Optimizing Deflection-Routed Butterfly Fat Tree Networks on FPGAs

Nevo Magnezi
Electrical Engineering
University of Maryland, College Park
SUNFEST Fellow
Email: magnezin@umd.edu

***Abstract*—Deflection-routed Butterfly Fat Tree networks implemented on Field Programmable Gate Arrays (FPGAs) can be used as resource and time-efficient communication networks between processing elements at leaf nodes. This work demonstrates how the routing function of switching nodes implementing a localized-deflection scheme can be optimized to minimize packet deflections and ensure an even distribution of packet bandwidth under high traffic. Our network generator program constructs arbitrarily-sized networks with a configurable bandwidth determined by the Rent parameter 0<p<1. Generated networks were tested under a range of packet injection rates and patterns. We expect our optimizations and network generator program to add to the robustness of constructing and utilizing Butterfly Fat Tree networks. We hypothesize that highly configurable deflection-routed Butterfly Fat Trees with localized-deflection schemes are ideal candidates to be implemented with partially reconfigurable FPGA regions in rapid-prototyping applications.**

I. Introduction

Advancements in semiconductor technology has allowed an exponential growth in transistor density and thus computational power [1]. However, designing an application-specific integrated circuit (ASIC) is not economical for every use case. When prototyping, manufacturing only hundreds or thousands of units (versus millions), or in cases where in-field upgrades are useful, FPGAs are both more economical than ASICs while still being more capable than microprocessors at doing high-performance tasks. Unfortunately, FPGA application design is slow, often taking hours from design synthesis to device programming. Additionally, unlike in software development, FPGA designs are often not portable across different families of devices.

To address these constraints, We envision an adaptive runtime strategy where high-level synthesis designs (i.e. written in the C language) are initially mapped as processors to partial reconfiguration regions (Figure 1a). Instrumentation can identify bottlenecks and accelerate processing elements to more optimally use the resources at hand. Meanwhile, a lightweight introspective interconnect network is necessary to allow processing elements to communicate. Because our strategy is adaptive, it is possible that over time, it may be desirable for neighboring partially-reconfigured processing elements to be combined into a single larger accelerated region (Figure 1b). Our interconnect network must also be adaptive in order to accommodate changes in size and traffic generation of these dynamic processing elements. We have identified the deflection-routed butterfly fat tree (BFT) topology as the most promising network for this use case.

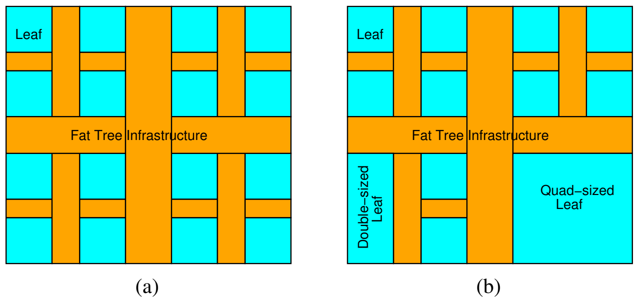


Fig. 1. Adaptation of network and processing elements over time.

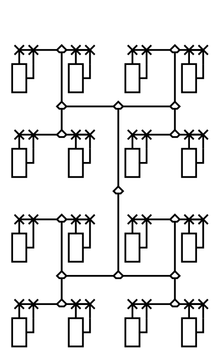
II. Background

A. Characterizing the Deflection-routed Butterfly Fat Tree

The deflection-routed butterfly fat tree (BFT) is a packet-switched network-on-chip (NoC) that allows different processing elements on the same chip, whether processors or otherwise, to intercommunicate. The BFT has two different types of nodes: leaf nodes, which are the processing elements that send and receive packets, and branch nodes, which are switches that route packets towards their destination. When two or more packets want to travel through the same switch port, the switch must arbitrate which packet is sent through the correct port, and which packet is deflected to a different port. In this paper, we explore only topologies with localized-deflection schemes, where all switches are capable of deflecting packets.

The BFT is composed of two different types of switches: t switches, which support two child and one parent connections, and π switches, which support two child and two parent connections. Switches are connected to either other switches, processing elements, or both. Switches and processing elements can be stacked hierarchically into levels, which identify how far away a node is from the root node(s). Any particular level of the BFT consists of one type of switch.

1) Binary Tree: The degenerate case of the BFT is the binary tree (Figure 2a). The binary tree consists solely of t switches, connected to processing elements at the bottom level. All processing elements are able to send and receive packets to all other processing elements through the network,


(a) Binary Tree ($p = 0$)

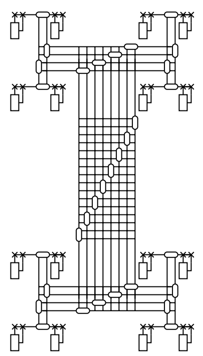

(b) Crossbar ($p = 1$)

Fig. 2. Two contrasting network topologies, with the same number of processing elements (rectangles). Triangles denote t switches and ovals denote π switches.

however the binary tree topology is only effective at providing nearest-neighbor style communication. Each t switch can only send one packet up the tree at a time, even if multiple packets should be routed upwards.

2) Crossbar: A contrasting BFT configuration is the crossbar (Figure 2b), where the component switches are π switches. π switches, which have an additional parent connection, are able to communicate with further nodes than t switches, rather than only nearest neighbors. Thus the crossbar has a lower degree of locality compared to the binary tree. A network containing more π switches than t switches is able to support a higher bandwidth through additional connections, at greater cost of resources [2].

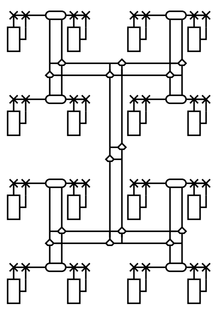
3) Bandwidth: The π switch's two parent connections guarantees that any particular packet has N different paths to reach a root switch of the crossbar, where N is the number of processing elements in the network. In contrast, the binary tree topology only has a single root switch that can be reached by any packet. Under the crossbar topology, any processing element can be reached through an independent non-conflicting path by any of the N root switches. In other words, every clock cycle, a non-conflicting permutation (one-to-one function) of N messages can be delivered to their destinations. In the binary tree, only one packet per clock cycle is guaranteed to be delivered with non-conflicting address permutations, as under worst case scenarios, all packets sent must be routed through the single root switch.

4) Resources: A binary tree contains $N-1$ switches, while a crossbar contains $\frac{N}{2}\log_2 N$ switches. A t switch has been empirically found to cost 141 look-up tables (LUTs) + 113 flip-flops (FFs), while a π switch costs 218 LUTS + 150 FFs [2]. The resource costs at $N = 1024$ processing elements is thus 144,000 LUTs and 115,000 FFs for a binary tree, and 1,116,000 LUTs and 768,000 FFs for a crossbar. There is a factor of 7.73 more LUTs and 6.64 more FFs for a crossbar than a binary tree, for the same number of processing elements. While these factors change for different N, the $\mathcal{O}(N)$ versus $\mathcal{O}(N \log N)$ growth of the two topologies demonstrates that crossbars will always cost more than binary trees for large N. Under many applications, a crossbar is too costly, and a binary tree is too slow.

5) Locality: Locality can be characterized via Rent's Rule [3]:

$$IO = cN^p \tag{1}$$

where IO is the number of exterior connections of the network and N is the number of nodes in the network. c is a tuning parameter that often corresponds to the average number of connections of a node, and the Rent parameter p is the measure of locality, where $p = 0$ corresponds to a binary tree and a high degree of locality, while $p = 1$ corresponds to a crossbar and a low degree of locality. We explore the construction and optimization of networks that have $0 < p < 1$, implemented through alternating levels containing t switches and π switches according to the given Rent parameter p.


(a) Mixed Switch ($p = 0.25$)

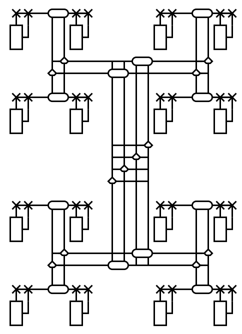

(b) Mixed Switch ($p = 0.50$)

Fig. 3. $p = 0.25$ corresponds to a $\pi - t - t - t$ network, while $p = 0.50$ corresponds to a $\pi - t - \pi - t$ network.

III. Construction

A. Binary Tree

A recursive program can construct a binary tree network by creating successively smaller sub-trees. Each sub-tree need only keep track of the number of leaves in the network and the address of the sub-tree's parent switch. The level of the sub-tree is calculated as the size of the sub-tree switch address, where the root switch of the entire network is level 0. The root switch does not require an address, because the switch only routes packets to the left child or right child according to the packet's first address bit. If an m level binary tree is to be constructed ($m = \log N$), first the $t_0 - t_1 \dots t_{m-1}$ sub-tree module is instantiated, containing the level 0 switch, connected to left and right $t_1 - t_2 \dots t_{m-1}$ sub-trees. The contents of the left and right sub-trees are recursively instantiated as level 1 switches, each with a 1 bit long address (0 or 1), and respective left and right sub-trees to which the level 1 switch is connected. These left and right sub-trees recursively follow the same process until level $m-1$ is reached. The level $m-1$ nodes are the processing element leaf nodes, which contain an interface that deflects misrouted packets back to the network and permits correctly sent packets to travel to the processing element.

B. Mixed Switch Networks

1) Calculating Switch Patterns: To construct a mixed switch network, we first must know where the switches are arranged in the network. To maximize bandwidth under general use conditions, we construct the switch pattern such that every sub-tree has as close of a Rent parameter p to the entire network. For example, given $p = 0.50$, we could construct a $\pi_0 - t_1 - \pi_2 - t_3 \dots \pi_{m-2} - t_{m-1}$ network or a $\pi_0 - \pi_1 \dots \pi_{m/2-1} - t_{m/2} - t_{m/2+1} \dots t_{m-1}$ network. Both switch patterns have the same Rent parameter p. If implemented as a network, only the former preserves the Rent parameter across sub-trees, and thus throughout the entire network. The switch pattern can be constructed iteratively by exploiting the property that t switch levels contribute $p = 0$ to the network and π switch levels contribute $p = 1$. The total p value of the network is the average of the levels of the network. As we construct the switch pattern, we consider adding an additional t to the end of our pattern, corresponding to the top of the sub-tree implementing the current pattern. If the addition of the t switch level decreases the overall p-value of the network below the desired value, we instead add a π switch level. Our method guarantees that the implemented network will have the smallest p value that is still greater than or equal to the desired value, and that all sub-trees will have close to identical p values.

TABLE I
NUMBER OF SWITCHES PER CLUSTER FOR DIFFERENT SWITCH LAYERS IN m LEVEL, $p = 0.50$ NETWORK

Level	Switch Type	π levels in sub-tree	Switches/Cluster
$m-2$	π	0	1
$m-3$	t	1	2
$m-4$	π	1	2
$m-5$	t	2	4
$\vdots$	$\vdots$	$\vdots$	$\vdots$
1	π	$\lfloor 0.50 \times (m-2) \rfloor$	$2^{\lfloor 0.50 \times (m-2) \rfloor}$
0	t	$\lfloor 0.50 \times (m-1) \rfloor$	$2^{\lfloor 0.50 \times (m-1) \rfloor}$

2) Modifying Construction to Suit Mixed Switch Networks: To build a mixed switch network, we follow the same process of building a binary tree, however instead of instantiating root sub-tree switches, we instantiate root sub-tree switch clusters. A switch cluster is an array of switches that share an address, whereby the IO connections of each switch in the cluster are concatenated. A t switch cluster containing k switches will have k left IO connections to the left sub-tree, k right IO connections to the right sub-tree, and k up IO connections to the parent switch cluster. A π switch cluster will also contain k left and k right IO connections, however will contain $2k$ up IO connections as both of the π switches up IO connections will be concatenated together.

Implementing our switch pattern construction scheme, the number of switches per cluster at a particular level can be calculated as:

$$switches/cluster = 2^{\lfloor p \times (m-lvl-1) \rfloor} \tag{2}$$

Alternatively, the number of switches per cluster can be calculated by raising 2 to the power of the number of π switch levels in the cluster's sub-tree (Table I).

IV. ROUTING FUNCTION IMPROVEMENTS

We have identified several areas that we hypothesize will improve bandwidth. While increasing the p value of the network also increases bandwidth, we expect these changes to make our network more efficient at minimal resource cost.

A. t Switch Up-Link Child Node Preference

In the traditional BFT implementation [2], the logic within a t switch that determines which child has preference sending packets to the parent switch has been state independent. One child always has preference sending packets along the up-link, thus, in high traffic scenarios, one sub-tree of the switch may be entirely ignored, adding to the deviation in time necessary for the average packet to travel through the network and creating unnecessary back-pressure. By modifying the preference to be dependent on a random bit, packets can be more evenly distributed through the network.

B. π Switch Down-link Arbitration

When both parents of a π switch are sending a packet to a single child node, a conflict results and one of the packets must be deflected while the other permitted to travel to the child node. Under a scheme where there is heavy traffic and the down-link preference is state independent, a packet may be constantly cycled between two switches while being consistently deflected at the contention switch. One solution may be to toggle which parent has preference. There is however the concern that a consistently rejected packet, which returns to the contention switch every two clock cycles, will continue to be constantly be rejected as the preference may also be toggled every two clock cycles. Our hypothesis is that an independent random bit will serve as the best arbiter to determine which parent has preference.

C. π Switch Up-link Arbitration

In previous implementation [2], a toggle bit has determined which up-link to send a certain packet to, whenever one or two packets are sent upwards. In cases where a upward-bound packet from a single child appears every other clock cycle, and a toggle is initiated in the other clock cycles, the network can aggregate packets from a certain source in one part of the network rather than distributing the packets through the network. The aggregation of packets in parts of the network may reduce the efficiency of the network, as many packets from the same source may compete in traveling towards similar directions. We hypothesize that implementing a random bit instead of a toggle bit will increase bandwidth, at a small increase in resource cost.

V. Expected Results

While we hypothesize that all routing function improvements will be measurable, the resource cost of implementing the improvements for all switches may outweigh the benefits. In low traffic conditions, which the network ideally runs under, the improvements may not show nearly as much change in efficiency of the network compared to high traffic conditions, and the special case improvements of modifying the π switch arbitration may not show a change in efficiency in low traffic at all. We hypothesize that changing the t switch up-link child node preference will have the greatest impact regardless of conditions, followed by π switch up-link arbitration, and finally π switch down-link arbitration. These hypotheses are determined by analyzing how often the improvement is likely to affect the efficiency of the network in real-world traffic conditions.

To test our network, we expect to run it through a range of of sizes and p values, as well as a variety of traffic conditions such as random, bitrev, bit complement, and tornado. Additionally, we expect to test our network with an altered random traffic condition whereby the probability of sending packets to neighboring processing elements is higher than further processing elements.

VI. Conclusion

With our network generator program, we are able to construct arbitrarily-sized networks with finely tuned switch contents via Rent parameter p. We hypothesize that our routing function improvements will add minimal cost to implementation with a notable change of efficiency, particularly in high traffic conditions. We expect to test our program under a wide variety of traffic conditions, including an altered random pattern that reflects real world placement of processing elements that communicate more with with one another nearer to one another. We expect to add synthesizable measuring hardware to allow the network to introspectively measure and reconfigure as its processing elements change, such as substituting more resource intense processing elements into areas that were formally sub-trees containing several processing elements, or reconfiguring the entire interconnect network to support higher bandwidth if the latency of the network becomes burdensome.

Acknowledgment

The author would like to acknowledge the support of the National Science Foundation, NSF REU grant no. EEC1659190. Additionally, the author would like to acknowledge the support of Professor André DeHon who sponsored this project, as well as his students Hans Giesen, Nick Roessler, Rafi Rubin, DJ Park, and Jared Winograd.

References

[1] G. E. Moore, "Cramming more components onto integrated circuits," *Electronics Magazine*, p. 4, 1965.

[2] N. Kapre, "Deflection-routed butterfly fat trees on fpgas," in *International Conference on Field-Programmable Logic and Applications*, September 2017.

[3] E. F. Rent, "Memorandum to: File, subject: Microminiature packaging-logic block to pin ratio," vol. 2, no. 1, pp. 40–41, Winter 2010, reprint of original 1960 IBM memo.