A Novel Modeling Approach for System-Level Application Mapping Targeted for Configurable Architecture

Abstract—Advances in chip fabrication technology and increasing demand for meeting time to market has led to the use of the electronic system-level (ESL) design methodology. An important challenge is to find a proper approach by which a given application can be mapped into a specific target architecture, usually called application mapping. In this paper, an abstract modeling approach based on the colored Petri net (CPN) is proposed to map an arbitrary application into a given target architecture. The mapping is at an abstract level and contains timing information that facilitates high-level design space exploration. A complete stepwise procedure is presented to illustrate how a CPN model of the application is refined and employed to extract the required tasks for a given target architecture. The fact that models obtained as such are executable makes the required performance evaluation and exploration possible, and thus, will result in better architectural decisions at the design stage. The usefulness of the proposed approach is assessed using several alternative mapping schemes of the JPEG encoder. The actual encoding time and the required simulation time indicate the advantages of this method over design-space exploration that would otherwise be done in SystemC, which is the natural choice in today's ESL designs.

Index Terms—Application mapping, colored Petri net (CPN), configurable architecture, design-space exploration, electronic system level (ESL), system-level modeling.

I. INTRODUCTION

Continues progress in the state of the art of the chip fabrication technology opens new major aspects in the digital system design realm. Traditional design methodology limitations in modern sophisticated systems, the ever-increasing pressure of meeting tight time to market, and the growing verification costs of digital circuits are just several challenges that force designers to move toward high-level abstractions in different domains, such as modeling and simulation [1].

Register transfer level (RTL) descriptions that are vastly used in digital system designs could neither follow the great advances in the semiconductor technology, nor properly manage the inherent complexity in modern complicated very large-scale integration circuits [2], [3]. Electronic system-level (ESL) design methodology is one of the promising approaches in this era. ESL is the utilization of appropriate abstractions to increase comprehension about a system and to enhance the
probability of a successful implementation of functionality in a cost-effective manner, while meeting the necessary design constraints [4]. Design reuse, hardware software partitioning, time-to-market, design space exploration, and management of the inherent extraordinary complexity in the recent modern systems are just several important issues that should be addressed in the ESL design flow.

Apparently, ESL design methodology necessitates the use of new languages and tool-sets for different aspects of chip fabrication process, e.g., modeling, simulation, and synthesis. SystemC [5] is becoming the most common hardware description language for system-level design. Incorporation of hardware aspects, e.g., time and concurrency, into a rich programming language like C++ facilitates designing a digital system hardware using well-known software concepts like inheritance, polymorphism, and templates. Novel approaches, such as transaction-level modeling (TLM), seem to be necessary to exploit these advanced concepts into the hardware design paradigm [6], [7]. The TLM library [8] provides facilities to reduce the required simulation time as much as possible. Omission of RTL implementation details, separation of computation from communication, definition of a transaction as a coherent unit of interaction, and modeling communication among the modules at the functional level are just some examples of these facilities. A design procedure that takes advantage of such utilities enables a higher level of design than RTL and closer to system level. This will be referred to as TLM-based ESL design methodology.

To facilitate the hardware realization of an arbitrary application using TLM-based ESL design methodology, an appropriate procedure to map the application into a given target architecture is required. Thus, the high-level specification might be broken into several smaller tasks to be properly mapped into the processing elements (PEs) of the target architecture. Several techniques like parallelism might be incorporated into the mapping process to achieve better performance measures. Based on the given application, the selected architecture and also the incorporated parallelism, the performance metrics and the hardware usage of the final system could be different. Apparently, providing a SystemC implementation for each different situation to assess the whole design space and to find an appropriate solution is tedious, error-prone, and time consuming [9]. In this paper, a novel modeling approach based on the colored Petri net (CPN) is presented to assist designers in the mapping process. A complete procedure is introduced to demonstrate how to create the CPN model for a given application, how to refine it to a preplacement model, and how the preplacement model is utilized for fast performance estimation and placement automation. The assessment procedure presented here can potentially be automated for high-level synthesis (HLS) of the original application. The modeling technique presented here not only provides the required provisions for exploration in different decision making dimensions but also facilitates rapid architectural realization of mapping on a target architecture.

The rest of this paper is organized as follows. In Section II, a brief overview of the related works is presented. Section III provides the required information and preliminary concepts of Petri net (PN) and TLM. A complete flow for application mapping and the required algorithms are thoroughly demonstrated and elaborated in Section IV. A complete example is presented in Section V to clearly show steps of the proposed method. Section VI provides the experimental results and their interpretation, while Section VII concludes this paper and summarizes the proposed approach.

II. RELATED WORKS

A vast range of research has been carried out for the application mapping flow that is one of the most important aspects in the system-level design. The NP-hardness of the application mapping problem motivates the researchers to propose a different category of solutions to partially solve the imposed issues in this area [10]. The proposed mapping algorithms are classified into two main categories: dynamic (i.e., online) mapping [11], and static mapping. The former uses heuristic techniques to find the network bottlenecks and properly distributes the workload among PEs at runtime to improve the desired performance metrics [11]. In spite of great performance improvements achieved in the dynamic mapping approaches, their usage is limited to some specific applications due to the imposed computational overhead and power consumption of the employed algorithm. In static mapping, resource assignment is performed before execution during the design time and it remains fixed at runtime [10].

The static mapping strategies might be categorized to be either exact mapping or search-based mapping. Mathematical programming-based approaches, such as linear programming (LP) or mixed integer LP, are employed in the exact mapping to find an optimal solution based on a given criteria [12]. The most important issue in this technique is the required runtime. In search-based mapping, deterministic or heuristic algorithms might be used to choose a proper mapping that satisfies our demands. Depending on the search type and the extracted results, numerous works are carried out [10]. Symmetric search methods like branch and bound are vastly used in deterministic search domain [13]. In this technique, all of the mapping solutions constitute a tree, called search space. The goal is to find an appropriate mapping in tree branches and bound unallowable solution to reduce the search space. The branch and bound search-based algorithms use significant amount of memory and require long CPU time.

Other search-based mapping techniques employ heuristic algorithms to find the best achievable solution regarding the application requirements and implementation issues. Evolutionary-based approaches such as genetic algorithm [14] or particle swarm optimization [15] or ant colony optimization [16], binomial IP mapping and optimization [17], reliability-aware application mapping [18], and minimum path routing in mesh architecture [19] are just several examples in this domain. Although most of the proposed heuristic techniques are able to partially solve the mapping problem, they are faced with different challenges, such as the slow rate of convergence, or demanding long CPU time.

In contrast to other works, this paper proposes a novel abstract mapping model that is well defined, and grants a
good interpretation and correspondence to the processing and communication tasks in the target architecture. The proposed method also facilitates modeling due its high abstraction level, supports parallelism and resource sharing, provides fast performance estimation, and makes the exploration possible. To reduce the required time for architectural realization of the final decision of the exploration, an automatic task extraction algorithm is also presented.

The experimental results represent speed and the accuracy of proposed model in performance estimation as well as the capability of proposed method in exploring various mapping alternatives in a given architecture. Efficient exploration in different dimension of decision making and rapid realization of mapping on the target architecture, together with our proposed stepwise mapping procedure provides a complete framework for automatic exploration and realization of the optimum result.

III. PRELIMINARY CONCEPTS

The main goal of application mapping can be interpreted as finding a proper implementation (e.g., SystemC) that satisfies the required performance [10]. We use TLM-based ESL design methodology and PN for the implementation and the required intermediate model, respectively. Thus, in this section, preliminary concepts in both TLM and PN domains are briefly described. Explanation of the exact details is out of the scope of this document. Further information can be found in [8] and [20].

A. TLM Basics

As stated before, embedding hardware concepts into a rich programming language like C++ (i.e., SystemC) and using TLM-based design methodology enable designers to analyze and evaluate their architectures in the early design stages. In a TLM, communications among the modules are separated from the computation components. Communications are modeled by transactions and performed through sockets, while transaction requests take place by calling interface functions. Unnecessary details of communication and computation are hidden in a TLM design and might be added later.

A process is a corner stone of SystemC language to describe a computation. Each interaction or communication between two or more concurrent processes is called a transaction. A SystemC module by which a transaction is being initiated is called an initiator. A target is also a SystemC module that accepts the given transaction and is expected to respond, thus it represents the final destination of the transaction.

In this paper, a configurable parametric platform for system-level design, modeling, and analysis is implemented. The platform provides some facilities to model, simulate, and analyze an arbitrary system. It also provides the required provisions to map an application into a chosen architecture. The platform consists of an arbitrary number of nodes, each of which could be used to produce, consume, process, and transfer data. Each node is required to be statically reconfigured to adapt based on the given application in the chosen architecture and the other criteria.

B. Basics of PN

PNs are graphical and mathematical models that are suitable to represent, simulate, analyze, and validate systems in which concurrency, communication, and synchronization have a major role [20], [21]. A PN is a directed graph with an additional element called token by which the simulation of the system’s concurrent activities becomes possible. A PN has two types of nodes, namely places and transitions represented by circles and boxes (or bars), respectively. Places and transitions are connected by arcs. Each place contains an integer (positive or zero) number of tokens (marks). A transition could just be fired when all input places (preconditions) of the transition contain at least one token. The transition is then said to be fire-able, or enabled. Firing a transition imposes withdrawal of a token from each input place of the transition, and also adds another token to all output places of that transition [20].

1) Different Extensions of PN: Different extensions of PN have been proposed to model various applications in different abstraction levels. CPN [22], transition/place timed PN [23], stochastic PN (SPN) [24], and generalized SPN [25] are just several examples in PN domain that are frequently used. In contrast to ordinary PN, data value assignment to each token, using complex transitions and sophisticated enabling/firing rules make CPN suitable to model complex systems. The capabilities of data and time assignments to the tokens make CPN a suitable candidate to model a complex application that is to be mapped into a given architecture. Here, we present a simplified extension of CPN by which the preplacement model of an application could be illustrated and examined.

2) Mathematical Definition of Proposed CPN: The proposed CPN model is a 10-tuple\( Q = (P, T, Pre, Post, Prd, Cns, D, TType, M_0, FFun)\). \(P\) and \(T\) are a finite, not empty, set of places and transition names, respectively. \(Pre\) (\(Post\)) is a set of lists. Each list corresponds to a transition and contains indices of its precondition (postcondition) places. \(Prd\) (\(Cns\)) is also a set of lists. Each list corresponds to a place and contains indices of its producer (consumer) transitions. \(D\) is an array of integer numbers; each specifies firing time delay of the corresponding transition. \(TType\) is an array of 0 and 1. Zero (0) indicates immediate while one (1) designates timed transitions [21]. \(M_0\) is an initial marking that represents initial state of the modeled system. \(FFun\) is a set of firing functions, each of which corresponds to a transition. Fig. 1 shows the graphical representation and the corresponding mathematical
description of a CPN example with four transitions and five places. This model plays an important role in mapping method presented in the next section.

IV. PROPOSED APPLICATION MAPPING ALGORITHM

A complete flow of the proposed system-level application mapping is shown in Fig. 2. Specification of an application and a given architecture are inputs of the algorithm. The algorithm begins with a CPN model that represents task dependency graph of the application. The CPN model is refined in three consecutive steps that are parallelism, node assignment, and time sharing. After refinement, the new CPN model (i.e., preplacement model) is ready to be placed into the target architecture.

The target architecture has already been configured using the TLM-based configurable platform described in Section III-A. The placement phase includes task extraction from the preplacement model, and task incorporation into the target architecture. The actual run time of the application is obtained by simulating the configured architecture after the placement phase (i.e., postplacement simulation). Before placement takes place, the performance estimation phase is conducted by simulating the preplacement model annotated with communication times. Preplacement simulation, and hence performance estimation is very fast due to the high abstraction level of the proposed CPN model.

To reach a better performance, refinement and performance estimation phases might be repeated several times to perform exploration in both dimensions of the design space, i.e., parallelism and node assignment. Different steps of the proposed algorithm are thoroughly discussed and explained in the rest of this section.

A. Application Dependency Graph Using CPN

A system-level application generally contains several dependent smaller tasks (e.g., the arithmetic functions represented by nodes in a data flow graph [26]) that constitute the overall function of the application. As the first step of the mapping procedure, all simple tasks and the corresponding dependencies are represented using a CPN model. Each simple task is shown using a separate transition connected to others through places. The way in which transitions are connected together is based exactly on the task dependency of the application. The granularity, and hence the number of simple tasks could vary based on the application and the processing capability of the corresponding node in the target architecture.

B. Refinement

The resulting application CPN model is then refined within three consecutive steps including parallelism, node assignment, and time sharing described separately in the following three sections. See refinement box in Fig. 2.

1) Parallelism: Simple tasks of the application have execution times that can be estimated based on the hardware platform of a given architecture. The estimated execution time of a simple task is associated with the corresponding transition in the CPN dependency model as the firing time delay. As described in Section III-B2, the firing time delay of a transition demonstrates how long the transition has to wait to be fired after it is enabled. The resulting CPN that is now a timed PN can be simulated again to preliminarily check the execution time of the application.

To improve the overall performance of the application, two kinds of parallelism can be incorporated into the application. All dependent tasks that are sequentially connected as a pipe and concurrently perform multiple processing functions on a sequence of data tokens are considered as a pipeline. We can also easily increase the order of parallelism by repeating (full parallelism of) each pipe or single task that performs a single function on multiple data tokens. A new transition, namely, distribution transition, is required to be added at the beginning of the repeated parts to truly distribute input data tokens among them. Full parallelism is recommended for tasks and pipes that are more time consuming than the others. The parallelism techniques necessitate sufficient redundancy in the hardware. To see different forms of parallelism in both types, refer to the experimental results in Section VI.

2) Node Assignment: In general, each transition in the CPN model (except distribution transitions added before fully paralleled parts) represents a simple processing task, while a place indicates intertask communication. An intertask communication could be carried out directly through communication links or through a memory module in the target architecture. Therefore, in the first step of node assignment, each place in the CPN model should be exclusively marked as a communication place or a memory place. Then, each memory place is assigned to an existing memory node in the given architecture. The number and size of memory modules, which vary based on the given architecture might be considered as constraints. The following guidelines are suggested for the memory node assignment on CPN model.

1) Each place of CPN model might be arbitrarily marked as memory or communication node. However, a place with large size of data token is mostly recommended to be marked as a memory place.

2) Each memory place should be assigned to just one memory node while a memory node may be assigned to more than one memory place.
3) A memory node is required to have the necessary communication paths (with appropriate direction) to communicate with those nodes that might initiate a read or write transaction.

On the other hand, all transitions in the CPN model (except distribution transitions) should be assigned to the processing node of the given architecture. A distribution transition will be interpreted in communication tasks at the task list extraction step discussed in Section IV-C1. The following are suggested as a set of guidelines to perform processing node assignment using CPN model.

1) Each processing transition of CPN model should be assigned to just one processing node, while it is possible to assign more than one transition to a specific processing node.

2) There must be at least one communication path (with an appropriate direction) between the two separate processing nodes that perform two consecutive processing tasks in the CPN model.

3) To achieve best performance of parallelism, different tasks of a pipe or fully paralleled tasks should be assigned to different processing nodes of the target architecture.

To get a better performance, it is highly recommended that the processing and memory node assignment to be done in such a way that the overall waiting for exclusion [8] that indicates the load of communication links is minimized.

3) Time Sharing: In the case that several processing tasks are assigned to just one processing node, the obtained CPN should be refined to be able to execute the realistic operation with respect to resource (i.e., processor time) sharing and exclusion. For those simple tasks that are serially dependent and, at the same time, are assigned to just one processing node [Fig. 3(a)], transition merging (i.e., aggregation) is suggested as a feasible solution. The aggregated transitions will act as a simple task with the total execution time of all the transitions in the sequence.

As shown in Fig. 3(b), a shared node is represented by a place (marked in yellow in the figure) with one or more black tokens (i.e., without any data). The number of tokens in the yellow place demonstrates the number of processing tasks (transitions) that could be simultaneously executed in the corresponding node. To appropriately model the mutual exclusion, an immediate transition marked in black is inserted before both tasks assigned to the shared node. The first block (data) token reaching the immediate transition will take the black (no data) token (i.e., the resource of the shared processing node) and others have to wait until the former is finished.

C. Placement

The placement phase (see the corresponding box in Fig. 2) includes two steps, namely, task list extraction from preplacement model and task incorporation into the target architecture resulted in postplacement model. Both steps are described separately in the following sections.

1) Task List Extraction: The resulting CPN model obtained after transition merging contains enough information to generate a list of processing and communication tasks. These tasks are supposed to be incorporated into the target architecture’s nodes. The pseudocode of the proposed task extraction algorithm is shown in Fig. 4. Extracted task list, i.e., the output of this algorithm, is fed into both task incorporation and preplacement steps discussed in Sections IV-C2 and IV-D, respectively. Before explaining different steps of the proposed algorithm, the mechanism by which tasks are extracted from CPN model is discussed and explained by giving an example.

Each transition in the CPN model explicitly demonstrates a processing task and is supposed to execute on the corresponding assigned processing node. Each place of the CPN model represents intertask communication that could be carried out directly through communication links or through a memory module in the target architecture. The transfer of a token within a memory place is interpreted as a write, a notify, and a read transaction while the transfer of a token within communication place is interpreted as a SendData in the target architecture.

As an example, suppose that both the producer and the consumer transition of a memory place (which is assigned to the memory node M) are associated with two separate nodes, called P and C, respectively [Fig. 5(a)]. In such a case, the complete communication between nodes P and C includes three consecutive dependent actions: a write action from node P to node M, a notify action from node P to node C, and a read action by node C from the node M. It also might be possible to have a situation that a communication place connects producer and consumer transitions together. These transitions are mapped to node P and node C, respectively [Fig. 5(b)]. In this case, node P communicates with node C through a SendData transaction.

Six different situations could occur while extracting tasks from a given processing transition of the CPN model. In what follows, details of each situation are discussed. Note that the node assigned to a transition is referred to as current node.

1) For each preplace assigned to a memory node, a read transaction is initiated by current node to get the required data from the memory. This situation is managed in lines 10–15 in Fig. 4.

2) If both the preplace of the current transition and the corresponding producer are not assigned to a specific node (i.e., distribution transition and its postplaces), current node will initiate a read transaction to get data from the memory that is assigned to the preplace of the producer transition (if any). Refer to line 17 in Fig. 4.

3) Every transition in the CPN model demonstrates at least one processing task. When the current transition is a merged (i.e., aggregated) transition, a set of consecutive
Fig. 4. Proposed algorithm to extract the task list.

1) TASKLISTEXTRACTION\((P, T, Pre, Post, Prd, Cns, MemNode, ProcNode, ProcFunc)\)
2) begin
3) // each Task is a 5-tuple \{Name, Type, Source, Destination, Dependency\}
4) for each Place \(P_i\) in \(P\) clear GlobalTaskDep\((P_i)\) List;
5) GlobalTaskDep\((TopP)\) = \{\}; \(Idx = 1\);
6) \((Tr, Deadlock) = \text{Find\_ENABLED\_Tr}\((P, T, Pre, Post, Prd, Cns, GlobalTaskDep)\);
7) While (!Deadlock) do
8) if (ProcNode\((Tr)\) != Null) then
9) clear LocalTaskDep List;
10) for each place Pre\(P_i\) in Pre\((Tr)\)\)
11) if (MemNode\((Pre\(P_i)\) != Null) then
12) Task\((Idx) = \{\text{read, communication, ProcNode}\((Tr)\), MemNode\((Pre\(P_i)\), GlobalTaskDep\((Pre\(P_i)\))\};
13) add \(Idx++\) to LocalTaskDep List;
14) else
15) \(PrdTr = Prd\((Pre\(P_i)\));\)
16) if ((ProcNode\((PrdTr) == Null) && (MemNode\((Pre\(P_i\), Prd\((Pre\(P_i)\)) != Null) ) then
17) Task\((Idx) = \{\text{read, communication, ProcNode}\((Tr)\), MemNode\((Pre\(P_i\), Prd\((Pre\(P_i)\))\), GlobalTaskDep\((Pre\(P_i)\))\};
18) add \(Idx++\) to LocalTaskDep List;
19) end
20) else
21) add GlobalTaskDep\((Pre\(P_i)\) to LocalTaskDep List;
22) for each func in ProcFunc\((Tr)\)
23) Task\((Idx) = \{\text{func, processing, ProcNode}\((Tr)\), Null, LocalTaskDep\};
24) LocalTaskDep = \{\(Idx++\}\};
25) for each Place Post\(P_i\) in Post\((Tr)\)
26) if (MemNode\((Post\(P_i)\) != Null) then
27) Task\((Idx) = \{\text{write, communication, ProcNode}\((Tr)\), MemNode\((Post\(P_i)\), LocalTaskDep\);
28) LocalTaskDep = \{\(Idx++\}\};
29) else
30) clear GlobalTaskDep\((Post\(P_i)\) List;
31) if (ProcNode\((Post\(P_i)\) != Null) then
32) Task\((Idx) = \{\text{notify, communication, ProcNode}\((Tr)\), ProcNode\((Cns(\(Post\(P_i)\)), LocalTaskDep\);
33) add \(Idx++\) to GlobalTaskDep\((Post\(P_i)\) List;
34) else
35) for each place Post\(P_i\)\(\_\)lv2 in Post\((Cns(\(Post\(P_i)\))
36) Task\((Idx) = \{\text{notify, communication, ProcNode}\((Tr)\), ProcNode\((Cns(\(Post\(P_i\))), LocalTaskDep\);
37) add \(Idx++\) to GlobalTaskDep\((Post\(P_i)\) List;
38) else if ((ProcNode\((Cns(\(Post\(P_i)\)) == Null) && (ProcNode\((Tr) != ProcNode\((Cns(\(Post\(P_i)\)))\) ) then
39) Task\((Idx) = \{\text{send data, communication}, ProcNode\((Cns(\(Post\(P_i)\)), LocalTaskDep\);
40) add \(Idx++\) to GlobalTaskDep\((Post\(P_i)\) List;
41) else
42) add LocalTaskDep to GlobalTaskDep\((Post\(P_i)\) List;
43) for each place Pre\(P_i\) in Pre\((Tr)\)
44) remove first item from GlobalTaskDep\((Post\(P_i)\) List;
45) \((Tr, Deadlock) = \text{Find\_ENABLED\_Tr}\((P, T, Pre, Post, Prd, Cns, GlobalTaskDep)\);
46) end.

Fig. 5. Interpretation of a place. (a) Memory place. (b) Communication.
place. The proposed algorithm begins with a warmup initialization and repeats until there is no other enabled transition (i.e., line 8). The rest of the algorithm is responsible for extracting proper tasks from the given CPN as stated before.

2) Task Incorporation: The last step of the application mapping process is task incorporation. In this step, the extracted tasks are placed into a given architecture. Each extracted task is assigned to the node reported as the source of the task (refer to Table I in Section V). To precisely guarantee the required dependencies, a checking routine is incorporated in each processing node of the target architecture modeled using SystemC. This routine will initiate the next transaction as soon as the previous one has been completed.

D. Performance Estimation

The extracted preplacement model (i.e., the CPN model after refinement phase) might be simulated instead of postplacement model (SystemC model after placement) to estimate the execution time of mapping instance. To get a more realistic result, communication times should be added to the preplacement model. The first one of the two following sections explains how communication times are added to the model while the second describes the preplacement simulation.

1) Add Communication Time: Although after transition merging phase the extracted CPN model implicitly represents different types of communications (i.e., read, write, notify, and SendData), it does not include their corresponding communication times. The precise communication times will be shown in the postplacement simulation using the corresponding SystemC model (Section IV-E). To get more accurate performance measures in the preplacement simulation, an approximate communication time may be added to the extracted preplacement model. Recall that, the communication times could be easily affected by the traffic pattern (due to the exclusion rules [8]), the length of the path (that is returned by the incorporated routing algorithm), and the latency of the switches and wires (represented by initiator, target, socket, and interconnect delay parameters) are described below.

Each initiator uses two delay parameters by which the required time to prepare BEGIN_REQ (tIBQ), and the required time to prepare END_REQ (tIBP) are shown. Likewise, a target is specified using a pair of delay parameters that are the required time to provide END_REQ (tIEQ), and the required time to prepare BEGIN_RESP (tIBE). Four other delay parameters are considered for the sockets and interconnects. These four parameters determine the required time for each socket to send BEGIN_REQ (tSBQ), END_REQ (tSEQ), BEGIN_RESP (tSBP), and END_RESP (tSEP), respectively. An initiator socket uses the first and the last delay parameters (i.e., tSBQ and tSEP), while a target socket uses the other two (i.e., tSEQ and tSBP) [8]. The required delay parameters to conduct our simulation are set based on the following values: (initiator: (tIBQ, tIBP, tIBE), target: (tIEQ, tSEQ, tEP), sockets: (tSBQ, tSEQ, tSBP, tSEP), interconnects: (tIBQ, tIEQ, tIBP, tIEQ)).

Using the above parameters, the following equation can be used to estimate communication delay (dcom) for read, write, notify, and SendData transactions:

\[d_{com} = d_1 + (l_p - 1) \times d_2\]

where \(l_p\) represents the path length between the source and the destination of the transaction, \(d_1 = tIBQ + tIEQ + tIBP + tSBQ + tSEQ + tSBP + tSEP\) and \(d_2 = tIBQ + tIEQ + tIBP + tIEQ\) when transaction is a read or write, otherwise \(d_1 = tIBQ + tIEQ + tSBQ + tSEQ\) and \(d_2 = tIBQ + tIEQ\) for notify and SendData in which all delays regarding to the response phase are excluded.

When a communication task (i.e., read, write, notify, and SendData) is executed, the corresponding delay is also computed and added to the related consumer or producer transition. The estimated times for write and SendData are added to the existing delay of the consumer while the extracted delay of read and notify are added to the producer. In the case that a communication task (e.g., a read or write transaction) needs an execution time, it will be added to the estimated delay time of communication. The resulting preplacement CPN model is then completed and is ready for preplacement simulation and performance estimation.

2) Preplacement Simulation: To assess the performance of an application, the extracted CPN model before the placement can be simulated. As it can be seen in Section VI, this simulation just provides us a fast and good approximation of the performance metrics that might be used in the first stages of the design process. To obtain an exact and precise timing information, the SystemC model after the placement step (i.e., the postplacement model) should be considered and simulated.
E. Postplacement Simulation

The precise communication times appear in the SystemC model after the placement step for the target architecture. Recall that, the preplacement simulation is conducted on the resulting CPN model after adding communication time, while postplacement simulation is performed on the SystemC model of the target architecture that employs the complete four-phase TLM communication protocol [8]. The difference between preplacement and postplacement simulation results is due to the exclusion/inclusion of the exact communication time.

This approach has the same analogy with the presynthesis and postsynthesis simulation in the RTL design methodology. Although the RTL presynthesis simulation provides a rough estimation of the complete digital circuit and could verify the correctness of the design, but it is not able to specify the exact timing values like pin-to-pin delays. However, in RTL postsynthesis simulation precise timing values are extracted based on the corresponding parameters of the target technology.

V. COMPLETE EXAMPLE

To clearly illustrate each step of the algorithm, a simple JPEG encoder is considered as an example to be mapped on a 4 × 4 mesh-based network architecture. The application specification [Fig. 6(a)] [27] and target architecture [Fig. 6(b)] are the inputs of the proposed algorithm, while the preplacement CPN model and the extracted postmapping tasks are considered as its output. Fig. 6(a) also demonstrates partitioning the JPEG encoder application into 10 simple tasks. These tasks as well as their estimated execution times are listed in a table in Fig. 6(c). Due to the lack of space, details of tasks are not shown here, and could be found in any JPEG document [27].

Fig. 5(a) shows the JPEG encoder dependency graph using a CPN model. Each simple task is shown using a separate transition that is connected to others through places. For all the processing tasks, the required execution time is given below the corresponding transition [Fig. 6(c)]. The token in the top place represents an image that should be encoded. After RGB2YUV (shown as \( f_1 \)) and DownSampling, i.e., \( f_2 \), the image will be split into \(#B\) blocks that appear as \(#B\) tokens with different ID numbers in the place \( P_4 \). Then, each block token will be delivered by the transition (named general) to both dc and ac pipes [i.e., left and right stems in Fig. 7(a)]. The dc pipe consists of DCT, QuantizeDC,
and DPCMCoding, while the ac pipe includes DCT–AC, QuantizeAC, and ZigzagCoding tasks.

Fig. 7(b) shows the CPN dependency graph of the JPEG encoder after time assignment. Both forms of parallelism are incorporated in this example. Tasks in ac (or dc) pipe in which multiple tasks concurrently perform multiple processing functions on a sequence of block tokens are considered as pipeline(s). Furthermore, the fully parallelism technique is applied by repeating the ac pipe. Fig. 7(c) shows this situation. Consider those ac pipes (two right most branches) into which different image blocks are fed, while both perform the same function in parallel for different data blocks. Based on the block ID parity (i.e., even or odd), one of the ac pipes is selected by the transition named Mod 2 distribution. A simple comparison between execution times of the ac and dc stems reveals our reason for choosing the ac stem for parallelism.

The extracted processing and communication tasks for the simple JPEG encoder [Fig. 7(f)] mapped on a $4 \times 4$ mesh architecture [Fig. 6(b)] are listed in Table I. For each task, task name, task type (either communication or processing), source node (i.e., initiator node for the communication task and processing node for the processing task), the destination node (i.e., target node for the communication tasks), and the corresponding dependency [i.e., related predecessor(s)] are shown in columns 1–5, respectively.

In addition to the transition merging, mutual exclusion is required to be done before preplacement simulation to get results that are more realistic. Mutual exclusion is done to properly model the prohibition of simultaneous execution of
tasks $T_4$ and $T_6$ that are assigned to the processing node 6. As shown in Fig. 7(g), a shared node is represented by a place (marked in yellow in the figure) with one or more black tokens (i.e., without any data).

The preplacement simulation is a way for carrying out performance estimation of the current mapping instance before being completely incorporated into the architecture. This advantage makes it possible to explore and examine different mapping instances to obtain more efficient mapping schemes. In the next section, several mapping schemes of the JPEG encoder will be examined. The results of preplacement and postplacement simulation (i.e., simulation speed and actual encoding time of JPEG) are compared with assess the effectiveness of the proposed modeling approach.

VI. EXPERIMENTAL RESULTS

One possible mapping of the JPEG encoder on a $4 \times 4$ mesh architecture is presented in Section V. Several mapping alternatives, shown in Fig. 8, have also been examined to assess the proposed application mapping technique. The detail of each mapping is clearly demonstrated in this figure. As stated before, the postplacement simulation is conducted using the SystemC platform configured in such a way to employ the TLM complete four-phase communication mode. A delay parameter set shown in Table II is chosen to conduct the simulation. The values for the initiator, target, sockets, and interconnects are selected in such a way to proportionately imitate a communication scheme in the JPEG encoder.

Fig. 9(a) shows the extracted result of different mapping schemes. These results reveal the ability of the CPN model to precisely show the system behavior for different forms of parallelism in various mapping schemes. The bar chart clearly shows that the extracted JPEG encoding time obtained using the CPN model is almost similar to the actual encoding time that can be obtained after postplacement SystemC simulation. Hence, it could be used as a good approximation instead of the postplacement results.

Fig. 9(b) shows the required simulation time for the preplacement and postplacement simulations. As expected, the required simulation time for the postplacement model is much higher than the proposed CPN model [consider the logarithmic scale of the vertical axes in Fig. 9(b)]. The large amount of details in addition to the sophisticated programming concepts that are vastly used in both SystemC and TLM libraries reduce the speed of the postplacement SystemC simulation. Precise timing in initiators, targets, sockets, and interconnects add extra burden to the simulation engine, and hence, causes a relatively long simulation time.

Another advantage of the proposed approach is related to the required time to clearly model the whole system. Implementation of a model using even a rich hardware description language like SystemC with a strong library like TLM is tedious, error-prone, and time consuming. Utilization of a more abstract model like what we proposed in this paper not only eases the whole modeling process, but it also reduces the modeling time and effort.

Reducing the required time for modeling of an application, the speed and accuracy of the proposed model in performance estimation, together with the proposed stepwise mapping procedure provide a complete framework to explore the optimum result in mapping a high-level application into a given architecture. The proposed task list extraction algorithm run within a second reduces the timing of placement step (realizing final system) with a very high order of magnitude.

VII. CONCLUSION

A configurable parametric platform for system-level design, modeling, and analysis is implemented to map a high-level application into a given target architecture. An appropriate CPN extension and the required techniques are also presented to properly model and simulate the high-level application. The proposed CPN model not only graphically represents the task sequence, the incorporated parallelism, and the target nodes, but it also provides necessary provisions to manage communication and processing time, dependencies, and mutual exclusion.

A complete stepwise flow is introduced to illustrate how the preliminary CPN model of an application is refined to the corresponding preplacement model. A task extraction algorithm is also presented to automatically extract the required
communication and processing tasks of the given application to be mapped into the target architecture.

To clearly illustrate each step, a simple JPEG encoder is considered as an example to be placed on a 4 × 4 mesh-based network architecture. To examine the proposed modeling approach in various situations, fourteen different mapping schemes are chosen, each of which employs a specific kind of parallelism and uses distinct node assignments. The application mapping, preplacement simulation, and the corresponding postplacement simulation are performed for various mapping alternatives. Like presynthesis and postsynthesis simulation in an RTL design methodology, the major difference between preplacement and postplacement simulation is due to the details of communication in the postplacement models. The results indicate that the proposed modeling approach not only provides the required provisions for exploration in different dimensions through the fast and accurate performance estimation, but it also facilitates rapid architectural realization of the mapping alternatives on a target architecture.

REFERENCES


[15] Zainalabedin Navabi

Hossein Sabaghian-Bidgoli was born in Aran va Bigdol, Iran, in 1971. He received the B.S. degree from the Isfahan University of Technology, Isfahan, Iran, in 1994, and the M.S. degree from the Iran University of Science and Technology, Tehran, Iran, in 1999. He is currently pursuing the Ph.D. degree with the University of Tehran, Tehran, Iran.

He has been a Faculty Member of the Department of Computer Engineering at the University of Kashan, Kashan, Iran, since 2006. His current research interests include modeling for design and test applications in different abstraction levels.

Seyed Ali Shahabi was born in Borujen, Iran, in 1981. He received the B.S. degree from the University of Isfahan, Isfahan, Iran, in 2004, and the M.S. degree from the University of Tehran, Tehran, Iran, in 2007, where he is currently pursuing the Ph.D. degree.

He has been a Faculty Member of the Branch of Borujen, Islamic Azad University, Borujen, since 2010. His current research interests include system level design, reconfigurable architectures, and design space exploration.

Zainalabedin Navabi (S’81–M’81–SM’92) was born in Tehran, Iran, in 1952. He received the B.S. degree from the University of Texas at Austin, Austin, TX, USA, in 1975, and the M.S. and Ph.D. degrees from the University of Arizona, Tucson, AZ, USA, in 1978 and 1981, respectively.

He is a Professor of Electrical and Computer Engineering with the University of Tehran, Tehran, Iran, and an Adjunct Professor with the Worcester Polytechnic Institute, Worcester, MA, USA. He has authored 10 books on various aspects of design of test with hardware description languages (HDL). He has written numerous papers on HDLs, design automation, and digital system test. His current research interests include high-level design and description methodologies, digital system testing, and design and definition of HDLs.