# NoC-Based FPGA: Architecture and Routing

Roman Gindin, Israel Cidon, Idit Keidar
*Electrical Engineering Department*
*Technion - Israel Institute of Technology*
*Haifa 32000, Israel*
*{rgindin@tx,idish@ee,cidon@ee}.technion.ac.il*

## Abstract

*We present a novel network-on-chip-based architecture for future programmable chips (FPGAs). A key challenge for FPGA design is supporting numerous highly variable design instances with good performance and low cost. Our architecture minimizes the cost of supporting a wide range of design instances with given throughput requirements by balancing the amount of efficient hard-coded NoC infrastructure and the allocation of "soft" networking resources at configuration time. Although traffic patterns are design-specific, the physical link infrastructure is a performance bottleneck, and hence should be hard-coded. It is therefore important to employ routing schemes that allow for high flexibility to efficiently accommodate different traffic patterns during configuration. We examine the required capacity allocation for supporting a collection of typical traffic patterns on such chips under a number of routing schemes. We propose a new routing scheme, Weighted Ordered Toggle (WOT), and show that it allows high design flexibility with low infrastructure cost. Moreover, WOT utilizes simple, small-area, on-chip routers, and has low memory demands.*

## 1. Introduction

Networks-on-Chip (NoCs) e.g. [1], [4], [5], [8], [9], [12], [15],[17] are commonly considered as a scalable solution for on-chip communication. However, it is also understood that there is no "one size fits all" NoC architecture [5], as different silicon systems have very different requirements from their NoCs. For example, in a System-on-Chip (SoC), the network usage patterns are many times known a priory. Hence, the NoC can be synthesized with the "correct" link capacities for supporting the required usage [10]. In CMP designs [14], [17] the traffic patterns depend on the various executed programs and hence can vary dramatically within the same system implementation. Therefore, CMPs' NoC resources should support all perceivable software scenarios. In contrast to both of the above, in a Field Programmable Gate Array (FPGA), the communication patterns are not known at fabrication time but may become much better determined when the chip is configured for a specific functionality. This unique scenario implies that there a cost and performance advantage in splitting an FPGA's NoC construction among the two design stages.

In this paper, we introduce a NoC design process and architecture for FPGAs. A distinctive feature of FPGA systems is that they include a combination of *hard* and *soft* functionalities. The hard functionality is implemented in silicon; it typically includes special purpose modules like processors, multipliers, external network and memory interfaces, etc. The soft functionality is configured using programmable elements (gate arrays, flip-flops, etc.) and routing components. Modern FPGAs contain hundreds of thousands of programmable elements, in addition to special purpose modules [22]. As technology scales, the sheer number of logic units will render a flat FPGA chip design unmanageable. FPGA engineers are already experiencing unacceptable place-and-route times for large designs with tight timing constraints. To remedy this problem, modern FPGA CAD tools, like Xilinx's PlanAhead, are already supporting a certain degree of hierarchical design: they allow designers to implement independent modules on the chip, which are later connected. We thus envision a future FPGA that is organized hierarchically, whereby the chip is divided into high-level regions (some programmable and some hard), interconnected by a NoC.

When architecting an FPGA NoC, one has to decide which functionalities are implemented as hard cores and which are left as soft. There is a tradeoff between the flexibility offered by the soft part and the higher performance offered by hard part. Since inter-module communication is often a bottleneck, it is important to

design the NoC architecture for high performance. We therefore advocate laying out the network infrastructure, including metal wires and hard-coded routers in silicon. At the same time, in order to allow for maximum flexibility, the NoC infrastructure should be able to accommodate multiple routing schemes and a large variety of traffic patterns. To this end, we allow network interfaces to be soft. Simplistic routing schemes, like XY, can employ small interfaces, whereas more elaborate source-routing schemes ([1],[13]) may have the interfaces store large routing tables. Our novel architecture is detailed in Section 2.

The main challenge is exploiting network resources efficiently, i.e., supporting a large number of program designs while investing minimal resources (wires and logic). In this context, there is an inter-play between the link capacity requirements and the routing scheme used to route packets between modules. A routing scheme that balances the load over all links readily supports more designs using smaller link capacities than an unbalanced one.

Section 3 formally defines FPGA routing (on our suggested architecture) as an optimization problem. In order to study the inter-play between routing and capacity requirements, we define a new concept called *design envelope*, capturing the required capacity for a collection of traffic patterns. The more traffic patterns the envelope accommodates, the more flexibility is offered to the designer configuring the chip. We study the design envelopes required to accommodate a collection of patterns with each of the routing schemes.

In Section 4, we present efficient solutions to the FPGA routing problem. Note that traditional routing algorithms like XY lead to unbalanced capacity allocation [10], and are therefore not suitable for programmable chips. It is possible to improve the balance by splitting the flow, toggling between sending on XY and YX routes [18] [20]; we call this approach *toggle XY (TXY)*. However, TXY is not optimal when traffic requirements are not symmetric, (which is very common in HW architectures). We improve it by adding *weights* to flow division (based on the design pattern), and call the resulting algorithm *weighted toggle XY* (WTXY).

Unfortunately, both TXY and WTXY have a major disadvantage – they split a single flow among two routes. This can lead to out-of-order arrivals, requiring large re-order buffers, especially in congested networks. Moreover, re-ordering requires the addition of sequence numbers to packet headers. We therefore suggest the use of *ordered* algorithms, which do not split flows, and ensure in-order arrivals. We do this by selecting one route (XY or YX) to each source-destination flow. We present the *weighted ordered toggle* (WOT) algorithm, which assigns XY and YX routes to source-destination pairs in a way that reduces the maximum network capacity for a given traffic pattern. The WOT routes are calculated

when the chip is programmed, and are loaded into a vector holding a bit per destination in the CNI.

In typical SoC and FPGA designs, the communication load is not divided evenly among all modules. Rather, a handful of modules are *hotspot modules* (or in short, hotspots), which communicate with many other modules. A hotspot can be an interface to external communication or memory, a master module that communicates with a number of slaves, a dispatcher that forwards requests from multiple masters to multiple slaves [17], etc. Additional examples are given by [1], e.g., in one of their reference designs, an SDRAM module has 7 connections, and an SRAM module has 4, while other modules have 2 or 3 connections each. We therefore focus on traffic patterns involving one ore more hotspots, (including a mapping of the reference example above to our architecture), when evaluating our solutions in Section 5.

Our evaluation shows that WOT routing reduces the link capacity cost across a wide range of designs. In some cases, its most loaded link requires 40% less capacity than the maximum required with XY, and up to 15% less capacity than with TXY.

We further compare *unconstrained* placement of hotspots, where the designer can locate hotspots anywhere on the chip, with *constrained* placement. Our results (cf. Section 5) show that constrained placement can significantly reduce the cost. For example, unconstrained placement of two hotspots with WOT routing requires 20% more capacity (in the busiest link) than constrained placement that dictates that the distance between the hotspots is at least three hops. With three hotspots, unconstrained placement requires 35% more capacity than placing the hotspots at a distance of at least three hops.

Finally, in Section 6, we present implementations of the four routing schemes studied in this paper. Since all our schemes can be supported by regular router [4], [20] we implemented only the logic required for the routing decision leaving out  other router design issues like buffers and scheduling. We show that all of our studied routing schemes require very few programmable elements (at most tens) and hence occupy very little area on the chip.

In summary, the main contributions of this paper are -

1. A new NoC-based architecture for FPGA, which balances between the flexibility of soft logic and the better performance of dedicated logic.

2. A design methodology for such an architecture.

3. A balanced in-order routing algorithm for this architecture, which minimizes the cost of supporting a large set of designs within given performance requirements.

## 1.1    Related Work

Sethuraman et al. [19] propose a fully soft NoC router design using current-day FPGA technology, which does not include any burned-in (hard) NoC infrastructure. In

contrast, we propose an architecture for future FPGA platforms, which is partly embedded supporting in silicon and partly defined and tuned at configuration-time. Hard routers offer better performance, lower power and better silicon area usage than their soft counterparts. On the other hand, any excess NoC resource that is not utilized in a specific design, cannot be converted to another usage.

DyNoC [14] is an architecture for adaptive routing using reconfigurable hardware. There are several additional works that present adaptive and dynamic routing, like DyAd [11], and Odd-Even routing by Chiu [6]. Unlike our solution, these solutions do not perform offline optimizations at configuration time; instead, they dynamically changes routing policies on the fly. This approach is very effective when the traffic pattern is not known and the network should balance itself during the runtime. However, an FPGA's communication pattern is many times determined at configuration time; offline optimization can be very effective. The simpler router design, in turn, reduces hardware cost in terms of area and power consumption. We therefore focus on static routing schemes in this paper.

The TXY algorithm was independently developed by Seo et al. for interconnection networks under the name of O1TURN [18]. It has been shown to be worst-case near-optimal for *uniform* traffic, where all nodes send and receive at the same rate [20]. In this paper, we focus on typical hardware traffic patterns, where some modules transmit and receive more than others. For such patterns, our weighted algorithms outperform TXY. Moreover, TXY results in out-of-order packet reception and hence requires large re-order buffers. Towels et al. [21] use linear programming for solving general flow problems for uniform traffic, but do not cover the case of asymmetric non-uniform patterns as considered herein.

# 2. High Level Architecture

This section presents the proposed architecture for NoC-based FPGA. It first discusses our proposed hierarchical chip structure and the advantages of such an organization. We then discuss how the NoC paradigm is applied to FPGA. Afterwards, we discuss which parts of the system are hard-coded on the chip, and which are configurable (*soft*). Finally, we discuss design methodology for NoC-based FPGA.

**Hierarchical Organization**

We propose a hierarchical architecture for future FPGAs, consisting of two types of regions connected by a NoC: (1) Configurable Regions (CR), resembling today's FPGA, consisting of programmable logic, which is a collection of possibly thousands of lookup tables (LUTs) and an internal programmable routing matrix; and (2) Functional Regions (FR), performing a predefined task, e.g., general purpose processor, DSP, fast external interface, etc. FRs are implemented as hard IP cores in

order to improve performance and reduce power consumption.

There are two main reasons for such hierarchical and modular structure for large programmable chips. First, CAD place-and-route tools cannot cope effectively with a flat design including a large number of modules. Even today, commercial tools like Xilinx's PlanAhead, and Synplicity's Amplify support a hierarchical design: the chip designer divides the flat chip into regions, designating which modules reside in each region. Then, place-and-route is run per region. In the last stage, connections among regions are wired using remaining resources. Second, long wires incur a high cost in terms of delay and power. In modern FPGAs, each LUT can be wired to any other element on the chip. Utilizing the long wires between remote elements induces long delays. Our design eliminates such long wires, and utilizes direct wires only within the relatively small CRs.

**NoC**

The regions are interconnected using a NoC. There are several reasons to base the future FPGA on NoC. First, NoC is much more scalable than all other interconnect solutions, such as point-to-point wires, buses, etc. [4]. Another reason is spatial reuse, which allows for scalable power cost compared to the increased routing matrix that is used in traditional FPGA. Moreover, FPGAs are often used as prototypes for ASIC. If the ASIC is migrating to NoC, the FPGA architecture should support NoC as well.

In our architecture, the NoC is a uniform mesh and each region is connected to a router via a local router interface. Like other NoCs, we use wormhole routing. The router structure is very similar to ones in previous NoC designs (e.g., [8],[3],[15],[1]) and includes support for multiple QoS classes, similar to QNoC [4]. Consequently, we do not detail the design here.

While it is possible to also provide direct communication wires between adjacent CRs not via the NoC, we chose to avoid such communication in order to simplify the design and the inter-region communication methodology.

**Soft vs. Hard**

The NoC consists of several components, like routers, wires and a network interface. The routers and links are part of the NoC's hard-core infrastructure, so as to allow maximum performance in terms of area and power. As this hard-coded infrastructure is always laid out, it is designed to support a large class of applications.

Due to the fact that FPGA has to support a lot of unknown applications with different traffic patterns, we suggest a *configurable network interface* (CNI), which is a mix of soft and hard parts. Each region is connected to the network via at least one CNI. A CR may have several CNIs, which can be configured to work together (when the CR acts as a single task module) or apart (when a CR is divided into multiple modules).

The CNI performs the following functions: network physical interface, buffering, reordering, fragmentation/reassembly (from application specific blocks to packets

and from packets to wormhole flits), interface to the region modules and routing support. Most of these functions are not application-specific, and are therefore hard coded in order to achieve better power, area, and performance. However, there are several application specific functions, like adaptation layers, playback and routing. The routing logic needs to be flexible, as different applications have different communication needs. Therefore, the routing layer and the application-specific layers are configurable, i.e., implemented from programmable elements.

**Example Layout**
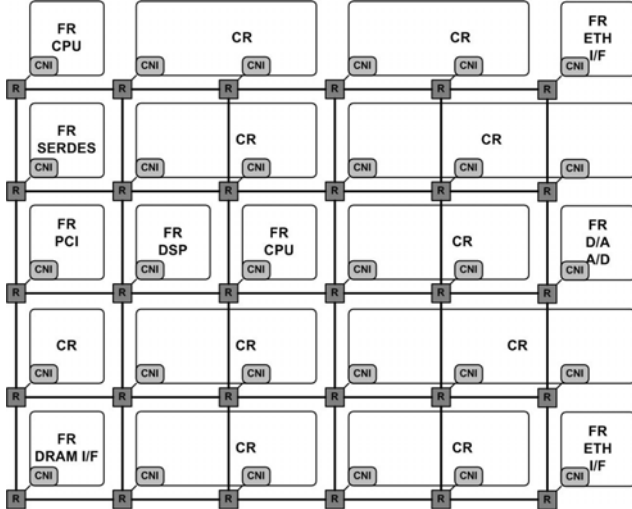


**Figure 1. Example of network on programmable chip.**

Figure 1 depicts an example chip design using our architecture. Some links (sets of parallel wires) between routers are routed across the CRs. This can be done since the network and CRs are designed by the same vendor at the same time, and therefore, it is possible to deploy long wire repeaters at the right places in the silicon of the CRs. This allows for a regular mesh topology, and avoids the complexity that can result from an irregular mesh [18].

**Design Methodology**
Given the hierarchical chip organization, the design methodology for programming such FPGA consists of the following phases: (i) division into high-level modules, roughly the size of CRs; (ii) placement of high-level modules; (iii) implementation of each module within a region; and (iv) inter-region routing.
In the placement phase, the designer performs the mapping of the high-level modules to the chip's functional and configurable regions. This process considers the gate count and functionality of the logical modules, and chooses CRs or FRs that can accommodate them. Clearly, modules with heavy communication requirements between them should be placed as close as possible, on the same CR when space allows.
Next, the routing of the inter-module communication is performed. The routing should comply with the constraints (capacity) of the pre-built network.

Placement was extensively studied in VLSI – e.g. [16], and known techniques are, by-and-large, applicable to our design. Therefore, in this paper, we focus on the routing phase of the design, which is unique to NoC-based FPGA.

## 3. FPGA Routing Problem Definition

A NoC-based FPGA is provided to the user with a pre-built NoC infrastructure. This network needs to be flexible enough to support many user applications that are unknown during the design of the network. A routing scheme for this network needs to address the challenge of providing high resource utilization with a low infrastructure cost. Thus, routing algorithms should be both efficient and flexible.
This section defines FPGA routing as an optimization problem. It assumes the placement phase is complete, and inter-region traffic requirements are known.
A **grid** is comprised a set of vertexes V and edges E. Every node on the grid has unique *(x,y)* coordinates. Each coordinate ranges between *0* and *n-1*.
A node represents a module attached to a router, which can be a region or part thereof.
A **traffic pattern** is a function $f : V^2 \rightarrow \Re^+$. This function defines the flows between every pair of nodes. A class of traffic patterns is a set **F**.
More accurately, these flows should be accommodated by the link capacities. They do not necessarily represent the peak traffic on the link, in case queuing delays are tolerated during peaks. For example, capacity can be allocated for accommodation 90% of the traffic demands, implying that say 99% of the traffic has a known delay bound.
A **path** from v1 to v2 is a connected sequence of links that starts at node $v_1$ and ends at node $v_2$.
A **routing algorithm A** defines the fractions of the flows that are sent over all possible routes from every source to every destination. For example, half of the flow is sent over one possible route and the other half over another route.
An **ordered routing** does not split flows, i.e. each pair is mapped to a single path.
The **link capacity for given A** and ***f*** is the sum of all the flows that are routed on this link.
The **maximum capacity for given A** and ***f*** is the capacity of the most loaded link in the network for a given traffic requirement pattern. On uniform meshes, this will dictate the capacity of all links. The maximum capacity reflects the **cost** of supporting the given traffic pattern.
A routing algorithm should minimize the maximum capacity, i.e., the cost for supporting a given ***f***.
The **design envelope** of a given algorithm for a class of traffic requirement patterns **F** is the minimal assignment of capacities to the links in the NoC graph, which supports every traffic pattern in **F**. Each link's capacity in

the envelope is its maximum capacity over all traffic patterns in the specified class with a given routing algorithm $A$. Uniform design envelopes are preferred.

Another metric that is used to evaluate the routing algorithms is the **gate count**. Gate count is the amount of logic required for the implementation of the routing decision circuit. Clearly, lower gate counts are preferred.

# 4. Routing Algorithms

We study and propose several routing algorithms and evaluate them in terms of maximum capacity and design envelope. This section presents the routing techniques and analyzes the efficiency of the algorithms.

## 4.1 Flow splitting algorithms

This section presents routing algorithms that improve the network's balance by splitting flows among several paths. Before presenting these algorithms, we recall the simple and well-known XY and YX routing algorithms, upon which they improve.

**XY routing** first routes packets horizontally, towards their X coordinate, and then vertically, towards their Y coordinate. XY is commonly used in NoCs thanks to its simplicity and inherent deadlock-freedom. However, it induces unbalanced link loads [10].

**YX routing** uses the same technique but reverses the order of the vertical and horizontal routing.

**Toggle XY Routing** (TXY) [20] improves XY's load-balancing by routing half the packets in the XY path and the other half in the YX path. Each packet header includes a bit indicating whether its route is XY or YX. Each CNI toggles between XY and YX packets. This scheme avoids deadlocks by using two virtual channels (VCs) per router: one for XY paths and one for YX with any fair scheduling scheme between VCs.

**Weighted Toggle XY Routing** (WTXY) improves TXY by sending different portions of flows on the XY and YX paths.

Splitting the traffic evenly between XY and YX routes, as in TXY, does not always achieve optimal load balancing. In some cases, sending a different portion of the traffic by each route may achieve better load-balancing and thus reduce capacity requirements. For example, Figure 2 shows how the maximum link capacity is affected by the fraction of traffic routed XY on a 5x5 grid with two hotspots at locations (1,1) and (2,1) on the grid, where each node send the same amount of data to every hotspot. Here, optimal capacity is achieved when roughly 2/3 of the traffic is routed XY, offering an almost 20% improvement over TXY.
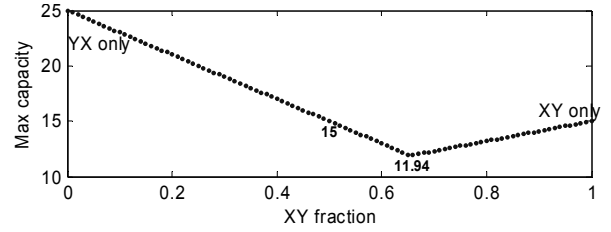


**Figure 2. The impact of the XY fraction on 5x5 grid with two hotspots at locations (2,1) and (1,1).**

**WTXY** therefore chooses the best XY fraction, $c_{xy}$, according to given traffic requirements. This fraction is calculated offline for each application before the chip programming phase, and is loaded to the CNI with the rest of the configuration data. The calculation of $c_{xy}$ is beyond the scope of this paper since our main focus is on ordered algorithms.

WTXY can be implemented (see Section 6) using a random number generator and a comparator in the network interface that assigns the route bit in the packet header according to $c_{xy}$. Deadlocks are avoided the same way as in TXY.

## 4.2 Ordered algorithms

The main problem of the algorithms described above is the flow splitting. When flows are sent over multiple routes, the packet arrival order may differ from the sending order. This requires the receiver CNI to maintain large reordering buffers. Consider, e.g. a relatively small window of $2n$ outstanding packets, $(n^2)$ nodes communicating with each hotspot and about 10% of the nodes are hotspots.

In this case total memory requirement is $(0.1 \cdot n^2) \cdot 2n^3$. If n is 7, we have to put enough memory for 16 K packets. The situation is even worse in case there are multiple classes of services or more source destination (S-D) pairs. Moreover, the flow splitting routing requires communication overhead to carry the numbering of the packets to allow the reordering at the destination.

We therefore propose *ordered* algorithms, in which all the packets between the same S-D pair are sent over the same route. Hence, packets arrive in the order that they are sent, and no re-order buffer is required.

A simple way to assign routes per flow is **Source Toggle XY - STXY**. This algorithm toggles the routes of entire flows. To create a better spreading of the flows, we use both source and destination addresses to assign the route. For example, a bitwise XOR of the source and destination addresses is used to determine the route. Using this technique gives us a close approximation to TXY with a small quantization error since half of the flows is sent on XY path and the other half on YX.

However, due to the previously shown advantages of weighted routing, we suggest the Weighted Ordered Toggle (WOT) algorithm. WOT divides the source-

destination routes in a way that produces best result in terms of maximum capacity.

The remaining challenge is to develop a technique for effectively choosing the route of each source-destination pair. First, we present analytical solution for a single hotspot and later show heuristic algorithm for the general cases.

**WOT Optimal Solution for Single Hotspot**

In case of a single hotspot, we have a single destination on the square grid and all other nodes on the grid send the same flow *f* to it. Consider a hotspot at coordinates *(x,y)*, as shown in Figure 3 (coordinates range from 0 to n-1).
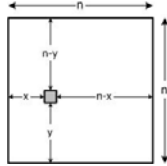


**Figure 3. Location of a single hotspot.**

For each node we define a binary indicator $r_{ij}$ that dictates whether the flow from node *i* to node *j* is routed XY or YX. We also divide the nodes to four groups - left-down (LD), left-up (LU), right-down (RD) and right-up (RU). For example, the LD group consists of the nodes with both coordinates smaller than $(x, y)$. In our case, index *i* refers to the hotspot node. In order to find the maximum we show two lemmas.

**Lemma 1:** In case of the single hotspot, and if all routes are either XY or YX, the most loaded link is one of the links adjacent to the hotspot.

**Proof:** Suppose by the way of contradiction that the most loaded link, $L_{\max}$, is not the incoming link of the hotspot.

Since the routing algorithm performs only one turn (XY/YX) we have two possible locations of $L_{\max}$.

(1) $L_{\max}$ shares one coordinate (X or Y) with the hotspot. In this case, the flow reaches the hotspot with the addition of the flow of the hotspot's closest neighbor.

(2) $L_{\max}$ does not share any coordinate with the hotspot

Without loss of generality, suppose that $L_{\max}$ located at RU. That means that in order to reach the destination within one turn, the flow is routed down/left or left/down (depending on whether $L_{\max}$ is horizontal or vertical) and contributes to link of the right of the hotspot.

In both cases the capacity of the adjacent link is larger or equal to $L_{\max}$ . ∎

Using $r_{ij}$ indicators and the hotspot location *(x,y)*, we can calculate the flows on the hotspot incoming links. For example, the flow on the left link is

$$F_{left} = \left( x + \sum_{j \in (LU,LD)} (r_{ij} == YX) \right) \cdot f$$

. This flow consists of the flows from *x* nodes located on the same coordinate, which are not split, and from the split flows from *x(n-1)* nodes residing to the left of the hotspot that send the data using

YX. We calculate similar formulas to all the incoming flows.

**Lemma 2:** The lowest maximum capacity achieved when the absolute value of the difference $D$ between maximum horizontal and the vertical flows on the adjacent links to the hotspot ($H_{\max} = \max(F_{left}, F_{rightt})$ and $V_{\max} = \max(F_{down}, F_{up})$) is minimal.

**Proof:** Suppose by the way of contradiction that it is possible to reduce the difference. This is done by re-routing any flow (all the flows are the same) from the most loaded link to any other link. If we reduce the difference, we reduce the most loaded link which is impossible. ∎

Our algorithm relies on Lemma 1 and 2. In order to find the optimum WOT route assignment we minimize the difference $D$ as described by the following equation -

$$\min(|D|) = \min(|H_{\max} - V_{\max}|) = \min(\max(F_{left}, F_{right}), \max(F_{left}, F_{right}))$$

while working with binary variables $r_{ij}$. The equation is solved with known techniques like linear programming.

**WOT General Heuristic**

In general case, when the analytical solution is more complex we suggest heuristic algorithm. We studied three techniques –

**Iterative Assignment** – in this technique we first calculate the $c_{xy}$ ratio. The grid is initialized to STXY. The algorithm serially scans the nodes and changes the routes in such a way that XY ratio will be as close as possible to the optimal ratio that was calculated before.

**Random Assignment** – in this technique we also calculate the optimal ratio and scan the (S-D) pairs. For each pair we assign XY routing with the probability equal to the optimal ratio and YX with the complement probability. In this way we'll have approximately the desired ratio.

**Min-Max Assignment** – seeks an optimal assignment without prior knowledge of the optimal ratio. The algorithm starts with STXY as its initial assignment. In each step, the algorithm first finds the most loaded link, *l,* and then goes over all the S-D pairs that contribute traffic to *l*. For each such S-D pair, the algorithm calculates the cost that would result from changing the route of this pair (and only this pair) from XY to YX or vice versa. Among these, the algorithm chooses the one that leads to the lowest maximum capacity on links affected by the route change. Subsequently, a new step starts with the new route assignment. The algorithm converges either after a predefined number of iterations or upon reaching a local minimum. In the latter case, WOT can achieve even lower cost than WTXY, since it is not limited to a single $c_{xy}$ ratio for the entire network.

The following graph shows the comparison between the three systems. We compare here by the relative difference between the achieved maximum capacity and the maximum capacity of the WTXY algorithm. In our

simulation we randomize the number of hotspots in range (1-3) and the also the location of the hotspots for grid size from 5 to 10. For each grid size we perform 1000 simulations. The y-axis is $C_{max}^{WOT} - C_{max}^{WTXY} \Big/ C_{max}^{WTXY}$.
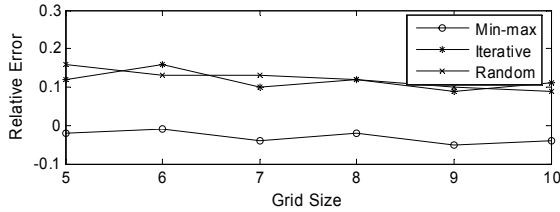


**Figure 4. Relative overhead for WOT route assignment schemes vs. WTXY.**

We can observe that the min-max scheme produces the best results and is very close and sometimes even better then WTXY in terms of maximum capacity.

In the rest of this paper, we use the min-max assignment for evaluating the WOT algorithm.

Finally, a well known technique for ordered routing is source-routing, where the source chooses the path to each destination and sends it in the packet header. We evaluated the cost of source routing in our architecture using efficient path assignments. The required capacities were very similar to those required by WOT, without taking into account the increase in flow induced by the large headers used in source-routing. Moreover, source routing requires more lookup-table hardware for storing the routes in each source. Therefore, we found source-routing to be inferior to WOT in our design.

## 5. Evaluation

We turn to evaluate the capacity requirements of the different routing schemes under typical on-chip traffic patterns using simulations. First four sections show the results for synthetic examples, and the last section shows the result of the real-world design mapped to our architecture.
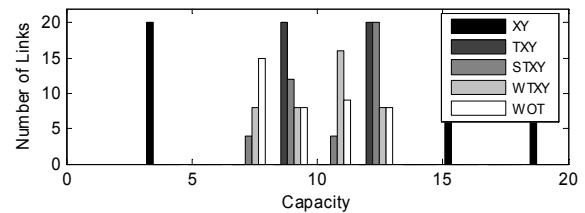
### 5.1 Single Hotspot

We first examine traffic patterns in which all nodes communicate with one hotspot with different routing schemes. We compare the results to a theoretical lower-bound, which is computed by dividing the hotspot's communication requirements by the number of links leading to it.
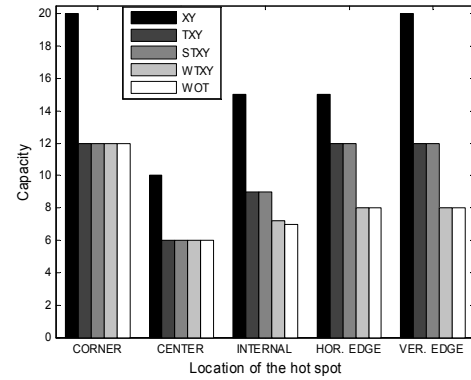


**Figure 5. Maximum capacity requirements for one hotspot on a 5x5 uniform grid at various locations with different routing schemes.**

We observe that XY requires the highest maximum capacity in the most loaded link. This is expected, since XY does not balance the load among links. By alternating between XY and YX routes, TXY and WTXY considerably reduce the capacity requirements, and are optimal for hotspots located in the center or at a corner of the grid. When the hotspot's X and Y coordinates are not symmetric, that is, when its distance from the edge on one axis is greater than the other, splitting the traffic evenly between the XY and YX routes is not optimal, and WTXY can improve on TXY by up to 33%. Observe that the WOT algorithm, despite of being in-order, produces a cost that is very close to that of WTXY.

We further observe that locating a hotspot in the center of the grid requires the smallest capacity, whereas a corner hotspot requires the highest capacity. This is because in the center, the load can be spread evenly among all four directions. This suggests that smaller link capacity can be allocated if the user is restricted in her placement of hotspots. Some hotspot modules provide external interfaces and must therefore reside at the edge of a chip. Nevertheless, it is reasonable to require locating them in the middle of an edge rather than in a corner, and consequently save 33% in capacity. Figure 6 shows the histogram of link capacities of the envelope for all possible hot spot location for all the routing schemes. It is evident that XY is highly unbalanced, with vertical links five times as loaded as horizontal ones. In contrast, WOT and WTXY can satisfy the design envelope with balanced grids.



**Figure 6. Histogram of link capacities in design envelopes of all possible single hotspot locations.**

### 5.2 Two hotspots

Next, we examine traffic patterns involving two hotspots. We observe that in this case, the cost is highly dependent on the distance between the hotspots. If the hotspots are close together, there is a high concentration of traffic in the area where they are located, whereas if they are far apart, they barely impact each other, and the capacity requirements are similar to those for a single hotspot. This trend is depicted in Figure 7. We conclude that by restricting the allowed distance between hotspots, one can save up to 25% capacity. We can see that the weighted algorithms present better results then regular and that the ordered algorithms show costs that are very similar to the un-ordered version (TXY vs. STXY and WTXY vs. WOT).
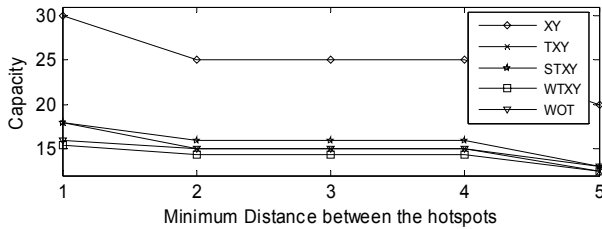


**Figure 7. Maximum worst case link capacity required for two hotspots versus the distance between them.**

## 5.3 Three hotspots

Here, we examine the case of three hotspots. The minimal distance between the closest pair among the three hotspots is dominant in determining the load, as is shown in Figure 8. Weighted algorithms present lower costs here and the ordered version are very close to the flow-splitting algorithms. In total, weighted algorithm outperforms the regular one about 20%. We can also see that for an unconstrained placement, WOT is even better.
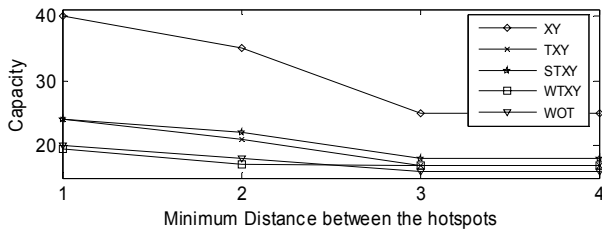


**Figure 8. Maximum link capacity required for three hotspots versus minimal distance between a pair of them.**

## 5.4 Complex traffic patterns

In this section, we compare the routing algorithm using more realistic data patterns. We use traffic model described in [3] . This random model has three parameters for each node – a probability to be a hotspot, a probability to send data to a hotspot node and a probability to send data to a non-hotspot.

We use this model for various values of the probability. For the symmetric graphs the weighted algorithms produce similar costs as the non-weighted ones, but as the

asymmetry of the traffic pattern increases the influence of the weight grows. One of the comparison graphs is shown below. The graph presents average maximum capacities for various grid sizes. Each grid size was simulated with 100 random patterns.
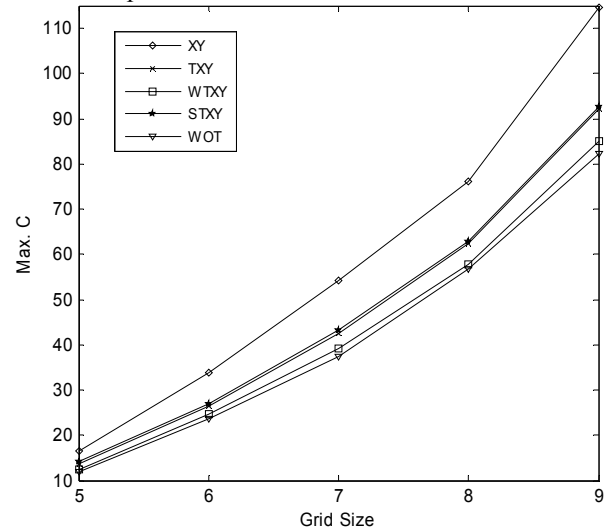


**Figure 9. Comparison of routing schemes for** $P_{hs} = 0.1$ **and** $P_{hs}^{send} = 0.8$ , $P_{no,hs}^{send} = 0.05$ **for various grid size.**

We can see that WOT incurs lower costs than all other routing algorithms despite the constraint of in-order routing. Further, STXY's maximum capacity is very close to that of TXY. The following graph shows the average improvement of the weighted algorithms for the same case as the previous figure.
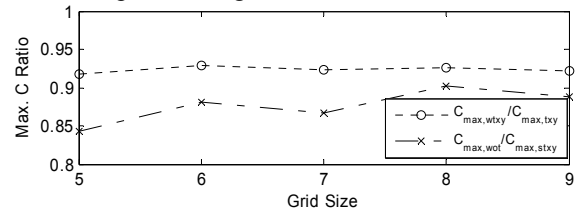


**Figure 10 – Average improvement of the weighted algorithms.**

## 5.5 A Real World Example

In this section we show the results of the routing algorithms applied to the MPEG 4 decoder design of Bertozzi et al. [1]. The logic diagram of the design is shown in Figure 11 .
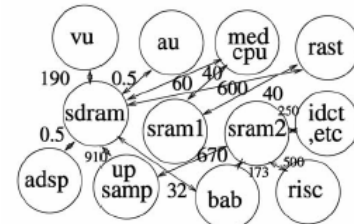


**Figure 11 – MPEG4 Decoder block diagram.**

We can see that the design includes hotspots: the SDRAM module has 7 connections; the SRAM module

has 4, while other modules have small connectivity to other modules.

As discussed in Section 2, the design process has two phases (1) Mapping of the logical graph to the NoC grid and the placement of the modules; (2) routing the inter-region flows on the NoC.

In previous examples, we worked with given mapped designs. In order to compare the various routing algorithms for the reference design, we need to first map and place its modules in regions in our architecture. We do so manually. The mapped graph is shown in Figure 12.
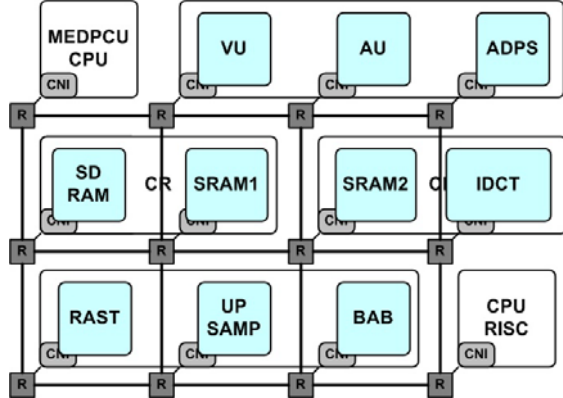


**Figure 12 – MPEG4 mapped design.**

The grid has 3 rows of regions. The first and the last rows contain one CPU each and one large CR with 3 CNIs. The middle row contains 2 CRs with 2 CNIs each. The reference design is relatively small and thus mapped to a small grid. Note that in future designs, modules are expected to be much more complex.

Figure 13 shows the histogram of the link capacities of this design for several routing schemes. We can see that WOT produces the least loaded maximum link (1053 compared to 1539) and generates a balanced capacity distribution, matching the results of our synthetic experiments above.
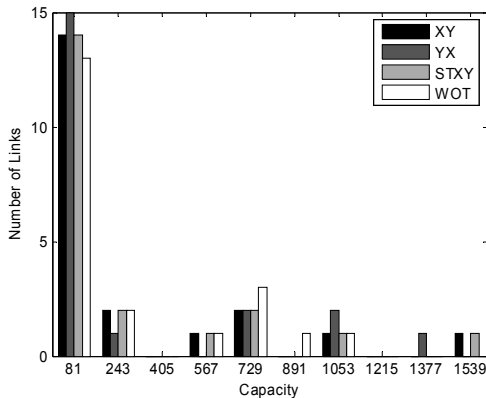


**Figure 13 – MPEG4 links capacity histogram.**

## 6. Hardware Costs

This section analyzes the incremental hardware costs for supporting each of the different routing schemes in the reference FPGA NoC design of Section 2. As noted above, a simple router with 2 virtual channels to avoid deadlock [18] supports all routing techniques presented in this paper. The router is a constant part of the hardware infrastructure laid by the vendor in the programmable chip. The data width of the router and the maximum frequency is dictated by the required capacity. The difference between the routing techniques is in the CNI. Recall that in our design, the CNI is located in the CR and is partially implemented using programmable logic to allow flexibility.

Figure 14 shows the schematic implementation of the circuits that produce the control bit of the packet header that determines the routing – XY or YX for each of the routing techniques.
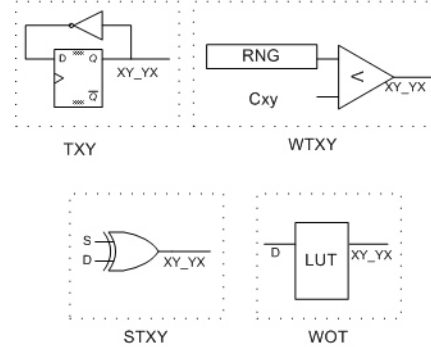


**Figure 14 – Routing circuit implementation.**

TXY is implemented with a simple flip-flop that inverts its state every sent packet.

WTXY is a random number generator (RNG) implemented using linear feedback shift register (LFSR) and a comparator that compares the random value to the predefined threshold (Cxy). If the random is larger than Cxy – the routing bit is XY, and otherwise it is YX.

The STXY implementation is a bitwise XOR of the source and destination IDs. The width of the XOR gate depends on the number of bits required for unique node ID representation. In our case, the ID width can be limited to the logarithm of the number of nodes in the grid - ($\log(n^2)$). We implement it with 4-way LUTs' tree configured to perform XOR.

WOT routing circuit implementation uses Look-Up-Tables (LUT) for logic implementation of the routing decision. 4-way LUT is one of the basic elements of the FPGA – c.f. [22] and is enough to implement up to 16 possible destinations. The 4-way LUT is easily expanded by cascading and multiplexing to the desired width.

We implemented the routing decision circuits and synthesized it using Synplify 8 synthesis tool. In our implementation we used 5-bit ID for source and destination identification. In Table 1 we show the actual and theoretical cost of each routing scheme.

**Table 1 – Routing circuit comparison for various schemes.**

| Routing Scheme | LUT count for n – worst case | LUT count for n = 5 | Notes |
|---|---|---|---|
| TXY | 1 | 1 | - |

| WTXY | 32 | 32 | We used 16-bits RNG and 16 bit comparison |
|---|---|---|---|
| STXY | $\left\lceil \dfrac{\log(2 \cdot \log(n^2))}{2} \right\rceil$ | 3 | Bitwise xor of $\log(n^2)$ words. Each LUT implements XOR of 4 bits |
| WOT | $\left\lceil n^2 / 16 \right\rceil$ | 2 | Each LUT acts as a 16 bit ROM. We need to cascade several LUTs to perform the lookup. |

This is the worst case. For sparse vectors the logic implementation can be more efficient.

We can see that WOT routing is very efficient in terms of hardware. Even though WTXY gate count does not depend on the grid size at all, for smaller grids (up to relatively large grids with n = 9) WOT performs better than WTXY.

All the routing schemes presented here introduce a very low overhead in terms of area and can be easily implemented on the chip.

## 7. CONCLUSIONS

We have presented new hybrid architecture for programmable chips based on NoC. We recognized that the main challenge of such programmable chips is designing flexible routing scheme to efficiently support variety of application on a pre-built network infrastructure. We studied routing schemes that can be used in this architecture, and their impact on the capacity requirements. We presented a simple yet efficient routing algorithm, WOT, which can be configured to balance link loads according to traffic patterns defined when the chip is configured. Since WOT is an ordered algorithm, it eliminates the need for large reordering buffers and reduces the cost of the network. The cost of the WOT implementation is low.

## 8. References

[1] D. Bertozzi, A. Jalabert, S. Murali, R. Tamhankar, S. Stergiou, L. Benini, G. De Micheli,"NoC Synthesis Flow for Customized Domain Specific Multiprocessor Systems-on-Chip", DATE 2005

[2] D. Bertozzi and L. Benini, "Xpipes: A network-on-chip architecture for gigascale systems-on-chip", Circuits and Systems Magazine, IEEE Volume 4, Issue 2, 2004.

[3] E. Bolotin, I. Cidon, R. Ginosar, A. Kolodny, "Efficient Routing in Irregular Mesh NoCs", CCIT Report #554, Elec. Eng. Dept, Technion, Sep. 2005.

[4] E. Bolotin, I. Cidon, R. Ginosar, A. Kolodny, "QNoC: QoS Architecture and Design Process for Networks on Chip", JSA, Feb 2004.

[5] Cidon and I. Keidar "Zooming in on Network-on-Chip Architectures", CCIT research report, December 2005.

[6] Chiu Ge-Ming. "The Odd-Even Turn Model for Adaptive Routing", IEEE Transactions on Parallel and Distributed Systems, Vol. 11. July 2000, pp. 729-738.

[7] W. Dally, B. Towles, "Route packets, not wires," DAC, Jun. 2001, pp. 684-689.

[8] K. Goossens, J. Dielissen, A. Radulescu. "AEthereal Network on Chip: Concepts, Architectures, and Implementations", IEEE Design and Test of Computers, September/October, 2005.

[9] P. Guerrier and A. Greiner: "A Generic Architecture for On-Chip Packet- Switched Interconnection". DATE, March 2000

[10] Z. Guz, I. Walter, E. Bolotin, I. Cidon, R. Ginosar, A. Kolodny. "Efficient Link Capacity and QoS Design for Network-on-Chip".In DATE 2006

[11] Jingcao Hu, Radu Marculescu, "DyAD - Smart Routing for Networks-on-Chip", DAC 2004

[12] Jantsch, J. Soininen, M. Forsell, L. Zheng, S. Kumar, M. Millberg, J. Öberg. "Networks on chip". In Workshop at ESSC Conference, September 2001

[13] N. Kavaldjiev, G. J. M. Smit, P. T. Wolkotte, P. G. Jansen, "Routing of guaranteed throughput traffic in a network-on-chip", Report Acquisitions Computer Hardware, November 2005

[14] M. Majer, C. Bobda, A. Ahmadinia, J. Teich, "Packet Routing in Dynamically Changing Networks on Chip". IPDPS 05 – Workshop 3, p 154b.

[15] F. Moraes, N. Calazans, A. Mello, L. Möller, and L. Ost, "Hermes: an Infrastructure for Low Area Overhead Packet-switching Networks on Chip", Integration, the VLSI Journal, Oct. 2004.

[16] H. Murata, K. Fujiyoshi,S. Nakatake, Y. Kajitani, "Rectangle-Packing-Based Module Placement", ICCAD-95, p 0472

[17] Radulescu, and K. Goossens, "Communication services for networks on chip, in Domain-Specific Processors: Systems, Architectures, Modeling, and Simulation", Marcel Dekker, pp. 193–213, 2004

[18] D. Seo, A. Ali, W.-T. Lim, N. Rafique, M. Thottethodi. "Near-optimal worst-case throughput routing for two-dimensional mesh networks". ISCA, 2005.

[19] B. Sethuraman, P. Bhattacharya, J. Khan, R. Vemuri. "LiPaR: A Light Weight Parallel Router for FPGA based Networks on Chip", GLS VLSI. April 2005.

[20] B. Towles and W. J. Dally. "Worst-case Traffic for Oblivious Routing Functions". Computer Architecture Letters, 1, February 2002.

[21] B. Towles, W. J. Dally, S. Boyd. "Throughput-Centric Routing Algorithm Design", ACM symposium on Parallel algorithms and architectures 2003, pp 200-209

[22] W. Wolf , FPGA-Based System Design, ISBN: 0131424610, June, 2004