# A Composable and Predictable MPSoC Design Flow for Multiple Real-Time Applications

Seyed-Hosein Attarzadeh-Niaki*, Ekrem Altinel*, Martijn Koedam†, Anca Molnos‡, Ingo Sander*, and Kees Goossens†

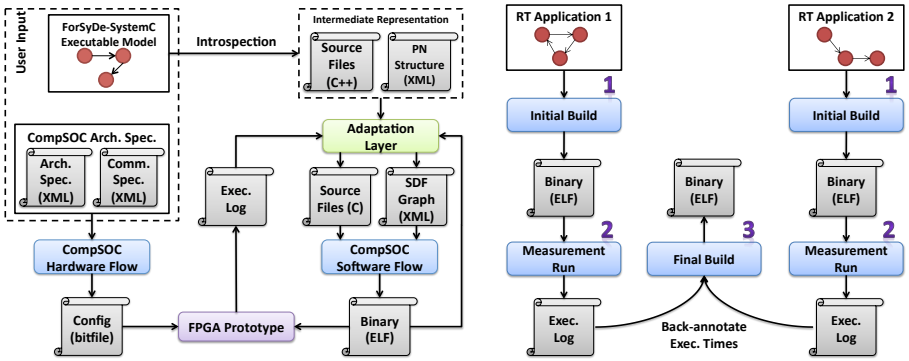
* KTH Royal Institute of Technology, Sweden
{shan2,altinel,ingo}@kth.se
† Eindhoven University of Technology, Netherlands
{M.L.P.J.Koedam,K.G.W.Goossens}@tue.nl
‡ CEA-LETI, France
Anca.MOLNOS@cea.fr

**Abstract.** Design of real-time MPSoC systems including multiple applications is challenging because temporal requirements of each application must be respected throughout the entire design flow. Currently the design of different applications is often interdependent, making converge to a solution for each application difficult. This paper proposes a compositional method to design applications independently, and then to execute them without interference. We define a formal modeling framework as a suitable entry point for application design. The models are executable, which enables early detection of specification errors, and include the formal properties of the applications based on well-defined models of computation. We combine this with a predictable MPSoC platform template that has a supporting design flow but lacks a simulation front-end. The structure and behavior of the application models are exported to an intermediate format via introspection which is iteratively adapted for the backend flow. We identify the problems arising in this adaptation and provide appropriate solutions. The design flow is demonstrated by a system consisting of two streaming applications where less than half of the design time is dedicated to operating on the integrated system model.

**Keywords:** System-level design languages, Automated design flow, Real-time applications, Composable system, Time-predictable architectures

## 1 Introduction

Embedded system designers are required to integrate an increasing number of complex applications running on a single system on chip. This calls for design flows which first, start from abstract application models and implement them in a fully automated fashion; and second, support designing each application in isolation while preserving its behavior in the integrated system. The challenge becomes more harsh for embedded systems that have real-time constraints where the design flows and the target platforms also need to preserve the timing properties of each application throughout the entire design flow.




(a) (b) The equivalent SDF graph

**Fig. 1.** (a) The proposed system design flow. (b) Build iterations of the flow.

Code generator backends from model-based design tools for real-time processors have been proposed [16, 13] but complexities of a real-time MPSoC design flow are not fully addressed in these works. Existing design flows that perform real-time analysis for MPSoCs are presented in [7, 15, 19, 8] without expressing the applications and components in an executable formal model and hence, a potential bug in the specification will be detected late in the design flow and makes it also harder to track it in the original specification. Additionally, running multiple applications on the same platform is not supported in these works.

We argue that proper design flows for real-time systems require to *a*) start from abstract application models with formal semantics that are executable to enable detection of specification bugs early in the design flow, avoiding long design iterations; *b*) apply *automated analysis and synthesis* methods supported by the applications' formalisms; and *c*) target platforms which provide *time-predictable* execution services to faithfully implement the application behavior; and support *composable* system design by integrating isolated applications.

Building on a common formal base in form of Models of Computation (MoCs), we introduce such a flow for multiple real-time streaming applications which integrates **For**mal **Sy**stem **De**sign (ForSyDe), as a modeling and simulation framework, with CompSOC, a design flow and platform template for predictable MPSoCs (Fig. 1a). However, our methods are applicable to any combination of system-level design languages and target platforms which satisfy the above requirements. In the SystemC implementation of ForSyDe [2] a formally defined representation of the executable system models can be exported as an intermediate format. CompSOC is a network-on-chip based MPSoC platform template which provides predictable and composable execution services to the applications. The platform currently has a design flow which performs automated mapping, compilation, and synthesis from a non-validatable implementation model, where the structure (as XML files) and behavior of the application (as a set of C files) are captured separately. By iterative adaptation of exported ForSyDe-SystemC models, we integrate them with the CompSOC design flow and demon-

strate an automated flow starting from high-level simulate-able formal application models, achieving a correct-by-construction design flow. Such an adaptation involves annotating platform-specific memory and timing requirements of the application elements which is obtained by rapid performance evaluation of the application on the platform.

The contributions of this work are summarized as:

- an automated and composable design flow implementing abstract executable models of multiple applications on predictable platforms (Section 3);
- adaptation of the specification models captured in the formal modeling framework (Section 4) to the implementation models accepted by the predictable platform design flow (Section 5) which involves rapid performance evaluation of the application models on the platform (Section 6);
- demonstration of the flow in action using two applications from the consumer electronics domain (Section 7).

## 2 Related Work

Several tools and design flows have been proposed for real-time MPSoCs, but none of them fully address different aspects of the problem.

The industrial tool Simulink can produce plain C code from executable real-time models. In [9], a model-based design flow for cyber-physical systems is presented in ten design steps, but it is not fully automated. These approaches bring interesting ideas in the field, however they do not consider real-time constraints throughout the entire design-flow.

The PTIDES flow [9] targets event-based applications described in a Programming Model with discrete-event semantics. First the temporal behavior and causality relations of the application model are analyzed and then actor and OS code are generated. This code may be compiled and bound to three hardware platforms, out of which one, namely the XMOS board, provides real-time services. The flow simulates the application code bound on the platform to verify and validate the design. The effects of the binding on the temporal behavior are hence not formally analyzed, which may lead to run-time constraints violations.

Code generator backends for real-time processors has been proposed for executable Ptolemy [5] models, notably for JOP [16], which is a Java real-time processor, and PRET [13] which aims at providing predictability in its architecture. However, complexities of a real-time MPSoC design flow are not fully addressed in these works.

Existing design flows that perform real-time analysis for MPSoCs are presented in [7, 15, 19]. The approaches in [15, 19] target the streaming domain and have as input an XML description of the application. The flow produces the code of the application, and the information necessary to bind this application on a predictable platform. Real-time constraints are guaranteed, as the temporal behavior of application, the binding and the platform are formally verified. Note, as mentioned before, we utilize the SDF$^3$ tool for formal temporal analysis in our flow. The SymTA/S [7] framework models an SoC as a set of inter-connected components. Each component is modeled by an event stream. Classical formal real-time analysis can be applied to individual components and a formalism for

composing streams is proposed to analyze an entire system. The distributed operation layer (DOL) MPSoC software design flow [8] targets dataflow real-time streaming applications and automatically creates an implementation as well as formal performance analysis models for system validation. However in these approaches applications and components are not expressed in an executable formal model, hence a potential bug in the specification will be detected late in the design flow and makes it also harder to track it in the original specification.

Deaedalus$^{RT}$ methodology [3] starts with a Static Affine Nested Loop Program (SANLP) and uses the hard real-time multiprocessor scheduling techniques to analyze the system from intermediate CSDF models and performs code generation via Polyhedral Process Networks (PPNs). In contrast, being based on the ForSyDe framework, our input models have explicit parallelism and are easier to extend and interact with other MoCs for a heterogeneous system design. On the platform side, CompSOC can run multiple applications based on resource reservation composably, reducing the run-time scheduling overhead for admission control. Also, our flow ends with a full FPGA-based prototype while Daedalus-based flows have reported simulation backends.

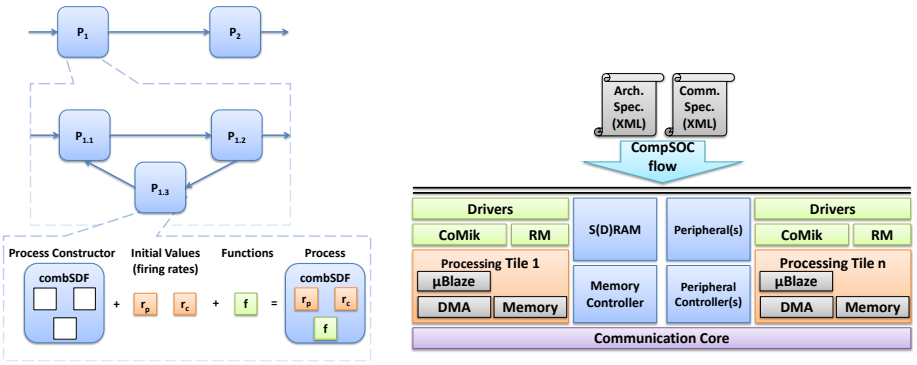
## 3 The Proposed Design Flow

In the high-level view of the proposed flow depicted in Fig. 1a, the designer uses the Synchronous Data Flow (SDF) MoC library of ForSyDe-SystemC to create formal executable system models. This model is used both for validation by simulation and also for automated generation of an intermediate representation of the specification model as a set of XML and C++ files via introspection. Since both ForSyDe and CompSOC are based on formally defined MoCs, they can be interconnected in a correct-by-construction manner by adapting the specification model to the target implementation model. We assume the hardware platform is given as an architecture description and a communication description file and it is not going to be explored in the design flow. To hand over ForSyDe-SystemC models to the CompSOC flow, an automated adaptation step is needed to iteratively add the missing platform information to the model (Section 6).

The proposed flow supports multiple applications and runs in three stages (Fig. 1b): *1*) *initial build*, which performs the model transformation assuming constant values for platform-dependent metrics and generates a binary file for a single core mapping of each application; *2*) *measurement run*, where the generated binary and source files are analyzed to extract the actor memory requirements and token sizes, then a multi-core mapping is done and applications are run to measure the execution times of the actors; and *3*) *final build*, in which the SDF graphs of all applications with all the platform-dependent metrics back-annotated are merged and then the final mapping and software synthesis is done to generate the final binary. The first two phases are performed individually for each applications, since CompSOC preserves composability [1].

## 4 The Modeling Framework

In ForSyDe-SystemC [2], a system model is structured as a hierarchical concurrent process network. Processes communicate and synchronize *only* using signals




(a) (b)

**Fig. 2.** (a) Example of a hierarchical system model in the SDF MoC of ForSyDe. Leaf processes are built using process constructors. (b) Overview of the CompSOC platform and its generation flow.

and there is no global state in the system. Hierarchy does not imply any semantics and enables IP reuse. Fig. 2a is an example of a system model in which $p_1$ is a composite process formed by composition of leaf processes $p_{1.1}$, $p_{1.2}$, and $p_{1.3}$.

Each leaf process in the process network belongs to a specific Model of Computation (MoC) [12]. Several MoCs are supported for system modeling in ForSyDe. This work addresses the synthesis of SDF models [11] which fits very well to many streaming applications and is supported by analysis methods for consistency checking, temporal scheduling and mapping to single- and multi-processor systems.

Leaf processes are created using formally defined constructs chosen from the ForSyDe library called *process constructor* which are provided with side-effect-free functions and/or initial values. In Fig. 2a, a process is created using the *combSDF* process constructor which is supplied with the firing function $f$ and two initial values representing the production and consumption rates $r_p$ and $r_c$ to gain a process with the semantics of an actor in the SDF MoC. By using the concept of process constructors the requirements for the computation and communication semantics of the processes are satisfied by construction and the designer is liberated from writing redundant code and focuses on the pure functionality of the processes.

There are two key process constructors in the SDF MoC; *1)* $combSDF_m$, which denotes an $m$-input combinational (i.e., stateless) process constructor in the SDF MoC; and *2)* $delaySDF_n$ which delays a signal by $n$ elements. Because processes in ForSyDe are formally defined as mathematical functions operating on input signals and returning a single output signal, tuples of values are used to model multiple signals. Such a signal can be converted into multiple signals using special family of processes called *unzip* and created using *zip* processes.

A special feature of ForSyDe-SystemC models is that in addition to simulation, the constructed executable models can export their internal structure and behavior as an intermediate representation via *introspection*. The exported repre-

sentation can be used to feed the models to analysis and synthesis tools, without developing a full-fledged compiler infrastructure. This intermediate representation contains the structure of the process network, the process constructors used to build leaf processes, and the parameters passed to build the processes. These parameters include both the initial values and also the source code of the functions which describing the behavior of the processes [2].

## 5 The Execution Platform

CompSOC [6] provides predictable execution services to the applications and can run multiple applications without interference in a composable manner [1]. Time-division multiplexing (TDM) is used to provide time-predictable execution, communication, and memory access services while composability is achieved by using arbiters which prevent indefinite locking of shared resources.

The hardware is a collection of processor and memory tiles interconnected by a dAElite NoC [18]. The NoC consists of protocol shells, which serialize the parallel protocol, network interfaces (NIs), which (de-)packetize the information and inject/collect them to/from the network in a TDM fashion, and routers. The processor tiles include a Microblaze softcore, local instruction and data memories, a set of communication memory blocks and direct memory access (DMA) engines for inter-tile communication, and other peripherals. Memory tiles are divided into two parts, namely front-end and back-end. The front-end contains a number of blocks to achieve composability while the back-end guarantees the predictability of the resource [1].

CompSOC runs a composable RTOS which uses two level scheduling for applications [14]. The inter-application scheduler is a part of the minimalistic RTOS implementation named CoMik, which uses the TDM technique to provides individual performance guarantees to multiple integrated applications. The intra-application scheduler (task scheduler) supports executing application tasks with different semantics, namely KPN, CSDF, and also time-triggered MoCs.

The supporting CompSOC design flow for SDF applications consists of three sub-flows, hardware generation, mapping and software compilation flow. The hardware generation flow takes in the communication and architecture models of the platform and performs dimensioning, allocation, verification, instantiation and synthesis of the hardware architecture. Based on the SDF graph of the application and the architecture model of the platform, the mapping flow invokes a mapping tool to explore the design space and generates a mapping and schedule of the application onto the platform, and finally synthesizes the software for each core. Consecutively, the compilation flow is invoked for each tile in the platform and the generated binary ELF file is merged with the platform bit-stream file generated by the hardware generation flow.

## 6 Adapting the Flows By Rapid Performance Evaluation

To enable full automation, the abstract untimed input model is adapted for synthesis. The execution times and memory requirements of the actors and also the token sizes are estimated using a rapid estimation technique. Helper processes

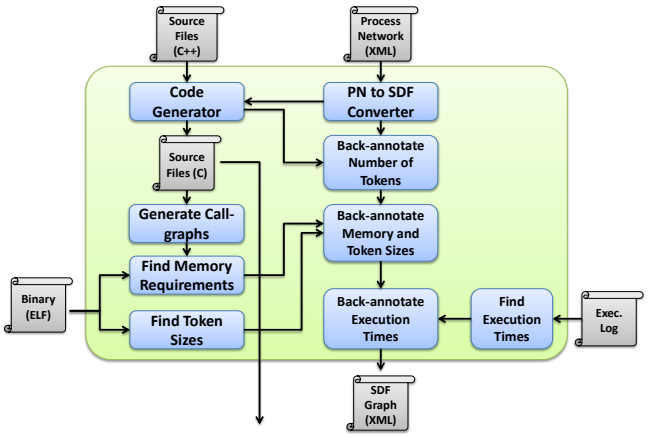
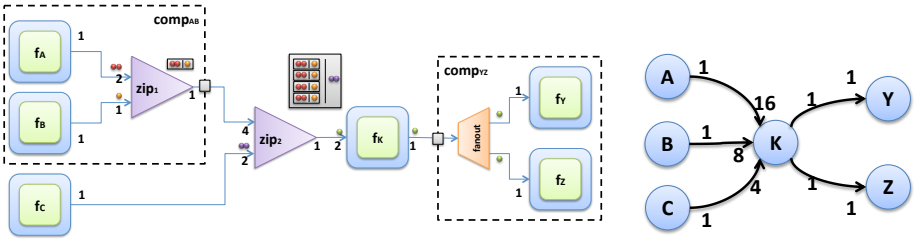



**Fig. 3.** The adaptation layer between the flows




(a) A process network modeled in the SDF MoC of ForSyDe

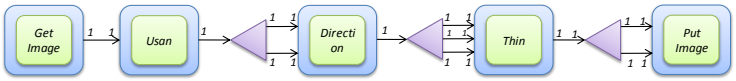
(b) The equivalent SDF graph

**Fig. 4.** Automatic conversion of ForSyDe process networks to SDF graphs

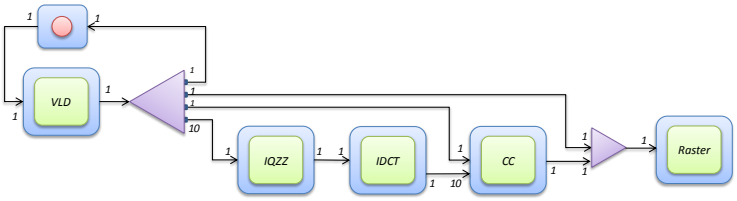
in ForSyDe process networks are transformed to their equivalent SDF actors. Additionally, the interface and API of the functions are different and conversion between them is required. Fig. 3 is a simplified view of the adaptation stage which transforms the models between the flows.

First, transforming ForSyDe process networks to SDF graphs involves *a*) flattening their static non-recursive hierarchy; *b*) removing *zip*, *unzip*, and *fanout* processes by integrating them with multi-input/output actors; *c*) converting *source* and *constant* state-full processes to actors with self-edges; and *d*) converting delay elements to initial tokens on the graph edges . Fig. 4a shows the structure of a process network as described in the SDF MoC of ForSyDe and how it should be exported to the CompSOC backend as an SDF graph. Note that while removing *zip*s and *unzip*s, all actor-to-actor paths are considered and the production and consumption rates are adapted using a traversal algorithm. Conversions are realized using XSL tranformations [10].

During the measurement run, the DWARF [4] data from the binary files are analyzed to extract the data token sizes after initial software compilation for the target platform. Together with the information retrieved about the memory sizes




(a) The ForSyDe process network of SUSAN




(b) The ForSyDe process network of the JPEG decoder

**Fig. 5.** The considered application models.

of the functions by the GNU tools, the call-graphs generated using the LLVM and Clang tool chains are analyzed to obtain the memory requirements of the actor functions. Also, the execution times of individual actors are measured by inspecting the execution log of each application running on hardware.

Generating code for the SDF actors not only needs adaptations between the API's for the two flows, but also requires extracting and matching information such as the initial token values from the ForSyDe intermediate representation, generating code for *constant* processes, and generating the required header functions and initialization code.

## 7 Case Study

To demonstrate the feasibility of the proposed design flow, we apply it to two applications from the multimedia domain, namely SUSAN edge detection and JPEG decoder. Smallest Univalue Segment Assimilating Nucleus (SUSAN) [17] includes three signal processing algorithms, out of which the edge detection algorithm is considered here. The image is partitioned into smaller blocks and the following steps are applied to each pixel: *1*) a mask is applied to an area (called USAN) centered around each pixel of interest; *2*) the direction of the edge is detected by calculating the momentums of the USAN area; and *3*) thinning is applied on the edges to clarify the pixels. JPEG decoding of an image in performed in five steps by: *1*) parsing the image headers and decompressing the input as a series of 8×8 pixel blocks; *2*) inverse quantization and reordering of blocks; *3*) combining pixel blocks into RGB pixel values; and *4*) putting the pixel values in the final image.

In the first step of the flow, a model of each applications is developed using the SDF MoC. Fig. 5a and Fig. 5b illustrate their process networks as captured by ForSyDe-SystemC. These models are verified by simulation with the same testbench that is used later in the measurement run by checking the produced output against a reference considered correct. The design flow is invoked from the command line using a make command. The first two main stages of the design

**Table 1.** Execution times of different parts of the flow (in seconds)

| | Simulate & Introspect | Individual Flow | Combined Flow |
|---|---|---|---|
| SUSAN | 0.16 | 549 | 1056 |
| JPEG | 0.01 | 509 | |

**Table 2.** Execution times of the actors and the mapped application

| | | | | | | |
|---|---|---|---|---|---|---|
| SUSAN | getImage | USAN | Direction | Thin | putImage | App. |
| | 20077 | 1177105 | 833912 | 35843 | 15866 | 1356352 |
| JPEG | VLD | IQZZ | IDCT | CC | Raster | App. |
| | 626884 | 4294 | 15505 | 21284 | 1327 | 61618352 |

flow, namely the initial build and the measurement run stages are executed for the two applications separately. After the back annotation of the execution times, the applications are merged and the final mapping is obtained. Table 1 summarizes the execution times of each part of the flow. [1] The final phase which is executed for both applications (1056 Sec) is less than half of the total execution time of the flow (2114 Sec). After producing the bit-stream file, the system is run and both of the applications are verified to produce the correct output. The maximum measured execution times of each actor and the mapped application are presented in Table 2. These times remain constant while the applications run individually and both together on the platform.

## 8 Conclusion and Future Work

We have proposed a fully automated design flow for multiple real-time signal processing applications which compiles formal executable specifications to a predictable MPSoC template. The design flow *a*) moves the design entry to a higher level of abstraction since functional models can be simulated efficiently in SystemC; *b*) provides an automated path to synthesis using the introspection feature of ForSyDe and the CompSoC tool suite; and *c*) uses rapid performance estimation of the applications on the target platform to estimate platform-specific metrics of the applications. Only half of the execution time of the flow is consumed for the combined application model since the platform ensures the composability of the individually analyzed applications.

We plan to enrich the flow by supporting additional MoCs such as the Synchronous (SY) MoC for control-oriented behavior and also aim at a mixed-criticality design flow.

## References

1. Akesson, B., Molnos, A., Hansson, A., Angelo, J., Goossens, K.: Composability and predictability for independent application development,verification, and execution.

[1] Experiments are run on a 64 bit Linux machine with a Core i7 CPU running at 3.07GHz with 24Gb of memory.

In: Hübner, M., Becker, J. (eds.) Multiprocessor System-on-Chip: Hardware Design and Tool Integration, pp. 25–56. Springer New York (2011)
2. Attarzadeh Niaki, S., Jakobsen, M., Sulonen, T., Sander, I.: Formal heterogeneous system modeling with SystemC. In: Proceedings of the Forum on Specification and Design Languages (FDL). pp. 160–167 (Sept 2012)
3. Bamakhrama, M.A., Zhai, J.T., Nikolov, H., Stefanov, T.: A methodology for automated design of hard-real-time embedded streaming systems. In: Proceedings of the Conference on Design, Automation and Test in Europe. pp. 941–946. DATE '12, EDA Consortium, San Jose, CA, USA (2012)
4. Eager, M.J., Consulting, E.: Introduction to the DWARF debugging format (2007), `http://www.dwarfstd.org`
5. Eker, J., Janneck, J., Lee, E., Liu, J., Liu, X., Ludvig, J., Neuendorffer, S., Sachs, S., Xiong, Y.: Taming heterogeneity - the Ptolemy approach. Proceedings of the IEEE 91(1), 127–144 (Jan 2003)
6. Goossens, K., Azevedo, A., Chandrasekar, K., Gomony, M.D., Goossens, S., Koedam, M., Li, Y., Mirzoyan, D., Molnos, A., Nejad, A.B., Nelson, A., Sinha, S.: Virtual execution platforms for mixed-time-criticality systems: The CompSOC architecture and design flow. SIGBED Rev. 10(3), 23–34 (Oct 2013)
7. Hamann, A., Jersak, M., Richter, K., Ernst, R.: A framework for modular analysis and exploration of heterogeneous embedded systems. Real-Time Systems (2006)
8. Huang, K., Haid, W., Bacivarov, I., Keller, M., Thiele, L.: Embedding formal performance analysis into the design cycle of MPSoCs for real-time streaming applications. ACM Trans. Embed. Comput. Syst. 11(1), 8:1–8:23 (Apr 2012)
9. Jensen, J., Chang, D., Lee, E.: A model-based design methodology for cyber-physical systems. In: Proceedings of the 7th International Conference on Wireless Communications and Mobile Computing (IWCMC). pp. 1666–1671 (July 2011)
10. Kay, M., et al.: XSL transformations (XSLT) version 2.0. W3C Recom. (2007)
11. Lee, E., Messerschmitt, D.: Synchronous data flow. Proceedings of the IEEE 75(9), 1235–1245 (Sept 1987)
12. Lee, E., Sangiovanni-Vincentelli, A.: A framework for comparing models of computation. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 17(12), 1217–1229 (Dec 1998)
13. Lickly, B., Liu, I., Kim, S., Patel, H.D., Edwards, S.A., Lee, E.A.: Predictable programming on a precision timed architecture. In: Proceedings of the International Conference on Compilers, Architectures and Synthesis for Embedded Systems. pp. 137–146. CASES '08, ACM, New York, NY, USA (2008)
14. Molnos, A., Nejad, A.B., Nguyen, B.T., Cotofana, S., Goossens, K.: Decoupled inter- and intra-application scheduling for composable and robust embedded MP-SoC platforms. In: Proceedings of the 15th International Workshop on Software and Compilers for Embedded Systems. pp. 13–21. SCOPES '12, ACM, New York, NY, USA (2012)
15. Moreira, O.: Temporal Analysis and Scheduling of Hard Real-Time Radios running on a Multi-Processor. Ph.D. thesis, Technical University of Eindhoven (2012)
16. Schoeberl, M., Brooks, C., Lee, E.: Code generation for embedded Java with Ptolemy. In: Min, S., Pettit, R., Puschner, P., Ungerer, T. (eds.) Software Technologies for Embedded and Ubiquitous Systems, Lecture Notes in Computer Science, vol. 6399, pp. 155–166. Springer Berlin Heidelberg (2010)
17. Smith, S., Brady, J.: SUSAN–a new approach to low level image processing. International Journal of Computer Vision 23(1), 45–78 (1997)
18. Stefan, R., Molnos, A., Goossens, K.: dAElite: A TDM NoC supporting QoS, multicast, and fast connection set-up. IEEE Transactions on Computers 99 (2012)
19. Stuijk, S., Geilen, M., Basten, T.: SDF3: SDF For Free. In: Proceedings of the Sixth International Conference on Application of Concurrency to System Design (ACSD). pp. 276–278 (June 2006)