Center for Embedded Computer Systems
Technical Report

Automatic Partitioning and Process Mapping for Model-Based Design

Kyoungwon Kim and Daniel Gajski

CECS Technical Report # 14-09
June 2014

WWW.CECS.UCI.EDU

ABSTRACT

This paper describes partitioning and mapping general MoCs to MPSoC platforms for a model-based design. We optimize execution time while meeting all the other design constraints. The execution time is greatly impacted by process scheduling as well as communication and computation. No previous work is applied to general MoCs such as PSMs and, at the same time, takes into account altogether process scheduling, communication, and computation.

Since mapping alone is not polynomial time problem, we explore several alternatives of heuristics and compare them to an exhaustive search and SPEA. A case study is performed with computation-intensive multimedia applications. The case study shows the proposed approach outperforms the competitions by at least 24.4% in terms of execution time.

University of California, Irvine

Automatic Partitioning and Process Mapping for Model-Based Design

Technical Report CECS-14-09

Kyoungwon Kim and Daniel D. Gajski

Center for Embedded Computer Systems,
University of California, Irvine
Irvine, CA, 92697, USA
{kyoungk1,gajski}@uci.edu

June, 3, 2014

Abstract

In this paper, we propose partitioning and mapping of general MoCs to heterogeneous MPSoC platforms for a model-based design. Our optimization goal is to reduce execution time while meeting all the other design constraints. We claim that the execution time is greatly impacted by process scheduling as well as communication and computation. No previous work is applied to general MoCs such as Program State Machines and, at the same time, takes into consideration altogether process scheduling, communication, and computation.

Since mapping alone is an NP-complete problem, we explore several alternatives of heuristics and compare them to an exhaustive search and SPEA. A case study is performed with computation-intensive multimedia applications such as canny edge detector and an application running MP3 decoder and JPEG encoder in parallel. The results show the proposed approach outperforms the competitions by at least 24.4% in terms of execution time.

Contents

List of Figures

List of Tables

1 Introduction

One of the most serious design challenges in contemporary embedded system designs [2] is the well-known "HW-including-SW design gap" or system design gap. CMOS scaling alone does not ensure simulataneous improvement in performance, power, cost and size, and market dynamics are ever pushing for shorter development times. Hence the design paradigm has shifted to so-called platform-based design [3], where heavy post-fabrication IP reuse is maximized to reduce design cost. Therefore, system design complexity dramatically increases with the intensive use of software (SW) and with the rapid adoption of multi-core or multiprocessor System-on-Chip (SoC) architectures.

Due to design complexity, in any design methodology, one must address productivity. In traditional design methodologies, the design process is not efficient enough, the main reason being the lack of HW/SW co-design. Virtual Platform (VP)-based design methodology allows HW/SW co-design since VP, which is a model of hardware platform, is used for software development before the prototype is ready. However, every change in the platform must be manually implemented and thus VP-based design is not flexible enough for new embedded applications.

One alternative that researchers have proposed is model-based design. The design begins not with platforms but with a Model of Computation capturing the system's functionality. Designers map MoC to Model of Architecture (MoA), which is an abstract model of the selected platform. Transaction Level Model (TLM) is automatically generated for cycle-approximate evaluation so that the design quality can be evaluated without the prototype board [4].

In model-based design, mapping is still intuitively done by experienced designers. Manual mapping is becoming infeasible regarding that the realistic systems are already too complex. A solution to tackle this problem is to automate mapping.

We present mapping techniques from general MoCs to hetoerougeneous MPSoC platforms. A general MoC should have the concept of states, dynamic data-oriented behavior limited by the flow of data and data dependencies across computations, hierarchy in both behavior and structure, concurrency, imperative programming languages, etc.

Our optimization goal is to minimize execution time. Execution time is defined as the time between when the system starts execution and when it ends. Due to tradeoffs between different metrics, no single algorithm can optimize all of metrics at the same time. Designers may choose among different algorithms, each optimizing its own set of the metrics. Our mapping technique is one of them.

Our contribution is that we do all the followings at the same time.

- We present automatic mapping of general MoCs to heterogeneous platforms.
- To reduce the execution time of the system as much as possible, we take process scheduling as well as both communication and computation into consideration
- To estimate the impact of process scheduling ahead of mapping general MoCs, we developed approximations for process scheduling.

Mapping alone is an NP-complete problem [5]. We decompose mapping into smaller subproblems: partitioning the given MoC, process mapping, channel mapping and scheduling. Note that we focus on partitioning and process mapping for a given platform to manage the problem size.

We present N-Way Clustering and Mapping (NWCM) in this paper. NWCM takes as an input a general MoC. Exhaustive one-to-one mapping the clusters to PEs is followed by N-Way clustering based on our new closeness function. In the closeness function, both communication overhead and computation are included. In addition, we develop a simple approximation for process scheduling and include it in the closeness function. The reason why approximation is required is this: process scheduling in a general MoC may heavily depend on runtime behavior and thus is far less analyzable than it is in MoCs that are not general enough. Note that the partitioning procedure in NWCM has polynomial time complexity in the worst case. The time complexity of the exhaustive search following partitioning is O($N!$). However, N is the number of PEs in the selected platform, which is usually small.

The case study performed with Canny Edge Detector and an application running MP3 decoder and JPEG encoder in parallel shows that NWCM outperforms any competitive algorithm by at least 24.4%.

The rest of this paper is organized as follows. In Section 2, earlier studies are reviewed and compared to the proposed work. Section 3 clearly defines the problem. In Section 4, the idea of NWCM is explained. NWCM is basically clustering based on closeness function and, in Section 5, the closeness function is explained and justified. Section 6 explains the algorithm. Section 8 presents a case study: canny edge detector and a streaming application running an MP3 decoder and JPEG encoder in parallel. Since those applications are not large enough for modern embedded platforms, we use old, low performance platforms instead. Section 9 serves as the conclusion.

2 Previous Work

There have, in the past few years, been a large amount of library mapping MoCs to MPSoC platforms. There have been several early studies focusing on Synchronous Data Flow models (SDF) [6]. [7] used graph-based solutions to optimize throughput in mapping SDF to MPSoC platforms. [8] presented mapping SDF to MPSoC platforms to optimzes cost with real-time constraints. [9] showed mapping SDF to heterogeneous MPSoC. Multi-objective optimization based on Strength Pareto Evolutionary Algorithm (SPEA) II [10] and Integer Linear Programming (ILP) is performed to achieve Pareto front in terms of latency, throughput, and cost. They take process scheduling into consideration as well as computation and/or communication. However, SDF does not cover the entire embedded application domains.

[11], [12], and [13] performed multiobjective optimization to find a Pareto optimal set. Multiple criteria such as maximum processing time [12], power consumption, cost, etc, are taken into account. The techniques proposed in these works can be applied to general MoCs as well. Nonetheless, since the input MoCs are either Process Networks or dynamic data flow models in which the concept of states is missing, process scheduling is not taken (or only limitedly so) into consideration. If the given input MoC does have the concept of states, these techniques cannot be facilitated by the information. Process scheduling can greatly impact the system's latency so that these techniques may not minimize latency depending on the type of the given MoC.

[14] took as an input Hierarchical Task Graph (HTG) [15] and mapped the input to a heterogeneous MPSoC platform. The optimization goal is to minimize overall execution time [14], which seems to be the time difference between the start time of the start task and the end time of the end node. HTG is an intermediate form. Therefore, designers may not specify the system's functionality in HTG. Most MoCs can be re-

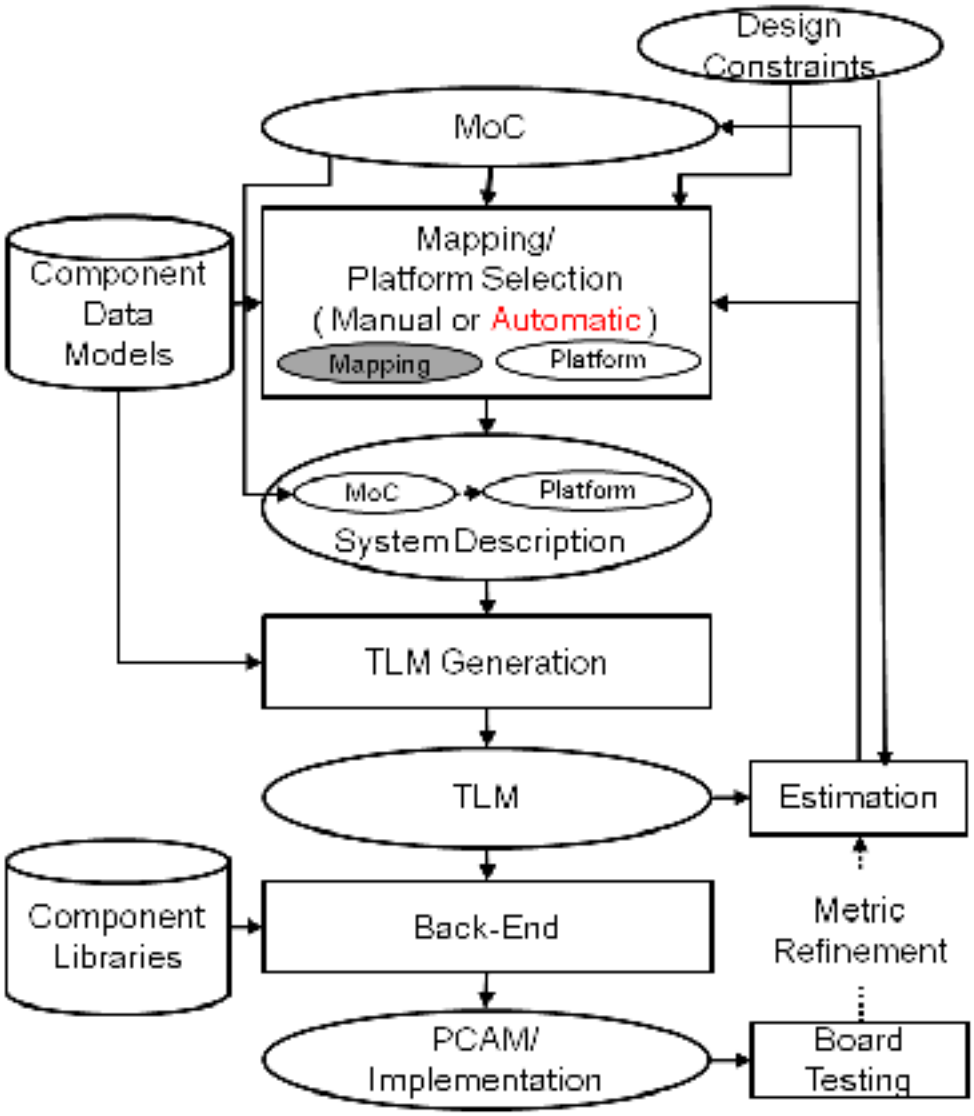


Figure 1: Design Flow in Model-Based Design

fined to this HTG and thus the tecniques are generally applicable. Nevertheless, HTG in its nature is not easy for communication overhead. In addition, impact on process scheduling caused by data dependency between parallel processes are not addressed.

We propose heuristics that take into account process scheduling as well as both computation and communication. The proposed heuristics can take as an input a general MoC to map it to heterogeneous MPSoC while latency is minimized as much as possible and all the other constraints are met. To the best of our knowledge, there is no previous work that maps general MoCs, while communication, computation and process scheduling are involved in the optimization processes.

3 Problem Definition

We add automatic partitioning and mapping to a model-based design as depicted in Figure 1. In model-based design, the design process begins with capturing the system's functionality in an MoC. Modeling the system's functionality and specifica-

tion of the design constraints are followed by platform selection and mapping. We call the result system description. From the system description, a Transaction Level Model (TLM) is automatically generated so that fast cycle-approximate estimation can be conducted by simulating the TLM. If the design constraints are not met, mapping and/or platform selection are performed to improve the design. Or, MoC itself can be modified. Once the design constraints seem to be met according to the TLM-based cycle-approximate estimation, the back-end design procedure will refine the TLM down to the implementation.

First of all, there are trade-offs in selecting an MoC for a given application. For example, there is a trade-off between expressive power and analyzability.

In this paper, we focus on general MoCs and use Program State Machine (PSM) [4].

"The Program-State Machine (PSM) unifies the concepts of hierarchical concurrent finite-state machines, dataflow graphs and imperative programming languages in a single model of computation." (Grütter & Nebel, 2008 [16]).

A program-state can be either of the following. First, it can be further decomposed into concurrent program-states. Second, it can be decomposed into sequential states. Third, it can be a leaf program-state, which we call *a process*. Fourth, it can be a pipeline: several sub program-states are executed in a pipelined manner. We do not cover the last case: the functionality of the entire system is captured in a single pipeline. Nonetheless, in reality, many computation-intensive applications modeled in such a way that we cover in this paper (e.g. [17] [18]).

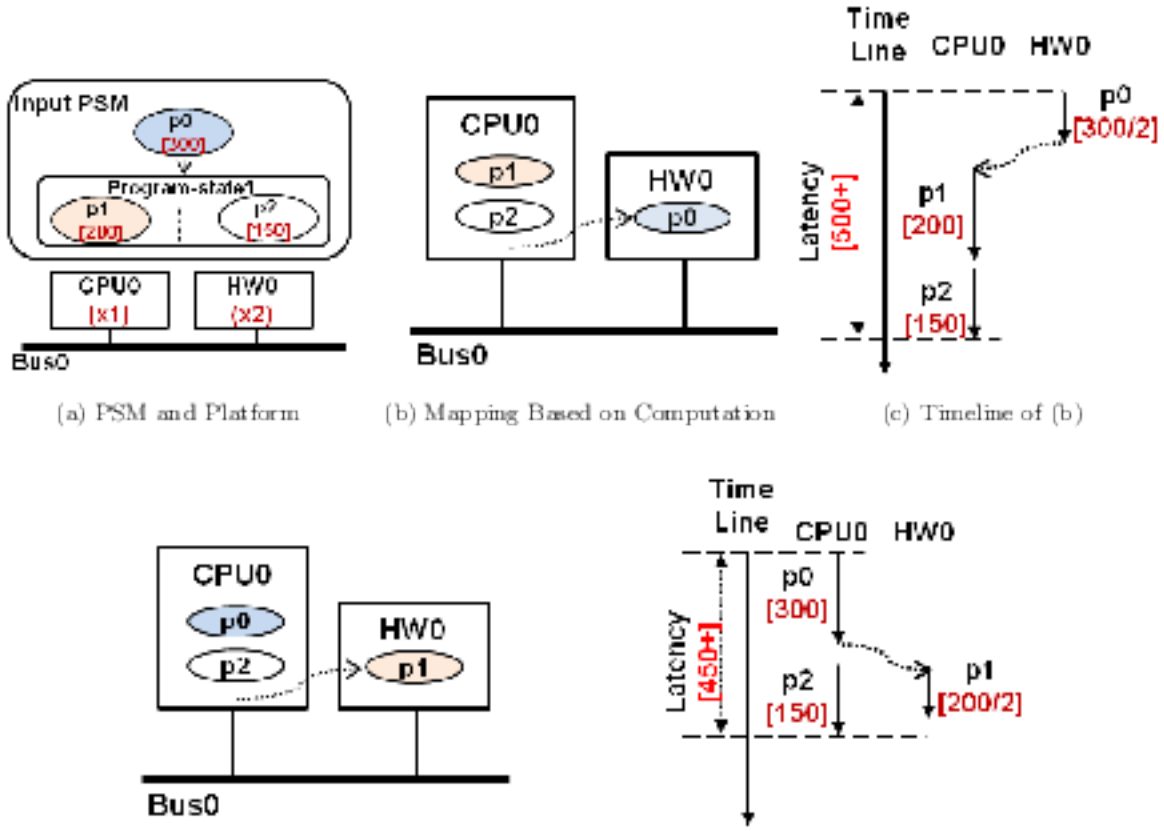
Our target platforms are a combination of processing elements (PEs) such as general processors, DSP, custom hardwares, hardware IPs, etc, as they are in [9]. We assume that a MPSoC platform is provided. In addition, the execution profile, which is the power consumption, cost and execution time of each process on each PE, is given.

Optimal partitioning and process mapping are to be constructed. Our optimization goal is low latency while meeting all the other constraints such as power consumption and cost. We use latency as defined in [19]: latency in transformative systems indicates the average time that a system takes to transform an input from the input stream to the output stream.

The assumptions made for the rest of this paper are as follows: the size of the local memory of each PE is large enough. A single type of RTOS is used for every processor and its scheduling policy is priority-based scheduling. All tasks mapped to a processor have the same priority. Channel mapping is given, once process mapping has been completed. Each process is statically mapped to a single PE. Two or more parallel processes can be mapped to a PE without any RTOS only when static scheduling of the processes is possible.

4 Motivation of N-Way Clustering and Mapping

Mapping processes in model-based designs has been conducted manually. Designers may initially map the entire MoC to a single host processor. Then, the designers may find task-level parallelism to move some processes from the host processor to other PEs. As we see in Figure 2, process scheduling may greatly impact the latency of the system. In the example, p0 and program-state1 run sequentially, while p1 and p2 run in parallel to each other. HW0 is assumed to execute only a single process in this example and to complete every operation twice as fast as CPU0 does. Without considering scheduling, Figure 2b is the optimal. However, in Figure 2b, there is no parallelism. The latency is even reduced in Figure 2d.



(a) PSM and Platform (b) Mapping Based on Computation (c) Timeline of (b)

(d) Mapping Based on Computation and Scheduling (e) Timeline of (d)

Figure 2: Simple Mapping Example with and without Considering Process Scheduling

In addition to finding task-level parallelism, designers often consider the processing time of each PE defined by C. Erbas et al [12] and communication overhead. If the communication overhead between two parallel or sequential tasks is very large, designers may map the two processes to the same PE. Besides, designers are always aware of various design constraints. Especially for real-time constraints, the processing time of each PE must be smaller than the real-time constraints. Finally, when the mapping seems to be infeasible, the designers may change some PEs to better ones and/or use additional PEs under cost constraints.

Based on these observations, we propose NWCM. NWCM performs partitioning by N-Way clustering without violating design constraints, where N is the number of PEs in the platform. For this purpose, we develop the closeness function in which communication overhead, computation, and approximatly estimated impact of process scheduling are taken into account. In addition, we formulate the constraints.

5 Closeness Function of NWCM

The problem is that process scheduling in PSM mostly depends on dynamic behavior. There is only limited knowledge on process scheduling available in PSM at static time; any pair of processes are either sequential or parallel to each other. However,

two parallel processes may run sequentially due to complicated data/control dependency. State transitions can entirely depend on runtime behavior so that the order of execution of two sequential processes is often limitedly predictable.

Nonetheless, there are still multiple ways to take process scheduling into consideration. One is N-Way clustering based on our new closeness function, to which two boolean variables, b_p and b_s, are added. The boolean variables show, respectively, whether the pair of sets of processes are running in parallel and whether the pair can run sequentially.

In general, to exploit parallelism and to reduce the system's latency, a mapping algorithm may as well map two processes to the same PE if there is a intensive/large data transfer between them and separate the processes if the sum of the estimated execution delays is large. In addition, there are two more basic observations. A mapping algorithm may as well:

- separate two parallel processes especially if the overall execution delay is large,
- map two sequential processes to the same PE especially if the processes run back-to-back.

In addition, multiple sequential processes can be mapped to a single HW but not multiple parallel processes. Therefore, by considering the basic observations, we can improve HW utilization and as a result exploit parallelism even further.

Since our work is targeting computation-intensive applications, mapping is finalized as follows: the N clusters are sorted by the order of execution delay and the N PEs are sorted by speed. One-to-one mapping between PEs and clusters is performed in order.

6 Algorithm

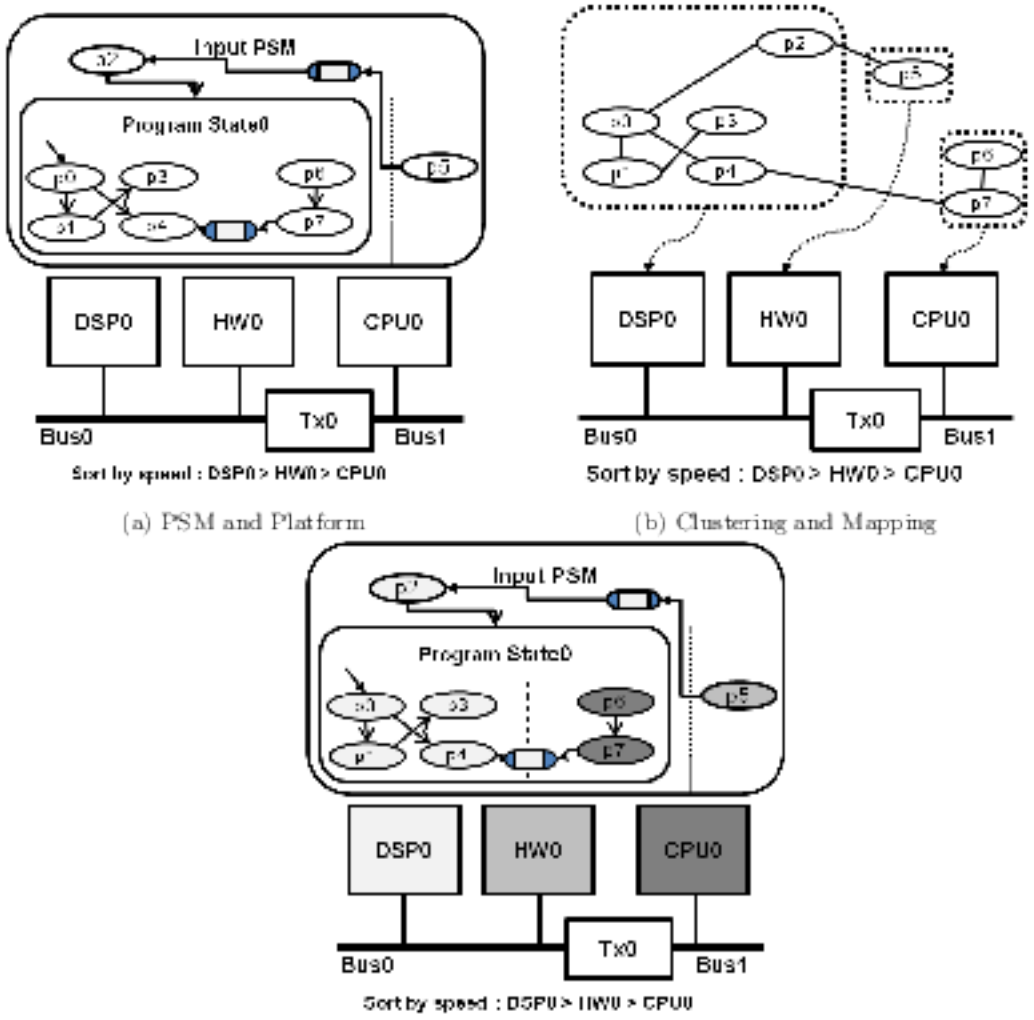
We propose N-Way clustering based on our new closeness function C of process $p0$ and $p1$ in Equation 1. In the Equation, b_s and b_p show, respectively, whether $p0$ and $p1$ may run back-to-back and whether $p0$ and $p1$ run in parallel. D_e and D_t present, respectively, estimated execution delay of the two processes and estimated communication overhead.

$$C(p0, p1) = (c_0 + c_1 * b_s) * D_t + (c_2 - c_3 * b_p) * D_e \quad (1)$$

We can extends the closeness function to take two clusters, each of which is a mutually exclusive set of processes. For two clusters, $Cl0$ and $Cl1$, b_s indicates whether all processes in the two clusters can run sequentially or not and b_p shows whether all processes in the two can run in parallel. D_t and D_e are, respectively, the sum of estimated execution delays of the two clusters and overall communication overheads between the two. Equation 2 shows the extended closeness function of the two clusters.

$$C(Cl0, Cl1) = (c_0 + c_1 * b_s) * D_t + (c_2 - c_3 * b_p) * D_e \quad (2)$$

After N-Way clustering completed, one-to-one mapping between clusters and PEs are performed. Since our approach is mainly targeting computation-intensive applications, clusters are selected one by one in order of the total estimated execution delay of the cluster and PEs are selected in order of speed.



(a) PSM and Platform

(b) Clustering and Mapping

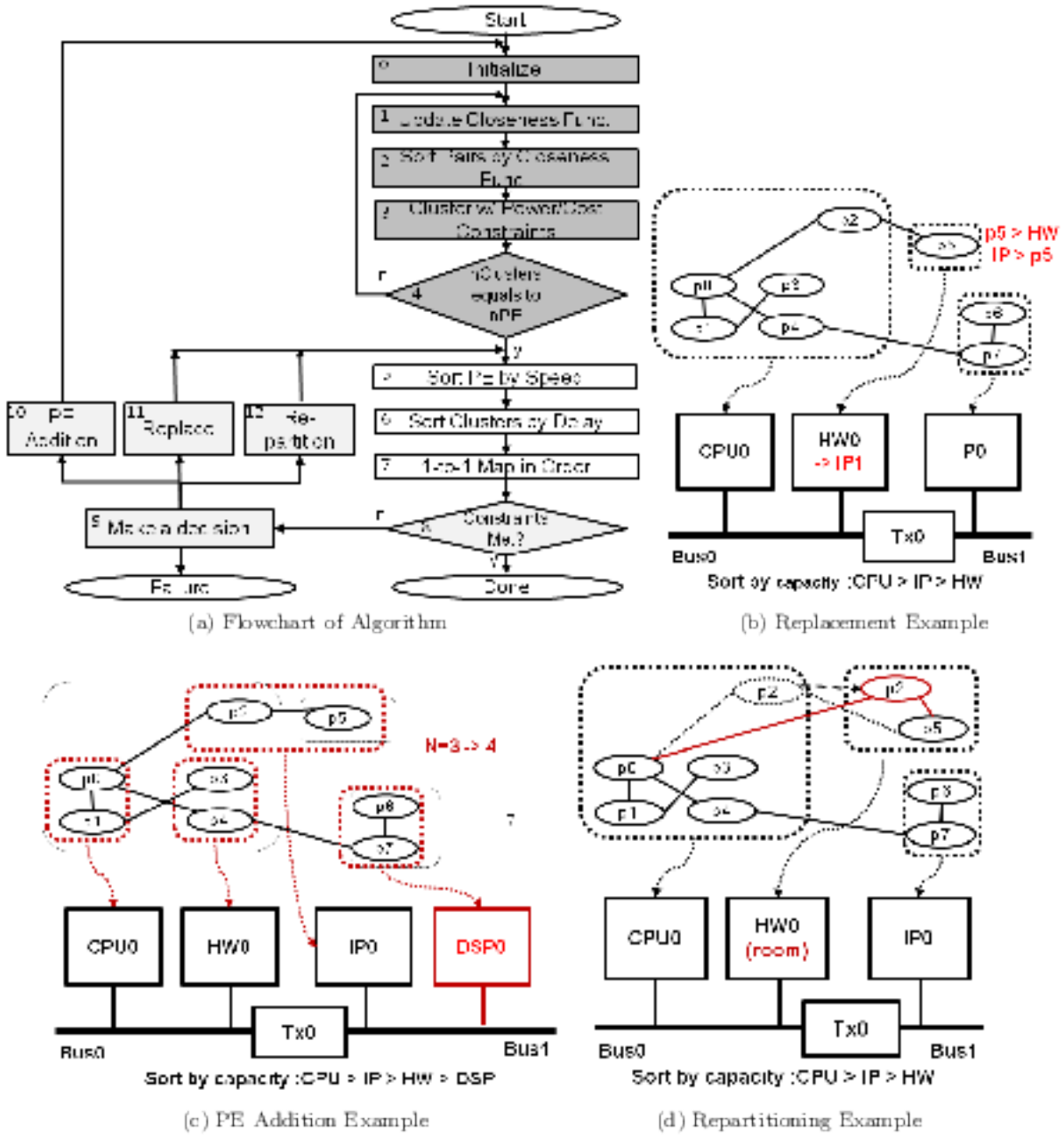
(c) Process Mapping Done

Figure 3: N-Way Clustering and Mapping Example

Mapping is an NP-hard problem, resulting in several approaches based on many different algorithms. Taking into consideration process scheduling, we can apply ILP or meta-heuristic-based approaches to our problem as well. However, ILP is not scaling well to large problem sizes and meta-heuristic based approaches offer no assurance that the required design quality is reached in a finite time. Putting this aside, we can start with a simple algorithm that works.

Figure 3 shows how the algorithm works. The input PSM and the initial target platform are given. From the input PSM, all leaf processes are enumerated. In this example, program state0 is not a leaf. Instead, from process p0 to process p7 are leaf processes. Between any of two processes, an edge is added and the closeness function is used as the weight of the edge. Note that edges with a zero or negative weight are omitted for simplicity. N-Way clustering is performed and what is following is one-to-one mapping depending on execution delays of the clusters and speeds of the PEs. The result is shown in Figure 3c.

Figure 4 shows the flow chart of the N-Way Clustering and Mapping Algorithm. The algorithm is decomposed into three phases, the N-Way clustering, mapping, and fixing



(a) Flowchart of Algorithm

(b) Replacement Example

(c) PE Addition Example

(d) Repartitioning Example

Figure 4: N-Way Clustering and Mapping (NWCM) Algorithm

phase, each of which is indicated by a different color.

Initially, each process constructs its own cluster. The N-Way clustering phase is a loop from box 1 to box 4 and colored dark gray. In the N-Way clustering phase, the following process is repeated;

- Update closeness function values for all pairs of clusters.
- Sort the pairs by closeness function.
- Select a cluster with the closeness function value as large as possible while keeping power/cost constraints met.

The one-to-one mapping process from box 5 to box 7 follows N-Way clustering. The

PEs are sorted by speed. The N clusters are sorted by estimated delays. One-to-one mapping is performed in decreasing order.

The last phase is to fix mapping if any of the design constraints are not met. Note that any heuristic algorithm may fail for many reasons. One reason is that the given constraints are too tight for any heuristic or are even impossible to meet. In the third phase of the proposed algorithm, the given mapping is fixed or the algorithm returns failed with the best design explored. The three ways to fix the mapping are depicted in Figure 4. Three options are selected by decision making schemes. In Figures 4b, 4c, and 4d, the real-time constraint is violated as an example. Other design constraints may be also violated. The first option is to replace PE. For example, in Figure 4b, HW0 is overloaded but replacing HW0 with IP1 solves the problem while meeting cost and power constraints. The algorithm replaces HW0 with IP1. As a result, the ordered list of PEs may change so that the one-to-one mapping process may need to be applied again. The second option is to add a new PE. For example, in Figure 4c, the real-time constraint is violated. Therefore, a new PE, DSP0 is added. In this case, 4-Way clustering is needed instead of the 3-Way clustering already completed. In general, after PE addition, with the updated N, N-Way clustering is performed again. The third option is to select a victim process and move the process to another cluster.

There are various ways to decide which option is selected to fix mapping. Also, there are multiple ways to select PE to replace or add. In addition, a number of ways exist to select both a victim process and the new cluster to locate the process. In this work, we try to replace PE first. If that fails, PE addition follows. The option selected last is repartitioning. The victim process is selected based on the closeness function at the first iteration of N-Way clustering. The cluster to which the process is moved is selected based on the constraint violated.

7 Tools : Embedded System Environment (ESE)

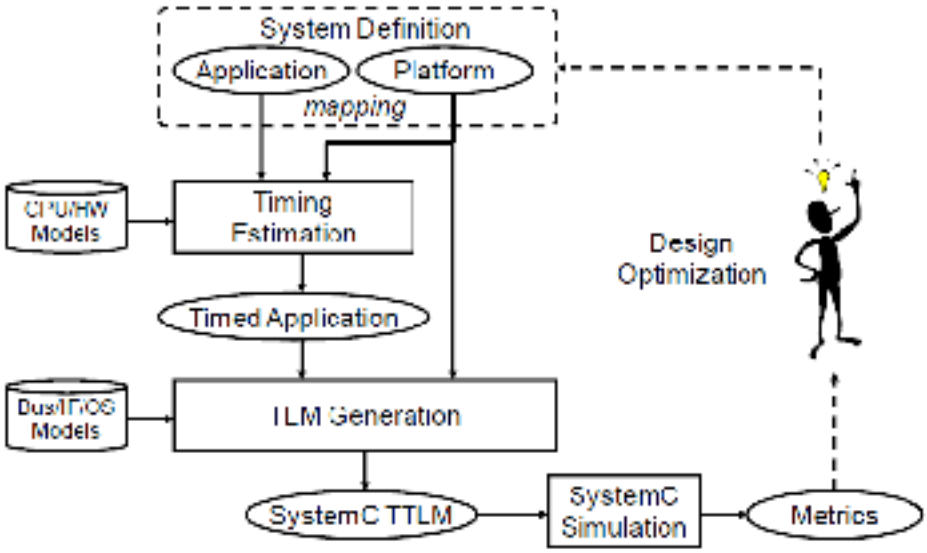


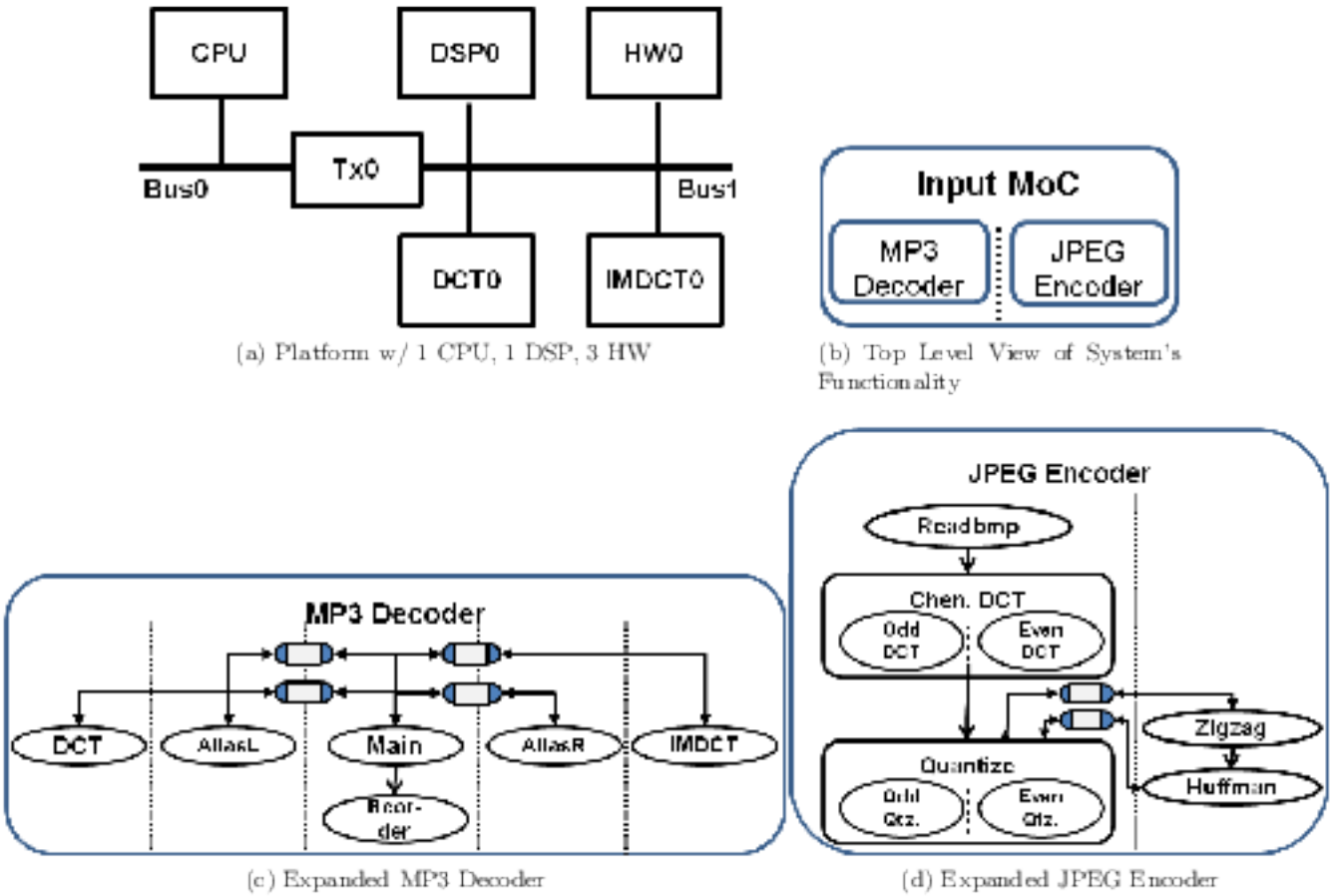
Figure 5: ESE Front-End Tools : source [1]

Our work uses Embedded System Environment (ESE) Front-End tools [1] described

in Figure 5. To give an overview, with the given platform and mapping, timed-TLM is generated and simulated to evaluate the design. In ESE Front-End, Timing Estimation annotates processes with timing based on the mapping and platform. TLM Generation generates TLM by taking as the inputs the time-annotated processes and platform model. The design quality is evaluated by simulating and profiling the generated TLM. The accuracy is accepted as cycle-approximate.

The N-Way clustering algorithm requires an estimated delay of each process. In our work, it is the average execution delay of the process on each PE in the ESE CPU/HW model library. Also, communication overhead is required between any pair of processes. They are measured by dividing the total number of bytes transferred through the channels between the pair by the speed of the communication route between them. Once the proposed mapping algorithm is applied to the given MoC and platform, the resultant system definition is fed to the TLM estimation tool and the TLM generation tool to generate the TLM. We evaluate the design quality by simulating and profiling this TLM.

8 Case Study



(a) Platform w/ 1 CPU, 1 DSP, 3 HW

(b) Top Level View of System's Functionality

(c) Expanded MP3 Decoder

(d) Expanded JPEG Encoder

Figure 6: Application and Platform for Experiment

8.1 MP3 Decoder And JPEG Encoder

We have chosen a computation intensive multimedia application running, in parallel, an MP3 decoder and JPEG encoder. Figure 6a shows the given platform including 1 general purpose processor (GPP), 1 digital signal processor and 3 different pieces of custom hardware. The GPP is connected to bus0 and the others to bus1. A transducer bridges two buses. Figure 6b describes the top level view of the system's functionality. Two program states, MP3 decoder and JPEG encoder, are running in parallel. Figure 6c and Figure 6d expand each program state, respectively. Shared variables accessed by sequential processes are omitted for simplicity. The frequency of each component is between 200 MHz and 400 MHz.

To prove our contribution, it is necessary to compare it to different approaches with similar algorithms and different closeness functions. For this purpose, we use CBB, LB and LPT as competitive algorithms. CBB is a modification of N-Way clustering [20]. CBB is similar to NWCM but uses a different closeness function, which includes only the number of bytes transferred between the two processes. We may compare NWCM to N-Way clustering and mapping algorithms with the closeness function taking only computation and/or communication. However, LB and LPT assure shorter latency than do the algorithms since LB and LPT considers PEs and the capacity of communication routes. Therefore, we compare NWCM to LB and LPT.

These comparisons are not sufficient to assert the quality of NWCM is acceptable. Therefore, NWCM also needs to be compared to previous, realistic approaches as well. We choose a modified version of SPEA for this kind of comparison. SPEA finds pareto optimal solutions in terms of power consumption, cost and maximum computation. For a fair comparison, we apply SPEA with a single objective function, maximum computation. The modified SPEA optimizes latency based on evolutionary algorithms. For SPEA, the number of generations, population size, crossover probability and mutation rate are kept the same as [12]. The input PSM is manually translated to a Khan Process Network and the Tx is used as a single FIFO memory. In the platform, nothing but Tx0 can be used as a shared memory.

Any of the five algorithms—NWCM, LB, LPT, CBB and modified SPEA—produces a single mapping between the given platform and the given MoC. The result of the mapping process is a system model, which is refined to a TLM. The TLM is simulated to evaluate the design quality.

In TABLE 1, the simulation results are enumerated. Each row shows the simulation result of the TLM obtained by each algorithm, respectively. Note that communication overhead is far smaller than latency so that it is omitted. Each row shows the distribution of the total execution delay in addition to the latency. The total execution delay is the sum of the execution times from each process, during which the process is actively running. Latency is the overall response time to process a given set of inputs.

First of all, communication overhead in this application was very small. Therefore, NWCM outperforms CBB in terms of both total execution delay and latency since CBB mainly optimizes communication.

Compared to LB, which optimizes computation only, we can point out the following:

- Execution delay is more evenly distributed in NWCM than it is in LB.
- The total execution delay of LB is, nonetheless, smaller than that of NWCM.
- Overall latency of NWCM is smaller than that of LB.

Table 1: Execution Delays on Each PE and Overall Latency

Algorithms	CPU	HW	DCT	IMDCT	DSP	Total Execution Delay	Latency
NWCM	8.5 ms	2.8 ms	2.3 ms	3.2 ms	5.9 ms	22.7 ms	16.8ms
Modified SPEA	9.3 ms	1.3 ms	1.4 ms	UNUSED	8.9 ms	20.9 ms	24.2ms
LPT	8.5 ms	1.2 ms	2.0 ms	1.7 ms	14.0 ms	27.4 ms	20.9ms
LB	0.9 ms	1.2 ms	2.0 ms	1.5 ms	12.7 ms	18.3 ms	22.6ms
CBB	21.6 ms	0.1 ms	0.1 ms	0.1 ms	19.8 ms	41.7 ms	24.2ms

Multiple sequential processes can be mapped to an HW but multiple parallel processes cannot in general. Therefore, NWCM can map more processes to HW instead of SW and, as a result, improve HW utilization. In spite of this fact, LB usually finds better PEs to the given processes so that the total execution delay of LB is smaller than that of NWCM. However, the latency of LB is larger than that of NWCM, which implies latency does in fact depend on process scheduling. Interestingly, between LB and CBB, the difference in latency is much smaller than the difference in total computation delay. In this given application, sequential processes usually perform a large amount of communication to each other via shared variables. Therefore, CBB often binds sequential processes together. It is also proof of impact of process scheduling.

Since communication overhead is miniscule, LPT produces similar mapping to that of LB. However, LPT exploits parallelism more than LB does due to process scheduling. Thus, the latency of LPT is shorter than that of LB. In the same sense, NWCM is better than LPT.

SPEA was originally applied to Process Network Models. Therefore, it does not consider process scheduling. Therefore, as shown in TABLE 1, even though SPEA shows lower total execution delay than NWCM does, the overall latency of SPEA is worse than that of NWCM.

In summary, the proposed algorithm, NWCM, is better than any other by at least 24.4% in terms of latency although its total execution delay is not the shortest. The reason is process scheduling and HW utilization.

9 Conclusion

This work is intended to automate the mapping process in Model-Based Design. However, mapping is a complex problem and all of its subproblems cannot be solved at once. The problem can be decomposed into smaller problems such as partitioning, process mapping, channel mapping, and scheduling. This work is focused on partitioning and process mapping while the rest is assumed to be given.

Among multiple metrics, the proposed approach reduces the system's latency as much as possible while all the other design constraints are met. Designers may have multiple algorithms with different design goals since any single algorithm cannot optimize all design metrics at the same time. A set of algorithms is chosen depending on the given applications. The proposed approach is one such algorithm especially designed to meet increasing user demands.

The basic idea is to exploit parallelism by considering process scheduling. Previous work that consider process scheduling cannot be applied to several MoCs with limited available knowledge on process scheduling. We propose an approximation so that process scheduling is taken into consideration. We develop a greedy approach

called NWCM that clusters processes based on our new closeness function taking as parameters process scheduling as well as communication and computation and perform one-to-one mapping between N clusters and N PEs based on execution delays and speeds of PEs.

To prove our contribution, the proposed approach is compared to four competitive algorithms; CBB, LB, LPT, and SPEA. While these algorithms may consider communication overhead and execution delays of processes, they all ignore process scheduling. A case study is performed with a computation-intensive multimedia application running an MP3 decoder and JPEG encoder at the same time. The given platform has 5 PEs connected to two buses, which a Tx is bridging. The result shows that the proposed algorithm perfoms the best by at least 24.4%.

References

[1] S. Abdi, Yonghyun Hwang, Lochi Yu, Hansu Cho, I. Viskic, and D.D. Gajski. Embedded system environment: A framework for tlm-based design and prototyping. In *Rapid System Prototyping (RSP), 2010 21st IEEE International Symposium on*, pages 1 –7, june 2010.

[2] International technology roadmap for semiconductor, 2011.

[3] K. Keutzer, S. Malik, A. R. Newton, J. M. Rabaey, and A. Sangiovanni-Vincetelli. System-level design : Orthogonalization of concerns and platform-based design. *Computer Aided Design of Integrated Circuits, IEEE Transactions on*, 2000.

[4] Daniel D. Gajski, Samar Abdi, Andreas Gerstlauer, and Gunar Schirner. *Embedded System Design: Modeling, Synthesis and Verification*. Springer Publishing Company, Incorporated, 1st edition, 2009.

[5] Hoeseok Yang and Soonhoi Ha. Pipelined data parallel task mapping/scheduling technique for MPSoC. In *Design, Automation Test in Europe Conference Exhibition, 2009. DATE '09.*, pages 69 –74, april 2009.

[6] S.S. Bhattacharyya, P.K. Murthy, and E.A. Lee. *Software Synthesis from Dataflow Graphs*. Springer, 1996.

[7] A. Bonfietti, L. Benini, M. Lombardi, and M. Milano. An efficient and complete approach for throughput-maximal sdf allocation and scheduling on multi-core platforms. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2010*, pages 897–902, 2010.

[8] Hyunok Oh and Soonhoi Ha. A hardware-software cosynthesis technique based on heterogeneous multiprocessor scheduling, 1999.

[9] Jing Lin, A. Srivatsa, A. Gerstlauer, and B.L. Evans. Heterogeneous multiprocessor mapping for real-time streaming systems. In *Acoustics, Speech and Signal Processing (ICASSP), 2011 IEEE International Conference on*, pages 1605–1608, 2011.

[10] Eckart Zitzler, Marco Laumanns, and Lothar Thiele. Spea2: Improving the strength pareto evolutionary algorithm. Technical report, 2001.

[11] Mark Thompson, Hristo Nikolov, Todor Stefanov, Andy D. Pimentel, Cagkan Erbas, Simon Polstra, and Ed F. Deprettere. A framework for rapid system-level exploration, synthesis, and programming of multimedia MP-SoCs. In *Proceedings of the 5th IEEE/ACM international conference on Hardware/software codesign and system synthesis*, CODES+ISSS '07, pages 9–14, New York, NY, USA, 2007. ACM.

[12] Cagkan Erbas, Selin C. Erbas, and Andy D. Pimentel. A multiobjective optimization model for exploring multiprocessor mappings of process networks. pages 182–187. ACM, 2003.

[13] Christian Haubelt, Thomas Schlichter, Joachim Keinert, and Mike Meredith. Systemcodesigner: automatic design space exploration and rapid prototyping from behavioral models. In *Proceedings of the 45th annual Design Automation Conference*, DAC '08, pages 580–585, New York, NY, USA, 2008. ACM.

[14] F. Ferrandi, C. Pilato, D. Sciuto, and A. Tumeo. Mapping and scheduling of parallel c applications with ant colony optimization onto heterogeneous reconfigurable mpsocs. In *Design Automation Conference (ASP-DAC), 2010 15th Asia and South Pacific*, pages 799–804, Jan.

[15] M. Girkar and C.D. Polychronopoulos. Automatic extraction of functional parallelism from ordinary programs. *Parallel and Distributed Systems, IEEE Transactions on*, 3(2):166–178, 1992.

[16] K. Gruttner and W. Nebel. Modeling program-state machines in SystemC^{TM}. In *Specification, Verification and Design Languages, 2008. FDL 2008. Forum on*, pages 7–12, 2008.

[17] Xu Han and Gunar Schirner. Real-time mp3 decoding on fpga: a case study of system model features. Technical report, Center for Embedded Computer Systems, July 2009.

[18] Samar Abdi Yongjin Ahn. Process network modeling and tlm generation for h.264 codec design. Technical report, Center for Embedded Computer Systems, October 2008.

[19] Zhengting He and A. Mok. Fast cosimulation of transformative systems with os support on smp computer. In *Hardware/Software Codesign and System Synthesis, 2004. CODES + ISSS 2004. International Conference on*, pages 164–169, 2004.

[20] Frank Vahid and Daniel D. Gajski. Clustering for improved system-level functional partitioning. In *in International Symposium on System Synthesis*, pages 28–33, 1995.