XNoC: A Non-intrusive TDM Circuit-Switched Network-on-Chip

Tuan D. A. Nguyen
Department of Electrical & Computer Engineering
National University of Singapore
4 Engineering Drive 3, Singapore 117583
Email: tuann@u.nus.edu

Akash Kumar
Center for Advancing Electronics Dresden
Technische Universität Dresden
Dresden 01187, Germany
Email: Akash.kumar@tu-dresden.de

***Abstract*—Network-on-Chip (NoC) is known as a scalable and high performance interconnect in Systems-on-Chip (SoCs) with multiple processing elements (PEs). Recently, the design paradigm of SoCs has shifted from static to dynamic runtime reconfigurable system. In these systems, the PEs can be loaded/unloaded on demand. Therefore, the NoC should be able to adapt as quickly as possible to the changes to maintain the performance of the systems. In this work, we present a non-intrusive runtime reconfigurable time-division-multiplexed circuit-switched NoC, *XNoC*, which offers the following benefits (1) it switches between different routes within a predictable latency that is strictly determined by the length of the route and the number of time slots; (2) the configuration process can be masked effectively by overlapping with communication and (3) the multi-cast service is supported with *aggregate feedback* from sink nodes. We propose an *XSwitch* which requires $3.5X$ less resource than the conventional switch with similar features. The overall resource cost of XNoC is also smaller than the most known NoC and the clock timing is up to 50% better. We also propose a novel *distributed control plane* to accelerate the reconfiguration process and to improve the scalability of NoC. The achieved reconfiguration speedup compared to the centralized control unit is up to $7.6X$ in certain conditions. On average, it takes only *74* clock cycles to activate a 12-hop connection.**

***Keywords*—NoC, TDM, multi-cast, distributed control, FPGA**

I. Introduction

Network-on-Chip (NoC) is a very heavily mined topic in the research community [1], [2]. However, with the advent of the partially reconfigurable (PR) FPGA-based System-on-Chip (SoC) consisting of runtime interchangeable and heterogeneous processing elements (PEs) [3], [4], many applications can be multiplexed on the same system. This becomes a challenge because the communication pattern as well as the behavior of the combinations of applications cannot be known entirely at design time. Therefore, the NoC must be designed efficiently to quickly adapt to the changing communication patterns as well as keeping up with the bandwidth requirements of the various applications at runtime.

Circuit-switched (CSw) NoC is known to perform better than packet switched (PktSw) NoC in real-time embedded systems [5], [6]. Some researchers proposed hybrid PktSw-CSw NoCs to inherit the advantages of both types such as [7], [8]. However, they do not provide much benefit in terms of cost-performance ratio and they break the composability of the system [5]. CSw NoCs are classified into Spatial Division Multiplexing (SDM) [9]–[11] or Time Division Multiplexing (TDM) [12]–[15] or both [16]. However, TDM is preferred to SDM because of lower data latency, less complex switch and cheaper network interfaces/switches (NIs/SWs) [17].

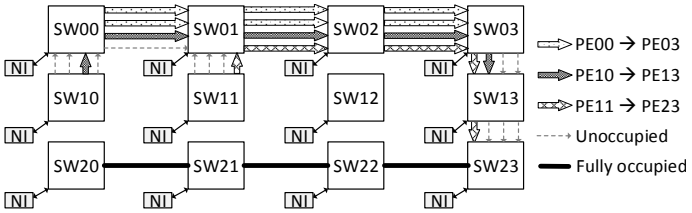

Fig. 1. Modifying path. PEs are attached to NIs and are numbered after their SW. PE00→PE03 is the path from PE00 to PE03. There are 4 time slots represented by 4 arrows. One more time slot is requested for PE00→PE03 to increase the bandwidth.

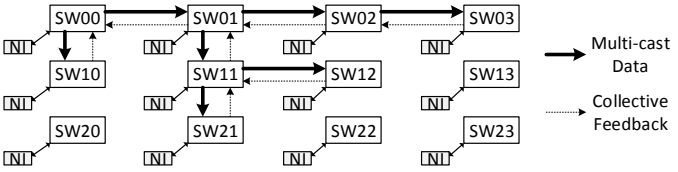

Fig. 2. Multi-cast. PE00 sends data to PE01, PE02, PE03, PE10, PE11, PE12 and PE21. SW01 collects feedbacks from PE01, SW02 and SW11 before sending to SW00 then PE00.

Therefore, we focus on the TDM CSw NoC. We address the following research gaps which we believe are very important for the applications nowadays. **First**, the path (de)-activation happens very frequently in the PR SoCs and it is expected to increase because of the dynamic nature of the systems (reconfiguring PEs at runtime). The latency of this process should be kept at minimum and predictable. **Second**, the reconfiguration process in the aforementioned NoCs is based on the assumption that it does not overlap with the data transfer. This is not the case in a scenario when the user changes from a low quality video to a higher one. The bandwidth of the existing path needs to be adjusted as shown in Fig. 1. This example is elaborated in Section III-D. **Third**, although it is trivial to provide the multi-cast service in CSw NoC [14], the feedback from the receiving nodes are ignored. To the best of our knowledge, there is no mechanism so far in CSw NoC to aggregate them back to the source node as illustrated in Fig. 2. **Lastly**, despite the benefits of centralized control unit over the decentralized one as discussed in Section II-B, it suffers from the scalability issues in the systems with up to thousand of PEs connected by one NoC.

Contributions. We propose a runtime reconfigurable TDM CSw NoC, *XNoC*, equipped with the XSwitch to tackle all of the above issues. Our contributions are following.

- *Low-latency and predictable path (de)-activation*. (De)-activating a single-slot path takes at most $N + D + 2$ clock cycles (D is the length of that path, N is the number of time slots of the NoC).
- *Non-intrusive reconfiguration process*. It does not interfere with the current operation of the switch *at all*.
- *Multi-cast service with aggregate feedback*. Any bit-wise combinational logic can be done at any SW *without extra latency and dedicated logic*. The scheduling of the feedback from the receiving nodes is directly inferred from the data path.
- *Distributed control plane*. All the benefits of the centralized control approach are retained while the reconfiguration overhead is greatly reduced. The long control links spanning across all SWs in the network are no longer needed.

We adopt a centralized management approach with distributed control plane to configure the NoC. The mesh topology is used throughout the paper but it is not restricted to that. The implementation results show that the XSwitch takes much less resources, $3.5X$ lower, than the conventional SW which offers the similar features. The whole XNoC also scales linearly with the number of time slots and the number of nodes. The minimum clock period obtained after placement and routing is 50% better than the well-known previous work; the resource cost is also lower. Moreover, the distributed control plane performs much faster than the centralized control unit. In some conditions, the speedup is up to $7.6X$. For a connection of which length is 12 and requires 8 time slots, it takes an average of 74 clock cycles to activate. The resource overhead of the distributed control plane is also negligible.

The remainder of this paper is organized as follows. The related work and motivation are discussed in Section II. The proposed XNoC and distributed control plane are presented in Section III and IV respectively. The experimental results are provided in Section V. Finally, the conclusion and future works are presented in Section VI.

II. Related Work and Motivation

A. Related Work

One of the highly regarded TDM CSw NoCs in literature is Æthereal [12]. It offers both PktSw for best-effort (BE) services and CSw for guaranteed services (GS). In a later article [5], the authors stated that the extra resources required by BE were not worth the trade-off of the overall performance of the NoC. Therefore, the subsequent NoCs inherited from Æthereal completely removed the BE service such as aelite [13] and dAElite [14]. aelite is the lite version of Æthereal without BE service. In dAElite, the configuration process is done without intervening with the current applications as long as the existing paths are not affected. dAElite utilizes end-to-end credit-based flow control for one-to-one transactions, but none for one-to-many.

In [16], the SDM-TDM CSw NoC was suggested to take advantage of SDM and TDM techniques to increase the number of possible paths between PEs. The path setup/tear-down operations are decentralized by integrating a small Control Unit in every router to process the request packets from PEs using the XY routing algorithm. However, there is no flow control in the proposed NoC.

Likewise, Liu et al. [15] proposed a distributed dynamic connection setup in their CSw TDM NoC using the parallel probing path searching algorithm. It was proved to be more effective than the XY routing. However, they do not support multi-time-slot path. The flow control for one-to-one connection is end-to-end ready-to-receive signal. The one-to-many and many-to-one connections are not supported.

B. Motivation

The disadvantages of using decentralized compared to centralized control unit in [15], [16] are (1) it cannot establish a multi-cast domain because the sender needs to wait for the acknowledged signals from all receivers to make sure that the paths are setup; the control units would become much more complex to handle the responses from the neighbors; (2) the routing algorithm cannot be changed at runtime unless the routers are partially reconfigurable; (3) it does not have the global view of the current state of the network to adapt to changes at runtime; (4) it only allocates one slot for one path at a time; there is no guarantee that the data in multi-time-slot path arrive at the destination in order.

The most concerned problem of CSw NoC with centralized control unit is the path setup and tear-down overhead. They can be over 1000 clock cycles in [12] or about 100 cycles for one short path in [14]. Having a predictable path setup/tear-down latency as low as possible is the first step in making it feasible to interleave multiple paths that share the same links and time slots. It is similar to the *context switching* in multi-threaded software environment. Another drawback of conventional centralized control unit is scalability. For large NoCs with hundreds or even thousand of PEs, the reconfiguration process becomes a bottleneck. Moreover, having the control links span the whole network severely affects the timing of the design. One may insert the buffers along the links but it again increases the reconfiguration latency.

CSw NoC also suffers from the obstruction of the reconfiguration process to the data communication. All of the existing non-static TDM CSw NoCs mentioned above suffer from the lengthy disruption of current traffic when they have to modify the existing flows as in the example shown in Fig. 1. The main reason is that they have to wait for the data in a path to clear up before tearing it down or adding more time slots to it. For the TDM CSw NoCs with decentralized control unit, it is not possible to do so because they do not have the global knowledge of the system.

Lastly, the multi-cast (one-to-many) with reduction operation (AND, OR, XOR, etc.) (many-to-one) as shown in Fig. 2 is not supported by the NoCs above. The multi-cast operation is provided implicitly by the CSw NoC [14]. In case of PktSw NoCs, the application running on the source node has to send the data to multiple destinations if the NoC does not support it natively. For the reduction operation, the SWs must perform logical operations on the incoming feedback signals before forwarding to the sender. Currently, there is no CSw NoC that supports this operation. The PktSw NoC proposed by [18] has addressed this issue. Nevertheless, it requires extra logic for *each* of the logical operations.

III. XNoC

A. Overview

Fig. 3 illustrates the general architecture of XNoC. We start with the centralized approach to explain our techniques and

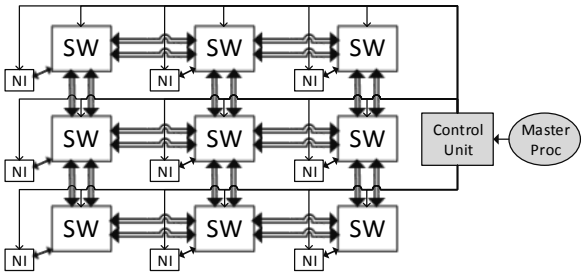

Fig. 3. The 3x3 XNoC. The SWs and NIs are in Data Plane. The PEs (not shown for simplicity) are attached to the NIs. The Control Unit receives instructions from the Master Processor (Master Proc) to control the Data Plane.

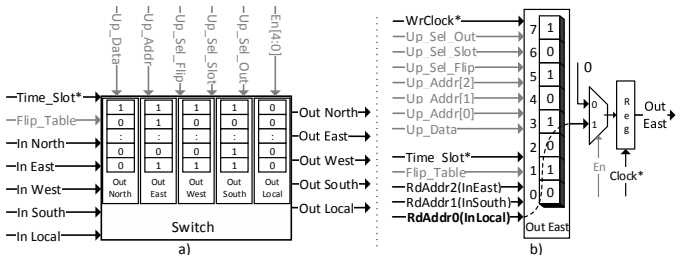

Fig. 4. *(a)* The 5-input XSwitch. It consists of a set of 5-input memory elements (MEs). The gray signals are from the Control Unit. The ones with asterisk are global. *(b)* Example of how a multiplexer is realized by changing the content of the ME. The ME is simplified to 3-input for illustration.

design considerations. After that, the distributed control plane is presented in Section IV which can preserve all advantages of our centralized control unit. The *Control Unit* takes and processes instructions sent by the *Master Processor* (Master Proc) to control all of the operational aspects of the XNoC. These instructions are: configuring the routing tables inside the SWs, enabling/disabling their outputs, suspending/resuming the NIs at a specific time slot or all time slots. Except the routing tables which are stored inside the SWs, all other configurations reside in this plane. The NIs, SWs and all the connections between them make up the *Data Plane*.

In our current implementation, the time slot counter and clocks are distributed globally to all components. The Control Unit manages the NIs and SWs via direct control signals. By this way, it can manipulate NIs and SWs at the precise timing when the paths are (de-)activated or at whatever time slot the Master Proc wants. The path (de-)activation processes are presented in Section III-D. The signals used to update the content of the routing tables are broadcasted to all SWs. Each of them only accepts the updates when the value of the address signals matches its pre-configured value.

In general, XNoC works similar to other TDM CSw NoCs with a central control unit [14]. The novelties of XNoC lie in the design of the switch, the connection (de-)activation mechanisms, the schedule of the feedback signal and the many-to-one operation. These are discussed in the subsequent sections. During the discussion, N denotes the ***number of time slots***, D represents the ***length of a path***.

B. XSwitch

Unlike the conventional 5-port switch which is implemented by multiplexers with configurable selectors, our XSwitch is designed from a different point of view. Fig. 4 depicts the architecture of our XSwitch. We make use of the intrinsic idea of FPGA in implementing digital circuits using arrays of small memory elements (MEs) (lookup-table in Xilinx FPGA). As shown in Fig. 4a, there is one set of N 5-input MEs per output direction per output bit. The inputs are from the North, East, West, South and Local ports. They are connected to the *asynchronous address read port* of the ME. The output port, connected to the read data output, is therefore the result of any 5-input combinational circuit. Multiplexer is just one of them. Fig. 4b shows how a multiplexer is implemented by changing the content of the ME.

The MEs are exposed to the software level to offer full controls over the functionality of the SWs at runtime. All signals in gray are originated from the Control Unit. The update signals (*Up_Data, Up_Addr*, etc.) are connected to the synchronous write port of the MEs to update their content. The signal *Up_Sel_Out* selects which output port to configure. There are N *MEs per output port bit*. Each corresponds to one time slot and is selected based on the global *Time_Slot* signal during the data communication.

In XNoC, the configuration process does not interfere with the existing data flow even when it is done at the same time slot on the same output of the switch. It is possible thanks to the *ping-pong* technique in which the number of MEs of the switch is doubled resulted in two separate sets of MEs. At one time, the configuration is taken place at one set which is not in use (selected by the signal *Up_Sel_Flip*). When the process is done, the SW can switch to the newly configured set in one clock cycle. This *flipping* process is managed by the Control Unit via the signal *Flip_Table*. Even though this method is not novel, XNoC is the first that applies it to the context of NoC. The incurred overhead is discussed in Section V-A.

C. Flow Control - Feedback

XNoC employs the end-to-end flow control with ready-to-receive feedback. The feedback from the sink node to the source node must arrive at the same time slot when the data is sent. This requirement ensures that no extra buffer at the NIs or other credit-based flow control is required to compensate for the design-time-unpredictable feedback arrival time. As a result, less memory and link resources are needed. Liu et al. [15] claim that their double orientation time wheel can assure this requirement. It is not entirely true. In their approach, the counter used for the feedback signal runs in the reversed direction of the main counter of data, both starting at 0. The problem is that the feedback will certainly arrive at the source node at the time slot when the data is sent, but with respect to (w.r.t) the reversed counter, not the main counter. For example, say if a path of length 6 starts at time 1, the time slot sequence of data (w.r.t the main counter) is $1\rightarrow2\rightarrow3\rightarrow0\rightarrow1\rightarrow2$. The time slot sequences of feedback w.r.t the reversed and main counter are $2\rightarrow1\rightarrow0\rightarrow3\rightarrow2\rightarrow1$ and $2\rightarrow3\rightarrow0\rightarrow1\rightarrow2\rightarrow3$ respectively. The feedback arrives at the source node when the main counter is 3.

In XNoC, only one counter is needed; the schedule of the feedback is directly inferred from the data as shown below. In the equations, $start_{dat}$/$start_{fb}$ is the slot when the data/feedback is injected into the network. end_{dat}/end_{fb} is the slot when the data/feedback reaches the destination.

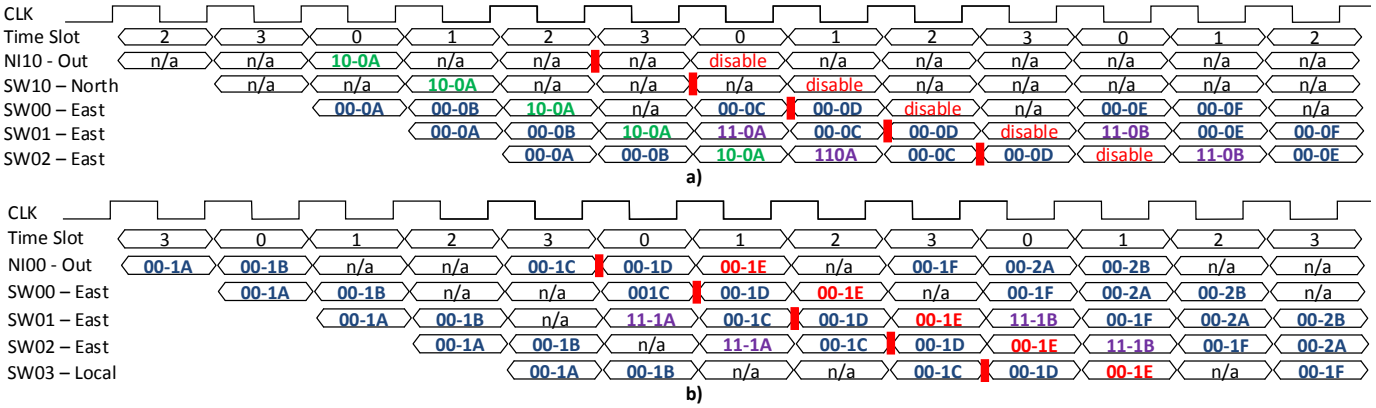

Fig. 5. *(a)* The deactivation process of PE10→PE13 and *(b)* the path activation process to add one more slot for PE00→PE03 in Fig. 1. The data flows belong to PE00→PE03, PE10→PE13 and PE11→PE23 are represented in blue, green and purple texts respectively. The red bars indicate the times when the Control Unit starts (de-)activating the paths. The texts in red denote the changes to the paths.

$$end_{dat} \equiv start_{dat} + D \ (mod\ N) \quad (1)$$
$$end_{fb} \equiv start_{fb} + D \ (mod\ N) \quad (2)$$
$$end_{fb} = start_{dat} \quad (3)$$
$$(2,3) \Rightarrow start_{fb} \equiv start_{dat} - D \ (mod\ N) \quad (4)$$
$$(1,4) \Rightarrow start_{fb} + end_{dat} \equiv 2 * start_{dat} \ (mod\ N) \quad (5)$$

(5) is **true for all D**:

$$\Rightarrow \mathbf{slot_{fb} + slot_{dat} \equiv 2 * start_{dat} \ (mod\ N)} \blacksquare \quad (6)$$

Equation 6 indicates that for a particular data flow that starts at $start_{dat}$, the time slot of the feedback signal, $slot_{fb}$ at any intermediate nodes (SWs/NIs) can be calculated based only on the $start_{dat}$ and the corresponding time slot of data at that node, $slot_{dat}$. The feedback signal always arrives at the $start_{dat}$ time slot at the source node.

There are cases that the feedback signals of two different paths (start at $start1_{dat}$ and $start2_{dat}$) that share the same link at two different time slots ($slot1_{dat}$ and $slot2_{dat}$) may overlap with each other. Therefore, the data path scheduling algorithm needs to perform one simple check as shown in Equation 7 to make sure that this issue does not happen.

$$slot1_{dat} - slot2_{dat} \not\equiv 2 * (start1_{dat} - start2_{dat}) \ (mod\ N) \quad (7)$$

Proof: Equations 8 and 9 for two paths are derived from Equation 6. The above issue only occurs when $slot1_{fb} = slot2_{fb}$. Subtracting Equation 9 from Equation 8, the condition shown in Equation 7 is obtained. ■

$$slot1_{fb} + slot1_{dat} \equiv 2 * start1_{dat} \ (mod\ N) \quad (8)$$
$$slot2_{fb} + slot2_{dat} \equiv 2 * start2_{dat} \ (mod\ N) \quad (9)$$

When the overlapping problem occurs between two paths, if it is possible to shift the start time slot of any one path in the *odd/even* number of slots, we only have to shift at most *two/one* times to resolve all the conflicts. Otherwise, the path finding algorithm needs to find another route. The proof is not provided because of the space constraints. The effect of Equation 7 on the number of paths that can be allocated is explored in Section V-C.

D. (De)-Activating Connection

In this section, the processes of (de-)activating the connection between PEs are presented for the example in Fig. 1. First, it is assumed that all necessary changes are loaded into the not-in-used ping-pong set of MEs of the SWs. After receiving the commands from the Master Proc to deactivate PE10→PE13 starting at time slot 0 at NI10, the Control Unit waits until the global *Time_Slot* is 3 (1 slot ahead of the designated slot). Right after that, it starts the deactivating process along the path as shown in Fig. 5a. The waveforms for SW03 and SW13 are not shown for simplicity. The red vertical bars indicate the time at which the Control Unit *disables* the output of the NIs/SWs without having to wait for the data to clear off the NoC. PE00→PE03 and PE11→PE23 are kept intact.

After that, the Master Proc decides whether to activate the new path for PE10→PE13 or the additional time slot for PE00→PE03. Suppose it does the latter first. The activating process is demonstrated in Fig. 5b. Similar to the deactivating process, the Control Unit only starts when the global *Time_Slot* is one slot ahead of the desired slot, in this case, 0. What it does are to (1) *flip* the signal *Flip_Table* to switch to the new configuration and (2) activate the NIs/SWs. As shown, the new slot is assigned to PE00→PE03 while the data is still flowing between the PEs/SWs and the order of the data is not affected. Likewise, the new path for PE10→PE13 is activated.

For every time slot of every path, the Control Unit needs 2 clock cycles to decode the first word of the (de)-activating instructions sent from the Master Proc. It takes 6 clock cycles to deactivate PE10→PE13 which is its length, D. The Control Unit waits for at most N clock cycles to begin execution. In total, the path deactivating process takes $N + D + 2$ clock cycles. For the path activation process, on the other hand, the NI can transfer the data as soon as the Control Unit starts the process. Hence the delay is $N + 2$. The feedback path is activated together with the data path.

Generally, if the number of time slots that is (de)-activated for a path is T, the number of clock cycles required to *fully* deactivate and activate are $T * (N + D + 2)$ and $T * (N + D + 2) - D$ respectively. However, it still takes only $N + 2$ clock cycles for the NI to inject data into the network. This mechanism is better than the previous works because the NI *is not suspended* until the whole path is configured.

E. Multi-cast operation

Multi-cast is one-to-many operation in which the same data is sent simultaneously from one node to a set of nodes. The

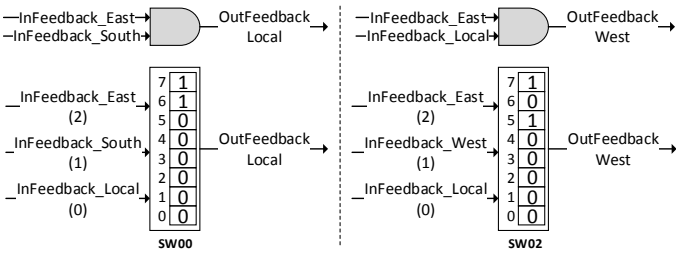

Fig. 6. The configuration for the feedback signals at SW00 and SW02 for the example in Fig. 2.

aggregation of the feedback signals from the receiving nodes in the multi-cast tree is the example of many-to-one. Setting up the multi-cast tree for TDM CSw NoC is straight forward as described in [14]. However, the aggregate feedback is not supported in that work and any other CSw NoC.

In XNoC, since the XSwitch can implement any 5-input combinational circuit for a particular output, aggregating feedback signal becomes trivially doable. In Fig. 2, SW00 performs the bit-wise *AND* operation to the feedbacks from SW10 and SW01 before forwarding them to the NI00. This can be denoted as follows: $Out\_Local = In\_South \wedge In\_East$. Similarly, at SW02: $Out\_West = In\_Local \wedge In\_East$. Fig. 6 shows the corresponding data for the MEs at SW00 and SW02. In other words, the MEs are the truth tables of any one-output combinational circuit of five inputs. Besides, the aggregation only performs as intended when the feedbacks from the receiving nodes arrive at the SWs at the same time slot. *This condition is easily satisfied if the schedule of the feedbacks follows Equation 6*. The time taken by the aggregated feedback to propagate to the source node equals to the longest path from the source node to the sink nodes.

IV. Distributed Control Plane

A. Overview

As discussed in Section II-B, the centralized control unit cannot scale well with large networks. In this section, a *distributed control plane* is presented to handle such networks. Fig. 7 shows our control plane with 4 *Sub-Controllers* (SubCtrls). Each SubCtrl manages one sub-region of the network. The *Control Agent* forwards the instructions received from the *Master Proc* to the respective SubCtrl to reconfigure the SWs or (de)-activate the paths. The Control Agent is also responsible for synchronizing the SubCtrls in some specific operations. In our architectural model, we adopt the centralized management system [3], [4]. The Master Proc has the knowledge of the current status of the system as well as the privilege of managing the applications, loading the partial bitstreams to the PEs and reconfiguring the network. The Master Proc is not necessarily a single core processor nor a single processor; it can be a multi-core processor or a set of processors in the runtime management domain. Therefore, the bottleneck of generating the configurations for the large system at runtime is not a big concern.

B. (De)-activate Connection — Inter-SubCtrl Synchronization

In the proposed distributed control plane, the instructions used to configure the routing tables inside the SWs, enabling/disabling their outputs, suspending/resuming the NIs at

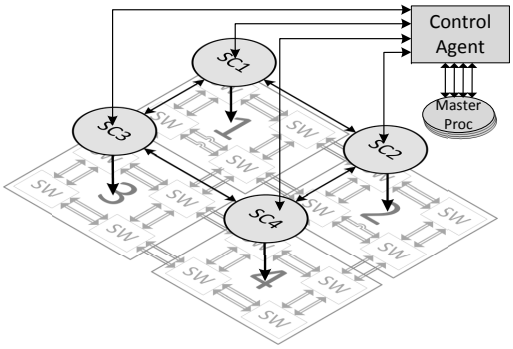

Fig. 7. Example of *distributed control plane* in a 4x4 NoC with 4 *Sub-Controllers* and 1 Control Agent. Each Sub-Controller controls 4 SWs. The Master Proc sends instructions to the Sub-Controllers via the Control Agent.

a specific time slot are called *single configuration*. These instructions are executed independently at their destined SubCtrl. In the connection (de)-activation mechanisms in Section III-D, the SubCtrls must communicate with each other to execute the instructions at the correct time slot. These connection (de)-activation instructions are called *chain configuration* because they have to be executed in sequence. In Fig. 7, it can be seen that the SubCtrls are connected together by a small network. This network is used to synchronize the SubCtrls.

The detailed micro-architecture of the synchronizer inside each SubCtrl is shown in Fig. 8. A connection between source SW and destination SW can span multiple SubCtrls. The SubCtrl where the source SW and destination SW resides are called *head-SubCtrl* and *tail-SubCtrl* respectively. Any SubCtrl in the middle of the path is called *mid-SubCtrl*.

The basic idea of the synchronization is that the head-SubCtrl must wait for all involved SubCtrls to be ready before (de)-activating the connection. The *Ready* signal propagates from the tail-SubCtrl toward the head-SubCtrl. After that, the head-SubCtrl starts (de)-activating the SWs in its region in the same way as presented Section III-D. When the head-SubCtrl reaches the last SW, it triggers the *Handover* signal to the succeeding SubCtrl to continue the operation. At this time, the head-SubCtrl can start processing the next instructions from its FIFO. The succeeding SubCtrl de-asserts the *Ready* signal as soon as it detects the *Handover*. The same process is repeated in the subsequent mid-SubCtrls until the tail-SubCtrl. The directions at which the SubCtrls should read the *Ready*

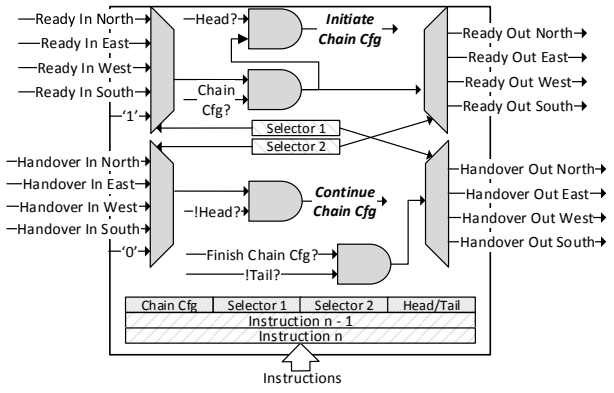

Fig. 8. The synchronizer inside each SubCtrl. It is used to synchronize the connection (de)-activating processes.

signal from and send the *Handover* signal to are indicated by the Master Proc in two fields *Selector 1* and *2* in the header of the *chain configuration*.

There might be a concern about deadlock when the SubCtrls are waiting for the *Ready* signal in a cycle. It occurs when the path from the source to the destination contains a cycle in terms of SubCtrl, eg., SubCtrl1→SubCtrl2→SubCtrl3→SubCtrl4→SubCtrl1. This can be avoided by the deadlock avoidance routing algorithms.

The state machine inside each SubCtrl is also designed to avoid a critical issue when a new chain configuration arrives at a SubCtrl which has just triggered the *Handover* signal for the previous chain configuration. That SubCtrl may be head and the subsequent SubCtrl in the new chain configuration may also be the same as the previous one. If the *Ready* signal from the succeeding SubCtrl has not been de-asserted yet, that head-SubCtrl will accidentally start executing the instructions instead of waiting for other SubCtrls to finish the previous one. It is because of the latency from the *de-multiplexer* and *multiplexer*. The state machine takes this latency into account, which is 5 clock cycles, before parsing the next instruction.

C. Activate Connection — Out-of-Order Execution

The synchronization method shown in Section IV-B retains the predictability as well as the benefits of the (de)-activating method presented in Section III-D. Some speedup can still be achieved. However, it cannot exploit the full potential of the distributed control plane, especially when multiple data channels are used to feed instructions directly to SubCtrls as discussed in Section IV-D. Therefore, we propose an out-of-order execution technique to activate connections. The out-of-order connection deactivation is not supported in this work because the activation process is more crucial to the performance of the system. We decide to keep the synchronization method for it and leave this as a future work.

To achieve out-of-order connection activation, each involved SubCtrl must be able to activate its SWs independently and the data must not be injected into the network during this time. However, it is impossible for the Master Proc to know exactly when the operation is finished to issue the enable instructions to the source NI. Thus, we design a synchronizer inside the Control Agent (illustrated in Fig. 9) to handle this issue at the hardware level. Each SubCtrl can independently activate the SWs that belong to its region without having to wait for the others. The NI is not enabled at the start of the chain configuration as before. It will be activated at the head-SubCtrl after all other SubCtrls finish their operations. With this method, head-SubCtrl is the only SubCtrl that is blocked.

As can be seen in Fig. 9, there are two types of request coming from SubCtrls: *update Header ID* and *check Header ID*. These two requests are executed independently by the Control Agent. The SubCtrls are selected to be served by a Round Robin selector. Inside each chain configuration header, there is a field called *Header ID*. It is to distinguish multiple chain configurations that are being executed. For all SubCtrls in a chain configuration, they are assigned the same Header ID. The status of the chain configuration is stored inside the true dual-port BRAM indexed by its Header ID.

When any of the mid-and-tail-SubCtrls has already reconfigured its SWs, it issues the *update Header ID* to increment the status value of the Header ID. That SubCtrl waits until its

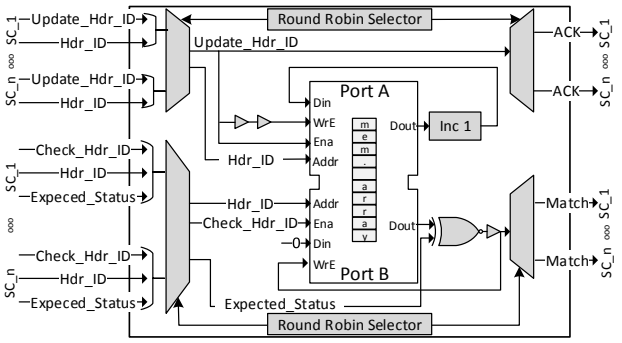

Fig. 9. The synchronizer at the Control Agent to support out-of-order connection activation.

turn to get the request served, then it is free to process the next instructions. The chain configuration header for the head-SubCtrl contains an *Expected Status* field which is the number of mid-and-tail-SubCtrls in the chain configuration. The head-SubCtrl keeps checking the current status of the Header ID until it matches the *Expected Status*. Only after that, the status value in BRAM is cleared, the NI is activated and finally, the data is injected into the network.

D. Discussion about the Speedup

It is worth noting that the reconfiguration speedup gained from the distributed control plane depends on how the instructions are fed to the SubCtrls. If the SubCtrls are not getting instructions fast enough, the speedup will be very modest. This problem happens when the Master Proc only uses one data channel to send instructions to the Control Agent. The Control Agent, in its turn, decodes the instructions, then forwards them to the corresponding SubCtrl instruction FIFO. In this case, the Control Agent becomes the bottleneck of the whole reconfiguration process.

Additionally, the size of the instruction FIFOs in SubCtrls also affects the speedup. Consider the case when one of the FIFOs gets full while its SubCtrl is synchronizing with other SubCtrls. The Control Agent must wait until there is available space in that FIFO to push the instruction in before proceeding to the next one. This is similar to the head-of-line blocking phenomenon. One solution to this problem is to use virtual output queues at output ports which is already implemented implicitly (the instruction FIFOs in SubCtrls). An efficient run-time scheduling algorithm for the instructions may mitigate the problem, but it would be quite complicated. Another solution is to increase the buffer size. The experiments carried out in Section V-D investigate this solution. Nevertheless, this requires large FPGA on-chip memory resources.

Therefore, instead of utilizing only one data channel to send instructions to the SubCtrls, we use multiple data channels to push instructions directly to the SubCtrls. In our implementation, these channels are AXI-Stream. This method can be realized in Master Proc with multiple cores and multi-threaded runtime manager. Each thread is responsible for sending the instructions to one SubCtrl. This method may bear some resemblance to the *Wormhole run-time reconfiguration* [19] with the idea of distributed streams of reconfiguration. However, in [19], each independent reconfiguration stream is responsible for configuring a set of continuous functional units. There is no need to synchronize and to make sure that the reconfiguration process across different streams happen at

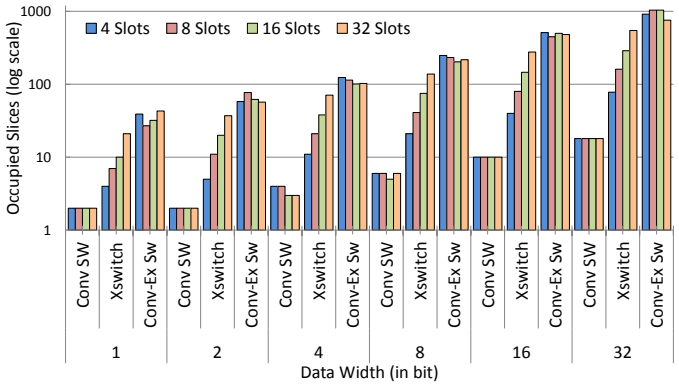

Fig. 10. The resource costs of three SWs with varied numbers of time slots and data widths. The number of slices is obtained after the Mapping phase.

the correct clock cycles. Therefore, that approach cannot be applied to the case of XNoC where the most important thing is to guarantee the timing of the reconfiguration process.

The speedups achieved in each of the above two approaches are analyzed and compared in Section V-D.

V. Experiments

In this section, we present the resource costs of XNoC. The impact on the path scheduling imposed by the Equation 7 is discussed. All implementation results are obtained from Xilinx ISE 14.7 with default settings and the target device is Xilinx Virtex-6 XC6VLX240T (unless stated otherwise).

A. Resource Cost - XSwitch

XSwitch is a better alternative to the conventional 5-input multiplexer-based SWs (Conv SW). Any 5-input combinational circuit can be implemented in the XSwitch to perform the reduction operations without any extra resource. Besides, every output bit of the XSwitch can be configured independently. For example, the first bit of the North output port selects the input from the West port while the second bit selects the input from the South port. Therefore, to have a fair comparison of the XSwitch with the Conv SW, we implement the *Conv-Ex SW*. It is the extension of the Conv SWs with the same features as the XSwitch. Fig. 10 shows the resource consumptions of these SWs. As expected, the XSwitch takes more resources than the *Conv SW*. Nevertheless, on the average, the resource requirement of the XSwitch is *3.5 times lower* than the Conv-Ex SW. However, we only implement 5 configurable combinational circuits for the Conv-Ex SW which are AND, NAND, OR, NOR and XOR. The cost of the XSwitch is expected to be much better than the Conv-Ex SW when more combinational circuits are integrated into the Conv-Ex SW.

It is worth noting that in the Conv SWs, the structure of the multiplexer is fixed, the selector is configurable by using a 3-bit wide memory. This memory consumes only 2 slices for up to 64-word depth. Thus, the resource cost of the Conv SWs varies insignificantly when the ping-pong technique is utilized (Section III-B). In contrast, in the XSwitch, the multiplexer is just one of the combinational functions realized by the MEs. If the ping-pong technique is used, the resources will be doubled. Therefore, in this current implementation, we decide to utilize the hybrid approach to inherit all the benefits of both SWs, i.e, the Conv SW is used for the data path (to save the resource) and the XSwitch is used for the feedback path (to support the reduction operations).

TABLE I. XNoC Resource Cost

XNoC	Other works
2x2, 8-Slot, 39-bit link, area optimization	
3073 (488) Slices, 4057ps	[14] 3098 Slices, 8190ps [13] 5500 Slices, 8379ps
2x2, 8-Slot, 39-bit link, timing optimization	
3639 (581) Slices, 3985ps	[14] 3655 Slices, 4968ps [13] 5393 Slices, 4986ps
2x2, 32-Slot, 39-bit link, area optimization	
3470 (697) Slices, 4753ps	[14] 3483 Slices, 8840ps
2x2, 32-Slot, 39-bit link, timing optimization	
4313 (777) Slices, 3988ps	[14] 4425 Slices, 4850ps
4x4, 16-Slot, 54-bit link	***4x4, 3 18-bit lanes, 5-slot***
68454 ALUTs, 292MHz	[16] 70815 ALUTs, 90MHz
4x4, 16-Slot, 90-bit link	***4x4, 5 18-bit lanes, 3-slot***
71698 ALUTs, 296MHz	[16] 90141 ALUTs, 102MHz

B. Resource Cost - XNoC

Since different NoCs have different features and architectures, it is difficult to compare them directly. Additionally, some works only report the ASIC implementation results. Therefore, we only compare our work against the ones which are similar to ours and are implemented on FPGA. Table I shows the place and route results of XNoC in comparison with dAElite [14] and aelite [13] with the same configurations. The synthesis results of XNoC on the Altera Stratix III EP3SL340F (using Quartus II 11.0sp1) are also presented to compare with [16]. The major difference between our current implementation and the closest NoC, dAElite, is the NI. Currently, we employ the AXI-Stream interface for NI. The mapping from different addresses to specific time slots is not yet supported. However, we did project the expected resources of XNoC with full-featured NIs based on the resource break-down information provided in [14]. In Table I, the numbers in parentheses are the slice resources of XNoC without the full-featured NIs.

As seen, XNoC requires less resources than dAElite while offering more features. The minimum clock period of XNoC is also better, which is up to 50% better than dAElite. In TDM CSw NoCs, the throughput of a connection is always guaranteed once it is set up. It only depends on the number of assigned time slots and the clock speed of the NoC. Therefore, 50% improvement in clock speed can be translated to 50% faster throughput. It is worth noting that XNoC's timing performance should not be affected by the full-featured NIs once implemented. It is because there are always buffers between NIs and SWes inside NoC to store the packets. These buffers set apart the path from NI to SW and SW to NoC. As a result, the NI would be able to operate at a frequency different from the NoC.

Additionally, the aeltie consumes much more resources than XNoC. The reason is that it requires a translator at each NI to decode/encode the configuration data on the regular network channels to update its path tables. In XNoC, the paths are embedded inside the MEs of the XSwitch and they are updated directly by the Control Unit or SubCtrls.

The resource cost of XNoC is slightly lower than [16] in the 3-lane-5-slot configuration and much lower in the 5-lane-3-slot configuration. In both cases, the reported *Fmax* for XNoC

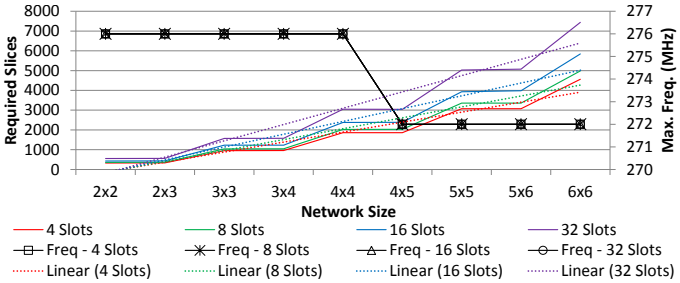

Fig. 11. The synthesis results of XNoC with different configurations.

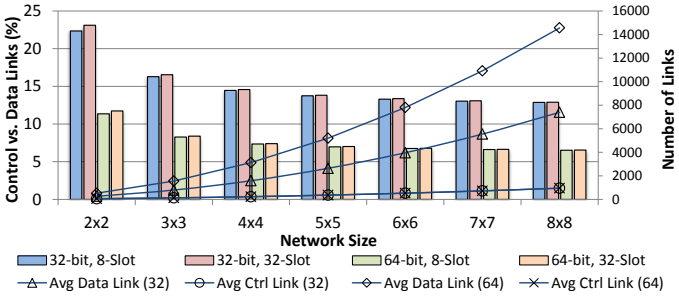

Fig. 12. The ratio between the number of control and data links in various network sizes, data widths and numbers of time slots. The average number of data links and control links are also reported.

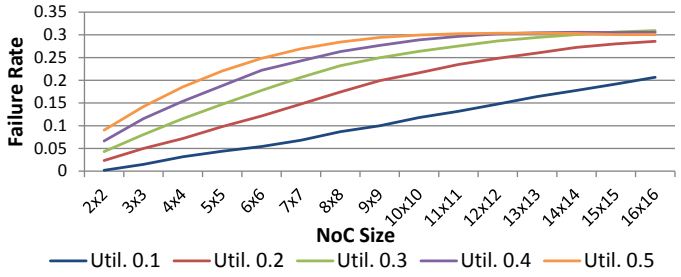

Fig. 13. The failure rate in allocating paths with feedback with varying network utilization. The number of time slots is 32.

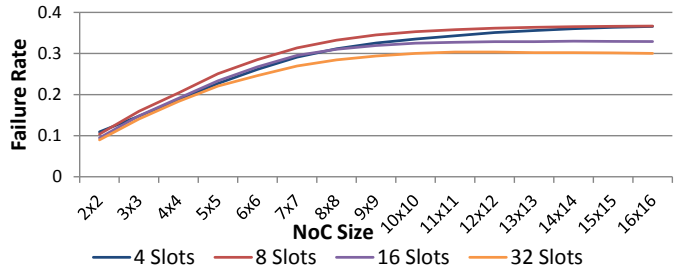

Fig. 14. The failure rate in allocating paths with feedback when the number of time slots is varied. The network utilization is 0.5.

is almost 3 times better than [16] thanks to the simple XSwitch architecture and the update mechanism.

The synthesis results of XNoC in different sizes and number of time slots are presented in Fig. 11. In these experiments, the data width is 32-bit and the feedback signal is 1-bit wide. The synthesis process is configured to optimize the area. As shown, the resource cost increases almost linearly with its size. The average cost incurred by doubling the number of time slots from *4 to 8*, *8 to 16* and *16 to 32* are 9%, 18% and 28%, respectively. Besides, the size of the XNoC has a little effect on the maximum achievable frequency. It only drops 1.5% when the number of nodes is more than 16.

Fig. 12 shows the ratios between the total number of the control links managed by the control plane and the total number of data links inside the Data Plane. It can be seen that the ratio decreases from 22% to 13% (for the data width of 32-bit) as the network size increases. Furthermore, the ratio is reduced by half when the data width is doubled. It is expected because the number of control links per node is relatively smaller than the data links and it is independent of the data width as illustrated by the black lines in Fig. 12.

The overhead of the distributed control plane is negligible in large network because of the following reasons. **First**, the state machine used to process instructions inside each SubCtrl is similar to the Control Unit. It takes 28 slices. **Second**, each multiplexer, de-multiplexer and AND gate in Fig. 8 occupies at most 1 slice since the *Ready* and *Handover* signals are only 1 bit each. **Third**, in Fig. 9, the size of the *Header ID* used in our experiments is 8 bits. This is large enough to avoid the roll-over effect where two headers have the same ID. The *Expected_Status* signal is 4 bits because we support at most 16 SubCtrls. The size of memory used to store the header status is therefore *256x4* bits. This memory can be implemented with only 1 Xilinx Virtex 6 true dual-port 18Kb BRAM. **Finally**, the number of wires used to connect 4 SubCtrls together as shown in Fig. 7 is 16. We need *10x4* wires for the update signals from SubCtrls to Control Agent. Similarly, *14x4* wires are required for the check signals. Adding all of the above resource costs, the incurred overhead of *2x2* distributed control plane is less than 0.5% of Xilinx Virtex 6 resources. Specifically, for the 6x6 XNoC reported in Fig. 11, the overhead of distributed control plane is just 2% and it is inversely proportional to the size of the network. More importantly, the configuration speedup can be up to $3X$ as shown later in Section V-D.

C. Path Scheduling with Feedback

In this section, we evaluate the impact of the condition presented in Equation 7 on the path scheduling. We develop a TDM path scheduling tool which utilizes the classic XY routing algorithm. The time slot allocations for the paths are selected randomly following the uniform distribution until the required number of slots is satisfied. We create a set of requests in which the source nodes, destination nodes and the required number of time slots are generated randomly in uniform distribution. The path scheduling algorithm tries to serve as many requests as possible until the total bandwidth occupation of the whole NoC reaches a predefined value. The *bandwidth utilization* of the network is the ratio of the total bandwidth *reserved* for all paths against the total bandwidth offered by the network. At this stage, the feedback scheduling is ignored. After that, all paths that are successfully allocated are used to create the requests (in the same order) for the new empty network. At this time, Equation 7 is taken into account. The number of requests that the algorithm cannot satisfy is used to calculate the *failure rate*. Every experiment is repeated 1000 times and the reported results are the average of them.

Fig. 13 and Fig. 14 present the failure rates across different network sizes, network utilizations and numbers of time slots. As expected, the failure rate increases with the network utilization as shown in Fig. 13 because the network becomes denser with less alternative choices. In Fig. 14, the failure rate for the same network with larger number of time slots

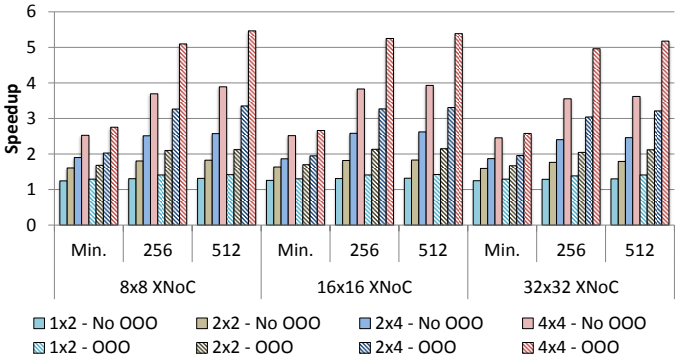

Fig. 15. The speedups gained from distributed control plane in different network sizes, instruction buffer sizes and number of SubCtrls. *The instructions are distributed to SubCtrls by Control Agent.*

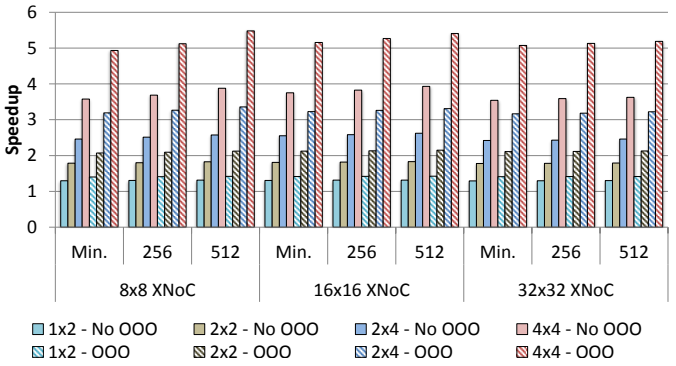

Fig. 16. The speedups gained from distributed control plane in different network sizes, instruction buffer sizes and number of SubCtrls. *The instructions are written directly to SubCtrls by Master Proc.*

is lower because of the observation we mentioned in Section III-C that we can shift the candidate time slots, if possible, to resolve the conflicts. In both experiments, the failure rate in bigger network is higher. The reason is that with the same number of time slots and network utilization, the number of paths allocated in bigger network is higher which causes more conflicts on path scheduling.

In these experiments, we use the XY routing algorithm for the paths with and without feedback, which can be a factor in causing the high failure rate. In future work, we need to design a better algorithm for this problem, in which the path length may be increased to find the suitable time slots. The Equation 7 would be analyzed further to have a smarter way in allocating time slots for the path rather than doing that randomly.

D. Distributed Control Plane

In this section, the performance of the XNoC with distributed control plane is compared against the XNoC with centralized approach. The performance is characterized in terms of speedup in different network settings. In Section IV-D, we mentioned about the head-of-line blocking phenomenon that hinders the performance of the distributed control plane. There are two solutions to alleviate this problem: (1) increase the instruction buffer size at SubCtrls; (2) use multiple data channels from Master Proc to write instructions directly to SubCtrls. We test these two approaches with different network sizes *8x8*, *16x16* and *32x32*. The number of network time slots in all three setups is 32. The size of the distributed control plane also varies from *1x2*, *2x2*, *2x4* to *4x4*. Additionally, we have three settings for the size of the SubCtrls instruction buffers: *minimum* number of words, *256* and *512* words. The width of instructions is 32 bits. The *minimum* number of words required for each instruction FIFO equals to the maximum possible length of the path in the region. For example, in 32x32 network with 4x4 control plane, each SubCtrl manages a region of 8x8. The maximum path length given by the XY routing algorithm in that region is 16 hence the size of the instruction FIFO. The size of the instruction FIFO in the centralized approach is computed similarly.

We have implemented a SystemC cycle-accurate model to simulate the operation of the control plane in XNoC. All connection (de)-activation (for both data and feedback) and the corresponding SW reconfiguration requests are generated based on the uniformly randomly chosen source and destination nodes. The SW reconfiguration request is not generated for connection deactivation. The XY routing algorithm is used to find the path. The maximum number of time slots requested by each connection is 8. Each node can request multiple connections. All instructions are created at the start of simulation. The configuration instruction is sent to the network in every clock cycle. The speedup is calculated based on the time needed by the network with and without distributed control plane to process all instructions. There are 1000 connection requests in each experiment. The experiments are run multiple times with different random seeds. The result reported is the average.

Fig. 15 and Fig. 16 show the speedup results of the network with two different methods of how instructions are fed to SubCtrls. The performance gained by using the out-of-order (OOO) technique presented in Section IV-C is also included in the graphs. It can be seen that the speedup increases with the size of the distributed control plane and the size of the instruction buffer. However, with the minimum instruction buffer size, the average speedup gained by doubling the number of SubCtrls in Fig. 15 is just 26% (28%) with (without) OOO. These numbers are 38% and 50% respectively in Fig. 16. This clearly shows the strong benefits of utilizing multiple data channel for instructions and the OOO technique. In all setups, the OOO technique improves the speedup from $4-9\%$ for centralized control unit and $8-38\%$ for distributed approach with minimum instruction buffer size.

Besides, for the head-of-line phenomenon, the results given in Fig. 15 show that it can be mitigated by increasing the instruction buffer size at SubCtrl. However, allocating that large amount of buffer to each SubCtrl is costly. Therefore, we suggest the second solution where multiple data channels are used to push instructions directly to SubCtrls. This method approximately offers the similar speedup compared with the large buffer size solution even with the minimum buffer size as reported in Fig. 16. The resource cost of this method is small because for each data channel, we only need one simple state machine to write data from the AXI-Stream slave interface to the instruction FIFO.

In large networks, many applications can be executed at the same time. Each application requires a set of PEs. The communication aware runtime mapping algorithm tends to allocate the PEs which are close to each other to each application to reduce the communication latency. It is very rare that the PEs from different corners of the network directly communicate with each other. This is one of the motivations for our distributed control plane where each SubCtrl manages its own region

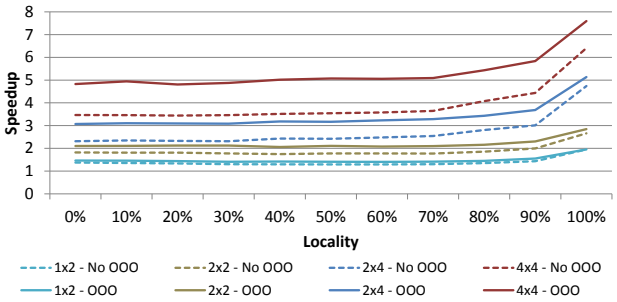

Fig. 17. The speedup achieved from 32x32 XNoC with default settings and varying *locality* values.

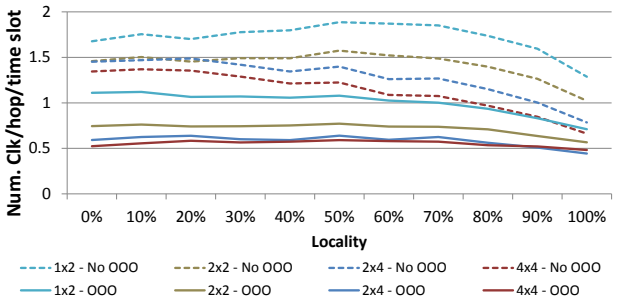

Fig. 18. The average number of clock cycles per hop per requested time slot taken to activate connections.

independently. Therefore, to further evaluate the performance of the distributed approach, we introduce the *locality* metric. It is used to model the behavior of the connection (de)-activation requests at runtime instead of fully randomizing them. The larger the locality, the higher the chance the connections are not crossing multiple regions. When the number of SubCtrls increases, the number of PEs in each physical region reduces. It becomes harder for the communication aware runtime application mapping algorithm to find a feasible mapping if it is only restricted to the available PEs in one region. Thus, we also define *communication regions*. These *regions* are superimposed over the physical ones as follows: *1x2* for *1x2* SubCtrls, *1x2* for *2x2* SubCtrls, *2x2* for *2x4* SubCtrls, and *2x2* for *4x4* SubCtrls.

Fig. 17 presents the speedup achieved from the 32x32 XNoC with different number of SubCtrls and minimum instruction buffer size. The instructions are written directly to SubCtrls by Master Proc. These are the default settings for XNoC. As expected, the speedup rises with the locality, especially from 60% to 100% range. The highest speedup is ***7.6X***. In the locality range of $0 - 60\%$, the speedup stays mostly the same for all network settings. The reason is that there is a large number of connections that cross multiple *communication regions*. They involve more SubCtrls to (de)-activate the connections. Therefore, at some point, they have to synchronize with each other. The synchronization lowers the freedom of each SubCtrl and hence the speedup.

The average number of clock cycles required to activate connections in a 32x32 XNoC with default settings is reported in Fig. 18. The results are calculated by dividing the time needed to activate a connection by its length and the requested number of time slots. The SWs reconfiguration time is not included because they can be hidden by the on-going data communication in the network. In this experiment, the average and maximum requested number of time slots is 8 and 16 respectively. The lengths of connections range from 10 to 26. This experiment is carried out based on the scenario where there are back-to-back connection activation requests generated by the Master Proc. In this case, the parallelism of distributed control plane is well exploited.

The average latency obtained from Fig. 18 (2x2 SubCtrls, OOO activation, locality of 50%) for XNoC with distributed control plane (*XNoC - D.*) and the worst-case latency for XNoC with centralized control unit (*XNoC - C.*) presented in Section III-D are used to compare with the results reported in [14] and [15]. One advantage of dAElite [14] is that it can configure all time slots at once with the trade-off of excessive usage of multiple distributed RAMs. However, in practice, it is unusual for one PE to require full bandwidth offered by the network just to communicate with another PE. Therefore, to have a fair comparison with our XNoC and [15] which activate 1 slot at a time, we set the total time slots of the network to 8. Each connection requests 8 time slots. The number of hops is from 6 to 12. The results are given in Table II. The numbers in parentheses indicate the worst case latency in which the data can be injected into the network. For *XNoC - D.* and [15], the path is ready to receive the first data after the first slot is activated plus the worst case turnaround time of 8 cycles. For *XNoC - C.*, this latency only depends on the total time slots of the network which is $8 + 2$. It can be seen from Table II that *XNoC - D.* outperforms all the other works.

TABLE II. CONNECTION ACTIVATION LATENCY

Works	6 Hops	8 Hops	10 Hops	12 Hops
dAElite [14]	81 (89)	94 (103)	119 (127)	133 (141)
Pr. Probe [15]	208 (34)	240 (38)	272 (42)	304 (46)
XNoC - C.	136 (**10**)	152 (**10**)	168 (**10**)	184 (**10**)
XNoC - D.	**37** (13)	**49** (14)	**62** (16)	**74** (17)

VI. CONCLUSIONS AND FUTURE WORKS

In this paper, a non-intrusive TDM CSw NoC, XNoC, is presented. The resource cost of our XNoC is shown to be smaller than the most known comparable NoC while offering more features. XNoC non-intrusive configuration process does not interfere with the data communication at all. The time needed to switch to the new configuration is strictly bounded by the length of the path and the number of time slots. XNoC is also the first NoC that supports many-to-one feedback for multi-cast service. The scheduling of the feedback signal is directly inferred from the data path. The distributed control plane is also proposed with low resource overhead and superior performance over the centralized approach and other works.

The forthcoming works are to design a better routing algorithm for the path-with-feedback and to explore the performance of XNoC in real-life SoC with multiple applications using partially reconfigurable PEs.

VII. ACKNOWLEDGMENTS

This work is supported in part by the German Research Foundation (DFG) within the Cluster of Excellence “Center for Advancing Electronics Dresden” (cfaed) at the Technische Universität Dresden.

REFERENCES

[1] A. Agarwal, C. Iskander, and R. Shankar, "Survey of network on chip (noc) architectures & contributions," *Journal of engineering, Computing and Architecture*, vol. 3, no. 1, pp. 21–27, 2009.

[2] S. Hesham, J. Rettkowski, D. Göhringer, and M. A. A. El Ghany, "Survey on Real-Time Network-on-Chip Architectures," in *Applied Reconfigurable Computing*. Springer, 2015, pp. 191–202.

[3] A. Agne, M. Happe, A. Keller, E. Lubbers, B. Plattner, M. Platzner, and C. Plessl, "ReconOS: An operating system approach for reconfigurable computing," *Micro, IEEE*, vol. 34, no. 1, pp. 60–71, 2014.

[4] T. D. Nguyen and A. Kumar, "PR-HMPSoC: A versatile partially reconfigurable heterogeneous Multiprocessor System-on-Chip for dynamic FPGA-based embedded systems," in *Field Programmable Logic and Applications (FPL), 2014 24th International Conference on*. IEEE, 2014, pp. 1–6.

[5] K. Goossens and A. Hansson, "The aethereal network on chip after ten years: Goals, evolution, lessons, and future," in *Design Automation Conference (DAC), 2010 47th ACM/IEEE*. IEEE, 2010, pp. 306–311.

[6] S. Liu, A. Jantsch, and Z. Lu, "Analysis and evaluation of circuit switched NoC and packet switched NoC," in *Digital System Design (DSD), 2013 Euromicro Conference on*. IEEE, 2013, pp. 21–28.

[7] N. D. E. Jerger, L.-S. Peh, and M. H. Lipasti, "Circuit-switched coherence," in *Proceedings of the second ACM/IEEE international symposium on networks-on-chip*. IEEE Computer Society, 2008, pp. 193–202.

[8] J. Yin, P. Zhou, S. S. Sapatnekar, and A. Zhai, "Energy-efficient time-division multiplexed hybrid-switched noc for heterogeneous multicore systems," in *Parallel and Distributed Processing Symposium, 2014 IEEE 28th International*. IEEE, 2014, pp. 293–303.

[9] A. Leroy, P. Marchal, A. Shickova, F. Catthoor, F. Robert, and D. Verkest, "Spatial division multiplexing: a novel approach for guaranteed throughput on NoCs," in *Proceedings of the 3rd IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*. ACM, 2005, pp. 81–86.

[10] Z. J. Yang, A. Kumar, and Y. Ha, "An area-efficient dynamically reconfigurable spatial division multiplexing network-on-chip with static throughput guarantee," in *Field-Programmable Technology (FPT), 2010 International Conference on*. IEEE, 2010, pp. 389–392.

[11] S. Liu, A. Jantsch, and Z. Lu, "MultiCS: Circuit Switched NoC with Multiple Sub-Networks and Sub-Channels," *Journal of Systems Architecture*, 2015.

[12] K. Goossens, J. Dielissen, and A. Radulescu, "Æthereal network on chip: concepts, architectures, and implementations," *Design & Test of Computers, IEEE*, vol. 22, no. 5, pp. 414–421, 2005.

[13] A. Hansson, M. Subburaman, and K. Goossens, "aelite: A flit-synchronous network on chip with composable and predictable services," in *Proceedings of the Conference on Design, Automation and Test in Europe*. European Design and Automation Association, 2009, pp. 250–255.

[14] R. A. Stefan, A. Molnos, and K. Goossens, "daelite: A tdm noc supporting qos, multicast, and fast connection set-up," *Computers, IEEE Transactions on*, vol. 63, no. 3, pp. 583–594, 2014.

[15] S. Liu, A. Jantsch, and Z. Lu, "Parallel probe based dynamic connection setup in TDM NoCs," in *Proceedings of the conference on Design, Automation & Test in Europe*. European Design and Automation Association, 2014, p. 239.

[16] A. K. Lusala and J.-D. Legat, "A SDM-TDM-Based Circuit-Switched Router for On-Chip Networks," *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 5, no. 3, p. 15, 2012.

[17] J. Sparso, E. Kasapaki, and M. Schoeberl, "An Area-efficient Network Interface for a TDM-based Network-on-Chip," *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 1044–1047, 2013.

[18] T. Krishna and L.-S. Peh, "Single-cycle collective communication over a shared network fabric," in *Networks-on-Chip (NoCS), 2014 Eighth IEEE/ACM International Symposium on*. IEEE, 2014, pp. 1–8.

[19] R. Bittner and P. Athanas, "Wormhole run-time reconfiguration," *Proceedings of the 1997 ACM Fifth International Symposium on Field Programmable Gate Arrays*, pp. 79–85, 1997.