Exploring Power and Throughput for Dataflow Applications on Predictable NoC Multiprocessors

Kathrin Rosvall, Tage Mohammadat, George Ungureanu, Johnny Öberg, and Ingo Sander
Department of Electronics
School of Electrical Engineering and Computer Science
KTH Royal Institute of Technology
Stockholm, Sweden
{krosvall, tagem, ugeorge, johnnyob, ingo}@kth.se

***Abstract*—[1]System level optimization for multiple mixed-criticality applications on shared networked multiprocessor platforms is extremely challenging. Substantial complexity arises from the interdependence between the multiple subproblems of mapping, scheduling and platform configuration under the consideration of several, potentially orthogonal, performance metrics and constraints. Instead of using heuristic algorithms and problem decomposition, novel unified design space exploration (DSE) approaches based on Constraint Programming (CP) have in the recent years shown promising results. The work in this paper takes advantage of the modularity of CP models, in order to support heterogeneous multiprocessor Network-on-Chip (NoC) with Temporally Disjoint Networks (TDNs) aware message injection. The DSE supports a range of design criteria, in particular the optimization and satisfaction of power and throughput. In addition, the DSE now provides a valid configuration for the TDNs that guarantees the performance required to fulfil the design goals. The experiments show the capability of the approach to find low-power and high-throughput designs, and validate a resulting design on a physical TDN-based NoC implementation.**

I. Introduction

Multiprocessor Network-on-Chip (NoC) architectures are a prominent choice for embedded platforms, aiming at providing high computational performance under a constrained power budget. Many embedded applications have tight timing constraints, e.g. to maintain a certain throughput for video and audio processing. Providing timing guarantees is facilitated by underlying platforms that provide predictable quality of service. A range of NoC platforms exhibit a guaranteed service interconnect, as for example Æthereal [1] and Nostrum [2]. These predictable platforms are a perfect match for mixed-criticality systems, because different applications can use parts of the NoC communication infrastructure without interfering with each other, allowing to give timing guarantees to individual applications. This is handled by assigning each application a share of the interconnect by means of time slots to the communicating resources.

However, the massive number of potential design solutions makes it extremely challenging to find a satisfying solution which is either optimal or efficient. Combining predictable NoC platforms with formal analyzable models based on the theory of models of computation (MoC) for applications [3] into a unified design space exploration (DSE) method is a promising approach to overcome this situation. The DSE approach used in this paper is based on constraint programming (CP). With CP, the interdependence between the sub-problems of mapping, scheduling, platform and interconnect configuration and performance analysis of potentially orthogonal performance metrics, as for example power consumption and throughput, can be captured and solved in a unified manner, instead of decomposing it into its sub-problems.

The paper presents an efficient DSE method for mixed-criticality applications that share a power-efficient and predictable multiprocessor platform. Both predictability and power-efficiency is achieved through the use of Temporally Disjoint Networks (TDN), which is a natural property in deflection routing NoCs that leads to minimal buffers in the NoC switches [2].

Although TDN-NoCs have a high potential by providing predictability at low power consumption, this potential has so far not been fully untapped. A main reason is the challenging problem of identifying the correct injection tables to avoid collisions in the network and to achieve guaranteed quality of service. This problem is addressed in this paper by modelling the problem and providing a DSE method that generates the correct injection tables for mixed-criticality applications with guaranteed quality of service.

In particular, the paper extends the current state-of-the-art in the field by

- including the configuration and analysis of predictable NoCs with Temporally Disjoint Networks (TDNs) into a Constraint Programming (CP) model for Design Space Exploration (DSE) which has the capability to optimize and satisfy a range of performance metrics;
- providing a power model for the TDN NoC interconnect and integrating it into the CP model for the DSE; and
- validating the model and the DSE method by the implementation of a solution produced by the DSE tool on a physical TDN-based Network-on-Chip (NoC).

II. The DSE Problem

This section introduces the type of NoC platforms considered, as well as the application model and CP as the method chosen for the DSE.

[1]N.B. This is a postprint version, K. Rosvall, T. Mohammadat, G. Ungureanu, J. berg and I. Sander, "Exploring Power and Throughput for Dataflow Applications on Predictable NoC Multiprocessors," 2018 21st Euromicro Conference on Digital System Design (DSD), Prague, 2018, pp. 719-726. doi: 10.1109/DSD.2018.00011

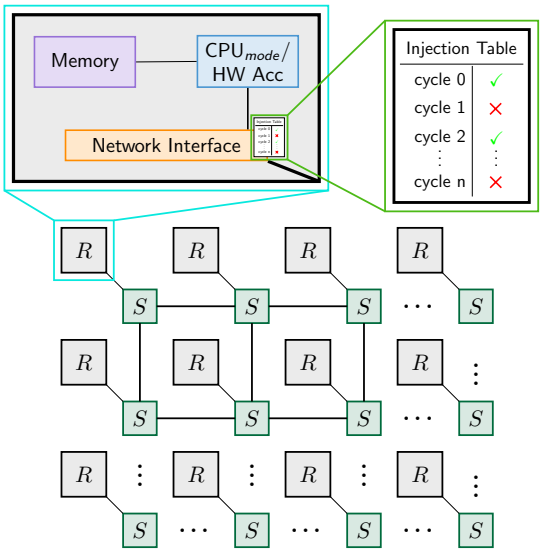


Figure 1. Illustration of a Network-on-Chip based on Temporally Disjoint Networks. 'R' corresponds to processing resource and 'S' to network switch.

A. TDN NoC Platform Model

The platform model, illustrated in Figure 1, consists of a set of processing resources connected by a NoC with mesh-topology and bi-dirctional links between the switches (S). Each processing resource (R) contains a central processing unit or an accelerator, a dedicated local memory, and a Network Interface (NI) to access the network interconnet. This paper targets a specific class of mesh-based NoCs based on deflective dimension-order routing and uses the inherent property of Temporally-Disjoint Networks (TDN) to achieve predictability as discussed in detail by Millberg et al. in [2]. Their paper showed that there are packets in such a network that can never collide, if they are injected into the network at certain times.

To understand the principle of TDN, consider two packets that are injected into the network at adjacent time slots from the same NI. Due to the mesh structure of the NoC, a packet needs an even number of hops to come back to its origin, and thus a packet injected at an odd time slot can never collide with a packet injected at an even time slot from the same resource. Furthermore, the deflection routing mechanism implies that a packet is never buffered in the switch. Based on these two properties, one can distuingish *temporally disjoint* virtual networks within the NoC. The number of TDNs can be increased by adding registers to the switches and is proportional to this number of registers.

Using the property of TDNs, guaranteed quality of service in form of predictable communication time can be achieved by scheduling the injection of packets through injection tables in such a way that packets never collide. Moreover, since deflection routing requires no additional buffers in the switches, it is a power-efficient architecture, which means that predictability comes at low cost. In that sense, TDN-NoCs are related to Time-Division Multiple Access (TDMA) NoCs which also assign time slots to messages. However, TDN has in comparison a much lower overhead in the hardware implementation, since injection tables are only needed at the NIs and not in the switches. By this means, TDNs provide guaranteed services through deadlock-free, livelock-free and collision-free, rate-constrained and temporarily deterministic communication. For brevity, the platform is referred to as TDN-NoC platform.

The platform model supports different modes for the NoC and processing resources. A mode captures design instances that can operate at various frequencies or be optimized for maximum performance or minimum area. Each mode yields different cost metrics such as computational speed, power consumption and area utilization. This flexible platform model enables the modeling of a range of real NoC instances with heterogeneous processing resources and features, as demonstrated for the Xilinx Zynq platform in the experiments.

B. Application Model

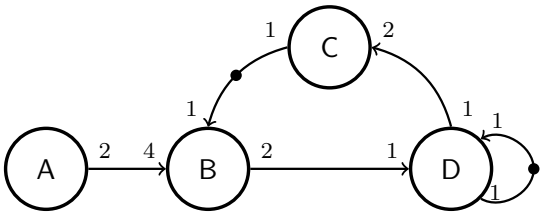


Figure 2. Example for a cyclic SDF graph

The DSE model used for the work in this paper supports synchronous dataflow graphs (SDFGs) [4] as application model. SDF graphs are widely used to model streaming applications. An example SDFG is shown in Figure 2. An SDFG $G(A, C)$ consists of a finite set of actors A and a finite set of channels C. A fixed rate p of tokens is produced by actor $a \in A$ on outport op and a fixed rate q of tokens is consumed by actor $b \in A$ from inport ip via a channel $c = (a,\ op,\ p,\ b,\ ip,\ q,\ tok)$. tok specifies the number of initial tokens held by channel c, which are are depicted with a dot on the arc for a channel, optionally with an adjacent integer signifying multiple initial tokens. The DSE maps and schedules multiple application graphs simultaneously on a shared platform, i.e. it takes a set of application graphs $G_z(A_z, C_z)$ as input.

For consistent SDFGs, the fixed production and consumption rates allow to derive a fixed number of repetitions for each actor, $\gamma : A \rightarrow \mathbb{N}$, within one iteration of the graph. All *firings* of the actors during one iteration lead back to the initial token distribution. All consistent SDFGs can be converted into single-rate, homogeneous dataflow graphs [5], [6]. Even though the conversion can lead to a massive increase in the size of the graph [7], the single-rate version brings about the significant advantage of exposing data-parallelism, which can be exploited for finding implementations with optimal or sufficient throughput, as in the DSE method in this paper.

The throughput of an SDFG is the inverse of its iteration period. The self-timed execution yields a pipelined operation with initial latency followed by the periodic phase. The length of the periodic phase for each application graph is its iteration period.

C. Method

Since Constraint Programming (CP) is a well-known technique for solving combinatorial problems, and the problem of

multi-processor scheduling under performance constraints is a typical application for combinatorial optimization, CP is a promising approach for tackling the DSE problem. The core of a CP model is a set of *decision variables* which capture the components of a solution to the problem. Each variable has a *domain* of possible values. Relationships between the variables are expressed in terms of *constraints*, e.g. the values of integer variables $\mathtt{x}$, $\mathtt{y}$ and $\mathtt{z}$ may have to satisfy the constraint $3 \cdot \mathtt{x} - \mathtt{y} = \mathtt{z}$.

A CP model is created with variables, their domains, constraints and optionally an optimization criterion. A constraint solver performs the intertwined steps of *propagation*, *branching* and *search* on the CP model. Propagation removes values from the variable domains if they are in conflict with the constraints. Branching creates a search tree from the remaining alternatives in the variable domains after propagation. The search operates on the tree to find assignments of values to all variables that satisfy all constraints, and, if requested, are optimal.

Dataflow modeling is used at the core of the CP model for the DSE, to capture the mapping and scheduling decisions and their implications. The CP variables reflect a *mapping- and scheduling-aware graph* (MSAG), which captures all input applications and the implications of mapping and scheduling decisions taken during the exploration. In particular, the MSAG is the basis to analyze each partial or complete mapping and scheduling for the minimal achievable throughput. An illustration of how a subset of the variables of the CP model form the MSAG is presented in Figure 3. They reflect actors, channels, tokens and actor properties, respectively. More details about the CP variables that form the MSAG are provided in Section III-A.

III. The DSE Method

The problem to solve is the mapping and scheduling of n SDF application graphs $G_z(A_z, C_z)$ onto a platform model with a set of processing nodes P and a $c{\times}r$-mesh NoC interconnect with c columns and r rows, where $c \cdot r = |P|$, and finding a conflict-free slot allocation for the TDN NoC. Processor $p_i \in P$ $(i \in [0, |P|{-}1])$ is located at NoC node $(i \bmod c,\ \lfloor \frac{i}{r} \rfloor)$. Each processing node p_i is associated with a set of processor modes $\mathcal{M}_{p_i}$ containing at least one mode. A mode $M \in \mathcal{M}_{p_i}$ determines the parameters of dynamic and static power consumption, area cost, monetary cost and local memory size and the WCETs of actors. The NoC has a set of modes $\mathcal{N}$ that determine for each of its components, i.e. the NIs, switches and links, the parameters of dynamic and static power consumption, area cost, monetary cost and cycle length, i.e. duration of one hop for one flit. Independent of the modes, the NoC specifies a flit size, number of TDN cycles and routing strategy. We denote the union set of application graphs $\mathcal{G} = \bigcup_{z \in [0,n-1]} G_z = (\mathcal{A}, \mathcal{C})$, with $\mathcal{A} = \bigcup_{z \in [0,n-1]} A_z$ and $\mathcal{C} = \bigcup_{z \in [0,n-1]} C_z$, respectively.

A. The CP DSE Model

The CP model captures static order-based schedules, i.e. it does not provide schedules with start or end times, but a fixed order in which actors execute on each processor. With this type of schedules and blocking read and write primitives, the system can be implemented as bare-metal solution, without a dynamic scheduler or operating system. The order-based schedules, along with execution times, can be used to infer time-triggered schedules. For the MSAG introduced in Section II-C, scheduling decisions potentially involve adding channels between the actors of the input application graphs. The order-based schedules are captured through the variables $\mathtt{next}_a$ (1), pointing to a successor actor or, in case a is the last actor in the schedule, to a special reserved value. Figure 3 shows how the decision $\mathtt{next}_{\mathtt{src_a}} = \mathtt{src_b}$ adds a channel between the two actors.

$$\forall a \in \{0..|\mathcal{A}|{+}|P|{-}1\} : \mathtt{next}_a \in \{0..|\mathcal{A}|{+}|P|{-}1\} \quad (1)$$

Mapping of actors to processors determines their WCET. The processor and processor mode assignments are captured by variables $\mathtt{proc}_a$ (2) and $\mathtt{proc\_mode}_p$ (3), respectively. The actors' WCETs are assigned based on this mapping (4). The mapping of actors can also inflict communication delays, which is taken up in Section III-B.

$$\forall a \in \mathcal{A} : \mathtt{proc}_a \in P \quad (2)$$

$$\forall p \in P : \mathtt{proc\_mode}_p \in \{0..|\mathcal{M}_p|{-}1\} \quad (3)$$

$$\forall a \in \mathcal{A} : \mathtt{wcet}_a(\mathtt{proc}_a, \mathtt{proc\_mode}_{\mathtt{proc}_a}) \in \mathbb{N}^+ \quad (4)$$

The performance metrics of the DSE model are listed in (5) - (9). The iteration periods (5) of the application graphs, i.e. inverse of their throughput, results from decisions for all CP variables that create the MSAG that was introduced in Section II-C. Its value is determined by a specialized throughput propagator [8] that performs maximum cycle ratio analysis [9] on the MSAG. The utilization of all processors is given by $\mathtt{system\_utilization}$ (6). Power consumption, area cost and monetary cost of the entire system during the periodic phase are captured by $\mathtt{system\_power}$ (7), $\mathtt{system\_area}$ (8) and $\mathtt{system\_cost}$ (9). The CP model also contains an alternative version of the CP variables (6) - (9), $\mathtt{systemUsed\_}*$, which capture the performance metric considering only the parts of the platform model that are used in the solution found by the DSE. This information allows the designer to make the platform smaller if possible, without having to perform the DSE again. The power model of the CP is described in more detail in Section III-C.

$$\forall g \in \mathcal{G} : \mathtt{period}_g \quad (5)$$

$$\mathtt{system\_utilization} \in \{0..maxU\} \quad (6)$$

$$\mathtt{system\_power} \in \mathbb{N} \quad (7)$$

$$\mathtt{system\_area} \in \mathbb{N} \quad (8)$$

$$\mathtt{system\_cost} \in \mathbb{N} \quad (9)$$

The model also ensures that the mapping is valid in terms of sufficient memory on all processing nodes. This involves memory consumed by actors instantiated on processing nodes, as well as communication buffers placed into the local memory. The available buffer locations on the sending (10) and receiving (11) side are represented by initial tokens in the MSAG. The non-filled circles in Figure 3 indicate that the

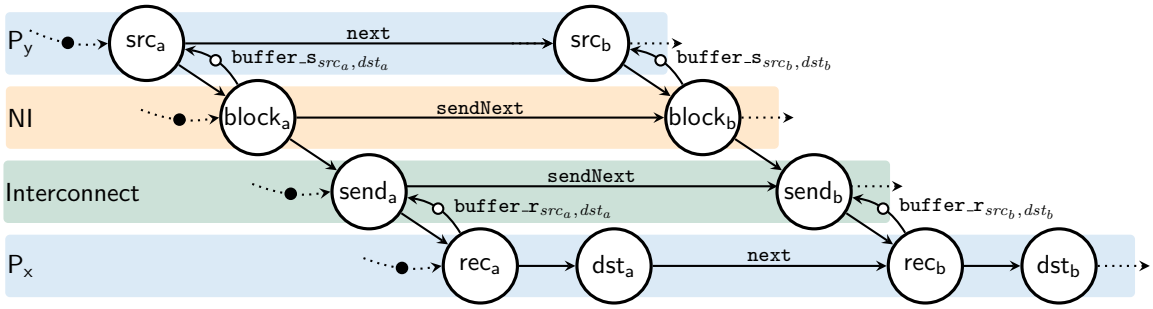


Figure 3. CP variables form a mapping- and scheduling aware graph (MSAG)

buffer sizes may be only bounded and not fixed during the ongoing exploration.

$$\forall c \in \mathcal{C} : \texttt{buffer\_s}_c \in \mathbb{N}^+ \quad (10)$$

$$\forall c \in \mathcal{C} : \texttt{buffer\_r}_c \in \mathbb{N}^+ \quad (11)$$

B. The TDN-NoC in the CP Model

The objective of the TDN NoC model in the CP model is to configure the interconnect through choice of NoC mode (12) and conflict-free injection tables for all NIs of the NoC (13), and to determine the resulting communication delays in terms of blocking and sending delays for all channels (17) -(18). The delays are the WCETs for the block and send actors of the MSAG depicted in Figure 3.

The TDN-NoC provides the following input parameters to the model: N is the set of NIs, S is the set of switches, L is the set of links, T is the set of TDNs and $\mathcal{N}$ is the set of modes of the NoC; $hops_{p_{src},p_{dst}}$ is the number of hops between two processors; $\texttt{cycleLength}(n)$, $n \in \mathcal{N}$ is the cycle length depending on the NoC mode; $flits_c$ is the number of flits needed for one token of channel c; $access_l$ is the set of processors that, based on the routing strategy, can send messages on link l; and $TDNpath_{p_{src},p_{dst},t} \subset L \times T$ is the route, as a set of links at a specific TDN slot, that is traversed by a flit injected at p_{src} at TDN slot t.

$$\texttt{noc\_mode} \in \{0..|\mathcal{N}|-1\} \quad (12)$$

$$\forall n \in N, \forall t \in T : \texttt{inj\_tbl}_{n,t} \in \{0..|P|\} \quad (13)$$

$$\forall c \in \mathcal{C} : \texttt{tdn\_slot}_c \in \{0..|T|\} \quad (14)$$

A processor that does not send any messages is not assigned any TDN slots. A processor that sends messages is assigned at most as many TDN slots as required to send the largest token of all channels originating from that processor. The designer can restrict the number of TDN slots that a processor can use. If each processor may not use more than one TDN slot, then all channels that originate from the same processor must be assigned the same slot with $\texttt{tdn\_slot}_c$ (14). For local channels, with source and destination actor on the same processor, $\texttt{tdn\_slot}_c$ is assigned the special value $|T|$ to indicate that the channel does not use any TDN slot. For all channels c that originate from a processor p, $\texttt{inj\_tbl}_{n_p,\texttt{tdn\_slot}_c} = p$.

In order to guarantee a conflict-free assignment of TDN slots for the traffic pattern resulting from the mapping of actors to processors, the state of all links of the NoC at all TDN slots needs to be included in the model. The CP variable $\texttt{noc\_state}_{l,t}$ (15) captures this by assigning either a processor p to link l at TDN slot t, which means that processor p can inject a message at a TDN slot t' that traverses link l in slot t, or the value $|P|$, which indicates that the link l is not used at slot t for the given traffic pattern and injection table configuration. The injection tables for the NIs (13) correlate directly with $\texttt{noc\_state}$ for all links from an NI n to a switch s (16). For a channel c with source actor on processor p_{src} and destination actor p_{dst}, the route through the TDN-NoC follows dimension-order routing and is reserved by assigning all links at the corresponding TDN slot to the sending processor: $\forall (l,t) \in TDNpath_{p_{src},p_{dst},\texttt{tdn\_slot}_c} : \texttt{noc\_state}_{l,t} = p_{src}$. Since only one processor can be assigned the access to each link at each slot, $\texttt{noc\_state}$ ensures a conflict-free TDN slot assignment in the injection tables.

$$\forall l \in L, \forall t \in T : \texttt{noc\_state}_{l,t} \in access_l \cup \{|P|\} \quad (15)$$

$$\forall l{=}(n,s) \in N{\times}S \subset L : \texttt{inj\_tbl}_{n,t} = \texttt{noc\_state}_{l,t} \quad (16)$$

The CP model also captures the communication delays for channels that use the TDN-NoC interconnect. For all channels with the source and destination actors mapped onto different processing nodes, additional actors reflecting the communication process and delays are added to the MSAG, as can be seen in Figure 3. Channels for which both source and destination actor are mapped onto the same processing node are implemented with the local memory, which is assumed to not inflict any delay. Since the data dependency is respected by the schedule, no change is done to the MSAG for this case. The communication delays consist of maximum blocking (17) and sending time (18). The blocking time is the time a token waits in the NI for the next assigned TDN slot. The sending time is the time after the blocking in the NI until the token reaches the destination processor. Since the communication buffers are assumed to be in the local memory of the destination processor, receiving tokens (19) does not cause any delay in the current model.

$$\forall c \in \mathcal{C}: \ \texttt{wcct\_b}_c = (|T| - \texttt{tdnSlots}_{\texttt{proc}_{\texttt{src}_c}}) \cdot \texttt{cycleLength}(\texttt{noc\_mode}) \quad (17)$$

$$\forall c \in \mathcal{C}: \ \texttt{wcct\_s}_c = (\left\lceil \frac{\mathit{flits}_c}{\texttt{tdnSlots}_{\texttt{proc}_{\texttt{src}_c}}} \right\rceil + \mathit{hops}_{\texttt{proc}_{\texttt{src}_c},\texttt{proc}_{\texttt{dst}_c}}) \cdot \texttt{cycleLength}(\texttt{noc\_mode}) \quad (18)$$

$$\forall c \in \mathcal{C}: \ \texttt{wcct\_r}_c = 0 \quad (19)$$

Channels send from (20), or received by (21), the same actor need to be scheduled for correct throughput analysis, therefore the CP model also schedules channels in an order-based manner, analogous to the `next` variables for actor schedules.

$$\forall c \in \{0..|\mathcal{C}|+|P|-1\} : \texttt{sendNext}_c \in \{0..|\mathcal{C}|+|P|-1\} \quad (20)$$

$$\forall c \in \{0..|\mathcal{C}|+|P|-1\} : \texttt{recNext}_c \in \{0..|\mathcal{C}|+|P|-1\} \quad (21)$$

C. The Refined Power Model

The power model of the CP DSE model, which determines the value of `system[Used]_power` (7), considered previously only the dynamic power consumption of the processors. For the work in this paper, the power model has been refined to consider both dynamic and static power of processors and also of all components of the NoC interconnect, i.e. NIs, switches and links. The parameters used for computing the system's power consumption are dynamic and static power consumption of each processor, depending on the processor mode, $\texttt{dynPow}(\texttt{proc\_mode}_p)$ and $\texttt{statPow}(\texttt{proc\_mode}_p)$, dynamic and static power consumption of the NoC components, depending on the NoC mode, $\texttt{dynPow}_{NI/switch/link}(\texttt{noc\_mode})$ and $\texttt{statPow}_{NI/switch/link}(\texttt{noc\_mode})$, the cycle length depending on the NoC mode, $\texttt{cycleLength}(\texttt{noc\_mode})$ and the iteration periods of the applications, $\texttt{period}_a$. The number of flits of an application a passing NI n, switch s or link l is provided by $\texttt{flits}(a, n/s/l)$. It is important to note that the notion of static and dynamic power consumption used in this paper differs from the standard notions in the field of circuit design in the following sense:

- static power consumption in this paper, $\texttt{statPow}_x$, models application-independent power consumption per component x as a baseline for estimating power consumption. This implies factoring in elements that are conventionally not counted as contributors to static power consumption such as clock toggling within the clocking tree and active power consumption for supervisory and control circuits.
- dynamic power consumption in this paper $\texttt{dynPow}_x$ models the maximum application-induced power consumption per component x as an upper bound for power consumption. This is deduced from the difference between the maximum power consumption and the baseline power consumption. The maximum power consumption can be obtained from the operating point where switching activities, off-chip accesses and temperature are maximised. For example, for processing elements, $\texttt{dynPow}_x$ can be obtained from the difference in power consumption between the idle state and active state where the processor is fully utilised while reading/writing to external memory. For networking elements, $\texttt{dynPow}_x$ can be obtained from the difference in power consumption between having no traffic and having full traffic.

$$\begin{aligned}
\texttt{system_power} &= (\sum_{p \in P} \texttt{dynPower}_p + \texttt{statPower}_p) \\
&\quad + \texttt{dynPower}_{NoC} + \texttt{statPower}_{NoC} \\
\texttt{dynPower}_p &= (\sum_{a \in \mathcal{A} | \texttt{proc}_a = p} \texttt{wcet}_a(p, \texttt{proc_mode}_p)) \\
&\quad \cdot \texttt{dynPow}(\texttt{proc_mode}_p) \div \texttt{period}_a \\
\texttt{dynPower}_{NoC} &= \texttt{dynPower}_{NIs} + \texttt{dynPower}_{switches} \\
&\quad + \texttt{dynPower}_{links} \\
\texttt{dynPower}_{NIs} &= \sum_{n \in NIs,\, a \in \mathcal{A}} \texttt{flits}(a, n) \\
&\quad \cdot \texttt{dynPow}_{NI}(\texttt{noc_mode}) \\
&\quad \cdot \texttt{cycleLength}(\texttt{noc_mode}) \div \texttt{period}_a \\
\texttt{dynPower}_{switches} &= \sum_{s \in switches,\, a \in \mathcal{A}} \texttt{flits}(a, s) \\
&\quad \cdot \texttt{dynPow}_{switch}(\texttt{noc_mode}) \\
&\quad \cdot \texttt{cycleLength}(\texttt{noc_mode}) \div \texttt{period}_a \\
\texttt{dynPower}_{links} &= \sum_{l \in links,\, a \in \mathcal{A}} \texttt{flits}(a, l) \\
&\quad \cdot \texttt{dynPow}_{link}(\texttt{noc_mode}) \\
&\quad \cdot \texttt{cycleLength}(\texttt{noc_mode}) \div \texttt{period}_a
\end{aligned}$$

For the static power consumption, $\texttt{statPower}_p$ and $\texttt{statPower}_{NoC}$, the static power consumption parameters of all components, which are determined by processor modes $\texttt{proc\_mode}_p$ and NoC mode `noc_mode`, are summed up. For `systemUsed_power`, which only considers the platform components that are used in the design, all idle processors do not contribute with static power of processor and NI local to the processor. Any switch for which all in- and out-going links are not used in the design does also not contribute to `systemUsed_power`, with neither the static power of the switch nor the links connected to it. The same principle applies to the performance metrics `systemUsed_area` and `systemUsed_cost`.

IV. Experiments

Five experiments have been conducted for this paper, enumerated in Table I. For these experiments, the CP model of our in-house publicly available DeSyDe tool [10], [11] has been extended with a TDN interconnect model and a refined power model, as described in Section III. The tool takes the application graphs with their design constraints, the platform description, and WCET and memory size figures as input parameters, generates the corresponding CP model for the DSE problem, invokes the CP solver and outputs the results in terms of mapping, actor and communication schedules, processor and interconnect configuration and performance data.

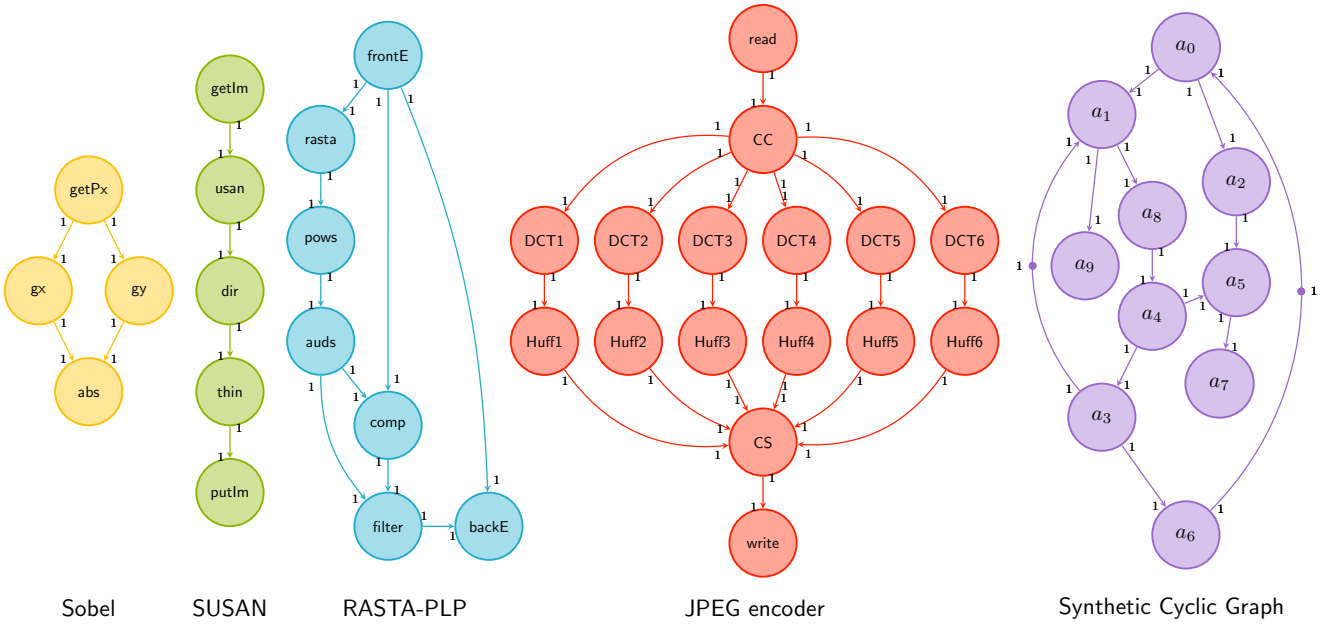


Figure 4. Application graphs used for the experiments.

Table I
CONDUCTED EXPERIMENTS

#	Description
1	validate DSE solution on physical TDN-NoC FPGA platform
2-5	DSE for applications in Figure 4 with the fixed mapping from Figure 5 with different optimization goals:
2	power consumption, one TDN slot/processor
3	power consumption, multiple TDN slots/processor
4	throughput for cyclic graph, one TDN slot/processor
5	throughput for cyclic graph, multiple TDN slot/processor

The application graphs used for the experiments are depicted in Figure 4: four models of streaming media applications and one synthetic graph as an example for a cyclic application graph. The CP environment Gecode [12] is used to create and solve the CP DSE model. For the search, a restart-based search engine with luby restart strategy [13] and no-good collection has been used, running in a single thread in order to enable meaningful no-good collection. The search engine uses random value selection for some of the branching strategies, therefore exploration time can vary between executions. The experiments have executed five different runs with slightly different exploration times, but with consistent results. For optimization, the search engine used branch-and-bound, which posts the result for the optimization criterion of the last found solution as additional constraint on the next solution. The search was combined with an iterative time-out mechanism, that reset the timer to a time-out of 20 minutes after each found solution. The experiments were carried out on a system with an Intel Xeon CPU E3 running at 3.6GHz with 32GB RAM and operating system Ubuntu 16.04.

A. Accuracy of DSE method

The goal of the first experiment, Experiment 1, is to validate the accuracy of the DSE method by running the implementation of a solution produced by the DSE tool on a physical FPGA platform. For this, the applications Sobel and Susan were chosen to run on a 2x2 NoC with two different processor types with two modes each and a TDN-NoC with three modes and four TDN cycles. The platform model reflects a physical system-on-chip (SoC) implemented on a Xilinx Zynq evaluation board, which consists of 1) a dual Cortex ARM A9 processor system as one node instance, available in two modes, maximum frequency and 50% frequency, 2) three instances of a Xilinx Microblaze soft-processor available in two modes, optimized for maximum performance and optimized for minimum area and 3) a Nostrum TDN-NoC [14] available in three modes and varying the flit size: 32 bit, 64 bits and 128 bits. The applications were profiled on all processor types for execution times and memory requirement. The static and dynamic power consumption factors were obtained with the Xilinx Vivado design tool as follows. The static power consumption factor for processor systems is the baseline total power consumed when no-op instructions are executed at room temperature. The dynamic power consumption factor is the average additional observed power consumption at 100% of utilization and at maximum off-chip accesses possible. For the network elements, i.e. NIs, switches and links, the static power consumption factor is the baseline power consumption when no inter-processor communication is occurring. The dynamic power consumption factor is obtained as the additional power consumption at the maximum flit injection rate.

The design goal for DeSyDe in Experiment 1 is to minimize the power consumption, while fulfilling an area constraint of at most 140000 LUTs and satisfying at least a throughput of 1420 images per second for Susan and 3850 images per second for Sobel, corresponding to an iteration period of 13000 cycles and 35000, respectively. The resulting design solution provided by DeSyDe mapped Susan on the ARM

Table II
RESULTS OF EXPERIMENT 1 [1 cycle = $2e^{-8}$s]

		DeSyDe	Xilinx Zynq SoC
Period (cycles)	Sobel	12203	11392
	Susan	31555	29221
Power (10μWatt)		419463	-
Area (LUTs)		139678	139678

processor at maximum frequency and Sobel on all three Microblaze instances optimized for minimum area. The chosen network elements were those supporting flit size of 32 bit, i.e. the smallest in area and the lowest in power consumption. The result of the performance metrics provided by DeSyDe versus the implementation results on the platform is reported in Table II. Area utilisation matches as expected. The TDN assignment was conflict-free and sufficient to reach the requested throughput. The slight overestimation for the iteration period can be accounted to a safety margin of 10% that was added to the actor's profiled execution times to account for potential variation, in lack of access to a method to determine accurate WCETs. Furthermore, due to the high timing cost to obtain accurate power measurements being in the order of micro- to milliseconds, it was not possible to obtain power measurements, while the application is running, without significantly violating the processor utilisation assumptions and invalidating the solution. Therefore, power consumption was not reported. Since the aim is to minimize the power consumption, the accuracy of its absolute value is not deemed as necessary for the experiment as the monotonic consistency of its value relative to other solutions.

B. Scalability of Method

Experiments 2-5 use all five application graphs from Figure 4 with a fixed mapping on a 6x5 NoC with one processor type with two different modes and a TDN-NoC with three modes and six TDN slots. The mapping is shown in Figure 5.

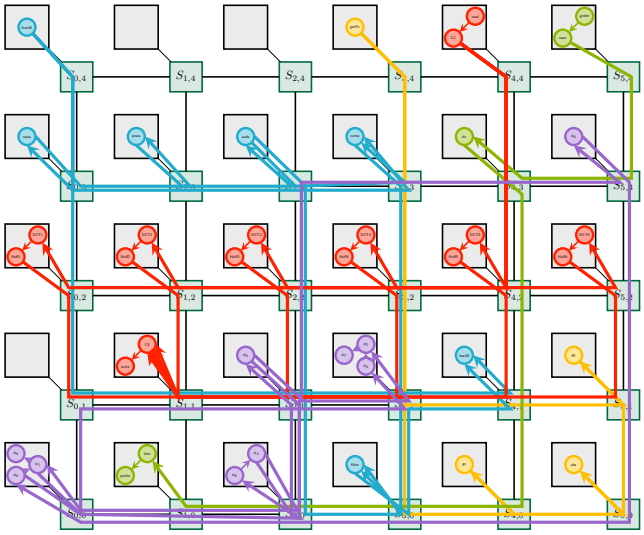


Figure 5. Fixed actor to processor mapping for Experiments 2-5

With a fixed mapping, the DSE has to determine schedules on processors and interconnect, choose a mode for each used processor and the NoC, find a configuration of the interconnect in terms of injection tables for each NI and determine the values of all performance metrics, i.e. power consumption, throughput, utilization, area and monetary cost. The four experiments use the same fixed mapping, but different exploration goals. Experiments 2 and 3 optimize for power consumption, with the difference that the number of TDN slots that can be assigned to each processor is restricted to 1 for Experiment 2, while each processor can use multiple TDN slots in Experiment 3. Experiment 4 and 5 optimize for throughput of the synthetic cyclic graph, and the configuration of assignable TDN slots is analogous to Experiments 2 and 3.

Experiments 2, 3 and 5 were interrupted by the iterative time-out, 20 minutes after the last, i.e. potentially optimal, solution was found. Experiment 4 completed within the time limit for all of its runs, i.e. the found solution was proven to be optimal in terms of iteration period of the synthetic cyclic graph. All runs of each experiment show consistent results in terms of the performance metrics. They differ only with respect to the configuration of the TDN NoC, which is legitimate since different assignments of TDN slots can yield the same performance. Experiments 2 and 4, for which the assignable TDN slots were restricted to one, found exactly one solution for each run. The fact that Experiment 2 could not prove optimality of this solution within the time limit indicates that it is more difficult for the CP model to optimize for power.

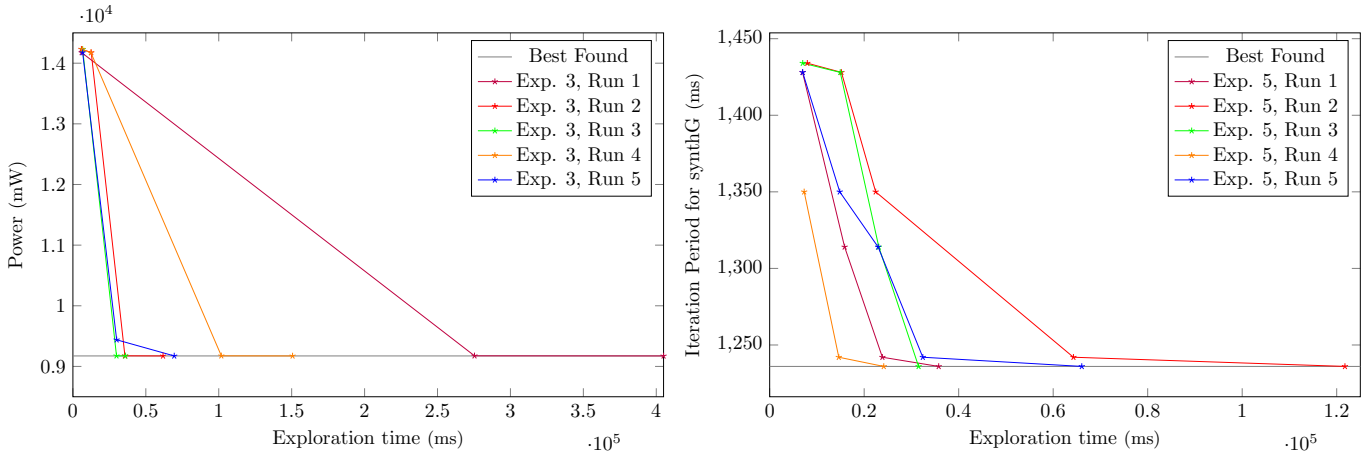
The results of Experiments 3 and 5, which were allowed to assign multiple TDN slots, are shown in Figure 6. For Experiment 3, the first solution was found in average after 6321.2ms (σ=396.08ms), the last solution after 144573.4ms (σ=151772.88ms) and the system power improved by 35.4% (σ=1.3%), corresponding to 5034.4mW (σ=29.6mW). For Experiment 5, the first solution was found in average after 7242.8ms (σ=428.75ms), the last solution after 55844.8ms (σ=40146.6ms) and the iteration period under optimization improved by 12.5% (σ=2.3%), corresponding to 178.8ms (σ=36.35ms).

These results show that efficient solutions for a larger problem with 5 SDF-graphs and a NoC architecture with 30 resource tiles can be found in reasonable time, if a fixed mapping is given. Including the additional problem of finding an efficient mapping into the DSE increases the solution space considerably. An idea to solve these larger problems in reasonable time was presented in [11] where a two-step approach was presented that uses heuristics in the first step followed by CP in the second step.

V. RELATED WORK

The TDN NoC architecture was first described in [2]. The research presented in [15] finds a contention-free slot allocation for TDN-NoCs, however there is no systematic design approach that combines TDN-NoC configuration with mapping, scheduling and performance analysis for throughput, power consumption, area and monetary cost, as the approach presented in this paper.

Mirza et al. [16] also deal with streaming applications on shared predictable multiprocessor NoCs, providing mapping,



(a) Results for Experiment 3

(b) Results for Experiment 5

Figure 6. Results for the optimization of power (a) and iteration period (b) with the fixed mapping from Figure 5.

scheduling and NoC configuration under throughput optimization with CP. The platform model uses a Time-Division-Multiplexing (TDM) NoC, which requires more implementation overhead than a TDN-NoC, and time-slicing schedulers on the processors, while the results of the DSE in this paper can be used for bare-metal solutions without dynamic schedulers. Furthermore, the approach does not support other relevant performance metrics in contrast to the presented work in this paper. The results of the approach in [16] have also not been validated with an implementation on an actual NoC system, as in the experiments of this paper.

The collective work in [17], [18] has shown effective methods to achieve timing guarantees based on a TDM approach along with the minimization of energy consumption for MPSoCs. However, this method requires both injection tables in the network interface and routing tables in the switches, which is a potentially more demanding approach with regards to both design space and hardware costs.

Other approaches that tackle the mapping problem for TDM NoC architectures are [19], [20]. Both approaches use greedy heuristics, while the CP approach in this paper captures the problem as a whole and therefore is trade-off aware, and neither includes analysis for power consumption, area or monetary cost.

VI. Conclusions and Future Work

This paper presented a unified approach for the simultaneous solving of mapping, scheduling, interconnect configuration and performance analysis in terms of timing, power, memory consumption, utilization and area, for multiple mixed-criticality applications on a shared NoC-based multiprocessor. In particular, the correct generation of injection tables is critical for exploiting the full potential of temporally-disjoint NoCs, i.e. providing predictability at a low power budget. The experiments have shown the correctness and flexibility of the approach in terms of support for different exploration goals. Furthermore, the DSE methodology is general and can also be easily adapted to other predictable NoC architectures.

As part of the future work, we plan to take further advantage of the flexibility inherent to the CP-based approach of the DSE tool to make platform design part of the exploration, e.g. for finding a minimal platform that allows to satisfy throughput constraints.

Acknowledgement

This work was funded by the EU H2020 project SAFEPOWER (687902).

References

[1] K. Goossens and A. Hansson, "The Æthereal network on chip after ten years: Goals, evolution, lessons, and future," in *Design Automation Conference*, June 2010, pp. 306–311.

[2] M. Millberg, E. Nilsson, R. Thid, and A. Jantsch, "Guaranteed bandwidth using looped containers in temporally disjoint networks within the Nostrum network on chip," in *DATE'04, Dresden, Germany, March 24-28, 2014*. IEEE, 2004, pp. 890–895.

[3] E. A. Lee and A. Sangiovanni-Vincentelli, "A framework for comparing models of computation," *IEEE TCAD*, vol. 17, no. 12, pp. 1217–1229, Dec. 1998.

[4] E. A. Lee and D. G. Messerschmitt, "Synchronous data flow," in *Proceedings of the IEEE*, ser. 9, vol. 75, Sept. 1987, pp. 1235–1245.

[5] S. Sriram and S. S. Bhattacharyya, *Embedded Multiprocessors: Scheduling and Synchronization*, 1st ed. Marcel Dekker, Inc., 2000.

[6] R. de Groote, P. K. F. Hölzenspies, J. Kuper, and H. Broersma, "Back to basics: Homogeneous representations of multi-rate synchronous dataflow graphs," in *MEMOCODE'13, Portland, OR, USA, October 18-20, 2013*. IEEE, 2013, pp. 35–46.

[7] A. H. Ghamarian, M. Geilen, S. Stuijk, T. Basten, A. Moonen, M. J. Bekooij, B. D. Theelen, and M. Mousavi, "Throughput analysis of synchronous data flow graphs," in *ACSD'06*. ACM, 2006, pp. 25–36.

[8] K. Rosvall, N. Khalilzad, G. Ungureanu, and I. Sander, "Throughput propagation in constraint-based design space exploration for mixed-criticality systems," in *RAPIDO '17*. ACM, January 2017.

[9] K. Ito and K. K. Parhi, "Determining the minimum iteration period of an algorithm," *VLSI Signal Processing*, vol. 11, no. 3, pp. 229–244, 1995.

[10] "DeSyDe," https://github.com/forsyde/DeSyDe, 2017.

[11] K. Rosvall and I. Sander, "Flexible and trade-off-aware constraint-based design space exploration for streaming applications on heterogeneous platforms," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 23, no. 2, 2017.

[12] "Gecode generic constraint development environment," http://www.gecode.org/.

[13] C. Schulte, G. Tack, and M. Z. Lagerkvist, *Modeling and Programming with Gecode*, 2017, http://www.gecode.org/doc-latest/MPG.pdf.

[14] M. Millberg, E. Nilsson, R. Thid, S. Kumar, and A. Jantsch, "The nostrum backbone-a communication protocol stack for networks on chip," in *Intern. Conf. on VLSI Design.*, 2004, pp. 693–696.

[15] Z. Lu and A. Jantsch, "TDM virtual-circuit configuration for network-on-chip," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 16, no. 8, pp. 1021–1034, Aug 2008.

[16] U. M. Mirza, F. Gruian, and K. Kuchcinski, "Mapping streaming applications on multiprocessors with time-division-multiplexed network-on-chip," *Computers & Electrical Engineering*, vol. 40, no. 8, pp. 276–291, 2014.

[17] S. Wildermann, A. Weichslgartner, and J. Teich, "Design methodology and run-time management for predictable many-core systems," in *2015 IEEE International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing Workshops*, April 2015, pp. 103–110.

[18] A. Weichslgartner, S. Wildermann, J. Götzfried, F. Freiling, M. Glaß, and J. Teich, "Design-time/run-time mapping of security-critical applications in heterogeneous mpsocs," in *Proceedings of the 19th International Workshop on Software and Compilers for Embedded Systems*, ser. SCOPES '16. New York, NY, USA: ACM, 2016, pp. 153–162.

[19] A. Hansson, K. Goossens, and A. Rădulescu, "A unified approach to mapping and routing on a network-on-chip for both best-effort and guaranteed service traffic," *VLSI design*, vol. 2007, 2007.

[20] S. Stuijk, T. Basten, M. Geilen, and H. Corporaal, "Multiprocessor resource allocation for throughput-constrained synchronous dataflow graphs," in *DAC*, 2007, pp. 777–782.