Configurable Links for Runtime Adaptive On-chip Communication

Mohammad Abdullah Al Faruque, Thomas Ebi, and Jörg Henkel
University of Karlsruhe, Chair for Embedded Systems, Karlsruhe, Germany
{alfaruque, ebi, henkel} @ informatik.uni-karlsruhe.de

***Abstract*— Reliability concerns associated with upcoming technology nodes coupled with unpredictable system scenarios resulting from increasingly complex systems require considering runtime adaptivity in all possible parts of future on-chip systems. We are presenting a novel configurable link which can change its supported bandwidth on-demand at runtime (2X-Links) for an adaptive on-chip communication architecture. We have evaluated our results using real-time multi-media and the E3S application benchmark suits. Our 2X-Links provide a higher throughput of up to 36%, with an average throughput increase of 21.3%, compared to the Normal-Full-Duplex-Links [12], [14], [17], [20] and keep performance-related guarantees with as low as 50% of the Normal-Full-Duplex-Links capacity. Our simulation shows when some links fail, the NoC with 2X-Links can recover from these faults with an average probability of 82.2% whereas these faults would be fatal for the Normal-Full-Duplex-Links.**

I. Introduction and Related Work

Semiconductor industries have kept alive Moore's law with the innovation of novel process technologies and this is expected to continue as long as our system requirements expand at the present rate. In the current process technology shift from 65nm to 45nm, the number of logic gates per square millimeter has increased from 700,000 to 1,4 million [10]. Due to several practical limitations the process technology is no longer able to provide reliable silicon fabrics with the continuing technology scaling and therefore, reliable systems have to be built from unreliable fabrics [1], [3], [16]. The burden of continuing on the technology roadmap to meet the increasing computational demand is now on the shoulders of architecture and system-level engineers. In the continuing process of novel architectures, a *Multi-Processor System on Chip* (MPSoC) plays an important role in satisfying the increased computational demand. The more complex a system grows using these unreliable fabrics, the more it must also be able to handle situations arising from these inherent defects efficiently. These situations cannot be predicted at design time and thus the *System on Chip* (SoC) needs to be designed with the capability of self-adaptiveness in mind [1], [3], [11], [8].

The design methodology for the MPSoC and its on-chip interconnect has already been shifted towards *Networks-on-Chip* (NoCs) as it is envisioned that future MPSoCs will be predominantly communication-centric [2], [5]. Recently, several general-purpose NoCs such as Tile64™ by Tilera [20] and an 80-core processor from Intel [12] have been fabricated. So far, the research in the domain of NoC has focused on application-specific[1] NoCs [2], [15] and design-time parameterized general-purpose[2] NoCs [12], [20]. In order to achieve reliable communication in a MPSoC, the NoC must be extended with adaptive capabilities e.g. the ability to change the traffic route at runtime in order to bypass faulty areas efficiently. Here, fault-tolerance ability is limited to the faults in the links between two adjacent routers.

Taking this reliability issue together with other issues, e.g. the user-behavior, into consideration, we have proposed a reliable adaptive on-chip communication infrastructure providing adaptivity in the architecture-level in [8]. Furthermore, the system-level adaptation is provided using a runtime agent-based distributed application mapping [9]. The architecture-level adaptation is implemented by using several novel methodologies to increase the resource utilization of the underlying silicon fabric, i.e. sharing the virtual channel buffer among different output ports and changing the routing at runtime [8]. In the scope of this paper we concentrate on a novel part of the architecture-level adaptation namely the runtime configurable links, the *2X-Links*.

Communication links in the NoCs typically employ full-duplex communication using two simplex links [12], [14], [17], [20]. In between two adjacent routers there are two simplex links i.e. one from *Router 1* to *Router 2* and another from *Router 2* to *Router 1*. Both of these uni-directional links together form the full-duplex links. The links are design-time parameterized (e.g. the bit-width of the link) and therefore, provide a fixed link capacity in terms of bandwidth for a certain clock frequency. We observed that there may be several scenarios (some are given below) where we may require configurable links which can adjust their supported bandwidth capacity.

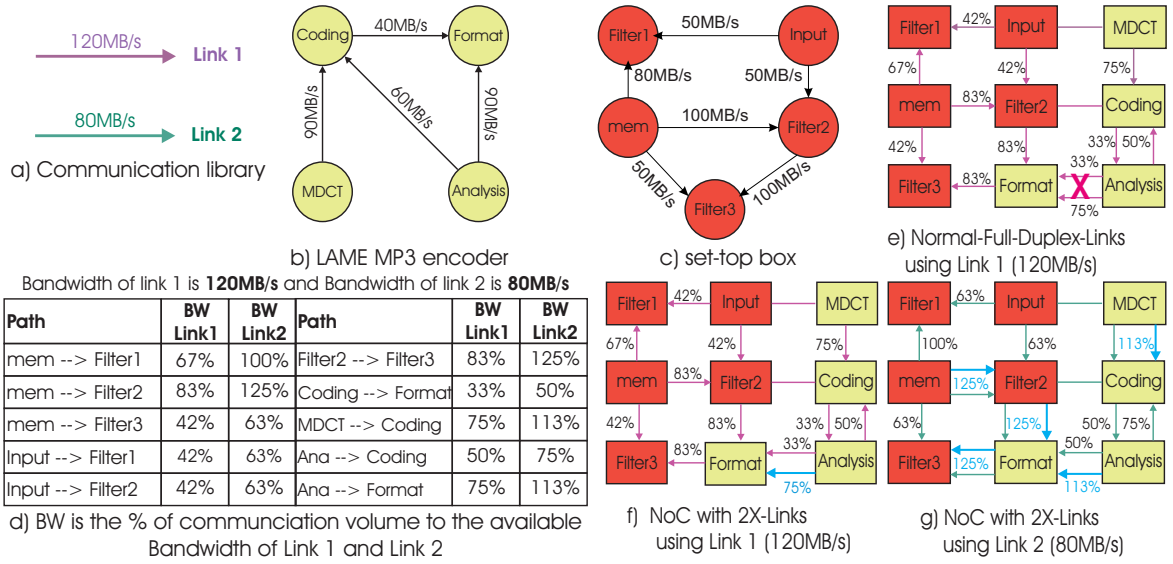
- As the number of possible simultaneous inter-task connections increases in a link in one direction, it becomes more difficult to meet the bandwidth requirements of all tasks. Therefore, there might be situations where a certain link requires an increase in link capacity (assuming all the transactions are flowing in a single direction).
- Sometimes, the bandwidth requirements of tasks expand according to user requirement changes (e.g. the user wants to switch video playback to a higher resolution). If in such a scenario current link capacity of one half of the duplex link cannot meet the requirement then the other half can be reversed to add more available bandwidth (doubled the link capacity in our *2X-Links*)
- When a hardware fault in a simplex link, e.g. in the link from *Router 1* to *Router 2* occurs, the transactions via this link in this direction will not succeed. In such a scenario, if the other simplex link (from *Router 2* to *Router 1*) has the ability to reverse its direction while free, it could transmit these transactions. Therefore, it increases the resource utilization and improves reliability factors through runtime adaptation.

To the best of our knowledge, there has been very little work which we are reviewing in the following on configurable links for NoC architectures. The state-of-the-art NoCs have full-duplex links which are design-time parameterized and these links do not consider scenarios where the demand for the given link capacity can increase depending on the current traffic load. A power-aware network which responds to the bursts

[1]Application specific NoCs are design-time parameterized architectures mainly with a custom topology, fixed routing scheme, and a fixed number of allowed virtual connections at each output port [2], [8].

[2]Over designed (e.g. number of virtual channels) considering different types of traffic scenarios and can not adapt architectural parameters i.e. buffer assignment or link capacity at runtime (lower resource utilization).

978-3-9810801-5-5/DATE09 © 2009 EDAA



Bandwidth of link 1 is **120MB/s** and Bandwidth of link 2 is **80MB/s**

Path	BW Link1	BW Link2	Path	BW Link1	BW Link2
mem --> Filter1	67%	100%	Filter2 --> Filter3	83%	125%
mem --> Filter2	83%	125%	Coding --> Format	33%	50%
mem --> Filter3	42%	63%	MDCT --> Coding	75%	113%
Input --> Filter1	42%	63%	Ana --> Coding	50%	75%
Input --> Filter2	42%	63%	Ana --> Format	75%	113%

d) BW is the % of communciation volume to the available Bandwidth of Link 1 and Link 2

Fig. 1. Motivational example to show the requirement of 2X-Links

and dips in traffic by turning links on and off at runtime has been proposed by the authors in [18]. Their approach, however, is based on the assumption that the future traffic patterns are predictable based on the observation of the past traffic. Hard-to-predict events e.g. rapid motion in a video sequence cannot be treated efficiently. The *FLUX* network [19] is proposed to establish interconnections on-demand before or during program execution by adapting the physical network. While creating point-to-point connections, the deployed links need to be reserved. This approach does not scale well as the number of required links increases exponentially with the number of processing elements.

The rest of the paper is organized as follows. After presenting the motivational case study analysis in Section II, in Section III we introduce our *runtime adaptive on-chip communication* architecture in short, whereas in Section IV our novel *2X-Links* mechanism is explained in detail. Our hardware implementation for the *2X-Links* is presented in Section V. Experimental results are discussed in Section VI with Section VII concluding the paper.

II. Motivational Case Study Analysis

For a motivational example of the proposed runtime configurable links we consider a 3×3 mesh NoC, two applications: the LAME MP3 encoder and a set-top box application (Fig. 1 (b,c)), and two macro libraries: the IP library not shown in the figure and the communication link library Fig. 1 (a). Fig. 1 (d) shows that using the current configuration $Link1$ can meet the requirement of each transaction while $Link2$ fails (showing more than 100%) in 5 of the given transactions.

We now assume that at a certain time t_0, both applications need to be executed in parallel. In an initial routing attempt by a normal NoC, the traffic from *Coding* (here task name represents the tile where it is mapped to) is routed through *Analysis* to *Format* in Fig. 1 (e). This however fails, as the combined bandwidth required by the transactions exceeds the link capacity. With the *wXY-routing* algorithm proposed in [8], the tasks are able to choose different routes but in this scenario, the algorithm is unable to find a valid route. We can also try to use the runtime (re-)mapping mechanism proposed in [9] to solve this problem, but for this case we have investigated no instance of mapping can meet the bandwidth constraints.

Under this presented scenario, the state-of-the-art mechanism proposed in [13], splitting the traffic across multiple routes between the source and destination (traffic-splitting) may be used to expand the link capacity. However, this approach leads to higher packet latency, flit rearranging, unpredicted network situations, and may cause an increase in communication bottlenecks. An ideal solution may be to expand the link capacity of that particular link at runtime. Therefore, we have investigated the possibility of using link reversal mechanisms e.g. if the link from $Format$ to $Analysis$ can be reversed, then the bandwidth capacity in the opposite direction will be doubled and all the transactions will be successful (see Fig. 1 (f)). We have further investigated that we are able to decrease the available bandwidth even more by using only $link2$ from the communication library while still being able to successfully accommodate all of the transactions (see Fig. 1 (g)).

Our novel contribution is as follows: we present a runtime configurable link (*2X-Link*) at the architecture-level to compliment our existing adaptive on-chip communication infrastructure. The *2X-Links* can adapt the link capacity by changing the direction at runtime on demand, thereby increasing the resource utilization while considering the reliability issues. The building blocks of *2X-Links* are two *half-duplex* links instead of *simplex* links.

III. Our Runtime Adaptive Architecture

Our runtime adaptive on-chip communication architecture is capable of supporting deadlock-free data transmission and meets required bandwidth guarantees in a network exposed to varying system constraints and/or mode switches. We have used a transaction-level connection-oriented approach on top of packet-based communication to provide performance-related guarantees i.e. bandwidth requirements for critical transactions, as well as a pure best effort approach for the rest of the transactions (a hybrid approach). Similar to most NoCs, we have taken a pipelined architecture. The communication is packet-based with each packet being further partitioned into flits to allow wormhole routing. The topology is kept as a regular 2D mesh. Our runtime adaptive scheme is divided into two main parts: the *system-level*, and the *architecture-level*.

A. System-level Adaptation

Adaptivity at system-level is deployed using a runtime agent-based (an *agent* is a computational entity, realized in software, and acts on behalf of other entities) distributed application mapping. The detailed scheme is given in [9] and is not part of this work. In summary, to obtain a scalable

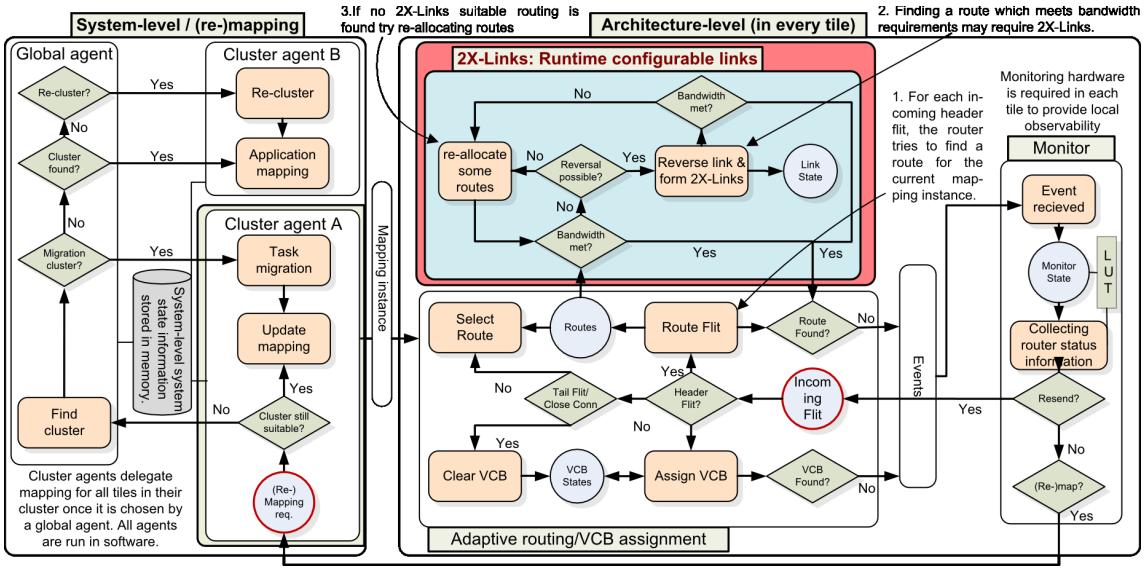


Fig. 2. 2X-Link workflow together with our complete adaptive on-chip communication architecture

mapping solution we have reduced the computational load by confining mapping to *clusters* which are a connected subset of tiles. The clusters have a variable size that can be adjusted at runtime and each cluster has one *cluster agent* which is responsible for (re-)mapping.

There are several reasons for (re-)mapping at different levels e.g. user-behavior from the application-level or hardware faults from the architecture-level. The cluster agent first tries to find a suitable (re-)mapping for a mapping request. If it is not able to establish a mapping instance it informs a global agent. The global agent then tries to resize the cluster associated with the cluster agent. If this fails, a different cluster is chosen and a new mapping is done. All agents are implemented in software and may be migrated to run on any PE in every tile within their deployment area (i.e. in a cluster or globally).

B. Architecture-level Adaptation

After the system-level has successfully set up a mapping instance, it is up to the architecture-level to configure each tile for the resulting connections. Once a transaction arrives at a tile, all possible directions must first be checked for suitable routes. To find a valid route which can meet the bandwidth constraints for the transaction, we may need to use the concept of *2X-links*. The number of virtual channel buffers per port has typically been a design-time parameter [2], [12]. On-demand buffer assignment where virtual channel buffers are tied to routers and not to individual ports allows a router to distribute the virtual channel buffers as needed to any possible route. Thus, the virtual channel buffers are assigned to transactions which in turn are assigned to an output port

C. On-chip Communication Links

Some link-related terms to explain our proposed *2X-Link* are discussed below:
A *simplex link* is a link where the transaction is allowed in only one direction. A *half-duplex link* provides communication in both directions, but only in one direction at a time. A *full-duplex link* allows communication in both directions simultaneously. Two *simplex links* may be combined together to form a *full-duplex link* e.g. links in the current NoC architectures [12], [14], [17], [20].

A *2X-Link* is a combination of two *half-duplex* links and their transmission can be configured to three different modes (both in one direction, both in the reverse direction, and both in opposite directions). For our runtime adaptive on-chip communication architecture we have used these *2X-Links* which is the main contribution in this paper.

A *Time-Division-Duplex-Link* (TDD-Link) uses time division multiplexing to separate inward and outward transactions. It emulates full-duplex communication over a single half-duplex link (the bit width of the single half-duplex link is doubled to compare it with the *2X-Links*). Time division duplex has a strong advantage in case where the asymmetry of the forward link and reverse link data speed is variable. As the amount of forward link data increases, more bandwidth can dynamically be allocated to that and as it shrinks, it can be taken away.

IV. Runtime Configurable Links

We have implemented configurable *2X-Links* for adapting the link capacity at runtime. If in a *Normal-Full-Duplex-Link* the link capacity is **"X"** then our proposed *2X-Links* may adjust its link capacity among **"Zero"**, **"X"**, and **"2X"**. We have also investigated the possibilities of incorporating the *TDD-Links* as a runtime configurable link in our on-chip communication architecture and then have compared the *TDD-Links* with our configurable *2X-Links*. The *2X-Links* have the advantage of lower area and fault-tolerance ability over the *TDD-Links*. Its limitations are: transmission can only be supported in one direction and the bandwidth granularity in the link is course-grained (0, X, and 2X). On the other hand, the *TDD-Links* can adjust their time slot to support bi-directional transactions and at the same time can configure the runtime link capacity in one direction in a fine-grained way but suffer from higher area overhead and fault-tolerance ability. Considering the area overhead and the fault-tolerance ability we have implemented the *2X-Links* as an integral part of our runtime adaptive on-chip communication architecture.

The concept of link reversal used in the *2X-Links* is not new and has been borrowed from *Telecoms* [7] and *Wireless Sensor Networks* [4] research. A diagram showing the workflow of *2X-Links* mechanism incorporated in our runtime adaptive on-chip communication architecture [8] is presented in Fig. 2. At a given time t, application G is requested to be mapped from the system-level part onto the NoC architecture-level. In the architecture-level our purpose is to find a suitable route allocation that meets all the requirements for the transactions, i.e. the bandwidth is met and then assigning virtual channel

buffer on-demand to that direction (on-demand virtual channel buffer assignment is not in the scope of this paper and it has been published in [8]) for the current instance of mapping. There are several possible situations after a mapping instance has been passed down to the architecture-level while searching for a suitable route (see Algorithm 1):

- If the adaptive router can find a suitable route for all the transactions, then the virtual channel buffers are assigned to the transaction direction.
- If the adaptive router for the current instance of mapping can not meet the performance-related guarantees for all the transactions, we search for an appropriate link which may be reversed to form a *2X-Link* in its transmission direction to help performance-critical transactions. When the feedback value is *true*, then the virtual channel buffers are assigned accordingly.
- If no suitable link can be reversed then we have to (re-)allocate some routes to relieve the link resource congestion, since the computational complexity for the route allocation is less than the (re-)mapping.
- If both the link reversal and route (re-)allocation do not work, (re-)mapping is the only way to meet the performance-related guarantees as proposed in [8], [9].

When the last option does not lead to a successful application execution that meets performance-related guarantee then the (re-)mapping request will be refused by the system.

Algorithm 1 2X-Links for Adaptive on-Chip Communication

avB_{dir}: Available bandwidth in the direction dir
$reqB_{dir}$: Bandwidth requested by transactions towards dir
$usedB_{dir^{-1}}$: Bandwidth used in the opposite direction of dir
QoS: performance-related guarantees (bandwidth, latency, etc.)

```
1: upon receiving Mapping, and Route do
2: if QoS are not met then
   // try to reverse links
3:    for all links with avB_dir < reqB_dir do
4:       if usedB_dir^-1 = 0 then
5:          reverse link
6:       end if
7:    end for
8:    if QoS requirements are not met then
      // link reversal failed
9:       re-route
10:      if QoS requirements are not met then
         // retry to reverse links
11:         for all links with avB_dir < reqB_dir do
12:            if usedB_dir^-1 = 0 then
13:               reverse link
14:            end if
15:         end for
16:      end if
17:      if QoS requirements are not met then
         // re-routing failed
18:         (re-)map task graph onto NoC
19:         call self
20:      end if
21:   end if
22: end if
```

A. Adaptive 2X-Link Routing Algorithm

A modified version of the *wXY-routing* algorithm [8] is able to successfully deal with the *2X-Links* by doubling the link bandwidth in the weight calculation. The *wXY-routing* algorithm is a dynamic routing method for 2D mesh networks. It is designed to be adaptive on the packet/transaction level but remains deterministic on the flit level. The *wXY-routing* algorithm selects a route based on *X* and *Y* distances and available bandwidth. Unlike the *XY-routing* algorithm which routes first in the *X* direction and then in the *Y* direction, it assigns a weight to each output port based on the available bandwidth as well as the horizontal and vertical distances from the current tile to the destination tile. The latter two are considered in order to maximize sensible routing choices along a packets route.

Eq. 1 shows the weight function which is suitable for the *2X-Links* system. In this equation, avB_{dir} represents the available bandwidth in the direction dir *(N,E,S,W)*, while O_{dir} indicates the number of hops between the current and the destination tile in the direction dir (i.e. Y for N, S; X for E, W). Adding the total link bandwidth B_{dir} assures that packets are routed in the appropriate direction if possible. For instance, if $|O_{dir}| = 1$, not adding the total bandwidth would result in the weights going towards the destination's *dir* and going away from it being the same. The key variable in this equation is *r*. If the *2X-Link* is activated, the link capacity will be doubled.

$$w_{dir} = \begin{cases} avB_{dir} \times |O_{dir}| + r \times B_{dir} & \text{if } dir \text{ towards } dest \\ 0 & \text{if } avB < reqB \\ r \times avB_{dir} & otherwise \end{cases} \quad (1)$$

subject to: r=2 if 2X-Link is activated; r=1 else.

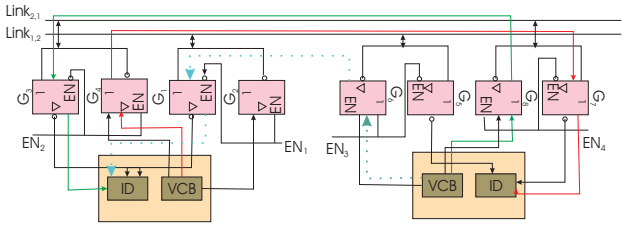


Fig. 3. Hardware of *2X-Links* using tri-state gates

V. Hardware Prototype

The architecture of the bi-directional link control using half-duplex links is presented in Fig. 3. Two tri-state logic gates e.g. G_1 and G_2 work as a group. They are controlled by the same control signal $E\bar{N}_1$. When the value of $E\bar{N}_1$ is *0*, the tri-state gate G_1 is active while G_2 is stalled. Then data can be transmitted from neighboring virtual channel buffers through $Link_{1,2}$ to the component *input decoder*. On the other hand, if the value of $E\bar{N}_1$ is *1*, the tri-state gate G_1 will be stalled and data can be sent from the virtual channel buffer to the adjacent router's input decoder via $Link_{1,2}$. Therefore, the half-duplex communication on one link is realized. In this architecture, the signal $E\bar{N}_1$ and $E\bar{N}_3$ control the data transmission via $Link_{1,2}$ while signal $E\bar{N}_2$ and $E\bar{N}_4$ control the data transmission via $Link_{2,1}$. They must be assigned different values, or there will be some conflict on the link and an unpredictable situation may occur.

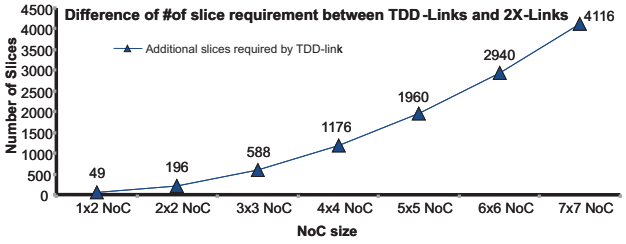


Fig. 4. Overhead of TDD-Links compared to the 2X-Links

We have implemented the *2X-Links* and the *TDD-Links* on a XILINX Virtex2 XC2V-6000 FPGA prototyping board for the analysis. The extra hardware that we need to implement either a single *2X-Link* or a single *TDD-Link* on top of our adaptive on-chip router as detailed in [8] is 74 slices. Each *TDD-Link* additionally requires one control unit to control the data transmission direction and the time slot allocation. The extra hardware needed to implement this control unit is another 49 slices. Fig. 4 shows the additional slice requirements for the *TDD-Links* compared to the *2X-Links*. With the increasing size of the NoC, a NoC with the *TDD-Links* requires more slices

compared to the NoC with the *2X-Links* e.g. in a 7×7 NoC, the *TDD-Links* require *4116* more slices than the *2X-Links*.

VI. Results and Case Study Analysis

In order to evaluate the proposed runtime configurable *2X-Links* different simulations are performed using the multimedia applications and the E3S benchmark [6] suit. In Fig. 5

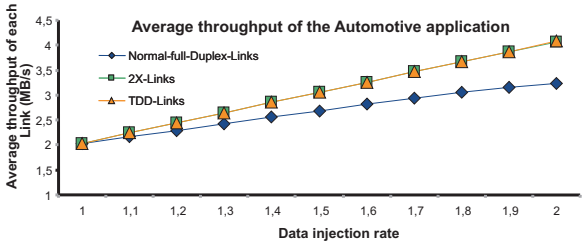


Fig. 5. Average throughput of Automotive application

the average throughput for different types of links for the **Automotive** application mapped onto a 5×5 NoC having the link capacity of 15 MB/s for the *Normal-Full-Duplex-Links* and for each half-duplex component of the *2X-Links*, and 30MB/s for the *TDD-Links* is shown. The average throughput is calculated as follows:

$$Th_{aver} = Th_{tot}/(2 \times ((M-1) \times N + (N-1) \times M)) \quad (2)$$

In Eq. 2, $(M \times N)$ is the dimension of the NoC and Th_{tot} the total throughput. We observe from the figure that the average throughput of the NoC with the *Normal-Full-Duplex-Links* is much lower than the average throughput of the NoC with the *2X-Links* or the *TDD-Links* at relatively high data injection rates. In scenarios where there is only asymmetric traffic on the NoC, there is no difference between the *2X-Links* and the *TDD-Links*. On the other hand, if bidirectional communication exists, the average throughput of the NoC with the *2X-Links* is slightly lower than the NoC with the *TDD-Links*.

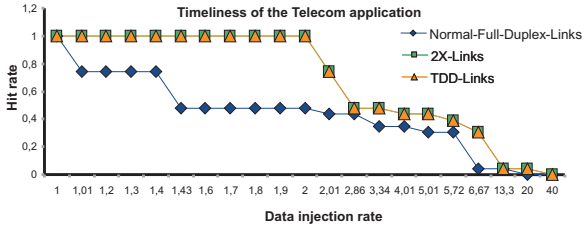


Fig. 6. Timeliness of the Telecom application

Fig. 6 shows the timeliness of the **Telecom** application (5×5 NoC having the link capacity of 20 MB/s for the *Normal-Full-Duplex-Links* and for each half of the *2X-Links* and 40MB/s for the *TDD-Links*) with different data injection rates. The timeliness is the ratio of the number of deadlines met to the total number of deadlines of an application (hit rate). Here, we see that most of the time the *Normal-Full-Duplex-Links* timeliness, is lower than the other two types.

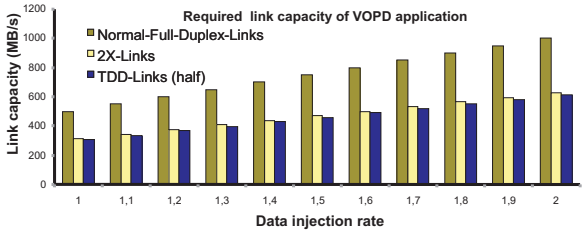


Fig. 7. Appropriate link capacity of the VOPD application

Fig. 7 depicts the required link capacity of the NoC with three different types of links when **VOPD** is running on the NoC (5×5 NoC having the link capacity of 500 MB/s for the *Normal-Full-Duplex-Links*, two times 500 MB/s for the *2X-Links*, and 1000MB/s for the *TDD-Links*). Obviously, the NoC with the *Normal-Full-Duplex-Links* requires much more link capacity than the *2X-Links* and the *TDD-Links* at design-time. Since the *2X-Links* can dynamically adapt their data-transmitting direction to meet the bandwidth requirement and the *TDD-Links* support bidirectional communication, both of them can guarantee the required bandwidth with lower link capacity compared to the *Normal-Full-Duplex-Links*.

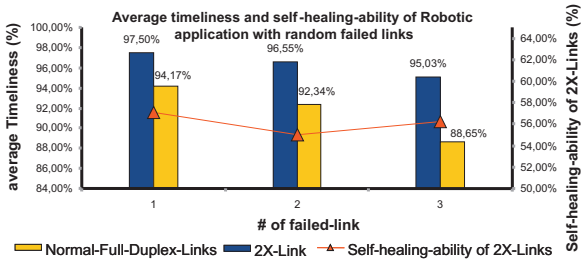


Fig. 8. Average timeliness and fault-tolerance ability of the Robot application

Fault-tolerance ability is important to the MPSoC design. In this simulation, we focus on the *average timeliness* (different from "Timeliness" shown in Fig. 6) of the NoC with different types of links when some links fail randomly. We enumerate all the possible combinations of the failed-links and calculate the average timeliness of them. We define the $T_{k,aver}$ as the average ratio of the number of deadlines met to the total number of deadlines of a given application when k links fail randomly. In Eq. 3 $\#Comb$ is the un-ordered collection of k failed links among all the links.

$$T_{k,aver} = \Big(\sum_{i=1}^{\#Comb} timeliness \Big) \; / \; (\#Comb) \quad (3)$$

The *Normal-Full-Duplex-Links* are static. If one link fails, the NoC will lose the ability to transmit data from the port which the link is tied to. Analog to the *Normal-Full-Duplex-Links*, the *TDD-Links* also do not have the ability of fault-tolerance. The *2X-Links* can adapt their data-transmitting direction to meet the performance-related guarantees. From the Fig. 8 we can see that the average timeliness of the NoC with the *2X-Links* is much higher than the NoC with the *Normal-Full-Duplex-Links* (for the **Robot** application which is mapped onto 3×3 NoC having the link capacity of 30.7 MB/s for the *Normal-Full-Duplex-Links*, two times 30.7 MB/s for the *2X-Links*, and 61.4 MB/s for the *TDD-Links*). The *fault-tolerance ability* (ft_k) is calculated according to the following equation (NFDL represent the *Normal-Full-Duplex-Link*):

$$ft_k = \frac{T_{k,aver,\text{2X-Links}} - T_{k,aver,\text{NFDL}}}{1 - T_{k,aver,\text{NFDL}}} \quad (4)$$

For a simple example, if the number of failed-links is 1, the fault-tolerance ability of the *2X-Links* is $(97.50\% - 94.17\%)/(1 - 94.17\%) = 57.12\%$. This result means that the NoC with the *2X-Links* can still work properly with the probability of 57.12%, when only one link fails randomly.

In the scope of this work load balancing means to spread communication over all the links to avoid producing communication bottlenecks and to relieve the link resource congestion. The advantage of applying load balancing techniques is that hardware resource utilization will be optimized and therefore, it may make the on-chip communication more capable to meet the requirements of unforeseen traffic circumstance. In our simulation, we measure the *bandwidth occupancy factor of each port* through $B\_occupancy\_factor = B_{utilized}/B$. The lower the factor is, the higher the potential available link

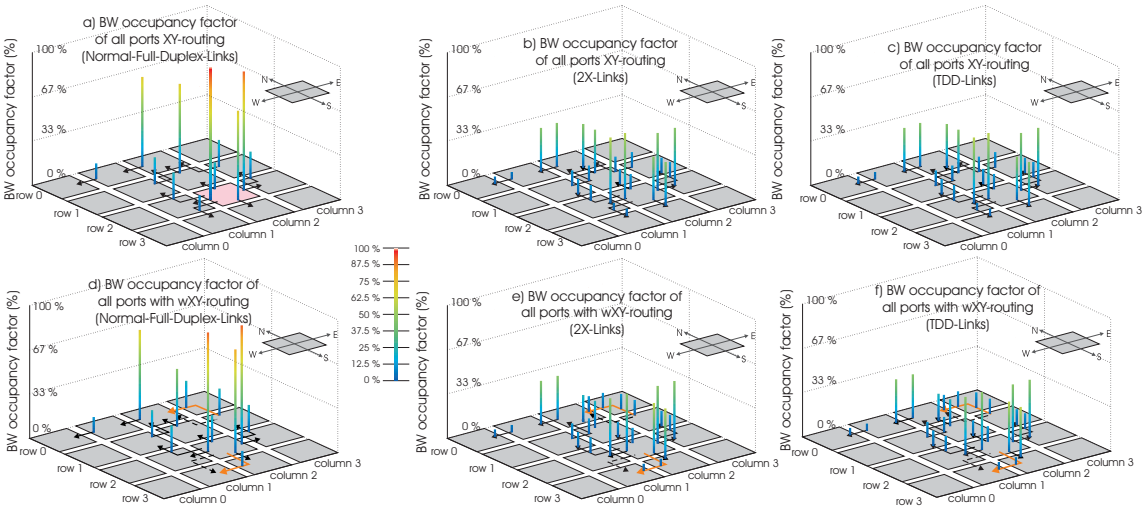


Fig. 9. Bandwidth occupancy factor of the consumer application

capacity is. Therefore, the NoC is more flexible to support unforeseen traffic (e.g. new connections etc).

Fig. 9 (**Consumer** application is mapped onto 4×4 NoC with the link capacity of 9 MB/s for the *Normal-Full-Duplex-Links*, two times 9 MB/s for the *2X-Links*, and 18 MB/s for the *TDD-Links*) represents the bandwidth occupancy factor of all ports with different types of links. Obviously, the bars of the *2X-Links* (Fig. 9 (b)) and the *TDD-Links* (Fig. 9 (c)) are much lower and the communication is spread more homogeneously on the NoC than with the *Normal-Full-Duplex-Links* (Fig. 9 (a)). This indicates that the NoCs with the *2X-Links* and the *TDD-Links* have more potentially available link capacity than the NoC with the *Normal-Full-Duplex-Links*.

In order to make the on-chip communication more adaptive, we additionally use the *wXY-routing* algorithm to determine the communication route instead of the static *XY-routing* proposed in [5]. Fig. 9 (d,e,f) depict the bandwidth occupancy factor using the *wXY-routing* algorithm. According to the new route, bandwidth requirements of several links are changed. Even after this adaptive routing algorithm, the result from the *2X-Links* and the *TDD-Links* are far more better than using the *Normal-Full-Duplex-Links*.

In our experiments we have observed that both the *2X-Links* and the *TDD-Links* provide better performance results than the *Normal-Full-Duplex-Links* considering the metrics: average throughput, timeliness, required link capacity, and the traffic load balancing. Our proposed *2X-Links* provide fault-tolerance ability unlike both the *Normal-Full-Duplex-Links* and the *TDD-Links*. Therefore, considering the area overhead and the lack of fault-tolerance ability of the *TDD-Links* we have integrated the *2X-Links* in our runtime adaptive on-chip communication architecture.

VII. Conclusion

We have introduced a novel runtime configurable link which can change its supported bandwidth on-demand at runtime for an adaptive on-chip communication architecture. We have achieved an increase in throughput of up to 36% (21.3% on average) using either the *2X-Links* or the *TDD-Links* compared to the *Normal-Full-Duplex-Links*. Our proposed *2X-Links* (the *TDD-Links* also provide similar result) can assure the performance-related guarantees with nearly 50% of the *Normal-Full-Duplex-Links* capacity. The *TDD-Links* have no *fault-tolerance ability* and utilize more hardware than *2X-Links* e.g. in a 7×7 NoC, *TDD-Links* require an additional 4116 slices more than the *2X-Links*. Our observation shows when some links fail, the NoC with the *2X-Links* can recover from faults (at most 3 faults) which would cause *Normal-Full-Duplex-Links* to fail with an average probability of 82.2% for the E3S benchmark, the VOPD, and the Robot applications (the *Normal-Full-Duplex-Links* and the *TDD-Links* have no capability to recover from the unforeseen link faults).

References

[1] T. Austin, V. Bertacco, S. Mahlke, and Y. Cao. "Reliable systems on unreliable fabrics". *IEEE Design & Test of Comp.*, 25(4):322–332, 2008.

[2] L. Benini and G. D. Micheli. "Networks on Chips: a new SoC paradigm". *Computer*, 35(1):70–78, 2002.

[3] S. Borkar. "Designing reliable systems from unreliable components: The challenges of transistor variability and degradation". *IEEE Micro*, 25(6):10–16, 2005.

[4] C. Busch, S. Surapaneni, and S. Tirthapura. "Analysis of link reversal routing algorithms for mobile ad hoc networks". *SPAA'03: Proceedings of the 15th annual ACM symposium on Parallel algorithms and architectures*, pages 210–219, 2003.

[5] W. J. Dally and B. Towles. "Route packets, not wires: on-chip interconnection networks". *DAC'01: Proc. of the 38th Conf. on Design Automation*, pages 684–689, 2001.

[6] E3S. http://ziyang.eecs.northwestern.edu/ dickrp/e3s/.

[7] R. Esmailzade, M. Nakagawa, and E. A. Sourour. "Time-division duplex CDMA communications". *IEEE Wireless Comm.*, 4(2):51–56, 1997.

[8] M. A. A. Faruque, T. Ebi, and J. Henkel. "Run-time adaptive on-chip communication scheme". *ICCAD'07: Proc. of the 2007 IEEE/ACM Int. Conf. on Computer-aided design*, pages 26–31, 2007.

[9] M. A. A. Faruque, R. Krist, and J. Henkel. "ADAM: run-time agent-based distributed application mapping for on-chip communication". *DAC'08: Proc. of the 45th Conf. on Design Auto.*, pages 760–765, 2008.

[10] I. T. R. for Semiconductors". http://www.itrs.net. 2007 Edition.

[11] P. Horn. "Autonomic computing: IBM's perspective on the state of information technology". *IBM Corporation*, 2001.

[12] Y. Hoskote, S. Vangal, A. Singh, N. Borkar, and S. Borkar. "A 5-GHz mesh interconnect for a teraflops processor". *Micro*, 27(5):51–61, 2007.

[13] S. Murali and G. D. Micheli. "Bandwidth-Constrained mapping of cores onto NoC architectures". *DATE'04: Proc. of the Design, Automation and Test in Europe Conf.*, pages 20896–20901, 2004.

[14] C. Nicopoulos, D. Park, J. Kim, N. Vijaykrishnan, M. S. Yousif, and C. R. Das. "ViChaR: a dynamic virtual channel regulator for network-on-chip routers". *MICRO'06: Proc. of the 39th Annual IEEE/ACM Int. Symposium on Microarchitecture*, pages 333–346, 2006.

[15] U. Y. Ogras and R. Marculescu. "Application-specific network-on-chip architecture customization via long-range link insertion". *ICCAD'05: Proc. of the 2005 IEEE/ACM Int. Conf. on Computer-aided design*, pages 246–253, 2005.

[16] J. D. Owens, W. J. Dally, R. Ho, D. N. J. Jayasimha, S. W. Keckler, and L.-S. Peh. Research challenges for on-chip interconnection networks. *IEEE Micro*, 27(5):96–108, 2007.

[17] E. Rijpkema, K. G. W. Goossens, A. Rădulescu, J. Dielissen, J. van Meerbergen, P. Wielage, and E. Waterlander. "Trade offs in the design of a router with both guaranteed and best-effort services for networks on chip". *DATE '03: Proc. of the Conf. on Design, Automation and Test in Europe*, pages 10350–10355, 2003.

[18] V. Soteriou and L.-S. Peh. "Design-Space exploration of power-aware on/off interconnection networks". *ICCD'04: Proc. of the IEEE Int. Conf. on Computer Design (ICCD'04)*, pages 510–517, 2004.

[19] S. Vassiliadis and I. Sourdis. "FLUX networks: Interconnects on demand". *Proc. of the Embedded Computer Systems: Architectures, Modeling and Simulation*, pages 160–167, 2006.

[20] D. Wentzlaff, P. Griffin, H. Hoffmann, L. Bao, B. Edwards, C. Ramey, M. Mattina, C.-C. Miao, and A. Agarwal. "On-Chip interconnection architecture of the tile processor". *IEEE Micro*, 27(5):15–31, 2007.