Automatic Topology Optimization for FPGA Interconnect Synthesis

Alex Rodionov and Jonathan Rose
The Edward S. Rogers Sr. Department of Electrical and Computer Engineering
University of Toronto
Toronto, Ontario, Canada
Email: {arod, Jonathan.Rose}@ece.utoronto.ca

***Abstract*—The goal of FPGA interconnect synthesis is to generate a physical network that connects user-supplied functional modules according to a logical specification of the desired connectivity. In this paper, we augment an existing FPGA interconnect synthesis flow with the ability to automatically design the *topology* of the generated network while reducing its area subject to user-supplied performance specifications. The key specification is a per-transmission *importance* value representing the designer's willingness to have a transmission contend with other transmissions. The designer may also optionally specify that certain transmissions will never temporally overlap. We present an iterative algorithm that generates a topology which respects these specifications, with the goal of reducing area. Optimization decisions are guided by pre-characterized area models of interconnect primitives and an analytical worst-case traffic contention model. We apply our approach to a case study of an FPGA-based linear algebra application, where we successfully optimize the topologies of two of its sub-networks resulting in area savings of 60% and 75% with no overall performance degredation.**

I. Introduction

Hardware design is difficult, in part, due to the multitude of architectural choices that are available. In designs with many communicating elements, one key choice is the topology of the communication network: given a set of desired transmissions, there often exist many possible physical implementations, each differing in the number and arrangement of routing primitives (soft logic modules used for switching) and communication links (wires) used to carry the transmissions.

The choice of topology affects not just area and clock frequency, but also cycle count: the more physical resources that transmissions share, the lower the worst-case throughput. This is dependent on an application's communication patterns, and knowledge of those patterns can be exploited to reduce the cost of the network.

Figure 1 illustrates this notion with an example in which two source modules, X and Y, communicate with four sinks A, B, C, and D, through routing primitives which are indicated by circles. On the left is the topology which provides the best *worst-case* throughput, as it has minimal sharing of routing primitives. However, if it is known that some communications are infrequent, such as the transmissions from X to C and D, as well as from Y to A and B (which are drawn as dashed lines in the figure), then these transmissions may tolerate the sharing of physical connections (as shown on the right) with little performance loss.

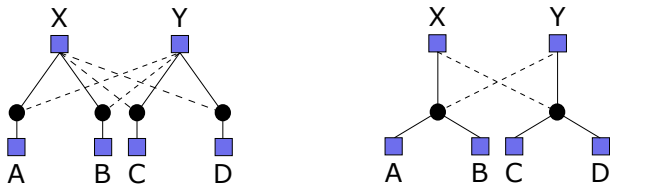

Fig. 1. Left: topology with maximal throughput. Right: optimized topology with greatly reduced area and slightly reduced throughput

In this paper, we augment GENIE [1], a system and interconnect building tool, with the ability to automatically generate application-specific topologies during its interconnect synthesis flow. The tool accepts as input a set of logical connections between user-created functional modules and generates a complete system containing the modules and soft interconnect that realizes the logical connectivity.

Connections are annotated with a new additional dimensionless parameter called *importance* that is a specification of desired relative throughput, independent of achieved clock frequency, which is otherwise difficult to accurately control when designing for FPGAs. A greedy optimization loop iteratively simplifies a unidirectional crossbar topology to minimize area while respecting transmission importance values. The designer may also optionally provide guarantees about transmissions' temporal exclusivity with one another, and this is also taken into account during optimization.

This paper is organized as follows: the next section discusses related work on automatic topology synthesis. Section III describes the topology optimization flow and its area and traffic models. This is followed by Section IV in which we evaluate the full new flow on an example application. Finally, we conclude in Section V.

II. Background

There is a body of existing work on application-specific topology synthesis for both FPGA and non-FPGA targets. Kapre et. al have studied the performance/area tradeoffs of Butterfly Fat Tree (BFT) topologies on FPGAs [2], [3]. They build BFT networks using two types of switches (t and π), each consisting of Split and Merge primitives [4] that are also

used in our work. By changing the ratio of t to π switches (controlled by a dimensionless Rent parameter), the bisection bandwidth of the BFT topology can be adjusted to match a particular application's needs.

The ShrinkWrap Compiler [5] is a tool that generates optimized interconnects for applications created using the CoRAM framework [6], which automates the creation of memory hierarchies. ShrinkWrap creates the interconnect between the auto-generated memory components and the user's application modules, optimized according to the application's communication patterns, which are explicitly specified by writing C-based control threads. Similar to our work, the ShrinkWrap interconnect uses segregated unidirectional tree-based networks, but the overall flow is specifically tailored to the generation of memory hierarchies.

In the more general non-FPGA context, there exist topology synthesis approaches that are based on iterative refinement starting from a trivial over-provisioned design point. The work by Murali *et al.* [7] emits a partial crossbar design that minimizes the latency caused by temporal overlap of transmissions sharing the network. This is achieved by beginning with a full crossbar and determining which nodes can share network resources while still satisfying performance requirements. Temporal overlap is measured through simulation, and the space of candidate crossbar configurations is tested via binary search.

A similar topology synthesis approach by Cong *et al.* [8] generates networks optimized for power and area containing arbitrary-radix routers. That approach begins with a topology that exactly matches the logical connectivity, and iteratively refines it such that network cost is reduced at the expense of resource sharing and performance degradation. Groups of transmissions can be mapped to a lower-cost shared bus when the user explicitly marks them as mutually temporally exclusive.

Ho *et al.* [9] proposed a technique to generate application-specific network topologies that have zero contention between transmissions, guided by knowledge of which transmissions overlap in time. They begin with a single network switch handling all communications, and then progressively create new switches and partition the traffic over them, using simulated annealing to re-route transmissions and approximate graph coloring to size inter-switch links.

Our work is specifically targeted for FPGA use, which has unique challenges compared to ASIC or generalized network contexts. For instance, the area and clock frequency of the complete interconnect are more difficult to estimate from independent characterizations of the individual network building blocks. Hence, we introduce a novel way for the user to specify their application performance requirements via a per-transmission dimensionless parameter that is not coupled to clock frequency. The topology synthesis process outlined in this paper is used within the context of an existing, full system generation tool, which inserts additional hardware beyond just network switching blocks that affects network area and performance.

III. Topology Optimization Flow

The automatic topology generation algorithm is implemented as an extension to the GENIE system-building tool [1], whose inputs are a set of functional modules and the desired transmissions between them. A transmission is a unidirectional flow of data, from a source to one or more sinks, that occurs one or more times during the lifetime of the application. The output of GENIE is a full system containing functional modules and interconnect composed of many types of primitive modules, notably including Split and Merge nodes [4] that direct the flow of traffic. Split nodes distribute their input to one of many possible outputs, based on an address. Merge nodes arbitrate among many incoming transmissions in a round-robin fashion. The arrangement of Split and Merge nodes defines the interconnect's topology.

Previously, GENIE was capable of generating one of a few pre-defined topologies (defaulting to a partial crossbar) in an automated fashion, or an arbitrary topology using explicitly instantiated Split and Merge nodes. Our work augments the existing automatic partial crossbar generation step with an iterative optimization loop that is guided by new specifications relating to the application's communication needs. For each transmission t, we now require an *importance* value $I(t)$ and a packet length $L(t)$.

A transmission's importance is a dimensionless value between 0 and 1 indicating how much the user desires that the transmission avoids contention with other transmissions. A value of 1 indicates that this transmission should encounter as little contention as possible, and a value of 0 means that the user does not care how much contention it encounters. Since contention results in throughput loss, importance becomes a way of specifying throughput requirements.

The second specification is the transmission's packet length in clock cycles, as measured from the sender's clock domain. Longer transmissions will cause longer delays, and more contention, at Merge nodes. The third (optional) specification is of mutually-exclusive groups of transmissions, indicating that the application will never initiate any two transmissions in the same group simultaneously. The GENIE tool already provided [10] the ability to specify mutual exclusion (which guides the tool to instantiate simpler Merge nodes when possible), but we now make additional use of it to guide topology optimizations. When left unspecified, a transmission is assumed to have an importance of 1, with a packet size of one cycle, and not be a member of any mutual exclusion group.

A. Flow Outer Loop

The topology generation flow first clusters the user's transmissions into connected components called *domains*, which are operated on independently. Each domain is initially mapped to a sparse crossbar implemented using Split and Merge nodes, an example of which is shown in Figure 2. A sparse crossbar is a good starting point, as it offers the least possible contention and greatest possible throughput, should that be the user's desire. Then, if possible, this initial topology is iteratively simplified via a series of refinement steps. Each refinement

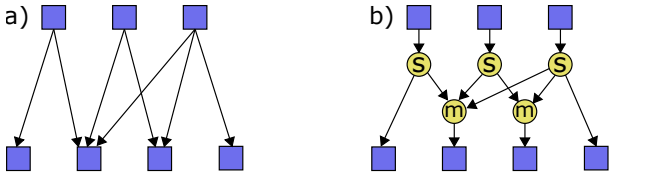

Fig. 2. **a)** A set of logical connections between functional modules forming a domain. **b)** The logical connectivity realized as a sparse crossbar topology using Split nodes, Merge nodes, and physical connections.

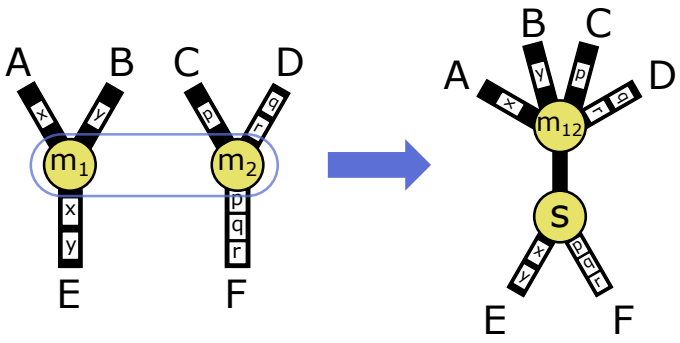

Fig. 3. A topology refinement step, which combines two Merge nodes and adds a Split node. The transmissions (x,y,p,q,r) routed over physical links (A-F) now face greater contention.

step combines two existing Merge nodes, replacing them with a single Merge node followed by a Split node, re-routing the affected transmissions. This is illustrated in Figure 3. By eliminating a Merge node, area consumption may be reduced, but it may also increase contention between transmissions. This is acceptable as long as the user's importance specifications are not violated by the refinement. Additionally, the area effect of each refinement is estimated by GENIE, and must be beneficial to be acceptable. After the initial crossbar topology, the flow thus proceeds as follows:

1) Every remaining pair of Merge nodes becomes a refinement candidate, as long as the proposed refinement yields a topology that satisfies importance requirements and an estimated area reduction.
2) If any refinement candidates exist, the one that yields the greatest estimated area reduction is chosen.
3) Steps 1 and 2 are repeated until no acceptable refinements can be found.

The proposed flow is a greedy traversal of a decision tree. Choosing the first refinement candidate requires the most work, as all pairs of Merge nodes must be examined, yielding an overall quadratic computation complexity. The estimation of the area effects of each candidate refinement is nontrivial. GENIE must take the proposed new topology and elaborate it further, adding additional interconnect primitives and performing automatic register insertion (which is beyond the scope of this paper). This is why we choose an iterative topology refinement flow, rather than, for example, an ILP-based one. The downside of the greedy approach is that it may not find the optimal topology.

B. Contention Model

In this section, we define and quantify transmission contention, which occurs between transmissions at Merge nodes. GENIE's Merge nodes use round-robin arbitration [10], so the forward progress of any stalled transmissions is guaranteed. In the worst case, a transmission will be forced to wait for all other contending transmissions to finish passing through the Merge node, being delayed for a number of clock cycles equal to the sums of the contending transmissions' packet lengths. More formally, consider two transmissions t_x and t_y. They *contend* if all the following are true:

- There exists a Merge node M with two physical interconnect links P_x and P_y as inputs, such that P_x carries transmission t_x and P_y carries t_y.
- Any physical link P_z that carries *both* t_x and t_y is downstream of M.
- t_x and t_y are not explicitly marked as mutually-exclusive by the user.

The second condition is equivalent to saying that if t_x and t_y have already been serialized before reaching M, they can not contend at M. Next, we define the *incremental contention* of transmission t_x *due to* transmission t_y as:

$$R(t_x, t_y) = \begin{cases} L(t_y) & \text{if } t_x, t_y \text{ contend} \\ 0 & \text{otherwise} \end{cases} \quad (1)$$

where $L(t_y)$ is the user-specified packet length of t_y.

Finally, the total worst-case contention experienced by a transmission t_x is obtained by summing its incremental contention due to every other transmission in the system:

$$C(t_x) = \sum_{t_y}^{t_y \neq t_x} R(t_x, t_y) \quad (2)$$

C. Refinement Step Criteria

A refinement step is only deemed acceptable if it respects the user's performance requirements as specified by the importance, packet lengths, and explicit mutual exclusivity of transmissions. Recall that the importance of a transmission t, $I(t)$, is a value between 0 and 1 with 1 reflecting the user's desire for maximum performance for t. Since a sparse crossbar ensures minimal contention, it is used as the baseline for defining the throughput experienced by a transmission when its importance is set to 1.

Let $C_0(t)$ be the total contention experienced by transmission t when sent through a sparse crossbar topology. A fresh topology x produced by a refinement step will potentially have a different contention, $C_x(t)$, for t. That topology x is deemed valid if the following is true:

$$\frac{C_0(t)}{C_x(t)} \geq I(t) \quad \forall\, t \quad (3)$$

The condition in Equation 3 requires that every transmission have a total contention in the new topology that is no worse than in the sparse crossbar topology, up to a factor equal

to the transmission's importance. There is elegance to this construction: in one extreme, if all transmission importances are set to 1, then only topologies that are as good as the crossbar will be acceptable. At the other extreme, with importances set to 0, all possible topologies are acceptable, up to and including ones consisting of a single Merge node. Levels of importance moving down from 1 will give rise to more contention in exchange for lower cost. In Section IV-B we illustrate this trade off enabled by various values of importance.

IV. Results

In this section, we evaluate the new topology optimization flow on an example application. The goal is to show that we can automatically exploit communication patterns to minimize interconnect area usage, while simultaneously providing the user an easy way to trade off performance and area by changing the importance of specific transmissions.

The application we use is a linear algebra solver that performs LU factorization on large matrices. It contains 16 compute elements (CEs) that process the input matrix in parallel, and four off-chip memory controllers that store the matrix data being operated on. Communication between the CEs and memories consists of three types of transmissions: read requests, write requests, and read replies. Communication and processing occur in two phases. In Phase 1, each CE i sends read and write requests to, and receive read replies from, memory controller $i \ \% \ 4$. In Phase 2, all CEs send a read request to an Aggregation Node, which after receiving all 16 requests, forwards one copy of it to one of the four memory controllers. This controller then *broadcasts* a read reply to all 16 CEs.

A. Experimental Description and Methodology

We perform two experiments to test our flow's ability to exploit the application's communication patterns. In the first experiment, we utilize the observation that traffic volume is dominated by read replies and write requests, whose data payload is 512 times larger than that of read requests. Therefore, we hypothesize that the interconnect that delivers read requests can be simplified without significant loss of application performance. We will attempt to do this in an automated manner with the new ability to specify an importance for the read request transmissions, which we specify and vary from 1 down to 0, with 1 being equivalent to the default behaviour of GENIE prior to this work.

In the second experiment, we focus on the read reply traffic, and observe that the Phase 2 read replies (which are broadcasted to all CEs) never temporally overlap with each other, nor with the Phase 1 read replies (which are only sent to one of four CEs from each controller). We compare the interconnect that results when this information is specified via 'mutual exclusivity' constraints, versus when it is not, which is the default GENIE behaviour.

For both experiments, GENIE is used to build the system and interconnect, with a total runtime of approximately 2 seconds on an Intel Xeon E5-2643. The generated SystemVerilog output is simulated to obtain application runtime (in cycles), and synthesized with Quartus Prime Pro 17.0 for an Arria 10 FPGA to obtain area utilization and clock frequency.

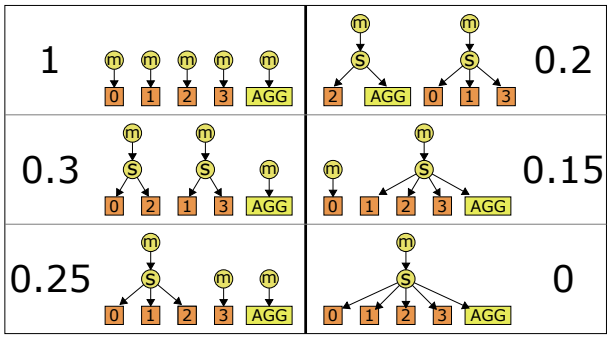

Fig. 4. Read request network topology vs. specified importance.

TABLE I
Results for Read Request Network

Importance		1	0.3	0.25	0.2	0.15	0
Fmax (MHz)		279	269	271	234	278	275
Area (Est.)	LUT	929	793	788	739	583	452
	REG	2111	2077	2052	1200	737	650
	TOTAL	3040	2870	2840	1939	1320	1102
Area (Act.)	LUT	683	595	566	622	472	402
	REG	1567	1551	1531	896	580	506
	TOTAL	2250	2146	2097	1518	1052	908
Latency (cycles)		7	11	15	15	15	19

B. Experiment 1: Read Request Network

Six different values of importance between 1 and 0 yielded measurable change in the topology of the read request network, shown in Figure 4. Here the 16 CEs are omitted for clarity, leaving only the memory controllers, aggregator node, and the GENIE Split and Merge nodes from the generated interconnect. At an importance of 1, the network begins as a full crossbar with five Merge nodes. It is gradually simplified as importance decreases, until only a single Merge node remains at importance 0. The resulting measurements of clock frequency, area, and latency in clock cycles are given by Table I. Estimated area (sum of LUTs and regsiters) is used internally by GENIE to guide topology optimization, and is displayed alongside the actual post-fitting results. An importance of 0 yields interconnect that is 60% smaller than using an importance of 1, which is what GENIE would have generated automatically prior to this work. The price of this area reduction is a 2.7× increase in worst-case latency. Despite this specific increase, the total simulated application runtime remained constant at 5.46 million clock cycles, deviating by no more than ± 0.02% across the six different values of importance. This confirms the initial intuition that the read request network is not the critical communication path in the main loop of the application.

C. Experiment 2: Read Response Network

By providing information about the temporal exclusivity of read reply transmissions, GENIE was able to generate simpler interconnect than a full crossbar. Both scenarios are depicted in Figure 5. The simplified topology has a single Merge node

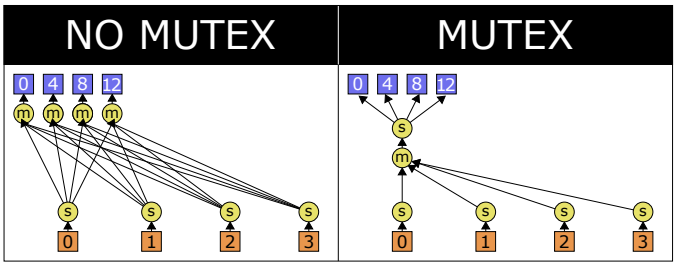

Fig. 5. Auto-generated read response network topologies, showing only 4 out of 16 CEs for clarity. **Left:** no mutual exclusivity information (default GENIE result). **Right:** mutual exclusivity specified.

TABLE II
RESULTS FOR READ RESPONSE NETWORK

		NO MUTEX	MUTEX	Change
Fmax (MHz)		273	279	+2.2%
Area (Est.)	**LUT**	4988	1340	-73%
	REG	38524	9212	-76%
	TOTAL	43512	10552	-76%
Area (Act.)	**LUT**	4694	1240	-74%
	REG	37773	8835	-77%
	TOTAL	42467	10075	-76%

per cluster of 4 CEs, rather than 1 per CE. Read reply links are 256 bits wide, and so the resulting area reduction is significant – averaging 75% between LUT, Register, and total area score (LUT+REG). The full clock frequency and area measurements (estimated and actual) are provided in Table II. There was no measured impact on the total application cycle count (5.46 million), due to the temporal exclusivity of the Phase 1 and Phase 2 read reply transmissions.

D. Modeling Accuracy

In both experiments, we observed that our area model consistently overestimated both LUT and register usage compared to actual post-synthesis results. Detailed inspection revealed this to be the result of logic synthesis optimizations removing signals that were stuck at 0 or 1 across module boundaries. This effect would be difficult to model without re-implementing back-end logic synthesis itself, and highlights a challenge for high-level hardware generation tools that target FPGAs.

The contention model described in Section III-B is meant to estimate worst-case throughput degradation and guide topology optimization. The latency measurements in Table I are taken under worst-case load and reflect the throughput degradation, showing a monotonically-increasing latency as the desired importance decreases. The latency also includes the effect of any registers automatically inserted by GENIE after topology synthesis.

In summary, the area and contention estimation models correctly tracked the relative trends in actual interconnect area and throughput, which led the optimizer to intelligent topology choices.

V. CONCLUSION

We have augmented the interconnect and system building flow within the existing GENIE tool to automatically generate network topologies that reduce area usage while respecting application-specific communication requirements given by the user. These requirements take the form of a per-transmission importance value and mutual exclusivity groups. The first specifies the maximum allowable amount of worst-case throughput degradation due to resource sharing, and the second guarantees that two or more transmissions never contend.

The flow, and the new specifications that control it, enable an easy and rapid design space exploration of unidirectional tree topologies: starting from a correct and functional, but possibly over-provisioned crossbar, the designer can reduce the importance of transmissions that are known to not be critical to total application runtime. Before time-intensive simulation and FPGA back-end synthesis takes place, it is possible to see the estimated area usage yielded by importance or mutual exclusivity constraints.

Using the above methodology on a real application, we demonstrated area reductions of 60% and 75% on two of the application's sub-networks, with no appreciable performance penalty. Results also showed that varying importance directly correlated with area usage and throughput, demonstrating the efficacy of the importance specification as a knob for trading off between the two. Since importance is a real-valued per-transmission parameter from 0–1, not all values will necessarily yield unique topologies. A future possible enhancement would be the ability to automatically discover the ranges of importance values that would yield unique topologies and report them to the user for evaluation.

The latest release of GENIE, complete with source code and examples, can be found at `http://www.eecg.toronto.edu/~jayar/software/GENIE/`

REFERENCES

[1] A. Rodionov and J. Rose, "Automatic FPGA System and Interconnect Construction with Multicast and Customizable Topology," in *FPT*, 2015.
[2] N. Kapre, N. Mehta, R. Rubin, H. Barnor, M. J. Wilson, M. Wrighton, A. DeHon *et al.*, "Packet Switched vs. Time Multiplexed FPGA Overlay Networks," in *FCCM*. IEEE, 2006, pp. 205–216.
[3] N. Kapre, "Deflection-Routed Butterfly Fat Trees on FPGAs," in *FPL*. IEEE, 2017, pp. 1–8.
[4] Y. Huan and A. DeHon, "FPGA Optimized Packet-Switched NoC using Split and Merge Primitives," in *FPT*, 2012, pp. 47–52.
[5] E. S. Chung and M. K. Papamichael, "ShrinkWrap: Compiler-Enabled Optimization and Customization of Soft Memory Interconnects," in *FCCM*. IEEE, 2013, pp. 113–116.
[6] E. S. Chung, J. C. Hoe, and K. Mai, "CoRAM: An In-Fabric Memory Architecture for FPGA-based Computing," in *FPGA*, 2011, pp. 97–106.
[7] S. Murali and G. D. Micheli, "An Application-Specific Design Methodology for STbus Crossbar Generation," in *Design, Automation and Test in Europe*, 2005, pp. 1176–1181.
[8] J. Cong, Y. Huang, and B. Yuan, "ATree-Based Topology Synthesis for On-Chip Network," in *IEEE/ACM International Conference on Computer-Aided Design*, 2011, pp. 651–658.
[9] W. H. Ho and T. M. Pinkston, "A Methodology for Designing Efficient On-Chip Interconnects on Well-Behaved Communication Patterns," in *The Ninth International Symposium on High-Performance Computer Architecture*, 2003, pp. 377–388.
[10] A. Rodionov, D. Biancolin, and J. Rose, "Fine-Grained Interconnect Synthesis," in *FPGA*, 2015, pp. 46–55.