Efficient and Exact Design Space Exploration for Heterogeneous and Multi-Bus Platforms

Amna Gharbi
LTCI, Télécom Paris
Institut Polytechnique de Paris
Paris, France
{firstname.lastname}@telecom-paris.fr

Andrea Enrici, Bogdan Uscumlic
Nokia Bell Labs
Nozay, France
{firstname.lastname}@nokia-bell-labs.com

Ludovic Apvrille, Renaud Pacalet
LTCI, Télécom Paris
Institut Polytechnique de Paris
Paris, France
{firstname.lastname}@telecom-paris.fr

***Abstract*—Design Space Exploration of data-flow Systems-on-Chip either focuses on classical shared bus or on complex network-on-chip (NoC) architectures. A lack of research work exists that targets segmented bus architectures. These offer performance improvements (latency, power consumption) with respect to a shared bus, while employing much simpler communication structures and algorithms than a NoC. Despite the lack in the research work, segmented buses are popular in multi-processor systems and in FPGA interconnects. This paper fills this lack with two contributions. First, we propose a Satisfiability Modulo Theory (SMT) formulation. Secondly, we provide a technique to reduce the design-space explosion problem that is portable to other formulations (e.g., ILP, MILP) and to problems where the scheduling on units (e.g., bus, CPU) is multiplexed in time. We integrated these contributions in a state-of-the-art design tool that we employ for evaluation purposes with a set of streaming applications and a MPSoC platform. The resulting framework can study the performance of fixed interconnects as well as determine the optimal architecture among a set of candidates. Our reduction technique improves considerably the scalability of DSE. For our testbench, we reduce the SMT solver run-time from 20 up to 589 times.**

***Index Terms*—Design Space Exploration, scheduling, Satisfiability Modulo Theory, Multi-Bus, FPGA, MPSoC**

I. Introduction

Segmented buses are interconnect architectures that were first proposed in the 90's [1] to increase the performance, communication parallelism and energy savings of shared-bus architectures. A segmented bus offers reduced wiring and load capacitance. This results from the partitioning of a system bus into two or more segments that are interfaced by bridge or switch units (e.g., to connect different clock domains) or buffers. Each segment behaves as a normal bus (e.g., with masters and slaves), shared between a reduced set of local modules. Each segment operates in parallel to other segments and unused segments can be selectively deactivated to save energy. Segmented buses are present in both Multi-Processor System-on-Chip (MPSoC) and FPGA architectures.

Common designs typically segment buses in only two parts: one for high-speed processing units and for low-speed or peripheral units. The main reason for this under-utilization is that communication management is harder than for shared system-buses and is comparable to some configurations of Network-on-Chips (NoCs), e.g., temporally disjoint networks. Inter-segment communications are typically governed by a central arbiter that receives and dispatches signals from/to arbiters that are local to segments. This complicates the inter-dependencies between the problems of mapping (i.e., selecting a spatial allocation of functions to platform units), scheduling (i.e., selecting a temporal allocation for units to execute functions) and interconnect configuration (i.e., number and type of units per segment). Solving these problems is commonly called Design Space Exploration (DSE).

Constraint Programming (CP) is a promising approach for combinatorial problems, such as DSE. It can solve the mapping, scheduling and interconnect configuration sub-problems for orthogonal performance metrics, e.g., latency, power consumption. CP provides complete solutions without the need to combine separate solutions, for each of the sub-problems, sub-optimally. CP also has the advantage that new constraints, cost or performance metrics can be added to an existing formulation, in a modular way, without the need for radical changes.

In this paper, we propose a CP formulation for the DSE problem as Satisfiability Modulo Theory (SMT). Our formulation allows to find a deadline-aware and an optimal solution in terms of latency for multiple applications sharing the same platform. It defines the spatial mapping of tasks to processing elements and communications to routes of segmented buses and derives schedules of tasks and communications on shared platform resources. This formulation is solved by the state-of-the-art solver Z3 [2]. To mitigate the computational costs of our complete search, we propose a static analysis technique that reduces the number of variables created by the solver and their domains. To the best of our knowledge, this is the first exact formulation for segmented bus interconnects that routes data-transfers to different segments and distributes the workload of communications over different bus slots. We integrated our work in a design environment based on UML/SysML diagrams that we used to evaluate our contributions on a testbench of real-world streaming applications and a MPSoC platform.

In the rest of this paper, Section II positions our contribution

The work in this paper is funded by Nokia Bell Labs France. It is part of an academic partnership between Nokia Bell Labs France and Telecom Paris on Models and Platforms for Network Configuration and Reprogrammability.

with respect to related work. In Section III, we present an overview of our DSE approach and the complete SMT formulation. Section IV presents our DSE reduction technique. Section V evaluates our contributions and Section VI provides conclusions and directions for future work.

II. Related work

In this paper, we focus on the problem of allocating in space and time, both computations and communications of a data-flow system subject to timing constraints (deadlines) with a segmented bus interconnect. With respect to platforms with a single shared-bus, our problem is complicated by the management of communications within and between multiple segments with different performance characteristics, the possibility to route communications over different paths composed of more than two segments, the partitioning of a segment time slots to different communications.

As described in [3], optimization techniques like Linear Programming (LP) or Constraint Programming (CP) are based on a set of decision variables, a set of constraints and possibly one or more objective functions to optimize. With respect to LP, a CP model supports only discrete decision variables (integer or boolean) but offers more flexibility to model logical constraints and arithmetic expressions (e.g., modulo, integer division, minimum, maximum). CP has no limitation on the arithmetic constraints that can be set on decision variables and can use ad-hoc constraints (e.g., the "all-different" constraint described in [3]), to accelerate the search of frequently used patterns. Satisfiability Modulo Theory (SMT) is an approach to solve CP problems that is more applicable and flexible than classical mathematical programming techniques. LP is applicable when the problem is specified in terms of linear inequalities and feasible solutions are in the space of a convex polyhedron. Relying only on linear formulations to solve resource-constrained scheduling problems might increase considerably the size of the problem and generate a large list of decision variables which will lead to serious shortcomings [4]. Because of these reasons, recent work explores the use of CP formulations for DSE of systems-on-chip. To the best of our knowledge, we are the first to provide a mathematical formulation for hierarchical pipelined bus interconnects. In the rest of this section, we position our formulation with respect to a selection of recent contributions that target the interconnects of MPSoCs.

An important work that paved the way to SMT techniques for resource-constrained scheduling problems and complex interconnects (multi-hop networks) can be found in [5]. In [6], [7], the authors describe a DSE framework and a SMT solver for mixed critical multi-core architectures using shared memory and message passing for communications. This work formulates a scheduling model that, theoretically, includes the presence of multiple buses. However, contrary to our work, this formulation does not capture the management of communications over multiple buses, e.g., selection of a route among multiple possibilities, splitting transactions over multiple segments. In [8], the authors propose a study that allocates tasks to an overloaded processor to maximize its utilization. An overloaded processor is one with a workload for which, because of dynamic changes in the execution environment, there is no feasible schedule where all tasks can meet their deadlines. While our work focuses on data-flow systems (e.g., image, signal processing), the authors in [8] target control systems where general-purpose processors are interconnected to a single-bus and the execution of control tasks can be preempted.

Relevant studies were conducted by Rosvall et al. in [9]–[11] for mixed-critical applications with both hard and best-effort timing requirements. In [9], [10] the authors study MPSoC architectures with a single Time Division Multiple Access (TDMA) bus. The contribution in [10] extends [9] by considering power consumption as a performance metric (in addition to latency and throughput studied in [9]), improves the formulation with new scheduling constraints and proposes a two-step approach, where constraint programming is applied after heuristics have pruned the design space around promising solutions. In [11], the authors target Temporally Disjoint NoCs, where links can be configured to guarantee the absence of collisions by injecting packets at proper time slots. The formulation in [11] models network links as TDMA shared buses: only one processor at a time can be assigned access to a link; given a link, a processor is assigned a complete TDMA slot. No buffering takes place in switches, between links, and switches implement deflection routing (i.e., in case of link contention, packets with lower priority are "misrouted" to unfavored links, according to a deflection policy). The formulation in [11] can be enriched with our model, where a slot on a bus is shared by multiple processors. The formulation in [11] does not explicitly model routing in order to limit the size of the design space. We also neglect buffering between bus segments, but we explicitly account for routing and explore possible routes between a source and destination processor located on different segments.

The work in [12] is, to the best of our knowledge, the most advanced SMT solution in terms of target platform complexity. It targets clustered multi-core platforms with a Network-on-Chip interconnect. Among other aspects, this formulation models DMA transfers and the buffering within NoC routers. The selection of network routes and contentions are left for future work. As we model the routing of communications and their split over multiple bus slots, ideally, our work fills the gap between formulations for simple single-buses, [9], [10], and complex NoCs, [12].

III. The DSE framework

An overview of the DSE framework is illustrated in Fig. 1. Application and platform models are first created by means of TTool/DIPLODOCUS [13] with UML/SyML diagrams. We selected TTool/DIPLODOCUS as it is free, open-source and targets early designs for which not all low-level details can be provided (e.g., size and policy of caches, platform-specific size of data). The tool offers a good compromise, in terms of abstraction, to *program* MPSoC platforms for users (e.g., IT engineers) that are not application experts or do not have deep

knowledge of high-performance embedded systems.
In Fig. 1, UML/SysML diagrams are parsed, scanned and analyzed in order to automatically extract an SMT formulation. The latter is given as input to the state-of-art SMT solver Z3 [2] which returns a solution model if the formulas are satisfiable. The output model is converted back to UML/SysML for animation purposes: the initial diagrams are animated with the mapping solution, Gantt charts are created to illustrate the scheduling of tasks and communications.

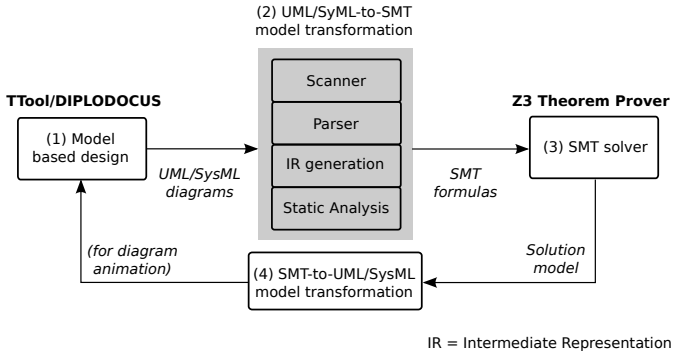


Fig. 1. Overview of the Design Space Exploration workflow.

Applications are denoted in TTool/DIPLODOCUS with UML Activity and SysML Block diagrams. These capture the data and control dependencies between tasks (i.e., units of work) as well as the internal behavior of each task (state machine). Execution costs of UML Activities are represented by an operator that denotes the algorithmic complexity of an Activity. This operator denotes the number of steps (e.g., machine instructions, lines of code) that are necessary to compute a task. The timing duration of a step in clock cycles is specified in the architecture diagram, for each unit.
Architectures are modeled with a UML Deployment diagram. This diagram captures the topology of a platform as well as the performance (e.g., bus bandwidth, number of CPU cores, memory size) of both the hardware circuitry and the software stack that compose a unit at Electronic System-Level of abstraction. In the rest of the paper, we refer to generic execution units as Processing Elements (PEs). PEs dispose of Direct Memory Access (DMA) units to transfer information on bus interconnects. UML/SysML diagrams are transformed in a graph-based **intermediate representation** to extract a SMT formulation for the mapping and scheduling problems (step 2 in Fig. 1). The architecture diagram is converted in an undirected graph $G_{arch} =< U, L >$ where nodes represent processing units as well as communication units (buses and bridges) that are annotated with performance parameters. Each application diagram is converted into a dependency graph $G^A_{app} =< T^A, C^A >$, where T^A is the set of tasks (UML Activities) and C^A the set of edges denoting inter-task dependencies, for application A. Edges that correspond to data dependencies are labeled with a positive integer that models the size of the First-In-First-Out (FIFO) buffer associated. In our models, if A is a periodic application, we require the user to explicitly represent each period under study as a sub-graph in G^A_{app}. We also require the user to explicitly model control dependencies between tasks that belong to different periods (e.g., to force the execution of tasks in period n before tasks in period $n+1$). Our SMT formulation is based on the following **design assumptions**:

- Buffers and tasks are not allowed to migrate at run-time.
- Each PE disposes of a local private memory. Tasks allocated to a PE read and write in its local memory. Buffers are alive as long as producer or consumer tasks are alive.
- Each PE can execute only one task at a time, without preemption, until the task completion.
- Latency for reading/writing data from/to a PE local memory is considered as negligible
- We consider the absence of deadlocks or livelocks on the interconnect and the absence of losses during a transfer.

A. The SMT problem formulation

In this subsection, we describe our notations and the entire SMT formulation. As modularity and extensibility are typical of constraint programming approaches, our formulation can be extended with new constraints, cost or performance metrics without the need to be entirely recoded and especially without changing the solving technique. This property makes the CP approach advantageous compared with heuristics where problem definition and solving are tightly coupled: a slight change in the problem definition might require reconfiguring pre-coded heuristic to find a viable solution.
We consider time to be divided in atomic slots that are used as a time-line to determine a scheduling. The length of a slot, l_{slot}, impacts the granularity and the computational cost (runtime) of the DSE process. The value of l_{slot} determines a compromise between precision and execution time of the DSE process. We specify to the reader that the values of variables that are related to scheduling aspects are integer multiples of l_{slot} (e.g., deadlines). The notations for our formulation are described in Table I. Variables and constraints apply to the set of all application graphs $A \in \mathcal{A}$.

1) Decision variables: We define a boolean decision variable to denote the task-to-PE mapping:

$$\forall t \in T^A, \forall c \in F(t),\ z_{t,c} = \begin{cases} 1 & \text{if t is mapped to c,} \\ 0 & \text{otherwise.} \end{cases} \quad (1)$$

Variable $x_{t,p}$ denotes the task-to-cluster mapping:

$$\forall t \in T^A, \forall p \in F'(t),\ x_{t,p} \in \{0,1\} \quad (2)$$

$y^{m,n}_{i,j,\rho_{i,j}}$ indicates the mapping of a data dependency to a route in the platform. It allows multiple routes exploration.

$$\forall c^A_{m,n} \in C^A, \forall i,j \in F'(m) \times F'(n), \forall \rho_{i,j} \in R, y^{m,n}_{i,j,\rho_{i,j}} \in \{0,1\} \quad (3)$$

$start_t$ indicates the starting time slot of task t. $start_t$ is a positive integer upper-bounded by N^A, as a deadline requirement should be respected for a valid deployment.

$$\forall t \in T^A, start_t \in \{0, .., N^A\} \quad (4)$$

TABLE I
NOTATIONS FOR THE SMT FORMULATION

Notation	Definition
$\mathcal{A}$	set of the application dependency graphs.
$deadline_A$	deadline of an application $A \in \mathcal{A}$.
$t \in T^A$	a task from application A.
bin_t, $bout_t$	total amount of input and output data that are consumed and produced by one execution of task t.
$c^A_{m,n} \in C^A$	a communication between tasks m and n in application A.
$d_{c^A_{m,n}}$	the amount of data to transfer on $c^A_{m,n}$.
c	a processing element (e.g., a CPU core, a hardware accelerator).
p	in case a PE models a stand-alone mono-core unit, p models it. In case a PE is part of a multi-core unit, p denotes the cluster of PEs that encapsulates all of the unit cores.
$F(t)$	a function that returns the set of PEs (c) where task t can execute.
$F'(t)$	a function that returns the set of processing blocks (p) where task t can execute.
mem_c	amount of available space in the local memory of c.
$\rho_{i,j}$	a route in the architecture for the transfer of data between PEs i and j. It is defined as a path in the architecture graph that starts from the DMA of PE i and terminates at local memory of j, or vice versa.
R	the set of routes available in target platform.
$B \subset U$	the set of buses b in the platform.
$R_{b1,b2}$	the set of routes including the bus-to-bridge-to-bus $b1, b2$ fragment.
bw_b	the bandwidth per atomic time slot of bus b.
$pt(t,c)$	return the processing time of task t on PE c. It includes the time needed to read/write data in local memory of c.
N^A	an upper bound on the number of time slots that are allocated to any unit in the platform, for any application $A \in \mathcal{A}$. It is defined as $\max_{A \in \mathcal{A}}\{deadline_A, A \in \mathcal{A}\}$.
$s \in \{0, ..., N^A\}$	the index of a time slot.
l_{slot}	the length of an atomic time slot.

Variable $e^{m,n}_{b,s}$ denotes the amount of data transferred through bus b at time slot s for communication from task m to task n.

$$\forall c^A_{m,n} \in C^A,\ \forall b \in B,\ \forall s \in \{0, .., N^A\},\ e^{m,n}_{b,s} \in \{0, .., bw_b\} \tag{5}$$

2) Constraints: Constraints (6) and (7) and (8) ensure a valid physical mapping of a task t. In (6), a unique and feasible task-to-PE mapping is granted. Formula (7) indicates that task t is assigned to the cluster p including processing element c. Constraint (8) states that a task can be mapped on a processing element only if the latter provides enough memory to accommodate its input and output data.

$$\forall t \in T^A,\ \sum_{c \in F(t)} z_{tc} = 1 \tag{6}$$

$$\forall t \in T^A, if\ (\sum_{c \in p} z_{tc} \geq 1)\ then\ x_{tp} = 1\ else:\ x_{tp} = 0 \tag{7}$$

$$\forall t \in T^A,\ \forall c \in F(t),\ (bin_t + bout_t) z_{tc} \leq mem_c \tag{8}$$

Constraint (9) ensures a unique and valid route is assigned to communication $c^A_{m,n}$ when tasks m and n are mapped respectively to processors i and j.

$$\begin{aligned} &\forall c^A_{m,n} \in C^A,\ \forall i, j \in F'(m) \times F'(n), i \neq j, \\ &if((x_{m,i} = 1) \wedge (x_{n,j} = 1))\ then \sum_{\rho_{ij} \in R} y^{m,n}_{i,j\rho_{ij}} = 1 \\ &\qquad else: \sum_{\rho_{ij} \in R} y^{m,n}_{i,j\rho_{ij}} = 0 \end{aligned} \tag{9}$$

Constraint (10) ensures that a single task executes on a mono-core processing element at a time. end_t denotes the end of execution of task t in time slots. It is calculated using (11).

$$\begin{aligned} &\forall t, t' \in T^A, t \neq t',\ \forall c \in F(t) \cap F(t'), \\ &\neg((z_{tc} = 1) \wedge (z_{t'c} = 1)) \vee ((start_{t'} \geq end_t) \vee (start_t \geq end_{t'})) \end{aligned} \tag{10}$$

$$\forall t \in T^A,\ end_t = start_t + \sum_{c \in F(t)} pt(t,c) z_{t,c} \tag{11}$$

Constraint (12) guarantees precedence relations between pairs of producer-consumer tasks (i.e., tasks related by a data dependency in application graphs).

$$\forall c^A_{m,n} \in C^A,\ start_n \geq end_m \tag{12}$$

We also implement task symmetry breaking constraints to enforce a lexicographic ordering of identical tasks as in [12]. Remaining constraints describe the time-slots allocation for transfers on a route of multi buses. When a bus b is selected to explore the possibility of being allocated for a data transfer, the bandwidth of b (per time slot), bw_b, can be allocated entirely to a data transfer or shared between different data transfers. Because the buses that form a route $\rho_{i,j}$ can have different capacities, our modeling proposes that all buses on a route operate at the capacity of the slowest bus. This avoids buffer overflows in bridges and is consistent with our assumption that data are not lost in the interconnect. The residual bandwidth per slot (i.e., difference between the bandwidth of a bus and that of the route slowest bus) can be allocated to other transfers on routes that partially overlap with $\rho_{i,j}$.

Constraints (13) and (14) ensure respectively for a bus to respect its capacity, and for a transfer, to assign enough time slots on a route of bus segments to transfer the required amount of data.

$$\forall b \in B,\ \forall s \in \{0, .., N^A\},\ \sum_{c^A_{m,n} \in C^A} e^{m,n}_{b,s} \leq bw_b \tag{13}$$

$$\begin{aligned} &\forall b \in B,\ \forall c^A_{m,n} \in C^A, \\ &\sum_{s \in \{0,..,N^A\}} e^{m,n}_{b,s} = d_{c^A_{m,n}} \times \sum_{i,j \in F'(m) \times F'(n)} \sum_{\rho_{i,j} \in R,\ b \in \rho} y^{m,n}_{i,j,\rho_{i,j}} \end{aligned} \tag{14}$$

Besides inter-task precedence constraints, a precedence constraint must also be enforced between a computation-communication pair: (i) a data transfer cannot start before producer task has completed execution and (ii) a consumer

task can start as soon as data transfer is completed. Given a data transfer, constraints (15) allocate no bus bandwidth in time slots that precede the end of producer task m or follow the start of consumer task n.

$$\begin{aligned} &\forall c^A_{m,n} \in C^A,\ \forall b \in B,\ \forall s \in \{0,..,N^A\}, \\ &\quad (s \geq end_m) \vee (e^{m,n}_{b,s} = 0) \\ &\quad (s < start_n) \vee (e^{m,n}_{b,s} = 0) \end{aligned} \tag{15}$$

Constraint (16) corresponds to the forwarding of data between two bus segments (i.e., at bridges level). Data transferred on bus b_1 at time slot s are transferred on next bus b_2 at time slot $s+1$. Note that we use the subset of routes $R_{b1,b2}$ defined in Table.I.

$$\begin{aligned} &\forall c^A_{m,n} \in C^A,\ \forall s \in \{0,..,N^A\},\ \forall b_1, b_2 \in B \\ &\sum_{\rho_{i,j} \in R_{b1,b2}} y^{m,n}_{i,j,\rho_{i,j}} = 1 \implies e^{m,n}_{b_2,s+1} = e^{m,n}_{b_1,s} \end{aligned} \tag{16}$$

B. Discussion

We discuss here some precisions concerning our formulation. In the latter, communications are performed in parallel to computations by dedicated Direct Memory Access (DMA) engine. These operate between a source PE local memory and a destination PE local memory, regardless the bus segment where PEs are located. Constraint (16) ensures that data are transferred across bus segments in a pipelined mode, without buffering: a data packet being transferred on bus n at slot s is forwarded, in-order, on bus $n+1$ at slot $s+1$. This corresponds to segments interfaced by router or switch units, without buffering. This configuration can be found in segmented buses as well as in Temporally Disjoint Networks [11].

Constraints (9), (13-16) relate to the routing and duration of data transfers. They account for the presence of concurrent transfers, the sharing of segments bandwidth and the allocation of slots to multiple transfers. Similar constraints are also common to the behavior of NoCs, which makes these formulas portable to target these more complex interconnects.

We highlight to the reader that function $F'(t)$ returns different units according to the type of PE where a task t can be mapped. This differentiation allows our formulation to reduce the number of variables that are allocated to study communications to/from t. In fact, a PE models either a stand-alone single-core unit (e.g., micro-controller, IP block) or a core within a multi-core unit (e.g., a Digital Signal Processor). In this second case, variables for communications must be allocated to consider the reception and dispatch of data at the level of abstraction of the entire multi-core unit rather than for each core (we neglect inter-core communications).

In terms of portability, our formulation can be exported to frameworks that operate at levels of abstraction lower than TTool/DIPLODOCUS (e.g., SystemC, cycle-accurate simulators). An example of such a framework where related work on SMT was integrated is ForSyDe [14]. Note that a framework abstraction level determines the coarseness of the information that is assigned to SMT variables. For instance, let us consider the value returned by function $pt(t,c)$ in Equation 11. In DIPLODOCUS, this value is the algorithmic cost of the UML Activity for task t multiplied by the number of cycles taken by the PE c to perform t. In ForSyDe, the same function returns the Worst-Case Execution Time of t on PE c.

IV. The Design Space Reduction Technique

Our holistic SMT formulation completely explores a design space. However, such an approach, suffers from the well-known space explosion problem that often leads to unacceptable exploration cost and scalability issues. Time slotted formulations allow to accurately analyze and schedule concurrent access to shared resources. However, they suffer from the combinatorial explosion of the number of variables required to study the scheduling of time-shared resources, in *all* slots, i.e., $s \in \{0,...,N^A\}$. In this section, we provide a solution to reduce the design space ***without compromising the quality of solutions***.

A. Description of the DSE Reduction Technique

Our solution consists in performing a ***static analysis*** (Fig. 1) on the dependency graph prior to the exploration. The purpose of this analysis is to determine ***temporal boundaries*** that narrow down the number and domains of some scheduling-related variable. These boundaries are, for a task t, the earliest start time (ES_t), the earliest finish time (EF_t), the latest start time (LS_t) and the latest finish time (LF_t). The analysis is based on the dependencies between tasks, the characteristics of units where a task can be potentially mapped, the deadline of applications (the application latest finish time LF_A) and an application earliest start time (ES_A). By default, the latter is null. It can be greater than zero in case of pipelined applications, where the execution of the initial task can be delayed because of scheduling constraints.

We implemented this static analysis based on the Critical-Path Method (CPM) [15] that is a collection of algorithms commonly used in project planning in many domains. CPM allows to compute the longest path of planned activities and to identify critical activities. We reuse two algorithms from CPM: the *forward pass* and the *backward pass*.

1) Forward Pass: It calculates the tuple <ES_t, EF_t> . ES_t is computed based on input dependencies: the earliest time at which t can start is when all its predecessors have finished. Formally, $ES_t = \max_{i \in \{1,..,k\}} \{EF(t_i)\}$, where $\{t_1, ...t_k\}$ is the set of predecessor tasks for t. The forward pass starts exploring a dependency graph G^A_{app} at the source, so that ES_{source} is assumed to be equal to ES_A. Then it browses G^A_{app} forward: once ES_t is computed for task t, EF_t is calculated as $ES_t + pt(t,c)$ (we remind that $pt(t,c)$ is the processing time of t on PE c). We select c as the fastest execution unit where t can be mapped: this allows the shallowest reduction of the domain interval, thus including *all* possible mappings.

2) Backward Pass: It calculates the tuple <LS_t, LF_t>. LF_t is first computed based on output dependencies. The backward pass examines G^A_{app} at the sink task, so that LF_{sink} corresponds to LF_A. Then it browses G^A_{app} backward until the source. A task t must complete before the earliest LS

of all its successors, hence $LF_t = \min_{i \in \{1,..,u\}} \{LS(t_i)\}$, where $\{t_1, ...t_u\}$ is the set of successor tasks for t. LS_t corresponds to $LF_t - pt(t, c)$, where c is also the fastest PE, where t can execute, as it provides the *latest* start and finish times. Selection of the fastest c provides the widest temporal boundaries thus reducing the variables domain without excluding feasible solutions.

Computing tuples $<ES_t, EF_t>$, $<LS_t, LF_t>$ for all tasks with the forward and backward pass simply requires to traverse the dependency graph G^A_{app}. Hence complexity is that of common graph traversal algorithms (e.g., depth-first), $O(T^A + C^A)$.

We apply two reductions. The first reduction concerns the definition domain of variables $start_t, \forall t$, initially defined in (4), to the following domain:

$$\forall t, start_t \in \{ES_t, ..., LS_t\} \tag{17}$$

The second reduction concerns the set of created variables $e^{m,n}_{b,s}$, initially defined in (5), to the following subset:

$$\forall c^A_{m,n} \in C^A,\ \forall b \in B,\ \forall s \in \{EF_m + 1, ..., LS_n - 1\},\ e^{m,n}_{b,s} \in \{0, ..., bw_b\} \tag{18}$$

These optimizations reduce as well the number of constraints (13-16) and result in a smaller problem size.

Unlike other heuristics our reduction technique improves scalability without any compromise on optimality. Our analysis refines the input formulation of the DSE problem. This allows solvers to create a smaller internal representation that requires less time to be explored. Our technique is not restricted to the specific formulation in this paper, nor to SMT formulations, in general. It can be applied to Linear Programming, to other solvers (e.g., IBM Cplex) and does not depend on the DIPLODOCUS abstraction level. It is applicable to any variant of the mapping-scheduling problem, where there is at least one resource (e.g., bus, PE) for which scheduling is multiplexed in time.

Concerning related work on reduction techniques, symmetry breaking is a family of techniques that is commonly used in constraint programming (see [12], [16]) to avoid searching through isomorphic solutions. In our formulation we already use task symmetry breaking, as in [12], to improve solver runtime. However, relying only on task symmetry breaking was not enough to significantly improve scalability. As opposed to homogeneous architectures, processor symmetry breaking on heterogeneous PEs is very limited. Hence, we needed new reduction techniques for the DSE problem. Our technique allows to improve considerably the scalability regardless of the platform type. To the best of our knowledge, this is the first generic and efficient reduction technique.

V. Experimental Evaluation

In this section we evaluate our contributions for the DSE of multi bus platforms. We borrow the same workload of dataflow applications as that in [9], Fig. 2. We select their testbench as their work is the closest to ours in terms of abstraction level and design assumptions. In all tests, time units for latency and processing times are expressed in clock cycles and data units (du) are used to express communication volumes. We set the length of a single slot l_{slot} to 1 clock cycle to allow for the most accurate analysis. In Fig. 2, tasks are annotated with execution times reported in [9]. To account for platforms with heterogeneous PEs, we assume the processing times of tasks on DSPs and that on hardware accelerators to respectively 1/5 and 1/50 of the times reported.

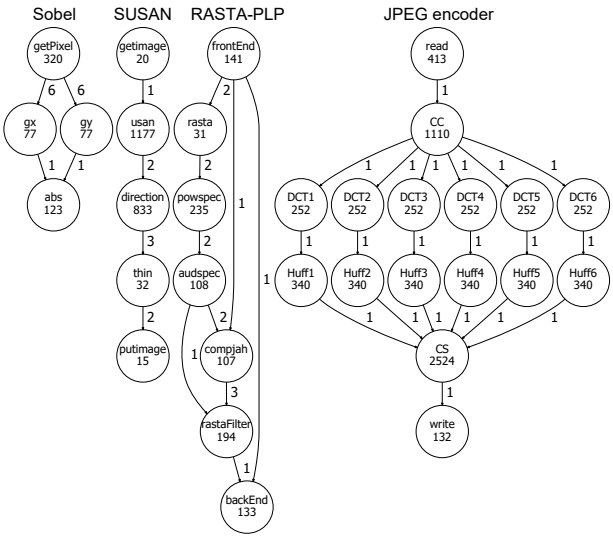


Fig. 2. The dependency graphs of our testbench. Edges are labeled with the size of buffers. Tasks are labeled with their processing times.

A. Optimal interconnect exploration

We first show how our contributions assist a user in selecting an optimal interconnect structure, so that overall system latency is minimized. We consider 4 instances of application *SUSAN* in Fig. 2. Target architecture contains a quad-core CPU and four sets of PEs, each composed of 1 DSP and 2 hardware accelerators (*usan* and *direction*). We consider three architecture instances: (*Architecture 1*) where all PEs are connected to a single shared bus, (*Architecture 2*) such that only the CPU is connected to a shared bus. The latter interfaces to 4 segments (Fig. 3), each with one DSP and two hardware accelerators. In these two instances, all buses have a bandwidth of 16 dus. We consider a third instance (*Architecture 3*), identical to the second, where the shared bus segment has a bandwidth of 32 dus.

In all cases, the optimal solution assigns tasks *getImage* and *putImage* of each *SUSAN* instance to a CPU core and tasks *usan*, *direction* and *thin* to DSPs and accelerators.

Fig. 4 shows the DSE results for instances where the size of data produced/consumed in SUSAN in Fig. 2 is equal to 32 data units. Here, *Architecture 1* solution includes the lowest and the highest achievable values of latency. Even though four applications are equivalent and parallel computing resources are allocated, we notice a gap of 22 cycles between the two extreme latency values (respectively 104 cycles for *susan2* and

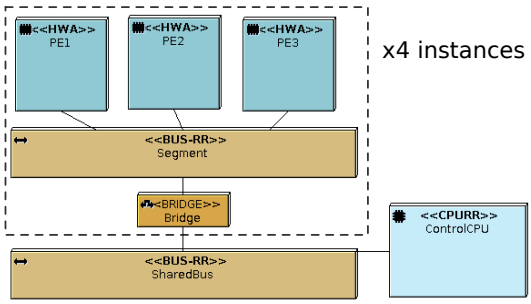


Fig. 3. A template for the target architectures evaluated in sub-section V-A.

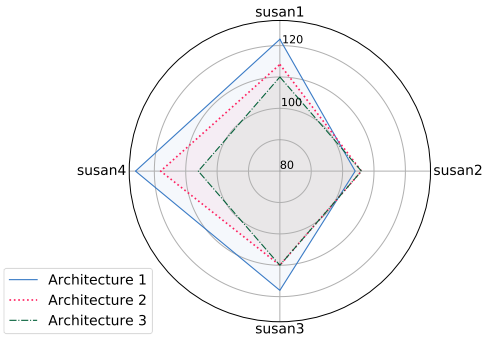


Fig. 4. Comparison of the optimal latency (in cycles) for a workload of 4 instances of the SUSAN filter in Fig. 2 on three architectures with different interconnects.

126 cycles for *susan 4*). This gap is explained by contention on bus due to competition of parallel data transfers for the single shared bus. *Architecture 2* provides more balanced latencies, with a latency gap equal to 12 cycles (respectively 106 cycles for *susan 2* and 118 cycles for *susan 4*). In Fig. 4, the diamond for *Architecture 1* encapsulates that of *Architecture 2*, which indicates that *Architecture 2* yields lower latencies. However, the gap between the shortest and the biggest latency values (12 cycles) still demonstrates contention on shared bus segment. *Architecture 3* solution reduces further contention (the gap is only equal to 4 cycles) as well as latencies. Thus, our DSE framework suggests to the user that a segmented interconnect, where segments have a bandwidth of 16 dus and the shared bus a bandwidth of 32 dus is the best architecture instance to the studied workload.

B. Performance evaluation experiment

In this sub-section, we evaluate the performance of our DSE formulation before and after the optimization that reduces the design space explosion problem (Section IV). Similarly to [9], we consider a set of workloads for all combinations of the applications in Fig. 2 and assume data produced/consumed have a size of 8 dus. Our results refer to solutions that guarantee *all* applications deadlines. In fact we are mainly interested in time-sensitive systems where a timing guarantee must be given for the system to work correctly (e.g., real-time systems). These systems are not required to respond as fast as possible (i.e., minimize $latency$) but rather have to respond fast enough so that timing requirements are respected (i.e., $latency \leq deadline$). Deadlines were set to 490 cycles (*Sobel*), 1170 cycles (*SUSAN*), 575 cycles (*RASTA-PLP*) and 1830 cycles (*JPEG*). These values correspond to the time requirement to complete one execution for the applications in Fig. 2. The target platform is an MPSoC including 1 quad-core CPU, 8 DSPs and 1 hardware accelerator. Its interconnect is a bus with 3 segments. The CPU is connected to a segment as in Fig. 3 and others PEs are disposed on the two other segments.

We set a timeout of 1800 seconds after which we stop exploration if no solution was found. Based on our experience, this is the average amount of time that a user is willing to await for the results of DSE, for early designs, at high abstraction levels such as that of TTool/DIPLODOCUS. Fig. 5 shows run-time evolution with a *logarithmic scale*. From Fig. 5, it can be seen that the reduction decreases the run-time by orders of magnitude. The red curve indicating run-time before applying reductions shows that the solver starts to violate the timeout with workload *jpeg* because of the large number of variables created for all slots. This violation results in the impossibility to study 8 out of 15 workloads within the fixed timeout. In contrast, the blue curve in Fig. 5 shows the benefits of our reduction technique. The solver run-time increases very slowly with the increase of workloads complexity. The run-time values vary in the range $[10^{-2}, 10]$ seconds. Focusing only on the heaviest workload (*sosurajp*), we noticed that the dimension of our problem (number of variables and constraints) was reduced drastically: number of variables decreased from 13K to 3K and number of constraints decreased from 230K to 17K. This demonstrates the effectiveness of our reduction techniques to reduce the solver run-time and improve scalability of DSE.

Fig. 6 shows the solver run-time for a fixed workload, *sosurajp* (heaviest workload in our testbench), as a function of l_{slot} that varies from 1 cycle to 10 cycles. This curve represents the impact of the DSE granularity on the solver run-time and allows to appreciate the impact of our reduction technique. By considering the same timeout as the one for Fig. 5, 1800 seconds, the curve plotting run-time evolution without our reduction technique shows that it is not possible to explore the design space with a granularity smaller than 3 cycles/slot. On the contrary, the curve plotting results of the optimized formulation shows that the latter allows to study the workload up to the maximum precision (l_{slot} = 1 cycle) within only 8.2 seconds.

C. Optimality gap analysis

A third experiment was conducted to evaluate the quality of the deadline-aware solution with respect to the optimal solution. Since the initial formulation times out to find feasible solutions for heavy workloads, we only use here the formulation with the reduction technique to find optimal solutions. We precise that we use a simple iterative algorithm to reduce

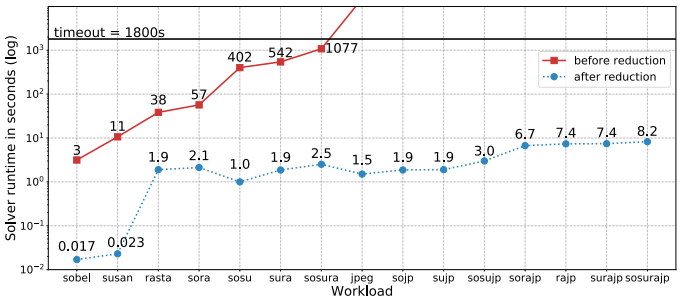


Fig. 5. Solver run-time to produce a deadline-aware solution, as a function of different workloads using initial formulation and optimized formulation, 1 slot = 1 cycle.

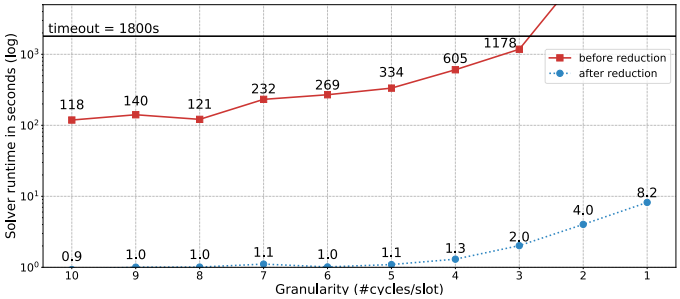


Fig. 6. Solver run-time to produce a deadline-aware solution, as a function of exploration granularity using initial formulation and optimized formulation for workload *sosurajp*.

incrementally latencies until finding optimal solutions. Table II shows, for workload *sosurajp*, the optimality gap evolution and the required time for the solver to produce optimized solutions that *co-minimize* latencies of *all* workload applications. From the table we can see that the first solution found by the solver within 8.2 seconds (same as in Fig. 5) is far from the optimal solution between 0.41% for *Sobel* and 6.28% for *RASTA-PLP*. Table II shows that the optimal latencies are reached for *Sobel*, *SUSAN*, *RASTA-PLP* and *JPEG* respectively after 68.91s, 402.91s, 615.2s and 660.07s. Hence, the *global* optimal solution (i.e., 0% gap for all applications) was also found before the time limit.

All experiments were conducted on a Linux notebook with Intel Core $i5$ processor at 2.50 GHz with 16 GB of RAM memory, running release 4.8.7 of Z3.

VI. Conclusions and Future Work

In this paper we presented a SMT formulation for the Design Space Exploration of data-flow systems on heterogeneous architectures with segmented bus interconnects. To the best of our knowledge, this is the first formulation to target multi-bus interconnects. To mitigate the design space explosion problem, we also proposed a solution that statically analyzes the application dependency graphs and narrows down the domains of variables related to the scheduling of functions on time-shared resources. We integrated these contributions in a UML/SysML design tool and illustrated how the resulting framework holistically studies the mapping, scheduling and

TABLE II
Optimality gap evolution in function of solver run-time for workload sosurajp using optimized formulation. Timeout of 1800s.

Optimality gap				Run-time (s)
sobel	susan	rasta	jpeg	
0.41%	2.63%	6.28%	0.49%	8.2
0.20%	2.46%	6.28%	0.49%	44.78
0%	2.28%	6.28%	0.49%	68.91
0%	1.67%	5.91%	0.49%	150.62
0%	0.35%	4.81%	0.49%	357.04
0%	0%	4.62%	0.49%	402.91
0%	0%	3.88%	0.49%	439.38
0%	0%	2.96%	0.49%	480.18
0%	0%	1.85%	0.49%	527.36
0%	0%	1.48%	0.44%	551.09
0%	0%	1.11%	0.38%	572.59
0%	0%	0%	0.38%	615.20
0%	0%	0%	0.22%	634.62
0%	0%	0%	0.11%	647.92
0%	0%	0%	0%	660.07

interconnect configuration, i.e., scheduling and routing of communications within and between segments, partitioning of a segment time slots to different communications. An interesting direction for future work is to the inclusion of a power model to investigate the interconnect power consumption.

References

[1] J. Y. Chen, W. B. Jone, J. S. Wang, H. . Lu, and T. F. Chen, "Segmented bus design for low-power systems," *IEEE Trans. on VLSI Systems*, vol. 7, no. 1, pp. 25–29, 1999.

[2] L. de Moura and N. Bjørner, "Z3: An Efficient SMT Solver," in *TACAS*, 2008, pp. 337–340.

[3] IBM, "Mathematical programming vs constraint programming," http://ibmdecisionoptimization.github.io/docplex-doc/mp_vs_cp.html.

[4] T. Davidović, L. Liberti, N. Maculan, and N. Mladenovic, "Mathematical programming-based approach to scheduling of communicating tasks," 01 2005.

[5] W. Steiner, "An Evaluation of SMT-Based Schedule Synthesis for Time-Triggered Multi-hop Networks," in *RTSS*, 2010, pp. 375–384.

[6] S. Voss and B. Schätz, "Deployment and scheduling synthesis for mixed-critical shared-memory applications," in *ECBS*, 2013, pp. 100–109.

[7] S. Voss, J. Eder, and F. Hölzl, "Design Space Exploration and its Visualization in AUTOFOCUS3," in *SE Workshop*, vol. 1129, 2014, pp. 57–66.

[8] Z. Cheng, H. Zhang, Y. Tan, and Y. Lim, "SMT-Based Scheduling for Overloaded Real-Time Systems," *IEICE Transactions on Information and Systems*, vol. E100.D, pp. 1055–1066, 2017.

[9] K. Rosvall and I. Sander, "A constraint-based design space exploration framework for real-time applications on MPSoCs," in *DATE*, 2014, pp. 1–6.

[10] K. Rosvall and I. Sander, "Flexible and Tradeoff-Aware Constraint-Based Design Space Exploration for Streaming Applications on Heterogeneous Platforms," *ACM TODAES*, vol. 23, no. 2, pp. 21:1–21:26, 2017.

[11] K. Rosvall, T. Mohammadat, G. Ungureanu, J. Öberg, and I. Sander, "Exploring Power and Throughput for Dataflow Applications on Predictable NoC Multiprocessors," in *DSD*, 2018, pp. 719–726.

[12] P. Tendulkar, "Mapping and Scheduling on Multi-core Processors using SMT Solvers," Ph.D. dissertation, University of Grenoble, 2014.

[13] TTool, http://ttool.telecom-paristech.fr/diplodocus.html, 2006.

[14] I. Sander, "System Modeling and Design Refinement in ForSyDe," Ph.D. dissertation, Royal Institute of Technology (KTH), 2003.

[15] J. E. Kelley and M. R. Walker, "Critical-path planning and scheduling," in *IRE-AIEE-ACM*, 1959, pp. 160–173.

[16] E. Kang, E. Jackson, and W. Schulte, "An Approach for Effective Design Space Exploration," in *Foundations of Computer Software. Modeling, Development, and Verification of Adaptive Systems*, 2011, pp. 33–54.