Architecture and Routing in NOC Based FPGAs

Roman Gindin
rgindin@tx.technion.ac.il

Israel Cidon
cidon@ee.technion.ac.il

Idit Keidar
idish@ee.technion.ac.il

ABSTRACT

We present a novel network-on-chip architecture for future programmable chips (FPGAs). We examine the required capacity allocation for supporting a collection of typical traffic patterns on such chips under a number of routing schemes. Since in FPGA, traffic patterns are determined at configuration time, after physical links have been synthesized, it is important to employ routing schemes that allow for high flexibility in the permissible traffic pattern during configuration. We propose a new routing scheme, Weighted Ordered Toggle (WOT), and show that it allows for high design flexibility with low per-link capacity. Moreover, WOT utilizes simple, small-area, on-chip routers and has low memory demands.

Categories and Subject Descriptors

Multiprocessors and Network-on-chip
• Network-on-Chip (NoC)

Keywords

Network-On-Chip (NoC) architecture, FPGA, routing, capacity.

1. INTRODUCTION

In recent years, much attention has been dedicated to Networks-on-Chip (NoCs) (see, e.g., [1],[3],[4],[5],[6],[8],[11], [12]), which provide scalable solutions for on-chip communication in the sub-micron era. However, there is no "one size fits all" NoC architecture, as different silicon systems have very different requirements from their NoCs. For example, in a System-on-Chip (SoC), the network usage patterns are known a priory. Hence, the NoC can be synthesized with exactly the right link capacities for supporting the required usage [7]. In contrast, in a Field Programmable Gate Array (FPGA), the communication pattern is determined when the chip is configured to implement some specific functionality, and hence its physical layout must provide the flexibility to support a variety of traffic patterns.

In this paper, we focus on NoC design for FPGAs. A distinguishing 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.). Modern FPGAs contain hundreds of thousands of programmable elements, in addition to special purpose modules (0). As technology scales, the sheer number of logic units will render a flat FPGA chip design unmanageable. 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. Our novel architecture is detailed in Section 2.

When architecting an FPGA NoC, one has to decide which functionalities to implement as hard cores and which to leave soft. There is a tradeoff between the flexibility offered by soft designs and the better performance offered by hard ones. 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. That is, each module has a *configurable network interface* (CNI) Simplistic routing schemes, like XY, can employ small interfaces, whereas more elaborate source-routing schemes ([1],[9]) may have the interfaces store large routing tables.

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. Traditional algorithms like XY lead to unbalanced capacities, 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[15] [13]; 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. We present a new method, *weighted toggle XY* (WTXY), which computes the optimal ratio between XY and YX traffic (cf. Section 3.2). When the chip is programmed for a specific functionality, the traffic pattern becomes known, and the optimal weight is computed and loaded into all CNIs.

Unfortunately, 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 adding sequence numbers to packet headers. We therefore advocate the use of *ordered* algorithms, which do not split flows, and hence ensure in-order arrivals. We do this by selecting one of XY or YX routes to each source-destination flow. First, we examine *source toggle XY* (STXY), which splits the routes evenly between XY and YX, according to the parity of the source and destination IDs (cf. Section 3.3.1). Finally, 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. We compare three approaches for choosing routes in WOT (cf. Section 3.3.2). Like WTXY, 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 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 typical SoC and FPGA designs, communication load is not divided evenly among all modules. Rather, a handful of modules are *hotspots,* which communicate with many others 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 [12], etc. We therefore focus on traffic patterns involving multiple hotspots. We compare *unconstrained* placement of hotspots, where the designer can locate hotspots anywhere on the chip, with *constrained* placement. Our results (cf. Section 4) show that constrained placement can significantly reduce the capacity requirements. 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 be at least three hops; with three hotspots, unconstrained placement requires 35% more than constrained placement at distance at least three. WOT also reduces the maximum capacity. Its most loaded link has 40% less capacity compared to the maximum with XY, and up to 15% less capacity than with TXY.

1.1 Related Work

Sethuraman et al. [14] propose a soft NoC router design using current-day FPGA technology, which does not include any explicit (hard) NoC infrastructure. In contrast, we propose an architecture for future FPGA platforms, with some embedded NoC support and some configuration-time tuning. Hard routers offer better performance and cost less than soft ones.

DyNoC [10] is an architecture for adaptive routing using reconfigurable hardware. Unlike our solution, it does not perform offline optimizations at configuration time; instead, it dynamically changes routing policies at run-time. Since FPGA usage is determined at configuration time, offline optimization can be very effective, and moreover, greatly simplifies the router design, reducing hardware cost in terms of area and power consumption.

The TXY algorithm was independently developed by Seo et al. for interconnection networks under the name of O1TURN [13]. It has been shown to be worst-case near-optimal for *uniform* traffic, where all nodes send and receive at the same rate [15]. 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 is an unordered algorithm, and hence its use in chips would necessitate large re-order buffers. Towels et al.

[16] use linear programming to 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

2.1 Chip Layout

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 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., implemented as hard IP cores.

The network is a uniform mesh and each region is connected to a router via a local interface. The routers and links are hard, i.e., embedded in silicon, for maximum performance in terms of area and power. As this infrastructure is laid out when the actual application is not known, it is designed to support a large variety of applications. The router structure is similar to ones in previous NoC designs (e.g. [1],[2][5],[11]), implementing wormhole routing, and possibly multiple QoS classes [2].

Each region is connected to the network via a *configurable network interface (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). A CNI performs several functions: physical network interface, fragmentation/reassembly, buffering, reordering, (from datagrams to packets and from packets to wormhole flits), routing, and interface to the region modules. Of these, only the routing layer is configurable. Other functions, which are always required regardless of the chosen routing scheme, are hard coded in order to achieve better power/area/performance.

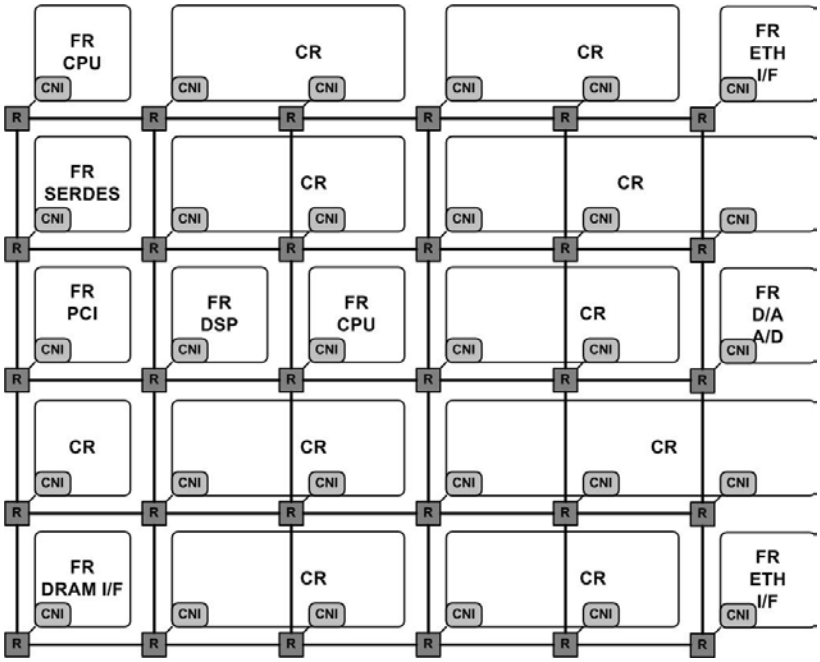

Figure 1. Example of network on programmable chip.

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 silicon. This allows for a regular mesh topology, and avoids the complexity that can result from an irregular mesh [2]. In the sequel, we consider an n x n uniform grid.

2.2 Cost Metrics

We define the following design efficiency measures to evaluate routing schemes under heterogeneous traffic requirements.

Maximum capacity – 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. A routing scheme should minimize the maximum capacity.

Design envelope – assignment of capacities to links so that the resulting mesh supports a given class of traffic requirement patterns. Each link's capacity in the envelope is its maximum capacity over all traffic patterns in the specified class. Uniform design envelopes are preferred.

Gate count –the number of gates required to implement a NoC using a specific routing technique. Specifically, we examine the router and the CNI, including routing table logic and buffers.

3. ROUTING SCHEMES

3.1 XY and Toggle XY

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 [7]. The YX routing uses the same technique but reverses the order of the vertical and horizontal routing.

Toggle XY (TXY) routes half the packets in the XY path and the other half in the YX path. The packet header includes a bit indicating whether its route is XY or YX. Each CNI toggles between XY and YX packets. Deadlocks are avoided using two virtual channels per router: one for XY paths and one for YX.

3.2 Weighted Toggle XY

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), where each node sends 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.

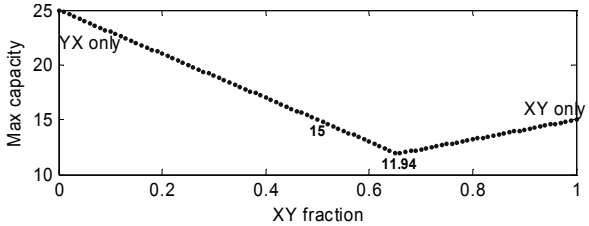

Figure 2. The impact of the XY fraction on 5x5 grid with two hotspots at locations (2,1) and (1,1).

We therefore introduce the **Weighted Toggle XY** algorithm, which chooses the optimal XY fraction, c_{xy}, according to given traffic requirements. The optimal fraction is calculated offline for each application during chip programming phase, and is loaded to the CNI with the rest of the configuration data. WTXY can be implemented using a counter 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. We next give formulas for computing the optimal ratio in various scenarios.

3.2.1 Analytical solution for single hotspot

In case of a single hotspot, the flows over the hotspot's incoming links are the largest in the grid. Consider a hotspot at coordinates (x,y), as shown in Figure 3 (coordinates range from 0 to n-1). Let c_{xy} be the fraction of traffic routed XY.

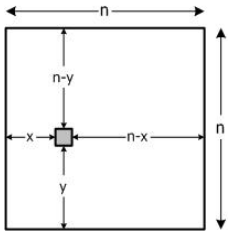

Figure 3. Location of a single hotspot.

The flows on the hotspots' incoming links are given by the following four equations:

$F_{left} = x + c_{xy} \cdot x \cdot (n-1); F_{right} = (n-x-1) + c_{xy} \cdot (n-x-1) \cdot (n-1)$
$F_{down} = y + (1-c_{xy}) \cdot y \cdot (n-1); F_{up} = (n-y-1) + (1-c_{xy}) \cdot (n-y-1) \cdot (n-1)$

For example, the flow from the left (F_{left}) 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. The best load balance occurs when the maximum horizontal and the vertical flows are the same. Define $\tilde{x}, \tilde{y}$ as follows: $\tilde{x} = \max(x, n-x-1); \tilde{y} = \max(y, n-y-1)$.

The above condition is satisfied if:

$\tilde{x} + c_{xy} \cdot \tilde{x} \cdot (n-1) = \tilde{y} + (1-c_{xy}) \cdot \tilde{y} \cdot (n-1)$. Consequently, the optimal ratio is: $c_{xy} = \frac{(\tilde{y} - \tilde{x}) + \tilde{y} \cdot (n-1)}{(n-1)(\tilde{x} + \tilde{y})}$. On a 5x5 grid, there are only six distinct scenarios out of the 25 possible hotspot locations, as shown in Figure 4 (Left). Figure 4 (right) depicts the maximum capacity required for each of these positions with each value of c_{xy}. The X axis in Figure 4 represents either the XY fraction or the YX fraction, depending on the side of the grid the hotspot is in.

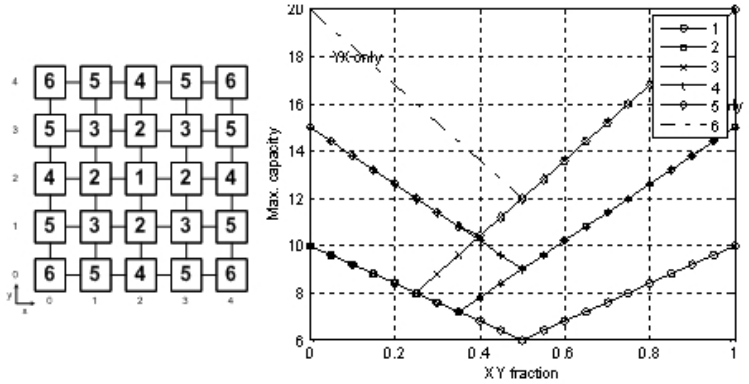

Figure 4. WTXY location categories for single hotspot and weighted routing for hotspots in various locations.

3.2.2 Analytical solution for two hotspots

Next, consider two hotspots HS1 and HS2 at coordinates (x_1, y_1) and (x_2, y_2), respectively, with the XY fraction c_{xy}. The worst-case load occurs when the two hotspots share a coordinate. We therefore focus on this case.

Without loss of generality, for symmetry reasons, we can assume the hotspots are located on the same vertical line, i.e., $x_1 = x_2 = x$. We compute the optimal fraction by equating the maximum horizontal and vertical flows. The formula

$F_{hor,\max} = (\tilde{x} + c_{xy} \cdot \tilde{x} \cdot (n-1)) + (c_{xy} \cdot \tilde{x}) = \tilde{x} + c_{xy} \cdot \tilde{x} \cdot n$ describes the maximum flow for horizontal links, where $\tilde{x}$ is as defined above. This is the sum of the flows induced by both hotspots. The vertical maximum flow occurs on the vertical links. We define: $\bar{y}_1 = n - y_1 - 1; \bar{y}_2 = n - y_2 - 1$. The flows are as follows:

$$F_{up,HS1} = 2 \cdot \left(\bar{y}_1 + \left(1 - c_{xy}\right) \cdot \bar{y}_1 \cdot (n-1) \right)$$
$$F_{down,HS1} = y_1 + \left(1 - c_{xy}\right) \cdot y_1 \cdot (n-1) + (\bar{y}_1 + 1) + \left(1 - c_{xy}\right) \cdot (\bar{y}_1 + 1) \cdot (n-1) =$$
$$= n^2 - c_{xy} \cdot (n-1) \cdot n$$
$$F_{up,HS2} = \bar{y}_2 + \left(1 - c_{xy}\right) \cdot \bar{y}_2 \cdot (n-1) + (y_2 + 1) + \left(1 - c_{xy}\right) \cdot (y_2 + 1) \cdot (n-1) =$$
$$= n^2 - c_{xy} \cdot (n-1) \cdot n$$
$$F_{down,HS2} = 2 \cdot \left(y_2 + \left(1 - c_{xy}\right) \cdot y_2 \cdot (n-1) \right)$$

We have to find the maximum of the four, and equate it with the horizontal flow. We show two cases here; the remaining two are similar. Case 1: $F_{vert,\max} = F_{up,HS1}$. In this case, the optimal fraction is: $c_{xy} = \frac{2 \cdot \bar{y}_1 \cdot n - \tilde{x}}{n \cdot (\tilde{x} + 2\bar{y}_1)}$. When $F_{vert,\max} = F_{down,HS2}$ the $\bar{y}_1$ is replaced with y_2. Case 2: $F_{vert,\max} = F_{down,HS1}$. In this case, the optimum is: $c_{xy} = \frac{n^2 - \tilde{x}}{n \cdot (\tilde{x} + n)}$.

Note that the fraction is independent of the hotspot's vertical coordinates.

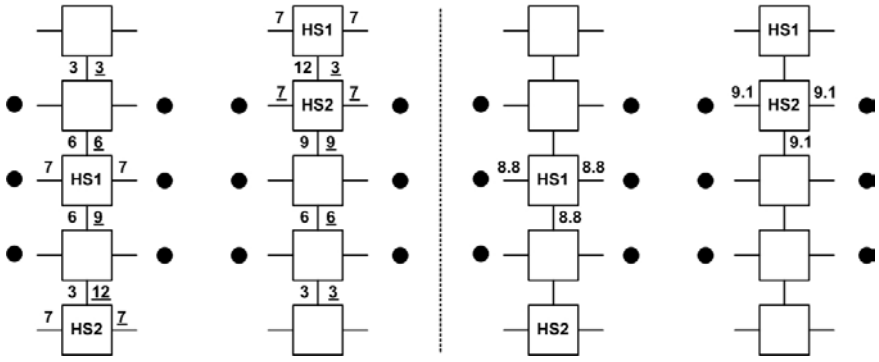

Figure 5. Two examples of flow balancing for two hotspots with WTXY (right) versus TXY (left).

Figure 5 illustrates examples of the two cases above. It shows first the (unbalanced) flows achieved with TXY, and then the balanced ones achieved with WTXY. The underlined numbers represent flows induced by HS2, and the others are flows by HS1. E.g., in Case 1 (depicted on the left), we see that the maximum vertical flow is 15 (12+3), and the maximum horizontal flow is 7. In contrast, with WTXY, according to the formula above, we get equal maximum horizontal and vertical flows of 8.8, a 40% improvement. For general location of the two hotspots we give the following analysis. We have two hotspots at (x_2, y_2) and (x_2, y_2). Without the loss of generality we assume that $x_1 < x_2$ and $y_1 > y_2$ as described at the following chart –

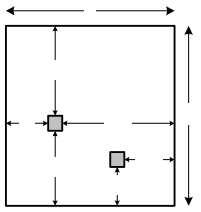

Figure 6. Location of two hotspots.

The flows on each hotspot caused by the hotspots thyself are –

$$F^1_{left,o} = x_1 + c_{xy} \cdot x_1 \cdot (n-1)$$
$$F^1_{rifht,o} = (n - x_1 - 1) + c_{xy} \cdot (n - x_1 - 1) \cdot (n-1)$$
$$F^1_{down,o} = y_1 + \left(1 - c_{xy}\right) \cdot y_1 \cdot (n-1)$$
$$F^1_{up,o} = (n - y_1 - 1) + \left(1 - c_{xy}\right) \cdot (n - y_1 - 1) \cdot (n-1)$$

And the second hotspots are –

$$F^2_{left,o} = x_2 + c_{xy} \cdot x_2 \cdot (n-1)$$
$$F^2_{rifht,o} = (n - x_2 - 1) + c_{xy} \cdot (n - x_2 - 1) \cdot (n-1)$$
$$F^2_{down,o} = y_2 + (1 - c_{xy}) \cdot y_2 \cdot (n-1)$$
$$F^2_{up,o} = (n - y_2 - 1) + (1 - c_{xy}) \cdot (n - y_2 - 1) \cdot (n-1)$$

The additional flows on the hotspots caused by another hotspot are

$$F^1_{left,i} = c_{xy} \cdot x_1 \qquad F^2_{left,i} = c_{xy} \cdot (n - x_2)$$
$$F^1_{rifht,i} = c_{xy} \cdot (x_1 + 1) \qquad F^1_{rifht,i} = c_{xy} \cdot (n - x_2 - 1)$$
$$F^1_{up,i} = (1 - c_{xy}) \cdot (n - y_1 - 1) \qquad F^1_{up,i} = (1 - c_{xy}) \cdot (y_2 + 1)$$
$$F^1_{down,i} = (1 - c_{xy}) \cdot (n - y_1) \qquad F^1_{down,i} = (1 - c_{xy}) \cdot (y_2)$$

Total flows on the hotspots are given by the sum of the flows on the links -

$$F^1_{left} = x_1 + c_{xy} \cdot x_1 \cdot (n-1) + c_{xy} \cdot x_1$$
$$F^1_{rifht} = (n - x_1 - 1) + c_{xy} \cdot (n - x_1 - 1) \cdot (n-1) + c_{xy} \cdot (x_1 + 1)$$
$$F^1_{down} = y_1 + (1 - c_{xy}) \cdot y_1 \cdot (n-1) + (1 - c_{xy})(n - y_1)$$
$$F^1_{up} = (n - y_1 - 1) + (1 - c_{xy}) \cdot (n - y_1 - 1) \cdot (n-1) + (1 - c_{xy})(n - y_1 - 1)$$

And the second hotspot -

$$F^2_{left} = x_2 + c_{xy} \cdot x_2 \cdot (n-1) + c_{xy} \cdot (n - x_2)$$
$$F^2_{rifht} = (n - x_2 - 1) + c_{xy} \cdot (n - x_2 - 1) \cdot (n-1) + c_{xy} \cdot (n - x_2 - 1)$$
$$F^2_{down} = y_2 + (1 - c_{xy}) \cdot y_2 \cdot (n-1) + (1 - c_{xy}) \cdot y_2$$
$$F^2_{up} = (n - y_2 - 1) + (1 - c_{xy}) \cdot (n - y_2 - 1) \cdot (n-1) + (1 - c_{xy})(y_2 + 1)$$

The minimum is achieved when –

$$\max(F^2_{up}, F^1_{up}, F^2_{down}, F^1_{down}) = \max(F^2_{right}, F^1_{left}, F^2_{right}, F^1_{left})$$

On a 5x5 grid we found that there are only 9 distinct categories of pairs of hotspot locations, so that all the pairs in each category behave similarly. For space considerations, we do not describe them here.

3.2.3 General algorithm

It is hard to analyze the general traffic pattern, so we present a heuristic algorithm -

```
Calculate the flows using Cxy = 0.5
IterCount = 0
While
   MAX,H = max (Horizontal Links) ;
   MAX,V = max (Vertical Links) ;
   IterCount++;
   if ( MAX,H == MAX,V | IterCount == MaxIter)   break;
   else if ( MAX,H > MAX.V )    Cxy = Cxy - D;
   else      Cxy = Cxy + D
  Calculate new flows;
```

The algorithm receives two parameters – i) maximum number of iterations ii) iteration step – D. The algorithm balances the worst horizontal link and the worst vertical link, by finding the most loaded links, and shifting the flow from the more loaded link to less loaded one by changing c_{xy} by D in each iteration. A larger value of D leads to faster running times, albeit less accurate results. In our simulations, we used a relatively small value of D – 0.01. The algorithm stops either after a predefined number of iterations, or when the maximum horizontal link load is equal to the maximum vertical one.

3.3 Ordered Routing Algorithms

The main problem of the algorithms described above is the flow splitting. The same flow is split over multiple routes and packet arrivals may be out of order. This requires the receiver CNI to maintain large reordering

buffers. Consider, e.g., a 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.

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.

The simplest ordered algorithm is **Source Toggle XY (STXY),** which toggles the routes according the source ID and destination ID: S-D pairs with odd parity of source and destination IDs send all their traffic using XY, while S-D pairs with even parity use YX. This gives an approximation to TXY, albeit with a quantization error.

However, as motivated above, equally splitting flows between XY and YX routes is not optimal. We therefore introduce **Weighted Ordered Toggle (WOT),** which divides S-D routes in a way that minimizes the maximum capacity. We present three techniques for route assignment:

Iterative Assignment – first calculates the c_{xy} ratio as for WTXY above. The algorithm then serially scans the nodes in the grid and assigns routes in such a way that XY ratio is as close as possible to the target c_{xy}.

Random Assignment – also first calculates the optimal ratio. Then, for each S-D pair, it chooses XY routing with a probability equal to the optimal ratio, and otherwise YX.

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 of the perpendicular direction to *l*, for example if *l is* a horizontal link, then the algorithm minimizes the load on vertical links. 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 better performance than WTXY, since it is not limited to a single c_{xy} ratio for the entire network.

4. NUMERICAL EVALUATION

We now evaluate the capacity requirements of the different routing schemes under typical on-chip traffic patterns.

4.1 Comparing WOT Assignment Schemes

The following graph compares the three assignment approaches for WOT. We depict the difference between the achieved maximum capacity and the maximum capacity of WTXY, for simulations with a random number of hotspots in the range (1-3) located at random places on grids with sizes ranging from 5x5 to 10x10. For each grid size, we perform 1000 simulations. The y-axis is $C_{\max}^{WOT} - C_{\max}^{WTXY} \Big/ C_{\max}^{WTXY}$.

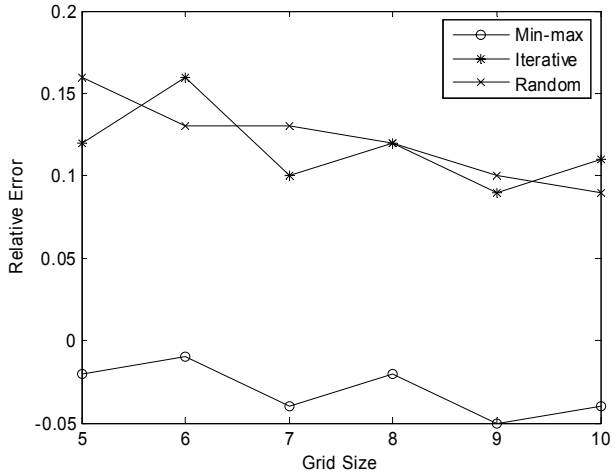

Figure 7. Relative overhead for WOT route assignment schemes vs. WTXY.

We observe that the min-max scheme produces the best results and is sometimes even better than WTXY. Subsequently, we evaluate WOT only using min-max assignment.

4.2 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.

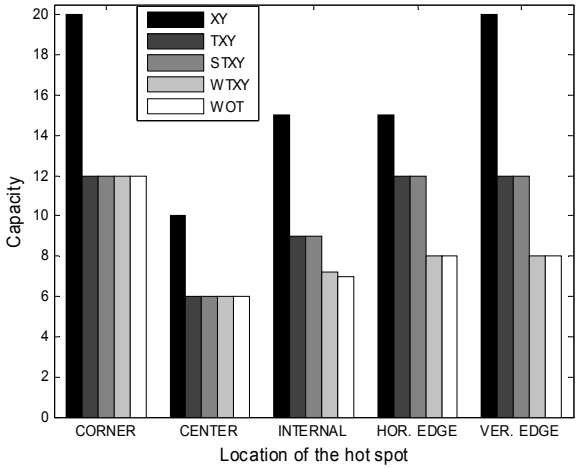

Figure 8. Maximum capacity requirements for one hotspot on a 5x5 uniform grid with different routing schemes.

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, the other algorithms considerably reduce the capacity requirements, and all 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 is not optimal, and WTXY and WOT can improve upon TXY and STXY by up to 33%.

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 the placement of hotspots. Some hotspot modules provide external interfaces and must therefore reside at the edge of a chip. Nevertheless, it is possible to require locating them in the middle of an edge rather than in a corner, and save 33% in capacity. **Figure 9** depicts the design envelope for all possible locations of the hotspots and Figure 108 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. We can also see that the number of links with the lowest capacity is higher for WOT the WTXY.

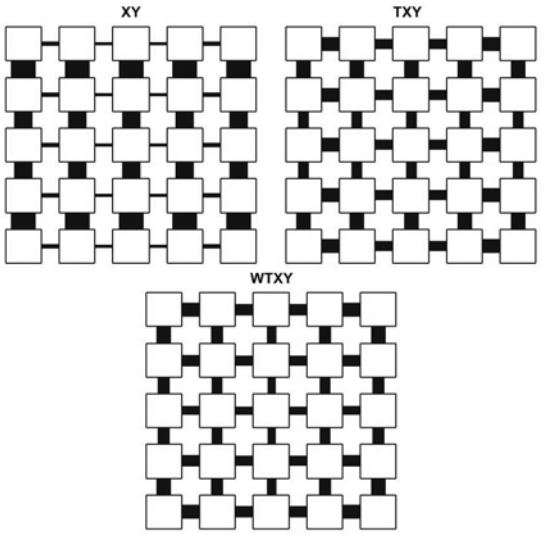

Figure 9. Design envelopes for all single hotspot locations for various routing schemes.

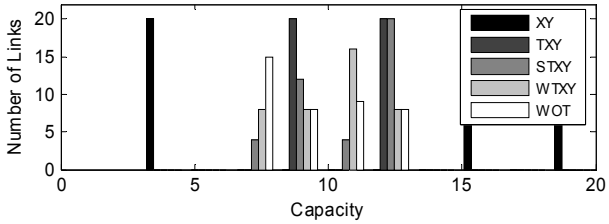

Figure 10. Histogram of link capacities in design envelopes of all possible single hotspot locations.

We also demonstrate the load balance of the algorithms on the following example. The following graph shows the performance of the various routing scheme for an example of 5x5 grid with single hotspot at (4,1) coordinate.

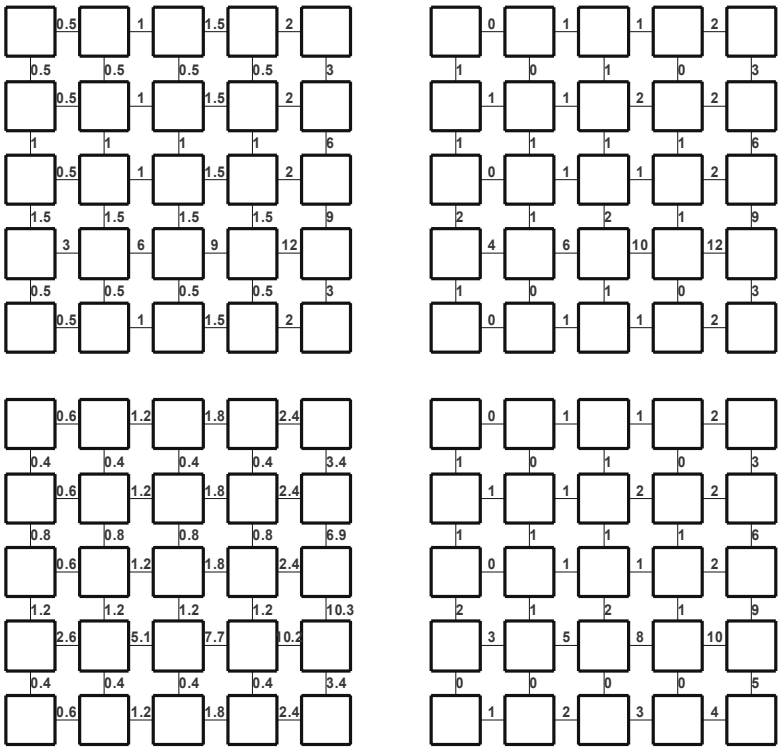

Figure 11. Example of the WOT algorithm performance.

We can see that the maximum capacity and capacity distribution of STXY are very similar to those of TXY. Moreover, the maximum capacity of WTXY is a bit larger (10.3) than the maximum capacity of WOT, (which is 10). In term of lexicographical score (when the link capacities are sorted and compared lexicographically), WOT also achieves a better result.

4.3 Two Hotspots

Next, we examine traffic patterns involving two hotspots. We simulated all possible locations of two hotspots on 5x5 grid and we depict the worst case maximum capacity for each possible distance between the hotspots. We observe that in this case, the capacity requirements are 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 enough 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 129. 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 than unweighted ones, and that ordering (STXY versus TXY and WOT versus WTXY) does not significantly impact the performance.

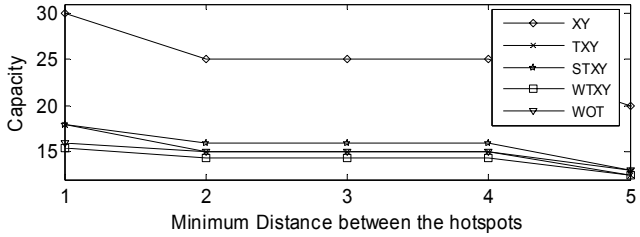

Figure 12. Maximum worst case link capacity required for two hotspots versus the distance between them.

Figure 14 presents the envelopes for supporting all possible locations of two hotspots at a given distance with WTXY. Figure 13 shows the corresponding histograms.

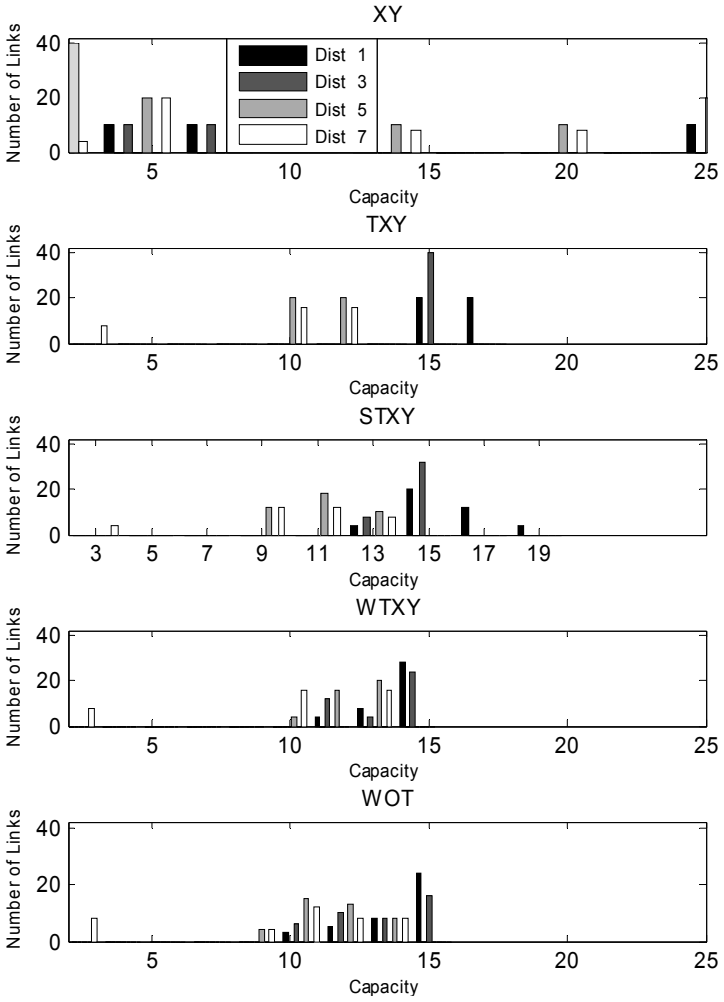

Figure 13. Histograms of link capacities in design envelopes of all possible two hotspot locations at various distances.

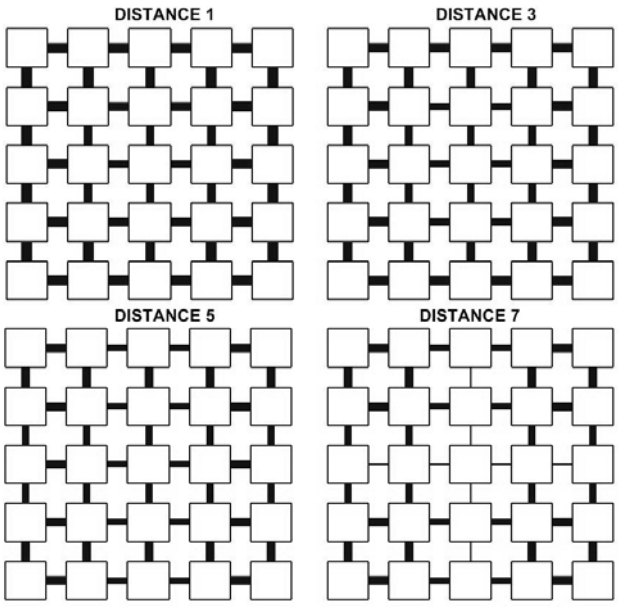

Figure 14. Envelopes for all locations of two hotspots at various distances with WOT routing.

4.4 Three Hotspots

Here, we examine the case of three hotspots. As in the previous case we simulated all possible locations, and show the worst case maximum capacity per minimal distance between the hotspots. The minimal distance between the closest pair among the three hotspots is dominant in determining the load, as shown in Figure 1510. The weighted algorithms outperform the non-weighted ones by 20% on average, and more so for unconstrained placement. Again, the ordered versions are very close to the flow-splitting algorithms.

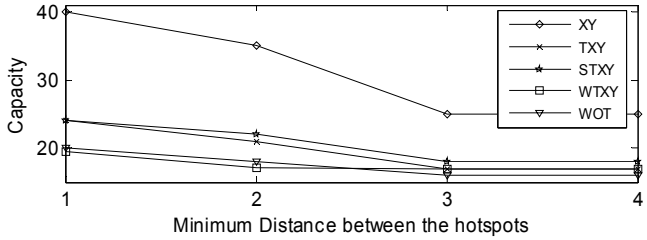

Figure 15. Maximum link capacity required for three hotspots versus minimal distance between a pair of them.

The histogram of required link capacities in the envelopes of all locations of three hotspots with different minimal distances appears in Figure 16. We observe that requirements for different links with WOT are very similar and balanced, and hence a uniform grid can be used without much waste.

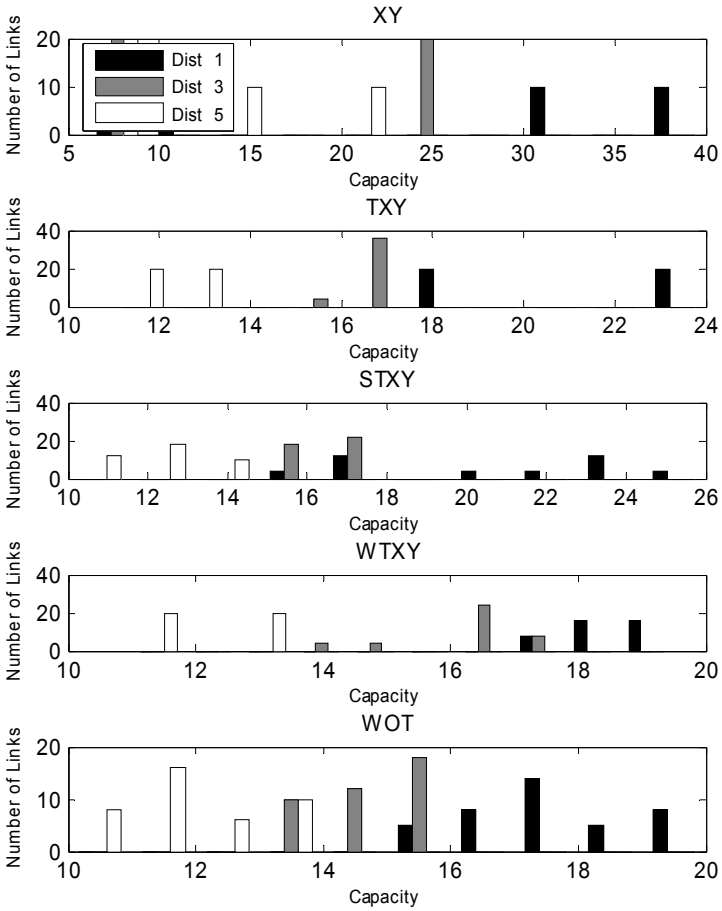

Figure 16 . Histograms of link capacities in design envelopes of all possible three hotspots locations at various distances.

4.5 Complex Traffic Patterns

Finally, we compare the routing algorithms using more complex traffic patterns. We use the traffic model described in [2]. This random model has three parameters: (i) the probability for each node to be a hotspot (P_{hs}), (ii) the probability for a node to send data to each hotspot (P_{hs}^{send}); and (iii) the probability to send data to every non-hotspot node ($P_{no,hs}^{send}$). For symmetric graphs, the weighted algorithms perform similarly to non-weighted ones, but as the asymmetry of the traffic pattern increases, the influence of the weight grows. **Figure 1711** presents the average maximum capacity observed in 100 random patterns for increasing grid sizes, when 10% of the nodes are hotspots, 80% of nodes send traffic to each hotspot, and 5% of the nodes send traffic to each non-hotspot.

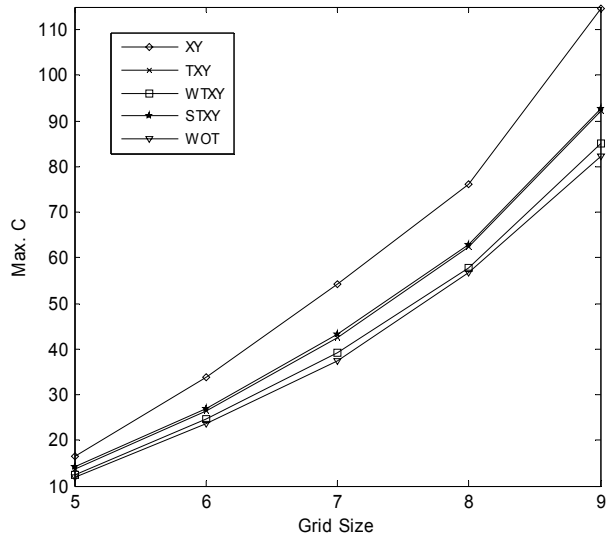

Figure 17. Comparison of routing schemes for $P_{hs} = 0.1$ and $P_{hs}^{send} = 0.8$, $P_{no,hs}^{send} = 0.05$ for various grid size.

We again see that WOT outperforms all other routing algorithms**, and that weighted algorithms outperform non-weighted. Average improvement is about 8-10% for WTXY vs. TXY and about 15% for WOT vs. STXY.** We also demonstrate on the following figures the design envelope for WOT algorithm for 5x5 case for the same communication parameters as above.

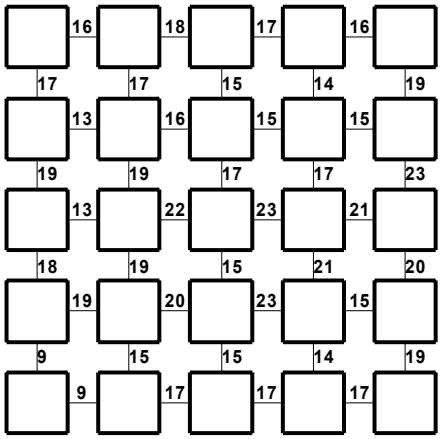

Figure 18 – WOT design envelope for complex traffic pattern.

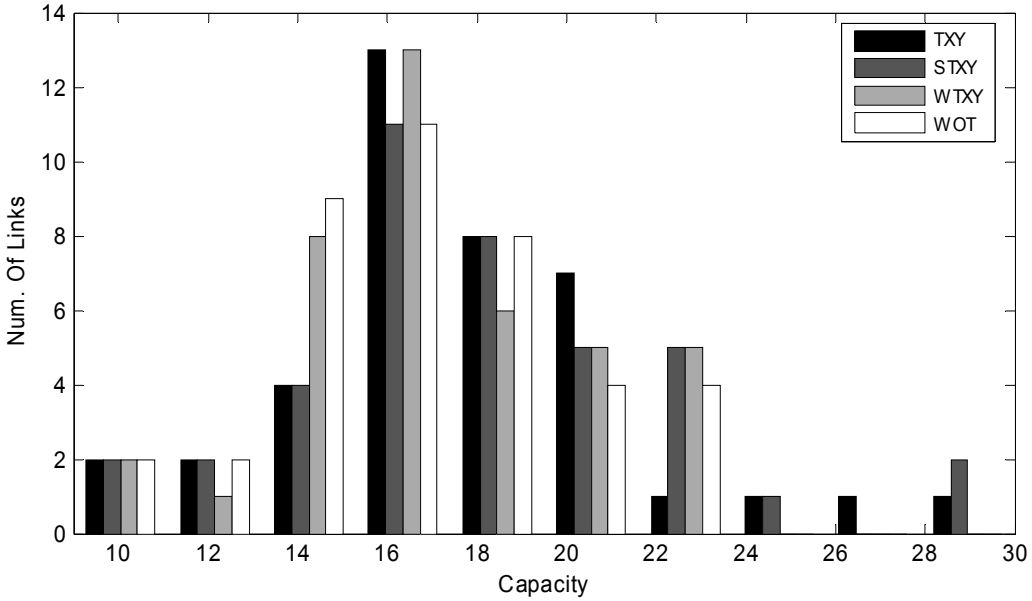

Figure 19 – Histogram of link capacities in the design envelope of complex data pattern on a 5x5 grid for various routing schemes.

We can see that WOT is more balanced compared to STXY and that the amount of links with large capacities for WOT is smaller for other algorithms, whereas the amount of links with smaller capacities is larger.

5. HARDWARE COSTS

This section analyzes the costs of the hardware for the network on programmable chip for the different routing schemes. As note above, a simple router with 2 virtual channels to avoid deadlock [13] supports all routing techniques presented in this work. 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 implemented using programmable logic to allow flexibility. Figure 20 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.

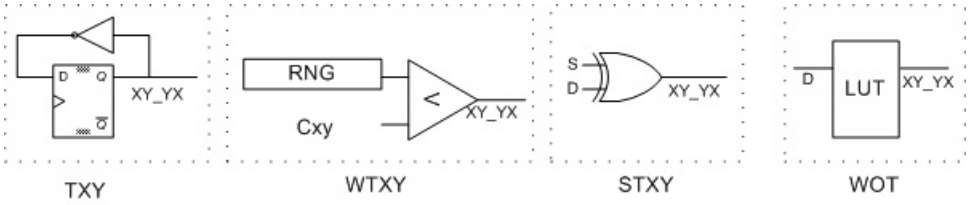

Figure 20 – Routing Level Implementation

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 - $(\log(n^2))$.

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. [17] 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. Table 1 summarizes the required gates –

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	$\left\lvert \frac{1}{2} \cdot \log\left(2 \cdot \log\left(n^2\right)\right) \right\rvert$	3	Bitwise xor of $\log\left(n^2\right)$ word width. Each LUT implements XOR of 4 bits
WOT	$\left\lvert n^2 / 16 \right\rvert$	2	Each LUT acts as a 16 bit ROM. We need to cascade several LUTs to perform the required 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.

6. CONCLUSIONS

We have presented a new architecture for future programmable chips. We have studied routing schemes that can be used in this architecture, and their impact on capacity requirements. We presented a simple yet efficient routing algorithm, WTXY, which can be configured to balance link loads according to traffic patterns announced when the chip is configured. Due to the flow splitting problem, we suggested an ordered algorithm, WOT; which exhibits similar performance to WTXY, and in some cases even outperforms it, while eliminating the need for packet re-ordering. In this algorithm all the traffic between the same source-destination pair is sent over the same route. Our algorithm is also very efficient in terms of gate counts, taking as little as 2 LUTs for 5-bit Ids.

7. REFERENCES

[1] 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.

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

[3] E. Bolotin, et al., "QNoC: QoS Architecture and Design Process for Networks on Chip", Journal of Systems Architecture (JSA), Feb 2004.

[4] W. Dally and B. Towles, "Route packets, not wires", Design Automation Conference (DAC), Jun. 2001, pp. 684-689.

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

[6] Guerrier, Greiner: "A Generic Architecture for On-Chip Packet- Switched Interconnection", Date Automation and Tests in Europe (DATE), March 2000.

[7] Z. Guz, I. Walter, E. Bolotin, I. Cidon, R. Ginosar, A. Kolodny. "Efficient Link Capacity and QoS Design for Network-on-Chip", Date Automation and Tests in Europe (DATE), 2006

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

[9] 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

[10] M. Majer, C. Bobda, A. Ahmadinia, J. Teich, "Packet Routing in Dynamically Changing Networks on Chip", IEEE International Parallel & Distributed Processing Symposium 05 (IPDPS) – Workshop 3, p 154b.

[11] 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.

[12] A. 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

[13] D. Seo, A. Ali, W.-T. Lim, N. Rafique, M. Thottethodi. "Near-optimal worst-case throughput routing for two-dimensional mesh networks", International Society for Computers and Their Applications (ISCA), 2005.

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

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

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

[17] Wayne Wolf , FPGA-Based System Design, ISBN: 0131424610, June, 2004