fastest, more than 200X faster than the partitioning approach on average. For moderate network sizes of between ten and fifteen routers the partitioning and ILP approaches take between $20-30ms$ to compute the coloring. For SA based topology generation tools such as [10] which we use for our experiments in Section **6** this might prove to be a significant overhead depending on the number of iterations, since the coloring algorithm will be included in the inner most loop of the optimization. The tool in [10] runs between ten thousand to a million iterations for topology generation, using the partitioning or ILP would result in a runtime increase of five to five-hundred minutes. Hence we use the heuristic approach for router coloring in our experiments in the next section.

TABLE I: ROUTER COLORING TECHNIQUES APPLIED TO COMMON NOC TOPOLOGIES

Topology	Network		ILP	KWAY	Partitioning	Optimal
	Routers	Colors				
Ring	4	3	8	9	8	8
Ring	8	4	18	19	18	18
Star	7	6	12	12	12	12
Star	5	4	8	8	8	8
Mesh	9	3	14	16	14	14
Mesh	16	3	29	30	29	29
Mesh	6	3	11	11	11	11
Torus	9	3	17	17	17	17
Torus	9	4	22	23	23	22
Torus	9	5	36	38	36	36

VI. Results

In this section we discuss the results of integrating our CDC optimization approach into an existing NoC Topology Generation tool that was introduced in [10]. It optimizes for both power and performance using a SA optimization technique. The tool aims at generating a topology with

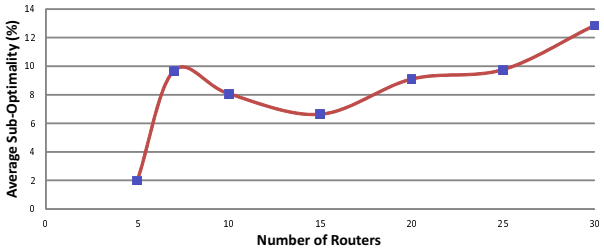


Fig. 7: Heuristic Sub-Optimality on random topologies with increasing Network Size for a fixed number of colors.

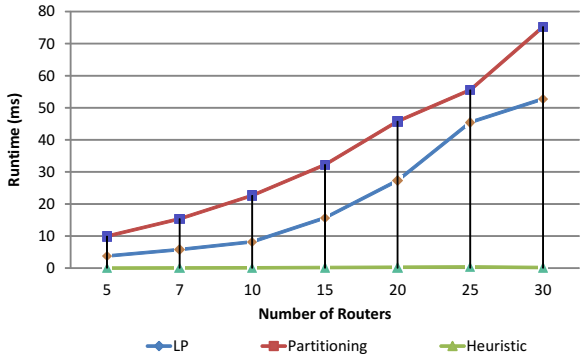


Fig. 8: Runtime Comparison. Five unique random topologies are used for each network size and average runtimes are reported.

minimal power consumption which meets the required latency demands of the application. Starting from a random initial topology the tool performs swaps between cores to generate new topologies with lower cost, while ensuring that latency constraints between cores are not violated. It is to this cost that we add our CDC optimization cost, modeling it as flops infered in the network. The tool is written in MATLAB, so we ported our heuristic based optimization approach to MATLAB and include it in the Simulated Annealing cost function.

We run two industrial benchmarks SoC designs with our CDC optimization enabled, Table II documents the results of the CDC optimization. We see a $5-39\%$ reduction in flop count in the NoC fabric, while still satisfying the original latency constraints. The design Chip2 in Table II is an updated version of the 4G ASIC meant for use in femto cells and base stations introduced and studied in [10]. Chip1 is an application processor based SoC for intended for use in mobile devices. Note that the significant increase in runtime is due to MATLAB related limitations enforced by the source code of the original tool, we expect significantly smaller runtime overhead in a C++ based implementation.

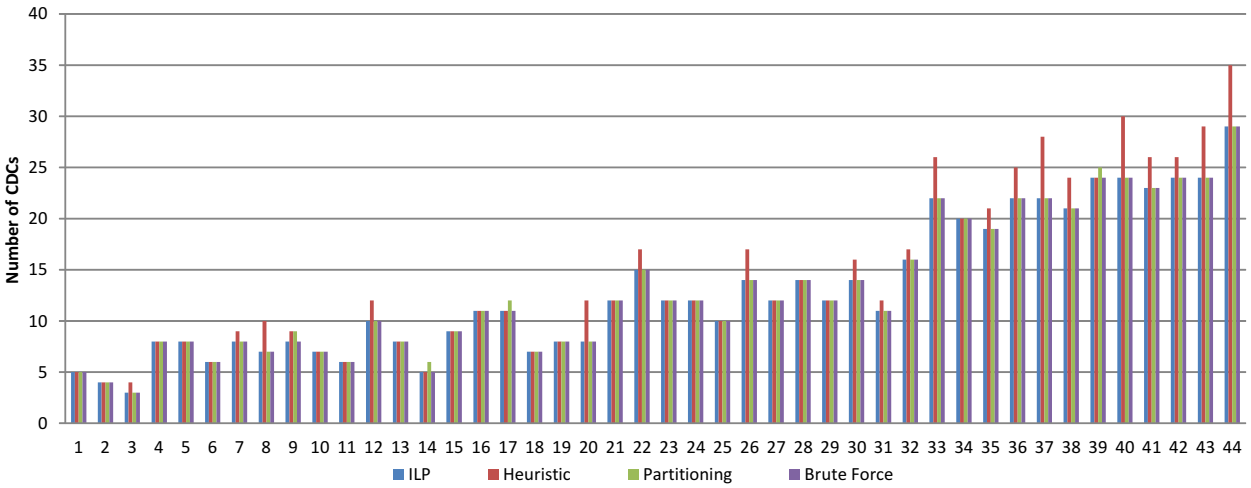


Fig. 6: Optimality Comparison between various Router Coloring techniques, sorted by number of routers. Both random and common NoC topologies.

TABLE II: FLOP COUNT REDUCTION IN TWO SOC BENCHMARKS BY CDC OPTIMIZATION

Name	Number of Cores	Network Type	Original Flop Count	Optimized Flop Count	Change	Runtime
Chip1	21	Response	5624	3424	-39%	+12%
	21	Request	5568	3616	-35%	+9%
Chip2	45	Response	10432	7424	-29%	+14%
	45	Request	7680	7290	-5.3%	+16.6%

VII. Conclusion

In this paper we propose the first CDC reduction method for NoC topology generation. With a $5\% - 39\%$ reduction in flop count on large real application specific NoCs from industry, the approaches presented allow for a large reduction in CDCs in the fabric while still preserving the original latency constraints of the fabric. We prove that the underlying router-coloring problem is NP-hard and present a heuristic that outperforms other approaches on the crucial metric of runtime by 200X, while remaining within 10% off the optimal solution. Future work includes greater analysis of the relationship between Network characteristics and optimality, exploring further heuristics along with integration into more openly available NoC topology tools. We also plan to explore and evaluate the effect of CDC-optmization on power and performance tradeoffs in the resulting NoC architecture.

References

[1] L. Benini and G. De Micheli, "Networks on chips: a new soc paradigm," *Computer*, vol. 35, no. 1, pp. 70 –78, jan 2002.

[2] R. Marculescu, U. Ogras, L.-S. Peh, N. Jerger, and Y. Hoskote, "Outstanding research problems in noc design: System, microarchitecture, and circuit perspectives," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 28, no. 1, pp. 3 –21, jan. 2009.

[3] D. Bertozzi, A. Jalabert, S. Murali, R. Tamhankar, S. Stergiou, L. Benini, and G. De Micheli, "Noc synthesis flow for customized domain specific multiprocessor systems-on-chip," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 16, no. 2, pp. 113 – 129, feb. 2005.

[4] M. Taylor, J. Kim, J. Miller, D. Wentzlaff, F. Ghodrat, B. Greenwald, H. Hoffman, P. Johnson, J.-W. Lee, W. Lee, A. Ma, A. Saraf, M. Seneski, N. Shnidman, V. Strumpen, M. Frank, S. Amarasinghe, and A. Agarwal, "The raw microprocessor: a computational fabric for software circuits and general-purpose programs," *Micro, IEEE*, vol. 22, no. 2, pp. 25 – 35, mar/apr 2002.

[5] K. Sankaralingam, R. Nagarajan, H. Liu, C. Kim, J. Huh, D. Burger, S. Keckler, and C. Moore, "Exploiting ilp, tlp, and dlp with the polymorphous trips architecture," *Micro, IEEE*, vol. 23, no. 6, pp. 46 – 51, nov.-dec. 2003.

[6] S. S. Ravi, M. V. Marathe, D. J. Rosenkrantz, S. S. Ravi, H. B. Hunt, and III, "Approximation algorithms for degree-constrained minimum-cost network-design problems," 2001.

[7] K. Srinivasan, K. Chatha, and G. Konjevod, "Linear-programming-based techniques for synthesis of network-on-chip architectures," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 14, no. 4, pp. 407 –420, april 2006.

[8] ——, "Linear programming based techniques for synthesis of network-on-chip architectures," in *Computer Design: VLSI in Computers and Processors, 2004. ICCD 2004. Proceedings. IEEE International Conference on*, oct. 2004, pp. 422 – 429.

[9] B. Yu, S. Dong, S. Chen, and S. Goto, "Floorplanning and topology generation for application-specific network-on-chip," in *Design Automation Conference (ASP-DAC), 2010 15th Asia and South Pacific*, jan. 2010.

[10] R. Beraha, I. Walter, I. Cidon, and A. Kolodny, "Leveraging application-level requirements in the design of a noc for a 4g soc - a case study," in *Design, Automation Test in Europe Conference Exhibition (DATE), 2010*, march 2010, pp. 1408 –1413.

[11] A. Tino and G. Khan, "Multi-objective tabu search based topology generation technique for application-specific network-on-chip architectures," in *Design, Automation Test in Europe Conference Exhibition (DATE), 2011*, march 2011, pp. 1 –6.

[12] G. Leary, K. Srinivasan, K. Mehta, and K. Chatha, "Design of network-on-chip architectures with a genetic algorithm-based technique," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 17, no. 5, pp. 674 –687, may 2009.

[13] K. Srinivasan and K. Chatha, "Isis: a genetic algorithm based technique for custom on-chip interconnection network synthesis," in *VLSI Design, 2005. 18th International Conference on*, jan. 2005, pp. 623 – 628.

[14] I. Walter, E. Kantor, I. Cidon, and S. Kutten, "Capacity optimized noc for multi-mode soc," in *Design Automation Conference (DAC), 2011 48th ACM/EDAC/IEEE*, june 2011, pp. 942 –947.

[15] S. Murali, P. Meloni, F. Angiolini, D. Atienza, S. Carta,

L. Benini, G. De Micheli, and L. Raffo, "Designing application-specific networks on chips with floorplan information," in *Computer-Aided Design, 2006. ICCAD '06. IEEE/ACM International Conference on*, nov. 2006, pp. 355 –362.

[16] "Challenges in verification of clock domain crossings," http://webadmin.dac.com/knowledgecenter/2010/documents/VIMJAM-CDCVERIF-ABK-FINAL2.pdf/.

[17] "Closing the loop on clock domain functional implementation problems," http://w2.cadence.com/whitepapers/cdc_wp.pdf/.

[18] C. E. Cummings, "Clock domain crossing (cdc) design and verification techniques using systemverilog," in *SNUG*, 2008.

[19] E. Dahlhaus, D. S. Johnson, C. H. Papadimitriou, P. D. Seymour, and M. Yannakakis, "The complexity of multiterminal cuts," *SIAM Journal on Computing*, vol. 23, pp. 864–894, 1994.

[20] M. Berkelaar, K. Eikland, and P. Notebaert, "lp_solve 5.5, open source (mixed-integer) linear programming system," Software, May 1 2004, available at ¡http://lpsolve.sourceforge.net/5.5/¿. Last accessed Dec, 18 2009. [Online]. Available: http://lpsolve.sourceforge.net/5.5/

[21] ——, "Absolute values in lp_solve," Software, May 1 2004, available at ¡http://lpsolve.sourceforge.net/5.5/absolute.htm¿. Last accessed Dec, 18 2009. [Online]. Available: http://lpsolve.sourceforge.net/5.5/

[22] G. Karypis and V. Kumar, "Multilevel k-way hypergraph partitioning," in *Proceedings of the 36th annual ACM/IEEE Design Automation Conference*, ser. DAC '99. New York, NY, USA: ACM, 1999, pp. 343–348. [Online]. Available: http://doi.acm.org/10.1145/309847.309954

[23] P. Gillard, C. Li *et al.*, "Network-on-chip (noc) topologies and performance: A review," 2011.