MULTI-PROCESSOR SYSTEM-LEVEL SYNTHESIS FOR MULTIPLE APPLICATIONS ON PLATFORM FPGA

Akash Kumar[1,2],* *Shakith Fernando*[2], *Yajun Ha*[2], *Bart Mesman*[1] *and Henk Corporaal*[1]
[1]Eindhoven University of Technology, Eindhoven, The Netherlands
[2]ECE Department, National University of Singapore, Singapore
Email: a.kumar@tue.nl

ABSTRACT

Multiprocessor systems-on-chip (MPSoC) are being developed in increasing numbers to support the high number of applications running on modern embedded systems. Designing and programming such systems prove to be a major challenge. Most of the current design methodologies rely on creating the design by hand, and are therefore error-prone and time-consuming. This also limits the number of design points that can be explored. While some efforts have been made to automate the flow and raise the abstraction level, these are still limited to single-application designs.

In this paper, we present a design methodology to generate *and* program MPSoC designs in a systematic and automated way for *multiple* applications. The architecture is automatically inferred from the application specifications, and customized for it. The flow is ideal for fast design space exploration (DSE) in MPSoC systems. We present results of a case study to compute the buffer-throughput trade-offs in real-life applications, H263 and JPEG decoders. The generation of the entire project takes about 100ms, and the whole DSE was completed in 45 minutes, including the FPGA mapping and synthesis.

1. INTRODUCTION

New applications for embedded systems demand complex multi-processor designs to meet real-time deadlines while achieving other critical design constraints like low power consumption and low area. With high consumer demand, the time-to-market has significantly reduced [1]. Multi-Processor Systems-on-Chips (MPSoCs) have been proposed as a promising solution for all such problems. But one of the key design challenges that remain is the fast design exploration of software and hardware implementation alternatives with accurate performance evaluation.

With the advent of multiple applications in embedded systems (e.g. smart phones, PDA, set-top boxes), this design problem extends to performance evaluation of multiple use-cases. A use-case is defined as the combination of applications that are active from all possible applications. Exploring their software and hardware implementation alternatives adds a new dimension to this design challenge. As shown in Figure 1, the ideal design flow to overcome this challenge is to extract the application specification from the C-code, sequential or parallel, and then use that to generate and synthesize the MPSoC platform. Unfortunately, the current design tools available only allow for single application, as shown by the shaded area in Figure 1. In addition, their design automation is limited [2, 3, 4], forcing designers to resort to manual architectural design which is time consuming and error-prone. This makes design space exploration (DSE) slow and often infeasible [3].

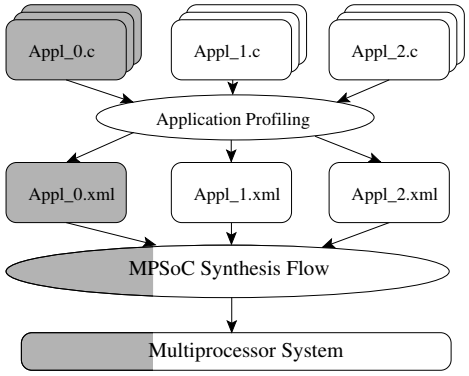

Fig. 1. Overall design flow

In this paper, we present *MAMPS (Multi-Application Multi-Processor Synthesis)* - a design-flow that takes in application(s) specifications and generates the entire MPSoC, specific to the input application(s) together with corresponding software projects for automated synthesis. This allows the design to be directly implemented on the target architecture. Applications are specified in the form of Synchronous Data Flow (SDF) graphs [5, 6]. SDF graphs are often used for modeling modern DSP applications [6] and for designing concurrent multimedia applications.

Our flow is unique in two aspects: (1) it allows fast DSE by automating the design generation and exploration, and (2) it supports multiple applications. To the best of our knowledge, there is no other existing flow to automatically map multiple applications to an MPSoC platform. The design space increases exponentially with increasing number of applications running concurrently; our flow provides a quick solution to that. To ensure multiple applications are able to execute concurrently, (1) we use non-blocking reads and writes that do not cause deadlock even with multiple

*This research was supported by HiPEAC Institution grant

applications, (2) we have an arbiter that ensures fairness and skips non-ready actors, and (3) we map channels to individual FIFOs to avoid head-of-line blocking.

The flow is used to develop a tool to generate designs targeting Xilinx platform FPGAs. This *MAMPS* tool is made available online for use by research community at [7]. FPGAs were selected as the target architecture as they allow rapid prototyping and testing. The tool is used to generate several multiple-application designs that have been tested on Xilinx University Virtex II Pro Board (XUPV2P) [8]. We present a case study on how our methodology can be used for design space exploration using JPEG and H263 decoders. We were able to explore 24 buffer-throughput tradeoff points with both applications running concurrently on real FPGA in about 45 minutes, including synthesis time.

The rest of the paper is organized as follows. Section 2 reviews the related work for architecture-generation and synthesis flows for multiprocessor systems. Section 3 introduces SDF Graphs. Section 4 gives an overview of our flow, *MAMPS*, while Section 5 describes the tool implementation. Section 6 presents results of experiments done to evaluate our methodology. Section 7 concludes the paper and gives a direction for future work.

2. RELATED WORK

The problem of mapping an application to architecture has been widely studied in literature. One of the recent works that is most related to our research is ESPAM [2]. This uses Kahn Process Networks (KPN) [9] for application specification. In our approach, we use SDF [5] for application specification instead. Further, our approach supports mapping of multiple applications, while ESPAM is limited for single applications. This is imperative for developing modern embedded systems which support more than tens of applications on a single MPSoC. The same difference can be seen between our approach and the one proposed in [4] where an exploration framework to build efficient FPGA multiprocessors is proposed.

The Compaan/Laura design-flow presented in [10] also uses KPN specification for mapping applications to FPGAs. However, their approach is limited to processor and a coprocessor. Our approach aims at synthesizing complete MPSoC designs. Another approach for generating application-specific MPSoC architectures is presented in [3]. However, most of the steps in this approach are done manually. Exploring multiple design iterations is therefore not feasible. In our flow, the entire flow is automated, including the generation of the final bit file that runs directly on the FPGA.

Yet another flow for generating MPSoC for FPGA has been presented in [11]. However, this flow focuses on generic MPSoC and not on application-specific architectures. Further, the work in [11] uses networks-on-chip for communication fabric, while in our approach dedicated links are used for communication to remove resource contention.

Table 1. Comparison of various approaches for providing performance estimates

	SDF^3 [12]	POOSL [13]	ESPAM [2]	*MAMPS*
Approach Used	Analysis	Simulation	FPGA	FPGA
Model Used	SDF	SDF	KPN	SDF
Single Appl	Yes	Yes	Yes	Yes
Multiple Appl	No	Yes	No	Yes
Speed	Fastest	Slow	Fast	Fast
Accuracy	Less	High	Highest	Highest
Dedicated FIFO	N. A.	No	No	Yes
Arbiter Support	N. A.	Yes	N. A.	Yes

Xilinx provides a tool-chain as well to generate designs with multiple processors and peripherals [8]. However, most of the features are limited to designs with only a bus-based processor-coprocessor pair with shared-memory. It is very time-consuming and error-prone to generate an MPSoC architecture and the corresponding software projects to run on the system. In our approach, MPSoC architecture is automatically generated together with the respective software projects for each core.

Table 1 shows the various design approaches that provide estimates of application performance by various means. The first method uses SDF models and computes the throughput of the application by analyzing the application graph. However, it is only able to predict the performance of single applications. The simulation approach presented in [13] uses POOSL [14] for providing application performance estimates. This is more accurate than analysis since more details can be modeled and their effects are measured using simulations. ESPAM is closest to our approach as they also use FPGA, but they do not support multiple applications. *MAMPS* supports multiple applications, and provides fast and accurate results.

3. SYNCHRONOUS DATA FLOW GRAPHS

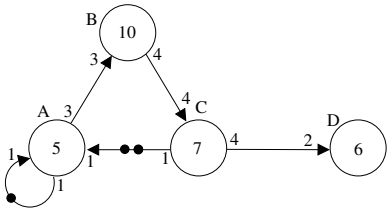

Fig. 2. Example of an SDF Graph

Figure 2 shows an example of an SDF Graph. There are four actors in this graph. As in a typical data flow graph, a directed edge represents the dependency between tasks. Tasks also need some input data (or control information) before they can start and usually also produce some output

data; such information is referred to as *tokens*. Actor execution is also called *firing*. An actor is called *ready* when it has sufficient input tokens on all its input edges and sufficient buffer space on all its output channels; an actor can only fire when it is ready.

The edges may also contain *initial tokens*, indicated by bullets on the edges, as seen on the edge from actor C to A in Figure 2. Buffers may be modeled as an edge with initial tokens. In such cases, the number of tokens on that edge indicates the buffer size available. When an actor writes data to such channels, the available size reduces; when the receiving actor consumes this data, the available buffer i.e. the token count, increases.

In the above example, only A can start execution from the initial state, since the required number of tokens are present on all of its incoming edges. Once A has finished execution it will produce 3 tokens on the edge to B. B can then proceed as it has enough tokens and upon completion produce 4 tokens on the edge to C.

4. *MAMPS* FLOW OVERVIEW

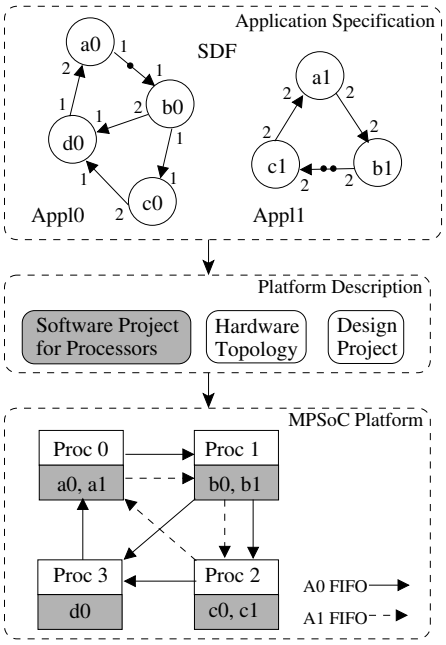

Fig. 3. Design flow

In this section, we present an overview of Multi-Application Multi-Processor Synthesis (*MAMPS*). Figure 3 shows an overview of our design flow. The application-descriptions are specified in the form of SDF graphs, which are used to generate the hardware topology. The software project for each core is produced to model the application(s) behavior. The project files specific to the target architecture are also produced to link the software and hardware topology. The desired MPSoC platform is then generated.

For example, in Figure 3, two example applications $Appl0$ and $Appl1$ are shown with 4 and 3 actors respectively. From these graphs, *MAMPS* generates the desired software and hardware components. The generated design in this example, has four processors with actors $a0$ and $a1$ sharing $Proc0$, while $d0$ being the only actor executing on $Proc3$. The corresponding edges in the graphs are mapped to FIFO (first-in-first-out) channels as shown.

The flow can be used to design multiprocessor systems that support multiple applications. The target platform can be either FPGA or even an ASIC design. The current tool implemented uses Xilinx tool-chain (explained more in Section 5). The target architecture in this tool is Xilinx Virtex II Pro FPGAs. Even for designs that target ASIC platforms, our tool is useful for doing rapid prototyping and performance evaluation.

4.1. Application Specification

```
<application id="H263">
  <actor name="VLD">
    <port name="Recon" type="in" rate="1"/>
    <port name="IQ" type="out" rate="2376"/>
    <executionTime time="120000"/>
  </actor>
  <actor name="IQ">
    <port name="VLD" type="in" rate="1188"/>
    <port name="IDCT" type="out" rate="1188"/>
    <executionTime time="9600"/>
  </actor>
```

Fig. 4. Snippet of H263 application specification.

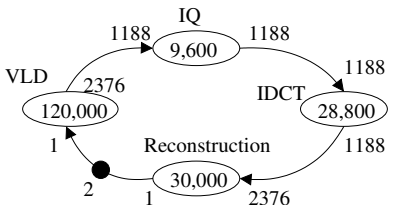

Fig. 5. SDF graph for H263 application

Application specification forms an important part of the flow. The applications for our flow are specified in *xml* format. A snippet of application specification file for H263 decoder is shown in Figure 4. The corresponding SDF graph is shown in Figure 5. The application here has been modeled from the data presented in [15]. The figures illustrate how easy it is to write the application-specification.

While the specification above is obtained through application profiling, it is also possible to use tools to obtain the SDF description for an application from its code directly. Compaan [10] is one such example that converts sequential description of an application into concurrent tasks[1]. These can then be converted into SDF graphs easily.

[1]It actually converts a sequential application into a limited set of KPN graph, namely graphs that are also cyclo-static data flow graphs (CDFG).

The specification file contains details about how many actors are present in the application, and how they are connected to the other actors. The execution time of the actors and their memory usage on the processing core is also specified. For each channel present in the graph, the file describes if there are any initial tokens present on it. The buffer capacity of a particular channel is specified as well.

When multiple applications are to be mapped to a common architecture, our flow allows use-case depiction. Very often in a given system, the system might support multiple applications, but only a few of them might be active at a given point in time. The use-case information may be supplied at design time together with application specification.

4.2. Platform Generation

From the *xml* descriptions, the platform description is generated. For single application, each actor is mapped on a separate processor node, while for multiple applications, nodes are shared among actors of different applications. The total number of processors in the final architecture corresponds to the maximum number of actors in any application. For example, in Figure 3, a total of 4 processors are used in the design. For processors that have multiple actors mapped onto them, an arbitration scheme is also generated.

All the edges in an application are mapped on a unique FIFO channel. This creates an architecture that mimics the applications directly. Unlike processor sharing for multiple applications, the FIFO links are dedicated as can be seen in Figure 3. As opposed to a network or a bus-based infrastructure, the dedicated links remove the possible sources of contention that can limit the performance.

Since we have multiple applications running concurrently, there is often more than one link between some processors. Even in such cases, multiple FIFO channels are created. This avoids head-of-line blocking that can occur if one FIFO is shared for multiple channels [16]. Further, multiple channels reduce the sources of contention in the system.

As mentioned in Section 3, firing of an actor requires sufficient input tokens to be present on all incoming edges. This implies that an actor might not be able to execute if the incoming buffers underflow. The same holds when the output buffers of an actor fill up. While this does not cause any problem when only one actor is mapped on a node, in the case of multiple actors, the other possibly *ready* actors might not be able to execute while the processor sits idle. To avoid this, non-blocking reads and writes are carried out, and if any read or write is unsuccessful, the processor is not blocked, but simply executes the other actor for which there are sufficient input tokens and space on all output edges.

5. TOOL IMPLEMENTATION

In this section, we describe the tool we developed based on our flow to target Xilinx FPGA architecture. The processors in the MPSoC flow are mapped to microblaze processors [8]. The FIFO links are mapped on to Fast Simplex Links (FSL). These are uni-directional point-to-point communication channels used to perform fast communication[2]. The FSL depth is set according to the buffer-size specified in the application.

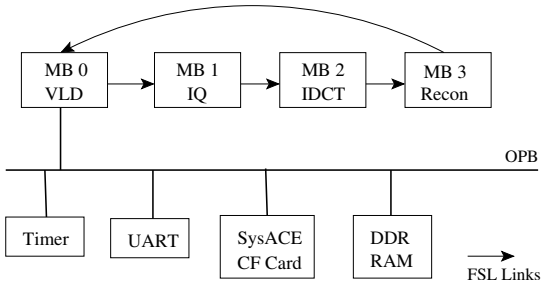

Fig. 6. Hardware topology of the generated design for H263

Example architecture for H263 application platform is shown in Figure 6. This consists of several microblazes with each actor mapped to a unique processor with additional peripherals such as Timer, UART, SysACE and DDR-RAM. While the UART is useful for debugging the system, SysACE Compact Flash card allows for convenient performance evaluation for multiple use-cases by running continuously without external user interaction. Timer Module and DDR RAM are used for profiling the application and for external memory access respectively.

In our tool, in addition to the hardware topology, the corresponding software for each processing core is also generated automatically. For each software project, appropriate functions are inserted that model the behavior of the task mapped on the processor. This can be a simple delay function if the behavior is not specified. If the actual source code for the function is provided, the same can be easily inserted to the project. This also allows functional verification of applications on a real hardware platform. Routines for measuring performance, and sending results to the serial-port and to the CF card on-board are also generated for *MB0*.

Our software generation ensures that the tokens are read from (and written to) the appropriate FSL link in order to maintain progress, and to ensure correct functionality. Writing data to the wrong link can easily throw the system in deadlock. XPS project files are also automatically generated to provide the necessary interface between hardware and software components.

6. EXPERIMENTS AND RESULTS

In this section, we present some of the results that were obtained by implementing several real and randomly-generated application SDF graphs using our design flow described in Section 4. The main objective of this experiment

[2]Current version of MicroBlaze from Xilinx supports up to 8 FSL links

is to show that our flow reduces the implementation gap between system level and RTL level design. We show that our flow allows for more accurate performance evaluation using an emulation platform compared to simulation [14] and analysis. In addition, we present a case study using JPEG and H263 applications to show how our tool can be used for efficient design space exploration for multiple application use-cases. Our implementation platform is the Xilinx XUP Virtex II Pro Development Board with an *xc2vp30* FPGA on-board. Xilinx EDK 7.1i and ISE 7.1i were used for synthesis and implementation. The newer versions of the corresponding tools can also use the generated designs by automatically upgrading them. All tools run on a Pentium Core at 3GHz with 1GB of RAM.

Table 2. Comparison of throughput for different applications obtained on FPGA with simulation

	Appl 0			Appl 1		
Use-case	Sim	FPGA	Var %	Sim	FPGA	Var %
A	3.96	3.30	-20.05	1.99	2.15	**7.49**
B	3.59	3.31	-8.63	1.80	1.61	-11.90
C	2.64	2.74	**3.67**	1.88	1.60	-17.37
D	3.82	3.59	-6.32	0.85	0.77	-10.51
E	4.31	4.04	-6.82	1.44	1.35	-6.80
F	5.10	4.73	-7.75	0.51	0.48	-5.79
G	4.45	4.25	-4.55	1.11	0.97	-14.66
H	4.63	4.18	-10.65	1.16	1.05	-10.29
I	4.54	4.03	-12.48	2.27	2.13	-6.51
J	4.33	3.97	-8.92	1.08	1.00	-8.41

In order to verify our design flow, we generated 10 random application graphs with 8 to 10 actors each using the tool SDF^3 [12], and generated designs with 2 applications running concurrently. Results of 10 such random combinations have been summarized in Table 2. The results are compared with those obtained through simulation. We observe that in general, the application throughput measured on FPGA is lower than simulation by about 8%. This is because our simulation model does not take into account the communication overhead. However, in some cases we observe that performance of some applications improved (shown in bold in Table 2). This is rather unexpected, but easily explained when going into a bit of detail.

Communication overhead leads to the actor execution taking somewhat longer than expected thereby delaying the start of the successive actor. This causes the performance of that application to drop. However, since we are dealing with multiple application use-cases, this late arrival of one actor might cause the other application to execute earlier than that in simulation. This is exactly what we see in the results. For the two use-cases in which this happens - namely A and C, the throughput of the other applications is significantly lower: 20 and 17 percent respectively. This also proves that the use-cases of multiple applications concurrently executing is more complex to analyze and reason about than a single application case.

6.1. DSE Case Study

Here we present a case study of using our design methodology for doing a design space exploration and computing the optimal buffer requirement. Minimizing buffer-size is an important objective when designing embedded systems. We explore the trade-off between buffer-size used and throughput obtained for multiple applications. For single applications, the analysis is easier and has been presented earlier [17]. For multiple applications, it is non-trivial to predict resource usage and performance, because multiple applications cause interference when they compete for resources. This has been proven as shown in Table 2 above.

The case study is performed for JPEG and H263 decoder applications. The SDF models of the two applications were obtained from the description in [18] and [15] respectively. In this case study, the buffer size has been modeled by the initial tokens present on the incoming edge of the first actor. The higher this initial-token count, the higher the buffer needed to store the output data. In the case of H263, each token corresponds to an entire decoded frame, while in the case of JPEG, it is the complete image.

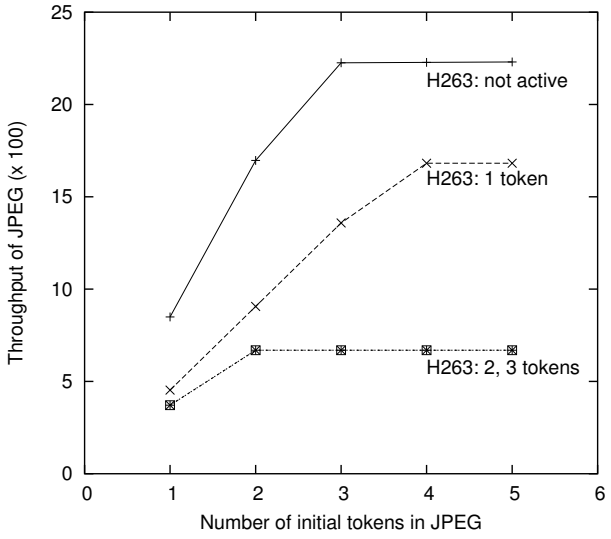

Fig. 7. Effect of varying initial tokens on JPEG throughput

Figure 7 shows how the throughput of JPEG decoder varies with increasing number of tokens in the graph. A couple of observations can be made from this figure. When the number of tokens (i.e. buffer-size in real application) is increased, the throughput also increases until a certain point, after which it saturates. When JPEG decoder is the only application running (obtained by setting the initial tokens in H263 to zero), we observe that its throughput increases al-

Table 3. Time spent on DSE of JPEG-H263 combination

	Manual Design	Generating Single Design	Complete DSE
Hardware Generation	$\sim$ 2 days	40ms	40ms
Software Generation	$\sim$ 3 days	60ms	60ms
Hardware Synthesis	35:40	35:40	35:40
Software Synthesis	0:25	0:25	10:00
Total Time	$\sim$ 5 days	36:05	45:40
Iterations	1	1	24
Average Time	$\sim$ 5 days	36:05	1:54
Speedup	–	1	19

most linearly till 3. We further observe that increasing the initial tokens of H263 worsens the performance of JPEG, but only until a certain point.

Design Time

The time spent on the exploration is an important aspect when estimating the performance of big designs. The JPEG-H263 system was also designed by hand to estimate the time gained by using our tool. The hardware and software development took about 5 days in total to obtain an operational system. In contrast, our tool takes a mere 100ms to generate the complete design. Table 3 shows the time spent on various parts of the flow. The Xilinx tools take about 36 min to generate the bit file together with the appropriate instruction and data memories for each core in the design.

Our approach is very fast and is further optimized by modifying only the relevant software and keeping the same hardware design for different use-cases. Since software synthesis step takes only about 25 sec in our case study, the entire DSE for 24 design points was carried out in about 45 min. This hardware-software co-design approach results in a speed-up of about 19 when compared to generating a new hardware for each iteration. As the number of design points are increased, the cost of generating the hardware becomes negligible and each iteration takes only 25 seconds. The design occupies about 40% of logic resources on FPGA and close to 50% of available memory. This study is only an illustration of the usefulness of our approach for DSE for multiprocessor systems.

7. CONCLUSIONS AND DISCUSSIONS

In this paper, we propose a design-flow to generate application-specific architecture designs. Our approach takes the description of the application(s) and produces the corresponding MPSoC platform. This is the first flow that allows mapping of multiple applications on a single platform. The tool developed using this flow is made available online for the benefit of research community [7]. The flow allows the designers to traverse the design space quickly, thus making DSE of even concurrently executing applications feasible. A case study is presented to find the trade-offs between buffer-size and performance when JPEG and H263 run together on a platform.

However, the number of applications that can be concurrently mapped on the FPGA is limited by the hardware resources present. When synthesizing designs with applications of 8-10 actors and 12-15 channels, we found that it was difficult to map more than four applications simultaneously due to resource constraints, namely block RAMs. A bigger FPGA would certainly allow bigger designs to be tested.

For future work, we would like to develop and automate more ways of design space exploration, for example trying different partitions of applications. We would also like to try different kinds of arbiters in the design to improve fairness and allow for load-balancing between multiple applications. We also wish to extend MAMPS to include support for generating heterogeneous platforms in our flow.

8. REFERENCES

[1] A. Jerraya and W. Wolf, *Multiprocessor Systems-on-Chips*. Morgan Kaufmann, 2004.

[2] H. Nikolov *et al.*, "Multi-processor system design with ESPAM," *4th CODES+ISSS*, pp. 211–216, 2006.

[3] D. Lyonnard *et al.*, "Automatic generation of application-specific architectures for heterogeneous multiprocessor system-on-chip," *38th DAC*, pp. 518–523, 2001.

[4] Y. Jin *et al.*, "An automated exploration framework for FPGA-based soft multiprocessor systems," *3rd CODES+ISSS*, pp. 273–278, 2005.

[5] E. A. Lee and D. G. Messerschmitt, "Static scheduling of synchronous dataflow programs for digital signal processing," *IEEE Transactions on Computers*, 1987.

[6] S. Siram and S. Bhattacharyya, *Embedded Multiprocessors; Scheduling and Synchronization*. Marcel Dekker, 2000.

[7] "Application-Specific MPSoC Architecture Synthesis," Available from: Username: `guest`, Password: `fpl` `http://www.ics.ele.tue.nl/~akash/mamps/`, 2007.

[8] "Xilinx," Available from: http://www.xilinx.com, 2007.

[9] G. Kahn, "The semantics of a simple language for parallel programming," *Information Processing*, vol. 74, pp. 471–475, 1974.

[10] T. Stefanov *et al.*, "System design using Kahn process networks: the Compaan/Laura approach," *DATE*, pp. 340–345, 2004.

[11] A. Kumar *et al.*, "An FPGA Design Flow For Reconfigurable Network-Based Multi-Processor Systems On Chip," *DATE*, pp. 117–122, 2007.

[12] S. Stuijk *et al.*, "SDF3: SDF For Free," *6th ACSD*, pp. 276–278, 2006.

[13] A. Kumar *et al.*, "Resource manager for non-preemptive heterogeneous multiprocessor system-on-chip," *4th Workshop on Estimedia*, pp. 33–38, 2006.

[14] B. Theelen *et al.*, "Software/Hardware Engineering with the Parallel Object-Oriented Specification Langauge," in *5th MEMOCODE*, pp. 139–148, 2007.

[15] R. Hoes, "Predictable Dynamic Behavior in NoC-based MPSoC," Available from: www.es.ele.tue.nl/epicurus/, 2004.

[16] "Head-of-line blocking," Available from: http://en.wikipedia.org/wiki/Head-of-line_blocking, 2007.

[17] S. Stuijk *et al.*, "Exploring trade-offs in buffer requirements and throughput constraints for synchronous dataflow graphs," *43rd DAC*, pp. 899–904, 2006.

[18] E. de Kock, "Multiprocessor mapping of process networks: a JPEG decoding case study," *15th ISSS*, pp. 68–73, 2002.