Synthesis of Low-Overhead Configurable Source Routing Tables for Network Interfaces

Igor Loi[†], Federico Angiolini[†‡], and Luca Benini[†]
[†]DEIS, University of Bologna, Bologna, Italy
[‡]Ecole Polytechnique Federale de Lausanne (EPFL), Lausanne, Switzerland
igor.loi@unibo.it, federico.angiolini@epfl.ch, luca.benini@unibo.it

***Abstract*—In on-chip multiprocessor communication, link failures and dynamically changing application scenarios represent demanding constraints for the provision of suitable Quality of Service. Networks-on-Chip (NoCs) featuring dynamic routing are a known way to tackle these issues, but deadlock freedom and message ordering concerns arise. NoCs with configurable routing, whereby the communication routes are explicitly chosen at runtime out of a set of statically predefined alternatives, provide intelligent adaptation without impacting the consistency of traffic flows.**

However, configurable source routing on a NoC platform requires a design that provides fast path lookup coupled with low area and power consumption. This paper presents an exploration and synthesis approach that, depending on the required amount of routing flexibility, can for example reduce by 3 to 15 times the area cost of the NoC routing tables by adopting partially reprogrammable routing logic instead of fully reprogrammable tables. Further optimizations based on path redundancy allow to reduce up to 17 times the silicon cost.

I. Introduction

Global on-chip communication is becoming a problem as silicon chips become larger, technology scales down, and the clock frequency increases. Signals are predicted to take several clock cycles to travel over the longest distances from corner to corner of a chip [1]. Simultaneously, the increasing performance requirements of highly parallel on-chip architectures are unmet due to the bottlenecks imposed by traditional, bus-based on-chip interconnects.

The Network-on-Chip (NoC) [2], [3] paradigm, which brings packet-switching networking concepts to the on-die level, has been proposed to systematically tackle these challenges. NoCs are a structured, predictable and scalable approach to the problem, centered around wire segmentation and point-to-point signaling.

Generally, cores attached to a NoC do not have any information about the NoC topology - only the destination addresses are known. One of the key functional features of NoCs is providing routing services among these endpoints. In a basic implementation, NoC routing can be extremely simple. For example, several proposed approaches just tag every packet with a routing field in the header. The tagging is done at the network endpoints (Network Interfaces or Network Adapters) by leveraging routing Look-Up Tables (LUTs). The route is deterministic, decided at design time. Yet, this minimal approach is not able to tolerate changes in the operating conditions at runtime, such as:

- Intervening faults (*e.g.* switch or link failures)
- Application switching (*e.g.* task migration or task switching, which may induce critical localized congestion)
- Power management events (*e.g.* power down of a portion of the NoC)

Dynamic routing has often been suggested as the answer to best handle these scenarios. Dynamic routing involves forwarding packets along different paths depending on a choice of decision variables, which are evaluated cycle-by-cycle at runtime. For example, with dynamic routing, packets would automatically find an alternate way around a faulty NoC node. Unfortunately, dynamic routing introduces two major problems: deadlocks and packet ordering. Since every packet may follow a different route, it becomes hard to avoid routing loops, which induce deadlocks. Sequential packets travelling among the same endpoints on different routes may also encounter different congestion, and reach their common destination in swapped order - a condition which is forbidden in several implementations, requiring reordering queues. Therefore, dynamic routing can become impractical in practice.

In this work, we follow an approach in between deterministic and dynamic routing, which we will call *configurable routing* in the remainder of the paper. We acknowledge the importance of adding reconfiguration capabilities into the NoC routing policies, but we adopt an architectural design flow which achieves them without incurring any of the major downsides of dynamic routing. Referring to Figure 1 (left side), we operate by leveraging one of the several proposed approaches for NoC topology and route design [4], [5], [6], [7], [8]. These works specifically focus on the generation of NoCs whereby, depending on application needs, routes are established to provide connectivity among communicating

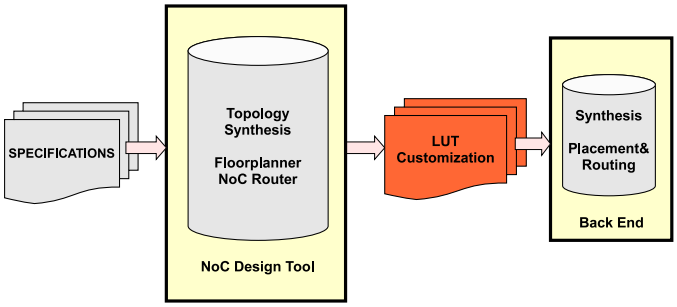

Fig. 1. Reference NoC design flow. The routing customization proposed in this paper operates on given NoC topologies to improve the results upon physical synthesis.

978-3-9810801-5-5/DATE09 © 2009 EDAA

nodes. We assume that multiple sets of routing schemes are available as an input. We then support runtime reconfiguration of the NoC routes among one of the possible alternatives, with the idea of performing a reconfiguration only upon one of the rare events which really demand it (*e.g.* a power down message), and not cycle-by-cycle. The rationale is that, by pre-characterizing a finite set of possible routes, these can be verified to be deadlock-free. Further, packets are again delivered in-order by construction.

A remaining issue with configurable routing is that programmable NoC routing tables can have a prohibitive hardware cost. The main novel goal of this paper is to mitigate this issue. In order to do so, we first observe that configurable routing crucially requires a certain amount of design-time activities - for example, coming up with the static routing sets. Therefore, in this paper, we present a design-time analysis step whose purpose is to identify, out of the various alternatives, the cheapest architecture to support variable amounts of routing reconfigurability, and to further optimize it. As can be seen again in Figure 1 (center), the proposed novelty plugs into existing NoC design flows: given a NoC optimized for a certain application, and namely given a set of possible routes over a topology, our objective is to assess and optimize the cost for rendering those routes into configurable routing LUTs.

In the following, we will present a detailed study of various possible mechanisms to support the routing LUT reconfigurability, keeping in mind the area cost metric. Such mechanisms include RAM- and register-based LUTs, or cheaper solutions whereby the routing LUTs can be only partially reconfigured. Completely static routing is kept as a benchmark. We will also present a comparison of three different synthesis flows of these architectures, with the goal of reaching the most effective physical implementation. Our experiments show that it is critical to pick the right architecture for a given set of requirements. For example, across a wide range of numbers of nodes in the NoC, partially configurable LUTs have shown an excellent trade off between route reconfigurability, area-power cost, and performance. Area savings from 3X to 15X can be achieved compared to fully reconfigurable circuitry, depending on the routing flexibility requirements. A further proposed optimization allows to drastically reduce the programmable elements, additionally saving up to 20% of the area cost in our test case.

II. Related Work

Networks-on-Chips (NoCs) have been proposed by numerous authors [3], [2], [9], [10] as a way to tackle multiple on-chip interconnection challenges of multicore devices, such as scalability to ever larger numbers of IP cores, increasing bandwidth demands, and worsening propagation delays of global on-chip wires.

As NoCs are becoming the focal point of the system integration process, several authors have investigated ways to embed advanced features into them. For example, work has been done on NoC-centered mechanisms for fault tolerance, power management and performance optimization. One of the crucial degrees of freedom in NoC design, which has been leveraged to solve some of the above problems, is routing. Routing is normally categorized as either deterministic (packets follow statically know routes) or adaptive, also called

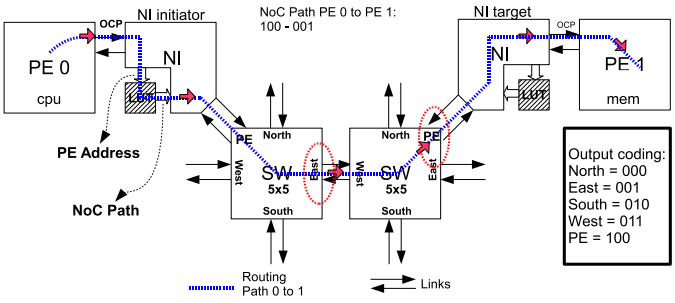

Fig. 2. Example routing mechanism for deterministic source routing NoCs. For each transaction, the Processing Element provides a destination address. This address is then translated into a NoC path, which is merely the ordered sequence of bits representing the codes of the switch ports that packets need to take to reach their destination. The NI uses a LUT to store the route map.

dynamic (packets follow different routes over time depending variables such as congestion states or fault conditions). Deterministic routing is often implemented with lookup tables at the NoC endpoints [11], although in some regular topologies, such as spidergon or mesh, it can be performed in switches based on a destination tag [12], [13]. The work in [14] proposes a mixed approach based on irregular meshes. Dynamic routing needs decentralized decision processes and is therefore often achieved with dedicated logic in every router [15]. Deterministic routing is characterized by its simplicity and minimal overhead; it can easily be configured to avoid deadlocks [16] and natively guarantees in-order delivery. Unfortunately, deterministic routing does not adjust to the system evolution over time; on the contrary, dynamic routing has been proposed to achieve goals such as bypassing faulty nodes and minimizing congestion [17], [15].

Unfortunately, dynamic routing generally induces deadlock conditions, which must be resolved, and, in many implementations, can deliver packets out-of-order, mandating the presence of reordering buffers. These provisions can become impractically expensive on silicon. In [18], a mechanism that compresses lookup tables for adaptive routing has been presented. This solution is however not suitable for irregular topologies. The work in [19] proposes a region-based routing mechanism (adaptive routing) to tackle unreliable hardware in network on chips.

In this paper, in order to support configurable routing instead, first of all we leverage several previous efforts aimed at NoC topology synthesis for a given application [4], [5], [6], [7], [8]. Configurable routing allows bypassing faulty nodes or links, safely shutting down chip regions, and readjusting traffic patterns upon a change in the software application running on the chip. Configurable routing has been proposed in several forms; for example, in [20], the authors propose a custom methodology, based on packet rerouting, to handle data transfers upon power management events or system faults. However, their approach is not general and is actually working around faults in the attached cores, not in the NoC itself. More in general, a novel contribution of this paper is an exploration of the design space for configurable routing implementation.

In this paper, we will explore the area impact of synthesizing routing tables in multiple ways. To this extent, we will use the built-in logic optimizer of industry-standard Synopsys tools [21], as well as BOOM II [22] and ABC [23]. BOOM is a

tool specialized in minimizing multiple-output combinational logic, and generates two-level logic as an output. ABC is more flexible and can handle generic circuits, including sequential and multi-level logic.

III. Configurable Source Routing NoCs

Network Interfaces (NI) seamlessly connect existing IP modules to a Network-on-Chip. They play a crucial role in a NoC context, determining the performance of the whole system. NIs, given a request from the attached processing element, generate packets that will be sent to the destination core, and all the information needed to manage the flow control. As commonly seen, we assume as a reference a NoC with deterministic source routing. For each message coming from the attached core, the NI generates in a deterministic manner the routing bits needed to traverse the NoC switches. An LUT is used for this purpose, converting the memory-mapped address into routing bit sequences. Figure 2 shows the routing mechanism: the core request is processed, and depending on the destination address, the NI generates the path across the NoC switches, up to the destination element. The routing bits are stored in the packet header. When the packet is sent through the NoC switches, a physical channel is created between the packet source and destination.

Depending on how reconfigurable they are, we classify routing LUTs in three categories:

- Hard Wired
- Fully Configurable
- Partially Configurable

As LUT configurability increases, the system becomes more flexible. A highly configurable LUT could for example be reprogrammed to route packets around a large number of NoC faults, while a less configurable LUT may not be able to work around more than one fault, and a hard wired LUT may not tolerate any single faulty link in the NoC. Similarly, more configurability could lead to better performance in a system where many different tasks may be switched over time, *etc.*. Unfortunately, in general, more configurable LUTs are also more expensive in area. This section gives an overview of various possible architectural implementations of NI LUTs. The designer is ultimately in charge of picking one alternative, but our approach makes clear the trade-off among the area cost and the achievable flexibility for the given NoC topology. In the following we explain in detail the possible architectural implementations of the LUTs.

A. Hard Wired

This first solution is presented as a reference. Logically, it is simply a ROM; circuit synthesis tools will actually render it with a netlist of combinational gates. The routing information is permanently stored and no changes are possible at runtime. The address decoder translates the addresses issued by the attached core to properly drive the multiplexer selector. Figure 3a illustrates in detail the logic implementation.

B. Fully Configurable

In this architecture, the routing information is stored on either registers or memory banks. Unlimited remapping of NoC routes is allowed at run time, with maximum routing flexibility. Very different reconfiguration circuits are needed depending on whether the memory elements are rendered as plain flip-flops or whether the designer instantiates a RAM macro instead. The register programming can take a place by injecting a setup vector through a scan chain, depicted in Figure 3b. On the contrary, the RAM-based LUT uses a dedicated data structure, and as depicted in Figure 3c, a serial-to parallel converter is needed to properly initialize the memory. The number of memory elements is in both cases $n * m$, n being the number of destinations (LUT entries) and m being the number of bits required to encode the longest path. Both solutions use a large amount of scan registers - namely $n * m$ for the former and m for the latter - for the programming operations. This also means that several clock cycles are needed to change the route map, during which the NI is forced to stay idle. This high programming latency is, however, consistent with the fact that the LUT reprogramming is expected to happen only upon rare events, such as upon failures or power downs.

C. Partially Configurable

Partially configurable LUTs represent a hybrid solution between the previous schemes. Figure 3d gives an overview of this solution. Up to i NoC paths are allowed for each possible destination. These are hard coded, and through a setting logic block, the desired path is enabled. No memory element is required to store the LUT content itself; however, some flip-flops are still needed as the choice of which route to enable is again performed via a scan chain. Since the scan chain injects only the ID of the desired configuration, and not the entire NoC map connectivity, sequential resources decrease drastically and the total amount of clock cycles needed to reprogram the table is much lower compared to the fully configurable scheme.

IV. Synthesis of Configurable Source Routing Logic

We chose to base our integration effort on the ×pipes [8], which supports arbitrary connectivity, and on its instantiation toolchain [24]. Thus, we can leverage a full design flow up to the layout level, as depicted in Figure 1.

For the sake of simplicity, and without losing generality, we choose a simple and regular NoC topology, a mesh. We assume a full connectivity among the cores: each core communicates with all others. An internal tool outputs LUT description, in Verilog, for each NI. As per the default, the tool creates a code which is interpreted as a hard wired structure, with each reachable endpoint node hard coded as one entry of the LUT.

In order to evaluate the impact of synthesis tools on the cost of routing LUTs, three different synthesis flows have been explored as a contribution of this work:

- Standard synthesis flow with Synopsys tools [21]
- Boolean Optimization with BOOM II [22] before the standard synthesis flow
- And-Inverter Graphs (AIG) Optimization with ABC [23] and GTECH mapping before the standard synthesis flow

In the standard Synopsys synthesis flow, the LUTs come as a Verilog behavioral description, where a parallel switch

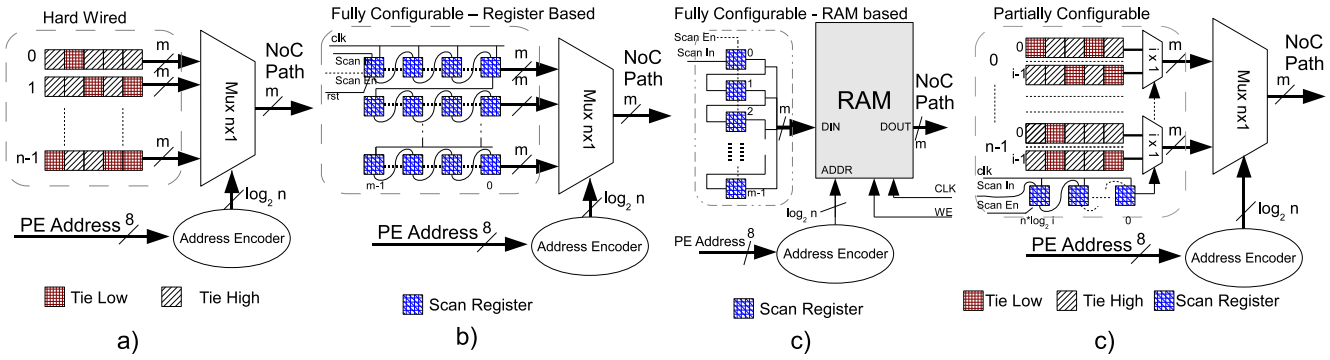

Fig. 3. Different logic implementation of the Source Routing LUT: a) Hard Wired: each route is encoded as hard wired sequence of 0s and 1s; b) Fully Configurable RAM based: each route is stored in a memory cells and can be fully reprogrammed at any time with a scan chain; c) Fully Configurable register based: each route is stored in a register and can be fully reprogrammed at any time with a scan chain; d) Partially Configurable: alternate fixed routes are available for each destination and can be selected by programming a few control bits via a scan chain.

statement implements the LUT logic. Since the NoC paths may show different depth, *don't care* usage may improve the quality of synthesis results. During the synthesis stage, Design Compiler performs a preliminary optimization check by using a PRESTO boolean minimizer.

As the number of paths grows, the LUT increases in complexity, suggesting that a preliminary optimization before the logical synthesis may be useful. This motivates our effort to explore different optimization alternatives, like BOOM II [22] and ABC [23].

BOOM II is a heuristic two-level multiple-output boolean minimizer. As it is compatible with the Berkeley standard PLA format, we use a Perl script to translate the behavioral Verilog description into a PLA file, getting an optimized PLA file as the output of BOOM II. This file is then used for the subsequent synthesis step.

We also try ABC [23], a powerful tool for the synthesis of combinational or sequential logic circuits. With a modification to its generic technology library, ABC can be asked to synthesize the same PLA files that we generate for BOOM II into a netlist of GTECH cells. GTECH is the generic netlist format used by Design Compiler to describe circuits just before mapping them onto a specific technology library, and can therefore be passed as an input to Design Compiler for the final synthesis.

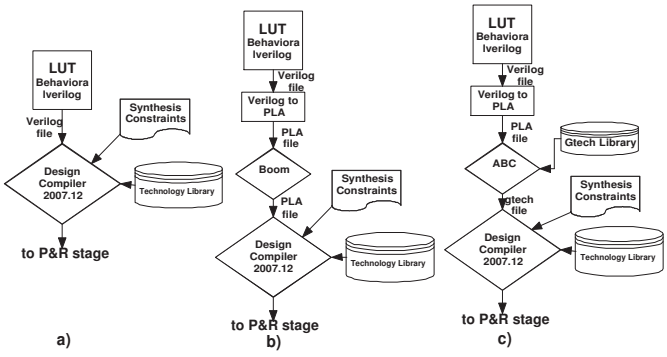

Fig. 4. Three different synthesis flow have been adopted; a) standard synthesis flow; b) Boolean Optimization before the standard synthesis flow; c) And-Inverter Graphs (AIG) Optimization and GTECH mapping before the standard synthesis flow

V. Experimental Results

The proposed solutions have been synthesized and mapped on the UMCE-Faraday 130nm CMOS Technology library, then the Place and Route and the verification have been performed to check the correctness and evaluate the area cost, timing performance and power consumption.

With respect to partially configurable LUTs, we identify two scenarios of interest. The first is the minimum cost one, when only one redundant path per possible NoC destination is provided. The maximum number of redundant entries, on the other hand, is potentially almost unbounded, depending on the topology and on its size. However, due to deadlock freedom constraints, many fewer routing combinations will be valid. If the fault tolerance is factored in, alternate routes should use as disjoint sets of links as possible, further reducing the solution space. It is not the focus of this paper to calculate how many possible route sets may exist in any given topology; for this specific experiment, based on our experience with enumeration in small meshes, we consider at most six such alternate routes. We present experimental results sweeping among these bounds - two to six alternate routes per LUT entry.

Our first experiment is summarized in Figure 5, where we compare the silicon cost, expressed in equivalent gates, of three possible LUT implementations. We assume 14 bits of routing information per LUT entry. The hard wired LUT shows a very efficient area utilization, respectively 18 and 30 times smaller compared to the fully configurable register- and RAM-based LUTs for 6 to 8 LUT entries (the lower bound). This can be explained by the fact that in completely static LUTs the synthesis tool can tap into a huge optimization potential, as the address decoder and the multiplexer can be merged as glue logic. As the number of LUT entries increases, the silicon cost increase in a linear fashion, mainly due to the multiplexer. Concerning the RAM-based solution, four memories have been generated with the UMCE-FARADAY 130nm memory compiler. This tool allows the designer to specify some parameters (word length, number of columns, number of words, *etc.*). We selected four values for the number of words in the LUT - 8, 16, 32 and 64. As expected, RAMs of few words experience a high hardware overhead due to memory handling logic, but this solution is quite efficient when the number of entries grows, since in fact its cost slope is

comparable to the hard wired solution. Finally, the register-based fully configurable LUT is very efficient for small LUTs, but its cost slope is steep. Beyond about 10 LUT entries, this solution shows poor area efficiency as compared to the RAM-based solution.

In many designs, the LUT may fall in the critical path of the Network Interface block, therefore the timing performance of the whole NoC may degrade if the LUT is too slow. Figure 5b summarizes the timing cost: hard wired and register-based solution show the same timing properties, since the latency is dominated by the multiplexer, while in case of a RAM-based LUT the latency is dominated by the memory access time. This cost however grows slowly as compared to the other solutions, confirming the intuitive finding that the RAM usage becomes suitable for very complex designs with large LUTs.

As we introduced in the previous section, we explored three different synthesis flows for partially configurable LUTs. A detailed comparison of the outcome is presented in Figure 6. BOOM II demonstrates clear area savings. Figure 6a illustrates the silicon cost for a fixed latency between input and outputs, while Figure 6b shows the area cost with a maximum operating frequency goal. Thanks to boolean minimization the area reduction ranges from 7% up to 20% with respect to the standard flow. ABC optimization on the other hand does not bring any significant benefits and shows results that are comparable whith a standard Synopsys flow.

Figure 7 illustrates the silicon cost to implement the partially configurable approach. This experiment is based on a 4x4 mesh topology, with a routing depth of 21 bits and boolean optimization before the synthesis. We sweep the number of nodes and redundant paths to investigate the trade offs in hardware complexity. Figure 7 also presents an exhaustive comparison between partially configurable, fully configurable and hard wired routing. Partially configurable solutions exhibit clearly better results across all the design points we consider here. LUTs with 6 and 2 alternate entries are respectively 3 and 15 times smaller than the equivalent register-based fully configurable LUT, and up to 40% of their area is required by the setting registers. The total area increase compared to a hard wired routing LUT is just approximately 20% for 6 options, and 7% in case of 2 options. Factoring in the area cost of the rest of the NI, the gain is still very clear. An NI equipped with partially configurable LUTs needs approximaltely 79% less area in case of 6 option, and 103% in case of 2 options,

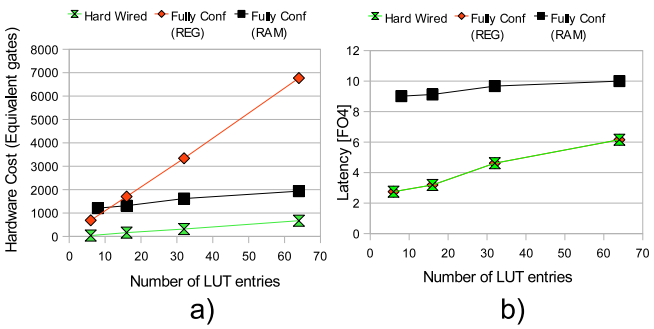

Fig. 5. Cost of hard wired and fully configurable LUTs for a 3x3 mesh: a) area cost (equivalent NAND2 gates); b) propagation delay between input and output ports (FO4 delays)

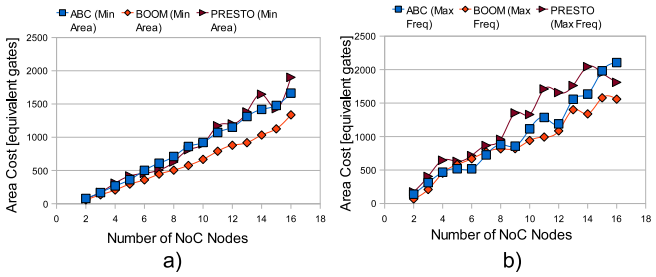

Fig. 6. Area Cost for three synthesis flows: standard flow (PRESTO), boolean minimization (BOOM) and AIG optimization (ABC). a) The area is optimized with fixed timing constraints; b) the area is optimized for maximum operating frequency

compared to one with fully configurable register-based LUTs. This proves the importance of the study in this paper and that a considerable amount of routing flexibility can be supplied at a low cost.

As a last experiment, we try to minimize the configuration register cost for partially configurable LUTs. n configuration bits (assuming just one per LUT entry) allow for 2^n possible route combinations, a number which is likely to be much larger than needed. A partially configurable LUT of 16 entries with two options each includes 15 registers used to store the setup information, therefore 2^{15} different configurations are possible. Since many of these configurations are certainly invalid, for example due to deadlocks, we now decrease the number of legal configurations, removing 10 of the 15 setting flip-flops. The 15 configuration bits will therefore be generated with 5 flip-flops and combinational gates.

Figure 7 illustrates the area savings by adopting this optimization. The sequential area becomes 3.6 times smaller, and the total cost is reduced by 20%.

A visual comparison of the considered alternative architectures is given in Figure 9, which presents layout screenshots.

VI. Conclusions and Future Work

In this paper we have presented an approach which allows NoC designers to deploy NoCs based on configurable routing. Our contributions include a way to evaluate configurability/cost/latency trade-offs among different alternate architectures, plus several investigations on how to optimize the quality of the physical-level results.

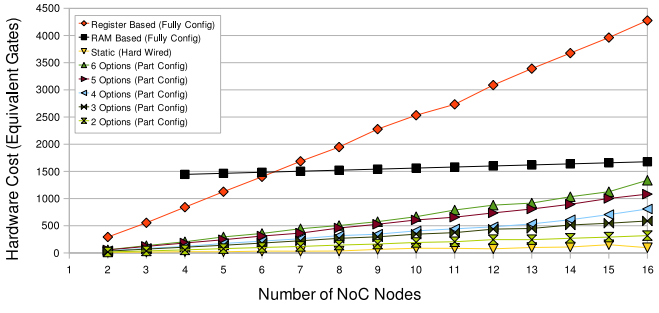

Fig. 7. Area cost of partially configurable LUTs for a 4x4 mesh when sweeping both the number of entries and the number of possible alternatives per entry

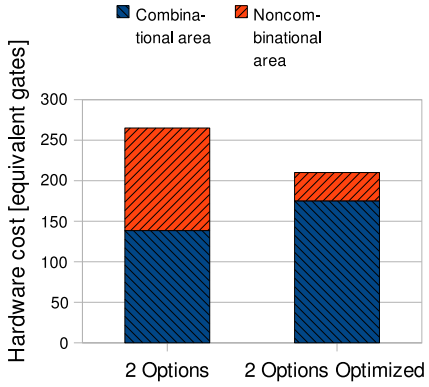

Fig. 8. 2-option and optimized partially configurable LUT. Sequential area is reduced by 3.6 times, with a total cost decrease of 20%

In our tests, hard wired LUTs prove the least expensive. But if routing flexibility is needed, reconfigurable tables are mandatory. Among register-based, RAM-based, and partially configurable LUTs, the first shows better performance for small tables (*e.g.* up to 10 words of 14 bits each). Beyond that size, RAM-based solutions are better in area. The area overhead however is still high. Partially configurable LUTs offer a good trade off between routing flexibility, area cost and performance. The area gain ranges from 3x to 15x (1,2x to 5,2x as regard to RAM based scheme) depending on the routing flexibility requirements. Further optimizations allows to drastically reduce the programmable elements, saving up to 20% extra area cost in our test case.

Future work will involve a more detailed study of the whole toolchain required to support configurable routing, *e.g.* by considering in more detail the process of selecting alternate routing table sets with deadlock freedom and fault tolerance objectives in mind.

ACKNOWLEDGEMENTS

This work has been partially supported by the GALAXY European Project (FP7-ICT-214364).

REFERENCES

[1] D. Sylvester and K. Keutzer, "A global wiring paradigm for deep submicron design," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 19, pp. 242–252, Feb 2000.

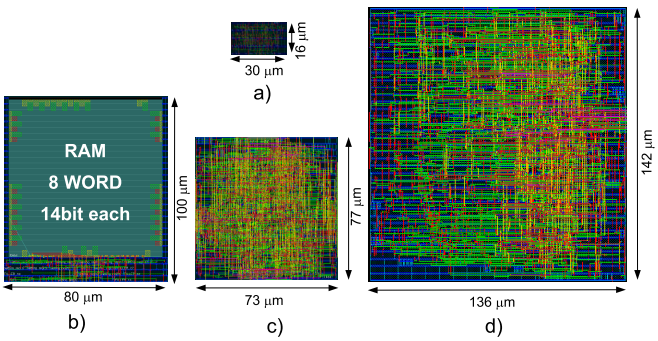

Fig. 9. Physical implementation in 130nm of the a) Hard wired LUT, b) Fully configurable RAM-based LUT, c) Partially configurable (with 6 options per path) LUT, and d) Fully configurable register-based LUT

[2] W. J. Dally and B. Towles, "Route packets, not wires: On-chip interconnection networks," in *Proceedings of the 38th Design Automation Conference*, June 2001, pp. 684–689.
[3] L. Benini and G. De Micheli, "Networks on chip: a new SoC paradigm," *IEEE Computer*, vol. 35, no. 1, pp. 70–78, January 2002.
[4] A. Pinto, L. Carloni, and A. Sangiovanni-Vincentelly, "Efficient synthesis of networks on chip," in *Proceedings of 21st International Conference in Computer Design*, October 2003, pp. 146 – 150.
[5] K. Srinivasan, K. S. Chatha, and G. Konjevod, "An automated technique for topology and route generation of application specific on-chip interconnection networks," in *Proceedings of the 2005 IEEE/ACM International conference on Computer-aided design (ICCAD '05)*. Washington, DC, USA: IEEE Computer Society, 2005, pp. 231–237.
[6] W. H. Ho and T. M. Pinkston, "A Methodology for Designing Efficient On-Chip Interconnects on Well-Behaved Communication Patterns," in *The Ninth International Symposium on High-Performance Computer Architecture (HPCA'03)*, Feb. 2003, p. 377.
[7] T. Ahonen, D. A. Siguenza-Tortosa, H. Bin, and J. Nurmi, "Topology optimization for application-specific networks-on-chip," in *Proceedings of the 2004 international workshop on System level interconnect prediction (SLIP '04)*. New York, NY, USA: ACM Press, 2004, pp. 53–60.
[8] S. Murali, P. Meloni, F. Angiolini, D. Atienza, S. Carta, L. Benini, , and G. D. Micheli, "Designing application-specific networks on chips with floorplan information," in *Proceedings of the 2006 International Conference on Computer-Aided Design (ICCAD)*, 2006, pp. 355–362.
[9] P. Guerrier and A. Greiner, "A generic architecture for on-chip packet-switched interconnections," in *Design Automation and Test in Europe, DATE'00*, March 2000, pp. 250 – 256.
[10] E. Salminen, A. Kulmala, and T. D. Hmlinen, "Survey of network-on-chip proposals," Tech. Rep., March 2008. [Online]. Available: www.ocpip.org/socket/whitepapers
[11] D. Bertozzi and L. Benini, "×pipes: A network-on-chip architecture for gigascale systems-on-chip," *IEEE Circuits and Systems Magazine*, vol. 4, no. 2, pp. 18–31, 2004.
[12] M. Coppola, R. Locatelli, G. Maruccia, L. Pieralisi, and A. Scandurra, "Spidergon: a novel on-chip communication network," in *Proceedings of 2004 International Symposium on System-on-Chip*, 2004, pp. 15–.
[13] F. Moraes, N. Calazans, A. Mello, L. Moller, and L. Ost, "Hermes: an infrastructure for low area overhead packet-switching networks on chip," *Integration, the VLSI Journal*, April 2004.
[14] E. Bolotin, I. Cidon, R. Ginosar, and A. Kolodny, "Routing table minimization for irregular mesh nocs," in *Proceedings of the conference on Design, Automation and Test in Europe*, 2007, pp. 942–947.
[15] E. Beigne, F. Clermidy, P. V. A. Clouard, and M. Renaudin, "An asynchronous noc architecture providing low latency service and its multi-level design framework," in *11th IEEE International Symposium on Asynchronous Circuits and Systems*, 2005, pp. 54–63.
[16] D. Starobinski, M. Karpovsky, and L. A. Zakrevski, "Application of network calculus to general topologies using turn-prohibition," *IEEE/ACM Transactions on Networking*, vol. 11, no. 3, pp. 411–421, Jun. 2003.
[17] M. Ali, M. Welzl, and S. Hellebrand, "A dynamic routing mechanism for network on chip," in *Proceedings of the 23rd NORCHIP Conference*, 2005, pp. 70–73.
[18] M. Palesi, S. Kumar, and R. Holsmark, "A method for router table compression for application specific routing in mesh topology noc architectures," in *SAMOS*, 2006, pp. 373–384.
[19] J. Flich, A. Mejia, P. Lopez, and J. Duato, "Region-based routing: An efficient routing mechanism to tackle unreliable hardware in network on chips," in *NOCS '07: Proceedings of the First International Symposium on Networks-on-Chip*. Washington, DC, USA: IEEE Computer Society, 2007, pp. 183–194.
[20] F. Angiolini, D. Atienza, S. Murali, L. Benini, and G. D. Micheli, "Reliability support for on-chip memories using networks-on-chip," in *ICCD*, 2006.
[21] Synopsys, "Physical Compiler," http://www.synopsys.com.
[22] P. Fiser and H. Kubatova, "Two-level boolean minimizer boom-ii," in *Proceedings of 6th Int. Workshop on Boolean Problems (IWSBP'04)*, 2004, pp. 221–228.
[23] B. L. Synthesis and V. Group, "Abc: A system for sequential synthesis and verification," Berkeley, Tech. Rep.
[24] F. Angiolini, P. Meloni, S. Carta, L. Raffo, , and L. Benini, "A layout-aware analysis of networks-on-chip and traditional interconnects for mpsocs," vol. 26, Mar 2007, pp. 421–434.