Published in IET Computers & Digital Techniques
Received on 18th July 2010
Revised on 15th August 2011
doi: 10.1049/iet-cdt.2010.0105

ISSN 1751-8601

Customisation of on-chip network interconnects and experiments in field-programmable gate arrays

J.Y. Hur[1] *T. Stefanov*[2] *S. Wong*[1] *K. Goossens*[3]

[1]*Computer Engineering Laboratory, TU Delft, The Netherlands*
[2]*Leiden Embedded Research Center, Leiden University, The Netherlands*
[3]*Faculty of Electrical Engineering, TU Eindhoven, The Netherlands*
E-mail: JaeYoung.Hur@gmail.com

Abstract: Conventional rigid and generalpurpose on-chip networks occupy significant logic and wire resources in field-programmable gate arrays (FPGAs). To reduce the area cost, the authors present a topology customisation technique, using which on-demand network interconnects are systematically established in reconfigurable hardware. First, the authors present a design of a customised crossbar switch, where physical topologies are identical to logical topologies for a given application. A multiprocessor system combined with the presented custom crossbar has been designed with the ESPAM design environment and prototyped in the FPGA device. Experiments with practical applications show that the custom crossbar occupies significantly less area, maintains higher performance and reduces the power consumption, when compared with the general-purpose crossbars. In addition, the authors present that configuration performance and cost can be improved by reducing the functional area cost in FPGAs. Second, a customisation technique for the circuit-switched network-on-chip (NoC) is presented, where only necessary half-duplex interconnects are established for a given application mapping. The presented customised NoC is implemented in FPGA and results indicate that the area is reduced by 66%, when compared with the general-purpose networks.

1 Introduction

In modern multiprocessor system on chip (MPSoC), the communication performance of the network interconnects is increasingly becoming a significant and determining factor for the overall system performance. It is well known that crossbar interconnects provide high performance and reduced traffic congestion. Compared with shared buses, the performance increases because of the parallel nature of communication of transactions. The conventional full crossbar establishes all-to-all interconnects to accommodate all possible traffic patterns, which are in most cases unknown. Therefore the advantage of the crossbar is that any logical topology can be mapped to a physical interconnect with single-hop latency. Nevertheless, a major bottleneck of the conventional crossbar is the limited scalability and high area cost because of the quadratically increasing amount of wires as the number of ports grows. Recently, network-on-chip (NoC) as a design paradigm has been introduced to enhance the scalability and related issues [1]. NoC can be referred to as a network of crossbars and achieves the scalability by sharing wires, over which packets are communicated on a multi-hop basis. The NoC can be packet-switched and circuit-switched networks. A typical packet-switched network provides only the best-effort service. It has problems such as unpredictable delay and throughput mainly because of blocking of traffic inside the network. The blocking problem can be alleviated by virtual channels using multiple queues in routers, while a router with congestion-handling mechanism occupies significant on-chip resources. Accordingly, we consider a circuit-switched NoC. However, many multi-core systems still incorporate rigid and general-purpose interconnection networks, where a significant portion of network resources are typically under-utilised by a given application.

Modern MPSoC requires a short time-to-market and adaptability for targeted applications. Field-programmable gate arrays (FPGAs) meet these requirements and are emerging as a main component in modern SoC platform. We draw a conceptual diagram of an FPGA comprised of a functional plane and a configuration plane, as depicted in Fig. 1. The functional plane contains the configurable logic blocks (CLBs), the input/output blocks and reconfigurable interconnects. The configuration plane contains a configuration controller and a datapath including the configuration memory cells. The flexibility in the functional plane is realised by the circuitry in the configuration plane. Each element in the functional plane is configured by writing bitstreams onto associated configuration memory cells in the configuration plane. Moreover, modern FPGAs have the capability to (dynamically) reconfigure only part of its resources. This operation is called partial reconfiguration. When the network intellectual properties (IPs) occupy significant reconfigurable resources, a significant portion of the configuration memory (or bitstream) is allocated for the partial reconfiguration of the

© The Institution of Engineering and Technology 2012

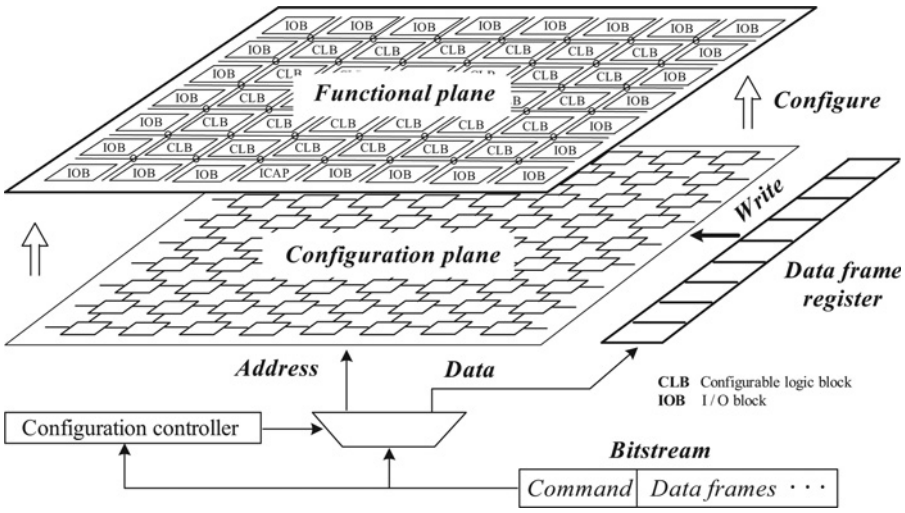


Fig. 1 *Conceptual diagram of FPGA*

on-chip networks. In this case, the bitstream size accordingly increases and the configuration time also increases. Therefore high area cost in the functional plane incurs decreased performance and increased cost in the configuration plane. In other words, the reduction of the functional area cost can be beneficial from the configuration perspective.

In this work, we reduce the high-cost problem in a crossbar and a circuit-switched NoC, by constructing on-demand interconnects in a reconfigurable platform. This work is motivated by the observation that communication patterns of different applications represent different logical topologies. The applications in most cases require only a small portion of all-to-all communications. Fig. 2 depicts data flow graphs of realistic applications indicating that the required topologies are much simpler than an all-to-all topology. Additionally, Fig. 2 depicts the number of nodes and the number of links for the task graphs. As an example, MJPEG{6,14} indicates that the MJPEG application requires six nodes and 14 links.

In this paper, we present a systematic design, an implementation and an analysis of our customised interconnects. The main contributions of this work are:

- We present a table-based topology customisation technique to implement crossbar switches, using which the crossbar provides identical physical topologies to arbitrary topologies that an application requires. The proposed custom crossbar has been integrated in the automated

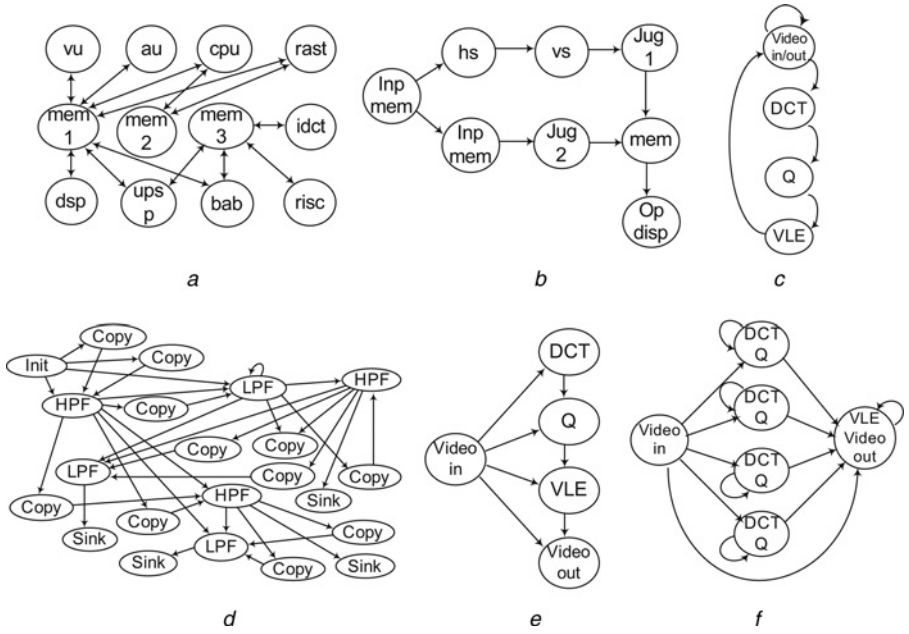


Fig. 2 *Parallel specifications of practical applications*

a MPEG4 {12,26}
b PIP {8,8}
c MJPEG {4,5}
d Wavelet {22,36}
e MJPEG {5,7}
f MJPEG {6,14}
Application{*m*, *n*} indicates that the application requires *m* nodes and *n* links

© The Institution of Engineering and Technology 2012

doi: 10.1049/iet-cdt.2010.0105

ESPAM design flow [2, 3] and prototyped. Implementation results for the MJPEG applications show that our custom crossbar reduces the area by 67% and the power consumption by 71%, compared to reference crossbar interconnects.

- To reduce the area cost, we present a customisation technique for the NoCs. The topology embedding is conducted, using which the utilised resources of the NoC and topology tables are determined. Using our table-based technique, only necessary inter-router and intra-router network resources are established. As a result, an experiment indicates that 66% of the area is reduced when compared with the general-purpose NoC.
- We examine the configuration performance and cost in FPGAs. A lower bound of the configuration time and required bitstream sizes are obtained. As a result, customised interconnects can provide better configuration performance and cost (66% on average).

The organisation of this paper is as follows. In Section 2, related work is presented. In Section 3, the design and implementation of the customised crossbar interconnects are described. In Section 4, the customised circuit-switched NoCs are described. Finally, conclusions are drawn in Section 5.

2 Related work

A crossbar consists of a switch module and a traffic controller. A design of a customised scheduler is described in [4], where the on-demand scheduler is systematically implemented. In [5], customised crossbar switch is presented. In this paper, we further present implementation and prototype results for realistic benchmarks. Additionally, we present a customisation technique for (circuit-switched) NoCs. Furthermore, we also analyse the performance and cost from the configuration perspective in FPGAs.

In [6, 7], an application-specific crossbar generation is presented, which is similar to our work in that the area is reduced. A simulation-based approach is adopted in [6, 7], where an application traffic tracing is conducted to synthesise the application-specific topology, which requires hours of design space exploration time. However, in our method, on-demand topology information is systematically extracted in the application specification step. Using the extracted topology information, variable-sized custom (shared) arbiters are connected to slaves and different-sized multiplexers are utilised. Our method allows that any topology (from a single node to an all-to-all topology) can be systematically implemented. In [6], the customised crossbar performs lower than the full crossbar. However, our method ensures that the customised crossbar does not perform lower than the full crossbar. Our experiment indicates that the presented customised crossbar performs better and reduces power consumption. Additionally, a multiprocessor system combined with our custom crossbar was rapidly prototyped on the reconfigurable hardware using the ESPAM tool chain. Finally, we obtained the cost and performance in the configuration layer as well as the functional layer.

Recently, a number of application-specific NoC designs are proposed. In [8], a topology-adaptive parameterised network component is presented. The physical topology in [8] is constructed between packet routers. Our NoC customises the topology based on the half-duplex routing paths. This means the intra-router resources in our NoC can be customised, while each router in [8] is not customised. In [9], an individually customised switch for NoCs is presented. In [9], parameters are specified for an individual multiplexer instance and an arbiter instance. Our work is similar to [9] in that on-demand topology is configured for a switch module. Based on topology embedding, our method further customises the intra-router buffers, as well as inter-router half-duplex links. In other words, our customisation allows the systematic removal of un-utilised buffers and interconnects in an entire circuit-switched NoC.

3 Construction of on-demand topology

Our goal is to design a crossbar in which the physical topology and logical topology are identical. The physical interconnects are additionally required to be instantly switched to adaptively meet the dynamic traffic patterns. In this section, we describe a system organisation and an implementation of the presented switch module.

3.1 System overview

In this work, the Kahn Process Network (KPN) model of computation is considered as a programming model. A KPN is a network of concurrent processes that communicate over unbounded first in first out (FIFO) channels and synchronise by a blocking read on an empty FIFO. The main advantage is that the synchronisation scheme is relatively simple. Processors synchronise only with the full/empty status of the hardware FIFO. Subsequently, a parallel programmer does not need to explicitly handle the synchronisation, since the synchronisation is inherently supported by the hardware. Fig. 3 depicts a simplified

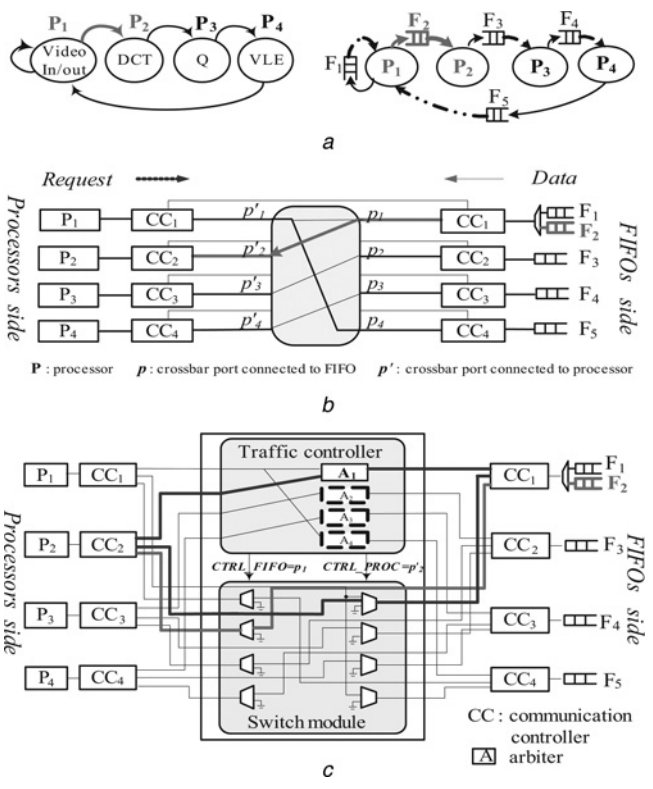


Fig. 3 *Simplified system organisation*

a Four-node MJPEG application
b Port-mapped system
c System organisation
Bold lines depict that processor P_2 reads from a remote FIFO F_2

© The Institution of Engineering and Technology 2012

system organisation for a four-node MJPEG application. In our system, the crossbar transfers requests (from processors) and data (from FIFOs). The communication controller in [2] is used in this work as a common network interface. Fig. 3*b* depicts that processor P_2 reads from a remote memory FIFO F_2 as represented by the bold line. The (simplified) customised switch module is depicted in Fig. 3*c*. Multiplexers for other ports are not depicted for the sake of clarity. P_2 sends a read-request to the traffic controller and the traffic controller forwards the request to the communication controller CC_1. The designated FIFO responds to the traffic controller whether the FIFO is empty or not. If the FIFO is not empty, the traffic controller generates control signals to establish a communication line.

3.2 Details of custom switches

We exploit the fact that the logical topology is represented by a task graph (KPN), in which each node has possibly a different number of incoming and outgoing links. In this work, a parameterised multiplexer array has been implemented for switch modules as a design technique. Topology-specific and different-sized multiplexers ensure that demanding interconnects are established. In our work, a topology table is systematically extracted from the application specification, such that any topology can be systematically implemented from single node to all-to-all topology. The custom crossbar interconnects are implemented with the following two steps. First, the graph topology is extracted from the KPN specifications. Each processor port has a set of incoming and outgoing links, as depicted in Fig. 4. Table A indicates the number of incoming links and a list of ports from which the links are originated (see Fig. 3*a*). As an example, port p_2' has one incoming link from port p_1, indicating that processor P_2 can possibly read the data located in the FIFOs connected to port p_1. Table B indicates the number of outgoing links and a list of ports to which the links are directed. As an example, port p_1 has two outgoing links to ports p_1' and p_2', indicating that either processor P_1 or P_2 reads the data located in the FIFOs connected to port p_1. Table A and B are used to systematically implement customised multiplexer arrays instead of full multiplexers. There are two types of multiplexers, namely processor-side multiplexers and FIFO-side multiplexers, as depicted in Fig. 4. Table A is used to implement processor-side multiplexers controlled by CTRL_FIFO signals and Table B is used to implement FIFO-side multiplexers controlled by CTRL_PROC signals.

Table A : Incoming links

Port	*NUM_LINK*	*LIST*
p_1'	2	p_1, p_4
p_2'	1	p_1
p_3'	1	p_2
p_4'	1	p_3

Table B: Outgoing links

Port	*NUM_LINK*	*LIST*
p_1	2	p_1', p_2'
p_2	1	p_3'
p_3	1	p_4'
p_4	1	p_1'

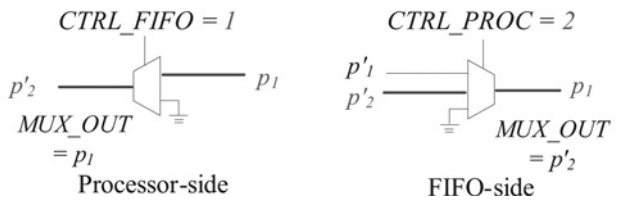


Fig. 4 *Parameterised switch for MJPEG{4,5} topology in Fig. 3*

Second, the two tables described above are passed to a VHSIC hardware description language (VHDL) function as static parameters to actually establish a circuit link. The function to generate parameterised multiplexers has been implemented with a simple priority encoder. Once the request is given the priority, two cycles are required to establish a circuit link to the designated target port. Once a link is established, a remote memory behaves as a local memory until the link is cleared. It can be noted that an *N*-port full crossbar contains *N*-way full multiplexers per port, while our network contains variable-way multiplexers per port, depending on the graph topology. In this way, the implemented custom crossbar provides significantly reduced area without performance degradation.

It can be noted that our custom crossbars constitute overlay functional interconnects on top of underlying reconfigurable FPGA fabrics. In this work, the interconnect is customised at compile time. The design flow instantiates on-demand interconnects, based on the extracted traffic information. The interconnect architecture uses the reconfigurability of the FPGAs which is an inherent part of the design flow.

3.3 Implementations

In this section, we present implementation results of the customised crossbars. We collected task graphs of realistic applications from the five-node H.264 application to the 40-node audio–video (AV) application [10–21]. Assuming that a single node is associated with a single crossbar port, the networks are synthesised, placed and routed using the Xilinx ISE tool on Virtex-II Pro (xc2vp100-6-1704) target FPGA. We targeted a Xilinx Virtex-II Pro device while any reconfigurable hardware can be a targeted device. The other state-of-the-art FPGA devices (that supports a partial reconfiguration) have reduced configuration frame size and increased logic density. However, these devices have minor architectural difference. The area cost of the functional plane is typically represented by the occupied logic slices required by an application. Fig. 5 depicts the area for different topologies. As a result, our custom crossbar occupies on average 67% less area than the full crossbars. The area of our network is not only dependent on the number of nodes that determines the network size but also on the network topology.

3.4 Integration in the ESPAM framework and prototype

Our custom crossbar has been integrated as a modular communication component in the `ESPAM` tool chain as depicted in Fig. 6, in which the MJPEG data flow specification in Fig. 2*c* is considered as an example. In `ESPAM`, three input specifications are required, namely application/mapping/platform specification in XML. A KPN application specification is automatically generated from a sequential Matlab program using the `COMPAAN` tool [22]. Each process is assigned to a specific processor in the mapping specification. The number of processors, the type of network and the port mapping information are specified in the platform specification. Fig. 6 depicts how the customised interconnects can be implemented from the four-node MJPEG application specification. In the platform specification, four processors are port-mapped on a crossbar. From the mapping and platform specifications, port-mapped logical network topology is extracted (using the `Yapi` profiler [23]) as a static parameter and passed to

© The Institution of Engineering and Technology 2012

doi: 10.1049/iet-cdt.2010.0105

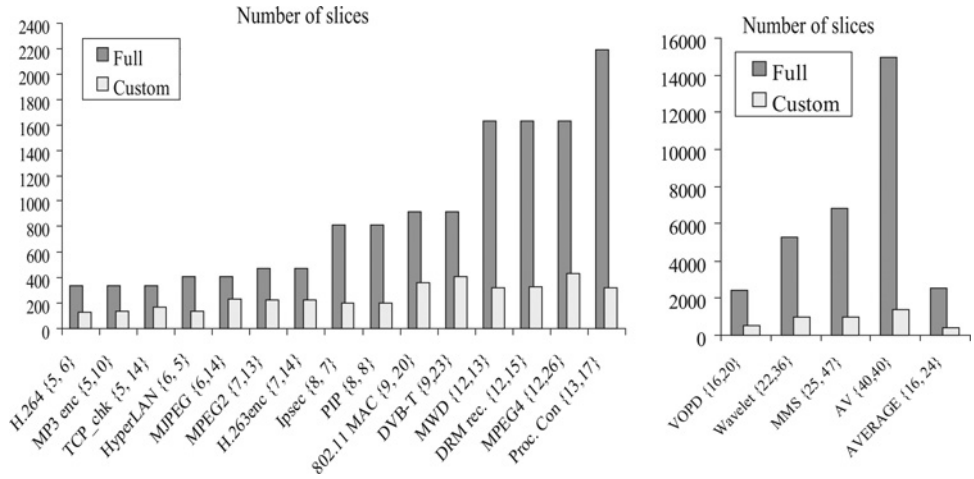


Fig. 5 *Implementation results of custom crossbar switches*

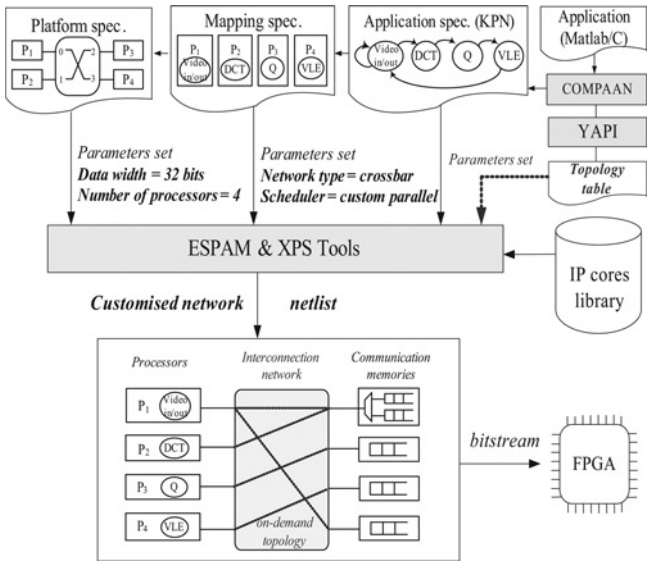


Fig. 6 *Integration of custom crossbar in the ESPAM [2, 3] design flow*

ESPAM. Subsequently, ESPAM refines the abstract platform model to an elaborate parameterised RTL (hardware) and C/C++ (software) models. Finally, the commercial Xilinx XPS tool generates the netlist with regard to the parameters passed from the input specifications and generates a bitstream for the FPGA prototype board to check the functionality and the performance.

Using the ESPAM tool chain, an actual system has been prototyped onto the ADM-XPL FPGA board [24]. We experimented on an MJPEG encoder application that operates on a single image with size 128 × 128 pixels. We have experimented with the three alternative task graphs in Figs. 2*c*, *e* and *f*, where our network provided the on-demand topologies. The implemented system is homogeneous in that each node contains a 32-bit MicroBlaze processor. Fig. 7 depicts the prototype results, in terms of performance and power consumption. Fig. 7*a* depicts the system cycles for different topologies. The system cycles decrease as the number of processors increase. When the system incorporates six-node network, the system performs 4.6 times better than a sequential processing. It can be observed that the topology plays an important role for the system performance. Fig. 7*b* depicts a comparison of power consumption. We use XPower tool [25] to measure the dynamic power consumption. As a result, the custom crossbar reduces the power by 71% compared to the full crossbar. This is because of the fact that the signal switching activity occurs only for on-demand interconnects. In addition, the capacitive load of the custom switch is much less than the loads of the full switch.

In this experiment, we compared the custom crossbar and full crossbar. In [2, 3], it is reported that the *ad hoc* point-to-point interconnects perform better than the crossbar for a particular application. It can be noted that the *ad hoc* point-to-point interconnects are implemented for only a particular single application. For different applications, the *ad hoc* interconnects should be (manually) re-designed. However, our customisation methodology is relatively more systematic such that the customised crossbar is generated for any application, without changing the crossbar implementation itself.

4 Customised circuit-switched NoC

As described earlier, when the general-purpose NoC is mapped onto FPGA fabrics, the main drawback is the increased area cost in the functional layer and the increased configuration overheads. In this work, considering Æthereal NoC [26] as an example, we apply the topology customisation technique for NoCs. The guaranteed throughput Æthereal NoC operates as a virtual single-hop crossbar with a physically pipelined transmission [26]. In this section, to reduce the high area cost in the general-purpose NoCs, we present a customised circuit-switched NoC (CCSN) targeting the reconfigurable fabric.

4.1 Customisation method and analysis

Our main method is to establish only necessary network resources. If two modules do not communicate, the methodology removes the corresponding *I*/*O* ports of the switch, the buffers and the corresponding links. In fact, logic-synthesis tool finds unconnected ports and optimises the un-connected resources away. It can be noted that logic synthesis tool automatically optimises them away only when the un-connectivity is properly specified. We present how to specify the un-connectivity (using the uni-directional topology information) to guide the logic synthesis tool to optimise away the un-connected resources.

© The Institution of Engineering and Technology 2012

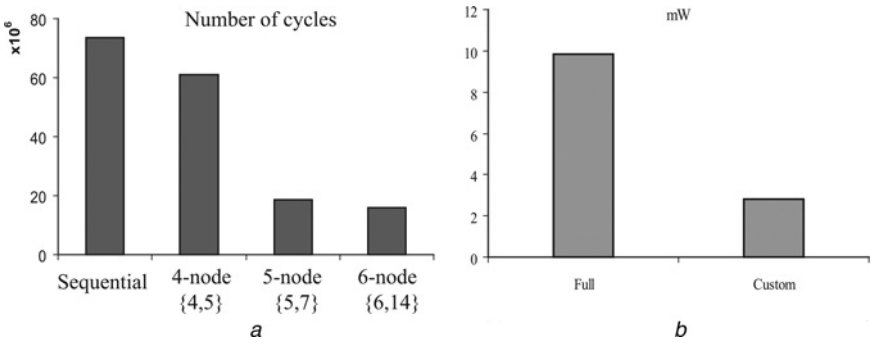


Fig. 7 *Prototype result of custom crossbars for MJPEG applications*

a Performance of different-sized systems
b Power consumption of different crossbars for 6-node MJPEG {6,14} application

The presented technique can be generally applied to any type of network including circuit-switched NoC with any topology. Unless the uni-directional topology information is properly specified, the network resources will be established after the logic synthesis step, even when the established network resources are not utilised by an application.

We consider the non-customised two-dimensional (2D)-mesh circuit-switched-network (CSN) as a reference. Fig. 8 depicts an example. Fig. 8*a* depicts a logical topology of a five-node MJPEG application. Fig. 8*b* depicts an embedding of the logical topology on the 2 × 3 2D-mesh topology. The logical topology is embedded in a way that the dilation is minimised. Subsequently, the maximum dilation for the MJPEG application is 2, (e.g. from P_1 to P_3). Fig. 8*c* depicts the corresponding physical 2D-mesh CSN after the topology embedding, where each tile corresponds to the router-IP pair. General-purpose M × N 2D-mesh router network has $(6MN - 2M - 2N)$ inter-router half-duplex links. A router internally accommodates network resources such as buffers, intra-router links and associated control logic. We define the switch wires inside a router as the intra-router links. As an example, the three-port general-purpose router R_5 in Fig. 8*c* establishes six links around the router, internally three buffers, $9(= 3 \times 3)$ intra-router links. In this way, total amount of network resources are calculated. Table 1 shows the number of network resources for different-sized 2D-mesh general-purpose router networks. Note that the topology is customised for both request and response channels.

The CCSN is obtained using the table-based customisation technique as described in the following. A logical connection consists of two channels, a request and a response channel. We assume that the XY-routing is used for both of the request and response channels. First, the path utilisation table is extracted for each port in router instances after the topology embedding. Fig. 9*a* depicts the connectivity graph and the corresponding topology table. The topology table contains the utilised connectivity information. As an example, the local port in the router R_5 is connected to two ports, specifically port index 1 and 3. This is represented by (2, 1, 3, 0, 0, 0). The router R_5 requires totally three intra-router links (south → local, local → west, local → south

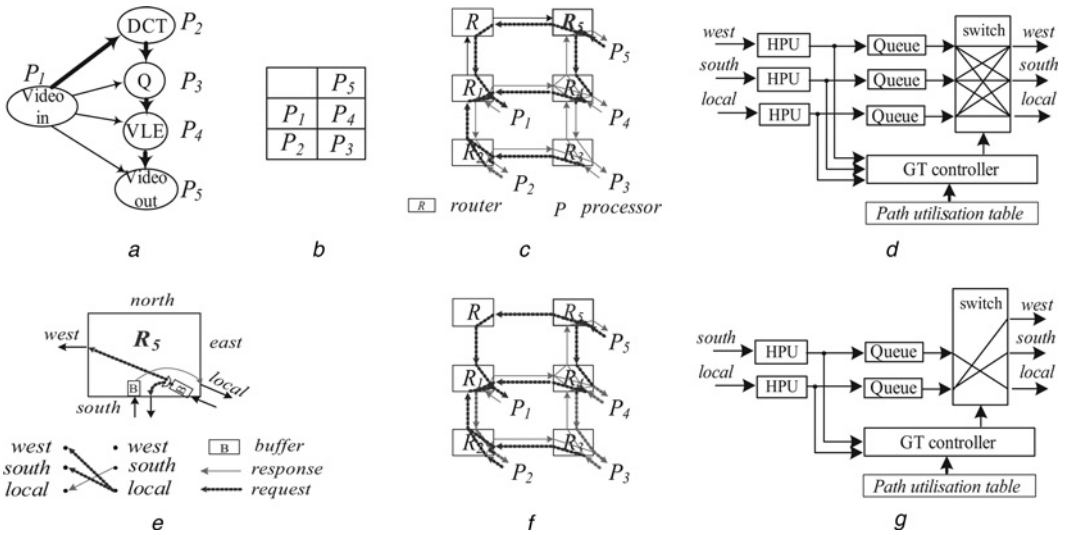


Fig. 8 *Customisation of circuit-switched NoCs*

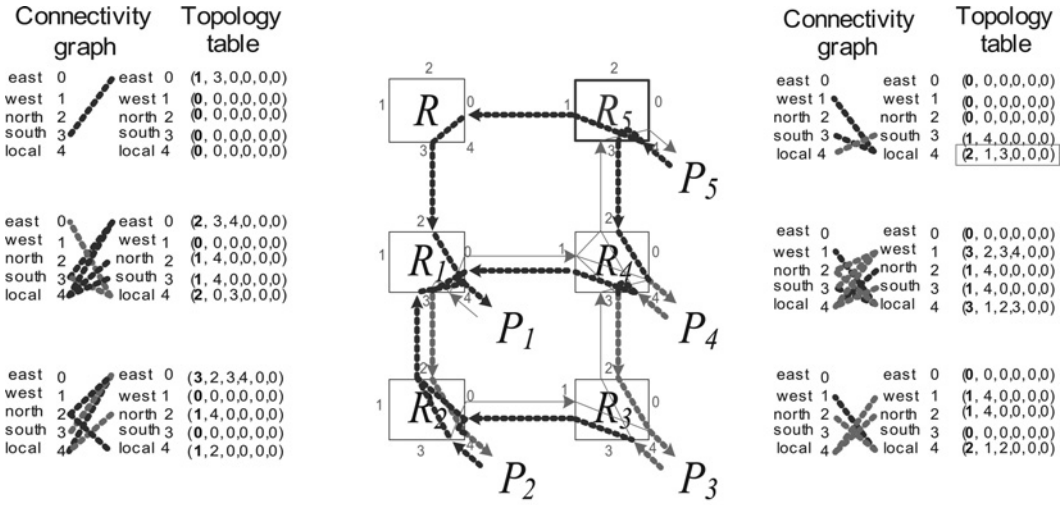
a Logical topology
b Topology embedding
c Topology-embedded physical network
d Router R5 before customisation
e Customising router R5
f Physical network after topology customisation
g Router R5 after customisation

© The Institution of Engineering and Technology 2012

doi: 10.1049/iet-cdt.2010.0105

Table 1 Resources in general-purpose 2D-mesh router network

2D-mesh		2 × 3	3 × 3	3 × 4	4 × 4	5 × 5
number of routers	three-port	4	4	4	4	4
	four-port	2	4	6	8	12
	five-port	0	1	2	4	9
inter-router links		26	42	58	80	130
intra-router links		68	125	182	264	453
number of buffers		20	33	46	64	105




```
type PortArrayType is array(0 to NPORT) of integer range0 to NPORT;
type MuxTableRecordType is record
  RequiredMuxIn: PortArrayType;              /* required MUX input*/
  SelectedMuxIn: integer range0 to NPORT-1;  /* mux select*/
end record
type TableArrayRecordType is array(0 to NPORT-1) of MuxTableRecordType;

/* Topology table for a router*/
ROUTER: for I in0 to NPORT-1 generate
   TableRouter(I).RequiredMuxIn <= TableAllRouters(RouterIndex)(I);  /* TableRouter: TableArrayRecordType*/
   TableRouter(I).SelectedMuxIn  <= MuxSel(I);                       /* TableAllRouters: Topology table for entire network*/
end generate;

 /* Custom switch */
CUSTOMMUX : for i in 0 to (NPORT-1) generate
  mux_out(i)      <= mux_in(GenerateCustomMux(TableRouter(i)))
end generate;

/* Function to generate custom MUX */
function GenerateCustomMux( MUXTABLE: MuxTableRecordType) return integer is
variable SELECTED  : integer range0 to NPORT;
variable NUMINPUT : integer range0 to NPORT;
begin
    NUMINPUT  := MUXTABLE.RequiredMuxIn(0);
    SELECTED  := 0;
    if ( MUXTABLE.RequiredMuxIn(0) = 0 or MUXTABLE.SelectedMuxIn = 0 ) then SELECTED:= 0;
    else
      for I in 1 to NUMINPUT loop
        if ( MUXTABLE.RequiredMuxIn(I) = MUXTABLE.SelectedMuxIn ) then SELECTED := MUXTABLE.RequiredMuxIn(I);
      end loop;
   return SELECTED;
end GenerateCustomMux;
```

b

Fig. 9 *Customisation method*

a Topology table
b Implementation

links), instead of 3^2 links. In this way, the table for the entire network is represented as the 3D array. Second, unnecessary network resources are eliminated in each router using the extracted topology table. Fig. 9*b* depicts how the multiplexer is customised. For a single multiplexer, we define a record type (MuxTableRecordType) that contains required an input port list and a selected input. Additionally, we implement a function GenerateCustom Mux that takes the (record type) topology table and returns a selected input. The customised multiplexers are instantiated using the generate statements in VHDL. As an example, Fig. 8*d* depicts the customised router R_5. Moreover, unnecessary buffers are also eliminated. As an example, two buffers are required for the south, local ports in the router R_5. Figs. 8*f* and *g* depict how the Æthereal router architecture is customised for the router R_5 using the topology table.

© The Institution of Engineering and Technology 2012

Table 2 Comparison between CSN and CCSN for MJPEG {5,7} topology

2 × 3 2D-mesh	CSN	CCSN	Utilisation (%)
inter-router links	26	22	85
intra-router links	68	27	40
number of buffers	20	17	85

Consequently, the CCSN is derived and depicted in Fig. 8*e*. Furthermore, the unnecessary inter-router links and the associated control logic are also eliminated, because these resources are logically/physically disconnected. As an example, one inter-router link to the west port is optimised away. Table 2 shows the utilised resources of the CCSN for the MJPEG{5,7} topologies. This indicates that the CCSN utilises significantly less resources of the general-purpose CSN.

Similar to this case study, we obtained the CCSN from the CSN for the benchmark topologies [10–21]. To do this, logical topologies with n-nodes are embedded onto $\lceil\sqrt{n}\rceil \times \lceil\sqrt{n}\rceil$ 2D-mesh. As a result, on average 70% of buffers, 28% of the intra-router links and 70% of inter-router links are utilised by the applications as depicted in Fig. 10. It can be noted that our customisation method can be applied to the general CSN with any topology. Our topology construction method is similar to [9] in that the switch wires are customised for individual routers. In [9], parameters are specified for an individual multiplexer instance and an arbiter instance. In our work, a topology table is extracted from the topology embedding. Our method also customises the intra-router buffers as well as inter-router half-duplex links. In other words, our table-based customisation allows the systematic removal of un-utilised buffers and interconnects in an entire multi-hop NoC.

In many cases, NoC is used for flexibility and scalability issue in a general-purpose system. It can be seen that we propose to construct an *ad hoc* interconnection starting from an NoC paradigm. Additionally, the proposed method can be well adapted for data-oriented application. If no additional path can be used in NoC owing to the simplification, we might consider that it would be better to use a dedicated interconnect. However, even though NoC is simplified (such that un-utilised NoC resources are not established), we consider that maintaining an NoC paradigm is important because a dedicated point-to-point interconnects become limited as technology scales and system becomes complex. The complexity of NoC can be reduced using the presented technique, even when applications are managed for hundreds of components as far as there are network resources which are un-utilised by the traffic. Either in the general-purpose system and application-specific system, there can be un-utilised network resources provided that the routing paths for traffics are typically statically determined.

We additionally conduct an analysis of the cost reductions (or the marginal cost). We define the marginal cost as the cost of the additional inputs needed to produce additional unit of output. As described earlier, the un-connectivity is represented using the uni-directional topology table. In our method, the topology table is immediately extracted from the application specification step and the actual removal is performed by the logic synthesis tool. Therefore there is no additional increase of the cost during the removal procedure. The cost reductions when we remove '1 buffer slot' and '1 port of an output multiplexer' are:

- Assuming that one 32-bit buffer slot is mapped onto 32 look-up-tables (LUTs) in an FPGA, the cost reductions for '1 buffer slot' are (i) 32 LUTs that are programmed as 'the buffer slot', (ii) wire segments between those LUTs, (iii) all the network resources that are driven by the LUTs and (iv) all the network resources that drive the LUTs. It can be noted that a single LUT has 1-bit output in our targeted Xilinx Virtex-series FPGAs.
- Assuming that one 32-bit input and N 32-bit output multiplexer is mapped onto $32 \times N$ LUTs, the cost reductions for '1 port of output multiplexer' are (i) 32 LUTs programmed as '1 port of output multiplexer', (ii) wire segments between those LUTs, (iii) all the network resources that are driven by the LUTs and (iv) all the network resources that drive the LUTs.

4.2 Implementation and experiments

Recent multi-core NoCs in the deep-submicron technology occupy less than 15% of chip area [27] even after using significant buffering resources. An NoC is indeed a small part of the chip taking mostly advantage of the ample wire resource. NoCs on FPGA and application specific integration circuit (ASIC) have different implications. In FPGAs, a logical NoC is required to be mapped onto physically fine-grained reconfigurable point-to-point pre-established fabrics [28]. Accordingly, the on-chip networks occupy significant logic and wire in FPGAs [29–31]. It is reported in [30] that FPGA implementations occupy 35 times more area, operate 3.5 times slower and use 14 times

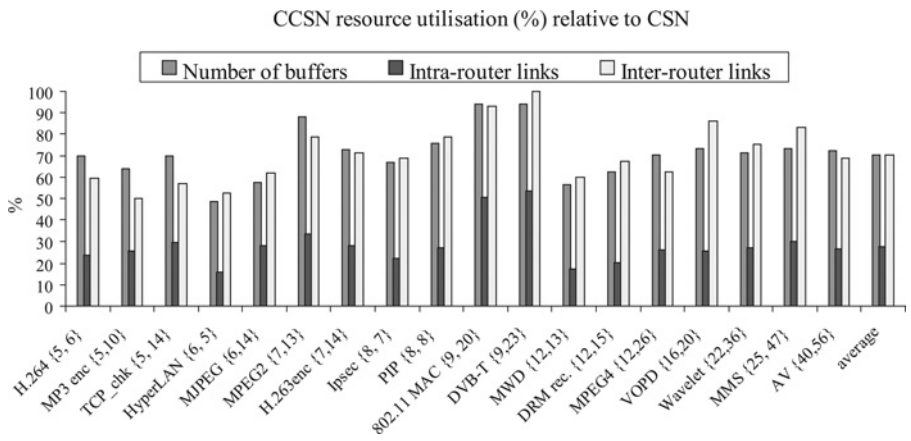


Fig. 10 *Network resource utilisation of CCSN relative to CSN*

© The Institution of Engineering and Technology 2012

doi: 10.1049/iet-cdt.2010.0105

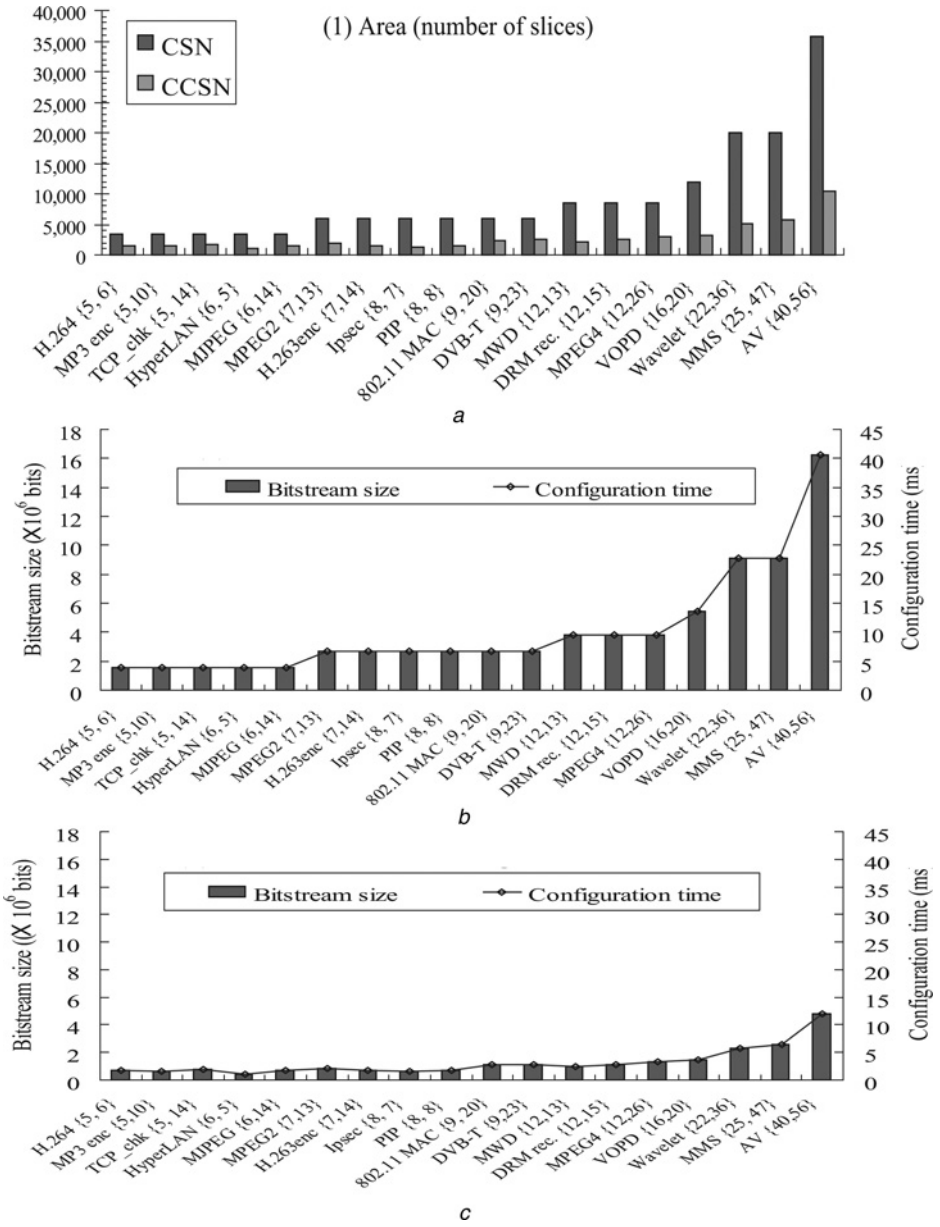


Fig. 11 *Area and configuration overheads of CSN and CCSN in Virtex-II Pro xc2vp100*

a Area (number of slices)
b Bitstream size and configuration time of CSN
c Bitstream size and configuration time of CCSN

more energy. Another study in [31] reports that the average area ratio of 80. We conduct two experiments to analyse the cost of NoCs.

First, to compare the area cost in the functional plane, the CSNs and CCSNs are implemented for the benchmark topologies. Fig. 11*a* depicts the area cost. The CSN occupies significant logic resources. As an example, 3 × 4 2D-mesh CSN occupies 8450 slices and 19% of total logic resources in the xc2v2p100 device, which is the largest Virtex-II Pro device [25]. For comparison, the CCSN for MWD{12,13} occupies 2161 slices and reduces the area by 75% of logic slices. In Fig. 11*a*, the CCSNs reduce the area of the CSNs by 66% on average for the benchmark topologies [10–21].

Second, in the configuration plane, we derive the configuration cost in terms of the bitstream size or the configuration memory size. As discussed in Section 1, the configuration performance and cost can be improved by reducing the functional area cost. To analyse the configuration overhead of the networks, we derive a lower bound configuration time based on the utilised functional area. The configuration time is determined by the required number of frames. The required number of frames varies with placement and routing policies. However, the lower bound of the configuration time can be derived from utilised logic slices. Assuming that the utilised logic slices are maximally packed into each frame, the lower bound of the number of frames can be derived as ⌈(number of utilised slices/number of slices per CLB × number of CLBs per column)⌉ × (number of frames per column). The ceiling operator is used because of the fact that the CLB column is the basic coherent unit for the configuration. As an example, 3 × 4 2D-mesh CSN occupies 8450 slices and requires at least ⌈(8450 slices) four slices per CLB × 80 CLBs per column⌉ × (22 frames per column) = 594 frames. Since a single frame requires 16.5 μs [= (206

© The Institution of Engineering and Technology 2012

words × 32 bits/8 bit interface × 50 MHz)], the configuration time is derived by 594 × 16.5 = 9801 us. The required bitstream size is derived by (number of frames) × (number of bits per frame). A single frame in Virtex-II Pro contains 6592 bits (206 words and each word is 32 bits wide [25]). Therefore the lower bound of the required bitstream size is 3915 648 bits (= 594 frames × 6592 bits). For comparison, the CCSN for MWD{12,13} occupies 2161 slices. Therefore the minimum number of required frames is ⌈(2161 slices/4 slices per CLB × 80 CLBs per column)⌉ × (22 frames per column) = 154 frames. The configuration time is derived by 154 × 16.5 = 2541 μs. The lower bound of the required bitstream size is 1015 168 bits (=154 frames × 6592 bits), which is 74% less than the 3 × 4 2D-mesh CSN. Fig. 11*c* depicts the lower bound of bitstream sizes and the configuration time of the CCSN for our benchmark topologies. As a result, the lower bound of the configuration overheads of CCSN is 66% less than the CSN. It can be noted that the lower bound of the configuration time and the bitstream size are directly proportional to the area cost in the functional plane. As the required resources in the functional plane increases, the required resource in the configuration plane accordingly and proportionally increases. Therefore our experiment indicates that the CCSN can significantly reduce the area cost as well as the configuration overheads compared to the CSN.

5 Conclusions

We presented topologically customised circuit-switched interconnects and the experiments in FPGAs. We showed that the custom crossbar can be implemented using parameterised multiplexer arrays. Multiprocessors interconnected with our custom crossbar were implemented and verified with the automated ESPAM design flow. We extended the topology customisation method to derive a customised circuit-switched NoC. The presented interconnects occupy significantly less area than conventional general-purpose interconnects. Additionally, we presented that the configuration performance and cost can be accordingly improved by reducing the functional area cost. By utilising the logical topology as a parameter, the network is adapted to a given application without modifying the network implementation. Our customised interconnects efficiently utilise the bandwidth by establishing on-demand on-chip resources.

6 References

1 Dally, W.J., Towles, B.: 'Route packets, not wires: on-chip interconnection networks'. Design Automation Conf. (DAC'01), June 2001, pp. 684–689
2 Nikolov, H., Stefanov, T., Deprettere, E.: 'Efficient automated synthesis, programming, and implementation of multi-processor platforms on FPGA chips'. Int. Conf. on Field Programmable Logic and Applications (FPL'06), August 2006, pp. 323–328
3 Nikolov, H., Stefanov, T., Deprettere, E.: 'Multi-processor system design with ESPAM'. Int. Conf. on HW/SW Codesign and System Synthesis (CODES-ISSS'06), October 2006, pp. 211–216
4 Hur, J.Y., Stefanov, T., Wong, S., Vassiliadis, S.: 'Customizing reconfigurable on-chip crossbar scheduler'. IEEE Int. Conf. on Application-Specific Systems, Architectures and Processors (ASAP'07), July 2007, pp. 210–215
5 Hur, J.Y., Stefanov, T., Wong, S., Vassiliadis, S.: 'Systematic customization of on-chip crossbar interconnects'. Int. Workshop on Applied Reconfigurable Computing (ARC'07), March 2007, pp. 61–72
6 Murali, S., Benini, L., De Micheli, G.: 'An application-specific design methodology for on-chip crossbar generation', *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, 2007, **26**, (7), pp. 1283–1296
7 Pasricha, S., Dutt, N., Ben-Romdhane, M.: 'Constraint-driven bus matrix synthesis for MPSoC'. Asia and South Pacific Design Automation Conf. (ASP-DAC'06), January 2006, pp. 30–35
8 Bartic, T.A., Mignolet, J.-Y., Nollet, V., *et al.*: 'Topology adaptive network-on-chip design and implementation', *IEE Proc. Comput. Digit. Tech.*, 2005, **152**, (4), pp. 467–472
9 Meloni, P., Camplani, M., Raffo, L., Carta, S.M., Murali, S., De Micheli, G.: 'Routing aware switch hardware customization for networks on chips'. Int. Conf. on Nano-Networks (Nano-Net'06), September 2006, pp. 1–5
10 Jiang, X., Wolf, W., Henkel, J., Chakradhar, S.: 'H. 264 HDTV decoder using application-specific networks-on-chip'. Int. Conf. on Multimedia and Expo (ICME'05), July 2005, pp. 1508–1511
11 Hu, J., Marculescu, R.: 'Energy-aware mapping for tile-based NoC architectures under performance constraints'. Asia and South Pacific Design Automation Conf. (ASP-DAC'03), January 2003, pp. 233–239
12 Lahiri, K., Raghunathan, A., Lakshminarayana, G., Dey, S.: 'Design of high-performance system-on-chips using communication architecture tuners', *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, 2004, **23**, (5), pp. 620–636
13 Rauwerda, G.K., Heysters, P.M., Smit, G.J.M.: 'Mapping wireless communication algorithms onto a reconfigurable architecture', *J. Supercomput.*, 2004, **30**, (3), pp. 263–282
14 Lee, H.G., Ogras, U.Y., Marculescu, R., Chang, N.: 'Design space exploration and prototyping for on-chip multimedia applications'. Design Automation Conf. (DAC'06), July 2006, pp. 137–142
15 Ramamurthi, V., McCollum, J., Ostler, C., Chatha, K.S.: 'System-level methodology for programming CMP based multi-threaded network processor architectures'. IEEE Computer Society Annual Symp. on VLSI (ISVLSI'05), May 2005, pp. 110–116
16 Bertozzi, D., Jalabert, A., Murali, S., *et al.*: 'NoC synthesis flow for customized domain specific multiprocessor systems-on-chip', *IEEE Trans. Parallel Distrib. Syst.*, 2005, **16**, (2), pp. 113–129
17 Sekar, K., Lahiri, K., Raghunathan, A., Dey, S.: 'FLEXBUS: a high-performance system-on-chip communication architecture with a dynamically configurable topology'. Design Automation Conf. (DAC'05), June 2005, pp. 571–574
18 Bartels, C., Huisken, J., Goossens, K., Groeneveld, P., Meerbergen, J.V.: 'Comparison of an Æthereal network on chip and a traditional interconnect for a multi-processor DVB-T system on chip'. Int. Conf. on Very Large Scale Integration (VLSI-SoC'06), October 2006, pp. 80–85
19 Wolkotte, P.T., Smit, G.J.M., Smit, L.T.: 'Partitioning of a DRM receiver'. Int. OFDM-Workshop, September 2004, pp. 299–304
20 Murali, S., De Micheli, G.: 'Bandwidth-constrained mapping of cores onto NoC architectures'. Int. Conf. on Design, Automation and Test in Europe (DATE'04), February 2004, pp. 896–901
21 Hu, J., Ogras, U.Y., Marculescu, R.: 'System-level buffer allocation for application-specific networks-on-chip Router design', *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, 2006, **25**, (12), pp. 2919–2933
22 Kienhuis, B., Rijpkema, E., Deprettere, E.: 'Compaan: deriving process networks from Matlab for embedded signal processing architectures'. Int. Workshop on Hardware/Software Codesign (CODES'00), May 2000, pp. 13–17
23 de Kock, E.A., Essink, G., Smit, W.J.M., *et al.*: 'YAPI: application modeling for signal processing systems'. Design Automation Conf. (DAC'00), June 2000, pp. 402–405
24 Alpha Data Parallel Systems, Ltd., http://www.alpha-data.com
25 Xilinx, Inc., http://www.xilinx.com
26 Goossens, K., Dielissen, J., Radulescu, A.: 'The Æthereal network on chip: concepts, architectures, and implementations', *IEEE Des. Test Comput.*, 2005, **22**, (5), pp. 414–421
27 Kim, M.M., Davis, J.D., Oskin, M., Austin, T.: 'Polymorphic on-chip networks'. Int. Symp. on Computer Architecture (ISCA'08), June 2008, pp. 101–112
28 DeHon, A.: 'Reconfigurable architectures for general-purpose computing'. PhD dissertation, Massachusetts Institute of Technology, September 1996
29 Hur, J.Y.: 'Customizing and hardwiring on-chip interconnects in FPGAs'. PhD dissertation, TU Delft, February 2011
30 Kuon, I., Rose, J.: 'Measuring the gap between FPGAs and ASICs', *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, 2007, **26**, (2), pp. 203–215
31 Wilton, S.J.E., Ho, C.H., Leong, P.H.W., Luk, W., Quinton, B.: 'A synthesizable datapath oriented embedded FPGA fabric'. Int. Symp. on Field-Programmable Gate Arrays (FPGA'07), February 2007, pp. 33–41

© The Institution of Engineering and Technology 2012

doi: 10.1049/iet-cdt.2010.0105