A Greedy Approach for Latency-bounded Deadlock-free Routing Path Allocation for Application-specific NoCs

Amit Verma†, Pritpal S. Multani†, Daniel Mueller-Gritschneder†, Vladimir Todorov‡ , Ulf Schlichtmann†
† Institute for Electronic Design Automation, TU Muenchen ‡ Intel Mobile Communications GmbH
amit.verma.nitrkl@gmail.com, daniel.mueller@tum.de, vladimir.todorov@intel.com, ulf.schlichtmann@tum.de

***Abstract*—Custom network-on-chip (NoC) structures have improved power and area metrics compared to regular NoC topologies for application-specific systems-on-a-chip (SoCs). The synthesis of an application-specific NoC is a combinatorial problem. This paper presents a novel heuristic for solving the routing path allocation step. Its main advantages are the support of realistic nonlinear cost estimation and the ability to handle latency constraints, which guarantee high performance of processing elements sensitive to communication delays. Additionally, the method generates deadlock-free routing by avoiding cycles in the channel dependency graph. The NoC is constructed sequentially in a greedy manner by selecting the routing path for each communication flow in such a way that the additional NoC HW resources are kept minimal. The routing path is found using a binary search cheapest bounded path (BSCBP) algorithm. The method is highly efficient and provides a NoC routing path allocation for a smart phone SoC with 25 processing elements and 96 flows in less than a minute.**

I. Introduction

Networks-on-chip (NoCs) provide better scalability for future systems-on-a-chip (SoCs) than traditional bus-based communication architectures. Much research has been conducted on the topic of routing algorithms for regularly structured NoCs such as meshes, tori, stars or fat-trees. These regular structured NoC topologies are well suited for homogeneous many-core general-purpose systems. However, SoCs, for which the application is known at design time, benefit from application-specific NoC structures, which outperform the regular topologies in terms of area and power consumption. These application-specific SoCs can supply communication between heterogeneous cores such as general purpose processors, DSPs, HW accelerators, memories, modems or serial interfaces in a cost efficient way.

Synthesizing application-specific NoC structures requires design choices such as selecting the number of routers for the network, connecting the cores to the appropriate routers, and allocating routing paths for the communication flows. These choices can be made at design time by the use of a predefined communication profile of the application. Such a profile contains multiple different use cases, which model the real life scenarios of the SoC usage. For a smart phone SoC, one use case may describe Internet surfing by using an LTE connection, another a simple phone call, yet another video capturing with the phone's camera. Each use case may result in different communication patterns between the SoC's cores. The synthesized NoC structure must support the required bandwidth to route all these communication flows. It also has to be optimal in terms of area or/and power. Additionally, performance of many cores is very sensitive to communication delays in the network (latency). For example, a processor's performance drops rapidly with increasing latency for loading data from memory after a cache miss. Therefore, latency constraints must also be included in the communication profiles.

This paper proposes a novel heuristic approach for the allocation of routing paths for the communication flows in an application-specific NoC with deterministic routing. It selects a routing path for each flow in a greedy manner, such that the additional hardware cost is kept minimal, thus optimizing power and area of the NoC. This step requires that the system has already been partitioned into groups of cores, connected to individual routers. This partitioning can be performed on the basis of communication bandwidth, latency, physical distance and power domains. This work utilizes spectral clustering as proposed in [1]. The routing is then constructed using the binary search cheapest bounded path (BSCBP) algorithm inspired by the LARAC algorithm [2], such that paths with minimal cost that also satisfy the latency constraints (e.g. number of hops) are chosen. This approach has several advantages:

- It runs in a polynomial time because of its greedy manner.
- It supports any nonlinear costs estimation based on the used NoC router technology.
- It handles latency constraints together with NoC cost during path allocation.
- It avoids deadlocks due to cycles in the channel dependency graph by rerouting flows.

The routing paths for 96 flows in a smart phone SoC with 25 cores can be generated in less than a minute. For this test case, the nonlinear HW cost computation is based on a router model for wormhole flow control. It includes analytical buffer size estimation based on the flow bandwidths, packet arrival rates and possible congestion at output ports of the router. This part of the cost is essential because the buffers are the main contributor to the NoC's area. The proposed algorithm is able to find a Routing Allocation that lead to about 20% estimated cost improvement compared to straight forward shortest path routing.

The remainder of the paper is structured as follows: Section II gives an overview over related work, Section IV briefly explains how the cores are mapped to different routers. Section V describes the proposed novel routing path allocation algorithm,

978-1-4673-6493-5/13/$31.00 ©2013 IEEE

Section VI shows experimental results for randomly generated SoCs of different complexity as well as an industrial smart phone chip, and Section VII presents the conclusions.

II. Related work

Routing Path Allocation (RPA) during application-specific NoC synthesis is an NP-hard problem. Different approaches exist to compute a good quality solution in reasonable time. In [3], methods for energy-aware system partitioning and RPA are presented. That RPA method is based on dynamic programming and uses energy consumption as cost metric. It does not consider any latency bounds. In [4], [5], a linear cost model is used to apply mixed integer linear programming (MILP) to allocate routing paths. In [6] a Modified Shortest Path (MSP) algorithm is presented as part of a NoC synthesis based on a Genetic algorithm (GA). It also routes flows sequentially taking into account the routing paths of previous flows. It does not guarantee that latency constraints are met. The method also does not explicitly avoid deadlocks, instead it inserts virtual channels if required. Another greedy RPA method for mesh topologies is described in [7]. The paper presents an idea that a constrained shortest path problem can be solved by a linear combination of the latency and cost. The same idea is used in our approach to achieve routing paths under latency constraints for application-specific NoCs by using an algorithm inspired by the Lagrangian Relaxation Based Aggregated Cost (LARAC) algorithm. In [8] and [9] the authors use linear programming (LP) and mixed-integer linear programming to approximate the solution of the topology synthesis problem. These approaches, however, do not consider delay constraints and multiple use cases. An algorithm with latency constraints is presented in [10]. In [11] a deadlock-free routing algorithm is presented that already assigns marginal cost to the edges in the network graph. It uses a similar restriction to the routing to avoid deadlocks. In [12] the authors use tabu search to find a solution to the application specific NoC synthesis problem. The method presented there uses a single use case specification and achieves deadlock avoidance by insertion of virtual channels and takes care of satisfying latency constraints. The method iteratively partitions the available routers until it find a feasible solution.

In contrast, the approach presented in this paper utilizes explicit spectral clustering to partition the system and assign the cores to routers. Furthermore, it uses an iterative algorithm to find the cheapest delay constrained paths in the routing. The search is performed on a channel routing graph and deadlocks are avoided by referencing a channel dependency graph, which indicates the occurrence of a deadlock. The channel routing graph allows the assignment of router cost for different pairs of input/output channels and to set continuous latency constraints on the flows.

III. SoC Specification

Firstly, the application-specific SoC is specified by a set of communication patterns. This specification can be given in the form of Core Communication Graphs (CCGs). Each CCG $CCG_x(C_x, F_x)$ describes a use case, where $\forall c_i \in C_x$ is a core and $\forall f_k \in F_x$ is a directed communication flow. Each flow is described with a tuple $f_k = (s_k, d_k, \beta_k, \epsilon_k, \delta_k)$, where s_k is its source core, d_k is its destination core, β_k is the average bandwidth in MBps, ϵ_k is the maximal tolerated latency (in ns, number of hops...), and δ_k is the flow's packet size distribution. An example specification of an SoC with 8 cores is illustrated in Fig. 1. The specification contains two use cases described by two distinct CCGs. The first one (Fig.1(a)) involves the cores c_0, c_1, c_2, c_3, and c_5 and the flows f_0 to f_5 which form the communication pattern between the cores. The second use case (Fig.1(b)) involves the cores c_0, c_1, c_3, c_4, c_5, c_6, and c_7 and seven flows f_6 to f_{12}.

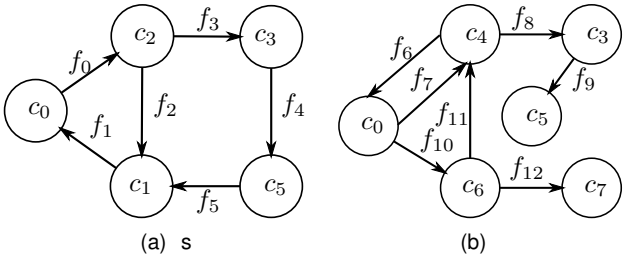

Fig. 1. Two use cases specifying an SoC

IV. System Partitioning

The first step of application-specific NoC generation requires associating the cores with specific routers. This is done by partitioning the cores into groups $\mathcal{G} = G_1, ..., G_w$. The cores inside each group G_w are connected to the local ports of one network router. Often, min-cut max-flow methods based on communication bandwidth are used to find a suitable partitioning. Additionally, latency, position on the floorplan and voltage-frequency islands might be considered during system partitioning.

In this work, the spectral clustering approach from [1], [13] is used. It finds a grouping of the cores, such that the local communication between cores inside one group is maximized and the traffic between the groups is minimized, while taking the latency constraints into account. Thus, it tries to construct the grouping in such a way that Eq.1 is minimized for every two groups G_a and G_b. In the equation $cut(G_a, G_b)$ is the inter-group communication and $vol(G)$ is the communication inside a group.

$$Ncut(G_a, G_b) = \frac{cut(G_a, G_b)}{vol(G_a)} + \frac{cut(G_a, G_b)}{vol(G_b)} \qquad (1)$$

In addition, the method can handle multi-use case specifications and can automatically estimate the number of partitions in the SoC.

Fig. 2 illustrates an example system partitioning and routers derived from the specification in Fig.1. Based on this partitioning the set of flows can be divided into two non-intersecting subsets. The first set $F_{in} = \{f_k | s_k, d_k \in G_v\}$ contains all intra-router flows. The second one $F_{out} = \{f_k | s_k \in G_v, d_k \in G_w, v \neq w\}$ contains all inter-router flows. Routing a flows in $f_k \in F_{in}$ is trivial and it is done only via the router in the corresponding group where f_k occurs. Routing of the flows

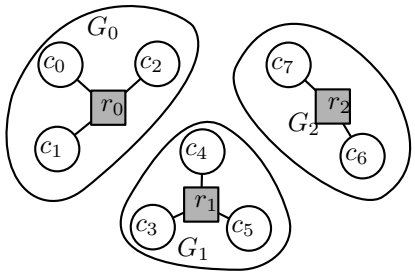

Fig. 2. Example partitioning of an SoC and the corresponding routers

in F_{out}, however, is a much harder task as multiple possible paths between the routers have to be considered.

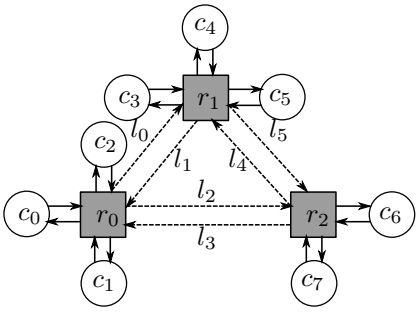

Fig. 3. Example of a system with all possible channels between the routers

V. Greedy Routing Path Allocation

The task of RPA is to find a routing path for all flows $f_k \in F_{out}$ such that the latency constraints are met, the routing is deadlock-free and the cost of the NoC structure is minimal.

This paper proposes an RPA method that greedily inserts the flows into the NoC such that the cost of the required additional hardware is minimal. For each flow $f_k \in F_{out}$, a cheapest-bounded-path algorithm based on the Dijkstra's [14] and LARAC [2] methods is used to find the optimal routing path in the channel routing graph (CRG). The weight of each edge and node in the CRG is a linear combination of the delay on the NoC channels or routers and the hardware cost. Additionally, the channel dependency graph (CDG) is analyzed to avoid any deadlocks. The CDG is a directed graph which contains all channels as nodes, but initially no edges between them. An edge between two channels in the CDG indicates a dependency between them. Thus, a cycle in the graph indicates a deadlock. Based on this the proposed algorithm checks whether selecting a link will cause a deadlock. After a flow f_k is routed the CDG is updated by inserting the edges corresponding to the path of f_k. The details of the proposed greedy RPA method are described in the following subsections.

A. Channel Routing Graph (CRG)

In the CRG $CRG_k(N_k, E_k)$ of a flow f_k, a node $l_{v,w}$ represents the inter-router channel between routers r_v and r_w as depicted in Fig. 3. The node l_s represents the local channel from the source core of the flow and l_d the local channel to the destination core.

The directed edge $e_{x,y} = (l_x, l_y)$ between two channels l_x and l_y represents a routing step, for which the flow f_k is forwarded from l_x to l_y by the router between them. Each cycle-free path from the source channel l_s to the destination channel l_d represents a possible routing path for the flow. The number of possible routing paths increases exponentially with the number of routers.

Fig. 4 shows the CRG for flow f_{10} of the example from Fig.1. There are two possible routing paths for f_{10}. The channels on those paths are labeled in Fig. 3.

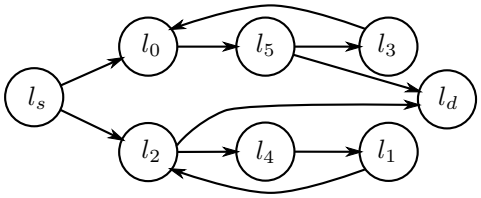

Fig. 4. CRG graph for flow f_{10} from Fig.1(b)

B. Edge Labeling

In order to select the optimal routing path for a flow f_k, each edge $e_{x,y}$=(l_x,l_y) of the CRG is labeled with a delay $\alpha_{x,y}$ and a cost $\Delta\kappa_{x,y}$. The value $\alpha_{x,y}$ of an edge $e_{x,y}$ is the sum of the predefined delay (in hops or seconds) on the channel l_x and the router r_i, to which it serves as an input:

$$\alpha_{r,k} = \alpha(e_{r,k}) = \alpha(l_x, l_y)) = \alpha_{l_x} + \alpha_{r_i} \tag{2}$$

The value of $\Delta\kappa_{x,y}$ is the cost of the additional hardware that needs to be added to support the flow f_k on the l_x and on the subsequent router r_i as well as the cost to route the flow to the next channel l_y using the router between the two channels. Therefore, it is expressed as the difference between the cost having the already inserted the new flow and the cost without the new flow (the old cost):

$$\begin{aligned} \Delta\kappa_{x,y} &= \Delta\kappa(e_{x,y}) = \Delta\kappa(l_x, l_y) = \\ &= \kappa_{new}(l_x, l_y) - \kappa_{old}(l_x, l_y) \end{aligned} \tag{3}$$

The contributors to the cost are the switching matrix size of the router, the estimated buffer sizes of the different ports and the port bitwidth (channel bitwidth) of the router. Additional hardware resources may be required because:

- the input or output channel were not yet used and need to be added to the router increasing the switching matrix size,
- the buffer size on the used input channel must be increased,
- the buffer size on another input channel needs to be increased due to congestion on a shared output channel or
- the port size of the router must be increased to support the new maximal bandwidth on the input or output channel.

For example, if the depicted edge $e_{0,5}$=(l_0,l_5) is included in the routing path, l_0 is used as an input channel and l_5 as an output channel for router r_1. To calculate the additional cost, first the old cost hardware cost $\kappa_{old}(l_0, l_5)$ is computed from all flows previously routed on channel l_0 and all flows

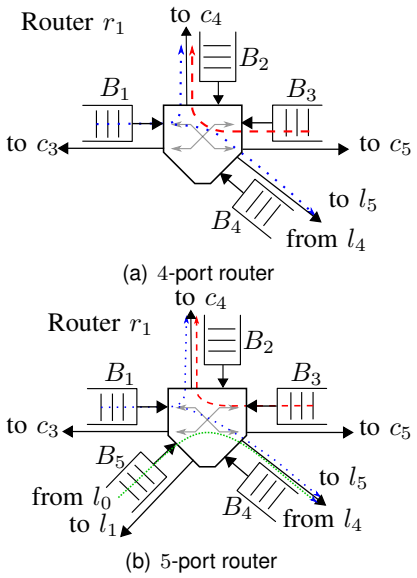

Fig. 5. Channel construction - only if flows use it

that use the router r_1. Next, the flow f_{10} is added to the existing flows that use the channel and the router. Thus, a new hardware cost $\kappa_{new}(l_0, l_5) \geq \kappa_{old}(l_0, l_5)$ is derived. $\Delta\kappa(l_0, l_5)$ is computed from these two according to Eq.3. If the channel l_1 has not yet been used, the cost is equal to $\kappa_{new}(l_0, l_5)$, which represents the complete hardware to create the channel and route the flow because $\kappa_{old}(l_0, l_5)$ equals zero. For example, in Fig. 5(a) the port interfaced with the channels l_0 and l_1 has not been created. Thus, the cost $\Delta\kappa$ of routing a flow through it will equal the complete hardware needed to instantiate this port for the first time. In Fig. 5(b), where the port is already present, $\Delta\kappa$ will represent only the additional hardware for routing the flow through the port interfaced with l_1 and l_0.

C. Buffer Estimation

Producing adequate buffer estimates is not a trivial task. For example, flows that use the same router may affect each other, thus, increasing the requirements of buffer space of the router as a whole. Figure 5(b) show the flows coming on different input ports, which interfere with each other while passing through the router. In the figure, a 5-port router is depicted, where B_1 to B_3 are the input buffers of the different ports. The flows coming on B_0 and B_2 are interfere with each other, while accessing c_4. This contention contributes to the increasing both the sizes of B_1 and B_3. Thus, adding a flow to one input channel of the router may or may not increase the buffers on multiple input channels of the same router, depending on the output channel chosen by the flow.

To cope with this problem, this work uses the the proposed apporach from [15] to estimate the buffer requirements and the bitwidth of the ports for each flow that is to be routed. The authors in [15] present an analytical method for estimating the average buffer sizes of NoC routers using queuing theory. To estimate the buffer sizes of the router, the method uses the clock frequency of the router, the port bitwidth of the router w, the flows on the router, together with their input and output channels and packet distributions δ. The buffer cost is combined with the cost of the switching matrix size to estimate the cost of routing a flow and to properly scale the port sizes.

D. Cost computation

The advantage of the proposed method is that it can be used to compute incrementally the non-linear cost estimate of the buffers. Furthermore, it allows computation of the tradeoff between buffer size and port bitwidth, which allows comparing the cost of the switching matrix and the estimate buffer sizes. This is due to the fact that the switching matrix cost increases with increases port width, but the buffer requirements decrease due to increased throughput. In this work, the buffer estimates, port bitwidths and switching matrices are recomputed for each flow that needs to be routed. The cost of a switching matrix (κ_{SM}) can be obtained from the bitwidth of the port (w), the number of output channels (out) and the number of input channels (in) as shown in Eq. 4. It roughly represents the number of gates needed to construct the matrix. The cost of the buffers κ_{BUF} is computed according to Eq. 5, where $size_{BUF}$ is the buffer size in number of bits and the factor 10 is an industrial *rule-of-the-thumb* used to estimate the number of gates needed for a flip-flop.

$$\kappa_{SM} = out \times w \times (in - 1) + (w - 1) \times out \times (in - 1) \approx \approx 2 \times out \times in \times w \quad (4)$$

$$\kappa_{BUF} = 10 \times size_{BUF} \quad (5)$$

Equations 4 and 5 are used to compute the cost $\kappa(l_x, l_y) = \kappa_{SM}(l_x, l_y) + \kappa_{BUF}(l_x, l_y)$. As both the estimation of the buffer size and the switching matrix use the bitwidth of the port in the cost computation, a tradeoff between the two can be computed. This is done by constantly increasing w until $\min(\kappa_{SM} + \kappa_{BUF}) = \min(\kappa(l_x, l_y))$ is reached. Such an iteration is done when computing the new cost $\kappa_{new}(l_x, l_y)$. It allows finding the minimum required $\Delta\kappa(l_x, l_y)$ because it minimizes $\kappa_{new}(l_x, l_y)$.

E. BSCBP Algorithm

The computation of the cost or delay on the channels and routers is done independently for each edge of the CRG for every flow f_k. The total cost or delay of any valid routing path of the flow f_k from its source channel l_s to its destination channel l_d is the sum of the edge costs or delays respectively along the path, which includes the required hardware and delays of all used channels and routers. The optimal routing path of one single flow f_k is the one that leads to minimal cost while assuring that the total delay on the path is smaller than the maximal tolerated latency ϵ_k on the flow. This is a constrained shortest path problem, which can be solved by our binary search cheapest bounded path (BSCBP) algorithm, which is inspired by the LARAC algorithm [2]. The pseudocode of the used method is shown in Algorithm 1. The BSCBP algorithm uses the linear combination of multiple edge weights to find a total edge weight $w_{x,y}$:

$$w_{x,y}(a) = a \times d_{x,y} + (1 - a) \times \Delta\kappa_{x,y} \quad (6)$$

The variable a is the scaling factor of the algorithm. The value of $d_{x,y}$ represents the delay of the edge $e_{x,y}$, which in this work has the value of 1 hop. The algorithm, however, can work with an arbitrary delay measure. BSCBP first runs Dijkstra's algorithm with $a = 0$ to find the shortest path for $w_{x,y}(0) = \Delta\kappa_{x,y}$ between the source and destination channel in the CRG of the flow. This will result in the cheapest routing path in terms of cost. If the total delay on this routing path is lower than the latency constraint, the path is optimal and the algorithm terminates. If the latency constraint is not met, it reruns Dijkstra's algorithm with $a = 1$ such that $w_{x,y}(1) = d_{r,k}$. This will result in the fastest routing path for the flow. If the latency constraint is not met, then the system partitioning must be changed because no valid routing path exists for this flow. Otherwise, there exists a range of a values that will give the cheapest path that meets the latency constraint using Dijkstra's algorithm with edge weights according to (6). Unlike the standard implementation of LARAC in [2], BSCBP uses binary search to find an a value in this range as described in Algorithm 1(line 15-25).

Algorithm 1 BSCBP Cheapest Bounded Path

```
1: PROCEDURE: BSCBP(CRG_k, ε_k, I_max)
2: BEGIN
3: a ← 0, i ← 0
4: path ← Dijkstra(CRG_k, w_{x,y}(a))
5: if delay(path) ≤ ε_k then
6:     return path
7: end if
8: a ← 1
9: path ← Dijkstra(CRG_k, w_{x,y}(a))
10: if delay(path) > ε_k then
11:     return FAIL
12: end if
13: a_l ← 0, a_u ← 1
14: path_old ← path
15: κ(path) = −1
16: while i < I_max ∧ κ(path) ≠ κ(path_old) do
17:     a ← (a_l + a_u)/2
18:     path ← Dijkstra(CRG_k, w_{x,y}(a))
19:     if delay(path) ≤ ε_k then
20:         a_u ← a
21:         path_old ← path
22:     else
23:         a_l ← a
24:     end if
25:     i ← i + 1
26: end while
27: return path_old
28: END
```

F. Deadlock Avoidance

To avoid deadlocks in the NoC, a channel dependency graph (CDG) is considered for each of the different use cases. It captures the sequence of link utilization by the different flows when they are routed through the network. A cycle in the graph indicates the occurrence of a deadlock and, hence, must be avoided. The nodes in the CDG are the same as the nodes of the CRG, except for the local channels l_s and l_d, which are not present. Whenever a flow uses two channels in sequence, a directed edge is added between the first and the second of them, indicating a dependence. An example is given for flow f_{10} in Fig. 6(a) and Fig. 6(b), representing respectively the CRG and CDG graphs of the flow. There flows f_6 and f_{11} have already been routed. The routing path for f_6 is $l_5 \rightarrow l_3$ and for f_{11} it is $l_3 \rightarrow l_0$. Both lead to the dependencies in Fig. 6(a). Thus, when trying to route f_{10} the transition $l_0 \rightarrow l_5$ will be prohibited because it will cause a deadlock. Hence, this transition and the subsequent ones origination after it will be invisible to the BSCBP algorithm, leaving only channel l_2 as an option.

This avoidance is achieved by labeling the nodes of the CRG with their predecessor nodes from the CDG. As the exploration of nodes proceeds with the BSCBP algorithm, labels are also propagated onto successor nodes(channels). The BSCBP algorithm is modified such that it checks the label of of the current node if it includes any of its successor nodes. Whenever a match occurs, the successor node is ignored. Once the a routing path is found, it is also used to update the CRG.

The introduction of the deadlock constraint causes the path allocation algorithm to find suboptimal paths as it is fundamentally based on Dijkstra's algorithm.

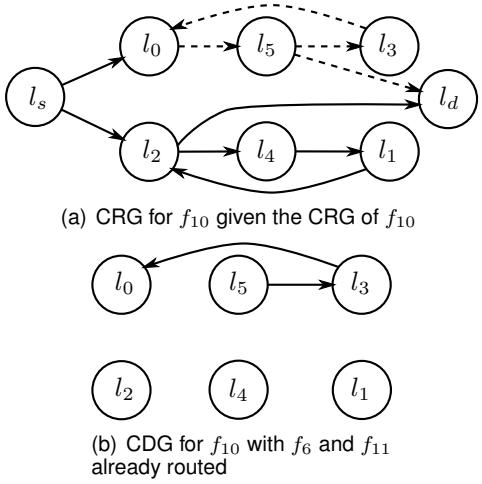

Fig. 6. Example of a routing restriction for flow f_{10}

G. Greedy Flow Insertion

The previous subsections described how to construct the deadlock free routing under latency constraints for an individual flow. To conduct the complete routing path allocation, the flows are inserted into the NoC in sequence as shown in Algorithm 2. The allocation of routing paths for all intra-router flows $f_k \in F_l$ is trivial. The routing path consists of the router input channel coming from the source core and the output channel leaving to the destination core. The intra-router routing paths are saved in the set of routing paths P_r of already routed flows F_r. Then, the NoC structure is updated such that it can support intra-router flows. First, the flows in F_g are sorted in descending order by their bandwidth requirements. The flows are then routed in order. A CRG is generated for each flow and the costs are computed for all edges in the CRG as described in the previous part of the paper. Its CRG is generated and the edge costs are computed based on the current NoC structure. The BSCBP Cheapest Bounded Path algorithm is run to find the cheapest routing path that meets

the latency constraint and is deadlock free. In this way, the routing path is chosen that leads to minimal additional costs. The found routing path is added to the set of routing paths P_r and the flow to the set of already routed flows F_r. The CDG of the corresponding use case is updated appropriately. This leads to a greedy approach that does not find the optimal global routing allocation for all flows, but leads to a very good solution in a very short time. The quality of the solution depends also on the choice of the order of the flow insertion. In this paper, we investigate flow ordering by latency constraints and bandwidth requirement.

Algorithm 2 Greedy Flow Insertion

```
1: PROCEDURE: GFI(F_g)
2: BEGIN
3: S ← sort(F_g)
4: for ∀f ∈ S do
5:    for ∀e_{x,y} ∈ CRG_f do
6:       κ_old(l_x, l_y) ← cost(CRG_f)
7:       κ_new(l_x, l_y) ← cost(CRG_f, f)
8:       Δκ(l_x, l_y) ← κ_new(l_x, l_y) − κ_old(l_x, l_y)
9:    end for
10:   path_k = BSCBP(CRG_f(N_f, E_f), ε_f, I_max)
11:   P_r ← P_r ∪ path_k
12:   update(CDG_f, path_k)
13: end for
14: END
```

H. Complexity Analysis

For a partitioning with R routers the number of possible channels (nodes) in the CRG grows in the order of $O(R^2)$. Dijkstra's algorithm has a complexity of $O(N^2)$, where N is the number of nodes in the graph. The BSCBP algorithm, which is based on Dijkstra's one, is run a maximum of I_{max} times. There are K flows describing the SoC that need to be routed. Therefore, the total complexity of the presented greedy routing path allocation method is polynomial of the order $O(KI_{max}R^4)$.

VI. Experimental Results

In this section, experimental results are presented to validate the presented routing path allocation (RPA) method. We compare the results to shortest path RPA. Shortest path RPA results in the fastest routing paths. These paths should always meet the latency constraints. The drawback of shortest path RPA is that it does not consider NOC cost, thus, leading to a possible rather expensive solution.

In contrast, the presented greedy RPA using the BSCBP algorithm always looks for the cheapest route that still meets the latency constraint. With these cheap routing paths, the method is able to reduce the NoC cost at the price of a slightly overall higher bandwidth-hop product. As the experimental results show, the method finds a far better compromise than shortest path RPA.

A. Testcases

To evaluate the effectiveness of the RPA method several random system specifications have been constructed. Each of them is constructed with different complexity in terms of number of cores, flows between the cores and number of routers. The results obtained from running the proposed routing method are compared against a shortest path approach, where inter-router flows are routed directly between the communicating routers. Furthermore, different flow orderings have been taken into account. Table VI-A shows the results obtained from the random specifications. In the table there are 5 random cases presented, starting from a very small one with 5 cores, 15 flows and 3 routers to a complex one having 40 cores, 160 flows and 15 routers. Both the combined switching matrix and buffer costs and the bandwidth-hop product are listed in the table. The bandwidth hop-product is a measure, which shows what amount of data is contained in the NoC on average at any point in time. Additionally, "B" and "L", show respectively sorting of flows according to flow with max. bandwidth first and flow with tightest latency constraint first. As it can be seen from the table the proposed routing method is always better than shortest path approach except once. In this case the rerouting of flows leads to higher cost. This shows that different orderings of the flows should be investigated to find always a better solution. However, as the complexity increases the proposed RPA method produces significant cost gains with little increase on the bandwidth-hop product. Furthermore, in most cases sorting the flows by either bandwidth or latency is more advantageous as compared to no sorting at all. Finally, the running time increases with the increase of complexity, but it is still in the order of several minutes.

TABLE I
Experimental for randomly generated specifications

Algorithm	Sort	BW-hop prod.	Cost	Time [s]
Case: 5 Cores, 15 Flows, 3 Routers, 1 Usecase				
Shortest path	-	64.4	17136	< 1
Greedy	-	66.2	16968	< 1
	B	70.3	16968	< 1
	L	69.2	**16288**	< 1
Case: 10 Cores, 30 Flows, 5 Routers, 2 Usecases				
Shortest path	-	51.0	17280	< 1
Greedy	-	56.4	17064	1.53
	B	56.0	18296	2.33
	L	53.0	**16408**	1.82
Case: 15 Cores, 45 Flows, 7 Routers, 3 Usecases				
Shortest path	-	61.6	21880	< 1
Greedy	-	72.1	20736	6.51
	B	72.1	**18472**	6.43
	L	71.3	19184	4.96
Case: 20 Cores, 80 Flows, 9 Routers, 4 Usecases				
Shortest path	-	65.5	44776	< 1
Greedy	-	80.0	40176	48.36
	B	78.4	38592	46.7
	L	86.3	**34544**	44.1
Case: 40 Cores, 160 Flows, 15 Routers, 5 Usecases				
Shortest path	-	62.2	82224	< 1
Greedy	-	83.3	61360	727
	B	80.0	62608	662
	L	82.6	**49528**	541

B. Smart Phone Specification

To test the feasibility of the method on a real SoC designs, it is applied to the specification of an industrial smartphone SoC,which features 25 cores and 96 flows. The SoC specification includes five different use cases: Full HD Video Playback, Video Capture, LTE Mobile Internet Access, WiFi Mobile Internet Access, Video Streaming via LTE or WiFi and Idle. The NoC architecture must support all flows of any of these use cases simultaneously. The SoC only operates one use case at a time, such that the NoC must not support the flows of two or more use cases simultaneously. The frequency of the NoC is chosen to 500 MHz. The partitioning is done as described in Section IV. It finds nine groups of cores, thus 9 routers are used for the NoC. Table II shows the results for the routing path allocation step. Shortest path RPA finds the solution with the optimal bandwidth-hop product. However, all flows are directly routed to the channel that leads to the destination router regardless of the NoC cost. This leads to minimal latencies for the flows, but results in high HW costs due to the many required inter-router channels. The proposed greedy RPA approach leads to slight degradation in bandwidth-hop product compared to shortest path routing at a significant improvement in the hardware cost of about 23% of the shortest path NoC cost. This is achived by rerouting flows to channels that require minimal additional cost. As can be seen, the greedy algorithm with latency bounds gives a very good compromise between bandwidth-hop product and NoC cost. The resulting topology of the NoC is shown in figure 7. The topology presented is the one with the least cost from Table II. As can be seen, there is one central router that connects to the memory and the CPU. It is strongly connected with the remaining network because flows from these two cores are most sensitive to latencies. This router would need to be designed carefully.

TABLE II
EXPERIMENTAL RESULTS FOR SMART PHONE SPECIFICATION

Algorithm	Sort	BW-hop prod.	Cost	Time [s]
Shortest path	-	154	31008	< 1
Greedy	-	156	28024	47.8
	B	158	23704	28.3
	L	163	27672	36.7

VII. CONCLUSIONS

This work proposes a novel greedy algorithm for the routing path allocation problem in the design of custom NoCs. The work uses a multi-use case specification of a SoC to produce and route a NoC, which satisfies all the use cases. It uses a greedy algorithm, which is deadlock free and satisfies latency constraints, combine with a cost function considering both switching matrix and buffer sizes to insert the links between the routers in a fast and cost-effective way.

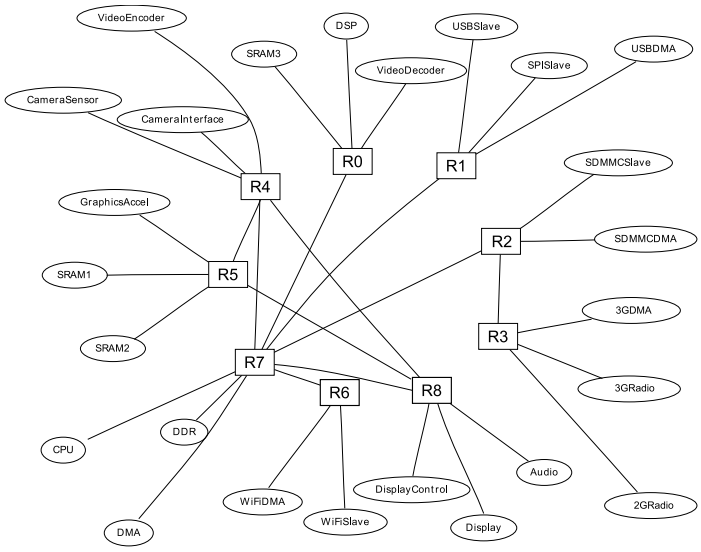

Fig. 7. Smartphone NoC Topology Graph for Greedy RPA

REFERENCES

[1] V. Todorov, D. Mueller-Gritschneder, H. Reinig, and U. Schlichtmann, "A Spectral Clustering Approach to Application-Specific Network-on-Chip Synthesis," in *Design, Automation and Test in Europe (DATE)*, Mar. 2013.

[2] Y. Xiao, K. Thulasiraman, G. Xue, and A. Juettner, "The Constrained Shortest Path Problem: Algorithmic Approaches and an Algebraic Study with Generalization," *AKCE International Journal of Graphs and Combinatorics*, vol. 2, no. 2, pp. 63–86, Dec. 2005.

[3] B. Yu, S. Dong, S. Chen, and S. Goto, "Floorplanning and topology generation for application-specific network-on-chip," in *Asia and South Pacific Design Automation Conference*, 2010.

[4] A. M'zah and O. Hammami, "Area/delay driven NoC synthesis," in *Microelectronics (ICM), 2011 International Conference on*, Dec. 2011, pp. 1–6.

[5] K. Srinivasan and K. S. Chatha, "A Methodology for Layout Aware Design and Optimization of Custom Network-on-Chip Architectures," in *IEEE International Symposium on Quality Electronic Design (ISQED)*, ser. ISQED '06. Washington, DC, USA: IEEE Computer Society, 2006, pp. 352–357.

[6] G. Leary, K. Srinivasan, K. Mehta, and K. Chatha, "Design of Network-on-Chip Architectures With a Genetic Algorithm-Based Technique," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 17, no. 5, pp. 674–687, May 2009.

[7] A. Hansson, K. Goossens, and A. Rădulescu, "A unified approach to constrained mapping and routing on network-on-chip architectures," in *IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*, ser. CODES+ISSS '05. New York, NY, USA: ACM, 2005, pp. 75–80.

[8] K. Srinivasan, K. S. Chatha, and G. Konjevod, "An Automated Technique for Topology and Route Generation of Application Specific On-Chip Interconnection Networks," in *ICCAD '05*, 2005.

[9] K. Srinivasan and K. S. Chatha, "A methodology for layout aware design and optimization of custom network-on-chip architectures," in *ISQED*, 2006.

[10] G. Leary and K. Chatha, "A Holistic Approach to Network-on-Chip Synthesis," in *CODES+ISSS'10*, 2010.

[11] S. Murali, P. Meloni, F. Angiolini, D. Atienza, S. Carta, L. Benini, G. D. Micheli, and L. Raffo, "Designing application-specific networks on chips with floorplan information," in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. New York, NY, USA: ACM, 2006, pp. 355–362.

[12] G. N. Khan and A. Tino, "Synthesis of NoC Interconnects for Custom MPSoC Architectures," in *IEEE Computer Society*, ser. 6^{th} International Symposium on Networks-on-Chip, March 2012.

[13] U. Luxburg, "A tutorial on spectral clustering," *Statistics and Computing*, 2007.

[14] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms, Third Edition*. The MIT Press, 2009.

[15] U. Y. Ogras, P. Bogdan, and R. Marculescu, "An analytical approach for network-on-chip performance analysis," *Trans. Comp.-Aided Des. Integ. Cir. Sys.*, 2010.