VERSAL NETWORK-on-CHIP (NoC)

Ian Swarbrick
Xilinx Inc.
San Jose, California
Email: iswarbri@xilinx.com

Dinesh Gaitonde
Xilinx Inc.
San Jose, California
Email: dineshg@xilinx.com

Sagheer Ahmad
Xilinx Inc.
San Jose, California
Email: sagheer@xilinx.com

Bala Jayadev
Xilinx Inc.
San Jose, California
Email: balakri@xilinx.com

Jeff Cuppett
Xilinx Inc.
San Jose, California
Email: jcuppett@xilinx.com

Abbas Morshed
Xilinx Inc.
San Jose, California
Email: morshed@xilinx.com

Brian Gaide
Xilinx Inc.
San Jose, California
Email: brian.gaide@xilinx.com

Ygal Arbel
Xilinx Inc.
San Jose, California
Email: ygala@xilinx.com

***Abstract*—Xilinx Versal Adaptable Compute Acceleration Platform (ACAP) is a new software-programmable heterogenous compute platform. The slowing of Moores law and the ever-present need for higher levels of compute performance has spurred the development of many domain specific accelerator architectures. ACAP devices are well suited to take advantage of this trend. They provide a combination of hardened heterogenous compute and IO elements and programmable logic. Programmable logic allows the accelerator to be customized in order to accelerate the whole application. The Versal Network-on-Chip (NoC) is a programmable resource that interconnects all of these elements. This paper outlines the motivation for a hardened NoC within a programmable accelerator platform and described the Versal NoC.**

1. Introduction

A few major trends require a rethinking of how data movement is implemented on FPGAs. Firstly, FPGAs are becoming platforms on which significantly more complex designs are being implemented than just a few years ago. Memory and IO interfaces have also increased their bandwidths by several factors. FPGAs have become an attractive platform for a different class of designs - compute acceleration [1]. For users looking to FPGAs to accelerate diverse compute applications, the expertise of the designers lies in individual domain specific architectures and not the details of FPGA interconnect architecture.

The slowdown of Moore's law scaling brings many challenges. Although transistor performance has been scaling with technology, metal parasitics have not [2]. Interconnect delays are rising even though distances travelled shrink due to scaling.

FPGAs today are able to customize buses and manage data movement in ways that can be customized down to the bit level. As data rates of IO and memory interfaces increase and in the face of wire scaling challenges some flexibility must be traded off to make device global data movement more efficient. This is one of the driving reasons to include a hardened NoC within an FPGA accelerator. This still leaves the option to build custom interconnect structures with a subdomain but chip wide memory mapped data movement can be handled with much greater resource efficiency.

1.1. Example Device

Figure 1 shows an abstracted example of a Versal device. The device includes hardened blocks such high speed IOs with hardened memory controllers, a processor system, platform manager, high speed Serdes and AI Engines. Details of the ACAP architecture elements can be found in [3]. The center of the device is populated with FPGA resources. The NoC globally connects all of these elements.

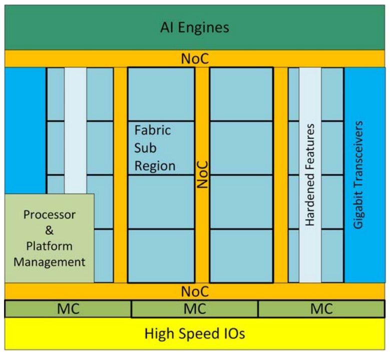

Figure 1: Example Versal Device (abstracted)

1.2. Motivation for Hardened NoC

NoCs have been used widely in ASICs and SoCs. Here we describe the motivations for the Versal NoC, most or all of which are unique to a programamble device. Firstly, Versal is the Xilinx 7nm family of devices. The NoC is constructed from a set of re-usable sub-blocks composed in

2332-5569/19/$31.00 ©2019 IEEE
DOI 10.1109/HOTI.2019.00016

different ways on each device. The combination of these sub-blocks is device specific and the topology of the network is not fully determined until integration time. The networks constructed from these sub-blocks must (1) meet the communication requirements of the application and (2) be deadlock free. The network can be of an arbitrary size and topology. Devices within the product family vary greatly in size. A small device may have limited fabric and a single memory controller with some NoC and low-single-digit master and slave NoC ports. A large device may have many memory controllers, a large fabric array and tens or hundreds of NoC ports. Different devices in the product family are built using the same sub-blocks to maximise re-use. The NoC network must be able to scale up with increasing device size. There are two main degrees of scaling. The NoC bandwidth needs to scale up with the number of DDR memory channels (for example) on the device. Additionally, larger devices need an increased number of ingress and egress ports to provide fabric access to the NoC. Programmable devices are deployed in a broad variety of end-markets, with differing requirements. The NoC must be able to adapt to the traffic flows of the target application. For this, it must be possible to define the NoC routes, control the bandwidth and apply Quality-of-Service requirements to the traffic flows. The traffic flows need to be programmable at boot time and re-programmable subsequently. In an ASIC or SoC NoC, the data width and memory addressing features of each master and slave are known at integration time. The network topology is fixed to match the application. In a programmable device the communication requirements of a kernel are determined at run time. The routes through the NoC must be programmable to meet the needs of the current application. Data width, interface protocol and memory map must be flexible too. The Versal NoC allows these features to be dynamically programmed and re-programmed.

1.3. NoC Overview and Features

The Versal NoC is a Packetized transport. The NoC employs Wormhole routing [4]: flow control is managed at the flit level. Routing uses distrbuted routing tables at the input port of each switch. There is one route table for each VC at each input port. A destination identifier for the packet is used at the input of each switch to look-up the output port. The links between switches are 1 GHz full duplex channels. The data payload within each flit is 128 bits. There 8 Virtual channels per physical link. This can support 2 full AXI connections in each direction on a full-duplex link. End-to-end Quality of service is implemented with three traffic classes: Low-latency (LL), Isochronous (ISOC) and best-effort (BE). Traffic classes are mapped onto virtual channels. Each virtual channel carries traffic of one traffic class, but may carry multiple flows. Deficit round robin arbitration is performed at every switch. QoS is described further in [5]

Figure 2 shows the basic NoC switch element. Each unidirectional link has flits transmitted in one direction and credits returned in the opposite directions. The switches have 4 inputs and 4 output ports. They are not full crossbars, a

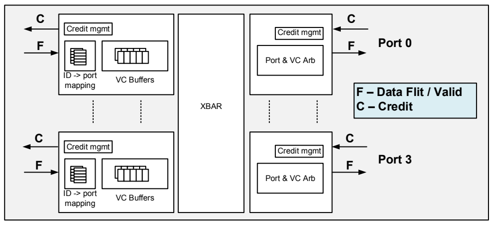

Figure 2: Example Versal Floorplan

packet cannot be routed ’backwards’. Each switch has a minimum latency of 2 cycles from any input port to any output port. The entry point to each NoC is termed NoC Master Unit (NMU) and the exit points are called NoC Slave Units (NSU). A master connects to an NMU and and slave connects to an NSU. Fabric interfaces have programmable data width from 32-512. Other interfaces connected to hard IPs have fixed data width. The fabric interafces are configurable to support the AXI Stream protocol as well. When addressing DDR the NoC support 1,2 and 4 channel memory interleaving. This simplifies the applications use of memory - multiple DDR channels appear as a single pool of memory and the system designer doesn’t need to be concerned about where data is placed and how load balancing works across channels. To simplify integration there is an asynchronous clock boundary at all ingress/egress ports of the NoC. The NoC has a separate peripheral interconnect that is configured at integration time. This provides a path to program the NoC that does not depend on any NoC resources.

1.4. Routing and topology

The NoC topology is not regular, being driven by the specific bandwidth requirements of the heterogenous elements in the device. A regular topology such as a mesh or torus is not desirable since it would impinge too much on resources such as accelerators, programmable logic and gigabit transceivers. The NoC components are arranged into horizontal and vertical sub-blocks (HNoC and VNoC). Horizontal NoCs are placed at the top and bottom of the device with differing numbers of physical channels depending on device requirements. Vertical NoCs are placed at regular intervals embedded within the programmable logic. The number of VNoCs is scaled up for bigger devices. Each VNoC has 2 physical channels. HNoCs typically have 2 or 4 physical channels. The set of VNoCs/HNoCs does not form a full mesh topology (for example) but there is parallelism and path diversity in the resulting network. Figure 3 shows a simplified example topology.

Routing is deterministic and can be minimal or non-minimal. The protocols mapped onto the NoC require certain ordering guarantees. Supporting these while using adaptive routing would add unnecessary complexity. Allowing non-minimal routes provides a richer set of routes and helps with load balancing. QoS requirements given by the user indicate which routes are more tolerant to using longer paths.

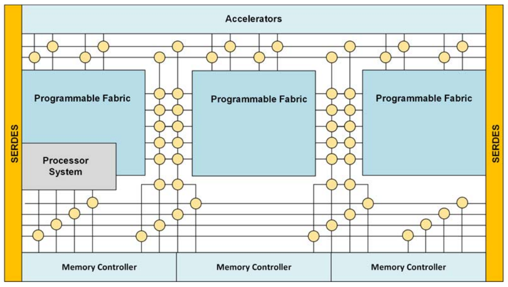

Figure 3: Example NoC Topology

Versal NoC uses distributed routing tables. There is a per-VC table at each input port of each switch. When a packet header flit arrives at a switch port, at 12-bit destination identifier (DestID) is checked against a table to lookup up the switch output port. To limit the size of the routing tables, the DestID lookup is organized hierarchically. The identifier is organized into three parts, high, mid and low. The routing table is partitioned into three parts. In the example shown where the high, mid and low ID bits are 6,2 and 4 bits respectively, each routing table contains $2^6+2^2+2^4$ entries. Each entry is 2 bits (4-port switches) so one routing table will be 168 bits in size.

The routing scheme allows the NoC to contains up to 4096 (input ports), forming an arbitrary topology.

Versal NoC topologies are constructed from re-usable sub-blocks that can be connected differently for different devices. A machine-readable represention of the resulting NoC topology is used by tools to generate programming information for the NoC.

2. NoC Compiler

Within the context of FPGAs, almost all aspects of the fabric are programmable. This results in several new design automation problems that are unique to this combination of traditional FPGA fabric and packet switched NoC. In this section we describe some of the problems unique to the platform described in this paper and how to pose and solve them.

In the case of SOCs, individual blocks are designed somewhat independently. Communication between blocks is then implemented using a custom designed global interconnect - for example a NoC. As a result, all aspects of the NoC are tailored for the specific problem under consideration. Specifically, the NoC topology, addressing and routing can all be designed specifically for the target design. In a manner similar to traditional FPGA fabric, Versal ACAP NoC provides enough functionality to efficiently map a variety of workloads. The tools supporting these mappings have to conform to the specific architectural features of the Versal NoC.

In order to motivate all the requirements of a toolchain we will use a very simple example shown in Figure 4.

In this example we have some blocks implemented in fabric that communicate with each other and with some

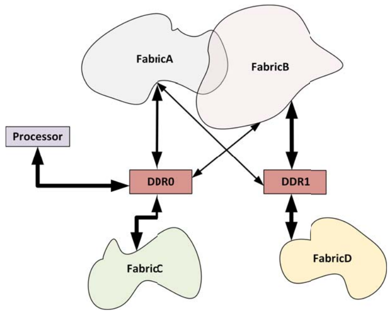

Figure 4: Example NoC Implementation Problem

other ACAP resources. There are two forms of communication in this example. There is the explicit strucutred communication that is expected to be routed over the NoC and there is the implicit communication that is not explicitly expressed but exists in the form of the connections that make up the netlist that is to be mapped on the fabric. In this network for example, the overlap between fabric mapped blocks *FabricA* & *FabricB* is used to indicate that a significant number of nets in the design have connections in both these blocks. On the other hand, the other two fabric blocks *FabricC* and *FabricD* have mostly structured communications with DDR. The communication requirements of the structured portion of the communication are expxlicitly enumerated by the user. In the example shown in Figure 4 one possible specification for the required structured communication is shown in Table 1.

Master	Slave	Bandwidth (GBytes/sec)	QoS
Processor	DDR0	2	Low Latency
FabricA	DDR0	6	Best Effort
FabricA	DDR1	2	Best Effort
FabricB	DDR0	2	Best Effort
FabricB	DDR1	6	Best Effort
FabricC	DDR0	6	Best Effort
FabricC	DDR1	2	Best Effort
FabricD	DDR0	2	Best Effort
FabricD	DDR1	6	Best Effort

TABLE 1: Required Bandwidth and QoS for example network

The first problem that the NoC compiler needs to solve is to decide which of the various fabric NoC interfaces it uses to implement the structured communication. This involves negotiating two (sometimes conflicting) goals. The compiler needs to ensure that it does not make the NoC routing problem too hard. That is it needs to ensure that the blocks that demand some bandwidth from a particular resource are placed close to the resource that supplies it. The compiler also needs to ensure that when it chooses the ports for the fabric blocks, it does not create a tough problem for the fabric implementation. As an example of a suboptimal mapping of the problem in Table 1, consider the one shown in Figure 5(a).

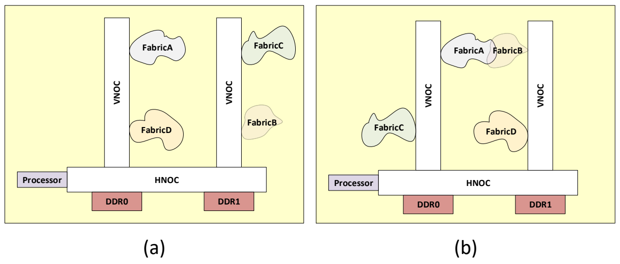

Figure 5: (a) SubOptimal and (b) Optimal Mapping Of the Example Problem in Table 1

There are a couple of things that are suboptimal in this mapping. First, the masters in the fabric are not placed in a manner that aligns with the resources from which the bandwidth is demanded. It is possible that all the demands will be met with this mapping, but more routing resources will be used in implementing the flows than necessary. Secondly, *FabricA* and *FabricB* blocks are placed far away from each other. There is substantial unstructured communication between them which will needed to be implemented in the fabric. Placing the ports far away imposes a more taxing burden on that portion of the implementation. Instead the mapping shown in Figure 5(b) performs a mapping that optimizes for routability for the NoC portion and also for the portion that will be implemented in the fabric.

Having decided on the which ports in the NoC topology each master and slave is connected to, one now needs to route the flows using the NoC so that all the requirements are satisfied. Routing implies finding a path from each master to each slave such that no link along the path is oversubscribed. In the context of NoC routing, the routes have to satisfy different constraints. These constraints arise both from what it means to route on a NoC in general and also what it means to route on a NoC given the choices we have made in defining the architecture of a Versal NoC. The rest of the section is focussed on understanding how the tool chain deals with these constraints.

Needless to say, any routing solution that claims to satisfy the flows required of a problem has to ensure that no link is oversubsribed. This is a constraint universal to all NoCs. In fact it is similar to constraints that traditional FPGA routing has to satisfy. In traditional FPGA routing every link is expected to route only one net. In NoC routing every link can route as many flows as needed provided the sum of expected bandwidth used by each flow utilizing a link is not greater than the link capacity. One major constraint when routing NoC flows is to avoid deadlock.

Consider the NoC topology shown in Figure 6 and associated routing problem from Table 2.

A natural solution to this problem is to route each of the flow in the counter clockwise direction. Unfortunately, even though this might not oversubscribe any of the links, such a solution could result in deadlock. Understanding why this creates a deadlock requires one to understand the dependencies between the various links that make up the

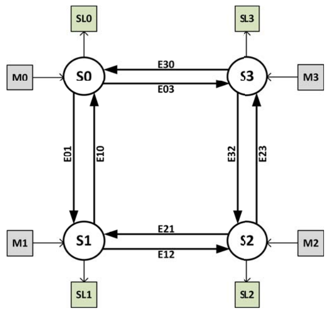

Figure 6: Example Network For Deadlock

flows. We say tat some edge EA depends on some edge EB if both EA and EB are used to route a particular flow. Figure 7 shows the dependencies between the various edges in our solution to the problem in Table 2.

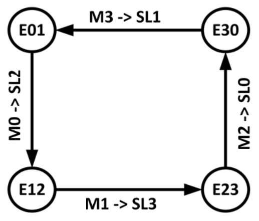

Figure 7: Channel Dependency Graph

Master	Slave
M0	SL2
M1	SL3
M2	SL0
M3	SL1

TABLE 2: Flows Required

Here, vertices represent edges in the network and an edge represents a dependency between the two edges in the original network. Each edge in this figure is annotated with the flow whose solution implies this dependency. The reason why this solution deadlocks is the presence of a cycle in the dependency graph in the solutuon shown in Figure 7. This relation between channel dependency graph and deadlock has been studied by other researchers [6] [7] . In the NoC compiler, we need to ensure that this situation never occurs. The router has to guarantee that the channel dependency graph is cycle free. This issue of deadlock free routing is also not unique to Versal NoCs. Every packet switched NoC routing has to ensure that deadlock does not occur. In Versal NoC as in other NoCs, there are usually two ways such a situation is avoided. The simplest is to use virtual channels to isolate the various flows. The second technique is to solve the routing problem in such a way that the implied channel dependency graph is acyclic. In the network shown in Figure 6, if the router insisted on routing the horizontal portion of the flow before the vertical portion, the resulting solution is guaranteed to be deadlock free. Such a simple prioritization of the routing algorithm is possible only when the underlying NoC topology is regular. Versal NoC compiler uses both virtual channels and constrained routing to ensure deadlock free routes.

As mentioned previously the NoC programming is done through routing tables. When a packet arrives at the input port of a switch the packet destination ID (destid) is compared with a programmed value for the switch. One of the tasks that the NoC compiler has to decide on is assigning the low, mid and high IDs to each entity in the topology. This is an optimization problem, since a suboptimal assignment could result in a problem being unroutable. In order to understand this consider the algorithm a switch uses to route a packet. The output port is now a function of the destination high, mid and low IDs, the virtual channel and the input port on which the packet arrives. The actual switching logic is shown in listing 1.

```
Result: Output port given input port & VC
if switch high ID $\neq$ destination high ID then
    output port is function of destination high ID;
else
    if switch mid ID $\neq$ destination mid ID then
        output port is function of destination mid ID;
    else
        output port is function of destination low ID;
    end
end
```

Algorithm 1: Output Port Algorithm At Each Switch

The use of hierarchical routing tables places constraints on the compiler. In particular when two flows to the same destination have merged together, they cannot subsequently split.

The NoC router solves the problem of realizing all the demanded flows ensuring that no link is oversubscribed and the solution is deadlock free and it all the various constraints imposed by the architecture. The result of programming each switch implies that we define a function at each switch input which considers the destination high, mid and low IDs, the virtual channel and determines which output port to route the traffic to.

Given that the topology of the NoC is not fixed or regular requires us to approach the NoC routing problem in a more general fashion than earlier solutions. For the Versal platform we pose the NoC routing problem as solving a SAT problem. We pose all the constraints mentioned above including the traditional NoC constraints of link subscription and deadlock avoidance as a set of boolean constraints that the eventual solution has to satisfy. We use the popular minisat program [8] as the satisfiability (SAT) solver to generate the routing solution from the constraints. SAT is a very general technique to pose constraints. In general, it can be used to solve for constraints which have very little structure to them. Their biggest downside is the potential for very large run times. For the NoC routing problem, however, the problem sizes are small enough for SAT to be a practical technique. Moreover, the complicated nature of deadlock and high, mid, low ID constraints make SAT a natural framework to pose this problem. In practice, for the NoC routing problem, SAT solver is able to route even the most demanding flows in a few minutes.

3. Conclusion

This paper outlined the Versal NoC that is part of Xilinx 7nm ACAP solutions. The NoC addresses many longstanding difficulties moving data in programmable devices in an efficient, scalable manner. Having a hardened high bandwidth network-on-chip has numerous benefits. Significant fabric area can be saved in designs that move large amounts of data and/or involve switching. The user design flow can be simplified by pushing timing closure from a global concern to one at the module level. Differential QoS ensures that chip-wide traffic can be appropriately classified and managed. A compiler tool raises the level of user abstraction to the traffic flow level. The compiler manages competing traffic constraints given by the user and provides reporting on the achievable solution. The Versal NoC is a new feature of Xilinx FPGAs in the 7nm generation. It raises the level of design abstraction for users. It fits within the ACAP concept by decoupling acceleration kernels from the various hardened pieces of the surrounding platform.

References

[1] A. M. Caulfield, E. S. Chung, A. Putnam, H. Angepat, J. Fowers, M. Haselman, S. Heil, M. Humphrey, P. Kaur, J. Kim, D. Lo, T. Massengill, K. Ovtcharov, M. Papamichael, L. Woods, S. Lanka, D. Chiou, and D. Burger, "A cloud-scale acceleration architecture," in *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, Oct 2016, pp. 1–13.

[2] M. T. Bohr, "Interconnect scaling - the real limiter to high performance ULSI," in *Proceedings of International Electron Devices Meeting*, Dec 1995, pp. 241–244.

[3] Xilinx White paper, Versal: The First Adaptive Compute Acceleration Platform (ACAP). [Online]. Available: https://www.xilinx.com

[4] L. M. Ni and P. K. McKinley, "A survey of wormhole routing techniques in direct networks," *Computer*, vol. 26, no. 2, pp. 62–76, Feb 1993.

[5] I. Swarbrick, D. Gaitonde, S. Ahmad, B. Gaide, and Y. Arbel, "Network-on-Chip Programmable Platform in Versal ACAP Architecture," in *Proceedings of the 28th ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 02 2019, pp. 212–221.

[6] C. Glass and L. Ni, "The turn model for adaptive routing," in *Proceedings the 19th Annual International Symposium on Computer Architecture*, May 1992.

[7] Dally and Seitz, "Deadlock-free message routing in multiprocessor interconnection networks," *IEEE Transactions on Computers*, vol. C-36, no. 5, pp. 547–553, May 1987.

[8] MiniSat Page. [Online]. Available: http://minisat.se/