A Programmable Interconnection Network for Multiple Communication Patterns

Václav Dvořák, Jiří Jaroš
Dept.of Computer Systems
Faculty of Information Technology BUT,
Brno, Czech Republic
dvorak, jarosjir@fit.vutbr.cz

Abstract–**Application-specific or embedded systems with less than 16 processing cores are too small to use some kind of network on chip (NoC) for interconnection. On the other hand, a crossbar and related circuitry (arbiters, memory elements) are too expensive in terms of chip area. As only few pair-wise and collective communication patterns are mostly used in specific applications, we explore an interconnection network that can support only selected communication patterns and no others. The main contribution of the paper is designing of such networks without routers or arbiters, in a form of programmable combinational logic, with limited crossbar functionality. The interconnection network can be implemented by multiplexers or block RAMs on the FPGA chip at a very low cost. A functional decomposition of the related multiple-output Boolean function into a cascade of block RAM devices is aided by multi-terminal binary decision diagrams and is illustrated on examples.**

***Keywords*– multiprocessor SoCs; programmable interconnection; on-chip interconnects; crossbar switch; logic decomposition; multi-terminal BDDs**

I. INTRODUCTION

Multiprocesor systems-on-chip (MPSoC) consist of multiple, usually heterogenous, processing elements (PEs) with local memory, and I/O components. They are usually targeted for embedded applications such as multimedia, telecommunication architecture, network security, and the like. In the implementation of MPSoC, an on-chip network comes to the forefront because of its impact on the performance of the system. Design of MPSoC relies at present mostly on point-to-point connections rather than on shared buses. Buses are not scalable beyond some limit and may not provide required performance because the available communication bandwidth is shared among all the units connected to the bus. Scalability and reusability were two features that led to the network on chip (NoC) paradigm for on-chip communication [1].

The topologies of choice for NoCs have been ring, mesh, fat tree, crossbar and spidergon [2]. Performance of these networks in pair-wise as well as in collective communications is well understood. The lower bounds for time complexity of collective communications are known and can be reached in some cases by optimum scheduling of communications [3].

Communication operations can be either point-to-point, with one source and one destination, or collective, with more than two participating processes. Collective communications (CCs) are invoked by nodes to distribute, gather, and exchange data. Some embedded parallel applications, like network or media processors, are characterized by independent data streams or by a small amount of inter-process communications [1]. However, many special-purpose parallel applications display a bulk synchronous processing (BSP) behavior: the processing nodes access the network according to a global, structured communication pattern.

A collective operation is usually defined in terms of a group of processes. The operation is executed when all processes in the group call the communication routine with matching parameters. We classify collective operations into three types according to their purpose: CCs (One-to-All, OA, All-to-One, AO, All-to-All, AA), global computation (reduction AOR or AAR and scan) and synchronization (barrier). The CCs are most important, as other collective operations are closely related to them. In a broadcast (OAB), one process sends the same message to every group member, whereas in a scatter (OAS), one process sends a different message to each member. Gather (AOG) is the dual operation of scatter, in that one process receives a message from each group member. These basic operations can be combined to form more complex operations. In all-to-all broadcast (AAB), every process sends a message to every other group member. In complete exchange, also referred to as all-to-all scatter-gather (AAS), every group member sends a different message to every other group member. Permutations, and partial permutations (i.e., permutations in which some source to destination pairs

Copyright (c) IARIA, 2011 ISBN:978-1-61208-114-4

are missing) are important CCs that can be used as building blocks to create more complex all-to-all CCs. Since complexities of some communications are similar (AOG ~ OAS, AOR ~ OAB, AAR ~ AAB), we will focus only on 4 basic types (OAB, OAS, AAB, AAS).

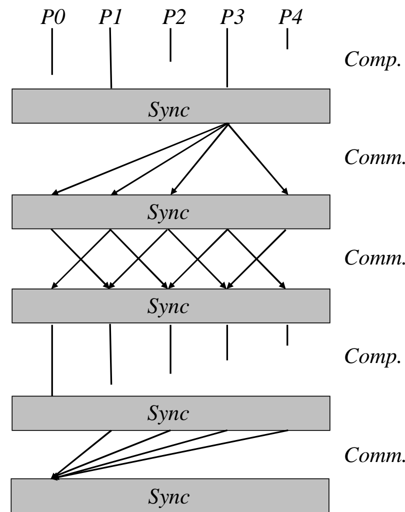

Figure 1. BSP algorithm with five supersteps executed on five processors.

The distributed-memory BSP model proposed by Valiant is essential to our discussion. It comprises computer architecture, a class of algorithms, and a function for charging costs to algorithms [6]. A BSP computer consists of a collection of processors, each with private memory, and a communication network that allows processors to access memories of other processors. A BSP algorithm consists of a sequence of super-steps, which contain a number of either computation steps or communication steps, followed by the global barrier synchronization. In a computation super-step, each processor performs a sequence of operations on local data. In a communication super-step, each processor sends and receives a number of messages (Figure1).

The time, or cost, of a communication super-step is

$$T_{comm} = hg + l, \quad (1)$$

where each processor sends/receives to/from other processors h or less data words, g is the time needed to transfer one data word under the continuous message traffic, and l is a fixed overhead (latency) of communication and of global synchronization.

Further on we will assume that application-specific systems of interest in this paper are compatible with the BSP model described above. In the following Section II, we map the state of the art in application-specific interconnects and formulate the problem to be dealt with. The main results are in Sections III. We first consider complexity of different implementations of a crossbar network and arrive at its low-cost, application-specific version in Section IIIA, and then present logic design of a programmable interconnect on a small example in Section IIIB. Synthesis of the application-specific NoC for multiple communication patterns is done on a small set of patterns in Section IV. The presented technique is assessed in Conclusions.

II. STATE OF THE ART AND THE PROBLEM STATEMENT

In the context of application-specific MPSoCs, communication architecture, more often than not does not have to support *all* pair-wise communications and efficient implementation of *all* collective communications such as broadcast, multicast, gather, scatter, and others. Quite a few applications running on MPSoCs use only a limited set of deterministic communication patterns and a general NoC infrastructure suitable for general-purpose computing is not needed. By taking advantage of the known application communication behavior, special-purpose networks may be designed for well-behaved communication requirements, resulting in networks that are more resource/ performance effective. We will therefore try to simplify the interconnection network by providing a support just for the required communication patterns.

The application-specific optimization of interconnection network with respect to performance and power consumption can be obtained by removal of some links. For example, the spidergon topology is opened for such optimization [4]. A design methodology using a recursive bisection technique for generating optimum topology for applications with well-behaved communication patterns has been introduced in [5]. In contrast, our approach treats the interconnection as combinational logic and thus covers not only the single instance of a problem, but a wide class of applications with limited number of CPU cores and with limited communication needs. The starting point is a non-blocking crossbar, which is the ideal on-chip interconnect, but it is prohibitively expensive for larger MPSoCs.

Beside static networks, it may also be of interest to allow the network to be reconfigurable, at run-time. For example, reconfigurable computing paradigms (e.g., FPGAs) have increasingly become more practical alternatives recently. Field-programmable interconnect devices (FPIDs), acting as SRAM-based switching

Copyright (c) IARIA, 2011 ISBN:978-1-61208-114-4

matrices, can be reconfigured dynamically, in the same way as standard SRAM-based FPGAs – by means of SRAM cells controlling switching elements (pass transistors or transmission gates). Reconfiguration times are therefore much too long and prevent dynamic reconfiguration before each communication pattern.

The problem to be addressed in this paper is to find as simple message-passing communication structure as possible for MPSoC with a small number (≤ 16) of PEs and with a set of deterministic communication patterns customized for a specific application. Unidirectional, one bit wide links will be assumed.

III. MAIN RESULTS

We will focus on interconnect programmable in run-time, with performance comparable to a crossbar. For the fastest operation, the control of a programmable intercon-nection network must come from inside the chip, e.g. from the master PE. Such in-system reconfiguration can achieve some of the flexibility of software with the performance of dedicated hardware. This idea is elaborated further on in two following sub-sections.

A. Crossbar Implementation

For multiprocessor SoCs of smaller size, a crossbar (Xbar) switch is utilized for an efficient on-chip network solution, Figure 2. Arbiters provided for each X-bar output select one request from all coming in and set the switch to the appropriate position. There is a CAD tool for generation of round-robin arbitration and $N\times M$ X-bar switch logic for MPSoCs [7] based on user specifications. To evaluate the hardware complexity, two possible implementations have to be considered.

A crosspoint-based implementation of a square $N\times N$ crossbar makes use of N columns of N crosspoints realized by transmission gates, pass transistors or tri-state inverters. A crosspoint-based crossbar implies a memory element per crosspoint (N^2 elements altogether) that lets a row signal be propagated or not to some column. Contents of memory elements are determined by (round-robin) arbiters, one per column.

A second approach to implement the crossbars is to use logic multiplexers. A full $N\times N$ crossbar requires N N-input multiplexers and N arbiters (one per each multiplexer). As $\log_2 N$ memory elements are required for each column multiplexer, a total of $N\log_2 N$ configuration memory elements are needed. This approach thus leads to a significant reduction in the number of memory elements.

As regards a support for pair-wise and collective communication, the X-bar in Figure 2 is able to implement any permutation of inputs or its subset, broadcast from any node, multicast, and also several non-conflicting parallel multicasts. Whereas a source PE can send up messages to all piers in parallel, the target can receive only a single message. Multiple messages targeted for a single PE are filtered out by arbiters.

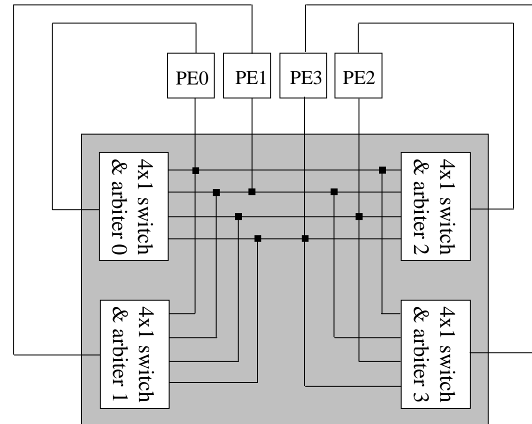

Figure 2. The example of 4×4 X-bar with four processors

The above limitation determines the number of communication steps needed for typical collective communications. Broadcast and multicasts need only a single step, gather (and reduction operations) require $\log_2 N$ communication steps. All-to-all communica-tions such as broadcast or scatter are implemented as a sequence of N-1 permutations.

If only a specific set of pair-wise and collective communications is needed in a certain BSP application, we can simplify the X-bar in Figure 2 a great deal. We can get rid of arbiters entirely and use switches of size $P\times 1$, where P is the number of communication patterns. All switches have identical control, so that the number of control signals is typically much lower than that in ordinary $N\times N$ crossbars,

$$\log_2 P < N \log_2 N. \quad (2)$$

Multiplexers with 16 to 32 data inputs are quite common and the number of required communication patterns may nicely fit into this range.

B. Logic Design of a Programmable Interconnect

The programmable interconnect can be implemented as a network of programmable switches. The switch itself is a logic device that can connect some or all inputs, one-to-one, to some or all outputs. Multiplexer and crossbar switches are building blocks of more complex programmable interconnection networks. E.g., an elementary 2×2 crossbar is used in

Copyright (c) IARIA, 2011 ISBN:978-1-61208-114-4

multi-stage interconnection networks that are much cheaper than crossbars.

We will illustrate logic design of programmable interconnect in more detail on a class of programmable Bit-Masking and Shifting interconnect devices with 4 to 8 inputs (BMS4 to BMS8). These units are useful when implementing multi-way branching in micro-programmed controllers; they enable efficient allocation of microcode memory to a cluster of multi-way dispatch tables [8]. For example, the task of the 16-way BMS4 is to shift 4 or less active inputs, selected by a 4-bit mask, to the lowest positions of the 4-bit output vector and reset the rest of outputs. The output vector then serves as an offset from the base address of a dispatch table; this way the dispatch tables of various size can be stored in control memory in a compact form.

Cube specification of BMS units has been generated automatically. For example, the BMS4 function was specified by 81 cubes:

```
.i 8
.o 4
.ilb m0 m1 m2 m3 x0 x1 x2 x3
.ob y0 y1 y2 y3
.type fr
.p 81
0000---- 0000
0001---0 0000
0001---1 1000
0010--0- 0000
0010--1- 1000
   …
   …
11111011 1011
11110111 0111
11111111 1111
.e
```

Synthesis of the related combinational logic for the 4-input BMS can be done in several ways. The simplest solution would be a single 256 × 4-bit look-up tables LUTs (ROM or Block RAM). However, if only smaller LUTs were available, we can decompose the single LUT into a cascade of two or more smaller LUTs. The latency will increase, but with possible pipeline operation the throughput will remain the same. Various decompositions are easily found from a Multi-Terminal BDD (MTBDD) representing the BMS4 function. The optimal variable ordering of the MTBDD can be found e.g., by the Heuristic Iterative Decomposition Tool HIDET [9] and is shown in Figure 3. The suitable cut of the MTBDD generates a cascade of two LUTs (Figure 4a) with the resulting capacity in bits less than a half of the single LUT capacity.

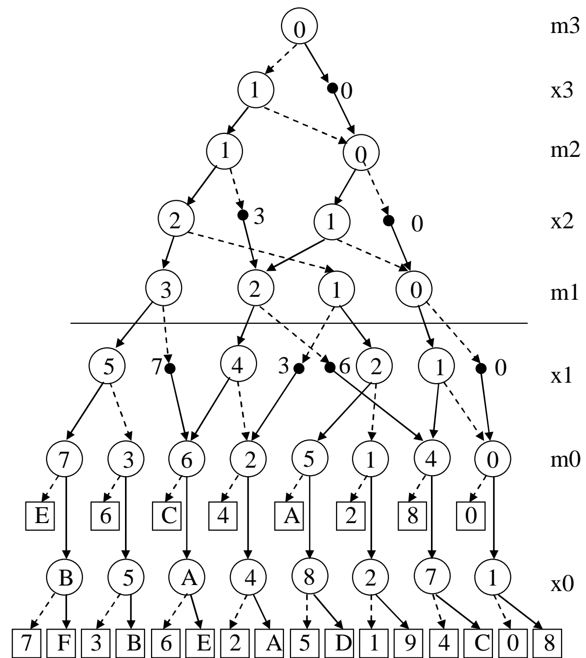

Figure 3. MTBDD of the 4-input BMS.

The traditional design of BMS module in a form of multiplexer network is shown in Figure 4b. Comparing both designs, the LUT cascade wins in the area size devoted to interconnections and in flexibility to implement other communication patterns. As for the performance, delays few ns per stage have been demonstrated for an experimental LUT cascade [10]. Note that the ordering of variables found by HIDET is the same as the optimal ordering of traditional design in Figure 4b.

Parameters of logic design obtained by Xilinx FPGA synthesis tool for BMSs with 4 to 8 inputs and parameters of MTBDDs obtained by the HIDET are shown in Table 1. The local LUT cascade width x relates logarithmically to the local values of MTBDD width w ($x = \lceil \log_2 w \rceil$) between neighbor LUTs. BMS units could be implemented as a cascade of LUTs eliminating messy wiring and reducing chip area for the interconnect. The delay of such switch-boxes is adjustable by the cascade length. If we take the delay of FPGA's 4-input LUTs plus wiring delay approximately equal to cascaded LUTs´ delay, we should use not more LUTs in a cascade than it is given in FPGA column "levels" for the same or better performance. Note that communication from PE's outputs to PE's inputs is now supported by regular wiring from addresses to data outputs of multi-bit memory modules and by external regular wiring among these modules.

Copyright (c) IARIA, 2011 ISBN:978-1-61208-114-4

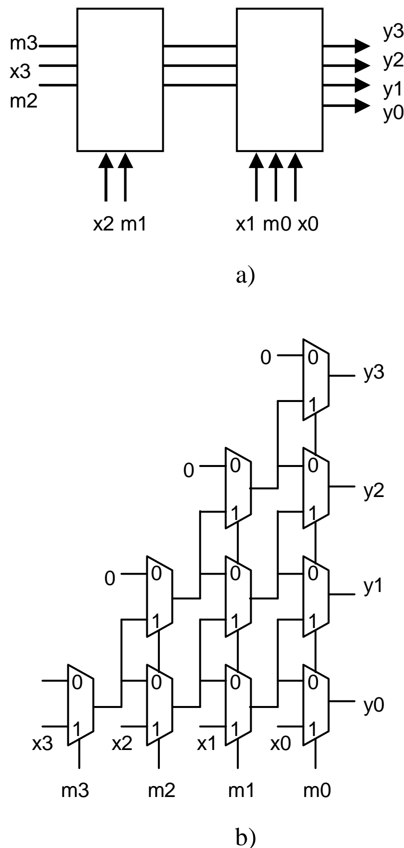

Figure 4. Implementation of the 4-input BMS
a) as the 2-LUT cascade b) as a MPX network

TABLE I. PARAMETERS OF FPGA AND MTBDD-BASED DESIGNS

	FPGA		MTBDD	
	#4-LUTs	levels	cost	width
BMS4	8	3	30	≤ 4
BMS6	20	5	126	≤ 6
BMS7	32	6	254	≤ 7
BMS8	80	7	447	≤ 8

IV. SYNTHESIS OF APPLICATION-SPECIFIC NOCS FOR MULTIPLE COMMUNICATION PATTERNS

The $N \times N$ multiplexer-based crossbar network without arbiters can also be visualized as a combinational logic network with $N + N\log_2 N$ inputs and N outputs. E.g. for $N = 8$ we get 32 inputs and 8 outputs. Out of 2^{24} programmable configurations, only a negligible fraction may be utilized in a certain MPSoC. The multiplexer-based crossbar as an interconnection network is thus for many MPSoC a luxury. A simpler way how to implement a low cost programmable unidirectional interconnect is to replace individual control of M crossbar multiplexers by identical control. Apparently, the number of multiplexer inputs P will now be determined by the number of required communication patterns. For example, barrel shifters can be implemented as crosspoint-based crossbars with common diagonal control [11] or as multiplexer-based crossbars with one multiplexer per output and a common control [12].

Let us design for illustration the application-specific NoC connecting 8 PEs (labeled 0 to 7) and supporting the following 7 communication patterns (encoded in 3 configuration bits):

1. Broadcast from node 0
2. Cyclic shift from node i to node (i+1)mod 8
3. Cyclic shift from node i to node (i+2)mod 8
4. Skew (0↔7, 1↔6, 2↔5, 3↔4)
5. Gather1 (7→6, 5→4, 3→ 2, 1→ 0)
6. Gather2 (6→4, 2→ 0)
7. Gather3 (4→ 0).

A crossbar-like implementation of the above set of communications without arbiters would require eight 8-input wide multiplexers and 8 arbiters. The same network can also be implemented as a regular ROM (multi-bit LUT) with 8+3 = 11 address bits and 8-bit wide output (single Xilinx BRAM block 2048 × 8 bit). The interconnection network is thus embedded in the regular ROM structure. If 2048 × 8 bit is too large an array, and a higher latency is tolerable, we can split it into 2 or more ROMs in a cascade by means of splitting the related MTBDD; one such decomposition is shown in Figure 5.

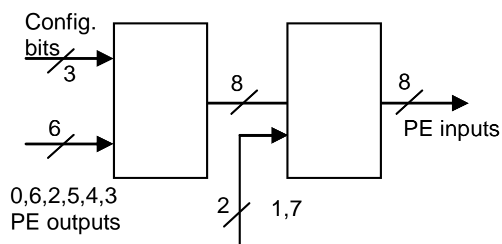

Figure 5. Implementation of the specific interconnection network as the cascade of two multi-bit LUTs

Copyright (c) IARIA, 2011 ISBN:978-1-61208-114-4

Similar cascade decompositions will be handy especially for larger problems (for example, 16 PEs, 32 patterns, that is 21 inputs). The communication bandwidth for larger messages will not be decreased when we use pipelining. The size of partial LUTs and their interconnection are again minimized by the HIDET [9] as in the previous section.

VI. CONCLUSIONS

Application-specific multiprocessor SoCs with a restricted set of deterministic communication patterns and not more than some 16 processing elements can operate with a low cost customized communication network synthesized as a logic subsystem. We assume that applications running on the system are well behaved (i.e., have similar communication patterns on every run) and data independent (i.e., have similar communication patterns for any data set). Contrary to a full crossbar network, $N \times 1$ switches (multiplexers) share control inputs and are not controlled by arbiters, but by one selected PE, for example by the first PE reaching the barrier before a communication step. Thus the arbiters may be eliminated completely. The width of multiplexers is given by the number of communication patterns in the application: one pattern is assigned to one multiplexer input. Instead of reconfiguring topology for a certain application, the communication module is programmed for communication patterns repeatedly in the runtime.

Beside multiplexers, other devices can be used to implement the specialized interconnection network. BRAM devices or cascaded BRAMs are another option. The kind of programmable interconnect suggested in the paper is run-time programmable much faster than FPGAs or FPIDs, because programming is reduced to processing a single store (output) machine instruction to a pattern holding register. Freedom from contention is an additional favorable side-effect of the presented approach.

Performance of the suggested communication module should be comparable to or better than crossbar performance, because of absence of arbitration logic. Prediction of overall overhead of a BSP algorithm is possible by means of (1). Other components of communication architecture like communication protocols or interface design between PEs and a communication module will be a subject of future research.

ACKNOWLEDGMENT

This research has been carried out under the financial support of the research grants GP103/10/1517 “Natural Computing on Unconventional Platforms”, GA 102/09/H042 "Mathematical and Engineering Approaches to Developing Reliable and Secure Concurrent and Distributed Computer Systems", GA102/08/1429 “Safety and security of networked embedded system applications”, MSM 0021630528 “Security-Oriented Research in Information Technology”, and FIT-S-10-1 "Secured, reliable and adaptive computer systems", Brno University of Technology.

REFERENCES

[1] Jantsch, A. and Tenhunen, H. Networks on Chip, Kluwer Academic Publ., Boston, 2003.

[2] Karim, F. and Nguyen, A.: An Interconnect Architecture for Networking Systems on Chips. IEEE Micro, 2002, pp.36-45.

[3] Jaroš J. and Dvořák V.: Evolutionary-Based Conflict-Free Scheduling of Collective Communications on Spidergon NoCs, In: Proceedings of 2010 Genetic and Evolutionary Computation Conference, GECCO 2010, New York, US, ACM, 2010, pp. 1171-1178.

[4] Palermo, G. et al.: Application-Specific Topology Design Customization for STNoC. Proc. of the 10th Euromicro Conference on Digital System Design Architectures, Methods and Tools, DSD 2007, pp.547-550.

[5] Ho, W.H. and Pinkston, T.M.: A Methodology for Designing Efficient On-Chip Interconnects on Well-Behaved Communication Patterns. Proc. Of the 9th Int. Symposium on High Performance Computer Architecture, Anaheim, 2003, pp. 377-388.

[6] Bisseling, R.H.: Parallel Scientific Computation. Oxford Univ. Press, New York, 2004.

[7] Shin, E.S.: Automated generation of round robin arbitration and crossbar switch logic. Ph.D. thesis, School of Electrical and Computer Engineering, Georgia Institute of Technology, November 2003.

[8] Dvořák, V.: LUT Cascade-Based Architectures for High Productivity Embedded Systems, In: International Review on Computers and Software, Vol. 2, No 4, Naples, Italy, pp. 357-365, 2007.

[9] Mikušek P. and Dvořák V.: On Lookup Table Cascade-Based Realizations of Arbiters, In: 11th EUROMICRO Conference on Digital System Design DSD 2008, Parma, IT, IEEE CS, 2008, pp. 795-802.

[10] K. Nakamura, T. Sasao, M. Matsuura, K. Tanaka, K. Yoshizumi, H. Qin, and Y. Iguchi, "Programmable logic device with an 8-stage cascade of 64K-bit asynchronous SRAMs," Cool Chips VIII, IEEE Symposium on Low-Power and High-Speed Chips, April 20-22, 2005, Yokohama, Japan.

[11] Asano, D. K.: Computer Architecture, Shift Circuits. 2001. http://www-comm.cs.shinshu-u.ac.jp/public/comparch/node45.html

[12] Gigliotti, P.: Implementing Barrel Shifters Using Multipliers. Xilinx Application note, XAPP195 (v1.1), 2004.